“十二五”普通高等教育本科国家级规划教材

新工科建设·计算机类系列教材

Principle of
Microcomputer

微机原理与接口技术

（第5版）

◆彭　虎　韦永梅　周佩玲　付忠谦　主编

中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

“十二五”普通高等教育本科国家级规划教材

新工科建设·计算机类系列教材

微机原理与接口技术

（第5版）

◆ 彭　虎　韦永梅　周佩玲　付忠谦　主编

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书为“十二五”普通高等教育本科国家级规划教材。

本书介绍信息在计算机中的存储形式、进制及相互转换、二进制数的算术和逻辑运算等基础知识；软件部分讲述 8086 指令系统、部分伪指令和 DOS 功能调用及汇编语言程序设计和调试的全过程；硬件部分介绍 8086 CPU 的内部特点、寄存器及相关概念、存储器的分类及层次结构、物理地址形成、译码电路等；讨论诸多 I/O 接口芯片的结构、编程及应用，在串行通信中还介绍了 USB 总线；讨论并举例说明了 A/D、D/A 芯片、微机接口及应用，本书还对 80286、80386 CPU 主要内容及其体系做了简要介绍。全书共 12 章，每章都附有习题，提供配套电子课件。

本书适合作为高等院校信息类理工科学生相关课程的教材，也可以作为相关技术人员或爱好者的参考书。

图书在版编目（CIP）数据

微机原理与接口技术 / 彭虎等主编. —5 版. —北京：电子工业出版社，2021.7
ISBN 978-7-121-41607-1

Ⅰ. ① 微… Ⅱ. ① 彭… Ⅲ. ① 微型计算机－理论－高等学校－教材 ② 微型计算机－接口－高等学校－教材 Ⅳ. ① TP36

中国版本图书馆 CIP 数据核字（2021）第 141210 号

责任编辑：章海涛
印　　刷：三河市鑫金马印装有限公司
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编　100036
开　　本：787×1092　1/16　　印张：21.25　　字数：544 千字
版　　次：2007 年 12 月第 1 版
　　2021 年 7 月第 5 版
印　　次：2021 年 7 月第 1 次印刷
定　　价：59.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：192910558（QQ 群）。

第 5 版前言

时光荏苒，岁月如梭，《微机原理与接口技术》自 2007 年第一版印刷到现在近 15 年了。曾经参与编写本书的作者有的退休，有的即将退休。作为教授计算机多年的老师，无论是在讲台上还是在科研工作上都积累了不少经验，把这些授课的经验在教材中体现出来，帮助学生们尽快掌握计算机基础知识，是一个深爱教育岗位老师的责任。本书自出版后一直受到广大读者的欢迎，多次印刷、再版，成为不少高校计算机教学的指定用书，借此次修订机会，我们再次把多年来积累的教学体会反映在书中。考虑到本书已对计算机接口部分给予了较丰富的论述，本次修订主要集中在基础部分、指令部分和 8086 汇编语言设计部分，具体如下：

(1) 第 1 章

在计算机基本知识中增加了数字电路中译码器和计数器的基础知识。结合译码器和计数器的原理和功能特点，我们设计了一个计算机的原理机。读者可以通过这个原理机的学习，初步了解微机结构，对计算机知识中的一些重要概念有所认识，如指令、地址译码、指令译码、程序计数器和一条指令的执行过程。这对后续课程的学习是大有裨益的。

(2) 第 2 章

在 8086 体系中，寄存器部分是后续学习指令系统和编程的一个重要环节。为了让读者能够深入理解寄存器的功能，我们对 8086 的通用寄存器进行了比较详细的介绍，特别举例说明了不同寄存器的特殊使用。读者通过学习，可以初步了解 8086 体系中寄存器在编程中的突出作用。

另外，我们对标志寄存器的状态标志也给出了比较详细的介绍。初学者对标志常常难以理解，我们尝试对标志的意义进行说明，并对比较两个数的大小涉及标志位的变化进行了深入的探讨和说明。

8086 体系的存储器分段是 8086 有别于其他 CPU 的重要部分，是 8086 体系的一个特色。我们在讲解 8086 内存分段时，通过对物理地址的分解，逐渐过渡到逻辑地址的概念。我们的教学效果已经表明，这种叙述可以使读者比较容易地掌握 8086 物理地址和 8086 逻辑地址的概念以及它们之间的关系。

(3) 第 3 章

在指令系统中，数据传送指令是指令系统中的核心部分。为了让读者尽快精准掌握 8086 的数据传送指令，我们用符号对寄存器、不同寻址方式的存储器进行了归纳。基于这些符号，精准给出了数据传送指令的几乎所有形式。利用这些形式，初学者可以正确地写出 8080 的指令，而不必担心在编程中所用的指令是否有错。这种学习指令的方法不仅可以使读者很快掌握 8086 的指令系统，还十分有助于学习其他类型 CPU 的指令系统。

(4) 第 4 章

在 8086 汇编程序设计的讲解中，为了让读者能够很好地了解和使用伪指令，我们把 C 语言的一些语句引入 8086 的指示性语句介绍。由于 C 语言是本科生低年级就学习的一种计算机语言，因此他们利用 C 语言的知识理解 8086 体系的伪指令语句就比较容易了。

另外，段定义语句是读者比较难以理解的部分，我们通过对汇编指令生成机器码的查表过程举例，逐渐引入段定义语句和其他相关语句，从而让学生容易掌握段定义语句并能编写基本的 8086 汇编语言源程序。

在本书的修订过程中，韦永梅整理了教学团队的教学笔记，完成了前 4 章的修订工作，彭虎负责全书的修订和统稿，研究生谢忠文绘制了部分插图和表格。

作　者

合肥工业大学仪器科学与光电工程学院
阜阳师范大学物理与电子工程学院
中国科学技术大学信息技术学院

第 4 版和第 3 版前言

2010 年，教育部启动“卓越工程师教育培养计划”，旨在培养造就一大批创新能力强、适应经济社会发展需要的高质量各类型工程技术人才。“卓越计划”具有三个特点：一是行业企业深度参与培养过程；二是学校按通用标准和行业标准培养工程人才；三是强化培养学生的工程能力和创新能力。显然，强化培养学生的工程能力和创新能力的一个途径就是改革教材，把书本上的知识，特别是工程技术方面的教材内容与实际结合起来，把提出问题、分析问题、解决问题的方法带入书本的讲解，使学生在学习中领会其工程思想，把握其工程特点，自觉地把所学知识高水平地应用到工作中。

我们在总结 20 多年的微机原理教学经验基础上，于 2006 年出版了相关教材，对计算机的软硬件的充满工程色彩的描述和讲解是本书的主要特色。本书主要在中国科学技术大学非电子系及专业使用，原书写于 2006 年，第 3 版于 2011 年出版，累计印刷了 10 次，超过 4 万册，显示了读者和有关教学部门对此书的重视和厚爱。

根据广大读者和从事微机原理教学同人的意见，并参考笔者历年来的讲稿，对本书进行了修订，修改内容包括：① 删去了一些不必要的内容；② 更正了原书中的一些错误；③ 对书中一些概念做了进一步说明；④ 增加了一些实例。具体来讲，在软件部分对一些重要指令在概念上、在编程使用上做了进一步的描述。为了提高读者的编程能力，本书增加了一些新的编程例子，并对程序尽量详细地给予了说明。

针对本书的讲课安排，笔者建议总学时在 60～80 学时，主要授课范围是第 1～10 章，各单位可以根据自己的教学安排对其内容进行有重点的讲授。第 11 章和 12 章是微机原理知识的扩展，对非信息类的学生不一定需要讲授。

我们从事微机原理教学已近 30 年，虽积累了许多教学经验，但不敢说本书能尽如人意。微机原理与接口内容很庞杂，书中的不妥之处在所难免，今后我们会继续努力，完善这部教材。

本书由彭虎、周佩玲、傅忠谦进行修订，彭虎负责统稿。

由于作者学识浅显，经验有限，书中难免会出现不足和遗漏，希望各位同行批评指正。

本书为教学老师提供相关教学课件，可从网站 http://www.hxedu.com.cn 下载。

读者反馈：192910558（QQ 群）。

作　者

中国科学技术大学信息技术学院
合肥工业大学医学工程学院

第 2 版前言

计算机的诞生是人类科学史上的一件大事。半个多世纪以来，计算机得到了迅猛的发展，从小型机到巨型机，从台式计算机到便携式计算机，从每秒 5000 次的定点算术运算到每秒几十万亿次的浮点算术运算，科学家实现了计算机一代接一代的跨越性的飞跃。现代计算机发展基本分为两个方向，一个是巨型机，另一个是微型机（或称为微机）。前者一个主要特征就是高速性，后者则在保证一定运算速度的前提下使机器微型化。微机的一个代表机型是个人计算机（PC）。现在的微机得到了极为广泛的应用，已经成为人们生活中的必不可少的一部分，人们用它上网来了解世界各地的资讯，用它进行财务处理来管理公司和家庭开支，用它控制机电部件来实现过程操作的自动化……对科研工作者而言，微机更是需要掌握的“第二语言”。

20 多年来，我们分别给中国科学技术大学的电子类专业的学生如电子科学与技术系，非电类专业的学生如化学系、管理系、地球物理系、少年班等其他专业讲授“微机原理与接口”课程，根据学生们的专业不同，讲课的内容和范围不同，因人施教，受到了学生的普遍欢迎。不少非电类的学生把自己所学的知识用于他们的毕业设计，取得了很好的结果。有些学生在学习后，对微机原理与接口技术产生了浓厚的兴趣，在毕业考研究生时选择了电类专业。由于我们长期不断地对微机教学的改革和完善，本课程成为我校最受学生欢迎的课程之一。

2006 年，微机原理与接口课程被评为安徽省精品课程。为了总结我们多年来微机原理教学的经验，应电子工业出版社的提议，我们对非电类的微机教学教案进行了系统的整理，于 2007 年出版了教材《微机原理与接口技术（基于 16 位机）》。该教材已在全国不少高校推广使用，重印多次，销售 4 万多册。作为第 2 版，本书增加了 80286 和 80386 的内容，主要包括如下几个方面：32 位机系统结构、保护模式下的内存管理、基本的存储器接口、高速缓存的数据更新、RISC 简介等。

本书是我们多年微机教学的系统总结，并入选普通高等教育“十一五”国家级规划教材，具有如下特点。

（1）对硬件的独特叙述方式是本书的主要特色，也是现有类似教材中所没有的特点。本书深入浅出地讲解了微机存储器接口、I/O 接口和中断原理等，难点部分讲解清晰，容易理解。本书在介绍存储器接口的章节中分析了 8086 CPU 对总线的读/写特点以及相关接口电路信号线的物理意义，重点讨论了接口电路的一个关键点——片选信号的产生；在此基础上，给出了关于 8086 存储器接口的详细设计实例。通过这些实例的学习，读者可以熟练掌握存储器接口和 I/O 接口的设计方法。

CPU 中断原理的讲解是本课程讲授的一大难点。本书在讲解这部分内容时，先举例让学生理解中断概念，并结合无条件数据传输和条件数据传输的特点，巧妙地从条件数据传输过渡

到中断传输。有了这些概念后再深入讲解 CPU 中断系统，学生就比较容易接受了。

可编程定时器计数器 8253 也是课程讲解的一个难点。8253 有 6 种工作方式，有些方式之间差别不太大，学生们学习起来很是头疼。本书从一个家庭厨房的自动定时系统讲起，通过对其进行完善，逐一介绍了方式 0、方式 4 和方式 5，这样非常有利于保持学生思维的延续性，并使 8253 工作方式有一个更加清晰的比较，从而理解、记忆得更加深刻。

(2) 本书的后两章是有关 80286 和 80386 的介绍。从我们的教学经验来看，32 位机体系中的保护虚地址方式的概念和寻址过程是一个教学关键点，也是一个难点。为了分散难点，我们首先简单介绍 80286 体系，对 80286 体系中保护虚地址方式的概念进行了比较详细的讨论，然后介绍 80386 体系，并进一步讨论 32 位机体系中更为复杂的保护虚地址方式的概念和寻址过程。我们认为，这样安排可以减轻学生的学习负担。

(3) 在传统的理论教学内容基础上，介绍最新的微机及其接口的新知识和新技术，如 USB 接口、串行数模转换接口和模数转换接口等，RISC 的特性和 RISC 的实施要点等。

(4) 全书章节安排合理，重点突出、结构清晰、紧凑，文字简练，没有多余的语句和段落。教学案例丰富多彩，并配有学习指导书、PPT 课件等。

(5) 配有辅导书《微机原理与接口技术学习指导（第 2 版）》。该书主要包括教学指导、习题参考答案和课程设计三部分内容，对课程的教学具有较好的支撑作用。

在教学指导部分中，“教学要求”指出通过该章的教学，应该让学生了解什么、熟悉什么、掌握什么；“教学关键点”中除对主教材中相关内容做出点评和解释外，还列举了更多的实例，用主教材的方法进行分析；“教学难点”是作者根据积累多年的教学心得和体会提出的。

课程设计部分是第 2 版新增加的内容，主要包括微机课程设计及设计分析和实现等内容。同时，该书增加了中国科学技术大学硕士研究生“微机原理与接口”入学考试试卷两套及其解答。读者通过这些内容的学习和训练，可以进一步强化对微机课程知识点的理解和记忆。

(6) 本书配套的电子课件、部分程序代码以及第 1 版中的“附录 A　8086 指令系统”等教学资源将上传到网站中供读者下载，读者可以登录到华信教育资源网站（http://www.hxedu.com.cn），注册后进行下载。如有问题，请发邮件至 unicode@phei.com.cn 咨询。

周佩玲老师编写了第 1、3、4 章，彭虎老师编写了第 5 章的 5.4 节和第 7～12 章，傅忠谦老师编写了第 2 章、第 5 章的其余部分、第 6 章和第 12 章的 12.7 节。全书由彭虎、周佩玲老师统稿。本书在编写过程中得到了中国科学技术大学电子科学技术系领导的大力支持和指导，林克明教授对本书提出了宝贵意见，在此表示衷心的感谢。

感谢林姿、于杰等所做的文字录入、校对和绘图工作，感谢研究生卢昊、李粤得、韩志会、郑驰超、程琴琴和余盛康对本书提供的资料和建议，感谢微机应用教研室的全体老师对本书的支持和帮助。特别感谢本书的编辑，是他们付出了艰辛和努力，终于使本书能与读者见面。

书中疏漏之处，望读者指正，深致谢忱！

作　者

于中国科学技术大学

第 1 版前言

可以说没有哪一门课程能像计算机科学这样高速发展，由笨重、高功耗、结构复杂、功能简单、运算速度慢、只有专家才会使用的电子管计算机，发展到集成数百万个晶体管、功能强大、价格便宜、普及千家万户的微型计算机，仅仅用了几十年的时间。特别是以微型计算机为主的互联网，将世界拉得如此之近，使得人们不用出门便知天下事。

尽管计算机发展迅速，但是其基本原理并没有改变。高档微型计算机在速度和技术上有很大突破，然而在计算机体系结构上还是在遵循冯·诺依曼思想。根据多年教学经验，作者认为：本书作为初学者掌握微型计算机的基本原理的教材是合适的，以 8086 微处理机为核心，介绍微型计算机的软件、硬件原理和接口技术，并通过大量例题和习题介绍其应用。

本书是在周佩玲老师曾经出版的《16 位微型计算机原理、接口及其应用》一书的基础上，经过重新整理、修订和扩充而编写的。

本书提供教学电子课件，请需要者登录 http://www.hxedu.com.cn（华信教育资源网），注册后免费下载。

本书还有配套的辅导书《微机原理与接口技术学习指导（基于 16 位机）》，主要包括教学指导和习题参考答案两部分内容。

周佩玲老师编写了第 1 章、第 3 章和第 4 章，彭虎老师编写了第 7～10 章和第 5 章的 5.4 节，傅忠谦老师编写了第 2 章、第 6 章和第 5 章的其他内容。全书由周佩玲老师统稿。

在本书的编写过程中，我们得到了中国科学技术大学电子科学技术系领导们的大力支持和指导，林克明教授对本书提出了宝贵意见，在此表示衷心的感谢。

同时，感谢林姿、于杰等人所做的文字录入、校对和绘图工作，感谢微机应用教研室的全体老师的支持和帮助。特别感谢本书的编辑们，他们付出了努力和艰辛，终于使本书与读者见面。

书中难免有一些错误，请读者批评指正，不胜感谢！

作　者

于中国科学技术大学

目　录

第 1 章　计算机基本知识

本章导读

- 微型计算机组成
- 微型计算机中信息的表示
- 微型计算机中信息的运算基础
- 几种进制之间的相互转换

电子计算机是 20 世纪科学技术最卓越的成就之一，它的飞速发展是任何其他学科都不能与之相提并论的。计算机技术的发展所带来的信息技术的飞速发展，给人类社会带来了进步，使人们的生产、生活发生了深刻的变化。

计算机在现代科学技术的发展中起着越来越重要的作用。多媒体技术、计算机网络技术、智能信息处理技术、自适应控制技术、数据挖掘与处理技术、机械设计 CAD、金融电子等都离不开计算机。

1.1　微型计算机组成

1946 年，世界上第一台电子计算机由美国宾夕法尼亚大学研制成功。尽管它重达 30 吨，占地 170 m^2，耗电 140 kW，用了 18 800 多个电子管，每秒钟仅能做 5000 次加法，但美国陆军用它计算弹道比人工计算效率提高 8400 倍。当时是用改变线路连接的方法来编排程序的，因此每解一道题都要依靠人工改接线路，准备时间大大超过实际计算时间，所以还称不上是自动计算机。

同时，第一台电子计算机研制时的顾问冯·诺依曼（Von Neumann）教授和他的同事们提出了以二进制和程序存储控制为核心的通用电子数字计算机体系结构原理，确立了计算机的五个基本部件：输入设备、输出设备、运算器、存储器和控制器，从而奠定了当代电子数字计算机体系结构的基础。现在的微型计算机就是采用这种结构，如图 1-1 所示。

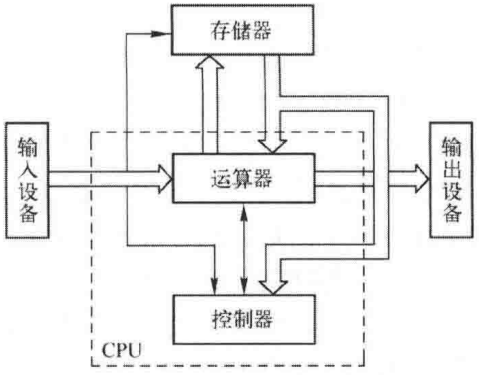


图 1-1　微型计算机的组成

CPU（Central Processing Unit，中央处理单元或

微处理器）：包括运算器和控制器，主要功能是让存储器中的程序被逐条地执行所指定的操作。CPU 是微型计算机的核心部件。从 20 世纪 70 年代初诞生第一片 CPU 以来，基本上每两三年就有更新产品出现。

存储器：其主要功能是存放程序和数据。CPU 在工作时，先从存储器取程序，再执行程序，从而完成对数据的处理。

外部设备：用户与机器之间的桥梁，包括输入设备和输出设备。输入设备的功能是，把用户要求计算机处理的数据、文字、图形和程序等各种形式的信息转换为计算机能接受的编码形式，存入到计算机中。输出设备的功能是，把计算机的处理结果以用户需要的形式（如屏幕显示、文字打印、图形图表、语言音响等）输出出来。

如果要求图 1-1 所示系统自动完成解题任务，必须事先将问题分解成计算机能够处理的各步骤，用某种语言将这些步骤描述出来，然后让计算机按规定的步骤控制计算机工作。这就是计算机设计语言，分为低级语言和高级语言。

低级语言有机器语言和汇编语言，前者就是 0、1 码语言，是计算机唯一能够理解且直接执行的语言。用户编写程序时，命令（指令）、数据和其他信息均以二进制编码书写，难读、难懂、难记、难查错、无法交流，给程序设计和计算机的推广、应用、开发等带来许多困难。对机器语言进行改进的第一步是用一些助记符号代替用 0 和 1 描述的某种机器的指令系统，如八进制数、十六进制数或英语单词的缩写等，称为机器语言的助记符号形式（或符号语言）。汇编语言就是在此基础上完善起来的，改善了机器语言的可读性、可记性，汇编语言指令与机器语言指令一一对应。汇编语言是能够利用计算机所有硬件特征且能直接控制硬件的一种程序设计语言，是计算机能够提供给用户的最快且最有效的编程语言，要求程序设计者必须掌握计算机的硬件知识，这对那些仅对问题感兴趣的用户无疑是一个极大的障碍。

面向问题的程序设计语言称为高级语言。用户面向的是自己领域内的问题，如数值计算、工业控制、专家系统、数据管理和数据库等。这些语言属于过程化语言，要求程序员为每个应用任务写出完成该任务的一系列明确的过程，如适用于数值计算的 FORTRAN 语言、适用于商用和行政管理的 COBOL 语言、适用于专家系统使用的 PROLOG 语言等，以及 BASIC、Pascal、C 语言等。用高级语言编写的程序称为源程序，必须通过编译或解释、连接等步骤才能被计算机处理。

1.2 微型计算机中信息的表示和运算基础

本节讨论计算机是如何存储信息的，计算机内部数据是怎样表示的。进行汇编语言程序计，掌握这些基本知识非常必要。目前使用的计算机是一种电设备，只认识电的信号，如F的高与低、电路的通与断、晶体管的导通与截止、电子开关的开与关等。将这两种状态用 1 两个符号表示，0 或 1 就是二进制数的一位，称为比特（bit）。因此在计算机中，任何信必须用 0 和 1 的数字组合形式来表示。也就是说，计算机存储和处理的仅仅是二进制信

1 个二进制位称为 1 bit；8 个二进制位称为 1 Byte，也称为 1 字节（8 位）；2 个字 1 个字（Word，16 位）；2 个字称为双字（Dword 或 Double Word，32 位）；4 个连续f 四字（Qword 或 Quad Word，64 位）；连续的 10 字节称为十字节，它是一个 80 位二进

人类最初使用十进制计数系统是因为人们有 10 个手指头。计算机却只认识 0 和 1 两个符号，这导致了人与计算机之间的通信问题。发明 ASCII（American Standard Code for Information Interchange，美国标准信息交换码）就是为了代表打字机键盘中的所有符号。几种数值代码的发明也是为了用计算机能够懂得的形式表示数字值。2 的补码这种数字编码使计算机能够表示整个数字的正和负。汉字编码是为解决用西文键盘输入中文的问题而研究的一种编码。对于汇编语言程序设计，最重要的数字编码是十六进制编码，十六进制编码大大简化了二进制编码的表示，缩短了书写的长度。

1.2.1 二进制数的表示和运算

1．二进制数的表示

二进制数仅有两个计数符号：0，1。一个 8 位的二进制数由 8 个 0 或 1 组成，如 11010010，计数符号在不同位置有不同的位权。例如：

$$11010010=1\times2^7+1\times2^6+0\times2^5+1\times2^4+0\times2^3+0\times2^2+1\times2^1+0\times2^0$$

我们习惯于在二进制数的后面加上字母 B（Binary），如 11001101B、10011B。

2．二进制数的运算

（1）算术运算

加法规则：“逢 2 进 1”。

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ +\ 0 & +\ 1 & +\ 0 & +\ 1 \\ \hline 0 & 1 & 1 & 10 \end{array}$$

减法规则：“借 1 当 2”。

$$\begin{array}{cccc} 0 & 1 & 1 & 10 \\ -\ 0 & -\ 0 & -\ 1 & -\ 1 \\ \hline 0 & 1 & 0 & 1 \end{array}$$

乘法规则：1 与 1 乘为 1，其他为 0。

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ \times 0 & \times 1 & \times 0 & \times 1 \\ \hline 0 & 0 & 0 & 1 \end{array}$$

（2）逻辑运算

逻辑非（NOT）运算：

$$0\rightarrow1 \qquad 1\rightarrow0$$

逻辑与（AND）运算：

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ \wedge\ 0 & \wedge\ 1 & \wedge\ 0 & \wedge\ 1 \\ \hline 0 & 0 & 0 & 1 \end{array}$$

逻辑或（OR）运算：

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ \vee\ 0 & \vee\ 1 & \vee\ 0 & \vee\ 1 \\ \hline 0 & 1 & 1 & 1 \end{array}$$

逻辑异或（XOR）运算，又称为“模 2 和”运算：

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ \forall\ 0 & \forall\ 1 & \forall\ 0 & \forall\ 1 \\ \hline 0 & 1 & 1 & 0 \end{array}$$

【例 1-1】 00011010B + 01101101B = 10000111B。

$$\begin{array}{r} 00011010 \\ +\ 01101101 \\ \hline 10000111 \end{array}$$

【例 1-2】 10011011B–00110101B = 01100110B。

$$\begin{array}{r} 10011011 \\ -\ 00110101 \\ \hline 01100110 \end{array}$$

【例 1-3】 01110101B×00110110B = 0001100010101110B。

$$\begin{array}{r} 01110101 \\ \times 00110110 \\ \hline 01110101 \\ 01110101 \\ 01110101 \\ +\quad 01110101 \\ \hline 0001100010101110 \end{array}$$

【例 1-4】 10111001B÷1011B = 00010000B，余 00001001B。

$$\begin{array}{r} 00010000 \\ 1011\ \overline{)\ 10111001} \\ -1011 \\ \hline 00001001 \end{array}$$

【例 1-5】 10011101B∧01101110B = 00001100B。

$$\begin{array}{r} 10011101 \\ \wedge\ 01101110 \\ \hline 00001100 \end{array}$$

【例 1-6】 10011101B∨01101110B = 11111111B。

$$\begin{array}{r} 10011101 \\ \vee\ 01101110 \\ \hline 11111111 \end{array}$$

【例 1-7】 10011101B∀01101110B = 11110011B。

$$\begin{array}{r} 10011101 \\ \forall\ 01101110 \\ \hline 11110011 \end{array}$$

1.2.2 二－十进制（BCD）数的表示和运算

1．二－十进制数的表示

十进制数有 10 个计数符号：0～9，而计算机仅认识 2 个符号：0、1，因此十进制数的 10

个计数符号需要改用 0 和 1 两个符号的编码表示。10 个符号必须用 4 位二进制编码表示：

0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

4 位二进制编码的其他组合不用。这种用二进制编码表示的十进制数称为 BCD（Binary Coded Decimal，二－十进制）数。

2．二－十进制数的加、减运算

BCD 数的运算规则遵循十进制数的运算规则“逢 10 进 1”。但计算机在进行这种运算时会出现潜在的错误。

【例 1-8】

BCD 数	十进制数
1000	8
+ 0101	+ 5
1101	13

【例 1-9】

BCD 数	十进制数
1001	9
+ 0111	+ 7
1 0000	16

例 1-8 的两个 BCD 数相加后，其结果已不是 BCD 数；而例 1-9 的运算结果不对。究其原因是：在计算机中，用 BCD 可以表示十进制数，但其运算规则还是按二进制数进行的。因此，4 位二进制数相加要到 16 才会进位，而不是逢十进位。

为了解决 BCD 数的运算问题，采取调整运算结果的措施。调整规则为：当 BCD 数加法运算结果的 4 位二进制超过 1001（9）或个位向十位有进位时，则加 0110（6）进行调整；当十位向百位有进位时，加 01100000（60）调整。这是人为地干预进位。

在汇编语言程序设计时，只要用一条指令就可实现，我们称为十进制加法和减法调整指令。其减法调整规则类似。

【例 1-10】 10001000（BCD）+ 01101001（BCD）= 101010111（BCD）。

```
    10001000
  + 01101001
  ----------
    11110001
  + 01100110        调整
  ----------
进位 → 1 01010111
```

【例 1-11】 10001000（BCD）－ 01101001（BCD）= 00011001（BCD）。

```
    10001000
  - 01101001
  ----------
    00011111
  -     0110  ← 调整
  ----------
    00011001
```

1.2.3 十六进制数的表示和运算

1．十六进制数的表示

十六进制数有16个计数符号：0～9和A～F。4个二进制位共16种组合状态，这样每个十六进制数的计数符号可对应4位二进制数的一种组合状态；反之，一个十六进制符号可以替代一种4位二进制数的组合状态。在阅读和编写汇编语言程序时，经常用十六进制数表示数据、存储单元地址或代码等。表1-1列出了十进制数、二进制数、二－十进制数、十六进制数之间的关系。

表1-1 十进制数、二进制数、二－十进制数、十六进制数之间的关系

十进制数（D）	二进制数（B）	二－十进制数（BCD）	十六进制数（H）
0	0000	0000	0
1	0001	0001	1
2	0010	0010	2
3	0011	0011	3
4	0100	0100	4
5	0101	0101	5
6	0110	0110	6
7	0111	0111	7
8	1000	1000	8
9	1001	1001	9
10	1010	×	A
11	1011	×	B
12	1100	×	C
13	1101	×	D
14	1110	×	E
15	1111	×	F

在书写数据时，为了区分不同进制的数据，在十进制数后加字母D或省略，在二进制数后加字母B，在十六进制数后加字母H，对于字母开头的十六进制数，还应在数据前加0，以表明它是十六进制数而不是其他，如25H、43、57D、0A5H、0CD3BH。

说明：采用十六进制主要是缩短二进制数的表示长度，方便程序书写和阅读，在计算机内的操作仍然是二进制数的形式。

2．十六进制数的加、减运算

加法运算：“逢16进1”。例如：

```
  1     2     3     A     E     7     9     F     FF
+ 9   + A   + 8   + 5   + C   + 9   + 9   + 1   + 1
---   ---   ---   ---   ---   ---   ---   ---   ---
  A     C     B     F    1A    10    12    10   100
```

减法运算：“借1当16”。例如：

```
  12     2B     D4     58     AB     17     10
- 5    -12    -6B    - E    -37    - 9    - 2
---    ---    ---    ---    ---    ---    ---
  D     19     69     4A     74     E      E
```

1.2.4 带符号二进制数的表示和运算

“+”和“–”在计算机中只能用0和1表示，0表示“+”，1表示“–”，并且符号放在最高有效位。一个8位的二进制数表示一个带符号数，最高有效位 D_7 为符号位。例如，+1表示为00000001B，+127表示为01111111B，–1表示为10000001B，–127表示为11111111B。

用此方法表示的带符号数，在进行运算时遇到了问题：+1加–1的运算是最明显的例子。

```
  (+1)        00000001
+ (–1)      + 10000001
-------     ----------
    0       10000010        (–2)
```

而+127加–127会得出下面的错误结果：

```
  (+127)      01111111
+ (–127)    + 11111111
---------   ----------
     0      1 01111110
              └──────┘
                符号参与运算
```

计算机的数据位长度是固定的，如4位、8位、16位、32位、64位等，最高有效位作为符号后，其数值的大小就减小一半。而且计算机在进行运算操作时，符号也作为数据参与运算。

1．带符号数的2的补码表示法

带符号数采用上面的编码方法不可取。考虑到计算机的数据位长度一定，因而可用补数的编码形式表示带符号数。在进行运算时，丢弃进位，就可得到正确结果。

对一个正的二进制数的每位求反再加1，可得在机器中表示的该数的负数，即2的补码表示法。在这种编码方式中，正数的补码是该正数。下面以8位二进制数为例，求负数的补码。

【例1-12】

```
(+1)          00000001
逐位求反      11111110
加 1      +           1
-----     ------------
–1            11111111
```

【例1-13】

```
(+45H)        01000101
逐位求反      10111010
加 1      +           1
-----     ------------
(–45H)        10111011
```

用2的补码对带符号数进行编码，是目前计算机中常用的方法。表1-2中列出了部分带符号数2的补码值。

表1-2 带符号数2的补码值（8位）

十进制数	十六进制数（H）	2的补码值（B）	十进制数	十六进制数（H）	2的补码值（B）
+127	7F	01111111	0	0	00000000
+100	64	01100100	–1	FF	11111111
+3	3	00000011	–2	FE	11111110
+2	2	00000010	–100	9C	10011100
+1	1	00000001	–128	80	10000000

2．带符号数的运算

带符号数在计算机中用其补码表示后，其运算方法与二进制数的运算相同，也会出现其特有的问题。下面通过例题来分析问题并且讨论如何解决这些问题。

【例 1-14】 用上面的编码方法完成(+1)+(–1)的运算。

```
    (+1)          00000001
  +(-1)       +  11111111
  -------     ------------
     0         1 00000000
               └────────── 进位丢弃
```

【例 1-15】 完成(+65H)+(–23H)的运算。

```
    (+65H)          01100101
  + (-23H)        + 11011101
  ---------       -----------
    +42H          1 01000010
                  └────────── 进位丢弃
```

从以上例题可知，减法可用加上该数的补码，丢弃进位运算完成。

【例 1-16】 完成(+5EH)+(+37H)运算。

```
   (+5EH)        01011110
  +(+37H)       +00110111
  --------      ---------
   +95H          10010101
                 └────────── 符号位变为"1"
```

【例 1-17】 完成(–62H)+(–3BH)运算。

```
    (-62H)          10011110
  + (-3BH)        + 11000101
  ---------       -----------
    -9DH          1 01100011
                  │ └──────── 符号位变为"0"
                  └────────── 进位丢弃
```

在例 1-16 和例 1-17 中，两正数相加后，其结果的符号变为“1”，即两正数相加得出结果为负数；两负数相加后，其结果的符号变为“0”。这显然是错误的。因为带符号数的符号占用了 1 个二进制数的最高有效位，当运算结果数值超出二进制数所能表达的范围时，其值就进入符号位，因而可能改变其运算结果的符号，得到错误结果。以上两例是 8 位二进制数运算，从表 1-2 可知：用 8 位二进制数表示带符号数，数值范围在+127（补码为 7FH）到–128（补码为 80H）之间。

例 1-16 的运算结果为 95H，运算结果进入了符号位；例 1-17 中运算结果为–9DH，运算结果也进入了符号位。因此这两例运算均得不到正确结果。在这种情况下，运算后的值不可取。因为运算结果超出了数值范围，故称为“溢出”。

【例 1-18】 完成(–1FH)+(–4AH)运算。

```
       (-1FH)         11100001
   +   (-4AH)       + 10110110
   -----------      -----------
        -69H         110010111
                     │└──────── 符号位不变
                     └───────── 进位丢弃
```

【例 1-19】 完成(+6EH)+(–7CH)运算。

```
   (+6EH)          01101110
 + (-7CH)        + 10000100
 ---------       ----------
   -0EH          11110010
                 |
                 +---------- 符号位不变
```

在例 1-18 的二进制数运算中，虽然运算结果数值的最高位（D_6）进入符号位（D_7），但因符号位也参与运算，且在运算中有进位，这样运算结果得到修正，直观地看，就是运算结果没有超出其表达范围。例 1-19 中的运算是正常的，运算中其值没有进入符号位，即符号位没有受到影响。

在带符号数的运算中，判别其运算结果是否溢出，从以上例题可以归结如下：若运算结果的数值向前进位（称为 CP），其符号运算向前进位（称为 CF），则溢出的判别为 CP∀CF。这种逻辑“异或”操作结果为 1，则运算结果有溢出。

3．带符号数的符号扩展表示

在计算机系统中，CPU 的数据宽度有 8 位、16 位、32 位甚至 64 位之分，而存储器的数据总是以 8 位（字节）宽度组织，所以在进行某些运算时需要将 8 位数据扩展成 16 位，或者将 16 位数据扩展成 32 位等。无符号数扩展仅在其数据之前加上若干位 0 即可；带符号数的扩展实际是扩展其符号，即正数前补 8 个（或 16 个）0，负数前补 8 个（或 16 个）1。例如，+5 的 16 位表示为 0000000000000101，–5 的 16 位表示为 1111111111111011，+3EH 的 16 位表示为 0000000000111110，–3EH 的 16 位表示为 1111111111000010。

1.2.5 字符的编码表示

计算机通过键盘输入、在显示器上显示或打印的信息大多是西文字母、汉字或其他符号。

1．ASCII 表示

任何信息在计算机内部都被转换成二进制编码。ASCII 是将数字、字母、通用符号、控制符号等按国际上常用的一种标准二进制编码方式对其进行编码。表 1-3 给出了 ASCII 字符集。

表 1-3 ASCII 字符集

$D_3 D_2 D_1 D_0$	$D_6 D_5 D_4$							
	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	}

续表

$D_3\ D_2\ D_1\ D_0$	$D_6\ D_5\ D_4$							
	000	001	010	011	100	101	110	111
1110	SO	RS	。	>	N	^	n	~
1111	SI	US	/	?	O	-	o	DEL

ASCII 规定 8 个二进制位的最高位为 0，其余 7 位可以有 128 种组合，表示 128 个字符，包括 52 个英文大小写字母、数字 0～9、通用运算符、标点符号和控制符。例如，LF 表示换行，CR 表示回车，BS 表示退格，ESC 是换码，DEL 是删除。

2．汉字编码表示

计算机汉字处理技术对在我国推广计算机应用、加强国际交流都具有重要意义。汉字也是一种字符，但它是一种象形文字，故汉字的计算机处理技术远比拼音文字复杂，且汉字数目多，常用的汉字约 3000 个，次常用的字约 4000 个。基于目前计算机的键盘，汉字想直接从键盘输入是不可能的，像西文一样用 7 位二进制数对几千个汉字进行编码，也不能满足要求。

为了能在不同的汉字系统之间互相通信、共享汉字信息，我国制定并推行一种汉字编码，即 GB2312—1980 国家标准信息交换用汉字编码字符集（基本集），简称国标码。在国标码中，每个图形字符都规定了二进制表示的编码，一个汉字用 2 字节编码，每字节用 7 位二进制数，高位置 0。国标码在计算机中容易与 ASCII 混淆，在中西文兼用时无法使用。若将国标码每字节的高位置 1，作为标识符，则可与 ASCII 区分。这种汉字编码又被称为内部码。

汉字内部码结构短，一个汉字只占 2 字节，足以表达数千个汉字和各种符号、图形。另外，汉字内部码便于与西文字符兼容，在同一计算机系统中，可从字节的最高位是 1 还是 0 来区分汉字、西文。当然，计算机的汉字内部码要经过汉字字模库检索后，找到该汉字的字形信息才能输出。

至于其他物理信息，都要通过相应的传感器将物理信息转换成电信号，经过 A/D 转换（模拟到数字的转换），通过接口电路进入计算机中。

1.3 进制及其转换

十进制数有 10 个计数符号 0～9，基数为 10；二进制数有 2 个计数符号 0 和 1，基数为 2；十六进制数有 16 个计数符号 0～F，基数为 16。同理，三进制数有 3 个计数符号 0、1、2，基数为 3；八进制数有 8 个计数符号 0～7，基数为 8。

1.3.1 十进制整数到任意进制整数的转换

十进制整数转换成任意进制整数，可按进位制的基数，通过“辗转相除法”进行。

1．十进制整数转换成二进制整数

【例 1-20】 将 205 转换成二进制整数。

```
2 | 205
2 | 102      ……1（最低有效位）
```

```
2 | 51      ·····0
2 | 25      ·····1
2 | 12      ·····1
2 | 6       ·····0
2 | 3       ·····0
2 | 1       ·····1
  0         ·····1（最高有效位）
```

故：205 = 11001101B。

2．十进制整数转换成十六进制整数

【例 1-21】 将 327 转换成十六进制数。

```
16 | 327
16 | 20    ····· 7
16 | 1     ····· 4
   0       ····· 1
```

故：327 = 147H。

3．十进制整数转换成八进制整数、三进制整数

【例 1-22】 将 628 转换成八进制数（用字母 Q 表示）。

```
8 | 628
8 | 78    ····· 4
8 | 9     ····· 6
8 | 1     ····· 1
  0       ····· 1
```

故：628 = 1164Q。

【例 1-23】 将 125 转换成三进制数（用字母 T 表示）。

```
3 | 125
3 | 41    ·····2
3 | 13    ·····2
3 | 4     ·····1
3 | 1     ·····1
  0       ·····1
```

故：125 = 11122T。

从以上例子可知：将十进制整数转换成任意进制整数，只要“除基取余”就能实现。

1.3.2 任意进制整数到十进制整数的转换

任意进制整数到十进制整数的转换，按基数位权展开可以实现。

【例 1-24】 将二进制数 1110110B 转换成十进制数。

$$1110110B = 1\times2^6 + 1\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0$$
$$= 64 + 32 + 16 + 4 + 2 = 118$$

【例 1-25】 将十六进制数 0A2EH 转换成十进制数。

$$0A2EH = 10\times16^2 + 2\times16^1 + 14\times16^0$$
$$= 2560 + 32 + 14 = 2606$$

【例 1-26】 将八进制数 1372Q 转换成十进制数。

$$1372Q = 1\times8^3 + 3\times8^2 + 7\times8^1 + 2\times8^0$$
$$= 512 + 192 + 56 + 2 = 762$$

【例 1-27】 将三进制数 1020T 转换成十进制数。

$$2021T = 2\times3^3 + 0\times3^2 + 2\times3^1 + 1\times3^0$$
$$= 54 + 6 + 1 = 61$$

1.3.3 二进制数与十六进制数的转换

4 位二进制数可表示 1 位十六进制数，因此二进制数与十六进制数之间的转换很简单。

【例 1-28】 将二进制数 101011B 和 110001110B 转换成十六进制数。

$$101011B = 2BH$$
$$110001110B = 18EH$$

从上例可以看出，将二进制整数从右边开始，每 4 位可分为 1 个十六进制数，左边不够 4 位则用 0 补齐。

【例 1-29】 将十六进制数 8BDH 和 0C5AFH 转换成二进制数。

$$8BDH = 1000010111101B$$
$$0C5AFH = 1100010110101111B$$

可以看出，将每位十六进制数用 4 位二进制数位表示即可。

注意：字母开头的十六进制数前面补 0，以区别于字符或名字。

1.3.4 带符号二进制整数与十进制整数的转换

二进制数用基数 2，按位权展开即可转换成十进制数。而带符号数的最高位为符号，而且带符号数在计算机中是用其补码表示的，如果符号位为 0，那么该数为正数，可按位权展开；如果符号位为 1，那么不能按正常位权那样展开。所以，对于带符号二进制负数，除了符号位，对每位求反再加 1 后，再按位权展开，并加上符号，才能将负的二进制数转换成十进制数。

【例 1-30】 将计算机中带符号二进制数 01011011B 转换成十进制整数。

$$+1011011B = +(1\times2^6 + 1\times2^4 + 1\times2^3 + 1\times2^1 + 1\times2^0) = +91$$

【例 1-31】 将计算机中带符号二进制数 10101101B 转换成十进制整数。

将数 10101101B 除符号位外每位求反再加 1：

$$-1010011B = -(1\times2^6 + 1\times2^4 + 1\times2^1 + 1\times2^0) = -83$$

上例也可用另一种方法实现，即将符号位和数值按正常位权展开，符号位取负，然后将两部分求和。例如：

$$10101101\text{B} = -1\times2^7 + 1\times2^5 + 1\times2^3 + 1\times2^2 + 1\times2^0$$
$$= -128 + 45 = -83$$

1.4 逻辑电路及应用

1.2 节和 1.3 节主要介绍了数制及其运算方式，这些是计算机数据处理的基础。本节讨论两种逻辑电路——译码器和计数器，以及基于这两种逻辑电路构造出的一种计算机原理机，目的是让读者在刚开始学习微机原理时就能初步了解计算机内部基本结构和计算机工作原理，这对后面内容的学习将大有裨益。

1.4.1 译码器及其应用

译码器是数字逻辑电路中常用器件之一，其功能就是将输入代码转换成特定的输出信号。

假设译码器有 n 个输入信号和 N 个输出信号，若 $N=2^n$，则称为全译码器，常见的全译码器有 2－4 线译码器、3－8 线译码器、4－16 线译码器等。如果 $N<2^n$，称为部分译码器，如二－十进制译码器（也称为 4－10 线译码器）等。下面以 2－4 线译码器为例说明译码器的工作原理和电路结构。2－4 线译码器的功能如表 1-4 所示。

表 1-4　2－4 线译码器的功能

	EI	A	B		Y_0	Y_1	Y_2	Y_3
输入	1	×	×	输出	1	1	1	1
	0	0	0		0	1	1	1
	0	0	1		1	0	1	1
	0	1	0		1	1	0	1
	0	1	1		1	1	1	0

由表 1-4 可写出各输出函数表达式：

$$Y_0 = \overline{\overline{EI}AB} \qquad Y_1 = \overline{\overline{EI}\overline{AB}} \qquad Y_2 = \overline{\overline{EI}A\overline{B}} \qquad Y_3 = \overline{\overline{EI}AB}$$

用门电路实现 2－4 线译码器的逻辑电路如图 1-2 所示。

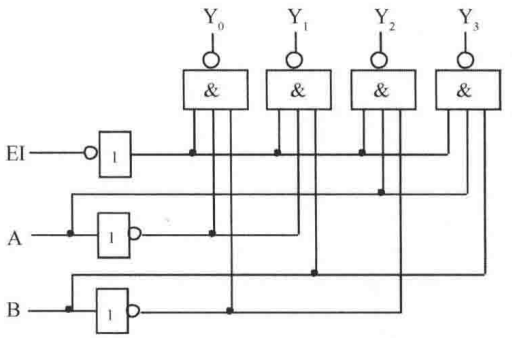


图 1-2　2－4 线译码器逻辑

根据表 1-4，2－4 线译码器的输入与输出之间的关系如下。

① 当控制线 EI 为高电平时，译码输出 $Y_0 \sim Y_3$ 的信号电平均为高电平，与输入 A、B 无关，这时译码器不工作，输出为无效电平。

② 当控制线 EI 为低电平时，输出 $Y_0 \sim Y_3$ 的有效信号电平和输入 A、B 一一对应，输入 AB 可以指定输出线 Y_{AB} 有效，注意输出为低电平有效。A、B 为输出的二进制下标，分别对应 Y_0、Y_1、Y_2 和 Y_3。

译码器的基本应用是把输入的二进制状态用输出线及其状态表示，如图 1-3 所示。当 EI 为高电平时，输出 $Y_0 \sim Y_3$ 为高电平，对应的指示灯全灭。在 EI 接 0 电平的情况下，如果 AB 输入为 00，则输出 Y_0 为低电平，其他译码输出线为高电平，这时 0 号灯点亮，表示当前的输入为 00 编码。同理，当 AB 输入为不同的二进制编码时，对应的指示灯就会被点亮。

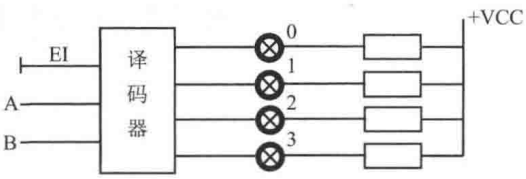


图 1-3　2－4 译码器应用

图 1-3 中的译码输出线连接的是指示灯，也可以连接其他部件，如继电器等其他功能性电路。输入译码线输入不同二进制代码，对应的译码输出线有效，这时其连接的电路就被“选中”或被“激活”。

1.4.2　计数器及其应用

计数器也是数字电路中常用的一种逻辑部件，主要功能是统计输入脉冲 CP 个数，用于实现分频、定时、产生节拍脉冲和脉冲序列及进行数字运算等。

计数器的分类有很多，如按计数进制可分为二进制计数器和非二进制计数器，按数字的增减趋势可分为加法计数器、减法计数器和可逆计数器等。这里简单介绍二进制加计数器。

图 1-4 为由 4 个下降沿触发的 JK 触发器组成的 4 位异步二进制加法计数器的逻辑。JK 触发器都接成 T' 触发器（即 J=K=1）。最低位触发器 FF0 的时钟脉冲输入端接计数脉冲 CP，其他触发器的时钟脉冲输入端接相邻低位触发器的 Q 端。

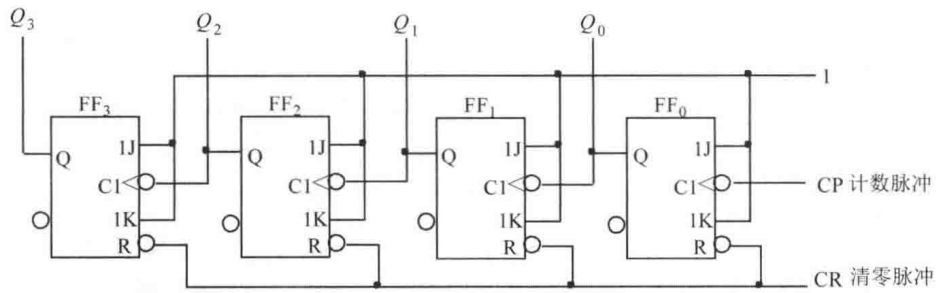


图 1-4　由 JK 触发器组成的 4 位异步二进制加法计数器的逻辑

由于该电路的连线简单且规律性强，当低位的触发器由高变成低时，高一位触发器就翻转。由此规律，其时序波形图如图 1-5 所示，状态图如图 1-6 所示。由状态图可知，从初态 0000（由清零脉冲所置）开始，每输入一个计数脉冲，计数器状态按二进制加法规律加 1，所以是二进制加法计数器（4 位）。又因为该计数器有 0000～1111 共 16 个状态，所以也被称为十六进制（1 位）加法计数器或模 16（M=16）加法计数器。

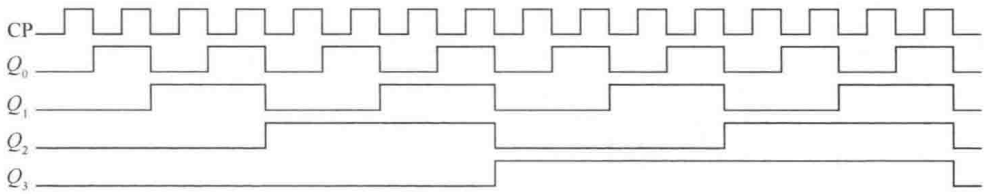


图 1-5　时序图

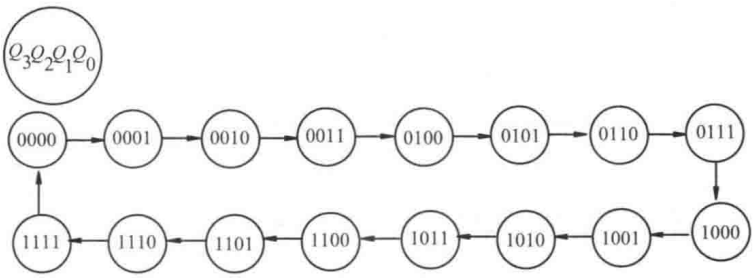


图 1-6　状态转换图

计数器应用非常广泛，常用的是计数功能，如药厂传输带上的药片通过光电检测电路转换为脉冲，这个脉冲作为 CP 脉冲输入计数器，就可以记录统计传输带上药片的个数。另一种常用的应用是产生有序的数据，如二进制加法计数器可以有序产生二进制数 0…000、0…001、0…010、0…011，即十进制数 0、1、2、3…，计数器的 CP 脉冲一般为高稳定度的晶体振荡电路产生。如果系统设定每变动一个数字，与数字对应的电路动作一次，那么通过计数器计数就可以产生一系列有序的动作，这些有序的动作集合就完成了一个任务。另外，当 CP 脉冲由高稳定度的晶体振荡电路产生时，计数器也可以作为定时器使用，这就是电子钟的工作原理。

1.4.3　原理机

计算机能够高速自动运行的原因从硬件上来讲是因为有高速的电子器件，另一个原因是任务被细化，形成一系列子任务，而这些子任务是计算机能认识和完成的。把这些子任务有序串接起来，逐一完成，综合效果就是完成了一个任务。

要使计算机自动处理一个任务，我们必须做到如下两点。第一，把任务分解成一系列子任务，然后有序排列，如子任务 1、子任务 2 等，并把这些子任务用二进制数来表述。第二，要有“有序”启动下一个子任务的器件，即自动跟踪子任务有序完成的器件。

针对第一点，我们可以用编写程序（或简称“编程”）来实现，第二点要用计数器来完成。

图 1-7 是一个计算机的原理机，预先把编程产生的并用二进制代码表示的任务 1、任务 2 等序列依次放在存储器的第 0 号单元、第 1 号单元等。

系统工作如下：计数器在上电时清零，然后在一个个 CP 脉冲到来时计数，计数器输出二进制计数值 0…000、0…001、0…010 等；计数器输出连接到译码器 1 的地址端，译码器 1 把其输入的二进制状态译码出来，当计数器输出为 0 时，译码器 0 号线有效，则存放在存储器的子任务 1 的单元被选中，其代码被输入译码器 2，而当计数器持续给出 1、2、3 等时，在译码器的作用下，子任务 2、子任务 3 被一一取出。

子任务 1、子任务 2 等就是 CPU 可以理解并执行的操作，因此被称为指令，并用二进制数表示，每个指令代表一种功能。

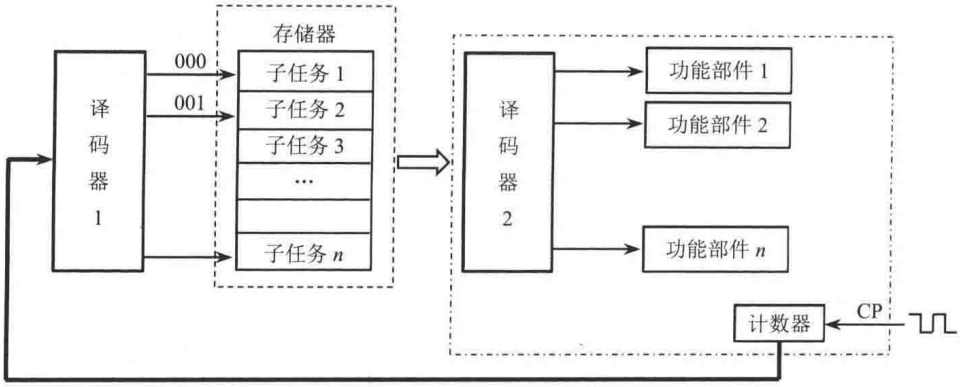


图 1-7　一种计算机的原理机

取子任务 1、子任务 2 等过程就是计算机工作中的取指令过程，简称取指过程。

连接计数器的译码器 1 可以根据编号确定存储器单元，具有地址译码的功能，因此被称为地址译码器。

所取指令被依次取出送到另一个译码器 2 中，译码器 2 对输入的二进制代码进行译码，其输出连接一些功能性硬件部件，这些部件功能与指令所代表的功能一一对应，如算术逻辑运算中的加减乘除和与或非运算。这样不同指令通过译码器 2，其译码输出线连接的功能性电路就会被激活而完成指令所规定的操作，这就是执行指令过程。当存储器中所有用二进制编码表示的子任务 1、子任务 2 等完成后，任务也就完成了。

上述过程就是程序的运行过程，简单直观地给出了一个计算机的基本工作原理。在图 1-7 所示中，译码器 2 起到把二进制指令进行解释使计算机进行相应操作的功能，因此被称为指令译码器。计数器在程序运行时保证了程序顺序执行的要求，其内容就是指令的地址，因此被称为程序计数器。

在微机体系中，计数器、指令译码器等常常集成在一个芯片中，这个芯片就是 CPU，它是微机系统中的核心部件。

从上面的原理机可以看到，一条指令的执行时序就是取指令和执行指令。

习 题 1

1．将下列二进制数转换成十进制数、BCD 数。

（1）01000100B　　（2）00110111B

（3）00101101B　　（4）01001111B

2．完成下列二进制无符号数的加法运算。

（1）00011101+00000101　　（2）10010110+01101111

（3）00111110+11100011　　（4）10101010+11001101

3．完成下列二进制数的逻辑“与”“或”“异或”运算。

（1）10110011 和 11100001　　（2）10101010 和 00110011

（3）01110001 和 11111111　　（4）00111110 和 00001111

4．完成下列十六进制无符号数的加、减运算。

（1）25A5 和 0043　　（2）62FC 和 0005　　（3）7889 和 0787

（4）7BCD 和 35B3　　（5）4CBE 和 0BAF

5．将下列十进制数转换成二进制数、十六进制数。

（1）19　　（2）35　　（3）86

（4）255　　（5）4094　　（6）62473

6．将下列二进制补码转换成十进制数。

（1）11000100　　（2）01111011

（3）01111100　　（4）10000000

7．将下列 BCD 数分别转换成十进制数、二进制数、十六进制数。

（1）01100101　　（2）10010010

（3）01111000　　（4）01000111

8．试指出 CR（回车符）、LF（换行符）的 ASCII 值。

9．试指出十六进制数计数符号 0～9 及 A～F 的 ASCII 值，说明它们 ASCII 值之间的数值关系。

10．大写字母 A～Z 与小写字母 a～z 的 ASCII 值有何区别？

第 2 章　8086 系统结构

本章导读

- 8086 CPU 的结构
- 8086 系统的结构
- 8086 系统的配置
- 8086 CPU 的内部时序

8086 CPU 曾是使用广泛的 16 位微处理器。80386、80486 和 Pentium 系列、Core 系列都是从 8086 发展而来的，称为 80x86 系列。8086 是由 Intel 公司设计生产的，具有 40 个引脚的双列直插式封装芯片，内外数据总线都为 16 位，地址总线为 20 位，直接寻址为 1 MB。最早用于 IBM PC 中的 8088 CPU，其内部结构与 8086 基本相同，只是 8088 只有 8 条外部数据总线，因此也称为准 16 位微处理器。

本章主要介绍 8086 微处理机，详细讲述 8086 CPU 的结构，然后围绕以 8086 CPU 组成的计算机系统，介绍有关功能部件及其相互作用、外部引脚和系统配置，为了便于读者进一步了解 8086 执行指令的过程，本章还将介绍 8086 CPU 的内部时序。

2.1　8086 CPU 结构

2.1.1　8086 CPU 的内部结构

8086 CPU 由两部分组成，如图 2-1 所示，即指令执行部件（Execution Unit，EU）和总线接口部件（Bus Interface Unit，BIU）组成，在图中用点画线隔开。

指令执行部件主要由算术逻辑运算单元（ALU）、标志寄存器（FR）、通用寄存器组和 EU 控制电路 4 个部件组成，其主要功能是执行指令。

总线接口部件（BIU）主要由地址加法器、专用寄存器组、指令队列和总线控制电路 4 个部件组成，其主要功能是形成访问存储器的物理地址、访问存储器并取指令暂存到指令队列中等待执行，访问存储器或 I/O 端口读取操作数来参加 EU 运算或存放运算结果等。

传统的 CPU 在执行一个程序时，总是先从存储器中取出下一条指令，读出一个操作数（如指令需要操作数的话），然后执行指令，如图 2-2 所示。

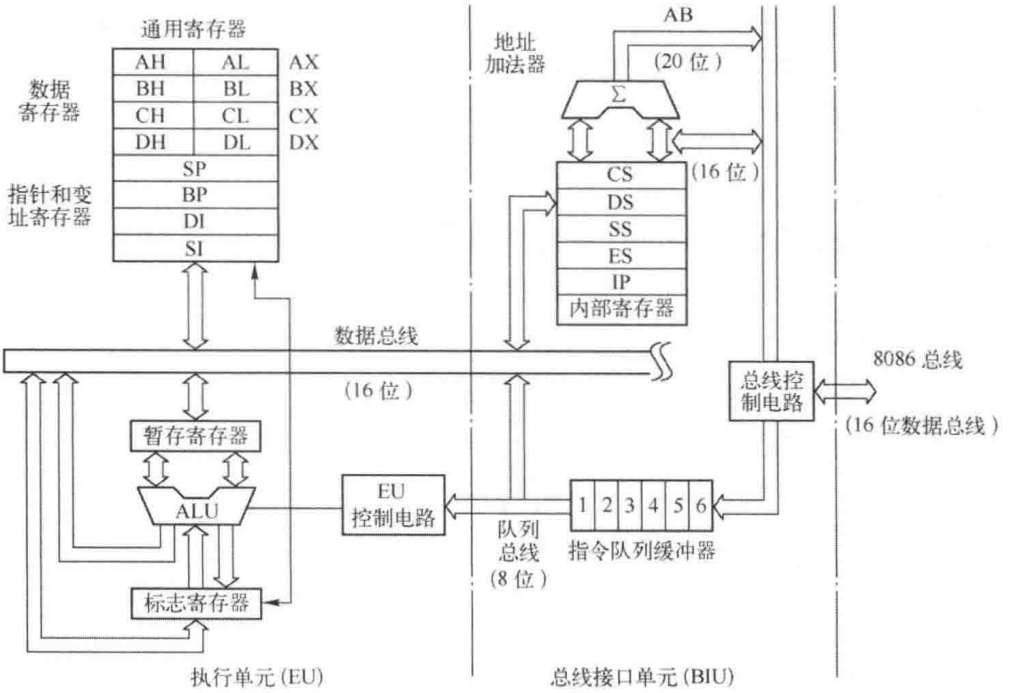


图 2-1　8086 CPU 的内部结构

在 8086 CPU 中，这些步骤分配给两个独立的处理单元进行，指令执行部件（EU）负责执行指令，总线接口部件（BIU）负责取指令、读出操作数和写入结果。这两个单元能够相互独立地工作，并在大多数情况下，大部分取指令和执行指令重叠进行，即在取指令的同时，指令执行部件也同时工作，这就有效地加快了系统的运算速率，如图 2-3 所示。

图 2-2　传统 CPU 的工作方式

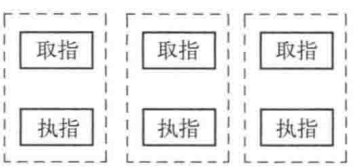


图 2-3　8086 CPU 工作方式

换句话说，执行部件（EU）在执行指令时不必通过访问存储器去取指令，而是从指令队列中取得指令代码，并分析执行它。若在指令执行过程中需要访问存储器或 I/O 端口，则 EU 只需向 BIU 送出访问存储器的逻辑地址。BIU 将根据 EU 要求形成访问存储器的物理地址，然后访问存储器或 I/O 端口，取得操作数后送到 EU 中参加运算，必要时可将运算结果再写回存储器中。所以，EU 实际上不与外界打交道，所有与外部的操作都在 BIU 控制下完成。

1．指令执行部件

EU 只负责执行指令。在一般情况下，指令是顺序执行的，EU 可源源不断地从指令队列中取得待执行的指令，达到满负荷地连续执行指令，从而节省“取指”时间。如果在指令执行过程中需要访问存储器取操作数，那么 EU 将访问地址送给 BIU 后，等待操作数到来后才能继续操作；遇到转移指令，BIU 会将指令队列中的后继指令作废。这时，EU 等待 BIU 重新从存储器中取出目标地址中的指令代码进入指令队列后，才继续执行指令。在这种情况下，EU 和 BIU 的并行操作会受到一定影响，这是采用重叠操作方式不可避免的现象。只要转移指令出现

率不是很高，两者的重叠操作仍然会取得良好效果。顺便指出，EU 处理的所有地址都是 16 位的，但是 BIU 能实现地址浮动，使 EU 可以访问 1 MB 的存储空间。

EU 的 ALU（算术逻辑运算单元）完成 8 位或 16 位的二进制数运算，运算结果通过内部总线送到通用寄存器组或 BIU 的内部寄存器，等待写入存储器。暂存寄存器用来暂存参加运算的操作数，经 ALU 运算后的结果状态如进位、溢出等信息置入 FR（标志寄存器）保存。

EU 控制电路负责从 BIU 的指令队列中取指令，并对指令译码，根据指令要求向 EU 内部各部件发出控制命令，以完成各条指令的功能。

2．总线接口部件

BIU 负责与外部存储器或 I/O 端口打交道。在正常情况下，BIU 通过地址加法器形成某条指令在存储器中的物理地址后，启动存储器，从给定的地址中取出指令代码，送 BIU 中的指令队列中等待执行，一旦指令队列中空出 2 字节，BIU 将自动进入读指令的操作，以填满指令队列。只要收到 EU 送来的操作数地址，BIU 就立即形成操作数的物理地址，完成读/写操作数或运算结果等功能。当遇到转移类指令执行转移时，BIU 将指令队列中尚存的指令“作废”，重新从存储器新的目标地址中取指令，并送指令队列中。

BIU 中的指令队列可以存放 6 字节的指令代码，一般应保证指令队列中总是填满指令，使得 EU 可以不断地得到等待执行的指令。20 位地址加法器专门用来完成由逻辑地址变换成物理地址的功能，实际上是进行一次地址加法，将两个 16 位的逻辑地址变换为 20 位的物理地址，从而使可寻址的存储空间达到 1 MB。

总线控制电路将 8086 CPU 的内部总线与外部总线相连，是 8086 CPU 与外部交换数据的必经之路，实际上包括 16 条数据总线、20 条地址总线和若干控制总线。CPU 正是通过这些总线与外部世界取得联系，从而形成各种规模的 8086 微型计算机。

关于 EU 和 BIU 中的寄存器结构，本书将在后续章节中专门讨论。

2.1.2 8086 CPU 的寄存器结构

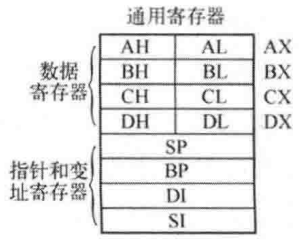


图 2-4　8086 通用寄存器

1．通用寄存器

微处理器的结构中许多寄存器，其作用是让程序暂存数据和地址。8086 CPU 的 EU 中有 8 个 16 位通用寄存器，分成两组，如图 2-4 所示，一组为数据寄存器，另一组为指针和变址寄存器。

数据寄存器由 AX、BX、CX 和 DX 构成，称为通用数据寄存器。这些通用数据寄存器除了具有保存数据作用，还各有特殊用途。通用数据寄存器既可以作为 16 位寄存器，也可以作为 8 位寄存器来使用，即把每个 16 位的通用寄存器分成高 8 位和低 8 位。低 8 位寄存器被命名为 AL、BL、CL 和 DL，高 8 位寄存器被命名为 AH、BH、CH 和 DH。寄存器一般存放 8 位数据，这样 8086 CPU 内部就有了 8 个 8 位寄存器。

① AX（Accumulator Register，累加器）：用来存放参加运算的数据和结果，在乘、除法运算、I/O 操作、BCD 数运算中有不可替代的作用。

【例 2-1】 AX 特殊用法举例。指令

```
MUL   BL
```

是一个 8 位的乘法指令，其功能为寄存器 AL×BL。其中，一个乘数一定放在 AL 中，另一个乘数可以放在 BL 中，也可以放在 BH、CL 等 8 位寄存器中，乘积一定放在 AX 中。

由于 AL 在某些 8 位指令中有不可替代的作用，具有 8 位 CPU 中的累加器功能，所以 AL 也被称为 8086CPU 中的 8 位累加器。

【例 2-2】 AX 特殊用法举例。指令

```
MUL   BX
```

是一个 16 位的乘法指令，其功能为寄存器 AX×BX。其中，一个乘数一定放在 AX 中，另一个乘数可以放在 BX 中，也可以放在 BX、CX 等 16 位通用寄存器中，乘积一定放在 DXAX 中，DX 中保存高 16 位。

② BX（Base Register，基址寄存器）：除了可作为数据寄存器，还可存放内存的逻辑偏移地址，而 AX、CX、DX 不能。

【例 2-3】 BX 特殊用法举例。指令

```
MOV   AL, [BX]
```

中，BX 所存内容为内存地址。如图 2-5 所示，如果 BX 的内容为 1000H，那么上述指令以 BX 的内容作为地址，从地址 1000H 的内存单元取数据给 AL。

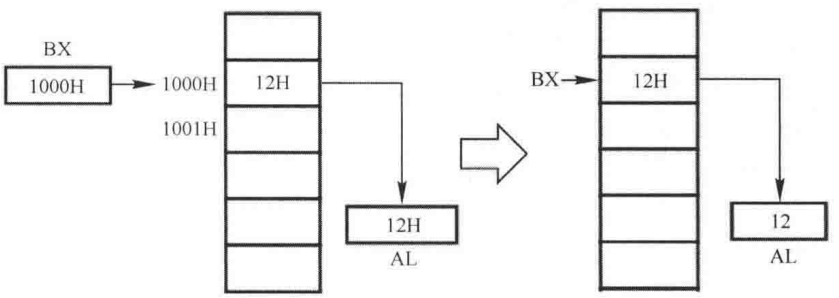


图 2-5　指令 MOV　AL, [BX]功能

说明： BX 作为地址时，这个地址是存储器单元的偏移地址。

BX 的内容可以作为地址指针的功能，是其他数据寄存器 AX、CX、DX 所不具备的。

③ CX（Counter Register，数据寄存器）：既可作为数据寄存器，又可在串指令和移位指令中作为计数用。

先考虑 C 语言编程中常常用到了循环控制语句：

```
for (i = 100; i > 0; i--)
{
   循环体程序段;
}
```

其中，计数器变量 i 初值为 100，每执行一次循环体，计数器变量 i 就会自动减 1，当循环体重复执行 100 次后，i 的内容递减为 0，循环过程结束。这种操作在 8086 体系中被称为串操作。

【例 2-4】 CX 特殊用法举例。指令

```
LOOP
```

其功能与上面的循环语句功能一致，其计数器 i 在 LOOP 语句中指定寄存器 CX，循环体结构如下：

```
again:
```

```
        循环体程序段;
        LOOP   again
```

其中，again 为标号，实际上是指令的符号位置，用来指定某一个指令的位置，这里指定的就是循环体程序段的第一条指令地址。

上面的程序段表示，执行完循环体程序段后就执行 LOOP 指令。LOOP 指令先对计数器 CX 做减 1 操作，并将结果送回 CX，再判断 CX 中的内容，若 CX 减 1 不为 0，则转到标号 again 处重复执行循环体程序段；若 CX 减 1 后为 0，则运行 LOOP 指令下面的程序段，这时循环过程结束。

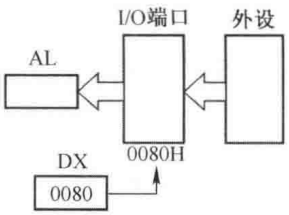


图 2-6　指令 IN　AL, DX 功能

④ DX（Data Register，数据寄存器）：除可作为通用数据寄存器外，还在乘、除法运算、带符号数的扩展指令中有特殊用途。

【例 2-5】 DX 特殊用法举例。指令

```
        IN   AL, DX
```

DX 所存内容可以作为 I/O 的端口地址。如图 2-6 所示，若 DX 的内容为 0080H，则上述指令就是从编号为 0080H 的 I/O 端口输入一个 8 位数据给 AL。这里，DX 的内容为 I/O 端口地址的功能，是其他寄存器如 AX、BX、CX 所不具备的。

数据寄存器 AX、BX、CX 和 DX 中，只有 BX 可以作为地址指针。考虑到实际编程中需要大量的地址指针，特别是涉及多个数组处理编程，只有 BX 指示或寻址数组中的元素是远远不够的，为此 8086 CPU 准备了另一组 16 位寄存器，称为指针和变址寄存器，由 SI、DI、BP 和 SP 组成，其特殊功能是存放存储器地址。

⑤ SI（Source Index，源变址寄存器）：用于存放内存的逻辑偏移地址（隐含的逻辑段地址在数据段寄存器中），也可存放数据。

⑥ DI（Destination Index，目标变址寄存器）：用于存放内存的逻辑偏移地址（隐含的逻辑段地址在数据段寄存器中），也可存放数据。

⑦ BP（Base Pointer，基址指针）：用于存放内存的逻辑偏移地址（隐含的逻辑段地址在堆栈段寄存器中）。

⑧ SP（Stack Pointer，堆栈指针）：用于存放栈顶的逻辑偏移地址（隐含的逻辑段地址在堆栈段寄存器中）。

SI、DI 和 BP 与 BX 的功能差不多，但 SP 有明显差异，主要用于指示堆栈栈顶。这 4 个寄存器详细功能将在第 3 章讲述。

为了更好地管理存储器，8086 CPU 把对应的存储空间分成几个逻辑段，存放在上述指针或变址寄存器中的内容往往是在某逻辑段中寻址的偏移地址。例如，一条 ADD 指令可以在当前数据段中指定它的一个操作数，办法之一是把该操作数的偏移量放在一个指示器或变址寄存器中。当然，这些寄存器也可以存放 16 位数据。

对于一些指令，通用寄存器具有一致性。例如，ADD 指令可将任意两个 8 位或 16 位通用寄存器的内容相加，结果可存放到这两个寄存器中的任何一个中。为了缩短指令代码长度，8086 CPU 的少数指令把某些通用寄存器作为专用。例如，串操作指令总是用 CX 寄存器存放串的长度，并在串操作指令执行过程中，CX 寄存器专用于计数。这样，所有串操作指令不必再在指令中指定 CX 寄存器，因而缩短了串操作指令的代码长度。如果在指令中没有明显指出

但指令中需要使用这些寄存器，通常称为“隐含寻址”。

隐含寻址实际上是在某类指令中指定某些通用寄存器作为特殊用法。例如，8 位乘法指令 MUL　BL，指令中没有标出的寄存器为 AL 和 AX，但 AL 被指定为一个乘数，而 AX 作为乘积；16 位的乘法指令 MUL　BX，指令中没有标出的寄存器为 AX 和 DX，但指定 AX 作为一个乘数，DXAX 作为乘积。程序设计者在编制程序时需遵循这些规定，将某些特殊数据放在特定寄存器中，才能正确地执行这些指令。这样也许会给程序设计者带来一些麻烦，但因其采用“隐含”方式，能有效地缩短指令代码的长度，并有可能提高指令的运行速度。

在 8086 CPU 中，有些寄存器具有上述隐含性质，即相应的指令中不必给出寄存器名；另有一些寄存器虽也具有特殊用途，但不能隐含寻址，指令中使用这些寄存器时必须给出它们的寄存器名，如表 2-1 所示。

表 2-1　寄存器的特殊用途和隐含性质

寄存器名	特 殊 用 途	隐含性质
AX，AL	在输入/输出指令中作为数据寄存器	不能隐含
	在乘法指令中存放被乘数或乘积，在除法指令中存放被除数或商	隐含
AH	在 LAHF 指令中，作为目标寄存器	隐含
AL	在十进制数运算指令中作为累加器	隐含
	在 XLAT 指令中作为累加器	隐含
BX	在间接寻址中作为基址寄存器	不能隐含
	在 XLAT 指令中作为基址寄存器	隐含
CX	在串操作指令和 LOOP 指令中作为计数器	隐含
CL	在移位/循环移位指令中作为移位次数计数器	不能隐含
DX	在字乘法/除法指令中存放乘积高位或被除数高位或余数	隐含
	在间接寻址的输入/输出指令中作为地址寄存器	不能隐含
SI	在字符串运算指令中作为源变址寄存器	隐含
	在间接寻址中作为变址寄存器	不能隐含
DI	在字符串运算指令中作为目标变址寄存器	隐含
	在间接寻址中作为变址寄存器	不能隐含
BP	在间接寻址中作为基址指针	不能隐含
SP	在堆栈操作中作为堆栈指针	隐含

2．段寄存器

8086 CPU 总线接口部件（BIU）有如下 4 个 16 位段寄存器。

- ❖ CS（Code Segment，代码段寄存器）：存放程序代码段起始地址的高 16 位。
- ❖ DS（Data Segment，数据段寄存器）：存放数据段起始地址的高 16 位。
- ❖ SS（Stack Segment，堆栈段寄存器）：存放堆栈段起始地址的高 16 位。
- ❖ ES（Extended Segment，扩展段寄存器）：存放扩展数据段起始地址的高 16 位。

段寄存器中存的也是地址，专门用于定位一个段的内存区域，通过指示这个区域的首个单元地址来定位一个段。在 8086 体系中，一个段所代表的内存区域最大为 64K。

8086 具有 1 MB 的存储空间，但放在指令指示器和变址寄存器中的地址都只有 16 位。16 位地址不能直接提供 1 MB 存储器寻址，只能在一个特定的 64 KB 段的偏移量中寻址。由此，划分地址段并且确定偏移量寻址是在哪一段中进行的就变得至关重要。在 8086 系统中，1 MB 存储空间可被分成许多逻辑段，每段最长为 64 KB，这些逻辑段可在整个 1 MB 存储空间中浮

动。同时，划分的各逻辑段首地址的高 16 位存放在该段寄存器中，高 16 位地址又称为段基址。这样，代码段寄存器 CS 用来存放当前代码段的段基址，数据段寄存器 DS 用来存放当前数据段的段基址，扩展段寄存器 ES 用来存放扩展段的段基址，堆栈段寄存器 SS 用来存放堆栈段的段基址。

系统中只设 4 个段寄存器，任何时候 CPU 只能识别当前可寻址的 4 个逻辑段。通常，代码段用来存放可执行的指令，数据段和扩展段用来存放参加运算的操作数和运算结果，堆栈段作为程序执行中需要使用的堆栈，即在存储器中开辟的堆栈区。如果程序量或数据量很大，超过 64 KB，就需要定义多个代码段、数据段、扩展段和堆栈段，只是在 4 个段寄存器中存放的应该是当前正在使用的逻辑段的段基址，必要时可以修改这些段寄存器的内容，以扩大程序的规模，这样就可以访问 8086 系统的 1 MB 存储器。

3．标志寄存器

8086 CPU 设置了一个 16 位标志寄存器（FR），如图 2-7 所示，其中规定了 9 个标志位，用来存放运算结果特征和控制 CPU 操作。

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

图 2-7　标志寄存器（FR）格式

标志寄存器（FR）中存放的 9 个标志位可以分成两类。一类为状态标志，用来表示运算结果的特征，它们是 CF、PF、AF、ZF、SF 和 OF；另一类为控制标志，用来控制 CPU 的操作，它们是 IF、DF 和 TF。

（1）CF（Carry Flag），进位标志位

CF=1，表示本次运算中最高位（D_{15} 或 D_7）有进位（加法运算时）或有借位（减法运算时）。CF 标志可以通过 STC 指令置位，通过 CLC 指令复位（清除进位标志），还可通过 CMC 指令将当前 CF 标志取反。

（2）PF（Parity Flag，奇偶校验标志位）

PF=1，表示本次运算结果中有偶数个“1”；PF=0，表示本次运算结果中有奇数个“1”。

（3）AF（Auxiliary Carry Flag，辅助进位标志位）

AF=1，表示运算结果的 8 位数据中，低 4 位向高 4 位有进位（加法运算时）或有借位（减法运算时），这个标志位只在十进制数即 BCD 码运算中有用。

（4）ZF（Zero Flag，零标志位）

ZF=1，表示本次运算结果为 0；ZF=0，运算结果非 0。

（5）SF（Sign Flag，符号标志位）

SF=1，表示本次运算结果的最高位（D_{15} 或 D_7）为“1”，否则 SF=0。

（6）OF（Overflow Flag，溢出标志位）

OF=1，表示本次算术运算结果产生溢出，否则 OF=0。所谓溢出，就是字节运算的结果超出了范围–128～+127，或者字运算超出了范围–32768～32767。在加法运算时，当次高位向最高位有进位而最高位没有向前进位时，便产生溢出，于是 OF=1；或者，当次高位向最高位无进位而最高位向前有进位时，同样产生溢出，于是 OF=1。在减法运算时，当判断出最高位需要借位而低位并不向最高位产生借位时，OF=1；或者，当判断出低位从最高位有借位而最高

位并不需要从更高位借位时，OF=1。计算机在进行算术运算时，判断是否溢出的一个简单方法是：运算结果是否合理。例如，当两个正数相加，和为负数时，OF=1，否则 OF=0。**说明：不管有符号数或无符号数，计算机在运算时会影响 OF 标志，但对编程者来说，该标志仅对带符号数有意义。**

① 算术操作和逻辑操作都会影响状态标志。为了对上述 6 个状态标志位有更具体了解，举两个例子。

【例 2-6】 加法运算一。

```
  0010 0011 0100 0101
+ 0011 0010 0001 1001
= 0101 0101 0101 1110
```

结果对各状态标志的影响如下：运算结果的最高位为 0，所以 SF=0；运算结果本身不为 0，所以 ZF=0；结果中所含 1 的个数为 9，即奇数个 1，所以 PF=0；最高位没有产生进位，所以 CF=0；D_3 位没有往 D_4 位产生进位，所以 AF=0；次高位没有往最高位产生进位，最高位也没有往前进位，所以 OF=0。

【例 2-7】 加法运算二。

```
  0101 0100 0011 1001
+ 0100 0101 0110 1010
= 1001 1001 1010 0011
```

结果对各状态标志的影响如下：运算结果的最高位为 1，所以 SF=1；运算结果本身不为 0，所以 ZF=0；结果中所含 1 的个数为 8，即偶数个 1，所以 PF=1；最高位没有产生进位，所以 CF=0；D_3 位向 D_4 位产生了进位，所以 AF=1；次高位往最高位产生了进位，而最高位没有往前产生进位，所以 OF=1。

【例 2-8】 逻辑“或”运算。

```
  0100 0100 0011 1001
OR 0100 0100 0011 1001
= 0100 0100 0011 1001
```

结果对各状态标志的影响如下：运算结果的最高位为 0，所以 SF=0；运算结果本身不为 0，所以 ZF=0；结果中所含 1 的个数为 5，即奇数个 1，所以 PF=0；因为是逻辑运算，不存在进位和溢出情况，所以 CF=0，AF=0，OF=0。

上面的例子说明：逻辑运算可以使标志位 CF 清零；如果想测试操作数中 1 的个数的奇偶性，可以采用两个相同内容的操作数做逻辑运算。

② 标志位 CF 可以直接参与运算。标志位 CF 不仅可以作为运算后的标志，说明运算的结果状态，也可以直接参与运算。

【例 2-9】 32 位数的加法。

为了书写方便，这里用十六进制列出算式。

```
  5B10 090A
+ 3A31 F706
= 9542 0610
```

8086 CPU 指令系统中只有 16 位加法指令，为了完成 32 位加法，必须采用编程的手段，

通过两次 16 位加法运算来实现 32 位加法。具体做法是：先对两个加数的低 16 位相加，得出和的低 16 位；再对两个加数的高 16 位相加，得出和的高 16 位。但是上面的运算中存在低 16 位向高 16 位的进位，这个进位不能被忽视。为此，8086 CPU 指令系统中配置了两种加法指令：一种是 ADD 指令，只能完成一般 16 位加法；另一种是 ADC 指令，把进位标志及两个数同时加在一起。因此在编程时，先采用 ADD 实现低 16 位的相加，再采用 ADC 完成高 16 位数相加，这样就完成了 32 位数的相加。

可见，标志位 CF 起到扩充多位数的算术加或减的作用。

③ 通过标志位比较数的大小。比较两个数的大小是编程中常常遇到的情况，参与比较的数也存在“有符号数”和“无符号数”的区别。

比较两个无符号数的大小相对容易。先对两个操作数 x 和 y 做减法运算，即 $x-y$，再通过 ZF 标志位和 CF 标志位完成比较。过程如下：先判别 ZF 是否有效，若 ZF 有效（即 ZF=1），则 $x=y$，否则 $x\neq y$；若 ZF 无效，则判别 CF 标志位，若 CF 标志位为 1，则 $x<y$，反之 $x>y$。

对于有符号数，由于存在溢出情况，比较两个数的大小就复杂些了。先对操作数 x 与 y 做减法运算，再通过标志位 ZF 标志位判断两数是否相等。

若 ZF 有效（即 ZF=1），则 x 与 y 相等，否则（即 ZF=0）x 与 y 不相等。若 ZF 无效，则判断符号标志位 SF 和溢出标志位 OF。若没有溢出，OF=0，同时若 SF=0，两数相减的结果是正，则 $x>y$；SF=1，则两数相减的结果是负数，$x<y$。若有溢出，则 OF=1。由于溢出就是实际运算中的符号位丢失，因此当 SF=1 时，两数相减的结果为正，则 $x>y$，否则两数相减的结果就是负数，则 $x<y$。

表 2-2　有符号数比较大小

OF	SF	结果
0	0	$x>y$
0	1	$x<y$
1	0	$x<y$
1	1	$x>y$

综上所述，我们可以得出表 2-2。当 OF 和 SF 同类符号时，结果是 $x>y$，而当 OF 和 SF 异号时，结果是 $x<y$。因此，可以用下面的表达式表示有符号数比较大小的结果：若 OF⊕SF=0，则 $x>y$；若 OF⊕SF=1，则 $x<y$。符号“⊕”表示逻辑异或。

下面继续介绍其他 3 个控制标志。

（7）IF（Interrupt Flag，中断标志位）

IF=1，表示允许 CPU 响应可屏蔽中断。IF 标志可以通过 STI 指令置位，也可通过 CLI 指令复位。

（8）DF（Direction Flag，方向标志位）

在串操作指令中，DF=0，表示串操作指令地址指针自动增量，即串操作的地址由低地址向高地址进行；DF=1，表示地址指针自动减量，即串操作的地址是由高地址向低地址进行的。DF 标志位可以通过 STD 指令置位，也可通过 CLD 指令复位。

（9）TF（Trap Flag，单步标志位）

TF=1，表示控制 CPU 进入单步工作方式。在这种工作方式下，CPU 每执行完一条指令就自动产生一次内部中断，这在程序调试过程中非常有用。对 TF 标志的设定和清除没有专用指令，但可用编程间接达到目的。例如，先将标志寄存器的内容推入堆栈，设法对标志寄存器的某一位（即 TF 位）进行置 1 或清零操作，再利用堆栈指令将修改后的数据送入标志寄存器中，这样就可以对该标志进行置位和清零。

4．指令指针寄存器

16 位指令指针寄存器（IP）与传统的 8 位微处理器中的程序计数器（PC）相似。IP 中的

内容可由 BIU 自动修改，使之始终存有相对于当前代码段起点偏移量的下一条指令，即 IP 总是指向下一条待执行的指令。当然，由于指令队列的缘故，这个定义并不十分确切。在正常执行过程中，IP 中存有 BIU 要取出的下一条指令的偏移量。程序是不能直接访问 IP 的，但可通过某些指令修改 IP 的内容，这类指令主要是程序控制类指令。该类指令可将转移目标地址送入 IP 中，以实现程序的转移，也可以将 IP 内容压入堆栈或从堆栈中弹出。

2.1.3 8086 CPU 的引脚及功能

8086 CPU 是 16 位 CPU，采用高性能的 N 沟道、耗尽型负载的硅栅工艺（HMOS）制造。由于受当时制造工艺的限制，部分引脚采用分时复用的方式，构成了 40 条引脚的双列直插式封装，如图 2-8 所示。

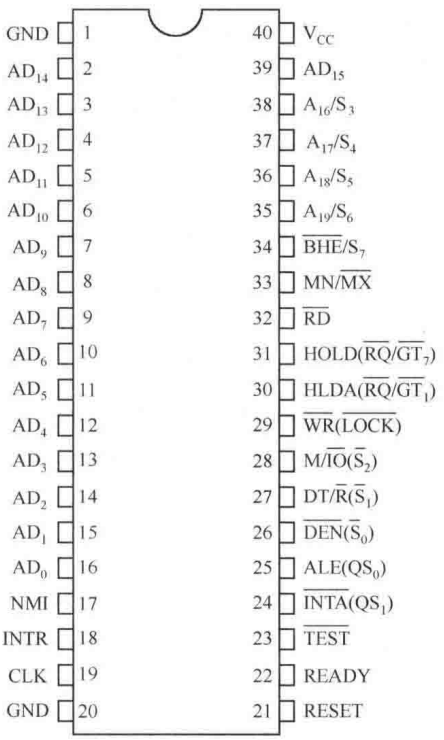


图 2-8 8086 CPU 引脚

8086 CPU 可以工作在最小模式和最大模式下，因此有 8 条引脚（24～31）在上述两种工作模式中具有不同的功能，图 2-8 括号中所示为最大模式下被重新定义的控制信号。

1．引脚功能

AD_{15} ~ AD_0（Address Data Bus）：分时复用的地址数据总线。传送地址时以三态输出，传送数据时可以以双向三态输入/输出。

A_{19}/S_6，A_{18}/S_5，A_{17}/S_4和A_{16}/S_3（Address/ Status）：分时复用的地址/状态线。作为地址线用时，A_{19} ~ A_{16}与 AD_{15} ~ AD_0 一起构成访问存储器的 20 位物理地址。当 CPU 访问 I/O 端口时，A_{19} ~ A_{16} 保持为“0”（低电平）。作为状态线用时，S_6 ~ S_3 用来输出状态信息，其中 S_3 和 S_4 表示当前使用的段寄存器名，如表 2-3 所示，S_3S_4 =10 时表示当前正在使用 CS 寄存器对存储器寻址，或者是当前正在对 I/O 端口或中断矢量寻址。S_5 用来表示中断标志状态，当 IF=1 时，S_5 =1。S_6 恒保持为 0。

$\overline{BHE}/S_7$（Bus High Enable/Status）：总线高位有效信号（三态输出，低电平有效），表示当前高 8 位数据总线上的数据有效。当读/写存储器或 I/O 端口以及中断响应时，$\overline{BHE}$ 与地址线 AD_0 配合表示当前总线使用情况，如表 2-4 所示。非数据传输期间，S_7 输出状态信息，低电平有效，在 CPU 处于保持响应期间被设置为高阻抗状态。

表 2-3 S_4、S_3 状态编码

S_4S_3	段寄存器
00	ES
01	SS
10	CS（I/O，INT）
11	DS

表 2-4 $\overline{BHE}$ 和 AD_0 编码的含义

$\overline{BHE}$	AD_0	总线使用情况
0	0	16 位数据总线上进行字节传送
0	1	高 8 位数据总线上进行字节传送
1	0	低 8 位数据总线上进行字节传送
1	1	无效

$\overline{RD}$（Read）：读信号（三态输出，低电平有效），表示当前 CPU 正在读存储器或 I/O 端口。

$\overline{WR}$（Write）：写信号（三态输出，低电平有效），表示当前 CPU 正在写存储器或 I/O 端口。

M / $\overline{IO}$（Memory/IO）：存储器或 I/O 端口访问信号（三态输出）。M / $\overline{IO}$ 为高电平时，表示当前 CPU 正在访问存储器；M / $\overline{IO}$ 为低电平时，表示当前 CPU 正在访问 I/O 端口。

READY：准备就绪信号（由外部输入，高电平有效），表示 CPU 访问的存储器或 I/O 端口已准备好传输数据。当 READY 无效时，要求 CPU 插入一个或多个等待周期 T_w，直到 READY 信号有效为止。

INTR（Interrupt Request）：中断请求信号（由外部输入，电平触发，高电平有效）。INTR 有效时，表示外部设备向 CPU 发出中断请求，CPU 在每条指令的最后一个时钟周期对 INTR 进行测试。一旦测试到有中断请求，并且当中断允许标志 IF=1 时，则暂停执行下条指令，转入中断响应周期。

$\overline{INTA}$（Interrupt Acknowledge）：中断响应信号（向外部输出，低电平有效），表示 CPU 响应了外部发来的 INTR 信号。在中断响应总线周期，它可作为读取中断矢量的选通信号。

NMI（Non-Maskable Interrupt Request）：不可屏蔽中断请求信号（由外部输入，边沿触发，正跳变有效）。NMI 不受中断允许标志的限制，CPU 一旦测试到 NMI 请求信号，待当前指令执行完，就自动从中断入口地址表中找到类型 2 中断服务程序的入口地址，并转去执行。NMI 是一种比 INTR 高级的中断请求。

$\overline{TEST}$：测试信号（由外部输入，低电平有效）。当 CPU 执行 WAIT 指令时（WAIT 指令使处理器与外部硬件同步），每隔 5 个时钟周期对 TEST 进行一次测试，若测试到该信号无效，则 CPU 继续执行 WAIT 指令，即处于空闲等待状态；若 CPU 测到 TEST 输入为低电平，则转而执行 WAIT 的下一条指令。由此可见，TEST 对 WAIT 指令起到监视的作用。

RESET：复位信号（由外部输入，高电平有效），至少要保持 4 个时钟周期。CPU 接收到该信号后，停止进行操作，并对寄存器 FR、IP、DS、SS、ES 及指令队列清零，而将 CS 设置为 FFFFH。当复位信号变为低电平时，CPU 从 FFFF0H 开始执行程序。由此可见，采用 8086 CPU 计算机系统的启动程序就保持在开始的存储器中。

ALE（Address Latch Enable）：地址锁存允许信号（向外部输出，高电平有效），在最小模式系统中作为地址锁存器 8282/8283 的片选信号。

DT / $\overline{R}$（Data Transmit/Receive）：数据发送/接收控制信号（三态输出），在最小模式系统中用来控制数据收发器 8286/8287 的数据传送方向。若 DT / $\overline{R}$ 为高电平，表示数据从 CPU 向外部输出，即完成写操作；若 DT / $\overline{R}$ 为低电平，表示数据从外部向 CPU 输入，即完成读操作。

$\overline{DEN}$（Data Enable）：数据允许信号（三态输出，低电平有效），在最小模式系统中作为数据收发器 8286/8287 的选通信号。

HOLD（Hold Request）：总线请求信号（由外部输入，高电平有效且向 CPU 请求使用总线）。

HLDA（Hold Acknowledge）：在最小模式系统中表示有其他共享总线的处理总线请求响应信号（向外部输出，高电平有效）。CPU 一旦测试到有 HOLD 请求时，就在当前总线周期结束时，使 HLDA 有效，表示响应这一总线请求，并且立即让出总线使用权，CPU 中的 EU 可以继续工作到下一次要求使用总线为止。CPU 只有当 HOLD 无效时，才将 HLDA 置成无效，并且收回对总线的使用权，继续自己的操作。

MN / $\overline{MX}$（Minimum/Maximum）：工作模式选择信号（由外部输入），为高电平时，表示 CPU 工作在最小模式系统中；为低电平时，表示 CPU 工作在最大模式系统中。

CLK（Clock）：主时钟信号（由时钟发生器 8284 输入）。8086 CPU 可以使用的时钟频率

随芯片型号不同而异，8086 为 5 MHz，8086-1 为 10 MHz，8086-2 为 8 MHz。

V_{CC}（电源）：8086 CPU 只需要单一的+5 V 电源，由 V_{CC} 输入。

2．8086 CPU 工作在最大模式系统时，有 8 个引脚（24～31）要重新定义

$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$（Bus Cycle Status）：总线周期状态信号（三态输出），在最大模式系统中由 CPU 传送给总线控制器 8288，8288 对它们译码后代替 CPU 输出相应的控制信号。详细情况将在本章 2.2 节中讨论。

$\overline{LOCK}$：封锁信号（三态输出，低电平有效）。$\overline{LOCK}$ 有效时，表示 CPU 不允许其他总线主控者占用总线。这个信号由软件设置。当在指令前加上 LOCK 前缀时，则在执行这条指令期间 $\overline{LOCK}$ 保持有效，即在此指令执行期间，CPU 封锁其他主控者使用总线。

$\overline{RQ}/\overline{GT_0}$ 和 $\overline{RQ}/\overline{GT_1}$（Request/Grant）：请求/同意信号（双向，低电平有效）。该信号为输入时，表示其他主控者向 CPU 请求使用总线；为输出时，表示 CPU 对总线请求的响应信号。两条线可同时与两个主控者相连，内部保证 $\overline{RQ}/\overline{GT_0}$ 比 $\overline{RQ}/\overline{GT_1}$ 有较高优先级。

QS_1 和 QS_0（Instruction Queue Status）：指令队列状态（向外部输出），表示 CPU 中指令队列当前的状态，如表 2-5 所示。设置这两个引脚的目的是让外部的设备监视 CPU 内部的指令队列，可让协处理器 8087 进行指令的扩展处理。

表 2-5　QS_1 和 QS_0 编码的含义

QS_1	QS_0	编码含义
0	0	无操作
0	1	从队列中取第一字节
1	0	队列已空
1	1	从队列中取后续字节

2.2　8086 CPU 的结构和配置

8086 CPU 是一个微处理器而不是一台微型计算机。微处理器并不包含任何存储单元或输入/输出接口。做一个形象的比喻，微处理器能够思考、判断，但是如果没有存储器就不能记忆，没有输入/输出接口就不能听（输入）或说（输出）。本节将简要介绍 8086 的存储系统和输入/输出结构，结合 8086 基本总线接口器件，给出 8086 最小模式系统和最大模式系统配置。

2.2.1　8086 存储器结构

8086 CPU 的存储器是一个最多可寻址的 1 MB 存储空间，系统为每字节分配一个 20 位的物理地址（对应的十六进制数地址范围为 00000H～FFFFFH）。在存储器中，任何两个相邻的字节被定义为一个字。在一个字中，每字节都有一个地址，并且这两个地址中的较小的一个被用来作为该字的地址。由表 2-6 可以看出，一个字的起始地址可以从偶地址开始，如数 6B07H，也可以从奇地址开始，如数 3E60H。并且，较高存储器地址的字节存放该字的高 8 位，较低存储器地址的字节存放该字的低 8 位。

表 2-6　字在存储器中的例子

内存地址	存储器	说　明	内存地址	存储器	说　明
00000H	07	低字节，字以偶地址开始	00004H	—	—
00001H	6B	高字节，表示的数为	00005H	60	低字节，字以奇地址开始
00002H	—	—	00006H	3E	高字节，表示的数为
00003H	—	—	00007H	—	—

1．存储器的组成

在 8086 系统中，存储器采用分体结构，即 1 MB 的存储空间分成两个 512 KB 的存储体，一个存储体中包含偶数地址，另一个存储体包含奇数地址。两个存储体采用字节交叉编址方式，如表 2-7 所示。

表 2-7　两个存储体采用交叉编址方式

奇地址	D_{15}～D_8	D_7～D_0	偶地址
00001H	—	—	00002H
00003H	—	—	00003H
00005H	—	—	00006H
…	512K×8　奇地址存储体（A_0=1）	512K×8　偶地址存储体（A_0=1）	…
FFFFFH	—	—	FFFFEH

对于任何一个存储体，只要 19 位地址（A_{19}～A_1）就够了。地址 A_0 用来区分当前访问的是哪一个存储体：A_0=0，表示访问偶地址存储体；A_0=1，表示访问奇地址存储体。8086 CPU 允许访问存储器中的 1 字节，也允许访问存储器中的 1 个字（相邻两字节），要求同时访问两个存储体，各取出 1 字节的信息。这时只用 A_0 控制读/写操作就不够了，为此增设了总线高位有效控制信号 $\overline{BHE}$。当 $\overline{BHE}$ 有效时，选定奇地址存储体，存储体内地址由 A_{19}～A_1 确定。当 A_0=0 时，选定偶地址存储体，存储体内地址同样由 A_{19}～A_1 确定。注意，偶地址存储体固定与低 8 位数据总线（D_7～D_0）相连，因此被称为低字节存储体；奇地址存储体固定与高 8 位数据总线（D_{15}～D_8）相连，因此被称为高字节存储体。$\overline{BHE}$ 与 A_0 互相配合（如表 2-8 所示），使 CPU 可访问一个存储体中的 1 字节或同时访问两个存储体中的 1 个字。

两个存储体与总线之间的连接如图 2-9 所示。显然，奇地址存储体的片选端 $\overline{SEL}$ 受控于 8086 CPU 的 $\overline{BHE}$，偶地址存储体的片选端 $\overline{SEL}$ 受控于地址线 A_0。

表 2-8　$\overline{BHE}$ 与 A_0 组合对应的控制

$\overline{BHE}$	A_0	对应的操作
0	0	从偶地址读/写 1 个字
0	1	从奇地址读/写 1 字节
1	0	从偶地址读/写 1 字节
0 1	1 0	从奇地址读/写 1 个字（分两次读/写）

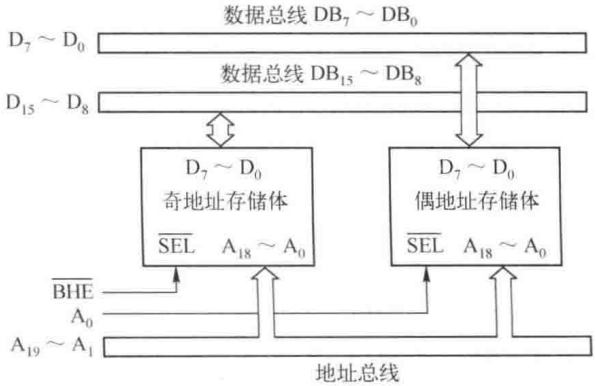


图 2-9　存储体与总线的连接

8086 的有些指令是访问（读或写）字节的，有些指令是访问字的。在同一个时间，8086 从存储器中取出的信息数量总是 16 位的，且该 16 位数据是在存储器中以偶地址开头的 2 字节的内容。当 8086 要访问字节时，在被读出的 16 位数据中，只要忽略高 8 位或低 8 位就可得到所要的 1 字节信息，如图 2-10(a)和(b)所示。当 8086 要访问 1 个字而这个字始于偶地址时，只要使 A_0=0，$\overline{BHE}$=0，就可一次访问到该字的内容，如图 2-10(c)所示；当要访问的字始于奇地址时，情况比较复杂，必须对两个连续的偶地址字做两次存储器访问，每次访问忽略不需要的 1 字节，并保留剩余的 1 字节，然后变换得到完整的一个字的信息，如图 2-10(d)所示。

必须指出，8086 的编程并不涉及这些细节，一条指令只是请求访问一个特定的字节或字，必须要做的操作都是在处理器控制下自动实现的。

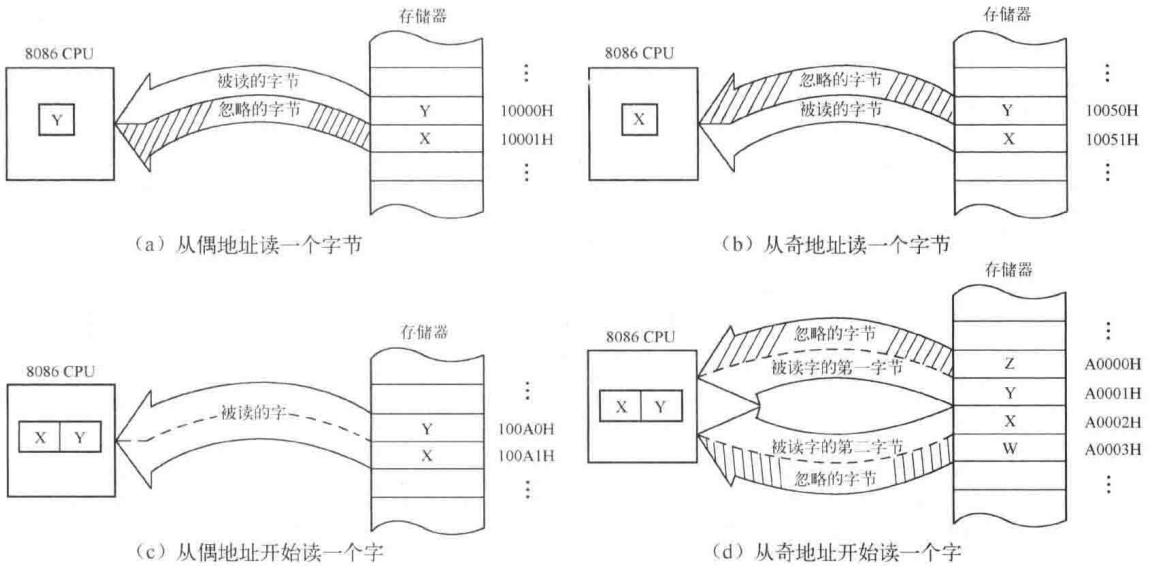


图 2-10　读存储器操作过程

如上所述，在字访问情况下，对奇地址存放的字需要进行两次读/写操作，对偶地址存放的字仅需要一次读/写操作。为了加快程序的运行速度，希望被访问的存储器的字地址为偶地址。通常，这种从偶地址开始的字称为“对准字”，而从奇地址开始的字称为“非对准字”。

2．存储器的分段

8086 的寻址空间是 1 MB，因此要对整个空间寻址需要 20 位长的地址码，但是所有寄存器都是 16 位寄存器。这样，如果仅是一个 16 位寄存器的内容寻址，就只能寻址 64 KB。要达到对 1 MB 空间的寻址，8086 采用分段并附以地址偏移量的办法形成 20 位的物理地址，得到对 1 MB 空间的寻址。被划分的存储器段称为逻辑段。这里先讨论段的特点。

① 在 8086 中，存储空间被设置成若干逻辑段，在一个编程任务中，存储空间一般被分为 4 个段，分别是代码段、附加段、堆栈段和数据段。实际编程应用时，根据需要，也可少于 4 个段。

② 段基地址。8086 中存在 8 位字节型变量、16 位字型变量和 32 位双字型变量，为了定位这些变量，这些变量在内存中的第一字节位置（或首地址）被作为其地址，称为段基址。同样一个 64 KB 的段，它在内存中第一字节的位置（或首地址）就是这个段的地址。

③ 为了管理段，8086 中设置了 4 个 16 位的段寄存器 CS、SS、DS、ES，用来指示段在内存中的位置。由于段基址是 20 位的，为了解决段寄存器是 16 位而段基址是 20 位的不兼容问题，8086 规定，段基址最低 4 位是 0000B，段寄存器只存段基址的第 4～19 位的信息，即存放相应段首地址的高 16 位（即段基址）。这样当段寄存器的内容确定后，在其内容最低位补 4 位二进制的 0000B，就可以生成段基址，相当于段寄存器内容乘以 16 或段寄存器内容左移 4 位。这样就可以确定该段的位置了。

④ 当段寄存器内容确定后，段的寻址范围已经确定，8086 规定其容量不大于 64 KB，同时通过修改段寄存器内容，可以使逻辑段在整个存储空间中浮动。各逻辑段之间可以紧密相

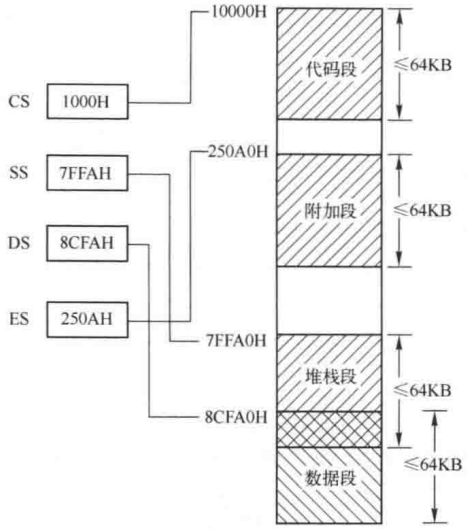


图 2-11　存储器分段的例子

连，可以中间有间隔，也可以相互重叠（部分重叠甚至完全重叠）。如图 2-11 所示，当前有效的代码段、附加段、堆栈段和数据段的段基址分别为 1000H、250AH、7FFAH 和 8CFAH。

在 8086 中，存储空间被设置成若干逻辑段，每个物理地址可被包含在一个逻辑段中，也可以包含在多个相互重叠的逻辑段中。

3．物理地址和逻辑地址

如何在存储器分段的前提下，使用 16 位的地址寄存器方便地在 1 MB 存储器空间中定位或找到目标存储单元呢？为了解决上面问题，我们先给出和存储器地址有关的两个概念。

① 物理地址。物理地址指存储器的绝对地址。在 8086 中，物理地址是 20 位的，直接对应 CPU 的地址线第 0～19 位，因此物理地址是 CPU 访问存储器的实际寻址地址，即存储单元对应 20 位的物理地址，其物理地址分布为 00000H～FFFFFH。

② 逻辑地址。图 2-12 清晰给出了物理地址到逻辑地址的演化过程。

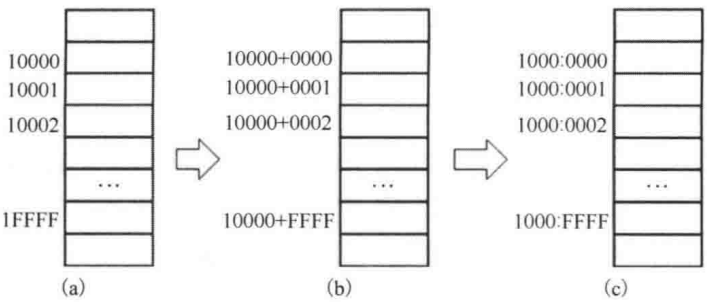


图 2-12　物理地址到逻辑地址的演化过程

图 2-12(a)是内存中的某一段，物理地址为 10000H～1FFFFH，段的首地址为 10000H。由于 8086 对段的规定，低 4 位默认为 0H（十六进制），高 16 位的 1000H 为段基址。段基址通常存在段寄存器中。其实，规定段首地址最低 4 为 0 的重要原因是段寄存器是 16 位的。而从段寄存器内容通过低位补 0 就可以直接得到段首地址。

图 2-12(b)是图 2-12(a)的另一种表示，把每个内存单元地址用段首地址和一个 16 位的地址之和来表示，这 16 位地址就是偏移地址。偏移地址即目标地址和段首地址之间的距离，或目标地址和段首地址之间的差值。比如，目标地址是 10002H 的内存单元，其偏移地址为 10002H−10000H =0002H。**注意：由于目标地址是一个段的段内单元地址，因此其偏移地址最大不可能超过 16 位，这样段内的偏移地址分布就是 0000H～FFFFH。**

图 2-12(c)是图 2-12(b)的另一种表示法。既然段首地址最低 4 位为 0，没有其他什么信息量，因此段首地址可以用 16 位的段基址表示，图 2-12(b)中的加号用冒号代替，因此图 2-12(c)所示段的第 2 号单元的地址表示为 1000H:0002H，显然它与该单元的物理地址 10002H 之间的关系是 1000H×10H+0002H。这里的 10H 为十进制数 16，表示 1000H 低位补 4 个二进制 0。

上面的 1000H:0002H 就是逻辑地址。因此，逻辑地址有两个分量：段基址和偏移地址，逻辑地址的格式是“段基址:偏移地址”。如图 2-13 所示，物理地址和逻辑地址之间的关系是：

物理地址=段基址×16+偏移地址

乘以 16 相当于 16 位的段基址最低位后添 4 个“0”。

采用分段结构的存储器中，任何一个逻辑地址都由段基址和偏移地址两部分构成，都是无符号的 16 位二进制数，程序设计时采用逻辑地址。8086 在程序运行时，根据操作属性，会自动确定目标单元的段寄存器。

如图 2-14 所示，在 CPU 运行过程中，当取指令时，选择代码段寄存器 CS，再与指令指针 IP 的内容一起形成指令所在单元的 20 位物理地址；当进行堆栈操作时，CPU 选择堆栈段寄存器 SS，再与堆栈指针 SP 或者基址指针 BP 一起形成 20 位堆栈指针；当往内存写一个数据或者从内存读一个数据时，CPU 会选择数据段寄存器 DS，再与变址寄存器 SI 和 DI 或者通用寄存器 BX 中（或指令中）的偏移值一起构成操作数所在存储单元的 20 位物理地址。

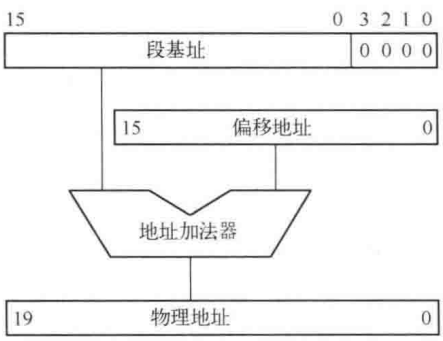


图 2-13　8086 CPU 地址完成过程

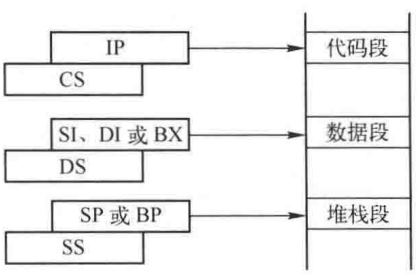


图 2-14　寄存器组合指向存储单元

8086 在执行复位信号后，从地址为 FFFF0H 开始执行程序。这个地址实际上是 CS 的内容（复位时置为 FFFFH）左移 4 位再加上 IP 的内容（复位时置为 0H）形成的。由于上述原因，在存储器安排时，将高地址端分配给 ROM（固化的只能读的存储器），在 FFFF0H 开始的几个单元中存放一条无条件转移指令，在复位时，自动转到系统初始化程序中。

4．堆栈段的使用

堆栈是在存储器中开辟的一个区域，用来存放需要暂时保存的数据和地址信息，采用“先进后出”或“后进先出”方式。8086 中的堆栈段是由段定义语句在存储器中定义的一个段，与其他逻辑段一样，可以在存储器的 1 MB 空间内任意浮动，堆栈段容量小于或等于 64 KB。段基址由堆栈寄存器 SS 指定，栈顶由堆栈指针 SP 指定。堆栈的地址增长方式是向下增长的，即随着堆栈内容的增加，堆栈指针的值是减小的。栈底设在存储器的高地址区，堆栈地址由高向低增长。

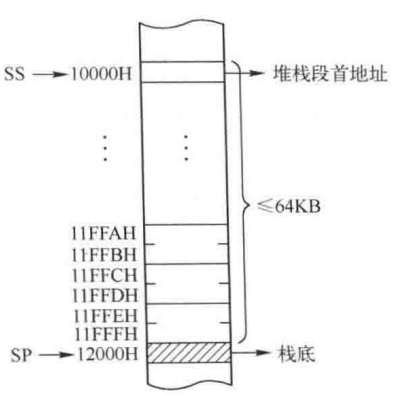


图 2-15　堆栈在存储器中的分布情况

若(SS)=1000H，(SP)=2000H（括号表示给定寄存器中存储的内容），则堆栈在存储器的分布情况如图 2-15 所示。

在 8086 中，堆栈仅以字为单位进行操作，并且堆栈中的数据项必须对准字存储，即低字

节在偶地址，高字节在奇地址，以保证每访问一次堆栈就能压入或弹出一个字的信息。

堆栈的主要操作是入栈操作（PUSH）和出栈操作（POP）。入栈操作时，总是先修改指针（SP－2⇒SP），再将信息入栈；出栈操作时，总是先将信息出栈，再修改指针（SP＋2⇒SP）。

2.2.2 8086 CPU 的输入/输出结构

8086 CPU 与外部设备的输入/输出是通过接口完成的。设计 CPU 时，人们并没有设计它与外部设备之间的接口部件，而是将输入/输出（I/O）端口设计成完全独立的部件，使它们成为通用的接口芯片。它们可将各种类型的输入或输出设备与 CPU 连接起来，构成完整的微机系统。专用的 I/O 端口芯片将在后面章节详细介绍，本节介绍 8086 CPU 的输入/输出结构。

8086 CPU 有一套灵活的 I/O 功能，不仅提供与存储空间分开的 I/O 空间，还提供与位于 I/O 空间内的设备进行数据传输用的指令。I/O 设备的地址也可位于存储空间内，这样可以利用整个指令系统和寻址方式的全部功能。对于高速的传输操作，8086 CPU 可以配置成最小模式，提供与传统的 DMA（Direct Memory Access，直接存储器存取）控制器兼容的信号线，还可配置成最大模式，与高性能通用 I/O 处理器 8089 配套使用。

1．输入/输出空间

8086 CPU 可访问的 I/O 空间所用的地址是 16 位的，这样有 64K 个 8 位端口或 32K 个 16 位端口。一个 8 位端口相当于一个存储器字节，分配 0000H～FFFFH 中的一个地址。任何两个相邻的 8 位端口可以组合成一个 16 位端口，并且类似存储器中的字。当程序访问偶端口的字时，需一次读（或写）操作；当访问奇端口的字时，则需两次读（或写）操作才能完成。

8086 CPU 提供专门的输入指令（IN）和输出指令（OUT），能够完成累加器（传输字节时是 AL，传输字时是 AX）和位于 I/O 空间的端口之间的数据传输。

8086 CPU 可以访问的 I/O 空间有 64 KB，可以满足绝大多数系统对 I/O 端口数目的要求，所以并不需要 I/O 空间像内存空间一样分段。8086 CPU 要访问某个端口，BIU 只需简单地把该端口的端口地址放到地址总线的低 16 位线上，就可以对该端口进行访问。

在 8086 CPU 指令系统中，有两种方式在 I/O 指令中指出端口地址：一种是把端口地址规定为在指令中的一个固定值，另一种是预先将端口地址送入 DX 寄存器中。

2．存储器编址的输入和输出

I/O 端口也可置于 8086 CPU 的存储空间内，只要这些设备能像存储器那样做出响应。存储器编址的输入和输出可以利用 8086 CPU 的指令系统和寻址方式的能力，进行输入和输出处理，且使程序设计更灵活。事实上，访问存储器的任何指令都能用来访问位于存储空间内的 I/O 端口，如传送指令 MOV 能在 8086 任意一个寄存器与一个端口之间传输数据，或者用“与”（AND）“或”（OR）等指令处理 I/O 设备寄存器中的各位。

8086 CPU 要为存储器编址的 I/O 付出代价：存储空间的一部分归 I/O 设备专用，减少了存储器的可用地址空间。实际上，在现代计算机系统中，对 I/O 端口的编址一般采用 CPU 设计时已给出的 I/O 访问方式，所以 8086 系统很少采用存储器编址的 I/O 方式对端口进行访问。

8086 CPU 为提供更高速的 I/O 数据传输功能，设计了满足 DMA 需要的引脚。当系统中有 DMA 控制器时，用于接管 CPU 对总线的控制权。在存储器与高速外设之间建立直接数据

块传输的高速通路，8086 CPU 配置成最小模式时，提供与 DMA 控制器兼容的 HOLD（保持）和 HLDA（保持响应）信号。DMA 控制器发出 HOLD 信号向 CPU 请求使用总线，如果 CPU 正在使用总线，将完成当前的总线周期（总线周期将在 2.3 节介绍），然后发出 HLDA 信号，将总线使用权交由 DMA 控制器管理。在 HOLD 信号变成无效前，CPU 不得使用总线。

2.2.3 8086 CPU 的最小模式和最大模式系统

以 8086 CPU 构成的微型计算机系统有最小模式和最大模式两种配置。最小模式是单机系统，系统中所需的控制信号全部由 8086 CPU 本身直接提供；最大模式可以构成多处理机系统，系统中所需的控制信号由总线控制器 8288 提供。CPU 工作模式的选择是由硬件决定的，当 CPU 的引脚 MN/$\overline{MX}$ 接高电平（+5 V）时，构成最小模式；当 MN/$\overline{MX}$ 接低电平（地）时，构成最大模式。两种不同模式下的主要区别是 8086 CPU 的第 24～31 号引脚具有不同的功能。

1．最小模式系统

8086 CPU 最小模式的基本配置如图 2-16 所示。其中，除了存储器、I/O 芯片和基本时钟发生器（8284A），还有用于地址的锁存器 8282（或 8283）、用于数据的缓冲器 8286（或 8287）。在 8086 系统中，地址线和数据线是复用的，所以地址锁存器是必要的。这些复用的引脚在某时刻只能体现地址线或数据线之一，所以在对存储器进行访问时，要先将地址输出。此时，复用的引脚是地址线，再利用地址锁存器保存这些地址。

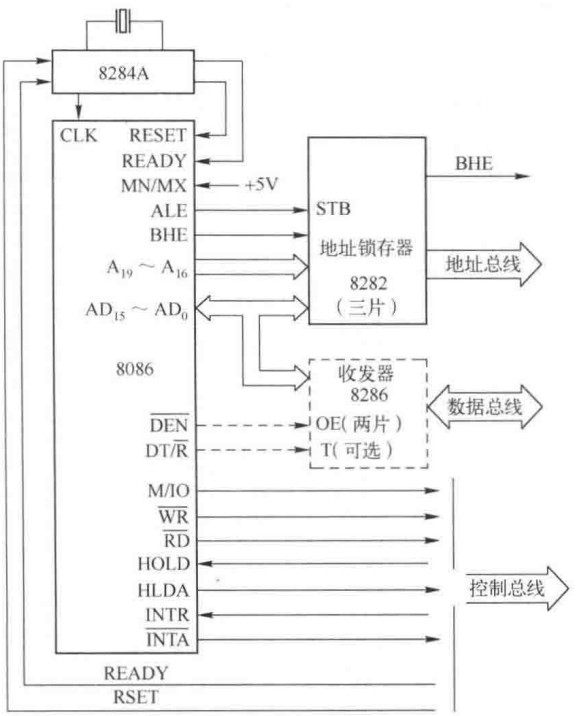


图 2-16　8086 CPU 最小模式

之后，这些引脚才是数据线，将数据读出或写到存储器。在某时刻，处理器把某个存储单元的地址发送到地址总线上，经锁存器将这些地址保存起来。只有这样，处理器才能把数据通过某些共享的引脚送到数据总线上，完成对存储器的读/写操作。从这个意义上讲，在 8086 最

小模式系统中，数据缓冲器就不是必要的。

8282（或 8283）是带三态缓冲器的通用 8 位数据锁存器。8282 的输入信号和输出信号是同相的，而 8283 的输入信号和输出信号是反相的。由于地址输出是单向的，因此选用单向的数据锁存器 8282 作为地址锁存。

8286（或 8287）是带三态输出的 8 位双向数据缓冲器，专用于需要双向传输的数据总线接口。8286 的输入或输出时信号是同相的，而 8287 的输入或输出时信号是反相的。在 8086 最小模式系统中，可以不用数据收发缓冲器 8286（或 8287），这时可将 CPU 的地址/数据总线直接与存储器或 I/O 端口的数据线相连。

最小模式系统还允许接入其他要求共享总线的设备。典型的例子是接入 DMA 控制器 8237 芯片，相关信号线是 HOLD 和 HLDA（2.2.2 节已介绍）。

2．最大模式系统

8086 CPU 最大模式的基本配置如图 2-17 所示，与最小模式系统相比，主要区别是最大模式系统中增加了总线控制器 8288 和总线仲裁器 8289。8086 CPU 输出的状态信号 $S_2 \sim S_0$ 同时送给 8288 和 8289，由 8288 输出 8086 CPU 系统所需的控制信号，而 8289 总线仲裁器对系统中多个处理器提出共享总线资源的要求做出裁决。因此，8086 CPU 的最大模式由于 8288 和 8289 的存在，很容易构成一个多处理器系统。

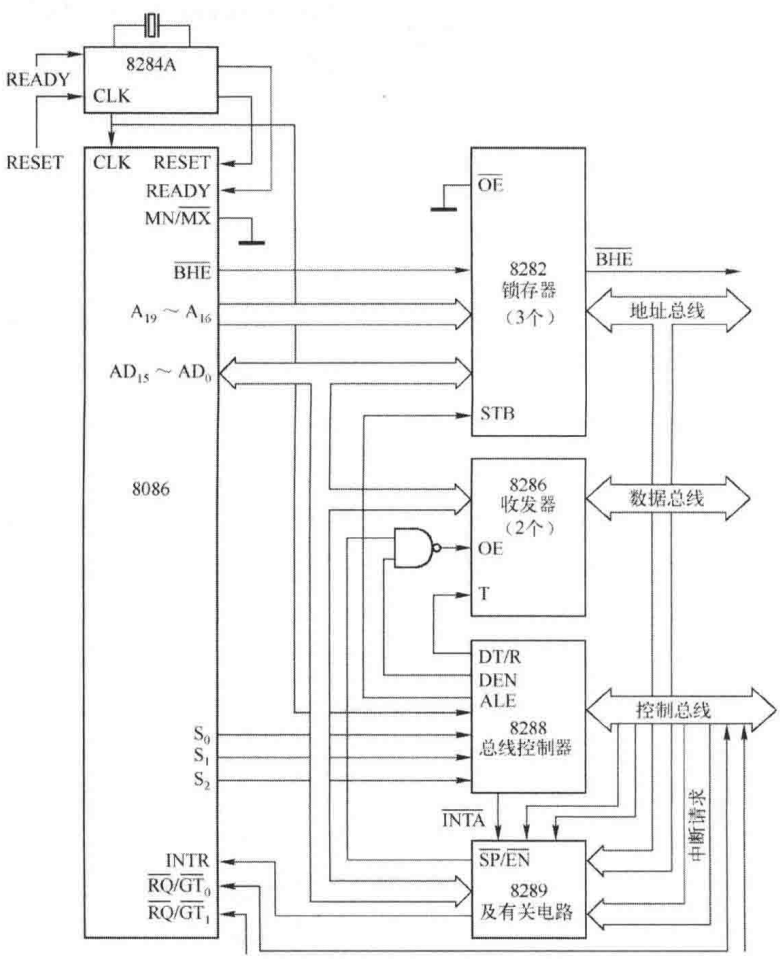


图 2-17　8086 CPU 最大模式

（1）总线控制器 8288

在最大模式中，8288 主要用来解释状态码 $\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$ 表示的总线状态，并产生 DT/$\overline{R}$、$\overline{DEN}$、MCE/$\overline{PDEN}$ 和 ALE 等一系列总线命令和控制信号。由 8288 产生的地址锁存允许信号 ALE、数据发送接收控制信号 DT/$\overline{R}$、数据传输允许信号 $\overline{DEN}$ 的功能与最小模式中直接由 8086 CPU 发出的控制信号相同，只是 $\overline{DEN}$ 信号的极性相反，因而系统中增设了一个非门。由 8086 CPU 输出的总线状态信号 $\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$ 与 8288 输出总线命令信号间的对应关系如表 2-9 所示。

表 2-9　8288 可提供的总线命令信号

总线状态信号			CPU 状态	8288 命令
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$		
0	0	0	中断响应	$\overline{INTA}$
0	0	1	读 I/O 端口	$\overline{IORC}$
0	1	0	写 I/O 端口	$\overline{IOWC}$，$\overline{AIOWC}$
0	1	1	暂停	无
1	0	0	取指令	$\overline{MRDC}$
1	0	1	读存储器	$\overline{MRDC}$
1	1	0	写存储器	$\overline{MWTC}$，$\overline{AMWC}$
1	1	1	无作用	无

（2）总线仲裁和总线仲裁器 8289

当 8086 构成最大模式系统时，HOLD 和 HLDA 引脚的信号演变成请求/同意信号 $\overline{RQ}/\overline{GT_0}$ 和 $\overline{RQ}/\overline{GT_1}$。这些双向信号传输引脚可以使 8086 和另外两个处理器共享局部总线。首先，要求使用总线的处理器通过 $\overline{RQ}/\overline{GT_0}$ 或 $\overline{RQ}/\overline{GT_1}$ 向 CPU 发出总线请求；当 CPU 接收这一请求后，即通过 $\overline{RQ}/\overline{GT_0}$ 或 $\overline{RQ}/\overline{GT_1}$ 回送总线同意信号。当占用总线的处理器用完总线后，它把一个脉冲发送给 8086，说明请求结束，CPU 这时可以收回对总线的控制权。共享总线的处理器数目一般是有限的，因为 8086 CPU 只提供两条请求/同意信号线，在硬件上保证 $\overline{RQ}/\overline{GT_0}$ 比 $\overline{RQ}/\overline{GT_1}$ 具有较高的优先级，因此当两个请求同时到达 CPU 时，8086 先应答通过 $\overline{RQ}/\overline{GT_0}$ 引脚提出请求的处理器。当 CPU 正在处理 $\overline{RQ}/\overline{GT_1}$ 上早先的请求时，来自 $\overline{RQ}/\overline{GT_0}$ 的请求要等到 $\overline{RQ}/\overline{GT_1}$ 的处理器释放对总线的控制权后，才能获得 CPU 的响应。

如果希望在一个系统中包含多个共享总线的处理器，即多处理器系统，就必须对总线进行仲裁，以保证优先级别高的处理器优先使用总线，这就需要一种协调各处理器使用共享总线的方法。总线仲裁器 8289 和总线控制器 8288 一起，为 8086 系统提供这种控制。

首先，在多处理器系统中，每个处理器（通常指主控者）必须配备一个 8288 总线控制器和一个 8289 总线仲裁器。总线上的每个处理器都分配不同的优先权。当总线请求同时到来时，8289 解决争用问题，经过仲裁把总线使用权转让给具有较高优先级的处理器。在解决总线争用的问题上，8289 采用 3 种优先权处理技术：并行优先权裁决方式、串行优先权裁决方式和循环优先权裁决方式（更详细的内容可查询 8289 总线仲裁器手册）。8289 对多处理器系统中每个处理器而言是透明的，每个处理器好像自己独占了总线一样。

2.3　8086 CPU 内部时序

各种微处理器的工作过程实际上就是执行指令的过程。它们所进行的操作是周期性的，即

取指令、执行指令、再取指令、再执行指令……由于传统的计算机对指令采用串行解释方式，因此总是一条指令执行完了再去取下一条指令，直到整个程序执行完毕。这种工作方式的优点是控制简单，指令之间不会产生任何关联，但速度慢，系统吞吐率比较低。

8086 CPU 因设置了可独立操作的指令执行部件（EU）和总线接口部件（BIU），且两者有明确的分工，所以系统的效率得到了提高。

为了更好地理解这一点，先来看 BIU 的工作情况。BIU 直接负责 CPU 与存储器和 I/O 端口交换数据，它的工作包括从存储器取出指令送入指令队列中，或者取出操作数去参加 EU 中的运算，或者将 EU 中的运算结果写入存储器中，而这些操作都要经过系统外部总线来完成。在 8086 CPU 中，把 BIU 完成一次访问存储器操作所需的时间称为一个总线周期。总线周期实际上就是一次访问存储器所需要的时间，即存储器的一个存取周期。在理想情况下，BIU 可处于连续工作状态，不断地访问存储器，或取指令，或读/写操作数。

再来看 EU 的工作情况。EU 负责执行指令，只需从指令队列中取得指令，并分析执行它。在指令执行过程中，可以根据需要随时要求 BIU 访问存储器，取操作数或写运算结果。关键的是，EU 的操作与 BIU 访问存储器的操作可以并行进行，因此在理想的情况下，EU 也可处于连续工作状态，不断地执行从指令队列中得到的指令。所谓“在理想情况下”，说明实际上 BIU 和 EU 不可能完全处于连续工作状态，因为 BIU 有可能由于 EU 执行某些复杂指令时内部操作时间很长，而不需要访问存储器，这时 BIU 处于空闲状态而不进入总线周期；另一方面，EU 有时需要等待 BIU 从存储器取出操作数后才能进行运算，尤其是遇到转移类指令时，就有可能使原来指令队列中已取出的指令全部作废，要等 BIU 重新从存储器中取出目标地址中的指令后，才能继续执行下条指令。EU 的内部操作过程可被 BIU 的总线周期覆盖，所以可以不考虑 EU 的内部操作时序。

在 8086 CPU 中，每个总线周期至少包含 4 个时钟周期（$T_1 \sim T_4$）。时钟周期是微处理器操作的最小单位，是由系统时钟的频率确定的。BIU 总是在 T_1 周期时将存储器的 20 位物理地址（或 16 位 I/O 端口地址）送上总线，在 $T_2 \sim T_4$ 周期期间，通过总线进行数据传输。

1．最小模式系统的读/写总线周期

（1）8086 CPU 读总线周期

最小模式下，8086 CPU 的读总线周期时序图如图 2-18 所示。

在 T_1 时钟周期，BIU 将被访问的存储单元的 20 位物理地址 $A_{19} \sim A_0$ 和总线高位有效信号 $\overline{BHE}$ 一起送到总线上。在地址锁存允许信号 ALE 的控制下，这些地址被锁存到 8282（或 8283）地址锁存器中，然后输出到地址总线上。以后由 $M/\overline{IO}$ 信号确定读存储器还是读 I/O 端口。

在 T_2 周期时，$AD_{15} \sim AD_0$ 成为高阻悬空状态，$A_{19}S_6 \sim A_{16}S_3$ 和 $\overline{BHE}/S_7$ 立即成为状态信息输出，与此同时，$\overline{RD}$ 信号有效，从而启动被选的存储器或 I/O 端口。

如果被选的存储器或 I/O 端口在 T_3 周期时来得及读出数据送到数据总线上，它们就将 READY 置为有效（高电平）。CPU 在 T_3 周期时钟脉冲的上升沿得知 READY 信号有效后，就在 T_3 周期结束时，在 $DT/\overline{R}=0$ 和 $\overline{DEN}=0$ 信号的控制下，将数据总线上的 16 位（或 8 位）有效数据经数据收发器 8286（或 8287）缓冲后向 CPU 输入，从而完成读存储器的任务。

如果配合工作的存储器或 I/O 端口由于本身速度或其他原因，来不及在 T_2 状态时读出所需信息，那么必须在 T_3 周期时钟脉冲的上升沿时使 READY 无效（低电平）。当 CPU 在此时得知 READY 信号无效时，就会在 T_3 周期之后插入一个等待周期 T_w，然后在 T_w 的时钟脉冲上升

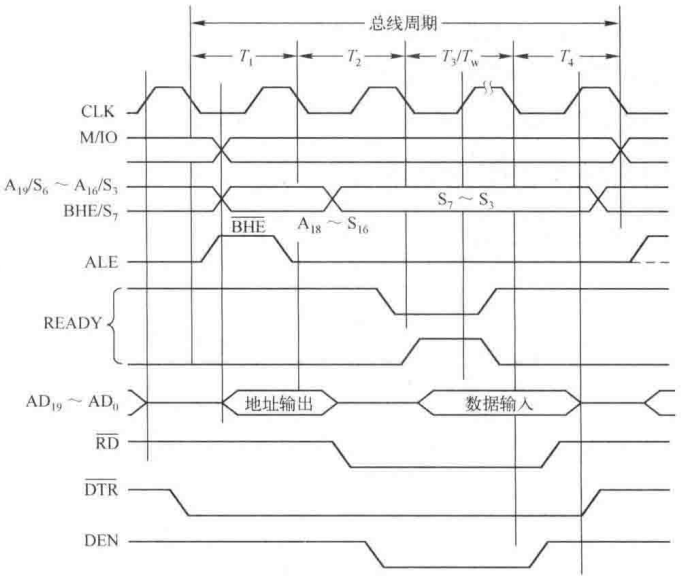


图 2-18　8086 CPU 的读总线周期时序图（最小模式）

沿再次测试 READY 信号，若还是无效则继续插入一个新的等待周期，直至 READY 有效为止。在插入 T_w 期间，其他控制信号保持同 T_3 状态时相同，在一个总线周期可插入若干个 T_w 周期，以协调高速 CPU 与低速存储器或 I/O 端口之间的数据传输。

（2）写总线周期

8086 CPU 在最小模式下的写总线周期时序图如图 2-19 所示，与读总线周期大同小异。下面主要说明它们的不同之处。

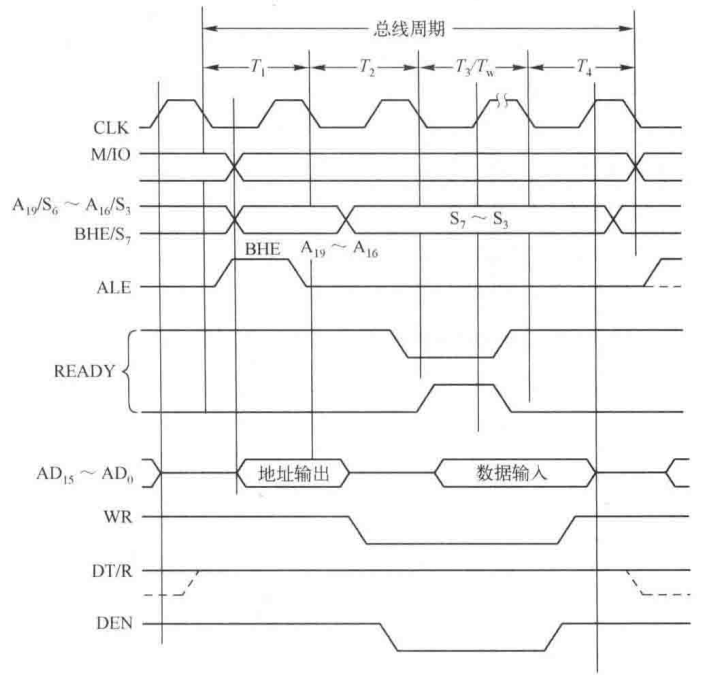


图 2-19　8086 CPU 的写总线周期时序图（最小模式）

在写总线周期，地址的传输过程与读总线周期完全相同，而当输出地址被锁存后，在地址数据复用总线 $AD_{15} \sim AD_0$ 上会输出 16 位数据，同时当 T_w 有效时，向存储器或 I/O 端口发出写

命令，要求将数据总线上的数据写入指定的存储单元或 I/O 端口中去。在写总线周期中，DT / $\overline{R}$ 线应输出高电平，使数据缓冲器 8286（或 8287）呈输出状态。必要时，存储器或 I/O 端口可以通过 READY 信号要求 CPU 在 T_3 和 T_4 状态之间插入等待周期 T_w，以延长写入过程。

2．最大模式系统中 8086 CPU 的读/写总线周期

图 2-20 和图 2-21 分半为最大模式系统中 8086 CPU 的读和写总线周期时序图。

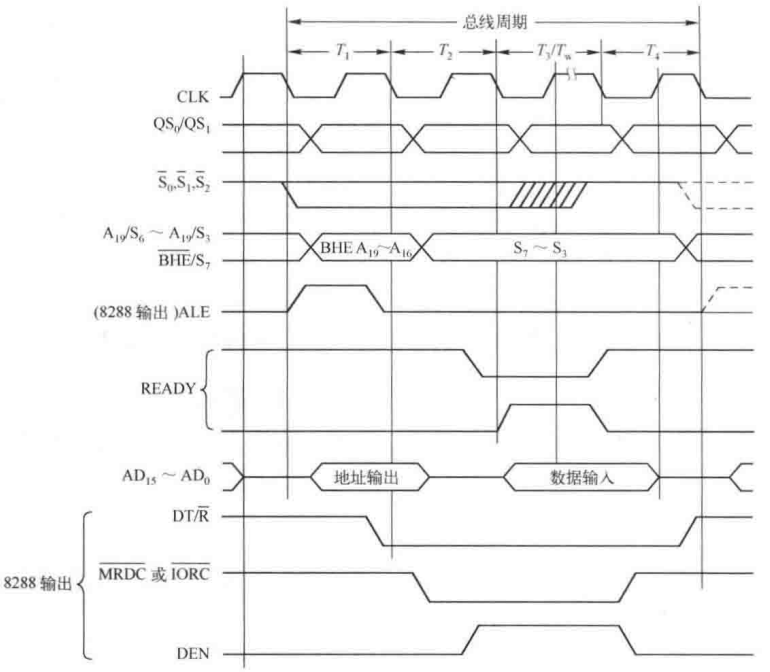


图 2-20　8086 CPU 的读总线周期时序图（最大模式）

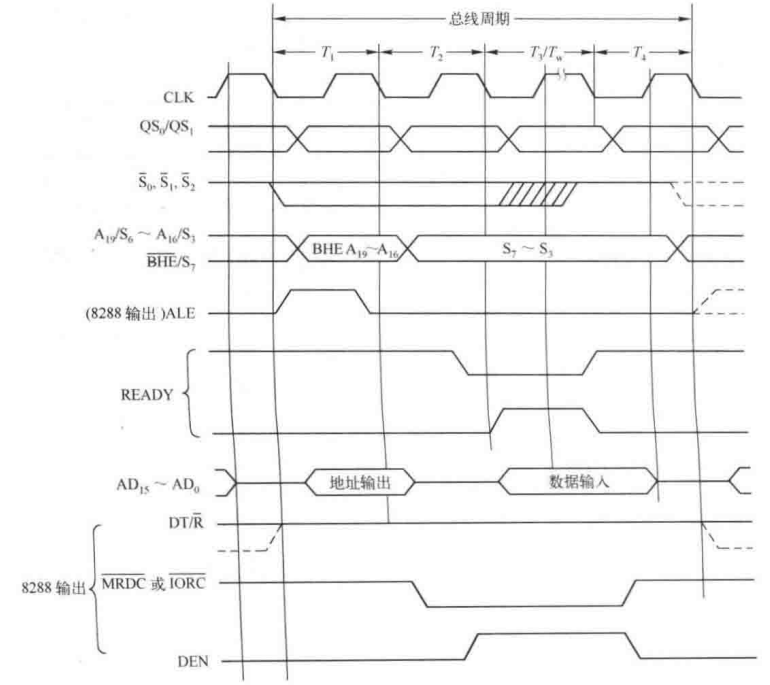


图 2-21　8086 CPU 的写总线周期时序图（最大模式）

同 8086 的最小模式相比，最大模式中增设了总线控制器 8288，因此有一些控制信号不再由 CPU 直接给出，而由 8288 依据 CPU 送来的三位状态信号 $\overline{S}_2$、$\overline{S}_1$ 和 $\overline{S}_0$ 译码后提供。

在图 2-20 和图 2-21 中，从总线周期的 T_1 时钟周期开始，CPU 就输出 $\overline{S}_2$、$\overline{S}_1$ 和 $\overline{S}_0$。8288 根据这些状态信息的不同编码分别输出对存储器或 I/O 端口的控制命令，完成读或写操作功能（详见表 2-9）。

习 题 2

1．8086 CPU 由哪两部分构成？它们的主要功能是什么？

2．8086 CPU 预取指令队列有什么好处？8086 CPU 内部的并行操作体现在哪里？

3．8086 CPU 中有哪些寄存器？各有什么用途？

4．下列情况下应判定哪个标志位并说明其状态。

（1）比较两个无符号数是否相等。

（2）两个无符号数相减后比较大小。

（3）两数运算后结果是正数还是负数。

（4）两数相加后是否产生溢出。

5．简述 8086 CPU 中物理地址的形成过程。8086 CPU 中的物理地址最多有多少个？逻辑地址呢？

6．8086 CPU 中的存储器为什么要采用分段结构？有什么好处？

7．在 8086 存储器中存放数据字时有“对准字”和“非对准字”之分，请说明它们的差别。

8．8086 CPU 工作在最小模式和最大模式系统中的主要区别是什么？各有什么主要特点？

9．在某系统中，已知当前(SS)=2360H，(SP)=0800H，请说明该堆栈段在存储器中的物理地址范围。若往堆栈中存放 20 字节数据，那么 SP 的内容为什么值?

10．已知当前数据段位于存储器的 B4000H～C3FFFH 范围内，则 DS 段寄存器的内容为多少？

11．8086 CPU 中为什么一定要有地址锁存器？需要锁存哪些信息？

12．8086 CPU 的读/写总线周期各包含多少个时钟周期？什么情况下需要插入等待周期 T_w？插入多少个 T_w 取决于什么因素？

13．若已知当前(DS)=7F06H，在偏移地址为 0075H 开始的存储器中连续存放 6 字节的数据，分别为 11H、22H、33H、44H、55H 和 66H。请指出这些数据在存储器中的物理地址。

14．某程序在当前数据段中存有两个数据字 0ABCDH 和 1234H，它们对应的物理地址分别为 3FF85H 和 40AFEH，若当前(DS)=3FB0H，请说明这两个数据字的偏移地址，并用图说明它们在存储器中的存放格式。

第 3 章　8086 指令系统

本章导读

- 8086 指令的特点
- 8086 的寻址方式
- 8086 的指令格式
- 8086 的数据类型
- 8086 的指令集

计算机是通过执行指令序列来工作的，每种计算机都有一组指令集提供给用户使用，这组指令集称为该计算机的指令系统。不同 CPU 的计算机使用不同的指令系统，8086 CPU 的指令系统不但包含 8 位机的全部指令，而且增加了一些功能较强的 16 位数据处理指令，如乘法、除法指令，因而同时具有 8 位和 16 位的处理能力。

3.1　8086 指令的特点

任何一条指令都由操作码（Opcode）和操作数（Operand）两部分组成。

指令中的操作码部分表明指令的操作性质，一般有 1～2 字节；操作数部分既可以表示参加操作的数，也可以表示参加操作的数所在位置。当表示参加操作的数所在位置时，操作数部分又称为地址码。操作数部分有 0～4 字节。在一条指令中，操作码部分是必需的，而操作数部分可能隐含在操作码中，或者由操作码后面的指令给出。

1．灵活的指令格式

8086 的指令中有 1 字节长，此时指令中的操作数部分隐含在操作码中，大部分对 16 位寄存器操作的指令都只有 1 字节长；当操作数在内存中，编程又需要复杂的寻址方式时，有的指令多达 6 字节长。比如，1 字节指令：

指令助记符	指令的十六进制数代码
DAA	27H

该指令是十进制数加法调整指令，这个十进制数就隐含在寄存器 AL 中。

再如，6 字节指令：

指令助记符	指令的十六进制数代码

MOV [BX+SI+1020H], 3040H C78020104030

表达式 BX+SI+1020H 代表一个地址，该指令是把数 3040H 送到该地址指示的内存单元中。

2．指令格式的一对多形式

用助记符编写的指令最终要翻译成二进制代码由 CPU 执行。为了方便用户理解，可用两种不同的助记符描述同一问题，如 JE/JZ。当两个无符号数进行比较操作时，用户理解为相等则转移，也可理解为 ZF=1 则转移。虽然助记符不同，但其二进制代码是一样的。

3．较强的运算指令

当用 8 位机完成乘法运算时，只能用连加或对位权移位等方式编写一段程序实现。而 8086 中有乘法、除法指令，给用户提供了极大方便。

4．指令有极强的寻址能力

在微机系统中，参加操作的数据有可能在 CPU 的寄存器、内存或外部设备中。指令中如何提供数据所在位置，以便提高程序执行效率，这就需要 CPU 提供强有力的寻址能力。细分 8086 指令的寻址方式多达 9 种，特别是对内存的寻址方式十分灵活。

5．指令有处理多种数据的能力

8086 指令能够处理 8 位/16 位数、带符号/无符号数以及压缩 BCD 数/非压缩 BCD 数。带符号数和无符号数有相应的乘法、除法指令，压缩 BCD/非压缩 BCD 数有相应的调整指令。

3.2 8086 CPU 的寻址方式

寻址方式是指令中用于说明操作数所在地址的方法，或寻址方式是指令寻找操作数的方法。可以说，寻址方式的多少也是衡量 CPU 功能的指标。8086 CPU 的寻址方式十分丰富，32 位微机的寻址方式在其基础上稍有增加。

8086 CPU 指令中的操作数有一个或两个，个别指令有三个，称为源操作数和目的操作数。除了目的操作数不允许为立即数（即立即寻址），其余寻址方式均适合源操作数和目的操作数。

3.2.1 8086 寻址方式的说明

当操作数在 CPU 内部的寄存器中时，寻址简单且有效。遗憾的是，CPU 内部的寄存器数目有限，大量的数据和运算结果需要存放在内存中。8086 CPU 的寻址能力是 1 MB 内存空间，指令中如何给出操作数所在地址，从而提高程序编制及指令执行效率，就在于有多种形式的寻址方式。当操作数从外部设备输入或输出时，寻址方式有其特殊性。虽然外部设备也像内存单元一样，用地址进行访问（与内存地址区别，一般称为 I/O 端口地址），但由于外设的多样性，使传输的数据宽度有 8 位和 16 位之分，同时一个微机系统配置的外部设备数目不同，I/O 端口地址的编址也就不同（如 8 位地址/16 位地址）。

8086 CPU 对外部设备的寻址方式就是针对以上需要设置的。

为了使读者便于理解 8086 CPU 的寻址方式，先说明以下两个问题。

1．有效地址（Effective Address，EA）

当操作数在内存中时，指令的地址码（操作码）给出所访问的内存单元的逻辑地址。在寻址方式中，逻辑地址由多个分量组合而成，又被称为有效地址，如图 3-1 所示。

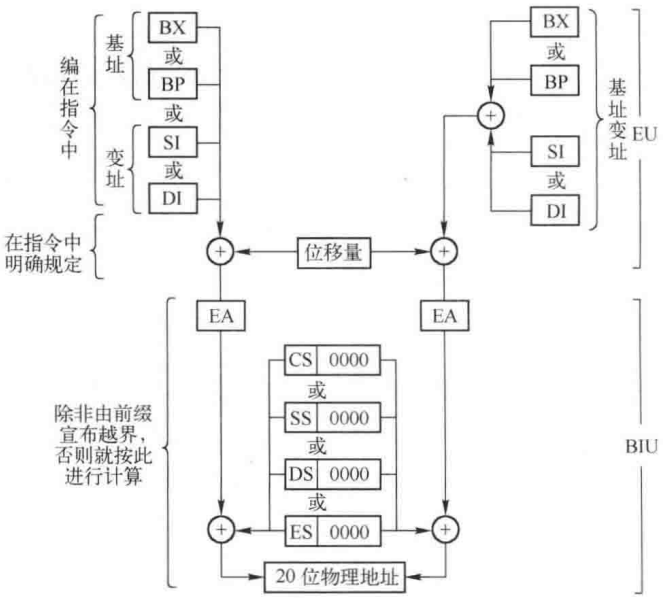


图 3-1　存储器地址

2．数据传输指令

为了理解寻址方式，需要以 8086 的指令为例，这里先介绍 MOV 指令。其格式为：

```
MOV     目的操作数，源操作数
```

目的操作数和源操作数均可采用不同的寻址方式，但两个操作数的类型必须一致。

3.2.2　寻址方式介绍

1．立即寻址（immediate addressing）

操作数就在指令中，紧跟在操作码后面，作为指令的一部分存放在内存的代码段中，这种操作数称为立即数。例如：

```
MOV     AX, 34EAH
```

该指令是将一个 16 位的十六进制源操作数 34EAH 送入 8086 CPU 的 16 位寄存器 AX。执行后，寄存器 AX 的内容为 34EAH。又如：

```
MOV     BL, 20
```

该指令是将一个 8 位的十六进制源操作数 14H 送入 8086 CPU 的 8 位寄存器 BL。执行后，寄存器 BL 的内容为 14H（即 0001 0100）。

说明：当指令中的立即数后面不加字母 H 时为十进制数，汇编时该立即数由汇编程序以二进制数形式存于代码区。

2．寄存器寻址（register addressing）

若操作数在寄存器中，指令中的源操作数和目的操作数都可用这种寻址方式。对于 16 位

操作数，寄存器可以是 AX、BX、CX、DX、SI、DI、SP、BP；对于 8 位操作数，寄存器可以是 AH、AL、BH、BL、CH、CL、DH、DL。下面两个例子中，目的操作数就是寄存器寻址方式。例如：

```
MOV     BP, SP
```

该指令执行后，寄存器 SP 的内容送入寄存器 BP。又如：

```
MOV     AX, 1234H
MOV     AL, AH
```

第一条指令执行后，寄存器 AX 的内容为 1234H，再执行下一条指令，则寄存器 AH 的内容为 12H，而寄存器 AL 的内容也为 12H。

3．直接寻址（direct addressing）

当操作数在内存单元时，在指令中必须给出被访问内存单元的逻辑地址，由 8086 CPU 的存储器管理部件计算出有效地址（按表 3-1 所示的方式），再转换成物理地址，才能对被选定的内存单元进行访问（读或写）。“存储器读”是指将内存单元的数据通过数据线传送给目的地，“存储器写”是指将数据通过数据线传送至内存单元中。

表 3-1　存储器存取时约定段和可修改段的基数

存储器存取方式	约 定 段	可超越使用的段	偏 移 量
取指令	CS	无	IP
堆栈操作	SS	无	SP
源字符串	DS	CS，ES，SS	SI
目的字符串	ES	无	DI
用 BP 做基址	SS	CS，ES，DS	有效地址
通用数据读写（BP 作为基址除外）	DS	CS，ES，SS	有效地址

当指令中的源操作数或目的操作数采用直接给出被访问内存单元的逻辑地址的方式时，这种寻址方式被称为直接寻址。例如：

```
MOV     AX, [3E4CH]
```

其中，源操作数用直接寻址方式，指令中由“[ ]”内给出的是被访问内存单元的逻辑偏移地址，逻辑段地址隐含在寄存器 DS 中。指令中，EA 的内容为 3E4CH，物理地址为(DS)×10H+3E4CH。由于存储器以 8 位数据为单元组织（字节），在传输 16 位数据时，是将两个连续的内存单元组成的 16 位数据在数据线上并行传输，低地址单元的 8 位数据送到寄存器 AL 中，高地址单元的 8 位数据送到寄存器 AH 中。其他寄存器的写入类似。

又如：

```
MOV     [1234H], BL
```

其中，目的操作数采用直接寻址方式，将寄存器 BL 的 8 位数据送到逻辑偏移地址为 1234H 的单元中。其中，EA 为 1234H，物理地址为(DS)×10H+1234H。

再如：

```
MOV     ES:[1234H], BL
```

该例为读者提供有关段超越的概念。第 2 章描述 8086 CPU 的寄存器时，提到过段寄存器与表示逻辑偏移地址的寄存器的隐式关系，如 CS:IP、SS:SP、DS:BX、DS:SI、DS:DI 和 DS:直接偏移地址。此例中，EA 为直接偏移地址 1234H，而物理地址为(ES)×10H+1234H。

说明：① 在以上例子中，指令中的直接地址是用十六进制数形式给出的。在实际编程应用中，一般使用符号地址，即由事先定义的符号表示逻辑偏移地址。例如：

```
MOV     AX, BUFF
```

② 指令中使用直接寻址方式时，在没有声明的情况下，逻辑段地址是指寄存器 DS 的内容；如果用户编程使用其他段寄存器，则需在指令中给出。例如：

```
MOV     ES:BUFF, DX
```

4．寄存器间接寻址（register indirect addressing）

当被访问的操作数在内存单元时，使用该寻址方式。与直接寻址方式不同的是，内存单元的逻辑偏移地址通过间接的方式给出，即先将被访问内存单元的逻辑偏移地址传送给寄存器，在指令中再由寄存器给出被访问的内存单元的逻辑偏移地址。例如：

```
MOV    SI, 61A8H
MOV    DX, [SI]
```

可以看出，第一条指令将偏移地址传给寄存器 SI，第二条指令采用寄存器间接方式，给出被访问内存单元的逻辑偏移地址。

说明：① 第一条指令的源操作数是立即数，在第二条指令中将此立即数作为偏移地址使用；② 若没有声明，寄存器间接寻址方式中的逻辑段地址使用隐式用法，即 DS 与 BX、SI 和 DI 寄存器组成物理地址，SS 与 BP、SP 寄存器组成物理地址；③ 8086 中的寄存器 AX、CX、DX 不能在寄存器间接寻址中使用。

5．基址/变址寻址（based/indexed addressing）

基址/变址寻址方式中提出了“位移量”的概念，即在寄存器间接寻址给出的地址信息上加一个相对位移量，所以也称为相对寻址。位移量是一个带符号的 16 位十六进制数。当使用寄存器 BX 或 BP 时，称为基址寻址；使用寄存器 SI 或 DI 时，称为变址寻址。例如：

```
MOV     CX, 36H[BX]
```

也可以写为：

```
MOV     CX, [BX+36H]
```

其中，EA 为 36H+(BX)，物理地址为(DS)×10H+36H+(BX)。又如：

```
MOV     -20[BP], AL
```

其中，EA 为(BP)–14H，物理地址为(SS)×10H–14H+(BP)。

说明：① 这种寻址方式适合寻找一维表格存储在内存中的操作数，用基址或变址寄存器存放表格的首地址，用位移量表示数据元素的位置，可以很方便地找到该操作数；② 这种寻址方式同样可以使用段超越。

6．基址+ 变址寻址（based indexed addressing）

如果用户的数据结构比较复杂，如二维表，考虑编程方便和程序执行的效率，那么可以采用基址+变址寻址方式。其有效地址 EA 由 3 部分组成：基址寄存器 BX 或 BP 的内容+变址寄存器 SI 或 DI 的内容+位移量。物理地址由基址寄存器按规则选择段寄存器，也可使用段超越。例如：

```
MOV     AX, 8AH[BX][SI]
```

也可以写为：

```
MOV     AX, [BX+SI+8AH]
```

其中，EA 为 8AH+(BX)+(SI)，物理地址为(DS)×10H+8AH+(BX)+(SI)。

7．串寻址（string addressing）

串寻址方式仅在 8086 的串指令中使用。串指令中的操作数由其他指令提供，且操作数在内存单元中。因此，规定源操作数的逻辑地址为 DS:SI，目的操作数的逻辑地址为 ES:DI。当执行串指令的重复操作时，根据设定的方向标志（DF）的内容，SI 和 DI 会自动调整。

8．I/O 端口寻址（I/O port addressing）

当操作数在外部设备时使用 I/O 指令。微机系统中采用地址来访问不同的外部设备。为了与内存地址区别，该地址被称为端口地址。由于外部设备的多样性，此时有两种不同的寻址方式访问 I/O 端口。当外部设备地址用 8 位寻址时，使用直接端口寻址方式。在 I/O 端口寻址方式中，I/O 地址仅有 256 个，即 0～255（00H～FFH，高 8 位为 00H）。在 I/O 指令中直接给出 0～255 中被选定的地址。当外部设备地址为 16 位（或超过 8 位）寻址时，采用寄存器间接寻址方式，用寄存器 DX 作为间接寻址寄存器。此时的端口地址多达 216 个。

由于外部设备的数据宽度不同，输入指令中目的操作数可为 AL 或 AX，输出指令中源操作数可为 AL 或 AX。例如：

```
IN      AL, 25H
```

将端口地址为 25H 的输入设备中的 8 位数据送到寄存器 AL 中。又如：

```
MOV     DX, 3E4H
OUT     DX, AL
```

将 AL 寄存器的数据传输至端口地址为 3E4H 的外部设备。

3.3 8086 CPU 的指令格式及数据类型

用汇编语言编写的源程序输入计算机后，由汇编程序将它翻译成机器指令组成的机器语言程序，计算机才能识别并执行。此过程一般不必由人工来干预，但必要时也可由程序员来完成，称为“手工汇编”。本节简略介绍机器指令格式及组成原理。

1．操作码

指令由操作码和操作数（地址码）组成。8086 CPU 的指令长度是可变的，一条指令一般由 1～6 字节组成（加上前缀字节，最长可为 7 字节）。指令的操作码采用二进制代码表示本指令所执行的操作，通常用指令的第 1 字节表示。有时由于用 8 位不够，因此在指令的第 2 字节中还可能占 3 位。除此以外的其他字节（或位）用来表示操作数。

2．指令中的操作数

单操作数指令中只有一个操作数，例如：

指令助记符		指令的十六进制代码
INC	AX	40H
INC	BX	43H

双操作数指令中有两个操作数，例如：

指令助记符		指令的十六进制代码
MOV	AL, 04	B004H
MOV	AX, 04	B80400
POP	DI	5FH
ADD	AX, BX	01D8H
MOV	BX, 1234H	BB3412
PUSH	AX	50H

在 8086 指令系统中，大多数指令中只有 1～2 个操作数，也有少数指令中有 3 个操作数，不过其中 1 个操作数隐含在操作码中。例如：

```
ADC     AX, BX
```

该指令完成操作数 AX、BX 和 CF 位相加。又如：

```
LDS     SI, [BX]
```

该指令有 2 个目的操作数，将源操作数 DS:[BX]组成的物理地址的连续 4 字节单元中的内容分别送到寄存器 SI 和 DS 中。

3．指令中的数据类型

在 8086 CPU 中，指令中处理的数据可为 8 位和 16 位，若要求处理位数更多的数，则需要编写程序，将其按从低到高的顺序按 8 位或 16 位进行处理。

① 无符号数：没有符号的 8 位数，其值为 00H～FFH（0～255）；没有符号的 16 位数，其值为 0000H～FFFFH（0～65535）。

② 带符号数：由于符号占用 8 位中的 1 位（最高位），故 8 位中仅有 7 位表示数据，且用其补码表示，其值范围为–80H～+7FH（–128～+127）；同样，16 位中仅有 15 位表示数据，且用其补码表示，其值范围为–8000H～+7FFFH（–32768～+32767）。

在程序中，带符号数的符号也像数据一样被处理。对带符号数的加、减运算结果要考虑溢出问题，对乘、除运算要使用正确的指令。

③ ASCII：西文字母和一些常用符号用 ASCII 编码表示。从计算机键盘输入的数据或符号，在机器中得到的是其对应的 ASCII 编码；将数据或符号在屏幕上显示，也必须先将其转换成 ASCII 编码。在程序设计中，ASCII 编码用单引号括起来。

④ BCD 数（压缩 BCD 和非压缩 BCD）：采用 BCD 数是由于十进制数符合人们的书写和阅读习惯，特别是在屏幕上显示运算结果，出现有字母的数据总是很别扭。用 4 位二进制数表示 1 位十进制数的编码，在运算时会出现一些问题，好在有相应的调整指令可以解决。

3.4 8086 的指令集

8086 指令系统按功能可分为 6 种。

① 数据传输类：其功能是将源操作数的内容送到目的操作数中。这类指令形式较多，寻址方式也十分丰富，在程序设计中使用频率很高。

② 算术运算类：其功能是完成加、减、乘、除运算。每次运算会对 6 种状态标志产生影

响。使用这类指令时，要注意指令的书写形式（源操作数和目的操作数的存放位置）以及乘/除运算中参加运算的数据类型。

③ 逻辑运算类：其功能有两种，一是完成逻辑“与”“或”“非”“异或”和测试，二是对数据的移位操作。这类指令是按二进制位进行的，所以又被称为位操作指令。这种运算也对状态标志位产生影响。

④ 串操作类：有不同的寻址方式，段寄存器和变址寄存器按 DS:SI 和 ES:DI 组合寻址。这类指令在执行前根据编程需要设置 DF 标志，在这些指令前加上指令前缀，可以完成指令的循环操作。

⑤ 程序控制类：程序中执行跳转、调子程序、中断服务等指令，都要使原来的 CS:IP 或 IP 寄存器的内容改变，使之指向下一条要执行指令的位置。程序中的分支、循环、中断等都会用到这些指令，这些指令可分为有条件（又可分为单条件和多条件）指令和无条件指令。其条件的产生是先执行运算指令，使状态标志发生变化，然后通过这些指令测试来控制程序转移。转移可在同一段中进行（仅改变 IP 内容），也可以在不同段中进行（CS 和 IP 都改变）。

⑥ 处理机控制类：提供程序控制 CPU 的各种功能，如使处理机暂停、等待、封锁总线等，还可以对 FR 寄存器中的一些标志进行置“1”或清“0”等操作。

由于 8086 指令系统较为复杂，为了能使读者尽快理解并掌握 8086 指令系统，我们结合 8086 寻址方式，给出一种快速掌握 8086 指令系统的学习方式，不仅可以有效掌握 8086 指令系统，也可以成为学习其他类型 CPU 指令系统的方法，如表 3-2 所示。r 代表寄存器，mem 代表内存，seg 代表段寄存器等。显然，只有 r、mem、seg 具有变量的属性。

表 3-2　指令符号

符　号	代表对象	说　明
a	AX	累加器
	AL	
r	AX，BX，CX，DX，SI，DI，BP，SP	16 位寄存器
	AL，AH，BL，BH，CL，CH，DL，CH	8 位寄存器
mem	[nn]	直接寻址
	[BX]，[BP]，[SI]，[DI]	寄存器间接寻址
	[BX+nn]，[BP+nn],	基址寻址
	[SI+nn]，[DI+nn],	变址寻址
	[BX+SI+nn]，[BX+DI+nn]，[BP+SI+nn]，[BP+DI+nn]	基址+变址寻址
seg	CS，DS，ES，SS	代表段寄存器
im	立即数	立即数寻址
oprd	目的操作数或源操作数	—

3.4.1　数据传输指令

数据传输指令负责把数据、地址等传送到寄存器、存储单元或外部设备中，以实现存储器和寄存器、寄存器和寄存器、AX 或 AL 寄存器与 I/O 端口之间的字节型或字型数据的传输。堆栈操作、标志操作等也属于这类指令。数据传输指令又可以分成 4 种：通用数据传输，累加器专用传输（输入/输出数据传输），目的地址传输和标志寄存器传输。

指令的共同特点如下：

❖ 除了 POPF 和 SAHF 指令，这类指令的操作结果不会影响 FR 寄存器中的标志。

❖ 指令中有两个操作数，即目的操作数和源操作数，其执行过程为：源操作数→目的操作数，当指令中仅列出一个操作数时，另一操作数为隐含。

1．通用数据传输指令

8086 CPU 有 4 个通用数据传输指令，允许以段寄存器作为操作数（XCHG 除外），这是其他类型指令所不能实现的。

（1）传送指令 MOV

指令格式为：

```
MOV     目的操作数，源操作数
```

利用表 3-2 中的符号，上述指令格式可以展开为 3 种传送 MOV 指令形式。

① MOV r, oprd

对于这种以寄存器 r 为目的操作数，指令中源操作数可以是寄存器 r、存储器 mem、段寄存器 seg 和立即数 im。根据上面通式，就可以写成多条 8086 体系的指令。比如：

```
MOV     CL, AH
MOV     AL, 12h
MOV     AX, [BX+1]
MOV     BX, DS
```

② MOV mem, oprd

对于这种以存储器 mem 为目的操作数，指令中源操作数可以是 r、seg 和 im。这里没有 mem。根据通式，可以写成许多 8086 体系的指令。比如：

```
MOV     [BX], 1234H
MOV     [SI+5], BL
MOV     [BX+SI+10],DS
```

③ MOV seg, oprd

对于这种以段寄存器 seg 为目的操作数，指令中源操作数可以是 r、mem。这里没有 im 和 seg。此时 SEG 不包括代码段段寄存器 CS。根据指令形式，可以写成许多 8086 CPU 的指令。比如：

```
MOV     DS, AX
MOV     ES, [SI+5]
```

上面的 3 种 MOV 传送指令形式如图 3-2 所示。

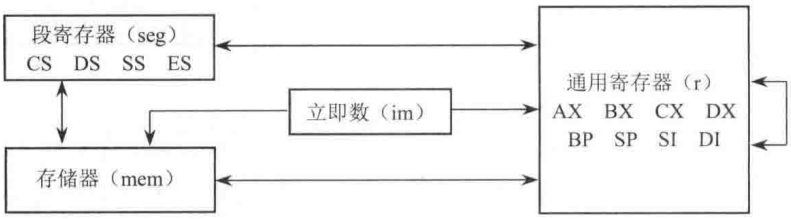


图 3-2 MOV 传送指令形式

所以，8086 CPU 的指令特点如下。

① 由于目的操作数是被赋值的，而可以被赋值的只有寄存器 r、存储器 mem 和段寄存器

seg，这样对目的操作数通过枚举变量类型的方法确定了三种传送指令的形式。请读者思考，为什么立即数 im 不能作为目的操作数？

② 对于目的操作数为 r 的 MOV 指令，源操作数的寻址方式最强，有 r、mem、seg 和 im 四种；对于目的操作数为 mem 的 MOV 指令，源操作数的寻址方式相对弱些，有 r、seg 和 im 三种；对于目的操作数为 seg 的 MOV 指令，源操作数的寻址方式最弱，有 r 和 mem 两种。这也提示我们，编程时尽量采用寄存器变量，出错相对较少，编程容易通过。

根据上面 MOV 指令的三个通式，MOV 指令要注意如下几点：

- ❖ 寄存器不包括 IP（根据 r 的定义）。
- ❖ 目的操作数不允许用段寄存器 CS。当指令涉及 CS 时要小心，因为执行一条改变 CS 寄存器内容的指令，会使一个新的段成为当前代码段，而此时 IP 仍然指示着以前代码段中的下一条指令。所以在改变 CS 值的同时，应该给 IP 一个相应的有效值，处理机才能正常工作（可以看 MOV　seg, oprd 的说明）。
- ❖ 目的操作数不允许是立即数，因为只有变量才能被赋值。
- ❖ 立即数不能直接送至段寄存器，如果需要，要通过其他寄存器转送（看 MOV　seg, oprd 的说明）。
- ❖ 源操作数和目的操作数的数据类型必须相同。比如，指令 MOV　AL, BX 是错误的，因为 8086 CPU 不存在 8 位变量和 16 位变量传送指令。又如，指令 MOV　[BX], 12H 也有问题的，因为立即数 12H 可能是 8 位数据传送指令或 16 位数据传送指令，而 MOV [BX], 12H 并没有对此说明。
- ❖ 不允许在两个存储单元中直接传送数据，如果需要，要通过寄存器转送（看 MOV mem, oprd 的说明）。
- ❖ 源操作数和目的操作数不可以同时为段寄存器（看 MOV　seg, oprd 的说明）。

为了说明是 8 位的，可用如下表达方式：

```
        MOV     [BX], byte ptr 12H
或      MOV     byte ptr [BX], 12H
```

上面的指令是 8 位的，byte ptr 说明其后面的对象是 8 位操作数

为了说明是 16 位的，可用如下表达方式：

```
        MOV     [BX],word ptr 12H
或      MOV     word ptr [BX], 12H
```

上面的指令是 16 位的，word ptr 说明其后面的对象是 16 位操作数，这时 12H 就是 0012H。

关于操作符 ptr 的功能将在第 4 章介绍。

（2）进栈指令 PUSH

指令格式为：

```
        PUSH    oprd
```

堆栈操作总是对 16 位的数据进行，指令中目的操作数隐含为堆栈。进栈操作把数据传输到以 SS 为段基址、SP 为偏移地址的栈中。其操作过程如下：

<1> SP 减 2，指示新栈顶。

<2> oprd 存入 SS:SP 所指示的栈顶，完成进栈操作。

这里 oprd 是源操作数，可以是各种类型的变量，即 r、mem、seg。比如：

```
PUSH    BX
PUSH    [BX]
PUSH    DS
```

（3）出栈指令 POP

指令格式为：

```
POP     oprd
```

其中，源操作数隐含为堆栈，出栈操作把以 SS 为段基址、SP 为偏移地址的栈顶内容传输到目的操作数中。操作过程如下：

<1> S 将 SS:SP 所指示的栈顶处的 2 字节的数据传输到目的操作数中。

<2> SP 加 2，指示当前栈顶位置，完成出栈操作。

这里 oprd 是目的操作数，它可以是各种类型的变量，即 r、mem、seg。注意，这里不能是 im，也不能是 CS（参看 MOV 指令的说明）。

根据上面的说明，我们很容易写出如下指令：

```
POP     AX
POP     [BX+SI+100H]
POP     ES
```

（4）交换指令 XCHG

指令格式为：

```
XCHG    r, oprd
```

其中，目的操作数和源操作数中的内容互换。

因为上述指令执行时数据互换，属于变量赋值，所以 oprd 为 r、mem。这里，操作数不能为立即数。

源操作数和目的操作数不能同时为存储单元，因为 MOV 指令作为最强寻址方式的数据传送指令，不能完成存储单元到存储单元的数据传送。作为比 MOV 指令寻址方式弱的 XCHG 指令也就不可能实现存储单元和存储单元的数据交换。另外，段寄存器不能作为操作数。

例如，将字型数据的偏移地址为 2040H 单元和 2050H 单元的内容交换。

方案一：用 MOV 指令

```
MOV     AX, [2040H]
MOV     BX, [2050H]
MOV     [2050H], AX
MOV     [2040H], BX
```

方案二：用 XCHG 指令

```
MOV     AX, [2040H]
XCHG    AX, [2050H]
MOV     [2040H], AX
```

方案三：用 PUSH、POP 指令

```
PUSH    [2040H]
PUSH    [2050H]
POP     [2040H]
POP     [2050H]
```

方案三巧妙地利用了堆栈区先进后出的工作方式。最后进入堆栈的数据是 2050H 单元的内容，然后利用指令 POP [2040H]，把此内容传给 2040H 单元。请读者体会这种方法。

2．累加器专用传输指令

8086 CPU 要与外部设备交换数据，必须通过累加器 AX（16 位数据）或 AL（8 位数据）传输给 I/O 端口，外设从输出端口取数据，完成数据输出。反之，外设将数据传输到 I/O 端口，CPU 从端口中将数据取到 AX 或 AL 中，完成数据输入。

硬件系统根据外设情况，设计端口地址为 16 位（外设较多）或 8 位（外设较少），而数据的大小也会根据具体情况设计，可以是 8 位 A/D 转换器转换得到 8 位数据，也可能是 16 位 A/D 转换器转换得到 16 位数据。8086 有相应的指令处理这些情况。

（1）输入指令 IN

指令格式为：

```
IN      a, oprd
```

IN 指令把 oprd 指示的端口内容传送到累加器 a。其目的操作数为累加器 a，代表 16 位的 AX 和 8 位的 AL，源操作数是 I/O 端口地址，即 n 或 DX。这里，n 是 8 位地址。

根据上面的说明，我们容易写出如下指令：

```
IN      AL, 80H
IN      AL, DX
IN      AX, 80H
IN      AX, DX
```

说明：以上 4 种指令格式分别是 8 位、16 位数据和 8 位、16 位端口地址的组合形式；指令中 n 表示为 8 位端口地址（00H～FFH），地址线高 8 位默认为 0；当端口地址为 16 位时，指令中采用寄存器间接寻址，应先将 16 位端口地址传到 DX，再通过 DX 中的地址在 IN 指令中间接寻址。

在 MOV 指令中，表示地址的一般都冠以“[ ]”，如[0080H]，但在 IN 指令中，n 为什么不用“[ ]”呢？因为在 MOV 指令中，0080H 可以作为立即数，也可以作为地址，为了区别这两点，0080 冠以“[ ]”，即[0080H]代表地址，区别立即数 0080H。而在 IN 指令中，其源操作数已被指定为地址，这样就不需要“[ ]”。这也可以解释为什么在 IN 指令中，DX 也不用“[ ]”。

比如，指令 IN AL, 80H，其目的操作数是寄存器寻址，源操作数寻址方式是直接寻址，就是从端口地址 80H 指定的 I/O 接口输入数据送给 AL。同理，对于 IN AL, DX 指令，其目的操作数是寄存器寻址，源操作数寻址方式是寄存器 DX 间接寻址，就是以 DX 内容作为端口地址，若 DX=0080H，则从端口地址 80H 指定的 I/O 接口输入数据送给 AL。

（2）输出指令 OUT

指令格式为：

```
OUT     oprd, a
```

OUT 指令把累加器 a 中的数据传送到 oprd 指示的端口。其目的操作数是 I/O 端口地址，即 n 或 DX。这里，n 是 8 位地址。

可见，无论是输入指令或输出指令，其指令中地址部分的寻址方式要么是直接寻址，要么是寄存器间接寻址。

（3）换码指令 XLAT

指令格式为：

```
XLAT
```

XLAT 指令没有明显的操作数，因此是隐含寻址。涉及的寄存器有 AL 和 BX。其功能是以 BX 内容加 AL 内容构成数据段中一个地址，然后从该地址中取数送给 AL。

所谓换码，是指令能够完成 1 字节的查表转换。

有时需要将一种代码转换成另一种代码，或在处理实际问题时采用映射关系来完成转换。前提是正确找到映射关系，利用存储器地址的连续性规律建立表格，然后用该指令完成转换。

地址	平方值
2000H	00
2001H	01
⋮	⋮
2009H	81

图 3-3　内存中的平方表

执行该指令之前，需先执行两条指令：

```
MOV     BX, 表的偏移首地址
MOV     AL, 被转换码
```

由于该指令不能单独执行，有时被称为复合指令。

例如，建立一个 0～9 的平方表，求 5 的平方值。将 0～9 的平方表建立在偏移地址为 2000H 的内存中，如图 3-3 所示。

完成求 5 的平方指令序列为：

```
MOV     BX, 2000H          ; 指向平方表的首地址
MOV     AL, 5
XLAT                       ; 执行换码指令，平方值放在 AL 中
```

以上例子完全是为说明 XLAT 指令而设计的，实际应用中求平方未必使用这种方法。

3．目标地址传输指令

目标地址传输指令是计算有效地址的指令，有效地址是指存储器地址。因此在这类指令中，源操作数必须是存储器。

（1）LEA（有效地址传输到寄存器）

指令格式为：

```
LEA     r, mem
```

指令的功能是取 mem 指示的地址（这里就是偏移地址）送寄存器 r。注意，r 是 16 位。

例如：

```
LEA     SI, [2040H]
```

指令执行后，SI 中的内容为 2040H。又如：

```
MOV     SI, [2040H]
```

指令执行后，SI 中有偏移地址为 2040H 单元中的内容，而不是 2040H 这个值。

请读者体会 LEA 和相关 MOV 指令的区别。

（2）LDS（装入一个新的物理地址）

指令格式为：

```
LDS     r, mem
```

LDS 指令是三个操作数指令，其中有两个目的操作数。其功能是：将源操作数指示的偏移

地址开始的4个连续字节中的内容传输到目的寄存器r和数据段寄存器DS中。注意，r是16位。

根据图3-4执行下列指令：

```
MOV     BX, 2080H   ; 用BX间接寻址
LDS     SI, [BX]        ; SI←[2080H], DS←[2082H]
MOV     AL, [SI]        ; AL=88H
```

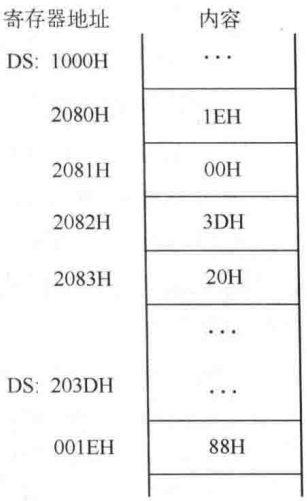


图3-4 LDS指令

（3）LES（装入一个新的物理地址）

除了段寄存器不同，LES指令其余与LDS指令类似。LES的段寄存器为附加数据段寄存器ES。

4．标志寄存器传送指令

FR寄存器是个比较特殊的寄存器，虽然是16位的，但8086 CPU中只定义了9位，定义的每位都有自己的含义，对于它的操作与其他寄存器不一样。其指令形式也有特点：操作数都是隐含的，其中8位的传输是对SF、ZF、AF、PF和CF的操作。

- LAHF：FR寄存器的低8位→AH。
- SAHF：AH→FR寄存器的低8位。
- PUSHF：FR寄存器推入堆栈中。
- POPF：从栈顶中弹出的内容存入FR寄存器中。

利用标志寄存器传送指令，特别是入栈、出栈指令，可以实现对FR寄存器的一些特殊操作要求。

【例3-1】 对FR寄存器中的TF标志位置1，其他位不变。

考虑到TF标志在FR寄存器的第9位，为了实现要求，设计如下指令序列：

```
PUSHF
POP     AX
OR      AX, 0200H
PUSH    AX
POPF
```

前两条指令通过堆栈把FR寄存器内容传递给累加器AX。第三条指令是一条“或”操作指令，其功能是把AX和立即数0200H相“或”。由于0200H中第9位为1，这样“或”的结果是AX的第9位为1，其他位不变。后两条指令把AX内容通过堆栈传递给FR。

3.4.2 算术运算指令

算术运算指令的共同点如下：① 运算指令影响状态标志；② 参加运算的数可以是无符号整型数、带符号整型数、压缩BCD数和非压缩BCD数；③ 乘法/除法指令中，乘数、被乘数以及除数、被除数、商和余数的存放位置有规定；④ 乘法和除法指令的书写形式有要求。

1．算术加法指令

（1）算术加法ADD

指令功能：目的操作数←目的操作数+源操作数。

指令格式：

```
ADD     目的操作数，源操作数
```

上述格式可以展开两种加法指令 ADD 指令形式。

① ADD r, oprd

对于这种以寄存器 r 为目的操作数，通式中的源操作数可以是 r、mem、im。

可以看出，与 MOV r, oprd 指令相比，ADD 指令寻址方式要弱一些，它的操作数没有段寄存器 seg。因此，段寄存器不仅不参与 ADD 指令的操作，也不参加任何其他任何算术逻辑指令的操作。

根据上面的说明，我们容易写出如下指令：

```
ADD     AX, BX
ADD     BX, [BP+SI+1000H]
ADD     AL, 12H
```

② ADD mem, oprd

对于这种以存储器 mem 为目的操作数是，通式中的源操作数可以是 r、im。比如：

```
ADD     WORD PTR [BX], 5B28H          ; WORD PTR 指明该存储器操作数是字型
```

ADD 指令说明：① 指令的目的操作数不能是立即寻址；② 两个操作数不能同时为存储器变量，与 MOV 指令的要求一致；③ 加法操作中产生的进位影响 CF 标志；④ 带符号操作数相加要考虑溢出。

（2）带进位算术加法 ADC

指令功能：目的操作数←目的操作数+源操作数+CF。

指令格式：

```
ADC     目的操作数，源操作数
```

上述格式可以展开两种传送指令 ADC 指令形式。

① ADC r, oprd

对于这种以寄存器 r 为目的操作数，指令中的源操作数可以是 r、mem、im。

② ADC mem, oprd

对于这种以存储器 mem 为目的操作数，指令中的源操作数可以是 r、mem。

可见，ADC 指令在寻址方式上与 ADD 完全一致。

ADC 指令说明：① 指令中有 3 个操作数，其中 CF 是本指令执行前的状态；② 需要完成多字节数（如 4 字节的 32 位数或更多字节）相加时使用该指令；③ 指令的目的操作数不能是立即寻址；④ 加法操作中产生的进位进入 CF 标志位；⑤ 带符号操作数相加要考虑溢出。

【例 3-2】 完成无符号数 5B68F271H 和 0AC6D5698H 加法操作。

由于操作数为 32 位而 8086 的寄存器只有 16 位，因此该操作要分两次进行，先对低 16 位做加法，再对高 16 位做加法并考虑进位。

```
MOV     AX, 0F271H              ; 加数的低 16 位
ADD     AX, 5698H               ; 与被加数的低 16 位相加，并影响 CF
MOV     DX, 5B68H               ; 加数的高 16 位
ADC     DX, 0AC6DH              ; 与被加数的高 16 位和 CF 相加，运算结果在 CF、DX 和 AX 中
```

（3）加 1 指令 INC

指令功能：目的操作数←目的操作数+1。

指令格式：

```
INC     oprd
```

根据此指令的功能，oprd 只能是变量，因此可选 r、mem。

INC 指令说明：立即数不是变量，因此操作数不能是立即寻址；该指令不影响 CF 标志；操作数为内存寻址时，因为无法确定其是 8 位或 16 位变量，需使用伪指令给予说明。例如：

```
INC     BYTE PTR [BX]                ; BYTE PTR用以指明该存储器操作数是字节型
INC     AX
```

显然，指令 INC　AX 可以由指令 ADD　AX, 1 代替。但注意，指令 INC　AX 不影响 CF 标志，而指令 ADD　AX, 1 影响 CF 标志。这是 INC 指令与 ADD 指令的一个主要区别。

（4）对压缩 BCD 数加法操作的结果进行校正 DAA

指令功能：对 AL 寄存器的内容进行十进制数调整。

指令格式：

```
DAA
```

DAA 指令说明：① 要求参加操作的数必须是 BCD 数；② 该指令用在压缩 BCD 数加法操作后，操作数隐含在 AL 中。

DDA 指令调整方法：若 AF 标志为 1，或 AL 寄存器的低 4 位超出 BCD 数的计数符号（即为 0AH～0FH），则 AL 寄存器的内容加 06H，且将 AF 置 1；若 CF 标志为 1，或 AL 寄存器的高 4 位超出 BCD 数的计数符号（即为 0AH～0FH），则 AL 寄存器的内容加 60H，且将 CF 置 1。例如：

```
MOV     AL, 85H
ADD     AL, 96H
DAA
```

在执行“ADD　AL, 96H”后，AL 的内容是 1BH，同时 CF 为 1，在执行 DAA 指令时，因为 AL 中的个位数是 B，同时 CF 为 1，所以要对 AL 进行加 66H 的修正，修改结果是 81H，同时 CF 中的内容不变，即为 1。

（5）对非压缩 BCD 数加法操作的结果进行校正 AAA

指令功能：对 AL 寄存器的内容进行十进制数调整。

指令格式：

```
AAA
```

AAA 指令说明：① 要求参加操作的数必须是非压缩 BCD 数；② 用在非压缩 BCD 数加法操作后，操作数隐含在 AL 中；③ 该调整指令使用 AH 寄存器，故应先将 AH 内容清零。

AAA 指令调整方法：<1> 若 AL 寄存器的低 4 位在 0～9 之间，且 AF 为 0，则跳过第<2>步，执行第<3>步；<2> 若 AL 寄存器的低 4 位在 0AH～0FH 之间或 AF 为 1，则 AL 寄存器的内容加 06H，同时 AH 寄存器内容加 1，且将 AF 置 1；<3> AL 寄存器的高 4 位清零；<4> AF 位的值送 CF。例如：

```
MOV     AX, 09               ; AH清零，AL中为加数
ADD     AL, 07
AAA                          ; 结果在AX中为0106
```

2．算术减法指令

（1）算术减法 SUB

指令功能：目的操作数←目的操作数–源操作数。

指令格式：

```
SUB     目的操作数，源操作数
```

上述格式可以展开两种 SUB 指令形式：

```
SUB     r, oprd                 ; oprd: r, mem, im
SUB     mem, oprd               ; oprd: r, mem
```

可见，SUB 指令在寻址方式上与 ADD 完全一致。说明，后面的其他逻辑指令的寻址方式与 ADD 也完全一致。

SUB 指令说明：① 指令的目的操作数不能是立即寻址；② 两个操作数不能同时为存储器变量；③ 减法操作中产生的借位进入 CF 标志；④ 无符号操作数相减，若 CF=1，则结果为补码；⑤ 带符号操作数相减要考虑溢出。

根据上面的说明，我们容易写出如下指令：

```
SUB     AX, BX
SUB     BYTE PTR [SI], 56H
```

（2）带进位算术减法 SBB

指令功能：目的操作数←目的操作数–源操作数–CF。

指令格式：

```
SBB     目的操作数，源操作数
```

上述格式可以展开两种 SBB 指令形式：

```
SBB     r, oprd                 ; oprd: r, mem, im
SBB     mem, oprd               ; oprd: r, mem
```

SBB 指令说明：① 指令中有 3 个操作数，其中 CF 是本指令执行前的借位；② 在需要完成多字节数（如 4 字节的 32 位数或更多字节）相减时使用；③ 目的操作数不能是立即寻址；④ 无符号操作数相减，若 CF=1，则结果为补码；⑤ 带符号操作数相减要考虑溢出。

【例 3-3】 完成无符号数 5B68F271H 和 0AC6D5698H 相减操作。

由于操作数为 32 位而 8086 的寄存器只有 16 位，因此该操作要分两次进行，先对低 16 位做减法，再对高 16 位做减法并考虑借位。

```
MOV     AX, 0F271H              ; 被减数的低 16 位
SUB     AX, 5698H               ; 与减数的低 16 位相减，并影响 CF
MOV     DX, 5B68H               ; 被减数的高 16 位
SBB     DX, 0AC6DH              ; 与减数的高 16 位和借位 CF 相减，结果在 CF、DX 和 AX 中
```

（3）减 1 指令 DEC

指令功能：目的操作数←目的操作数–1。

指令格式：

```
DEC     oprd                    ; oprd: r, mem
```

DEC 指令说明：① 操作数不能是立即寻址；② 不影响 CF 标志；③ 操作数为内存寻址时，需使用伪指令。例如：

```
DEC     CX
DEC     WORD PTR [BX]          ; PTR为宏汇编中的运算符，WORD用以指明该存储器操作数是字型
```

（4）对压缩 BCD 数减法操作的结果进行校正 DAS

指令功能：对 AL 寄存器中的内容进行十进制数调整。

指令格式：

```
DAS
```

DAS 指令说明：① 参加操作的数必须是压缩 BCD 数；② 用于压缩 BCD 数减法操作时，操作数隐含在 AL 中。

DAS 指令调整方法：若 AF 标志为 1，或 AL 寄存器的低 4 位超出 BCD 数的计数符号（即为 0AH～0FH），则 AL 寄存器的内容减 06H，且将 AF 置 1；若 CF 标志为 1，或 AL 寄存器的高 4 位超出 BCD 数的计数符号（即为 0AH～0FH），则 AL 寄存器的内容减 60H，且将 CF 置 1。例如：

```
MOV     AL, 86H
SUB     AL, 54 H
DAS
```

（5）对非压缩 BCD 数减法操作的结果进行校正 AAS

指令功能：对 AL 寄存器的内容进行十进制数调整。

指令格式：

```
AAS
```

AAS 指令说明：① 参加操作的数必须是非压缩 BCD 数；② 用于非压缩 BCD 数减法操作时，操作数隐含在 AL 中；③ 需用到 AH 寄存器，故应先将 AH 内容清零。

AAS 指令调整方法：<1> 若 AL 寄存器的低 4 位为 0～9，且 AF 为 0，则跳过第<2>步，执行第<3>步；<2> 若 AL 寄存器的低 4 位为 0AH～0FH 或 AF 为 1，则 AL 寄存器的内容减 06H，同时 AH 寄存器内容减 1，且将 AF 置 1；<3>AL 寄存器的高 4 位清零；<4>AF 位的值送 CF。例如：

```
MOV     AX, 09                 ; AH清零，AL中为非压缩BCD数
SUB     AL, 07
AAS
```

（6）比较指令 CMP

指令功能：完成两个操作数相减。

指令格式：

```
CMP     目的操作数, 源操作数
```

上述格式可以展开两种 CMP 指令形式：

```
CMP     r, oprd                ; oprd: r, mem, im
CMP     mem, oprd              ; oprd: r, mem
```

CMP 指令说明：① 执行“目的操作数–源操作数”，与 SUB 指令不同，不产生运算结果，仅影响标志；② 目的操作数不能是立即寻址；③ 目的操作数和源操作数不能同时为存储器操作数

（7）取补指令 NEG

指令功能：0–目的操作数。

指令格式：

```
NEG     oprd                    ; oprd: r, mem
```

NEG 指令说明：① 执行“0–oprd → oprd”，因此除了目的操作数为 0，CF 总是被置 1；② 操作数不能是立即寻址。

【例 3-4】 设计一程序段，把 DX、AX 组成的 32 位数取补，其中 DX 为高 16 位。

方案一：用减法指令。

```
MOV     BX, 0
SUB     BX, AX
MOV     AX, BX
MOV     BX, 0
SBB     BX, DX
MOV     DX, BX
```

方案二：用取补和减法指令。

```
NEG     DX                      ; 0-DX→DX
NEG     AX                      ; 0-AX→AX
SBB     DX, 0                   ; DX-0-CF→DX
```

注意，本例是说明取补指令的使用，并没有考虑 DX:AX 中 32 位数是正或是负，也就是说，只是对一个 32 位数的“绝对取补”。实际上，NEG 指令就是“绝对取补”的指令。

3．算术乘法指令

（1）无符号数乘法指令 MUL

指令功能：完成两个操作数相乘。

指令格式：

```
MUL     oprd                    ; oprd: r, mem
```

MUL 指令说明：① 8 位×8 位 →16 位，16 位×16 位 →32 位；② 乘数和被乘数都不能为立即寻址；③ 乘数或被乘数必须放在 AL 或 AX 中，在指令中则隐含；④ 16 位运算结果在 AX 中，32 位运算结果在 DX 和 AX 中。

【例 3-5】 完成 3EH×5DH 运算。

```
MOV     AL, 3EH
MOV     BL, 5DH
MUL     BL                      ; 执行 AL×BL→AX
```

又如：

```
MOV     AL, 3EH
MUL     5DH                     ; 语法错误
```

（2）带符号数乘法指令 IMUL

指令功能：完成两个操作数相乘。

指令格式：

```
IMUL    oprd                    ; oprd: r, mem
```

IMUL 指令说明：① 8 位×8 位 →16 位，16 位×16 位 →32 位；② 乘数和被乘数都不能为立即寻址；③ 乘数或被乘数必须放在 AL 或 AX 中，在指令中则隐含；④ 16 位运算结果在

AX 中，32 位运算结果在 DX 和 AX 中；⑤ 有符号数在计算机中是其补码，且符号位也参加运算，此时用 MUL 指令就得不到正确结果，IMUL 指令则会将符号部分和数值部分分别处理。

（3）非压缩 BCD 数乘法操作结果校正 AAM

指令功能：完成两个非压缩 BCD 数乘法结果的十进制数调整。

指令格式：

```
AAM
```

AAM 指令说明：① 参加操作的数必须是非压缩 BCD 数，用于 MUL 指令后；② 指令调整方法为，两个非压缩 BCD 数相乘的结果为 0～81，不会到十进制数的百位，所以调整方法是将 AL 寄存器的内容除以 0AH，商放在 AH 寄存器（与除法指令不同）中，表示转换的十位数，余数放在 AL 寄存器（与除法指令不同）中，表示转换的个位数。

【例 3-6】 完成 09H×07H 运算。

```
MOV     AL, 09H
MOV     DL, 07H
MUL     DL                      ; AX=003FH
AAM                             ; AX=0603H
```

AAM 指令说明：该指令段中若不执行 AAM 指令，则 AX 中的结果不是非压缩 BCD 数，结果是十六进制数（3FH=63）。

4．算术除法指令

（1）无符号数除法指令 DIV

指令功能：完成两个操作数相除。

指令格式：

```
DIV     oprd                    ; oprd: r, mem
```

DIV 指令说明：① 用 16 位除 8 位，32 位除 16 位的格式，被除数不够 16 位或 32 位，则需扩展；② 被除数、除数都不能为立即寻址，除数必须是寄存器或存储器寻址；③ 被除数必须放在 AX 或 DX:AX 中，在指令中则隐含；④ 16 位运算的商放在 AL 中，余数放在 AH 中；⑤ 32 位运算的商放在 AX 中，余数放在 DX 中。

【例 3-7】 完成无符号数 3E2CH÷5BH 运算。

```
MOV     AX, 3E2CH
MOV     BL, 5BH
DIV     BL
```

【例 3-8】 完成无符号数 8A73H÷185BH 运算。

```
MOV     AX, 8A73H
MOV     DX, 0
MOV     CX, 185BH
DIV     CX
```

（2）带符号数除法指令 IDIV

指令功能：完成两个操作数相除。

指令格式：

```
IDIV    oprd            ; oprd: r, mem
```

IDIV 指令说明：① 用 16 位除 8 位，32 位除 16 位的格式，若被除数不够 16 位或 32 位，则需用扩展指令将其扩展；② 被除数、除数都不能为立即寻址，除数必须是寄存器或存储器寻址；③ 被除数必须放在 AX 或 DX:AX 中，在指令中则隐含；④ 16 位运算的商放在 AL 中，余数放在 AH 中；⑤ 32 位运算的商放在 AX 中，余数放在 DX 中。

（3）带符号数字节扩展指令 CBW

指令功能：将 AL 中的 8 位数扩展为 16 位数。

指令格式：

```
CBW
```

CBW 指令说明：① 在带符号除法指令中，被除数要扩展成 16 位时使用该指令；② 其扩展规则是根据 AL 寄存器的最高位，若 D7 为 1，则 AH 置 FFH，若 D7 为 0，则 AH 置 0。

【例 3-9】 在偏移地址为 2340H 和 2341H 内存单元中存有两个带符号字节数，求这两个数相除的结果，并将结果分别存入存储单元 2342H 和 2343H 中。

```
MOV     AL, [2340H]
CBW
IDIV    BYTE PTR [2341H]          ; 符号 BYTE PTR 说明内存单元[2341H]中的数是字节型的
MOV     [2342H], AL
MOV     [2343H], AH
```

（4）带符号数字扩展指令 CWD

指令功能：将 AX 中的 16 位数扩展为 32 位数。

指令格式：

```
CWD
```

CWD 指令说明：① 被除数要扩展成 32 位时使用该指令；② 扩展规则是根据 AX 寄存器的最高位，若 D_{15} 位为 1，则 DX 置 FFFFH，否则 DX 置 0。

阅读下列程序段，指出其执行功能和执行后结果。

```
MOV     AX, 9EB2H
CWD
XOR     AX, DX
SUB     AX, DX
```

说明：

① 第二条指令是该程序段中的关键，将 AX 中的字型数扩展为 32 位，扩展到 DX 的内容与 AX 的 D_{15} 位有关，DX 的内容为 FFFFH。第三条指令是对原 AX 内容求反，因为 AX 中的每一位都与“1”进行异或操作。第四条指令是对 AX 求反后加 1（实为减去 1）。

② 若 AX 的 D_{15} 位的内容为 0，则 DX 为 0。此时，执行第三条指令原 AX 内容不变，因为 AX 中的每一位都与“0”进行异或操作。第四条指令是 AX 内容减 0。

综上分析，该程序段执行对带符号数求绝对值，结果是 694EH。

（5）非压缩 BCD 数除法校正 AAD

指令功能：将两个非压缩 BCD 数除法操作调整为二进制数除法操作。

指令格式：

```
AAD
```

AAD 指令说明：① 要满足 16 位除 8 位的除法操作要求，即非压缩 BCD 数须放在 AX 寄

存器中；② 用在 DIV 指令之前，先调整，后做除法操作。

【例 3-10】 完成非压缩 BCD 数 0304 除 5 的运算，结果放在 AL、AH 中。

```
MOV     AX, 0304H
MOV     BL, 05
AAD                         ; AX=0022H
DIV     BL                  ; 商(AL)=06，余数(AH)=04
```

本例实质是 34 除 5，AAD 指令将非压缩的 BCD 数 0304H 转换成二进制数，然后用二进制数除法规则进行除法操作。

3.4.3 位操作指令

位操作指令的共同点如下：

- ❖ 可以按二进制位进行操作。
- ❖ 逻辑运算指令按逻辑门电路的运算规则。
- ❖ 逻辑移位指令有左移和右移，移出的位都进入 CF 标志。
- ❖ 因移空位的补充方式不同，有多种指令形式。
- ❖ 逻辑移位指令中，移动超过 1 次，则用 CL 寄存器作为计数器。
- ❖ 执行逻辑（不包括逻辑移位）操作指令，CF 均被清零。

1．逻辑运算指令

（1）逻辑非操作指令 NOT

指令功能：将 8 位、16 位寄存器或存储器内容求反。

指令格式：

```
NOT     oprd                ; oprd: r, mem
```

NOT 指令说明：① 目的操作数不能为立即数寻址；② 对 8 位或 16 位一次性全部取反。

例如：

```
MOV     AX, 1234H
NOT     AX
```

指令执行后，AX 的内容为 0EDCBH。

（2）逻辑与操作指令 AND

指令功能：将 8 位、16 位寄存器或存储器内容和源操作数“与”。

指令格式：

```
AND     目的操作数, 源操作数
```

上述格式可以展开二种 AND 指令形式：

```
AND     r, oprd             ; oped: r, mem, im
AND     mem, oprd           ; oprd: r, mem
```

AND 指令说明：① 目的操作数不能为立即数寻址；② 可由源操作数控制，对 8 位或 16 位数的某些位进行屏蔽或保留；③ 目的操作数和源操作数不能同时为存储器操作数。

例如，将 AX 寄存器中的 D_1、D_5、D_6、D_{11} 和 D_{15} 位保留，其余位清零。

```
AND     AX, 1000100001100010B
```

（3）逻辑或操作指令 OR

指令功能：将 8 位、16 位寄存器或存储器内容和源操作数“或”。

指令格式：

```
OR      目的操作数, 源操作数
```

上述格式可以展开两种 OR 指令形式：

```
OR      r, oprd                 ; oped: r, mem, im
OR      mem, oprd               ; oprd: r, mem
```

OR 指令说明：① 目的操作数不能为立即数寻址；② 可由源操作数控制，对 8 位或 16 位数的某些位进行置 1 或保留。③ 目的操作数和源操作数不能同时为存储器操作数。

例如，将 DX 寄存器中的低 8 位置 1，其余位不变。

```
OR      DX, 00FFH
```

（4）逻辑异或操作指令 XOR

指令功能：将 8 位、16 位寄存器或存储器内容和源操作数“异或”。

指令格式：

```
XOR     目的操作数, 源操作数
```

上述格式可以展开两种 XOR 指令形式：

```
XOR     r, oprd                 ; oprd: r, mem, im
XOR     mem, oprd               ; oprd: r, mem
```

XOR 指令说明：① 目的操作数不能为立即数寻址；可由源操作数控制，对 8 位或 16 位数的某些位进行求反或保留；② 逻辑异或操作又称为“模 2 加”，即以 2 为模的加法操作；③ 某位与 1 做“异或”，则求反；与 0 做“异或”，则不变；④ 操作数和源操作数不能同时为存储器操作数。

例如，将 AX 寄存器中的 D_1、D_5、D_6、D_{11} 和 D_{15} 位求反，其余位保留。

```
XOR     AX, 1000100001100010
```

这里对上面三个逻辑指令的特点做个总结：

- AND 指令可以使操作数中指定位为 0。
- OR 指令可以使操作数中指定位为 1。
- XOR 指令可以使操作数中指定位取反。

比如，将 AL 中的第 5 位分别取 0、取 1 和取反，而 AL 其他位不变的指令分别如下：

```
AND     AL, 11011111B           ; AL.5←0
OR      AL, 00100000B           ; AL.5←1
XOR     AL, 00100000B           ; AL.5取反
```

（5）测试指令 TEST

指令功能：将 8 位、16 位寄存器或存储器内容和源操作数“与”。

指令格式：

```
TEST    目的操作数, 源操作数
```

上述格式可以展开两种 TEST 指令形式：

```
TEST    r, oprd                 ; oprd: r, mem, im
TEST    mem, oprd               ; oprd: r, mem
```

说明：① 与 AND 指令的寻址方式和运算规则相同，但 TEST 指令不产生运算结果，仅影响状态标志；② 常用于对某位是“1”或“0”的检测。

例如，判断 AX 寄存器中的 D_1 位是否为“1”。

```
TEST    AX, 0002H
```

执行 TEST 指令后，检测 ZF 标志，若 ZF=0，则 D_1 位为 1；否则，D_1 位为 0。

2. 逻辑移位指令

（1）逻辑左移指令 SHL

指令功能：将 8 位、16 位寄存器或存储器内容左移，移空的位补 0。

指令格式：

```
SHL     oprd, 1                 ; oprd: r, mem
SHL     oprd, CL                ; oprd: r, mem
```

SHL 指令说明：① 移动 1 位时，源操作数为 1；移动超过 1 位时，用 CL 寄存器控制移动次数；② 移动 1 位，相当于原数据乘 2（在无进位的情况下）。

例如：

```
MOV     AL, 42H
SHL     AL, 1
```

指令执行后，寄存器 AL 内容为 84H，CF=0。

又如：

```
MOV     SI, 2B4CH
MOV     CL, 3
SHL     SI, CL
```

指令序列执行后，寄存器 SI 内容为 5A60H，CF=1。

（2）算术左移指令 SAL

指令功能：将 8 位、16 位寄存器或存储器内容左移，移空的位补 0。

指令格式：

```
SAL     oprd, 1                 ; oprd: r, mem
SAL     oprd, CL                ; oprd: r, mem
```

SAL 指令说明：① 移动 1 位时，源操作数为 1；移动超过 1 位时，用 CL 寄存器控制移动次数；算术左移指令移动 1 位，则原数据乘 2（在无进位的情况下）；② 与逻辑左移指令功能相同。

例如：

```
MOV     DL, 0A5H
SAL     DL, 1
```

指令执行后，寄存器 DL 中的内容为 4AH，CF=1。

又如：

```
MOV     BX, 6E8CH
MOV     CL, 2
SAL     BX, CL
```

指令序列执行后，寄存器 BX 中的内容为 0BA30H，CF=1。

（3）逻辑右移指令 SHR

指令功能：将 8 位、16 位寄存器或存储器内容右移，移空的位补 0。

指令格式：

```
SHR     oped, 1                         ; oprd: r, mem
SHR     oped, CL                        ; oprd: r, mem
```

SHR 指令说明：① 移动 1 位时，源操作数为 1；移动超过 1 位时，用 CL 寄存器控制移动次数；② 移动 1 位，相当于原数据除 2（与除法指令有差别）。

例如：

```
MOV     AL, 42H
SHR     AL, 1
```

指令执行后，寄存器 AL 中的内容为 21H，CF=0。

又如：

```
MOV     SI, 2B4CH
MOV     CL, 3
SHR     SI, CL
```

指令序列执行后，寄存器 SI 中的内容为 0569H，CF=1。

（4）算术右移指令 SAR

指令功能：将 8 位、16 位寄存器或存储器内容右移，移空的位由最高位补充。

指令格式：

```
SAR     oprd, 1                         ; oprd: r, mem
SAR     oprd, CL                        ; oprd: r, mem
```

SAR 指令说明：① 移动 1 位时，源操作数为 1；移动超过 1 位时，用 CL 寄存器控制移动次数；② 算术右移指令实质上补充的是符号位。

例如：

```
MOV     DL, 0A5H
SAR     DL, 1
```

这两条指令执行后，寄存器 DL 中的内容为 0D2H，CF=1。

又如：

```
MOV     BX, 6E8CH
MOV     CL, 2
SAR     BX, CL
```

指令序列执行后，寄存器 BX 中的内容为 1BA3H，CF=0。

（5）不带进位循环左移指令 ROL

指令功能：将 8 位、16 位寄存器或存储器内容左移，移空的位由移出位补充。

指令格式：

```
ROL     oprd, 1                         ; oprd: r, mem
ROL     oprd, CL                        ; oprd: r, mem
```

ROL 指令说明：移动 1 位时，源操作数为 1；移动超过 1 位时，则用 CL 寄存器控制移动次数。

例如：

```
MOV     BX, 6E8CH
MOV     CL, 2
ROL     BX, CL
```

指令执行序列后，寄存器 BX 中的内容为 0BA31H，CF=1。

（6）不带进位循环右移指令 ROR

指令功能：将 8 位、16 位寄存器或存储器内容右移，移空的位由移出位补充。

指令格式：

```
ROR     oprd, 1                         ; oprd: r, mem
ROR     oprd, CL                        ; oprd: r, mem
```

ROR 指令说明：移动 1 位时，源操作数为 1；移动超过 1 位时，则用 CL 寄存器控制移动次数。

例如：

```
MOV     AX, 8C36H
MOV     CL, 5
ROR     AX, CL
```

指令执行后，寄存器 AX 中的内容为 0B461H，CF=1。

（7）带进位循环左移指令 RCL

指令功能：将 8 位、16 位寄存器或存储器内容左移，移空的位由 CF 位补充。

指令格式：

```
RCL     oprd, 1                         ; oprd: r, mem
RCL     oprd, CL                        ; oprd: r, mem
```

RCL 指令说明：① 移动 1 位时，源操作数为 1；移动超过 1 位时，则用 CL 寄存器控制移动次数；② 执行前，若 CF 没有置“1”或清“0”，则第一次移位时，移空位由 CF 中的随机数补充。

例如：

```
XOR     AX, AX
MOV     AX, 8C36H
MOV     CL, 2
RCL     AX, CL
```

RCL 指令执行前，由 XOR 指令将 CF 清“0”，指令序列执行后，寄存器 AX 的内容为 30D9H，CF=0。

（8）带进位循环右移指令 RCR

指令功能：将 8 位、16 位寄存器或存储器内容右移，移空的位由 CF 位补充。

指令格式：

```
RCR     oprd, 1                         ; oprd: r, mem
RCR     oprd, CL                        ; oprd: r, mem
```

RCL 指令说明：① 移动 1 位时，源操作数为 1；移动超过 1 位时，则用 CL 寄存器控制移动次数；② 执行前，若 CF 没有置“1”或清“0”，则第一次移位时，移空位由 CF 中的随机数补充。

例如：

```
MOV     AX, 0E3BH
MOV     CL, 3
AND     AX, AX
RCR     AX, CL
```

RCR 指令执行前，由 AND 指令将 CF 清“0”，执行后，AX 中的内容为 0C1C7H，CF=0。

利用带进位的循环移位指令，可以实现多字节的数的移位。比如，把 DX、AX 寄存器中的 32 位二进制数（DX 为高 16 位）乘 2，结果再送 DX、AX，相关程序段如下：

```
MOV     AX, 8421H
MOV     DX, 0421H
SHL     AX, 1
RCL     DX, 1
```

上面程序执行后，DX:AX=08430842H。【思考】 如果把 DX、AX 寄存器中的 32 位二进制数（DX 为高 16 位）除 2，结果再送 DX、AX，如何用移位指令实现？

3.4.4 串处理指令

串处理指令是针对存储器的操作，其共同点如下：

① 指令有特殊的寻址方式，规定源操作数的逻辑地址由 DS:SI 给出，目的操作数的逻辑地址由 ES:DI 给出。

② 由于存储单元有字型数据和字节型数据，指令的助记符则有 B 或 W 之分。

③ 使用这类指令时，存储单元的地址指针是自动移动的，由 DF 标志控制指针的移动方向。DF=0，地址往增大方向移动；DF=1，地址往减小方向移动。

④ 串的长度由 CX 给定。

⑤ 在这类指令前面一般可以使用指令前缀。

⑥ 这类指令后不带操作数，操作数隐含给定。

串处理指令中需要注意的问题：

① 当串指令在同段之间执行传送或比较等操作时，应将 DS 和 ES 取同样的值。

② 当使用指令前缀进行重复操作时，CX 的值是指数据个数，不是存储单元的字节数。

③ 当使用 DF=1，即地址指针为减小方向移动时，源操作数和目的操作数的初始地址要正确设定。

1．串传输指令 MOVSB 或 MOVSW

指令功能：目的操作数←源操作数。

指令格式：

```
MOVSB
MOVSW
```

说明：① 源操作数地址由 DS:SI 指定，目的操作数由 ES:DI 指定；② 指令执行时，由 DF 标志控制 SI 和 DI 是增大还是减小；③ 由指令中的 B 和 W 控制 SI 和 DI 是加 1/加 2，还是减 1/减 2；④ 指令执行一次，CX 寄存器的内容不改变。

例如，将偏移地址为 BUFF1 的内存区中的 100 个字型数据，传输到偏移地址为 BUFF2 的

内存区。

```
        LEA     SI, BUFF1           ; 源操作数地址指针
        LEA     DI, BUFF2           ; 目的操作数地址指针
        CLD                         ; DF=0
        MOV     CX, 100             ; 数据区长度
        MOVSW
```

说明：① 在段定义中分配段地址，通过后面内容可知 DS 和 ES 的赋值过程；② MOVSW 指令仅执行一次，因为 CX 寄存器的内容不被操作，要连续运行需加循环控制指令或指令前缀。

下面是加循环控制指令的数据块搬移：

```
        LEA     SI, BUFF1           ; 源操作数地址指针
        LEA     DI, BUFF2           ; 目的操作数地址指针
        CLD                         ; DF=0
        MOV     CX, 100             ; 数据区长度
AA1:    MOVSW
        LOOP    AA1                 ; CX 减 1，CX 不为 0；循环执行 MOVSW 指令
```

2．串比较指令 CMPSB 或 CMPSW

指令功能：目的操作数–源操作数。

指令格式：

```
        CMPSB
        CMPSW
```

说明：① 完成内存中两串数据对应元素的减法操作，但不产生运算结果，仅影响状态标志；② 其余与串传送指令相同；③ 多与其他指令或指令前缀配合。

3．串搜索指令 SCASB 或 SCASW

指令功能：AL/AX–目的操作数。

指令格式：

```
        SCASB
        SCASW
```

说明：① 执行前，将需检索的数据存入 AL 或 AX 寄存器；② 指令完成内存数据串中每个元素的搜索操作，即 AL/AX–串元素；③ 多与其他指令或指令前缀配合；④ 其余与串传输指令相同。

4．串装入指令 LDSB 或 LDSW

指令功能：AL/AX←源操作数。

指令格式：

```
        LDSB
        LDSW
```

说明：① 每执行一次，将存储单元中的内容写入累加器；② 多与其他指令配合，但不能有指令前缀；③ 在高级语言的循环中与其他指令配合，可完成复杂字符串的处理；④ 指令的执行不影响标志位。

5．串存储指令 STOSB 或 STOSW

指令功能：目的操作数←AL/AX。

指令格式：

```
STOSB
STOSW
```

说明：① 每执行一次，将累加器的内容写入存储单元中；② 可以有指令前缀；③ 执行不影响标志位。

6．指令前缀

由于在串处理指令中，串的长度（即数据个数）存入 CX 寄存器，而指令执行时，并不对 CX 进行操作。重复执行这些指令时，需要 CX 中的值来控制循环的结束。指令前缀的作用是控制串处理指令循环执行，每执行一次，CX 内容减 1，直到 CX 内容为 0。带 ZF 标志的重复前缀，执行时除考虑 CX 外，还要对 ZF 标志进行判别，满足条件才能重复执行串操作。

如果在执行串处理指令的过程中，有一个外部中断进入，在完成中断处理后，将返回继续执行串操作。因为处理机在中断时只能“记住”一个前缀，如果串处理指令中又有段超越前缀，那么中断执行过程可能会不正常。

（1）REP

REP 前缀是指重复，每执行一次，CX 内容减 1，直到 CX 内容为 0，才退出串处理过程，一般与 MOVS 指令和 STOS 指令配合使用。

例如，将偏移地址为 BUFF1 的内存区中的 100 个字型数据，传送到偏移地址为 BUFF2 的内存区。

```
LEA     SI, BUFF1                ; 源操作数地址指针
LEA     DI, BUFF2                ; 目的操作数地址指针
CLD                              ; DF=0
MOV     CX, 100                  ; 数据区长度
REP     MOVSW
```

（2）REPZ/REPE

REPZ/REPE 前缀往往与 CMPS 指令配合使用。在两个串数据进行比较时，其比较操作是否继续进行，可以由 CX 内容和 ZF 标志两个条件控制。每执行一次 CMPS 指令，CX 内容减 1，若 CX≠0 且 ZF=1，比较操作继续进行；若 CX≠0 且 ZF=0，比较操作停止；若 CX=0，比较操作停止。

REPZ/REPE 前缀适合用于判别两个串数据是否全等，在串中找到不相等的元素则停止，否则比较操作继续进行，直到全部数据操作一遍。

（3）REPNZ/REPNE

REPNZ/REPNE 前缀往往与 SCAS 指令配合使用。对某个串数据进行搜索时，其操作是否继续进行，可以由 CX 内容和 ZF 标志两个条件控制。每执行一次 SCAS 指令，CX 内容减 1，若 CX≠0 且 ZF=0，搜索操作继续进行；若 CX≠0 且 ZF=1，搜索操作停止；若 CX=0，搜索操作停止。

REPNZ/REPNE 前缀适合用于在一个串数据中找出期望的元素，搜索到目标数据则停止，否则搜索操作继续进行，直到全部数据查找一遍。

3.4.5 程序控制转移指令

一般情况下，指令是逐条顺序执行的，但实际上程序不可能全部按顺序执行。实际中要解决的问题多种多样，稍微复杂一点的问题都涉及转移、分支、重复等操作，这类指令能用来改变程序的流程。

寄存器 CS 和 IP 中的逻辑地址指示下一条要执行的指令所在存储单元的位置，所以程序控制转移指令会改变 IP 和/或 CS 的值。

这类指令分为无条件转移指令、有条件转移指令、循环控制转移指令、子程序调用指令和中断指令。不同的指令对转移距离的大小（转移相对字节数）有不同的限制。

条件转移指令中的条件是由影响标志状态的运算指令产生的，所以对参加运算的带符号数和不带符号数之条件判别，需分别对待。标志寄存器 FR 中的 AF 标志不在条件转移指令中作为条件使用，仅在前面介绍的调整指令中自动检测，用以完成必要的调整操作。

1．无条件转移指令

无条件转移指令必须指定转移的目标地址（或称为转向地址），将程序无条件地转移到目标地址，执行从该地址开始的指令。因转移距离的不同，这种转移指令分为段内转移和段间转移。段内转移即在同一段范围内转移，只改变 IP 的值；段间转移，则 IP 和 CS 的值都会改变。

由于指令中给出转移目标地址的方式不同，因此无条件转移指令有 4 种格式。

（1）段内直接转移指令

在一个段内进行直接无条件转移，是在指令中给出一个相对位移量。位移量是相对于 IP 寄存器来计算的，即有效转移地址是在 IP 当前的内容上加上一个 8 位或 16 位的位移量，所以也称为相对寻址。其位移量是带符号的数，这就使转移可在向前或向后的方向进行。

① 当位移量为 16 位带符号数时，允许在±32 KB 的范围内寻找目标地址，称为段内直接近转移。其指令格式为：

```
JMP     NEAR PTR 目标地址
```

② 当位移量为 8 位带符号数时，允许在±127 字节的范围内寻找目标地址，称为段内直接短转移。其指令格式为：

```
JMP     SHORT 目标地址
```

（2）段内间接转移指令

在 JMP 指令中，间接给出转移目标地址，即由一个 16 位寄存器或由存储单元寻址目标地址。其指令格式为：

```
JMP     CX
JMP     WORD PTR [BX]
```

（3）段间直接转移指令

段间转移意味着寄存器 CS 和 IP 的值都要改变，指令中直接采用汇编中的符号地址作为直接目标地址。其指令格式为：

```
JMP     FAR PTR 目标地址
```

（4）段间间接转移指令

段间转移意味着寄存器 CS 和 IP 的值都要改变，所以间接给出目标地址，只能由存储单

元寻址。也就是说，目标地址存放在连续的 4 个存储单元中，低字节两个单元的内容代替 IP，高字节两个单元的内容代替 CS。其指令格式为：

```
JMP     DWORD PTR [BX][SI]
```

注：有关 PTR 运算符将在第 4 章中介绍。

2．条件转移指令

条件转移指令是根据执行该指令前标志位的状态而决定是否发生控制转移的指令。每条指令测试不同的标志位组合，看是否满足条件。若条件不满足，则继续执行跟在条件转移指令之后的指令；若条件满足，则将程序控制转移到该指令给出的目标地址，去执行那里的程序。所有的条件转移指令都应为短距离的转移，即其目标地址必须在当前指令段内，且与下一条指令的第 1 字节的距离为–128～+127 字节，IP←IP+符号扩展到 16 位后的 8 位位移量。由于转移操作是通过把目标地址的相对位移量加到指令指针上实现的，因此所有的条件转移指令的目标地址都是相对于该指令本身而言的。条件转移指令适用于与绝对地址位置无关的程序。

所有条件转移指令之所以采用短距离的相对转移形式，是为了控制指令长度和提高程序执行速度。当需要往一个较远的位置进行条件转移时，可以先用条件转移指令转到附近一个单元，然后在此单元加一条无条件转移指令，使之转移到需要的目标地址。在实际应用中，多数情况下需要的是根据条件转移到附近的区域。

在条件转移指令中，有相当一部分指令是在比较完两个数的大小之后而决定是否转移的，但对于具体的某两个被比较的二进制数，将它们看成带符号数或者无符号数，比较后，则会得到不同的结果。为了做出正确判断，8086 指令系统分别为无符号数的比较和带符号数的比较提供了条件转移指令。对于无符号数的比较，判断结果时，用“高于”或“低于”的概念作为判断依据进行条件转移；对于带符号数的比较，判断结果时，用“大于”或“小于”的概念作为判断依据进行条件转移。

8086 的条件转移指令中，大部分指令可以用两种不同的助记符来表示。比如，对带符号数，“一个数大于另一个数”和“一个数不小于也不等于另一个数”的结论是等同的，因此使用的两个助记符 JG 和 JNLE 是等同的。

条件转移指令本身不影响标志状态，它只是根据该指令执行前标志状态情况来决定是否转移。8086 的标志状态作为转移的条件有单条件和多条件（多个状态标志组合的条件）的转移指令，某些问题只需要一个状态标志就可以决定程序流程是否改变，但有些问题必须考虑多个条件才能准确判断是否转移。

条件转移指令的格式都是在助记符后面直接跟一个转移目标地址。

（1）单条件转移指令

8086 的标志寄存器 FR 中有 6 个标志是状态，反映运算结果的情况。除了 AF 半进位标志，其余 5 个标志可以反映 10 种不同条件，故单条件转移指令有 10 种。

- JC：CF 标志为 1，则转移。
- JNC：CF 标志为 0，则转移。
- JE/JZ：ZF 标志为 1，则转移。
- JNE/JNZ：ZF 标志为 0，则转移。
- JS：SF 标志为 1，则转移。

❖ JNS：SF 标志为 0，则转移。

❖ JO：OF 标志为 1，则转移。

❖ JNO：OF 标志为 0，则转移。

❖ JP/JPE：PF 标志为 1，则转移。

❖ JNP/JPO：PF 标志为 0，则转移。

（2）无符号数的条件转移指令

用于无符号数的条件转移指令如下。

❖ JA/JNBE：高于/不低于等于转移，CF∨ZF=0。

❖ JNA/JBE：不高于/低于等于转移，CF∨ZF=1。

❖ JB/JNAE：低于/不高于等于转移，CF=1。

❖ JNB/JAE：不低于/高于等于转移，CF=0。

（3）带符号数的条件转移指令

用于带符号数的条件转移指令如下。

❖ JG/JNLE：大于/不小于等于转移，(SF∨OF)∨ZF=0。

❖ JGE/JNL：大于等于/不小于转移，(SF∨OF)=0。

❖ JL/JNGE：小于/不大于等于转移，(SF∨OF)=1。

❖ JLE/JNG：小于等于/不大于转移，(SF∨OF)∨ZF=1。

用带符号数和无符号数的条件转移指令时，必须严格加以区别，否则会得到错误的结果。

3．循环控制指令

在设计循环程序时，可用控制指令来控制循环是否继续，下面介绍的循环控制指令可以用来管理软件循环的重复过程。这些指令的特点是用 CX 寄存器作为循环控制计数器，与条件转移指令一样，都是短距离转移指令，循环控制指令所给出的目标地址与下一条指令首地址的距离只能在–128～+127 字节的范围内。

（1）LOOP 指令

指令格式：

```
LOOP    目标地址
```

执行该指令，CX–1，若 CX≠0，转移到目标地址，即 IP←IP+8 位位移量（带符号扩展到 16 位）。由于指令自动对 CX 寄存器做减 1 操作，故使用 LOOP 指令前，需将循环操作次数值赋给 CX 寄存器。

（2）LOOPZ/LOOPE 指令

除了与 LOOP 指令一样，LOOPZ/LOOPE 指令还执行 CX–1，判断 CX 内容是否为 0 外，还有一个条件为 ZF，即程序循环部分是否继续执行，由 CX≠0 和 ZF 两个条件决定。当 CX≠0 且 ZF=1 时，循环操作继续，其转移目标地址为 IP←IP+8 位位移量（带符号扩展到 16 位）；若 CX=0 或 ZF=0，则退出循环，执行下一条指令。该指令中的 CX 减至 0 时，不影响 ZF 标志，ZF 的状态是由前一条指令执行结果影响的。

（3）LOOPNZ/LOOPNE 指令

这条指令与 LOOPZ/LOOPE 指令相对应，程序循环部分是否继续执行，由 CX 和 ZF 两个条件决定。当 CX≠0 且 ZF=0 时，循环继续执行，其转移目标地址为 IP←IP+8 位位移量（带符

号扩展到16位)；若CX=0或ZF=1，则退出循环，执行下一条指令。

(4) JCXZ指令

从CX寄存器的命名来看，与单词“Counter”有关联，在循环控制指令中，CX寄存器用来设置计数值；在移位指令中，CL寄存器用来控制移动次数。

JCXZ指令在CX=0时，将控制转移到目标地址，即IP←IP+8位位移量(带符号扩展到16位)。在程序中，JCXZ指令经常被安排在循环的开端，若CX中的值为0，则越过该循环。

4．子程序调用和返回指令

如果在一个程序中的多个地方，或多个程序的多个地方用到同一段程序，那么可将这段程序单独放在内存的某一区域中。每当需要执行这段程序时，就用调用指令转到这段程序去执行，执行完毕再返回到原来的程序。为了便于模块化程序设计，把程序中的某些具有独立功能的部分，编写成独立的程序模块，这种程序模块被称为“子程序”或“过程”。程序中可由调用程序(或称为主程序)调用这些子程序，而在子程序执行完后返回调用程序继续执行。8086系统为此提供了调用指令CALL和返回指令RET。

(1) CALL指令

CALL指令实现子程序(或过程)的调用，调用结束后，要由所调用的子程序(或过程)中的返回指令RET返回CALL指令的下一条指令继续执行。为了保证返回的正确性，CPU会自动将CALL指令的下一条指令所在的内存地址推入堆栈；当子程序执行RET指令时，会将堆栈栈顶的内容弹出，放到IP和/或CS寄存器中，以保证子程序的调用过程和返回过程的正确性。

8086允许子程序在现行代码段区域中(此时子程序属性为NEAR)，也可以不在现行代码区域中(此时子程序属性为FAR)。为了确保正确返回，CALL指令的类型必须与RET指令的类型相匹配。

CALL指令有两种得到目标地址的方法：直接寻址和间接寻址。

直接寻址意味着目标地址在CALL指令的操作码部分直接给出。间接寻址意味着目标地址在指定的寄存器或内存单元中，以间接方式给出，故CALL指令有4种类型。

一个过程用于段内调用，则在过程定义中属性应为NEAR。而一个过程用于段间调用，则在过程定义中属性应为FAR。

① 段内直接调用

```
CALL    NEAR 目标地址
```

② 段内间接调用

```
CALL    WORD PTR [SI]
CALL    BX
```

③ 段间直接调用

```
CALL    FAR 目标地址
```

④ 段间间接调用

```
CALL    WORD PTR [SI]
CALL    BX
CALL    WORD PTR 62 [BX][DI]
```

（2）子程序返回指令 RET

与调用指令 CALL 相对应的是返回指令 RET。返回指令通常作为一个子程序或过程的最后一条指令，用于返回调用这个子程序的断点处。

返回指令在执行时，会从堆栈顶部弹出返回地址。为了能够正确返回，返回指令的类型要与调用指令的类型相一致。也就是说，如果一个子程序是供段内调用的，那么末尾用段内返回指令，执行时从栈顶弹出 2 字节作为返回地址的偏移量；如果一个子程序是供段间调用的，那么末尾用段间返回指令，执行时，从栈顶弹出 4 字节，前 2 字节作为返回地址的偏移量，后 2 字节作为返回地址的段地址。

段内返回和段间返回指令的形式是一样的，差别在于汇编后的机器代码不同。在汇编语言子程序被汇编成机器代码时，对于 RET 指令究竟产生段内返回还是段间返回对应的机器代码，是根据在汇编语言程序中加入的伪指令决定的。

“RET　n”指令被称为带参数返回指令。其中，n 可以是 0000H～FFFFH 范围内的任何一个偶数。带参数返回指令是在完成通常的 RET 指令操作后，再根据所带参数值 n 修改堆栈指针，在所得 SP 上加上 n 值，即跳过堆栈中的 n 个单元空间，这是便于调用程序在用 CALL 指令调用子程序以前，把子程序所需要的参数入栈（为 n/2 个参数），以便子程序运行时使用这些参数。当子程序返回后，这些参数已不再有用，就可以修改指针，使其指向参数入栈以前的值。例如，“RET　4”，在返回主程序后，使原来存放在堆栈中的 4÷2（即 2）个参数被舍弃，表示从栈顶弹出返回地址后，再使 SP 的值加 4。

5．中断指令和中断返回指令

当程序运行期间遇到某些特殊情况时，需要计算机暂停现行程序的执行，转而执行一组专门的例行程序来进行处理，这种情况被称为中断，转而执行的这组程序被称为中断服务子程序。8086 的中断分为内部中断和外部中断两类。内部中断包括像除法运算中遇到除以 0 时产生的中断，或者程序中为了做某些处理而设置的中断指令等。外部中断主要用来处理 I/O 设备与 CPU 之间的通信。

内部中断都是通过软件调用的非屏蔽型中断，又称为软件中断，其中包括单步中断、除法出错中断、溢出中断（INTO）和指令中断（“INT　n”）。与中断指令对应的是中断返回指令（IRET），与子程序的最后一条指令必是 RET 返回指令一样，任何中断处理程序的最后一条指令必是 IRET。

（1）中断指令“INT　n”

与 CALL 指令一样，中断指令执行时转到中断服务（又称为中断例行）程序，中断服务程序执行完毕，返回 INT 指令的下一条指令继续执行。为了正确返回，计算机自动将 INT 指令的下一条指令所在地址推入堆栈；当执行中断服务程序中的 IRET 指令时，将从栈顶弹出返回地址。与 CALL 指令不同的是，中断执行时还要保存现场信息，需要把反映现场状态的标志寄存器 FR 的内容入栈，然后转到中断服务程序去执行。当从中断返回时，除要恢复 IP 和 CS 外，还要恢复原有 FR 中的标志。

指令执行的操作如下。

<1> 将标志寄存器 FR 的内容推入堆栈，并将中断允许标志 IF 和单步标志 TF 清零。

```
SP ← SP-2
(SP+1, SP) ← 标志寄存器 FR
```

```
TF ← 0, IF ← 0
```

<2> 将返回地址的段地址推入堆栈。

```
SP ← SP-2
(SP+1, SP) ← 段寄存器 CS
```

<3> 将返回地址的偏移地址推入堆栈。

```
SP ← SP-2
(SP+1, SP) ← 指令指针寄存器 IP
```

<4> 分别将中断服务（例行）程序的偏移地址和段地址存入 IP 和 CS 寄存器，以指示下一条要执行的程序位置。

指令中的 n 为中断类型号（起到索引的作用），可以是常数或常数表达式，其值在 0～255 的范围内。指令“INT　n”启动一个有中断类型号规定的中断过程。

中断例行程序的入口地址称为中断向量（或称矢量）。8086 CPU 安排存储器的最低地址区的 1024 字节（物理地址从 00000H～003FFH）为中断向量区，其中存放着 256 种类型的中断例行程序的入口地址，构成一个中断向量表。由于每个中断向量占 4 个单元，所以中断指令中的中断类型号乘以 4 得到一个单元地址，由此地址开始的前 2 个单元中存放的是中断处理程序入口地址的偏移量，后 2 个单元中存放的是中断处理程序入口地址的段地址。

类型 0～4 的内部中断，对应于 8086 系统的某一特定功能的实现。如 n=0，程序中发生被 0 除事件时，CPU 自身产生该中断，故没有“INT　0”这条指令。

n=1，为单步中断，CPU 进入单步中断的依据是标志寄存器中的单步标志 TF=1，也就是说，该中断是由 CPU 测试 TF 标志为 1 而自动产生的。

n=2，为非屏蔽中断，当 CPU 的外部引脚 NMI 上有中断引入时，则产生一个类型为 2 的中断。

n=3，称为断点中断，用户在进行汇编语言程序调试时可设置一个调试断点。

（2）溢出中断指令 INTO

8086 指令系统中没有专门针对带符号数的加法、减法指令（有专门针对带符号数的乘法、除法指令），但 8086 指令系统提供了一条溢出中断指令，专门用来判断带符号数在加、减法运算时是否有溢出。当运算使 OF 标志为 1 时，执行 INTO 指令就会进入类型 4 的溢出中断。用户在进行带符号数加、减法运算时，考虑运算结果溢出的处理程序入口地址应存放在中断矢量地址为 10H（4×4=16）的中断向量表中。

（3）中断返回指令 IRET

与中断指令对应的是中断返回指令 IRET，与子程序的最后一条指令必是 RET 返回指令一样，任何中断处理程序的最后一条指令必是 IRET。

3.4.6　处理器控制指令

处理器控制指令可以用来控制 CPU 的操作。从前面对标志寄存器 FR 的介绍中可知，8086 CPU 的 FR 中有 3 位控制标志，即中断允许标志（IF）、方向标志（DF）和单步（陷阱）标志（TF）。用处理机控制指令，对这些标志置“1”或清“0”，则可以控制 CPU 按要求工作。例如，将 IF 置“1”，则 CPU 将被允许接收外部的可屏蔽中断；若将 IF 清“0”，则 CPU 将被拒绝接

收外部的可屏蔽中断。在使用串指令编程时，将 DF 置“1”，则内存单元地址指针自动向减小的方向移动；若 DF 清“0”，则内存单元地址指针自动向增加的方向移动。

控制 CPU 处于等待状态指令、让 CPU 封锁总线指令、使 CPU 空操作指令等指令在简单的汇编语言程序设计中使用较少。

1．标志控制指令

标志状态的改变是由这些指令直接控制的，而不会被某些运算指令的运算结果影响，可以控制的标志有 IF、DF 和 CF。

- STC：使 CF 置“1”。
- CLC：使 CF 清“0”。
- CMC：使 CF 取反。
- STD：使 DF 置“1”。
- CLD：使 DF 清“0”。
- STI：使 IF 置“1”。
- CLI：使 IF 清“0”。

2．外同步指令

（1）处理器暂停指令 HLT

HLT 指令经常和中断过程联系在一起，它的执行实际上是用软件的方法使 CPU 处于暂停状态等待硬件中断，而硬件中断的进入又使 CPU 退出暂停状态。由 NMI 引脚上引入的不可屏蔽（非屏蔽）中断和在中断允许（IF=1）的情况下，INTR 引脚上引入的可屏蔽中断，会使 CPU 退出暂停状态；同时，对系统的复位操作也会使 CPU 退出暂停状态。

（2）等待指令 WAIT

WAIT 指令使 CPU 处于等待状态，直到 CPU 的 $\overline{\text{TEST}}$ 引脚上的信号为有效。8086 CPU 的第 23 号引脚 $\overline{\text{TEST}}$，是用来使处理器与外部硬件同步的，也被称为同步指令，为低电平有效的输入信号。若测试到该信号无效，CPU 继续执行 WAIT 指令，以令 CPU 处于空闲等待状态；若测试到该信号有效，CPU 转去执行 WAIT 指令的下一条指令。

（3）换码指令/交权指令 ESC

8086 CPU 与数值数据处理器 8087 和输入/输出处理器 8089 可以组成多处理机系统。在这种系统中，8087 和 8089 被称为协处理器。此时 8086 CPU 按最大模式运行。程序在执行过程中，遇到 ESC 指令，就表示 8086 调用协处理器工作，同时协处理器在系统加电后，不断检测 8086 CPU 是否需要自己协助工作；当执行 ESC 指令时，协处理器便立即响应。

ESC 指令可以使其他处理器接收来自 8086 指令流中的指令，且可用 8086 的寻址方式对存储器操作数进行操作。执行 ESC 指令，仅仅完成访问一个存储器操作数并把它放在总线上。

（4）总线封锁指令 LOCK

LOCK 指令可以放在任何一条指令前面，所以 LOCK 指令实际上是一个特殊的 1 字节封锁线。LOCK 指令使工作在最大模式下的 8086 CPU，在执行下一条指令期间发出总线封锁信号，即 LOCK 引脚变为低电平有效信号。这样，在 CPU 访问存储器或外设时，总线控制器会对总线进行封锁，使其他处理器得不到总线控制权，从而不能访问存储器或外设。

（5）空操作指令 NOP

NOP 指令不执行任何操作，也不影响任何标志，其机器码占 1 个存储单元，其功能等同于交换指令“XCHG　AX, AX”。在调试程序时往往用 NOP 指令占据一定的存储单元，以便在正式运行前用其他必要的指令代替。由于 CPU 执行任何一条指令都是需要时间的，因此当程序中需要用软件进行延时时，可以使用 NOP 指令来较精确地定时，而不会影响程序的其他功能。

习 题 3

1．分别指出下列指令中的源操作数和目的操作数的寻址方式。

（1）

```
MOV     AX, [SI]
```

（2）

```
MOV     DI, 100
```

（3）

```
MOV     [BX], AL
```

（4）

```
MOV     [BX][SI], CX
```

（5）

```
ADD     DX, 106H[SI]
```

（6）

```
PUSH    AX
```

（7）

```
AND     DS:[BP], AX
```

（8）

```
OR      AX, DX
```

2．已知寄存器(DS)=2000H，(SS)=1500H，(ES)=3200H，(SI)=0A0H，(BX)=100H，(BP)=10H，数据段中变量 VAL 的偏移地址值为 50H。试指出下列各条指令中源操作数的寻址方式是什么？对于存储器操作数，其物理地址是多少？

（1）

```
MOV     AX, [100H]
```

（2）

```
MOV     CX, ES:[BX]
```

（3）

```
MOV     DX, [BX][SI]
```

（4）

```
MOV     AX, VAL[SI]
```

（5）

```
MOV     BX, 1234[BX]
```

（6）

```
MOV     AX, [BP]
```

3．指令理解题。

判断下列指令有无错误，若有，则改正。

（1）

```
PUSH    CL
```

（2）

```
ADC     AX, 0ABH
```

（3）

```
OUT     3EBH, AX
```

（4）

```
MUL     AL, CL
```

（5）

```
MUL     AX, 25
```

（6）

```
ROL     DX, 5
```

（7）

```
CALL    FAR SUM
```

（8）

```
POP     FR
```

（9）

```
MOV     234H, BX
```

（10）

```
INC     [SI]
```

（11）

```
ADD     [BX], 456H
```

（12）

```
INT     0
```

（13）

```
DIV     AX, BX
```

（14）

```
DEC     [BP]
```

（15）

```
XLAT    BX
```

（16）

```
ADD     CX+1
```

（17）

```
DAA     AX
```

4．选择题

（1）带符号数–86 在微机中所表示的二进制数值是（　　）。

A．10101010B　　B．01100101B

C．10011101B　　D．11001011B

（2）执行“MOV　DL, 2AH”和“SHR　DL, 1”两条指令后，DL 寄存器与 CF 标志分别为（　　）。

A．DL=10110110　CF=1　　B．DL=00110101　CF=0

C．DL=00110100　CF=1　　D．DL=00010101　CF=0

（3）可将 AX 寄存器中的 D_0、D_5、D_8 和 D_{11} 位求反，其余位不变的指令是（　　）。

A．AND　AX, 921H　　B．OR　AX, 910H

C．XOR　AX, 0921H　　D．XOR　AX, 0110H

（4）某存储单元的物理地址为 3B4FEH，其段地址和偏移地址可分别为（　　）。

A．3B4FH 和 104EH　　B．3B40H 和 00FEH

C．2A00H 和 114FEH　　D．3B4FEH 和 0

（5）两个 8 位二进制数 00110101 及 10110110 做“异或”操作后，寄存器 FR 的下面 3 个状态标志分别是（　　）。

A．PF=1　SF=0　ZF=0　　B．PF=0　SF=1　ZF=1

C．PF=0　SF=1　ZF=0　　D．PF=1　SF=1　ZF=1

（6）已知（DX）=1234H，执行“SHL　DL, 1”和“RCL　DH, 1”两条指令后，DX 寄存器与 CF 标志分别为（　　）。

A．DX=1234H　CF=1　　B．DX=2468H　CF=0

C．DX=1234H　CF=0　　D．DX=2468H　CF=1

5．已知，(DS)=2000H，(BX)=100H，(SI)=02H，从物理地址 20100H 单元开始，依次存放数据 12H、34H、56H、78H；而从物理地址 21200H 单元开始，依次存放数据 2AH、4CH、8BH、98H。试说明下列各条指令单独执行后 AX 寄存器的内容。

（1）

```
MOV     AX, 3600H
```

（2）

```
MOV     AX, [1200H]
```

（3）

```
MOV     AX, BX
```

（4）

```
MOV     AX, [BX]
```

（5）

```
MOV     AX, 1100H[BX]
```

（6）

```
MOV     AX, [BX][SI]
```

6．设堆栈指针 SP 的初值为 2400H，(AX)=4000H，(BX)=3600H。问：

（1）执行指令“PUSH　AX”后，SP=?

（2）再执行“PUSH　BX”和“POP　AX”后，(SP)=? (AX)=? (BX)=?

试画出堆栈变化示意图。

7. 指出下面指令序列的执行结果。

(1)

```
MOV     DX, 2000H
MOV     BX, 1000H
XCHG    BX, DX
```

BX=　　DX=

(2)

```
MOV     AX, 1234H
PUSH    AX
POP     BX
```

AX=　　BX=

(3)

```
LEA     DX, [2000H]
MOV     BX, DX
```

BX=

(4)

```
MOV     AL, 08
ADD     AL, 08
AAA
```

AX=

(5)

```
MOV     AL, 48H
ADD     AL, 39H
DAA
```

AL=

(6)

```
AND     AL, AL
MOV     AL, 80
ADC     AL, AL
```

AL=

(7)

```
MOV     DX, 0FFFFH
NEG     DX
```

DX=

(8)

```
MOV     BL, 0B8H
ROR     BL, 1
```

BL=　　CF=

(9)

```
SUB     AX, AX
AND     DX, AX
```

DX=

(10)

```
MOV     CL, 3
MOV     AH, 42H
SHR     AH, CL
```

AH=　　CF=

(11)

```
MOV     AX, 34EBH
MOV     CL, 5FH
DIV     CL
```

AX=

(12)

```
MOV     AL, 98H
CBW
```

AX=

(13)

```
MOV     BL, 9
MOV     AX, 0702H
AAD
DIV     BL
```

AX=

(14)

```
MOV     AL, 08
MOV     BL, 09
MUL     BL
AAM
```

AX=

(15)

```
MOV     [1000H], WORD PTR 12H
MOV     BX, [1000]
BX=?
```

第 4 章　8086 汇编语言程序设计

本章导读

- 8086 汇编语言的语句
- 8086 汇编语言中的伪指令
- 8086 汇编语言中的运算符
- 汇编语言程序设计
- 宏定义与宏调用
- 汇编语言程序设计实例与上机调试

汇编语言是用指令的助记符、符号地址、标号等书写程序的语言，实际上是机器语言的一种符号表示，主要特点是可以使用助记符来表示机器指令的操作码和操作数，可以用标号和符号来代替地址、常量和变量。助记符一般是表示一个操作的英文单词的缩写，易于记忆和识别。程序员不必具体安排程序（代码和数据）在存储器中的物理位置。

用汇编语言编写的程序被称为汇编语言源程序（简称源程序），指令系统中的每条指令都能作为源程序的基本语句。汇编语言是与机器语言密切相关的，是面向机器的语言。CPU 不同的计算机有着不同的汇编语言。汇编语言源程序不能直接在计算机上运行，需要翻译成机器语言程序（目标程序）。

将汇编语言源程序翻译成机器语言程序的过程叫汇编。完成汇编任务的程序叫汇编程序，是一种计算机应用程序。汇编程序除完成将汇编语言源程序翻译成机器语言程序外，还有以下任务：① 按用户要求自动分配存储区（包括程序区、数据区等）；② 自动把各种进制数转换成二进制数；③ 计算表达式的值；④ 对源程序进行语法检查并且给出语法出错信息。

4.1　8086 汇编语言的语句

编写一个汇编语言源程序需要两种语句：一种是由 CPU 执行的语句，被称为指令性语句；另一种是由汇编程序执行的语句，被称为指示性语句。第 3 章中介绍的指令是在指令性语句中使用的，而指示性语句中使用的指令被称为伪指令（或指示符），以便与指令分开。伪指令的功能是提供汇编信息，如操作数定义、属性（是字节型还是字型等）。下面分别介绍这两种语句的格式。

1．指令性语句格式

指令性语句包括4段，格式如下：

```
[标号：] 操作码 [操作数1 ] [, 操作数2]    [; 注释]
```

语句中用“[ ]”括起来的段是可选段。

标号段：以“:”分界，该段不是每条指令必需的，为提供其他指令引用而设。一个标号与一条指令的地址符号名（即在当前程序段内的偏移量）相联系。在同一程序段中，同样的标号名只允许定义一次。

操作码段：操作码助记符是指令系统规定的，任何指令性语句必须有该段，因为它表明程序中的一个环节，有着一定的操作性质并完成一个操作。操作码助记符通常被称为关键字或保留字，用户不能用这些字或词作为变量名、标号、标识符等。

操作数段：表明操作的对象，操作数可以是常数、寄存器、标号、变量和表达式。在8086指令系统中，有些操作中可能有不止一个操作对象，有的操作对象隐含在操作码中，若指令中有两个操作数，则需用逗号分界。

注释段：语句中以“;”开始的部分为注释，这部分不被汇编程序翻译，仅作为对该语句的一种说明，便于他人理解代码、展开交流等。

2．指示性语句格式

指示性语句也包括4段，格式如下：

```
[标识符（名字）] 指示符（伪指令）  表达式    [; 注释]
```

标识符段：标识符是一个用字母、数字或带有下画线表示的符号，标识符定义的性质由伪指令指定。

指示符段：指示符又称为伪指令，是汇编程序规定并执行的命令，能将标识符定义为变量、程序段、常数、过程等，且能给出其属性。

表达式段：表达式是常数、寄存器、标号、变量与一些操作符相结合的序列，可以有数字表达式和地址表达式两种。在汇编期间，汇编程序按照一定的优先规则，对表达式进行计算后得到一个数值或一个地址值。

3．有关属性

存储器操作数的属性有3种：段值、段内偏移量和类型。

段值属性：存储器操作数的段起始地址，此值必须在一个段寄存器中，而标号的段则总是在CS寄存器中。

段内偏移量属性：16位无符号数，代表从段起始地址到该操作数所在位置之间的字节数。在当前段内给出变量的偏移量值等于当前地址计数器的值，当前地址计数器的值可以用“$”来表示。

类型属性：标号的属性用来指出该标号在本段内引用还是在其他段中引用，在段内引用，称为NEAR，指针长度为2字节；在段间引用，则称为FAR，指针长度为4字节。变量的类型属性用来指出该变量所保留的字节数，主要是指BYTE（1字节的字节型）、WORD（2字节的字型）或DWORD（4字节的双字型）。

4.2 8086 汇编语言中的伪指令

伪指令又称为伪操作，在汇编程序的指示性语句中作为指示符，在对汇编语言源程序进行汇编期间，是由汇编程序处理的操作。伪指令可以对数据进行定义，为变量分配存储区，定义一个程序段或一个过程，指示程序结束等。

本节介绍 8086 中几个常用的伪指令，并给出用伪指令定义的语句格式。

4.2.1 符号定义语句

1．等值语句

等值语句的格式如下：

```
符号名    EQU  表达式
```

在程序中有时会多次出现同一个表达式，为了方便起见，可以用赋值伪操作给表达式赋予一个名字，此后凡需要用到该表达式之处，都可以用这个名字来代替。表达式可以是任何有效的操作数格式、有效的助记符，或能求出常数值的表达式，甚至是一条可执行的指令。例如：

```
PORT    EQU  1234          ①
BUFF    EQU  PORT+58       ②
MEM     EQU  DS:[BP+20H]   ③
COUNT   EQU  CX            ④
ABC     EQU  AAA           ⑤
```

当程序中出现如下语句时

```
MOV     AX, PORT
MOV     BX, BUFF
```

根据 EQU 的功能，上面语句实际上等价于

```
MOV     AX, 1234
MOV     BX, 1292
```

语句③中，若要引用加段前缀 DS（段超越）的 BP 基址寻址，直接用符号名 MEM 即可。语句④中，符号名 COUNT 定义为 CX 寄存器，程序中使用 CX 作为计数器时，就可以直接用 COUNT 表示。语句⑤中，把符号名 ABC 定义为一条 ASCII 码加法后的调整指令 AAA。

在同一源程序中，一个符号名只允许用 EQU 语句定义一次，若再次定义同一符号名，程序在汇编时会给出语法错误。

这里补充说明一点，8086 汇编语言中不少语句和 C 语言中的语句非常类似，比如 EQU 语句和 C 语言中的 define 语句的功能就几乎完全一样。比如：

```
#define  PORT  1234
…
int  xx=PORT;
…
```

可以看出，xx=PORT 相当于 xx=1234。

2．等号语句

等号语句的格式如下：

```
NUM = 34
...
NUM = 34+1
```

用“=”为符号名赋值与 EQU 类似，但等号语句允许对同一符号名多次赋不同的值，即对已赋值的符号名引用过后，可再次赋予新的值，以便下次引用。

4.2.2 变量定义语句

变量定义语句的格式如下：

```
符号名  DB/DW/DD 表达式
```

当一个符号名用伪指令 DB、DW 或 DD 等定义后，就成为了一个变量。

- 用 DB 定义，表明变量为字节型数据（8 位）。
- 用 DW 定义，表明变量为字型数据（16 位）。
- 用 DD 定义，表明变量为双字型数据（32 位）。

8086 中还可以用 DQ、DT 等定义变量，在此不一一介绍。

用变量定义语句对变量定义后，变量就有了属性，这些属性可用 8086 汇编程序中的某些运算符分析出来。变量被定义后的属性如下：

- 字节型、字型和双字型等数据类型。
- 分配内存单元。分配原则是如果没有特别指定，变量按出现的先后顺序从偏移地址 0000H 单元开始存放。
- 按低位字节数据存放在低地址单元、高位字节数据存放在高地址单元的原则（或称为反向存储）给内存赋值。
- 变量定义一般在数据段中，故一个变量被定义后，就有了段地址和在该段的偏移地址。

变量定义语句中的表达式形式有多种，定义的功能也有所不同。下面给出变量定义语句的具体形式。

1．定义一组数据

例如：

```
BUFF1        DW  1234H, 0ABCDH, 8EH, -79DH
BUFF2        DB  12H, 34H, CDH, 8EH
```

定义 BUFF1 为字型变量，共有 4 个参数；定义 BUFF2 为字节型变量，有 4 个参数。显然，BUFF1 就是字型的数组，BUFF2 是字节型数组，它们的元素个数都是 4 个。在 BUFF1 数组中，虽然 8EH 是按两位的十六进制书写的，但属于字型，实际上是 008EH。这 8 个参数放在内存中，地址默认为从某段的偏移地址为 0000 处开始放数据。

在早期的 C 语言中，整型数是 16 位的，因此 DB 和 DW 与 C 语言中的定义变量类型语句 char 和 int 型是非常类似的。上述 BUFF1 和 BUFF2 的 DW 和 DB 定义语句和下面的 C 语言语句等效：

```
int  buff1[4] = {0x1234, 0x0abcd, 0x8e, -0x79d};
char  buff2[4] = {0x12, 0x34, 0x0cd, 0x8e};
```

对于 BUFF1，按反向存储的原则，每个数据占 2 字节。第 1 个参数为十六进制数；第 2 个参数为字母开头的十六进制数，为与符号名区分开，在字母开头的十六进制数前加 0；第 3 个参数看上去为 8 位数，但定义时用的是 DW，故为其分配 2 字节，汇编程序会将高 8 位补 0；第 4 个参数是带符号的，从前面的章节可知，带符号数在计算机中以补码存放，故负数–79DH 以 F863H 存放在存储单元中，如图 4-1 所示。

下面给出几个指令，供读者理解变量定义语句的作用。

```
MOV     AX, BUFF1
MOV     BX, BUFF1+2
MOV     DL, BUFF2
```

上述指令实际上是：

```
MOV     AX, [0000]
MOV     BX, [0002]
MOV     DL, [0008]
```

0000	34
0001	12
0002	CD
0003	AB
0004	8E
0005	00
0006	63
0007	F8
0008	12
0009	34
000A	CD
000B	8E

图 4-1　变量的存储

注意，对于语句 MOV　BX, BUFF1+2，不要理解为 BUFF1 内容加 2 后再送给 BX，这是因为包含伪指令结构的指示性语句只能提供汇编信息，生成标准的汇编指令，它们不会对操作数进行加工处理，对操作数进行加工处理只有指令性语句才能完成。另外，BUFF1 具有字的属性，BUFF2 具有字节的属性，所以下面的指令是错误的：

```
MOV     AL, BUFF1
MOV     DX, BUFF2
```

2．定义一串字符

例如：

```
STR  DB  'Welcome !'
```

定义了 STR 为一个字节型变量，单引号内表示是字符串，字符以 ASCII 码的形式存放，每个字符占 1 字节。故对字符串只用 DB 定义。

3．定义保留存储单元

在程序设计中，如果希望将运算结果保存到内存中，则在设计中就要预留一部分存储单元。也就是说，这些内存单元不需要预先赋值。无论是存储器还是寄存器，它们都是一种双稳态门电路，故每一位（bit）不是“1”就是“0”，这些器件中总是有值，当没有进行特定赋值时，其内容为随机值。例如：

```
SUM     DW    ?, ?
```

从 SUM 偏移地址开始，为两个字型数据保留了 4 字节的内存单元。

4．复制操作

复制操作符 DUP（Duplication）可以预置重复的数值，DUP 之前的数字表示重复的次数，在 DUP 后面用括号将重复的内容括起来。例如：

```
ALL_ZERO  DB  0, 0, 0, 0, 0
```

将连续的 5 个字节赋 0。若用复制操作，改为

```
        ALL_ZERO    DB  5  DUP(0)
```

可达到同样的效果。

5．将已定义的地址存入内存单元

定义过的标号或用 PROC 过程定义过的过程名（子程序名或中断服务程序名）都有段地址和偏移地址属性。若希望将变量、标号或过程名的段地址和偏移地址保存到存储单元，可以用下面的方式完成。例如：

```
        LIT     DD  CYC
        ......
CYC:    MOV     AX, BX
```

将标号 CYC 的段地址和偏移地址存放在以变量 LIT 开始的 4 字节单元中。有时可以将程序中所有子程序的首地址列在一起，采用查表的方式动态地转入所需的子程序入口。

4.2.3 段定义语句

段定义语句可按段来组织程序和使用存储器。这些语句包括 SEGMENT、ENDS、ASSUME、ORG 等。

为了清晰表达上面语句的必要性和实用性，首先定义

```
xx1     DB  12H, 34H
xx2     DW  1200H, 3400H
```

然后利用上面的变量名就可以编写汇编语言源程序了，如在程序中出现如下指令性语句：

```
        MOV     AL, xx1
        MOV     DX, xx2
```

8086 体系对存储器的管理是分段的。对于上面的指令，在翻译为对应的机器码时，汇编程序不能确定 xx1 和 xx2 是在哪一个段而无法进行正确的汇编。xx1 和 xx2 不一定在数据段，它们可以在其他三个段中。在不同段，对应的 MOV 指令的机器码是不一样的。比如，指令 MOV AL, DS:[0000]和指令 MOV AL, ES:[0000]的机器码是不一样的。

为了解决上面的问题，8086 汇编系统给出了段定义语句。

1．段定义语句格式

段定义语句的格式如下：

```
段名    SEGMENT [定位类型] [组合类型] ['类别']
......
段名    ENDS
```

存储器的物理地址是由逻辑段基地址和逻辑偏移地址组合而成的，语句 SEGMENT 和 ENDS 把汇编语言源程序分成段，这些段就相当于存储器区段。

对于段内指令的转移和调用，在指令中只需包含目标地址单元的 16 位偏移量，在段间的转移和调用指令才需要给出段地址和偏移地址。使用当前数据段和当前堆栈段的数据访问指令，也只需在指令中给出数据所在内存单元的 16 位偏移地址。8086 按照规定的组合方式生成

物理地址。

当指令访问的是当前段之外的数据单元时，必须在指令中加入段超越前缀或修改段寄存器的内容。汇编程序在将源程序转换成目标程序时，必须确定标号和变量的偏移地址，并且需要把有关信息通过目标模块传递给连接程序，这样才能把解决同一个问题的不同段、不同模块的程序连接起来，形成一个可执行程序。

段定义语句中的 SEGMENT 和 ENDS 必须成对，其中省略号部分可以是汇编语言的指令性语句和指示性语句。

段名可以是包括下画线在内的字母、数字串，由程序设计者自定。定位类型、组合类型、类别是赋给段名的属性，加上方括号表示这些属性可以省略，省略表示该程序段与其他段没有联系，是独立的；若不能省略，各属性项的书写顺序不能错，并以空格分界。

（1）定位类型

定位类型表示此段在内存中的起始边界要求，可以是 PAGE（页）、PARA（节）、WORD（字）、BYTE（字节）。

PAGE 要求该段从页的边界开始，段地址能被 256 整除，即十六进制的段地址最后两位为 0。PARA 要求该段从节的边界开始，段地址能被 16 整除，即十六进制的段地址最后一位为 0。当定位类型省略时，隐含为 PARA。WORD 要求该段从字的边界开始，段地址为偶数值。BYTE 可以从该段边界任何地址开始。

（2）组合类型

组合类型用来告诉链接程序本段与其他段的关系，包括 NONE、PUBLIC、COMMON、STACK、MEMORY 和 AT。

NONE：表示本段与其他段逻辑上没有关系，每段都有自己的基地址。组合类型省略时属于该类型。

PUBLIC：链接程序先把本段与其他模块中同名、同类别的段链接在一起，然后为所有段指定一个共同的段基地址，并连接成一个物理段。各段的链接顺序由链接命令指定。

COMMON：链接程序为本段与其他模块中同名、同类别的段指定一个相同的段基地址。这些段可以相互覆盖，段的长度取决于最长的 COMMON 段。

STACK：规定被链接的程序中必须有至少一个 STACK 属性的段，即堆栈段。如果多于一个，则在初始化时会将第一个 STACK 段的地址送入 SS 寄存器。而段与段之间的链接按 PUBLIC 方式处理。

MEMORY：链接程序把本段定位为几个互连段中地址最高的段。若有多个 MEMORY 段，链接程序认为所遇到的第一个为 MEMORY，其余段则具有 COMMON 属性。

AT 表达式：链接程序将表达式计算出来的 16 位地址作为段地址，但不能用来指定代码段，这个类型使得在某一固定的存储区内的某一固定偏移地址处定义标号或变量，以便程序以标号或变量形式访问这些存储单元。

（3）类别

类别可以是任何合法的名称，用单引号括起来，如'STACK'、'CODE'。在定位时，链接程序把同类别的段集中在一起。

这里要说明的是，段定义语句功能比较复杂，为了让读者能尽快使用段定义语句编程，建议编程时不必过多考虑 segment 伪指令的参数设置，直接使用如下精简形式：

```
段名        SEGMENT
…
段名        ENDS
```

比如，对于本节开始时提出的问题，结合段定义语句，给出如下程序段：

```
Mydata    SEGMENT
xx1       DB  12H, 34H
xx2       DW  1200H, 3400H
Mydata    ENDS
```

这样，当在程序中出现如下指令性语句时

```
          MOV     AL, xx1
          MOV     DX, xx2
```

汇编源程序对指令的解释是：

```
          MOV     AL, Mydata: xx1
          MOV     DX, Mydata: xx2
```

上面指令说明了变量 xx1 和 xx2 在段名为 Mydata 的存储器段中，但这个 Mydata 段是什么属性的段并没有给出说明，也就是 Mydata 段是数据段还是其他三个段中的一个是不明确的，为此 8086 汇编语言给出了和段定义语句配套的另一个伪指令 ASSUME。

2．段说明语句

ASSUME 伪指令在汇编时能提供正确的段码，使汇编程序知道程序的段结构，在各种指令执行时该访问哪一段。其格式如下：

```
          ASSUME  段寄存器名: 段名[, …]
```

段寄存器可以是 CS、DS、SS 或 ES，而段名则是用 SEGMENT 定义过的标识符。因一个程序中可能不止一个程序段，方括号中的内容表示可按实际情况添加。

在程序中，ASSUME 伪指令只是指定某段分配给哪个段寄存器，并不能把段地址装入寄存器中（因为伪指令不由 CPU 执行）。

利用 ASSUME 语句，可以对变量 xx1 和 xx2 进行完整定义。比如：

```
Mydata    SEGMENT
xx1       DB      12H, 34H
xx2       DW      1200H, 3400H
Mydata    ENDS
Mycode    SEGMENT
          ASSUME  CS:Mycode, DS:Mydata
          …
          MOV     AL, xx1
          MOV     DX, xx2
          …
Mycode    ENDS
```

上面的语句在计算机内存中定义了两个段，分别是 Mydata 和 Mycode。伪指令 ASSUME 对这两个段名的属性做了说明：Mydata 段为数据段，Mycode 段为代码段。显然由指令性语句构成的程序段就应该放在 Mycode 段中。汇编源程序对 MOV 指令的解释是：

```
          MOV     AL, DS:xx1
```

```
        MOV       DX, DS:xx2
```

显然，对于上面的 MOV 指令，汇编程序完全可以理解并把这两条指令翻译为 8086 指令机器码。

3．ORG 伪指令与地址计数器$

ORG 伪指令的格式如下：

```
        ORG  <表达式>
```

此语句指定在它之后的代码或数据存放的起始地址的偏移量，以表达式的值作为起始地址，连续存放程序或数据，除非遇到一个新的 ORG 语句。

比较下面两个例子，请读者体会 ORG 的功能。

（一）

```
Mydata    SEGMENT
xx1       DB    1,2,3,4
Mydata    ENDS
```

（二）

```
Mydata    SEGMENT
ORG       0010H
xx1       DB    1,2,3,4
Mydata    ENDS
```

上面两个例子都是在计算机内存中建立一个 Mydata 的段，（一）中的数组 xx1 分布在 Mydata 段内偏移地址 0000～0003 号区域中，而（二）中的数组 xx1 由于 ORG 指令的作用，分布在 Mydata 段内偏移地址 0010～0013 号区域中。

ORG 伪指令在 8086 汇编语言编程中不常用，但在 MCS51 单片机编程中经常被使用。

任何时候在使用存储器时，都先要给出存储单元地址。汇编程序在汇编时给出一个隐含的地址计数器，“$”是地址计数器的值，也就是当前所使用的存储单元的偏移地址。

4．PUBLIC 和 EXTRN 伪指令

当一个程序由多个模块组成时，必须通过命令将各模块连接成一个完整的、可执行的程序。所谓程序模块，是指单独编辑和汇编的、能够完成某个功能的程序，如主程序模块、各种功能的子程序模块等。正因为它们是独立汇编的，所以在编写程序时要使用 EXTRN 伪指令，表示本模块引用了在其他模块中定义的信息；使用 PUBLIC 伪指令，表示本模块提供被其他模块使用的信息。

（1）PUBLIC 伪指令

PUBLIC 伪指令表示，在连接时本模块中能够提供在其他模块中使用的名字，格式如下：

```
        PUBLIC  名字[, …]
```

其中，“名字”可以是模块中定义的一个变量或标号（包括过程名）。PUBLIC 伪指令可在一个汇编模块的任何一行出现，未经定义的符号名字不能被说明为 PUBLIC。

（2）EXTRN 伪指令

EXTRN 伪指令把某些名字的段和类型的属性告诉汇编程序，这些名字是本模块所要用的，但是它们在其他模块中定义。其格式如下：

```
        EXTRN  名字: 类型[, …]
```

其中，“名字”是在其他模块中定义过的，类型必须与说明它为 PUBLIC 的模块中的类型一致。

4.2.4 过程定义语句

在汇编语言中，用过程的定义来实现子程序功能。过程是程序的一部分，它们可被程序调用，每次可调用一个过程。当过程中的指令执行完后，控制返回调用点。段间的调用指令把过程返回的段地址和偏移地址同时推入堆栈，而段内的调用指令只将偏移地址入栈。

过程定义语句的格式如下：

```
过程名    PROC  NEAR/FAR
          ...
          RET
过程名    ENDP
```

其中，“过程名”是标识符，又是子程序入口的符号地址；NEAR 或 FAR 是类型属性，NEAR 属性是指该过程是一个段内的调用，而 FAR 指段间的调用，当属性省略时，自动设为 NEAR。伪指令 PROC 和 ENDP 必须成对出现。为了保证过程正确返回，CALL 指令的类型必须与过程的类型相匹配。

过程定义语句可把程序分段，以便理解、调试和修改。若整个程序由主程序和若干个子程序组成，则主程序和这些子程序都应包含在代码中。

一般，在过程的最后是一条 RET 语句，表示从栈顶弹出返回地址，以便返回调用点。ENDP 则告诉汇编程序，该过程在哪儿结束。

过程是可以嵌套的。一个过程中可以包括多个过程定义，堆栈的大小决定嵌套的深度，但过程不允许交叉。

4.2.5 结束语句

1．编辑结束语句——END

当编辑一个汇编语言源程序，而该程序不能单独执行，只是一个程序段时，可能还要与其他程序段连接，这时该程序的结束处应写上一条 END 语句。它告知汇编程序到此汇编结束，可以形成一个独立的文件。

2．可执行程序结束语句——END　标号

一个可执行的汇编语言源程序，在程序结束处应写一条带标号的 END 语句。一个可执行程序只能有一条这样的语句，END 后的标号是该程序要执行的第一条语句所在的存储器地址，这样汇编程序在汇编时，将该存储器的段地址送代码段寄存器 CS，将存储器偏移地址送指令寄存器 IP，也就是开始执行的指令位置由 CS:IP 指定了。

4.3 8086 汇编语言中的运算符

4.3.1 常用运算符和操作符

在 8086 汇编语言程序的指示性语句中可以有表达式。表达式中可以使用 3 种运算符和 2

种操作符。

1．算术运算符

算术运算符包括+（加）、–（减）、*（乘）、/（除）、MOD（求余），可以用于数字操作数或存储器地址操作数的运算中。用于地址表达式时，只有当结果有明确的物理意义时，算术运算符才有效，如对存储器地址操作数，有意义的运算符是“+”“–”。如果把两个不同段的偏移地址做加（减）运算是没有物理意义的。

2．逻辑运算符

8086 汇编的 4 种逻辑运算符分别为 AND、OR、XOR 和 NOT，它们只适用于数字操作数，其运算规则与逻辑运算规则相同。逻辑运算符与 8086 逻辑运算指令的助记符一样，但作为汇编的运算符时，由汇编程序在汇编时计算出结果，该结果作为指令性语句的操作数使用。

3．关系运算符

关系运算符连接的两个运算对象，必须都是数字或是同一段内的存储器地址。关系运算符有 6 种：EQ（相等），NE（不等），LT（小于），GT（大于），LE（小于等于）和 GE（大于等于）。

关系运算符的运算规则是：两个运算对象的关系是否满足某种关系，若满足，结果为全“1”，否则结果为“0”。例如：

```
        MOV     DL, 10H  LT 16
```

其中的源操作数在汇编时由汇编程序进行关系运算。根据运算规则可知，10H 不小于 16，其关系不成立（假），结果为 0。故上述指令实际上就是“MOV　DL, 0”，指令执行后，DL 寄存器的内容为 0。又如：

```
        AND     AX, 555  GT 222
```

其中的源操作数关系满足。指令执行后，AX 内容与 FFFFH 做“与”操作，则 AX 原内容不变。

4．分析操作符

分析运算能将定义过的变量、标号的存储器地址分解成它们的组成部分，如逻辑段基址、逻辑偏移地址、类型等。比如，有变量定义语句

```
BUFF    DW      1234H
```

变量名 BUFF 携带的信息非常丰富，如地址信息，包括偏移地址、段基地址，变量的类型是字型，变量的内容等。例如，指令

```
        MOV     AX, BUFF
```

是把名字为 BUFF 的内存单元的内容送给寄存器 AX，对于目的操作数来讲是寄存器寻址，而对于源操作数来讲是直接寻址。该指令执行完毕，寄存器 AX 的内容为 1234H。

上述指令是读取内存单元的操作，那么如果指令的功能是取名字 BUFF 所代表的地址或类型，将如何操作呢？这就要使用分析操作符。

（1）SEG 操作符

例如：

```
        MOV     AX, SEG  BUFF
```

设变量 BUFF 已在数据段中定义过，指令执行后，AX 寄存器有变量 BUFF 所在段的段地址。

（2）OFFSET 操作符

例如：

```
        MOV     BX, OFFSET  BUFF
```

设变量 BUFF 已在数据段中定义过，指令执行后，BX 寄存器有变量 BUFF 所在段的偏移地址。

又如：

```
        LEA     BX, BUFF
```

上面两语句执行后有相同的结果，区别在于：第一个语句是由汇编程序在汇编阶段，通过分析运算 OFFSET 求得变量 BUFF 的偏移地址，然后由 CPU 运行 MOV 指令，将它作为源操作数传送到 BX 寄存器；而第二个语句是直接由 CPU 执行有效地址传送指令完成的。

（3）TYPE 操作符

TYPE 运算可求出变量或标号的类型，类型用数字表示。变量有 3 种：1，字节型；2，字型；4，双字型。标号有 2 种：–1，NEAR（段内）；–2，FAR（段间）。例如：

```
DATA    SEGMENT
VAL1    DB      12H, 8EH
VAL2    DW      0A234H, 5B78H
VAL3    DD      9457B68DH
DATA    ENDS
CODE    SEGMENT
        ASSUME  DS:DATA, CS: CODE
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     DL, TYPE VAL1
        MOV     BL, TYPE VAL2
        MOV     CL, TYPE VAL3
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

上面是一个完整的汇编语言源程序，对于它的程序结构和格式在稍后进行解释，这里要了解的是 TYPE 运算符，程序可以通过上机调试得到结果。程序运行的结果是 DL 的内容为 1，BL 的内容为 2，CL 的内容为 4。若将其中的 TYPE 运算符换成 SEG、OFFSET、LENGTH 和 SIZE 运算符，并对程序相应处稍加修改，就可在计算机上得到另外一种结果，请读者对此自行分析。

（4）LENGTH 操作符

LENGTH 运算对使用 DUP 定义过的变量求元素个数，对其他方式定义的变量总是给出 1 作为结果。例如：

```
BUFF    DW      10 DUP(?)
```

执行如下命令

```
        MOV     CL, LENGTH BUF
```

则 CL 的内容为 10。注意：操作符 LENGTH 只对 DUP 有效。如果 DW 定义的是一系列数字，

如上例中的 VAL2，则 LENGTH 对其作用后返回的值为 1。

（5）SIZE 操作符

SIZE 操作符可以分析出一个使用 DUP 定义过的变量所有元素所分配的内存字节数。它与变量的类型和元素个数有关：SIZE = TYPE×LENGTH。

5．综合运算符（合成操作符）

（1）PTR 运算符

PTR 运算符的格式如下：

```
类型      PTR 表达式
```

对一个存储器操作数，不管原来是何种类型，现在都以 PTR 前的类型为准。也就是说，PTR 能建立一个存储器操作数，与其后的存储器操作数有相同的段地址和偏移量，但有不同的类型。PTR 仅仅为已分配存储器单元的操作数赋予另外的意思（PTR 不为存储器操作数分配内存单元）。例如：

```
INC     WORD PTR [BX]
ADD     BYTE PTR [SI], 4BH
```

汇编程序在汇编上面两个语句时，PTR 操作符指明其中由 BX 和 SI 寻址的存储器操作数的类型，以便能正确汇编出二进制目标码。前一条指令中 BX 间址的内存单元为字型（WORD）的，后一条指令中 SI 间址的内存单元是字节型（BYTE）的。

（2）THIS 操作符

THIS 操作符的格式如下：

```
THIS    类型（或属性）
```

THIS 可以像 PTR 一样建立一个指定类型（BYTE、WORD 或 DWORD）或指定距离（NEAR、FAR）的存储器地址操作数，但并不为其分配存储单元，所建立的存储器操作数的段地址和偏移地址与下一个存储单元地址相同。可以认为操作符 THIS 的主要作用就是给同一存储单元取一个别名，别名的类型与原名有所不同。例如：

```
FIRST     EQU  THIS  BYTE
SECOND    DW  100 DUP (?)
```

此时，FIRST 的偏移地址值与 SECOND 完全相同，但 FIRST 是字节型，SECOND 是字型。

4.3.2 运算符的优先级别

在使用以上 5 种类型的常用运算符或操作符计算表达式的值时，应按规定的优先级进行，程序设计者在写表达式时应注意。优先级别从高到低排序为：① 圆括号，LENGTH，SIZE；② PTR，OFFSET，SEG，TYPE，THIS；③ *，/，MOD；④ +，–；⑤ EQ，NE，LT，LE，GT，GE；⑥ NOT；⑦ AND；⑧ OR，XOR。

4.4 汇编语言程序设计

综合前面的知识，可以使用两种语句来设计一个汇编语言源程序。而程序设计首先要将问

题分解成一个一个的步骤，每步都可以用汇编语言中的指令性语句，按照先后顺序表达。

一个好的程序不仅要满足设计要求，能正常运行，实现预定功能，还应满足：① 结构化、简明、易读、易调试、易维护（修改和扩充）；② 执行速度快；③ 占用存储空间尽量少。

执行速度和占用存储空间两者有时是矛盾的，这两个指标往往不能同时满足，在许多情况下要加以权衡，看哪个指标对于程序设计更重要。对于较大的程序，如何使程序结构化、模块化，便于阅读、调试，以及与其他程序方便连接，则显得更加重要。

4.4.1 汇编语言程序设计基本步骤

汇编语言程序设计的步骤如下：

<1> 分析问题，抽象出问题的数学模型，确定解决问题的合理算法。

<2> 绘制流程图或写出程序步骤，可以从粗到细地把算法逐步具体化。

<3> 分配存储空间及工作单元，根据流程图编写程序。

<4> 静态检查，设计者仔细阅读所设计的程序，尽量找出诸如语法、逻辑等错误。

<5> 在计算机上调试程序。

在调试程序的过程中，读者应该善于利用计算机提供的软件调试工具，编程技巧是通过大量阅读程序和自己动手编制程序获得的。在汇编语言程序调试的集成环境中调试汇编语言程序更方便，具备编辑、汇编、连接、运行调试等一系列功能。这个集成环境还可以用不同颜色表示汇编语言语句中的不同部分，如指令性语句中的指令助记符用蓝色、寄存器名用红色等，如果操作码输入错误，便不会变成红色。这样可以提示设计者，在程序编辑过程中不易出现语法错误。

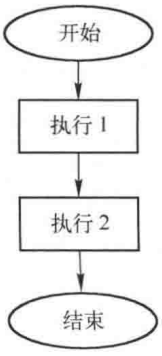


图 4-2 顺序结构程序流程

4.4.2 汇编语言程序的基本结构

汇编语言程序的基本结构形式有 4 种：顺序结构、分支结构、循环结构和子程序。

1．顺序结构

顺序结构程序一般是简单程序，它是顺序执行的，无分支、无循环，也无转移，因此也被称为直线程序。顺序结构程序流程如图 4-2 所示。顺序结构程序设计示例如下。

【例 4-1】 阅读下面的程序，指出程序的运行结果。

```
DATA    SEGMENT
BLOCK   DW      0ABCDH
BUFF    DD      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     DX, BLOCK
```

```
        MOV     AX, DX
        AND     AX, 0F0FH
        AND     DX, 0F0F0H
        MOV     CL, 4
        SHR     DX, CL
        LEA     BX, BUFF
        MOV     [BX+0], AL
        MOV     [BX+1], DL
        MOV     [BX+2], AH
        MOV     [BX+3], DH
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 该程序有两个程序段，以 DATA 为段名的数据段和以 CODE 为段名的代码段。② 数据段定义了一个字型的变量 BLOCK，并赋值为十六进制数（因该数以字母开头，故在数据前加上 0）；定义另一个变量 BUFF 为双字型，即保留 4 字节单元，用于存放结果。③ 程序的主要功能是将字型数据分成 4 部分，每部分都是 4 位二进制数，分别存储在 BUFF 缓冲区中。④ 程序最后采用 DOS 功能（后面将介绍）结束。⑤ 运行结果为：从存储器缓冲区 BUFF 开始，顺序存入 0DH、0CH、0BH 和 0AH。

2．分支结构

分支结构程序是指程序在按指令先后顺序执行的过程中，遇到不同的计算结果值，需要计算机自动进行判断、选择，以决定转向那个要执行的程序段。计算机的智能化、分析判断能力就是这样实现的。分支程序一般是利用比较、转移指令来实现的：用于比较、判断的指令有两数比较指令 CMP、串比较指令 CMPS、串搜索指令 SCAS，用于实现转移的指令有无条件转移指令 JMP 和各种类型的条件转移指令，它们可以互相配合实现不同情况的分支。多路分支情况可以采用多次判断转移的方法实现，每次判断转移形成两路分支，n 次判断转移形成 n+1 路分支，也可以利用跳转表来实现程序分支。

分支结构程序一般根据条件进行判别，满足条件则转移到标号所指示的程序部分执行，不满足条件，则执行下一条指令。

分支结构程序流程如图 4-3 所示。分支结构程序设计如下。

【例 4-2】 判断 MEMS 单元数据，将结果存入 MEMD 单元。

若数据>0，结果为 1；若数据<0，结果为–1；若数据=0，结果为 0。

```
MY_D    SEGMENT
MEMS    DB      08h
MEMD    DB      ?
MY_D    ENDS
MY_C    SEGMENT
```

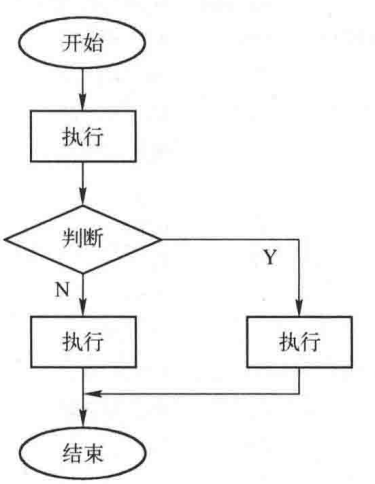


图 4-3　分支结构程序流程

```
        ASSUME  DS:MY_D, CS:MY_C
START:  MOV     AX, MY_D
        MOV     DS, AX
        MOV     AL, MEMS                ; 取数据进行判别
        CMP     AL, 0
        JGE     NEXT                    ; 大于等于 0，转移
        MOV     AL, -1                  ; 小于 0，结果为-1
        JMP     DONE
NEXT:   JE      DONE                    ; 为 0，则结果为 0
        MOV     AL, 1                   ; 否则，结果为 1
DONE:   MOV     MEMD, AL
        MOV     AX, 4C00H
        INT     21H
MY_C    ENDS
        END     START
```

说明：① 该程序是一个典型的分支结构程序，根据一个数据的 3 种情况，执行 3 个分支程序段。

② 程序转移的根据是与 0 比较，这是一个条件判断转移。由于参加比较的数据是带符号数，故使用“大于”“不大于”等条件判别，此处用“大于等于”即 JGE 为条件。

③ 由于一次分支只有两路，故从 JGE 判别分支后，有第二次判别分支，即 JE。

④ 由于本例中被测数据为正数，故结果为 1。

⑤ 若改变 MEMS 单元的数，可以得到不同的结果。

⑥ 对于多分支的程序，在上机调试时，对各分支程序段都应进行检验。

图 4-4 是上述程序的流程图。本例也可以用下面的程序实现。

```
MY_D    SEGMENT
MEMS    DB      08h
MEMD    DB      ?
MY_D    ENDS
MY_C    SEGMENT
        ASSUME  DS:MY_D, CS:MY_C
START:  MOV     AX, MY_D
        MOV     DS, AX
        MOV     AL, MEMS                ; 取数据进行判别
        CMP     AL, 0
        MOV     AL, -1
        JL      DONE                    ; 小于 0，转移
        MOV     AL, 1
        JG      DONE                    ; 大于 0，转移
        MOV     AL, 0                   ; 等于 0
DONE:   MOV     MEMD, AL
        MOV     AX, 4C00H
        INT     21H
MY_C    ENDS
        END     START
```

说明：① 程序转移的根据是与 0 比较，并对产生的不同情况，如小于 0、大于 0 和等于 0 的顺序进行判断测试。

② 先对 AL 赋值为-1，并按比较结果小于 0 判断，由于先送-1 给 AL，因此如果比较结果成立，就可以直接转到标号 DONE 指示的语句。如果比较结果不成立，那么对 AL 赋值为 1，并按比较结果大于 0 判断；如果比较结果成立，那么可以直接转到标号 DONE 指示的语句。如果比较结果不成立，那么对 AL 赋值为 0，接着执行标号 DONE 指示的语句。

③ 本程序是先对结果进行赋值，再按对应的条件进行判断实现转移。这种方法可以简化分支程序结构，降低其复杂度，提高其可阅读性。

④ 对 AL 的赋值不影响标志位，否则使用这种方法必须对标志位进行保护。

图 4-5 是上述程序的流程图。

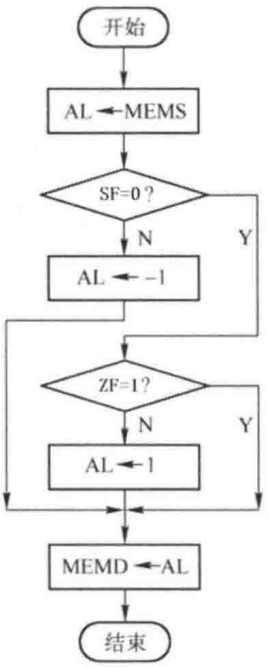


图 4-4　例 4-2 程序流程图（一）

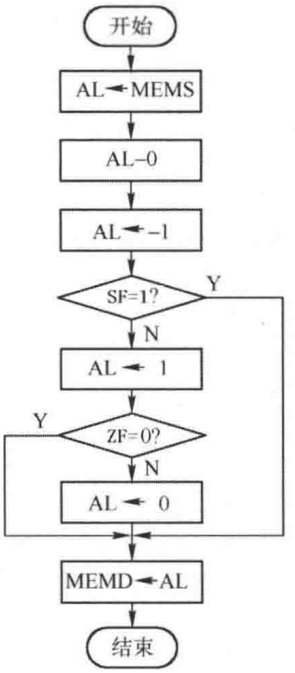


图 4-5　例 4-2 程序流程图（二）

3．循环结构

程序中的某些部分需要重复执行，设计者不可能将重复部分反复地书写，程序会显得冗长。只要选好参数，将程序中重复执行的部分构成循环结构，那么程序既美观又便于修改。

循环结构每次测试循环条件，当满足条件时，重复执行这一段程序；否则结束循环，顺序往下执行。由于循环程序需要循环准备、修改变量、结束控制等指令，执行的速度会稍慢些。

循环结构的程序流程图如图 4-6 所示。在循环程序中，循环控制有如下 3 种。

- ❖ 采用计数法（一般用减 1 计数）：当循环控制次数已知时，常用方法是将循环控制次数送到 CX 寄存器，做完一次循环，就利用 LOOP 指令减 1 计数，并判断循环是否结束。
- ❖ 比较条件结束：当循环控制次数未知时，采用此方法；满足比较条件，则结束循环，否则继续做循环操作。
- ❖ 设定标志结束：当循环套循环而循环次数未知时采用此方法，如设 0FFH 为结束标志。

循环结构程序设计例题见例 4-3。

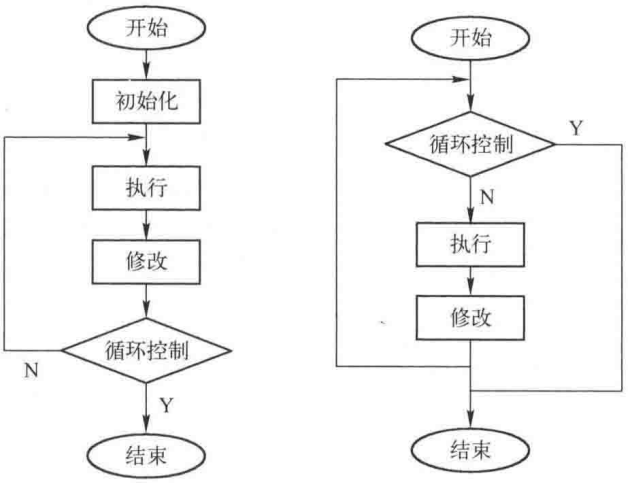


图 4-6　循环结构程序流程

【例 4-3】 编程统计 BUFF 缓冲区数据中负数的个数。

```
DATA    SEGMENT
BUFF    DB      67H, 9EH, -6AH, 0ABH, 6DH
MEM     DB      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     CX, 5                   ; 循环控制次数
        LEA     BX, BUFF                ; 设置缓冲区指针
        MOV     DL, 0                   ; 统计计数器清零
NEXT:   MOV     AL, [BX]                ; 取数据
        CMP     AL, 0                   ; 做运算，影响标志
        JNS     AA1                     ; 是正数，转移
        INC     DL                      ; 是负数，统计值加 1
AA1:    INC     BX                      ; 移动指针
        LOOP    NEXT                    ; 循环控制
        MOV     MEM, DL                 ; 保存统计结果
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 该循环程序属于图 4-6(a)所示结构。程序的初始化包括设置缓冲区指针、设定循环控制次数、统计计数器先清零。

② 执行部分从 BUFF 缓冲区取数，执行比较指令；判断符号标志 SF，是负数，则统计值加 1。

③ 修改部分含移动缓冲区指针，循环次数减 1。

④ 循环控制部分为 CX，内容不为 0，继续循环操作，否则脱离循环。

⑤ 结束处理部分将统计结果（DL 寄存器的内容）存入 MEM 单元并将控制交操作系统。

思考题：① 该例中的数据是字节型，若数据为字型，该如何修改程序？② 判断一个数是否为负数，还有其他方法吗？试举两种方法。③ 该程序循环中用了分支，是否可以不用分支而达到目的？如何修改程序？④ 该例中，循环控制次数直接用了已知的数据个数 5，采用这种方法，当数据个数变化时，程序必须做相应改变。能利用数据在内存单元的地址关系，自动计算数据个数作为循环控制吗？

【例 4-4】 编程统计 AX 寄存器中“1”的个数。

```
CODE    SEGMENT
        ASSUME  CS:CODE
START:  MOV     CX, 16                  ; 循环控制次数
        XOR     DL, DL                  ; 统计计数清零
        CMP     AX, 0                   ; AX 的内容为 0 吗
        JZ      DONE                    ; 是 0，结束循环
BB1:    SHL     AX, 1                   ; 否则移动 AX
        ADC     DL, 0                   ; 统计“1”的个数
        LOOP    BB1
DONE:   MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 该循环程序属于图 4-3(b)所示结构，程序的初始化包括设定循环控制次数、统计计数器先清零。

② 先判断循环是否继续，若 AX 内容为 0，则没有必要执行循环。

③ 执行部分采用移位操作，并用 ADC 指令统计 CF 的值，避免了使用程序分支。

【例 4-5】 在 BLOCK 内存区中有一串字符，试编程统计“%”之前的字符个数。

```
DATA    SEGMENT
BLOCK   DB      'ANDEPO139%WR'
COUNT   EQU     $-BLOCK
MEM     DB      0
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     SI, OFFSET BLOCK
        MOV     CX, COUNT
LOOP1:  MOV     AL, [SI]                ; 取字符
        CMP     AL, '%'                 ; 是“%”吗
        JZ      DONE                    ; 是，结束循环
        INC     BYTE PTR MEM            ; 否，统计值加 1
        INC     SI                      ; 移动指针
        LOOP    LOOP1                   ; 继续循环
DONE:   MOV     AX, 4C00H
        INT     21H
```

```
CODE    ENDS
        END     START
```

说明：① 字符变量的定义必须用 DB 伪指令，因为字符的 ASCII 值是 7 位二进制数。

② 本例采用内存地址关系（$–BLOCK）自动计算循环控制次数。

③ 本例的统计计数器直接在内存单元 MEM 中。

4．子程序

我们将一个具有特定功能的代码块定义为一个过程（或子程序）。过程可以与主程序在同一段，这时过程的属性为 NEAR，即主程序调用时只将 IP 寄存器的值入栈保存，然后把子程序的入口地址存入 IP 就实现了过程调用；当过程与主程序在不同段时，过程的属性为 FAR，主程序调用时，CS 和 IP 寄存器的值都要入栈保存。子程序的最后一条语句是 RET，执行该语句，返回地址从堆栈中弹出，控制返回到被调用处。

【例 4-6】 用子程序结构实现寄存器 AX 内容乘 10，结果仍在 AX 中。

```
CCC     EQU     1000
CODE    SEGMENT
        ASSUME  CS:CODE
START:  MOV     AX, CCC             ; 把 AX 赋值为 1000=03E8H
        CALL    MUL10               ; 调用把 AX 内容乘 10 子程序
        MOV     AX, 4C00H
        INT     21H

MUL10   PROC                        ; 乘 10 子程序，入口参数是 AX，出口参数是 AX
        PUSHF                       ; 保护现场，保护标志寄存器和 BX
        PUSH    BX
                                    ; 下面是功能程序段，实现 10*AX→AX
        ADD     AX, AX              ; 2XX→AX
        MOV     BX, AX              ; 2XX→BX
        ADD     AX, AX              ; 4XX→AX
        ADD     AX, AX              ; 8XX→AX
        ADD     AX, BX              ; 8XX+2XX→AX

        POP     BX                  ; 恢复现场
        POPF
        RET                         ; 返回主程序
MUL10   ENDP

CODE    ENDS
        END     START
```

说明：① 该程序只用了代码段，未用数据段，所以程序只对代码段进行了定义。

② 主程序对 AX 进行赋值，然后调用子程序 MUL10 对 AX 内容进行乘 10 的操作。

③ 子程序 MUL10 包括如下 5 部分：子程序功能说明、入口和出口参数说明、保护现场、实现具体操作的功能段程序及恢复现场。一个标准的子程序都应该具备这 5 部分。

④ 由于子程序使用了加法指令，它将影响标志寄存器；程序中还使用了 BX 作为中间变

量，所以在子程序保护现场部分，用 PUSH 指令把标志寄存器和 BX 推入堆栈，完成对这两个寄存器的现场保护。

⑤ 在功能程序段中，利用加法指令先后完成了 2 倍 XX 的操作和 8 倍 XX 的操作，然后把 2XX 和 8XX 相加实现了 10XX。

⑥ 在恢复现场部分，用 POP 指令从堆栈中推出两个数，分别送给 BX 和标志寄存器，完成了恢复现场的操作。要注意的是，现场恢复过程是按照先进后出的操作顺序进行的。

4.5 宏定义和宏调用

在程序设计中，有时可将一段具有特定功能的代码块定义为一个过程，使整个程序清晰、便于理解和调试，这就是过程或子程序。子程序仅编写一次，汇编后仅有一段代码，主程序可以多次调用它。由于有调用和返回的一系列操作，故执行的速度相对较慢。

在汇编语言源程序中，有的程序部分要多次使用，为了在源程序中不重复书写这一部分，可以用一条宏指令代替。这条宏指令通过宏定义，再经过宏汇编产生所需代码序列，然后将这些代码序列嵌在调用处。与过程调用不同，它不使用堆栈，仅仅减少程序的书写，每调用一次，程序代码就会嵌入一次。

1．宏定义

宏定义是用一组伪操作来实现的，其格式如下：

```
宏指令名    MACRO <形式参数表>
            …
            ENDM
```

MACRO 和 ENDM 是一对伪操作，在这对伪操作之间是宏定义体，即用宏指令代替的程序部分，是一组有独立功能的程序段，是由指令性语句和指示性语句所构成的。“宏指令名”给出该宏定义的名称，调用时就使用“宏指令名”来调用该宏定义。“宏指令名”是由程序设计者设定的，必须以字母开头，后面可跟字母、数字或下画线。“形式参数表”给出宏定义中所用到的形式参数（或称为哑元、虚参），每两个参数之间用“,”隔开。

2．宏调用

经宏定义后的宏指令就可以在源程序中被调用。为了与子程序调用区分，用宏指令的调用被称为宏调用。其格式如下：

```
            宏指令名  实元表
```

这里，实元表中的每一项为实际参量值，即实元。实元之间用“,”隔开。

当源程序被汇编时，汇编程序将对每个宏调用进行宏展开。宏展开就是用宏定义体取代源程序中的宏指令名，且用实际参量值取代宏定义中的形式参数，即实元取代哑元。在取代时，实元和哑元是一一对应的，即第一个实元取代第一个哑元，第二个实元取代第二个哑元，以此类推。8086 宏汇编并不要求实元的个数和哑元的个数相等，若调用时实元个数多于哑元个数，则多余的实元被忽略；反之，若哑元个数多于实元个数，则多余的哑元变为“空”。

例如，用宏指令定义两个带符号的字节型变量操作数相乘，乘法结果为字型。

```
MULTY   MACRO   OPR1, OPR2, RESULT
        PUSH    AX
        MOV     AL, OPR1
        MOV     BL, OPR2
        IMUL    BL
        MOV     RESULT, AX
        POP     AX
        ENDM                                    ; 宏定义结束

DATA    SEGMENT
XX      DB      4EH
YY      DB      8AH
ZZ      DW      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MULTY   XX, YY, ZZ                      ; 宏调用，用实元代替哑元
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

宏调用是在汇编期间展开的，因此事实上的程序代码如下：

```
DATA    SEGMENT
XX      DB      4EH
YY      DB      8AH
ZZ      DW      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX

        PUSH    AX                              ; 宏展开
        MOV     AL, XX
        MOV     BL, YY
        IMUL    BL
        MOV     ZZ, AX
        POP     AX
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

每调用一次，就把宏定义体展开一次，所以程序占用的存储空间与调用次数有关，调用次数越多，占用的存储空间也越大。因此，宏指令的使用简化了源程序，但并不节省目标程序所

占用的内存单元。子程序是在程序运行期间由主程序调用的，在目标代码中，它只占用自身大小的内存空间，无论子程序被调用多少次，主程序中只有调用指令的目标代码。用调用子程序的方法，汇编后产生的目标代码少，节约了存储空间。

子程序在执行时，每被调用一次，都要先保护断点，通常在程序中还要保护现场；返回时先恢复现场，再恢复断点，然后返回。这些操作增加了额外的时间，因此执行时间长，速度相对慢。用宏定义和宏调用可以免去执行时间上的这些额外开销，所以生成的目标代码执行时间短，速度相对快。

宏指令可以带哑元，调用时用实元取代，这避免了子程序中因变量传送带来的麻烦。宏调用中的实元可以是常数、寄存器、存储单元名和用寻址方式能够找到的地址或表达式，还可以是指令的操作码或操作码的一部分等，使编程增加了灵活性。这一特性是子程序所不及的。

4.6 汇编语言程序设计与上机调试

前面介绍了编写汇编语言程序所需要掌握的基本知识，如基本语句、格式，以及如何用伪指令定义变量、常数和定义程序段等，也根据内容需要给出了简单例题。下面从问题入手，设计汇编语言程序，编写一个汇编语言程序的框架。

4.6.1 汇编语言程序设计实例

汇编语言精练、简洁、快速、面向机器编程，特别适合要求执行速度快的场合，尤其是对输入、输出设备的控制，其效率是一般高级语言无法比拟的。

1．算术运算例题

汇编语言一般不适合编写复杂的算术运算，但对于简单的加、减、乘、除运算还是可行的。

【例 4-7】 在偏移首地址为 ARRAY 的内存储区有 100 个字型数据，要求将数组的每个元素加 1。试编写汇编语言程序完成该要求。

```
DATA    SEGMENT
ARRAY   DW      100 DUP (?)                ; 定义 100 个字型随机数
DATA    EDNS
CODE    SEGMENT
        ASSUME  DS:DATA, CS:CODE
START:  MOV     AX, DATA
        MOV     DS, AX
        LEA     BX, ARRAY                  ; 设数组首地址指针
        MOV     CX, LENGTH ARRAY           ; 数组数据长度
AA1:    INC     WORD PTR [BX], 1           ; 指定为字型数加 1
        ADD     BX , 2                     ; 移动地址指针
        LOOP    AA1                        ; 循环操作
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 程序设计了一个数据段，段名为 DATA，段内定义了一个名为 ARRAY 的变量，其类型属性用 DW 伪指令定义为字型，并将其赋值为 100 个随机数。

② 段说明语句一般写在代码段中，把段名与段寄存器联系起来，仅供汇编时使用，而段寄存器中的值是由 CPU 执行指令时给定的。

③ INC 指令让内存单元数值加 1（不影响 CF 标志），由于直接对存储单元进行运算，故采用伪指令中的综合运算符 PTR 来确定存储单元的类型属性。

④ 本例中声明了字型数据，即每个数组元素占两个存储单元。因此，在指定下一个数据时，地址指针要移动 2 字节。

思考题：例题中没有考虑进位，若考虑进位，程序要做哪些修改？

【例 4-8】 有一个 100 个元素的 BCD 数组，编写程序对数组元素进行求和。

```
DATA    SEGMENT
XBCD    DB      12H, 34H, …, 98H          ; 定义 100 个 BCD 数
YBCD    DW      ?                         ; 求和结果放在变量 YBCD 中
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        LEA     BX, XBCD                  ; 设置寄存器 BX 为数组 YBCD 地址指针
        MOV     CX, 100                   ; 数组长度送给寄存器 CX
        MOV     AX,0                      ; 计算结果在 AX 中，初始值为 0
AGAIN:  ADD     AL, [BX]                  ; AL+一个数值元素→AL
        DAA                               ; 十进制调整
        XCHG    AL, AH                    ; AL 与 AH 交换
        ADC     AL, 0                     ; AL + CF→AL
        DAA                               ; 十进制调整
        XCHG    AH, AL                    ; AL 与 AH 交换
        INC     BX                        ; 指针加 1
        LOOP    AGAIN                     ; 循环
        MOV     YBCD, AX                  ; 计算结果存入变量 YBCD 中
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 本程序设计了一个数据段，段名为 DATA，段内定义了一个名为 XBCD 的数组，数组长度为 100，数组元素为两位的 BCD 数。段内还定义了一个变量 YBCD，用于存放数组元素的求和。

② 指令“LEA　BX, XBCD”可以用指令“MOV　BX, OFFSET XBCD”代替。

③ 从指令语句“ADD　AL, [BX]”到指令语句“XCHG　AH, AL”实际上是实现 AX 加一个数组元素的功能，参与运算的数为 BCD 类型。由于 DAA 指令只能对 AL 进行调整，因此程序要采用以 AL 作为目的操作数的八位加法指令。

【例 4-9】 有两个无符号字节型数组，设数组元素个数相等，编程将数组中的对应元素相加，结果存入另一个内存区。

```
DATA    SEGMENT
M1      DB      20  DUP (?)
M2      DB      20  DUP (?)
M3      DW      20  DUP (0)
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        LEA     SI, M1                  ; 设数组 1 的地址指针
        LEA     DI, M2                  ; 设数组 2 的地址指针
        LEA     BX, M3                  ; 设结果区的地址指针
        MOV     CX, 20
AA1:    MOV     AL, [SI]
        ADD     AL, [DI]
        MOV     [BX], AL
        ADC     BYTE PTR [BX+1], 0      ; 保存结果的进位
        INC     SI
        INC     DI
        ADD     BX, 2
        LOOP    AA1
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：本例考虑了两数相加时产生的进位问题，在设置结果存放区时用了 DW 定义，且将其清零，直接将进位加进去。

思考题：当两个数组的元素个数不同时，如何编写或修改程序？

2．逻辑处理题

【例 4-10】 将寄存器 AL 中的高、低 4 位交换。

```
CODE    SEGMENT
        ASSUME  CS:CODE
START:  MOV     AL, 0ABH
        MOV     CL, 4
        ROL     AL, CL                  ; 移出位补充移空位 4 次
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

【例 4-11】 将 AX 中的内容按相反顺序存入 BX。

```
CODE    SEGMENT
        ASSUME  CS:CODE
START:  MOV     AX, 1234H
        MOV     CX, 16
```

```
AA1:    SHL     AX, 1                          ; 移出的位进到 CF
        RCR     BX, 1                          ; AX 中移出的位进入 BX
        LOOP    AA1
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 本例中，利用移位指令中移出位进入 CF 标志位的特性，先把 AX 中的最高位（左移）或最低位（右移）移至 CF，再带进位移动 BX，若 AX 右移，则 BX 左移；反之亦然。

② 本例的结果是将 AX 中的 1234H 按相反顺序存入，故 BX 中的内容为 2C48H。

3．代码转换例题

【例 4-12】 编程将以“$”结束的字符串中的小写字母改为大写字母。

```
DATA    SEGMENT
STR     DB      'heLLo,eveRyboDY !', '$'
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        LEA     BX, STR
A1:     MOV     AL, [BX]
        CMP     AL, '$'                 ; 是“$”，则结束
        JE      DONE
        CMP     AL, 'a'
        JB      NEXT                    ; 低于，则为大写字母
        CMP     AL, 'z'
        JA      NEXT                    ; 高于，则不是字母
        SUB     AL, 20H                 ; 将小写字母改为大写字母
        MOV     [BX], AL
NEXT:   INC     BX
        JMP     A1
DONE:   MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 大写字母和小写字母的 ASCII 值分别为：'A'=0100 0001（41H），'a'=0110 0001（61H），'Z'=0101 1010，'z'＝0111 1010。从大、小写字母的二进制表示 ASCII 值看，仅有 D_5 位不同，即小写字母比大写字母 ASCII 值大 20H。这样它们之间的转换就变得很简单。

② 此处循环是通过判断字母是否为'$'而结束的。

③ 程序中有多个分支，用来判别字符是否为大写字母或根本不是字母。

【例 4-13】 十六进制数到十进制数的转换。

十六进制数有 16 个计数符号：0～9 和 A～F。人们习惯 0～9 的十进制表示方法，而对于十六进制数中的字母总是感到别扭。前面谈到，任意进制数到十进制数的转换可以按位权展

开，但是用这种方法编程会遇到一些困难。我们从所学过的指令 AAM 中得到一些启发，AAM 指令是对两个非压缩的 BCD 码相乘后产生的十六进制结果进行调整，调整方法就是将乘法结果除 10，其商为十进制数的十位，余数则是个位；同理，对于 3 位十进制数，若除 10，余数则是十进制数个位，对商再除 10，则余数为十进制数的十位。其余以此类推。

```
DATA    SEGMENT
YY      DB      5 DUP (0)
DATA    ENDS                                ; 设置数据段
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     AX, 4B6CH
        MOV     BX, OFFSET YY
        MOV     CX, 10
A1:     MOV     DX, 0                       ; 被除数扩展为 32 位
        DIV     CX
        MOV     [BX], DL                    ; 将转换好的数存入堆栈
        INC     BX
        OR      AX, AX                      ; 转换，直到商为 0
        JNZ     A1
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 本例设置了数据段，其中建立了名字为 YY 的数组，用于存放转换结果。

② 除法运算中，若除数为 16 位数，被除数必须为 32 位数。本例中将被除数的高 16 位（DX）扩展为 0，故程序仅能对无符号数进行转换。

③ 本例为循环结构，在循环体最后是循环结束判断，因此要求转换的二进制不能为 0。

④ 转换结果为 19308（4B6CH）。

【例 4-14】 十进制数到十六进制数的转换。

设在名字为 XBCD 的数组中，存放 3 个扩展的 BCD 数 8、7、5，代表 3 位十进制数 578。编写程序，把该值转换为十六进制数，并把结果放在变量 YH 中，YH=5×100+7×10+8。

```
DATA    SEGMENT
XBCD    DB      08, 07, 05                  ; 设置 BCD 数
YH      DW      ?                           ; 转换结果存入 YH
SST     ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     AL, XBCD+2                  ; 取百位数
        MOV     BL, 100
        MUL     BL                          ; 百位数乘 100
        MOV     DX, AX                      ; 结果暂存在 DX 中
```

```
        MOV     AL, XBCD+1                      ; 取十位数
        MOV     BL, 10
        MUL     BL                              ; 十位数乘 10
        ADD     AX, DX                          ; 十位数乘 10 + 百位数乘 100→AX
        ADD     AL, XBCD                        ; AX+XBCD→AX
        ADC     AH, 0
        MOV     YH, AX
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 语句“MOV　AL, XBCD+2”的功能是取数组 XBCD 中第 2 个元素，即取百位数。本例中的百位数是 5，所以此语句被执行后，AL 内容为 5。如果认为“MOV　AL, XBCD+2”是把 XBCD 的内容加 2，那么结果送给 AL，则是错误的。XBCD+2 是表达式，不能对寄存器和内存进行操作。

② 语句“ADD　AL, XBCD”“ADC　AH, 0”“MOV　YH, AX”的功能是实现 AX 中的 16 位数与字节类型的个位数 XBCD 相加。

③ 转换结果为 0242H。

4．表格处理题

当一个信息与另一个信息之间不能用数学公式来描述其关系时，人们就用一种映射方法来表明它们的关系。在计算机编程中，这种映射是通过表格实现的。例如，8086 系统的中断矢量表中，无论是内部中断还是外部中断，都是通过提供索引号（中断类型号）与该表取得联系的，从而找到中断服务程序的地址，去执行中断服务。

计算机中的表格建立在内存区。因为内存地址是连续而有规律的，所以映射关系往往通过查找内存地址来完成。

【例 4-15】 编写程序，将 0～9 的数字转换成所要求的密码。

明码和密码的映射关系为：

明码	0	1	2	3	4	5	6	7	8	9
密码	2	7	1	6	9	0	8	3	4	5

```
DATA    SEGMENT
TABLE   DB      2, 7, 1, 6 ,9, 0, 8, 3, 4, 5        ; 密码表
VAL1    DB      8, 4, 1, 7, 5                       ; 被加密码（明码）
VAL2    DB      5  DUP(?)                           ; 密码
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     SI, OFFSET VAL1
        MOV     DI, OFFSET VAL2
        MOV     CX, 5
        LEA     BX, TABLE                           ; 表格首地址指针
```

```
A1:     MOV     AL, [SI]                    ; 取明码
        XLAT                                ; 转换为密码
        MOV     [DI], AL
        INC     SI
        INC     DI
        LOOP    A1
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 表格建立在偏移首地址为 TABLE 的内存区；表中每字节为一项，其内容为数字 0～9 对应的密码。

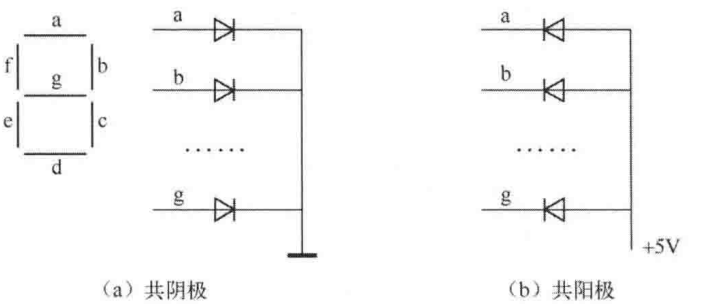
② XLAT 换码指令可以说是一种复合指令，即不能单独使用。在使用该指令之前，必须先将表格首地址送 BX，被转换的码送 AL。

思考题：要对加密过的数据进行解密，如何修改程序？

【例 4-16】 七段代码转换。

七段发光二极管可以作为简单的输出设备，能显示 0～9、A～F 和一些基本符号。控制它显示的代码与所显示的符号之间没有数学联系，而与所选择的器件以及与计算机的连接方式有关。因此，首先要找出控制代码和显示字符之间的映射关系，通过代码转换来完成。

七段发光二极管有 a、b、c、d、e、f、g 等七段，并有共阳极和共阴极之分，控制二极管导通所需的电平不同，如图 4-7 所示。



（a）共阴极　　（b）共阳极

图 4-7　七段发光二极管

七段发光二极管作为计算机的输出设备，所显示的字符是由计算机输出的。设七段发光二极管的 a 段由计算机的 D0 位控制，b 段由计算机的 D1 位控制，以此类推。若计算机输出 3EH，即二进制 00111110，当选用共阳极管时（计算机的输出控制二极管阴极），由“0”电平控制二极管点亮，所显示的是 a 段和 g 段；而选用共阴极管时（计算机的输出控制二极管阳极），由“1”电平控制二极管点亮，所显示的是 b、c、d、e、f 段。

下面选用共阴极七段发光二极管和由 D0 位控制 a 段，其余以此类推的连接方式，设 D7=0。找出代码对应关系，建立 0～9 的数字与七段代码控制转换表。照此，程序的设计也就很容易了，读者可参考例 7-1 自行编制。

0	1	2	3	4	5	6	7	8	9
3F	06	5B	4F	66	6D	7D	07	7F	6F

5．串指令题

【例 4-17】 将偏移地址为 STG1、长度为 COUNT 的数据块，传送到偏移地址为 STG2 的内存区中。

```
DATA    SEGMENT
STG1    DW      25  DUP (?)
STG2    DW      25  DUP (?)
COUNT   EQU     25
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DARA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        MOV     CX, COUNT
        LEA     SI, STG1
        LEA     DI, STG2
        CLD
        REP     MOVSW
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 使用串指令前先做一些准备，如设置源地址指针和目的地址指针、数据长度、方向标志 DF 等。

② 串指令的寻址方式为源地址用 DS:SI，而目的地址用 ES:DI。在本例中，源变量和目的变量都定义在同一数据段，因而 ES 和 DS 具有同样的段基址。

③ 因为变量是字型的，所以串指令用 MOVSW。

思考题：若串传输中的源地址和目的地址有重叠，即将 STG1 中的数据块送到 STG1+10H 的内存区，如何修改程序？

【例 4-18】 将 BLOCK 内存区的带符号字节型数据按正数、负数分开，并分别存入 BUFF1 和 BUFF2 中。

```
DATA  SEGMENT
BLOCK   DB      60  DUP (?)
BUFF1   DB      60  DUP (?)
BUFF2   DB      60  DUP (?)
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        LEA     SI, BLOCK               ; 数据区
        LEA     DI, BUFF1               ; 正数缓冲区
        LEA     BX, BUFF2               ; 负数缓冲区
```

```
        MOV     CX, 60
        CLD
LOP:    LODSB                           ; 将 SI 指示的源数据取到 AL
        TEST    AL, 80H                 ; 测数据的最高位
        JNZ     FU                      ; 测试结果不为 0，即负数
        STOSB                           ; 否则，存入正数区
        JMP     AGAIN
FU:     XCHG    BX, DI                  ; 交换目的地址
        STOSB
        XCHG    BX, DI                  ; 还原目的地址
AGAIN:  LOOP    LOP
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 在串指令中，目的地址指针要用 ES:DI。由于本例中有两个目的地址，而寄存器 DI 仅有一个，因此使用 XCHG 交换指令。

② 这是一个标准的二分支程序（只考虑正、负），故不要忘了使用 JMP 指令。

6．按数据大小排序题

【例 4-19】 编程实现，从一串带符号字型数据中找出最大值。

```
DATA    SEGMENT
BLOCK   DW      762EH, 6A8BH, -664AH, 0B945H, -85DH
COUNT   EQU     ($-BLOCK)/2
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        LEA     SI, BLOCK
        MOV     CX, COUNT
        MOV     AX, [SI]                ; 取第 1 个数
        DEC     CX                      ; 准备与下个数比较
AA1:    ADD     SI, 2
        CMP     AX, [SI]                ; 与下个数比较
        JG      AA2                     ; 前一个数大，则保留，下次再比
        MOV     AX, [SI]                ; 否则，取后一个数，下次再比
AA2:    LOOP    AA1
        MOV     BX, AX
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 在数据串中找最大值、最小值是数据排序中较容易编程实现的，因为两两比较中，只要留下大（或小）的数，另一个数可以不管。

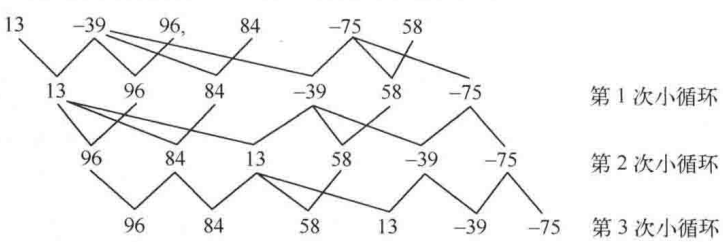
② 由于带符号数在计算机中用其补码表示，故带符号数的比较对 FR 寄存器中标志的影响，要用大于（JG）/不小于等于（JNLE）、小于 JL/不大于等于（JNGE）等。

③ 将本例改成找最小值，则容易实现。

【例 4-20】 编程实现，将一个带符号字节型数据组中的数据按从大到小的顺序排列。

```
DATA    SEGMENT
BUFF    DB      13, -39, 96, 84, -75, 58
COUNT   EQU     $-BUFF
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     CX, COUNT-1
LOOP1:  MOV     DX, CX                  ; 保存循环次数
        MOV     SI , 0                  ; 采用变址寻址
LOOP2:  MOV     AL, BUFF[SI]
        CMP     AL, BUFF[SI+1]          ; 前数与后数比较
        JGE     COT                     ; 前一个数大（或相等），则跳转
        XCHG    AL, BUFF[SI+1]          ; 否则交换内存位置
        MOV     BUFF[SI], AL
COT:    INC     SI
        LOOP    LOOP2                   ; 所有数据排列一次
        MOV     CX, DX                  ; 开始下一次排序
        LOOP    LOOP1
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 本例排序采用“冒泡法”，将数据两两比较。前一个数较大，则不改变原位置，否则两数交换，依次将全部数据排序一次（称为小循环）。



② 由于数据的原始排列情况未知，按以上方法排序一次，不一定符合要求，如何才能知道已经排序成功呢？本例采用多次小循环方法，用数据个数控制小循环次数。这种方法效率低，特别是在数据个数很多的情况下。

③ 实际上，本例仅需 3 次小循环就可以排序好，因此程序可以修改，使得程序执行效率更高。

4.6.2 DOS 功能调用和子程序设计

DOS 是美国 Microsoft 公司为 IBM PC 研制的磁盘操作系统（Disk Operating System），也称为 IBM-DOS 或 MS-DOS。DOS 不仅为用户提供了许多操作命令，还可以直接调用的上百个常用子程序。对这些子程序的调用，称为系统功能调用。这些子程序的功能主要是进行磁盘读/写、控制管理、内存管理、基本输入/输出管理等。在使用时，用户不需了解各种 I/O 接口硬件的详细情况，就能直接完成对 I/O 接口的控制和管理。为了使用方便，DOS 将所有子程序从 1 号开始顺序编号，这些编号称为 DOS 功能调用号。其调用过程为：

<1> DOS 功能调用号送 AH 寄存器。

<2> 如果需要，按要求给定输入参数（有的不需要输入参数）。

<3> 写入中断指令“INT　21H”。

<4> 调用结束，按功能使用其输出参数。

调用格式如下：

```
        …                               ; 设置输入参数
        MOV     AH, 功能调用号          ; 设置功能调用号
        INT     21H                     ; 调用
```

1．单字符输入——1 号

功　　能：从键盘输入 1 个字符。

输入参数：无。

输出参数：AL=ASCII 值。

2．单字符输出——2 号

功　　能：在屏幕上显示 1 个字符（ASCII 值）。

输入参数：DL=ASCII 值。

输出参数：无。

【例 4-21】 从键盘输入两个 1 位十进制数，求两数之和且在屏幕上显示结果。

```
CODE    SEGMENT
        ASSUME  CS:CODE
START:  MOV     AH, 1                   ; DOS 调用输入第一个数
        INT     21H
        MOV     BL, AL                  ; 保存输入的第一个数
        MOV     AH, 1
        INT     21H
        ADD     AL, BL                  ; 两个 ASCII 值相加
        AAA                             ; 调整加法结果为非压缩 BCD 数
        MOV     DL, AL
        ADD     DL, 30H                 ; 加法结果转换成 ASCII 值
        MOV     AH, 2                   ; DOS 调用输出到屏幕
        INT     21H
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
```

```
        END     START
```

说明：① 从键盘输入的字符在计算机的寄存器或内存单元中是以字符对应的ASCII值（即二进制数）存放的。所以，从键盘输入的数据并不是数据本身，需要进行ASCII值到十六进制数的转换。

② 同样，计算结果输出在屏幕上，也得先将其数据转换成ASCII值。

思考题：本例中只考虑加法结果为1位的十进制数，若结果为2位，如何修改程序？

3．多字符输入——0AH

功　　能：多个字符输入到缓冲区。

输入参数：DS:DX=输入缓冲区首地址。

输出参数：DS:DX=输入字符串所在缓冲区地址。

设置缓冲区要注意以下几点：

- 缓冲区第1字节存放预定字符个数，最多255个。
- 缓冲区第2字节保留，用于调用返回时存放实际输入的字符个数。
- 缓冲区第3字节开始，存放输入的字符。
- 缓冲区要考虑留1字节作为回车符。

4．多字符输出——9号

功　　能：多个字符输出到屏幕显示。

输入参数：DS:DX=输出字符缓冲区首地址。

输出参数：无。

该功能对输出字符的个数没有要求，但输出字符串要以'$'结束。

【例4-22】 在屏幕上显示一串字符。

```
DATA    SEGMENT
BUFF    DB      'How do you do? ', 0DH, 0AH, '$'
DATA    ENDS
CODE    SEGMENT
        ASSUME  DS:DATA, CS:CODE
START:  MOV     AX, DATA
        MOV     DS, AX
        LEA     DX, BUFF
        MOV     AH, 9
        INT     21H
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

说明：① 字符定义要使用DB伪指令，字符用单引号括起来。

② 本例中的0AH、0DH为回车和换行符号的ASCII值。

5．单字符输入/输出——6号

6号DOS功能实际上是1号和2号功能的组合。当DL寄存器的内容为0FFH时，6号功

能与 1 号功能同，即从键盘输入单个字符；当 DL 寄存器中放入字符的 ASCII 值时，就是 2 号功能。

6．过程终止——4CH 号

过程终止调用的功能是结束当前程序，并且返回调用它的程序。如果在 DEBUG 状态下运行，那么返回 DEBUG；如果在 DOS 下运行，那么返回 DOS。在汇编语言程序结束处加上“MOV AX, 4C00H”和“INT　21H”两条指令，以利于程序执行完毕返回操作系统控制。

7．子程序设计

子程序设计思想与主程序设计思想没有什么区别。若程序设计中采用调用子程序的方式，则使程序阅读起来清晰。因为一个子程序完成一个功能，使得程序调试也方便。子程序不能单独运行，仅可以提供给主程序调用。设计子程序要提供以下信息：子程序的功能，调用时需要的输入参数（入口参数），调用后提供的输出参数（出口参数），子程序中所使用的寄存器（以便调用前做必要的保护）。

【例 4-23】 用子程序结构编写程序统计 BX 和 DX 中 1 的个数，结果分别放在 CL 和 CH 中。

```
CODE     SEGMENT
         ASSUME  CS:CODE
START:   MOV     BX, 1234H                ; 设置子程序入口参数
         CALL    COUNT                    ; 统计 BX 中 1 的个数
         MOV     CL, AL
         MOV     DX, 8432H
         MOV     BX, DX                   ; 把 DX 内容传送给 BX，设置子程序入口参数
         CALL    COUNT                    ; 统计 DX 中 1 的个数
         MOV     CH, AL
         MOV     AX, 4C00H
         INT     21H
; 子程序 COUNT 的功能是统计 BX 中 1 的个数
COUNT    PROC                             ; 入口参数为 BX，出口参数为 AL
         PUSH    CX                       ; 保护 CX
         PUSHF                            ; 保护标志寄存器
         MOV     AL, 0                    ; 寄存器 AL 清零
         MOV     CX,16                    ; 设置循环次数
COUNT1:  ROR     BX, 1                    ; 循环右移 BX，移出位进入进位标志 CF
         JNC     COUNT2                   ; 检测 CF
         INC     AL                       ; CF 为 1，AL 加 1
COUNT2:  LOOP    COUNT1                   ; 循环
         POPF                             ; 恢复标志寄存器
         POP     CX                       ; 恢复 CX
         RET
         COUNT   ENDP                     ; 子程序结束
CODE     ENDS
         END     START
```

说明：① 本例利用子程序 COUNT 统计了两个寄存器 BX 和 DX 中 1 的个数。

② COUNT 子程序开始的注释中给出了 COUNT 子程序的功能和入口出口参数，这是编

写子程序应该包含的部分，希望读者注意。

③ 由于子程序使用了寄存器 CX 和标志寄存器，所以子程序对这两个寄存器进行了保护。另外，由于子程序使用了循环右移指令，16 次逐位移动后，寄存器 BX 内容不变，所以子程序不必对 BX 进行入栈和出栈的保护。

④ 希望读者考虑，为什么子程序没有对寄存器 AX 进行保护？

【例 4-24】 编写一个子程序，完成一个 2 位十六进制数到 ASCII 值的转换。

子程序名：CONHA。

功　　能：将 2 位十六进制数转换成 ASCII 值。

输入参数：AL=待转换的数。

输出参数：BX=转换好的 ASCII 值。

使用寄存器：AL、AH、BX、CL。

```
CODE    SEGMENT
        ASSUME  CS:CODE
CONHA   PROC    FAR
        MOV     AH, AL                  ; 保存待转换的数
        AND     AL, 0FH                 ; 处理十六进制数低位
        CMP     AL, 0AH                 ; 是十六进制数中的字母吗
        JB      ASC1                    ; 否，转移
        ADD     AL, 07                  ; 是，先加 7
ASC1:   ADD     AL, 30H                 ; 转换为 ASCII 值
        MOV     BL, AL                  ; 保存转换好的低位
        MOV     CL, 4                   ; 移位控制
        SHR     AH, CL                  ; 将原数右移，处理高位
        CMP     AH, 0AH                 ; 是十六进制数中的字母吗
        JB      ASC2                    ; 否，转移
        ADD     AH, 07                  ; 是，先加 7
ASC2:   ADD     AH, 30H
        MOV     BH, AH                  ; 保存转换好的高位
        RET
CONHA   ENDP
CODE    ENDS
        END
```

说明：① 该子程序设计为独立段，属性为 FAR。

② 程序最后有 3 条结束语句，ENDP 为过程结束，ENDS 为段结束，而 END 为汇编模块结束，所以本程序可以单独编辑、汇编，但要与主程序连接在一起才能运行。这还需要读者具备多个模块设计的一些知识，即几个伪指令。

【例 4-25】 用子程序调用形式，编写从键盘输入 4 位十六进制数的程序。

子程序名：ZH。

功　　能：检查键盘输入错误，将输入值转换成十六进制数。

输入参数：AL。

输出参数：AL。

使用寄存器：BX 和 CX。

```
CODE    SEGMENT
        ASSUME  CS :CODE
START:  MOV     CX, 4                   ; 输入 4 次
        MOV     BX, 0                   ; 用 BX 保存输入数据
RE1:    MOV     AH, 1                   ; 从键盘输入
        INT     21H
        CALL    ZH                      ; 通过子程序转换
        SHL     BX, 1                   ; 组合成十六进制数
        SHL     BX, 1
        SHL     BX, 1
        SHL     BX, 1
        ADD     BL, AL
        LOOP    RE1                     ; 循环输入 4 个数
        MOV     AX, 4C00H
        INT     21H
ZH      PROC
        CMP     AL, '9'                 ; 将 ASCII 值转换为十六进制数
        JBE     A2                      ; 输入值≤'9'（'0'~'9'），则减 30H
        CMP     AL, 'a'                 ; 输入值<'a'（'A'~'F'），则减 37H
        JB      A1
        SUB     AL, 20H                 ; 值为'a'~'f'，则先减 20H，再减 37H
A1:     SUB     AL, 7
A2:     SUB     AL, 30H
        RET
ZH      ENDP
CODE    ENDS
        END     START
```

说明：① 输入的十六进制符号可能有 3 种：'0'～'9'、'A'～'F'和'a'～'f'。子程序的功能是判断这 3 种情况，并且将其转换为对应的十六进制数。

② 分析字符的 ASCII 值之间的关系，有利于程序设计。'0'～'9'的 ASCII 值为 30H～39H，'A'～'F'的 ASCII 值为 41H～46H，'a'～'f'的 ASCII 值为 61H～66H。由此可知，'9'与'A'的 ASCII 码值相差 7，'A'与'a'的 ASCII 值相差 20H。

③ 在主程序中，当从键盘输入一个值后，立即调用子程序将其转换成十六进制数，并且存入 BX 寄存器。

④ 主程序和子程序在同一段，在“CODE　SEGMENT”和“CODE　ENDS”之间。整个程序的结束是在“MOV　AX, 4C00H”和“INT　21H”处。

4.6.3　汇编语言程序上机调试

编写好的汇编语言源程序需要在计算机上进行调试，以验证其正确性。首先，在进行调试的机器上必须具备工作环境，即要有进行编辑、汇编、连接和调试汇编程序的应用程序。8086 仿真软件很多，其中常用的有 emu8086。emu8086 是一个调试汇编语言源程序的集成环境，可以在 Windows 操作系统下直接运行，而不需转到 DOS 操作系统。emu8086 提供颜色确认，使

程序设计者一般不会出现汇编语言语法错误，如 8086 的指令助记符用蓝色、寄存器名用红色显示等。emu8086 可以一次性完成汇编语言程序从编辑到调试的全过程。

汇编语言程序上机调试步骤的流程如图 4-8 所示。

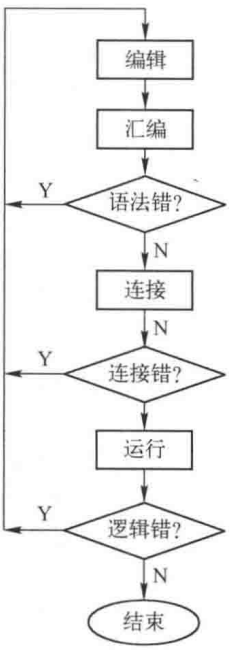


图 4-8　汇编语言程序上机调试步骤流程图

习 题 4

1．编写 8086 汇编语言程序，将寄存器 AX 的高 8 位送到寄存器 BL，AX 的低 8 位送到寄存器 DL。

2．将 DX 寄存器的内容按照从低位到高位的顺序分成 4 组，且将各组数分别送到寄存器 AL、BL、CL 和 DL 的低 4 位。

3．判断 MEM 单元的数据的奇偶，编程将奇数存入 MEMA 单元，将偶数存入 MEMB 单元。

4．试统计 9 个数中的偶数的个数，并将结果在屏幕上显示。

5．试将一串 16 位无符号数加密，加密方法是每个数乘 2。

6．根据 DL 寄存器的 D_3 位，完成两个压缩 BCD 数 X 和 Y 的加减运算，并将结果存入寄存器 DX。D_3 =0，做加法运算；D_3 =1，做减法运算。

7．在 BLOCK1 和 BLOCK2 两个单元数据中，统计对应位不同的有多少位。

8．完成 10 个压缩 BCD 数相加，并显示结果。

9．编写程序，将一串字母序列按字母序从小到大排列。

10．试编程统计寄存器 AX 中，相邻位的变化次数。

第 5 章　存储器原理与接口

本章导读

- 存储器分类
- 多层存储结构
- 主存储器及存储控制
- 8086 系统的存储器组织
- 现代内存芯片技术

存储器是组成计算机系统的重要部分。自从冯·诺依曼提出存储程序计算机概念以后，存储器的性能一直是计算机性能的主要指标之一。所谓存储器，是指许多存储单元的集合，用以存放计算机要执行的程序和有关数据。存储器根据其在计算机系统的地位和位置分为内存储器（内存）和外存储器（外存）。

本章将介绍计算机的存储体系的基本概念和组成，讲述典型的半导体存储器芯片的结构、工作原理和外特征，以及内存储器的基本技术、CPU 与内存储器的接口技术。

5.1　存储器分类

一台计算机的内存是指 CPU 能够通过指令中的地址码直接访问的存储器，一般直接与计算机的三大总线即数据总线、地址总线和控制总线相连，常用于存放处于活动状态的程序和数据，如操作系统的常驻部分、正在运行的用户程序等。这要求各存储单元的内容都可以被 CPU 随机访问，目前一般采用半导体存储器实现。内存是计算机系统必不可少的部件，也叫主存储器。外存一般不能为 CPU 直接访问，通常用来存放当前不活跃的程序和数据。外存是主存储器的补充，所以又叫辅助存储器，目前一般用磁盘、磁带等磁介质、光盘和半导体存储器（如 U 盘）等实现。

由于 CPU 的寄存器和算术逻辑单元都由高速器件组成的，因此指令执行速度在很大程度上取决于数据存入和读出主存储器的速度。计算机运算能力的提高、服务范围的扩大、系统软件的日益丰富，对主存储器技术也提出了更高的要求，当新的计算机系统问世时，都伴随着主存储器工艺的更新和改进。正因为这样，存储器的种类日益繁多，分类的方法也有很多种。下面分别从存储器的存储介质、存取方式和在计算机中的作用等角度对存储器进行分类。

1．按构成存储器的器件和存储介质分类

按构成存储器的器件和存储介质分，存储器可分为半导体存储器、磁表面存储器（磁盘和磁带等）、光电存储器等。从原理上讲，只要具有两个明显稳定的物理状态的器件和介质都能用来存储二进制信息。当然，真正作为计算机存储器的器件和介质还必须满足一些技术上的要求，如便于与电信号转换，便于读写，存取速度高、可靠性高等。

计算机诞生后不久，磁芯存储器曾一度成为主存储器的主要存储介质。随着微电子技术的发展，半导体存储器在存取速度、容量、价格和可靠性、体积、制造工艺等方面全面优于磁芯存储器，从而取代磁芯存储器的地位。目前，计算机主存储器使用的都是半导体存储器。

磁表面介质和光电技术实现的存储器由于其存储容量大、访问速度较慢、信息不易丢失的特性，常用于计算机的外存储器。

2．按存取方式分类

（1）随机存储器（Random Access Memory，RAM）

随机存储器 RAM 是指计算机可以随机地、个别地对每个存储单元进行访问，访问所需的时间基本固定，与存储单元的地址无关。这里所说的“与地址无关”是与下面要介绍的串行存取方式比较而言的。随机存储器通常又叫读写存储器，在所有计算机系统中，大、中、小型和微型计算机的主存储器主要采用半导体技术实现的随机存储器。

（2）只读存储器（Read-Only Memory，ROM）

只读存储器 ROM 是一种对其内容只能读、不能写入的存储器。它的内容一般是一次性预先写入的，以后不再随着计算机程序的运行而频繁更改。ROM 通常用来存放固定不变的程序、汉字字型库、字符及图形符号等。由于它与读写存储器分享主存储器的同一个地址空间，故仍属于主存储器的一部分。只读存储器还经常作为微程序控制存储器。

随着半导体技术的发展，只读存储器也根据不同需要设计出不同种类，如：可编程的只读存储器（Programmable ROM，PROM）允许用户根据自己的需要，一次性地写入程序和数据，同时一经写入，就无法更改；可擦除的可编程只读存储器（Erasable Programmable ROM，EPROM）允许用户根据需要多次写入或用紫外线擦去 ROM 的内容；电可擦除只读存储器（Electrically Erasable Programmable ROM，EEPROM）可用电信号进行清除和改写的存储器，与 EPROM 相比，不需要采用紫外线擦除。

与 RAM 相比，ROM 除了集成度高、成本低外，还有一个重大优点就是当电源去掉后，其中的信息是不丢失的。

（3）串行访问存储器（Serial Access Storage，SAS）

串行访问是指对存储器的信息进行读写时，需要顺序地访问。访问指定信息所花费的时间与信息所在地的地址或位置有关。

串行存储器又可分为顺序存取存储器和直接存取存储器。顺序存取存储器是完全的串行访问存储器，如磁带等，它的信息是以顺序的方式从存储介质的始端开始写入或读出；直接存取存储器是部分串行访问存储器，如磁盘，它介于顺序存取和随机存取之间。例如，磁盘存储器的操作包括两步：首先通过磁头或磁道，直接指向整个存储器的一个区域（这一步不按顺序），然后对这一小部分区域按顺序存取。这样，其存取速度就与上述两种有差别，一般用于外存。

对一个具体器件来说，读取方式并不是唯一的，如串行访问的方式并不仅限于外存储器。

在一些特殊用途中，半导体存储器也设计成具有串行特点的结构，如在计算机高级的显示存储器中就设计了双功能的存储器，即包括能对存储器随机读写的功能和串行读出的功能。

3．按在计算机中的作用分类

按存储器在计算机中的作用，存储器可分为主存储器（内存）、辅助存储器（外存）、高速缓冲存储器等。

主存储器用来存放活动的程序和数据，存取速度快、容量较少、每位价格高。主存储器主要采用半导体存储器，根据速度要求，可以分别采用 MOS 工艺、TTL 工艺和 ECL 工艺等。目前，微型计算机中主要采用 MOS 工艺实现的半导体存储器。辅助存储器主要用于存放当前不活跃的程序和数据，其存取速度慢、容量大、每位价格低。缓冲存储器主要在两个不同工作速度的部件之间起缓冲作用。

由于目前计算机使用半导体存储器，半导体存储器的构成将在后面进一步介绍。半导体存储器分类如图 5-1 所示。

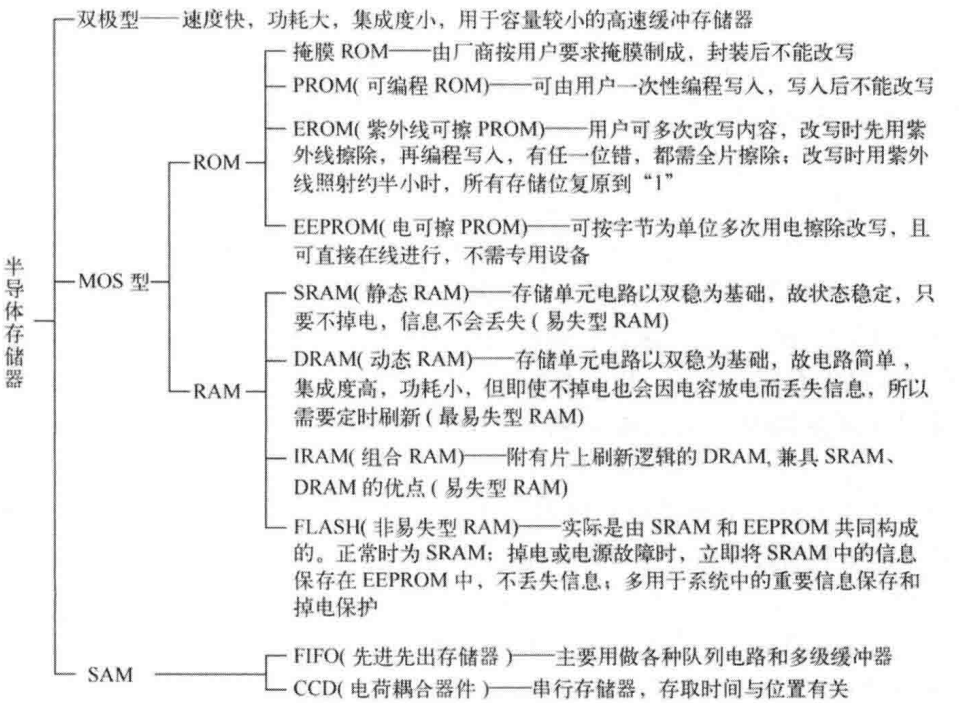


图 5-1　半导体存储器分类

5.2　多层存储结构

半导体存储器尽管在存取速度、容量、价格和制造工艺上有很大提高，但从计算机技术发展来看，仅从改进内存工艺技术着眼，内存的工作速度总是不能满足 CPU 的需要，同时内存在容量上总是落后于系统软件和应用软件的需求。因此，要取得一个兼有大容量、高速度和低成本的存储系统，应该在系统结构的设计上综合利用各种存储工艺的特长，回避其弱点，构成一个较为合理的存储系统，这样就提出了多层存储结构概念。

图 5-2 所示结构就是目前各类计算机中广泛采用的多层存储体系结构，这是一个金字塔的

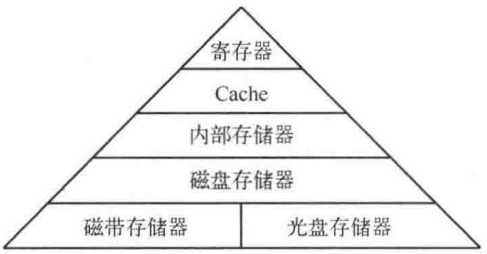


图 5-2　微型计算机存储体系的分层结构

结构。从观念上说，在计算机中凡是能存储程序和数据的都属于计算机的存储体系。这样，CPU 中的寄存器以其访问速度最快、容量最小而处于存储体系的最上端。这些寄存器一般位于微处理器内部，如 8086/8088 微处理器中有 8 个 16 位的寄存器，后来的 80x86 系列微处理器中既有 16 位的段寄存器，也有 32 位的数据寄存器和控制寄存器，以及 64 位、80 位的其他寄存器等。这些寄存器的用途有些是固定的，有些是用户指定的，因此也被称为通用寄存器组。微处理器存取这些寄存器的速度最高，一般在一个时钟周期中完成。对于用户来说，只要充分利用并恰当地安排这些寄存器，就可以极大地提高系统性能。总体来说，设置系列寄存器一个重要的功能是为了尽可能减少微处理器从存储器读写数据的次数。由于寄存器设计在微处理器内部，受芯片面积、集成度、管理等方面的限制，寄存器的数量相对来说是很少的，合理和充分利用寄存器资源对提高软件性能很有必要的。

第二级是高速缓存（Cache）。高速缓存是计算机提高整体性能的一种技术。在微机系统中引入该项技术是从 80486 CPU 开始的。目前的微机系统中采用两级高速缓存（L1 和 L2）。第一级高速缓存 Cache L1 设计在微处理器内部，微处理器对高速缓存的访问非常快捷；第二级高速缓存 Cache L2 安排在微处理器外，通常采用静态的随机存储器，以保证其存取速度。

第三级是内存，用于存放运行的程序和数据。从传统意义上看，微处理器直接对内存进行访问和操作；但在现代微型计算机系统中，微处理器的实际操作是针对内存存在高速缓存中的副本的，微处理器访问时一般直接访问高速缓存，而不是内存，因此内存就可以采用存取速度较慢、集成度较高的存储器芯片，以提高整个微型计算机系统的性能价格比。这类存储器通常采用动态随机访问存储器（DRAM），现在的内存大都采用价廉的同步动态随机访问存储器（SDRAM）。

内存除了采用大量的动态随机存储器，还有部分是用于保存固化程序和数据的只读存储器。这些只读存储器通常是 ROM、EPROM、E^2PROM 或 FLASH（也叫闪存）。现代微型计算机大都采用 FLASH 来存放这些固化程序和数据。相对而言，FLASH 的存取速度较慢，但保存其中的程序主要是系统启动时执行的，对其存取速度并不需要有特殊要求。这样，使用较低速度的 FLASH 对整个微型计算机系统影响不大。

接下来就是具有大存储容量的外部存储器（又称为海量存储器、辅助存储器等），目前多数使用磁盘、磁带、光盘或 U 盘等，保存在这类存储器上的信息非易失性是显然的。在微型计算机系统中，更多使用的是硬盘和光盘。这类存储器的容量已达 TB 级。由于受机械动作的限制，外部存储器在存取速度上与半导体存储器有较大的差距，这类存储器主要用于保存后备的程序和数据。现代微型计算机系统中广泛采用虚拟存储器的技术，即计算机已经具有虚拟存储管理的能力。利用虚拟存储管理，可在硬盘中开辟一块存储空间，作为主存空间的延续，CPU 可以对这部分空间进行“直接”访问，这样就可以在实际内存较小的情况下运行较大的程序。由于体积小、可靠性高和携带方便，U 盘目前已经替代了软盘作为便携式小容量的外存。

从整个微型计算机的分层结构看，整个结构主要是两个层次，即 Cache - 主存层次和主存 - 辅存层次。这样，结构中的每种存储器不再是孤立的存储部件，而是组成一个有机的整体。

整个系统要达到的理想指标具有 Cache 的存取速度和辅存的容量。

1. Cache - 主存层次

Cache - 主存层次解决的是 CPU 与主存在存取速度上的差距。由于容量、价格、功耗等原因，主存不可能设计成完全满足 CPU 立即直接访问的速度，一般与 CPU 中的运算器、控制器相差一个数量级。如果在两者之间设置高速缓冲存储器（Cache），就能较好地解决存取速度问题。高速缓冲存储器设计的要求是在存取速度上基本满足 CPU 中运算器和主控器的速度，同时具有一定的存储容量。

Cache–主存间的地址映像和调度，与下面介绍的主存–辅存层次所采用的技术相仿，不同的是因其存取速度要求高，所有程序和数据的调度完全由硬件来实现。

从 CPU 的角度看，Cache–主存层次的存取速度接近 Cache，但容量是主存的，使用价格接近于主存，因此解决了存取速度与成本之间的矛盾。

2. 主存 - 辅存层次

主存 - 辅存层次解决的是存储器的大容量要求与低成本之间的矛盾。辅存作为外部设备的组成部分，它的信息存储的访问编址与主存的编址无关，早期的计算机要做到程序对外存进行访问，需要程序员花费精力和时间把大程序预先分成块，确定好这些程序块在辅存中的位置和装入主存的地址，而且在运行时要预先安排各块如何和何时调入调出等。现代操作系统已经完全解决了这个问题，不再需要程序员自己去安排主存、辅存间的地址定位，而是由操作系统和辅助软/硬件自动实现。程序员可以把主存、辅存看成统一的整体，利用比主存实际容量大得多的逻辑地址编写程序。该系统不断发展和完善，逐步形成了现在广泛使用的虚拟存储系统。在该系统中，程序员可用机器指令地址码对整个程序统一编址，就像程序员具有对于这个地址码宽度的全部虚存空间一样。这个空间可以比主存实际空间大得多，以便存储整个用户程序。这种指令地址码称为虚拟地址、逻辑地址或程序地址等，其对应的存储容量称为虚存容量或程序空间；实际主存的地址称为物理地址、实（主）地址，其对应的存储容量称为主存容量、实存容量或实（主）存空间。

当用虚拟地址访问主存时，机器自动把它经辅助软件、硬件变换成主存实地址，然后查看这个地址所对应的单元内容是否已经装入主存。如果已在主存中，就进行访问；否则，通过辅助的硬件、软件，把它所在的那块程序或数据由辅存调入主存，然后进行访问。这些操作对程序员来说是透明的，即不用程序员来安排，由计算机自动实现。

5.3 主存储器及存储控制

5.3.1 主存储器

1. 主存储器的主要指标

衡量一个主存储器的性能指标主要有存储容量、存取时间/存储周期和可靠性等。

存储器可以容纳的二进制信息量称为存储容量。计算机能够利用的主存储器最大容量又取决于计算机设计的体系结构和指令的寻址方式。各种寻址方式最终形成的有效的直接地址用

以访问主存储器，因此 CPU 的地址位数影响其对主存储器的可寻址空间。一般的存储单元以字节为单位。例如，具有 16 位地址的计算机能够寻址的存储单元是 64 KB，以 8086/8088 CPU 组成的计算机，因其有 20 根地址线，可寻址的存储单元为 1 MB，而主存容量为 16 MB 时，就需要计算机能产生 24 位地址。

微型计算机系统的存储容量有两个概念要清楚：一个是 CPU 的最大容量，这是由 CPU 的地址线的多少决定的；另一个是实际容量，是指在一个实际的微型计算机系统中具体安装了多大的内存。通常，计算机系统提供的最大容量要远大于实际的装机容量。例如，Pentium 4 CPU 设计的地址线是 36 位，其最大容量应该是 236，即 64 GB，但在采用 Pentium 4 的微型计算机系统中一般也只装配了 512 MB 的内存容量。

主存储器的另一个重要的性能指标是存储器的速度，常用存储器存取时间和存储器周期时间来表示。存储器存取时间（Memory Access Time），又称为存储器访问时间，是指从启动一次存储器操作到完成该操作所需要的时间。例如，从读操作命令发出到该操作完成，并将数据读入数据缓冲寄存器为止所经历的时间，即为存储器存取时间。

存储器周期时间（Memory Cycle Time）是指连续启动两次独立的存储操作（如连续两次读操作）所需间隔的最小时间。通常，存储器周期时间略大于存储器存取时间，其差别与主存储器的物理实现细节有关。

可靠性用平均故障间隔时间 MTBF（Mean Time Between Failures）来衡量。显然，MTBF 越长，可靠性越高。对于某些可靠性要求高的计算机系统（如银行系统的服务器等），除了选择 MTBF 好的芯片外，还可以采用纠错编码技术来延长 MTBF，以提高存储器的可靠性。

2．主存储器的基本操作

主存储器与 CPU 的关系极为密切，这是因为 CPU 需要执行的指令和待处理的数据及结果都暂存在那里。主存储器和 CPU 的连接是通过总线来完成的，如图 5-3 所示。

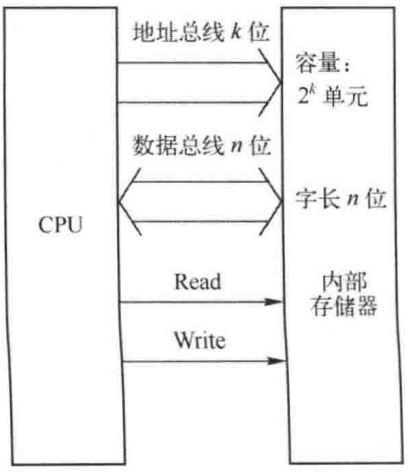


图 5-3　CPU 与内存连接示意

从系统结构的观点看，可把主存储器视为一个黑匣子，在 CPU 中设置两个寄存器来实现存储器和 CPU 之间的数据传输：存储器地址寄存器（Memory Address Register，MAR）和存储器缓冲寄存器（Memory Buffer Register，MBR）。当然，存储器本身内部也有地址寄存器和数据寄存器分别与 MAR 和 MBR 相联系，这一点在研究存储器内部构造时再介绍。MAR 的位数就是地址线的多少，设其为 k 位二进制数，MBR 的位数就是其字长，即为 n。于是，存储器可包含多达 2^k 个可寻址单元，在一个存储周期中有 n 位数据在存储器和 CPU 之间传输。总之，存储器总线包括 x 条地址线和 n 条数据线，加上必要的控制线如 Read（读）、Write（写）等。控制线的作用是规定数据传输的性质和协调操作步骤，在字节寻址的计算机中，还要增加一些控制线，表示什么时候只需传输一字节，而不是 n 位整字长等。

现在来看主存储器的基本操作过程，即 CPU 对主存储器的一次访问：读或写。

为了从存储器中读一个数据并送到 CPU 中，CPU 先将指令中利用寻址方式获得的物理地

址送入 MAR 中，经地址总线送往主存储器，CPU 接着置存储器读控制线 READ 有效，存储器的一次读操作即可开始，在读控制线 READ 的作用下，被地址线选中的存储器某个单元的内容就送到了数据总线。CPU 为等待从存储器读出数据，需要暂时封锁自己的流程，或者安排一些与该数据无关的操作。当存储器完成本次操作，CPU 通过数据线把所读的数据送入 MBR。

与读类似，为了“写”一个数据到主存，CPU 先通过指令确定需要保存数据在主存某个单元的地址，并把此地址经 MAR 送地址总线，将数据送数据总线。接着，CPU 置存储器写控制线有效，存储器的写操作即开始，主存储器从数据总线接收数据，并按地址总线指定的地址存储。

5.3.2 主存储器的基本组成

目前，半导体存储器以其存取速度快、功耗低、价格低、模块化程度高和装配密度大等优点，广泛应用在各种计算机系统中。本节介绍 RAM 的基本存储电路和结构。

MOS 型器件构成的 RAM 可分为静态和动态两种。静态 RAM 通常由六管构成的触发器作为基本存储电路。下面介绍如图 5-4 所示的六管静态存储单元电路的工作过程。其中，虚线框中央由 T_1～T_4 组成的是一个交叉耦合而成的双稳态触发器（T_3 和 T_4 是负载管），T_5 和 T_6 作为控制管。当 X 的译码输出线为高电平时，T_5 和 T_6 导通，A、B 端就与位线 D 和 $\overline{D}$ 相连；当该电路被选中时，相应的 Y 译码输出也是高电平，故 T_7 和 T_8 也导通，于是 D 和 $\overline{D}$ 与外部数据线的输入/输出电路 I/O 和 $\overline{I/O}$ 相通。需要对存储器写入时，写入信息自 I/O 和 $\overline{I/O}$ 线输入。若要写“1”，则 I/O 线为“1”，$\overline{I/O}$ 线为“0”。它们通过 T_7、T_8 以及 T_5、T_6 分别与 A 端和 B 端相连，使 A 端为“1”，B 端为“0”，强迫 T_2 导通。T_1 截止，相当于把输入电荷存储于 T_1 和 T_2 的栅极。由于存储单元有电源和两个负载管，可以不断地向栅极补充电荷，只要不掉电就能保持写入的信号不丢失。与动态 RAM 相比，它不用刷新。其读出的情况同写入类似，而且这种读出是非破坏性的，即信息在读出后，仍保留在存储电路内。

动态 RAM 通常由单管组成，基本存储电路如图 5-5 所示，T_s 和电容 C_s 构成最简单的动态存储单元。写入时，字选择线为“1”，T_s 导通，写入信号由位线（数据线）存入电容 C_s 中；在读出时，字选择线为“1”，存储在电容 C_s 上的电荷通过 T_s 输出到数据线上，通过读出放大器即可得到存储信息。为了节省面积，单管存储电荷的电容不可能做得很大，一般比数据线上的分布电容 C_d 小，因此每次读出后，存储内容就被破坏，必须采取刷新技术恢复原来的信息。

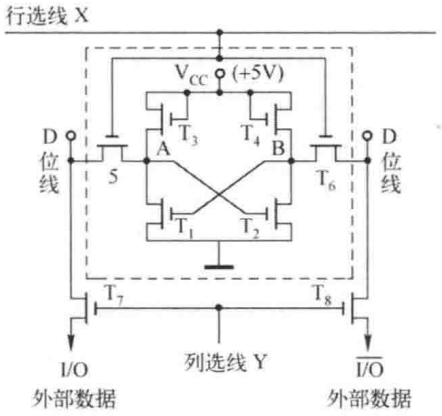


图 5-4 六管静态存储单元电路

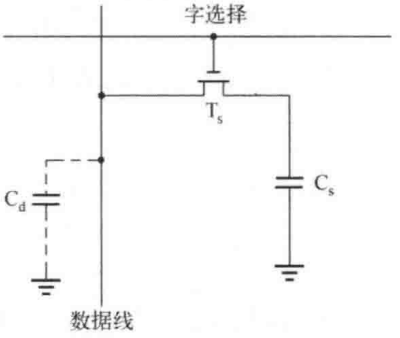


图 5-5 单管动态存储单元

1．存储体

存储器由大量的基本存储电路组成。这些存储电路有规则地组合起来就成为存储体。在较大容量的存储器中，往往把各字对应的位组织在一个片中，这样的存储芯片称为多字一位片，如 256K×l 位、512K×1 位等。现在多采用把每个字的几位组织在一个片中，称为多字多位片，如 256K×4 位、2K×8 位等。图 5-6 是一个典型的 RAM，它的存储体是 4096×1，排成矩阵的形式，即 64×64=4096，这是为了便于译码寻址。

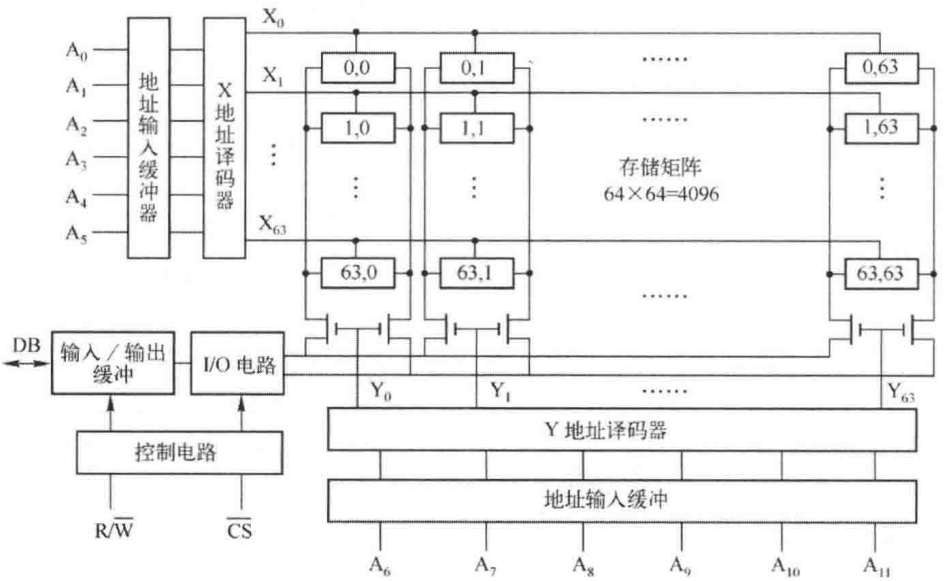


图 5-6　典型的 RAM

2．外围电路

一个存储器芯片除了存储体，还有许多外围电路，见图 5-6。

地址译码器：用于对 n 条地址线译码，以选择 $2n$ 个存储单元中的一个。

I/O 电路：处于数据总线和被选用的单元之间，用以控制被选中的单元读出或写入，具有放大信号的作用。

片选控制端 $\overline{CS}$（Chip Select）：由于每片芯片的存储容量总是有限的，所以一个存储体往往由一定数量的芯片组成。在地址选择时，首先要选片，用地址译码器输出和一些控制信号（如 8086 CPU 的 M/$\overline{IO}$ ）形成选片信号。只有当某片的 $\overline{CS}$ 输入信号有效时，该片所连的地址线才有效，这样才能对该片上的存储单元进行读或写的操作。

集电极开路或三态输出缓冲器：在实际系统中，常需将几片 RAM 的数据线并联使用，或与双向的数据总线相接，因而需要用到集电极开路或三态输出缓冲器。

另外，在动态 MOS 型 RAM 中还有预充、刷新等方面的控制电路等。

3．地址译码方式

存储器的地址译码有两种方式：一种是单译码方式，又称为字结构；另一种是双译码方式，又称为复合译码结构。在字结构中，n 根地址输入经全译码有 2^n 个输出，用以选择 2^n 个字，如 16 个字对应 $A_3 \sim A_0$ 共 4 根地址线，经译码获得 16 根选择线。显然，随着存储字的增加，译码输出线及相应的驱动电路会急剧增加，存储器成本也将迅速增加，这种译码方式仅适用于

小容量存储器。复合译码结构往往用于地址位数 n 很大时，把 n 位地址线分成接近相等的两段，分别译码，产生一组 X 地址线和一组 Y 地址线，然后让 X 地址线和 Y 地址线在字存储单元列成矩阵的存储体中一一相“与”，选择相应的字存储单元。

图 5-7 是一个含有 1×1024 个字的存储器的双译码电路，1024 个字按 32×32 的矩阵排列。

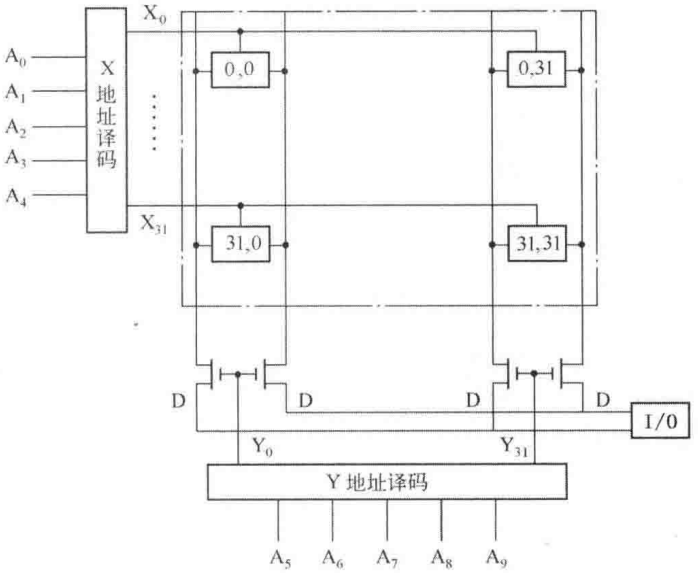


图 5-7　双译码存储电路

要对 1024 个字进行访问，需要 10 根地址线，即 $1024=2^{10}$，10 根地址线分成两组，$A_0 \sim A_4$ 为一组，$A_5 \sim A_9$ 为另一组，前组经 X 译码器输出 32 条行选择线，后组经 Y 译码器输出 32 条字选择线。行选择线和字选择线的组合可以方便地找到 1024 个字中的任何一个，而译码器输出的总端线仅为 64（2^5+2^5）根，而不是采用单译码时的 1024（2^{10}）根。由此可见，双译码电路比单译码电路具有优越性。

5.4　8086 系统的存储器组织

5.4.1　8086 CPU 的存储器接口

1．不同模式下 CPU 的存储器接口

在 8086 最小模式系统和最大模式系统中，8086 CPU 可寻址的最大存储空间为 1 MB。正如 2.2 节中指出的那样，8086 最小模式系统和最大模式系统的配置是不一样的。8086 最大模式系统中增设了一个总线控制器 8288 和一个总线仲裁器 8289，因此 8086 CPU 和存储器系统的接口在这两种模式中是不同的。

图 5-8 是 8086 最小模式系统的存储器接口示意图，寻址存储单元的信号由多路复用的地址/数据总线 $AD_{15} \sim AD_0$、地址线 $A_{19} \sim A_0$ 和总线高位有效信号 $\overline{BHE}$ 提供。存储器的控制信号 ALE、$\overline{RD}$、$\overline{WR}$、M / $\overline{IO}$、DT / $\overline{R}$ 和 $\overline{DEN}$ 直接由 8086 CPU 产生。

图 5-9 是 8086 最大模式系统的存储器接口示意，包括一片 8288 总线控制器芯片。8288 接收 8086 发送的总线状态信息 $\overline{S}_2$、$\overline{S}_1$ 和 $\overline{S}_0$，将这 3 位标识总线周期类型的状态信号译码，产

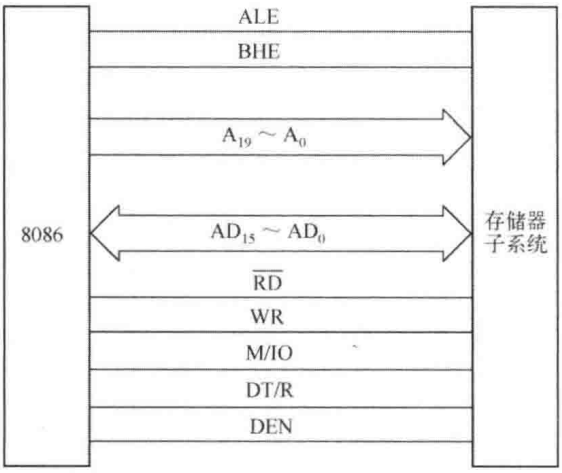


图 5-8　8086 最小模式系统存储器接口

生读/写信号 $\overline{MRDC}$、$\overline{MWTC}$ 和 $\overline{AMWC}$，以及控制信号 ALE、DT / $\overline{R}$ 和 $\overline{DEN}$。由此可见，在最大模式系统中，8288 代替了 8086 CPU 产生和存储器接口的大多数定时和控制信号，仅 $\overline{BHE}$ 和 $\overline{RD}$ 信号仍然由 8086 CPU 提供。

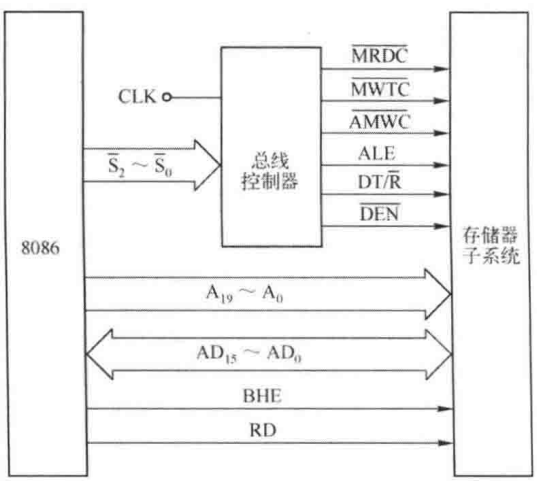


图 5-9　8086 最大模式系统的存储器接口

在 8086 存储器系统中，20 位地址总线（$A_{19} \sim A_0$）的最大寻址存储空间是 1 MB，其地址范围为 00000H～FFFFFH。显然，在 8086 微机系统中，存储器系统实际上是以字节为单位组成的一维线性空间。

2.2 节介绍 8086 存储器的组成时指出，8086 寻址的 1 MB 存储器空间可以分成两个 512 KB 的存储体，一个存储体包含偶数地址，另一个存储体包含奇数地址。任何两个连续的字节可以作为一个字来访问，显然其中一字节必定来自偶地址存储体，另一字节必定来自奇地址存储体，地址值较低的字节是低位有效字节，地址值较高的字节是高位有效字节。

为了有效地使用存储空间，一个字可以存储在以偶地址或奇地址开始的连续两个字节单元中，地址的最低有效位 A_0 决定了字的边界。如果 A_0 是 0，则字存放在偶地址边界中，其低 8 位有效字节存储于偶地址单元中，高 8 位有效字节存储于相邻的奇地址单元中。同理，如果 A_0 是 1，那么字是存放在奇地址边界上。

对所有位于偶地址边界上的字节或字的访问，8086 只需一个总线周期就能完成，而对于

在奇地址边界上的字的访问，8086 需要两个总线周期才能实现。

8086 系统的 1 MB 存储空间的安排是有要求的，其最高和最低地址空间是留给某些特殊的处理功能使用的。如存储单元 00000H～003FFH 共 1 KB 用于存放 Intel 保留的 256 种中断矢量，FFFF0H～FFFFFH 共 16 字节用于存放启动程序。8086 应用程序不能把这些区域改作其他用途，否则会使系统与未来的 Intel 产品不兼容。除此以外，ROM 和 RAM 可位于 1 MB 存储空间的任何位置。

2．接口设计中的一些问题

上面介绍了 8086 CPU 的存储器接口，当 8086 CPU 与存储器系统实际连接时，还要考虑许多具体问题。

① CPU 总线的负载能力。CPU 总线在设计时负载能力都有一定限制。在小型系统中，CPU 可以直接与存储器相连，而在较大的系统中，就需要增加缓冲器、驱动器等。

② CPU 的时序与存储器的存取速度之间的配合问题。系统中，CPU 的读写时序是固定的，这时要考虑对存储器存取速度的要求；若存储器已经确定，则考虑是否要插入等待周期 T_w。例如，8086 CPU 的主频采用 5 MHz，则 1 个时钟周期为 200 ns，将每个时钟周期称为 1 个 T 状态。由本书 2.3 节中的介绍可知，CPU 与存储器交换数据，或者从存储器取出指令，必须执行 1 个总线周期，而最小总线周期由 4 个 T 状态组成。如果存储器速度比较慢，CPU 就会根据存储器送来的“未准备好”信号（READY 信号无效），在 T_3 状态后插入等待状态 T_w，从而延长了总线周期，直到存储器准备完成。

③ 存储器的地址分配和选片问题。内存的扩展和因不同用途的划区都涉及存储器的地址分配和选择。当多片存储芯片组成存储器时，还有一个选片信号的问题。

本节在介绍 8086 CPU 与 ROM、静态 RAM 的具体连接方法的同时，还将介绍存储器扩展以及地址译码技术。

3．CPU 提供的信号线

无论 8086 最小模式系统存储器接口或 8086 最大模式系统的存储器接口，其提供的相关信号线对存储器一方来说应该是一样的。下面重点讨论这些信号线的作用及接口电路。

在实际的存储器接口电路中，CPU 直接提供的或通过一些简单电路如总线控制芯片提供的信号线如下。

① D_{15}～D_0：16 位数据线，在接口电路中分成两部分，即低 8 位数据线 D_7～D_0 和高 8 位数据线 D_{15}～D_8。CPU 可以分别对低 8 位数据线和高 8 位数据线进行操作，也可以同时对 16 位数据线进行操作。例如：

```
MOV     [0000], AL                ; 对低 8 位数据线进行写操作
MOV     [0001], AL                ; 对高 8 位数据线进行写操作
MOV     [0000], AX                ; 对 16 位数据线进行写操作
```

由于 8086 CPU 对数据线操作的多样性，接口电路较一般的 8 位 CPU 如 Z80 或 8088 等存储器接口电路复杂得多。

② A_{19}～A_0：20 位地址线。尽管 8086 CPU 是 16 位，但其内存单元仍是 8 位的。也就是说，8086 CPU 所允许的最大内存容量是 1M 字节，而不是 1M 个字。

③ $M/\overline{IO}$：存储器或 I/O 端口访问信号。8086 CPU 地址线上的信息有两种，一种是内存

地址，另一种是外设地址。M/$\overline{IO}$ 信号指定了当前地址线上地址信息的类型。如M/$\overline{IO}$=0，表明地址线上的信息为外设地址；M/$\overline{IO}$=1，表明地址线上的信息为内存地址信息。

④ $\overline{RD}$：读信号线。当其有效时（即低电平），表明 CPU 从内存读数据，这时数据线的数据流沿内存、数据总线到 CPU 的方向流动。

⑤ $\overline{WR}$：写信号线。当其有效时，表明 CPU 写数据到内存，这时数据线的数据流沿 CPU、数据总线到内存的方向流动。

⑥ $\overline{BHE}$：总线高位有效信号。当其有效时，表示 CPU 是对高 8 位的数据线的操作。对初学者来说，也许 $\overline{BHE}$ 的作用和功能是比较令人困惑的。与一般的 8 位 CPU 相比，$\overline{BHE}$ 是 8086 CPU 系列所特有的，是掌握 8086 CPU 与内存接口的关键点之一。

5.4.2 存储器接口举例

不同类型的存储器所提供的信号线大同小异，这里仅讨论两种：EPROM 和 RAM。

1．ROM 扩展电路

只读存储器在计算机系统中的功能主要是存储程序、常数和系统参数等。这些信息有一个共同的特点就是在计算机工作过程中保持不变，CPU 只能对其所在单元进行读操作而不能进行写操作。如果需要对存储单元的数据进行更新，则需要特殊手段进行数据写入。如对于 EPROM，把信息写入芯片，首先要把芯片放在一定强度的紫外线下照射一段时间，然后对芯片的一个相关引脚施加大于 12 V 以上电压并保持一定的写入时间，才能把数据写入。我们称这个操作过程为芯片编程，称这个 12 V 以上的电压为编程电压。

目前常用的 EPROM 芯片有一个系列，型号为 2716、2732、…、27256 等。型号和容量有直接的关系，即“27”后面的数字除以 8 就是容量，单位是 KB。比如，2716 的容量是 16/8=2，即 2 KB，它有 11 根地址线，2732 的容量是 4 KB（32/8），因此它有 12 根地址线。

27 系列 EPROM 芯片的信号线（如图 5-10 所示）可分为如下几类。

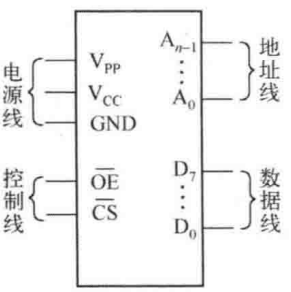


图 5-10　EPROM 引脚

（1）总线部分

$D_7 \sim D_0$：数据线。

$A_{n-1} \sim A_0$：地址线，n 是地址线个数。对于 2716，n=11；对于 27256，n=15。

（2）电源部分

V_{CC}，GND：电源线和地线。

V_{PP}：编程电压，在 CPU 仅对芯片进行读操作时，一般直接接电源电压。

（3）控制部分

$\overline{OE}$：读控制线。当其有效时，数据从 EPROM 内的某个单元通过数据线送到 CPU。

$\overline{CS}$：片选线。该信号一般为低电平有效，有效时表示本芯片工作。在芯片编程时，这根线常作为编程控制线。

下面讨论 $\overline{CS}$ 信号的意义。对于一个内存容量需求较大的系统，往往有多个内存芯片。当然，CPU 对这些芯片进行访问时，不可能所有芯片都同时与 CPU 交换数据，只有 CPU 所指定的芯片才能与其传输数据。$\overline{CS}$ 就起了这个“指定”的作用。也就是说，当 CPU 对内存进行访

问时，只有指定芯片的$\overline{CS}$有效，而其他芯片的$\overline{CS}$无效。

$\overline{CS}$信号一般是由高位地址线产生的，它是计算机接口中的一个极为重要的部分。掌握了它，也就基本掌握了计算机的存储器接口和I/O接口电路的设计。

【例 5-1】 设计一 ROM 扩展电路，容量为 32K 字，地址从 00000H 开始。

在着手设计这个接口电路前，首先确定内存的容量及选用的 EPROM 型号。前面讲过，尽管 8086 CPU 是 16 位的，但其内存单元仍是 8 位的，而 32K 字的容量实际就是 64 KB。如果选用 27256，因为它是 32 KB 的容量，所以要计算所需 27256 芯片的个数：(32K×16)/(32K×8)=2，即用 2 片 27256。其中一片存储低 8 位信息，接 CPU 数据线的 $D_7 \sim D_0$；另一片存储高 8 位信息，接 CPU 数据线的 $D_{15} \sim D_8$。最后合成的效果就是存储 32K 字的信息量。图 5-11 为 27256 的引脚。

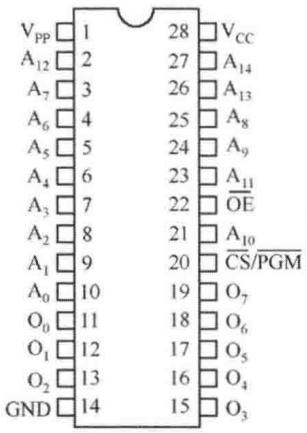


图 5-11 27256 引脚

然后计算它的地址范围。对于地址从 00000H 开始的 64 KB 容量的存储器，其地址范围为 00000H～0FFFFH。表 5-1 给出了地址的变化范围。

表 5-1 64 KB EPROM 内存地址范围

	A_{19} A_{18} A_{17} A_{16}	A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
最小地址序号	0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
最大地址序号	0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

当 CPU 的低位地址线 A_{15}～A_0 从 0000H 变化到 FFFFH 时，其高位地址 A_{19}～A_{16} 保持不变，仍为 0H。这就提示我们，片选信号由 $A_{19} \sim A_{16}$ 产生。当其为 0H 时，片选信号有效。而 CPU 的 $A_{15} \sim A_0$ 作为 EPROM 的片内单元的译码线，可以寻址 64 KB（2^{16}），应直接与芯片的地址输入线 $A_{14} \sim A_0$ 相连接。当其发生变化时，可以选中芯片内的任一指定单元。

这里有一个小问题还没有解决，27256 的地址输入线为 15 根（$A_{14} \sim A_0$），而 CPU 的地址线除去产生片选信号的 4 根线，还剩下 16 根（$A_{15} \sim A_0$），它们如何和 27256 相连呢？解决这个问题的关键就要考虑 8086 CPU 是一个 16 位 CPU，可以进行 16 位操作。8086 CPU 在读数据时，是 16 位的操作，同时读一个偶地址单元和一个奇地址单元。这就表明 CPU 的地址线 A_0 是不能参与地址译码的，因为一旦 A_0 参与地址的译码，就会区别奇、偶地址，这样就不可能同时读出两个单元的内容，即 16 位信息了。这样得出了如图 5-12 所示的接口电路。

下面讨论图 5-12 电路读数据的过程。如果 DS 内容为 0000H，当 CPU 执行指令 MOV AX, [0000] 时，对 ROM 芯片的操作过程如下：首先 CPU 通过其地址线发地址信息 00000H；接着 M/$\overline{IO}$ 有效，表示地址线上的信息为内存地址；于是$\overline{CS}$信号有效，两片 EPROM 都可以工作，同时由于 A_{15}～A_1 全为 0，两个芯片的第 0 号单元均被选中；最后 CPU 发读信号，$\overline{RD}$ 有效，数据从两片 EPROM 的 0 号单元通过数据线 D_0～D_{15} 传输到 CPU 内。

接 CPU 数据线 D_0～D_7 的存储器芯片为偶片，接 CPU 数据线 D_8～D_{15} 的存储器芯片为奇片。对于指令 MOV AX, [0000]，CPU 实际是对地址 0000H 单元和 0001H 单元进行了操作。所以，偶片 EPROM 的第 0 号单元在整个内存系统中的地址为 00000H，而奇片 EPROM 的第 0 号单元在整个内存系统中的地址为 00001H。

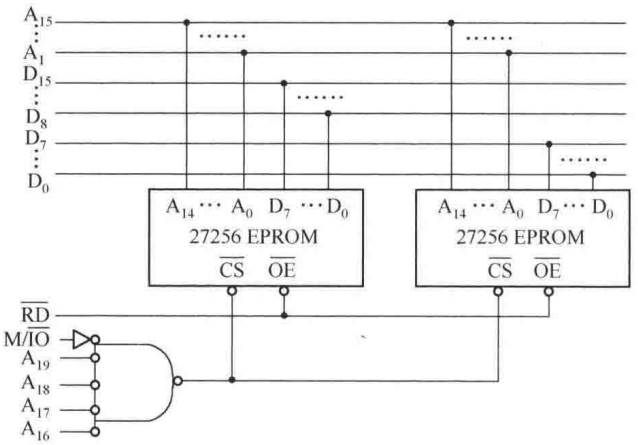


图 5-12　64 KB ROM 扩展电路

2．RAM 扩展电路

随机读写存储器在计算机系统中的功能主要是存储程序、变量等。在计算机运行过程中，程序所处理的变量可能要随时更新，甚至运行的程序都有可能被系统动态删除以腾出空间给其他进程。这类信息用 ROM 来存储是不行的。当然用 RAM 存储时，如果计算机关机，这些信息也将不再存在。目前，静态 RAM 常用的芯片也有一个系列，型号为 6116、6264、62256 等。与 27 系列的 EPROM 一样，型号和容量有直接的关系，即“61”或“62”后面的数字除以 8 就是容量，单位是 KB。比如，6116 的容量是 2 KB（16/8），6264 的容量是 8 KB（64/8）。

RAM 芯片的信号线（如图 5-13 所示）可分为如下几类。

（1）总线部分

$D_7 \sim D_0$：数据线。

$A_{n-1} \sim A_0$：地址线，n 是地址线个数。对于 6116，n=11；对于 62256，n=15。

（2）电源部分

VCC，GND：电源线和地线。

（3）控制部分

$\overline{WR}$：写控制线。当其有效时，CPU 把数据通过数据线传输到 RAM 中的某个单元。

$\overline{OE}$：读控制线。当其有效时，数据从 RAM 内的某个单元通过数据线送到 CPU。

$\overline{CS}$：片选线。该信号一般为低电平有效，有效时表示本芯片工作。

RAM 的接口电路要比 ROM 复杂些，下面结合一个例子来讨论 RAM 的接口电路。

【例 5-2】 设计一个 RAM 扩展电路，容量为 32K 字，地址从 10000H 开始，芯片采用 62256。图 5-14 为 62256 的引脚图。

首先确定内存的容量及所需芯片的个数。因为 32K 字的容量实际就是 64 KB，而选用的 62256 的容量是 32KB 的容量，所以要计算所需 62256 芯片的个数：(32K×16)/(32K×8)=2，即用 62256 芯片 2 片。其中一片为偶片存储，接 CPU 数据线的 $D_7 \sim D_0$；另一片为奇片存储，接 CPU 数据线的 $D_{15} \sim D_8$。最后合成的效果就是存储 32K 字的信息量。

然后讨论 $\overline{CS}$ 信号的产生。对于地址从 10000H 开始的 64 KB 容量的存储器，其地址为 10000H～1FFFFH。表 5-2 给出了地址的变化范围。

可以看出，当 CPU 的低位地址线 $A_{15} \sim A_8$ 从 0000H 变化到 FFFFH 时，其高位地址 $A_{19} \sim$

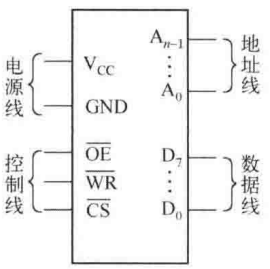


图 5-13　RAM 引脚

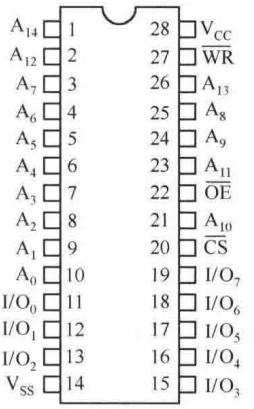


图 5-14　62256 引脚

表 5-2　64 KB RAM 地址范围

	$A_{19}A_{18}A_{17}A_{16}$	$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9\ A_8A_7A_6\ A_5A_4\ A_3A_2A_1A_0$
最小地址序号	0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
最大地址序号	0 0 0 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

A_{16} 保持不变，显然片选信号应该由 A_{19} ～ A_{16} 产生。当其为 0001H 时，片选信号有效。CPU 的 A_{15} ～ A_0 作为 RAM 的片内单元的译码线，可以寻址 64 KB，应直接与芯片的地址输入线 A_{14} ～ A_0 相连接，而 CPU 的 A_0 作为奇/偶片选择。

下面讨论与 EPROM 接口电路不同的方面。ROM 在正常工作过程中，CPU 对它仅做只读不写的操作，即总是 16 位的操作；但对 RAM 的操作要复杂得多，因为 CPU 不仅对它进行读操作，还要进行写操作。写操作有 3 种类型：写 16 位数据，写低 8 位数据和写高 8 位数据。对于 16 位的数据读/写操作，接口电路中 RAM 区的偶片和奇片同时工作；而写 8 位的数据操作，接口电路中只有偶片 RAM 或奇片 RAM 中的一片工作。这表明为了区别偶片和奇片，在设计 RAM 的接口电路中应该把 CPU 的地址线 A_0 用上，因为 A_0 可以区别出偶地址和奇地址。但当 CPU 对 16 位数据读和写时，偶片和奇片同时工作，A_0 又是不能用的。

如何解决这个问题呢？写操作有 3 种类型，而 A_0 只能提供 2 个逻辑状态，所以仅用 A_0 是不可能同时解决 3 种类型的写操作的。为此，8086 CPU 提供了另一根控制线，即总线高位有效信号 $\overline{BHE}$。当它有效时，表明 CPU 对高 8 位的数据线进行操作。A_0 和 $\overline{BHE}$ 有 4 种逻辑组合，因此完全可以表示 3 种数据操作，从而解决了上述的问题。表 5-3 给出了 A_0 和 $\overline{BHE}$ 逻辑组合所对应的 8086 CPU 不同类型的数据操作。这样就得出了图 5-15 所示的 RAM 与 8086 CPU 的接口电路。

表 5-3　A_0 和 $\overline{BHE}$ 编码含义

$\overline{BHE}$	A_0	总线使用情况
0	0	16 位数据总线上进行字传输
0	1	高 8 位数据总线上进行字节传输
1	0	低 8 位数据总线上进行字节传输
1	1	无效

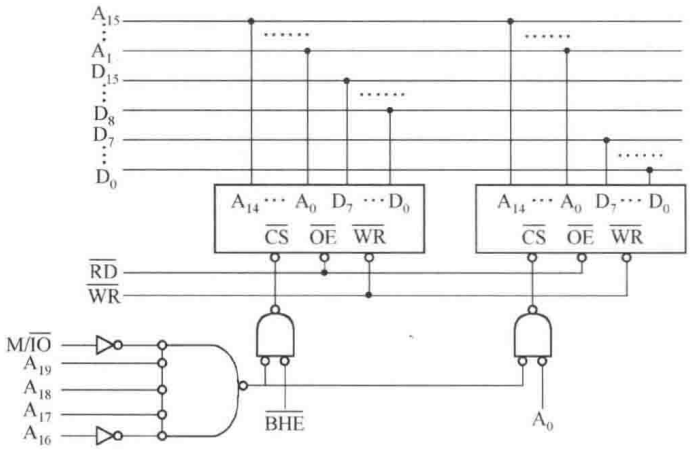


图 5-15　64 KB RAM 扩展电路

如果 DS 的内容为 0001H，当 CPU 执行指令“MOV　[0000], AX”时，CPU 产生如下相关信息：AX 内容进入数据线 D_{15}～D_0，除了 $A_{16}=1$，地址线其余全为 0，$\overline{BHE}$ 和 A_0 也为 0，$\overline{WR}$ 有效，这样两片 RAM 都可以工作，两个芯片的第 0 号单元均被选中，16 位数据低 8 位和高 8 位分别进入偶片 RAM 和奇片 RAM 的 0 号单元。

当 CPU 执行指令“MOV　[0001], AL”时，CPU 发相关信息如下：AL 内容送数据线 D_{15}～D_8，除了 $A_{16}=1$，地址线全为 0，$\overline{BHE}$ 为 0，A_0 为 1，$\overline{WR}$ 有效，这样奇片 RAM 可以工作，数据便进入奇片 RAM 的 0 号单元。

读者可自行分析，当 CPU 执行指令“MOV　[0000], AL”时，CPU 和 RAM 接口的工作情况。

3．3－8 译码器 74LS138

前面所举的例子中片选信号是由组合逻辑电路产生的。这种电路的不足之处在于市场上可能没有相应的定型芯片，必须自己用相关元件来组合形成。这样，$\overline{CS}$ 的产生电路就比较复杂。若要扩展大容量内存，每组的存储器都要有自己的译码电路，这个问题就更加突出。

在实际译码电路的设计中，常用 3－8 译码器 74LS138，而不是采用上面的组合逻辑电路。用 74LS138 作为译码电路的主体器件有如下两个优点：

① 速度快。电路中往往使用一片 74LS138 芯片就完成了 $\overline{CS}$ 信号的产生，信号由级联而造成的延时大大减少，从而提高了译码速度。

② 译码系统简单。74LS138 译码芯片可以提供 8 个片选信号，大大简化了译码电路。

74LS138 是 3－8 译码器，有 3 个“选择输入端”C、B、A，可以选择 8 个输出线 $\overline{Y}_0$～$\overline{Y}_7$。当 C、B、A 的信号组合选择到某个输出线时，输出线有效，即输出为低电平。74LS138 还有 3 个“使能输入端”（又称为“允许端”或“控制端”）G_1、$\overline{G}_{2A}$ 和 $\overline{G}_{2B}$，当其有效时，即 $G_1=1$，$\overline{G}_{2A}=0$，$\overline{G}_{2B}=0$，译码器才能工作。其功能如表 5-4 所示。根据表 5-4，可以容易实现本节所举的存储器接口电路，如图 5-16 所示。译码电路可以提供 8 个片选信号，最大可扩展 512 KB 内存，而电路中只有一个 74LS138 芯片。采用 74LS138 产生片选信号的一般方案是：地址总线的低位部分接存储器，高位部分接 74LS138 选择输入端和使能输入端，同时 M/$\overline{IO}$ 接 74LS138 使能输入端。当高位地址的信号线数目大于 74LS138 选择输入端和使能输入端的数目时，可考虑部分译码方案和采用组合逻辑电路，如或门对部分地址线进行综合后再接

74LS138。

表 5-4　74LS138 功能表

输　入						输　出							
使　能			选　择										
G_1	$\overline{G}_{2A}$	$\overline{G}_{2B}$	C	B	A	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1
其　他			×	×	×	1	1	1	1	1	1	1	1

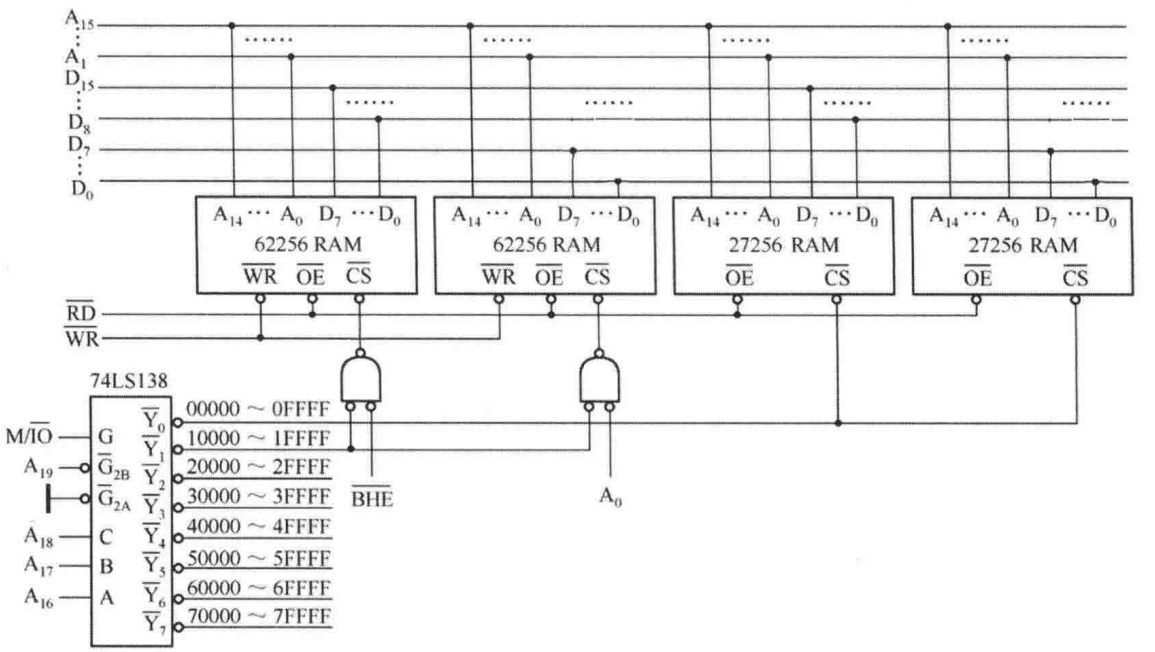


图 5-16　128 KB 内存扩展电路

5.5　现代内存芯片技术

在目前的实际应用中，动态随机半导体存储器不再是以单独的芯片形式提供最终用户，都是采用内存条的形式出现，而微型计算机的主板上提供与内存条匹配的插槽，使用十分方便。存储器，尤其是动态的 RAM 有了许多新的技术和手段，包括 EDO DRAM（扩展数据输出动态随机访问存储器）、SDRAM（同步动态随机访问存储器）和 DDR（双数据速率）、RDRAM（突发存取的高速动态随机访问存储器）等。

习 题 5

1．按存储器在计算机中的作用，存储器可分成哪几类？简述其特点。

2．什么叫 RAM 和 ROM？RAM 和 ROM 各自的特点是什么？

3．什么是多层次存储结构？它有什么作用？

4．什么是虚拟存储系统？什么是虚存容量和实存容量？

5．主存储器的主要技术指标有哪些？

6．8086 CPU 与存储器连接时要考虑哪几方面的因素？

7．在 8086 系统中，若用 1024×1 位的 RAM 芯片组成 16K×8 位的存储器，需要多少芯片？8086 CPU 的地址线中有多少位参与片内寻址？多少位用作芯片组选择信号？

8．在 8086 系统中，若从存储器奇地址体中写 1 字节数据，请列出存储器有关的控制信号和它们的有效逻辑电平信号。

9．在 8086 系统中，试用 8K×8 位的 EPROM 2764、8K×8 位的静态 6264 和 74LS138 译码器，构成一个 16 KB 的 ROM（从 F0000H 开始）和 16 KB 的 RAM（从 C0000H 开始），设 8086 工作于最小模式。画出硬件连接图，写出 ROM 和 RAM 的地址范围。

第 6 章　微型计算机的输入和输出

本章导读

- CPU 与外设通信的特点
- 输入/输出方式
- CPU 与外设通信的接口
- 8086 CPU 的输入/输出

6.1　CPU 与外设通信的特点

前面介绍的存储器与 CPU 交换信息时，它们在数据格式、存取速度等方面基本上是匹配的。也就是说，CPU 要从存储器读入指令、数据或向存储器写入新的结果和数据，只要一条存储器访问指令就可完成；在硬件连接方面，只需芯片与芯片之间的引脚直接连接。但 CPU 与外部设备通信至少有两方面的困难：第一，CPU 的运行速率要比外设的处理速度高得多，通常简单地用一条输入/输出指令是无法完成 CPU 与外设之间的信息交换的；第二，外设的数据线和控制线也不可能与 CPU 直接连接，如一台打印机不能将其数据线直接与 CPU 的引脚连接，键盘或者其他外设也是如此。综上所述，CPU 与外设通信具有如下特点：

- 需要接口作为 CPU 与外设通信的桥梁。
- 需要有数据传送之前的“联络”。
- 要传递的信息有三方面内容：状态、数据和控制信息。

CPU 与外设通信必须借助必要的电路来实现，这样的电路被称为接口或 I/O 接口。由于接口是 CPU 与外设间的桥梁，这就要求接口具有以下几方面的功能：

- 进行地址译码或设备选择，以便使 CPU 能与某一指定的外部设备通信。
- 状态信息的应答，以协调数据传送之前的准备工作。
- 进行中断管理，提供中断信号。
- 进行数据格式转换，如正负逻辑的转换、串行与并行数据转换等。
- 进行电平转换，如 TTL 电平与 MOS 电平间的转换。
- 协调速度，如采用锁存、缓冲、驱动等。
- 时序控制，提供实时时钟信号。

6.1.1 I/O 端口的寻址方式

在微机系统中，存储器的每个单元分配一个唯一的物理地址，对存储器的访问必须直接或间接地提供被访问的存储单元的地址。同理，CPU 与外部设备通信，要区分系统中的不同外设，就必须为每个外设分配必要的地址。为了与存储单元地址相区别，这样的地址称为端口地址。一个外部设备可能分配一个或一个以上的端口地址，其形成端口地址的方式与形成存储单元地址的方式类似。

微处理器设计时，有两种 I/O 端口的处理方式：存储器映像的 I/O 寻址和 I/O 映像的 I/O 寻址。

在系统中将存储单元和 I/O 端口地址统一编址，此时一个 I/O 端口地址就是一个存储单元地址，从硬件上没有区别，对 I/O 端口的访问与对存储器的访问相同，如图 6-1 所示，这种方式称为存储器映像的 I/O 寻址。

在系统中，I/O 端口地址与存储单元地址分别编址，有各自的地址，使用不同的指令，如图 6-2 所示，如对外设通信用 IN 或 OUT 指令、对存储单元用存储器访问指令，这种方式称为 I/O 映像的 I/O 寻址。

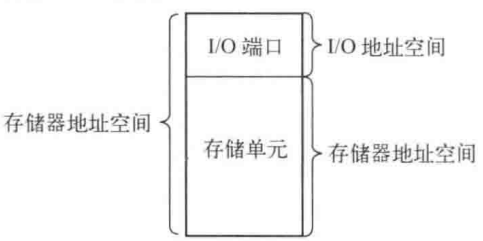


图 6-1　存储器映像的 I/O 寻址

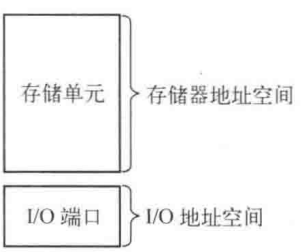


图 6-2　I/O 映像的 I/O 寻址

存储器映像的 I/O 寻址方式的优点：由于 I/O 和存储器在地址上没有区别，在程序设计时可以使用丰富的指令对端口进行操作，甚至包括对端口数据的运算等。

存储器映像的 I/O 寻址方式的缺点：I/O 端口需要占用部分微处理器的地址空间，由于存储器和 I/O 端口地址在形式上没有区别，相对增加了程序设计和阅读的难度。

I/O 映像的 I/O 寻址方式的优点：程序阅读方便，使用 IN 或 OUT 指令就一定是对外设的通信；由于 I/O 端口有自己的地址，使系统存储器地址范围扩大，适合大系统使用。

I/O 映像的 I/O 寻址方式的缺点：指令少，编程灵活性相对减少；硬件上需要 I/O 端口的译码芯片，增加了硬件开支。

8086 CPU 中采用的是 I/O 映像的 I/O 寻址方式，同时在硬件上区分存储器和 I/O 端口的访问，用 $M/\overline{IO}$ 引脚信号来区分。对于存储器，要求其地址必须是连续的，即在硬件设计时，一个连续的存储器地址空间必须有对应的硬件存储器芯片；而 I/O 端口并不需要这样的要求，即对外设的 I/O 端口并不要求其端口地址必须连续。

6.1.2 I/O 端口地址的形成

1．系统中使用存储器映像的 I/O 寻址方式

使用这种寻址方式时，系统中可以与存储器统一使用译码器芯片。从译码器的输出端既可

接至存储器芯片的片选端形成存储单元地址，也可接至 I/O 接口芯片的控制端或片选端形成 I/O 端口地址。这里，译码器控制端 G1 与 M/$\overline{IO}$ 连接，高电平有效，表示对存储器的访问，实际上还包括了对 I/O 端口的访问。图 6-3 中形成了一个 2K 的存储单元地址和一个 2K 的 I/O 端口地址。根据其硬件连接可知，存储单元地址范围为 00000H～007FFH，I/O 端口地址为 00800H～00FFFH。

2．系统中使用 I/O 映像的 I/O 寻址方式

使用这种寻址方式时，系统中的 I/O 端口地址需要单独的一个译码器芯片，译码器的输出仅允许接 I/O 接口芯片的控制端或片选端。此时，译码器的控制输入端要接 CPU 的 M/$\overline{IO}$ 引脚，且在其上产生低电平时有效。其原因是在执行 IN 或 OUT 指令时，CPU 的 M/$\overline{IO}$ 输出低电平有效信号。使用 I/O 映像的 I/O 寻址时，端口地址仅需要 A_{15}～A_0 这 16 根地址线或 A_7～A_0 这 8 根地址线。图 6-4 中形成了 2 位十六进制的端口地址，第一片 I/O 芯片端口地址为 80H～87H，第二片 I/O 芯片端口地址为 88H～8FH。

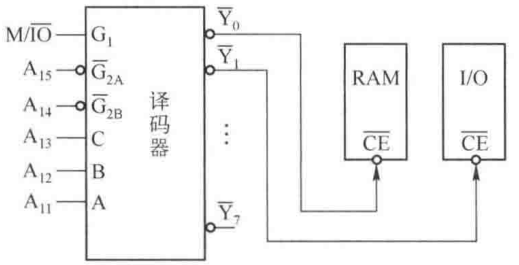


图 6-3　存储器映像 I/O 地址的形成

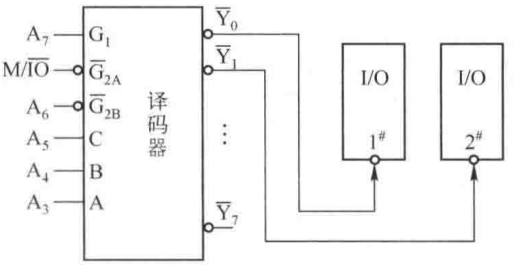


图 6-4　I/O 映像 I/O 地址的形成

6.2　输入方式和输出方式

从 CPU 与外设通信的特点可知，由于外设数据传送速度慢，CPU 不能直接与之进行通信，必须了解外设工作状态后才能决定是否和外设进行数据交流，为此 CPU 在数据传送之前一般要进行状态的“联络”，要么由 CPU 查询外设状态，要么由外设向 CPU 发出请求等，这种方式称为程序控制方式；另一种方式是在程序对专用的输入/输出控制器进行设置后，脱离 CPU 对输入/输出的管理，而由专用控制器完成计算机与外设之间的信息交换，这种方式称为直接存储器存取。一个系统在设计时，会针对不同的外设，采用不同的输入/输出方式。

1．程序控制传输方式

程序控制传输方式是指其状态和数据的传输是由 CPU 执行一系列指令完成的，可分为如下 3 种。

① 同步传输方式：在这种方式下，CPU 直接与外设传输数据并不需要了解外设状态，如按钮开关、发光二极管等。其特点是，外设可以处于 CPU 控制下。

② 异步查询方式：当慢速的外设与 CPU 交换数据时，常用这种方式。

在这种方式下，CPU 与外设传输数据前，先检查外设状态后才可传输数据。外设状态的检查是 CPU 执行一段程序后完成的。

③ 中断方式：在异步查询方式时，CPU 要用大量时间去执行状态查询程序，使 CPU 的

效率大大降低。可以不让 CPU 主动去查询外设的状态，而是让外设在数据准备好后再通知 CPU。这样，CPU 在没接到外设通知前只管做自己的事情，只有接到通知时才执行与外设的数据传输工作。这可大大提高 CPU 的利用率，这种方式称为中断方式。关于输入、输出采用中断方式的内容将在第 9 章中具体介绍。

2．直接存储器存取方式

对于高速的外设以及成块交换数据的情况，采用程序控制传输数据的方法，甚至中断方式传输，都不能满足对速度的要求。因为采用程序控制方式进行数据传输时，CPU 必须加入其中，需要利用 CPU 中的寄存器作为中转。例如，当有数据从外设保存到内存中，首先就必须用 IN 指令将外设的数据送至寄存器（在 8086 中是 AL 或 AX），再使用 MOV 指令将寄存器中的数据送至内存，这样才完成数据从外设到内存的过程。如果系统中大量地采用这种方式与外设交换信息，会使系统效率大大下降，也可能无法满足数据存储的要求，如内存和磁盘间的数据交换。

直接存储器存取（Direct Memory Access，DMA）方式就是在系统中建立一种机制，将外设与内存间建立起直接的通道，CPU 不再直接参加外设与内存间的数据传输，而是在系统需要进行 DMA 传输时，将 CPU 对地址总线、数据总线及控制总线的管理权交由 DMA 控制器进行控制。当完成一次 DMA 数据传输后，再将这个控制权还给 CPU。当然，这些工作都是由硬件自动实现的，并不需要程序进行控制。采用 DMA 方式需要一个硬件 DMAC（称为 DMA 控制器）芯片来完成相关工作，如内存地址的修改、字节长度的控制。当 CPU 放弃数据总线、地址总线及控制总线的控制权时，由 DMAC 实现外设和内存间的数据交换，同时包括与 CPU 之间必要的连接。

6.3 CPU 与外设通信的接口

6.3.1 同步传输方式与接口

同步传输方式，又称为无条件传输方式，主要应用于外设的时序和控制可以完全处于 CPU 控制之下的场合。这类设备必须在 CPU 限定的时间内准备就绪，并且完成数据的发送和接收。在程序中，由于外设总是处于“等待”状态，因此进行数据传输时，只要简单地将 I/O 指令放在程序需要的位置即可。当程序执行到 I/O 指令时，由于外设随时都为传输数据做好准备，因此在此指令执行时间内，就可完成数据的传输。同步传输是最简单的传输方式，与其他输入、输出方式相比，所需的硬件和软件都是最少的。

1．同步输入方式

（1）同步输入过程

<1> 提供端口地址，以便 CPU 从指定的外设中取出数据。

<2> 执行 IN 指令或存储器读指令。

<3> 地址译码器输出，同时产生 M/$\overline{IO}$ 和 $\overline{RD}$ 控制信号。

<4> 数据从端口中输入至 CPU 寄存器。

（2）同步输入硬件接口电路

为了防止 CPU 在取外设数据时，数据发生变化，往往在硬件上采用缓冲器或锁存器，把外设数据保存起来。缓冲器或锁存器是可编程或不可编程的芯片，称为 I/O 接口芯片。硬件接口电路必须保证同步输入过程的正确执行。图 6-5 是一个同步输入的硬件接口电路。

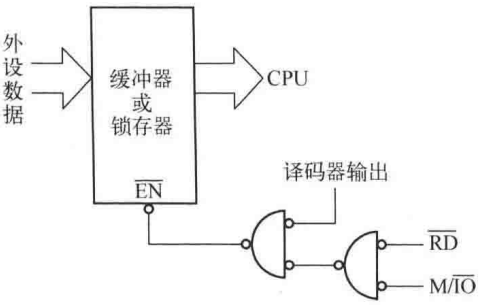


图 6-5　同步输入的硬件接口电路

（3）缓冲器 74LS244

74LS244 是一种具有三态输出的 8 位缓冲器，具有 20 个引脚的双列直插式 TTL 芯片。图 6-6 是其引脚图，有 2 个低电平有效的片选端 $\overline{CE_1}$ 和 $\overline{CE_2}$，8 个输入端 $D_7 \sim D_0$ 和 8 个输出端 $Q_7 \sim Q_0$ 分成两组，每组 4 位；$\overline{CE_1}$ 和 $\overline{CE_2}$ 信号作为两个 4 位缓冲器的控制端，$\overline{CE_1}$ 和 $\overline{CE_2}$ 信号连接在一起，可将一片 74LS244 作为 8 位缓冲器使用，其内部结构实质上是 8 个带“允许输出”的三态器件，仅能用于输入接口。图 6-7 是它作为一个 8 位输入接口的电路图，输入外设为按键开关。

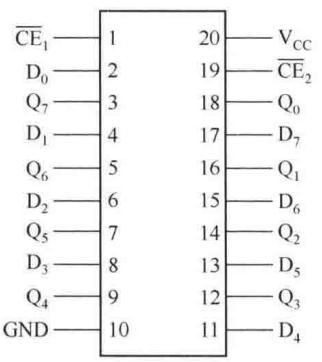


图 6-6　74LS244 引脚

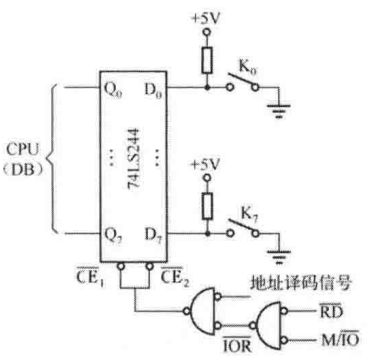


图 6-7　采用 74LS244 实现的输入接口电路

2．同步输出方式

（1）同步输出过程

<1> 提供端口地址，以便 CPU 将数据送到指定的外设。

<2> 执行 OUT 指令或存储器写指令。

<3> 地址译码器输出，同时产生 M/$\overline{IO}$ 和 $\overline{WR}$ 信号。

<4> CPU 将数据输出到端口。

（2）同步输出硬件接口电路

由于 CPU 数据线上挂接的负载很多，为了将 CPU 数据线上的信息准确传输，除了正确提供端口地址，还需将数据锁存或驱动后提供给外设。图 6-8 是一个同步输出的硬件接口电路。

（3）8 位 D 锁存器 74LS273

74LS273 是 8 位 D 锁存器，具有 20 个引脚的双列直插式 TTL 芯片，图 6-9 是其引脚图。它具有清零端 CLR 和锁存控制端 $\overline{CP}$，只有当 $\overline{CP}$ 端具有低电平有效信号时，$D_7 \sim D_0$ 输入端上的信号才会被锁存到 74LS273 内，并在 $Q_7 \sim Q_0$ 的输出端上输出；当 $\overline{CP}$ 端为高电平无效信号时，原被锁存的信号不会因输入端 $D_7 \sim D_0$ 上信号的变化而改变。74LS273 芯片适合作为输出接口，图 6-10 是用一片 74LS273 组成的输出接口电路图，输出外设为发光二极管。

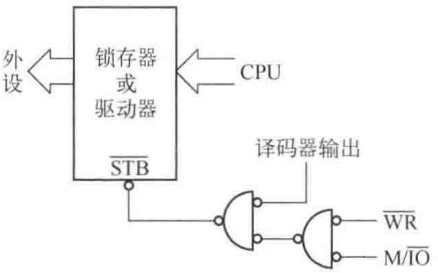


图 6-8　同步输出接口电路

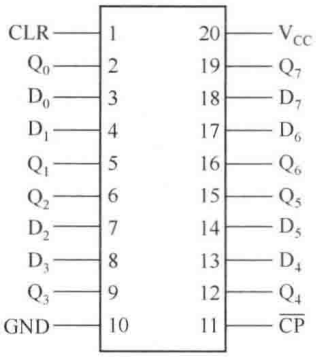


图 6-9　74LS273 引脚

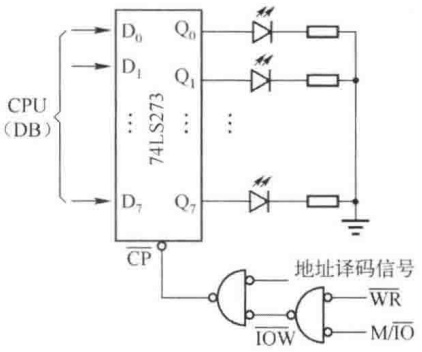


图 6-10　输出接口电路

6.3.2　异步查询方式与接口

在大多数情况下，外设一般不会处于 CPU 控制下，常常是 CPU 与外设工作不能同步。由于外设不能总处于“准备好”状态，如果仍采用同步方式，就会出现数据丢失。一个简单的方法是采用异步查询的方式，CPU 与外设之间通过“握手”信号进行交流，以确保数据传输的准确性。

异步查询方式又称为条件传输方式。当 CPU 与外设用异步查询方式通信时，要求外部设备提供状态信息。状态信息是通过状态端口检测的。当状态满足条件时，CPU 从数据端口与外设交换数据；当状态不满足条件时，CPU 则不断从状态端口检测状态，直至状态满足为止。

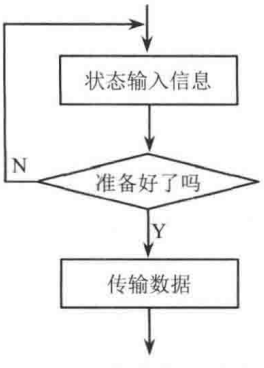


图 6-11　异步输入的查

1．异步查询输入方式与接口

当 CPU 从慢速的外设取数时，取数之前需要查询外设是否把数据准备好。也就是说，当外设的数据准备好时，应发出相应的状态信息，如$\overline{\text{STB}}$信号，将数据锁存起来。在 CPU 用 IN 指令从数据端口读取数据之前，首先要用 IN 指令从状态端口读取状态，并且根据状态信息 RDY 获得是否有数据需要读入。异步查询输入方式的查询框图如图 6-11 所示，为异步查询输入设计的状态端口和数据端口电路如图 6-12 所示。

下面讨论图 6-12 电路的工作原理。外设数据准备好时发出低电平有效的状态信号。该信号有两个作用：第一，作为 8 位锁存器的控制信号，当$\overline{\text{STB}}$为低电平时，外设将准备好的数据锁存起来，且锁存器的输出为缓冲器的输入；第二，$\overline{\text{STB}}$信号使 D 触发器的输出端 Q 变成高电平，该高电平为缓冲器 1 的输入信号。至此，外设输出的$\overline{\text{STB}}$信

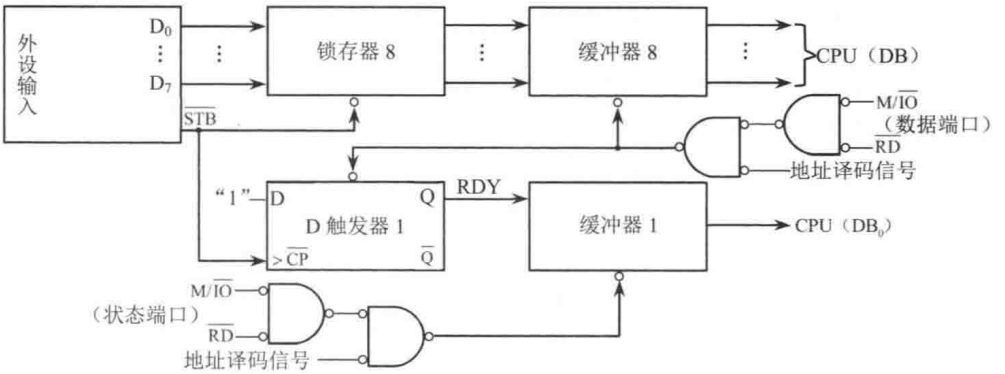


图 6-12　异步查询输入的状态端口和数据端口电路

号使数据锁存，使状态信号保存于缓冲器 1 中。缓冲器 1 的输出接于 CPU 数据线 D_0 位上，执行一条 IN 指令，从状态端口取数，测试其 D_0 位是否为“1”。若 D_0=1，则表明状态满足条件，从而可执行从数据端口取数指令，CPU 将数据从缓冲器 8 取走的同时，将 D 触发器的 Q 端清零，使外设状态条件不满足。当外设再次发出低电平 $\overline{STB}$ 信号时，准备好的数据被锁存，状态信号又使 D 触发器的 Q 端变为“1”，CPU 检查状态满足后，进行下一个数据的读取。与图 6-12 相应的软件查询程序如下：

```
SPORT    EQU    300H                    ; 状态端口
DPORT    EQU    310H                    ; 数据端口
         ……
TEST1:   MOV    DX, SPORT
         IN     AL, DX                  ; 读取状态信息
         TEST   AL, 01                  ; 检查 D0 位
         JZ     TEST1                   ; 为 0，表示无数据输入
         MOV    DX, DPORT               ; 为 1，读入数据
         IN     AL, DX
         ……
```

2．异步查询输出方式与接口

当 CPU 将数据送到外部设备时，由于 CPU 执行速率很快，外设能否及时把数据取走是解决问题的关键。若外设没有取走前一个数据，CPU 就不能立即输出下一个数据，否则数据就会丢失（覆盖）。因此，外设取走一个数据就要发出一个状态信息，表示缓冲区的数据已被外设取走。CPU 要输出下一个数据，仍要读取状态端口的状态信息，才决定是否输出下一个数据。异步查询输出方式的查询框图如图 6-13 所示。异步查询输出设置的状态端口和数据端口电路如图 6-14 所示。

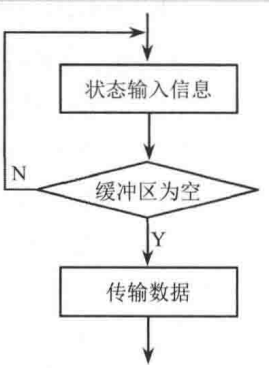


图 6-13　异步输出的查

下面讨论图 6-14 的工作原理。当 CPU 执行一条 OUT 指令，将数据输出到锁存器中时，会使 D 触发器的 Q 端输出高电平有效信号 OBF，表示输出缓冲区“满”，通知外设可以取数。外设取走数据后，发出一个回答信号 $\overline{ACK}$，这个低电平有效的 $\overline{ACK}$ 信号清除 D 触发器，使

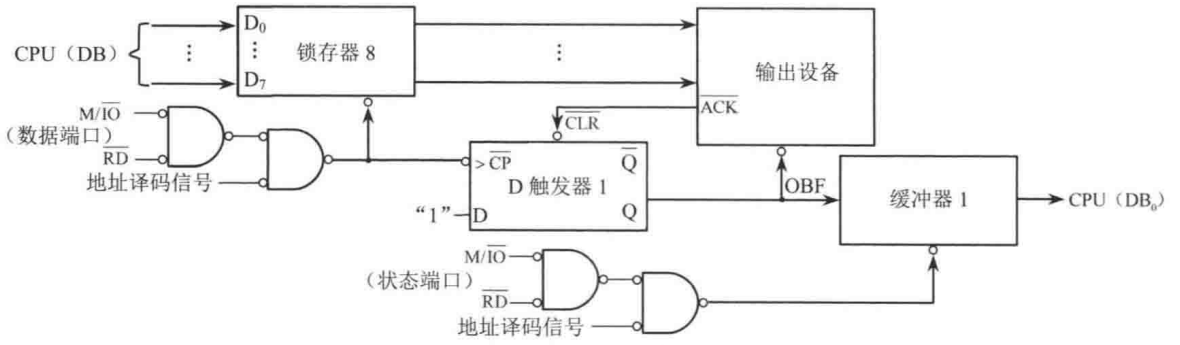


图 6-14　异步查询输出接口电路图

D 触发器的 Q 端输出为 0，也使得状态缓冲器的输出为 0。状态信号接到 CPU 的数据总线 D_0 位上，CPU 执行一条取状态指令，测试其 D_0 位。若 $D_0=0$，表明缓冲区空，外设已将前一个数据取走，CPU 可以输出下一个数据；否则，CPU 要一直测试状态信息 D_0 位，若 $D_0=1$，则 CPU 不会送下一个数据，以免前面的数据被覆盖而丢失。

与图 6-12 对应的软件查询程序如下：

```
SPORT   EQU   300H                    ; 状态端口
DPORT   EQU   310H                    ; 数据端口
        MOV   DX, SPORT
TEST2:  IN    AL, DX                  ; 取状态信息
        TEST  AL, 01
        JNZ   TEST2                   ; 区数据未被读取，继续等待
        MOV   DX, DPORT
        MOV   AL, [BX]                ; 待输出的数据在缓冲区
        OUT   DX, AL                  ; 输出数据
```

6.4　8086 CPU 的输入和输出

1．8086 CPU 的 I/O 指令

如前所述，8086 CPU 采用 I/O 映射的 I/O 指令，在 AL 或 AX 寄存器与 I/O 端口之间进行传输。8086 的 I/O 端口寻址包括直接寻址和 DX 寄存器间接寻址两种。输入、输出指令的直接寻址是指仅用低 8 位地址线 $A_7 \sim A_0$ 译码产生的 I/O 端口地址，因而端口地址仅是 2 位十六进制数，即仅可访问 256 个端口。此时，$A_{15} \sim A_8$ 的输出为 0。用 DX 寄存器间接寻址，则由 $A_{15} \sim A_0$ 地址线译码产生 I/O 端口地址，此时端口地址为 4 位十六进制数，这样 8086 CPU 可有 64K 个端口寻址。在 CPU 访问 I/O 时，地址线 $A_{19} \sim A_{16}$ 均输出低电平。

直接寻址输入/输出指令（8 位端口地址）如下：

```
        IN    AL, n                   ; 字节输入
        IN    AX, n                   ; 字输入
        OUT   n, AL                   ; 字节输出
        OUT   n, AX                   ; 字输出
```

DX 寄存器间接寻址输入（16 位端口地址）如下：

```
IN      AL, DX                    ; 字节输入
IN      AX, DX                    ; 字输入
OUT     DX, AL                    ; 字节输出
OUT     DX, AX                    ; 字输出
```

注意，作为寄存器间接寻址功能的 DX 在 IN 和 OUT 指令中没有像 MOV 指令格式那样把地址寄存器用中括号“括”起来。这是为什么呢？因为在 IN 和 OUT 指令中，DX 功能很单纯，它的功能只能是间接寻址，所以不需要加括号了。

2．8086 CPU 的 I/O 特点

8086 CPU 和 I/O 接口电路之间的数据通路是时分多路复用的地址/数据总线。在采用 I/O 独立编址方式时，8086 只能用地址线 A_{15}～A_0 来寻址端口，其他控制信号有 ALE、$\overline{BHE}$、$\overline{WR}$、$\overline{RD}$、M/$\overline{IO}$、DT/$\overline{R}$ 和 $\overline{DEN}$。

由于 8086 CPU 有两种工作模式，当工作在不同模式时，其控制信号会发生变化。具体地说，当 8086 CPU 工作在最小模式时，输入、输出的控制信号由 CPU 直接提供，如图 6-15 所示。当 8086 CPU 工作在最大模式时，输入、输出的某些控制信号由 CPU 的状态线 $\overline{S}_0$、$\overline{S}_1$ 和 $\overline{S}_2$ 经过总线控制器芯片 8288 译码产生，如图 6-16 所示。

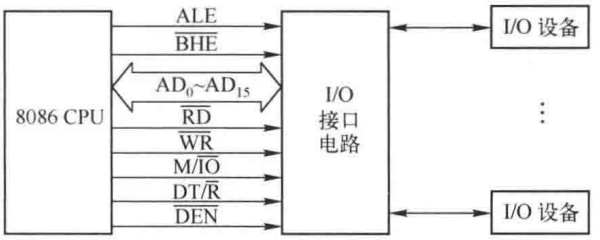


图 6-15　8086 最小模式时的 I/O 接口

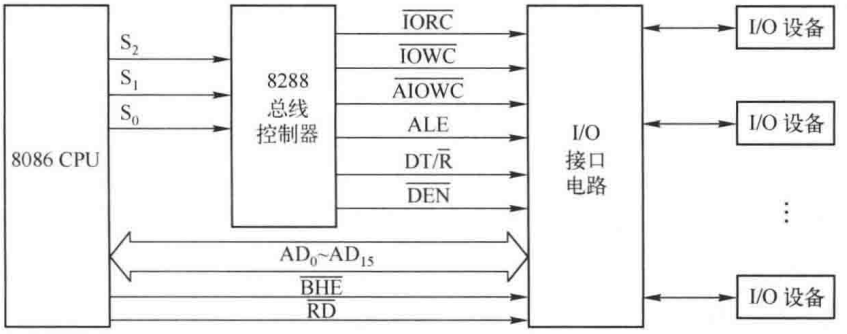


图 6-16　8086 最大模式时的 I/O 接口

8086 CPU 与外设交换数据可按字或字节进行。当按字节进行时，偶地址端口的字节数据由低 8 位数据线 D_7～D_0 位传输，奇地址端口的字节数据由高 8 位数据线 D_{15}～D_8 传输。当用户在安排外设的端口地址时，如果外设是以 8 位方式与 CPU 连接，就只能将其数据线与 CPU 的低 8 位连接，或者只能与 CPU 的高 8 位连接。这样，同一台外设的所有寄存器端口地址都只能是偶地址或者奇地址，所以设备的端口地址往往是不连续的。

习 题 6

1．CPU 与外部设备通信为什么要使用接口？

2．I/O 端口有什么用途？

3．I/O 端口有哪两种寻址方式？各有何优缺点？

4．在某 8086 微机系统中有一个外设，使用存储器映像的 I/O 寻址方式，该外设地址为 01000H。试画出其译码器的连接电路，使其译码器输出满足上述地址要求，译码器使用 74LS138 芯片。

5．微机系统的输入/输出指什么？输入和输出有什么不同？

6．输入、输出有哪几种方式？各有何优缺点？

7．在输入、输出的电路中，为什么常常要用锁存器和缓冲器？

8．采用异步查询方式时，输入查询和输出查询有什么不同？

9．8086 CPU 在执行输入、输出指令时，CPU 的哪些控制引脚起作用，什么样的电平有效？

10．8086 CPU 分配的端口地址有何限制？为什么？

11．现有 I/O 映像的输入设备，使用软件查询方式与 CPU 通信。当状态位 D_0=l 时，为 1 号设备输入字符；当状态位 D_1=1 时，为 2 号设备输入字符；当状态位 D_3=l 时，为 1 号设备输入结束；当状态位 D_4=l 时，为 2 号设备输入结束。设状态端口地址为 0624H，1 号设备数据端口地址为 0626H，2 号设备端口地址为 0628H，输入字符缓冲区首地址分别为 BUFFERl 和 BUFFER2，试编程序完成该功能的查询程序。

12．有 8 个发光二极管，其阴极上加低电平则亮，用 74LS273 芯片作为 I/O 接口与 8086 CPU 通信。要求这些二极管同时亮或灭，同时二极管亮和灭的时间分别为 50 ms 和 20 ms。试画出其硬件接口电路，并编写程序完成上述要求（时间控制可调用软件延时子程序）。

13．同上题，要求发光二极管由低位向高位依次循环显示，每个二极管显示时间为 1 秒。编写实现该功能的程序。

14．现有一台硬币兑换器，平时等待纸币输入。当状态端查到 D_2=1 时，表示有纸币输入，此时可从输入数据端口中读出纸币面值，一角纸币代码为 01，二角纸币代码为 02，五角纸币代码为 03。当状态端口 D_3=1 时，把兑换的 5 分硬币数（十六进制）从输出数据端口输出。设状态端口地址为 03FAH，数据输入端口地址为 03FCH，数据输出端口地址为 03FEH。画出其接口电路示意图，并编程完成以上要求。

第 7 章　可编程接口芯片

本章导读

- 芯片 8255A 的内部结构和引脚分配
- 芯片 8255A 的工作方式及编程
- 芯片 8255A 的三种工作方式的功能
- 芯片 8255A 的应用举例
- 芯片 8253 的内部结构和引脚分配
- 芯片 8253 的工作方式及编程
- 芯片 8253 的应用举例

计算机系统是由软件和硬件组成的。软件的特点是具有极高的灵活性，只要硬件允许，用户就可通过编程构成任意功能的软件。硬件则很不灵活，一旦电路设计完成，其功能也就确定了，很难更改。这样，在一定程度上降低了一个计算机系统功能的发挥。当然，我们也希望硬件接口电路最好具有一定的可变性，即希望存在这样一种芯片：当这个芯片与 CPU 三总线相连后，尽管电路不可能改变，但其功能可以通过程序来改变。比如在设计某一端口时，使其同时具有输入和输出的能力，用户可以根据需要通过指令来选择输入接口或输出接口，就可以大大提高计算机系统的灵活性。这种可被用户通过程序来改变其功能的电路芯片称为可编程芯片，而用程序改变芯片工作方式的过程称为芯片编程或芯片初始化。

为了便于读者的理解，图 7-1 给出了一个简单的具有输入功能和输出功能的可编程接口电路方案。这个电路包括：一个输入接口，其组成主要是 8 位三态门；一个输出接口，其组成主要是 8 位锁存器；还有 8 位多路转换开关及控制这个开关的寄存器 FF。

当寄存器 FF 为 0 时，多路转换开关接位置“0”，I/O 线接锁存器，这个电路就可作为输出接口；当寄存器 FF 为 1 时，多路转换开关接位置“1”，I/O 线接三态门，这个电路就可作为输入接口。这样，用户通过指令把寄存器 FF 写入“0”或“1”，就可选取所需的接口工作状态。

上述方案就是可编程芯片设计的主要思想。用户对寄存器 FF 写入的内容称为命令字或方式控制字，而寄存器 FF 称为命令寄存器，相应的端口称为命令端口或控制端口。对可编程芯片初始化的过程实际上就是对芯片的控制端口写入各种命令字的操作过程。

尽管上述电路比较简单，但具有可编程器件的共性。比如，器件内部已集成用户可选的各种功能的电路模块及其可以控制选择这些功能的命令寄存器。

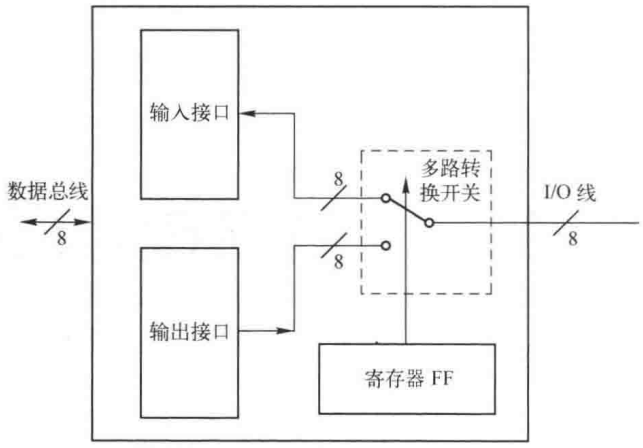


图 7-1　一个可编程接口电路方案

目前常用的可编程芯片有如下几种：8255A，并行 I/O 接口；8253，计数/定时器，8251A，串行 I/O 接口；8259A，中断控制器。本章主要介绍 8255A 和 8253 芯片，8251A 将在第 8 章介绍，8259A 将在第 9 章介绍。

7.1　可编程并行接口芯片 8255A

8255A 是一种通用的可编程并行 I/O 接口芯片，广泛用于几乎所有系列的微机系统中，如 8086、MCS51、Z80 等。8255A 具有 3 个带锁存或缓冲的数据端口，可与外设并行进行数据交换。用户可用程序来选择多种操作方式，通用性强，使用灵活，可为 CPU 与外设之间提供并行输入/输出通道。8255A 的各端口内具有中断控制逻辑，在外设与 CPU 之间可用中断方式进行信息交换，使用条件传输方式时可用“联络”线进行控制。8255A 的并行数据宽度为 8 位。

7.1.1　8255A 的内部结构

8255A 内部结构框图如图 7-2 所示。在 8255 内部结构中，厂家除了把输入/输出接口电路集成在一块芯片中，还包括控制这些接口电路的控制部分及与 CPU 接口的总线接口部分。

1．并行输入/输出端口

8255A 芯片中包含 3 个 8 位端口：A 口、B 口和 C 口。这 3 个端口均可作为 CPU 与外设通信时的缓冲器或锁存器。一般来说，它们作为缓冲器使用时，就是输入接口；作为锁存器使用时，就是输出接口。

条件传输方式需要“状态”或“联络”信号，中断传输方式需要“中断”信号。由于 8255A 没有预先从芯片引脚上给出这些信号，因此当用户选择这两种工作方式时，8255A 将从 C 口的 8 位 I/O 线中提取若干根线作为“状态”“联络”或“中断”线。在这种情况下，C 口剩余的线仍然可以作为 I/O 线。3 个端口通过各自的 I/O 线与外设联系。

2．A 组和 B 组控制

8255A 有 3 个端口，但不是每个端口都有自己独立的控制部件。实际上，它只有两个控制

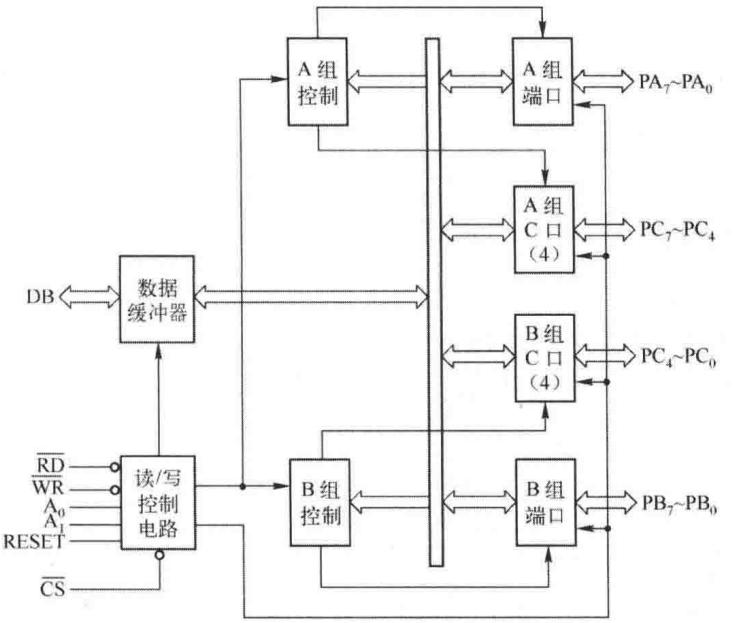


图 7-2　8255A 内部结构

部件，这样 8255A 内部的 3 个端口就分为两组。A 组由 A 口和 C 口的高 4 位组成，B 组由 B 口和 C 口的低 4 位组成。A 组和 B 组分别有自己的控制部件，可同时接收来自读/写控制电路的命令和 CPU 送来的控制字，并且根据它们来定义各个端口的操作方式。

3．数据总线缓冲器

双向三态的 8 位数据缓冲器实现 8255A 与 CPU 之间的数据传输接口。CPU 执行输出指令时，可将控制字或数据通过该缓冲器传送给 8255A 的控制口或数据端口；CPU 执行输入指令时，8255A 可将数据端口的状态信息或数据通过它传输给 CPU。因此，数据总线缓冲器是 CPU 与 8255A 交换信息的必经之路。

4．读/写控制电路

8255A 的读/写控制电路接收来自 CPU 的控制命令，并根据命令向片内各功能部件发出操作命令。例如，片选信号 $\overline{CS}$ 为低电平时，表示 8255A 芯片被选中。该片选信号则由 CPU 的地址线通过译码产生。读、写信号 $\overline{RD}$ 和 $\overline{WR}$ 控制 8255A 与 CPU 之间的数据或信息传输方向。端口选择控制则由 A1 和 A0 的组合状态提供，由这两个控制信号可提供 4 个端口地址，即 A、B、C 三个端口地址及一个控制端口地址。8255A 可用 RESET 控制信号复位，当该控制信号有效时，清除 8255A 中所有控制寄存器内容，并将各端口置成输入方式。

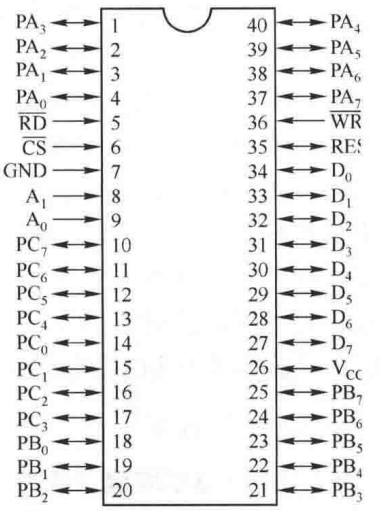


图 7-3　8255A 引脚

7.1.2　8255A 的引脚

8255A 是一个标准的 40 引脚芯片（如图 7-3 所示），可分为三部分：与外设连接的 I/O 线，与 CPU 连接的系统总线，

以及电源线。

1．与外设连接的引脚

8255A 有 3 个数据端口，每个端口 8 位，由此推算，与外设相连接的引脚共 24 位。其中，A 口、B 口、C 口均有 8 个 I/O 引脚（$PA_7 \sim PA_0$，$PB_7 \sim PB_0$，$PC_7 \sim PC_0$）。$PC_7 \sim PC_0$ 中可有若干根复用线用于“联络”信号或状态信号，其具体定义与端口的工作方式有关。

2．与 CPU 连接的引脚

与 CPU 连接的引脚包括：数据线 $D_7 \sim D_0$，读写控制线 $\overline{RD}$ 和 $\overline{WR}$，复位线 RESET，以及与 CPU 地址线相连接的片选信号 $\overline{CS}$、端口地址控制线 A_0 和 A_1。

一般，CPU 的数据线及其读写控制线直接与 8255A 的 $D_7 \sim D_0$ 及 $\overline{RD}$ 和 $\overline{WR}$ 连接。

RESET 线是高电平有效。因为 8086 CPU 也是高电平复位，所以可以直接与 8086 CPU 的复位线相连。有时为了便于调试，8255A 复位电路与 CPU 的复位电路也可以是分开的。

片选信号 $\overline{CS}$ 是低电平有效。当其有效时，表示本片的 8255A 被 CPU 选中，可以工作。$\overline{CS}$ 一般由 CPU 的高位地址线及其地址译码电路产生。

表 7-1　A_0、A_1 的组合

A_1 A_0	端　口
0　0	A 口地址
0　1	B 口地址
1　0	C 口地址
1　1	控 制 口

A_0 和 A_1 的组合状态如表 7-1 所示，可以选择 8255A 的 3 个 I/O 端口和控制口，一般由 CPU 的低位地址线直接产生。

3．电源线和地线

8255A 的电源引脚为 VCC 和 GND。VCC 为电源线，一般取+5 V。GND 为地线。

7.1.3　8255A 的工作方式及编程

1．8255A 的工作方式

8255A 有 3 种工作方式：方式 0，基本输入/输出方式；方式 1，选通输入/输出方式；方式 2，双向传输方式。

方式 0 主要工作在无条件的输入/输出方式下，不需要“联络”信号。A 口、B 口和 C 口均可工作在此方式下。在方式 0 下，C 口的输出位可由用户直接独立设置为“0”或“1”。

方式 1 主要工作在异步或条件传输方式（必须先检查状态，然后才能传输数据）下。此时，仅有 A 口和 B 口可工作于方式 1。由于条件传输需要联络线，所以在方式 1 下 C 口的某些位分别为 A 口和 B 口提供 3 根联络线。

方式 2 的双向传输方式是指在同一端口内分时进行输入/输出的操作。8255A 中只有 A 口可工作在这种方式下，此时需要 5 个控制信号进行“联络”，由 C 口提供，所以此时 B 口只能工作在方式 0 或方式 1 下。当 B 口工作在方式 1 下时，又需要 3 根联络线。所以当 A 口工作在方式 2 下，同时 B 口又工作在方式 1 下时，8255A 的 C 口 8 根线将全部作为联络线使用，C 口也就因没有 I/O 功能而“消失”了。关于 C 口“联络”信号的定义下面将详细讨论。

2．8255A 编程

所谓 8255A 编程，就是用户在使用 8255A 前，可用软件来定义端口的工作方式，选择所需的功能。掌握 8255A 编程是正确使用该芯片的前提，为此须先了解 8255A 的控制命令。

（1）方式控制字

这是一个8位的控制字，代表的信息非常丰富。上面提到8255A内部的3个端口分为A、B两组，因此方式控制字也就相应地分成两个部分，分别控制A组和B组，其格式如图7-4所示。

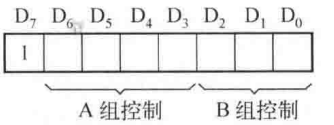


图7-4　方式控制字

D_7=1，为该控制字的标志。A组由D_6、D_5、D_4和D_3组成。其中，D_6和D_5为工作方式选择，见表7-2；D_4为A口的输入、输出选择，见表7-3；D_3为C口高4位的输入、输出选择，见表7-4。

B组由D_2、D_1和D_0组成。其中，D_2为工作方式选择，见表7-5；D_1为B口的输入/输出选择，见表7-6；D_0为C口低4位的输入/输出选择，见表7-7。方式控制字未规定C口的工作方式，只规定了C口数据的传输方向，这就表明C口要么作为联络线用，要么只能工作在方式0下。

表7-2　A口工作方式选择

D_6 D_5	A口工作方式
0 0	方式0
0 1	方式1
1 X	方式2

表7-3　A口I/O选择

D_4	输入/输出选择
0	A口为输出
1	A口为输入

表7-4　C口上半部I/O选择

D_3	输入/输出选择
0	C口高4位为输出
1	C口高4位为输入

表7-5　B口工作方式选择

D_2	B口工作方式
0	工作方式0
1	工作方式1

表7-6　B口I/O选择

D_1	输入/输出选择
0	B口为输出
1	B口为输入

表7-7　C口下半部I/O选择

D_0	输入/输出选择
0	C口低4位为输
1	C口低4位为输

例如，某个外设工作在无条件传输方式，通过8255A与CPU交换数据，要求A口、B口为输出，C口为输入，则方式控制字如下：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1

如果对8255A的控制端口写入10001001B（89H），则8255A的3个I/O口就会工作在用户所设定的功能上，即A口、B口为输出，C口为输入。

（2）C口置“1”/清“0”控制字

置“1”又称为置位操作，而清“0”称为复位操作。我们举一个例子，让读者先有一个置位/复位的概念。例如，在某个外设接口电路中，8255A的C口为输出，控制8个继电器。设定C口的I/O线为“1”，表示继电器闭合；为“0”，表示继电器断开。现在要求某个继电器闭合，如与PC_2对应的继电器闭合，而其他继电器状态不变。如何实现？一个可选的方案如下：

```
IN      AL, C口                    ; 取C口开关信息
OR      AL, 00000100B              ; 设置AL的第2位为1
OUT     C口, AL                    ; 重设C口开关状态
```

虽然上面程序段的第3条语句对C口的8根线重新进行了设置，但由于前两条语句操作的实际效果是只有PC_2发生了变化，即PC_2=1，所以C口的其他位并未发生变化。另外，8255允许CPU对其设置输出的端口进行“回”读操作，如上述程序段的“IN　AL,C口”，这时读

到 CPU 的内容是前面时刻的 CPU 写到 C 口的内容。

如果要求 PC_2 对应的继电器由闭合转为断开，那么把语句“OR　AL, 00000100B”改为“AND　AL, 11111011B”即可。这就是典型的置位/复位操作，其特点是只有指定位发生变化，其他位保持不变。

对于上述例子，有一个更直接的方案来实现，就是采用 8255A 的 C 口置“1”/清“0”控制字，它仅对 C 口有效，可以直接对 C 口的指定位置“1”或清“0”。其格式如图 7-5 所示。

$D_7=0$ 为该控制字的标志。

$D_6 \sim D_4$ 没有定义，可为随意态，通常取 0。

$D_3 \sim D_1$ 的 8 种编码对应 C 口的 $PC_7 \sim PC_0$ 位，如 $D_3D_2D_1$ =010，表示要对 PC_2 进行置位/复位操作。

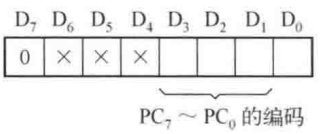


图 7-5　置位复位控制字

$D_0=0$，则将 $D_3D_2D_1$ 编码对应的 PC_i 位清“0”。$D_0=1$，则将 $D_3D_2D_1$ 编码所对应的 PC_i 位置“1”。

用 8255A 的置位/复位控制字完成上例要求，即 $PC_2=1$ 的程序段如下：

```
MOV     AL, 00000101
OUT     控制口, AL
```

显然，这种操作更简捷。**注意**：该控制字是通过写入 8255A 的控制寄存器来达到对 C 口的指定位进行置位/复位操作的。

（3）读入状态字

当 8255A 由程序设定在方式 1 或方式 2 工作时，C 口根据不同的情况产生或接收“联络”信号。如果此时对 C 口进行读操作，那么读出的内容就包含两部分内容：一部分是那些作为 I/O 线上的内容，另一部分是与“联络”状态有关的内容。通过读取 C 口的内容，程序员可测试或检查外部设备的状态，相应地改变程序流程。图 7-6 给出了 3 个状态字的格式。

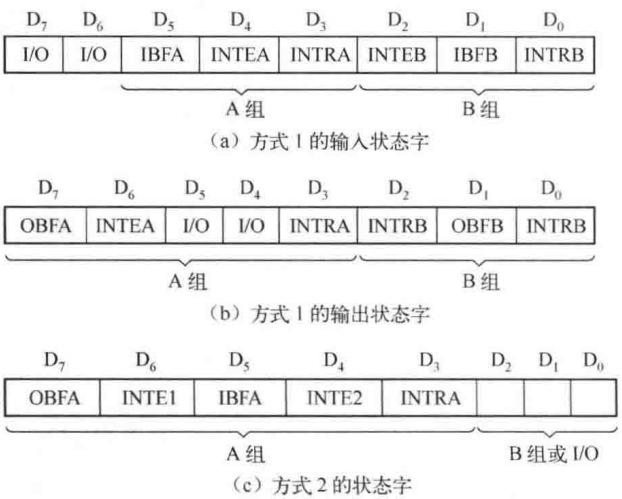


图 7-6　C 口状态字

有关状态字中各符号的意义，将在后面有关章节中结合实例进行详细说明。

7.1.4　8255A 的功能

下面介绍 8255A 的 3 种工作方式，重点是方式 0。

1．8255A 工作在方式 0 下

8255A 的 3 个端口均可工作在这种方式下，特别是 C 口，只能工作在方式 0 下。在这种方式下，CPU 与端口之间交换数据可以直接由 CPU 执行 IN 或 OUT 指令来完成，不需检测状态线。由于在方式 0 下，8255A 的 3 个端口可以分别定义为输入或输出端口，因此 8255A 的 3 个端口可有如表 7-8 所示的 16 种输入/输出组合。

表 7-8　方式 0 下 A 口、B 口和 C 口的输入/输出组合

序　号	A 口	B 口	C 口高 4 位	C 口低 4 位
0	输出	输出	输出	输出
1	输出	输出	输出	输入
2	输出	输出	输入	输出
3	输出	输出	输入	输入
4	输出	输入	输出	输出
5	输出	输入	输出	输入
6	输出	输入	输入	输出
7	输出	输入	输入	输入
8	输入	输出	输出	输出
9	输入	输出	输出	输入
10	输入	输出	输入	输出
11	输入	输出	输入	输入
12	输入	输入	输出	输出
11	输入	输出	输入	输入
12	输入	输入	输出	输出
13	输入	输入	输出	输入
14	输入	输入	输入	输出
15	输入	输入	输入	输入

这意味着当 8255A 与 CPU 总线相连接后，可以提供用户 16 种不同功能的输入/输出端口。可见，采用可编程芯片作为接口电路可以大大提高计算机硬件系统的灵活性。对于 C 口，其高 4 位和低 4 位的输入输出方向可以是独立设置的。

【例 7-1】 8255A 的 A 口和 B 口工作在方式 0 下，A 口为输入端口，接有 4 个开关，B 口为输出端，接有一个七段发光二极管，连接电路如图 7-7 所示。编写一个程序，要求七段发光二极管显示开关所拨通的值。

本例中，8255A 的端口地址由两部分电路构成。由 CPU 高位地址线 A_{15} ~ A_3 通过 74LS138 译码器产生片选信号，CPU 的低位地址线 A_2 和 A_1 分别组合成 4 个端口地址，而 CPU 的 A0 与译码器输出端 $\overline{Y}_4$ 通过逻辑组合，保证 8255A 的 4 个端口地址为偶地址。这样，8 位的 8255A 与 16 位的 8086 CPU 可以通过数据总线 D_7 ~ D_0 传输 8 位信息。具体端口地址分配为：A 口地址，8020H；B 口地址，8022H；C 口地址，8024H；控制口地址，8026H。

七段发光二极管为共阳极 LED 器件。要让 a 段点亮，要求从 PB0 输出高电平“1”；要使 b 段熄灭，要求从 PB1 输出低电平“0”。其余各段以此类推。8255A 的 A 口接有开关，4 位开关的组合可为 0H～FH。为此，可将在 LED 上显示 0H～FH 各字符的段码列于表 7-9 中。

本例的一个关键点就是控制字的确定：根据本例的要求，8255 的 A 口为输入，B 口为输出，C 口未用。这样方式控制字为 1001X00XB，这里 X 表示可以为 0 也可以为 1。建议读者取 1，即方式控制字为 10011001B，即 C 口为输入。这样选取的好处是 I/O 线为输入时，其对外电路曾现高阻态，芯片和外电路之间影响最小。

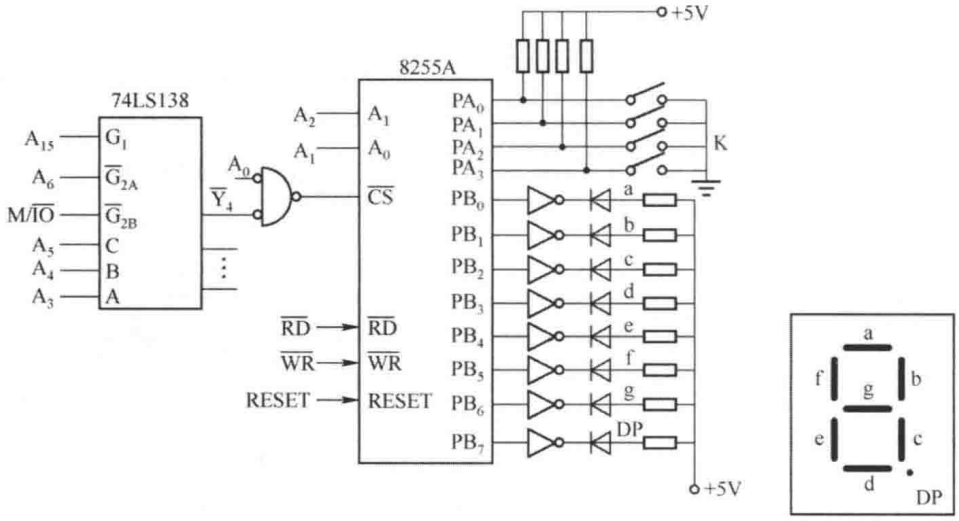


图 7-7　方式 0 举例

表 7-9　段码表

显示字符	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
七段代码（H）	3F	06	5B	4F	66	6D	7D	07	7F	6F	77	7C	39	5E	79	31

源程序如下：

```
A_PORT    EQU     8020H                          ; 定义端口的符号地址
B_PORT    EQU     8022H
C_PORT    EQU     8024H
CTRL_P    EQU     8026H
DATA      SEGMENT                                ; 定义数据段
TAB1      DB      3FH, 06H, 5BH, 4FH, …, 31H     ; 定义段码表
DATA      ENDS
CODE      SEGMENT                                ; 定义代码段
          ASSUME  DS:DATA, CS:CODE
START:    MOV     AX, DATA
          MOV     DS, AX
          MOV     AL, 90H                        ; 设置 8255A 方式字
          MOV     DX, CTRL_PORT
          OUT     DX, AL
AGAIN:    MOV     DX, A_PORT
          IN      AL, DX                         ; 取键盘信息
          AND     AL, 0FH                        ; 屏蔽高 4 位
          MOV     BX, OFFSET TAB1                ; 取段码表首地址
          XLAT                                   ; 查表得段码
          MOV     DX, B_PORT                     ; 输出显示
          OUT     DX, AL
          MOV     CX, 0600H
DELAY:    LOOP    DELAY                          ; 循环延时
          JMP     AGAIN
CODE      ENDS
          END  START
```

2．8255A 工作在方式 1 下

工作方式 1 被称为选通输入/输出方式。在这种工作方式下，数据的输入、输出操作要在选通信号控制下完成，适合条件传输。A 口和 B 口可工作在此方式下，这时 C 口的某些位就用来作为“联络线”。工作在方式 1 的 A 口和 B 口可以作为输入接口，也可作为输出接口。由于输入接口和输出接口所需的选通控制不同，相应“联络线”的定义及功能也不同，因而有必要分别介绍。

（1）选通输入方式

在这种方式下，A 口或 B 口需有 C 口 3 根线（如图 7-8 所示）。以 A 口为例（B 口的联络线可参阅图 7-8），PC_3、PC_4 和 PC_5 作为 A 口的“联络”线。PC_3 称为 INTRA，PC_4 称为 $\overline{STBA}$，PC_5 称为 IBFA。具体功能如下。

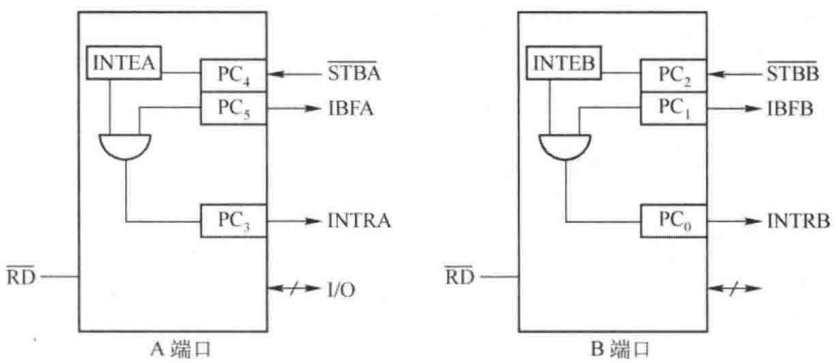


图 7-8　8255A 方式 1 输入

INTRA：A 口的中断请求信号。当其有效时，8255A 的 A 口向 CPU 申请中断，要求 CPU 从 A 口取数。

IBFA：A 口的输入缓冲器“满”信号。当其有效时，表示 A 口的输入缓冲器已暂存一个有效数据。

$\overline{STBA}$：A 口的选通信号（外设发出，低电平有效）。当其有效时，外设把数据输入 A 口的输入缓冲器。

在这种方式下，A 口的逻辑电路有一个 INTEA 信号，称为中断允许信号。当这个信号为逻辑高时，与门导通，INTEA 信号就可以作为中断信号了。INTEA 对应于 C 口输出锁存器的第 4 位，即通过对 C 口 PC_4 的置位/复位就可以设置 INTEA 是逻辑高或低了。

在方式 1 下，外设把一个数据通过 A 口送给 CPU 的过程如下：

<1> 外设把数据送到 A 口的数据线 $PA_7 \sim PA_0$ 后，使选通信号 $\overline{STBA}$ 有效，数据进入 A 口的输入缓冲器。

<2> A 口的 IBFA 有效，通知外设或 CPU，表示 A 口接收了一个有效数据。

<3> A 口的 INTRA 有效，以中断方式通知 CPU 取走 A 口中的数据。

<4> CPU 读 A 口，数据进入 CPU。

<5> IBFA 及 INTRA 转为“无效”。

上述过程可以用图 7-9 所示的时序图来表示。$\overline{STBA}$ 有效时低电平宽度不能小于 500 ns。

可见，当 A 口接收外设数据后，有两种方式通知 CPU 取数：① 用条件查询方式，通过查询缓冲器是否“满”，即 IBFA 是否为高电平来取数；② 用中断方式，要允许 8255A 中断，

即用控制字置 C 口的 INTEA 位为“1”。当外设的选通信号 $\overline{STBA}$ 把数据输入 A 口的输入缓冲器时，IBFA 有效并且通过一个“与”门产生中断申请信号 INTRA。CPU 若允许中断，则中断正在执行的程序，转到对 A 口读数的中断服务子程序。

当 CPU 把 A 口缓冲器的数据取走，缓冲器变“空”，使 IBFA 和 INTRA 变低，通知外设可送下一个数据到 8255A。

在条件传输中，一般要有所谓的“握手”信号来协调数据的传输。“握手”信号至少要有两位信号线。其中一位由接口电路发给外设，功能是向外设提供接口电路的信息。另一位由外设发给接口，功能是向接口提供外设的信息。显然，在 8255A 的选通输入方式中，$\overline{STBA}$ 和 IBFA 是一对“握手”信号。$\overline{STBA}$ 是外设所发，当其有效时，通知 A 口外设给它一个数据。IBFA 是 8255A 发出的，当其有效时，告诉外设 A 口已经接收这个数据。这个概念非常重要，因为比较复杂的外设，如打印机、A/D 转换器，要使其接口正确工作，一般都会提供握手信号。

【例 7-2】 8255A 的 A 口和 B 口分别工作在方式 1 和方式 0 下，A 口为输入端口，接有 8 个开关。B 口为输出端口，接有 8 个发光二极管，连接电路如图 7-10 所示。现要求用方式 1 把改变后的开关信息输入 CPU 并且通过 B 口显示。

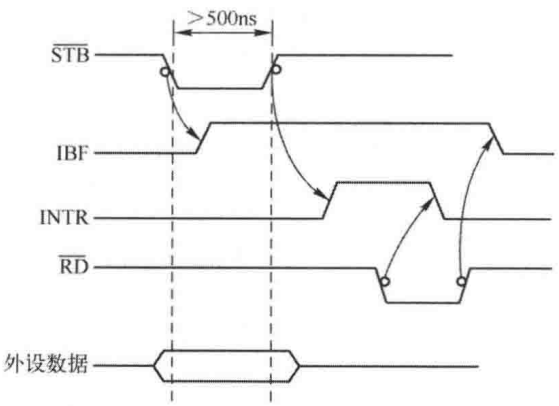


图 7-9　8255A 工作在方式 1 输入时的时序图

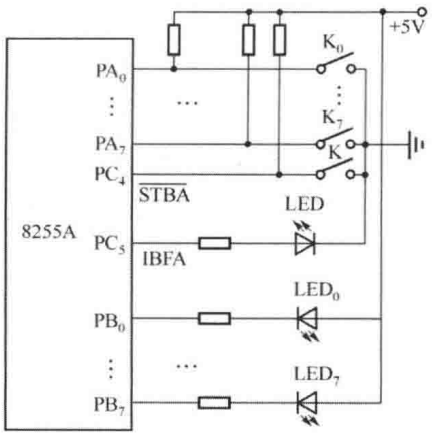


图 7-10　8255A 方式 1 输入举例

这是一个典型的条件输入数据过程的仿真。本例要求把改变后的开关值通过 B 口的发光二极管显示出来。这和上面所介绍的方式 0 的例子是有本质区别的。因为方式 0 的例子采用的是无条件传输方式，CPU 每次从 A 口输入的信息不一定是开关改变后的新值。本例要求只有当开关值改变后 CPU 才对 A 口读数，输入有效的信息。该系统成功地传输数据依赖两个条件：一个是 CPU 需要知道用户什么时候把数据送入 8255A 的 A 口，另一个是用户需要知道 CPU 何时把数据取走。

这个系统的工作过程如下：

<1> 用户通过改变 $K_0 \sim K_7$，产生新的开关值信息。

<2> 按下开关 K，产生选通信号 $\overline{STBA}$，数据进入 A 口的缓冲器。此步骤实际上告诉 CPU，8255A 的 A 口输入了一个新数据。

<3> IBFA 有效，使 LED 灯亮。这里含有两个信息：一是 8255A 通知 CPU 其 A 口有了新数据；二是告诉用户 CPU 尚未取走这个数据，用户不得再送其他数据。

<4> CPU 取走这个数据，LED 灯灭。

<5> 转步骤<1>。

设 8255A 的 I/O 地址分布为 88H～8EH，相应的程序段如下：

```
        MOV     AL, 10111001B          ; 设置 A 口为方式 1 的输入
        OUT     8EH, AL
LOOP1:  IN      AL, 8CH                ; 取 C 口的状态线
        TEST    AL, 00100000B          ; 测试 IBFA 信息
        JZ      LOOP1                  ; 等待用户设定新的键值
        MOV     CX, 0FFFFH             ; 延时，LED 灯亮（相对于步骤<3>）
LOOP2:  LOOP    LOOP2
        IN      AL, 88H                ; 取数，LED 灯灭（相对于步骤<4>）
        OUT     8AH, AL                ; 更新 B 口的显示
        JMP     LOOP1                  ; 重复
```

也许读者感到困惑，为什么在这个程序段中间，安排了延时程序？这里延时的目的是让用户能够观察到 LED 灯的变化。如果没有延时，当用户通过开关 K 发送选通信号 $\overline{\text{STBA}}$ 后，CPU 很快就把数据取走，LED 因显示时间太短而无法被用户观察到。

（2）选通输出方式

仍以 A 口为例（B 口的联络线可以参阅图 7-9）。与选通输入方式相似，A 口需有 C 口的 3 根线作为“联络”线，有一点区别就是取 PC_3、PC_6 和 PC_7 作为 A 口的“联络”线。PC_3 称为 INTRA，PC_6 称为 $\overline{\text{ACKA}}$，PC_7 称为 $\overline{\text{OBFA}}$。

INTRA：A 口的中断请求信号。当其有效时，8255A 的 A 口向 CPU 申请中断，要求 CPU 送数据给 A 口。

$\overline{\text{OBFA}}$：A 口的输出缓冲器“满”信号。当其有效时表示 A 口的输出缓冲器已经暂存一个有效数据。

$\overline{\text{ACKA}}$：外设应答信号。INTRA 和 $\overline{\text{OBFA}}$ 由 8255A 发出，而 $\overline{\text{ACKA}}$ 由外设发出。$\overline{\text{ACKA}}$ 为低电平有效。当其有效时，表示外设已经接收数据。

在方式 1 输出方式下（如图 7-11 所示），CPU 把数据通过 A 口送给外设的过程如下：

<1> CPU 执行 OUT 指令，把数据写入 A 口的输出缓冲器。

<2> 当有效数据进入 A 口的数据线 $PA_7 \sim PA_0$ 时，$\overline{\text{OBFA}}$ 有效，通知外设 CPU 已把一个有效数据输出给 A 口，外设可从 A 口取数据了。

<3> 外设取走数据时，发 $\overline{\text{ACKA}}$ 信号给 8255A，告诉 A 口外设正在取数据。

<4> A 口 $\overline{\text{OBFA}}$ 无效，表示 A 口数据已被外设取走。

<5> INTRA 有效，以中断方式通知 CPU 再输出数据给 A 口。

上述过程可用图 7-12 的时序图来表示。

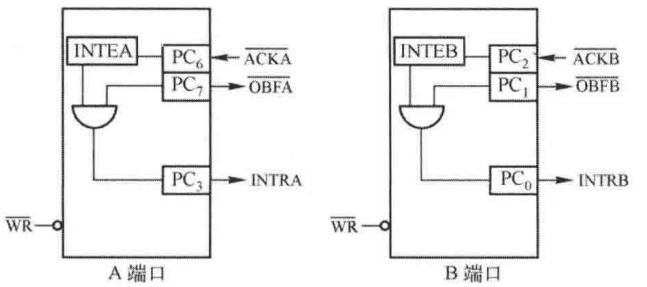


图 7-11　8255A 方式 1 输出

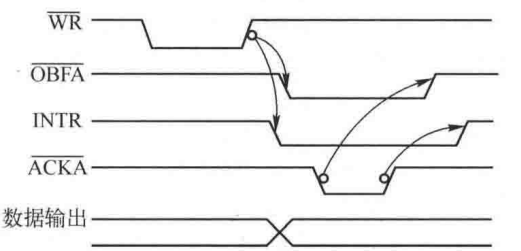


图 7-12　8255A 工作方式 1 输出时的时序图

同样，当 A 口的输出缓冲器数据被外设取走后，可有两种方式通知 CPU 再对 A 口写入数

据：① 用条件查询方式，通过查询输出缓冲器是否“空”，即$\overline{\text{OBFA}}$是否为高电平来决定CPU是否转向对A口输出数据的程序段；② 用中断方式，允许8255A中断，先用控制字置A口的INTEA位为“1”。若外设取走A口的数据，其应答信号$\overline{\text{ACKA}}$有效，使$\overline{\text{OBFA}}$为高电平并且通过一个“与”门产生中断申请信号INTRA。若CPU允许中断，则中断正在执行的程序，转到对A口写数据的中断服务子程序。

在这种方式下，$\overline{\text{ACKA}}$和$\overline{\text{OBFA}}$是一对“握手”信号。$\overline{\text{OBFA}}$是8255A产生的，当其有效时，告诉外设A口已有一个新数据。$\overline{\text{ACKA}}$是外设产生的，当其有效时，通知A口外设已把数据取走。

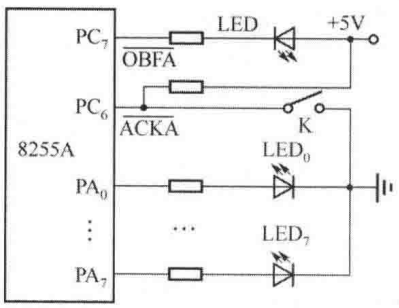


图 7-13　8255A 方式 1 输出连接电路

【例 7-3】 8255A的A口工作在方式1的输出，接有8个发光二极管，连接电路如图7-13所示。现要求把内存中的10个数据通过A口发送给发光二极管（以二进制的形式供用户抄录），设8255A的I/O地址分布为88H～8EH。

这是一个典型的条件输出数据过程的仿真。此系统要实现成功的数据传输应该有两个必备条件：一是用户需要知道LED显示的数据是否是新数据；二是CPU要知道用户是否已抄录数据。

这个系统的工作过程如下：

<1> CPU把内存中的一个数据写入A口。

<2> LED灯亮，告诉用户LED显示的是新数据。

<3> 用户抄录数据。

<4> 用户按下开关K，发ACK信号，告诉CPU数据已取走。

<5> 转第<1>步。

相应的程序段如下：

```
DATA    SEGMENT                          ; 定义数据段
XX      DB      X0, X1, …, X9            ; 定义数组，10个元素
DATA    ENDS
CODE    SEGMENT                          ; 定义代码段
        ASSUME  DS:CS, CS:CODE
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     CX, 10                   ; 送10个数
        MOV     BX, OFFSET XX            ; 数组指针送BX
        MOV     AL, 10101001B            ; 设置A口为方式1的输出
        OUT     8EH, AL
LOOP1:  MOV     AL, [BX]                 ; 取数
        OUT     88H, AL                  ; 送数到A口
LOOP2:  IN      AL, 8CH                  ; 取C口状态线
        AND     AL, 80H                  ; 测试OBFA
        JNZ     LOOP2                    ; 用户尚未抄录数据等待，此时LED灯亮（对应步骤<2>）
        CALL    DELAY                    ; 用户已抄录数据，LED灯灭（对应步骤<4>）
        INC     BX                       ; 准备送下一个数
        LOOP    LOOP1                    ; 循环10次
```

```
        MOV     AX, 4C00H
        INT     21H                          ; 返回系统
CODE    ENDS
        END  START
```

思考题： 为什么在这个程序段中间要安排延时程序？

3．8255A 工作在方式 2 下

8255A 只允许 A 口处于工作方式 2，用来在两台处理机之间实现双向并行通信。由于方式 2 下的 A 口既能发送，也能接收数据，所以 A 口的引脚在“空闲”状态下是三态的。当然，A 口在某一时刻下，输入或输出是由相应“联络”线确定的。工作时可用软件查询方式，也可用中断方式。当 A 口工作于方式 2 时，C 口的“联络”线既要提供 A 口的输入“联络”线 $\overline{\text{STBA}}$ 和 IBFA，又要提供 A 口的输出“联络”线 $\overline{\text{ACKA}}$ 和 $\overline{\text{OBFA}}$，还要处理 A 口的中断申请线 INTRA，因此占用了 C 口的 5 根线。这时 B 口就只能工作在方式 1 和方式 0 下。显然，如果设定某个 8255A 的 A 口和 B 口分别工作在方式 2 和方式 1 下，则 8255A 的 C 口的 I/O 功能将不再存在。如果 B 口是工作在方式 0 下，则 C 口还有 3 根线可作为 I/O 线。

8255A 工作在方式 2 时，“联络”信号的定义如图 7-14 所示。图 7-15 为时序图。

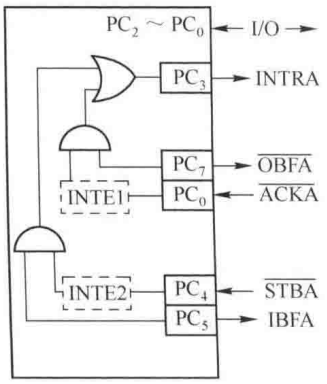


图 7-14　方式 2 下的 A 口联络线

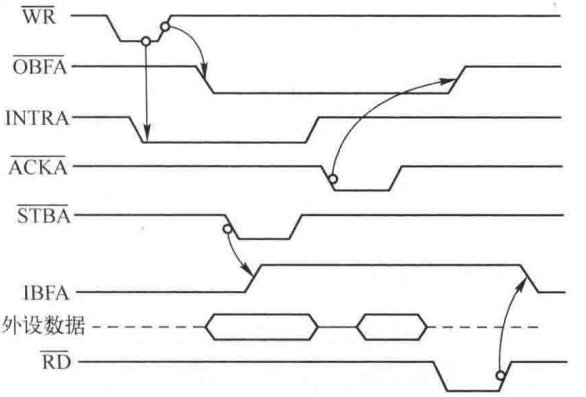


图 7-15　8255A 的 A 口工作在方式 2 的时序图

可以看出，8255A 的 A 口工作在方式 2 时，其“联络”信号的功能和前面介绍的一样，其中 PC_4 和 PC_5 提供 A 口在输入时的“联络”，PC_6 和 PC_7 提供 A 口在输出时的“联络”，PC_3 作为 A 口的中断请求线。

当 8255A 的 A 口工作在方式 2 时，输入和输出共用同一个中断请求信号线。所以，当使用中断方式传输数据时，可以通过 C 口置“1”/清“0”控制字来设置中断屏蔽触发器 INTE1 和 INTE2，实现对中断源的控制，禁止/允许输入时中断或输出时中断。如果同时允许输入中断和输出中断，则在中断程序中首先从 C 口读取状态字；通过对状态线 IBFA 和 $\overline{\text{OBFA}}$ 的查询检测，进一步确定是输入中断还是输出中断。

7.1.5　8255A 应用举例

并行接口应用非常广泛，如 CPU 与打印机接口、键盘接口、A/D 与 D/A 接口等。8255A 作为接口电路的主要器件，常出现在这些电路中。所以，8255A 是计算机应用中最热门器件之一。下面给出几个实用的并行接口的例子，让读者对并行接口的设计有更深入的了解。

1．与打印机接口

【例 7-4】 某 8086 系统中接有一台打印机，8255A 作为输出接口，工作在方式 0 下。编写一程序，将缓冲区 BUFF 内的 400H 字节的 ASCII 值送打印机打印。

打印机并行接口中，最常用的接口信号线有 10 根，包括 8 位数据线 $D_7 \sim D_0$、选通信号线 $\overline{STB}$ 和打印机“忙”线 BUSY。正确理解和使用 $\overline{STB}$ 线和 BUSY 线是实现打印机与 CPU 接口的关键。

从接口信号的功能来看，选通信号 $\overline{STB}$ 是由接口电路产生的，所以对打印机来说，它是控制信号。在标准打印机接口电路中，选通信号被设置为低电平有效。当其有效时，数据线 $D_7 \sim D_0$ 上要打印的数据就进入打印机缓冲区内，打印机就可以打印这个数据了。由于打印机的速度大大低于 CPU 的执行速度，因此打印机在打印某数据时，BUSY 有效（高电平），表示打印机正在打印。故 BUSY 线是打印机的状态线，由打印机产生。如果打印机的缓冲区不大，在此信号有效时，CPU 将不再送数据给打印机。

打印机具体工作过程如下：

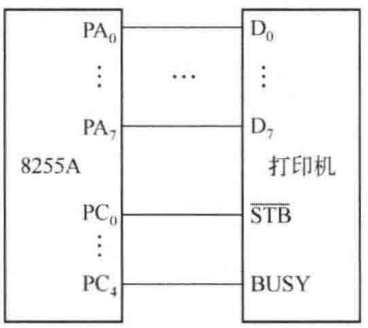


图 7-16　8255A 作为打印机接口电路

<1> 数据线 $D_7 \sim D_0$ 出现有效数据。

<2> $\overline{STB}$ 有效，通知打印机，接口给打印机一个数据，数据从数据线进入打印机。

<3> BUSY 有效，告知接口，打印机正在打印数据。打印完毕，BUSY 变为无效，表示打印结束。

<4> 转步骤<1>。

尽管打印机的工作过程决定了打印机接口应该工作在条件传输方式下，但 8255A 仍可选用方式 0 来方便地实现打印机接口。图 7-16 给出了其接口电路。

在接口电路中，8255A 的 A 口充当打印机数据线，C 口的 PC0 接 $\overline{STB}$ 线，C 口的 PC4 接 BUSY 线。这样 A 口应该设置为输出，PC0 为输出，而 PC4 为输入。故方式字为 10001010B，这里 B 口没用，设为输入方式。

相关打印程序比较简单，过程如下：

<1> 首先检测 BUSY 是否有效，是，则循环检测；否则执行第<2>步。

<2> CPU 写有效数据到 A 口，发 $\overline{STB}$ 信号，把 A 口数据输出给打印机。

相关程序段如下：

```
A_PORT    EQU     80H                          ; 定义 8255A 的 4 个端口
B_PORT    EQU     82H
C_PORT    EQU     84H
CTRL_PORT EQU     86H
          ......
          MOV     AL, 10001010B                ; 8255A 初始化
          OUT     CTRL_PORT, AL
          MOV     AL, 00000001B                ; PC0 置位
          OUT     CTRL_PORT, AL
          MOV     CX, 400                      ; 循环参数设置
          MOV     BX, OFFSET BUFF
LOOP1:    MOV     DL, [BX]                     ; 取数
```

```
        CALL    PRINT_DATE                  ; 打印
        INC     BX
        LOOP    LOOP1                       ; 打印结束，进行其他处理
        …
PRINT_DATA  PROC                            ; 打印子程序，入口在 DL 中
PRINT1:  IN     AL, C_PORT                  ; 无条件读 C 口数据
        TEST    AL, 00010000B               ; 测试 BUSY 线
        JNZ     PRINT1                      ; BUSY 有效，循环测试
        MOV     AL, DL
        OUT     A_PORT, AL                  ; 打印数据进入 A 口
        MOV     AL, 00000000B               ; 发 STB 信号
        OUT     CTRL_PORT, AL
        MOV     AL, 00000001B
        OUT     CTRL_PORT, AL
        RET                                 ; 返回主程序
PRINT_DATA  ENDP
```

可以看出，上述程序通过对 C 口的无条件读取方式，也就是测试打印机的状态线 BUSY，实现把 A 口的数据有条件传输给打印机。

2．人机交互接口

人机交互是指人与计算机之间使用某种对话手段，以一定方式，为完成确定任务而进行的人机之间信息交换的过程。作为人机系统的计算机，由于技术等各方面因素的影响，它的理解能力、表达能力等方面都仍是有限的，在有了多媒体技术之后虽然仍有种种限制，但已逐渐丰富和生动。

人机交互系统是指实际完成人机交互的系统，由参与交互的各方（包括人和计算机）组成的交互系统。广义而言，交互系统的组成应包括参与交互的实体和实体间的交互作用及其环境。例如，人使用计算机来分析教育资料的信息，就包括人、计算机及数据采集系统。又如，在包括计算机的人机系统中，其组成应包括人、硬件、软件和作为环境的有关文档手册。

人机交互方式是指人机之间交互信息的组织形式或语言方式，又称为对话方式、交互技术等。系统通过不同的人机交互方式实际完成人向计算机输入信息和计算机输出信息的工作。目前，常用的人机交互方式有问答式对话、菜单技术、命令语言、填表技术、查询语言、自然语言、图形方式及直接操纵等。

交互介质是指用户和计算机完成人机交互的媒体，一般可分为输入介质和输出介质。其中，输入介质是指实现人向计算机传递信息的媒体，常用的输入介质有键盘、鼠标、光笔、跟踪球、操纵杆、图形输入板、声音输入设备等。

强化和完善人机交互系统功能是导致计算机结构复杂的主要原因之一，其内容和概念远远超过了本书的范围。在此仅讨论两种最简单也是数字化仪器仪表最常用的交互介质：键盘和 LED 显示。

（1）显示接口

许多微型或小型数字化仪器仪表为了使用方便、携带容易，不采用 PC 那样的 CRT 显示方式，而是采用七段数码的发光二极管或者液晶数码显示方法。发光二极管数码显示具有不需定制、在暗背景下显示清楚、功耗大等特点，而一般的液晶数码显示具有需要定制（如 3 位

半、4 位等)、不易在暗背景下观察显示、功耗小等特点。由于多位的发光二极管数码显示系统多采用典型的扫描显示技术，因此了解和掌握它与计算机接口的方法，非常有益于了解其他数码显示技术。

1 位数码显示已在前面介绍，但对于多位，如 8 位数码显示，如果直接采用这种接口技术就会出现一些问题，如图 7-17 所示。

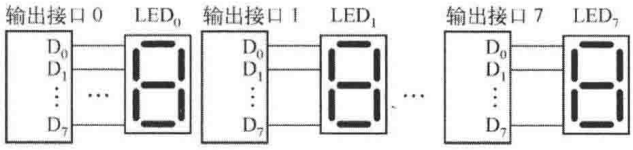


图 7-17　静态显示

由于一个 1 位的输出接口只能实现 1 位数据的显示，因此 8 位的数码显示需要 8 个输出接口。这就需要 3 个 8255A 芯片并占用 8 个地址，对于一个微型仪器仪表来说，这不是一个易接受的方案，因为增加了系统的复杂性。由于 8 个数码同时显示，需要更多的电流，增加了系统的功耗。我们常把这种显示称为静态显示。

还有一种更常用的显示称为动态显示，采用扫描显示技术，可使硬件开销降低很多。一个 8 位数据显示需要两个输出端口即可，其电路如图 7-18 所示。

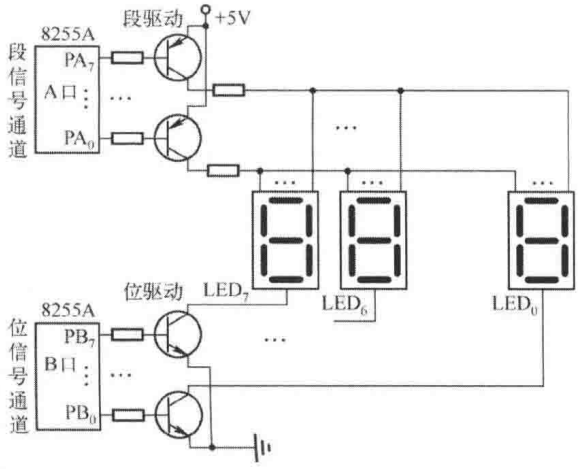


图 7-18　动态显示电路

在两个输出端口中，一个称为段信号通道，用来输出要显示数据的段码；另一个称为位信号通道，用来决定当前显示数据的位置。CPU 先输出段信号经 8255A 的 A 口锁存，再通过晶体管放大增加驱动能力后，送到 LED 以显示数据。至于是哪一位显示则由位信号控制，如果 CPU 输出给 B 口的位信号是 01H，则 PB0 有效，则由最右边的那个 LED0 数码显示器显示数据，并且通过计算机延时保持适当的显示时间。采用这种方法，就可使 LED1、LED2、…、LED7 依次点亮来显示数据。再重复之。显然，每个数码是闪烁显示。只要数码闪烁频率大于 25Hz，则由于观察者的视觉暂时记忆而成为连续显示。

这种技术以增加显示软件的复杂性来达到简化硬件的目的。显示系统的具体步骤如下。

<1> 设要显示的初始位码 i=0。

<2> 送第 i 位的段信号。

<3> 送第 i 位的位信号。

<4> 延时。

<5> i+1→i。

<6> 如果 i 小于 8，转步骤<2>。

<7> 8 位数据显示结束。

相关程序段如下：

```
A_PORT    EQU     80H                                    ; 定义 8255A 的 4 个端口
B_PORT    EQU     82H
C_PORT    EQU     84H
CTRL_PORT EQU     86H
          ……
DATA      SEGMENT                                        ; 定义数据段
          TAB1          DB  3FH, 06H, 5BH, 4FH,…         ; 定义段码表
          DISPBUFF      DB  2, 0, 0, 4, 0, 8, 2, 2       ; 定义显示缓冲区
DATA      ENDS
CODE      SEGMENT                                        ; 定义代码段
          ……
          ; 下面 8255 初始化，A 口、B 口为输出，C 口上部输出，下部输入
          MOV     AL, 10000001B
          OUT     CTRL_PORT, AL
          ……
LOOP2:    ……                                             ; 其他处理
          CALL    DISPLAY                                ; 显示
          JMP     LOOP2                                  ; 循环
          ……
DISPLAY   PROC                                           ; 显示子程序，数据在 DISPBUFF 中
          ……                                             ; 保护现场
          MOV     BX, OFFSET TAB1
          MOV     SI, OFFSET DISPBUFF+7                  ; 步骤<1>
          MOV     CX, 8
          MOV     AH, 01
DISP1:    MOV     AL, [SI]                               ; 步骤<2>
          XLAT
          OUT     A_PORT, AL
          MOV     AL, AH                                 ; 步骤<3>
          OUT     B_PORT, AL
          CALL    DELAY5MS                               ; 步骤<4>，延时 5ms
          SHL     AH, 1                                  ; 步骤<5>
          DEC     SI
          LOOP    DISP1
          ……                                             ; 恢复现场
          RET
DISPLAY   ENDP
```

根据程序中对显示缓冲区的设置可以知道，显示器应显示数据为 20040822。当然，如果在注释“其他处理”的程序段中对缓冲区内容修改的话，则显示新的内容。

（2）键盘接口

在微机化仪器仪表中，键盘是最常用的一种输入设备。键盘有两种类型：全编码键盘和非编码键盘。

全编码键盘多是商品化的计算机输入设备，自动提供对应于被按键的 ASCII 值，且能同时产生一个控制信号通知微处理器。此外，这种键盘具有处理抖动和多键串键的保护电路，具有使用方便、价格较贵、体积较大、按键较多等特点。非编码键盘恰如一组开关，一般组成行和列矩阵。其全部工作过程，如按键的识别、键的代码获取、防止串键及消抖动等问题，都靠程序完成。因此，它所需的硬件少，价格便宜，一般作为单板机、智能仪表等简单的输入设备。

键盘处理程序任务包括如下 3 步。

<1> 键输入是键盘处理程序最主要的工作。考虑到键盘的机械结构，当键按下和松开时，键将有一个抖动过程（如图 7-19 所示）。因此键输入的首要工作是检查键盘是否有键被按下，消除按键抖动。然后确定被按的键在矩阵中的位置，获取键号。软件上采用延时采样的方法就可以有效地消除抖动。

<2> 键译码。键号为键盘位置码，根据键号查表得出被按键的键值，如数字键 0～9、字符键 0AH～0FH、功能键 10H～15H。

<3> 键处理根据键值转移到不同程序段。若键值属于数字、字符键，则调用显示数字和字符的子程序。若键值属于功能键，则进行多分支转移，执行各功能程序段。

在键盘处理程序中，只有键输入部分是与接口有关的，其他部分实际上是一个数据转换和相关处理的过程。所以，这里主要讨论键输入的实现。

目前键盘电路常用的有两种：一种是独立式键盘电路，另一种是矩阵式键盘。独立式键盘电路如图 7-20 所示，其特点是每个按键独占一根 I/O 线。因此键识别软件非常简单。对于只有几个按键的系统，常采用这种电路，如 8255A 方式 0 中的例子。对于多按键系统来讲，这种电路由于将占用更多的 I/O 线而变得无法实用。

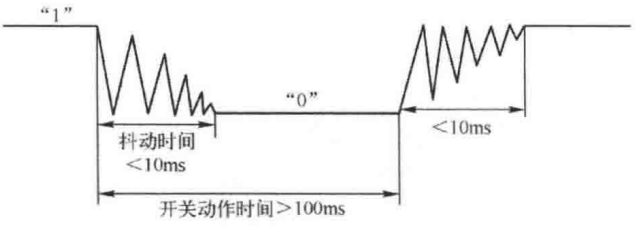


图 7-19　键抖动

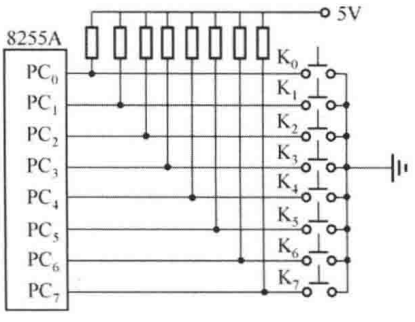


图 7-20　独立式键盘电路

图 7-21 为矩阵式键盘电路。其特点是由按键组成一个矩阵，矩阵的行线和列线分别作为两个传输方向相反的 I/O 接口信号线，如行线作为输入接口信号线，列线作为输出接口信号线，或反之。同独立式键盘相比，这种电路所需的 I/O 线很少，但键盘容量很大，对于有 N 个 I/O 线作为键盘接口线的系统，可连接按键的个数最多可达 $N^2/4$。在图 7-21 中，用 C 口的低 4 位作为键盘矩阵的行线，高 4 位作为列线，则可连接 16 个按键。

虽然矩阵式键盘电路对 CPU 硬件资源要求不多，但相应的软件却复杂得多。目前有两种方法来获取键的位置或键号：扫描法和翻转法。

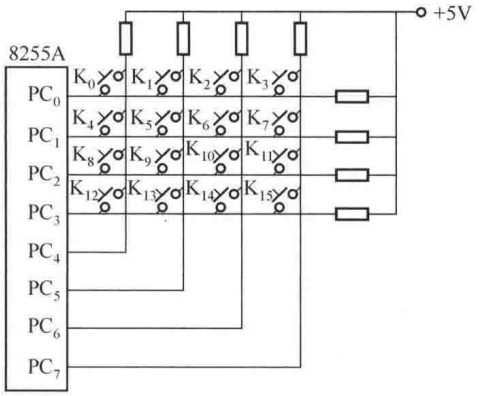


图 7-21　矩阵式键盘电路

在扫描法中，设定行线输出，列线输入（当然，也可反之）。行线逐行输出 0，如果某列有按键，则列线输入为 0；若无按键，列线输入全部为 1。这样，根据行线和列线提供的信息就可以确定是哪个键被按下。以第 5 号键按下为例，在逐行扫描下，第 0 行首先为 0，由于 0 行上无键按下，故列线输入全为 1。然后扫描第 1 行，此时第 1 列输入为 0，其他为 1。由于列线 1 出现信号 0，认定第 1 列中有键按下。由行线 1 输出为 0 和列线 1 输入为 0，就可知道按下的键是在第 1 行与第 1 列的交叉点。

在翻转法中，行、列线交换输入、输出，两步就可获取键位置信息。可见，这种方法要比扫描法效率高。仍以第 5 号键按下为例，首先行线为输出，列线为输入。当行线输出全部为 0 时，列线 1 输入为 0，其他列线输入为 1。反之，列线为输出，行线为输入。当列线输出全部为 0 时，行线 1 输入为 0，其他行线输入为 1。由行线 1 输入为 0 和列线 1 输入为 0，也能认定已按下的键是在第 1 行与第 1 列的交叉点。

翻转法程序流程如下：

<1> 设定行为输出，列为输入。

<2> 行输出为 0，输入列信号 j。

<3> 检查列信号是否全为 1，若是，无键按下，转步骤<2>；如果不全为 1，表明有键按下，执行步骤<4>。

<4> 延时 10 ms，消除抖动。

<5> 逐列检测，找出为 0 的列信号 j。

<6> 设定列为输出，行为输入。

<7> 列输出为 0，输入行信号 i。

<8> 检查行信号是否全为 1，若是，表明按键有错，转错误处理；如果不全为 1，表明有键按下，执行步骤<9>。

<9> 逐行检测，找出为 0 的行信号 i。

<10> 用公式 key_num=4*i*+*j* 计算键号 key_num。

基于翻转法的键盘输入信息的程序段如下：

```
A_PORT    EQU     80H                    ; 定义 8255A 的 4 个端口地址
B_PORT    EQU     82H
C_PORT    EQU     84H
CTRL_PORT EQU     86H
```

```
        ......
        MOV     AL, 10001000B           ; 步骤<1>
        OUT     CTRL_PORT, AL
        MOV     AL, 0                   ; 步骤<2>
        OUT     C_PORT, AL
NO_KEY: IN      AL, C_PORT
        AND     AL, 0F0H
        CMP     AL, 0F0H                ; 步骤<3>
        JZ      NO_KEY
        CALL    DELAY10MS               ; 步骤<4>
        IN      AL, C_PORT
        SHR     AL, 1                   ; 列信号信息移到低4位
        SHR     AL, 1
        SHR     AL, 1
        SHR     AL, 1
        MOV     DL, 0                   ; 置初始列变量DL为0
        MOV     CX, 4
LOOP1:  SHR     AL, 1                   ; 该循环为步骤<5>
        JNC     LOOP2
        INC     DL
        LOOP    LOOP1
LOOP2:  MOV     AL, 10000001B           ; 步骤<6>
        OUT     CTRL_PORT, AL
        MOV     AL, 0                   ; 步骤<7>
        OUT     C_PORT, AL
        IN      AL, C_PORT
        AND     AL, 0FH
        CMP     AL, 0FH                 ; 步骤<8>
        JZ      ERROR                   ; 键盘有错，转错误处理程序
        MOV     DH, 0                   ; 置初始行变量DH为0
        MOV     CX, 4
LOOP3:  SHR     AL, 1                   ; 该循环为步骤<9>
        JNC     LOOP4
        INC     DH
        LOOP    LOOP3
LOOP4:  SHL     DH, 1                   ; 步骤<10>
        SHL     DH, 1
        ADD     DH, DL                  ; DH为键号
        ......                          ; 其他处理
```

7.2 可编程定时/计数器接口芯片 8253

并行接口主要用于并行数据的传输，如打印机、A/D、D/A 等设备，传输的信息是二进制代码或开关量。在实际应用中，还存在其他类型的信息及其相应的处理方式。如在微机系统中，常常要求有一些实时时钟以实现定时或延时控制，像定时中断、定时检测、定时扫描等。如有

时要求对脉冲信号进行处理，即用计数器对外部事件进行计数，如定额包装、事件发生频率等。这里先举两个例子。

1．软件定时

用计算机实现定时或延时有两种基本办法：利用软件定时或使用可编程硬件芯片。

软件定时，即让机器执行一段程序，这个程序本身没有具体的执行目的，但由于执行每条指令 CPU 都要花费时间，因而执行一个程序段就有一个固定的时间。用户可以选用指令并且安排循环来实现定时，这种方法容易实现，定时时间调整方便，但不能做到精确定时。时间调整以一条指令执行时间为基准，且占用 CPU 资源，降低了 CPU 的利用率。

使用可编程定时/计数硬件芯片可以很好地完成定时任务。这种芯片内部有一个可编程定时器，其定时值、定时范围可以很容易地由软件改变，定时时间到时可发出某种形式的信号来通知外设或 CPU，定时器的输出频率和波形等均由程序设定。它具有使用灵活、功能较强等特点。

2．外部事件计数

外部事件计数是对外部脉冲信号计数。产生脉冲信号的外部原因就是外部事件。这方面的例子俯拾皆是。

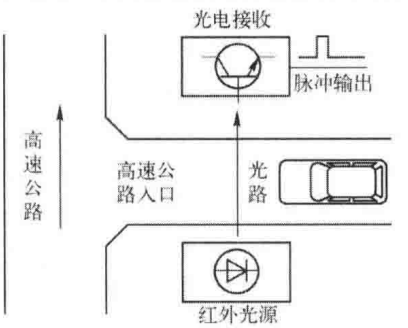


图 7-22　车辆计数系统

图 7-22 是高速公路入口处一个专用计算机检测系统，它可以自动对进入高速公路的车辆进行计数。这个系统的信息采集系统很简单，由一对红外发射和接收装置组成。当机动车辆通过采集路段时，车辆会阻断红外光路，当车辆通过后，光路恢复。红外接收电路的功能把光信号变化转换为电信号的变化即脉冲信号。这样每过一辆车，接收电路就会发出一个脉冲，它表示了一个外部事件，即有一辆车进入高速公路。显然，系统通过对脉冲的计数和累加，就会知道在规定的时间内，如一天，有多少车辆从这个入口处进入高速公路。

同样，用计算机实现对外部事件计数也有两种基本办法：一是用软件进行计数，二是用可编程计数器。

用软件进行计数，就是外部脉冲通过并口（如 8255A 的 C 口的某根 I/O 线）送入计算机，软件不停地检测这根线的状态。当发现其逻辑电平发生变化时，就认定有一次外部事件到来。这时，软件将某一个变量加 1。这种方法要求 CPU 始终查询输入线的状态，否则就有可能漏掉一次外部事件。这种方案占用了 CPU 大量的资源。

可编程计数器核心电路就是一个事先可以设置计数常数的计数器，其记录脉冲方式和计满“溢出”方式都可通过编程设定。外部脉冲可以通过脉冲输入线进入计数器进行计数，CPU 可在任何时刻通过并口访问这个计数器，读取已记录的数据。所以这种方式非常灵活，完全可以代替软件计数，并使 CPU 对计数事件处理的开销费用降到最低。

综上所述，有关定时和脉冲信号的处理与接口是完全有别于并行信号的。其特点是信号形式简单但需要连续检测，用 8255A 作为接口和通过 CPU 进行检测、计数处理的方法也许是可行的，但不一定可取，因为它占用太多的 CPU 时间。那么选用什么样的芯片来完成上面的例子呢？下面介绍的 Intel 8253 可编程定时/计数器就是一个最佳的选择，它可以轻松地实现所要求的功能。8253 内部有 3 个独立的 16 位定时/计数器通道。计数器按照二进制或十进制计数，计数和定时范围为 1～65535，每个通道有 6 种工作方式，计数频率高达 2 MHz。

7.2.1 8253 的内部结构

8253 内部结构框图如图 7-23 所示。在 8253 内部结构中，厂家已把计数器电路集成在一些称为通道的部件中。8253 有 3 个这样的通道，还包括控制这些接口电路的控制部分以及与 CPU 接口的总线接口部分。

1．计数通道

8253 芯片中包含 3 个功能完全相同的计数通道，称为通道 0、通道 1 和通道 2。这 3 个通道与外部电路相连的信号线有 3 根：CLK、GATE 和 OUT。CLK 是计数器的脉冲输入端，GATE 是计数器的门控信号，OUT 是计数器的输出信号，一般与计数溢出有关。这 3 个信号和计数器的逻辑关系可用图 7-24 表示。

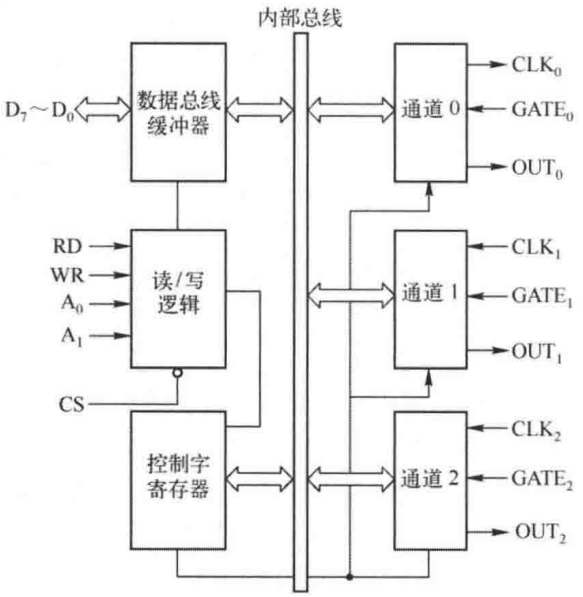


图 7-23　8253 内部结构框图

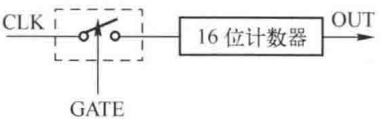


图 7-24　各信号与计数器的逻辑关系

首先，外电路的脉冲信号通过 CLK 信号线进入通道，如果 GATE 有效，使 CLK 上的脉冲信号进入计数器计数，8253 计数器工作在减 1 状态，每输入一个计数脉冲，计数器值减 1。当计数器计数到零时，OUT 信号有效通知外设计数器产生溢出。在通道工作过程中，CPU 可以随时通过对端口的读/写，读取计数器的数据。

在 8253 中，每个通道内部设置一个 16 位计数器，可进行二进制或 BCD 码计数。采用二进制时，最大计数值为 0FFFFH；采用 BCD 码计数时，最大计数值为 9999。与此计数器相对应，每个通道内部设有一个 16 位计数值锁存器，必要时用来锁存计数值。

当某通道用作计数器时，应将要计数的次数预置到该通道计数器中；当用作定时器时，从 CLK 输入一固定频率的时钟脉冲，再根据要求定时的时间算出定时所需的计数值（或称时间常数），并预置到计数器中。计数值与定时时间、CLK 端上时钟脉冲信号周期的关系如下：

计数值=定时时间/时钟脉冲周期

各通道的输入与输出之间的关系与门控电路的控制有关，门控电路的作用随各通道选用的工作方式不同而不同。

2．通道控制寄存器

虽然 8253 有 3 个计数通道，但只有 1 个通道控制寄存器。CPU 通过对控制寄存器的读/写，可以分别对 3 个计数通道的工作方式进行设置。通道控制寄存器只能写不能读。

3．数据总线缓冲器

数据总线缓冲器是双向三态的 8 位数据缓冲器，实现 8253 与 CPU 之间的数据接口。CPU 执行输出指令时，可将控制字或数据通过该缓冲器传送给 8253 的控制寄存器或计数通道端口。CPU 执行输入指令时，8253 可将通道计数器当前的计数内容传送给 CPU。因此，数据总线缓冲器是 CPU 与 8253 交换信息的必经之路。

4．读/写控制电路

这是 8253 内部操作的控制部分。首先是片选信号 $\overline{CS}$，低电平有效。当其有效时，表示本 8253 芯片被 CPU 选中，可以工作；当其为高电平时，数据总线缓冲器处于三态，与系统数据总线脱开，故不能进行编程，也不能进行读/写操作。8253 内部有 3 个独立通道和 1 个控制字寄存器，构成芯片的 4 个端口。CPU 可对 3 个通道进行读/写操作，但对控制字寄存器仅进行写操作，这 4 个端口地址由 A_1 和 A_0 的组合状态提供。

读信号 $\overline{RD}$ 和写信号 $\overline{WR}$ 由 CPU 提供，低电平有效。当 $\overline{RD}$ 有效时，CPU 可以根据要求读取 3 个通道中的指定通道的计数器值；当 $\overline{WR}$ 信号有效时，CPU 将计数值或命令字写入计数器或控制字寄存器中。

7.2.2 8253 的引脚分配

8253 是一个标准的 24 引脚芯片，分为 3 部分（如图 7-25 所示）：与外设相连的通道引脚，与 CPU 相连的系统总线，电源线。

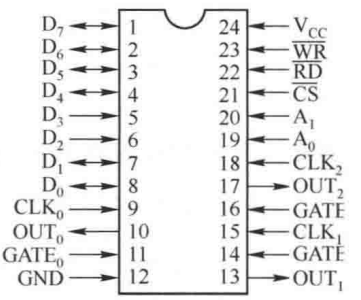


图 7-25　8253 引脚

1．通道引脚

8253 有 3 个通道，每个通道有 3 条信号线，因此与外设相联系的引脚有 3 组 9 根。

CLK_n：通道 n 的脉冲输入引脚，外部事件或定时脉冲由这些引脚输入。

OUT_n：通道 n 的输出引脚，当计数值减到 0 时，在 OUT 引脚上输出，输出波形取决于 8253 通道的工作方式。

$GATE_n$：门控信号输入引脚，这是控制计数器工作的一个外部信号。当 GATE 引脚上输入低电平时，通常是禁止计数器工作的。n 表示通道序号，可以取 0，1，2。

表 7-10　8253 端口选择

A_1	A_0	端　口
0	0	通道 0
0	1	通道 1
1	0	通道 2
1	1	控制口

2．与 CPU 相关的引脚

与 8255A 相似，8253 的引脚包括数据线 $D_7 \sim D_0$，读/写控制线 $\overline{WR}$、$\overline{RD}$，以及与 CPU 地址线相连接的片选信号 $\overline{CS}$、端口地址控制引脚 A_1 和 A_0。这两个端口地址线的组合状态可选择 8253 内的 3 个通道端口和 1 个控制口，如表 7-10 所示。

3．电源线

8253 的电源引脚为 VCC 和 GND。GND 为地线，VCC 为电源线，一般取+5 V。

7.2.3 8253 的编程

8253 的初始化编程就是对其工作方式的确定。具体实现就是在 8253 上电后，由 CPU 向 8253 的控制寄存器写入一个控制字，就可以规定 8253 的工作方式、计数值的长度以及计数所用的数制等，另外根据要求将计数值写入 8253 相应的通道。

8255A 有两个控制字，即工作方式控制字和置位/复位控制字；8253 只有一个控制字，故不需要特征位。与 8255A 的方式控制字的另一个区别是，8255A 一个方式控制字就同时决定了 A 口、B 口和 C 口的工作方式，8253 的一个方式控制字只决定一个计数通道的工作模式。

8253 的控制字格式如图 7-26 所示，共分为 4 部分：通道选择，计数器读/写方式，工作方式和计数码的选择。

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	RL_1	RL_0	M_2	M_1	M_0	BCD

图 7-26 8253 控制字格式

1．计数器选择（SC_1，SC_0）

计数器选择用于指定本控制字所对应的通道。控制字中的 D7 和 D6 位决定这个控制字写入哪个通道，如表 7-11 所示。

由于 8253 三个通道的工作是完全独立的，每一个通道都有自己的方式控制字，而 8253 只有一个控制端口，所以控制字需要标明是针对哪一个通道的，这就是计数器选择的意义所在。因此，当三个通道的编程需要向同一个地址（控制寄存器地址）写入三个控制字时，就由 SCl 和 SC0 分别指定不同的通道。注意：每个通道均有自己的计数器地址，写入的计数值（或时间常数）应分别写入各通道的相应地址。

2．计数器读/写方式（RL1，RL0）

8253 在初始化时，CPU 向计数通道写入计数初值，然后计数器就在这个初值基础上进行减 1 计数操作。在计数器工作时，允许 CPU 随时读取通道的当前计数值。为了灵活地对计数通道进行读写操作，8253 提供给用户 3 个不同格式的读/写方法，如对计数通道可以写入 8 位数据也可写入 16 位数据，见表 7-12。从表 7-12 可以看出，对 8 位数据写入的操作可分为两种格式：仅写低 8 位和仅写高 8 位，前者高 8 位自动置 0，而后者低 8 位自动为 0；对 16 读/写位操作时，应先读/写低 8 位，后读写高 8 位。在对 8253 编程时，一般采用 16 位的读/写操作。

表 7-11 8253 通道选择

SC_1 SC_0	对应的通道
0 0	通道 0
0 1	通道 1
1 0	通道 1
1 1	不用

表 7-12 8253 读/写方式

RL_1 RL_0	通道读/写操作
0 0	计数器锁存
0 1	只读/写低 8 位字节
1 0	只读/写高 8 位字节
1 1	读/写 16 位

现在重点讨论计数器的锁存操作。

（1）16 位计数器读过程中可能出现的问题

在一些计数或定时功能应用中，要求读出 8253 通道计数器中的计数值。8253 一旦初始

化，就不需要 CPU 参与而自动计数。为了读出计数值时不干扰实际计数过程，同时读出的值又是稳定的，就要求对通道计数器中的计数值进行锁存。

例如，把 8253 通道 0 当前计数值取出并送到 CPU 的 CX 寄存器中。因为 8253 计数通道是 16 位的，但 8253 与 CPU 的接口是 8 位的，所以要对计数通道进行两次读取数据的操作。程序段如下：

```
IN      AL, 通道0              ; 读通道计数器低8位
MOV     CL, AL
IN      AL, 通道0              ; 读通道计数器高8位
MOV     CH, AL
```

尽管这种方案很简单，但存在一个致命的问题。8253 在编程后通道 0 的计数器一直是工作的，每来一个脉冲，计数器内容就会自动减 1。所以上述程序段在执行时，通道 0 计数器有可能正好处于低 8 位向高 8 位的借位计数状态，这时对通道 0 读出的 16 位的计数值就有可能是错误的。比如在第一次对通道 0 读数，也就是读取计数器低 8 位内容时，通道 0 的计数值是 1000H，因此 CL 的内容是 00H，但在 CPU 执行指令

```
MOV     CL, AL
```

时，外电路通过 CLK 端又给通道 0 一个脉冲信号，则通道 0 对这个脉冲计数后，计数内容为 0FFFH。这样在第二次对通道 0 读数，也就是读取计数器高 8 位内容时，CH 的内容将是 0FH。最后结果是寄存器 CX 中的数据为 0F00H，这显然是错误的。正确的读出数据应该是 1000H 或 0FFFH。

（2）解决方法

① 硬件锁存暂停计数

利用 GATE 门控信号的作用，禁止计数，再读取 8253 通道的 16 位计数值。这种方法会遗漏在禁止计数时 CLK 信号上出现的脉冲计数。

② 软件控制命令锁存

8253 的计数器锁存功能可以有效地解决这个问题。在读取通道计数器当前值时，可先锁存计数值，再读取。锁存计数值的操作称为计数器锁存命令。8253 的每个通道都有一个输出锁存器（16 位），向通道写入锁存的控制命令时，它把计数器的现行值锁存，而计数器的计数过程照样进行。这样，CPU 读取的是锁存器中的值。当重新写入一个命令字或 CPU 读取计数值后，计数通道会自动解除锁存状态。相应程序如下：

```
MOV     AL, 0000XXXXB
OUT     控制口, AL             ; 发送锁存命令
IN      AL, 通道0              ; 读通道计数器内容
MOV     CL, AL
IN      AL, 通道0
MOV     CH, AL
```

立即数 0000××××表示对通道 0 进行锁存操作。前面两个 0 表示通道 0，后面的两个 0 表示锁存操作，最后的 4 位可以为随意态。

3．工作方式选择（M_2，M_1，M_0）

表 7-13 为 8253 的每个通道的 6 种工作方式。

表 7-13　8253 工作方式选择

M_2	M_1	M_0	工作方式选择
0	0	0	工作方式 0
0	0	1	工作方式 1
×	1	0	工作方式 2
×	1	1	工作方式 3
1	0	0	工作方式 4
1	0	1	工作方式 5

这里要指出的是如果控制字的 $D_5D_4=00$，表示为锁存命令。这时控制字中的低 4 位无效，即 $D_3D_2D_1$ 的组合并不表示工作方式。如果 D_5D_4 不为 00，表示为对通道计数器的读/写方式，这时控制字的低 4 位有效，$D_3D_2D_1$ 的组合就表示工作方式选择。

4．计数码选择

表 7-14　计数码选择

BCD	计数进制
0	二进制
1	BCD

8253 提供了比较丰富的计数制式。通道计数器可以工作在二进制计数状态，也可以工作在十进制计数状态。控制字的最低一位 D0，就是用来决定通道计数器在减 1 计数过程中采用的是二进制计数还是十进制计数，如表 7-14 所示。

在 BCD 计数制下，写入初值范围为 0000～9999，而 0000 是最大值，代表 10000。在二进制计数制下，写入初值范围为 0000H～FFFFH，其中 0000 为最大值，代表 65536。

8253 的使用非常简单，每个 8253 的通道在工作之前先写入控制字，再写入计数初值就可以了。初值的选择应考虑两个方面：数制和大小。

7.2.4　8253 的工作方式

1．方式 0：计数结束中断方式

方式 0 的作用就是用户可以在设定时间上产生中断信号。当控制字写入控制寄存器后，输出端 OUT 变低，计数初值再写入通道后计数器就可以工作了。

（1）工作方式 0 的特点

门控信号 GATE 必须为 1，计数器才能计数。

计数时通道输出端 OUT 一直为 0。

通道计数器计数到 0 后，OUT 由 0 到 1，同时计数器停止工作。

OUT 输出从低到高的正跳变或高电平，可以作为中断请求信号 INTR，向 CPU 发出中断请求。在这种方式下，计数初值为一次性使用有效。当再次向 8253 写入控制字和新的计数初值后，可重新开始定时或计数。在计数过程中，可以通过 GATE 信号，允许或禁止计数。GATE 信号为低电平时，停止计数，一旦 GATE 变高，则继续计数。

8253 工作在方式下 0 的定时波形如图 7-27 所示。

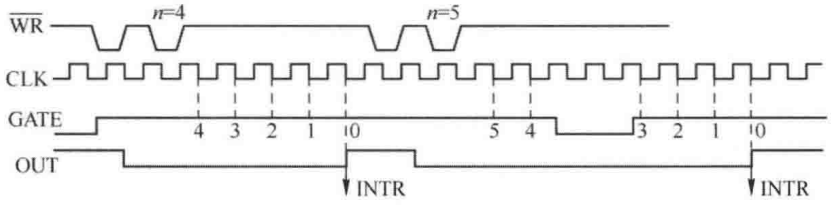


图 7-27　8253 工作在方式 0 下的定时波形

（2）工作方式 0 的应用

这里举一个有趣的例子，帮助读者了解和掌握方式 0 的应用。

每天中午下班时，希望回到家时饭菜准备好了。如何做到这一点？可以有多种方案，这里用 8253 来完成这个任务。具体过程是先按图 7-28 所示做一个家庭厨房系统。

这个系统由 8253、8255A、继电器以及一些家用电器组成，如微波炉、电饭煲等。8255A 的 C 口第 0 位即 PC_0 控制继电器。PC_0 为高时，继电器闭合，家用电器与电源线相连而工作。

如何使用这个家庭厨房系统呢？当然，首先把要做的饭、菜加工处理好，放在相应的家用电器中，然后编写一个程序来运行这个系统。这个程序先对 8253、8255A 进行设置。对 8255A 的设置比较简单，只要把 C 口低 4 位置为输出就可以了。对 8253 的设置稍微复杂一些，包括方式字、时间常数。如果选用其计数通道 0，16 位读/写方式，工作方式 0，二进制计数，则方式字为 00110000B。对时间常数的选择要通过简单计算。如果 CLK 输入的定时脉冲频率是 1 Hz，4 个小时后通道计数器“溢出”，产生中断信号给 CPU，则时间常数=4×3600=14400（秒）=3840H（秒）。

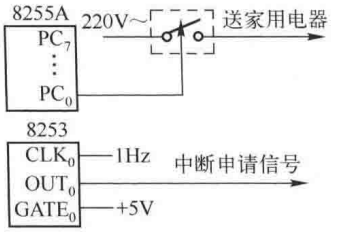


图 7-28　方式 0 下的家庭厨房系统

相应的程序段如下：

```
……
MOV     AL, 10011010B                ; 置 8255A 方式控制字
OUT     8255 控制口, AL
MOV     AL, 00H                      ; 8255A 置位/复位控制字，使 PC0=0
OUT     8255 控制口, AL
MOV     AL, 30H                      ; 置 8253 通道 0 方式控制字
OUT     8253 控制口, AL
MOV     AL, 40H                      ; 置 8253 通道 0 时间常数
OUT     通道 0 端口, AL
MOV     AL, 38H
OUT     通道 0 端口, AL
……
```

中断程序中有关程序段如下：

```
……
MOV     AL, 00000001B                ; 8255 置位/复位控制字，使 PC0=1
OUT     8255 控制口, AL
……
```

这样，你可以早晨启动计算机，8 点钟时执行包含上面程序段的程序，然后就可以上班了。整个上午，8253 都在进行计数，当记录 14 400 个定时脉冲后，4 小时到，也就是 12 点，计数器溢出，OUT 信号由低变高，产生中断信号。这时 CPU 如果满足一定的条件，则中断正在运行的程序，转向执行中断服务子程序。因为已在中断服务子程序中设置了对 8255A 的 C 口的操作，所以这时 C 口的 PC0 由 0 变为 1，继电器闭合，家用电器工作。所以当你 12 点 20 分回到家时，就可以吃到刚刚做好的饭菜了。当然，这时你千万不要忘记关掉所有相关电器和电路的电源。

这个简单的例子清楚地表明，8253 的工作方式 0 的功能就是用户可以使 CPU 在预先设定的时间点上处理某个事务。

2．方式 1：可编程的单稳负脉冲

所谓单稳电路，就是在输入的激励下产生固定宽度的脉冲电路。它的输入/输出关系是：输入端输入一个不低于规定的最小宽度的脉冲后，单稳电路就输出一个用户事先设定宽度的脉冲。输入脉冲称为触发脉冲，单稳电路一般由触发脉冲的沿触发。常见的单稳电路主要由触发器和定时电阻、定时电容组成，输出脉冲的宽度由定时电阻和电容的数据确定，不能随意更

改。

8253 的方式 1 是一个单稳电路。与纯硬件组成的单稳电路不同的是，用户可以通过编程，灵活地选择输出脉冲的宽度，因此使用起来非常方便。

在这种方式下，当控制字写入控制寄存器后，输出端 OUT 变高。另外，初始计数值写入通道后并不开始计数工作，而是等待触发信号的到来。

（1）工作方式 1 的特点

门控信号 GATE 是触发信号，上升沿有效，即开始计数是由 GATE 的上升沿触发的。

触发后，通道计数器开始计数，输出端 OUT 由高变低。

计数器计数到 0，OUT 再由低变高。

上述过程可以一直重复下去，只要给 GATE 门输入一个触发信号，其 OUT 就会输出一个固定脉冲宽度的负脉冲。因而这种方式又称为单拍脉冲方式。输出的负脉冲宽度（Pulse Width，PW）可由下式计算：

$$PW=NT_{CLK}$$

这里，N 为初始计数值，T_{CLK} 为 CLK 端的脉冲周期。

在通道计数的过程中，新来的 GATE 上升沿可使计数器重新开始从初值起计数。由于在新的 GATE 上升沿到来时，计数器没减到 0，故 OUT 输出仍保持低电平，直到重新计数时计数器减到 0，OUT 输出才跳变为高。这种情况的最终效果是使 OUT 输出的单拍脉冲宽度增加。这个特点的应用将在下面例子中给出。

方式 1 的定时波形如图 7-29 所示。

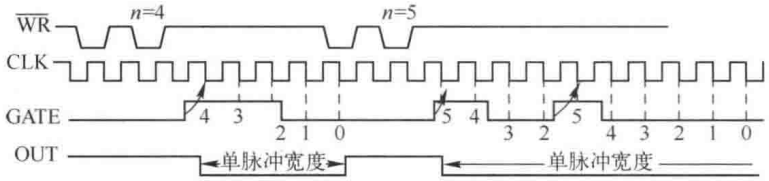


图 7-29　方式 1 的定时波形图

（2）方式 1 的应用

在本节开始时曾经给读者一个例子，介绍计数器电路在高速公路汽车检测中的应用，现在继续对这个课题进行讨论。不同种类的车对光路的作用是不同的，在保持一定速度下，小车对光路阻断的时间短，大车对光路阻断的时间长。有些货车有两节车厢，它对光路阻断两次，因此测试电路将发两个脉冲给计数器（如图 7-30 所示）。从有效信号角度来看，这就是一个光路的干扰，它使计数系统把一个货车当作两辆汽车来处理，这显然是错误的。

用 8253 的方式 1 可以轻松地解决这个问题。首先把原始计数脉冲送到一个通道（如通道 0）进行预处理，输出经过非门处理后的信号再送给另一个通道（如通道 1）进行车辆计数（如图 7-31 所示）。

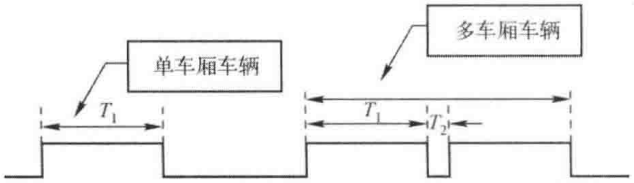


图 7-30　车型与输出脉冲的关系

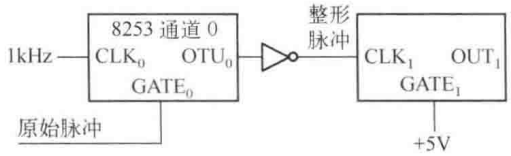


图 7-31　整形脉冲电路

作为预处理信号的通道 0 设置为方式 1 工作模式。原始计数脉冲不通过 CLK 端而是通过 GATE 端经通道 0 进行脉冲整形，其输出的脉冲宽度将不是原始脉冲的宽度，而是和用户精心选取的初始计数值 N 有关（如图 7-32 所示）。

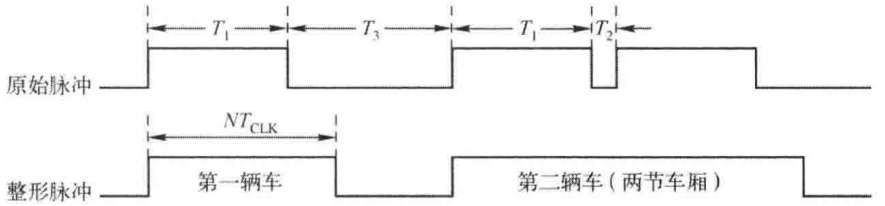


图 7-32　整形脉冲

在单车厢的车辆通过时，通道 0 输出的脉冲宽度为 NTCLK；而多车厢的车辆通过时，通道 0 的输出将有所变化。在要求保持一定速度的前提下，车厢对光路的作用时间是有一定范围的，设最大时间为 T_1，两车厢之间的间隙对光路的最大作用时间为 T_2，而前后两车辆之间的时间间隔为 T_3，只要选取的时间常数 N 满足下面关系，就可以滤去由同一车的不同车厢对光路的干扰，即：

$$T_1 + T_3 > NT_{CLK} > T_1 + T_2$$

3．方式 2：速率发生器

一般来讲，在这种工作方式下，输入信号是周期性的脉冲信号，从 CLK 端引入，输出信号也是周期性的脉冲信号，由 OUT 端输出。从输入信号的频率和输出信号的频率关系上来看，方式 2 实际上是一个可编程的分频电路，它把输入信号分频后以脉冲的形式输出，而分频系数就是用户事先对通道计数器写入的初始计数值。

在这种方式下，当控制字写入控制寄存器后，输出端 OUT 变高。当初始计数值写入通道后，计数器就可以对外电路做出响应了。

方式 2 的特点如下：

GATE 门为 1，计数器才能工作，对 CLK 端上的脉冲进行计数。

当计数器“减 1”计数到 1 时，输出端由高变低，再经过一个 CLK 周期，即计数器计数到 0 时，输出端 OUT 又跳变为高。所以方式 2 输出周期性负脉冲信号，其宽度固定为一个 CLK 周期。

当计数器的值减为 0 时，自动重新装入计数初值，实现循环计数，如图 7-33 所示。

在计数过程中如果 GATE 信号为低电平，则停止计数，待 GATE 信号变为高电平后，从初始值开始重新计数。通常在使用方式 2 时，GATE 端事先被置为高电平。

方式 2 的定时波形如图 7-34 所示。

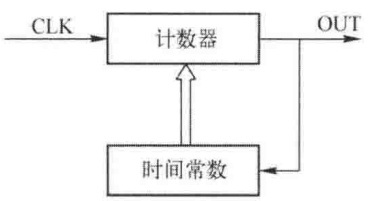


图 7-33　方式 2 计数初值重装

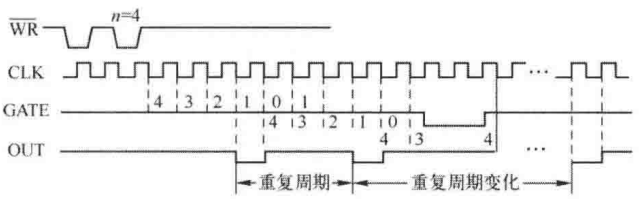


图 7-34　方式 2 的定时波形图

4．方式 3：方波发生器

方式 2 虽然可以作为分频电路，但其输出是窄脉冲。如果要求做一个秒信号发生器，其输出带动一个发光二极管，在一个周期内，发光二极管点亮 0.5 秒，熄灭 0.5 秒，即 OUT 端输出方波信号，用方式 2 就不可能实现了。

方式 3 就可以实现这种要求，与方式 2 工作过程几乎完全一样。在方式 3 中，当控制字写入控制寄存器后，输出端 OUT 变高。当计数初值写入通道且 GATE 为高电平时，计数器开始计数，OUT 保持高电平。它与方式 2 的区别是：若计数初值 n 为偶数，则当计数值减到 $n/2$ 时，输出端 OUT 变为低电平，然后此低电平一直保持到计数值减为 0，OUT 再次变为高电平。

当计数值减到 0 时，计数器重新装入计数值，实现循环计数。可见，当计数值为偶数时，输出端 OUT 输出重复周期为 $N\times$CLK，占空比为 1:1 的方波。当计数初值为奇数时，输出重复周期为 $N\times$CLK，占空比为 $[(N+1)/2]:[(N-1)/2]$，因而输出是近似方波。

在该方式下，若在计数期间写入一个新的计数值，并不立即影响现行计数过程，等到计数值减到 0 后，重新装入新的计数值，开始以新的速率输出方波。方式 3 的定时波形如图 7-35 所示。

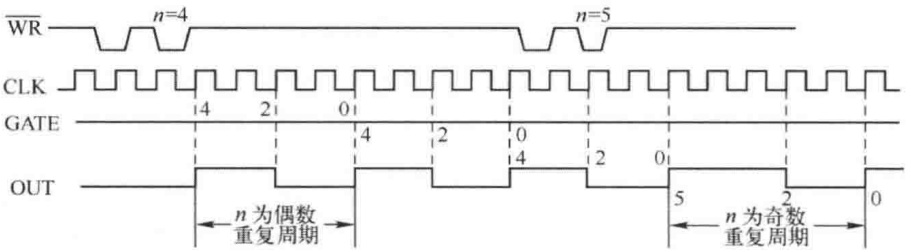


图 7-35　方式 3 的定时波形图

在计数过程中，GATE 信号变低，则禁止计数。当 GATE 变为有效时，重新从初始值开始计数。与方式 2 相似，在方式 3 中，GATE 端通常事先被置为高电平。

下面举一个例子说明 8253 芯片在 PC 中的应用。在 PC/XT/AT 家族中，计时器芯片的 3 个通道都有各自的专用功能。

通道 0 是系统的时钟节拍计时器。当计算机冷启动时，ROM BIOS 对计数器初始化编程，每秒产生约 18.2 个节拍的中断信号，向中断控制器 8259A（IR_0）提出中断申请。

通道 1 专用于产生动态 RAM 的定时刷新信号。通道 2 用来控制计算机的扬声器的声音频率。8253 的电路图如图 7-36 所示。

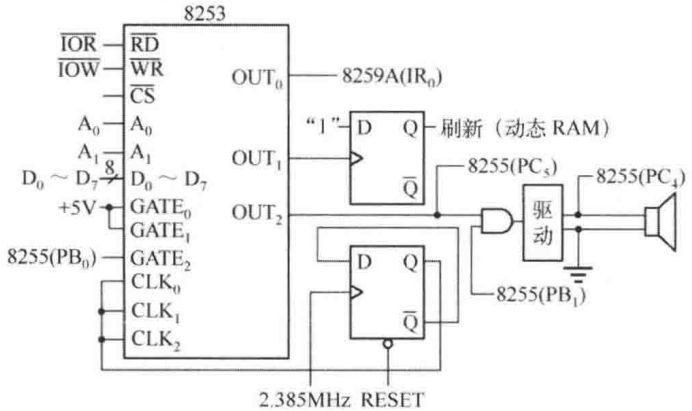


图 7-36　8253 电路连接图

8253 的通道 0 用于产生固定频率的时钟节拍，故设定工作在方式 3，初值为 0。所以从 OUT0 引脚的输出方波频率为 1.19 MHz/216 =18.2 Hz，通道 0 的控制字为 36H。

8253 的通道 1 用于固定频率的刷新信号，故设定工作在方式 2，计数值为 12H=18，速率输出频率为 1.19 MHz/18=66.1 kHz，即定时时间为 15.13 μs，通道 1 的控制字为 54H。

8253 的通道 2 用于产生频率信号，故取工作在方式 3，计数值为 6A4H=1190，方波输出频率为 1.19 MHz/1190=1 kHz。此信号频率决定扬声器的音调，通道 2 的控制字为 0B6H。

下面来看看扬声器的接口。

8253 通道 2 的计数由 8255A 的 PB0 控制，当 PB_0 输出为高电平时，使门控 $GATE_2$ 为高电平。此时，8253 通道 2 允许计数，故通道 2 的输出方波受 PB_0 的控制，从而控制扬声器的音调高低。通道 2 的输出能否对扬声器产生持续控制还取决于 8255A 的 PB_1。当 PB_1 为“0”时，OUT_2 不能通过“与门”；反之，则可通过“与门”控制扬声器。所以，扬声器发音时间的长短取决于 8255A 的 PB1 信号。另外，CPU 通过读 8255A 的 C 口，得知 8253 通道 2 的状态和扬声器驱动器的状态。

5．方式 4：软件触发方式

软件触发方式实际上就是 CPU 通过指令触发一个选通信号给外部设备，选通信号在触发后设定时间点上发出。选通信号的作用是什么呢？几乎所有的外设只要工作在条件传输方式中都要有选通信号，如 8255A 的 A 口或 B 口在方式 1 的输入工作状态下就有选通信号 $\overline{STB}$。当其有效时，外设的数据就会进入端口的输入锁存器中，启动 CPU 从相应端口取数。打印机也有一个选通信号，当其有效时，打印的数据进入打印机的内部缓冲区，打印机就可以打印这个数据了。从这两个例子可以看出，选通信号的功能就是启动一个事件，如数据传输、打印等。

因此，方式 4 就是 CPU 通过触发 8253 的某个通道，定时启动一个事件或工作过程，或 CPU 启动某个事件的命令延迟设定时间后才执行。

如何产生这个触发信号呢？8253 给用户两种方法：一种是软件触发方式，即方式 4；另一种是后面要讲的硬件触发方式，即方式 5。在方式 4 中，当写入控制字后，输出端 OUT 变为高电平。当计数初值写入通道后，CPU 就完成了对通道的触发。当计数器计数到 0 时，通道的 OUT 端就输出负脉冲。

（1）方式 4 的特点

门控信号 GATE 为高电平，计数器开始减 1 计数，OUT 维持高电平。

计数器减到 0 时，输出端 OUT 变低，再经过一个 CLK 输入时钟周期，OUT 输出又变高。

所以，输出端 OUT 在计数器溢出时产生一个宽度为 1 个 CLK 周期的负脉冲。而这个负脉冲就可作为外设的选通信号，其波形如图 7-37 所示。

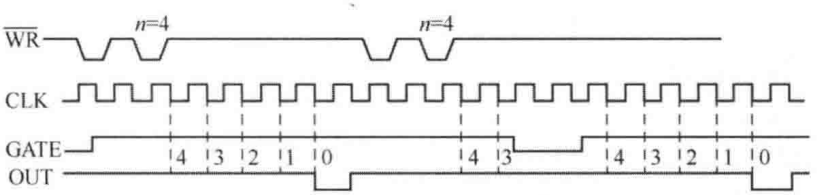


图 7-37　方式 4 的定时波形图

在此方式中，计数值一次有效。若在计数过程中，GATE 信号变低，则停止计数；到 GATE 信号变高，重新开始从初值减 1 计数。在实际工作中，GATE 门通常是保持高电平的。

利用这种工作方式可以完成定时功能，定时时间从装入计数值开始，计数值减到0，OUT端输出负脉冲，表示定时时间到，其定时时间为 n 个CLK端上的脉冲周期（n 为计数值）。这种方式也可完成计数功能，要求计数事件以脉冲的形式从CLK端输入，将已知的计数值装入计数器后，GATE为高电平，则开始计数；每计数一个事件（即CLK脉冲），计数器减1，减到0，则OUT端输出负脉冲，表示计数次数到。所以，方式4也适合于计数值为已知的情况，如定额包装等。

8253的工作方式4与方式0很相似，只是在方式0下，计数“溢出”时输出端为从低到高的正阶跃信号；而在方式4下，计数“溢出”时输出端为负脉冲信号。这两种信号均可向CPU发出中断申请。

（2）方式4的应用

让我们结合方式4再讨论前面所讲的家庭厨房系统。在那个系统中采用的是方式0，当设定的时间到时，OUT产生中断信号给CPU，然后CPU响应中断，在中断服务子程序中完成对家用电器的启动。这个方案不是很理想，因为CPU在8253计数期间一直是运行的。即使在计数期间CPU可能什么事情也没做，但它必须处于开机状态，等待中断信号的到来。现在我们就可以用方式4来完善这个系统了。图7-38是修改后的电路图。在图7-38中，D触发器74LS74取代了8255A。8253通道0的OUT接D触发器的置1端。在上电时，电容C的电压不能跳变，清零端 $\overline{R}_D$ 有效，D触发器清零。当OUT输出一个负脉冲后，D触发器就会被置1。

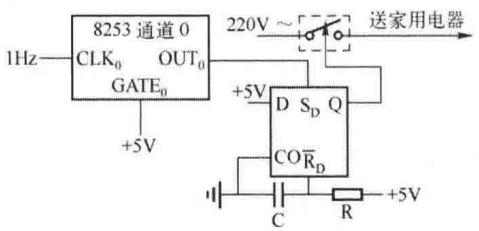


图 7-38　方式4下的家庭厨房系统

这样，可以早晨启动计算机，在8点钟执行对8253通道0的初始化程序，程序仅含有对通道0的方式4控制字设置以及对通道0的初始计数值设置。接着，可以关掉CPU等其他电路的电源，仅保留该系统电路的电源，然后可以上班了。整个上午，8253都在进行计数，记录14400个时钟脉冲后，4小时到，也就是12点，计数器溢出，OUT端输出负脉冲信号，D触发器置位，继电器闭合，家用电器工作。12点20分回到家时就可以吃到刚刚做好的饭菜了。

这个例子实际上相当于CPU对家用电器的启动命令被延迟设定时间后发出的。

这个例子仍有让人不满意之处：必须在早晨8点执行对8253通道0的初始化程序，否则这个系统就不可能精确地在中午12点启动家用电器工作。如此严格地按时开机和运行程序要求可能会使用户对此系统略感遗憾。所以，这个系统还有待完善的空间。

6．方式5：硬件触发方式

硬件触发方式实际上是外部通过一个有效沿触发信号，启动一个选通信号给外部设备，选通信号在触发后设定时间点上发出。

（1）方式5的工作过程

这种工作方式与方式4相似，控制字写入控制寄存器后，输出端OUT变高。同方式4不同的是，当计数值写入通道计数器后，通道并未被触发，即计数器并不立即开始计数。只有当GATE信号的上升沿触发通道后，通道计数器才开始计数，所以方式5称为硬件触发。当计数值减到0，输出端OUT变低，再经过一个CLK时钟周期，OUT端输出又变高。由此可知，OUT输出的也是一个宽度固定为1个CLK周期的负脉冲。方式5的定时波形如图7-39所示。

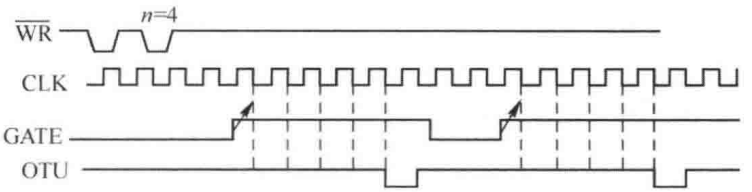


图 7-39　方式 5 的定时波形图

在这种方式下，计数器的计数值减到 0 后，将自动重新装入计数值，但并不开始计数，待到再一次有 GATE 的上升沿触发才开始计数。如果需要改变计数值，CPU 可在任何时候用输出指令装入新的计数值，但不影响正在进行的计数过程，而是当上次计数操作结束，且有 GATE 上升沿触发后，才开始以新的计数值计数。

（2）方式 5 的应用

利用方式 5，可以继续完善上面讨论的家庭厨房系统，图 7-40 是相应的修改电路。

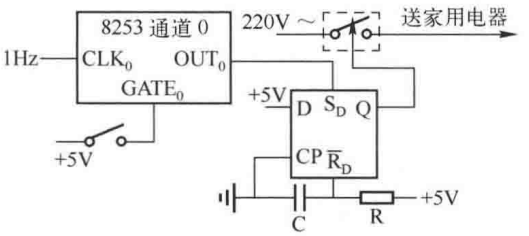


图 7-40　方式 5 下的家庭厨房系统

在电路中增加一个开关电路，用来产生 GATE 信号。当相应的通道被初始化后，如果按下开关，则通道被触发，计数器就开始计数工作。这样，可在早晨 8 点前的任何一个时刻启动计算机，执行对 8253 通道 0 的初始化程序，程序仅含有对通道 0 的方式 5 控制字设置以及对通道 0 的初始计数值设置。接着，你就可以关掉 CPU 等其他电路的电源，而仅保留该系统电路的电源。在 8 点时按下开关，计数器开始计数，然后可以上班了。在 12 点时，计数器计数到 0，OUT 信号输出负脉冲，D 触发器置位，继电器闭合，家用电器工作。

这个例子实际上相当于硬件开关对家用电器的启动命令被延迟设定时间后发出。

7．8253 的工作方式小结

8253 共有 6 种工作方式，方式之间既有相似之处又有区别。对初学者来讲，这就增加了学习和掌握 8253 的难度。从信号和系统的角度来看 8253，了解其输入信号和输出信号的关系，也许是掌握这类器件的一个有效方法。8253 的每个通道给用户 3 条信号线：CLK、GATE 和 OUT。CLK 的使用很简单，在 6 种方式中，它一直充当脉冲输入线。GATE 和 OUT 的使用相对复杂些，因此它们（特别是 GATE 信号）是正确使用 8253 的关键。比如，GATE 信号作为各通道的门控信号，对于各种不同的工作方式，它起的作用是不相同的。在 8253 的应用中，必须正确使用 GATE 信号，才能保证各通道的正常操作。另外，8253 在各种工作方式下，输出端 OUT 输出的波形也不一样。还有，计数初值在某些工作方式下只能用一次，而在另外一些方式下可以自动重新装入，实现循环计数。

从上述各工作方式看出，8253 的 6 种工作方式可以归为两类：一类充当频率发生器，另一类主要作为计数器来使用。下面从这个角度来总结 OUT 和 GATE 的作用。

（1）与频率发生器有关的工作方式

8253 有两种方式与频率发生器有关，即方式 2 和方式 3。对于 OUT 端，方式 2 提供给用户的是负脉冲，方式 3 提供给用户的是方波。在这两种方式下，GATE 信号需要始终保持为高。另外，计数初值被自动重新装入，实现循环计数。

（2）与计数器有关的工作方式

对于计数器类，有方式 0、1 和方式 4、5。启动计数器的方式有两种。一种是 CPU 把时间常数写入相应通道后，计数器开始工作，这称为软件启动方式。在这种启动方式下，GATE 需要始终保持为高电平，所以方式 0 和方式 4 可以称为软件启动方式。另一种是硬件启动计数器，即 CPU 把时间常数写入计数器后，即使 GATE 为高电平，计数器并不工作。只有 GATE 发生跳变，其上升沿启动计数器工作，所以方式 1 和方式 5 可以称为硬件启动方式。计数器溢出时，OUT 有两种输出形式：要么是正电平，要么是负脉冲。前者有方式 0 和方式 1，后者有方式 4 和方式 5。

表 7-15 总结了 8253 工作在各种方式下，对 GATE 信号的要求以及计数初值的使用情况。

表 7-15　8253 在不同方式下的工作特点

工作方式	启动方式	“溢出”方式	计数值使用	工作方式	启动方式	“溢出”方式	计数值使用
方式 0	软件启动	OUT 为正电平	一次有效	方式 3	软件启动	OUT 为方波	自动重装
方式 1	GATE 上升沿	OUT 为正电平	自动重装	方式 4	软件启动	OUT 为负脉冲	一次有效
方式 2	软件启动	OUT 为负脉冲	自动重装	方式 5	GATE 上升沿	OUT 为负脉冲	自动重装

7.2.5　8253 应用举例

与 8255A 一样，8253 在正常工作前，必须首先对其初始化编程。

<1> 通过 8253 的控制端口向控制字寄存器写入相应通道的控制字，控制字包括：指定通道的工作方式，对通道计数器的读写方式，通道计数器计数时所采用的数制信息。

<2> 通过 8253 的通道端口向相应的通道计数器写入初始计数值。若在控制字中已确定 16 位的读/写方式，则对通道端口写操作两次，第一次写初始计数值的低 8 位，第二次写高 8 位。

【例 7-5】 现有一个高精密晶体振荡电路，输出信号是脉冲波，频率为 1 MHz。要求利用 8253 做一个秒信号发生器，其输出接一个发光二极管，以 0.5 s 点亮、0.5 s 熄灭的方式闪烁指示。设 8253 的通道地址为 80H～86H（偶地址）。

显然，这个例子要求用 8253 做一个分频电路，而且其输出应该是方波，否则发光二极管不可能等间隔闪烁指示。频率为 1 MHz 信号的周期为 1 μs，而 1 Hz 信号的周期为 1 s，所以分频系数 N 可按下式进行计算：

$$N=\frac{1\text{s}}{1\mu\text{s}}=\frac{1000000\mu\text{s}}{1\mu\text{s}}=1000000$$

由于 8253 一个通道最大的计数值是 65536，所以对于 N=1000000 这样的数，一个通道是不可能完成上述分频要求的。由于 $N=1000000=1000\times1000=N_1\times N_2$，可用如图 7-41 所示的通道计数器级联的方法来实现分频系数超过 216 的分频要求。图 7-41 所示电路的物理意义非常明显，通道 0 首先把 1 MHz 信号 1000 分频，产生 1 kHz 的信号，通道 1 再把 1 kHz 信号 1000 分频，结果就得到 1 Hz 信号。

现在，讨论通道 0 和通道 1 的工作方式。由于通道 1 要输出方波信号推动发光二极管，所以通道 1 应选工作方式 3。对于通道 0，它只要能起分频作用就行，对输出波形不作要求，所以方式 2 和方式 3 都可以选用。

这样，通道 0 取工作方式 2，BCD 计数；通道 1 取工作方式 3，二进制计数（当然也可选 BCD 计数）。注意：在方式 2 和方式 3 中，GATE 门要保持高电平。

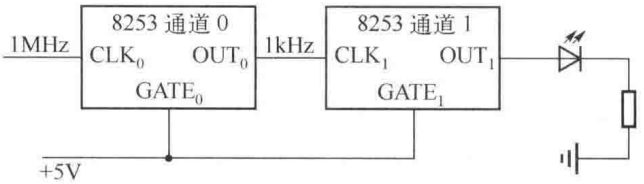


图 7-41　8253 通道级联

相关程序如下：

```
MOV     AL, 00110101B           ; 通道 0 控制字
OUT     86H, AL
MOV     AL, 00                  ; 通道 0 初始计数值
OUT     80H, AL
MOV     AL, 10H
OUT     80H, AL
MOV     AL, 01110110B           ; 通道 1 控制字
OUT     86H, AL
MOV     AL, 0E8H                ; 通道 1 初始计数值，03E8H=1000BCD
OUT     82H, AL
MOV     AL, 03H
OUT     82H, AL
……
```

【例 7-6】 计件系统。计件系统的功能就是记录脉冲的个数。一个脉冲代表一个事件，如交通道路检测系统中通过检测点的车辆，工业控制系统中流水线上加工好的工件。要求在计件过程中，计算机可以显示当前计数器的内容，当完成 10 000 个工件记录后，系统发出 1 kHz 信号推动喇叭发音，通知用户。

这是一个典型的 8253 作为计数器应用的例子。这里显然需要两个通道：一个作为计数，选用通道 0；另一个产生 1 kHz 信号，选用通道 1。通道 1 的工作方式很容易确定，由于要产生 1 kHz 信号，故选用工作方式 3。至于通道 0 的工作方式，稍微要复杂些。如果要求初始计数值写入计数通道后，计数器就可以工作，则通道 0 的启动方式应是软件启动。由于要求计数溢出后产生一个信号来启动一个事件，即喇叭发音，故可选的工作方式为方式 0 和方式 4。为了简化电路，采用如图 7-42 所示方案。通道 1 的 GATE 信号由通道 0 的 OUT 信号产生，这个 OUT 信号应该是电平型的，所以通道 0 应选用方式 0。

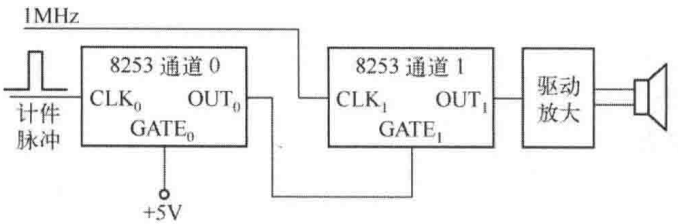


图 7-42　计件系统方案

传感器电路把物理事件转换为脉冲信号输入通道 0 计数，当记录 10000 个事件后，通道 0 计数器溢出，OUT_0 端输出高电平，即 $GATE_1$ 为高电平，这时通道 1 开始工作，产生 1 kHz 信号推动喇叭发音。

如果设通道 1 的 CLK 端输入的时钟信号为 1 MHz，则通道 1 的初始计数值为：

$$N_1=\frac{1\text{MHz}}{1\text{kHz}}=\frac{1000\text{kHz}}{1\text{kHz}}=1000$$

如果设定 BCD 计数，通道 1 的方式控制字为 01110111B。

通道 0 的初始计数值为 10000=2710H。设定通道 0 是二进制计数，其方式控制字为 00110000B。

设 8253 地址为 300H～306H，程序如下：

```
        MOV     DX, 306H                ; 通道 1 初始化
        MOV     AL, 01110111B
        OUT     DX, AL
        MOV     DX, 302H
        MOV     AL, 00
        OUT     DX, AL
        MOV     AL, 10H
        OUT     DX, AL

        MOV     DX, 306H                ; 通道 0 初始化
        MOV     AL, 00110000B
        OUT     DX, AL
        MOV     DX, 300H
        MOV     AL, 10H
        OUT     DX, AL
        MOV     AL, 27H
        OUT     DX, AL

GETDAT: CALL    DELAY                   ; 延时
        MOV     DX, 306H                ; 通道 0 锁存命令
        MOV     AL, 00000000B
        OUT     DX, AL
        MOV     DX, 300H                ; 读通道 0 计数器
        IN      AL, DX
        MOV     CL, AL
        IN      AL, DX
        MOV     CH, AL                  ; 数据存在 CX 寄存器

        CALL    DISPLAY                 ; 显示 CL 内容
        CMP     CX, 0                   ; 检查计数器内容是否为 0
        JNZ     GETDAT                  ; 不为 0，重复
        …                               ; 完成计数，进行其他处理
```

显示程序 DISPLAY 的入口参数是 CX，有两个功能：一个是把 CX 中的二进制转化为十进制及 ASCII 码，另一个功能就是把 ASCII 码通过系统调用显示出来。延时程序 DELAY 的作用是使 CPU 对 CRT 操作不要太频繁。对于 DISPLAY 和 DELAY 程序，读者可以自行编写，这里不再详细讨论了。

习 题 7

1．设某 8086 系统中有 2 片 8255A 芯片，由 74LS138 译码器产生两个芯片的片选信号，如图 7-43 所示。

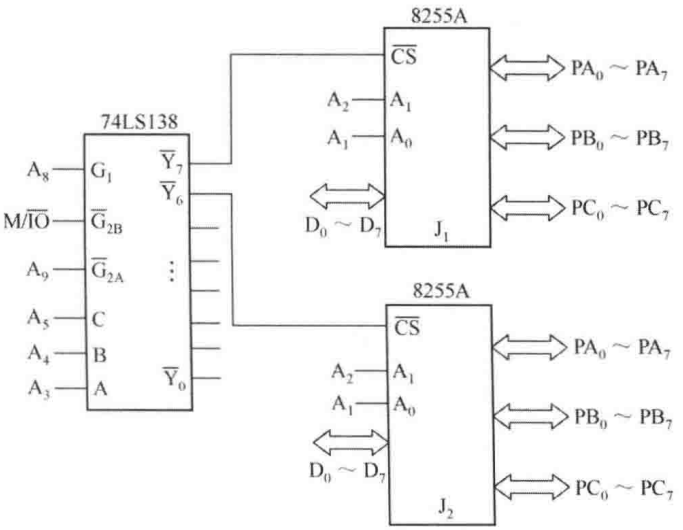


图 7-43　两片 8255A 连接图

要求：第 1 片 8255A 的 A 口工作在方式 0 输出，B 口工作在方式 0 输入，C 口高 4 位为输出，低 4 位为输入；第 2 片 8255A 的 A 口为方式 0 输入，B 口为方式 1 输出，C 口其他输出。

（1）试指出两片 8255A 芯片各自的端口地址。

（2）试写出两片 8255A 芯片各自的方式控制字。

（3）试写出两片 8255A 芯片各自的初始化程序。

2．试指出下列工作方式组合使用时，8255A 芯片 C 口各位的作用。

（1）A 口工作在方式 2，B 口工作在方式 0，输入。

（2）A 口工作在方式 2，B 口工作在方式 1，输入。

（3）A 口工作在方式 2，B 口工作在方式 1，输出。

3．在 IBM/XT 机中，用 74LS138 译码器产生各 I/O 接口芯片的片选信号，如图 7-44 所示。试指出各芯片的端口地址范围。

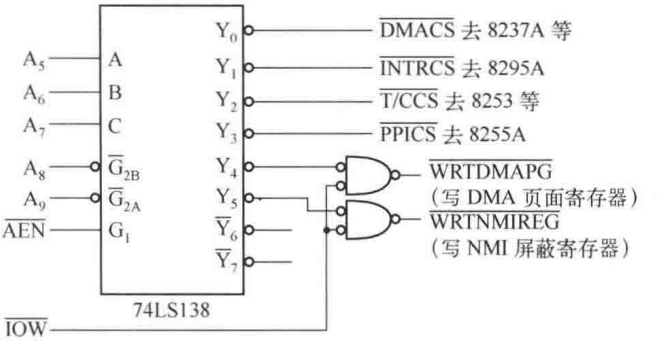


图 7-44　片选信号的产生

4．计数器/定时器 8253 有哪几种工作方式，各有何特点？其用途如何？

5．在某个应用系统中，计数器/定时器 8253 地址为 310H～316H，定时器 0 作为分频器（N 为分频系数），定时器 2 作为外部事件计数器。如何编制初始化程序？

6. 某系统中 8253 芯片的通道 0～通道 2 和控制端口地址分别为 0FFF0H～0FFF6H，定义通道 0 工作在方式 2，CLK_0=2 MHz。要求：输出 OUT_0 为 1 kHz 的速率波；定义通道 1 工作在方式 0，其 CLK_1 输入外部计数事件，每计满 1000 个，向 CPU 发出中断请求。试写出 8253 通过 0 和通道 1 的初始化程序，并画出电路图。

第 8 章　串行输入/输出接口

本章导读

- ✡ 串行通信的实现
- ✡ 串行通信的基本概念
- ✡ 可编程串行通信接口芯片 8251 简介
- ✡ 串行通信 RS-232C
- ✡ USB 简介
- ✡ USB 工作原理和传输方式
- ✡ USB 设备列举

计算机与外部信息交换方式有两种：一种是并行通信，另一种是串行通信。并行通信时，数据各位同时传输；而串行通信时，数据和控制信息是一位接一位地串行传输，这样虽然速度会慢一些，但传输距离比并行通信长，硬件电路也相应简单些。因此在长距离通信系统及各类计算机网中，串行传输方式是主要的通信方式。计算机提供给用户的 RS-232C 接口就是一个标准的串行通信接口，主要用来把数据处理装置与数据通信装置连接在一起，如把终端与调制解调器连接在一起。RS-232C 标准包括接口的机械、电气及功能方面的内容。许多场合都以这种接口规范作为连接标准，如老式的绘图仪、逻辑分析仪、打印机等都有 RS-232C 接口。

近 20 多年来，串行接口的另一个发展成果是 USB。USB 串行接口标准是由 Microsoft、Intel、Compaq、IBM 等公司共同推出的，提供机箱外的即插即用连接，用户在连接外设时不用再打开机箱、关闭电源，而是采用 USB 集线器（Hub）方式。每个 USB 设备用一个 USB 插头连接到 USB 集线器的一个插座上，与计算机相连的 USB 集线器又可以与另一个 USB 集线器相连以扩展 USB 插座。通过这种方式的连接，一个 USB 控制器可以连接包括集线器在内的 127 个外设，而每个外设间的距离可达 5 m。一般认为，USB 接口总线将取代计算机机箱后的众多的串/并口（鼠标、Modem、键盘）等插头，甚至取代计算机主板上的 PCI 总线接口和图形板接口。USB 能够智能识别 USB 链上外围设备的插入或拆卸。除了能够连接键盘、鼠标外，目前的 USB 总线还可连接 ISDN、电话系统、数字音响、打印机以及扫描仪等较高速外设。

串行通信技术特别是 USB 技术的日益成熟和接口电路更为简单，数据传输速率也大幅度提高，甚至超过了并行打印机的数据传输速率。所以，串行通信取代并行通信只是时间上的问题了。

8.1 串行通信接口

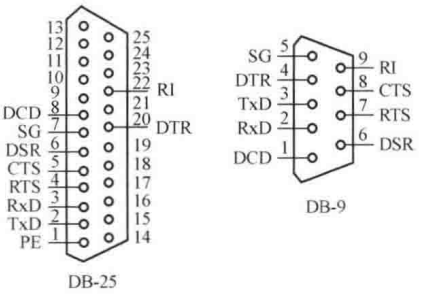


图 8-1 RS-232 串行口

过去的台式计算机一般至少有两个 RS-232 串行口 COM1 和 COM2（如图 8-1 所示），通常 COM1 使用的是 DB-9（9 针 D 形）连接器，COM2 使用的是 DB-25 连接器。目前，不少流行的台式计算机和绝大多数笔记本电脑由于空间的限制，去掉串口和并口，而增加 USB 接口的数量。当需要用笔记本电脑与串口做一个数据传送系统时，可以购买一个 USB 转串口的装置，使用时把这个装置插入 USB 插头即可。这时，操作系统建立一个虚拟的串口环境，用户就可以像使用台式计算机一样使用串口了。

8.1.1 串行通信的实现

1．并行通信和串行通信的特点

并行通信就是同时传输多位数据。CPU 与存储器和 I/O 接口的通信就是并行通信工作方式。并行通信主要由 4 位、8 位、16 位、32 位甚至 64 位以上数据线组成。过去的计算机与外部设备的并行通信如打印机、扫描仪都是 8 位的。当传输字节型数据时，则数据的各位同时传输；而当传输 16 位的字型数据时，则需两次数据传输过程。

并行通信的特点如下。

① 速度快：可以一次传输多位数据。

② 引线较多：如计算机与外设的并行接口就有 8 根数据线，还有数根与外设联系的控制线和状态线，所以总数在 10 位以上（如图 8-2 所示）。

③ 距离短：由于引线较多，排线上线与线之间分布电容的影响比较严重，所以很难实现远距离通信。

串行通信是逐位传输数据的，所以一个多位数据需要多次传输。比如，一个 8 位的字节型数据至少需要 8 次传输，特点如下。

① 速度较慢：如果并行通信一次可以传输 8 位数据，则在相同的数据传输速率下，串行通信只有并行通信的 1/8 的速度。

② 引线少：如果结合其他措施且不考虑地线，单向串行通信用一根线就可以实现数据传输，当串行通信是双向传输时，则两根线就可以了，常用符号 TxD 表示数据的发送线，RxD 表示数据的接收线，所以当两台计算机通过串行线相连时，TxD 线和 RxD 线要交叉相连，连接方式非常简单，如图 8-3 所示。

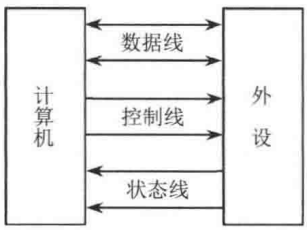


图 8-2 并行通信

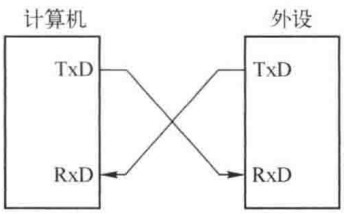


图 8-3 串行通信

③ 距离长：由于串行通信用线很少，故分布电容对通信速度的影响很小，再加上有效的驱动电路，从而使通信距离大大长于并行数据传输。

2．串行通信的硬件条件

串行通信实际上把数据一位一位地发送和接收，而计算机处理数据是并行的，它要传输的数据也是并行的，因此需要一个部件把并行数据与串行数据进行转换。如图 8-4 所示，对于发送数据端来说，这个部件就是并行输入串行输出的移位寄存器，CPU 通过对相应端口的写操作，把要传输的数据写入这个并入串出移位寄存器中，然后移位寄存器在同步时钟的作用下，把数据逐位移出，发送给接收端；对于接收端来说，相应的部件是串行输入并行输出移位寄存器，在同步脉冲的作用下，发送端送来的数据逐位移入这个串入并出移位寄存器中，然后 CPU 对相应端口进行读数操作，把串入并出移位寄存器的数据读入 CPU 中。

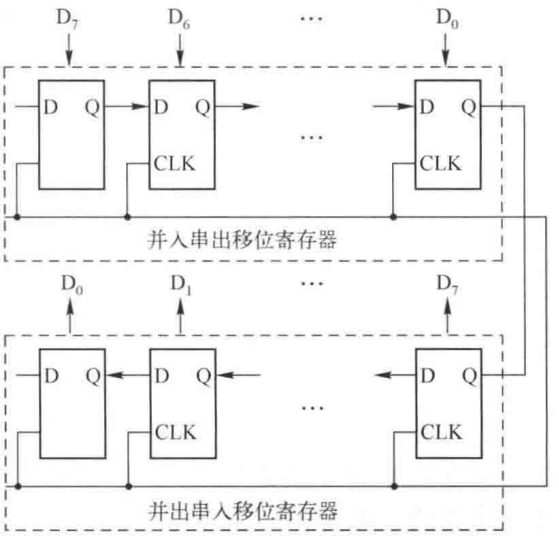


图 8-4　串行通信与移位寄存器

在硬件上，串行通信系统的核心部件是移位寄存器，其中在发送端要有一个并入串出移位寄存器，在接收端要有一个串入并出移位寄存器。

3．串行通信的同步

为了可靠地串行通信，同步信号起着至关重要的作用。在理想情况下，同步信号应该在数据信号线上出现有效数据期间的中心点有效，因为这时数据线上的数据是最稳定的。

注意，串行通信的一个目的就是解决并行通信中数据线太多的问题。为此，在设计串行通信接口时尽量减少所需连接的信号线。那么，可以省去哪些信号线呢？通信中所用的串行数据线是不可能省掉的，唯一可能的方案就是把串行通信中的同步信号线去掉。因此在典型的串行通信系统中，同步信号线是不存在的。

如何在通信系统的接收端恢复这个同步信号？这是实现串行通信必须解决的问题，也是读者理解和掌握串行通信的一个关键。

如何实现串行通信中的数据传输的同步？系统从软件和硬件两方面采取了如下 3 个措施。

（1）设置波特率

波特率是指单位时间内传输的位数，单位是 bps。尽管波特率在理论上可以是任意的，但

考虑到接口的标准性，国际上还是规定了一个标准的波特率系列。常用的波特率有 110、300、600、1200、2400、4800、9600 和 19200（bps）。大多数接口的接收波特率和发送波特率是可以分别设置的，也就是说，它们可以分别由编程来设定。当然，在一个串行通信系统工作时，应该设定接收方和发送方的波特率相同，如图 8-5 所示。

规定波特率可以使串行通信的收、发两端的同步信号的频率一致，这就为在接收端恢复发送端的同步信号提供了可能，但它仍然不能保证接收端的同步信号在数据线有效时出现这个苛刻的要求。

（2）设置数据的传输格式

一般，串行通信在传输数据时并不是单纯地传输数据位信息。为了使数据传送可靠，还需设定其他一些辅助位。如在一种常用的称为异步通信方式中，任何一组数据总是以起始位（低电平）开始、停止位（高电平）结束，在起始位和停止位之间才是有效数据位。另外，数据位的末尾是否用奇偶校验、起始位和停止位选用宽度等，都有一定的规定。例如，传输一个 7 位的 ASCII 字符，附加位有一个偶校验位、一个起始位和一个停止位，则传输的字符由 10 位组成。从起始位开始到停止位结束构成一帧，字符可以一个一个地传输。异步通信的数据格式可用图 8-6 表示。

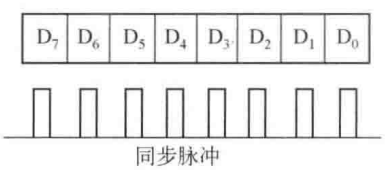


图 8-5　串行通信同步

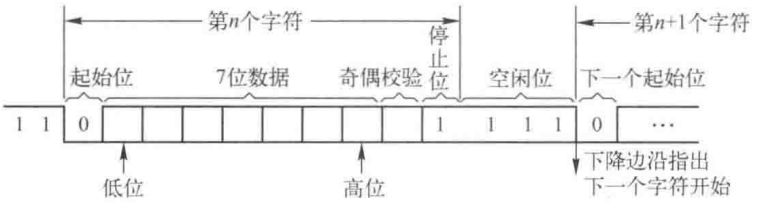


图 8-6　异步通信数据格式

常用的数据格式是 1 个起始位，7～8 个数据位，1 个奇偶校验位（可无），1 个、1 个半或两个停止位。读者也许对 1 个半位的停止位感到困惑，因为半个位在物理上是不存在的。正确的理解或解释停止位如下：停止位就是规定了 2 次收发数据之间的最小时间间隔。比如，对于波特率为 1200 bps 的串行通信，0.833 ms 传输一位数据，1 个半停止位就意味发送端发完一个数据后，至少要等 0.833 ms +0.833 ms/2≈1.25 ms 才能发下一个数据。

有了波特率及数据的传输格式，就可以实现接收端同步信号的恢复和数据的正确接收了。因为波特率确定了同步信号的频率，而数据格式是使接收方知道数据何时发送。下面是一个串行通信的软件仿真实现的例子。

【例 8-1】 设有一个微机化仪器，其内部 8086 CPU 要通过 8255A 实现与计算机的串行通信。设数据的传输格式为 1 个起始位、8 个数据位、1 个停止位。要求传送率为 1200 bps，8255A 的地址分布为 88H～8EH，C 口的第 0 位作为发送数据线 TxD，C 口的第 4 位线作为接收数据线 RxD。

这是一个双向通信，在用软件实现前，先计算数据在传送时每一位所占用的时间。

$$T_d = 1s/1200=1000ms/1200=0.833ms$$

假设程序库中已有两个重要的延时程序：一个是 delay1，其功能是延时 0.833 ms；另一个是 delay2，其功能是延时 0.417 ms，即约为 delay1 延时时间的一半。

相应软件分 3 个程序段：8255A 的初始化、串行发送程序段和串行接收程序段。

① 8255A 初始化

根据上述要求，C 口的低 4 位为输出，高 4 位为输入。A 口、B 口未用，设为方式 0 的输入方式。

```
8255INIT    PROC    FAR                     ; 8255 初始化程序段
            MOV     AL, 10011010B           ; 8255A 方式控制字
            OUT     8EH, AL
            MOV     AL, 00000001B           ; 置位复位控制字，置 PC0 为 1
            OUT     8EH, AL
            RET
8255INIT    ENDP
```

② 送数据程序段

其功能是把 DL 中的数据按照传输的格式发送出去。具体过程如下：

<1> 发送起始位。

<2> 通过寄存器 DL 右移发送数据位，过程如下：令 i=0，发送 DL 中的第 i 位，i+1→i，重复上两步 8 次。

<3> 发送停止位。

<4> 结束。

程序如下：

```
SIO_SEND    PROC    FAR                     ; 把 DL 中的数据通过 PC0 发送
            PUSH    AX                      ; 保护现场
            PUSH    CX
            PUSH    DX
            PUSHF
            CALL    SEND0                   ; 送起始位（发送数据 0）
            MOV     CX, 8                   ; 循环送 8 位
SIO_SEND3:  SHR     DL, 1                   ; 第 1 位进 CF 标志
            JC      SIO_SEND1               ; 判断第 1 位是否为 1
            CALL    SNED0                   ; 第 1 位为 0，则发送数据 0
            JMP     SIO_SEND2
SIO_SEND1:  CALL    SEND1                   ; 第 1 位为 1，则发送数据 1
SIO_SEND2:  LOOP    SIO_SEND3               ; 循环 8 次
            CALL    SEND1                   ; 送停止位（发送数据 1）
            POPF                            ; 恢复现场
            POP     DX
            POP     CX
            POP     AX
            RET                             ; 返回
SIO_SEND    ENDP

SEND0       PROC    FAR                     ; 发送数据 0
            MOV     AL, 00000000B           ; 复位 PC0，PC0=0
            OUT     8CH, AL
            CALL    delay1                  ; 延时 0.833ms
            RET                             ; 返回
SEND0       ENDP
```

```
SEND1       PROC    FAR                     ; 发送数据 1
            MOV     AL, 00000001B           ; 置位 PC0, PC0=1
            OUT     8CH, AL
            CALL    delay1                  ; 延时 0.833ms
            RET                             ; 返回
SEND1       ENDP
```

③ 接收数据程序段

其功能是把接收的数据存入寄存器 DL 中。这段程序比发送程序要稍微复杂些，包括检测发送端是否有数据发送过来，以及如何实现在数据有效期间的中点对接收信号线进行采样。具体算法如下：

<1> 检测 RxD 线上是否是低电平：否，则重复本步骤；是，则执行下一步。

<2> 延时 0.417 ms 对数据再采样，检测是否是低电平：否，则重复上一步；是，则认为是起始位，执行下一步。

<3> 通过寄存器 DL 移位采集数据。

<4> 判断最后 1 位是否是停止位（即逻辑 1）。若是，表示正确接收数据，置 CF 有效。

步骤<1>、<2>的功能就是启动一次接收数据过程，同时使 CPU 在数据位中心点进行采样。

程序如下：

```
; 如果正确接收，那么 CF=1，数据放在 DL 中
SIO_REV     PROC    FAR
            PUSH    AX                      ; 保护现场
            PUSH    CX
            PUSH    DX
SIO_REV1:   IN      AL, 8CH                 ; 等待起始位
            TEST    AL, 00010000B
            JNZ     SIO_REV1
            CALL    DELAY2                  ; 延时 0.417 ns
            IN      AL, 8CH                 ; 在数据中心点再取样
            TEST    AL, 00010000B           ; 再次确认是否是起始位
            JNZ     SIO_REV1                ; 不是起始位，转移
            MOV     CX, 8                   ; 置循环次数，开始采集 8 位数据
SIO_REV2:   CALL    DELAY1                  ; 延时 0.833 ms
            IN      AL, 8CH                 ; 采样
            TEST    AL, 00010000B           ; 判断数据位是否是 0
            CLC                             ;  CF 清零
            JZ      SIO_REV3                ; 数据位为 0，CF=0
            STC                             ; 数据位为 1，CF=1
SIO_REV3:   RCR     DL, 1                   ; 把 CF 右移到寄存器 DL
            LOOP    SIO_REV2                ; 重复采集 8 次
            CALL    DELAY1                  ; 延时 0.833 ms
            IN      AL, 8CH                 ; 取停止数据位
            TEST    AL, 00010000B
            CLC
            JZ      SIO_REV4                ; 数据位为 0，CF=0
            STC                             ; 数据位为 1，CF=1
SIO_REV4:   POP     DX                      ; 恢复现场
```

```
            POP     CX
            POP     AX
            RET                              ; 返回
SIO_REV     ENDP
```

在软件中，通过选择合适的延时，达到在传送数据位中心点对接收信号线采样的目的，有效地接收发送端传送来的数据。

上例用软件实现了在传送数据位中心点对接收信号线采样。现在的问题是：如何在硬件上实现在传送数据位中心点对接收信号线采样？这就引出了波特率因子的概念。

（3）设置波特率因子

用异步通信方式进行通信时，发送端需要用时钟来决定每一位对应的时间长度，接收端也需要用一个时钟来测定每一位的时间长度。前一个时钟叫发送时钟，后一个时钟叫接收时钟。这两个时钟的频率可以是波特率的数倍，一般取 16、32 或 64。这个倍数就称为波特率因子。

图 8-7 取波特率因子为 16。通信时，接收端在检测到电平由高到低变化后，便开始计数，计数时钟就是接收时钟。当计到 8 个时钟以后，就对输入信号进行采样，如仍为低电平，则确认这是起始位，而不是干扰信号。此后，接收端每隔 16 个时钟脉冲 T 对输入线进行一次采样，直到各个信息位以及停止位都输入以后，采样才停止。当下一个出现由 1 到 0 跳变时，接收端重新开始采样。显然，这种方法可以在数据位的中心点采样数据。

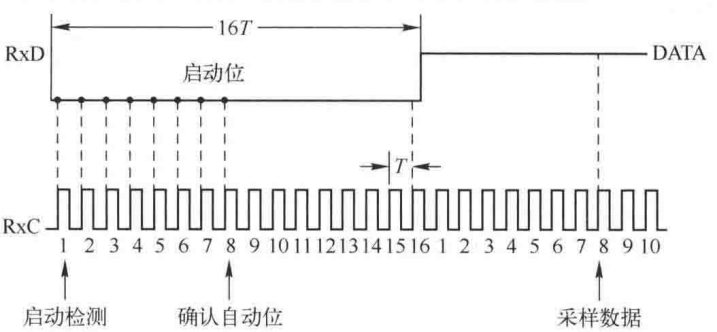


图 8-7　波特率因子为 16 的数据采样

8.1.2　串行通信的基本概念

1．串行通信方式

上面讨论的串行通信为异步串行通信。实际上，串行通信有两种工作方式：异步通信和同步通信。异步通信由于不需要同步信号，硬件连接简单，而被广泛使用于串行部件、计算机与计算机、计算机与单片机及其仪表之间的数据交换。

（1）异步通信

所谓异步通信，是指以字符为单位传输数据，用起始位和停止位标志每个字符的开始和结束，两次传输时间间隔不固定。异步通信不需要同步信号线，为了实现异步通信的要求，CPU 与外设之间有两项规定，即前面已经介绍的字符格式的规定和波特率的规定。这里不再赘述。

（2）同步通信

异步通信为了可靠地传输数据，在每次传输数据时附加一些标志位。在大量数据传输时，为了提高速度，就去掉这些标志，这就是同步通信。采用同步传输时，数据块开始处要用同步

字符来指示，且在发送端和接收端之间要用时钟来实现同步。

作为传输速率比较，考虑这样一个异步传输过程。设每帧对应 1 个起始位、8 个数据位、无奇偶校验位、1 个停止位，如果波特率为 1200，那么每秒所能传输的最大字符数为 1200/10=120 个。再考虑一个同步传输过程。如果选用同样的波特率工作，用 4 个同步字符头作为信息帧头部，数据块为 120，传输时间只用了 8×(120+4)/1200=0.8267 s，不到 1 s。所以在相同的数据传输波特率下，异步通信所传输的数据量要小于同步通信所传输的数据量。

为了保证通信的可靠，同步通信对同步信号的要求要严格得多。因为异步通信中接收端每接收一个数据帧，都要重新检查数据的起始位，对同步信号进行一次新设定。

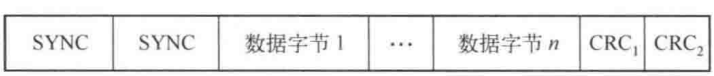
同步通信中使用的数据格式根据所采用的控制规程而定。为在国际上或范围广大的地区内实现数据通信，有必要对数据编码、数据传输速度、同步方式、传输控制步骤、出错控制方式、通信报文格式及控制字符的定义等问题做出统一的规定。通信双方间就如何交换信息所建立的一些规定和过程称为数据通信控制规程，在计算机网络中称为协议。

同步通信中使用的数据格式所采用的控制规程可分为面向字符型和面向位型两种。

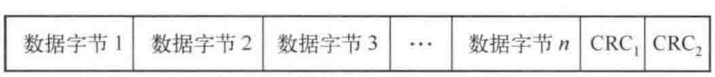
① 面向字符型的数据格式

面向字符型的控制规程的特点是规定一些字符作为传输控制专用，信息长度为 8 的整数倍，传输速率为 200～4800 bps。面向字符型的数据格式又有单同步、双同步和外同步之分，它们的数据格式如图 8-8 所示。

（a）单同步信息式



（b）双同步信息式



（c）外同步信息式

图 8-8　面向字符型的数据格式

单同步是指在要传输的数据块之前，安排一个同步字符 SYNC，接收端检测到该同步字符后开始接收数据；双同步是指在数据块之前，必须安排两个同步字符 SYNC，表示一帧数据开始；外同步格式中数据之前不需同步字符，而是用一条专用控制线来传输同步字符，以实现收发双方的同步操作。任何一帧信息都以 2 字节的循环控制码 CRC 为结束。

② 面向位型的数据格式

面向位型的控制规程，它的特点是没有采用传输控制字符，而是采用某些位组合作为控制用，其信息长度可变，传输速率在 2400 bps 以上。这一类型中最有代表性的规程是 IBM 的同步数据链路控制规程，简称 SDLC。

SDLC 的数据格式如图 8-9 所示。数据以帧为单位传输，每帧由 6 部分组成，以标志字节 01111110B 开始，后跟 1 个字节的地址场及根据需要设置 1 字节的控制场，后面是需要传输的数据（位的集合）、1～2 字节的帧校验码 CRC，最后以标志字节 01111110B 为结束。显然，这种方式不允许在地址段、数据段和 CRC 字段中出现连续的 6 个“1”，否则会误认为是结束标

志，因此要求在发送端查询，一旦连续出现 5 个“1”，则立即插入一个“0”，到接收端要将这个插入的“0”去掉，恢复为原来数据，只有这样才能保证通信的正常进行。

2．串行通信中的数据传输方向

通常，串行通信数据在两个站之间是双向传输的，A 站可作为发送端，B 站作为接收端；也可以 A 站作为接收端，B 站作为发送端，根据要求可分为半双工和全双工。

半双工：每次只能有一个站发送，即只能由 A 站发送到 B 站，或由 B 站发送到 A 站，A 站和 B 站不能同时发送。

全双工：两个站都能同时发送称为全双工。数据传输方向示意图如图 8-10 所示。

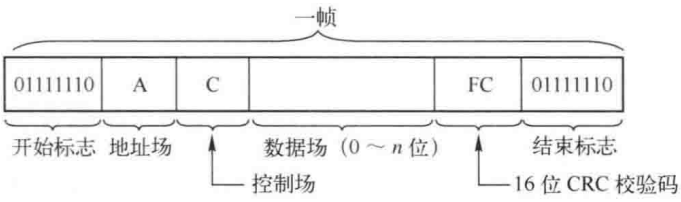


图 8-9　SDLC 的数据格式

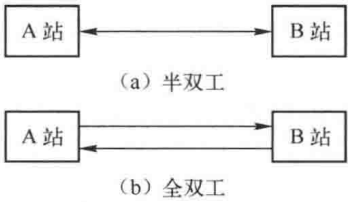


图 8-10　串行通信中数据传输方向

3．异步接收/发送器（UART）

串行接口的基本结构主要是异步接收/发送器（UART），它不仅包括并行数据和串行数据之间的相互转换，还包括检测串行通信在传输过程中可能发生错误的逻辑部件。

（1）基本结构

UART 的内部结构如图 8-11 所示，包含前面所叙述的串行移位寄存器和其他辅助部件。它既能发送，由并行输入转串行输出，又能接收，由串行输入转并行输出；接收和发送部分都是一个双缓冲器结构。

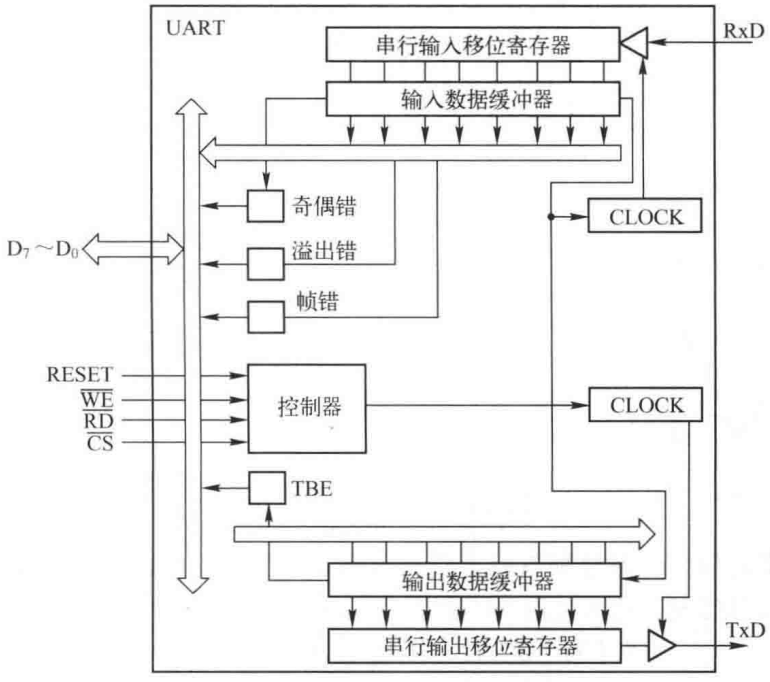


图 8-11　硬件 UART 的内部结构

当输入时，由接收端 RxD 接收到的串行数据先进入移位寄存器，然后并行输入缓冲器，变为并行数据，由数据总线送至 CPU。在发送时，由 CPU 来的并行数据被缓冲器接收，然后送到移位寄存器 TxD 输出端一位接一位地输出，最低有效位先输出。

为了检测长距离传输中可能发生的错误，通常增加一个奇偶校验位和各种出错标志位。比如 UART 在发送时，检查每个要传输的字符中的“1”的个数，自动在奇偶校验位上添“1”或“0”，使得“1”的个数满足校验要求。在接收时，UART 检查每个字符的各位及奇偶校验位，若“1”的个数不满足校验要求，则表明传输过程中可能发生了错误。

（2）错误检测

为了传输过程更加可靠，在 UART 中还设立了各种出错标志，常用的有以下 3 种。

① 奇偶错误。在接收时，UART 检查接收到的每个字符的“1”的个数的标志，发出奇偶校验出错信息。

② 帧错误。若接收到的字符格式不符合规定（如缺少停止位等），则置位该标志，发出帧出错信息。

③ 丢失（溢出）错误。UART 是一种双缓冲器结构。例如，在接收时，接收的数据先由移位寄存器移位，把串行数据变成并行数据，然后送到接收数据寄存器，由输入指令将数据送到 CPU 中。由于数据送到接收数据寄存器，所以即使 CPU 还没取走这个数据，UART 也可以接收另一个新的字符。倘若 UABT 接收到第二个字符的停止位，且要把第二个字符传输到接收数据寄存器时，CPU 还没取走上一个数据，于是第一个数据会被丢失。如果 UART 出现这种情况，就置位丢失（溢出）标志，发出丢失出错信息。

4．信号的调制和解调

计算机的通信是由一种由 0 和 1 组成的数字信号的通信，要求传输线的频带很宽，而在长距离通信时，通常是用电话线传输的，不可能有这样宽的频带。如果使用数字信号直接通信，经过传输线，信号会发生畸变。由于模拟信号的传输比数字信号传输更为有效，因而可将数字信号调制成模拟信号进行传输，用解调器把接收的模拟信号再转换成数字信号。

FSK（Frequency Shift Keying）是一种常用的调制方法，它把数字信号的 1 和 0 调制成不同频率的模拟信号，其原理如图 8-12 所示。

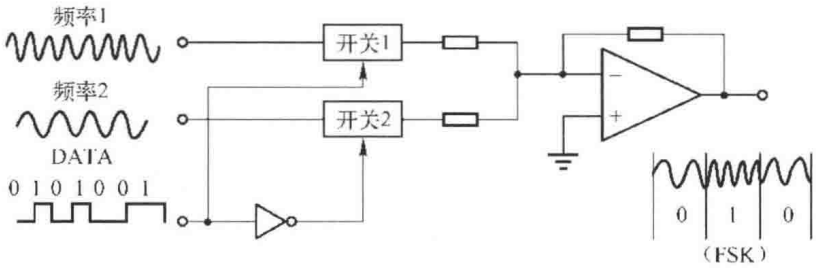


图 8-12　FSK 调制法原理

两个不同频率的模拟信号分别由电子开关控制，在运算放大器的输入端相加，而电子开关由要传输的数字信号控制。当信号为“1”时，控制 1 号电子开关导通，送出一串频率较高（如 4800 Hz）的模拟信号；当信号为“0”时，控制 2 号电子开关导通，送出一串频率较低（如 2400 Hz）的模拟信号，于是在运算放大器的输出端，就得到了调制后的信号。

8.1.3 可编程串行通信接口芯片 8251A 简介

8251A 芯片是 Intel 公司生产的大规模集成电路芯片，是与 Intel 系列 CPU 兼容的可编程的串行通信接口。虽然 8251A 功能较强，但它需要外部时钟电路。当取标准的波特率如 1200、2400 等时，外部时钟的选取一般不是任意的，这往往需要专门的时钟产生电路，如采用 8253 可编程定时器/计数器。可见，为了构造一个串行通信系统，采用 8251A 作为接口电路时需要比较复杂的外围电路。目前流行的单片机，如 MCS51 系列，CPU 内部就集成了串行接口部件及定时器/计数器，几乎不需要外围辅助电路，使用起来非常简单，性价比很高。因此，越来越多的数字化仪器仪表电路中不再采用 8251A，而是使用单片机作为串行通信接口。

1．8251A 的基本性能与内部结构

其基本性能可用于同步和异步传输。

- 同步传输：5～8 位/字符，内部或外部同步，可自动插入同步字符。
- 异步传输：5～8 位/字符，波特率因子可取 1，16 或 64。
- 可产生中止字符：可产生 1 位、1 位半或 2 位的停止位，可检测假启动位，自动检测和处理终止字符。
- 波特率：DC（异步），19.2 kbps；DC（同步），64 kbps。
- 完全双工：双缓冲发送和接收。
- 出错检测：具有奇偶、溢出和帧错误等检测电路。

8251A 的内部结构如图 8-13 所示，由发送器、接收器、数据总线缓冲器、读/写控制电路和调制解调控制电路 5 部分组成。

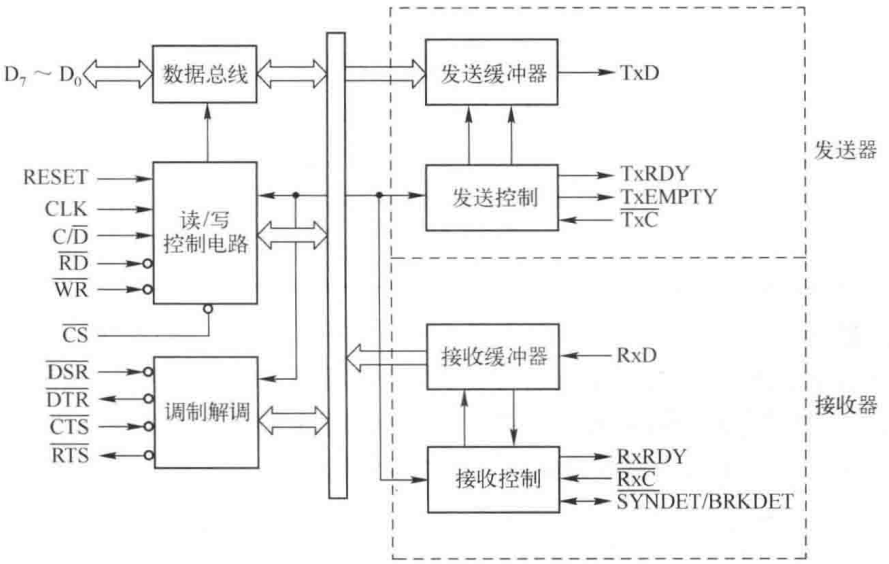


图 8-13　8251 内部结构

（1）数据总线缓冲器

数据总线缓冲器是 CPU 与 8251A 之间的数据接口，包含 3 个 8 位缓冲寄存器。其中 2 个用来存放 CPU 从 8251A 读取的状态信息或数据，另 1 个存放 CPU 向 8251A 写入的控制字或数据。

（2）发送器

发送器由发送缓冲器和发送控制电路两部分组成。需要发送的数据经数据发送缓冲器并行锁入发送缓冲器中。如果采用异步方式，则由发送控制电路在其首尾加上起始位和停止位，然后从起始位开始，经移位寄存器从数据输出线 TxD 逐位串行输出，其发送速率取决于 TxC 端上收到的发送时钟频率。如果采用同步方式，则在发送数据之前，发送器将自动送出 1 个（单同步）或 2 个（双同步）同步字符，然后才逐位串行输出数据。当发送器做好发送数据准备时，由发送控制电路向 CPU 发出 TxRDY 有效信号，CPU 可立即向 8251A 并行输出数据。如果 8251A 与 CPU 之间采用中断方式交换信息，则 TxRDY 信号可作为向 CPU 发出的中断请求信号。待发送器中的 8 位数据串行发送完毕，由发送控制电路向 CPU 发出 TxEMPY 有效信号，表示发送器中移位寄存器已空。因此，发送数据缓冲器将发送移位寄存器构成发送器的双缓冲结构。

（3）接收器

接收器由接收缓冲器和控制电路组成。从外部通过数据接收端 RxD 接收的串行数据逐位进入接收移位寄存器中。如果是异步方式，则应识别并删除起始位和停止位；如果是同步方式，则要检测到同步字符，确认达到同步，接收器才可以开始接收串行数据，待一组数据接收完毕，可将移位寄存器中的数据并行置入接收数据缓冲器中，同时输出 RxRDY 有效信号，表示接收器已准备好数据等待向 CPU 传输。其接收数据的速率取决于从 $\overline{\text{RxC}}$ 端输入的接收时钟频率。

（4）读/写控制电路

读/写控制电路对 CPU 输出的控制信号进行译码以实现对 8251A 的读/写功能。8251A 的读/写操作控制如表 8-1 所示。

表 8-1　8251A 读/写功能表

$\overline{\text{CS}}$	C/$\overline{\text{D}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	功　能
0	0	0	1	CPU 从 8251A 读数据
0	1	0	1	CPU 从 8251A 读状态
0	0	1	0	CPU 从 8251A 写数据
0	1	1	0	CPU 从 8251A 写命令
1	×	×	×	无操作

（5）调制解调控制电路

这部分电路是 8251A 将数据输出端的数字信号转换成模拟信号，或将数据接收端的模拟信号解调成数字信号的接口电路。8251A 要与调制解调器相连，它提供的接口信号一部分为与 CPU 接口的信号，另一部分为与外设或调制器的接口信号。

2．8251A 的引脚功能

8251A 是 28 个引脚的双列直插式大规模集成电路芯片，根据其内部结构，引脚也分为 5 部分介绍。其引脚封装如图 8-14 所示。

8251A 数据总线 D_7～D_0 是三态双向的与 CPU 相连的数据总线。CPU 通过数据总线并行传送命令，交换数据，检测状态。控制信号引脚如下。

① CLK：输入时钟，产生 8251A 的内部时序。CLK 的频率在同步方式工作时，必须大于接收器和发送器输入时钟频率的 30 倍；在异步方式工作时，必须大于输入时钟的 4.5 倍。

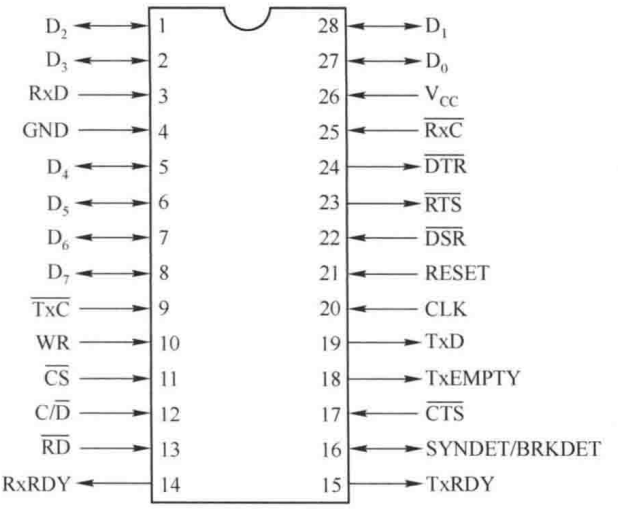


图 8-14　8251A 引脚封装

② M/$\overline{IO}$：片选信号，一般由 CPU 的高位地址信号和 M/$\overline{IO}$ 控制信号译码后供给，产生 8251A 端口地址的高地址部分。

③ C/$\overline{D}$：控制/数据端，此引脚为高电平，则数据总线上的信息是状态信息或命令信息；此引脚为低电平，则数据总线上的信息是数据信息。它一般由 CPU 低位地址线供给，产生 8251A 的端口地址的低地址部分。

④ RESET：复位信号，高电平有效。RESET 信号有效时，8251A 的收发线路均处于空闲状态，等待 CPU 对其的初始化编程。

⑤ $\overline{WR}$，$\overline{RD}$：读、写控制信号，低电平有效，其功能见表 8-1。

发送器部分引脚包括：

① TxD：数据发送线，8251A 将并行数据变换成串行格式后逐位由此线输出。

② TxRDY：发送器准备好信号。此信号有效，则 CPU 可向 8251A 写入待发送的数据，8251A 将 CPU 来的并行数据锁入数据发送缓冲器中。

③ TxEMPTY：发送器空信号。此信号有效，表示发送移位寄存器已空，此时发送缓冲器的数据可送入发送移位寄存中逐位输出。

④ $\overline{TxC}$：发送器时钟，由外部输入。$\overline{TxC}$ 确定 8251A 的发送速率。对于同步方式，$\overline{TxC}$ 端输入的时钟频率应等于发送数据波特率。对于异步方式，可由软件定义发送时钟是发送波持率的 1 倍、16 倍或 64 倍，即发送波特率因子的倍数。

TxRDY 和 TxEMPY 的区别在于，TxRDY 有效表示发送数据缓冲器已空，而 TxEMPY 有效表示发送移位寄存器已空。

接收器部分引脚包括：

① RxD：数据接收线。外部串行数据通过该引脚逐位移入接收移位寄存器中，变换为并行格式后便送入接收数据缓冲器，等待 CPU 取数。

② RxRDY：接收器准备好信号，高电平有效。接收缓冲器中收到一个数据字符，则 RxRDY 信号有效，通知 CPU 取数，若 8251A 采用中断方式与 CPU 交换数据，则 RxRDY 信号可用做向 CPU 发出的中断请求信号。CPU 取走接收缓冲器中的数据，RxRDY 变为低电平。

③ SYNDET/BRKDKT：双功能引脚，高电平有效。

④ $\overline{\text{RxC}}$：接收器时钟，由外部输入。$\overline{\text{TxC}}$ 确定 8251A 的接收数据速率。采用同步方式，从 $\overline{\text{RxC}}$ 端输入的时钟频率应等于接收数据的波特率；若采用异步方式，可由软件定义接收器时钟为接收数据波特率的 1、16 或 64 倍，即发送波特率因子的倍数。

对于异步方式，SYNDET/BRKDET 功能为断缺检测端 BRKDET。若在起始位之后，从 RxD 端上连续收到 8 个“0”信号，则输出端 BRKDET 为高电平，表示当前处于数据断缺状态，无数据可接收。若从 RxD 端上接收到“1”信号，BRKDET 的高电平变低。

对于同步方式，SYNDET/BRKDKT 功能为同步检测端 SYNDET。如果采用内同步，则 SYNDET 为输出端，高电平有效。当从 RxD 端上检测到一个（单同步）或两个（双同步）同步字符时，SYNDET 输出高电平有效信号，表示接收数据已处于同步状态，后面接收到的是有效数据。如果采用外同步，则 SYNDET 为输入端，外同步字符从该端输入。SYNDET 为高电平输入有效信号，表示已达到同步，接收器可开始串行接收数据。

调制解调接口控制引脚如下。

① $\overline{\text{DTR}}$：数据终端准备好信号，向调制解调器输出的低电平有效信号，CPU 准备好接收数据，使 $\overline{\text{DTR}}$ 有效，可由控制字中的 DTR 位置“1”输出该有效信号。

② $\overline{\text{DSR}}$：数据装置准备好信号，调制解调器输入的低电平有效信号。调制解调器已做好发送数据准备，就发出 $\overline{\text{DSR}}$ 信号，CPU 可用 IN 指令读入 8251A 的状态寄存器，检测 DSR 位，$\overline{\text{DSR}}$ 位为“1”时，表示 $\overline{\text{DSR}}$ 信号有效。该信号实际上是对 $\overline{\text{DTR}}$ 信号的回答，通常用于接收数据。

③ $\overline{\text{RTS}}$：请求发送信号，向调制解调器输出的低电平有效信号，CPU 准备好发送数据，由软件定义，使控制字中的 RTS 位置“1”，则 $\overline{\text{RTS}}$ 输出低电平有效信号。

④ $\overline{\text{CTS}}$：准许发送信号，由调制解调器输入的低电平有效信号，是对 $\overline{\text{RTS}}$ 的回答信号，将控制字中的 TxEN 位置“1”，则 $\overline{\text{CTS}}$ 为低电平有效，发送器可串行发送数据。若在数据发送过程中使 $\overline{\text{CTS}}$ 无效，或控制字中的 TxEN 位为“0”，发送器将正在发送的字符结束后停止继续发送。

3．8251A 的编程

8251A 是一个可编程的多功能串行通信接口芯片，在使用前必须对它进行初始化，以确定它的工作方式、传输速率、字符格式以及停止位长度等。8251A 有 3 个控制字，分别为方式选择字、操作命令字和状态字。

（1）方式选择控制字

其格式如图 8-15 所示。控制字中，D_1D_0 有 4 种组合，D_1D_0=00，则 8251A 选择为同步方式工作；否则，8251A 选择为异步方式工作。异步方式下的输入时钟和波特率之间的系数可由 D_1D_0 的其他 3 种组合规定。D_3D_2 用来确定字符的长度。D_5D_4 可确定是否要奇偶校验，是奇校验还是偶校验。D_7D_6 的定义分两种情况：同步方式工作时，表示选用的是内同步还是外同步以及同步字符的个数；异步方式工作时，表示停止位的长度。

（2）操作命令控制字

操作命令字直接使 8251A 处于规定的工作状态，其格式如图 8-16 所示。

操作命令控制字中，每位都有自己的定义。

TxEN 位置 1，发送器才能通过 TxD 引脚向外部串行发送数据。

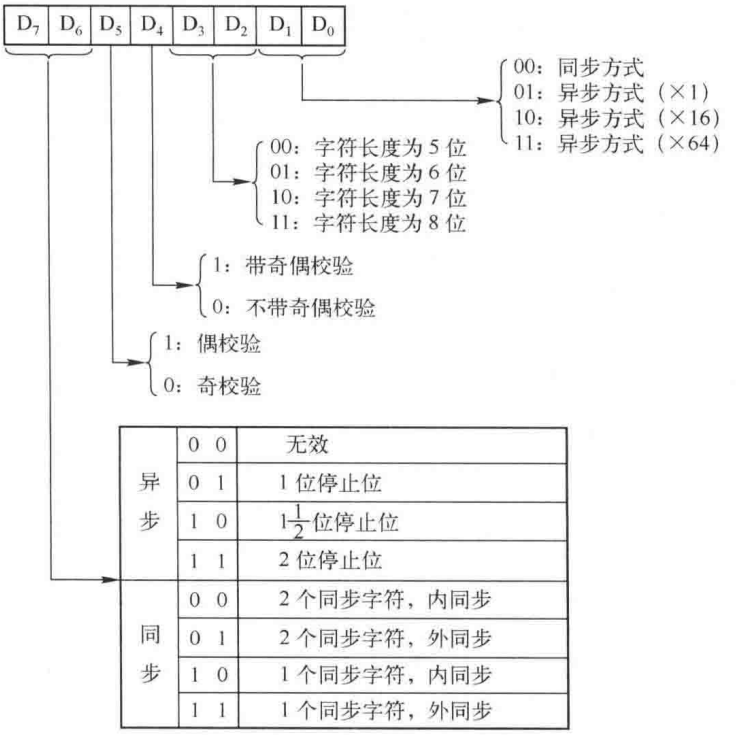


图 8-15　8251A 方式选择控制字格式

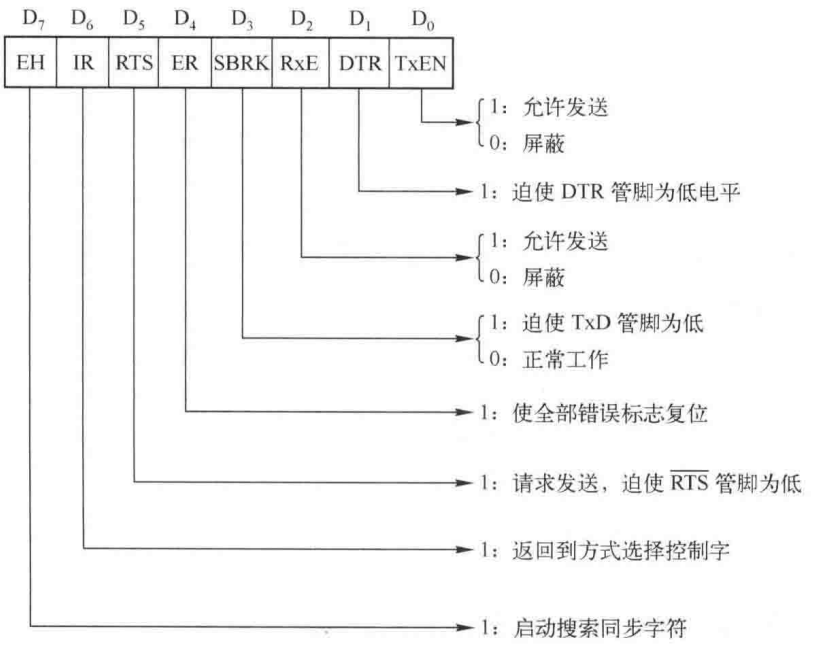


图 8-16　8251A 操作命令控制字格式

DTR 位置 1，表示 CPU 已准备好接收数据，这时 8251A 的 $\overline{\text{DTR}}$ 引脚向调制解调器输出低电平有效信号。

RxE 位置 1，接收器可通过 RxD 引脚接收外部串行数据。

SBRK 位是发送断缺字符，SBRK 位为 1，迫使 TxD 引脚处于低电平，发送“0”信号。正常通信过程中，SBRK 应保持为 0。

ER 位为 1，则清除奇偶出错标志（PE）、溢出错误标志（OE）和帧校验出错标志（FE）。

RTS 位是请求发送信号，该位置 1，迫使 $\overline{\text{RTS}}$ 引脚输出低电平，表示 CPU 已做好发送数据准备。

IR 位是内部复位信号，该位置 1，迫使 8251A 回到方式选择控制字状态。这样用户有两种方法复位 8151A：一是硬件复位，即通过引脚 RESET 为高电平使 8251A 进入复位状态；二是软件复位，即通过 IR 位置 1，使 8251A 进入方式重新选择的状态。

EH 位只对同步方式有效，该位为 1，表示开始搜索同步字符。因此，对于同步方式，一旦允许接收（RxE=1），必须同时使 EH=1 和 ER=1，清除全部错误标志后，才能开始搜索同步字符。

（3）状态控制字

在 8251A 工作过程中，CPU 可用 IN 指令读取当前 8251A 的状态控制字。状态控制字的格式如图 8-17 所示。

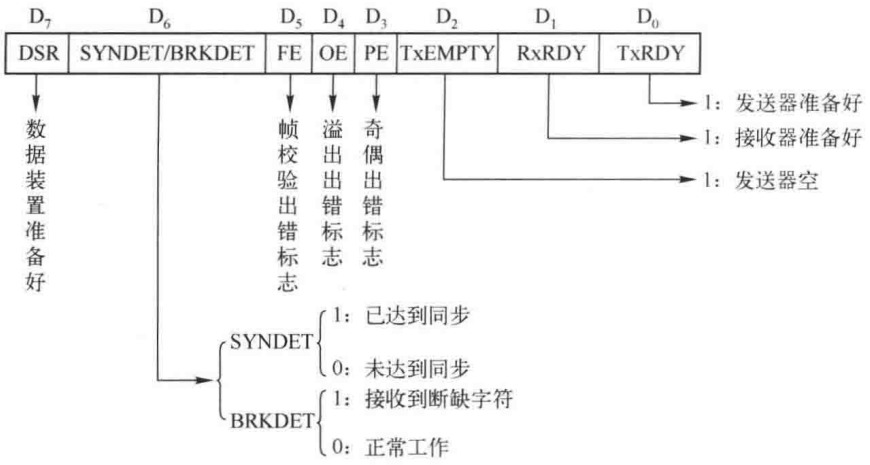


图 8-17　8251A 状态控制字格式

TxRDY 位是发送准备好标志，反映当前发送数据缓冲器已空。也就是说，一旦发送缓冲器空该位就置 1，它仅表示一种 8251A 的此时的工作状态。TxRDY 引脚要为高电平还必须有其他两个条件，即要对 8251A 发操作命令，使其允许发送，TxEN=1；同时，8251A 要从调制解调器输入一低电平使 $\overline{\text{CTS}}$ 引脚为低电平有效。在数据发送过程中，TxRDY 状态和 TxRDY 引脚信号总是相同的，这就可以由 TxRDY 状态供 CPU 查询，而 TxRDY 引脚信号向 CPU 发出中断请求信号。

RxRDY、TxEMPTY、SYNDET/BRKDET 这 3 个位状态的定义与其相应的引脚定义相同可供 CPU 查询。

DSR 状态位为 1，表示外设或调制解调器已做好发送数据准备，并发出低电平信号使 8251A 的 DSR 引脚为低电平有效信号。

PE 是奇偶错标志位。PE=l 表示当前发生了奇偶错误，但 8251A 还是照样工作。

OE 是溢出错标志位。当下一个字符从 RxD 端输入而 CPU 还没来得及读取上一个字符时，上一个字符将被丢失。它不终止 8251A 的工作，但使该位置 1。

FE 是帧校验错标志。FE 只对异步方式有效，当在任意字符的结尾没有检测到规定的停止位时，这个标志置 1。它不禁止 8251A 的工作。

PE、OE、FE 这 3 个标志可由操作命令控制字中的 ER 位为 1 来全部复位。

CPU 可在任意时刻用 IN 指令读 8251A 的状态位，这时 C/$\overline{D}$ 引脚端输入为“1”（决定于 8251A 的端口地址）。在 CPU 读状态期间，8251A 将自动禁止改变状态位。

对 8251A 进行初始化编程，必须在系统复位之后（因为，RESET 会使收发引脚处于空闲状态，各寄存器均处于复位状态）。编程过程如下：先使用方式选择控制字，如果定义 8251A 工作在异步方式下，那么必须紧跟操作命令字进行定义，然后才可以开始传输数据。在数据传输过程中，可使用操作命令字重新定义，或使用状态控制字读入 8251A 的状态。如果要设定新的工作方式，必须用操作命令控制字将 IR 位置 1，使其返回到方式选择控制字，接收新的方式选择命令，从而改变工作方式。

如果采用同步工作方式，在方式选择控制字之后输出同步字符，在一个或两个同步字符之后再使用操作命令控制字，以后的过程同异步方式。

8251A 初始化编程的操作过程可用图 8-18 来描述。

4．8251A 应用举例

为了让读者进一步了解和掌握 8251A 的使用，这里给出一个最基本的例子。

【例 8-2】 设 8086 CPU 通过 8251A 串行接口与外部设备进行双向通信。要求 8251A 工作于异步方式，传输速率为 1200 bps，波特率因子为 16，字符格式为 8 位数据位，1 位停止位，无偶校验。8251A 的端口如图 8-19 所示，试编写 8251A 的初始化程序及接收和发送程序。

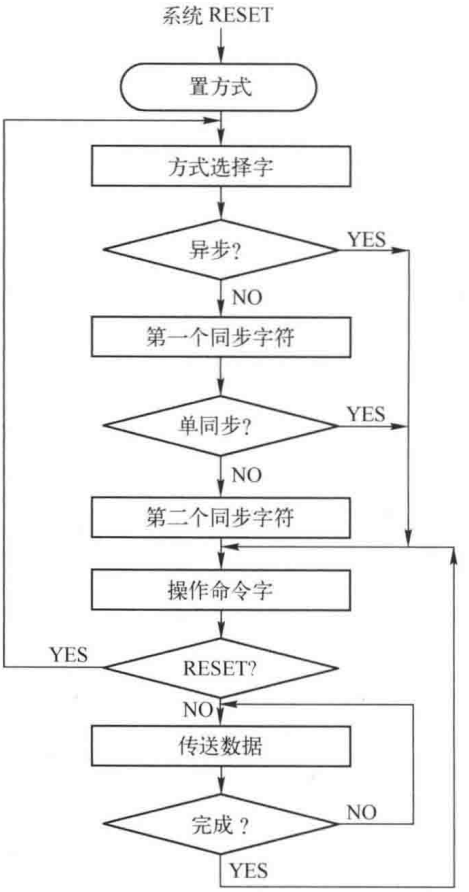


图 8-18　8251A 初始化编程的流程

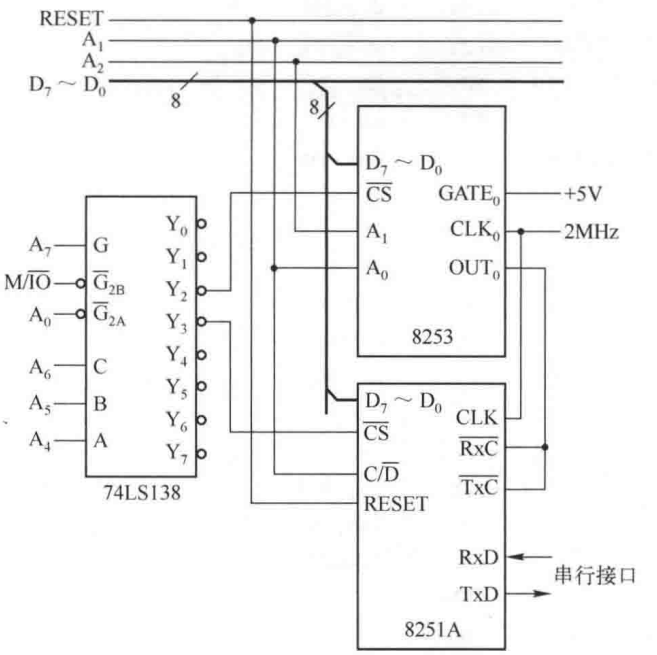


图 8-19　8251A 的端口

从图 8-19 可知，接口电路由两个接口部件组成，即计数器/定时器 8253 和 8251A。8253

的作用是通过通道 0 向 8251A 提供发送和接收时钟。根据译码电路，它们的端口地址为偶地址，因而 8251A 和 8253 的数据线应与 8086 CPU 的低 8 位数据线 D_7～D_0 相连接，具体端口地址如下：8253 分布在 90H～96H，8251A 分布在 98H～9AH。

*首先讨论 8252 通道 0 的工作方式。由于通道 0 的输出为时钟信号，所以通道 0 应工作在*方式 3 下，其功能是对 2 MHz 信号进行分频。根据已知条件，OUT0 输出的方波信号频率为

$$f_{\mathrm{OUT_0}} = 1200 \times 16 = 19.2(\mathrm{kHz})$$

所以，计数常数为

$$N = \frac{f_{\mathrm{CLK_0}}}{f_{\mathrm{OUT_0}}} = \frac{2000}{19.2} \approx 104$$

下面讨论 8251A 的编程操作。8251A 的方式选择控制字为 01001110B（4EH），操作命令控制字用了两个：一是使 8251A 复位，即控制字为 0001000B（40H）；二是启动 8251A 接收和发送，其控制字为 00010101B（45H）。

虽然在 8251A 上电时，RESET 信号可以使 8251A 进入复位状态，但在实际使用时，为了使 8251A 可靠复位，我们常在初始化程序前用软件再次让 8251A 进行复位操作。具体实现过程是先向 8251A 的控制口连续写入 3 个 0，再写入复位控制字 40H。由于 8251A 内部操作需要一定的时间，所以写入操作后还要有一定的延迟。

初始化程序为：

```
INIT_PROC     PROC    FAR
              MOV     AL, 00110110B          ; 写入 8253 控制字
              OUT     96H, AL
              MOV     AL, 0                  ; 写入计数常数
              OUT     90H, AL
              MOV     AL, 104
              OUT     90H, AL
              MOV     CX, 3                  ; 以下为 8251A 复位操作
              MOV     AL, 0
INIT_LOOP:    OUT     9AH, AL                ; 连续送 3 个 0
              CALL    DELAY
              LOOP    INIT_LOOP
              MOV     AL, 40H                ; 送复位控制字
              OUT     9AH, AL
              CALL    DELAY
              MOV     AL, 4EH                ; 送方式控制字
              OUT     9AH, AL
              CALL    DELAY
              MOV     AL, 45H                ; 启动接收和发送
              OUT     9AH, AL
              CALL    DELAY
              RET
INIT_PROC     ENDP
```

设发送数据在寄存器 DL 中，发送程序如下：

```
SEND_CHAR     PROC    FAR
```

```
SEND_CHECK:   IN      AL, 9AH                ; 输入状态信息
              TEST    AL, 01H                ; 检测状态位 TxRDY
              JZ      SEND_CHECK             ; TxRDY 无效，循环检测
              MOV     AL, DL                 ; TxRDY 有效，发送数据
              OUT     98H, AL
              RET
SEND_CHAR     ENDP
```

接收程序和发送程序相似，先检测状态位，然后对 8251A 的数据口进行操作。设接收数据在寄存器 AL 中，接收程序如下：

```
REV_CHAR      PRO     FAR
REV_CHECK:    IN      AL, 9AH                ; 输入状态信息
              TEST    AL, 02H                ; 检测状态位 RxRDY
              JZ      REV_CHECK              ; RxRDY 无效，循环检测
              IN      AL, 98H                ; RxRDY 有效，接收数据
              RET
REV_CHAR      ENDP
```

8.1.4 串行通信接口 RS-232C

串行通信接口标准都是在 RS-232 标准的基础上经过改进而形成的，所以本节以 RS-232C 为主来讨论。RS-232C 标准是美国 EIA（电子工业联合会）与 Bell 等公司一起开发的 1969 年公布的通信协议，全称是 EIA-RS-232C 标准，其中 EIA（Electronic Industry Association）代表美国电子工业协会，RS（Recommended Standard）代表推荐标准，232 是标识号，C 代表 RS-232 的最新一次修改（1969 年）。RS-232C 适合数据传输速率在 0～20000 bps 范围内的通信。这个标准对串行通信接口的有关问题，如信号线功能、电器特性都做了明确规定。由于通用设备厂商都生产与 RS-232C 制式兼容的通信设备，因此它作为一种标准，已在计算机通信接口中广泛采用。目前，计算机上的 COM1、COM2 接口就是 RS-232C 接口。

1．RS-232C 电器特性及接口信号

（1）电气特性

目前，较为常用的 RS-232C 串口有 9 针串口（DB-9）和 25 针串口（DB-25），见图 8-1。它对电器特性、逻辑电平和各种信号线功能都做了规定。

- ❖ 在数据线 TxD 和 RxD 上：逻辑 1，–3～–15 V；逻辑 0，+3～+15 V。
- ❖ 在控制线和状态线 RTS、CTS、DSR、DTR 和 DCD 上：信号有效，+3～+15 V；信号无效，–3～–15 V。

以上规定说明了 RS-323C 标准对逻辑电平的定义。由于逻辑电平在–3～–15 V 及+3～+15 V 范围内，所以只有当传输电平的绝对值大于 3 V 时，电路才能有效地检查出来，而介于–3～+3 V 之间的电压无意义。同样，低于–15 V 或高于+15 V 的电压也被认为无意义。因此，实际工作时，应保证电平在±(3～15 V)之间。

（2）RS-232C 与 TTL 转换

RS-232C 用正负电压来表示逻辑状态，与 TTL 以高低电平表示逻辑状态的规定不同。因此，为了能够使计算机接口与终端的 TTL 器件连接，必须在 RS-232C 与 TTL 电路之间进行电

平和逻辑关系变换。实现这种变换可用分立元件，也可用集成电路芯片。采用 MC1488 和 MC1489 芯片的转换接口为早期的 RS-232C 至 TTL 逻辑电平的转换电路，图 8-20(a)为实际电路。该电路的不便之处是需要±12 V 电压，并且功耗较大，不适用于低功耗的系统。其中，TxD、RxD 分别接串行接口（TTL 电平）的发送和接收端。

图 8-20(b)采用 MAX232 芯片的转换接口。MAX232 是 MAXIM 公司生产的，包含两路驱动器和接收器。芯片内部有一个电压转换器，可以把输入的+5 V 电压转换为 RS-232C 接口所需的±10 V 电压，尤其适用于没有±12 V 的单电源系统，并具有功耗低、价格低廉、外围电路简单等优点，受到广大用户的青睐。图 8-20(c)为采用分立元件实现的 RS-232 - TTL 电平的转换接口电路，其特点是利用 PC 的 RS-232C 接口的脚信号出来供给负电源。其逻辑“1”电平时电压为–10 V 左右，其驱动能力为 20 mA。利用这个特性，用一个二极管和电解电容，即在电解电容上获取 RS-232C 通信所需的负电源。该电路简单、功耗小，在没有专用芯片时不失为一种替代方法。

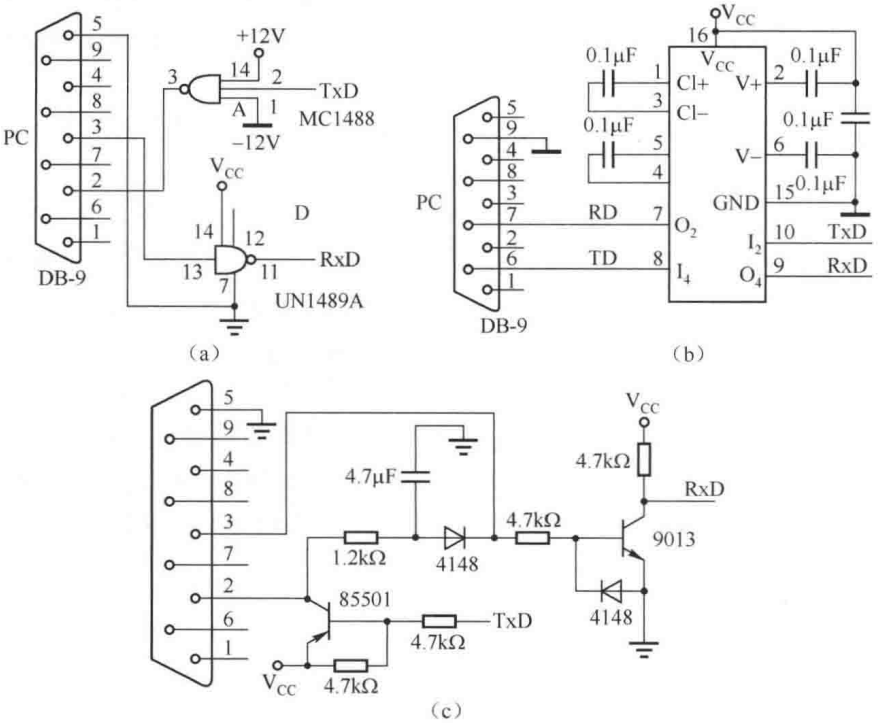


图 8-20　RS-232C 电平转换以及与计算机的接口电路

（3）RS-232C 的接口信号

可以用电缆线直接连接标准 RS-232 端口，但通信距离较近（小于 12 m）。若距离较远，则可附加调制解调器（Modem）。RS-232C 标准接口有 25 条线，包括 4 条数据线、11 条控制线、3 条定时线、7 条备用和未定义线。表 8-2 为 RS-232C 最基本、最常用的信号线。

① 联络控制信号线

数据设备准备好（Data Set Ready，DSR）：有效时，表明数字通信设备（Data Communication Equipment，DCE）处于可以使用的状态。

数据终端准备好（Data Set Ready，DTR）：有效时，表明数据终端设备（Data Terminal Equipment，DTE）可以使用。

表 8-2　RS-232C 信号线

9 针串口（DB-9）			25 针串口（DB-25）		
针号	功 能 说 明	缩写	针号	功 能 说 明	缩写
1	数据载波检测	DCD	2	发送数据	TxD
2	接收数据	RxD	3	接收数据	RxD
3	发送数据	TxD	4	请求发送	RTS
4	数据终端准备好	DTR	5	清除发送	CTS
5	信号地	GND	6	数据终端准备好	DSR
6	数据设备准备好	DSR	7	信号地	GND
7	请求发送	RTS	8	数据载波检测	DCD
8	清除发送	CTS	20	数据设备准备好	DTR
9	振铃指示	RI	22	振铃指示	RI

有时这两个信号直接连到电源上，一上电就立即有效。这两个设备状态信号有效，只表示设备本身可用，并不说明通信链路可以开始进行通信，能否开始进行通信要由下面的控制信号决定。

- ❖ 请求发送（Request To Send，RTS）：表示 DTE 请求发数据给 DCE。即当终端要发送数据时，使该信号有效。
- ❖ 清除发送（Clear To Send，CTS）：表示 DCE 准备好接收 DTE 发来的数据，是对请求发送信号 RTS 的响应信号。

RTS/CTS 请求应答联络信号用于半双工串行通信系统中发送方式和接收方式之间的切换。在全双工系统中，因配置双向通道，故不需要 RTS/CTS 联络信号，使其变高。

- ❖ 数据载波检出（Data Carrier Detection，DCD）：表示 DCE 已接通通信链路，告知 DTE 准备接收数据。当本地的 Modem 收到由通信链路另一端（远地）的 Modem 送来的载波信号时，使 DCD 信号有效，通知终端准备接收，并且由 Modem 将接收下来的载波信号解调成数字数据后，沿接收数据线 RxD 送到终端。
- ❖ 振铃指示（Ringing，RI）：当 Modem 收到交换台送来的振铃呼叫信号时，使该信号有效，通知终端，已被呼叫。

② 数据发送与接收线

- ❖ 发送数据（Transmitted Data，TxD）：通过 TxD 线，终端将串行数据发送到 Modem（DTE→DCE）。
- ❖ 接收数据（Received Data，RxD）：通过 RxD 线，终端接收从 Modem 发来的串行数据（DCE→DTE）。

③ 地线

信号地（Signal Ground，SG）：无方向，在一些参考书上用符号 GND 表示。

上述控制信号线主要是对半双工通信。控制信号线上的信息何时有效，何时无效的顺序表示了接口信号的传输过程。例如，只有当 DSR 和 DTR 都处于有效状态时，才能在 DTE 和 DCE 之间进行传输操作。若 DTE 要发送数据，则预先将 DTR 线置成有效状态，等 CTS 线上收到有效状态的回答后，才能在 TxD 线上发送串行数据。这种规定对半双工的通信线路是有用的，因为半双工的通信才能确定 DCE 已由接收方向改为发送方向，这时线路才能开始发送。

2．RS-232C 应用举例

利用计算机的 RS-232C 接口，可以方便地与另一台计算机或单片机系统，如 MCS51 系统之间相互传输数据。

（1）RS-232C 串口通信接线方法（三线制）

虽然标准串口的信号线很多，但由于 RS-232C 是全双工通信，在实际应用时，如 PC 与其他系统相连，采用三线制就可以了。三线制就是指发送数据线 TxD、接收数据线 RxD 和信号地线 GND。连接时，双方的地线直接相连，收发数据线交叉相连。图 8-3 是一个计算机与一个外设或两个计算机通过 RS-232C 相连接。图 8-21 为计算机与 MCS51 单片机通信，由于单片机是 TTL 电平，故需要电平转换电路。

（2）BIOS 串行通信口功能

这里只讨论计算机方的数据传输问题，至于另一方的数据传输，如果系统不是计算机而是单片机等，读者可查阅有关参考书。

IBM PC 及其兼容机提供比较灵活的关于串行口的 BIOS 中断调用方法，即通过“INT 14H”调用 ROM BIOS 串行通信口例行程序。该例行程序包括将串行口初始化为指定的字节结构和传输速率，检查控制器的状态、读写字符等功能。下面介绍“INT　14H”中断调用功能。

① 初始化串行通信口（AH=0）

调用参数：AL=初始化参数

　　DX=通信口号，0：COM1，1：COM2

返回参数：AH=通信口状态

　　AL=调制解调器状态

初始化参数字格式如图 8-22 所示。比如，要求 COM1 口的传输率为 4800 bps，字长为 8 位，1 位停止位，无奇偶校验。程序段如下：

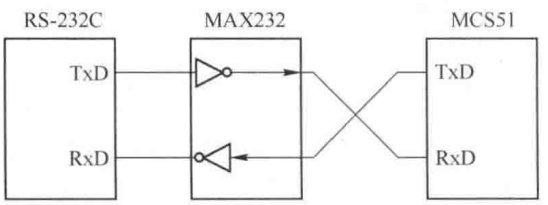


图 8-21　计算机与 MCS51 单片机通信

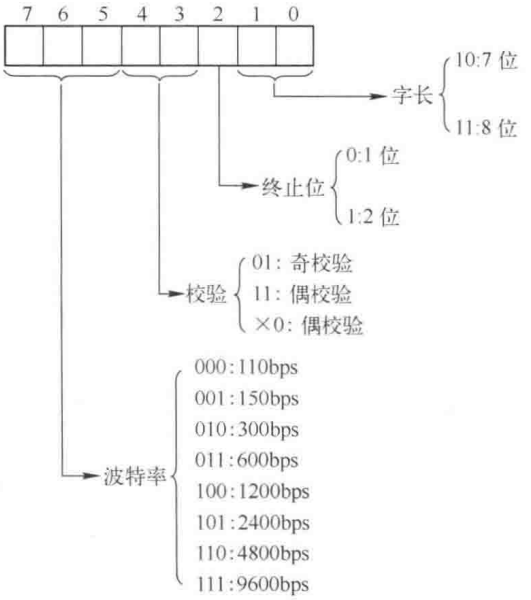


图 8-22　初始化参数字格式

```
MOV     AH, 0
MOV     AL, 11000011B
```

```
MOV     DX, 0
INT     14H
```

通信口状态字格式如图 8-23 所示。在接收和发送过程中，错误状态位（1、2、3、4 位）一旦被置为 1，则读入的接收数据已不是有效数据，所以在串行通信应用程序中，应检测数据传输是否出错。

- ❖ 奇偶错：通信线上（尤其是用电话线传输时）的噪声引起某些数据位的改变，产生奇偶错。通常检测出奇偶错时，要求被接收的数据至少应重新发送一次。
- ❖ 超越错：上一个字符还未被处理机取走，又有字符要传送到数据寄存器时，则会引起超越错。如果处理机处理字符的速度小于串行通信口的波特率，则会产生这种错误。
- ❖ 帧格式错：接收/发送器未接收到一个字符数据的停止位，则会引起帧格式错。这种错误可能出于通信线上的噪声引起停止位的丢失，或者由于接收方和发送初始化不匹配。
- ❖ 间断：间断有时候并不能算一个错误，而是为某些特殊的通信环境设置的“空格”状态。当间断位为 1 时，说明接收的“空格”状态超过了一个完整的数据字传输时间。

调制解调器状态字格式如图 8-24 所示。

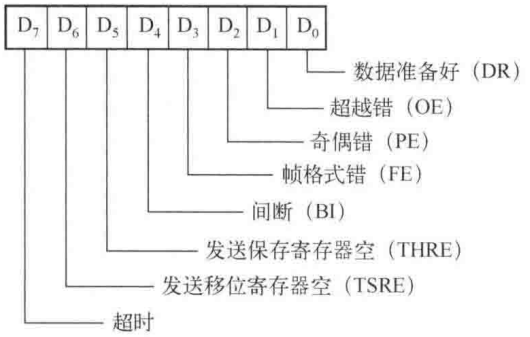


图 8-23　串行通信口状态字格式

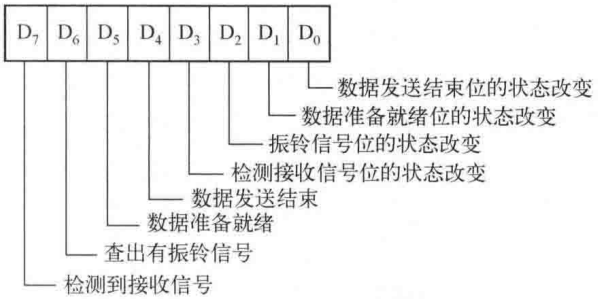


图 8-24　调制解调器状态字格式

② 向串行通信口写字符（AH=1）

输入参数：　AL=所写字符

　　DX=通信口号　　0：COM1，1：COM2

输出参数：　写字符成功：AH.7=0，AL=已写入的字符

　　写字符失败：AH.7=1，AH.0～6=通信口状态

③ 从串行通信口读字符（AH=2）

输入参数：　DX=通信口号　　0：COM1，1：COM2

输出参数：　读字符成功：AH.7=0，AL=字符

　　读字符失败：AH.7=1，AH.0～6=通信口状态

④ 取通信口状态

输入参数：　DX=通信口号　　0：COM1，1：COM2

输出参数：　AH=通信口状态，AL=调制解调器状态

（3）软件编程

设通信双方有一台计算机为 PC，使用 COM1 端口。在串行通信中，必须首先设定通信双方所使用字符串的数据结构，才能进行软件编程。这里设数据在接收和发送的字符串中，序号为 0 的字节为数据长度，其后的字节为所接收的数据。

另外，设 COM1 口的传输率为 2400 bps，字长为 8 位，1 位停止位，无奇偶校验。

对于接收过程，程序首先用“INT 14H, AH=3”来获得 COM1 端口的状态，如果检测到“数据准备好”位有效，表明 COM1 口接收到一个数据，则用“INT 14H, AH=2”功能，将字符读到 AL 寄存器。程序分主程序和子程序两部分。

主程序如下：

```
BUFFER      DB      100DUP(?)                ; 定义字符串缓冲单元
            ……
            MOV     AH, 0                    ; 设置 COM1 口
            MOV     AL, 0A3H
            MOV     DX, 0
            INT     14H
            ……
            CALL    RECEIVE                  ; 接收第 0 号数据
            TEST    AH, 80H                  ; 测试读是否成功
            JNZ     REC_ERROR                ; 不成功，转出错处理
            MOV     CH, 0
            MOV     CL, AL                   ;  CL 为接收字符串长度
            MOV     BX, OFFSET BUFFER        ;  BX 字符串首地址
            MOV     [BX], AL                 ; 存数据长度
REC_LOOP1:  INC     BX                       ; 指针加 1
            CALL    RECEIVE                  ; 接收数据
            TEST    AH,80H
            JNZ     REC_ERROR
            MOV     [BX], AL                 ; 存数据
            LOOP    REC_LOOP1                ; 循环
            ……
REC_ERROR:  ……                               ; 接收出错处理
```

子程序如下：

```
RECEIVE     PROC    FAR                      ; 这是接收数据子程序，出口 AL，AH
REC_CHECK:  MOV     AH, 3                    ; 读通信口状态字
            MOV     DX, 0
            INT     14H
            TEST    AH, 01                   ; 测试数据准备好位
            JZ      REC_CHECK                ; 数据未准备好，再读状态字
            MOV     AH, 2                    ; 读通信端口数据
            MOV     DX, 0
            INT     14H
            RET
RECEIVE     ENDP
```

发送程序与接收程序相似。程序先用“INT 14H, AH=3”来获得 COM1 端口的状态，如果检测到“发送保存寄存器空”位有效，表明可以写入一个数据到 COM1，就用“INT 14H, AH=1”功能，将字符写到 COM1 的发送保存寄存器中。程序也分为主程序和子程序两部分。

主程序如下：

```
            MOV     BX, OFFSET BUFFER        ; BX 为字符串首地址
```

```
                MOV     AL, [BX]                    ; 取数据长度
                MOV     CL, AL
                MOV     CH, 0                       ; CX 为发送数据长度
                CALL    SEND                        ; 发送数据长度
                TEST    AH, 80H                     ; 测试发送是否成功
                JNZ     SEND_ERROR                  ; 不成功，转出错处理
SEND_LOOP1:     INC     BX                          ; 指针加 1
                MOV     AL, [BX]                    ; 取数据
                CALL    SEND                        ; 发送数据
                TEST    AH, 80H                     ; 测试发送是否成功
                JNZ     SEND_ERROR                  ; 不成功，转出错处理
                LOOP    SEND_LOOP1                  ; 循环
                ……
SEND_ERROR:     ……                                  ; 接收出错处理
```

子程序如下：

```
; 这是发送数据子程序，输入参数：AL，输出参数：AL，AH
SEND            PROC    FAR
                PUSH    AX
SEND_CHECK:     MOV     AH,                         ; 读通信口状态字
                MOV     DX, 0
                INT     14H
                TEST    AH, 20H                     ; 测试“发送保存寄存器空”位
                JZ      SEND_CHECK                  ; 发送保存寄存器满，再读状态字
                POP     AX
                MOV     AH, 2                       ; 发送数据
                MOV     DX, 0
                INT     14H
                RET
SEND            ENDP
```

8.2 USB 简介

8.2.1 USB 概述

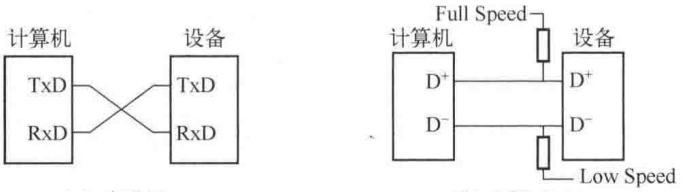
在讨论 USB 技术前，不妨看看外设接口技术的发展历程。多年来，PC 的串口与并口的功能和结构并没有什么变化。串口出现于 1980 年前后，数据传输速率为 115～230 kbps，串口一般用来连接鼠标和外置 Modem；并口的数据传输率比串口快 8 倍，标准并口的数据传输率为 1 Mbps，一般用来连接打印机、扫描仪等。原则上，每个外设必须插在一个接口上，如果所有的接口均被用上，只能通过添加插卡来追加接口了。

串口和并口不但速度有限，而且在使用上很不方便。

1．USB 的定义

USB（Universal Serial Bus，通用串行总线）是连接有 USB 接口的计算机外围设备到计算

机的一种计算机外部总线结构，目前不是用来作为内部总线连接 CPU 到主存以及其他主板设备。USB 是一个通信协议，用来支持计算机与 USB 接口的外设之间的串行数据传输。图 8-25 为串行接口与 USB 接口的信号连接。对于 USB 接口，如果 D^+接上拉电阻，则为全速传输，此时传输速率为 12 Mbps。如果 D^-接上拉电阻，则为低速传输，此时传输速率为 1.5 Mbps。



（a）串行接口　　（b）USB 接口

图 8-25　串行接口与 USB 接口

2．USB 的开发背景

USB 开发的背景源于以下实际需要：USB 出现前，计算机的外接规格十分混乱，如键盘要接 AT 规格的接孔，鼠标要接 COM 口或 PS/2 接口，Modem 要接另一个 COM 口，打印机要接 Parallel Port（并口），每个周边外设都是单独与计算机连接，导致线路紊乱，安装时容易混淆接口。统一这些接口，简化计算机体系结构，这正是 USB 要解决的一个问题。

除了线路紊乱与安装模糊混淆，这些线路都是不可以随意插拔的，必须在开机前装妥才能正常使用。如果在计算机工作期间插上或拔出，虽然有时计算机还是可以继续工作，但大多数时候计算机会停止响应，或者插入的装置无法工作，甚至死机。有时安装使用如打印机之类的外围设备，对于一个普通用户来说都是一个极大的难题。显然，PC 要迅速推广，就必须解决这样的“易用”问题，不能期望一个普通用户需要经过计算机技术培训之后才买计算机产品。

另外，由于以前的计算机的 IRQ 中断控制和输入/输出地址位资源的限制，能同时支持的外设很少。随着计算机的推广，与计算机相连的如投影仪、数码设备等外设迅速增加，用户需要解决大量外设共同使用的问题。这些外设比鼠标、键盘更复杂，但是用户要求它们使用起来简单方便。由此，支持大量外设，使复杂设备简易使用，也是一种新型接口要解决的问题。

尽管在此之前 Apple 公司推出的 IEEE 1394（一种被称为“FireWire（火线）”的串口外部总线）能够完整地解决上述问题，但出于种种原因，这种比 USB 技术更先进的规范并没有得到商业上的推广。

在这样复杂的用户需求与商业背景下，1994 年，Intel 公司提出了 USB 的构想，获得 Digital Equipment Corporation、IBM、NEC、Microsoft、Compaq、Northern Telcom 的支持，并于 1995 年 7 月成立 USB Implementer's Forum（USB-IF，USB 开发者论坛），基于以上问题制订了如下 USB 规范。

① 连接计算机与电话：计算机具有很强的运算能力，而电话提供最广泛的通信互连。运算和通信成为计算机应用的基础，而计算机与通信是两个相对独立发展的产业。USB 旨在提供可以广泛应用于计算机到电话的普遍性的连接。

② Plug-and-Play（即插即用）：从用户端来看，计算机的串行口、并行口和键盘鼠标端口都不能即插即用。USB 则提供真正的即插即用。

③ 端口扩展：计算机已有的串行、并行口等端口只适用于一两种外设并且不易扩展。USB 提供双向低成本低速到中速（USB 3.0 可达 5 Gbps）的通用外设总线，适于连接各种外设且易

于扩展。

3．USB 的变革

到目前为止，USB 共经历以下重大变革。

0.7 版本：1994 年 11 月 11 日发布，是 USB 的最早版本。

1.0 版本：1995 年 11 月 13 日制定 1.0 规格版本，规定 USB 具有两种传输速度：Low-speed，1.5 Mbps；Full-speed，12 Mbps。

2.0 草案版本：1999 年 10 月 5 日发布，制定了 High-speed 的概念、规格。

2.0 版本：2000 年 4 月 27 日由康柏、惠普、英特尔、微软、飞利浦、Lucent、NEC 制订了 2.0 规格版本，有 3 种传输速率：Hi-speed，480Mbps；Low-speed，1.5Mbps；Full-speed，12Mbps，于 2001 年 6 月 21 日测试规格完成。

3.0 版本：2008 年 11 月 18 日由英特尔、微软、惠普、德州仪器、NEC、ST-NXP 等业界巨头组成的 USB 3.0 Promoter Group 宣布，标准接口与线缆样品新一代 USB 3.0 标准已经正式完成并公开发布。新规范提供了数倍于 USB 2.0 的传输速率和更高的节能效率，可广泛用于计算机外围设备和消费电子产品。

表 8-3 是接口传输速率的比较。

表 8-3　一些接口传输速率的比较

接口名称	传输速率
串行端口	115kbps（0.115Mbps）
标准并行端口（Standard Parallel Port，SPP）	115kbps（0.11Mbps）
USB 1.1	12Mbps（1.5Mbps）
ECP/EPP 并行端口	3Mbps
IDE	3.3Mbps～16.7Mbps
IEEE 1394	100Mbps～400Mbps（12.5Mbps～50Mbps）
USB 2.0	480Mbps
USB 3.0	5Gbps

4．USB 系统拓扑结构

USB 系统包含 3 类硬件设备：USB 主机（USB Host），USB 设备（USB Device）和 USB 集线器（USB Hub）。图 8-26 是一个典型的 USB 基本框架。

一个 USB 系统中只有一个 USB 主机。USB 主机有以下功能：管理 USB 系统；每毫秒产生一帧数据；发送配置请求，对 USB 设备进行配置操作；对总线上的错误进行管理和恢复。

USB Hub 用于设备扩展连接，所有 USB 设备都连接在 USB Hub 的端口上。一个 USB 主机总与一个根集线器 （USB Root Hub）相连。USB Hub 为其每个端口提供 100 mA 电流供设备使用。同时，USB Hub 可以通过端口的电气变化诊断出设备的插拔操作，并通过响应 USB Host 的数据包把端口状态汇报给 USB 主机。一般来说，USB 设备与 USB Hub 间的连线长度不超过 5 m，USB 系统的级联不能超过 5 级（包括根集线器）。

USB 设备是指连接在 USB 总线上的外部设备，如 USB 键盘、USB 打印机等。在一个 USB 系统中，USB 设备和 USB Hub 总数不能超过 127 个。USB 设备接收 USB 总线上的所有数据包，通过数据包的地址域来判断是不是发给自己的数据包：若地址不符，则简单地丢弃该数据包；若地址相符，则通过响应 USB 主机的数据包与 USB 设备进行数据传输。

从物理结构上，USB 系统是一个星型结构；但在逻辑结构上，每个 USB 逻辑设备都是如图 8-27 所示直接与 USB 主机相连进行数据传输的。

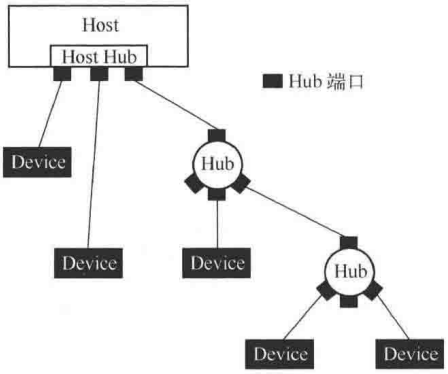


图 8-26　USB 基本框架

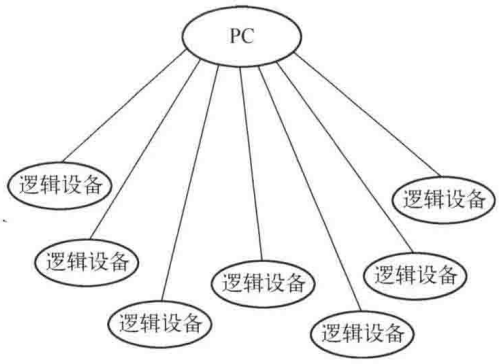


图 8-27　USB 系统逻辑结构

5．USB 的特点

作为外接串行接口，USB 针对用户的需求以及 USB 设备开发商和程序员的要求，具有以下特点（也是其优点）：

① USB 为所有的 USB 外设提供单一的、易于操作的标准的连接类型，这就简化了 USB 外设的设计，同时简化了用户在判断哪个插头对应哪个插槽时的任务，实现了单一的数据通用接口。

② USB 排除了各设备（如鼠标、调制解调器、键盘和打印机）对系统资源的需求，因而减少了硬件的复杂性和对端口的占用，整个 USB 的系统只有一个端口和一个中断，节省了系统资源。

③ USB 支持热插拔（Hot Plug）。也就是说，在不关机的情况下，可以安全地插上和断开 USB 设备，动态地加载驱动程序。其他普通的外围连接标准（如 SCSI 等）必须在关机的情况下增加或移走外围设备。

④ USB 支持 PNP。当插入 USB 设备的时候，计算机系统检测该外设，并且通过自动加载相关的驱动程序来对该设备进行配置，使其正常工作。

⑤ USB 在设备供电方面提供了灵活性。USB 直接连接到 Hub 或者是连接到 Host 的设备可以通过 USB 电缆供电，也可采用自供电方式，如通过电池或者其他电力设备来供电，或使用两种供电方式的组合。同时，这支持节约能源的挂机和唤醒模式。当使用 USB 电缆供电时，USB 总线可为连接在其上的设备提供 5 V 电压、100 mA 电流，最大可提供 500 mA 的电流。这样，新的设备就不需要专门的交流电源了，从而降低了这些设备的成本并提高了性价比。

⑥ USB 提供全速 12 Mbps 的速率和低速 1.5 Mbps 的速率来适应各种不同类型的外设，USB 2.0 还支持 480 Mbps 的高速传输速率。

⑦ USB 2.0 的 High-speed 模式支持音频和视频设备，可以保证其固定带宽。

为了适应各种不同类型外围设备的要求，USB 提供 4 种不同的数据传输类型：控制传输、Bulk 数据传输、中断数据传输和同步数据传输。同步数据传输可为音频和视频等实时设备的实时数据传输提供固定带宽。

USB 端口具有很灵活的扩展性。一个 USB 端口串接上一个 USB Hub，就可以扩展为多个 USB 端口。USB 可以扩展到 127 个外设端口。

8.2.2 USB 工作原理

下面讨论 USB 传输速率比较高，以及连接在 USB 端口上多个外设可以同时与计算机进行数据传输的原因。

1．NRZI 编码

与 RS-232 串行接口不同，USB 在其数据总线上不直接用电平代表逻辑“0”和“1”来传输数据。而是对在其总线上传输的数据进行 NRZI 编码，以确保数据传输的完整性。这种编码方式不需要单独的时钟信号和数据一起发送。

NRZI 编码中数据总线有两种状态，“J”状态和“K”状态，由 D+和 D– 线上出现的电平组合来表示。表 8-4 给出了对应关系。

表 8-4　NRZI 编码

	全速 12 Mbps	低速 1.5 Mbps
“J”状态	D^+=1　D^-=0	D^+=0　D^-=1
“K”状态	D^+=0　D^-=1	D^+=1　D^-=0

NRZI 编码用状态的转变代表“0”，无状态转变代表“1”，如图 8-28 所示。可以看出，NRZI 编码用其数据流中的跳变表示同步信号，即传输数据“0”，就可以保证接收方和发送方的同步。但是，一长串的“1”会导致无电平跳变，从而引起接收方失去同步信号。为避免这种情况的发生，USB 使用了位填充机制，这一点有点像串行通信中同步传输的 IBM 同步数据链路控制规程。

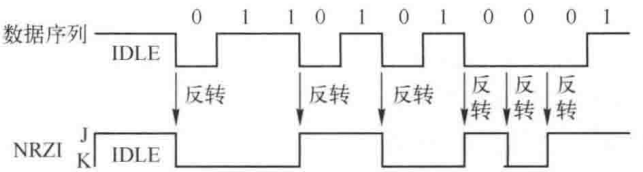


图 8-28　NRZI 编码举例

位填充机制具体做法为：如果传输的数据中有 6 个连续的“1”，那么发送方会在 6 个“1”后填充一个“0”，这就保证了在 7 个位时间里至少有一个跳变。数据接收方在检测到 6 个连续的“1”之后，会把第 7 位的“0”抛弃掉。当然，如果原始数据的第 7 位本来就是“0”，位填充机制还是会在第 7 位填充一个“0”。

NRZI 编码以差分的方式传输数据，使传输线的分布电容对传送速度的影响降低到最小，从而保证 USB 总线能以较高速率传输数据。

2．分组传输

与 IP 技术一样，USB 技术也是基于分组传输方式的。分组交换方式采用长度一定、结构统一的数据分组作为数据传输的基本单位。对 USB 总线来讲，当传输一个文件时，程序按照 USB 协议先把数据分成若干块，再在每块数据前面添上同步信号、包标识，后面添上 CRC 校验，就形成了 USB 封包。一个文件可能有多个封包。全速 USB 总线把 1ms 作为一个时间帧，总线在一个帧内依次传输不同文件的封包。因此从宏观上看，仿佛总线同时对不同的 USB 外设进行数据传输。

分组交换方式在通信之前不需要为通信双方分配一条独占的逻辑链路，可以根据传输数据的性质以及网络能够提供的带宽，为用户动态分配网络资源，从而极大地提高了 USB 总线带宽的利用率。所以，分组交换方式是 USB 数据通信的理想选择。

当系统传输数据时，首先要把数据打包，形成 USB 封包，再通过 NRZI 编码把数据传输给对方。形成 USB 封包是实现 USB 通信的关键。

3．USB 封包

根据信息包所实现的功能可分为 3 种类型：令牌包、数据包和握手包。

令牌包定义数据传输的类型。IN 令牌包（如图 8-29 所示）指示数据是从 USB 设备传到计算机，而 OUT 令牌包表示数据的传输方向正相反，指示数据从计算机传到 USB 设备。IN 令牌包和 OUT 令牌包的结构相同，主要包括命令信息（包标示符 PID）、USB 设备的地址信息（ADDR）、USB 设备接口中要接收数据的端点信息（ENDP）和对包的校验信息（CRC）。在 USB 中，只有主机即计算机才能发送令牌包。

数据包中含有需要传输的数据，主要包括 PID、数据信息和 CRC，如图 8-30 所示。

握手包指明了数据接收是否成功。比如，握手包中有一个图 8-31 所示的 ACK 包，它是由接收数据方发出的，表示数据已正确接收。握手包只包含 PID 字段。

图 8-29　IN 令牌包

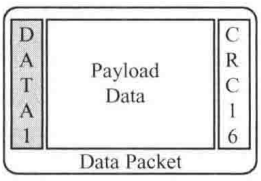


图 8-30　DATA1 数据包

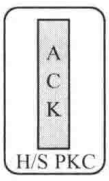


图 8-31　ACK 包

一个 USB 数据交换动作需要三个阶段，即设置阶段、数据阶段和状态阶段。

图 8-32 为主机把数据传输给 USB 外设的数据交换动作。主机首先送出一个 OUT 令牌包，表示数据要输出。紧接着，主机将数据通过 DATA0 数据包传递至设备。如果设备正确接收了数据，则以 ACK 握手包加以响应。

这个过程与前面章节中介绍的条件传输并无太大区别。比如，在 8255A 的 A 口方式 1 输出中，当 A 口的数据要传输到外设时，首先 A 口的联络线 $\overline{\text{OBFA}}$ 有效，通知外设数据准备好；当数据被外设接收后，外设发 $\overline{\text{ACKA}}$ 信号，告诉 8255A 的 A 口数据已被外设取走。显然，这里 A 口的联络线 OBFA 功能和 USB 的 OUT 令牌包功能相似，而 A 口的 $\overline{\text{ACKA}}$ 线和 USB 的 ACK 外设包相似，如图 8-33 所示。

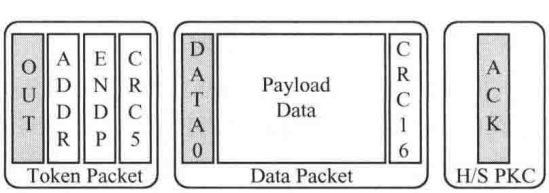


图 8-32　主机输出数据所涉及的封包

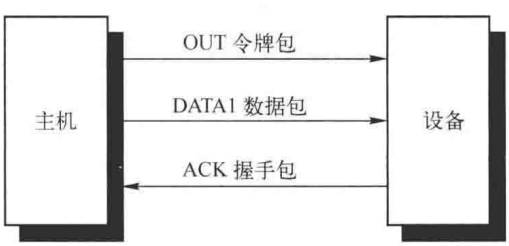


图 8-33　主机输出数据的程序示意

8.2.3 USB 传输方式

针对设备对系统资源需求的不同，USB 规范中规定了 4 种数据传输方式，而且通常所有的传输方式下的主动权都由 PC 控制，即 Host。

1．USB 的 4 种传输方式

（1）同步传输方式（Isochronous）

同步传输提供了确定的带宽和间隔时间，以固定的传输速率连续不断地在主机与 USB 设备之间传输数据。在数据传输发生错误时，USB 并不处理这些错误，而是继续传输新的数据，它被用于时间严格并具有较强容错性的流数据传输，或者用于要求恒定的数据传输率的即时应用中。例如，执行即时通话的网络电话应用时，使用同步传输模式是很好的选择。同步数据要求确定的带宽值和确定的最大传输次数。对于同步传输来说，即时的数据传递比完美的精度和数据的完整性更重要。

（2）中断传输方式（Interrupt）

中断传输方式传输的数据量很小，但这些数据需要及时处理，以达到实时效果。此方式主要用于定时查询设备是否有中断数据要传输。设备的端点模式器的结构决定了它的查询周期为 1～255 ms。这种传输方式典型的应用是少量的分散的不可预测数据的传输。键盘、操纵杆和鼠标就属于这一类型。中断方式传输是单向的，并且对于 Host 来说只有输入。

（3）控制传输方式（Control）

控制传输是双向传输，数据量通常较小，用来处理主机到 USB 设备的数据传输，包括设备控制指令、设备状态查询及确认命令。当 USB 设备收到这些数据和命令后，将依据先进先出的原则处理到达的数据。USB 系统软件主要用来进行查询、配置和给 USB 设备发送通用的命令。控制传输方式可以包括 8、16、32 和 64 字节的数据，这依赖于设备和传输速度。控制传输典型地用于主计算机和 USB 外设之间的端点之间的传输，但是指定供应商的控制传输可能用到其他端点。

（4）批传输方式（Bulk）

批传输主要应用于数据大量传输和接收数据，同时又没有带宽和间隔时间要求的情况下，要求保证传输正确无误的数据。打印机和扫描仪、数码相机属于这种类型。这种类型的设备适合于传输非常慢和大量被延迟的传输，可以等到所有其他类型的数据传输完成之后再传输和接收数据。

2．USB 设备类型

USB 的设备可以接在计算机的任意 USB 接口上。使用 Hub 还可以扩展，使更多的 USB 设备连接到系统中。USB Hub 有一个上行端口（到 Host），有多个下行端口（连接其他设备），从而使整个系统可以扩展连接 127 个外设，其中 Hub 也算外设。对于 USB 系统来说，USB 的 Host 永远在计算机方，所有其他连接到 Host 的都称为设备。设备与设备之间是无法实现直线通信的，只有通过 Host 的管理与调节才能够实现数据的互相传输。在系统中通常会有一个根 Hub，它一般有两个或两个以上/下行端口。

虽然 USB 设备都会表现 USB 的一些基本的特征，但是 USB 设备还是可以分成不同的类型。同类型的设备可以拥有一些共同的行为特征和工作协议，从而使设备驱动程序的书写变得

简单一些。表 8-5 中给出了一些基本的 USB 设备类型。

表 8-5　一些 USB 的设备分类

设备类型	设备举例	类型常量
音频（audio）	扬声器	USB_DEVICE_CLASS_AUDIO
通信	MODEM	USB_DEVICE_CLASS_COMMUNICATIONS
HID	键盘、鼠标	USB_DEVICE_CLASS_HUMAN_INTERFACE
图像	摄像机、扫描仪	USB_DEVICE_CLASS_IMAGE
显示	监视器	USB_DEVICE_CLASS_MONITOR
物理回应设备	动力回馈式游戏操纵杆	USB_DEVICE_CLASS_PHYSICAL_INTERFACE
电源	不间断电源供应	USB_DEVICE_CLASS_POWER
打印机		USB_DEVICE_CLASS_PRINTER
Bulk 存储器	硬盘	USB_DEVICE_CLASS_STORAGE
Hub		USB_DEVICE_CLASS_HUB

8.2.4　USB 设备列举

USB 实现了真正意义上的即插即用。在 USB 规范中有一个非常重要的“动作”或“过程”，这个动作将让计算机知道何种 USB 设备刚接上以及其所含的各种信息。这样，计算机就可以与这个 USB 设备开始进行数据传输了。这个动作称为设备列举（enumeration）。若完成一个设备列举的动作，需要执行诸多的数据交换和设备请求。因此，设备列举是一个非常复杂的过程。

1．USB 描述符

在设备列举过程中，USB 设备向主机传输一个重要的信息，使主机知道这个设备是“谁”，并且启动该设备的驱动程序。这个信息就是 USB 描述符。USB 描述符好像是 USB 外围设备的“履历表”或“身份证”，详细记录着外围设备相关的一切信息。因此，USB 描述符掌握了有关于设备的各种信息与相关的设置。

USB 描述符是一个复杂的数据结构，由设备描述符、配置描述符、接口描述符、端点描述符等其他描述符组合而成。每种描述符都有自己特定的信息，如设备描述符，包含 USB 设备的制造商 ID（Product ID，PID）、产品的 ID（Vendor ID，VID）等信息。

2．设备列举

设备列举包含两方面的功能：一是主机搜集 USB 设备的信息，也就是 USB 描述符；二是主机根据 USB 描述符对设备进行配置，如 USB 设备地址设置、USB 设备各端点数据传输方式设置。

当一个 USB 设备插入计算机时，主机将会进行一个连接过程。

<1> 主机通过地址 0 向设备发送获取描述符的请求。每个刚连接上的设备都必须响应地址 0 的请求。

<2> 设备响应请求，将部分描述符的信息发送到主机端，确认自己已连上。

<3> 主机向设备发送设置地址的请求，分配给设备一个唯一的通信地址，可以与其他设备区分开。

<4> 主机通过新分配的地址向设备发送更多的索要描述符的请求，进一步了解设备的信

息，包括端点数目、电力要求、带宽要求、需要什么样的驱动程序等。

<5> 加载符合 USB 指定的 PID 和 VID 的驱动程序。

8.3 USB 总线转接芯片——CH341 简介

1．概述

CH341 是一个 USB 总线的转接芯片，提供如图 8-34 所示的异步串口、打印口、并口及常用的 2 线和 4 线等同步串行接口。用户可以直接在计算机和 Windows 平台上编写串口、打印口等终端接口程序，通过 USB 总线对 CH341 进行操作，以实现串口、打印口等接口的数据传输的功能。CH341 是全速 USB 设备接口，兼容 USB 2.0，外围元器件只需要晶体和电容，接口电路十分简单，由于厂方提供了丰富的接口程序，非常有利于用户开发基于 PC 的 USB 终端产品。限于篇幅，本书简单介绍两种 CH341 的应用。

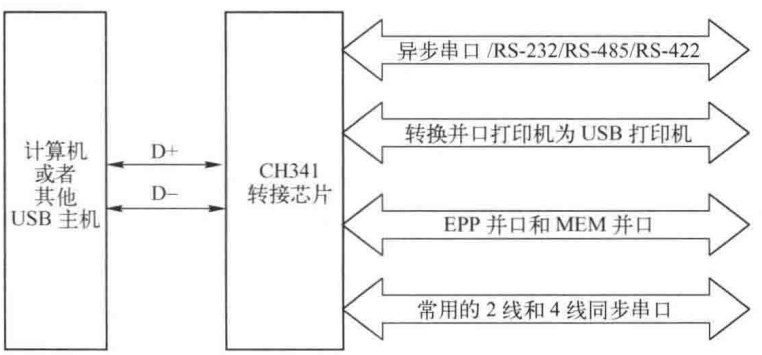


图 8-34　CH341 的功能

2．CH341 应用举例

（1）USB 转 RS232 串口

CH341 内置了独立的收发缓冲区，支持单工、半双工或者全双工异步串行通信。串行数据包括 1 个低电平起始位、5～9 个数据位、1 或 2 个高电平停止位，支持奇校验、偶校验、标志校验、空白校验。CH341 支持常用通信速率有：50 bps、75 bps、100 bps、110 bps、134.5 bps、150 bps、300 bps、600 bps、900 bps、1200 bps、1800 bps、2400 bps、3600 bps、4800 bps、9600 bps、14400 bps、19200 bps、28800 bps、33600 bps、38400 bps、56000 bps、57600 bps、76800 bps、115200 bps、128000 bps、153600 bps、230400 bps、460800 bps、921600 bps、1500000 bps、2000000 bps 等。串口发送信号的波特率误差小于 0.3%，串口接收信号的允许波特率误差不小于 2%。在计算机端的 Windows 操作系统下，CH341 的驱动程序能够仿真标准串口，与绝大部分原串口应用程序完全兼容，通常不需任何修改。

图 8-35 中的是 USB 转 RS-232 串口，P6 是 DB-9 插针，是最基本也是最常用的异步串口。

（2）MEM 并口

在 MEM 并口方式下，CH341 提供了 8 位双向三态数据总线、主要控制线包括 $\overline{WR}$ 引脚、$\overline{RD}$ 引脚（$\overline{DS}$ 引脚的别名）、A_0 引脚（$\overline{AS}$ 引脚的别名）。MEM 方式类似于存储器的读写方式，$\overline{WR}$ 和 $\overline{RD}$ 都是低电平有效的脉冲信号。MEM 的实际操作发生于 $\overline{WR}$ 或者 $\overline{RD}$ 有效期间，对计算机端而言，当 $\overline{WR}$ 有效时对外部电路执行写操作，当 $\overline{RD}$ 有效时对外部电路执行读操作。

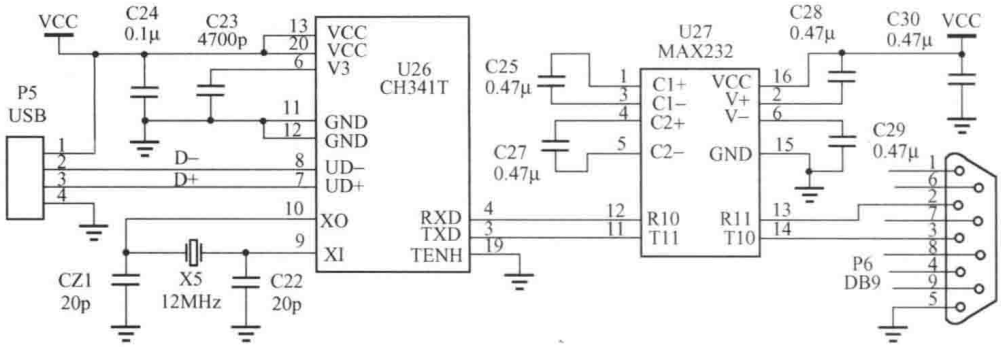


图 8-35　USB 转 RS-232 串口

A_0 用于指示当前读写操作的地址，例如，将 A_0 和 A_0 的反相分别用于两个外部设备的片选，如将 $A_0=1$ 指向外部设备的命令端口，将 $A_0=0$ 指向数据端口。CH341 的 MEM 读写操作支持 $\overline{WAIT}$ 等待信号。$\overline{WR}$ 的低电平有效宽度最小是 0.25 μs，$\overline{RD}$ 的低电平有效宽度最小是 0.33 μs，理想状态下的最大传输速率是 800 KBps。实测传输速率与 EPP 数据读写差不多，但略低于 EPP 地址读写操作的速率。

图 8-36 表示 USB 转 RAM 接口的一种方案，$\overline{ACT}$ 表示 USB 设备配置完成，低电平有效。当 J1 连接 USB 总线，计算机完成枚举后，$\overline{ACT}$ 输出为低，这时发光二极管 D1 发光，通知用户 USB 设备配置完成。RSTI 为外部复位输入，高电平有效。当图 8-36 的电路上电后，电容 C1 电压不能突变，RSTI 线为高电平，CH341 进行复位，随后电容充电，RSTI 线上的电压减小至低电位，这时 CH341 工作在所设置的状态。

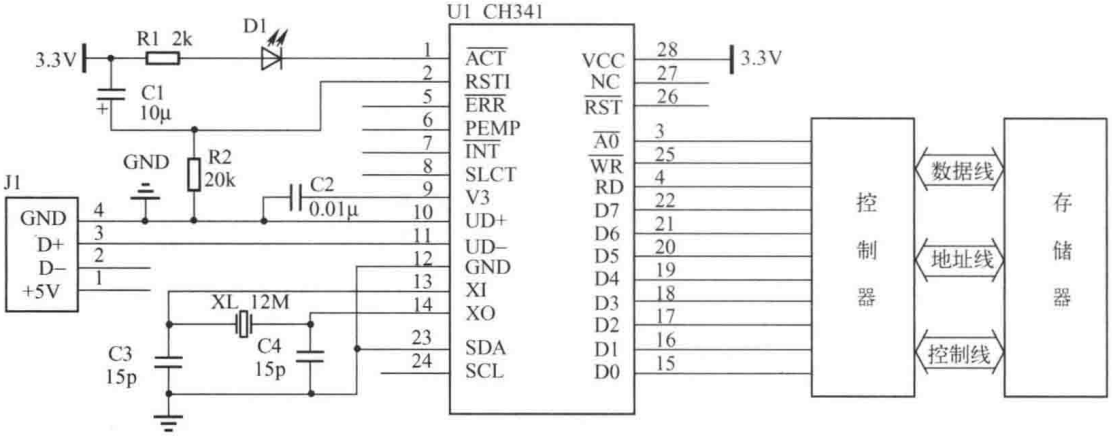


图 8-36　USB 转 RAM 接口

在 MEM 接口方式中，信号线 A_0、$\overline{WR}$ 和 $\overline{RD}$ 接控制器，可设 $A_0=0$ 时表示数据口、$A_0=1$ 为控制口，计算机可以通过 $A_0=1$ 时与控制器传送命令、存储器地址和外电路工作状态，通过 $A_0=0$ 时与数据口传送数据。控制器把数据和地址信息转送到与存储器相连接的数据线和地址线上，并通过控制线向存储器发读写信号，实现存储器的读写操作。

控制器可以由 FPGA 实现，存储器也可换为 AD 芯片和 DA 芯片。当采用 AD 芯片时就建立了一个基于 USB 总线的数据采集终端系统。

习 题 8

1．什么叫同步通信方式？什么叫异步通信方式？它们各有什么优缺点？

2．什么叫波特率因子？什么叫波特率？设波特率因子为 64，波特率为 1200 bps，那么接收时钟频率是多少？

3．标准波特率系列指的是什么？

4．在 RS-232C 标准中，信号电平与 TTL 电平不兼容，RS-232C 标准的 1 和 0 分别对应什么电平，用什么器件完成 TTL 电平之间的转换？

5．什么叫异步工作方式？什么叫同步工作方式？

6．某系统采用串行异步方式与外设通信，发送字符格式由 1 位起始位、7 位数据位、1 位奇偶校验位和 2 位停止位组成，波特率为 1200 bps。试问，该系统每分钟发送多少个字符？若选波特率因子为 16，问发送时钟频率为多少？

7．两台计算机通过 COM2 端口进行串行数据通信，画出电路图并编写汇编语言程序。要求：从一台计算机上键盘输入的字符能传输到另一台计算机，若按下 Esc 键，则退出程序；在程序中，COM2 端口初始化为 4800 bps，8 位数据位，无校验，1 位终止位。

8．简述 USB 的特性和优点。

9．USB 基本框架包含哪几部分？

10．简述 USB 的 NRZI 编码。

11．如何在 D^+与 D^-引线上设置 USB 低速和全速的传输模式？

12．USB 有几种传输方式？简述各种传输的特性，列举相应的 USB 设备。

第 9 章　中断和中断管理

本章导读

- 中断原理
- 中断系统组成及其功能
- 中断源识别及中断优先权
- 8086 中断系统
- 8086 CPU 的中断管理
- 可编程中断控制器 8259A 简介
- IBM PC 硬件中断

当 CPU 用查询的方式与外设交换信息时，CPU 就要浪费很多时间去等待外设。这样就引出一个快速的 CPU 与慢速的外设之间数据传输的矛盾，这也是计算机在发展过程中遇到的一个严重问题。为了解决这个问题，一方面是提高外设的工作速度，另一方面则是利用中断。中断系统是计算机的重要指标之一。

中断功能很有用，常常是必不可少的。中断功能的主要优点是，只有接口需要服务时才能得到 CPU 的响应，而不需要 CPU 不断地去查询。这样，CPU 就可以空出时间去做其他事情，直到接口需要它服务时为止。PC 的键盘管理就是使用中断功能的一个比较好的例子。键盘在工作时，每次按键就要求对 CPU 传输数据，因为 CPU 正在执行其他程序，所以不可能同时又处于等待用户按键的查询循环之中。如果真的等待用户击键，则其他程序永远得不到执行，因为 CPU 的全部注意力都集中在寻找下一次的按键事件。一个简单的解决办法是：使正在执行的程序暂停一下，以便查看键盘接口，看是否已有键被按下。当然，应用程序必须知道什么时候去查看，以及查看多少次，否则它的许多处理时间将白白浪费在对键盘接口的查询之中。PC 的中断功能就是在按键事件刚发生时自动暂停程序，并使 CPU 去执行与击键事件相对应的键盘服务程序。在完成接收输入信息的程序之后，CPU 自动回到原程序去执行下一条指令。

9.1　中断原理

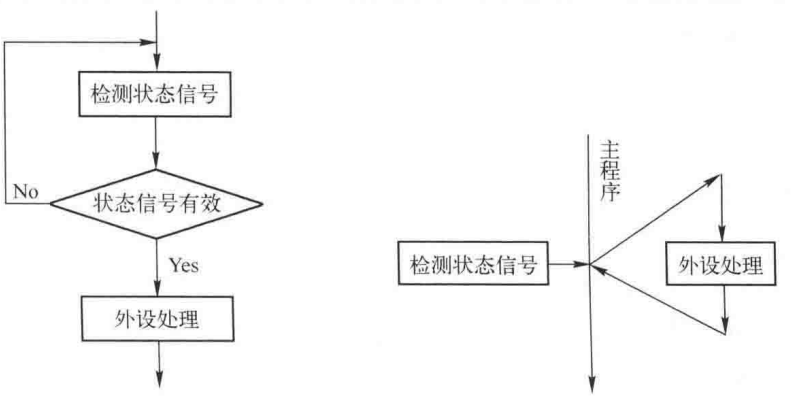
中断传输是计算机接口内容中最精彩也是较为复杂的部分。本节将讨论一些最基本的中断概念。理解和掌握这些概念，对于进一步学习 8086 CPU 中断系统大有裨益。

9.1.1 从无条件传输、条件传输到中断传输

在程序控制下的输入、输出方式有三种，即无条件传输、条件传输和中断传输。现在结合状态线的功能再来讨论这三种方式的关系。

对于无条件传输，由于在任何时候 CPU 都准备为外设服务，因此在速率上可以等效为计算机内部的存储器，显然这种服务不需要外设提供状态线。对于条件传输，由于外设的数据处理速率慢于 CPU 的数据处理速率，外设无法使用无条件传输方式与 CPU 交换数据。为使快速 CPU 和慢速外设在数据传输上的时间达到匹配，设备配置了状态线。CPU 先用无条件方式对状态线进行查询，了解外设的工作状态，判断外设是否准备好与 CPU 交换数据。如果外设已经准备好与 CPU 交换数据，则通过状态线上逻辑信号的改变来通知 CPU，CPU 随即进入与外设的数据交换状态。

中断传输实际上是从条件传输演变而来的，如图 9-1 所示。条件传输最大的缺点就是，为了 CPU 与外设在时间上配合正确，CPU 花大量的时间用无条件方式对状态线进行查询，从而降低了整个系统的工作效率。实际上，CPU 与外设交换数据的时间可能只占 CPU 访问外设时间的非常小的一部分，而这部分才是真正的微机与外设交换数据的主体。当然，造成这个问题的关键就是 CPU 通过软件对状态线的查询，而解决这个问题的一个方案就是中断。



（a）条件传输（程序检测状态信号）　　（b）中断传输（CPU 硬件检测状态信号）

图 9-1　条件传输和中断传输之间的关系

在具有中断功能的 CPU 中有一个硬件部件专门用于检测外设的状态线。平时，CPU 可以专注于程序的执行而不用检测状态线。当外设通过充分的准备，可以与 CPU 进行数据交换时，就通过改变状态线上的信号状态来通知 CPU。当 CPU 通过硬件检测发现状态线上的信号状态发生改变，并且满足一定条件时，CPU 就可以从正在执行的主程序转入与外设的数据交换程序；完成数据交换后，再回到主程序继续执行程序。这就是所谓的中断传输。

在这种 CPU 与外设数据交换的模式中，状态线称为中断请求线，状态线有效表示外设向 CPU 发出中断请求，而专门用于检测外设状态线的硬件则称为中断逻辑电路。

可见，对于外设来讲，条件传输和中断传输没有什么区别，外设只要提供状态线就可以了。状态线实际上就是中断线。如对于 8255A 的 A 口，当其工作在方式 1 输入时，状态线 IBFA 通过芯片内的与门转为中断请求线 INTRA。对于 CPU 来说，条件传输与中断传输却有本质区别。条件传输是用软件查询状态线，CPU 花大量的时间来查询状态线，浪费了 CPU 资源，但系统的结构简单。中断传输是用硬件查询状态线。在查询状态线时，CPU 仍然可以执行其他程

序，从而提高了 CPU 的利用率，但由于多了一个中断逻辑而使系统变得复杂。

9.1.2 中断概念

中断有效地解决了高速 CPU 与慢速设备之间的数据交换时间难以匹配的问题。在条件方式中，CPU 虽然是总线控制器，但在与外设进行数据传输过程中，却由于对状态线的查询工作方式而受控于外设，不能做其他处理。这种功能上的矛盾在中断方式中得到了完美的解决。在中断方式中，尽管 CPU 仍然不停地查询中断请求线，但这是由 CPU 内中断逻辑电路完成的，因此主程序仍然可以执行。即使外设的中断请求信号有效，也不可能随便中断 CPU 当前正在执行的程序。只有 CPU 满足一定的条件时才可能中断当前执行的程序，为这个外设服务。

这里给出图 9-2 的例子。比如，某文秘的日常工作主要是处理文件档案，在工作的大部分时间内，他都可以专注于文档工作；但当有电话铃声来时，如果他不是正在处理一项非常重要的文档，就可以接听电话，完毕后再进行文档工作。这个过程就是典型的中断过程。文秘的处理文档对应于 CPU 的执行主程序或日常事务程序，电话铃响对应于外设的中断请求信号有效，暂停文档相对于 CPU 暂停执行当前的主程序而进入中断响应状态等。

在中断传输方式下，外设应有请求 CPU 服务的权利：当外部设备准备好向 CPU 传输数据时，或者外设已准备就绪接收 CPU 的数据时，或者有某些紧急情况要求处理时，或者定时时间到时，外设向 CPU 发出中断请求，CPU 接收到请求并在一定条件下，暂时停止执行原来的程序而转去进行中断处理，处理好中断服务再返回来执行原来的程序，这就是中断概念。

9.1.3 中断应用

中断方式有效地提高了 CPU 的工作效率，许多用其他工作方式难以处理的操作，往往可以通过中断得以圆满解决。

1．实时故障处理

实时处理计算机部件损坏和计算过程中出现的错误。如计算机内存电路出现问题，由于内存中存储的是程序代码和需要处理的数据，一旦内存出现问题，计算机就无法正常运行，或者处理结果的可靠性无法得到保证。在这种情况下，常用如下方案来检测和处理。在如图 9-3 所示电路中，存取 1 字节的信息包括两方面的内容：一是 8 位的数据，二是该数据的奇偶校验位。当内存电路出现故障时，这 9 位数据原有的奇偶性关系就会被破坏，奇偶检测电路的输出就会有效，产生中断请求信号。

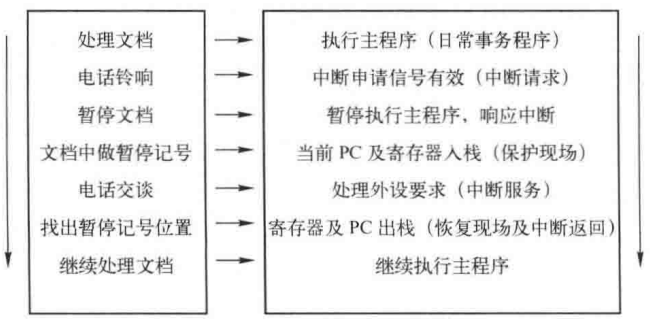


图 9-2 中断举例

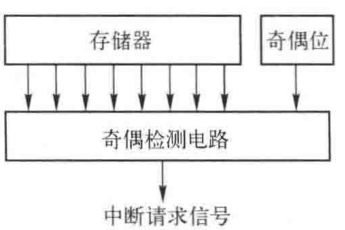


图 9-3 存储器出错检测电路

对于计算出错，有些功能较强的 CPU 安排了软件中断。软件中断的中断源是指令。比如，当 CPU 执行一条除法指令时，如果运行结果溢出，表明所得的商是不正确的，这时就会产生中断，而相应的中断服务程序就处理由于运算结果不正确而带来的问题。

2．分时操作，同时处理

有了中断功能，就可以使 CPU 和外设同时工作。CPU 在启动外设工作后，就继续执行其他程序，同时外设也在工作。当外设把数据准备好后，发出中断请求，请求 CPU 中断它的程序，执行输入或输出（中断处理）。处理完以后，CPU 继续执行主程序，外设也继续工作。有了中断功能，CPU 可以控制多个外设同时工作。虽然 CPU 在不同的时间点上为不同外设工作，从宏观上看，CPU 几乎同时为不同外设工作，极大地发挥了 CPU 高速性的特点。

9.2 中断系统组成及其功能

9.2.1 与中断有关的触发器

正确可靠地实现中断工作方式需要一个非常复杂的中断系统，它实际上是由 CPU 内部的中断逻辑电路和与外部设备相关的中断逻辑部件组成的。因此，详细了解中断系统的组成是比较困难和耗时的。中断系统中有三个与中断有关的重要的触发器，它们是理解和把握中断系统及其处理方式的关键，它们是中断请求触发器、中断屏蔽触发器和中断允许触发器。这三个触发器中，前两个设置在与外部设备相关的中断逻辑电路中，后一个设置在 CPU 的内部电路中。

1．中断请求触发器

中断请求触发器的作用就是产生中断请求信号给 CPU。从本质上讲，它是把外设的状态信号保存在一个触发器中作为中断信号。中断请求触发器有两个特点：① 它的输出可以作为中断请求信号，在满足一定条件的情况下把信号发送给 CPU，且在 CPU 未响应中断时一直保持下去；② 当 CPU 满足一定条件下响应该中断请求信号，执行相关的操作后，该中断请求信号可以被撤销。图 9-4 就是一个简单的中断请求信号产生电路。当状态线高电平有效时，D 触发器输出为 1，向 CPU 请求中断，当 CPU 在中断程序中执行对外设读、写操作时，清除这个中断信号。有时产生状态信号的触发器就直接作为中断请求触发器。

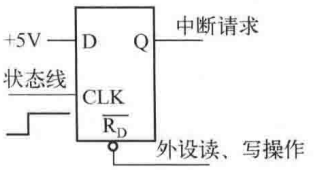


图 9-4 中断请求信号产生电路

发出中断请求信号的外部设备称为中断源。中断源有多种，如：数据输入/输出外设请求中断，定时时间到请求中断，满足规定状态请求中断，电源掉电请求中断，故障报警请求中断，程序调试设置中断。

2．中断屏蔽触发器

中断屏蔽触发器的功能是决定中断请求触发器的输出信号是否可以作为中断请求信号而发送给 CPU。CPU 通过对中断屏蔽触发器的设置，达到对中断源的控制。图 9-5 表示中断屏蔽触发器的作用。对于 CPU 来说，中断屏蔽触发器实际上就是某输出接口中的一位输出线。如果 CPU 不希望某个设备发出中断请求信号，即 CPU 不希望它中断正在执行的程序，那么

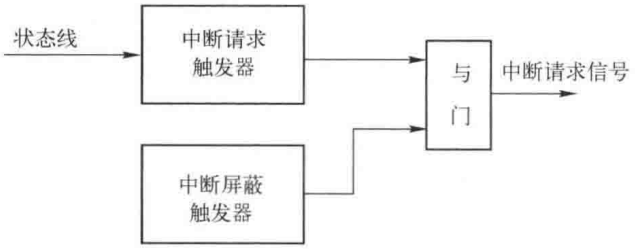


图 9-5　中断屏蔽触发器的作用

CPU 可以通过对输出接口的操作使这个中断屏蔽触发器复位，这样中断请求触发器的信号就不可能通过与门发送给 CPU，此操作称为中断屏蔽。如果 CPU 通过对输出接口的操作使中断屏蔽触发器置 1，中断请求触发器的信号就可以作为有效的中断请求信号送给 CPU。

在第 7 章的图 7-8（8255A 方式 1 输入工作方式）中，A 口的状态线 IBFA 通过一个与门成为中断请求线 INTRA。与门的另一输入端是一个中断允许信号 INTEA，它实际上就是中断屏蔽信号。INTEA 对应于 C 口输出锁存器的第 4 位，可以通过对 C 口 PC4 的置位/复位来设置中断屏蔽信号为逻辑高或低，从而决定 INTRA 是否有效。因此，C 口第 4 位的输出锁存器就是中断屏蔽触发器。

3．中断允许触发器

CPU 控制中断系统还有另一个途径，那就是 CPU 内部的中断允许触发器。CPU 通过对它进行设置，来决定是否对发给它的中断请求信号进行响应。8086 CPU 的标志寄存器中有一个 IF 标志位，这就是中断允许触发器。如果 IF=1，那么允许 CPU 响应中断，称 CPU 是开中断的；如果 IF=0，那么不允许 CPU 响应中断，称 CPU 是关中断的。可以用专门的指令来置位或复位 IF。置位 IF 指令称为开中断指令，复位 IF 指令称为关中断指令。如果 CPU 执行某一程序，不允许任何外设中断，这时可用关中断指令来禁止 CPU 对外设中断请求信号的响应。如果 CPU 允许外设中断当前正在运行的程序，就可用开中断指令来允许 CPU 对外设的中断请求信号做出响应。

有些 CPU，如 8086 CPU，设置了两种中断类型：可屏蔽中断和不可屏蔽中断。可屏蔽中断受中断允许触发器控制，只有当 IF=1 时，CPU 才能响应中断请求信号。而不可屏蔽中断不受中断允许触发器的控制，只要中断请求信号有效，不管 IF 是否为 1，CPU 就必须响应。因此，不可屏蔽中断的中断优先级要大于可屏蔽中断的中断优先级。

9.2.2　中断条件

从中断屏蔽触发器和中断允许触发器的功能来看，如果外设在中断工作方式下与 CPU 交换数据，外设的中断请求信号要想发给 CPU 并能最终得到 CPU 的响应，必须满足如下两个条件。一是中断屏蔽触发器处于非屏蔽状态，在这种情况下，中断请求信号才能发给 CPU。CPU 是否响应这个中断，还要看中断允许触发器是否处于开中断状态。只有 CPU 在开中断的条件下，CPU 才能进入中断响应过程，处理中断事务。这就是第二个条件。

设置中断屏蔽触发器和中断允许触发器，可使 CPU 灵活控制整个中断系统。比如，CPU 通过对中断屏蔽触发器的操作，可以控制到单个中断源；而 CPU 对中断允许触发器的操作，则可以控制整个中断系统。当计算机在加电时，CPU 执行的是对计算机内各部件的初始化操

作。这是非常重要的处理过程，一般不希望受外设影响。这时，CPU 对中断允许触发器进行复位操作，实现关中断。这样，任何外设发中断请求信号给 CPU，都不可能中断 CPU 正在进行的初始化操作。

在正常工作状态时，CPU 通过对中断允许触发器置位操作，实现开中断，即允许 CPU 响应中断。但究竟哪些中断源可以发中断请求信号给 CPU，还要由中断屏蔽触发器的内容来决定。CPU 通过输出指令对中断屏蔽触发器的设置，可以选择和调整中断源，使中断系统处于一个和目前正在运行的程序相适应的工作状态。

由于中断屏蔽触发器和中断允许触发器是用户可以通过指令来设置的，所以由这两个中断触发器所限定的中断条件是用户通过程序施加给中断系统的。还有一些不是用户所能控制的中断条件，如 8259A 的中断服务寄存器，将在下面的章节中讨论。

9.2.3 中断响应过程

中断过程主要包括 3 方面：外设发中断请求信号给 CPU，即中断请求；CPU 对中断请求信号做出反应，即中断响应；CPU 执行对外设操作的子程序，即中断处理。

1．中断请求

当外设通过充分的准备，可以和 CPU 进行数据交换时，就设置中断请求触发器有效，它的输出信号并不是无条件地送给 CPU，只有当中断屏蔽触发器状态为 1，中断请求触发器输出的中断请求信号才会发给 CPU。

2．中断响应

CPU 在没有接到中断请求信号时，一直执行原来的程序（称为主程序）。当 CPU 接到外设的中断请求信号时，CPU 能否马上去为其服务呢？这就要看中断的类型，若为非屏蔽中断请求，则 CPU 执行完当前指令后，做好断点保护工作即可去服务；若为可屏蔽中断请求，CPU 只能得到允许才能去服务。CPU 响应可屏蔽中断申请必须满足的三个条件为：无总线请求，CPU 被允许中断，CPU 执行完现行指令。

3．中断处理

CPU 响应中断后要自动完成 3 项任务：关闭中断；CS、IP 以及 FR 的内容推入堆栈；中断服务程序段地址送入 CS 中，偏移地址送入 IP 中。

关闭中断的原因有两个：① 对于电平触发的中断，当 CPU 响应中断后，如果不关中断，则本次中断有可能会触发新的中断；② 由于中断是 CPU 从正在执行的主程序转向执行中断服务程序，执行完毕后再回到主程序的过程，因此它实质上就是调用子程序的过程。所以，在 CPU 响应中断后，要保护断点和保护现场，这些都是非常重要的工作，是不允许其他外设的中断请求信号打断的。一旦 CPU 响应中断，就可转入中断服务程序之中。中断服务程序的结构如下：

```
PUSH    AX                          ; 保护现场
……
PUSH    BX
STI                                 ; 开中断
……                                  ; 中断处理
```

```
CLI                          ; 关中断
POP     BX                   ; 恢复现场
......
POP     AX
STI                          ; 开中断
IRET                         ; 中断返回
```

可见，中断服务子程序要做以下 6 件事。

（1）保护现场

CPU 响应中断时自动完成寄存器 CS 和 IP 以及标志寄存器 FR 的保护，但主程序中使用的寄存器的保护则由用户视使用情况而定。由于中断服务程序中也要用到某些寄存器，若不保护这些寄存器在中断前的内容，中断服务程序会将其修改。这样，从中断服务程序返回主程序后，程序就有可能无法正确执行下去。由用户保护寄存器的这段程序称为保护现场，实质上是执行 PUSH 指令将需要保护的寄存器内容推入堆栈。

（2）开中断

CPU 接收并响应一个中断后自动关闭中断。但在 CPU 正在处理当前中断源的中断时，有可能出现更优先的中断源发出中断请求信号给 CPU 的情况。此时，应停止对该中断的服务而转入优先级更高的中断处理，故需要再开中断。否则，优先级高的中断将只有在低级中断源的中断处理结束后才能得到响应。当然，没有更高级别的中断，在此也就不必开中断了。

（3）中断服务

中断服务程序的核心就是对某些中断情况进行处理，如传输数据、处理掉电紧急保护和各种报警状态等。

（4）关中断

由于有上述的开中断，因而在此处应对应一个关中断过程，以便下面恢复现场的工作顺利进行而不被打断。

（5）恢复现场

在返回主程序前要将用户保护的寄存器内容从堆栈中弹出，以便返回主程序后继续正确执行主程序。恢复现场用 POP 指令，堆栈为“先进后出”的数据结构，保护现场时，寄存器的先后入栈次序要与出栈时的次序相反。

（6）开中断返回

此处的开中断对应 CPU 响应中断后自动关闭中断。在返回主程序前，也就是中断服务程序的倒数第二条指令往往是开中断指令，最后一条是返回主程序指令 IRET。

9.3 中断源识别及中断优先权

当外设的中断请求信号送给 CPU 且 CPU 满足一定的条件时，CPU 就会进入中断响应过程。由于同类中断源不止一个，但是 CPU 芯片的同类中断输入信号线一般只有一根，于是带来两个非常重要的问题。第一个问题是，CPU 如何知道是哪个中断源发出的中断请求信号？这是一个非常关键的问题，因为只有正确地确定中断源，CPU 才能转到相应的中断服务程序为之服务。这里，确定中断源的方法被称为中断源识别或中断方式。第二个问题是，如果几个外设同时发中断给 CPU，或 CPU 正在执行某个外设中断时，又有其他外设发中断请求信号给

CPU，中断系统应采取什么样的策略来处理？这就是所谓的中断优先权问题。

9.3.1 中断源识别

一个微机系统往往有多个外设。当外设与 CPU 以中断方式进行通信时，CPU 必须从多个外设中判别是谁正在申请中断，以找到相应的服务程序首地址，才能转去为其服务。因而要解决的问题包括两方面：确定中断源，找到该中断服务程序的首地址。下面给出解决问题的两种方案。

1．查询中断

查询中断是用软件查询的方法来确定中断源。这里的软件查询与前面谈到的软件查询方法实现 CPU 与外设通信的概念不同。前面谈到的软件查询是检查外设状态，用来协调外设与 CPU 在时间上的不同步，是 CPU 主动询问外设是否要进行信息交换。此处则是在外设要求与 CPU 交换信息的前提下，从多个设备中查找请求交换信息的那个设备。有关查询中断和条件传输方式的关系将结合下面的例子做进一步讨论。

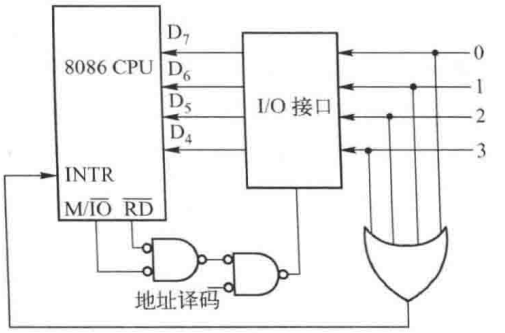


图 9-6　查询中断

对于发出中断请求的设备较多，而 CPU 的中断请求输入线较少，如只有一根的情况，我们可以设置一个中断查询接口电路，用来锁存中断请求信号给 CPU 查询。中断查询接口是一个输入接口，前面谈到的有关输入接口均可用于此处。下面给出一个具有 4 个中断源的中断查询接口电路和分析查询软件，接口电路如图 9-6 所示。在图 9-6 中，4 个外设的中断请求信号通过“或”门接到 CPU 的可屏蔽中断引脚 INTR，设中断请求信号为高电平有效。在该电路中，只要有一个外设申请中断，则 CPU 的 INTR 引脚上就有高电平信号，CPU 在执行一条指令快结束的时候来检测 INTR 信号。当该信号有效，并且满足如下 3 个条件时：无总线请求、CPU 开中断和 CPU 执行完现行指令，就转去中断服务。

现在的问题是，CPU 如何确定中断源并转到对应的服务程序呢？

对于查询中断，一旦 CPU 响应了中断，它就转到一个固定地址去执行程序，我们事先在这个地址处安排了一段查询程序，它可以确定请求服务的设备，转至相应的服务程序。当然，服务程序是预先编好存放在内存区的。对于图 9-6，其查询程序如下：

```
IN      AL, IPORT          ; 从输入接口取中断信息
TEST    AL, 80H            ; 是0号设备请求吗?
JNZ     SEVO               ; 是，转0号设备服务程序
TEST    AL, 40H            ; 否，是1号设备请求吗?
JNZ     SEV1               ; 是，转1号设备服务程序
TEST    AL, 20H            ; 否，是2号设备请求吗?
JNZ     SEV2               ; 是，转2号设备服务程序
TEST    AL, 10H            ; 否，是3号设备请求吗?
JNZ     SEV3               ; 是，转3号设备服务程序
```

查询中断与条件传输之间的工作方式有相似之处，它们都是通过对外设提供的信号即中断

请求信号或状态信号进行查询的，其目的是实现从当前正在执行的程序向为外设服务的子程序的转移。尽管如此，查询中断与条件传输仍然存在本质区别。查询中断是在外设提供的信号有效时才开始查询的，CPU 不需判断外设是否准备好，因此没有浪费 CPU 的时间。条件传输的特点是在状态线处于无效时，程序就开始了查询，判断外设是否准备好，这个过程是 CPU 在等待外设。由于这个状态可能持续很长时间，因此浪费了 CPU 的大量时间。

从上面的程序中可以看出，首先被查询的设备具有较高的中断优先权，因此查询中断可以方便地调整外设的中断优先权。不足的是：对于优先级别高的中断源，CPU 首先进行查询，很快就可以转移到相应的中断服务子程序上去，中断反应快；但对于低级别的中断，即使当前没有其他高级别的中断源的中断，CPU 也要从级别高的中断源到级别低的中断源进行查询。由于级别相对低的中断源的查询语句被安排在查询程序段的后面，所以 CPU 可能会花较长时间才能转移到相应的中断服务程序上去，中断反应较慢。

2．矢量中断

对中断源识别最快的方法是矢量中断，如图 9-7 所示。该方法要求外部设备不仅提供中断请求信号，还要提供一个设备号。比如，当某个外设需要 CPU 为之服务时，就发中断请求信号，CPU 在满足一定的条件时，响应这个外设的中断请求。然后，外设把它的一个设备号通过数据总线送给 CPU。CPU 接收这个设备号并且通过简单的处理，就可从当前正在执行的主程序转移到相应的中断服务子程序上去。

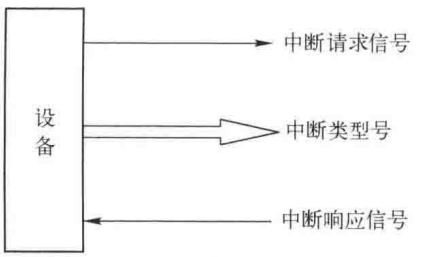


图 9-7　矢量中断

在中断理论中，这种确定中断源的方法称为矢量中断，外设的设备号为中断矢量或中断类型号。每个 I/O 设备都预先指定好各自的中断类型号。当 CPU 响应某个 I/O 设备的中断请求时，控制逻辑就将该 I/O 设备的中断类型号送入 CPU，CPU 根据该中断矢量自动找到相应的中断程序的入口地址，转入中断服务。8086 CPU 中断方式采用的就是这种矢量中断。

这里还有一个问题尚未解决，即外设在发送中断请求信号给 CPU 后，什么时候再把中断类型号发给 CPU？一个最佳的时机就是在 CPU 进入中断响应状态时，外设提供中断类型号给 CPU。那么外设又如何知道 CPU 进入中断响应状态呢？解决这个问题的一个办法就是 CPU 从硬件上提供一个逻辑信号线，该信号线有效时，向外设表明 CPU 已经进入中断响应状态，外设可以把中断类型号送给 CPU 了。这个逻辑信号线称为中断响应线。在 8086 CPU 中，它的符号为 $\overline{\text{INTA}}$。

中断请求信号和中断响应信号是一对握手信号。在驱动一个中断事件过程中，中断请求信号是外设发给 CPU 的，当其有效时，表示外设请求 CPU 为之服务。而中断响应信号是 CPU 发给外设的，当其有效时，表明 CPU 可以为这个外设服务，同时要求外设提供中断类型号。

中断类型号可以由专用的中断控制芯片提供，也可以由接口芯片 74LS245 提供。图 9-8 是 74LS245 芯片作为提供中断类型号 80H 的接口电路图。从图中可以看出，中断类型号由 A_0～A_7 输入端的状态决定，用户可以通过对其与电源或地线的连接而预先设定，也可以由某个输出接口来确定，其输出线与 74LS245 输入端相连。读者可以看出图 9-8 产生的中断类型号是 80H。当外设需要 CPU 为之服务，其状态线有效，D 触发器向 CPU 发出中断请求信号 INTR，CPU 响应中断时，中断响应信号 $\overline{\text{INTA}}$ 有效，使 74LS245 的三态门打开，中断类型号从 74LS245

的三态门送到 CPU 的数据总线。

当 74LS245 要为多个中断请求提供中断矢量时，可用图 9-9 所示的中断优先编码器 74LS148 提供 8 个中断类型号。有关 74LS148 的说明参看 9.3.2 节。

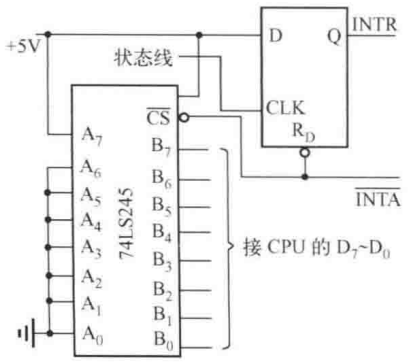


图 9-8　中断类型号 80H 产生电路

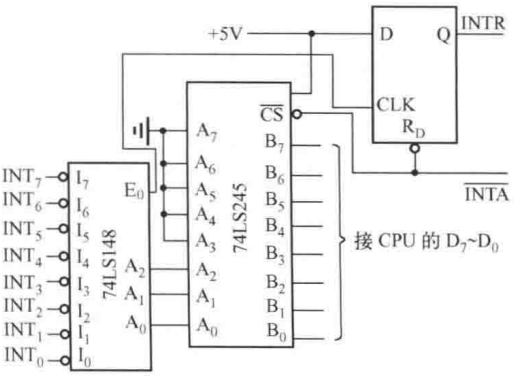


图 9-9　8 个中断类型号 00～07 产生电路

当从 74LS148 的 INT7 上引入中断时（低电平信号），74LS245 提供的中断类型号为 00，而从 INT0 上引入中断时，提供的中断类型号为 07。

9.3.2　中断优先权

微机系统中常常遇到多个中断源同时申请中断的情况，CPU 必须确定优先为哪一个中断源服务和服务的顺序。当 CPU 已在中断处理状态时，如果另一个外设又发出了中断请求信号，这时 CPU 必须确定是否中断当前的中断处理程序而接受更需要紧急处理的中断。解决这些问题就是解决中断的优先排队问题。有两种方法解决中断优先权的问题。

1．软件方案

这种方法常称为查询中断。其中断优先权由查询顺序决定，先被查询的中断源具有高的优先权。使用这种方法需要设置一个中断请求信号的锁存接口，将每个申请中断的请求情况保存下来，以便查询并可对还没有服务的中断请求做一个备忘录。图 9-6 就具有这种功能。软件查询的好处在于可以通过软件修改来改变中断优先权。在图 9-6 的查询程序中，0 号设备具有最高优先权，3 号设备具有最低优先权，只要改变查询顺序就可让 3 号设备具有最高优先权，而不必更改硬件。软件查询确定优先权的缺点是，响应中断慢，服务效率低，因为优先权最低的设备申请服务，必须先将优先权高的设备查询一遍，若设备较多，有可能优先权低的设备很难得到及时服务。

查询中断与第 6 章的异步查询数据传送方式本质上是不同的，虽然这两种方法都是对状态线进行查询，但是查询中断是在状态线有效时进行查询的，不存在查询等待的问题，而异步查询数据传送方式总是在不断查询状态线，这个查询等待过程花费了 CPU 大量的时间，极大地降低了 CPU 的工作效率。

2．硬件方案

（1）链形电路

链形电路是利用外设在系统中的物理位置来决定其中断优先权的，电路如图 9-10 所示。

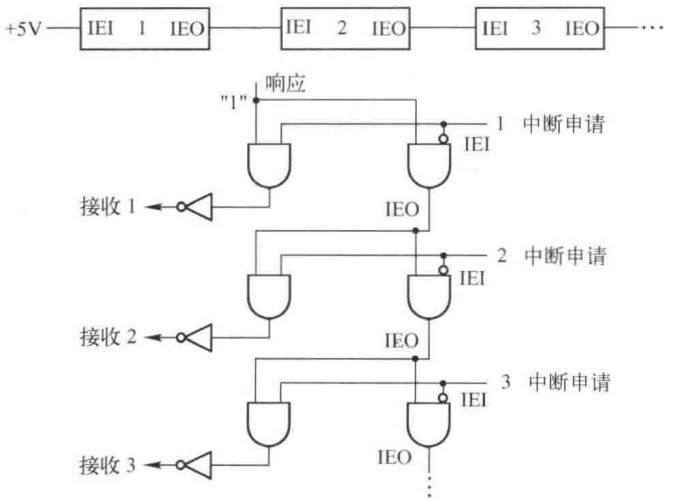
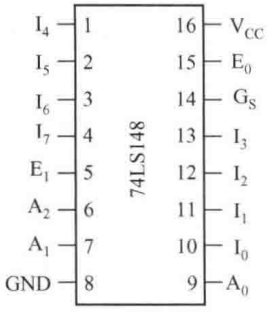


图 9-10 链式中断优先权电路

若 1 号设备发出中断请求（高电平信号）且 CPU 响应时，1 号设备的申请被接收，同时封锁 2 号、3 号设备的中断请求。也就是说，即使 2 号产生中断请求，也不会发送给 CPU。当 1 号设备无中断请求，2 号设备有中断请求时，CPU“响应”信号通过第一级 IEO 传递给 2 号中断请求的“与”门，使 2 号设备的中断请求被接收，同时封锁 3 号设备的中断请求。在响应 2 号设备的中断并为其服务期间，1 号设备发出中断请求，则 CPU 会挂起 2 号设备的服务转去接收优先权高的 1 号设备的申请并为其服务。1 号设备服务完毕，再继续为 2 号设备服务。显然，链式优先权排队电路使优先级别高的设备的中断不被优先级别低的设备所打断，但可随时中断优先级别低的服务。在某些可编程的 I/O 接口芯片中，如 Z80 系列的 PIO、CTC，都有中断优先链，从引脚上体现为链的输入（IEI）和链的输出（IEO）。

（2）编码电路

74LSl48 是一个优先权编码器，它是一个 16 引脚双列直插式 TTL 器件。其引脚图及功能真值表如图 9-11 所示。



输入									输出				
E_1	0	1	2	3	4	5	6	7	A_2	A_1	A_0	G_S	E_0
1	×	×	×	×	×	×	×	×	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	×	×	×	×	×	×	×	0	0	0	0	0	1
0	×	×	×	×	×	×	0	1	0	0	1	0	1
0	×	×	×	×	×	0	1	1	0	1	0	0	1
0	×	×	×	×	0	1	1	1	0	1	1	0	1
0	×	×	×	0	1	1	1	1	1	0	0	0	1
0	×	×	0	1	1	1	1	1	1	0	1	0	1
0	×	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1

图 9-11 74LS148 编码器引脚图及真值表

74LSl48 编码器有 I_0～I_7 共 8 个输入引脚，可接收来自外设的 8 个中断请求信号（低电平有效），E_1 为片选输入信号，低电平有效。从真值表中可知，I_7 输入引脚上的中断请求具有最高优先权，因为无论其他引脚上有无中断请求，即“×”态，只要 I_7 输入引脚上为“0”，则输出引脚 A_2、A_1 和 A_0 的组合就为 000，I_6 引脚上的中断具有次高优先权，因为只有最高优先权引脚上为“1”，即无中断请求时，I_6 引脚上的中断请求才接收，且 $A_2A_1A_0$ 输出为 001。其他以

此类推，I_0 号输入引脚上的中断具有最低优先权。

9.4 8086 中断系统

8086 中断系统有两种类型的中断源：一类是由外部设备产生的中断，称为硬件中断，有时又称为外中断；另一类是由指令在某种运行结果时产生的中断，称为软件中断。硬件中断又分为不可屏蔽中断和可屏蔽中断，硬件中断是通过 CPU 芯片的 INTR 引脚或 NMI 引脚从外部引入的。

无论什么类型的中断，其中断服务子程序的调用都是通过中断类型号来完成的。由于外设送给 CPU 的中断类型号是 8 位的，所以在采用 8086 CPU 的微机系统中有 256 级中断，以 0～255 的序号表示。当中断源多于 256 个时，可以结合查询中断方法来扩展中断源，即几个中断源共用一个中断类型号。中断类型号又如何与中断服务子程序的入口地址相联系呢？首先，8086 CPU 要求在 RAM 中开辟一个区域，作为中断服务程序的地址表。该区域规定，在地址为 00000H～003FFH 的 1 KB 的 RAM 内，中断服务程序的入口地址事先就存入这个区域中，每个入口地址占用 4 个连续字节。高地址存储单元中存放服务程序入口地址的段地址，低地址存储单元中存放偏移地址。若发生某一类型的中断，则该中断服务程序入口地址可从“中断类型号×4”的 RAM 区中找到（如图 9-12 所示）。8086 CPU 的中断类型号有两种方法提供：一种用软件指定，另一种由与外部设备相关的硬件提供。

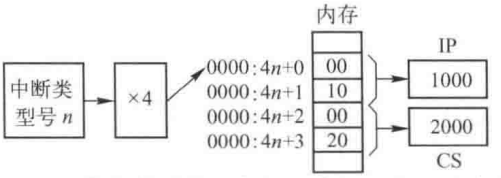


图 9-12 中断类型号和中断程序入口地址的关系

9.4.1 不可屏蔽中断

所谓不可屏蔽中断，就是用户不能通过 CPU 内的中断允许触发器 IF 控制的中断，由 8086 CPU 的 NMI 引脚引入。因为这种中断不受 IF 内容的控制，所以不管 CPU 是否处于开/关中断状态，只要 NMI 有效，CPU 就必须中断当前的程序，转向 NMI 源的中断服务子程序。

NMI 中断请求采用上升沿触发方式，如图 9-13 所示。这种中断一旦产生，在 CPU 内部直接生成中断类型号 02，再取 02 的 4 倍即 08 作为中断服务入口地址表的地址，通过查表得到相应的中断服务程序首地址，转去执行中断服务。由于中断类型号 02 不是由外部设备送给 CPU 的，所以 NMI 中断在时序上不存在 $\overline{\text{INTA}}$ 周期。

9.4.2 可屏蔽中断

可屏蔽中断就是用户可以控制的中断，是通过对 CPU 内的中断允许触发器 IF 的设置来禁止/允许 CPU 响应中断。可屏蔽中断由 8086 CPU 的 INTR 引脚引入。因为这种中断受 IF 内容的控制，所以如果 CPU 处于关中断状态，即使 INTR 有效，CPU 也不会中断当前正在执行的

程序。

INTR 中断请求采用电平触发方式，高电平有效。INTR 中断与 NMI 中断主要有两个区别。① 这种中断一旦发生，CPU 在当前指令执行完后，首先检查标志寄存器中的 IF 是否置“1”，若 IF=0，则该中断请求不被响应，只有当 IF=1 时，CPU 才能响应这个中断请求。② 这种中断请求需要设备提供中断类型号。CPU 响应中断后，取中断类型号的 4 倍作为中断服务入口地址表的地址，通过查表得到相应的中断服务程序首地址，转去执行相应的中断服务程序（如图 9-14 所示）。

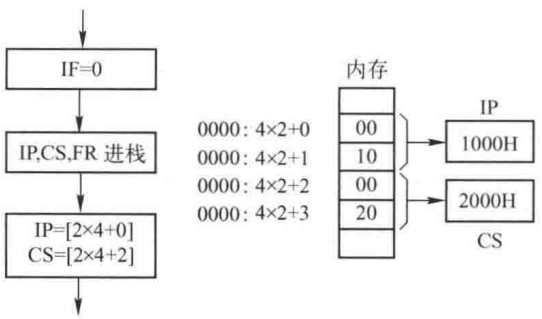


图 9-13　NMI 中断

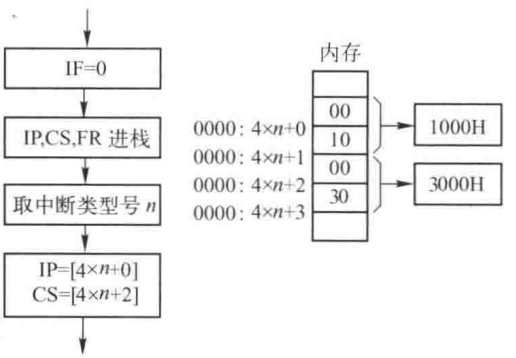


图 9-14　INTR 中断

由于 8086 CPU 芯片仅有一条 INTR 线，直接由 INTR 引脚引入的中断只有一个，但可处理的中断类型号可以是上百个。为了增强其处理外部中断的能力，Intel 公司设计了专用的可编程中断控制器 8259A，用来管理多个外部中断。

9.4.3　软件中断

软件中断是由 8086 指令系统中的某些指令产生的，或由这些指令运行后某种特定的结果产生。其特点如下：

- ❖ 中断类型号包含在指令中或隐含规定。
- ❖ 除单步中断外，任何内部中断不能被禁止。
- ❖ 不运行中断响应总线周期 $\overline{\text{INTA}}$。
- ❖ 正在服务于某种中断类型的中断处理过程中，不能再发生同种类型的中断。

1．除法中断

当进行除法运算时，若除数为 0 或除数太小，使得商数大于相应寄存器所能表示的最大值，则除法出错。这时除法指令就相当于一个中断源，它向 CPU 发出类型 0 中断。

2．溢出中断

当算术运算产生溢出时，将在 INTO 指令控制下向 CPU 发出类型 4 的中断，即溢出中断。

也许有些读者可能提出这样的问题：既然除法指令可以直接产生除法中断，为什么产生算术运算结果溢出的加法指令不能直接产生中断呢？这是因为与加法指令操作有关的数据有两种：一种是无符号的数，对于这种数的处理，是不存在溢出的；另一种是有符号的数，所谓溢出的概念就是针对这种数的操作的。这样用户在编程时，如果处理的是无符号数，就不需考虑

溢出问题。如果处理的是有符号数，运行结果的溢出问题就必须给予解决，相应的方法就是在算术指令的后面语句取溢出中断指令，并编写相应的中断处理程序及设置中断入口地址表。

下面通过例题了解一个软件中断的产生和执行过程。

```
DATA    SEGMENT                          ; 定义数据段
ADD1    DB      ?
ADD2    DB      ?
DATA    ENDS
CODE    SEGMENT                          ; 定义代码段
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     AX, 0                    ; 填写中断地址表
        MOV     ES, AX
        MOV     DI, 04*4
        MOV     AX, OFFSET INTO1         ; 存中断程序首地址的偏移量
        CLD
        STOSW
        MOV     AX, CS                   ; 存中断程序首地址的段地址
        STOSW
        MOV     BL, 0
        MOV     AL, ADD1
        ADD     AL, ADD2                 ; 计算 ADD1+ADD2
        INTO                             ; 若有溢出，则转溢出处理
        MOV     AL, BL
        MOV     AX, 4C00H
        INT     21H
INTO1   PROC                             ; 溢出处理
        PUSH    AX
        MOV     BL,0FFH
        POP     AX
        IRET                             ; 中断返回
INTO1   ENDP
CODE    ENDS
        END     START
```

上例中，中断服务程序和主程序在同一段，中断服务程序的首地址分两次置入中断服务入口地址表中。段寄存器 ES 中置入表的段地址即 0000H，寄存器 DI 中放入中断类型 4 所对应表的偏移地址，以增址型指令将 AX 中的中断服务程序的偏移地址置入 ES:DI 指示的中断服务程序入口地址表的低字节单元中，同样将服务程序的段地址置入表的高字节单元。

该程序的执行结果由两个加数决定，如 92H+8AH，则产生溢出中断，执行中断服务程序后，BL=0FFH。又如，62H–1AH 不产生溢出，也不执行中断服务，BL=00。

3．单步中断和断点中断

单步和断点是很有价值的程序调试手段，任何一个开发装置几乎都具有这两种基本的调试方法。8086 CPU 可用软件中断方便地实现这两种功能。

（1）单步中断

当 8086 CPU 的标志寄存器 TF 为 1 时，8086 CPU 处于单步工作方式，CPU 在每条指令

执行后自动产生类型1的中断。在类型1的中断服务程序中，再单步执行指令是无意义的，因此CPU先将标志位推入堆栈，再清除TF和IF标志，以禁止外部中断和单步中断。中断服务程序可以写在存储器的任意区域，但该服务程序的入口地址应置入地址为1×4的连续4个存储单元中。8086 CPU中没有对TF标志置“1”和清“0”的指令，所以在执行单步中断指令之前，利用下面程序将TF标志置“1”；同理，退出单步工作方式时，执行类似程序将TF清“0”。

```
PUSHF                          ; 标志寄存器内容入栈
POP     AX                     ; 将标志寄存器内容弹进AX
OR      AX, 0100H              ; 置AX的第8位为“1”，对应于TF=“1”
PUSH    AX                     ; 置对应于TF=“1”的AX入栈
POPF                           ; 恢复标志寄存器内容
INT     01                     ; 单步中断
```

（2）断点中断

断点可设在程序任何一条指令的开始处，可以阻止程序的正常运行，以进行某些检查和处理。“INT 3”指令是1字节指令，可将这条指令的目的代码嵌入任意条指令的操作码处，从而实现断点中断。

若在调试程序时，发现要在某处插入一些新的指令，那么可用“INT 3”指令帮助实现。方法是先保护某一指令字节，用“INT 3”的机器码代替它。中断处理程序中包含所要插入的新指令，处理过程返回之前，恢复被保护指令字节，再把保护在堆栈里IP的值减1，当从中断返回时，就从被恢复的指令开始执行。

4．软中断

软中断对读者来说并不生疏，在软件编程章节中所介绍的系统调用就是软中断应用的典型例子。软中断是由中断指令引起的。中断指令的指令格式为“INT n”，操作数n就是中断类型号。当CPU执行中断指令“INT n”后，立即产生一个中断。

“INT n”是一条双字节指令，目的代码为“CD n”，只要提供n，就可从n×4的存储单元中找到中断服务程序的入口地址。

可以利用中断指令“INT n”对上述的软件中断进行另一种诠释。对于除法中断，当CPU执行完指令DIV后，如果结果满足中断条件，则产生中断。这时，可以认为这条除法指令相当于“INT 0”指令。对于溢出中断指令INTO，当算术运算产生溢出时，产生中断，这时可以认为INTO相当于指令“INT 4”。如果算术运算不产生溢出，INTO指令相当于空操作指令NOP。

9.4.4 中断概念的再讨论

中断过程实际上是CPU从执行当前主程序转到执行为外设服务的子程序。因此从这个角度来看，中断过程是一个调用子程序的过程。所以，子程序中的断点与现场保护、断点与现场恢复等概念在中断服务子程序中都是存在的。当然，中断过程与子程序调用还是有很大差别的。首先，调用子程序的过程是一个无条件过程，程序中只要有CALL语句，就一定能够实现主程序向子程序的转移；中断过程的中断服务程序的调用一般是有条件的，如当CPU处于关中断状态下，可屏蔽中断请求就不可能实现从当前正在执行的主程序向中断服务程序的转移。其次，子程序调用在整个程序执行中的位置是固定的，只有有CALL语句的地方才会发生调

用过程。对于硬件中断过程，只要条件满足，在整个程序执行的任一时间点都有可能发生从主程序向中断服务子程序的转移事件，即硬件中断产生的调用过程是随机的，不可预测的。

结合中断指令“INT　n”，可以这样理解外部中断：当外部中断源发中断给 CPU 时，如果 CPU 满足一定的条件，处于开中断状态，CPU 就可以响应中断；这时，外设在 CPU 正在执行指令与其下一个指令之间，等效“插入”了一个“INT　n”指令。这里的 n 就是外设提供的中断类型号。注意：这里用“等效”两字表示，实际过程中是不存在插入“INT　n”指令的操作的，但 CPU 确实完成了类似“INT　n”指令的功能，实现了主程序向中断程序的转移。

编写中断处理程序和编写子程序一样，所使用的汇编语言指令没有特殊限制，只是中断程序返回时使用 IRET 指令。这条指令的工作步骤和中断发生时的工作步骤正好相反。它先把 IP、CS 和标志寄存器的内容出栈，然后返回到中断发生时紧接着的下一条指令。

9.5　8086 CPU 的中断管理

9.5.1　8086 CPU 的中断处理顺序

8086 CPU 的中断优先权排列从高到低为：① 内部中断，即除法出错中断，溢出中断，中断指令“INT　n”；② NMI；③ INTR；④ 单步中断。8086 CPU 对中断的处理可用如图 9-15 所示的流程表示。

9.5.2　8086 CPU 的中断服务入口地址表

存放中断子程序入口地址的内存区域为中断服务入口地址表。8086 系统把中断服务入口地址表中的中断入口明确地分成三部分。

第一部分是类型 0 到类型 4，共 5 种类型，定义为专用中断，占表中 000～013H 的位置，共 20 字节。这 5 种中断的入口已由系统定义，不允许用户做任何修改。其中：INT　0，除法出错中断；INT　1，单步中断；INT　2，外部引入不可屏蔽中断；INT　3，断点中断；INT　4 或 INTO，溢出中断。

第二部分是类型 5 到类型 31H，为系统备用中断。这是 Intel 公司为软件、硬件开发保留的中断类型，一般不允许用户改做其他用途。在微机系统中，许多中断已被系统开发使用，如类型 21H 已用做系统功能调用的软件中断。

第三部分是类型 32H 到类型 0FFH，可供用户使用。这些中断可由用户定义为软中断，由“INT　n”指令引入，也可以是通过 INTR 引脚直接引入的或通过中断控制器 8259A 引入的可屏蔽硬件中断。

中断服务入口地址表又可称为中断指针表或中断矢量表，每个入口都是低位 2 字节为偏移地址，高位 2 字节为段地址，如图 9-16 所示。

9.5.3　中断入口地址设置

前面谈到 8086 利用矢量中断的方法，一旦响应中断就可由中断类型号通过对中断入口地

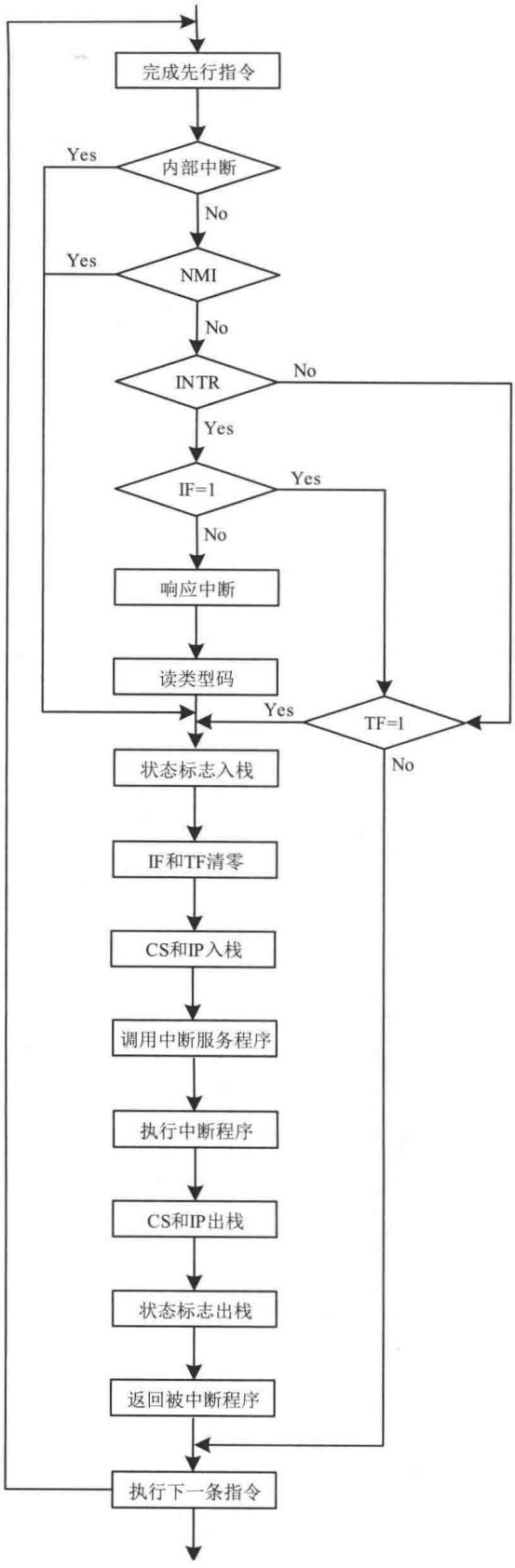


图 9-15　8086 CPU 中断处理顺序流程

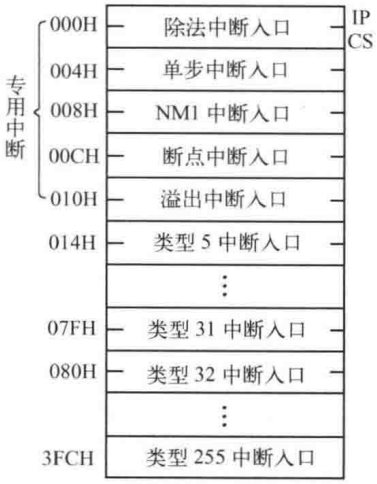


图 9-16　中断服务入口地址

址表的查表，方便地找到中断服务程序的入口地址。在中断入口地址表规定的内存区中，每4个连续字节存放一个中断服务程序首地址。由于8086 CPU的中断类型号只有256个，因此中断入口地址表是一个1 KB大小的表格。尽管表格规定了内存区域，但表中的内容即中断服务程序地址是用户任选的。为了让CPU响应中断后正确转入中断服务，中断矢量表的建立是非常重要的。这里介绍建立这个表格的4种方法。

1．用串指令

串指令STOSW将AX寄存器中的内容写入附加段的DI所指向的目标偏移地址单元。只要将ES设定为0，DI中设定为n×4，使用STOSW指令即可完成中断服务程序首地址的装入。

```
        ……
        CLI                                    ; 关中断
        MOV     AX, 0
        MOV     ES, AX                         ; 置附件段基地址为0
        MOV     DI, n*4                        ; 置附件段偏移地址到DI
        MOV     AX, OFFSET INT_VCE             ; 置中断程序首地址的偏移量到AX
        CLD
        STOSW                                  ; 偏移量填到中断地址表
        MOV     AX, SEG INT_VCE                ; 置中断程序首地址的段基地址到AX
        STOSW                                  ; 段基地址填到中断地址表
        STI                                    ; 开中断
        ……
```

2．用伪指令

指示性语句AT和ORG均可指定存储单元的绝对地址。AT可指定段地址（16位），而ORG将指定偏移地址。中断矢量表的段地址可用指令“INT-TBL SEGMENT AT 0”设定；中断矢量表的偏移地址可用指令“ORG n*4”设定，n为中断类型号。然后，可用DD伪指令将中断服务程序的首地址装入。

```
INT-TBL   SEGMENT  AT 0                        ; 定义INT-TBL段，段基地址为0
          ORG      n*4                          ; 指定偏移地址
          DD       INT-VCE                      ; 存中断程序入口地址
INT-TBL   ENDS
          ……                                   ; 其他处理
MCODE     SEGMENT                              ; 主程序
          ……                                   ; 其他处理
INT-VCE   PROC     FAR                          ; 中断服务程序
          ……
          IRET
INT-VCE   ENDP
          ……
```

3．用系统调用

利用软中断指令“INT　21H”以及专门为更新中断服务程序地址的25H号功能来设置中断地址有两个非常显著的优点：其一，DOS会采取措施用最安全可行的方法来存放中断矢量；其二，使用时范围更广泛。

使用25H功能时要求：AL=中断类型号；DS:DX=中断服务程序首地址的段、偏移地址。

下面的程序完成中断类型号为60H的中断地址置入。

```
        PUSH    DS
        MOV     DX, SEG INT60H              ; 段基地址送 DS
        MOV     DS, DX
        MOV     DX, OFFSET INT60H           ; 偏移地址送 DX
        MOV     AL, 60H                     ; 中断类型号送 AL
        MOV     AH, 25H
        INT     21H
        POP     DS
```

由于在系统中断类型号0～255中，许多类型号已由系统使用，如果用户希望修改某个中断服务程序的首地址，应将原有的中断服务程序地址用DOS调用的功能35H保存起来，以便从用户程序中退出来时，再用25H功能恢复，而不影响系统的正常工作。

系统调用的35H功能是针对指定的中断类型号，得到其中断处理程序的地址。

使用35H功能时要求：AL=中断类型号；返回时ES中是段地址，而BX中是偏移地址。

下面的例子是对机器的中断类型0，将其当前中断服务程序入口地址取出，并且保存到变量INTOSEG和INTOFF中。

```
INTOSEG DW      ?
INTOFF  DW      ?
        MOV     AH, 35H                     ; 功能号送 AH
        MOV     AL, 0                       ; 中断类型号送 AL
        INT     21H                         ; 中断调用
        MOV     INTOSEG, ES                 ; 存段基地址
        MOV     INTOFF, BX                  ; 存偏移地址
        …
```

下面是用户更改系统中断类型1CH，而写入新的中断服务程序的例子。

```
DATA    SEGMENT                             ; 定义数据段
CC1     DW      0
SS1     DW      0
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     AL, 1CH                     ; 中断类型号送 AL
        MOV     AH, 35H                     ; 功能号送 AH
        INT     21H                         ; 系统调用
        PUSH    ES                          ; 存原来的中断地址
        PUSH    BX
        PUSH    DS
        MOV     DX, OFFSET CLINT            ; 设置新的中断地址
        MOV     AX, SEG_CLINT
        MOV     DS, AX
        MOV     AL, 1CH
        MOV     AH, 25H
        INT     21H
        POP     DS
```

```
        ...
CLINT   PROC    NEAR
        PUSH    DS
        PUSH    BX
        ...
        IRET
CLINT   ENDP
CODE    ENDS
        END  START
```

4．直接装入法

若外设的中断类型号为 6BH，则此中断类型号对应的中断矢量表地址为从 001ACH 开始的 4 个存储单元。设中断服务程序段地址是 1000H，偏移地址为 2000H，可用传输指令将已知的中断服务程序首地址置入中断入口地址表中。

```
        MOV     AX, 0
        MOV     DS, AX                          ; 置数据段段基地址为 0
        MOV     AX, 2000H
        MOV     WORD PTR [01ACH], AX            ; 对偏移地址为 01ACH 的单元
        MOV     AX, 1000H                       ; 送双字
        MOV     WORD PTR [01ACH+2l, AX
```

9.6 可编程中断控制器 8259A 简介

8259A 可编程中断控制器（PIC）又称为优先权控制器，可为 CPU 处理 8 级向量优先中断，具有单+5 V 电源供电，可与其他 8259A 芯片级联来扩大中断功能，优先权方式在执行主程序的任何时间里能够动态地改变。无论 8086 CPU 工作在最小模式还是最大模式，都可以与之配套使用。

9.6.1 8259A 的内部结构及引脚分配

1．内部结构

图 9-17 为 8259A 的内部结构图，由以下 8 部分组成。

（1）中断请求寄存器（IRR）

该寄存器用来存放由外部输入的中断请求信号 IR_7～IR_0，当某个输入端为高电平时，该寄存器的相应位置“1”。

（2）中断服务寄存器（ISR）

该寄存器记录正在处理中的中断请求。当任何一级中断被响应，CPU 正在执行它的中断服务程序时，ISR 寄存器中的相应位置“1”，一直保持到该级中断处理过程结束为止。多重中断情况下，ISR 寄存器中可有多位被同时置“1”。

（3）优先权判别器（PR）

当输入端 IR_7～IR_0 中有多个中断请求信号同时产生时，由 PR 判定哪个中断请求具有最高优先权，且在 $\overline{INTA}$ 脉冲期间把它置入中断服务寄存器 ISR 的相应位。

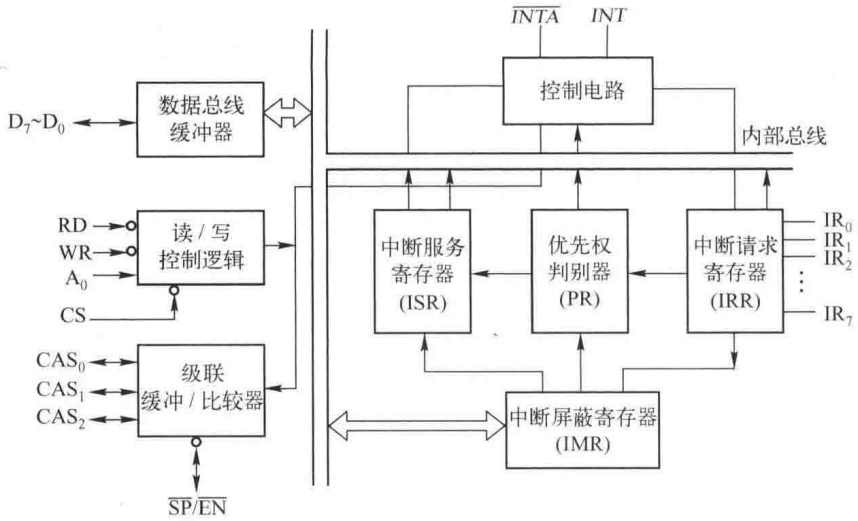


图 9-17　8259A 内部结构

（4）级联缓冲/比较器

一片 8259A 只能接收 8 级中断。当超过 8 级时，可用多片 8259A 级联使用，构成主从关系。对于主 8259A，其级联信号 CAS_2～CAS_0 是输出信号；而对于从 8259A，级联信号 CAS_2～CAS_0 是输入信号。此时，主 8259A 的 $\overline{SP}$ 端为“1”，从 8259A 的 $\overline{SP}$ 端为“0”，且从 8259A 的 INT 输出接到主 8259A 的中断输入端 IR 上，因而可把中断扩展到 64 级。图 9-18 为三片 8259A 级联的连接图。

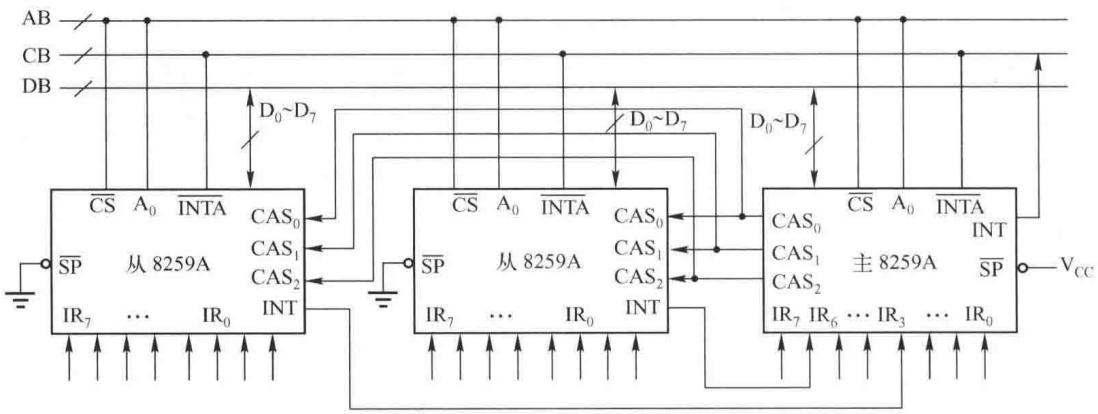


图 9-18　三片 8259A 级联电路

（5）中断屏蔽寄存器（IMR）

该寄存器中存放有关被屏蔽的中断线上的信息，表示禁止这一级中断请求进入系统，通过 IMR 寄存器可实现对各级中断的有选择的屏蔽。

前面这 4 部分的详细原理如图 9-19 所示。

（6）控制电路

8259A 内部的控制电路根据中断请求寄存器 IRR 的置位情况和优先权判别器 PR 的判定结果，向 8259A 内部其他部件发出控制信号，且向 CPU 发出中断请求信号 INT 和接收来自 CPU 的中断响应信号 $\overline{INTA}$，控制 8259A 进入中断服务状态。

（7）读/写控制逻辑

一片 8259A 只占两个 I/O 端口地址，用地址线 A0 来选端口，端口地址的其他高位通过地址译码电路产生片选信号 $\overline{CS}$。由 $\overline{RD}$ 和 $\overline{WR}$ 信号控制可将命令写入有关的控制寄存器，或读出内部寄存器的内容。

（8）数据总线缓冲器

数据总线缓冲器是一个双向 8 位三态缓冲器，构成 8259A 与 CPU 之间的数据接口。

2．8259A 的引脚分配

前面介绍了 8259A 芯片的内部结构，这些结构的功能由外部引脚的正确连接来实现，8259A 芯片的引脚可分为如下 4 类：

- 与外部设备连接的中断请求输入引脚 IR_0～IR_7。
- 与 CPU 连接的数据通路和控制信号：D_0～D_7，$\overline{WR}$，$\overline{RD}$，INT，$\overline{INTA}$。
- 用于 8259A 级联的引脚 CAS_0～CAS_2，$\overline{SP}/\overline{EN}$。
- 端口地址选择信号 $\overline{CS}$ 和 A_0。

8259A 芯片的引脚分配如图 9-20 所示。

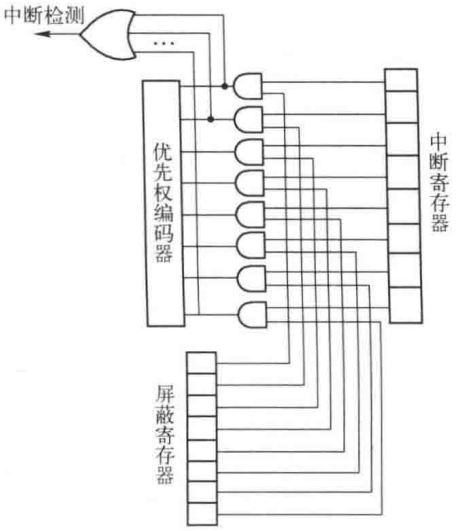


图 9-19　8259 中断逻辑

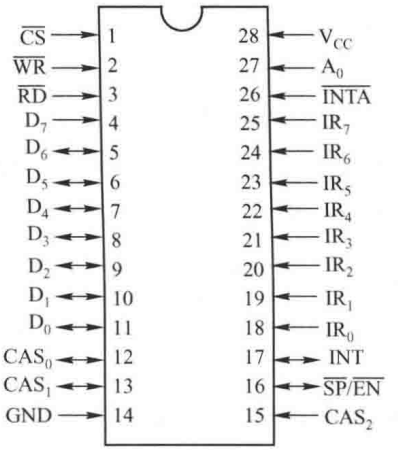


图 9-20　8259A 芯片的引脚

9.6.2　8259A 的中断管理方式

8259A 具有非常灵活的中断管理方式，可以满足使用者的各种不同要求。中断优先权的管理是中断管理的核心问题，8259A 对中断的管理可分为对优先权的管理和对中断结束的管理。

1．中断优先权管理

8259A 对中断优先权的管理可分为 4 种情况：完全嵌套方式，自动循环方式，中断屏蔽方式和特殊完全嵌套方式。

（1）完全嵌套方式

在此种方式下，8259A 的中断请求输入端引入的中断具有固定的优先权排队顺序，IR_0 为最高优先级，IR_1 为次高优先级……IR_7 为最低优先级。同时，高优先级的中断可进入低优先

级，但低优先级不能进入高优先级或同等优先级。

（2）自动循环方式

在完全嵌套方式下，中断请求 IR_0～IR_7 的优先级别是固定不变的，使得从 IR0 引入的中断总是具有最高优先权。在某些情况下需要改变这种优先级别，这时可采用自动循环方式，从 IR_0～IR_7 引入的中断轮流具有最高优先权。也就是说，当任何一级中断被处理完，它的优先级别就被改变为最低，而最高优先级分配给该中断的下一级中断。例如，为 IR_3 引入中断服务，若服务完毕，则 IR_3 为最低优先级，IR_4 有最高优先级，IR_5 有次高优先级。

（3）中断屏蔽方式

用中断屏蔽方式管理优先权有两种方法。

① 普通屏蔽方式。这种方式是在中断屏蔽寄存器 IMR 中，将某一位或某几位置“1”，屏蔽掉相应级别的中断请求，CPU 在执行主程序时将 IMR 寄存器的相应的位置“1”。也可以在 CPU 执行某级的中断服务中，禁止比它高的中断进入，在服务程序中将 IMR 寄存器的相应位置“1”屏蔽。

② 采用特殊屏蔽方式，这时可以使低优先级别的中断进入正在服务的高优先级别中。

（4）特殊完全嵌套方式

特殊完全嵌套方式用在 8259A 有级联的情况下。当任何一个 8259A 从片接收到一个中断请求，判别其为当前最高优先级时，则响应这一中断，通过 INT 端向 8259A 主片相应的 IR 端提出中断请求。如果这时 8259A 主片中 ISR 相应的位已置“1”，则说明该 8259A 从片的其他输入端已提出过申请，且正在服务。由于 8259A 从片的判优电路判别到刚申请的中断优先级最高，故应停止现行中断服务转去为刚申请的中断服务。8259A 有级联的情况下，按完全嵌套方式管理优先权。显然，接在主片 IR3 上的从片比接在 IR4 上的从片具有高的优先权，而主片 IR_0、IR_1 和 IR_2 上的中断比从片具有更高优先权。

2．8259A 中断结束的管理方式

当 8259A 响应某一级中断而为其服务时，中断服务寄存器 ISR 的相应位置“1”，当有更高级的中断请求进入时，ISR 的相应的位又要置“1”，因而 ISR 寄存器中可有多位同时置“1”，在中断服务结束时，ISR 中相应位清“0”，以便再次接收同级别的中断。中断结束的管理就是用不同的方式使 ISR 的相应的位清“0”，并且确定下面的优先排队。8259A 中断结束的管理分 3 种情况来讨论。

（1）完全嵌套情况

采用完全嵌套方式是指中断嵌套过程中，中断服务寄存器 ISR 的内容在不断变化。例如，8259A 正在为 IR7 的中断请求服务时，ISR 中的 D_7 位置“1”。此时，有高级中断 IR_6 请求服务，若允许为其服务，ISR 中的 D_6 位又需置“1”。假如有更高级的中断，且为其服务，ISR 将对应的位置“1”。总之，在完全嵌套方式下，最多可达 8 级中断嵌套，在中断服务结束时，也应按从高到低的次序结束。每个中断结束，要使 ISR 中相应的位清“0”，待全部嵌套中断均结束后，则 ISR 中每位均为“0”。

8259A 在完全嵌套方式下，可采用 3 种中断结束方式。

① 一般 EOI 方式。当任何一级中断服务程序结束时，给 8259A 传输一个 EOI 命令，8259A 将 ISR 寄存器中级别最高的置“1”的位清“0”。这种方式只有在当前结束的中断总是尚未处

理完的级别最高的中断时，才能使用这种结束方式。如果在中断服务中修改过中断级别，则不能采用这种方式。

② 特殊 EOI 方式。在一般 EOI 方式的基础上，当中断服务程序结束给 8259A 发出 EOI 命令的同时，将当前结束的中断级别也传输给 8259A，这就称为特殊 EOI 方式。在这种情况下，8259A 将 ISR 寄存器中指定级别的相应置“1”位清“0”。这种方式适合任何情况。

③ 自动 EOI 方式。CPU 进入中断响应总线周期的第二个中断响应信号 $\overline{\text{INTA}}$ 结束时，自动将 ISR 寄存器相应置“1”的位清“0”。中断结束时，不需要向 8259A 送 EOI 命令。这是一种最简单的结束方式，但是存在一个明显的缺点：任何一级中断在执行中断服务程序期间，在 8259A 中没有留下任何标志，如果在此过程中出现新的中断请求，则只要 CPU 允许中断，不管出现的中断级别如何，都将打断正在执行的中断服务而被优先执行。这显然是不合理的。因此，自动 EOI 方式只能用在一些以预定速率发生中断，且不会发生同级中断互相打断或低级中断打断高级中断的情况下。

（2）自动循环情况

在这种情况也可采用 3 种中断结束方式，与完全嵌套的情况相似，但结束后的优先权处理有所不同。

① 一般 EOI 方式。当任何一级中断服务处理完毕，给 8259A 送一个一般 EOI 结束命令后，8259A 将 ISR 寄存器中级别最高的置“1”的位清“0”，同时赋给它最低优先级，将最高优先级赋给原来比它低一级的中断请求，其他中断请求的优先级别以循环方式类推。

② 特殊 EOI 方式，主要用在自动循环优先权管理方式下又有嵌套的情况。在一般 EOI 方式下，结束一个中断就将它置为最低优先级，若在服务程序中安排一条优先权置位指令，使得优先权次序发生了变化，就必须使用特殊 EOI 方式；在向 8259A 发结束命令的同时，将其中断优先级别也传输给 8259A。这样，8259A 可以根据用户要求将最低优先级别传送给指定的中断源，其余中断源的优先级别按自动循环方式类推。

③ 自动 EOI 方式。在 CPU 响应中断的第二个 $\overline{\text{INTA}}$ 结束时，自动将 ISR 寄存器中相应的位清“0”，并立即按自动循环方式改变各级中断的优先级别。

（3）特殊完全嵌套情况

这种情况是因为 8259A 有级联，因而 CPU 应发出两个中断结束命令 EOI。一个送主 8259A，用来将其主 8259A 的 ISR 寄存器相应的位清“0”；另一个送从 8259A，用来将其从 8259A 中的 ISR 寄存器相应的位清“0”。

9.6.3 8259A 的编程与应用

8259A 是一个可编程器件，在正常操作之前可用程序规定其优先权管理方式、中断结束方式，如果有级联，还要规定级联的引入等，这段程序一般称为对可编程器件的初始化。8259A 的初始化分两部分。

第一部分称为预置命令字，预置命令有 4 个：ICW_1～ICW_4。不是任何情况下都需要设置这 4 个预置命令，可根据 8259A 的使用情况而定。预置命令字是要按规定顺序写入 8259A 中的，其顺序可用图 9-21 所示的流程来表示。

预置命令字可以完成以下功能：

① 按 8259A 的选用情况规定是单一式还是主从式。

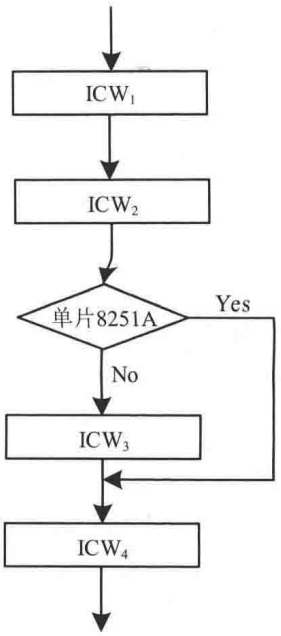


图 9-21　预置命令流程

② 可设置 8 个中断请求设备的类型号。

③ 若为主从式，则规定主 8259A 中每个 IR 端是否带从 8259A，从 8259A 则要规定从主 8259A 的哪个端引入。

④ 完成中断管理方式的设定。

第二部分称为操作命令字，8259A 通过预置命令字初始化后，将自动进入操作模式。在操作过程中可通过操作命令字来定义 8259A 的操作方式，并允许重置操作命令字改变操作方式。操作命令字有 3 个：OCW_1～OCW_3，不按顺序写入。

8259A 占两个端口地址，用 A_0 的两种状态选择，而命令字有多个，每个命令字写入 8259A 时应存入相应的寄存器中。为此，对某些命令字设置了标志，以表明其命令字进入端口地址后再正确存入相应的寄存器中，有些命令字无法给出标志或特征，仅是按写入顺序决定存入某寄存器。例如，8259A 的预置命令字 ICW_1 写入 A_0=0 的端口，而 ICW_2～ICW_4 写入 A_0=1 的端口，因为它们必须按顺序写入，所以以顺序为标志存入各自的寄存器中。

下面将具体讨论 8259A 的命令字。

1．预置命令字 ICW

ICW_1 的格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	×	×	×	1	LTIM	×	SNGL	1

其中，“×”表示为随意态，这些位没有定义（可为 0 或 1）。在 8086 系统中使用 8259A，则 D_0=1，而 D_1 和 D_3 位是用户可选择的，D_1=1，表明系统中仅用单片 8259A，D_1=0，则 8259A 有级联。D_3=1，则 8259A 的中断请求输入信号为高电平有效的电平触发方式；D_3=0，则 8259A 的中断请求输入信号为上升沿有效的边缘触发方式。

ICW_2 的格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	T_7	T_6	T_5	T_4	T_3	IR 编码		

ICW_2 预置命令用来提供中断类型号，中断类型号的高 5 位 D_7～D_3 由用户指定，低 3 位 D_2～D_0 由 8259A 的中断请求输入端 IR_7～IR_0 的编码决定，如 IR0 的编码为 000，IR_1 的编码为 001，则 IR_7 的编码为 111。由此可见，ICW_2 为 8259A 芯片提供的中断类型码是连续的。

ICW_3 只有在 8259A 有级联的情况下使用，主片与从片的 ICW_3 格式不同。主片的 ICW_3 表明主 8259A 的哪个 IR 端接有从 8259A。从片的 ICW_3 表明它接在主 8259A 的哪个 IR 端。主 8259A 的 ICW_2 格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

每位对应一个相应的 IR 端，D_0 对应 IR_0，…，D_7 对应 IR_7。若某个 IR 上接有从 8259A，则该位为 1，否则为 0。

从 8259A 的 ICW_3 格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	×	×	×	×	×	ID_2	ID_1	ID_0

“×”表示为随意态，ID_2～ID_0为该从 8259A 接入主 8259A 的 IR 端的编码。例如，从 8259A 接在主 8259A 的 IR_5 端，则 $ID_2ID_1ID_0$=101。

若系统为 8086 CPU，就必须设置 ICW_4 预置命令，其格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	SFNM	BUF	M/S	AEOI	1

D_1 位 AEOI 用来定义该 8259A 是否采用自动中断结束方式。D_1=1，为自动 EOI，D_1=0，为非自动 EOI。D_2 位和 D3 位的选用相关，这两位的关系如表 9-1 所示，用来指定缓冲方式的选定。

表 9-1　BUF、M/S 和 $\overline{SP}$ / $\overline{EN}$ 的定义

BUF 位		M/S 位		$\overline{SP}$ / $\overline{EN}$ 引脚		
0	非缓冲方式	无意义		$\overline{SP}$ 有效（输入信号）	$\overline{SP}$=1	主 8259A
					$\overline{SP}$=0	从 8259A
1	缓冲方式	1	主 8259A	$\overline{EN}$ 有效（输出信号）	$\overline{EN}$=1	CPU→8259A
		0	从 8259A		$\overline{EN}$=0	8259A→CPU

ICW_4 的 D_3 位将指定 8086 的数据总线是否进行缓冲。D_3=1 时，指定为缓冲方式，$\overline{SP}$ / $\overline{EN}$ 端是输出端，同时主/从 8259A 的识别在 D_2 位 M/S 上进行；D_3=0 时，指定为非缓冲方式，$\overline{SP}$ / $\overline{EN}$ 为输入端，此时主/从 8259A 的识别在 $\overline{SP}$ / $\overline{EN}$ 引脚上进行。D_4 位是 SFNM 专为 8259A 有级联时定义的，用来指定级联方式下是否采用特殊完全嵌套方式来管理中断，D_4=1，为特殊完全嵌套方式，D_4=0，为非特殊完全嵌套方式。

8259A 在任何情况下，从 A_0=0 的端口中接收到一个 D_4 位为 1 的命令就是 ICW_1 预置命令字，紧接着的命令字 ICW_2～ICW_4 进入初始化状态，可接收来自 IR 端的中断请求，并准备接收 CPU 写入的操作命令字 OCW。

2．操作命令字 OCW

8259A 可接收的操作命令字有 3 个：OCW_1、OCW_2 和 OCW_3。这些命令字不像 ICW，可以随时动态地写入，不需按顺序写入，因为这些字均有各自的标志来引导进入相应的寄存器。OCW_1 必须写入 A_0=1 的端口，OCW2、OCW3 均写入 A_0=0 的端口。OCW_2 的标志 D_4D_3 为 00，而 OCW_3 的标志 D_4D_3 为 01。

OCW_1 用来实现屏蔽功能，OCW1 的内容被置入中断屏蔽寄存器 IMR 中，IMR 中相应为“1”的位禁止该位对应的 IR 端的中断请求进入。OCW_1 的格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

OCW_2 用于控制中断结束、优先权循环等操作，与这些操作有关的命令和控制大都以组合格式使用 OCW_2。OCW_2 的格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	R	SL	EOI	0	0	L_2	L_1	L_0

D_7 为 R 位，用于优先权控制，R=1 为循环优先权，R=0 为固定优先权（IR_0 最高，IR_7 最

低)。D_6为 SL 位，在中断结束时若使用特殊 EOI 方式，总是将当前结束中断的级别同时通知 8259A。当 SL=1 时，则由 L_2～L_0的编码指定相应的 IR 端为当前结束中断。SL=0，则 L_2～L_0三位无效。D_5为 EOI 位，用来在非自动 EOI 方式下回送给 8259A 的中断结束命令，当 ICW_4预置命令中 AEOI=0 表明中断结束是非自动的，要将 OCW_2的 EOI 置 1，以便使中断服务寄存器 ISR 中最高优先权位或指定优先权位清零。AEOI=0，则不起作用。OCW_2的组合控制功能如表 9-2 所示。

表 9-2　OCW_2的组合控制功能

R	SL	EOI	功　能	R	SL	EOI	功　能
0	0	1	一般 EOI 命令	0	0	0	清除自动循环 AEOI 方式
0	1	1	特殊 EOI 命令	1	1	1	自动循环特殊 EOI 命令
1	0	1	自动循环的一般 EOI 命令	1	1	0	置位优先权命令
1	0	0	设置自动循环 AEOI 方式	0	1	0	无效

OCW_3主要用来控制 8259A 的中断屏蔽和读取寄存器的状态，其格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	×	ESMM	SMM	0	1	P	RR	RIS

D_1和 D_0两位组合使用，D_1的 RR 表示读寄存器的命令：RR=1，读寄存器；RR=0，不读。D_0的 RIS 在 RR=1 的情况下，指出要读的寄存器名：RIS=1，读取中断服务寄存器内容；RIS=0，则读取中断寄存器 IRR 的内容。若 RR 位为 0，无论 RIS 位为何态均无意义。

D_6和 D_5两位组合使用来设置或取消特殊屏蔽方式。ESMM=1，表示允许设置或取消特殊屏蔽，此时若 SMM=1，则允许设置，SMM=0 则允许取消。ESMM=0 时，则无论 SMM 为何态均不能设置也不能取消特殊屏蔽。当需要设置低级中断打断高级中断的特殊屏蔽时，将这两位置成“11”。

D_2位称为查询位 P，将 P 置“1”后，表示向 8259A 发查询命令，查询当前是否有中断请求正在被处理，如果有，则给出当前处理的最高优先级是哪一级，可查询的中断状态字格式为：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	IR	×	×	×	×	L_2	L_1	L_0
						IR 编码		

在状态字中，D_2～D_0位给出当前处理的具有最高优先级的 IR 编码，D_7位有两种情况：D_7=0，表明无中断请求；D_7=1，则有中断请求。

例如，某系统中 8259A 采用完全嵌套方式，当前 ISR 寄存器中第 2 位和第 5 位置“1”，即第 2 级中断打断了第 5 级中断，当前执行的应该是第 2 级中断服务程序，若在第 2 级的中断服务程序中将 OCW_3的 P 位置“1”后，执行指令“IN　AL, PORT”，则 AL 中的内容如下：

1	×	×	×	×	0	1	0

3．8259A 的应用举例

下面给出关于 8259A 一个简单而又非常典型的应用。在某 8086 最小方式系统中接有一片 8259A，某外设中断请求从 IR_7引入，8259A 的端口地址由图 9-20 给出。试写出 8259A 的初始化程序。要求：

❖ 中断请求信号触发方式为上升沿触发。

- ❖ 单片工作方式。
- ❖ 中断类型号分布为 0C0H～0C7H。
- ❖ 优先级方式为完全嵌套方式。
- ❖ 中断结束方式为一般 EOI 方式。
- ❖ 屏蔽 IR_0～IR_6 端的中断请求，只允许 IR_7 端的中断请求。

8259A 端口地址电路如图 9-22 所示。8259A 具有两个端口地址，由 CPU 的地址线 A_1 控制，其端口地址的高位由译码器的输出端 $\overline{Y_1}$ 提供，组合逻辑电路可使其 8259A 为偶地址，因而 8259A 的命令和类型号等可由 8086 CPU 的低 8 位数据线传输，端口地址为 84H 和 86H。从 IR_7 引入的中断类型号为 0C7H，8086 CPU 获得中断类型号并乘 4，到中断服务入口地址表中找到相应的中断服务程序地址，然后转入中断服务。初始化程序包括对 8259A 的设置，以及将中断服务程序首地址填入中断矢量表中。

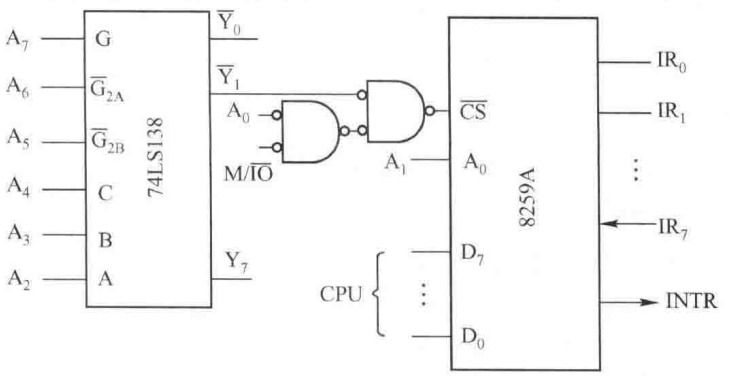


图 9-22　8259A 端口地址形成电路

```
INTRRUP   SEGMENT  AT  0
          ORG      0C7H*4
          DW       OFFSET  INTR_IR7          ; 填写中断矢量表
          DW       SEG  INTR_IR7
          ......
MAIN      SEGMENT
          ......
          CLI
          MOV      AL,00010011B              ; 写 ICW1，表示上升沿触发及单片工作方式，并设 ICW4
          OUT      84H, AL
          MOV      AL, 0C0H                  ; 写 ICW2，表示中断矢量基值为 0C0H
          OUT      86H, AL
          MOV      AL, 00000001B             ; 写 ICW4，表示完全嵌套方式非缓冲，一般 EOI 方式
          OUT      86H, AL
          MOV      AL, 01111111B             ; 写 OCW1，中断屏蔽字，仅允许 IR7 中断
          OUT      86H, AL
          STI
          ......
```

由于 ICW_4 所设定的优先级方式为完全嵌套方式，中断结束管理方式为一般 EOI 方式，因此在中断服务子程序 INTR_IR_7 结束时，要给 8259A 传输一个 EOI 命令，使 8259A 将 ISR 寄存器中级别最高的置“1”位清“0”。即执行如下两条指令：

```
          MOV      AL, 20H                   ; 写 OCW2
          OUT      84H, AL
```

有关中断更详细的应用，请读者参考本章 9.7.2 节。

9.7 IBM PC 硬件中断

在 PC 中，CPU 的一个基本任务是响应中断。例如，系统时钟、键盘以及磁盘控制器等所有的硬件在不同的时间均产生中断。由于每个硬件中断在 ROM BIOS 或在 DOS 中具有对应的中断处理程序，所以 CPU 可以识别和响应每个中断。PC/XT 机中有一片 8259A 中断控制芯片，可以管理 8 级中断，I/O 端口地址为 20H、21H，中断输入 IR_0～IR_7 分配的中断类型号为 08H～0FH。PC/AT 机中有两片 8259A 中断控制芯片，主片的端口地址和中断类型号的分配与 PC/XT 机相同。增加的 8259A 为从片，端口地址为 0A0H、0A1H，中断输入 IR_8～IR_{15}（从片的 8 根中断引线）分配的中断类型号为 70H～77H。

9.7.1 中断设置

在计算机中，中断优先级被设置成完全嵌套方式下的中断优先级。在多级的 8259A 中断系统中，从片 8259A 连接到主 8259A 的哪一端上，它就具有哪一端的中断优先权级别。由于计算机中从片 8259A 接在主片 8259A 的 IR_2 上，所以各中断源的优先级排列如下：

主片：IR_0，IR_1

从片：IR_0，IR_1，IR_2，IR_3，IR_4，IR_5，IR_6，IR_7

主片：IR_3，IR_4，IR_5，IR_6，IR_7

最高优先级中断（IR_0）为定时节拍中断。定时器以每秒产生 18.2 次的速率向 CPU 发出中断请求。其中断服务包含两项内容：一是计时，二是管理软磁盘驱动器的启用时间。

次高级优先权中断（IR_1）为键盘中断。键盘接口接收从键盘发来的串行数据，在移位寄存器中获得按键的扫描码。串行数据的最后一位进入移位寄存器后，就向 CPU 发出中断请求。键盘在每次按键过程中发送两个码：按下时发送的是“通码”，放开时发送的是“断码”。断码的特征是最高位为 1。键盘中断处理程序从接口中读入键盘扫描码，除了 Shift、Ctrl、Alt 键，仅对“通码”进行处理，对“断码”不处理。普通字符键的扫描码转换成相应的 ASCII 值，存入键盘输入缓冲区。一些不能用 ASCII 表示的键，如←、↑、↓、→等键，则以扩展码的形式存入键盘输入缓冲区。其他控制键则存入键盘状态单元。

计算机硬件中断的引入及中断源如表 9-3 所示。

表 9-3 计算机中断源及引入表

中断引入引脚		中 断 源	中断引入引脚		中 断 源
主片	IR_0	计数器 0 通道输出	从片	IR_0	定时时钟
	IR_1	键盘中断		IR_1	IRQ_9 引入
	IR_2	从片 8259A		IR_2	IRQ_{10} 引入
	IR_3	网络通信		IR_3	IRQ_{11} 引入
	IR_4	串行通信		IR_4	IRQ_{12} 引入
	IR_5	硬盘控制		IR_5	IRQ_{13} 引入
	IR_6	软盘控制		IR_6	IRQ_{14} 引入
	IR_7	打印机控制		IR_7	IRQ_{15} 引入

9.7.2 计算机中断资源的使用

在计算机上设计一个接口电路上免不了要使用中断传输方式。计算机提供了 7 个中断类型号，使用起来比较方便。硬件上只要把外设电路的中断请求信号线或状态线接到从片 8259A 的 IR_n（n=1～7）即可。软件上做两项操作：① 要对 8259A 的中断屏蔽寄存器进行设置，以开启响应中断请求信号的通路；② 当中断服务子程序完毕时，对 8259A 发中断结束命令。

1．对中断屏蔽寄存器的操作

如前所述，外设发出中断请求到 CPU 响应中断，有两个控制条件是起决定性作用的，一是该外设的中断请求是否屏蔽，另一个是 CPU 是否允许响应中断。这两个条件分别由 8259A 的中断屏蔽寄存器（IMR）和 8086 CPU 的标志寄存器（FR）中的中断允许位 IF 控制。对于后者，可用开中断指令和关中断指令来实现。所以下面重点讨论 8259A 的中断屏蔽寄存器。

主片 8259A 中断屏蔽寄存器的 I/O 端口地址是 21H，它的 8 位对应控制 8 个外部设备（见图 9-17），通过设置这个寄存器的某位为 0 或为 1 来允许或禁止某外部设备的中断。某位为 0 表示允许某种外设中断请求，为 1 表示某种外设的中断请求被屏蔽（禁止）。

例如，只允许键盘中断，可设置如下中断屏蔽字：

```
MOV     AL, 11111101B
OUT     21H, AL
```

如果系统重新增设键盘中断，则可用下列指令实现。

```
IN      AL, 21H
AND     AL, 11111101B
OUT     21H, AL
```

在编写中断程序时，应在主程序的初始化部分设置中断屏蔽寄存器，以确定中断方式工作的外部设备。如果中断是从片 8259A 的中断输入线产生的，则要同时设置主片和从片的中断屏蔽寄存器。从片 8259A 中断屏蔽寄存器的 I/O 端口地址是 A1H。

2．中断结束命令

在一次中断处理结束前，还应给 8259A 可编程中断控制器的中断命令寄存器发出中断结束命令 EOI，可以采用一般的 EOI 方式，中断命令寄存器的 I/O 端口地址为 20H。结束外中断用下面的指令：

```
MOV     AL, 20H
OUT     20H, AL
```

同样，如果中断是从片 8259A 的中断输入线产生的，就要同时对主片和从片的中断命令寄存器发 EOI 命令。从片 8259A 中断命令寄存器的 I/O 端口地址是 A0H。

9.7.3 中断举例

我们对中断是如何工作的以及计算机的中断系统已有了一定的了解，为了让读者知道中断传输方式的设计过程，现在给一个中断应用的例子。

中断传输方式的设计过程如下。

① 硬件方面：设定硬件的中断类型号。

② 软件方面：关中断；填写中断入口地址表；设置 8259A 及其他编程芯片；开中断；编写中断服务子程序。

图 9-23 是计算机的一个接口电路，用中断传输方式使 8255A 的 A 口的发光二极管依次发光，要求每个二极管在一个循环中发光 1 秒。设 8255A 端口地址为 80H～86H，8253 端口地址为 88H～8EH。

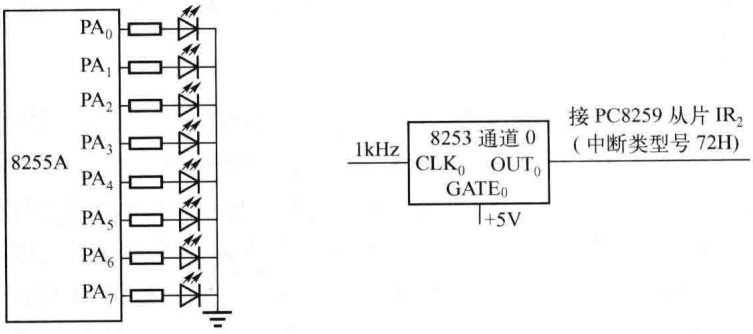


图 9-23　中断传输举例

硬件上这个接口电路有两部分，8255A 为数据接口，数据通过 A 口的发光二极管显示。8253 为定时器接口，其通道 0 的作用是产生 1 秒定时信号作为中断请求信号，在中断服务程序中，CPU 对 A 口的数据进行刷新，使发光二极管依次发光。OUT_0 接中断电路的从片 8259A 的 IR_2，即 ISA 接口的 IRQ_{10}，所以中断类型号为 72H。

8255A 的 A 口工作方式可选方式 0 的输出，而 8253 通道 0 的工作方式可定为方式 0，即计数结束产生中断信号。

8255A 的方式控制字为 10001011B，即 A 口输出、B 口输入、C 口输入。

8253 通道 0 的方式控制字为 00110001B，即工作方式为方式 0，BCD 码计数。

程序分主程序和中断服务子程序。主程序包括对 8255A、8253 和计算机中 8259A 的初始化设置，以及中断入口地址表的填写。相关程序如下：

```
DATA    SEGMENT
SHIFT_X DB      01H
DATA    ENDS
CODE    SEGMENT
        ASSUME  DS:DATA,CS:CODE
        ...
        CLI                                  ; 关中断
        CLD                                  ; 填写中断入口地址表
        MOV     AX, 0
        MOV     ES, AX
        MOV     DI, 4*72H
        MOV     AX, OFFSET INTPROC
        STOSW
        MOV     AX, SEG INTPROC
        STOSW
        IN      AL, 21H                      ; 设置 8259A
        AND     AL, 11111011B                ; 取主 8259A 的中断屏蔽字
```

```
        OUT     21H, AL                         ; 允许主 8259A 的 IR2 中断
        IN      AL, 0A1H                        ; 取从 8259A 的中断屏蔽字
        AND     AL, 11111011B                   ; 允许从 8259A 的 IR2 中断
        OUT     0A1H, AL
        ......
        MOV     AL, 10001011B                   ; 设置 8255A
        OUT     86H, AL
        MOV     AL, 0
        OUT     80H, AL
        MOV     AL, 00110001B                   ; 设置 8253 通道 0
        OUT     8EH, A
        MOV     AL, 00
        OUT     88H, AL
        MOV     AL, 10H
        OUT     88H, AL
        STI                                     ; 开中断
        …                                       ; 执行其他程序
```

中断服务子程序主要是对 8255A 的 A 口进行操作，使变量 SHIFT_X 的内容左移，重新设置 8253 通道 0，为下一次中断信号产生做准备，并对 8259A 传输一个 EOI 命令，表示中断程序完成。程序如下：

```
INTPROC PROC    FAR                             ; 中断服务子程序
        PUSH    AX                              ; 保护现场
        PUSHF
        MOV     AL, SHIFT_X                     ; 对 8255 的 A 口送数
        OUT     80H, AL
        ROL     AL, 1                           ; 变量 SHIFT_X 左移
        MOV     SHIFT_X, AL                     ; 准备下一次数据
        MOV     AL, 00110001B                   ; 重新设置 8253
        OUT     8EH, AL
        MOV     AL, 00
        OUT     88H, AL
        MOV     AL, 10H
        OUT     88H, AL

        MOV     AL, 20H                         ; 传输 EOI 命令给 8259A
        OUT     0A0H, AL                        ; 向从 8259A 发 EOI 命令
        OUT     20H, AL                         ; 向主 8259A 发 EOI 命令

        POPF                                    ; 恢复现场
        POP     AX
        IRET                                    ; 中断返回
INTPROC ENDP
```

当主程序完成初始化工作后，8253 通道 0 的 OUT_0 将在 1 s 后产生中断信号，使从片的 IR_2 输入有效，8259A 便输出中断请求信号给 CPU。在 CPU 响应中断时，8259A 从片提供中断类型号 72H 给 CPU，CPU 通过它对中断入口地址表的查表找到中断服务程序的入口地址，实现了从主程序向中断服务子程序的转移。中断服务子程序中，程序对 A 口操作，使发光二极管

闪亮移位。

习 题 9

1．简述一个中断过程并画出示意图。

2．简述中断过程和调子程序过程的异同。

3．中断响应有哪些条件？为什么 CPU 响应中断后立即关中断？

4．什么叫中断源，中断源如何确定？

5．在有多个中断源申请中断时，有几种方法确定它们的优先级别？

6．何谓矢量中断？何谓中断矢量？

7．8086 CPU 是如何获取中断矢量的？

8．写出分配给下列中断类型号在中断向量表中的物理地址。

（1） INT 12H （2） INT 8

9．假设中断类型 9 的中断处理程序的首地址为 INT_PROC09，试写出主程序中为建立这个中断矢量表而编制的程序段。

10．编写指令序列，使类型 1CH 的中断向量指向中断处理程序 SHOW_CLOCK。

11．设备 D1、D2、D3、D4、D5 是按优先级别排队的，设备 D1 具有最高优先级，若设备按 D5、D3、D1、D4、D2 的次序申请中断，CPU 以什么样的次序为各设备服务（假设所有的中断处理程序开始后就有 STI 命令，并在中断返回之前发出结束命令 EOI）？

12．用 DOS 调用方法，建立中断类型号 49H 的中断向量，设中断服务程序地址为 INT_PROC49。

13．设系统中有主、从 8259A 芯片共 3 片，并设从片 1 连接在主片 IR_1 端，从片 2 连接在主片的 IR_2 端，若它们工作在完全嵌套方式下，试指出该系统的中断优先级排队。

第 10 章　DAC 和 ADC 及其应用

本章导读

- 从物理信号到电信号的转换
- D/A 转换器及其接口技术
- A/D 转换器及其接口技术
- 微机在辅助科学实验中的应用
- 微机在生物科学中的应用
- 微机在控制中的应用
- 微机在临床医疗仪器中的应用

微机处理的是数字量，而实际上外界事物大多是模拟量，如温度、压力、流量、光信号、煤气浓度、速度、水位、距离等。这些都是非电的物理量，它们必须经过一个传感器转变成电模拟信号，然后进一步转化为数字量才能为微机处理。这种把模拟量转换为数字量的转换过程称为模数转换，即 A/D 转换，如图 10-1(a)所示。由微机加工处理后的数字量，往往也需要转换为模拟量，因为许多执行部件和某些显示器件（如示波器）需要由模拟量电信号来启动。将数字量转换为电模拟量的过程称为数模转换，即 D/A 转换，如图 10-1(b)所示。

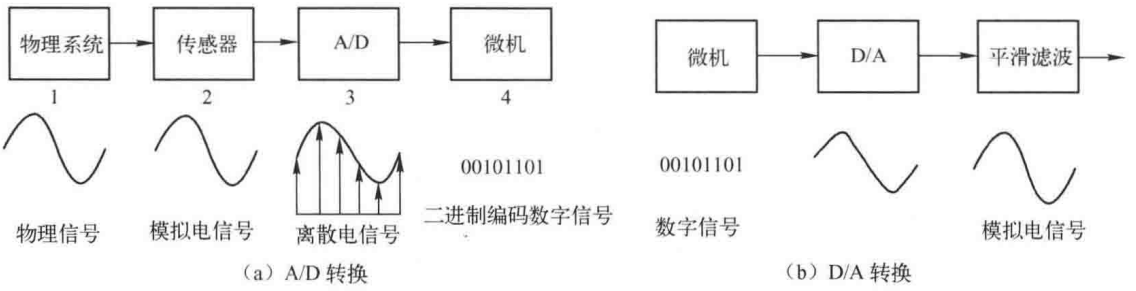


图 10-1　A/D 和 D/A 转换

A/D 和 D/A 转换器是把微机应用领域扩展到检测和过程控制的必要装置，是把计算机和生产过程、科学实验过程联系起来的重要桥梁。图 10-2 给出了 ADC 和 DAC 在微机检测和控制系统中的应用。

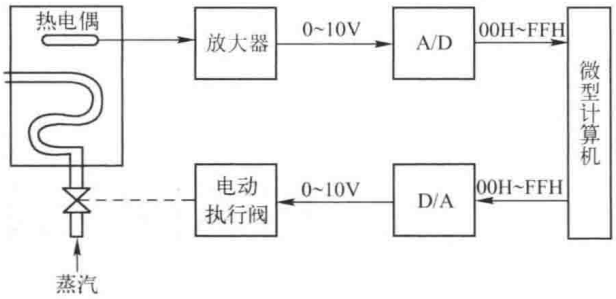


图 10-2　ADC 和 DAC 在计算机检测和控制系统中的应用

10.1　从物理信号到电信号的转换

A/D 转换器的功能是将模拟的电信号转换成数字信号，所以，将物理量转换成数字量之前应先将物理量转换成电模拟量，这种转换是靠传感器完成的。

传感器是一种能将物理量、化学量、生物量等转换成电信号的器件。输出信号有不同形式，如电压、电流、频率、脉冲等，能满足信息传输、处理、记录、显示、控制要求，是自动检测系统和自动控制系统中不可缺少的元件。如果把计算机比作大脑，那么传感器相当于五官，传感器能正确感受被测量并转换成相应输出量，对系统的质量起决定性作用。自动化程度越高，系统对传感器要求越高。由于物理量的多样性使得传感器的种类繁多，如温度传感器、压力传感器、光电传感器、气敏传感器等。

1．温度传感器

热电偶是一种大量使用的温度传感器，它是利用热电势效应来工作的，室温下的典型输出电压为毫伏数量级。温度测量范围与热电偶的材料有关，常用的有镍铝–镍硅热电偶和铂锭–铂热电偶。热电偶的热电势–温度曲线一般是非线性的，需要采取措施进行非线性校正。

另一种温度传感器为热敏电阻，它是一种半导体新型感温元件，具有负的电阻温度系数，当温度升高时，其电阻值减小，在使用热敏电阻作为温度传感器时，将温度的变化反映在电阻值的变化中，从而改变电流或电压值。

2．湿度传感器

湿度传感器大多利用湿度变化引起其电阻值或电容量变化原理制成，即将湿度变化转换成电量变化。

热敏电阻湿度传感器利用潮湿空气和干燥空气的热传导之差来测定湿度。氯化锂湿度传感器利用氯化锂在吸收水分后，其电阻值发生变化的原理来测量湿度。高分子湿度传感器利用导电性高分子对蒸汽的物理吸附作用引起电导率变化的特性。

3．气敏传感器

气敏传感器是利用半导体与某种气体接触时电阻及功率函数变化的效应来检测气体的成分或浓度的传感器，可用于家用液化气泄漏报警、城市煤气、煤气爆炸浓度、一氧化碳中毒危险浓度报警等。有的敏感元件对酒精特别敏感，可以用于酒后驾车报警控制。

4．压电式和压阻式传感器

某些电解质（如石英晶体，压电陶瓷）在沿一定方向上受到外力的作用而变形时，内部会产生极化现象，同时在其表面上产生电荷，当外力去掉后又重新回到不带电的状态，从而可以将机械能转变成电能。因此，可把压电式传感器看成一个静电荷发生器，也就是一个电容。这些介质可做成压电式传感器。

由于固体物理的发展，固体的各种效应已逐渐被人们所发现。固体受到作用力后，电阻率（或电阻）就要发生变化，这种效应称为压阻式效应。它可做成压阻式传感器。

压电式或压阻式传感器可测量压力、加速度、载荷等。压电式传感器可以测量频率从几赫兹至几万赫兹的动态压力，如内燃机汽缸、油管、进排气管压力、枪炮的膛压、航空发动机燃烧室压力等。

5．光纤传感器

光纤传感器是20世纪70年代迅速发展起来的一种新型传感器，具有灵敏度高、电绝缘性能好、抗电磁干扰、耐腐蚀、耐高温、体积小、质量轻等优点，广泛用于位移、速度、加速度、压力、温度、液位、流量、水声、电流、磁场、放射性射线等物理量的测量。

功能型光纤传感器不仅起传光作用，还是敏感元件。它利用光纤本身的传输特性受被测物理量作用而发生变化，使光纤中波导光的属性（光强、相位、偏振态、波长等）被调制。因此，功能型光纤传感器又分为光强调制型、相位调制型、偏振态调制型和波长调制型。

相位调制型光纤传感器用来测量压力、温度。因为光纤受力或温度的变化时，光纤的长度、直径、折射率会发生变化，从而引起传播光的相位角变化。如果利用一个光电探测器将光的相位变化转换成电信号，其大小变化就可以将压力或温度等物理量转换成电的模拟量。

在流体流动的管导中横贯一根多模光纤。当液体流过光纤时，在液流的下游会产生有规则的涡流。这种涡流在光纤的两侧交替地离开，使光纤受到交变的作用力，光纤就会产生周期性振动。光纤的振动频率与流体的速度和光纤的直径有关，在光纤直径一定时，近似正比于流速。光纤流量传感器原理图如图10-3所示。

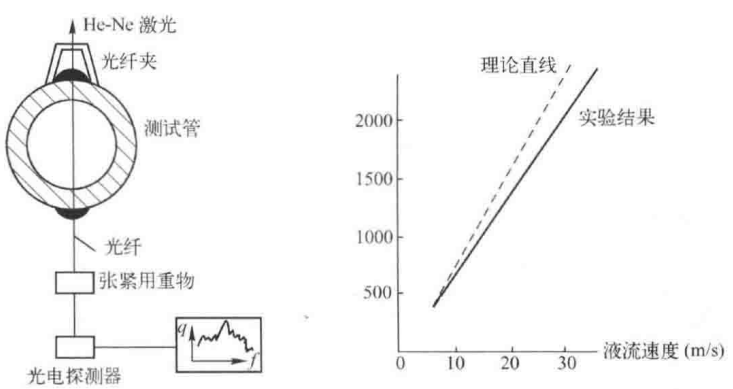


图10-3　光纤流量传感器原理

6．位移－数字转换器

前面介绍的几种传感器可以将物理量转换成连续变化的电量，这些电量往往要经过放大、滤波、整形后才能进行数字化，以便计算机处理。连续变化的电量较容易引进干扰，会降低测量精度并使电路复杂化。某些传感器可将物理量直接转换成数字量或电脉冲，如码盘、光栅等。

（1）脉冲盘式角度－数字转换器

脉冲盘式角度－数字转换器的结构如图 10-4 所示。在一个圆盘的边缘上开有相等角距的缝隙，开缝圆盘两边分别安装光源和光敏元件。当圆盘随工作轴一起转动时，每转过一个缝隙，就发生一次光线明暗变化，经过光敏元件，产生一次电信号的变化。将这电信号整形放大，可得到一定幅度和功率的电脉冲输出信号，脉冲数就等于转过的缝隙数。上述脉冲信号可由计数器计数，计数值就能反映圆盘转过的角度。

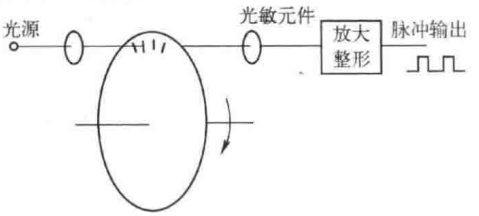
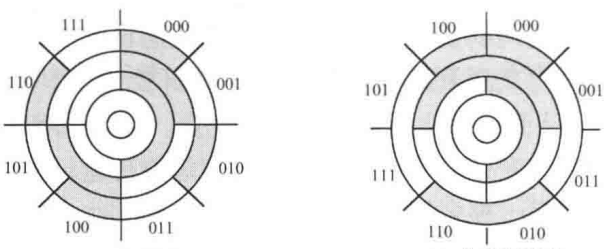


图 10-4　脉冲盘式角度－数字转换器的结构

（2）码盘式角度－数字转换器

脉冲盘式输出的是与角度对应的脉冲个数，要经过计数器才能进行数值编码。码盘式则是按角度直接进行编码的转换器，通常被安装在检测轴上。编码的方式有二进制和格雷码制，按结构分为接触式和光电式等。图 10-5 为接触式 3 位二进制和 3 位格雷码的码盘。



（a）二进制码盘　　（b）格雷码码盘

图 10-5　码盘式传感器

码盘涂黑部分是导电区，所有导电部分连在一起接高电位，空白部分代表绝缘区。每个码道上都有一个电刷，电刷经电阻接地。当码盘与轴一起转动时，电刷上将出现相应电位，对应一定的数码。二进制编码的码盘在实际应用中对码盘的制作和电刷的安装要求十分严格，否则会出错。采用格雷码（又叫循环码）对安装和制作要求较低。格雷码的特点是相邻两个数码间只有一位是变化的，即使产生错误也只是最低一位出错。

十进制数、二进制数与格雷码之间的关系如表 10-1 所示。

表 10-1　十进制数、二进制数与格雷码之间的关系

十进制数	二进制数	格 雷 码	十进制数	二进制数	格 雷 码
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

（3）光电码盘式角度－数字转换器

光电码盘是目前用得较多的一种，码盘用透明及不透明区按一定编码构成。码盘上码道的条数就是数码的位数。对应每条码道有一个光电元件，当码盘处于不同角度时，光电转换器的输出就呈现出不同的数码。它的优点是没有接触磨损，因而允许转速高。光电码盘式角度－数字转换器的结构如图 10-6 所示。

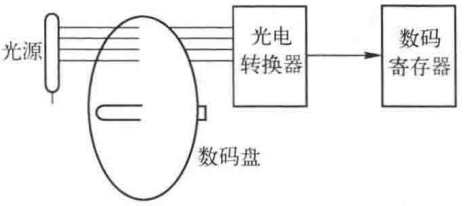


图 10-6　光电码盘式角度–数字转换器的结构

10.2　DAC 及其接口技术

DAC 的功能是把二进制数字量电信号转换为与其数值成正比的模拟量电信号，其中一个最重要的参数就是分辨率。分辨率是指输入数字量发生单位数码变化时，对应输出模拟量（电压或电流）的变化量。因此，分辨率反映了 DAC 分辨输出最小模拟电压的能力。规定分辨率用输出模拟电压的最大值 U_{omax} 与最大输入数 2^N-1 之比进行衡量。

例如，若 $U_{omax}=10V$，则 10 位 DAC 的分辨率为 0.009 775 V，而 8 位 DAC 的分辨率为 0.039 215 V。可见，输入数字量位数越多，分辨模拟电压的能力越强，分辨率越高。在实际使用中，表示分辨率高低更常用的方法是采用输入数字量的位数或最大输入码的个数表示。例如，8 位二进制 DAC 的分辨率为 8 位。

DAC 可以视为微机的一种输出设备，实现 DAC 和微机接口技术的关键是数据锁存问题。当 CPU 向 DAC 输出一个数据时，这个数据在数据总线上只持续很短的时间，必须有数据锁存器为 DAC 保持住这个数据，才能得到持续稳定的模拟量输出。

有些 DAC 芯片本身带有锁存器，有些则不带锁存器，一些并口芯片如 8212、74LS273 及可编程并行 I/O 接口芯片 8255A 均可作为 D/A 转换的锁存器。

10.2.1　AD558（并行 8 位 DAC）

DAC AD558 由内部锁存器、利用 R-2R 的 T 型解码网络和晶体管开关组成，仅需+5 V 电压供电，输出模拟电压范围为 0～2.56 V。图 10-7 是 AD558 的内部结构框图，图 10-8 是 AD558 与计算机的连接图。

AD558 的锁存器通过 $\overline{CS}$ 和 $\overline{CE}$ 两信号控制。当 CPU 执行一条 OUT 指令时，$\overline{BIOW}$ 信号为低电平，地址 30BH 为计算机中译码器的输出，该输出信号使 $\overline{CS}$ 为低电平。因此，CPU 的输出数据被锁存于 AD558 中，并通过 T 形解码网络转换成模拟电信号从 V_{out} 端输出。AD558 的分辨率约为 10 mV。下面的程序是用 AD558 产生锯齿波模拟信号。

```
CODE    SEGMENT
        ASSUME  CS:CODE
```

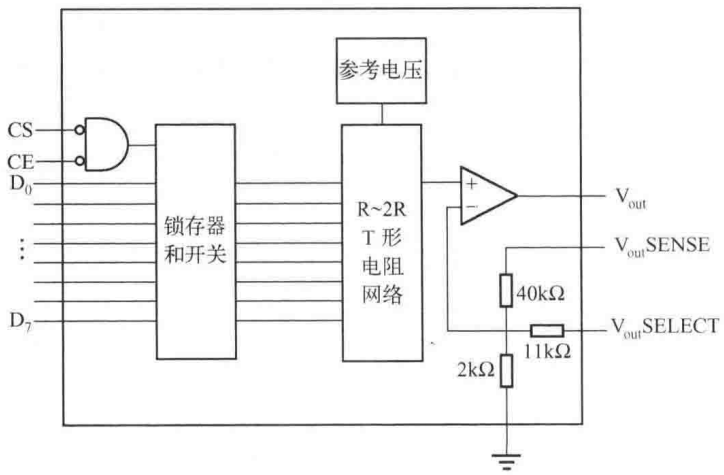


图 10-7　D/A 转换芯片 AD558 内部结构

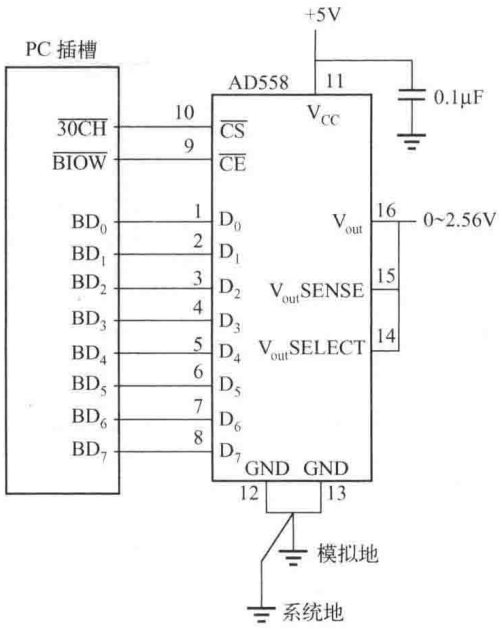


图 10-8　AD558 与计算机的连接电路

```
START:   MOV     CX, 256
         MOV     AL, 0
         MOV     DX, 30CH
LOOP1:   OUT     DX, AL                ; 输出 AL 内容
         CALL    DELAY                 ; 延时
         INC     AL                    ;  AL 内容加 1
         LOOP    LOOP1                 ; 循环 256 次
         JMP     START                 ; 重新输出下一个锯齿波
CODE     ENDS
         END     START
```

用户可以通过改变延时子程序的延时时间来改变锯齿波的周期。

10.2.2 TLC5620（串行 8 位 DAC）

近年来，D/A 转换应用中出现了串行接口芯片，以极少的信号线实现了 8 位甚至更多位的 D/A 转换，使 D/A 接口非常简单，如串行 8 位 D/A 转换器 TLC5620。

TLC5620 是一个 4 路串行 8 位电压型输出 DAC，带缓冲参考电压输入（高阻抗）。转换器输出电压的范围为参考电压的 1～2 倍，且为单调变化。此芯片易于操作，在单 5 V 电源下工作。加电时，内部自动复位确保了可重复启动。只需要通过 3 根串行总线就可以完成 8 位数据的串行输入，易于和工业标准的微处理器或微控制器（单片机）接口。TLC5620 适用于可编程电压源、数字控制放大器或衰减器、移动通信、自动检测设备、程序监控、信号合成等场合。DAC 寄存器是双缓冲型的，支持一个对于新数据写入设备的完全设置，这样所有的 DAC 输出通过 LDAC 的控制将同时更新。其主要特点如下：

- ❖ 4 路 8 位电压输出。
- ❖ 单 5 V 电源工作。
- ❖ 串行接口。
- ❖ 高阻抗基准输入端。
- ❖ 可编程 1 倍或 2 倍输出量程。
- ❖ 易于同步更新。
- ❖ 加电时内部自动复位。
- ❖ 微功耗。
- ❖ 半缓冲输出。

1．功能框图

TLC5620 的内部功能框图如图 10-9 所示。

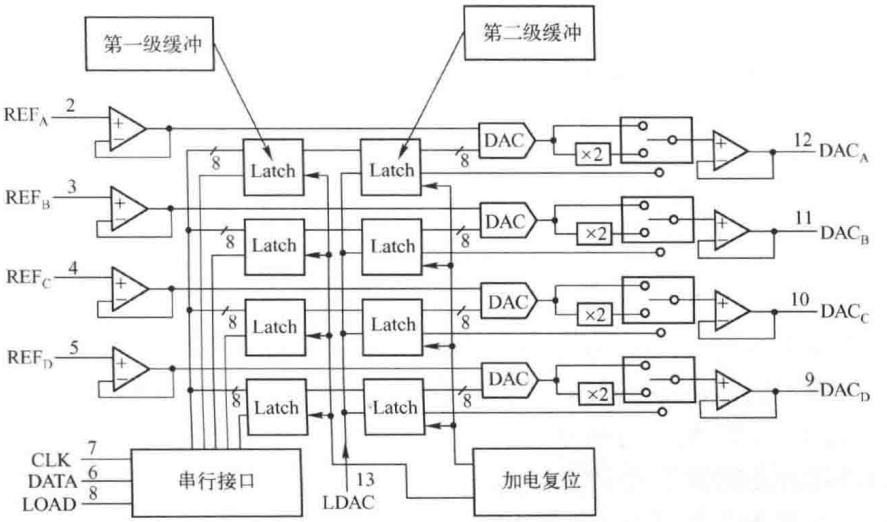


图 10-9 TLC5620 内部功能

TLC5620 主要由以下几部分组成：8 位 DAC 电路；8 位移位寄存器，接收串行移入的二进制数；4 路参考电压输入端 REF_A、REF_B、REF_C 和 REF_D；2 倍电路提供最大值为 2 倍于参考电压的输出；加电复位电路和控制电路。

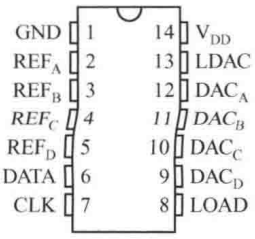


图 10-10　TLC5620 引脚

2．引脚功能

14 脚直插式 TLC5620 的引脚分布如图 10-10 所示。

① *CLK*：串行接口时钟。在由 *CLK* 终端响应的时钟信号的下降沿，输入的数字数据即被移入串行接口寄存器。

② DATA：串行接口二进制数输入端。

③ LDAC：装载 DAC。只有当 LDAC 为下降沿时，DAC 输出才会更新。

④ LOAD：串行接口装载控制。当 LDAC 为低电平时，该信号的下降沿将二进制数写入第一级数据缓冲寄存器中。

⑤ GND：模拟地和数字地。

⑥ V_{DD}：正电源电压端，通常取+5 V。

⑦ DAC_A～DAC_D：DAC 的 A 路到 D 路模拟输出，其负载电阻不得小于 10 kΩ。

⑧ REF_A～REF_D：参考电压输入至 DAC_A～DAC_D（定义模拟输出范围），不得高于 V_{DD}–1.5 V，通常取 2 V。

3．TLC5620 的工作原理

TLC5620 基本的数据写入方式是 LDAC 控制更新方式，分三步操作。第一步是串行输入数据，当 LOAD 为高时，在 CLK 的每个下降沿数据通过 DATA 端串行输入到移位寄存器中。第二步是所有数据位均被写入后，LOAD 发送负脉冲将数据从串行寄存器写入第一级数据缓冲寄存器中。第三步是 LDAC 发送负脉冲，把数据打入第二级寄存器，DAC 输出电压被更新。

数据格式（如图 10-11 所示）为：2 位的 DAC 选择信号 A_1A_0，1 位范围信号位 RNG（用于选择以 1 倍或 2 倍参考电压范围输出），最后是 8 位的数据，最高位在前。

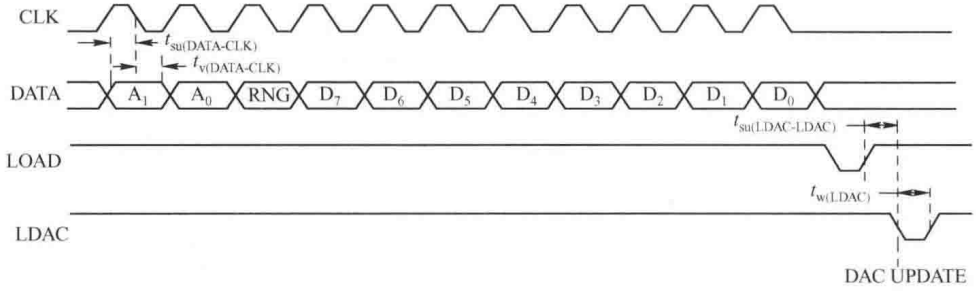


图 10-11　数据写入方式（LDAC 更新 DAC 输出）

有时可以简化写入过程，如电路上 LDAC 接地，则 LOAD 下降沿将二进制数锁入第一级寄存器，并通过第二级寄存器在 DAC 输出端产生模拟，如图 10-12 所示。

表 10-2 列出了 A_1 和 A_0 组合所对应的 DAC 通道。RNG 控制了 DAC 输出范围。当 RNG 为低，输出范围在给定的参考电压与地之间；当 RNG 为高，输出范围在两倍的参考电压与地之间。输出电压 V_o 可由下式得出（如表 10-3 所示）：

$$V_o(\mathrm{DAC_{A|B|C|D}}) = \mathrm{REF} \times \frac{\mathrm{CODE}}{256} \times (1 + \mathrm{RNG})$$

4．TLC5620 与 8086 CPU 的接口

图 10-13 为通过 8255A 的 C 口实现 TLC5620 与 CPU 之间的数据传输。可以看出，电路

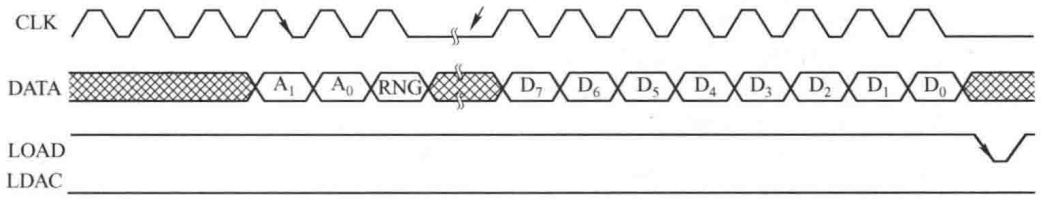


图 10-12　数据写入方式（LOAD 更新 DAC 输出）

表 10-2　DAC 通道编码

A_1 A_0	DAC UPDATE
0　0	DAC_A
0　1	DAC_B
1　0	DAC_C

表 10-3　DAC 输出电压

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	输出电压
0	0	0	0	0	0	0	0	GND
0	0	0	0	0	0	0	0	(1/256)×REF(1+RNG)
·	·	·	·	·	·	·	·	⋮
·	·	·	·	·	·	·	·	
0	1	1	1	1	1	1	1	(127/256)×REF(1+RNG)
1	0	0	0	0	0	0	0	(128/256)×REF(1+RNG)
·	·	·	·	·	·	·	·	⋮
·	·	·	·	·	·	·	·	
1	1	1	1	1	1	1	1	(255/256)×REF(1+RNG)

连接非常简单，只用了 C 口的 4 根线，却控制了 4 路 DAC 通道。其中，V_{REF} 接 2V 作为参考电压，而 DATA、CLK、LOAD、LDAC 端则分别与 PC_0、PC_1、PC_2 和 PC_3 相连。同硬件相比，相应的程序却有些复杂，好在可以利用 C 口的位控方式来实现串行数据传输。

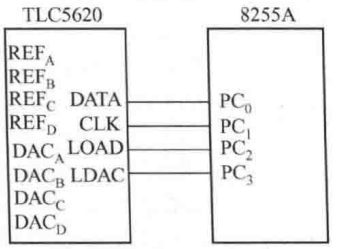


图 10-13　TLC5620 与 8255A 的连接

下面是对 TLC5620 操作的子程序。取 LDAC 更新 DAC 输出方式。程序分为 3 部分：串行输入数据，在 LOAD 负脉冲作用下数据进入第一级缓冲寄存器，以及 LDAC 负脉冲作用下数据进入第二级缓冲寄存器。程序入口是 AL 为传送给 DAC 的数据，AH 的 D_2、D_1 位为指定的 DAC 通道，即 A_1 和 A_0，D_0 位为 DAC 输出的电压范围 RNG。设 8255A 地址分布为 80H～86H。

```
DAC_PROC      PROC    FAR                 ; DAC 的子程序
              PUSH    AX
              PUSH    CX
              PUSH    DX
              PUSHF
              MOV     CL, 5               ; 先把 AX 内容左移 5 位
              SHL     AX, CL
              MOV     DX, AX              ; -DX 为串行输出的数据，最高位为通道选择
              MOV     CX, 11              ; 循环 11 次
DAC_PROC1:    MOV     AL, 0               ; 预置对 DATA 线的置位复位字
              SHL     DX, 1               ; 取串行输出位
              ADC     AL, 0               ; 把串行输出位送到置位复位字的 D0 位
              OUT     86H, AL             ; 在 DATA 线上串行输出位内容
              MOV     AL, 00000010B       ; 发送 CLK 负脉冲
              OUT     86H, AL
```

```
            MOV     AL, 00000011B
            OUT     86H, AL
            LOOP    DAC_PROC1           ; 循环
            MOV     AL, 00000100B       ; 循环完毕，发 LOAD 负脉冲
            OUT     86H, AL
            MOV     AL, 00000101B
            OUT     86H, AL
            MOV     AL, 00000110B       ; 发 LDAC 负脉冲
            OUT     86H, AL
            MOV     AL, 00000111B
            OUT     86H, AL
            POPF
            POP     DX
            POP     CX
            POP     AX
            RET
ADC_PROC    ENDP
```

主程序中，相关程序段如下：

```
            MOV     AL, 10010010B       ; 8255A 初始化
            OUT     86H, AL
            MOV     AL, 0FFH            ; C 口各位初始值全为 1
            OUT     84H, AL
            …                           ; 其他处理
            MOV     CX, 256             ; 下面程序段使 D/A 通道 B 产生一锯齿波
            MOV     AL, 0               ;  D/A 初始数据为 0
            MOV     AH, 00000010B       ; 选取通道 B，最大输出电压为参考电压
AGAIN:      CALL    DAC_PROC            ; 把 AX 里的内容送到 DAC
            INC     AL                  ; 产生锯齿波的下一个数据
            CALL    DELAY               ; 延迟
            LOOP    AGAIN               ; 循环 256 次
            …                           ; 其他处理
```

10.2.3 12 位 DAC

图 10-14 为 12 位 DAC 通过锁存器 8255A 与 8086 CPU 的连接图。此时，8086 CPU 以字的形式组织该接口。两片 8255A 具有相同的口地址，且使用 $\overline{\text{BHE}}$ 控制信号允许高 8 位数据传输。一个 8255A 的 PA_0～PA_7 提供 12 位 DAC 的低字节数据，另一个 8255A 的 PA_0～PA_3 提供 12 位 DAC 的高 4 位数据。

10.3 ADC 及其接口

ADC 的功能是将模拟量电信号转换为数字量电信号。衡量 ADC 性能的最重要参数也是分辨率。分辨率是指在量化输入的模拟信号时，ADC 所能分辨的最小模拟电压的能力。ADC 的分辨率用输出二进制数的位数表示，位数越多，误差越小，转换精度越高。例如，输入模拟电压的变化范围为 0～5 V，输出 8 位二进制数可以分辨的最小模拟电压为 5 V/(2^8–1)≈20 mV；

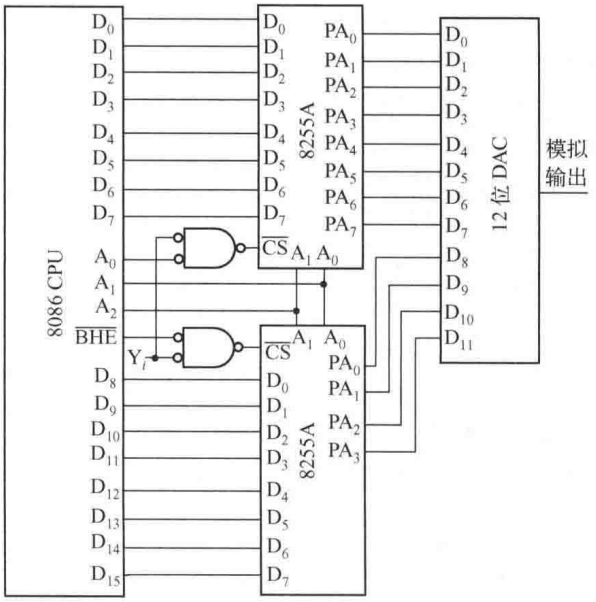


图 10-14　12 位 DAC 及接口

若 0～5 V 电压对应 0～255℃温度值，则 1℃的温度变化可引起 ADC 输出的数字量变化 1 个单位。

10.3.1　A/D 转换原理

实现 A/D 转换的方法很多，按照转换原理的不同，可将 A/D 电路分成直接转换和间接转换两大类，每类中又包含许多结构不同、性能各异的电路。直接 A/D 就是把模拟量直接转换为数字量，如并行 A/D 和逐次逼近 A/D。间接 A/D 就是把模拟量转换为其他形式的量，然后通过简单电路（如定时器/计数器）转换为数字量，如电压－时间变换的双积分 A/D 和电压－频率变换 A/D。

在微机应用系统中经常采用的是逐次逼近式的 ADC，如图 10-15 所示。寄存器内存放有逼近输入电压的二进制数码，一开始其值为最大量程的 1/2。例如，8 位 ADC 由 10000000B 为第一次比较值，寄存器的输出与一个 DAC 相连接，产生比较电压，比较结果返回寄存器，决定当前位是否清零。与此同时，时钟脉冲作为节拍去触发寄存器，使之逐位（从左到右）置 1，进行比较。其比较过程如图 10-16 所示。这是一个 6 位 ADC 转换过程，该输入电压经过转换为 110010B 二进制代码。从 ADC 的逼近过程可知，从模拟量输入到数字量输出要经过一定的时间，通常称为 ADC 的转换时间。为了保证 ADC 的转换精度，在转换期间，应保证模拟输

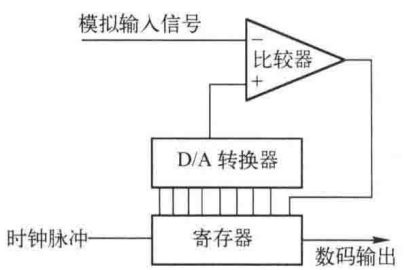


图 10-15　逐次逼近 ADC

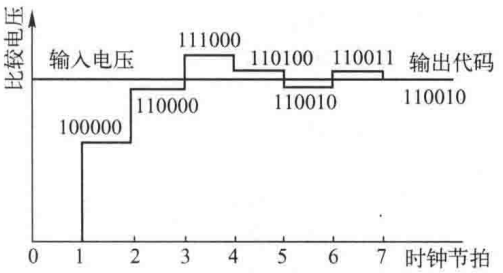


图 10-16　逐次逼近过程

入电压不变。当模拟信号变化比较快时，应设置采样－保持电路，这个电路获取该时刻的模拟信号值，并把这个值暂时保存下来，直到转换结束为止。

有关 A/D 转换原理的详细内容已超过本书讨论的范围，感兴趣的读者可以参阅相关书籍。

10.3.2 A/D 转换与微机接口技术的一般原理

ADC 是微机的一种输入设备，其接口技术的关键是三态总线输入问题和时间配合问题。

1．三态总线输入问题

A/D 转换的结果在数据寄存器中保留，直到下一次启动转换为止。大多数微机不希望这个数据简单地加在其数据总线上，而是在执行相应的输入指令时才加到数据总线上。因此，ADC 转换好的数据必须经过缓冲三态器件与微机数据总线相连。有的 ADC 芯片带有三态输出缓冲器，其控制端为 OE（输出允许）。不带三态缓冲器的 ADC 芯片（如 AD570 芯片）与微机接口，必须使用三态器件，如 8255A 或 74LS273 等。

2．时间配合问题

ADC 从启动转换到转换结束经过的时间快则几微秒，慢则有几毫秒或更长。在一般情况下，A/D 转换所需时间大于微机的指令周期。为了输入正确的转换结果，必须解决 ADC 与 CPU 取数之间的时间配合问题。ADC 芯片一般有三个信号要求控制：启动转换信号（START），转换结束信号（EOC）和允许输出信号（OE）。其中，启动转换是由 CPU 提供给 ADC 芯片的，ADC 芯片接到该信号则开始 A/D 转换过程，转换完毕，ADC 输出一个 EOC 信号，通知 CPU 转换结束，数据可用且被送入输出缓冲器中保存。CPU 若准备取数，则发出 OE 信号开通三态门，让 ADC 将数据输出，完成一个数据的转换过程。

可见，A/D 接口的工作方式应该是一个典型的条件传输方式。

10.3.3 A/D 转换与微机接口电路

1．延时等待法接口电路

延时等待法是利用 CPU 执行一条 OUT 指令，启动 A/D 转换，然后 CPU 执行软件延时程序。延时时间一般较所选用的 ADC 芯片转换时间要长。延时结束，CPU 执行 IN 指令，打开三态门获取 ADC 转换好的数据。此时需要两个端口地址：一个是输出端口，其功能是启动 ADC；另一个是输入端口，其功能是输入转换结束的有效数据。如图 10-17 所示，ADC 转换结束信号悬空没用。下面是用延时等待法进行 256 个数据转换的程序。

```
N1      EQU     START_PORT
N2      EQU     OE_PORT
…                                       ; 其他定义
BUFF    DB      256 DUP（?）            ; 定义一个数组，其元素个数为 256
        …                               ; 其他程序段
        MOV     BX, OFFEST BUFF         ; 定义子程序入口参数
        MOV     CX, 256
        CALL    PROC_ADC                ; 调数据采集子程序
        …                               ; 其他处理
```

```
PROC_ADC  PROC    FAR                    ; 这是一个数据采集子程序
AGAIN:    OUT     N1, AL                 ; 启动 ADC
          CALL    DELAY                  ; 延时
          IN      AL, N2                 ; 取数
          MOV     [BX], AL               ; 存入数组
          INC     BX                     ; 数组指针加 1
          LOOP    AGAIN                  ; 循环
          RET
PROC_     ADC  ENDP
```

2．查询法接口电路

查询法则是由 CPU 来检查 EOC 信号。当 CPU 启动 ADC 芯片开始转换之后，可去执行其他任务，再通过状态端口检查 ADC 是否转换结束。其查询过程如第 6 章所述。图 10-18 是查询 ADC 的接口电路，有两个端口。

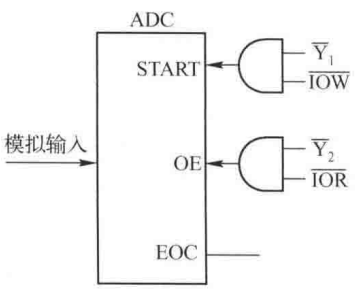


图 10-17　延时等待法 ADC 接口电路

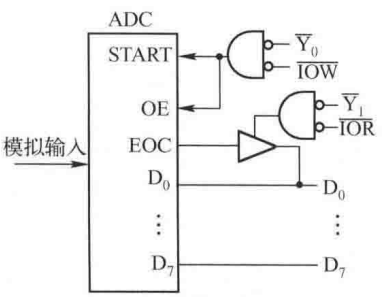


图 10-18　查询法 ADC 接口电路

CPU 先通过 $\overline{Y}_0$（译码器输出）所示端口地址执行一条 IN 指令，产生一个高电平有效的 START 信号，启动 ADC 开始转换。当 ADC 转换结束产生 EOC 信号，CPU 通过 $\overline{Y}_1$ 端口地址执行一条 IN 指令，查询 EOC 信号。在图 10-18 中，EOC 信号通过三态门接数据线 D_0 上，查询到 D_0 为“1”，则 ADC 转换好数据；CPU 再执行一条 IN 指令，从 $\overline{Y}_0$ 端口取出数据。

在实际应用时，由于 ADC 直接与外部的模拟信号相接，如果现场的干扰信号比较强，则有可能通过 ADC 芯片影响 CPU 的正常工作。因此在实际 ADC 接口电路中，常用 8255A 作为 ADC 与 CPU 的接口电路。图 10-19 就是使用 8255A 作为接口的软件查询方式下的 A/D 转换。

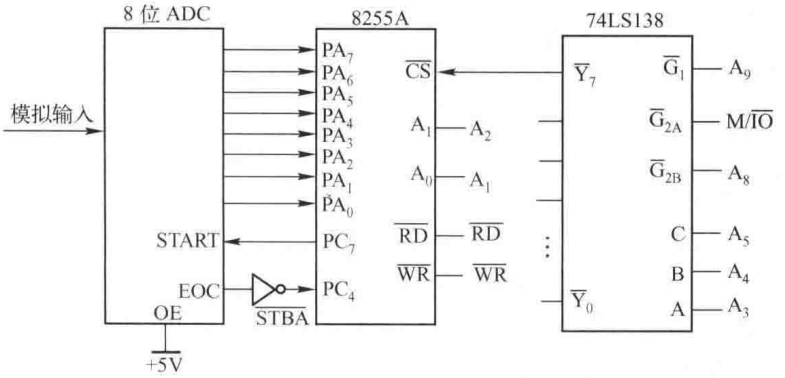


图 10-19　软件查询方式下的 A/D 转换

在图 10-19 中，8255A 的 A 口作为输入，使用方式 1 工作。A/D 转换器的启动转换“START”信号由 8255A 的 C 口提供（PC_7），A/D 转换结束信号 EOC 接 8255A 的 C 口 PC4。由于 8255A

的 A 口工作在方式 1，故 PC_4 接收 EOC 信号后，将 PA_7～PA_0 上的数据锁存到 8255A 的 A 口数据输入缓冲器，同时从 PC_5 上发出输入缓冲区满信号 IBFA；若此时读入 C 口的状态，查询到 IBFA 为“1”，则 CPU 可将 8255A 中的数据取走，同时启动下一个 A/D 转换。

由图 10-19 可知，8255A 的端口地址为 238H～23EH。下面是采集 256 个数据点的程序。

```
DATA    SEGMNET                          ; 定义数据段
BUFF    DB      256 DUP(0)               ; 定义数组变量 BUFF
DATA    ENDS
8255-A  EQU     238H                     ; 8255A 的 A 口地址 238H
8255-C  EQU     23CH                     ; 8255A 的 C 口地址 23CH
8255-S  EQU     23EH                     ; 8255A 的控制口地址 23EH
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     CX, 256
        MOV     BX, OFFSET BUFF          ; BX 为数组首地址
        MOV     DX, 8255-S               ; 初始化 8255A，A 口为输入，C 口上半
        MOV     AL, 0B0H                 ; 部为输入，下半部为输出，B 口为输出
        OUT     DX, AL
LOPP:   MOV     AL, 0FH                  ; 发 START 信号
        OUT     DX, AL
        MOV     AL, 0EH
        OUT     DX, AL
POL:    MOV     DX, 8255-C
        IN      AL, DX                   ; 输入状态信号
        TEST    AL, 20H                  ; 检测 IBFA
        JZ      POL                      ; 无效，循环检测
        MOV     DX, 8255-A               ; A 口数据有效，取数据
        IN      AL, DX
        MOV     [BX], AL                 ; 数据送数组
        INC     BX                       ; 数组指针加 1
        LOOP    LOPP                     ; 循环 256 次
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

思考题：如果 8255A 采用方式 0 工作，系统如何工作？程序又如何编写？

3．中断法接口电路

中断方法可以提高 CPU 的利用率。当 ADC 转换结束时，由 EOC 信号向 CPU 发出中断请求，CPU 响应中断在中断服务子程序中读取转换结果。图 10-20 为中断法 ADC 的接口电路。

在图 10-20 中，由 CPU 执行一条 IN 指令，启动 ADC 转换；同时将 D 触发器清“0”，使得中断请求信号无效。此时，CPU 只管去执行其他程序，一旦 ADC 转换好数据，EOC 信号使 D 触发器置“1”，向 8259A 中断控制器发出中断请求。CPU 若响应中断，则转去执行中断服务程序；CPU 执行一条 IN 指令，使 OE 信号为有效，读入 ADC 缓冲区转换好的数据。同时，

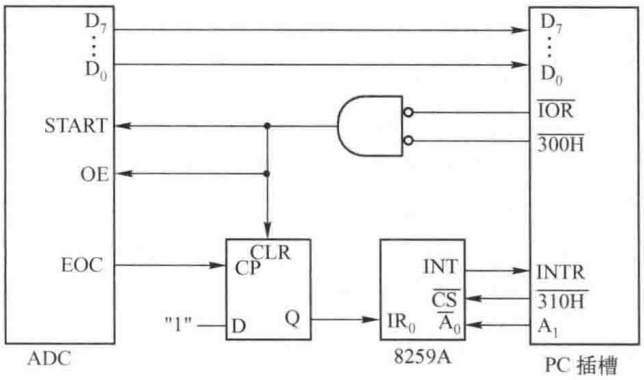


图 10-20　中断法 ADC 接口电路

START 信号有效，启动下一个数据转换，并将中断请求信号变为无效，开始下一个数的转换。如此循环，直到要求转换的数据全部转换完毕。

10.3.4　ADC0809

ADC0809 是 CMOS 单片双列直插式模数转换器件，采用逐次逼近原理。ADC0809 包括模拟多路转换开关和 A/D 转换两大部分，可对 8 路模拟电压分时进行转换。

1．ADC0809 内部结构

ADC0809 内部结构框图如图 10-21 所示。模拟多路转换开关由 8 路模拟开关和 3 位地址锁存器组成，可以选通 8 路模拟输入中的任何一路。地址锁存允许信号 ALE 将地址线 ADD_A、ADD_B 和 ADD_C 进行锁存，然后由译码电路选通其中的一路，被选中的通道进行 A/D 转换。

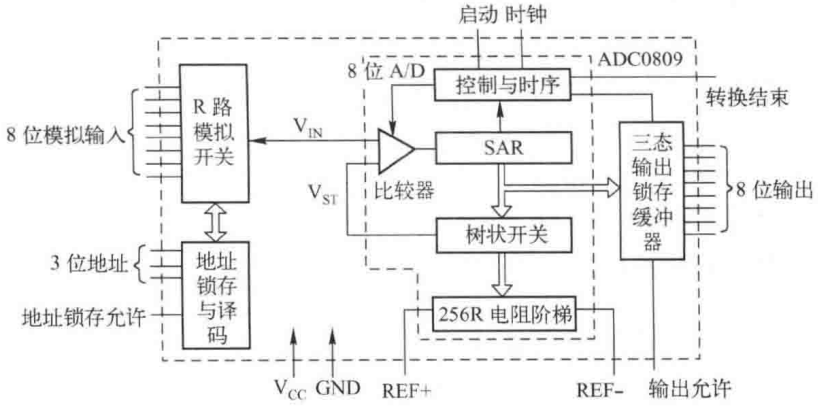


图 10-21　ADC0809 内部结构框图

A/D 转换部分包括比较器、逐次逼近寄存器（SAR）、256R 电阻、树状开关、控制与定时电路。

2．引脚说明

ADC0809 引脚图如图 10-22 所示。IN_0～IN_7 为模拟信号的 8 个输入通道。REF_+、REF_- 为基准电压的正极和负极。ADD_A、ADD_B 和 ADD_C 为模拟信号输入通道的地址选择线。

ALE 为地址锁存信号，由低电平到高电平正跳变时将地址选择线的状态锁存，以选通相应的输入通道。

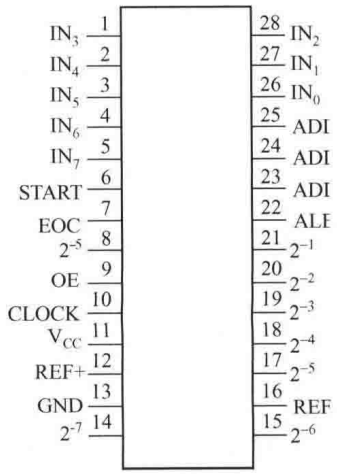


图 10-22　ADC0809 引脚

START 为启动信号，正脉冲的上升沿使所有内部寄存器清零，从下降沿开始进行 A/D 转换。

EOC 为转换结束信号，在 START 信号之后变低，转换结束变为高电平，用来申请中断。

OE 为输出允许信号，有效时将输出寄存器中的数据放到数据总线上。

2^{-8}～2^{-1} 为数码输出端，2^{-8} 为最低有效位，2^{-1} 为最高有效位。

3．ADC0809 的有关参数

ADC0809 为 8 位 ADC，其分辨率为满量程电压的 1/256。当基准电压选定为 V_{REF_+} =+5 V、V_{REF_-} =0 V 时，若输入模拟电压为 +1.5 V，则转换成数字量为 77，即 01001101B，模拟输入与数字量输出的关系为：

$$N=\frac{V_{IN}-V_{REF_-}}{V_{REF_+}-V_{REF_-}}\times 256$$

ADC0809 的转换时间为 100 μs。要求 V_{REF_+} 不超过 V_{CC} +0.1 V，V_{REF_-} 不低于 V_{CC} –0.1 V，中心值 $(V_{REF_+}+V_{REF_-})/2$ 的偏差值不超过 ±0.1 V。若只转换正电压，则将 REF_+ 接 V_{CC}，REF_- 接地。

4．ADC0809 的多路转换

在实时控制与实时数据处理系统中，被控制与被测量的电路往往是几路或几十路。对这些电路的参量进行模数、数模转换时，常采用公共的模数、数模转换电路。因此，对各路进行转换是分时进行的。此时，必须轮流切换各被测（或控制）电路与数模或模数转换电路之间的通道，以达到分时的目的。多路开关可实现分时切换的功能。

ADC0809 在模拟输入部分有 8 路多路开关，可由 3 位地址输入 ADD_C、ADD_B 和 ADD_A 的不同组合选择。例如，当 ADD_C、ADD_B 和 ADD_A 三个引脚接成“100”状态，且在 ALE 有效时，ADC0809 将对 IN_4 引脚的模拟输入信号进行转换。若 3 位地址输入信号接 CPU 的数据线 D_0～D_2，其状态由 CPU 提供，则可同时对 8 路不同的测量或控制电路进行 A/D 转换。图 10-23 为 ADC0809 作为多路转换的连接图。

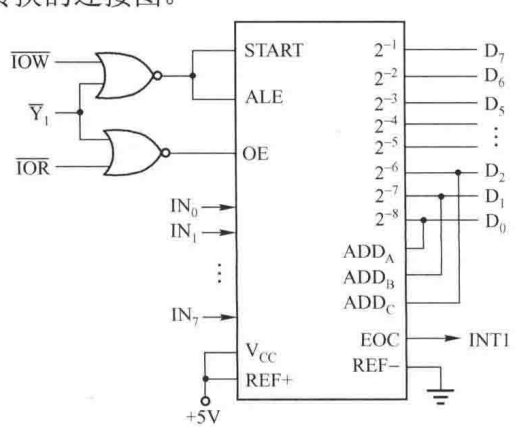


图 10-23　ADC0809 多路转换连接

在数据采集系统中，一般要求等间隔采样。为此电路中要含有定时部件，如 8253 等。采集数据时，用定时器产生定时启动，用中断方式进行数据传输的 A/D 转换。下面是一个完整的数据采集例子，定时器 8253 的通道 0 发定时钟即采样频率，以产生任意所需的 A/D 转换时间间隔。电路连接如图 10-24 所示。

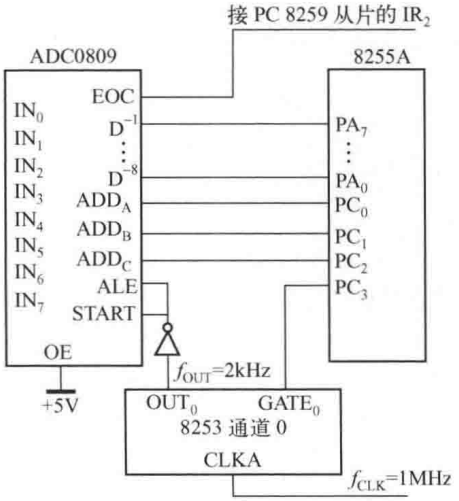


图 10-24　定时中断方式的 A/D 转换

在此例中，8253 通道 0 工作在方式 2，其 OUT_0 输出负脉冲经反向后作为 START 信号，启动 ADC 进行转换；ADC0809 的 EOC 作为中断请求信号接 8259A 从片的 IR_2。ADC 转换结束后 EOC 有效，触发 8259A 发中断请求信号给 CPU。CPU 在中断服务程序中通过 8255A 的 A 口取走 ADC0809 转换好的数据。

8255A 的 A 口为方式 0 输入，接收 ADC 转换来的数据。C 口下半部分为输出，其端口上的 PC_3 作为 8253 通道 0 的 GATE 信号；当其为 1 时，通道 0 工作，发定时脉冲。PC_0～PC_2 提供 ADC0809 的通道选择信号。

下面计算 8253 通道 0 的计数常数值。已知时钟 f=1 MHz，数据采集要求 0.1 s 内采集 200 个数据。显然，采样频率 f_{out} 为

$$f_{out}=200/0.1\ \text{s}=2000\ \text{Hz}=2\ \text{kHz}$$

这意味着 8253 通道 0 在方式 2 工作下，要产生 f_{out}=2 kHz 信号。因为通道 0 的输入频率为 f_{CLK}=1 MHz，故分频系数也就是计数常数 N 为

$$N=f_{CLK}/f_{out}=1\ \text{MHz}/2\ \text{kHz}=500$$

下面讨论这个数据采集系统的编程，该程序包括以下几方面：各种可编程接口芯片的初始化，中断矢量表的建立，主程序，中断服务程序。在程序编写中，应当考虑程序各部分的先后次序。一般先建立中断矢量表，对计算机中的中断控制芯片 8259A 进行设置后，初始化编程接口芯片。对 8255A 的 C 口 PC_3 置位应在初始化的最后进行，因为 8253 通道 0 的 GATE 信号一旦为高，定时开始采集系统就进行数据转换工作了。

```
8255_A      EQU     80H             ; 定义 8255A 的 A 口的符号地址
8255_C      EQU     84H             ; 定义 8255A 的 C 口的符号地址
8255_S      EQU     86H             ; 定义 8255A 的控制口的符号地址
8253_0      EQU     88H             ; 定义 8253 通道 0 的符号地址
8253_C      EQU     8EH             ; 定义 8253 的控制口的符号地址
```

```
DATA     SEGMENT
BUFF     DB      200 DUP(?)                ; 定义数组变量，用于存储采集数据
BUFF_P   DW      ?                         ; 定义数组变量指针
COUNTER  DB      ?                         ; 定义计数变量
DATA     ENDS
CODE     SEGMNET
         ASSUME  DS:DATA, CODE:CS
START:   MOV     AX, DATA
         MOV     DS, AX
         CLI                               ; 关中断
         MOV     AX, 0
         MOV     ES, AX
         MOV     BX, 72H*4                 ; 建立中断矢量表
         MOV     AX, OFFSET INT_ADC
         MOV     ES:[BX], AX
         MOV     AX, SEG INT_ADC
         MOV     EX:[BX+2], AX
         IN      AL, 21H                   ; 取主 8259A 的中断屏蔽字
         AND     AL, 11111011B             ; 允许主 8259A 的 IR2 中断
         OUT     21H, AL
         IN      AL, 0A1H                  ; 取从 8259A 的中断屏蔽字
         AND     AL, 11111011B             ; 允许从 8259A 的 IR2 中断
         OUT     0A1H, AL

         MOV     AL, 10010000B             ;  8255A 初始化
         OUT     8255_S, AL
         MOV     AL, 00000000B             ; 选 ADC0809 通道 0
         OUT     8255_C, AL                ; 并使 GATE0=0
         MOV     AL, 00000101B             ; 8253 通道 0 初始化
         OUT     8253_S, AL
         MOV     AL, 00                    ; 送通道 0 初始计数值（BCD）
         OUT     8253_0, AL
         MOV     AL, 05
         OUT     8253_0, AL

         MOV     BUFF_P, OFFSET BUFF       ; 置数组指针首地址
         MOV     BYTE PTR COUNTER,200      ; 置计数变量初始值
         STI                               ; 开中断
         MOV     AL, 00000111B             ; 置位 PC3
         OUT     8255_S, AL                ; GATE0=1，启动定时
         …                                 ; 主程序的其他处理
INT_ADC  PROC                              ; 这是 ADC 中断服务子程序
         PUSH    AX
         PUSH    BX
         PUSHF
         IN      AL, 8255_A                ; 取 ADC 转换后的数据
         MOV     BX, BUFF_P
         MOV     [BX], AL                  ; 存这个数据
         INC     WORD PTR BUFF_P           ; 数组指针加 1
```

```
        DEC     COUTER                          ; 计数变量减 1
        JNZ     INT_ADC1                        ; 200 个数据未采集完，返回主程序
        MOV     AL, 00001000B                   ; 200 个数据采集完，使 GATE0=0
        OUT     8255_S, AL
        MOV     AL, 20H
        OUT     0A0H, AL                        ; 向从 8259A 发 EOI 命令
        OUT     20H, AL                         ; 向主 8259A 发 EOI 命令
INT_ADC1: POPF
        POP     BX
        POP     AX
        IRET
INT_ADC ENDP
CODE    ENDS
        END  START
```

10.3.5 TLC0831（串行 8 位 ADC）

TLC0831 是 8 位逐次逼近电压型 ADC，主要特点如下：

- 分辨率为 8 位。
- 单 5 V 供电，此时输入范围为 0～5 V。
- 单信道差分输入。
- 输入/输出可与 TTL 和 MOS 兼容。
- 时钟频率为 250 kHz 时，转换时间为 32 μs。
- 总失调误差为 1LSB。

1．TLC0831 引脚

TLC0831 的引脚如图 10-25 所示，各引脚的功能如下。

- $\overline{CS}$：片选信号（低电平有效）。当 $\overline{CS}$=0 时，TLC0831 正常工作。
- IN_+，IN_-：模拟电压差分输入引脚。
- GND：地。
- REF：参考电压。
- DO：A/D 转换结果数字量输出端。
- CLK：实时时钟，最高允许值为 250 kHz，可用外接 RC 电路改变频率。
- V_{CC}：电源电压，一般接+5 V。

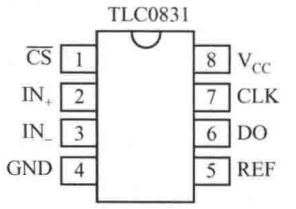


图 10-25　TLC0831 的引脚

2．TLC0831 的工作原理

TLC0831 内部集成一个逐次逼近型 A/D 部件，用来转换由差分（IN_+，IN_-）输入端来的差分信号，当不需要差分输入时，IN_-可接地，信号连到 IN_+，作为单端输入。TLC0831 的 REF 端为参考电压输入端，为简化外围电路一般接 V_{CC}。转换后的数据以串行方式输出。

当 CS 置为低电平时，TLC0831 芯片开始工作，在整个转换过程中 CS 必须保持低电平。转换开始后，时钟信号从 CLK 端输入。为了保证输入到芯片内的信号稳定，逐次逼近电路在第一个时钟时间内未开始工作，而是从第二时钟起，才进行数模转换处理。在转换过程中，转

换数据同时从 DO 端输出，第一位是最高位（MSB），8 个时钟周期后转换结束。当 CS 变为高，内部寄存器将被清零。此时，输出电路回到高阻状态。如果需要另一次转换，CS 必须有一个下降沿的跳变。再重复上面的操作，如图 10-26 所示。

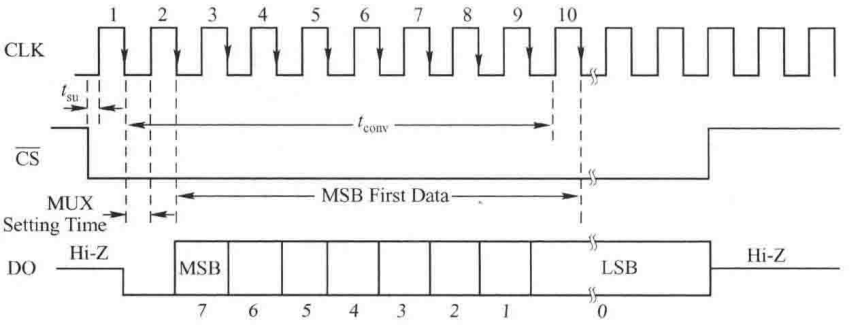


图 10-26　TLC0831 工作原理

3．TLC0831 与 8086 CPU 的接口

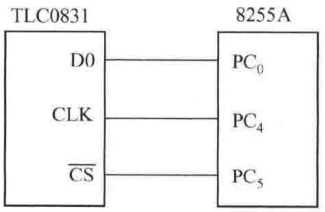


图 10-27　TLC0831

图 10-27 为通过 8255A 的 C 口实现 TLC0831 与 CPU 之间的数据传输。可以看出，电路连接非常简单，只用了 C 口的 3 根线。其中 V_{CC} 和 REF 接+5 V。而 DO、CLK、$\overline{CS}$ 端则分别与 PC_0、PC_4、PC_5 相连。程序中利用 C 口的位控方式，实现串行数据传输。

下面是对 TLC0831 操作的程序。设 8255A 地址分布为 80H～86H。

```
ADC_PROC    PROC    FAR                    ; 这是 ADC 子程序，转换结果在 AH 中
            PUSH    CX
            PUSHF
            MOV     AL, 00001010B          ; 置 CS=0
            OUT     86H, AL
            CALL    SEND_CLK               ; 发第一个 CLK 脉冲
            MOV     CX, 8
ADC_PROC:   CALL    SEND_CLK               ; 再发 CLK 脉冲
            IN      AL, 84H                ; AL.0=DO
            SHR     AL, 1                  ; CF=AL.0
            RCL     AH, 1                  ; AH.0=CF，数据进入 AH
            LOOP    ADC_PROC               ; 循环
            MOV     AL, 00001011B          ; 置 CS=1
            OUT     86H, AL
            POPF
            POP     CX
            RET
ADC_PROC    ENDP
SEND_CLK    PROC    FAR                    ; 这个程序产生 CLK 脉冲
            MOV     AL, 0001001B
            OUT     86H, AL                ; 置 PC4=0
            CALL    DELAY                  ; 延时
            MOV     AL, 0001000B
            OUT     86H, AL                ; 置 PC4=1
            CALL    DELAY                  ; 延时
```

```
        RET
SEND_CLK    ENDP
```

主程序中，相关程序段如下：

```
        MOV     AL, 10000011B          ; 8255A 初始化
        OUT     86H, AL
        MOV     AL, 0EFH               ; C 口的 PC4=0
        OUT     84H, AL
        …                              ; 其他处理
        CALL    ADC_PROC               ; 采集一个数据
        MOV     X1, AH                 ; 数据赋予字节型变量 X1
        …                              ; 其他处理
```

10.4 微机应用实例

毫无疑问，在科学实验中，最有用的研究工具应该是灵活性好，计算能力强，有可靠的测试和控制功能，有大的容量能够存储复杂的或长时间的实验过程中获得的数据，且能和许多实验室以及与大型中央实验室进行通信的设备。计算机便是科学实验中最优选的设备之一。

1．在辅助科学实验中的应用

美国西北太平拜特尔实验室用计算机作为计算和控制装置成功地完成了原子能反应堆内的模拟实验。该实验是测量核燃料涂层在失效状态下的材料性质。该涂层是镀在燃料元件上的金属外套，以防止裂变物的腐蚀和散落。在反应堆内，涂层经受温度变化的考验，在假想的反应堆发生事故过程中，温度变化约为 1649 ℃/s，样品经受内应力约为 6.89 MPa。当温度和应力增加时，样品膨胀。实验的目的是，保持应力不变（由控制内压力实现）把样品温度线性地提高到同一温度然后保持温度不变，在一定速度下提高应力。记录下全周期过程中的样品膨胀情况。如果精确知道膨胀情况，那么可以算出样品的应变和应力（由所加的压力计算）与应变（单位长度的膨胀）的比即弹性系数。该弹性系数是反应堆温度下模拟危急状态的临界参量。其他机械性能，如延伸性和强度，同样可以测定出来。

在实验室里，温度的变化是利用大功率高频感应炉，用 10 kV/250 kHz 高频电源产生 10 kW 的功率，通过热电偶测量。热电偶贴在质量为 n 克的燃料涂层样品上，如图 10-28 所示。

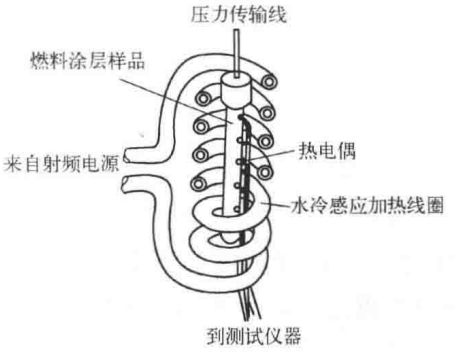


图 10-28　热电偶测量温度

在这些条件下，要求进行快速数据采集，500～800 点/秒，测量数据存到 RAM 中，以便

打印。计算机要能随时给出温度变化情况并在 CRT 上显示出来，还要完成温度值转换为热电偶电动势的转换计算，以便在温度控制器中进行比较。整个计算机控制、测量与计算装置的框图如图 10-29 所示。

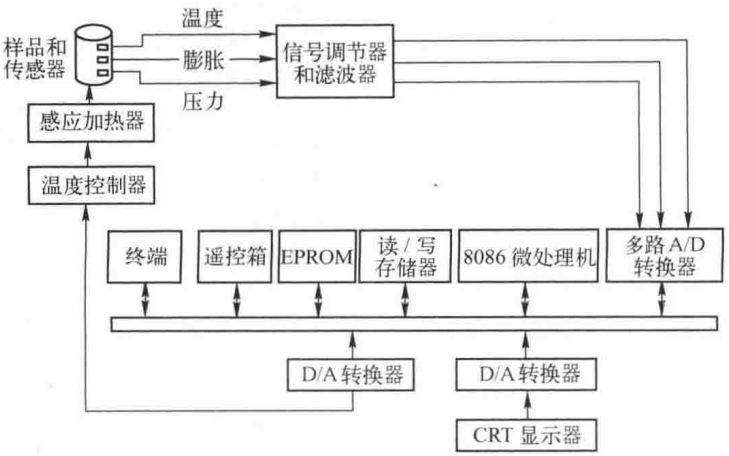


图 10-29　实验装置框图

涂层样品通过三种传感器，提供三路模拟信号，通过多路 A/D 转换器转换成数字信号供微处理机采集、存储、处理计算，通过 D/A 转换到 CRT 上进行显示，同时提供反馈控制的模拟信号。该实验装置能够快速而可靠地利用已有的数据做出结论，以及关于未来实验的决定。

2．在生物科学中的应用

生物学研究包括从呼吸毒理学到植物生理学，采用多种类型的动物实验。动物实验一般要长期进行，有的要进行十多年之久。例如，对 120 条狗吸入不同剂量的钚的情况，持续时间是 5 年。在这段时间内，要进行广泛的测试，诸如狗的血样气体色谱分析、呼吸率和呼吸量等。该实验很复杂，要求有如硬盘之类的海量存储器，一次记录几天的数据，有通过电话线传输的串行通信，将积累的数据送到中央计算机，由显示器显示呼吸参量的直方图等。

呼吸率根据计算两次呼吸之间的时间间隔来决定，呼吸量则要将外部模拟量通过传感器转换成电信号，再通过 A/D 转换由计算机测量。由于使用计算机，实验工作可在通宵无人陪伴的情况下自动进行。

3．在控制中的应用

计算机用于控制的尝试是从 20 世纪 60 年代中期开始的，以后得到迅速发展。工业生产自动控制、工业机器人控制、汽车生产线、计算机外围设备的控制等都离不开计算机。而微型计算机的超小型化和超低价格化，使微型计算机广泛地应用于控制方面，在经济上、空间大小上、稳定可靠及操作简便上都大大优于以往传统的控制系统。

微型计算机对于实现非相关方式的多变数控制是很方便的。例如对大面积的纸张表面或铁板温度或湿度控制系统中，加热器通过绝缘物体加热大面积表面的纸张或铁板，绝缘物体下放多个温度或湿度传感器，微型计算机独立、互不相干地控制电流通过各加热器的灯丝。

设灯丝的输入热量为

$$U^i(t)\quad (i=1, 2, \cdots, k)$$

传感器的输出热量为

$$Y^i(t)\ (i=1, 2, \cdots, m)$$

实际要求保持的传感器输出热量为

$$W^j(t)\ (j=1, 2, \cdots, m)$$

这样，$Y^i(t)-W^j(t)$ 作为微型计算机对灯丝加热进行控制的依据。

下面再举一个例子，以磁头定位机构来看看步进电机是如何控制磁头的。磁头定位机构主要由步进电机、钢带、磁头小车、小车导轨等组成，如图 10-30 所示。小车导轨起固定小车和导向的作用，步进电机的转角为 1.8°，步距角 3.6°/步，步进电机在控制电路控制下，每转动一个步距角，磁头就移动一个磁道的距离。

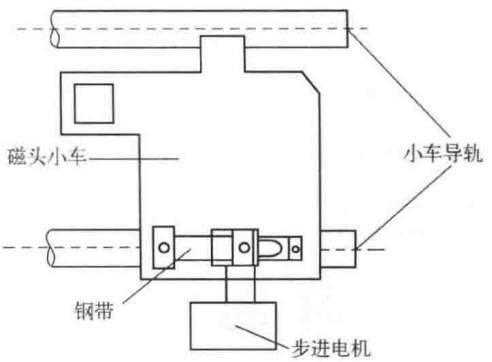


图 10-30　磁头定位结构

为了把磁头小车定位到目的磁道，步进电机与磁头小车的定位部件在凸轮上精确刻有 V 形阿基米德螺旋线，磁头小车通过该球机构接受凸轮的定位控制，随着步进电机凸轮的转动，磁头小车便向目的磁道移动。步进电机的进退根据 ϕ_0、ϕ_1、ϕ_2、ϕ_3 四相脉冲的先后时序决定。在寻道过程中，步进电机走两步移动一个磁道，即只有在 ϕ_0 和 ϕ_2 通电时，磁头位于磁道中心才能进行读/写；而 ϕ_1 和 ϕ_3 通电时，为过渡状态，磁头位于相邻两道之间不能进行读/写，其时序关系如图 10-31 所示。要使用一个并行的 I/O 接口，输出时序为如图 10-31 所示的时序信号来控制四相步进电机。

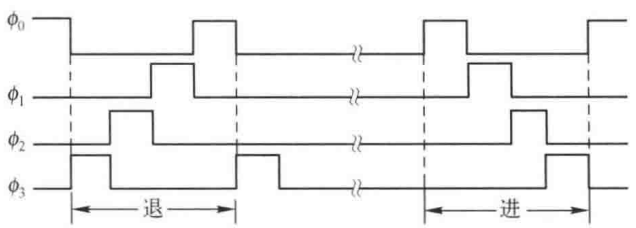


图 10-31　磁头读/写状态

4．在临床医疗仪器中的应用

这里简单介绍心电图的测量。正常人体内，由窦房结发出的一次兴奋，按一定的途径和时程，依次传向心房和心室，引起整个心脏的兴奋。因此，在每个心动周期中，心脏每个部分兴奋过程中出现的电变化的方向、途径、次序和时间都有一定规律。这种生物电变化通过心脏周围的导电组织和体液反映到身体表面，使身体各部位在每个心动周期内也都发生有规律的电变化。把测量电极放置在人体表面的一定部位，记录的心脏电变化曲线就是临床常规心电图（ElectroCadioGram，ECG）。心电图反映了心脏兴奋的产生、传导和恢复过程中的生物电变化，

具有重要的临床应用价值。目前国际医学界已经能够通过对心电信号的特征、规律的研究来对部分心脏病病变进行早期的预测和及时的诊断。

图 10-32 是心电图的肢体导联测量模式，通过左、右臂的心脏生物电信号通过电极送到心电放大器放大，放大后的模拟信号经 A/D 转换成为数字信号，计算机对此数字信号进行必要的处理，包括数字滤波，信号分类和识别，最后显示并打印结果。其框图如图 10-33 所示。

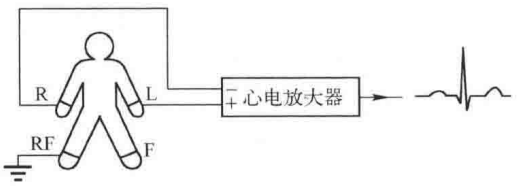


图 10-32　心电图的肢体导联测量模式

图 10-33　心电图的肢体导联测量模式框图

习 题 10

1．编写用 AD558 D/A 转换芯片产生方波的程序，试问其方波频率如何控制？

2．根据图 10-14 编写完成 12 位 D/A 转换产生三角波的程序。设端口地址为 308H，用 74LS138 译码器产生该端口地址，试画出端口地址形成电路图。此时，V_i 中的 i=？

3．一个 8 位的 A/D 转换芯片，当最大模拟量程 V_m 为+5 V 时，其分辨率为多少？若是 12 位 A/D 转换芯片，其分辨率又为多少？

4．为了测试某材料的性质，要求以 5000 点/秒的速度采样，若要采样 1 分钟，试问：至少要选用转换时间为多少的 8 位 ADC 芯片？要多少字节的 RAM 存储采样数据？

5．试设计一个智能工业供水系统。

目的：在某些工业生产中，一刻也不能离开水，否则会造成产品的报废或设备损坏。智能供水系统既能保证用水，又能节约用水。

原理：在供水控制系统中有一台小水泵（22 kW）、一台大水泵（30 kW）和一台备用水泵（30 kW），一个电动阀，一台涡轮流量计。根据流量计的信号测出水网的水压，确定直接由水网供水或需要用泵从水池抽水加压泵供水。

要求：

（1）在加压泵供水期间，根据流量信号自动切换大泵、小泵，以便节约用水。

（2）在泵供水期间检测电机是否温度过高，当超过某一温度时，三台泵自动切换。

（3）当小泵电机（1#）、大泵电机（2#）和备用泵电机（3#）均超过温升时，自动转入由水网供水方式。

（4）在任一电机发生故障时报警。

问题：

（1）试叙述你的设计思想。

（2）写出该智能系统的硬件配置。

（3）画出系统的硬件框图。

第 11 章　80286 微处理器

本章导读

- 80286 的内部结构
- 80286 的系统配置
- 80286 支持的数据类型
- 80286 指令系统
- 80286 存储器管理

8086 虽然在 20 世纪 80 年代初取得了很大的成功，但随着用户对个人微机的要求不断地提高，8086 越来越凸现出自己的不足，如运行速度较低、可寻址内存容量较小、不能方便地进行多任务操作等。在这种情况下，Intel 公司于 1982 年 2 月推出了 80286 CPU，并及时推出了以它为中央处理机的 PC/AT 机。简单地讲，80286 与 8086 向上兼容，采用 8 MHz 和 10 MHz 两种主频。采用 10 MHz 主频的 80286 的运算速度比 8086 快 5～6 倍。80286 集成度提高到 130 000 个门每片，40%的电路用来增强其性能，如内存管理、任务管理和保护设施等，这些主要是为满足多用户和多任务系统的需要而设计的。80286 内部设有存储管理部件和存储保护机构，能使用 4 个特权级支持操作系统与任务的分离，而且支持程序和数据的保密。

由于从 16 位微处理器 8086 到 32 位微处理器，80386 经历了 80286 的发展阶段，80286 对 8086 性能的提升举措，特别是 80286 阶段提出的一些全新概念，在 80386 阶段得到了全面的增强，可以讲，80386 是对 80286 的扩充和改进。所以，学习 80386 最好要了解 80286，而 80286 体系中的一个最主要部分就是内存管理。因此，本章主要介绍 80286 微处理器基本原理，重点介绍 80286 保护虚地址方式的概念和寻址过程。任务管理和保护设施等将在第 12 章中介绍。

11.1　80286 微处理器基本原理概述

80286 是 Intel 公司继 8086 之后与 80186 几乎同时推出的产品，它们都是 8086 的改进型微处理器。同 8086 和 80186 相比，80286 是一种更先进的超级 16 位微处理器，除了能与 8086 兼容，还具有许多新的特性和功能。

11.1.1 80286 内部结构简介

从外电路来看，80286 与 8086 已有较大差别，80286 CPU 芯片有 68 条引脚，其封装外形采用 4 列直插式封装，如图 11-1 所示。8086 CPU 为了节省引脚，降低对 CPU 生产工艺的要求，仅配置了 40 条引脚。由于引脚数目少，设计者不得不采用地址数据线复用方式实现数据线、地址线的功能，这种复用显然是以花费额外时间为代价来换取总线上信息的切换，会降低 CPU 对总线操作的速率。80286 在其引脚上直接安排了数据线和地址线，可以有效地提高 CPU 对总线的操作速率。

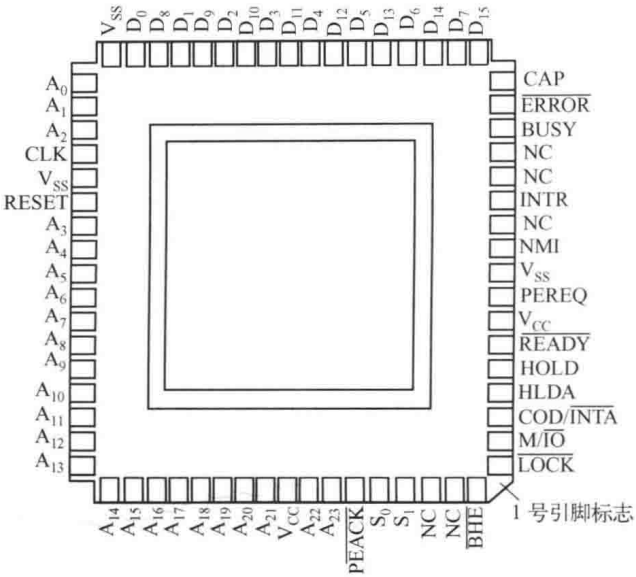


图 11-1　80286 封装外形

从内部结构来看，80286 与 8086 也有较大的差别。80286 芯片内部包含 CPU 和内存管理部件，主要由指令部件 IU、总线部件 BU、地址部件 AU 和执行部件 EU 等 4 部分构成，如图 11-2 所示。可见，80286 把 8086 中的总线接口部件 BIU 分成了 BU、AU 和 IU。这 4 个部件并行工作，具有了比 8086 更高的并行操作程度，有效地加快了系统的处理速度。

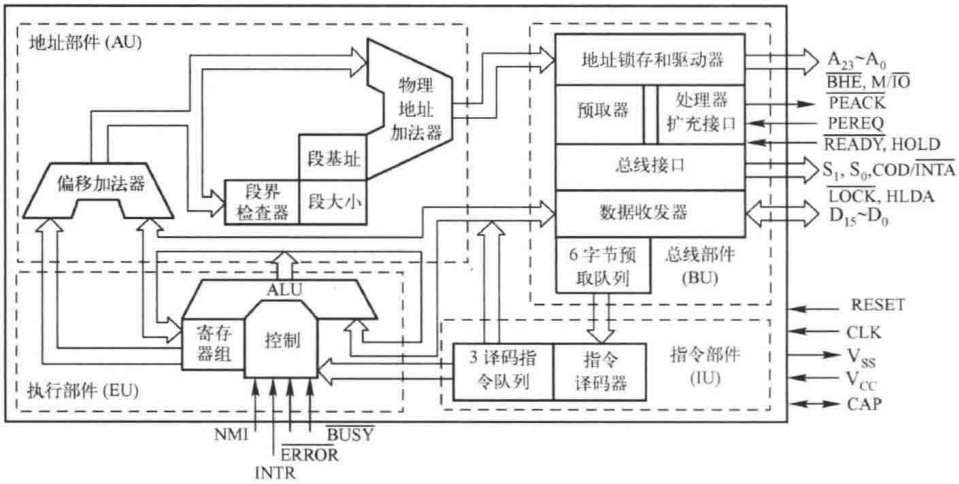


图 11-2　80286 内部结构

80286 各功能部件连接及其相互关系可以用图 11-3 来简单表示。

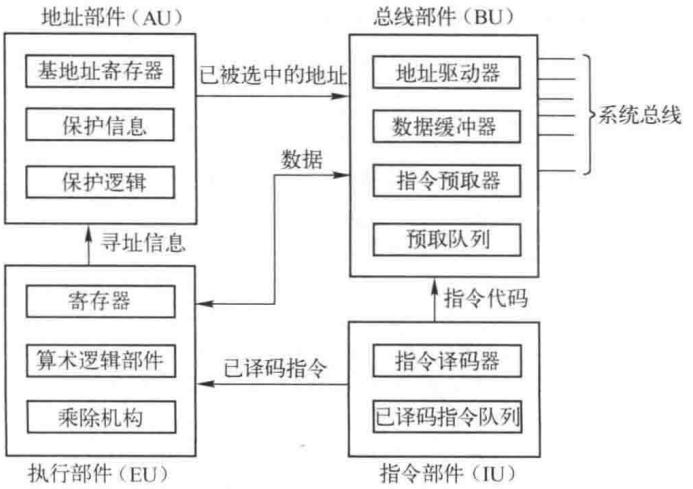


图 11-3　80286 功能部件连接

8086 CPU 中有一个指令队列，它由 6 字节的寄存器组成。而在 80286 CPU 内部形成了两个队列，一个是预取队列，另一个是已译码指令队列。其中预取队列是一个 6 字节的队列，它包含在总线部件 BU 中，只要队列中空出 2 字节，BU 就会去访问存储器，读出后续指令来填充预取队列。显然，这就是 8086 CPU 中的指令队列。预取队列中的指令经译码后，可按指令的执行顺序进入已译码的指令队列中，这里可存放 3 条已译码的指令，等待进入执行部件去执行。如果遇到转换类指令，在译码时就可发现，可提前通知 BU 将原预取队列中的指令作废，重新从目标地址中取得新的指令置入预取队列中，而 EU 仍可从已译码队列中得到可执行指令。只有条件转换指令有可能使已译码队列中的指令作废，这时其并行操作程度会受到一定的影响。

80286 的地址部件中设置有两个地址加法器：一个用来计算偏移地址值（16 位），另一个用来计算 24 位的物理地址。

与 8086 一样，80286 具有 16 位系统总线，内部完成 16 位的运算，所以 80286 仍属于典型的 16 位微处理器。与 8086 相比，80286 具有更大的存储空间，支持虚拟存储体系，能以实地址和保护虚地址两种方式运行。

80286 对存储器管理分为两种方式：实地址方式和保护虚地址方式。在实地址方式下，80286 与 8086 目标地址兼容，可寻址 1 MB 的存储空间；在保护虚地址方式下，可直接寻址的物理空间扩大为 16 MB（224），因此访问存储器的物理地址由 20 位扩展为 24 位。另外，虚存空间可达到 1000 MB 以上。

80286 中的寄存器组与 8086 基本相同，主要不同有以下几点。

① 标志寄存器增设了两个标志位字段，其具体格式如图 11-4 所示。其中，IOPL 字段为特权标志，占用 2 位，用来定义当前任务的特权级。这样 80286 就有 4 个如图 11-5 所示的特权级，即特权级 0、特权级 1、特权级 2 和特权级 3。序号低的特权级的级别高于序号高的特权级，因此特权级 0 的级别最高，而特权级 3 的级别最低。一般操作系统被设置为较高特权，而用户程序被设置为较低特权。NT 位为任务嵌套标志，占用 1 位。NT=1，表示当前执行的任务嵌套于另一任务中，否则 NT=0。其他 9 个标志位都与 8086 相同。

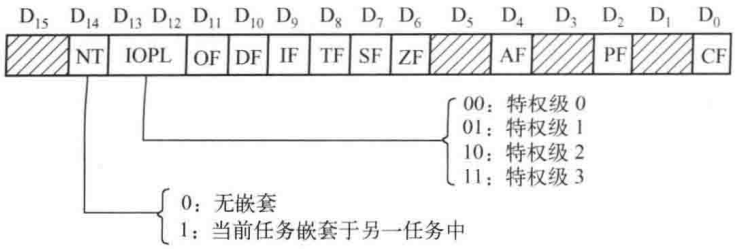


图 11-4　80286 标志寄存器

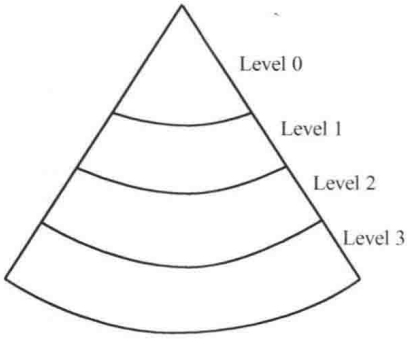


图 11-5　80286 特权级

② 80286 内部还设有 16 位的机器状态字（MSW），只使用低 4 位，系统复位时，MSW 寄存器被置成 FFF0H，使 80286 处于实地址方式。MSW 的具体格式如图 11-6 所示。

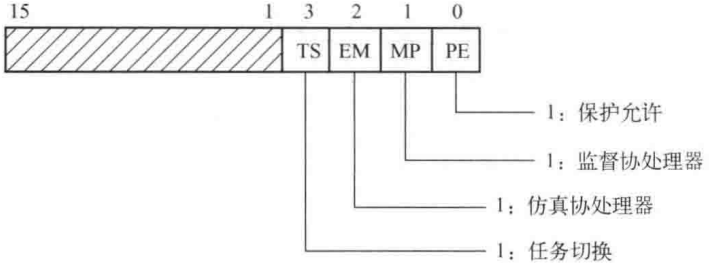


图 11-6　80286 机器状态字

PE：保护允许位，用来启动 80286 进入保护虚地址方式。PE=0，表示 CPU 当前处于实地址方式；PE=1，表示 CPU 当前已进入保护虚地址方式。

MP：监督协处理器位，用来定义系统中是否存在协处理器。MP=1，表示系统中存在协处理器；否则 MP=0。

EM：仿真协处理器位，用来定义是否要用软件仿真协处理机。EM=1，表示系统无协处理器，要求用软件仿真协处理器；否则 EM=0。

TS：任务切换位，用来表明是否产生了任务切换。TS=1，表示当前产生了任务切换，一旦任务切换操作完成，应立即将 TS 置“0”。

系统可用 LMSW 和 SMSW 指令将机器状态字取出和存入存储器中保留，其中 TS、MP 和 EM 位可组合编码表明特定意义，如表 11-1 所示。

③ 80286 增加了任务寄存器（TR），如图 11-7 所示。TR 是一个 64 位的寄存器，只能在保护方式下使用，用于存放表示当前正在执行的任务的状态。当进行任务切换时，就用它来自动保存和恢复机器状态。它的 16 位段选择器字段由 CPU 运行程序来装入 16 位段选择字，再

表 11-1　TS、EM、MP 组合编码意义

TS	EM	MP	特 定 含 义
0	0	0	80286 处于实地址方式下，当前是复位后的初始状态
0	1	0	没有协处理机可供使用，要求用软件仿真
0	0	1	有协处理机，不需要软件仿真
1	1	0	无协处理机，要求软件仿真，当前产生了任务切换，协处理机可属下一个任务
1	0	1	有协处理机，不需要软件仿真，产生了任务切换，协处理机可属下一个任务

由段选择字选择 48 位的段描述符装入相应的“描述符高速缓存器”，即 Cache。描述符包含了当前任务状态段的容量、基地址与访问权等信息。

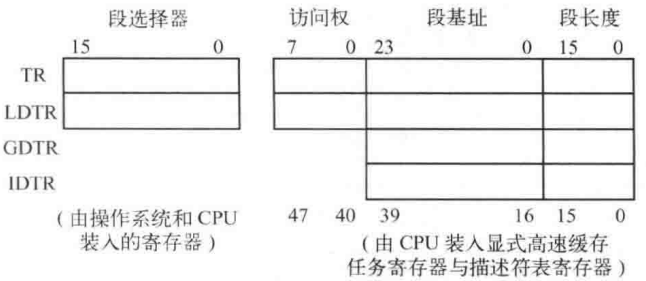


图 11-7　任务寄存器和描述符表寄存器

④ 80286 增加了 3 个描述符表寄存器（见图 11-7）：64 位的局部描述符表寄存器（LDTR）、40 位的全局描述符表寄存器（GDTR）和 40 位的中断描述符表寄存器（IDTR）。它们总是存放包含各种段描述符的描述符表基地址。在用保护方式寻址时，将要用描述符表作指针，有关内容在本节后面部分介绍。

TR、GDTR、LDTR 和 IDTR 称为系统地址寄存器。

11.1.2　80286 芯片引脚功能

80286 芯片引脚的符号与名称如表 11-2 所示。与 8086 比较，80286 芯片引脚功能差别较大，它不采用地址数据线复用方式。由于物理空间可达 16 MB，输出的地址线应有 24 条（A_0～A_{23}），另外数据线 16 条（D_0～D_{15}）。

表 11-2　80286 引脚定义

符号	输入/输出	名　称	符号	输入/输出	名　称	符号	输入/输出	名　称
A_{23}～A_0	O	地址总线	$\overline{ERROR}$	I	协处理器出错	$\overline{PEACK}$	O	协处理器操作数响应
$\overline{BHE}$	O	地址高位有效	HOLD	I	总线保持请求	PEREQ	I	协处理器操作数请求
$\overline{BUSY}$	I	协处理器忙	HLDA	O	总线保持响应	$\overline{READY}$	I	总线准备就绪
CAP	I	衬底滤波电容器	INTR	I	中断请求	RESET	I	系统总清
CLK	I	系统时钟	$\overline{LOCK}$	O	总线封锁	$\overline{S}_0$, $\overline{S}_1$	O	总线周期状态
COD/$\overline{INTA}$	O	代码中断响应	M/$\overline{IO}$	O	存储器、I/O 选择	V_{SS}	I	系统地
D_{15}～D_0	I/O	数据总线	NMI	I	不可屏蔽中断请求	V_{CC}	I	+5 V 电源

80286 一些有特殊含义的引脚功能说明列于表 11-3 内。

表 11-4 为总线周期状态定义。

表 11-3　80286 部分引脚功能

符　号	输入/输出	名　称
COD/$\overline{\text{INTA}}$	O	代码/中断响应信号线，双功能三态输出线。当 COD=1 且 M/$\overline{\text{IO}}$ =1 时，表示当前正处于读存储器取指令代码的操作周期；$\overline{\text{INTA}}$ =0 且 M/$\overline{\text{IO}}$ =0 时，表示当前正处于中断响应周期，CPU 让出总线控制权时，COD/$\overline{\text{INTA}}$ 处于高阻浮空状态
$\overline{S}_0$, $\overline{S}_1$	O	总线状态周期信号，低电平有效。它与 COD/$\overline{\text{INTA}}$ 和 M/$\overline{\text{IO}}$ 相结合，表示 80286 的总线周期的状态，具体见表 11-4
$\overline{\text{PEACK}}$	I	协处理器认可和请求信号，80286 执行需要访问存储器的 ESC 指令时，与 80287 协处理器的应答信息。当由 80287 送出的 PEREQ 信号有效时，表示协处理器请求 80286 传输数据，80286 收到这一请求后，应向协处理器回送一个认可信号 $\overline{\text{PEACK}}$，并给它传送一个操作数
PEREQ	O	
$\overline{\text{BUSY}}$	I	协处理器忙和出错信号。这两个信号均由协处理器输入，低电平有效，反映了 80287 当前的操作情况。当 80286 执行 WAIT 指令时，若 $\overline{\text{BUSY}}$ =0，表示当前协处理器忙，80286 应处于等到状态，直到 $\overline{\text{BUSY}}$ 无效为止，80286 才能继续执行后续指令。80286 执行 WAIT 或 ESC 指令时，$\overline{\text{ERROR}}$ =0，表示协处理器发现出错，80286 应做相应处理
$\overline{\text{ERROR}}$	I	
CAP	I	要求在 CAP 与 VSS 之间链接 0.047μF±20%、耐压为 12V 的电容器，是 80286 内部的基片偏置发生器输出的滤波器

表 11-4　总线周期状态

COD/$\overline{\text{INTA}}$	M/$\overline{\text{IO}}$	$\overline{S}_0$	$\overline{S}_1$	总线周期类型
0	0	0	0	中断响应周期
0	1	0	0	若 A_1=1，则暂停，否则停机
0	1	0	1	从存储器读数据
0	1	1	0	向存储器写数据
1	0	0	1	读 I/O 端口
1	0	1	0	写 I/O 端口
1	1	0	1	从存储器读指令

11.1.3　80286 支持的数据类型和指令系统

80286 可直接支持的数据类型主要如下。

- ❖ 无符号整数：8/16 位二进制数据。
- ❖ 带符号整数：包括 8 位、16 位、32 位和 64 位二进制整数，都是用最高位作为符号位，正数用“0”表示，负数用“1”表示，均采用补码表示法。
- ❖ 十进制数：用 BCD 码表示，有两种表示法，一种是每字节存放 1 位十进制数，另一种是每字节存放 2 位十进制数。
- ❖ 字符串：用 ASCII 值表示，每字节存放一个字符的 ASCII 值。
- ❖ 32 位地址指针：实际是一个数据结构，分为两部分，高 16 位是段选择字，低 16 位是偏移地址。
- ❖ 80 位长的带符号浮点数：最高位是浮点数的符号位，高端 23 位是阶码，采用移码表示，低端 56 位是尾数。

80286 指令系统是由 8086 指令系统扩展而来的，大部分指令与 8086 相同，新增设的指令大体上可分成两类：一类是在实地址和保护虚地址方式下都可使用的一般指令，另一类是实现存储器管理和保护的特权指令。表 11-5 列出了 80286 指令系统中扩充的指令名称和功能。

表 11-5　80286 新增指令

一般指令		保护控制特权指令	
BOUND	检测超出范围的值	ARPL	从寄存器/存储器调整已请求的特权层
CLTS	清任务转换标志	LAR	从寄存器/存储器装入访问页
ENTER	进入过程	LGDT	装入全局描述符表寄存器
IMUL	带符号整数乘立即数	SGDT	存放全局描述符表寄存器
INS	输入字节或字串，或从 DX 指定的接口输入字节/字串	LIDT	装入中断描述符表寄存器
		SIDT	存放中断描述符表寄存器
LEAVE	离开过程	LLDT	从寄存器/存储器装入局部描述符表寄存器
OUTS	输出字节或字串，获得从 DX 指定的接口输出字节/字串	SLDT	把局部描述符表寄存器送到寄存器/存储器
		LMSW	从寄存器/存储器装入机器状态字
POPA	从堆栈中弹出所有寄存器	SMSW	存放机器状态字
PUSH	把立即数压入堆栈	LSL	从寄存器/存储器装入段限值
PUSHA	把所有的寄存器压入堆栈	LTR	从寄存器/存储器任务寄存器
SHIFT/ ROTATE	带计数的移位指令	STR	把任务寄存器送到寄存器/存储器
		VERE	对寄存器/存储器读进行验证
		VERR	对寄存器/存储器写进行验证

下面主要对保护控制特权指令作简要说明，有关描述符、访问权字节、限值字段和特权请求级等内容在 11.1.5 节中给出。

① ARPL 指令。ARPL 为调整请求特权级 RPL 字段指令。其功能是，如果目标操作数所指向的 RPL 字段的特权级小于源操作数所指向的 RPL 字段的特权级，则将零标志 ZF 置“1”，并引导目标操作数中的 RPL 字段增大到与源操作数所指向的 RPL 字段内容相等，否则不做任何修改。

② LAR 指令。LAR 为装入访问权字节指令，其功能是将约定的寄存器或存储器中的源操作数装入描述符中访问权字节。

③ LGDT/SGDT 指令。LGDT 为装入全局描述符表寄存器指令，其功能是从由源操作数所指向的存储器开始的连续 6 个单元中取出 6 字节的信息装入全局描述符表寄存器中。SGDT 为存入全局描述符表寄存器指令，实现反向传输，即从全局描述符表寄存器中取出信息，存入指定的存储器单元中。

④ LIDT/SIDT 指令。LIDT 为装入中断描述符表寄存器指令，其功能是将源操作数所指向的 16 位寄存器或存储单元中的 2 字节内容装入中断描述符表寄存器中。SIDT 为存放中断描述符寄存器指令，其功能与 LIDT 类似，只是传输方向相反。

⑤ LLDT/SLDT 指令。LLDT 为装入局部描述符表寄存器指令，其功能是将源操作数所指向的寄存器或存储器中的 2 字节内容装入局部描述符表寄存器中。SLDT 为存放局部描述符表寄存器指令，其功能与 LLDT 指令类似，只是传输方向相反。

⑥ LMSW/SMSW 指令。LMSW 为装入机器状态字指令，其功能是将寄存器或存储器中的源操作数装入机器状态字寄存器中。SMSW 为存放机器状态字指令，其功能与 LMSW 类似，只是传输方向相反。

⑦ LSL 指令。LSL 为装入段限值指令，其功能是将约定的寄存器或存储器中的源操作数装入段描述符的限值字段中。

⑧ LTR/STR 指令。LTR 为装入任务寄存器指令，其功能是将寄存器或存储器中的 16 位

源操作数装入任务寄存器中。STR 为存放任务寄存器指令，其功能与 LTR 指令类似，只是传输方向相反。

⑨ VERR/VERW 指令。VERR/VERW 为校验段的读/写操作指令。其功能是检验当前的读/写操作的特权级是否符合规定，并确定该段是否允许读/写，若该段允许读/写，则将零标志 ZF 置“1”，否则 ZF 置“0”。

80286 新增设的上述指令均属于特权指令，只允许 CPU 在操作系统中使用，主要用来完成保护控制功能。

11.1.4 80286 的存储器管理

我们知道，80286 有两种工作模式，即实地址模式和保护虚地址方式。在实地址模式下可把 80286 作为一个高速的 8086 来使用，因此也称实地址模式为 86 模式。当然，在实地址模式下不能全部使用 80286 的所有功能。例如，24 位地址总线中，只能使用低 20 位 A_{19}～A_0，而 A_{20}～A_{23} 位无效，20 位物理地址的形成方式与 8086 完全相同。因此在实地址模式下，80286 寻址的空间与 8086 一样，只有 1 MB。

保护虚地址方式是 80286 的一个十分重要的工作方式，是区别 8086 系列的一个全新的概念。在保护虚地址方式下，MMU 具有虚拟存储管理功能。在 8086/8088 系统中，程序占有的存储器就是 8086 的物理存储器，其大小为 1 MB。而从 80286 开始，CPU 在 MMU 的支持下可以工作在保护虚地址方式下，支持对虚拟存储器的访问。

虚拟存储器和物理存储器是有区别的，其空间大小也不相同。虚拟存储器是程序可以占有的空间，实际上这个空间是靠磁盘、光盘等外部存储器的支持来实现的，而物理存储器是 CPU 可以访问的存储器。用户编写的程序是放在磁盘存储器上，当执行程序时，必须把程序加载到物理存储器上。我们把从虚拟地址空间到物理地址空间的转换称为映射。虚拟地址对物理地址的映射如图 11-8 所示。我们把解决如何把较小的物理存储器空间分配给具有较大虚拟存储器空间的多用户/多任务的问题称为虚拟存储管理。

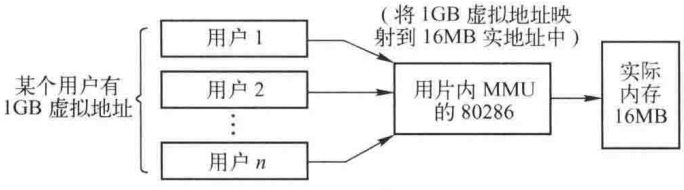


图 11-8　80286 虚拟地址到物理地址的映射

在保护虚地址方式下，80286 支持虚拟存储系统，这时地址位 A_{23}～A_0 有效，可直接寻址的物理空间达到 16 MB，而虚存空间可达 1 GB（2^{30}）。换句话说，对每个用户来说，都可认为有 1 GB 的内存空间使用，这些是存储器管理部件 MMU 来完成的。

可见，为了发挥 80286 强有力的功能，可采用保护虚地址方式，或称为 286 模式，因为在这种模式下能发挥 80286 的固有特性。实地址模式和保护虚地址方式的控制由 MSW 的 PE 位决定。PE=0 时，80286 工作于实地址模式；PE=1 时，工作于保护虚地址方式。

80286 复位后，PE=0，即自动进入实地址模式。若想从实地址模式转为保护虚地址方式，就需置位 MSW 的 PE 位。通常可用 SMSW 指令将 MSW 的值读入某寄存器（如 AX）中，再通过逻辑“或”指令置 AX 寄存器的第 0 位为“1”（这样做不改变 AX 的其他位），最后用

LMSW 指令将 AX 的内容复写到 MSW 中，这就完成了两种模式的转换。保护虚地址方式完整的初始化处理还包括在存储器中定义 GDT 等工作。

80286 实地址模式和保护虚地址方式的基本差异见表 11-6。由表可知，在指令系统和功能方面，保护虚地址模式是在实地址模式的基础上增加了一些新的指令和功能。而这两种模式最明显的差别是存储空间的大小。

表 11-6　实地址模式和保护虚地址方式的基本特征

项　目	实模式特征	特定含义
数据总线	16 bit	16 bit
地址总线	20 bit（使用 A_{19}～A_0） 存储空间为 1 MB	24 bit（使用 A_{23}～A_0） 实存储空间为 16 MB，虚存储空间为 1000 MB
使用的寄存器	AX，BX，CX，DX，SI，DI	AX，BX，CX，DX，SI，DI
使用的寄存器	AX，BX，CX，DX，SI，DI SP，BP，IP，FLAG，CS，DS SS，ES，MSW，GDTR， IDTR	AX，BX，CX，DX，SI，DI SP，BP，IP，FLAG，CS，DS，SS，ES MSW，GDTR，IDTR，LDTR，TR
指令与功能	略	与 80286 实模式共有的指令外，增加的有：LLDT，SLDT，LTR，STR，LAR，LSL，VERR，VERE 和 ARPL。取决于特权级的存储器、I/O 管理、指令、任务切换指令和虚拟存储器的支持

11.1.5　保护虚地址方式下存储器管理

80286 在保护虚地址方式中，采用 32 位虚地址指示器寻址，这是一个数据结构，它包含了 16 位段选择字和 16 位偏移地址。其中，偏移地址的作用与实地址方式相同，但 16 位的段选择字不再是段基址，而成为进入存储器中一个称为描述符表的参数。从这个描述符表中可得到 24 位的段基地址，将它与 16 位的偏移地址相加，就可以形成访问存储器的 24 位物理地址，其形成过程如图 11-9 所示。

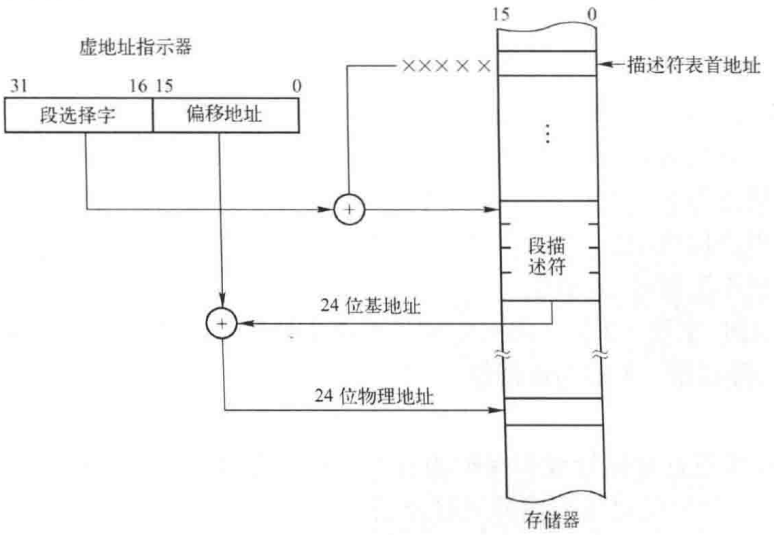


图 11-9　保护虚地址方式下的物理地址形成过程

为了加深读者的理解，这里对图 11-9 的工作过程再进行如下说明：

<1>32 位虚地址指示器中的高 16 位作为段选择字，它与描述符表首地址相加产生所需描述符的物理地址。

<2> 取出描述符中 24 位数据作为物理段的 24 位段基地址。

<3> 32 位虚地址指示器中的低 16 位作为段内偏移地址，它与 24 位段基地址相加形成 24 位物理地址。

描述符表由若干个描述符组成，每个描述符指向存储器中的一个逻辑段。

存储器管理部件 MMU 以描述符为基础对存储器进行管理。描述符均包含在描述符表中，一个描述符表最多可存放 8K 个描述符，每个描述符长 8 字节。一个最大的描述符表将占用 64 KB（8K×8B=64 KB）。

描述符表有全局描述符表（GDT）和局部描述符表（LDT）之分。前者用来存放供所有任务共享的描述符，供操作系统使用和各任务公用。后者用来存放供一个任务专用的描述符。

下面简要说明段选择字、描述符和描述符表，它们是 MMU 进行存储管理的依据和手段。

（1）段选择字

段选择字是由虚地址指示器直接提供的，是 32 位虚地址指示器中的高 16 位，其格式如图 11-10 所示。其中，描述符表指示器 TI 字段（1 位）用来定义当前使用的描述符是在全局描述符表（Global Descriptor Table，GDT）中（TI=0），还是在局部描述符表（Local Descriptor Table，LDT）中（TI=1）。

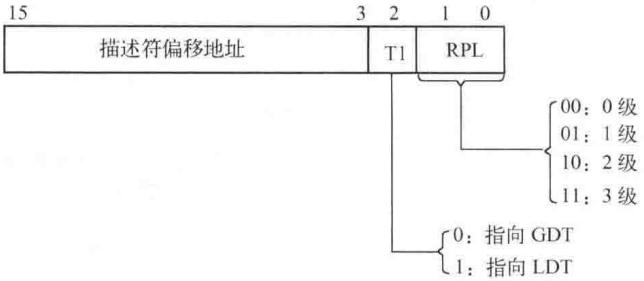


图 11-10　段选择字

描述符偏移地址字段（13 位）用来给定当前使用的描述符在描述符表中的位置。可寻址 8K（2^{13}）个描述符。这里要注意的是，由于每个描述符占用 8 字节，故描述符偏移地址字段用来在描述符表中寻址时，首先要对描述符偏移地址字段乘 8 以产生描述符的偏移地址。全局描述符表和局部描述符表一起可包含 16K 个描述符，每个描述符可定义 64 KB 的一个逻辑段，因此 80286 可给用户提供的最大虚存空间为 16K×64 KB=1024 MB。但必须记住，任何时候进入物理空间的信息不能超过 16 MB。

请求特权级 RPL 字段（2 位）用来定义当前请求的特权级级别，共有 4 级待权级（3～0）可供选择。0 层为最高级，3 层为最低级。

（2）描述符

描述符用来存放进行存储管理和保护的有关信息，是 MMU 使用的工具。描述符的分类情况如图 11-11 所示。这里仅对段描述符进行介绍。

段描述符可用来定义代码段、数据段、堆栈段和附加段，其格式如图 11-12 所示。

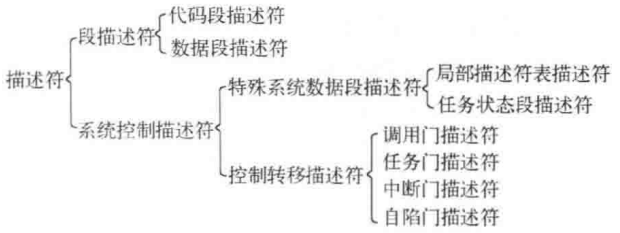


图 11-11　描述符分类

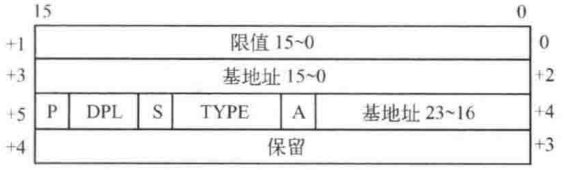


图 11-12　段描述符格式

段描述符占 8 字节，由 4 部分组成：16 位的段限值用来限制各逻辑段的长度不能超过 64KB；24 位段基地址用来指向该段的首地址在存储器中的物理地址；8 位访问权字节用来定义该段的有关特性；最后 2 字节保留供 80386 用，在 80286 中该 2 字节置 0，以便与 32 位微处理器 80386 的描述符保持互换性。访问权字节内部又分成 5 个字段，各字段的名称及其含义如表 11-7 所示。

表 11-7　访问权字节

字段名称		占用位	功　　能	备　注
P		7	P=1，该段已在实存中，段基址和限值有效 P=0，该段未在实存中，段基址和限值无效	
DPL		6～5	该段具有的特权级（0～3）	
S		4	S=1，该段为代码段或者数据段；S=0，该段不是代码段或者数据段	
TYPE	E	3	E=1，该段为数据段	对数据段有效
	ED	2	ED=0，该段向上生长，偏移地址≤限值；ED=1，该段向下生长，偏移地址＞限值	
	W	1	W=0，该段为不可写数据段；W=1，该段为可写数据段	
TYPE	E	3	E=1，该段为数据段	对代码段有效
	C	2	C=0，当 CPL≥DPL 时，该代码段只能执行；C=1，无此要求	
	R	1	R=0，该段为不可读数据段（只可执行）；R=1，该段为可读数据段	
A		0	A=0，该段未被访问；A=1，该段已被访问	

80286 还有两种特权级：CPL 和 DPL。CPL（当前特权级）是当前正在执行的代码段所具有的访问特权级。存放在 CS 寄存器的最低两位中。DPL（描述符特权级）是段的被访问的特权级，在段描述符中的访问权字节中给出。在程序的执行过程中，处理器要进行一系列的特权级检查。

（3）描述表

80286 在保护虚地址方式下需设置 3 个描述符表：全局描述符表（GDT）、局部描述符表（LDT）和中断描述符表（IDT）。对应 3 个描述符表在 CPU 中各设置一对寄存器：其一是 24 位基地址寄存器，用来存放该描述符表在存储器中的首地址；其二是 16 位限值寄存器，用来存放该描述符表的最大字节数，即任何一个描述符表的最大容量为 64 KB。全局和局部描述符

表最多可存放 8K 个描述符（8K×8 B=64 KB），中断描述符最多只需存放 256 个描述符，这是因为系统中最多只允许定义 256 种类型的中断。由 3 对基地址和限值寄存器把 3 种描述符表限制在存储器的一定范围内，如图 11-13 所示。

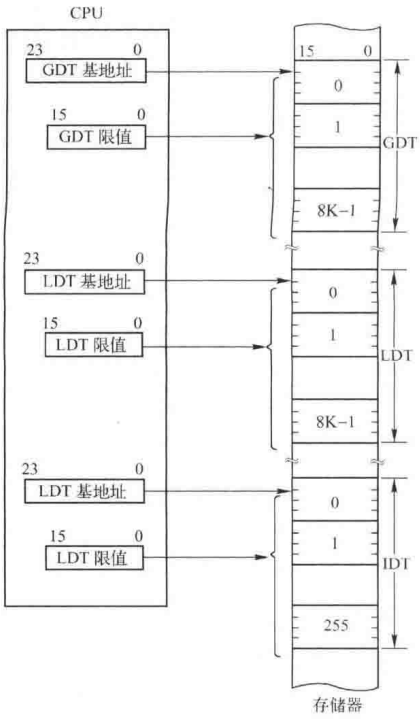


图 11-13　描述符表的内存分配

全局描述符表 GDT 中包含可供所有用户使用的描述符，实际上可包含除中断门和自陷门之外的所有其他描述符，整个系统只设置一个全局描述符表。

局部描述符表 LDT 只包含一个任务专用的描述符，实际上可包含段描述符、任务门描述符和调用门描述符。每个任务都有自己的专用 LDT，因此系统中可设置许多个局部描述符表，任何时候由 LDT 基地址和限值寄存器指向一个当前有效的局部描述符表。中断描述符表 IDT 只包含任务门、中断门和自陷门描述符，整个系统中只设置一个中断描述符表。

80286 可利用 LGDT、LLDT 和 LIDT 指令将 GDT、当前有效的 LDT 和 IDT 的基地址及限值分别装入各自的基地址和限值寄存器中。也可用 SGDT、SLDT 和 SIDT 指令把它们存入存储器，这些指令是系统中的特权指令，只有在最高特权级（0 级）操作的程序才能使用它们。为了加快在保护虚地址方式下进行地址变换的操作速度，80286 中给每个段寄存器（CS、DS、SS 和 ES）设置了一个段描述符高速缓冲存储器（Cache），其容量为 4×48 位。这是对描述符数据结构提供的一种有效的硬件支持，实际上可看成实地址方式下的段寄存器的扩展，其具体格式如图 11-14 所示。

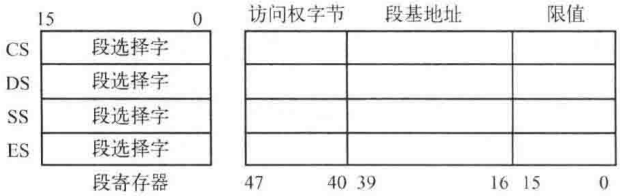


图 11-14　Cache 格式

每个段寄存器对应 Cache 中的一个单元，它由访问权字节、段基地址（24 位）和段限值（16 位）构成，每当将段选择字装入某个段寄存器时，这个段选择字所选定的段描述符中的相应字段将自动装入 Cache 的相应单元中，以后对每个段的访问就不需要再去访问段描述符以获得有关的参数，而是可直接从相应的 Cache 单元中得到，显然这样访问速度会快得多。

综上所述，80286 在保护虚地址方式下是采用段选择字和描述符数据结构来实现寻址 16 MB 的物理空间和 1000 MB 的虚存空间。这就是说，在 80286 系统中运行的程序可大大地超过 16 MB，可在 1000 MB 的虚存空间中运行。这是 20 世纪 80 年代的小型机乃至中型机才能达到的高性能。

（4）保护虚地址方式下寻址过程举例

下面通过一个具体实例来说明 80286 在保护虚地址方式下寻址的全过程。例如，在 80286 系统中，用户程序在存储器寻址时，首先得到的是虚地址指示器，它由段选择字（16 位）和偏移地址（16 位）构成，将其中的段选择字送入相应段寄存器中。与此同时，系统将根据段选择字中的 TI 和变址值字段从相应描述符表中找到指定的描述符，并将其中的访问权字节、段基地址和段限值写入 Cache 相应单元中，以后的访问只需读 Cache 即可正确寻址。

假定某用户程序中，某条指令要求从当前数据段某地址中取出 16 位操作数送 AX 寄存器，当前得到的虚地址指示器如图 11-15 所示。

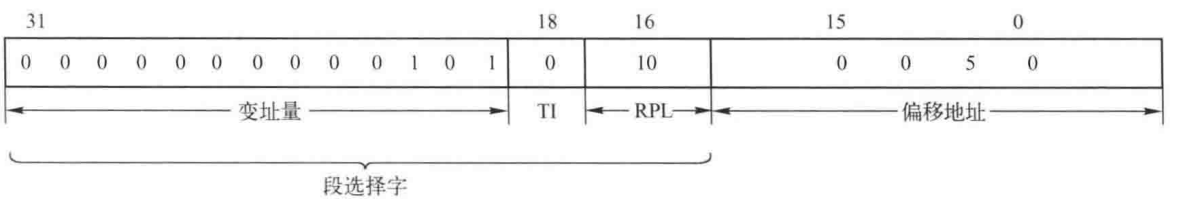


图 11-15　虚地址指示器

已知当前寻址的操作数在数据段中，故应将段选择字送入 DS 段寄存器中。同时，系统将根据 TI=0，变址值为 5，从全局描述符表中找到第 5 个描述符，具体取值如图 11-16 所示。

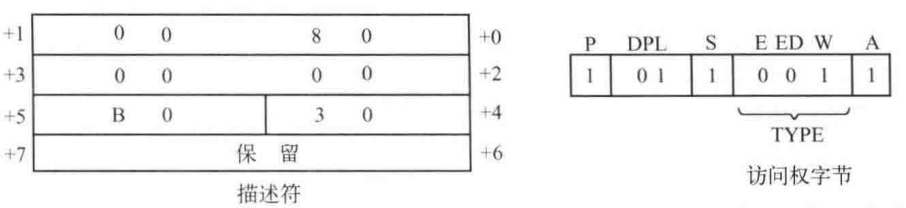


图 11-16　所寻址描述符格式

S=1，表示当前指向的是段描述符。TYPE 字段中：E=0，表示数据段描述符；ED=0，表示该数据段向上生长；W=1，表示可写入的数据段。P=1，表示该段已在物理空间中，段基址和段限值有效。DPL=01，表示该数据段特权级为 1。A=1，表示该数据段已访问过。段基地址 =300000H；段限值=0080H。

虚地址指示器给定的偏移地址 0050H<0080H，于是操作数在物理空间中的物理地址应为 300050H，CPU 将从物理地址 300050H 和 30005lH 中取得 16 位操作数送 AX 寄存器中。

上述寻址的全过程如图 11-17 所示，如此烦琐的操作过程对应用程序员是透明的，在系统控制下（实际上是在存储管理部件 MMU 控制下）自动完成。

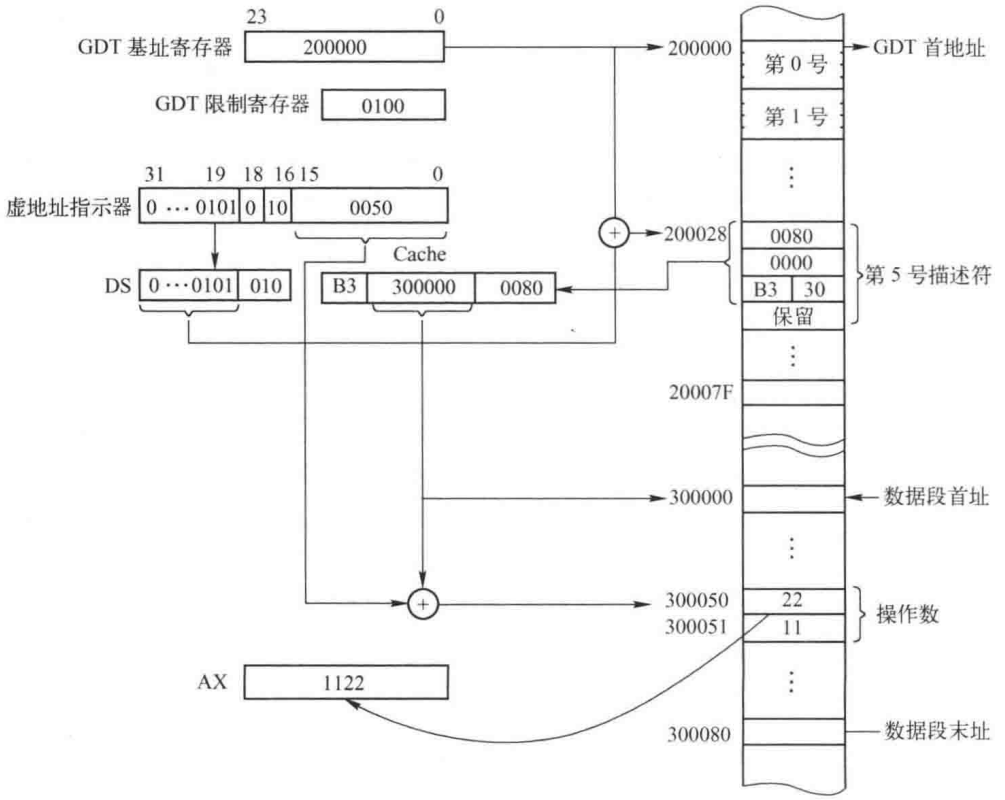


图 11-17　保护虚地址方式下寻址过程举例

操作数物理地址形成的主要过程如下：

<1> GDT 基址寄存器的内容为 200000H，指向 GDT 的首地址。

<2> 虚地址指示器的高 13 位 D31～D19 送数据段寄存器 DS 的高 13 位。

<3> DS 的高 13 位和默认为 0 的低 3 位构成 16 位的偏移量，与 GDT 基址寄存器的内容相加，得到第 5 号描述符的首地址。

<4> 取出第 5 号描述符的第 2、3 和 4 单元的内容，构成 24 位数据段基地址 300000H 并送 Cache 相应单元。这个 Cache 单元的内容即 300000H，与虚地址指示器的低 16 位 D15～D0 相加，就形成了操作数物理地址 300050H。

11.2　80286 的系统配置

80286 微处理器具有一整套支持芯片，包括时钟发生器 82284、总线控制器 82288、总线仲裁器 82289 和协处理器 80287 等。支持 8086 微处理器的数据收发器 8286/8287、地址锁存器 8282/8283 等也可应用于 80286 的微机系统。

图 11-18 是以 80286 为主体构成的微机系统，虚线部件表示可加接协处理器 80287 等，类似于 8086 的最大模式系统。如果在系统中增设总线仲裁器 80289，可与标准的多总线系统相结合，以便进一步提高系统的性能。

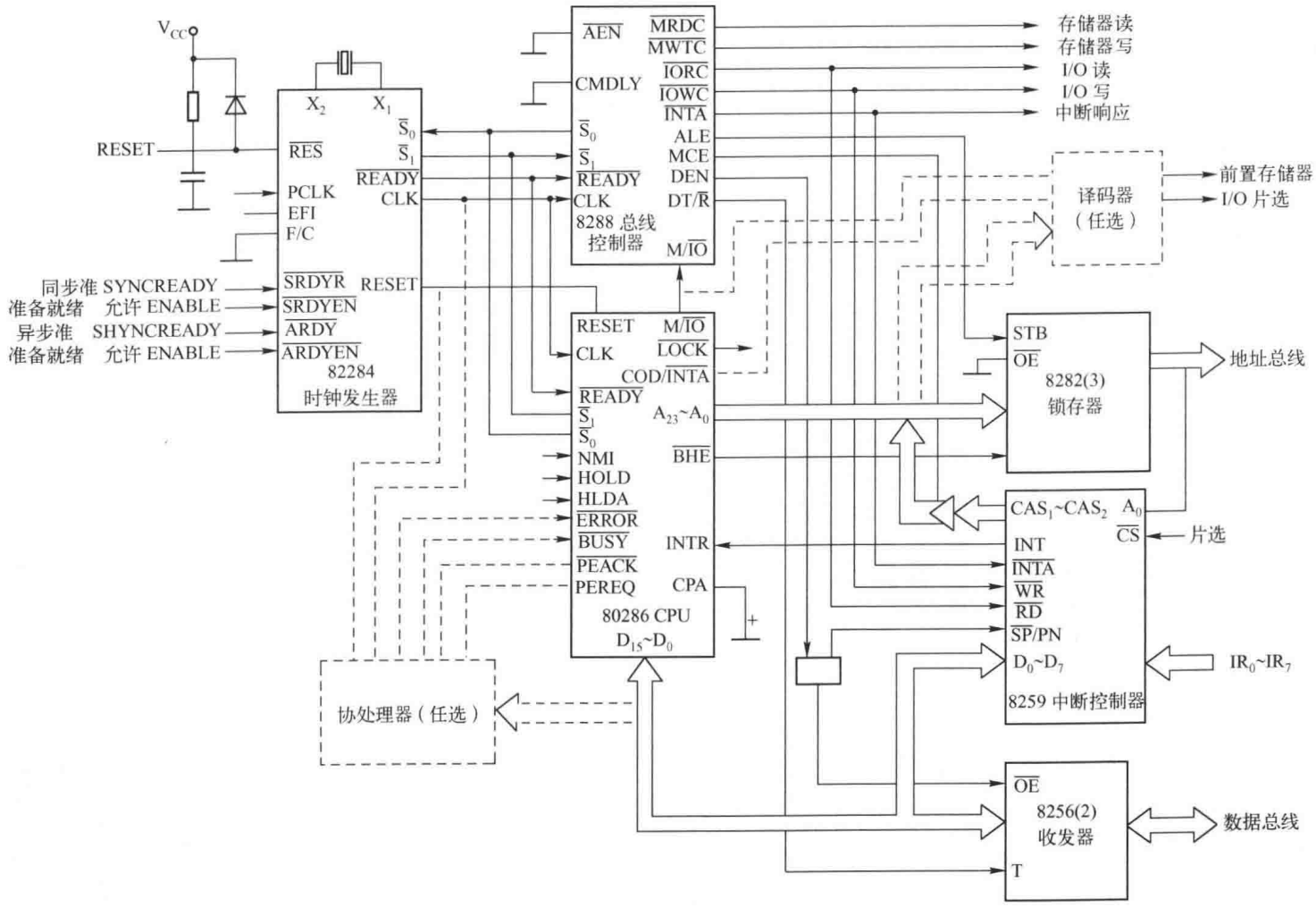


图 11-18　80286 微处理机系统举例

习 题 11

1. 80286 内部由哪几个功能部件组成？它们的主要功能是什么？
2. 8086 与 80286 比较有哪些主要区别？
3. 80286 有几个特权级？
4. 80286 访问存储器有哪两种方式，各提供多大的存储空间？
5. 简述 80286 在保护虚地址方式下的寻址过程。
6. 什么叫描述符？它们的分类情况如何？各类描述符的主要功能是什么？
7. 什么叫描述符表？为了进行存储管理，机内需设置哪些描述符表？它们各有什么特点？
8. 简述 80286 在保护虚地址方式下段寄存器 CS、DS、ES、SS 的功能。

第 12 章　80386 微处理器

本章导读

- 80386 的系统结构
- 80386 的指令系统
- 80386 的存储器扩展
- 80386 的输入/输出接口
- 80386 的异常和中断及其处理
- RISC 简介

Intel 公司于 1985 年 10 月推出的 80386（DX）是微处理器发展史上的一个里程碑，标志着微处理器进入了 32 位时代。80386 微处理器内含 275 000 个晶体管，这款 32 位处理器支持多任务设计，能同时执行多个程序。80386 是 Intel 高档微处理器的基础，要学习现代微机的组成原理必须首先了解 80386。80386 与 8086、80286 相兼容，它是为高性能的应用领域与用户、多任务操作系统设计的一种高集成度的芯片。与 8086 相比，80386 可谓复杂太多，这是因为 80386 的设计考虑了软件特别是操作系统的需要，为它们提供硬件方面的支持。

12.1　80386 系统结构

80386 及 80386 系统所涉及的知识点相当多，限于篇幅，这里只能介绍其中一些最基本的、最重要的。本节内容包括：80386 微处理器的基本结构、寄存器组成、工作模式，以及 80386 系统的构成。

12.1.1　80386 微处理器的基本结构

人们谈及 80386 时，如果不特别说明，一般是指 80386DX。80386 具有片内集成的存储器管理和保护机构，80386DX 是标准的 80386，它对内、对外的数据总线都是 32 位，地址线为 32 根，可直接寻址的 4 GB 物理地址空间，虚拟存储空间为 64 TB。与 80386DX 相对应，还有一个 80386SX。正如由 8088 CPU 填补 8 位机和 16 位机之间的空白一样，80386SX 用来填补 16 位机和 32 位机之间的空白。80386SX 的内部结构与 80386DX 一样，但是地址线减少为 24 根，对外的数据线减少为 16 根。80386 微处理器内部结构如图 12-1 所示，封装外形如图 12-2 所示。

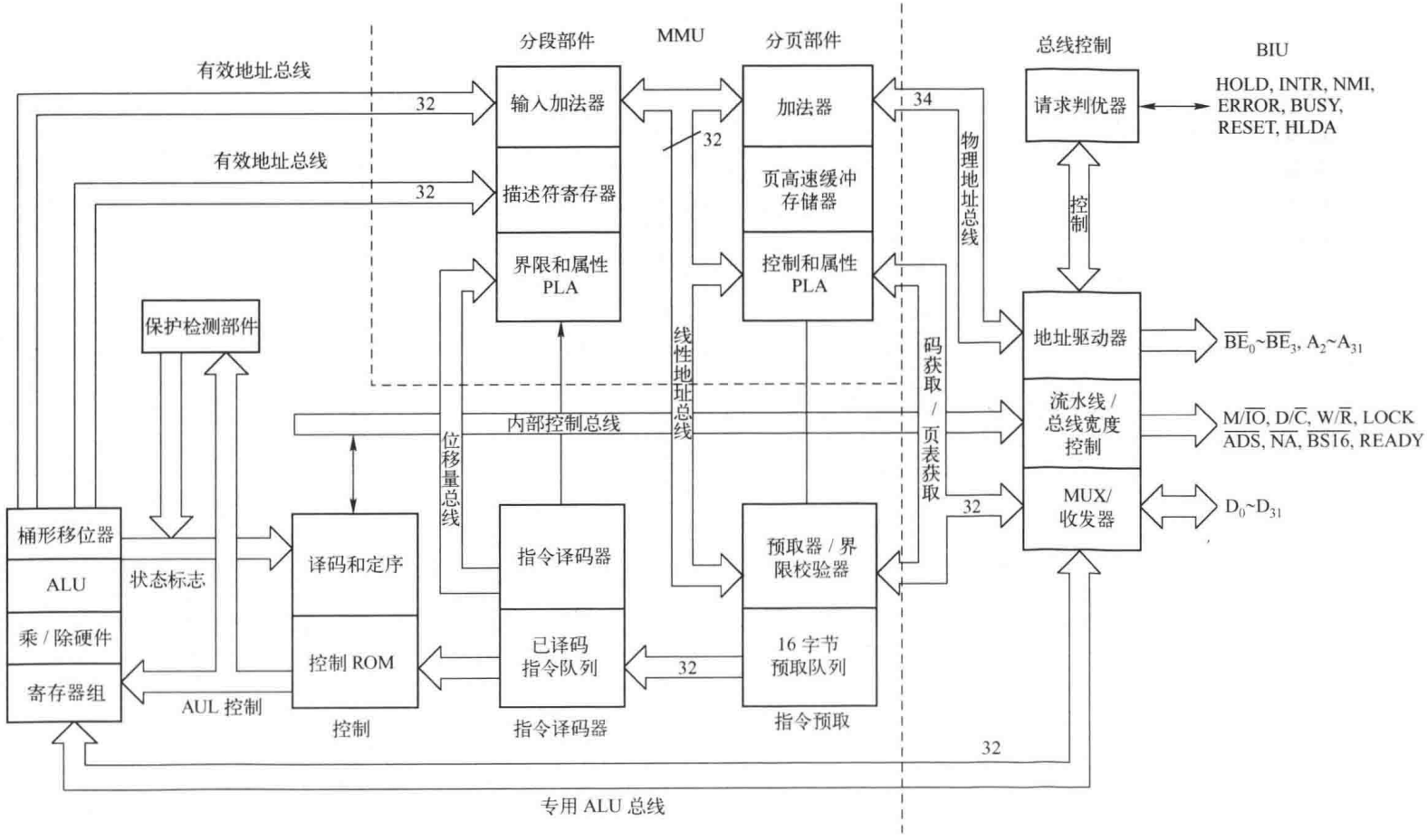


图 12-1　80386 微机处理器的功能结构

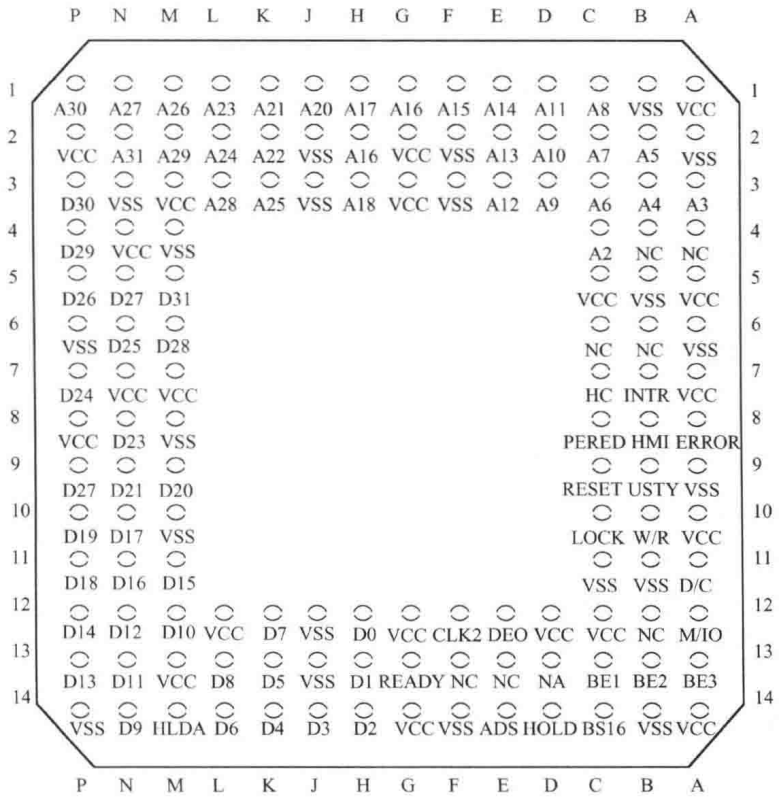


图 12-2　80386 封装外形

由图 12-2 可见，80386 由三大部件组成：中央处理部件、总线接口部件和存储管理部件。

1．中央处理部件（Central Process Unit，CPU）

CPU 包括指令预取部件（Instruction Prefetch Unit，IPU）、指令译码部件（Instruction Predecode Unit，IDU）、执行部件（Execution Unit，EU）。

IPU 包含 16 字节的指令预取队列寄存器，当总线空闲周期到来时，把指令流的 4 字节读出，存到指令预取队列寄存器中。80386 的平均指令长度为 3.5 字节，所以预取队列寄存器中一般可以存放 5 条指令。

IDU 的作用是对指令的操作码进行预译码，可以完成从指令到微指令的转换，并将其存放在已译码的指令队列中，供执行部件使用。这样可以节省取指令和译码的时间。

EU 包含 8 个 32 位的通用寄存器、1 个 64 位的桶形移位寄存器和 1 个乘/除法器。通用寄存器可以用来进行数据操作及地址计算。桶形移位寄存器能在一个时钟周期内将任何数据类型数据移动任意位，用来实现指令的移位、循环移位和位操作，也可以帮助进行乘法和其他操作。乘/除法器在 1 个周期内完成 1 位的乘除法，最快在 40 个周期内完成 32 位乘/除法。

2．总线接口部件（Bus Interface Unit，BIU）

BIU 为 CPU 与系统之间的连接提供高速接口，其功能是产生访问存储器和 I/O 端口所需的地址、数据和命令信号，这些动作能与当前任何的操作同时进行。BIU 被设计成能够接收并优化多个内部总线的请求，使其在工作时能够最大限度的利用所提供的总线宽度。

3．存储器管理部件（Memory Management Unit，MMU）

MMU 包括分段部件（Segmentation Unit，SU）和分页部件（Paging Unit，PU）。

SU 能实现有效地址的计算，完成从逻辑地址到线性地址的转换，同时完成总线周期分段的违法检查，然后将线性地址和总线周期事务处理信息发到 PU。

PU 提供对物理地址空间的管理，通过两级页面重定位机构，把由 SU 或 IPU 产生的线性地址转换成物理地址。每页为 4 KB，每段可以是一页，或者多页。PU 是 80386 芯片的新增部件，可选。若不使用 PU，80386 的线性地址即物理地址。80386 工作时，BIU 通过系统总线同外部联系，它从 MMU 接收已被选中的地址，而当 IPU 中的 16 字节的指令预取队列有部分空字节时，BIU 就会去访问存储器读出后续指令并填充指令预取队列。

预取队列中的指令代码送入 IDU，经指令译码器译码后，可按指令的执行顺序进入已译码的指令队列，其中可存放 3 条已译码的指令，等待进入 EU 去执行。

EU 所需的原始数据来自 BIU，而经过运算所得的结果将送回给寄存器或存储单元。由 EU 运算所求得的有关寻址信息送入 MMU。

12.1.2 80386 的寄存器组成

80386 的寄存器是 8086/8088 及 80286 寄存器的超集，除了将原有寄存器扩展为 32 位，还增加了一些新的寄存器。段寄存器虽然仍为 16 位，但已赋予新的内涵。

1．通用寄存器

80386 的通用寄存器如图 12-3 所示，共有 8 个 32 位通用寄存器。与 8086 相同，这些寄存器也可分别作 8 位、16 位寄存器用。如果当作 32 位寄存器使用，寄存器名前要增加字符 E，即分别为 EAX、EBX、ECX、EDX、ESP、EBP、ESI 和 EDI。

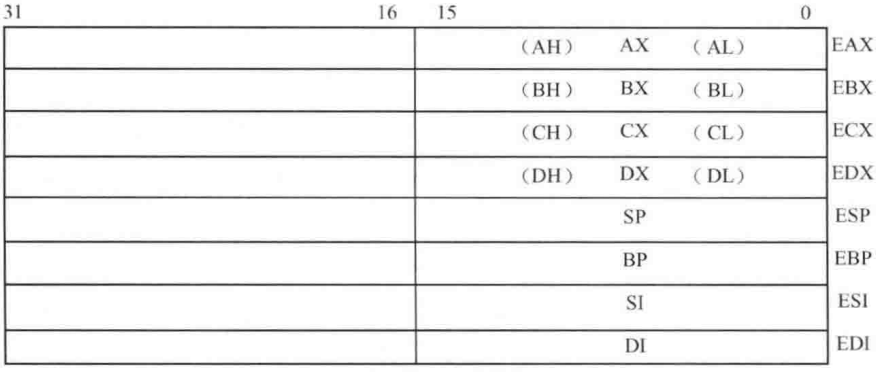


图 12-3　80386 通用寄存器

2．段寄存器和描述符寄存器

80386 为了描述每个段的基址、段限值和其他属性，为每个段定义了一个描述符。与 80286 一样，供操作系统使用和各任务公用的段描述符放在一起称为全局描述符表（GDT），而单个任务私有的所有段描述符放在一起称该任务的局部描述符（LDT）。为了说明一个段的描述符在哪个表中、序号是多少、特权级的高低，为每个段定义了一个 16 位的段选择字，如图 12-4 所示。

图 12-4　段寄存器和描述符寄存器

段寄存器有 6 个：CS、SS、DS、ES、FS 和 GS。在保护虚地址方式下，当把虚地址指示器中的段选择字装入一个段寄存器时，处理器自动将其对应的描述符装入相应描述符寄存器中。描述符的装入对程序员来说是透明的。在实地址方式下，段寄存器使用的方法与 8086 相同，即通过将段选择字左移 4 位得到段基地址，不必用描述符来说明段的性质。

3．指令指针和标志寄存器

80386 中设置一个 32 位的指令指针（EIP）如图 12-5 所示。利用 EIP 寄存器可直接寻址 4 GB（2^{32}）的实存空间。

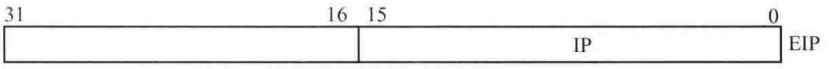


图 12-5　指令指针和标志寄存器

80386 还设置一个 32 位的标志寄存器（EFLAGS），如图 12-6 所示，其低端 16 位与 80286 标志寄存器完全相同，高端 16 位目前只设置了 2 个新的标志位：虚拟方式标志位 VM 和恢复标志位 RF。如果 VM 位置“1”，表示 80386 是在保护虚地址方式。RF 位置“1”，表示下边指令中的所有调试故障都被忽略，当成功地执行完每条指令时，RF 将被置位。

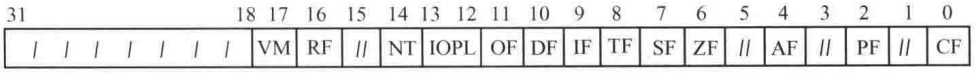


图 12-6　EFLAGS 寄存器的位结构

4．控制寄存器

80386 设置了 4 个 32 位的控制寄存器，如图 12-7 所示。CR_0 寄存器中的低 16 位为机器状态字，80386 定义了低 5 位（D_0～D_4），其中 D_0～D_3 的定义同 80286 机器状态字的含义一样，D_4 即 ET，称为协处理器类型位，用于指示出现在系统中的协处理器的类型。如果 ET=1，表示系统使用与 80387 兼容的 32 位协处理器；ET=0，表示使用与 80287 兼容的 16 位协处理器。最高位 PG 为页式管理使能位。PG=1，表示允许 80386 内部分页部件工作，否则禁止分页部件工作。CR1 寄存器保留给将来开发的 Intel 微处理器使用；CR2 寄存器中包含一个 32 位的线性地址，指向造成最后一次页故障的地址；CR3 寄存器包含页目录表的物理基地址，因为 80386 的页目录表总是在页的整数边界上，每 4KB 为一页，所以 CR3 的低端 12 位应保持为“0”。

5．系统地址寄存器

80386 中设置了 4 个专用的系统地址寄存器如图 12-8 所示。其中，GDTR 和 IDTR 占 48

位，LDTR 和 TR 占 16 位。GDTR 寄存器用来存放全局描述符表的基地址（32 位）和限值（16 位）；IDTR 寄存器用来存放中断描述符表的基地址（32 位）和限值（16 位）；LDTR 寄存器用来存放局部描述符表的段选择字；TR 寄存器用来存放任务状态段表的段选择字。可见，80386 的系统地址寄存器与 80286 的非常相似。

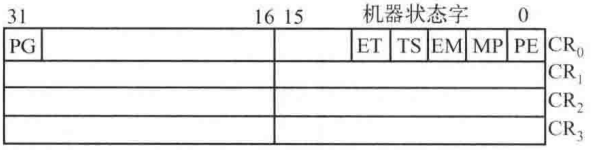


图 12-7　控制寄存器结构

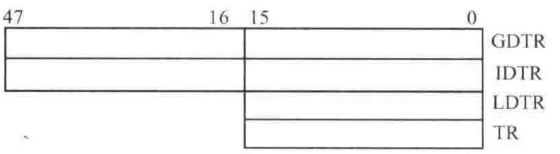


图 12-8　系统地址寄存器

6．调试寄存器组

80386 为程序员提供了 8 个 32 位的调试（DEBUG）寄存器，如图 12-9 所示。DR_7 用来设置断点；DR_6 用来保留断点状态；DR_3～DR_0 可用来设置 4 个断点；DR_4 和 DR_5 保留待用。

7．测试寄存器组

80386 设置 2 个 32 位测试（TEST）寄存器如图 12-10 所示。TR_6 用于测试命令寄存器，可对 RAM 和相联存储器进行测试；TR_7 用来保留测试后的结果。

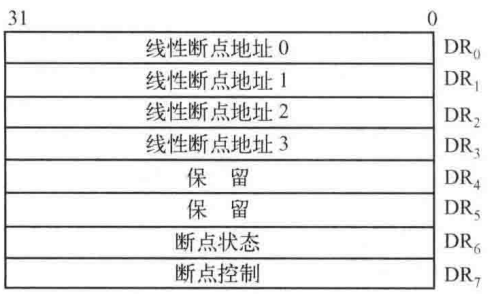


图 12-9　调试寄存器

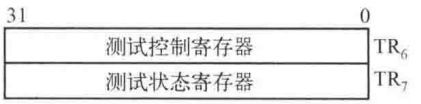


图 12-10　测试寄存器

12.1.3　80386 的存储器管理

因为 80386 所有的通用寄存器都是 32 位的（相当于 4 GB），所以用任何一个通用寄存器来间接寻址，不必分段就已经可以访问到所有的内存地址。这是不是说，在保护虚地址方式下，段寄存器就不再有用了呢？答案是否定的。虽然在寻址上不再有分段的限制问题，但在保护虚地址方式下，一个地址空间是否可以被写入，可以被多少优先级的代码写入，是不是允许执行等涉及保护的问题就出来了。要解决这些问题，必须对一个地址空间定义一些安全上的属性。段寄存器这时就派上了用途。但是由于涉及属性和保护虚地址方式下段的其他参数，要表示的信息太多了，要用 64 位长的数据才能表示。我们把这 64 位的属性数据叫做段描述符。把所有段的段描述符顺序放在内存中的指定位置，组成一个段描述符表；而段寄存器中的 16 位作为索引信息，指定这个段的属性用段描述符表中的第几个描述符来表示。这时，段寄存器中的信息不再是段地址了，而是段选择器。可以通过它在段描述符表中“选择”一个项目以得到段的全部信息。可见，80386 的保护虚地址方式与 80286 的在概念上完全一样，但 80386 在功能上有所延伸。

80386 与 80286 的一个明显区别是，80386 的内存引入了分页机制。80386 处理器把 4 KB 大小的一块内存当做一“页”内存，每页物理内存可以根据“页目录”和“页表”，随意映射到不同的线性地址上。这样可以将物理地址不连续的内存映射连到一起，在线性地址上视为连续。页表可以指定一个页面并不真正映射到物理内存中，从而访问这个页的指令会引发页异常错误。这时，处理器会自动转移到页异常处理程序中去。操作系统可以在异常处理程序中将硬盘上的虚拟内存读到内存中并修改页表重新映射，然后重新执行引发异常的指令。这样指令可以正常执行下去。

80386 有三种工作模式：实地址模式、保护虚地址方式和虚拟 8086 模式。CPU 被复位后就进入实模式，通过修改 CR0 中的控制位 PE，可使 CPU 从实模式转换到保护虚地址方式，或者从保护虚地址方式转换到实模式。通过执行 IRETD 指令，或者进行任务切换，可从保护虚地址方式转变到虚拟 8086 模式。采用终端操作，可以从虚拟的 8086 模式转变到保护虚地址方式。

1．实模式（实地址模式）

实模式下的工作原理与 8086 相同。当 80386 加电或复位后，就进入实地址工作模式。80386 所有指令在实地址模式下都是有效的，不过操作数默认长度是 16 位。物理地址形成与 8088/8086 一样，将段寄存器内容左移 4 位与有效偏移量地址相加而得到，寻址空间为 1 MB，只有地址线 A_2～A_{19} 和 $\overline{BE_0}$～$\overline{BE_3}$ 是有效的，A_{20}～A_{31} 总是低电平。唯一的例外是在复位后，在执行第一条段间转移或调用指令前，所有访问代码段的总线周期的地址 A_{20}～A_{31} 输出总是高电平，以保证执行高端内存引导 ROM 中的指令。实地址模式下段的大小为 64 KB，因此 32 位的有效地址必须小于 0000FFFFH。此模式保留了两个固定的存储区域，它们是专用的。

中断向量表区：00000H～003FFH，在 1 KB 存储空间保留 256 个中断服务程序的入口地址，每个入口地址占用 4 字节，与 8088/8086 一样。

系统初始化区：FFFFFFF0H～FFFFFFFFH，存放 ROM 引导程序。

实地址模式下的物理地址生成如图 12-11 所示。

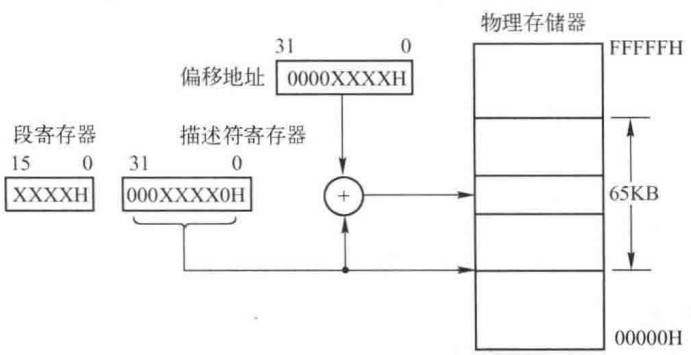


图 12-11　实地址模式下的物理地址生成

2．保护虚地址方式

保护虚地址方式是 80386 处理器的主要工作模式。在此方式下，80386 提供了复杂的存储管理和硬件辅助的保护机构，且可运行现有 8088/8086/80286 的所有软件。80386 可以寻址 4 GB 的地址空间，同时保护虚地址方式提供了 80386 的多任务、内存分页管理和优先级保护等机制。当 80386 工作在保护虚地址方式下时，其所有功能都是可用的。这时，80386 所有的 32 根

地址线都可供寻址，物理寻址空间高达 4 GB。在保护虚地址方式下，支持内存分页机制，提供了对虚拟内存的良好支持。另外，在保护虚地址方式下，80386 处理器与 80286 一样，支持优先级机制。优先级分 4 个级别（0 级～3 级），操作系统运行在最高的优先级 0 上，应用程序则运行在比较低的级别上。

DOS 操作系统运行于实模式下，而 Windows 操作系统运行于保护虚地址方式下。

（1）保护虚地址方式的地址变换

在保护虚地址方式下，某个段寄存器中的内容不再是段的基地址，32 位的段基地址是存放在一个段描述符表中，而段寄存器的内容作为选择符，即作为段描述符表的索引，用它来从表中取出相应的段描述符（包括 32 位段基地址、段界限和访问权等）。在这个地址转换过程中，由选择符的高 13 位作为偏移量，再以 CPU 内部预先初始化好的 GDTR 中的内容作为基地址，就可获得相应的描述符。该描述符将存入描述符寄存器中。描述符中的段基地址（32 位），同指令给出的 32 位偏移地址相加得到线性地址，再通过分页机构进行变换，最后得到物理地址。如果不分页，线性地址就等于物理地址，这与 80286 的保护虚地址方式下的地址变换基本一样。不过要注意两点：一是 80286 的偏移地址是 16 位的，而 80386 的偏移地址是 32 位的；二是 80286 的段界限是 16 位，而 80386 的段界限扩大到 20 位，这就意味着 80386 的寻址空间比 80286 大得多。

（2）段选择符

在保护虚地址方式下，段寄存器就成为一个选择符，由 3 个字段组成，各字段定义与 80286 相同，如图 12-12 所示，这里不再赘述。

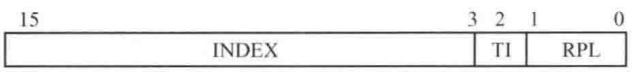


图 12-12　段选择符

（3）段描述符

80386 的段描述符格式与 80286 的段描述符格式大体相同，也是 8 个字符，包含了一个存储段的所有信息，其格式如图 12-13 所示。其中包括：段的线性基地址、段的界限（大小）。

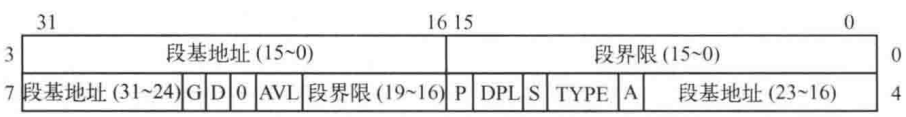


图 12-13　段描述符

段的一些属性包括：段的保护等级，读、写、执行特权和保护特权级别，操作数的默认长度（16 或 32 位），段的类型和段的粒度（即段的长度单位）。

80386 的段描述符将 80286 的两个保留字节做了新的定义。其中，段限值占最低 2 字节及 16 字节的低 4 位，构成 20 位的段限值，使逻辑段的最大容量由 64 KB 扩大为 1 MB。段基址共占 4 字节，在段描述符中分 3 处存放，它们构成 32 位的段基地址，可寻址的实存空间由 80286 的 16 MB 增大为 4 GB。

① G：粒度位，当 G=0 时，段长是以字节为单位，段的最大地址空间是 1 MB；G=1 时，段长是以页（4 KB）为单位，分段的最大地址空间是 4 GB。

② D：指示操作数和有效地址的默认长度。D=1，使用 32 位操作数和 32 位寻址方式；D=0，使用 16 位操作数和 16 位寻址方式。

③ P：表示段是否在内存中。P=1，表示该段已在内存；P=0，表示该段在硬盘交换区。

④ DPL：描述符的优先级。

⑤ S、TYPE：这两个字段说明了段的类型：数据段或堆栈段（S=1，Type 字段的 E=0），或者代码段（S=1，Type 字段的 E=1），或者系统段（S=0）。

⑥ A：段是否被访问过。A=0，段未被访问过；A=1，段已被访问过。

与 80286 一样，80386 有 8K 个全局描述符和 8K 个局部描述符，共是 16K 个描述符。但每个逻辑段的空间比 80286 大得多。当粒度位 G=0 时，每个逻辑段可达 1 MB 空间，这时 80386 可给用户提供的最大虚存空间为 1 MB×16K=16 KMB，即 16 GB 空间。而当 G=1 时，每个逻辑段可达 4 GB，这时 80386 可给用户提供的最大虚存空间为 1 GB×16K=64 TB。

保护虚地址方式下的地址变换如图 12-14 所示。由 32 位偏移地址和通过对描述符表查表得到的段基地址相加得到的地址称为线性地址，如果 80386 CPU 中的控制寄存器 CR0 的页式管理使能位 PG=0，80386 的内部分页部件不工作，这时线性地址就是物理地址。当 PG=1 时，80386 的内部分页部件工作，线性地址必须通过分页机制的转换生成物理地址。

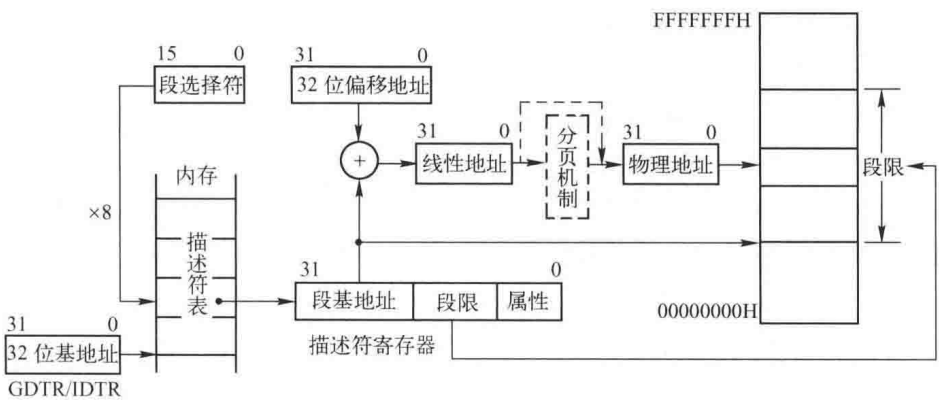


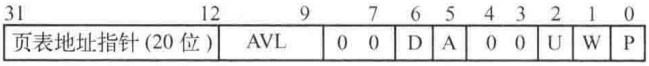
图 12-14　保护虚地址方式下的地址变换

（4）分页机制

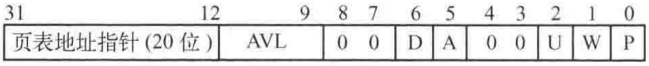
80386 分页机制管理的对象是固定大小的存储块，称为页。80386 使用 4 KB 为一页，并在 4 KB 的边界上对齐，即每页的开始地址都能被 4K 整除，这样 4 GB 的线性地址空间就划分成了 220 个页面。分页机制把整个线性地址空间和整个物理地址空间都看成由页组成，线性地址中的任何一页都可以映射到物理地址空间中的任何一页。

为了优化存储管理功能和扩大保护方式下的虚拟存储空间，80386 采用了两级页表结构，即以存储器为基础的页目录表和页表。当允许分页时，分页机制将实现两级地址转换，在低一级，由页表映射页；在较高一级，由页目录表中的一个页目录项映射页表。页目录表中包含若干个页目录描述符，页表中包含若干个页描述符，如图 12-15 所示。它们各占 4 字节，基本格式相同，区别只是 32 位中的高 20 位给定的是页表地址指针还是页地址指针，其他特征定义如表 12-1 所示。

系统中的页目录表和页表容量可根据需要设置，页目录表在存储器中的首地址由页目录基址寄存器给出（事先由系统装入），寻址页目录表可得到页目录描述符，从页目录描述符中可得到页表首地址在存储器中的位置，再寻址页表可得到页描述符，由页描述符的页地址指针给定页首地址在存储器中的位置，从而可寻址到某个页的页内存储单元。



（a）页目录描述符



（b）页描述符

图 12-15　页目录和页描述符格式

表 12-1　页目录和页描述符特征位

	定　义	1	0
P	存在位	表示该页/页表存在	表示该页/页表不存在
W	写允许位	表示该页/页表可写	表示该页/页表不可写
V	用户位	用户使用	系统使用
A	访问位	该页/页表已访问过	该页/页表未访问过
D	出错位	出错	未出错
AVL	可使用位	—	

具体来讲，在这种段页式存储结构中，首先通过段描述符获得 32 值段基址。它与虚地址指示器中的 32 位偏移地址相加，得到 32 值线性地址。然后该地址分成 3 部分，前 10 位乘 4 指向页目录表的位移量，中间 10 位乘 4 指向页表的位移量，后 12 位是所寻址的操作数在页内的偏移地址。因此，80386 系统中的页目录表和页表可含有 1K 个项。每个项为 32 位，占 4 字节，其中页目录表中的项就是页目录描述符，表示页表的首地址，页表中的项称为页描述符，表示操作数所在页的首地址。上述寻址过程如图 12-16 所示。

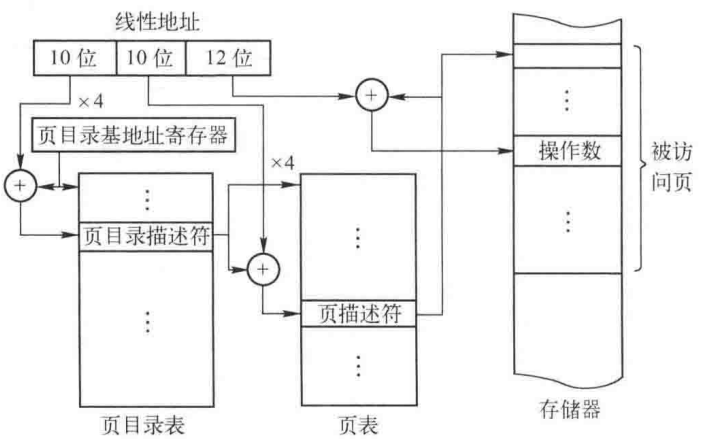


图 12-16　分应机制的寻址过程

为了加深读者的理解，这里对图 12-16 表示的寻址过程再进行如下说明。

① 线性地址的高 10 位左移 2 位，实现乘 4，其结果构成页目录表中的偏移地址。它和页目录基地址寄存器 CR3 的内容相加，构成页目录描述符的物理地址。

② 从页目录表中取出长度为 32 位的页目录描述符（其中低 12 位为 0）作为页表的首地址（也称为页表地址指针）。

③ 线性地址的中间 10 位左移 2 位，实现乘 4，其结果构成页表中的偏移地址，与页表的首地址相加，形成页描述符的物理地址。

④ 从页表中取出长度为 32 位的页描述符（其中低 12 位为 0）作为操作数所在页的首地

址（也称为页地址指针）。

⑤ 线性地址的低 12 位作为操作数的偏移地址，与页的首地址相加，从而产生操作数的物理地址。

3．虚拟 8086 模式

这是既有保护功能又能执行 8086 代码的工作模式。采用与保护虚地址方式相同的工作原理，但在程序中指定的逻辑地址可以与 8086 一样进行解释。在这种模式下，运行 8086 程序就像在 8086 CPU 上运行一样。

虚拟 86 模式是为了在保护虚地址方式下提供与 8086 处理器的兼容。虚拟 86 模式同样支持任务切换、内存分页管理和优先级，但内存的寻址方式与 8086 相同，也是可以寻址 1 MB 的空间。虚拟 86 模式是为了在保护虚地址方式下执行 8086 程序而设置的，虽然 80386 处理器已经提供了实模式来兼容 8086 程序，但此时 8086 程序实际上只是运行得快了一点，对 CPU 的资源还是独占的。虚拟 86 模式是以任务形式在保护虚地址方式上执行的，在 80386 上可以同时支持由多个真正的 80386 任务和虚拟 86 模式构成的任务。在虚拟 86 模式下，80386 支持任务切换和内存分页。在 Windows 操作系统中，有一部分程序专门用来管理虚拟 86 模式的任务，称为虚拟 86 管理程序。为了与 8086 程序的寻址方式兼容，虚拟 86 模式采用与 8086 一样的寻址方式，即用段寄存器乘以 16 作为基址，再配合偏移地址形成线性地址，寻址空间为 1 MB。但显然，多个虚拟 86 任务不能同时使用同一位置的 1 MB 地址空间，否则会引起冲突。操作系统利用分页机制将不同虚拟 86 任务的地址空间映射到不同的物理地址上去，这样每个虚拟 86 任务看起来都认为自己在使用 0～1 MB 的地址空间。

12.1.4 80386 的保护机制

80286 之前的处理器只支持单任务，操作系统并没有什么安全性可言，计算机的全部资源包括操作系统的内部资源都可以任凭程序员访问。但对于多任务的操作系统，某个捣乱的程序为所欲为会使所有程序都无法运行，所以 80286 及以上的处理器引入了优先级的概念。与 80286 一样，80386 处理器共设置 4 个优先级（0 级～3 级）。0 级是最高级（特权级），3 级是最低级（用户级），1 级和 2 级介于它们之间。特权级代码一般是操作系统的代码，可以访问全部系统资源；其他级别的代码一般是用户程序，可以访问的资源受到限制。

80386 采用保护机制主要为了检查和防止低级别代码的越权操作，如访问不该访问的数据、端口以及调用高优先级的代码等。保护机制比较复杂，由若干几方面组成，包括对段的类型检查、页的类型检查、访问数据时的级别检查、控制转移的检查、指令集的检查和 I/O 操作的保护。在一般情况下，对数据操作而言，优先级高的代码可以访问优先级低的数据段，而优先级低的代码不能访问优先级高的数据段；对调用程序而言，优先级高的代码可以调用优先级低的代码，而优先级低的代码不能随意转移到优先级高的代码中。

图 12-17 是一个软件系统，其操作系统核心的代码段和数据段安排在特权级 0，操作系统其余部分的代码段和数据段工作安排在特权级 1，而应用程序的代码段和数据段工作在特权级 3。从图中可以看出工作在特权级 3 应用程序的代码段只能访问在特权级 3 中的数据，只能在特权级 3 的程序之间实现转移（用实线表示）；特权级 3 的代码段不能访问工作在特权级 0 和 1 的操作系统的代码段和数据段以及特权级 2 的代码段和数据段（用虚线表示）。

图 12-18 是特权级 1 中的代码可以访问的范围。从图中可以看出，工作在特权级 1 的代码不仅可以访问特权级 1 的数据，实现特权级 1 中代码之间的转移，也可以访问特权级 3 的数据，调用特权级 3 的代码（用实线表示）。

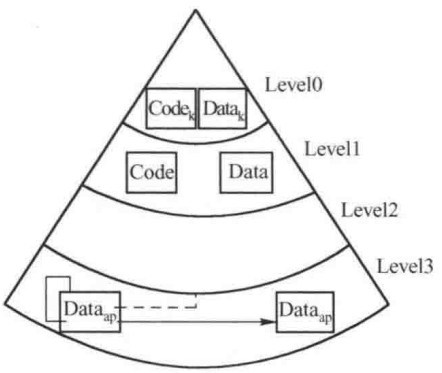


图 12-17　特权级 3 访问的范围

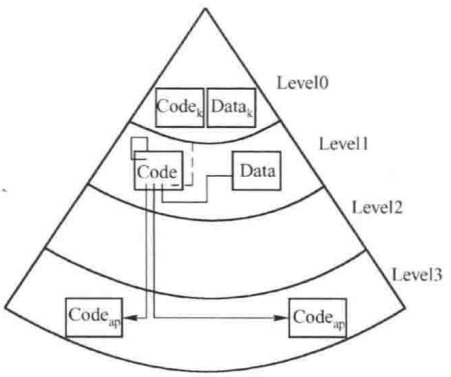


图 12-18　特权级 1 访问的范围

12.1.5　80386 系统组成

80386 基本系统组成的原理框图如图 12-19 所示。

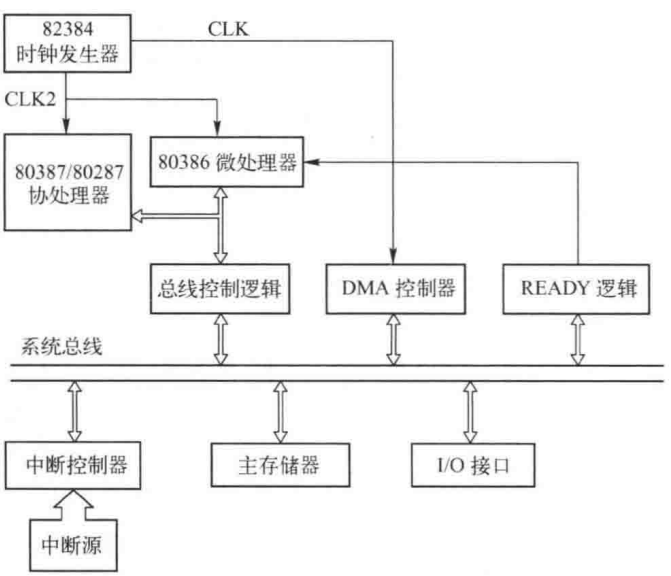


图 12-19　80386 的基本组成

1．各单元功能介绍

（1）82384 时钟发生器

82384 时钟发生器为 80386 及其支持芯片提供时钟信号，输出两种时钟信号：CLK2 和 CLK。CLK2 是 CLK 的倍频。CLK2 作为 80386 的时钟输入，80386 内部将其二分频产生内部时钟（CLK）。

82384 输出的 CLK 与 80386 内部工作时钟 CLK 在相位上一致，可作为与 80386 内部 CLK 同步的时钟信号，提供给其他电路。

（2）协处理器

协处理器可帮助微处理器实现浮点数运算和高精度整型数运算。与 80386 相配的协处理器可以是 80387 或 80287。协处理器在设计时都考虑方便与微处理器的连接，80387 针对 80386 设计，而 80287 是针对 80286 设计的。因此，80386 与 80387 的连接简单，只需将相关信号线直接相连。

（3）总线控制逻辑

总线控制逻辑对 80386 输出的总线周期定义信号（如 $M/\overline{IO}$ 、$D/\overline{C}$ 和 $W/\overline{R}$ ）译码，产生相应周期的操作命令（如存储器读命令 $\overline{MRDC}$ 、存储器写命令 $\overline{MWTC}$ 、I/O 读命令 $\overline{IORC}$ 、I/O 写命令 $\overline{IOWC}$ 和中断响应信号 $\overline{INTA}$ 等）以及控制信号（数据收发控制信号 $DT/\overline{R}$ 、数据传送允许信号 DEN 和地址锁存信号 ALE）。总线控制逻辑的作用以及所产生的信号与 8086 最大模式系统中的总线控制器 8288 相似，不同之处在于，没有单独设计一种类似 8288 的芯片，而通常由可编程阵列逻辑（Programmable Array Logic，PAL）元件实现。此外，用于延长总线周期的 READY 逻辑也可安排在 PAL 中。

（4）中断控制器

在 80386 系统中，也采用 8259A 或与之相当的逻辑来管理外部硬件中断。

（5）DMA 控制器

DMA 控制器用来控制内存与 I/O 设备之间的直接数据传输。80386 采用 DMA 方式进行数据传输的典型 I/O 设备是硬盘和软盘驱动器。

2．82380 多功能芯片

在早期的 80386 系统中，中断控制逻辑、DMA 控制器等控制逻辑分别采用独立的高集成度芯片，后来出现了控制芯片组。这里仅看一个高集成度芯片的例子 82380。一片 82380 中包括以下逻辑：

- 32 位的 8 通道 DMA 控制器。
- 与 3 片 82C59A 相当的中断控制器。
- 与 4 片 82C54 相当的可编程计数器/定时器。
- DRAM 刷新控制器（含 24 位的刷新地址计数器和判断逻辑）。
- 其他，如可编程的 READY 信号产生器。

因此，图 12-19 中的中断控制逻辑、DMA 控制器以及 READY 逻辑用一片 82380 即可实现。需要指出，图 12-19 仅是 80386 系统组成的原理示意图，还有一些重要的组成部分没有被反映出来，如高速缓冲存储器及其控制器。在早期的 80386 系统中，高速缓存控制器一般采用 82385，它可控制容量为 32KB 的高速缓存。

12.2 80386 的指令系统

80386 指令系统全面兼容了 8086、80286 系统，并增加了一些针对 32 位数据传送与处理、内存管理等指令。

12.2.1 80386 的寻址方式

80386 的寻址方式包括立即数方式、寄存器方式和存储器寻址方式三大类。寄存器方式操作数在某一个 8 位、16 位或者 32 位通用寄存器之中。立即数方式操作数作为操作码的一部分被包括在指令中。存储器寻址方式操作数在某一个存储单元之中。存储单元的线性地址由段地址和偏移量（有效地址）组成，段地址由某个段寄存器确定，偏移量是通过位移量、基址、变址和比例因子四种地址元素的任意组合相加而计算出来。通常情况下，偏移量（有效地址）的计算公式为：

EA=基址寄存器+(变址寄存器×比例因子)+位移量

除了保留 8086 的寻址方式，80386 还提供了下面几种寻址方式。

① 带比例因子的变址寻址。在这种方式中，变址寄存器的内容乘以比例因子，乘积再加上位移量形成操作数的有效地址（EA）。例如：

```
MOV     EAX, [ESI*4]
```

② 带比例因子的基址变址寻址。在这种方式中，变址寄存器的内容乘以比例因子，乘积加上基址寄存器的内容，形成操作数的有效地址（EA）。例如：

```
MOV     EAX, [EBX] [ESI*4]
```

③ 带位移量，带比例因子的基址变址寻址。在这种方式中，变址寄存器的内容乘以比例因子，乘积再加上基址寄存器的内容和位移量，形成操作数的有效地址（EA）。例如：

```
MOV     EAX, LOCALTABLE [ESI*4] [EBX+40H]
```

12.2.2 80386 的指令系统

在与 8086 全面兼容的同时，80386 指令系统增加了 32 位操作、系统管理等指令，下面就常用的一些指令作简要的介绍。

1．数据传输指令

传输指令将字节、字、或者双字在寄存器与寄存器之间，寄存器与存储器之间，存储器与存储器之间进行传输。这部分内容基本与 8086 一致，我们重点讨论 80386 的独有指令。

（1）传输指令

MOV 指令要求两个操作数具体相同的尺寸，不过在 80386 编程中，若尺寸大小不同，80386 提供了零扩展传输指令和带符号扩展传输指令，即 MOVZX 和 MOVSX 两条指令。

例如：

```
MOVZX     AX, BL              ; 将 BL 的内容零扩展成 16 位送入 AX
MOVSX     EAX, BX             ; 将 BX 的内容带符号扩展成 32 位送入 EAX
```

（2）交换指令

与 8086 的基本一样，这里不再赘述。

（3）I/O 口指令

与 8086 的基本一样，这里不再赘述。

（4）堆栈操作指令

与 8086 相比，80386 增加了全部 16 位或者 32 位寄存器进栈和出栈指令。

PUSHA 和 PUSHAD 指令是压栈全部 16 位或者 32 位寄存器的指令。

POPA 和 POPAD 则是全部 16 位或者 32 位寄存器的出栈指令。

（5）标志传送指令

80386 增加的指令如下：

PUSHFD 指令压入某双字入栈，并送所有的标志位进入这个字的指定位。

POPFD 指令从堆栈移出双字，并送这些位入标志寄存器。

（6）换码指令

与 8086 的基本一样，这里不再赘述。

（7）目标地址指令

与 8086 的基本一样，这里不再赘述。

2．算术运算指令

算术运算指令包括加法、减法、乘法、除法操作以及十进制加减法的结果调整等，与 8086 的基本一样，这里不再赘述。

3．位操作指令

位操作指令包括逻辑运算指令、移位操作、位测试指令。80386 主要增加的指令如下。

（1）移位指令

80386 增加了 32 位移位操作，如 SHLD 指令是双精度左移指令，SHRD 指令是双精度右移指令。各移位指令的移位方式如图 12-20 所示。

（a）SHLD 双精度左移指令　　（b）SHRD 双精度右移指令

图 12-20　32 位移位指令

（2）位测试指令

位测试指令用于测试目的操作数的单个位，该位的位号由指令中源操作数指出，源操作数可以是寄存器操作数也可以是立即数。

BT 指令将指定位送入 CF，但是该位的值并不改变。

BTS 指令将指定位送入 CF，并将该位的值置为 1。

BTC 指令将指定位送入 CF，并将该位的值取反。

BTR 指令将指定位送入 CF，并将该位的值清零。

BSF 指令源操作数从低位往高位进行扫描，并将扫描到的第一个“1”的位序送入目标寄存器中。如果被扫描数为全“0”，则置 ZF 标志为“1”，否则 ZF 标志清零。

BSR 指令功能与 BSF 类似，只是扫描时从高位到低位进行。

4．串操作指令

（1）位串操作指令

IBTS 指令是位串写指令。IBTS 指令有 4 个操作数：位串的基地址，要插入的子串起始位的偏移量，子串的长度，获得插入值的寄存器。基地址可以是寄存器或者存储器地址，偏移量包含在(E)AX 中，子串的长度由 CL 给出，最后一个是要插入的通用寄存器。例如：

```
IBTS    BX, AX, CL, DX
```

XBTS 指令是位串读指令。XBTS 指令也有 4 个操作数：位串的基地址，子串起始位的偏移量，子串的长度，保存抽取的子串的寄存器。例如：

```
XBTS    BX, AX, CL, DX
```

（2）串输出指令

OUTB/OUTW/OUTD 指令按字节/字/双字格式，将 DS:(E)SI 指定的串元素输出到由 DX 寄存器内容指令的 I/O 端口。

（3）串输入指令

INSB/INSW/INSD 指令将由 DX 寄存器内容指定的 I/O 端口，按字节/字/双字格式输入到由 ES:(E)DI 指定串存储单元。

5．程序控制转移指令

与 8086 的基本一样，这里不再赘述。

6．处理器控制指令

与 8086 的基本一样，这里不再赘述。

7．系统寄存器的装入与存储指令

系统寄存器包括控制寄存器（CR）、调试寄存器（DR）和测试寄存器（TR）。通常，这些寄存器只有具有最高特权级的程序才能使用这些寄存器存取指令。

LMSW 指令将操作数装入 CR_0 中的 0～15 位中。

SMSW 指令将 CR_0 中的 0～15 位存入操作数所指定的通用寄存器或存储单元中。

SIDT 指令将 IDTR 中的 48 位数存入操作数所指定的 6 字节单元中。

LGDT 指令将操作数所指定的 6 字节单元内容装入全局描述符表寄存器 GDTR 中。

SGDT 指令将 GDTR 中的 48 位数存入操作数所指定的 6 字节单元中。

LLDT 指令将操作数所指定的 16 位选择符装入局部描述符表寄存器 LDTR 中。

SLDT 指令将 LDTR 中的 16 位选择符存入到操作数所指定的地址单元中。

LTR 指令将操作数所表示的任务状态段选择符装入到任务寄存器 TR 中，并自动将任务状态段描述符装入到内部相应的 64 位的描述符寄存器中。

STR 指令将 TR 的 16 位选择符到操作数所指定的双字节寄存器或内存单元中。

8．条件设置字节指令

SET 指令根据 80386 定义的 16 种条件中的任意一种，设置所选择的字节，如表 12-2 所示。SET 指令唯一的操作数是一个字节的寄存器或存储器，若条件成立，则设置操作数为“1”，否则设置为“0”。

9．保护属性检查指令

保护属性检查指令只能在保护虚地址方式下使用：

LAR 指令为装入访问权限指令。

LSL 指令为装入段限制指令。

VERR 指令作用为验证段的可读性。

表 12-2　SET 指令

指　令	含　义	指　令	含　义	指　令	含　义
SETO	溢出	SETBE	低于或等于	SETL	小于（有符号）
SETNO	无溢出	SETNA	不高于	SETNGE	不大于或等于
SETB	低于（无符号）	SETNBE	不低于或等于	SETNL	不小于
SETNAE	不高于或等于	SETA	高于	SETGE	大于或等于
SETNB	不低于	SETS	负	SETLE	小于或等于
SETAE	高于或等于（无符号）	SETNS	非负	SETNG	不大于
SETE	相等	SETP	偶	SETNLE	不小于或等于
SETZ	等于零	SETPE	偶	SETL	小于（有符号）
SETNE	不相等	SETNP	奇	SETG	大于（有符号）
SETNZ	非零	SETPO	奇	—	

VERW 指令作用为验证段的可写性。

ARPL 指令作用为调整选择符的特权级。

10．高级语言指令

BOUND 指令作用为检查指定 16 位或 32 位寄存器数值是否在第二操作数所指定的两个存储器的界限内。

ENTER 指令的作用是为过程参数建立一个堆栈区。

LEAVE 指令的作用是撤销 ENTER 指令的动作。

12.3　80x86 典型微处理机介绍

12.3.1　80486 CPU

80486 CPU 是在 80386 CPU 的基础上改进并发展起来的 32 位机，内部寄存器和数据总线宽度都是 32 位，地址总线也是 32 位，使得可以寻址的内存空间达到 4 GB，虚拟内存空间达到 64 TB。80486 CPU 将浮点运算部件（FPU）、8KB 高速缓冲（Cache）存储器等集成在一块芯片内，极大地提高了微处理机的处理速度。80486 CPU 采用了若干新技术，如 RISC 技术、突发式总线技术等，使微处理机的性能大大提高。

1．80486 CPU 的内部结构

80486 CPU 内部集成了 120 万只晶体管，如图 12-21 所示，工作频率可达 100 MHz 以上。

内部的 FPU（Floating Point Unit，浮点运算部件）是对浮点数进行运算的。浮点数由阶码和尾数两部分组成，阶码是定点整数形式，尾数是定点小数形式，这两部分执行的操作不同。80486 CPU 的 FPU 配有一个用来解释 IEEE 标准规定的单精度数（32 位）、双精度数（64 位）和扩展精度数（80 位）的数据格式的专用硬件，因此能够全面支持 IEEE 标准规定的 32 位、64 位及 80 位数据格式。80486 内部的 FPU 增加了正弦和余弦函数，且能与整数部件并行运行，比 80386 CPU 外部的 80387 协处理器性能高出 4 倍。

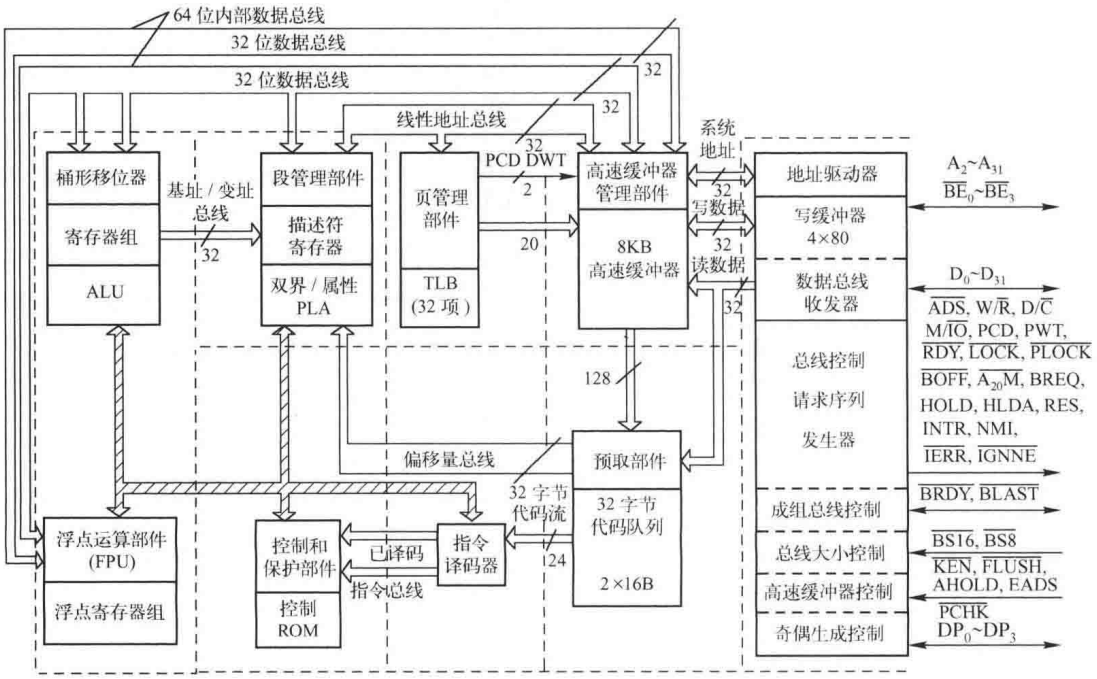


图 12-21　80486 CPU

MMU（Memory Management Unit，存储管理部件）包含一个分段部件和一个分页部件。分段部件为应用程序和操作系统相互隔离提供保护，通过提供对数据和代码的再定位与全局资源共享的方法，管理存储器的逻辑地址。存储器组成一个或多个可变长度的段，每段的存储容量最大可达 4 GB。段具有位置、长度、类型（堆栈、数据、代码）和保护等特性。分页部件在分段之下操作，分页是可选的，并且可用系统软件将其禁止，每页固定为 4 KB，每段可分成一个或多个 4 KB 的页面。在 80486 CPU 上运行的每个任务最大可具有 16381 段，因此每个任务最大可具有 64 TB 的虚拟地址空间。

片内有 32 字节指令队列，指令部件通过总线接口部件，顺序预先取出几条指令放在指令队列中，采用 RISC 技术，执行 5 级流水线操作，提高了处理机的速度。片内的 8 KB 高速缓存（Cache）用来存储常用的数据和指令，以减少对外部总线的访问，突发式总线技术使高速缓存能够快速填充。

2．80486 CPU 的寄存器

80486 CPU 的寄存器可分为 4 类：基本结构寄存器，系统级寄存器，浮点寄存器和调试/测试寄存器。

（1）基本结构寄存器

这类寄存器因功能不同又可分为以下 4 种。

① 通用寄存器

80486 CPU 内部有 8 个 32 位的通用寄存器，这些寄存器用来保存数据或存储器的部分地址量，可以支持 8 位、16 位、32 位的数据操作和 16 位、32 位的地址操作。这些寄存器分别命名为 EAX，EBX，ECX，EDX，ESI，EDI，ESP，EBP。根据它们的命名可知其用途：

- EAX，Accumulator Register（累加器寄存器）。
- EBX，Base address Register（基地址寄存器）。

❖ ECX，Counter Register（计数寄存器）。

❖ EDX，Data Register（存放数据寄存器）。

❖ ESI，Source Index Register（源变址寄存器）。

❖ EDI，Destination Index Register（目标变址寄存器）。

❖ ESP，Stack Pointer Register（堆栈指针寄存器）。

❖ EBP，Base Pointer Register（基址指针寄存器）。

当它们作为 16 位寄存器时，仅用低 16 位，8 个 16 位寄存器命名为 AX，BX，CX，DX，SI，DI，SP，BP，其用途与作为 32 位寄存器时相同。寄存器 AX，BX，CX 和 DX 又可用于 8 位数据操作。这 4 个 16 位的寄存器被拆开使用，拆开后的寄存器可以单独访问，它们被命名为 AH，AL，BH，BL，CH，CL，DH，DL。

通用寄存器的结构如表 12-3 所示。

表 12-3　80486 CPU 通用寄存器结构

	D_{31} … D_{16}	D_{15} … D_8	D_7 … D_0	
EAX		AH	AL	AX
EBX		BH	BL	BX
ECX		CH	CL	CX
EDX		DH	DL	DX
ESI				SI
EDI				DI
ESP				SP
EBP				BP

80486 CPU 为了维持与 8088 及 8086、80286 等 8 位和 16 位微处理器的向上兼容，其寄存器结构可允许 8 位、16 位和 32 位操作数的操作。其中 16 位寄存器的名称（AX～DX，SI，DI，SP，BP）就是 8086 CPU 的 8 个 16 位通用寄存器名；而 8 个 8 位的寄存器名（AH～DL）则与 8086 的寄存器名相同，用法也一样。

对 16 位和 8 位操作数进行操作只影响相应 16 位和 8 位名称的寄存器。例如在做 8 位操作数加法运算时，从 D7 位向前的进位并不是进入 D8 位，而是影响标志寄存器 EFLAGS 中的进位标志；同样，在做 16 位操作数运算时，其进位或借位不是对 D16 位操作，而是使进位标志置 1。

② 段寄存器

80486 CPU 中有 6 个 16 位段寄存器，分别称为代码段寄存器（CS）、堆栈段寄存器（SS）、数据段寄存器（DS、ES、FS 和 GS）。当 80486 CPU 访问存储器时，若工作在保护模式，则段寄存器用来存放段选择符，由选择符找到段描述符。段描述符中说明该段的位置、界限、大小、保护权限等属性。一个存储器的地址由段基址和有效偏移地址组成，80486 CPU 中的偏移地址是 32 位而不是 8086 CPU 中的 16 位。

段寄存器 CS 与指令指针寄存器（EIP）的内容作为偏移值，共同指明下一条要执行的指令所在存储器的地址。在发生分段之间的控制转移指令（如 CALL、JMP），中断和异常事故时，CS 的值要发生改变。

CPU 在执行子程序调用和过程参数传递时，需要把存储器的一个区域规定为堆栈。所有的堆栈操作要用段寄存器 SS 来分配堆栈。

80486 CPU 中定义了 4 个数据段寄存器，每个数据段寄存器都可被当前执行的程序用来寻址，这样可访问 4 个分开的数据区，从而使程序可以有效地访问不同类型的数据结构。

③ 指令指针寄存器 EIP

EIP 是一个 32 位寄存器，在该寄存器中存放的是下一条要执行指令的偏移地址。当 80486 系统不工作在保护模式下时，仅用低 16 位（寄存器名为 IP）。这个地址偏移量和代码段寄存器（CS）的段基址，形成下一条指令所在存储器的地址。

④ 标志寄存器 EFLAGS

标志寄存器是一个 32 位寄存器，它用来存放 80486 CPU 的状态、控制、系统标志等信息。80486 CPU 中定义了 14 种标志，而 8086 CPU 中仅定义了 9 种标志。

- ❖ 状态标志：当 80486 CPU 执行有关运算指令时，其运算结果对状态标志进行修改，这些标志 CF、PF、AF、ZF、SF 和 OF 能够反映运算状态。这 6 个状态标志与 8086 CPU 中完全相同。
- ❖ 控制标志：控制标志也与 8086 CPU 中的一样，有 DF、IF、TF。
- ❖ 系统标志：指将这些标志置“1”或清“0”时，系统可按要求操作。系统标志有 5 个。
- ❖ AC（Alignment Check Flag）：对准校验标志。若 AC=1，80486 CPU 允许对没有对准的数据进行对准检查，发现在进行存储器操作时出现没按边界对准情况，就发生数据访问异常事故；反之，则不检查。
- ❖ VM（Virtual Mode）：虚拟 8086 模式标志。当 VM 置“1”时，表示该任务是执行一个 8086 程序。
- ❖ RF（Resume Flag）：恢复标志，用来暂时“禁止使能”调试异常事故，所以一条指令可以在一个调试异常事故以后重新启动，而不致引起另一个调试异常事故。
- ❖ NT（N Task）：嵌套任务标志，控制被中断和被调用的任务的级链，它会影响 IRET 指令的操作。
- ❖ IOPL（Input /Output Privilege Level）：占 2 位，输入/输出特权级标志。

80486 CPU 的 EFLAGS 寄存器结构如图 12-22 所示。注：“×”为随意态，即不受其影响。

31…19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	AC	VM	RF	×	NT	IOPL	OF	DF	IF	TF	SF	ZF	×	AF	×	PF	×	CF	

图 12-22　80486 CPU 的 EFLAGS 寄存器结构

（2）系统级寄存器

系统级寄存器主要用来控制 CPU 芯片上的浮点运算部件（FPU）和超高速缓存（Cache），以及分段、分页机构。这些寄存器仅可由在特权级 0（最高特权级）上运行的程序访问。

系统级寄存器包括 4 个 32 位的控制寄存器：CR_0，CR_1，CR_2 和 CR_3（CR_1 保留，供 Intel 处理器使用），以及 4 个系统地址寄存器。

① 系统控制寄存器

CR_0 寄存器的最高位 D_{31} 位——PG（Paging Enable）允许分页。若 PG=1，允许片内分页部件工作；否则，禁止分页部件工作。

CR_3 控制寄存器是页目录基地址寄存器，保存着页目录表的物理基地址。由于 80486 的页目录表是按页排列的，所以低端 12 位中的 10 位不起作用，即便写上了内容，也不会被理睬。

② 系统地址寄存器

80486 CPU 配备 4 个系统地址寄存器，保存着操作系统需要的保护信息和地址转换信息。这 4 个专用寄存器用来引用 80486 CPU 在保护方式下所需要的表或段。也就是说，系统地址寄存器只能在保护方式下使用，所以又称为保护方式寄存器。80486 CPU 用这 4 个寄存器把在保护方式下时常使用的数据结构的基地址、限界以及相关属性保存起来，以确保其快速性。

这 4 个系统地址寄存器分别为 GDTR、IDTR、LDTR 和 TR。

GDTR（Global Descriptor Table Register，全局描述符表寄存器）：48 位寄存器，用来保存全局描述符表 GDT 的 32 位线性基地址和 16 位的界限。全局描述符表中包含系统中的所有任务都可使用的描述符，它们有代码段描述符、数据段描述符、系统控制描述符等。

IDTR（Interrupt Descriptor Table Register，中断描述符表寄存器）：48 位寄存器，用来保存中断描述符表（IDT）的 32 位线性基地址和 16 位的界限。其实，80486 CPU 为每个中断或异常都定义了一个中断描述符，中断描述符表中的内容有中断或异常服务程序的首地址等属性。IDT 中最多有 256 个描述符，至少有 256 个字节，因为 Intel 公司保留有 32 个中断描述符。系统中所使用的每个中断在 IDT 中都必须有一项。IDT 中的描述符是通过 INT 指令、外部中断和内部发生的异常来访问的。

LDTR（Local Descriptor Table Register，局部描述符表寄存器）：16 位寄存器。描述符表中包含与一个给定的任务有关的描述符。通常，操作系统的设计者使每个任务都各自有一个 LDT。LDT 可能只包含代码、数据、堆栈、任务码等的描述符。使用 LDT 这样的数据结构，就可以使给定任务码、数据等与操作系统和别的任务相隔离。

TR（Task Register，任务状态寄存器）：16 位寄存器。与 LDT 一样，每个任务都有自己的任务状态段 TSS（Task State Segment），在进行特权级之间的控制转移时，要对 SS 寄存器和 ESP 寄存器进行修改，这就需要通过 TSS 读取它们的修改值。

因为在 80486 CPU 中，LDTR 和 TR 只有一个，而 LDT 和 TSS 是每个任务一个，因此在任务切换时，要对 LDTR 和 TR 进行修改。

3．80486 CPU 的外部引脚信号

80486 CPU 系列因工作频率不同有几种型号，封装引脚个数不同。下面仅就引脚的功能做介绍，图 12-23 为引脚功能信号。

（1）地址总线及地址屏蔽

80486 CPU 的地址总线分成两部分：A_{31}～A_2 提供高位地址，用来选择能被 4 整除的字节地址，字节允许信号 $\overline{BE_0}$ ～ $\overline{BE_3}$ 构成低位地址。其选择关系如表 12-4 所示。

（2）数据总线及宽度控制

80486 CPU 拥有 32 条数据线，可以传送 8 位、16 位或 32 位不同宽度的数据。$\overline{BS8}$ 和 $\overline{BS16}$ 为总线定宽引脚，使 80486 可运行于多总线周期，解决单个周期中不能提供或不能接收 32 位数据的设备的请求，CPU 在每个时钟周期都取样这些信号。

（3）控制总线

任何型号的 CPU 的外部引脚都可分为三大总线：地址总线（Address Bus，AB）、数据总线（Data Bus，DB）和控制总线（Control Bus，CB）。相对来讲，AB 和 DB 比较简单，也容易理解和记忆，不同型号的 CPU 所拥有的 AB 和 DB 数量不同，所谓 8 位机、16 位机、32 位

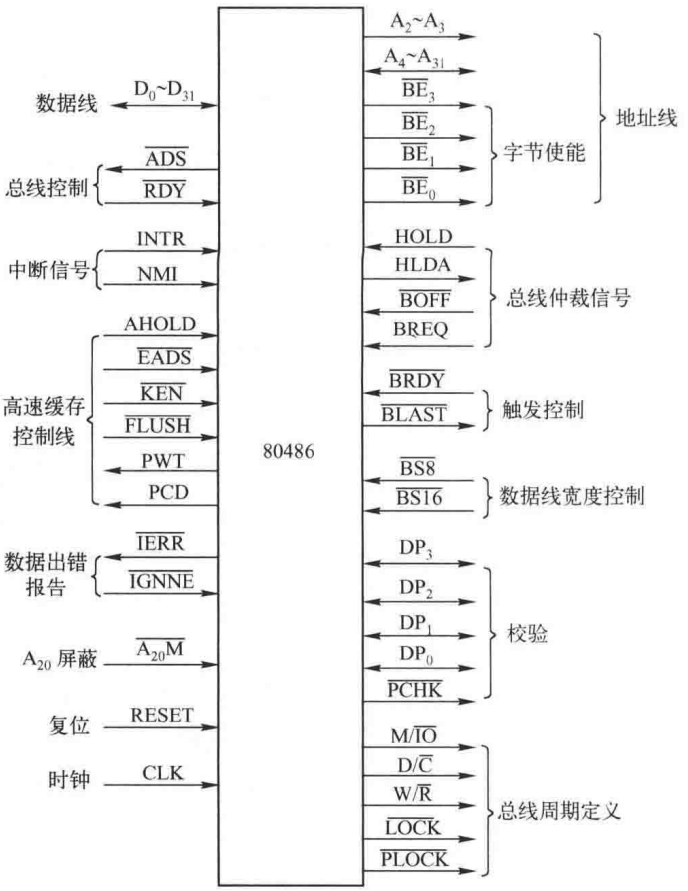


图 12-23　80486 引脚功能

表 12-4　$\overline{BE_0}$～$\overline{BE_3}$ 和相关的数据字节表

字节允许信号	数据字节操作	字节允许信号	数据字节操作
$\overline{BE_0}$	D_0～D_7（第 0 字节，最低有效字节）	$\overline{BE_2}$	D_{16}～D_{23}（第 2 字节）
$\overline{BE_1}$	D_8～D_{15}（第 1 字节）	$\overline{BE_3}$	D_{24}～D_{31}（第 3 字节，最高有效字节）

机和 64 位机，往往是指 CPU 的数据宽度，即 DB 的数目；寻址空间的大小则与 AB 的数目有关。CB 就要复杂多了，处理机越高档，功能越强，其 CB 就越复杂。如 80486 CPU 的 CB，其中一部分 CB 是输入，即外面提供给 CPU 的，另一部分是输出，即 CPU 向外发出的（见图 12-23 中的箭头所示）。

12.3.2　Pentium 系列微处理机

Pentium 微处理机（如图 12-24 所示）是 Intel 公司 1993 年推出的第 5 代微处理机，经过不断的改进和完善，加之采用新技术，到 1999 年推出 Pentium Ⅲ微处理机。

Pentium 微处理机内部有两条流水线，浮点部件拥有专用的加法和乘/除法单元，使得复杂而精确的运算速度大大提高，在指令预处理中增加了分支预测逻辑，使分支指令不停顿流水线的执行。Pentium 内部有两路 Cache，即 8 KB 指令 Cache 和 8 KB 数据 Cache，并可扩展到 12 KB。对存储器的管理也采用分段和分页，但页的大小有两种，即 4 KB 的页和 4 MB 的页。

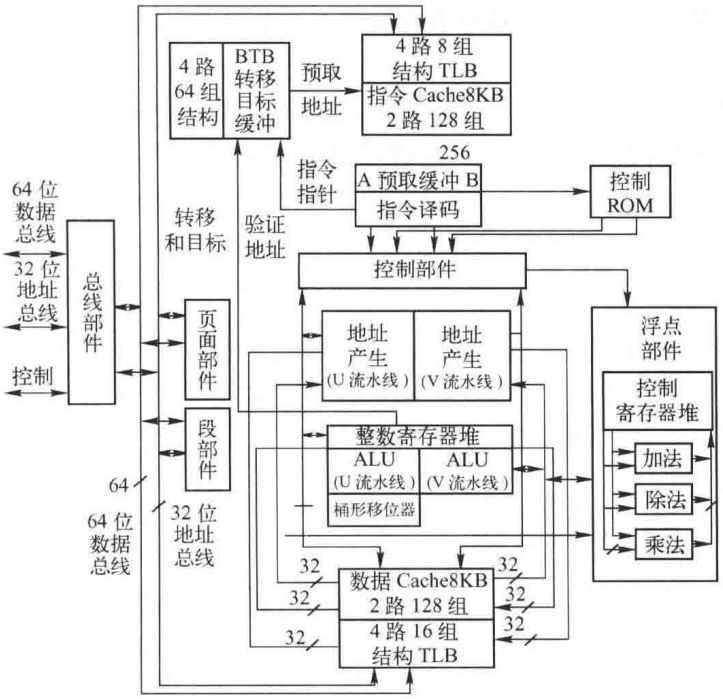


图 12-24　Pentium 微处理机

Pentium 微处理机的技术特点如下：

① 采用超标量双流水线结构，每条流水线包含 5 个步级。所谓超标量，就是微处理器内含多个指令执行部件和多条指令流水线，从而使处理器的运行速度成倍提高。Pentium 有两条流水线，称为 U 流水线和 V 流水线。两条流水线功能类似，各自有独立的算术逻辑部件 ALU、地址生成部件和高速缓存接口。这种结构使 Pentium 可在一个时钟周期执行 2 条简化指令，比 80486 DX 性能提高 1 倍。

② 内部采用彼此独立的 8 KB 代码高速缓存和 8 KB 数据高速缓存，这两个高速缓存可以同时被访问。这种双路高速缓存结构减少了争用高速缓存所造成的冲突，改进了处理器效能。此外，在数据高速缓存中采用一种称为 MESI 的高速缓存一致性协议，为确保多处理器环境下的数据一致性建立了有效前提。

③ Pentium 微处理机内部数据总线为 32 位，但是 CPU 和内存进行数据交换的外部数据总线为 64 位，使总线周期的数据传输量提高了 1 倍，使得与内存的数据交换速率达 528 MBps。它还支持突发式读操作和突发式写操作，使数据传输速度很高。

④ Pentium 微处理机采用 36 位地址总线，扩展了寻址空间；另外，在存储管理中，页面大小可以任选，除了采用与 80386、80486 兼容的 4 KB 页面，还可用其他容量的页面，最高可达 4 MB；通过页面规格的扩大可以减少页面交换的次数，加快了大容量程序的运行速度。

⑤ Pentium 微处理机对 MOV、PUSH、DEC、INC 等常用指令不用微程序而用硬件实现，这样执行速度进一步加快。

⑥ 采用分支预测技术提高流水线效能，在应用软件中包括大量的条件转移指令，这些转移指令由于产生分支，可能使预取和预译码的指令作废，从而造成流水线一时失去应有的效能。Pentium 微处理机内部有两个预取指令缓冲队列，在遇到条件转移前，一个以顺序方式预取指令，一个以转移方式预取指令，后者也称为分支目标缓冲器。这种动态预测程序分支技术

使得不管是否产生转移，所需要的指令都在执行前预先取好，因此有效地避免了条件转移指令可能带来的流水线效能的损失。

⑦ 浮点运算执行过程分为 8 个流水步级。此外，浮点运算部件对 MUL、LOAD 等常用指令不是采取微程序而是由硬件来实现，这使浮点运算速度明显提高。

12.4 RISC 简介

RISC 是 Reduced Instruction Set Computer 的缩写，即“精简指令集计算机”。至今，人们对 RISC 尚没有一个统一的正式定义。严格地说，RISC 既不是一种体系结构，也不是一种设计技术，它只是一种计算机体系结构的设计思想。

12.4.1 RISC 的基本原理

RISC 的指令系统相对简单，它只要求硬件执行很有限且最常用的那部分指令，大部分复杂的操作则使用成熟的编译技术，由简单指令合成。

1．RISC 的产生和发展

在计算机的发展过程中，计算机的体系结构及其性能始终决定着其相关指令系统的演化和发展。早期的计算机结构简单，性能较弱，因而相对的指令数目也就少。但随着 VLSI（超大规模集成电路）技术的迅速发展，硬件成本不断下降，软件成本不断上升，促进人们在指令系统中增加了更多的指令和更复杂的指令，以实现更强大、稳定的功能，从而适应不同应用领域的需求。此外，由于系列机问世之后，为了实现同一系列机型上程序的兼容，高档机除了要继承老机型的指令系统中的全部指令外，还要增加若干新的指令，从而导致同一系列计算机的指令系统越来越复杂，机器结构也越来越复杂。目前，大多数计算机的指令系统都有几百条指令，这也体现了计算机性能越高，其指令系统越复杂的传统设计思想。对于这类计算机统称为复杂指令集计算机（Complex Instruction Set Computer，CISC）。

然而 CISC 这种不断扩充的指令系统，在增加了计算机的研制周期和成本的同时，由于要对复杂指令进行复杂的操作，从而必然的会降低机器的执行速度，这与计算机向高速发展的大趋势产生了矛盾。为此，IBM 公司在 20 世纪 70 年代中期开始研究 RISC。

IBM 的研究人员在研究设计 801 小型机的过程中发现：在典型的程序运算过程中，复杂程序指令很少被调用，计算机执行程序时大部分时间用在执行简单指令上。应用程序 80%是用指令系统中 20%的指令实现的，那些最简单的指令往往是最常用的指令。因此从指令系统中删除不常用的复杂指令，以简单指令的组合来实现，这样就可以加快运行速度，从而诱导出了 RISC 的基本原理。IBM801 小型机于 1979 年研制成功，着重于在硬件中直接执行一组精选的简单指令，以及与“优化编译”相结合。它的研制成功为 RISC 的研究和应用奠定了基础。

与此同时，美国加州大学伯克利分校和斯坦福大学的研究人员也各自进行着对 RISC 处理器的研究。

美国加州大学伯克利分校的研究人员专注利用精简的指令系统所腾出来的更多的芯片空间区域来增强处理机的性能和功能的多样化。他们使用大部分芯片空间作为寄存器，而用寄存

器做临时数据存储的高速存储区，有效地降低了机器在子程序调用上的耗时，并于 1982 年研制成功 RISC-I 芯片，1984 年又完成 RISC-H 32 位微处理器。这些研究成果后来发展成为 SUN 微系统公司著名的 SPARC（Scalable Processor Architecture）结构。

而斯坦福大学 RISC 的研究课题为 MIPS（Microprocessor without Interlocking Pipeline Stages，消除流水线各段互锁的微处理器），结合了 IBM 公司优化编译程序的研究与伯克利的设计思想，利用 RISC 机器指令简单、格式统一，而且指令周期相对固定这些特点，加上优化编译器，有效地提高了 RISC 机器流水线的效率。该大学于 1981 年研究成功了 MIPS 机，被认为是 RISC 设计思想最为典型的产品。

到目前为止，RISC 芯片总体上已经历了三代。

第一代 RISC 处理器，如 IBM 801、伯克利的 SPARC 和斯坦福的 MIPS，一般为 32 位数据通路，支持 Cache，软件支持较少，性能与 CISC 相当。

第二代 RISC 芯片提高了集成度，增加了对多处理机系统的支持，提高了时钟频率，建立了完善的 Cache 分级存储管理体系，软件支持系统也完善起来，已经超过了 CISC 体系结构芯片的性能，如 MIPS 公司的 R3000 处理器。

第三代 RISC 处理器，如 MIPS 公司的 R4000 处理器，采用三种并行技术——超流水线技术、超标量技术和超长指令字。第三代 RISC 处理器的时钟频率和芯片集成度都比上一代有很大提高。另外，64 位或准 64 位结构是第三代 MSC 处理器的特点。由于程序并行化提高，随之产生的各种资源冲突与协调问题在第三代中变得更加明显，因此需要更复杂的控制和调度逻辑。

2．RISC 的基本概念

RISC 的主要思想是简化指令格式和寻址方式，从而提高指令的运行速度。相对于原来具有复杂指令的 CISC 来说，RISC 体系结构大大减少了指令的数量，采用固定长度的指令系统，优化编译程序，寻址方式简单等，从而使 RISC 体系结构具有较高的性能和较小的芯片面积。采用 RISC 设计的处理器更容易实现流水线操作，从而使处理器在每个周期能执行的指令条数接近于 1。

虽然 RISC 指令系统删除了 CISC 指令系统中的复杂指令部分，改用多条简单指令来实现一条复杂指令，但这不意味着 RISC 总的执行程序的速度要慢于 CISC。任何一个程序在计算机上的执行时间可以用公式 $P=I\times CPI\times T$ 来计算。其中，P 是执行这个程序所使用的总的时间；I 是这个程序所需执行的总的指令条数；CPI 是每条指令执行的平均周期数；T 是一个周期的时间长度。RISC 与 CISC 的 I、CPI 和 T 的比较如表 12-5 所示。

表 12-5　RISC 与 CISC 主要参数比较

类　型	指令条数（I）	指令平均周期数（CPI）	周期时间（T）
RISC	1.3～1.4	1.1～1.4	2～10 ns
CISC	1	2～15	5～33 ns

① 对于程序所执行的总的指令条数 I：CISC 中的一条复杂指令完成的功能在 RISC 中可能要用几条指令才能实现，对于同一个源程序，分别编译后生成的动态目标代码，显然 RISC 的要比 CISC 的长。但是，由于 CISC 中复杂指令使用的频度很低，程序中使用的绝大多数指

令都是与 RISC 一样的简单指令，因此，实际上的统计结果表明 RISC 的 *I* 长度只比 CISC 的长 30%～40%。

② 对于指令平均执行周期数 CPI：CISC 一般是用微程序实现的，一条指令往往要用好几个周期才能完成，一些复杂指令所要的周期数就更多。而 RISC 的大多数指令都是单周期执行的，它们的 CPI 应该是 1。但是，由于 RISC 中还有 LOAD 和 STORE 指令这类的少数复杂指令，所以 CPI 要略大于 1。例如，SUN 公司的 SPARC 处理机的 CPI 为 1.3～1.4，SGI 公司的 MIPS 处理机的 CPI 为 1.1～1.2。

③ 对于一个周期的时间长度 *T*：由于 RISC 一般采用硬布线逻辑实现，指令要实现的功能都比较简单，所以 RISC 的 *T* 通常要比 CISC 的小。目前，使用 RISC 处理机的工作主频一般要比 CISC 处理机高。

根据表 12-6 中的数据，结合前面给出的公式，可以计算出 RISC 的速率要比 CISC 快 3 倍左右。其中的关键在于 RISC 的指令平均执行周期数 CPI 减小了，这也正是 RISC 设计思想的精华。

12.4.2 RISC 的特色和难点

1．RISC 的特点

RISC 的特点主要可归纳为如下几点：

① 优化精简了指令系统，选取使用频度最高的一些简单指令，只保留 LOAD 和 STORE 两种访问存储器的复杂指令。

② 指令长度固定，指令格式种类少，指令规整简单，基本寻址方式为 2～3 种，如此简化了逻辑并缩短了译码时间，确保指令在一个周期内执行完。这也有利于流水线技术的实现，因为指令的固定格式能有效的确保指令译码和取操作数在流水线重叠操作时能同时执行，从而提高了运行速度和信息处理能力。

③ 原来 CISC 使用的那些复杂指令的功能，通过使用频度高的简单指令的组合来实现。

④ 大量使用寄存器，数据处理指令只对寄存器进行操作，只有加载、存储指令可以访问存储器，以提高指令的执行效率。

⑤ 采用优化编译程序，力求有效地支持高级语言程序，最大化地保证指令执行过程中的流畅性以及对硬件资源的利用性，从而大大减小了指令的平均运行时间。

⑥ 指令的执行采用流水和延迟转移技术。采用流水技术就是让每一时刻有多条指令重叠执行，使得每条指令平均只需一个周期。而延迟转移技术是针对执行转移指令时，有可能造成预取的下一条指令失效，增加辅助开销这种情况而提出的。其思想是在转移指令后安排一条与它不存在依赖关系的可立即执行的指令，确保流水线的高效运行。

⑦ 使用重叠寄存器窗口技术，将 RISC 中配置的大规模寄存器划分给若干个过程使用，并重叠使用各寄存器，从而实现各寄存器间无延迟的数据交换。

⑧ 采用硬布线控制逻辑直接控制指令的执行，不再使用微程序控制。这有效克服了使用微程序时无法保证平均一个机器周期执行一条指令的问题，同时又能节约芯片面积，从而用以扩充寄存器组或其他功能电路。

实践证明，采用 RISC 结构相较 CISC 结构有如下优势：

① 适合超大规模集成电路实现。由于指令条数相对较少，寻址方式简单，指令格式规整，与 CISC 结构相比，其控制器的译码和执行硬件相对简单，因此 VLSI 片子中实现控制器的部分所占面积可大大减小。

② 直接支持高级语言的能力，简化编译程序的设计。指令的减少，缩小了编译过程中对功能类似的机器指令进行选择的范围，减轻了对寻址方式进行选择、分析和变换的负担，且编译程序易于调整指令顺序，从而提高了程序的运行速度。此外，由于主要操作在寄存器间进行以及重叠寄存器窗口，从而直接支持了子程序和过程调用的高级语言处理。

③ 提高了机器的执行速度、效率和系统的可靠性，降低了设计成本。指令系统的精简可以加快指令的译码，控制器的简化可以缩短指令的执行延时，这些都可以提高程序执行的速度。在功能大抵相等的情况下，RISC 机往往是 CISC 机处理速度的 2～4 倍。另外，采用相对精简的控制器，缩短了设计周期，减小了设计中潜在的风险，这都会降低设计成本，提高系统的可靠性。

2．RISC 的难点

虽然 RISC 架构有上述诸多的优点，但现在还不能认为 RISC 架构就可以完全取代 CISC 架构，因为 RISC 也存在着不少技术难点，从而减缓了 RISC 全面推行的步伐。

由于 RISC 采用较少的指令，从而加重了程序员的负担，增加了机器语言程序的长度，导致占用了较大的存储空间。

由于 RISC 处理器的采用许多并行技术以提高指令执行速率，难免的在系统内部会出现相关问题和资源冲突等，因而其相对来说需要更严格和复杂的调度和控制。

指令系统的兼容问题。而 RISC 机将指令做了简化，数量减少，格式也不同，不像 CISC 机继承了低端机全部的指令，因而 CISC 上的用户代码在 RISC 机上要进行相应的处理或修改。

早期的 RISC 结构对浮点运算的支持不够，对虚拟存储器的支持也不够理想。

编译程序的复杂性。RISC 结构的性能优势依赖于编译程序的有效性，同时因为有大量的寄存器，那么寄存器的分配策略等变得更复杂了，这都增加了编译程序的复杂性，因而 RISC 结构的最主要难点就是必须编写一个很好的编译程序，否则其潜在优势也就难以发挥。

正因为 RISC 存在着这些技术难点，使得现今许多的主流 CPU 设计中并没有完全使用 RISC 的思想，大多采用了 RISC 和 CISC 相结合的方案，取长补短。

12.4.3 RISC 的关键技术

对于 RISC 来说，力求要减少的是程序的执行时间，而程序执行时间的长短主要取决于 3 个因数：程序指令数目、指令周期时间和每条指令所需的周期数。RISC 要达到其很高的性能就必须有相应的技术支持。目前，RISC 处理机中主要是对以下关键技术进行讨论。

1．延时转移技术

在 RISC 处理机中，指令一般采用流水线方式工作。取指令和执行指令并行进行。如果取指令和执行指令各需要一个周期，那么在正常情况下，每个周期就能执行完一条指令。但是当遇到转移指令时，流水线有可能会断流。而在采用指令延迟转移技术时，编译器能自动调整指令序列，一般不需要外部干预。

2．指令流调整技术

为了使 RISC 处理机中的指令流水线高效率地工作，尽量不断流，优化编译器必须分析程序的数据流和控制流，当发现指令流有断流可能时要调整指令序列，对有些可以通过变量重新命名来消除的数据相关，要尽量消除。这样可以提高流水线的执行效率，缩短程序的执行时间。

3．硬件为主、固件为辅

RISC 采用以硬件为主、固件为辅的技术。控制器采用硬连线逻辑部件，而不采用微程序控制；处理器从缓存或主存获取指令后的指令解码也通常是硬连线实现而不是微解码。

硬连线是用逻辑门来生成控制信号，而微程序控制器是把控制信号存储在查找表 ROM 中通过从查找表 ROM 中读出数据，然后输出控制信号值。对于 RISC，硬连线控制单元要比微序列控制单元要快，因为使用硬连线控制省去了将机器指令转化为原始代码这一中间步骤，也就减少了执行一条指令所需的机器周期个数。

微程序的主要好处是便于实现复杂指令，便于修改指令系统，增加机器的灵活性和适应性，但存在执行速度低的缺点。而 RISC 要求指令能在单周期内执行完成，所以采用微程序技术是不可能做到的。因此，RISC 必须采用硬连逻辑来实现指令系统。但那些使用频率低但必需的复杂指令还是可用固件（微程序技术）来实现的。因此，目前商用的 RISC 处理机在实现指令系统时，一般都采用以硬件为主、固件为辅的方法。

4．重叠寄存器窗口

重叠寄存窗口的基本思想是：在机器中配置一个很大的寄存器群，将其划分给若干个过程使用，每个过程所用的一组寄存器又可分为三部分：第一部分用来与高一级（本过程的主调过程）交换参数；第二部分只供本过程使用；第三部分用来与低一级过程（本过程的受调过程）交换参数。每个过程的第三部分寄存器与低一级过程的第一部分寄存器合用相同的物理寄存器，以达到寄存器的重叠使用。当一个过程去使用其参数寄存器时，它实际上已完成了与其他过程的参数交换，这种交换过程没有任何延迟。

使用重叠寄存器窗口是 RISC 结构的一个重要特性。重叠窗口可以有效地节省过程间通信所需要的时间，这会使协作过程之间的上下文切换变得更快。

5．流水线技术

RISC 系统广泛使用流水线技术，提高 CPU 运行效率。

（1）流水线技术的基本原理及其特点

对于流水线技术，每条指令在执行过程中被分成若干执行阶段。只有当每个指令执行阶段都完成之后，一条指令才算执行完毕。在每个指令执行阶段中，当一条指令在该阶段中完成执行之后，下一条指令将立即进入该执行阶段来执行操作。当流水线处于饱和状态时，在 CPU 中将同时有流水线级数的指令在同时执行。

RISC 技术的关键是减少 CPI 即每条指令的周期数，采用流水技术就是让每一时刻有多条指令重叠执行。虽然一条指令的执行仍需几个周期的时间，但从平均效应来看，每条指令的周期数大大减少，能够达到每条指令只需一个周期。流水线执行示意图如图 12-25 所示。

其中，每个执行阶段的名称和功能如下：IF，取指令阶段；ID，指令译码阶段；EXE，指令执行阶段；MEM，访问存储器阶段；WB，数据回写阶段。

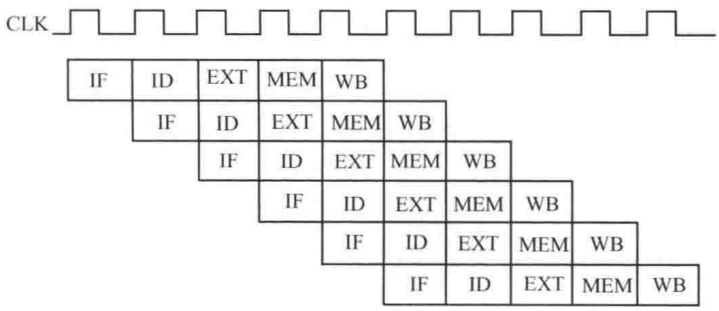


图 12-25　流水线执行示意

可以看出，流水线具有如下特点：

① 流水过程由多个相联系的子过程组成，每个子过程称为流水线的“级”或“段”。流水线的段数也称为流水线的“深度”或“流水深度”。

② 每个子过程由专用的功能段实现。

③ 每个功能段所需的时间应尽量相等，否则时间长的功能段将成为流水线的瓶颈，会造成流水线的“堵塞”和“断流”。

④ 流水线需要“通过时间”（第一个任务流出结果所需时间），此后流水过程进入稳定工作状态，每一个时钟周期都有结果。

⑤ 流水线技术适合于大量重复的时序过程，只有输入端能连续的提供任务，流水线的效率才能充分发挥。

（2）流水线的相关问题

流水线技术虽然能够提高指令的执行速度，但由于流水线的各个段之间存在很强的依赖关系。如果处理不当，指令的运行达不到预期的结果。因此对于一个基于流水线的 RISC 处理器，必须充分考虑流水线的相关问题，即结构相关、数据相关、控制相关问题。

① 结构相关：如果某些指令组合在流水线中重叠执行时，产生资源冲突，则称该流水线有结构相关。许多的流水线机器都是将数据和指令保存在同一个存储器中。如果在某个时钟周期内，流水线既要完成某条指令对数据的存储器访问操作，又要完成取指令的操作，那么就会发生存储器的访问冲突问题，产生结构相关。结构相关通常采用资源重复的方法来解决。

② 数据相关：当指令在流水线中重叠执行时，流水线有可能改变指令读/写操作数的顺序，使得读/写操作顺序不同于它们非流水线实现时的顺序，这将导致数据相关。数据相关的问题可以采用定向技术（也称为旁路或短路）来解决。主要思想是：在某条指令产生结果之前，其他指令并不真正需要计算结果，将该计算结果从其他地方（寄存器文件 EX/MEM）直接送到其他指令需要它的地方（ALU 的输入寄存器），那么就可以避免数据相关带来的暂停。

③ 控制相关：当流水线遇到分支指令和其他会改变 PC 值的指令时就会发生控制相关。处理分支指令最简单的方法是，一旦在流水线检测到某条指令是分支指令，就暂停执行该分支指令后的所有指令，直到分支指令到达流水线的 MEM 段，确定新的 PC 值为止。但是这样做就会给流水线带来多个时钟周期的暂停。

减少流水线分支指令的暂停周期数有如下两种途径：一是流水线中尽早判断出分支转移是否成功，二是尽早计算出分支转移成功时的 PC 值。通常，降低流水线分支指令损失的方法有：冻结（Freeze）或排空（Flush）流水线的方法，“预测分支失败”方法，“预测分支成功”方法，“延迟分支”方法。限于篇幅，这里就不再详细介绍了。

习 题 12

1. 80386 内部由哪几个功能部件组成？它们的主要功能是什么？
2. 8086 与 80386 比较有哪些主要区别？
3. 80386 中包含哪些寄存器？各有什么主要用途？
4. 80386 访问存储器有哪两种方式？各提供多大的存储空间？
5. 什么是线性地址？
6. 简述 80386 对段、页式存储器结构的寻址过程。
7. 什么是 RISC？什么是 CISC？
8. RISC 系统有什么特点？

参考文献

[1] 彭虎，周佩玲，傅忠谦．微机原理与接口技术（第4版）．北京：电子工业出版社，2016.

[2] 周佩玲，吴耿峰，万炳奎编．十六位微型计算机原理·接口及其应用．合肥：中国科学技术大学出版社，1995.

[3] Barry B. BREY．80x86、奔腾机汇编语言程序设计．金惠华，曹庆华，李雅倩译．北京：电子工业出版社，1998.

[4] 艾德才等．Pentium/80486 实用汇编语言程序设计．北京：清华大学出版社，1997.

[5] 艾德才．Pentium 系列微型计算机原理与接口技术．北京：高等教育出版社，2000.

[6] 马维华，奚抗生，易仲芳，毛建国．从 8086 到 Pentium Ⅲ微型计算机及接口技术．北京：科学出版社，2000.

[7] 贾志平．计算机硬件技术教程——微机原理与接口技术．北京：中国水利水电出版社，1999.

[8] 吴秀清，周荷琴．微型计算机原理与接口技术（第二版）．合肥：中国科学技术大学出版社，2002.

[9] 许用和．EZ-USB FX 系列单片机 USB 外围设备设计与应用．北京：北京航空航天大学出版社，2002.

[10] 周明德．微型计算机系统原理及应用（第四版）习题集与实验指导书．北京：清华大学出版社，2002.

[11] 戴梅萼．微型计算机技术及应用（第三版）．北京：清华大学出版社，2003.

[12] 杨振江，孙占彪，王曙梅，步线涛．智能仪器与数据采集系统中的新器件及应用．西安：西安电子科技大学出版社，2001.

[13] RS-232C 详解．http://www.gjwtech.com/.

[14] Universal Serial Bus Specification Revision 2.0．http://www.usb.org/.

[15] USB 系统结构与应用设计．http://www.fuja.com.cn/.

[16] USB 细说从头．http://www.fuja.com.cn/.

[17] 如何编写 Windows CE.NET 的 USB 驱动程序．http://www.ccw.com.cn/.

[18] 高速数据传输 USB 2.0 技术全面接触．http://www.pconline.com.cn/.

19] usb-features．http://www.usb.org/.

[20] Universal Serial Bus-FAQs．http://www.usb.org/.

[21] John L. Hennessy, David A. Patterson．计算机体系结构．北京：电子工业出版社，2004.

[22] 孙德文．微型计算机技术．北京：高等教育出版社，2001．
[23] 钱晓捷，陈涛．16/32 位微机原理、汇编语言及接口技术．北京：机械工业出版社，2001．
[24] Windows 环境下 32 位汇编语言程序设计．http://book.csdn.net/bookfiles/.
[25] 中断和异常的转移方法．http://www.vipcn.com/InfoView/Article_213456.html/.
[26] 80x86 保护模式之中断和异常．http://www.cnsharenet.com/DOS/.
[27] 李继灿，李华贵，沈疆海，郭麦成．新编 16-32 微型计算机原理及应用．北京：清华大学出版社，2001．

微机原理与接口技术（第5版）

本书为普通高等教育“十二五”本科国家级规划教材。

本书介绍信息在计算机中的存储形式、数制及相互转换、二进制数的算术和逻辑运算等基础知识；软件部分讲述8086指令系统、部分伪指令和DOS功能调用及汇编语言程序设计和调试的全过程；硬件部分介绍8086 CPU的内部特点、寄存器及相关概念、存储器的分类及层次结构、物理地址形成、译码电路等；讨论诸多I/O接口芯片的结构、编程及应用，在串行通信中还介绍了USB总线；讨论并举例说明了A/D、D/A芯片、微机接口及应用，本书还对80286、80386 CPU主要内容及其体系做了简要介绍。全书共12章，每章附有习题，提供配套电子课件。

本书适合作为高等院校信息类理工科学生相关课程的教材，也可以作为相关技术人员或爱好者的参考书。

提升学生“知识—能力—素质”	体现“基础—技术—应用”内容
把握教学“难度—深度—强度”	提供“教材—教辅—课件”支持

ISBN 978-7-121-41607-1

9 787121 416071 >

定价：59.80 元

责任编辑：章海涛
封面设计：张　昱

[General Information]
书名=微机原理与接口技术 第5版
页数=324
SS号=14978787