普通高等教育“十一五”国家级规划教材
PUTONG GAODENG JIAOYU SHIYIWU GUOJIAJI GUIHUA JIAOCAI

16/32WEI WEIJI YUANLI JI JIEKOU JISHU

16/32位微机原理及接口技术

朱有产　刘淑平　王桂兰　编著

中国电力出版社
http://jc.cepp.com.cn

普通高等教育“十一五”国家级规划教材
PUTONG GAODENG JIAOYU SHIYIWU GUOJIAJI GUIHUA JIAOCAI

自动控制理论（第四版）	孙扬声
自动控制理论（第三版）	文　锋
自动控制原理	于希宁
自动控制原理-理论篇	杨　平
16/32位微机原理及接口技术	朱有产
可编程序控制器原理及应用	郁汉琪
接口与通信	王保义
电路理论基础（第三版）	梁贵书
电磁测量技术	李宝树
电磁场导论	孟昭敦
电机学（第三版）	牛维扬
电机学	陈世元
电机理论基础	凌跃胜
微特电机及系统	程　明
电气工程概论	熊信银
电气工程概论	李建华
电气工程概论	肖登明
电气工程CAD技术	冯林桥
电力系统概论	杨淑英
电气信息技术基础	夏安邦
电力系统基础（第二版）	杨以涵
电力工程	尹克宁
发电厂电气部分	姚春球
发电厂及变电站的二次回路	何永华
电力系统继电保护原理（第四版）	贺家李
微型机继电保护基础（第三版）	杨奇逊
电力网继电保护原理	孙国凯
现代高压电网继电保护原理	李晓明
电力系统自动装置原理（第四版）	杨冠城
电力系统自动化（第五版）	李先彬
电力系统自动化（第二版）	王　葵
电力系统故障分析（第三版）	刘万顺
电力系统分析（第三版）	于永源
电力系统规划	程浩忠
高电压技术（第二版）	赵智大
高电压技术（第二版）	张一尘
电能计量技术	王月志
电力建设工程造价控制与管理	黄宵宁
电能质量分析与控制	肖湘宁
智能控制系统及应用	张国忠
MATLAB在电气工程中的应用	李维波
交直流调速系统与MATLAB仿真	周渊深
过程参数检测及仪表	常太华
电力技术经济原理	肖先勇
电力企业管理	卢建昌
电力市场运营系统（第二版）	刘继春
电气工程及其自动化专业英语	苏小林
电力新技术概论	王仁祥

ISBN 978-7-5083-9171-7
9 787508 391717 >
定价：39.20 元

普通高等教育“十一五”国家级规划教材
PUTONG GAODENG JIAOYU SHIYIWU GUOJIAJI GUIHUA JIAOCAI

16/32WEI WEIJI YUANLI JI JIEKOU JISHU

16/32位微机原理及接口技术

编著　朱有产　刘淑平　王桂兰
主审　许　会　王振旗

中国电力出版社
http://jc.cepp.com.cn

内 容 提 要

本书为普通高等教育“十一五”国家级规划教材。

本书将微机原理、汇编语言程序设计和微机接口技术的内容进行整合，以当前国内外广泛使用的 Intel 80X86 系列微处理器为主线，系统地介绍了微型计算机系统及工作原理、16/32 位微处理器的内部结构、微机总线、指令系统、汇编语言程序设计、存储器管理、中断技术、输入/输出技术、常用 I/O 接口芯片、A/D 和 D/A 接口技术。本书内容简明扼要、由浅入深、循序渐进、结构层次清晰，着重阐明基本概念、基本思路、基本方法及工程应用，并且配有大量的图示、例题和详细的程序注释，便于教学与自学。

本书可作为普通高等院校电气信息类等非计算机专业的本科教材，也可作为成人高等教育、高职高专和各类培训班的教材，还可以作为研究生和从事微机测控及接口技术应用的工程技术人员的参考书。

图书在版编目（CIP）数据

16/32 位微机原理及接口技术/朱有产，刘淑平，王桂兰编著．—北京：中国电力出版社，2009

普通高等教育“十一五”国家级规划教材

ISBN 978-7-5083-9171-7

Ⅰ.1… Ⅱ.①朱…②刘…③王… 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.TP36

中国版本图书馆 CIP 数据核字（2009）第 124551 号

中国电力出版社出版、发行

（北京三里河路 6 号 100044 http://jc.cepp.com.cn）

北京丰源印刷厂印刷

各地新华书店经售

*

2009 年 8 月第一版 2009 年 8 月北京第一次印刷

787 毫米×1092 毫米 16 开本 24.5 印张 599 千字

定价 **39.20** 元

敬 告 读 者

本书封面贴有防伪标签，加热后中心图案消失

本书如有印装质量问题，我社发行部负责退换

版 权 专 有 翻 印 必 究

前　言

随着微型计算机的广泛应用，掌握微型计算机系统知识及接口技术尤为重要。在微机接口技术及原理的教学中，学生不仅应该掌握原理，更要学会应用，做到理论与实际相结合。同时，接口技术的学习不仅仅是硬件上的设计与分析，还应该包括接口软件的编写。目前，各种微处理器、芯片的发展都是建立在8086/8088和常用的各种基本接口芯片基础上的，只是在速度、功能、专用性上有所增强。所以，本书以16位微机为主、32位微机为辅组织内容，较好兼顾基础性、先进性和前沿性。把微机原理、汇编语言和接口技术结合起来，使读者自行可以设计微型计算机系统。在内容的组织上，本着由浅入深、循序渐进的原则，注重基本知识和基本概念的介绍，结合实际应用重点举例介绍，使学生有的放矢，利于掌握所学内容。

16/32位微机原理及接口技术是非计算机电类及相关专业的一门重要技术基础课。本书不仅适用于偏硬专业的学生，也适合于偏软专业的学生，试图达到硬件、软件一样强。本书主要有下列特点：

（1）微型计算机的原理讲得透彻，结合模型机讲清计算机的工作原理。

（2）对16/32位CPU进行有效的内容组合，在掌握基本原理的基础上，介绍高端CPU的不同之处。

（3）内容组织由浅入深、由理论到应用，介绍构建计算机系统组成部分及应用领域。

（4）外围接口芯片着重举例说明实际应用编程过程。

本书共分为10章：第1章是对微型计算机系统基本概念、计算机中数据的表示方法、计算机工作原理的介绍；第2章主要介绍8086微处理器的结构、工作模式、段结构及总线技术，并分析了80486和Pentium微处理器的内部结构、工作方式及工作原理的发展变化；第3章主要讲述寻址方式、指令系统、汇编语言和汇编程序设计；第4章讲述半导体存储器、主存储器的设计及常用外存储器；第5章讲述输入输出技术、简单接口电路、DMA控制器8237；第6章讲述中断系统概念、8086/8088中断系统、中断控制器8259A和键盘中断；第7章讲述可编程计数器/定时器8253及8254；第8章讲述并行I/O接口概念、8255A及应用；第9章讲述串行通信概念、RS-232C标准、8251A及应用；第10章讲述D/A、A/D转换器的工作原理及应用。本书参考学时为64～70学时。

本书由朱有产、刘淑平任主编，其中第1、3、5、7、8章由朱有产编写，第2、4、6、10章由刘淑平编写，第9章由王桂兰编写，附录由朱有产整理。华北电力大学的郑颖、郭学英、申鹏飞、许炜强和魏秋娟同学参与了部分图表的绘制。全书由沈阳工业大学许会、华北电力大学王振旗主审，他们认真审阅了稿件并提出了大量宝贵意见，在此编者一并表示最诚挚的感谢。

由于笔者水平有限，书中难免有错误和不妥之处，敬请广大同仁和读者提出宝贵意见。

编　者

2009年6月

目　录

前言
第1章　微型计算机基础知识概述 …… 1
1.1　微型计算机系统构成 …… 1
1.2　微型计算机的基本知识 …… 6
1.3　常用术语解析 …… 16
1.4　计算机基础 …… 18
习题 …… 25
第2章　微处理器 …… 26
2.1　微处理器概述 …… 26
2.2　8086/8088 微处理器 …… 30
2.3　80486 微处理器 …… 51
2.4　Pentium 微处理器 …… 65
2.5　总线 …… 74
2.6　外部通信总线 …… 85
习题 …… 91
第3章　指令系统及汇编语言程序设计 …… 92
3.1　概述 …… 92
3.2　寻址方式 …… 93
3.3　指令系统 …… 100
3.4　汇编语言的编程格式 …… 137
3.5　汇编语言程序设计基础 …… 159
习题 …… 185
第4章　存储器系统 …… 192
4.1　存储器概述 …… 192
4.2　随机读写存储器 RAM …… 198
4.3　只读存储器 ROM …… 203
4.4　存储器系统设计 …… 208
4.5　高速缓冲存储器 Cache …… 221
4.6　虚拟存储器及其管理技术 …… 225
4.7　外存储器 …… 230
习题 …… 232
第5章　输入输出技术 …… 234
5.1　I/O 接口 …… 234
5.2　简单接口电路 …… 236

5.3 微机与 I/O 设备的信息交换方式 …… 243
5.4 DMA 控制器 8237A …… 248
习题 …… 260
第 6 章 中断技术 …… 261
6.1 中断系统的基本概念 …… 261
6.2 8086/ 8088 中断系统 …… 266
6.3 可编程中断控制器 8259A …… 270
6.4 键盘中断 …… 289
习题 …… 292
第 7 章 可编程计数器/定时器 Intel8253 …… 294
7.1 概述 …… 294
7.2 8253 的控制字/锁存字 …… 296
7.3 8253 的工作方式 …… 298
7.4 8253 的应用编程 …… 302
7.5 8254 简介 …… 306
习题 …… 308
第 8 章 微机系统的并行 I/O 接口 …… 309
8.1 并行 I/O 接口概述 …… 309
8.2 并行 I/O 接口芯片 Intel 8255A …… 309
8.3 Intel 8255A 应用举例 …… 319
8.4 82C55A 应用于 32 位 CPU 的 I/O 接口 …… 327
习题 …… 329
第 9 章 微机系统串行通信及接口 …… 331
9.1 串行通信 …… 331
9.2 可编程串行接口 8251A …… 339
习题 …… 347
第 10 章 数模 D/A 和模数 A/D 转换接口 …… 348
10.1 模拟量输入输出通道 …… 348
10.2 D/A 转换器 …… 349
10.3 A/D 转换器 …… 360
习题 …… 367
附录 …… 368
附录 1 美国信息交换标准代码（ASCII） …… 368
附录 2 8086/8088 指令执行时间及简表 …… 368
附录 3 DOS 及 BIOS 功能调用 …… 373
附录 4 80X86 的浮点指令和操作系统型指令 …… 380
参考文献 …… 383

第 1 章　微型计算机基础知识概述

本章从介绍计算机的软硬件基本组成、数制、编码及数据类型开始，重点介绍计算机的基本结构及其整机工作原理。

1.1　微型计算机系统构成

完整的微型计算机系统以微型计算机（简称微机）为中心，配以相应的外围设备以及控制微型计算机工作的软件构成。微型计算机以微处理器为核心，配上由大规模集成电路制作的只读存储器（ROM）、读写存储器（RAM）、输入/输出接口电路及系统总线等组成。微处理器由算术逻辑部件（ALU）和控制部件两大主要部分组成，实现运算功能和控制功能。

微型计算机随着互联网技术的普及与发展，从功能上分为网络工作站（客户端 Client）和网络服务器（Server）两类。网络工作站主要是指家庭用或办公用的个人计算机（PC），从外观上看一般有台式和立式两种，如图 1.1（a）、（b）所示。它主要由主机箱、显示器、键盘、鼠标等组成，主机箱中主要有主板、中央处理单元（CPU，处理器、微处理器）、内存、硬盘、软驱（盘）、显示卡、声卡、网卡、光驱、电源等。主机箱打开以后如图 1.1（c）所示。

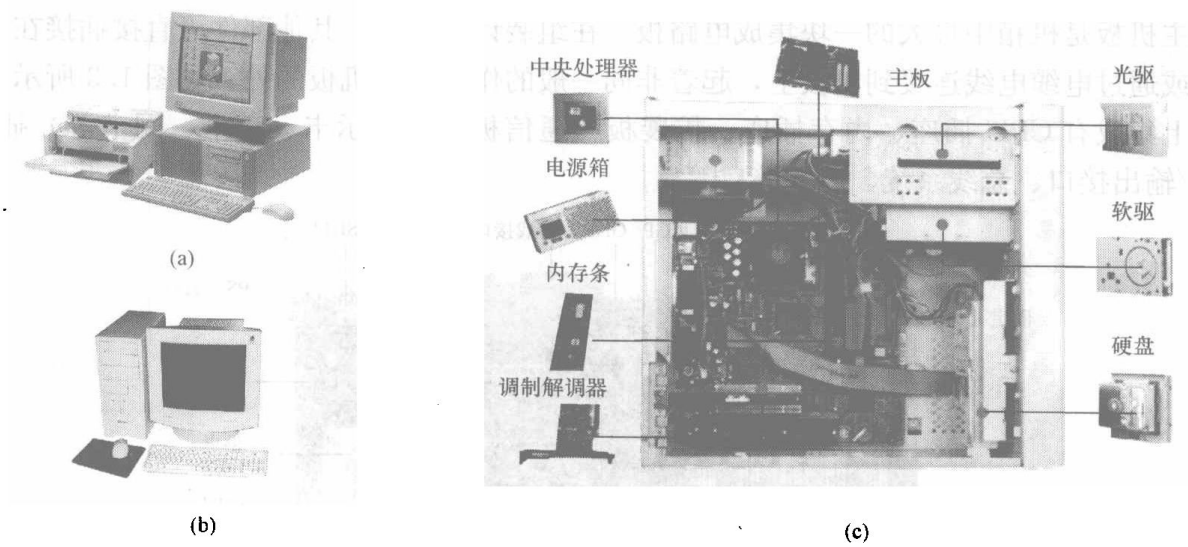


图 1.1　微型计算机系统构成

（a）台式微型计算机的组成；（b）立式微型计算机的组成；（c）主机箱内各组成部分

1.1.1　中央处理单元（CPU）

微型计算机的核心是中央处理单元 CPU(Central Processor Unit)，主要由运算器和控制器构成，完成指令的解释和运算功能。日常生活或工作中所做的文字编辑、报表制作、画图、声音的处理、看影碟、玩游戏等都是将每一个动作分解为基本指令，由 CPU 来执行。

目前生产微型计算机 CPU 的主要厂商是美国的 Intel 公司和 AMD 公司，占市场的主要

份额，另外还有 Cyrix、Motorola、NEC 等公司。Intel 公司一直代表着微型计算机 CPU 的最先进技术，推出了一系列产品，从早期的 Intel 8086/8088、80286、80386、80486 到 Pentium、Pentium Pro、Pentium Ⅱ、Pentium Ⅲ、Pentium Ⅳ等。

封装后 CPU 的形状如图 1.2 所示。在它们的内部集成了成千上万的逻辑门阵电路，这些电路组成了用于运算的算术逻辑单元、译码器、数据缓冲器、触发器、寄存器、计算器等基本运算单元。由计算机的各种高级语言书写的程序最终都翻译成机器认识的机器语言程序，由上述基本运算单元来处理。

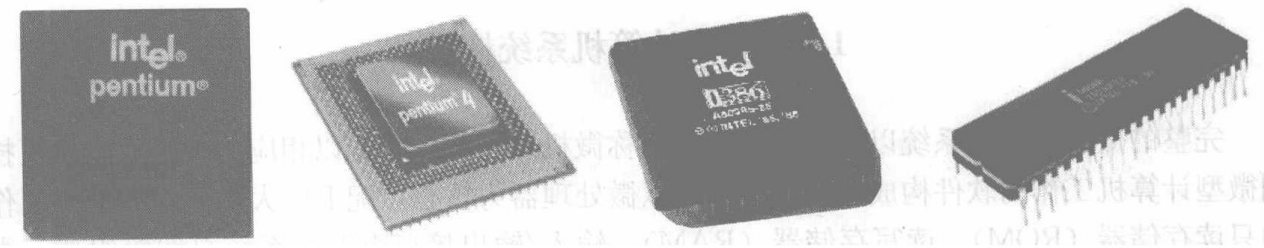


图 1.2　封装后 CPU 的形状

通常我们评价一个 CPU 的性能看下列四个指标：一看主频。主频越高，其运算速度越快。二看 CPU 的缓存大小和速率。缓存大且读写速度快，则 CPU 的运算速度会提高，反之会变低。三看字长。字长是 CPU 一次能并行处理的二进制位数。位数越长，单位时间内处理的二进制信息越多，从而提高 CPU 的处理能力。四看集成度。集成度越高，CPU 的处理能力越强。

1.1.2　系统主机板

主机板是机箱中最大的一块集成电路板。在组装计算机时，其他部件可直接插接在主板上，或通过电缆电线连接到主板上，起着非同一般的作用，主机板的外观如图 1.3 所示。主机板上集成有 CPU 插座、内存插座、扩展板（通信板卡、显示卡、声卡、网卡等）插座、输入/输出接口、总线系统、电源接口等。

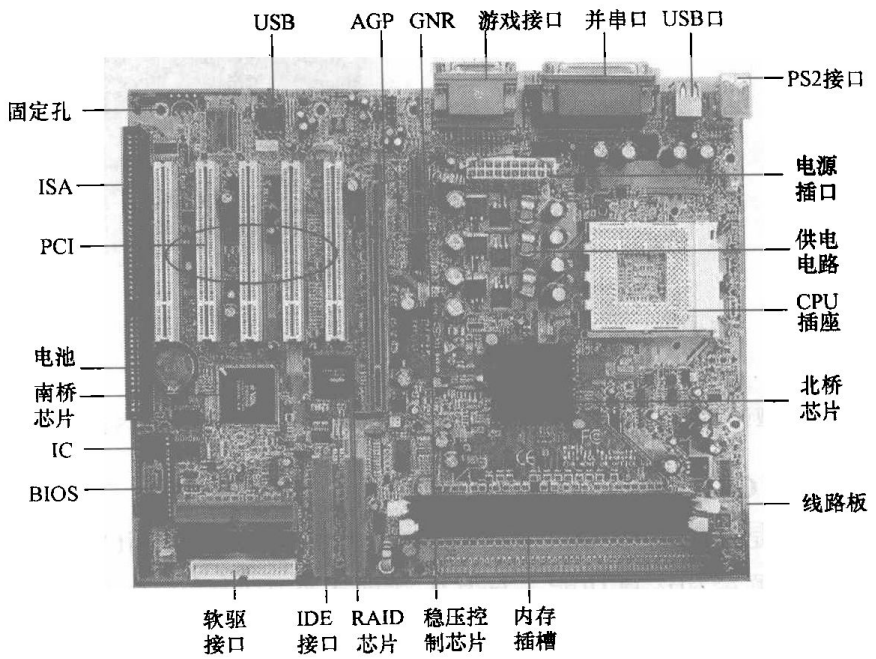


图 1.3　微机系统主机板构成

最早推出的微型计算机（IBM-PC、IBM-PC/XT）主机板采用近百个中小规模集成电路构成，其中有 Intel 8255 并行接口芯片、8253 定时计数芯片、8259 中断控制芯片、8237DMA 控制器芯片、8250 串行接口芯片等外设接口芯片。随着集成电路技术的发展和微型计算机的迅速普及，一些专用芯片集成了上述中小规模集成电路和接口芯片，形成了 PC 机的芯片组。

从体系结构角度来看，传统芯片组采用分控体系结构，即 PCI 总线。随后 Intel 在其芯片组采用加速集线器结构取代原有的 PCI 总线，并用专用总线连接各个设备和 CPU 来达到高速处理的目的。目前芯片组采用新型内存控制集线器和 I/O 控制集线器，其带宽比传统的 PCI 总线速度增加一倍。另外，还有一些新型的控制器如固件集线器、图形存储控制集线器、音频解码控制器等。现在主机板中不再配 ISA 总线插槽，而是将 PCI 插槽作为基本插槽。

现在微型计算机除了通过并口、串口或主机板上的总线扩展槽方式与外部设备连接外，有一种全新的串行接口方式——USB 接口。主流微型机主机板支持 2～4 个 USB 接口，并可以连接如键盘、鼠标、MODEM、摄像头、数码相机、扫描仪、打印机等多种设备。

USB 的英文原名是 Universal Serial Bus，即通用串行接口，是由 Intel、Microsoft、IBM、DEC、Compaq、Northen Telecom 等公司共同提出的一种全新的串行总线式接口，可以完成输入/输出功能，详细介绍见 2.6.3。

1.1.3　内存储器

存储器分为内存储器和外存储器。内存储器简称为内存，由半导体存储器构成；外存储器则包括硬盘、软盘、光盘、磁带等。

内存储器以内存条的形式插在主机板的内存扩展槽中。常用的内存条有 32、64、128、256MB 等，可由用户自行决定插入内存条的数量和容量。

一台微机当 CPU 确定后，决定机器速度的最主要因素就是内存了，即内存的容量和内存的工作速度。当应用程序运行时，程序和数据先要装入内存；在运行时 CPU 要频繁地访问内存，从中取出指令和数据以便执行任务。内存通常由大规模的 MOS 电路构成，其工作速度比硬盘高，但仍比 CPU 慢一个数量级。为此，现在微机常采用更高速度的存储器（当然容量更小）作为 CPU 与内存之间的高速缓存（Cache）。而从 486CPU 以后，把 Cache 集成到 CPU 内部，以形成多级 Cache 结构。

现代微机采用高速缓存（若干级）、内存和后备存储器（如硬盘）这样的多级存储体系，以追求存储容量大、存取速度快和性能价格比高。

1.1.4　外存储器

外存储器常用的有硬盘、软盘和光盘存储器。

硬盘是微机主要的外存储器设备，由驱动器、控制器、盘片三部分组成。上述部件密封组合为硬盘。硬盘的主轴马达带动盘片高速旋转，产生浮力使磁头漂浮在盘片上方。当存取资料的扇区到了磁头下方时，才能读取所需的内容。所以，转速在很大程度上决定了硬盘的读/写速度。一个硬盘通常由多个盘片组成。决定硬盘性能的主要因素有单盘片容量、转速、缓存大小、平均寻道时间、接口类型、平均数据传输率、突发性数据传输率、噪声、抗震性能等。

在微机系统中使用的硬盘均是 IDE（集成设备电路）或 SCSI（小型计算机系统接口）

标准的硬盘，SCSI接口比IDE接口复杂、传输速度快、价格也贵一些，一般用于微型计算机服务器中。一台计算机安装多块硬盘时可利用跳线将其设置为主盘或辅盘，主盘是用于启动系统的硬盘。硬盘出厂时默认设置为主盘，即跳线连接在“Master”一侧。微机中已有主盘，现要连接一个辅盘，则要将跳线连接在“Slave”或“Cable Selection”一侧。标准主机板上设计了两个IDE硬盘接口，一个IDE接口可以用一根数据电缆接一块主盘和一块辅盘，所以一台微机最多可以接四块硬盘。

软盘存储器由软盘驱动器、软盘控制器以及软盘盘片三部分组成。因其存储的盘片类似于塑料薄膜唱片的柔性材料，故俗称为软盘。其存储数据的原理同硬盘。软盘片有8英寸、2.25英寸和3.5英寸，主流是3.5英寸软驱和软盘，但现在已基本退出市场。

光盘存储器是光盘和光盘驱动器的统称，是利用光字方式进行读写信息的介质，与磁盘相比，光盘记录密度更高，更容易保存。光盘信息存储格式类似磁盘，是分轨道（这里不能说磁道）、扇区来记录信息的字节，但它的轨道不是像磁盘那样的同心圆，而是螺旋形由中心向外散布。

光盘分为下列三种：

（1）只读型光盘（CD-ROM）：只能读出其中信息，而不能更改。存储容量650～740MB。

（2）一次性写入光盘：用户购买此类光盘后，配以可读写光驱，用户可向光盘写入想要的信息。只能写入一次，但写入后可以读出。

（3）可擦写型光盘：可对光盘反复进行读写操作，但须配可擦写光驱。存储容量可达10GB以上。

随着CD-ROM技术的日益成熟，现在微机中已可以配置数字化视频光盘（DVD-ROM）驱动器，通过个人电脑刻录时其存储容量达4.7GB。

在微机中CD-ROM光驱像硬盘一样，有IDE和SCSI接口且认为是像硬盘一样的设备。

1.1.5 显示卡与显示器

显示卡是插在主机板上总线扩展槽上的一块电路卡，它是主机与显示器之间的接口，它接收CPU和内存发送来的信息，并且将这些信息处理转换后向显示器发送视频信号和同步信号。现在显卡本身已成为一个功能完善的小计算机了，高档的显示卡有自己的CPU、存储器等。

目前，显示卡大约有三种发展方向：一是注重游戏市场，二是注重高端三维设计和CAD辅助设计，三是注重上述两者的结合。

评价显示卡的主要指标是显示卡的工作频率、显示缓存的大小、像素填充率等。

显示卡的输出电缆接显示器，显示器是当前微机中最主要的输出设备，目前主流是液晶显示器。显示器性能的好坏要看下面一些指标：

（1）显像管外形：会影响图像质量、色彩、亮度、反射和对比度等。它主要有球面屏幕、平面直角屏幕、柱面屏幕、LCD液晶显示器等。

（2）扫描方式：分为“逐行扫描”和“隔行扫描”两种。隔行扫描是指每隔一行显示，如先从第一行开始，一行、三行、五行、……到底后再返回显示刚才未显示的行；而逐行扫描是指顺序显示每一行。逐行扫描比隔行扫描拥有更稳定的显示效果。

（3）点距：是指屏幕上两个相邻荧光点的距离，点距越小，显示器显示图形越清晰。

（4）刷新频率：分为垂直刷新率和水平刷新率，垂直刷新率表示屏幕的图像每秒钟重绘

多少次，也就是指每秒钟屏幕刷新的次数，以 Hz（赫兹）为单位。水平刷新率又称行频，它表示显示器从左到右绘制一条水平线所用的时间，以 kHz 为单位。当刷新率低于 60Hz 的时候，屏幕会有明显的抖动现象，而一般要到 72Hz 以上才能较好地保护用户的眼睛。

（5）频率带宽：指每秒钟电子枪扫描过的总像素数，等于“水平分辨率×垂直分辨率×场频（画面刷新次数）”。

（6）分辨率：是以乘法形式表现的，比如 800×600，其中“800”表示屏幕水平方向显示的点数，“600”表示垂直方向显示的点数。所谓的分辨率就是指画面的解析度，由多个像素点构成，其数值越大，图像也就越清晰。

（7）控制方式：由模拟式调节到现在的数码式调节，能对亮度、对比度、图像大小、位置、失真等屏幕参数进行调节。

1.1.6　声卡与音箱

声卡插在主机板的总线扩展槽中，也有的主机板上集成有声卡。机箱上对外留有用来接音箱、耳机、麦克风的插口。声卡的好坏取决于所用的声音处理芯片，同时也需要好的音箱。

音箱的选择通常看下述指标：

（1）频率响应：一般人耳的听力范围是 25Hz～20kHz。因此，音箱的频率响应至少要达到 40Hz～20kHz，这样才能保证基本覆盖人耳的有效听力范围。该指标范围越宽，音箱的性能越好。

（2）灵敏度：该指标越高，音箱的性能越好。一般计算机用户选择 85dB 左右即可。

（3）谐波失真：是指由于音箱所产生的谐振现象而导致声音重放失真。

（4）输出功率：这是标称功率而非峰值功率。一般，音箱功率越大，音质效果就越好。

1.1.7　网卡

网络接口卡简称网卡，插在主机板的总线扩展槽中，也有集成在主机板上的。机箱外留有电缆接口。一旦上网，计算机内信息可以通过网卡转换再通过网络电缆传输出去，外面信息通过网络电缆线传输到网卡上，经转换传入计算机。

网卡按传输速率分为 10、100、1000M，按传输介质分为粗缆接口、细缆接口、光纤接口等网卡。

1.1.8　机箱、鼠标、键盘

除上述设备外，微机中还有机箱、鼠标、键盘等。甚至配有手写板输入设备、语音输入设备、微型摄像输入设备、打印机设备、扫描仪等。

机箱是用来固定计算机主板，并将所有设备安装在其中的装置，在选择时要注意它的散热条件，电源系统是否稳定可靠，近来更考虑机箱的美观性。

鼠标是一种坐标定位输入设备，分为机械式、光电式等。机械式鼠标是通过鼠标下面的小球与桌面摩擦，小球与 4 个方向的电位器接触，测量出各方位的相对位移量，从而换算出在屏幕上的光标位置；光电式鼠标需要在一块布满小方格的平板上移动，安装在鼠标下面的光电转换装置可以定位鼠标在平板上的坐标点，从而确定在屏幕上的位置。

键盘是数字和字符输入装置，它是从早期的机械式打字机的基础上演变过来的输入设备，近年来已经研制出了无线键盘和无线鼠标。

1.1.9　微机的基本结构

微型计算机系统由硬件系统和软件系统两大部分构成。

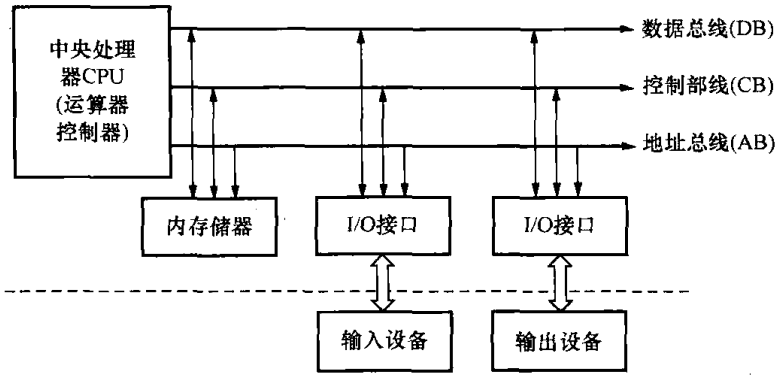


图 1.4 微型计算机的基本结构

以上介绍了微机系统的外观、主要构成，即硬件系统，俗称裸机。随着技术的不断进步，这些部件的功能与性能都会不断发展，但计算机的基本结构不变，如图 1.4 所示。

首先要有能进行运算的部件，称为运算器；其次要有能记忆程序、原始数据和中间结果以及为了使机器能自动进行运算而编制的各种命令的器件，这种器件就称为存储器；再次，要有控制器，能根据事先给定的命令发出各种控制信息，使整个计算过程能一步步地自动进行。原始的数据与命令要输入，要有输入设备；计算结果（或中间过程）要输出，要有输出设备。

目前，微型计算机都采用总线结构。所谓总线，就是用来传送信息的一组通信线，分内总线和外总线。内总线将微处理器、内存储器和输入、输出接口部件连接起来，分为地址总线、数据总线和控制总线。数据总线用来传送各种原始数据、中间结果、程序等。计算机的各种命令（即程序），以数据形式由存储器送入控制器，控制器经过译码后变为各种控制信号，由控制总线传送，以控制运算器按规定一步步地进行各种运算和处理，控制存储器的读和写，控制输入、输出设备的启停。CPU 从内存储器读写数据、程序等信息时，首先要给出信息所在的内存储器中的位置信息，即地址信息，由地址总线传送。微机与外部设备（包括其他计算机）的连接线称作外总线，其功能是实现计算机与计算机或计算机与其他外部设备的信息传送。

通常把运算器、控制器和存储器合在一起称为微机的主机；把各种输入、输出设备称为计算机的外围设备或外部设备（Peripheral）；把运算器和控制器合在一起称为中央处理单元 CPU；把整个 CPU 集成在一个集成电路芯片上，常称为微处理器（Microprocessor）；整个微型计算机只安装在一块印刷电路板上，常称为单板计算机；把整个计算机集成在一个芯片上，就称为单片机。不论计算机规模大小，必须同时具有 CPU、存储器和输入、输出设备，才能构成一台计算机。

微机软件系统分为系统软件和应用软件两大类。系统软件用来对构成微机的各部件进行管理和协调，使它们有条不紊、高效率地工作。如操作系统、各种高级语言、数据库系统等为系统软件；应用软件是针对不同应用、实现用户要求的功能软件；如因特网网站上的 Web 页、各部门的 MIS 程序、高校的综合教务管理等。

1.2 微型计算机的基本知识

1.2.1 计算机中的常用数制

人们通常用十进制数来记数和计算。而计算机只识别由“0”和“1”构成的二进制数。当一个较大的数用二进制来表示或人们在书写计算机程序时，数据位数既长又难记忆，常采用八进制、十六进制和十进制计数制。

表 1.1 列出了 4 种进位制中数的表示法，其中数后 B(Binary) 表示为二进制数；Q(Octal 的缩写为字母“O”，为区别于数字“0”写为“Q”) 表示为八进制数；H(Hexadecimal) 表示为十六进制数；D(Decimal) 表示为十进制数，可省略。

表 1.1　　十进制、二进制、八进制、十六进制数码对照表

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0	0000B	0Q	0H	8	1000B	10Q	8H
1	0001B	1Q	1H	9	1001B	11Q	9H
2	0010B	2Q	2H	10	1010B	12Q	AH
3	0011B	3Q	3H	11	1011B	13Q	BH
4	0100B	4Q	4H	12	1100B	14Q	CH
5	0101B	5Q	5H	13	1101B	15Q	DH
6	0110B	6Q	6H	14	1110B	16Q	EH
7	0111B	7Q	7H	15	1111B	17Q	FH

那什么是计数制呢？对于任意进制的数 N 均可表示为：

$$N = a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \cdots + a_0X^0 + a_{-1}X^{-1} + \cdots + a_{-(m-1)}X^{-(m-1)} + a_{-m}X^{-m}$$

式中，数 N 的大小不但取决于其中的数字本身，而且还取决于各数字所在的位置，故称为数的位置记数法。

其中，对每一个数位 a_{n-1}、…、a_0、a_{-1}、…、a_{-m} 赋予一定的位值 X^{n-1}、…、X^0、X^{-1}、…、X^{-m}，则称各位值为权。每个数位上的数字所表示的量是这个数字和权的乘积。相邻两位中高位的权与低位的权之比如果是个常数，则此常数称为基数，上式中用 X 表示。式中、从 a_0X^0 起向左是数的整数部分，向右是数的小数部分。数位 $a_i(n-1 \geqslant i \geqslant -m)$ 可以在 0、1、…、$X-1$ 共 X 种基数中任意取值。m 和 n 为幂指数，均为正整数。基数 X 取不同值，便得到不同进位计数制的表达式。

1. 十进制数

十进制数的表达式为：$(N)_{10} = \sum_{i=-m}^{n-1} a_i \times 10^i$。其特点是：$a_i$ 只能在 0～9 这 10 个数字中取值；每个数位上的权是 10 的 i 次幂；在加、减运算中，采用“逢十进一”、“借一当十”的规则。例如：

$$2346.18 = 2\times10^3 + 3\times10^2 + 4\times10^1 + 6\times10^0 + 1\times10^{-1} + 8\times10^{-2}$$

2. 二进制数

二进制数的表达式为：$(N)_2 = \sum_{i=-m}^{n-1} a_i \times 2^i$。其特点是：$a_i$ 只能是 0 或 1；每个数位上的权是 2 的 i 次幂；在加、减运算中，采用“逢二进一”、“借一当二”的规则。例如：

$$11\,001.101B = 1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$$

因为二进制计数制中只有 0 和 1 两个数字，用电路实现最为方便且运算也特别简单，所以电子计算机内部均采用此计数制。

3. 八进制数

八进制数的表达式为：$(N)_8 = \sum_{i=-m}^{n-1} a_i \times 8^i$。其特点是：$a_i$ 只能在 0～7 这 8 个数字中取

值；每个数位上的权是 8 的 i 次幂；在加、减运算中，采用“逢八进一”、“借一当八”的规则。例如：

$$257.16Q=2\times8^2+5\times8^1+7\times8^0+1\times8^{-1}+6\times8^{-2}$$

4. 十六进制数

十六进制数的表达式为：$(N)_{16}=\sum_{i=-m}^{n-1}a_i\times16^i$。其特点是：$a_i$ 只能在 0～15 这 16 个数字中取值（其中 0～9 这十个数字借用十进制中的数码，10～15 这 6 个数用 A、B、C、D、E、F 表示）；每个数位上的权是 16 的 i 次幂；在加、减法运算中，采用“逢十六进一”、“借一当十六”的规则。例如：

$$63BE.FAH=6\times16^3+3\times16^2+11\times16^1+14\times10^0+15\times16^{-1}+10\times16^{-2}$$

1.2.2　各种数制间的转换

人们习惯使用十进制数，而计算机只认识二进制数。为了书写方便，编写汇编程序时多采用十六进制数，因此在不同场合需要不同进位计数制之间的转换。

1. 非十进制数到十进制数的转换

非十进制数到十进制数的转换只要将它们按相应的权表达式展开，按十进制运算规则求表达式的值，即得到它所对应的十进制数。

[例 1.1]　将二进制数 1101.001B、十六进制数 1A.CH 转换为十进制数。

解　根据二进制数的权展开式，有：

$$1101.001B=1\times2^3+1\times2^2+0\times2^1+1\times2^0+0\times2^{-1}+0\times2^{-2}+1\times2^{-3}=13.125$$

根据十六进制数的权展开式，有

$$1A.CH=1\times16^1+10\times16^0+12\times16^{-1}=26.75$$

2. 十进制数转换为非十进制数

(1) 十进制数转换为二进制数。十进制数转换为二进制数时，整数和小数部分应分别转换。整数部分采用“除 2（基）取余”法，即连续除以 2 并取余数作为结果，直至商为 0，得到的余数从低位（最先得到的余数）到高位（最后得到的余数）依次排列即得到转换后二进制数的整数部分；对小数部分采用“乘 2（基）取整”法，即对小数部分连续乘 2，以最先得到的乘积的整数部分为最高位，直至达到所要求的精度或小数部分为零为止。

[例 1.2]　将十进制数 100.625 转化为二进制数。

解　整数部分：

```
2 | 100
  2 |  50 ……  余数 0
    2 |  25 ……  余数 0
      2 |  12 ……  余数 1
        2 |  6 ……  余数 0
          2 |  3 ……  余数 0
            2 |  1 ……  余数 1
              0 ……  余数 1
```

最低位 ↑　　最高位

小数部分：

```
0.625
×    2
1.250  ……  整数 1
×    2
0.500  ……  整数 0
×    2
1.000  ……  整数 1
```

最高位 ↓　　最低位

所以得 100.625＝1100100.101B

(2) 十进制数转换为八进制数、十六进制数。与十进制数转换为二进制数的方法类似，整数部分采用“除 8、16（基）取余”法，小数部分采用“乘 8、16（基）取整”法。

[例 1.3] 将十进制数 100.625 转化为十六进制数。

解　整数部分：

```
16 | 100
16 |  6   ····· 余数 4
     100  ····· 余数 6
```

小数部分：

```
  0.625
×   16
------
  3750
   625
------
 10.000   ····· 整数 A
```

所以得 100.625＝64.AH

同理得 100.625＝144.5Q

3. 二进制数与八进制数、十六进制数间的转换

由于 $2^4=16$，故一位十六进制数能够表示的数值恰好相当于四位二进制数能够表示的数值。据此将二进制数转换为十六进制数的方法是：对整数部分从小数点开始向左每四位一组，若最后一组不足 4 位，则在其左边补零直到 4 位；对小数部分从小数点开始向右每四位一组，若最后一组不足 4 位，则在其右边补零直到 4 位。然后将每组二进制数用对应的十六进制数代替，则得到转换结果。

[例 1.4] 将二进制 110011011.101101B 转换为十六进制数。

解　二进制数：0001　1001　1011.1011　0100

十六进制数：1　9　B.　B　4

所以得 110011011.101101B＝19B.B4H

与上述过程相反，可将十六进制数转换为二进制数，可自己试试。

由于 $2^3=8$，故一位八进制数恰好等于三位二进制数能够表示的数值。转换方法同上。

[例 1.5] 将二进制 1110011011.1011011B 转换为八进制数。

解　二进制数：001　110　011　011.101　101　100

十六进制数：1　6　3　3.　5　5　4

所以得 1110011011.1011011B＝1633.554Q

1.2.3　无符号二进制数

在编程解决实际问题时，首先要涉及到数据及数据类型。80X86 系列微机中，常用的数据类型包括无符号整数、有符号整数、BCD 数、字符串、位、浮点数。

在 80X86 系列微处理器中，参加运算的整数操作数可为 8 位长的字节、16 位长的字；80386/80486CPU 及其以后的微处理器中，参加运算的整数操作数还可为 32 位长的双字。整数分为无符号数和有符号数两种。所谓无符号数，是指对应的 8 位、16 位、32 位二进制数全部用来表示数值本身，没有用来表示符号位的位，因而为正整数。

1. 二进制的算术运算

(1) 加法运算：

0＋0＝0，0＋1＝1，1＋0＝1，1＋1＝0（有进位）

（2）减法运算：

0－0＝0，1－0＝1，1－1＝0，0－1＝1（有借位）

（3）乘法运算：

0＊0＝0，0＊1＝0，1＊0＝0，1＊1＝1

计算机中的乘法运算是通过加法和移位运算实现的。

（4）除法运算。除法是乘法的逆运算，计算机中是通过减法和右移运算实现的。每右移一位相当于除以 2，右移 n 位就相当于除以 2^n，手工计算时与十进制除法一样。

（5）无符号二进制数表示的范围。一个 n 位无符号整数所表示的数值范围是 $0\sim2^n-1$。例如：8 位无符号整数所表示的数值范围是 0～255；16 位无符号整数所表示的数值范围是 0～65535；32 位无符号整数所表示的数值范围是 0～(4G－1)，其中 G 为 2^{30}。

（6）无符号二进制数的溢出判断。最高有效位有进位或借位，则产生溢出。

2. 二进制数的逻辑运算

（1）“与”（又称逻辑乘）运算：

1∧1＝1，1∧0＝0，0∧1＝0，0∧0＝0

（2）“或”（又称逻辑加）运算：

0∨0＝0，0∨1＝1，1∨0＝1，1∨1＝1

（3）“非”运算。二进制 0 做逻辑非运算得 1，二进制 1 做逻辑非运算得 0。

（4）“异或”运算：

$0\oplus0=0$，$1\oplus1=0$，$0\oplus1=1$，$1\oplus0=1$

1.2.4 有符号数的表示方法

由于计算机只能识别 0 和 1 组成的数或代码，所以有符号数的符号也只能用 0 和 1 来表示，一般用 0 表示正，用 1 表示负，这种将符号数码化了，连同一个符号位在一起的一个数称为机器数。而直接用“＋”号和“－”号来表示其正负的数为有符号数（该机器数）的真值。例如：

机器数 01010101B，真值：＋85 或＋1010101B；

机器数 11010101B，真值：－85 或－1010101B。

但由于数值部分的表示方法不同，机器数有原码、反码和补码三种表示方法，它们同样具有字节、字及双字三种不同长度的整数类型。

1. 机器数的表示法

（1）原码。对一个二进制数而言，若用最高位表示数的符号（常以“0”表示正数，“1”表示负数），其余各位表示数值本身，则称为该机器数的原码表示法。例如，设 X＝＋1011100B，Y＝－1011100B，则 $[X]_{原}$＝01011100B，$[Y]_{原}$＝11011100B。$[X]_{原}$ 和 $[Y]_{原}$ 分别为 X 和 Y 的原码，称为机器数。下面介绍原码 $[X]_{原}$ 和真值 X 之间的关系：

当 $X\geqslant+0$ 时，$[X]_{原}=X$

当 $X\leqslant-0$ 时，$[X]_{原}=2^{n-1}-X$

其中，n 为二进制数位数，最高位为符号位。

值得注意的是，二进制原码表示中有＋0 和－0 之分，即：

$[+0]_{原}$＝000…00B（n 位二进制数，最高位为符号位）

$[-0]_{原}$＝100…00B（n 位二进制数，最高位为符号位）

(2) 补码。补码的定义根据同余的概念 $X+NK=X(\mathrm{mod}K)$，其中 K 为模，N 为任意整数。其含义是数 X 与该数加上其模的任意整数倍之和相等。

一个二进制数，若以 2^n 为模（n 为二进制数位数，它通常与计算机中机器数的长度一致），它的补码叫做 2 补码，以后把 2 补码简称补码，即：

当 $0\leqslant X\leqslant 2^{n-2}$ 时，$[X]_{补}=X$；

当 $-2^{n-1}\leqslant X\leqslant 0$ 时，$[X]_{补}=2^n+K$。

同理，一个十进制数若以 10^n 为模，它的补码叫做 10 补码。

可见，正数的补码与原码相同，只有负数才有求补码的问题。

1) 根据定义求补法。$[X]_{补}=2^n+X=2^n-|X|$，$X<0$，$\mathrm{mod}2^n$，即负数 X 的补码等于模 2^n 加上其真值，或者减去其真值的绝对值。例如，$X=-1010111\mathrm{B}$，$n=8$，则 $[X]_{补}=2^8+(-1010111\mathrm{B})=100000000\mathrm{B}-1010111\mathrm{B}=10101001\mathrm{B}(\mathrm{mod}2^8)$。

这种方法因为要做一次减法，很不方便。

2) 利用原码求补法。一个负数的补码等于其原码除符号位不变外，其余各位按位求反，再在最低位加 1，简称取反加 1。例如，$X=-1010111\mathrm{B}$，则 $[X]_{原}=11010111$，$[X]_{补}=10101000\mathrm{B}+1=10101001\mathrm{B}$。

如果将 $[X]_{补}$ 再求一次补，就得到 $[X]_{原}$，即有 $[\,[X]_{补}]_{补}=[X]_{原}$。例如，$[X]_{原}$ 11010111B，$[X]_{补}=10101001\mathrm{B}$，则 $[\,[X]_{补}]_{补}=11010110\mathrm{B}+1=11010111\mathrm{B}=[X]_{原}$。

上述求法数学上都可以证明，在此从略。

3) 简便的直接求补法。直接从原码求它的补码是：从最低位起，到出现第一个 1 以前（包括第一个 1）原码中的数字不变，以后逐位取反，但符号位不变。

[例 1.6]　试求 $X_1=-1010111\mathrm{B}$，$X_2=-1110000\mathrm{B}$ 及 $X_3=+1110001\mathrm{B}$ 的补码 $[X_1]_{补}$、$[X_2]_{补}$ 和 $[X_3]_{补}$。

解　方法自选，在此利用原码求。

$X_1=-1010111\mathrm{B}$

由原码求补码：$[X_1]_{原}=1\ \underbrace{1010111}_{取反}\mathrm{B}+1\rightarrow[X_1]_{补}=1010\ 1001\mathrm{B}$

符号位不变——↑　　↑——末位加 1

同理　$X_2=-1110000\mathrm{B}\rightarrow[X_2]_{原}=11110000\mathrm{B}\rightarrow[X_2]_{补}=10010000\mathrm{B}$

对于正整数　$X_3=+1110001\mathrm{B}$，$[X_3]_{原}=01110001\mathrm{B}=[X_3]_{补}$

值得注意的是：$[-128]_{补}=10000000\mathrm{B}$，$[0]_{补}=00000000\mathrm{B}$。

(3) 反码。若二进制数 $X=X_{n-1}X_{n-2}\cdots X_1X_0$，则反码表示的严格定义如下：

$$[X]_{反}=X \qquad (2^{n-1}>X\geqslant 0)$$

$$[X]_{反}=2^{n-1}-1+X \qquad (0\geqslant X>-2^{n-1})$$

其中，X 表示二进制数真值，其反码记为 $[X]_{反}$，n 表示包括符号位和数值部分在内的二进制数位数。

在求反码时，对正数来讲，其表示方法同原码。但对负数而言，其反码的数值部分为真值的各位按位取反，符号位不变。例如，若 $X=+10010\mathrm{B}=18$，$Y=-10010\mathrm{B}=-18$，设 n 为 8。则 $[X]_{原}=00010010\mathrm{B}$，$[Y]_{原}=10010010\mathrm{B}$，$[X]_{反}=00010010\mathrm{B}$，$[Y]_{反}=11101101\mathrm{B}$。

在反码表示中，数 0 也有两种表示形式：$[+0]_{反}=00000000\mathrm{B}$，$[-0]_{反}=11111111\mathrm{B}$。

2. 有符号数的运算

上面介绍了计算机中有符号数的表示方法：原码、补码及反码。原码法表示数时，最大的优点是直观。浮点数的有效数字常用原码表示，二进制乘除法运算时，也多采用原码表示法。

计算机中有符号二进制数进行加减运算时采用补码形式。由于补码进行减法运算可用加法来代替，且符号位也可以和数一起参加运算，使计算机的运算速度大大提高，同时也简化了计算机的硬件结构。即：

$[X\pm Y]_{补}=[X]_{补}+[\pm Y]_{补}$，其中 $|X|$、$|Y|$、$|X+Y|<2^{n-1}$。

[例 1.7] 用补码进行下列运算（设 $n=8$）：

① (+20)+(−13)　② (−20)+(+13)　③ (−20)+(−13)　④ (+20)+(+13)

解 ①
$$\begin{array}{rl} & 00010100B\ \ [+20]_{补} \\ +& 11110011B\ \ [-13]_{补} \\ \hline & 100000111B\ \ [+7]_{补} \end{array}$$

↑最高位（符号位）为 0，结果为正符号位的进位，自然丢掉

②
$$\begin{array}{rl} & 11101100B\ \ [-20]_{补} \\ +& 00001101B\ \ [+13]_{补} \\ \hline & 11111001B\ \ [-7]_{补} \end{array}$$

最高位（符号位）为 1，结果为负

③
$$\begin{array}{rl} & 11101100B\ \ [-20]_{补} \\ +& 11110011B\ \ [-13]_{补} \\ \hline & 111011111B\ \ [-33]_{补} \end{array}$$

↑最高位（符号位）为 1，结果为负符号位的进位，自然丢掉

④
$$\begin{array}{rl} & 00010100B\ \ [+20]_{补} \\ +& 00001101B\ \ [+13]_{补} \\ \hline & 00100001B\ \ [+33]_{补} \end{array}$$

最高位（符号位）为 0，结果为正

[例 1.8] 用补码进行下列运算（设 $n=8$）：

①96−19；　② (−56)−(−17)

解 ①$X=96$，$Y=19$，则

$[X]_{补}=[X]_{原}=01100000B$

$[Y]_{补}=[Y]_{原}=00010011B$

$[-Y]_{补}=11101101B$

$$\begin{array}{rl} & 01100000B\ \ [X]_{补} \\ +& 11101101B\ \ [-Y]_{补} \\ \hline & 01001101B\ \ [X-Y]_{补}=[X-Y]_{原}=+77 \end{array}$$

↑符号位为 0，结果为正

②$X=-56$，$Y=-17$，则

$[X]_{原}=10111000B$，$[X]_{补}=11001000B$

$[Y]_{原}=10010001B$，$[Y]_{补}=11101111B$

$[-Y]_{补}=00010001B$

$$\begin{array}{rl} & 11001000B\ \ [X]_{补} \\ +& 00010001B\ \ [-Y]_{补} \\ \hline & 11011001B\ \ [X-Y]_{补} \end{array}$$

符号位为 1，结果为负，对 $[X-Y]_{补}$ 再求补，得 $[X-Y]_{原}=10100111B$

前面曾讲过无符号数的运算规则，但计算机中两个无符号数到底是怎样做加减运算的呢？

两个无符号数进行加法，只要和的绝对值不超过整个字长，就不溢出，则和也一定为正数的补码形式，它等于和的原码；两个无符号数相减，可用减数变补与被减数相加来求得。所谓减数变补，是将整个减数各位取反后末位加 1。

[例 1.9] 两无符号数进行下列运算（$n=8$）：

①129−79　②79−129

解 ①设 $X=129$，$Y=79$，则

$X=10000001B$

$Y=01001111B$

$[-Y]_{补}=[Y]_{变补}=10110001B$

②设 $X=79$，$Y=129$，则

$X=01001111B$

$Y=10000001B$

$[-Y]_{补}=[Y]_{变补}=01111111B$

```
  10000001B   X
+ 10110001B   [−Y]变补
-----------------------
1 00110010B   [X−Y]补
↑_|
```

即 $[X-Y]_{原}=[X-Y]_{补}=00110010B=50$

```
  01001111B   X
+ 01111111B   [−Y]补
-----------------------
0 11001110B   [X−Y]补
↑_|
```

即 $[X-Y]_{补}=11001110B$，$[X-Y]_{原}=10110010B=-50$

综上所述，不管参加运算的两个 n 位二进制数是有符号的补码形式，还是无符号的数，对计算机来说，处理方法都一样。做加法时，直接将两数相加即可；做减法时，是用减数变补与被减数相加来实现。

实际运算时，若将补码加/减法的公式稍作改变，便可得：

$$[X\pm Y]_{补}=[X]_{补}+[\pm Y]_{补}=[X]_{补}\pm[Y]_{补}$$

即对加法来说，两数补码之和等于和的补码；对减法来说，两数补码之差等于差的补码。

3. 溢出判别

一个 n 位有符号二进制数补码所能表示的最大正数是 $2^{n-1}-1$，最小负数的数值是 -2^{n-1}。例如：

（1）8 位字长，用补码所表示的数值范围是－128～＋127；

（2）16 位字长，用补码所表示的数值范围是－32768～＋32767；

（3）32 位字长，用补码所表示的数值范围是－2G～＋(2G－1)。

当两个有符号的二进制数进行补码运算时，若运算结果超出上述范围，数值部分便会发生溢出，引起计算结果不正确。微机中常用的溢出判别法是双高位判别法。先引进两个附加的符号，即 C_s 和 C_p。

C_s：表示最高位（符号位）的进位（对加法）或借位（对减法）情况。如有进位或借位，$C_s=1$；否则，$C_s=0$。

C_p：表示数值部分最高位的进位（对加法）或借位（对减法）情况。如有进位或借位，$C_p=1$；否则，$C_p=0$。

判别产生溢出的依据是：若 C_s 和 C_p 同为 0 或同为 1，则结果无溢出发生，结果正确；若 $C_s=1$、$C_p=0$ 或 $C_s=0$、$C_p=1$，则结果发生溢出。

通过分析可知，只有两个正数或负数相加、正数减负数、负数减正数这四种情况有可能产生溢出。

[例 1.10] 试判断下面加法、减法的溢出情况（$n=8$）。

```
  01011010B   +90
+ 01101011B   +107
-----------------
  10011011B   +197
```

即 $C_s=0$、$C_p=1$，正溢出，结果出错。

（两个正数相加）

```
  10010010B   [−110]补
+ 10100100B   [−92]补
---------------------
  00110110B   →+54
```

即 $C_s=1$、$C_p=0$，负溢出，结果出错。

（两个负数相加）

```
  10010010B   [−110]补
− 00100000B   [+32]补
---------------------
  01110010B   [+114]补
```

即 $C_s=0$、$C_p=1$，负溢出，结果出错。

（负数减正数，结果应为正）

```
  01101011B   [+107]补
− 10100100B   [−92]补
---------------------
  11000111B   [−57]补
```

即 $C_s=1$、$C_p=0$，正溢出，结果出错。

（正数减负数，结果应为负）

4. 算术移位

二进制数在寄存器或存储器中进行算术移位时，每左移一位，其绝对值应增大一倍（如果没有溢出）；每右移一位，其绝对值应减小一半。为此，在不同情况下对溢出的空位应补入不同的数。

(1) 对于正数，左移或右移时的低位或高位都补以 0。设一个 8 位正二进制数为 00001110B(+14)，左移一位（最高位移出自然丢弃，最低位空位补 0）后为 00011100B (+28)；右移一位（最低位移出自然丢弃，最高位空位补 0）后为 00011100B(+7)。

(2) 补码法表示的负数，左移时最低位补以 0，右移时最高位补以 1。设一个 8 位负二进制补码数为 11110010B($[-14]_{补}$) 左移一位（最高位移出自然丢弃，最低位空位补 0）后为 11100100B($[-28]_{补}$)；右移一位（最低位移出自然丢弃，最高位空位补 1）后为 11111001B($[-7]_{补}$)。

1.2.5 计算机中信息的编码

计算机除了处理进位制的数据外，在实际应用中需要处理不同语种的文字符号和各种图像信息等。为了处理这些信息，计算机中采用编码的形式。常用的有三种编码表示：十进制的二进制编码表示，字母与字符的二进制编码表示及汉字的二进制编码表示。

1. 十进制数的二进制编码表示

实际应用中，一般计算问题的原始数据大多数是十进制数。为了使计算机能输入、输出和直接进行十进制数的运算，必须用二进制数为它编码，称二—十进制码或 BCD 码 (Binary Coded Decimal)。BCD 码的编码形式很多，最常用的 BCD 码是四位二进制数的权分别为 8、4、2、1 的 BCD 码，称为 8421BCD 码，如表 1.2 所示。例如，若 BCD 码为 0001 0101 0011.0110 0101B（或 153.65H），则很容易写出相应的十进制数为 153.65。

表 1.2　　BCD 编 码 表

十 进 制	8421 BCD 码	十 进 制	8421 BCD 码
0	0000B	8	1000B
1	0001B	9	1001B
2	0010B	10	0001 0000B
3	0011B	11	0001 0001B
4	0100B	12	0001 0010B
5	0101B	13	0001 0011B
6	0110B	14	0001 0100B
7	0111B	15	0001 0101B

2. 字母与字符的二进制编码表示

由于计算机硬件只能识别二进制数，字母和字符也必须用二进制编码来表示。目前，用来表示字母和字符的二进制编码方式有多种，最常用的是 ASCII 码，即美国信息交换标准码 (American Standard Code for Information Interchange)。它是 7 位 ASCII 码，可表示 128 (0000000B～1111111B) 种字符，详见附录 1，其中包括字母、数字和控制符号。例如，字母 A 的 ASCII 码为 100 0001B 或 41H；英文大写字母 A～Z 的 ASCII 码为 41H～5AH；英文小写字母 a～z 的 ASCII 码是 61H～7AH；数字 0～9 的 ASCII 码为 30H～39H 等。

3. 汉字的二进制编码表示

计算机要处理汉字信息，就必须首先解决汉字的表示问题。同英文字符一样，汉字的表示也采用二进制编码形式，目前普遍使用的是 1981 年我国制定的中华人民共和国标准信息交换用汉字编码，即 GB 2312—1980 国标码。该标准共包含一、二级汉字 6763 个，其他符号 682 个（新的国标汉字库已包括两万多个汉字和字符）。该标准编码分为以下三部分。

（1）字母、数字和符号共 682 个，它们是：

1）包括间隔符、标点、运算符、单位符号和制表符在内的一般符号 202 个；

2）序号 60 个，包括 1.～20.、（1）～（20）、①～⑩、（一）～（十）；

3）数字 22 个，包括 0～9 和Ⅰ～Ⅻ；

4）英文字母大小写共 52 个；

5）日文平假名 83 个；

6）日文片假名 86 个；

7）希腊字母大小写各 24 个，共 48 个；

8）俄文字母大小写各 33 个，共 66 个；

9）汉语拼音符号 26 个；

10）汉语注音字母 37 个。

（2）一级常用汉字 3755 个，按汉语拼音排列。

（3）二级常用汉字 3008 个，按偏旁部首排列。

GB 2312 国标字符集是排列成 94 行×94 列的二维码表。每个汉字或符号在码表中都有各自固定的位置，对应着一个唯一的位置编码，即该汉字或符号所在的行号（也称区号）和列号（也称位号）组成的二进制编码（区号及位号均为 7 位，且区号在左、位号在右，共 14 位二进制编码），称为区位码。在实际应用中，将区位码的区号和位号分别加 100000B（32），便形成相应的国标码，例如汉字“常”的区位码为 0010011B 0000011B(13H 03H)，则对应的国标码为 0110011B 0100011B(33H 23H)。

为区别汉字编码与 ASCII 码，汉字在存储和传送时以内码表示。所谓汉字的内码，是将国标码的 7 位区号及位号的最高位分别置“1”扩展为 8 位而成。例如，汉字“常”的国标码为 011 0011B 010 0011B(33H 23H)，机器内码为 1011 0011B 1010 0011B(B3H A3H)。

1.2.6　定点数与数学协处理器的数据

根据在定点数中小数点位置的固定方法不同，分为定点整数及定点小数表示法。定点整数是小数点固定在数的最低位之后，定点小数是小数点固定在数的最高位之前。定点整数表示比较简单，但能表示的数值范围较小，运算时易超出范围而溢出，为此引入数学协处理器。

数学协处理器 8087、80287、80387 分别与 8086/80186、80286、80386 相配，80486 的数学协处理器集成在芯片内部。数学协处理器的主要任务是支持浮点运算。80387 等数学协处理器可支持 7 种数据类型。

1. 整型数

80387 支持字整型（16 位）、短整型（32 位）、长整型（64 位）三种整型数格式，它们都以二进制补码形式表示。80387 支持的字整数和短整数分别与 80386 支持的带符号字及双字整数相对应；80387 的长整数 80386 不支持，而 80386 的字节整数 80387 不支持。

2. BCD码

80387支持的BCD码数为80个二进制位的压缩BCD码数，即可以表示20位BCD码数。但在80387支持的BCD码数中，只用了18位BCD码数，而最高8位二进制数中用一位二进制数表示BCD码数的符号（该位为0表示为正，为1表示为负），其余7位二进制数没有用。其中，每一位BCD码数均采用8421 BCD码。

3. 实型数

80387支持的实型数有短实型、长实型和临时实型。任何一个二进制数可以表示成如下形式：$N=2^{\pm E}\times(\pm S)$，由阶符（±）、阶码（E）、数符（±）、有效数字或尾数（S）四部分组成，在计算机中若把一个二进制数分成指数（$2^{\pm E}$）和尾数（$\pm S$）两部分来表示，叫做浮点表示法（或称科学表示法）。80387中一般实型数格式如图1.5所示，其中数符表示实型数的符号，若数符位为1，表示该数为负；反之为正。阶符表示阶码的符号，与阶码一起指明了实型数中小数点的位置。尾数是实型数中的有效数字部分。80387规定，任何实型数只能用如下格式表示：1.××…××·2^n（×表示1或0），即实型数中的尾数部分小数点前有且仅有一位，该位永远为1。

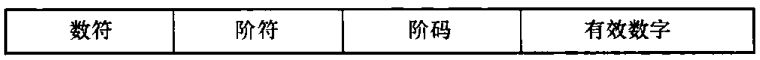


图1.5 一般实型数格式

对32位的短实型数来说，数符1位、阶符1位、阶码7位、有效数字23位；对64位的长实型数来说，数符1位、阶符1位、阶码10位、有效数字52位；对80位的临时实型数来说，数符1位、阶符1位、阶码14位、有效数字64位。例如，对于80387若尾数为0101…001B，则它的有效数字实际应为1.0101…001B。

1.3 常用术语解析

1. 位和字节

（1）位（bit）：是计算机所能表示的最小、最基本的数据单位，它指的是取值只能为0或1的一个二进制数值位。位作为单位时记作b。

（2）字节（byte）：由8个二进制位组成，通常用作计算存储容量的单位。字节作为单位时记作B。

（3）K是kelo的缩写，1K=1024=2^{10}。

（4）M是mega的缩写，1M=1024K=2^{20}。

（5）G是Giga的缩写，1G=1024M=2^{30}。

（6）T是tera的缩写，1T=1024G=2^{40}。

2. 主频

（1）主频也叫做时钟频率，用来表示微处理器的运行速度，主频越高表明微处理器运行越快，主频的单位是MHz。早期微处理器的主频与外部总线的频率相同，从80486DX2开始，主频=外部总线频率×倍频系数。

（2）外部总线频率通常简称为外频，它的单位也是MHz，外频越高说明微处理器与系统内存数据交换的速度越快，因而微型计算机的运行速度也越快。

（3）倍频系数是微处理器的主频与外频之间的相对比例系数。

（4）通过提高外频或倍频系数，可以使微处理器工作在比标称主频更高的时钟频率上，这就是所谓的超频。

3. MIPS

MIPS 是 Millions of Instruction Per Second 的缩写，用来表示微处理器的性能，意思是每秒钟能执行多少百万条指令。主频为 25MHz 的 80486，其性能大约是 20MIPS，主频为 400MHz 的 Pentium Ⅱ的性能为 832MIPS。

4. 微处理器的生产工艺

它指在硅材料上生产微处理器时内部各元器件间连接线的宽度，一般以 mm 为单位，数值越小，生产工艺越先进，微处理器的功耗和发热量越小。目前微处理器的生产工艺已经达到 0.18μm。

5. 数据在内存储器中的存储方式

微机中的内存储器（简称内存）用来存储参加运算的操作数、运算的中间结果和最后结果。数据在内存中常以字节为单位进行存储，这样的存储空间称为一个存储单元。每个存储单元分配一个编号，这个编号就称为内存的一个地址，存储单元的地址从 0 开始，直到 CPU 所能支持的最高地址。例如，80X86 有 32 根地址总线，共有 2^{32}（4G）个存储单元。

通常称两个相邻字节组成的 16 位二进制数为一个字（Word），称 4 个相邻字节组成的 32 位二进制数为一个双字（Dword 或 Double Word）。80X86 系列微机中，多字节数据的存储按高位字节在地址号高的存储单元中，低位字节在地址号低的存储单元中的规则排列，如图 1.6 所示。

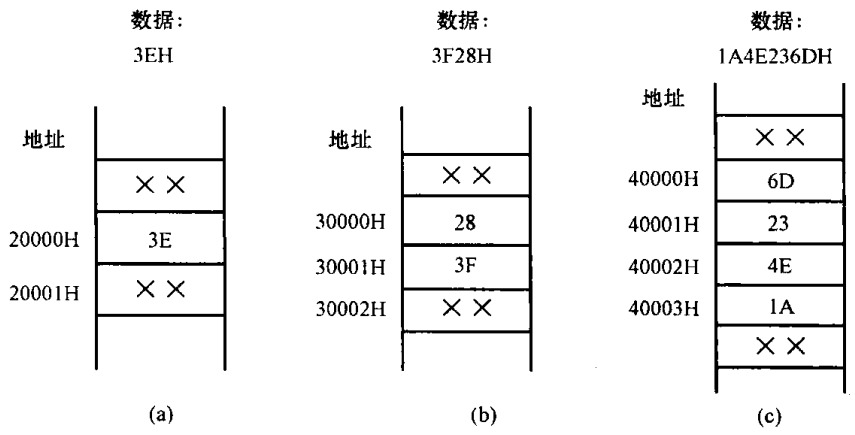


图 1.6　多字节数据的存储

（a）字节存储；（b）字存储；（c）双字存储

对一个字节数据，若用来存储它的存储单元地址号为 20000H，则称该字节地址为 20000H。对一个字数据，在内存中占两个相邻的存储单元，低位字节的地址号称为该字的地址，如图 1.6（b）所示称字数据 3F28H 的地址为 30000H。依此类推，称双字数据 IA4E236DH 的地址为 40000H。

6. 字符串

字符串是 80X86 系列微处理器处理的数据类型之一。字符串包括字节串、字串和双字串（在 80386/80486 CPU 及其以后的微处理器中）。所谓字节串的字符串，在计算机处理时

以一个个字符为单位（8 位二进制）；所谓字串的字符串，在计算机处理时以两个字符为单位（16 位二进制）；所谓双字串的字符串，在计算机处理时以四个字符为单位（32 位二进制）。它们分别对应上述数据存储的字节、字、双字的相邻序列。

1.4 计 算 机 基 础

下面简化设计了一个假想的初级计算机的模型机，通过介绍指令及程序的执行过程来理解计算机整机的工作原理。

1.4.1 指令程序和指令系统

指令是一组二进制编码信息，它主要包括两个内容：告诉计算机进行什么操作；指出操作数或操作数地址。通常，一条指令执行一种操作，因而，要解一个数学题目，必须先按解题步骤把所需指令按顺序排好。例如求“$6a+2b$”的值，可以先安排进行 $6a$ 操作的指令，然后安排进行 $2b$ 操作的指令，再安排进行“$6a+2b$”操作的指令。这种按解题顺序编排好的、用一系列指令表示的计算步骤叫做程序。计算机执行一个解题程序时，便按顺序执行这些指令。如果需要改变指令的执行顺序，也由指令给出。改变顺序之前，有时要根据某些条件，判断是否改变顺序，这些条件也要由指令规定。因而，可以说要让计算机能自动完成某项运算或数据处理任务，不仅需要计算机硬件的支持，而且还需要计算机软件（程序）的配合。一台计算机所能执行的各种不同指令的集合叫做计算机的指令系统。每台计算机均有自己特定的指令系统。这个指令系统反映了计算机的基本功能，是在设计计算机时规定好的。

在计算机中，指令都由二进制编码表示，包括操作码和操作数两部分。所以，一条指令的基本格式如下：

操作码	操作数

整条指令以二进制编码的形式存放在存储器中，该二进制编码称为指令的机器代码或简称指令码。在微型计算机中，一个字节通常不能充分表示各种操作码和操作数，故有一字节指令、二字节指令或多字节指令。

早期的计算机直接采用机器码进行程序设计。由于机器码不便记忆和理解，编程非常困难和繁琐，且易出错。为此，人们采用指令助记符（Memonic，通常是指令功能的英文缩写）来代替指令机器码中的操作码，而用一些符号（Symbol）来代替操作数或操作数地址。例如，8086/8088CPU 中指令“ADD AL，[3000H]”表示将内存单元地址号为 3000H 的内容与 AL 的内容相加，结果送回 AL 寄存器。用指令助记符代替指令机器码进行编程称为汇编语言编程。显然，这比机器语言编程方便了许多，但必须要配备翻译程序把助记符和操作数或操作数地址的符号翻译成机器码，计算机才能识别。

1.4.2 初级计算机的基本结构

1. 概述

初级计算机的结构框图如图 1.7 所示。图中虚线以上部分为运算器和控制器，构成计算机的中央处理单元（Central Processing Unit），简称 CPU；虚线以下为 RAM 和 I/O 接口电路。RAM 的容量为 256×8，即 256 个存储单元，每个存储单元存 8 位二进制数。I/O 接口

电路是输入设备键盘、输出设备显示器（CRT）和 CPU 之间的缓冲和连接部件。键盘用来输入原始操作数及解题程序，CRT 用来显示运算结果。

通常，在微型计算机中，中央处理单元 CPU 做成一个独立的芯片，称为微处理器或微处理机。存储器是位于 CPU 之外的另一种芯片，称为内存储器或主存储器。它是计算机的一个记忆装置，用来存放以二进制编码形式表示的程序、原始操作数、运算和处理的中间结果及最后结果，需要时可以把它们读出来。

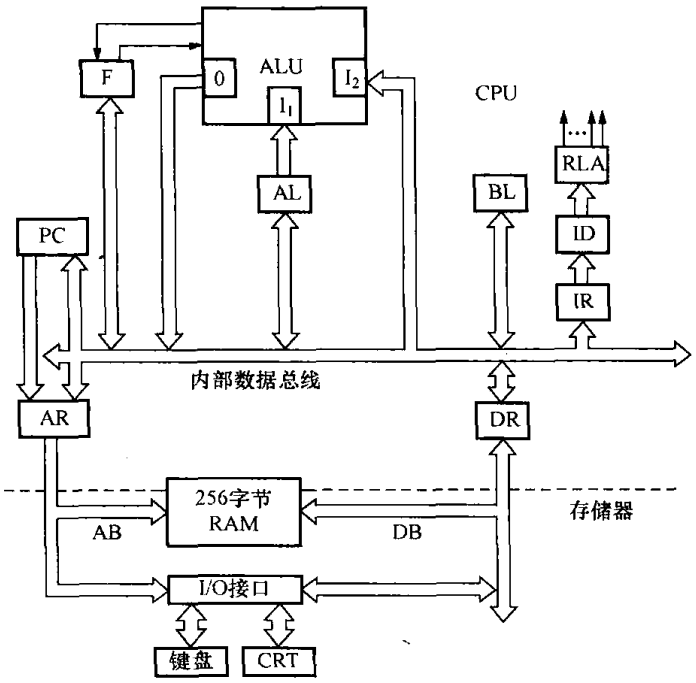


图 1.7　初级计算机的结构框图

一个存储器包含很多存储单元，被存储的二进制信息分别存放在这些存储单元中。每个存储单元都有自己的编号，这个编号叫做地址。计算机中存储单元的地址也采用二进制编码方式表示。例如，本模型机中存储器有 256 个存储单元，地址编号可以是 0～255。256 个存储单元若要用二进制编码表示地址号时，需 8 位二进制数，即二进制和十六进制表示的地址编号分别为 00000000B～11111111B 和 00H～FFH。显然，存储器的存储单元数越多，所需要的地址信号位数就越多，如果 1024 个存储单元需要 10 位二进制的地址信号（$2^{10}=1024$）。存储单元数是地址信号位数 N 的幂指数关系，即存储单元数$=2^N$。在微型计算机中，存储器的每个存储单元一般存放一个字节的二进制信息，一个字节为 8 位二进制数。这里必须注意，存储单元的地址及存储单元的内容是两个截然不同的概念，不得混淆。存储器中所存二进制编码信息基本上分两类，一类是程序中的指令码，一类是操作数。它们在外表上毫无区别，但程序设计人员必须十分清楚哪些地址编号的存储单元存放的是指令码，而哪些地址编号的存储单元存放的是操作数。这样，存储单元的地址编号也分两类，即指令地址和操作数地址。在微型计算机执行程序时，存储单元的二进制地址信号，不管是指令地址还是操作数地址均由 CPU 发出，以便选中某个存储单元，并对其进行存取操作。

CPU 内部各部件通过内部数据总线交换信息，该内部数据总线经具有三态控制功能的数据寄存器 DR(Data Resister) 驱动后与外部数据总线 DB(Data Bus) 相连。CPU 之外的内存储器及 I/O 接口电路均通过外部数据总线 DB 与 CPU 相连。CPU 对存储单元进行操作时，所读/写的二进制编码信息（即存储单元内容）必须经外部数据总线 DB 进行。

2. 运算器的结构和功能

运算器主要包括算术逻辑部件（Arithmetic Logic Unit，ALU）、累加器（Accumulator，AL）、操作数暂存器（Operand Temprary Register，OTR）及标志寄存器（Flags）。它们均是 8 位的。ALU 一般是直接执行各种操作和运算的部件，它在控制器的控制下完成各种算术运算（加、减、乘、除等）、逻辑运算（与、或，非、异或等）以及其他操作（取

数、送数、移位等)。在初级计算机中，假设 ALU 仅为一个 8 位全加器，可以实现两个 8 位补码数的加、减运算。累加器 AL 用来存放两个操作数中的一个，如被加数或被减数，且存放运算结果。运算的最终结果通过 CRT 显示出来。操作数暂存器用来存放两个操作数中的另一个，如加数或减数。累加器和操作数暂存器都属于 CPU 内部的通用寄存器，用来存放操作数和运算的中间结果。微型计算机中通用寄存器的数目有多个，组成一个通用寄存器组。原始操作数通常通过键盘送入内存，CPU 执行程序时再从内存取出。标志寄存器用来存放运算结果的标志信息，如运算结果的正、负情况用符号标志 SF（Sign Flag）来表示。在不溢出的情况下，若运算结果最高位为 1，则 SF=1，表明结果为负；否则 SF=0，表明结果为正。又如运算结果最高位的进位（对加法）或借位（对减法）情况用进位/借位标志 CF (Cary Flag) 来表示。加法运算时，若最高位有进位，则 CF=1；否则 CF=0。减法运算时，若最高位有借位，则 CF=1；否则 CF=0。再如，运算结果的溢出情况可用溢出标志 OF (Overflow Flag) 来表示。若运算结果有溢出，则 OF=1；否则 OF=0。

3. 控制器的结构和功能

(1) 控制器的结构。初级计算机中的控制器主要包括程序计数器（Program Counter）或指令指针 IP(Instruction Pointer)、指令寄存器 IR(Instruction Register)、指令译码器 ID (Instruction Decode)、内存地址寄存器 MAR(Memory Address Register) 及定时与控制部件 TC(Timing and Control) 等。计算机的整个工作过程就是执行程序的过程。程序就是一系列按一定顺序排列的指令。控制器靠指令指挥计算机工作，人则用指令表达自己的意图并交给控制器指挥机器执行。

(2) 控制器的功能。我们已知道，为使计算机能自动执行一个解题程序，必须将程序中的指令按解题顺序预先存放在内存中。同时，计算机在执行程序时应能把这些指令自动按顺序逐条取出并执行。为此，要有一个追踪指令地址的程序计数器 PC。程序开始执行时，由计算机的操作系统给 PC 赋一个初值，这个初值就是内存中要执行的解题程序的起始地址。然后，每取出一个指令字节，PC 的内容便自动加 1，指向下一个指令字节地址，从而保证指令的顺序执行。只有当程序需要转移时，PC 才置入转移到所需要的指令处的新值。

为了完成一条指令所规定的操作，计算机的运算器、内存储器等部件必须在控制器的控制下，相应地完成一系列基本动作，而这些基本动作又必须按时间先后次序互相配合。为此，首先必须由指令寄存器 IR 根据 PC 所指指令地址接收要被执行的指令操作码，直接送指令译码器 ID 进行译码。然后，该操作码译码信号便作为一条指令的特征信号，送定时及控制部件，变成一系列按时间顺序排列的控制信号，发向运算器、控制器、存储器及 I/O 接口电路等，从而控制它们完成该指令所规定的操作。

如果指令执行中需要从内存或 I/O 接口中取操作数，该操作数的地址由指令的操作数地址部分给出。内存地址寄存器 MAR 具有三态控制功能，它接收 PC 发来的指令地址或来自指令的操作数地址，经三态控制并驱动后，至外部地址总线 AB 送存储器或 I/O 接口电路。地址总线 AB 是单向的，即存储器或 I/O 接口电路的地址信号均由 CPU 发出。

由于 MAR 及 DR 具有三态控制功能，只有当 CPU 需要通过 AB 及 DB 访问存储器或 I/O接口时，CPU 内部的地址信号、数据信号才能与 AB 及 DB 连通；否则，它们之间会呈现高阻（即断开）状态。

4. 内存储器的结构及工作原理

设初级计算机中的内存储器由 256 个存储单元组成，每个存储单元存放 8 位二进制信息，该内存储器的结构框图如图 1.8 所示。由图可见，它由存储阵列、地址译码器、三态数据缓冲器及控制电路组成。在本例中，256 个存储单元其地址号为：00H，01H，…，FFH 等。从 CPU 发来的 8 位指令地址或操作数地址经地址总线 AB 送入存储器的地址译码器进行译码，从而选中所需要的存储单元，进行读/写操作。

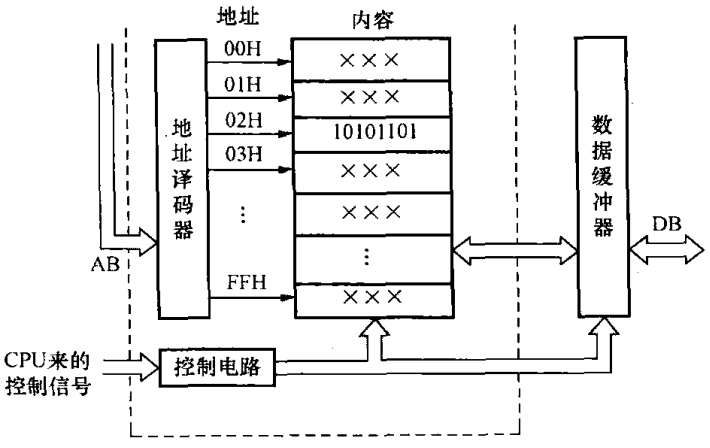


图 1.8　初级计算机的内存储器结构

三态数据缓冲器对 RAM 来说是双向的，它用来将存储单元的数据进行三态缓冲控制后，与 CPU 的外部数据总线 DB 相连。只有当存储器中的存储单元被选中时，存储器内部的数据信号线才能与外部数据总线 DB 连通；否则，将呈现高阻（即断开）状态。存储器中的控制电路则接收来自 CPU 的控制信号（如“访问存储器信号”、“读信号”或“写信号”等），经组合变换后对存储器的地址译码、数据存取操作等进行控制。

（1）读操作。如图 1.9 所示，若设图中 02H 存储单元的内容为 10100011B（即 A3H），则读出时，首先由 CPU 的 MAR 发出 8 位二进制地址信号 00000010B（即 02H），经地址总线 AB 送存储器的地址译码器。同时，由 CPU 的控制部件发出“访问存储器信号”和“读信号”等，这些控制信号经存储器的控制电路组合变换后送往存储器的相应部件，从而选中 02H 号存储单元，并将其内容 A3H 经三态数据缓冲器及 DB 送 CPU 中某寄存器（如累加器 AL）。数据信息 A3H 从存储单元 02H 中读出至 CPU 后，02H 中的内容保持不变。

（2）写操作。如图 1.10 所示，若要把 CPU 中某寄存器的内容写入到存储器的某存储单元中，则首先由 CPU 的 MAR 发出地址信号（如 01H），经地址总线 AB 送存储器的地址译码器，并由 CPU 的控制部件发出“访问存储器信号”及“写信号”等，经控制总线 CB 送往存储器相应部件，从而选中 01H 号单元。然后，CPU 就将要写入的数据经 DB 写入到 01H 号单元。数据写入 01H 号单元后，该单元原存数据将被新写入的数据信息所代替。若 01H 单元原内容为 0AH，AL 中的内容为 05H，则将 AL 的内容写入 01H 单元后，01H 单元的内容变为 05H，AL 中的内容不变。

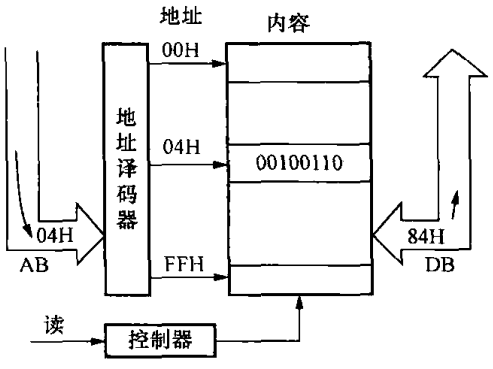


图 1.9　存储器读操作示意图

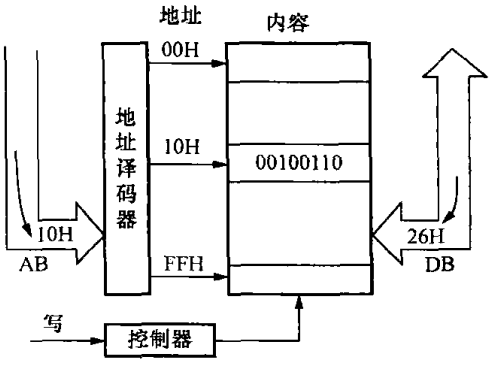


图 1.10　存储器写操作示意图

1.4.3 简单程序举例

下面以一个简单的例子来说明程序执行的过程。例如，求两个数 7 和 10 的和。首先要为这样的操作编制一个程序。用助记符形式表示的程序为：

```
MOV    AL,7        ;数 7 送 AL 寄存器
ADD    AL,10       ;AL 寄存器内容与数 10 做加法,和值保存在 AL 寄存器中
HLT                ;CPU 暂停
```

但是，模型机并不认得助记符，指令必须用机器码来表示，操作数同样也只能用二进制或十六进制表示。则对应的机器码程序如下：

```
1011 0000        ;第一条指令 MOV AL,n
0000 0111        ;n = 7
0000 0100        ;第二条指令 ADD AL,n
0000 1010        ;n = 10
1111 0100        ;第三条指令 HLT
```

总共是三条指令 5 个字节。

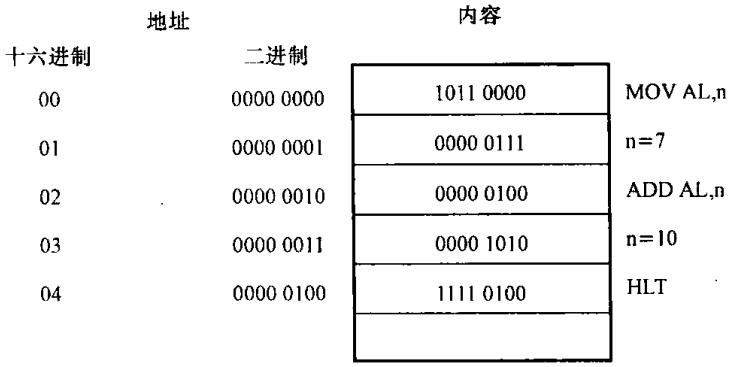


图 1.11 指令的存放

第二步程序应该放入存储器中，若它们放在以 00H（两位十六进制数）开始的存储单元内，则需要 5 个存储单元，如图 1.11 所示。

第三步是执行程序。在执行时，给 PC 赋以第一条指令的地址 00H，然后就进入第一条指令的取指阶段，具体操作过程如下：

1）PC 的内容（00H）送至地址寄存器；

2）当 PC 的内容可靠地送入地址寄存器后，PC 的内容自动加 1 变为 01H；

3）地址寄存器把地址号 00H 通过地址总线送至存储器，经过地址译码器译码，选中 00 号单元；

4）CPU 给出读命令；

5）所选中的 00 号单元的内容 B0H 读出至数据总线上；

6）读出的内容经过数据总线送至数据寄存器；

7）因是取指阶段，取出的是指令，故 DR 把它送至指令寄存器 IR，然后经过译码发出执行这条指令的各种控制命令。

取出第一条指令的操作过程如图 1.12 所示。

此后就转入了执行第一条指令的阶段。经过对操作码译码后知道，这是一条把操作数送累加器 AL 的指令，而操作数在指令的第二个字节。所以，执行第一条指令就必须把指令第二个字节中的操作数取出来。

取指令第二个字节的过程为：

1）把 PC 的内容 01H 送至地址寄存器；

2）待 PC 的内容可靠地送至地址寄存器后，PC 自动加 1 变为 02H；

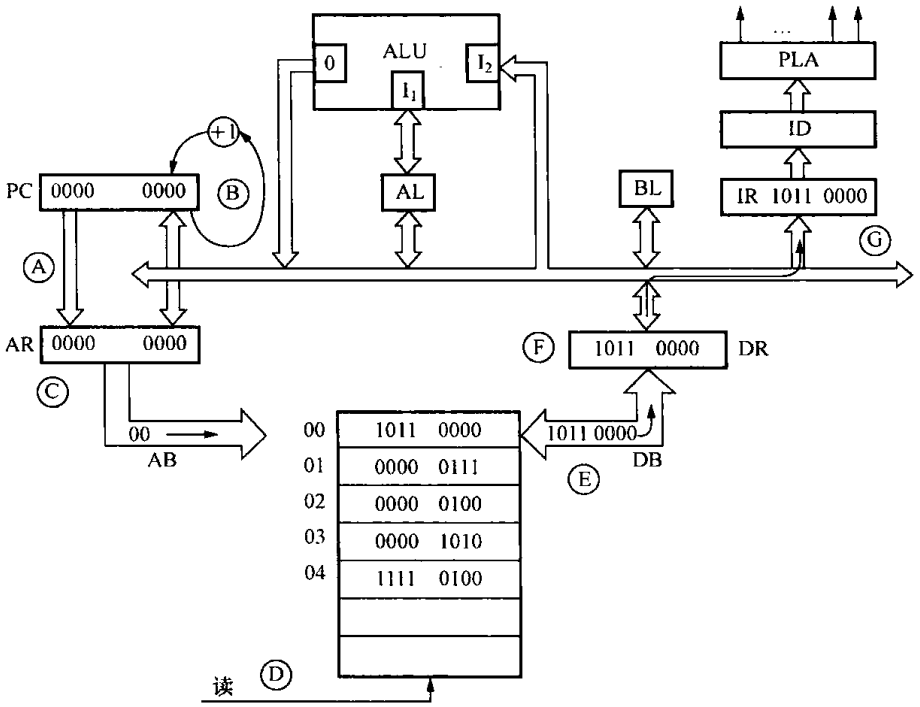


图 1.12　取第一指令的操作过程

3）地址寄存器通过地址总线把地址号 01H 送至存储器，经过译码选中相应的 01 号存储单元；

4）CPU 发出读指令；

5）选中的存储单元的内容 07H 读至数据总线上；

6）通过数据总线，把读出的内容送至 DR；

7）因为已读出的是操作数，且指令要求把它送累加器 AL，故由 DR 通过内部数据总线送至 AL。

取第一条指令的操作数的过程如图 1.13 所示。

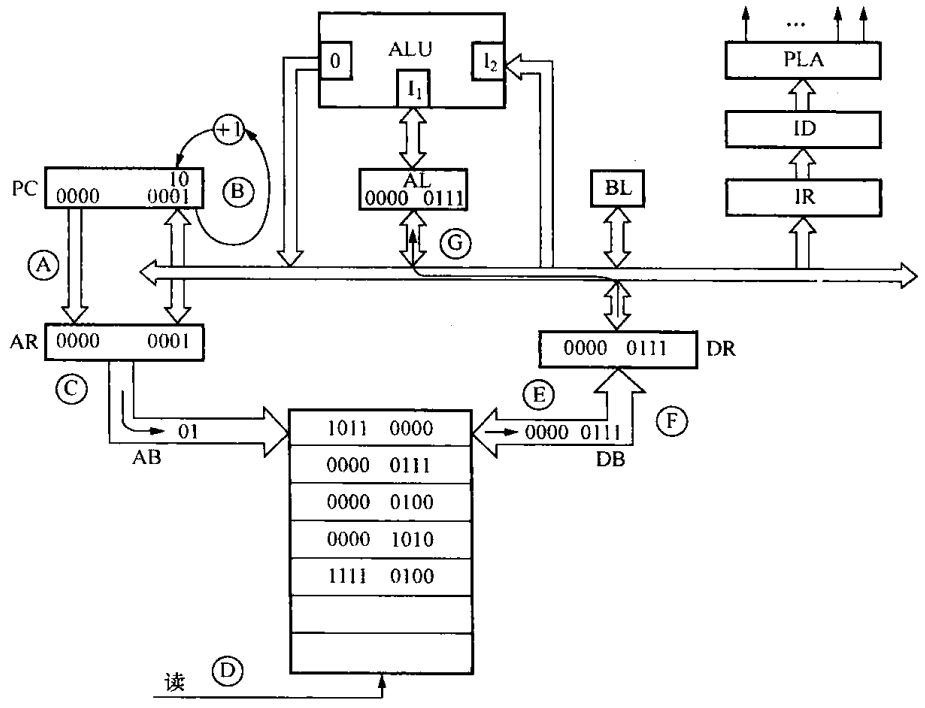


图 1.13　取第一条指令的操作数的过程

至此，第一条指令执行完毕，进入第二条指令的取指阶段。

取第二条指令的过程为：

1）把 PC 的内容 02H 送至地址寄存器；

2）在 PC 的内容已可靠送入地址寄存器后，PC 自动加 1；

3）AR 通过地址总线把地址号 02H 送至存储器，经译码后，选中相应的 02 号存储单元；

4）CPU 发出读命令；

5）选中的存储单元的内容 04H，读出到数据总线上；

6）读出的内容通过数据总线送至 DR；

7）因是取指阶段，所以读出的为指令，DR 把它送至 IR，经过译码后发出各种控制信息。

取第二条指令的过程如图 1.14 所示。

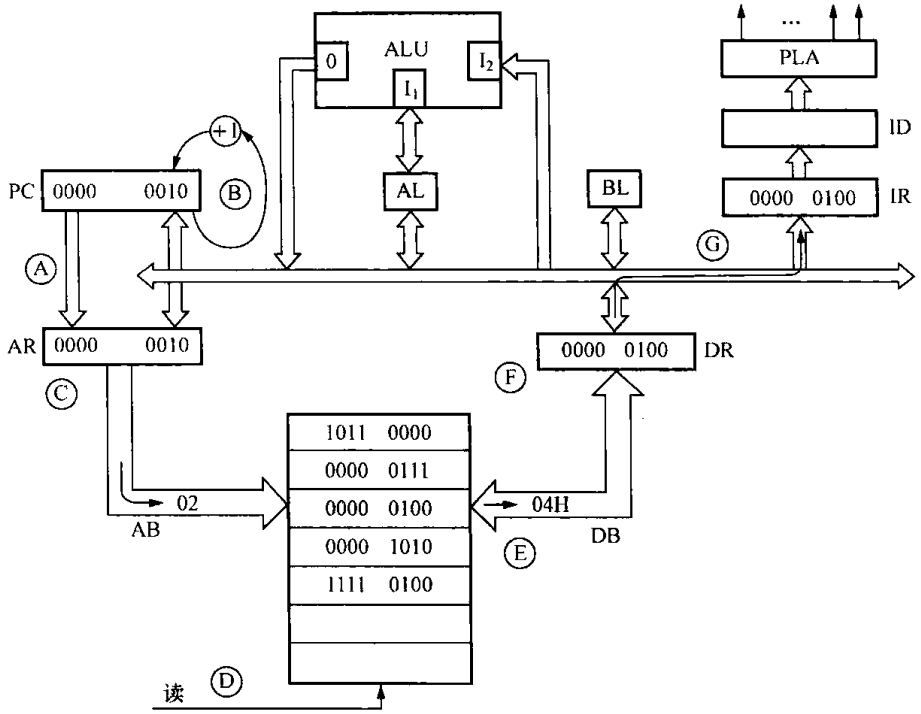


图 1.14 取第二指令的过程

经过对指令译码后知道，此为加法指令，以 AL 的内容为一个操作数，另一个操作数在指令的第二个字节中，执行第二条指令，必须取出指令的第二字节。

取第二条指令的第二字节及执行指令的过程为：

1）把 PC 的内容 03H 送至 AR；

2）当把 PC 内容可靠地送至 AR 以后，PC 自动加 1；

3）AR 通过地址总线把地址号 03H 送至存储器，经过译码，选中相应的 03 号存储单元；

4）CPU 发出读命令；

5）选中的存储单元的内容 0AH 读出至数据总线上；

6）数据通过数据总线送至 DR；

7）因为由指令译码已知读出的为操作数，而且要与 AL 中的内容相加，故数据由 DR 通过内部数据总线送至 ALU 的另一个输入端；

8）AL 中的内容送 ALU，且执行加法操作；

9）相加的结果由 ALU 输出至累加器 AL 中。

执行第二条指令的操作过程如图 1.15 所示。

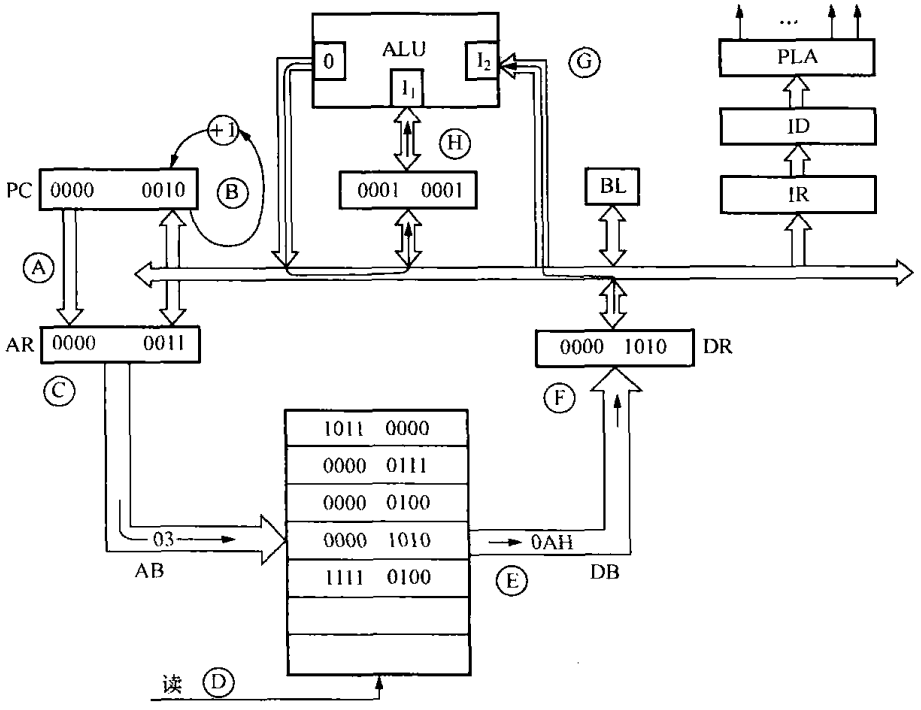


图 1.15　执行第二指令的操作过程

至此，第二条指令的执行阶段结束了，就转入第三条指令的取指阶段。

按照上述类似的过程取出第三条指令，经译码后就停机。

习　　题

1.1　计算机中常用的计数制有哪些?

1.2　请简述机器数和真值的概念。

1.3　将下列十进制数分别转化为二进制数、八进制数、十六进制数和压缩 BCD 数。

(1) 125.74　　(2) 513.85　　(3) 742.24　　(4) 69.357

1.4　将下列二进制数分别转化为十进制数、八进制数和十六进制数。

(1) 101011.101B　　(2) 110110.1101B

(3) 1001.11001B　　(4) 100111.0101B

1.5　将下列十六进制数分别转化为二进制数、八进制数、十进制数和压缩 BCD 数。

(1) 5A.26H　　(2) 143.B5H　　(3) 6CB.24H　　(4) E2F3.2CH

1.6　8 位和 16 位二进制数的原码、补码和反码可表示的数的范围分别是多少?

1.7　写出下列十进制数的原码、反码、补码表示（采用八位二进制，最高位为符号位）。

(1) 120　　(2) 62　　(3) −26　　(4) −127

1.8　已知补码求出其真值。

(1) 46H　　(2) 9EH　　(3) B6H　　(4) 6C20H

1.9　已知某个 8 位的机器数 65H，在其作为无符号数、补码带符号数、BCD 码以及 ASCII 码时，分别表示什么真值和含义?

1.10　给字符 8 和 6 的 ASCII 码加奇校验，应是多少? 若加偶校验呢?

第2章 微 处 理 器

2.1 微处理器概述

通常把微型计算机的中央处理单元称为微处理器（Microprocessor），是计算机内部对数据进行处理并对处理过程进行控制的部件。微处理器是影响和决定微型计算机系统性能的关键部件，随着半导体技术的不断发展，微处理器的性能在不断提高。

2.1.1 微处理器的基本概念与组成

微处理器是一个大规模集成电路器件，又称为中央处理单元（Center Processor Unit，CPU）。一般都具有下列基本功能：

1）能够进行算术和逻辑运算；

2）能够对指令进行译码，并执行指令所规定的操作；

3）能够提供整个系统所需要的定时和控制信号；

4）具有对存储器和I/O端口发送或接收数据的能力；

5）可响应其他部件发出的中断请求；

6）可暂存少量的数据。

人们通常所说的8位机、16位机、32位机是指微处理器内部可以同时操作的二进制数据的位数（内部数据总线的宽度）。目前常用的CPU都是32位的，即一次可操作32位二进制数。一般微处理器从它的功能结构上看，内部由运算器、控制器、寄存器组和内部总线组成。

1. 运算器

运算器的核心功能是实现数据的算术运算和逻辑运算，又称为算术逻辑运算单元（ALU）。运算器内部具有负责加、减、乘、除运算的加法器，以及实现与、或、非、异或、移位等逻辑运算的单元。新型的微处理器的运算器还能够完成浮点数运算。

运算器中除了核心部件ALU外，还有提供操作数和暂存中间运算结果的寄存器，以及存放特征位的标志寄存器和用于传送数据和指令的内部总线等部件。

2. 控制器

控制器的作用是控制程序的执行，是整个系统的指挥中心。控制器内部一般由指令寄存器、指令译码器和定时与控制逻辑电路组成。指令寄存器用于存放从存储器中取出的待执行的指令；指令译码器是对指令寄存器中的指令进行译码，确定该指令要什么样的操作。指令译码后通过定时与控制逻辑产生所需要的各种时序信号，并完成该指令规定的操作。总的来说，控制器的工作过程就是从存储器中取指令、指令译码和执行指令，以及给出操作数地址、取操作数、完成指令规定的操作，并将运算结果传送到存储器或I/O端口。它还向微机的其他各部件发出相应的控制信号，使CPU内、外各部件间协调工作。

除此之外，控制器中还有一个专用寄存器，即程序计数器PC（Program Counter）。程序计数器PC用来存放下一条将要执行的指令在存储器中的地址，控制着程序的执行顺序。在程序执行之前，应将程序的首地址（程序中第一条指令的地址）置入程序计数器。在顺序

执行指令时，每取出指令的一个字节，PC 的内容自动加 1；当程序发生转移时，通过转移指令把新的指令地址（目标地址）装入 PC，从而实现程序的转移。

3. 寄存器组

寄存器组相当于微处理器内部的随机读写存储器 RAM，分为通用寄存器组和专用寄存器组两部分。通用寄存器组用来存放参加运算的数据、中间结果或地址。微处理器内部有了这些寄存器之后，就可避免频繁地访问存储器，以缩短指令长度和指令执行时间，提高微处理器的运行速度。专用寄存器包括指令指针 IP（或称为程序计数器 PC）、堆栈指示器 SP 和标志寄存器 FR，它们的作用是固定的。其中：

（1）堆栈指示器 SP：用来存放堆栈区栈顶单元的地址，具有自动加 1 或减 1 的功能。堆栈是存储器中用于暂时保存数据的一个特定区域。它按“后进先出”方式工作，当数据压入堆栈时，堆栈指针减 1 指向栈顶单元，然后将数据压入栈顶单元；对于弹出操作，弹出的是当前栈顶单元的数据，然后堆栈指针加 1 指向新的栈顶单元。也就是说，数据在进行压栈、出栈操作时，总是在栈顶进行。堆栈一旦初始化（即确定了栈底在内存中的位置）后，SP 的内容（即栈顶位置）便由 CPU 自动管理。

（2）标志寄存器 FR 也称程序状态字（PSW）寄存器，用来存放算术和逻辑运算指令执行结果的特征，如结果为零、产生进位或溢出标志等。

4. 内部总线

内部总线是微处理器内部各个部件之间传送数据和控制信号的通路。

2.1.2 微处理器的主要技术参数

评价微处理器的主要技术参数通常指系统时钟频率、字长、高速缓冲存储器（Cache）以及指令集的大小等参数。

1. 时钟频率

CPU 的时钟频率又称为主频，也就是 CPU 运算时的工作频率，单位是 MHz。主频决定着微处理器内部数据传输和指令执行的速度或频率，决定计算机在一定时间内能够执行的指令数。

2. 字长

字长是 CPU 在单位时间内（同一时间）能一次处理的二进制数的位数。通常字长决定了 CPU 的寄存器和总线的数据宽度。微处理器按字长可以分为 8 位微处理器、16 位微处理器、32 位微处理器以及 64 位微处理器等。

3. 最大可直接寻址的内存

CPU 地址总线宽度决定了其可以直接寻址的物理地址空间，简单地说就是 CPU 到底能够使用多大容量的内存。具有 16 位地址总线宽度的 CPU，直接寻址的内存空间为 2^{16}（64KB）；具有 20 位地址总线宽度的 CPU，直接寻址的内存空间为 2^{20}（1MB）；而对于 386 以上的微机系统，地址总线的宽度为 32 位，最多可以直接访问 2^{32}（4096MB，即 4GB）的物理空间。

4. 数据总线宽度

数据总线宽度则决定了 CPU 与二级高速缓存、内存以及输入/输出端口之间进行一次数据传输的数据位数。

5. 高速缓冲存储器容量和频率

高速缓冲存储器又称为缓存（Cache），分为 L1 Cache（一级缓存）和 L2 Cache（二级缓存）。CPU 工作时，首先到缓存中读取所需要的数据；如果没有，CPU 再到内存中读取。而 CPU 内置的一级缓存频率极高，一般和微处理器同频运作，工作效率远远大于系统内存和硬盘。当缓存容量较大时，可以大幅度提升 CPU 读取数据的命中率，以此提高系统性能。

L1 Cache（一级缓存）是 CPU 第一层高速缓存，分为数据缓存和指令缓存。内置的 L1 高速缓存的容量和结构对 CPU 的性能影响较大，不过，在 CPU 芯片面积不能太大的情况下，L1 级高速缓存的容量不可能做得太大。一般服务器 CPU 的 L1 缓存的容量通常为 32～256KB。

L2 Cache（二级缓存）是 CPU 的第二层高速缓存，分为 CPU 内部和 CPU 外部两种芯片。CPU 内部的二级缓存运行速度与主频相同，而 CPU 外部的二级缓存则只有主频的一半。L2 高速缓存容量也会影响 CPU 的性能，一般 CPU 使用的 L2 容量是 512KB，而高档 CPU 的 L2 高速缓存可高达 256～1MB，甚至高达 2MB 或者 3MB。

6. 流水线和并行处理

我们知道，计算机执行程序时，CPU 总是有规则地重复下列步骤：

（1）从存储器中取出一条指令；

（2）分析指令操作码（即指令译码）；

（3）如果需要的话，从存储器中读取操作数；

（4）执行该指令；

（5）如果需要的话，将结果写入存储器或其他部件。

非流水线结构的 CPU 通常是以串行交替的工作方式来完成上述操作的，在 CPU 内部，由不同的电路来完成，当 CPU 以串行工作方式执行上述操作时，这些电路将交替工作，从而造成部分电路处于空闲状态。当总线处于间歇的空闲状态，导致指令的执行时间较长。

流水线与非流水线操作的比较如图 2.1 所示。

流水线技术是通过增加计算机硬件来实现的。在 CPU 内部由 5～6 个不同功能的单元电路组成一条指令处理流水线，然后将一条 X86 指令分成 5～6 步后再由这些单元电路分别执行，这样就能实现在一个 CPU 时钟周期完成一条指令，因此提高 CPU 的运算速度。经典奔腾每条整数流水线都分为四级流水，即指令预取、译码、执行、写回结果，浮点流水又分为八级流水。

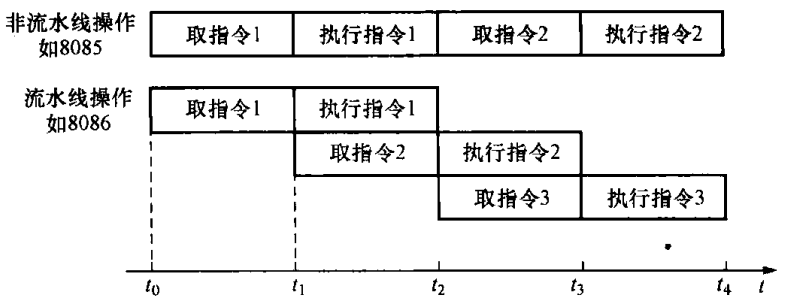


图 2.1 流水线与非流水线操作的比较

市场上推出的各种不同的 16/32 位微处理器基本上都采用了流水线技术。如 80486 和 Pentium 处理器均使用了流水线结构。Pentium、Pentium Pro 和 Pentium Ⅱ处理器的超标量设计更是分别结合了两条和三条独立的指令流水线，每条流水线平均在一个时钟周期内执行一条指令，所以它们平均一个时钟周期分别可执行 2 条和 3 条指令。

7. 微处理器芯片集成度

芯片集成度是指CPU集成内部晶体管数的能力，这个能力越大表明这种CPU性能就越好。随着CPU技术的不断发展，IC设计技术也越来越先进。目前的CPU晶体管数目都有几千万，Athlon XP达到了5000万之多。

2.1.3 80X86微处理器

80X86微处理器系列是指英特尔公司从20世纪70年代开始所研制出的微处理器的总称，即以前的80286、80386、80486等，到Intel开发出第五代产品586时，商标注册改为Pentium，同时为其注册中文商标名为“奔腾”。由于Intel的CPU在产量、技术上仍然占主要地位，所以其他厂家生产的CPU与Intel的CPU之间软硬件兼容。软件兼容是指非Intel生产的CPU对应用软件和X86扩展指令的软件兼容；硬件兼容是指非Intel的产品在机械安装和电性能两方面的兼容。

1. 从8080/8085到8086/8088

1978年英特尔公司推出了16位微处理器，命名为8086。8086微处理器在技术上与其前一代8位微处理器8080/8085相比，在技术上有了较大提高：8086微处理器有16位数据总线，一次可传送16位二进制数；8086的地址总线为20位，寻址空间提高到1MB；8086采用了流水线技术，在一个具有流水线结构的处理器中，当处理器执行内部操作时，其地址总线和数据总线可用来与存储器或输入输出端口之间进行数据传送。8086具有一个初级的流水线结构，可以实现片内操作与片外操作的重叠，从而提高了微处理器的性能。

英特尔公司在8086之后很快推出了8088微处理器，8088微处理器的内部结构与8086基本相同，是16位微处理器结构，但对外的数据总线设计为8位，称为准16位机。这样的设计使8088可以很容易与已有的8位外围芯片组成系统，既提供了16位处理器能力，又适应了已被广泛使用的8位外围芯片。

2. 80286/80386/80486

英特尔公司在1982年推出了80286微处理器，它仍然是16位结构。它的内部及外部数据都是16位的，但是80286具有24位地址线，可寻址16MB(2^{24}B)的存储器空间。80286有两种工作方式，即实模式和保护模式。实模式与8086工作模式相同，但速度比8086快。保护模式除了仍然具有16MB的存储器物理地址空间外，它还能为每个任务提供1GB(2^{30}B)的虚拟存储器地址空间。保护方式把操作系统及各任务所分配到的地址空间隔离开，避免程序之间的相互干扰，保证系统在多任务环境下正常工作。

在1985年英特尔公司研制出功能更强的80386微处理器。80386是一个32位结构，内部及外部数据均为32位，它的地址线为32位，可寻址4GB(2^{32}B)的物理存储器空间。80386为每个任务提供的虚拟存储器地址空间增加到64TB(2^{64}B)。

在1989年英特尔公司又研制出新一代微处理器80486。80486在结构上比80386有了很大改进：片内除了有一个与80386相同结构的主处理器外，还集成了一个浮点处理部件EPU以及一个8KB的高速缓冲存储器（Cache）。80486内部操作经改进设计后比80386速度快得多，总体性能比80386有明显提高。

3. Pentium微处理器

自英特尔公司推出80486微处理器后，又推出了新一代产品，按照习惯人们预测其处理

器命名为 80586。但英特尔公司在 1993 年正式命名新推出的微处理器为 Pentium，中文名字为“奔腾”，这是为申请版权而注册的商标。有时，人们也习惯地称 Pentium 为 80586。

Pentium 微处理器采用了超标量结构，提高了微处理器执行指令的并行性。超标量是 Pentium 微处理器的一个重要特点，其超标量结构是指内部实现了两条整数指令执行流水线硬件。每条流水线的结构都与英特尔公司的前一代 32 位微处理器 80486 的整数流水线结构兼容。

Pentium 微处理器采用 273 条引线的 PGA（网格阵列式引线）封装，引脚数量比 80486（168 条）增加许多。Pentium 微处理器对外有 64 条数据总线，使得微处理器与存储器之间的信息交换速率提高一倍，并可以充分保证微处理器的超标量执行部件，快速得到所需的指令代码及快速进行数据访问，这一点正是 Pentium 微处理器为提高性能在结构上所做的改进。

Pentium 微处理器的地址总线仍然设计成 32 位。Pentium 微处理器内部为 32 位寄存器，是 32 位微处理器。

2.2 8086/8088 微处理器

本节将对 8086/8088 微处理器的内部结构、引脚信号、存储器组织、系统组成、总线操作和时序等进行介绍。

2.2.1 8086/8088CPU 概述

8086/8088 微处理器是同时代微处理器中具有代表性的高性能 16 位微处理器，后续推出的各种微处理器均保持与其兼容。

8086 为 16 位微处理器，数据总线宽度为 16 位，地址总线宽度为 20 位，最大可直接寻址的内存空间为 1MB，时钟频率为 5～10MHz。8088CPU 的内部数据总线宽度是 16 位，外部数据总线宽度是 8 位，所以 8088CPU 称为准 16 位微处理器。8088CPU 的内部结构及外部引脚与 8086CPU 大部分相同。8086/8088 CPU 为了缩短指令执行周期，提高总线传输率，内部结构由两个独立的工作单元组成，即总线接口单元 BIU(Bus Interface Unit) 和指令执行单元 EU(Execution Unit)，将取指令、指令译码及执行该指令等步骤由该两个独立单元分别处理。

2.2.2 8086 CPU 的内部结构

8086 CPU 的内部结构如图 2.2 所示。

1. 总线接口单元 BIU(Bus Interface Unit)

总线接口单元 BIU 是 8086 CPU 与存储器和 I/O 端口之间传送信息的接口。它提供了 16 位双向的数据总线和 20 位地址总线，完成所有外部总线操作。

BIU 具有下列功能：地址形成、取指令、指令排队、读/写操作数和总线控制。BIU 由 16 位段寄存器、16 位指令指针寄存器 IP、20 位物理地址加法器、一个 6 字节的指令队列及总线控制电路组成。

(1) 16 位段寄存器

CS(Code Segment)：代码段寄存器，存放当前代码段的段基地址。

DS(Data Segment)：数据段寄存器，存放当前数据段的段基地址。

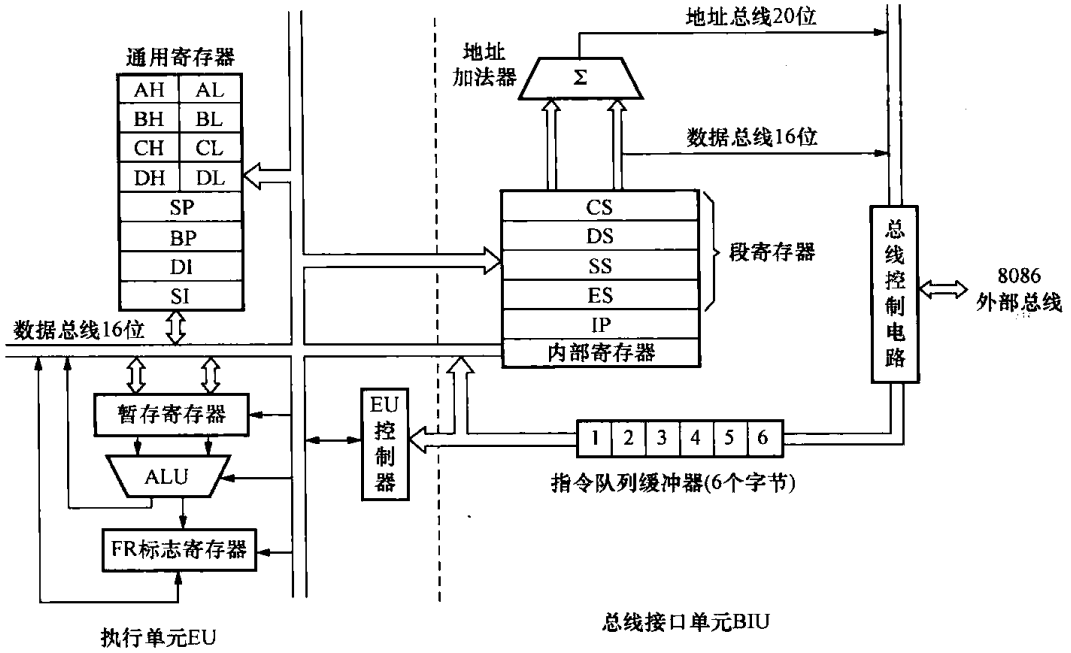


图 2.2 8086 CPU 的内部结构

ES(Extra Data Segment)：附加数据段寄存器，存放附加数据段的段基地址。

SS(Stack Segment)：堆栈段寄存器，存放当前堆栈段的段基地址。

8086 CPU 外部引脚中有 20 位地址总线，可直接寻址 1MB 的内存空间。而 CPU 内部是 16 位寄存器，无法装载 20 位的物理地址。为解决这个问题，8086/8088 CPU 中采用了分段技术，即将 1MB 的存储空间划分成若干逻辑段，每个逻辑段最大为 64KB，逻辑段可以在整个 1MB 存储空间内浮动，每个段的起始地址的高 16 位（称为段基地址或段基址）由段寄存器存放。

在 8086/8088 系统中，将存储器划分为代码段、数据段、附加数据段和堆栈段四种类型的逻辑段，并通过段寄存器 CS、DS、ES 和 SS 进行寻址。

代码段是一个用于保存程序指令代码的存储区域，数据段是一个存储原始数据以及运算结果的存储区域，附加数据段用于串操作指令时作为另一个数据区使用，堆栈段是内存中按照先进后出原则组织的、暂时保存数据的区域。由此看出，段寄存器指明一个特定的现行段，且用来存放各段的段基址。当用户用指令设定了它们的初值后，实际上就确定了一个 64KB 的存储区段。

(2) 16 位指令指针寄存器 IP

16 位的指令指针寄存器 IP 用来存放下一条将要执行指令的偏移地址（也叫有效地址 EA）。IP 由 BIU 按照 IP←IP+1 自动修改，当 EU 执行转移指令、调用指令时，BIU 将目标地址装入 IP。

(3) 20 位地址加法器

20 位的地址加法器用来将 16 位的逻辑地址变换成存储器读/写所需要的 20 位物理地址，实际上完成地址加法操作。比如，一条指令的逻辑地址由代码段寄存器 CS 和指令指针寄存器 IP 构成，通过 20 位地址加法器，将 CS 的内容左移 4 位与指令指针寄存器 IP 的内容相加，使 CPU 输出 20 位的物理地址。

（4）6字节指令队列

6字节指令队列是“先进先出”的RAM存储器，预存6字节的指令代码。

（5）总线控制电路

总线控制电路发出总线控制信号。例如：发出对内存和I/O端口的读/写控制信号等。它将8086CPU内部总线与外部总线相连，是CPU与外部交换信息的必经之路。

BIU总线接口单元的工作过程为：

BIU取指令过程：BIU将当前代码段寄存器CS中的16位段基地址最低位后面填写4个0（即将CS中的数据左移4位），与指令指针寄存器IP中16位偏移地址在地址加法器中形成20位物理地址并直接送往地址总线，然后通过总线控制逻辑电路发出存储器读信号$\overline{RD}$按给定地址从存储器中取出指令，送入指令队列中等待EU执行。一般情况下，BIU总是保证指令队列是满的，使得EU可源源不断地从指令队列中得到待执行的指令。一旦指令队列中空出2个字节，BIU将自动执行取指令操作以填满指令队列。

BIU取操作数过程：当EU从指令队列中取走指令，经指令译码后执行该指令。在执行该指令的过程中，若需要向存储器或I/O端口读/写操作数，只要EU向BIU提出申请并给出操作数的有效地址，BIU就通过地址加法器将现行数据段的段基址及EU给出的有效地址组成20位的物理地址，在当前取指令总线周期完成后，在读/写总线周期访问存储器或I/O端口，完成读/写操作。最后EU执行指令，由BIU给出结果。当EU执行转移指令时，则BIU清除指令队列中的指令，按照转移指令给出的新地址从存储器中取出指令，立即送给EU执行，然后从后续指令序列中取指令并填满指令队列。

BIU的空闲状态：当指令队列已满，而指令执行单元EU对总线接口单元BIU又没有提出总线请求，则此时总线接口单元BIU便进入空闲状态。

2. 指令执行单元EU(Execution Unit)

指令执行单元EU完成指令译码和执行指令的工作。它由以下几个部分组成：

（1）算术逻辑运算单元（ALU）。算术逻辑运算单元ALU完成8位或16位的二进制算术运算和逻辑运算，运算结果可通过片内总线送到通用寄存器或标志寄存器或经BIU写入存储器。

（2）FR标志寄存器。FR标志寄存器也叫PSW程序状态字寄存器，简称状态寄存器，用来存放ALU运算后的结果特征或机器运行状态，标志寄存器长16位，实际使用了9位。

（3）暂存寄存器。16位的暂存寄存器可暂时存放参加运算的操作数，是不可编程的寄存器。

（4）寄存器组。寄存器组中包含4个通用16位寄存器AX、BX、CX、DX，其中AX又称为累加器，也称为通用数据寄存器，用来存放操作数或地址。4个专用16位寄存器DI、SI、SP和BP，其中每个寄存器分别有各自的专门用途，SI称为源变址寄存器；DI称为目的变址寄存器；SP称为堆栈指示器，即堆栈指针，存放堆栈段栈顶单元的偏移地址；BP为对堆栈操作的基址指示器，BP中存放的是堆栈段中某一存储单元的偏移地址。

（5）EU控制电路。EU控制电路是取指令控制和时序控制部件。接收从BIU指令队列中取出的指令代码，经过分析、译码后形成各种实时控制信号，对各个部件进行操作。

指令执行单元EU工作过程为：

指令执行单元 EU 并不直接与外部发生联系，它从总线接口单元 BIU 的指令队列中不断地获取指令并执行之，省去了访问存储器取指令的时间，提高了 CPU 的利用率和整个系统的运行速度。如果在指令执行过程中需要访问存储器或需要从 I/O 端口取操作数，则 EU 向 BIU 发出操作请求，并将访问地址（有效地址 EA）送给 BIU，由 BIU 从外部取回操作数送给 EU。当遇到转移指令、调用指令和返回指令时，EU 要等待 BIU 将指令队列中预取的指令清除，并按转移的目标地址从存储器取出指令送入指令队列后，EU 才能继续执行指令。这时 EU 和 BIU 的并行操作显然要受到一定的影响，这是采用并行操作方式不可避免的。但只要转移指令、调用指令出现的概率不是很高，EU 和 BIU 间既相互配合又相互独立的工作方式仍将大大提高 CPU 的工作效率。

2.2.3 8086 CPU 的内部寄存器

8086 CPU 的内部寄存器用来存放参加运算的操作数地址、操作数及中间结果。其按功能分为 8 个通用数据寄存器（8 个）、段寄存器（4 个）和控制寄存器（2 个）三大类，如图 2.3 所示。

1. 通用寄存器

通用寄存器既可以作为算术、逻辑运算的源操作数，向 ALU 提供参加运算的原始数据，也可以作为目标操作数，保存运算的中间结果或最后结果。通用寄存器包括数据寄存器、地址指针寄存器和变址寄存器。

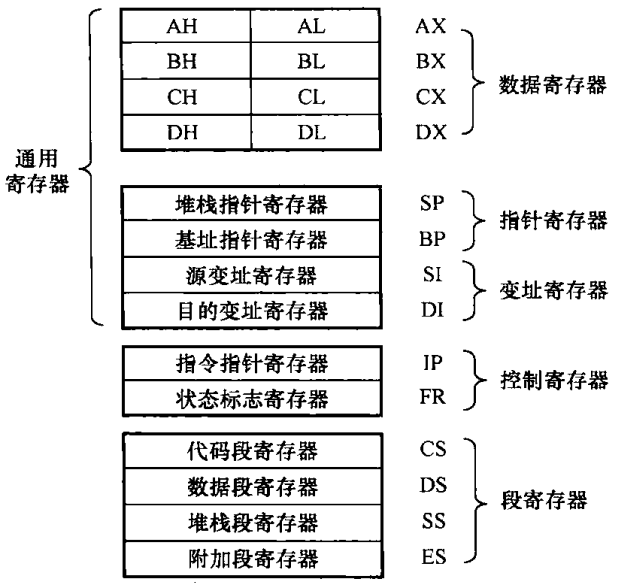


图 2.3 8086CPU 内部寄存器

（1）数据寄存器 AX，BX，CX，DX。数据寄存器一般用于存放参加运算的数据或运算的结果。它们既可以作为 16 位寄存器使用，也可以将高、低 8 位分别作为两个独立的 8 位寄存器使用。它们的高 8 位寄存器用 AH，BH，CH，DH 表示，低 8 位寄存器用 AL，BL，CL，DL 表示。这些寄存器的双重性使得 8088CPU 可以处理字数据也可以处理字节数据，使 16 位 CPU 与 8 位 CPU 具有较好的兼容性。

除此之外，数据寄存器除了作为通用寄存器使用外，它们还有各自的习惯用法。例如，AX 称为累加器，常用于存放算术逻辑运算中的操作数，另外所有的 I/O 指令都使用累加器与外设接口传送信息；BX 称为基址寄存器，常用来存放访问内存时的偏移地址；CX 称为计数寄存器，在循环和串操作指令中用作计数器；DX 称为数据寄存器，在寄存器间接寻址的 I/O 指令中存放 I/O 端口的地址。表 2.1 给出了 8086CPU 部分寄存器在某些指令中的特殊用途。

表 2.1 中，隐含是指在指令中不写出该寄存器名称。例如，乘法指令 MUL BL；AX←AL×BL，AL、AX 在乘法指令中没有写出，但它们被用来存放被乘数和乘积，采用了隐含方式。

表 2.1 8086 寄存器的特殊用途和隐含性质

寄存器名称	特 殊 用 途	隐含性质
AX，AL	在输入、输出指令中作数据寄存器 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	不能隐含 隐含
AH	在 LAHF 指令中，作目标寄存器	隐含
AL	在十进制调整指令中作寄存器	隐含
BX	在间接寻址中作基址寄存器 在 XLAT 指令中作基址寄存器	不能隐含 隐含
CX	在串操作指令和 LOOP 指令中作计数器	隐含
CL	在移位/循环移位指令中作为移位次数寄存器	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 在间接寻址的输入输出指令中作地址寄存器	隐含 不能隐含
SI	在字符串运算指令中作源变址寄存器 在间接寻址中作变址寄存器	隐含 不能隐含
DI	在字符串运算指令中作目的变址寄存器 在间接寻址中作变址寄存器	隐含 不能隐含
BP	在间接寻址中作基址指针	不能隐含
SP	在堆栈操作中作堆栈指针	隐含

(2) 指针寄存器 SP、BP。

SP(Stack Pointer)：堆栈指针寄存器。它在堆栈操作中存放堆栈段栈顶单元的偏移地址，即指向堆栈的栈顶单元。堆栈操作指令 PUSH 和 POP 就是从 SP 中得到段内偏移地址的。

BP(Base Pointer)：堆栈基址寄存器。BP 中存放的是堆栈中某一存储单元的偏移地址，而不一定是栈顶单元的偏移地址。

堆栈段是在内存中按照“先进后出”原则组织的内存区域，用于临时保存数据。CPU 对堆栈内存区可以进行压入（PUSH）和弹出（POP）两种操作。使用堆栈段寄存器 SS 来存放堆栈段的段基址，使用 SP 和 BP 分别存放栈顶单元和堆栈段中任意单元的偏移地址。BP 也可以作为一般数据寄存器使用，而 SP 只能作堆栈指针寄存器。

(3) 变址寄存器 SI、DI。

SI(source index)：源变址寄存器。在串操作指令中存放源操作数的偏移地址。

DI(destination index)：目的变址寄存器。在串操作指令中存放目的操作数的偏移地址。

对于串操作指令，规定源操作数必须位于当前数据段中，其源操作数所在单元的的段基地址用 DS 存放，偏移地址用 SI 存放；规定目的操作数必须位于附加数据段中，其目的操作数所在单元的段寄存器为 ES 存放，偏移地址用 DI 存放，这是一种约定。

变址寄存器 SI、DI 也可以作为一般数据寄存器使用，用来存放操作数或运算结果，它们对应的段寄存器均为 DS。

2. *段寄存器*

代码段寄存器 CS、数据段寄存器 DS、附加数据段寄存器 ES 和堆栈段寄存器 SS，分别

用来存放相应段的段基地址，即内存段起始地址的高 16 位。

3. 控制寄存器

IP(Instruction Pointer) 称为指令指针寄存器，用以存放将要执行指令的偏移地址。CPU 取指令时总是以 CS 为段基地址，以 IP 为段内偏移地址。当 CPU 从 CS 段中偏移地址为 IP 的内存单元中取出指令代码的一个字节后，IP 自动加 1，指向指令代码的下一个字节。需要说明的是，IP 不能作为指令操作数，但有些指令的执行可以修改它的内容。例如，转移指令、子程序调用指令和返回指令，使 IP 的值改变。

FR(Flag Register) 称为标志寄存器，也称为程序状态字 PSW。FR 为 16 位寄存器，实际仅用了 9 位，这 9 位包括 6 个状态标志位和 3 个控制标志位。标志寄存器格式如图 2.4 所示。

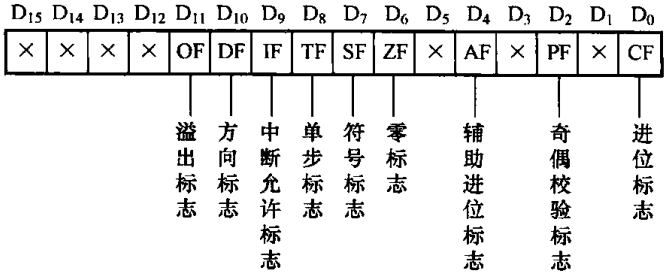


图 2.4 标志寄存器格式

状态标志位记录了算术和逻辑运算结果的一些特征，是指令执行后自动建立的，这些特征可以作为一种先决条件来决定下一步的操作。它们具体的含义是：

(1) CF(Carry Flag)：进位/借位标志位。当本次算术运算结果使最高位产生进位（加法运算）或借位（减法运算）时，则此标志位置“1”，即 CF=1；若加法运算结果最高位无进位，或减法运算结果最高位无借位，则 CF=0。此外，循环移位指令执行过程也会影响这一标志。

(2) PF(Parity Flag)：奇偶标志位。表示本次运算结果的低 8 位中包含“1”的奇偶性，含“1”的个数为偶数时，PF=1；为奇数时，PF=0。

(3) AF(Auxiliary Carry Flag)：辅助进位/借位标志位。表示加法或减法运算结果中第 3 位向第 4 位产生进位或借位的情况。AF=1 表示有进位，AF=0 表示无进位或借位。AF 标志在 BCD 码的十进制算术指令中作为是否要进行十进制调整的依据。

(4) ZF(Zero Flag)：零标志位。表示当前的运算结果是否为零。ZF=1 表示运算结果为零，ZF=0 表示运算结果不为零。

(5) SF(Sign Flag)：符号标志位。表示本次运算结果的正、负情况。SF=1 表示运算结果为负，SF=0 表示运算结果为正。

(6) OF(Overflow Flag)：溢出标志位。表示运算结果产生溢出情况。OF=1 表示当前正在进行的补码运算有溢出，OF=0 表示无溢出。

所谓溢出，指带符号数进行字节运算，其结果超出－128～＋127 范围，或字运算的结果超出－32768～＋32767 范围。实际上，当溢出时其运算结果已超出目标单元所能表示的数值范围，将会丢失有效数字，而出现错误结果。

例：将十六进制数 5349H 和 465AH 相加，并说明其标志位状态：

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

两正数相加（补码加），结果为负，显然运算产生了溢出，即超出了计算机所能表示数的

范围，故 OF=1；由于运算结果的最高位为 1，所以 SF=1；运算结果本身不为 0，故 ZF=0；又由于运算结果的低 8 位中含 1 个数为偶数，故 PF=1；运算结果的最高位没有向前产生进位，故 CF=0；运算过程中第 3 位向第 4 位（即低 4 位向高 4 位）产生了进位，故 AF=1。

控制标志位通过指令设置，每一种控制标志被设置后都对 CPU 其后的操作产生控制作用。它们具体的含义是：

(1) IF(Interrupt Enable Flag)：中断允许标志。用来控制可屏蔽中断的标志位。IF=1 时，开中断，CPU 可以接收可屏蔽中断请求；IF=0 时，关中断，CPU 不能接收可屏蔽中断请求。用 STI 指令可使 IF 标志位置“1”，CLI 指令可使 IF 标志位置“0”。

(2) DF(Direction Flag)：方向标志。用来设定和控制串操作指令的步进方向。DF=1 时，串操作过程中的地址会自动递减 1；DF=0 时，地址自动递增 1。用 STD 指令可使 DF 标志位置“1”，用 CLD 指令可使 DF 标志位置“0”。

(3) TF(Trap Flag)：单步标志。用来控制 CPU 进入单步工作方式。TF=1 时，8086 CPU 处于单步工作方式，每执行完一条指令就自动产生一次内部中断；TF=0 时，CPU 不能以单步方式工作。用户可利用 CPU 的单步工作方式来检查每条指令的执行情况。通过 DEBUG 调试程序使 TF=1，实现程序单步运行。

在 DEBUG 调试环境中，提供了测试标志位的方法，它用符号来表示标志位的值。表 2.2 说明了各标志在 DEBUG 中的符号表示（TF 在 DEBUG 中不提供符号）。

表 2.2　　DEBUG 调试环境标志位的符号表示

标志位名称	标志位为 1	标志位为 0	标志位名称	标志位为 1	标志位为 0
OF　溢出（有/无）	OV	NV	ZF　零（是/否）	ZR	NZ
DF　方向（减址/增址）	DN	UP	AF　辅助进位（有/无）	AC	ZA
IF　中断（允许/关闭）	EI	DI	PF　奇偶（偶/奇）	PE	PO
SF　符号（正/负）	NG	PL	CF　进位（有/无）	CY	NC

2.2.4　8086CPU 的工作模式和引脚信号

8086CPU 是采用 40 条引脚双列直插式封装的集成电路芯片，芯片外部通过引脚与存储器、I/O 端口等外部部件交换信息。其各引脚定义如图 2.5 所示。

为了适应广泛的应用环境，8086CPU 设置了最小工作模式和最大工作模式两种工作模式。所谓最小工作模式，就是在系统中只有一个 8086 微处理器，所有的总线控制信号都直接由 8086CPU 产生，因此，系统中的总线控制电路被减到最少。最大工作模式是相对最小工作模式而言的。在最大工作模式系统中，总是包含两个或多个微处理器，其中一个主处理器是 8086，其他的处理器称为协工作处理器，它们是协助主处理器工作的。如数学运算协处理器 8087，输入/输出协处理器 8089。8086CPU 到底工作在最大工作模式还是最小工作模式，完全由 8086 引脚 MN/$\overline{MX}$决定：当该引脚接+5V 时设置为最小工作模式，当该引脚接地时设置为最大工作模式。两种模式下 8086CPU 的第 24～31 号引脚具有不同的功能。

图 2.5 中括号内标注的为最大工作模式下的引脚功能。8086 和 8088 在两种工作模式下的特点如表 2.3 所示。

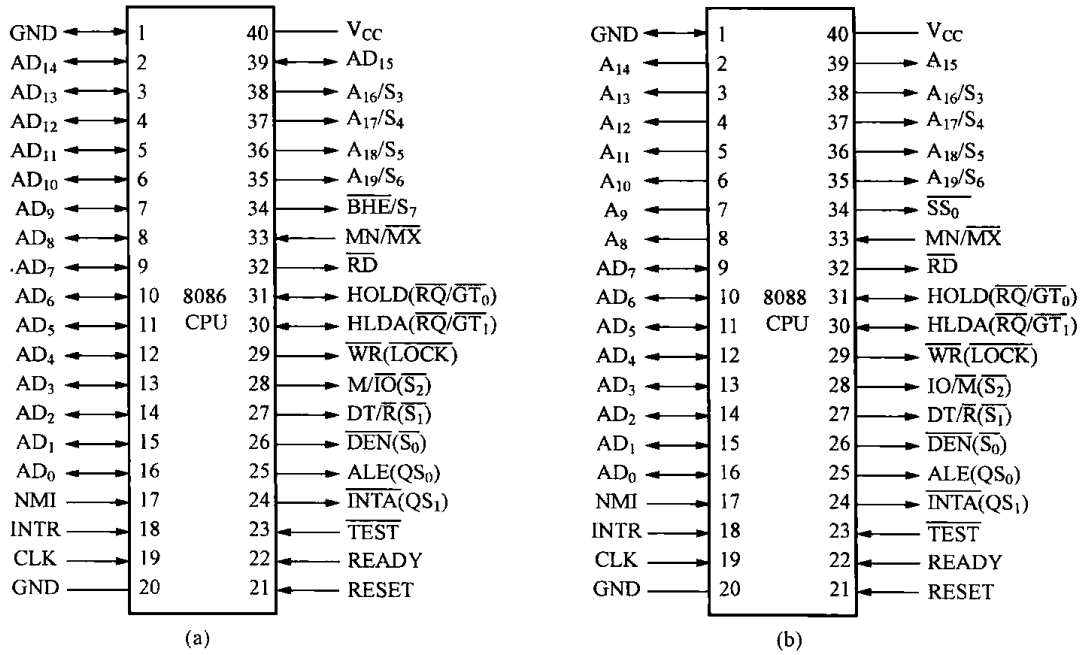


图 2.5 8086/8088CPU 引脚图

表 2.3 最小模式和最大模式的特点

最 小 模 式	最 大 模 式
MN/$\overline{MX}$接+5V 系统中只有 8086 一个处理器 所有的控制信号都是由 8086CPU 产生	MN/$\overline{MX}$接地 系统中可包含一个以上的处理器，构成多处理器系统，其中 8086 是主处理器，其他是协处理器 系统控制信号由总线控制器提供

首先来分析 8086/8088 引脚信号的特点：

（1）总线的三态性。总线的三态性是微处理器的共性，任何微处理器的地址总线、数据总线及部分控制总线均采用三态缓冲器式总线电路。所谓三态，是指它们的输出可以有逻辑“1”、逻辑“0”和“高阻”三种状态。当处于高阻状态时，总线电路呈现极高的输出阻抗，如同与外界“隔绝”一样。总线电路的这种三态性，一方面保证了在任何时刻，只能允许相互交换信息的设备占用总线，其他设备和总线隔绝，降低对总线的影响；另一方面为数据的快速传送方式（即直接存储器存取方式 DMA）提供了必要的条件，因为当进行 DMA 传送时，CPU 将与外部总线“断开”，外部设备将直接利用总线和存储器交换数据。

（2）地址/数据总线分时复用特性。由于微处理器外部引脚数量的限制，常采用总线分时复用技术。在 8086CPU 中，数据总线与地址总线低 16 位就是分时复用的，即 AD_{15}～AD_0 在 T_1 状态传输地址信息，在 T_2～T_4 状态传输数据信息；而且，A_{19}/S_6～A_{16}/S_3 也是地址总线高 4 位与状态信号线分时复用的。

由于 8088 只能传输 8 位数据，所以 8088 只有 8 根地址/数据总线复用。

下面介绍 8086 在两种模式下的引脚功能。

1. 8086CPU 最小模式下的引脚信号

（1）AD_{15}～AD_0（Address/Data Bus）地址/数据总线，双向，三态。这是一组采用分时的方法传送地址或数据的复用引脚。在总线周期的 T_1 状态，CPU 从这些引脚上输出访问存

储器或 I/O 端口的低 16 位地址，在 T_2～T_4 状态用于传送数据。在 CPU 响应中断及系统总线“保持响应”周期时，AD_{15}～AD_0 被置成高阻状态。

(2) A_{19}/S_6～A_{16}/S_3(Address/Status) 地址/状态线，分时复用、输出、三态。这是采用分时的方法传送地址或状态的复用引脚。其中，A_{19}～A_{16}为 20 位地址总线的高 4 位地址，S_6～S_3 是状态信号。A_{19}～A_{16}在总线周期 T_1 状态输出高 4 位地址，与 AD_{15}～AD_0 一起构成 20 位物理地址，可访问 1MB 的存储器空间；当 CPU 访问 I/O 端口时，A_{19}～A_{16}为“0”。在总线周期的 T_2、T_3、T_4 状态输出状态信息 S_6～S_3，其中 S_6 恒为“0”，表示 8086/8088 当前与总线相连。S_5 指示中断允许标志 IF 的当前状态，$S_5=1$，表示当前允许可屏蔽中断请求，$S_5=0$，则禁止一切可屏蔽中断。S_4 和 S_3 合起来指示当前使用段寄存器的情况，如表 2.4 所示。

表 2.4　　S_4、S_3 状态编码含义

S_4	S_3	当前正在使用的段寄存器
0	0	ES
0	1	SS
1	0	CS 或不需要使用段寄存器 (I/O 操作，中断响应)
1	1	DS

当系统总线处于“保持响应”状态时，A_{19}/S_6～A_{16}/S_3 被置为高阻状态。

(3) ALE(Address Latch Enable) 地址锁存允许信号。ALE 用作地址锁存器 8282/8283 的选通信号，输出高电平有效。在任何一个总线周期的 T_1 状态，ALE 输出有效信号，其下降沿将地址/数据复用总线上的地址信息锁入地址锁存器 8282/8283 中。ALE 信号不能处于高阻状态。

(4) $\overline{BHE}/S_7$(Bus High Enable/Status) 高 8 位数据总线允许/状态信号，输出、三态，低电平有效。在总线周期的 T_1 状态，$\overline{BHE}$有效，表明在高 8 位数据总线 D_{15}～D_8 上传送 1 个字节的数据。在 T_2、T_3、T_4 状态，$\overline{BHE}/S_7$ 输出状态信号 S_7，但在 8086 中 S_7 无意义，在总线保持响应周期被置成高阻状态。

(5) $\overline{DEN}$(Data Enable) 数据允许信号，输出、三态，低电平有效。在最小模式中，有时要利用总线收发器 8286/8287 (也称总线驱动器) 来增加总线的数据驱动能力，而$\overline{DEN}$用作总线收发器 8286/8287 的选通控制信号。$\overline{DEN}$有效表明 CPU 进行数据的读/写操作。在 DMA 工作方式时，$\overline{DEN}$被置为高阻状态。

(6) DT/$\overline{R}$(Data Transmit/Receive) 数据发送/接收控制信号。DT/$\overline{R}$ 信号用来控制数据传送的方向，三态、输出。DT/$\overline{R}$ 为高电平时，CPU 发送数据到存储器或 I/O 端口；DT/$\overline{R}$为低电平时，CPU 接收来自存储器或 I/O 端口的数据。在 DMA 方式时，DT/$\overline{R}$ 被置成高阻状态。

(7) M/$\overline{IO}$(Memory/Input Output) 存储器、I/O 端口选择控制信号。M/$\overline{IO}$信号指明当前 CPU 是访问存储器还是访问 I/O 端口，三态，输出。M/$\overline{IO}$为高电平时，访问存储器，表示当前要进行 CPU 与存储器之间的数据传送。M/$\overline{IO}$为低电平时，访问 I/O 端口，表示当前要进行 CPU 与 I/O 端口之间的数据传送。

对于 8088 CPU，该信号为 IO/$\overline{M}$，其定义正好与 8086 相反，目的是为了使 8088 CPU 与 8 位微处理器 8085 兼容。

(8) $\overline{RD}$(Read) 读信号，输出、三态、低电平有效。$\overline{RD}$信号有效时，表明 CPU 正在执行读总线周期，同时由 M/$\overline{IO}$信号决定是对存储器还是对 I/O 端口执行读操作。

(9) $\overline{WR}$(White) 写信号，输出、三态、低电平有效。$\overline{WR}$信号有效时，表明 CPU 正在执行写总线周期，同时由 M/$\overline{IO}$信号决定是对存储器还是对 I/O 端口执行写操作。

(10) READY(Ready) 准备就绪信号。READY 信号用来实现 CPU 与存储器或 I/O 端口之间的时序匹配，输入、高电平有效。当 READY 信号为高电平有效时，表示 CPU 要访问的存储器或 I/O 端口已经做好了输入/输出数据的准备工作，CPU 可以进行读/写操作。当 READY 信号为低电平时，则表示存储器或 I/O 端口还未准备就绪，CPU 需要插入若干个等待周期 T_W。

(11) RESET(Reset) 复位信号，输入、高电平有效。CPU 接收到复位信号后，立即结束现行操作，进入复位状态：CS 寄存器置为 FFFFH，其他所有的内部寄存器清零。当 RESET 信号由高电平变为低电平时，CPU 自动从 FFFF0H 地址开始重新执行程序。通常 FFFF0H 是 ROM BIOS 区中的一个单元，在此开始的单元中存放着一条无条件转移指令，使 CPU 转去执行引导和装配程序，实现对系统的初始化，并启动监控程序或操作系统。

(12) MN/$\overline{MX}$(Minimum/Maximum Mode Control) 最小/最大工作模式控制信号。MN/$\overline{MX}$引脚用来设置 8086 CPU 的工作模式，输入。当 MN/$\overline{MX}$为高电平（接+5V）时，CPU 工作在最小模式；当 MN/$\overline{MX}$为低电平（接地）时，CPU 工作在最大模式。

(13) INTR(Interrupt Request) 可屏蔽中断请求信号，输入，高电平有效。当 INTR 引脚为高电平，表明有 I/O 设备向 CPU 申请中断。8086 CPU 在每条指令执行到最后一个时钟周期时，都要检测 INTR 引脚信号。当 CPU 内部的中断允许标志 IF=1，则 CPU 在当前指令执行完后，进入中断响应周期。

(14) $\overline{INTA}$(Interrupt Acknowledge) 中断响应信号。$\overline{INTA}$信号是 CPU 对外部来的中断申请信号 INTR 的响应信号，输出、低电平有效。$\overline{INTA}$为低电平时，表示 CPU 已经响应外设的中断请求，即将执行中断服务程序。

对 8086 来说，中断响应周期要占用两个连续的总线周期，出现两个$\overline{INTA}$负脉冲。第一个$\overline{INTA}$负脉冲通知外设接口，它的中断申请已被响应，第二个$\overline{INTA}$负脉冲通知外设接口（中断控制器）向数据总线上发中断类型码。

(15) NMI(Non-Maskable Interrupt) 不可屏蔽中断请求信号，输入、正跳变上升沿有效。当 NMI 有效时，表明 CPU 内部或 I/O 设备提出了非屏蔽中断请求，CPU 会在结束当前所执行的指令后，立即响应中断请求。不可屏蔽中断请求不受中断允许标志 IF 的影响，也不能用软件来屏蔽（禁止），一旦从 NMI 引脚收到一个正跳变触发信号，CPU 执行完当前指令，便自动引起一个类型码为 2 的中断，并转入对应的不可屏蔽中断服务程序。

(16) $\overline{TEST}$(Test) 测试信号。$\overline{TEST}$信号用来支持构成多处理器系统，输入、低电平有效。实现 8086 CPU 与协处理器之间同步协调的功能，只有当 CPU 执行 WAIT 指令时才使用。

(17) HOLD(Hold Request) 总线保持请求信号。HOLD 信号也叫做“请求占用总线”请求信号，输入、高电平有效。在 DMA 数据传送方式中，由总线控制器 8237A 发出一个高电平有效的总线请求信号，通过 HOLD 引脚输入到 CPU，请求 CPU 让出总线控制权。

(18) HLDA(Hold Acknowledge) 总线保持响应信号。HLDA 是与 HOLD 配合使用的联络信号，输出、高电乎有效。在 HLDA 有效期间，HLDA 引脚输出一个高电平有效的响

应信号，CPU 让出对总线的控制权，同时总线将处于高阻状态。当 DMA 使用总线后，CPU 会使 HOLD 信号变为低电平，又重新获得对总线的控制权。

(19) CLK(Clock) 时钟信号。CLK 为 CPU 提供基本的定时脉冲信号，输入。8086 CPU 一般使用时钟发生器 8284A 来产生时钟信号，时钟频率为 5～8MHz，占空比为 1∶3。

(20) V_{CC}(+5V)、GND (地线)。8086/8088CPU 需要的电源 V_{CC}为+5V，GND 为地线。

2. 8086 最大模式下的引脚信号

当引脚 MN/$\overline{MX}$接低电平时，CPU 工作于最大模式，此时，引脚信号除 24～31，其他引脚与最小模式时相同。现分别说明如下：

(1) $\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$(Bus Cycle Status) 总线周期状态信号，输出、三态，低电平有效。由 CPU 产生的总线周期状态信号送给总线控制器 8288，表明当前总线周期所进行的操作类型。通过 8288 综合译码后产生相应的访问存储器或 I/O 端口的总线控制信号。$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$的信号组合与总线操作类型的对应关系如表 2.5 所示。

表 2.5　　$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$信号组合与总线操作类型的对应关系

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CPU 状态	$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CPU 状态
0	0	0	中断响应	1	0	0	取指令
0	0	1	读 I/O 端口	1	0	1	读存储器
0	1	0	写 I/O 端口	1	1	0	写存储器
0	1	1	暂停	1	1	1	无作用

(2) QS_1，QS_0(Instruction Queue Status) 指令队列状态信号，输出，高电平有效。QS_1 和 QS_0 信号的组合表示总线接口单元 BIU 中指令队列的状态，以便于外部对 CPU 内部指令队列的动作跟踪。QS_1、QS_0 的信号与指令队列状态见表 2.6。

表 2.6　　QS_1、QS_0 信号与指令队列状态

QS_1	QS_0	队列状态	QS_1	QS_0	队列状态
0	0	无操作	1	0	队列已空
0	1	从队列中取第一个字节	1	1	从队列中取后续字节

(3) $\overline{LOCK}$(Lock) 总线封锁信号，输出、低电平有效。该信号有效时，表示此时 8086 CPU 不允许其他总线部件占用总线。

(4) $\overline{RQ}/\overline{GT_1}$，$\overline{RQ}/\overline{GT_0}$(Request/Grant) 总线请求信号（输入）/总线响应（输出）引脚信号，双向，低电平有效。该信号是特意为多处理器系统而设计的。引脚$\overline{RQ}/\overline{GT_0}$的优先级高于$\overline{RQ}/\overline{GT_1}$。两个引脚的功能如下：

当其他的总线控制设备要使用系统总线时，会产生一个总线请求信号（一个时钟周期宽的负脉冲），并把它送到$\overline{RQ}/\overline{GT}$引脚，类似于最小工作模式下的 HOLD 信号。CPU 检测到总线请求信号后，在下一个 T_4 或 T_1 期间，在$\overline{RQ}/\overline{GT}$引脚送出总线响应信号（一个时钟周期宽的负脉冲）给请求总线的设备，它类似于最小工作模式下的 HLDA 信号。然后从下一个时钟周期开始，CPU 释放总线。总线请求设备使用完总线后，再产生一个$\overline{RQ}/\overline{GT}$信号。CPU 检索到该信号后，从下一个时钟周期开始重新控制总线。

2.2.5 8086的内存储器和I/O端口

1. 存储器的分段和物理地址的形成

(1) 地址空间。8086CPU有20条地址线，可直接寻址1MB的存储空间，地址范围用十六进制表示为00000H～FFFFFH。每一个存储单元存放一个字节（8位）的二进制信息。为了便于对存储器进行存取操作，每一个存储单元都有一个唯一的地址与之对应，其存储器按照地址顺序排列如图2.6所示。

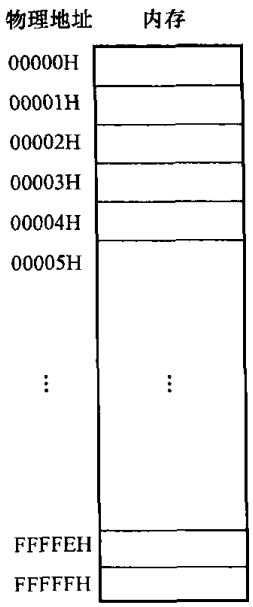


图2.6 存储器按照地址顺序排列示意图

每个存储单元存储一个字节数据，而在进行数据存取操作时，数据可以是字节、字、双字，甚至是多字，它们分别占用一个存储单元、两个存储单元、四个存储单元和多个存储单元。多个字节数据在内存中占连续的字节单元，按照低字节存放在低地址中，高字节存放在高地址中，并以低字节地址作为该数据的地址（称为低位在前，高位在后）。

例如，在20000H地址中存放一个双字数据76543210H，则表示该数据存放在20000H～20003H连续4个字节单元中，依次存放的数据是10H、32H、54H、76H。

需要说明的是，存放多字节数据时，如果其低位字节从偶数地址开始存放，称为“对准”存放，该数据称为“对准字”，而低位字节从奇数地址开始，这种方式为“非对准”存放，该数据称为“非对准”字；8086CPU存取对准字可在一个总线周期完成，而存取非对准字则需要两个总线周期才能完成。

例如，字数据1200H存放在30000H开始的单元，意味着（30000H）＝00H，(30001H)＝12H，该字数据的地址为30000H，字数据1200H为“对准”存放，并称1200H为“对准字”，如图2.7（a）所示，CPU读取该字时，使用一个总线周期。如果字数据存放在30001H开始的单元，则为非对准存放，如图2.7（b）所示，CPU读取该字时，使用2个总线周期。

需要说明的是：8086系统的堆栈以字为单位进行操作，并且堆栈中的数据项必须按“对准字”的形式存储，保证每访问一次堆栈就能压入/弹出一个字的数据。堆栈操作进栈指令PUSH和出栈指令POP在第3章中进行介绍。

(2) 存储器的分段。8086系统中，地址总线是20位的，可直接寻址空间为1MB(2^{20}＝1M）字节，而8086CPU内部寄存器是16位的，最多只能寻址64KB(2^{16}＝64K）空间。为了能达到对1MB的存储器寻址，8086系统中引入了存储空间分段技术，即将整个1MB的存储空间分成若干个可独立寻址的逻辑单位，称为逻辑段，每个逻辑段大小为64KB，并且每个段的起始地址是一个可以被16整除的数（即段的起始地址的最低四位为0）。逻辑段分为代码段、数据段、堆栈段和附加段四种类型。将这四种逻辑段起始地址的最高16位地址值（称为段基地址或段基址）分别存放在CS、DS、SS和ES

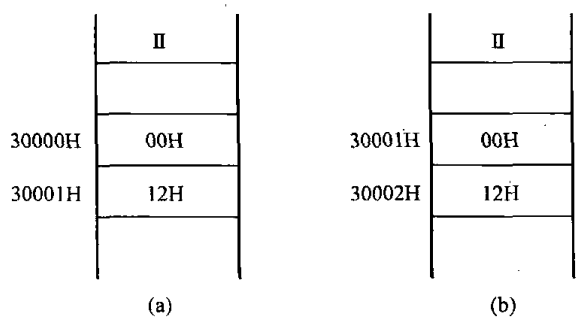


图2.7 对准字与非对准字

(a) 对准存放形式；(b) 非对准存放形式

四个段寄存器中。当程序中确定了段寄存器的段基址，就意味着在内存中确定了一个 64KB 大小逻辑段的具体位置，并从给出的段基址开始存取指令代码或数据。相对段基地址的偏移量称为偏移地址，指令代码的偏移地址存放在 IP 寄存器中，而数据的偏移地址存放在 BX、BP、SI、DI 寄存器中。

存储空间的分段方式多种多样，一个逻辑段可以在 1MB 的内存中浮动，段与段之间可以部分重叠、完全重叠或完全分离。存储器分段如图 2.8 所示。

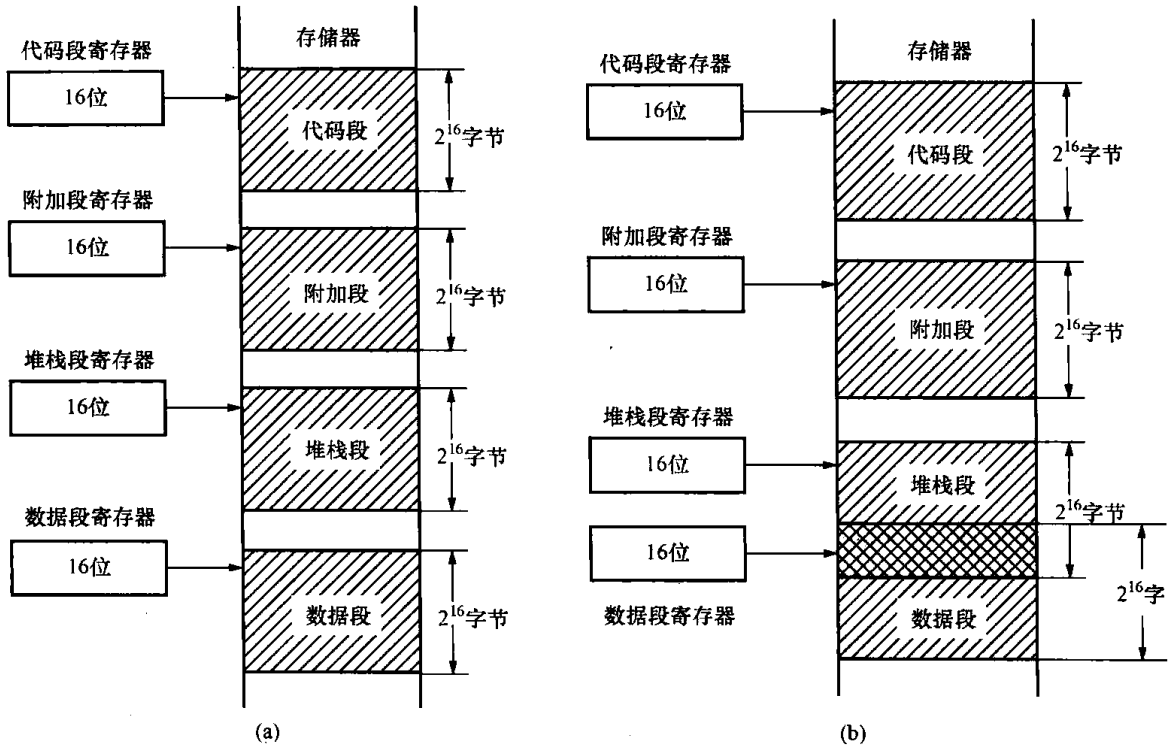


图 2.8 存储器分段示意图

(a) 段与段完全分离；(b) 段与段部分重叠

在图 2.8 中看到，1MB 的存储器除已经被定义的四个段外，还剩下一些空白（未用）区域，如果要用到这些区域，则必须首先改变相应段寄存器的内容，重新设置四个段寄存器，一旦加以定义，就可以通过段寄存器来访问不同的段。

(3) 逻辑地址和物理地址。存储器采用分段结构以后，对存储器的访问可以使用两种地址，即逻辑地址和物理地址。

物理地址也叫实际地址或绝对地址，是 CPU 访问存储器时实际使用的地址，由 20 位二进制数构成，地址总线上传送的就是这个地址。1MB 内存空间中每个存储单元的物理地址是唯一的。物理地址是 CPU 访问内存时使用的地址。

逻辑地址由段基址（存放在段寄存器中）和偏移地址（由寻址方式提供）两部分构成，它们都是无符号的 16 位二进制数。逻辑地址是用户进行程序设计时采用的地址。

当用户通过编制程序将 16 位逻辑地址送入 CPU 的总线接口单元 BIU 时，地址加法器通过地址运算变换为 20 位的物理地址。产生 20 位物理地址的公式为：

物理地址＝段基址×16＋偏移地址

其中，段基址×16 的操作是将 16 位段寄存器的内容（二进制形式）左移四位，末位补四个

0 来实现的。8086 存储器物理地址的形成过程如图 2.9 所示。

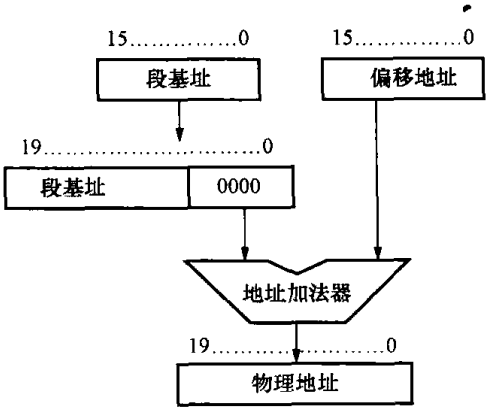


图 2.9 存储器物理地址的形成过程

[例 2.1] 若数据段寄存器 DS=2100H，试确定该数据区物理地址的范围。

解 首先需要确定该数据区段中第一个存储单元和最后一个存储单元的 16 位偏移地址。因为一个逻辑段的最大容量为 64KB，所以第一个存储单元的偏移地址为 0，最后一个存储单元的偏移地址为 FFFFH。该数据区段由低至高相应存储单元的偏移地址为 0000H～FFFFH。

数据区的首地址=DS×16+偏移地址=2100H×16+0000H=21000H

数据区的末地址=DS×16+偏移地址=2100H×16+FFFFH=30FFFH

从而可知：该数据段的地址范围是 21000H～30FFFH。

[例 2.2] ①已知 CS=6A00H，偏移地址=2200H 时，请计算物理地址；②已知 CS=5D00H，偏移地址=F200H 时，请计算物理地址。

解 根据物理地址的计算公式，可得：

题①的物理地址=CS×16+偏移地址=6A00H×16+2200H=6C200H

题②的物理地址=CS×16+偏移地址=5D00H×16+F200H=6C200H

从 [例 2.2] 可以看出：在题①和②中给定的段基址和偏移地址各不相同，而计算所得的物理地址却是一样的，均为 6C200H。这说明，对于存储器的任意存储单元来说，物理地址是唯一的，而逻辑地址却有无数个。不同的段基址和相应的偏移地址可以形成同一个物理地址。

通常存储单元可用逻辑地址或物理地址来表示，而逻辑地址常写成：“段基址：偏移地址”的形式。在程序设计中使用逻辑地址来表示内存单元的地址，并隐含段基址，CPU 寻址时根据指令中提供的偏移地址来自动匹配对应的段寄存器。例如，MOV AL，[2000H]，方括号中的 2000H 为偏移地址，默认的段寄存器为 DS。

80868CPU 对访问不同逻辑段所使用的段寄存器和相应的偏移地址有一些具体的约定：

(1) 对程序区访问：专门用于存放程序指令代码的存储区域称为程序区。段基址由代码段寄存器 CS 指定，IP 的内容表示段内的偏移地址。当前所取指令的物理地址为 CS×16+IP。若要访问不同的程序区时，只需修改代码段寄存器 CS 的内容即可。

(2) 对数据区的访问：数据信息包括 CPU 要处理的原始数据、运算的中间结果和最后结果。访问数据区时，DS 的内容用来表示数据段的段基址，而偏移地址由指令的寻址方式所求得的有效地址 EA(Effective Address) 来确定。其物理地址为 DS×16+EA，有关 EA 的计算在第 3 章中介绍。

(3) 对堆栈区的访问：堆栈是特殊的存储区域，用来存放由 PUSH 指令压入的需要进行保护的数据和状态信息。访问堆栈区时，用堆栈段寄存器 SS 指示堆栈段的段基址，SP 的内容表示栈顶的偏移地址。BP 的内容表示栈内单元的偏移地址。堆栈操作时，存储单元的物理地址为=SS×16+SP。

（4）串操作：在存储器中，串操作指的是对两个数据块进行传送或比较，这就需要指定传送的源数据区和目标数据区。通常用 DS 作为源数据区的段寄存器保存段基址，源变址寄存器 SI 的内容表示偏移地址，用 ES 作为目标数据区的段寄存器保存段基址，目标变址寄存器 DI 表示偏移地址。物理地址的计算公式为：源数据区物理地址＝DS×16＋SI；目地数据区物理地址＝ES×16＋DI。

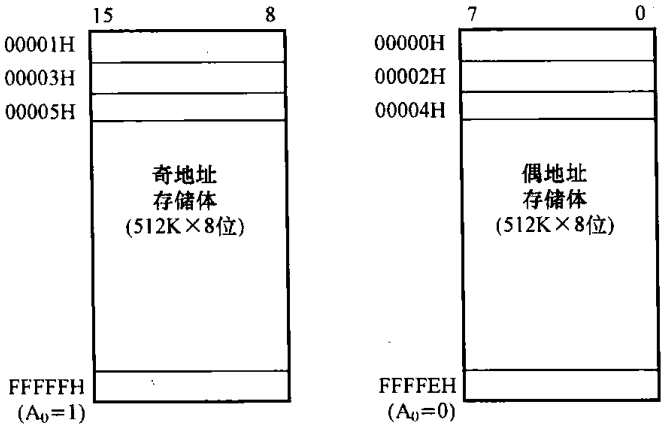


图 2.10　存储器分体结构

2. 8086 的内存储器结构

8086 系统中，为了实现 16 位数据的传送，存储器采用分体结构，将 1MB 的存储空间分成两个 512KB 的存储体，采用字节交叉编址方式，如图 2.10 所示。

8086 系统中存储器与总线的连接方式如图 2.11 所示。

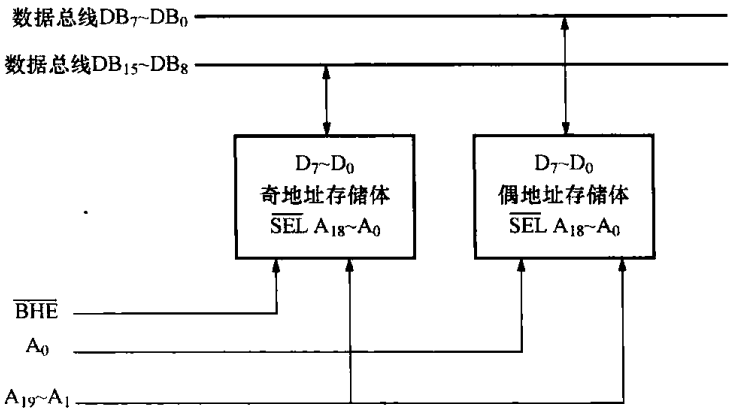


图 2.11　8086 系统中存储器与总线的连接方式

偶存储体与数据总线的低 8 位相连，奇存储体与数据总线的高 8 位相连。对于存储体的选择和数据的传送，使用 A_0 地址线和数据总线高 8 位控制信号$\overline{BHE}$来实现字节和字的读/写操作。A_0 和$\overline{BHE}$的组合及其对应的读/写操作如表 2.7 所示。

表 2.7　　A_0 和$\overline{BHE}$的组合及其对应的读/写操作

$\overline{BHE}$	A_0	操　　作	使用的数据引脚
0	0	读或写偶地址的一个字	AD_{15}～AD_0
1	0	读或写偶地址的一个字节	AD_7～AD_0
0	1	读或写奇地址的一个字节	AD_{15}～AD_8
0	1	读或写奇地址的一个字	AD_{15}～AD_8（第一个总线周期放低 8 位数据字节）
1	0		AD_7～AD_0（第二个总线周期放高 8 位数据字节）

8086CPU 访问存储器的情况分析如下：

(1) 字节访问：当对存储器读/写一个字节时，仅选中某个存储体（偶存储体或奇存储体），对应的 8 位数据在数据总线上有效，另 8 位数据被忽略。

(2) 字访问：当 CPU 要对存储器读/写一个字时，则有两种可能情况：

如果要访问的字单元地址是从偶地址开始的两个字节（即高字节在奇地址单元中，低字节在偶地址单元中），则只需要访问一次存储器，即执行一个总线周期就可读/写一个字（16 位）信息。

如果要访问存储器的字单元地址是从奇地址开始的两个字节（高字节在偶地址单元、低地址在奇地址单元），则 CPU 要两次访问存储器，即需要用 2 个总线周期才能读/写这个字的信息。第一次读/写奇地址单元中的数据（偶地址单元中的 8 位数据被忽略），第二次读/写偶地址单元中的数据（奇地址单元中的 8 位数据被忽略）。显然，访问从奇地址开始的字比访问从偶地址开始的字的速度要慢。因此为了加快程序的运行速度，编程时应注意从存储器的偶地址开始存放字数据，这种存放方式也叫“对准存放”。

必须指出，在 8086 系统的编程中并不涉及上述这些细节，一条指令只是请求访问一个特定的字或字节，而实现这一访问所必须做的一切（选存储体、忽略哪个字节、地址变换等）都是由 CPU 自动完成的。也就是说，存储器的这种分体结构对用户来说是透明的。

3. 8086 的 I/O 端口

在 8086 微机系统中，配置了一定数量的输入/输出设备，而这些设备必须通过输入/输出接口芯片与 CPU 相连接。每个 I/O 接口芯片都有一个或几个 I/O 端口（端口指接口中可被 CPU 直接读/写的寄存器），像存储器一样，每个 I/O 端口都有一个唯一的端口地址，以供 CPU 访问。

8086 用地址总线的低 16 位 A_{15}～A_0 来确定端口地址，所以 8086 CPU 可以访问的 I/O端口地址共有 64KB，其地址范围为 0000H～FFFFH。详细内容在第 3 章和第 5 章中介绍。

4. 8086 专用存储区

需要指出的是，8086 系统是一个通用微机系统，在存储空间的安排上，有一部分空间被系统占用，用户不能使用。具体占用情况如下：

00000H～003FFH 共 1KB 单元用来存放中断向量，该区域称为中断向量表。

FFFF0H～FFFFFH（存储器底部）共 16 个单元，一般用来存放一条无条件转移指令，将系统转入系统初始化程序，这样当系统加电或复位后就自动进入系统程序。

2.2.6 8086 最大模式和最小模式系统构成

用 8086 微处理器构成一个完整的微机系统，还需要配置诸如时钟发生器 8284、地址锁存器 8282、总线收发器 8286、总线控制器 8288 等支持芯片。针对 8086/8088 CPU 的两种最大和最小工作模式，存在最小模式系统和最大模式系统。

1. 最小模式系统

8086 CPU 最小模式下的典型配置如图 2.12 所示。这是一个以 8086 CPU 为主体的单机系统，这时 MN/$\overline{MX}$=+5V。系统中除了 8086CPU、存储器及 I/O 接口芯片外，还要 1 片时钟发生器 8284，3 片地址锁存器 8282/8283 或 74LS373，2 片总线收发器 8286/8287。

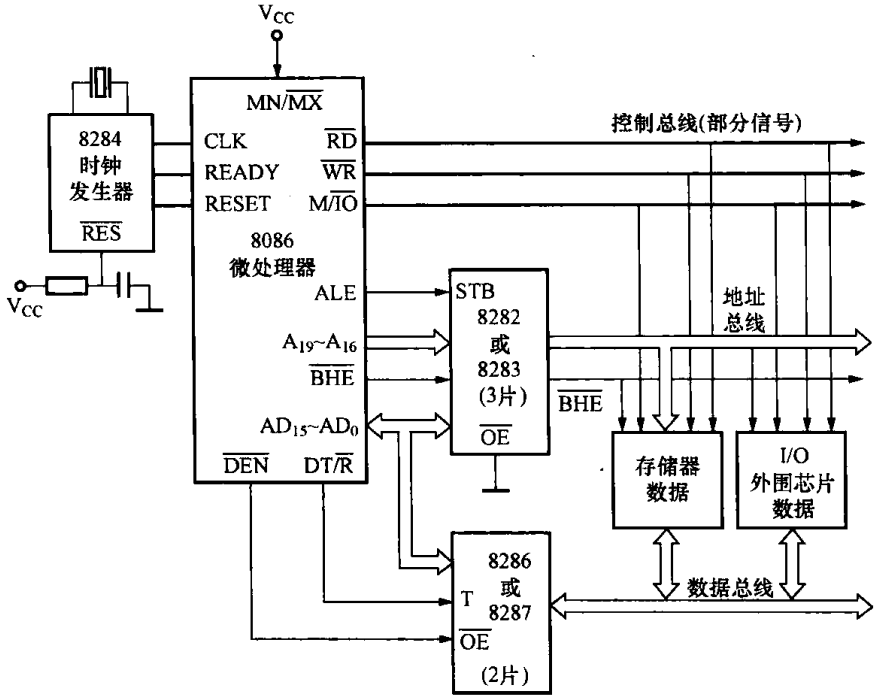
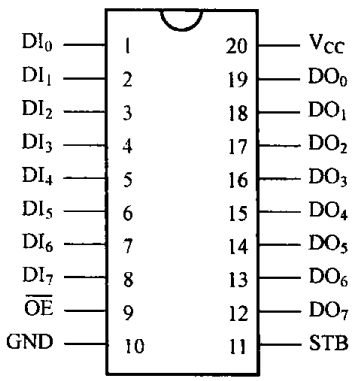


图 2.12 8086 最小模式下的典型配置

(1) 时钟发生器 8284。在 8086CPU 内部没有时钟信号发生器，所需的时钟信号需由外部时钟发生器电路提供。8284 就是为 8086 设计的时钟发生器。8284 在系统中最主要的作用有两个：产生和向 CPU 提供时钟信号 CLK、复位信号 RESET 和准备就绪信号 READY；其次，对外部输入的复位信号 RES 和准备好信号 READY 进行同步控制。

(2) 地址锁存器 8282/8283。8086CPU 部分引脚分时复用，CPU 在总线周期的 T_1 状态 $AD_{15}\sim AD_0$ 和 $A_{19}/S_6\sim A_{16}/S_3$ 用作地址总线，T_2 状态以后这些引脚用作数据和状态总线。而存储器或 I/O 接口电路与 CPU 进行数据传送时需要在整个总线周期内地址不变，因而必须加入地址锁存器，在总线周期的 T_1 状态（即在数据送上总线之前）先将地址锁存起来。8282 芯片及真值表如图 2.13 所示。



8282真值表

$\overline{OE}$	STB	输出
1	X	高阻
0	1	$DO_i=DI_i$
0	↓	锁存

图 2.13 8282 芯片及真值表

(3) 数据总线收发器 8286/8287。当一个系统中数据总线上挂接的外设接口部件较多时，必须在数据总线上接入总线收发器来增加总线的驱动能力。

图 2.14 中，引脚$\overline{OE}$=0 时，数据通过；否则，禁止数据通过，且输出被置为高阻。

引脚 T 控制芯片的收发方向，T=0 时，A_7～A_0 为输入；反之，A_7～A_0 为输出。

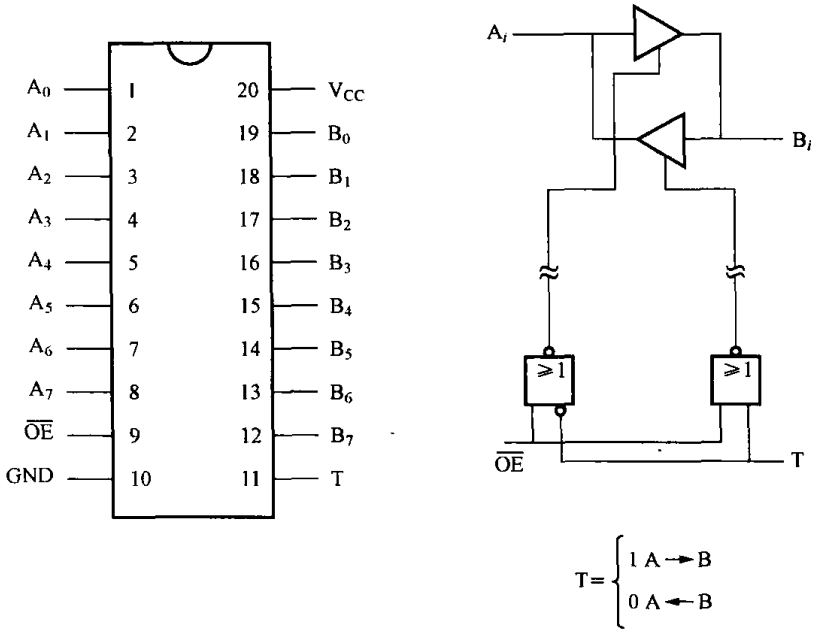


图 2.14 8286 芯片及逻辑

2. 最大模式系统

将 8086CPU 的引脚 MN/$\overline{MX}$接地，使 CPU 工作于最大工作模式。图 2.15 给出了 8086 CPU 最大模式下的典型配置。图 2.15 和图 2.12 的最小模式系统比较，增加了一个总线控制器 8288。

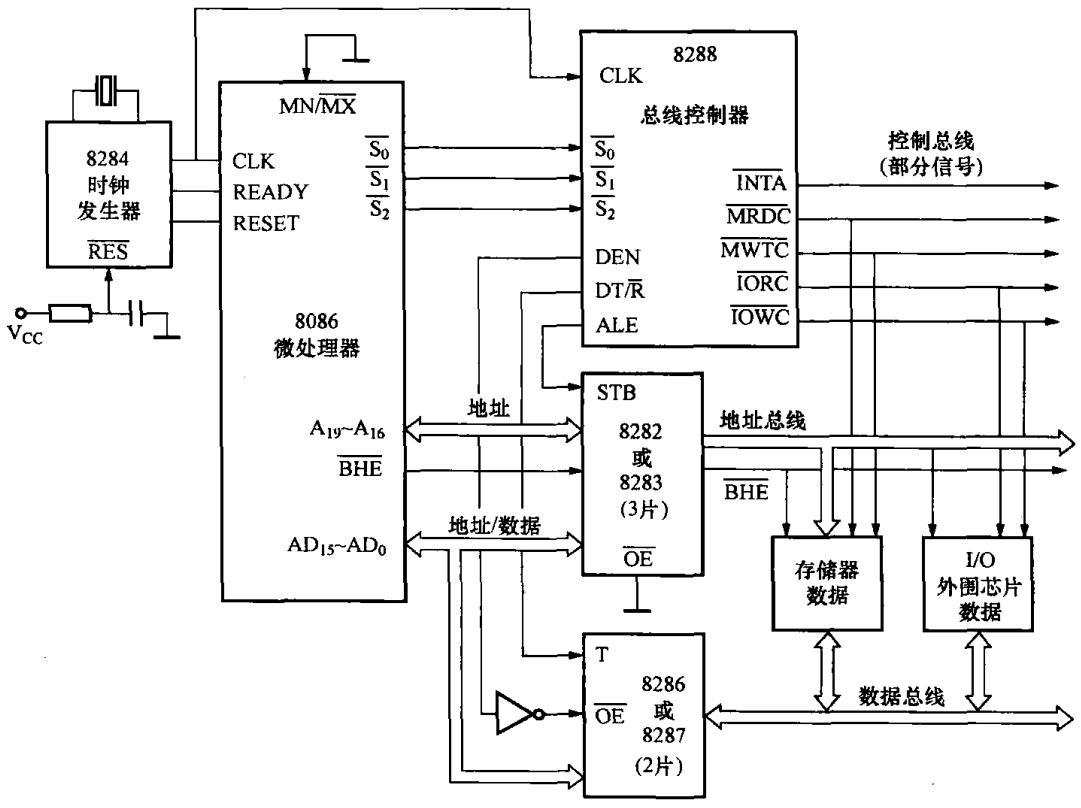


图 2.15 8086CPU 最大模式下的典型配置

在最大模式下，CPU 的总线状态信号$\overline{S_2}$，$\overline{S_1}$，$\overline{S_0}$输入总线控制器 8288 后，由 8288 译码，输出系统所需的总线命令和控制信号。如在最大模式下，由 8288 产生对存储器和 I/O 端口的读/写信号，而在最小模式下则是用 M/$\overline{IO}$与$\overline{RD}$或$\overline{WR}$信号的组合来控制读/写操作的。

2.2.7 8086CPU 的工作时序

计算机的工作过程是执行指令的过程，8086CPU 的操作是在时钟脉冲 CLK 的统一控制下进行的。微处理器的工作时序是指各引脚在时序上的工作关系，CPU 的每条指令都有各自固定的工作时序。

1. 时钟周期、总线周期和指令周期

(1) 时钟周期：每个时钟脉冲持续时间称为时钟周期，是微处理器执行指令的最小时间单位，又称 T 状态。时钟周期等于微机主频的倒数。例如，微处理器主频为 5MHz，则时钟周期为 200ns。

(2) 总线周期：CPU 通过总线对内存或 I/O 端口进行一次读/写过程所需的时间称为总线周期，一个总线周期包括多个时钟周期。如 8086 微处理器的基本总线周期由四个时钟周期 $T_1 \sim T_4$ 组成，80486 微处理器的基本总线周期由 T_1 和 T_2 两个时钟周期组成。当外设速度较慢时，可插入若干个时钟周期，此时的时钟周期称为等待周期，用 T_W 表示。

(3) 指令周期：CPU 执行一条指令所需要的时间称为指令周期。指令周期由若干个总线周期构成，不同功能的指令或同一功能的指令而寻址方式不同时，指令周期也不同。

8086 基本总线周期如图 2.16 所示。

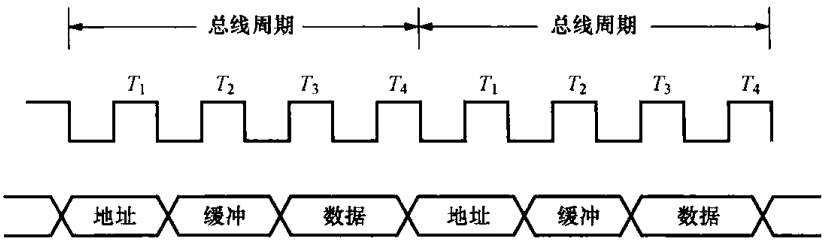


图 2.16 8086 基本总线周期

2. 总线操作时序

总线操作时序是地址信号、数据信号、控制信号和时钟信号之间的定时关系，这种关系一般用时序图来表示。了解 CPU 的总线周期和时序是分析和设计微机应用系统的重要基础。

分析总线操作时序，应该首先了解各引脚的意义，按照时间的先后顺序，分析每个引脚的变化过程，掌握相关信号之间的逻辑关系。

需要指出的是：

1) 基本总线周期由 4 个时钟周期组成。如果在一个基本总线周期 4 个 T 状态内不能完成一次读/写操作，则要在总线周期的 T_3 和 T_4 之间插入一个或若干个等待状态 T_W。

2) 根据总线周期的定义，只有当 BIU 要访问存储器或 I/O 端口时，才执行总线周期。另外，在完成一个总线周期后，如果不立即执行下一个总线操作（如指令队列已满，EU 又未申请总线操作），这时 BIU 便进入总线空闲状态（用 T_i 表示），一个空闲状态等于一个时钟周期。

8086 CPU 的总线操作按数据传送方向可分为总线读操作和总线写操作。前者是指 CPU

从存储器或I/O端口读取数据，对CPU来说，是一次输入操作；后者则是指CPU把数据写入到存储器或I/O端口，对CPU来说，是一次输出操作。

8086的主要操作时序有以下几种：

1）系统的复位和启动；

2）最小模式下的读总线周期、写总线周期；

3）最小模式下的总线请求和响应周期；

4）最大模式下的读总线周期、写总线周期；

5）最大模式下的总线请求/允许周期；

6）中断响应周期；

7）总线空操作周期。

下面主要介绍系统的复位和启动以及存储器的读/写总线周期时序。8086CPU在最大模式下的时序除有些信号是由总线控制器（8288）产生的以外，其基本时间关系与最小模式基本相同。

（1）系统的复位和启动。当时钟发生器8284向CPU的RESET引脚输入一个复位触发信号RESET，使8086 CPU复位和重新启动。8086要求复位信号RESET至少维持四个时钟周期的高电平。在复位状态，CPU内部的CS寄存器被置为FFFFH，其他寄存器被置为0000H。所以，RESET恢复低电平后，8086 CPU便从FFFF0H单元处开始启动。FFFF0H称为系统的启动地址。

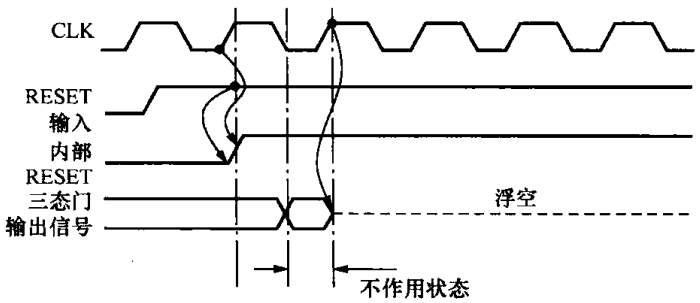


图2.17 8086复位操作的时序

复位时的操作时序如图2.17所示。当RESET信号有效后，再经过一个T状态，将执行：

1）把所有三态输出线（包括$AD_{15}\sim AD_0$，$A_{19}/S_6\sim A_{16}/S_3$，$\overline{BHE}/S_7$，$M/\overline{IO}$，$DT/\overline{R}$，$\overline{DEN}$，$\overline{RD}$，$\overline{WR}$和$\overline{INTA}$）都置成高阻抗状态，直到RESET信号为低电平结束复位操作为止。而且这些信号在进入高阻抗状态的前半个时钟周期先被置为不起作用状态。

2）把不具有三态功能的控制信号（ALE，$\overline{HLDA}$，$\overline{RQ}/\overline{GT_0}$，$\overline{RQ}/\overline{GT_1}$，$QS_0$，$QS_1$）都置为无效状态。

（2）最小模式下的总线读操作。总线读操作就是指CPU从存储器或I/O端口读取数据。基本读总线周期包含4个时钟周期T_1、T_2、T_3和T_4。当存储器或外设速度较慢时，在T_3和T_4之间插入等待状态T_W。图2.18所示为8086在最小模式下的总线读操作时序图。

1）T_1状态。为了从存储器或I/O端口读出数据，首先要用$M/\overline{IO}$信号指出CPU是要从内存还是I/O端口读，所以$M/\overline{IO}$信号在T_1状态成为有效（见图2.18①）。$M/\overline{IO}$信号的有效电平一直保持到整个总线周期的结束即T_4状态。

为指出CPU要读取的存储单元或I/O端口的地址，8086的20位地址信号通过$A_{19}/S_6\sim A_{16}/S_3$和$AD_{15}\sim AD_0$输出，送到存储器和I/O端口（见图2.18②）。

为了实现对地址的锁存，CPU便在T_1状态从ALE引脚上输出一个正脉冲作为地址锁

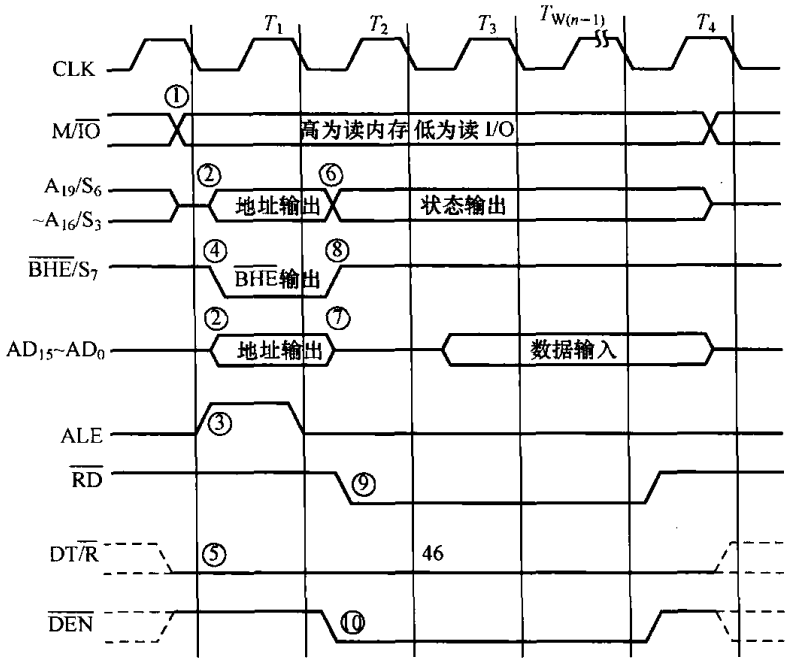


图 2.18　8086 在最小模式下的总线读操作时序图

存信号（见图 2.18③）。在 ALE 的下降沿到来之前，M/$\overline{IO}$信号、地址信号均已有效。锁存器 8282 正是用 ALE 的下降沿对地址进行锁存的。

$\overline{BHE}$信号也通过$\overline{BHE}/S_7$ 引脚送出（见图 2.18④），用来表示高 8 位数据总线上的信息可以使用。

此外，当系统中接有数据总线收发器时，在 T_1 状态 DT/$\overline{R}$输出低电平，表示本总线周期为读周期，即让数据总线收发器接收数据（见图 2.18⑤）。

2）T_2 状态。地址信号消失（见图 2.18⑦），AD_{15}～AD_0 进入高阻状态，为读入数据作准备；而 A_{19}/S_6～A_{16}/S_3 和$\overline{BHE}/S_7$ 输出状态信息 S_7～S_3（见图 2.18⑥和⑧）。

$\overline{DEN}$信号变为低电平（见图 2.18⑩），使总线收发器获得数据允许信号。

CPU 在$\overline{RD}$引脚上输出读有效信号（见图 2.18⑨），送到系统中所有存储器和 I/O 接口芯片。但是，$\overline{RD}$信号只能从地址信号选中的存储单元或 I/O 端口读出数据，送到系统数据总线上。

3）T_3 状态。在 T_3 状态前沿（下降沿处），CPU 对引脚 READY 进行采样，如果 READY 信号为高，则 CPU 在 T_3 状态后沿（上升沿处）通过 AD_{15}～AD_0 获取数据；如果 READY 信号为低，将插入等待状态 T_W，直到 READY 信号变为高电平。

4）T_W 状态。当系统中所用的存储器或外设的工作速度较慢，从而不能用最基本的总线周期执行读操作时，系统中就要用一个电路来产生 READY 信号。低电平的 READY 信号必须在 T_3 状态启动之前向 CPU 发出，则 CPU 将会在 T_3 状态和 T_4 状态之间插入若干个等待状态 T_W，直到 READY 信号变高。在执行最后一个等待状态 T_W 的后沿（上升沿）处，CPU 通过 AD_{15}～AD_0 获取数据。

5）T_4 状态。总线操作结束，相关系统总线变为无效电平。

（3）最小模式下的总线写操作。总线写操作就是指 CPU 向存储器或 I/O 端口写入数据。图 2.19 所示为 8086 在最小模式下的总线写操作时序图。

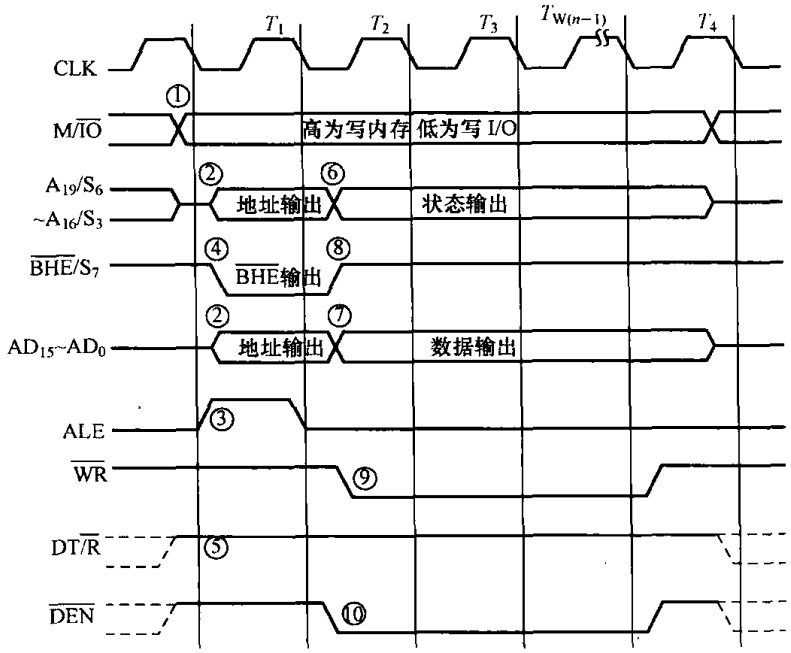


图 2.19 8086 在最小模式下的总线写操作时序图

总线写操作时序与总线读操作时序进行对比有许多基本相同的地方，它们不同的地方有两点：

1）对存储器或 I/O 端口操作选通信号的不同。总线读操作中，选通信号是$\overline{RD}$，而总线写操作中是$\overline{WR}$。

2）在 T_2 状态中，AD_{15}～AD_0 上地址信号消失后，AD_{15}～AD_0 的状态不同。总线读操作中，此时 AD_{15}～AD_0 进入高阻状态，并在随后的状态中为输入数据；而在总线写操作中，此时 CPU 立即通过 AD_{15}～AD_0 输出数据，并一直保持到 T_4 状态中间。

2.3 80486 微 处 理 器

80486 微处理器采用先进的集成电路工艺，把主处理器部件、数值协同处理器部件和一个 8K 字节的高速缓冲存储器部件 Cache 集成在一个芯片上。80486 微处理器内外部数据总线均为 32 位，地址总线为 32 位，可直接寻址 4GB 的存储空间，支持虚拟存储管理技术，虚拟存储空间为 64TB。并且在 80X86 系列中首次采用了 RISC（精简指令集）技术，许多指令能够在一个时钟周期时间内完成。此外，80486 微处理器还引进了时钟倍频技术和新的内部总线结构。

2.3.1 80486CPU 的工作方式

从操作系统的角度看，Intel 80486 微处理器有实地址模式、保护模式和虚拟 8086 模式三种工作模式。通过设置控制寄存器 CR0 中的保护模式允许位 PE，可以进行实地址模式和保护模式之间的转换。执行 IRET 指令或进行任务切换，可由保护模式转移到虚拟 8086 模式。虚拟 8086 方式与实地址方式都是为了兼容 8086/8088/80186/80188 而设置的 16 位操作方式，它们的段首址都为段寄存器值乘以 16，最大段容量均为 64KB，最大寻址空间

为1MB。

1. 实地址模式

实地址模式（Real Mode）是最基本的工作方式，当CPU复位后，系统自动进入实地址模式。实地址模式与16位微处理器8086微处理器具有同样的基本结构，只不过是在这种操作方式下，80486可以访问32位的寄存器，增加了寄存器，扩充了指令，具有更强的功能。16位微处理器的程序不加任何修改就可以在80486微处理器实地址模式下运行。

80486微处理器在实模式操作方式下，只允许寻址第一个1MB存储器空间，缺省操作数的大小是16位的，为了能在这种方式下也可以使用32位寄存器和寻址方式，必须加上指令前缀。当80486微处理器工作于实地址模式时，存储器的管理方式与8086微处理器存储器的管理方式完全相同。通常，存储器中第一个1MB存储单元称为实模式存储器或常规内存，DOS操作系统要求微处理器工作于实模式。

在实地址方式下，有两个物理存储空间是需要保留的。物理地址0000，0000H～0000，03FFH是中断向量区，每一中断向量占用4个字节；物理地址FFFF、FFF0H～FFFF，FFFFH为系统初始化区，当加电或复位时，物理地址自动置为FFFF、FFF0H，程序就从此地址开始运行。当首次遇到段间转移或段间调用指令时，物理地址又自动置为000×，××××H（×为任意值），从而进入实地址方式下的物理地址空间。

2. 保护工作模式

保护工作模式也称为保护的虚拟地址方式。在程序运行过程中，应防止应用程序破坏系统程序、某一应用程序破坏了其他应用程序、错误地把数据当作程序运行等情形的出现。为避免出现这些情形所采取的措施称作“保护”。

保护模式（Protected Virtual Address Mode）的特点是引入了虚拟存储器的概念，同时可使用附加的指令集，使80486支持多任务操作。80286及其以上的微处理器允许访问第一个1MB及其以上的存储段内的数据和程序。寻址这个扩展的存储段，需要更改用于实模式存储器寻址的段基址加偏移地址的机制。

80486CPU工作在保护模式时，使用46位虚拟地址，虚拟地址空间最大可达64TB（2^{46}字节），采用分段分页管理机制进行管理。将80486的低32位地址称为线性地址，所确定的线性地址空间称为段，一个段的最大长度为4GB（2^{32}字节）段数最多可达2^{14}个。采用分页管理机制时，用户在程序中所使用的地址是由“段选择符”和“偏移量”组成的逻辑地址，程序在系统中运行时，由存储管理部件把逻辑地址转换成线性地址。当不分页时，线性地址就是物理地址。

在保护模式下，当寻址扩展内存的数据和程序时，仍然使用偏移地址访问位于存储器段内的信息。区别是，实模式下的段基址由段寄存器提供，而保护模式下段寄存器提供的是一个段选择符（Selector），用于确定描述表内的一个描述符。描述符（Descriptor）描述存储器段的位置、长度和访问权限。由于段基址加偏移地址仍然用于访问第一个1MB存储器内的数据，所以保护模式下的指令和实模式下的指令完全相同。

3. 虚拟8086模式

虚拟8086模式是80486保护模式下的一种子方式。它的最大特点是，8086程序可以使用80486保护模式下的分段和分页保护机制运行。虚拟8086模式下，80486微处理器可以

在所使用的 80486 的操作系统之下，既可以同时执行 8086 操作系统和应用程序，也可以执行 80486 操作系统和应用程序。因此，在一台多用户的 80486 微处理器的计算机里，多个用户都可以同时使用计算机。

虚拟 8086 方式与实地址方式的区别主要表现在以下两个方面：

(1) 实地址方式是整个 CPU 的工作方式，是实实在在的 8086 方式，此时的 CPU 相当于一个快速的 8086，只能运行 8086 程序的任务。而虚拟 8086 方式是一种模拟（仿真）8086 方式，是在虚拟存储器、保护和多任务操作等环境支持下的工作方式。所以，在一个多任务操作系统环境中，既可在虚拟 8086 这种仿真方式下创建和运行 8086 程序的任务，又可在保护虚地址方式下创建和运行 80286 程序的任务和 80386/80486 程序的任务。

(2) 在实地址方式下，各个段的特权级都为 0 级（最高级）；而在虚拟 8086 方式下，各个段的特权级都为 3 级（最低级）。这样，在虚拟 8086 方式下，就限制了一些特权指令的执行。

2.3.2 80486CPU 的内部结构

80486CPU 内部包括总线接口单元、指令预取单元、指令译码单元、控制和保护测试单元单元、整数执行单元、分段单元、分页单元，以及浮点运算部件和高速缓存（Cache）管理单元，其内部结构如图 2.20 所示。

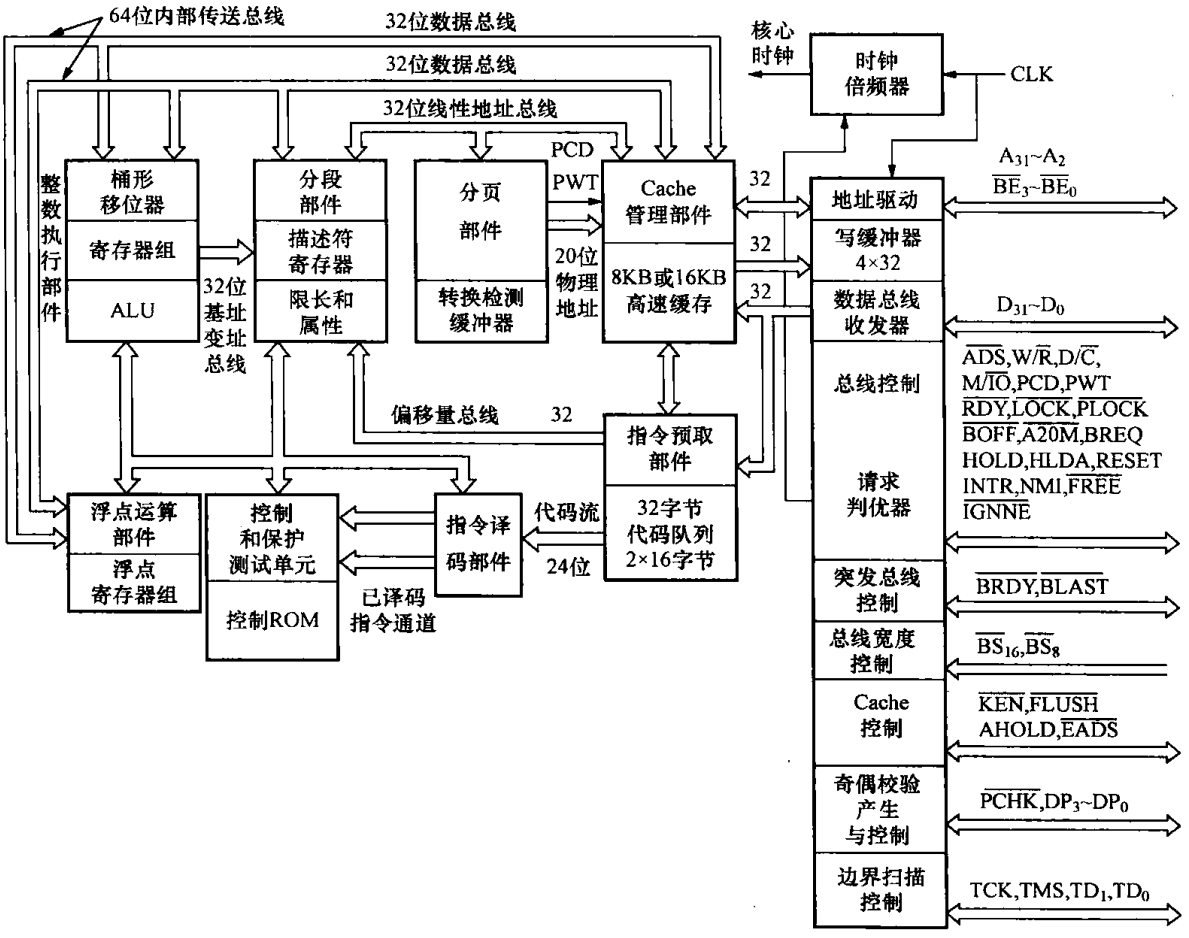


图 2.20 80486 微处理器的内部结构

1. 总线接口单元

总线接口单元（BIU）主要有两方面的作用：

（1）BIU与外部总线连接，用于管理访问外部存储器和I/O端口的地址、数据和控制总线。在预取指令代码时，BIU把从外部存储器取出的指令代码同时传送给代码预取单元和内部高速缓冲存储器，以便在下一次预取相同的指令时，可直接访问高速缓冲存储器。

（2）在微处理器内部，BIU主要与指令预取单元和高速缓存单元交换信息，将预取指令存入指令代码队列。BIU与Cache单元交换数据有三种情况：一是向高速缓冲存储器填充数据，BIU一次从片外总线读取16个字节到Cache；二是如果高速缓冲存储器的内容被处理器内部操作修改了，则修改的内容也由BIU写回到外部存储器中去；三是如果一个读操作请求所要访问的存储器操作数不在高速缓冲存储器中，则这个读操作便由BIU控制总线直接对外部存储器进行操作。

2. 指令预取单元

80486 CPU内部有一个32字节的指令预取队列，在总线空闲周期，指令预取单元（Pre-fetcher）形成存储器地址，并向BIU发出预取指令请求。预取单元一次读取16个字节的指令代码存入预取队列中，指令队列遵循先进先出FIFO(First in First Out）的规则，自动地向输出端移动。如果Cache在指令预取时命中，则不产生总线周期。当遇到跳转、中断、子程序调用等操作时，预取队列被清空。

3. 指令译码单元

指令译码单元IDU(Instruction Decode Unit）从指令预取队列中读取指令并译码，将其转换成相应控制信号。译码过程分两步：首先确定指令执行时是否需要访问存储器，若需要则立即产生总线访问周期，使存储器操作数在指令译码后能准备好；然后产生对其他单元的控制信号。

4. 控制和保护测试单元

控制单元CPTU(Control and Protection Test Unit）对整数执行单元、浮点运算单元和分段管理单元进行控制，使它们执行已译码的指令。

5. 整数执行单元

整数执行单元IU(Integer Data-path Unit）包括四个32位通用寄存器、两个32位间址寄存器、两个32位指针寄存器、一个标志寄存器、一个64位桶形移位寄存器和算术逻辑运算单元等。它能在一个时钟周期内完成整数的传送、加减运算、逻辑操作等。80486 CPU采用了RISC技术，并将微程序逻辑控制改为硬件布线逻辑控制，缩短了指令的译码和执行时间，一些基本指令可在一个时钟周期内完成。

两组32位双向总线将整数单元和浮点单元联系起来，这些总线合起来可以传送64位操作数。这组总线还将处理器单元与Cache联系起来，通用寄存器的内容通过这组总线传向分段单元，并用于产生存储器单元的有效地址。

6. 浮点运算单元

80486CPU内部集成了一个增强型80487数学协处理器，称为浮点运算单元FPU（Floating Point Unit），用于完成浮点数运算。由于FPU与CPU集成封装在一个芯片内，而且它与CPU之间的数据通道是64位的，所以当它在内部寄存器和片内Cache取数时，运行速度会极大提高。

7. 分段单元和分页单元

80486 CPU 设置了分段单元 SU(Segmentation Unit) 和分页单元 PU(Paging Unit)，实现存储器保护和虚拟存储器管理。分段单元将逻辑地址转换成线性地址，采用分段 Cache 可以提高转换速度。分页单元用来完成虚拟存储，把分段单元形成的线性地址进行分页，转换成物理地址。为提高页转换速度，分页单元中还集成了一个转换后援缓冲器 TLB(the Translation Look-aside Buffer)。

8. Cache 管理单元

80486 CPU 内部集成了一个数据/指令混合型 Cache，称为高速缓冲存储器管理单元 CU (Cache Unit)。在绝大多数的情况下，CPU 都能在片内 Cache 中存取数据和指令，减少了 CPU 的访问时间。在与 80486 DX 配套的主板设计中，采用 128～256KB 的大容量二级 Cache 来提高 Cache 的命中率，片内 Cache(L1 Cache) 与片外 Cache(L2 Cache) 合起来的命中率可达 98%。CPU 片内总线宽度高达 128 位，总线接口单元将以一次 16 个字节的方式在 Cache 和内存之间传输数据，大大提高了数据处理速度。80486 CPU 中的 Cache 单元与指令预取单元紧密配合，一旦预取代码未在 Cache 中命中，BIU 就对 Cache 进行填充，从内存中取出指令代码，同时送给 Cache 单元和指令预取单元。

2.3.3 80486CPU 的寄存器

微处理器的寄存器按功能可分为基本寄存器、系统寄存器、浮点寄存器和调试测试寄存器四类。应用程序只能访问基本寄存器和浮点寄存器；而系统程序可访问所有的寄存器。

1. 基本寄存器

基本寄存器用于汇编语言所编写的程序访问。基本寄存器包括 8 个通用寄存器 EAX，EBX，ECX，EDX，EBP，ESP，EDI，ESI；一个指令指针寄存器 EIP；6 个段寄存器 CS，DS，ES，SS，FS 和 GS；一个标志寄存器 EFLAGS。图 2.21 给出了 80486 微处理器的基本寄存器。

(1) 通用寄存器。80486 共配置了 8 个 32 位的通用寄存器，如图 2.21 (a) 所示。它们的名字和用途分别为：

EAX：用来作累加器寄存器；

EBX：用来作基址寄存器；

ECX：用来作计数器寄存器；

EDX：用来作数据寄存器；

EBP：用来作基址指针寄存器；

ESP：用来作堆栈指针寄存器；

EDI：用来作目的变址寄存器；

ESI：用来作源变址寄存器。

这 8 个 32 位寄存器不仅可以存放 32 位数据，以支持 32 位的数据操作，而且也可以进行 16 位的操作，以使与 Intel 系列 16 位微处理机兼容。所以可以把这 8 个 32 位的低 16 位寄存器看成 8 个 16 位的通用寄存器。它们分别以 AX，BX，CX，DX，SI，DI，BP 和 SP 命名，并且 AX，BX，CX，DX 的低位字节或高位字节也可作为独立的 8 位寄存器使用，与 8 位机兼容。

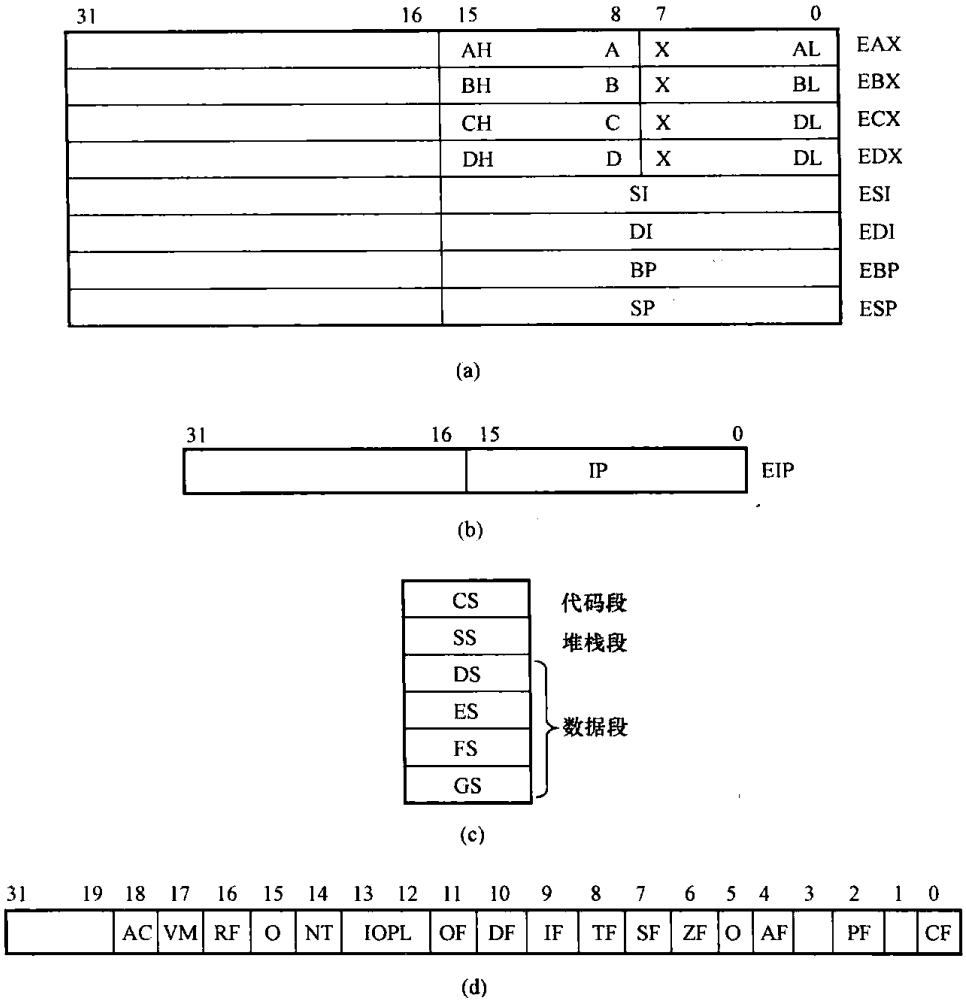


图 2.21 基本寄存器

(a) 通用寄存器；(b) 指令指针；(c) 段寄存器；(d) 标志寄存器

在进行地址计算和进行绝大多数算术及逻辑运算时，这 8 个通用寄存器都可以使用，只有少数指令指定所使用的寄存器。例如，80486 串操作指令规定寄存器 ECX 作为计数器、源变址寄存器 ESI 和目的变址寄存器 EDI 作为操作数的地址指针。

(2) 指令指针寄存器 EIP。指令指针寄存器 EIP(Extra Instruction Pointer) 是一个 32 位的寄存器，如图 2.21 (b) 所示。在指令指针寄存器内存放的是当前代码段内下一条要执行指令的偏移地址。微处理器工作于实模式下，EIP 是 IP (16 位) 寄存器。

程序设计人员不能直接对指令指针寄存器编程，每当从当前段取出指令的一个字节后，EIP 自动加 1，当执行了控制转移指令、调用指令或异常中断，则由这些指令置入新的偏移地址，开始新的程序执行。

(3) 段寄存器。80486 内部有 6 个 16 位的段寄存器，分别存放段基址 (实地址模式下) 或选择符 (保护模式下)。它们的命名如图 2.21 (c) 所示，分别为代码段寄存器 CS、数据段寄存器 DS、堆栈段寄存器 SS、附加数据段寄存器 ES、FS 和 GS。

80486 为每个段都定义了一个描述符，用来说明每个段的基地址、界限以及其他一些段属性。把操作系统使用的段描述符和各项任务共用的段描述符放在一起，就称为全局描述符表 GDT (Global Descriptor Table)。为了能正确地标识出一个段描述符在哪一个表中，其

编号又是多少，它的特权级为多少，所以要为每个段定义一个 16 位的选择符。通过段选择符才能检测到存储器中的描述符表。段寄存器中存放着内存 6 个存储器段的段选择符，所以也把段寄存器称为段选择符。

在实模式下，段寄存器存放段基址，所确定的段最大空间是 64KB；在保护模式下，段寄存器存放段选择符，所描述的段最大空间是 4GB。上述 6 个段寄存器，除 CS 是用于指示指令代码的地址空间之外，其他段寄存器都用于指示数据的地址空间。

在保护模式下，每个段寄存器都含有一个程序不可见区域。

(4) 标志寄存器。标志寄存器是 32 位的寄存器，称作 EFLAGS。EFLAGS 中的位可分为标志位和控制位系统标志位和 IDPL 字段两类，标志位指明程序运行时微处理器的实时状态；控制位由程序设计者设置，以控制 80486 进行某种操作。EFLAGS 的低 16 位也可作为一个独立的标志寄存器使用，它被称作 FLAGS，在实地址模式下使用，如图 2.21 (d) 所示。

1) 状态标志位：包括进位/借位标志 CF、奇偶标志 PF、辅助进位标志 AF、零标志 ZF、符号标志 SF 和溢出标志 OF。

2) 控制标志位：包括陷阱标志（单步操作标志）TF、中断标志 IF 和方向标志 DF。80486 CPU 标志寄存器中的状态标志位和控制标志位与 8086 CPU 标志寄存器中的状态标志位和控制标志位的功能完全一样，这里就不再赘述。

3) 系统标志位和 IOPL 字段：在 EFR 寄存器中的系统标志和 IOPL 字段，用于控制操作系统或执行某种操作。它们不能被应用程序修改。

IOPL(I/O Privilege Level Field)：输入/输出特权级标志位。它规定了能使用 I/O 敏感指令的特权级。在保护模式下，利用这两位编码可以分别表示 0，1，2，3 这四种特权级，0 级特权最高，3 级特权最低。在 80286 以上的处理器中有一些 I/O 敏感指令，如 CLI（关中断指令）、STI（开中断指令）、IN（输入）、OUT（输出）。IOPL 的值规定了能执行这些指令的特权级。只有特权高于 IOPL 的程序才能执行 I/O 敏感指令，而特权低于 IOPL 的程序，若企图执行敏感指令，则会引起异常中断。

NT(Nested Task Flag)：任务嵌套标志。在保护模式下，指示当前执行的任务嵌套于另一任务中。当任务被嵌套时，NT=1，否则 NT=0。

RF(Resume Flag)：恢复标志。与调试寄存器一起使用，用于保证不重复处理断点。当 RF=1 时，即使遇到断点或故障，也不产生异常中断。

VM(Virtual 8086 Mode Flag)：虚拟 8086 模式标志。用于在保护模式系统中选择虚拟操作模式。VM=1，启用虚拟 8086 模式；VM=0，返回保护模式。

AC(Alignment Check Flag)：队列检查标志。如果在不是字或双字的边界上寻址一个字，或双字，队列检查标志将被激活。

2. *系统级寄存器*

80486 CPU 系统级寄存器控制着片内 Cache、浮点部件 FPU 以及对存储器的分段分页管理操作。在保护模式下操作时，存储器系统中增加了全局描述符表、局部描述符表和中断描述符表。为了访问和指定这些表的地址，设置了 4 个控制寄存器和 4 个系统地址寄存器，如图 2.22 所示。

(1) 控制寄存器。80486 微处理器中有 4 个 32 位控制寄存器 CR0～CR3(Control Registers)，保存着全局性和任务无关的机器状态，如图 2.22 (a) 所示。

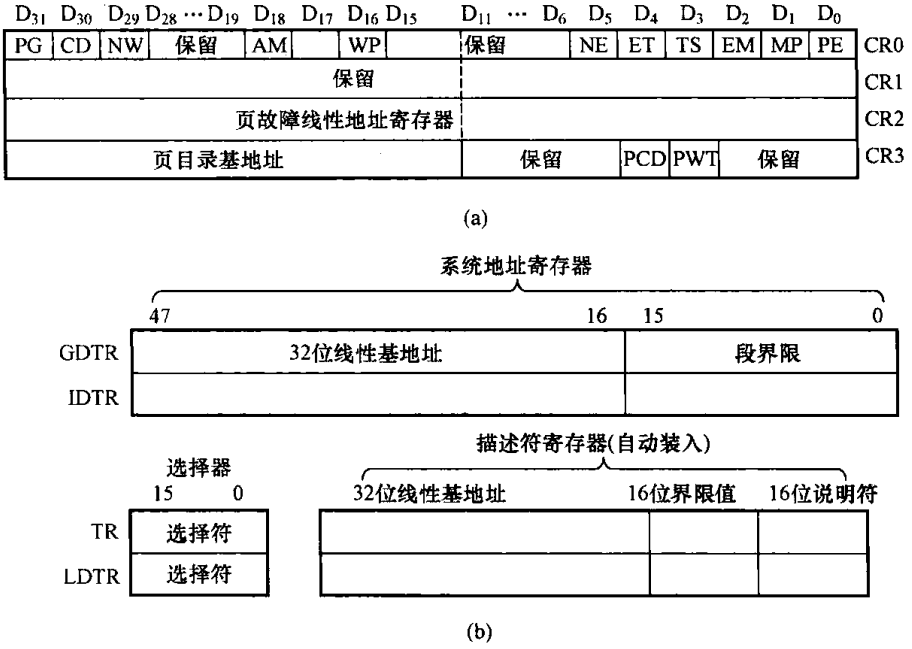


图 2.22 系统寄存器

(a) 控制寄存器；(b) 系统地址寄存器

1) CR0 控制寄存器。CR0 控制寄存器中有 11 位进行了定义，分别表示和控制着机器的状态，剩余的 21 位是 Intel 公司自己保留的。其中：

PE(Protection Enable)：保护模式允许位。PE＝1，系统在保护模式下运行；PE＝0，系统在实地址模式下运行。

MP(Monitor Coprocessor)：监视协处理器位。MP＝1，表示系统中有一个协处理器，否则 MP＝0。

EM(Emulation)：仿真协处理器位。EM＝1，表示微处理器中没有仿真协处理器，否则 EM＝0。

TS(Task Switched)：任务转换位。每次任务转换操作完成时，主处理器都自动将 TS 位置 1。在执行浮点运算指令时，需要对该位进行测试。

ET(Extension Type)：协处理器类型标志。用于指示系统中协处理器的类型。ET＝1，表示系统使用与 80387 兼容的 32 位协处理器。

NE(Numeric Error)：数值异常位。控制浮点运算中未被屏蔽的异常事故。NE＝1，则允许报告浮点数值异常。当 NE＝0，且$\overline{\text{IGNNE}}$输入引脚的信号有效时，则忽略数值异常。当 NE＝0，且$\overline{\text{IGNNE}}$输入引脚的信号无效时，将导致处理器停止工作，产生系统外部中断。

WP(Write Protect)：写保护。当 WP＝1 时，表示禁止系统对用户级只读页的访问，实行写保护。WP＝0 时，允许系统对用户级只读页进行访问。

AM(Alignment Mask)：对准标志位。当 AM＝1 时，允许自动对准校验。当 AM＝0 时，则不允许。

NW(Not Write-through)：不通写位，用这一位控制 Cache 的功能。当 NW＝0 和 CD＝0 时，允许对命中的 Cache 进行通写，这是 Cache 工作的必要条件。当 NW＝1，CD＝1 时，则不允许通写。

CD(Cache Disable)：允许 Cache 位。当 CD=0 时，系统内部的 Cache 允许使用。当 CD=1 时，且又不命中 Cache 时，则进行填充 Cache，但是，如果访问 Cache “命中”，则 Cache 仍可正常工作。若要完全使 Cache 停止工作，则必须刷新 Cache。

PG(Paging)：允许分页位。PG 位用来控制分页管理机构是否允许分页。PG=1，则允许分页；PG=0，则禁止分页。

2）CR1 控制寄存器。系统保留。

3）CR2 控制寄存器。当在分页操作期间出现了异常，此时在 CR2 寄存器内保存的是引起这次异常事故的全 32 位线性地址。当控制寄存器 CR0 中位 31 允许分页位被置成 1 时，CR2 才是有效的。当把页故障处理程序启动之后，由被压入该页故障处理程序堆栈中的错误码来提供页故障的状态信息。

4）CR3 控制寄存器。CR3 控制寄存器是页目录基址寄存器，CR3 的 20 位（12～31 位）保存着页目录表的物理基地址。CR3 还可由 PWT（第 3 位）和 PCD（第 4 位）对 Cache 进行控制。

（2）系统地址寄存器。系统地址寄存器又称作保护模式寄存器，仅在保护模式下使用，其内保存着操作系统需要的保护信息和地址转换信息，与段寄存器一起完成内存管理，为多任务环境、任务保护提供硬件支持。这些寄存器都是程序不可见的。系统地址寄存器如图 2.22（b）所示。

1）全局描述符表寄存器 GDTR(Global Descriptor Table Register)，是 48 位寄存器，其中高 32 位是全局描述符表 GDT 的线性基地址，指示 GDT 在存储器中的起始地址，低 16 位是段限值，规定了全局描述符表 GDT 的大小（按字节计算）。也就是说，由 GDTR 来确定全局描述表 GDT 所在存储器中的地址空间。

例如，若 GDTR＝0800、0000、0FFFH，则全局描述符表的第 1 个地址为 0800、0000H，其最后一个地址为 0800、0FFFH，全局描述符表的尺寸是 1000H。

在对存储器中的数据进行存取操作时，先要用段选择符到全局描述符表 GDT 或局部描述符表中去查找这个段的描述符，因为在这个段描述符中保存着这个段的基址及其他一些有关这个段的信息。

2）中断描述符表寄存器 IDTR(Interrupt Descriptor Table Register)，是一个 48 位的寄存器，用来存放中断描述符表 IDT 的 32 位基地址和 16 位段限值。80486 微处理器为每个中断定义了一个中断描述符，所有的中断描述符集中存放在中断描述符表 IDT 中，IDTR 指出中断描述符表 IDT 在内存中的位置。

3）局部描述符表寄存器 LDTR(Local Descriptor Table Register)，是一个 80 位的寄存器，由 16 位段选择符（程序可见部分）和 64 位描述符（程序不可见部分，由硬件自动装入内容）组成，用于保存局部描述符表 LDT(Local Descriptor Table) 的 32 位基地址、16 位段限值和 16 位访问权限等。因此，在指令系统中所用到的 LDTR 只是指 16 位的选择符。

4）任务寄存器 TR(Task Register)，任务状态寄存器完成任务的切换，是一个 80 位的寄存器，由 16 位段选择符和 64 位描述符组成，该寄存器用于存放当前正在执行的任务的线性基地址、界限、描述符的属性和 16 位的选择符。与 LDTR 类似，只有 16 位的选择符可由程序访问，因此，在指令系统中所用到的 TR 只是指 16 位的选择符。在微机中，

任务通常就是程序的进程或应用程序。任务切换允许微处理器在足够短的时间内实现任务之间的切换，也允许多任务系统以简单、规则的方式，从一个任务切换到另一个任务。

（3）调试与测试寄存器。80486 微处理器为程序设计人员提供了 8 个调试寄存器，如图 2.23（a）所示。DR0～DR3 用于设置数据存取断点和代码执行断点；DR7 是调试控制寄存器，用于选择调试功能和设置断点；DR6 是调试状态寄存器，它主要用于指明断点的当前状态。DR4 和 DR5 是 Intel 公司为 80486 微处理器之后所要开发的微处理器而保留的。

80486 微处理器还包含 5 个测试寄存器，如图 2.23（b）所示。TR3～TR5 用来实施对片内 Cacle 的测试。TR6～TR7 可用来控制分页机构中的转换旁视缓冲存储器 TLB 的运行。

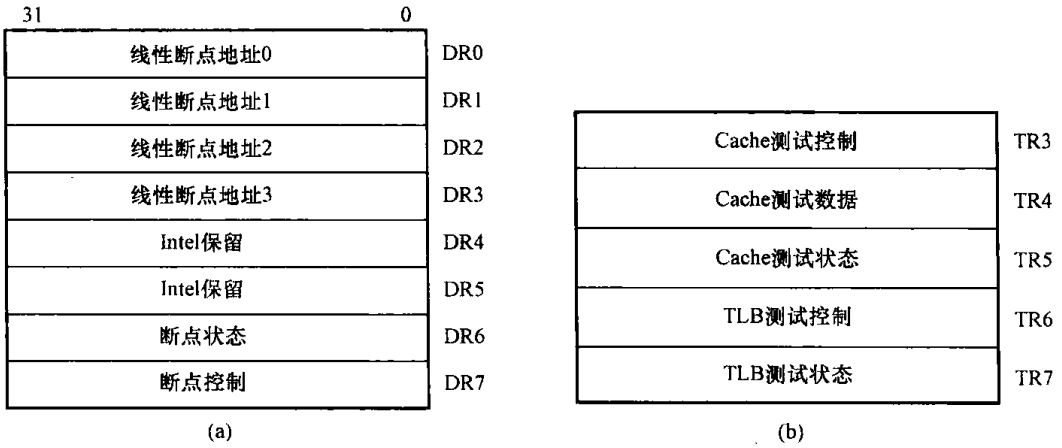


图 2.23　调试与测试寄存器

（a）调试寄存器；（b）测试寄存器

2.3.4　80486 的存储器组织和 I/O 结构

80486 微处理器在保护模式下对存储器管理有两个特点：一是设置了分段部件和分页部件，引入存储器地址转换机制；二是引入对存储器的保护机制。地址转换机制使操作系统可以灵活地把存储区域分配给各个任务，而保护机制用来避免系统中的一个任务越权访问属于另一个任务的存储区域或属于操作系统的存储区域。

1. 80486 存储器地址空间

80486 微处理器有 3 个明确的存储地址空间，它们是虚拟地址空间、线性地址空间和物理地址空间。

虚拟地址又称为逻辑地址，是程序员编写程序所使用的地址的表示方式，逻辑地址空间非常大，80486 微处理器的逻辑空间可达 2^{46} 字节（64TB）。逻辑地址由 16 位段选择符和 32 位偏移地址构成。

物理地址也称为主存地址，是计算机中主存储器的实际地址。80486 微处理器的主存地址空间为 2^{32} 字节（4GB）。

线性地址是 80486 微处理器通过分段部件把虚拟地址空间分为若干个各自独立的被保护的空间，称为一个段，是一个 32 位的线性空间，线性地址就是逻辑地址经分段部件转换后所形成的地址，由 32 位段基址和 32 位偏移地址构成。如果分页部件未被选用，线性地址就

是物理地址，如图 2.24（a）所示。

逻辑地址、线性地址和物理地址的关系如图 2.24（b）所示。

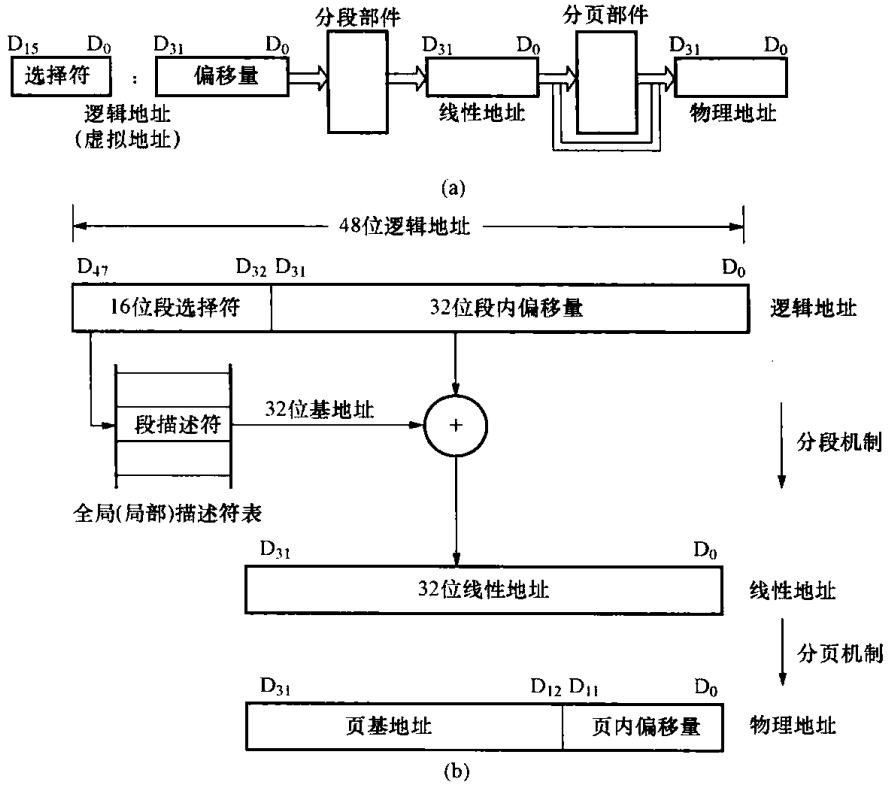


图 2.24 存储地址空间

（a）地址变换；（b）逻辑地址、线性地址和物理地址的关系

2. 存储器寻址

80486 CPU 的存储器与 I/O 地址空间采用独立的编址方式。

80486 CPU 具有 32 条地址总线，A_{31}～A_2 与字节允许信号$\overline{BE_3}$～$\overline{BE_0}$共同形成 32 位地址，可寻址 2^{32} 字节（4GB）的物理存储器空间，地址范围是 00000000H～FFFFFFFFH。从图 2.27 给出的 80486 外部引脚中可发现，地址总线中没有 A_1 和 A_0 这两条地址线，而是在 CPU 内部利用 A_1 和 A_0 这两条地址线经译码产生了字节允许信号$\overline{BE_3}$～$\overline{BE_0}$。地址总线 A_{31}～A_2 与字节允许信号$\overline{BE_3}$～$\overline{BE_0}$的作用，如表 2.8 所示。$\overline{BE_3}$～$\overline{BE_0}$与 32 位数据总线的对应关系表，如表 2.9 所示。

表 2.8 A_{31}～A_2 与$\overline{BE_3}$～$\overline{BE_0}$的作用

80486CPU 地址信号						
A_{31}～A_2 物理地址			$\overline{BE_3}$	$\overline{BE_2}$	$\overline{BE_1}$	$\overline{BE_0}$
A_{31}…A_2	A_1	A_0				
A_{31}…A_2	0	0	×	×	×	0
A_{31}…A_2	0	1	×	×	0	1
A_{31}…A_2	1	0	×	0	1	1
A_{31}…A_2	1	1	0	1	1	1

表 2.9　$\overline{BE_3}$～$\overline{BE_0}$与 32 位数据总线对应关系表

字节允许信号	数据总线信号	
$\overline{BE_0}$	D_7～D_0	字节 0（最低位）
$\overline{BE_1}$	D_{15}～D_8	字节 1
$\overline{BE_2}$	D_{23}～D_{16}	字节 2
$\overline{BE_3}$	D_{31}～D_{24}	字节 3

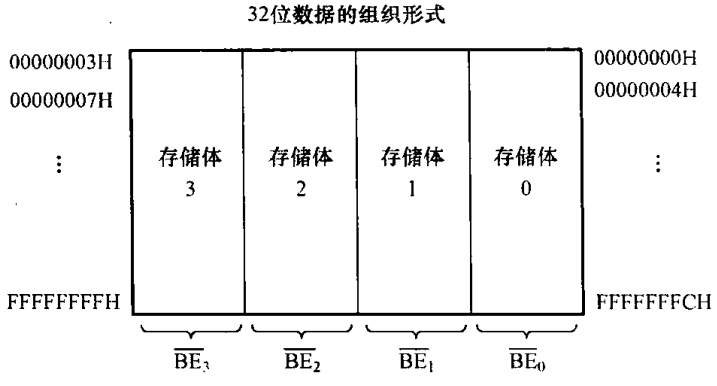


图 2.25　80486 微处理器存储器组织

由于 80486 CPU 的数据总线为 32 位，所以存储器和 I/O 地址空间都是针对 32 位数据宽度来组织的，32 位的数据组织分为 4 个存储体设计，如图 2.25 所示。

3. I/O 端口寻址

80486CPU 的 32 条地址总线中的低 16 位，用作对 I/O 端口的寻址，可寻址 64KB I/O 地址空间，地址范围是 0000H～FFFFH，如图 2.26 所示。

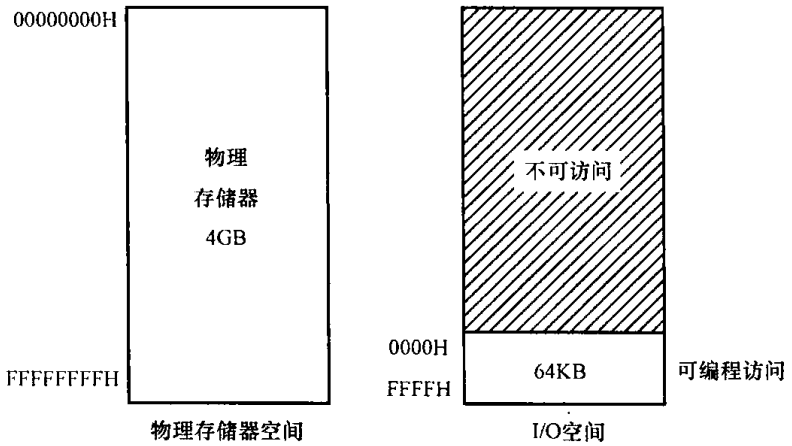


图 2.26　80486 微处理器存储器和 I/O 空间

2.3.5　80486CPU 的引脚及其功能

Intel 80486 CPU 采用 PGA 封装形式，共有 168 个引脚信号，其中包括 30 个地址引脚信号、32 个数据引脚信号、35 个控制引脚信号、24 个 V_{CC}引脚信号、28 个 V_{SS}引脚信号和 19 个空脚，如图 2.27 所示。

1. 地址线类（A_2～A_3，A_4～A_{31}，$\overline{BE_0}$，$\overline{BE_1}$，$\overline{BE_2}$，$\overline{BE_3}$）

（1）A_{31}～A_2(Address Lines)：32 位地址总线，三态，输出。

（2）$\overline{BE_3}$～$\overline{BE_0}$(Byte Enable)：字节允许信号，低电平有效。

A_2～A_3 与$\overline{BE_3}$～$\overline{BE_0}$形成完整的地址线，可访问 4GB 内存空间和 64KB I/O 端口空间。4GB 的内存空间分为 4 个 1GB 的存储体，每个存储体分别由字节允许信号$\overline{BE_3}$～$\overline{BE_0}$选通。当$\overline{BE_i}$(i=0～3) 有效时，选择相应的存储体。寻址 I/O 空间时，只有 A_{15}～A_2 和$\overline{BE_i}$(i=0～3)有效，可寻址 64KB 的 I/O 空间。

A_2～A_{31}用来确定一个 4 字节的存储单位；$\overline{BE_3}$～$\overline{BE_0}$则用来确定在当前的操作中所涉及到 4 字节存储单位中的那些字节。80486 微处理器规定：$\overline{BE_0}$对应于数据线 D_0～D_7；$\overline{BE_1}$对

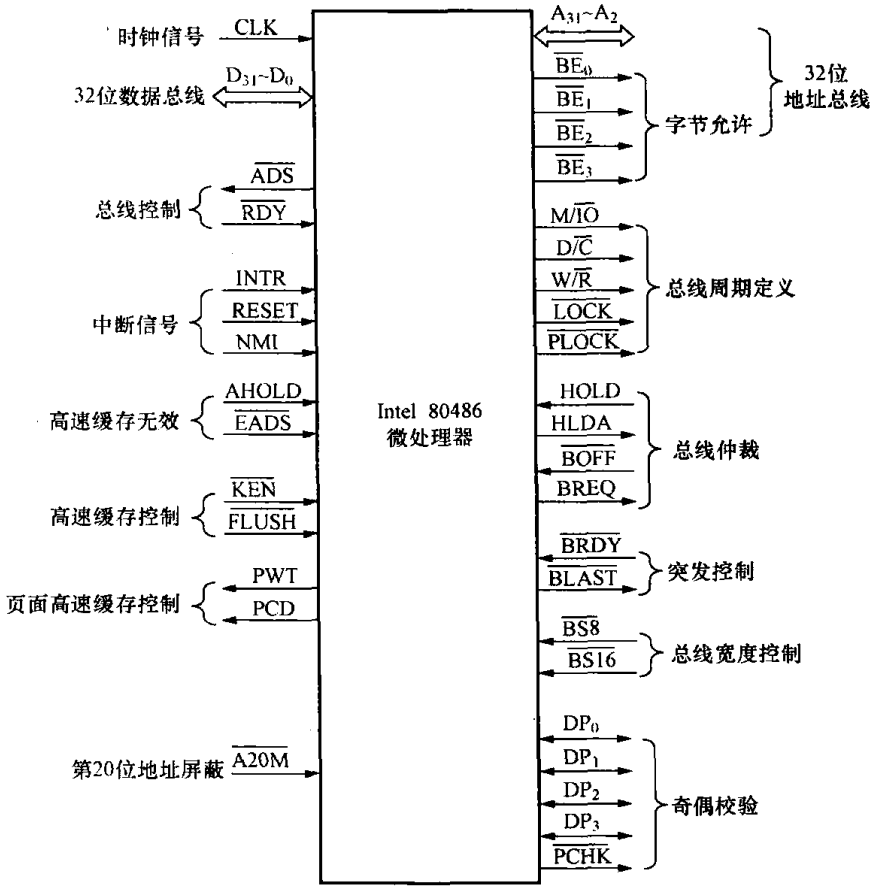


图 2.27 80486 外部引脚

应于 $D_8 \sim D_{15}$；$\overline{BE_2}$对应于 $D_{16} \sim D_{23}$；$\overline{BE_3}$对应于 $D_{24} \sim D_{31}$。

80486 微处理器的$\overline{BE_3} \sim \overline{BE_0}$与 A_0、A_1 引脚和 32 位数据总线的关系，如表 2.10、表 2.11 所示。当$\overline{BE_0}$（对应于 $D_0 \sim D_7$）有效时，A_0 为“0”；当$\overline{BE_1}$（对应于 $D_8 \sim D_{15}$）有效时，$\overline{BHE}$为“0”，这就与 16 位的 8086 完全兼容了。

2. 数据线类（$D_0 \sim D_{31}$）

80486 微处理器 32 位数据线可分为 4 个字节，其中 $D_0 \sim D_7$ 定义为最低字节，$D_{24} \sim D_{31}$ 定义为最高字节。数据线具有双向和三态特性。

3. 控制线类

根据引脚的具体作用，80486 微处理器的控制线可以分为多组，下面对部分控制引脚进行介绍：

（1）时钟（CLK），输入。CLK 为 80486 微处理器提供基本的定时和内部的工作频率，该引脚的输入信号应具有 TTL 电平。所有外部的定时参数都是以 CLK 的上升沿为基准而确定的。

（2）数据校验位组（$DP_3 \sim DP_0$，PCHK），双向。

$DP_3 \sim DP_0$(Data Parity)：数据奇偶校验信号，双向。$DP_3 \sim DP_0$ 分别对应 32 位数据中字节 3～字节 0 的校验位。当数据写入存储时，由 $DP_0 \sim DP_3$ 自动地对每个字节加入偶检验位（有偶数个“1”为高电平）；在从存储器读出数据时，也自动地对每个字节的数据进行偶校验。

$\overline{PCHK}$是“奇偶检验状态”信号引脚，单向输出。当奇偶检验出错时，该引脚是低电平。

（3）数据总线宽度控制（$\overline{BS16}$，$\overline{BS8}$），输入。

$\overline{BS8}$为 8 位数据总线宽度定义信号。$\overline{BS8}$为低电平时，规定数据总线中传输 8 位数据。

$\overline{BS16}$为 16 位数据总线宽度定义信号。$\overline{BS16}$为低电平时，规定数据总线中传输 16 位数据。

在上述情形下，80486 微处理器将把比较长的数据换为相对宽度比较小的数据多次传送，$\overline{BE_3}$～$\overline{BE_0}$也会作出相应的变化，以适应所确定的数据线宽度。

（4）总线周期定义组（W/$\overline{R}$，D/$\overline{C}$，M/$\overline{IO}$，$\overline{LOCK}$，$\overline{PLOCK}$），单向输出。微处理器与存储器或 I/O 之间交换一个数据的时间称为总线周期。80486 微处理器的一个总线周期至少由两个 CLK（时钟周期）组成。

W/$\overline{R}$（写或读）、D/$\overline{C}$（数据/控制）和 M/$\overline{IO}$（存储器/输入输出）是总线周期定义的基本信号。其中，D/$\overline{C}$(Data/Control) 称为数据/控制信号，D/$\overline{C}$=1，表示数据传输周期；D/$\overline{C}$=0，表示指令代码传输周期。这三个信号的不同组合可以决定当前的总线周期所要完成的操作，如表 2.10 所示。

表 2.10　　总线周期定义的操作

M/$\overline{IO}$	D/$\overline{C}$	W/$\overline{R}$	总线周期类型	M/$\overline{IO}$	D/$\overline{C}$	W/$\overline{R}$	总线周期类型
0	0	0	中断响应	1	0	0	读存储器代码
0	0	1	空闲	1	0	1	停止/关机
0	1	0	读 I/O 数据	1	1	0	读存储器数据
0	1	1	写 I/O 数据	1	1	1	写存储器数据

$\overline{LOCK}$是“总线锁定”引脚，低电平有效。当$\overline{LOCK}$为低电平时，表示当前的总线周期被锁定，此时，80486CPU 独占系统总线。该信号由 LOCK 指令前缀设置或存储器操作时自动锁定。

$\overline{PLOCK}$引脚是“锁定”信号。当它有效时，表示 CPU 需要多个总线周期才能完成传输。该信号主要用于浮点运算和对 Cache 的操作。

（5）总线控制组（$\overline{ADS}$，RDY）。总线控制信号指明总线周期从何时开始，到何时结束。

$\overline{ADS}$引脚是地址选通信号，输出，低电平有效。当$\overline{ADS}$为低电平时，表明地址总线上已输出有效地址。

$\overline{RDY}$引脚是准备就绪信号，输入，低电平有效。当$\overline{RDY}$为低电平时，表示现行总线周期已经完成。对于读周期时，该信号有效表示数据线上已存在有效的数据；对于写周期时，该信号有效表示外部已接收了 80486 微处理器发出的信息。

（6）总线仲裁信号组（HOLD，$\overline{HLDA}$，$\overline{BOFF}$，BREQ）。

HOLD(Bus Hold Request)：总线请求信号，输入，高电平有效。HOLD 信号由另一个总线主控设备产生，请求 CPU 让出对总线的控制权。

$\overline{HLDA}$(Hold Acknowledge)：总线请求响应信号，输出，高电平有效。$\overline{HLDA}$信号是对 HOLD 信号的应答，$\overline{HLDA}$为高电平时，表示 CPU 已让出对总线的控制权。

BREQ(Bus Request)：内部总线请求信号，输出，高电平有效。当 BREQ 为高电平时，表示 CPU 内部提出了一个总线请求，此时 CPU 正在控制总线。

$\overline{\text{BOFF}}$(Back Off)：强制 CPU 放弃系统总线信号，输入，低电平有效。当 CPU 接收到该信号时，便立即放弃对系统总线的控制权，并使其所有引脚处于浮空状态。

值得指出的是，在总线控制权交出后，80486 微处理器仍然可运行程序，这是因为装入片内 Cache 的程序中许多操作都能满足无需访问系统总线的要求。

(7) 突发控制组（$\overline{\text{BRDY}}$，$\overline{\text{BLAST}}$）。除基本总线周期外，80486 微处理器还具有突发总线周期。所谓突发就是在总线周期期间，交换第 1 个信息至少需要两个时钟，此后的每一个时钟都可交换一个信息。因此在突发总线周期时可以比基本总线周期更快地交换信息。

$\overline{\text{BRDY}}$是“突发准备好”引脚，在突发周期，它与$\overline{\text{RDY}}$具有同样的功能。微处理器在突发周期的第 2 个时钟及以后的每个时钟都对其进行采样，以决定是否与外部交换信息。

$\overline{\text{BLAST}}$引脚是“突发周期结束”信号，单向输出。当此信号有效时，下一个$\overline{\text{BRDY}}$出现就表示突发总线周期结束。

注意：如果$\overline{\text{BRDY}}$和$\overline{\text{RDY}}$同时出现，则$\overline{\text{BRDY}}$被忽略，且提前结束突发周期。

(8) 地址屏蔽引脚（$\overline{\text{A20M}}$）。第 20 位地址屏蔽信号，输入，低电平有效。当$\overline{\text{A20M}}$有效时，将屏蔽 A_{20}及以上地址，使 80486CPU 仿真 8086CPU 的 1MB 存储器地址。只有 CPU 工作在实模式下，$\overline{\text{A20M}}$才有意义。

(9) 系统复位引脚（RESET）。RESET(Reset) 复位信号，输入，高电平有效。当RESET有效时，系统复位。复位后，微处理器内部寄存器的值，如表 2.11 所示。

表 2.11　　复位后寄存器的值

寄存器	初始值	寄存器	初始值
EAX	不定	ES	0000H
EBX	不定	CS	F000H
ECX	不定	SS	0000H
EDX	0400H+版本 ID	DS	0000H
EBP	不定	FS	0000H
ESP	不定	GS	0000H
EDI	不定	IDTR	基址=0，界限=3FFH
ESI	不定	CR0	6000～0000H
EFLAGS	0000～0002H	DR7	0000～0000H
EIP	OFFFOH	浮点寄存器	不变

(10) 中断请求信号。

INTR(Maskable Interrupt)：可屏蔽中断请求信号，输入，高电平有效。

NMI(Non-maskable Interrupt)：非屏蔽中断请求信号，输入，上升沿有效。

上述介绍了 80486 微处理器的大部分引脚，其他引脚功能请自行参考有关 80486CPU 的技术手册。

2.4 Pentium 微处理器

Pentium 微处理器如同 80486 CPU 一样是 32 位微处理器，其内部主要功能部件为 32 位，内部关键数据通路为 64 位。为了提高与片外主存储器的数据交换速率，Pentium 微处

理器对外的数据总线宽度为 64 位，对外数据传输率在 66MHz 主频下可达 528MB/s，Pentium 微处理器地址总线仍然设计为 32 位。

Pentium 微处理器的超标量结构使得在每个时钟周期内可以同时执行两条整型指令，它将芯片内的 8KB 数据 Cache 与 8KB 指令 Cache 分开，以及分支预测和流水线浮点单元等设计都大大提升了系统性能，从而使全球的微处理器技术步入了一个崭新的 Pentium 时代。

2.4.1 Pentium 微处理器的体系结构

图 2.28 所示为 Pentium 微处理器的内部结构框图。Pentium 微处理器与 80486 相比，其最突出的特点简介如下。

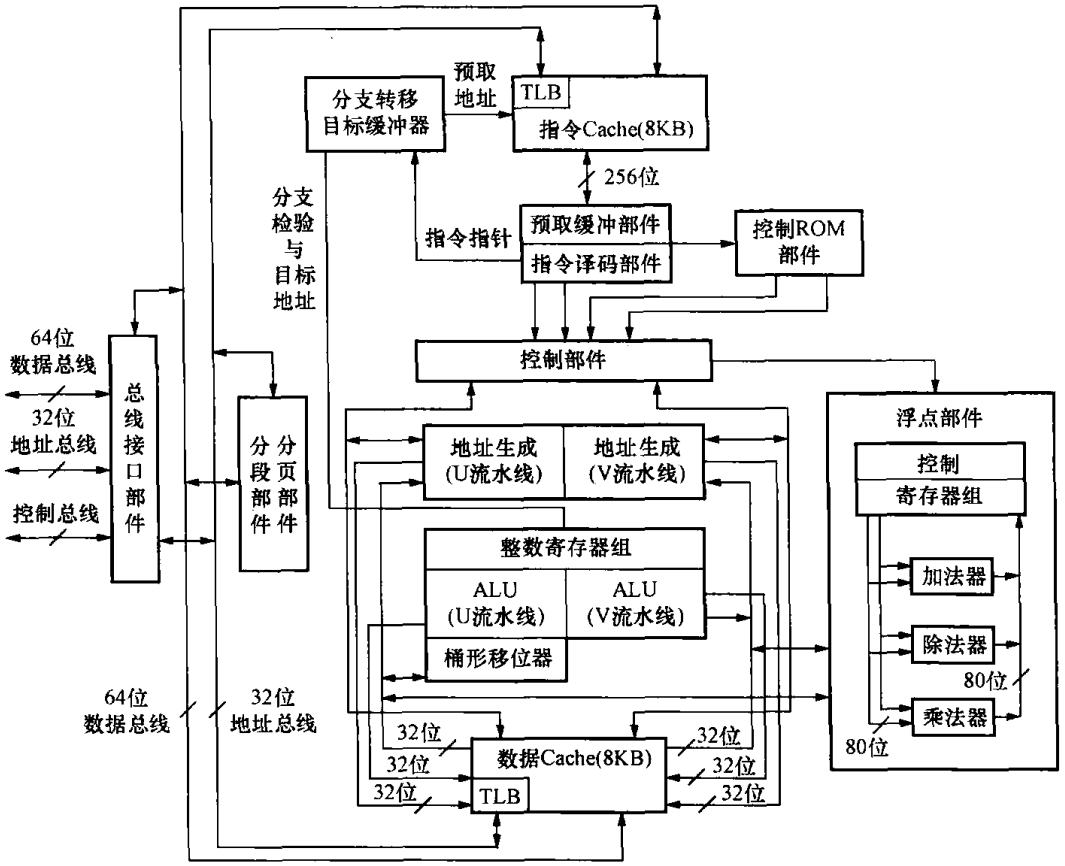


图 2.28 Pentium 微处理器的内部结构框图

1. 超标量流水线

80486 微处理器只有一条流水线（Pipelines），而 Pentium 微处理器具有 U、V 两条流水线，每个时钟周期能同时执行两条整型指令。超标量流水线（Super-Pipelines）设计是 Pentium 微处理器新技术的核心。

Pentium 微处理器把生成地址的硬件和算术逻辑运算部件配置成两个各自独立的整数流水线，一个叫 U 流水线，另一个叫 V 流水线。U 流水线可以执行 Intel 体系结构的全部整型与浮点指令，包括微代码构成的复杂指令。而 V 流水线则只能执行简单的整型指令和浮点部件的 FXCHG 指令。

2. 独立的指令 Cache 和数据 Cache

80486 微处理器片内只有 8KB Cache，Pentium 微处理器则有 16KB，且将指令 Cache 与数据 Cache 完全分开，各为 8KB，这样就完全避免了预取指令与数据两者之间的冲突。而且指令

Cache 与数据 Cache 都有各自的转换后援缓冲器 TLB，因而存储器管理部件 MMU(Memory Management Unit) 中的分页部件就能迅速地将代码或数据的线性地址转换成物理地址。

3. 全新的浮点部件

Pentium 微处理器的浮点部件相对 80486 微处理器有了彻底的改进，它的浮点流水线 U 有八级，前五级与 V 流水线共享，后三级则有自己独立的浮点流水线，每个时钟周期可以执行一条浮点指令。

4. 分支转移的动态预测

所谓分支预测，是指当 CPU 遇到无条件或有条件转移指令、CALL 调用指令、RET 返回指令、INT n 中断调用，以及中断返回指令 IRET 等跳转指令时，指令预取单元能够较准确地判断是发生转移取指，还是依据 EIP 指针顺序往下取指。

80486 微处理器的指令流水线没有分支预取功能，它不能将转移指令与其他指令区别开来，预取单元只能依据 IP/EIP 指针顺序地取出下一条指令的代码送入预取队列，因此取出跳转指令后也总是继续读取下一条指令。当轮到执行阶段时，如果真的不发生跳转，则流水线上不会发生任何问题，继续操作下去即可。倘若发生跳转，那么预取队列中已经装入的指令，以及前几个流水线阶段的工作都要“作废”，必须清除并重新装入转移地址处的指令，这样就使流水线操作出现了一次“返工”，影响了整机操作的效率。

Pentium 微处理器借助跳转目标缓冲器 BTB(Branch Target Buffer) 等逻辑部件实现了分支转移的动态预测。BTB 是一个具有 256 行的四路组合相关映射高速 Cache，以跳转的 32 位目标地址、两位历史状态及一位有效状态作为一个 Cache 存储的内容。被预取的指令送入 U 和 V 两条流水线，同时将指令所在的 EIP 地址送入 BTB 中进行查找比较。如果在 BTB 中没有这个地址，就不进行预测。倘若在 BTB 中找到了该地址，即为命中，那么微处理器就将根据 BTB 中对应记录的历史状态来预测当前是否发生跳转，这样分支预测逻辑就能比较可靠地预测分支走向，指令预测队列就可以预取较多的指令。当执行调转指令时，其跳转的目标地址就将用于更新 BTB 中相应的地址记录。Pentium 微处理器的分支转移动态预测功能，使得主流水线不会空闲，而且大大加速了程序的执行。

5. 64 位外部数据总线

Pentium 微处理器的外部数据总线经总线接口部件扩展到了 64 位，该接口电路与内部高速缓冲存储器 Cache 连接，因而外部数据与指令的传输速率比 80486 微处理器要高很多，能有效地解决外部总线上的瓶颈问题。

2.4.2 Pentium 微处理器寄存器结构

Pentium 微处理器与 80486 微处理器的寄存器一样，都包括整数部件中的基本寄存器、系统寄存器、调试和测试寄存器以及浮点寄存器，而且功能基本相同。只是系统寄存器中增加了 CR4 控制寄存器和 TR1、TR2 测试寄存器。另外，在 Pentium 微处理器的标志寄存器中又增加了三个新的功能位。下面只对新增加的寄存器和功能位加以简单介绍。

1. CR4 控制寄存器

CR4 是 Pentium 微处理器在 80486 微处理器基础上新增加的控制寄存器，包括七个控制位，格式如图 2.29 所示。

D_{31} … D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
保留	MCE	PAE	PSE	DE	TSD	PVI	VME

图 2.29 Pentium 控制寄存器 CR4 格式

VME(Virtual 8086 Mode Extension)：虚拟 8086 模式扩展位。

VME=1，允许虚拟 8086 扩展，即允许 8086 模式和虚拟 8086 中断。VME=0，禁止虚拟 8086 模式扩展。

PVI(Protected Mode Virtual Interrupts)：保护模式虚拟中断位。PVI=1，允许保护模式虚拟中断。PVI=0，禁止保护模式虚拟中断。

TSD(Time Stamp Disable)：禁止 RDTSC 指令位。TSD=0，则允许 RDTSC（读时间标志计算器）指令在任何特权级上执行。TSD=1，仅允许 RDTSC 指令在 0 级特权级上执行，否则将发生一般保护模式异常。

DE(Debugging Extensions)：调试扩展位。DE=1，允许输入/输出断点。DE=0，不支持输入/输出断点。

PSE(Page Size Extensions)：允许页容量大小扩展位。PSE=1，允许每页容量为 4MB。PSE=0，只允许每页容量为 4KB。

PAE(Physical Address Extension)：允许物理地址扩展位。PAE=1，允许采用 32 位以上的物理地址（包括 32 位和 64 位地址）。PAE=0，只允许采用 32 位物理地址。

MCE(Machine Check Exception)：允许机器检查异常位。MCE=1，允许机器检查异常。MCE=0，不允许机器检查异常。

2. 测试寄存器 TR

Pentium 微处理器在 80486 微处理器五个测试寄存器（TR7～TR3）基础上新增加了两个测试寄存器 TR1 和 TR2。TR1 用于 Pentium 微处理器的奇偶校验测试，TR2 作为指令 Cache 结束位测试。Pentium 微处理器中 TR7～TR3 的功能与 80486 微处理器的完全相同。

新增加的三个标志位分别是：

VIF(Virtual Interrupt Flag)：虚拟中断标志位。在虚拟 8086 模式下，VIF 是中断标志位 IF 的副本，与 VIP 一同使用。实模式下不使用 VIF 标志。

VIP(Virtual Interrupt Pending)：虚拟中断挂起标志位。VIP 用于多任务环境下，为操作系统提供虚拟中断标志和中断挂起信息。VIP=1，表明有中断被挂起。VIP=0，表明没有中断被挂起。VIP 与 VIF 一同使用，使得在虚拟 8086 模式下具有中断标志位 IF 的虚拟方式。这样处理后，中断过程会明显加速。实模式下不使用 VIP 标志。

ID(Identification)：识别标志位。ID 指示 Pentium 微处理器是否支持 CPUID 指令。ID=1，则支持 CPUID 指令。执行 CPUID 指令，能够读取 CPU 的有关信息。

2.4.3 Pentium 微处理器的引脚及功能

Pentium 微处理器采用 PGA 封装形式，共有 237 个引脚信号，其中包括 29 个地址引脚信号，64 个数据引脚信号，75 个控制引脚信号，69 个 V_{CC}、V_{SS}和 NC 空脚。其按功能可分为地址总线、数据总线和控制总线三大类。其中，控制总线又分为总线周期定义、总线控制、总线仲裁、高速缓存、中断、错误检测和系统管理模式等引脚信号。Pentium 微处理器的引脚信号如图 2.30 所示。

1. 时钟信号

CLK(clock)：时钟信号，为 CPU 提供基本的定时信号。

2. 地址总线（Address Bus）

A_{31}～A_3(Address)：地址总线，三态、输出，用于定义存储器和 I/O 端口地址。

$\overline{BE_7}$～$\overline{BE_0}$(Byte Enable)：字节允许信号，低电平有效。

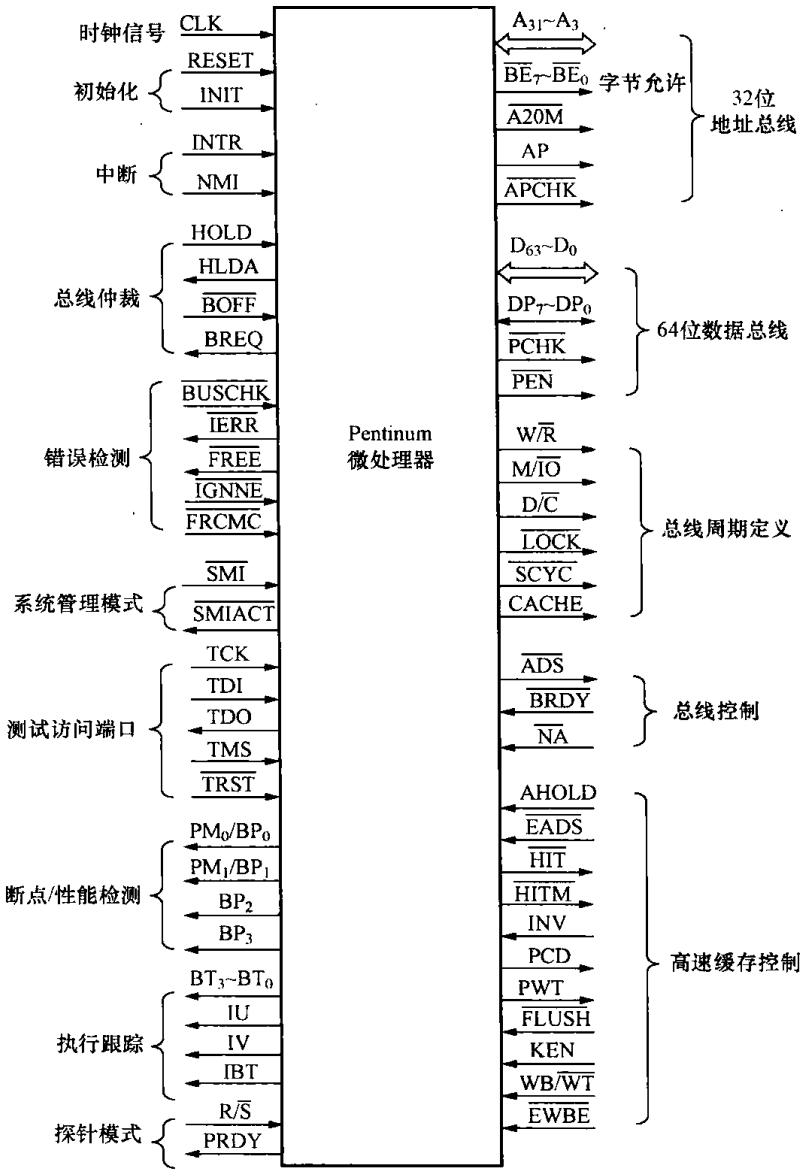


图 2.30 Pentium 微处理器的引脚信号

A_{31}～A_3 和$\overline{BE_7}$～$\overline{BE_0}$构成 32 位地址总线，可寻址 4GB 的内存空间和 64KB 的 I/O 空间。4GB 的内存空间分为八个 512MB 的存储体，每个存储体分别由字节允许信号$\overline{BE_7}$～$\overline{BE_0}$选通，当$\overline{BE_i}$(i=0～7) 有效时，选择相应的存储体，然后由 A_{31}～A_3 选择相应的字节进行读/写操作。寻址 I/O 空间时，只有 A_{15}～A_3 和$\overline{BE_i}$(i=0～7) 有效，寻址 64KB 的 I/O 空间。

$\overline{A20M}$(Address Bit 20 Mask)：第 20 位地址屏蔽信号，输入，低电平有效。当$\overline{A20M}$有效时，将屏蔽 A_{20} 及以上地址，使 Pentium 微处理器仿真 8086CPU 的 1MB 存储器地址。$\overline{A20M}$只有在 CPU 工作在实模式下才有意义。

AP(Address Parity)：地址奇偶校验位，双向，高电平有效，指示地址总线 A_{31}～A_3 上偶检验信息。

$\overline{APCHK}$(Address Parity Check)：地址奇偶位检测，输出，低电平有效，指示微处理器检测到询问周期地址总线奇偶错。Pentium 微处理器对地址总线增加了奇偶校验功能，它输

出的地址信号 A_{31}～A_3 会产生一个奇偶校验位，在 AP 信号线上输出，存储器子系统可据此对地址进行校验。在询问周期中，要对请求的地址进行校验操作，分别由$\overline{EADS}$和 AP 输入外部地址有效和奇偶校验信息。如果发现奇偶校验错误，则在$\overline{APCHK}$信号线上输出低电平。

3. 数据总线（Data Bus）

D_{63}～D_0(Data Lines)：64 位双向数据总线，可以传输 8 位、16 位、32 位和 64 位数据。

DP_7～DP_0(Data Parity)：数据奇偶校验信号，双向。DP_7～DP_0 分别对应 64 位数据中字节 7～字节 0 的校验位，即 DP_7 对应 D_{63}～D_{56}，DP_0 对应 D_7～D_0。

$\overline{PCHK}$(Parity Check)：奇偶校验状态信号，输出，低电平有效。当$\overline{PCHK}$低电平时，表示 CPU 在上一个读周期采样的数据奇偶校验出错。

$\overline{PEN}$(Parity Enable)：奇偶校验允许信号，输入，低电平有效，用来规定在发生校验时，是否进行异常处理。若$\overline{PEN}$为低电平，且控制寄存器 CR4 的 MCE 位为 1，则出现校验错误时，微处理器将会自动执行异常处理。

4. 总线周期定义（Bus Cycle Definition）

读/写信号 W/$\overline{R}$、存储器或 I/O 访问信号 M/$\overline{IO}$、数据/控制信号 D/$\overline{C}$、总线锁定信号$\overline{LOCK}$与 80486 微处理器中这些信号功能相同。

SCYC(Split Cycle)：分隔周期信号，输出，高电平有效，用于总线锁定周期，表示未对齐操作锁定周期期间有两个以上的周期被锁定。

$\overline{CACHE}$(Cache)：高速缓存信号，输出，低电平有效。在读总线周期，$\overline{CACHE}$有效表明从存储器读出的数据可送入高速缓冲存储器。当$\overline{CACHE}$为高电平时，即使 Cache 控制信号$\overline{KEN}$有效，对读回的数据也不进行缓存。在写总线周期，$\overline{CACHE}$有效表示写操作对片内高速缓冲存储器中被修改的数据执行突发式回写操作。由 W/$\overline{R}$、M/$\overline{IO}$、D/$\overline{C}$、$\overline{CACHE}$和$\overline{KEN}$信号定义的总线周期，如表 2.12 所示。

表 2.12　　Pentium 微处理器总线周期

M/$\overline{IO}$	D/$\overline{C}$	W/$\overline{R}$	$\overline{CACHE}$	$\overline{KEN}$	总线周期类型	传送次数
0	0	0	1	×	中断确认（两个锁定周期）	每周期 1 次
0	0	1	1	×	专用周期	1
0	1	0	1	×	I/O 读，32 位、16 位、8 位，非高速缓冲	1
0	1	1	1	×	I/O 写，32 位、16 位、8 位，非高速缓冲	1
1	0	0	1	×	读代码，64 位非高速缓冲	1
1	0	0	×	1	读代码，64 位高速缓冲	1
1	0	0	0	0	读代码，256 位，组成 Cache 行填充	4
1	0	1	×	×	Intel 保留	
1	1	0	1	×	存储器读，64 位、32 位、16 位、8 位非高速缓冲	1
1	1	0	×	1	存储器读，64 位、32 位、16 位、8 位高速缓冲	1
1	1	0	0	0	存储器读，256 位，组成 Cache 行填充	4
1	1	1	1	×	存储器写，64 位、32 位、16 位、8 位非高速缓冲	1
1	1	1	0	×	256 位成组回写	4

5. 总线控制（Bus Control）

Pentium微处理器中地址选通信号$\overline{\text{ADS}}$、突发就绪信号$\overline{\text{BRDY}}$与80486微处理器中这些信号功能相同。

$\overline{\text{NA}}$(Next Address)：下一个地址信号，输入，低电平有效，用于形成流水线式总线周期。$\overline{\text{NA}}$为低电平有效时，表明即使当前总线周期还没有完成，外部存储系统已经准备就绪，将下一个地址输出到总线上，用以开始一个新的总线周期。

6. 高速缓存控制（Cache Control）

Pentium微处理器中的地址保持请求信号AHOLD、页面通写控制信号PWT、页面高速缓存禁止信号PCD、高速缓存允许信号$\overline{\text{KEN}}$、高速缓存清除信号$\overline{\text{FLUSH}}$与80486微处理器中这些信号功能相同。

$\overline{\text{EADS}}$(External Address)：外部地址有效信号，输入，低电平有效。当$\overline{\text{EADS}}$有效时，产生一个询问周期。

WB/$\overline{\text{WT}}$(Write Back/Write Through)：回写/通写方式信号，输入。WB/$\overline{\text{WT}}$为高电平，对片内数据Cache行采用回写方式，WB/$\overline{\text{WT}}$为低电平，则采用通写方式。

$\overline{\text{HIT}}$(Hit)：询问周期命中信号，输出，低电平有效。

$\overline{\text{HITM}}$(Hit to a Modified Line)：命中数据Cache的修改行信号，输出，低电平有效。

INV(Invalidation)：无效请求信号，输入，高电平有效，在询问周期被命中时，用来确定最后Cache行是无效状态还是共享状态。

$\overline{\text{HIT}}$、$\overline{\text{HITM}}$和INV信号用于询问周期。询问周期的作用是通过$\overline{\text{EADS}}$引脚查询数据Cache和指令Cache，以检查现行地址是否命中Cache，若命中，则$\overline{\text{HIT}}$输出低电平。若该地址命中Cache中一个已经修改过的行，则$\overline{\text{HITM}}$输出低电平，然后将已经修改的行写回到总线上。当INV引脚输入高电平时，将使当前Cache行的下一个状态变为无效状态（即该行不可以使用）。

$\overline{\text{EWBE}}$(External Write Buffer Empty)：外部写缓冲器空信号，输入，低电平有效。当$\overline{\text{EWBE}}$有效时，表明在外部系统中有一个通写周期被挂起，因而将停止写入数据Cache行。

7. 初始化（Initialization）

RESET：复位信号，输入，高电平有效。当RESET有效时，内部Cache全部无效，同时，在数据Cache中的修改行不再回写。RESET的基本功能与80486微处理器的RESET信号相同。

INIT(Initialization)：初始化引脚，输入，高电平有效。INIT的作用与RESET的作用类似，不过，它在进行微处理器初始化时，将保持片内Cache、写缓冲器和浮点寄存器的内容不变，所以INIT不能取代RESET系统上电后的复位工作。当RESET信号由高电平变为低电平时，如果INIT为高电平，则Pentium在程序执行前进行内部自检测。

8. 中断请求（Interrupts）

可屏蔽中断请求INTR信号、非屏蔽中断请求NMI信号与80486微处理器中这些信号功能相同。

9. 总线仲裁（Bus Arbitration）

Pentium微处理器中的总线请求信号HOLD、总线请求响应信号$\overline{\text{HLDA}}$、内部总线请求信号BREQ、强制CPU放弃系统总线信号$\overline{\text{BOFF}}$与80486微处理器中这些信号功能相同。

10. 错误检测（Error Reporting）

$\overline{\text{FREE}}$(Floating-point Error)：浮点错误报告，输出。当系统产生非屏蔽浮点错误中断时，$\overline{\text{FREE}}$引脚输出低电平。

$\overline{\text{IGNNE}}$(Ignore Numeric Error)：忽略数字错误，输入，低电平有效。当$\overline{\text{IGNNE}}$引脚为低电平时，CPU 将忽略已经被挂起的非屏蔽数字异常，而继续执行浮点指令。

$\overline{\text{BUSCHK}}$(Bus Check)：总线周期检测，输入，低电平有效。用于检测总线周期是否正常结束。当$\overline{\text{BUSCHK}}$引脚输入为低电平时，则将错误的总线周期类型和地址锁存到系统寄存器中，若此时控制寄存器 CR4 中的 MCE 位为 1，系统则自动执行异常处理程序。

$\overline{\text{FRCMC}}$(Function Redundancy Checking)：功能冗余检测，输入，低电平有效。

$\overline{\text{IERR}}$(Internal Error)：内部出错指示，输出，低电平有效。如果在读周期产生奇偶错误，系统将强制$\overline{\text{IERR}}$输出一个时钟周期的低电平。

11. 系统管理模式（System Management Mode）

$\overline{\text{SMI}}$(System Management Interrupt)：系统管理中断请求，输入，低电平有效。当$\overline{\text{SMI}}$有效时，进入系统管理模式。

$\overline{\text{SMACT}}$(System Management Interrupt Active)：启动系统管理模式工作，输出，低电平有效，指示 Pentium 进入系统管理模式。

系统管理模式主要用于实现系统电源管理功能，进入该模式后，处理器的状态被保存起来，并可以有单独的存储空间。执行 RSM 指令，则退出系统管理模式。

12. 测试访问端口（Tap Port）

TCK(Testability Clock)：测试时钟，输入，为系统的边界扫描提供时钟信号。

TDI(Test Data Input)：测试数据输入引脚，输入。

TDO(Test Data Output)：测试数据输出引脚，输出。

TMS(Test Mode Select)：测试模式选择，输入。

$\overline{\text{TRST}}$(Test Rest)：测试复位引脚，输入。

13. 断点/性能检测（Breakpoint/Performance Monitoring）

BP_3～BP_0(Breakpoint)：断点匹配检测，输出，高电平有效。BP_3～BP_0 与调试寄存器 DR3～DR0 相对应，当调试寄存器编程设置为断点匹配测试时，若 BP3～BP0 输出为高电平，则表示断点匹配。

PM_1，PM_0(Performance Monitoring)：性能监测，输出高电平有效，与 BP_1 和 BP_0 多路复用，由调试模型寄存器（Debug Mode Control Register）中的 PB_1 和 PB_0 位来确定它们是用于断点匹配 BP_1，BP_0 还是性能监测 PM_1，PM_0。

14. 执行跟踪

BT_3～BT_0(Branch Trace)：分支跟踪，输出。在分支跟踪的特殊周期，BT_3～BT_0 提供分支目标的线性地址的位 2～位 0，BT_3 代表特定的操作尺度。

IU(Instruction U-pipeline)：U 流水线指令执行完成，输出。

IV(Instruction V-pipeline)：V 流水线指令执行完成，输出。

IBT(Instruction Branch Trace)：指令分支发生，输出。

15. 探针模式（Probe Mode）

R/$\overline{\text{S}}$(Run/Stop)：异步边沿中断请求，输入，停止 CPU 正常的指令执行，并使它进入

空闲状态。R/$\overline{S}$引脚上由高电平到低电平的跳变，将引起中断，使 CPU 在下一条指令的边界停止执行程序。

PRDY(Probe Ready)：中断响应信号，输出，高电平有效，PRDY 输出高电平是对 R/$\overline{S}$中断请求的响应，指示 CPU 停止正常的程序执行，或者是进入允许用户检查并修改 CPU 的状态。

2.4.4 Pentium 微处理器的存储器组织及 I/O 结构

1. *存储器组织与 I/O 结构*

Pentium 微处理器由 A_{31}～A_3 和$\overline{BE_7}$～$\overline{BE_0}$构成 32 位地址总线，可寻址 4GB（2^{32}）的存储器空间，地址范围是 00000000H～FFFFFFFFH。32 条地址总线中的低 16 位地址 A_{15}～A_3 和$\overline{BE_7}$～$\overline{BE_0}$，用作对 64KB I/O 端口的寻址，地址范围是 0000H～FFFFH。Pentium 微处理器的存储器组织及 I/O 地址空间与 80486 微处理器完全相同。

2. *存储器寻址*

Pentium 微处理器的地址总线 A_{31}～A_3 与字节允许信号$\overline{BE_7}$～$\overline{BE_0}$共同形成 32 位地址，寻址 4GB 的存储器空间。地址总线 A_{31}～A_3 与字节允许信号$\overline{BE_7}$～$\overline{BE_0}$的作用，如表 2.13 所示。字节允许信号$\overline{BE_7}$～$\overline{BE_0}$与 64 位数据总线对应情况，如表 2.14 所示。

表 2.13　　A_{31}～A_3 与$\overline{BE_7}$～$\overline{BE_0}$的作用

Pentium 微处理器地址信号											
A_{31}～A_3 物理地址				$\overline{BE_7}$	$\overline{BE_6}$	$\overline{BE_5}$	$\overline{BE_4}$	$\overline{BE_3}$	$\overline{BE_2}$	$\overline{BE_1}$	$\overline{BE_0}$
A_{31}…A_3	A_2	A_1	A_0								
A_{31}…A_3	0	0	0	×	×	×	×	×	×	×	0
A_{31}…A_3	0	0	1	×	×	×	×	×	×	0	1
A_{31}…A_3	0	1	0	×	×	×	×	×	0	1	1
A_{31}…A_3	0	1	1	×	×	×	×	0	1	1	1
A_{31}…A_3	1	0	0	×	×	×	0	1	1	1	1
A_{31}…A_3	1	0	1	×	×	0	1	1	1	1	1
A_{31}…A_3	1	1	0	×	0	1	1	1	1	1	1
A_{31}…A_3	1	1	1	0	1	1	1	1	1	1	1

表 2.14　　$\overline{BE_7}$～$\overline{BE_0}$与 64 位数据总线对应情况表

字节允许信号	数据总线信号		字节允许信号	数据总线信号	
$\overline{BE_0}$	D_7～D_0	字节 0（最低位）	$\overline{BE_4}$	D_{39}～D_{32}	字节 4
$\overline{BE_1}$	D_{15}～D_8	字节 1	$\overline{BE_5}$	D_{47}～D_{40}	字节 5
$\overline{BE_2}$	D_{23}～D_{16}	字节 2	$\overline{BE_6}$	D_{55}～D_{48}	字节 6
$\overline{BE_3}$	D_{31}～D_{24}	字节 3	$\overline{BE_7}$	D_{63}～D_{56}	字节 7（最高位）

由于 Pentium 微处理器数据总线为 64 位，所以存储器和 I/O 地址空间都是针对 64 位数据宽度来组织的，如图 2.31 所示。

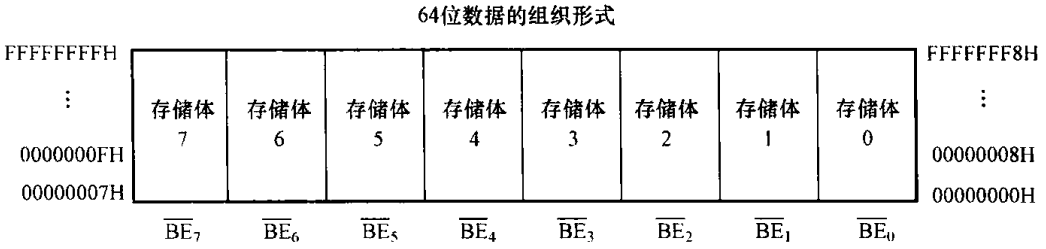


图 2.31 Pentium 微处理器存储器组织

有关 Pentium 微处理器的其他详细内容请参考相关书籍。

2.5 总 线

2.5.1 总线概述

总线是各种信号线的集合，是计算机各部件之间传输数据、地址和控制信息的公共通道。总线由传输信息的物理介质以及一套管理信息传输的通信规则（协议）所组成。总线的特点是“公用性”，即同时挂接多个模块或设备。两个模块或设备之间专用的信号连接线不能称为总线。微机系统采用总线以后，不仅简化了系统硬件的设计过程及系统结构，也减少了软件的设计和调试工作，缩短了软、硬件的研制周期。

1. 微机总线分类

在微机系统中，有各种各样的总线，这些总线可以从不同的层次和角度进行分类。

（1）片总线。又称“元件级总线”、“芯片总线”，是微处理器芯片内部引出的总线。

（2）内总线（I-BUS，系统总线）。又称“系统总线”或“板级总线”，也就是常说的“微机总线”。它是用于微机系统中各插件之间信息传输的通路，是微机系统所特有的。内总线常用的有 STD 总线、MULTIBUS（多总线）线、AT 总线等，32 位微机系统出现以后又推出许多 32 位微机总线，如 MCA、VME 总线、EISA 总线和 PCI 总线等。

（3）外总线（E-BUS）。又称“通信总线”，它是微机系统之间或是微机系统与其他系统（仪器、仪表、控制装置）之间信息传输的通路，往往借用电子工业其他领域已有的总线标准。外总线常用的有 RS-232C、RS449、488 等。

（4）局部总线。局部总线可看作是 CPU 总线和系统总线之间的一种总线。它具有较高的时钟频率和传输率，在一定程度上克服了系统总线的瓶颈问题，提高了系统性能。

局部总线信号独立于 CPU，处理器的更换不会影响系统结构。现在常用的有 EISA 局部总线、PCI 局部总线、AGP 总线 3 种局部总线。

2. 片总线的作用

片总线通常包括地址总线、数据总线和控制总线等三组总线。了解这三组的具体组成、用途及其相互关系，对于解决微机系统的应用及接口问题十分重要。

（1）地址总线。地址总线通常是单向总线，由 CPU 输出。16 位微处理器有 20 条或 24 条地址总线，32 位微处理器一般有 32 条地址总线。地址总线既用于存储器的操作，又用于 I/O 操作。

（2）数据总线。数据总线是双向总线，16 位微处理器有 16 条数据总线，32 位微处理器通常有 32 条数据总线。数据总线用来传送各类数据，由于数据总线的作用是把信息送入

CPU 或从 CPU 送出，所以要求严格地时序控制电路和转接电路（如锁存器、三态器件和各种门电路）加以配合和协调。

通过数据总线可以传送的数据类型为：数值数据、指令码、地址信息、设备码、控制字和状态字。

（3）控制总线。不同型号的微处理器有不同数目的控制总线，且其方向和用途也不一样，但几乎所有的控制总线都与系统的同步功能有关。下面这些控制线是一般微处理器所共有的：

1）读出线和写入线；

2）中断请求线和中断响应线；

3）同步（选通或时钟）信号线；

4）保持、等待就绪（准备好）线。

总之，控制总线用来传送保证计算机同步和协调的定时信号和控制信号，从而保证正确地通过数据总线传送各项信息的操作。

3. 总线的性能指标

（1）总线宽度。总线宽度指的是总线中数据总线的位数，用 bit（位）表示，总线宽度有 8 位、16 位、32 位和 64 位。微计算机的总线宽度一般不会超过 CPU 外部数据总线的宽度。显然，总线的数据传输量与总线宽度成正比。

（2）总线时钟。总线时钟是总线中各种信号的定时标准。总线通常都有一个基本时钟，总线上其他信号都以这个时钟为基准，这个时钟的频率也是总线工作的最高频率。一般来说，总线时钟频率越高，其单位时间内数据传输量越大，但不完全成正比例关系。EISA 总线的时钟频率为 8MHz，PCI 总线为 33.3MHz，PCI-2 总线可达 66MHz。

（3）最大数据传输速率。最大数据传输速率指的是在总线中每秒钟传输的最大字节量，用 MB/s 表示，即每秒多少兆字节。最大数据传输速率有时也被称为带宽（Bandwidth）。

在现代微机中，一般可做到一个总线时钟周期完成一次数据传输，因此，总线的最大数据传输速率为总线宽度除以 8（每次传输的字节数）再乘以总线时钟频率。

例如，PCI 总线的宽度为 32 位，总线时钟频率为 33MHz，则最大数据传输速率为 32÷8×33=132(MB/s)。但有些总线采用了一些新技术（如在时钟脉冲的上升沿和下降沿都选通等），使最大数据传输速率比上面的计算结果高。

总线是用来传输数据的，所采取的各项提高性能的措施，最终都要反映在传输速率上，所以在诸多指标中最大数据传输速率是最重要的。

4. 总线标准

总线标准是对总线的插头及插头座的详细和明确的规范说明，包括几何尺寸、插头针数、各个插针的定义及工作的时序。总线标准为计算机系统（或计算机应用系统）中各个模块的互联提供一个标准接口，按总线标准设计的接口是通用接口。

目前总线标准有两类：一类是 IEEE（美国电气及电子工程师协会）标准委员会定义与解释的标准，如 IEEE-488 总线和 RS-232C 串行接口标准等；另一类是因广泛应用而被大家接受与公认的标准，如 S-100 总线、IBM PC 总线、ISA 总线、EISA 总线、STD 总线接口标准等。不同的总线标准可以用于不同的微机系统或者同一微机系统的不同位置。

总线标准一般包括如下几个部分：

（1）机械结构规范：确定模板尺寸、总线插头、边沿连接器等的规范及位置；

（2）功能规范：确定各引脚信号的名称、定义、功能与逻辑关系，对相互作用的协议（定时）进行说明；

（3）电气规范：规定信号工作时的高低电平、动态转换时间、负载能力以及最大额定值。

随着微机系统的发展，总线在不断地发展和完善。

2.5.2 ISA 总线

ISA(Industrial Standard Architecture) 总线标准是 IBM 公司 1984 年为推出 PC/AT 机而建立的系统总线标准，所以也叫 AT 总线，适用 8/16 位数据总线要求。

ISA 总线的主要性能指标如下：

（1）I/O 地址空间 0100H～03FFH。

（2）24 位地址线可直接寻址的内存容量为 16MB。

（3）数据总线宽度 8/16 位，最高时钟频率 8MHz，最大稳态传输率 16MB/s。

（4）支持 15 级中断。

（5）7 个 DMA 通道结构。

（6）开放式总线结构，允许多个 CPU 共享系统资源。

1. ISA 总线的机械规范

ISA 总线物理示意图如图 2.32 所示。

该插槽有长短两个插口，长插口有 62 个引脚，以 A_1～A_{31} 和 B_1～B_{31} 分列于槽的两面；短插槽有 36 个引脚，以 C_1～C_{18} 和 D_1～D_{18} 分列于槽的两面。

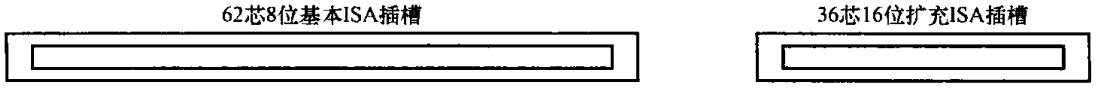


图 2.32 ISA 总线物理示意图

2. ISA 总线的引线定义

图 2.33 所示为 ISA 总线的引脚示意图。

按照 ISA 总线的功能分类，各总线信号的功能说明如下：

（1）数据传输总线。

SD_0～SD_7，SD_8～SD_{15}：双向，数据总线 I/O 通道上的所有 8 位设备与 CPU 之间的数据传送使用 SD_0～SD_7，16 位设备的数据传送使用 SD_0～SD_{15}。为了支持 8 位设备，16 位 CPU 向 8 位设备的传送将变换为两个 8 位在 SD_0～SD_7 上进行传送。

$\overline{SBHE}$：双向，总线高字节允许信号。表示数据总线传送的是高位字节（SD_8～SD_{15}）。16 位设备用$\overline{SBHE}$信号控制数据总线缓冲器接到 SD_8～SD_{15}。

$\overline{MEMCS_{16}}$：输入，存储器 16 位芯片选择信号。用来通知主板，当前的数据传送是一个等待状态的 16 位存储器周期。它必须由来自 LA_{17}～LA_{23} 的译码所驱动。

$\overline{IOCS_{16}}$：输入，I/O16 位芯片选择信号。用来通知主板，当前的数据传送是一个等待状态的 16 位 I/O 周期。其驱动来自地址译码器。

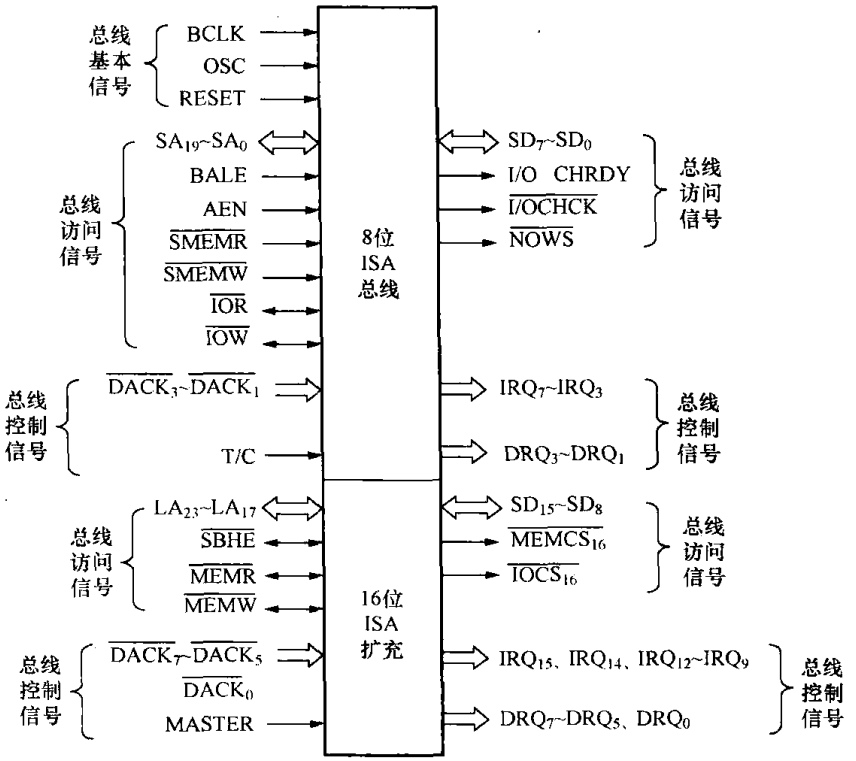


图 2.33 ISA 总线的引脚示意图

(2) 地址总线。

SA_0～SA_{19}：输出，系统地址总线。这 20 条地址线再加 LA_{17}～LA_{23} 就可对最多可达 16MB 的存储空间进行寻址操作。当 BALE=“H”时，SA_0～SAI_{19} 就挂到系统总线上，而在 BALE 的下降沿，它们被锁存。

LA_{17}～LA_{23}(I/O)：非锁存地址总线。当 BALE=“H”时，这些信号才有效。在 CPU 总线周期期间，这些信号不被锁存，所以在整个周期期间不保持有效，它们的用途是为一个等待状态的存储周期生成存储器译码信号，这些译码信号应由 I/O 适配器锁定在 BALE 的下降沿上。

BALE：输出，地址锁存允许信号。由 8288 总线控制器提供，用于在主板上锁存从处理器来的有效地址。在 I/O 通道上作为处理器或 DMAC 的地址指示。该信号与 AEN 一起锁存处理器地址，并在 BALE 的下降沿锁存。

AEN：输出，地址允许信号。用来使主板上的微处理器进入保持状态，以便进行 DMA 传送，该信号有效时，由 DMAC 控制系统总线。

(3) 控制总线

$\overline{IOR}$：双向，I/O 读命令信号。

$\overline{IOW}$：双向，I/O 写命令信号。

$\overline{MEMR}$：双向，存储器读命令信号。

$\overline{MEMW}$：双向，存储器写命令信号。

$\overline{SMEMR}$、$\overline{SMEMW}$：输出，存储器读、写命令信号。仅当存储器的地址译码信号选中的存储空间小于 1MB 时，这些信号才有效。

IRQ_3～IRQ_7、IRQ_9～IRQ_{12}、IRQ_{14}、IRQ_{15}：输入，中断请求信号。其中 IRQ_9 优先最

高，其次是 IRQ_{10}～IRQ_{12}、IRQ_{14}、IRQ_{15}，最低的是 IRQ_3～IRQ_7（IRQ_7 最低）。

DRQ_0～DRQ_3、DRQ_5～DRQ_7：输入，DMA 请求信号。其中 DRQ_0 的优先级最高，DRQ_7 的优先级最低。其中 DRQ_0～DRQ_3 执行 8 位 DMA 传送，而 DRQ_5～DRQ_7 执行 16 位 DMA 传送。

$DACK_0$～$DACD_3$、$DACK_5$～$DACK_7$：输出，DMA 响应信号用来响应 DMA 请求。

I/O $\overline{CHCK}$：输入，I/O 通道校验信号。为主板提供有关 I/O 通道上存储器或 I/O 设备的奇偶校验出错信息。当该信号有效时，表示有无法更正的系统错误。

I/O $\overline{CHRDY}$：输入，I/O 通道就绪信号。受存储器或 I/O 设备控制，当该信号为低电平时，就要延长 I/O 或存储器的周期。

T/C：输出，计数结束信号。当任一 DMA 通道的字计数器计满时，送出一个脉冲信号。

REFRESH：双向，刷新信号。指示一个存储器刷新周期，输出低电平时，它启动外部 RAM 的刷新周期。

MASTER：输入，主设备信号。此信号与 DRQ 信号一起使用，以便对系统进行控制。

（4）时钟、定时与电源。

OSC：输出，振荡器信号。一个周期为 70ns（14.31818 MHz）的高速时钟，该信号与系统时钟 SYSCLK 相同，占空比为 50%，提供给其他功能部件使用。

SYSCLK：输出，系统时钟信号。用于与微处理器的时钟周期同步的时钟信号，只应用于同步，而不能用作固定的频率源。

RESETDRV：输出，复位驱动。

OWS：输入，零等待状态信号。用于通知微处理器，它可以完成当前的总线周期，不必增加任何附加的等待状态。

电源及地线：共有+5V 电源线 3 条，−5V 电源线 1 条、+12V 电源线 1 条、−12V 电源线 1 条，以及地线 4 条。

2.5.3 EISA 总线

在 ISA 总线之后，先后出现了 EISA（Extended Industrial Standard Architecture，扩展的工业标准体系结构）和 VESA（Video Electronic Standards Association）总线。EISA 总线是 1988 年由 Compaq 等 9 家公司联合推出的总线标准，是一种支持多处理器的高性能 32 位标准总线。

1. EISA 总线概述

EISA 是扩展的 ISA 总线，引脚由原来 ISA 总线的 62 个扩展到了 198 个，其数据总线宽度 32 位，地址总线宽度 32 位，并具有高度同步传送功能。但时钟速度仍维持在 8MHz，传输速率为 33MB/s，由于 EISA 总线性能稳定，适用于网络服务器、高速图像处理、多媒体等领域，常用作磁盘控制器和视频图形适配器。

EISA 总线的信号线可分为地址总线和数据总线组、数据传送控制组、总线仲裁信号组及其他功能总线组 4 组。

EISA 总线扩展槽（连接器）的插脚分为上、下两层，上层是原 ISA 总线的连接，下层是 EISA 总线新增信号的连接。当 ISA 插卡插入时，它仅能接触到槽上的上层接线，而当 EISA 插卡插入时，则同全部总线信号的连线相接触，总线能自动识别所连接的插卡采用的

是 ISA 插卡还是 EISA 插卡，并运行相应的周期，这样就保证了 EISA 总线同 ISA 总线的兼容性。

与 ISA 总线相比，EISA 总线有如下特点：

(1) EISA 总线用于 32 位微型计算机中，支持 32 位地址总线寻址，可寻址 4GB 的存储空间，也支持 64KB 的 I/O 端口寻址。

(2) 它具有 32 位数据线，大大提高了数据传输能力，使最大数据传输速率达 33MB/s。

(3) EISA 总线支持多处理器结构，支持多主控总线设备，具有较强的 I/O 扩展能力和负载能力。

(4) 具有自动配置功能，可以根据配置文件自动地初始化，配置系统板和多扩展卡。

(5) 扩展了 DMA 的范围和传输速度，支持 7 个 DMA 通道，DMA 数据传输既可在 ISA 方式下进行，也可在 EISA 方式下进行。而且在 EISA 方式下进行 DMA 数据传输时，使用的数据总线和地址总线都是 32 位的。

(6) 采用同步数据传送协议，可支持常规的一次传送，也可支持突发方式即高速分组传送。

2. EISA 总线引脚信号

EISA 总线将 198 个引脚分为 8 列，其中 A 列、B 列和 C 列、D 列为上部触点，前者是 62 个引脚，后者是 36 个引脚，这 4 列同 ISA 总线信号完全兼容。而 E 列、F 列和 G 列、H 列为下部触点，前者是 62 个引脚，后者是 38 个引脚，这 4 列是 EISA 总线新增加的总线引脚。

EISA 总线引脚信号，在 ISA 原有信号外，主要增加的信号如下：

D_{31}～D_{16}：数据总线高 16 位，以支持 32 位数据传送；

LA_{31}～LA_{24}：地址总线高 8 位，以支持 32 位寻址；

LA_{16}～LA_2：新增的未锁存的 A_{16}～A_2 地址总线；

$\overline{BE_3}$～$\overline{BE_0}$：对 32 位数据总线 4 字节中各字节的选通信号；

LOCK：总线锁定信号，提供互斥的总线访问周期；

EX_{32}：内存或 I/O slave（从设备）用该信号表示其对 32 位传送的支持；

EX_{16}：内存或 I/O slave 用该信号表示支持 16 位数据传送；

START：指明 EISA 总线周期开始；

CMD：该信号给出在一个周期中的时序控制；

M/$\overline{IO}$：指示当前周期类型是内存访问还是 I/O 访问；

W/$\overline{R}$：指示当前周期类型是读还是写；

EXRDY：用于 EISA 的 READ 线，指示是否需要加入等待周期；

MREQx：对应每一插槽的 EISA 总线请求线；

MAKx：对应每一插槽的 EISA 总线请求响应线；

SLBURST：slave 用该信号表示对突发方式传送的支持；

MSBURST：master（主设备）用该信号向 slave 表示该 master 可产生突发方式的传送。

3. EISA 总线的数据传送周期

(1) 突发方式传送（Burst Type）。EISA 突发方式传送是指在总线上进行 16 位或 32 位的成块数据传送。当访问这一组连续数据时，只有在传送第一个数据时需要 2 个时钟周期，

第一个时钟周期给出地址，第二个时钟周期传送数据，而传送其后的连续数据时，传送一个数据只要一个时钟周期，不必每次都给出地址，这种传送称为“突发传送”或“成组传送”。

(2) 非突发方式传送 (Non-burst Type)。非突发方式传送是指在 EISA 总线上进行的一般数据传送，每个数据传送时间需要 2 个 BCLK。

(3) Peek-Poke 和锁定交换传送 (Peek-Poke and Locked Exchange)。该传送方式允许本地 CPU 通过此方式对 EISA 系统的 I/O 空间和内存空间的某一地址访问以及进行互斥的访问。

(4) 非匹配数据传送 (Mismatched Type)。当进行数据传送的源和目的数据宽度不一致时，就应进行数据宽度的转换。这一转换由 EISA 总线完成，从而使 8 位、16 位以及 32 位的不同设备进行直接数据交换而无需考虑数据宽度匹配问题。前三种传送中任何时候一旦需要进行数据的宽度转换，EISA 总线的控制器都会自动进行本方式的传送。

4. EISA 总线说明

对于要求多个高性能的并行处理外部设备的系统而言，EISA 总线就显得特别有用。采用智能 EISA 主设备，可以分担很多主 CPU 本来需要承担的工作，使得大量数据的传送和相当一部分的处理工作可由智能设备完成，从而提高了整个系统的性能。

为了便于开发者设计 EISA 系统，Intel 公司推出了符合 EISA 标准的一套超大规模集成电路 (VLSI) 芯片组——82350EISA 套片。包括：

(1) 82352 EISA Bus Buffers (EBB)，EISA 总线缓冲器；

(2) 82357 Integrate System Periphral (IsP)，集成系统外设接口；

(3) 82358 EISA Bus Controller (BEC)，EISA 总线控制器；

(4) 82355 Bus Master Interface Controller (BMIC)，总线主设备接口控制器。

其中，82355 BMIC 是专为构成 EISA 总线主设备而设计的 VLSI，特别适合于用 BMIC 来设计的智能主设备。采用 BMIC 可代替 20 个以上的中小规模集成电路芯片，从而大大简化了设计过程，加快了开发速度。

2.5.4 PCI 总线

1991 年下半年，Intel 公司首先提出了 PCI(Peripheral Component Interconnect，外围部件互连) 总线。PCI 是一种同步且独立于处理器的 32 位或 64 位的局部总线，支持多总线主控和线性突发方式。PCI 能够支持微处理器快速访问系统存储器，并支持适配器之间的相互访问。PCI 局部总线不能兼容现有的 ISA、EISA、MCA(Micro Channel Architecture) 总线，但它不受制于处理器，是基于奔腾等新一代微处理器而发展的总线。

1. PCI 系统结构

图 2.34 所示为基于 PCI 总线的微机系统典型结构框图。目前，一般的奔腾系统中，都采用 PCI 与 ISA 总线并存的系统。

典型的 PCI 系统包括两个桥接器：Host/PCI 桥和 PCI/ISA 桥。

Host/PCI 桥也称为北桥 (North Bridge)，连接 CPU 和基本 PCI 总线，其中包括存储器管理部件和 AGP 接口部件，使得 PCI 总线上的部件可以与 CPU 并行工作。

PCI/ISA 桥也称为南桥 (South Bridge)，连接基本 PCI 总线到 ISA 或 EISA 总线，其中包括中断控制器、IDE 控制器、USB 主控制器和 DMA 控制器，它可将 PCI 总线转换为标准总线，如 ISA，EISA 等，以便在标准总线上挂接低速设备，如打印机、调制解调器 (MODEM)、传真机、扫描仪等。

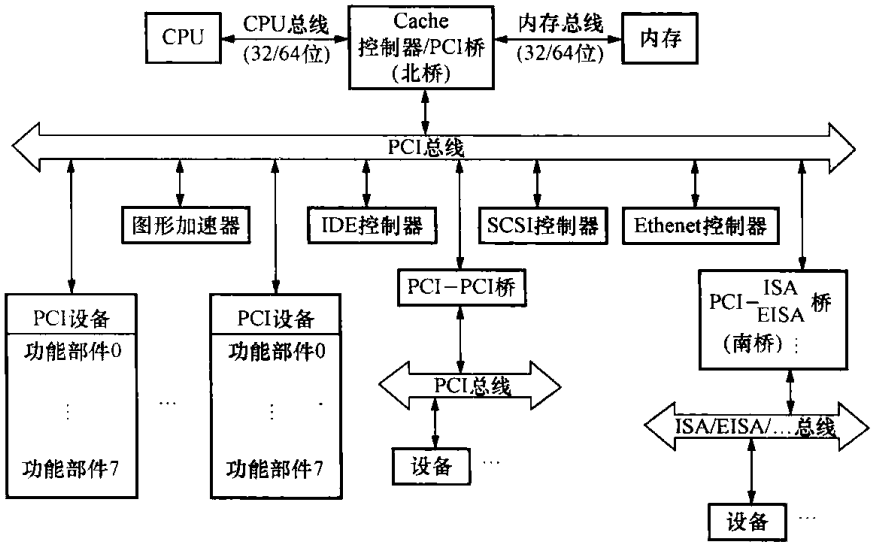


图 2.34 基于 PCI 总线的微机系统典型结构框图

北桥和南桥构成芯片组，基本 PCI 总线上可以连接一个或多个 PCI 桥，一个芯片组可以支持一个以上北桥。PCI 系统由桥接器将处理器、存储器、PCI 和扩展系统联系在一起。

2. PCI 局部总线的主要性能和特点

PCI 总线是一种不依附于某个具体微处理器的局部总线。从结构上看，PCI 是在 CPU 和原来的系统总线之间插入的一级总线，由一个桥接电路实现对这一层的管理，并作为接口以协调数据的传送。

（1）数据传输率高。PCI 的数据总线宽度为 32 位，可扩充到 64 位。它以 33MHz 的时钟频率工作。因此，若采用 32 位数据总线，数据传输率可达 132MB/s；而采用 64 位宽度数据总线，则最高传输率可达 264MB/s。

（2）支持突发传输。通常的数据传输是先输出地址后进行数据操作，即使所要传输数据的地址是连续的，每次也要有输出和建立地址的阶段。而 PCI 支持突发数据传输周期，这意味着从某一个地址开始后，可以连续对数据进行操作，而每次的操作数地址是自动加 1 的。显然，这减少了无谓的地址操作，加快了传输速度。

（3）支持多主控器。在同一条 PCI 总线上可以有多个总线主控器（主设备），各主控器通过 PCI 总线专门设置的总线占用请求信号和总线占用允许信号竞争总线的控制权。

（4）减少存取延迟。PCI 总线能够大幅度减少外设取得总线控制权所需的时间，以保证数据传输的畅通。

（5）支持即插即用。即插即用（Plug and Play）是指在新的接口卡插入 PCI 总线插槽时，系统能自动识别并装入相应的设备驱动程序，因而立即可以使用。

（6）独立于处理器。传统的系统总线实际上是中央处理器信号的延伸或再驱动，而 PCI 总线以一种独特的中间缓冲器方式独立于处理器，并将中央处理器子系统与外设分开。

通常在中央处理总线上增加更多的设备或部件会降低系统的性能和可靠性。而有了这种缓冲器的设计方式，用户可随意增添外设而不必担心会导致系统性能下降。

（7）PCI 总线提供了数据和地址的奇偶校验功能，保证了数据的完整性和准确性。通过转换 5V 和 3.3V 工作环境，PCI 总线可适用于各种规格的计算机系统。PCI 总线插槽短而

精致，PCI 芯片均为超大规模集成电路，体积小而可靠性高，PCI 总线采用地址/数据引脚复用技术，减少了引脚需求。

3. PCI 总线信号

PCI 总线引脚数为 120 条（包含电源、地、保留引脚等），如图 2.35 所示，左边为必要信号，右边为任选信号。这些总线信号按功能分为如下九组：

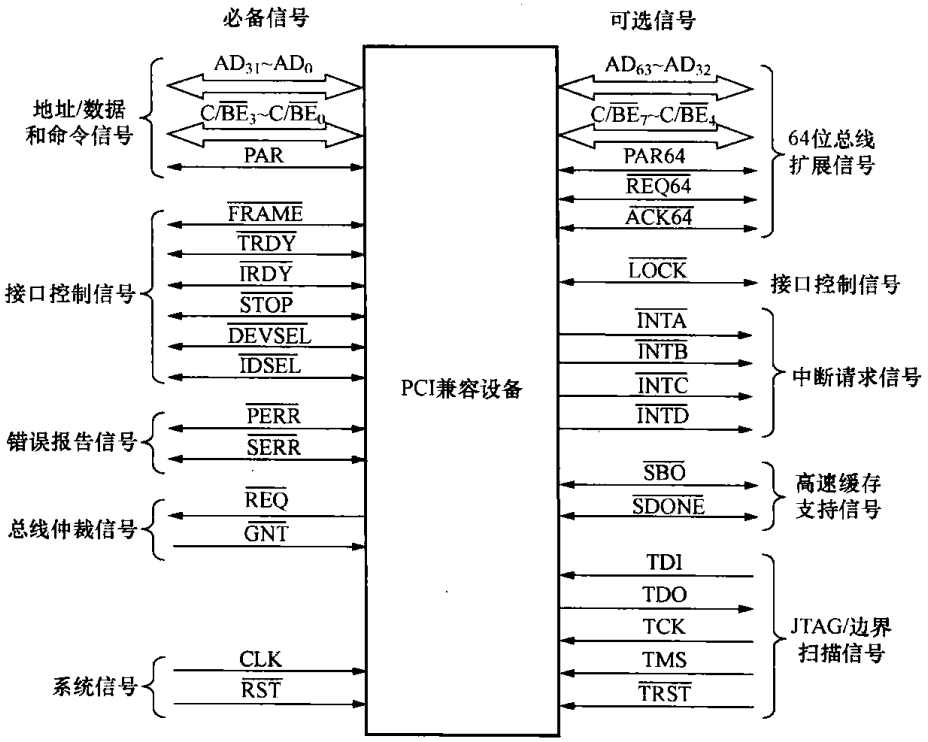


图 2.35 PCI 总线信号

（1）系统信号。

CLK(Clock)：PCI 系统时钟信号，输入。时钟信号的频率范围是 0～33MHz，或 33.33～66.66MHz。而 66MHz 时钟仅支持 3.3V 的信号环境。

$\overline{\text{RST}}$(reset)：复位信号，输入，低电平有效。当复位信号有效时，将所有 PCI 专用的寄存器、定时器和信号复位到指定状态。一般情况下，全部输出信号处于高阻状态。

（2）地址/数据和命令信号。

AD_{31}～AD_0(Address Data)：地址和数据分时复用。在$\overline{\text{FRAME}}$低电平有效时，该组信号线上传送 32 位物理地址：对于 I/O 端口，这是一个字节地址；在$\overline{\text{IRDY}}$和$\overline{\text{TRDY}}$都有效期间，该组信号线上传送 32 位数据，AD_7～AD_0 为最低字节数据，AD_{31}～AD_{24}为最高字节数据。当$\overline{\text{IRDY}}$有效时，表示写数据稳定有效；当$\overline{\text{TRDY}}$有效时，则表示读数据稳定有效。

$C/\overline{BE_3}$～$C/\overline{BE_0}$(Command/Byte Enable)：命令或字节使能信号，双向、三态，由主设备驱动。这组信号定义了总线操作类型，或者作为字节使能信号，指出当前寻址的双字中传送的字节和用于传送数据的数据通道。

当$C/\overline{BE_3}$～$C/\overline{BE_0}$表示 PCI 命令类型时，它的 16 种状态中的 12 种分别表示了 12 种总线操作，如表 2.15 所示。

表 2.15 PCI 命 令

$C/\overline{BE_3}\sim C/\overline{BE_0}$	命 令 类 型	$C/\overline{BE_3}\sim C/\overline{BE_0}$	命 令 类 型
0000	中断响应	1000	保留
0001	特殊周期	1001	
0010	I/O 读	1010	配置读
0011	I/O 写	1011	配置写
0100	保留	1100	存储器多行读
0101		1101	双地址周期
0110	存储器读	1110	存储器行读
0111	存储器写	1111	存储器写和读无效

当 $C/\overline{BE_3}\sim C/\overline{BE_0}$ 表示字节使能时，它指出当前操作使用的数据通道。其字节使能信号在数据通道和当前寻址的双字空间的映射关系，如表 2.16 所示。

表 2.16 字节使能信号在数据通道和当前寻址的双字空间的映射关系

字节使能	映 射	字节使能	映 射
$C/\overline{BE_3}$	数据通道 3，$AD_{31}\sim AD_{24}$ 和当前寻址双字的第 4 个字节	$C/\overline{BE_1}$	数据通道 1，$AD_{15}\sim AD_8$ 和当前寻址双字的第 2 个字节
$C/\overline{BE_2}$	数据通道 2，$AD_{23}\sim AD_{16}$ 和当前寻址双字的第 3 个字节	$C/\overline{BE_0}$	数据通道 0，$AD_7\sim AD_0$ 和当前寻址双字的第 1 个字节

PAR(parity)：奇偶校验信号，双向、三态。对 $AD_0\sim AD_{31}$ 和 $C/\overline{BE_0}\sim C/\overline{BE_3}$ 信号作奇偶校验（偶校验），以保证数据的有效性，分别被主设备和从设备驱动。

(3) 接口控制信号。PCI 总线接口控制信号是总线操作过程中主设备和从设备的联络信号。

$\overline{FRAME}$：帧周期信号。由当前总线主设备控制，表示一个总线周期的开始和结束。

$\overline{TRDY}$(Target Ready)：从设备准备好信号。由从设备控制，表示从设备准备好传送数据。

$\overline{IRDY}$(Initiator Ready)：主设备准备好信号。由系统主设备控制，与$\overline{TRDY}$信号同时有效可完成数据传输。

$\overline{STOP}$：停止信号。从设备要求主设备停止当前数据传送。

$\overline{DEVSEL}$(Device Select)：设备选择信号。该信号有效时，当作为输出信号时，表示所译码的地址是在设备的地址范围内，当作为输入信号时，表示总线上某设备是否被选中。

$\overline{IDSEL}$(Initiatization Device Seleet)：初始化设备选择信号。在配置读写期间，用作芯片选择。

$\overline{LOCK}$：锁定信号。用于保证主设备对存储器的锁定操作。

(4) 仲裁信号。在两个以上的总线主设备请求使用总线时，就需要对请求总线的设备进行仲裁。

(5) 中断请求信号。$\overline{INTA}$、$\overline{INTB}$、$\overline{INTC}$和$\overline{INTD}$(Interrupt) 是 PCI 的四个中断请求信号，漏极开路，低电平触发有效。

(6) 错误报告信号。

$\overline{\text{PERR}}$(Parity Error)：数据奇偶校验错信号。

$\overline{\text{SERR}}$(System Error)：系统错误信息。

用于报告地址奇偶错、数据奇偶错和命令错等。

(7) 64 位总线扩展信号。PCI 规范定义了基于 32 位结构的 64 位扩展功能，扩展系统支持在一个 64 位主设备和一个 64 位从设备之间进行 8 字节的数据传送。

(8) 高速缓存支持信号。

$\overline{\text{SBO}}$(Snoop Back Off)：双向、三态探测返回信号。当该信号有效时，关闭预测命令中的一个缓冲行。

$\overline{\text{SDONE}}$(Snoop Done)：双向、三态探测完成信号。当该信号有效时，表示探测完成，命中一个缓冲行。

(9) JTAG 边界扫描信号。边界扫描电路用于测试 PCI 上设备的内部电路，对 PCI 设备进行功能测试。

4. 输入/输出控制方式

(1) 突发传送。突发传送是 PCI 总线操作的一个主要特点，几乎所有 PCI 支持的数据交换都可以由突发传送来实现。突发传送是一种包含一个地址段，后面跟着两个或两个以上数据项的数据传送方式。

对于总线拥有者来说，总线主设备只需进行一次仲裁。在突发周期内，主设备在地址段发出起始地址和操作类型，总线上所有的设备都锁定地址和操作类型，并将其译码，以确定谁是从设备。从设备将起始地址锁存在地址计数器（假设它支持突发模式）中，并且随着一个个数据项的传送递增地址。

在主设备和从设备都没有插入等待状态的情况下，数据项（双字或四字）在每个 PCI 时钟周期的上升沿传送。66MHz 的 PCI 总线采用 32 位或 64 位数据传送时，可以达到 264MB/s 或 528MB/s 的传送速率。

(2) 主设备、从设备和代理。每个 PCI 突发传送都有两个参与者：主设备和从设备。总线主设备也称为启动方，是启动传送的设备。在 PCI 规范中，总线主设备和启动方两个术语意义相同。

从设备是指启动方为实现数据传送目的而正在寻址的设备。通常，PCI 启动方和从设备称为与 PCI 兼容的代理（Agent）。

(3) 地址段。PCI 操作都是以第一个 PCI 时钟周期内的地址段开始的。在地址段，主设备通过驱动地址总线来鉴别从设备，通过驱动 PCI 命令/字节使能总线来确定操作类型（也称为命令类型）。在地址段，主设备使$\overline{\text{FRAME}}$信号有效，表示总线的起始地址和命令类型有效。每个 PCI 从设备必须在时钟的下一个上升沿锁存地址和命令，以便顺序译码。

(4) 声明一次操作。当 PCI 从设备确定自己是操作的从设备以后，通过有效的$\overline{\text{DEVSEL}}$（设备选择）声明本次操作有效。如果主设备在一个预定的时间段内没有采样到有效的$\overline{\text{DEVSEL}}$，就使操作失败。

(5) 数据项传输。数据项传输是主设备和从设备之间传输数据的阶段。通常从一次操作的第二个 PCI 时钟周期开始，数据项传输的数据字节数由命令/字节使能信号决定，这些信号在数据项中是由主设备驱动的。

每个数据项传输至少持续一个 PCI 时钟周期，主设备和从设备都必须表明它们准备好进行数据项传输，否则数据项便通过插入一个持续的 PCI CLK 等待周期进行扩展。PCI 总线定义了准备好信号线，主设备使用$\overline{\text{IRDY}}$，而从设备使用$\overline{\text{TRDY}}$。

（6）操作过程。PCI 操作中所有的数据传送都可以是突发传送。主设备并不是向从设备发出传送数据项的数目，而是在每个数据项传输阶段，通过控制信号指出是否准备好传送当前数据项；如果准备好传送，指出该数据项是否是最后的数据项。在地址段的开始，主设备令$\overline{\text{FRAME}}$信号有效，并一直保持其有效状态，直到主设备准备好（$\overline{\text{IRDY}}$有效）完成最后一个数据项传输。当从设备在数据项传输过程中采样到有效的$\overline{\text{IRDY}}$信号和无效的$\overline{\text{FRAME}}$信号时，就知道了这是最后一个数据项。但是，直到从设备令$\overline{\text{TRDY}}$有效时，数据项传送才能完成。

（7）传送结束和总线空闲。主设备通过无效$\overline{\text{FRAME}}$和有效$\overline{\text{IRDY}}$表明，突发传送的最后一个数据在传送过程中，当最后的数据传送完成时，主设备通过使$\overline{\text{IRDY}}$无效，将 PCI 总线返回空闲状态，$\overline{\text{TRDY}}$和 C/$\overline{\text{BE}}_0$自然也是无效状态。如果另一个主设备已经被 PCI 总线仲裁器授权拥有总线，并正在等待当前主设备放弃总线，此时，它在 PCI 时钟的同一个上升沿检测到无效的$\overline{\text{FRAME}}$和$\overline{\text{IRDY}}$，则可知总线已经返回空闲状态。

2.6 外 部 通 信 总 线

2.6.1 EPP 并行总线

EPP(Enhanced Parallel Port) 是一种增强双向并行传输口，由 Intel、Xircom、Zenith 和其他一些公司共同开发，目的是在外部设备间进行双向通信。自 1991 年开始，笔记本电脑率先配备了 EPP 口。EPP 与标准并行接口 SPP（Standard Parallel Port）兼容。

1. EPP 概述

为了实现大量数据的快速、准确传输，数据通信已经成为自动化系统的关键技术之一。但是由于串行通信口的通信效率较低，难以完成大量、实时性高的数据传输工作。而传统的 SPP 并行接口只能实现数据的单向传输，EPP 并行口则可以进行高速的双向数据传输，是进行大量数据传输的一种理想选择，最高传输速度为 1.5Mb/s。

在实际应用中，由于 EPP 并行接口具有 8 位双向数据端口、状态端口和控制端口，可作为 8 位接口灵活使用。EPP 总线标准定义了各种单/双向信号以满足各自的特殊需要。

2. EPP 寄存器介绍

EPP 并行口定义的信号分别对应三个不同的寄存器。在 PC 机中，并行数据端口寄存器地址一般为 378H，数据端口是一个输入/输出双向缓冲寄存器，对应 $AD_7 \sim AD_0$ 双向数据/地址端口；输入操作 EPP 信号的定义及其描述，如表 2.17 所示。单向状态端口和单向控制端口的寄存器地址分别为 379H 和 37AH，其定义见表 2.18。这三个寄存器是与标准并行口 SPP 兼容的，但实际上除了这三个寄存器之外，EPP 还定义了其他几个寄存器。

表 2.17　　EPP 信号的定义及其描述

并行口引脚号	EPP 信号	数据方向	定　　义
1	$\overline{\text{WRITE}}$	输出	低电平写，高电平读
9～2	$AD_7 \sim AD_0$	双向	双向数据/地址线

续表

并行口引脚号	EPP 信号	数据方向	定　　义
10	INTR	输入	外设中断
11	WAIT	输入	握手信号
12	用户定义	输入	按不同的外设自定义
13	用户定义	输入	按不同的外设自定义
14	$\overline{\text{DATASTB}}$	输出	低电平有效，表示正在进行数据读写
15	用户定义	输入	按不同的外设自定义
16	RESET	输出	低电平有效，外设复位
17	$\overline{\text{ADDSTB}}$	输出	低电平有效，表示正在进行地址读写
18～25	GND		信号地

表 2.18　　EPP 寄存器定义

378H	AD_7	AD_6	AD_5	AD_4	AD_3	AD_2	AD_1	AD_0
379H	WAIT	INTR	自定义	自定义	自定义	无定义	无定义	无定义
37AH	无定义	无定义	READ ENABLE	IRQ ENBALE	$\overline{\text{ADDSTB}}$	$\overline{\text{BESTE}}$	$\overline{\text{DATA}}$	$\overline{\text{WRITE}}$

2.6.2 SCSI 总线

SCSI 总线是小型计算机系统接口（Small Computer System Interface，SCSI）总线，现已普遍应用于服务器或高档台式机和工作站。SCSI 总线是高度智能化的接口，它在传输过程中仅需 5%的 CPU 干预，这意味着 SCSI 在几乎不用 CPU 干预的情况下完成数据传输，从而提高了整个系统的性能。SCSI 定义了一种用来支持计算机和外围设备互连的总线，它被设计成一种有效的外设总线，用来支持多个设备，允许包括多个主机。这样通过单一的 SCSI，可使不同的磁盘、磁带、打印机和光驱能加入到主机系统中去，而不需要修改系统的硬件或软件。

8 位数据宽度的 SCSI，称为窄总线，仅能驱动 7 个 SCSI 设备。现在 16 位数据宽度的宽总线 SCSI 则可驱动 15 个设备。SCSI 接口定义，如表 2.19 所示。Ultra SCSI 接口定义采用了 LVD（低电压差分）信号电平，它较 HVD（高电压差分）信号电平可以达到更高的传输速度。SCSI 窄总线采用 50 芯连线和接插件；SCSI 宽总线则是 68 芯的电缆，当需要设计成具有热插拔功能的 SCSI 设备时，则使用 80 芯的接口。

表 2.19　　SCSI 接口定义

总线类型	数据宽度（位）	信号电平	最大传输率（Mb/s）
SCSI-1	8	HVD	5
SCSI-2	8	HVD	10
SCSI-3	8 或 16	HVD	20
ultra SCSI	8 或 16	LVD	40
ultra SCSI-2	8 或 16	LVD	80
ultra SCSI-3	16	LVD	160
ultra SCSI320	16	LVD	320

SCSI 总线是一种很规范的总线逻辑接口。SCSI 有$\overline{BUSY}$、$\overline{SEL}$、$\overline{RST}$、$\overline{MSG}$、C/D、I/O、$\overline{REQ}$、$\overline{ATN}$和$\overline{ACK}$ 9 条控制线，这些控制线协同 8 位或 16 位数据线完成总线的操作。信息传送过程通过 MSG、C/D、I/O 信号组合，来决定信息传送的种类和方向，如表 2.20 所示。

表 2.20　　SCSI 信息传送的种类和方向

MSG	C/D	I/O	传送类型	传送方向
1	0	1	命令	I→T
1	1	1	数据	I→T
1	1	0	数据	I←T
1	0	0	状态	I←T
0	0	1	消息	I→T
0	0	0	消息	I←T

注　I为启动设备；T为目的设备。

SCSI 信息传送的不仅仅是数据，还包括命令、状态和消息。它们都是通过数据线进行传送的，信息传送过程中启动设备和目的设备用$\overline{REQ}$和$\overline{ACK}$进行同步。启动设备发$\overline{REQ}$同时将数据放入总线（I→T 操作），而目的设备则用$\overline{ACK}$来确定数据已接收。有关其他信号的定义请参考相关资料。

2.6.3　通用串行总线 USB

通用串行总线 USB(Universal Serial Bus) 实际上是一个万能的插口，可以取代 PC 机上所有的端口（包括串行端口和并行端口），用户可以将几乎所有的外设装置等插头插入标准的 USB 口中。同时还可以将一些 USB 外设进行串接，这样可以使多个设备共用 PC 机上的端口。此外，一些 USB 产品，如数码相机和扫描仪，甚至可以不要使用独立的电源即可工作，因为 USB 可提供电源。

1. USB 总线的功能特点

USB 总线是 Intel，DEC，Microsoft，IBM 等公司联合推出的一种新的串行总线标准，主要用于 PC 与外设的互联。

（1）USB 减轻了各个设备（像鼠标、调制解调器、键盘和打印机等）对目前 PC 中所有标准端口的需求，因而降低了硬件的复杂性和对端口的占用。整个 USB 系统只有一个端口，使用一个中断，节省了系统资源。

（2）USB 支持热插拔（Hot Plug)。也就是说，在不关闭微型机的情况下，可以安全地插上和断开 USB 设备，动态地加载驱动程序。

（3）USB 支持即插即用（Plug and Play，PnP)。当插入 USB 设备的时候，计算机系统检测该外设，并且自动加载相关驱动程序，对该设备进行配置，使其正常工作。

（4）USB 在设备供电方面提供了灵活性。USB 接口不仅可以通过电缆为连接到 USB 集线器（Hub）或主机（Host）的设备供电，而且可以通过电池或者其他的电力设备为其供电，或使用两种供电方式的组合，并且支持节约能源的挂机和唤醒模式。

（5）USB 提供全速 12MB/s、低速 1.5MB/s 和高速 480MB/s(USB 2.0) 三种速率来适应各种不同类型的外设。

（6）为了适应各种不同类型外设的要求，USB 提供了四种不同的数据传送类型。

（7）USB 具有很强的连接能力，最多可以以链接形式连接 127 个外设到同一系统，这

对一般的计算机系统已经足够了。

(8) USB 具有很高的容错性能。因为在协议中规定了出错处理和差错恢复的机制，所以可以对有缺陷的设备进行认定，并对错误的数据进行恢复或报告。

总之，USB 在传统的计算机组织结构的基础上，引入网络的拓扑结构思想。其具有终端用户的易用性、广泛的应用性、带宽的动态分配、优越的容错性能、较高的性能价格比等特点，方便了外设的添加，适应了现代计算机的多媒体功能拓展，已逐步成为计算机的主流接口。

2. USB 物理接口

(1) USB 设备。USB 设备有集线器和功能部件两类。在 USB 系统中，集线器也是一种设备，即集线器设备。集线器可内置于某个设备（如键盘、显示器等）中，这种集线器被看成设备的一种功能。集线器简化了 USB 互连的复杂性。集线器串接在集线器上，可让不同性质的更多设备连在 USB 接口上，其连接点称作端口，每个集线器的上行端口向主机方向进行连接，下行端口允许连接其他集线器或功能部件。集线器可检测每个下行端口设备的安装或拆卸，并可为下行端口的设备分配资源。每个下行端口可分辨出连接的是高速设备还是低速设备。在设备与设备之间是无法实现直接通信的，只有通过主机的管理与调节，才能实现数据的互相传送。

(2) USB 电缆。USB 通过一种四芯电缆传送信号和电源。USB 2.0 在 USB 1.0 的基础上增加了另一种数据传输速率：高速 480MB/s。如图 2.36 所示，USB 电缆中包括 V_{BUS}、GND 两条电源线，用来向设备提供电源。V_{BUS}的电压为+5V。为了保证足够的输入电压和终端阻抗，重要的终端设备应位于电缆尾部，每个端口都可检测终端是否连接或分离，并区分出高速或低速设备。电缆中还有两条互相缠绕的数据线。所有设备都有一个上行和下行的连接，上行连接器和下行连接器不可互换，因而避免了集线器间非法的、循环往复的连接。



图 2.36 USB 连线的定义

(3) USB 电源。它主要包括电源分配和电源管理两方面的内容。电源分配是指 USB 如何分配计算机所提供的能源。需要主机提供电源的设备称为总线供能设备，如键盘、输入笔和鼠标等。而一些 USB 设备可能自带电源，该类设备称为自供能设备。USB 主机有与 USB 设备相互独立的电源管理系统，系统软件可以与主机的电源管理系统结合，共同处理各种电源事件，如挂起、唤醒等。每个设备可以从总线上获得 100mA 的电流，如果有特殊情况向系统申请，最多可以获得 500mA 的电流，在挂机的状态下，电流只有 500mA。

USB 总线只有一个主机，其余的都是 USB 设备。主机和 USB 设备可以直接相连，或者通过 USB 集线器（Hub）使一个 USB 端口同时接多个 USB 设备。还可以通过多个集线器级联，接更多的 USB 设备。

3. USB 系统的拓扑结构

USB 系统采用级联星形拓扑结构，USB 系统由主机（USB Host）、集线器（USB Hub）和 USB 设备（USB Device）三部分组成。

主机（USB Host）：安装在主板上或作为适配卡安装在计算机上。主机由主控制器和根集线器组成，控制着 USB 总线上的数据和控制信息的流动。在一个 USB 系统中，当且仅当

有一个 USB Host 时，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 设备（USB Device）：在一个 USB 系统中，USB Device 和 USB Hub 总数不能超过 127 个。USB Device 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 Host 的数据包与 USB Host 进行数据传输。USB 设备通过端口与总线连接，USB 设备同时可作为 Hub 使用。

集线器（USB Hub）：用于设备扩展连接，所有 USB Device 都连接在 USB Hub 的端口上。一个 USB Host 总与一个根 Hub(USB Root Hub) 相连。USB Hub 为其每个端口提供 100mA 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 Host。一般来说，USB 设备与 USB Hub 间的连线长度不超过 5m，USB 系统的级联不能超过 5 级（包括 Root Hub）。

4. USB 总线传输协议

USB 是一种轮询方式的总线，主机控制器初始化所有的数据传输。

每个总线执行动作按照传输前制定的原则，最多传输三个数据包。每次传输开始，主机控制器发送一个描述传输动作的种类、方向、USB 设备地址和端口号的数据包，这个数据包通常称为标志包 PID(Packet ID)，USB 设备从解码后的数据包中取出属于自己的数据。

传输开始时，由标志包来标志数据的传输方向，然后发送端发送数据包，接收端相应地发送一个握手的数据包，以表明传输是否成功。发送端和接收端之间的数据传输，可视为在主机和设备端口之间的一条通道中进行。

通道可分为流通道和消息通道两类。各通道之间的数据流动是相互独立的，一个 USB 设备可以有几条通道。例如，一个 USB 设备可建立向其他设备发送数据和从其他设备接收数据的两条通道。

5. USB 总线数据传输方式

从物理结构上，USB 系统是一个星形结构；但在逻辑结构上，每个 USB 逻辑设备都是直接与 USB Host 相连进行数据传输的。在 USB 总线上，每毫秒传输 1 帧数据。每帧数据可由多个数据包的传输过程组成。USB 设备可根据数据包中的地址信息来判断是否响应该数据传输。为了满足不同的通信要求，USB 提供了四种传输方式：

（1）控制方式传输（Control Transfer）。控制方式传输是双向传输，数据量通常较小。控制传输类型支持外设与主机之间的控制、状态、配置等信息的传输，为外设与主机之间提供一条控制通道。每种外设都支持控制传输类型，这样，主机与外设之间就可以传输配置和命令/状态信息。

（2）同步方式传输（Isochronous Transfer）。同步方式传输提供了确定的带宽和间隔时间（Latency）。它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速率和即时应用中。例如，在执行即时通话的网络电话应用中，使用同步传输方式是很好的选择。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，即时数据传递比精度和数据的完整性更重要一些。

（3）中断方式传输（Interrupt Transfer）。中断方式传输主要用于定时查询设备是否有中断申请。这种传输方式的典型应用是在少量的、分散的、不可预测数据的传输方面，键

盘、操纵杆和鼠标等就属于这一类型。这些设备与主机间的数据传输量小、无周期性，但对响应时间敏感，要求马上响应。中断方式传输是单向的，并且对于主机来说只有输入方式。

(4) 块数据方式传输 (Bulk Transfer)。主要应用于大量传输数据又没有带宽和间隔时间要求的情况下，要求保证传输。打印机和扫描仪就属于这种类型，在满足带宽的情况下，才进行该类型的数据传输。

USB采用分块带宽分配方案，若外设超过当前或潜在的带宽分配要求，则主机将拒绝与外设进行数据传输。等时和中断传输类型的终端保留带宽，并保证数据按一定的速率传输，集中和控制终端按可用的最佳带宽来传输数据。但是，10%的带宽为批传输和控制传输保留，数据块传输仅在带宽满足要求的情况下才会出现。

2.6.4 AGP总线

AGP(Accelerate Graphical Port) 是一种显示卡专用的局部总线接口。严格地说，AGP不能称为总线，它与PCI总线不同，因为它是点对点连接，即连接控制芯片和AGP显示卡，但在习惯上依然称其为AGP总线。AGP接口是基于PCI 2.1版规范并进行扩充修改而成，工作频率为66MHz。

AGP总线直接与主板的北桥芯片相连，且通过该接口让显示芯片与系统主内存直接相连，避免了窄带宽的PCI总线形成的系统瓶颈，增加3D图形数据传输速度，同时在显存不足的情况下还可以调用系统主内存。所以它拥有很高的传输速率，这是PCI等总线无法与其相比拟的。

由于采用了数据读写的流水线操作减少了内存等待时间，数据传输速度有了很大提高；具有133MHz及更高的数据传输频率；地址信号与数据信号分离可提高随机内存访问的速度；采用并行操作允许在CPU访问系统RAM的同时AGP显示卡访问AGP内存；显示带宽也不与其他设备共享，从而进一步提高了系统性能。

AGP标准在使用32位总线时，有66MHz和133MHz两种工作频率，最高数据传输率为266Mb/s和533Mb/s，而PCI总线理论上的最大传输率仅为133Mb/s。目前最高规格的AGP 8X模式下，数据传输速度达到了2.1GB/s。

AGP接口的发展经历了AGP1.0(AGP1X、AGP2X)、AGP2.0(AGP Pro、AGP4X)、AGP3.0(AGP8X) 等阶段，其传输速度也从最早的AGP1X的266MB/s的带宽发展到了AGP8X的2.1GB/s。

不同AGP接口的特性，如表2.21所示。

表2.21　　不同AGP接口的特性

	AGP10		AGP2.0 (AGP4X)	AGP3.0 (AGP8X)
	AGP1X	AGP2X		
工作频率	66MHz	66MHz	66MHz	66MHz
传输带宽	266MB/s	533MB/s	1066MB/s	2132MB/s
工作电压	3.3V	3.3V	1.5V	1.5V
单信号触发次数	1	2	4	4
数据传输位宽	32bit	32bit	32bit	32bit
触发信号频率	66MHz	66MHz	133MHz	266MHz

目前常用的 AGP 接口为 AGP4X、AGPPRO、AGP 通用及 AGP8X 接口。需要说明的是，由于 AGP3.0 显卡的额定电压为 0.8～1.5V，因此不能把 AGP8X 的显卡插接到 AGP1.0 规格的插槽中。这就是说 AGP8X 规格与旧有的 AGP1X/2X 模式不兼容。而对于 AGP4X 系统，AGP8X 显卡仍旧在其上工作，但仅会以 AGP4X 模式工作，无法发挥 AGP8X 的优势。

习 题

2.1 8086/8088 CPU 由哪两部分组成？它们的主要功能各是什么？

2.2 8086/8088 CPU 为什么要采用地址/数据线分时复用？有何好处？

2.3 8086/8088 CPU 中的标志寄存器分为哪两类标志？两者有何区别？

2.4 设段寄存器 CS=2400H，指令指示器 IP=6F30H，此时指令的物理地址 PA 是多少？指向这一物理地址的 CS 值和 IP 值是否是唯一的？

2.5 什么叫总线周期？8086/8088 系统中的总线周期由几个时钟周期组成？如果 CPU 的主时钟频率为 25MHz，一个时钟周期是多少？一个基本总线周期是多少时间？

2.6 在总线周期的 T_1，T_2，T_3，T_4 状态，CPU 分别执行什么动作？什么情况下需插入等待状态 T_W？何时插入？怎样插入？

2.7 RESET 信号到来后，CPU 的状态有何特征？系统从何处开始启动？

2.8 8086/8088 在最大模式和最小模式下各有什么特点和不同？

2.9 8086/8088 的存储器空间各是多少？两者的存储器结构有何不同？寻址一个字节存储单元时有何不同？

2.10 什么是指令周期？什么是时钟周期？什么是总线周期？三者有何关系？简述 8086/8088 最小模式下的总线读操作和写操作的过程及所涉及的主要控制信号。

2.11 设存储器内数据段中存放了两个字 2FE5H 和 3EA8H，已知 DS=3500H，数据存放的偏移地址为 4F25H 和 3E5AH，画图说明这两个字在存储器中的存放情况。若要读取这两个字，需要对存储进行几次读操作？

2.12 80486 微处理器有几个存储空间，存储空间分别是多少？

2.13 请叙述 80486 微处理器的工作方式。

2.14 请比较 8086、80486、pentium 微处理器的特点。

2.15 什么是总线？简述微机总线的分类。

2.16 简述 PCI 总线的特点。

2.17 简述 SUB 总线的特点。

第3章　指令系统及汇编语言程序设计

3.1　概　　述

计算机程序是一系列指令的有序集合。指令是计算机能够识别和执行的操作命令，一种计算机所能执行全部操作命令构成了计算机的指令系统或称指令集，其表征了计算机的基本功能。计算机的CPU不同，指令系统也不同。本章以80X86 CPU为主，介绍一般计算机指令系统的格式、寻址方式、常用指令的功能、汇编语言格式和基本程序设计。

指令系统还规定了每条指令的表示格式。一条完整的指令主要描述CPU应进行何种操作、完成该操作所需要的操作数以及下一条指令存放在何处。

指令有机器指令和符号指令两种书写格式。

3.1.1　机器指令格式

一条机器指令一般由操作码和操作数两部分组成。如图3.1所示。每部分都是用"0"、"1"排列的代码来表示。

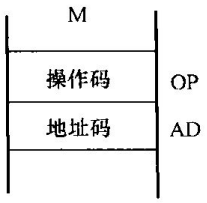


图3.1　机器指令格式

(1) 操作码 (Operation Code)。操作码用于说明指令操作的性质与所完成的功能，常用OP表示。一般计算机有几十种甚至几百种不同的指令，每条指令都有自己的操作码。计算机通过译码电路来识别指令，并进行相应的处理。操作码的长度取决于指令系统规模的大小。例如，长度为 n 的操作码可表示 2^n 条指令。在微型计算机中，操作码长度通常以字节为单位。

(2) 操作数 (Operation Data)。在计算机中，操作数可能在主存中，也可能在寄存器中。操作数给出了参与运算的操作数本身、操作数所在寄存器或操作数在存储器中的地址，运算结果放至何处，或给出下一条执行指令的地址信息。根据一条指令中所含操作数地址的数量，可分为零地址、单地址、双地址、二地址和四地址指令五种。在80X86系列指令格式中，多数为双地址指令。一般主存是按字节长度（即8位二进制）编址的，通常称为一个存储单元，双地址指令是指该指令机器码占两个8位存储单元（或地址）。

8086/8088指令编码由1～6个字节组成，它包括操作码、寻址方式以及操作数三部分，如图3.2所示。通常，指令的第一字节为操作码，规定指令的操作类型。第二字节规定操作数的寻址方式（或操作码），是指令编码中最复杂的字节。第三部分指定操作数或操作数所在地址，大多数是第二字节的补充。例如，ADD　AX，[BX+DI+2000H] 其指令编码是03H、41H、00H、20H。

操作码(1字节)	寻址方式(无或1字节)	操作数(无或1~4字节)

图3.2　8086/8088指令编码格式

80386/80486 CPU允许的指令字长可达1～16个字节之多，具有灵活多变、功能强、应用范围广等特点。如图3.3所示，其指令可由6个字段组成。字段1为附加字段，它由

1～4 个字节组成，可包括指令前缀字节、地址尺寸前缀字节、操作数尺寸前缀字节、段超越字节。字段 2～6 为基本字段，是指令格式中必不可少的部分。

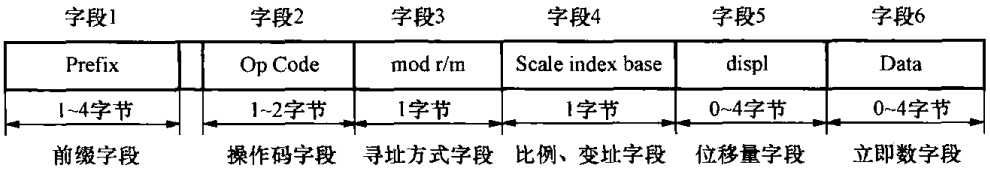


图 3.3　80386/80486 CPU 指令编码的一般格式

3.1.2　符号指令格式

符号指令的书写格式如下：

标号：操作码助记符　操作数助记符；注释

其中，操作码助记符规定了完成何种操作。操作数助记符规定了操作数个数和位置，有两个操作数时通常称第一个为目的操作数，第二个为源操作数。

标号代表该条指令的存放地址。它为程序分支、循环、跳转提供了转移目标地址。标号与符号指令之间用冒号“:”作为间隔符。为阅读方便，每条符号指令后面，可以有注释，并用分号“;”作为间隔符。

用机器指令编写的程序称为目标程序，用符号指令编写的程序称为符号程序或汇编语言源程序。汇编语言源程序要经过汇编程序将其编译和链接后才能生成 CPU 能识别和执行的目标程序。

3.2　寻　址　方　式

寻址方式就是用于说明指令中如何提供操作数或提供操作数存放地址的方法。在微机系统中，指令中的操作数有四类寻址方式：

（1）立即寻址方式。这种操作数就包含在指令中，是指令字节中的一部分。相应的寻址方式称为立即寻址（Immediate Addressing）。

（2）寄存器寻址方式。这种操作数存放在 CPU 某个寄存器（通用寄存器或段寄存器）中。相应的寻址方式称为寄存器寻址（Register Addressing）。

（3）存储器寻址方式。这种操作数存放在存储器中。相应的寻址方式称为存储器寻址。

（4）I/O 端口寻址方式。这种操作数存放在 I/O 接口电路的端口中。相应的寻址方式称为 I/O 端口寻址。

下面，结合 80X86 介绍常用的寻址方式。

3.2.1　立即寻址方式

立即寻址的操作数包含在指令字节中。指令操作码的后面字节就是操作数本身，故称为立即数。在微机系统中，立即数可以是 8 位或 16 位数。在 80386/80486 及以后微机处理器中，立即数还可以是 32 位数。在 80X86 系列微机中，立即数在指令字节中的存放顺序为：其低位字节放在低地址单元，高位字节放在后继的高地址单元中，如图 3.4 所示。

这种寻址操作，操作数紧随操作码后，都在代码段中。在取操作码后，随后取操作数，指令执行速度较快。主要用于寄存器赋值，并且只有源操作数部分才允许用立即数。例如：

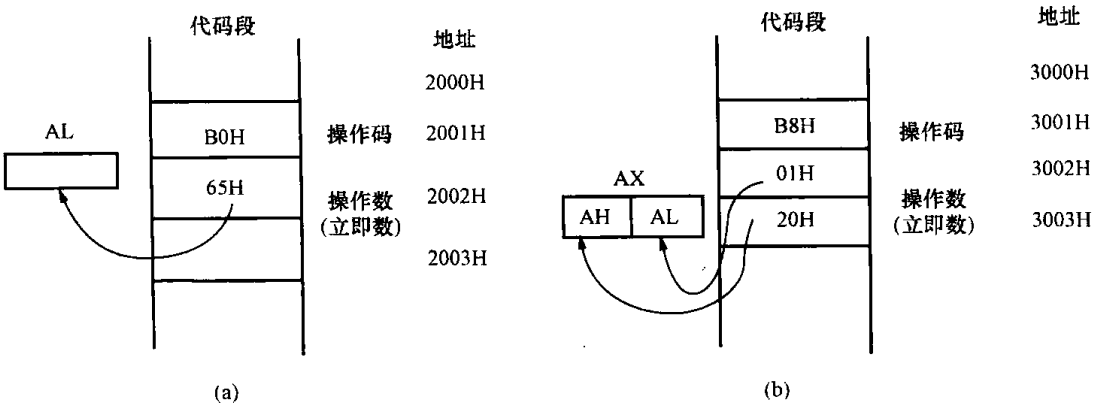


图 3.4 立即数寻址示意图

(a) "MOV AL, 65H" 操作示意图; (b) "MOV AX, 2001H" 操作示意图

```
MOV     AX, 3254H          ;16 位立即数 3254H 送 AX
MOV     BL, 65H            ;8 位立即数 65H 送 BL
MOV     EBX, 82746621H     ;32 位立即数 82746621H 送 EBX
```

3.2.2 寄存器寻址方式

该寻址规定 CPU 中某个寄存器的内容就是操作数。在符号指令的操作数部分写出相应的寄存器名称即可。例如:

```
MOV     BX, AX             ;AX 寄存器的内容送入 BX 寄存器,AX 值不变
MOV     DS, AX             ;AX 的内容送入 DS 段寄存器,AX 值不变
INC     SI                 ;SI 寄存器的内容加 1 再送回 SI
ADD     EAX, ECX           ;32 位寄存器 EAX, ECX 的内容相加,结果送 EAX,ECX 不变
MOV     AX, 2345H          ;2345H 送 AX,源操作数为立即寻址,目的操作数为寄存器寻址
```

3.2.3 存储器寻址方式

该寻址方式的操作数存放在某个逻辑段的存储单元中。CPU 要访问存储单元中的操作数必须先计算其在存储器中的物理地址，然后对该存储单元进行读/写操作。

我们知道，80X86 系列 CPU 对存储器采用分段管理。在编程时操作数地址都是用逻辑地址来表示，逻辑地址写成段基址 SBA (Segment Basic Address)：有效地址 EA (Effective Address)。其中，SBA 是逻辑段在存储器中的段基地址，由各段寄存器给定；EA 是此存储单元与段基址间的字节距离——段内偏移地址（偏移地址、偏移量、有效地址），即存储器操作数逻辑地址＝段寄存器：有效地址 EA。例如，指令 MOV ES: ALPHA, AL 表示将 AL 中内容送至 ES 逻辑段中偏移量为 ALPHA 的存储单元中，如图 3.5 所示。

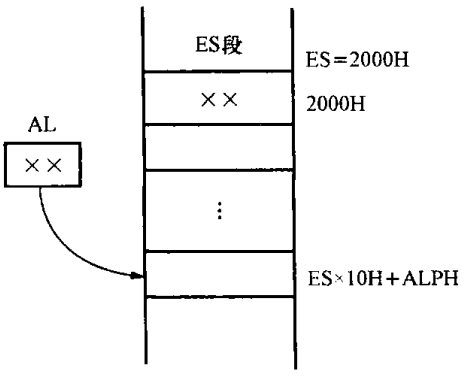


图 3.5 EA 计算示意图

(1) 段寄存器：称为段超越前缀助记符（简称段超越前缀），它用于说明 CPU 要访问的逻辑段。

(2) 有效地址 EA：冒号的后面部分，用于表示在此逻辑段中，存放操作数的存储单元相对于段基址之间的字节地址偏移量。

在指令系统中，有些指令是由操作数的寻址方式和操作性质决定使用哪个段寄存器，这时所使用的段寄存器称为指令隐含的或称默认的段寄存器。例如，跳转指令使用代码段寄存器 CS，堆栈操作指令使用堆栈段寄存器 SS，直接寻址操作数使用数据段寄存器 DS，若操作数不在隐含的段，这样就需要在操作数字段用指定的段寄存器代替隐含的段寄存器。此时指令中操作数必须用“段寄存器名：有效地址”加以指明，即用段超越前缀。

80X86 CPU 在实地址方式下，存储器操作数寻址方式的各种指令所隐含的段寄存器或允许另外指明的段超越所用的段寄存器、偏移地址的计算应根据第 2 章的表 2.1 来决定。根据有效地址 EA 的不同形式有九种存储器操作数寻址方式。

1. 直接寻址

存储器直接寻址是把操作数地址直接作为指令中的地址码，该操作数地址通常存放在操作码之后的指令字节中。例如：

```
MOV AL,[0064H]     ;将 DS * 16 + 0064H 单元内容送 AL 寄存器
```

图 3.6 所示为“MOV AL，[0064H]”指令直接寻址示意图。首先从代码段取出指令字节，再求出物理地址。若 DS=2000H，指令中包含直接地址为 0064H，则存放操作数的物理地址＝DS * 10H＋0064H＝20064H。此指令的功能是将 20064H 单元中的字节操作数 1FH 送 AL 寄存器中。

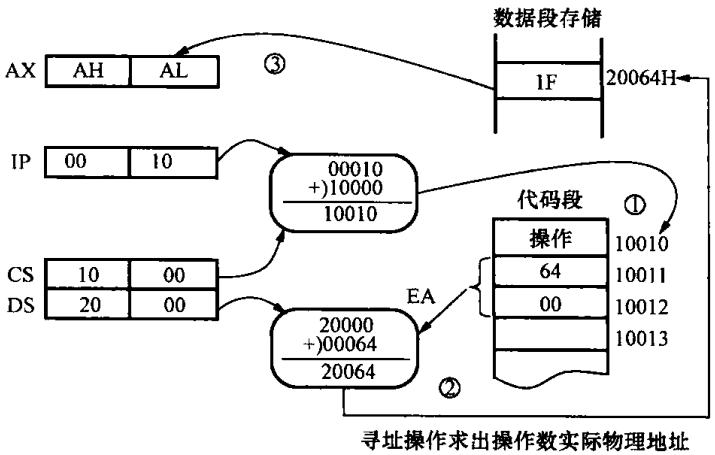


图 3.6　“MOV AL，[0064H]”指令直接寻址示意图

```
MOV AX, SS:[2000H]  ;
```

将 SS * 10H＋2000H 单元内容送 AL，SS * 10H＋2001H 单元内容送 AH。此条指令指明源操作数在堆栈段，在汇编语句中，常用 [ ] 括号表示存储器操作数，SS 为段超越前缀。

```
MOV AX, VALUE  ;
```

将 DS * 10H＋VALUE 单元内容送 AL，DS * 10H＋VALUE＋1 单元内容送 AH，其中 VALUE 为变量名。

```
MOV EAX, ES:BUFFER  ;
```

将 ES * 10H+BUFFER 开始连续四个单元内容送 EAX 寄存器，前两个单元是低 16 位，后两个单元是高 16 位。

在直接寻址中，指令中不允许两个操作数都是存储器操作数。例如，如欲将 DS 中 0100H 字单元的内容传送至 0200H 单元中，不能用“MOV [0200H], [0100H]”，而只能用如下两条指令来完成：

```
MOV AX,[0100H]     ;将 0100H 字单元内容先送 AX
MOV [0200H],AX     ;再通过 AX 送 0200H 字单元
```

2. 寄存器间接寻址

操作数的有效地址 EA 存放在基址寄存器 BX、BP 或变址寄存器 SI、DI 中，这种寻址方式称为寄存器间接寻址。此时，指令中操作数的逻辑地址表达式为：段寄存器：[间接寻址寄存器]。如果操作数不在规定默认逻辑段，则必须在指令相应操作数前加上段超越前缀。

寄存器间接寻址的有效地址和物理地址表达式如下：

$$EA=\begin{cases}BX\\SI\\DI\end{cases}\quad ,操作数物理地址 = DS * 10H + EA$$

$$EA=\ BP\qquad ,操作数物理地址 = SS * 10H + EA$$

例如：

```
MOV AX, [BX]          ;将物理地址为 DS×10H+BX 开始的连续两个单元内容(操作数)送 AX
MOV AX, ES:[SI]       ;将物理地址为 ES×10H+SI 开始的连续两个单元内容(操作数)送 AX
MOV DS:[BP], AX       ;指明目的操作数在 DS 段
MOV [DI], DX          ;将 DX 的内容送 DS * 10H+DI 开始连续的两个单元,其中 DL
                      ;内容存入 DS×10H+DI 单元,DH 内容存入 DS×10H+DI+1 单元
```

图 3.7 所示为“MOV AX, [BX]”指令寄存器间接寻址过程示意图。

首先，由“CS：IP”取出指令字节；再由“DS：[BX]”求出存放存储器操作数的物理地址；从物理地址中取出字操作数送 AX 中。若设 DS＝2000H，BX＝1000H，则物理地址＝20000H＋1000H＝21000H；若 21000H 和 21001H 中分别存放的数值为 30H、50H，则指令执行结果 AX＝5030H。

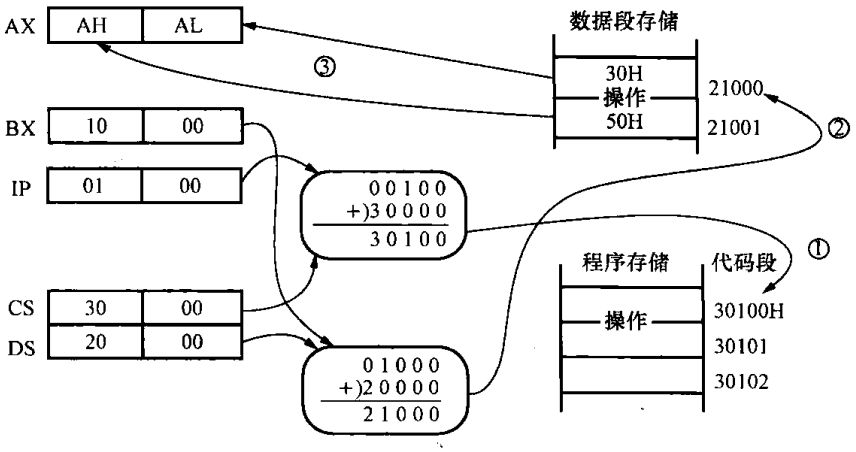


图 3.7　“MOV AX，IBXJ”指令寄存器间接寻址过程示意图

对于 32 位 CPU，8 个 32 位通用寄存器均可用作寄存器间接寻址，其有效地址和物理地址表达式为：

$$EA=\begin{cases}EAX\\EBX\\ECX\\EDX\\ESI\\EDI\end{cases}\text{，操作数物理地址} = DS*10H+EA$$

$$EA=\begin{cases}ESP\\EBP\end{cases}\text{，操作数物理地址} = SS*10H+EA$$

例如：

```
MOV EBX,[EAX]          ;由 DS 段寄存器形成段基址,从 EA=(EAX)所指的双字单元中传送双字给 EBX
MOV BX,[EAX]           ;由 DS 形成段基址,从 EA=(EAX)所指的字单元中传送字给 BX
MOV EAX,DS:[EBP]       ;指定 DS 为段基址,传送双字给 EAX
MOV FS:[SI],AX         ;指定 FS 为段基址,将 AX 内容传送至由 SI 指定的有效地址单元中
```

在 80386/80486 中除可用 DS、ES 作为数据段寄存器外，还可用 FS 和 GS 作为数据段寄存器的段超越。

3. 寄存器相对寻址

在寄存器相对寻址方式中，操作数的有效地址 EA 是由基址寄存器或变址寄存器加上一个 8 位或 16 位或 32 位（32 位 CPU）位移量（Disp）而得到的。其中，位移量是一个常量或变量，是指令字节中的一部分。指令中操作数的逻辑地址表达式为：段寄存器：[基址寄存器/变址寄存器＋位移量]，也可写为：段寄存器：[基址寄存器/变址寄存器] 位移量。

如果访问内存操作不是默认段寄存器，则允许段超越前缀。

其有效地址和物理地址表达式为：

$$EA=\begin{cases}BX\\SI\\DI\end{cases}+8\text{ 位或 }16\text{ 位位移量，操作数物理地址} = DS*10H+EA$$

$$EA = BP + 8\text{ 位或 }16\text{ 位位移量，操作数物理地址} = DS*10H+EA$$

$$EA=\begin{cases}EAX\\EBX\\ECX\\EDX\\ESI\\EDI\end{cases}+8\text{ 位或 }32\text{ 位位移量，操作数物理地址} = DS*10H+EA$$

$$EA=\begin{cases}ESP\\EBP\end{cases}+8\text{ 位或 }32\text{ 位位移量，操作数物理地址} = SS*10H+EA$$

例如：

```
MOV     AX,[BX+DATA]       ;将 DS*10H+BX+DATA 开始连续两个单元内容送 AX
MOV     AL,[SI+2000H]      ;将 DS*10H+SI+2000H 单元内容送 AL
```

```
MOV     EAX,5[EBP]          ;将 SS * 10H + EBP + 5 开始连续四个单元内容送 EAX
MOV     AL,ES:[SI + 5]      ;将 ES * 10H + SI + 5 单元内容送 AL
```

图 3.8 所示为“MOV AX，[BX+DATA]”指令寻址过程示意图。设 DATA=0100H，DS=2000H，BX=1000H，物理地址=2000 * 10H+1000H+0100H=21100H。

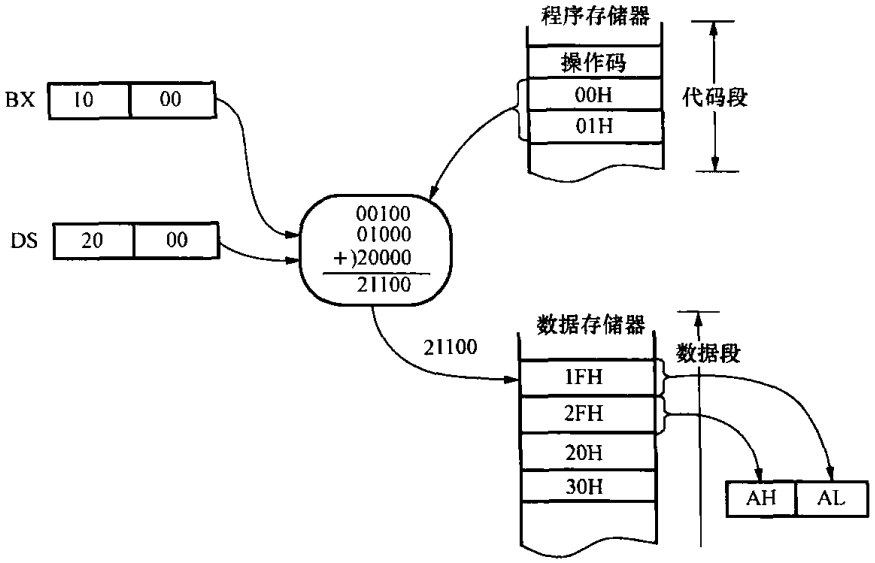


图 3.8 “MOV AX，[BX+DATA]”指令寻址过程示意图

寄存器相对寻址方式常应用于对一维数据结构或表格中某存储单元的寻址，如图 3.9 所示。若一个数据单元占用一个字节，则位移量为所选择单元序号 i；若一个数据单元占用 N 个字节，则位移量 disp=$N * i$ (i=0～n)。则第 i 个数据单元的有效地址 EA 表示为：

EA = [基址寄存器] + $N * i$

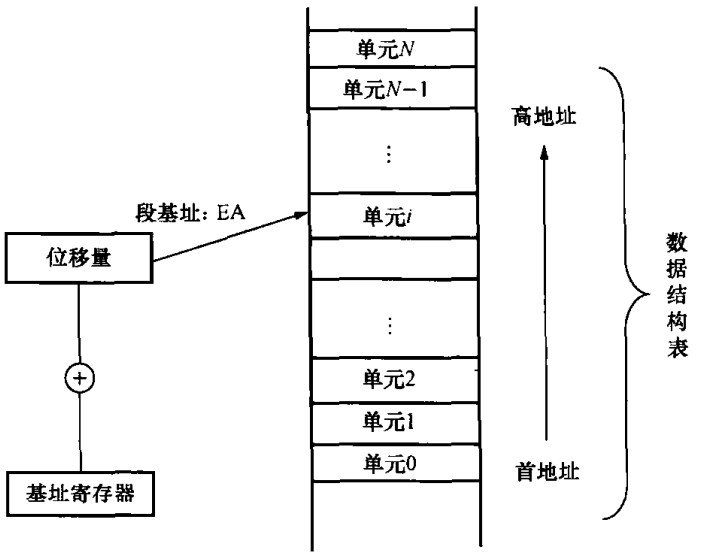


图 3.9 位移量、数据单元与单元序号间关系

4. 基址变址寻址 (Based Indexed Addressing)

该寻址方式的有效地址 EA 由基址寄存器内容加上变址寄存器内容组成。其操作数逻辑地址表达式为：

段寄存器：[基址寄存器＋变址寄存器]或段寄存器：[基址寄存器][变址寄存器]

其有效地址和物理地址表达式为：

$$EA=\begin{cases}BX+\begin{cases}SI\\DI\end{cases}, & \text{操作数物理地址}=DS*10H+EA\\ BP+\begin{cases}SI\\DI\end{cases}, & \text{操作数物理地址}=SS*10H+EA\end{cases}$$

例如：

MOV AX，[BX][SI]；或写成 MOV AX，[BX＋SI]。

设 DS＝6000H，BX＝A500H，SI＝2200H，则源操作数的物理地址＝DS＊10H＋A500H＋2200H＝60000H＋A500H＋2200H＝60700H。

对于 32 位 CPU，任何一个通用寄存器均可用作基址、变址（除 ESP）寄存器来使用。一般规定由基址寄存器来选取默认的段寄存器，默认段寄存器与寄存器间接寻址规定相同。

例如：

```
MOV AX,[BX][SI]        ;将 DS＊10H＋BX＋SI 开始连续两个单元内容送 AX 寄存器
MOV EAX,[EBP][EDI]     ;将 SS＊10H＋EBP＋EDI 开始连续四个单元内容送 EAX 寄存器
```

这种寻址方式主要用于二维数组中检索数组元素和二重循环程序等应用场合。

5. 相对基址变址寻址（Based Indexed Addressing with Displacement）

此寻址方式操作数有效地址由基址寄存器加变址寄存器再加 8 位或 16 位或 32 位（32 位 CPU）位移量组成，即：

EA＝［基址寄存器］＋［变址寄存器］＋位移量

这种寻址方式同样分 16 位和 32 位寻址两种情况。基址、变址寄存器使用约定和默认的段寄存器与寄存器相对寻址相同。例如：

```
        MOV AH,[BX][SI]＋1234H          ;将 DS＊10H＋BX＋SI＋1234H 单元内容送 AH
或      MOV AH,[BX＋SI＋1234H]
        MOV AX,CUNT[BX][SI]             ;将 DS＊10H＋BX＋SI＋CUNT 开始连续两个单元内容送 AX
或写成  MOV AX,[BX＋CUNT][SI]
        MOV EDX,[EBP＋0FFFF000H][EDI]   ;将 SS＊10H＋EBP＋FFFF000H＋EDI 开始连续四
或      MOV EDX,[EBP＋EDI＋0FFFF000H]   ;个单元内容送 EDX
```

图 3.10 所示为“MOV AX，CUNT［BX］［SI］”指令寻址过程示意图，设 DS＝3000H，BX＝2000H，SI＝1000H，CUNT＝3000H。源操作数物理地址＝3000H＊10H＋2000H＋1000H＋3000H＝36000H。

这种相对基址变址寻址主要用于二维数组操作，而位移量即为数组起始地址。图 3.11 所示即为相对基址变址，用于二维数据阵列检索。

对于 80486 还可以通过使用宽度超越前缀来强行指定执行 16 位数据，或者在默认的 32 位方式下寻址 16 位数据；采用地址宽度前缀使处理器在默认的 16 位方式下进行 32 位寻址，或者在默认的 32 位方式下进行 16 位寻址。事实上，用户在编程时，可以随意使用 16 位指令或 32 位指令，而不必考虑处理器工作在什么方式下，由汇编程序自动判断和完成的。

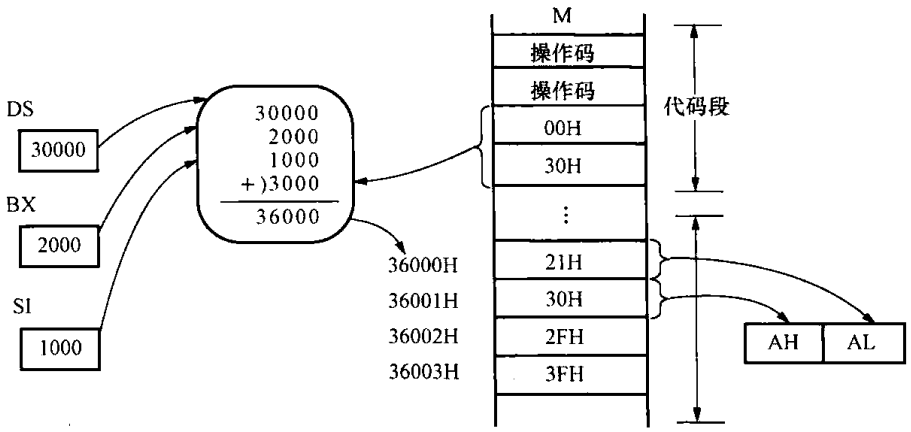


图 3.10 “MOV AX，CUNT [BX] [SI]”指令寻址过程示意图

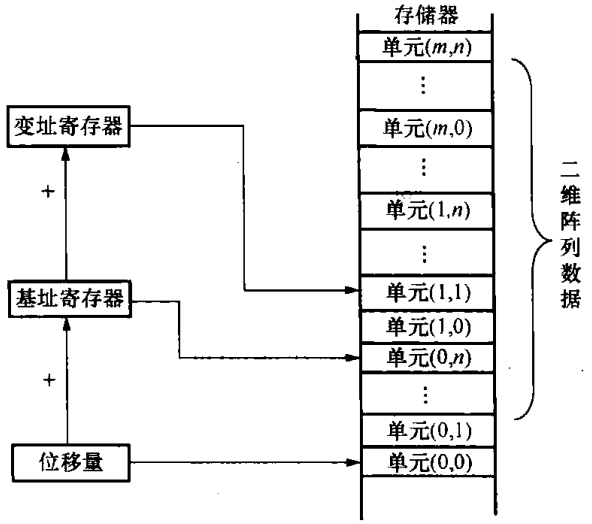


图 3.11 相对基址变址用于二维数据阵列

用户编程应注意在实地址方式下，寻址范围（偏移量）不应超过 FFFFH（64KB）；操作数可以是 8 位、16 位、32 位三种。

3.3 指 令 系 统

计算机功能的强弱和性能的优劣主要取决于计算机采用的 CPU 及其相应的指令系统。在 80X86 系列的 CPU 中，80486 的指令系统是在 8086/8088、80186、80286、80386 的指令基础上发展形成的，其机器代码完全向上兼容。80486 与其他的 80X86 相比，增强了指令功能，还增加了指令种类；它提供了 32 位的寻址方式和 32 位操作方式。

80486 的指令大致可分为整数指令、浮点指令和操作系统型指令三大类。整数指令是算术逻辑部件（ALU）执行的指令；浮点指令是浮点运算部件（FPU）执行的指令；操作系统型指令是为支持 CPU 保护工作方式的指令。80486 指令的长度是可变的，最短为 1 字节，最长为 16 字节。操作数可以是 0～3 个，其宽度可以是 8 位、16 位和 32 位。

本节主要介绍 80X86 的常用指令。对浮点指令和操作系统型指令见附录 4。

整数指令按功能分为数据传送类指令、算术运算指令、逻辑运算与移位指令、串操作指令、位操作指令、控制转移指令、标志操作指令、CPU 控制指令和高级语言指令等。

为便于指令介绍，把指令中要出现的一些符号说明如表 3.1 所示。

表 3.1　　指 令 符 号 说 明

符　号	意　义　说　明
OP1，OP2	操作数。多操作数指令中，OP1 为目的操作数，OP2 为源操作数
reg	通用寄存器。可以是 8 位、16 位或 32 位
sreg	段寄存器：CS，DS，ES，SS，FS，GS
reg8	8 位通用寄存器：AH，AL，BH，BL，CH，CL，DH，DL
reg16	16 位通用寄存器：AX，BX，CX，DX，DI，SI，SP，BP
reg32	32 位通用寄存器：EAX，EBX，ECX，EDX，EDI，ESI，ESP，EBP
mem	存储器。可以是 8 位、16 位或 32 位
mem8	8 位存储器。相应操作数只能是 8 位
mem16	16 位存储器。相应操作数只能是 16 位
mem32	32 位存储存器。相应操作数只能是 32 位
imm	立即数。可以是 8 位、16 位或 32 位
imm8	8 位立即数。有关操作数只能是 8 位

3.3.1　数据传送类指令

数据传送类指令用于寄存器、存储单元与 I/O 端口之间传送数据或地址。传送类指令是将源操作数复制到目的操作数（地址）中。除标志传送指令外，指令执行的结果不影响标志寄存器中的标志位。

这类指令分为通用数据传送指令、交换指令、堆栈操作指令、地址传送指令、输入/输出指令。

1. 通用数据传送指令 MOV

指令格式：　　MOV OP1，OP2

指令功能是将源操作数 OP2 内容（复制）传送到目标操作数 OP1 中。MOV 指令常用于对寄存器、存储器赋值，数据暂存等。

MOV 指令允许的源、目的操作数传送关系如图 3.12 所示。具体指令格式有：

```
MOV    reg/sreg,reg
MOV    reg,sreg
MOV    reg/sreg,mem
MOV    mem,reg/sreg
MOV    reg,imm
MOV    mem,imm
```

有关 MOV 指令的几点说明：

（1）目的操作数不能为 CS、IP（EIP）和立即数。

（2）源操作数和目的操作数必须等长，即同为字节、字、双字类型。

（3）源操作数与目的操作数不能同时为存储器操作数或段寄存器。若要传送，则要经过通用寄存器作为桥梁。例如：欲将存储器字变

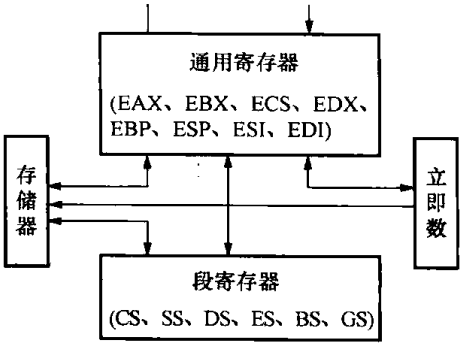


图 3.12　MOV 指令源、目的操作数传送关系

量 DATA 的内容传送至存储器字变量 REUS 单元中，需通过以下程序实现：

```
MOV AX,DATA     ;将 DATA 内容→AX
MOV REUS,AX     ;经 AX 送→REUS 单元中,没有 MOV REUS,DATA 指令
```

若要使段寄存器 ES 和 DS 指向同一段基址，则以通用寄存器为桥梁：

```
MOV AX,DS       ;DS 段基址→AX
MOV ES,AX       ;经 AX 再送→ES 段寄存器
```

(4) 不能将一个立即数传送给段寄存器，只能通过通用寄存器作为桥梁来实现：

```
MOV AX,1234H    ;立即数 1234H 经 AX 送入 DS 段寄存器中
MOV DS,AX       ;没有 MOV DS,1234H 指令
```

例如：

```
MOV AL,BL
MOV CL,4
MOV AX,03FFH
MOV AX,[BX]
MOV AL,BUFFER
MOV DST[DI],CX
MOV DS,CUNT[BX+SI]
MOV DEST[BP+DI],ES
MOV ES,CX
MOV BP,SS
```

2. 交换类指令

80X86 系列中交换类指令主要有交换指令 XCHG 和查表转换指令 XLAT/XLATB。

(1) 交换指令。

指令格式：XCHG　OP1，OP2

指令功能是实现源操作数 OP2 与目的操作数 OP1 的相互交换。OP1、OP2 两操作数可以是 8、16、32 位数；两操作数长度必须一致；两操作数不允许都是存储器操作数；两操作数中不允许有立即数；两操作数中不允许有段寄存器。例如：

```
XCHG EAX,ECX          ;两个 32 位寄存器 EAX,ECX 的内容相互交换
XCHG BX,[DI+2400H]    ;EA=DI+2400H,(EA)所指存储器与 BX 内容交换
XCHG AH,CL            ;两个 8 位寄存器 AH,CL 的内容相互交换
```

(2) 查表转换指令。

指令格式：XLAT/XLATB　　；AL←DS：[BX+AL]

指令功能是将存储器地址为 DS＊10H＋BX＋AL 单元的内容取出送入 AL 寄存器。此指令的操作数全都是隐含的。XLAT 指令规定事先应在 BX 中存放某一内存表格的首地址；AL 为表中某一元素项与表格首地址之间的偏移量（长度在 0～255 之间）。执行 XLAT 指令时，将（EA）放入 AL 中。此指令对标志位没有影响。

该指令常用于代码间的相互转换。例如，以 BCD 码表示的十进制数字（0～9）转换成

共阴极 LED 显示的字形代码，表 3.2 是十进制数、BCD 码、显示代码（共阴极 LED）对照关系表。将字形码按十进制数从小到大的次序排成一个表放在内存某个区域中，把此区域的首地址 TABLE 送 BX。设字形码在内存中定义如下：

```
TABLE   DB    3FH,06H,5BH,4FH,66H
        DB    6DH,7DH,07H,7FH,67H
NUMB    DB    ××H                    ;××为某十进制数的 BCD 码
```

采用以下几条指令可实现转换。

```
MOV     AX, ××××H
MOV     DS,AX                ;给 DS 段寄存器设段基址
MOV     BX,OFFSET TABLE      ;将字形码表首地址中的偏移量(有效地址)
                             ;送 BX(在汇编语言中 OFFSET 是取偏移地址)
MOV     AL,NUMB              ;将某数的 BCD 码 NUMB = ××H→AL
XLAT                         ;将 BCD 码××H 的字形码→AL 中
```

XLAT 指令所操作的数据一般都是 DS 段的数据，也可用段超越前缀，改变段寄存器。XLATB 指令的功能与 XLAT 指令相同，但此指令不允许使用段超越前缀，即只能对 DS 段的操作数进行传送操作。

表 3.2　　十进制数、BCD 码、显示代码（共阴极 LED）对照关系表

十进制数	BCD 码	字形代码	十进制数	BCD 码	字形代码
0	0000	3FH	5	0101	6DH
1	0001	06H	6	0110	7DH
2	0010	5BH	7	0111	07H
3	0011	4FH	8	1000	7FH
4	0100	66H	9	1001	67H

3. 地址传送类指令

80X86 的地址传送类指令分为有效地址传送指令和指针传送指令，且操作结果对标志寄存器的标志位没有影响。

（1）有效地址 EA 传送指令 LEA。

指令格式：

```
LEA     reg16, mem       ;mem 的有效地址 EA 送 16 位寄存器
LEA     reg32, mem       ;mem 的有效地址 EA 送 32 位寄存器
```

指令功能是将内存单元的有效地址 EA 传送到 16/32 位的目标寄存器中。

例如：

```
LEA     BX,TABLE                ;将变量为 TABLE 内存单元的偏移地址(有效地址)
                                ;送 BX(等同于 MOV BX,OFFSET TABLE)
LEA     SI,[BX + DI + 05H]      ;源操作数的有效地址 EA = BX + DI + 05H 送 SI
LEA     ESI,[EBX + 5]           ;将 EA = EBX + 5 内存单元的 32 位偏移地址送 ESI
```

(2) 指针传送指令。80386/80486 有 5 条地址指针传送指令，其功能是将 16/32 位有效地址存入通用寄存器、16 位段基址存入指令指定的段寄存器中。从而方便地利用一条指令来设定逻辑地址（段基址：有效地址）。具体指令格式为：

```
LDS     reg16/ reg32, mem       ;mem 单元开始 16 位/32 位内容→reg16/ reg32
                                ;mem + 2(或 + 4)单元开始 16 位内容→DS
LSS     reg16/ reg32, mem       ;mem 单元开始 16 位/32 位内容→reg16/ reg32
                                ;mem + 2(或 + 4)单元开始 16 位内容→SS
LES     reg16/ reg32, mem       ;mem 单元开始 16 位/32 位内容→reg16/ reg32
                                ;mem + 2(或 + 4)单元开始 16 位内容→ES
LFS     reg16/ reg32, mem       ;mem 单元开始 16 位/32 位内容→reg16/ reg32
                                ;mem + 2(或 + 4)单元开始 16 位内容→FS
LGS     reg16/ reg32, mem       ;mem 单元开始 16 位/32 位内容→reg16/ reg32
                                ;mem + 2(或 + 4)单元开始 16 位内容→GS
```

每条指令助记符后面的字母代表了指定的段寄存器。

如果目标寄存器是 16 位，则源操作数应是 32 位内存操作数，指令执行后操作数中的低 16 位数作为有效地址 EA 送指令指定的 16 位通用寄存器中，高 16 位数作为段基址送指令指定的段寄存器中；目标寄存器是 32 位，则源操作数应是 48 位内存操作数，其中低 32 位作为有效地址 EA 送指定的 32 位通用寄存器中，高 16 位作为段基址送指定的段寄存器中。

例，在数据段定义如下：

```
DATA1 DD 3000100AH          ;DD 为定义双字变量助记符
DATA2 DF 500011335577H      ;DF 为定义 6 个字节变量助记符
```

则数据段中数据单元的排列顺序如图 3.13 所示。执行下列指令可将有效地址和段基址送相应寄存器：

```
LDS SI,DATA1                ;装入 DS 段寄存器 DS = 3000H,SI = 100AH
LES EBX,DATA2               ;装入 ES 段寄存器 ES = 5000H,EBX = 11335577H
```

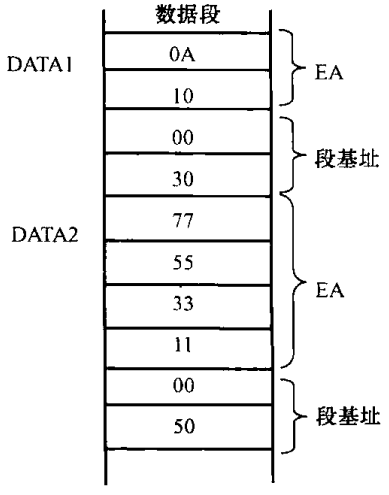


图 3.13 数据段中数据单元的排列顺序

4. 堆栈操作指令

堆栈是在内存中开辟的一个暂存单元区。在 80X86 系列中，堆栈是用段定义语句在内存中定义一个堆栈段。堆栈段的段基址存放在堆栈段寄存器 SS（ESS）中。偏移地址放在堆栈指针寄存器 SP（ESP）中，它总是指向堆栈区的顶端即栈顶。

CPU 对堆栈的操作采用后进先出（或先进后出）的存取方法，即最后存放堆栈中的数（在栈顶的数），最先从堆栈中取出。通过堆栈指针寄存器 SP（ESP）的间接寻址方式来获得相应的操作数。

CPU 对堆栈段中的存储单元进行存（压入堆栈）、取（弹出堆栈）操作。对应 80X86 的两类堆栈指令：一

类是把数据压入堆栈的 PUSH 指令；另一类是把数据从堆栈中弹出的 POP 指令。这些指令多数是单操作数指令，少量是无操作数指令，其他操作数是指令隐含的。

80X86 的堆栈是向下增长的，以实地址方式为例，在执行压入堆栈 PUSH 指令时，以 SP 作为堆栈指针时，每当压入一个字数据，SP 中的值减 2，指向低二字节的地址位置。相反，当执行弹出堆栈 POP 指令时，SP 的值加 2，指向高二字节的地址位置。如图 3.14 所示。80X86 初始堆栈指针应是偶数，以便在堆栈中按规则字进行存取操作。

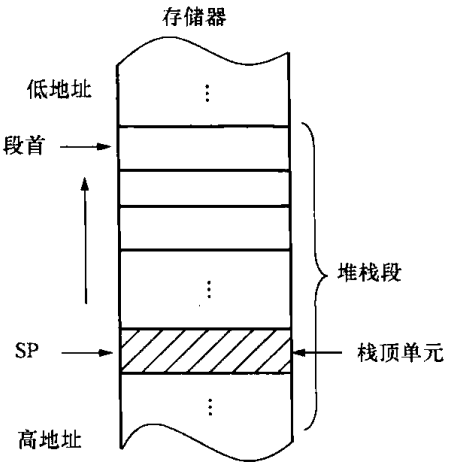


图 3.14　堆栈段示意图

在 80386/486 CPU 中，由 ESP 作为堆栈地址指针，还提供了双字操作，当执行压入/弹出 32 位操作数时，ESP 进行减 4 或加 4 操作，以修改堆栈指针。

在程序中，堆栈主要用于子程序调用、中断响应等操作时的参数保护，也可用于实现参数传递。

（1）压栈/出栈指令 PUSH/POP。

指令格式：　PUSH　　OP2（源操作数）

　　POP　　OP1（目的操作数）

指令中的 OP2 可以是 16/32 位的立即数、通用寄存器、段寄存器、存储器；OP1 可以是 16/32 位的通用寄存器、段寄存器（CS 除外）、存储器。

PUSH 指令功能是将指令中指定的 16/32 位操作数压入堆栈。执行过程为：

1）SP-2→SP 或 ESP-4→ESP

2）OP2 高 8/16 位→［SP-1］/［ESP-2］

3）OP2 低 8/16 位→［SP-2］/［ESP-4］

POP 指令功能是从栈顶把 16/32 位操作数从堆栈中弹出至目的操作数中。执行过程为：

1）［SP］/［ESP］→OP1 低 8/16 位

2）［SP+1］/［ESP+2］→ OP1 高 8/16 位

3）SP ＋ 2→SP 或 ESP ＋ 4→ESP

例如：

```
PUSH EBX        ;先修改堆栈指针 ESP←ESP-4;再将 EBX 双字压入堆栈
PUSH 1234H      ;先 SP←SP-2;将 1234H 字立即数压入堆栈
POP EAX         ;先栈顶双字弹出堆栈送 EAX;再修改指针 ESP ← ESP + 4
```

若已知压栈操作前 SS＝2000H，SP＝03AAH，AX＝1234H，BX＝5678H，则在实地址工作方式下执行如下指令：

```
PUSH AX         ;先修改指针 SP←SP -2;再数据压入堆栈[SP]←AX
PUSH BX         ;先修改指针 SP←SP-2;再数据压入堆栈[SP]←BX
POP AX          ;先数据弹出堆栈 AX←[SP];再修改指针 SP←SP + 2
POP BX          ;先数据弹出堆栈 BX←[SP];再修改指针 SP←SP + 2
```

其操作过程如图 3.15 所示。

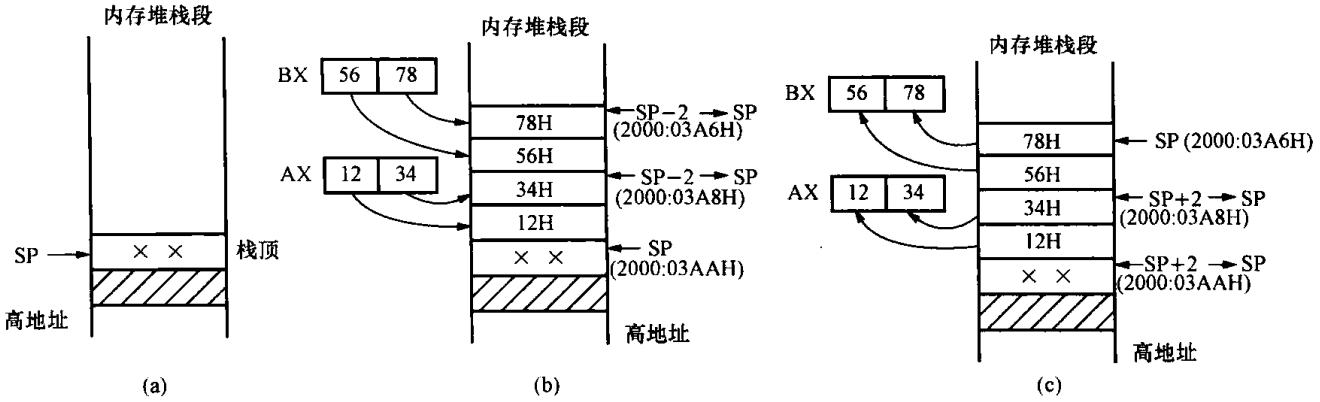


图 3.15 指令序列操作过程示意图

(a) 执行前；(b) PUSH 执行后；(c) POP 执行后

(2) 标志寄存器压栈/出栈指令。

指令格式：PUSHF/ PUSHFD

POPF/POPFD

PUSHF/PUSHFD 功能是将 FLAGS/EFLAGS 寄存器内容压入堆栈；POPF/POPFD 的功能是从堆栈中弹出字或双字至 FLAGS/EFLAGS 标志寄存器中。

PUSHF 指令的执行完成两步操作：

1) 修改堆栈指针 SP←SP-2；

2) 将 FLAGS 内容压入堆栈顶部 [SP] ←FLAGS。

POP 指令的执行同样完成两步操作：

1) 从堆栈顶弹出一个字至 FLGAS 中，LAGS← [SP]；

2) 修改堆栈指针 SP←SP-2。

80386/80486 在 32 位寻址时，堆栈指针应改为 ESP 寄存器。PUSHF/PUSHFD 具有相同的指令编码，POPF/POPFD 也具有相同的编码，均由当前操作数的字长决定应执行哪条指令。

5. 输入输出 (I/O) 指令

指令格式：IN AL/AX/EAX，imm8/DX

OUT imm8/DX，AL/AX/EAX

输入指令 IN 用于从 I/O 端口读数据到累加器 AL 或 AX 或 EAX 中；输出指令 OUT 用于把累加器 AL 或 AX 或 EAX 的内容写到 I/O 端口。只有累加器 AL/AX/EAX 能与 I/O 端口进行数据传送，也称累加器专用传送指令。

80X86 系统可以连接多个外部设备，像存储器一样用不同的地址（称 I/O 端口地址）来区分它们。根据端口地址长度不同，I/O 端口寻址方式有直接寻址方式和 DX 寄存器间接寻址方式两种。

(1) 直接寻址方式。当 I/O 端口地址用 8 位立即数表示时，I/O 端口地址可直接写在输入输出指令中。可寻址的 I/O 端口地址范围是 00～FFH。例如：

```
IN AL,3FH       ;表示从地址号为 3FH 的端口输入一个字节数→AL
IN AX, 3FH      ;表示从地址号为 3FH、40H 的 2 个端口中输入一个 16 位数(字)→AX
```

```
                ;其中从 3FH 端口读入的 8 位数→AL,从 40H 端口读入的 8 位数→AH 中
IN EAX,3FH      ;表示从地址号为 3FH～42H 的 4 个端口中输入 32 位数(双字)→EAX
OUT 43H,AL      ;把 AL 的内容(一个字节数据)→43H 端口
OUT 43H,AX      ;把 AX 的内容(一个字数据)→43H、44H 端口,AL→43H 端口,AH→44H 端口
OUT 43H,EAX     ;把 EAX 的内容(双字数据)→43H～46H 四个端口
```

(2) DX 寄存器间接寻址方式。当 I/O 端口地址为 16 位时，可寻址的 I/O 端口地址范围为 0000～FFFFH。此时 16 位端口地址由 DX 寄存器指定。例如：

```
MOV DX,03B0H    ;端口地址 03B0H 送 DX 寄存器
IN AL,DX        ;从 DX 指向的端口读一个字节数→AL
IN AX,DX        ;从 DX 及 DX+1 指向的两个端口读一个字数据→AX
IN EAX,DX       ;从 DX ～ DX+3 指向的四个端口读双字数据→EAX
OUT DX,AL       ;将 AL 的内容输出到 DX 所指的端口中
OUT DX,AX       ;将 AX 的内容输出到 DX 和 DX+1 所指的两个端口中
OUT DX,EAX      ;将 EAX 的内容输出到 DX ～ DX+3 所指的四个端口中
```

6. 其他传送指令

除以上传送指令外，80X86 指令系统中还有一些其他数据传送指令，其格式和功能如表 3.3 所示。

表 3.3　　其他传送类指令

指令类型	汇编格式	指令的操作	举例或说明
全部通用寄存器压栈/出栈指令	PUSHA	将所有 16 位通用寄存器内容压入堆栈。压入堆栈的顺序是 AX、CX、DX、BX、SP、BP、SI 和 DI。其中，SP 入栈内容是本条指令未执行前的 SP 值	
	PUSHAD	将所有 32 位通用寄存器内容压入堆栈。压入堆栈的顺序是 EAX、ECX、EDX、EBX、ESP、EBP、ESI 和 EDI。其中，ESP 入栈内容是本条指令未执行前的 ESP 的值	
	POPA POPAD	将弹出由 PUSHA 和 PUSHAD 指令保存在堆栈中的全部 16 位或 32 位通用寄存器内容返回到相应的通用寄存器中。出栈顺序与上述压栈顺序相反。但弹出 (E) SP 值丢弃，不再返回到 (E) SP 中	
标志传送指令	LAHF	将标志寄存器 FLAGS 的低 8 位传送到 AH 中	设 SP=1，ZF=0，AF=1，PF=1，CF=0 执行指令 LAHF，则 AH 各位内容为 10X1X1X0，其中 X 为任意值
	SAHF	将 AH 的内容传送到 FLAGS 的低 8 位	

续表

指令类型	汇编格式	指令的操作	举例或说明
符号扩展指令（此四条指令一般用在累加器作为被除数，在进行除法指令前使被除数位数符合运算规则）	CBW	将 AL 的符号位扩展到 AH 的所有位，由字节数扩展成字	MOV AL，7FHCBW ；结果 AX=007FH
	CWD	将 AX 的符号位扩展到 DX 的所有位，由字扩展成双字	MOV AX，9F45HCWD；结果 AX=9F45H，DX=FFFFH
	CWDE	将 AX 的符号位扩展到 EAX 的高 16 位，由字扩展成双字	
	CDQ	将 EAX 的符号位扩展到 EDX 的所有位，由双字扩展成四字	

3.3.2 算术运算指令

80X86 微处理器提供了一套加、减、乘、除四种基本算术运算指令，其操作数可以是 8 位、16 位、32 位有符号和无符号的二进制整数及无符号的 BCD 码；同时还提供了完成 BCD 码运算的调整指令。这类指令的执行会根据执行结果设置标志寄存器中的状态标志位。

1. 加法和减法指令

(1) 不带进位/借位的加、减法指令。

指令格式：

```
ADD    reg/mem,reg/mem/imm      ;reg/mem←reg/mem/imm + reg/mem
SUB    reg/mem,reg/mem/imm      ;reg/mem←reg/mem-reg/mem/imm
```

ADD 指令将源操作数和目的操作数相加，结果（和）存放在目的操作数中，源操作数不变；SUB 指令将目的操作数减源操作数，结果（差）存放在目的操作数中，源操作数不变。两者执行后均根据结果置标志寄存器中相应的标志位。

源和目的操作数均可以是 8/16/32 位的寄存器数、存储器，但不能同时为存储器操作数。源操作数还可以是 8/16/32 位立即数。

例如：

```
ADD AL,12H                ;AL←AL + 12H
ADD [BX + DI],AX          ;将 DS * 10H + BX + DI 和 DS * 10H + BX + DI + 1 所指两个存储单元内容组成
                          ;字操作数和 AX 相加，结果存入 DS * 10H + BX + DI 开始的两个存储单元中
ADD EBX,EAX               ;EBX←EAX + EBX,EAX 内容不变
SUB DTAT1,11223344H       ;DATA1←DATA1 开始(DATA1 为已定义的符号地址)连续 4 个单元的
                          ;内容组成的双字减 11223344H
```

[例 3.1] 说明执行“SUB [SI+14H]，0136H”指令后对标志位的影响。

解 设 DS=3000H，SI=0040H，(30054H) =4336H，指令功能是将 DS * 10H+SI+14H 开始连续两个单元内容减 0136H，差值送回目的操作数。指令执行结果对标志位的影响分析如下：

$$
\begin{array}{rr}
 & 4336H = 0100\ \ 0011\ \ 0011\ \ 0110 \\
+ & [-0136H]_{补} = 1111\ \ 1110\ \ 1100\ \ 1010 \\
\hline
 & 0100\ \ 0010\ \ 0000\ \ 0000
\end{array}
$$

从运算结果和过程可见：结果为正，SF＝0；结果低 8 位中有偶数个 1，PF＝1；结果不等于零，ZF＝0；辅助进位位没有借位，AF＝0；结果无溢出（0⊕0＝0），OF＝0；最高位没有借位，CF＝0。

[例 3.2]　下列指令完成无符号数相加，求相加结果对标志位的影响。

解

```
MOV AL,3AH;        AL←3AH
ADD AL,7CH;        AL←3AH + 7CH,AL = B6H
```

```
      3AH＝0011  1010
  ＋ 7CH＝0111  1100
-------------------------
        1011  0110  ＝ B6H
```

从运算结果和过程可见：结果为负（最高位是 1），SF＝1；结果中有奇数个 1，PF＝0；结果不等于零，ZF＝0；辅助进位位有进位，AF＝1；结果发生溢出（1⊕0＝1），OF＝1；最高位没有借位，CF＝0。

本例两个正数相加，结果为负数，肯定不正确。从溢出标志位 OF＝1 可知，两个 8 位数求和，产生了溢出（结果超出 8 位所能表示的范围），所得结果是不对的。

（2）带进位/借位的加减法指令。

指令格式：

```
ADC    reg/mem, reg←mem/imm     ;reg/mem←reg/mem + reg/mem/imm + CF
SBB    reg/mem, reg/mem/imm     ;reg/mem←reg/mem - reg/mem/imm - CF
```

ADC/SBB 指令主要用于多字节算术运算，ADC 比 ADD 多加一个低位来的进位，SBB 多减一个低位来的借位。其他规定与 ADD/SUB 指令相同。

[例 3.3]　求两个 4 字节无符号数 01180A6FH，1001B0EDH 的和。

解

```
MOV AX,0A6FH
ADD AX,0B0EDH
MOV DX,0118H
ADC DX,1001H     ;高位字加需要用 ADC,以便把低位可能产生的进位加进来
```

2. 加 1/减 1 指令

指令格式：

```
INC  reg/mem
DEC  reg/mem
```

INC 指令功能是将指定的 8、16、32 位操作数加 1 后再送回该操作数；DEC 指令是将指定的 8、16、32 位操作数减 1 后再送回该操作数。指令的执行结果影响 OF、SF、ZF、AF、PF，但不影响 CF。

这两条指令主要用于循环结构程序中以加 1/减 1 方式修改指针或计数。例如：

```
INC DI      ;DI←DI + 1
DEC SI      ;SI←SI-1
INC BYTE PTR[SI]     (DS * 10H + SI)←(DS * 10H + SI) + 1
```

[例 3.4] 延时程序。

解

```
        MOV CX,0FFFFH      ;送计数初值→CX
NEXT:   DEC CX             ;CX-1→CX
        JNZ NEXT           ;若 CX≠0 则转 NEXT 处继续执行
        HLT                ;停机
```

3. 求补指令

指令格式： NEG reg/mem ; reg/mem←0-reg/mem

指令功能是对 8/16 位寄存器或存储器操作数，连符号位一起逐位取反后末位加 1 运算，结果再送回寄存器或存储器。运算结果是操作数的符号位由正数变负数或负数变正数，但绝对值不变。

NEG 指令可用于已知某负数的绝对值求其补码，或已知某负数的补码求其绝对值。例如：

```
MOV AL,56H     ;AL = 01010110B = +86
NEG AL,        ;AL = 10101010B = (-86)补
NEG AL,        ;AL = 01010110B = +86
```

NEG 指令对六个状态标志位均有影响，应注意以下两点：

(1) 执行 NEG 指令，一般都会使 CF=1，除非给定的操作数为零才会使 CF=0。

(2) 当指定的操作数为 80H (−128) 或 8000H (−32768)，则执行 NEG 指令后结果不变，即仍为 80H 或 8000H，但 OF 置 1，其他情况下 OF 均置 0。

4. 比较指令 CMP

指令格式： CMP reg/ mem, reg/ mem/imm

指令功能是用目的操作数减源操作数，结果不送回目的操作数，源和目的操作数的值不变，但影响 6 个状态标志位。对操作数的规定同 SUB 指令。

比较指令主要用来比较两个数的大小关系。可在比较指令执行后，根据标志位的状态判别两操作数大小或相等。判断方法如下：

(1) 相等关系。如果 ZF=1，则两个操作数相等，否则不相等。

(2) 大小关系。分无符号数和有符号数两种情况考虑：

1) 对两个无符号数，根据 CF 标志位的状态确定。若 CF=0，则被减数大于减数（无借位），否则被减数小于减数。

2) 对两个有符号数，情况要稍微复杂一些，判断两个有符号数（A 和 B，做 A−B）大小关系的方法：

当 OF⊕SF=0 时，被减数大于减数 (A>B)；

当 OF⊕SF=1 时，被减数小于减数 (A<B)。

[例 3.5] 在内存数据段 DATA 开始的单元中存放了两个 8 位无符号数，试比较它们的大小，并将大的数送 MAX 单元。

解

```
LEA BX,DATA        ;DATA 偏移地址送 BX
MOV AL,[BX]        ;第一个无符号数送 AL
INC BX             ;BX 加 1 指向第二个数
```

```
        CMP AL,[BX]        ;两个无符号数进行比较
        JNC DONE           ;若 CF = 0(无进位,表示第一个数大),转 DONE
        MOV AL,[BX]        ;否则,第二个无符号数送 AL
DONE:   MOV MAX,AL         ;将较大的无符号数送 MAX
        HLT                ;停机
```

5. 乘法和除法指令

乘/除法指令包括无符号数乘/除法和有符号数乘/除法指令两种。采用隐含寻址方式。乘法隐含目标操作数为 AX（与 DX），源操作数由指令给出；除法隐含了被除数 AX（与 DX），除数由指令给出，除数不能为立即数。

无符号数的乘/除法指令格式：

```
MUL reg/mem     ;(EDX,EAX)/(DX,AX)/AX←reg/mem * EAX/AX/AL
DIV reg/mem     ;AX 除 8 位 reg/mem,商→AL,余数→AH
                ;DX,AX 除 16 位 reg/mem,商→AX,余数→DX
                ;EDX,EAX 除 32 位 reg/mem,商→EAX,余数→EDX
```

例如：

```
MUL CX                ;AX * CX→DX,AX(乘积高 16 位送 DX,乘积低 16 位送 AX)
DIV BL                ;AX/BL 执行后商→AL,余数→AH
MUL BYTE PTR[SI]      ;AX←AL * [SI]
```

对于无符号数乘法，如果乘积的高半部分（在字节相乘时为 AH，在字相乘时为 DX）不为零，则 CF＝OF＝1，代表 AH 或 DX 中包含乘积的有效数字；否则 CF＝OF＝0。对有符号数乘法，若乘积的高半部分是低半部分的符号位的扩展，则 CF＝OF＝0，否则 CF＝OF＝1。对其他标志位均无定义。

[例 3.6]　设 AL＝FEH，CL＝11H，均为无符号数，求 AL 与 CL 的乘积。

解

```
MUL CL    ;AL * CL→AX = 10DEH,因 AH 结果不为零,故 CF = OF = 1
```

当 DIV 指令的被除数不是除数的双倍长度时，则应将其扩展成双倍长度。当除数为零或商超过了保存商的累加器允许的数值范围时，会出现溢出，并产生一个 0 型中断，CPU 会进入溢出错处理程序。除法指令对六个标志位均无影响。

[例 3.7]　计算 7FA2H÷03DDH。

解

```
MOV AX,7FA2H          ;AX = 7FA2H
MOV BX,03DDH          ;BX = 03DDH
CWD                   ;DX,AX = 00007FA2H
DIV BX                ;商 AX = 0021H, 余数 = 0025H
```

6. BCD 码（十进制数）调整指令

人们习惯使用十进制数，在计算机码制中，专门设立了用二进制表示十进制数的 BCD 码。BCD 码有两种形式：压缩 BCD 码和非压缩 BCD 码。在 80X86 系列中，有相对应加、减、乘、除指令的 BCD 调整指令。

(1) BCD 码加法调整指令。对应压缩 BCD 码和非压缩 BCD 码的加法十进制调整指令有 DAA 和 AAA。两者的指令助记符后面均不带操作数。但 DAA 指令隐含操作数 AL 和 AH，AAA 指令隐含 AL 操作数。

BCD 码运算步骤是：首先对两个 BCD 数，按一般二进制数加法运算指令进行运算；然后用相应的调整指令对上述结果进行调整。所以，调整指令必须紧跟在二进制运算类指令的后面。下面举例说明 BCD 码加法运算。

压缩 BCD 码：

```
ADD AL,reg/mem/imm8
DAA                    ;完成压缩 BCD 数的加法运算

ADC AL, reg/mem/imm8
DAA                    ;带进位位的 BCD 数加法指令
```

非压缩 BCD 码：

```
ADD AL, reg/mem/imm8
AAA                    ;非压缩 BCD 数的加法运算

ADC AL, reg/mem/imm8
AAA                    ;带进位位的非压缩 BCD 数加法运算
```

那么为什么要进行调整以及怎样进行调整呢?

因为压缩 BCD 码是由 4 位二进制数表示一位十进制数，即用 0000～1001 表示十进制数 0～9，最大值为 9。当两个压缩 BCD 数先按二进制数运算时，可能会出现大于 9 的结果。这样的运算结果已不是压缩 BCD 码了，称为非法码或冗余码，所以必须对存放在 AL 中的中间结果进行调整，使其成为压缩 BCD 码。

由于 4 位二进制数是逢 16 进 1，而 BCD 数是逢 10 进 1，所以两者差 6。

压缩 BCD 码加法调整指令 DAA 调整过程：

1) 两数相加，高 4 位与低 4 位的和均在 0～9 之间，则无需修正或称加 00H 修正。

2) 若两数相加后，低 4 位的和大于 9（即出现非法码）或低 4 位向高 4 位有进位 (AF=1)；但高 4 位相加的和在 0～9 之间，则 DAA 指令对中间结果进行加 06H 修正。

3) 若两数相加后，高 4 位的和大于 9 或有向更高位进位 (CF=1)；而低 4 位的和在 0～9 之间，则 DAA 指令对中间结果进行加 60H 修正。

4) 若两数相加后，低 4 位、高 4 位结果均大于 9 或都有进位位 (AF=1，CF=1)；或高 4 位等于 9，而低 4 位有加 6 修正情况 (大于 9 或有进位 AF=1)，则 DAA 指令对中间结果进行加 66H 修正。

现举例说明如下：

1)

	高 4 位	低 4 位	
28H=	0010	1000B	
+53H=	0101	0011B	
81H	0100	1011B	中间结果
+	0000	0110B	加 06H 修正
	1000	0001B	

由于低 4 位出现非法码需修正，高 4 位无需修正，因此为加 06H 修正。

2）

	高 4 位	低 4 位	
84H=	1000	0100B	
+94H=	1001	0100B	
178H	10001	1000B	中间结果
+	0110	0000B	加 60H 修正
	10111	1000B	

由于高 4 位向更高位进位需修正、低 4 位无需修正，因此为加 60H 修正。

（2）BCD 码减法调整指令。DAS/AAS 指令是对两个压缩 BCD 数/非压缩 BCD 数相减后的差进行修正，得到压缩 BCD 码/非压缩 BCD 码表示的结果（差）。DAS 指令隐含的操作数为 AL，其执行结果影响 SF、ZF、AF、PF、CF 和 OF。AAS 指令隐含的操作数为 AL 和 AH，其执行结果只影响 AF 和 CF，而其他 4 位标志为不确定，没有意义。

减法调整指令也都必须紧跟在减法指令之后。

用于压缩 BCD 码	用于非压缩 BCD 码
SUB AL,reg/mem/imm8 DAS	SUB AL, AL,reg/mem/imm8 AAS
SBB AL, AL,reg/mem/imm8 DAS	SBB AL, AL,reg/mem/imm8 AAS

DAS 指令调整的操作过程如下：

1）两数相减后，若 AL 的低 4 位大于 9 或辅助借位标志 AF＝1，低 4 位应减 6 修正；否则不需修正，AF 标志不变。

2）两数相减后，若 AL 的高 4 位大于 9 或借位标志 CF＝1，高 4 位应减 6 修正；否则不需修正，AF 标志不变。

3）若两数相减，结果为负数（即被减数小于减数），执行 DAS 调整指令后 CF＝1，表明从 AL 寄存器中的差值是负数，它是相对于模 100 的 BCD 的补数。

[例 3.8]　完成 BCD 码 83H－39H 的减法运算。

解　程序段：

```
MOV AL, 83H     ;AL = 83H
MOV BL, 39H     ;BL = 39H
SUB AL, BL      ;AL = 4AH,按二进制数格式进行减法,得到中间结果
DAS             ;因 AL 低 4 位大于 9,应减 6 修正;高 4 位不要减 6 修正,
                ;调整运行后,AL = 45H
```

[例 3.9]　完成 BCD 码 34H－85H＝？的减法运算。

解　程序段：

```
MOV AL, 34H     ;AL = 34H
MOV BL, 85H     ;BL = 85H
SUB AL, BL      ;AL = AFH,CF = 1,AF = 1
DAS             ;AL 高 4 位及低 4 位均需修正,即减 66H,结果 AL = 0100 1001B = 49H
```

由于 CF=1，表示有借位，结果为负数，表明存于 AL 中的 49H 是负数，相对于模 100H 而言，49H 是－51H 的补码，即 34H－85H=－51H，存于 AL 中。

7. 其他算术运算指令

除以上指令外，80X86 指令系统还有其他一些算术运算指令，如表 3.4 所示。

表 3.4　　其他算术运算指令

指令类型	汇编格式	指令的操作	举　例
交换加法指令	XADD reg/mem，reg	先将 reg，mem 与 reg 互换；然后完成 reg/mem+reg，和值送 reg/mem。根据结果影响 6 个标志位	XADD AX，CX ；AX 与 CX 内容互换， ；AX+CX→AX
比较交换指令	CMPXCHG reg/mem，reg	将目的操作数 reg/mem 减累加器 AL、AX 或 EAX，源和目的操作数值不变。若相等（ZF=1），则将源操作数 reg 复制到目的操作数 reg/mem 中；若不相等（ZF=0），则目的操作数 reg/mem 送入 AL、AX 或 EAX 中	设 AL=11H，BL=22H，(2000H) =12H；执行 CMPXCHG [2000H]，BL 执行后 AL = 12H，(2000H) =12H，BL=22H
有符号数乘法指令	IMUL reg/mem IMUL reg，reg/mem/ imm IMUL reg，reg/mem，imm	单操作数 IMUL 中，被乘数 AL、AX、EAX，乘数 reg/mem，乘积对应放在 AX、DX，AX、EDX，EAX 中； 双操作数 IMUL 中，rge←reg * reg/mem/imm； 三操作数 IMUL 中，reg←reg/mem * imm	设 AL=FEH，CL=11H 为有符号数，执行： IMUL CL ；AX=FFDEH=－34。 因 AH 中内容为 AL 中的符号扩展，故 CF=OF=0
有符号数除法指令	IDIV reg/mem	功能和操作与 DIV 指令类似，商和余数均为有符号数，且余数符号与被除数符号相同	执行：IDIV CX ；DX，AX 中的 32 位数除以 CX，商在 AX 中，余数在 DX 中
非压缩 BCD 码乘法调整指令	AAM	对两个非压缩的 BCD 码相乘结果（AX 中）进行调整。得到正确的非压缩 BCD 码结果，乘积的高位在 AH 中，乘积的低位在 AL 中	MOV AL，07H MOV BL，09H MUL BL ；AX=003FH AAM ；结果：AX=0603H，即非压缩 BCD 数 63
非压缩 BCD 码除法调整指令	AAD	在进行非压缩 BCD 码除法之前执行。将 AX 中的非压缩 BCD 码（十位数放在 AH，个位数放在 AL）调整为二进制数，并将结果放 AL 中	MOV AX，0203H；AX=23 MOV BL，4 AAD ；AX=0017H DIV BL ；结果： ；AH=03H，AL=05H

3.3.3　逻辑运算与移位指令

80X86 提供了丰富的逻辑运算及移位指令，包括与、或、非、异或、测试（TEST）和左移、右移、循环左/右移指令，这类指令除逻辑非（NOT）指令外，都会根据操作结果影响标志寄存器的相应标志位。

1. 逻辑运算指令

逻辑运算指令有逻辑与（AND）、或（OR）、非（NOT）、异或（XOR）、测试（TEST）共 5 条指令。逻辑与、或、异或指令是将目的操作数与源操作数分别按位进行“与”、“或”、“异或”逻辑操作，结果送目的操作数的相应位；逻辑非指令是将目的操作数的每位取反，结果送目的操作数中；测试指令是将目的操作数与源操作数按位进行逻辑“与”操作，源和目的操作数的值不变，根据“与”结果置标志寄存器的相应标志位。

源和目的操作数均可为 8/16/32 位寄存器或存储器操作数，源操作数还可以是立即数。同一指令中两个操作数的位数必须一致，并且不能同时为存储器操作数。

NOT 指令的执行对所有标志位都不影响。其他指令的执行都会使 CF＝OF＝0，AF 值不定，并对 SF、PF 和 ZF 有影响。

（1）逻辑与指令 AND。

指令格式：

AND reg/mem，reg/mem/imm；reg/mem∧reg/mem/imm→reg/mem，不能同时为 mem。

AND 指令常用于屏蔽某些位（使该位置 0），其余位不变。欲屏蔽位和“0”进行“与”运算，不变位和“1”进行“与”运算。

例如，将 AL 中 0～9 的 ASCII 码转换成相应的二进制数。

```
MOV AL,39H          ;9 的 ASCII 代码→AL
AND AL,0FH          ; AL 高 4 位被屏蔽,低 4 位不变,即 AL = 00001001B = 09H。
```

AND 指令还可用于检测某一位是否是 1。如通过检测 AL 中 D_1 位是否是 1 来控制程序走向，程序段如下：

```
AND AL,00000010B     ;AL 的 D1 位不变;
JZ LOOP1             ;若 AL 的 D1 = 0 则程序转 LOOP1,若 D1 = 1 则程序顺序执行。
      ⋮
```

AND 指令的源和目的操作数相同，则进行“与”操作后，操作数不变，但影响 6 个状态标志位，并使 CF＝OF＝0。例如：AND AL，AL。

（2）逻辑或指令 OR。

指令格式：

OR reg/mem，reg/mem/imm；reg/mem∨reg/mem/imm→reg/mem，不能同时为 mem

OR 指令常用于将某些位置 1，其余位保持不变。例如，将 AL 中的 09H（可看成是非压缩 BCD 码）转换成相应的 ASCII 码。程序段如下：

```
MOV AL,09H          ;非压缩的 BCD 码 09→AL
OR AL,30H           ;AL = 39H = '9'
```

OR 指令的源和目的操作数相同，则进行“或”操作后，操作数不变，但影响 6 个状态标志位，并使 CF＝OF＝0。例如：OR AL，AL。

（3）逻辑异或指令 XOR。

指令格式：

XOR reg/mem，reg/mem/imm；reg/mem⊕reg/mem/imm→reg/mem，不能同时为 mem

XOR 指令常用于将寄存器内容清 0；测试某一数是否与另一数相等。例如：

```
①XOR AX,AX      ;AX 的内容清 0
②XOR AL,22H     ;测试 AL 的内容是否等于 22H,若是,执行 XOR 指令后 AL = 00H
   JZ LOOP1     ;判断,若 AL = 0,则程序转 LOOP1 处执行;否则顺序执行
```

XOR 指令可用于对某些特定位“求反”，而其余位不变。只要将“求反”位与 1 进行 XOR 运算，不变位与 0 进行 XOR 运算即可。

例如，已知 AL=11000101B，要求对高 4 位求反，低 4 位不变，程序段如下：

```
MOV BL,0F0H     ;高四位是全 1 低四位是全 0 的数是 11110000B = F0H
XOR AL,BL       ;执行 XOR 指令后,AL = 00110101B
```

(4) 测试指令 TEST。

指令格式：TEST reg/mem，reg/mem/imm；reg/mem∧reg/mem/imm，不能同时为 mem

其所完成的操作是根据与结果状态置标志寄存器的相应标志位，源和目的操作数不变。

测试指令 TEST 常用于对某特定位进行测试，后跟条件转移指令，共同完成对某特定位 0 或 1 的判断，以实现程序的相应转移。例如：

检测 AL 中的 D_2 位是否为 1，若 $D_2=1$，则转移。程序段如下：

```
TEST AL,04H     ;04H = 00000100B,检测 D2 = 1?
JNZ LOOP1       ;结果不为 0,即 D2 = 1,则程序转 LOOP1 处执行,否则顺序执行
```

[例 3.10] 从 3000H 开始的内存单元中存放有 64 个有符号数，要求统计其中负数的个数，并将统计结果存入 NUM 单元。

解 程序段如下：

```
      XOR DX,DX       ;清 DX 内容,DX 用于存放负数个数
      MOV SI,3000H    ;数据区起始地址存 SI,即地址指针
      MOV CX,40H      ;数据区数据个数送 CX,CX 为计数器,控制程序循环次数
AGN:  MOV AL,[SI]     ;取数据区一个数送 AL
      INC SI          ;地址指针 SI 加 1
      TEST AL,80H     ;80H = 10000000B,检测 D7 = 1?,是 1 为负数
      JZ NEXT         ;结果为 0,即 D7 = 0,AL 为正数,则程序转 NEXT 处执行
      INC DX          ;结果不为 0,即 D7 = 1,AL 为负数,则 DX 加 1
NEXT: DEC CX          ;计数器 CX 减 1
      JNZ AGN         ;若 CX≠0,则继续检测下一个数
      MOV NUM,DX      ;否则,负数个数送 NUM 单元
      HLT             ;停机
```

(5) 逻辑非指令 NOT。

指令格式：NOT reg/mem ；将操作数 reg/mem 按位取反再送回 reg/mem

NOT 指令的操作对标志位无影响。例如：

```
NOT AX                  ;AX 的内容按位取反,结果再送回 AX
NOT WORD PTR[SI]        ;将[SI]所指两个单元中的内容按位取反,结果再送回这两个单元
```

2. 移位指令

移位指令有两种类型：一是逻辑移位指令，移位时连同操作数的符号位一起移位，主要

用于无符号数或非数值数进行操作；二是算术移位指令，移位过程中保留原操作数的符号位状态不变，主要用于对带符号数进行移位操作。

（1）逻辑与算术移位有如下四条指令：

1）逻辑左移指令：SHL　reg/mem，imm8/CL　；

2）算术左移指令：SAL　reg/mem，imm8/CL　；

3）逻辑右移指令：SHR　reg/mem，imm8/CL　；

4）算术右移指令：SAR　reg/mem，imm8/CL　；

上述四条指令都是将目的数按源操作数（8 位立即数或 CL 寄存器中的值）指出的移位次数进行相应移位，并将移出的位送入进位标志 CF 中，移位结束，结果存于目的操作数中。

其中，SHL/SAL 是逻辑/算术左移指令，这两条指令的功能实质上是相同的。操作数每左移一位，在最低有效位 LSB 上补 0，把最高有效值 MSB 移进 CF，如图 3.16 所示。

注意：Imm8/CL 为移位次数，对于 8086/8088 移位次数用立即数 Imm8 时只能是 1，若用 CL 寄存器，CL 最大值是 255，即最多可移位 255 次。对于 80386/80486 移位次数最大为 31，可用立即数表示，也可用 CL 寄存器。

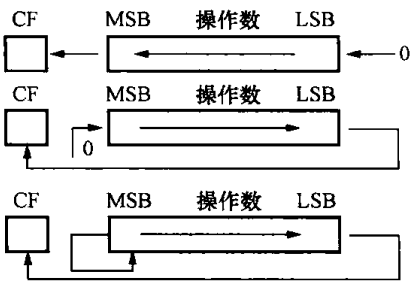


图 3.16　SHL/SAL、SHR、SAR 移位操作示意图

例如：

```
SHL AL,1                ;AL 寄存器左移 1 位,最高位 MSB 移入 CF
SHL AL,CL               ;AL 寄存器左移 n 位,n 是 CL 的内容
SHL BETA[SI],1          ;从有效地址 EA = BETA + SI 取出字节操作数左移 1 位后,再送入原单元中
SAL AX,CL               ;AX 寄存器左移 n 位,n 是 CL 的内容
SAL WORD PTR[BX + 100],1 ;从有效地址 EA = BX + 100 开始连续两个单元取出字操作数
                        ;左移 1 位后,再送入原两个单元中
```

在移位次数为 1 的情况下，如果移位结束后，CF 和操作数的最高位相等，则 OF=0；否则 OF=1。若移位次数大于 1 时，OF 为不确定状态。指令还根据结果影响标志位 PF、SF 和 ZF。

左移指令常用于实现乘以 2^n 的操作。因为将一个二进制无符号数左移一位，相当于将该数乘 2，因此，可以利用左移指令实现乘以 2^n 的运算。

[例 3.11]　以 DATA 为首址的连续单元中的 16 位无符号数乘以 10。

分析：因为 $10x = 8x + 2x = 2^3x + 2^1x$，所以可以用左移指令实现该乘法。

解　程序段如下：

```
LEA SI,DATA     ;DATA 单元的偏移地址送 SI,即地址指针
MOV AX,[SI]     ;取被乘数送 AX
SHL AX,1        ;AX = DATA * 2
MOV BX,AX       ;AX 暂存入 BX
MOV CL,2        ;移位次数 2 送 CL
SHL AX,CL       ;AX = DATA * 8
ADD AX,BX       ;AX = DATA * 10
HLT             ;停机
```

SHR 是逻辑右移指令，用于对无符号数右移。每移一次，操作数各位依次右移，最低位移入 CF，而最高位则补 0。

SAR 是算术右移指令，用于对有符号数右移。每移一次，操作数各位依次右移，最低位移入 CF，而最高位是符号位保持不变。

当移位位数为 1 时，对于 SHR 指令，移位之后新的最高位和次高位不相等，则 OF＝1，否则 OF＝0，若移位次数不为 1，则 OF 状态不确定；SAR 指令对标志位 CF、PF、SF 和 ZF 有影响，但不影响 OF、AF。

同样，算术右移指令可用于完成有符号数除以 2^n 的运算。

（2）双精度左移/右移指令。

双精度左移指令：SHLD reg/mem，reg，imm8/CL

双精度右移指令：SHLRD reg/mem，reg，imm8/CL

指令功能：将 reg/mem 和 reg 两个 16/32 位操作数连接成双精度数（32/64 位）向左或向右移位，移位次数是 imm8/CL 的内容。移位过程是将 reg 内容移入 reg/mem，而 reg 本身不变。进位位 CF 中是 reg/mem 移出的最后一位。双精度移位示意图如图 3.17 所示。

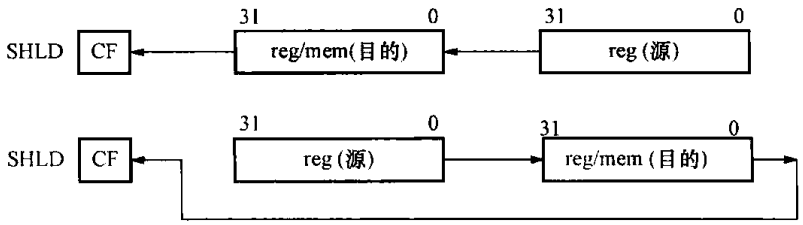


图 3.17　双精度移位示意图

```
SHLD TABLE[EBX],EAX,8      ;将 TABLE[EBX]指向的双字单元内容左移 8 位,移空位由 EAX
                           ;内容左移 8 位填入,但 EAX 内容不变
SHRD EAX,EDX,5             ;表示将 EAX 中内容右移 5 位,移空位由 EDX 内容右移 5 位填入,
                           ;但 EDX 内容不变
```

这两条指令对标志位 SF、ZF、PF、CF 有影响，OF 和 AF 位不确定。

双精度移位指令可以用来实现对一个任意的、未对齐的位串进行快速移动、嵌入和删除操作。

（3）循环移位指令。

1）循环左移指令：ROL reg/mem，imm8/CL

2）循环右移指令：ROR reg/mem，imm8/CL

3）带进位循环左移指令：RCL reg/mem，imm8/CL

4）带进位循环右移指令：RCR reg/mem，imm8/CL

指令功能：ROL 和 ROR 指令分别将目的操作数按源操作数给出的移位次数进行循环左移和右移。每移位一次，目的操作数移出的位同时送入标志位 CF 和另一端空出的位中，如图 3.18（a）、（c）所示。

RCL 和 RCR 指令分别将目的操作数按由源操作数给出的移位次数进行带进位位的循环左移和循环右移。每当目的操作数移一次，移出的位送入标志位 CF，而 CF 原有的内容则送入另一端空出的位，如图 3.18（b）、（d）所示。

对于 8086/8088CPU 上述四条指令源操作数中 imm8 只能是 1。四条循环移位指令对标志位 OP、SF、ZF、AF、PF 有影响。

循环移位指令与非循环移位指令不同，循环移位后，操作中原来各位数的信息不会丢失，而只是改变了位置而已，如果需要还可恢复（反向移动即可）。

利用循环移位指令可以测试操作数某一位的状态（即 0 或 1）。

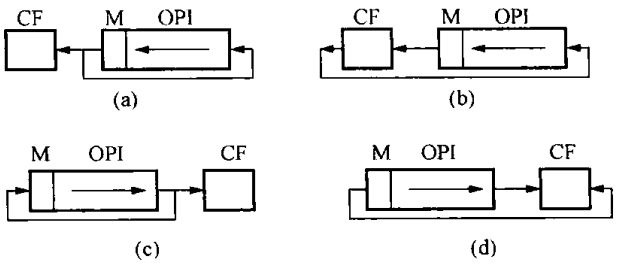


图 3.18　循环移位指令功能示意
（a）ROL 指令；（b）RCL 指令；
（c）ROR 指令；（d）RCR 指令

[例 3.12]　测试 BL 寄存器中第 4 位的状态，并保持原内容不变。

解　程序段如下：

```
        MOV CL,4          ;移位次数送 CL
        ROL BL,CL         ;BL 第 4 位移入 CF
        JNC ZERO          ;没借位或进位,即 CF = 0 则转到 ZERO
        ROR BL,CL         ;否则恢复原 BL 的内容
        …                 ;
ZERO:   ROR BL,CL         ;恢复原 BL 的内容
        …                 ;
        HLT               ;停机
```

3.3.4　串操作指令

通常把存放在存储器某一连续地址区域的字符或数据称为字符串或数据串。由字节组成的字符串为字节字符串，简称字节串；由字组成的字符串称字字符串，简称字串；由双字组成的字符串称为双字串。在存储器中存放字符串各字符时可以是二进制数、BCD 码或 ASCII 码等信息。

串操作指令可处理字节串、字串、双字串，最大串长度为 64KB。它们有以下共同点：

（1）80X86 串操作指令包括串传送、串装入、串存储、串比较、串扫描、串输入、串输出等。指令中源串和目的串的存储及寻址方式都隐含规定：源串要放在数据段（DS 段），用（E）SI 间接寻址方式访问数据段，即约定以 DS:（E）SI 来寻址源串；目的串要放在附加段（ES 段），用（E）DI 间接寻址方式访问数据段，即约定以 ES:（E）DI 来寻址目的串。

（2）源串的段寄存器 DS 可以通过加段超越前缀来改变，而目的串的段寄存器不能改变。

（3）（E）SI 和（E）DI，在每次操作后都将根据方向标志 DF 的值自动增量（DF＝0 时）或减量（DF＝1 时），以指向下一个字节、字或双字存放的地址。增量/减量的大小是：字节串为加/减 1；字串为加/减 2；双字串为加/减 4。

（4）串长度值放在（E）CX 寄存器中。若在串操作指令前加重复前缀，则用（E）CX 寄存器作为重复次数计数器，指令每执行一次串操作，（E）CX 内容自动减 1，直到减为 0 止。

1. 串传送指令

```
MOVSB  ;字节串数据传送。(ES*16+(E)DI) ← (DS*16+(E)SI),(E)SI ← (E)SI±1, (E)DI ← (E)DI±1
MOVSW  ;字串数据传送。(ES*16+(E)DI)←(DS*16+(E)SI), (ES*16+(E)DI+1)←(DS*16+(E)SI+1),
       ;(E)SI ← (E)SI±2, (E)DI ← (E)DI±2
MOVSD  ;双字串数据传送。(ES*16+(E)DI)←(DS*16+(E)SI),(ES*16+(E)DI+2)←(DS*16+(E)SI+2)
       ;(E)SI ←(E)SI±4, (E)DI ←(E)DI±4
MOVS mem1,mem2    ;将一个由 DS:(E)SI 指出的存储器源操作数传送到由 ES:(E)DI 指出的
                  ;目的存储器单元中,并修改有效地址。可以传送字节、字或双字数据。
                  ;此指令不常用
```

这四种串传送指令要求源与目的串的数据类型必须一致。操作的结果均不影响标志位。

[例 3.13] 将源串首地址为 2000H：1200H 开始的 100 个字节转送到首地址为 6000H：4000H 开始的内存单元中。

解 程序段如下：

```
        MOV AX,2000H      ; 源串段基地址通过 AX 送 DS
        MOV DS,AX
        MOV AX,6000H      ; 目的串段基地址通过 AX 送 ES
        MOV ES,AX
        MOV SI,1200H      ; 源串偏移地址→SI
        MOV DI,4000H      ; 目的串偏移地址→DI
        MOV CX,100        ; 数据串长度→CX
        CLD               ; 方向标志 DF=0,地址增量(若用 STD 指令则 DF=1,地址减量)
LOOP1:  MOVSB             ; 从源串传送一个字节到目的串,并 SI、DI 分别加 1
        DEC CX            ; 每传一次,长度 CX 减 1
        JNZ LOOP1         ; 100 个字节数据没有传送完(CX≠0),转 LOOP1 继续传送
        HLT               ; 否则 100 个字节数据传送完毕,停机结束串传送,对标志位无影响
```

串操作类的指令还可利用有 REP 重复前缀的指令格式。例如，“REP MOVSB” 就可完成连续将源串数据存入目的串中，直到计数器 (E) CX 等于 0 为止。上述程序段如下：

```
MOV AX,2000H      ; 源串段基地址通过 AX 送 DS
MOV DS,AX
MOV AX,6000H      ; 目的串段基地址通过 AX 送 ES
MOV ES,AX
MOV SI,1200H      ;
MOV DI,4000H      ;
MOV CX,100        ;
REP MOVSB         ;连续传送,直到 100 个字节传送完为止
HLT
```

串传送指令加上重复前缀 REP MOVSB、REP MOVSW、REP MOVSD 指令所完成的功能示意图如图 3.19 所示。

2. 串比较指令

串比较指令有四种格式：

```
CMPS mem1,mem2      ;不常用
CMPSB               ;字节串比较
CMPSW               ;字串比较
CMPSD               ;双字串比较
```

串比较指令的功能是：将 DS：(E) SI 所指的源串中的字节、字或双字与 ES：(E) DI 所指的目的串中的字节、字或双字做减法，若相等则 ZF=1，否则 ZF=0，且源串与目的串的内容不变。每比较一次后，地址指针 (E) SI 和 (E) DI 按 DF 方向标志自动修改。根据执行的结果影响 OF、SF、ZF、AF、PF 和 CF 标志位。串比较指令功能示意图如图 3.20 所示。

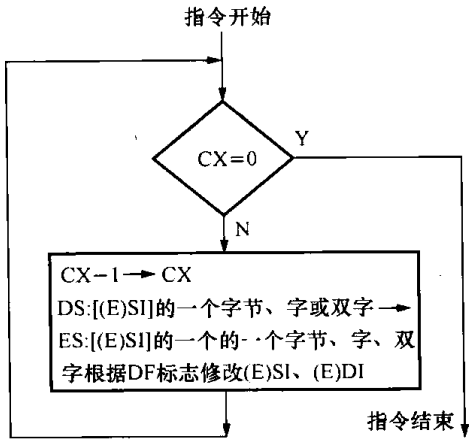


图 3.19　串传送指令功能示意图

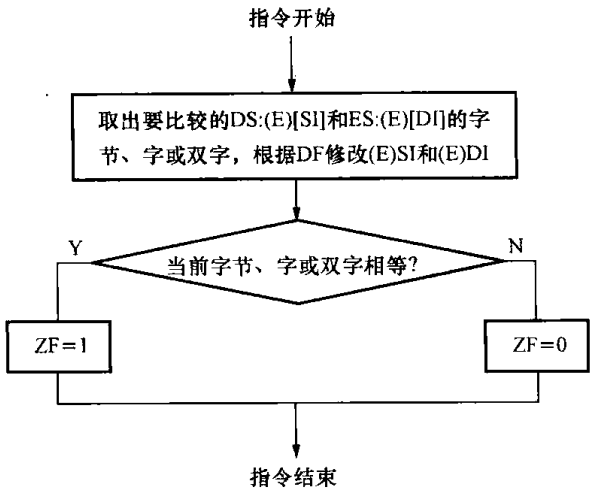


图 3.20　串比较指令功能示意图

通常，在串比较指令后紧跟一个条件转移指令，用以测试比较结果。这类指令常用于比较串数据是否对应相等。

[例 3.14]　分别从 SOURCE 和 DEST 开始存放两个串长为 100 个字节的字符串。试比较两个字符串是否相同，若全部相同，则使 BX=0；若不相同，则将源串中第一个不相同字符的偏移地址送 BX，不相同字符送 AL 中。

```
        CLD             ;置方向标志 DF = 0,为增量
        LEA SI,SOURCE   ;取源串 SOURCE 的偏移地址(首地址)→SI
        LEA DI,DEST     ;取目的串 DEST 的偏移地址(首地址)→DI
        MOV CX,100      ;字节串长度→CX
NEXT:   CMPSB           ;字节串比较
        JNZ STOP        ;找到不相同字符(ZF = 0),程序转 STOP
        DEC CX          ;否则字符相同(ZF = 1),继续比较,长度减 1
        JNZ NEXT        ;字符串没有全部比较完毕(CX≠0),继续比较
        MOV BX,00H      ;两字符串完全相等,00H→BX
        JMP DONE        ;程序转 DONE, 继续
STOP:   DEC SI          ;求不相同字符的源串偏移地址,并将其送 BX
        MOV BX,SI
        MOV AL,[BX]     ;将不相同的源串字符→AL 保存
DONE:   HLT
```

3. 串扫描指令

串扫描指令有四种格式：

```
SCAS mem    ;mem 为内存目的串,此格式不常用
SCASB       ;字节串扫描,AL-(DS*16+(E)DI),(E)DI←(E)DI±1
SCASW       ;字串扫描,AX-(DS*16+(E)DI+1),(DS*16+(E)DI),(E)DI←(E)DI±2
SCASD       ;双字串扫描,EAX-(DS*16+(E)DI+2),(DS*16+(E)DI),(E)DI←(E)DI±4
```

串扫描指令的功能与 CMPS 的指令类似，也是进行比较操作。它是完成累加器（AL、AX 或 EAX）与目的串（由 ES：DI 指定）中字节、字或双字进行比较，根据比较结果影响标志位，但累加器和目的串内容不变。

串扫描指令常用来在一个字符串中搜索特定的关键字，把要找的关键字放在 AL、AX 或 EAX，再用此类指令与字符串中各字符逐一比较。

[**例 3.15**] 在 ES：STRING 附加数据区首地址开始，按地址减量方向顺序存放有 100 个字节字符串，在其中查找是否有“A”字符；若有，则将“A”字符所在偏移地址送 BX，否则置 BX=0。

解 程序段如下：

```
        STD                     ;置 DF=1,地址减量方向
        MOV DI,OFFSET STRING    ;目的串首址的偏移地址送 DI
        MOV CX,100              ;串长度 100 送 CX
        MOV AL,'A'              ;字符 A 的 ASCII 码→AL
NEXT:   SCASB                   ;字节串扫描
        JZ STOP                 ;ZF=1,找到与"A"相同的字符,转 STOP
        DEX CX                  ;没找到(ZF=0),长度减 1
        JNZ NEXT                ;CX≠0,没全部搜索完,继续查找
        JMP DONE                ;CX=0 全部搜索完,没找到转 DONE 处执行
STOP:   DEC DI                  ;字符 A 所在偏移地址→BX
        MOV BX,DI
        JMP EXIT
DONE:   MOV BX,0000H            ; 0000H→BX
EXIT:   HLT
```

4. 串装入指令

有四种串装入指令格式：

```
LODS mem    ;mem 是源串。该指令格式不常用
LODSB       ;字节装入,(DS*16+(E)SI)→AL,(E)SI←(E)SI ±1
LODSW       ;字装入,(DS*16+(E)SI+1),(DS*16+(E)SI)→AX,(E)SI←(E)SI ±2
LODSD       ;双字装入,(DS*16+(E)SI+2),(DS*16+(E)SI)→EAX,(E)SI←(E)SI± 4
```

指令功能是将 DS：（E）SI 所指源串字节、字或双字装入累加器（AL、AX 或 EAX）中，并按标志位 DF 的状态及源串类型自动修改（E）SI 值。

串装入指令操作结果不影响标志位，且一般不连续重复执行，因为每重复一次，累加器的内容就被后一次装入的字符所取代。

[例 3.16]　以 MEM 为首地址的内存区域中有 10 个以非压缩 BCD 码形式存放的十进制数，它们的值可能是 0～9 中的任意一个，现编程序将这 10 个数顺序显示在屏幕上。

解　程序段如下：

```
        LEA SI,MEM      ;源串偏移地址送 SI
        MOV CX,10       ;串长度 10 送 CX
        CLD             ;置 DF = 0，地址增量
        MOV AH,02H      ;功能号(表示单字符显示输出)→AH
NEXT:   LODSB           ;取一个 BCD 码→AL,且 SI + 1→SI
        ADD AL,30H      ;BCD 码转换为对应的 ASCII 码,数字加 30H 就是其 ASCII 码值
        MOV DL,AL       ;字符的 ASCII 码送 DL
        INT 21H         ;输出显示
        DEC CX          ;CX-1→CX
        JNZ NEXT        ;ZF = 0(CX-0),则重复
        HLT
```

5. 串存储指令

有四种串存储指令格式：

```
STOS mem    ; mem 为内存目的操作数,该指令格式不常用
STOSB       ;字节串存储,AL→(ES * 16 + (E)DI),(E)DI←(E)DI ± 1
STOSW       ;字串存储,AX→(ES * 16 + (E)DI + 1),(ES * 16 + (E)DI),(E)DI←(E)DI ± 2
STOSD       ;双字串存储,EAX→(ES * 16 + (E)DI + 2),(ES * 16 + (E)DI),(E)DI←(E)DI ± 4
```

串存储指令功能是将累加器（AL、AX 或 EAX）的值存入 ES：（E）DI 所指的目的串中，并按标志位 DF 的值及目的串的类型自动修改（E）DI。

指令前加重复前缀 REP，可用于对一个连续内存区赋同一值（即对内存区域初始化）。

[例 3.17]　把 6000H：1200H 单元开始的 100 个字存储单元内容清零。

解　程序段如下：

```
MOV AX,6000H        ;
MOV ES,AX           ;目标串段地址送 ES
MOV DI,1200H        ;存储目的串的偏移地址送 DI
MOV CX,100          ;串长度送 CX
CLD                 ; DF = 0,自动增量,从低地址到高地址的方向先进行存储
MOV AX,0            ;要存入的目的串内容 0 送 AX
REP STOSW           ;将 100 个单元清零
HLT
```

6. 重复前缀

重复前缀不能单独使用，只能加在串操作指令之前，用来控制跟在其后的串操作指令，使之重复执行。重复前缀串操作指令，其执行速度要比通常用循环程序所能达到的速度快得多。有重复前缀的串比较、串扫描指令功能示意如图 3.21 所示。共有 5 条：

（1）REP 串操作指令。它使 CPU 重复执行其后的串指令，重复次数由（E）CX 寄存器指定，直到 CX＝0 时结束。

（2）REPZ/REPE CMPS/SCAS。相等且 CX≠0 时 CPU 重复执行其后的串指令；否则 ZF＝0 或（E）CX＝0 结束，重复次数由（E）CX 寄存器指定。

（3）REPNE/REPNZ CMPS/SACS。不相等且 CX≠0 时 CPU 重复执行其后的串指令；否则 ZF＝1 或（E）CX＝0 结束，重复次数由（E）CX 寄存器指定。

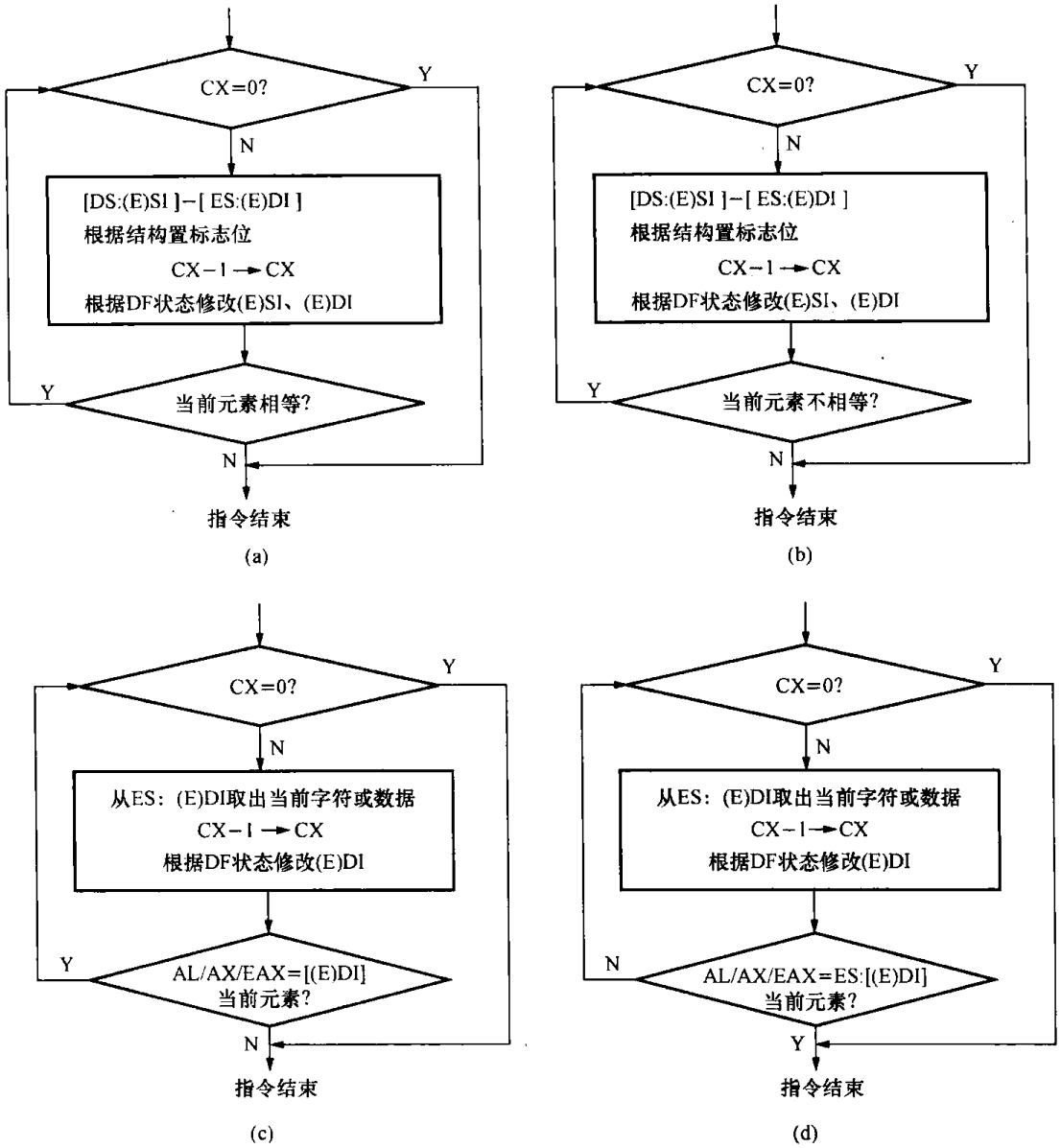


图 3.21 有重复前缀串比较、串扫描指令功能示意图

REPE CMPSB　　REPNE CMPSB
(a)REPE CMPSW 指令功能示意图；(b)REPNE CMPSW 指令功能示意图；
REPE CMPSD　　REPNE CMPSD
REPE SCASB　　REPNE SCASB
(c)REPE SCASW 指令功能示意图；(d)REPNE SCASW 指令功能示意图
REPE SCASD　　REPNE SCASD

[例 3.18] 在 ES 段中从 2000H 单元开始存放了 10 个字符，寻找其中有无字符 A。若有则记下搜索次数（次数放 DATA 1 单元），并记下存放字符 A 的地址（地址放 DATA 2 单元）。

解 程序段如下：

```
MOV DI,2000H          ;目的字符串首地址送 DI
```

```
        MOV BX,DI          ;首地址暂存 BX
        MOV CX,0AH         ;串长度送 CX
        MOV AL,'A'         ;关键字符 A 的 ASCII 码送 AL
        CLD                ; DF = 0,自动增量,每次扫描后地址指针递增
        REPNZ SCASB        ;扫描字符串,直到找到字符 A 或 CX = 0
        JZ FOUND           ;若找到则转移
        MOV DI,0           ;没找到要搜索的关键字,使 DI = 0
        JMP DONE
FOUND:  DEC DI             ;DI - 1,指向找到的关键字所在地址
        MOV DATA2,DI       ;将关键字地址送 DATA2 单元
        INC DI
        SUB DI,BX          ;用找到的关键字地址减去首地址得到搜索次数
DONE:   MOV DATA1,DI       ;将搜索次数送 DATA1 单元
        HLT
```

此例中，退出 REPNZ SCASB 串循环有两种可能：一种可能是已找到关键字，从而退出，此时 ZF=1；另一种可能是没搜到关键字，但串已搜索完毕，从而退出，此时 ZF=0，CX=0。所以退出之后，可根据对 ZF 标志的检测来判断是属于哪种情况。

执行 REPNZ SCASB 操作时，每比较一次，目的串指针自动加 1（因 DF=0），所以找到关键字后，需将 DI 内容减 1 才能得到关键字所在地址。

7. 其他串和位操作类指令

80386 及以上微处理器还增加了一些串和位操作类指令，如表 3.5 所示。

表 3.5　80386 及以上微处理器增加的主要串和位操作指令

指令类型	汇编格式	指令的操作	举　例
串输入指令	INS mem，DX INSB INSW INSD	从 DX 指定端口地址读入数据送入 ES:(E) DI 所指的目的串（mem）中，同时根据 DF 值和目的串类型选择自动增/减及修改(E) DI。后三条指令同样分别为字节串、字串、双字串输入指令。 可利用 REP 重复前缀连续从 DX 所指端口输入串数据存入目的串中，直到（E）CX 等于零为止。指令执行对标志位无影响	要求从连续输入 10 个字节串数据。程序段如下： CLD　；DF=0，地址增量 MOV CX，10　；串长度 10 MOV DI，OFFSET DEST ；目的串首址→DI MOV DX，2300H ；16 位端口地址→DX REP INSB ；输入 10 个字节串数据
串输出指令	OUTS DX，mem OUTSB OUTSW OUTSD	从 DS:（E）SI 所指的源串中读出串数据输出至 DX 寄存器指定端口，并根据 DF 值和源串类型选样自动增/减及修改（E）SI 的内容。后三条指令同样分别为字节串、字串、双字串输出指令。 指令执行对标志位无影响	将内存 SOUE 开始的 10 个字节串数据从 2200 H 端口输出。程序段如下： CLD　；DF=0，地址增量 MOV CX，10　；串长度 10 MOV SI，OFFSET SOUE ；源串首址→SI MOV DX，2200H ；16 位端口地址→DX REP OUTSB ；输出 10 个字节串数据

续表

指令类型	汇编格式	指令的操作	举 例
位测试指令	BT reg16/reg32/mem，reg16/reg32/imm8	将 reg16/reg32/mem 中的某一位复制到进行标志 CF 中，该位的序号由 reg16/reg32/imm8 指定。 注意：reg16/reg32/imm8 的值可以是 0～15 或 0～31	例如： BT W-DATA，CX ；将字变量 W-DATA 中由 CX 指定的位送 CF
位测试和置位指令	BTS reg16/reg32/mem，reg16/reg32/imm8	将 reg16/reg32/mem 中的某一位复制 CF 中，然后 reg16/reg32/mem 中该位置 1，该位序号由 reg16/reg32/imm8 指定。 注意：同上	
位测试和复位指令	BTR reg16/reg32/mem，reg16/reg32/imm8	将 reg16/reg32/mem 中的某一位复制 CF 中，然后 reg16/reg32/mem 中该位复位（置 0），该位序号由 reg16/reg32/imm8 指定。 注意：同上	例如： BTR DX，BX ；将 DX 中的由 BX 指定的位送 CF，并将此位置 0
位测试和求反指令	BTC reg16/reg32/mem，reg16/reg32/imm8	将 reg16/reg32/mem 中的某一位复制 CF 中，然后 reg16/reg32/mem 中该位取反，该位序号由 reg16/reg32/imm8 指定。 注意：同上	例如： BTC EBX，2 ；将 EBX 中的第 2 位送 CF，并将此位状态取反
向前位扫描指令	BSF reg，reg/mem；	对源操作数 reg/mem 进行扫描，从其最低位开始向高位扫描测试，当遇到第 1 个置“1”的位，则置 ZF=0，并将该位的序号送到 reg 中。如果 reg/mem 中所有位均为 0，则置 ZF=1，且 reg 中的值没有意义。 注意：指令对其他标志位无意义	例如： 若 EBX＝3B014F0H，执行如下指令： BSF EAX，EBX ；结果 EAX=4（D_4 位）
向后位扫描指令	BSR reg，reg/mem；	从 reg/mem 中的最高位开始向低位扫描测试，当遇到第 1 个置“1”的位，则置 ZF=0，并将该位的序号送到 reg 中。如果 reg/mem 中所有位均为 0，则置 ZF=1，且 reg 中的值没有意义。 注意：同上	例如： 若 EBX＝3B014F0H，执行如下指令： BSR EAX，EBX ；结果 EAX=29（D_{29} 位）

3.3.5 控制转移类指令

计算机程序一般是顺序逐条执行的，如果要改变程序的顺序执行，就必须改变（E）IP 的内容或同时改变代码段段基址 CS 和（E）IP 内容，才能使程序改变执行顺序。在 80X86 指令系统中，控制转移类指令分为无条件转移指令、条件转移指令、循环控制类指令、过程调用和返回指令、中断指令五类。

1. 无条件转移指令 JMP

JMP 指令功能是无条件地使程序转移到指定的目标地址，并从此地址开始执行新的程

序段。根据与转移内指令有关的寻址方式，无条件转移指令分成下列 4 种：

（1）段内直接转移。

指令格式：JMP LABEL

这里，LABEL 是一个标号，也称为符号地址，它表示转移的目的地。该标号在本程序所在代码段内。指令被汇编时，汇编程序会计算出指令的下一条指令到所指示的目标地址之间的位移量（即相距多少个字节单元），该地址位移量可正可负，可以是 8 位或 16 位。若是 8 位，表示转移范围为－128 ～ ＋127 字节（又称为段内直接短转移，可写为 JMP SHORT OP1）；若位移量是 16 位，表示转移范围为－32768 ～ ＋32767 字节（又称为段内直接近转移，可写为 JMP NEAR PTR OP1）。在 80386/80486 中，位移量可以是 32 位有符号数。默认时为段内转移。

指令的操作是将（E）IP 的当前值加上计算出的地址位移量，形成新的（E）IP，并使 CS 保持不变，从而使程序按新地址继续运行。

[**例 3.19**]　下列程序段是一个无条件转移例子：

```
        ......
        MOV AX,BX
        JMP NEXT        ;无条件段内转移,转向符号地址 NEXT 处
        AND CL,0FH
        ......
NEXT:   OR CL,7FH
        HLT
```

例中，NEXT 是一个段内标号，汇编程序计算出 JMP 的下一条指令（即 AND CL，0FH）的地址到 NEXT 标号代表的地址之间的距离（即相对位移量）。执行 JMP 指令时，将这个位移量加到当前 IP 上，而转去执行 OR CL，7FH 指令。

（2）段间直接转移。指令中直接提供了要转移的 16 位段基址 CS 值和 16（32）位偏移地址（E）IP 值。

指令格式：JMP FAR PTR LABEL

指令中 FAR 表明其后的标号 LABEL 是一个远标号，即它在另一个代码段内。汇编程序根据 LABEL 的位置确定出 LABEL 所在段基地址和偏移地址，分别放在指令操作码后面。当执行指令时，将指令操作码后面的前两（四）个字节装入（E）IP 寄存器，后两个字节装入 CS 寄存器，从而实现段间转移。例如：

```
JMP FAR PTR NEXT        ;转移到另一段的 NEXT 初开始执行
JMP 6000H:1200H         ; 1200H→IP, 6000H→CS
```

（3）段内间接转移。

指令格式：

```
JMP reg             ;16 位或 32 位 reg 的内容→(E)IP,即 reg 为目标地址中的偏移地址,CS 不变
JMP WORD PTR mem    ;mem 为内存寻址方式,求出有效地址 EA,将(EA)→IP
```

指令执行时，将 16 或 32 位寄存器内容或各种内存寻址方式得到的有效地址 EA 中的内容写入 IP 或 EIP 中，CS 不变，实现段内转移。例如：

```
JMP DX              ;DX→IP,实现转移
JMP WORD PTR[BX]    ;(DS * 16 + BX + 1), (DS * 16 + BX)→IP,实现转移。
```

[例 3.20] 已知在数据段 DATA1 及 DATA2 地址中存 16 位偏移地址。编程实现段内转移。

解

```
地址          程序                    注释
              数据段(DS)
              ......
    DATA1     DW 2000H                ;存放了 16 位的偏移地址
    DATA2     DW 4200H                ;另一个 16 位偏移地址
              ......
              代码段(CS)
              ......
CS:1000H  MOV BX,OFFSET DATA1         ;DATA1 的偏移地址→BX
              JMP WORD PTR[BX]        ;(DS * 16 + BX + 1), (DS * 16 + BX)→IP,即 2000H→IP,程序转
              ......                  ;移至 CS：2000H 处继续执行
              MOV SI,DATA2            ;将 DATA2 符号地址中的内容 4200H→SI
              JMP SI                  ;寄存器间接寻址。SI→IP,程序转 CS：4200H 继续执行
              ......
CS:2000H      MOV AX,0000H
              ......
CS:4200H      MOV CX,100
              ......
              HLT
```

(4) 段间间接转移。

指令格式：

```
JMP DWORD PTR mem   ;mem 是与存储器有关的寻址方式,求其 EA, IP←(EA),CS←(EA + 2)
JMP FWORD PTR mem   ; mem 是与存储器有关的寻址方式,求其 EA, EIP←(EA)、CS←(EA + 4)
```

目标地址与 JMP 指令不在同一代码段的转移称为段间转移。段间间接转移用与存储器有关的各种寻址方式来指定目标地址所在存储单元。指令执行时将指定的连续四（六）个存储单元的内容送（E）IP 和 CS，从而程序转到另一代码段继续执行。

[例 3.21] JMP DWORD PTR [BX]

设指令执行前：DS=3000H，BX=3000H，[33000H] =0BH，[33001H] =20H，
[33002H] =10H，[33003H] =80H；

则指令执行后，IP=200BH，CS=8010H。

转移的目的地址=8210BH。段间间接转移指令执行过程如图 3.22 所示。

段间转移不仅 IP 的内容要改变，CS 内容也要改变，即转移地址一定是 32 位字长。因此，在操作数前要加上 DWORD PTR，表示其后的操作数为双字。

无条件转移指令的目标地址常用属性操作符表达式，举例归述如下：

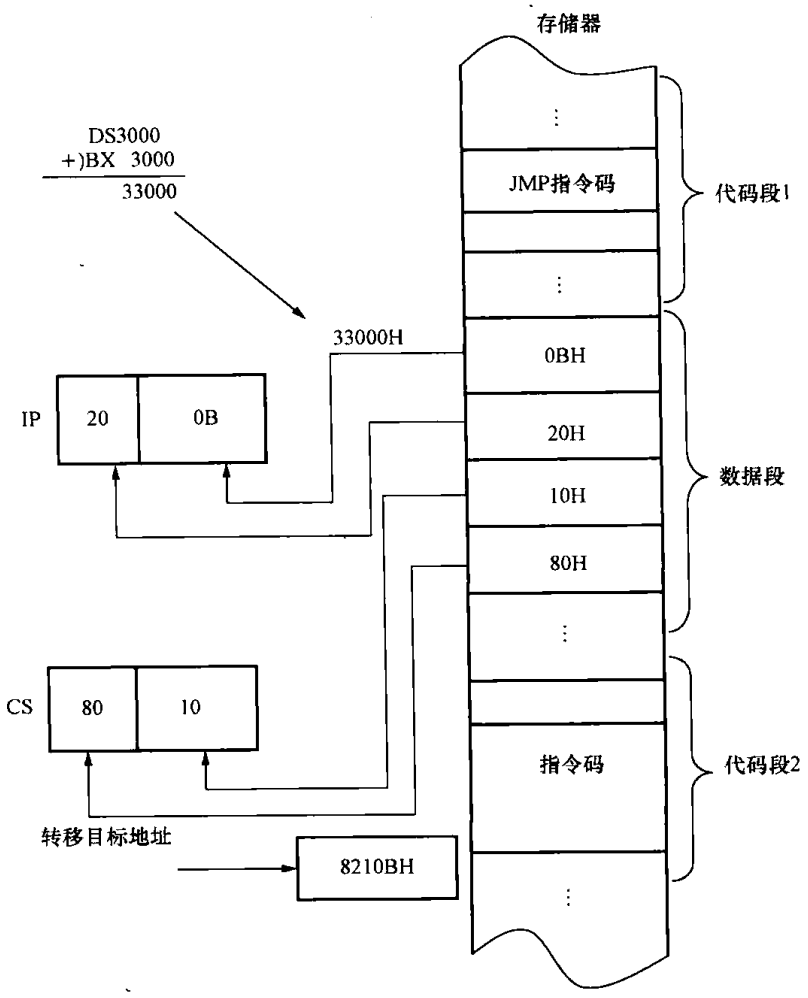


图 3.22　段间间接转移指令执行过程

```
JMP SHORT PTR 标号        ;短距离段内转移
JMP NEAR PTR 标号         ;段内直接近转移
JMP FAR PTR 标号          ;段间直接转移
JMP WORD PTR 变量         ;段内间接转移
JMP DWORD PTR 变量        ;段间 16 位寻址间接转移
JMP FWORD PTR 变量        ;段间 32 位寻址间接转移
```

以上所有无条件转移指令，其执行结果均对标志位无影响。

2. 条件转移指令 JCC

条件转移指令格式：JCC OP1

指令执行时根据前一条指令对标志寄存器各标志位设定的状态来决定程序是否转移，即对预先指定的条件标志进行测试，如果条件满足，则程序转移到指令指定的目标地址处开始执行；若条件不满足，则顺序执行下一条指令。条件转移指令不影响标志位。表 3.6 列出了各种条件转移指令的转移条件及其功能。

在 8086/8088 中，所有条件转移指令都为短转移（转移距离为－128～＋127）。80386/80486 则将其扩展到段内转移。16 位寻址方式时，即转移的距离为 64K；32 位寻址方式时可为 4G。

表 3.6　　条件转移指令的转移条件及其功能

无符号数 A、B 的比较（A－B）

助记符	指令功能描述	条　件	备　注
JA/JNBE OP1	高于/不低于且不等于，转移	CF=0 且 ZF=0	A>B
JAE/JNBE OP1	高于或等于/不低于，转移	CF=0	A≥B
JB/JNAE OP1	低于/不高且不等于，转移	CF=1	A<B
JBE/JNA OP1	低于或等于/不高于，转移	CF=1 或 ZF=1	A≤B
JE/JZ OP1	等于/为零，转移	ZF=1	A=B
JNE/JNZ OP1	不等于/不为零，转移	ZF=0	A≠B

带符号数 A、B 的比较（A－B）

助记符	指令功能描述	条　件	备　注
JE/JZ OP1	等于/为零，转移	ZF=1	A=B
JG/JNLE OP1	大于/不小于且不等于，转移	ZF=0 且 SF=OF	A>B
JGE/JNL OP1	大于或等于/不小于，转移	SF=OF	A≥B
JL/JNGE OP1	小于/不大于且不等于，转移	SF≠OF	A<B
JLE/JNG OP1	不小于或等于/不大于，转移	ZF=1 或 SF≠OF	A≤B
JNE/JNZ OP1	不等于/不为零，转移	ZF=0	A≠B

其　他　条　件

助记符	指令功能描述	条　件	备　注
JC	如果有进位，转移	CF=1	
JNC	如果无进位，转移	CF=0	
JO	如果溢出，转移	OF=1	
JNO	如果不溢出，转移	OF=0	
JS	如果是负数，转移	SF=1	
JNS	如果非负数，转移	SF=0	
JPE	如果 1 的个数为偶，转移	PF=1	
JPO	如果 1 的个数为奇，转移	PF=0	

在使用条件转移指令时，其前一条指令应是执行后能够对相应状态标志位产生影响的指令，有的根据一个标志位、两个标志位组合或两个以上标志位组合来判断是否实现转移。例如，要判断两个无符号数的大小，应当使用 CMP 指令，然后根据 CF 的状态，在其后使用 JNC（或 JC）指令决定程序转移到何处执行；或者根据 CF、ZF 的状态，在其后使用 JA（或 JNBE）指令决定程序转移到何处执行。

[例 3.22] 有 10 个无符号字节数据顺序存放在以 2000H 单元为首地址的数据存储区中，编写程序找出其中最大数，并将其存入 2200H 单元。

解　程序段如下：

```
NAXSTA:MOV BX,2000H     ;数据区首地址 2000H→BX
       MOV AL,[BX]      ;取第一个数据
       MOV CX,10        ;数据块长度→CX
NEXT1: INC BX           ;BX 加 1,指向下一个数据
       CMP AL,[BX]      ;和下一个数相比较
       JAE NEXT2        ;如果 AL 中的数比 BX 指针所取的数大或相等则转 NEXT2
       MOV AL,[BX]      ;否则将大数存至 AL 中
NEXT2: DEC CX           ;CX 计数值减 1,若不为零转 NEXT1,取下一个数比较
       JNZ NEXT1
       MOV BX,2200H     ;否则 10 个数都比较完毕,将最大数→2200H 单元
       MOV [BX],AL
       HLT
```

[例 3.23]　在内存首地址为 TABLE 处顺序存放了 100 个带符号字节数。编写程序统计出其中正数、负数和零的个数，分别将个数存入 PLUS、MINUS 和 ZERO 单元。

编程思路：先将 PLUS、MINUS、ZERO 三个单元清零，设计数器 CX，数据区地址指针 SI，通过地址指针 SI 逐个取出带符号数放入 AL，利用条件转移指令测试 AL 是正数、负数还是零，分别在对应单元中加 1，构成一个循环直到 100 个数都处理完。

解　程序段如下：

```
START:  XOR AL,AL          ;AL 清零
        MOV PLUS,AL        ;PLUS 单元清零
        MOV MINUS,AL       ;MINUS 单元清零
        MOV ZERO,AL        ;ZERO 单元清零
        LEA SI,TABLE       ;数据区首地址送 SI
        MOV CX,100         ;数据区数据个数送 CX
        CLD                ;使 DF = 0
CHECK:  LODSB              ;取一个数到 AL,且 SI + 1→SI
        OR AL,AL           ;操作数自身相或,AL 值不变,根据结果影响标志位
        JS NEXT1           ;若为负,转 NEXT1
        JZ NEXT2           ;若为 0,转 NEXT2
        INC PLUS           ;否则为正,PLUS 单元加 1
        JMP NEXT           ;
NEXT1:  INC MINUS          ; MINUS 单元加 1
        JMP NEXT           ;
NEXT2:  INC ZERO           ;ZERO 单元加 1
NEXT:   DEC CX             ;CX 减 1 再送 CX
        JNZ CHECK          ;若 CX 不为 0,则转 CHECK
        HLT
```

3. 循环控制指令

循环控制指令与一般条件转移指令相同，也是根据是否满足给定条件来决定程序执行顺序：当条件满足时，转目标地址处执行；否则（不满足），顺序执行程序。循环控制指令用

存放在（E）CX 中的数作为循环重复计数值，每执行一次，计数值减 1，直到减至 0 时终止循环，或把（E）CX 内容与 ZF 标志位相结合作为转移条件。循环控制指令都采用相对寻址的短转移，即以当前 IP 值为中心的－128～＋127 范围内。指令执行结果对标志位没有影响。

循环控制指令有四条，其指令格式及跳转条件如表 3.7 所示。

表 3.7　　循环控制指令格式及跳转条件

指 令 格 式	条 件	指令功能说明
LOOP OP1	（E）CX≠0	（E）CX－1→（E）CX；满足条件则转 OP1 处执行，否则顺序执行
LOOPE/LOOPZ OP1	（E）CX≠0 且 ZF=1	（E）CX－1→（E）CX；满足条件则转 OP1 处执行，否则顺序执行
LOOPNZ/LOOPNE OP1	（E）CX≠0 且 ZF=0	（E）CX－1→（E）CX；满足条件则转 OP1 处执行，否则顺序执行
JCXZ OP1	（E）CX=0	（E）CX－1→（E）CX；满足条件就转 OP1 处执行，否则顺序执行

其中，LOOP 和 JCXZ 指令仅是根据（E）CX 是否为零来作为转移条件。其功能分别如同如下指令组：

```
DEC (E)CX                          DEC (E)CX
JNZ OP1   ；等价于 LOOP OP1        JZ OP1   ；等价于 JCXZ OP1
```

[例 3.24] 在以 DATA 为首地址的内存数据段中，存放 200 个 16 位带符号数，找出其中最大和最小数，分别放在以 MAX 和 MIN 为首地址的内存单元中。

编程思路：先取数据区中的一个数据作为标准，将其同时暂存于 MAX 和 MIN 中，然后使其他数据分别与 MAX 和 MIN 中的数进行比较，若大于则取代原 MAX 中的数，若小于 MIN 内容，则取代原 MIN 中的数，最后就得到了数据块中最大和最小的带符号数。注意：比较带符号数大小时，应采用 JG 和 JL 等有符号数条件转移指令。

解　程序段如下：

```
START:  LEA SI,DATA      ;数据块首址偏移地址→SI
        MOV CX,200       ;数据块长度→CX
        CLD              ;清方向标志 DF,按增量
        LODSW            ;取数据块第一个 16 位符号数→AX
        MOV MAX,AX       ;该数送→MAX
        MOV MIN,AX       ;该数送→MIN
        DEC CX           ;CX 内容减 1→CX
NEXT:   LODSW            ;取下一个 16 位带符号数→AX
        CMP AX,MAX       ;与 MAX 单元内容进行比较
        JG LARGER        ;若大于则转 LARGER
```

```
        CMP AX,MIN      ;与 MIN 单元内容进行比较
        JL SMALL        ;若小于则转 SMALL
        JMP GOON        ;否则转 GOON
LARGER: MOV MAX,AX      ; AX→MAX
        JMP GOON        ;
SMALL:  MOV MIN,AX      ; AX→MIN
GOON:   LOOP NEXT       ;CX 减 1→CX,若 CX≠0 则转 NEXT,否则顺序执行
        HLT
```

[例 3.25]　有两个字数组分别放在首地址为 DATA1 和 DATA2 的存储区，数组长度为 100。计算两数组对应数据项之和，和值送首地址为 SUM 的数据区中，且当两数组对应项均出现“0”时，停止求和。试编程序段。

解

```
        MOV AX,0            ;清零
        MOV SI,0FFFFH       ;设指针初值为 - 1
        MOV CX,64H          ;数据块长度→CX
NZERO:  INC SI              ;
        MOV AX,DATA1[SI]    ;取被加数
        ADD AX,DATA2[SI]    ;相加
        MOV SUM[SI],AX      ;和存入 SUM 数据区
        LOOPNZ NZERO        ;CX-1→CX,CX≠0 且 ZF = 0 循环转 NZERO 处执行,否则顺序执行
        HLT                 ;暂停
```

4. 子程序（过程）调用和返回指令

为便于模块化程序设计和节省内存单元，把经常重复出现的一段程序独立出来，成为一个子程序（过程）。其他程序可用子程序调用指令 CALL 来调用该子程序。CALL 指令能迫使 CPU 暂停执行下一条指令（断点），转去执行指定的子程序。待子程序执行完成返回指令 RET 后，再返回到断点继续执行程序，如图 3.23 所示。

通常把同一代码段内（CS 不变，IP 变）调用的过程称为近过程，用 NEAR 表示；而把调用其他代码段（段间调用，CS、IP 同时改变）的过程称为远过程，用 FAR 表示。

CALL 指令与 JMP 指令一样，也有段内直接调用、段内间接调用、段间直接调用、段间间接调用四种形式。下面分别进行介绍。

（1）段内直接调用。

指令格式：CALL NEAR PROC

指令中 PROC 是一个近过程名或标号，实际上是一个近过程的符号地址。段内直接调用表示 CALL 指令和被调过程 PROC 在同一代码段内。指令汇编后，CALL 指令的操作数部分给出 CALL 指令的下一条指令与被调用过程的入口地址之间相差的 16 位相对位移量。执行时先将 CALL 指令的下一条指令的偏移地址（E）IP（断点地址）压入堆栈，然后将指令中 16 位的相对位移量和当前（E）IP 的内容相

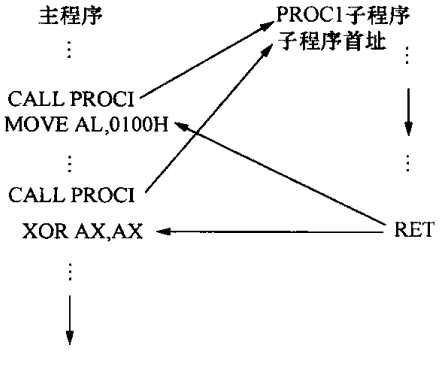


图 3.23　调用子程序的过程

加送（E）IP，即为所调用过程的入口地址。从而实现了程序转移。

指令执行过程如下：

1）修改指针 SP←SP － 2（或 SP←SP －4）。

2）断点地址压入堆栈，IP→SP ＋ 1、SP 所指的字单元或 EIP→SP＋4 ～ SP 所指的双字单元。

3）目标地址（E）IP＋16 位偏移量→（E）IP。

对于段内调用，指令中的 NEAR 可以省略。例如，“CALL SUB1”指令表示要调用一个名为 SUB1 的子程序。

（2）段内间接调用。

指令格式：CALL reg/mem

指令执行时，先将 CALL 指令的下一条指令的偏移地址 IP 压入堆栈，当操作数是一个 16 位通用寄存器时，寄存器内容即为入口地址送 IP；当是存储器寻址时，操作数给出的是有效地址 EA，从 EA 中取出内容送 IP。例如：

CALL BX ；子程序入口地址是 BX 寄存器内容，即 IP＝BX

CALL WORD PTR [BX] ；（BX）＝EA。从 EA 或 EA 和 EA＋1 中取出内容→（E）IP 低 8/16 位，EA＋1 或 EA＋2 和 EA＋3 中取出内容→（E）IP 高 8/16 位。

（3）段间直接调用。

有两种指令格式：

```
CALL PROC              ;PROC 是已定义的过程名,在另外一个代码段中
CALL FAR PTR PROC      ;PROC 是标号或过程名,用 FAR PTR 操作说明符说明是段间调用
```

指令执行时使程序转到另一代码段的过程去执行程序。以偏移地址为 16 位为例，执行过程如下：

1）SP←SP－2；[SP＋1]、[SP] ←CS ；修改 SP 指针，断点段基址压入堆栈

2）SP←SP－2；[SP＋1]、[SP] ←IP ；修改 SP 指针，断点偏移地址压入堆栈

3）从指令机器码的第 2，3 字节取出 16 位偏移地址→IP。

4）从指令机器码的第 4，5 字节取出 16 位段基地址→CS。

例如：CALL PROCB

PROCB 为一个已定义的远过程名。该指令经汇编程序汇编后在内存存放，其操作示意图如图 3.24 所示。执行此指令时，先将断点地址 CS＝2000H、IP＝6200H＋05H＝6205H 压入堆栈；再将指令给出的偏移地址 3000H 送至 IP，段基址 4000H 送入 CS，程序转到物理地址为 43000H 处执行。

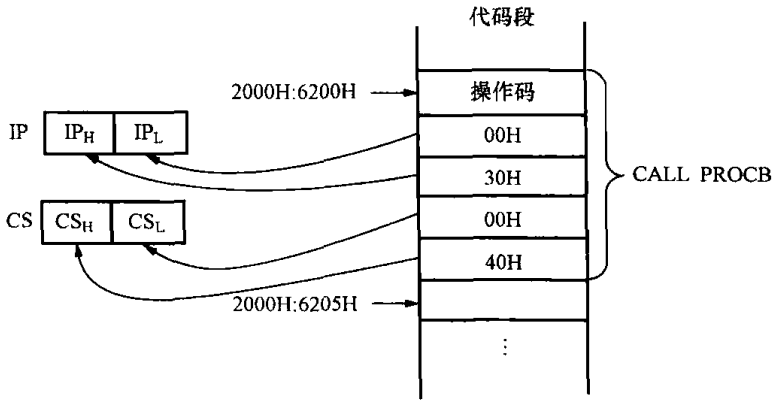


图 3.24　CALL PROCB 操作示意图

（4）段间间接调用。

指令格式：CALL mem

其中，mem 是由存储器寻址

方式确定的有效地址。指令执行时先将 CALL 的下一条指令地址（即断点）CS 和 IP 压入堆栈，然后由存储器寻址方式求得有效地址 EA 指定的连续 4 个存储单元中的内容送 IP 和 CS，其中前两个单元送 IP，后两个单元内容送 CS。例如：

CALL DWORD PRT [BX]

以上所有的 CALL 指令执行结果不影响标志位。

（5）返回指令。过程中至少要有一条返回指令 RET，它不带操作数。功能是使子程序执行完后返回主程序断点处继续往下执行。

如果程序定义的是一个 NEAR（段内）过程，则对应 RET 指令就执行段内过程返回主程序的操作（从堆栈弹出断点偏移地址送 IP）；如果程序定义的是一个 FAR（段间）过程，则 RET 指令执行段间返回主程序操作；从栈顶弹出第一个字（断点偏移地址）送 IP，弹出第二个字（断点段基地址）送 CS。汇编程序会根据过程的 NEAR、FAR 属性将 RET 指令翻译成相应的机器目标代码存入源程序中。

5. 中断和中断返回指令

所谓中断是指在程序运行期间因某种随机或异常事件，请求 CPU 暂时中止正在运行的程序，转去执行处理这些事件的程序（称中断服务程序），处理完毕后又返回到被中止的原程序处继续执行的过程。

引起中断的事件叫中断源，它可以是在 CPU 内部，也可以是在 CPU 外部。内部中断源引起的中断称为内部中断，相应地，外部中断源引起的中断称为外部中断。8086/8088 中断系统分为外部中断（或叫硬件中断）和内部中断（或叫软件中断）。

关于中断的详细论述详见本书第 6 章，这里仅介绍 8086/8088 指令系统提供的 3 条与软件中断相关的指令格式及操作。

（1）INT 指令。

指令格式：

INT　n　；n 为中断向量码（也称中断类型码），是一个常数，取值范围为 0～255

指令执行时，CPU 根据 n 的值计算出中断向量的地址，然后从该地址中取出中断服务程序的入口地址并转到该中断服务程序去执行。中断向量地址的计算方法是将中断向量码 n 乘 4。INT 指令的具体操作步骤如下：

```
1) SP←SP－2;[SP＋1]、[SP]←FLAGS ;修改 SP 指针,标志寄存器内容压入堆栈
2) TF← 0;IF← 0 ;保证不会中断正在执行的中断服务子程序且不响应单步中断
3)SP←SP－2;[SP＋1]、[SP]← CS
  SP←SP－2;[SP＋1]、[SP]← IP ;断点地址(INT 指令的下一条指令地址)入堆栈
4)IP←[n＊4＋1]、[n＊4];CS←[n＊4＋3]、[n＊4＋2]
  ;由 n＊4 得到中断向量地址,并进而得到中断服务程序入口地址
```

以上操作完成后，CS：IP 就指向中断服务程序的第一条指令，此后 CPU 开始执行中断服务程序。

INT　n 指令除对 IF 和 TF 位置 0 外，对其他标志无影响。

（2）中断返回指令。中断返回指令 IRET 用于从中断服务程序返回到被中断的程序继续执行。任何中断服务程序无论是由外部中断引起的还是由内部中断引起的，其最后一条指令都是 IRET 指令。此指令首先将堆栈中的断点地址弹出到 IP 和 CS 中，接着将 INT 指令执

行时压入堆栈的标志字弹出到标志寄存器，恢复到中断前的标志状态，使程序返回到原来发生中断时所指向的指令。本指令对各标志位有影响。指令具体操作为：

1) IP←[SP+1]、[SP]；SP←SP+2

2) CS←[SP+1]、[SP]；SP← SP+2

3)FLAGS←[SP+1]、[SP]；SP← SP+2

3.3.6 处理器控制指令

处理器控制指令包括标志操作指令、操作系统型指令、外同步类指令、空操作指令四类。操作系统型指令将作为附录4介绍。其他指令格式及功能如表3.8所示，表中的OP可以是8位寄存器或存储单元。

表3.8　　标志位、外同步、空操作类指令及功能

类别	指令格式	功能	说明
单个标志位操作指令	CLC	清进位标志（CF←0）	
	STC	置进位标志（CF←1）	
	CMC	进位标志取反（CF←CF）	
	CLD	清方向标志（DF←0）	使串操作指令的地址指针为增量
	STD	置方向标志（DF←0）	使串操作指令的地址指针为减量
	CLI	清中断允许标志（IF←0）	IF=0表示禁止可屏蔽中断（关中断）
	STI	置中断允许标志（IF←1）	IF=1表示允许可屏蔽中断（开中断）
条件设置字节指令	SETC/SETB/SETNAE OP	CF=1置OP为01H，否则为00H	有进位/低于/不高于且不等于
	SETNC/SETAE/SETNB OP	CF=0置OP为01H，否则为00H	无进位/高于或等于/不低于
	SETO OP	OF=1置OP为01H，否则为00H	溢出
	SETNO OP	OF=0置OP为01H，否则为00H	无溢出
	SETP/SETPE OP	PF=1置OP为01H，否则为00H	校验为偶
	SETNP/SETPO OP	PF=0置OP为01H，否则为00H	检验为奇
	SETS OP	SF=1置OP为01H，否则为00H	为负数
	SETNS OP	SF=0置OP为01H，否则为00H	为正数
	SETA/SETNBE OP	CF=ZF=0置OP为01H，否则为00H	高于/不低于且不等于
	SETBE/SETNA OP	CF=1或ZF=1置OP为01H，否则为00H	低于或等于/不高于
	SETE/SETZ OP	ZF=1置OP为01H，否则为00H	等于/为零
	SETNE/SETNZ OP	ZF=0置OP为01H，否则为00H	不等于/非零
	SETG/SETNLE OP	ZF=0且SF=OF置OP为01H，否则为00H	大于/不小于且不等于
	SETGE/SEGNL OP	SF=OF置OP为01H，否则为00H	大于或等于/不小于
	SETL/SETNGE OP	SF≠OF置OP为01H，否则为00H	小于/不大于且不等于
	SETLE/SETNG OP	ZF=1或SF≠OF置OP为01H，否则为00H	小于或等于/不大于

续表

类别	指令格式	功能	说明
外同步和空操作类指令	HLT	程序停止执行，处理器进入暂停状态	不影响任何标志位。只有复位或外部中断能重新执行程序。常在程序中用于等待中断
	ESC	使协处理器从 CPU 的程序中取一条指令或一个存储器操作数，以完成主处理器对协处理器的某种操作要求	不影响任何标志位。协处理器完成任务后，经 TEST 状态线向 CPU 送一个低电平信号，CPU 检测到此信号后，继续执行后续指令
	WAIT	使处理器处于等待状态	不影响任何标志位。出现外部中断能重新执行程序
	LOCK	锁住指令中目的存储器操作数，使该指令执行期间一直被保护，其他主控器不能访问	共享存储器的多处理机中使用。加在目的操作数为存储器的指令前。80386/80486 的 BT/BTS/BTR/BTC、XCHG、ADD/ADC、SUB/SBB、AND、OR、XOR、NOT、NEG、INC、DEC 等指令前可加前缀 LOCK
	NOP	CPU 保持原状态不做任何工作	一是延时，CPU 每执行一条 NOP 指令，80386 需要 3 个时钟周期，80486 需要 1 个时钟周期，重复多次 NOP 指令实现延时；二是程序中多加几条 NOP 指令，需要时将 NOP 指令改成其他指令，在调试程序时常用到

3.4　汇编语言的编程格式

前面我们学习了微处理器的组成原理及 80X86 指令系统，具备了用汇编语言进行程序设计的能力。汇编语言是以处理器指令系统为基础的低级程序设计语言，采用助记符表达指令操作码，标识符表示指令操作数。汇编语言编写的程序主要优点是直接控制计算机硬件，完成同样功能比高级语言代码序列短、运行速度快。当然，汇编语言也存在不足，如功能有限、编程繁难、依赖处理器指令等。

汇编语言用英文字母缩写表示的助记符来表示指令操作码和操作数，也可以用标号和符号来代替地址、常量和变量。由于不同 CPU 指令系统的指令编码不同，因此与之相应的汇编语言也不尽相同。用汇编语言编写的程序不能由机器直接执行。必须通过具有“翻译”功能的系统程序——汇编程序（Assembler）将这种符号化的汇编语言转换成相应的机器代码，再通过连接程序得到可执行文件，如图 3.25 所示。用汇编语言编写的程序叫源程序，源程序经汇编程序翻译后所得的机器指令代码称为机器语言目标程序，简称目标程序。通常称这样的翻译过程为汇编过程，简称汇编。为保证这种“翻译”正确，编写的源程序必须符

合汇编程序规定的一系列汇编语句格式。

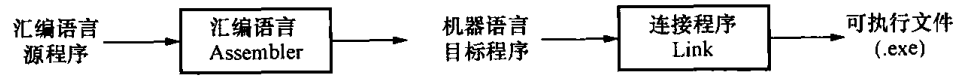


图 3.25 汇编、连接过程

汇编语言的基本语句与机器指令是一一对应的。另外，汇编程序还引入了新的汇编指令——伪指令和宏指令，使得采用汇编语言进行程序设计更方便灵活。至今在计算机系统程序、在线实时控制程序以及图像处理等方面仍广泛使用。

本节以 Microsoft 公司的宏汇编程序 MASM 为背景，介绍面向 80X86 的汇编语言基本概念、语法规则、基本程序设计方法。

3.4.1 汇编语言程序结构

汇编语言与高级语言程序一样也有一系列语法规则。一个汇编语言源程序中有三种基本语句：指令语句、伪指令语句和宏指令语句。

指令语句是 CPU 可执行的语句，用助记符形式书写与机器指令一一对应。伪指令语句是说明性语句，又称指示性语句，是程序员发给汇编程序的命令，没有相应的机器指令。宏指令语句是用户在源程序中定义的一段具有独立功能的指令语句。

80X86 按照逻辑段组织程序，具有代码段、数据段、堆栈段和附加段。因此，完善的汇编语言源程序也由段组成。一个汇编语言源程序可包含若干个代码段、数据段、堆栈段或附加段，段与段之间的顺序可随意排列。能独立运行的程序至少包含一个代码段，所有指令语句必须位于某一个代码段内，伪指令语句和宏指令语句可根据需要位于任一个段内。按段组织的源程序在汇编后将指令码和数据分别装入存储器的相应物理段中。

下面是一个完整的 MASM-86 汇编语言规范的源程序。程序完成将两个 16 位的字数据相加，和值放在 SUM 内存单元。

```
DSEG    SEGMENT                  ;定义数据段,段名为 DSEG
DATA1   DW  0F865H               ;定义被加数 F865H
DATA2   DW  360CH                ;定义加数 360CH
DSEG    ENDS                     ;数据段结束
;
ESEG    SEGMENT                  ;定义附加数据段,段名为 ESEG
SUM     DW  2 DUP(?)             ;定义存放和值的内存区域
ESEG    ENDS                     ;附加段结束
;
CSEG    SEGMENT                  ;定义代码段,段名为 CSEG
ASSUME  CS:CSEG,DS:DSEG, ES:ESEG ;说明程序中定义的各段分别用哪个段寄存器寻址
START:  MOV  AX,DSEG             ;程序可执行代码起始点
        MOV  DS,AX               ;将数据段寄存器 DS 指向用户定义数据段
        MOV  AX, ESEG
        MOV  ES, AX              ;将附加段寄存器 ES 指向用户定义数据段
        LEA  SI, SUM             ;存放和值的偏移地址送 SI
        MOV  AX, DATA1           ;取被加数送 AX
```

```
        ADD   AX, DATA2              ;两数求和,和值放 AX 中
        MOV   ES:[SI], AX            ;和值送附加数据段的 SUM 单元
        MOV   AH,4CH                 ;系统功调用,程序结束返回操作系统
        INT   21H
CSEG ENDS                            ;代码段结束
        END   START                  ;源程序结束,程序起始点为 START
```

完整的汇编语言源程序格式按段组织。段定义由 SEGMENT 和 ENDS 伪指令实现，同时需要伪指令 ASSUME 说明该逻辑段的类型。本例中 ASSUME 语句将 CS、DS、ES 依次指向名为 CSEG、DSEG 和 ESEG 的逻辑段，即依次设置为代码段、数据段和堆栈段。程序开始先用传送指令将 DS、ES 分别赋值为数据段 DSEG 和附加数据段 ESEG 的段基地址，便于后续指令访问到 DSEG、ESEG 中的数据，但 CS 不用传送指令赋值，当用 ASSUME 时已自动赋值了。最后，利用 4CH 号系统功能调用结束执行，并返回操作系统。

了解了汇编源程序结构后，我们来看看汇编语言源程序中的语句格式，包括指令语句、指令性语句和宏指令语句。

3.4.2　汇编语言语句

指令语句一般格式：

[标号:]　[前缀指令]　指令助记符　[操作数 [，操作数]]　[；注释]

指示性（伪指令）语句一般格式：

[名字]　伪操作符　[操作数 [，操作数，……]]　[；注释]

可见有四项组成，其中加方括号项按需要可有可无。有此项书写时不加方括号，项与项之间用空格分隔。

(1) 标号和名字：标号表示指令的符号地址，即对应机器指令在存储器中的位置，须加以“:”作为结束符。通常用在转移类指令中作为转移目标地址。名字通常表示变量名、段名和过程名等，其后不加“:”。名字在多数情况下表示的是变量名，用来表示存储器中一个数据区的地址。

标号和名字的起名规则：必须以字母开头，后跟字母、数字（0，…，9）及特殊符号（?，@，_，$等）等，长度为1～31个字符，不能是属于系统的专用保留字。保留字主要有 CPU 中各寄存器名；指令助记符（如 MOV、ADD)；伪指令；表达式中的运算符和属性操作符（如 PTR、OFFSET、SEG）等。

(2) 前缀：加在某些指令之前，不能单独使用，如 LOCK、REP 前缀等。前缀与指令助记符之间应加空格，如 REP MOVSB。

(3) 指令助记符和伪操作符：是为指令操作码和伪指令规定的符号。任何汇编语句都需要此部分，它表示该汇编语句的基本功能。如 MOV 是数据传送指令助记符，ADD 是加法指令助记符，DB 是字节数据定义伪指令。

(4) 操作数：根据指令功能需要，可不带操作数或带一个、二个、若干个操作数。若有两个及以上操作数时，操作数间用逗号分开。操作数部分，可以是寄存器、常量或存储器操作数，甚至还可以是一个表达式。例如：

```
CLD                          ；不带操作数
START：MOV AX，DATA          ；带两个操作数
```

```
DATA1  DB  30H，31H，32H    ；带若干个操作数
```

（5）注释：为使汇编语言源程序更便于阅读和理解，常在源程序中加上注释，注释是以分号开始后跟字符序列直到行结尾。汇编时汇编程序对注释部分不予处理。

程序注释直接影响源程序的可读性。好的程序注释，使程序易于阅读、调试及维护。

3.4.3 汇编语句的操作数

汇编语句中的操作数（Operands）可以是寄存器、常量或存储器操作数，甚至可以是表达式。

1. 寄存器

80X86 的寄存器常作为指令的一个操作数，可以是 8 位或 16 位寄存器。例如，AL 是字节型寄存器，AX 是字型寄存器，而在 32 位机中，EAX 是双字型寄存器。

2. 常量

常量就是在程序运行期间不会改变的那些固定值，有数字常量和字符串常量。例如，立即数寻址中的立即数、直接寻址时所用的地址、ASCII 字符等都是常量。

（1）数字常量。就是数字（整数）常量，MASM 宏汇编支持多种进制形式的数值。

1）二进制常数：由一串“0”和“1”组成的序列，最后加写大写字母 B，如 00101100B。

2）十进制常数：由若干个 0～9 数字组成的序列，最后加写大写字母 D，通常省略不写，如 1234D 或 1234。

3）八进制常数：由若干个 0～7 数字组成的序列，最后加写大写字母 Q，如 255Q 等。

4）十六进制常数：由若干个 0～9 数字、A～F 字母所组成的序列，最后加写大写字母 H。为避免与标号、变量等标识符相混淆，十六进制数的最高位是字母 A～F 时，须在前面加写数字 0。例如 58H，0BA3FH 等。

（2）字符串常量。字符串常量由单引号括起来的一个或多个 ASCII 码字符构成。汇编程序将其中每个字符翻译成它对应的 ASCII 码值。例如，字符串“A”汇编时翻译成 41 H，字符串“AB”汇编时翻译成 41H、42H。

3. 存储器操作数

存储器操作数通常是一个标识符，有标号（Label）和变量（Variable）两种。

指令语句的标号代表存放该指令的存储单元的符号地址，其后须加冒号。通常是作为转移类指令、过程调用指令和循环控制类指令的操作数。

变量是存储器中某个数据区的名字，由于数据区中内容是可以改变的，因此，变量的值也可以改变。变量在指令中可以作为存储器操作数引用。变量的定义将在 3.4.4 节介绍。

标号和变量一经定义，都具有三种基本属性：段地址、偏移地址和类型属性。

（1）段地址属性：标号段地址属性是标号所在段的段基地址，当程序中引用一个标号时，该标号应在代码段中。变量段地址属性就是它所在段的段基地址，变量一般在存储器的数据段或附加段中。

（2）偏移地址属性：是标号或变量所在段的段首到定义该标号或变量的地址之间的字节数（即偏移地址），是一个 16 位无符号数。

（3）类型属性：标号的类型有 NEAR 和 FAR 两种。前一种称为近标号，只能在段内被引用。后一种称为远标号，可以在其他段被引用。变量类型可以是 BYTE（字节）、WORD（字）、DWORD（双字）、QWORD（四字）和 TBYTE（十个字节）等，表示数据区中存取

操作对象的大小。

4. 表达式

表达式是由各种运算符、操作数所构成的序列，汇编程序在汇编过程中计算表达式并得到一个确定的数值。组成表达式的各部分必须在汇编时就能有确定的值。表达式中常用运算符有以下几种。

(1) 算术运算符。常用算术运算符有加（+）、减（-）、乘（*）、除（/）和取模运算(MOD)，例如：

```
MOV  AL,8+6          ;AL = 0EH
MOV  AL,20H  MOD 7   ;AL = 4,取模运算(MOD)是"取余数"
MOV  AL,VAR+2        ;VAR+2 是一个地址表达式时,表示从 VAR+2 的存储单元内容送 AL
```

(2) 逻辑运算符。逻辑运算符包括与（AND）、或（OR）、非（NOT）和异或（XOR）。逻辑运算符只能用在数值表达式中，两数相应位进行逻辑运算，并得到数值结果。例如：

```
MOV  AL,11001100B AND 11110000B     ;执行后 AL = 11000000B
MOV  BL,NOT 0FFH                    ;执行后 BL = 00H
```

注意：

1) 逻辑运算符只能对常数进行运算。如："MOV　CX,　AX AND 1" 是非法的。

2) 逻辑运算符与逻辑运算指令助记符相同，但在指令中出现的位置不同。出现在指令操作数部分时，汇编程序把它看成是逻辑运算符，逻辑运算功能在汇编时完成。逻辑运算指令功能在程序执行阶段完成。例如：

AND　DX,　PORTA AND 0FEH

其中，PORTA 是伪指令定义的常量，第一个 AND 是逻辑操作指令助记符，第二个 AND 是逻辑运算符。

(3) 关系运算符。关系运算符有等于（EQ）、不等于（NE）、小于（LT）、大于(GT)、小于等于（LE）、大于等于（GE）。参与关系运算必须是两个数值或同一段中的两个存储单元地址，运算结果是一个逻辑值，当关系成立（为真）时，其结果为全 1，即 0FFFFH；关系不成立（为假）时，结果为全 0，即 0000H。例如：

```
MOV BX, 6 LT 5       ;关系不成立,汇编后等效于指令 MOV BX,0000H
MOV AX,6 NE 5        ;关系成立,汇编后等效于指令 MOV AX,0FFFFH
```

(4) 分析运算符和合成运算符。分析运算符可以把存储器操作数分解为它的组成部分，如它的段地址值、段内偏移量和类型。合成运算符可以由已经存在的存储器操作数生成一个段地址值与偏移量相同、而类型不同的新存储器操作数。详见 3.4.4 节。

(5) 其他运算符。

1) 方括号 []。指令中用方括号表示存储器操作数，方括号里的内容表示操作数的偏移地址。

2) 段超越运算符 ":"。跟在某个段寄存器名（DS、ES、SS）后表示重设段基地址，用来指定一个存储器操作数的段属性，而不管其原来隐含的段是什么。例如：

```
MOV  AX,ES:[DI]     ;把 ES 段中由 DI 指向的字操作数送 AX,而不是默认的 DS 段
```

3.4.4 伪指令

指示性语句中的伪操作命令，无论表示形式或其在语句中所处的位置，都与 CPU 指令相似，因此也称伪指令，但两者间是有区别的。

指令语句在程序运行时由 CPU 执行，每条指令对应 CPU 的一种特定操作，如加法、减法、数据传送等；伪指令在汇编过程中由汇编程序执行，提供汇编所需的信息，如程序的开始、结束、数据区的定义及原始数据、存储区的分配等。另外，汇编以后每条 CPU 指令都被汇编并产生一条对应的目标代码，而伪指令汇编后不产生相应的目标代码。

宏汇编程序 MASM 提供了几十种伪指令，这里介绍几种常用伪指令。

1. 符号定义伪指令

(1) 等值语句 EQU。等值语句用来给某个表达式赋予一个名字，在源程序中就可以用该名字代替表达式。表达式可以是数据、符号、数据地址、程序地址，甚至可以定义为一条可以执行的指令等。EQU 语句格式为：

```
名字  EQU  表达式      ;把表达式的值赋给名字
```

例如：

```
ABC   EQU  2000H          ;表示 ABC = 2000H
ADR   EQU  [BP + 5]       ;表示 ADR 代表地址表达式[BP + 5]
VAR   EQU  ABC * 2 + 256  ;表示 VAR = 表达式的值
COUNT  EQU  CX            ;表示在以后的程序中可用 COUNT 代表 CX 寄存器
```

如果在程序段有了上述语句，则指令语句为：

```
MOV  AX,ABC       ;执行后 AX = 2000H,等同于 MOV  AX,2000H
MOV  AX,COUNT     ;等同于 MOV  AX,CX
```

注：EQU 定义的名字在未解除之前是不能重新定义的。

(2) 解除语句 PURGE。已经用 EQU 命令定义的名字，若要重新定义或以后不再用了，可用 PURGE 语句来解除。PURGE 语句格式：

PURGE 名字 1，名字 2，……，名字 n

用 PURGE 语句解除后的名字可以重新定义。例如：

```
PURGE  VAR
VAR  EQU 3 * 2+56
```

(3) 等号语句“=”。语句功能与 EQU 相同，只是用等号语句定义的名字可以重复定义。例如：

```
COUNT=8
COUNT=10
COUNT=COUNT+1
```

2. 数据定义伪指令

数据定义伪指令是为数据项分配存储单元，用一个变量与这个存储单元相联系，并给变量分配的存储单元初始化。其汇编语言格式为：

[变量名] 数据定义伪指令 操作数 [，操作数……]

变量名：是用户自定义标识符，表示操作数表首个数据项的逻辑地址，即用这个符号表示地址，常称为符号地址。若没有变量名，汇编程序将直接为各操作数分配空间。设置变量名是为了方便存取它指示的存储单元。

数据定义伪指令有 DB、DW、DD、DF、DQ、DT，它们的功能如表 3.9 所示。

表 3.9　　数据定义伪指令功能表

助记符	变量类型	功　能
DB	字节（Byte）	给每个操作数项分配一个 8 位的字节单元，可用于定义字符串
DW	字（word）	给每个操作数项分配一个 16 位的字单元，可用于定义数据、段地址、偏移地址
DD	双字（Dword）	给每个操作数项分配一个 32 位的双字单元，可用于定义数据、含段地址和偏移地址的远指针
DF	3 个字（Fword）	给每个操作数项分配一个 6 字节单元，常用于表示 32 位 CPU 的 48 位远指针
DQ	4 个字（Qword）	给每个操作数项分配一个 8 字节单元，常用于表示 64 位数据
DT	10 个字节（Tbyte）	给每个操作数项分配一个 10 字节单元，常表示 BCD 码、数据（用于浮点运算）

多个操作数项间用逗号分隔，操作数可以是常量、?、带 DUP（重复操作符）的表达式和地址表达式。常量包括常数、字符串常量、符号常量和数值表达式；“?”表示未赋初值，只预留内存空间；多个存储单元有相同初值，可以用重复操作符 DUP 定义。

[例 3.26] 有如下一些变量定义，图 3.26 是相应的内存分配示意图。

```
DATA1    DB   -1*3              ;有符号数以补码形式，-3 补码是 0FDH,存放在 DATA1 所指内存单元
DATA2    DW   0204H,100H        ;DATA2 地址开始定义两个字操作数 0204H、0100H,占 4 个存储单元
DATA3    DD   12345H            ;DATA3 地址开始定义一个双字操作数,位数不足,高位补 0,所以
                                ;相当于 00012345H,占 4 个存储单元
STRING   DB   '0123','A','BC'  ;STRING 开始存放对应字符串中各字符的 ASCII 码,一个字符占一个
                                ;存储单元,共占 7 个存储单元,字符串一般须用 DB 来定义
DATA5    DW   '01','23'         ;如果用 DW 来定义字符串,则操作数项单引号内须是两个字符,每个字符
                                ;占一个存储单元,共占 4 个存储单元
DATA6    DB   ?                 ;从 DATA6 开始预留一个字节单元,单元内容是原来值
DATA7    DB   3  DUP(00)        ;从 DATA7 开始分配 3 个字节单元,初值都为 0
DATA8    DW   2  DUP(?)         ;从 DATA8 开始预留 2 个字,占 4 个单元,单元内容不变
DATA9    DW   DATA5             ;将符号地址 DATA5 的值(偏移地址)存在 DATA9 开始的单元
```

多字节数在存储器中的存放规则：低字节在低地址单元，高字节存在高地址单元；多个操作数时，从左到右由低地址到高地址排列。图 3.26 假设数据段从 2000H：0000H 开始存放，则 DATA5 的偏移地址为 0010H，所以符号地址 DATA9 对应的单元中存放 10H、00H。

3. 分析运算符和合成运算符

在汇编语言源程序中运用标号或变量时会用到分析运算符或合成运算符。分析运算符可从标号或变量中分解出某些属性值，如表 3.10 所示。而合成运算符可对标号或变量的类型属性进行设置，如表 3.11 所示。

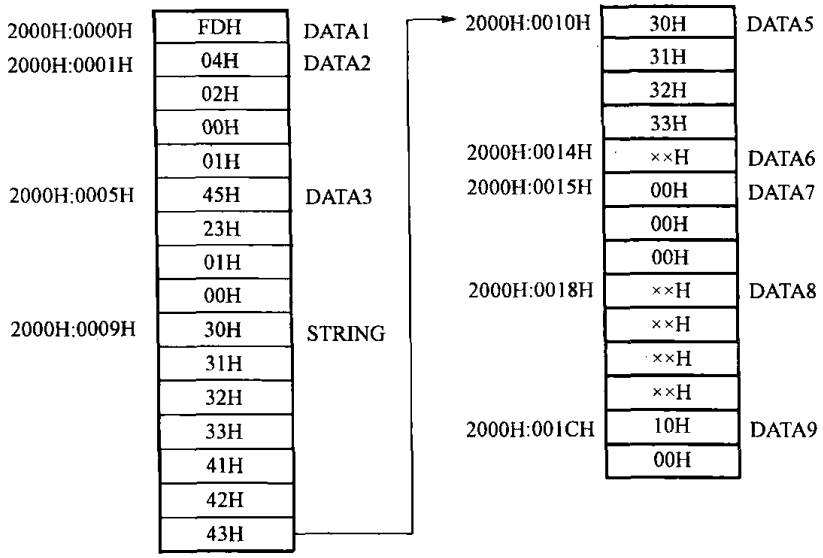


图 3.26 ［例 3.26］相应的内存分配示意图

表 3.10 分析运算符表达式

带分析运算符的表达式	表达式的值
OFFSET 变量名或标号	取出变量名或标号所在段的偏移地址
SEG 变量名或标号	取出变量名或标号所在段的段基地址
TYPE 变量名或标号	对字节型变量名返回值 1，字型变量名返回值 2，双字型变量名返回值 4，三字型变量名返回值 6，四字型变量名返回值 8，十字型变量名返回值 10。对标号进行操作，如果标号类型为 NEAR，则返回－1；如果标号类型为 FAR，则返回－2
SIZE 变量名	取变量存储区的总长度（以字节为单位），即 SIZE＝LENGTH×TYPE
LENGTH 变量名	取变量存储区中变量的元素个数（即变量定义中使用 DUP 的次数，无 DUP 时值为 1）

表 3.11 常用合成运算符

带合成运算符的表达式	表达式的作用
类型名 PTR 变量名或标号	将已定义过的变量名或标号按该表达式指定类型名使用
SHORT 标号	将标号作为短处理
变量名或标号 EQU THIS	用于创建采用当前地址但为指定类型的操作数

说明：

（1）对于变量名，类型名可以是 BYTE（字节）、WORD（字）和 DWORD（双字）；对于标号，类型名可以是 NEAR（近属性，段内）和 FAR（远属性，段间）。

（2）PTR 合成操作符常用于：临时改变变量或标号的类型，使指令中各操作数类型相同；指令中存储器操作数类型不确定时，用 PTR 指定类型。

［例 3.27］ 在［例 3.26］的定义下，有下列语句：

```
MOV     DI,OFFSET DATA1         ;取 DATA1 的偏移地址送 DI,即 DI = 0000H
MOV     AL,BYTE PTR[DI]         ;将 DI 所指的一个存储单元内容送 AL,AL = 0FDH,即取一个字节
MOV     AX,WORD PTR[DI]         ;将 DI 所指的连续两个单元内容送 AX,AX = 04FDH,即取一个字
MOV     AX,SEG DATA1            ;取变量 DATA1 的段基地址送 AX,AX = 2000H
MOV     DS,AX                   ;段基地址送 DS
MOV     AL,LENGTH DATA2         ;执行后 AL = 1
MOV     AL,LENGTH DATA7         ;执行后 AL = 3
MOV     AL,LENGTH DATA8         ;执行后 AL = 2
MOV     AL,SIZE DATA7           ;执行后 AL = 3(即 3×1)
MOV     AL,SIZE DATA8           ;执行后 AL = 4(即 2×2)
```

4. 段定义伪指令

前面已讲过汇编语言源程序是采用分段结构来组织程序（或程序模块）、数据、堆栈的。一个源程序有若干个逻辑段组成。段定义伪指令专门用于定义汇编语言源程序中的逻辑段。段定义伪指令格式为：

```
段名    SEGMENT    [定位类型][组合类型][类别名]
        …          ;本段语句序列(定义数据段、附加数据段和堆栈段时,只能包括伪指令语句,不
                   ;能包括指令语句;定义代码段时,可包括指令语句和与指令有关的伪指令语句。)
段名    ENDS
```

伪指令 SEGMENT 和 ENDS 总是成对使用，用来指定段的开始、结束、段的名称和段属性。段名是程序员给该逻辑段起的一个名称，不能与指令助记符或伪指令符等保留字同名，同一逻辑段在 SEGMENT/ENDS 伪指令前的段名必须相同。带有“[ ]”部分是段属性的说明，按需要可有可无，段属性有定位类型、组合类型和类别名。分别说明如下：

（1）定位类型。定位类型给出逻辑段在内存中的实际地址边界。它有 PAGE、PARA、WORD、BYTE 四种类型：

1）PAGE：表示相应段必须从存储器的某一页（256 个字节）边界开始。即段的起始地址能被 256 整除。此时该段 20 位物理地址为×××00H。

2）PARA：表示相应段必须从存储器的某一个节（16 个字节）的边界开始。即段的起始地址能被 16 整除。此时该段 20 位物理地址为××××0H。

3）WORD：表示相应段可以从存储器的任何一个字边界（偶地址）开始。即段的起始地址能被 2 整除。

4）BYTE：表示相应段可以从存储器的任何地址开始。

合理选择定位类型，能够在进行段和模块的定位连接时，充分地利用存储器空间。

（2）组合类型。组合类型在多模块程序设计中告诉汇编程序该段和其他同名段间如何组合连接。如果没有指明组合类型，则该程序段不与其他段相连接。组合类型有五种：

1）PUBLIC：表示该段与其他同名段在满足定位类型的前提下依次连接构成一个逻辑段。连接的顺序由连接程序 LINK 确定。

2）COMMON：表示该段与其他同名同类别段共享相同的存储空间。即各段都是从相

同的地址开始，互相覆盖。连接后，段的长度等于最长的 COMMON 段的长度。

3）AT 表达式：表示相应段根据表达式的值定位段地址。例如，AT 2000H，表示该段的段基地址为 2000H，本段起始的物理地址为 20000H。

4）STACK：与 PUBLIC 组合类型的处理方式相同，但 STACK 组合方式仅用于堆栈段，并且，在可执行文件装入内存后，段寄存器内已是该段的段基地址，堆栈指针 SP 已指向栈顶。

5）MEMORY：表示在连接时本段应装在同名段中具有最高地址的地方。若连接时有多个 MEMORY 组合类型的段，则只有第一段才当成 MEMORY 组合类型来处理，其他的段将重叠，即按 COMMON 组合类型来处理。

（3）类别名。类别名是程序员任选的一个字符串，使用时必须用单引号括起来。连接时，将把不同模块中相同类别名的各段在物理地址上相邻地连接在一起，其顺序则与 LINK 时提供的各模块顺序一致。

5. ORG 伪指令

一般格式为：　ORG　表达式

伪指令 ORG 用来规定其后的程序或数据块存放的起始地址，即以表达式的值为该起始地址（偏移地址）开始连续存放。ORG 伪指令可以放置在源程序中的任何位置。

```
MY-SEG  SEGMENT
  ORG  1000H              ;后面的数据区从段内偏移地址为 1000H 开始存放
  DATA1  DB  12H,22,34H
  ......
MY-SEG  ENDS
```

6. ASSUME 伪指令

ASSUME 伪指令语句用来告诉汇编程序在指令执行期间使用哪个数据段、附加数据段、堆栈段、代码段，即指明段寄存器所对应的段。其一般格式为：

ASSUME 段寄存器名：段名［，段寄存器名：段名，…］

ASSUME 语句只能出现在代码段内，且一般是代码段内第一条语句。ASSUME 语句中的“段寄存器名：段名”按需要可以有一项或多项，如果一行写不下，可分成两个 ASSUME 语句。

一个源程序模块能允许包括多个代码段和其他段，也允许多次使用 ASSUME 语句，重新约定段寄存器和段的关系。但 ASSUME 语句并不意味着汇编后这些段地址已经装入相应的段寄存器中了。除了 CS 以外，各个段寄存器的实际值，还要用 MOV 指令来赋值。例如：

```
MYDATA  SEGMENT          ;定义数据段 MYDATA
  BUF  DB  10 DUP(11H)
MYDATA  ENDS
MYEXTRA  SEGMENT         ;定义附加数据段 MYEXTRA
  EBUF  DB 10 DUP(?)
MYEXTRA  ENDS
MYSTACK  SEGMENT         ;定义堆栈段 MYSTACK
  SBUF  DB  200 DUP(0)
```

```
MYSTACK  ENDS
MYCODE   SEGMENT
  ASSUME  CS:MYCOED,DS:MYDATA,ES:MYEXTRA,SS:MYSTACK
START:MOV  AX,MYDATA    ;段名具有段地址属性,取数据段 MYDATA 的段基地址给 AX
      MOV  DS,AX        ;段基地址再送 DS
      MOV  AX,MYEXTRA
      MOV  ES,AX
      MOV  AX,MYSTACK
      MOV  SS,AX
      …
MYCODE   ENDS
      …
      END  START
```

7. 过程定义伪指令

通常将具有某种功能的程序段定义成一个过程（即子程序）。一个过程可以被其他程序调用（CALL），它总是至少有一条返回指令，以控制此过程执行完毕后返回到调用程序（主程序）。过程定义伪指令的一般格式为：

```
过程名  PROC  [NEAR/FAR]
    ……
    RET
过程名  ENDP
```

过程定义伪指令 PROC/ENDP 总是成对出现的，其前面的过程名必须相同。过程可以是近过程，此时伪操作 PROC 后面的类型是 NEAR，但可以省略；若过程为远过程，则伪操作 PROC 后面的类型是 FAR，不能省略。过程可以嵌套，即一个过程调用另一个过程。过程也可以递归，即过程可以调用过程本身。

[例 3.28] 编写一个 10ms 延时子程序。

解

```
DELY    PROC              ;定义一个近过程
        PUSH    BX        ;保护 BX 原来内容
        PUSH    CX        ;保护 CX 原来内容
        MOV     BL,2      ;外循环次数
NEXT:   MOV     CX,4167   ;内循环次数(实现延时 5ms)
Y10MS:  LOOP    Y10MS     ;CX≠0 则循环
        DEC     BL        ;修改外循环计数值
        JNZ     NEXT      ;BL≠0 则进行第 2 轮循环
        POP     CX        ;恢复 CX 原来内容
        POP     BX        ;恢复 BX 原来内容
        RET               ;过程返回
DELY    ENDP              ;过程结束
```

[例 3.29] 写一个程序完成多字节 BCD 码相加，并将单字节加法写成过程。

解

```
DATA    SEGMENT                          ;定义数据段
FIRST   DB  11H,22H,33H,44H              ;第一个加数
SECOND  DB  55H,66H,77H,88H              ;第二个加数
SUM     DB  5 DUP(?)                     ;存放结果单元
DATA    ENDS
CODE    SEGMENT                          ;定义代码段
    ASSUME  CS:CODE,DS:DATA,ES:DATA
MAIN    PROC    FAR
START:  MOV     AX,DATA                  ;装入数据段寄存器实际值
        MOV     DS,AX
        MOV     ES,AX
        MOV     SI,OFFSET  FIRST         ;SI 指向第一个加数
        MOV     DI,OFFSET  SUM           ;DI 指向结果单元
        MOV     BX,OFFSET SECOND         ;BX 指向第二加数
        MOV     CX,04                    ;共 4 个字节数
        CLD                              ;清方向标志
        CLC                              ;清进位标志
ADITI:  CALL    AAA1                     ;调用单字节加法子程序
        LOOP    ADITI
        MOV     AL,0
        ADC     AL,AL
        STOSB
MAIN    ENDP                             ;主程序结束
AAA1    PROC NEAR                        ;单字节加法子程序
        LODSB                            ;取第一个加数
        ADC     AL,[BX]                  ;相加
        DAA                              ;十进制调整
        STOSB                            ;结果送 DI 所指单元
        INC     BX                       ;修改指针
        RET                              ;返回
AAA1    ENDP                             ;子程序结束
CODE    ENDS                             ;程序段结束
END     START                            ;程序结束
```

8. *源程序结束伪指令* END

伪指令 END 用来表明本源程序到此处结束。其格式如下：

```
END [表达式]
```

这里的表达式通常就是程序第一条可执行指令语句的标号。该标号就是程序在汇编、连接后，将目标代码装入内存要执行的起始地址。

9. *其他常用伪指令列表*

其他常用伪指令如表 3.12 所示。

表 3.12　　其他常用伪指令简表

伪指令名称	指令格式	功 能 及 说 明
LABEL	变量/标号 LABEL 类型	定义变量、标号等的类型。对变量类型有：BYTE、WORD、DWORD；对记录名或结构名和标号类型有：NEAR 或 FAR
PUBLIC	PUBLIC 符号［，…］	说明符号可被另外的源文件引用，符号可以是数、变量或标号（过程名）、寄存器名或用“EQU”或“=”定义的符号
EXTRN	EXTRN 名字：类型 ［，名字：类型］… ［，名字：类型］	表示该名字由另外源文件定义并由 PUBLIC 说明。类型可以是 BYTE、WORD、DWORD、NEAR、FAR 或用 EQU 定义的符号
NAME	NAME 模块名	给程序模块命名。一个只能有一个 NAME，若没有则从 TITLE 中取或从程序源文件名中取。模块名不能是保留字
COMMENT	COMMENT 定界符注释，定界符	指示注释部分，可自定义定界符。定界符为 COMMENT 之后第一个非空字符
GROUP	组名 GROUP 分段名［，分段名］…，…［，分段名］	表示所说明的分段全部分配在一个物理段中
INCLUDE	INCLUDE 文件说明	源程序级引用源文件，把它调入该语句处
STRUC	结构名 STRUC 域名 DB … ［若干数据定义语句］ 结构名 ENDS	定义一个结构形式。其中域名可用 DB～DT 之间任意一个定义，形式和数据定义语句完全一样，如果有赋值，则引用该结构的变量都赋了值
IF	IF…条件 …（语句串 1） ［ELSE］ …（语句串 2） ENDIF	在条件汇编中，如果满足条件则汇编语句串 1；否则，汇编语句串 2，如果不选择 ELSE，就不汇编这些语句
	IF 表达式…	表达式不为零为满足条件
IFE	IFE 表达式…	表达式是零为满足条件
IF1		如果是第一遍扫描则为真
IF2		如果是第二遍扫描则为真
IFB	IFB<变量>	其中变量为空格时为真
IFNB	IFNB<变量>	其中变量为非空格时为真
ENDIF	ENDIF	表示条件汇编语句定义结束
ELSE	ELSE	表示条件不真时应汇编下面
LOCAL	LOCAL 符号 1［，符号 2］…［，…］	把要出现在宏体中的标号/变量定义成局部标号/变量，符号 1、符号 2 等只在宏展开中有意义且是第一条语句
EXITM	EXITM	该语句用于 MACRO 及重复块中。它表示撤离该部分扩展，即退出宏定义

续表

伪指令名称	指令格式	功 能 及 说 明
.CREF	.CREF	表示输出交叉参考信息
.XCREF	.XCREF	表示不输出交叉参考信息
.LALL	.LALL	列出所有宏扩展文本
.SALL	.SALL	删掉列出的所有的扩展及其目标代码
.XALL	.XALL	凡产生目标代码的就列出其相应源程序
.LIST	.LIST	表示从以下源文件行开始产生列表文件，除非遇到 .XLIST
.XLIST	.XLIST	从以下源文件行开始不产生列表文件，除非遇到 .LIST
%OUT	%OUT text	在汇编时遇到% OUT 则显示 text 部分，如该显示与其他显示同时产生，则删掉该部分
PAGE	PAGE［每页行数］,［每行字符数］或 PAGE+	表示列表文件为页式。没参数时表示准备打印新页。页数加“1”；每页行数在 10～255 之间，标准数是 50，每行字符数在 60～132 之间，标准数是 80
SUBTIL	SUBTIL 子标题 或 SUBTIL	表示每页标题后打印子标题。如果是 SUBTIL，表示以后不印子标题
TITLE	TITLE 标题	表示每页第一行要打印“标题”，每个文件只能有一个 TITLE 语句，标题最长为 80 个字符

3.4.5 宏指令

1. 宏指令、宏定义和宏调用

宏指令是源程序中具有独立功能的一段程序代码。它可以根据用户需要，由用户自己在源程序中定义。宏指令一经定义，便可在以后的程序中多次调用。宏定义由 MASM 宏汇编程序提供的伪指令实现，一般格式为：

```
宏指令名    MACRO  <形式参数>
            …              ;宏体
            ENDM
```

其中，MACRO 和 ENDM 均为伪指令，必须成对出现在源程序中。且必须以 MACRO 作为宏定义的开始，以 ENDM 作为宏定义的结尾。MACRO 和 ENDM 之间为宏体，是实现宏指令功能的语句序列。宏指令名（Macro Name）是为宏指令起的名字，以供在源程序中调用该宏指令时用。形式参数（也称虚拟参数）的设置可根据需要而定，可用一个或多个（最多 132 个），也可以没有。当有多个形式参数时，参数之间必须用逗号隔开。

宏调用的格式为：

```
宏指令名 <实际参数>
```

在源程序中像一般语句一样写上已定义过的宏指令名就可以调用该宏指令了。若宏定义时有形式参数，则宏指令名后写上实际参数，并和形式参数一一对应；若宏定义时该宏指令没有形式参数，则调用宏指令时也不用写实际参数。

具有宏调用的源程序被汇编时，每个宏调用将被汇编程序宏展开。宏展开就是用宏定义

时设计的宏体去代替相应的宏指令名，并且用实际参数取代形式参数，构成完成特定功能的程序代码。实际参数与形式参数的个数不等，将多余的形式参数做“空”处理，而对多余的实际参数自动丢弃。下面我们通过例子说明宏定义、宏调用及宏展开的方法。

[例 3.30] 写4个宏定义分别是保护有关寄存器、取内存单元某数据送AX、AX内容左移4位、结果再存入内存单元。依据此4个宏定义写一段程序，将某变量指针WVAR指向的内存单元的内容送AX寄存器。左移4位后，再存入WVAR变量处。

解

```
PUSHREG   MACRO              ;一个无形式参数的宏定义 PUSHREG
          PUSH   AX
          PUSH   BX
          PUSH   CX
          PUSH   DX
          PUSH   SI
          PUSH   DI
          ENDM
LOADW     MACRO  PR,VAR      ;有 2 个形式参数,互相之间必须用逗号分开
          MOV    PR,VAR      ;形式参数在操作数项中应用
          MOV    AX,[PR]
          ENDM
SHIFT     MACRO  N,REG,CC    ;有 3 个形式参数的宏定义 SHIFT
          MOV    CL,N        ;形式参数 N、REG 在操作数项中应用
          S&CC   REG,CL      ;形式参数 CC 作为操作码一部分,其前须加分隔符“&”
                             ;且该形式参数用实际参数代替时,形成的操作码须是指令系统中的合
                             法的助记符指令
          ENDM
SAVEW     MACRO  PR,REG,OPC  ;有 3 个形式参数宏定义 SAVEW
          MOV    [PR],REG    ;形式参数 PR、REG 在操作数项使用
          OPC    PR          ;形式参数 OPC 是操作码,实参应是指令助记符
          ENDM
```

采用宏调用方法编写的程序段：

```
LOADW     SI,WVAR
SHIFT     4,AX,AR
SAVEW     SI,AX,INC
...
```

宏展开时，汇编程序在所展开的指令前加“+”，以便与其他指令区别。上述程序段宏展开后的目标代码如下：

```
+MOV      SI,WVAR        ;宏 LOADW    SI,WVAR 展开
+MOV      AX,[SI]
+MOV      CL,4           ;宏 SHIFT    4,AX,AR 展开
+SAR      AX,CL
```

```
+MOV    [SI],AX      ;宏 SAVEW    SI,AX,INC 展开
+INC    SI
…
```

又若需将变量 FIRST 内容取入 BX，左移 4 位，则有如下程序段：

```
LOADW   DI,FIRST
MOV     BX,AX
SHIFT   4,BX,AL
SAVEW   DI,BX,INC
…
```

宏展开后程序段如下：

```
+MOV    DI,FIRST     ;宏 LOADW    DI,FIRST 展开
+MOV    AX,[DI]
MOV     BX,AX
+MOV    CL,4         ;宏 SHIFT    4,BX,AL 展开
+SAL    BX,CL
+MOV    [DI],BX      ;宏 SAVEW    DI,BX,INC 展开
+INC    DI
…
```

2. 宏嵌套

宏定义中允许使用宏调用，但所用的宏指令必须先定义过。不仅如此，宏定义中还可以包含宏定义。

[例 3.31] 设计一个程序段实现字节数 FIRST1 * FIRST2＋SECOND1 * SECOND2 运算，并将结果存储起来（不考虑最后结果溢出）。

解 宏定义：

```
MULTIPLY   MACRO      OPR1 ,OPR2,RESULT
           MOV        AL,OPR1
           IMUL       OPR2
           MOV        RESULT,AX
           ENDM
ADDMULT    MACRO          REG,VAR1,VAR2
           MULTIPLY       FIRST1,FIRST2,MULT1
           MULTIPLY       SECOND1,SECOND2,MULT2
           MOV            REG,VAR1
           ADD            REG,VAR2
           MOV            SUM,REG
           ENDM
```

可以看出，在 ADDMULT 宏定义中两次调用了定义过的宏指令 MULTIPLY。

若有宏调用：

```
ADDMULT AX,MULT1,MULT2
```

则展开后的程序段如下：

```
MULTIPLY      FIRST1,FIRST2,MULT1
 + MOV        AL,FIRST1
 + IMUL       FIRST2
MOV           MULT1,AX
MULTIPLY      SECOND1,SECOND2,MULT2
 + MOV        AL,SECOND1
 + IMUL       SECOND2
 + MOV        MULT2,AX
ADDMULT       AX,MULT1,MULT2
 + MOV        AX,MULT1
 + ADD        AX,MULT2
 + MOV        SUM,AX
...
```

[例 3.32] 采用宏定义体内包含宏定义的方法设计一个运算宏指令。

解

宏定义：

```
DEFCALCU  MACRO    CALCULATION ,OPERATOR
   CALCULATION     MACRO       X,Y,Z
                   PUSH        AX
                   MOV         AX,X
                   OPERATOR    AX,Y
                   MOV         Z,AX
                   POP         AX
                   ENDM
   ENDM
```

DEFCALCU 宏指令定义体内包含了一个宏定义 CALCULATION。并且，内层宏定义的宏指令名 CALCULATION 又是外层宏定义的形式参数。由于 CALCULATION 宏指令的定义包含在 DEFCALCU 宏指令的定义体内，要调用 CALCULATION 宏指令，必须先调用 DEFCALCU 宏指令，以便使 CALCULATION 宏指令先得到定义，即需按如下步骤进行：

宏调用

```
DEFCALCU  ADDITION,ADD
```

宏展开

```
 + ADDITION    MACRO    X,Y,Z
               PUSH     AX
               MOV      AX,X
               ADD      AX,Y
               MOV      Z,AX
               POP      AX
               ENDM
```

得到宏指令 ADDITION 的宏定义。这样便可有宏调用：

```
ADDITION  FIRST,SECOND,SUM
```

宏展开：

```
+ PUSH    AX
+ MOV     AX,FIRST
+ ADD     AX,SECOND
+ MOV     SUM,AX
+ POP     AX
```

对于宏指令 DEFCALCU 宏定义体中的形式参数 CALCULATION 及 OPERATIOR，给予不同的实际参数去取代，便会得到不同的运算宏指令，如下宏调用可形成减法的宏定义：

```
DEFCALCU  SUBTRACT,SUB
```

如下宏调用可形成逻辑与的宏定义：

```
DEFCALCU  LOGAND,AND
```

3. 宏指令与子程序

在汇编语言程序设计中，宏指令和子程序都给设计者提供了很大方便。它们都是可被程序多次调用的程序段，并且调用前必须由设计者自己根据需要按一定格式进行定义。然而，宏指令和子程序由于定义方法及格式不同，使用中还有许多不同之处。子程序由 CALL 指令调用，由 RET 指令返回，所以汇编后子程序的机器码只占有一个程序段，不管调用多少次均如此，较为节约内存。宏指令每调用一次宏展开时宏体都要占一个程序段，调用次数愈多，占用内存愈多，程序代码越长。但每调用一次子程序都要保护和恢复返回地址及寄存器内容等，要消耗较多的时间。宏指令调用时不需要这个过程，执行时间较短。

综上所述，当某一需要多次访问的程序段较长、访问次数又不是太多时，选用子程序结构较好。当某一需多次访问的程序段较短、访问次数又很频繁时，选用宏指令结构显然要更好些。

3.4.6 常用 DOS 系统功能调用

微型机的系统软件提供了很多可供用户调用的功能子系统，包括控制台输入输出、基本硬件操作、文件管理、进程管理等。用户可在自己的程序中直接调用这些功能，而无须再自行编写程序来实现它们。

系统软件中提供的功能调用有 DOS（Disk Operation System）功能调用和 BIOS（Basic Input and Output System）功能调用两种。用户调用这些系统服务程序时采用软中断指令 INT n 来实现。用 DOS 和 BIOS 功能调用，会使编写的程序简单、清晰，可读性好而且代码紧凑，调试方便。

BIOS 是被固化在计算机主机板上 Flash ROM 型芯片中的一组程序，包括系统测试程序、初始化引导程序、一部分中断矢量装入程序及外部设备服务程序。BIOS 服务功能见附录 3。

DOS 是 IBM PC 系列微型机的操作系统（最新的 Windows 操作系统也继续提供所有的

DOS 功能调用），负责管理系统的所有资源，协调微型机的操作，其中包括大量可供用户调用的服务程序。

所有 DOS 系统功能调用都是利用软中断指令 INT 21H 来实现，即在程序中需要调用 DOS 功能的时候，只要使用一条 INT 21H 指令即可。INT 21H 是一个具有 90 多个子功能的中断服务程序，可分为设备管理、目录管理、文件管理及其他四个方面，详见附录 3。为便于用户使用这些子功能，INT 21H 对每一个子功能都进行了编号——称为功能号。这样，用户就能通过指定功能号来调用 INT 21H 的不同子功能。DOS 系统功能调用方法如下：

1）功能号送 AH；

2）在指定寄存器中放入该功能所要求的入口参数；

3）执行 INT 21H 指令；

4）分析出口参数。

下面介绍几个最常用的 INT 21H 功能。

1. 键盘输入

键盘上有三类键：一是字符键；二是功能键；三是组合键。

通过 DOS 系统功能的字符输入子功能，可以接收从键盘上输入字符，并以对应的 ASCII 码形式存放。例如，如果程序要求的是其他类型的值，则应自行编程进行转换。这里只介绍单字符输入和字符串输入两种。

（1）单字符输入。功能号 1、7 和 8 都可以接收键盘输入的单字符，并以 ASCII 码形式存放在累加器 AL 中。其中，7、8 号功能无回显，1 号功能有回显（即键盘输入的内容同时也显示在显示器上）。编程时，可根据键入的信息是否需要自动显示来选择三者之一。这些功能常用来回答程序中的提示信息，或选择菜单中的可选项以执行不同的程序段。

[例 3.33] 从键盘接收一个“Y”或“N”字符，以便执行不同程序段。

解

```
...
QKEY:   MOV  AH,1          ;有回显键盘输入。功能号 1 送 AH
        INT  21H           ;等待按键,当按下键时,返回键字符的 ASCII 码在 AL 中
        CMP  AL,'Y'        ;比较键入的是否是 Y
        JE   YES           ;键入字符是'Y'则转 YES 处执行
        CMP  AL,'N'        ;比较键入的是否是 N
        JE   NOY           ;键入字符是'N'则转 NOY 处执行
        JMP  QKEY          ;键入其他字符则转 QKEY 处继续等待键入
YES:    ...
        ...
NOY:    ...
        ...
```

（2）字符串输入。通过调用 DOS 的 0AH 号系统功能调用来实现字符串输入。该功能要求用户指定一个键入缓冲区来存放输入的字符串。缓冲区定义在数据段，其定义格式需按照图 3.27 所示的结构。第一个字节为用户定义的缓冲区长度，若键入的字符数（包括回车符）

大于此值，则蜂鸣器会发出嘟嘟声，且光标不再右移，直到键入回车符为止。缓冲区第二个字节为实际键入的字符数（不包括回车符），由 0AH 号功能自动填入。从第三个字节开始存放键入的字符。可见，缓冲区总长度等于缓冲区长度加 2。在调用本功能前，应把键入缓冲区的起始偏移地址预送 DX 寄存器。

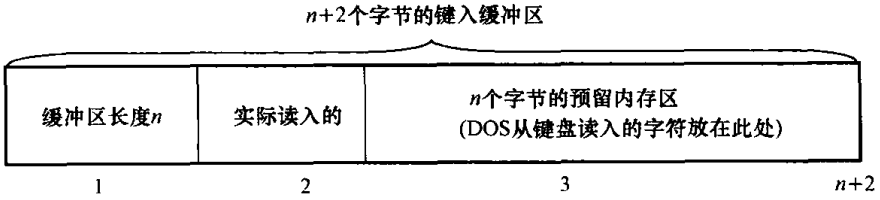


图 3.27 字符串输入缓冲区定义格式

[例 3.34] 编写程序完成从键盘上输入字符串“HELLO”，并在串尾加结束符“$”。

解

```
DATA    SEGMENT
        STRING  DB 10,0,10 DUP(?)       ;定义缓冲区
DATA    ENDS
CODE    SEGMENT
  ASSUME CS:CODE,DS:DATA
START:  MOV  AX,DATA
        MOV  DS,AX
        LEA  DX,STRING                  ;缓冲区偏移地址送 DX
        MOV  AH,0AH                     ;字符串输入功能号 0AH 送 AH 寄存器
        INT  21H                        ;等待键盘输入并从键盘读入字符串
        MOV  CL,STRING+1                ;实际输入的字符个数送 CL
        XOR  CH,CH
        ADD  DX,CX                      ;得到字符串尾地址
        MOV  BX,DX
        MOV  BYTE PTR[BX+2],'$'         ;插入字符串结束符'$'
        MOV  AH,4CH                     ;返回 DOS
        INT  21H
CODE    ENDS
        END START
```

2. 显示器输出

在显示器上显示的内容都是字符形式，如果是数字应是其对应的 ASCII 码。例如，若要在显示器上显示 6，需要先将二进制的 6 转换为 ASCII 码 36H。

要将一个字符串送到显示器上显示，可调用 DOS 功能的 2、6、9 号功能实现。其中，功能号 2、6 用于显示单个字符，功能号 9 用于显示一个字符串。

（1）单字符显示。用功能号 2 显示一个字符的调用格式为：

```
MOV     DL,要显示字符 ASCII 码      ;要显示字符的 ASCII 码须放在 DL 寄存器中
MOV     AH,2                        ;功能号送 AH 寄存器
INT     21H                         ;在 CRT 上显示字符。如果 DL 中为<CTRL>+<BREAK>的 ASCII
                                    ;码,则退出
```

用功能号 6 显示一个字符的调用格式为：

```
MOV    DL,要显示字符 ASCII 码   ;要显示字符的 ASCII 码须放在 DL 寄存器中(不能是 0FFH)
MOV    AH,2                    ;功能号送 AH 寄存器
INT    21H                     ;在 CRT 上显示字符。
```

[例 3.35] 编写程序完成在屏幕上依次显示“1A 2B 3C”6 个字符。

解

```
DATA  SEGMENT
STR   DB '1A 2B 3C'
DATA  ENDS
CODE  SEGMENT
  ASSUME  CS:CODE,DS:DATA
START:  MOV  AX,DATA
        MOV  DS,AX              ;初始化相关段寄存器
        LEA  BX,STR             ;显示字符串第一个字符在内存的偏移地址
        MOV  CX,8               ;循环次数,即要显示的字符数送 CX
LPP:    MOV  AH,2               ;功能号 2 送 AH
        MOV  DL,[BX]            ;取一个要显示的字符送 DL 寄存器
        INC  BX                 ;修改指针,指向下一个字符
        INT  21H                ;调用 21H 中断,在 CRT 上显示字符
        LOOP  LPP
        MOV  AH,4CH             ;返回 DOS
        INT  21H
CODE    ENDS
        END  START
```

(2) 字符串显示。要在显示器上显示字符串，可调用 9 号 DOS 功能号。该功能要求被显示的字符串必须以“$”字符作为结束符，否则会引起屏幕混乱。显示时如果希望光标能自动换行，则应在字符串结束前加上回车及换行的 ASCII 码 0DH 和 0AH。其调用格式：

```
MOV    DX,OFFSET  BUF           ;要显示字符串在内存缓冲区首地址送 DX
MOV    AH,9                     ;功能号 9AH 寄存器
INT    21H                      ;执行系统功能调用
```

[例 3.36] 编写程序完成在屏幕上显示字符串“Example of string display!”。

解

```
DATA  SEGMENT
STR   DB  0DH,0AH,'Example of string display!  $'   ;定义显示字符串,0DH 回车、0AH 换行
DATA  ENDS
STACK SEGMENT  PARA  STACK 'STACK'                  ;定义堆栈段
      DB   100 DUP(0)
STACK ENDS
CODE  SEGMENT
  ASSUME  DS:DATA,CS:CODE,SS:STACK
```

```
BEGIN:MOV  AX,DATA
      MOV  DS,AX
      MOV  AX,STACK
      MOV  SS,AX
      LEA  DX,STR                    ;要显示字符串的首地址送 DX
      MOV  AH ,9                     ;字符串显示功能号 9 送 AH 寄存器;
      INT  21H                       ;在 CRT 上显示字符串
      MOV  AH,4CH                    ;返回 DOS
      INT  21H
CODE  ENDS
      END  BEGIN
```

[例 3.37] 编写程序完成从键盘上输入一字符串到输入缓冲区，然后将输入的字符串在显示器上以相反的顺序显示。

解题思路 先按 0AH 号、9 号功能要求定义输入、输出缓冲区，调用 0AH 号功能输入字符串到 BUFA 缓冲区，通过一段循环程序将输入缓冲区的字符按相反顺序传送到输出缓冲区 BUFB 中，再通过调用 9 号系统功能显示输出。

解

```
DATA  SEGMENT
  INFO1  DB  0DH,0AH,'INPUT  STRING: $'     ;输入字符串开始提示信息
  INFO2  DB  0DH,0AH,'OUTPUT  STRING: $'    ;输出开始提示信息
  BUFA    DB  81                            ;定义输入缓冲区长度
          DB  ?                             ;预留一个字节存放实际输入字符串长度
          DB  80 DUP(0)                     ;定义存放输入字符缓冲区
  BUFB    DB  81 DUP(0)                     ;定义输出字符缓冲区
DATA  ENDS
STACK  SEGMENT  PARA  STACK'STACK'          ;定义堆栈段
          DB  200 DUP(0)
STACK  ENDS
CODE  SEGMENT
  ASSUME  DS: DATA, SS: STACK, CS: CODE
START:MOV  AX,DATA
      MOV  DS,AX
      MOV  AX,STACK
      MOV  SS,AX
      LEA  DX,INFO1
      MOV  AH,9
      INT  21H                          ;9 号功能调用,显示输入提示信息
      LEA  DX,BUFA                      ;输入字符串缓冲区首地址送 DX
      MOV  AH,10                        ;0AH 号功能调用,功能号送 AH 寄存器
      INT  21H                          ;等待键盘输入字符串到缓冲区 BUFA,以回车符结束
      LEA  SI,BUFA + 1
      MOV  CH,0
```

```
        MOV  CL,[SI]                    ;取字符长度→CX
        ADD  SI,CX                      ;SI 为源数据指针指向字符串尾部
        LEA  DI,BUFB                    ;DI 为目的数据指针指向输出字符串缓冲区首地址 BUFB
NEXT:   MOV  AL,[SI]
        MOV  [DI],AL
        DEC  SI
        INC  DI
        LOOP  NEXT                      ;该循环按相反顺序将字符串存放到输出缓冲区
        MOV  BYTE PTR [DI],'$'          ;在输出缓冲区尾部加字符串结束符'$'
        LEA  DX,INFO2
        MOV  AH,9
        INT  21H                        ;9 号功能调用,显示输出提示信息
        LEA  DX,BUFB
        MOV  AH,9
        INT  21H                        ;9 号功能调用,反向显示输入字符串
        MOV  AH,4CH                     ;返回 DOS
        INT  21H
CODE    ENDS
        END  START
```

3. 返回操作系统

一个实际可运行的用户程序在执行完后，应该返回到 DOS 提示符状态，为此，使用正常终止当前程序并返回 DOS 的系统功能调用 4CH 号。其格式为：

```
MOV  AH,4CH
INT  21H
```

其具体使用在前面的举例中大家已见过。

3.5　汇编语言程序设计基础

前面几节分别介绍了 80X86 CPU 的指令系统、汇编语言源程序的格式、伪指令、宏指令及 DOS 系统功能调用。本节结合实例介绍如何综合运用这些知识来设计汇编语言源程序的基本方法，并解决一定的实际工程问题。

参考软件工程理论，汇编语言程序设计与高级语言程序设计一样可按以下步骤进行：

（1）通过对编程任务的分析，抽象出其数学模型，并建立模块结构图。

（2）确定各程序模块的数据结构及算法。对同一个问题可能有不同的算法，一个算法的好坏对程序执行的效率会有很大影响。

（3）绘程序流程图。常用的标准流程图符号如图 3.28 所示。根据算法把程序设计的大纲绘制出来，仔细分析各部分之间的关系并加以修正和完善。

（4）为数据和程序代码分配内存单元和寄存器。

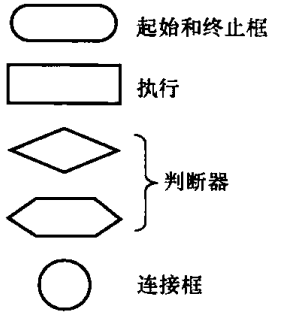


图 3.28　标准流程图符号

(5) 根据流程图编写汇编语言源程序并形成源程序文件 (.ASM)。

(6) 通过汇编生成目标代码文件 (.OBJ)，同时完成静态语法检查。

(7) 对目标文件进行链接生成可执行文件 (.EXE)。

(8) 程序调试。上机运行验证，通过后可进行整个系统测试。

任何一个复杂的程序都是由简单的基本程序构成，汇编语言程序设计也常用到：顺序结构、分支结构、循环结构和子程序结构四种基本程序结构。本节分别介绍这四种程序结构的设计方法。

3.5.1 顺序结构程序

顺序结构也称线性结构，其特点是其中的语句或结构按排列顺序逐条执行。

[**例 3.38**] 试编制一程序，求出下列公式中的 Z 值，并存放在 RESULT 单元中。$Z=\dfrac{(X+Y)\times 8-X}{2}$，其中，$X$、$Y$ 为字值，分别存放在 VARX、VARY 单元中。

解题思路 有两个字值数据 X、Y 参加运算，所以要定义两个 16 位源操作数分别放在 VARX 和 VARY 开始单元；还需要定义一个存放运算结果的单元，由 RESULT 指定，且是 16 位；运算的乘法、除法可以用 MUL、DIV 或用 SAL (算术左移)、SRL (算术右移) 指令。

解 源程序编写如下：

```
   TITLE    EXAMPLE3-41
DATA        SEGMENT
   VARX     DW  6                    ;定义两个 16 位源数据 X、Y
   VARY     DW  7
   RESULT   DW  ?                    ;定义结果存储单元
DATA        ENDS
STACK1      SEGMENT  PARA  STACK
            DW    50 DUP(?)
STACK1      ENDS
COSEG       SEGMENT
   ASSUMENT  CS:COSEG,DS:DATA,SS:STACK1
START:  MOV    AX,DATA
        MOV    DS,AX
        MOV    AX,STACK1
        MOV    SS,AX               ;给各段寄存器赋段基地址,指向本程序定义的各段
        MOV    DX,VARX             ;DX←X
        ADD    DX,VARY             ;DX←(X+Y)
        MOV    CL,3
        SAL    DX,CL               ;DX←(X+Y)×8,用除法指令如何修改?
        SUB    DX,VARX             ;DX←(X+Y)×8-X
        SAR    DX,1                ;DX←((X+Y)×8-X)/2,用除法指令如何修改?
        MOV    RESULT,DX           ;存结果
        MOV    AH,4CH              ;返回 DOS
        INT  21H
COSEG ENDS
        END    START
```

[例 3.39]　自内存 TABLE 开始连续 16 个单元中存放着 0～15 的平方值（即平方表），查表求任意数 X（$0 \leqslant X \leqslant 15$）的平方值，并将结果存入 RESULT 单元中。

解题思路　查表是一个普遍的问题，如查学生成绩表、将一位十六进制数转换成与它相应的 ASCII 码等。以本题为例首先要建立一个表 TABLE，表中按照 0～15 的平方值从小到大的顺序存放在数据段；由表的存放规律可见，表的起始地址与数 X 的和就是 X 的平方值所在单元的地址。

解　编写程序如下：

```
DSEG    SEGMENT
TABLE   DB  0,1,4,9,16,25,36,49,64,81
        DB  100,121,144,169,196,225          ;定义平方表
DATA    DB  4                                ;定义要查表的数 X
RESULT  DB  ?                                ;定义结果存放单元
DSEG    ENDS
SSEG    SEGMENT  PARA  STACK
        DW  50 DUP(0)                        ;定义堆栈段空间
SSEG    ENDS
COSEG   SEGMENT
   ASSUME  CS:COSEG,DS:DSEG,SS:SSEG
BEGIN:  MOV     AX,DSEG                      ;初始化数据段
        MOV     DS,AX
        MOV     AX,SSEG                      ;初始化堆栈段
        MOV     SS,AX
        MOV     BX,OFFSET TABLE              ;置数据指针
        MOV     AH,0
        MOV     AL,DATA                      ;取待查数
        ADD     BX,AX                        ;求得查表地址
        MOV     AL,[BX]                      ;查表
        MOV     RESULT,AL                    ;平方数送 RUSULT 单元
        MOV     AH,4CH
        INT     21H
COSEG   ENDS
        END     BEGIN
```

类似这种查表常使用换码指令 XLAT，会使程序更加精练。请读者试做一下。

3.5.2　分支结构程序

分支结构程序也称条件结构，是根据不同的条件转移到不同的程序段执行相关分支程序。它通常有两种形式，如图 3.29 所示。由图可见，有两分支结构和多个分支结构。它们的共同点是：在某一种确定条件下，只能执行多个分支中的一个分支，而程序的分支要靠条件转移指令来实现。

[例 3.40]　编写计算下面函数值的程序：

$$Y=\begin{cases}1 & ;X>0\\0 & ;X=0\\-1 & ;X<0\end{cases}$$

解题思路　分段函数根据 X 的值给 Y 赋值。先定义一个数据段分别存放 X、Y；取 X 值与数 0 进行比较，如果大于等于 0，再进一步判是否大于 0，是，则给 Y 赋数值 1 并退出程序；否则给 Y 赋数值 0 并退出程序。如果不是大于等于 0，则给 Y 赋数值 -1 并退出程序。为此可画出如图 3.30 所示的流程图。

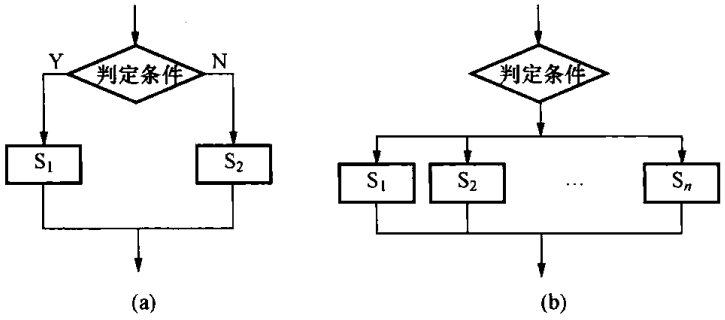


图 3.29　分支程序的结构形式

(a) IF-THEN-ELSE 结构；(b) CASE 结构

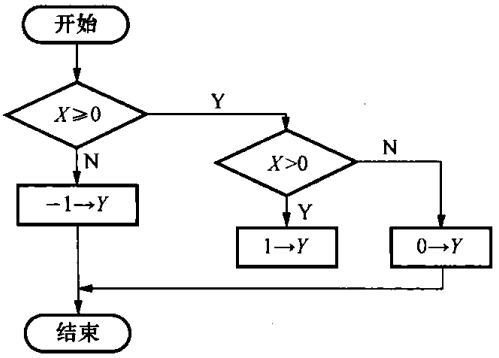


图 3.30　分段（符号）函数流程图

解　程序清单如下：

```
DATA    SEGMENT
        X  DB  -10                  ;定义数据段,假设 X 值为 -10
        Y  DB  ?                    ;给函数值 Y 预留一个单元
DATA    ENDS
SSEG  SEGMENT  PARA  STACK 'STACK'
        DB  200 DUP(0)
SSEG  ENDS
CODE    SEGMENT
   ASSUME  DS:DATA,SS:SSEG,CS:CODE
START:  MOV  AX,DATA                ;给相关段寄存器赋值
        MOV  DS,AX
        MOV  AX,SSEG
        MOV  SS,AX
        CMP  X,0                    ;与 0 进行比较
        JGE  A1                     ;X≥0 转 A1
        MOV  Y,-1                   ;X <0 时,-1→Y
        JMP  EXIT
A1:     JG  A2                      ;X>0 转 A2
        MOV  Y,0                    ;X=0 时,0→Y
        JMP  EXIT
A2:     MOV  Y,1                    ;X>0,1→Y
EXIT:   MOV  AH,4CH                 ;返回 DOS
        INT  21H
CODE    ENDS
        END  START
```

[例 3.41]　从数据段 DATA1 开始的连续 80 个单元中存放 80 位同学某门课的考试成绩（0～100）请编程序统计大于等于 90，80～89 分，70～79 分，60～69 分，小于 60 分的人

数，分别放在同一数据段的 DATA2 开始的 5 个单元中。

解题思路　这是一个典型的多分支结构。先建一个数据段存放 80 位学生的考试成绩，并预留 5 个单元用来存放各分数段的统计结果；取每位学生的成绩放入 AL 并依次与 90、80、70、60 进行比较。因是无符号数，所以用 CF 标志作为分支条件，相应指令为 JC；根据每次比较确定成绩所在范围，用 INC 指令将存放相应结果单元的内容加 1；因为学生和成绩都没有超过一个字节所能表示的范围，故所定义的变量均为字节类型；因每次只能处理一个学生的成绩，所以要通过一个循环来处理 80 位学生的成绩。请读者自行画出流程图。

解　程序清单如下：

```
DATA  SEGMENT
   DATA1  DB  80,69,78,65,45,88,…,92,90      ;80 个学生的考试成绩表
   DATA2  DB  5 DUP(0)                       ;为统计结果留出存储单元
DATA    ENDS
CODE    SEGMENT
   ASSUME  DS:DATA,CS:CODE
START:  MOV  AX,DATA
        MOV  DS,AX
        MOV  CX,80                  ;统计人数送 CX,即循环次数
        LEA  SI,DATA1               ;设学生成绩表指针 SI
        LEA  DI,DATA2               ;设统计结果表指针 DI
AGAIN:  MOV  AL,[SI]                ;取一个学生成绩送 AL
        CMP  AL,90                  ;学生成绩与 90 比较
        JC   NEXT1                  ;若小于 90 分,继续比较判断
        INC  BYTE  PTR[DI]          ;否则是大于等于 90 分,对应结果单元人数加 1
        JMP  STO                    ;转循环控制处理下一个学生成绩
NEXT1:  CMP  AL,80                  ;学生成绩与 80 比较
        JC   NEXT2                  ;若小于 80 分,继续比较判断
        INC  BYTE  PTR[DI+1]        ;否则是 80～89 分,对应结果单元人数加 1
        JMP  STO                    ;转循环控制处理下一个学生成绩
NEXT2:  CMP  AL,70                  ;学生成绩与 70 比较
        JC   NEXT3                  ;若小于 70 分,继续比较判断
        INC  BYTE  PTR[DI+2]        ;否则是 70～79 分,对应结果单元人数加 1
        JMP  STO                    ;转循环控制处理下一个学生成绩
NEXT3:  CMP  AL,60                  ;学生成绩与 60 比较
        JC   NEXT4                  ;若小于 60 分,则转到 60 分以下结果单元加 1
        INC  BYTE  PTR[DI+3]        ;否则是 60～69 分,对应结果单元人数加 1
        JMP  STO                    ;转循环控制处理下一个学生成绩
NEXT4:  INC  BYTE  PTR[DI+4]        ;60 分以下结果单元加 1
STO:    INC  SI                     ;指针指向下一个学生成绩
        LOOP  AGAIN                 ;学生个数减 1,循环,直到 80 个学生成绩处理完
        MOV  AH,4CH                 ;返回 DOS
        INT  21H
CODE    ENDS
        END  START
```

3.5.3 循环结构程序

在程序设计中常碰到某些操作需多次重复执行，这时采用循环结构程序是最为合适的。常见循环结构程序有两种：WHILE _ DO 结构，如图 3.31（b）所示；DO _ UNTIL 结构，如图 3.31（a）所示。WHILE _ DO 结构的主要思想是：当循环控制条件满足时，执行循环体程序，否则退出循环。DO _ UNTIL 结构的主要思想是：先执行循环体程序，再判循环控制条件是否满足。若不满足则执行循环体程序，否则退出循环。编程时可根据具体情况选择。

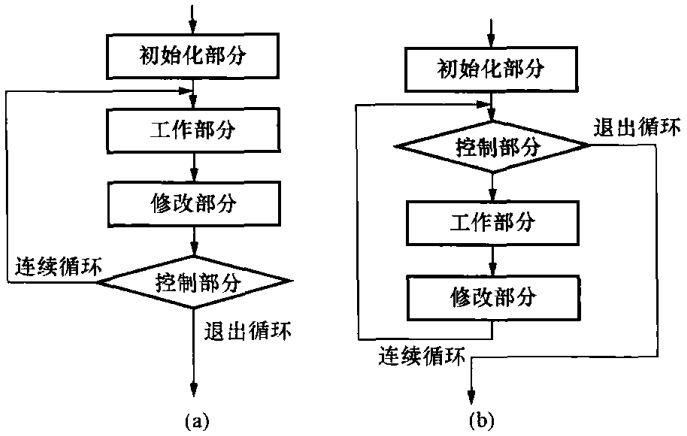


图 3.31　循环程序的结构形式

（a）DO _ UNTIL 结构；（b）WHILE _ DO 结构

每个循环程序必须选择一个控制循环运行和结束的条件。常用循环控制方法有：

（1）计数器控制。循环次数已知的情况。

（2）条件控制。循环次数未知，循环过程中某个特定条件是否满足。

（3）开关变量控制。根据开关变量的值确定进入下一个循环体，有循环条件来控制整个循环的执行。

（4）逻辑尺控制。以某一存储单元中某位是 1 或 0 去执行不同的循环体。

[例 3.42] 在内存某区域有一批 16 位的数，找出其最大值（或最小值）并放到指定单元。

解 本题的解题思路是将这批数一个个拿来比较，每比较一个保留最大（或最小）值，当所有数都比较完后就找到最大值了。程序如下：

```
    NAME  SEARCH_MAX
DSEG  SEGMENT
    BUFFER  DW  23,45,87,…,98          ;定义一批数
    COUNT  EQU  $ - BUFFER             ;求这批数的字节数，$ 表示当前偏移地址值
    MAX  DW  ?                         ;留一个字单元存放最大值
DSEG  ENDS
SSEG  SEGMENT  PARA  STACK 'STACK'     ;定义堆栈段
          DB  100 DUP(?)
SSEG  ENDS
CSEG  SEGMENT
    ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
START:  MOV  AX,DSEG                   ;赋有关段寄存器的值，以指向定义各段
        MOV  DS,AX
        MOV  AX,SSEG
        MOV  SS,AX
        MOV  CX,COUNT/2                ;数据个数送 CX。COUNT 是字节数，除 2 是字数
        LEA  BX,BUFFER                 ;取 BUFFER 的偏移地址送 BX，即数据区指针
```

```
        MOV  AX,[BX]                ;取第一个数送 AX
        INC  BX                     ;指针 BX 加 1
        INC  BX                     ;指针 BX 加 1,指向第二个字
        DEC  CX                     ;数据个数减 1
AGAIN:  CMP  AX,[BX]                ;AX-[BX],即 AX 与下一个数进行比较
        JGE  NEXT                   ;大于等于转 NEXT
        MOV  AX,[BX]                ;否则把大的数送 AX
NEXT:   INC  BX
        INC  BX                     ;调整指针,指向第二个字
        LOOP AGAIN                  ;CX 减 1,CX≠0 转 AGAIN 继续循环
        MOV  MAX,AX                 ;否则 CX=0,保存最大值到 MAX 单元
        MOV AH,4CH
        INT 21H
CSEG    ENDS
        END  START
```

[例 3.43]　内存 ADDR 开始放有 N 个字数据按从大到小的顺序排列。

解题思路　这是一个排序问题，通常采用气泡排序法。从第一个数开始依次对相邻两个数进行两两比较，若次序对，则不交换两数位置；如次序不对则使两数位置交换。第一遍比较共进行 $N-1$ 次后，最小数放到数据区最后。进行第二遍比较，只需考虑前 $N-1$ 个数，即依次对相邻两数进行 $N-2$ 次比较。同理第三遍只需进行 $N-3$ 次……，总共最多进行 $N-1$ 遍就完成排序。这是个多重循环结构，其中 COUNT1 为外循环变量存于 DI 寄存器中，COUNT2 为内循环变量存于 CX 寄存器中。

解　程序清单如下：

```
DSEG    SEGMENT
ADDR    DW  3746, 45, 67, 6721, …, 9855
CUN     EQU  $ - ADDR
DSEG    ENDS
CSEG    SEGMENT
START:  MOV      AX, DSEG
        MOV      DS, AX
        MOV      CX, CUN/2          ;内循环变量存于 CX 中,初始值为 N-1
        DEC      CX
LOOP1:  MOV      DI, CX             ;外循环变量存于 DI 中,初始值为 N-1
        MOV      BX, 0              ;地址指针预置为 0
LOOP2   MOV      AX, ADDR[BX]       ;取相邻两数进行比较
        CMP      AX, ADDR[BX+2]
        JGE      COTINUM            ;若符合排列次序,转移
        XCHG     AX, ADDR[BX+2]     ;若不符合排列次序,两数交换
        MOV      ADDR[BX], AX       ;存大数
COTINUE:ADD      BX, 2              ;修改地址指针
        LOOP     LOOP2              ;若一遍未比较完,继续
        MOV      CX, DI
```

```
        LOOP    LOOP1           ;若 N-1 遍未做完,继续
        MOV     AH,4CH
        INT     21H
CSEG    ENDS
        END     START
```

3.5.4 子程序设计

在程序设计中，某一程序段多处出现，只是某些变量（参数）的赋值不同，这时就应将该程序段设计成一个子程序。它相当于高级语言的过程和函数。子程序执行完成后便返回调用程序处。

1. 过程定义与调用

前面已介绍了过程定义和调用指令为 PROC/ENDP 和 CALL。过程和主程序使用时可在同一代码段内，也可不在同一代码段内。

（1）当过程和主程序在同一代码段时，可采用段内调用方式，即过程具有 NEAR 属性。过程定义和调用格式如下：

```
CSEG    SEGMENT             ;代码段定义
MAIN    PROC    FAR         ;主程序属性应为 FAR,被看作 DOS 的子程序
        …
CALL    SUBT                ;调用子过程 SUBT
        …
        RET                 ;执行完返回 DOS
MAIN    ENDP                ;主程序结束
SUBT    PROC    NEAR        ;SUBT 过程定义,属性为 NEAR
        …                   ;子过程语句序列
        RET                 ;子过程返回
SUBT    ENDP                ;子过程结束
CSEG    ENDS                ;代码段结束
```

（2）当过程和主程序不在一个代码段时，只能采用段间调用方式，即过程具有 FAR 属性。则过程定义和调用格式如下：

```
XSEG    SEGMENT             ;定义代码段 XSEG
SUBT1   PROC  FAR           ;定义过程 SUBT1,具有 FAR 属性
        …
        RET
SUBT1   ENDP
        …
        CALL  SUBT1         ;同一代码段内调用 SUBT1
XSEG    ENDS
YSEG    SEGMENT             ;定义代码段 YSEG
        …
        CALL  SUBT1         ;调用 XSEG 代码段定义的过程 SUBT1
YSEG    ENDS
```

2. 寄存器内容的保护和恢复

一个 CPU 的寄存器数量是有限的，在进行主程序和过程设计时，它们所使用的寄存器会发生冲突。如主程序中某些寄存器内容过程调用后还要用，而被调用过程也用到这些寄存器，这就会造成程序运行出错。为此，在进入过程时（或过程前）将该过程所用寄存器的内容保存起来，称为保护现场。而从过程返回主程序前，再恢复这些寄存器的内容，称为恢复现场。保护现场和恢复现场分别用堆栈压入指令和弹出指令来实现。但在主程序和过程间传递参数和向主程序回送结果的寄存器除外。例如：

```
SUBRP   PROC   FAR
        PUSH   AX          ;按需要保护现场。进入子过程首先保护有关寄存器
        PUSH   BX          ;也可以放在主程序 CALL 指令前,即保护现场放到主程序中
        PUSH   CX
        PUSH   DX
        …                  ;过程语句序列
        POP    DX          ;恢复现场,注意与保护时的顺序,放在子过程 RET 之前
        POP    CX          ;也可以放到主程序 CALL 指令之后恢复现场
        POP    BX
        POP    AX
        RET                ;子过程的返回
SUBPR   ENDP
```

3. 主程序和过程间的参数传送

主程序调用过程时，往往需要一些参数（入口参数）传给过程，过程执行完毕返回主程序时，也往往将过程运行结果和状态（出口参数）回送给主程序。这种主程序和过程间的参数传送可通过寄存器、内存单元、地址表、堆栈等方式进行。

（1）寄存器传送参数。过程的入口参数和出口参数均用寄存器传送。

[例 3.44] 编写一个将存放在内存 4 位十六进制 ASCII 码转换为等值二进制数的过程。

解题思路 十六进制 ASCII 码可能是数字 0～9 的 30H～39H、字母 A～F 的 41H～46H，从最低位 ASCII 码开始，逐位将一个字节的 ASCII 码对于数字减 30H、字母减 37H 转换为等值二进制数，最后将它们拼装起来即可。入口参数：存放 4 位十六进制 ASCII 码的内存首地址送入 DS：SI，ASCII 码的位数 4 存放在 CL 中；出口参数：转换结果（等值二进制数）存放在 DX 中。

解 程序清单如下：

```
DATA        SEGMENT
ASC_STG     DB  32H,35H,42H,38H          ;4 位 ASCII 码数据
BIN_RESULT  DW  ?                        ;存放等值二进制数
DATA        ENDS
XSEG        SEGMENT
            ASSUME CS:XSEG,DS:DATA
MAIN        PROC    FAR
START:      MOV     AX,DATA
```

```
        MOV     DS,AX
        MOV     SI,OFFSET ASC_STG
        MOV     CL,4
        CALL    ASC_BIN
        MOV     BIN_RESULT,DX
        RET                          ;可用 MOV AH,4CH;INT 21H 代替
MAIN    ENDP
ASC_BIN PROC
        PUSH    AX                   ;保护寄存器内容
        MOV     CH,CL                ;位数送 CH
        CLD                          ;方向标志清零
        XOR     AX,AX                ;AX 及 DX 清零
        MOV     DX,AX
AGAIN:  LODSB                        ;(DS:SI)→AL,SI+1→SI
        AND     AL,7FH               ;设置标志位并 AL 最高位清零
        CMP     AL,'9'               ;判该位 ASCII 码是否大于 39H
        JG      A_TO_F
        SUB     AL,30H
        JMP     ROTATE
A_TO_F: SUB     AL,37H
ROTATE: OR      DL,AL                ;拼装各位转换结果
        ROR     DX,CL
        DEC     CH
        JNZ     AGAIN
        POP     AX                   ;恢复 AX 值
        RET
ASC_BIN ENDP
XSEG    ENDS
        END     START
```

(2) 内存单元传递参数。

[例 3.45] 在程序中需对 N 个元素的数组求和。完成数组 N 个元素求和设计为属性是 NEAR 的过程，且过程及主程序在同一代码段。

主程序中将入口参数直接定义到一个数据段里，子程序从该数据段内存单元读取入口参数，子程序执行结果（返回参数）直接写入内存单元。程序如下：

```
DSEG    SEGMENT
ARY     DW  100,23,456,12,45,…,87    ;定义 N 个字元素
COUNT   EQU  ($-ARY)/2               ;求元素个数并赋给 COUNT
SUM     DW  2  DUP(?)                ;留两个字单元存放和值
DSEG    ENDS
SSEG    SEGMENT  STACK  'STACK'
        DB  100 DUP(?)
SSEG    ENDS
```

```
CSEG    SEGMENT
    ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
START:  MOV     AX, DSEG
        MOV     DS, AX
        MOV     AX, SSEG
        MOV     SS, AX
        CALL    PROADD          ;调过程
        MOV     AH, 4CH         ;返回 DOS
        INT     21H
PROADD  PROC    NEAR            ;过程 PROADD 定义
        PUSH    AX              ;保护现场
        PUSH    CX
        PUSH    DX
        PUSH    SI
        LEA     SI, ARY         ;取数组首地址
        MOV     CX, COUNT       ;数组元素个数
        XOR     AX, AX          ;和低 16 位清零
        MOV     DX, AX          ;和高 16 位清零
AGAIN   ADD     AX, [SI]        ;累加 N 个元素的低位字
        JNC     NEXT
        INC     DX              ;若有进位,则和的高位字加 1
NEXT:   ADD     SI, 2           ;调整指针,指向数组下一个数
        LOOP    AGAIN
        MOV     SUM, AX         ;和值低 16 位送结果单元
        MOV     SUM + 2, DX     ;和值高 16 位送结果单元
        POP     SI              ;恢复现场
        POP     DX
        POP     CX
        POP     AX
        RET                     ;过程返回
PROADD  ENDP
CSEG    ENDS
        END     START
```

(3) 地址表传送参数。仍以数组元素求和为例，从［例 3.45］可见有 3 个变量：与数组首元素地址联系的数组名、数组元素个数及累加和。每个变量都有对应的偏移地址，段基地址均在 DS 寄存器中。把它们的偏移地址顺序存放在一张地址表中，该表长度为 3 个字。若需调用过程，应先将地址表的首地址送某寄存器，如 BX。进入过程后可采用寄存器间接寻址方式从地址表中取出变量地址，以便访问所需变量。请读者试着编写程序。

(4) 堆栈传送参数或参数地址。其方法是调用过程前在主程序中用 PUSH 指令将参数或参数地址压入堆栈；进入过程后再用 POP 从堆栈中取出参数或参数地址。

［例 3.46］ 从一个字符串中删除一个字符。

解 程序如下：

```
DSEG    SEGMENT
   STRING  DB  'Exxperiene…'          ;定义字符串
   LENG    DW  $-STRING               ;取字符串长度
   KEY     DB  'x'                    ;定义要从字符串删除的字符
DSEG    ENDS
SSEG    SEGMENT  PARA STACK 'STACK'
        DB  100 DUP(?)
SSEG    ENDS
CSEG    SEGMENT
   ASSUME  CS:CSEG,DS:DSEG,ES:DSEG,SS:SSEG
MAIN    PROC       FAR
START:  MOV        AX, DSEG
        MOV        DS, AX
        MOV        ES, AX
        MOV        AX, SSEG
        MOV        SS, AX
        LEA        BX, STRING
        LEA        CX, LENG
        PUSH       BX              ;参数地址 STRING、LENG 压入堆栈
        PUSH       CX
        MOV        AL, KEY         ;要删除字符送 AL 寄存器
        CALL       DELCHR          ;调用删除一个字符的子程序
        MOV        AH, 4CH         ;返回 DOS
        INT        21H
MAIN    ENDP
DELCHR  PROC                       ;定义删除一个字符子程序
        PUSH       BP              ;保存 BP 内容
        MOV        BP, SP          ;将 BP 指向当前栈顶
        PUSH       SI              ;保护 SI、DI
        PUSH       DI
        CLD                        ;设增量方式
        MOV        SI, [BP+4]      ;得到 LENG 地址
        MOV        CX, [SI]        ;取串长度送 CX
        MOV        DI, [BP+6]      ;得到 STRING 地址
        REPNE      SCASB           ;查找待删除字符
        JNE        DONE            ;若没有找到则退出
        MOV        SI, [BP+4]      ;否则找到,SI 指向串长度单元
        DEC        WORD PTR[SI]    ;串长度减 1
        MOV        SI, DI
        DEC        DI
        REP        MOVSB           ;被删除字符后的字符依次向前移位
DONE:   POP        DI              ;恢复寄存器内容
```

```
        POP     SI
        POP     BP
        RET                                ;返回
DELCHR  ENDP
CSEG    ENDS
        END     START
```

4. 过程的嵌套、递归调用和可重入性

过程也可以作为调用程序去调用其他过程，称为过程的嵌套。一般来说，嵌套层次没有限制，只要堆栈空间允许即可。

当子程序嵌套时要调用的子程序就是该子程序本身，称为子程序的递归调用。能够进行递归调用的子程序称为递归子程序（递归过程）。递归过程被递归调用时必须保证不破坏前面调用所用到的参数及产生的结果，否则，就不能求出最后结果。此外，递归过程还必须具有递归结束的条件，以便在递归调用一定次数后退出，否则，递归调用将无限地嵌套下去。为了能在每次递归调用后保留该次所用到的参数和运行结果，并且不互相冲掉，必须对每次递归调用所用到的参数和运行结果都专门分配一个存储区域。通常将一次递归调用所存储的信息称为帧（Frame）。一帧信息包括递归调用时的人口参数、出口参数、寄存器内容及返回地址等。解决递归调用每帧信息存储的最好方法是采用堆栈，每次递归调用时用 PUSH 指令将一帧信息压入堆栈；每次返回时，再从堆栈中弹出一帧信息。

[例 3.47] 写一个汇编语言程序，实现 $N!$（$N\geqslant 0$）[$N!=N\times(N-1)\times(N-2)\times\cdots\times 1$]。

解题思路　求 $N!$ 本身是一个子程序，由于 $N!=N\times(N-1)!$，求 $(N-1)!$ 必须调用求 $N!$ 的子程序，但每次调用的入口参数和中间结果都不一样，可见是个递归调用过程。具体来说，每次调用时应将入口参数 N 减 1，以便求 $(N-1)!$ 用。当 N 减为 0 时，有 $0!=1$ 的结果。因而 $N=0$ 为递归调用的约束条件来控制退出递归调用的时间。最后，将每次递归调用的入口参数 N 相乘，即可得到最后结果。由于每次递归调用时，一帧信息是按次序逐个、逐层压入堆栈的，故当 $N=0$ 退出递归调用开始返回时，便按嵌套的层次逐层返回，并逐层取出相应的一帧信息，进行乘法运算。得到最终结果后，便按堆栈中内容返回主程序。程序框图如图 3.32 所示。

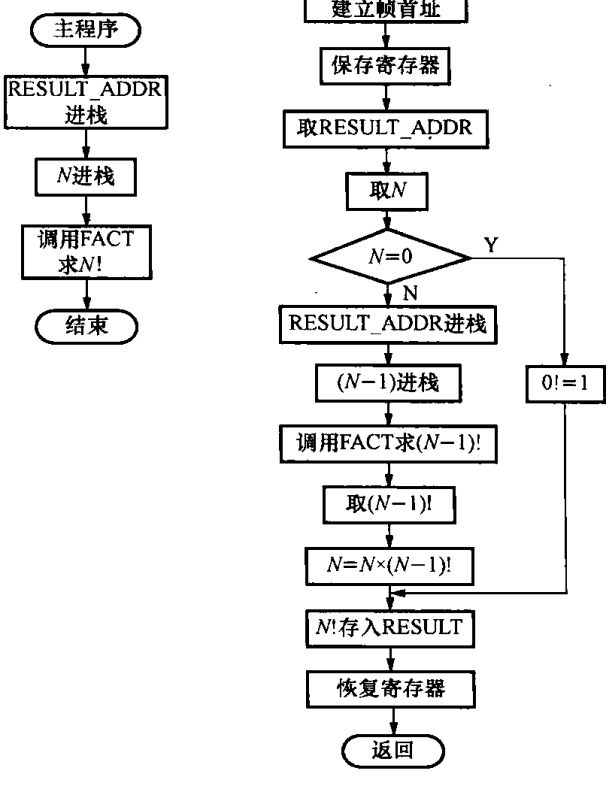


图 3.32　计算 $N!$ 的程序框图

解　程序清单如下：

```
DATA    SEGMENT                          ;定义数据段
VAL-N   DW  N
RESULT  DW  ?
DATA    ENDS
SSEG    SEGMENT  PARA STACK 'STACK'      ;定义堆栈段
        DW 128 DUP(0)
TOP     LABEL  WORD
SSEG    ENDS
CODE1   SEGMENT
MAIN    PROC     FAR
   ASSUME CS:CODE1, DS:DATA, SS:SSEG
START:  MOV      AX, SSEG                ;设置 SS 及 SP
        MOV      SS, AX
        MOV      SP, OFFSET TOP
        PUSH     DS                      ;将 PSP 中 INT 20H 的首址压入堆栈
        SUB      AX, AX
        PUSH     AX
        MOV      AX, DATA
        MOV      DS, AX
        MOV      BX, OFFSET RESULT       ;将运算结果的地址压入栈
        PUSH     BX
        MOV      BX,VAL_N                ;将参数 N 压入栈
        PUSH     BX
        CALL     FAR PTR FACT            ;递归调用 FACT 子程序
        RET
MAIN    ENDP
FRAME   STRUC                            ;定义结构数据模式
        SAVE_BP      DW  ?
        SAVE_CS_IP   DW  2  DUP(?)
        N            DW  ?
        RESULT_ADDR  DW  ?
FACT    PROC     NEAR
        PUSH     BP                      ;存 BP
        MOV      BP, SP                  ;BP 作为帧指针
        PUSH     BX                      ;保护寄存器内容
        PUSH     AX
        MOV      BX, RESULT_ADDR[BP]     ;结果地址送 BX
        MOV      AX, [BP + N]            ;取 N
        CMP      AX, 0                   ;N = 0?
        JE       DONE
        PUSH     BX                      ;下一次的结果地址入栈
        DEC      AX                      ;N - 1 压入栈
        PUSH     AX
```

```
        CALL    PTR FACT
        MOV     BX, RESULT_ADDR[BP]     ;结果地址送 BX
        MOV     AX, [BX]                ;上次的结果取入 AX
        MUL     [BP + N]                ;(AX) = N * RESULT
        JMP     SHORT RETURN
DONE:   MOV     AX, 1                   ;0! = 1
RETURN: MOV     [BX], AX                ;存最终结果
        POP     AX
        POP     BX
        POP     BP
        RET     4
FACT    ENDP
CODE1   ENDS
        END     START
```

为了弄清子程序递归调用中堆栈的压入和弹出情况，图 3.33 给出了 $N=3$ 时该子程序递归调用中堆栈的变化状态。可以看出，每调用一次 FACT 递归子程序，就向堆栈中压入一帧信息，直到 $N=0$ 为止，并开始返回。返回时每推出一帧信息便计算一次中间结果，最后返回主程序，堆栈恢复原状，并在 RESULT 单元得到运算结果为 6。

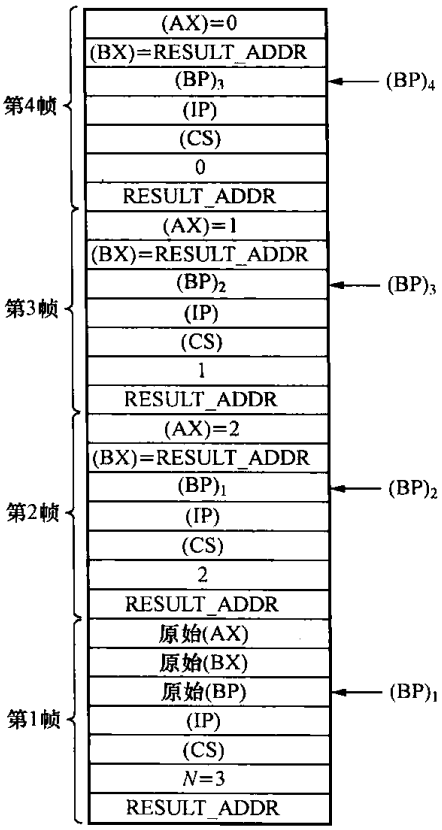


图 3.33　求 3! 时的堆栈状态

3.5.5　常用子程序及程序设计举例

编制好常用子程序后应写出子程序说明文件，以利于其他程序调用。本节列举一些常用子程序，并给出说明文件。另外介绍了一些常见的汇编语言程序设计的实例，供读者参考。

[例 3.48]　两个多字二进制数加法子程序。

子程序说明文档如下：

（1）子程序名：MW _ ADD。

（2）子程序功能：完成两个多字二进制数加法。

（3）入口参数及其传送方法：数据段中的被加数、加数，它们的首地址分别存放在 DS：SI 及 DS：DI 中；被加数或加数的字数在 CX 中。

（4）出口参数及其传送方法：运算结果在内存首地址为 DS：BX 中存放。

（5）所用寄存器：该子程序返回调用程序后不改变任何寄存器原来的内容。

解　程序清单如下：

```
MW_ADD  PROC    FAR             ;保护寄存器内容
        PUSH    AX
        CLC                     ;CF = 0
REPEAT: MOV     AX,[SI]         ;取被加数的一个字
```

```
        INC     SI
        INC     SI
        ADC     AX,[DI]          ;加上加数的相应字位
        INC     DI
        INC     DI
        MOV     [BX],AX          ;将一个字位的和存放内存
        INC     BX
        INC     BX
        LOOP    REPEAT           ;重复循环执行字加法运算
        POP     AX
        RET
MW-ADD  ENDP
```

[例 3.49] 双字乘法子程序。

子程序说明文件如下：

(1) 子程序名：DWMULT。

(2) 功能：完成双字乘双字的运算。

(3) 入口参数及其传送方法：被乘数在内存中的首地址存放在 DS：SI 中；乘数在内存中的首地址存放在 DS：DI 中。

(4) 出口参数及其传送方法：乘积在内存中的首地址存放在 DS：DI 中。

(5) 所用寄存器：该子程序返回调用程序后，不改变任何寄存器的原来内容。

解 子程序清单如下：

```
DWMULT  PROC    FAR
        PUSH    AX
        PUSH    DX
        MOV     AX, [SI]              ;取低 16 位
        MUL     WORD PTR[DI]          ;低 16 位相乘运算
        MOV     [BX], AX              ;保存低 16 位乘积
        MOV     [BX + 2], DX
        MOV     AX, [SI + 2]          ;取高 16 位
        MUL     WORD PTR [DI]         ;高 16 位相乘运算
        MOV     [BX + 4]. DX          ;高位字暂存
        ADD     [BX + 2], AX          ;低位字加到高位字存储单元
        ADC     [BX + 4], 0           ;向高位加进位
        MOV     AX, [SI]              ;取低 16 位
        MUL     WORD PTR [DI + 2]
        ADD     [BX + 2], AX
        ADC     [BX + 6], DX
        MOV     AX,[SI + 2]           ;取高 16 位
        MUL     WORD PTR [DI + 2]
        ADD     [BX + 4], AX
        ADC     [BX + 6], DX
```

```
        POP     DX
        POP     AX
        RET
DWMULT  ENDP
```

[例 3.50] 将内存二进制数转化成以压缩 BCD 码形式存储的十进制数。

解题思路 设一个 32 位无符号二进制数存储在双字单元 BINNUM 中，将它转化成压缩 BCD 码表示的十进制数，存储在 5 个字节单元 DECINUM 中，DECINUM 的低字节存放高位。设计一个表示十进制数的表，用 BINNUM 中的数先减表中最大的数，直到不够减，则减的次数就是最高位的值；然后再减第二大的数，直到不够减，则减的次数就是次高位的值，……，2^{32} 大约相当于 40 亿，故在程序中设 10 个十进制位数就可以了，用 5 个字节单元。

解 程序清单如下：

```
.MODEL  SMALL3-55
.386                                        ;切换到 386 实地址方式
DATA    SEGMENT
        BINNUM  DD 12345678H                ;内存中的二进制数
        DECINUM DB 5 DUP(0)                 ;转化成压缩的 BCD 码的存储区
        NUMBER  DD 1000000000,10000000,10000000,1000000
                DD 100000,10000,1000,100,10,1;减数
DATA    ENDS
CODE    SEGMENT
   ASSUME CS:CODE, DS:DATA
MAIN    PROC    FAR
START:  PUSH    DS
        MOV     AX,0
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX                      ;初始化
        MOV     EAX, BINNUM                 ;将数放入 EAX
        MOV     SI, OFFSET DECINUM          ;SI 指向 BCD 码存储区
        MOV     DI, OFFSET NUMBER           ;DI 指向减数区
        MOV     CX, 5                       ;循环 5 次
ROTATE: CALL    BINDECI                     ;调用二进制转化成十进制子程序
        ADD     DI, 8                       ;DI 下移两个双字
        INC     SI                          ;SI 下移一个字节
        LOOP    ROTATE                      ;若 CX≠0 则循环,输出下两位
        RET
MAIN    ENDP
BINDECI PROC    NEAR
        PUSHF
        PUSH    CX                          ;入栈
        MOV     CX, 0                       ;CX 存放商,初始清 0
        CLC                                 ;清除 CF
```

```
DO_AGAIN1:SUB     EAX, DWORD PTR[DI]          ;EAX 减去相应的减数
          JC      NEXT1                       ;若不够减,则跳出循环
          JNC     CL                          ;否则,CL 加 1(商 0)
          JMP     DO_AGAIN1                   ;继续减
NEXT1:    MOV     CH, CL                      ;将商移入 CH
          MOV     CL, 0                       ;CL 清零
          ADD     EAX, DWORD PTR[DI]          ;恢复最后一次减前的值
          CLC                                 ;清除 CF
DO_AGAIN2:SUB     EAX, DWORD PTR[DI + 4]      ;EAX 减去下一个数
          JC      NEXT2                       ;若不够减,则跳出循环
          INC     CL                          ;否则,CL 加 1(商)
          JMP     DO_AGAIN2                   ;继续减
NEXT2:    ADD     EAX, DWORD PTR[DI + 4]      ;恢复最后一次减前的值
                                              ;CH:CL 中放的是高两位商,是非压缩的 BCD 码
          SHL     CH, 1                       ;下面将它们转化成压缩的 BCD 码并放入存储单元
          SHL     CH, 1
          SHL     CH, 1
          SHL     CH, 1                       ;CH 左移 4 位
          OR      CH, CL                      ;CH 中存放压缩的 BCD 码
          MOV     BYTE PTR[SI], CH            ;放入存储区
          POP     CX
          POPF
          RET
          BINDECI ENDP
CODE      ENDS
          END     START
```

[例 3.51] 将内存中以压缩的 BCD 码形式存储的十进制数转化成二制数。

解题思路 一个 10 位 BCD 码表示的十进制数存储在 DECINUM 开始 5 个字节单元中，高位数放在低地址上。将它转化成 32 位二进制数，存储在 BINNUM 开始的 4 个字节单元中。程序算法是依次从高位（低地址字节）取 DECINUM 的各位，然后乘以系数 10，将它与下一位取出的数相加，其和再乘以 100，再取下一个字节的两个 BCD 码，高位乘 10 再加低位，和再加到原来的和上，再乘以 100……这样依次循环，直至取完为止。乘以 10 的操作可用循环移位来实现，这样可加快速度。

解 程序清单如下：

```
.MODEL    SMALL3-56
.386
DATA      SEGMENT
          BINNUM      DD ?
          DECINUM     DB 12H,34H,56H,78H,90H   ;用十六进制数表示压缩的 BCD 码
DATA      ENDS                                 ;十进制数是 1234567890
CODE      SEGMENT
          ASSUME      CS:CODE, DS:DATA
```

```
MAIN    PROC    FAR
START:  PUSH    DS
        MOV     AX, 0
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX                  ;初始化
        MOV     EAX, 0                  ;将 32 位二进制初始值为 0
        MOV     SI, OFFSET DECINUM      ;DI:SI 指向 DECINUM
        MOV     DI, OFFSET NUMBER       ;DI 指向减数区
        MOV     CX, 5                   ;循环 5 次
ROTATE: CALL    DECIBIN
        INC     SI                      ;SI 指向下一个字节
        LOOP    ROTATE
        MOV     DWORD PTR BINNUM, EAX   ;将 32 位二进制数放入 BINNUM
        RET
MAIN    ENDP
BECIBN  PROC    NEAR
        PUSHF
        PUSH    ECX
        PUSH    EBX
BEGIN:  MOV     ECX, 100                ;ECX 放置乘数 100
        MUL     ECX                     ;ECX 乘 100
        MOV     CH, BYTE PTR[SI]        ;取出一个压缩的 BCD 码到 CH
        MOV     CL, CH                  ;复制到 CL
        AND     CL, 0FH                 ;CL 中保留低 4 位
        AND     CH, 0F0H                ;CH 中保留高 4 位,这时 CH 相当于非压缩的 BCD 码×16
        SHR     CH, 1                   ;算术右移 1 位,相当于非压缩的 BCD 码乘 8
        MOV     BH, CH                  ;移入 BH 暂时保存
        SHR     CH, 1                   ;再算术右移 1 位
        SHR     CH, 1                   ;再算术右移 1 位,相当于非压缩的 BCD 码乘 2
        ADD     CH, BH                  ;相当于非压缩的 BCD 码乘 10
        ADD     CL, CH                  ;加到 CL 上
        MOV     CH, 0                   ;ECX 保存两位 BCD 码转化成二进制数的结果
        ADD     EAX, ECX                ;加到 EAX 上去
        POP     EBX
        POP     ECX
        POPF
        RET
BINDECI ENDP
CODE    ENDS
        END     START
```

[例 3.52] 从键盘输入一字符串，并在内存中已有的一张表中查找该字符串，若找到则在屏幕上显示“OK!”，否则显示“NO!”；若输入字符串长度大于表中字符串长度，则显

示“Wrong!”。

解题思路 首先判断输入字符串长度是否大于表中字符串长度，是则显示“Wrong!”。否则就进行比较；在表中先查找字符串的第一个字符，若找到再比较字符串的其他字符是否一致；显示一个字符串可用9号DOS系统功能调用，从键盘上输入一个字符串可用0AH号DOS系统功能调用。

解 程序如下：

```
DSEG    SEGMENT
TABLE   DB  'SGFGGHYTBJDOWALAJQZV'                ;表字符串数据
STRING1 DB  'Please enter a string:', 0DH,0AH, '$'  ;有关提示信息定义
STRING2 DB  'Wrong!', 0DH, 0AH, '$'
STRING3 DB  'OK!', '$'
STRING4 DB  'NO!', '$'
BUFFER  DB  40, ?, 40 DUP(?)                      ;键盘输入缓冲区
TAB_LEN EQU  STRING1 - TABLE                      ;表字符串长度
DSEG    ENDS
SSEG    SEGMENT
        DB  100 DUP(?)
SSEG    ENDS
CSEG    SEGMENT
   ASSUME  CS:CSEG, DS:DSEG, SS:SSEG
START:  MOV     AX, DSEG
        MOV     DS, AX
        MOV     ES, AX
        LEA     DX, STRING1                 ;显示"Please enter a string:"
        MOV     AH, 09H
        INT     21H
        LEA     DX, BUFFER                  ;键盘输入缓冲区首地址送 DX
        MOV     AH, 0AH
        INT     21H                         ;等待和接收键盘输入
        MOV     SI, DX                      ;键盘输入缓冲区首地址送 SI
        INC     SI
        MOV     BL, [SI]                    ;输入串长度 BX
        MOV     BH, 0
        INC     SI                          ;输入串首地址送 SI
        LEA     DI, TABLE                   ;表首地址送 DI
        MOV     CX, TAB_LEN                 ;表字符串长度送 CX
        CMP     CX, BX                      ;表长与串长比较
        JNC     GOON                        ;表长大于串长则转 GOON
        LEA     DX, STRING2                 ;否则显示 Wrong!
        JMP     EXIT
GOON:   CLD                                 ;设地址增量方式
        MOV     AL, [SI]
```

```
SCAN:   REPNZ   SCASB                   ;在表中搜第一个字符
        JZ      MATCH                   ;找到则转 NATCH
ERROR:  LEA     DX, STRING4             ;没找到显示 NO!
        JMP     EXIT
MATCH:  INC     CX
        CMP     CX, BX                  ;剩余表长和串长比较
        JC      ERROR                   ;剩余表长不大于串长显示 NO!
        PUSH    CX                      ;保存循环变量
        PUSH    SI
        PUSH    DI
        MOV     CX, BX
        DEC     DI
        REPZ    CMPSB                   ;比较串中其余字符
        POP     DI                      ;恢复循环变量
        POP     SI
        POP     CX
        JZ      FOUND                   ;找到字符串转 FOUND
        JCXZ    ERROR                   ;未找到字符串且全表搜完,转 ERROR
        JMP     SCAN                    ;全表未搜完,转 SCAN
FOUND:  DEC     DI                      ;找到字符串偏移地址送 DI
        LEA     DX, STRING3             ;显示 OK!
EXIT:   MOV     AH, 09H
        INT     21H
        MOV     AH, 4CH                 ;返回 DOS
        INT     21H
CSEG    ENDS
        END     START
```

[例 3.53]　在分辨率为 640×480、16 色的屏幕上绘制一个周期的正弦波。

解题思路　正弦波一个周期的角度值范围为 0～360°，函数值范围为－1～1。要使曲线居于屏幕正中，须调整水平和垂直方向的坐标值；在给定 0～90°的函数值情况下，绘制正弦波曲线时须先知道角度所在的象限：在第Ⅰ象限，函数值为正，此时可直接查表取函数值；在第Ⅱ象限，函数值为正，可用 $\sin(x)=\sin(180°-x)$ 将角度转换到第Ⅰ象限后再查表取函数值；在第Ⅲ、Ⅳ象限，函数值为负，先将 $x-180°$ 转换到第Ⅰ象限或第Ⅱ象限，再按前述处理；为简化程序设计在绘图前先计算出曲线各点的坐标值并存入表格，在画图时只需访问这个表格就可以了；设正弦波图形范围为 360×400，表格 SINE 中为从 0～90°的已放大 200 倍后取整的正弦值。

解　程序如下：

```
SETSCREEN   MACRO                   ;设置屏幕分辨率为 640×480、16 色图形方式
    MOV     AH,0
    MOV     AL,12H
    INT     10H
```

```
    ENDM
WRITEDOT    MACRO                    ;画点宏定义
    MOV     AH,0CH
    MOV     AL,02H                   ;像素颜色代码
    MOV     CX,ANGLE                 ;像素点对应的列号送 CX
    ADD     CX,140                   ;X方向屏幕中心=(640-360)/2
    MOV     DX,TEMP                  ;像素点所在的行号送 DX
    INT     10H
    ENDM
DSEG    SEGMENT                      ;定义坐标表格
SINE    DB  00,03,07,10,14,17,21,24,28,31,35,38,42,45,48,52,55,58,62,65
        DB  68,72,75,78,81,85,88,91,94,97,100,103,106,109,112,115,118,120
        DB  123,126,129,131,134,136,139,141,144,146,149,151,153,155,158,160
        DB  162,164,166,168,170,171,173,175,177,178,180,181,183,184,185,187
        DB  188,189,190,191,192,193,194,195,196,196,197,198,198,199,199,199
        DB  200,200,200,200
ANGLE   DW  0                        ;定义角度变量,初值为0
TEMP    DW  0                        ;定义点的正弦函数值,初值为0
DSEG    ENDS
SSEG    SEGMENT    PARA STACK 'STACK'
        DB  100 DUP(?)
SSEG    ENDS
CSEG    SEGMENT
    ASSUME  CS:CSEG, DS:DSEG, SS:SSEG
MAIN    PROC       FAR
START:  PUSH       DS                ;保护参数
        PUSH       AX
        PUSH       BX
        MOV        AX, DSEG
        MOV        DS, AX
        MOV        AX, SSEG
        MOV        SS, AX
                                     ;查表确定正弦波函数值,逐点绘制正弦波
        SETSCREEN                    ;屏幕置为640×480的彩色图形方式
AGAIN:  LEA        BX, SINE          ;表的偏移地址送 BX
        MOV        AX, ANGLE         ;角度值送 AX
        CMP        AX, 180           ;AX与180比较
        JLE        QUAD1             ;不大于则角度在第Ⅰ或第Ⅱ象限
        SUB        AX, 180           ;若大于则调整角度
QUAD1:  CMP        AX, 90            ;AX与90比较
        JLE        QUAD2             ;不大于则角度在第Ⅰ象限
        NEG        AX                ;否则角度在第Ⅱ象限
        ADD        AX, 180           ;调整角度为180
```

```
QUAD2:  ADD     BX, AX          ;形成查表偏移量
        MOV     AL, SINE[BX]    ;函数值送 AL
        PUSH    AX
        MOV     AH,0
        CMP     ANGLE, 180      ;角度与 180 比较
        JGE     BIGDIS          ;若大于则转 BIGDIS
        NEG     AL              ;否则在第Ⅰ或第Ⅱ象限
        SUB     AL, 240         ;调整显示点的纵坐标为 AL－240
        JMP     READY
BIGDIS: ADD     AX, 240         ;调整显示点的纵坐标为 AL＋240
READY:  MOV     TEMP, AX        ;保存到 TEMP
        POP     AX
        WRITEDOT                ;调用画点宏操作
        ADD     ANGLE, 1        ;角度值加 1
        CMP     ANGLE, 360      ;角度值与 360 比较
        JLE     AGAIN           ;不超过则继续画
        MOV     AH, 07          ;若有键按下则继续执行,否则等待按键输入
        INT     21H
        MOV     AH, 0           ;设置屏幕参数
        MOV     AL, 3           ;设置 80×25 彩色文本方式
        INT     10H
        POP     BX              ;恢复参数
        POP     AX
        POP     DS
        RET                     ;返回
MAIN    ENDP
CSEG    ENDS
        END     START
```

3.5.6　多模块程序设计简介

多模块程序设计也称模块化程序设计。各个源程序模块可单独汇编而产生相应的目标模块，由连接程序将各目标模块连接构成一个完整的可执行程序。

采用模块化程序设计方法要注意合理划分模块，明确各模块的入口参数、出口参数及模块间的通信方式。

1. 多模块程序连接

LINK 程序在进行多模块程序连接时，是按 SEGMENT 语句中提供的组合类型和类别名进行连接的。下面举例说明多模块程序的连接。

若某程序有两个模块，它们的结构如下：

模块1

```
DSEG    SEGMENT COMMON
        DW 50 DUP(?)
DSEG    ENDS
```

```
    CODE            SEGMENT PUBLIC
                    ...
    CODE            ENDS
模块 2
    DSEG            SEGMENT COMMON
                    DW 40 DUP(?)
    DSEG            ENDS
    CODE            SEGMENT PUBLIC
                    ...
    CODE            ENDS
```

连接后各模块所占内存情况如图 3.34 所示。两个数据段模块都为 COMMON 组合类型，LINK 后组合成一个互相覆盖的段，数据区长度取两者长者，即为 50 个字长度。两个代码段模块都为 PUBLIC 组合类型，LINK 后组合成一个互不覆盖、相邻连接的段，连接顺序由 LINK 确定，组合后形成的代码长度等于两者之和。该例中两个模块的数据段和代码段都没有给出“类别名”，这时，LINK 程序视它们为无类别名，并予以连接。当一个模块有“类别名”，另一个模块没有“类别名”时，不能连接成一个段。

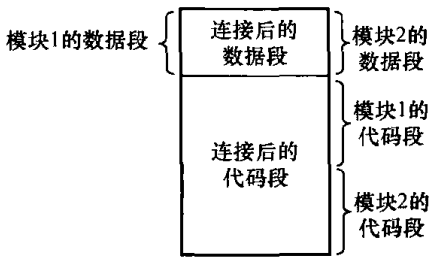


图 3.34 组合类型为 COMMON 和 PUBLLC 时的连接结果

2. 模块间标识符的交叉访问

模块间标识符的交叉访问是指一个模块要引用在另一个模块中定义的标识符（如变量、标号等），因而一个源模块就有供本模块使用的局部标识符和可供本模块及其他模块共用的全局标识符（或称外部标识符）两种。MASM 汇编程序为此提供了 PUBLIC 和 EXTRN 两条伪指令。要注意的是，由 EXTRN 语句定义的标识符（全局变量）必须在其他模块的 PUBLIC 语句中也声明，否则认为出错；另外某模块的 PUBLIC 语句定义某变量为全局变量，但其他模块的 EXTRN 语句未定义，这是允许的；EXTRN 语句中提出的变量名及类型应与其他模块定义的一致。

关于模块化程序设计步骤和应该遵循的一些原则，请读者参考有关软件工程的书籍。

3.5.7 汇编语言与高级语言 C 的混合编程简介

在用高级语言（如 C、PASCAL、FORTRON）进行应用程序的开发研制过程中，为了提高功能模块执行速度、缩短代码长度、硬件操作等，常会用到汇编语言与高级语言的混合编程技术。

高级语言与汇编语言混编时主要考虑两个问题：调用程序与被调用程序之间的程序控制权和参数传递。被调用的过程或函数必须声明为外部类型，如汇编中声明为 PUBLIC 型；参数传递采用堆栈，调用程序（高级语言）将初始数据依次压入堆栈，被调用程序再从堆栈中逐个取出入口参数（直接数据或数据地址）。

下面给出的例子中参数传递均采用堆栈方式。高级语言程序调用函数时，将参数表中的参数按顺序压入堆栈，再压入返回地址，然后将控制权交给过程或函数。下面以 BORLAND C（简写为 BC）语言与汇编语言为例说明。

BC 提供了与汇编语言程序的接口和在 C 程序中直接插入指令代码等功能。BC 支持以

远程方式和近程方式使用汇编语言书写的各种函数。BC对汇编语言程序的书写格式及大小写字母有严格要求。建议使用BC的非集成化编译程序BCC来建立一个汇编语言程序格式框架。例如，要编写BCADD16.ASM的汇编语言程序，先用BCADD16.C程序编写一个C语言的空函数，其形式为：

```
BCADD16()
{
    }
```

然后使用BCC的“－S”选择项编译BCADD16.C生成BCADD16.ASM汇编语言格式的程序。其编译命令行为：

```
BCC-S BCADD16
```

它生成一个框架式的汇编语言程序格式如下：

```
                NAME        BCADD16
_TEXT           SEGMENT     BYTE  PUBLIC,CODE,
DGROUP          GROUP       _DATA,_BSS
                ASSUME      CS:_TEXT,DS:GROUP,SS:DGROUP
_TEXT           ENDS
_DATA           SEGMENT     WORD PUBLIC 'DATA'
_D@             LABEL       BYTE
_DATA           ENDS
_BSS            SEGMENT     WORD PUBLIC 'BSS'
_B@             LABEL       BYTE
_BSS            END

_TEXT           SEGMENT     BYTE PUBLIC 'CODE'
;LINE 2
_BCADD16        PROC        NEAR
;LINE 3
@1:
                            ;子程序汇编语句
                RET
_BCADD16        ENDP
_TEXT           ENDS

_DATA           SEGMENT     WORD PUBLIC 'DATA'
_S@             LABEL       BYTE
_DATA           ENDS

_TEXT           SEGMENT     BYTE PUBLIC 'CODE'
                PUBLIC      _BCADD16
_TEXT           ENDS
                END
```

再在这个汇编程序框架中的@1 处填上 BCADD16 所要完成的汇编代码，使用这种方式建立的汇编接口程序可保证与 BC 有正确的接口。

为考虑与其他高级语言调用汇编接口的要求，未使用上述汇编程序框架。给出 BC 以近程方式调用的汇编接口程序的完整代码如下：

```
                PUBLIC      _BCADD16
                EXTRN       B2BCDW:FAR, BCDW2B:FAR, ADDPD16:FAR

_TEXT           SEGMENT     BYTE PUBLIC'CODE'
                ASSUME      CS:_TEXT
_BCADD16        PROC        NEAR
                PUSH        BP
                MOV         BP, SP
                MOV         AX, [BP+4]
                CALL        B2BCDW
                MOV         BX, AX
                MOV         AX, [BP+6]
                CALL        B2BCDW
                CALL        ADDPD16
                CALL        BCDW2B
                MOV         DI, [BP+8]
                MOV         [DI], AX
                POP         BP
                RET
_BCADD16        ENDP

_TEXT           ENDS
                END
```

这个汇编程序被声明为 PUBLIC，且具有近程属性。它从堆栈的［BP+4］与［BP+6］（若为远程属性则为［BP+6］与［BP+8］）的两个字单元中直接取得两个十六位的二进制数，通过接口程序的另一汇编子程序 B2BCDW 转换成压缩型 BCD 数；并由 ADDPD16 完成两个压缩型 BCD 数的加法操作，其结果（由子程序 BCDW2B 将压缩型 BCD 数转换成二进制数）通过 DI 寄存器返回给 BC 主程序。返回变量的地址为［BP+8］（若为远程属性则为［BP+10］）开始的两个字节。在进入该汇编接口程序之后，将 BP 压入堆栈，以便通过 BP 对堆栈中的操作数进行寻址。在离开汇编接口程序之前，必须恢复 BP 的内容，并执行一条无参数的段内（或段间）返回指令。堆栈中的参数由调用程序自行废弃，无须汇编子程序完成，故 RET 指令不带参数。

该程序在 BC 编译、连接主程序之前，必须汇编成 .OBJ 文件，供 BC 连接时使用。

BC 的主程序（名为 BCNEAR.C）清单为：

```
# include<stdio.h>
int r;
```

```
main( )
{extern bcadd16(int i, int j, int * r)
int I, j;
printf("enter two number:");
scanf(" % d % d", &I, &j );
bcadd16(i, j, &r);
printf("The sum is % d   n", r );
}
```

请注意，汇编接口程序中 BCADD16 是通过外部函数调用的。C 语言程序在调用时，将 i，j 和 r 三个参数从右向左依次压入堆栈，最后压入返回地址。在 BC 主程序中被引用的外部函数必须使用函数原型，即函数参数表的每一个参数必须有明确的类型说明。

现在介绍一下在 BC 集成开发环境下如何编译、连接、执行上述 BC 主程序与汇编子程序。首先在 BC 的 EDIT 状态下，建立一个工程文件 MYBC. PRJ，它的内容为：

```
BCNEAR. C
PATH:BCADD. OBJ                ;BCADD 为上述汇编子程序名称
PATH:ASMTOOL. LIB
```

其中，PATH 为文件路径。若不明确指明，则 BC 所要连接的相应文件在当前驱动器的当前目录下。

由于 BC 对大小写字母非常敏感，它在连接多个文件时将严格检查标识符的大小写。而 MASM 汇编生成的 BCADD. OBJ 文件均为大写字母。故须把 LINK 选择项中的 CASE-SENSITIVE LINK（大小写敏感连接）开关设置成关闭（OFF）状态。当所需参数全部设置完成后，使用选择项菜单中的 STORE OPETIONS 将设置的参数保存到配置文件 BC CONFIG. TC 中。ASMTOOL. LIB 为本书的工具库，从中取出 B2BCDW、ADDPD16 和 BCDW2B 三个子程序供 LINK 程序使用。然后按 F9（MARK）键，使 BC 对工程文件 MY-BC. PRJ 进行编译、连接生成 MYBC. EXE 文件。最后可按 ALT-R 或退出 BC 到 DOS 环境下，键入 MYBC 即可执行。

习　　题

3.1　请解释名词：操作码、操作数、立即数、寄存器操作数、存储器操作数。

3.2　什么叫寻址方式？8086 指令系统有哪几种寻址方式？

3.3　指出下列指令中源操作数的寻址方式：

```
(1) MOV SI,200
(2) MOV  AL,[2000H]
(3) MOV CX,DATA[SI]
(4) ADD AX,[BX + DI]
(5) AND AX,BX
(6) MOV [SI],AX
(7) MOV AX,DATA[BP + SI]
```

(8) PUSHF

(9) MOV AX,ES:[BX]

(10) JMP FAR PTR PROCS_1

3.4 设 DS=1000H，ES=2000H，BX=2865H，SI=0120H，偏移量 D=47A8H，试问下列各指令中源操作数所在位置，若有物理地址请计算出其物理地址值。

(1) MOV AL,D

(2) MOV AX,BX

(3) MOV AL,[BX+D]

(4) MOV AL,[BX+SI+D]

(5) MOV BX,[BX+SI+2]

(6) MOV [BX+5],AX

(7) INC BYTE PTR[SI+3]

(8) MOV DL,ES:[BX+SI]

(9) MOV AX,2010H

(10) MOV AX,DS:[2010H]

3.5 现有 DS=2000H，BX=0100H，SI=0002H，(20100H)=12H，(20101H)=34H，(20102H)=56H，(20103H)=78H，(21200H)=2AH，(21201H)=4CH，(21202H)=B7H，(21203H)=65H，试说明下列指令执行后，AX 寄存器中的内容。

(1) MOV AX,1200H

(2) MOV AX,BX

(3) MOV AX,[1200H]

(4) MOV AX,[BX]

(5) MOV AX,1100H[BX]

(6) MOV AX,[BX+SI]

(7) MOV AX,[1100H+BX+SI]

3.6 已知 AX=75A4H，CF=1，分别写出下列指令执行后 AX、CF、SF、ZF、OF 的值。

(1) ADD AX,08FFH　　(2) INC AX

(3) SUB AX,4455H　　(4) AND AX,0FFFH

(5) OR AX,0101H　　(6) SAR AX,1

(7) ROR AX,1　　(8) ADC AX,5

3.7 假如 AL=20H，BL=10H，当执行 CMP AL，BL 后，问：

(1) 若 AL，BL 中内容是两个无符号数，比较结果如何？影响哪几个标志位？

(2) 若 AL，BL 中内容是两个有符号数，结果又如何？影响哪几个标志位？

3.8 已知 AX=2040H，DX=380H，端口 (PORT)=(80H)=1FH，(PORT+1)=45H，指出执行下列指令后，结果是什么？

(1) OUT DX,AL　　(2) OUT DX,AX

(3) IN AL,PORT　　(4) IN AX,80H

3.9 假设下列程序执行前 SS=8000H，SP=2000H，AX=7A6CH，DX=3158H。执

行下列程序段，画出每条指令执行后，寄存器 AX，BX，CX，DX 的内容和堆栈存储的内容的变化情况，执行完毕后，SP＝？

```
PUSH    AX
PUSH    DX
POP     BX
POP     CX
```

3.10　编程序段分别完成如下功能：

(1) AX 寄存器低 4 位清零。

(2) BX 寄存器低 4 位置“1”。

(3) CX 寄存器低 4 位变反。

(4) 测试 DL 寄存器位 3，位 6 是否同时为 0，若是，将 0 送 DL；否则 1 送 DH。

3.11　写出三种不同类型的指令将寄存器 BX 清零。

3.12　已知从 DS：2200H，ES：3200H 单元起分别存放 20 个 ASCII 的字符。找出这两个字符串中第一个不同字符的位置（段内偏地址），并放入 DS：22A0H 开始的连续两个单元中。请设计完成此任务的程序段。

(1) 用比较指令（CMP）实现。

(2) 用数据串比较指令（CMPSB）实现。

3.13　读下面程序段，请问在什么情况下，本段程序的执行结果是 AH＝0？

```
BEGIN:    IN   AL,5FH
          TEST  AL,80H
          JZ   BRCH1
          MOV  AH,0
          JMP  STOP
BRCH1:    MOV  AH,0FFH
STOP:     HLT
```

3.14　读程序：

```
START:    IN   AL,20H
          MOV  BL,AL
          IN   AL,30H
          MOV  CL,AL
          MOV  AX,0
ADLOP:    ADD  AL,BL
          ADC  AH,0
          DEC  CL
          JNZ  ADLOP
          HLT
```

请问：(1) 本程序实现什么功能？

(2) 结果在哪里？

3.15　读程序段，回答问题。

```
MOV     AL,05H
XOR     AH,AH
ADD     AX,AX
MOV     BX,AX
MOV     CX,2
SHL     BX,CL
ADD     AX,BX
```

该程序段的功能是什么？执行程序段后 AX＝?，你是否可用更简单的程序段完成此功能，请写出这段程序。

3.16 阅读下列程序：

```
NEXT:   MOV AL,[SI]
        MOV ES:[DI],AL
        INC SI
        INC DI
        LOOP NEXT
```

写出用串指令完成上述功能的程序段。

3.17 阅读程序，并回答问题。

```
START:  IN AL,20H
        MOV BL,AL
        IN AL,30H
        MOV CL,AL
        MOV AX,0
        MOV CH,AL
L1:     ADD AL,BL
        ADC AH,0
        LOOP  L1
        HLT
```

问：(1) 本程序实现什么功能?

(2) 结果在哪里?

(3) 用乘法指令 MUL BL 编程并使结果不变。(假设 20H、30H 端口输入的数据均为无符号数)。

3.18 假设寄存器 AX＝1234H，DX＝0A000H，阅读下列程序段：

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

上述程序执行后 AX＝?，DX＝?，程序功能是什么?

3.19　比较 AX，BX，CX 中带符号数的大小，将最大的数放在 AX 中，请编程。

3.20　AX 和 BX 中的两个带符号数相加，如果没有溢出则转去执行 EXE，请写出完成这个工作的程序段。

3.21　编写程序段，在数据区 0000H：2000H 开始 100 字节范围内，查找字符'A'，若找到，则将偏移地址送入 DX，没有找到，则结束。

3.22　什么叫汇编？汇编程序的功能有哪些？

3.23　一个汇编源程序应该由哪些逻辑段组成？各段如何定义？各段的作用和使用注意事项是什么？

3.24　汇编语言源程序的语句类型有哪几种？各自的作用是什么？

3.25　语句标号和变量应具备的三种属性是什么？各属性的作用是什么？

3.26　指令性语句和指示性语句的本质区别是什么？

3.27　有数据段为：

```
DATA  SEGMENT
   ORG    200H
   TAB1   DB          16,-3,5,'ABCD'
   TAB2   DW          'XY',-2,0,0AH
   ARR1   DW          TAB1
   ARR2   DD          TAB2
DATA  ENDS
```

汇编后，设数据段从 200H 开始的单元存放，请画出存放示意图。

3.28　已知数据段 DATA 从存储器实际地址 02000H 开始，作如下定义：

```
DATA  SEGMENT
     VAR1    DB    2 DUP(0,1,?)
     VAR2    DW    50 DUP(?)
     VAR3    DB    10 DUP(0,1,2 DUP(4),5)
DATA ENDS
```

求出 3 个变量 SEG、OFFSET、TYPE、LENGTH 和 SIZE 的属性值。

3.29　已知数据区定义了下列语句，采用图示说明变量在内存单元的分配情况以及数据的预置情况。

```
DATA  SEGMENT
   A1  DB  20H,52H,2 DUP(0,?)
   A2  DB  2 DUP(2,3 DUP(1,2),0,8)
   A3  DB  'GOOD!'
   A4  DW  1020H,3050H
   A5  DD  A3
DATA  ENDS
```

3.30　采用示意图来说明下列变量在内存单元的分配以及数据的预置。

```
DATA    SEGMENT
        ORG 4
```

```
        VAR1   DW 9
        VAR2   DW 2 DUP(0)
        CONT   EQU 2
        VAR3   DB  CONT DUP(?,8)
        VAR4   DB 2 DUP(?,CONT DUP(0),'AB')
DATA    ENDS
```

3.31 已知 3 个变量的数据定义如下所示，分析给定的指令是否正确，有错误时加以改正。

```
DATA   SEGMENT
   VAR1   DB ?
   VAR2   DB 10
   VAR3   EQU 100
DATA   ENDS
(1)  MOV      VAR1,AX
(2)  MOV      VAR3,AX
(3)  MOV      BX,VAR1
     MOV      [BX],10
(4)  CMP      VAR1,VAR2
(5)  VAR3     EQU  20
```

3.32 执行下列指令后，AX 寄存器中的内容是什么？

```
TABLE DW 10,20,30,40,50
ENTRY DW 3
  .
MOV BX,OFFSET TABLE
ADD BX,ENTRY
MOV AX,[BX]
```

3.33 编程实现求 $S=(X^2+Y^2)/Z$ 的值，并将结果放入 RESULT 单元。

3.34 在数据区中，以 TABLE 开始连续存放 0～6 的立方值（称为立方表），设任给一数 x（$0\leqslant x\leqslant 6$），x 在 TAB1 单元，查表求 x 的立方值，并把结果存入 TAB2 单元。

3.35 以 BUF 为首址的存储区，存放有若干个有符号字数据，试编写一程序求出其中负数的平均值（负数之和>−32768），并存放在 MEA2 单元中。

3.36 编写程序，计算下面函数的值。

$$s=\begin{cases}2x & (x<0)\\ 3x & (0\leqslant x\leqslant 10)\\ 4x & x>10\end{cases}$$

3.37 设 AX 寄存器中有一个 16 位二进制数，编一程序统计其中“1”的个数，个数送 CX 中。

3.38 假设学生某门课的成绩存放在数据区中，请编一程序，统计该成绩中小于 60 分的人数，60～90 分的人数，大于 90 分的人数，并显示在屏幕上。

3.39 现有两个多字节压缩 BCD 码数 9876543219H 和 1234567891H，它们分别按低位

字节在前高位字节在后存放在变量 A1 和 A2 中，求它们的和与差，并将结果放在变量 SUM 和 DEF 中。

3.40　设变量 K 中存放了由 100 个有符号整数组成的数组，编写程序段找出其中最大的一个，放到 AX 中。

3.41　编写程序，比较两个字符串 STRING1 和 STRING2 所含字符是否完全相同，若相同则显示“MATCH”，若不同则显示“NO MATCH”。

3.42　在 A1 单元开始定义了一长度为 *N* 的字符串，找出其中所有的小写字母并存放到 A2 单元开始的存储区中。统计出小写字母的个数，存放到 SL 单元中。请编写一完整的源程序。数据段如下：

```
DATA SEGMENT
   A1   DB   'AB1243ADCBBCDED94R3'
   N    EQU   $-A1
   A2   DB N DUP(?)
   SL   DB   ?
DATA   ENDS
```

3.43　在数据段中有一个字节数组，编程统计其中正数的个数，放入 A 单元，负数的个数放入 B 单元。

3.44　编写完整程序，用十六进制输出一个按键的 ASCII 值，要求输出形式是：A=41H。

3.45　编写完整程序，判断一个按键是不是回车键。若是，输出“Yes!”，否则输出“No!”。

3.46　编一子程序利用 XLAT 指令把十六进制数转换成 ASCII 码。假设 ASCII 码存放在以 DAT1 为首地址的数据区中，对应的十六进制数放在以 DAT2 为首地址的数据区中，转换结果送以 DAT3 为首地址的数据区中。

3.47　编写排序子程序 SORT，以 DS、SI 和 CX 作为入口参数，把 DS：SI 为起始地址的一个带符号的字型数组进行由小到大排序，参数 CX 中存放的是数组中元素个数。

3.48　编写递归子程序，求两个正整数的最大公约数。

第4章 存储器系统

4.1 存储器概述

存储器是计算机的重要组成部分之一，用来存放计算机的程序和数据，CPU通过总线直接访问内存。存储器可分为内部存储器（简称内存）和外部存储器（简称外存），内存是主存储器，一般由MOS型半导体存储器组成。

在高档微机系统中，存储器系统采用多层结构，内存又分为高速缓冲存储器（Cache）和主存储器。高速缓冲存储器（Cache）是计算机系统中的一个高速但容量小的存储器，在中高档微机中用来存放CPU正在使用和可能将要使用的局部指令和数据。高速缓冲存储器通常用双极型半导体存储器组成。

内存容量的大小受CPU地址总线条数的限制，对于8086系统，20位地址总线直接寻址的内存空间为1M字节（$2^{20}=1M$）。若是80386系统，地址总线32位，可以直接寻址4G字节（$2^{30}=1G$）。计算机启动时，一般由只读存储器ROM中的引导程序来启动系统，随后从外存中读取系统程序和应用程序，送到随机存储器RAM中，程序运行的结果放在RAM或外部存储器。

外存是辅助存储器，其主要的特点是容量大，也称为海量存储器，但存取速度较慢。另外，外存可以长期保存程序和数据。在微型计算机系统中，常见的外存有软磁盘、硬磁盘、光盘、存储卡、U盘等。但外存要配置专门的驱动设备才能完成对它的读写。

4.1.1 半导体存储器的分类

微型计算机的内存储器通常是由半导体存储器构成。按存取方式来分，半导体存储器分为随机读写存储器（RAM）和只读存储器（ROM）。随机读写存储器是一种易失性存储器，其特点是在使用过程中，信息可以随机写入或读出，使用灵活，但信息不能永久保存，一旦掉电，信息就会自动丢失，常用来存放正在运行的程序和数据；只读存储器ROM是一种非易失性存储器，其特点是信息在一定条件下写入，掉电时信息也不会丢失。在正常使用过程中，ROM存储器中的存储信息只能读出，而不能写入。ROM存储器常用于保存固定不变的程序和数据，如主板上的基本输入/输出系统程序BIOS、打印机中的汉字库、外部设备的驱动程序等，也可作为I/O数据缓冲存储器、堆栈等。

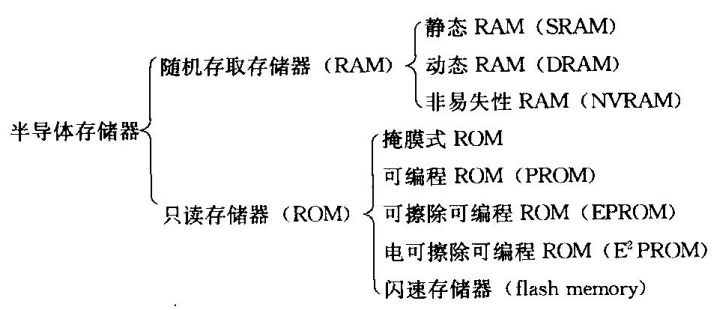


图4.1 半导体存储器的分类

随机读写存储器分为静态存储器（SRAM）和动态存储器（DRAM）；只读存储器又可分为掩膜（或ROM）、可编程ROM（PROM）、可擦除可编程ROM（EPROM）、电可擦除可编程ROM（E^2 PROM）和闪速存储器（FLASH）。半导体存储器的分类如图4.1所示。

1. 随机读写存储器 RAM

随机读写存储器 RAM 可以随机地在任意位置上读/写信息，电源切断后，所存数据全部丢失。通常我们所说的 CPU 内存容量一般指 RAM 存储器的容量。根据存储器芯片内部基本单元电路的结构不同，RAM 又分为：

（1）静态读写存储器 SRAM。静态 RAM 的基本存储电路又称为存储元，是存放一位二进制信息“0”或“1”的电路，通常用双稳态触发器电路来构成。静态 RAM 只要不断电，“0”或“1”的状态能一直保持，除非写入新的数据。同样，对存储元的读出过程也是非破坏性的，读出操作后，所保存的信息不变，故称其为“静态”RAM。SRAM 的主要特点是存取时间短（几十到几百纳秒），外部电路简单，便于使用。常见的 SRAM 芯片容量为1～64KB 之间。

（2）动态读写存储器 DRAM。动态读写存储器 DRAM 的存储元以电容来存储信息，电路简单。但电容具有漏电特性，时间长了存放的信息就会丢失或出现错误。因此需要对这些电容定时充电，这个过程称为“刷新”，即定时地将存储元中的内容读出再写入。由于需要刷新，所以这种 RAM 称为“动态”RAM。DRAM 的存取速度较 SRAM 的存取速度慢，但其最大的特点是集成度非常高，容量大。目前 DRAM 芯片的容量已达几百兆比特，另外 DRAM 功耗低，价格比较便宜。

由于用 MOS 工艺制造的 RAM 集成度高，存取速度能满足各种类型微型机的要求，而且其价格也比较便宜，因此，现在微型计算机中的内存主要由 MOS 型 DRAM 组成。

（3）非易失性读写存储器 NVRAM。NVRAM（Non Volatile RAM）是一种非易失性随机读写存储器。它的存储电路由 SRAM 和 E^2PROM 共同构成，在正常运行时和 SRAM 的功能相同，既可以随时写入，又可以随时读出。但在掉电或电源发生故障的瞬间，它可以立即把 SRAM 中的信息保存到 E^2PROM 中，使信息得到自动保护。NVRAM 多用于掉电保护和保存存储系统中的重要信息。

2. 只读存储器（ROM）

ROM 存储器是将程序和数据固化在芯片中，数据只能读出、不能写入，断电后数据也不会丢失，ROM 中通常存储操作系统的引导程序（BIOS）或用户的固定程序。

根据不同的编程写入方式，ROM 分为以下几种：

（1）掩膜式 ROM。掩膜式 ROM 存储的信息是由生产厂家根据用户的要求，在生产过程中采用掩膜工艺（即光刻图形技术）一次性直接写入的。掩膜 ROM 一旦制成后，其内容不能再改写，因此它只适合于存储永久性保存的程序和数据，常用于批量生产。

（2）可编程 PROM（PROM）。PROM（Programmable ROM）为一次编程 ROM。它的编程逻辑器件靠存储元中熔丝的断开与接通来表示存储的信息：当熔丝被烧断时，表示信息“0”；当熔丝接通时，表示信息“1”。由于存储元的熔丝一旦被烧断就不能恢复，因此 PROM 存储的信息只能写入一次，不能擦除和改写。

（3）可擦除可编程 ROM（EPROM）。EPROM（Erasable Programmable ROM）是一种紫外线可擦除可编程 ROM。写入信息是在专用编程器上实现的，具有多次改写的功能。EPROM 芯片的上方有一个石英玻璃窗口，当需要改写时，将它放在紫外线灯光下照射约15～20min 便可擦除信息，使所有的擦除单元恢复到初始状态，然后编程写入新的内容。由于 EPROM 在紫外线照射下信息易丢失，故在使用时应在玻璃窗口处用不透明的纸封严，以免信息丢失。

(4) 电可擦除可编程 ROM (E²PROM)。E²PROM (Electrically Erasable Programmable ROM) 是一种电可擦除可编程 ROM。它是一种在线 (即不用拔下来) 可擦除可编程只读存储器。它能像 RAM 那样随机地进行改写，又能像 ROM 那样在掉电的情况下使所保存的信息不丢失，即 E²PROM 兼有 RAM 和 ROM 的双重功能特点。E²PROM 的编程改写不需要使用专用编程设备，只需在指定的引脚加上规定的电压即可进行在线擦除和改写，使用起来更加方便灵活。

(5) 闪速存储器 FLASH (Flash Memory)。闪速存储器 (Flash Memory)，简称 FLASH 或闪存。它与 E²PROM 类似，也是一种电可擦写型 ROM。与 E²PROM 的主要区别是：E²PROM是按字节擦写，速度慢；而闪存是按块擦写，速度快，一般读写时间在 65～170ns 之间。

Flash 是近年来发展非常快的一种新型半导体存储器。由于它具有在线电擦写、低功耗、大容量、擦写速度快的特点，同时，还具有与 DRAM 等同的低价位、低成本的优势，因此受到广大用户的青睐。目前，Flash 在微机系统、嵌入式系统和智能仪器仪表等领域得到了广泛应用。

4.1.2 半导体存储器的主要性能指标

存储器的主要性能指标反映了计算机对其性能的要求，常用性能指标如下。

1. *存储容量*

存储容量是指存储器可以存储的二进制信息总量。存储器芯片的存储容量用“存储单元个数×每个存储单元的位数”来表示。例如，SRAM 芯片 6264 的容量为 8K×8bit，即它有 8K 个单元 (1K＝1024)，每个单元存储 8 位 (一个字节) 二进制数据。DRAM 芯片 NMC41257 的容量为 256K×1 bit，即它有 256K 个单元，每个单元存储 1 位二进制数据。各半导体器件生产厂家为用户提供了许多种不同容量的存储器芯片，用户在构成计算机内存系统时，可以根据要求加以选用。但一般计算机的内存容量确定后，建议选用单片容量大的芯片，则可以少用几片，这样不仅使电路连接简单，而且功耗也可以降低。

存储容量的大小在一定程度上影响了计算机对信息的处理能力。目前使用的存储容量达 MB (兆字节)、GB (千兆字节)、TB (兆字节) 或更大的存储空间。存储容量通常以字节 (Byte) 为单位来表示，各层次之间的换算关系为：

$1KB=2^{10}B=1024B$

$1MB=2^{20}B=1024KB$

$1GB=2^{30}B=1024MB$

$1TB=2^{40}B=1024GB$

2. *存取时间和存取周期*

存取时间又称存储器访问时间，即启动一次存储器操作 (读或写) 到完成该操作所需要的时间。CPU 在读写存储器时，其读写时间必须大于存储器芯片的额定存取时间。如果不能满足这一要求，CPU 不能从存储器读到正确的数据。

内存的存取时间通常用 ns (纳秒) 表示。在一般情况下，超高速存储器的存取时间约为 20ns，高速存储器的存取时间约为几十纳秒。例如，SRAM 的存取时间约为 60ns，属于高速存储器，DRAM 的存取时间约为 120～250ns，则属于低速存储器。

存取周期是连续启动两次独立的存储器操作所需的最小时间间隔。存取周期应大于等于存取时间。

3. 可靠性

可靠性指存储器对磁场、热噪声等外界干扰的抵抗能力。微型计算机正常运行，要求存储器系统具有很高的可靠性，而内存的任何错误都足以使计算机不能正常工作。存储器的可靠性直接与构成它的芯片有关，其可靠性用平均无故障时间 MTBF（Mean Time Between Failures）来衡量，MTBF 越长，可靠性越高。目前，所用的存储器无故障连续工作时间可达数百万小时。

4. 功耗

功耗反映存储器件耗电的多少，同时也反映了其发热的程度。功耗越小，存储器件的工作稳定性越好。

功耗分为“操作功耗”和“维持功耗“。当 CPU 读/写存储器芯片数据时，该芯片的功耗称为“操作功耗”，当 CPU 不读/写存储器芯片时，该芯片的消耗称为“维持功耗”。存储器芯片的“操作功耗”极短，大部分时间处于“维持功耗”状态。大多数半导体存储器的维持功耗小于工作功耗。

5. 性能/价格比

常用性能/价格比来衡量存储器经济性能的好坏，是以上指标的综合参数。目前常用芯片每位的价格来衡量存储器芯片的性能。

4.1.3 半导体存储器的基本结构

半导体存储器芯片内部的基本结构框图如图 4.2 所示。图中，虚线框为半导体存储器，由存储体、地址寄存器、地址译码器、读写驱动电路、数据寄存器和读写控制逻辑等组成。通过 n 位地址线、m 位数据线和一些相关的控制线同 CPU 交换信息。n 位地址线用来确定存储单元的地址，m 位数据线用来传送数据信息，而控制线用来协调和控制 CPU 与内存之间的读/写操作。例如，当 CPU 对存储器进行一次读操作时，先将地址码由 CPU 通过地址线送入地址寄存器，然后控制线中的读信号线有效，地址寄存器中的地址码经过地址译码器后选中该地址所对应的存储单元，并通过读/写驱动电路，将选中单元的数据送入数据寄存器，最后 CPU 从数据总线上读入数据。

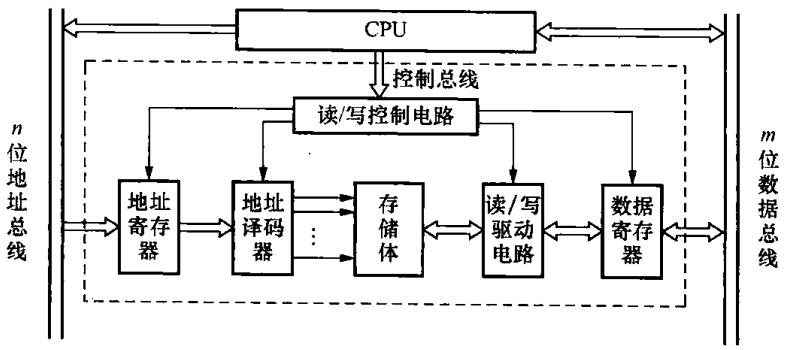


图 4.2 半导体存储器芯片内部的基本结构框图

1. 存储体

存储体是存储器中存储信息的实体，是存储元的集合体，存储元是存储一位二进制“0”或“1”的基本存储电路，若干个存储元构成一个存储单元。通常，8 个存储元构成一个存储单元，即一个存储单元存放 8 位二进制数。为了区分不同的存储单元和读/写操作，每个存储单元分配一个地址（称为存储单元地址），CPU 按地址访问存储单元。为了减少存储器芯片的封装引脚数和简化存储器内部译码器结构，存储体内的存储元总是按照二维矩阵的形式来排列。

存储体内基本存储单元的排列结构通常有字结构和位结构两种方式。例如，某存储器芯片内部有 1K 个基本存储电路（存储元），如果按字节结构方式排列，排列为 128×8，即该

芯片中有 128（2^7）个存储单元，每个单元有 8 个二进制位，需要 7 位地址线，8 位数据线，再加控制线和电源线若干；如果按位结构排列，排列为 1024×1bit，即该芯片中有 1024（2^{10}）个存储单元，每个单元有 1 个二进制位，需要 10 位地址线，1 位数据线，再加控制线和电源线若干。采用字结构方式排列的主要缺点是芯片封装时引线较多；而位结构芯片封装时引线较少，但需要得到一个字节数据时，需要从 8 个芯片上同一地址读出信息组合在一起。在容量较大的存储器中，通常采用位结构方式的存储器芯片。

存储器系统的最大存储容量取决于 CPU 本身提供的地址线位数，这些地址线的每一个编码对应一个存储单元的地址。因此，当 CPU 的地址线为 n 位时，可生成的编码状态有 2^n 个，也就是说 CPU 可直接寻址的存储单元个数为 2^n。若采用字节编址，那么存储器的最大容量为 $2^n \times 8$bit。例如，80486CPU 的地址线为 32 位，可直接寻址的最大存储空间为 2^{32}B=$2^{32} \times 8$bit=4GB。

2. 地址译码器

地址译码器的功能是接受 CPU 的地址信息，并完成译码。通常 CPU 读/写一个存储单元时，首先通过地址总线送出存储单元地址，其中部分高位地址经译码器后产生片选信号选中存储器芯片，部分低位地址送到存储器芯片的地址线，由地址寄存器寄存并通过地址译码器译码选中芯片内的存储单元，最后在读/写信号的控制下将存储单元内容读出或写入。

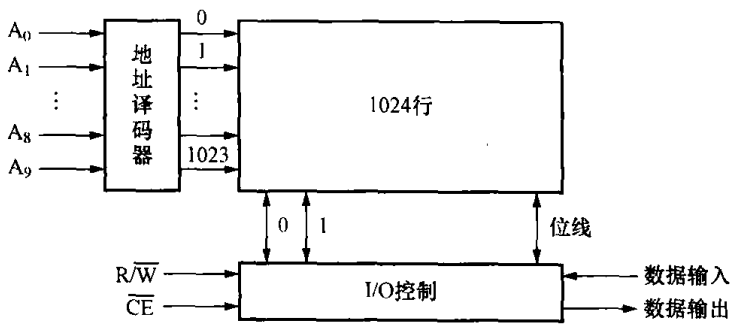


图 4.3　单译码方式

地址译码器通常有两种译码方式：一种是单译码方式，仅有一个译码器。译码器输出的每条译码线对应一个存储单元。如地址位数 n=10，即译码器可以有 2^{10}=1024 种状态，对应有 1024 条译码线（字线），即 1024 个存储单元。单译码方式如图 4.3 所示。

另外一种是双译码方式，是将输入的地址码分成两部分，低位地址部分接入 X 译码器称为行地址译码器，其输出的字选择线称为行选择线，去选中存储矩阵中位于同一行的所有存储单元；高位地址部分接入 Y 译码器称为列译码，其输出的字选择线称为列译码，去选中位于同一列中的存储单元。当行地址、列地址同时有效选中一个存储单元时，才能对该存储单元进行读/写操作。采用双译码电路可以减少译码器输出线的条数。例如，设地址长度 n 仍为 10，将其中的低 5 位输入到 X 地址译码器中，译出 $X_0 \sim X_{31}$ 译码线，分别选择 0～31 行。将高 5 位输入到 Y 地址译码器中译出 $Y_0 \sim Y_{31}$ 译码线，分别选择0～31 列。用 X 译码器和 Y 译码器交叉控制可以选择从存储单元（0，0）至（31，31）共 $2^5 \times 2^5$=1024 个存储单元。X 和 Y 输出的地址线共计 64 条。双译码方式如图 4.4 所示。

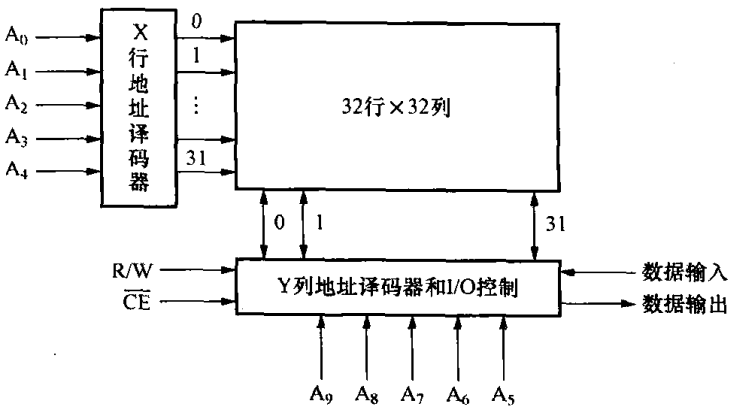


图 4.4　双译码方式

3. 读/写控制电路

读/写控制电路接收CPU传来的控制命令，经过控制电路一系列的处理，产生一组时序信号对存储器进行读/写控制。

4.1.4 存储器的读写操作

CPU通过执行指令对存储单元进行存储器读/写操作。半导体存储器在工作时，有其固有的读/写时序。存储器在与CPU协调工作时，只有CPU提供合适的时序信号，才能正确地完成数据的读出或写入。

1. 存储器读周期

存储器的读周期是指启动一次读操作到启动下一次读操作之间所需要的时间，读出周期 t_{RC} = 读取时间 t_A + 读恢复时间 t_{AR}。

读取时间 t_A：地址有效到数据读出有效之间的时间，MOS器件在50～500ns之间。

片选到数据稳定输出时间 t_{CO}：从$\overline{CS}$片选信号有效到数据输出稳定的时间，一般 $t_A > t_{CO}$。

片选到输出有效 t_{CX}：从$\overline{CS}$片选信号有效到数据输出有效的时间。

读恢复时间 t_{AR}：输出数据有效之后，存储器不能立即输入新的地址来启动下一次读操作，因为存储器在输出数据后要有一定的时间来进行内部操作，这段时间称为恢复时间。

图4.5所示为静态RAM存储器读周期。

从图4.5中A点开始进入读周期，首先CPU通过地址总线送出存储单元地址，在地址信号有效后，不超过 $t_A - t_{CO}$ 的时间段中，片选信号$\overline{CS}$必须有效，存储单元输出的数据送到系统数据总线（图4.5中C点）。

当片选信号$\overline{CS}$有效后，只要地址信号和输出允许信号没有撤销，输出数据一直保持有效。在整个读周期，要求R/$\overline{W}$应保持高电平。

在对存储器芯片和CPU连接时，必须保证下面时间要求：

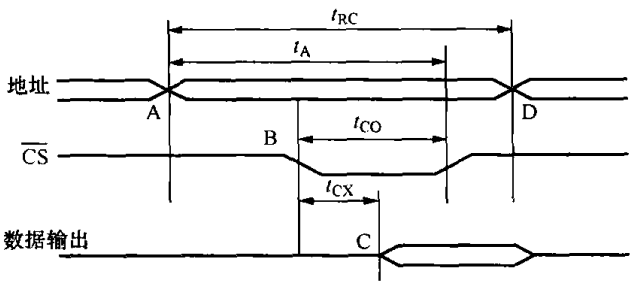


图4.5 存储器读周期

（1）从地址信号有效到CPU要求的数据稳定之间的时间间隔必须大于 t_A；

（2）从片选信号有效到CPU要求的数据稳定之间的时间间隔必须大于 t_{CO}；否则，外部电路必然产生$\overline{WAIT}$信号，迫使CPU插入 T_W 周期来满足上面的时间要求。

2. 存储器写周期

存储器写周期指将数据写入存储器所需要的时间，写周期时间用 t_{WC} 表示。图4.6所示为静态RAM存储器写周期。

地址建立时间 t_{AW}：地址出现到稳定的时间。

写脉冲的宽度 t_W：读/写控制线维持低电平的时间。

数据有效时间 t_{DW}：数据出现在总线上的时间。

数据保持时间 t_{DH}：数据在总线上停留的时间。

写操作恢复时间 t_{WR}：存储器完成内部操作所需时间。

在写周期中地址建立时间 T_{AW} 和写脉冲的宽度 t_W 是要求最严格的两个参数。

从图 4.6 中 A 点开始进入写周期。首先 CPU 通过地址总线送出存储单元地址，经过地址建立时间 T_{AW}（从 A 点到 B 点，使地址稳定在地址总线上）。从 B 点开始，$\overline{CS}$、$\overline{WE}$为低电平有效，经过写脉冲宽度时间 t_W，数据写入存储单元。

为了保证可靠地写入，要求在地址建立时间内，$\overline{WR}$必须为高电平，否则在地址变化期间可能会有误写入操作，使存储单元内容出错。同样在$\overline{WE}$有效到变高电平之后要经过写操作恢复时间，地址信号才能改变。要求 t_W 写脉冲宽度必须大于规定的值，以保证可靠的写入。要写入的数据必须在$\overline{CS}$和$\overline{WE}$有效前已稳定地出现在数据总线上，并在$\overline{CS}$和$\overline{WE}$变高电平之前保持稳定。

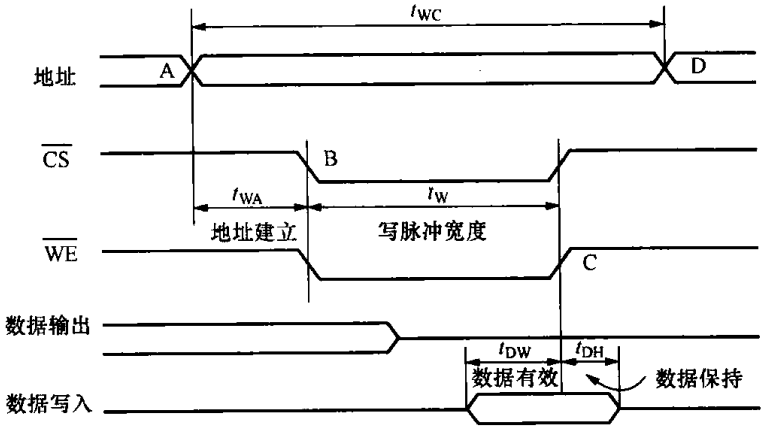


图 4.6 存储器写周期

写周期时间（图 4.6 中 A 点到 D 点）为地址建立时间、写脉冲宽度和写操作宽度和写操作恢复时间三者之和。

上述讨论的存储器读周期和写周期都是指存储器芯片本身能达到的最小时间要求，当将存储器系统作为一个整体考虑时，涉及到系统总线驱动电路和存储器接口电路的延迟，实际的读/写周期要长些。

4.1.5 内存条

从接口形式上看，RAM 早期使用 DIP（Double In-line Package，双列直插式封装）RAM，后多采用 SIMM（Single In-line Memory Module，单在线存储模块）RAM 和当前最流行的 DIMM（Dual In-line Memory Module，双在线存储模块）RAM。Pentium 级主板一般提供 SIMM 和 DIMM 两种 RAM 插槽，而 PII 级主板往往只提供 DIMM 内存插槽。

内存条模块有统一的引线标准，有 30 线与 72 线的 SIMM 模块、新型 168 线与 200 线 DIMM 模块，分别用于不同档次的计算机。DIMM 与早期的 SIMM 有很大区别，其特点是模块增加了总线宽度、可制成体积非常小的数据 32 位模块。DIMM 内存条可单条使用，不同容量的标准条可以混用。

30 线内存条的数据线是 8 位，奔腾系列机具有 64 位数据线，因此若用 30 线内存条，每次至少要用 8 条，故在 Pentium 主板上已不用此形式，而直接使用 72 线和 168 线内存条。72 线内存条的数据线为 32 位，因此在 32 位的主板上，可以单独使用。64 位的主板上，需要两个 72 线内存条同时使用。

常见的 SIMM 内存条容量为 4M、8M 和 16M，最大可支持到 64M，168 线的 DIMM 内存条容量为 8M、16M、32M、64M 和 128M 等。

4.2 随机读写存储器 RAM

随机读写存储器 RAM 是对其中存储单元可以随机访问的存储器芯片。根据存储器芯片内部基本单元电路的结构，RAM 分为静态 RAM 和动态 RAM。

4.2.1 静态读写存储器 SRAM

1. SRAM 的基本结构

SRAM 静态存储器所存储的“0”或“1”信息，只要不断电就能一直保持，除非重新写入新的数据。同样对存储信息的读出过程也是非破坏性的，读出操作后，所保存的信息不变。

SRAM 的内部基本结构如图 4.7 所示，通常由存储单元、地址译码器（分为 X 译码器和 Y 译码器）、控制电路和数据驱动电路等部分组成。

（1）存储单元。存储单元由基本存储电路构成，基本存储电路用来存储一位二进制信息“0”或“1”。通常一块存储器芯片内的基本存储电路按矩阵形式排列。典型的基本存储电路由 6 个 MOS 管（T1～T6）构成，如图 4.8 所示。其工作原理如下：

T1～T4 构成一个双稳态触发器，当 T1 截止时，A 点为高电平，即 A=1，它使 T2 导通，于是 B 点=0，而 B=0 又保证了 T1 截止，因此这时一个稳定状态。与此类似，T1 导通而 T2 截止时，也是一个稳定状态。可见，这个电路具有两个稳定状态，是一个双稳态触发器。于是，我们可以设 T1 截止而 T2 导通的状态为“1”状态，T1 导通而 T2 截止的状态为“0”状态。

由此，我们知道了 SRAM 保存信息的特点是和双稳态触发器的稳态特点密切相关的。但是，仅有两个状态是不够的，还要设法对状态进行控制，于是，就增加了控制管 T5 和 T6，并分别由地址译码信号（X 地址选择线）来控制。

图 4.8 中虚线部分，T7、T8 构成读/写控制电路，由 Y 地址译码线控制，为所有存储单元所共用。

在写入操作时，若写入“1”，则 I/O=1，$\overline{I/O}$=0，X 地址选择线为高电平，使 T5、T6 导通，同时 Y 地址选择线也为高电平，使 T7 和 T8 导通，要写入的数据通过 T5、T6 和 A、B 端相连，使 A=“1”，B=“0”，这样就迫使 T1 截止、T2 导通。当输入的信号和地址选择信号消失后，T5、T6、T7、T8 截止，T1、T2 保持被写入的状态不变。如此只要不掉电，写入的信息“1”就能保持不变。写入“0”的原理与此类似。

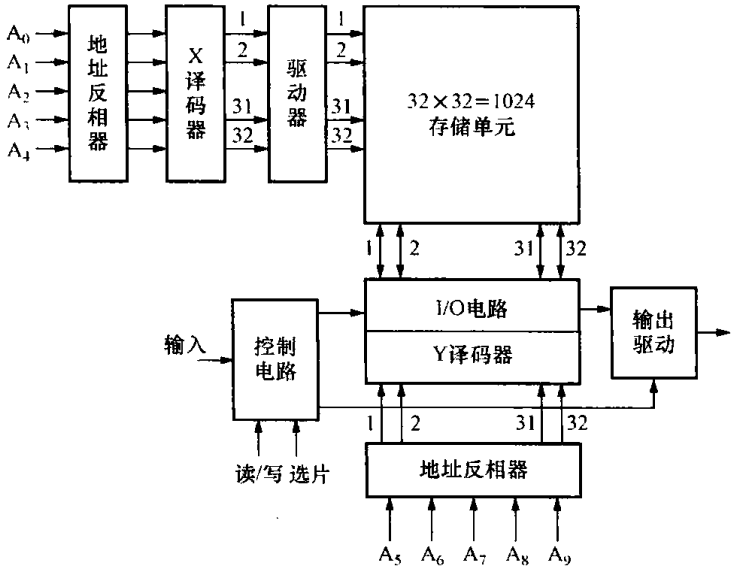


图 4.7 SRAM 的基本结构

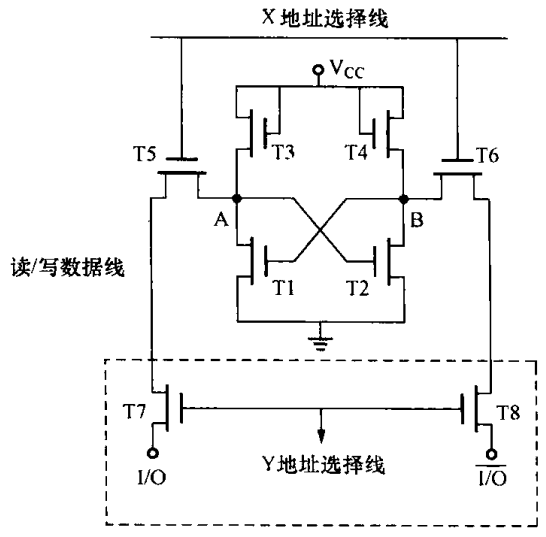


图 4.8 基本存储电路

读操作时，若某个单元被选中（X、Y 地址选择线均为高电平），使 T5、T6、T7、T8 都导通，存储的信息被送到 I/O 端和$\overline{I/O}$线上。I/O 端和$\overline{I/O}$线连接到一个差动读出放大器上，从其电流方向即可判断出所存信息是“1”还是“0”。

（2）地址译码器。地址译码器完成存储单元的选择。通常，接入存储器芯片的地址信号分为行地址和列地址，分别由 X 译码器和 Y 译码器进行译码，选中存储单元。例如，接入存储器芯片的地址信号为 A_0～A_9，通过 X 和 Y 译码器各输出 32 根线，所确定的存储单元数为 32×32=1024。

（3）控制逻辑与数据驱动电路。存储器芯片是否被选中由片选信号$\overline{CS}$控制。当$\overline{CS}$有效，允许对该芯片进行读/写，当读写信号送到存储器芯片的 R/W 端时，存储器中的数据经驱动电路送到数据总线或将数据写入存储器。

使用 SRAM 的优点是访问速度快，读写周期达 20～40ns。SRAM 工作稳定，外部电路简单，但基本存储电路所包含的晶体管的数目比较多，和动态的 RAM 相比，集成度较低；此外，由于双稳态电路中总有一个（或多个）晶体管处于导通状态，因而功耗较大。

2. 典型 SRAM 芯片

典型静态 RAM 芯片有 Intel 公司生产的 6264（8K×8 位）、62128（16K×8 位）、62256（32K×8 位）、62512（64K×8 位）等。

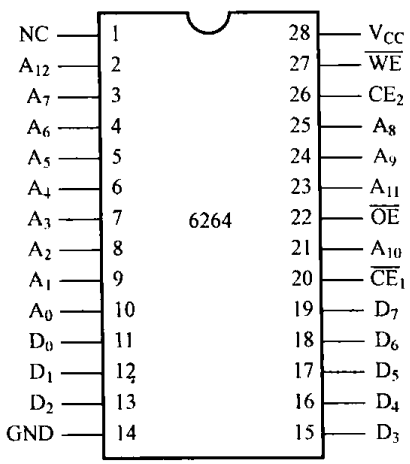


图 4.9　6264 芯片引脚图

图 4.9 给出了 Intel 公司生产的 6264 芯片的引脚图。下面以 6264 芯片为例，说明其引线功能。

A_0～A_{12}：地址信号线，13 位，用于片内寻址 8K 个存储单元。这 13 位地址线在使用时通常与 CPU 地址总线的低位地址相连。

D_0～D_7：数据线，8 位，双向。在使用中，该芯片的数据线与 CPU 数据总线相连。CPU 对存储芯片的读/写数据通过数据总线对指定的单元读出或写入。

$\overline{CE_1}$、CE_2 为片选信号。当两个片选信号有效时（$\overline{CE_1}$=0、CE_2=1 状态时），才选中该芯片，CPU 与该芯片可以进行数据交换。不同类型的芯片，其片选信号的个数也不同。片选信号通常由高位部分地址线通过外部译码电路产生。

$\overline{OE}$为输出允许信号。只有当$\overline{OE}$=0，才允许该芯片将某个单元中的数据送到 D_0～D_7 上。

$\overline{WE}$是写允许信号。当$\overline{WE}$=0 时，允许将数据写入芯片；当$\overline{WE}$=1 时，允许芯片的数据读出。

其他引线：V_{CC}为+5V 电源，GND 是接地端，NC 表示空端。

Intel 6264 的操作方式如表 4.1 所示。

表 4.1　　Intel 6264 的操作方式

操作方式	$\overline{CE_1}$	CE_2	$\overline{WE}$	$\overline{OE}$	D_7～D_0
读出	0	1	1	0	数据输出
保持	1	×	×	×	高阻
写入	0	1	0	1	数据输入

写入操作：当$\overline{CE_1}$和$\overline{WE}$为低电平，且$\overline{OE}$和CE_2为高电平时，数据输入缓冲器打开，数据由数据线$D_7 \sim D_0$写入被选中的存储单元。

读出操作：当$\overline{CE_1}$和$\overline{OE}$为低电平，且$\overline{WE}$和CE_2为高电平时，数据输出缓冲器选通，被选中单元的数据送到数据线$D_7 \sim D_0$上。

保持：当$\overline{CE_1}$为高电平，CE_2为任意时，芯片未被选中，处于保持状态，数据线呈现高阻管隔离状态。

4.2.2 动态随机读写存储器 DRAM

1. 动态 RAM 基本存储单元

动态 RAM 是利用 MOS 管的栅极对其衬底间的分布电容来保存信息，即用电容端电压的高低来表示“1”和“0”信息。由于电容特性，所存储信息在$10^{-3} \sim 10^{-6}$s之后自动消失，因此必须周期性地在内容消失之前进行刷新。所谓刷新，就是定期地将动态存储器中存放的每一位信息读出并重新写入的过程。

动态 RAM 存储元电路主要由 4 管动态 RAM、3 管动态 RAM 或单管动态 RAM 组成，它们各有特点。动态 RAM 与静态 RAM 一样，内部也是将存储元电路按矩阵方式排列。

单管动态存储元电路由一个 MOS 管 T1 和一个电容 C 构成，如图 4.10 所示（虚线部分）。T2 为控制管。写入操作时，X 地址选择线和 Y 地址选择线均为“1”，T1 管导通，写入的信息通过位线（数据线）存入电容 C 中；读出操作时，字选择线为“1”，存储在存储电容 C 上的电荷通过 T1、T2 输出到位线上，根据位线上有无电流即可得知存储的信息是“1”还是“0”。

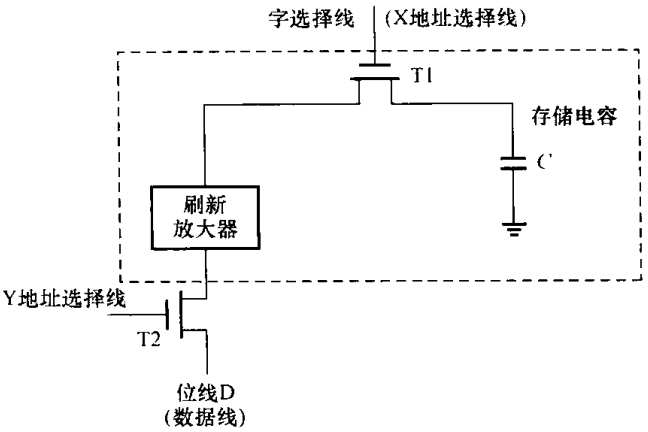


图 4.10 单管动态 RAM 的基本存储电路

DRAM 使用中的一个重要问题就是必须对它所存储的信息定时进行刷新。另外，为了提高集成度，减少引脚的封装数，DRAM 的地址线分成行地址和列地址两部分，因此，在对存储器进行访问时，总是先由行地址选通信号$\overline{RAS}$把行地址送入内部设置的行地址锁存器，再由列地址选通信号$\overline{CAS}$把列地址送入列地址锁存器，并由读/写信号控制数据的读出和写入。

动态 DRAM 的刷新是按行进行的，每刷新一行的时间称为刷新周期，通常为2～8ms。虽然每进行一次读/写操作，实际上也进行了刷新，但读/写是随机的，不能在 2ms 内保证所有的单元有读写操作。因此，要设置专门的刷新控制电路来完成对动态 RAM 的刷新。所以刷新和地址两次打入是 DRAM 芯片的主要特点。

DRAM 集成度高、价格低，被广泛应用微型计算机系统中。构成微机内存的内存条一般是由 DRAM 组成的。

2. 典型 DRAM 芯片

Intel2164 是 64K×1bit 的 DRAM 芯片，外部引脚如图 4.11 所示。

地址信号线$A_0 \sim A_7$，用于行列地址分时输入，用来确定 2164 中的 64K（2^{16}）个单元

地址。

数据线 D_{IN} 和 D_{OUT}，其中 D_{IN} 为数据输入线，当 CPU 写芯片的某一单元时，要写入的数据由 D_{IN} 送到芯片内部。同样，D_{OUT} 是数据输出线，当 CPU 读芯片的某个单元时，数据由此引脚输出。

行地址锁存信号$\overline{RAS}$，利用该信号将行地址锁存在芯片内部的行地址缓冲寄存器中。

列地址锁存信号$\overline{CAS}$，利用该信号将列地址锁存在芯片内部的列地址缓冲寄存器中。

写允许信号$\overline{WE}$，当该信号为低电平时，允许将数据写入。反之，当$\overline{WE}$高电平时，可以从芯片读出数据。

2164DRAM 有下面三种工作方式：

（1）读出数据。当要从 DRAM 芯片读出数据时，CPU 首先将行地址加在 $A_0 \sim A_7$ 上，然后送出$\overline{RAS}$锁存信号，该信号的下降沿将行地址锁存在芯片内部。接着将列地址加到芯片的 $A_0 \sim A_7$ 上，再送$\overline{CAS}$锁存信号，该信号的下降沿将列地址锁存在芯片内部。然后保持$\overline{WE}=1$，则在$\overline{CAS}$有效期间（低电平），数据输出在 D_{OUT} 上并保持。其读出过程如图 4.12 所示。

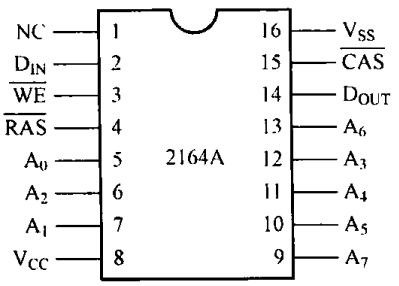


图 4.11 DRAM 芯片外部引脚图

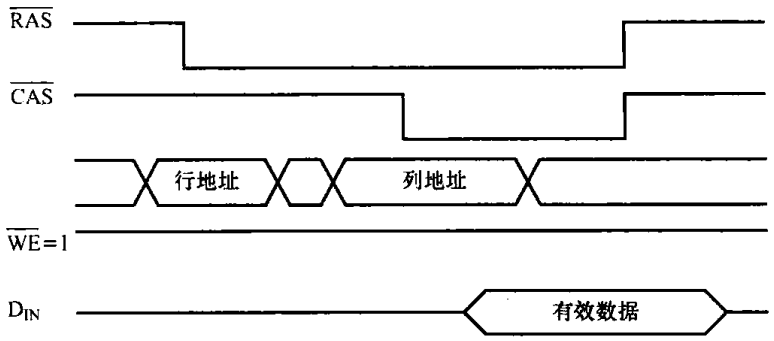


图 4.12 DRAM 2164 的读出过程

（2）写入数据。数据写入 DRAM 芯片时，其过程与读出数据基本相似，行列地址先由$\overline{RAS}$和$\overline{CAS}$信号锁存在芯片内部，然后$\overline{WE}$有效（低电平），随后把要写入的数据从 D_{IN} 端输入，该数据写入选中的存储单元，如图 4.13 所示。

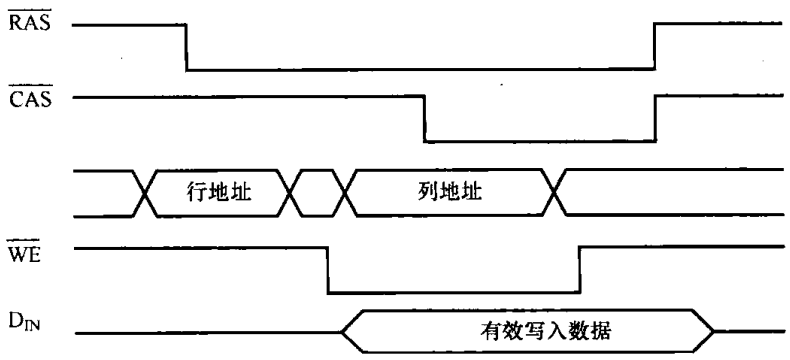


图 4.13 DRAM 2164 的数据写入过程

在图 4.13 中，$\overline{WE}$变为低电平是在$\overline{CAS}$有效之前，通常称为提前写。这能够将输入端 D_{IN} 的数据写入，而 D_{OUT} 保持高阻状态。对于 DRAM 芯片的其他功能，请参考相关资料。

（3）刷新。DRAM 2164 的存储矩阵由 256 行和 256 列构成。存储单元的刷新过程是，每次送出行地址加到芯片上去，利用$\overline{RAS}$有效将行地址锁存于芯片内部的行地址寄存器，这

时$\overline{CAS}$保持无效（高电平），芯片内部的刷新电路就会对所选中行上的各单元的信息进行刷新（对原来为“1”的电容补充电荷，原来为“0”的则保持不变）。每次送出不同的行地址，就可以刷新不同行的存储单元。将行地址循环一遍，就可以刷新整个芯片的所有单元。由于刷新时$\overline{CAS}$无效，故位线上的信息不会送到数据总线上。

DRAM芯片的刷新过程如图4.14所示。图中$\overline{CAS}$保持无效，利用$\overline{RAS}$所存刷新的行地址，进行逐行刷新。在刷新周期中，DRAM不能进行正常的读写操作的，这一点由刷新控制电路予以保证。

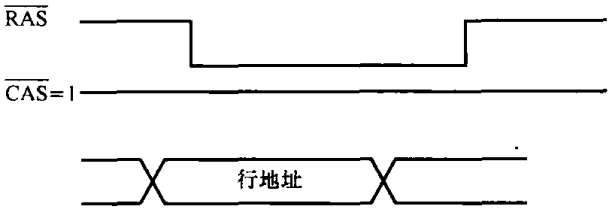


图4.14 DRAM芯片的刷新过程

4.3 只读存储器ROM

微机中常用的ROM芯片主要有掩膜式ROM、可编程ROM（PROM）、可擦除PROM（EPROM）和电可擦除EPROM（E^2PROM）几种。这里主要介绍常用的EPROM、E^2PROM和FLASH。

4.3.1 EPROM

1. EPROM工作原理

EPROM允许用户多次进行擦除和重写，EPROM的存储单元电路原理图，如图4.15所示。

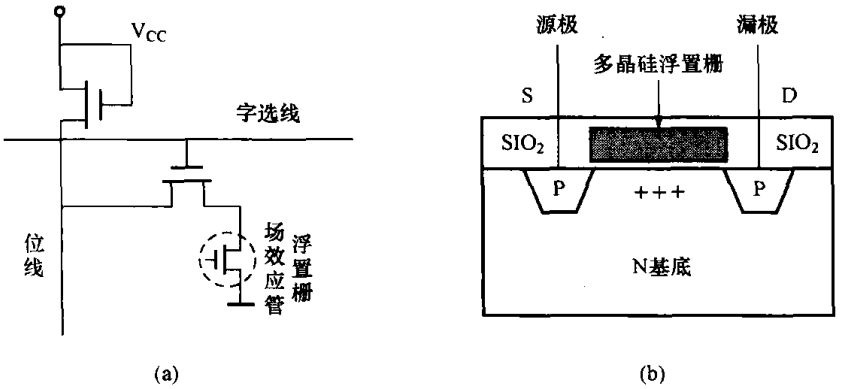


图4.15 EPROM存储单元电路原理图

（a）EPROM的基本存储结构；（b）浮置栅雪崩注入型场效应管结构

一块EPROM在初始状态下，所有的位均为“1”。编程写入时若在漏极和衬底、漏级和源级间加+25V电压，使内部PN结反向击穿，形成较大的电流，部分电荷会注入浮置栅上，当电压移去后由于绝缘层的包围，注入的电荷无法泄漏，相当于管子开启电压提高，控制栅上加上正向电压（+5V）后，管子仍截止，ROM该位存储信息变为“0”。

EPROM芯片上有一个石英窗口，当紫外光源照射到石英窗口上，电路浮置栅上的电荷就会形成光电流泄漏走，使电路恢复起始状态，从而把写入的信息擦去，将“0”变为“1”。这样又可以对EPROM重新编程。但EPROM写入过程很慢，所以它仍然作为只读存储器使用。

一般用紫外光照射 15min 左右才能擦除信息，照射时间视具体器件型号而定，光照时间过长，也会影响器件使用寿命。

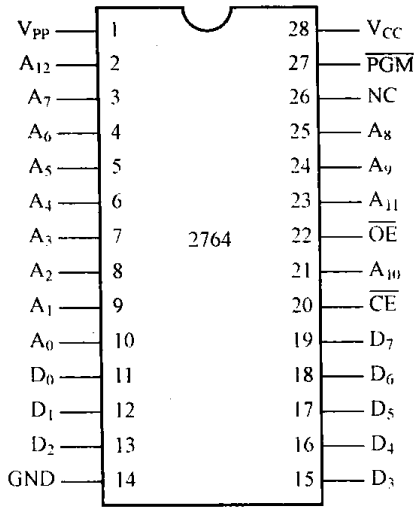


图 4.16 Intel 2764EPROM 的外形和引脚信号

2. 典型 EPROM 芯片

常用的 EPROM 芯片有 Intel 2732（4KB）、2764（8KB）和 27128（16KB）等。下面我们以 Intel 2764 芯片为例说明 EPROM 的应用，2764EPROM 的外形和引脚信号如图 4.16 所示。

Intel 2764 是一块容量为 8K×8bit 的 EPROM 芯片，它的引线与前面介绍的 RAM 芯片 6264 是兼容的。在进行软件调试时，先将程序放在 RAM 中调试和修改。一旦调试成功，再把程序固化在 EPROM 中，并将 EPROM 插在原 RAM 的插座上即可运行。

（1）Intel 2764 引脚功能。A_{12}～A_0 为地址信号输入线。表明该芯片有 8K（2^{13}=8K）个存储单元。

D_7～D_0 为 8 位数据线。表明该芯片的每个存储单元可存放一个字节（8 位二进制数）。在其工作过程中，D_7～D_0 为数据输出线；当对芯片编程时，D_7～D_0 为数据输入线，输入要编程的数据。

$\overline{CE}$为片选信号，输入。当它有效（低电平）时，能选中该芯片，故$\overline{CE}$又称为允许芯片工作信号。

$\overline{OE}$为输出允许信号。当$\overline{OE}$为低电平时，芯片中的数据可由 D_7～D_0 输出。

$\overline{PGM}$为编程脉冲输入信号。当对 EPROM 编程时，由此加入编程脉冲（低电平脉冲）；读操作时$\overline{PGM}$为高电平。

其他引脚：V_{PP}和 V_{CC}为 2 个电源输入端。GND 是接地端，NC 表示空端，没用。

（2）Intel 2764 的工作方式。Intel 2764 可以工作在读出、编程写入和擦除三种方式下。表 4.2 给出了 Intel 2764 的工作方式。

表 4.2　　Intel 2764 的工作方式

操作方式	$\overline{CE}$	$\overline{OE}$	V_{PP}	$\overline{PGM}$	D_7～D_0
读出	0	0	+5V	1	数据输出
保持	1	×	+5V	×	高阻
编程	0	1	+12.5V	0	数据输入
编程校验	0	0	+12.5V	1	数据输出
编程禁止	1	×	+12.5V	×	高阻

1）数据读出。这是 2764 的基本工作方式，用于读出 2764 中存储的内容。此时$\overline{CE}$和$\overline{DE}$为低电平，V_{PP}接+5V，$\overline{PGM}$接高电平，数据线处于输出状态。其工作过程与 RAM 类似。即先把要读出的存储单元地址送到 A_0～A_{12}地址线，然后使$\overline{CE}$=0、$\overline{OE}$=0，就可在芯片的 D_0～D_7 上读出数据。

2）编程写入。将信息写入芯片内。当 V_{CC} 接＋5V，V_{PP} 加上芯片要求的编程电压＋12.5V时，在地址线 A_0～A_{12} 上给出要编程存储单元的地址，然后使$\overline{OE}$=1，$\overline{CE}$=0；并在数据线上给出写入的数据。上述信号稳定后，在$\overline{PGM}$端输入宽度为（50±5）ms 的负脉冲，就可将一个字节的数据写入相应的地址单元中。不断重复这个过程，将要写入的数据逐一写入对应的存储单元中。

编程校验：在编程过程中，如果其他信号状态不变，在一个字节编程完成后，$\overline{OE}$变为低电平，则可以立即对刚写入的数据进行校验，即将同一单元的内容由数据线输出，检验写入的内容是否正确。

上述编程过程称为标准编程方式。目前新型的 EPROM 采用快速编程方式。快速编程与标准编程的工作过程类似，但编程脉冲要窄得多。例如，EPROM27C040 芯片的编程脉冲宽度仅为 100μs。不同厂家、不同型号的 EPROM 芯片，对编程的要求不一定相同，编程的脉冲宽度也不一样，但编程的思想是相同的。

3）擦除方式。EPROM 的一个重要特点是可以擦除重写，而且可以多次擦除。一个新的或擦除干净的 EPROM 芯片，其每个存储单元的内容都是 FFH。要对一个使用过的 EPROM 进行编程，则首先应该将其放到专门的擦除器上进行擦除操作。擦除器利用紫外线光照 EPROM 窗口，一般经过 15～20min 即可擦除干净。擦除完毕后可读取 EPROM 的每个单元，若其内容均为 FFH，就认为擦除干净了。

4.3.2 E^2PROM

E^2PROM（Electrically Erasable PROM，EEPROM）是电擦除可编程只读存储器。其主要特点是在工作过程中能进行读写和擦除，在断电情况下信息不会丢失，兼顾了 RAM 和 ROM 的功能。因此，在智能仪表、控制及开发装置中得到广泛应用。

1. 典型 E^2PROM

E^2PROM 以其制造工艺及芯片容量的不同而有多种型号。E^2PROM 的主要产品有高压编程的 2816/2817（2K×8），低压编程的 2816A/2817A（2K×8）、2864A（8K×8）和 28512（64K×8）等。高压编程 E^2PROM，是指芯片内有编程所需的高压脉冲产生电路，因而不需要外加编程电压和编程脉冲即可工作。有的 E^2PROM 与相同容量的 EPROM 完全兼容，如 2864 与 2764 就完全兼容。

2816A 和 2817A 均属于 5V 电擦除可编程只读存储器，其容量都是 2K×8 位。2816A 与 2817A 的不同之处在于：2816A 的写入时间为 9～15ms，完全由软件延时控制，与硬件电路无关；2817A 利用硬件引脚来检测写操作是否完成。

下面以 2817A 为例，介绍 E^2PROM 芯片的基本特点和工作方式。

（1）2817A 的特性及引脚信号。Intel 2817A 是 28 引脚双列直插式芯片，容量为 2KB，最大读出时间为 250ns，单一＋5V 电源供电，最大工作电流为 150mA，维持电流为 55mA。2817A 有 28 个引脚，双列直插式封装，引脚排列，其引脚信号如图 4.17 所示。

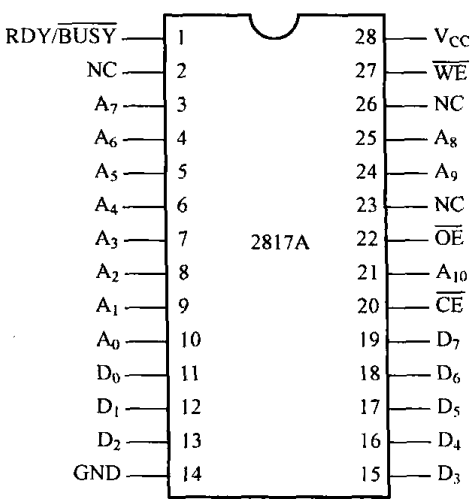


图 4.17 2817A 引脚信号示意图

A_{10}～A_0 为地址信号线，输入，可寻址 2K（2^{11}=2K）存储空间，通常与系统低位地址总线相连。

D_7～D_0 为数据线，8 位，双向，通常与系统数据总线相连。

$\overline{OE}$为读出允许信号，输入，低电平有效。

$\overline{WE}$为写允许信号，输入，低电平有效。

$\overline{CE}$为片选信号，输入，低电平有效。

RDY/$\overline{BUSY}$为闲忙状态指示信号，输出。

V_{CC}为+5V 电源；GND 为接地端。

（2）2817A 的工作方式。2817A 在写入一个字节的指令码（当它用于存储程序时）或数据之前，会自动对要写入的单元进行擦除操作，因而不需要专门对芯片进行字节擦除和整块擦除操作。

2817A 共有三种工作方式，由$\overline{CE}$、$\overline{OE}$、$\overline{WE}$、RDY/$\overline{BUSY}$信号决定，如表 4.3 所示。

表 4.3　　2817A 的 工 作 方 式

操作方式	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	RDY/$\overline{BUSY}$	数据线状态
读	0	0	1	高阻	D_{OUT}
未选中	1	×	×	高阻	高阻
字节编程	0	1	0	0	D_{IN}
字节擦除	在每个字节编程前该单元的内容将自动擦除				

1）读出：从 E^2PROM 读出数据的过程与从 EPROM 及 RAM 读出数据的过程一样。CPU 将地址加在 A_0～A_{10} 经内部地址译码器译码，选中存储单元时，当$\overline{CE}$=0，$\overline{OE}$=0、$\overline{WE}$=1，RDY/$\overline{BUSY}$为高阻状态时，只要满足芯片所要求的读出时序关系，则可从选中的存储单元中将数据读出。

2）未选中：当$\overline{CE}$为高电平，芯片未选中，数据线呈现高阻状态，禁止数据传送，处于保持状态，功耗降低。$\overline{OE}$、$\overline{WE}$为任意态，而 RDY/$\overline{BUSY}$及 D_0～D_7 均呈高阻态。

3）字节写入：当$\overline{CE}$、$\overline{WE}$、$\overline{OE}$为高电平时，进行字节写入操作。写入操作之前自动对要写入的单元进行擦除，而要写入的数据从数据线 D_0～D_7 写入被选中的单元。这时 RDY/$\overline{BUSY}$为低电平，直到写操作完成后，该引脚变为高电平。

2. 闪存（FLASH）E^2PROM

闪存（Flash Memory）是一种非易失性（在断电情况下仍能保持所存储的数据信息）的存储器。闪存是电可擦除只读存储器（E^2PROM）的变种，Flash 擦除时以扇区（又称 Block）为单位进行，擦除速度快，而写入时才以字节为单位进行。因此，闪存不仅具有 E^2PROM的特点，又有很高的存取速度，单片容量更大，功耗也更低。目前闪存主要用来构成存储卡，大量用于便携式计算机、数码相机、MP3 播放器等设备中。

下面以 AT29C010A 芯片为例，简单介绍闪存的工作原理。

（1）AT29C010A 内部结构和引脚。AT29C010A 是 ATMEL 公司推出的 5V 闪速电擦除存储器，是一种并行、高性能、单一+5V 电源供电和+5V 在线擦除的闪存芯片，片内有 1M 位的存储空间，分成 1024 个分区，每一个分区为 128 个字节，以分区为单位进行编

程。AT29C010A的快速读取时间为70ns，快速的分区编程周期为10ms，低功率消耗为50mA有效电流，100mA CMOS维持电流。

AT29C010A的内部结构如图4.18所示。片内有两个8KB的可锁定的自举块，用来存储系统的自举代码和参数表，主块用来存放应用程序和数据，地址和数据信号都具有锁存功能。

AT29C010A的引脚信号定义如图4.19所示，各引脚信号的功能如下：

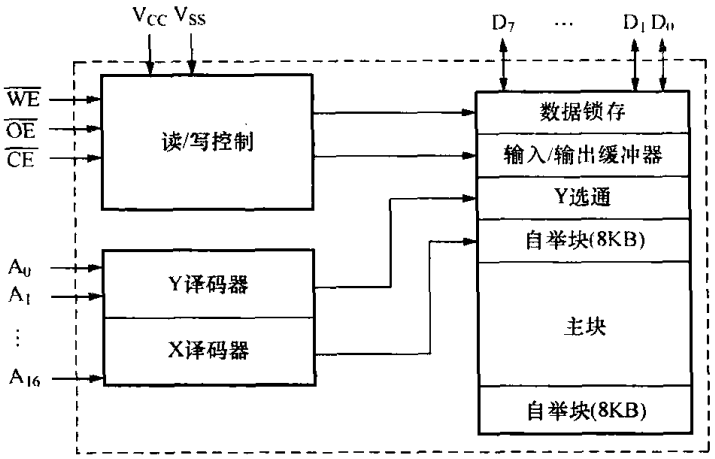


图4.18 AT29C010A的内部结构

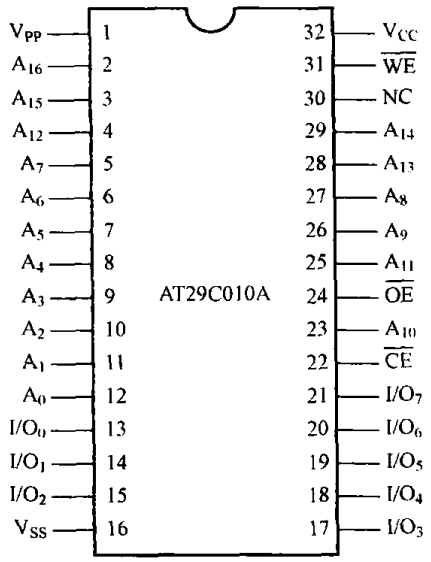


图4.19 AT29C010A引脚信号定义

A_{16}～A_0：地址线，可寻址1M位的存储空间，由高位地址线A_{16}～A_7提供1024个分区的地址，由低位地址线A_6～A_0提供每个分区内128个字节单元的地址。

I/O_7～I/O_0：数据线，双向，三态。

$\overline{OE}$：读出允许信号，输入，低电平有效。

$\overline{WE}$：写允许信号，输入，低电平有效。

$\overline{CE}$：片选信号，输入，低电平有效。

V_{PP}：+5V编程电压。

V_{CC}：+5V工作电压。

V_{SS}：接地端。

（2）操作方式。AT29C010A的读操作与E^2PROM相同，是按字节读出。但在写入（编程）时与E^2PROM不同，是按分区编程，每个分区的容量为128个字节，如果某一分区中的一个数据需要改写，那么这一分区中的所有数据必须重新装入，其操作方式如表4.4所示。

表4.4 AT29C010A操作方式

操作方式	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	I/O_7～I/O_0
读出	0	0	1	数据输出
保持	1	1	×	高阻
编程	0	1	0	数据输入

1）读出。AT29C010A 的读取数据过程类似于普通 E^2PROM，当$\overline{CE}$和$\overline{OE}$为低电平，$\overline{WE}$为高电平时，由 $A_0 \sim A_{16}$寻址的内存单元中的数据会读到 $I/O_0 \sim I/O_7$ 输出管脚；若$\overline{CE}$和$\overline{OE}$为高电平，则 $I/O_0 \sim I/O_7$ 输出管脚为高阻态。

2）字节装载。AT29C010A 的字节装载可用来装入每一分区待编程的 128K 字节数据或用来进行数据保护的软件编码。每一字节的装载是通过$\overline{CE}$或$\overline{WE}$各自为低电平而 OE 为高电平来实现的，数据在$\overline{CE}$或$\overline{OE}$的一个上升沿时锁存。

3）编程。AT29C010A 和普通 E^2PROM 的编程不同，是以分区为单位进行再编程，每个分区大小为 128 个字节，如果某一分区中的一个数据需要改变，那么这一分区中的所有数据必须重新装入。当第一字节数据装入 AT29C010A 之后，接着其余字节将以同一方式依次装入，字节不需按顺序装载，可以以任意方式装载。每一新装载的数据若要被编程则必须有$\overline{WE}$（或$\overline{CE}$）由高到低的跳变，这一跳变需在 150μs 内完成，同时前面字节的$\overline{WE}$（或$\overline{CE}$）由低到高的跳变时间也是 150μs。如果一个由高到低的跳变在最后一个由低到高的 150μs 内没有被检测到，那么字节装载的时间段将结束，此时内部编程时间段开始。$A_7 \sim A_{16}$用于提供分区地址，分区地址只在每一个$\overline{WE}$（或$\overline{CE}$）由高到低的跳变时才有效；而 $A_0 \sim A_6$ 用来提供分区中每一字节的地址。

编程周期开始时，AT29C010A 会自动擦除分区的内容，然后对锁存的数据在定时器的作用下进行编程。当编程将要结束时，在装载最后一个字节数据使 I/O_7 产生装载数据结束信号。通过查询 I/O_7 的状态来判断编程周期是否结束。一旦编程周期结束，就可开始一个新的读或编程操作。

4）软件数据保护。AT29C010A 还具有软件控制数据保护特性，这一特性在 ATMEL 公司出厂时没有开启，用户可根据需要开启或关闭，一旦开启，在未执行关闭指令之前这一特性始终有效；供电情况的改变不会重新设置这一特性，但在供电情况改变时随机的编程周期可能会改变这一特性。若要开启软件数据保护，必须执行三条针对存放有特殊数据的特殊地址单元的程序指令。在开启软件数据保护之后，所需的程序若要出现，也必须执行同样的三条程序指令。软件保护使能的软件算法必须在程序可能执行之前发送到 AT29C010A，所有的软件程序指令要按照分区程序的时序要求进行。在给出软件数据保护的指令代码之后，便可进行字节的装载。

AT29C010A 还具有硬件保护、查询和重复位工作方式，详细内容请参考相关手册。

4.4 存储器系统设计

存储器系统设计是将 CPU 与存储器各个芯片按照给定的地址空间进行连接，保证存储器芯片以及芯片内的存储单元与实际地址一一对应，使 CPU 能通过地址信号选中存储器芯片及存储单元并进行读/写。

本节主要介绍 8086/8088 微机存储器系统物理地址空间的分配和主存储器的逻辑设计方法。

4.4.1 存储器系统设计的主要内容

1. 系统内存配置

系统内存配置是从硬件结构上采用配置基本内存和根据需要扩展内存的方法，并使计算

机的内存容量有一定的伸缩性，以适应不同程序运行的要求。首先应考虑计算机运行环境的需求，其中包括系统软件、应用软件以及数据所占用的存储单元的大小。对于专用的计算机而言，这部分内容是确定的，而通用计算机往往由于运行程序不同，对内存的需求有很大差异。图 4.20 所示为 IBM PC/XT 内存配置示意图。

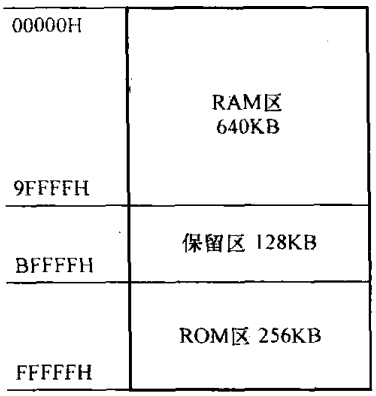


图 4.20 IBM PC/XT 内存配置示意图

配置系统内存容量大小取决于微处理器直接寻址内存的能力。当系统的微处理器确定之后，它的地址总线的条数也就确定了系统内存的最大容量。实际上从硬件上扩大内存的容量，已经不可能满足某些软件的要求。所以，目前的微机系统是采用软、硬件结合的虚拟存储方式，把大容量的外存与内存融为一体，从而使系统廉价地获得一个大的虚拟内存。

在确定了系统的总容量之后，就要对各地址区域的使用进行合理分配，特别是特殊区域的分配。

首先要考虑 ROM 区的设置。为了使系统能在加电后，正常进入指定程序运行，ROM 不仅设有固定的启动地址，而且还有一些特殊用途的固定地址。例如，8086CPU 的中断矢量地址（即中断服务程序入口地址），它们被固定存放在内存的 00000H～003FFH 的地址空间。对于这类存储区，可以设置成 ROM 区，使之能够固定地转入某一地址。当然，也可以把该区域设置为 RAM 区，利用 RAM 的可读/写特性，根据需要编程确定转移地址，从而使程序设计更加灵活。

RAM 区域内的各地址基本上都是由程序操作的，即 RAM 的读/写操作主要受软件的控制。但是系统中也有特殊的 RAM 区域，它的工作不但受软件控制，同时也受硬件的控制。例如，显示缓冲存储区就是一例，对这部分 RAM 区要进行特殊设计，除了和一般的 ROM、RAM 一样与微处理器的有关信号连接外，还要与显示控制器相连。

2. *存储器系统设计应注意的问题*

存储器系统设计的具体内容指 CPU 与存储器的连接，包括地址线、数据线和控制线的连接。在连接时要考虑以下几个问题：

（1）CPU 总线的负载能力。一般来说，CPU 总线的直流负载能力可带一个 TTL 负载，对于 MOS 存储器来说，它的直流负载很小，主要是电容负载，因此在小型系统中，CPU 可以直接和存储器芯片相连，在较大系统中，考虑到 CPU 的驱动能力，应加上数据缓冲器或总线驱动器来驱动存储器负载。

（2）CPU 的时序和存储器存取速度之间的配合。CPU 在取指令和读/写操作数时有它自己固定的时序，应考虑选择何种存储器来与 CPU 时序相配合。若存储器芯片不能满足要求，应考虑插入 T_W 等待周期。

（3）存储器的地址分配和片选。内存分为 ROM 区和 RAM 区，RAM 又分为系统区和用户区。一个存储器系统有多片芯片组成，微机系统中对存储单元的寻址，首先要确定某个存储器芯片的片选信号。存储器芯片的片选信号由 CPU 的高位地址线和控制信号线通过译码后形成，并由此决定该芯片在内存的地址范围。存储器芯片内的存储单元由 CPU 低位地址线来决定。

（4）控制信号的连接。8086 CPU 与存储器交换信息时，提供以下几个控制信号：M/

$\overline{IO}$、$\overline{RD}$、$\overline{WR}$、ALE、READY、$\overline{WAIT}$、DT/$\overline{R}$、$\overline{DEN}$，需要考虑这些信号与存储器芯片要求的控制信号如何连接才能实现所需要的控制功能。

4.4.2　存储器扩展与译码方式

在实际应用中，由于单片存储芯片的容量总是有限的，很难满足实际存储容量的要求，因此需要将若干个存储芯片连接在一起，构成大容量的存储器，这就是存储器的扩展。前已述及，CPU 对于寻址单元的选择首先是进行片选（又称为字选），然后进行片内存储单元的选择（又称为位选）。

1. 存储器的扩展

存储器的扩展通常有位扩展、字扩展，以及字和位同时扩展三种方式。

（1）位数扩展。如果单片存储芯片的数据位数（即字长）不能满足系统字长要求，就需要进行位数的扩展。例如，系统需要容量为 1K×8 位存储器，现有容量为 1K×4 位的 SRAM2114 芯片。因为单片 SRAM2114 的位数不能满足要求，需要 2 片 SRAM2114，构成 1K×8 位存储器。存储器位扩展设计如图 4.21 所示，图中 1 号芯片的数据线接数据总线的低 4 位，2 号芯片的数据线接数据总线的高 4 位。为了读/写 8 位数据，两片的物理地址完全一致，其地址线、片选线与读/写控制线要并联连接。

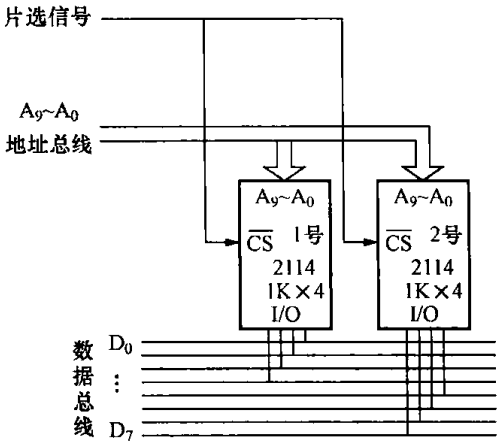


图 4.21　位扩展设计

（用 1K×4 位的芯片构成 1K×8 位的存储器）

（2）字扩展。字扩展就是对地址空间的扩充。如果单片存储芯片的容量不能满足存储系统的空间要求，需要进行字扩展。

字扩展可利用芯片地址串联的方式实现。例如，存储器系统需要配置 64K×8 位的存储器，现有存储器芯片容量为 16K×8 位。为满足系统存储器容量要求，需要 4 片 16K×8 位的芯片来构成 64K×8 位的存储器，如图 4.22 所示。图中 4 个芯片的数据线与数据总线相

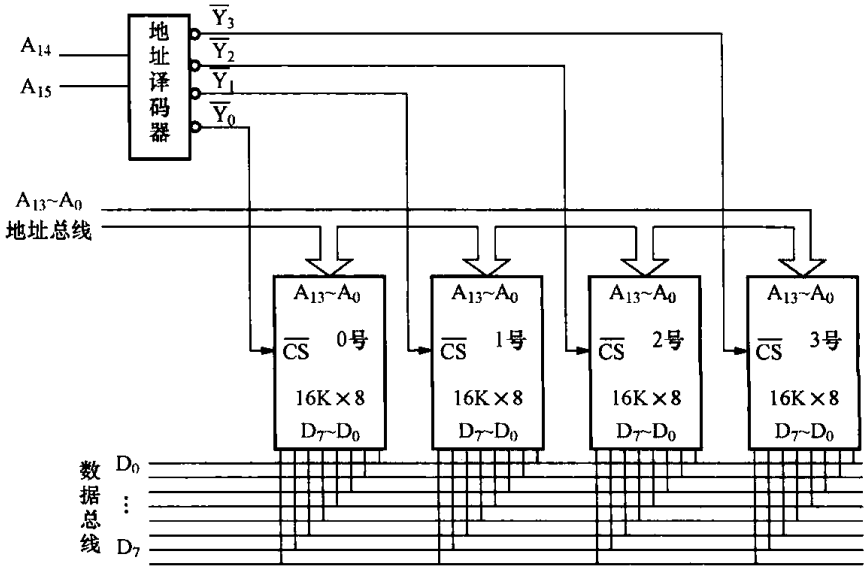


图 4.22　字扩展设计

（用 16K×8 位的芯片构成 64K×8 位的存储器）

连，地址总线 $A_0 \sim A_{13}$（$2^{14}=16K$），形成并联方式直接连接到各个芯片上。为了保证在任何时候只有一块芯片被选中，片选信号 A_{14}、A_{15} 由高位地址经译码器译码来提供。当 A_{14} A_{15} 为 00 时，选中 0 号芯片，芯片内的地址 $A_0 \sim A_{13}$ 由全 0 到全 1 变化时，地址由 0000H 到 3FFFH，当 $A_{14}A_{15}$ 为 10 时，选中 1 号芯片，芯片内的地址 $A_0 \sim A_{13}$ 由全 0 到全 1 变化时，地址由 4000H 到 7FFFH。依此类推，不难确定其他芯片的地址。

（3）字位同时扩展。当存储器芯片的位数和地址单元数都不满足系统的要求时，需要同时扩展位数与地址空间。例如，假设存储系统容量为 $M \times N$ 时，而存储芯片容量为 $m \times n$，则所需存储芯片数量为 $(M/m) \times (N/n)$。

如图 4.23 所示，用 1K×4 位的芯片构成 2K×8 位的存储器，需要用 (2K/1K)×(8/4) =4 片。将 4 片分成两组，将每 2 片组成一组 1K×8 的存储器，采用位线并联方式，其中一片接数据总线的低 4 位，另一片接数据总线的高 4 位；再使两组组合成 2KB 的存储器系统，用地址扩展的方式。因为 1KB 的芯片需要 $A_0 \sim A_9$ 共 10 位地址线，所以地址线串联到 4 个芯片。A_{10} 用来作为两组的片选控制信号。

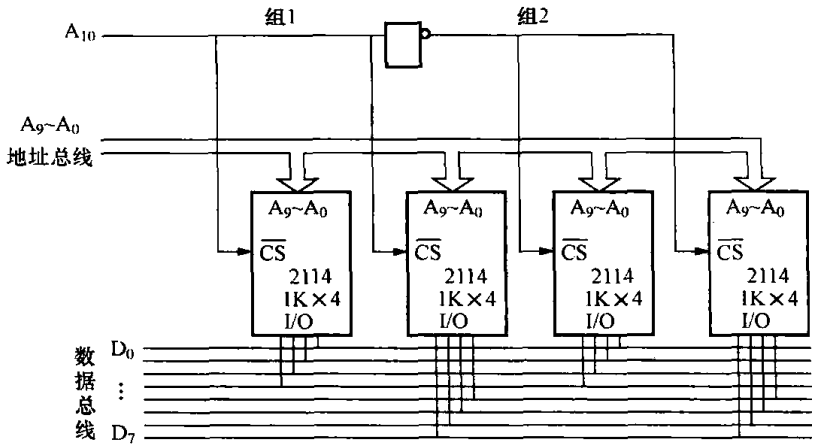


图 4.23 字和位扩展设计

（用 1K×4 位的芯片构成 2K×8 位的存储器）

可见，无论需要多大容量的存储器系统，均可利用有限的存储器芯片，通过字和位的扩展来构成。

2. *存储器系统的地址译码方式*

存储器系统通常由多片存储器芯片组成，CPU 要实现对存储单元的访问，首先要选择存储器芯片，然后再从选中的芯片中依照地址码选择相应的存储单元读/写数据。通常，存储器芯片选择信号是 CPU 输出的高位地址经过译码器产生，而芯片内部存储单元的地址由 CPU 输出的低位地址来确定。由此可见，存储单元的地址是由片选信号线和片内地址信号的状态共同决定。片选信号需要自行设计，片内地址单元确定由存储器芯片的内部译码电路产生，这部分译码电路不需用户设计。目前主要有三种片选信号的产生方法：线选法、部分地址译码法和全地址译码法。

（1）线选法。线选法是指用存储器芯片片内寻址线以外的系统高位地址线中的一条线来作为存储器芯片的片选控制信号。例如，在图 4.23 中 A_{10} 作为片选信号，就采用了线选法，A_{10} 为“0”选中组 1 芯片，A_{10} 为“1”选中组 2 芯片。

从图 4.23 中还可以看出，如果是 8086CPU，则系统的其他高位地址线 A_{19}～A_{11} 这 6 位高位地址线未参加译码，可以是任意值而不会影响芯片内部的地址编码，因而使得每组芯片的地址不唯一，即可以有出多组地址，这些地址称为重叠地址，即重叠地址是这些地址对应芯片的同一个单元。而且用不同的地址线做片选控制时，它们的地址空间也不相同。通常把 A_{19}～A_{11} 设为“全 0”所确定的一组地址，称为基本地址区域。

用 A_{10} 线选，假设 A_{19}～A_{11}＝000000000：

当 A_{10}＝0，选中组 1 芯片，其地址范围：00000H 和～003FFH；

当 A_{10}＝1，选中组 2 芯片，其地址范围：00400H 和～007FFH。

上述两组地址是连续的。

而用 A_{11} 线选，假设 A_{19}～A_{12}＝00000000，A10＝0：

当 A_{11}＝0，选中组 1 芯片，其地址范围：00000H 和～003FFH；

当 A_{11}＝1，选中组 2 芯片，其地址范围：00800H 和～00BFFH。

上述两组地址是不连续的。

当假设 A_{19}～A_{11} 为不同的值时，还会得出其他的地址范围。

总之，线选法的优点是选择芯片不需要外加逻辑电路，线路简单；但地址重叠，且会使地址空间不连续。CPU 的地址总线中有 n 位未参与译码，则有 2^n 个地址对应于一个内存单元。

（2）部分地址译码法。部分地址译码法是指用 CPU 高位地址线的一部分地址线，经过译码电路产生片选信号。例如，在图 4.22 中，16KB 容量的片内地址线 A_{13}～A_0 与系统低位地址线 A_{13}～A_0 对应连接，选择剩余的高位地址线中的 A_{15} 和 A_{14} 作为 2—4 线译码器的输入，译码器输出 $\overline{Y_0}$ 接至第一片的片选输入端 $\overline{CS}$，$\overline{Y_1}$ 接至第二片的片选输入端 $\overline{CS}$，依次类推。在确定芯片地址时，未连接的高位地址线 A_{19}～A_{16} 中每一条地址线的状态原则上可以是任意值。显然，部分译码也存在地址重叠的问题，这些地址被称为重叠地址。

（3）全地址译码法。全地址译码法是使存储器芯片片内寻址以外的系统的全部高位地址线都参与地址译码，经译码电路全译码后输出，作为各存储器芯片的片选信号，以实现对存储器芯片的读/写操作。

全地址译码法的优点是可以使每片（或组）芯片的地址范围不仅是唯一确定的，而且也是连续的，不会产生地址重叠现象，但对译码电路要求较高。

实现片选的译码电路可采用门电路或地址译码器完成（如 74LS138 译码器），图 4.24

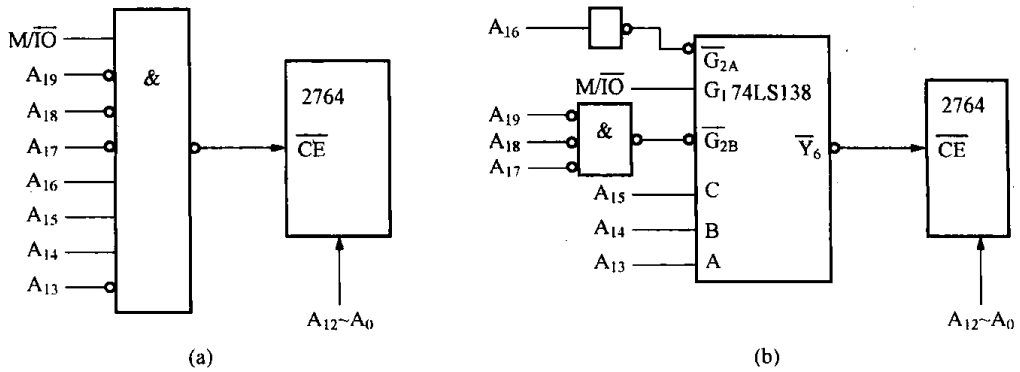


图 4.24 全地址译码法

（a）用门电路全译码；（b）用译码器全译码

为一种可能的全地址译码法实现的译码电路，2764芯片的片选信号是A_{19}～A_{13}=0001110，2764片内地址线A_{12}～A_0可以是全0到全1，则该芯片的地址范围是1C000H～1DFFFH。

图4.24中，74LS138译码器引脚如图4.25所示。74LS138的G_1、$\overline{G_{2A}}$、$\overline{G_{2B}}$为使能控制端，当它们为1，0，0时输出才有效，当输入端C、B、A三位相应组合为000～111时，每组译码使一个输出端为0，其余输出端为1。其译码真值表如表4.5所示。

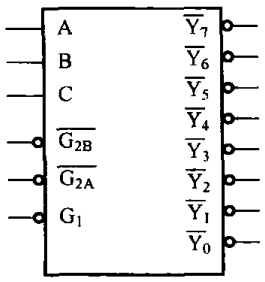


图4.25 74LS138译码器引脚

表4.5 74LS138译码器真值表

输入						输出
使能			选择			
G_1	$\overline{G_{2A}}$	$\overline{G_{2B}}$	C	B	A	$\overline{Y_7}$～$\overline{Y_0}$
1	0	0	0	0	0	$\overline{Y_0}$=0 其他均为1
1	0	0	0	0	1	$\overline{Y_1}$=0 其他均为1
1	0	0	0	1	0	$\overline{Y_2}$=0 其他均为1
1	0	0	0	1	1	$\overline{Y_3}$=0 其他均为1
1	0	0	1	0	0	$\overline{Y_4}$=0 其他均为1
1	0	0	1	0	1	$\overline{Y_5}$=0 其他均为1
1	0	0	1	1	0	$\overline{Y_6}$=0 其他均为1
1	0	0	1	1	1	$\overline{Y_7}$=0 其他均为1
0	×	×	×	×	×	$\overline{Y_7}$～$\overline{Y_0}$均为1
×	1	×	×	×	×	
×	×	1	×	×	×	

4.4.3 存储器与CPU的接口设计

存储器与CPU的接口设计是指存储器芯片与所确定的地址空间联系起来，即将芯片中的存储单元与实际地址一一对应，这样CPU才能通过寻址对存储单元进行读/写操作。

存储器与CPU的接口设计包括下列内容：

(1) 根据系统实际装机存储容量，确定存储器地址范围。

(2) 选择合适的存储器芯片及数量。

(3) 根据所选用存储芯片，选择译码方式和器件，列出地址分配表，确定片选和片内单元的地址线，设计片选译码电路。

(4) 将低位地址、数据线和控制线与存储芯片相连，画出存储器与CPU系统总线的连接图。

控制线的连接一般指存储器的$\overline{WE}$、$\overline{OE}$、$\overline{CS}$等与CPU的$\overline{RD}$、$\overline{WR}$等相连，不同的存储器和CPU连接时其使用的控制信号也不完全相同。

1. 8位CPU的存储器接口设计

由于存储器按字节进行组织，因此与8位CPU的接口设计比较简单。8088是准16位微处理器，内部寄存器和运算器为16位，其外部数据总线为8位，一个总线周期只能访问

一个字节，要进行字操作，必须用两个总线周期，第一个总线周期访问低位字节，第二个总线周期访问高位字节。

[例 4.1] 为8088CPU设计一个32KB容量的存储器。要求采用2732芯片构成8KB EPROM区，地址从00000H开始；采用6264芯片构成24KB RAM区，地址从02000H开始。片选信号采用全地址译码法。

解 第一步，确定实现8KB ROM和24KB RAM所需要的芯片数量。

由于每片2732提供4K×8位的存储容量，所以实现8KB ROM存储容量所需要的芯片数量是：

$$\frac{8K\times 8}{4K\times 8}=2(片)$$

因为每片6264提供8K×8位的存储容量，所以实现24KB RAM存储容量所需要的芯片数量是：

$$\frac{24K\times 8}{8K\times 8}=3(片)$$

第二步，存储器芯片片选信号的产生及电路设计。

采用74LS138译码器实现全地址译码法产生片选信号。存储器地址分配表如表4.6所示。

表4.6　　存储器地址分配表

		片选择码		片内译码	
芯片	地址范围	$A_{19}\sim A_{16}$	$A_{15}\sim A_{13}$	A_{12}	$A_{11}\sim A_0$
1号（2732）	0 0000H～0 0FFFH	0000	000	0 0	000…0 111…1
2号（2732）	01000H～01FFFH	0000	000	1 1	000…0 111…1
3号（6264）	02000H～03FFFH	0000	001	0 1	000…0 111…1
4号（6264）	0 4000H～0 5FFFH	0000	010	0 1	000…0 111…1
5号（6264）	0 6000H～0 7FFFH	0000	011	0 1	000…0 111…1

由于2732的片内地址为$A_{11}\sim A_0$，6264的片内地址为$A_{12}\sim A_0$，从表4.6地址分配情况分析，将$A_{12}\sim A_0$作为片内地址线，系统地址线$A_{15}\sim A_{13}$连接译码器74LS138的输入端C、B、A，$A_{19}\sim A_{16}$作为译码器的控制使能端，译码的输出作为芯片的片选信号。存储器系统设计电路如图4.26所示。两片2732的片内地址$A_{11}\sim A_0$与系统地址线$A_{11}\sim A_0$连接，译码器输出端$\overline{Y_0}$和A_{12}经过二次译码产生片选。$\overline{Y_0}$和A_{12}经“或门”输出与1号2732的$\overline{CE}$连接，A_{12}反相后和译码器输出端$\overline{Y_0}$经“或门”输出与2号2732的$\overline{CE}$连接。三片6264的片内地址$A_{12}\sim A_0$与系统地址线$A_{12}\sim A_0$连接，它们的片选$\overline{CE}$分别连接译码器的输出端$\overline{Y_1}$、$\overline{Y_2}$、$\overline{Y_3}$。

第三步，由于存储器芯片具有8根数据线，与CPU的8位数据总线相连时，按位一一对应相连即可。

第四步，控制线的连接。M/$\overline{IO}$参加译码控制是必要的，它使$\overline{CE}$仅在访问内存时产生有效信号，保证正确寻址存储器单元。EPROM的控制信号通常有$\overline{CE}$及$\overline{OE}$，SRAM的控制信号通常有$\overline{CE}$、$\overline{OE}$及$\overline{WE}$。$\overline{CE}$为片选信号与译码器的输出相连；$\overline{OE}$为输出使能信号，应与CPU的$\overline{RD}$信号相连。$\overline{WE}$为写使能信号，应与CPU的$\overline{WR}$相连。

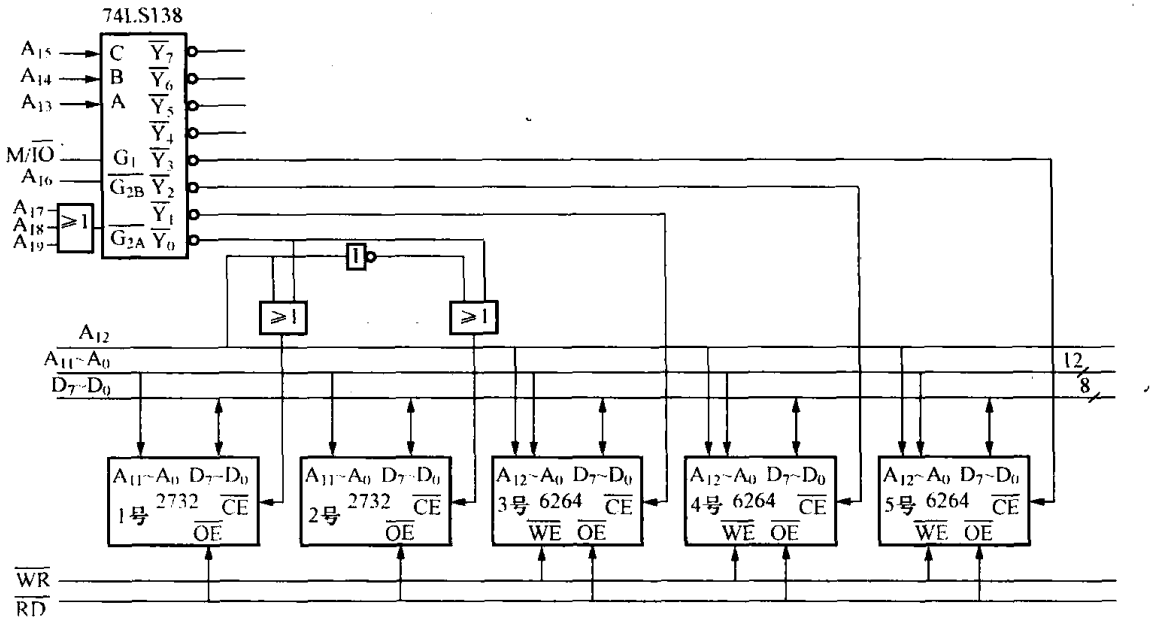


图 4.26 存储器系统设计电路

2. 16 位 CPU 的存储器接口设计

(1) 16 位微机存储器奇偶分体。在 16 位微机系统中，CPU 既可以进行字节操作也可以进行字操作，因此在存储器系统设计中，存储器芯片连接到 16 位微机系统时，要按照奇偶存储体（高 8 位存储体与低 8 位存储体）进行设计。

以 8086 系统为例，8086CPU 有 20 条地址线，可直接寻址 1MB 内存地址空间，而这 1MB 的地址空间是按字节顺序排列的。由于 8086CPU 的数据总线是 16 位的，为了能满足一次存取一个字又能存取一个字节的数据要求，8086 系统中将 1MB 的地址空间分成两个 512K 字节的存储体——“偶存储体”（所有偶数地址单元构成的存储体）和“奇存储体”（所以奇数地址单元构成的存储体）。偶存储体的数据线同 8086 的低 8 位数据总线 $D_7 \sim D_0$ 相连接（也称为低 8 位存储体），奇存储体同 8086 的高 8 位数据总线 $D_{15} \sim D_8$ 相连接（也称为高 8 位存储体），地址总线的 $A_{19} \sim A_1$ 同两个存储体中的地址线 $A_{18} \sim A_0$ 相连接，最低位地址线 A_0 和 8086 的“总线高 8 位允许”信号$\overline{BHE}$用来选择存储体，如图 4.27 所示。A_0 和$\overline{BHE}$对奇偶存储体的选择如见表 4.7。

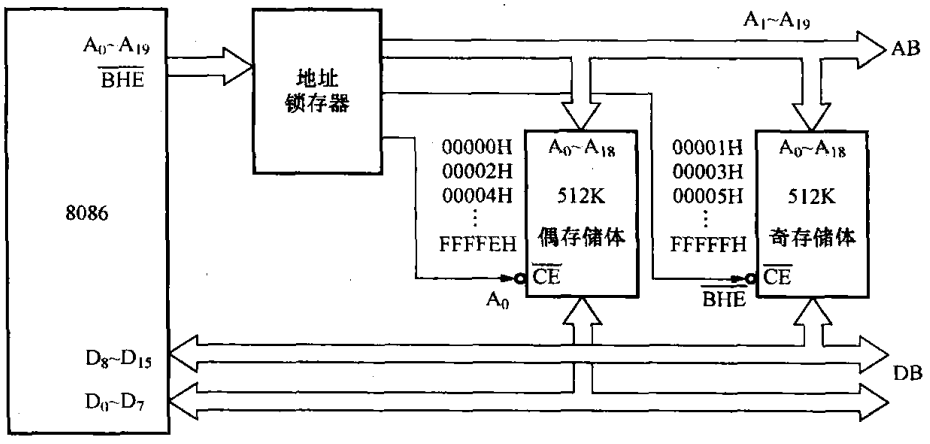


图 4.27 8086 系统存储器的奇偶分体

表 4.7 奇偶存储体选择编码表

$\overline{BHE}$	A_0	传送的字节
0	0	两个字节 （偶地址单元为低 8 位数据，奇地址单元为高 8 位数据）
0	1	奇地址单元的高位字节
1	0	偶地址单元的低地位字节
1	1	不传送

（2）存储器与 8086 CPU 的连接。图 4.28 给出了 8086 CPU 工作在最小方式下与 EPROM、静态 RAM 连接的实例。

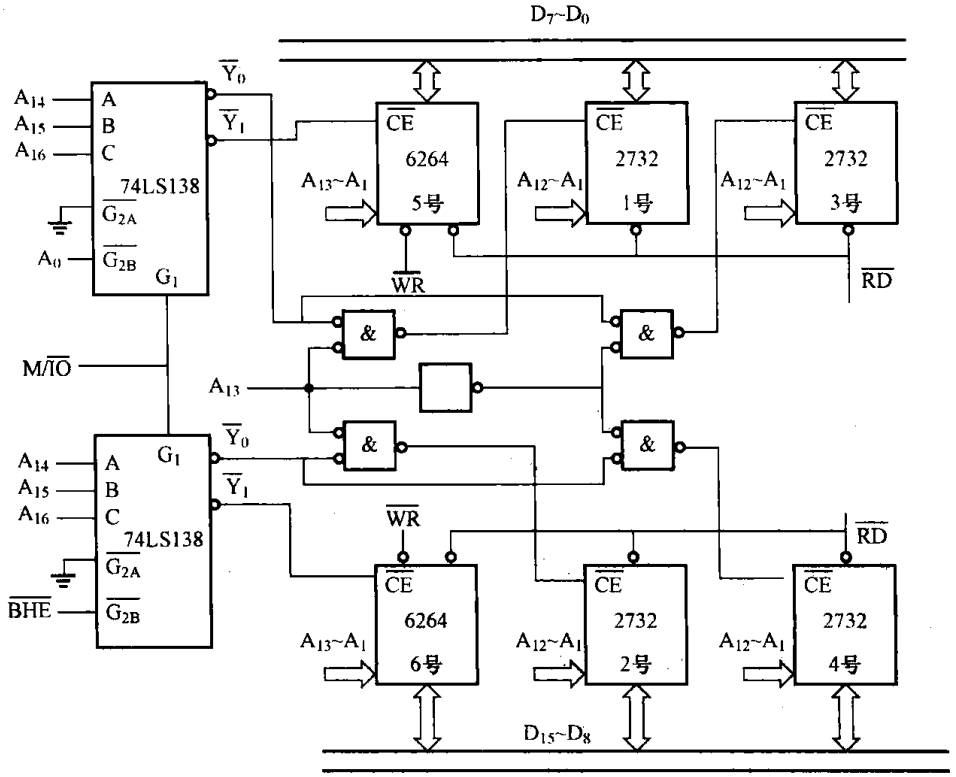


图 4.28 EPROM、静态 RAM 与 8086 CPU 连接

本系统使用了 4 片 2732（4K×8 位）EPROM 芯片来构成 16KB 的 EPROM 存储区，分别用 1～4 号表示，每两片构成一组奇偶存储体，共 2 组。2732 片内用 12 位地址线 A_1～A_{12} 寻址，6264 片内用 13 位地址线 A_1～A_{13} 寻址，74LS138 译码器的输入端 C、B、A 分别连地址线 A_{16}～A_{14}。通过两个 74LS138 译码器的输出端信号 $\overline{Y}_0$ 控制该 2 组 2732 的输出允许信号端 $\overline{OE}$。同时，用 8086 的 A_0 和 $\overline{BHE}$ 作为输入信号分别接至两个 74LS138 译码器 $\overline{G}_{2B}$ 端，来实现偶地址译码器和奇地址译码器的控制，从而控制各组内奇偶存储体的选择。显然，1、3 号是受 A_0（$A_0=0$）控制的偶存储体，而 2、4 号是受 $\overline{BHE}$（$\overline{BHE}=0$）控制的奇存储体。

本系统还使用了 2 片 6264（8K×8 位）静态 RAM 芯片构成 16K 字节的 RAM 区，它们分别以 5、6 号表示，该两片为一组构成奇偶存储体，通过两个 74LS138 译码器的输出端信号 $\overline{Y}_1$ 控制该组 6264 的片选端 $\overline{CE}$。

系统读（$\overline{RD}$）、写（$\overline{WR}$）信号直接接到RAM芯片的$\overline{OE}$和$\overline{WE}$端，以控制数据的传送方向。由于6264和2732的容量不同，芯片地址线的条数不同，所以对于2732来说74LS138译码器的输出与A_{13}又进行了二次译码作为片选。

说明：高位地址线A_{19}、A_{18}、A_{17}未使用表示取0、1均可，所以每块芯片将同时有$2^3=8$个重叠区，假设未使用的地址线用0表示，地址范围可以很容易被确定，如表4.8所示。

表4.8　　EPROM和SRAM地址分配表

芯片	高位地址线状态						低位地址线状态			确定的地址范围
	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	$A_{12}\cdots A_1$	A_0	
1号2732	0	0	0	0	0	0	0	0……0 1……1	0	00000H 01FFEH
2号2732	0	0	0	0	0	0	0	0……0 1……1	1	00001H 01FFFH
3号2732	0	0	0	0	0	0	1	0……0 1……1	0	02000H 03FFEH
4号2732	0	0	0	0	0	0	1	0……0 1……1	1	02001H 03FFFH
5号6264	0	0	0	0	0	1	0……0 1……1		0	04000H 07FFEH
6号6264	0	0	0	0	0	1	0……0 1……1		1	04001H 07FFFH

3. 32位CPU的存储器接口设计

将16位微机系统的存储器设计方法用于32位微机系统中，由于微处理器有32位地址总线，32位数据总线，因此存储器系统分为4个存储体，如图4.29所示。

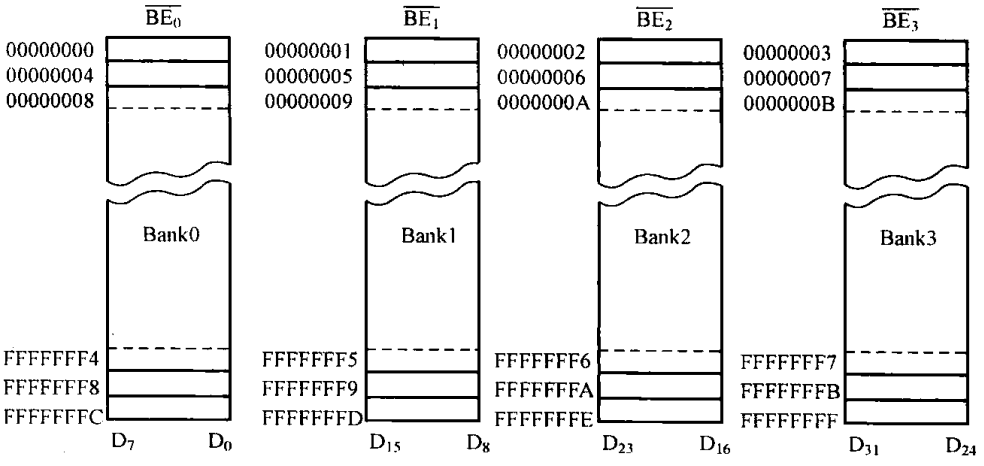


图4.29　32位系统中的存储器组织

由图4.29可见，每个存储体占1GB存储空间，存储体选择通过字节允许线$\overline{BE_3}\sim\overline{BE_0}$实现。每个存储体都需要独立的写选通信号，以实现32位或16位或8位的读/写操作。

32位系统中由于地址线多，通常采用PLD（可编程逻辑器件）译码器。地址总线A_0与A_1不作考虑，而地址总线A_2与存储器地址线A_0相连接，对16位、32位数的读写也存在“对准”的问题（参考2.2.5内容）。

[例 4.2] 为 80486 CPU 设计一个存储器子系统，该存储器子系统包含 64K×8 位的 RAM 存储器和 64K×8 位的 ROM 存储器。用 SRAM4363（16K×4）实现 RAM 存储器，用 EPROM27C128（16K×8）实现 ROM 存储器。这两种存储器芯片引脚如图 4.30 所示。RAM 存储器的地址范围是 00000H～0FFFFH，ROM 存储器的地址范围是 F0000H～FFFFFH。

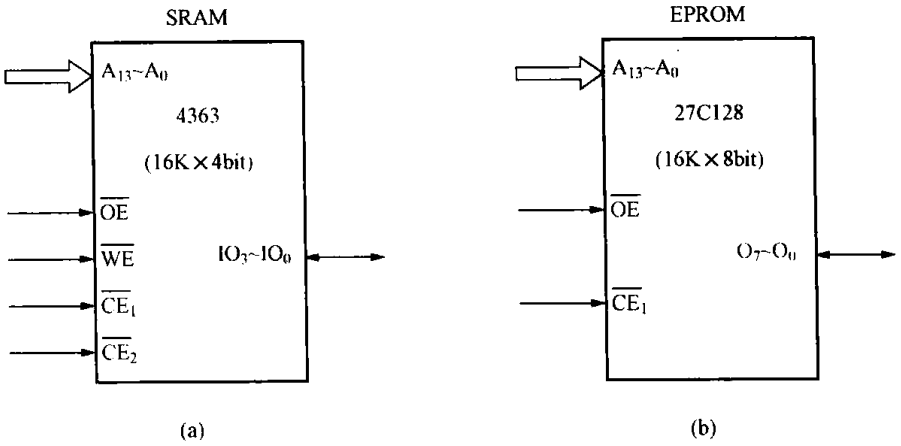


图 4.30　两种存储器电路

（a）4363 芯片引脚信号；（b）27C128 芯片引脚信号

解　第一步，确定实现 64K×8 位 RAM 及 64K×8 位 ROM 存储体所需要的芯片的数量。

因为每片 4363 提供 16K×4 位的存储容量，所以实现 64K×8 位存储容量所需要的 SRAM 芯片数量是：

$$\text{SRAM 数量} = \frac{64\times 8}{16\times 4} = 8(\text{片})$$

为了连接到 32 位数据总线上，必须将全部八片 SRAM4363 芯片（1～8 号）的数据线依次并联，与 32 位数据总线相连，每两片 4363 构成 16K×8 的存储体，共 4 个。

由于每片 27C128 提供 16K×8 位的存储容量，所以实现 64K×8 位的存储容量所需要的芯片数量是：

$$\frac{64\times 8}{16\times 8} = 4(\text{片})$$

第二步，列出存储体地址分配表。

80486 微处理器中没有 A_1A_0 两条地址线，对于存储器的 A_1A_0 的组合状态编码是由 80486 微处理器$\overline{BE_3}$～$\overline{BE_0}$控制产生，即$\overline{BE_0}$=0，表示 A_1A_0=00；$\overline{BE_1}$=0，表示 A_1A_0=01；$\overline{BE_2}$=0，表示 A_1A_0=10；$\overline{BE_3}$=0，表示 A_1A_0=11。存储器的地址分配如见表 4.9 所示。

表 4.9　　存储器地址分配表

芯片		地址范围	地址线													
			A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	…	A_2	A_1	A_0	$\overline{BE_3}$	$\overline{BE_2}$	$\overline{BE_1}$	$\overline{BE_0}$
SRAM 4363	1 号	00000H	0	0	0	0	0	0	0	…	0	0	0	1	1	1	0
	2 号	00004H	0	0	0	0	0	0	0	…	1						
		…						……									
		0FFFCH	0	0	0	0	1	1	1	…	1						

续表

芯片		地址范围	地址线													
			A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	…	A_2	A_1 A_0	$\overline{BE_3}$	$\overline{BE_2}$	$\overline{BE_1}$	$\overline{BE_0}$
SRAM 4363	3号 4号	00001H	0	0	0	0	0	0	0	…	0	0 1	1	1	0	1
		00005H	0	0	0	0	0	0	0	…	1					
		…					……									
		0FFFDH	0	0	0	0	1	1	1	…	1					
	5号 6号	00002H	0	0	0	0	0	0	0	…	0	0 0	1	0	1	1
		00006H	0	0	0	0	0	0	0	…	1					
		…					……									
		0FFFEH	0	0	0	0	1	1	1	…	1					
	7号 8号	00003H	0	0	0	0	0	0	0	…	0	1 1	0	1	1	1
		00007H	0	0	0	0	0	0	0	…	1					
		…					……									
		0FFFFH	0	0	0	0	1	1	1	…	1					
EPROM 27C128	1号	F0000H	1	1	1	1	0	0	0	…	0	0 0	1	1	1	0
		F0004H	1	1	1	1	0	0	0	…	1					
		…					……									
		FFFFCH	1	1	1	1	1	1	1	…	1					
	2号	F0001H	1	1	1	1	0	0	0	…	0	0 1	1	1	0	1
		F0005H	1	1	1	1	0	0	0	…	1					
		…					……									
		FFFFDH	1	1	1	1	1	1	1	…	1					
	3号	F0002H	1	1	1	1	0	0	0	…	0	1 0	1	0	1	1
		F0006H	1	1	1	1	0	0	0	…	1					
		…					……									
		FFFFEH	1	1	1	1	1	1	1	…	1					
	4号	F0003H	1	1	1	1	0	0	0	…	0	1 1	0	1	1	1
		F0007H	1	1	1	1	0	0	0	…	1					
		…					……									
		FFFFFH	0	0	0	0	1	1	1		1					

第三步，存储器芯片片选信号、写信号的产生及电路设计。

存储器子系统需要两个片选信号$\overline{CS_{RAM}}$和$\overline{CS_{RCM}}$。系统中 $A_{19}A_{18}A_{17}A_{16}=0000$ 译码产生$\overline{CS_{RAM}}$，$A_{19}A_{18}A_{17}A_{16}=1111$ 译码产生$\overline{CS_{RCM}}$。片选信号逻辑电路的设计如图 4.31 所示。其中采用一块 74LS7400 “与非”门，一块 74LS7425 “或非”门和一块 74LS7404 “非”门。

系统的写控制逻辑由存储器写命令信号$\overline{MWTC}$和$\overline{BE_3}$～$\overline{BE_0}$组合产生，以便在对 RAM 存储体进行写操作时，为 32 位数据字的每个字节产生独立的写允许信号。这里用“或”门 74LS7432 来实现，如图 4.32 所示。

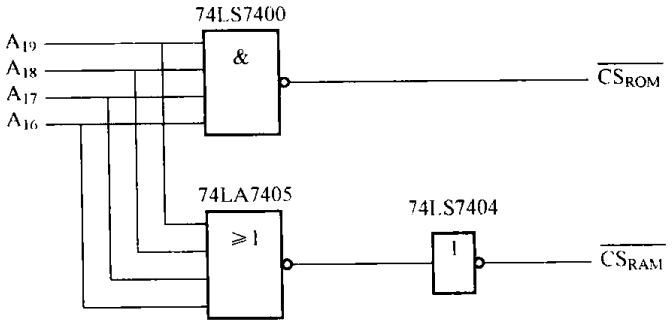


图 4.31 片选信号逻辑电路设计

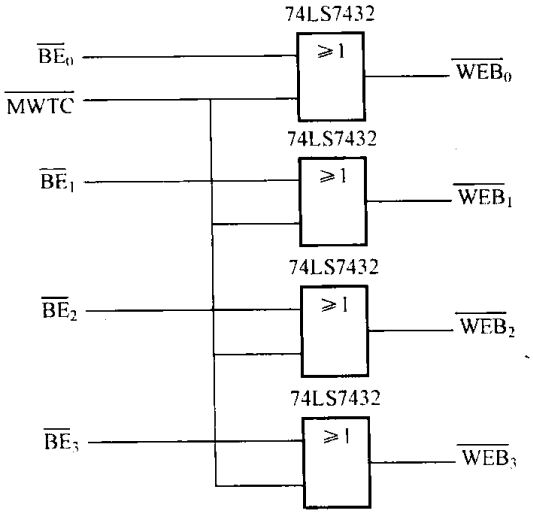


图 4.32 写信号逻辑电路

第四步，画出存储器子系统的硬件连接图。

存储器子系统硬件连接电路如图 4.33 所示。图 4.32 中的存储器写命令信号$\overline{MWTC}$和图 4.33 中的存储器读命令信号$\overline{MRDC}$由 80486 CPU 的$\overline{ADS}$、M/$\overline{IO}$、D/$\overline{C}$和 W/$\overline{R}$通过总线控制逻辑电路产生。

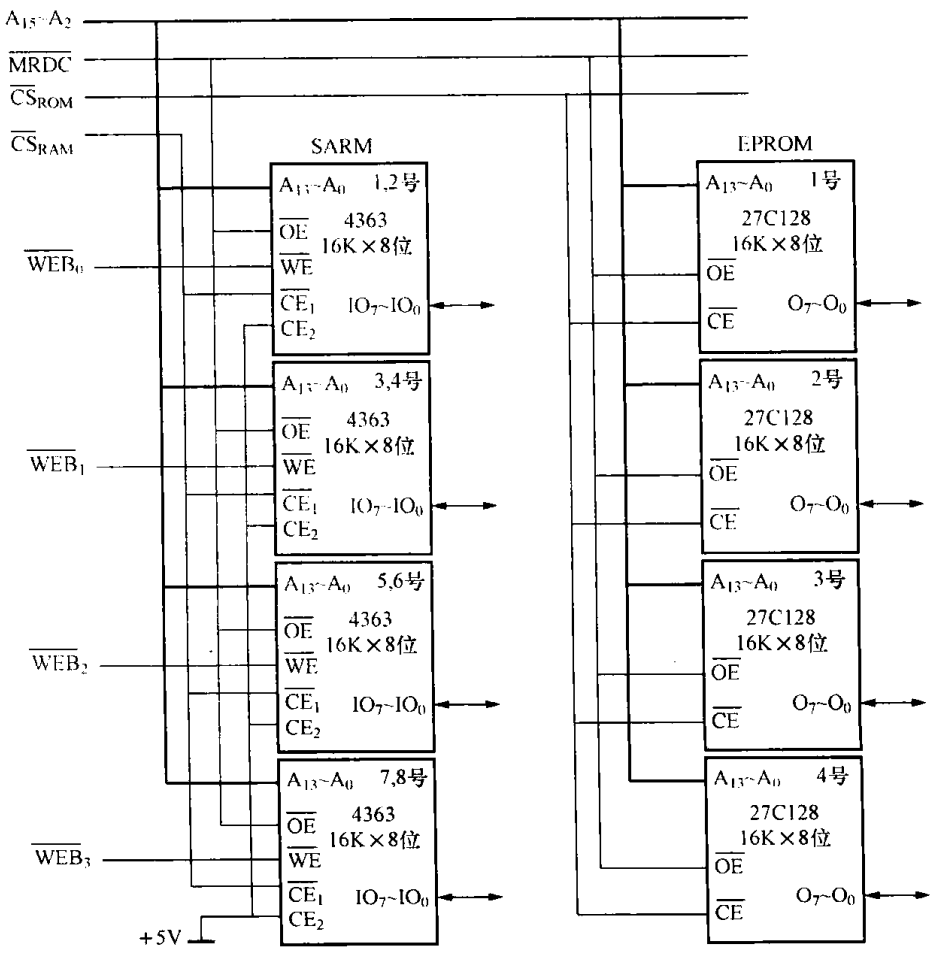


图 4.33 存储器子系统硬件连接电路图

4.5 高速缓冲存储器Cache

对于微机系统性能的提高不仅仅取决于CPU，还与系统架构、指令结构、以及信息在各个部件之间的传送速度及存储部件的存取速度等因素有关，特别是CPU与内存之间的存取速度有关。若CPU工作速度较高，但内存存取速度相对较低，则造成CPU等待，降低处理速度，浪费CPU的能力。例如，主频为500MHz的CPU，一次指令执行时间为2ns，而与其相配的内存（SDRAM）存取时间为10ns，则不能发挥出CPU所具有的高性能。

在80386以后的微机系统中，在保证系统性能价格比的前提下，为了减少CPU与内存之间的速度差异，在慢速的DRAM和快速CPU之间接入一速度较快、容量较小的SRAM，起到缓冲作用；使CPU既可以以较快速度存取SRAM中的数据，又不使系统成本增加。这里小容量的SRAM芯片组成高速缓存器（Cache），低价格大容量的DRAM芯片（或内存条）组成主存储器。

统计表明，利用Cache可使存储器的存取速度提高4～10倍。当速度差更大时，可采用多级Cache。目前大多数微机的高速缓存分为L1 Cache和L2 Cache两级。L1 Cache集成在CPU芯片内，时钟周期与CPU相同；L2 Cache通常封装在CPU芯片之外，采用SRAM芯片。就容量而言，L2 Cache的容量通常比L1 Cache大一个数量级以上，从几百KB到几千KB不等。80486CPU芯片内有8KB的Cache，存放程序和数据，并支持L2 Cache。

4.5.1 Cache的工作原理

Cache的工作原理是基于程序访问的局部性原理。通过对大量典型程序运行情况的结果分析，在一个较短的时间间隔内，由程序产生的地址往往集中在存储器逻辑地址空间的很小范围内。由于指令地址是连续分布的，另外循环程序段和子程序段要重复执行多次。因此，对这些程序地址的访问具有时间上的集中倾向。数据分布的这种集中倾向不如指令明显，但对数组的存储和访问以及工作单元的选择都可以使存储器地址相对集中。这种对局部范围的存储器地址频繁访问，而对此范围以外的地址则访问甚少的现象，就称为程序访问的局部性。

高速缓冲存储器Cache的工作原理如图4.34所示。

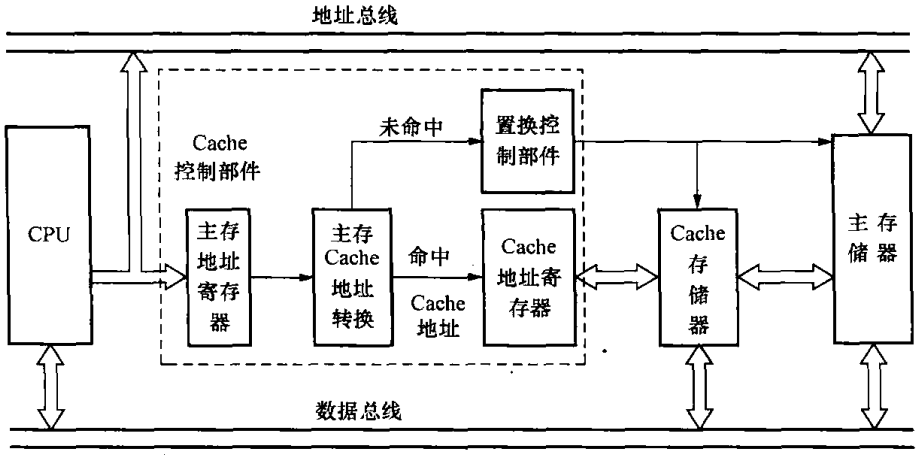


图4.34 Cache工作原理图

微机系统正是依据程序的局部性原理，不断地将与当前指令集相关联的一个不太大的后继指令集从内存读到 Cache，然后再与 CPU 高速传送，从而达到速度匹配。CPU 对存储器进行数据请求时，通常先访问 Cache。由于局部性原理不能保证所请求的数据百分之百地在 Cache 中，这里便存在一个命中率。即 CPU 在任一时刻从 Cache 中可靠获取数据的几率。命中率越高，正确获取数据的可靠性就越大。一般来说，Cache 的存储容量比主存储器的容量小得多，但不能太小，太小会使命中率太低；而过大不仅会增加成本，而且当容量超过一定值后，命中率随容量的增加将不会有明显的增长。只要 Cache 的空间与主存储器空间在一定范围内保持适当比例的映像关系，Cache 的命中率还是相当高的。

$$\text{命中率}=\frac{\text{命中次数}}{\text{存储器访问总次数}}\times 100\%$$

一般规定 Cache 与内存的空间比为 4∶1000，即 128KB Cache 可映像 32MB 内存；256KB Cache 可映像 64MB 内存。在这种情况下，命中率都在 90%以上。至于没有命中的数据，CPU 只好直接从内存获取。获取的同时，也把它拷进 Cache，以备下次访问。

4.5.2 主存储器与 Cache 的地址映像

主存储器和 Cache 的存储区均划分成块（Block），每块由多个信息字组成，两者之间以块为单位交换信息。Cache 中应能容纳多个信息块，但由于 Cache 比主存储器小得多，因此必须使用一种机制将主存储器地址定位到 Cache 中，即地址映像。

Cache 的地址映像以块为单位进行映像：将主存储器和 Cache 都分为大小相等的若干块（或称页），每块的大小为 2^n 个字节，通常有 2^9（512B）的块、2^{10}（1024B）块、2^{11}（2048B）的块等。假设某系统的 Cache 容量为 1MB，每块容量为 1KB，则被分为 1024 块；Cache 容量为 8KB，每块容量也是 1KB，则被分为 8 块。不同的映像关系决定了 Cache 的不同结构。下面以此为例，介绍三种 Cache 的地址映像方法：直接地址映像 Cache、全相连 Cache 和组相连 Cache。

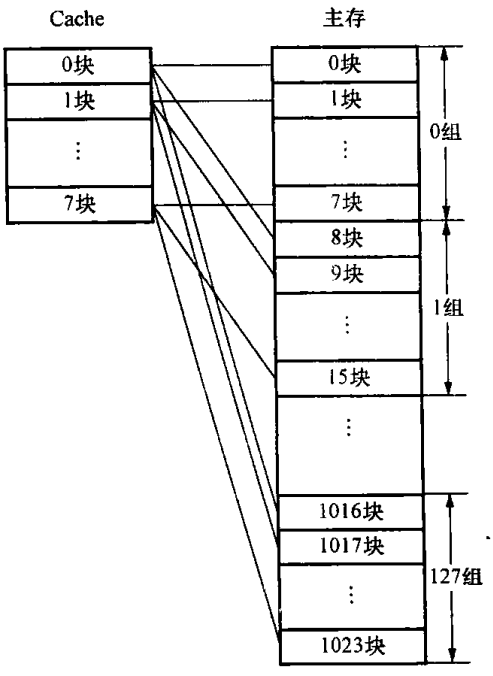


图 4.35 Cache 直接地址映像

1. 直接地址映像（Direct Mapping）

直接地址映像是指主存储器中块只能映像到 Cache 中某一固定的块，而不能映像到其他的块。例如，把主存储器按 Cache 大小分为若干组，每一组按对应的块号进行映像。如主存储器的第 0 块、第 8 块、…、第 1016 块，只能映像到 Cache 的第 0 块；而主存储器的第 1 块、第 9 块、…、第 1017 块只能映像到 Cache 的第 1 块，依次类推，如图 4.35 所示。

直接地址映像方法比较简单，且地址转换速度快，但缺点是对于每个主存储器块规定了固定的 Cache 块的替换位置，某几个主存储器块只能对应唯一的 Cache 块中，不够灵活，使得 Cache 的存储空间得不到充分利用。

2. 全相连地址映像（Fully Associative Mapping）。

与直接映像固定对应关系的方法相反，全相连地

址映像是允许每个主存储器块可以映像到 Cache 的任意块中，如图 4.36 所示。主存储器块与 Cache 块的映像是无序的，这就需要主存储器块的地址和 Cache 中每块的标记位相对应，由 Cache 块的标记域指出目前装入该 Cache 块的数据来源于主存储器中的哪一块，标记域是标识主存储器块的唯一标志。全相连 Cache 映像方式给 Cache 数据替换带来较大的灵活性，不易产生冲突。其缺点是标记位需要表示存储器的大部分地址，比较电路难于实现，且大范围无序数据的查找十分耗时，效率低，速度慢。

3. 组相连地址映像（Set Associative Mapping）

组相连地址映像是介于直接地址映像和全相连地址映像之间的一种结构，如图 4.37 所示。主存储器和 Cache 都分组，主存储器中一个组内的块数与 Cache 中的分组数相同。组间采用直接地址映像，而组内采用全相连地址映像。主存储器中的各块与 Cache 的组号间有固定的映像关系，但可以自由映像到对应的 Cache 组中的任何一块。如主存储器中的第 0 块可映像到 Cache 的第 0 组的第 0 块或第 1 块；主存储器中的第 1 块可映像到 Cache 的第 1 组的第 2 块或第 3 块等。这种映像方法比直接地址映像灵活，比全相连地址映像速度快。实际上，若组的大小为 1 时，就变成了直接映像；若组的大小为整个 Cache 的尺寸时，就变成了全相连映像。

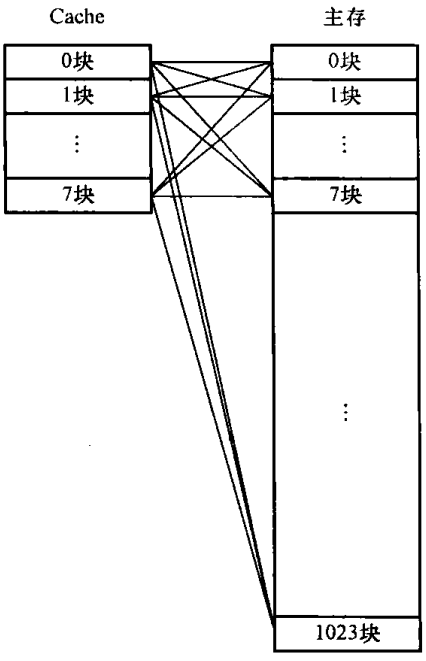


图 4.36 Cache 全相连地址映像

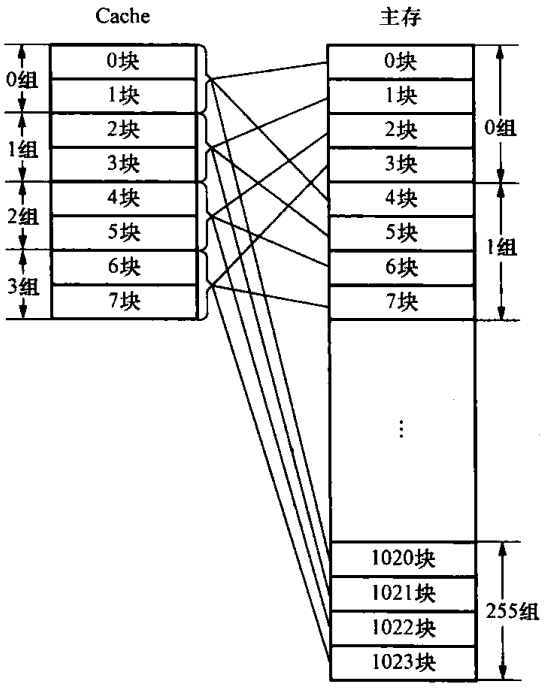


图 4.37 Cache 组相连地址映像

4.5.3 Cache 与 DRAM 存取的一致性

在 CPU 与主存储器之间增加了 Cache 之后，便存在数据在 CPU 和 Cache、CPU 和主存储器之间如何存取的问题，即读/写策略。

读/写策略依然是在 Cache 中应尽量存放 CPU 最近一直在使用的数据。当 Cache 装满后，可将长期不用的数据删除，以提高 Cache 的使用效率。为保持 Cache 中的数据与主存储器中的数据的一致性，同时避免 CPU 在读/写过程中遗失新数据，确保 Cache 中

更新过的数据不会因覆盖而消失，必须将 Cache 中的数据及时更新并准确地反映到主存储器。由于涉及 CPU、Cache 与主存储器三者之间的协调关系，使得读/写操作较复杂。

1. 读策略

(1) 贯穿读出式 (Look Through)。贯穿读出式的原理如图 4.38 所示。Cache 位于 CPU 与主存储器之间，CPU 对主存储器的所有数据请求都首先送到 Cache，由 Cache 在自身查找。如果命中，则切断 CPU 对主存储器的请求，并将数据送出；如果未命中，则将数据请求传给主存储器。该方法的优点是降低了 CPU 对主存储器的请求次数，缺点是延迟了 CPU 对主存储器的访问时间。

(2) 旁路读出式 (Look Aside)。旁路读出式的原理如图 4.39 所示。CPU 发出数据请求，并不是单通道地穿过 Cache，而是向 Cache 和主存储器同时发出请求。由于 Cache 速度更快，如果命中，则 Cache 在将数据回送给 CPU 的同时，还来得及中断 CPU 对主存储器的请求；若未命中，则 Cache 不做任何动作，由 CPU 直接访问主存储器。它的优点是没有时间延迟，缺点是每次 CPU 都要访问主存储器，这样就占用了部分总线时间。

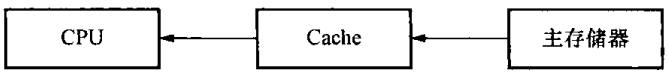


图 4.38 Cache 贯穿读出式原理

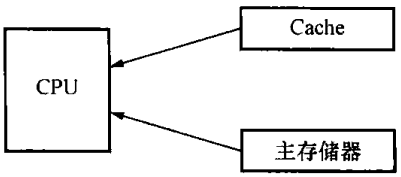


图 4.39 Cache 旁路读出式原理

在旁路读出式原理中，还有另外一个值得注意的概念，那就是“行填充”。每当 CPU 由主存储器读入数据时，同时还要将该数据复制到 Cache 中。即使当前 CPU 仅读一个字节，Cache 控制器也总是要将主存储器中包含该字节的一个完整的 Cache 行复制到 Cache 中。Intel 体系的 32 位微处理器中一个 Cache 行由 32 个字节组成，这种从主存储器向 Cache 传送一行数据的操作就称为 Cache 行填充 (Line Fill)，这是考虑到数据与代码往往具有连续性，CPU 下一次所需的数据或代码通常还是处在上一次的数据或代码附近，即同处一 Cache 行中的可能性最大，这样处理就提高了 Cache 的命中率。由此可见，Cache 中的数据是由空到满逐行建立起来的。

2. 写策略

写操作时 Cache 控制器同样会判断其地址是否定位在 Cache 中。如果在，CPU 的数据就会写到 Cache 中。Cache 控制器有以下几种主要的写策略：

(1) 全写方式 (Write Through)。全写方式的原理如图 4.40 所示。任一从 CPU 发出的写信号送到 Cache 的同时，也写入主存，以保证主存的数据能同步地更新。该方式的优点是操作简单，缺点是产生大量的写存储器操作，Cache 的优越性不能发挥，使系统效率降低。

(2) 回写方式 (Write Back)。回写方式的原理如图 4.41 所示。这种方式下，数据一般只写到 Cache，这样有可能出现 Cache 中的数据得到更新而主存储器中的数据却没有变（数据没有同时更新）的情况。因此，在 Cache 中设一标志地址及更新标志信息，只有当 Cache

中的数据被再次更新时，才将原更新的数据写入主存储器相应的单元中，然后再接受再次更新的数据。这样保证了 Cache 和主存中的数据不致产生冲突。

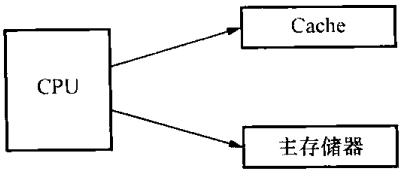


图 4.40 Cache 全写方式原理

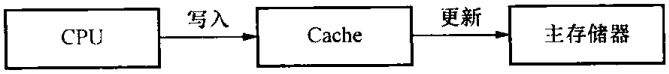


图 4.41 Cache 回写方式原理

4.5.4 Cache 与内存间的替换策略

当 Cache 访问不命中，需要访问主存储器并将主存储器数据块装入某一 Cache 块中，而当 Cache 已经装满后，这就产生了 Cache 块数据的替换策略。那么应替换哪些 Cache 块才能提高命中率呢？理想的替换策略应该使得 Cache 中总是保存着最近将要使用的数据，不用的数据则被替换掉，这样才能保证很高的命中率。当使用全相连和组相连方式时，就会使用到替换策略或是替换算法。常用的算法有下面三种。

1. 近期最少使用（LRU）替换策略

近期最少使用 LRU（Least Recently Used）替换策略的基本思想是：替换在 Cache 中驻留时间最长且未被引用的块。这种替换算法相对合理，命中率最高，是目前最常采用的方法。它需要随时记录 Cache 中各块的使用情况，以便确定哪个块是近期最少使用的块，实现起来比较复杂，系统开销较大。

2. 先进先出（FIFO）替换策略

先进先出（FIFO）替换策略的基本思想是：替换在 Cache 中驻留时间最长的块。这种策略不需要随时记录各个块的使用情况，容易实现，且系统开销小。其缺点是一些需要经常使用的程序块可能会被调入的新块替换掉。

3. 随机（Random）替换策略

随机替换候选块中的一个，与以前使用的 Cache 块无关，这是一种最简单的方法。不管是哪种策略，它们都只能用硬件电路来实现。

4.6 虚拟存储器及其管理技术

前面我们介绍了 Cache 和主存储器的概念，这类存储器称为物理存储器。而虚拟存储器由主存储器和外存储器共有，把外存储器作为物理存储器的扩充。从用户的角度看，可以在计算机有限的物理存储器上运行非常大的程序，好像计算机系统具有一个容量很大的主存储器，所以称之为虚拟存储器。

本节重点介绍虚拟存储器的概念及虚拟存储器管理的三种方法：分段存储管理、分页存储管理和段页存储管理。

4.6.1 虚拟存储器的工作原理

1. 虚拟存储器的基本概念

虚拟存储器将主存储器与外存储器相结合，将程序分成若干程序块，部分程序块装入主存储器中运行，其余程序块被放在外存储器中。虚拟存储系统负责主存储器中废弃程序块的

删除，释放内存空间，以及外存储器程序块调入。虚拟存储系统实际上就是内在功能很强的一个操作系统的子系统，为程序分配和释放空闲地址，同时还负责维护完成虚拟存储器向物理存储器存储空间的转换，并保持跟踪未被使用的物理存储器。

在虚拟存储器中需要注意如下三个概念：

（1）虚拟地址空间。是程序员编写程序，在指令中表述存储器操作数所使用的地址空间，其相应的地址称为虚拟地址或逻辑地址。

（2）主存储器地址空间。又称为实存地址空间，是存储、运行程序的空间，其相应的地址称为主存储器物理地址或实地址。

（3）外存储器地址空间或磁盘地址空间。

当用虚拟地址访问主存储器时，系统首先查看所用虚拟地址对应的单元内容是否已装入主存储器。如果在主存储器中，可以通过辅助软、硬件自动把虚拟地址变成主存储器的物理地址，对主存储器的相应单元进行访问。如果不在主存储器中，通过辅助的软、硬件将虚拟地址对应的内容调入主存储器中，然后再进行访问。因此，对虚拟存储器的每次访问都必须进行虚实地址的变换。

80486 在保护方式下具有 64TB（2^{46}）的逻辑地址空间和 4GB（2^{32}）的物理地址空间。由于虚存地址空间比主存储器地址空间大得多，必须根据某种规则把按逻辑地址编写的程序装入到主存储器中去，并将逻辑地址转换成对应的主存储器物理地址，这一过程称为地址变换。通过地址变换后，程序才能运行。

虚拟存储器中涉及的主存储器与外存储器的关系类似于主存储器与高速缓冲存储器的关系，但主存储器—Cache 体系结构和主存储器—外存储器体系结构还有一些差别。

（1）主存储器—Cache 体系结构是为了满足程序对速度的要求，容量小，传送的数据块长度短，读/写速度快；而主存储器—外存储器体系结构是为了满足容量的要求，容量大，传送的数据块长度长，读/写速度相对较慢。

（2）在主存储器—Cache 体系结构中，CPU 可以直接访问 Cache 和主存储器；而在主存储器—外存储器体系结构中，CPU 不可以直接访问外存储器。

（3）为了保证速度，主存储器—Cache 体系结构的存取信息过程、地址变换和替换策略全部采用硬件来实现，而主存储器—外存储器体系结构基本上由操作系统的存储管理软件辅助一些硬件进行数据块的划分来实现主存储器—外存储器之间的调度，所以需要设计存储管理软件来实现这些功能。

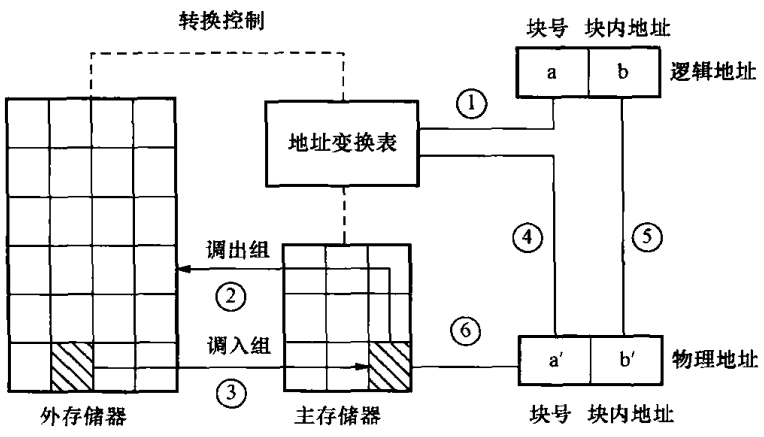


图 4.42 虚拟存储器工作过程

2. 虚拟存储器的工作过程

虚拟存储器由硬件和操作系统自动实现对存储信息的调度和管理。为了便于主存储器与外存储器之间信息的交换，将整个存储空间分为若干块，块是主存储器与外存储器之间数据传送的基本单位，虚拟存储器的工作过程包括 6 个步骤，如图 4.42 所示。

①中央处理器将访问主存储器的逻辑地址分解成块号 a 和块内地址 b，并对块号 a 进行地址变换，即将逻辑块号 a 作为索引，查地址变换表，以确定该组信息是否存放在主存储器内。

②如该块号已在主存储器内，则转而执行④；如果该块号不在主存储器内，则检查主存储器中是否有空闲区，如果没有，便将某个暂时不用的块调出送往外存储器，以便将这组信息调入主存储器。

③从外存储器读出所要的块，并送到主存储器空闲区，然后将那个空闲的物理块号 a′和逻辑块号 a 登录在地址变换表中。

④从地址变换表读出与逻辑块号 a 对应的物理组号 a′。

⑤从物理块号 a′和块内字节地址 b′得到物理地址。

⑥根据物理地址从主存储器中存取必要的信息。

4.6.2 虚拟存储器管理技术

虚拟存储技术，实际上就是在程序运行时随时进行将编写程序时所用的虚拟地址（逻辑地址）变换成较小的物理地址。根据虚拟存储器地址格式及对虚拟存储器管理方式的不同，程序块可以具体化为段、页和段页三种形式。这三种形式在 80486 中相应地形成了段式存储器、页式存储器和段页式存储器。

虚拟存储器一般采用二维或三维的复合地址格式。当采用二维地址格式时，将整个存储器划分为若干个页（或段），每个页（或段）又包括若干存储单元。当采用三维地址格式时将整个存储空间分为若干段，每段分为若干页，每页又包括若干存储单元。

1. 分段存储管理

分段存储管理的基本原理是：按程序的逻辑结构，以段为单位划分，各个段的长度因程序而异。为了说明逻辑段的各种属性，系统为每一个段建立一个段表（驻留在内存），记录段的若干信息，如段号、段起点、段长度和段装入情况等。CPU 通过访问段表，判断该段是否已调入主存储器，并完成逻辑地址与物理地址之间的转换。分段地址转换如图 4.43 所示。

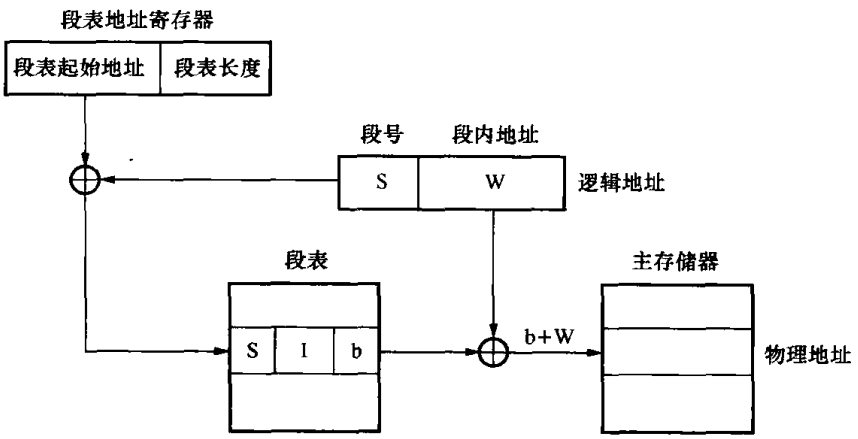


图 4.43 分段地址转换

逻辑地址由段号 S 和段内地址 W 组成，段号 S 相当于逻辑段的段名，它表示该逻辑段的起始地址。在进行地址转换时，操作系统用 S 检索段表，段表中记录的信息 1 表明该段已

调入主存储器，b 是 S 段装入主存储器的起始地址，因此该逻辑地址对应的物理地址为 b+W。

在分段存储管理方式中，由于段的分界与程序的自然分界相对应，所以具有逻辑独立性，易于程序的编译、管理、修改和保护，也便于多道程序共享。但是，因为段的长度参差不齐，起点和终点不定，给主存储器空间分配带来了麻烦，容易在段间留下不能利用的“零头”，造成浪费。

2. 分页存储管理

分页存储管理的基本原理是：将主存储器空间和外存储器空间分别等分为大小相等的若干页，页的大小为 2^n 个字节，如 2^{10}（1KB）、2^{11}（2KB）、2^{12}（4KB）等，并且为每个页按顺序指定一个页号，即 0 页、1 页、2 页、…。为了叙述方便，把主存储器的页（物理页或绝对页）称为页面。例如，若主存储器空间为 8KB，外存储器空间为 16KB，页的大小为 1KB，则主存储器空间可分为 8 个页面，其页面号为 0～7；外存储器空间可分为 16 个页，其页号为 0～15，如图 4.44 所示。当程序运行时，以“页”为单位进行地址映像，即操作系统以页为单位把逻辑页从外存储器调入主存储器，存放在物理页面上，供 CPU 执行。在分页存储管理机制中，把逻辑页对应的逻辑地址称为线性地址。

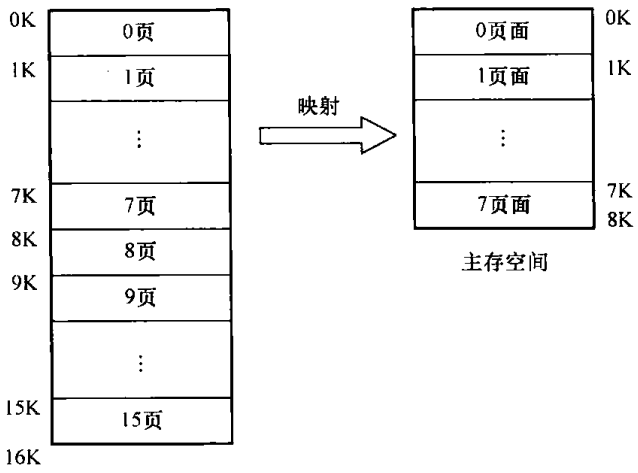


图 4.44 主存储器与辅存储器分页结构

在分页存储管理中，需要解决的关键问题是分页存储管理的地址转换：选择哪一个物理页存放调入的逻辑页？如何将线性地址转换为物理地址？为了解决这些问题，系统为每一个页建立一个页表，保存在主存储器中，存放页的若干信息，如页号、容量、是否装入主存储器、存放在主存储器的哪一个页面上等。

CPU 访问某页时，首先要查找页表，判断要访问的页是否在主存储器，若在主存储器为命中，否则为未命中。然后将未命中的页按照某种调度算法由外存储器调入主存储器，并根据逻辑页号和存放的物理页面号的对应关系，将线性地址转换为物理地址。分页存储管理的地址转换如图 4.45 所示。

页表在存储器中的位置由页表地址寄存器定位。图 4.45 的页表中记录的状态信息为：第 1 项页号指示逻辑页；第 2 项特征位记录该逻辑页是否装入主存储器，“0”表示未装入，“1”表示已装入；第 3 项记录该逻辑页装入主存储器所存放的物理页的页面号，即 1 号逻辑页从外存储器调入主存储器后，存放在第 7 号物理页面上。

存储单元的物理地址由页面号和页内地址两部分组成，如图 4.46（a）所示。8KB（$2^{13}\times 8$）主存储器的页内地址由地址线 A_9～A_0 提供，可寻址 1KB 的页内存储空间，地址范围为0～1023；页面号由高 3 位地址线 A_{12}～A_{10}提供，8 个页面的页面号为 0～7。由此可以看出，页面存储单元的物理地址为：

物理地址＝页的大小×页面号＋页内地址

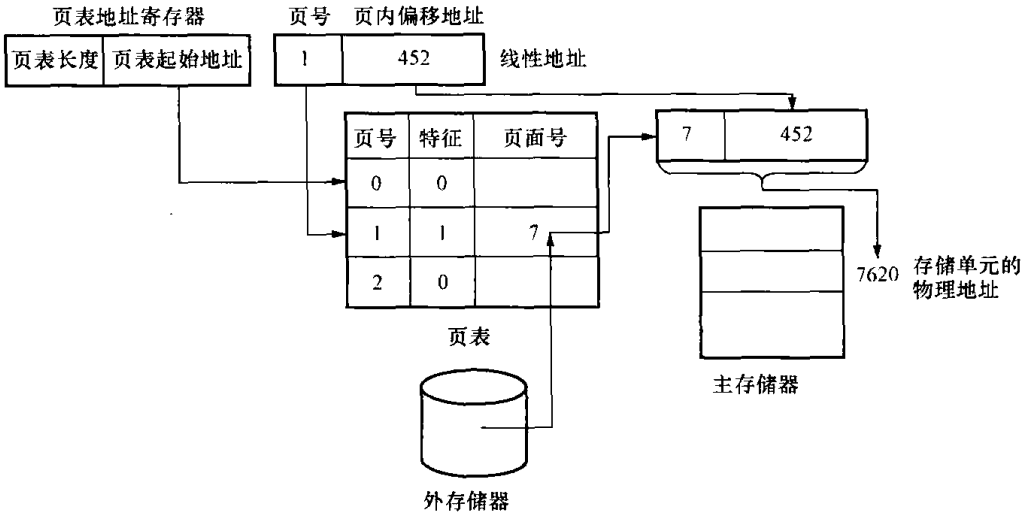


图 4.45 分页存储管理的地址转换

线性地址的确定方法与物理地址的确定方法完全相同，也是由页号和页内地址两部分组成，如图 4.46（b）所示。16KB 外存储器空间可分为 16 页（每页 1K 空间），页内地址由地址线 $A_9 \sim A_0$ 提供，其页号 0～15 由高 4 位地址线 $A_{13} \sim A_{10}$ 提供。

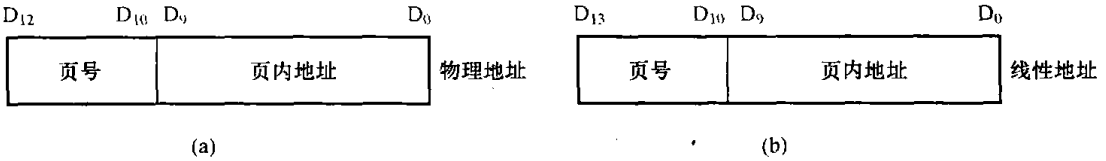


图 4.46 物理地址与线性地址结构

（a）物理地址结构；（b）线性地址结构

在进行地址转换时，由于逻辑页和物理页的大小相等，它们的页内地址是相同的，所不同的是页号，只要将线性地址的页号转换为物理地址的页面号即可。在图 4.45 中，给出 1 号逻辑页中某条指令访问数据的逻辑地址为 1024×1＋452＝1476，它存入主存储器 7 页面上所对应的物理地址为 1024×7＋452＝7620。

3. 段页存储管理

段页存储管理方式综合了段式管理和页式管理的优点，但需要经过两级查表才能完成地址转换，消耗时间多。分页存储管理的主要特点是主存储器利用率高且对外存储器管理容易，但模块化性能差；分段存储管理的主要特点是模块化性能好，但主存储器利用率不高且对外存储器的管理比较困难。

段页存储管理首先将程序按其逻辑结构划分为若干个大小不等的逻辑段，然后再将每个逻辑段划分为若干个大小相等的逻辑页。主存储器空间也划分为若干个同样大小的物理页。外存储器和主存储器之间的信息调度以页为基本传送单位，每个程序段对应一个段表，每页对应一个页表。CPU 访问时，段表指示每段对应的页表地址，每一段的页表确定页所在的主存储器空间的位置，最后与页表内地址拼接，确定 CPU 要访问单元的物理地址。段页地址转换如图 4.47 所示。

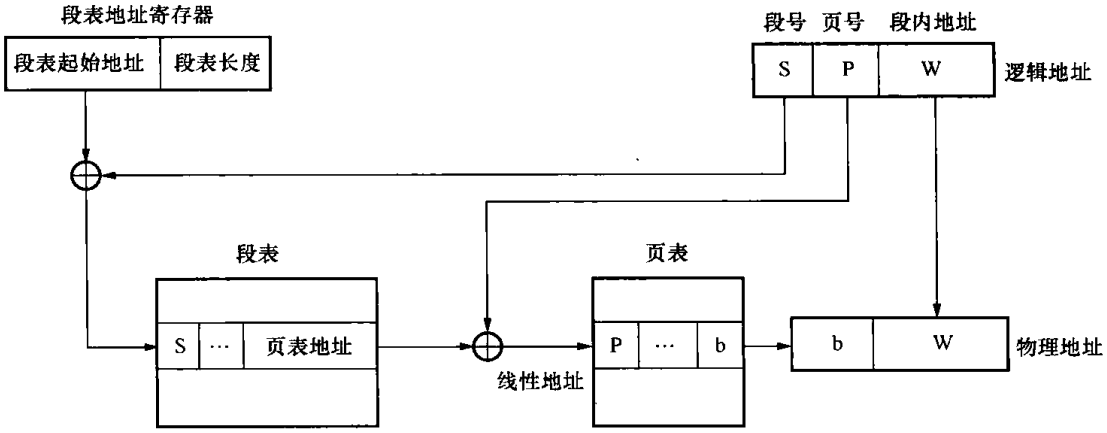


图 4.47 段页地址转换

4.7 外 存 储 器

外存储器通常是磁性介质或光盘，像硬盘、软盘、磁带、CD 等，容量大，是用来长期保存程序和数据的存储器。

4.7.1 硬盘存储器

硬盘存储器是一种采用磁介质的数据存储设备，数据存储在密封于洁净的硬盘驱动器内腔的若干个涂有磁性材料盘片上，厚度一般在 0.5mm 左右，磁盘材料硬度和耐磨性要求很高，多数为铝合金材料。硬盘存储器主要由盘体、控制电路板和接口部件组成。控制电路板上主要有硬盘 BIOS、硬盘缓存（Cache）和主控制芯片等单元。硬盘接口包括插座、数据接口和主、从跳线等。

1. *硬盘存储器物理结构*

硬盘盘体是完全密封的，里面主要有磁头、盘片等部件。有些硬盘只装一张盘片，有些则有多张。硬盘盘片安装在主轴电机的转轴上，在主轴电机的带动下作高速旋转。每张盘片的容量称为单碟容量，而一块硬盘的总容量就是所有盘片容量的总和。早期硬盘由于单碟容量低，所以盘片较多。现代的硬盘盘片一般只有少数几片。一块硬盘内的所有盘片都是完全一样的，盘片上的记录密度很大，而且盘片工作时会高速旋转。硬盘采用高精度、轻型磁头驱动和定位系统。这种系统能使磁头在盘面上快速移动，读写硬盘时，磁头依靠磁盘的高速旋转引起的空气动力效应悬浮在盘面上，与盘面的距离不到 1μm，可以在极短的时间内精确定位到计算机指令指定的磁道上。盘片的转速与盘片大小有关，考虑到惯性及盘片稳定性，盘片越大转速越低。

磁头用来读取或者修改盘片上磁性物质的状态，一般说来，每一个磁面都会有一个磁头，从最上面起由 0 开始编号。磁头在停止工作时与磁盘是接触的，停放在启停区［离主轴最近的，即线速度最小的地方，是一个特殊的区域，它不存放任何数据，称为启停区或着陆区（Landing Zone），启停区外是数据区］，但是在工作时呈飞行状态。读取数据时盘片高速旋转，由于空气的“浮力”作用，此时磁头离盘面数据区 0.2～0.5μm 的高度进行飞行。这样既不与盘面接触，又能很好地读取数据。

2. 硬盘存储器的逻辑结构

硬盘存储器盘片一般是在以铝为主要成分的片基表面涂上磁性介质所形成，在磁盘片的每一面上，以转动轴为轴心、以一定的磁密度为间隔的若干个同心圆就被划分成磁道（Track），每个磁道又被划分为若干个扇区（Sector），数据就按扇区存放在硬盘上。在每一面上都相应地有一个读写磁头（Head），所以不同磁头的所有相同位置的磁道就构成了所谓的柱面（Cylinder）。传统的硬盘读写都是以柱面、磁头、扇区为寻址方式的（CHS 寻址）。硬盘在上电后保持高速旋转（5400r/min 以上），位于磁头臂上的磁头悬浮在磁盘表面，可以通过步进电机在不同柱面之间移动，对不同的柱面进行读写。所以在上电期间如果硬盘受到剧烈振荡，磁盘表面就容易被划伤，磁头也容易损坏，这都将给盘上存储的数据带来灾难性的后果。

我们大家知道，硬盘有面、柱面、磁道和扇区等概念，这些指的是硬盘存储器的逻辑结构。计算一个磁盘容量的公式是：磁盘存储容量＝盘面数×每盘面磁道数×每磁道扇区数×每扇区存储容量。

面（盘面）：上面已经了解，硬盘一般会有一个或多个盘片，每个盘片可以有两个面（Side），依次可称为 0 面、1 面等。每个面对应一个磁头，用于读写数据。

磁道：盘片在中心轴带动下做高速旋转，磁头连续写入的数据是排列在一个圆周上的，我们称这样的圆周为一个磁道（Track）。可以假想有很多个同心圆，大小不同的磁道组成盘面，每个盘片的每个面都有一个读写磁头。磁道在最外圈，离主轴最远的地方是“0”磁道，硬盘数据的存放就是从最外圈开始的。硬盘有一个“0”磁道检测器，由它来完成硬盘的初始定位。“0”磁道十分重要，如果“0”磁道损坏，则该硬盘不可修复。

柱面：我们认为离圆心最远的磁道为 0 磁道，依次往里为 1 磁道……。不同面上相同磁道编号则组成了一个圆柱面，即所称的柱面（Cylinder）。每个柱面上的磁头由上而下从“0”开始编号。数据读写按柱面进行，即磁头读写数据时首先在同一柱面内从“0”磁头开始进行操作，依次向下在同一柱面的不同盘面（即磁头上）进行操作，只有在同一柱面所有的磁头全部读写完毕后磁头才转移到下一柱面。

扇区：把磁道划分成若干段，每段称为一个扇区（Sector）。扇区是硬盘上存储的物理单位，包括 512 个字节的数据和一些其他信息。即使计算机只需要硬盘上存储的某个字节，也须一次把这个字节所在的扇区中的全部 512 字节读入内存，再选择所需的那个字节。扇区数据主要有存储数据的标识符和存储数据两个部分。标识符包括扇区所在的磁头（或盘面）、磁道（或柱面号）以及扇区在磁道上位置（即扇区编号）。标识符还包括一个字段，显示扇区是否可以存储数据或者已有故障不宜使用的标记。有些硬盘控制器在扇区标志符中还记录提示信息，可在原扇区出错时指引磁盘转到对应替换扇区或磁道。最后，标识符以循环冗余校验（CRC）值作为结束，以供控制器检验扇区标识符读出情况，确保准确无误。扇区的第二个主要部分是存储数据的数据段，可分为数据和保护数据的纠错码（ECC）。

簇：我们上边已经了解扇区是实际物理单位，簇就是硬盘上存储文件的一个逻辑单位。物理相邻的若干个扇区其实就组成了一个簇。操作系统读写磁盘的基本单位是扇区，而文件系统的基本单位是簇（Cluster）。

3. 硬盘数据结构

硬盘的第一个扇区为主引导扇区。在主引导区内主要有主引导记录和硬盘分区表两项内

容。主引导记录是一段程序代码，其作用主要是对硬盘上安装的操作系统进行引导；硬盘分区表则存储了硬盘的分区信息。计算机启动时将读取该扇区的数据，并对其合法性进行判断（扇区最后两个字节是否为 0X55AA 或 0XAA55），如合法则跳转执行该扇区的第一条指令。所以硬盘的主引导区常常成为病毒攻击的对象，从而被篡改甚至被破坏。

初买来一块硬盘，需要将它分区、格式化，然后再安装上操作系统才可以使用。一般要将硬盘分成主引导扇区、操作系统引导扇区、FAT、DIR 和 DATA 等五部分（其中只有主引导扇区是唯一的，其他的随分区数的增加而增加）。

4.7.2 光盘存储器

所谓光盘（CD）是利用光学原理读写信息的存储器。由于光盘的容量大、速度较快、不易受干扰等特点，光盘的应用愈来愈广泛。

光盘系统一般是由光学、电气和机械部件组成。从结构上看，光盘存储器同磁盘存储器基本相同，两者均有存储信息的盘片、机械驱动部件、定位部件和读写机构。不同的是，后者利用磁性原理存储信息，利用磁头存取信息；而前者是利用光学原理存储信息，并用光学读写头来存取这些信息。

光盘本身是靠盘面上一些能够影响光线反射的表面特征存储信息，例如现在常用的只读光盘（CD-ROM）上利用光盘表面的凹凸不平表示“0”和“1”。以 CD-ROM 为例，读取数据时，由机械驱动部件和定位部件负责确定读取的位置。激光器发出激光经光学线路至聚焦透镜射向光盘表面，表面的凹凸不平造成反射光的变化，利用数据光检测器将这些变化转换为数据“0”和“1”的电信号传输到数据输出端，整个读取工作完成。其他类型光盘的写入过程大体与此相同，唯一的差别是数据自数据输入端传来。

一般将光盘存储器分为只读式（Read Only）、一次写入式（Write Once）和可擦式（Erasable）或可逆式（Reversible）三种。只读式光盘利用材料表面的凹凸不平的特征记录信息，在出厂前由生产厂家将有关信息存放到光盘上。对于一次写入式光盘，用户可以利用汇聚的激光束在光盘表面照射使材料发生永久性变化而记录信息。这种光盘现已普遍用于多媒体系统。可擦式光盘利用激光在磁性材料上或相变材料上实现信息的存储和擦除。

光盘存储器的记录密度高，存储容量大，一片 5.25 英寸大小的一次写入式光盘可以存储 680MB 的信息，其容量远远大于外形同样大小的软磁盘。光盘信息的保存时间也比磁盘的长。目前影响光盘普遍应用的主要原因是光盘存储器的读写速度慢和光盘驱动器的成本高。随着技术的进步，以上问题是可以解决的，因此光盘存储器有广泛的应用前景。

习　　题

4.1　分析半导体随机存储器和只读存储器的特点和分类。静态存储器和动态存储器的最大区别是什么？它们各有什么优缺点？

4.2　说明字扩展和位扩展的实现方法。

4.3　存储器的寻址范围是怎样确定的？举例说明它的确定方法。

4.4　设计 RAM 存储器系统，起始地址为 20000H，容量为 512K。使用你熟悉的芯片完成之。

4.5　用下列芯片构成存储系统，各需要多少个 RAM 芯片？需要多少位地址作为片外

地址译码？设系统为20位地址线，采用全译码方式。

（1）512×4位RAM构成16KB的存储系统；

（2）1024×1位RAM构成128KB的存储系统；

（3）2K×4位RAM构成64KB的存储系统；

（4）64K×1位RAM构成256KB的存储系统。

4.6 现有一种存储芯片容量为512×4位，若要用它组成4KB的存储容量，需多少这样的存储芯片？4KB存储系统最少需多少寻址线？每块芯片需多少寻址线？

4.7 有一个6264SRAM芯片的译码电路，如图4.48所示，请计算该芯片的地址范围及存储容量。

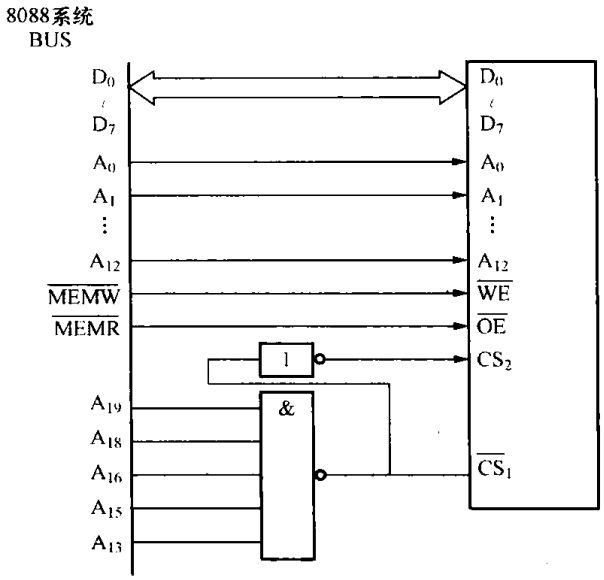


图4.48 习题4.7图

4.8 使用4K×8位的EPROM2732和2K×8位的静态RAM6116以及LS138译码器，构成一个存储容量为16KB ROM（00000H-03FFFH）、8KB RAM（03000H～04FFFH）的存储系统。系统地址总线为20位，数据总线为8位。请画出系统连接图。

4.9 使用4K×8位的EPROM2732和2K×8位的静态RAM6116以及LS138译码器，构成一个8KB的ROM，地址范围为FE000H～FFFFFH；4KB的RAM存储器系统，地址范围为00000H～00FFFH。系统CPU8086工作于最小模式。请画出系统连接图。

4.10 试为某8位微机系统设计一个具有8KB ROM和40KB RAM的存储器。

（1）选用EPROM芯片2716组成只读存储器（ROM），从0000H地址开始；

（2）选用SRAM芯片6264组成随机存取存储器（RAM），从4000H地址开始；

（3）分析出每个存储芯片的地址范围。

4.11 什么是Cache？它能够提高计算机处理能力是基于什么原理？

4.12 现代计算机中设置高速缓冲存储器与虚拟存储器的目的是什么？

第5章 输入输出技术

到此我们已经学习了计算机系统的中央处理器（CPU）、存储器、指令系统和汇编程序设计，它们分别用来处理信息、存储信息和描述信息处理过程。那么信息是怎么进入计算机系统的呢？这就靠输入输出技术来解决。计算机所处理的各种信息，通过输入设备输入，而处理结果通过输出设备输出人们才能查看。本章主要介绍输入/输出（I/O）接口的基本概念、输入/输出方法、简单接口电路和DMA控制器8237。

5.1 I/O 接 口

5.1.1 I/O接口概述

微型计算机还需配上各种外部设备，才能使微型计算机进行信息的输入输出。计算机所配置的外部设备（或称输入输出设备或称外围设备）多种多样，常见的有键盘、显示器、软/硬盘驱动器、鼠标、打印机、扫描仪、绘图仪、调制解调器（MODEM）、网络适配器、影视、音频识别系统、模数转换器（ADC）和数模转换器（DAC）等。

通常，在计算机系统把处理器和主存储器以外的部分称为输入/输出系统，包括输入输出设备、输入输出接口和输入输出软件。微型计算机上的所有部件都是通过总线互联的，外部设备也不例外。通常，I/O接口是将外设连接到系统总线上的一组逻辑电路的总称，如图5.1所示。对于主机，接口提供了外部设备的工作状态及数据；对于外部设备，接口电路记忆了主机下达给外设的命令和数据，从而使主机与外设之间能协调一致地工作。

为什么主机与外设间交换信息要通过接口，接口应具备哪些功能呢？

我们知道计算机只能处理0、1的数字信息，而外设种类很多，CPU无法对各种类型的外设提供的信息进行直接处理，它们之间存在以下问题：

（1）速度匹配：CPU速度越来越快，而不同的外设速度有高有低且差异很大。

（2）信号的驱动能力：CPU的信号一般是TTL电平且功率也小。而外设需要的电平宽，驱动功率也较大。

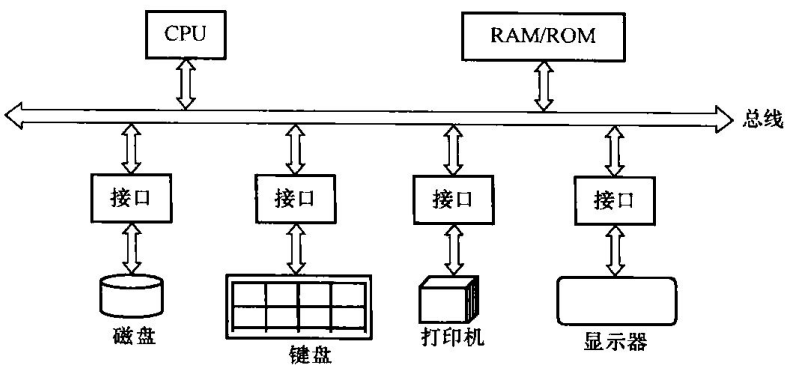


图5.1 主机通过接口与外设相连

（3）信号形式的匹配：CPU只能处理数字信号，而外设信号可能是数字量、开关量、模拟量（电流、电压、频率、相位）、压力、流量、温度、速度等。

（4）信息格式：CPU处理的信息一般是8、16、32位并行二进制数据，而不同外设使用不同形式的信息格式，如字节流、

块、数据包、帧、并行数据、串行数据等。

(5) 时序匹配：CPU 的各种操作都规定了自己的总线周期，是在统一的时钟信号作用下完成的。而外设有自己的定时与控制逻辑，与 CPU 的时序不一致。

这些问题是通过 CPU 与外设之间设置相应的 I/O 接口来解决的。因此，I/O 接口应具有以下基本功能：

(1) 命令、数据和状态的缓冲、隔离和锁存。外设种类繁多而不能直接和 CPU 总线相连，要借助于接口电路使外设与总线隔离（高阻状态），用于缓冲或锁存数据、命令和状态的作用，以确保计算机和外设之间可靠地进行信息传送。一般外设经接口与总线相连，其连接方法必须遵循“输入要三态、输出要锁存”的原则。

(2) 信息形式和格式转换。CPU 只能处理数字信息，其电平一般在 0～5V 之间。而外部设备的信息形式和格式多种多样。所以，I/O 接口应能实现信息格式变换、电平转换、码制转换、传送管理以及联络控制等。

(3) 信息的输入输出。通过 I/O 接口，CPU 可以从外部设备输入各种信息，也可将处理结果输出到外设；CPU 可以控制 I/O 接口的工作，还可以随时监测与管理 I/O 接口与外设的工作状态；必要时，I/O 接口还可以通过接口向 CPU 发出中断请求。

(4) 根据寻址信息选择外设。一个计算机系统往往有多种外部设备，CPU 在某一时刻只能与一台外设交换信息，因此需要通过接口的地址译码对外设进行寻址，以选定所需外设，只有选中的设备才能与 CPU 交换信息，而未被选中的 I/O 接口呈现为高阻隔离状态，与总线隔离；当同时有多个外设需要与 CPU 交换数据时，由 I/O 接口来安排其优先顺序。

5.1.2 I/O 端口的编址方式

在接口电路中，为了完成上述功能，CPU 与 I/O 接口进行通信是通过 I/O 接口内部的一组寄存器来实现的，这些寄存器统称为 I/O 端口。有数据端口、状态端口、控制（或命令）端口三种类型。根据需要，不同外设具有端口数各不相同。CPU 通过数据端口从外设读入数据或向外设输出数据；从外设的状态端口读入其当前状态；通过控制端口向外设发出控制命令。

8086/8088CPU 最多能管理 64K 个端口，那么当前的操作是针对哪一个端口呢？就要像为内存单元分配地址那样为每一个端口赋予一个唯一编号——称为端口地址（或端口号）。因为一个外设总是对应着一个或多个端口分别传送数据、状态、命令信息，所以有时将端口地址称为外设地址。当一个外设有多个端口时，通常为其分配一个连续的地址块，这个地址块中最小的那个地址称为（外设的）基地址。

在微型机系统中，I/O 端口的编址通常有两种方式：I/O 端口与内存统一编址；I/O端口与内存独立编址。

1. I/O 端口与内存统一编址

这种编址方式也称为存储器映射编址方式，它把内存的一部分地址分配给 I/O 端口，即端口与存储器单元在同一个地址空间中进行编址。已经用于 I/O 端口的地址，存储器不能再使用。例如，将内存地址的最后 64K 空间（F0000H～FFFFFH）分配给外设。

该编址方式的优点是：访问内存储器单元和 I/O 端口使用相同指令，且有相同的控制

信号，有助于降低 CPU 电路的复杂性，并给使用者提供方便。但其缺点是：I/O 端口占用内存地址，减少了内存可用的地址范围；难以区分当前是访问内存还是外设。

2. I/O 端口与内存独立编址

I/O 端口与内存独立编址是内存储器和 I/O 端口各自有自己独立的地址空间。访问 I/O 端口需要专门的 I/O 指令。例如，8086/8088 CPU 就采用这种方式。访问内存储器使用 20 根地址线 A_0～A_{19}，同时使 8088 CPU 的存储器与 I/O 选择信号 IO/$\overline{M}$=0（8086 是 M/$\overline{IO}$=1），内存地址范围为 00000H～FFFFFH 共 1MB；访问 I/O 端口时使用低 16 根地址线 A_0～A_{15}，同时使 8088 CPU 的存储器与 I/O 选择信号 IO/$\overline{M}$=1（8086 是 M/$\overline{IO}$=0），I/O 端口地址范围为 0000H～FFFFH。两个地址空间相互独立，互不影响。

8086/8088 CPU 中，I/O 端口寻址方式有直接寻址和间接寻址，直接寻址方式的地址范围为 0～0FFH，间接寻址方式的地址范围为 0～0FFFFH。输入输出指令是 IN 和 OUT，具体格式及功能详见第 3 章的介绍。

可见，I/O 端口独立编址方式有如下特点：

（1）I/O 端口的地址空间与内存地址空间完全独立。

（2）I/O 端口与内存使用不同的控制信号。

（3）指令系统中设有专门用于访问外设的 I/O 指令。

另外，微型计算机 I/O 端口地址分配一般分为两部分：主板上寻址 I/O 接口芯片用端口地址；供扩展槽接口卡用端口地址。具体分配方案可参见厂家提供的随计算机或电路板卡的资料。接口电路可通过“跳线器（JUMPER）”（一组开关）在几组可选的地址内选定其端口地址。

5.2 简单接口电路

5.2.1 接口电路的基本结构

CPU 与外设交换的信息有数据信息、状态信息和控制信息三类。

1. 数据信息

数据信息主要指键盘、磁盘、扫描仪输入设备及过程通道读入的信息；CPU 输出至打印机、显示器、绘图仪等输出设备及过程通道的信息。

2. 状态信息

状态信息反映外部设备工作状态的信息。如输入时，CPU 先检测外设欲输入的信息是否准备就绪，如果是，则 CPU 可以读入信息，否则 CPU 等待“就绪”信号的出现后再读入；输出时，CPU 先检测外设是否已处于“空闲”状态，若是，则 CPU 输出数据至外设。若外设处于“忙”状态，CPU 不能向外设输出信息。这种“空闲”、“忙”、“就绪”均为状态信息。

3. 控制信息

控制信息主要指 CPU 向接口内部控制寄存器发出的各种控制命令，用于设置接口的工作方式及功能，使接口与不同的外设协调工作，扩大接口的应用范围。

通用 I/O 接口基本结构框图如图 5.2 所示。通常是一块中、大规模或超大规模集成电路芯片，常称为 I/O 接口电路芯片，简称接口芯片。它主要由以下几部分组成：

（1）数据输入寄存器和数据输出寄存器。又分别称输入端口和输出端口，合称为数据端口，用来暂存 CPU 和内存送往外设的数据或外设送往 CPU 和内存的数据，在 CPU 与外设间进行数据传送时起桥梁作用。输入端口要求具有三态输出能力，输出端口常用锁存器实现。

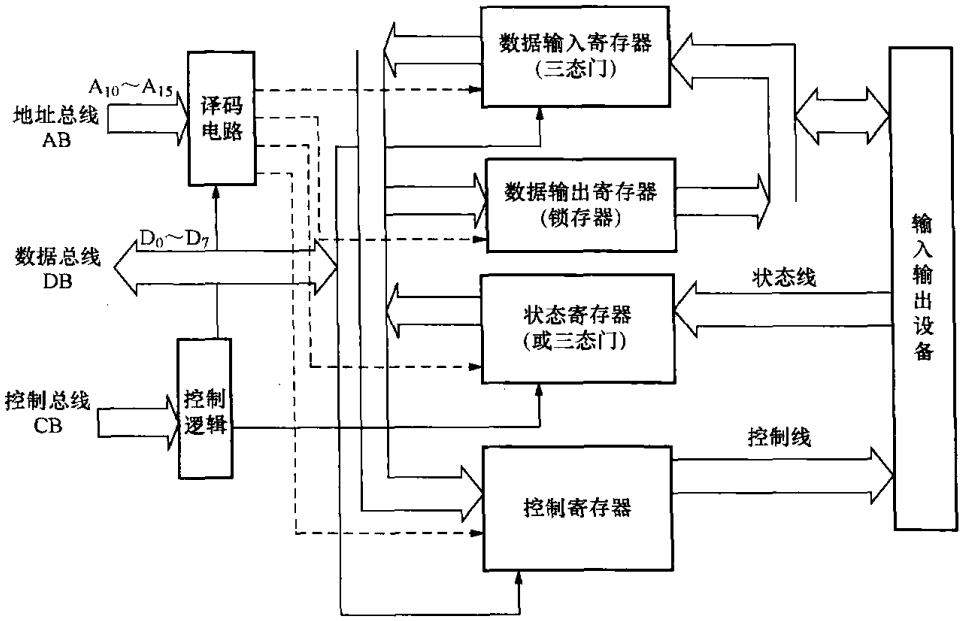


图 5.2 I/O 接口基本结构框图

（2）控制寄存器。又称命令端口或控制端口，用来存放 CPU 发出的控制命令（信息），以控制接口的工作方式及功能和控制外设按命令要求动作。

（3）状态寄存器。又称状态端口，用于存放外设或接口本身的状态信息。CPU 可以通过读状态端口来检测外设或接口当前的工作状态。此端口一般是只读寄存器。

（4）译码电路。CPU 在执行输入输出指令时，向地址总线发送 16 位外部设备的端口地址。由片选信号是否有效判定 CPU 是否选中本接口芯片，若选中，则译码电路产生相应的选通信号，使相关端口的寄存器/缓冲器进行数据、命令或状态的传输，完成一次 I/O 操作。通常把 16 位地址码分解为高位地址码用作对接口的选择，低位地址码用来选择接口内不同的端口，实现 CPU 与此端口的信息传输。

（5）控制逻辑。用于产生接口的内部控制信号和对外控制信号，以实现处理器和外设间相互协调的输入/输出操作。

不是所有接口都具备上述全部功能。接口需要哪些功能取决于 I/O 设备的特点，通常有以下几类接口：

1）按数据传送方式分，可分为并行接口和串行接口。

2）按功能选择的灵活性分，可分为可编程接口和不可编程接口。

3）按通用性分，可分为通用接口和专用接口。

4）按数据控制方式分，可分为程序型接口和 DMA（Direct Memory Access）型接口。

近年来，由于大规模集成电路和计算机技术的发展，I/O 接口电路大多采用大规模、超大规模集成电路，并向智能化、系列化和一体化方向发展。但它们基本上是一些功能单一的

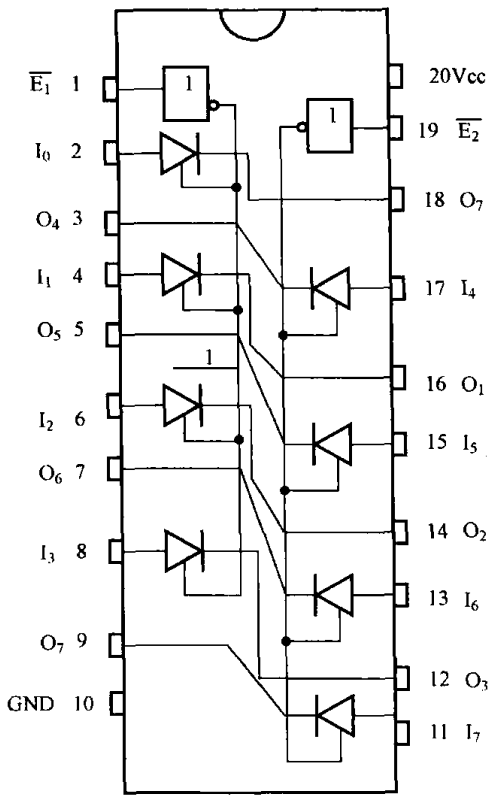


图 5.3 74LS244 芯片引脚图

接口电路的组合与集成。为此，下面介绍一些结构简单又较为常用的通用接口芯片。

5.2.2 三态门接口

典型的三态门芯片 74LS244 如图 5.3 所示。该芯片由 8 个三态门构成。74LS244 有$\overline{E_1}$和$\overline{E_2}$两个控制端。每个控制端各控制 4 个三态门。当某一控制端有效（低电平）时，相应的 4 个三态门导通；否则，相应的三态门呈现高阻状态（断开）。实际使用中，通常是将两个控制端并联，这样就可用一个控制信号来使 8 个三态门同时导通或断开。

由于三态门具有“通断”控制能力的缺点，故可利用其作输入接口。利用三态门作为输入信号接口时，要求信号的状态是能够保持的。这是因为三态门本身没有对信号的保持或锁存能力。图 5.4 所示为一个利用三态门 74LS244 作为开关量输入接口的例子。图中，74LS244 的输入端接有 8 个开关 K1，K2，…，K8。当 CPU 读该接口时，总线上的 16 位地址信号通过译码使$\overline{E_1}$和$\overline{E_2}$有效，三态门导通，8 个开关的状态经数据线 D_0～D_7 被读入到 CPU 中。这样，就可测量出这些开关当前的状态是打开还是闭合。当 CPU 不读此接口地址时，$\overline{E_1}$和$\overline{E_2}$为高电平，则三态门的输出为高阻状态，使其与数据总线断开。

用一片 74LS244 芯片作为输入接口，最多可以连接 8 个开关或其他具有信号保持能力的外设。当然也可只读一个外设而让其他端悬空，对空着未用的端，其对应位的数据是任意值，在程序中常用逻辑“与”指令将其屏蔽掉。

如果有更多的开关状态需要输入时，可用类似的方法将两片或更多的芯片并联使用。

74LS244 芯片除用作输入接口外，还常用来作为信号的驱动器。

[例 5.1] 编写程序判断图 5.4 中的开关的状态。如果所有的开关都闭合，则程序转向标号 NEXT1 的程序段执行，否则转向标号为 NEXT2 的程序段执行。

图 5.4 中，作为输入接口的三态门 74LS244，其 I/O 地址采用了部分地址译码，地址线 A_1 和 A_0 未参加译码，故它所占用的地址为 83FCH～83FFH。我们可以用其中任何一个地址，而其他重叠的 3 个地址空着不用。另外，由图 5.4 可以看出，当开关闭合时输入低电平（=0）。程序段如下：

```
MOV  DX,83FCH
IN   AL,DX
AND  AL,0FFH
JZ   NEXT1
JMP  NEXT2
```

可见，利用三态门作为输入接口，使用和连接都是很容易的。

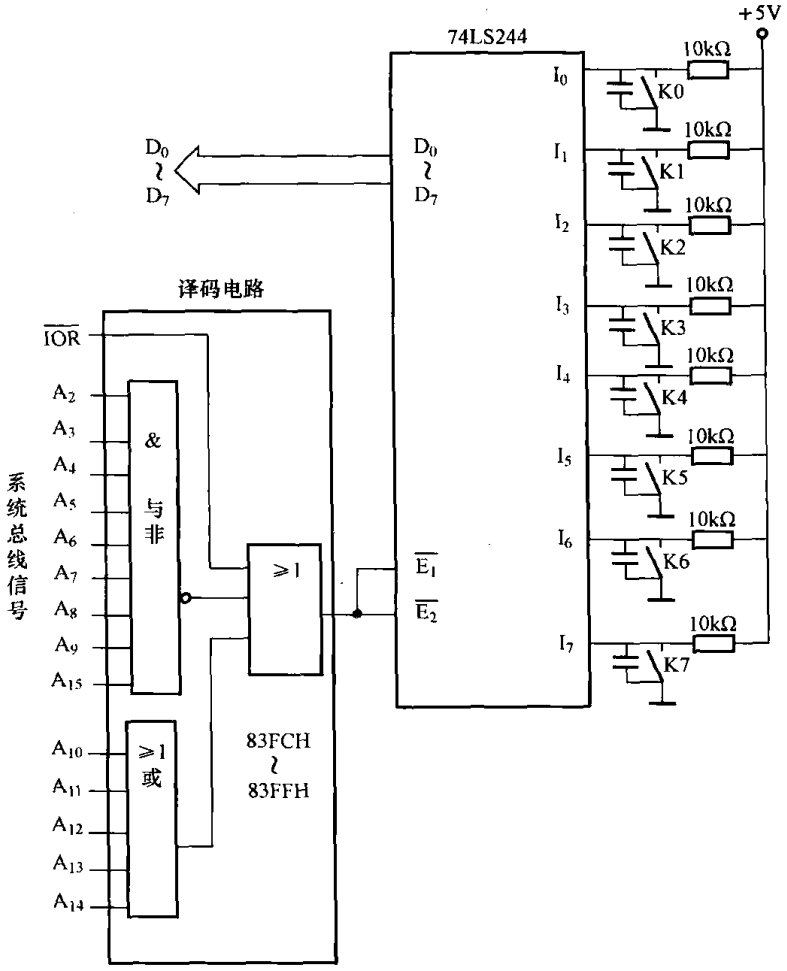
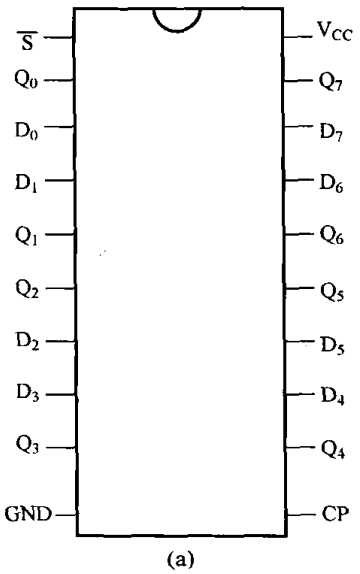


图 5.4 三态门作输入接口

5.2.3 锁存器接口

由于三态门器件不具备数据的保存（或称锁存）能力，它要求信号源能够将信号保持足够长的时间直到被 CPU 读取，所以它一般只用作输入接口，而不能直接用作数据输出接口。

数据输出接口通常采用具有信息存储能力的双隐态触发器来实现。最简单的输出接口可用 D 触发器构成。例如，常用的锁存器 74LS273，它内部包含了 8 个 D 触发器。其引脚线图及真值表如图 5.5 所示。74LS273 共有 8 个数据输入端（D_0～D_7）和 8 个数据输出端 Q_0～Q_7。S 为复位端，低电平有效。CP 为脉冲输入端，在每个脉冲的上升沿将输入端 D_i 的状态锁存在 Q_i 输出端，并将此状态保持到下一个时钟脉冲的上升沿。74LS273 常用来作为并行输出接口。另外，使用



$\overline{S}$	CP	D_i	Q_i
0	x	x	0
1	↑	1	1
1	↑	0	0

(b)

图 5.5 74LS273 引脚图和真值表

(a) 引脚图；(b) 真值表

其中的某一个 D 触发器也可通过软件编程实现简单的串行输出。

图 5.6 所示的是应用 74LS273 作为输出接口的例子。8 个 Q 端与 8 个发光二极管相连接，要使接到 Q_0 端和 Q_6 端的发光二极管发光，其对应的 Q_0 、Q_6 端须为“1”状态，而其他 Q 端则为“0”状态。假定该输出接口的地址为 0FFFFH，则程序段如下：

```
MOV  DX,0FFFFH
MOV  AL,01000001B
OUT  DX,AL
```

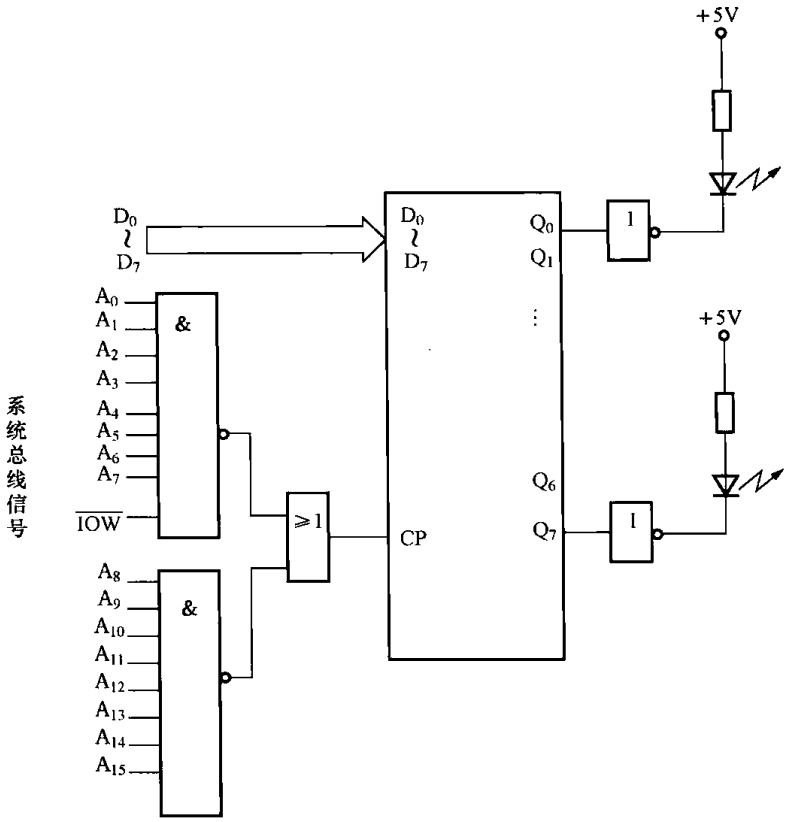
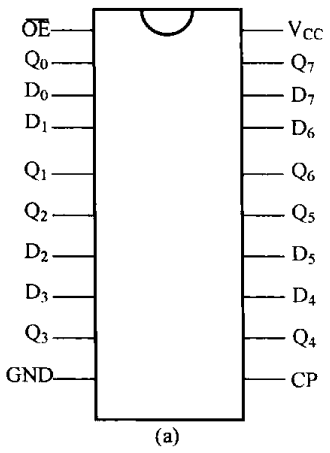


图 5.6　74LS273 作输出接口的应用

74LS273 的数据锁存输出端 Q 是通过一个一般的门（非三态门）输出的。也就是说，只要 74LS273 正常工作，其 Q 端总有一个确定的逻辑状态（0 或 1）输出。因此，74LS273 无法直接用作输入接口，即它的 Q 端绝对不允许直接与系统的数据总线相连接。那么，有没有既可作输入接口又能用作输出接口的芯片呢？下面介绍一种带有三态输出的锁存器 74LS374。其引脚图和真值表如图 5.7 所示。从引脚上可以看出，它比 74LS273 多了一个输出允许端$\overline{OE}$。只有当$\overline{OE}$=0 时，74LS374 的输出三态门才导通；$\overline{OE}$=1 时，则呈高阻状态。图 5.8 所示为 74LS374 中一个锁存器的结构图。由图可



D_i	CP	$\overline{OE}$	Q_i
1	↑	0	1
0	↑	0	0
X	X	1	高阻

(b)

图 5.7　74LS374 引脚图和真值表

（a）引脚图；（b）真值表

知，74LS374 在 D 触发器输出端加有一个三态门。

74LS374 在用作输入接口时，端口地址信号经译码电路接到$\overline{OE}$端，外设数据由外设提供的选通脉冲锁存在 74LS374 内部。当 CPU 读该接口时，译码器输出低电平，使 74LS374 的输出三态门打开，读出外设的数据；如果用作输出接口，也可将$\overline{OE}$端接地，使其输出三态门一直处于导通状态，这样就与 74LS273 一样使用了。

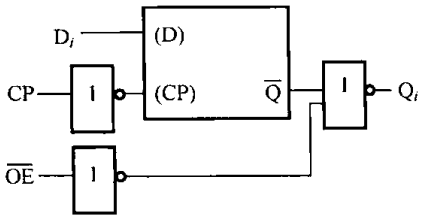


图 5.8 74LS374 中一个锁存器的结构图

分别用 74LS374 作为输入和输出接口的电路如图 5.9 所示。

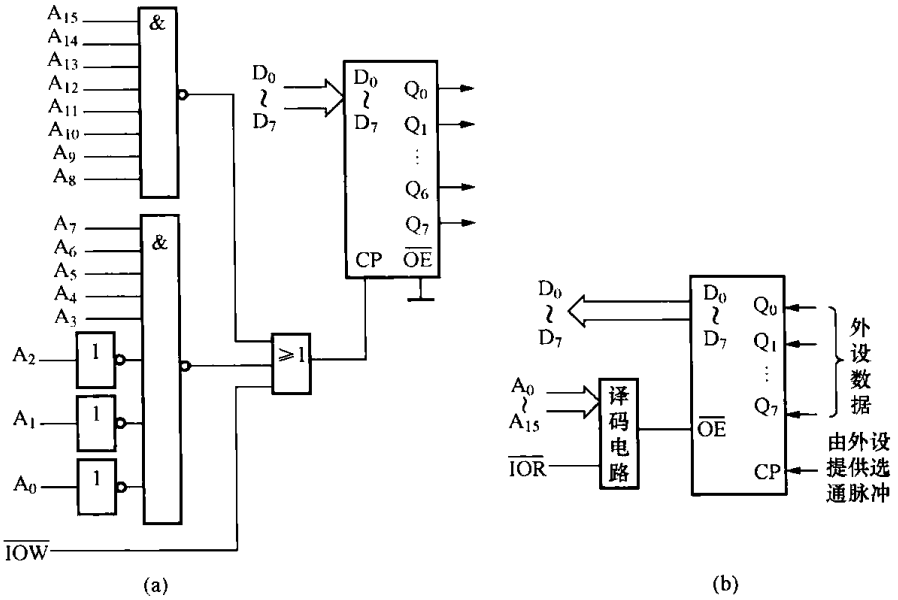


图 5.9 74LS374 用作输入和输出接口

(a) 74LS374 用作输出接口；(b) 74LS374 用作输入接口

另外还有一种常用的带有三态门的锁存器芯片 74LS373，它与 74LS374 在结构和功能上完全一样，区别是数据锁存的时机不同，带有三态门的锁存器芯片 74LS373 是在 CP 脉冲的高电平期间将数据锁存。

总之，简单接口电路芯片在构造上比较简单，使用也很方便。常作为一些功能简单的外部设备的接口电路。

5.2.4 简单接口的应用举例

下面将利用 74LS244 和 74LS273 作为输入和输出接口，通过编写程序，控制 LED 数码管显示不同的数字或符号。

1. LED 数码管

LED 数码管分为共阳极和共阴极两种结构。在封装上有将一位、二位或更多位封装在一起的。由于篇幅限制，这里只介绍一种共阳极封装的 LED 数码管，如图 5.10 所示。当某一段的发光二极管流过一定电流（如 10mA 左右）时，它所对应的段就发光，而无电流流过时，则不发光。不同发光段的组合就可显示出不同的数字和符合。显示十六进制 16 个数字的七段码如表 5.1 所示。

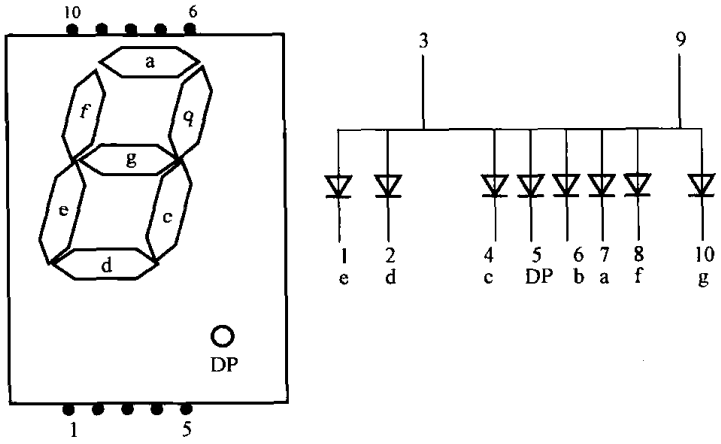


图 5.10 共阳极 LED 数码管示意图

表 5.1 十六进制数 0～F 对应的七段码值（输出端接反相器）

符号	形状	七段码·gfedcba	符号	形状	七段码·gfedcba
0	0	00111111	8	8	01111111
1	1	00000110	9	9	01100111
2	2	01011011	A	A	01110111
3	3	01001111	B	b	01111100
4	4	01100110	C	C	00111001
5	5	01101101	D	d	01011110
6	6	01111101	E	E	01111001
7	7	00000111	F	F	01110001

2. 应用与连接

七段数码管作为一种外设与系统总线有多种接口方式，这里利用前面学到的 74LS273 作为输出接口，用集电极开路门 7406 作为驱动器与 LED 数码管连接。另外。采用 74LS244 作为输入接口，输入开关的状态，线路连接如图 5.11 所示。图中的电路功能是：当开关 K 处于闭合状态时，在 LED 数码管上显示“0”，当开关 K 处于断开状态时，在 LED 数码管上显示“1”。与硬件电路相配合完成此功能的程序段如下：

```
FOREVER:  MOV   DX,0F1H        ;输入端口地址为 0F1H
          IN    AL,DX          ;读入开关状态
          TEST  AL,  1         ;判断开关状态
          MOV   AL,3FH         ;显示“0”
          JZ    DISP
```

```
        MOV  AL,06H        ;显示"1"
DISP:   MOV  DX,0F0H       ;输出端口地址为 0F0H
        OUT  DX,AL
        JMP  FOREVER
```

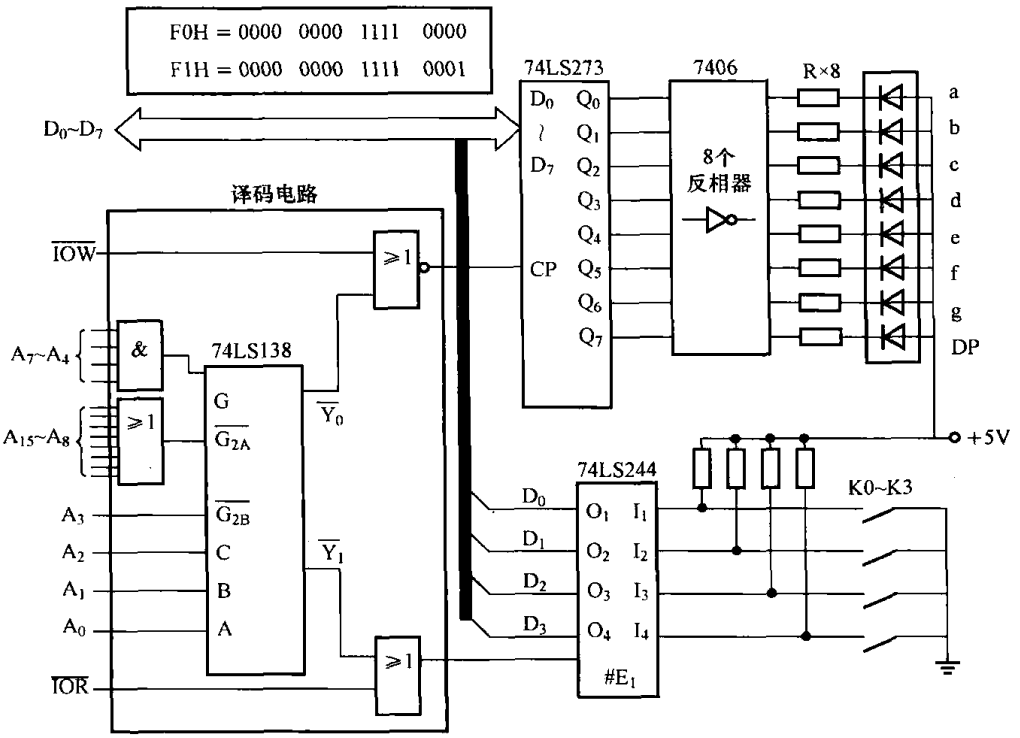


图 5.11 简单接口的应用

5.3 微机与I/O设备的信息交换方式

微型计算机与外设之间数据的输入输出，实际上是CPU、内存与外设接口之间进行的信息传送。其主要有4种方式：

（1）无条件传送方式；

（2）条件传送（查询工作）方式；

（3）中断控制传送方式；

（4）直接存储器存取（DMA）方式。

5.3.1 无条件传送方式

在程序编制中利用输入/输出指令来完成CPU与接口的信息交换。何时进行这种信息传送是事先知道的，把输入/输出指令放到程序所需要的位置。当程序执行到I/O指令时，无条件地立即执行相应操作。这种传送方式只适用于外设始终准备好接收CPU数据或CPU读数据，指令执行的过程即数据传送过程。例如，前面举过的输出代码使数码管显示的例子，数码管随时都可接收输出信号。这类简单设备的输入信号一般不需要锁存，可以通过三态缓冲器与系统数据总线直接相连。图5.12给出了无条件传送时接口的组成方式，它是5.2中介绍的几个部分的简单组合。

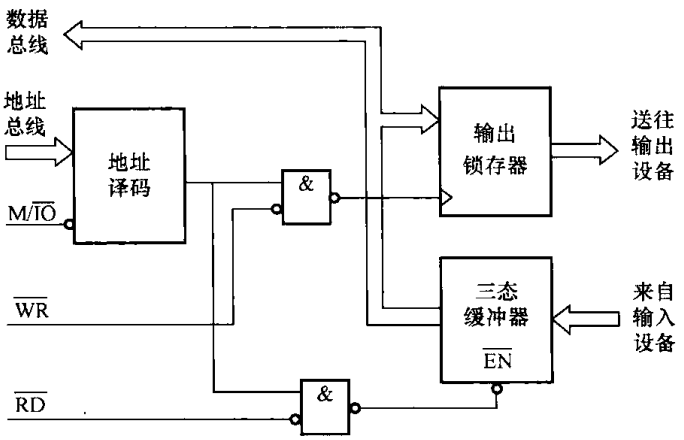


图 5.12　无条件传送接口组成方式

5.3.2　条件传送（查询工作）方式

条件传送也称为查询工作方式，CPU 通过程序不断读取并测试外设状态。如果输入设备处于准备好状态，则 CPU 执行输入指令从该设备输入数据。或者，输出设备处于空闲状态，CPU 向该设备输出数据。为此，接口电路除了有传送数据的端口以外，还应有传送状态的端口。对于输入过程来说，外设将数据准备好时，同时使接口状态端口中“准备好”标志位置 1。对于输出过程来说，外设取走一个数据后，接口便将状态端口的对应标志位清零。表示当前输出寄存器已经处于“空闲”状态，可以接收下一个数据。单个外设时有条件数据传送流程图如图 5.13 所示，多个外设时有条件数据传送流程图如图 5.14 所示。采用这种方式传送数据的工作过程描述如下：

（1）将描述外设工作状态的信息（如“准备好”）——状态字读入 CPU 相应寄存器，检测相应状态位，看收发数据是否“准备就绪”。

（2）若是，则重复执行（1），等待“准备就绪”。

（3）若不是，则执行数据传送（输入或输出）。

（4）数据传送后，CPU 向外设发响应信号，表示数据已传送。外设收到响应信号之后，即开始下一个数据的准备工作。

（5）CPU 判断是否已传送完全部数据，若没有传送完，则重新进行（1）～(4)；否则就结束传送。

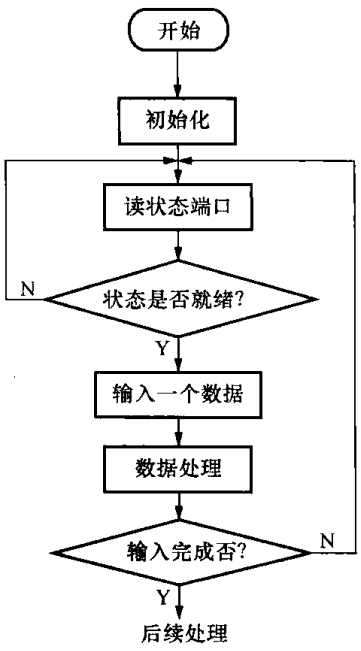


图 5.13　单个外设时有条件数据传送流程图

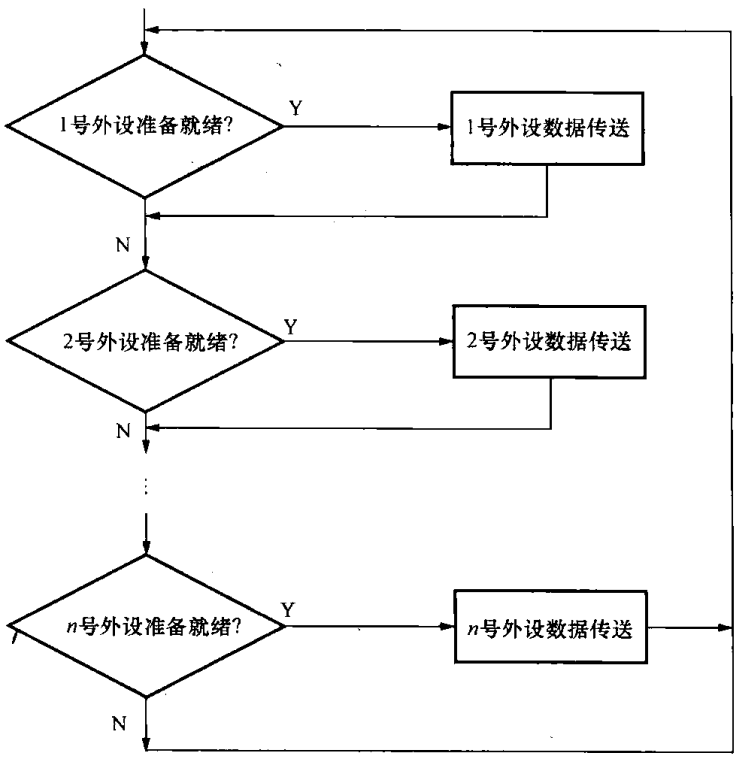


图 5.14　多个外设时有条件数据传送流程图

[例 5.2] 以查询方式工作的字符输入设备，数据输入端口地址为 54H，状态端口地址为 56H。状态寄存器中 D_0 位为 1，表示输入缓冲器中已经有一个字节准备好，可以进行输入；D_1 位为 1 表示输入设备发生故障，则显示错误信息后停止。要求从该设备上输入 80 个字符，配成偶校验。然后从输出设备输出，其数据输出端口地址 55H，状态端口地址 57H，状态寄存器 D_7 位为 0 表示“空闲”，可以输出一个字符。试编写汇编语言程序。

说明：计算机内的字符用七位 ASCII 码表示，需要进行奇偶校验时通常用一个字节的最高位作为校验位。程序中产生偶校验方法是：从设备读入数据后，清除最高位，然后根据剩余七位的奇偶特性决定最高位置 1 或 0。

解

汇编语言程序如下：

```
DSEG    SEGMENT
BUFFER    DB  81 DUP(?)
MESSAGE   DB  'Device  Fault! ',0DH,0AH,'$'
DSEG      ENDS
CSEG      SEGMENT
  ASSUME CS:CSEG,DS:DSEG
START:MOV AX,DSEG          ;对 DS 初始化
      MOV   DS,AX
      LEA   SI,BUFFER      ;SI 为输入缓冲区指针
      MOV   CX,  80        ;设置 CX 为计数器,字符个数
NEXT: IN    AL,  56H       ;读入状态信息
      TEST  AL,02H         ;测状态寄存器的 D1
      JNZ   ERROR          ;D1 为 1,设备故障,转 ERROR
      TEST  AL,01H         ;测状态寄存器 D0
      JZ    NEXT           ;D0 为 0 未准备好,则转 NEXT,再测
      IN    AL,54H         ;否则 D0 为 1 准备好,读入字符信息
      AND   AL,7FH         ;清最高位,进行校验
      JPE   STORE          ;已经是偶数个 1,则转 STORE
      OR    AL,80H         ;奇数个 1,将最高位置为 1
STORE:MOV   [SI],AL        ;将字符送缓冲区
      INC   SI             ;修改地址指针
      LOOP  NEXT           ;80 个字符未输入完成,继续输入
      LEA   SI,BUFFER      ;否则 80 个字符输入完,准备发送,SI 中置字符串首址
      MOV   CX,80          ;发送字符数
ONE:  IN    AL,55H         ;读输出设备状态信息
      TEST  AL,80H         ;测 D7 位
      JNZ   ONE            ;D7 不为 0,转 ONE 继续测
      MOV   AL,[SI]        ;否则 D7 为 0,取出一个字符
      OUT   57H,AL         ;从输出设备输出一个字符
      INC   SI             ;修改指针
      LOOP  ONE            ;输出下一个字符
      JMP   DONE
```

```
ERROR:MOV    AH,09H          ;设备故障,输出出错信息
      LEA    DX,MESSAGE
      INT    21H
DONE: MOV    AH,4CH
      INT    21H             ;返回 DOS
CSEG  ENDS
      END    START
```

上述介绍了单个外设利用查询方式进行数据传送的工作过程。实际系统中通常连接有多个外设，这时 CPU 可采用循环查询的方法。发现哪个外设准备就绪，就对该外设进行数据传送，然后再查询下一个外设，依次循环。其流程图如图 5.14 所示。有条件传送方式的主要优点是能保证主机与外设之间协调同步工作；这种传送方式硬件线路简单，程序容易实现。其主要缺点是 CPU 需要不断地查询外设是否“准备就绪”而不能进行任何其他操作，大大降低了 CPU 的效率；数据交换的实时性差。

5.3.3 中断控制方式

从上面的介绍可以看到，条件和无条件传送方式都是在满足一定条件下才能采用，在与一些中、慢速的外设交换信息时，真正用于传送数据的时间是很短的，大大降低了 CPU 的使用效率。在查询工作方式中，CPU 处于主动地位，外设处于消极等待查询的被动地位。在一个实际控制系统中，外设常常可能有数十个，甚至上百个。由于它们工作速度不同，要求 CPU 服务时间又有随机性，有些要求又是很紧迫的。查询工作方式的数据传送很难使系统中每一个外设都能工作在最佳状态。

为了使 CPU 能有效地管理多个外设，提高 CPU 的工作效率，可以赋予系统中的外设某种主动申请、配合 CPU 工作的“权利”。例如，某外设已把数据准备好，它可以主动向 CPU 发一个请求信号。CPU 在接收到这个请求信号之后，暂停当前的工作，转而进行该外设的数据传送。赋予外设这样一种“主动权”之后，CPU 不必反复查询该设备状态，而是正常地处理系统任务，CPU 与外设处于并行工作状态，从而提高 CPU 的工作效率。

为此引入了中断概念。中断就是 CPU 在执行程序时某个外设需要进行数据传送而向 CPU 发出中断请求，CPU 在接到请求后若条件允许，则打断（或中断）正在进行的任务而转去对该外设的服务，并在外设服务结束后回到原来被中断的地方继续执行原来任务。可见，只有当外设要传送数据时才向 CPU 发中断请求信号，外设是主动的，CPU 是被动的。

利用中断控制方式进行数据传送，不仅大大提高了 CPU 效率，而且提高了服务的实时性。有关中断的进一步讨论将在第 6 章中展开。

5.3.4 直接存储器存取（DMA）方式

上述各种方式，数据传送是通过 CPU 执行输入/输出指令完成的，传送速度不高。尤其是与慢速外设进行数据传送时，会影响 CPU 的利用率。若存储器与快速 I/O 设备间有大量数据需要传送，为了提高传送速度和效率能否让存储器与快速 I/O 设备间直接进行传送而不经过 CPU 呢？答案是肯定的，这就是直接存储器存取方式（Direct Memory Access，DMA）。新型的 DMA 传送可扩展到存储器的两个区域之间，或两种高速外围设备之间。

DMA 方式的主要优点是速度快，数据传送速率只受存储器存取时间和 I/O 设备的速度限制；缺点是需要专用硬件支持，硬件连接也复杂些。在 IBM/XT 中，采用的是 Intel 8237

DMAC。

1. DMA 控制器（DMAC）的工作过程

为了更好地说明 DMAC 的工作过程，图 5.15 所示给出了 DMA 传送操作流程。图 5.16 给出了 DMAC 与 CPU、外设接口、存储器间应答联络信号及传送原理示意图。

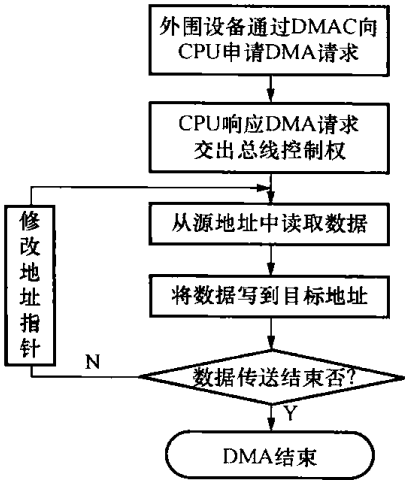


图 5.15 DMA 传送操作流程图

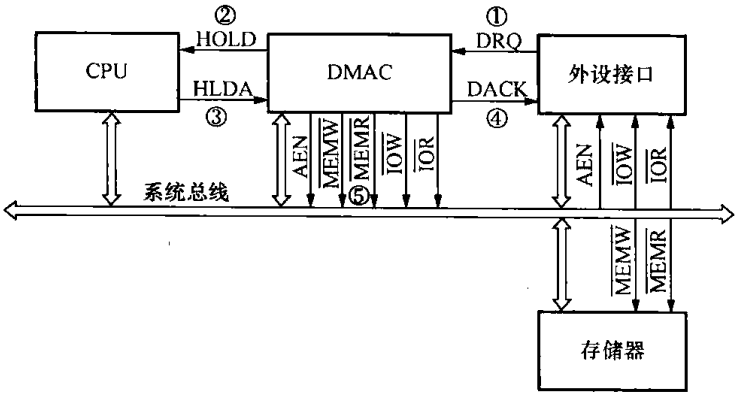


图 5.16 DMA 传送原理示意图

DMA 工作过程大致如下：

（1）当外设准备好，要求进行 DMA 传送时，外设向 DMAC 发出传送请求信号（DRQ）。

（2）DMAC 收到请求后，向 CPU 发出“总线请求”信号 HOLD，希望占用总线。

（3）CPU 在完成当前总线周期后会立即对 HOLD 信号进行响应。响应包括两个方面：一是 CPU 将数据总线、地址总线和相应的控制信号线均置为高阻态，由此放弃对总线的控制权；另一方面，CPU 向 DMAC 发出“总线响应”信号（HLDA）。

（4）DMAC 收到 HLDA 信号后，就开始控制总线，并向外设发出 DMA 响应信号 DACK。

（5）DMAC 送出地址信号和相应的控制信号，实现内存与外设或内存与内存之间的直接数据传送。

（6）DMAC 自动修改地址和字节计数器，并据此判断是否需要重复传送操作。规定的数据传送完后，DMAC 就撤销发往 CPU 的 HOLD 信号。CPU 检测到 HOLD 失效后，紧接着撤销 HLDA 信号，并在下一时钟周期重新开始控制总线，继续执行原来的程序。

2. DMA 控制器的基本功能

通过以上分析，一个 DMAC 应具有如下基本功能：

（1）能接收外设的请求，向 CPU 发出 DMA 请求信号。

（2）当 CPU 发出 DMA 响应信号后，接管对总线的控制，进入 DMA 方式。

（3）能寻址存储器，即能输出地址信息和修改地址。

（4）能向存储器和外设发出相应的读/写控制信号。

（5）能控制传送的字节数，判断 DMA 传送是否结束。

（6）在 DMA 传送结束后，能结束 DMA 请求信号，释放总线，CPU 接管总线并恢复正常工作。

3. DMA 传送方式

DMAC 一般有两种基本的 DMA 传送方式：

（1）单字节方式：每次 DMA 请求只传送一个字节数据，每传送完一个字节，都撤除 DMA 请求信号释放总线。

（2）字节（字符）组方式：每次 DMA 请求连续传送一个数据块，待规定长度的数据块传送完后才撤除 DMA 请求信号释放总线。

在 DMA 传送中，为了源和目的之间的数据传送取得同步，不同的 DMAC 在操作时都受到外设请求信号或准备就绪信号——READY 信号的限制。

5.4 DMA 控制器 8237A

8237A 及 82C37A 是专为 80X86 系统配备的具有 DMA 功能的大规模集成电路的 DMA 控制器，主要用于数据块传送。在 5MHz 时钟频率下，每通道的传送速度可达 1.6MB/s，最多可传送 64KB 的数据块。在 IBM PC/XT 中用了一片 Intel 8237A-5，提供四个 DMA 通道。在 IBM PC/AT 中用了两片 8237A-5，提供七个 DMA 通道。本节主要介绍 NMOS 8237A 的引脚及功能、工作方式、内部寄存器、寻址和编程应用。

5.4.1 DMA 8237A 的引脚及功能

图 5.17 所示为 DMA 8237A 芯片的引脚排列图，具有以下基本功能：

（1）有 4 个独立 DMA 通道，通常称为通道 0、通道 1、通道 2、通道 3。每个通道的 DMA 请求可分别被允许或禁止，时钟频率为 5MHz，每个 DMA 通道一次最大可传送 64KB 数据。

（2）可在存储器与外设、存储器与存储器之间传送数据。

（3）支持 4 种传送方式：单字节、数据块、请求传送和级连传送。

（4）允许外部用 EOP 信号结束 DMA 传送或重新初始化。

（5）每一个通道的 DMA 请求优先级可以是固定的，也可以由程序设定循环优先级。

（6）多个 DMA 芯片可以级连以扩展 DMA 通道。

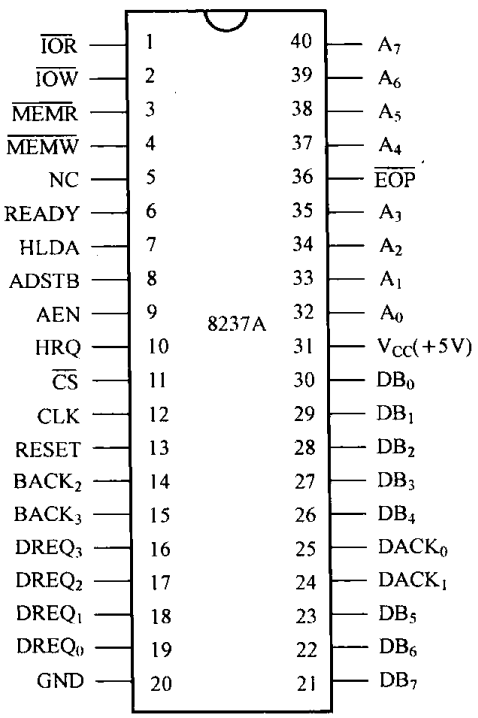


图 5.17 8237A 芯片的引脚排列

主要引脚信号功能及意义如下：

CLK：时钟输入。芯片内部操作的定时信号，决定 DMA 传输速度，最高频率可达 5MHz。

$\overline{CS}$：片选输入信号，低电平有效。有效时，CPU 可对其进行通信，初始化；通常与接口地址译码器连接。

RESET：复位输入信号，高电平有效。复位后，屏蔽寄存器被置位（禁止 DMA 请求）外，其余寄存器均清零。

READY：外设准备就绪输入信号，高电平有效。当 8237A 芯片处于 DMA 传送遇到慢速存储器或 I/O 设备时，由它们提供 READY 信号，据此插入等待时钟周期，延长总线传送周期。

AEN：地址允许输出信号、高电平有效。DMA 传送期间，该信号有效将 DMAC 内部 16 位地址的高 8 位送至系统地址总线上，使用系统总线。

ADSTB：地址选通输出信号，高电平有效。有效时将 DB_0～DB_7 上输出的高 8 位地址信息（A_{15}～A_8）锁存。和地址线（A_7～A_0）一起组成 16 位地址信息 A_{15}～A_0。

$\overline{MEMR}$：存储器读输出信号，三态，低电平有效。DMA 传送期间，控制存储器读操作。

$\overline{MEMW}$：存储器写输出信号，三态，低电平有效。DMA 传送期间，控制存储器写操作。

$\overline{IOR}$：I/O 读信号，双向三态，低电平有效。在非 DMA 传送时为输入控制信号，CPU 可读取 8237 内部寄存器的状态信息；在 DMA 传送时，为输出控制信号，与$\overline{MEMW}$配合把数据由外设写入存储器。

$\overline{IOW}$：I/O 写信号，双向三态，低电平有效。在空闲周期为输入控制信号，CPU 对 8237A 进行初始化编程；在 DMA 传送时为输出信号，与$\overline{MEMR}$配合把数据由存储器传至外设。

$\overline{EOP}$：过程结束双向信号，低电平有效。DMA 任一通道传送结束产生$\overline{EOP}$输出信号。当外部输入$\overline{EOP}$信号也会终止当前通道的 DMA 传送。若$\overline{EOP}$端不用，应接数千欧电阻再接到高电平上。

$DREQ_0$～$DREQ_3$：DMA 请求输入信号，分别对应通道 0～3，有效电平可通过初始化程序设定，芯片复位时均为低电平。通道上的外部设备需要 DMA 传送时，需在相应的 DREQ 端发出有效 DMA 请求。该有效请求应保持到 8237A 发出相应的响应信号 DACK 为止。

$DACK_0$～$DACK_3$：DMA 响应输出信号，分别对应通道 0～3，有效电平可通过初始化程序设定。它是 8237 对 DREQ 信号的响应信号。

HRQ：总线请求输出信号，高电平有效。它是 8237A 发向 CPU 的，以请求总线控制权。它一般连接到 CPU 的 HOLD 信号端。

HLDA：总线应答输入信号，高电平有效。它是 CPU 发给 8237A 的，表示 CPU 已让出系统总线控制权。

A_0～A_3：低 4 位地址信号线，双向三态。在空闲周期，CPU 输出低 4 位地址信号，用来选择 8237 内部寄存器；当进行 DMA 传送时，作为输出信号送出地址线的低 4 位。

A_4～A_7：高 4 位地址信号线，三态。进行 DMA 传送时，作为输出信号送出地址线的高 4 位。

DB_0～DB_7：8 位数据信号线，双向三态。在空闲周期，$\overline{IOR}$有效时，CPU 通过 DB_0～DB_7 读内部寄存器的内容；$\overline{IOW}$有效时，CPU 通过 DB_0～DB_7 对内部寄存器初始化编程。在 DMA 工作周期，由 DB_0～DB_7 送出高位地址信息 A_8～A_{15}。

在存储器到存储器的 DMA 传送时，除如上送出高 8 位地址信息外，在存储器读出的 DMA 周期数据经 DB_0～DB_7 送入 8237A 的数据缓冲器，然后在存储器写 DMA 周期时，再将此数据经 DB_0～DB_7 从 8237A 的数据缓冲器写入指定存储器单元。

8237A 芯片提供 16 位地址信息，故对存储器的寻址范围为 64KB。如果进一步扩大寻址范围，可在 DMA 系统中为每一个通道配置一个页面寄存器。

5.4.2 8237A 的工作方式

1. 8237A 的工作周期

8237A 的工作有空闲周期和工作周期两种周期，每个周期由多个 DMA 时序周期组成，类似 CPU 的时钟周期。

（1）空闲周期。当 8237A 的任一通道无 DMA 请求或 DMA 传送工作结束后均进入空闲

周期。处于此状态下的 8237A 相当于接在总线上的接口，CPU 可对其初始化编程，设置其工作状态。

在空闲周期，8237A 每个时钟周期均采样请求输入线 DREQ，同时也采样片选信号$\overline{CS}$，检测有无 DMA 请求发生和 CPU 对其内部寄存器寻址。

(2) 工作周期。处于空闲状态的 8237A 的某一通道提出 DMA 请求时，向 CPU 发出 HRQ 有效信号，在未收到 CPU 的 HLDA 应答信号时，8237A 仍处于编程状态，即初始状态。当 CPU 执行完当前的总线周期，便响应 DMAC 的请求，CPU 发出 HLDA 应答信号，8237A 收到了 HLDA 应答信号后，进入工作周期，开始 DMA 传送。

如果 RAM 或 I/O 设备传送数据速度较慢，不能在规定的时间内完成数据写入，应设计等待电路，自动插入一个等待状态，从而延长了读/写时间。

8237A 进行存储器到存储器的 DMA 传送时，每一次传送需要两个总线周期；第一个总线周期进行存储器读操作；第二个总线周期进行存储器写操作。

8237A 的通道 DMA 请求有硬件请求和软件请求两种。硬件请求是当 I/O 设备准备好数据后将 8237A 的 $DREQ_0$～$DREQ_3$ 某个通道 DMA 请求端置成有效来提出 DMA 请求；软件请求是利用编程将 DMA 请求触发器置成有效来提出 DMA 请求。软件请求时，不能用屏蔽寄存器加以屏蔽。

2. DMA 数据传送的工作方式

8237A 在 DMA 传送时有以下四种工作方式：

(1) 单字节传送方式。每一次 DMA 请求仅传送一个字节数据。传送后，当前字节计数器自动减 1，同时当前地址寄存器的内容加 1 或减 1。传送完一个字节，HRQ 变为无效，DMAC 释放系统总线，将总线控制权交还给 CPU 至少一个总线周期，并检查 DREQ 输入信号，一旦有效，再进行下一字节数据传送。重复上述过程直到字节计数器为 0，还要再传送一个字节，又从 0 减到 0FFFFH，才发出$\overline{EOP}$信号，结束整个 DMA 传送过程。

(2) 数据块传送方式。在这种方式下，一旦 8237A 控制了系统总线，就一直占用总线，连续地传送字节数据。每传送一个字节自动修改地址寄存器，并使当前字节计数器减 1，直到将所有规定的字节全部传送完，即字节计数器减至 0FFFFH，并产生有效$\overline{EOP}$信号为止。也可由外部输入$\overline{EOP}$有效低电平信号强行终止 DMA 传送。在传送期间不检测 DREQ 请求信号。

(3) 请求传送方式。类似数据块传送方式，不同点在于每传送一个字节 8237A 都要对 DREQ 信号进行检查，若 DREQ 变为无效，则暂停传送。当测得 DREQ 又变为有效后，则在原来基础上继续传送，直到当前字节计数器为 0，或外部加入$\overline{EOP}$有效低电平信号时停止 DMA 传送。

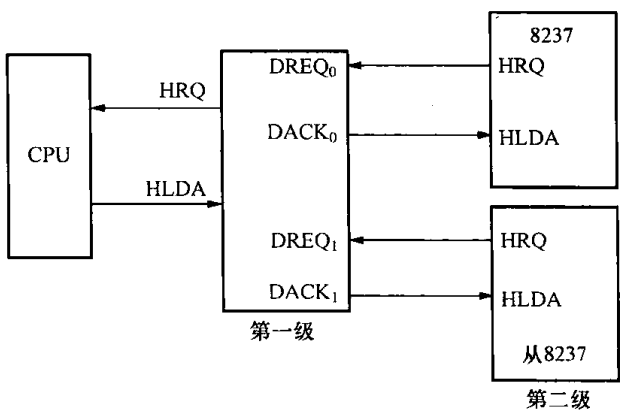


图 5.18 DMA 级联方式连接框图

(4) 级联方式。这种方式可以把多个 8237A 连接在一起以便扩展系统的 DMA 通道数。其中一片 8237A 为主 DMAC，其余作为从 DMAC 进行级联，其连接如图 5.18 所示。二级 8237A 芯片的 HRQ 接到一级

8237A 某一个通道的 DREQ，二级 HLDA 接到一级 8237A 某一个通道的 DACK，一级 8237A 的 HRQ 和 HLDA 信号与 CPU 相连。

在级联方式下，8237A 初始化为级联方式，并要求一级 8237A 级联通道的优先级要高于二级 8237A 的通道。当二级某个通道有 DMA 请求通过一级 8237A 通道得到 CPU 响应后，一级 8237A 只给二级 8237A 发出 DACK 响应信号，由提出 DMA 请求的二级 8237A 的相应通道输出地址与控制信号。

5.4.3 8237A 内部寄存器组

8237A 的内部寄存器组分为两类：一类是 4 个通道公用的寄存器，共 7 个；另一类是各通道都有一个的寄存器，共 5 个，如表 5.2 所示。

表 5.2　　8237A 内部寄存器

寄存器名	长度（位）	数量（个）	寄存器名	长度（位）	数量（个）
基地址寄存器	16	4	基字节数计数器	16	4
当前地址寄存器	16	4	当前字节数计数器	16	4
临时地址寄存器	16	1	临时字节数计数器	16	1
状态寄存器	8	1	命令寄存器	8	1
临时寄存器	8	1	模式寄存器	6	4
屏蔽寄存器	4	1	请求寄存器	4	1

（1）基地址寄存器和基字节数寄存器。每一个通道有一对 16 位的基地址寄存器和基字节数寄存器，存放当前寄存器和当前字节数寄存器的初值，在“自动预置”时用于恢复当前地址寄存器和当前字节数寄存器的初值。初始化编程时，基地址寄存器和基字节数寄存器与相应的当前寄存器同时由 CPU 写入，但该对寄存器不能被 CPU 读出。

（2）当前地址寄存器。每一个通道有一个 16 位当前地址寄存器，提供本次 DMA 传送的内存地址（低 16 位）。在每次传送后该寄存器的值自动加 1 或减 1，其中的值可被 CPU 写入或读出。当初始化为“自动预置”时，一旦全部字节传送完毕，基地址寄存器的内容会自动装入该寄存器。

（3）当前字节数寄存器。每一个通道有一个 16 位当前字节数寄存器，用来保存 DMA 要传送的字节数，在每次传送一个字节后该寄存器值减 1。当该寄存器值减到 0FFFFH 时，产生计数结束信号$\overline{EOP}$。同样“自动预置”时重装入基字节寄存器的初值。

（4）临时地址寄存器和临时字节数寄存器。这两个 16 位寄存器与使用者使用 8237A 没有关系。

（5）命令寄存器。4 个通道共用一个 8 位命令寄存器，用于控制 8237A 的工作。命令字格式如图 5.19 所示。

D_0：规定是否允许存储器到存储器的传送方式。0：禁止；1：允许。

D_1：规定通道 0 的地址是否保持不变。在存储器到存储器传送中，源地址由通道 0 提供，读出数据到暂存寄存器，由通道 1 送出目的地址，将数据写入。若 $D_1=1$，则在整个数据块传送中保持存储器地址不变。因此，就会将同一个数据写入目的存储器块中。

D_3：规定一个 DMA 周期是由两个时钟周期完成，还是由三个时钟周期完成。

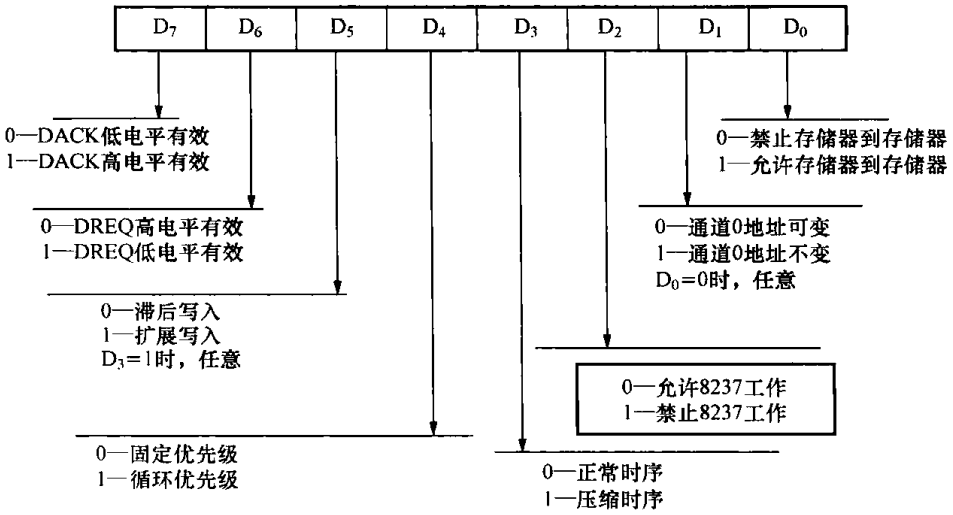


图 5.19 8237 命令字格式

D_4：多个通道同时提出 DMA 请求时进行优先级排队管理。0：固定优先级；1：循环优先级。固定优先级是通道 0 优先级最高，通道 1、2、3 优先级依次递降。PC 机采用固定优先级；循环优先级是最近被服务过的通道，在下一次循环中置成最低优先级的通道。例如，最近被服务过的是通道 1，则下一次的优先级顺序是：通道 2 最高，通道 3、0、1 依次降低。循环优先级可以使任一通道都能得到服务的机会。

D_5：规定写脉冲，扩展写入比滞后写入提前（宽）一个时钟周期。

命令字的其他各位易理解，不再详细说明。

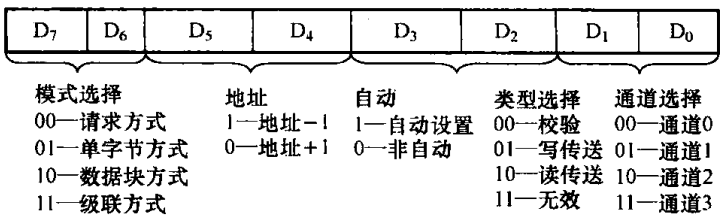


图 5.20 模式字格式

（6）模式寄存器。4 个通道共用一个 8 位模式（方式）寄存器，规定 8237A 通道工作模式，如图 5.20 所示。

D_1、D_0：选择设置通道。事实上，每个通道有一个 6 位方式寄存器，用于存放 $D_7 \sim D_2$ 的 6 位模式设定，但不能寻址。

D_3、D_2：选择传送类型。DMA 有读传送、写传送和校验三种传送类型。读传送是把数据由存储器传送到 I/O 设备。当$\overline{\text{MEMR}}$有效时，从存储器读出数据，在$\overline{\text{IOW}}$有效时，将数据传送给外设；写传送是将由外设输入的数据写至存储器中。当$\overline{\text{IOW}}$信号有效时，从外设输入数据，在$\overline{\text{MEMR}}$有效时，将数据写入内存；校验传送是对 8237A 读/写功能进行校验。

D_4：预置方式选择。1：自动预置；0：非自动预置。预置方式是指预置地址寄存器和字节计数器的初始值。

自动预置。当某通道完成一个 DMA 服务，就自动将基地址、基字节计数器值复制到当前地址寄存器和当前字节数寄存器中。需要注意，如果一个通道设置成自动预置功能，那么该通道的对应屏蔽位必须为 0。

非自动预置，也称 CPU 预置方式。由 CPU 对 DMAC 的初始化编程对基地址寄存器和基字节计数器置初值，当前地址寄存器和当前字节寄存器也自动置成相应值。

其他各位易理解，不再详细说明。

（7）请求寄存器。4个通道共用一个请求寄存器，用于在软件控制下让某通道产生一个DMA请求，如同外部DREQ请求一样。请求命令字格式如图5.21所示。

$D_1 \sim D_0$：选择不同通道的“软件DMA”请求。软件请求是非屏蔽的，但它们仍然受优先级控制。

一般当8237A工作在数据块传送方式时可用“软件DMA”请求。若存储器到存储器传送，则通道0必须使用软件请求，以启动传送过程。

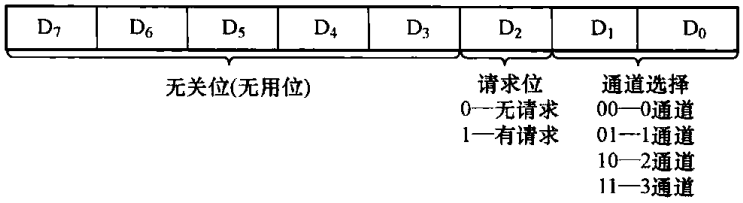


图5.21 请求命令字格式

（8）屏蔽寄存器。通过屏蔽字可屏蔽或允许某个或某些个通道发出DMA请求，8237A的屏蔽字有两种形式，如图5.22所示。

1）单通道屏蔽字。如图5.22（a）所示，只能对某个通道设置允许或屏蔽接收DREQ请求。

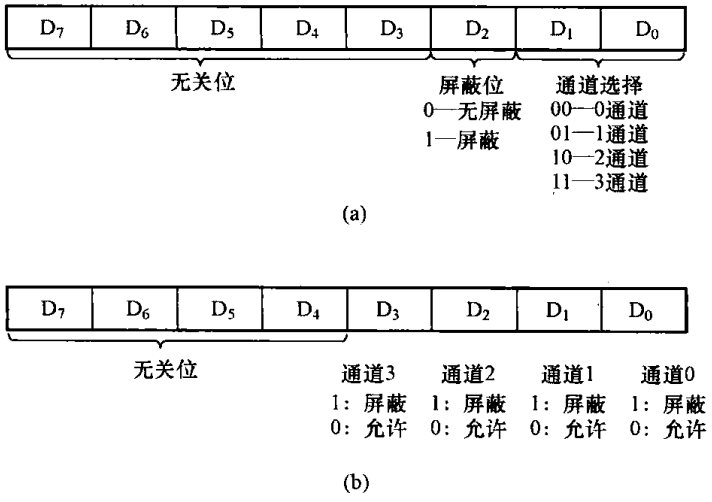


图5.22 8237A通道屏蔽字格式

（a）单通道屏蔽字格式；（b）多通道屏蔽字格式

2）多通道屏蔽字。如图5.22（b）所示，能对4个通道设置允许或屏蔽接收DREQ请求。

另外需注意：在RESET信号作用后，8237A的4个通道全处于屏蔽状态。当某个通道进行DMA传送后，如果不是工作在自动初始化方式，则该通道处于屏蔽状态。

（9）状态寄存器。状态寄存器存放各通道的状态，CPU读取其内容后，可得知8237A的工作状态，如图5.23所示。状态寄存器的低4位反映当前每个通道计数是否已减到0，高4位反映每个通道是否有未处理的请求。

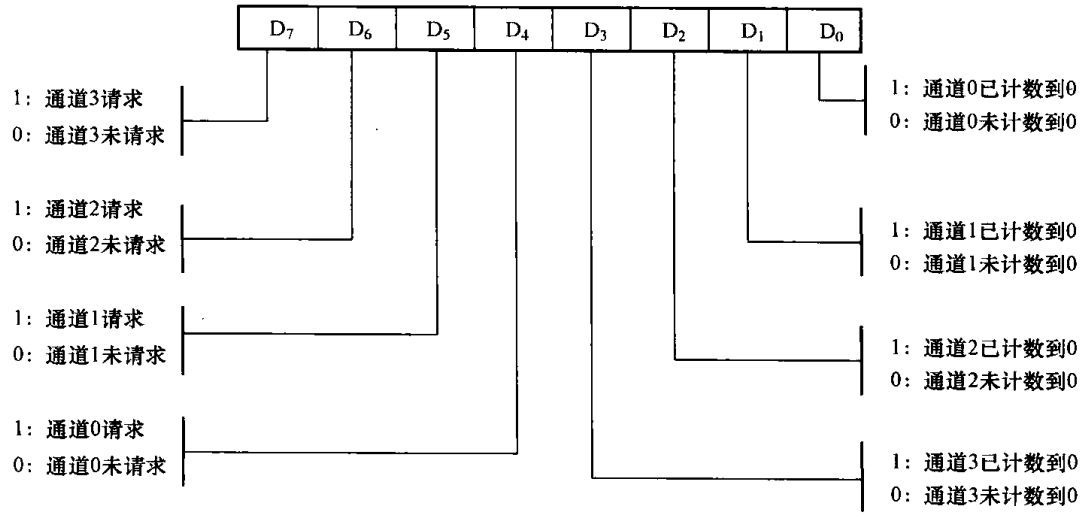


图5.23 8237A状态字格式

（10）临时寄存器。临时寄存器又称暂存器，是 4 个通道共用的 8 位寄存器。用于存储器到存储器的传送方式，临时保存从源存储单元读出的数据。

5.4.4 寻址与连接

（1）寻址信息。每片 8237 有 4 条片内地址选择线 $A_3 \sim A_0$，故可占用 16 个连续的 I/O 端口地址。在 CPU 用片选信号$\overline{CS}$选中芯片 8237A 时，由片内地址线 $A_3 \sim A_0$ 选择 8237A 内部的一个端口，再用$\overline{IOR}$信号或$\overline{IOW}$信号决定是对某个内部寄存器读出还是写入。CPU 对这些寄存器的寻址信息列于表 5.3 中。表中各内部寄存器的端口地址以符号地址 DMA+0，DMA+1，…，DMA+0FH 表示。

表 5.3　　8237A 各通道寄存器寻址

A_3	A_2	A_1	A_0	端口地址	通道	读操作	写操作
0	0	0	0	DMA+1	0	读当前地址寄存器	写基/当前地址寄存器
0	0	0	1	DMA+1		读当前字节计数器	写基/当前字节计数器
0	0	1	0	DMA+2	1	读当前地址寄存器	写基/当前地址寄存器
0	0	1	1	DMA+3		读当前字节计数器	写基/当前字节计数器
0	1	0	0	DMA+4	2	读当前地址计数器	写基/当前地址寄存器
0	1	0	1	DMA+5		读当前字节计数器	写基/当前字节计数器
0	0	0	0	DMA+6	3	读当前地址计数器	读当前字节计数器
0	1	1	1	DMA+7		写基/当前地址寄存器	写基/当前字节计数器
1	0	0	0	DMA+8	公用	读状态寄存器	写命令寄存器
1	0	0	1	DMA+9		—	写请求寄存器
1	0	1	0	DMA+10		—	写单个通道屏蔽字
1	0	1	1	DMA+11		—	写方式字寄存器
1	1	0	0	DMA+12		—	清除先/后触发器
1	1	0	1	DMA+13		读暂存寄存器	复位芯片（主清除）
1	1	1	0	DMA+14		—	清除屏蔽寄存器
1	1	1	1	DMA+15		—	写 4 通道屏蔽寄存器

从表 5.3 中可见，前 8 个地址（$A_3=0$）是各个通道单独占有的，每两个地址对应一个通道内部的四个寄存器。由于这 4 个寄存器是 16 位的，而 8237A 的数据线仅 8 位，所以对它们的读/写要分高低字节连续两次完成。

表中后 8 个地址（$A_3=1$）是 4 个通道公共的，在空闲周期，根据 I/O 地址缓冲器送来的最低 4 位地址线 $A_3 \sim A_0$ 与$\overline{IOW}$、$\overline{IOR}$信号相配合，通过写入命令对 8237A 的控制与状态寄存器进行设置，规定其某些工作状态。这些寄存器寻址信息如表 5.4 所示。

表 5.4 控制和状态寄存器寻址信息

信号							操作
$\overline{CS}$	$\overline{IOR}$	$\overline{IOW}$	A_3	A_2	A_1	A_0	
0	1	0	1	0	0	0	写命令寄存器
0	1	0	1	0	0	1	写请求寄存器
0	1	0	1	0	1	0	写单个屏蔽寄存器
0	1	0	1	0	1	1	写工作方式寄存器
0	1	0	1	1	1	1	写所有屏蔽位
0	0	1	1	1	0	1	读临时寄存器
0	0	1	1	0	0	0	读状态寄存器
×	1	0	1	1	0	0	清高低触发器命令
×	1	0	1	1	0	1	主清除命令

8237A 在空闲周期有主清除、清高/低触发器和清屏蔽寄存器命令三条软件命令。

1）主清除命令。与硬件复位信号 RESET 相同，其作用是清除命令寄存器、状态寄存器、各通道的请求标志位、临时寄存器和高/低位触发器。并把各通道的屏蔽标志位置“1”，使 8237A 进入空闲周期，此主清命令寄存器的地址低 4 位为 0DH。

2）清除高/低触发器命令。高/低触发器又称字节指示器或先/后触发器。因为 8237A 各通道的地址和字节计数器都是 16 位，而 8237A 每次只能接收一个字节数据，所以 CPU 访问这些寄存器时，要进行连续两次字节操作。首先 CPU 使触发器置“0”，则 CPU 第一次访问这些 16 位寄存器的低字节，然后触发器自动置“1”，则下一次访问是高字节，然后触发器又自动恢复成“0”状态，此高/低触发器的地址低 4 位是 0CH。

3）清除屏蔽寄存器命令。清除 4 个通道的全部屏蔽位（屏蔽标志位清零），使各通道均能接受 DMA 请求，此命令寄存器地址的低 4 位为 0EH。

（2）连接。图 5.24 给出了 8237A 与 8088 CPU 的连接示意图。

5.4.5 初始化编程

（1）初始化步骤。通过 8237A 进行 DMA 传送，必须对其进行初始化编程。步骤如下：

1）发送总清命令。用复位信号（RESET）或软件命令对 8237A 总清。

2）写基和当前地址寄存器。

3）写传送的字节数给基和当前字节计数器。

4）写模式寄存器确定 8237A 的工作方式和传送类型。

5）写屏蔽寄存器。

6）写命令寄存器，控制 8237A 工作。

7）写请求寄存器，或用硬件方法等待 DREQ 引线端发出 DMA 传送申请。

（2）编程应用举例。

[例 5.3] 利用通道 1 从外设输入 54KB 的一个数据块，传送至 3000H 开始的存储区域（增量传送），采用块传送方式，传送完不自动初始化，外设的 DREQ 和 DACK 都为高电平有效。已知 8237A 的端口地址为 50H～5FH。

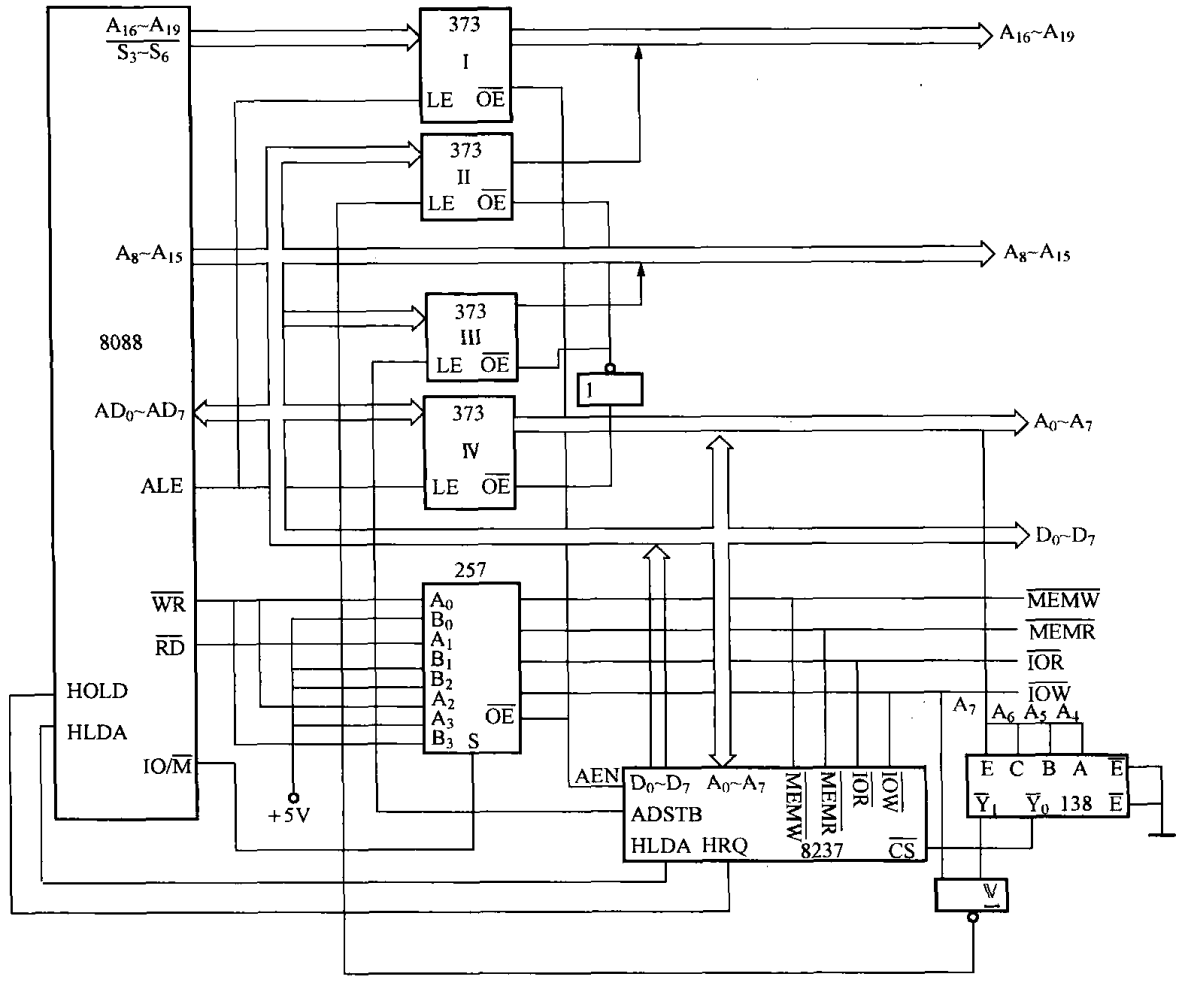


图 5.24　8237A 与 8088 CPU 连接示意图

解　根据题意，模式控制字为：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	1	0	1
块传送		增量	非自动		写传送		通道 1

屏蔽字为：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	1
					复位		通道 1

命令字为：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	0	0	0	0	0
DACK 高	DERQ 高	扩展写	固定优先	正常时序	启动	无关	非存储器至存储器

由于给定的 DMA 控制器的端口地址为 50H～5FH，则主清除命令端口为 5DH，基地址和当前地址寄存器的端口地址为 52H，基字节数和当前字节数计数器的端口地址为 53H，模式控制字的端口地址为 5BH，写单通道的屏蔽字端口地址为 5AH，命令寄存器端口地址为 58H。

初始化程序段如下：

```
OUT     5DH,AL          ;主清除命令
MOV     AL,00H          ;基地址和当前地址的低8位
OUT     52H,AL
MOV     AL,30H          ;基地址和当前地址的高8位
OUT     52H,AL
MOV     AL,00H          ;基字节数和当前字节数低8位(54K=0D800H)
OUT     53H,AL
MOV     AL,0D8H         ;基字节数当前字节数低8位
OUT     53H,AL
MOV     AL,85H          ;模式控制字
OUT     5BH,AL
MOV     AL,01H          ;屏蔽控制字,使通道1的屏蔽位复位(不屏蔽)
OUT     5AH,AL
MOV     AL,0A0H         ;命令字
OUT     58H,AL
```

本段程序执行后，真正的数据传送不需要用CPU的指令，DMA控制器8237A会自动将外部设备的54KB数据传送到从3000H开始的内存区域中。

5.4.6 8237A在PC/XT系统中的应用

8237A DMA控制器接入微机系统并取得总线控制权以后，就成为系统的主控制器，它如何向存储器和I/O设备发出地址信息，这是在系统连接时应考虑的问题。

8237A只能提供16位地址信息（A_0～A_7为低8位，DB_0～DB_7为高8位）。而PC/XT的地址总线有20位，1MB的寻址空间；PC/AT的地址总线有24位，寻址空间为16MB。8237A与PC系统两者的地址线不能直接相连，解决的办法是在系统中另设置DMA页面地址寄存器，对PC/XT而言产生DMA通道的高4位地址A_{16}～A_{19}；对PC/AT要产生高8位地址A_{16}～A_{23}，这样与8237A提供的16位地址一起组成20位地址线或24位地址线，从而能访问到存储器的全部存储单元和I/O设备端口。图5.25所示为PC/XT DMA系统产生地址信息的示意图。表5.5列出了PC/XT与PC/AT的DMA芯片的端口地址。

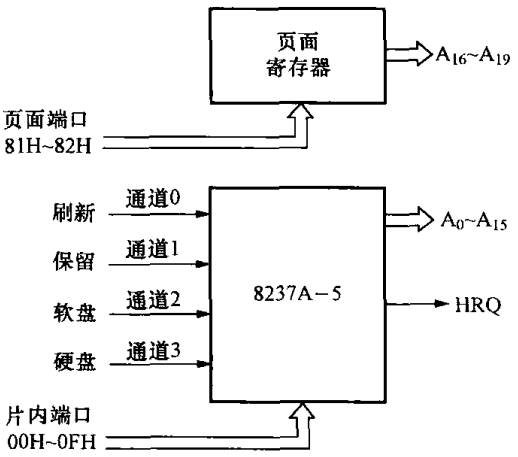


图5.25 PC/XT DMA系统产生地址信息示意图

表5.5 PC/XT的DMA芯片端口地址

端口名称		0号8237A	1号8237A
通道0	基/当前地址寄存器	0（00H）	192（C0H）
	基/当前字节计数器	1（01H）	194（C2H）
通道1	基/当前地址寄存器	2（02H）	196（C4H）
	基/当前字节计数器	3（03H）	198（C6H）

续表

端口名称		0 号 8237A	1 号 8237A
通道 2	基/当前地址寄存器	4 (04H)	200 (C8H)
	基/当前字节计数器	5 (05H)	202 (C6H)
通道 3	基/当前地址寄存器	6 (06H)	204 (CCH)
	基/当前字节计数器	7 (07H)	206 (CEH)
读状态寄存器/写命令寄存器		8 (08H)	208 (D0H)
写请求寄存器		9 (09H)	210 (D2H)
写单个通道屏蔽寄存器		10 (0AH)	212 (D4H)
写方式字寄存器		11 (0BH)	214 (D6H)
写清除先/后触发器		12 (0CH)	216 (D8H)
读暂存寄存器/写总清除		13 (0DH)	218 (DAH)
写清除屏蔽寄存器		14 (0EH)	220 (DCH)
写 4 合个通道屏蔽寄存器		15 (0FH)	222 (DEH)

1. PC/XT 的 DMA 系统

在 PC/XT 机的 DMA 系统中采用 1 片 8237A，可支持 4 个通道 DMA 传送。其中，通道 0 用于对动态 RAM 刷新（刷新请求周期为 16.08μs），通过对存储器读操作实现其刷新功能；通道 1 保留（有同步通信时，留给网络数据链路控制卡使用；当系统未配置网络卡时，由用户安排使用）；通道 2 用于软盘；通道 3 用于硬盘传送数据。

所有通道均传送 8 位数据，每次 DMA 传送最多 64KB。图 5.26 所示为 8237A 在 PC/XT 机产生 20 位地址线的连接方法，可在 1M 空间范围寻址。CPU 对 8237A 访问时各端口地址为 00H～0FH，即 DMA+0～DMA+15，而符号地址定义为 DMA EQU 00H。

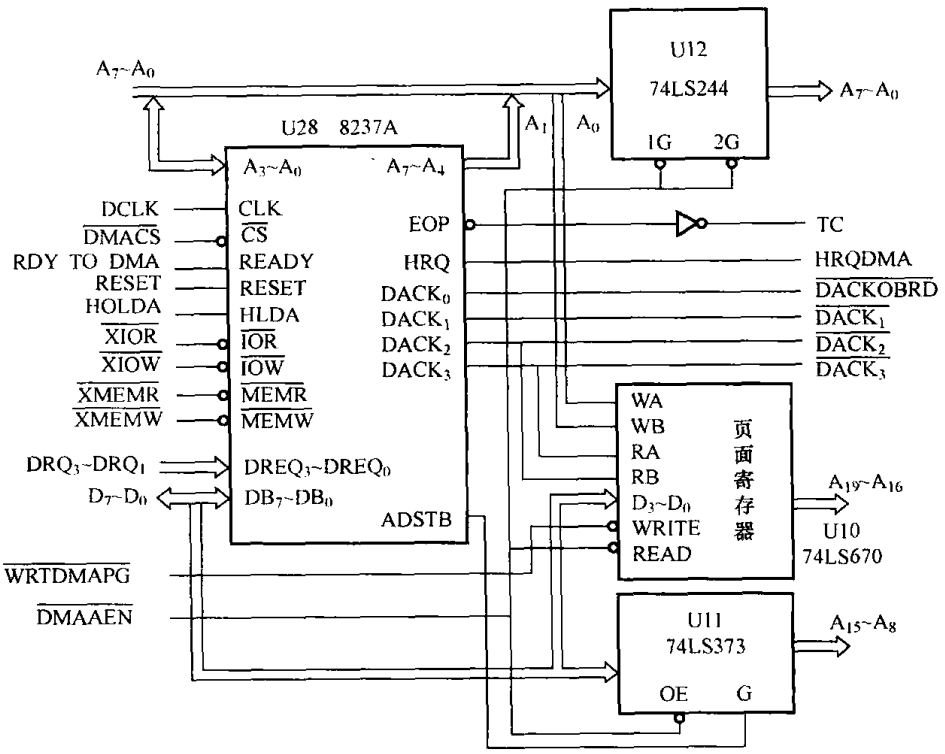


图 5.26 8237 产生 20 位地址线的连接方法

74LS670为页面地址寄存器，其内部有第0～3号共4组寄存器，每组4位，4个端口地址号80H、83H、81H、82H，分别对应通道0、1、2、3。页面寄存器中4组寄存器用于存放各通道中16位地址寄存器所对应的高4位A_{19}～A_{16}的地址信号（页面号）。

2. 82C37A与32位微处理器80386DX的接口

图5.27所示为82C37A与32位80386DX相连的接口方法。82C37A含有4个独立的DMA通道，通常把每一个通道指定给一个专门的外围设备。82C37A允许I/O设备输入4个DMA请求信号$DREQ_0$～$DREQ_3$，分别对应通道0、1、2、3。在空闲周期，82C37A不断地检测这些输入信号是否有效。当某一外设要求进行DMA操作时，就使相应的DREQ输入信号变为有效，向82C37A请求DMA服务。

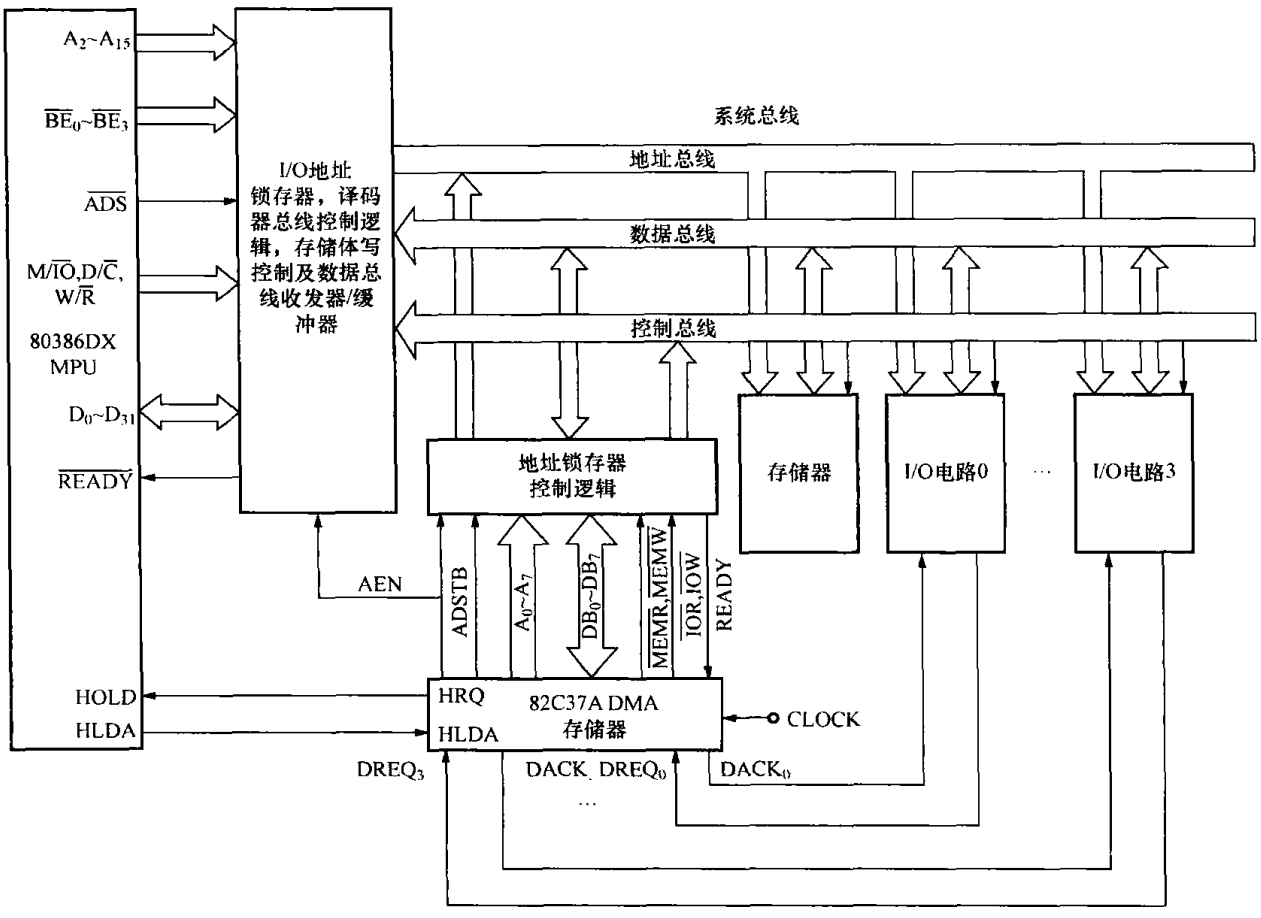


图5.27 82C37A与32位80386DX相连的接口方法

当82C37A采样到任何一个通道中出现一个DREQ有效，且未被屏蔽DMA，则向CPU发出总线请求，保持信号HRQ有效，并传送至80386DX的HOLD输入端，通知CPU外设有DMA请求，要求获得对总线控制权。当80386DX允许DMA请求后，发出响应信号HLDA告之DMAC，CPU已放弃了对总线控制权，DMAC可接管对总线的控制权。

当82C37A获得了对总线的控制权后，就输出一个DMA响应信号DACK，告之请求DMA传送的外部I/O设备，82C37A已处于就绪状态。这样选中的I/O设备就可在82C37A的控制下进行DMA操作。

在DMA工作周期，82C37A产生相应的地址信息及产生存储器或I/O数据的全部控制信号。DMA总线周期开始时刻，在地址选通信号ADSTB的控制下，形成有效的A_{15}～A_0

的 16 位地址信息，使 82C37A 能够直接寻址 64K 字节存储单元。地址允许 AEN 信号在整个 DMA 工作周期均为有效状态。

82C37A 的 DMA 操作可以在 I/O 设备与存储器之间或存储器与存储器之间进行。

例如：I/O 外部设备将数据传送到存储器时，82C37A 利用$\overline{IOR}$输出信号通知 I/O 设备把数据放置到数据总线 DB_0～DB_7 上。与此同时，它利用$\overline{MEMW}$信号把数据总线上的数据写入存储器。82C37A 仅产生地址及控制信息，就完成了直接从 I/O 设备将数据传送到存储器而没有经 82C37A。同样利用$\overline{MEMR}$和$\overline{IOW}$信号可完成从存储器到 I/O 设备的数据直接传送。上述 I/O 设备与存储器间的 DMA 操作都仅用 4 个时钟周期即可完成。

82C37A 完成存储器间的 DMA 传送，要用到$\overline{MEMR}$和$\overline{MEMW}$两个信号，故要用到两个总线周期，占 8 个时钟周期：一个是总线读周期，另一个是总线写周期。与 I/O 写存储器间的 DMA 传送不同的是：要用到 82C37A 中的临时寄存器。在总线读周期中，要将从存储器中读出的数据先暂存到 82C37A 中的临时寄存器，在总线写周期中，将数据从临时寄存器中传送到目的存储单元。在 DMAC 用 5MHz 时钟时，一个存储器到存储器的 DMA 操作周期需 1～6μs。

顺便指出，目前，许多公司已生产大规模集成电路的 DMA 控制器，如 Zilog 公司的 Z8410 和 Z8410A。Motorola 公司的 MC6844、MC68A44 和 MC68B44。Intel 公司生产的 DMAC 主要有 8257/8257-5 和 8237A/8237-5。其中，8257A-5 是与 8085 系列兼容的 DMAC，8237A 可用于 3MHz 时钟，而 8237A-5 时钟可提高到 5MHz。

在高档微机中，DMAC 和相关页面寄存器都被兼容的多功能外围芯片所取代。虽然电路结构相差甚远，但它们的基本功能都是相似的。

习　　题

5.1　输入/输出设备有哪些特点？CPU 通过什么与输入/输出设备通信？

5.2　CPU 与输入/输出设备通信时所用到的接口电路通常应具备哪些功能？

5.3　计算机与外设之间的数据传送控制方式有哪些？它们各有什么特点？

5.4　何谓“独立的 I/O 编址方式”？何谓“与存储器统一编址方式”？这两种编址方式各有什么特点？

5.5　CPU 与外设采用查询方式传送数据的过程是怎样的？现有一输入设备，其数据端口的地址为 FFE0H，状态端口地址 FFE2H，当其 D_0 位为 1 时表明输入数据备好。请编写采用查询方式进行数据传送的程序段，要求从该设备读取 100 个字节并输入到从 2000H：2000H 开始的内存中。

5.6　以外中断为例，一次完整的中断过程主要包括哪几个阶段？并简述各阶段。

5.7　DMA 控制器应具有哪些功能？

5.8　说明 8237A 单字节 DMA 传送数据的全过程。

5.9　说明 8237A 初始化编程的步骤。

5.10　利用 8237A 的通道 2，由一个输入设备输入一个 32KB 的数据块至内存，内存的首地址为 34000H，采用增量、块传送方式，传送完不自动初始化，输入设备的 DREQ 和 DACK 都是高电平有效。请编写初始化程序，8237A 的首地址用标号 DMA 表示。

第6章 中 断 技 术

中断技术是微处理器与外部设备交换信息的一种方式，是现代计算机系统中非常重要的功能，广泛应用于计算机故障检测与处理、实时信息处理、多道程序分时操作及人机交互等，不仅能够实现CPU与外部设备并行工作，而且可以及时处理系统内部和外部的随机事件，使系统更加有效地发挥效能，提高计算机的工作效率。

6.1 中断系统的基本概念

中断系统是完成中断功能的软件及硬件电路的总称，它可以使计算机实现如下操作：

(1) 故障检测和自动处理。计算机系统出现故障或程序执行错误都是随机事件，事先无法预料。如电源掉电、存储器出错、运算溢出等，采用中断技术可以有效地进行系统的故障检测和自动处理。

(2) 实时信息处理。在实时信息处理系统中，需要对采集的信息立即做出响应，以避免丢失信息，采用中断技术可以进行信息的实时处理。

(3) 分时处理。现代操作系统具有多任务处理功能，使同一个微处理器可以同时运行多道程序，通过定时和中断方式，使CPU按时间分配给每个程序，从而实现多任务之间的定时切换与处理。

(4) 并行工作。当外部设备与CPU以中断方式传送数据时，可以实现CPU与外部设备之间的并行操作。通常，快速的CPU与慢速的外部设备在传送数据的速率上存在着矛盾。CPU在中断处理程序中完成外设的请求操作后，便返回原程序继续执行下去。与此同时，外部设备接收到CPU的工作命令后，执行相应的输入/输出操作。因此，采用中断技术后，CPU在大部分时间内与外部设备并行工作，工作效率大大提高。如键盘字符的输入操作、打印件的字符输出操作，以及模拟信号的A/D转换等都要用到中断。

6.1.1 中断及中断源

1. 中断

中断是指CPU在执行程序过程中，由于外部或内部随机事件，导致CPU暂时停止正在执行的程序而转去执行一个用于处理该事件的程序——称为中断处理程序，待处理结束后，又返回被中止的程序断点继续执行，这一过程称为中断。相对被中断的原程序来说，中断处理程序是临时嵌入的一段程序，所以，一般将被中断的原程序称为主程序，而将中断处理程序称为中断子程序（或中断服务子程序）。主程序被中止的地方，称为断点，也就是下一条指令所在内存的地址。中断服务子程序的入口地址一般存放在内存中一个固定的区域内。

图6.1给出了中断过程的示意图，从图中看出CPU响应中断而导致了程序的转移过程，能实现这一中断处理过程的技术，就称为中断技术。

事实上，在日常生活中，"中断"也是很常见的。例如，当你看书时，突然电话铃

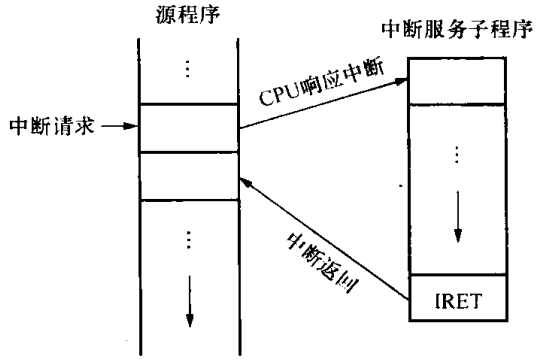


图 6.1 中断过程示意图

响了，这时你对这个事件做出反应，暂时停止看书并对当前正看的书页做出记号，然后接听电话，该事件处理完后，再回到原来中断的地方继续看书。

2. 中断源及分类

能够引起计算机中断的事件，称为中断源。中断可以是硬件或软件产生的，按照中断源与 CPU 的位置关系可分为两大类：一类来自 CPU 内部，称为内部中断源；另一类来自 CPU 外部，称为外部中断源。

（1）内部中断。内部中断是来自 CPU 内部的中断，也称为软件中断。主要包括下列情况：首先是 CPU 执行指令时产生的异常，如被 0 除、溢出、断点、单步操作等；其次是特殊操作引起的异常运行程序错误，如存储器越界、缺页等；第三类是由程序员安排在程序中的 INT n 软件中断指令。

（2）外部中断。外部中断是指由外部设备通过硬件请求的方式产生的中断，也称为硬件中断。外部中断又可分为不可屏蔽中断和可屏蔽中断。

可屏蔽中断 INTR（Interrupt Request）：CPU 对可屏蔽中断请求的响应是有条件的，它受中断允许标志位 IF 的控制。当 IF=1 时，允许 CPU 响应 INTR 请求，并发出中断响应信号$\overline{\text{INTA}}$；当 IF=0 时，禁止 CPU 响应 INTR 请求。可屏蔽中断常用于 CPU 与外设进行数据交换。

不可屏蔽中断（Non-Maskable Interrupt，NMI）：当外设通过非屏蔽中断请求信号 NMI 向微处理器提出中断请求时，微处理器在当前指令执行结束后，就立即无条件地予以响应，而与中断标志 IF 无关，这样的中断就是不可屏蔽中断。不可屏蔽中断在外部中断源中优先级最高，主要用于处理系统的意外或故障，如电源掉电、存储器读/写错误、扩展槽中输入/输出通道错误等。

6.1.2 中断的一般过程

在微机系统中，对于外部中断，中断请求信号是由外部设备产生，并施加到 CPU 的 NMI 或 INTR 引脚上，CPU 通过不断地检测 NMI 和 INTR 引脚信号来识别是否有中断请求发生。

对于内部中断，中断的控制完全是在 CPU 内部实现的，通过内部中断控制逻辑产生中断和响应中断。不同类型的中断，其中断过程略有区别。但无论是外部中断还是内部中断，中断处理过程都要经历以下步骤：中断请求及检测→中断源识别与判优→中断响应→中断处理→中断返回。下面以外部可屏蔽中断为例，简要介绍中断处理过程的 5 个步骤。

1. 中断请求及检测

当外部中断源希望 CPU 对它服务时，就产生一个中断请求信号加载到 CPU 中断请求输入端，通知 CPU，这就形成了对 CPU 的中断请求。

每个中断源向 CPU 发出的中断请求信号是随机的，而 CPU 在执行每条指令的最后一个时钟周期来检测中断请求信号。每一个中断源都设置了一个中断请求触发器，记录中断源的

请求标志。当有中断请求时，该触发器被置位；当CPU响应中断请求后，该触发器被清除。8088/8086CPU的INTR中断请求信号为电平触发（不可屏蔽中断NMI为边沿触发），INTR应保持到该请求被CPU响应为止，并且当CPU响应后，INTR信号应及时撤除，以免造成多次响应。

2. 中断源识别及判优

在微机系统中，每个中断源都有相应的中断服务子程序。当系统中有多个中断源时一旦发生中断，CPU必须确定是哪一个中断源提出了中断请求，以便获取相应的中断服务子程序的入口地址，转入中断处理。中断源识别的目的就是找到该中断服务程序的入口地址。

在微机系统中，由于中断源种类繁多、功能各异，接受CPU服务时的优先等级不同，因此，需要对中断源进行优先级排队。当多个中断源同时提出中断时，CPU首先响应优先级高的中断源提出的中断。可以通过软件查询方式和硬件电路两种方法判断中断源的优先级别。

（1）软件查询方式。软件查询方式的基本原理是：当CPU接收到中断请求信号后，通过执行优先级判优的查询程序，按优先级的高低逐个检测外设中断请求标志位的状态，CPU首先响应优先级最高的中断请求，在处理完优先级最高的中断请求后，再转去响应并处理优先级较低的中断源请求。

软件查询判优原理如图6.2所示，中断源A、B、C的中断请求信号被锁存在中断请求寄存器中，并通过或门进行“或”操作，送到CPU的INTR端，同时送入三态缓冲器（作为并行端口）供CPU查询。

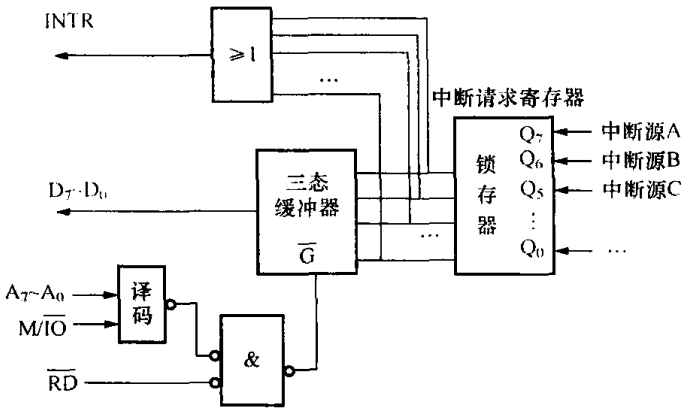


图6.2 软件查询判优原理

软件查询程序如下：

```
IN   AL,  n      ;n为并行输入接口的端口地址
ROL  AL,1
JC   FWA         ;转中断源A服务程序
ROL  AL,1
JC   FWB         ;转中断源B服务程序
ROL  AL,1
JC   FWC         ;转中断源C服务程序
.....
```

上述查询程序确定了中断源的优先级，中断源优先级从高到低顺序排列，中断源A优先级最高，中断源B次之，后面的依次排列。通过软件查询中断源和判优，硬件电路简单，中断源优先级设置灵活，但当中断源较多的时候，查询所花费的时间较长，会影响到中断响应的实时性。

（2）硬件优先级判优。硬件优先级判优是采用硬件电路来实现的，可节省CPU的时

间，而且速度较快，但是成本较高。常用的有链式判优电路，其基本思想是将所有的中断源构成一个链（称菊花链，见图 6.3），每个外设对应的端口都有一个中断控制逻辑电路。

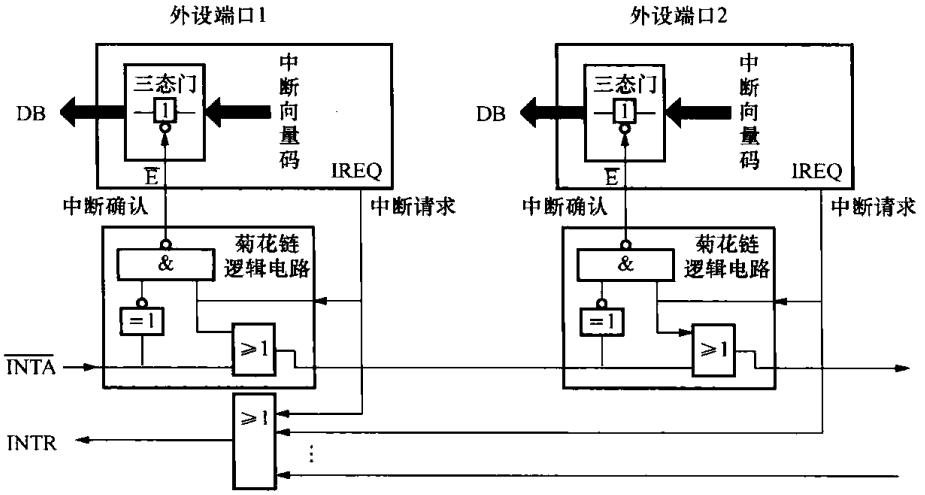


图 6.3 菊花链中断判优电路

链式电路中的所有中断源均可以发出中断请求，排在链前面的中断优先级别较高。CPU 响应中断后发出的中断响应信号$\overline{\text{INTA}}$，沿着链式电路按优先级从高到低传递，优先级别高的先得到响应。如果高优先级的中断源没有请求中断，则使$\overline{\text{INTA}}$传递到下一级。

从图 6.3 可以看出，假设外设端口 1 和外设端口 2 都有中断请求（IREQ=1），均可以向 CPU 发出中断请求。如果 CPU 允许中断，则会发出$\overline{\text{INTA}}$信号。但是，$\overline{\text{INTA}}$信号首先经过外设端口 1 的菊花链电路，并得到中断确认，并且外设端口 2 的中断响应被封锁。得到中断确认的外设端口，打开三态门，把自己的中断类型码放到数据总线上，CPU 读取中断类型码，并据此转到相应的中断服务入口地址，执行中断服务程序。

（3）可编程中断控制器判优。在 Intel 80X86 CPU 系统中，用中断控制器来识别和设置管理中断源的优先级别。中断发生前，已对每个中断源进行编号称为中断类型码，CPU 根据中断类型码来确定中断源。当有多个中断源同时请求中断时，中断控制器不仅能够自动地进行中断优先级判断，而且能够提供中断源的中断类型码，CPU 通过读取中断类型码而转去执行相应的中断服务子程序。与 8086/8088CPU 配套的中断控制器为 8259A 芯片，可对 64 级中断源进行优先级管理，该芯片将在 6.3 节中进行详细介绍。

3. 中断响应

中断优先级确定后，当有中断请求时，CPU 对当前优先级最高的中断请求进行响应。中断响应过程是暂时停止当前程序的执行，进行断点保护并从当前程序跳转到中断服务子程序的过程。CPU 响应中断，必须满足下列 4 个条件：

（1）当前执行的指令结束。CPU 在每条指令执行的最后一个时钟周期对中断请求进行检测，当满足本条件和以下三个条件时，指令执行一结束，CPU 即可响应中断。

（2）CPU 处于开中断状态。只有 CPU 的 IF=1，即处于开中断状态时，CPU 才有可能

响应可屏蔽中断（INTR）请求（对 NMI 及内部中断无此要求）。

（3）没有复位（RESET）、保持（HOLD）、内部中断和非屏蔽请求（NMI）。在复位或保持状态，CPU 不工作，不可能响应中断请求；而 NMI 的优先级高于 INTR，当两者同时产生中断时，CPU 首先响应 NMI 中断。

（4）若当前执行的是开中断指令（STI）和中断返回指令（IRET），只有它们执行完后，再执行一条指令才能响应 INTR 请求。

CPU 响应 INTR 中断，向发出中断请求的外设回送一个低电平有效的中断响应信号 $\overline{\text{INTA}}$，作为对中断请求 INTR 的应答。CPU 响应内部中断和 NMI 中断，对外不发出任何中断响应信号。CPU 响应中断后，内部要完成以下内容：

（1）关中断。内部自动关中断（即使 IF=0），以禁止接受其他的中断请求。

（2）保护断点。把断点处的 FLAGS、CS、IP 内容压入堆栈，以备中断结束后能正确地返回被中断的程序。

（3）读取中断类型码，获得中断服务程序的入口地址，执行该中断源的中断处理程序。

4. 中断处理

当 CPU 获得中断服务程序的入口地址后，则转入中断处理过程。中断服务程序在形式上与一般的子程序基本相同，区别在于：中断服务程序只能是远过程（类型为 FAR）；中断服务程序要用 IRET 指令返回被中断的程序。

中断服务程序完成的主要工作：

（1）保护现场。主程序和中断服务子程序都要使用 CPU 内部寄存器等资源，为使中断处理程序不破坏主程序中寄存器的内容，用户要使用 PUSH 指令将断点处各寄存器的内容压入堆栈。

（2）开中断。因 CPU 响应中断后已自动关中断。若要进入中断服务程序后允许中断嵌套，则需要用指令开中断（开中断指令 STI，使 IF=1）。中断嵌套指令高优先级中断源可以中断低优先级中断源的服务程序。

（3）执行中断处理程序。中断处理程序是中断服务的主要内容，不同的中断有不同的中断处理内容，根据中断源所要完成的功能来编写。

（4）关中断。当中断处理程序结束后需要关中断，以确保有效地恢复被中断的程序现场。关中断指令 CLI，使 IF=0。

（5）恢复现场。当中断处理完毕后，用户通过 POP 指令将保存在堆栈中的各个寄存器的内容弹出，即恢复主程序断点处寄存器的原值。

5. 中断返回

在中断服务子程序的最后要安排开中断指令 STI 和中断返回指令 IRET。开中断指令 STI 使中断返回后可以响应新的中断；而中断返回指令 IRET，是将堆栈内保存的 IP 和 CS 值弹出，从而恢复主程序断点处的地址值，同时还自动恢复标志寄存器 FLAGS 的内容，使 CPU 转到被中断的程序中继续执行。

上述中断处理过程流程图如图 6.4 所示。

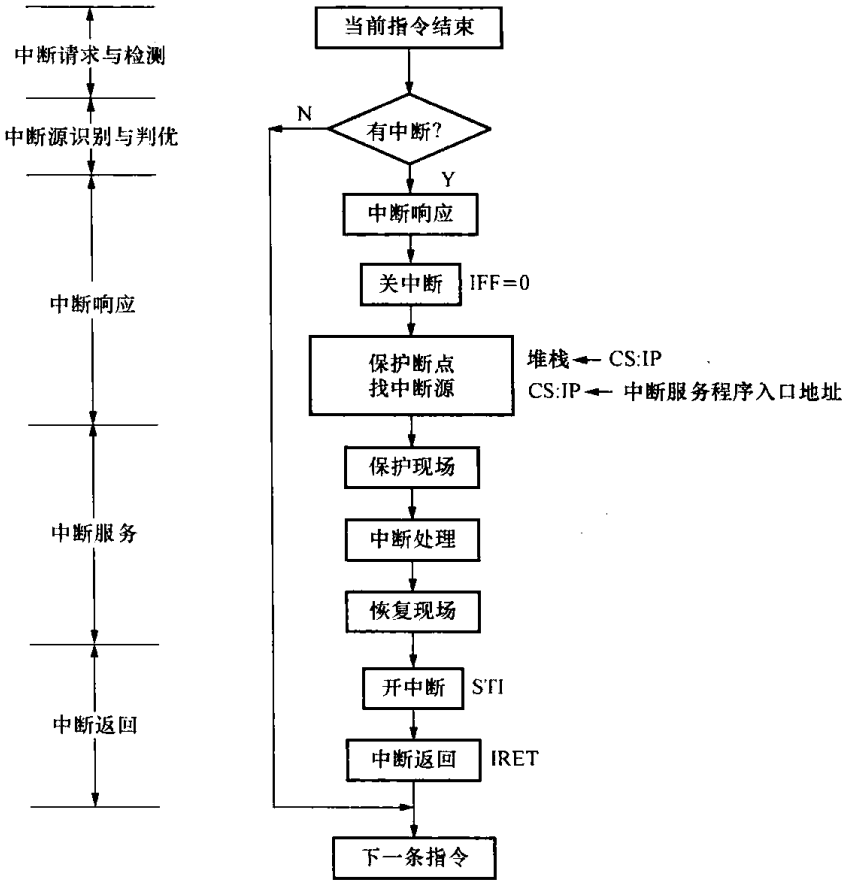


图 6.4 中断处理过程流程图

6.2 8086/8088 中断系统

6.2.1 8086/8088 中断分类

8086/8088CPU 的中断系统功能很强，使用非常灵活，它能够处理 256 种不同类型的中断。为了便于识别，给每个中断源分配一个中断类型编码 n，称为中断类型码，长度为一个字节，故 8086/8088 最多允许处理 256 种类型的中断（中断类型码为 $n=0\sim255$）。CPU 可根据中断类型码的不同来识别不同的中断源。这 256 种中断包含了内部中断源和外部中断源两大类，它们的优先级排列原则是：单步中断优先级别最低，内部中断高于外部中断，以及中断类型码越小优先级越高。例如，中断类型码为 0 号的中断源优先级最高。8086/8088 中断分类如图 6.5 所示。

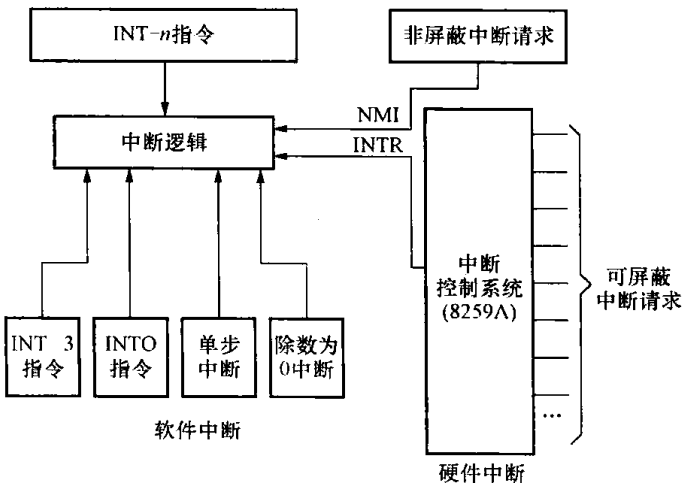


图 6.5 8086 中断分类

1. 内部中断

内部中断也称为软件中断，由 CPU 内部或中断指令引起的中断，根据中断源 CPU 自动提供中断类型码。在 8086/

8088 系统中，内部中断主要有以下 5 种：

（1）除法错中断。在 8086/8088 执行除法指令时，若除数太小，致使所得的商超过了 CPU 所能表示的数值范围，则 CPU 内部硬件电路自动产生一个中断类型码为 0 的中断。因此，除法错中断又称为 0 号中断。

（2）单步中断。在 8086/8088CPU 内部的标志寄存器中有一位 TF 标志。CPU 每执行一条指令都检测 TF 的状态。如果 TF=1，则 CPU 硬件电路自动产生一个中断类型为 01H 的中断，使 CPU 转去执行单步中断的程序。单步中断用于程序的调试，使 CPU 一次执行一条指令。

（3）断点中断。断点中断在调试程序的过程中用于设置断点。8086/8088 指令系统中有一条专门用于设置断点的指令，其操作码为单字节 CCH。CPU 执行该指令通过硬件电路产生中断类型码为 03H 的中断。

（4）溢出中断（INTO 指令）。当 CPU 进行算术运算时，如果发生溢出，则会使标志寄存器的 OF 标志置 1。如果在算术运算后加一条溢出中断指令 INTO，则溢出中断指令测试 OF 标志位，若发现 OF=1，则硬件电路产生中断类型码为 4 的中断。

（5）用户自定义的软件中断（INT n）。用户可以用 INT n 指令产生软件中断。其中 INT 为指令助记符，n 为用户指定的中断类型码。

通过上述分析可以看出，内部中断的中断类型码或者是 CPU 硬件电路产生的，或者是用户指定的。

2. 外部中断

8086CPU 有 NMI 和 INTR 两条外部中断请求信号线，分别接收非屏蔽中断和可屏蔽中断请求信号。所谓“屏蔽”是指微处理器拒绝响应中断请求信号，不允许中断当前所执行的主程序，这通常是由 CPU 内部的中断允许触发器来控制的。NMI 中断不受中断允许触发器的影响，NMI 中断优先级高于 INTR。

（1）可屏蔽中断 INTR。可屏蔽中断请求 INTR 通常被称为中断请求，它受中断允许标志 IF 的控制。当 INTR 信号有效（为高电平）时，CPU 是否响应该中断取决于 IF 的状态。如果 IF=0，CPU 屏蔽 INTR 中断；如果 IF=1，则允许 INTR 中断。开中断指令 STI 使 IF 置 1，关中断指令 CLI 使 IF 清零。

CPU 响应 INTR 中断后：首先产生两个连续的中断响应总线周期，如图 6.6 所示。在第一个中断响应周期，送出第一个中断响应$\overline{\text{INTA}}$有效信号，表明响应中断；在第二个中断响应周期，送出第二个$\overline{\text{INTA}}$有效信号，使外部中断系统将提出中断的中断源的中断类型码送上数据总线。CPU 从数据总线读入该中断类型码后，CPU 内部硬件电路进行断点保护（FLAG、CS、IP 压入堆栈），并自动关中断（IF=0）。然后，根据中断类型码，获得中断服务程序的入口地址，执行中断服务程序对中断源进行服务。

（2）不可屏蔽中断 NMI。NMI 中断是微处理器内部不能“屏蔽”的中断，不受 IF 标志的影响，称为不可屏蔽中断，常用于电源掉电、存储器读/写出错、总线奇偶位出错等重要的中断事件。NMI 中断请求能被 CPU 锁存，因此 NMI 可以采用边沿触发方式。CPU 响应 NMI 中断，硬件电路自动产生中断类型码为 2 的中断。

CPU 对 NMI 中断和内部中断的响应过程与 INTR 中断响应的方式不同：没有中断响应周期，获取中断类型码的方法不同。当 NMI 中断和内部中断产生时，首先 CPU 内

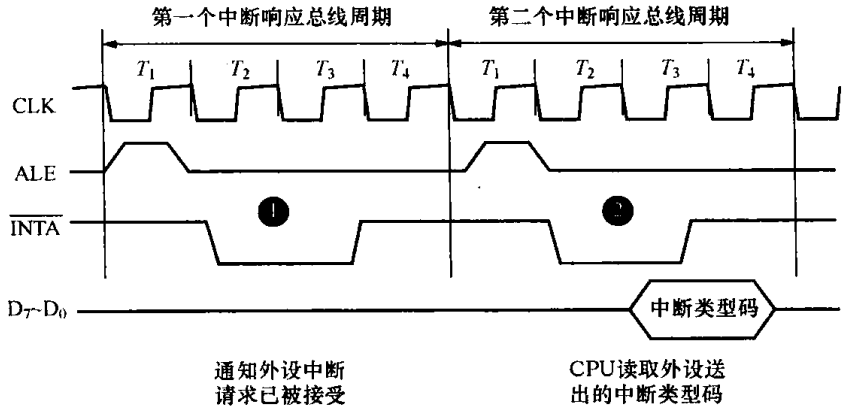


图 6.6 CPU 对 INTR 的响应过程

部硬件电路进行断点保护，关中断，然后由 CPU 硬件自动形成或由软件指令提供中断类型码。CPU 根据中断类型码决定中断服务程序的入口地址，转向相应的中断服务程序去执行。

6.2.2 CPU 中断响应过程

8086/8088 对不同类型中断的响应过程不同，主要区别在于如何获得相应的中断类型码。除法错、溢出、断点、单步及 NMI 均由 CPU 内部硬件产生；软件中断指令的中断类型码包含在指令中；INTR 的中断类型码由外设提供，CPU 从数据总线上读取。中断响应过程归纳如图 6.7 所示。

从图 6.7 中可以看出：

(1) 当前指令结束，按优先级从高到低进行查询。除单步中断外，内部中断高于外部中断，类型码越小优先级越高；外部中断 NMI 高于 INTR 中断；单步中断优先级最低。

(2) 从 CPU 中断查询源到 CPU 去执行中断服务程序这一复杂过程全部由 CPU 的硬件自动完成。当用户用中断方式解决实际问题时，主要的工作就是利用硬件电路和软件编程使 CPU 完成整个中断过程。

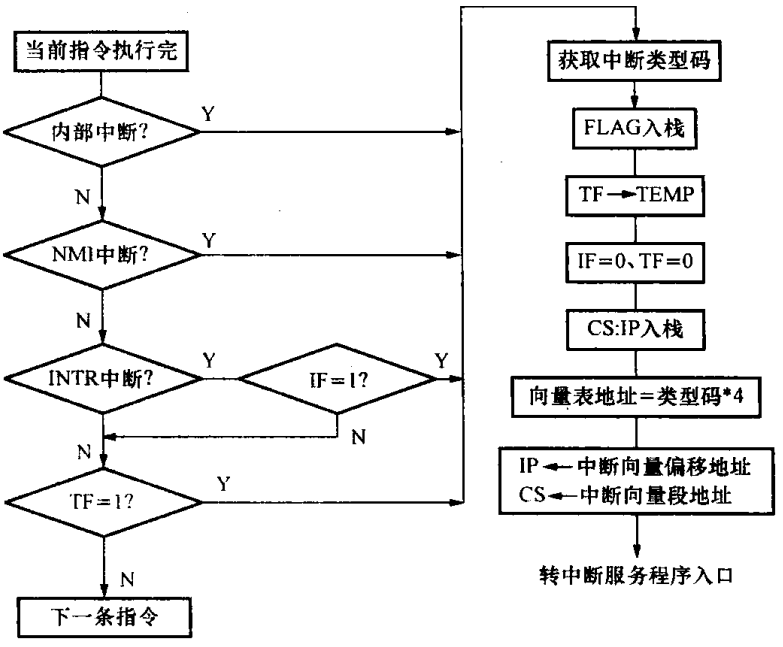


图 6.7 8086/8088 的中断响应过程

6.2.3 中断向量和中断向量表

1. 中断向量

每个中断源都有自己的中断类型码和中断服务程序，中断服务程序的入口地址又称为中断向量；每个中断向量为 4 个字节。

2. 中断向量表

为了能够根据所得到的中断类型码来找到中断服务程序的入口地址，8086/8088 系统规定所有中断服务程序的入口地址都必

须按顺序存放在一个内存固定的区域中，该区域称为中断向量表。中断向量表位于内存00000H～003FFH区域，总计1024个存储单元，可以存放256个中断源的中断向量（256个中断服务程序的入口地址），每个中断向量占4个字节，其中低位2个字节存放中断服务程序入口地址的偏移地址，高位2个字节存放中断服务程序入口地址的段基地址。按照中断类型的大小，对应的中断向量在中断向量表中有规则地顺序存放，如图6.8所示。

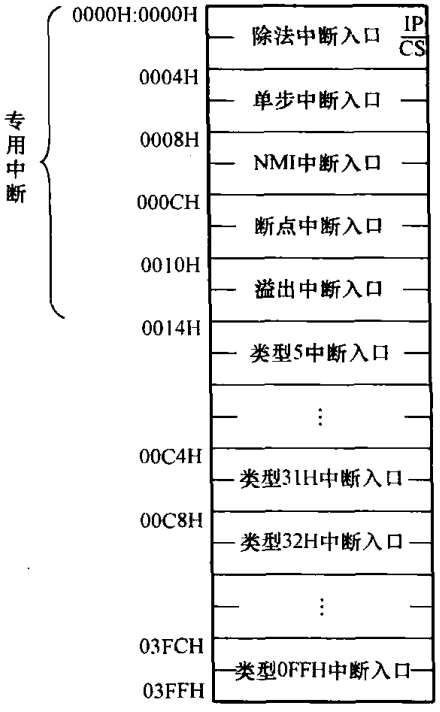


图6.8 8086中断向量表

根据中断向量表，只要知道了中断类型码n就可以找到所对应的中断向量在中断向量表中的位置。中断类型码n、中断向量以及中断向量在中断向量表中的地址的对应关系可由下式得到：

段基地址＝0000H，偏移地址＝中断类型码$n\times4$

例如，中断类型码为21H的中断，其中断向量存放在0000H：0084H（21H×4＝84H）开始的4个字节单元中。CPU响应中断时，根据中断类型码计算出所在中断向量表的地址，只要取$4n$和$4n+1$单元的内容装入IP，取$4n+2$和$4n+3$单元的内容装入CS，就可以转入中断服务程序了。

当然，要使CPU能够正确完成上述操作，中断产生之前，应通过程序初始化中断向量表，将中断类型码为n的中断向量的偏移地址存放在$4n$和$4n+1$单元，将中断向量的段地址存放在$4n+2$和$4n+3$单元。

需要说明的是，在80386以后的微机中，由于虚拟存储器级保护方式的出现，中断向量表不再是固定放在00000H～003FFH区域（中断向量表的名字也改为中断描述符IDT），而是可以位于内存的任意区域，表的首地址放在CPU内部的IDT基址寄存器中。每个中断服务程序的入口地址也从4字节增加到了8个字节，包括2字节的选择器、4字节的偏移量和2字节的其他属性。

3. *中断向量表初始化*

中断服务程序的入口地址（中断向量）必须在中断之前写入中断向量表中。常用方法如下：

方法1：直接编程填写中断向量表。例如，若某中断源的中断向量码为40H，中断服务程序的入口地址为INTF。初始化中断向量表的程序如下：

```
MOV AX,0000H
MOV DS,AX
MOV SI,0100H                ;中断向量地址:40×4=0100H
MOV  AX,OFFSET  INTF        ;取中断向量偏移地址
MOV  [SI],AX
MOV  AX,SEG  INTF           ;取中断向量段基地址
MOV  [SI+2],AX
```

方法 2：DOS 系统调用填写中断向量表。例如，若在 DOS 下工作，可采用 DOS 系统功能调用 INT 21H 的 25H 功能号，其调用方法是：AH←25H；AL←中断向量码；DS:DX←中断向量段基地址：偏移地址。程序如下：

```
MOV AH,25H                  ;功能号
MOV AL,40H                  ;中断类型码
MOV DX,SEG  INTF            ;取中断向量段基地址
MOV DS,DX
MOV DX,;OFFSET  INTF        ;取中断向量偏移地址
INT 21H
```

综上所述，中断类型码和中断向量通过中断向量表联系在一起。

6.3　可编程中断控制器 8259A

可编程中断控制器 8259A 是 Intel 公司专为 80X86 CPU 控制外部中断而设计开发的芯片。它将中断源优先级判优、中断源识别和中断屏蔽电路集于一体，不需要附加任何电路就可以对外部中断进行管理。8259A 的主要功能有：

（1）单片可以管理 8 级外部中断，9 片级联方式下可以管理多达 64 级的外部中断。

（2）对任何一个优先级别的中断源都可单独进行屏蔽设置，即屏蔽和取消屏蔽。

（3）能向 CPU 提供可编程的标识码，对于 8086/8088 的 CPU 来说就是中断类型码。这个功能可以使不能提供中断类型码的可编程接口芯片 8255A、8253、8251A 等采用中断方式。

（4）具有多种中断优先权管理方式，有完全嵌套方式、自动循环方式、特殊循环方式、特殊屏蔽方式和查询方式五种，这些管理方式均可通过程序动态地进行变化。

6.3.1　8259A 内部结构和引脚功能

可编程中断控制器 8259A 是 28 引脚双列直插式芯片，单一＋5V 电源供电，引脚信号如图 6.9 所示。

1. 8259A 引脚功能

D_7～D_0：双向、三态数据线，与 CPU 系统总线连接。编程时控制字、命令字由此写入，中断响应时，中断类型码由此送给 CPU。

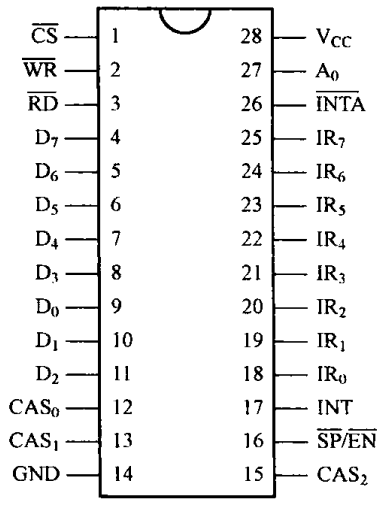


图 6.9　8259A 的引脚信号

$\overline{RD}$：读控制信号，输入，低电平有效。当$\overline{RD}$有效时，CPU 对 8259A 进行读操作。

$\overline{WR}$：写控制信号，输入，低电平有效。当$\overline{WR}$有效时，CPU 对 8259A 进行写操作。

$\overline{CS}$：片选信号，输入，低电平有效。当$\overline{CS}$有效时，8259A 被选中。

A_0：8259A 内部寄存器的选择信号。由 8259A 片内译码，选择内部寄存器。使用中，通常接地址总线的某一位，如 A_1 或 A_0。

IR_7～IR_0：为中断请求输入信号，与外设的中断请求线相连。上升沿或高电平（可通过编程设定）时，表示外设有

中断请求。

INT：为8259A中断请求输出信号，可直接接到CPU的INTR输入端。

$\overline{INTA}$：为中断响应输入信号，与CPU的中断响应信号$\overline{INTA}$相连，在中断响应过程中CPU的中断响应信号由此端进入。

CAS_2～CAS_0：为级联控制线。当多片8259A级联工作时，其中一片为主控芯片，其他均为从属芯片。对于主片8259A，其CAS_2～CAS_0为输出，对于从片8259A，其CAS_2～CAS_0为输入，主片CAS_2～CAS_0与从片CAS_2～CAS_0相连，主片通过CAS_2～CAS_0来选择和管理从片。例如，当主片IR_2端所接的从片8259A提出中断请求时，主片8259A通过CAS_2～CAS_0送出编码010给从片，使从片的中断被允许。

$\overline{SP}/\overline{EN}$：为双功能信号线，用于指定8259A是主片还是从片，或作为总线驱动器的控制信号。当8259A工作于非缓冲方式时，$\overline{SP}/\overline{EN}$作为输入信号线，用于指定8259A是主片还是从片。级联中的从片$\overline{SP}/\overline{EN}$接低电平，主片$\overline{SP}/\overline{EN}$接高电平；当8259A工作于缓冲方式时，$\overline{SP}/\overline{EN}$作为输出信号线，用作8259A与系统总线驱动器的控制信号，而缓冲器方式下的多片级联时，主从片的确定由初始化命令字ICW_4确定。

V_{CC}：+5V电源输入信号。

GND：电源地。

2. 8259A内部结构

8259A的内部结构如图6.10所示。它主要包括中断请求寄存器IRR、中断服务寄存器ISR、中断屏蔽寄存器IMR、优先权判优电路PR、中断控制逻辑、数据总线缓冲器、读/写电路和级联缓冲/比较器等。

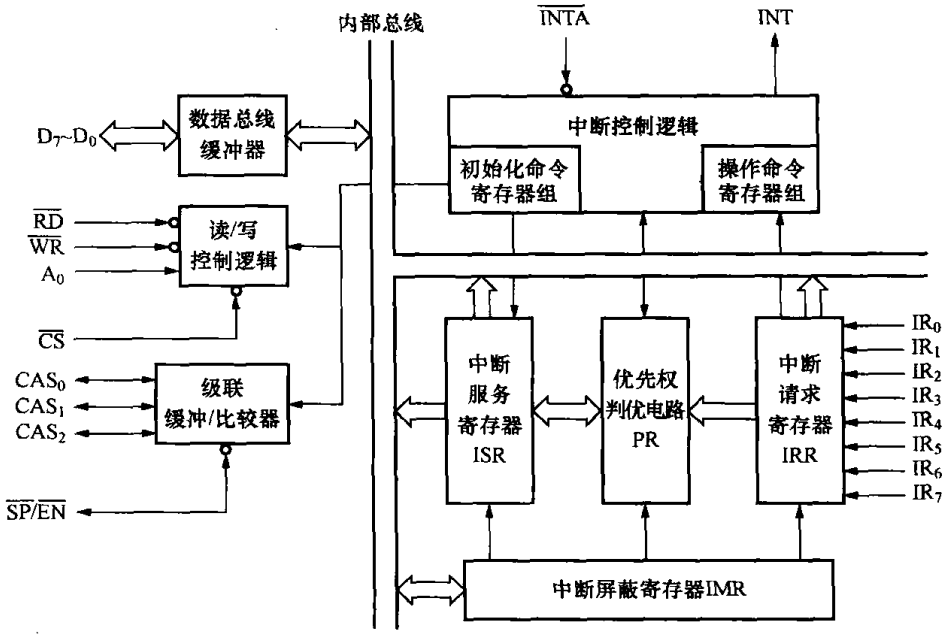


图6.10 8259A内部结构

(1) 中断请求寄存器(Interrupt Request Register，IRR)。中断请求寄存器IRR是一个8位寄存器，用于保存外部中断请求信号IR_7～IR_0的状态。当IR_i(i=0～7)有请求(电平或边沿触发)时，IRR中的相应位置1，在中断响应信号$\overline{INTA}$有效时，相应位复位。无论

采用何种触发方式，中断请求信号的高电平状态必须保持到第一个中断响应信号$\overline{INTA}$变为有效之后，否则将丢失中断请求信号。

（2）中断屏蔽寄存器（Interrupt Mask Register，IMR）。中断屏蔽寄存器 IMR 是一个 8 位寄存器，用来存放 $IR_7 \sim IR_0$ 的中断屏蔽位。中断屏蔽位用于控制 $IR_7 \sim IR_0$ 的中断请求是否允许中断。当 IMR 中的相应位为 1 时，对应的 IR_i 请求被禁止；当 IMR 中的相应位为 0 时，则允许相应的中断请求中断。它可以由软件设置或清除，通过编程设定屏蔽字，可以改变原来的优先级别。

（3）中断服务寄存器（Iinterrupt Sservice Register，ISR）。中断服务寄存器 ISR 是一个 8 位寄存器，保存正在被服务的中断源。当 CPU 正为某个中断源服务时，8259A 则使 ISR 中的相应位置“1”。当外部中断 IR_i（$i=0 \sim 7$）的请求得到 CPU 响应进入服务时，由 CPU 发来的第一个中断响应脉冲$\overline{INTA}$将 ISR 中的相应位 D_i（$i=0 \sim 7$）置 1，而 ISR 的复位则由 8259A 中断结束方式决定。若定义为自动结束方式，则由 CPU 发来的第二个中断响应脉冲$\overline{INTA}$的后沿将 D_i 复位为 0；若定义为非自动结束方式，则由 CPU 发送来的中断结束命令将其复位。

（4）优先权判优电路（Priority Register，PR）。优先权判优电路 PR 对 IRR、ISR 和 IMR 进行监测，对 IRR 中记录的内容与当前 ISR 中记录的内容进行比较，并对它们进行排队判优，以便选出当前最高优先级的中断请求。如果 IRR 中记录的中断请求的优先级高于 ISR 中记录的中断请求的优先级，则由中断控制逻辑向 CPU 发出中断请求信号 INT，中止当前的中断服务，进行中断嵌套。如果 IRR 中记录的中断请求的优先级低于 ISR 中记录的中断请求的优先级，则 CPU 继续执行当前的中断服务程序。在中断响应时，优先权判优电路要确定 ISR 寄存器哪一位应置 1，并将相应的中断类型码送给 CPU。在接到 EOI 命令时，它还要决定 ISR 的哪一位应复位。

（5）中断控制逻辑。中断控制逻辑按照编程设定的工作方式管理中断，负责向片内各部件发送控制信号，向 CPU 发送中断请求信号 INT 和接收 CPU 回送的中断响应信号$\overline{INTA}$，控制 8259A 进入中断管理状态。

（6）数据总线缓冲器为三态、双向、8 位寄存器，数据线 $D_7 \sim D_0$ 与 CPU 系统总线连接，构成 CPU 与 8259A 之间信息传送的通道。

（7）读/写控制逻辑用来接收 CPU 系统总线的读/写控制信号和端口地址选择信号，用于控制 8259A 内部寄存器的读/写操作。

（8）级联缓冲器/比较器用于多片 8259A 的管理和选择功能，其中一片为主片，其余为从片。

级联方式硬件连接如图 6.11 所示。

需要说明的是：

在级联应用中只有一片 8259A 为主片，可接 1～8 片从片。此时各从片 8259A 的 INT 将与主片 8259A 的 IR 相连接，而它们的三个级联信号 $CAS_0 \sim CAS_2$ 将分别互连起来。此时，主 8259A 在第一个$\overline{INTA}$响应周期内通过 $CAS_0 \sim CAS_2$ 送出三位识别码，而和此识别码相符的从 8259A 将在第二个$\overline{INTA}$响应周期内释放中断类型码到数据总线上，使 CPU 进入相应的中断处理程序。

级联缓冲器/比较器的第二个功能是提供一个$\overline{SP}/\overline{EN}$引脚信号，它是一个双功能信号。

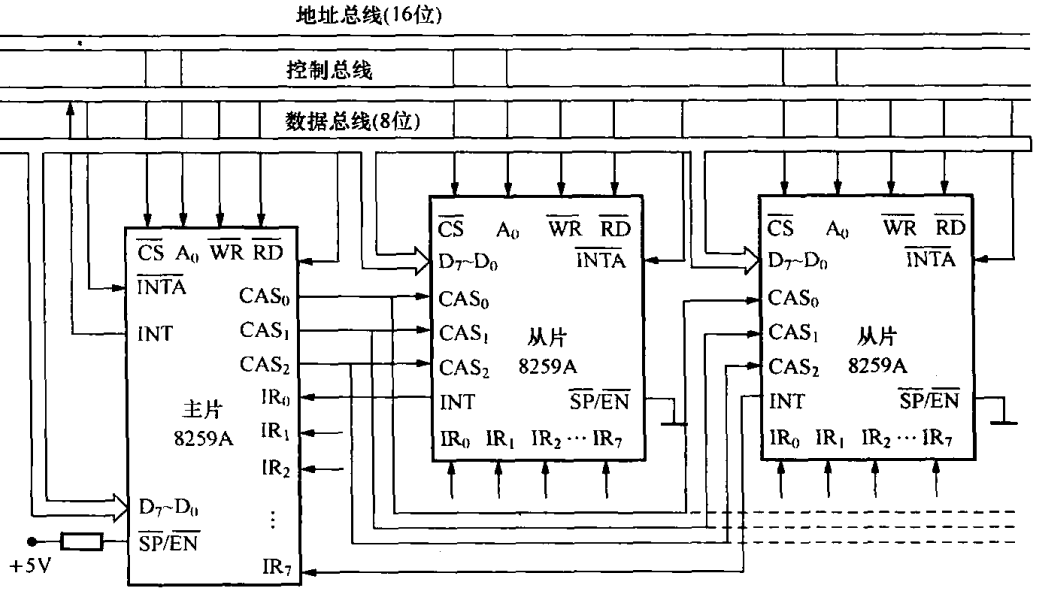


图 6.11 级联方式硬件连接图

当 8259A 工作于缓冲方式时，它作为控制系统总线缓冲器传送方向的输出信号；当 8259A 工作于非缓冲方式时，作为输入信号，用于规定该片 8259A 是作为主片（$\overline{SP}=1$）还是从片（$\overline{SP}=0$），因此一个系统中，只可能主片 8259A 的$\overline{SP}/\overline{EN}$引脚接高电平，其他从片均应接地。

此外，8259A 只有两个端口地址，它们由片选$\overline{CS}$和 A_0 输入的信号决定，一般片选$\overline{CS}$应连接到 I/O 地址译码器的输出，而 A_0 通常是和地址总线 A_0 或 A_1 相连接。当 8259A 应用于 8088CPU 时，A_0 端常和地址总线 A_0 相连接；当 8259A 应用于 8086CPU 时，A_0 端常和地址总线 A_1 相连接。

3. 8259A 可编程寄存器

8259A 可编程寄存器分为两类：一类是初始化编程寄存器，有 ICW_1、ICW_2、ICW_3 和 ICW_4；另一类是命令编程寄存器，有 OCW_1、OCW_2 和 OCW_3。8259A 寄存器的端口地址分配及读/写操作功能如表 6.1 所示。

6.3.2 8259A 的工作过程

8259A 的工作过程就是微机系统响应硬件可屏蔽中断的过程，这一过程如下：

（1）当 $IR_0 \sim IR_7$ 引脚上产生有效的中断请求信号时，则使中断请求寄存器对应位置 1。

（2）在中断屏蔽寄存器的管理下，没有被屏蔽的中断请求信号被送到优先权判优电路。

（3）在优先权判优电路中，与当前级别最高的中断源进行比较，当优先级高时，则从引脚 INT 向 CPU 发出中断请求信号。

（4）当 CPU 内部 IF=1 允许中断、而

表 6.1 8259A 端口分配及读/写操作功能表

$\overline{CS}$	$\overline{WR}$	$\overline{RD}$	A_0	D_4	D_3	功 能
0	0	1	0	1	×	写 ICW_1
0	0	1	1	×	×	写 ICW_2
0	0	1	1	×	×	写 ICW_3
0	0	1	1	×	×	写 ICW_4
0	0	1	1	×	×	写 OCW_1
0	0	1	0	0	0	写 OCW_2
0	0	1	0	0	1	写 OCW_3
0	1	0	0	×	×	读 IRR
0	1	0	0	×	×	读 ISR
0	1	0	1	×	×	读 IMR
0	1	0	1	×	×	读状态寄存器

注 D_4、D_3 为对应寄存器中的标志位。

且没有其他高优先级中断时，CPU 响应中断后，向 8259A 发出两个连续的中断响应周期信号$\overline{\text{INTA}}$；

（5）8259A 收到第 1 个中断响应信号$\overline{\text{INTA}}$之后，立即使中断服务寄存器与中断源对应的那一位置 1，同时把中断请求寄存器中的相应位清 0。

（6）8259A 收到第 2 个中断响应信号之后，把该中断源类型码 n 送上数据总线，供 CPU 读取。

（7）在实模式下，CPU 从 $4n\sim4n+3$ 单元取出该中断源的中断向量传送给 IP 和 CS，从而使 CPU 执行该中断源的中断服务程序。

6.3.3 8259A 的中断管理方式

8259A 的中断管理方式涉及 5 个主要方面的内容，即优先级管理方式、中断屏蔽方式、中断触发方式、中断结束方式和总线连接方式。每一种方式又有几种模式可供选择，如图 6.12 所示。

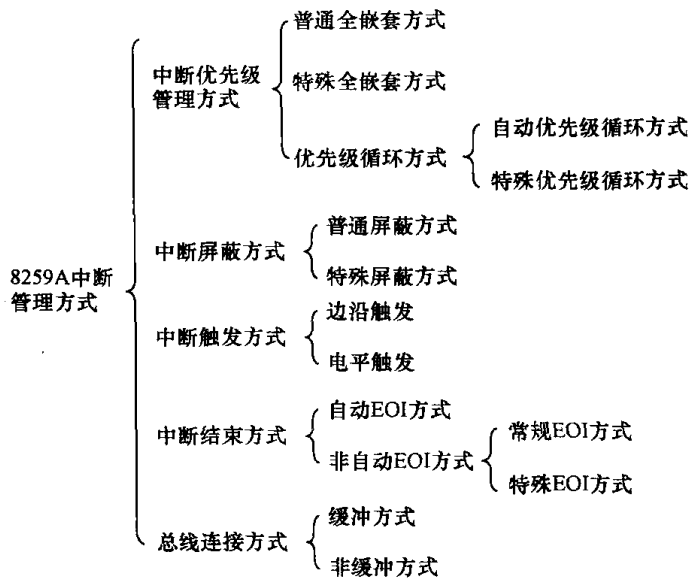


图 6.12 8259A 的中断管理方式分类

1. 中断优先级管理方式

8259A 对中断源的优先级管理主要有普通全嵌套方式、特殊全嵌套方式、优先级循环方式三种。三种方式中都允许中断嵌套，即允许更高优先级的中断打断当前的中断处理过程。

（1）普通全嵌套方式。普通全嵌套方式是 8259A 在初始化时自动进入的一种最基本的优先权管理方式，简称全嵌套方式。其特点是：中断优先权管理为固定方式，又称固定优先级方式，即 IR_0 优先权最高，IR_7 优先权最低。当 CPU 响应中断时，8259A 将申请中断的中断源中优先级最高的那个中断源在 ISR 中的相应位置 1，并且把它的中断类型码送到数据总线，在此中断源的中断服务程序完成之前，与它同级或优先级更低的中断源的申请被屏蔽，只有优先级比它高的中断源的申请才被允许。

（2）特殊全嵌套方式。特殊完全嵌套方式是 8259A 在多片级联方式下使用的一种最基本的优先权管理方式。特殊完全嵌套方式仍然是固定优先级方式，与普通全嵌套方式的区别是：当处理某一级中断时，如果有同级的中断请求，8259A 也会给予响应，从而实现一个中断处理过程能被另一个同等级别的中断请求所打断。

来自同一从片的两个中断源针对主片来说称为同级中断。在级联方式下，主片通常设置为特殊全嵌套方式，从片设置为普通全嵌套方式。当主片为某一个从片的中断请求服务时，从片中的 $IR_7\sim IR_0$ 的请求都是通过主片中的某个 IR_i 请求引入的。因此从片的 $IR_7\sim IR_0$ 对于主片 IR_i 来说，它们属于同级，只有主片工作于特殊全嵌套方式时，从片才能实现普通全嵌套方式。

（3）优先级循环方式。在实际应用中，许多中断源的优先级是一样的，若采用固定优先

级，则低级别的中断源的中断请求有可能总是得不到服务。解决的办法是使这些中断源轮流处于最高优先级。这就是优先级循环方式。

在优先级循环方式中，IR_0～IR_7 的中断优先级顺序是变化的，又分为自动循环优先级和特殊优先级循环方式。

在自动循环优先级方式中，中断优先级顺序从高到低初始化为 IR_0～IR_7，由此自动循环。但优先级顺序是可以改变的。其变化规律是：当某一个中断请求 IR_i 服务结束后，该中断的优先级自动降为最低，而紧跟其后的中断请求 $IR_{(i+1)}$ 的优先级自动升为最高。例如，IR_3 有中断请求，该中断被处理后，优先级变为最低，IR_4 优先级变为最高，这时优先级顺序从高到低变为 IR_4、IR_5、IR_6、IR_7、IR_0、IR_1、IR_2、IR_3。因此，队列中的任何一个中断源都有享受最高优先级的资格，这种自动循环方式又称为“等优先级方式”。

特殊优先级循环方式，是通过在主程序或中断服务程序中发出特殊优先级循环方式操作命令来指定某个中断源的优先级为最低级，其后的中断源优先级为最高，其余中断源的优先级顺序也随着循环变化。例如，指定 IR_3 优先级最低，则确定的优先级顺序从高到低依次排列为 IR_4、IR_5、IR_6、IR_7、IR_0、IR_1、IR_2、IR_3。此时，如果 IR_1 和 IR_6 有中断请求，则先处理 IR_6 中断。在 IR_6 被服务以后，IR_6 自动降为最低优先级，IR_7 成为最高优先级，这时中断源的优先级顺序为 IR_7、IR_0、IR_1、IR_2、IR_3、IR_4、IR_5、IR_6。

2. 中断屏蔽方式

中断屏蔽方式是对 8259A 的外部中断源 IR_7～IR_0 实现屏蔽的一种中断管理方式，有普通屏蔽方式和特殊屏蔽方式两种。

（1）普通屏蔽方式。普通屏蔽方式是通过 8259A 的中断屏蔽寄存器（IMR）来实现对中断请求 IR_i 的屏蔽。由编程写入操作命令字 OCW_1，将 IMR 中的 D_i 位置 1，以达到对 IR_i（i=0～7）中断请求的屏蔽。在 80X86 微机系统中采用普通屏蔽方式。

（2）特殊屏蔽方式。在通常情况下，当一个中断被响应时，则自动禁止同级和较低级别的中断源中断请求。但是如果希望一个中断服务程序能动态地改变系统的优先权结构，允许低优先级中断请求中断正在服务的高优先级中断。这种屏蔽方式通常用于级联方式中的主片，对于同一个请求 IR_i 上连接有多个中断源的场合，可以通过编程写入操作命令字 OCW_3 来设置或取消。

在特殊屏蔽方式中，可在中断服务子程序中用中断屏蔽命令来屏蔽当前正在处理的中断，同时可使 ISR 中的对应当前中断的相应位清 0，这样一来不仅屏蔽了当前正在处理的中断，而且也真正开放了较低级别的中断请求。在这种情况下，虽然 CPU 仍然继续执行较高级别的中断服务子程序，但由于 ISR 中对应当前中断的相应位已经清 0，如同没有响应该中断一样。所以，此时对于较低级别的中断请求，8259A 仍然能产生 INT 中断请求，CPU 也会响应较低级别的中断请求。

值得注意的是，在这种方式下，由于打乱了正常的全嵌套结构，被处理的程序不见得是当前优先级最高的事件，所以不能用正常的 EOI（End of Interrupt）命令时期 ISR 位复位。

3. 中断触发方式

8259A 中断请求输入端 IR_7～IR_0 的触发方式有电平触发和边沿触发两种，由初始化命令字 ICW_1 中的 LTIM 位来设定。

（1）边沿触发方式。当 IR_0～IR_7 出现低电平到高电平的跳变，表示有中断请求。在

80X86 微机系统中采用边沿触发方式。

（2）电平触发方式。当 $IR_0 \sim IR_7$ 出现高电平时，表示有中断请求，其高电平持续的时间必须保持到 CPU 响应中断，而且保持到 8259A 收到第 1 个中断响应脉冲之前，否则本次中断可能丢失。而当中断服务结束，ISR 相应位被清 0 之前，IR_0 的高电平必须撤销，否则可能引起第 2 次中断。

4. 中断结束方式

中断结束方式是指 CPU 为某个中断请求服务结束后，应及时清除中断服务标志位，否则就意味着中断服务还在继续，致使比它优先级低的中断请求无法得到响应。中断服务标志位存放在中断服务寄存器（ISR）中，当某个中断源 IR_i 被响应后，ISR 中的 D_i 位被置 1，服务完毕应及时清除。

8259A 提供了以下三种中断结束方式：

（1）自动结束方式。自动结束方式是利用中断响应信号$\overline{INTA}$的第二个负脉冲的后沿，将 ISR 中的中断服务标志位清除。

这种中断服务结束方式是由硬件自动完成的，需要注意的是：ISR 中为“1”位的清除是在中断响应过程中完成的，并非中断服务子程序的真正结束，若在中断服务子程序的执行过程中有另外一个比当前中断优先级低的请求信号到来，因 8259A 并没有保存任何标志来表示当前服务尚未结束，致使低优先级中断请求进入，打乱正在服务的程序，因此这种方式只适合用在没有中断嵌套的场合。

（2）普通结束方式。普通结束方式是通过在中断服务子程序中编程写入操作命令字 OCW_2，向 8259A 传送一个普通 EOI 命令（不指定被复位的中断的级号）来清除 ISR 中当前优先级别最高位。

由于这种结束方式是清除 ISR 中优先级最高的那一位，适合使用在完全嵌套方式下的中断结束。因为在完全嵌套方式下，中断优先级是固定的，8259A 总是响应优先级最高的中断，保存在 ISR 中的最高优先级的对应位，一定对应于正在执行的服务程序。

（3）特殊结束方式。特殊结束方式是通过在中断服务子程序中编程写入操作命令字 OCW_2，向 8259A 传送一个特殊 EOI 命令（指定被复位的中断的级号）来清除 ISR 中的指定位。

由于在特殊 EOI 命令中明确指出了复位 ISR 中的哪一位，不会因嵌套结构出现错误。因此，它可以用于完全嵌套方式下的中断结束，更适用于嵌套结构有可能遭到破坏的中断结束。

5. 总线连接方式

8259A 数据线与微机系统数据总线的连接有两种方式：

（1）缓冲器方式。如果 8259A 通过总线驱动器和系统数据总线连接，此时，8259A 应选择缓冲方式。当定义为缓冲器方式后，$\overline{SP}/\overline{EN}$即为输出引脚。在 8259A 输出中断类型号的时候，$\overline{SP}/\overline{EN}$输出一个低电平，用此信号作为总线驱动器的启动信号。缓冲器方式如图 6.13 所示。

（2）非缓冲器方式。如果 8259A 数据线与系统数据总线直接相连，那么 8259A 工作在非缓冲器方式。非缓冲器方式如图 6.14 所示。

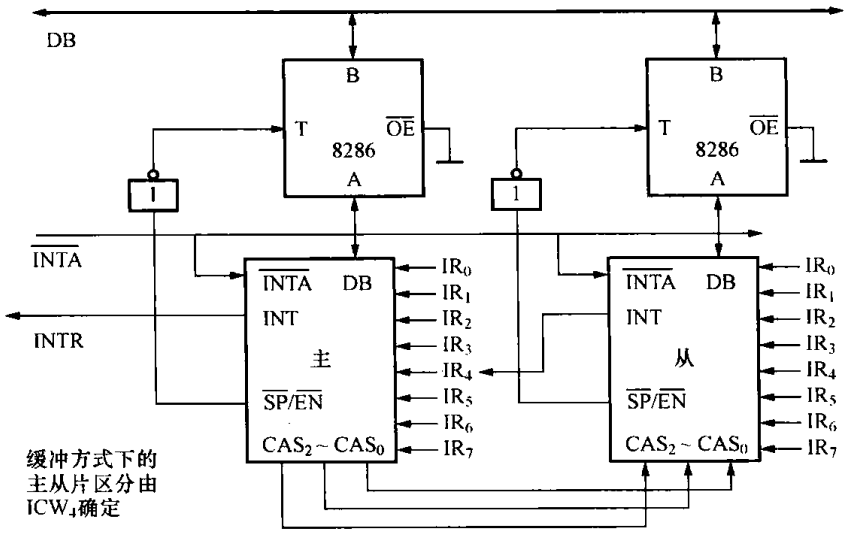


图 6.13 缓冲器方式

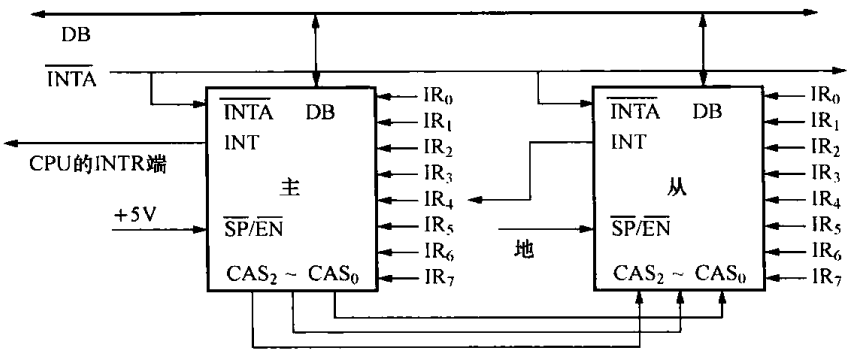


图 6.14 非缓冲器方式

6.3.4 8259A 的初始化及编程

8259A 在工作之前必须通过软件向其写入控制命令的方式来让它工作在我们所希望的状态下，这就是 8259A 的编程。控制命令分为初始化命令（Initialization Command Words，ICW）和操作命令（Operation Command Words，OCW）。在 8259A 的内部有两组可编程寄存器来存放初始化命令和操作命令。

1. *初始化命令 ICW 格式及编程*

初始化命令字共有四个，它们称为 ICW_1、ICW_2、ICW_3、ICW_4，用来对 8259A 进行初始化编程。它主要完成以下几项任务：

1）设定中断请求信号的有效方式，是高电平有效还是上升沿有效。

2）确定 8259A 工作于单片方式还是工作于级联方式。

3）若为级联工作方式，规定主片 8259A 中每个 IR 端是否带从片 8259A，从片 8259A 则要规定由主片 8259A 的哪个 IR 端引入。

4）设置中断类型码。

5）设定中断管理方式。

（1）初始化命令字。初始化命令字 ICW_1～ICW_4 的格式如图 6.15 所示。

1）ICW_1——芯片控制字。ICW_1 各位的功能如图 6.15（a）所示。标志位 D_4 恒为 1，

表示该控制字为 ICW_1，并要求写入地址 $A_0=0$ 的偶端口地址中。

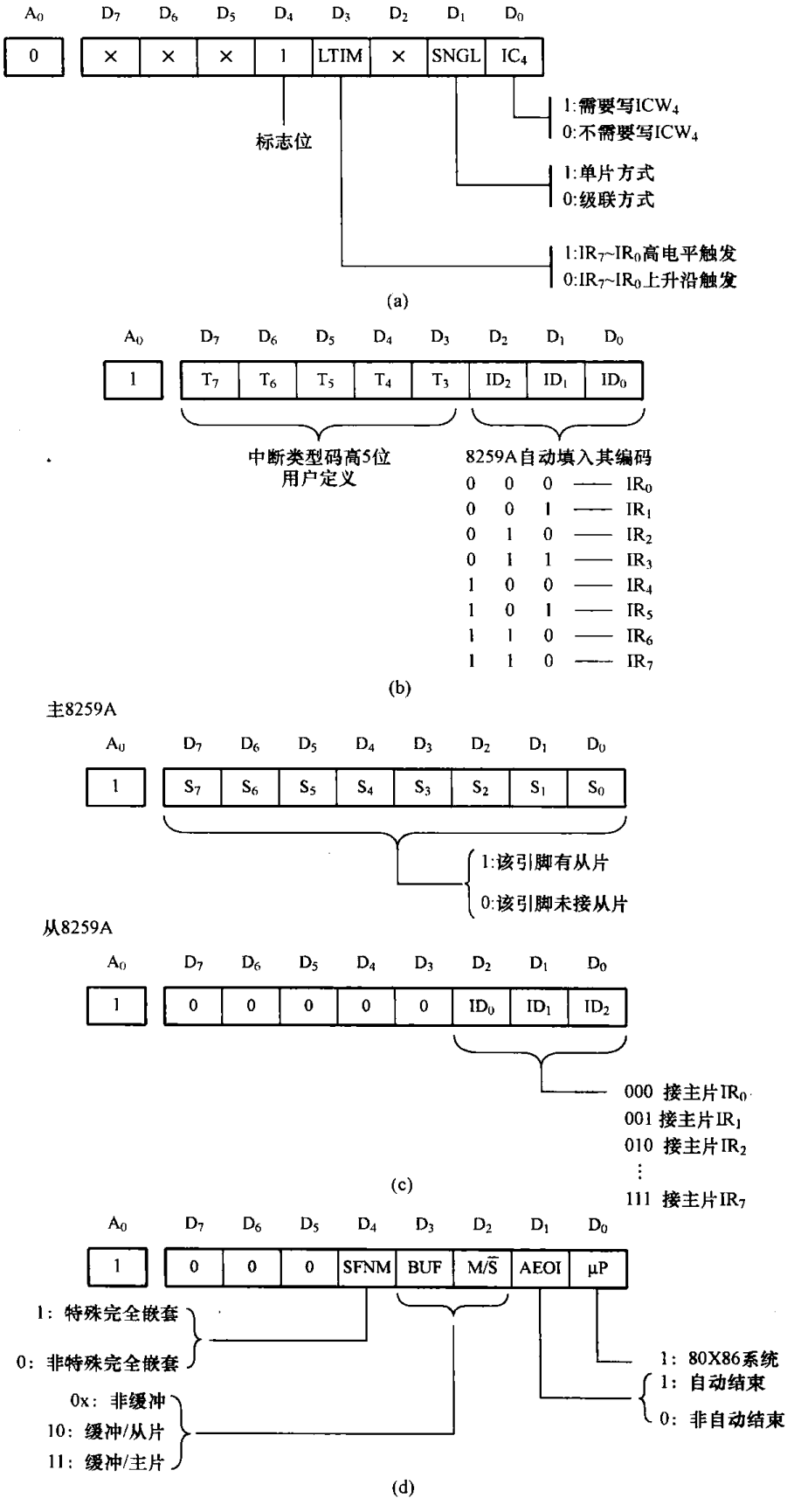


图 6.15 初始化命令字 ICW 格式

(a) ICW_1 格式；(b) ICW_2 格式；(c) ICW_3 格式；(d) ICW_4 格式

D_0(IC_4) 位：用来说明是否需要写入初始化命令字 ICW_4。$IC_4=1$，表示需要写入 ICW_4，在 80X86 CPU 系统中需要写入 ICW_4；其他 CPU 系统不需要写入 ICW_4，使$IC_4=0$。

D_1(SNGL) 位：用来确定系统中使用 8259A 的数量。SNGL＝1，使用单片 8259A；SNGL=0，使用多片 8259A，级联方式。

D_3(LTIM) 位：用来确定中断源 IR_0～IR_7 的触发方式。LTIM＝1，高电平触发；LTIM=0，低电平触发。

D_4 位：ICW_1 命令标示位，$D_4=0$，表示该控制字为初始化命令字 ICW_1。

D_2、D_5～D_7 未用，通常设置为 0。

说明：写入 ICW_1 的同时，8259A 完成以下工作：

①清除 ISR 和 IMR。

②将中断优先级设定为初始状态：IR_0（最高）～IR_7（最低）。

③设定为普通屏蔽方式。

④采用非自动 EOI 中断结束方式。

⑤状态读出电路预置为读 IRR。

2）ICW_2——设置中断类型码。ICW_2 用来确定中断源的中断类型码。ICW_2 各位的功能如图 6.15（b）所示。ICW_2 要求写入地址 $A_0=1$ 的奇地址的端口中。

D_2～D_0（ID_2 ID_1 ID_0）位：根据被处理的中断源 IR_7～IR_0 的编码自动填入。例如，IR_2 请求中断被响应，则 $ID_2ID_1ID_0=010$。

D_7～D_3（T_7～T_3）位：由用户编程写入中断类型码高 5 位。例如，编程设置 $T_7T_6T_5T_4T_3=01000$，当 IR_0 中断被响应，8259A 提供的 IR_0 中断类型码为 40H；当 IR_1 中断被响应，8259A 提供的 IR_1 中断类型码为 41H；依次类推。通常，设定 IR_0 的中断类型码，其他中断源的中断类型码按顺序排列。例如，ICW_2 被初始化为当编程 $ICW_2=40H$，则指定的 IR_0～IR_7 的中断类型码为 40H～47H。

3）ICW_3——级联命令字。ICW_3 仅在级联方式下才需要写入。主片和从片所对应的 ICW_3 的格式不同。分别写入主片、从片对应地址 $A_0=1$ 的奇端口地址中。各位的功能如图 6.15（c）所示。

主片 D_7～D_0（S_7～S_0）位：当 $S_i=1$，表示对应的 IR_i 线上连接了从片，当 $S_i=0$，表示对应的 IR_i 线上未连接从片。

从片 D_2～D_0（ID_2～ID_0）位：表示该从片接入主片的引脚编码。D_3～D_7 没有使用，通常设置为 0。

例如，当从片的中断请求信号 INT 与主片的 IR_2 连接时，从片 ICW_3 的 ID_2～ID_0 应设置为 010；而主片 IR_2 位应为 1。

在中断响应时，主片通过级联信号线 CAS_2～CAS_0 送出被允许中断的从片的编码，各从片用自己的 ICW_3 和 CAS_2～CAS_0 进行比较，两者一致的从片被确定为当前中断源，可以发送该从片的中断类型码。

4）ICW_4——中断结束方式字。ICW_4 要求写入地址 $A_0=1$ 的奇端口地址，各位的功能如图 6.15（d）所示。

$D_0=1$：8259A 用于 80X86 系统。

D_1（AEOI）位：自动中断结束方式控制。AEOI＝1，设置为自动中断结束方式；AE-

OI=0，设置正常中断结束方式。

D_3，D_2（BUF，M/S）位：缓冲器方式设置。BUF=0，非缓冲器方式，BUF=1，缓冲器方式；缓冲器方式下 M/S=0，设置主片缓冲器方式，M/S=1，设置从片缓冲器方式。

D_4（SFNM）位：中断嵌套方式设置。

需要注意：当多片 8259A 级联时，若在 8259A 的数据线与系统总线之间加入总线驱动器，以增大驱动能力。这时$\overline{SP}/\overline{EN}$引脚作为总线驱动器的控制信号，$D_3$ 位 BUF 应设置为 1，并且主片和从片的区分不能依靠$\overline{SP}/\overline{EN}$引脚，而是由 M/$\overline{S}$来选择，当 M/$\overline{S}$=0 时为从片；当 M/$\overline{S}$=1 时为主片。如果 BUF=0，则 M/$\overline{S}$定义无意义。

（2）初始化编程。初始化编程指在 8259A 工作之前，必须写入初始化命令字使其处于准备就绪状态。通过 CPU 向 8259A 送 2～4 个字节的初始化命令字 ICW。初始化命令必须按顺序写入，即先写入 ICW_1，接着写 ICW_2、ICW_3（仅在级联方式下写入），最后写入 ICW_4。写入顺序如图 6.16 所示。

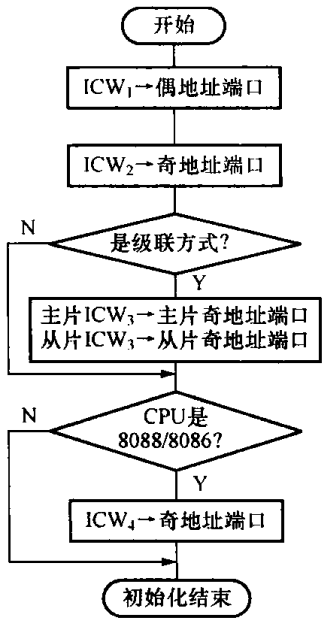


图 6.16　8259A 初始化流程图

ICW_1 和 ICW_2 是必须输入的，ICW_3、ICW_4 是否要输入则由 ICW_1 的相应位来决定。当 ICW_3 的 SNGL 位＝“0”时，需要 ICW_3 分别对主片和从片编程，其格式是不同的。当 ICW_1 的 IC_4＝“1”时，需要输入 ICW_4。

对于 8086/8088 系统，ICW_4 总是需要的。CPU 向 8259A 写入命令时，A_0＝0 和 D_4＝1 标志着写入 ICW_1，初始化过程开始。随后写入的初始化命令字由 A_0＝1 作为标志。初始化过程结束后，才能写入操作控制字。

要求 ICW_1 写入地址 A_0＝0 的偶地址端口中，ICW_2～ICW_4 写入地址 A_0＝1 的奇地址端口中。一旦初始化完成以后，若要改变某一个初始化命令字，则必须重新再进行初始化编程，不能只写入单独的一个初始化命令字。

2. 操作控制字 OCW 及编程

在对 8259A 用初始化命令字进行初始化后，就进入工作状态，准备好接收 IR 输入的中断请求信号。在 8259A 工作期间，通过操作控制字来改变 8259A 的中断控制方式、屏蔽某些中断源，以及读出 8259A 的工作状态信息（IRR/ISR/IMR）。操作命令字在初始化完成后任意时刻均可写入，可以在主程序中写入，也可以在中断服务子程序中写入，视需要而定。

各操作命令字写入的端口地址也有严格的规定，OCW_1 必须写入奇地址（A_0＝1），而 OCW_2 和 OCW_3 必须写入偶地址（A_0＝0）。操作控制字格式如图 6.17 所示。

（1）OCW_1——中断屏蔽字。OCW_1 写入奇地址（A_0＝1）端口，如图 6.17（a）所示。OCW_1 写入中断屏蔽寄存器（IMR）中，用于决定外部中断请求信号 IR_i 是否被屏蔽。当设置某位 M_i＝1 时，则对应的 IR_i 请求被屏蔽；当 M_i＝0 时，则对应的 IR_i 请求被允许。在 8259A 工作期间可根据需要随时写入或读出。

（2）OCW_2——设置中断优先级方式和中断结束方式。OCW_2 的作用是对 8259A 发出中断结束命令 EOI，它还可以控制中断优先级的循环。其格式如图 6.17（b）所示。OCW_2 写入偶地址（A_0＝0）端口，与 OCW_3 共用端口地址，但其特征位 D_4、D_3 为 00，因此不会发

生混淆。

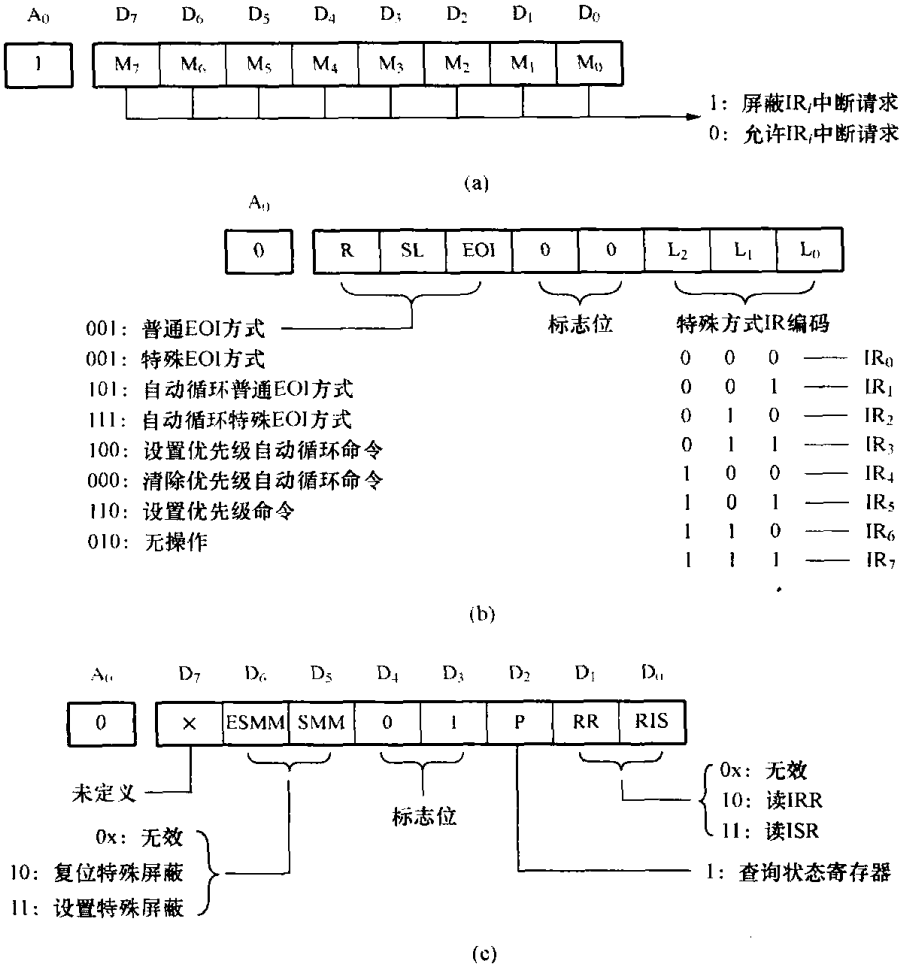


图 6.17 操作命令字格式

(a) OCW_1 格式；(b) OCW_2 格式；(c) OCW_3 格式

D_7（R）位：优先权循环方式控制位。R=1 为优先级自动循环方式，当一个中断源的中断请求被处理，则它的优先级就变为最低，而下一个中断源的优先级变为最高；R=0 为优先权固定方式，IR_0 为最高优先级，IR_7 为最低优先级。

D_6（SL）位：特殊循环控制。当 SL=1 时，使 L_2～L_0 编码有效，L_2～L_0 编码对应的 IR_i 为最低优先级；当 SL=0 时，L_2～L_0 编码无效。

D5（EOI）位：正常中断结束命令，又称为 EOI 命令。若 EOI=1 时，8259A 正常中断结束，使中断服务寄存器（ISR）中当前最高优先级的对应位置 0。正常中断结束命令是在中断服务子程序结束返回主程序前向 8259A 回送，以便允许其他中断源申请中断。正常中断结束方式是通过 ICW_4 的 AEOI=0（非自动 EOI）进行设置的。特殊中断结束命令是使 L_2～L_0 确定的 ISR 位置 0。

D_4、D_3 位：D_4D_3=00，表示该控制字为操作命令字 QCW_2。

D_2～D_0（L_2～L_0）位：L_2～L_0 第一个作用是特殊循环方式下设定哪个 IR_i 优先级最低，用来改变 8259A 复位后所设置的默认优先级；第二个作用是在特殊中断结束方式下结束哪个中断源的中断，即 ISR 哪一位要被置 0。

（3）OCW_3——设置或清除特殊屏蔽方式和读取寄存器的状态。OCW_3 写入偶地址

($A_0=0$) 端口中，OCW_3 格式如图 6.17 (c) 所示。

D_6、D_5 (ESMM，SMM) 位：设置或取消特殊屏蔽方式。当 ESMM=1，SMM=1 时，设置特殊屏蔽方式，8259A 可以响应未被屏蔽的中断源中断；若再使 ESMM=1，SMM=0 时，表示取消特殊屏蔽方式。

D_4、D_3 位：$D_4D_3=10$，表示该控制字为操作命令字 QCW_3。

D_2 (P) 位：为中断状态查询位。

当 CPU 禁止中断或不通过 8259A 向 CPU 申请中断时，就可以采用中断查询方式，常用在中断源超过 64 个的情况。

CPU 先写一个 P=1 的 OCW_3 给 8259A 的偶端口，再对该端口进行读，读取的是最高级中断请求状态。中断状态寄存器如图 6.18 所示。

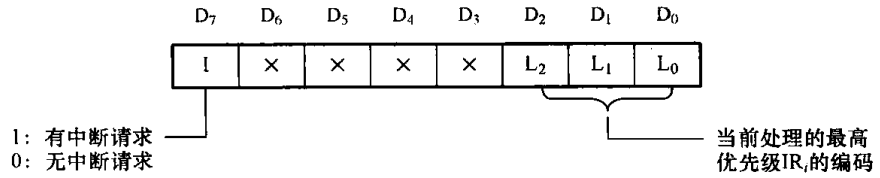


图 6.18　中断状态寄存器

若 I=1，表示有中断；I=0，表示无中断请求产生。$L_2\sim L_0$ 是最高优先级的 IR 编码(此查询步骤可反复执行，以响应多个中断同时发生的情况)。

例如，在读取中断状态字时，先写入中断查询命令，然后读取中断状态字，程序如下：

```
MOV  AL,00001111B
OUT  20H,AL          ;读中断状态字命令写入 OCW3
IN   AL,20H          ;读中断状态字
```

D_1、D_0 (RR，RIS) 位：读 IRR 还是 ISR 的操作命令。RR 称为读 ISR 和 IRR 的命令位；RIS 称为读 ISR 和 IRR 的选择位，ISR 和 IRR 共用一个端口地址。

需要读取 IRR 内容时：使 CPU 先写一个 RR=1，RIS=0 的 OCW_3 到 8259A 偶端口地址，再对该端口读，读取的是 IRR 内容。需要读取 ISR 内容时：使 CPU 先写一个 RR=1，RIS=1 的 OCW_3 到 8259A 偶端口地址，再对该端口读，读取的是 ISR 内容。

需要说明的是，如果对奇端口直接进行读操作，读取的是 IMM 内容。

例如，设 8259A 的两个端口地址为 20H 和 21H，OCW_3、ISR 和 IRR 共用一个地址 20H。读取 ISR 内容的程序段为：

```
MOV  AL,00001011B
OUT  20H,AL          ;读 ISR 命令写入 OCW3
IN   AL,20H          ;读 ISR 内容至 AL 中
```

读取 IRR 内容的程序段为：

```
MOV  AL,00001010B
OUT  20H,AL          ;读 IRR 命令写入 OCW3
IN   AL,20H          ;读 IRR 内容至 AL 中
```

3. 8259A 初始化编程

[例 6.1] 如图 6.19 所示，微机系统使用一片 8259A 实现中断管理，将其设计为主片结构，可处理 8 个外部中断。在 I/O 地址空间中，分配给 8259A 的 I/O 端口地址为 20H 和 21H。对 8259A 的初始化规定：边沿触发方式，非缓冲器方式，中断结束为 EOI 命令方式，中断优先级管理采用全嵌套方式。

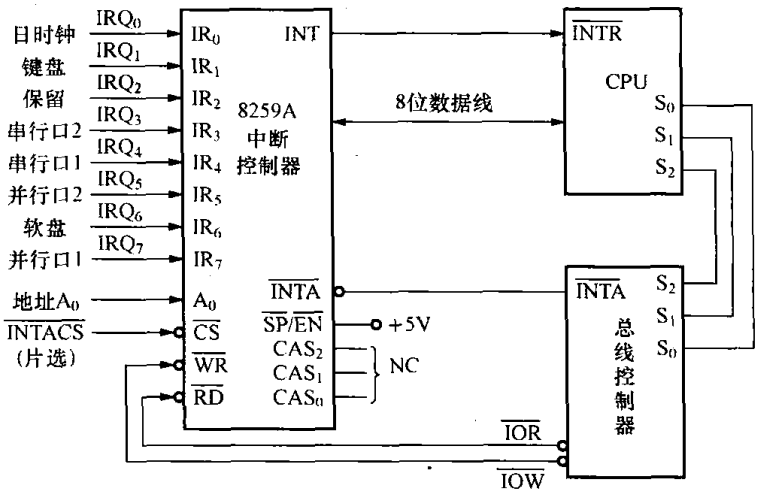


图 6.19 8259A 应用

(1) 8259A 初始化编程。根据系统要求，8259A 初始化编程如下：

```
MOV  AL,13H        ;写入 ICW1 为边沿触发,单片 8259A 需要 ICW4
OUT  20H,AL
MOV  AL,08H        ;写入 ICW2 中断类型号,起始中断号为 08H
OUT  21H,AL
MOV  AL,01H        ;写入 ICW4,设定  普通全嵌套方式,一般 EOI 方式
OUT  21H,AL
```

(2) 8259A 操作方式编程。在用户程序中，允许用 OCW_1 来设置中断屏蔽寄存器 IMR，以控制各个外设申请中断允许或屏蔽。但注意不要破坏原设定工作方式。如允许日时钟中断 IRQ_0 和键盘中断 IRQ_1，其他状态不变，则可送入以下指令：

```
IN    AL,21H       ;读出 IMR
AND   AL,0FCH      ;只允许 IRQ0 和 IRQ1,其他不变
OUT   21H,AL       ;写入 OCW1,即 IMR
```

由于中断采用的是非自动结束方式因此若中断服务程序结束，则在返回断点前，必须对 OCW_2 写入 00100000B，即 20H，发出中断结束命令：

```
MOV  AL,20H        ;设置 OCW2 的值为 20H
OUT  20H,ALP       ;写入 OCW2 的端口地址 20H
IRET               ;中断返回
```

在程序中，通过设置 OCW_3，亦可读出 IRR、ISR 的状态以及查询当前的中断源。如要读出 IRR 内容以查看申请中断的信号线，这时可先写入 OCW_3，再读出 IRR：

```
MOV  AL,20H          ;写入 OCW3,读 IRR 命令
OUT  21H,AL
NOP                  ;延时,等待 8259A 的操作结束
IN   AL,20H          ;读出 IRR
```

当 $A_0=1$ 时，IMR 的内容可以随时方便地读出，如在 BIOS 中，中断屏蔽寄存器 IMR 的检查程序如下：

```
    MOV  AL,0            ;设置 OCW1 为 0,送入奇端口地址,表示 IMR 为全 0
    OUT  21H,AL
    IN   AL,21H          ;读 IMR 状态
    OR   AL,AL           ;若不为 0,则转出错程序 ERR
    JNZ  ERR
    MOV  AL,0FFH         ;设置 OCW2 为 FFH,送奇地址端口中,表示 IMR 为全 1
    OUT  21H,AL
    IN   AL,21H          ;读 IMR 状态
    ADD  AL,1            ;IMR = 0FFH?
    JNZ  ERR             ;若不是 0FFH,则转出错程序 ERR
    ......
ERR:
```

4. 中断服务程序设计的一般过程

在进行中断服务程序设计时，需要首先设置中断向量表，把需要执行的中断服务程序的入口地址事先放入中断向量表的相应存储单元，然后才能允许中断。几种中断服务程序的编程原则大致相同。但是，由于可屏蔽中断要涉及到中断控制器的操作，所以其编程较内部中断和非屏蔽中断复杂一些。

下面以可屏蔽中断为例，介绍中断服务程序设计的一般过程。

（1）主程序部分。为了实现中断过程，在中断产生之前，应在主程序中完成下列程序设计内容：

1）初始化中断向量表。在初始化中断向量表时，可以利用数据传送指令直接对中断向量表的相应存储单元写入中断向量，也可以利用 DOS 系统功能调用 INT 21H 的 25H 和 35H 号功能修改中断向量。

2）设置中断控制器屏蔽位。通过 8259A 控制的硬件中断，必须使 8259A 的中断屏蔽寄存器 IMR 的相应位置 0，才能允许中断请求。而中断服务程序结束返回 DOS 后应恢复系统原状态，因此应在修改 IMR 之前保存原内容，并于程序退出前予以恢复。在程序中还可以通过控制屏蔽位，随时允许或禁止有关中断的产生。

3）设置 CPU 的中断允许标志 IF。硬件中断来自计算机的外部设备，它随时都可能提出申请，除利用 IMR 控制某一个或几个中断响应外，还可以通过关中断指令 CLI 和开中断指令 STI 来控制所有可屏蔽中断的产生。当不需要中断或不能中断时，就必须关中断，防止不可预测的后果。而在其他时间则要开中断，以便及时响应中断，为外设提供服务。

（2）设计中断服务程序。中断服务程序中通常须完成保护现场、中断处理、恢复现场、向 8259A 发送中断结束命令、中断返回等任务。中断服务子程序设计如下：

```
INTPROC   PROC  FAR        ;定义中断服务子程序,远过程
          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
          MOV   AL,20H     ;向 8259A 发送 EOI 命令
          OUT   20H,AL
          IRET             ;中断返回
INTPROC ENDP
```

6.3.5 8259A 的应用

[例 6.2] 设计一个中断处理程序。要求中断请求信号以跳变方式由 IR_2 提出中断请求(可为任一定时脉冲信号),当 CPU 响应 IR_2 请求时,输出字符串 "8259A INTERRUPT!",中断 10 次,程序退出(设 8259A 的端口地址为 20H 和 21H,中断类型号为 40H)。

解 中断处理程序如下:

```
DATA            SEGMENT
    MESS        DB  '8259A? INTERRUPT!',0AH,0DH,'$'
    COUNT       DB  10                    ;计数值为 10
    DATA        ENDS
;
    STACK       SEGMENT  STACK
    STA         DB  100H  DUP(?)
    TOP         EQU  LENGTH    STA
    STACK       ENDS
;
    CODE        SEGMENT
    ASSUME      CS:CODE,DS:DATA,SS:STACK
    MAIN:       CLI
                MOV      AX,DATA
                NOV      DS,AX
                MOV      AX,STACK
                MOV      SS,AX
                MOV      SP,TOP
                PUSH  DS
                MOV      AX,0000H
                MOV      DS,AX
                MOV      SI,0100H         ;中断向量地址 40H×4 = 0100H
                MOV      AX,OFFSET INTF   ;取中断入口偏移地址
                MOV      [SI],AX          ;写 INTF 的偏移地址
```

```
        MOV     AX,SEG  INTF    ;取中断入口段地址
        MOV     [SI+2],AX       ;写 INTF 的段地址
        MOV     AL,13H          ;8259A 初始化
        OUT     20H,AL          ;ICW1,单片,边沿触发
        MOV     AL,40H          ;ICW2 中断类型号 40H
        OUT     21H,AL
        MOV     AL,01H          ;ICW4 非自动结束
        OUT     21H,AL
        IN      AL,21H          ;读 IMR
        AND     AL,0FBH         ;允许 IR2 请求中断
        OUT     21H,AL          ;写中断屏蔽字 OCW1
WAIT1:  STI                     ;开中断
        CMP     COUNT,0         ;判断 10 次中断是否结束
        JNZ     WAIT1           ;未结束,等待
        IN      AL,21H          ;读 IMR
        OR      AL,04H          ;屏蔽 IR2 请求
        OUT     21H,AL
        CLI                     ;关中断
        MOV     AH,4CH          ;结束,返回 DOS
        INT     21H
                                ;中断服务子程序
INTF    PROC
        PUSH    DS              ;保护现场
        PUSH    AX
        PUSH    DX
        STI                     ;开中断
        MOV     DS,AX
        MOV     DX,OFFSET MESS
        MOV     AH,09H
        INT     21H
        DEC     COUNT           ;控制 10 次循环
        MOV     AL,20H          ;写 OCW2,送中断结束命令 EOI
        OUT     20H,AL
        POP     DX              ;恢复现场
        POP     AX
        POP     DS
        IRET                    ;中断返回
INTF    ENDP
CODE    ENDS
        END     MAIN
```

[例 6.3] 8259A 的初始化编程。在 Intel 8088 CPU 系统中，使用 2 片 8259A 管理中断，采用级联方式。主片中的 8 个中断请求 $IR_7 \sim IR_0$ 除 IR_2 扩展从片以外，其他均为系统使用，从片中的 8 个中断请求 $IR_7 \sim IR_0$ 供用户使用。其硬件连接如图 6.20 所示。设主片工

作于特殊完全嵌套、非缓冲和非自动结束方式，中断类型号为 40H，端口地址为 20H 和 21H。从片工作于完全嵌套、非缓冲和非自动结束方式，中断类型号为 70H，端口地址为 60H 和 61H。试编写主片和从片的初始化程序。

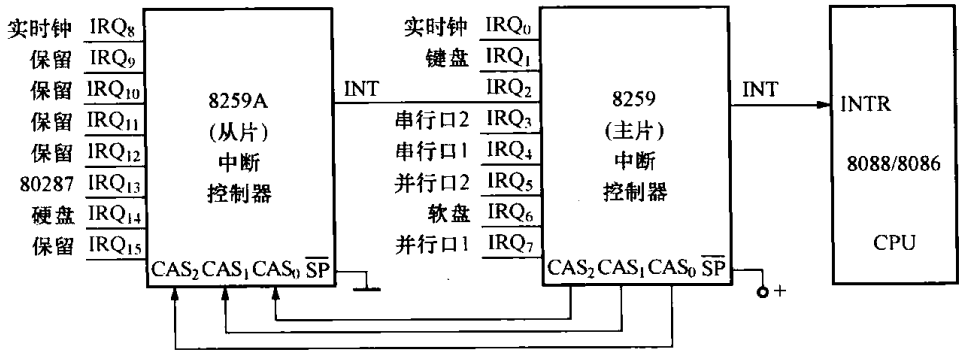


图 6.20 8259A 与 CPU 的硬件连接

解 (1) 系统的工作过程分析：当 8088 CPU 响应 8259A 的中断请求 INT 时，向 8259A 回送中断响应信号$\overline{\text{INTA}}$，自动进入中断响应总线周期，进行中断处理。

在第一个$\overline{\text{INTA}}$周期，CPU 向 8259A 发送第一个$\overline{\text{INTA}}$脉冲，表示响应 INT 请求，并利用这个信号将 8259A 的请求信号 IR_i 复位，同时使总线锁存信号$\overline{\text{LOCK}}$有效，以禁止其他总线主控设备争用总线。

在第二个$\overline{\text{INTA}}$周期，8259A 将中断类型号 n 通过数据总线送至 CPU，由 CPU 自动完成向量表地址 $4n$ 运算后，从中断向量表中取出相应的中断向量送入 CS：IP 中，继而转去执行中断服务子程序。同时，还利用第二个$\overline{\text{INTA}}$周期的后沿将 INT 复位。在自动结束方式中，也是利用这个后沿，将 ISR 中的中断服务标志位清除。

(2) 根据题意，写出 ICW_1，ICW_2，ICW_3 和 ICW_4 命令字，按顺序编程写入。编写初始化程序如下：

主片 8259A 的初始化程序如下：

```
MOV     AL,00010001B        ;级联,边沿触发,需要写 ICW4
OUT     20H,AL              ;写 ICW1
MOV     AL,01000000B        ;中断类型号 40H
OUT     21H,AL              ;写 ICW2
MOV     AL,00000100B        ;主片的 IR2 引脚接从片
OUT     21H,AL              ;写 ICW3
MOV     AL,00010001B        ;特殊完全嵌套、非缓冲、自动结束
OUT     21H,AL              ;写 ICW4
```

从片 8259A 初始化程序如下：

```
MOV     AL,00010001B        ;级联,边沿触发,需要写 ICW4
OUT     60H,AL              ;写 ICW1
MOV     AL,01110000B        ;中断类型号 70H
OUT     61H,AL              ;写 ICW2
```

```
MOV     AL,00000010B            ;接主片的 IR2 引脚
OUT     61H,AL                  ;写 ICW3
MOV     AL,00000001B            ;完全嵌套、非缓冲、非自动结束
OUT     61H,AL                  ;写 ICW4
```

（3）级联工作编程。

对 OCW_1 的编程：假设使主片屏蔽 $IR_4 \sim IR_7$、使从片屏蔽 IR_{15} 和 IR_8，则应该向主片 21H 端口写入 11110000B＝0F0H，向从片 081H 端口写入 10000001B＝81H，程序设计如下：

```
MOV  AL,0F0H
OUT  21H,AL
MOV  AL,81H
OUT  61H,AL
```

对 OCW_2 的编程：当从片 8259A 中的某中断请求进入服务时，主片 8259A 的优先权控制逻辑不封锁这个从片，从而使来自从片的更高优先级的中断请求能被主片所识别，并向 CPU 发出中断请求信号。因此，当中断服务程序结束时必须用软件来检查被服务的中断是否是该从片中唯一的中断请求。先向从片发出一个 EOI 命令，清除已完成服务的 ISR 位，然后再读出 ISR 的内容检查它是否为 0。如果 ISR 的内容为 0，则向主片发一个 EOI 命令，清除与从片相对应的 ISR 位；否则，就不向主片发 EOI 命令，继续执行从片的中断处理，直到 ISR 的内容为 0，再向主片发出 EOI 命令。按 OCW_2 格式，EOI 命令代码是 00100000B＝20H。

；读 ISR 的内容的程序如下：

```
MOV  AL,0BH
OUT  60H,AL
NOP
IN   AL,60H
......
```

；从片发 EOI 命令的程序如下：

```
MOV  AL,20H
OUT  60H,AL
......
```

；主片发 EOI 命令的程序如下：

```
MOV  AL,  20H
OUT  20H,AL
```

对 OCW_3 的编程：OCW_3 也是通过端口 20H 和 60H 写入的，不同之处在于 OCW_2 中 D_4D_3＝00，而在 OCW_3 中 D_4D_3＝01。8259A 通过 D_4D_3 判断是 OCW_2 还是 OCW_3，然后写入各自的寄存器。

6.4 键 盘 中 断

6.4.1 键盘的工作原理

键盘是计算机最基本的一种输入设备，通过键盘可以将英文字母、数字、标点符号等键的按键信息输入到计算机中。键盘上的每个按键起一个开关作用，故又称为键开关。键盘有很多种类型，其工作原理也不尽相同，按照接触方式可以分为触点式和无触点式两种。

一般每个键在按下和松开时，都会经历短时间的抖动后才到达稳定接通或断开。抖动持续时间因键的质量有所不同，通常为5～20ms。在识别按键和键释放时必须避开这一段不稳定的抖动状态，才能正确检测识别。去抖动的方法一般有两种：一种是软件延时，即发现有键按下或释放时，软件延时一段时间再检测；另一种是硬件消抖法，如用基本RS触发器、单稳电路、RC滤波器等。

键盘结构的关键是如何把键盘上的每一次按键动作转换成相应的ASCII码送到处理机。按照键的编码方式的不同，可以将键盘分为编码键盘和非编码键盘两种。

编码键盘是用硬件电路来识别按键代码的键盘，当按下某一键后，相应电路即给出一组编码信息（如ASCII码），送到主机去进行识别和处理。编码键盘的响应速度快，但它以复杂的硬件结构为代价，并且其硬件的复杂程度随着键数的增加而增加。

非编码键盘是用较为简单的硬件和专门的键盘处理程序来识别按键的位置、编码、传送等功能。即当按下某键以后，并不给出相应的ASCII码，而提供与按键对应的中间代码，然后再把中间代码转换成对应的ASCII码。非编码键盘的响应速度不如编码键盘快，但它通过软件编程可为键盘中某些键的重新定义提供更大的灵活性，因此得到广泛的应用。

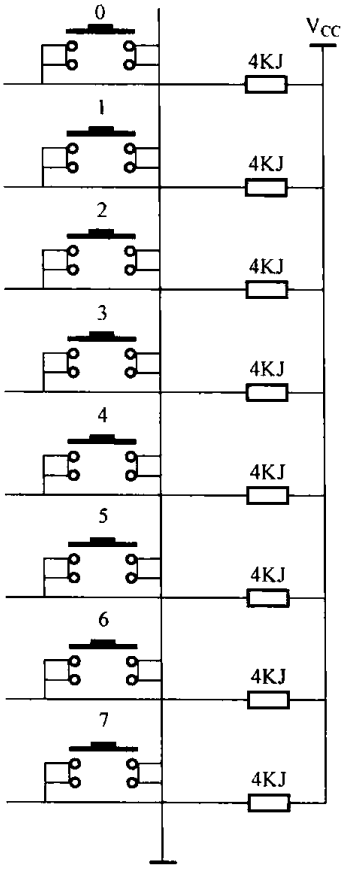


图6.21 线性键盘结构

1. 键盘结构

键盘是由一组规则排列的按键组成，键盘的结构两种形式：一种是线性键盘，另一种是矩阵键盘。线性键盘就是将各按键顺序排列，各键的一端接电源，另一端与微机的输入端口相连，微机通过查询程序读取按键各位的状态，然后判断是否有键按下以及按键信息。线性键盘结构如图6.21所示。由于线性键盘的按键直接与微机输入端口相连，一个按键对应端口中的一位，因此线性键盘只适用于按键较少的场合。例如，用一个8位的数据输入端口来控制线性键盘，按键的个数只能是8个。

另一种结构的键盘是矩阵键盘，是将按键排列成m行n列的矩阵形式，行、列线通过按键相连，按键总数最多为$m\times n$个，键盘控制线共$m+n$条。矩阵键盘如图6.22所示。当用一个8位的数据输入端口来控制矩阵键盘时，按键的个数可以是16个。矩阵键盘所需要的连接线少，是一般微机中常用的键盘结构。

对于矩阵式键盘，按键的位置由行号和列号唯一确定，因此利用按键所在的行号和列号构成该按键的行列值编码。图

6.22 中，6 号键位于第 1 行、第 2 列，因此，该按键的行列值编码为 11011011（$X_3X_2X_1X_0Y_3Y_2Y_1Y_0$）。可以将所有按键的行列值编码构成编码表，供查表使用。

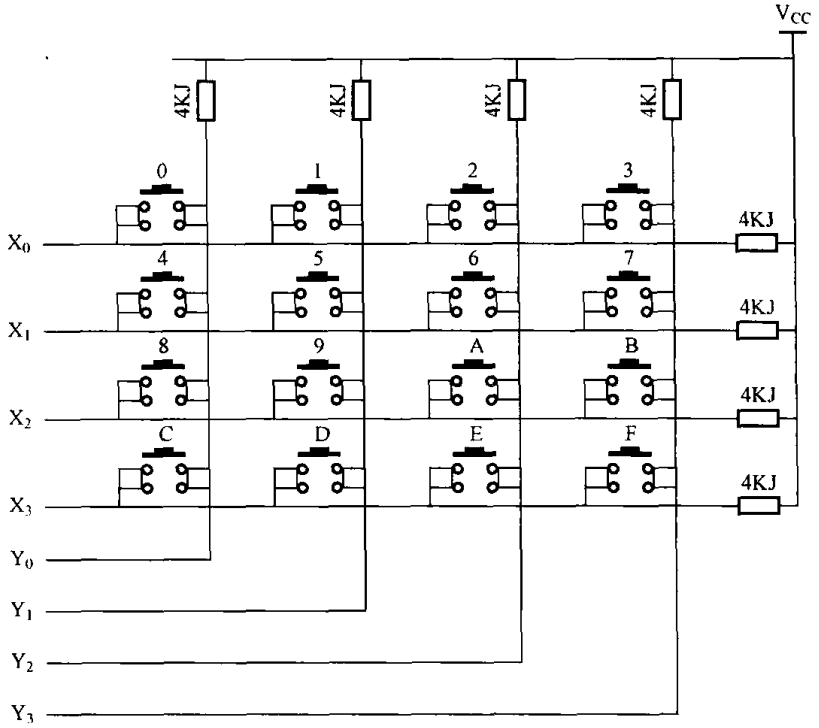


图 6.22 矩阵键盘

2. 矩阵键盘按键识别方法

矩阵键盘按键的识别方法主要有行扫描法和反转法两种。

（1）行扫描法。行扫描法又称为逐行扫描查询法，是一种最常用的按键识别方法。在图 6.22 所示的矩阵键盘中，将按键的行线作为输入、列线作为输出。按键的识别过程如下：

1）判断键盘中有无键按下。首先将全部行线 X_0～X_3 置低电平，然后读取列线（Y_0～Y_3）数据。若读取的列线数据每一位均为高电平，则表示键盘中无键按下。若读入的列线数据中有一位为低电平，则表示当前行中有键被按下。

2）当检测到有键按下后，延时一段时间应再判断一次键盘中有无键按下，以剔除键抖动导致的按键闭合，这个过程称为去除键抖动。

3）判断闭合键所在的位置。当确认有键按下后，再进一步确定闭合键的位置。其方法是：从第 0 行开始，顺序逐行扫描，即逐行输入低电平。每扫描一行，读入列线数据，若数据中有一位为低电平，则表示该位对应的列与当前扫描行的交点处的按键被按下，并由此得到闭合键的行值和列值。

例如，图 6.22 中，键盘上某个键被按下（假设键“6”被按下）。行扫描判断方法如下：首先将全部行线 X_0～X_3 置 0，读取列线数据，该数据中有一位为 0，表明有键按下（按照假设键“6”被按下，Y_2=0）。与列线 Y_2 相连的键有 4 个，它们是“2”、“6”、“A”、“E”。为了进一步判断哪一个键被按下，使用行扫描原理，先使第 0 行输入 0，其他行为 1，即 $X_3X_2X_1X_0$=1110，再读取列线数据，并判断列线数据中是否有 0 存在；如果没有，表示按下键没在该行上；继续使 $X_3X_2X_1X_0$=1101，再读取列线数据，如此继续判断是否有 0 存

在；按照假设读入的列值应为1011，表示按下的键在该行上。扫描结果为被按下键的行值为1101，列值为1011，该键的行列值编码为11011011（$X_3X_2X_1X_0Y_3Y_2Y_1Y_0$）。

（2）反转法。反转法不需要逐行输出，以图6.22键盘矩阵为例，方法如下：

1）首先，将行线作为输入、列线作为输出。并使 $X_3\ X_2\ X_1X_0=0000$，从 $Y_3\ Y_2\ Y_1\ Y_0$ 读入列值数据，若读入的列值数据中有一位为0，则表明有键被按下，存储此值作为“列值”，转第2）步，否则在本步骤中循环，继续查询。

2）将列线作输入、行线作为输出。将上述得到的列值再从 $Y_3\ Y_2\ Y_1\ Y_0$ 输入，接着读取行值数据，该数据中必有一位为0，为0的位所对应的行线就是被按下键所在的行，存储此值作为“行值”。将行值和列值组合在一起构成行列值编码。

例如，图6.22中，假设键“6”被按下，反转法判断方法如下：首先使行线为输入、列线为输出。置 $X_3\ X_2\ X_1\ X_0=0000$，读取列值数据，该数据中有一位为0，假设键“6”被按下，读取的列值数据为1011，存储该“列值”。然后使列线为输入，行线为输出。将存储的列值从 $Y_3\ Y_2\ Y_1\ Y_0$ 输入，从 $X_3X_2X_1X_0$ 上读行值数据，该数据中必有一位为0，存储此值为“行值”，该键的行列值为11011011。

当得到按键的行列值后，用查表法即可得到按键的键号。查表法中，将所有按键的键号构成键号表，将每个键号的行列值按照键号的排列顺序构成行列值表。根据按键的行列值在行列值表中查表，找到对应的行列值所在位置，然后到键号表中查表即可。

3. *键盘的工作方式*

CPU对键盘的响应取决于键盘的工作方式，键盘的工作方式应根据实际应用系统中CPU的工作状况而定，其选取的原则是既要保证CPU能及时响应按键操作，又不要过多占用CPU的工作时间。通常，键盘的工作方式有编程扫描、定时扫描和中断扫描三种。

（1）编程扫描方式。编程扫描方式是利用CPU完成其他工作的空余时间调用键盘扫描子程序来响应键盘输入的要求。在执行键功能程序时，CPU不再响应其他键输入要求，直到CPU重新扫描键盘为止。键盘扫描程序一般应包括以下内容：

1）判别有无键按下。

2）键盘扫描取得闭合键的行、列值。

3）用计算法或查表法得到键值。

4）判断闭合键是否释放，如没释放则继续等待。

5）将闭合键键号保存，同时转去执行该闭合键的功能。

（2）定时扫描方式。定时扫描方式就是每隔一段时间对键盘扫描一次，当定时时间到，CPU响应中断后对键盘进行扫描，并在有键按下时识别出该键，再执行该键的功能程序。定时扫描方式的硬件电路与编程扫描方式基本相同。

（3）中断扫描方式。采用上述两种键盘扫描方式时，无论是否按键，CPU都要定时扫描键盘。因此，CPU可能经常处于空扫描状态，为提高CPU工作效率，可采用中断扫描工作方式。其工作过程如下：当无键按下时，CPU处理自己的工作，当有键按下时，产生中断请求，CPU转去执行键盘扫描程序，并识别键号。

6.4.2 PC机键盘

PC机键盘一般由键盘矩阵和以单片机或专用控制器为核心的键盘控制电路组成，被称为智能键盘。单片机通过执行固化在ROM中的键盘管理和扫描程序，对键盘矩阵进行扫

描，发现、识别按键的位置，形成与按键位置对应的扫描码，并以串行的方式送给微机主板上的键盘接口电路，供系统使用。

PC 机键盘采用非编码键盘原理，包括两部分：一部分组装在键盘盒内部，称为键盘电路；另一部分安装在主机板上，称为键盘接口电路。

识别键盘上的键是否按下，通过扫描码就能唯一地确定哪个键改变了状态。扫描码从 01（Esc）到 83（Del），或从 01H 到 53H。当在键盘上“按下”或“放开”一个键时，如果键盘中断是允许的（21H 端口第 1 位＝0）就会产生一个类型 9 的中断，并转入到 BIOS 的键盘中断处理程序。该处理程序从 8255 可编程外围接口芯片的输入端口 60H 读取一个字节，这个字节的低 7 位是键的扫描码。最高位为 0 或 1，分别表示键是“按下”状态还是“放开”状态。按下时，取得字节称为接通扫描码，放开时取得字节称为断开扫描码。如按下 Esc 键时产生一个为 01H（00000001B）的接通扫描码，放开 Esc 键时产生一个断开扫描码为 81H（10000001B）。

键盘电路是以单片机为核心的键盘扫描电路。系统加电后，固化在单片机中的键盘扫描程序周期地扫视每一个按键，一旦有按键按下，键盘扫描程序立即识别闭合键的行列位置，然后通过信号线串行地发出闭合键的接通扫描码。闭合键断开后，键盘电路又发出该键的断开扫描码。

位于主机内的键盘接口电路用单片机作为接口控制器，在单片机软件的支持下，完成下列任务：接收来自键盘电路的按键扫描码；对串行数据进行奇偶校验；完成串/并转换；将表示行列位置的扫描码转换成系统扫描码。转换完毕，由键盘接口电路向主 8259A IR_1 端子提出中断请求。

CPU 响应键盘中断后，转入键盘中断处理程序，主要完成以下工作：开中断，保护现场，从键盘接口电路（口地址 60H）读取按键扫描码，对扫描码进行分析（译码）、处理，最终生成相应的键代码存入键盘缓冲区，最后恢复现场向 8259A 发出中断结束命令，中断返回。

综上所述，整个键盘处理程序由查询程序、传送程序、译码程序三部分组成。工作过程总结如下：

（1）主程序首先调用查询程序，通过查询接口逐行扫描键位矩阵，同时检测行列的输出，由行与列的交连信号确定某闭合键的坐标，即得到被按键对应的扫描码。

（2）主程序调用传送程序将得到的扫描码传送给位于主机内的键盘接口电路。

（3）主程序调用译码程序将键盘接口内的扫描码翻译为相应键的编码信息。

（4）在需要的时候，键盘接口电路把上述编码信息传送给主机。

习　　题

6.1　什么是中断？计算机采用中断有什么好处？简述中断的响应和处理过程。

6.2　CPU 响应外设可屏蔽中断请求的条件是什么？

6.3　什么是硬件中断和软件中断？在 PC 机中两者的处理过程有什么不同？

6.4　在 8086 系统中，下面的中断请求优先级最高的请求是哪一个？

（1）NMI　　（2）INTR　　（3）内部硬件中断　　（4）单步中断

6.5 中断服务程序结束时，用指令 RET 代替 IRET 指令能否返回主程序？这样做存在什么问题？

6.6 简述中断向量、中断向量表、中断类型码之间的关系。在基于 8086/8088 的微机系统中，中断类型码和中断向量之间有什么关系？若某中断类型码是 12H，则其中断向量存于何处？若软中断指令 INT 30H，其中的中断类型号为多少，该中断的服务程序的入口地址在内存单元什么位置？

6.7 简述 8259A 的内部结构和主要功能。8259A 的中断屏蔽寄存器 IMR 与 8086 中断允许标志 IF 有什么区别？

6.8 中断控制系统由 3 片 8259A 级连而成，两片从 8259A 分别接入主 8259A 的 IR_3 和 IR_4 端。

(1) 考虑 8 位数据的 8259A 如何与 16 位数据的 8086CPU 系统相连接。

(2) 简述主、从 8259A 硬件接线的电位特点，CAS 信号端的作用。

(3) 画出系统的硬件连接图。

(4) 确定该系统最多可接受的中断源数量。

(5) 若已知中断类型码和中断矢量，采用电平触发方式，完全嵌套，普通 EOI 结束，试编写全部初始化程序。

6.9 若 8086 系统采用单片 8259A 中断控制器控制中断，中断类型码给定为 20H，中断源的请求线与 8259A 的 IR_4 相连，试问：对应该中断源的中断向量表入口地址是什么？若中断服务程序入口地址为 4FE24H，则对应该中断源的中断向量表内容是什么？如何定位？

6.10 试按照如下要求对 8259A 设定初始化命令字：8086 系统中只有一片 8259A，中断请求信号使用电平触发方式，全嵌套中断优先级，数据总线无缓冲，采用中断自动结束方式。中断类型码为 20H～27H，8259A 的端口地址为 B0H 和 B1H。

第 7 章　可编程计数器/定时器 Intel8253

7.1　概　　述

在微机控制及应用系统中常需要一些实时时钟以提供定时、延时控制、检测、中断、扫描等，或对外部事件进行计数。实现定时/延时控制主要有软件定时、不可编程的硬件定时、可编程的硬件定时器三种方法。

软件定时是让计算机执行一个循环程序段，程序段本身没有具体的执行任务，通过改变循环次数就可以实现不同定时时间的软件定时。定时较准确，但占用了 CPU 时间；对不同 CPU 的主机，同一个定时软件，定时时间不同。

不可编程的硬件定时一般选用集成电路定时芯片，如定时器 555、单稳态触发器等，不占用 CPU 时间，定时电路简单。但一旦硬件电路参数确定后，定时时间不易改变。

可编程硬件定时器是通过执行初始化程序来设定定时时间和计数控制，所以执行不同的初始化程序就很容易地改变定时或计数。可编程计数器/定时器 8253/8254 就是用软、硬件相结合的方法实现定时和计数控制。

7.1.1　计数器/定时器 8253 的内部结构

8253 是 Intel 公司生产的通用可编程计数/定时器（Counter/Timer Circuit，CTC），采用 NMOS 工艺由单一＋5V 电源供电，是 24 个引脚的双列直式芯片。其主要功能为：

（1）一个芯片有三个独立的 16 位减法计数器。

（2）每个计数器都可按二进制或十进制减法计数。

（3）每个计数器的最高计数频率可达 2MHz。

（4）每个计数器有 6 种工作方式，可由程序设定。

（5）所有输入输出均与 TTL 电平兼容。

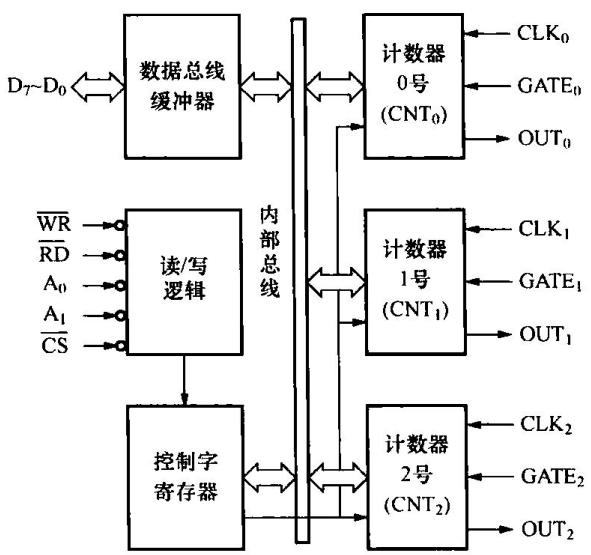


图 7.1　8253 内部结构框图

图 7.1 所示为 8253 的内部结构框图。它由计数器、控制字寄存器、读/写控制逻辑和总线缓冲器四部分组成。

（1）三个独立的 16 位计数器。8253 有计数器 0、计数器 1 和计数器 2 三个独立的计数通道，每个通道的内部结构完全相同。每个计数器有时钟信号 CLK 和门控信号 GATE 两个输入信号。若 CLK 的频率是由精确的时钟脉冲提供，则计数器就能作为定时器使用；若 CLK 是由外部引入的输入脉冲，则就作为计数器使用。门控信号 GATE 是用于控制计数器启/停工作的外部信号。每个计数器还有输出信号 OUT，可以用编程的方法来控制在计数/定时

完成时，在此引脚输出所规定的波形信号。

计数器的计数初值在开始计数之前，由 CPU 执行输出指令来预置，在计数过程中，CPU 随时可用输入指令读取任一计数器的当前计数值，并不影响当前计数。

（2）控制字寄存器。用来保存由 CPU 送来的控制字。每个计数器都有一个控制字寄存器，用于保存本计数器的控制信息。如计数器的工作方式、计数制形式、输出波形及 CPU 如何装入计数器初值等。控制字寄存器只能写入而不能读出。

（3）读/写逻辑。读/写逻辑接收由 CPU 送入的读（$\overline{RD}$）、写（$\overline{WR}$）信号、片选和地址信号（$\overline{CS}$、A_0、A_1）选择相应寄存器，并确定数据传送方向是读出还是写入。数据由 8253 传向 CPU 为读出；数据由 CPU 传向 8253 为写入。

（4）数据总线缓冲器。这是一个双向、三态 8 位缓冲器。用于将 8253 与系统数据总线（如 D_0～D_7）相连。CPU 用输入输出指令对 8253 进行的读写操作都是通过数据总线缓冲器进行信息传送，包括 CPU 向 8253 写入工作方式控制字；CPU 向计数寄存器写入计数初值；从 8253 读出计数器初值或当前值送 CPU。

7.1.2　计数器/定时器 8253 引脚及功能

图 7.2 所示为 8253 的外部引脚图，其主要引脚的功能如下：

（1）D_0～D_7：8 位双向数据线。用来传送数据、控制字和计数器的计数初值。

（2）$\overline{CS}$：片选信号，输入，低电平有效。有系统高位 I/O 地址译码产生。当它有效时，此 8253 芯片被选中。

（3）$\overline{RD}$：读控制信号，输入，低电平有效。当它有效时，表示 CPU 要对此 8253 芯片进行读操作。

（4）$\overline{WR}$：写控制信号，输入，低电平有效。当它有效时，表示 CPU 要对此 8253 芯片进行写操作。

（5）A_0，A_1：地址信号线。高位地址信号经译码产生 $\overline{CS}$片选信号，决定了 8253 芯片所具有的地址范围。而 A_0 和 A_1 地址信号则经片内译码产生 4 个有效地址，分别对应了芯片内部 3 个独立的计数器和一个控制寄存器。具体规定如下：

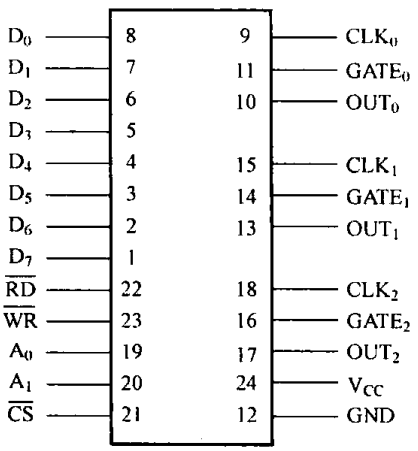


图 7.2　8253 外部引脚图

A_1	A_0	
0	0	选择计数器 0
0	1	选择计数器 1
1	0	选择计数器 2
1	1	选择控制寄存器

（6）CLK_0～CLK_2：各计数器的时钟信号输入端。计数器对此时钟信号进行计数。

（7）$GATE_0$～$GATE_2$：门控信号，用于控制计数的启动和停止。多数情况下，GATE=1时允许计数，GATE=0 时停止计数。但有时仅用 GATE 的上升沿启动计数，启动后则 GATE 的状态不再影响计数过程。

（8）OUT_0～OUT_2：计数器输出信号。在不同的工作方式下将产生不同的输出波形。

7.2 8253 的控制字/锁存字

1. 8253 的控制字格式

8253 的控制字格式如图 7.3 所示。各位意义如下：

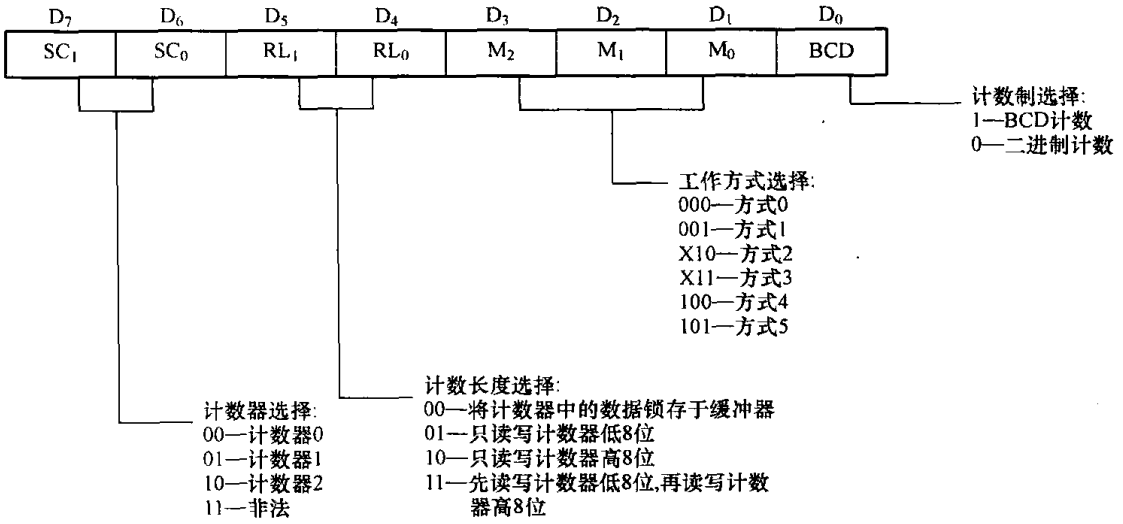


图 7.3 8253 控制字格式

D_7、D_6：计数器选择位，规定本控制字是哪个计数器。指明控制字是写入哪个计数器的控制寄存器。

D_5、D_4：计数长度选择位。

00—为锁存命令。当计数器中控制字寄存器接收到此锁存命令信号时，会立即将 16 位减 1 计数器的内容锁存到输出锁存寄存器中，不再变化。直到 CPU 读取锁存器内容后，或重新写入控制字，才会自动解除锁存状态。

01 或 10—只读/写计数器低 8 位或高 8 位。因计数寄存器是 16 位的，没有写入的高 8 位或低 8 位缺省自动置为 00H。

D_3、D_2、D_1：计数器工作方式选择位。8253 每个计数器有 6 种工作方式。

D_0：用户所使用的计数值是二进制数还是 BCD（十进制）数。因每个计数器的字长是 16 位，所以采用二进制计数则计数范围为 0000H～FFFFH，如果采用 BCD 计数则计数范围为 0000～9999。由于是减 1 计数器，当计数初值为 0000 时，对应的是最大计数值（二进制计数时为 65536，十进制计数时为 10000）。

2. 8253 的锁存字

为了了解 8253 某个计数器的计数状态，通常需读出某计数器的计数值至 CPU 中。有两种读计数方法：

（1）直接读。8253 计数工作时，输出锁存器的内容是跟随减 1 计数器内容而变化的，故两者的值一样。当采用这种读操作时，先用门控信号 GATE 暂停计数或采用外部逻辑电路暂停时钟 CLK 输入，以暂停计数过程。计数器停止计数后，按控制字中 D_4D_3 位规定的读/写顺序，直接用一条或两条输入（IN）指令读出输出锁存器中的当前计数值。

（2）锁存后读。在计数过程中既读出计数值又不影响减 1 计数器的计数操作。首先 CPU 向要读的 8253 计数器发出一个锁存命令字，其格式如图 7.4 所示。它是控制字的特殊形式，最高两位 D_7D_6 指定要锁存的计数器 0、1、2；$D_5D_4=00$ 为锁存命令的标志，而低 4 位 $D_3\sim D_0$ 可以是任意值。当 8253 计数器接收到此锁存命令后，输出锁存器中的计数值就被锁存，不再随减 1 计数器的变化而变化。故读数时先送锁存命令，然后再用输入（IN）指令读取锁存器的低 8 位和高 8 位计数值。

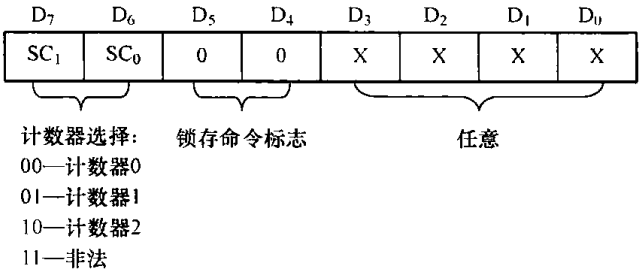


图 7.4　8253 锁存命令字

锁存命令不影响原已选定的工作方式，读操作也不影响计数过程，常用于经常需要读出计数过程的计数值，根据计数值再作判断，决定程序的走向。

例如，读出并检查计数器 2 的计数值是否为 1000H，若不是则等待再读；否则，程序继续执行。（设各计数器和控制寄存器地址为：COUNT＋0～COUNT＋3）程序段如下：

```
COUNT  EQV   040H              ;设计数器 0 的符号地址为 040H
  ⋮
LPCN:  MOV   AL,10000100B      ;对计数器 2 送锁存命令,仅使 RL1,RL0 = 00
       OUT   COUNT + 3,AL
       IN    AL,COUNT + 2      ;读计数器 2 当前计数值
       MOV   AH,AL             ;低 8 位暂存 AH 中
       IN    AL,COUNT + 2      ;读高 8 位
       XCHG  AH,AL             ;16 位计数值存 AX 中
       CMP   AX,1000H          ;计数值与 1000H 相比较
       JNE   LPCN              ;若不相等则继续等待
```

3. *计数器的启动方法*

8253 计数器的计数过程启动有程序启动和硬件启动。

（1）程序启动。首先在初始化程序时，CPU 向 8253 送入控制字，当 CPU 再向 8253 送入计数初值后就自动启动计数；CPU 写入初值后的第 1 个 CLK 信号将初值寄存器中的内容装入减 1 计数器中，从第二个 CLK 脉冲的下降沿才使减 1 计数器开始减 1 计数；以后，每来一个 CLK 脉冲，使减 1 计数器减 1 直到零为止，计数过程结束。

从 CPU 执行输出指令写入计数初值到计数结束，实际的 CLK 脉冲个数比编程写入的计数初值 N 要多一个，即 $N+1$ 个。只要是用软件启动计数，这种误差是不可避免的。

（2）硬件启动。硬件启动是写入计数初值后并不自动启动计数，而是靠外加在门控信号 GATE 端的信号由低电平变高电平，再经 CLK 信号的上升沿采样之后在该 CLK 的下降沿才开始计数；由于 GATE 信号与 CLK 信号不一定同步，故在极端情况下，从 GATE 变高到 CLK 采样之间的延时可能会经历一个 CLK 脉冲宽度，因此在计数初值与实际的 CLK 脉冲个数之间也会有一个误差。

8253 有 6 种工作方式，有些工作方式计数器每启动一次只计数一次（从初值减到零），

要想重复计数过程则必须重新启动，因此称它们为不自动重复的计数方式；另一些工作方式，一旦计数启动，只要门控信号 GATE 保持高电平，计数过程就会自动周而复始地重复下去，这时 OUT 端可以产生连续的波形输出，我们称这种计数过程为自动重复的计数方式。

7.3 8253 的工作方式

8253 各计数器共有 6 种工作方式可供选择。用户可根据所需输出波形、启动方式及 GATE 门控信号的应用方法来选择不同工作方式。下面分别进行介绍。

1. 方式 0——计数结束产生中断

方式 0 为软件启动，不自动重复计数的方式。图 7.5 所示为方式 0 时的输出波形。

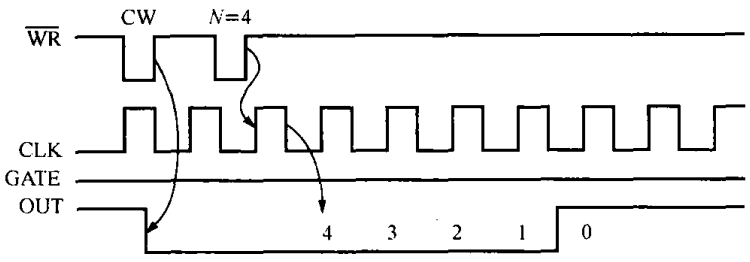


图 7.5 8253 方式 0 的输出波形

若 GATE 始终保持高电平情况下，CPU 先用 OUT 指令对 8253 写入工作方式控制字，执行 OUT 指令时会产生有效$\overline{WR}$脉冲信号，利用$\overline{WR}$的上升沿使得输出端由高电平变为低电平；然后，CPU 又用 OUT 指令对 8253 送入计数初值，经过一个 CLK 信号的上升沿和下降沿将初值装入减 1 计数器。以后每一个 CLK 下降沿进行减 1 计数。当计数结束后，输出端变为高电平。用户利用输出端信号的上升沿作为向 CPU 计数/定时到的中断请求信号。

另外，在实际系统中可以先对 8253 送入方式控制字，并不随后马上送入计数初值，可以在系统需要时再对 8253 送入计数初值，达到用软件方法控制启动计数时刻。

在计数期间 GATE 变低电平，则会暂停计数，直到 GATE 恢复到高电平后，才会继续减 1 计数。故用户可以用 GATE 端作为外加的计数启/停控制端，图 7.6 所示为方式 0 时，有 GATE 信号作用时的波形图。

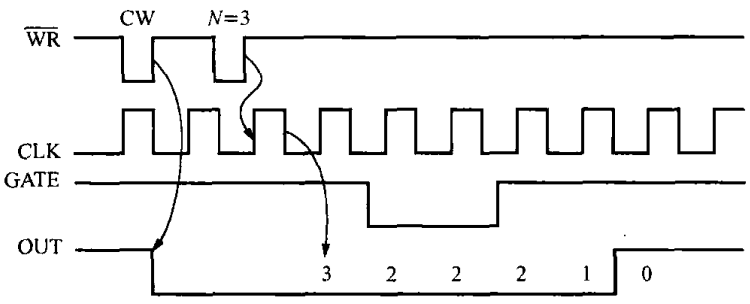


图 7.6 方式 0 时 GATE 信号作用下的波形

在计数期间，如果又重新写入新计数初值，则不管原来计数过程是否结束，计数器也用新计数初值重新计数。如果新计数初值是 16 位的，则在写入其低位字节后，停止原来计数，只有当写入高位字节后，计数器才开始按新计数值重新计数。

不自动重复计数的特点：每写入一次计数初值只计数一个周期。若要重新计数，需 CPU 再次写入计数初值。

2. 方式 1——可重复触发的单稳态触发器

方式 1 是一种硬件启动、不自动重复的工作方式。当写入方式 1 控制字后，OUT 端输

出高电平。在 CPU 写入计数初值后，计数器并不开始计数，而是要等门控信号 GATE 由低到高的跳变后，在下一个 CLK 脉冲的下降沿才将计数初值装入减 1 计数器，此时 OUT 端立刻变为低电平，在下一个 CLK 开始减 1 计数。当计数结束时，OUT 端输出高电平。这样就可以从 OUT 端得到一个负脉冲，其宽度为计数初值 N 乘以 CLK 的周期。

方式 1 的主要特点是：

(1) 计数过程一旦启动，GATE 端即使变低也不会影响计数。

(2) 可重复触发。当计数到 0 后，只要给出一个 GATE 由低到高的跳变，则又可将原初值重新装入减 1 计数器进行计数。计数过程中，GATE 信号变低也不影响计数工作。

(3) 在计数过程中，若写入新计数初值，现行计数过程的输出不受影响。本次计数结束后，GATE 信号再次从低到高，计数器才开始按新的初值进行计数，并在 OUT 端按新值输出负脉冲宽度。

(4) 在计数过程中，外部的 GATE 上升沿提前到来，则在下一个 CLK 脉冲的上升沿重装计数初值，在该 CLK 的下降沿重新开始减 1 计数，直到计数结束，OUT 端才变高电平。这时宽度为重新触发前的已有宽度与新一轮计数过程的宽度之和。

方式 1 的工作波形如图 7.7 所示。利用 OUT 端输出的负脉冲波形，作为请求或控制信号。

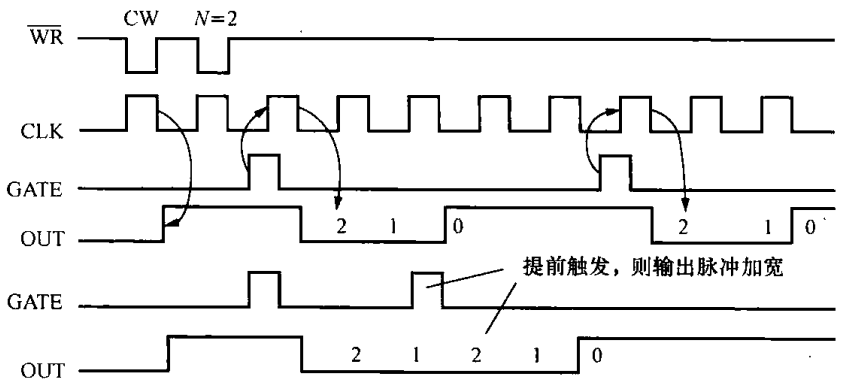


图 7.7　方式 1 的工作波形

3. *方式 2——频率发生器*

工作于方式 2 的计数器既可软件启动，也可硬件启动，并自动重复计数。图 7.8 所示为方式 2 的工作波形。写入方式 2 控制字后，OUT 端变为高电平。然后 CPU 将初值写入计数寄存器，由下一个 CLK 脉冲将计数寄存器的值装入减 1 计数器，开始减 1 计数。当减 1 计数器的值减至 1 时，OUT 端由高电平变低电平，再经过一个 CLK 周期（记为 T_{CLK}），计数器减到零，OUT 又恢复为高电平，OUT 端低电平的宽度为一个 CLK 周期。计数器又自动装入计数初值到减 1 计数器，并开始新一轮计数过程。这样，在 OUT 端就会连续输出宽度为 T_{CLK} 的负脉冲，其周期为 $N \times T_{CLK}$，即 OUT 端输出的脉

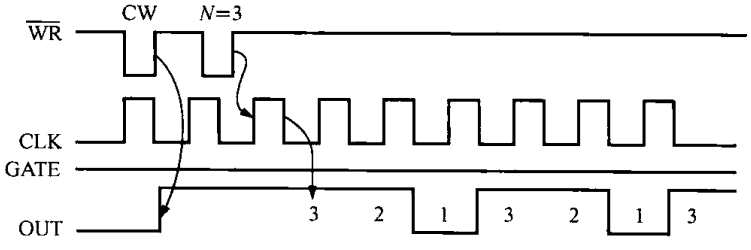


图 7.8　方式 2 的工作波形

冲频率为 CLK 的 $1/N$。所以方式 2 也称为分频器，分频系数就是计数初值 N。由于减 1 计数器为 16 位，通过装入不同初值实现对 CLK 输入时钟脉冲进行 1～65536 的分频。

在方式 2 中，门控信号 GATE 可被用作控制信号。当 GATE 为低电平时，计数暂停，强迫 OUT 端输出高电平。在 GATE 变高后的下一个 CLK 时钟下降沿，计数器又自动装入初值从头开始计数，之后的过程和软件启动相同。此特点可用于实现计数器的硬件同步。

在计数过程中，若 CPU 重新写入新的计数初值，则不影响当前的计数时，而在下一轮计数时，才按新的计数初值进行计数。

4. 方式 3——方波发生器

方式 3 既可软件启动，也可硬件启动，并自动重复计数。在计数过程中，其输出端 OUT 前一半时间为高电平，后一半时间为低电平，计数到 0 时 OUT 端又变为高电平，并开始新一轮计数。此时 OUT 端输出的是方波信号，其输出周期是初值 N 乘上 CLK 脉冲周期。常用作方波频率发生器或波特率发生器。图 7.9 所示为方式 3 的工作波形。

由图 7.9 可见，在写入方式 3 的控制字后，OUT 端立刻变为高电平。若此时 GATE＝1，则装入计数初值 N 后开始计数。如果装入的计数值 N 为偶数，则计数到 $N/2$ 时，OUT 端变低电平。计完其余的 $N/2$ 后，OUT 端又变为高电平。如此这般重复下去，OUT 端输出周期为 $N\times T_{CLK}$ 的对称方波。如果装入的计数值 N 为奇数，输出波形不对称，其中 $(N+1)/2$ 个时钟周期为高电平；$(N-1)/2$ 个时钟周期为低电平。

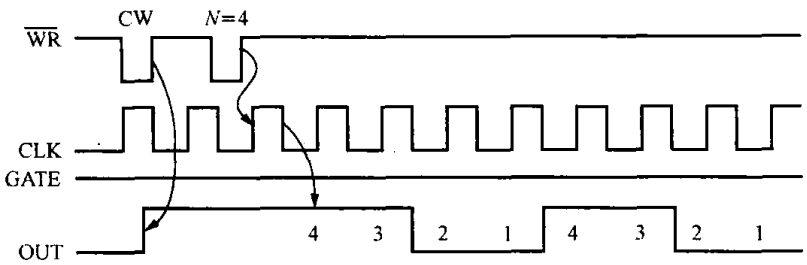


图 7.9 方式 3 的工作波形

GATE＝高电平，允许计数；GATE＝低电平，停止计数。如果在 OUT 低电平期间，使 GATE＝低电平，则 OUT 端马上变高电平，停止计数。当 GATE 变高以后，将会重装初值，重新开始新的计数。

在计数的前半周期内，CPU 若写入新计数初值，并不影响当前的计数过程，只有当前半周期结束后才启用新的计数初值，开始新计数值计数。如果在前半周期写入计数初值后，GATE 由低变高电平，则计数器会立即以新计数初值开始计数。

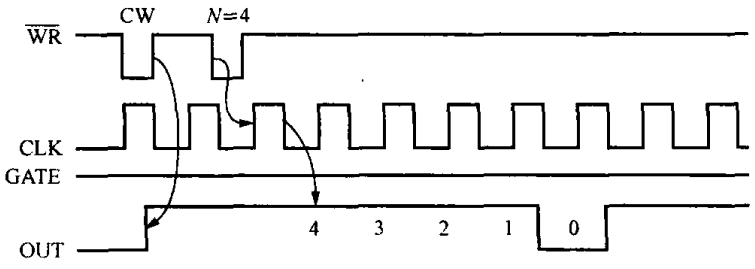


图 7.10 方式 4 的工作波形

5. 方式 4——软件触发选通方式

方式 4 是软件启动、不自动重复计数方式。是由 CPU 写入计数初值来启动计数的工作方式，图 7.10 所示为其工作波形。当 CPU 送入控制字后，OUT 端就输出高电平。若 GATE＝1，则写入计数

初值后计数器立即开始计数。当计数到 0 后计数结束，OUT 端输出一个 CLK 周期宽的负脉冲。只有当 CPU 再将计数初值写入计数寄存器，才会启动另一次计数过程。此方式也受 GATE 信号控制，只有当 GATE 为高电平时，计数才进行，当 GATE 为低电平时则禁止计数。

如果在计数过程中装入新的计数初值，则计数器从下一时钟周期开始就按新的计数值重新开始计数。请注意方式 4 与方式 2 下 OUT 端输出波形的不同。

6. 方式 5——硬件触发选通

本方式为硬件启动、不自动重复计数。CPU 写入控制字后，输出 OUT 变为高电平。CPU 再写入计数初值，8253 不启动计数器工作。当外加门控信号 GATE 端出现一个上升沿跳变时启动计数。计数结束时，OUT 端输出一个宽度为 T_{CLK} 的负脉冲，减 1 计数器为零。之后，OUT 端又变高电平且一直保持到下一次计数结束。方式 5 的工作波形如图 7.11 所示。

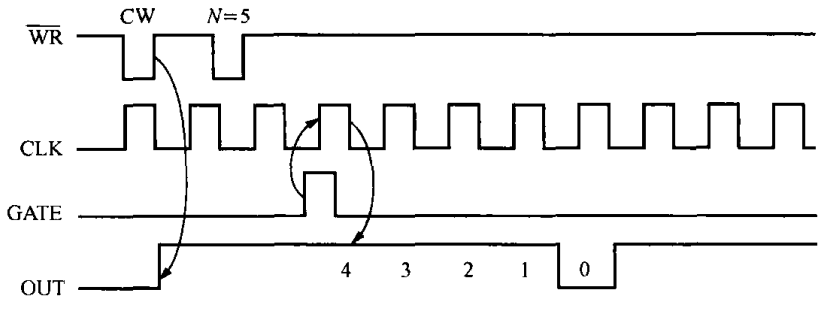


图 7.11　方式 5 的工作波形

在计数过程中，GATE 变低电平不影响计数过程。如果 GATE 产生一个上升沿跳变，则不论当前计数是否完成，会给减 1 计数器重新装入初值，开始新一轮计数。

在计数过程中 CPU 又写入新的计数初值，则新计数值只写入到计数初值寄存器中，而不影响当前计数，只在 GATE 产生一个上升沿的跳变后才按新计数值计数。

表 7.1 总结了 8253 六种工作方式的主要特点，供读者比较。

表 7.1　　8253 六种工作方式的主要特点

工作方式	启动计数	中止计数	自动重复	更新初值	输出波形
0	软件，GATE 高	GATE=0 或 GATE=1→0	否	立即有效	延时时间可变的上升沿
1	硬件，GATE↑	—	否	下一轮有效	宽度为 $N\times T_{CLK}$ 的单一负脉冲
2	软/硬件，GATE↑	GATE=0 或 GATE=1→0	是	下一轮有效	周期为 $N\times T_{CLK}$，宽度为 T_{CLK} 的连续负脉冲
3	软/硬件，GATE↑	GATE=0 或 GATE=1→0	是	下半轮有效	周期为 $N\times T_{CLK}$ 的连续方波
4	软件，GATE 高	GATE=0 或 GATE=1→0	否	立即有效	宽度为 T_{CLK} 的单一负脉冲
5	硬件，GATE↑	—	否	下一轮有效	宽度为 T_{CLK} 的单一负脉冲

注　表中 GATE↑表示该信号上升沿有效。

7.4 8253 的应用编程

7.4.1 8253 的寻址及连接

8253 共占用 4 个 I/O 端口地址；由片内地址 A_1A_0 和$\overline{CS}$来确定，同时与控制信号$\overline{RD}$、$\overline{WR}$相配合，实现 CPU 对 8253 各计数器的读写操作。上述信号的组合对 8253 各端口地址分配与操作功能如表 7.2 所示。

表 7.2 8253 各端口地址及操作功能

$\overline{CS}$	A_1	A_0	$\overline{WR}$	$\overline{RD}$	功能	
0	0	0	0	1	选中计数器 0 号	写计数器 0 的计数初值
			1	0		读计数器 0 的当前计数值
0	0	1	0	1	选中计数器 1 号	写计数器 1 的计数初值
			1	0		读计数器 0 的当前计数值
0	1	0	0	1	选中计数器 2 号	写计数器 2 的计数初值
			1	0		读计数器 2 的当前计数值
0	1	1	0	1	选中控制寄存器	写控制寄存器，由控制字格式中 SC_1、SC_0 位确定写入哪个计数器
0	1	1	1	0	选中控制寄存器	无效

由表 7.2 可知对 8253 任一计数器均可通过各自的端口地址进行读写操作，对控制寄存器根据其端口地址可写入控制字。在向某一计数器写入计数初值时应按控制字中 RL_1、RL_0 的编码规定。当编码为 01 或 10 时，可只写入计数初值的低 8 位或高 8 位，另一字节 8253 默认为 0；当编码为 11 时，必须先写入低字节再写入高字节。若此时只写入了一个字节，就去写其他计数器或控制字，则写入的字节将被解释为计数初值的高字节，从而发生错误。

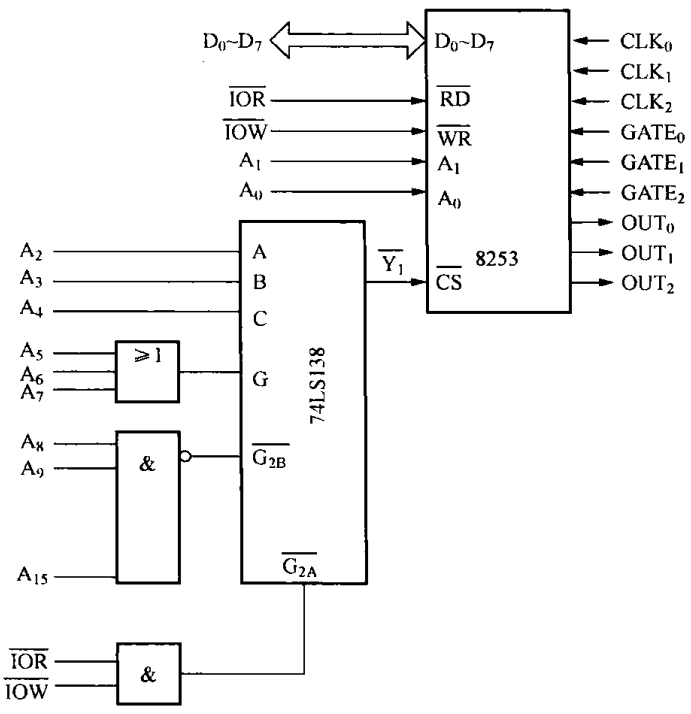


图 7.12 8253 与 8088 系统总线连接图

通过读操作可对 8253 各计数器读出其当前计数值，有直接读和锁存后读两种读出方法，请参见 7.2 的介绍。

可编程计数/定时器 8253 可直接连接到系统总线上。图 7.12 所示是 8253 与 8088 系统总线连接的例子。图中，系统地址总线 A_{15}～A_2 经译码电路产生片选信号选种 8253，8253 各计数器和控制寄存器分别占用 FF04H～FF07H4 个端口地址。

7.4.2　8253 的初始化及应用

8253 计数/定时器有多种功能，在系统中使用 8253 之前必须进行功能的初始化编程。初始化程序通常要放在加电复位后进行或放在用户程序的开始。由于 8253 每个计数器都有自己的地址，控制字中又有专门两位来指定计数器，这使得对计数器的初始化可按任何顺序进行。初始化过程可按以下两种方法进行。

（1）以计数器为单位逐个进行初始化。对某个计数器先写入方式控制字，再写入计数初值的一个字节或高低两个字节。用同样的顺序初始化下一个计数器，直至要初始化的计数器全部初始化完为止。先初始化哪个计数器无关紧要，重要的是对某个计数器的初始化必须按照图 7.13 顺序进行。

（2）先写各计数器的控制字，再写各计数器的计数初值。其初始化顺序如图 7.14 所示。对某个计数器先写控制字再写计数值，这一顺序不能错。但在写控制字或写计数值时，先写哪个计数器则无关紧要。

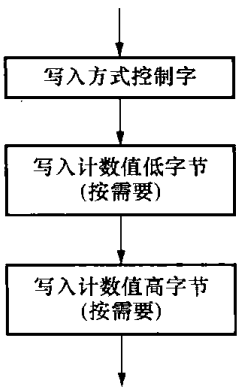


图 7.13　一个计数器初始化顺序

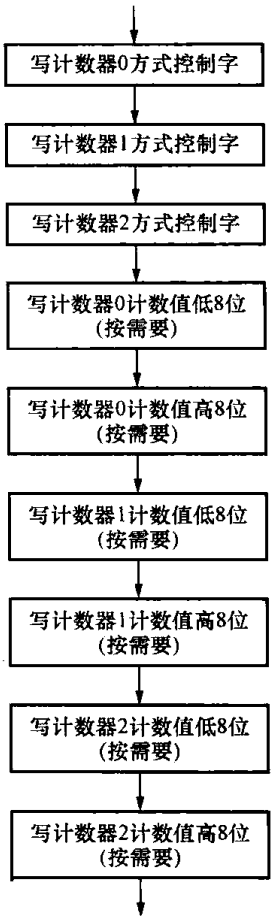


图 7.14　另一种初始化编程顺序

[例 7.1]　计数器 0 工作于方式 3，计数初值为 3412H，按 BCD 计数；计数器 2 工作于方式 2，计数初值为 67H，按二进制计数，设 8253 的端口地址为 40～43H。请编写其初始化编程。

解　按第一种方法初始化程序段：

```
MOV  AL,00110111B  ;写计数器 0 号方式字
OUT  43H,AL
MOV  AX,3412H      ;写计数器 0 号初值
OUT  40H,AL        ;先写低 8 位
MOV  AL,AH         ;再写高 8 位
OUT  40H AL
MOV  AL,10010100B  ;写计数器 2 号方式字
OUT  43H AL
MOV  AL,67H        ;写计数器 2 号初值
OUT  42H,AL
```

按第二种方法初始化程序段：

```
MOV  AL,00110111B  ;写计数器 0 号方式字
OUT  43H,AL
MOV  AL,10010100B  ;写计数器 2 号方式字
OUT  43H AL
MOV  AX,3412H      ;写计数器 0 号初值
OUT  40H,AL        ;先写低 8 位
MOV  AL,AH         ;再写高 8 位
OUT  40H AL
MOV  AL,67H        ;写计数器 2 号初值
OUT  42H,AL
```

[例 7.2] 设 8253 端口地址分别为 304H～307H。计数器 0 工作在方式 0，按二进制计数，计数初值为 2A3BH；计数器 1 工作在方式 1，按二进制计数，计数初值为 4CH；计数器 2 工作在方式 3，作为方波发生器，要求输出 40kHz 方波，已知 CLK_2 时钟输入信号频率为 2MHz。请编写完成上述功能的初始化程序。

解 计数器 0 的初始化程序段如下：

```
MOV    DX,0307H          ;写控制字
MOV    AL,00110000B      ;计数器 0 号方式 0,16 位二进制计数
OUT    DX,AL
MOV    DX,0304H          ;写计数器 0 号的计数初值
MOV    AL,3BH            ;先写低 8 位
OUT    DX,AL
MOV    AL,2AH            ;再写高 8 位
OUT    DX,AL
;计数器 1 初始化程序段如下：
MOV    DX,0307H          ;写计数器 1 号控制字
MOV    AL,01010010B      ;方式 1,8 位二进制计数
OUT    DX,AL
MOV    AL,4CH            ;写计数初值
MOV    DX,0305H
OUT    DX,AL
;计数器 2 初始化程序段,产生 40kHz 方波,初值 = 2MHz/40kHz = 50 = 32H
MOV    DX,0307H          ;写计数器 2 号控制字
MOV    AL,10110110
OUT    DX,AL
MOV    AL,32H            ;写计数初值低 8 位
MOV    DX,0306H
MOV    AL,00H            ;写计数初值高 8 位
OUT    DX,AL
```

[例 7.3] 在 IBM-PC 系统板上使用了一片 8253 定时/计数器，其计数器 0 用于为系统的电子钟提供时间基准，它的输出端作为系统的中断源，接到 8259 的 IR_0 端；计数器 1 用于 DRAM 的定时刷新；计数器 2 主要用作机内扬声器的音频信号源，可输出不同频率的方波信号。图 7.15 是简化了的 IBM-PC 内 8253 的连接图，其接口地址采用部分译码方式，占用的设备端口地址为 40H～5FH。本例编程中用到了其中的 40H～43H 这四个地址。三个计数器的输入时钟频率均为 1.19MHz。

解 由于计数器 0 的作用是为系统提供时间基准，将其初始化为方式 3，产生周期的方波信号，计数初值选为最大计数值，即十六进制的0000H (65536)。根据方式 3 的工作原理可知，OUT_0 输出方波信号的频率为 1.19MHz/65536≈18.2Hz。由于 OUT_0 与 8259 的中断请求输入线 IR_0 相连接，所以每秒将会产生 18.2 次中断请求，该中断请求用于维护系统的日历钟。

计数器 1 初始化为方式 2，计数初值取 18，18/1.19MHz≈15μs，即每 15μs 对动态存储器刷新一次。

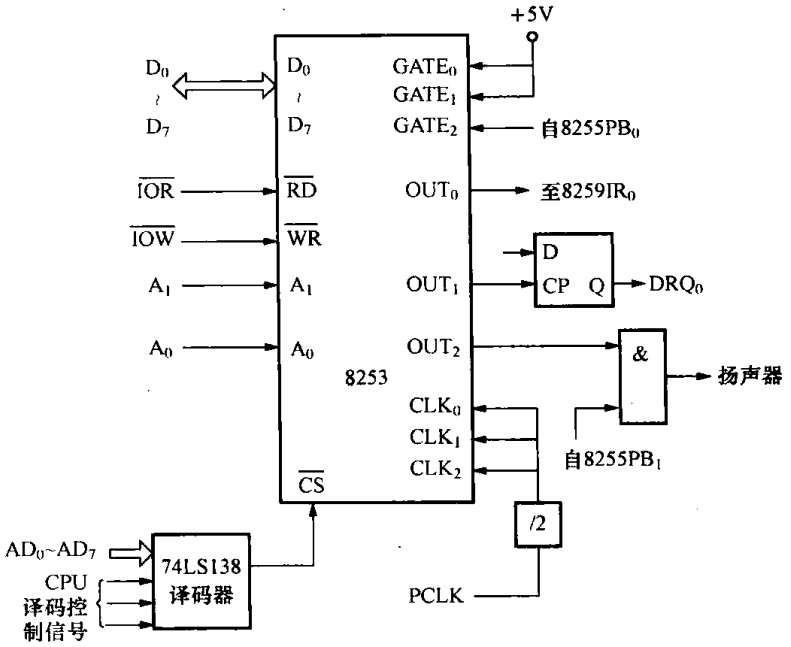


图 7.15　IBM-PC 中 8253 的连接简图

计数器 2 初始化为方式 3，控制扬声器发出频率为 1kHz 的声音，故取时间常数为 1190。PC 中，要使扬声器发声，还必须使 8255 的 PB_1 和 PB_0 输出高电平（设 8255 的 B 口地址为 61H）。

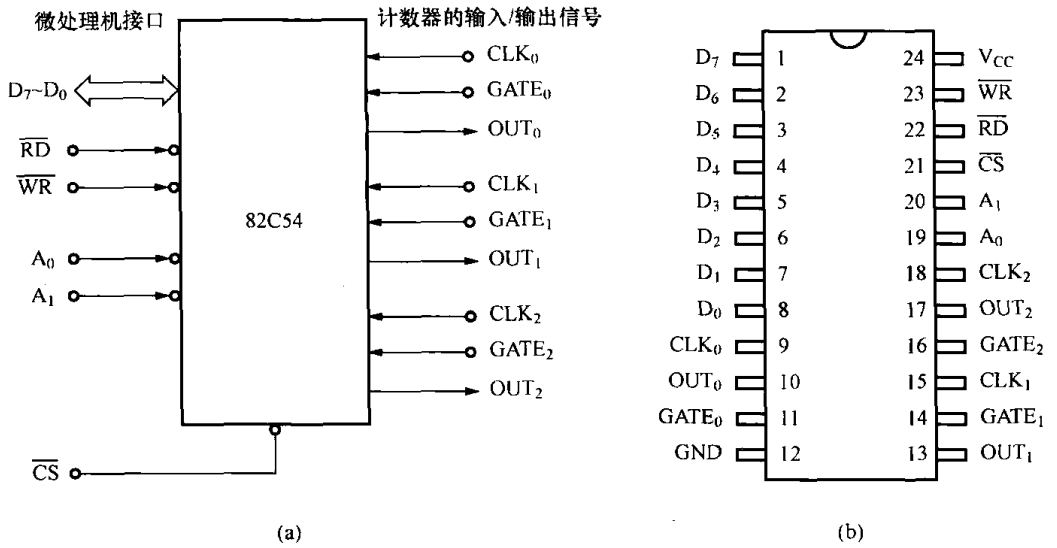
IBM-PC 中 8253 的初始化程序如下：

```
;计数器 0 初始化
        MOV   AL,36H          ;选择计数器 0,写双字节计数器,方式 3,二进制计数
        OUT   43H,AL          ;控制字写入控制寄存器
        MOV   AL,0            ;选最大计数值(65536)
        OUT   40H,AL          ;写低 8 位计数值
        OUT   40H,AL          ;写高 8 位计数值
;计数器 1 初始化
        MOV   AL,54H          ;选择计数器 1,低 8 位单字节计数值,方式 2,二进制计数
        OUT   43H,AL          ;
        MOV   AL,18           ;
        OUT   41H,AL          ;计数值写入计数器 1
;计数器 2 初始化
        MOV   AL,0B6H         ;选择计数器 2,双字节计数值,方式 3,二进制计数
        OUT   43H,AL          ;
        MOV   AX,1190         ;
        OUT   42H,AL          ;送低字节到计数器 2
        MOV   AL,AH           ;高字节计数值送 AL
        OUT   42H,AL          ;高 8 位计数值写入计数器 2
        IN    AL,61H          ;读 8255 的 B 口
        MOV   AH,AL           ;将 B 口内容保存
        OR    AL,03           ;使 PB0 = PB1 = 1
```

```
;OUT   61H,AL          ;使扬声器发声
      ⋮
MOV   AL,AH           ;恢复 8255 的 B 口状态
OUT   61H,AL
```

7.5 8254　简　介

8254 是 8253 的增强型芯片，具备 8253 的全部功能，凡是用 8253 的场合完全可用 8254 代替，其源程序也不用修改可直接使用。8253 和 8254 的引脚定义完全相同，图 7.16 所示为 82C54 定时/计数器的框图及引脚图。两者的差别如下。



(a)　　(b)

图 7.16　82C54 定时/计数器的框图及引脚图

(a) 框图；(b) 引脚图

(1) 计数脉冲的频率范围不同。8253-4 型允许的频率范围为 0～2MHz，8254 允许的频率范围为 0～8MHz，而 8254-2 型允许的频率范围为 0～10MHz。IBM PC/XT机中使用的定时/计数器接口芯片是 8253-5 型，实现声音和时钟接口。IBM PC/AT 机内使用的定时/计数器接口芯片是 8254-2 型。

(2) 内部结构有所不同。在每个计数通道内，8253 仅包括一个控制寄存器，8254 增加了一个状态寄存器，且状态寄存器的信息可由 CPU 读出。

(3) 8254 提供了同时锁存三个计数器当前计数值和状态信息的操作。程序员可用一条指令锁存全部三个计数器的当前计数值和状态信息，称该工作方式为读回命令，格式如下：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	COUNT	STATUS	CNT2	CNT1	CNT0	0

D_7，D_6＝11　　读回命令特征标志；

D_5＝0　　锁存所选计数器的计数值；

$D_4=0$　　锁存所选计数器的状态；

$D_3=1$　　选计数器 2 号；

$D_2=1$　　选计数器 1 号；

$D_1=1$　　选计数器 0 号；

$D_0=0$　　保留，用于将来扩充，现在必须为 0。

例如，同时锁存计数器 1 号和计数器 2 号的当前计数值，读回命令字为 11011100B=DCH；若同时锁存三个计数器的状态信息，则读回命令字为 11101110B=EEH。该读回命令必须写入 8254 的控制寄存器。

由读回命令锁存的计数值和状态信息可用输入指令读入 CPU。一个计数器如果同时锁存了计数值和状态，用两条输入指令从同一个 I/O 地址读入。第一次输入指令读入的一定是状态；第二条输入指令读入计数值。如果对同一个计数器发出多次读回命令，但每次并不读取锁存值，那么只有第一次发出的读回命令是有效的，后面的均无效。

每个计数器锁存的状态，由输入指令读回一个字节的状态信息，其格式如下：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
OUTPUT	NULL COUNT	RW_1	RW_0	M_2	M_1	M_0	BCD

1）D_7：计数器输出端 OUT_i 的状态。1：输出引脚为高电平；0：输出引脚为低电平。

2）D_6：计数寄存器的初值是否装入减 1 计数寄存器中。1：向计数器写入了方式控制字或向计数寄存器写入了初值但没有装入减 1 计数寄存器；0：计数寄存器初值已装入减 1 计数寄存器。所以在读入计数值之前，应该读回和测试状态信息的 D_6 位是否为 0 以确定读的计数值是新的还是旧的。

3）$D_5\sim D_0$：为初始化编程时写入该计数器的控制字的相应部分。

[例 7.4]　图 7.17 所示为 82C54 与 80386DX 系统总线的连线图，试编写初始化程序完成计数器 0 号，方式 0，计数初值=3412H，按二进制计数；计数器 1 号，方式 2，计数初值=200H，按 BCD 计数；计数器 2 号，方式 4，计数初值=1EFEH，按二进制计数。

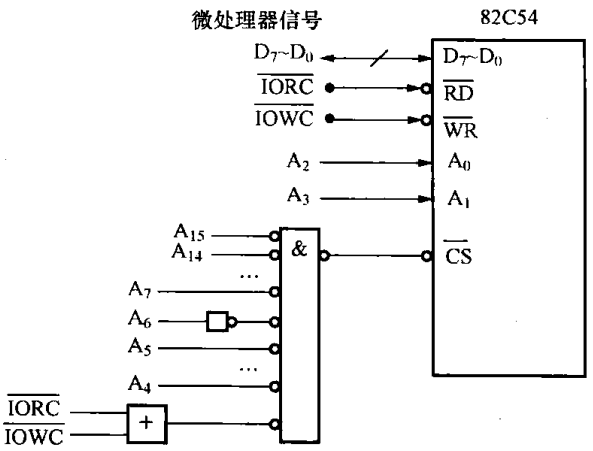


图 7.17　82C54 与 80386DX 系统总线的连接图

解　从图中片选信号的译码电路可知：

$A_{15}A_{14}\cdots\cdots A_6A_5A_4=00\times\times\times\times\times\times0100B=0000\ 0000\ 0100B$

对 82C54 中计数器 0 号，其片内 $A_1A_0=00$，故 CPU 地址线 $A_3A_2=00$。则：

计数器 0 号的地址码=0000 0000 0100 0000B=40H；

计数器 1 号的地址码=0000 0000 0100 0100B=44H；

计数器 2 号的地址码=0000 0000 0100 1000B=48H；

控制寄存器的地址码=0000 0000 0100 1100B=4CH。

初始化程序段如下：

```
MOV     AL,30H          ;计数器 0 号,方式 0,二进制计数
OUT     4CH,AL
MOV     AL,12H          ;写计数器 0 号计数初值低字节
OUT     40H,AL
MOV     AL,34H          ;写计数器 0 号计数初值高字节
OUT     40H,AL
MOV     AL,55H          ;计数器 1 号,方式 2,BCD 计数
OUT     4CH,AL
MOV     AL,00H          ;写计数器 1 号计数初值低字节
OUT     44H,AL
MOV     AL,02H          ;写计数器 1 号计数初值高字节
OUT     44H,AL
MOV     AL,0B8H         ;计数器 2 号,方式 4,二进制计数
OUT     48H,AL
MOV     AL,0FEH         ;写计数器 2 号计数初值低字节
OUT     48H,AL
MOV     AL,1EH          ;写计数器 2 号计数初值高字节
OUT     44H,AL
```

习　　题

7.1　8253 的功能是什么？请举几个应用 8253 芯片的例子。

7.2　8253 有几个独立的计数器？各有几种工作方式？各种工作方式的名称是什么？

7.3　8253 中计数器 2 的输入、输出是什么？假设计数器 8253 工作在方式 4 下，其计数器装入初值为 200H，问选通脉冲输出时有多长的时间延迟？

7.4　若写入计数初值相同，8253 的方式 0 和方式 1 的不同之处是什么？

7.5　8253 计数器工作在哪些方式时，是 GATE 的上升沿启动计数？

7.6　设 8253 三个计数器的端口地址为 200H、201H、202H，控制寄存器端口地址 203H。试编写程序段，读出计数器 2 的内容，并把读出的数据装入寄存器 AX。

7.7　设 8253 三个计数器的端口地址为 40H、41H、42H，控制寄存器端口地址 43H。输入时钟为 2MHz，使计数器 1 周期性的发出脉冲，其脉冲周期为 1ms，试编写初化程序段。

7.8　设 8253 计数器的时钟输入频率为 1.91MHz，为产生 25kHz 的方波输出信号，应向计数器装入的计数初值为多少？

第8章 微机系统的并行I/O接口

8.1 并行I/O接口概述

如前所述，CPU与外设进行输入输出可以采用程序查询方式、中断方式和DMA方式。但是无论采用哪种方式，CPU总是通过接口电路才能与外设连接并通信。所以，接口电路一边与CPU连接，另一边与外设连接。对于高速I/O设备通常以并行通信方式与微机系统通信。随着大规模集成电路技术的发展，生产了许多通用可编程接口芯片。按接口芯片数据传送的方式可以分为并行接口与串行接口。并行接口就是在通信过程中能够同时传送数据所有位的接口芯片。

并行接口通常以字节或字为单位传送，速度快，效率高；适合近距离传送；无固定数据格式。并行接口按数据传送方向可分为输入接口和输出接口；按传输数据形式可分为单向传送接口和双向传送接口；按接口电路结构可分为简单接口和可编程接口；按传送信息类型可分为数字接口和模拟接口。本章主要介绍可编程并行接口。可编程接口是指其功能方式可由微型机指令来设定，通过执行不同初始化程序，使一个可编程接口芯片工作于不同功能方式，从而能与多种外部设备进行连接。

8.2 并行I/O接口芯片Intel 8255A

Intel 8255A是一种通用可编程并行I/O接口电路芯片，可为多种并行I/O设备提供接口，是Intel公司为8085、80X86系列微处理器配套的接口芯片，也可以和其他微处理器系统相配。一片8255A内部有3个独立的输入输出端口（简称A口、B口、C口）及一个控制寄存器；既可作输入接口，也可作为输出接口。80X86系统中常采用8255A作为键盘、扬声器、打印机等外设的接口电路芯片。

82C55A接口芯片，其功能内部结构与8255A一样，引脚及功能全兼容。

8.2.1 8255A的内部结构及外部引脚

8255A芯片采用40脚双列直插封装，单一+5V电源，全部输入/输出信号均与TTL电平兼容；有三个输入/输出端口，每个端口都可通过编程设定为输入端口或输出端口；芯片还有为输入/输出提供控制联络信号、端口寻址信号的读写控制逻辑、A组和B组控制电路；还有用来与CPU数据总线连接的数据总线缓冲器。8255A内部结构框图与引脚排列如图8.1所示，它由以下四部分组成。

1. 三个并行输入/输出端口（端口A、端口B、端口C）

8255A有A、B、C三个并行输入/输出端口，都是8位并行数据端口，可作为输入或输出端口，其功能可由程序设定。端口A（PA_0～PA_7）为数据输入/输出端口，输出按锁存器/缓冲器工作，输入按锁存器工作。端口B（PB_0～PB_7）为数据输入/输出端口，输出按锁存器/缓冲器工作，输入按缓冲器工作。端口C（PC_0～PC_7）为数据端口或控制端口，分成4位的两组，与A口和B口组合，以输出控制信号/输入状态信号。端口C口也可以作为一般的

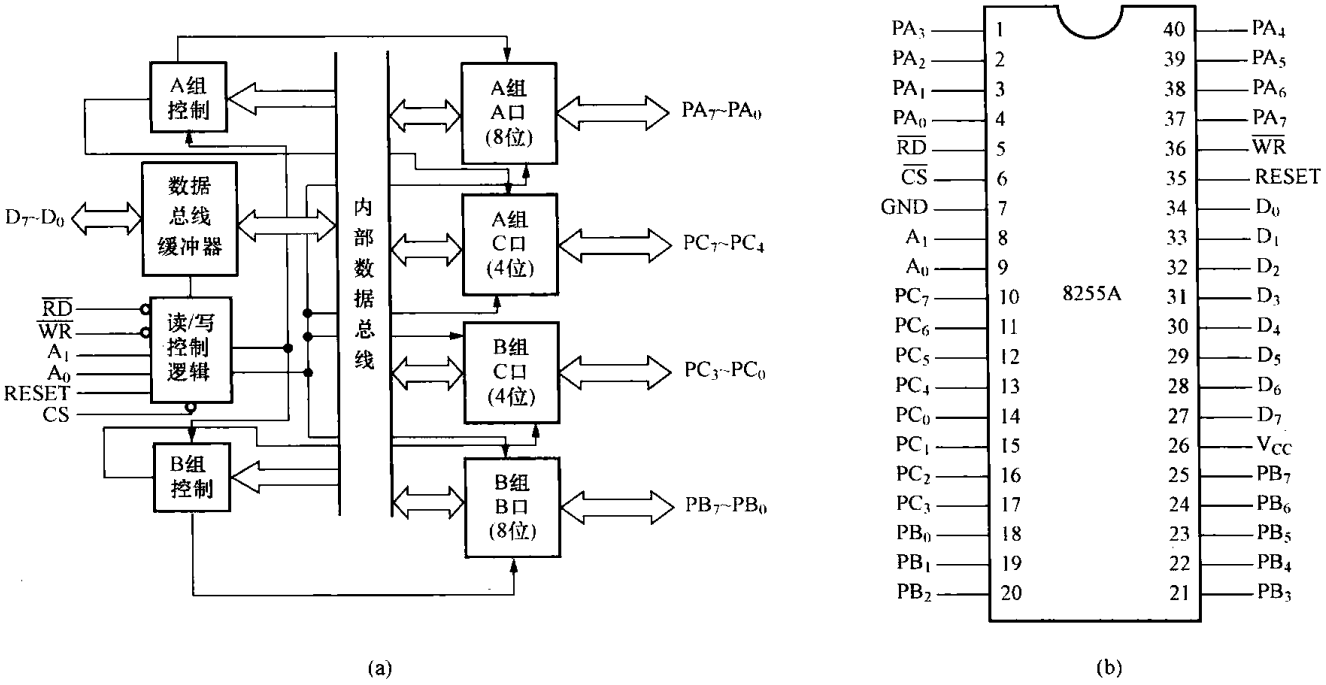


图 8.1 8255A 内部结构框图与引脚排列图

(a) 内部结构框图；(b) 引脚排列图

I/O端口使用。当 A 口、B 口作为应答式的 I/O 口使用时，C 口分为 A 组 C 口（PC_4～PC_7）、B 组 C 口（PC_0～PC_3），规定分别用来作为 A 口和 B 口的应答控制线使用。各端口的功能如表 8.1 所示。

表 8.1　　8255A 端 口 功 能

工作方式	A 口	B 口	C 口
0	基本输入/输出端口 输入不锁存、输出锁存	同 A 口	同 A 口
1	选通输入/输出端口 输入/输出均可锁存	同 A 口	PC_4～PC_7 作为选通式 A 口的联络信号线；PC_0～PC_3 作为选通式 B 口的联络信号线
2	选通双向输入/输出端口，均可锁存	不用	PC_0～PC_7 用作 A 口的选通双向联络信号线

A 口、B 口、C 口的三组 8 位输入输出信号线分别与连接的外设数据线连接，以实现数据的并行传送。当 A 口、B 口工作于方式 1 或 2 时，C 口的输入输出信号线作为控制联络线与外设的控制与状态信号线连接。

2. 读/写控制逻辑

读/写控制逻辑接收来自 CPU 的地址信息及一些控制信号，然后向 A 组、B 组控制电路发送命令。其控制信号有以下几种：

$\overline{CS}$—片选信号，低电平有效。该信号有效时允许 8255A 与 CPU 交换信息。通常由端口

地址线（16 位地址号取自 A_{15}～A_2，8 位地址号取自 A_7～A_2）译码产生。CPU A_0 地址线一般为低电平，保证端口都是偶地址，数据在 D_0～D_7 总线上传输。

$\overline{RD}$—读信号，低电平有效。允许 CPU 从 8255A 端口中读取数据或外设状态信息。通常接系统总线的$\overline{IOR}$信号。

$\overline{WR}$—写信号，低电平有效。允许 CPU 将数据、控制字写入到 8255A 中。通常接系统总线的$\overline{IOW}$信号。

RESET—复位信号，高电平有效。它清除 8255A 所有控制寄存器内容，并将各端口都置成输入方式。

A_1、A_0—8255A 片内端口地址选择信号线。实际使用时，A_0、A_1 通常接系统地址总线的 A_0、A_1，通过片内译码用来选择各端口及内部控制寄存器的地址，如表 8.2 所示。与$\overline{RD}$、$\overline{RW}$、$\overline{CS}$信号相配合控制信息传送方向，如表 8.3 所示。

表 8.2　　片内各端口地址选择

A_1	A_0	选择	A_1	A_0	选择
0	0	A 口	1	0	C 口
0	1	B 口	1	1	控制寄存器

表 8.3　　与控制信息配合后的操作组合

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	操　作	
0	0	0	1	0	端口 A→数据总线	读操作
0	1	0	1	0	端口 B→数据总线	
1	0	0	1	0	端口 C→数据总线	
0	0	1	0	0	数据总线→端口 A	写操作
0	1	1	0	0	数据总线→端口 B	
1	0	1	0	0	数据总线→端口 C	
1	1	1	0	0	数据总线→控制字寄存器	

3. A 组和 B 组控制电路

这两组接收来自 CPU 的读/写控制部分的信号和 CPU 送入的控制字，然后分别决定各端口的功能。A 组控制电路控制 A 口和 C 口的高 4 位；B 组控制电路控制 B 口和 C 口的低 4 位。还可根据控制字的要求对 C 口的某位实现“置 0”或“置 1”操作。

4. 数据总线缓冲器

这是一个双向三态的 8 位缓冲器，8255A 通过它与系统的数据总线直接相连，实现在 CPU 和 8255A 间传送控制字、数据和状态信息。

图 8.2 给出了 8255A 各引脚及端口在系统中的连接示意图。

8.2.2　8255A 的控制字与初始化编程

8255A 可与各种并行 I/O 设备连接，为适应不同 I/O 设备 8255A 有三种工作方式，

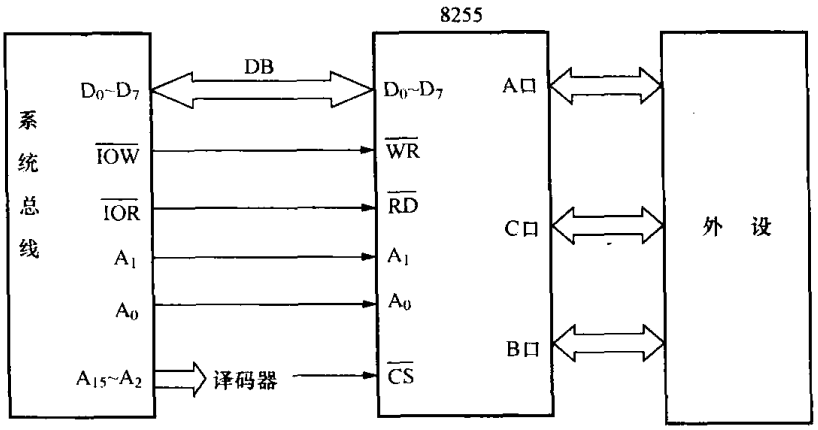


图 8.2 8255A 各引脚及端口在系统中的连接示意图

且对端口 C 各位又可进行按位操作，这些由 CPU 输出到 8255A 的控制字来设定。8255A 有两个控制字：一个是工作方式控制字；另一个是对端口 C 的“置位/复位”控制字。

1. 工作方式控制字

工作方式控制字用来设定各端口的工作方式及数据传送方向，8255A 开始工作前 CPU 通过执行接口初始化程序来设定，接口初始化程序一般放在程序开始。端口 A 可工作在方式 0、1、2 三种方式之一；端口 B 可工作在方式 0、1 两种方式之一；而端口 C 只能工作在方式 0。8255A 工作方式控制字格式如图 8.3 所示。

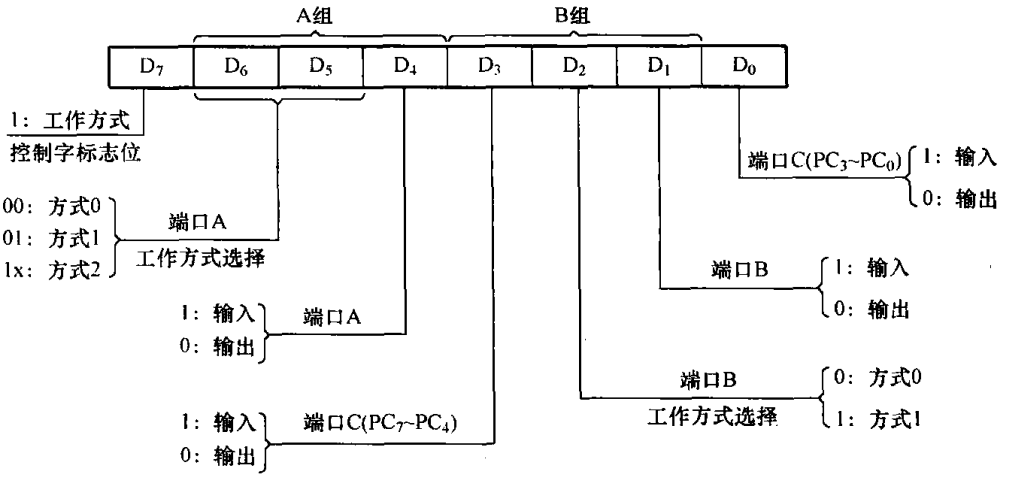


图 8.3 8255A 工作方式控制字格式

其中，D_7 位用于区分 8255A 的两种控制字。当 $D_7=1$ 时，为工作方式控制字；$D_7=0$ 时为对端口 C 置位/复位控制字。D_6～D_3 用来控制 A 组的工作方式，即 A 口的工作方式及输入输出，C 口的高四位工作方式及输入输出。D_2～D_0 用来控制 B 组的工作方式，即 B 口的工作方式及输入输出，C 口的低四位工作方式及输入输出。只要 CPU 对 8255A 送入方式控制字就可以决定 A 口、B 口、C 口的工作方式及相应的操作功能。这种对可编程序接口电路送入控制字，从而设定接口功能的程序称为接口初始化程序。

例如，系统要求 8255A 的端口 A 工作在方式 0、输入；端口 B 工作在方式 0、输出；端口 C 高 4 位为输出，低 4 位为输入。根据上述要求可求得方式控制字为 91H，设 8255A 控

制寄存器的地址号为 43H，则其初始化程序如下：

```
MOV    AL,  91H          ;CPU 将控制字 91H 经 AL 输出
OUT    43H, AL           ;送至 8255A 控制寄存器中
```

当 8255A 控制寄存器为 16 位地址号时，CPU 对 8255A 输出控制字，应采用寄存器间接寻址的输出指令。如控制寄存器地址号为 0D43H 时，则初始化程序为：

```
MOV    DX,  0D43H        ;控制寄存器地址号存入 DX 中
MOV    AL,  91H          ;控制字经 AL 送控制寄存器
OUT    DX,  AX
```

2. 置位/复位控制字

置位/复位控制字格式如图 8.4 所示。

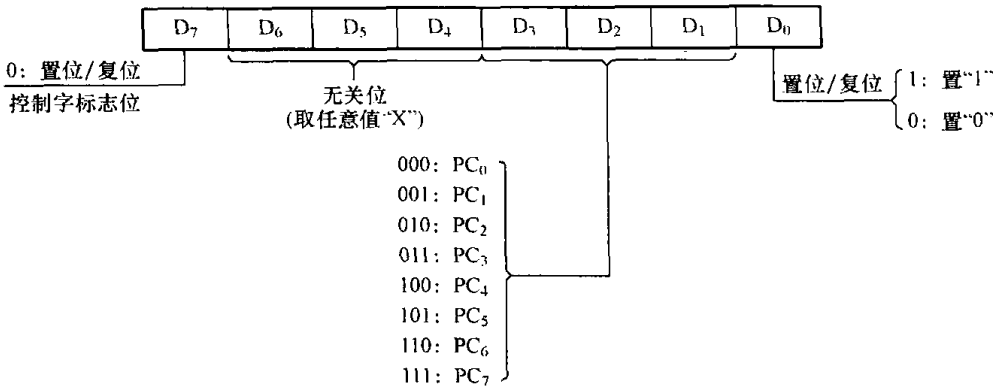


图 8.4　8255A 端口 C 置位/复位控制字格式

对 C 口的某位置位/复位控制字，用于指定 C 口某位输出高电平还是低电平，作为输出控制信号。如用于控制开关的通（置 1）/断（置 0），继电器的吸合/释放，电机的启/停控制等。其中，D_0 位用于设定是置 1 还是置 0 操作；D_3、D_2、D_1 用于指定 C 口的 PC_0～PC_7 中的哪一位按位操作；D_6、D_5、D_4 没有用，可以是任意值，一般置为全 0。

例如，要求对 C 口 PC_7 位实现置 0 操作，其控制字为 00001110B（0EH，其中 D_6～D_4 可取 8 种值，故控制字不唯一）。设控制寄存器的端口地址为 43H，则初始化程序为：

```
MOV    AL,  00001110B    ;置 PC7 = 0 的控制字
OUT    43H, AL           ;控制字送 8255A 控制寄存器中
```

8.2.3　8255A 的三种工作方式

8255A 的端口 A、B、C 均可通过编程（执行初始化程序）分别设置成输入或输出端口。但同样是输入或输出端口，若工作方式不同，其应用场合、工作时序及引脚信号的定义也不一样。现分别介绍这三种工作方式。

1. 方式 0——基本输入输出

端口 A、B、C 均可以工作在方式 0，是一种基本输入/输出工作方式。即当外设始终处于传送数据的准备就绪状态时，不需要专用应答联络信号，故 CPU 可以通过 8255A 随时与外设进行输入/输出数据。这就是输入输出技术中无条件输入输出，典型的例子是以开关或计数器状态作为输入信号，以发光二极管（LED）作为显示输出，如图 8.5 所示。

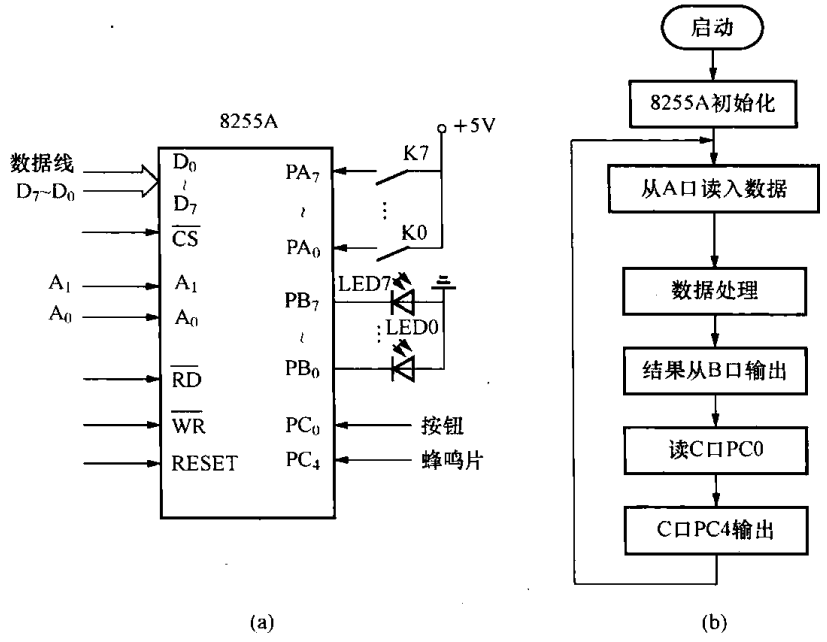


图 8.5　8255A 方式应用举例

（a）开关，发光二极管显示接口；（b）操作流程图

在方式 0 下，8255A 是分成彼此独立的 8 位 A 口、8 位 B 口、C 口高 4 位、C 口低 4 位共 4 组，每组只能同时设定为输入或输出，不能其中的某几位作为输入，另一部分作为输出。

在方式 0 时，可用查询方式来进行数据传送。此时，通常以 A 口和 B 口作为数据口，而用 C 口某些位作为查询式控制与状态信号线使用。具体用 C 口的哪条引线作为控制与状态信号线可以由用户来指定。例如，可用 C 口高 4 位引线作为选通信号线，用置 0 及置 1 的位操作产生正脉冲的选通信号；用 C 口低 4 位作为外设状态输入线（RDY）使用。图 8.6 所示为方式 0 时应答查询式读/写操作流程图。

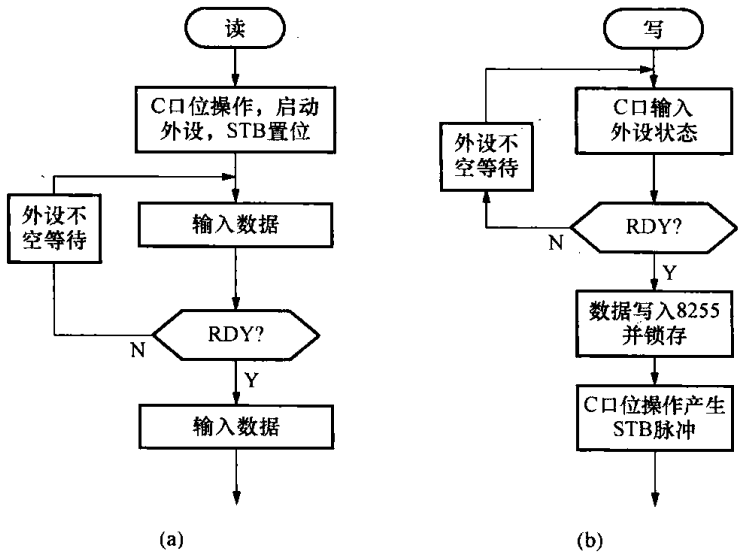


图 8.6　8255A 方式 0 时应答查询式读/写操作流程图

（a）读操作流程图；（b）写操作流程图

2. 方式 1——选通输入输出

8255A 的工作方式 1 主要是与异步工作或中断方式的外设实现选通输入输出的。在这种方式下，A 口和 B 口作为独立的数据输入口或输出口，可由初始化程序指定，但数据的输入输出要在选通信号控制下来完成。这些选通信号用 C 口的某些位来提供。为叙述方便起见，下面分别以 A 口、B 口均以方式 1 选通输入或方式 1 选通输出说明其操作过程。

（1）方式 1 下 A 口、B 口均为选通输入。此时利用 C 口的 6 条线

作为选通信号线，选通信号线连接如图 8.7 所示。所用到 C 口的信号线是固定不变的，A 口使用 PC_4、PC_5 和 PC_3，B 口使用 PC_2、PC_1 和 PC_0，分别表示 A 口和 B 口的 3 条控制线 $\overline{STB}$、IBF 和 INTR。

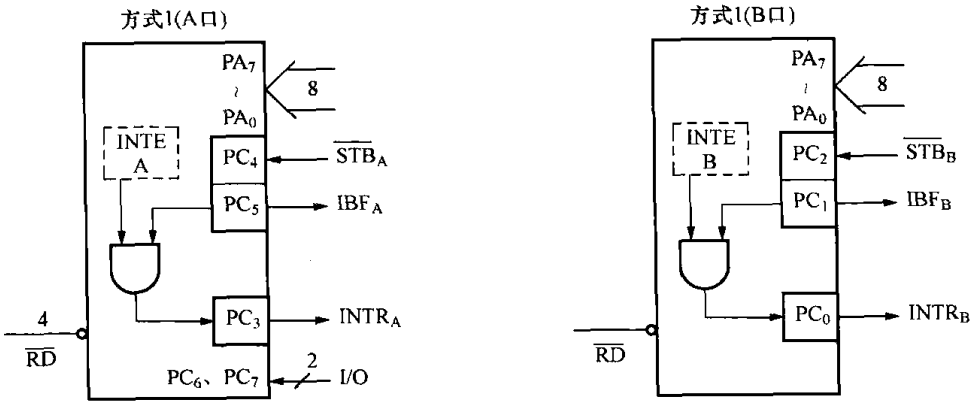


图 8.7　方式 1 输入时选通信号线连接

$\overline{STB}$选通信号，低电平有效。它是外设给 8255A 的信号，表示外设输入数据已准备好。此信号规定用 C 口的 PC_4 位（对 A 口）和 PC_2 位（对 B 口）。当 PC_4（或 PC_2）变低电平时，外设已将数据放置在 PA_0～PA_7（或 PB_0～PB_7）数据线上。当$\overline{STB}$由低变高时，数据锁存入端口中。

IBF 输入缓冲器满信号，高电平有效。它是 8255A 向外设发出的响应信号（ACK）。当 IBF 有效时，表示数据已锁存到端口的数据输入缓冲器。此信号规定使用 C 口的 PC_5（对 A 口）和 PC_1（对 B 口）。当 CPU 从 8255A 读取数据后，利用$\overline{RD}$的上升沿使 IBF 复位为低电平。IBF 为低电平是 8255A 向外设表明此端口原输入数据已被 CPU 取走，外设可输入新数据。

INTR 中断请求信号，高电平有效。是向 CPU 发出中断请求。CPU 响应中断后，在中断服务程序中 CPU 执行从 8255A 端口读取数据指令（IN 指令）时，产生$\overline{RD}$有效信号，它一方面将 8255A 锁存的数据读入到 CPU 中并延迟一段时间，使 INTR 无效。另一方面利用$\overline{RD}$信号的上升沿使 IBF 复位。

方式 1 下数据输入过程描述如下：

1）外部设备发出低电平有效的选通信号$\overline{STB}$，在$\overline{STB}$有效期间将数据锁存入 8255A 的数据输入缓冲器。

2）8255A 数据输入缓冲器满，则接口使 IBF＝1 作为$\overline{STB}$的应答信号，表示 8255A 的输入缓冲器有一个数据还没被 CPU 读走；外部设备据此信号暂缓送下一个数据。

3）若 INTE＝1（中断允许），则$\overline{STB}$从低电平到高电平的上升沿使 INTR＝1，向 CPU 提出中断请求。

4）INTR 信号可通过 8259 向 CPU 提出中断请求，要求 CPU 从 8255A 的端口上读取数据。CPU 响应中断并通过输入指令读取数据后使 IBF、INTR 变为无效。上述过程各控制信号的操作时序如图 8.8 所示。

应指出，8255A 片内有一个中断允许触发器 INTE，当 INTE＝0 时，表示禁止中断，当 INTE＝1 时，表示允许中断。其置“0”与置“1”是通过对 PC_4（A 组）和 PC_2（B 组）进行位操作来实现的。在方式 1 中，对 PC_4（或 PC_2）的位操作只影响 INTE 触发器的状

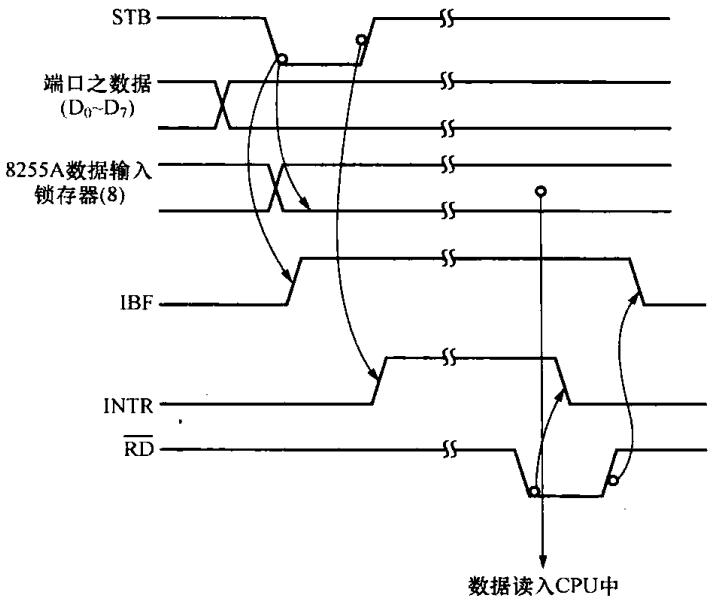


图 8.8 方式输入操作时序图

态，而不影响 PC_4（或 PC_2）引脚的电平状态。

在方式 1 输入时，C 口多余的两条线（PC_6，PC_7）归入 A 组，它可以作为方式 0 的输入/输出线或作为位操作用，其工作状态及初始化编程与 A 口无关。

（2）方式 1 下 A 口、B 口均为选通输出。此时利用 C 口的 6 条线作为选通信号线，选通信号线连接如图 8.9 所示。所用到 C 口的信号线是固定不变的，A 口使用 PC_7、PC_6 和 PC_3，B 口使用 PC_1、PC_2 和 PC_0，分别表示 A 口和 B 口的 3 条控制线$\overline{OBF}$，$\overline{ACK}$和 INTR。

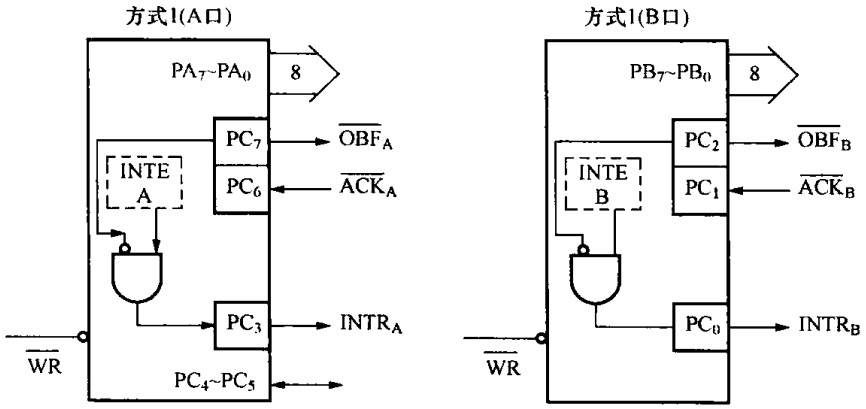


图 8.9 方式 1 输出时选通信号线连接

$\overline{OBF}$输出缓冲器满信号，低电平有效。是 8255A 输出给外部设备的控制信号。当其有效时，表示 CPU 已将数据输出到 8255A 端口，外部设备可从此端口取数据。

$\overline{ACK}$响应信号，低电平有效。是外部设备从 8255A 端口取得数据后，发给 8255A 的响应信号。$\overline{ACK}$有效，表明外设已取走数据。8255A 收到$\overline{ACK}$信号后，在此信号的下降沿使$\overline{OBF}$变为高电平，通知外部设备，8255A 没有新输出数据；在$\overline{ACK}$的上升沿使 INTR 变高电平，向 CPU 提出中断请求，要求 CPU 向 8255A 发出下一个输出数据。

INTR 中断请求信号，高电平有效。如果允许中断（INTE=1），且$\overline{ACK}$、$\overline{OBF}$均为高电平时，则经 PC_3（A 口）或 PC_0（B 口）引脚发出此中断请求信号。

方式 1 下数据输出过程描述如下：

1）CPU 接收中断请求，中断服务程序中的输出指令（$\overline{WR}$低电平有效）将输出数据送 A 口或 B 口锁存，并使 INTR=0，变为无效。

2）$\overline{WR}$信号的上升沿使$\overline{OBF}$为低电平，向外部设备表明在连接的端口上已有新的有效数据，外设可以到此端口的输出锁存器中取数据了。

3）外部设备取数据，发选通响应信号$\overline{ACK}=0$，使$\overline{OBF}=1$。

4）外部设备取走数据后，$\overline{ACK}=1$，$\overline{OBF}=1$，若 INTE=1，则又使 INTR=1，通知 CPU 输出下一次数据。上述过程各控制信号的操作时序如图 8.10 所示。

3. 方式 2——选通双向输入输出

8255A 只有端口 A 可以工作在选通双向输入/输出工作方式下，是方式 1 情况下 A 口输入、输出的结合。外设利用端口 A 的 PA_0～PA_7 的 8 位数据线与 CPU 进行双向通信，既能发送数据，又能接收数据。此时 C 口的控制线$\overline{STB}_A$，$\overline{OBF}_A$，$\overline{ACK}_A$，IBF_A 的意义与方式 1 相同，且 INTE1 和 INTE2 分别由 PC_5 和 PC_4 的置位/复位操作来控制。其 A 口选通信号连接如图 8.11 所示。

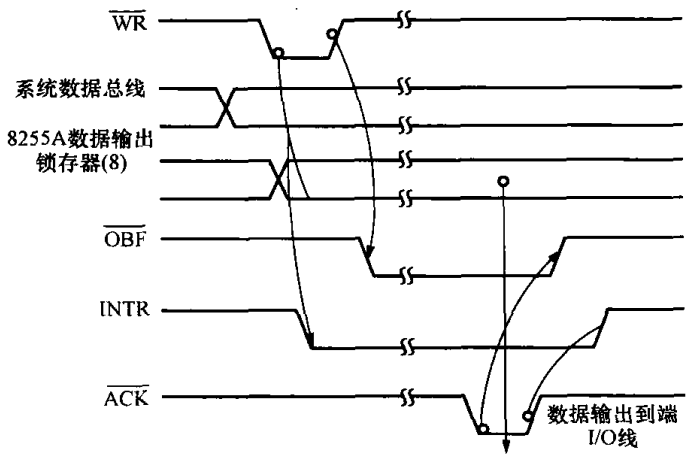


图 8.10　方式 1 输出操作时序图

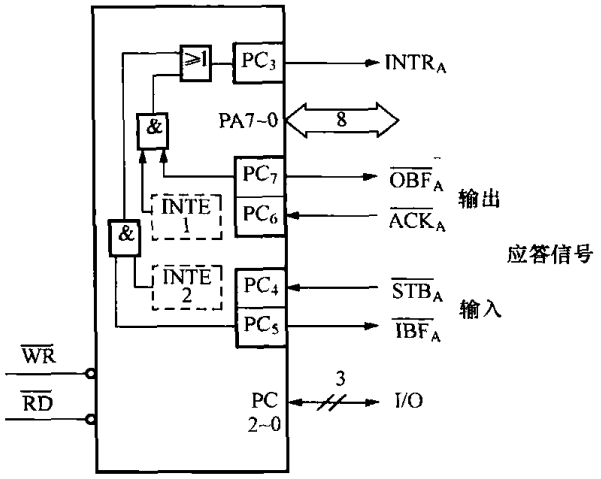


图 8.11　方式 2 时 A 口选通信号线连接

方式 2 下 A 口数据输入过程：

1）外设发出$\overline{STB}_A$ 信号给 8255A，向 8255A 送数据，使数据锁存入 A 口。然后 8255A 发出 IBF_A 有效信号给外设，表示 A 口已收到数据。

2）$\overline{STB}_A$ 信号无效。若 8255A 允许中断（INTE1=1），则向 CPU 发送中断请求信号，使 $INTR_A$ 有效。

3）CPU 响应中断，在中断服务程序中执行输入指令读取 A 口锁存器中的数据。

4）$\overline{RD}$信号无效，使 $INTR_A$、IBF_A 无效，开始下个数据的读入过程。

方式 2 下 A 口数据输出过程：

1）CPU 执行输出指令向 A 口写数据，使输出数据锁存至 A 口。

2）$\overline{WR}$信号使 $INTR_A$ 变低电平，同时$\overline{OBF}_A$ 有效，通知外设可从 A 口读取数据。

3）外设接到$\overline{OBF}_A$ 信号后给 8255A 发$\overline{ACK}_A$ 低电平有效信号，将锁存于 A 口的数据读入外设中。

4）有效的$\overline{ACK}_A$ 信号使$\overline{OBF}_A$ 无效、$INTR_A$ 变有效，向 CPU 发中断请求准备输出下个数据。方式 2 的上述过程各控制信号的操作时序如图 8.12 所示。

应该注意的是：8255A 方式 1 主要用于中断方式输入/输出操作，但也可用于连续查询方式。此时，可根据输入/输出操作的不同，A 口和 B 口的 $INTR_A$ 和 $INTR_B$ 分别由 C 口的 PC_6、PC_7 位和 PC_4、PC_5 位作为状态信号。当用查询方式输入/输出时，则必须首先查询状

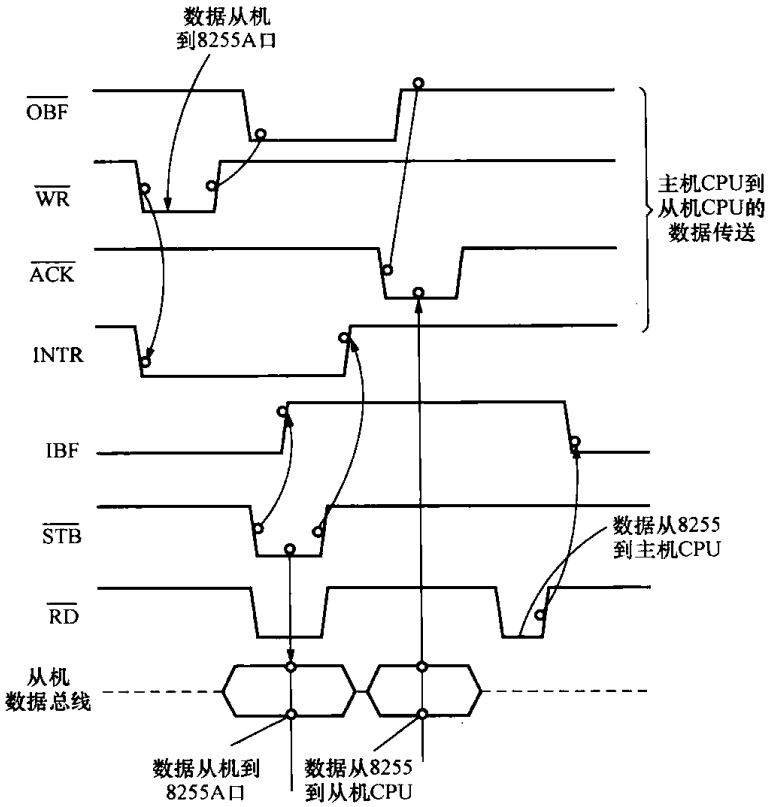


图 8.12 方式 2 操作时序图

态字内容，才能和有关 A 端口/B 端口进行数据交换。

8255A 工作于方式 0 时，端口 C 可用于数据传送，但当 8255A 工作于方式 1 或方式 2 时，端口 C 的部分引脚或全部引脚便用作应答联络线使用，这时端口 C 的内容是反映端口 A 或 B 及相应外部设备的状态，称为方式 1 或方式 2 的状态字，且其状态字的格式各不相同，如图 8.13 所示。CPU 如需了解各端口的工作状态，可通过读 C 口的内容，对相应的位进行检测，便可读出状态信息。

若在方式 2 时采用查询式中断，则可从 C 口读入状态字，并检查$\overline{IBF_A}$及$\overline{OBF_A}$标志位来实现查询式输入/输出操作。

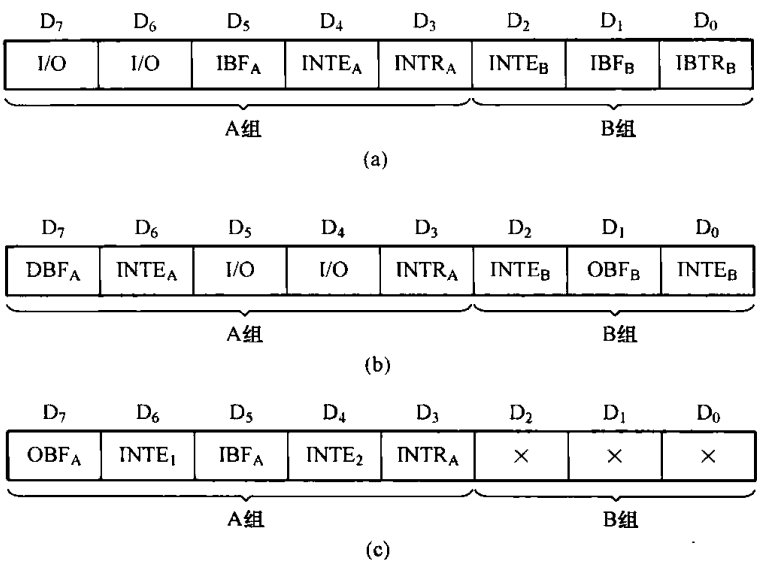


图 8.13 8255A 的状态字

(a) 方式 1 输入口时的状态字；(b) 方式 1 输出口时的状态字；

(c) A 口方式 2 时的状态字

目前，随着微处理技术的日益普及，多微处理机系统已日趋增多，而且这种多机系统有时采用一个主机多个从机形式。为了实现主—从机间并行传送数据，并避免多个从机同时使用总线，采用 8255A 器件作为这种系统的接口极为方便。

8.3　Intel 8255A 应用举例

8.3.1　PC 机系统板上的 8255A

在 PC/XT 中，系统板上的外围接口电路主要是由可编程接口芯片 8255A-5 以及相关电路组成，其连接示意图如图 8.14 所示。

PC/XT 微机中，8255A 的端口地址范围为 60H～63H。A、B、C 三个口均工作于方式 0。A 口在加电自检时工作在输出状态，输出当前被检测部件的标识信号。此时的 B 口也工作于输出状态，而 C 口工作为输入状态，因此其方式控制字为 89H。

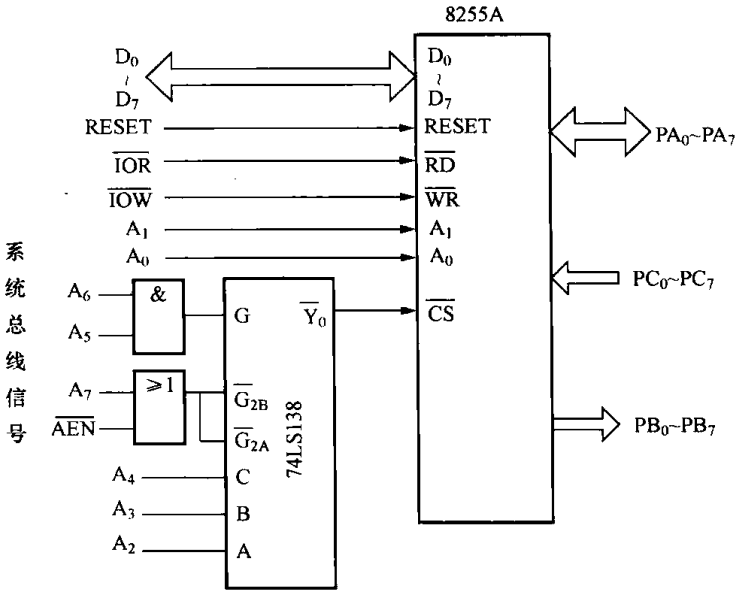


图 8.14　PC/XT 中的 8255A-5 连接示意图

在正常工作时，A 口作为输入口，用来读取键盘扫描码；B 口和 C 口仍分别为输出和输入口。B 口用于输出系统内部的控制信号，控制系统板部分电路的动作，如定时器，扬声器，键盘，允许 RAM 奇偶校验，允许 I/O 通道校验以及控制系统配置开关信号的读取。C 口用来读取系统内部的状态信号，包括系统配置开关的状态，8253 的 OUT_2，I/O 通道奇偶校验和 RAM 奇偶校验的状态等。此时的控制字为 99H。

8.3.2　8255A 应用举例

8255A 的应用编程首先根据题意和系统、接口、外设的连接图确定其工作过程；根据工作过程选择各接口芯片的工作方式；确定各芯片的方式控制字；编写相应控制程序。图 8.15 所示是按中断方式通过 A 口进行输入/输出工作的主程序和中断服务程序流程图。

[**例 8.1**]　利用 8255A 作为输出设备打印机的接口，此时 8255A 通常设置成工作方式 0，采用查询方式，图 8.16（a）为打印机接口电路原理图。设 8255A 端口及控制寄存器的地址分别为：A 口：0380H，B 口：0381H，C 口：0382H，控制寄存器：0383H。当 CPU 需要打印一个数据时，打印机接收 CPU 传送数据的过程如下：

（1）CPU 经 8255A 先查询打印机忙（BUSY）信号。若打印机处于忙状态（BUSY＝1），CPU 不能向打印机输出新数据；当打印机不忙（BUSY＝0）时，CPU 能向打印机输出新数据。

（2）CPU 经 8255A 的端口 A 输出数据，即 PA_0～PA_7 数据线上有数据。

（3）CPU 经 8255A 的 PC_7 送出一个选通信号$\overline{STB}$，低电平有效，把 PA_0～PA_7 数据线上的数据存入打印机缓冲器中。

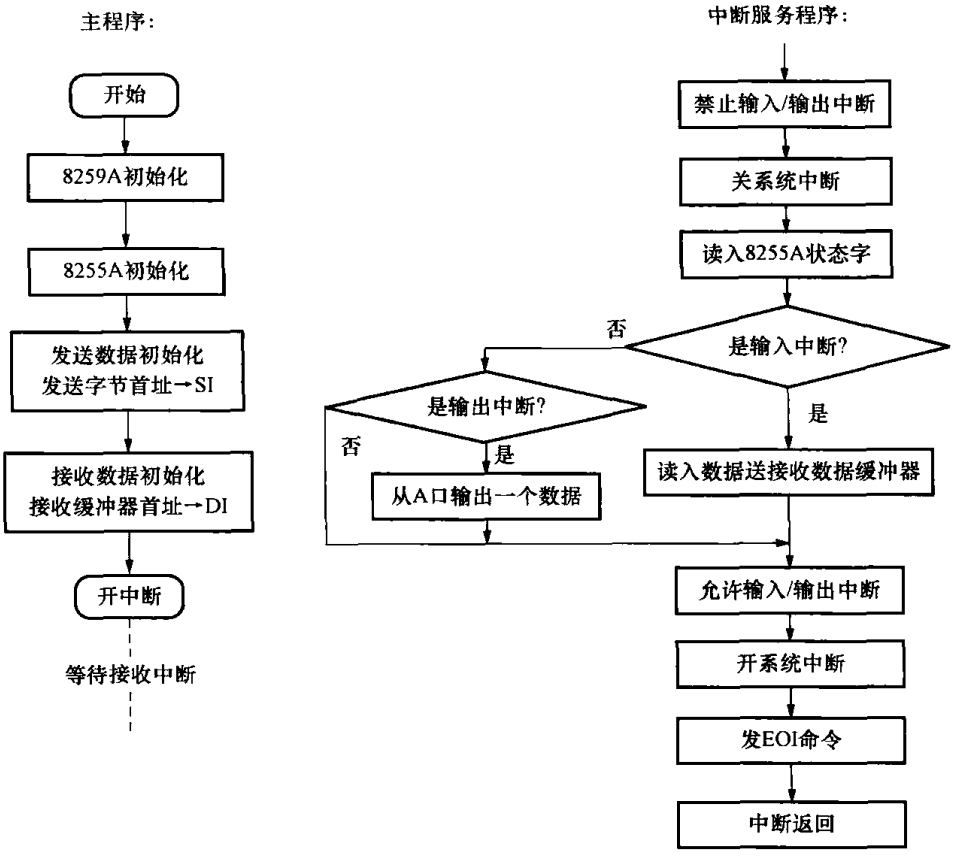


图 8.15 8255A 按中断方式工作的主程序和中断服务程序流程图

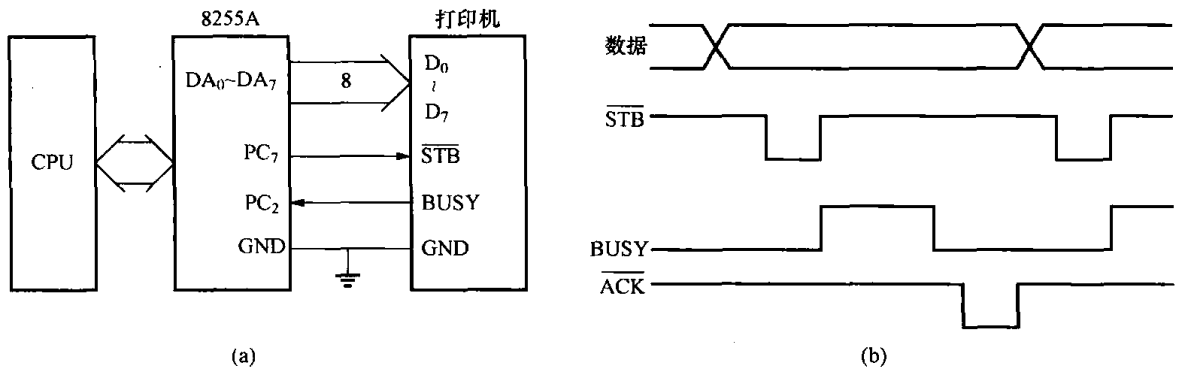


图 8.16 8255A 与打印机接口

(a) 电路原理图；(b) 打印机操作时序图

(4) 打印机接收到数据后，使 BUSY=1 将“忙”信号送至 PC_2，表示打印机正在处理 CPU 输入的数据，数据处理完毕置 BUSY=0，表示打印机准备接受 CPU 输出的下一个数据。如此重复，直至打印完所有的数据。

另外，如果用中断方式实现打印机输出，当一个数据输出完毕，一般打印机会向外送出一个负脉冲回答信号$\overline{ACK}$给 CPU，表示一个数据已经处理完毕。打印机可利用$\overline{ACK}$信号向 CPU 申请中断，CPU 响应中断后，又可送出下一个打印数据。本例中 8255A 工作于方式 0，按查询方式输出，故没有利用此信号。

根据上述打印机的工作过程，其操作时序如图 8.16 (b) 所示。

8255A 接口的初始化及打印机驱动程序流程如图 8.17 所示。初始化程序如下：

```
BEGIN:  MOV  DX,0383H          ;控制寄存器地址 DX
        MOV  AL,10000001B      ;送工作方式 0 控制字(A 口)方式 0 输出，
                               ;PC7～PC4 输出，PC3～PC0 输入
        OUT  DX,AL
        MOV  AL,00001111B      ;送 C 口置 1/置 0 控制字，置 PC7 = 1，STB = 1
        MOV  SI,2000H          ;待打印机数据存放内存的首地址
        MOV  CX,0FFH           ;待打印数据个数
CONP:   MOV  DX,0382H          ;从 C 口检测 BUSY = 0?。若忙 = 1，则等待，= 0，则
                               ;CPU 可送打印数据
LPST:   IN   AL,DXH            ;读 PC2(判 BUSY = 0?)
        AND  AL,04H
        JNZ  LPST              ;若 BUSY = 1，等待；BUSY = 0 则向下执行
        MOV  AL,[SI]           ;从内存中取待打印的数据
        MOV  DX,0380H
        OUT  DX,AL             ;待打印数据输出至 8255A 的端口 A
        MOV  AL,00001110B      ;送置 PC7 = 0 的置位控制字符至控制寄存器
        MOV  DX,0383H          ;输出 PC7 = 0 信号，则STB = 0 低电平，产生选通信号
        OUT  DX,AL
        NOP                    ;使STB信号低电平有一定宽度，以保证传送
        NOP                    ;至打印机数据稳定
        NOP
        MOV  AL,00001111B      ;置 PC7 = 1 即STB = 1 高电平，利用STB上升
        OUT  DX,AL             ;沿将数据打入到打印机数据寄存器中
        NIC  SI                ;修改指针，指向下一次欲打印的数据
        NEC  CX                ;数据个数 - 1
        JNZ  CONP              ;未打完，继续
        HLT                    ;所有数据打印完毕，暂停
```

上述程序执行前，应事先在内存 2000H 处开始存放 256 个欲打印字符的 ASCII 码值。

[例 8.2] 利用 8255 作为打印机接口来打印字符串，字符串长度放在 DS 段的 COUNT 单元，要打印的字符存放在从 DATA 单元开始的数据区中。若使 8255 工作在方式 1 下，并利用中断方式进行数据传送，则 8255 与打印机的电路连接示意如图 8.18 所示。设 8255 的端口地址为 0FBC0H～0FBC3H，分别对应 A 口、B 口、C 口和内部控制寄存器。

从打印机的工作时序图可知，打印机每接收一个字符后，会送出一个低电平的响应信号$\overline{ACK}$。利用这个信号，可使工作于方式 1 的 8255 通过中断来打印字符。

设置 8255 为方式 1，A 口为输出口，PC_7 作为$\overline{OBF}$信号输出端，PC_6 作为$\overline{ACK}$信号输入端，PC_3 作为 INTR 信号输出端接到 8259 的 IR_2 端，所以中断类型号为 0AH。

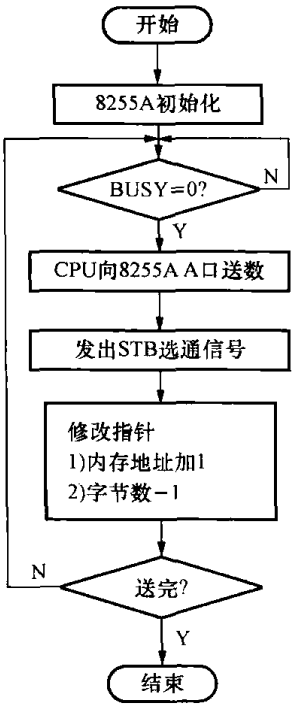


图 8.17 打印机驱动程序流程图

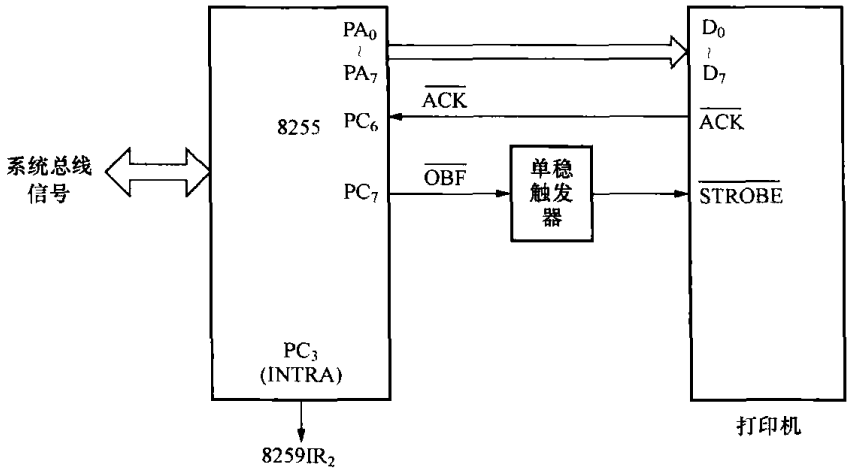


图 8.18 8255 工作于方式 1 下与打印机的连接示意图

输出时，先输出一个空字符，以引起中断过程。在中断中输出要打印的字符，利用$\overline{OBF}$的下降沿触发一单稳触发器，产生打印机所需要的$\overline{STROBE}$脉冲，将字符锁存到打印机中。接收到字符后，打印机发出$\overline{ACK}$，清除$\overline{OBF}$标志并产生有效的 INTR 输出，形成新的中断请求，CPU 响应中断后再输出下一个字符。

为简单起见，在初始化 8255 时，使 B 口工作于方式 0 输出，C 口其余 5 条线均定义为输出，故控制字为 10100000B，即 0A0H。

要使 PC_3 能够产生中断请求信号 INTR，还必须使 A 口的中断请求允许状态 INTE=1。这可通过 8255 的置位/复位操作将 PC_6 置 1 来实现，即要写 C 口的位操作控制字。

以下是向打印机输出字符的程序，包括主程序和中断处理子程序两部分。主程序完成以下三项工作：将中断服务子程序的入口地址送中断向量表，开中断以及 8255 的初始化。中断服务子程序完成字符输出。设 8259 的端口地址为 0FF00H 和 0FF01H。

```
MAIN:   PUSH DS
        LEA  DX, PRINT
        MOV  AX, SEG PRINT          ;设置中断向量
        MOV  DS,  AX
        MOV  AL,  0AH
        MOV  AH,  25H
        INT  21H
        POP  DS                     ;8255 初始化:A 口方式 1,输出,B 口方式 0,输出
        MOV  DX, 0FBC3H             ;C 口其余的 5 条线输出
        MOV  AL, 0A0H               ;使 PC6 置 1(INTE = 1),允许 8255 产生中断
        OUT  DX,  AL
        MOV  AL,  0DH
```

```
        OUT   DX,   AL
        MOV   AL,   00H
        MOV   DX,   0FBC0H
        OUT   DX,   AL          ;从 A 口输出一个空字符,引发第一次中断
        MOV   AX, OFFSET  DATA
        MOV   STRD,   AX        ; 设置字符串偏移地址
        MOV   AX,   SEG  DATA
        MOV   STRD+2,   AX      ; 设置字符串段地址
        ST1                     ; 开中断
        ......
```

中断服务子程序如下：

```
PRINT:  PUSH    SI
        PUSH    AX
        PUSH    DS
        LDS     SI,             DWORD PTR STRD
        MOV     STRD,   SI      ;保存新的串指针
        MOV     DX,   0FBC0H
        OUT     DX, AL          ;输出字符到 8255 的 A 口
        MOV     AL,   20H
        MOV     DX, 0FF00H      ;8259 的 OCW2
        OUT     DX,   AL        ;送中断结束命令给 8259
        POP     DS
        POP     AX
        POP     SI
        IRET                    ;中断返回
```

[例 8.3] 用 8255A 并行接口芯片实现键盘接口，其电路原理如图 8.19 所示。图中按键排列成 4 行 4 列，8255A 的 C 口设置为方式 0，并将 PC_7～PC_4 与各行线相连，PC_3～PC_0 与各列线相连。

键盘是一种常用的微机系统输入设备，键盘结构有线性键盘和矩阵键盘两种形式。详见 6.4.1。

下面以 4×4 矩阵键盘为例来说明用 8255A 作为矩阵键盘接口的反转法识别按下的键。以图 8.19 所示为例，具体步骤如下：

1）将 PC_7～PC_4 设定为输出，PC_3～PC_0 设定为输入。从行线输出全 0，即 PC_7～PC_4 输出全 0，然后从 PC_3～PC_0 读入列线的内容，若读入的数据中有一位为 0，则表明与该位对应的列线上有一个键按下，存储此内容作为列值，转第 2）步，否则循环执行本步骤。

2）将 PC_7～PC_4 设定为输入，PC_3～PC_0 设定为输出。把第 1）步读入的内容输出到列线上，即把“列值”从 PC_3～PC_0 输出，接着从 PC_7～PC_4 读入行线的内容，其中必有一位为 0，为 0 的位所对应的行线就是被按下键所在的行，存储此内容作为“行值”。将行值和列值组合在一起，用查表的方法即可得到按键的键号。

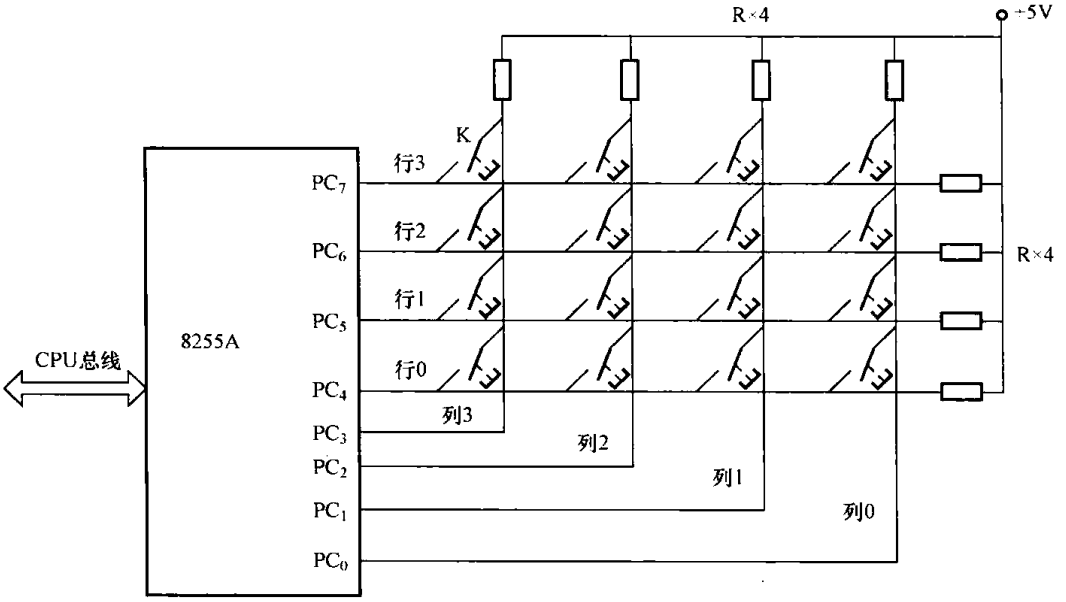


图 8.19 矩阵式键盘接口电路原理图

例如，若第 0 行第 2 列的键被按下，则第 1）步从列线读入的列值为 1011B；第 2 步中再将 1011B 从列线输出，从行线读入的行值为 1110B，两者组合得到该键的行列值组合为 11101011B。

因为在键盘扫描过程中要反转行线与列线的输入输出方向，所以此法被称为反转法。

如果读到的行、列内容中有一个以上的“0”，说明有多键按下，一般需要进行重新扫描。如果是组合键，则需要另行处理。读者若有需要，请自行参考有关资料。

关于抖动问题。一般每个键在按下和松开时，都会经历短时间的抖动后才到达稳定接通或断开。抖动的持续时间因键的质量有所不同，通常为 5～20ms。在确定按键和键释放时必须避开这段不稳定的抖动时间，才能正确检测识别。去抖动一般有两种方法：一种是软件延时，即发现有键按下或释放时，软件延时一段时间再检测；另一种是硬件消抖法，即用基本 RS 触发器、单稳电路、RC 滤波器等来消除抖动。

解 以下是与图 8.19 相对应的采用反转法的按键识别程序。设 8255A 端口 A 的地址为 40H，端口 B 的地址为 41H，端口 C 的地址为 42H，控制寄存器地址为 43H。

```
START:  MOV AL,10000001B        ;设方式 0,端口 C 高 4 位输出,低 4 位输入
        OUT 43H,AL
        MOV AL,0
        OUT 42H,AL              ;端口 C 高 4 位(行线)为 0
WAIT:   IN AL,42H               ;读入端口 C 低 4 位(列线)的状态
        AND AL,0FH              ;保留低 4 位
        CMP AL,0FH              ;检查有键按下否(即是否存在有 0 的位)
        JE WAIT                 ;相等,即全 1 表示无键按下,循环继续检测
        MOV AH,AL               ;否则,保存列值
        MOV AL,10001000B        ;设方式 0,端口 C 高 4 位输入,低 4 位输出
        OUT 43H,AL              ;即端口 C 高低 4 位反转输入输出方向
        MOV AL,AH
```

```
OUT 42H,AL                ;把列值反向输出到列线(端口 C 低 4 位)上
IN AL,42H                 ;读入行线(端口 C 高 4 位)状态
AND AL,0F0H               ;保留高 4 位
OR AL,AH                  ;组合行值和列值

<查表确定按键的键号>
……
```

[例 8.4] 用 8255A 并行接口芯片实现 24 键的键盘接口，其接口电路如图 8.20 所示。图中按键排列成 8 行 3 列，8255A 的端口 A 与各行线相连，端口 B 的低 3 位 PB_2～PB_0 与各列线相连。

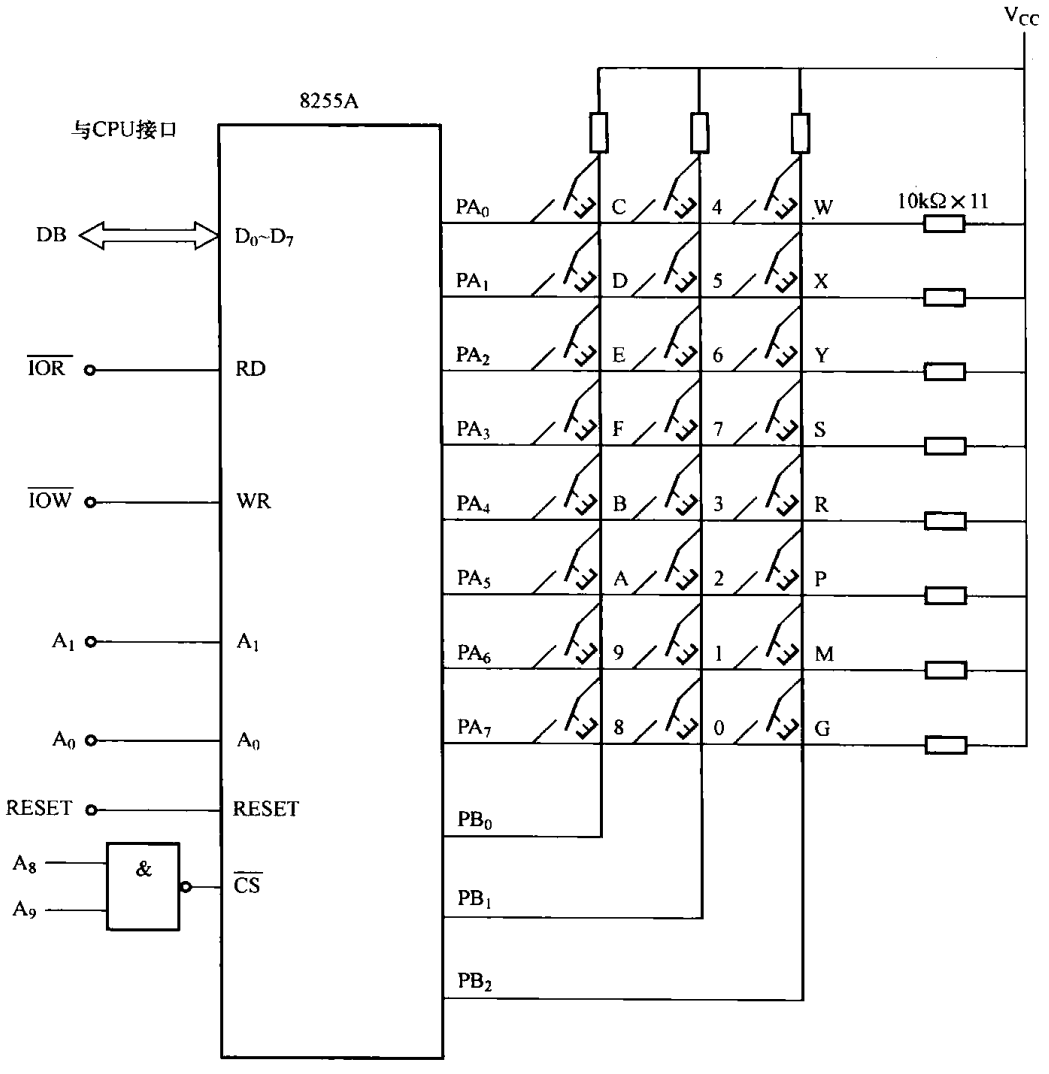


图 8.20　8255A 实现键盘接口电路

解　根据反转法原理，扫描 24 个键需要两个 I/O 端口，且这两个 I/O 端口还应该既能作为输入口又能作为输出口。设 8255A 各端口 A、B、C 及控制寄存器地址为 300H～303H。以下程序实现对键盘的扫描，并将按键显示在 CRT 上，如果按下 Y 键，则退出该程序返回到 DOS 状态下。

```
PA55    EQU    300H
PB55    EQU    301H
P55CTL  EQU    303H
DATA SEGMENT
TABLE   DW  0101H,0102H,0104H,0108H,0110H,0120H,0140H,0180H    ;PB0 列中各键键值
        DW  0201H,0202H,0204H,0208H,0210H,0220H,0240H,0280H    ;PB1 列中各键键值
        DW  0401H,0402H,0404H,0408H,0410H,0420H,0440H,0480H    ;PB2 列中各键键值
CHAR    DB  'CDEFBA9845673210WXYSRPMG'          ;字符 ASCII
MES     DB  'PLAY ANY KEY IN THE SMALL KEYBOARD',0aH,0dH
        DB  'IT WILL BE ON THE SCREEN! END WITE Y ! ', 0aH,0dH,'S'
DATA    ENDS
    CODE    SEHMENT
    MAIN    PROC FAR                            ;定义为段间过程
        ASSUME CS:CODE,DS:DATA
START:  MOV  AX,DATA                            ;装入段基址
        MOV  DX,AX
        MOV  DX,OFFSET MES                      ;9 号功能调用
        MOV  AH,09
        INT  21H
KY:     CALL KEY                                ;从缓冲区中取一字符,并显示
        CMP  [DI],'Y'
        JNZ  KY
        MOV  AX,4000H
        INT  21H                                ;若[DI] = 'Y',则返回 DOS
        RET
    MAIN  ENDP                                  ;主程序结束
    KEY   PROC  NEAR                            ;段内过程(找出闭合键的行、列值)
KST:    MOV  AL,82H
        MOV  DX,P55CTL
        OUT  DX,AL
WAIT1:  MOV  AL,00                              ;A 口低电平输出,行线低电平
        MOV  DX,PA55
        OUT  DX,AL
        MOV  DX,PB55                            ;读列线,查找有没有闭合键
        IN   AL,DX
        CMP  AL,0FFH                            ;没有闭合键,等待
        JZ   WAIT1
        PUSH AX                                 ;列线值暂存堆栈
        PUSH AX
        MOV  CX,1000H                           ;去抖动延时
DELAY:  LOOP DELAY
        MOV  DX,P55CTL                          ;反转 A 口为输入口,B 口为输出口
        MOV  AL,90H
```

```
        OUT   DX,AL
        MOV   DX,PB55                ;列值由 B 口输入
        POP   AX
        OUT   DX,AL
        MOV   DX,PA55                ;读行值
        IN    AL,DX
        POP   BX                     ;从堆栈中弹出列值至 BX 中
        MOV   AH,BL                  ;列值→AH
        NOT   AH                     ;AH 取反
        MOV   SI,OFFSET TABLE        ;取参数 Table 的偏移地址→SI
        MOV   DI,OFFSET CHAR         ;取字符参数 Char 的偏移地址→DI
        MOV   CX,24                  ;24 个字符长
TT:     CMP   AX,[SI]                ;键值与 Table 中字符相比较
        JZ    NN                     ;找到相应字符,转 NN 语句
        DEC   CX                     ;长度修正
        JZ    KST                    ;没找到,转 KST 语句继续查找
        ADD   SI,2
        INC   DI
        JMP   TT
NN:     MOV   DL,[DI]                ;闭合键字符→DL
        MOV   AH,02                  ;2 号功能调用,显示字符
        INT   21H
        PUSH  DX                     ;DX 暂存堆栈
        MOV   AL,82H                 ;A 口控制字→控制寄存器,A 口输出,B 口输入
        MOV   DX,P55CTL
        OUT   DX,AL
WAIT2:  MOV   AL,00                  ;等待闭合键释放
        MOV   DX,PA55
        OUT   DX,AL
        MOV   DX,PB55
        IN    AL,DX
        CMP   AL,0FFH
        JNZ   WAIT2
        POP   DX
        RET
KEY:    ENDP                         ;过程结束
CODE:   ENDS                         ;代码段结束
        END   START                  ;源程序结束
```

8.4　82C55A 应用于 32 位 CPU 的 I/O 接口

80386 和 80486 在 I/O 独立编址方式时，也是用地址总线的 A_{15}～A_0 来确定 I/O 端口地

址的，共有 $2^{16}=64K$ 个端口。由于 80386/80486 CPU 都是 32 位外部数据总线，可直接进行 16 位（字）或 32 位（双字）的 I/O 操作。为此，若用 82C55A 作为接口电路，应把地址连续的 2 个 82C55A 组合成一个 16 位端口；用 4 个 82C55A 组合成一个 32 位端口。组合形成的多字节端口用其最低字节地址作为组合端口的起始地址。数据线的连接，仍是低地址存放组合数据的低字节；高地址存放组合数据的高字节。

如图 8.21 所示，即为其连接的原理示意图。为了能实现 CPU 32 位数据通过 82C55A

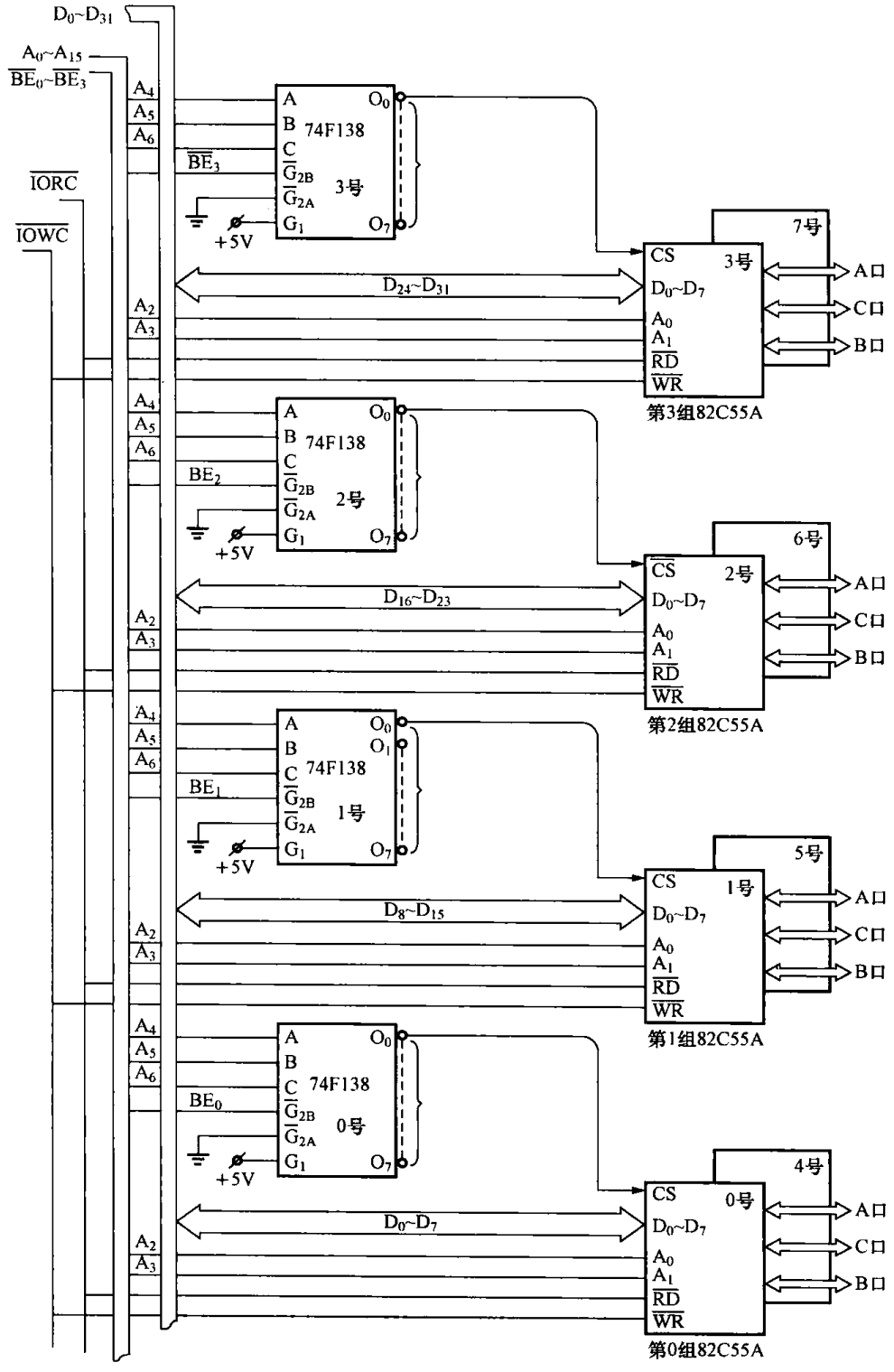


图 8.21 82C55A 32 位 I/O 端口与 80386DX 的连接

接口输入、输出，需要 4 个 82C55A 芯片。图中芯片号 0～3 组成一个 32 位组合的 I/O 端口。若要进一步扩展，再增加 4 个从芯片，4 个从芯片又可组成另一个 32 位组合的 I/O 端口。图中采用 4 个 74F138（功能同 74LS138）作为地址译码器分别用作第 0～3 组各芯片的片选信号。每个 82C55A 芯片均可以提供 A、B、C 三个端口，每个端口可设置成所需的工作方式。按照低地址存放组合数据的低字节原则，则字节选通信号 BE_0～BE_3 的连接应是：第 0 组的 82C55A 芯片的数据线与 CPU 的 D_0～D_7 数据线相连。依次类推，第 3 组的 82C55A 芯片的数据线与 CPU 的 D_{24}～D_{31} 数据线相连。

因为在一个总线周期内，80386DX CPU 经过数据总线可以传送字节、字和双字。所以，必须通知外部电路发生何种数据传送类型以及数据将通过数据总线的哪一部分进行传送。CPU 在执行 IN/OUT 指令时通过产生的字节选通输出信号（$\overline{BE_0}$～$\overline{BE_3}$）来实现。

例如，$\overline{BE_0}=0$ 对应于数据线 D_7～D_0。当需传送一个字节数据时，仅需一个 $\overline{BE_0}=0$ 即可。若要传送一个字数据，则应两个$\overline{BE}$输出低电平有效。若要传送双字数据，则 4 个$\overline{BE}$均应低电平有效。即：

（1）字节选通信号的编码$\overline{BE_3}$ $\overline{BE_2}$ $\overline{BE_1}$ $\overline{BE_0}=1110B$ 时，数据经数据线 D_7～D_0 传送，为字节传送。

（2）字节选通信号的编码$\overline{BE_3}$ $\overline{BE_2}$ $\overline{BE_1}$ $\overline{BE_0}=1100B$ 时，数据经数据线 D_{15}～D_0 传送，为字传送。

（3）字节选通信号的编码$\overline{BE_3}$ $\overline{BE_2}$ $\overline{BE_1}$ $\overline{BE_0}=0011B$ 时，数据经数据线 D_{31}～D_{16} 传送，为字传送。

（4）字节选通信号的编码$\overline{BE_3}$ $\overline{BE_2}$ $\overline{BE_1}$ $\overline{BE_0}=0000B$ 时，数据经数据线 D_{31}～D_0 传送，为双字传送。

按图 8.21 所示的连接方式两个 32 位组合 I/O 端口的 16 位寻址信号如下：

		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	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
	A 口：	×	×	×	×	×	×	×	×	×	0	0	0	0	0	×	×	0	0	0	0
0～3 号 82C55A	B 口：	×	×	×	×	×	×	×	×	×	0	0	0	0	1	×	×	0	0	0	0
32 位 I/O 口	C 口：	×	×	×	×	×	×	×	×	×	0	0	0	1	0	×	×	0	0	0	0
控制寄存器：		×	×	×	×	×	×	×	×	×	0	0	0	1	1	×	×	0	0	0	0
	A 口：	×	×	×	×	×	×	×	×	×	0	0	1	0	0	×	×	0	0	0	0
4～7 号 82C55A	B 口：	×	×	×	×	×	×	×	×	×	0	0	1	0	1	×	×	0	0	0	0
32 位 I/O 口	C 口：	×	×	×	×	×	×	×	×	×	0	0	1	1	0	×	×	0	0	0	0
控制寄存器：		×	×	×	×	×	×	×	×	×	0	0	1	1	1	×	×	0	0	0	0

由于本例中 A_{15}～A_8 均为无关地址位，A_1A_0 一般取 00，故各端口的地址号均可选用 8 位地址号，采用直接寻址方式进行输入/输出操作。CPU 根据 IN/OUT 指令规定的 8/16/32 位操作数类型，会自动产生相应的字节选通信号$\overline{BE_3}$～$\overline{BE_0}$，完成字节、字、双字的输入/输出操作。

习　　题

8.1　简述 8255A 的工作方式 0、方式 1、方式 2 的特点。

8.2 8255A 工作方式控制字的功能是什么?

8.3 8255A 工作在方式 1 输入输出，$\overline{STB}$、$\overline{ACK}$信号的功能是什么?

8.4 8255A 工作在方式 1 输入，用查询方式与 CPU 交换信息，CPU 应查询 8255A 的什么信号? 查询$\overline{STB}$可以吗? 为什么?

8.5 设 8255A 端口 A 工作在双向方式，允许输入中断，禁止输出中断，B 口工作在方式 0 输出，C 口剩余数据线全部输入，请初始化编程。设 8255A 端口地址为 60H、62H、64H、66H。

8.6 8255A 的 A 口与共阴极的 LED 显示器相连，若片选信号 A10～A3 =11000100，问 8255A 的端口地址是多少? A 口应工作在什么方式? 画出 8255A、74LS138、8086CPU 微机总线接口图，写出 8255A 的初始化程序。

8.7 以一个 5×5 键开关矩阵为例，用 8255A 的 A、B 口对矩阵进行扫描，请:

(1) 画出硬件连接图;

(2) 根据你的设计，对 8255A 进行初始化编程;

(3) 编一段程序实现一次完整的扫描。

第9章 微机系统串行通信及接口

9.1 串 行 通 信

我们知道并行通信时，数据各位同时传送。而串行通信时，数据和控制信息是一位接一位串行地传送出去。串行通信使用的传输线少，硬件电路相应简单，但传输速率较低，主要适用于长距离、低速率的通信中。

9.1.1 串行通信类型

串行通信的数据是逐位传送的，那么如何区分前后两个数据呢？为此串行通信对数据传送格式进行了规定，称为通信协议或规程。常用的串行通信协议有异步协议（Asynchronous communication protocol）和同步协议（Synchronous communication protocol）两种。其中，同步协议又分为面向字符（Character-Oriented）、面向比特（Bit-Oriented）和面向字节计数（Count Byte-Oriented）等方式。在此，面向字节计数的同步协议不作讨论。

1. 异步串行通信

异步串行通信的基本特点是：异步串行通信是以字符为信息单位进行数据传送的。每个字符作为一个独立的信息单位（1帧数据），可以随机出现在数据流中。即发送端发出的每个字符在数据流中出现的时间是任意的，接收端预先并不知道。这就是说，异步通信的"异步"主要体现在字符与字符之间通信没有严格的定时要求。然而，一旦传送开始，收/发双方则以预先约定的传输速率，在时钟的作用下，传送这个字符中的每一位，即要求位与位之间有严格而精确的定时，也就是说，异步通信在传送同一个字符的每一位时，是同步的。所以，所谓异步通信是指字符与字符之间的传送是异步的，而字符内部位与位之间传送是同步的。因此，异步串行通信一般用在数据传送时间不能确知、发送数据不连续、数据量较少和数据传输速率较低的场合。

异步通信时，被传送的字符出现在数据流中的相对时间是任意的、随机的。为了确保异步通信的正确性，必须使收/发双方在随机传送的字符与字符之间实现同步，即在字符数据格式中设置起始位和停止位，发送端在一个字符正式发送之前先发一个起始位，而在该字符结束时再发一个（或几个）停止位。接收端在检测到起始位时，便知道字符已到达，应开始接收字符；当检测到停止位时，则知道字符已结束。

异步通信帧数据格式如图9.1所示。每帧信息（每个字符）包括起始位、数据位、奇偶校验位、停止位四部分。其中各位的意义如下：

（1）起始位：位数为1，低电平，逻辑值0，表示开始传送字符。

（2）数据位：紧接着起始位之后是数据位。数据位的位数可以是5～8位，是要传送的有效信息。通常采用ASCII码，从最低位开始传送，由时钟进行定位。

（3）奇偶校验位：在数据位后，一般为1位（也可以没有校验位）。数据位加上这一位后，使得"1"的位数应为偶数（偶校验）或奇数（奇校验），以此来校验数据传送的正确性。

（4）停止位：标志一个字符的结束，表示一个字符传送完毕。可以是1位、1.5位、2位的高电平。

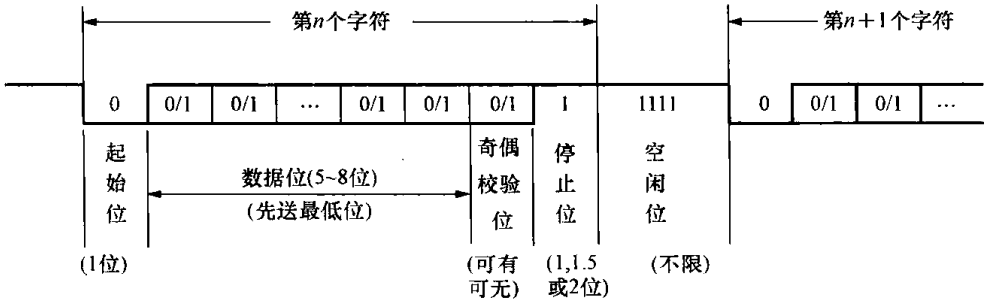


图 9.1 异步通信帧数据格式

(5) 空闲位：长度不定的高电平，逻辑值 1，表示当前线路上没有数据传送，保证起始位开始处一定有 1 个下跳沿。

异步通信是按字符传输的，接收设备在收到起始信号之后只要在一个字符的传输时间内能和发送设备保持同步就能正确接收。下一个字符起始位的到来又使同步重新校准。各字符之间允许有间隙，而且两个字符之间的间隔是不固定的。

为了在接收字符时发送和接收双方保持同步，异步通信要求发送和接收双方选定同样的波特率和波特率系数，在以起始位开始、停止位结束的一个字符内按约定的频率进行同步发送和接收。

所谓波特率，是指每秒钟传送的二进制位数，它是衡量数据传送速率的指标。例如数据传送速率为 120 字符/s，而每一个字符为 10 位，则其传送的波特率为 10×120=1200 (b/s)。波特率系数是预先规定的接收时钟的频率，是波特率的倍数，例如波特率系数为 16、32 或 64 倍等。

国际上规定了一个标准波特率系列，标准波特率也是最常用的波特率，标准波特率系列为 110、300、600、1200、4800、9600 和 19 200 等。

2. 同步串行通信

同步串行通信的基本特点是：同步串行通信是以数据块（字符块）为信息单位传送，而每帧信息包括成百上千个字符，因此，传送一旦开始，要求每帧信息内部的每一位都要同步。所谓同步通信是指字符内部位与位之间传送是同步的，字符与字符之间传送也是同步的。显然，这种通信方式对时钟同步要求非常严格，为此，收/发两端必须使用同一时钟来控制数据块传输中字符与字符、字符内部位与位之间的定时。主要用在要求快速、连续传输大批量数据的场合。

(1) 面向字符的同步串行通信。面向字符的同步协议是最早提出的同步协议，其典型代表是 IBM 的二进制同步通信协议 BSC (Binary Synchronous Communication)，随后 ANSI 和 ISO 都提出了类似的相应标准。它的特点是一次传送由若干个字符组成的数据块，而不是只传送一个字符，并规定了 10 个特殊字符作为这个数据块的开头与结束标志以及整个传输过程的控制信息，它们也叫做通信控制字。由于被传送的数据块是由字符组成，故被称作面向字符的数据格式。一帧数据格式如图 9.2 所示。

SYN	SYN	SOH	标题	STX	数据块	ETB/ETX	块校验

图 9.2 面向字符同步通信数据的帧格式

由图 9.2 可以看出，数据块的前、后都加了几个特定字符。SYN 是同步字符（Synchronous Character），每一帧开始处都有 SYN，加一个 SYN 称为单同步，加两个 SYN 称为双同步。设置同步字符是起联络作用，传送数据时，接收端不断检测，一旦出现同步字符，就知道是一帧开始了。接着的 SOH 是序始字符（Start of Header），它表示标题的开始。标题中包括源地址、目标地址和路由指示等信息。STX 是文始字符（Start of Text），它标志着传送的正文（数据块）开始。数据块就是被传送的正文内容，由多个字符组成。数据块后面是组终字符 ETB（End of Transmission Block）或文终字符 EXT（End of Text）。其中 ETB 用在正文很长，需要分成若干个分数据块，分别在不同帧中发送的场合，这时在每个分数据块后面用组终字符 ETB，而在最后一个分数据块后面用文终字符 ETX。一帧的最后是校验码，它对从 SOH 开始直到 ETX（或 ETB）字段进行校验，校验方式可以是奇偶校验或 16 位循环冗余校验码 CRC（Cyclic Redundancy Code）。

（2）面向比特的同步通信数据格式。面向比特（bit）型协议是以二进制位作为信息单位。现代计算机网络大多采用此类规程。最典型的是 IBM 的 SDLC（Synchronous Data Link Control，同步数据链路控制规程）和国际标准化组织 ISO 的 HDLC（High-Level Data Link Control，高级数据链路控制）通信规程。它们的特点是所传输的一帧数据可以是任意位，而且它是靠约定的位组合模式，而不是靠特定字符来标志帧的开始和结束，故称“面向比特”的协议。这种通信的数据格式如图 9.3 所示。

8位	8位	8位	≥0位	16位	8位
01111110	A	C	I	FC	01111110
开始标志	地址场	控制场	信息场	校验场	结束标志

图 9.3　面向比特同步通信数据的帧格式

由图 9.3 可见，面向比特型协议的一帧信息包括以下几个场（Field），所有场都从最低有效位开始传送。

开始标志 01111110：称标志场。SLDC/HDLC 协议规定，所有信息传输必须以一个标志字符开始，且以同一个标志字符结束，这个字符即 01111110。从开始标志到结束标志之间构成一个完整的信息单位，称为帧。所有的信息是以帧的形式传输的，而标志字符提供了每一帧的边界。接收端可通过搜索标志字符来探知帧的开始和结束，以此建立帧同步。

地址场和控制场：在标志字符之后，可以有一个地址场 A（Address）和一个控制场 C（Control）。地址场用来规定与之通信的次站的地址，控制场规定若干个命令。SLDC 规定 A 场和 C 场的宽度为 8 位，HDLC 则允许 A 场为任意长度，C 场为 8 位或 16 位。接收方必须检查每个地址字节的第一位，如果为“0”，则后边跟着另一个地址字节；若为“1”，则该字节就是最后一个地址字节。同理，如果控制场第一个字节的第一位为“0”，则还有第二个控制场字节，否则就只有一个字节。

跟在控制场之后的是信息场 I（Infomation），I 场包含有要传送的数据，并不是每一帧都必须有信息场。即数据场可以为 0，当它为 0 时，则这一帧主要是控制命令。

紧跟在信息场之后的是两字节的帧校验场，帧校验场称为 FC（Frame Check）场或帧校验序列 FCS（Frame check Sequence）。SDLC/HDLC 均采用校验码 CRC。除了标志场和自动插入的“0”位外，所有的信息都参加 CRC 计算。

9.1.2 串行数据传输方式

1. 信号的调制与解调

计算机的通信是要求传送数字信号，而在进行远程数据通信时，通信线路往往是借用现存的公用电话网，但是，电话网是为 300～3400Hz 间的音频模拟信号设计的，这对二进制数据的传输不适合。为此，在发送时需要将二进制信号调制成相应的音频信号，以适合在电话网上传输。在接收时，需要对音频信号进行解调还原成数字信号。

采用调制器（Modulator）把数字信号转换为模拟信号，送到通信链路上去，而用解调器（Demodulator）再把从通信链路上收到的模拟信号转换成数字信号。大多数情况下，通信是双向的，调制器和解调器合在一个装置中，这就是调制解调器 MODEM，如图 9.4 所示。可见调制器和解调器是在利用电话网进行远距离数据通信时所需的设备，因此把它叫做数据通信设备 DCE 或数传机。

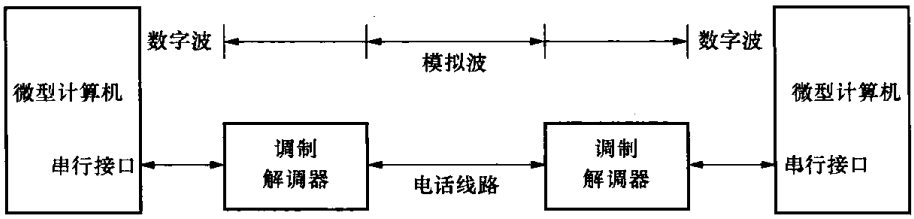


图 9.4 信号的调制与解调

调制解调器的类型很多，有幅移键控、频移键控和相移键控。当波特率小于 300 时，一般采用频移键控调制方式。

（1）幅移键控 ASK（Amplitude Shift Keying）。就是把数据信号的‘1’和‘0’变为载频幅度的变化，而在接收端根据载频幅度的变化恢复出原来的信号‘1’和‘0’。

（2）频移键控 FSK（Frequency Shift Keying）。就是把数据信号的‘1’和‘0’调制成不同的频率，接收端根据频率恢复出原来的数据信号。

（3）相移键控 PSK（Phase Shift Keying）。就是把数据信号的‘1’和‘0’变为载频相位的变化，相移键控可分为绝对相移和相对相移。绝对相移信号的特点是若‘1’信号的相位为 0 相，则‘0’信号的相位为 π 相，当然也可以相反。相对相移信号的相位是根据前一位信号的相位和本位信号的符号来决定。接收端解调器根据信号相位的不同或相位的改变情况而恢复出原来的数据信号。

数据进行高速传输时，经常用多元调制方式，如多电平调幅（MASK）、多元调频（MFSK）及多元调相（MPSK）；或者进行混合调制，如既调幅又调相等。

2. 串行通信的数据传送方式

串行通信是将数据一位接一位地顺序通过同一信号线进行传送的方式。它的通信线路可以只有一条，此时发送信息和接收信息不能同时进行，而采用分时占用通信线路的方法，如 A 发送信息，B 接收，而当 B 发送信息时，A 接收。根据数据传送方向的不同，串行通信的数据传送可分为单工、半双工和全双工三种工作方式。

（1）单工方式（Simplex）。只允许数据按照一个固定的方向传送，即一方只能作为发送站，另一方只能作为接收站。如图 9.5（a）所示。

（2）半双工方式（Half-Duplex）。数据能从 A 站传送到 B 站，也能从 B 站传送到 A 站，

但是不能同时在两个方向上传送，每次只能有一个站发送，另一个站接收。通信双方可以轮流地进行发送和接收，如图 9.5（b）所示。

（3）全双工方式（Full Duplex）。允许通信双方同时进行发送和接收。这时，A 站在发送的同时也可以接收，B 站亦同。全双工方式相当于把两个方向相反的单工方式组合在一起，因此它需要两条传输线，如图 9.5（c）所示。

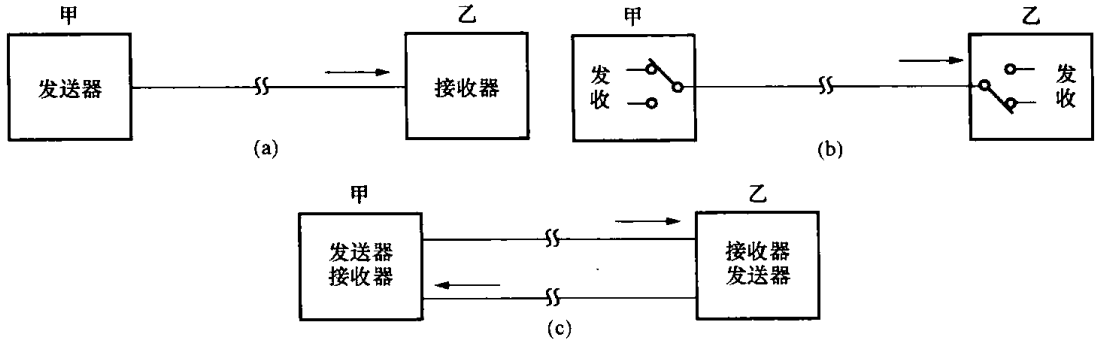


图 9.5　串行通信工作方式

（a）单工；（b）半双工；（c）全双工

在计算机串行通信中主要使用半双工和全双工方式。

9.1.3　串行接口标准 EIA RS-232-C

RS-232-C 是使用最早、应用最多的一种异步串行通信总线，它是美国电子工业协会（Electronic Industries Association，EIA）1962 年公布的，1969 年最后一次修订而成。其中 RS 是 Recommended Stander 的缩写，232 是该标准的标志，C 表示修订版本。

RS-232-C 主要用来定义计算机系统的数据终端设备（DTE）和数据通信设备（DCE）之间接口的电气特性。如 CRT、打印机与 CPU 的通信大都采用 RS-232-C 总线。由于诸多嵌入式处理机本身具有异步串行通信接口，因此，应用中使用 RS-232-C 串行总线极为方便。

1. RS-232-C 传递信息的格式标准

RS-232-C 采用按位串行通信方式。该标准对所传递的信息规定如下：信息的开始为起始位；信息结尾为停止位，它可以是 1 位，1.5 位或 2 位；信息本身可以是 5、6、7、8 位；再加 1 位奇偶校验位。如果两个信息间无信息，则应写“1”，表示空闲位。

RS-232-C 传送的波特率规定为 56 000、28 000、115 200、19 200、9600、4800、2400、1200、600、300、150 等。RS-232-C 总线的传送距离一般都不超过 15m。

2. RS-232-C 电气特性

由于 RS-232-C 是在 TTL 电路出现之前研制的，所以它的电平不是＋5V 和地，它使用负逻辑，其低电平“0”在＋5～＋15V 之间，高电平“1”在－5～－15V 之间。有专门的集成电路，以便进行电平转换，如 MC1488 和 MC1489 就是专门用在计算机（或终端）与 RS-232-C 标准进行电平转换的接口芯片。

3. RS-232-C 机械特性及引脚功能

RS-232-C 标准接口是 25 针（空）的 D 型连接器。25 针就是有 25 条信号线，其中 4 条数据线、11 条控制线、3 条定时线、7 条备用和未定义线。其信号定义及功能如表 9.1 所示。

表 9.1 RS-232-C 接口信号定义及功能

脚号	信 号 名	缩写名	方向与功能说明
1	保护地	PG	无方向设备地
2	发送数据 *	TxD	DTE→DCE 终端发送串行数据
3	接收数据 *	RxD	DTE←DCE 终端接收串行数据
4	请求发送 *	RTS	DTE→DCE 终端请求通信设备切换到发送方向
5	清除发送 *	CTS	DTE←DCE 通信设备已切换到发送方向
6	线路设备准备好 *	DSR	DTE←DCE 通信设备就绪，设备可用
7	信号地 *	SG	无方向信号地，所有信号公共地
8	载波检出 *	DCD	DTE←DCE 通信设备正在接收通信链路的信号
	（接收线信号检出）	(RLSD)	
9	保留		
10	保留		
11	保留		
12	辅信道接收信号检测		
13	辅信道清除发送		
14	辅信道发送数据		
15	发送定时时钟（DCE 源）		
16	辅信道接收数据		
17	接收器定时时钟		
18	保留		
19	辅信道请求发送		
20	数据终端准备好 *	DTR	DTE→DCE 终端设备就绪，设备可用
21	信号质量测定器		
22	振铃指示器 *	RI	DTE←DCE 通信设备通知终端，通信链路有振铃
23	数据信号速率选择器 DTE 源/DCE 源		
24	发送器定时时钟（DTE 源）		
25	保留		

由表 9.1 可以看出，RS-232C 标准为主信道和辅信道共分配了 25 根线，其中辅信道的信号线几乎没有使用，而主信道的信号线有 9 根（表中打 * 号），它们才是远距离串行通信接口标准中的基本信号线。它可分为两类：一类是在 DTE 与 DCE 之间交换信息的信号；另一类是为了正确无误地传输上述信息而设计的联络信号。各信号线功能如下：

（1）传送信息信号：

1）发送数据 TxD（Transmitted Data）：通过 TxD 线将串行数据发送到 MODEM。

2）接收数据 RxD（Received Data）：通过 RxD 线接收从 MODEM 发来的数据。

（2）联络信号，这类信号共有 6 个：

1）请求传送 RTS（Request to Send）：用来表示 DTE 请求 DCE 发送数据，即当终端要发送数据时，使该信号有效，向 MODEM 请求发送。

2）清除发送 CTS（Clear to Send）：用来表示 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。当 MODEM 已准备好接收终端发来的数据时，使该信号有效，通知终端开始沿发送数据线 TxD 发送数据。

这对 RTS/CTS 请求应答联络信号用于半双工采用 MODEM 的系统中，作为发送方式和接收方式之间的切换控制信号。在全双工系统中，因配置双向通道，故不需 RTS/CTS 联络信号，使其变高。

3）数据准备就绪 DSR（Data Set Ready）：有效时表明 MODEM 处于可以使用状态。

4）数据终端就绪信号 DTR（Data Terminal Ready）：有效时表明数据终端可以使用。

DST 和 DTR 这两个信号有时连到电源上，一上电就立即有效。目前有些 RS-232C 接口甚至省去了用以指示设备是否准备好的这类信号，认为设备始终是准备好的。可见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了。

5）数据载波检测信号 DCD（Data Carrier Detect）：用来表示 DCE 已接通通信链路，告之 DTE 准备接收数据。当本地的 MODEM 收到由通信链路另一端的 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量数据后，沿接收数据线 RxD 送到终端。

6）振铃指示信号 RI（Ring Indication）：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。

（3）地线。

有两根线 SG、PG：信号地和保护地信号线，无方向。

上述控制信号线何时有效，何时无效的顺序表示了接口信号的传送过程。例如，只有当 DSR 和 DTR 都处于有效状态时，才能在 DTE 和 DCE 之间进行传送操作。若 DTE 要发送数据，则预先将 RTS 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。这种顺序的规定对半双工的通信线路特别有用，因为半双工的通信线路进行双向传送时，有一个换向问题，只有当收到 DCE 的 CTS 线为有效状态后，才能确定 DCE 已由接收方向改为发送方向了，这时线路才能开始发送。

4. RS-232-C 的应用

RS-232-C 接口标准中，包括了主辅两条信道，辅信道用来传送一些辅助控制信息，传输速率较低，在一般串行通信中很少使用。即使是主信道，也不是所有的信号线都一定要用，最常用是其中几条基本信号线。根据具体应用场合不同，有下面几种连接方式：

（1）使用 MODEM 连接。计算机通过 MODEM 或其他数据通信设备（DCE）使用一条电话线进行通信时，一般只需要其中的 8 条线（不包括两根地线），如图 9.6 所示。

在图 9.6 中，计算机终端（DTE）向远程终端（DCE）发送数据的过程如下：首先 DTE 向本地 DCE（MODEM）发出 DTR=1 和 RTS=1 的信号，表示 DTE 请求发送数据，即为本地和远程 DCE 之间建立通道开了绿灯，一旦通道建立好了，DCE 发回信号 DSR=1。当 DCE 做好发送数据准备后，又向 DTE 发回信号 CTS=1。只有当 DTE 收到本地 DCE 发回肯定的 DSR 和 CTS 信号后，DTE 才能由 TxD 线向 DCE 发送数据。因此，RTS、DTR、DSR 和 CTS 四个信号同时为 1 是 TxD 发送数据的条件。

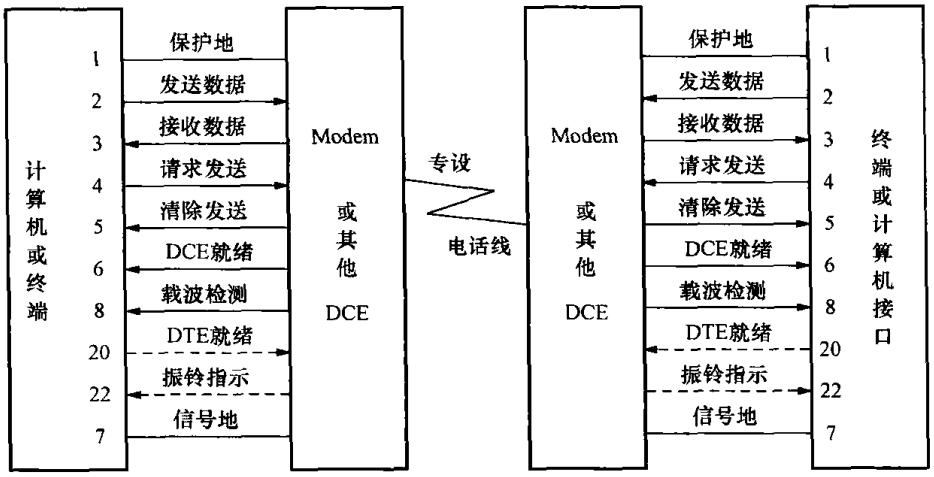


图 9.6 使用 MODEM 时 RS-232-C 引脚的连接

当接收数据时，DTE 先向本地 DCE 发出 DTR＝1 信号，表示本地和远程 DCE 之间可以建立通道。一旦通道建立好，DCE 向 DTE 发出 DSR＝1 信号。这时，数据就可以通过 RxD 线传到 DTE。因此，RxD 信号产生的条件是 DTR 和 DSR 两个信号同时为 1。这只是 RxD 信号的产生条件，至于 RxD 线上是否有信号，取决于远程 DCE 是否发送数据。

(2) 直接连接。当计算机和终端之间不使用 MODEM 或其他通信设备（DCE）而直接通过 RS-232-C 接口连接时，一般只需要 5 根线（不包括抗保护地线以及本地 4、5 之间的连线），但其中多数采用反馈与交叉结合的连接法，如图 9.7 所示。

在图 9.7 中，2↔3 交叉线为最基本的连线，以保证 DTE 和 DCE 间能正常地进行全双工通信。20↔6 也是交叉线，用于两端的通信联络，使两端能相互检测出对方“数据已就绪”的状态。4→5 为反馈线，使传送请求总是被允许的。由于是全双工通信，这根反馈线意味着任何时候都可以双向传送数据，用不着再去发“请求发送”（RTS）信号。这种没有 MODEM 的串行通信方式，一般只用于近程通信（不超过 15m）。

(3) 最简单的 RS-232-C 连线方式。只需 2↔3 交叉连接线以及信号地线，而将各自的 RTS 和 DTR 分别接到自己的 CTS 和 DSR 端，如图 9.8 所示。

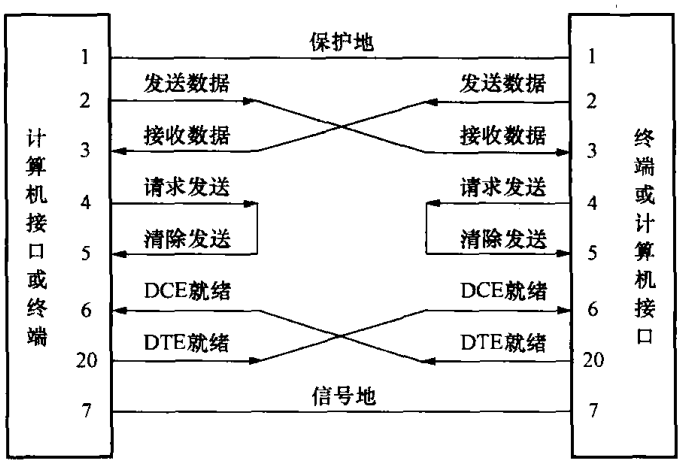


图 9.7 使用 RS-232-C 的直接连接法

在图 9.8 (a) 中，只要一方使自己的 RTS 和 DTR 为 1，那么它的 CTS、DSR 也就是 1，从而进入了发送和接收的就绪状态，这种接法常用于一方是主动设备，而另一方为被动设备的通信中，如计算机与打印机或绘图仪之间的通信。这样，被动的一方 RTS 与 DTR 常置 1，同时 CTS、DSR 也常置 1，从而使其长期处于接收就绪状态。只要主动一方令线路就绪（DTR＝1），并发出发送请求（RST＝1），即可立即向被动的一方传送信息。

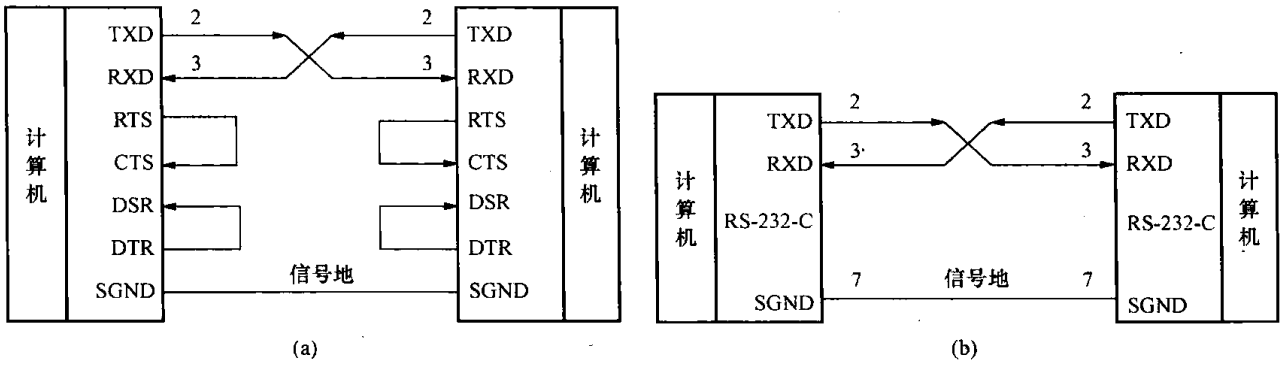


图 9.8　最简单的 RS-232-C 连接方式

图 9.8（b）所示为更简单的连接方法，如果说图（a）所示的连接方法在软件设计上还需要检测“清除发送”（CTS）和“数据设备就绪”（DSR）的话，那么图（b）所示的连接方法则完全不需要检测上述信号，随时都可发送和接收。这种连接方法无论在软件和硬件上，都是最简单的一种方法。

9.2　可编程串行接口 8251A

9.2.1　8251A 的主要功能

8251A 是可编程串行通信接口芯片，它的主要功能有：

（1）可以工作在同步方式也可以工作在异步方式。

（2）同步方式下：5～8 位/字符；可以用外部同步，也可以内部自动检测同步字符而实现内同步；可以自动插入同步字符。

（3）异步方式下：5～8 位/字符；时钟速率为波特率的 1、16、64 倍；能选择 1 位、1.5 位或 2 位停止位；可检查假启动位；可产生中止字符；也能自动检测和处理中止字符。

（4）波特率：同步方式下，0～64K，异步方式下，0～19.2K。

（5）完全双工工作，具有双缓冲的发送器和接收器。

（6）具备奇偶校验、溢出和帧错误检测等功能电路。

（7）全部输入输出电路是 TTL 电平。

由于 8251A 的接口电平和 EIA RS-232-C 规定的电平不同，所以，在和 RS-232-C 接口连接时，必须转换电平。

9.2.2　8251A 芯片引脚

8251A 是用来作为 CPU 与外设或调制解调器之间的接口芯片，如图 9.9 所示。它的信号线可以分为 8251A 与 CPU 连接的信号线、与外设（或调制解调器）连接的信号线和时钟信号线。

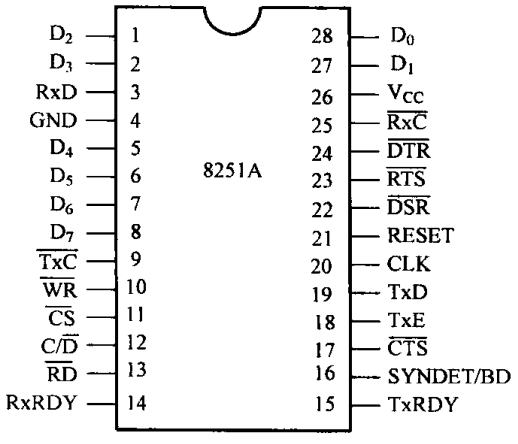


图 9.9　8251A 芯片引脚图

1. 8251A 和 CPU 的连接信号

（1）数据信号。D_0～D_7 是 8251A 的 8 根双向数据线，通过它们和系统数据总线相连，传输数据、CPU 对 8251A 的编程命令以及 8251A 的状态信息。

(2) 片选信号。

$\overline{CS}$：CPU 的外设地址经译码以后提供给 8251A 的片选信号$\overline{CS}$。如果$\overline{CS}$为低电平，该 8251A 芯片被选中，才能对该 8251A 进行读或写操作；反之，该 8251A 的数据总线处于高阻三态，读信号$\overline{RD}$和写信号$\overline{WR}$对该 8251A 芯片均不起作用。

(3) 读/写控制信号。

1) $\overline{RD}$：读信号$\overline{RD}$为低电平时，表示 CPU 正在从该 8251A 读取数据或状态信息。

2) $\overline{WR}$：写信号$\overline{WR}$为低电平时，表示 CPU 正在从该 8251A 写数据或控制字。

3) C/$\overline{D}$：控制/数据信号，用来区分当前读写的是数据还是控制信息。该信号也可看作是 8251A 数据口/控制口的选择信号。

由此可知，$\overline{RD}$、$\overline{WR}$、C/$\overline{D}$这 3 个信号的组合，决定了 8251A 的具体操作，它们的关系如表 9.2 所示。

表 9.2　　C/$\overline{D}$与$\overline{RD}$和$\overline{WR}$操作

C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	操　作
0	0	1	0	CPU 从 8251A 读取数据
0	1	0	0	CPU 往 8251A 写入数据
1	0	1	0	CPU 读取 8251A 的状态
1	1	0	0	CPU 往 8251A 写入控制命令
×	×	×	1	8251A 高阻，与 CPU 隔离

(4) 收发联络信号。

1) TxRDY (Transmitter Ready)：发送器准备好信号，高电平有效。当它有效时，表示发送器已经准备好接收 CPU 送来的数据字符，通知 CPU 可以向 8251A 发送数据。CPU 向 8251A 写入了一个字符以后，TxRDY 自动复位。在用查询方式时，此信号作为一个状态位，CPU 可从状态寄存器的 D_0 位检测这个信号；在用中断方式时，此信号作为中断请求信号。

2) RxRDY (Receiver Ready)：接收器准备好信号，高电平有效。当它有效时，表示当前 8251A 已经从外设或者调制解调器接收到一个字符，通知 CPU 来读取。当 CPU 从 8251A 读取数据后，RxRDY 则变为低电平。在查询方式时，此信号可作为状态位，CPU 通过读状态寄存器的 D_1 位检测这个信号。在中断方式时，可作为中断请求信号。

3) TxE (Transmitter Empty)：发送器空信号，高电平有效。当它有效时，指示发送器中的数据已发送出去。当 8251A 从 CPU 接收待发的字符后，自动复位。TxE 既然表示发送已经结束，这样在半双工方式中，CPU 从它就可以知道何时切换数据的传送方向，由发送转为接收。此信号可从状态寄存器的 D_2 位检测到。

4) SYNDET (Synchronous Detection) /BD (Break Detection)：双功能引脚。这个引脚在同步方式时，作同步字符检出信号，为双向线。SYNDET 是作为输入还是输出，取决于初始化程序指定 8251A 是工作于外同步还是外同步。当 8251A 工作于内同步方式时，SYNDET 是输出。当 8251A 工作于外同步方式时，SYNDET 是输入。这个引脚在异步方式时，作间断信号检出 BD，是输出。当检测到间断码时，输出高电平。

2. 8251A 和外设的连接信号

8251A 和外设的连接信号分为数据信号和收发联络信号两类。

（1）数据信号。

1）TxD：发送数据线。CPU 送往 8251A 的并行数据，经移位寄存器变成串行数据，通过 TxD 发送给外设。

2）RxD：接收数据线。它接收外设送往 8251A 的串行数据，经转换后成为并行数据，供 CPU 读取。

（2）收发联络信号。

1）$\overline{\text{DTR}}$：数据终端准备好，输出信号，低电平有效。通知外设（或调制解调器），CPU 当前已经准备就绪。它由工作命令字的 D_1 置“1”变为有效。

2）$\overline{\text{DSR}}$：数据设备准备好，输入信号，低电平有效。当它有效时，表示外设（或调制解调器）已准备好。CPU 通过读状态寄存器的 D_7 位来检测这个信号。

3）$\overline{\text{RTS}}$：请求发送信号，输出，低电平有效。用来通知外设（或调制解调器）CPU 已经准备好发送数据。由命令字的 D_5 置“1”使其有效。

4）$\overline{\text{CTS}}$：允许发送信号，输入，低电平有效。这是对$\overline{\text{RTS}}$的响应信号，由外设（或调制解调器）送往 8251A。当$\overline{\text{CTS}}$有效时，8251A 才能发送数据。

3. 时钟信号

（1）$\overline{\text{TxC}}$（Transmitter Clock）：发送器时钟，由外部（波特率时钟发生器）提供。由它控制 8251A 发送数据的速率。在异步方式下，$\overline{\text{TxC}}$的频率可以等于波特率，也可以是波特率的 16 倍或 64 倍。在同步方式下，$\overline{\text{TxC}}$的频率与数据速率相同。

（2）$\overline{\text{RxC}}$（Receiver Clock）：接收器时钟，由外部（波特率时钟发生器）提供。其频率的选择和$\overline{\text{TxC}}$相同。在实际应用中，将$\overline{\text{TxC}}$和$\overline{\text{RxC}}$连接在一起，使用同一个时钟源——波特率时钟发生器。

（3）CLK：工作时钟，由外部时钟源提供。为芯片内部电路提供定时，并非发送或接收数据时钟。在同步方式下，CLK 的频率要大于接收器或发送器输入时钟（$\overline{\text{TxC}}$或$\overline{\text{RxC}}$）频率的 30 倍。在异步方式下，CLK 的频率要大于接收器或发送器输入时钟频率的 4～5 倍。

9.2.3　8251A 的内部结构

8251A 的内部结构如图 9.10 所示，分成 5 个主要部分：数据总线缓冲器、读/写控制逻辑电路、调制解调控制电路、接收器和发送器。

（1）数据总线缓冲器是一个三态、双向的 8 位缓冲器，用来和系统数据总线相连。在 CPU 执行输入或输出指令时，通过 8251A 的数据总线缓冲器进行数据、控制字、命令字、状态信息的传送。

（2）读/写控制逻辑电路接收来自系统控制总线的信号，产生整个芯片的控制信号。

（3）调制解调控制电路有一组输入输出的控制信号，它们用来简化 8251A 和调制解调器的连接。在远程通信时，要把数字信号调制为模拟信号，再发送出去，而在接收端则要把模拟信号解调为数字信号，再由串行接口送往计算机。调制解调控制电路使得 8251A 与调制解调电路的连接非常方便。

（4）发送器由发送缓冲器和发送控制电路两部分组成。发送器的功能是，首先把待发送的并行数据转换成所要求的帧格式并加上校验位，然后在发送时钟$\overline{\text{TxC}}$的作用下，由 TxD 引脚一位接一位地串行发送出去。发送完一帧数据后，发送器准备好信号置位（TxRDY＝1），通知 CPU 发送下一个数据。

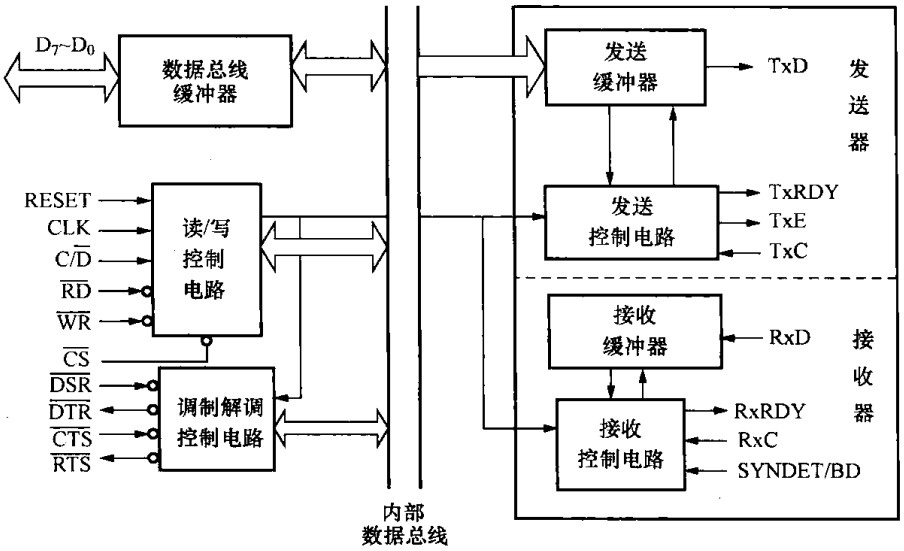


图 9.10 8251A 内部结构

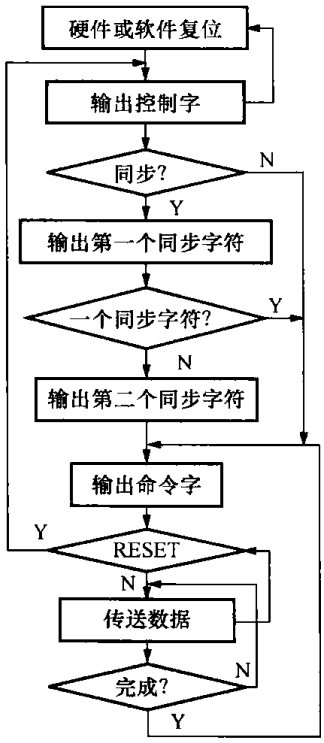


图 9.11 8251A 初始化流程

(5) 接收器由接收缓冲器和接收控制电路两部分组成。接收器的功能是在接收时钟$\overline{RxC}$作用下接收 RxD 引脚上的帧格式化串行数据并把它转换为并行数据。同时进行校验，若发现错误，则在状态寄存器中保存，以便 CPU 处理。当校验无误时，将并行数据存放在数据总线缓冲器中，并发出接收器准备好信号(RxRDY=1)，通知 CPU 读数据。

9.2.4 8251A 的编程命令

8251A 有多种通信方式，可通过编程来定义各种功能。8251A 的编程有两种，一是控制字的设置，二是命令字的设置。用控制字可实现选择是同步还是异步的工作方式，并按照其工作方式指定帧数据格式；命令字的作用是确定 8251A 的实际操作，迫使 8251A 进行某种操作或处于某种工作状态，以便接收或发送数据。

8251A 初始化编程总是从设置控制字开始，随后是命令字，如图 9.11 所示。控制字必须紧跟在复位之后设置。当硬件复位或通过软件复位后，便向控制字寄存器中写入控制字，设置 8251A 工作在异步或同步方式。如果是同步方式，必须指出同步字符的个数，并随后写入同步字符。

设置控制字之后，写入命令字。命令字包括 8251A 操作的各种控制命令，若 D_6 位为 1，即使 8251A 复位，则 8251A 恢复到初始化状态；否则，8251A 便开始数据传输。

1. 控制字

控制字的格式如图 9.12 所示。控制字为 8 位，可以分为 4 组，每组 2 位。

D_1D_0：用以确定是工作于同步方式还是异步方式。当 $D_1D_0=00$ 时为同步方式；当 $D_1D_0\neq00$ 时为异步方式，且 D_1D_0 的三种组合用以选择输入时时钟频率与波特率之间的比例系数。

D_3D_2：用以确定 1 个数据包含的位数。

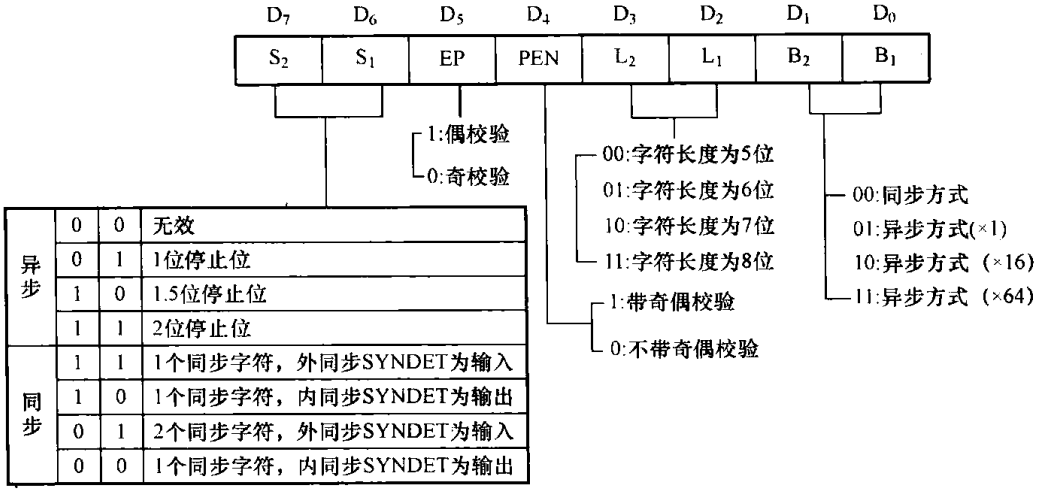


图 9.12　8251A 控制字的格式

D_5D_4：用以确定要不要校验以及是奇校验还是偶校验。

D_7D_6：在同步和异步方式的意义是不同的。异步时用以规定停止位的位数；同步时用以确定是内同步还是外同步，以及同步字符的个数。

2. 命令字

命令字的作用是确定 8251A 的实际操作，迫使 8251A 进行某种操作或处于某种工作状态，以便接收或发送数据。8251A 的命令字格式如图 9.13 所示。

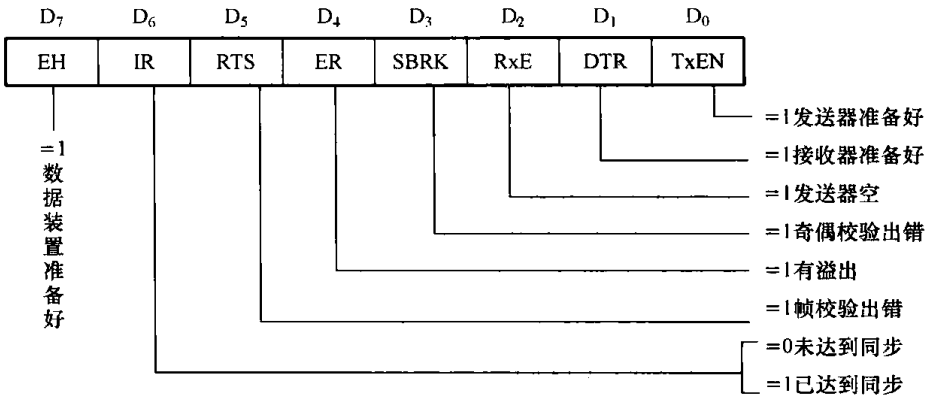


图 9.13　命令字格式

D_0：允许发送 TxEN（Transmit Enable）。$D_0=1$，允许发送；$D_0=0$，禁止发送。可作为发送中断屏蔽位。

D_1：数据终端准备就绪 DTR。$D_1=1$，设置$\overline{\text{DTR}}$有效，表示终端设备已准备好；$D_1=0$，置$\overline{\text{DTR}}$无效。

D_2：允许接收 RxE（Receive Enable）。$D_2=1$，允许接收；$D_2=0$，禁止接收。可作接收中断屏蔽位。

D_3：发中止字符 SBRK（Send Break Characer）。$D_3=1$，强迫 TxD 为低电平，输出连续的空号；$D_3=0$，正常操作。

D_4：错误标志复位 ER（Error Reset）。$D_4=1$，使状态字中的错误标志位（PE/OE/FE）复位；$D_4=0$，错误标志不复位。

D_5：请求发送 RTS（Request To Send）。$D_5=1$，强迫$\overline{RTS}$为低电平，置请求发送$\overline{RTS}$有效；$D_5=0$，置$\overline{RTS}$无效。

D_6：内部复位 IR（Internal Reset）。$D_6=1$，进行内部复位，使 8251A 回到等待控制字状态。发控制字之前，一定要先进行内部复位，即复位后一定要跟控制字。$D_6=0$，不进行内部复位。

D_7：进入搜索方式 EH（Enter Hunt Mode）。$D_7=1$，启动搜索同步字符；$D_7=0$，不搜索同步字符。

3. 状态字

8251A 执行命令进行数据传送后的状态信息放在状态字寄存器中，CPU 可以通过读入 8251A 的状态字，进行分析和判断，以决定下一步该怎么办。8251A 的状态字格式如图 9.14 所示（所有状态位是置“1”有效）。

有一点需要指出，状态字寄存器的状态位 RxRDY、TxE、SYNDET 以及 DSR 的定义与芯片引脚的定义相同，只有 TxRDY 的含义同 8251A 芯片引脚上的 TxRDY 的含义是不同的。状态寄存器的状态位 TxRDY，只要发送缓冲器一空就置位；而引脚 TxRDY 还要满足$\overline{CTS}$=0 和 TxE=1 时，即满足三个条件时才置位。

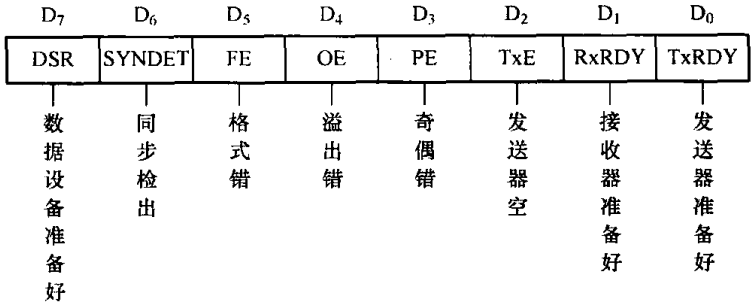


图 9.14　状态字格式

D_3～D_5 三位是错误状态信息。其中：

D_3：奇偶错 PE（Parity Error）。当奇偶错被接收端检测出来的时候，PE 置“1”。PE 有效并不禁止 8251A 工作，它由命令字中的 ER 位复位。

D_4：溢出错 OE（Overun Error）。若前一个字符尚未被 CPU 取走，后一个字符已变为有效，则 OE 置“1”。OE 有效并不禁止 8251A 的操作，但是被溢出的字符丢掉了，OE 被命令字的 ER 复位。

D_5：帧出错 FE（Frame Error）（只用于异步方式）。若接收端在任一字符的后面没有检测到规定的停止位，则 FE 置“1”。由命令字的 ER 复位，不影响 8251A 的操作。

9.2.5　8251A 初始化编程及应用举例

1. 异步方式下的初始化程序举例

[例 9.1]　设 8251A 工作在异步方式，波特率系数（因子）为 16，7 个数据位字符，偶校验，2 个停止位，发送、接收允许，设端口地址为 51H。完成初始化程序。

分析：根据题目要求，可以确定控制字为：11111010B，即 FAH；而命令字为：00110111B，即 37H。

解　初始化程序如下：

```
MOV    AL,   0FAH          ;设置控制字:异步方式,7 位/字符,偶校验,2 个停止位
OUT    51H,  AL
MOV    AL,   37H           ;设置命令字
OUT    51H,  AL
```

2. 同步方式下初始化程序举例

[例 9.2] 设端口地址为 51H，采用内同步方式，2 个同步字符（设同步字符为 16H），偶校验，7 位数据位/字符。

分析：根据题目要求，可以确定控制字为：00111000B，即 38H；命令字为：10010111B，即 97H。它使 8251A 对同步字符进行检索；同时使状态字寄存器中的 3 个出错标志复位；此外，使 8251A 的发送器启动，接收器也启动；命令字还通知 8251A，CPU 当前已经准备好进行数据传输。

解　具体程序段如下：

```
MOV    AL,  38H          ;设置控制字:同步方式,用 2 个同步字符,7 个数据位,偶校验
OUT    51H, AL
MOV    AL,  16H
OUT    51H, AL           ;送第一个同步字符
OUT    51H, AL           ;送第二个同步字符
MOV    AL,  97H          ;设置命令字,使发送器和接收器启动
OUT    51H, AL
```

3. 利用状态字进行编程的举例

[例 9.3] 对 8251A 进行初始化，使其工作于异步方式，波特率因子为 16，每个字符有 7 个数据位，2 个停止位，采用偶校验。然后对状态字进行测试，输入 80 个字符。

分析：设 8251A 的控制和状态端口地址为 51H，数据输入和输出端口地址为 50H。字符输入后，放在 BUFFER 标号所指的内存缓冲区中。

解　具体的程序段如下：

```
        MOV   AL,0FAH
        OUT   51H,AL            ;设置控制字
        MOV   AL,35H            ;设置命令字,使发送器和接收器启动,并清除出错指示位
        OUT   51H,AL
        MOV   DI,0              ;变址寄存器初始化
        MOV   CX,80             ;计数器初始化,共收取 80 个字符
BEGIN:  IN    AL,51H            ;读取状态字,测试 RxRDY 位是否为 1,
        TEST  AL,02H            ;如为 0,表示未收到字符,继续读取状态字并测试
        JZ    BEGIN
        IN    AL,50H            ;读取字符
        MOV   DX,OFFSET BUFFER
        MOV   [DX+DI],AL
        INC   DI                ;修改缓冲区指针
        IN    AL,51H            ;读取状态字
        TEST  AL,38H            ;测试有无帧校验错,奇/偶校验错和溢出错,如有,则转出错
        JZ    ERROR             ;处理程序;如没有错误,则再接收下一个字符
        LOOP  BEGIN
        JMP   EXIT              ;如输入满足 80 个字符,则结束
ERROR:  CALL  ERR-OUT           ;调出错处理程序
EXIT:
```

4. 两台微型计算机通过 8251A 相互通信的举例

[例 9.4] 通过 8251A 实现相距较远的两台微型计算机相互通信的系统连接简化框图如 9.15 所示。这时，利用两片 8251A 通过标准串行接口 RS-232-C 实现两台 8086 微机之间的串行通信，可采用异步或同步工作方式。设收发双方均工作于异步方式，每个字符包含 8 位数据，1 位停止位，偶校验，波特率系数为 64。

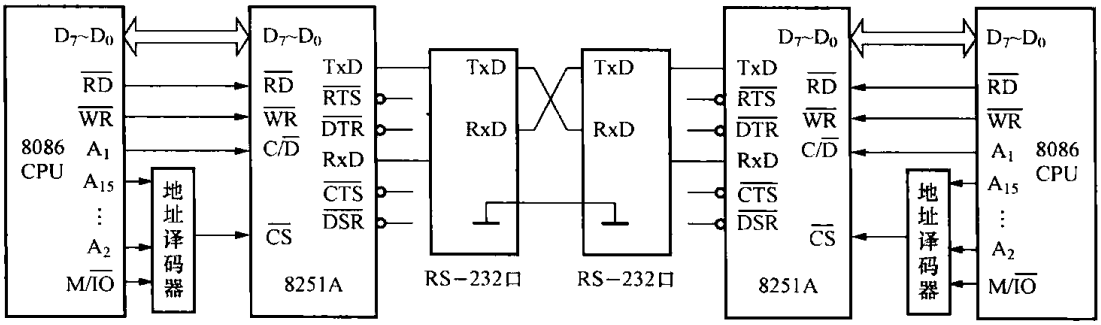


图 9.15 8251A 异步传送系统连接简化框图

分析：设系统采用查询方式控制传输过程，异步传送。

初始化程序由两部分组成：

(1) 将一方定义为发送器。发送端 CPU 每查询到 TXRDY 有效，则向 8251A 并行输出一个字节数据。

(2) 将另一方定义为接收器。接收端 CPU 每查询到 RXRDY 有效，则从 8251A 输入一个字节数据，一直将全部数据传送完毕为止。

解 发送端初始化程序与发送控制程序如下：

```
STT:  MOV  AL,7FH
      OUT  51H,AL              ;将 8251A 定义为异步方式,8 位数据,1 位停止位
      MOV  AL,11H              ;偶校验,取波特率系数为 64,允许发送
      OUT  51H,AL
      MOV  DI,发送数据块首地址   ;设置地址指针
      MOV  CX,发送数据块字节数   ;设置计数器初值
NEXT: MOV  DX,8251A 控制端口
      IN   AL,DX
      AND  AL,01H              ;查询 TXRDY 有效否?
      JZ   NEXT                ;无效则等待
      MOV  DX,8251A 数据端口
      MOV  AL,[DI]             ;向 8251A 输出一个字节数据
      OUT  DX,AL
      INC  DI                  ;修改地址指针
      LOOP NEXT                ;未传输完,则继续下一个
      HLT
```

接收端初始化程序和接收控制程序如下：

```
SRR:  MOV  DX,8251A 控制端口
```

```
        MOV  AL,7FH
        OUT  DX,AL                    ;初始化 8251A,异步方式,8 位数据,1 位
        MOV  AL, 14H                  ;停止位,偶校验,波特率系数 64,允许接收
        OUT  DX,AL
        MOV  DI,接收数据块首地址       ;设置地址指针
        MOV  CX,接收数据块字节数       ;设置计数器初值
COMT:   MOV  DX,8251A 控制端口
        IN   AL,DX
        ROR  AL,1                     ;查询 RXRDY 有效否?
        ROR  AL,1
        JNC  COMT                     ;无效则等待
        ROR  AL,1                     ;有效时,进一步查询是否有奇偶校验错
        JC   ERR                      ;有错时,转出错处理
        MOV  DX,8251A 数据端口
        IN   AL,DX                    ;无错时,输入一个字节到接收数据块
        MOV  [DI],AL
        INC  DI                       ;修改地址指针
        LOOP COMT                     ;未传输完,则继续下一个
        HLT
ERR:    CALL ERR - OUT
```

习　　题

9.1　串行通信的主要特点是什么?

9.2　串行通信有几种传输方式? 各自有哪些特点?

9.3　设异步传输时，每个字符对应 1 个起始位，7 个数据位，1 个奇校验和 1 个停止位。若波特率为 1200，则每秒钟传输的最大字符数为多少?

9.4　异步通信时采用什么方法使发送方和接收方的信息得到同步? 用图示说明异步通信协议一帧数据的格式。

9.5　RS-232-C 是哪两种设备之间的通信接口的标准? RS-232-C 专用的接口信号有哪些?

9.6　简述 8251A 的内部结构及各部分的作用。

9.7　8251A 在接收和发送数据时，分别通过哪个引脚向 CPU 发中断请求信号?

9.8　8251A 与外设之间有哪些接口信号?

9.9　说明 8251A 异步方式与同步方式初始化编程的主要区别。

9.10　设某系统中使用串行 I/O 接口 8251A 工作在异步方式、8 位字符、带偶校验、停止位长 1 位、波特率系数为 16、允许发送，也允许接收。若已知其控制端口地址为 0FFFH，数据端口地址为 0FF2H，试编写初始化程序。

第 10 章　数模 D/A 和模数 A/D 转换接口

在微型计算机控制系统中，微型计算机的主要工作是对现场信息进行采集、处理、分析和实时控制。这些信息通常是温度、速度、压力、流量、电流和电压等一些连续变化的模拟量。微型计算机只能接收数字量并进行运算和处理，因此必须将这些模拟量转换为数字量（简称 A/D 转换）。而微型计算机运算的结果也只能以数字量输出，所以还需将微型计算机输出的数字量转换成为模拟量（简称 D/A 转换），去控制或驱动被控设备。

10.1　模拟量输入输出通道

模拟量输入输出通道的一般结构如图 10.1 所示。

10.1.1　模拟量输入通道

在计算机控制系统中，需要通过模拟量输入通道将生产现场的模拟量信号转换为数字量信号。模拟量输入通道主要包括下列六个环节。

1. 传感器

传感器是将工业生产现场的某些非电物理量（如温度、速度、流量、压力等）转换为电量的器件。例如，温度、压力、位移、速度、流量等多种传感器。

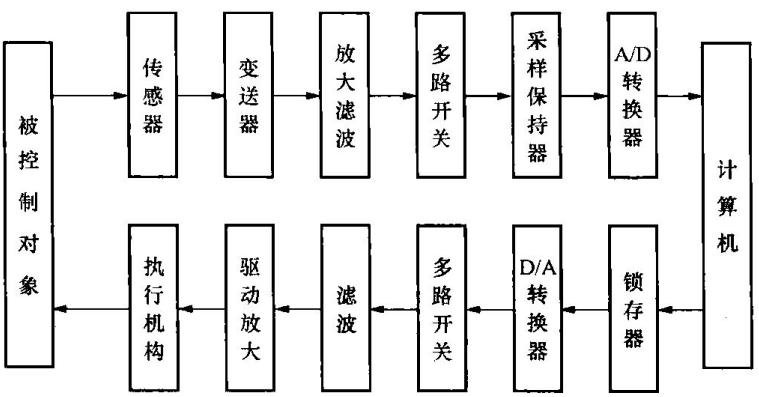


图 10.1　模拟量输入输出通道的一般结构

2. 变送器

一般来讲，传感器输出的电信号都比较弱，并且传感器输出的不一定是电信号，因此，通过变送器将传感器输出的信号转换成一种统一的电信号，一般转换成 0～10mA，4～20mA 的电流信号，或者 0～5V 的电压信号。

3. 放大滤波

传感器和变送器通常都安装在现场，环境比较恶劣，其输出信号常叠加有高频干扰信号。因此，需要通过放大滤波环节将信号放大并去除干扰信号。

4. 多路转换开关

在微机控制系统中，一台计算机可以对生产过程的多个回路进行检测和控制，但是这些检测和控制不是同时进行的，而是按照分时的方式工作。用多路转换开关，使多个模拟信号共用一个 A/D 转换器进行采样和转换。此外，在模拟量输出通道中，为了实现多回路控制，也需要通过多路开关将控制信号分配到各输出控制回路。

5. 采样保持电路

在数据采样期间，保持输入信号不变的电路称为采样保持电路。由于 A/D 转换过程需

要一定时间，为了保证 A/D 转换的精度，对于一些变化较快的模拟量信号必须在 A/D 转换进行期间保持该值不变，而在 A/D 转换结束后又能跟踪输入信号的变化。因此，在 A/D 转换器前面增加采样保持电路，以保证 A/D 转换器在转换过程中其输入信号保持不变。

6. A/D 转换器

A/D 转换器是模拟量输入通道的中心环节，它的作用是将输入的模拟量信号转换成计算机能够识别的数字量信号，以便计算机进行分析和处理。A/D 转换器又称为模数转换器。

10.1.2　模拟量输出通道

由于计算机输出的是数字量信号，而有的控制系统要求使用电压或电流信号进行控制。因此，需要将计算机输出的数字信号转换为模拟信号。这个过程的实现由模拟量输出通道来完成。

模拟量输出通道中的核心部件是 D/A 转换器，D/A 转换器又称为数模转换器，D/A 转换是 A/D 转换的逆过程。

由于计算机运行速度很快，其输出的数据在数据总线上稳定的时间很短。因此，在计算机与 D/A 之间必须增加一级锁存器以保持数字量信号的稳定。D/A 转换器的输出端一般还要增加低通滤波器，以平滑输出波形，这就是滤波。另外，为了能够驱动执行机构，还需要将输出的小功率的模拟量信号进行放大，即驱动放大。

10.2　D/A　转　换　器

10.2.1　D/A 转换器的一般工作原理

D/A 转换器用于将计算机送出的数字量信号转换成与此数值成正比的模拟量电压或电流信号。我们知道，一个二进制数中的各位由于在数中的位置不同，所具有的位权也不同。为了将数字量转换成模拟量，应将每一位代码按权大小转换成相应的模拟输出分量，然后根据叠加原理将各位代码对应的模拟输出分量相加，其总和就是与数字量成正比的模拟量，由此完成 D/A 转换。典型的 D/A 转换器基本电路由模拟开关、电阻网络和运算放大器组成，如图 10.2 所示。

图 10.2 中，模拟开关和电阻网络用来构成解码网络，是实现 D/A 转换的核心部件。U_{ref}为基准电压，R_f 为反馈电阻，计算机输出的数字量通过解码网络和运算放大器后输出电压 U_o。通常有两种形式的解码网络：二进制权电阻型 D/A 解码网络和 R-2R 梯形 D/A 解码网络。

首先，我们来介绍运算放大器的原理，然后引出 D/A 转换器原理。一个简单的运算放大器电路如图 10.3 所示。

运算放大器的同相输入端（＋）接地，由于运算放大器的输入阻抗非常大，流入反相输入端的电流几乎为 0。又由于运算放大器同相端与反相端之间的电流也非常小，Σ点被称为虚地点。基本运算放大器输出电压与输入电压的关系如下：

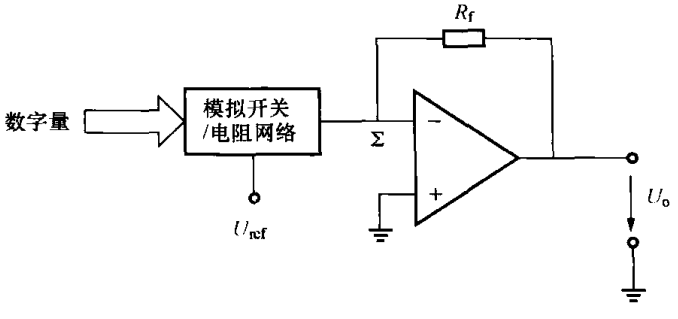


图 10.2　D/A 转换器基本电路

$$U_o = -\frac{R_f}{R}U_i \tag{10.1}$$

式中，R 为输入电阻；R_f 为反馈电阻；U_i为输入电压；U_o 为输出电压。

若基本运算放大器的输入端有 n 个支路，如图 10.4 所示，则输出与输入电压的关系为：

$$U_o = -R_f\sum_{i=1}^{n}\frac{1}{R_i}U_i = -R_fU_i\sum_{i=1}^{n}\frac{1}{R_i} \tag{10.2}$$

式中，R_i 为第 i 个支路的输入电阻。

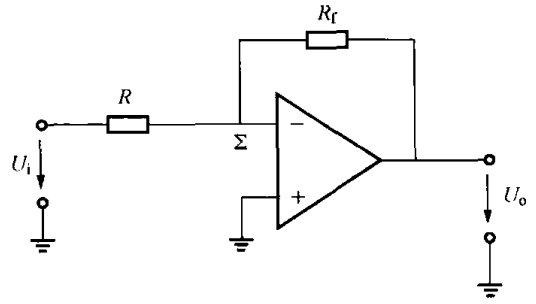


图 10.3　基本运算放大器电路

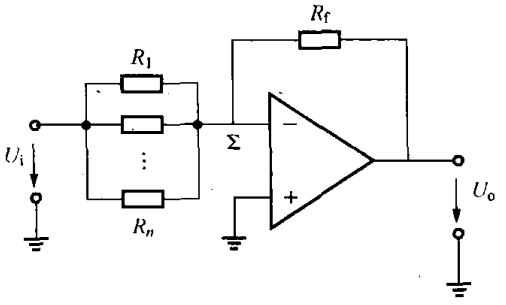


图 10.4　多路输入的运算放大器电路

下面再来看以基本运算放大器为基础的 D/A 转换器将数字量转换为模拟量的两种主要解码网络形式。

1. 权电阻解码网络

对于图 10.4，如果使各支路上的输入电阻 R_1，R_2，R_3，…，R_n 分别等于 2^0R，2^1R，2^2R，2^3R，…，2^nR，2^iR，即称为加权电阻值，则有：

$$U_o = -R_f\sum_{i=0}^{n-1}\frac{1}{2^iR}U_{ref} = -\frac{R_f}{R}U_{ref}\sum_{i=0}^{n-1}\frac{1}{2^i} \tag{10.3}$$

如果每个支路接入一个开关 Si，开关闭合时用 Si=1 表示，开关断开时用 Si=0 表示，开关状态由二进制数字量来控制，就构成了权电阻解码网络。如图 10.5 所示，是一个 8 位权电阻解码网络。则输入和输出电压的关系变为：

$$U_o = -\frac{R_f}{R}U_{ref}\sum_{i=0}^{n-1}\frac{1}{2^i}S^{n-i-1} \tag{10.4}$$

图 10.5 中，如果用二进制数的各位来控制权解码网络中的权电阻开关 Si（i=0～7），当输入一个 8 位二进制数字量时，二进制数字量的最高位 D_7 使开关 S7 闭合，二进制数字量的次高位 D_6 使开关 S6 闭合，依次顺序进行，二进制数字量的 D_0 使 S0 闭合。因此，二进制数字量通过权电阻网络来控制开关的通断，使电路输出对应的电压信号，从而实现了二进制数字量到模拟量的转换，这就是 D/A 转换器的基本原理。

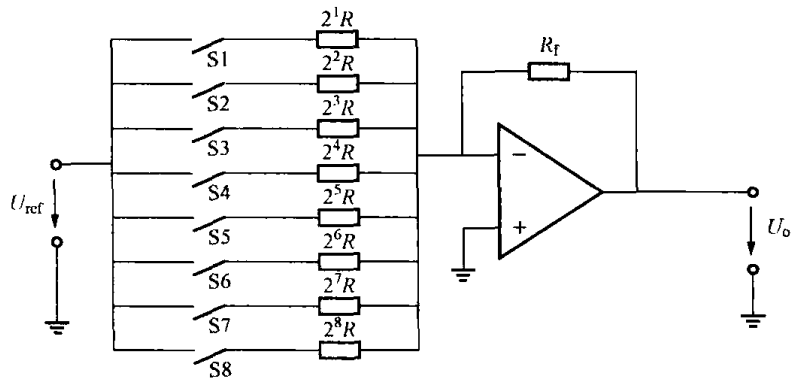


图 10.5　8 位权电阻网络

当所有开关 Si 断开时，U_o=0；

当所有开关 Si 闭合时，输出电压 U_o 为最大，即 $U_o=-\frac{2^i-1}{2^i}U_{ref}$；

当输入数字量 B=$D_7D_6D_5D_4D_3D_2D_1D_0$，则输出电压为：

$$\begin{aligned}U_o&=-\frac{U_{ref}}{2}\sum_{i=1}^{8}\frac{1}{2^{i-1}}Si\\&=-\frac{U_{ref}}{2}\times\left(D_7\times\frac{1}{2^0}+D_6\times\frac{1}{2^1}+D_5\times\frac{1}{2^2}+D_4\times\frac{1}{2^3}\right.\\&\quad\left.+D_3\times\frac{1}{2^4}+D_2\times\frac{1}{2^5}+D_1\times\frac{1}{2^6}+D_0\times\frac{1}{2^7}\right)\end{aligned}$$

权电阻 D/A 解码网络的输出电压 U_o 绝对值正比于输入的二进制数字量，D/A 转换器的精度与基准电压 U_{ref} 和权电阻的精度以及数字量的位数有关。显然，位数越多，转换精度就越高，但同时所需要的权电阻的数量越多，并且以 2 的加权形式递增。由于在集成电路中制造高阻值的精密电阻比较困难，使权电阻网络的 D/A 转换容易带来较大的转换误差。因此常用 R-2R 梯形解码网络来替代权电阻网络。

2. R-2R 梯形解码网络

图 10.6 所示为一个简化了的 R-2R 梯形解码网络，它仅使用了两种阻值的电阻（R 和 $2R$），由于 R-2R 接成梯形，故称梯形网络。梯形解码网络使用的电阻种类较少，易于集成工艺生产，精度也容易保证，因此得到比较广泛的应用。

在 R-2R 梯形网络中，整个电路是由相同的支路组成，每个支路有两个电阻、一个开关，每个开关由对应的二进制数位来控制，即一位二进制数来控制一个支路。

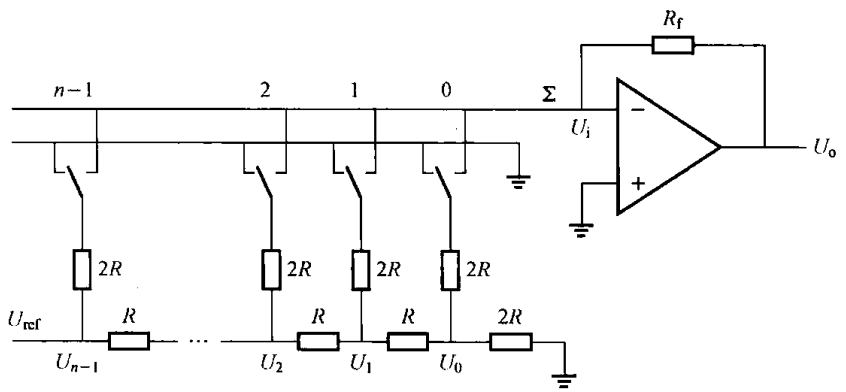


图 10.6　R-2R 梯形解码网络

R-2R 梯形电阻网络基本组成包括 U_{ref} 基准电压源，S0～Sn－1模拟二进制开关，R-2R 电阻网络和运算放大器。分析 R-2R 梯形解码网络的方法和分析权电阻网络一样，先求出其中一个支路的电压输出分量，再用叠加法求得总的输出电压。由图 10.6 看出，当模拟开关切换至左侧，支路电阻接到地端；当模拟开关切换至右侧，支路电阻则接到虚地端。所以无论开关接到哪个方向，都可以认为是接“地”。不过，只有开关倒向右边时，才给运算放大器输入端提供电流。因此，用二进制数据的各位来控制网络中的开关，当输入的二进制数不同时，输出的电流也不同，从而实现数字量到模拟量的转换。

梯形电阻网络中，从 U_{ref} 基准电压源至接地点之间的等效电阻均为 R，R-2R 梯形电阻网络的输出和输入电压的关系表达式为：

$$U_o = -\frac{R_f}{2R} \times U_{ref} \sum_{i=0}^{n-1} \frac{1}{2^{(n-i-1)}} \tag{10.5}$$

式中，n 为电阻网络的支路数。

由式（10.5）可知，输出电压 U_o 正比于输入的数字量，若输入 8 位的数字量，则式（10.5）可简化为式（10.6），即 8 位 D/A 转换器的输出电压和数字量的关系式为：

$$U_o = -\frac{R_f}{2R} \times U_{ref} \sum_{i=0}^{7} \frac{1}{2^{(7-i)}} \tag{10.6}$$

当一个二进制数为 $B=D_7D_6D_5D_4D_3D_2D_1D_0$，时，输出的电压为：

$$V_o = -\frac{R_f}{2R} \times U_{ref} \left(D_7 \times \frac{1}{2^0} + D_6 \times \frac{1}{2^1} + D_5 \times \frac{1}{2^2} + D_4 \times \frac{1}{2^3} + D_3 \times \frac{1}{2^4} + D_2 \times \frac{1}{2^5} + D_1 \times \frac{1}{2^6} + D_0 \times \frac{1}{2^7} \right)$$

分析可知，R-2R 梯形网络输出电压 U_o 的绝对值正比于输入的二进制数字量，所以该网络可以实现二进制数字量到模拟电压的转换，并且随着位数的增加，误差减小，转换精度越高。

10.2.2 D/A 转换器的主要技术指标

衡量一个 D/A 转换器性能的主要参数有分辨率、转换时间和精度、线性度等，应根据需求合理选用 D/A 转换器芯片。

1. 分辨率

分辨率指能够分辨最小量化信号的能力。它表示输入数据发生 1LSB（最低一位二进制数）的变化时所对应的输出模拟量的变化值，可用数字量的位数来表示，如 8 位、10 位等。例如，对于一个 n 位的分辨率 Δ 与输入数字量的位数 n 之间有如下关系：

$$\Delta = \frac{FSR}{2^n}$$

式中，FSR 是 D/A 转换器的满量程，它近似等于输入满码时所对应的输出电压值。

通常也用百分比来表示分辨率。例如：

对于 8 位的 D/A 转换器，$n=8$，$2^n=256$，其分辨率为 $FSR/256=0.39\%FSR$；

对于 12 位的 D/A 转换器，$n=12$，$2^n=4096$，其分辨率为 $FSR/4096=0.0244\%FSR$。

设一个 D/A 转换器能转换 8 位二进制，转换后的电压满量程是 5V，则它的分辨率为 5V/256≈20mV。即转换器能正确地分辨出 20mV 的电压变化。由于分辨率与 D/A 转换器的位数之间有固定的对应关系，一般简单地用它们的位数来表示分辨率。输入数字量位数愈多，输出电压可分的等级愈多，即分辨率愈高。

2. 转换时间

转换时间指当输入数字量满刻度变化（如数字量从全 0 到全 1）时，从数字量输入到输出模拟量达到与最终值相差±1/2LSB（最低有效位）相当的模拟量值所需要的时间。电流型 D/A 转换较快，一般在几纳秒到几百微秒之内。电压型 D/A 转换较慢，取决于运算放大器的响应时间。

3. 转换精度

由于转换器内部电路的误差等原因，当一个确定的数字量输入 D/A 后，它的实际输出值与该数字量所对应的理论输出值之间会有一定的误差，它就是 D/A 转换器的精度，通常

用此差值与满量程输出电压或电流的百分比来表示。例如，某一 D/A 转换器的电压满量程为 10V、精度为 0.02%，则输出电压的最大误差为 10.00V×0.02%=20mV。一般 D/A 转换器的误差不大于 1/2LSB。

4. 线性度

线性度指 D/A 转换器输入的数字量变化时，其输出的模拟量按比例关系变化的程度。理想的 D/A 转换器是线性的，但实际上有误差，模拟输出偏离理想输出的最大值称为线性误差。

5. 输出形式

D/A 转换器输出形式主要有两大类，即电压型和电流型，如一般为 0～5V、高电压输出型为 0～24V，电流输出型一般为 4～20mA 或 0～20mA，最高可达 3A。

10.2.3　典型 D/A 转换器芯片

D/A 转换器的种类很多，在常用的 D/A 芯片中根据转换位数分有 8 位、10 位、16 位等；根据输出模拟量信号类型分有电流输出和电压输出；根据内部结构又可分为芯片内部含数据锁存器的和不含数据锁存器的两类。对内部含数据锁存器的 D/A 芯片，能够作为 CPU 的一个外部设备端口直接与微机系统总线连接。在需要进行 D/A 转换时，CPU 通过片选信号和写控制信号将数据送到 D/A 转换器。

1. 8 位 D/A 转换器 DAC0832

DAC0832 是 CMOS 单片 8 位 D/A 转换器，采用 20 引脚双列直插式封装，片内有一个 T 型电阻解码网络，用于对基准电流进行分流以完成 D/A 转换。它内部具有数据输入寄存器做锁存器，可以直接与 8086、8088 等 CPU 连接。其内部结构和外部引脚如图 10.7 所示。

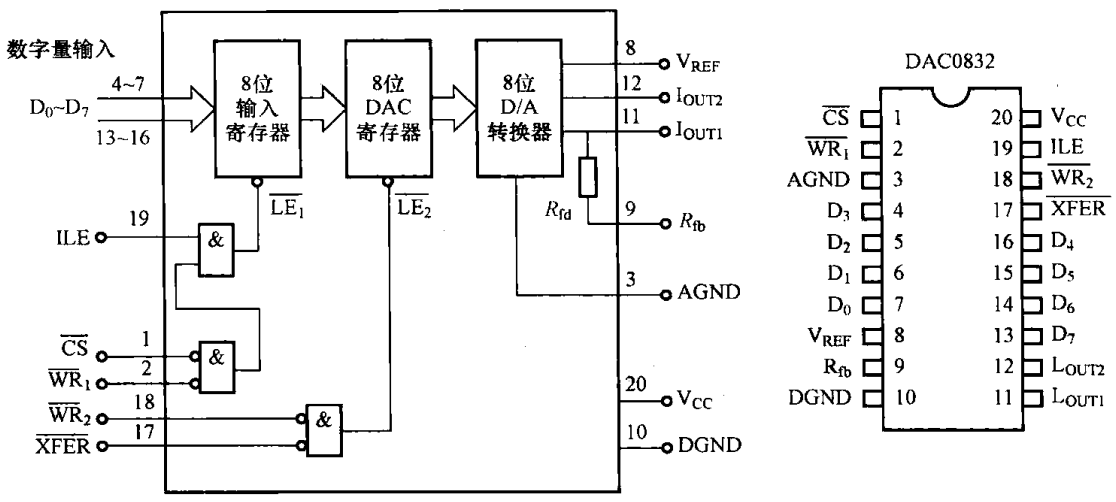


图 10.7　DAC0832 内部结构及外部引脚

(1) 内部结构。从图 10.7 可以看出，DAC0832 的内部有两级锁存器：第一级是 8 位的数据输入寄存器，由控制信号$\overline{CS}$、ILE 及$\overline{WR_1}$控制；第二级是 8 位的 DAC 寄存器，由控制信号$\overline{WR_2}$、$\overline{XFER}$控制。因此，DAC0832 具有双缓器工作方式，可以在输出模拟信号的同时采集下一个数字量，以提高转换速度。另外，具有两级锁存器，可构成多片 DAC0832 同时输出模拟量的系统。即实现多个 D/A 转换器同时工作，并输出多个模拟量信号，这种情

况下，每一种参数需要一片 DAC0832。

图 10.7 中，当 ILE 为高电平、$\overline{CS}$和$\overline{WR_1}$为低电平时，LE_1 为高电平，这种情况下，输入寄存器的输出跟随输入而变化。此后，当$\overline{WR_1}$由低电平变高电平时，LE 成为低电平，此时，数据被锁存到输入寄存器中，这样，输入寄存器的输出端不再跟随外部数据的变化而变化。

对第二级锁存器来说，$\overline{XFER}$和$\overline{WR_2}$同时为低电平时，LE 为高电平，这时，8 位的 DAC 寄存器的输出跟随输入而变化，此后，当数据$\overline{WR_2}$由低变高时，LE 为低电平，于是，将输入寄存器的信息锁存到 DAC 寄存器中。

为了使用 DAC0832 进行数/模转换，可分别对二级锁存器进行控制，称为单缓冲方式、双缓冲方式和直通方式。

（2）DAC0832 各引脚功能说明如下：

D_0～D_7：为 8 位数据输入端，与系统数据总线相连，接收计算机输出的数字量。

ILE：允许输入锁存信号，输入、高电平有效。此信号与$\overline{CS}$、$\overline{WR_1}$一起将要转换的数据送入输入寄存器。

$\overline{CS}$：为片选信号，输入、低电平有效。此信号与 ILE 信号一起用于控制$\overline{WR_1}$信号能否起作用。

$\overline{WR_1}$：写信号 1，为输入寄存器的写信号，输入、低电平有效。

$\overline{WR_2}$：写信号 2，为 DAC 寄存器写信号，输入、低电平有效。

$\overline{XFER}$：是数据传送控制信号，输入、低电平有效。当$\overline{WR_2}$、$\overline{XFER}$同时为低电平时，将输入寄存器的信息锁存在 DAC 寄存器中。

I_{OUT1}和 I_{OUT2}：它们是 DAC0832 的两个模拟电流输出端。当 DAC 寄存器中的内容为 0FFH 时，输出电流 I_{OUT1}最大，当 DAC 寄存器中的内容为 00H 时，I_{OUT1}为 0。I_{OUT2}为一个常数和 I_{OUT1}的差，即 $I_{OUT1}+I_{OUT2}=$常数。因而，常把 I_{OUT2}接地，I_{OUT1}作为输出端。

R_{fb}：为反馈内阻引出端，接运算放大器的输出。

V_{REF}：是基准电压输入端，可在－10～＋10V 范围内选用。

V_{CC}：是芯片的电源输入端，可在＋5～＋15V 范围内选用，最佳工作状态是＋15V。

AGND 和 DGND：AGND 是芯片模拟电路接地端；DGND 是芯片数字电路接地端。

需要指出的是，关于模拟电路和数字电路共存的系统中，模拟电路接地端和数字电路接地端问题。对于 D/A 转换器（或后面讲到的 A/D 转换器），由于转换器内部主要是模拟电路，运算放大器内部则完全是模拟电路，因此，它们属于模拟电路芯片。而 CPU、锁存器、译码器则属于数字芯片。这两类芯片要用两组独立的电源供电，并且在系统中要把“模拟地”连接在一起，“数字地”要连接在一起，但两种“地”一般不连接在一起，以免造成数字信号通过数字地线干扰模拟信号。

（3）DAC0832 的工作方式。DAC0832 的工作过程如下：CPU 执行输出指令，输出 8 位数据给 DAC0832；在 CPU 执行输出指令期间，使 ILE、$\overline{WR_1}$、$\overline{CS}$三个控制信号端都有效，8 位数据锁存在 8 位输入寄存器中；当$\overline{WR_2}$、XFER 二个控制信号端都有效时，8 位数据再次被锁存到 8 位 DAC 寄存器，这时 8 位 D/A 转换器开始工作，8 位数据转换为相对应的模拟电流，从 I_{OUT1}和 I_{OUT2}输出。

根据这两个锁存器使用方法的不同，DAC0832 有三种工作方式：

1）单缓冲方式。单缓冲方式是使两个寄存器中的任意一个处于直通状态，而另一个工作在受控锁存状态。例如，要想使输入寄存器受控，DAC 寄存器直通，则可将$\overline{WR_2}$、$\overline{XFER}$接数字地，ILE 接 V_{CC}。此时，将$\overline{CS}$接端口地址译码器输出，$\overline{WR_1}$接$\overline{WR}$信号，当 CPU 向输入寄存器的端口地址发出写命令时，数据就写入输入寄存器。因为 DAC 为直通状态，所以写入到数据寄存器的数据立刻进行 D/A 转换，如图 10.8 所示。

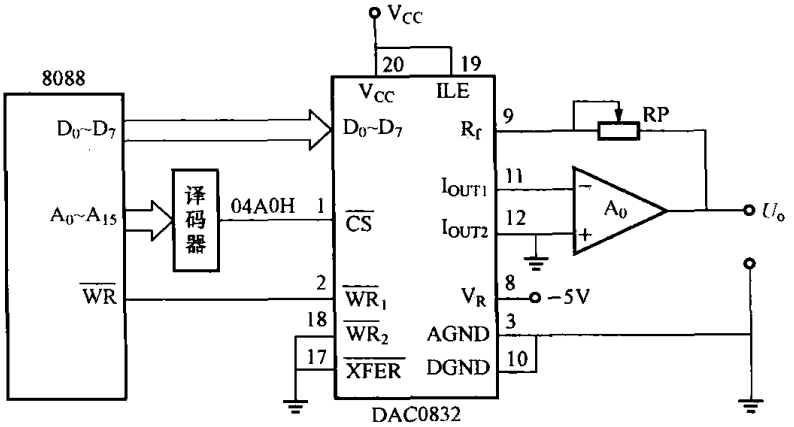


图 10.8　DAC0832 单缓冲方式下的电路连接

2）双缓冲方式。在这种方式下，数据通过二个寄存器锁存后送入 D/A 转换电路，执行两次写操作才能完成一次 D/A 转换。该方式下数据接收和启动转换可以异步进行，可以在 D/A 转换的同时，接收下一个数据，提高了模数转换效率。它还可用于多个通道进行 D/A 转换，同时输出多个模拟量信号的场合。DAC0832 在双缓冲方式下的接线如图 10.9 所示。

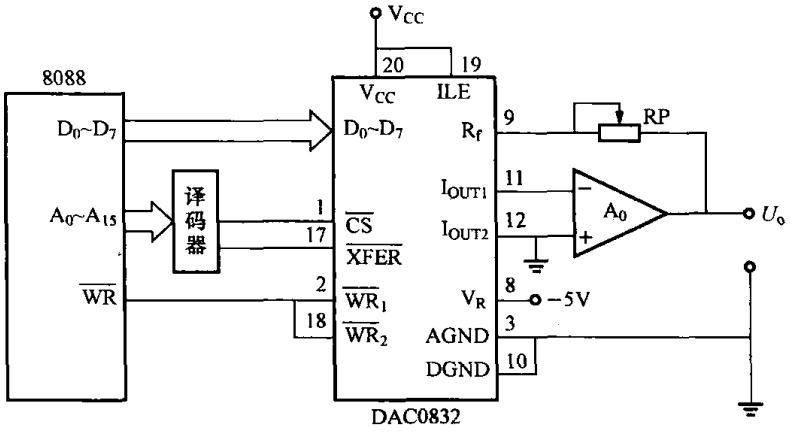


图 10.9　DAC0832 双缓冲方式下的电路连接

3）直通方式。这种方式下是将$\overline{CS}$、$\overline{WR_1}$、$\overline{WR_2}$以及$\overline{XFER}$引脚都直接接数字地，ILE 接 V_{CC}。芯片处于直通状态。数据直接送入 D/A 转换器电路进行 D/A 转换，即模拟量输出始终跟踪输入端 D_0～D_7 的变换。由于这种方式下 DAC0832 不能直接与 8088/8086CPU 的数据总线相连，故在实际工程中很少使用。

(4) DAC0832 的主要技术参数。

分辨率：8 位。

转换时间：1μs。

线性误差：(0.05%～0.2%)*FSR*。

参考电压：±10V。

功耗：20mW。

2. 12 位 D/A 转换器 DAC1210

DAC1210 是 12 位的高分辨率电流输出型 D/A 转换器，具有 24 根引脚双列直插式器件。它的主要指标为：电流建立时间 $t_s=1\mu s$，工作电压 5～15V，参考电压范围为±25V。图 10.10 所示为 DAC1210 的内部结构和引脚图。

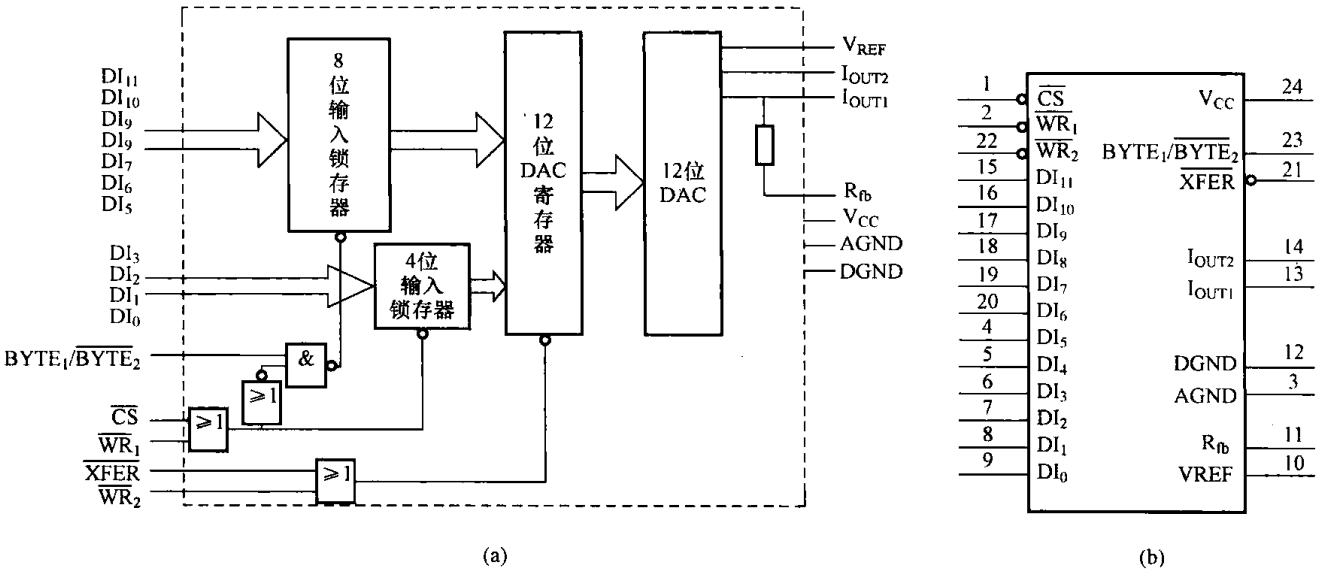


图 10.10 DAC1210 的内部结构和引脚图

(a) 内部结构；(b) 引脚

DAC1210 工作原理与 8 位 DAC0832 没有多大区别，其内部结构与 DAC0832 非常相似，也具有两级锁存器，不同的是 DAC1210 有 12 根数据输入线，内部有 12 位双缓冲寄存器和 D/A 转换器，其中 12 位输入寄存器由一个 8 位寄存器和一个 4 位寄存器组成。当$\overline{CS}=0$，$\overline{WR_1}=0$ 时，若 $BYTE_1/\overline{BYTE_2}$为高电平，选通 8 位输入寄存器；而 $BYTE_1/\overline{BYTE_2}$为低电平时，选通 4 位输入寄存器。

DAC1210 引脚简述如下：

DI_0～DI_{11}：数据输入端，与系统数据总线相连，接收计算机输出的数字量。

$BYTE_1/\overline{BYTE_2}$：字节控制端。当 $BYTE_1/\overline{BYTE_2}$高电平时，输入高 8 位数据；$BYTE_1/\overline{BYTE_2}$输入低电平时，输入低 4 位数据。

$\overline{CS}$：片选信号，低电平有效。

$\overline{WR_1}$：写信号 1，低电平有效。

$\overline{WR_2}$：写信号 2，低电平有效。

$\overline{XFER}$：12 位 DAC 寄存器控制端，低电平有效。

DAC1210 可以用于 16 位 CPU 系统，接口电路的详细设计见 10.2.4 节。

10.2.4　CPU 与 D/A 转换芯片的接口设计

1. CPU 与 8 位 D/A 转换芯片的接口

DAC0832 是一种 8 位的 D/A 芯片，芯片内部包含数据锁存器，可以直接与系统总线相连，看作一个输出端口。CPU 向该端口输出一个 8 位数据，意味着在 D/A 输出端就可以得到一个相应的输出电压。在 8088CPU 与 DAC0832 的接口设计中，CPU 的地址信号通过译码器产生片选信号与 DAC0832 的$\overline{CS}$端相连，该信号既是片选信号，也是输入寄存器的控制信号。另外，由于 CPU 的控制信号$\overline{WR}$与 DAC0832 的写信号$\overline{WR_1}$相连，当执行 OUT 指令，会使$\overline{WR_1}$写信号有效，并与$\overline{CS}$信号一起，将输出数据锁存到输入寄存器中。前面讲到的图 10.9，就是 DAC0832 工作在单缓冲方式下的接口电路，在微机控制下就可产生各种变化规则的控制电压波形。

[例 10.1]　在图 10.9 中，使 DAC0832 工作在单缓冲方式，用来产生最高电压 5V、最低电压 0V 的三角波。假设 DAC0832 地址为 04A0H，当 CPU 输出数据 00H～FFH，则 DAC0832 输出 0～5V 的电压，为三角波上升部分；当 CPU 输出数据 FFH～00H，则 DAC0832 输出 5～0V 的电压，为三角波下降部分。三角波程序流程图如图 10.11 所示。

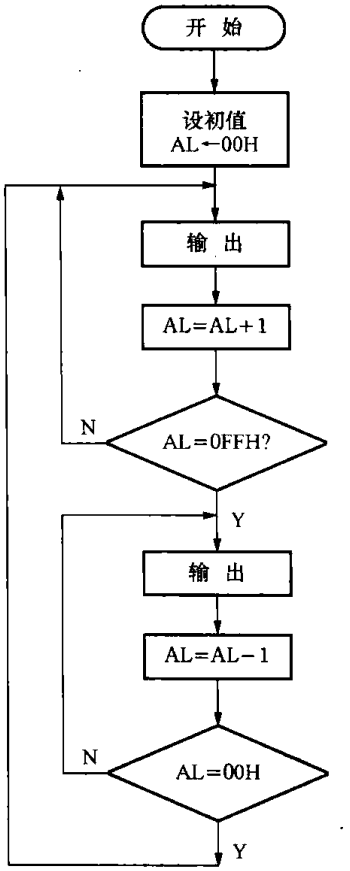


图 10.11　三角波程序流程图

解　DAC0832 单缓冲方式下，产生三角波的程序如下：

```
      MOV DX, 04A0H   ; DAC0832 芯片端口地址送 DX
      MOV AL, 00H     ; 设置起始输出数据
AA1 : OUT  DX, AL
      INC AL          ; 修改输出数据
      CMP AL, 0FFH
      JNZ AA1
AA2 : OUT DX, AL
      DEC AL          ; 修改输出数据
      CMP AL, 00H
      JNZ AA2
      JMP AA1
```

若把 DAC0832 的输出端接到示波器的输入端，运行上述程序，便可在示波器上看到连续的三角波波形。

如果使 DAC0832 产生锯齿波，程序如下：

```
      MOV DX, 04A0H   ; DAC0832 端口地址送 DX
      MOV AL, 00H     ; 设置起始输出数据
AA1 : OUT  DX, AL
```

```
DEC AL          ；修改输出数据
JMP AA1         ；循环形成周期锯齿波
```

可以看出，利用 D/A 转换器可以产生频率比较低的任意波形。这些波形是由程序产生的，当 CPU 的速度一定时，所产生的波形频率不可能很高。

[例 10.2] 采用并行接口芯片 8255A 作为 DAC 与 CPU 之间的接口电路，电路如图 10.12 所示。

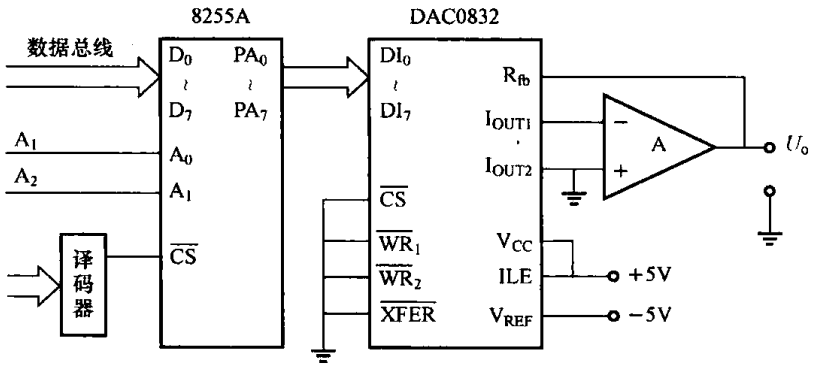


图 10.12 8255A 控制的 DAC 接口电路

解 通过 8255A 的 A 口将数据传送到 DAC0832 进行 D/A 转换。设 8255A 端口地址为 0FFF0H～0FFF3H，程序如下：

```
；8255A 初始化
MOV DX, 0FFFF3H     ；8255 的控制口地址
MOV AL, 10000000B   ；8255 的方式字
OUT DX, AL
；生成三角波的循环程序
MOV DX, 0FFF0H      ；8255 的 A 口地址
MOV AL, 00H
L1: OUT DX, AL
    INC AL
    JNZ L1
L2: OUT DX, AL
    DEC AL
    JNZ L2
    JMP L1
；生成锯齿波的循环程序
    MOV DX, 0FFF8H      ；8255 的 A 口地址
    XOR AL, AL
L3: INC AL
    OUT DX, AL          ；AL 加 1 后输出
    JMP L3              ；齿高为 FFH 对应的电压
```

注：改变锯齿波的频率，只需在 JMP L3 前插入延时程序即可实现。

```
    ；生成矩形波的循环程序
    MOV DX，0FFF0H      ；8255 的 A 口地址
L4：MOV AL，10H         ；置输出矩形波下限
    OUT DX，AL
    CALL DELAY1         ；调 DELAY1 延时程序
    MOV AL，0C0H        ；置输出矩形波上限
    OUT DX，AL
    CALL DELAY2         ；调 DELAY2 延时程序
    JMP L4
    ；生成梯形波的循环程序
    MOV DX，0FFF0H      ；8255 的 A 口地址
    MOV AL，20H         ；置下限
    OUT CX，AL
L5：INC AL
    OUT DX，AL
    CMP 0D0H            ；与上限比较
    JC L5
    CA11 DELAY3         ；为上限延时程序
L6：DEC AL
    OUT DX，AL
    CMP 20H
    JNC L6
    JMP L5
```

2. CPU 与 12 位 D/A 转换芯片的接口

[例 10.3] DAC1210 与 8086CPU 的连接如图 10.13 所示。

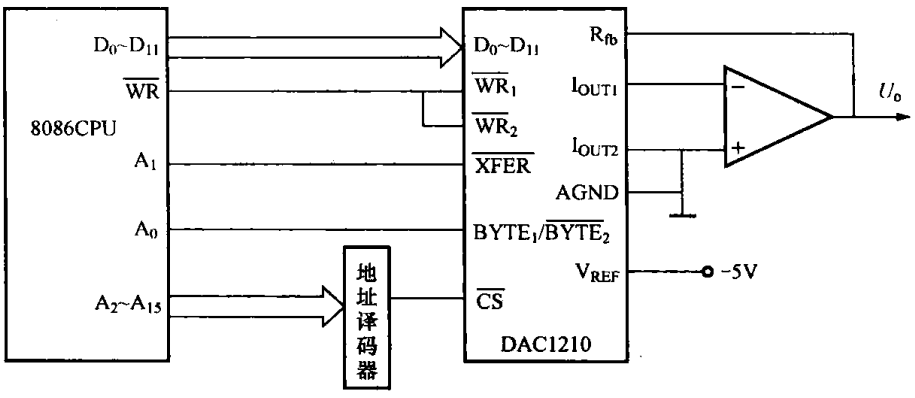


图 10.13　DAC1210 与 8086CPU 的接口电路

解　设 DAC1210 的端口地址为 PORT，待转换数据在 DATA 单元中，完成 D/A 转换的程序如下：

```
MOV  DX，PORT  ；DAC1210 的端口地址送 DX
MOV  AX，DATA  ；要转换的数据送
OUT  DX，AX    ；将数字量送 D/A 转换器进行转换
```

10.3 A/D 转 换 器

A/D 转换器是指通过一定的电路将模拟量转换为数字量，以便于计算机进行处理。它与 D/A 转换器一样，是微型计算机应用系统中的一种重要接口，常用于数据采集系统。

10.3.1 A/D 转换器的工作原理及技术指标

实现 A/D 转换的方法很多，常见的有计数型 A/D 转换、逐次逼近型 A/D 转换和双积分型 A/D 转换三种。其中，计数型 A/D 转换是早期的形式，目前常用后两种方法。这里以常用的逐次逼近型 A/D 转换器为例，来介绍 A/D 转换器的一般工作原理。

1. 逐次逼近式 A/D 转换器的工作原理

图 10.14 为 8 位逐次逼近式 A/D 转换器的内部结构，由时序控制逻辑、SAR 移位寄存器、D/A 网络以及比较器等组成。设 U_i 是要转换的模拟量电压输入信号。

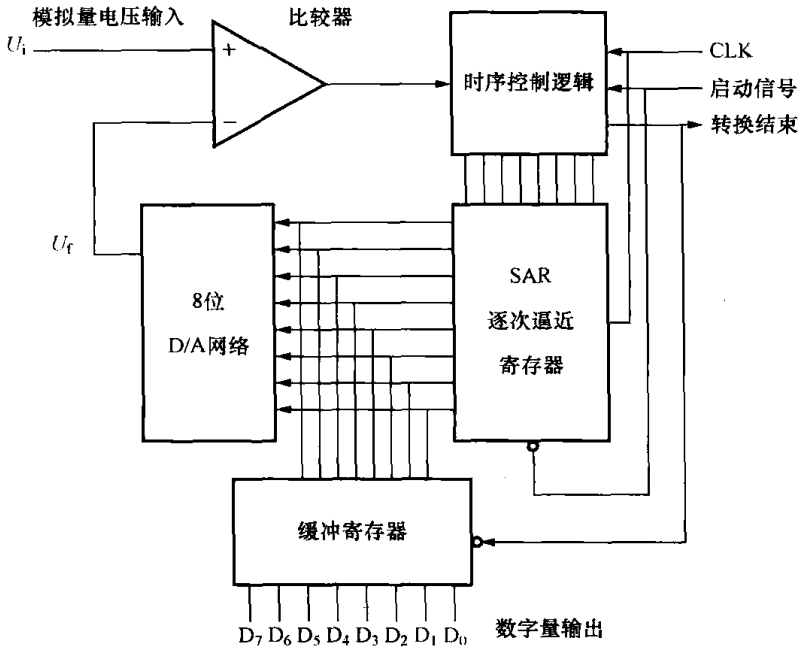


图 10.14 逐次逼近式 A/D 转换原理

逐次逼近式 A/D 转换器的工作原理如下：

由 CPU 发出启动信号，时序控制逻辑电路工作，并将 SAR 逐次逼近寄存器各位清零，这时，D/A 网络输出电压为 0。通过时序控制逻辑电路采用设置试探值的方法，首先从最高位开始，将 D_7 置“1”（比如 8 位 A/D 转换器），其余位为 0，送入 SAR 逐次逼近寄存器。逐次逼近寄存器将该数据送入控制 D/A 网络开关的闭合与断开，D/A 网络根据输入的二进制量输出一个模拟量电压 U_f。U_f 与输入电压 U_i 通过比较器进行比较，如果 $U_f \leqslant U_i$，则保留 SAR 寄存器 D_7 位的置“1”，并使 $U_i = U_i - U_f$；若 $U_f > U_i$，则将 SAR 逐次逼近寄存器 D_7 位复位。接下来使 SAR 次高位 D_6 置“1”，其他位仍为 0，根据相同的原理判断置位是否保留，这样一直比较至最低位 D_0，当 D_0 比较结束后 A/D 转换过程结束，此时 $D_7 \sim D_0$ 即为 A/D 转换输出的数字量。同时，A/D 转换器输出转换结束信号去申请中断或提供 CPU 程序查询信号来读取数据。

例如，某个 8 位的 A/D 转换器，如果 $U_{ref}=5V$ 输入的模拟电压为 0～5V，则输出对应的值为 0～FFH，并且最低有效位所对应的输出电压为 $5/(2^8-1)=19.61(mV)$。现设输入模拟电压为 3.5V，其变换过程如下：

位序号	比较表达式	二进制值
D_7	$3.5V-2^7\times19.61mV=0.99V>0$	$D_7=1$
D_6	$0.99V-2^6\times19.61mV=-0.265<0$	$D_6=0$
D_5	$0.99V-2^5\times19.61mV=0.362>0$	$D_5=1$
D_4	$0.362V-2^4\times19.61mV=0.048>0$	$D_4=1$
D_3	$0.048V-2^3\times19.61mV=-0.109<0$	$D_3=0$
D_2	$0.048V-2^2\times19.61mV=-0.03<0$	$D_2=0$
D_1	$0.048V-2^1\times19.61mV=0.009>0$	$D_1=1$
D_0	$0.009V-2^0\times19.61mV=-0.187<0$	$D_0=0$

这样，就把 3.5V 模拟量转换成了数字量 10110010。

2. A/D 转换器主要技术指标

A/D 转换器的主要技术指标有分辨率、转换误差、转换时间、量程、输出电平、输出编码和工作温度等。

（1）分辨率。指 A/D 转换器可转换成数字量的最小输入电压值。例如：1 个 10 位 A/D 转换器，满量程输入电压为 5V，则分辨率就为 $\frac{5000}{2^{10}}=\frac{5000}{1024}\approx5mV$。若模拟输入电压值的变化小于 5mV，则 A/D 转换器输出保持不变。同样 5V 电压，若采用 12 位 A/D 转换器，则分辨率提高为 $\frac{5000}{2^{12}}=\frac{5000}{4096}\approx1mV$。所以，A/D 转换器常用它的数字量输出位数来表示。当最大输入电压一定时，输出位数愈多，分辨率愈高。

（2）转换误差。转换误差通常是以输出误差的最大值形式给出。它表示 A/D 转换器实际输出的数字量和理论输出数字量之间的差别。常用最低有效位的倍数表示。例如，给出相对误差≤±LSB/2，表明实际输出的数字量和理论上应得到的输出数字量之间的误差小于最低位的半个字（即 $\frac{1}{2}$LSB）。

（3）转换时间。转换时间是指 A/D 转换器从转换控制信号到来开始，到输出端得到稳定的数字信号所经过的时间。A/D 转换器的转换时间与转换电路的类型有关。不同类型的转换器转换速度相差甚远。其中并行式 A/D 转换器的转换速度最高，8 位二进制输出的 A/D 转换器转换时间可达到 50ns 以内，逐次逼近式 A/D 转换器次之，间接 A/D 转换器的速度最慢，如双斜率积分式 A/D 转换器的转换时间大都在几十毫秒至几百毫秒之间。

[例 10.4] 某微机采集系统要求用一片 A/D 转换芯片在 1s 内对 16 个热电偶的输出电压进行 A/D 转换。已知热电偶输出电压范围为 0～0.025V（对应温度范围 0～450℃），要求分辨的温度为 0.1℃，试问应选择多少位的 A/D 转换器，其转换时间是多少？

解　对于 0～450℃温度范围，信号电压为 0～0.025V，分辨温度为 0.1℃，这相当于 $\frac{0.1}{450}=\frac{1}{4500}$的分辨率。12 位 A/D 转换器的分辨率为 $\frac{1}{2^{12}}=\frac{1}{4096}$，所以必须选用 13 位以上的 A/D 转换器。系统的采样速率为每秒 16 点，每点处理时间为 62.5ms。对于这样慢速的取

样，任何一个 A/D 转换器都可达到。

（4）量程。量程就是 A/D 转换器所能够转换的模拟量输入电压的范围，如 0～5V、0～10V。

（5）输出电平。一般 A/D 转换器输出的二进制数字量电平要求能够与 TTL、CMOS 电平兼容。

（6）输出编码。A/D 转换器的数字量输出编码主要包括二进制和十进制（BCD）。

（7）工作温度。一般工业级环境要求－40～＋85℃。

10.3.2 典型 A/D 转换器芯片 ADC0809

A/D 转换器芯片的种类很多，为便于用户构成多通道数据采集系统，一些厂家将多路模拟开关和 8 位 A/D 转换器集成在一个芯片内，构成多通道 ADC，其中以 NSC 公司的 8 通道 8 位 A/D 转换器 ADC0809 应用最为广泛。下面以常用的 A/D 转换器 ADC0809 为例，介绍它的工作原理和使用方法。

1. ADC0809 的内部结构及工作原理

ADC0809 是 8 位、8 通道逐次逼近式 A/D 转换器，片内有 8 路模拟开关，可以同时连接 8 路模拟量输入信号，单极性，量程为 0～5V。ADC0809 的内部结构如图 10.15 所示。

图 10.15 中，内部由 256R 电阻分压器、树状模拟开关（这两部分组成一个 D/A 变换器）、电压比较器、逐次逼近寄存器、逻辑控制和定时电路组成。其基本工作原理是采用对分搜索方法逐次比较，找出最逼近于输入模拟量的数字量。电阻分压器需外接正负基准电源 $V_{REF}(+)$ 和 $V_{REF}(-)$。CLOCK 端外接时钟信号，A/D 转换器的启动由 START 信号控制，转换结束时控制电路将数字量送入三态输出锁存器锁存，并产生转换结束信号 EOC。

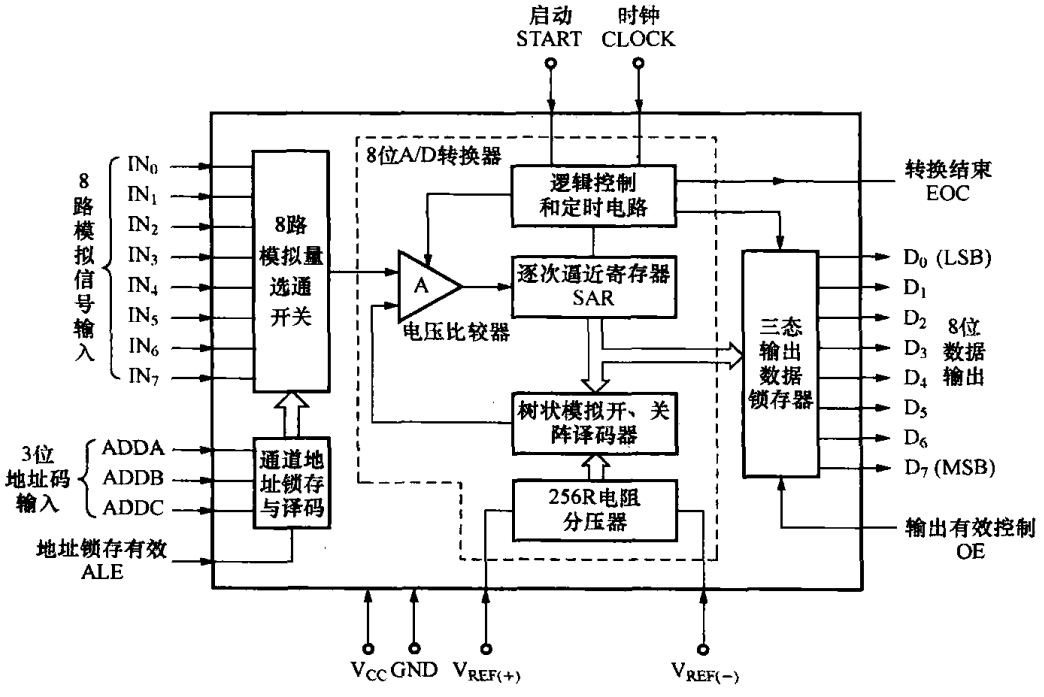


图 10.15 ADC0809 内部结构

2. ADC0809 引脚的功能

ADC0809 引脚排列如图 10.16 所示。各引脚的功能如下：

IN_7～IN_0：8 路模拟量输入端。

$D_7 \sim D_0$：数字量输出端，其中 D_7 为最高有效位 MSB，D_0 为最低有效位 LSB。

START：为 A/D 转换的启动信号，输入，当该引脚加高电平，即开始转换。

EOC：转换结束信号。A/D 转换过程中，EOC 为低电平，当 A/D 转换完成，该引脚输出一高电平，可用作中断请求信号或程序查询信号。

OE：输出允许信号。当转换结束后，由此引脚输入一个高电平信号，打开三态缓冲器输出门，读出数据。

ADDA、ADDB、ADDC：通道选择输入端，该引脚加电平编码为 000～111 时，分别对应于选通通道 $IN_7 \sim IN_0$。

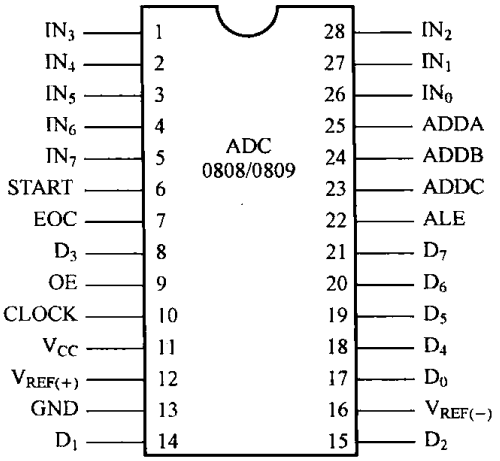


图 10.16　ADC0809 引脚排列图

ALE：通道号锁存控制端。当 ALE 为高电平时，将 ADDA、ADDB、ADDC 三个引脚上的通道号进行锁存，然后由译码器选通 8 路中的相应通道的模拟开关处于闭合状态，进行 A/D 转换。实际使用时，常把 ALE 和 START 连在一起，在 START 端加高电平启动信号的同时，将通道号锁存起来。地址译码与对应通道的关系如表 10.1 所示。

表 10.1　　ADC0809 地址译码模拟量输入通道关系

通道地址	ADDA	ADDB	ADDC	通道地址	ADDA	ADDB	ADDC
IN_0	0	0	0	IN_4	1	0	0
IN_1	0	0	1	IN_5	1	0	1
IN_2	0	1	0	IN_6	1	1	0
IN_3	0	1	1	IN_7	1	1	1

CLK：时钟脉冲输入端，ADC0809 要求时钟脉冲频率不高于 640kHz。一般需要将外部时钟脉冲信号经过分频后接入该输入端。

V_{CC}、$\pm V_{REF}$、GND：V_{CC}和 GND 分别为电源电压输入端和接地（数字地）端，$V_{CC}=+5V$。$\pm V_{REF}$为基准电压输入端，一般可使$+V_{REF}=+5V$，$-V_{REF}$接地。

3. ADC0809 的工作时序

ADC0809 工作时序如图 10.17 所示。对指定的通道采集一个数据的过程如下：

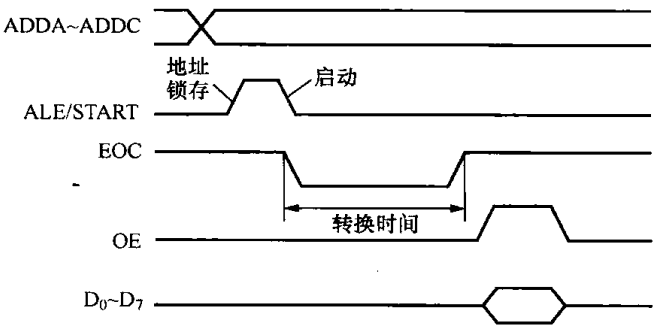


图 10.17　ADC0809 的工作时序图

首先选择当前转换的通道，即将通道号编码送入 ADDA、ADDB、ADDC 端。使 START 和 ALE 端加一正脉冲信号，使该路通道选择码锁存并启动 A/D 转换。可以通过执行 OUT 指令产生负脉冲，经反相后形成正脉冲，也可以由定时电路或可编程定时器提供启动脉冲。转换开始后，EOC 输出信号变低，指示转换正在进

行。经过 63 个时钟周期后，转换结束，数据已保存到 8 位锁存器中，此时 EOC 信号变高。转换结束后，可通过执行 IN 指令，设法在 OE 引脚上形成一个高电平脉冲，打开输出缓冲器的三态门，让转换后的数字量出现在数据总线上，并被读入 CPU 中。

用 ADC 0809 来设计实用的数据采集系统时，除了要考虑采样频率的控制和转换结束检测方法外，还要设计合适的通道选择方法。例如，可用软件延时、定时中断或周期脉冲控制采样频率，以及用延时程序、查询 EOC 电平或用 EOC 正跳变请求中断来判断某个通道转换是否结束。

10.3.3 A/D 转换器与 CPU 的接口应用

通常使用的 A/D 转换芯片一般都具有数据输出、启动转换、转换结束、时钟和参考电平等引脚。ADC 芯片与主机的连接就是处理这些引脚的连接问题。

当 ADC 0809 用来作多通道的数据采集时，需要考虑 ADC 0809 通道地址的选择问题。一种方式是 ADC0808 直接与 CPU 相连，ADDA、ADDB、ADDC 三端直接连接到 CPU 地址总线 A_0、A_1、A_2 三端，则占用的 I/O 口地址多。这样，每一个模拟量输入端对应一个端口地址，8 个模拟量输入则占用 8 个端口地址，对于微机系统外设资源占用太多，如图 10.18 所示。另一种方式是 ADC0809 芯片通过并行接口芯片来控制启动转换和通道选择，如图 10.19 所示。

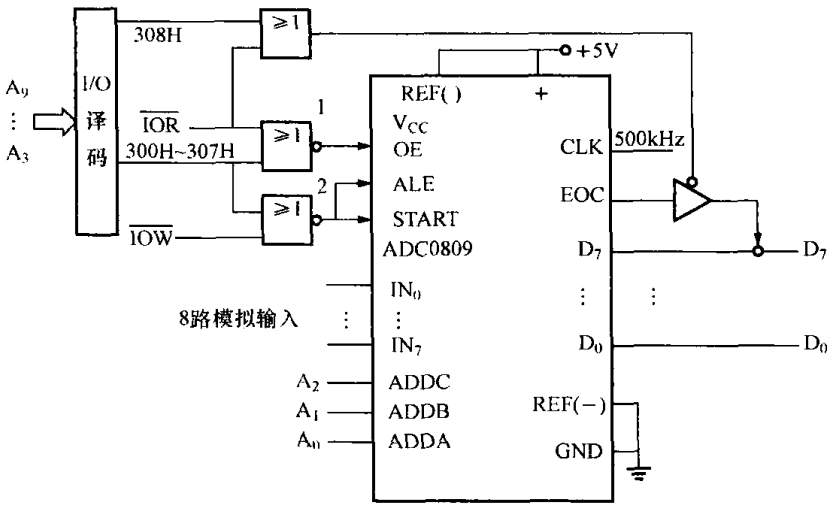


图 10.18 ADC0809 与 CPU 直接连接

对于 A/D 转换器应用，其编程依据仍然是 ADC0809 的工作时序。首先，要给出各通道地址，选择要变换的模拟信号。接着给出通道锁存和启动变换信号。再接下来等待变换结束。当转换结束后，又要使 OE 有效，使 ADC0809 变换好的数据输出。最后是得到变换好的数据并存放在某个地址单元或寄存器中。

1. ADC0809 与 CPU 直接连接

[例 10.5] 要求顺序采样 IN_0～IN_7 8 个输入通道的模拟信号，并将结果依次保存在 ADDBUF 开始的 8 个内存单元中；每隔 100ms 循环采样一次，设 DELAY 为延时 100ms 子程序名。

ADC0809 与系统采用直接连接方法如图 10.19 所示。模拟输入通道 IN_0～IN_7 由 A_0～A_2 决定其通道选择，端口地址为 300H～307H，用来向 0809 输出模拟通道号并锁存；端口

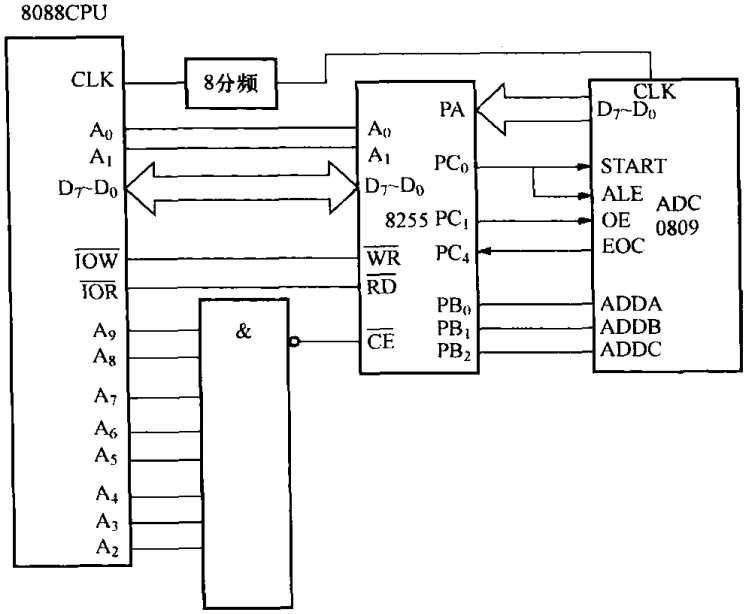


图 10.19　ADC0809 与 8255A 的连接

308H 为查询端口。

A/D 转换程序如下：

```
...
     ADDBUF  DB  8  DUP (?)
...
LOP: MOV  CX, 0008H              ; 通道计数单元，初值=8
     MOV  DI, OFFSET  ADDBUF     ; 转换结果存储区
START: MOV  DX, 300H             ; 取 IN0 启动地址
LOP1:  OUT  DX, AL               ; 启动 A/D 转换，AL 可为任意值
       PUSH  DX                  ; 保存通道地址
      MOV  DX, 308H              ; 取查询 EOC 状态的端口地址
WAIT:  IN  AL, DX                ; 读 EOC 状态
      TEST  AL, 80H              ; 测试 A/D 转换是否结束
      JZ  WAIT                   ; 未结束，则跳到 WAIT 处
      POP  DX                    ; 取读 A/D 转换结果寄存器的端口地址
      IN  AL, DX                 ; 读 A/D 转换结果
      MOV  [DI], AL              ; 保存转换结果
      INC  DI                    ; 指向下一保存单元
      INC  DX                    ; 指向下一个模拟通道
      LOOP  LOP1                 ; 未完，转入下一通道采样
      CALL  DELAY                ; 延时 100ms
      JMP  LOP                   ; 进行下一次循环采样
```

2. ADC0809 通过并行接口芯片 8255A 与系统连接

ADC0809 通过并行接口芯片 8255A 系统连接，对 8 路模拟量分时进行数据采集，转换结果可采用查询方式或中断方式传送。使用 8255A 的一个端口作输入端口，接收 ADC0809

的转换结果；用一个端口作输出端口，控制 8 个模拟量的通道选择信号，再用一个端口作控制及状态端口。硬件电路的连接如图 10.19 所示。

[例 10.6] 在图 10.19 中，ADC0809 需要外接转换时钟和参考电压。实际应用中转换时钟常将 CPU 的时钟信号经分频得到。将 8255 的 A 口输入方式设定为方式 0，B 口的 $PB_0 \sim PB_2$ 输出作为 8 路模拟通道选择信号，PC_4 输入 ADC0809 的转换结束信号，PC_0 作为启动信号。由于 ADC0809 需要脉冲启动，所以通过软件编程让 PC_0 输出一个正脉冲，OE 信号直接接 PC_1。设 8255 的端口地址为 3FC0H～3FC3H。

解 采集程序如下：

```
    PROCADC  PROC  NEAR
        PUSH  BX
        PUSH  DX
        PUSH  DS
        PUSH  AX
        PUSH  SI
        MOV  DX, SEG  ADATA
        MOV  DS, AX
        MOV  SI, OFFSET  ADATA
        MOV BL, 00H
        MOV BH, 08H
INTI55: MOV  DX, 3FC3H
        MOV  AL, 10011000B        ; 8255 控制字，A 口方式 0，输入，B 口方式 0，输出，
                                  ; C 口高 4 位输入，低 4 位输出
        OUT    DX, AL;
        MOV  DX, 3FC1H
        MOV AL, BL
        OUT    DX, AL             ; 送各通道地址
ACD:    MOV  DX, 3FC2H
        MOV  AL, 00H
        OUT  DX, AL
        MOV  AL, 01H
        OUT  DX, AL               ; 由 PC0 输出脉冲启动变换开始
        NOP
        NOP
WAITS:  IN  AL, DX                ; 读取转换结束状态信号
        AND  AL, 10H              ; 判断变换结束否?
        JZ  WAITS                 ; 未结束等待
        MOV  AL, 02H
        OUT DX, AL                ; 使 OE=1
        MOV DX, 3FC0H
        IN AL, DX                 ; 读 A 口，读取 A/D 转换结果
        MOV [SI], AL
        MOV DX, 3FC2H
```

```
MOV AL, 0
OUT DX, AL
INC SI
INC BL
DEC BH
JNZ  ACD
POP SI
POP AX
POP DS
POP DX
POP BX
RET
PROCADC ENDP
```

上面的采集子程序每调用一次，便顺序对 8 路模拟输入 IN_1～IN_7 进行一次 A/D 转换，并将转换的结果存放在 ADATA 所在的顺序 8 个单元中。

在 ADC0809 的硬件电路设计中，其连接还有其他形式。限于篇幅，在此不再说明。

习　　题

10.1　请画出 DAC0832 与微处理器的接口连接图，并编制产生锯齿波、三角波和梯形的程序。

10.2　画出 DAC0809 与微处理器的接口连接图，在什么情况下宜采用查询方式？什么情况下宜采用中断方式？

10.3　一个完整的微机系统的 A/D 通道和 D/A 通道应包括哪几个环节？它们的功能是什么？

10.4　说明 A/D 转换器的电路及其工作原理。

10.5　求逐次比较式 A/D 转换器在输入电压等于 2、4、1.5、－2.5、－4.5 时输出的二进制编码等于多少（设输入电压范围为－5V～＋5V）？

10.6　试编制一段源程序。要求通过 ADC0809 采用中断方式采集 50 个数据，并画出系统连接图。

10.7　试编制一段源程序。要求采用查询方式，从 ADC0809 转换器的 0～7 通道轮流采集 8 路模拟信号的电压量，并把转换后的数据存入 0100H 开始的单元，并画出连接图。

10.8　利用 ADC0809 和 8255A 接口芯片编制一段采集和显示程序，并画出硬件接线图。要求通过 ADC0809 采集 2 通道转换后的数据，再把采集的数据转换成对应的电压量，通过 8255A 输出到 4 个 LED 显示器显示。（ADC0809 占用地址 04A0H～04A7H，8255A 占用地址 04B0H～04B6H，设输入电压范围为 0V～＋5V。）

10.9　使用 8253 控制 A/D 采样，定义通道 0 作计数脉冲，计数值为 1000，计数器计到 0 向 CPU 发一次中断申请，每接到一次中断申请，CPU 采集一次 A/D 转换结果，并把数据存入 0100H 开始的单元，采集 100 个数据后停止工作。

附　　录

附录 1　美国信息交换标准代码（ASCII）

附表 1　　**美国信息交换标准代码（ASCII）**

低位 $b_3b_2b_1b_0$ \ 高位 $b_6b_5b_4$		0H 000B	1H 001B	2H 010B	3H 011B	4H 100B	5H 101B	6H 110B	7H 111B
0H	0000B	NUL	DLE	SP	0	@	P	、	p
1H	0001B	SOH	DC1	!	1	A	Q	a	q
2H	0010B	STX	DC2	"	2	B	R	b	r
3H	0011B	ETX	DC3	#	3	C	S	c	s
4H	0100B	EOT	DC4	$	4	D	T	d	t
5H	0101B	ENQ	NAK	%	5	E	U	e	u
6H	0110B	NCK	SYN	&	6	F	V	f	v
7H	0111B	BEL	ETB	?	7	G	W	g	w
8H	1000B	BS	CAN	(	8	H	X	h	x
9H	1001B	HT	EM	)	9	I	Y	i	y
AH	1010B	LF	SUB	*	:	J	Z	j	z
BH	1011B	VT	ESC	+	;	K	[	k	{
CH	1100B	FF	FS	,	<	L	\	l	|
DH	1101B	CR	GS		=	M	}	m	]
EH	1110B	SO	RS	.	>	N	↑	n	~
FH	1111B	SI	US	/	?	O	←	o	DEL

附录 2　8086/8088 指令执行时间及简表

附表 2　　**常用指令执行时间**

指　令	指 令 内 容	所需时钟周期数	访问内存次数
MOV	累加器到内存	10（14）	1
	内存到累加器	10（14）	1
	寄存器到寄存器	2	0
	内存到寄存器	8（12）＋EA	1
	寄存器到内存	9（13）＋EA	1
	立即数到寄存器	4	0
	立即数到内存	10（14）＋EA	1

续表

指　令	指 令 内 容	所需时钟周期数	访问内存次数
MOV	寄存器到段寄存器	2	0
	内存到段寄存器	8（12）＋EA	1
	段寄存器到寄存器	2	0
	段寄存器到内存	9（13）＋EA	1
ADD 或 SUB	寄存器到寄存器	3	0
	内存到寄存器	9（13）＋EA	1
	寄存器到内存	16（24）＋EA	2
	立即数到寄存器	4	0
	立即数到内存	17（25）＋EA	2
MUL	累加器乘 8 位寄存器	70～77	0
	累加器乘 16 位寄存器	118～133	0
	累加器和内存字节乘	（76～83）＋EA	1
	累加器和内存字乘	［124（128）～139（143）］＋EA	1
IMUL	累加器乘 8 位寄存器	80～98	0
	累加器乘 16 位寄存器	128～154	0
	累加器和内存字节乘	（86～104）＋EA	1
	累加器和内存字乘	［134（138）～160（164）］＋EA	1
DIV	除数在 8 位寄存器中	80～90	0
	除数在 16 位寄存器中	144～162	0
	除数为 8 位内存数	（86～96）＋EA	1
	除数为 16 位内存数	［150（154）～168（172）］＋EA	1
IDIV	除数在 8 位寄存器中	101～112	0
	除数在 16 位寄存器中	165～184	0
	除数为 8 位内存数	（107～118）＋EA	1
	除数为 16 位内存数	［171（175）～190（194）］＋EA	1
循环和移位	在寄存器中移 1 位	2	
	在寄存器中移若干位	8＋4*位数	
	内存数据移 1 位	15（23）＋EA	
	内存数据移若干位	20（28）＋EA＋4*位数	
JPM	段内/段间直接转移	15	
	段内间接转移	8（12）＋EA	
	段间间接转移	24（32）＋EA	
条件转移	JCXZ	6（不转移）	
		18（转移）	
	其他条件转移指令	4（不转移）	
		16（转移）	

注　1. 表中 EA 表示偏移地址。小括弧内的数为 8088 CPU 时的时钟数。

2. 条件转移指令，若条件满足，执行时间比较长。否则，执行时间就短。

附表 3　　计算偏移地址 EA 所需时间

寻址方式		计算 EA 所需时钟数
直接寻址		6
寄存器间接寻址		5
寄存器相对寻址		9
基址、变址寻址	[BX+SI]，[BX+DI]	7
	[BP+SI]，[BP+DI]	8
基址、变址寻址加相对寻址	[BX+SI+位移量]，[BX+DI+位移量]	11
	[BP+SI+位移量]，[BP+DI+位移量]	12

注　1. 若有段超越，则需再加上两个时钟周期。

2. 寻址方式的介绍参见下面内容。

附表 4　　8086/8088 指令简表

指　令	汇编格式	指令完成的操作
数据传送指令	MOV　dst，src	数据传送
	CBW	字节转换成字
	CWD	字转换成双字
	LAHF	FLAGS 低 8 位送 AH 寄存器
	SAHF	AH 寄存器内容送 FLAGS 低 8 位
	LDS　dst，src	设数据段指针
	LES　dst，src	设附加段指针
	LEA　dst，src	送有效地址
	PUSH　src	将一个字压入栈顶
	POP　dst	将栈顶的一个字弹出
	PUSHF	标志寄存器 FLAGS 的内容压入栈顶
	POPF	栈顶内容弹出到 FLAGS
	XCHG　dst，src	交换
	XLAT　src	表转换
算术运算指令	AAA	加法的 ASCII 调整
	AAD	除法的 ASCII 调整
	AAM	乘法的 ASCII 调整
	AAS	减法的 ASCII 调整
	DAA	加法十进制调整
	DAS	减法十进制调整
	MUL　src	无符号数乘法
	IMUL　src	有符号数乘法
	DIV　src	无符号数除法
	IDIV　src	有符号数除法
	ADD　dst，src	加法

续表

指　令	汇 编 格 式	指令完成的操作
算术运算指令	ADC　dst，src	带进位的加法
	SUB　dst，src	减法
	SBB　dst，src	带借位的减法
	CMP　dst，src	比较
	INC　dst	加 1
	DEC　dst	减 1
	NEG　dst	求补
逻辑运算指令	AND　dst，src	逻辑“与”
	OR　dst，src	逻辑“或”
	XOR　dst，src	逻辑“异或”
	NOT　dst	逻辑“非”
	TEST　dst，src	测试（非破坏性逻辑“与”）
移位指令	RCL　dst，count	通过进位循环左移
	RCR　dst，count	通过进位循环右移
	ROL　dst，count	循环左移
	ROR　dst，count	循环右移
	SHL/SAL　dst，count	逻辑左移/算术左移
	SHR　dst，count	逻辑右移
	SAR　dst，count	算术右移
串操作指令	MOVS/MOVSB/MOVSW dst，src	字符串传送
	CMPS/CMPSB/CMPSW dst，src	字符串比较
	LODS/LODSB/LODSW src	送字节串或字串到累加器
	STOS/STOSB/STOSW dst	存储字节串或字串
	SCAS/SCASB/SCASW dst	字符串扫描
程序控制指令	CALL　dst	调用一个过程（子程序）
	RET［弹出字节数（必须为偶数）］	从过程（子程序）返回
	INT int _ type	软件中断
	INTO	溢出中断
	IRET	从中断返回
	JMP　dst	无条件转移
	JG/JNLE short _ label	大于或不小于等于转移
	JGE/JNL short _ label	大于等于或不小于转移
	JL/JNGE short _ label	小于或不大于等于转移
	JLE/JNG short _ label	小于等于或不大于转移
	JA/JNBE short _ label	高于或不低于等于转移
	JAE/JNB short _ label	高于等于或不低于转移

续表

指　令	汇 编 格 式	指令完成的操作
程序控制指令	JB/JNAE short _ label	低于或不高于等于转移
	JBE/JNA short _ label	低于等于或不高于转移
	JO short _ label	溢出标志为 1 转移（溢出转移）
	JNO short _ label	溢出标志为 0 转移（无溢出转移）
	JS short _ label	符号标志为 1 转移（结果为负转移）
	JNS short _ label	符号标志为 0 转移（结果为正转移）
	JC short _ label	进位标志为 1 转移（有进位转移）
	JNC short _ label	进位标志为 0 转移（无进位转移）
	JZ/JE short _ label	零标志为 1 转移（等于或为 0 转移）
	JNZ/JNE short _ label	零标志为 0 转移（不等于或不为 0 转移）
	JP/JPE short _ label	奇偶标志为 1 转移（结果中有偶数个 1 转移）
	JNP/JPO short _ label	奇偶标志为 0 转移（结果中有奇数个 1 转移）
	JCXZ short _ label	若 CX=0，则转移
	LOOP short _ label	CX≠0，则循环
	LOOPE/LOOPZ short _ label	CX≠0 且 ZF=1，则循环
	LOOPNE/LOOPNZ short _ label	CX≠0 且 ZF=0，则循环
	STC	进位标志置 1
	CLC	进位标志置 0
	CMC	进位标志取反
	STD	方向标志置 1
	CLD	方向标志置 0
	STI	中断标志置 1（允许可屏蔽中断）
	CLI	中断标志置 0（禁止可屏蔽中断）
	ESC	CPU 交权
	HLT	停机
	LOCK	总线封锁
	NOP	无操作
	WAIT	等待至TEST信号有效
输入输出指令	IN　ac，src	从外设接口输入字节或字
	OUT　dst，ac	向外设接口输出字节或字

注　dst　目的操作数，目的串

src　源操作数，目的串

ac　累加器

count　计数值

int _ type　中断类型号

short _ label　短距离标号

附录3　DOS及BIOS功能调用

80386/80486微机系统兼容8086/8088，软件中断可分为三部分：

（1）DOS中断，占用类型号为20H～3FH。目前使用20H～27H和2FH，其余保留。

（2）ROM BIOS中断，占用类型号为10H～1FH。

（3）自由中断，占用类型号40H～FFH，可供系统或应用程序设置开发的中断处理程序用。

1. DOS中断及功能调用

DOS常用的9类中断（20H～27H和2FH）分为DOS专用中断和DOS可调用中断两种。

（1）DOS专用中断：是指INT 22H、INT 23H和INT24H三个中断，属DOS操作时专用，用户不要直接使用。

（2）DOS可调用中断：是指INT 20H、INT 21H、INT 25H、INT 26H、IN27H和INT 2FH六个中断。供用户直接调用，但必须满足一定的入口条件。

1）磁盘读/写中断。INT 25H和INT 26H分别用来读/写磁盘上的若干扇区。调用这两个中断服务程序前，必须设置参数如下：

①AL　置入驱动器号（0＝A，1＝B，…）

②CX　置入读/写的扇区数

③DX　置入起始扇区的逻辑号（0号表示0柱0面1区）

④DS：BX　置入内存起始地址

2）程序退出中断。“INT20H”和“INT 27H”都为程序退出命令。其中“INT 20H”为正常退出命令，当用户程序执行完后，可用该命令退回操作系统；“INT 27H”为驻留退出命令，留下的程序被视为DOS的一部分，不被其他程序覆盖。在其他用户程序中，可通过使用软中断调用这部分驻留程序。

3）假脱机打印文件中断。实现假脱机打印文件命令PRINT用“INT 2FH”。当用户执行其他任务时，打印机仍可打印打印队列中的数据文件。一个打印队列最多容纳32个文件。

4）系统功能调用。“INT 21H”中断是供系统程序和应用程序调用的一个极其重要的中断，含近百个功能子系统。操作系统的内核主要由它构成。调用时不依赖于任何硬件设备，可在任何支持DOS的机器上运行。DOS启动成功后，“INT 21H”的中断向量（中断处理程序首地址）被初始化在中断向量表中向量地址为84H～87H的4个存储单元内。

调用INT 21H的方法如下：先在AH寄存器中放入功能号，再按照该功能调用的要求设置好其他有关寄存器，然后执行“INT 21H”。

附表5～附表13分别列出了程序结束、字符I/O、磁盘控制、文件操作、记录操作、目录操作、日期时间、内存分配、网络共享、其他功能和保留功能共11类系统功能调用的类型号、功能及入/出口参数。

附表 5　　程序结束系统功能调用

功能号	功　能	入口参数	出口参数	DOS 版本
00H	退出用户程序并返回操作状态			1，2，3，6
31H	终止用户程序并驻留在内存	AL＝退出码 DX＝程序长度		2，3，6
4CH	终止当前程序并返回调用程序	AL＝退出码		2，3，6

附表 6　　字符 I/O 系统功能

功能号	功　能	入口参数	出口参数	DOS 版本
01H	带回显的键符输入		AL＝输入字符	1，2，3，6
03H	串行口输入字符		AL＝输入字符	1，2，3，6
06H	直接控制台 I/O	DL＝FF（输入） DL＝字符（输出）	AL＝输入字符	1，2，3，6
07H	直接控制台输入（无回显）		AL＝输入字符	1，2，3，6
08H	无回显的键符输入		AL＝输入字符	1，2，3，6
0AH	字符串缓冲输入	DS：DX＝缓冲区首址		1，2，3，6
0BH	取键盘输入状态		AL＝00 无键入；AL＝FF 有键入	1，2，3，6
0CH	清键盘缓冲区后，输入	AL＝功能号（01、06、07、08 或 0A）		1，2，3，6
02H	字符显示	DL＝输出字符		1，2，3，6
04H	串行口输出字符	DL＝输出字符		1，2，3，6
05H	字符打印	DL＝输出字符		1，2，3，6
09H	字符串显示	DS：DX＝缓冲区首址		1，2，3，6

附表 7　　磁盘控制系统功能调用

功能号	功　能	入口参数	出口参数	DOS 版本
0DH	初始化盘状态			1，2，3，6
0EH	置缺省驱动器代码	DL＝盘号	AL＝系统中盘的数目	1，2，3，6
19H	取缺省驱动器代码		AL＝盘	1，2，3，6
1BH	取缺省盘分配表信息（FAT 表）		DS：BX＝盘类型字节地址 DX＝FAT 表项数 AL＝每簇扇区数 CX＝每扇区字节数	1，2，3，6
1CH	取指定盘分配表信息（FAT 表）	DL＝盘号	DS：BX＝盘类型字节地址 DX＝FAT 表项数 AL＝每簇扇区数 CX＝每扇区字节数	2，3，6

续表

功能号	功　能	入口参数	出　口　参　数	DOS版本
2EH	置写校验状态	DL=0，AL=状态		1，2，3，6
36H	取盘的剩余空间数	DL=盘号	BX=可用簇数	2，3，6
			DX=总簇数	
			CX=每扇区字节数	
			AX=每簇扇区数	
54H	取写校验状态		AL=状态	2，3，6

附表 8　　文件操作系统功能调用

功能号	功　能	入口参数	出　口　参　数	DOS版本
1AH	置磁盘缓冲区	DS：DX=缓冲区首址		1，2，3，6
2FH	取磁盘缓冲区首址	DS：DX=FCB首址	ES：BX=缓冲区首址	2，3，6
0FH	打开文件	DS：DX=FCB首址	AL=00，成功；AL=FF，未找到	1，2，3，6
10H	关闭文件	DS：DX=FCB首址	AL=00，成功；AL=FF，未找到	1，2，3，6
11H	查找第一个匹配文件	DS：DX=FCB首址	AL=00，成功；AL=FF，未找到	1，2，3，6
12H	查找下一个匹配文件	DS：DX=FCB首址	AL=00，成功；AL=FF，未找到	1，2，3，6
13H	删除文件	DS：DX=FCB首址	AL=00，成功；AL=FF，未找到	1，2，3，6
16H	建立文件	DS：DX=FCB首址	AL=00，成功；AL=FF，目录区满	1，2，3，6
17H	文件更名	DS：DX=FCB首址 (DS：DX+17)=新名		1，2，3，6
23H	取文件长度	DS：DX=FCB首址	AL=00，成功 AL=FF，未找到	1，2，3，6
3CH	建立文件	DS：DX=字符串地址 CX=文件属性	AX=文件号	2，3，6
3DH	打开文件	DS：DX=字符串地址 AL=0读，AL=1写 AL=2读/写	AX=文件号	2，3，6
3EH	关闭文件	BX=文件号		2，3，6
41H	删除文件	DS：DX=字符串地址		2，3，6
43H	取或置文件属性	DS：DX=字符串地址 AL=0，取文件属性 AL=1，置CX=属性	CX=文件属性	2，3，6
45H	复制文件号	BX=文件号1	AX=文件号2	2，3，6
46H	强制复制文件号	BX=文件号1 CX=文件号2	AX=文件号2	2，3，6

续表

功能号	功　能	入口参数	出口参数	DOS版本
4EH	查找第一个匹配文件	DS：DX=字符串地址 CX=属性	DTA	2，3，6
4FH	查找下一个匹配文件	DTA	DTA	2，3，6
56H	文件更名	DS：DX=字符串地址 ES：DI=新名地址		2，3，6
57H	置或取文件日期和时间	BX=文件号，AL=0，读 AL=1，写 DX：CX	DX：CX=日期和时间	2，3，6
5AH	建立暂时文件	DS：DX=字符串地址 CX=文件属性	AX=文件号	2，3，6
5BH	建立新文	DS：DX=字符串地址 CX=文件属性	AX=文件号	2，3，6

附表9　　记录操作系统功能调用

功能号	功　能	入口参数	出口参数	DOS版本
14H	顺序读一个记录	DS：DX=FCB首址	AL=00，成功；AL=01，文件结束； AL=03，缓冲不满	1，2，3，6
15H	顺序写一个记录	DS：DX=FCB首址	AL=00，成功；AL=FF，盘满	1，2，3，6
21H	随机读一个记录	DS：DX=FCB首址	AL=00，成功；AL=01，文件结束 AL=03，缓冲不满	1，2，3，6
22H	随机写一个记录	DS：DX=FCB首址	AL=00，成功；AL=FF，盘满	1，2，3，6
24H	置随机记录号	DS：DX=FCB首址		1，2，3，6
27H	随机读若干记录	DS：DX=FCB首址 CX=记录数	AL=00，成功；AL=01，文件结束 AL=03，缓冲不满	1，2，3，6
28H	随机写若干记录	DS：DX=FCB首址 CX=记录数	AL=00，成功 AL=FF，盘满	1，2，3，6
3FH	读文件或设备	BX=文件号	AX=实际读出的字节数；CX=读入字节数；DS：DX=缓冲区首址	2，3，6
40H	写文件或设备	BX=文件号 CX=写盘字节数 DS：DX=缓冲区首址	AX=实际写入的字节数	2，3，6
42H	改变文件读/写指针	BX=文件号 CX：DX=位移量 AL=0，绝对移动； AL=1，相对移动 AL=2，绝对倒移	DX：AX=新的指针位置	2，3，6

附表 10　目录操作系统功能调用

功能号	功　能	入口参数	出口参数	DOS版本
39H	建立一个子目录	DS：DX＝字符串地址	AL＝00，成功	2，3，6
3AH	删除一个子目录	DS：DX＝字符串地址	AL＝00，成功	2，3，6
3BH	改变当前目录	DS：DX＝字符串地址	AL＝00，成功	2，3，6
47H	取当前目录	DL＝盘号；DS：SI＝字符串地址	DS：SI＝字符串地址	2，3，6

附表 11　时间日期系统功能调用

功能号	功　能	入口参数	出口参数	DOS版本
2AH	取日期		CX：DX＝日期	1，2，3，6
2BH	置日期	CX：DX＝日期	AL＝00，成功；AL＝FF，失败	1，2，3，6
2CH	取时间		CX：DX＝时间	1，2，3，6
2DH	置时间	CX：DX＝时间	AL＝00，成功；AL＝FF，失败	1，2，3，6

附表 12　内存分配系统功能调用

功能号	功　能	入口参数	出口参数	DOS版本
48H	分配内存空间	BX＝申请内存数量	AX：0 分配内存首值	2，3，6
49H	释放内存空间	ES＝内存始址	BX＝最大可用内存空间（失败时）	2，3，6
4AH	修改分配的内存空间	ES＝原内存始址 BX＝再申请的数量	BX＝最大可用内存空间（失败时）	2，3，6
58H	取或置内存分配策略	AL＝00H 取当前分配策略码 AL＝01H，置当前分配策略码，BX＝策略码	成功 C＝0 AX＝策略码 00 最先符号 01 最佳符号 02 最后符号 失败 C＝1 且 AX＝错误码 成功 C＝0 失败 C＝1 且 AX＝错误码	2，3，6

附表 13　网络共享、其他功能及保留功能列表

功能分类	功能号	含　义	DOS版本
网络共享	5CH	记录共享或锁定	3，6
	5EH	取网络名或置打印机	3，6
	5FH	网络设置重定向	3，6
其他功能	25H	置中断向量	1，2，3，6
	35H	取中断向量	2，3，6
	26H	建立程序段前缀	1，2，3，6
	29H	分析文件名	2，3，6

续表

功能分类	功能号	含义	DOS版本
其他功能	30H	取DOS版本号	2，3，6
	33H	取或置Ctrl-Break标志	2，3，6
	38H	取或置国别	2，3，6
	44H	设备驱动控制	2，3，6
	4BH	加载执行程序	2，3，6
	4DH	取出口码	2，3，6
	59H	取扩展错误信息	3，6
	62H	取程序前缀地址	3，6
	63H	取扩展字符表地址	2，6
保留功能	18H，1DH，1EH，1FH，20H		1，2，3，6
	32H，34H，37H，50H，51H，52H，53H，55H		2，3，6
	5DH，60H，61H		3，6

2. BIOS 中断及功能调用

在 IBM-PC 机的 ROM 存储区中包含一些基本输入/输出子程序。这些基本输入/输出子程序简称 BIOS。用户可通过软件中断方式直接调用它们。由于 BIOS 提供的字符 I/O 功能直接依赖于硬件，因此调用它们比调用 DOS 字符 I/O 功能速度更快。

（1）键盘输入子程序。调用指令为“INT 16H”，入口参数为功能码，要求送 AH 寄存器。其共有 3 种子功能，如附表 14 所示。

（2）显示程序。调用指令为“INT 10H”；入口参数为功能码，要求送 AH 寄存器，共有 15 种子功能，如附表 15 所示。“INT 10H”的调用程序只限于在 IBM-PC 及兼容机。

附表 14　　BIOS 中键盘输入子程序表

功能号	作用	输出结果
00H	从键盘读入 1 个字符	AL=字符的 ASCII 值；AH=字符的扫描码
01H	判断键盘缓冲区是否有字符	Z=1，无字符；Z=0，有字符，且字符在 AL，AH 中 AL=字符的 ASCII 值；AH=字符的扫描码
02H	读键盘当前的换挡状态	AL=换挡状态码

附表 15　　BIOS 显示输出功能列表

功能名称	功能号	输入参数	输出结果
显示方式控制	00H	AL=0	字符分辨率为 40×25 的单色字符方式
		AL=1	字符分辨率为 40×25 的彩色字符方式
		AL=2	字符分辨率为 80×25 的单色字符方式
		AL=3	字符分辨率为 80×25 的彩色字符方式
		AL=4	字符分辨率为 320×200 的彩色图形方式
		AL=5	字符分辨率为 320×200 的单色图形方式

续表

功能名称	功能号	输入参数	输出结果
显示方式控制	00H	AL=6	字符分辨率为640×200的单色图形方式
	0FH		BH=当前页号，AH=字符列数，AL=显示方式
字符显示	05H	AL=选择的页面	设置显示的页面
	08H	BH=显示页面	在当前光标位置返回，AL=字符。AH=属性
	09H	BH=页面，CX=字符总数，AL=字符，BL=属性	在当前光标位置显示字符及属性
	0AH	BH=页面，CX=字符总数，AL=字符	在当前光标位置仅显示字符
	0EH	AL=字符，BL=前景色	在当前页面按电传打字方式显示字符
图形显示	0CH	DX=行数，CX=列数，AL=彩色值	在指定位置上显示彩色点
	0DH	DX=行数，CX=列数	AL返回指定位置的彩色点
	0BH	BH=调色板色别值，BL=彩色值	设置彩色调色板
光标控制	01H	CH=光标开始行，CL=光标结束行	设置光标位置
	02H	DH=行数，DL=列数，BH=页面	设置光标位置
	03H	BH=页面	返回光标位置
屏幕滚动	06H	AL=行数，BH=空白行属性，CH，CL=左上角行、列，DH，DL=右下角行、列	当前页向上滚动
	07H		当前页向下滚动

(3) 打印输出子程序。调用指令为“INT 17H”，入口参数为功能号，送AH寄存器，共有三个子功能，如附表16所示。

附表16　　BIOS打印输出功能列表

子功能号	入口参数	输出结果
00H	AL=打印字符，DX=打印机号（0—2）	送AL中字符到指定的打印机，并在AH返回打印机状态
01H	DX=打印机号（0—2）	初始化指定的打印机，并在AH返回打印机状态
02H	DX=打印机号（0—2）	在AH返回指定的打印机状态

AH寄存器中返回当前打印机状态位含义如下：

D7	D6	D5	D4	D3	D2	D1	D0
不忙	应答	无纸	联机	I/O错	未用	未用	超时

附录 4 80X86 的浮点指令和操作系统型指令

操作系统型指令见附表 17，FPU 数字处理指令见附表 18。

附表 17 操作系统型指令简表

类 型	指 令 格 式	功 能 提 要
系统寄存器的装入与存储指令	MOV CR_n，OP	将 OP 的 32 位操作数装入 CR_n（n=0，2，3）
	MOV OP，CR_n	将 CR_n 的 32 内容存储到 OP（n=0，2，3）
	MOV DR_n，OP	装入调试寄存器 DR_n～DR_7
	MOV OP，DR_n	存储调试寄存器 TR_n～DR_7
	MOV TR_n，OP	装入测试寄存器 DR_3～DR_7
	MOV OP，TR_n	存储测试寄存器 TR_3～DR_7
	LMSW OP	装入机器状态字（即 CR_0 的低 16 位），目的是与 80286 兼容
	SMSW OP	存储机器状态字，目的是与 80286 兼容
	LGDT/LIDT OP	装入全局/中断描述符表寄存器（GDTR/IDTR）
	SGDT/SIDT OP	存储全局/中断描述符表寄存器（GDTR/IDTR）
	LLDT OP	装局部描述符表寄存器 LDTR
	SLDT OP	存局部描述符表寄存器 LDTR
	LTR OP	装任务寄存器
	STR OP	存任务寄存器
保护属性检查指令	CETS	清除机器状态字中任务转换标志
	LAR，OP1，OP2	装入访问权限
	LSl OP1，OP2	装入段界限
	VERR OP1	验证段的可读性
	VERW OP1	验证段的可写性
	ARPL OP1，OP2	调整选择符的特权级

注 1. 操作系统型指令包括系统寄存器（控制寄存器 CR、调试寄存器 DR、测试寄存器 TR、描述符表寄存器和任务寄存器）装入与存储指令，以及保持属性检查指令（简称保护型指令）两类。这些指令通常只由系统程序员使用，也就是说，只有具有最高特权级（0 级）的程序即操作系统程序才能使用。

2. LLDT、SLDT、LTR 和 STR 四条指令只能在保护方式下使用。

附表 18 FPU 数字处理指令一览表

类 型	指 令 助 记 符	功 能
数据传送指令	FBLD	装压缩 BCD 数入栈（FPU 堆栈）
	FBSTP	装压缩十进制（BCD）并弹出栈
	FILD	装整数入栈
	FIST	整数存储
	FISTP	整数存储并弹出栈
	FLD	装实数入栈

续表

类　型	指令助记符	功　能
数据传送指令	FST	存实数
	FSTP	存实数并弹出栈
	FXCH	交换寄存器内容
算术运算指令	FABS	取绝对值
	FADD/FIADD/FADDP	实数加法/整数加/实数加并弹出
	FCHS	变符号
	FDIV/FIDIV/FEIVP	实数除法/整数除/实数除并弹出
	FDIVR/FIDIVR/FDIVRP	反向实数除法/反向整数除/反向实数除并弹出
	FMUL/FIMUL/FMULP	实数乘法/整数乘/实数乘并弹出
	FPREM	部分余数
	FPREMI	部分余数
	FRNDINT	舍入成整数
	FSCALE	比例
	FSUB/FISU/FSUBP	实数减法/整数减/实数减并海底
	FSUBR/FISUBR/FSUBRP	反向实数减法/反向整数减/反向实数减并弹出
	FSQRT	平方根
比较指令	FXTRACT	析取指数和有效数
	FCOM/FCOMP/FCOMPP	实数比较/比较并弹出/比较并弹出两次
	FICOM/FICOMP	整数比较/比较并弹出
	FTST	测试
	FUCOM/FUCOMP/FUCOMPP	无序实数比较/比较并弹出/比较并弹出两次
	FXAM	检验
超越函数指令	F2XM	计算 2^x-1
	FCOS	余弦
	FPATAN	局部反正切
	FPTAN	局部正切
	FSIN	正弦
	FSINCOS	正弦/余弦
	FYL2X	计算 $Y\cdot\log_2 X$
	FYL2XP1	计算 $Y\cdot\log_2(X+1)$
装常数指令	FLD1	装 1.0
	FLDL2E	装 $\log_2 e$
	FLDL2T	装 $\log_2 10$
	FLDLG2	装 $\lg^2$
	FLDLN2	装 ln2
	FLDP1	装 π
	FLDZ	装 0

续表

类 型	指令助记符	功 能
控制指令	FCLEX/FNCLEX	清除异常（清所有事故标志、中断请示标志和忙标志）
	FDECSTP	堆栈指针减1
	FFREE	释放寄存器
	FINCSTP	堆栈指针增1
	FINIT/FNINIT	初始化处理器
	FLDCW	装控制字
	FLDFNV	装环境
	FNOP	空操作
	FRSTOR	恢复状态
	FSAVE/FNSAVE	存状态
	FSTCW/FNSTCW	存控制字
	FSTENV/FNSTENV	存环境
	FSTSW/FNSTSW	存状态字
	FSTSW AX/FNSTSW AX	把状态字存入AX
	FWAIT	等待

注 FPU数字处理指令是指80486的FPU用以支持高精度、高速度的整数和浮点数运算指令。为了与CPU指令相区别，数字处理指令的机器码均带有ESC前缀，其汇编形式均以F字母起头。CPU遇到此类指令均交由FPU来处理。

参 考 文 献

[1] 姚燕南，薛均义. 微型计算机原理. 4 版. 西安：西安电子科技大学出版社，2004.
[2] 周明德. 微型计算机系统原理及应用. 4 版. 北京：清华大学出版社，2002.
[3] 冯博琴，等. 微型计算机原理与接口技术. 北京：清华大学出版社，2002.
[4] 李伯成. 微型计算机原理及接口技术. 北京：清华大学出版社，2005.
[5] Abel P. IBM PC 汇编语言与程序设计（影印版）. 4 版. 北京：清华大学出版社，2002.
[6] 仇玉章. 32 位微型计算机原理与接口技术. 北京：清华大学出版社，2000.
[7] 吴秀清. 微型计算机原理与接口技术. 合肥：中国科学技术大学出版社，2001.

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):

{
  "filename": "MTYzMuS9jeW+ruacuuWOn+eQhuWPiuaOpeWPo+aKgOacr18xMjMwNzYzMy56aXA=",
  "filename_decoded": "1632\u4f4d\u5fae\u673a\u539f\u7406\u53ca\u63a5\u53e3\u6280\u672f_12307633.zip",
  "filesize": 44538984,
  "md5": "fd7cf339f5cac1e6ca20889d85398886",
  "header_md5": "a789eb9c4625db8eeee77d6e28ab61a7",
  "sha1": "64cf35d54a34c25935a8a43b8aec38978176332f",
  "sha256": "a2c8ba6e190a30d2ea7308fbb1ebca6ee45a231bb046596a0016913148d2a672",
  "crc32": 2974845379,
  "zip_password": "",
  "uncompressed_size": 46801318,
  "pdg_dir_name": "1632\u256c\u2557\u256c\u00f3\u2557\u00b7\u2558\u00a1\u2514\u03c6\u255d\u2591\u255c\u2559\u2510\u250c\u255d\u255d\u2569\u2321_12307633",
  "pdg_main_pages_found": 383,
  "pdg_main_pages_max": 383,
  "total_pages": 390,
  "total_pixels": 2451676416,
  "pdf_generation_missing_pages": false
}