第 1 章　微型计算机基础知识

微型计算机是一种能自动、高速、精确地处理信息的现代化电子设备，具有算术运算和逻辑判断能力，并能通过预先编好的程序来自动完成数据的加工处理，因此，可以说计算机是一种帮助人类从事脑力劳动的工具。对于学习计算机专业的学生来说，了解计算机的发展及应用领域，熟悉计算机的组成结构和信息在计算机中的表示方法十分必要。

1.1　计算机的发展及应用

1.1.1　计算机的发展

自第一台电子数字计算机问世以来，计算机的发展以计算机硬件的逻辑元件为标志，大致经历了由电子管、晶体管、中小规模集成电路到大规模和超大规模集成电路 4 个发展阶段。

1. 电子管计算机

第一代计算机采用电子管作为基本电子元件，主存储器采用磁芯、磁鼓，外存储器采用磁带，程序设计主要采用机器语言和汇编语言，主要应用于科学计算。电子管计算机的主要特点是体积大、功耗大、运算速度每秒只有几千次到几万次、价格昂贵、可靠性差。虽然电子管计算机有很多缺陷，但是它的体系结构和程序设计思想为以后计算机的高速发展奠定了科学基础。第一台电子计算机 ENIAC 是典型代表，于 1946 年 2 月在美国宾夕法尼亚大学诞生，是一台电子数字积分计算机。这台计算机共用了 18 000 多个电子管、1 500 个继电器，重量超过 30 t，占地面积 170 m^2，每小时耗电 140 kW，计算速度为每秒 5 000 次加法运算。

2. 晶体管计算机

第二代计算机采用晶体管作为基本电子元件，主存储器采用磁芯，外存储器开始使用磁盘，并且软件也开始有很大的发展，出现了各种高级语言及编译程序。这个时代计算机软件配置开始出现，一些高级程序设计语言相继问世。如用于科学计算的 FORTRAN，用于商业事务处理的 COBOL，用于符号处理的 LISP 等高级语言。操作系统也初步成型，使计算机的使用方式由手工操作变为自动作业管理。此时，计算机速度明显提高，耗电下降，寿命延长。计算机已发展至用于各种事务处理，并开始用于工业控制。

3. 中小规模集成电路计算机

第三代计算机采用中小规模集成电路作为基本电子元件，用半导体存储器代替磁芯存储器，采用流水线、多道程序和并行处理技术。集成电路计算机的主要特点是体积更小、速度快、精度高、功能强、计算机成本进一步下降。在此期间软件向系列化、多样化发展，并逐渐完善，分时操作系统、会话式语言等多种高级语言已经出现，并且提出了模块化与结构化

程序设计的思想。在发展大型机的同时,“小型计算机”开始出现。计算机品种开始向多样化、系列化发展,应用领域不断扩大。

4.大规模和超大规模集成电路计算机

第四代计算机采用大规模和超大规模集成电路作为主要功能部件。大规模和超大规模集成电路计算机的主要特点是速度更快、集成度更高、软件丰富、有通信功能、软硬件密切配合。在此期间硬件和软件的技术日益完善,计算速度高达每秒千万次甚至亿次以上,计算机结构也开始以分布式处理来组织系统。同时,大型机、中型机、超小型机、计算机网络、智能模拟、软件工程等都有了新的发展。我国在2004年研制的超级计算机曙光4000A的运算能力就已经达到每秒11万亿次,是继美国、日本之后第三个跨越10万亿次的计算机研发和应用的国家。

小知识:

从20世纪80年代开始,日本、美国和欧洲纷纷展开新一代计算机的研制工作,目前新一代计算机有以下几个研究方向:

(1)神经网络计算机——模拟人的大脑思维,可同时并行处理大量实时变化的数据,并引出结论。

(2)生物计算机——运用生物工程技术,用蛋白分子作为芯片,可以使计算机体积更小,存储量更大,智能化更强。

(3)光子计算机——用光作为信息载体,通过对光的处理来完成对信息的处理,可提高运算速度、降低能耗。

(4)量子计算机——一种利用量子力学特有的物理现象(特别是量子干涉)来实现全新的信息处理方式(传统计算机遵循物理定律)。量子计算机利用一种链状分子聚合物的特性来表示开与关的状态,利用激光脉冲来改变分子的状态,使信息沿着聚合物移动,从而进行运算。

新一代计算机与前四代计算机的本质区别为:计算机的主要功能将从信息处理上升为知识处理,使计算机具有人的某些智能,所以又称为人工智能计算机。

1.1.2 计算机的分类

计算机发展的“分代”代表了计算机在时间轴上纵向的发展历程,而“分类”可用来说明计算机横向的发展。计算机种类很多,分类方法也有多种。按照原理不同,可分为模拟计算机和电子数字计算机,根据其用途,可分为通用计算机和专用计算机,等等。目前更常用的一种分类方法是按照运算速度、字长、存储性能等综合指标,将计算机分为巨型机、小巨型机、大型机、小型机、工作站、微型机6类。

1.巨型机

巨型机也称为超级计算机,它是一个国家科技水平、经济实力和军事威力的象征。巨型机速度最快,性能最强,技术最复杂,具有巨大的数值计算能力和信息处理能力,是每个时代计算机高精尖技术的集中代表。目前,巨型计算机字长一般64位,每秒平均执行上百亿次浮点运算,主存容量达100万~400万字以上,其高速数据通道每秒可传送几千万字以上数据,并且具有丰富的系统软件。世界上最快速的巨型机都采用大规模并行处理(massively parallel processing,MPP)技术,且都拥有数百甚至上万个处理器。1983年我国首次自行研

制出了银河-Ⅰ巨型机。此后，我国又自行研制出银河-Ⅲ巨型机，每秒运算可达到 120 亿次。现在，我国正在研制更高性能的巨型机。

2. 小巨型机

小巨型机的特点表现在通用性强、具有很强的综合处理能力、性能覆盖面广等方面，主要应用在公司、银行、政府部门、社会管理机构和制造厂家等，通常人们称小巨型机为企业计算机。小巨型机在未来将被赋予更多的使命，如大型事务处理、企业内部的信息管理与安全保护、科学计算等。

3. 大型机

大型机包括国内常说的大、中型机。其特点是大型、通用，整机运算速度高达 300 750 MIPS，具有很强的处理和管理能力，大型机主要用于大银行、大公司、规模较大的高校和科研院所。在计算机向网络迈进的时代，大型机的生存空间也将逐渐扩大。

4. 小型机

小型机规模小，结构简单，设计周期短，便于及时采用先进工艺。这类机器由于可靠性高，对运行环境要求低，因此易于操作且便于维护。小型机符合部门性的要求，为中小型企事业单位所常用，具有规模小、成本低、维护方便等优点。

5. 工作站

工作站是一种高档微机系统。它具有较高的运算速度，具有大小型机的多任务、多用户功能，且兼具微型机的操作便利和良好的人机界面的特点。它可以连接到多种输入/输出设备，易于连网、处理功能强。其应用领域也已从最初的计算机辅助设备扩展到商业、金融、办公领域，并充当网络服务器的角色。

6. 微型机

微型机又称个人计算机(personal computer，PC)，是日常生活中使用最多、最普遍的计算机，具有价格低廉、性能强、体积小、功耗低等特点。现在微型计算机已进入千家万户，成为人们工作、生活的重要工具。

微型机按字长可分为 8 位、16 位、32 位、64 位机；按组装形式可分为非便携式和便携式微型机，前者称为台式机，后者是一种可移动的微型机，如笔记本电脑和掌上电脑。

1.1.3　微型计算机的性能指标

一台微型计算机功能的强弱或性能的好坏，不是由某项指标来决定的，而是由它的系统结构、指令系统、硬件组成、软件配置等多方面的因素综合决定的。但对于大多数普通用户来说，可以从以下几个指标来大体评价计算机的性能。

1. 运算速度

运算速度是衡量计算机性能的一项重要指标。通常所说的计算机运算速度(平均运算速度)是指每秒钟所能执行的指令条数，一般用“百万条指令/秒”(million instructions per second，MIPS)来描述。同一台计算机，执行不同的运算所需时间可能不同，因而对运算速度的描述常采用不同的方法。常用的有 CPU 时钟频率(主频)、每秒平均执行指令数(IPS)等。微型计算机一般采用主频来描述运算速度，例如，Pentium Ⅲ/800 的主频为 800 MHz，Pentium 4 3.2 G 的主频为 3.2 GHz。一般来说，主频越高，运算速度就越快。

2. 字长

计算机在同一时间内处理的一组二进制数称为计算机的一个“字”，而这组二进制数的位数就是“字长”。在其他指标相同时，字长越大计算机处理数据的速度就越快。早期的微型计算机的字长一般是 8 位和 16 位，目前 Pentium 4 是 64 位。

3. 内存储器的容量

内存储器，简称主存，是 CPU 可以直接访问的存储器。计算机需要执行的程序与需要处理的数据存放在主存中。内存储器容量的大小反映了计算机即时存储信息的能力。随着操作系统的升级，应用软件的不断丰富及其功能的不断扩展，人们对计算机内存储器的容量的需求也不断提高。目前，运行 Windows 7 操作系统至少需要 1 GB 的内存容量，运行 Windows 8则需要 1 GB(32 位)或 2 GB(64 位)的内存容量。内存容量越大，系统功能就越强，能处理的数据量就越大。

4. 外存储器的容量

外存储器的容量通常是指硬盘容量(包括内置硬盘和移动硬盘)。外存储器的容量越大，可存储的信息就越多，可安装的应用软件就越丰富。目前，硬盘容量一般为 500 GB～2 TB，有的甚至已达到 4 TB。

以上只是一些计算机的主要性能指标。除了上述主要性能指标外，微型计算机还有其他一些指标，例如，所配置外围设备的性能指标以及所配置系统软件的情况等。另外，各项指标之间也不是彼此孤立的，在实际应用时，应该把它们综合起来考虑，而且还要遵循“性能价格比”的原则。

1.1.4 微型计算机的应用

20 世纪 90 年代以来，计算机技术作为科技的先导技术之一，得到了飞跃式的发展，超级并行计算机技术、高速网络技术、多媒体技术、人工智能技术等相互渗透，改变了人们使用计算机的方式，从而使计算机几乎渗透到人类生产和生活的各个领域。计算机的应用范围归纳起来主要有以下一些方面。

1. 科学计算

科学计算也称“数值计算”，是指用计算机完成科学研究和工程技术中所提出的数学问题。计算机作为一种计算工具，科学计算是它最早的应用领域，也是计算机最重要的应用之一。在科学技术和工程设计中存在着大量的各类数字计算，如求解几百阶乃至上千阶的线性方程组、大型矩阵运算等。这些问题广泛出现在导弹实验、卫星发射、灾情预测等领域，其特点是数据量大、计算工作复杂。在数学、物理、化学、天文等众多学科的科学研究中，经常遇到许多用传统的计算工具难以完成的数学问题，而使用计算机则只需要几天、几小时甚至几分钟就可以精确地解决。所以，计算机是发展现代尖端科学技术必不可少的重要工具。

2. 数据处理

人类已从工业化社会进入信息化社会，信息已成为非常重要的资源。数据处理也称“信息处理”，是指对数值、字符、文字、声音、图形和图像等各种类型的数据进行收集、存储、分类、加工、排序、检索、打印和传送等工作。数据处理具有数据量大、输入/输出频繁、时间性

强等特点，一般不涉及复杂的数值计算。计算机的应用从数值计算到非数值计算，是计算机发展史上的一个飞跃。据统计，在计算机的所有应用中，数据处理方面的应用，约占全部应用的 3/4 以上。数据处理是现代化管理的基础，广泛地用于情报检索、统计、事务管理、生产管理自动化、决策系统、办公自动化等方面。

3. 过程控制

过程控制也称“实时控制”，是用计算机及时采集数据，按最佳值迅速对控制对象进行自动控制或采用自动调节的过程。利用计算机进行过程控制，不仅大大提高了控制的自动化水平，而且大大提高了控制的及时性和准确性。

过程控制的特点是及时收集并检测数据，按最佳值调节控制对象。在电力、机械制造、化工、冶金、交通等部门采用过程控制，可以提高劳动生产效率、产品质量、自动化水平和控制精确度，减少生产成本，减轻劳动强度。在军事上，可使用计算机对导弹实时控制，根据目标的移动情况修正导弹飞行状态，以准确击中目标。

4. 计算机辅助工程

计算机辅助工程是以计算机为工具，配备专用软件辅助人们完成特定任务的工作，以提高工作效率和工作质量。

计算机辅助制造(computer aided manufacturing，CAM)是指利用计算机通过各种数值控制生产设备，完成产品的加工、装配、检测、包装等生产过程的技术。将 CAM 进一步集成形成了计算机集成制造系统 CIMS，从而实现设计生产自动化。利用 CAM 可提高产品质量，降低成本和劳动强度。

计算机辅助设计(computer aided design，CAD)和计算机辅助制造 CAM 是方便设计人员利用计算机来协助进行最优化设计，制造人员利用计算机进行生产设备的管理、控制和操作的软件。目前，在电子、机械、造船、航空、建筑、化工、电器等方面都有 CAD 和 CAM 的应用，这样可以提高设计质量，缩短设计和生产周期，提高自动化水平。

计算机辅助教学(computer aided instruction，CAI)是利用计算机的功能程序把教学内容变成软件，使得学生可以在计算机上学习，使教学方式更加多样化、形象化，以取得更好的教学效果。

电子设计自动化(electronic design automation，EDA)技术是利用计算机中安装的专用软件和接口设备，用硬件描述语言开发可编程芯片，将软件固化，从而扩充硬件系统的功能，提高系统的可靠性和运行速度。

5. 办公自动化

办公自动化(office automation，OA)指用计算机帮助办公室人员处理日常工作。例如，用计算机进行文字处理、文档管理、资料、图像处理、声音处理和网络通信等。它既属于信息处理的范围，又是目前计算机应用的一个较独立的领域。

6. 数据通信与网络

数据通信主要是利用通信卫星群和光导纤维构成的计算机应用网络，实现信息双向交流，同时利用多媒体技术扩大计算机的应用范围。通信卫星的覆盖面广，光导纤维传输的信息量大，保密性好，它们的优势互补，利用计算机将两者结合起来，可在全球范围内双向传送包括电视图像在内的各种信号，把整个地球网络连接起来，使人们在家里就可以收看世界上任何一家电视台的节目，通过屏幕与远在千里之外的友人“面对面”通话。以计算机为核心

的信息高速公路的发展，人们的生活方式将进一步改变。

7. 人工智能

人工智能(artificial intelligence，AI)是用计算机模拟人类的智能活动，如判断、理解、学习、图像识别、问题求解等。它涉及计算机科学、信息论、仿生学、神经学和心理学等诸多学科。在人工智能中，最具代表性、应用最成功的两个领域是计算机专家系统和机器人。

计算机专家系统是一个具有大量专门知识的计算机程序系统。它总结了某个领域的知识并构建知识库。根据这些知识，系统可以对输入的原始数据进行推理，作出判断和决策，以回答用户的咨询，这是人工智能应用的一个成功的例子。

机器人是人工智能技术的另一个重要应用。目前，世界上有许多机器人工作在各种恶劣环境中，如高温、高辐射、剧毒等。机器人的应用前景非常广阔。现在有很多国家正在研制机器人。

8. 电子商务

电子商务(electronic commerce)，是指利用简单、快捷、低成本的电子通信方式，买卖双方互不谋面地进行各种商贸活动，如网上购物的淘宝网、拍拍网、当当网等。电子商务可通过多种电子通信方式来完成，但目前主要是以 EDI(电子数据交换)和 Internet 来完成的。作为一种新型的商务模式，电子商务具有普遍性、方便性、整体性、安全性、协调性等特征。

9. 娱乐

计算机正在走向家庭，在工作之余人们可以使用计算机欣赏 DVD 影碟、听音乐、玩游戏等。

1.2 微型计算机的组成

一个完整的微型计算机系统由硬件系统和软件系统两大部分组成。硬件是计算机物质基础，硬件系统主要由主机(CPU、主存储器)和外部设备(输入/输出设备、辅存)构成。软件是支持计算机工作的程序，它需要人根据机器的硬件结构将要解决的实际问题预先编制好，并且输入计算机的主存中，软件系统由系统软件和应用软件等组成。硬件和软件是一个有机的整体，必须协同工作才能发挥计算机的作用。

1.2.1 微型计算机的基本结构

20 世纪 40 年代中期，科学家冯·诺依曼大胆地提出了采用二进制作为数字计算机的数制基础的理论。同时，他还提出了计算机组成结构、程序存储和程序设计等思想。人们把冯·诺依曼的这些理论总结为冯·诺依曼体系结构。从 ENIAC 到当前最先进的计算机采用的都是冯·诺依曼体系结构。

冯·诺依曼指出计算机硬件系统应由运算器、控制器、存储器、输入设备和输出设备构成，基本结构框图如图 1-1 所示。根据冯·诺依曼体系结构构成的计算机必须具有如下特征：

- 程序和数据在计算机中以二进制的方式执行。
- 必须能够记忆程序、数据、中间结果及最终运算结果。

- 具有完成各种算术、逻辑运算和数据传送等数据加工处理的能力。
- 指令由操作码和地址码组成。
- 能够根据需要控制程序走向，并能根据指令控制机器的各部件协调操作。

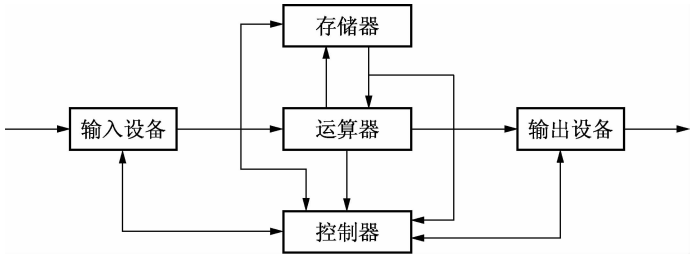


图 1-1　冯·诺依曼型计算机的基本结构

微型计算机系统的组成由小到大可分为微处理器、微型计算机和微型计算机系统，如图 1-2所示。

(1)微处理器(microprocessor)是微型计算机的核心部件，是一个大规模集成电路芯片，其上集成了运算器、控制器、寄存器组和内部总线等部件。微处理器本身不是计算机，常简称为 CPU。

(2)微型计算机(microcomputer)是以微处理器为基础，配以存储器、系统总线及输入/输出接口电路所组成的裸机。它包括微型计算机运行时所需要的硬件支持。

(3)以微型计算机为主体，配上电源系统、输入/输出设备及软件系统就构成了微型计算机系统(microcomputer system)。其中，软件系统包括系统软件和应用软件。系统软件主要包括操作系统、诊断系统、服务程序、汇编程序、语言编译系统等；应用软件也称用户程序，是用户为利用计算机来解决自己的某些问题所编制的程序。

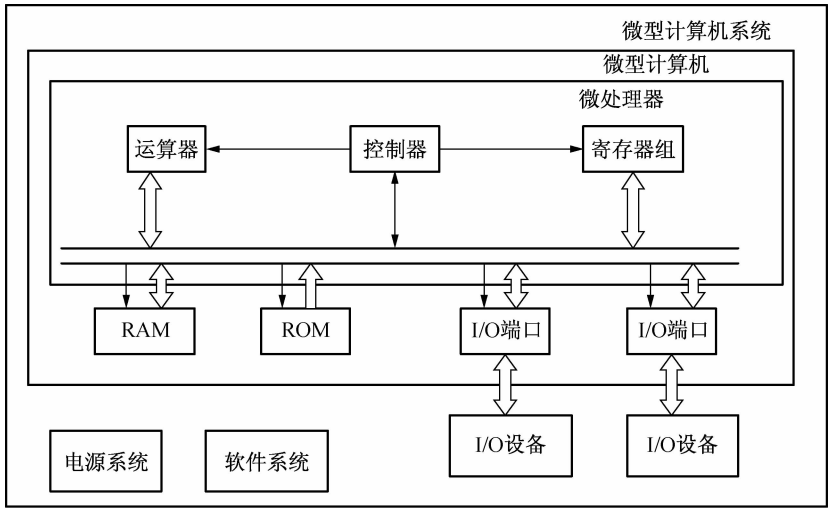


图 1-2　微型计算机系统的层次结构

1.2.2　微型计算机的硬件组成

从大的功能部件来看，微型计算机的硬件主要由微处理器(CPU)、存储器、输入/输出接口

(I/O 接口)与输入/输出设备(I/O 设备)组成,各组成部分通过系统总线联系起来。系统总线是各部件之间传送信息的公共通道,包括地址总线(AB)、数据总线(DB)和控制总线(CB)。

微型计算机的硬件结构如图 1-3 所示。

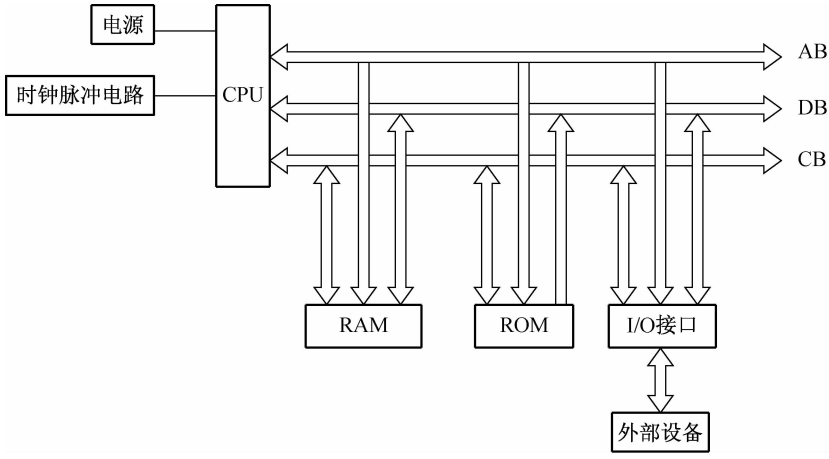


图 1-3 微型计算机的硬件结构框图

1. 微处理器

不同型号的微型计算机,其性能优劣主要取决于其微处理器性能的不同。微处理器由控制器、运算器和寄存器组 3 个主要部分组成。

(1)控制器:控制器是计算机的指挥中心。它的作用是从存储器中取出指令,然后分析指令,发出由该指令规定的一系列操作命令,完成指令的功能。控制器主要由程序计数器(PC)、指令寄存器(IR)、指令译码器(ID)、时序信号发生器等部件构成。控制器是计算机的关键部件,它的功能直接关系到计算机的性能。

(2)运算器:运算器又称算术逻辑单元(arithmetic and logical unit,ALU),是用二进制进行算术运算和逻辑运算的部件。它以加法运算为核心,可以完成加、减、乘、除四则运算和各种逻辑运算,新型 CPU 的运算器还可以完成各种浮点运算。运算器的功能和速度对计算机来说至关重要。

(3)寄存器组:寄存器组是 CPU 内部的若干个存储单元,用来存放参加运算的二进制数据以及保存运算结果。一般可分为通用寄存器和专用寄存器,通用寄存器可供程序员编程使用,专用寄存器的作用是固定的,如堆栈指针、标志寄存器等。

2. 存储器

这里讲的存储器是指内存储器(内存),它用来存放计算机的指令和数据。存储器以单元为单位线性编址,CPU 按其地址读/写单元,通常一个单元可存放 8 位二进制数(即一个字节)。计算机程序只有存放到内存中才能被执行。内存可分为随机存储器(random access memory,RAM)和只读存储器(read only memory,ROM) 两大类。本书第 5 章将对存储器作详细介绍。

3. 输入/输出接口与输入/输出设备

输入/输出设备(简称 I/O 设备)是计算机与外界联系的设备, 简称为外设。计算机通过外设获得各种外界信息,并且通过外设输出运算处理结果。常用的输入设备有键盘、鼠

标、扫描仪等，常用的输出设备有显示器、打印机、绘图仪等。

微型计算机与 I/O 设备间的信息交换不能直接进行，必须通过输入/输出接口（I/O 接口）将两者连接起来，I/O 接口实质上是将外设连接到计算机总线上的一组逻辑电路的总称。

4. 系统总线

总线是一组信号线的集合，是在计算机系统各部件之间传输地址、数据和控制信息的公共通路，从物理结构来看，它由一组导线和相关的控制、驱动电路组成。

微型计算机采用了总线结构，CPU 通过总线实现读取指令，并通过它与内存、外设进行数据交换。

在 CPU、存储器、I/O 接口之间传输信息的总线称为“系统总线”。系统总线包括：

（1）地址总线（address bus，AB）。它是单向总线，用于传送 CPU 发出的地址信息，以指明与 CPU 交换信息的内存单元或 I/O 设备。

（2）数据总线（data bus，DB）。它是双向的，用于 CPU 与内存或外设之间进行数据交换时传输数据信息。

（3）控制总线（control bus，CB）。控制总线用于传送控制信号、时序信号和状态信号等。CPU 向内存或外设发出的控制信息，内存或外设向 CPU 发出的状态信息均可通过它来传送。可见作为一个整体而言，CB 是双向的，而对 CB 中的每一根数据线来说，它是单向的。

1.2.3　微型计算机的软件系统

软件是计算机系统中使用的各种程序，而软件系统是指挥整个计算机硬件系统工作的程序集合。软件系统所包含的内容非常丰富，因而对其分类也较为困难。图 1-4 给出了现代计算机系统的软件分类，整个软件系统按其功能分为系统软件和应用软件两大类。

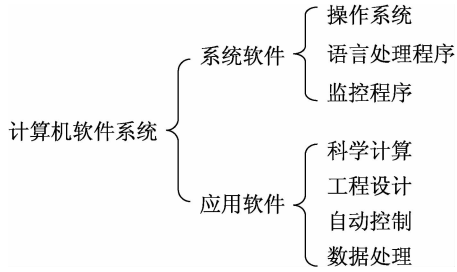


图 1-4　计算机系统的软件分类

系统软件主要功能是对整个计算机系统进行调度、管理、监视及服务等。它能够使系统的各种资源得到合理的调度和高效的使用，并能监视系统的运行状态，一旦出现故障就能自动保护现场信息使之不受破坏，并诊断出故障部位。它还可以帮助用户调试程序、查找程序中的错误等。

应用软件是指除了系统软件以外的所有软件，由各种应用软件包和面向问题的各种应用程序组成。应用软件是为用户提供在各个具体应用领域中的辅助功能，它也是绝大多数用户学习、使用计算机时最感兴趣的内容。如计算机辅助绘图软件 AutoCAD、办公软件 Office、图形图像处理软件 Photoshop、网络下载软件网际快车和迅雷等。

1.3 计算机中数据信息的表示方法

计算机只能处理二进制的 0 和 1，因此必须将数值、字符、声音、图形和图像、视频等各种信息转换成为计算机能够处理的二进制数据。计算机可处理的数值、文字、图形、声音、视频，甚至各种模拟信息量等数据，在计算机系统内部，主要表示成定点数（整数）、浮点数（实数）、逻辑数（布尔数）、字符、字符串等形式，并且都必须采用数字化编码。

1.3.1 进位计数制及其转换

数制即表示数的方法，按进位的原则进行计数的数制称为进位计数制，简称“进制”。通常是以十进制来进行计算，另外，还有二进制、八进制和十六进制等。在计算机的数制中，要掌握 3 个概念，即数码、基数和位权。

1. 数码

数码是指一个数制中表示基本数值大小的不同数字的计数符号。例如：

(1)十进制(decimal notation)有 10 个计数符号：0、1、2、3、4、5、6、7、8、9。

(2)二进制(binary notation)有两个计数符号：0 和 1。

(3)八进制(octal notation)有 8 个计数符号：0、1、2、3、4、5、6、7。

(4)十六进制(hexadecimal notation)有 16 个计数符号：0～9，A，B，C，D，E，F，其中 A～F 对应十进制的 10～15。

2. 基数

基数是指一个数制所使用数码的个数。例如：

(1)十进制的基数为 10。

(2)八进制的基数为 8。

(3)二进制的基数为 2。

(4)十六进制的基数为 16。

3. 位权

位权是指一个数制中某一位上的 1 所表示数值的大小。例如：

(1)对于任意一个 n 位整数和 m 位小数的十进制数，均可按权展开为

$$D=D_{n-1}\cdot 10^{n-1}+\cdots+D_1\cdot 10^1+D_0\cdot 10^0+D_{-1}\cdot 10^{-1}+\cdots+D_{-m}\cdot 10^{-m}$$

(2)对于任意一个 n 位整数和 m 位小数的二进制数，均可按权展开为

$$D=B_{n-1}\cdot 2^{n-1}+B_{n-2}\cdot 2^{n-2}+\cdots+B_1\cdot 2^1+B_0\cdot 2^0+B_{-1}\cdot 2^{-1}+\cdots+B_{-m}\cdot 2^{-m}$$

(3)对于任意一个 n 位整数和 m 位小数的八进制数，均可按权展开为

$$D=O_{n-1}\cdot 8^{n-1}+\cdots+O_1\cdot 8^1+O_0\cdot 8^0+O_{-1}\cdot 8^{-1}+\cdots+O_{-m}\cdot 8^{-m}$$

(4)对于任意一个 n 位整数和 m 位小数的十六进制数，均可按权展开为

$$D=H_{n-1}\cdot 16^{n-1}+\cdots+H_1\cdot 16^1+H_0\cdot 16^0+H_{-1}\cdot 16^{-1}+\cdots+H_{-m}\cdot 16^{-m}$$

4. 进制之间的转换

不同数进制之间进行转换应遵循转换原则。转换原则是：两个有理数如果相等，则有理

数的整数部分和分数部分一定分别相等。

1)二进制、八进制、十六进制数转换为十进制数

按权相乘相加,即各数位与相应位权值相乘以后再相加即为对应的十进制数。

(1)二进制数转换成十进制数。将二进制数转换成十进制数,只要将二进制数用计数制通用形式表示出来,计算出结果,便得到相应的十进制数。例如:

$$\begin{aligned}(11001.1)_2&=1\times2^4+1\times2^3+0\times2^2+0\times2^1+1\times2^0+1\times2^{-1}\\&=16+8+0+0+1+0.5\\&=(25.5)_{10}\end{aligned}$$

(2)八进制数转换为十进制数。八进制数转换成十进制数,将八进制数以 8 为基数按权展开并相加。例如:

$$\begin{aligned}(305.13)_8&=3\times8^2+0\times8^1+5\times8^0+1\times8^{-1}+3\times8^{-2}\\&=192+0+5+0.125+0.046\ 875\\&=(197.171\ 875)_{10}\end{aligned}$$

(3)十六进制数转换为十进制数。十六进制数转换成十进制数,将十六进制数以 16 为基数按权展开并相加。例如:

$$\begin{aligned}(10B.E5)_{16}&=1\times16^2+0\times16^1+11\times16^0+14\times16^{-1}+5\times16^{-2}\\&=256+0+11+0.875+0.019\ 531\ 25\\&=(267.894\ 531\ 25)_{10}\end{aligned}$$

2)十进制数转换为二进制数

(1)整数部分的转换。整数部分的转换采用的是除 2 取余法。其转换原则是:将该十进制数除以 2,得到一个商和余数(K_0),再将商除以 2,又得到一个新商和余数(K_1),如此反复,得到的商是 0 时得到余数(K_{n-1}),然后将所得到的各位余数,以最后余数为最高位,最初余数为最低位依次排列,即 $K_{n-1}K_{n-2}\cdots K_1K_0$,这就是该十进制数对应的二进制数。这种方法又称为“倒序法”。

(2)小数部分的转换。小数部分的转换采用乘 2 取整法。其转换原则是:将十进制数的小数乘以 2,取乘积中的整数部分作为相应二进制数小数点后最高位 K_{-1},反复乘 2,逐次得到 K_{-2}、K_{-3}、…、K_{-m},直到乘积的小数部分为 0 或 1 的位数达到精确度要求为止。然后把每次乘积的整数部分由上而下依次排列起来($K_{-1}K_{-2}\cdots K_{-m}$),即是所求的二进制数。这种方法又称为“顺序法”。

例如,将$(75.453)_{10}$转换成二进制数(取 4 位小数)。

整数部分

		取余数	
2	75	1	低
2	37	1	↑
2	18	0	
2	9	1	
2	4	0	
2	2	0	
2	1	1	高
	0		

小数部分

	取整数	
0.453		
× 2		
0.906	0	高
× 2		↓
1.812	1	
× 2		
1.624	1	
× 2		
1.248	1	低

结果:$(75.453)_{10}=(1001011.0111)_2$。

3)八进制数与二进制数之间的转换

(1)八进制转换为二进制数。八进制数转换成二进制数所使用的转换原则是“一位拆三位”,即把一位八进制数对应于三位二进制数,然后按顺序连接即可。例如:

$$(751.42)_8=(\underset{7}{111}\ \underset{5}{101}\ \underset{1.}{001.}\ \underset{4}{100}\ \underset{2}{010})_2$$

结果:$(751.42)_8=(111101001.100010)_2$。

(2)二进制数转换成八进制数。二进制数转换成八进制数可概括为“三位并一位”,即从小数点开始向左右两边以每三位为一组,不足三位时补 0,然后每组改成等值的一位八进制数即可。例如:将二进制数 11011111101.110011 转换为八进制数。

$$(\underset{3}{011}\ \underset{3}{011}\ \underset{7}{111}\ \underset{5.}{101.}\ \underset{6}{110}\ \underset{3}{011})_2=(3375.63)_8$$

结果:$(11011111101.110011)_2=(3375.63)_8$。

4)十六进制数与二进制数之间的转换

(1)二进制数转换成十六进制数。二进制数转换成十六进制数的转换原则是“四位并一位”,即以小数点为界,整数部分从右向左每 4 位为一组,若最后一组不足 4 位,则在最高位前面添 0 补足 4 位,然后从左边第一组起,将每组中的二进制数按权数相加得到对应的十六进制数,并依次写出即可;小数部分从左向右每 4 位为一组,最后一组不足 4 位时,尾部用 0 补足 4 位,然后按顺序写出每组二进制数对应的十六进制数。例如:将二进制数 11001101101.1100101 转换为十六进制数。

$$(\underset{6}{0110}\ \underset{6}{0110}\ \underset{D.}{1101.}\ \underset{C}{1100}\ \underset{A}{1010})_2=(66D.CA)_{16}$$

结果:$(11001101101.1100101)_2=(66D.CA)_{16}$。

(2)十六进制数转换成二进制数。十六进制数转换成二进制数的转换原则是“一位拆四位”,即把 1 位十六进制数写成对应的 4 位二进制数,然后按顺序连接即可。例如:

$$(76E.F5)_{16}=(\underset{7}{0111}\ \underset{6}{0110}\ \underset{E.}{1110.}\ \underset{F}{1111}\ \underset{5}{0101})_2$$

结果:$(76E.F5)_{16}=(011101101110.11110101)_2$。

1.3.2 带符号数的表示

在计算机中参加运算的数有正负之分,通常在计算机中用 $X=X_0X_1X_2\cdots X_{n-1}$ 来表示一个二进制数,并规定当 $X_0=0$ 时 X 为正数,$X_0=1$ 时 X 为负数。机器数可以有不同的表示方法。对有符号数,机器数常用的表示方法有原码、反码和补码。

1.原码

上述机器数表示方法,即最高位表示符号、数值位用二进制绝对值表示的方法,便为原码表示方法。可见,原码的定义可表示为:

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

$[X]_{原}$所能表示数的范围为:$[-(2^{n-1}-1),(2^{n-1}-1)]$,它对应于原码的 111…1~011…1。

数 0 的原码有两种不同形式:

$$[+0]_{原}=000\cdots0$$

$$[-0]_{原}=100\cdots0$$

原码表示简单、直观，与真值间转换方便，但用它作加减法运算不方便，而且 0 有+0 和−0 两种表示方法。

2. 反码

正数的反码表示与原码相同；负数的反码是将其对应的正数各位（连同符号位）取反得到，或将其原码除符号位各位取反得到。可见，反码的定义可表示为：

$$[X]_{反}=\begin{cases}X & 0\leqslant X<2^{n-1}\\ 2^{n-1}-1-|X| & -2^{n-1}<X\leqslant 0\end{cases}$$

$[X]_{反}$所能表示数的范围为：$[-(2^{n-1}-1),(2^{n-1}-1)]$，它对应于反码的 100…0～011…1。

例如，$[+3]_{反}=00000011$（设为 8 位），$[-3]_{反}=11111100$（设为 8 位）。

数 0 的反码也是两种形式：

$$[+0]_{反}=000\cdots0(全\ 0)$$

$$[-0]_{反}=111\cdots1(全\ 1)$$

将反码还原为真值的方法是：反码→原码→真值，即$[X]_{原}=[[X]_{反}]_{反}$。或者说，当反码的最高位为 0 时，后面的二进制序列值即为真值，且为正；最高位为 1 时，则为负数，后面的数值位要按位求反才为真值。

3. 补码

正数的补码表示与原码相同；负数的补码是将对应的正数各位（连同符号位）取反加 1（最低位加 1）而得到的，或将其原码除符号位外各位取反加 1 而得到。可见，补码的定义可表示为：

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

$[X]_{补}$所能表示数的范围为：$[-2^{n-1},(2^{n-1}-1)]$，它对应于补码的 100…0～011…1。

例如，$[+3]_{补}=00000011$（设为 8 位），$[-3]_{补}=11111101$（设为 8 位）。

数 0 的补码只有一个：

$$[+0]_{补}=[-0]_{补}=000\cdots0(全\ 0)$$

将补码还原为真值的方法是：补码→原码→真值，而$[X]_{原}=[[X]_{补}]_{补}$。或者说，若补码的符号位为 0，则其后的数值的值即为真值，且为正；若符号位为 1，则应将其后的数值位按位取反加 1，所得结果才是真值，且为负。

1.3.3　定点表示与浮点表示

在计算机中涉及小数点位置时，有定点和浮点两种表示方法。顾名思义，定点表示就是小数点在数中的位置是固定不变的；而浮点则是表示小数点的位置是浮动的。

1. 定点表示

任何一个二进制数都可以表示成一个纯整数或纯小数与一个 2 的正数次幂的乘积的形式：

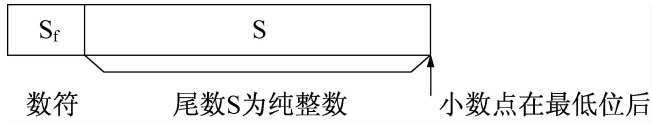
$$N=2^{P}\times S$$

其中：S 表示全部有效数字，称为 N 的尾数；P 称为 N 的阶码，它指明了小数点的位置；2 称

为阶码的底。P 和 S 均为用二进制表示的数。

1)定点整数

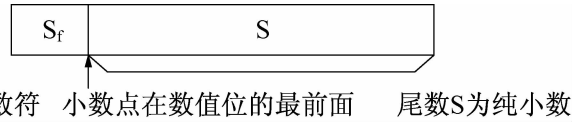
当 P=0 且尾数为纯整数时,定点数只能表示整数。形式为:



通常数符 0 表示正数,1 表示负数,尾数常以原码表示。

2)定点小数

当 P=0 且尾数为纯小数时,定点数只能表示纯小数。形式为:



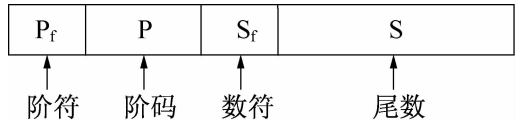
例如:

+0.1010101	在机内表示为	0 1010101
−0.1010101	在机内表示为	1 1010101

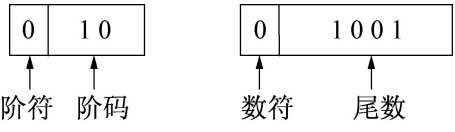
在机器中小数点不占位置,因而不能区分是定点小数还是定点整数。小数点位置是由程序员预先约定好的。

2.浮点表示

如果阶码 P 不为 0,且可以在一定范围内取值,这样的数称为浮点数。形式为:

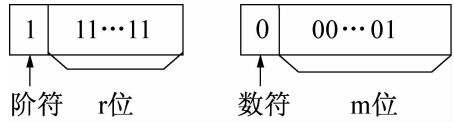


阶码常用补码表示法,尾数常为原码表示的纯小数。浮点数的格式、字长因机器而异。对于一个浮点数,如 $N=2^{10}\times0.1001$,它在计算机中的表示形式为:

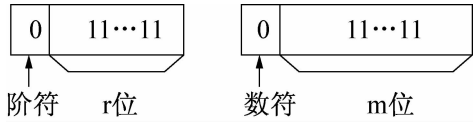


由于在浮点法中,小数点的位置不是固定的,因此在要求数值不变的情况下,小数点往左移一位,相应阶码加 1。

在浮点数法中,若用 r 位表示阶码,m 位表示尾数,以正数为例,浮点法所能表示的最小数为:



最大数则为:



所以浮点法表示的正数范围为：

$$2^{-(2^r-1)}\times 2^{-m}\leqslant N\leqslant 2^{(2^r-1)}\times(1-2^{-m})$$

若表示数的总位数(r+m+2)不变，则分给阶码的位数越多，表示数的范围就越大。但由于表示尾数的位数减少，故使表示数的有效位数减少，这也是不希望的，通常要求阶码的位数所表示的数码的最大值等于或略大于尾数的位数。一般对于位数相同的计算机，浮点法能表示的范围比定点法大。

1.4　算术运算和逻辑运算基础

在计算机中可进行两种运算：算术运算和逻辑运算。

1.4.1　补码加减运算

算术运算时，参与运算的二进制数码表示的是数值大小。常见的算术运算有加、减、乘、除、乘方、开方等。一般计算机中都提供了加、减、乘、除指令，其他更复杂的算术运算要利用算术变换成基本的四则运算来实现。从硬件实现的角度看，各种算术运算的基础是加、减运算。对于采用补码运算的计算机，加法运算又是基础的基础。

补码的加减法运算规则：

$$[X\pm Y]_{补}=[X]_{补}+[\pm Y]_{补}$$

其中，X、Y 为正、负数均可。该式说明，无论加法还是减法运算，都可由补码的加法运算实现，运算结果(和或差)也以补码表示。若运算结果不产生溢出，且最高位(符号位)为 0，则表示结果为正数；最高位为 1，则结果为负数。

1. 补码加法

运算公式为：

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$

根据补码定义，分 4 种情况可以证明该式的正确性。

(1)X>0，Y>0，则(X+Y)>0。由补码定义$[X]_{补}=X$，$[Y]_{补}=Y$，得$[X]_{补}+[Y]_{补}=X+Y=[X+Y]_{补}$。

(2)X<0，Y<0，则(X+Y)<0。由补码定义$[X]_{补}=M+X$，$[Y]_{补}=M+Y\ (\mathrm{mod}\ M)$，得$[X]_{补}+[Y]_{补}=M+X+M+Y=M+M+X+Y=M+[X+Y]_{补}=[X+Y]_{补}(\mathrm{mod}\ M)$。

(3)X>0，Y<0。由补码定义$[X]_{补}=X$，$[Y]_{补}=M+Y\ (\mathrm{mod}\ M)$，得$[X]_{补}+[Y]_{补}=X+M+Y=M+X+Y$。有两种情况：

- 当(X+Y)≥0 时，M 被丢掉，因此$[X]_{补}+[Y]_{补}=X+Y=[X+Y]_{补}(\mathrm{mod}\ M)$。
- 当(X+Y)<0 时，有$[X]_{补}+[Y]_{补}=M+X+Y=[X+Y]_{补}(\mathrm{mod}\ M)$。

(4)X<0，Y>0。情况与(3)类似，只需将 X，Y 位置对调即可证明。

2. 补码减法

运算公式为：

$$[X]_{补}-[Y]_{补}=[X-Y]_{补}$$

通过$[Y]_{补}$求得$[-Y]_{补}$，可以将减法运算转化为补码的加法运算，运算公式为：

$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

已知$[Y]_{补}$求$[-Y]_{补}$的法则是:对$[Y]_{补}$各位(包括符号位)取反,末位加1,就可以得到$[-Y]_{补}$。例如:

- $[Y]_{补}=1.1011$,则$[-Y]_{补}=0.0101$
- $[Y]_{补}=0.1011$,则$[-Y]_{补}=1.0101$

根据以上讲述,补码减法运算法则如下:

(1)参加运算操作的数都用补码表示。

(2)数据的符号与数据一样参加运算。

(3)求差时将负减数求补,用求和代替求差,将减法运算转化为补码的加法运算。

(4)运算结果为补码。如果符号位为0,表明运算结果为正;如果符号位为1,则表明结果为负。

(5)符号位的进位为模值,应该去掉。

例如,设X=33,Y=45,用补码求X+Y和X-Y(设X、Y为8位)。

计算过程如下:

$$[X]_{补}=00100001$$
$$[Y]_{补}=00101101,\ [-Y]_{补}=11010011$$
$$[X+Y]_{补}=[X]_{补}+[Y]_{补}=01001110$$
$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}=11110100$$

3.溢出判断

运算过程中,运算结果超出了数的表示范围的现象称为溢出。若运算结果为正,绝对值超过表示范围时,称为正溢;运算结果为负,绝对值超过表示范围时,称为负溢。

(1)对于加法,只有正数加正数和负数加负数两种情况下才会产生溢出,符号不同的两个数相加不会产生溢出。

(2)对于减法,只有正数减负数和负数减正数两种情况下才会产生溢出,符号相同的两个数相减不会产生溢出。

因此,在判断溢出时,可以根据参加运算的两个数据和结果的符号进行。判断溢出常采用的方法之一是用双符号位进行运算。

从上述补码运算规则和举例可看出,用补码表示计算机中的有符号数优点明显。第一,负数的补码对应正数的补码之间的转换可用同一方法——求补运算实现,因而可简化硬件;第二,可将减法变为加法运算,从而省去减法器电路;第三,有符号数和无符号数的加法运算可用同一加法器电路完成,结果都是正确的。

1.4.2 逻辑运算

计算机的逻辑运算主要是与、或、非、异或等运算。

1.“与”运算(AND)

“与”运算又称逻辑乘,用符号“·”或“∧”来表示。运算规则如下:

$$0\wedge0=0 \quad 0\wedge1=0 \quad 1\wedge0=0 \quad 1\wedge1=1$$

即当两个参与运算的数中有一个数为0,则运算结果为0,都为1结果为1。

2.“或”运算(OR)

“或”运算又称逻辑加,用符号“＋”或“∨”表示。运算规则如下:

$$0 \vee 0=0 \quad 0 \vee 1=1 \quad 1 \vee 0=1 \quad 1 \vee 1=1$$

即当两个参与运算的数中有一个数为 1,则运算结果为 1,都为 0 结果为 0。

3.“非”运算(NOT)

非运算 NOT 由符号“～”表示。如果数值为 1,则它的非运算结果为 0;如果数值为 0,则它的非运算结果为 1。

4.“异或”运算(XOR)

异或运算通常用符号“⊕”表示。“异或”运算的规则如下:

$$0 \oplus 0=0 \quad 0 \oplus 1=1 \quad 1 \oplus 0=1 \quad 1 \oplus 1=0$$

即当两个参与运算的数取值相异时,运算结果为 1,否则为 0。

1.4.3　移位运算

移位运算包含有逻辑移位、算术移位和循环移位 3 种类型。

1.逻辑移位运算

1)逻辑左移

逻辑左移是把数据向左逐位移动 N 次,每次逐位移动后,最低位用 0 来补充,最高位丢失。例如:

11110000 逻辑左移 1 次(N＝1)后,结果为:11100000。

2)逻辑右移

逻辑右移是把数据向右逐位移动 N 次,每次逐位移动后,最高位用 0 来补充,最低位丢失。例如:

11110000 逻辑右移 1 次(N＝1)后,结果为:01111000。

2.算术移位运算

1)算术左移

算术左移是把数据向左逐位移动 N 次,每次逐位移动后,最高位还用原来的符号位来补充,最低位用 0 来补充。例如:

11110000 算术左移 1 次(N＝1)后,结果为:11100000。

2)算术右移

算术右移是把数据向右逐位移动 N 次,每次逐位移动后,最高位还用原来的符号位来补充,最低位丢失。例如:

11110000 算术右移 1 次(N＝1)后,结果为:11111000。

3.循环移位运算

1)循环左移

循环左移是把指定的寄存器或存储器操作数向左循环移动所指定的次数,每左移一次,把最高位移入操作数最低位。例如:

11110000 循环左移 1 次(N＝1)后,结果为:11100001。

2)循环右移

循环右移是把指定的寄存器或存储器操作数向右循环移动所指定的次数,每右移一次,把最低位移入操作数最高位。例如:

11110000 循环右移 1 次(N=1)后,结果为:01111000。

本 章 小 结

本章介绍了计算机的基础知识,包括计算机的发展过程、计算机的应用领域和性能指标、计算机系统的组成、不同进制之间数据转换、数据在计算机中的表示以及算术逻辑运算等知识。

本章重点是掌握计算机硬件系统和软件系统的组成以及数制转换、数据在计算机中的表示等内容。学习过程中应注重基础知识的理解和掌握,为后续章节的学习打下基础。

习 题 1

1.计算机的发展经历了几个时期? 每个时期的特点是什么?

2.计算机的性能指标是什么?

3.微型计算机的硬件组成有哪些?

4.把下列二进制数转换为十进制数。

(1)1001 (2)10101001 (3)1110001011

(4)10111111 (5)100000000011 (6)101000111

5.把下列十进制数转换为二进制数。

(1)231 (2)1765 (3)505

(4)639 (5)1024 (6)61

6.把下列二进制数转换为八进制数和十六进制数。

(1)10110111 (2)110001 (3)10110001

(4)1111111111 (5)10110000 (6)100111.1101

第 2 章　微 处 理 器

计算机必须有一个控制并执行指令的部件，该部件不仅要与计算机的其他功能部件进行信息交换，还要控制它们的操作，所以该部件被称为微处理器或 CPU。本章以 8086 为例主要介绍微处理器的工作原理和系统组成。

2.1　8086 微处理器

8086 微处理器是 Intel 系列的 16 位微处理器，它采用 HMOS 工艺制造，双列直插式封装，有 40 个引脚。8086 微处理器的电源为 5 V，主时钟频率为 5～10 MHz。它的外部数据总线为 16 位，地址总线为 20 根。因为可用 20 位地址，所以可寻址的地址空间达 1 MB。

8086 微处理器在内部采用并行流水线结构，这种结构可以提高微处理器的利用率和处理速度。

2.1.1　8086 微处理器的结构

8086 微处理器的内部结构框图如图 2-1 所示。

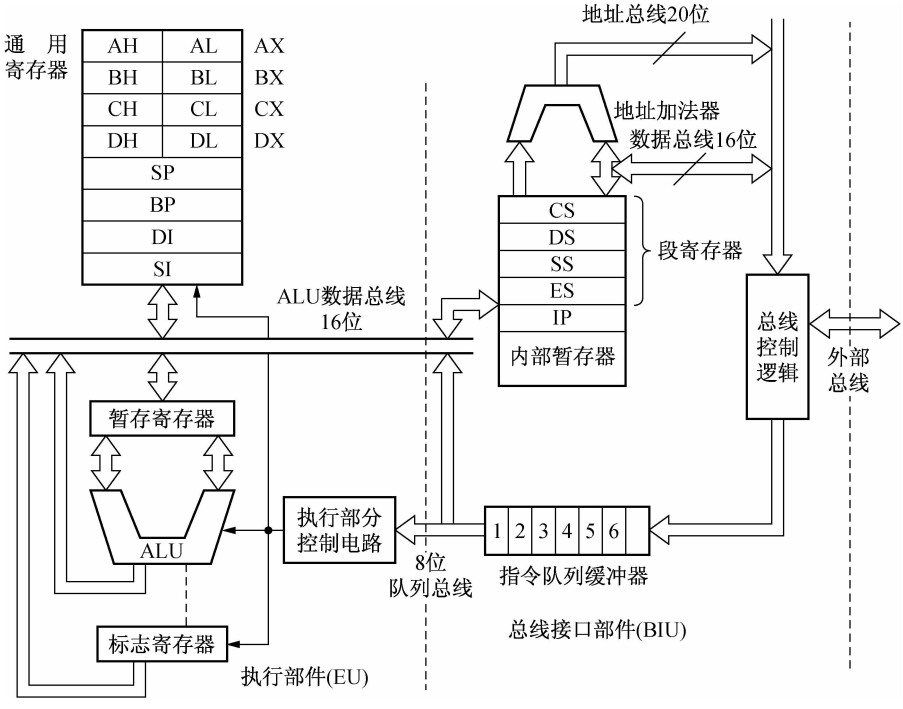


图 2-1　8086 微处理器的内部结构

从功能上讲,8086 分为两部分,即总线接口部件(bus interface unit,BIU)和执行部件(execution unit,EU)。

1. 总线接口部件 BIU

总线接口部件是 CPU 与外部存储器及 I/O 的接口,负责与存储器和 I/O 系统进行数据交换。其组成部分如下:

(1)4 个 16 位段地址寄存器,即代码段寄存器 CS、数据段寄存器 DS、附加段寄存器 ES 和堆栈段寄存器 SS,它们分别用于存放当前代码段、数据段、附加数据段和堆栈段的段基址。

(2)16 位指令指针寄存器 IP:IP 用于存放下一条要执行指令的有效地址 EA(即偏移地址),IP 的内容由 BIU 自动修改,通常是进行加 1 修改。当执行转移指令、调用指令时,BIU 装入 IP 中的是要转移的目的地址。

(3)地址加法器:加法器用于将逻辑地址变换成读/写存储器所需的 20 位物理地址,即完成地址加法操作。方法是将某一段寄存器的内容(代表段基值)左移 4 位(相当于乘 16)再加上 16 位偏移地址以形成 20 位物理地址。

(4)6 字节的指令队列:当执行部件 EU 正在执行指令,且不需要占用总线时,BIU 会自动进行预取下一条或几条指令的操作,并按先后次序存入指令队列中排队,由 EU 按顺序取来执行。

(5)总线控制逻辑:总线控制逻辑用于产生并发出总线控制信号,以实现对存储器和I/O端口的读/写控制。它将 CPU 的内部总线与 16 位的外部总线相连,是 CPU 与外部数据交换(读/写操作)必不可少的路径。

2. 执行部件 EU

执行部件的功能就是负责指令的执行。它包括以下几个部分:

(1)算术逻辑单元 ALU:ALU 完成 16 位或 8 位的二进制数的算术逻辑运算,绝大部分指令的执行都由 ALU 完成。在运算时数据先传送至 16 位的暂存寄存器中,经 ALU 处理后,运算结果可通过内部总线送入通用寄存器或由 BIU 存入存储器。

(2)标志寄存器:它用来反映 CPU 最近一次运算结果的状态特征或存放控制标志。标志寄存器为 16 位,其中 7 位暂时未用。

(3)通用寄存器组:它包括 4 个数据寄存器 AX、BX、CX、DX,其中 AX 又称累加器,4 个指针寄存器,即堆栈指针寄存器 SP、基址指针寄存器 BP、目的变址寄存器 DI 和源变址寄存器 SI。

(4)EU 控制器:它接收从 BIU 中指令队列取来的指令,经过指令译码形成各种定时控制信号,向 EU 内各功能部件发送相应的控制命令,以完成每条指令所规定的操作。

2.1.2 8086 的寄存器

1. 通用寄存器

8086 的通用寄存器是 16 位的,同时都兼容使用 8 位寄存器。通用寄存器除了可以存放通用数据以外,还有自己的特定功能,具体如下:

(1)AX(accumulator):累加器,主要用于算术运算中存放操作数或运算结果,另外,所有对 I/O 操作的指令都使用该寄存器。

(2)BX(base):基址寄存器,用于存放操作数的偏移地址。

(3)CX(count):计数寄存器,用于移位指令、循环指令及串操作指令的自动计数控制。

(4)DX(data):数据寄存器,在算术乘除运算中,与 AX 组合在一起存放双字节数据;在对 I/O 操作的指令中,用于存放 I/O 端口地址。

(5)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。

(6)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。

(7)BP(base pointer):基址指针寄存器,用于存放堆栈中操作数的偏移地址。

以上寄存器均是 16 位的,可以存放一个字型数据。其中前 4 个 16 位寄存器还可以分别拆成两个 8 位寄存器来使用,分别为 AH、AL、BH、BL、CH、CL、DH、DL。

2. 专用寄存器

8086 及以前的机器使用的专用寄存器包括 IP、SP 和 FR 三个 16 位寄存器。

IP(instruction pointer):指令指针寄存器,专门用于存放下一条将要执行的指令所在的偏移地址,它与段寄存器 CS 配合,用来确定程序下一条指令在内存中的位置。计算机指令的执行就是依靠 IP 寄存器来控制指令序列的执行流程。

SP(stack pointer):堆栈指针寄存器,记载栈顶的当前位置(偏移地址)。

FR(flags register):标志寄存器,又称为程序状态字(program status word,PSW),用来存放条件标志位、控制标志位及系统标志位等,每一位都有各自的功能,8086/8088 使用了其中的 9 位,用于存放当前程序的执行状况(控制标志位)和运算结果的特征(条件标志位),各标志位的分布情况如图 2-2 所示。

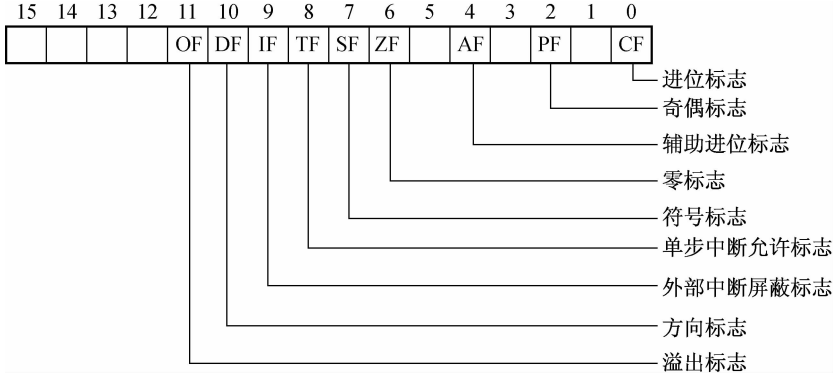


图 2-2　标志寄存器中各标志位的分布情况

1)条件标志位

(1)CF(carry flag):进位标志,记录加法运算的进位或减法运算的借位情况,有进位或借位时置 1,否则置 0。

(2)PF(parity flag):奇偶标志,记录字节型数值运算结果中 1 的个数情况,1 的个数为奇数时置 1,否则置 0。

(3)AF(auxiliary carry flag):辅助进位标志,记录加/减运算中低 4 位向高位的进/借位情况,有进/借位置 1,无置 0。

(4)ZF(zero flag):零标志,记录运算结果是否为 0,为 0 时置 1,否则置 0。

(5)SF(sign flag):符号标志,记录结果的正负情况,为负置 1,为正置 0。

(6)OF(overflow flag):溢出标志,记录结果是否超出带符号数的表示范围,超出置 1,否则置 0。

2)控制标志位

(1)TF(trap flag):单步中断允许标志,又称为跟踪标志或陷阱标志,表示系统当前是否允许单步中断,允许时置 1,此时每条指令执行完后产生中断,由系统控制计算机,不允许时置 0,此时系统正常运行,不产生中断。

(2)IF(interrupt flag):外部中断屏蔽标志,表示系统当前是否允许可屏蔽外部中断,允许时置 1,CPU 响应可屏蔽中断的请求,不允许时置 0。

(3)DF(direction flag):方向标志,表示串操作按减量方向还是增量方向进行,当按减量方向进行时,每次操作后变址寄存器 SI 和 DI 自动减 1,DF 置 1,按增量方向则置 0。

3.段寄存器

段寄存器也是一组专用寄存器,8086 有 4 个 16 位段寄存器,用于存放逻辑地址中的段值部分,各个段寄存器的名称和代号如下:

(1)CS(code segment):代码段寄存器,存放当前正在运行程序的指令代码。

(2)DS(data segment):数据段寄存器,存放当前运行程序所用的数据。

(3)ES(extra segment):附加段寄存器,辅助数据存储区域,也作为串操作中目的操作数的存放地。

(4)SS(stack segment):堆栈段寄存器,作为堆栈使用的内存区域,是一个比较特殊的、以"先进后出"方式进行访问的数据存储区域。

2.1.3 8086 系统存储器的组织

8086 系统在实模式下允许的最大寻址空间是 1 MB,8086 工作在实模式时,其地址宽度为 20 位,采用存储器地址分段的方法来解决如何给 16 位字长机器提供 20 位地址的问题。

1.存储器地址分段

将 1 MB 的存储空间分成若干个段(segment),每个程序由一个或多个段组成,段的个数与大小是由程序本身决定,每个段中存储的代码或数据可以存放在段内任意单元中。但是,这些段的起始单元不是任意的,而是有所限制。机器规定:存储器地址从 0 开始,每 16 个字节为一小节(paragraph)。例如:

第 1 小节:00000H,00001H,…,0000EH,0000FH

第 2 小节:00010H,00011H,…,0001EH,0001FH

第 3 小节:00020H,00021H,…,0002EH,0002FH

⋮

第 65535 小节:FFFE0H,FFFE1H,…,FFFEEH,FFFEFH

第 65536 小节:FFFF0H,FFFF1H,…,FFFFEH,FFFFFH

上述每个小节的起始地址末位为 0,可以作为段的起始地址,即段的起始地址从某一小节开始,称为段基址(segment base address),取其前 16 位为段基址,末位 0 可在求物理地址时通过左移得到。

程序在访问内存时,需要用两部分来指出内存地址:"段基址"和"段内偏移(offset)地

址”。其中，段基址保存在某个段寄存器中，段内偏移地址表示内存单元相对于段起始位置的位移(displacement)，即距离，简称偏移地址，也叫有效地址(effective address，EA)。这种分段地址常记作“段基址：偏移地址”，这种地址称为逻辑地址(logical address)。内存单元的实地址称为物理地址(physical address)，存储器与 CPU 之间的任何信息交换都是使用物理地址的。逻辑地址是程序员在编写程序中使用的，物理地址是由系统自动转换的。

2. 逻辑地址与物理地址的计算

在实模式下工作的 80x86 CPU，段地址与偏移地址都是 16 位，因此，段的大小不能超过 2^{16} B＝64 KB 连续单元。系统采用下列方法将逻辑地址转换为 20 位的物理地址：

物理地址＝段基值×16＋偏移地址

即将段基值左移 4 位加上偏移地址得到物理地址(系统自动完成)。例如，已知段基值是 7856H，段内偏移地址是 4321H，则物理地址＝78560H＋4321H＝7C881H，转换过程如图 2-3所示。

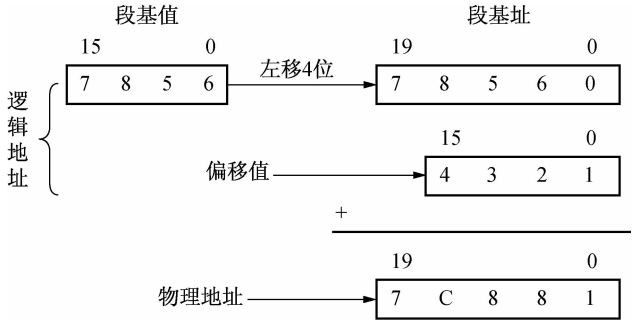


图 2-3　逻辑地址到物理地址的转换

在内存中段与段之间的关系可以是间隔的、相邻的、部分重叠或全部重叠，因此不同的逻辑地址可以得到同一物理地址。

显然，每个存储单元只有唯一物理地址，但它却可由不同的段基址和不同的偏移地址组成。在程序的执行过程中，CPU 根据不同操作数的类型访问存储器，逻辑地址的来源和操作数的类型密切相关。表 2-1 给出了不同操作类型获得的段基址和偏移量的不同来源。

表 2-1　逻辑地址计算的来源

操作类型	逻辑地址		
	段 基 址		偏 移 量
	隐含来源	允许替代来源	
取指令	CS	无	IP
堆栈操作	SS	无	SP
取源串	DS	CS、SS、ES	SI
存目的串	ES	无	DI
以 BP 为基址	SS	CS、SS、ES	EA
存取一般量	DS	CS、SS、ES	EA

2.1.4 8086/8088 微处理器的引脚功能

8086/8088 CPU 的 40 条引脚信号按功能可分为地址总线、数据总线、控制总线和其他(时钟与电源)4 类,如图 2-4 所示。

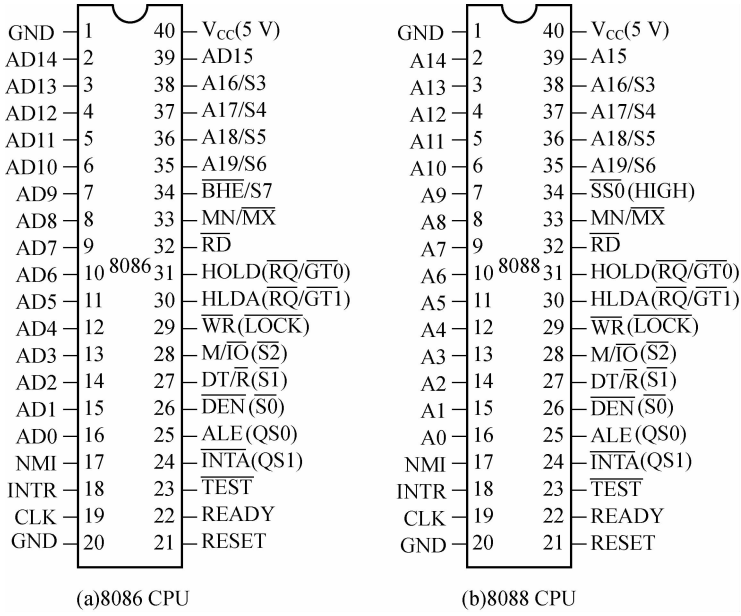


图 2-4 8086/8088 CPU 的引脚图

(1) GND, V_{CC}(输入):GND 为接地端,V_{CC} 为电源端。8086 CPU 采用的电源为(5±10%) V。

(2)AD15~AD0(address/data bus)地址/数据复用总线(双向、三态):CPU 访问一次存储器或 I/O 端口称完成一次总线操作,或执行一次总线周期。一个总线周期通常包括 T_1、T_2、T_3、T_4,在每个状态 CPU 将发出不同的信号。

AD15~AD0 作为复用引脚,在总线周期的 T_1 状态,CPU 在这些引脚上输出要访问的存储器或 I/O 端口的地址。在 T_2~T_3 状态,如果是读周期,则处于浮空(高阻)状态,如果是写周期,则传送数据。在中断响应及系统总线处于"保持响应"周期时,AD15~AD0 都被置为高阻状态。

(3)A19/S6~A16/S3(address/status)地址/状态复用线(输出、三态):这 4 根引脚也是分时复用引脚。在总线周期的 T_1 状态,用来输出地址的最高 4 位,在总线周期的其他状态(T_2,T_3 和 T_4 状态),用来输出状态信息。

S6 总是为 0,表示 8086 CPU 当前与总线相连。

S5 表明中断屏蔽标志的当前设置。如果 IF=1,则 S5=1,表示当前允许可屏蔽中断;如果 IF=0,则 S5=0,表示当前禁止一切可屏蔽中断。

S4 和 S3 状态的组合指出当前正使用哪个段寄存器,具体规定如表 2-2 所示。

当系统总线处于"保持响应"周期时,A19/S6~A16/S3 被置为高阻状态。

表 2-2　S4 和 S3 的组合及对应含义

S4	S3	当前正使用的段寄存器
0	0	附加数据段寄存器 ES
0	1	堆栈段寄存器 SS
1	0	代码段寄存器 CS 或未使用任何段寄存器
1	1	数据段寄存器 DS

(4)$\overline{\text{BHE}}$/S7(bus high enable/status)高 8 位数据总线允许/状态复用引脚(输出、三态):这是 8086 CPU 上的一个复用信号,低电平表示高 8 位数据总线有效。在总线周期的 T_1 状态,8086 在$\overline{\text{BHE}}$/S7 脚输出低电平,表示高 8 位数据总线 AD15～AD8 上的数据有效;若$\overline{\text{BHE}}$/S7 脚输出高电平,表示仅在低 8 位数据总线 AD7～AD0 上传送数据。在总线周期的 T_2、T_3、T_4 状态,$\overline{\text{BHE}}$/S7 引脚输出状态信号,但在 8086 芯片设计中,没有赋予 S7 实际意义。在“保持响应”周期,$\overline{\text{BHE}}$/S7 引脚被置成高阻状态。

(5)NMI(non-maskable interrupt)非屏蔽中断输入信号(输入):该信号边沿触发,上升沿有效。此类中断请求不受中断屏蔽标志 IF 的控制,也不能用软件进行屏蔽。只要该引脚上出现由低到高的变化,就会在当前指令结束后引起中断。NMI 中断经常由电源掉电等紧急情况引起。

(6)INTR(interrupt request)可屏蔽中断请求信号(输入):高电平有效。当 INTR 信号变为高电平时,表示外部设备有中断请求,CPU 在每个指令周期的最后一个 T 状态检测此引脚,一旦测得此引脚为高电平,并且中断屏蔽标志位 IF=1,则 CPU 在当前指令周期结束后,即转入中断响应周期。

(7)CLK(clock)时钟信号(输入):提供了 CPU 和总线控制的基本定时脉冲。8086 CPU 要求时钟信号是非对称性的,占空比为 33%。它由时钟发生器产生。8086 CPU 的时钟频率有以下几种:8086 为5 MHz,8086-1 为 10 MHz,8086-2 为 8 MHz。

(8)$\overline{\text{RD}}$(read)读信号(输出、三态):低电平有效,表示 CPU 正在对存储器或 I/O 端口进行读操作,具体是对存储器读,还是对 I/O 端口读,取决于 M/$\overline{\text{IO}}$信号。在读总线周期的 T_2、T_3 状态,$\overline{\text{RD}}$均保持低电平,在“保持响应”周期它被置成高阻状态。

(9)RESET(reset)复位信号(输入):高电平有效,至少要保持 4 个时钟周期的高电平,才能停止 CPU 的现行操作,完成内部的复位过程。在复位状态,CPU 内部的寄存器初始化,除 CS=FFFFH 外,包括 IP 在内的其余各寄存器的值均为 0,故复位后将从 FFFFH:0000H的逻辑地址,即物理地址 FFFF0H 处开始执行程序。一般在该地址内放置一条转移指令,以转到程序真正的入口地址。

当复位信号变为低电平时,CPU 重新启动执行程序。

(10)READY(ready)准备就绪信号(输入):这是一个用来使 CPU 和低速的存储器或 I/O设备之间实现速度匹配的信号。当 READY 为高电平时,表示内存或 I/O 设备已准备就绪,可以立即进行一次数据传输。CPU 在每个总线周期的 T_3 状态对 READY 引脚进行检测,若检测到 READY=1,则总线周期按正常时序进行读/写操作,不需要插入等待状态 T_w。若测得 READY=0,则表示存储器或 I/O 设备工作速度慢,没有准备好数据,则 CPU 在 T_3 和 T_4 之间自动插入一个或几个等待状态 T_w 来延长总线周期,直至检测到 READY 信号为高电平后,才使 CPU 退出等待进入 T_4 状态,完成数据传送。

插入一个 T_w 后的总线周期为 5 个 T 状态：T_1、T_2、T_3、T_w、T_4。

(11)$\overline{TEST}$(test)测试信号(输入)：与等待指令 WAIT 配合使用。当 CPU 执行 WAIT 指令时，CPU 处于空转等待状态，它每 5 个时钟周期检测一次$\overline{TEST}$引脚。当测得$\overline{TEST}$为高电平时，则 CPU 继续处于空转等待状态；当$\overline{TEST}$变为低电平后，就会退出等待状态，继续执行下一条指令。$\overline{TEST}$信号用于多处理器系统中，实现 8086 主 CPU 与协处理器(8087 或 8089)间的同步协调功能。

(12)MN/$\overline{MX}$(minimum/maximum mode control)模式控制信号(输入)：由该引脚选择工作在最大或最小模式。当此引脚接＋5 V(高电平)时，CPU 工作于最小模式；若接地(即为低电平时)，CPU 工作于最大模式。

以上 12 类共 32 个引脚是 8086 CPU 工作在最小模式和最大模式(后续章节会详细讲述)时都要用到的信号，是公共引脚信号。还有 8 个引脚信号(第 24～31 号引脚)在不同模式下有不同的名称和定义，是双功能引脚。

(13)M/$\overline{IO}$(memory/input and output)存储器或输入、输出操作选择信号(输出、三态)：这是 CPU 工作时会自动产生的输出信号，用来区分 CPU 当前是访问存储器还是访问端口。当 M/$\overline{IO}$为高电平时，表示 CPU 当前访问存储器；当 M/$\overline{IO}$为低电平时，表示当前 CPU 访问 I/O端口。M/$\overline{IO}$信号一般在前一个总线周期的 T_4 状态就可以产生有效电平，在新总线周期中，M/$\overline{IO}$一直保持有效直至本周期的 T_4 状态为止。

在 DMA(存储器直接存取)方式时，M/$\overline{IO}$为高阻状态。

(14)$\overline{DEN}$(data enable)数据允许信号(输出、三态)：作为双向数据总线收发器 8286/8287的选通信号。它在每一次存储器访问或 I/O 访问或中断响应周期有效。此信号只用于最小模式。

在 DMA 方式时，此引脚为高阻状态。

(15)DT/$\overline{R}$(data transmit/receive)数据发送/接收控制信号(输出、三态)：在使用 8286/8287作为数据总线收发器时，8286/8287 的数据传送方向由 DT/$\overline{R}$控制。DT/$\overline{R}$＝1 时，数据发送；DT/$\overline{R}$＝0 时，数据接收。

在 DMA 方式时，DT/$\overline{R}$为高阻状态。此信号只用于最小模式。

(16)$\overline{WR}$(write)写信号(三态、输出)：在最小模式下作为写信号，$\overline{WR}$信号表示 CPU 当前正在对存储器或 I/O 端口进行写操作，由 M/$\overline{IO}$来区分是写存储器还是写 I/O 端口。对任何总线“写”周期，$\overline{WR}$只在 T_2、T_3、T_w 期间有效。

在 DMA 方式时，$\overline{WR}$被置成高阻状态。

(17)$\overline{INTA}$(interrupt acknowledge)中断响应信号(输出、三态)：在最小模式下，$\overline{INTA}$是 CPU 响应可屏蔽中断后发给请求中断的设备的回答信号，是对中断请求信号 INTR 的响应。CPU 的中断响应周期共占据两个连续的总线周期，在中断响应的每个总线周期的 T_2、T_3 和 T_w 期间，$\overline{INTA}$引脚变为有效低电平。第一个$\overline{INTA}$负脉冲通知申请中断的外设，其中断请求已得到 CPU 响应；第二个负脉冲用来作为读取中断类型码的选通信号。外设接口利用这个信号向数据总线上送中断类型码。

(18)ALE(address latch enable)地址锁存允许信号(输出)：在最小模式下，作为 8086 CPU 提供给地址锁存器 8282/8283 的控制信号，在任何一个总线周期的 T_1 状态，ALE 输出有效电平(实际是一个正脉冲)，以表示当前地址/数据、地址/状态复用总线上输出的是地址

信息，并利用它的下降沿将地址锁存到锁存器。ALE 信号不能浮空。

(19)HOLD(hold request)总线保持请求信号(输入)：该信号是最小模式系统中除主 CPU(8086/8088)以外的其他总线控制器(如 DMA 控制器)申请使用系统总线的请求信号。

(20)HLDA(hold acknowledge)总线保持响应信号(输出)：该信号是对 HOLD 的响应信号。当 CPU 测得总线请求信号 HOLD 引脚为高电平，如果 CPU 又允许让出总线，则在当前总线周期结束时，T_4 状态期间发出 HLDA 信号，表示 CPU 放弃对总线的控制权，并立即使 3 条总线(地址总线、数据总线、控制总线，即所有的三态线)都置为高阻状态，表示让出总线使用权。申请使用总线的控制器在收到 HLDA 信号后，就获得了总线控制权。在此后的一段时间内，HOLD 和 HLDA 均保持高电平。当获得总线使用权的其他控制器用完总线后，使 HOLD 信号变为低电平，表示放弃对总线的控制权。8086 CPU 检测到 HOLD 变为低电平后，会将 HLDA 变为低电平，同时恢复对总线的控制。

(21)$\overline{S2}$，$\overline{S1}$，$\overline{S0}$(bus cycle status)总线周期状态信号(输出、三态)：在最大模式下，这 3 个信号组合起来指出当前总线周期所进行的操作类型，如表 2-3 所示。最大模式系统中的总线控制器 8282 就是利用这些状态信号产生访问存储器和 I/O 端口的控制信号。

表 2-3　$\overline{S2}$，$\overline{S1}$，$\overline{S0}$组合产生的总线控制功能

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	控制信号	操作过程
0	0	0	$\overline{INTA}$	发中断响应信号
0	0	1	$\overline{IORC}$	读 I/O 端口
0	1	0	$\overline{IOWC}$，$\overline{AIOWC}$	写 I/O 端口
0	1	1	—	暂停
1	0	0	$\overline{MRDC}$	取指令
1	0	1	$\overline{MRDC}$	读内存
1	1	0	$\overline{MWTC}$，$\overline{AMWC}$	写内存
1	1	1	—	无源状态

当$\overline{S2}$，$\overline{S1}$，$\overline{S0}$中至少有一个信号为低电平时，每一种组合都对应了一种具体的总线操作，因而称之为有源状态。这些总线操作都发生在前一个总线周期的 T_4 状态和下一总线周期的 T_1、T_2 状态期间；在总线周期的 T_3(包括 T_w)状态，且准备就绪信号 READY 为高电平时，$\overline{S2}$，$\overline{S1}$，$\overline{S0}$三个信号同时为高电平(即 111)，此时一个总线操作过程将要结束，而另一个新的总线周期还未开始，通常称为无源状态；而在总线周期的最后一个 T_4 状态，$\overline{S2}$，$\overline{S1}$，$\overline{S0}$中任何一个或几个信号的改变，都意味着下一个新的总线周期的开始。

(22)$\overline{RQ}/\overline{GT1}$，$\overline{RQ}/\overline{GT0}$(request/grant)总线请求信号/总线响应信号(输入/输出)：这两个引脚是双向的，低电平有效。这两个信号是最大模式系统中 8086 主 CPU 和其他协处理器(如 8087，8089)之间交换总线使用权的联络控制信号。其含义与最小模式下的 HOLD 和 HLDA 两信号类同，但 HOLD 和 HLDA 是占两个引脚，而$\overline{RQ}/\overline{GT}$(请求/允许)是出于同一个引脚。$\overline{RQ}/\overline{GT1}$和$\overline{RQ}/\overline{GT0}$是两个同类型的信号，表示可同时连接两个协处理器，其中$\overline{RQ}/\overline{GT0}$的优先级高于$\overline{RQ}/\overline{GT1}$。

(23)$\overline{\text{LOCK}}$(lock)总线封锁信号(输出、三态):当$\overline{\text{LOCK}}$为低电平时,表明此时 CPU 不允许其他总线主模块占用总线。$\overline{\text{LOCK}}$信号由指令前缀 LOCK 产生。当含有 LOCK 指令前缀的指令执行完后,$\overline{\text{LOCK}}$引脚变为高电平,从而撤销了总线封锁。此外,在 8086 CPU 处于两个中断响应周期期间,$\overline{\text{LOCK}}$信号会自动变为有效的低电平,以防止其他总线主模块在中断响应过程中占有总线而使一个完整的中断响应过程被打断。

在 DMA 期间,$\overline{\text{LOCK}}$被置为高阻状态。

(24)QS1,QS0(instruction queue status)指令队列状态信号(输出):QS1,QS0 两信号用来指示 CPU 内的指令队列的当前状态,以使外部(主要是协处理器 8087)对 CPU 内指令队列的动作进行跟踪。QS1,QS0 的组合与指令队列的状态对应关系如表 2-4 所示。

表 2-4 QS1,QS0 的组合和对应的含义

QS1	QS0	性 能
0	0	无操作
0	1	队列中操作码的第一个字节
1	0	队列空
1	1	队列中非操作码第一个字节

2.1.5 最小与最大模式

为尽可能适应各种使用场合,在设计 8086/8088 CPU 芯片时,使其可以在两种不同的模式下工作,即最大模式和最小模式。

1.最小模式

所谓最小模式,就是微型计算机系统中只有 8086 或 8088 一个微处理器。在这个系统中,所有的总线控制信号直接由 CPU 提供。要使 8086 工作在最小模式下,只需将 8086 CPU 的 MN/$\overline{\text{MX}}$引脚接+5 V。

如图 2-5 所示是 8086 在最小模式下系统的典型配置,它具有以下几个方面的特点:

(1)MN/$\overline{\text{MX}}$端接+5 V,决定了 CPU 的工作模式。

(2)有 1 片 8284A,作为时钟信号发生器。

(3)有 3 片 8282,用来作为地址信号的锁存器。

(4)当系统中所连的存储器和外设端口较多时,需要增加数据总线的驱动能力,这时,需用 2 片 8286 作为总线收发器。

2.最大模式

所谓最大模式,就是微型计算机系统中包含有两个或多个微处理器,其中一个主处理器是 8086 或 8088 微处理器,其他处理器称为协处理器,它们协助主处理器工作。常用的协处理器有 8087 协处理器和 8089 协处理器。前者是专用于数值运算的处理器,后者是专用于控制输入/输出操作的协处理器。要使 8086 CPU 按最大模式工作,只需将 MN/$\overline{\text{MX}}$引脚接地即可。

如图 2-6 所示是 8086 系统在最大模式下的典型配置。

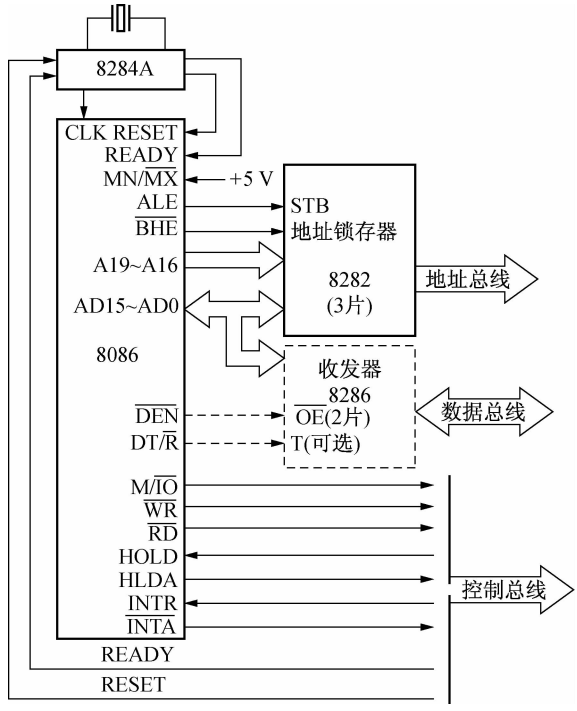


图 2-5　8086 最小模式下的典型配置

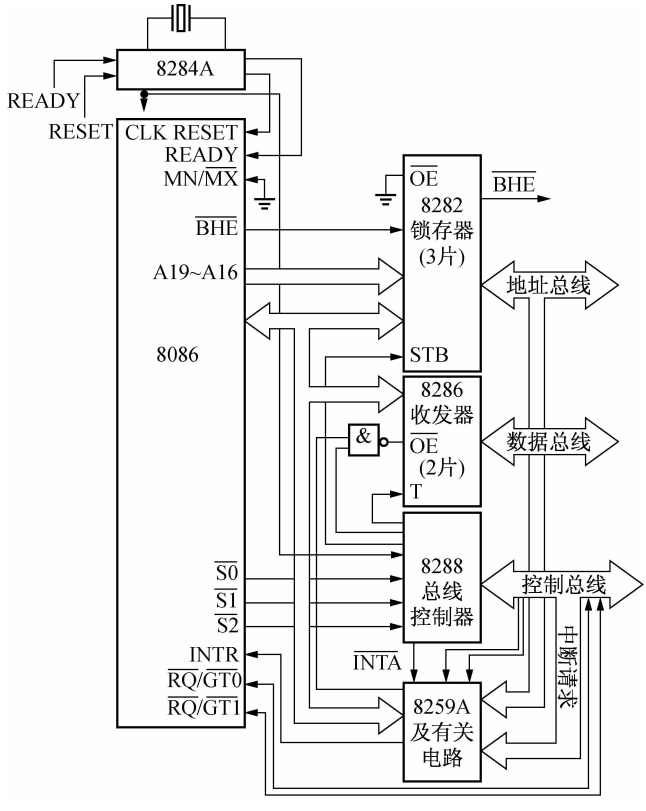


图 2-6　8086 系统最大模式下的典型配置

从图 2-6 可以看到，在最大模式下，除了 8282 锁存器和 8286 数据收发器外，还增加了 8288 总线控制器。8288 对 CPU 发出的控制信号进行变换和组合，以获得对存储器和 I/O 端口的读/写信号及对锁存器 8282 和数据收发器 8286 的控制信号。

3.最小模式和最大模式的比较

1)不同之处

最小模式下系统控制信号直接由 8086 CPU 提供；最大模式下因系统复杂，芯片数量较多，为提高驱动能力和改善总线控制能力，大多数系统控制信号由总线控制器 8288 提供。

最小模式下 8086 的 31、30 脚提供一组总线请求/响应信号(HOLD、HLDA)，而最大模式下 8086 的 31、30 脚将提供两组总线请求/响应信号($\overline{RQ}/\overline{GT0}$和$\overline{RQ}/\overline{GT1}$)。

在最大模式的系统中，一般还有中断优先级管理部件。8259A 用以对多个中断源进行中断优先级的管理；但如果中断源不多，也可以不用中断优先级管理部件。

2)相同之处

8086 的低位地址线与数据线复用，为保证地址信号维持足够的时间，需使用 ALE 信号将低位地址线锁存(通过锁存器 8282)，以形成真正的系统地址总线；8086 的数据线通过数据收发器 8286 后形成系统数据总线，以增大驱动能力，8286 主要由$\overline{DEN}$和 DT/$\overline{R}$ 两个信号控制。

2.2 8086 总线的操作时序

8086 CPU 的操作可分为内部操作与外部操作两种，内部操作是 CPU 内部执行指令的过程，外部操作是 CPU 与外部进行信息交换的过程。微机系统设计和应用时，CPU 的外部操作及外部特性是重点，外部操作主要指的是总线操作。

8086 CPU 要通过总线才能与外部交换信息，CPU 通过总线接口单元与外部交换一次信息，称为一次总线操作，所耗用的时间称为一个总线周期(bus cycle)，也称机器周期(machine cycle)。一个总线周期由若干个时钟周期组成，总线操作的类型不同，总线周期也不同。一个总线周期内完成的数据传输，一般有传送地址和传送数据两个过程。

8086 CPU 执行一条指令所需要的时间称为指令周期(instruction cycle)，一个指令周期由若干个总线周期组成。8086 的一个最基本的总线周期由 4 个时钟周期组成，分别用 T_1、T_2、T_3、T_4 表示，称为 T_1、T_2、T_3、T_4 状态。T_1 状态，CPU 输出地址信号，T_2、T_3、T_4 状态传送数据，如果在 T_2、T_3、T_4 状态无法完成数据传送，就在 T_3 与 T_4 状态之间插入 T_w。T_w 状态也叫等待状态或等待周期，是 CPU 在读写存储器或与外设交换信息时，为了与存储器或外设的速度匹配，在 T_3 状态之后插入的等待时间。T_w 状态的插入是通过系统中的等待信号电路产生一个 WAIT 信号，经时钟发生器 8284A 同步后传递给 CPU 的 READY 线来实现的，此时，总线用于数据传送，其状态一直不变，当 CPU 接到有效的 READY 信号后，数据传送结束，进入 T_4 状态。T_4 状态后，就要进入下一个总线周期的 T_1 状态，否则，就进入空闲状态，称为 T_i 状态。两个总线周期之间插入的 T_i 状态，表明总线上没有操作，也叫空操作，此时，总线上没有数据传送，也没有指令装填，但 CPU 内部仍在进行有效操作，实际上是 BIU 对 EU 的等待。在 T_i 状态，高 4 位地址线(A19/S6～A16/S3)上仍保持前一总线周期的状态信息，低 16 位地址线(AD15～AD0)上，根据前一总线周期是读周期还是写周期，分

别处于高阻态或保持原有数据信息。T_i 状态与暂停状态不同，暂停状态停止了一切操作，等待复位或中断的发生，而 T_i 状态 CPU 内部仍在进行有效操作，一旦内部操作结束，就进入下一个总线周期的 T_1 状态。

8086 的主要总线操作有：系统复位和启动操作、最大与最小模式总线读/写操作、中断响应与空闲周期操作。

2.2.1　系统的复位和启动

8086/8088 的复位和启动操作，是通过 RESET 引脚上的触发信号来执行的。当RESET引脚上有高电平时，CPU 就结束当前操作，进入初始化(复位)过程，包括把各内部寄存器(除 CS)清 0，标志寄存器清 0，指令队列清 0，将 FFFFH 送 CS。重新启动后，系统从 FFFF0H 开始执行指令。重新启动的动作是当 RESET 从高到低跳变时触发 CPU 内部的一个复位逻辑电路，经过 7 个 T 状态，CPU 即自动启动。

要注意的是，由于在复位操作时，标志寄存器被清 0，因此其中的中断屏蔽标志 IF 也被清 0，这样就阻止了所有的可屏蔽中断请求，因此，复位以后，若需要获得可屏蔽中断请求必须用开中断指令来重新设置 IF 标志。

复位操作的时序如图 2-7 所示，表 2-5 给出了复位后寄存器的状态。

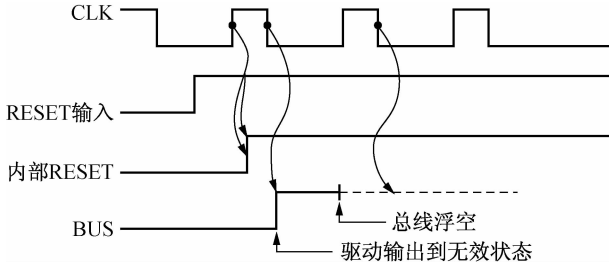


图 2-7　系统复位时序

表 2-5　复位后寄存器的状态

寄 存 器	状　　态	寄 存 器	状　　态	寄 存 器	状　　态
FR	0000H	IP	0000H	CS	FFFFH
DS	0000H	SS	0000H	ES	0000H
指令队列	空	IF	0(禁止中断)		

2.2.2　最小与最大模式总线读/写操作

CPU 为了与存储器或 I/O 端口进行一个字节的数据交换，需要执行一次总线操作，按数据传输的方向来分，可将总线操作分为读操作和写操作两种类型；按照读/写的不同对象，总线操作又可分为存储器读/写操作与 I/O 读/写操作。

1. 最小模式下的总线读周期

时序如图 2-8 所示，一个最基本的读周期包含有 4 个状态，即 T_1、T_2、T_3、T_4，必要时可插入一个或几个 T_w。

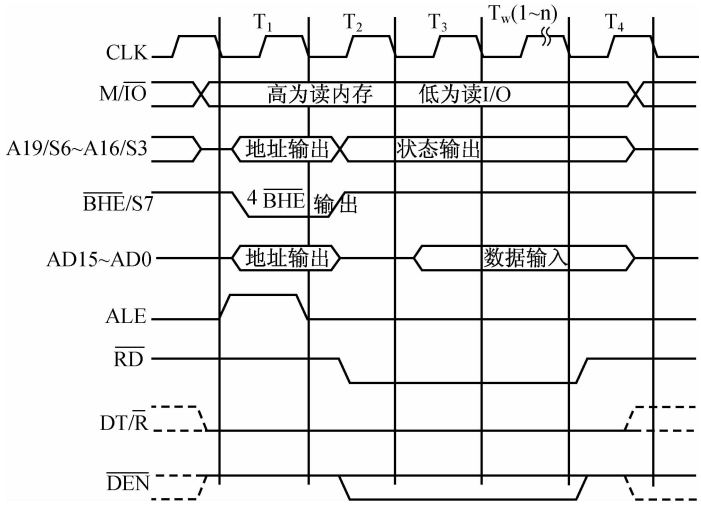


图 2-8 最小模式下总线读周期时序

1)T_1 状态

8086 输出的信号如下：

(1)M/$\overline{IO}$有效，用来指出本次读周期是读存储器还是读 I/O，它一直保持有效。

(2)地址线信号有效，高 4 位通过地址/状态线送出，低 16 位通过地址/数据线送出，用来指出操作对象的地址，即存储器单元地址或 I/O 端口地址。

(3)ALE 有效，在最小模式的系统配置中，地址信号通过地址锁存器 8282 锁存，ALE 即为 8282 的锁存信号，下降沿有效。

(4)当系统中配有总线驱动器时，T_1 使 DT/$\overline{R}$变低，用来表示本周期为读周期，并通知总线驱动器接收数据。

2)T_2 状态

(1)S3～S6 高 4 位地址/状态线送出状态信息，低 16 位 AD15～AD0 浮空，为传送数据准备。

(2)$\overline{BHE}$/S7 引脚成为 S7(无定义)。

(3)$\overline{RD}$有效，表示要对存储器或 I/O 端口进行读操作。

(4)$\overline{DEN}$有效，使得总线收发器可以传输数据。

3)T_3 状态

数据从存储器、I/O 端口读出，8086 通过 A15～A0 送上数据总线，8088 通过 AD7～AD0 送上数据总线。

4)T_w 状态

若存储器或外设速度较慢，不能及时送上数据，则通过 READY 线通知 CPU，CPU 在 T_3 的前沿(即 T_2 结束末的下降沿)检测 READY 信号，若发现 READY=0，则在 T_3 结束后自动插入一个或几个 T_w，并在每个 T_w 的前沿处检测 READY，等到 READY=1，则自动脱离 T_w 进入 T_4。

5)T_4 状态

在 T_4 与 T_3(或 T_w)的交界处(下降沿)，采集数据，使各控制线及状态线进入无效。

2.最小模式下的总线写周期

时序如图 2-9 所示，最基本的总线写周期也包括 4 个状态 T_1～T_4，必要时插入 T_w。

1)T_1 状态

基本上同读周期，但此时 DT/$\overline{R}$为高电平。

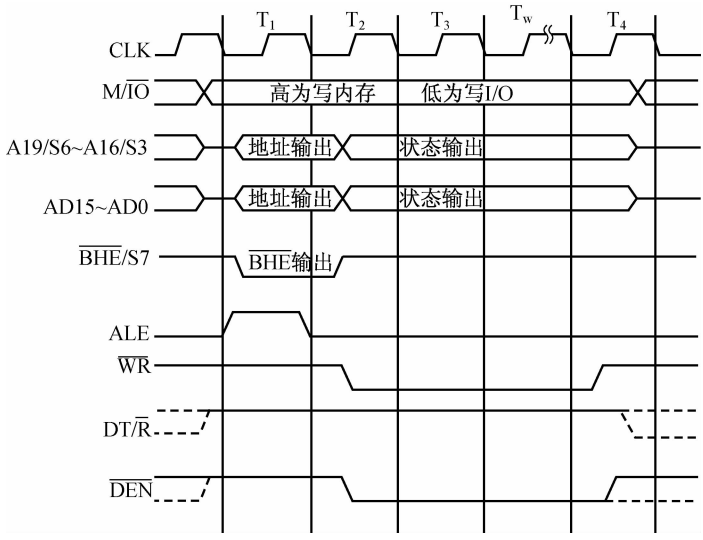


图 2-9　最小模式下总线写周期时序

2)T_2 状态

与读周期有两点不同：

(1)$\overline{RD}$变成$\overline{WR}$。

(2)AD15～AD0 不是浮空，而是发出要写入存储器或 I/O 端口的数据。

3)T_3、T_w、T_4 状态

这 3 个状态同读周期。

3.最小模式下总线请求与响应

8086 最小模式下的总线控制信号由 CPU 直接产生，用于总线控制的信号是 HOLD(总线保持请求信号，输入)、HLDA(总线保持响应信号，输出)。当系统中其他部件，如 DMA 控制器，需要占用总线时，向 CPU 发出总线请求信号。CPU 收到有效的 HOLD 信号后，如果允许让出总线，就在当前总线周期完成时，发出 HLDA 信号，同时使地址/数据总线和控制总线处于高阻态，表示让出总线，在下一个时钟周期，总线请求部件收到 HLDA 信号，获得总线控制权。在这期间，HOLD 和 HLDA 都保持高电平，直到总线请求部件完成对总线的占用后，使 HOLD 变为低电平，撤销总线请求，CPU 收到后，将 HLDA 信号变为低电平，恢复对总线的控制。

总线请求与响应操作时序如图 2-10 所示。

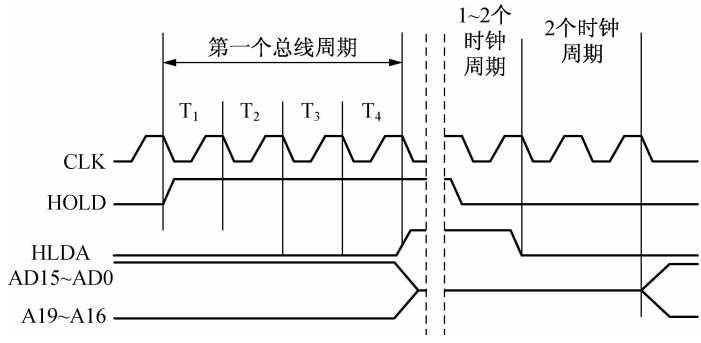


图 2-10　总线请求与响应操作时序图

4. 最大模式下的总线读周期

时序图如图 2-11 所示。与最小模式下的读周期相比，不同的就是在加入总线控制器后，可以由$\overline{S2}$、$\overline{S1}$、$\overline{S0}$状态信号来产生读信号$\overline{MRDC}$和$\overline{IORC}$。这两个信号与原$\overline{RD}$相比，不仅明确指出了操作对象，而且信号的交流特性也好，因此用它们而不用$\overline{RD}$，若仍用$\overline{RD}$信号，则最大模式与最小模式相同。

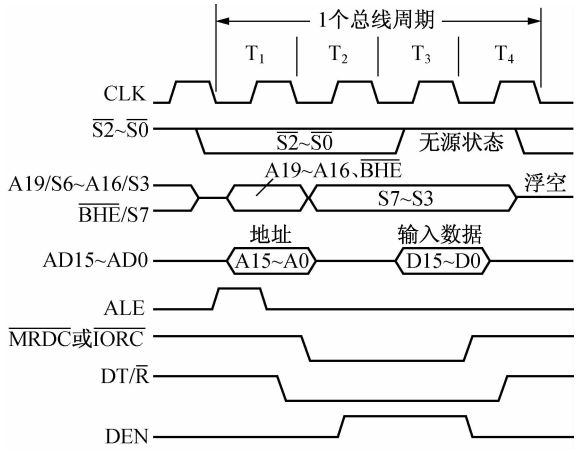


图 2-11　最大模式下总线读周期时序

1）T_1 状态

基本与最小模式相同，不同的是 ALE、DT/$\overline{R}$是由总线控制器发出的。

2）T_2 状态

与最小模式不同的是此时$\overline{RD}$变成$\overline{MRDC}$或$\overline{IORC}$，送到存储器或 I/O 端口。

3）T_3 状态

数据已读出送入数据总线，这时$\overline{S2}$、$\overline{S1}$、$\overline{S0}$三个信号均为 111（高电平）进入无源状态。若数据没能及时读出，则同最小模式一样自动插入 T_w。

4）T_4 状态

数据消失，状态信号进入高阻，$\overline{S2}$、$\overline{S1}$、$\overline{S0}$根据下一个总线周期的类型进行变化。

5. 最大模式下的总线写周期

时序图如图 2-12 所示。与上述最小模式下的总线读周期相比，就是$\overline{MRDC}$和$\overline{IORC}$成为$\overline{MWTC}$和$\overline{IOWC}$，另外还有一组$\overline{AMWC}$或$\overline{AIOWC}$（比$\overline{MWTC}$和$\overline{IOWC}$提前一个 T 有效），这时$\overline{MWTC}$（$\overline{AMWC}$）或$\overline{IOWC}$（$\overline{AIOWC}$）取代最小模式下的$\overline{WR}$。

1）T_1 状态

与最大模式下的读周期相同。

2）T_2 状态

$\overline{AMWC}$或$\overline{AIOWC}$有效，要写入的数据送入数据总线，$\overline{DEN}$有效。

3）T_3 状态

$\overline{MWTC}$或$\overline{IOWC}$有效，比$\overline{AMWC}$等慢一个 T，$\overline{S2}$、$\overline{S1}$、$\overline{S0}$进入无源状态。若需要，自动插入 T_w。

4）T_4 状态

$\overline{AMWC}$等被撤销，$\overline{S2}$、$\overline{S1}$、$\overline{S0}$根据下一总线周期的类型变化，$\overline{DEN}$失效，从而停止总线收

发器的工作，其他引脚呈高阻态。

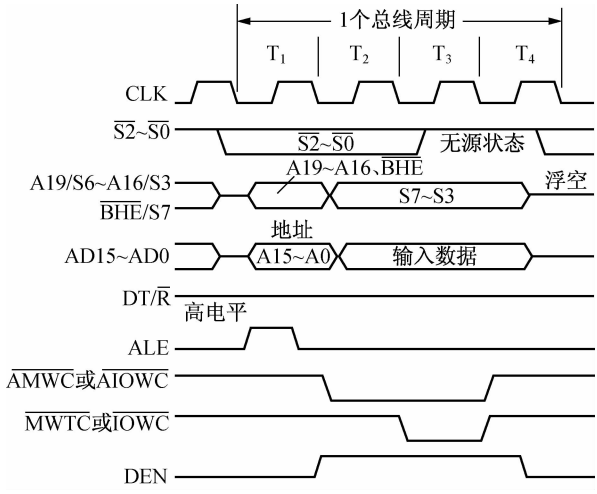


图 2-12　最大模式下总线写周期时序

2.2.3　中断响应周期

8086 有 INTR、NMI 两条中断请求信号输入引脚，用于外部中断，其中 NMI 输入的是非屏蔽中断请求信号，上升沿触发，一旦该信号有效，在现行指令执行完后立即中断，执行对应的中断处理程序，不需中断响应周期。下面主要讨论 INTR 引起的可屏蔽中断响应周期，其时序图如图 2-13 所示。

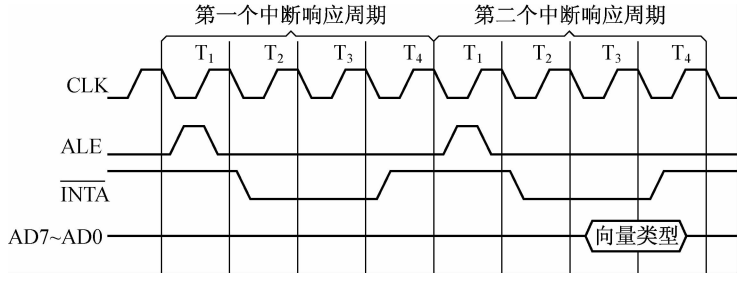


图 2-13　可屏蔽中断响应周期时序

当外部中断源通过 INTR 向 CPU 发出中断请求信号后，如果 CPU 开放了中断，在执行完当前指令之后就进入中断响应周期。中断响应周期占用两个连续的总线周期。在第一个总线周期 T_1 状态，CPU 发出 ALE 信号，作为地址锁存信号，T_2～T_4 状态，$\overline{INTA}$信号为低电平，通知外设 CPU 已接受其中断请求，同时使数据总线、地址总线浮空。紧接着经过 3 个空闲状态 T_i（8088 CPU 无须经过空闲状态），进入第二个总线周期，此时 ALE 信号和$\overline{INTA}$信号与第一个总线周期相同，不同的是在该总线周期，被响应的外设向数据总线发送一个字节的中断类型号，CPU 读入后查中断向量表，找到中断服务程序入口地址，转去执行中断服务程序。

说明：8086 最大模式下的中断响应周期与最小模式下的中断响应周期基本相同，但最大模式下，ALE 信号和$\overline{INTA}$信号是由 8288 产生的。

2.3 80x86 微处理器

从第一块微处理器产生开始，微处理器就以 80x86 系列的形式在发展。对初学者来讲，需要了解 80x86 主要微处理器的发展历程、主要性能，内部结构和寄存器组织方式。

2.3.1 80x86 微处理器的发展历程及性能

80x86 高性能微处理器是 8086/8088 向上兼容的微处理器，各型号的编程结构是非常相似的。80x86 高性能微处理器有 32 位、64 位，它们都支持多任务和多用户，存储管理采用分段、分页或不分页的虚拟存储体系，寻址空间为 4 GB，其中 Pentium 的寻址空间达到 64 GB。80x86 微处理器能够有效地处理数据、文字、图形、图像、语音等信息，已成为当今微型计算机的主流型号。

1. 80286 微处理器

80286 微处理器是 Intel 公司 1982 年推出的产品，主频 6～20 MHz，内外部数据总线 16 位，地址总线 24 位，可寻址 16 MB，PC/AT 机是最早的 286 机。它支持两种工作方式：实地址方式和保护方式。

2. 80386 微处理器

80386 微处理器是 Intel 公司 1985 年推出的高性能 32 位 CPU，主要特点如下：有 32 根双向数据线，能够形成相当于 32 位地址的地址线和 32 位数据通路；提供了实地址方式、保护方式、虚拟 8086 方式 3 种工作方式；具有逻辑、线性、物理 3 类地址。80386 微处理器的主要芯片有 80386DX、80386SX、80386SL 和 80386DL。

3. 80486 微处理器

80486 微处理器是 Intel 公司 1989 年生产的，其芯片集成有：总线接口、指令预取、浮点运算、高速缓存等功能部件，是一种新型的 32 位处理器。其主要特点是：运算速度比 80386 快；采用了 RISC 技术（指令流水处理）；内部集成浮点运算部件 FPU；片内高速缓存；内部采用新型总线。80486 微处理器的主要芯片有 80486DX、80486SX、80486DX2 和 80486DX4。

4. Pentium 微处理器

Pentium 微处理器是 Intel 公司 1993 年推出的产品，Pentium 处理器集成了 310 万个晶体管，最初推出的频率是 60 MHz、66 MHz，后来提升到 200 MHz 以上。

第一代的 Pentium 代号为 P54C，其后又发布了代号为 P55C，内建 MMX（多媒体指令集）的新版 Pentium 处理器。Pentium 微处理器内部的主要寄存器为 32 位，但有 64 位外部数据总线宽度。外部地址总线宽为 36 位，但一般使用 32 位宽。

5. Pentium MMX

Pentium MMX 是 Intel 在 Pentium 内核基础上改进的，其最大的特点是增加了 57 条 MMX 扩展指令集。这些指令专门用来处理与音频、视频相关的计算，目的是提高 CPU 处理多媒体数据的效率，加速计算机的多媒体应用。

MMX 指令集的推出非常成功，在之后生产的各种类型 CPU 都包括这些指令集，只不

过其后的产品对其原有指令进行了改进和扩展。随着微软 Windows 系统的普遍采用，音频、视频信号在个人计算机应用中已非常普及，加入 MMX 指令集正好满足了当时的这种多媒体应用需求，因此 MMX 指令集对整个处理器性能的发挥起着非常重要的作用。

6. Pentium Pro

Pentium 处理器中除了 Pentium MMX 外，还有一种版本，即 1995 年推出的 Pentium Pro 处理器。Pentium Pro 处理器是 Intel 首个专门为 32 位服务器、工作站设计的处理器，可以应用在高速辅助设计、机械引擎、科学计算等领域。Intel 在 Pentium Pro 的设计与制造上达到了新的高度，共集成了 550 万个晶体管，并且整合了高速二级缓存芯片。

由于 Intel 当时对处理器流水线深度把握不够(整个处理器的流水线级数达 14 级)，且与之配套的技术没跟上，使得初期的 Pentium Pro 版本在执行效率上还不及同频率或者低频率的 Pentium 处理器。但后来 Intel 及时发现了这一问题，并在技术上作出了相应处理，使得 Pentium Pro 处理器呈现出应有的效能水平，赢得了广大用户的信任。

7. Pentium Ⅱ

1997 年 Intel 发布了 Pentium Ⅱ处理器，它包括了 Intel 许多新的技术。在这种芯片内部集成了 750 万个晶体管(比最近一代 Pentium Pro 处理器所集成的晶体管数多出了 200 万个)，并整合了 MMX 指令集技术，可以更快更流畅地播放影音 Video、Audio 以及图像等多媒体数据，使得计算机中多媒体的应用得到前所未有的普及。在 Intel 的大力宣传下，当时计算机多媒体的处理能力在相当大程度上成了计算机档次高低的一个重要标志。

Pentium Ⅱ首次引入了单边接触(single edge contact，SEC)封装技术，将高速缓存与处理器整合在一块 PCB 板上，通过类似于内置板卡一样的金手指与主板相应插槽电路接触。

由于处理器功能的不断增强，计算机的应用范围也得到了空前扩张。借助于 Microsoft 的 Windows 操作系统应用功能的支持，Pentium Ⅱ处理器在多媒体、互联网方面的应用水平逐步得到提高，并且为广大用户所接受。

8. Pentium Ⅲ

1999 年 Intel 发布了 Pentium Ⅲ处理器。Pentium Ⅲ处理器最大的改进就是增加了 70 条新指令，这些新增加的指令主要用于互联网、3D、流式音频、视频和语音识别功能。Pentium Ⅲ可以使用户有机会在网络上享受到高质量的影片，并能以 3D 的形式参观在线博物馆等。

Intel 的 Pentium Ⅲ处理器集成了从 Compaq 公司购买的 P6 动态执行体系结构、双独立系统总线(DIB)架构、多路数据传输系统总线和 MMX 多媒体增强技术。Pentium Ⅲ处理器同样全面适合工作站和服务器应用领域。整个 Pentium Ⅲ是一个相当庞大的系列，所涉及的处理器主频种类非常多，最低的是 450 MHz，最高的可达 1.33 GHz，系统总线频率也有两种：133 MHz 和 100 MHz。内存寻址可以支持到 4 GB，物理内存可以支持到 64 GB。制造工艺也有多种，最初是采用 0.25 μm，后期版本采用的是 0.18 μm，而服务器所用的 Pentium Ⅲ Xeon处理器是采用最先进的 0.13 μm 制造工艺的。

Pentium Ⅲ处理器快速系统总线(FSB)设定在 133 MHz，每个时钟周期传输 64 位数据，提供 8 B×133 MHz=1 064 MB/s 的数据带宽。在处理器的封装上存在着两种方式，既有 Pentium 处理器以前的 Socket 架构，也可采用 Pentium Ⅱ处理器的 SEC 构架。这里的

Socket 就是在此之前著名的 Socket 370。

9. Pentium 4

2000 年 Intel 发布了 Pentium 4 处理器。用户使用基于 Pentium 4 处理器的个人计算机,可以创建专业品质的影片,通过互联网传递电视品质的影像,进行实时语音、影像通信、3D 渲染,快速进行 MP3 编码译码运算,在连接互联网时运行多个多媒体软件。

Pentium 4 采用 Socket 处理器架构。由于有不同的内核,所以在 P4 处理器家族中也存在多种不同的 Socket 架构。Pentium 4 处理器集成了 4 200 万个晶体管,改进版的 Pentium 4(Northwood)更是集成了 5 500 万个晶体管,并且开始采用 0.18 μm 进造工艺,主频达到了 1.5 GHz。

10. Intel Core 2 Duo

Intel Core 2 Duo 是 Intel 公司 2006 年 7 月发布的双核芯微处理器,采用 65 nm 和 45 nm(Wolfdale 内核)工艺制造。其主要特点为:增加 SSSE3 指令;增加 SSE4.1 指令;采用 800/1 066/1 333 MHz 前置总线;支持多操作系统;增强远端管理计算机的功能。

2.3.2 80386 的内部结构

80386 是 80x86 系列微处理器中具有代表性的微处理器。下面主要介绍 80386 的主要特性和内部结构。

1. 80386 的主要特性

(1)灵活的 32 位微处理器,提供 32 位指令,可采用 8 位、16 位或 32 位的数据宽度。

(2)提供 32 位外部总线接口,最大数据传输速率为 32 Mb/s。

(3)具有片内集成的存储器管理部件 MMU,可支持虚拟存储和特权保护。

(4)具有实地址模式、保护模式和虚拟 8086 模式 3 种工作方式。

(5)具有 4 GB(2^{32})的物理寻址空间和 64 TB(2^{46})的虚拟存储空间。

(6)通过配用 80287 或 80387 数值协处理器可支持高速数值处理。

(7)与 8086、80286 芯片完全兼容。

2. 80386 的内部结构

80386 内部结构如图 2-14 所示。80386 的六大部件分别属于三大部分,即中央处理部件(CPU)、总线接口部件(BIU)和存储器管理部件(MMU)。下面具体介绍这三大部分的详细划分和功能。

1)中央处理部件

中央处理部件(CPU)包括指令预取部件(IPU)、指令译码部件(IDU)和指令执行部件(EU)3 部分。指令预取部件包括 16 字节的预取队列寄存器,可存放 5 条左右的指令;指令译码部件对指令操作码进行译码,并将其存放在已译码的指令队列中,供执行部件取用;指令执行部件包括 8 个 32 位通用寄存器、1 个 64 位移位加法器和一个 ALU。通用寄存器组既可用于数据操作,又可用于地址计算。移位寄存器用来有效地实现指令的移位、循环移位和位操作,同时也可用于乘法和除法等操作,以加快运算速度。

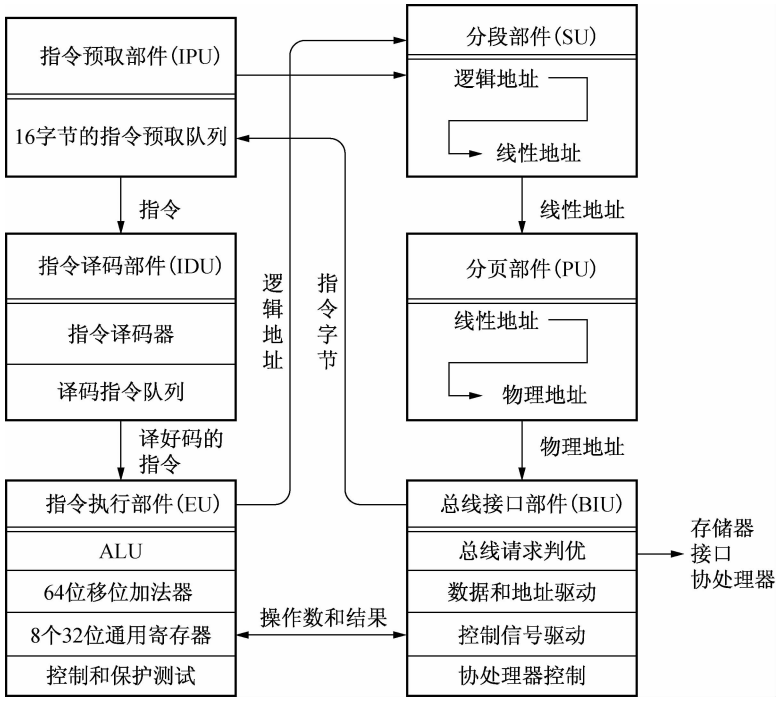


图 2-14　80386 结构图

2)总线接口部件

总线接口部件(BIU)提供中央处理部件和外部系统之间的高速接口。其功能是产生访问存储器和 I/O 端口所必需的地址、数据和命令信号。80386 的总线周期仅为两个时钟周期。

3)存储器管理部件

存储器管理部件(MMU)又分为分段部件(SU)和分页部件(PU)两部分。分段部件完成从逻辑地址到线性地址之间的转换,把线性地址送到分页部件,同时完成总线周期分段的违法检查;分页部件完成线性地址到物理地址之间的转换。若不采用分页单元功能,则线性地址即是物理地址。

2.3.3　80386 的寄存器

80386 的寄存器如图 2-15 所示。

80386 共有 40 个寄存器,按功能分为通用寄存器、段寄存器、标志和控制寄存器、系统地址寄存器、调试寄存器和测试寄存器。

1.通用寄存器

8 个通用寄存器和 8086 通用寄存器相同,只是扩展到了 32 位,寄存器名字前加了一个字符 E,即 EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP,仍然支持 8 位和 16 位操作,用法和 8086 系统相同。

2.段寄存器及段描述符高速缓存器

80386 有 6 个 16 位段寄存器,分别为 CS、SS、DS、ES、FS 和 GS,每个段寄存器对应一个 64 位的描述符。比 8086 增加了两个附加段寄存器 FS、GS。在实模式下,段寄存器和 8086

用法相同;在保护模式下,段寄存器称为段选择器,它与描述符配合实现段寻址。64 位段描述符寄存器对程序员是不可见的。为了加快对内存中描述符表的查询速度,在选择符内容装入时,段描述符同时装入段描述符寄存器。这样,只要段选择符内容不变,就不需要到内存中查询描述符表,从而加快了段地址寻址的速度,如图 2-16 所示。

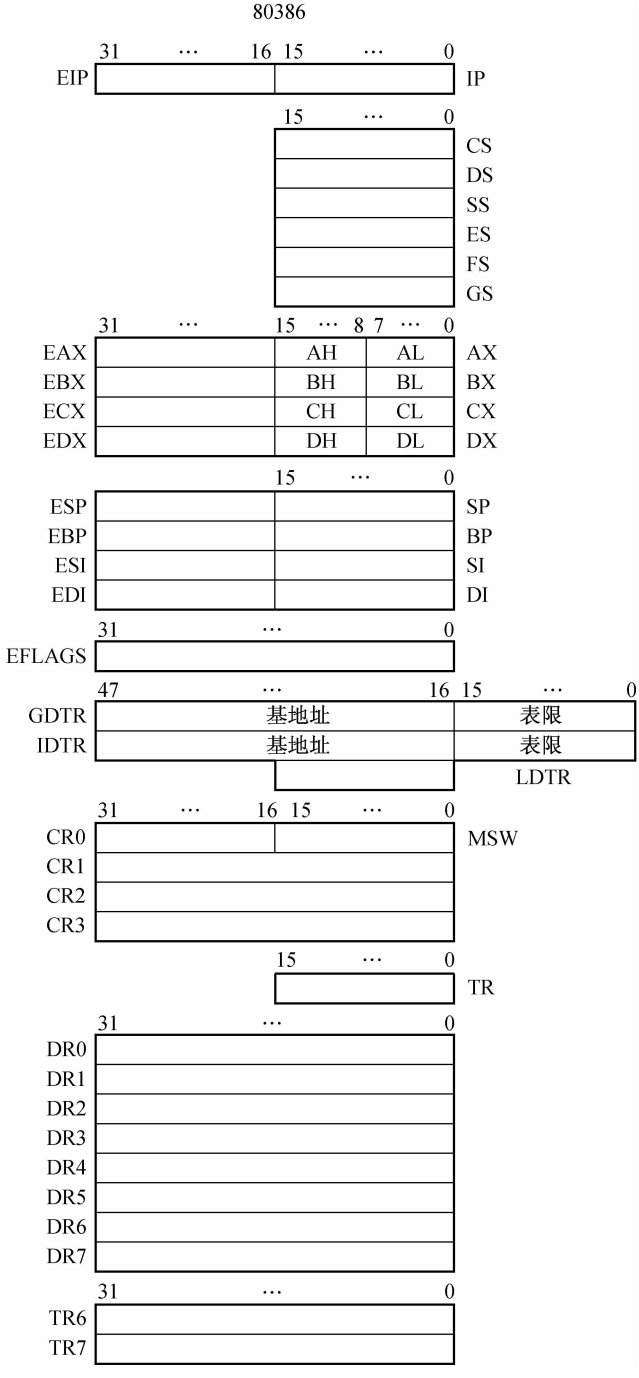


图 2-15 80386 的寄存器

段寄存器 (16位)		段描述符高速缓存器		
CS		段基地址 (32位)	段限值 (20位)	属性 (12位)
DS		段基地址 (32位)	段限值 (20位)	属性 (12位)
ES		段基地址 (32位)	段限值 (20位)	属性 (12位)
FS		段基地址 (32位)	段限值 (20位)	属性 (12位)
GS		段基地址 (32位)	段限值 (20位)	属性 (12位)
SS		段基地址 (32位)	段限值 (20位)	属性 (12位)

图 2-16　段寄存器及段描述符高速缓存器

1)6 个 16 位段寄存器

16 位段寄存器的组成如图 2-17 所示。

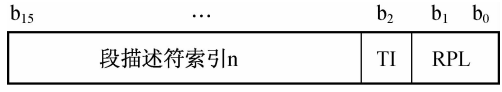


图 2-17　段寄存器组成

- 在保护模式下,段寄存器被称为一个 16 位的段选择字,其中 b_1、b_0 位为请求特权级 RPL,可以请求特权层的级别为 0～3 级。
- b_2 位是表的指针,TI＝0,表示访问全局描述符表 GDT;TI＝1,表示访问局部描述符表 LDT。
- 段选择字的高 13 位(b_{15}～b_3)×8 作为描述符表的索引值,即作为描述符表的偏移地址,由表基地址寄存器中的值指向描述符表的首地址。
- 由段选择字的高 13 位指向 2^{13}＝8 K 个描述符中的任意一个描述符,每个描述符由 8 字节组成。
- 一旦段选择字的内容得到了改变,就由分段部件自动从位于内存的描述符表中取出对应的 8 字节描述符,且装载到相应段寄存器所对应的描述符高速缓存器中。
- 一旦装入后,对应段的访问全部由此高速缓存器中该 64 位描述符确定。不需要每次从慢速的内存中去查描述符表,大大加快了访问速度。

2)6 个段描述符

每个段对应一个段描述符(8 个字节),6 个段描述符存放在 CPU 内的段描述符高速缓存器中,它们均由内存的描述符表拷贝而成,CPU 访问某一段时,均按存放在 CPU 内该段的段描述符所描述的信息进行操作。

每个段描述符共 8 个字节,包括 32 位段基址,20 位段限值,12 位段属性信息,如图 2-18 所示。

3. 指令指针和标志寄存器

80386 CPU 中有一个 32 位的指令指针(EIP)和一个 32 位的标志寄存器(EFLAGS),如图 2-19 所示。

指令指针(EIP)保存下一条待执行指令所在代码段内的偏移值,也就是偏离代码段首地址的字节地址数值。EIP 的低 16 位为 IP,供实地址方式下采用。

EFLAGS 在 8086 的 16 位 FLAGS 基础上扩充了高 16 位,其中,b_{11}～b_0 中保留了 8086

CPU 中 6 个状态标志和 3 个控制标志，其余位增加了 NT 与 IOPL，高 16 位中新增了 6 个标志位。这些扩充标志位的含义如下：

- NT，嵌套任务标志。
- IOPL，I/O 特权级别。
- RF(resume flag)，恢复标志。
- VM(virtual 8086 mode)，虚拟 8086 方式标志位。
- AC，对准检查标志位。
- VIF，虚拟中断标志位。
- VIP，虚拟中断位。
- ID，识别标志位。

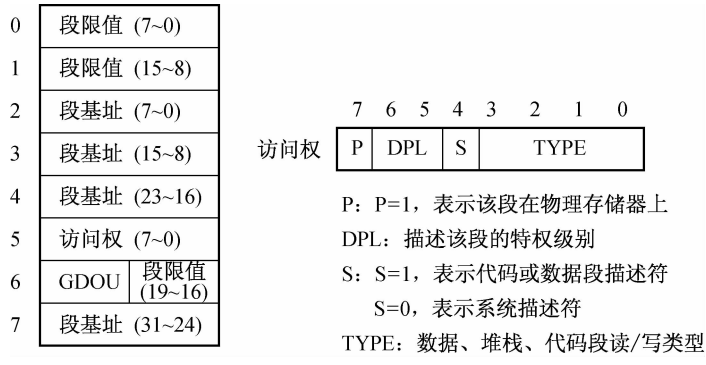


图 2-18 段描述符的具体组成

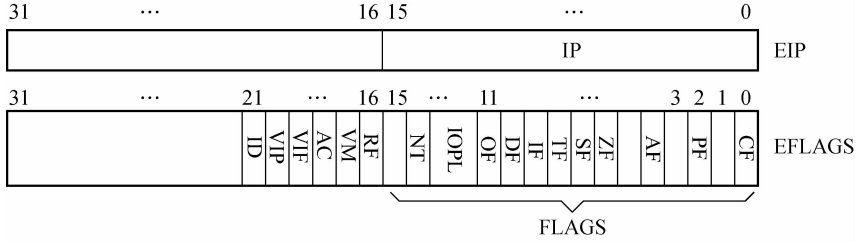


图 2-19 指令指针和标志寄存器

4. 系统地址寄存器

80386 有 4 个系统地址寄存器，用来保护操作系统需要保护的信息和地址转换表信息，定义目前正在执行任务的环境、地址空间和中断向量空间，如图 2-20 所示。

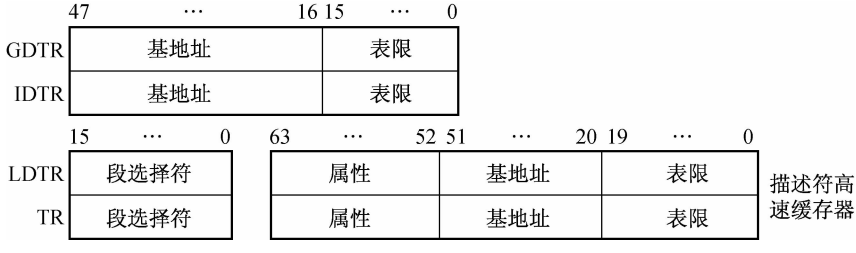


图 2-20 系统地址寄存器

(1)GDTR(global descriptor table register),全局描述符表寄存器,共有 48 位。其中,高 32 位保存全局描述符表的线性基地址,低 16 位是表限字段,即表的最大长度仅 64 KB。

(2)IDTR(interrupt descriptor table register),中断描述符表寄存器,共有 48 位。其中高 32 位用于保存中断描述符表 IDT 的 32 位线性基地址,低 16 位是表限字段,表的最大长度也是 64 KB。

说明:当 CPU 上电复位后,GDTR、IDTR 中的基地址均为 0,表限值为 FFFFH,一旦切换到保护模式时,GDTR、IDTR 必须装入一个新值,作为系统初始化的一部分。

(3)LDTR(local descriptor table register),局部描述符寄存器,包括 16 位段选择符和不可编程的 64 位描述符高速缓存器。在 64 位描述符寄存器中,有 32 位 LDT 的线性基地址,20 位的表限及 12 位的描述符属性。

(4)TR(task register),任务寄存器,包括 16 位段选择符和 64 位描述符高速缓存器。其中包括 32 位任务状态段的线性基地址,20 位的表限及 12 位的描述符属性。

说明:当 CPU 上电复位后,32 位基地址为 0,表限值为 FFFFH,一旦发生任务转换时,对于新任务的任务状态段 TSS,将其 16 位段选择符、64 位描述符自动调入 TR 中。

5.4 个控制寄存器

从 80386 CPU 开始,片内就有 CR0、CR1、CR2、CR3 共 4 个 32 位的控制寄存器,其中,CR1 一直被 Intel 公司保留未用。

1)CR0 控制寄存器

CR0 控制寄存器如图 2-21 所示。

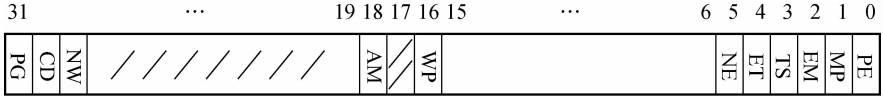


图 2-21　CR0 控制寄存器

32 位的 CR0 选用了 11 位用于控制微处理器的操作模式和状态。各标志位含义如下:

- PE(protect enable),保护允许控制位。
- MP(math present),监督协处理器控制位(monitor coprocessor)。
- EM(emulate coprocessor),仿真协处理器控制位。
- TS(task switched),任务转移位。
- ET(extension type),扩展类型控制位。
- NE(numeric error),浮点异常控制位。
- WP(write protect),写保护控制位。
- AM(alignment mask),对齐标志控制位。
- NW(not write-through),片内 cache 非通写控制位。
- CD(cache disable),cache 禁止控制位。
- PG(page),分页控制位。

2)CR2 页故障线性地址寄存器

CR2 用于保存最后出现页故障的 32 位线性地址。操作系统中的页异常处理程序可以通过检查 CR2 的内容,得知 32 位的线性地址。

3)CR3 页目录基址寄存器

CR3 中高 20 位存放页目录表的物理基地址。在进行分页变换时,加上 10 位线性地址乘 4,找到某一存储容量为 4 字节的页描述符。在页目录基址寄存器的低 12 位中,有 PCD 和 PWT 两位控制位,其余 10 位保留。

CR2、CR3 寄存器如图 2-22 所示。

图 2-22 控制寄存器 CR2、CR3

- PCD,页面 cache 禁止控制位。PCD=1 时,禁止片内 cache 分页;PCD=0 时,允许片内 cache 分页。
- PWT,页面 cache 通写控制位。PWT=1 时,外部 cache 用通写法;PWT=0 时,外部 cache 用回写法。

6. 调试寄存器

80386 CPU 芯片内有 8 个调试寄存器 DR0～DR7,为调试提供了硬件支持。其中前 4 个为保存 4 个线性断点地址寄存器;DR4、DR5 为备用寄存器;DR6 为断点状态寄存器,通过该寄存器的标志可以检测事故并进入事故处理程序或禁止进入事故处理程序;最后一个 DR7 是断点控制寄存器,用来规定断点字段的长度、断点访问类型等。

7. 测试寄存器

80386 有 8 个 32 位的测试寄存器 TR0～TR7,其中前 6 个为系统保留,后面两个用于控制对转换后备缓冲器中 RAM 和 CAM 的测试。TR6 是测试命令寄存器,最后一个 TR7 是测试数据寄存器,用来保存测试结果的状态。

2.4 80386 的工作方式

80386 工作模式分为 3 种:实地址方式、保护方式和虚拟 8086 方式。

2.4.1 实地址方式

实地址方式的工作原理与 8086 基本相同,其主要区别是 32 位微处理器能处理 32 位数据。系统启动后,80386 自动进入实地址模式。在此方式下,采用类似于 8086 的体系结构。实地址模式与 8086 的寻址原理一致,即采用段寄存器(16 位)的内容乘以 16,再加上段内偏移量(16 位)以形成物理地址,寻址范围为 1 MB。实地址方式存储器寻址时物理地址的形成与计算过程如图 2-23 所示。

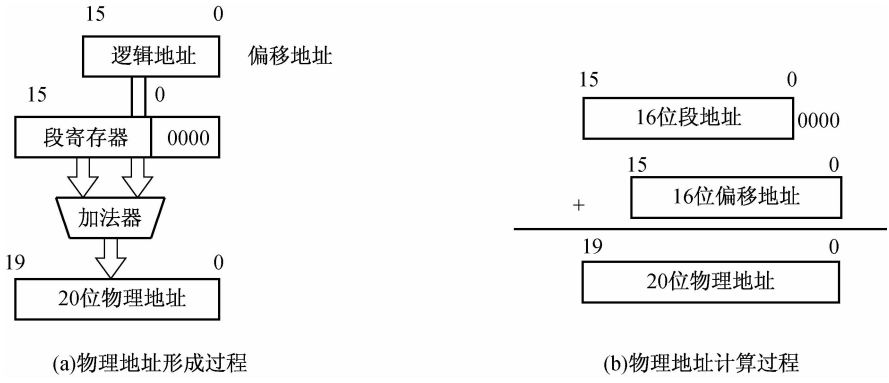


图 2-23　实地址方式存储器寻址时物理地址的形成与计算过程

熟悉实地址方式存储器寻址时物理地址的形成与计算过程后，已知存储器段地址和偏移地址，实地址方式的寻址过程如图 2-24 所示。

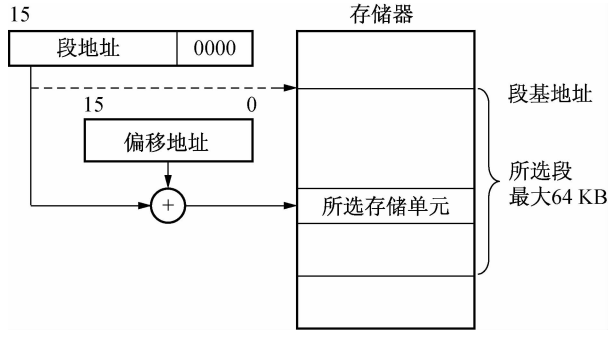


图 2-24　实地址方式存储器寻址过程

2.4.2　保护方式

80386 保护工作方式可以工作在只分段、只分页或既分段又分页 3 种方式下。

1. 利用 GDTR 与 LDTR 分别访问 GDT 与 LDT

当前任务访问 LDT 时，由 LDTR 中对应的 64 位描述符高速缓存器中的 32 位基地址作为 LDT 的基地址，再由段选择符的高 13 位左移 3 位后作为 LDT 的偏移地址，指向所要访问的 8 字节的段描述符。

当任务发生转换之前，系统要把 LDT 描述符的段选择符的值加载到 LDTR 的段选择符字段，当任务发生转换时，由 LDTR 中段选择符的高 13 位左移 3 位后，作为 GDT 中的偏移地址，在 GDT 中取出该任务的 LDT 描述符，并装入 LDTR 对应的描述符高速缓存器，于是在 LDTR 中的高速缓存器中，存入了当前 LDT 的基地址，表限以及属性等。

由 LDTR 访问 LDT 的过程如图 2-25 所示，由 GDTR 访问 GDT 的过程如图 2-26 所示。

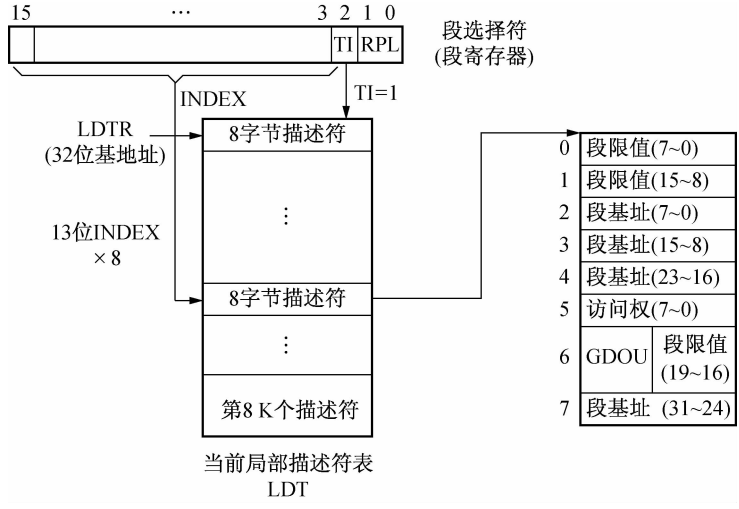


图 2-25 LDTR 访问 LDT 的过程

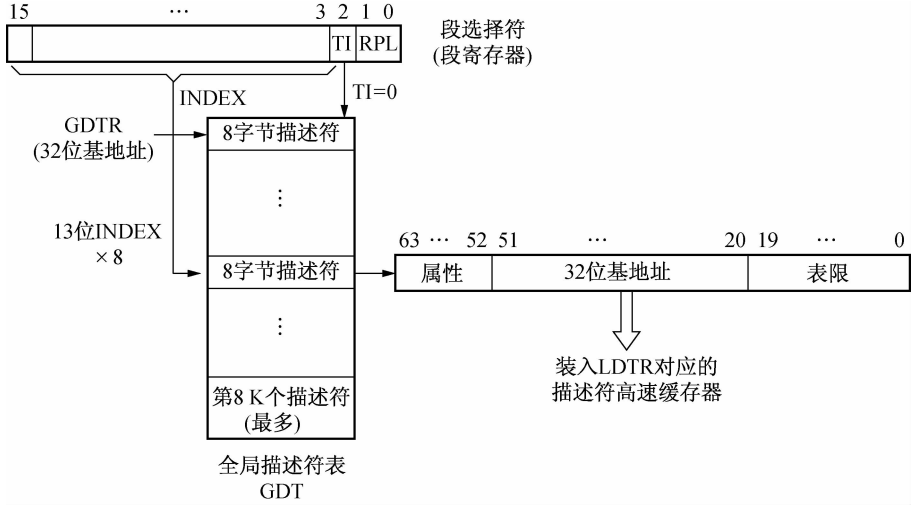


图 2-26 由 GDTR 访问 GDT 过程

2. 分段地址的转换

80386 的分段地址转换过程如图 2-27 所示。根据段寄存器即 TI,确定访问当前 LDT(TI=1)或 GDT(TI=0)。如果访问 LDT,将 13 位索引值左移 3 位后,作为 LDT 基地址的偏移量,指向 8 字节描述符,段描述符中的 32 位基地址加上指令中的 32 位偏移量成为 32 位线性地址,如果不分页只分段,结果为该机器指令所寻址的物理地址,如果还需要分页,此线性地址分为 3 段,即页目录(号)、页面(号)和偏移量。

系统还有一个 48 位的中断描述符表寄存器,其中 32 位的基地址是全系统中仅有的一个中断描述符表 IDT 的基地址。Pentium 机中所有的中断,包括软件中断、硬件中断以及 CPU 内部的异常中断,在 IDT 中均有一个描述符,IDT 中最多可有 256 个描述符。每个中断描述符也是 8 字节,主要包括中断服务程序的入口地址。

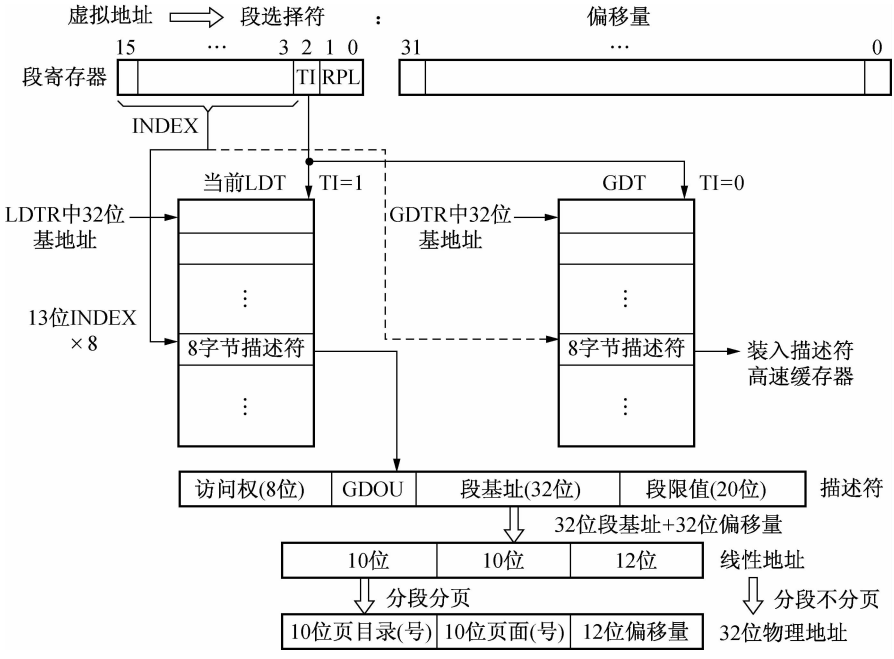


图 2-27　80386 分段地址转换过程图

3. 分页地址转换

分页地址转换由 CPU 内的分页部件 PU 来实现，它将 32 位的线性地址转换成 32 位的物理地址。32 位线性地址可能来自分段部件 SU(既分段又分页方式)，也可能是不分段只分页的情况。程序不提供段选择符，只由指令寄存器提供的 32 位地址作为线性地址，即 10 位的页目录(号)、10 位的页表(号)和 12 位的页内偏移量。Pentium 的分页部件 PU 可以按 80386/80486 每页 4 KB 分页(PSE＝0)，也可按每页 4 MB 分页(PSE＝1)。

1)4 KB 分页方式

4 KB 分页方式采用两级分页方式：第一级有一个 4 KB 的页目录表，可存放 1 024 个页目录项，称之为高级管理；第二级有一个 4 KB 的页表，可以存放 1 024 个页表项，称之为低级管理。页目录项与页表项均为 32 位(4 字节)，如图 2-28 所示。

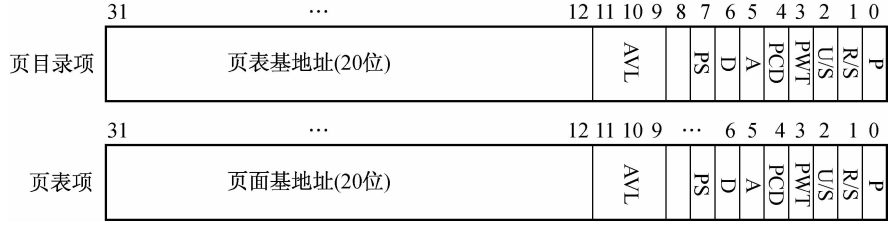


图 2-28　页目录项与页表项

页目录项与页表项的格式基本相同，其中，高 20 位分别是 4 KB 页表的基地址以及最终指令所要访问页(4 KB)的基地址。其他位为标志位或控制位。

80386 的 4 KB 分页方式地址转换如图 2-29 所示。将 32 位线性地址定义为 3 个字段，页目录(号)、页面(号)以及偏移量分别为 10 位、10 位和 12 位。

如果不使用物理地址扩充方式，即 80386 CPU 工作在 32 位物理地址，而不是 36 位物理地址的情况下，那么，全系统只有一个页目录表，由 32 位控制寄存器 CR3 指向页目录表的起始地址，CR3 中低 12 位全为 0。线性地址中的 10 位页目录(号)乘以 4，就是页目录表中的偏移地址，与 32 位的 CR3 相加，指向一个 4 B 的页目录项。在所查询的某一页表项中，将高 20 位页表基地址的低 12 位补 0，相当于左移 12 位作为页表的基地址，将线性地址中的页面(号)乘以 4，变成 12 位的偏移地址，与 32 位页表的基地址相加，指向某一个页表项，同样，将所选中页表项中高 20 位页面基地址的低 12 位补 0，作为物理内存页的起始地址，这高 20 位就被称为物理页号，每页大小 4 KB，物理内存页的起始地址加上线性地址中的低 12 位偏移量，于是求得最终要寻址的 32 位物理地址，完成了线性地址到物理地址的转换。

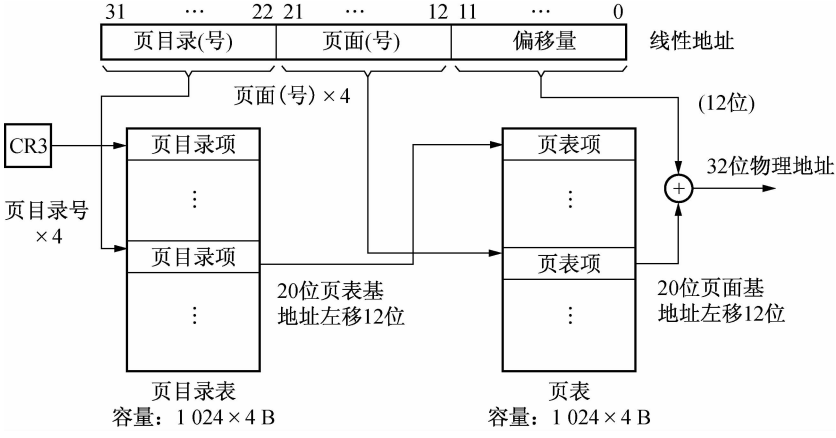


图 2-29 4 KB 分页方式地址转换

2)4 MB 分页方式

80386 将 32 位线性地址分为两个字段，页面(号)10 位，偏移量 22 位，采用单级页表分页方式。由于页面(号)仅 10 位，页表中共有 1 024 个页表项，每个页表项 32 位，页表仅占 4 KB，这是 Pentium 较 80386/80486 增加的分页方式。全系统只有一个页表，由控制寄存器 CR3 指向页表的起始地址，如图 2-30 所示。

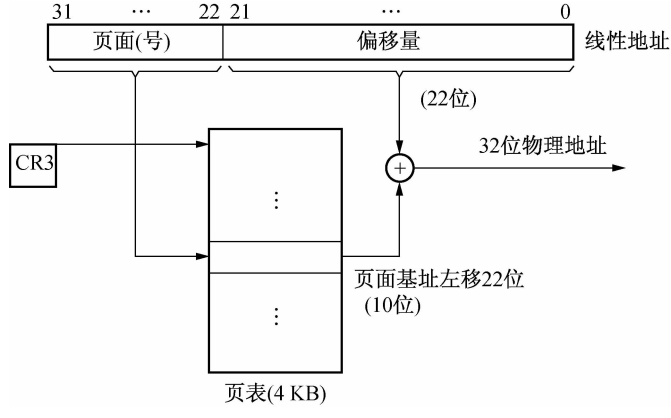


图 2-30 4 MB 分页方式的地址转换

4 MB 分页方式地址的转换过程：10 位页面(号)左移 22 位，与 32 位 CR3 相加产生页表

项的物理地址，但所寻址页表项中仅有高 10 位为页面基地址，而不是 4 KB 分页方式中的 20 位为页面基地址。将此 10 位地址左移 22 位，相当于低 22 位补 0，然后与线性地址中的 22 位偏移量相加，最终产生 32 位的物理地址。

4. 既分段又分页

80386 微处理器利用片内存储器管理部件 MMU，对存储器系统实行既分段又分页的两级管理。分段分页方式是先分段后分页，在分段的基础上进行分页，分段所形成的 32 位线性地址不是最后的物理地址，而是提供给分页部件，作为页目录(号)、页表(号)以及页内偏移量，按 4 KB 大小分页。分段分页地址转换过程如图 2-31 所示。

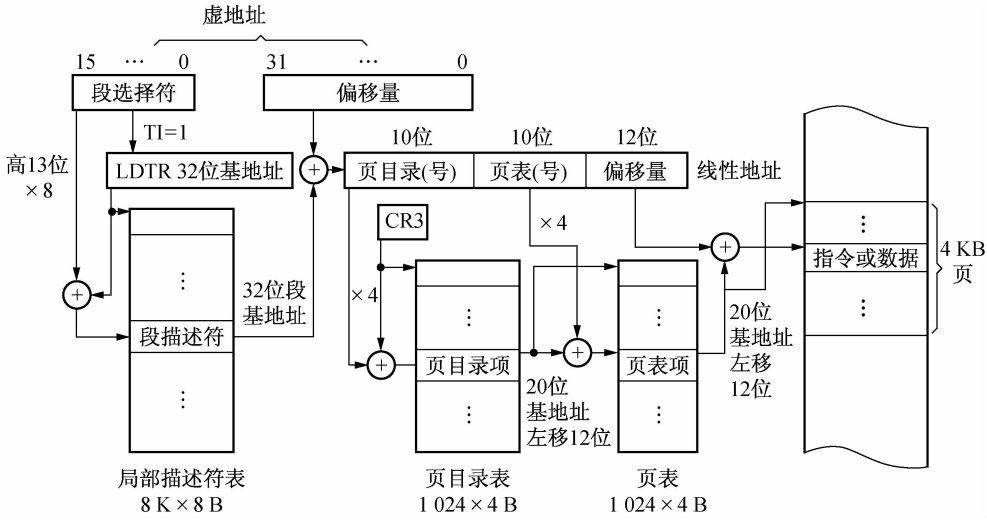


图 2-31　分段分页地址转换过程

分段分页地址转换过程如下：

(1)由于段选择符中 TI＝1，所以从 64 KB 的局部描述符表中查找对应的段描述符，该段描述符的指针通过以下方式计算：取 LDTR 中 32 位基地址与段选择符高 13 位乘以 8 的值相加。

(2)段描述符中 32 位线性地址主要是通过 32 位段基地址与虚地址中 32 位偏移量相加计算出来的。线性地址的高 10 位为页目录(号)，中间 10 位为页表(号)，低 12 位为偏移量。

(3)以 CR3 中 32 位基地址为页目录表中基地址，按 4 KB 大小分页，其分页的原理与上述分页原理完全相同。

2.4.3　虚拟 8086 方式

虚拟 8086 方式简称虚拟 86(V86)方式，它是在 32 位保护方式下支持 16 位实地址方式应用程序的一种保护方式。从 80386 一直到 Pentium CPU，都支持虚拟 86 方式。在一般保护方式下，执行指令 IRETD，则进入虚拟 8086 方式，此时 32 位标志寄存器 EFLAGS 中的 b_{17}＝0，即 VM＝0，当 VM＝1 时，切换到 V86 方式。在虚拟 8086 方式下，运行 8086 程序可以尽量利用 32 位微处理器的保护机制。尤其是 32 位微处理器允许同时执行 8086 的操作系统及其应用程序和 32 位微处理器操作系统的应用程序。虚拟 8086 方式内存寻址与实地址方式相同，不用描述符。与实地址方式相比，虚拟 8086 方式多出了约 64 KB 寻址空间，因

为在虚拟 8086 方式下,可用到 A20 地址线,而 8086 中只有 20 位地址线。

虚拟 8086 方式是多任务的,具有 80386 全部的保护功能,既可以执行 8086 程序,也可以执行 80386 程序。

1. 虚拟 8086 方式下不分页地址转换过程

虚拟 8086(V86)方式和实地址方式下的地址转换方式基本相同。虚拟 8086 方式下的不分页地址转换过程如图 2-32 所示。如果在 V86 方式下禁止分页,那么其地址转换方式完全与实方式下的地址转换方式相同。如果允许分页功能的话,可以通过分页机制,在 4 GB 的物理地址范围内分配多个 8086 的 1 MB 地址空间。

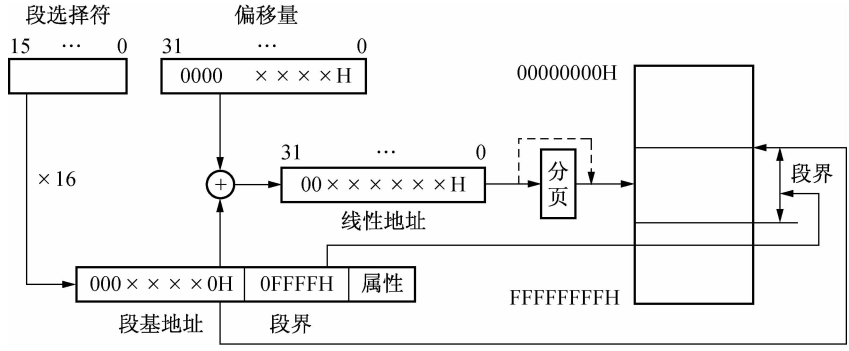


图 2-32 虚拟 8086 方式下不分页地址转换过程

在进行地址转换时,把段寄存器的描述符高速缓存器中的基地址同偏移量相加而得到的地址称为线性地址,将线性地址再通过分页机制进行分页转换,就可以产生最终的物理地址。

2. 在虚拟 8086 方式下分页地址转换方法

在虚拟 8086 方式下分页地址转换的过程如图 2-33 所示。

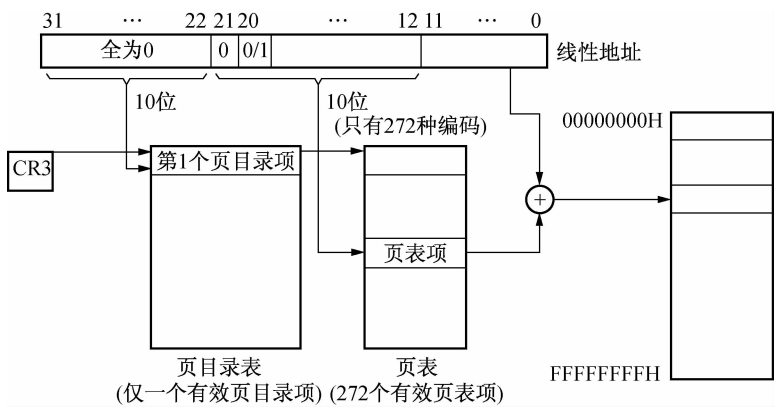


图 2-33 虚拟 8086 方式下分页地址转换的过程

在虚拟 8086 方式下分页地址转换的方法如下:

(1)32 位线性地址的最大值为 0010FFEFH,高 11 位恒为 0,那么 b_{31}～b_{22} 这高 10 位一定全为 0,线性地址的高 10 位作为页目录项的基地址,所以 V86 方式下,只访问页目录表中的第 1 个页目录项,而本来可以有 1 024 个页目录项。

(2)线性地址的中间 10 位地址中,一般情况下高 2 位(b_{21}、b_{20})均为 0,所以只有低 8 位是有效位,那么可以访问 256 个页表项,考虑到 FFFFH×16+FFFFH=10FFEFH 的特殊情况,则相加后产生上溢出,使得 b_{20} 上溢出为 1,在 $b_{20}=1$ 的溢出情况下,而 b_{19}～b_0 只有 000H～00FH 共 16 种可能,所以可查找的页表项增加 16 个,变为 272 个。

(3)本来共计可查找 1 024 个页表项,但是 V86 模式下只可查找 272 个页表项。由每个页表项中的 20 位页面基地址与线性地址低 12 位相拼而成为 32 位的物理地址。

(4)V86 模式可寻址物理空间,由页表项中的 20 位基地址与线性地址的低 12 位相拼而成为 32 位的物理地址,因此,可访问 4 GB 存储空间。

(5)在 V86 模式下,如果禁止分页,存储器寻址空间仅为 1 MB。这与 8086 基本上相同,每段存储空间最大为 64 KB。

(6)在 V86 模式允许分页的情况下,物理页的大小为 4 KB,即按 4 KB 大小分页,多任务中的每一个任务所用的全部页面可以定位在一个物理空间,这个空间大小为 1 MB,不同任务的代码定义在不同的 1 MB 内存空间,4 GB 物理空间可以定义若干个虚拟 8086 的地址空间(1 MB),把 4 GB 物理存储器虚拟化了,这就是取名为虚拟 8086 模式的原因。

本 章 小 结

本章主要介绍了 8086 微处理器的结构及其外部引脚和功能。在此基础上讲述了 8086 最小工作模式及最大工作模式,详细介绍了基本的总线周期、主要总线操作,以及总线操作的时序。重点分析了 80386 的内部结构、寄存器组织以及 80386 的 3 种工作方式。

通过本章学习,可以帮助读者掌握 16 位微处理器 8086 的组织结构和工作原理,进一步了解 32 位微处理器 80386 的相关知识及工作方式。学习过程中读者应重点围绕 8086 微处理器的工作方式,总线周期、引脚与总线的连接方法等内容进行学习,为后续接口芯片的学习打下基础。

习　题　2

1. 8086 是多少位的微处理器? 为什么?

2. EU 与 BIU 各自的功能是什么? 如何协同工作?

3. 8086/8088 微处理器内部有哪些寄存器? 它们的主要作用是什么?

4. 8086 对存储器的管理为什么采用分段的办法?

5. 在 8086 中,逻辑地址、偏移地址、物理地址分别指的是什么? 试具体说明。

6. 给定一个存放数据的内存单元的偏移地址是 20C0H,(DS)=0C00EH,求出该内存单元的物理地址。

7. 8086/8088 为什么采用地址/数据引脚复用技术?

8. 怎样确定 8086 的最大或最小工作模式? 最大、最小模式产生控制信号的方法有何不同?

9. 8086 被复位以后,有关寄存器的状态是什么? 微处理器从何处开始执行程序?

10. 8086 基本总线周期是如何组成的？各状态中完成什么基本操作？

11. 结合 8086 最小模式下总线操作时序图，说明 ALE、M/$\overline{IO}$、DT/$\overline{R}$、$\overline{RD}$、READY 信号的功能。

12. 在基于 8086 的微机系统中，存储器是如何组织的？它是如何与处理器总线连接的？$\overline{BHE}$信号起什么作用？

13. “80386 是一个 32 位微处理器”，这句话的含义主要指的是什么？

14. 80386 内部结构由哪几部分组成？简述各部分的作用。

15. 80386 有几种存储器管理模式？都是什么？

16. 在不同的存储器管理模式下，80386 的段寄存器的作用是什么？

17. 试说明虚拟存储器的含义，它与物理存储器有什么区别？80386 虚拟地址空间有多大？

18. 试说明描述符的分类及各描述符的作用。

19. 描述符表的作用是什么？有几类描述符表？

20. 80386 的分段部件是如何将逻辑地址变为线性地址的？

21. 80386 中如何把线性地址变为物理地址？

第3章　指令系统

计算机是通过执行指令序列来解决问题的，因而每种计算机都有一组指令集供给用户使用，这组指令集就称为计算机的指令系统。指令系统是表征一台计算机性能的重要因素，它的格式与功能不仅直接影响计算机的硬件结构，而且也直接影响系统软件和计算机的适用范围。不同计算机的指令系统包含的指令种类和数目也不同。目前的微型计算机的指令系统可以包括几百余种指令，一般常用指令包含算术运算类指令、逻辑运算指令、数据传送指令、控制转移指令等。

3.1　8086/8088的指令格式及寻址方式

汇编语言是一种接近于机器语言的低级计算机语言，在汇编语言中，助记符代替了操作码，而操作数部分也像机器语言一样需要指明具体位置，它具有机器语言相应的寻址方式，但表现形式不是二进制符号，而是数值、寄存器名、变量等。对有操作数的指令，在执行指令所规定的操作之前首先要寻找操作数。

3.1.1　指令格式

指令格式是指令字用二进制代码表示的结构形式。计算机中的指令由操作码字段和操作数字段两部分组成。操作码字段指示计算机所要执行的操作，操作数字段指出在指令执行操作过程的所需要的操作数。例如，加法指令除要指定做加法操作外，还需要提供加数和被加数。操作数字段可以是操作数本身，也可以是操作数地址或是地址的一部分，还可以是指向操作数地址的指针或其他有关的操作数信息。

汇编语言语句用符号或符号地址来表示操作数或操作数地址，它的操作码与机器指令是一一对应的。用助记符表达的指令格式通常为：

操作码	操作数	…	操作数

操作数字段可以有一个、两个或三个，通常称为一地址、二地址、三地址指令。例如，单操作数指令就是一地址指令，它只需要指定一个操作数，如加1指令只需要指出加1的操作数。大多数运算型指令可以使用三地址指令，即除给出参加运算的两个操作数的数值外，还要指出运算结果的存放地址。也可以用二地址指令，此时分别称两个操作数为源操作数和目的操作数，在指令执行前这两个操作数都是输入操作数，但是指令执行后将把运算结果存放到目的操作数的地址之中。80x86的大多数运算型指令就采用二地址指令，少数采用三地址指令。

指令的操作码字段在机器里的表示比较简单，只需对每一种操作指定确定的二进制代码就可以。指令的操作数字段的情况就比较复杂，如果操作数存放在寄存器中，则由于寄存

器的数量较少，因而需要指定的操作数的位数就较少；但如果操作数存放在内存里，那么一个存储单元的地址对 8086 CPU 就需要 20 位，对于 80386 CPU 及其后的机型则需要 32 位甚至 64 位。因此，采用科学的寻址方式就十分必要。

3.1.2 寻址方式

指令中操作的对象称为操作数。8086/8088 指令系统中，操作数分为两大类：数据操作数和转移地址操作数。寻找这些操作数的方式称为寻址方式，即指令中用于说明操作数或操作数所在地址的方法。

8086/8088 有 7 种基本的寻址方式：立即数寻址、寄存器寻址、直接寻址、寄存器间接寻址、寄存器相对寻址、基址加变址寻址和相对基址加变址寻址。

立即数寻址是操作数直接放在指令中，寄存器寻址是指操作数放在寄存器中。除此之外的 5 种寻址方式属于存储器寻址，用于说明操作数所在存储单元的地址。由于总线接口部件(BIU)能根据需要自动引用段寄存器得到段值，所以这 5 种方式也就是确定存放操作数的存储单元有效地址(EA)的方法。在利用这 5 种方法计算有效地址时，所得的结果被认为是一个无符号数。

1)立即数寻址方式

操作数包含在指令中，作为指令的一部分，跟在操作码后存放在代码前，这种操作数称为立即数。

立即数可以是 8 位，也可以是 16 位，按“高高低低”原则存放，即高位字节在高地址存储单元，低位字节在低地址存储单元。

例如：

```
MOV     BL,80H
MOV     AX,1090H
```

指令执行情况如图 3-1 所示。执行结果为：(BL)＝80H，(AX)＝1090H。

说明：立即数寻址方式只能作为源操作数，主要用来给寄存器或存储单元赋值。

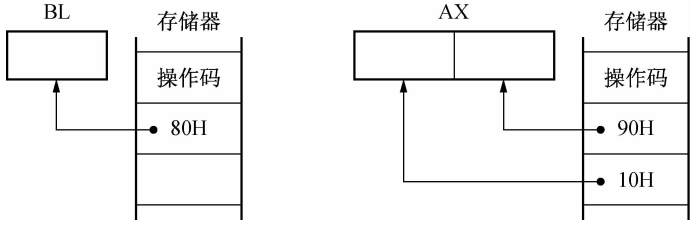


图 3-1 立即数寻址方式的存储和执行示意图

2)寄存器寻址方式

该寻址方式的操作数在 CPU 内部的寄存器中，指令中指定寄存器号。对于 16 位操作数，寄存器可以是 AX、BX、CX、DX、SI、DI、SP 和 BP 等；对于 8 位操作数，寄存器可以是 AL、AH、BL、BH、CL、CH、DL 和 DH。

例如，指令“MOV SI，AX”和指令“MOV AL，DH”中的源操作数和目的操作数均是寄存器寻址。再如，指令“MOV CL，DL”和“MOV AX，BX”，如果(DL)＝50H，(BX)＝1234H，则执行结果为：(CL)＝50H，(AX)＝1234H。寄存器寻址方式由于操作数就在寄存

器中，不需要访问存储器来取得操作数，因而可以取得较高的运行速度，通常用于 CPU 内部操作。

3)直接寻址方式

该寻址方式的操作数一般存放在存储器的数据段，直接寻址的有效地址(EA)(16 位偏移量)在指令的操作码后面直接给出，它与指令的操作码一起，存放在存储器的代码段中，也是高位字节存放在高地址中，低位字节存放在低地址中。但是，操作数本身一般存放在存储器的数据段中。所以操作数的地址由段基址加上指令中直接给出的 16 位偏移地址得到。如果采用段超越前缀，则操作数也可含在数据段外其他段中。

例如：

```
MOV   AL,[1064H]
```

如果(DS)＝2000H，DS 值左移一位，产生的地址为 20000H，则指令执行情况如图 3-2 所示，执行结果为(AL)＝45H。

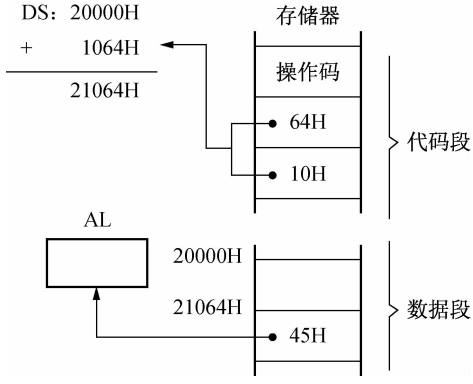


图 3-2　直接寻址方式指令的执行情况

如果没有特殊指明，直接寻址方式的操作数一般在存储器的数据段，即隐含的段寄存器是 DS。但是 8086/8088 也允许段超越前缀，即允许使用 CS、SS 或 ES 作为段寄存器，此时需要在指令中特别标明。方法是在有关操作数的前面写上超越的段寄存器的名字，再加上冒号。例如，若以上指令使用 ES 作为段寄存器，则指令应表示成为以下形式：

```
MOV   AL,ES:[1064H]
```

这种寻址方式常用于处理单个存储器变量的情况。它可实现在 64 KB 的段内寻找操作数。直接寻址的操作数通常是程序使用的变量。

4)寄存器间接寻址方式

操作数在存储器中，操作数有效地址存放在 SI、DI、BX、BP 这 4 个寄存器中。在一般情况(即不使用段超越前缀明确指定段寄存器)下，如果有效地址在 SI、DI 和 BX 中，则以 DS 段寄存器的内容为段值；如果有效地址在 BP 中，则以 SS 段寄存器的内容为段值。

寄存器间接寻址方式指令的书写应加方括号，避免与一般的寄存器寻址方式混淆。例如：

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

如果(DS)＝3000H，(SI)＝2000H，(BX)＝1000H，(AL)＝64H，则指令执行情况如图 3-3所示，执行结果为：(AX)＝4050H，(31000H)＝64H。

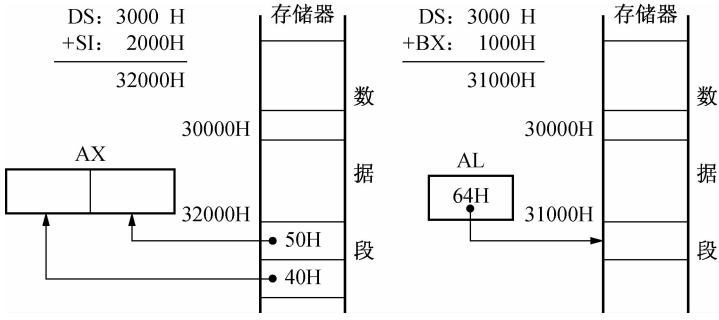


图 3-3 寄存器间接寻址方式指令的执行情况

5)寄存器相对寻址方式

操作数在存储器中，操作数的有效地址是一个基址寄存器(BX、BP)或变址寄存器(SI、DI)的内容加上指令中给定的 8 位或 16 位位移量之和。

在一般情况(即不使用段超越前缀明确指定段寄存器)下，如果使用 SI、DI 或 BX 的内容作为有效地址的一部分，那么引用的段寄存器是 DS；如果使用 BP 的内容作为有效地址的一部分，那么引用的段寄存器是 SS。比如，“MOV BX,[BP－4]”指令中，源操作数采用寄存器相对寻址，引用的段寄存器是 SS；再如，“MOV ES:[BX＋5],AL”指令中，目的操作数采用寄存器相对寻址，引用的段寄存器是 ES。

例如：

```
MOV   [SI＋10H],AX
MOV   CX,[BX＋COUNT]
```

如果(DS)＝3000H，(SI)＝2000H，(BX)＝1000H，COUNT＝1050H，(AX)＝4050H 则指令执行情况如图 3-4 所示。执行结果为：(32010H)＝4050H，(CX)＝4030H。

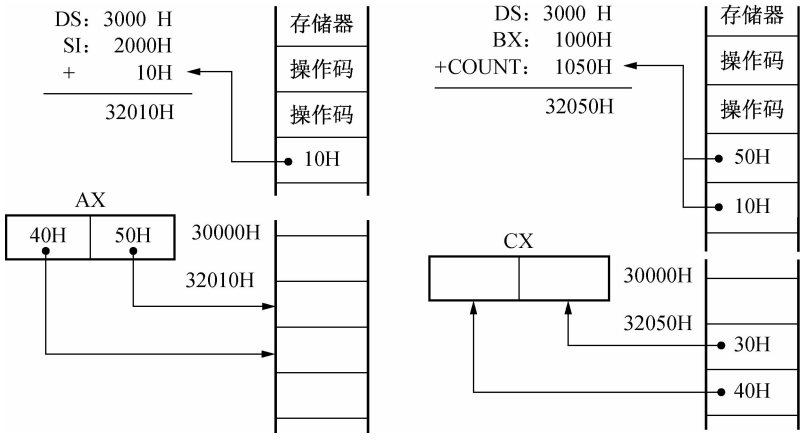


图 3-4 寄存器相对寻址方式指令的执行情况

6)基址加变址寻址方式

操作数在存储器中，操作数的有效地址由基址寄存器之一的内容与变址寄存器之一的内容相加得到。即在一般情况(不使用段超越前缀明确指定段寄存器)下，如果以 BP 的内容作为有效地址的一部分，则以 SS 的内容为段值，否则以 DS 的内容为段值。

例如：

```
MOV   AX,[BX+DI]
```

假设(DS)=5000H,(BX)=1223H,(DI)=54H,那么,存取的物理存储单元地址是51277H。再设该字存储单元的内容是0168H,那么,在执行该指令后,(AX)=0168H。

下面指令中,源操作数采用基址加变址寻址,通过增加段超越前缀来引用段寄存器ES:

```
MOV   AX,ES:[BX+SI]
```

下面指令中,目的操作数采用基址加变址寻址,引用的段寄存器是DS:

```
MOV   DS:[BP+SI],AL
```

基址加变址这种寻址方式适用于数组或表格处理。用基址寄存器存放数组首地址,而用变址寄存器来定位数组中的各元素,或反之。由于两个寄存器位置都可以改变,所以能更加灵活地访问数组或表格中的元素。

7)相对基址加变址寻址方式

操作数在存储器中,操作数的有效地址由基址寄存器之一的内容、变址寄存器之一的内容、指令中给定的8位或16位位移量相加得到。也就是说,相对基址加变址寻址方式的有效地址是由指令中指定的8位或16位位移量(disp)、一个基址寄存器内容和一个变址寄存器内容之和组成。即：

```
[BX]+[SI]+[8 位 disp]
```

或

```
[BP]+[DI]+[16 位 disp]
```

例如：

```
MOV   AH,[BX+DI+1234H]
```

说明:方括号中BX是基址寄存器,DI是变址寄存器,1234H是16位位移量(disp)。

相对基址加变址这种寻址方式最复杂,但也最灵活。其寻址方式的表示方法多种多样,下面的表示方法是等价的：

- `MOV   AX,[BX+DI+1234H]`
- `MOV   AX,1234H[BX+DI]`
- `MOV   AX,1234H[BX][DI]`
- `MOV   AX,1234H[DI][BX]`

3.2 微处理器的基本指令系统

80x86的指令系统大致分为数据传送指令、算术运算类指令、逻辑运算指令、移位指令、串处理指令、控制转移指令和处理机控制指令。

3.2.1 数据传送指令

数据传送是机器内部最基本的操作之一。这些指令不仅能实现寄存器之间、寄存器与内存之间、寄存器与I/O端口之间的字节或字数据的传送,而且能传送目标地址和状态标志以及完成堆栈的操作。数据传送指令负责把数据地址或立即数传送到寄存器或存储单元中,它又可以分为以下5种。

1.通用数据传送指令

1)MOV 传送指令

格式:MOV DST,SRC

功能:将源操作数 SRC 送到目的操作数 DST 中。

操作:(DST)←(SRC)

需要注意以下几点:

- 目的操作数 DST 和源操作数 SRC 不能同时用内存寻址方式,这个限制适用于所有指令。
- 目的操作数 DST 不能是 CS,也不能用立即数寻址方式。
- MOV 指令不影响标志位。
- MOV 允许传送字或双字。

MOV 指令可以在 CPU 内或 CPU 和内存之间传送字或字节。它传送信息的形式如下:

(1)寄存器到寄存器。例如:

```
MOV    AX,BX          ;将 BX 中的内容送到 AX 中
```

(2)寄存器或内存到段寄存器(CS 除外)。例如:

```
MOV  DS,DATA          ;将 DATA(内存单元)的内容送到 DS 中
```

(3)寄存器到内存。例如:

```
MOV    TABLE,AX       ;将 AX 中的内容送到 TABLE(内存单元)中
```

(4)段寄存器到寄存器或内存。例如:

```
MOV    AX,DS          ;将段寄存器 DS 的内容送到 AX 中
```

(5)内存到寄存器。例如:

```
MOV    AX,Y[BP][SI]   ;将基地址为 SS 的值左移 4 位+(BP)+(SI)+位移量 Y 的
                      ;存储单元的内容送给 AX 寄存器
```

(6)立即数到寄存器。例如:

```
MOV    AX,03H         ;将立即数 03H 送到 AX 中
```

(7)立即数到内存。例如:

```
MOV    BYTE PTR [2000H],12H    ;将立即数 12H 送到 2000H 中
```

2)入栈指令

格式:PUSH SRC

功能:将操作数压入堆栈中。

操作:16 位指令 (SP)←(SP)-2

((SP)+1,(SP))←(SRC)

32 位指令 (ESP)←(ESP)-4

((ESP)+3,(ESP)+2,(ESP)+1,(ESP))←(SRC)

3)出栈指令

格式:POP DST

功能:将数据从堆栈中取出送到 DST 指定的地址。

操作:16 位指令 (DST)←((SP)+1,(SP))

(SP)←(SP)+2

32位指令　　(DST)←((ESP)+3,(ESP)+2,(ESP)+1,(ESP))

(ESP)←(ESP)+4

PUSH和POP指令分别将数据存入堆栈或把堆栈中的数据取出。堆栈是以“后进先出”(LIFO)方式工作的一个存储区,程序中定义的堆栈段就是这样一个LIFO存储区。数据存入堆栈单元或从堆栈单元中取出都由堆栈指针SP指示,而SP总是指向栈顶,所以进栈和出栈指令都会自动修改SP。

PUSH指令执行时,SP的内容先减2,然后将数据压入SP所指示的字单元,存储的方法是高8位存入高地址字节,低8位存入低地址字节;POP指令执行时,将SP所指示的栈顶地址的内容取出放入目的地址,然后SP增2,指向新的栈顶地址。

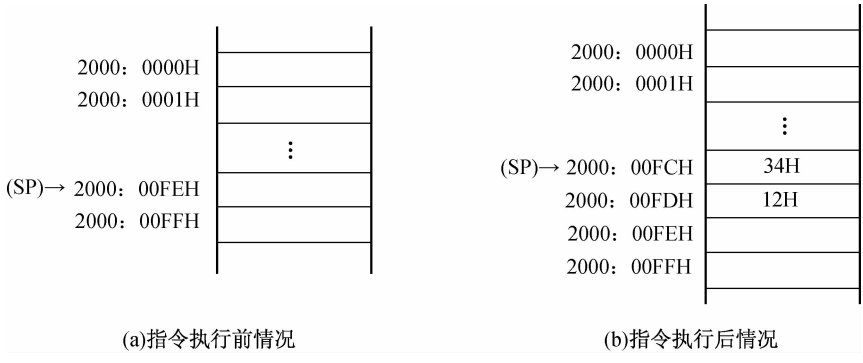
需要注意以下几点:

- PUSH和POP指令允许字操作或双字操作,因此存取数据后,堆栈指针寄存器的修改必须是±2或±4。
- PUSH和POP指令不能使用立即数寻址方式。
- POP指令的DST不允许是CS寄存器。
- PUSH和POP指令都不影响标志位。

【例3-1】 通用数据传送指令示例。

```
MOV  AX,1234H
PUSH  AX
```

设执行前(SS)=2000H,(SP)=00FEH。指令执行过程如图3-5所示。执行后(SS)=2000H,(SP)=00FCH。



(a)指令执行前情况　　(b)指令执行后情况

图3-5　压栈操作示意图

2.交换指令

格式:XCHG　OPR1,OPR2

功能:使两个操作数OPR1和OPR2互相交换,其中一个操作数必须在寄存器中,另一个操作数可以在寄存器或内存中。

操作:(OPR1)←→(OPR2)

需要注意以下几点:

- 不允许使用段寄存器。
- 不能在两个内存之间传送数据。
- 不影响标志位。

• 不能用立即数寻址方式。

• 允许字或字节操作。

【例 3-2】 已知(AX)=6634H,(BX)=0F24H,(SI)=0012H,(DS)=1200H,(12F36H)=2500H,写出下列指令执行的结果。

```
XCHG    AH,AL          ;执行前:(AH)=66H,(AL)=34H;
                       ;执行后:(AH)=34H,(AL)=66H
XCHG    AX,[BX+SI]     ;执行前:(AX)=6634H,[BX+SI]的地址单元(12F36H)=2500H;
                       ;执行后:(AX)=2500H,[BX+SI]的地址单元(12F36H)=6634H
```

3. 地址传送指令

1)LEA 指令

格式:LEA REG,SRC

功能:取有效地址送到寄存器。

操作:(REG)←(SRC)

需要注意以下几点:

• 源操作数可以是除寄存器和立即数以外的任意一种内存寻址方式。

• 目的操作数可以是 16 位和 32 位,但不能用段寄存器。

• 不影响标志位。

比如:

```
LEA    DX,VALUE
MOV    AX,VALUE
```

若执行前,VALUE 的有效地址为 0100H,以 VALUE 为符号地址的字单元的内容为 3056H,则执行后(DX)=0100H,(AX)=3056H。

2)LDS 指令

格式:LDS REG,SRC

功能:把源操作数指定的内存中的字节单元中的低地址中的字送到指定的寄存器中,高地址中的字送到 DS 寄存器中。

操作:(REG)←(SRC)

注意以下几点:

• SRC 不能是段寄存器。

• SRC 必须使用内存寻址方式。

• 不影响标志位。

比如:

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

若执行前(DS)=0C00H,(BX)=0200H,(0C000H)=2A35H,(0C202H)=4000H,则执行后(DS)=4000H,(SI)=2A35H。

3)LES 指令

格式:LES REG,SRC

功能:把源操作数指定的内存中的字节单元中的低地址中的字送到指定的寄存器中,高地址中的字送到 ES 寄存器中。

操作:(REG)←(SRC)

注意以下几点：

- SRC 不能是段寄存器。
- SRC 必须使用内存寻址方式。
- 不影响标志位。

比如：

```
LES  DI,[BX]
```

若执行前(DS)＝B000H,(BX)＝0020H,(B0020H)＝0076H,(B0022H)＝1F00H,则执行后(ES)＝1F00H,(DI)＝0076H。

4.状态标志位传送指令

有4条专门传送状态标志位的指令 LAHF、SAHF、PUSHF 和 POPF。

1)LAHF 指令

格式：LAHF

功能：将标志送到 AH 中。

操作：(AH)←(FLAGS 的低字节)

2)SAHF 指令

格式：SAHF

功能：将 AH 中的内容送到 FLAGS 的低8位。

操作：(FLAGS 的低字节)←(AH)

3)PUSHF/POPF 指令

格式：PUSHF

　　POPF

功能：标志寄存器进栈和出栈。

操作：PUSHF　(SP)←(SP)－2

　　((SP)＋1,(SP))←(FLAGS)

　　POPF　(FLAGS)←((SP),(SP)＋1)

　　(SP)←(SP)＋2

注意以下几点：

- LAHF 和 PUSHF 不影响标志位。
- POPF 会以弹出值设置标志寄存器值,SAHF 由新装入的值确定标志寄存器的值。

5.专用传送指令

专用传送指令主要限于使用累加器 EAX、AX 或 AL,与计算机 I/O 接口之间传送信息。

1)IN 输入指令

长格式：IN　AL,PORT(字节)

　　IN　AX,PORT(字)

　　IN　EAX,PORT(双字)

操作：(AL)←(PORT)(字节)

　　(AX)←(PORT＋1,PORT)(字)

　　(EAX)←(PORT＋3,PORT＋2,PORT＋1,PORT)(双字)

短格式：IN　AL,DX(字节)

IN AX,DX(字)
IN EAX,DX(双字)

操作:(AL)←((DX))(字节)
(AX)←((DX)+1,(DX))(字)
(EAX)←((DX)+3, (DX)+2,(DX)+1,(DX))(双字)

2)OUT 输出指令

长格式:OUT PORT,AL(字节)
OUT PORT,AX (字)
OUT PORT,EAX (双字)

操作:(PORT)←(AL)(字节)
(PORT+1,PORT)←(AX)(字)
(PORT+3,PORT+2,PORT+1,PORT)←(EAX)(双字)

短格式:OUT DX,AL (字节)
OUT DX,AX (字)
OUT DX,EAX (双字)

操作:((DX))←(AL)(字节)
((DX)+1,(DX))←(AX)(字)
((DX)+3, (DX)+2,(DX)+1,(DX))←(EAX)(双字)

在 80x86 中,所有 I/O 端口与 CPU 之间的通信都由 IN 和 OUT 指令来完成。IN 完成从端口到 CPU 的信息传送,而 OUT 完成从 CPU 到端口的信息传送。CPU 只能用累加器接收和发送数据。外部设备最多可能有 65 536 个端口,端口号为 0000H～0FFFFH。其中前 256(不含 256)个端口(0～0FFH)可以直接在指令中指定,这就是长格式中的 PORT;当端口号大于等于 256 时,只能使用短格式,此时端口号必须放到 DX 寄存器中,然后再用 IN 或 OUT 指令来完成传送信息。IN 和 OUT 指令提供了双字、字和字节 3 种使用方式,具体用哪一种方式,则取决于外设端口号的宽度。若端口宽度中有 8 位,则只能用字节指令传送信息。输入、输出指令不影响标志位。

比如:

```
IN      AX,28H
MOV     DATA_WORD,AX    ;把 28 号端口的内容经 AX 输入存储单元 DATA_WORD 中
```

3.2.2 算术运算类指令

80x86 提供加、减、乘和除 4 种基本算术运算操作。这些操作都可用于字节或字的运算,也可以用于无符号数的运算或有符号数的运算。有符号数用补码表示,因此加减运算指令不再分为无符号数运算指令和有符号数运算指令,而乘除运算指令还分为无符号数运算指令和有符号数运算指令。另外,80x86 还提供了各种十进制算术运算调整指令。

1. 加法指令

1)普通加法指令 ADD

格式:ADD OPRD,OPRS

功能:完成两个操作数相加,结果送至目的操作数 OPRD。

操作：(OPRD)←(OPRD)＋(OPRS)(用的是各个字符指的内容，因而要加上括号)

注意以下几点：

- 如果参与运算的操作数有两个，则它们的类型必须一致，即同时为字节，或同时为字。
- 源操作数可以存放在通用寄存器或存储器中，也可以是立即数。
- 目的操作数只能在寄存器或存储单元中，不能是立即数。
- 对无符号数和有符号数的处理须对应一致，即如果作为无符号数则影响标志 CF 和 AF，如果作为有符号数则影响标志 OF 和 SF，同时都会影响标志 ZF 和 PF。

2)带进位加指令 ADC

格式：ADC　　OPRD，OPRS

功能：与 ADD 指令类似，完成两个操作数相加，但还要把进位标志 CF 的当前值加上去，把结果送到目的操作数 OPRD。

操作：(OPRD)←(OPRD)＋(OPRS)＋CF

注意：ADC 与 ADD 指令相比，唯一的差别是，若在 ADC 指令前 CF＝1，则在两个操作数相加之后要再加 1。

ADC 指令主要用于多字节运算。尽管在 8086/8088 中可以进行 16 位运算，但 16 位二进制数运算能表示的整数的范围还是很有限的，为了扩大数的范围，仍然需要多字节运算，例如，有两个 4 字节的数相加，加法要分两次进行，先进行低两字节相加，然后再进行高两字节相加，在高两字节相加时，还要把低两字节相加以后可能出现的进位考虑进去，用 ADC 指令实现这点很方便。

3)INC 指令

格式：INC　　OPRD

功能：完成对操作数 OPRD 加 1，然后把结果回送 OPRD。

操作：(OPRD)←(OPRD)＋1

注意以下几点：

- 操作数 OPRD 可以是通用寄存器，也可以是存储单元。
- 这条指令执行的结果影响标志 ZF、SF、OF、PF 和 AF，但它不影响 CF。
- 该指令主要用于调整地址指针和计数器。

比如：

```
MOV    SI,3000H
INC    SI
```

指令执行后(SI)＝3001H。

2.减法指令

1)普通减法指令 SUB

格式：SUB　　OPRD，OPRS

功能：完成两个操作数相减，从 OPRD 中减去 OPRS，结果送到目标操作数 OPRD 中。

操作：(OPRD)←(OPRD)－(OPRS)

注意事项同 ADD 指令。

2)带借位减法指令 SBB

格式：SBB　　OPRD，OPRS

功能:与 SUB 指令类似,但在操作数 OPRD 减去操作数 OPRS 的同时还要减借位标志 CF 的当前值。

操作:(OPRD)←(OPRD)－(OPRS)－CF

3)减 1 指令 DEC

格式:DEC　　OPRD

功能:这条指令将操作数 OPRD 减 1,并把结果回送 OPRD。

操作:(OPRD)←(OPRD)－1

注意以下几点:

- 操作数 OPRD 可以是通用寄存器,也可以是存储单元。
- 在相减时,把操作数作为一个无符号数对待。
- 这条指令执行的结果影响标志 ZF、SF、OF、PF 和 AF,但它不影响 CF。
- 该指令主要用于调整地址指针和计数器。

4)取补指令 NEG

格式:NEG　　OPRD

功能:这条指令对操作数取补,就是用 0 减去操作数 OPRD,再把结果回送 OPRD。

操作:(OPRD)←0－(OPRD)

注意以下几点:

- 在字节操作时对－128 取补,或在字操作时对－32 768 取补,则操作数没有变化,但 OF 被置位。
- 操作数可以是通用寄存器,也可以是存储单元。
- 此指令的执行结果影响 CF、ZF、SF、OF、AF 和 PF,一般总使 CF 为 1,除非操作数为 0。

5)比较指令 CMP

格式:CMP　　OPRD,OPRS

功能:完成操作数 OPRD 减去操作数 OPRS,运算结果不送到 OPRD,但影响标志 CF、ZF、SF、OF、AF 和 PF。

操作:(OPRD)－(OPRS)

需要注意的是,比较指令主要用于比较两个数的关系,是否相等,谁大谁小。在执行了比较指令后,可根据 ZF 是否置 1 来判断两数是否相等,或者根据 SF 和 OF 判断两数大小。

3.乘除指令

1)无符号数乘法指令 MUL

格式:MUL　　OPRD

功能:如果 OPRD 是字节操作数,则把 AL 中的无符号数与 OPRD 相乘,16 位结果送到 AX 中;如果 OPRD 是字操作数,则把 AX 中的无符号数与 OPRD 相乘,32 位结果送到 DX 和 AX 中,DX 含高 16 位,AX 含低 16 位。所以由操作数 OPRD 决定是字节相乘,还是字相乘。

操作:字节操作　　(AX)←(AL)×(OPRD)

　　字操作　　(DX,AX)←(AX)×(OPRD)

注意以下几点:

- 如果乘积结果的高半部分(字节相乘时为 AH,在字相乘时为 DX)不等于零,则标志

CF=1 和 OF=1,否则 CF=0,OF=0。所以如果 CF=1 和 OF=1,表示在 AX 或 DX 中含有结果的有效数值。

- 该指令对其他标志位无定义。
- 一个操作数总是隐含在寄存器 AL(8 位数相乘)或者 AX(16 位数相乘)中,另一个操作数可以采用除立即数方式以外的任意一种寻址方式。

2)带符号数乘法指令 IMUL

格式:IMUL OPRD

功能:这条指令把被乘数和乘数均作为有符号数,此外与指令 MUL 操作类似。

操作:字节操作 (AX)←(AL)×(OPRD)

字操作 (DX,AX)←(AX)×(OPRD)

注意以下几点:

- 如果乘积结果的高半部分(字节相乘时为 AX,在字相乘时为 DX)不是低半部分的符号扩展,则标志 CF=1,OF=1,否则 CF=0,OF=0。所以如果 CF=1 和 OF=1 表示在 AH 或 DX 中含有结果的有效数值。
- 该指令对其他标志位无定义。
- 在 IMUL 指令中,一个操作数总是隐含在寄存器 AL(8 位数相乘)或者 AX(16 位数相乘)中,另一个操作数可以采用除立即数方式以外的任意一种寻址方式。

3)无符号数除法指令 DIV

格式:DIV OPRD

功能:如果 OPRD 是字节操作数,则把 AX 中的无符号除数以 OPRD,8 位的商送到 AL 中,8 位的余数送到 AH 中;如果 OPRD 是字操作数,则把 DX(高 16 位)和 AX 中无符号数的组合数除以 OPRD,16 位的商送到 AX 中,16 位的余数送到 DX 中。所以由操作数 OPRD 决定是字节除,还是字除。

操作:字节操作 (AL)←(AX)/(OPRD)的商

(AH)←(AX)/(OPRD)的余数

字操作 (AX)←(DX,AX)/(OPRD)的商

(DX)←(DX,AX)/(OPRD)的余数

注意以下几点:

- 如果除数为 0,或者在 8 位数除时商超过 8 位,或者在 16 位除时商超过 16 位,则认为是除法溢出引起 0 号中断。
- 除法指令对标志位无定义。
- 在除法指令中,被除数总是隐含在寄存器 AX(除数是 8 位)或者 DX 和 AX(除数是 16 位)中,另一个操作数可以采用除立即数方式外的任意一种寻址方式。

4)带符号数除法指令 IDIV

格式:IDIV OPRD

功能:这条指令把被除数和除数均作为有符号数,此外与指令 DIV 操作类似。

操作:字节操作 (AL)←(AX)/(OPRD)的商

(AH)←(AX)/(OPRD)的余数

字操作 (AX)←(DX,AX)/(OPRD)的商

(DX)←(DX,AX)/(OPRD)的余数

注意以下几点：

- 如果除数为 0，或者在 8 位数除时商超过 8 位，或者在 16 位除时商超过 16 位，则认为是除法溢出引起 0 号中断。
- 除法指令对标志位无定义。
- 在 IDIV 指令中，被除数总是隐含在寄存器 AX(除数是 8 位)或者 DX 和 AX(除数是 16 位)中，另一个操作数可以采用除立即数方式外的任意一种寻址方式。

3.2.3 逻辑运算指令

逻辑运算指令用来对字或字节按位进行逻辑运算，包括逻辑与 AND、逻辑或 OR、逻辑非 NOT、逻辑异或 XOR 和测试 TEST 这 5 条指令。

1. 逻辑与指令 AND

格式：AND DST,SRC

功能：对两个操作数执行按位的逻辑与运算，即只有相“与”的两位都是 1，结果才是 1；否则，“与”的结果为 0。逻辑与的结果送到目的操作数。

操作：(DST)←(DST)∧(SRC)

注意以下几点：

- 源操作数可以是任意寻址方式。
- 目的操作数只能是立即数之外的其他寻址方式。
- 两个操作数不能同时为内存寻址方式。
- 所有双操作数的逻辑指令均设置 CF＝OF＝0，根据结果设置 SF、ZF 和 PF 状态，而对 AF 未定义。

2. 逻辑或指令 OR

格式：OR DST,SRC

功能：对两个操作数执行按位的逻辑或运算，即只要相“或”的两位有一位是 1，结果就是 1。结果送到目的操作数。

操作：(DST)←(DST)∨(SRC)

注意事项同逻辑与指令 AND。

3. 逻辑异或指令 XOR

格式：XOR DST,SRC

功能：对两个操作数执行按位的逻辑异或运算，相异或的两位不相同时，结果就是 1，否则，异或的结果为 0。结果送到目的操作数。

操作：(DST)←(DST)⊕(SRC)

注意事项同逻辑与指令 AND。

4. 逻辑非指令 NOT

格式：NOT OPR

功能：对操作数按位求反，NOT 指令是一个单操作数指令，结果送回操作数。

操作：(OPR)←(～OPR)

注意以下几点：

• 操作数可以是任意寻址方式。

• 不影响标志位。

5. 测试指令 TEST

格式：TEST　DST,SRC

功能：对两个操作数执行按位的逻辑与运算，但结果不回送目的操作数，只根据结果来设置状态标志。

操作：(DST) ∧ (SRC)

3.2.4　移位指令

移位指令包含有逻辑移位、算术移位和循环移位3种类型6条指令。

1. 逻辑左移位

格式：SHL　OPR,CNT

功能：SHL指令向左逐位移动CNT次，每次移动后，最低位用0来补充，最高位移入CF，如图3-6所示。

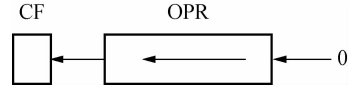


图3-6　逻辑左移

注意以下几点：

(1)目的操作数OPR可以是除立即数外的任何寻址方式。

(2)移位次数CNT＝1时，1可以直接写在指令中，CNT＞1时，CNT必须放入CL寄存器中。

(3)指令对标志位的影响：CF＝移入的数值。CF＝1，操作数左移的最高位的值为1，CF＝0，操作数左移的最高位的值为0。SF、ZF、PF根据移动后的结果设置。

【例3-3】 已知(DS)＝7B00H，(SI)＝1B79H，(7CB79H)＝07H，执行以下指令：

```
MOV   CL,3
SHL   [SI],CL
```

指令执行过程：DS值7B00H左移一位加上SI中的1B79H，生成的物理地址为7CB79H，执行“SHL　[SI],CL”指令，就是将7CB79H单元中的数据07H左移3位，指令执行结果：(7CB79H)＝38H，CF＝0。

说明：程序设计中，连续测试循环移位指令执行后CF的值，可确定操作数中含有1或0的个数。另外，将算术逻辑移位指令和循环移位指令结合使用，可以实现多字节(多精度)数据的移位操作。

2. 逻辑右移位

格式：SHR　OPR,CNT

功能：SHR指令向右逐位移动CNT次，每次移动后，最高位用0来补充，最低位移入CF，如图3-7所示。

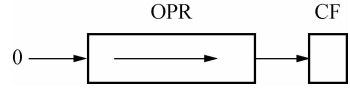


图3-7　逻辑右移

注意事项同逻辑左移指令(SHL)中的(1)、(2)。不同的是：指令对标志位的影响，CF＝移入的数值。CF＝1，移动后操作数最高位的值补0，最低位的1移走；CF＝0，移动时最高位的值补0，最末位的0移走。SF、ZF、PF根据移动后的结果设置。

比如：

```
MOV  AL,0F0H
SHR  AL,1    ;AL=78H,OF=1,CF=0,SF=0,ZF=0,PF=0
```

3.算术左移

格式：SAL OPR,CNT

功能：SAL 指令向左逐位移动 CNT 次，每次移动后，最低位用 0 来补充，最高位移入 CF，如图 3-8 所示。

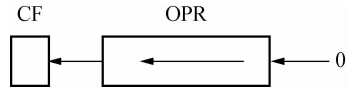


图 3-8　算术左移

注意事项同逻辑左移指令(SHL)中的(1)、(2)。不同的是指令对标志位的影响，CF＝移入的数值。CF＝1，操作数左移的最高位的值为 1，操作数的最高位补 1，最末位补 0。CF＝0，操作数左移的最高位的值为 0，操作数的最高位补 0，最末位也补 0。SF、ZF、PF 根据移动后的结果设置。

4.算术右移

格式：SAR OPR,CNT

功能：SAR 指令向右逐位移动 CNT 次，每次逐位移动后，最高位用符号位来补充，最低位移入 CF，如图 3-9 所示。

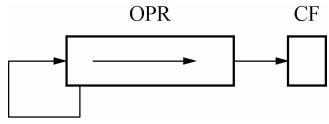


图 3-9　算术右移

注意事项同逻辑左移指令(SHL)中的(1)、(2)。不同的是：指令对标志位的影响，CF＝移入的数值。CF＝1，移动时操作数最高位的值补原来的二进制的值，最低位的 1 移走。CF＝0，移动时最高位的值补原来的二进制的值，最末位的 0 移走。SF、ZF、PF 根据移动后的结果设置。

5.循环左移

格式：ROL OPR,CNT

功能：ROL 对由 OPR 指定的寄存器或存储器操作数向左循环移动 CNT 所指定的次数，每左移一次，把最高位同时移入 CF 和操作数最低位，如图 3-10 所示。

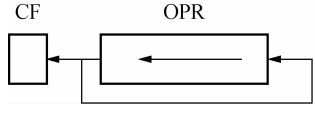


图 3-10　循环左移

注意事项同逻辑左移指令(SHL)中的(1)、(2)。不同的是：指令对标志位的影响，CF＝移入的数值。CF＝1，移动后操作数最高位的值 1 移到操作数最末位，最低位的值向前移动。CF＝0，移动时最高位的值 0 移到操作数最末位，最低位的值向前移动。SF、ZF、PF 根据移动后的结果设置。

6.循环右移

格式：ROR OPR,CNT

功能：ROR 对由 OPR 指定的寄存器或存储器操作数向右循环移动 CNT 所指定的次数，每右移一次，把最低位同时移入 CF 和操作数最高位，如图 3-11 所示。

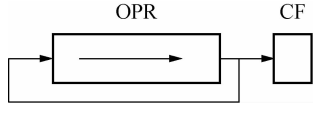


图 3-11　循环右移

注意事项同逻辑左移指令(SHL)中的(1)、(2)。不同的是：指令对标志位的影响，CF＝移入的数值。CF＝1，移动后操作数最末位的值 1 移到操作数最高位。CF＝0，移动后操作数最高位的值 0 移到操作数末位。

SF、ZF、PF 根据移动后的结果设置。

3.2.5 串处理指令

在计算机中,大部分数据要存放在主存中,不可能都存放在有限的寄存器中。80x86 提供了一组处理主存中连续存放的数据串指令——串处理指令。这类指令不仅可以传送数据串,还可以对数据串进行一些操作,如比较、搜索、赋值等;这些操作的对象可以是以字为单位的字串,也可以是以字节为单位的字节串。另外,80x86 还为串操作指令提供了重复前缀,以便重复进行相同的操作。

串处理指令中,源操作数用寄存器 SI 寻址,默认在数据段 DS 中,但允许段超越前缀;目的操作数用寄存器 DI 寻址,默认附加段 ES,不允许段超越前缀。每执行一次串处理指令,作为源操作数地址指针的 SI 和作为目的地址指针的 DI 将自动修改:1(对于字节串)或 2(对于字串)。地址指针是增加还是减少则取决于方向标志 DF。在系统初始化后或执行指令 CLD 指令(后续会讲到该指令)后,DF=0,此时地址指针增 1 或 2;在执行指令 STD 指令(后续会讲到该指令)后,DF=1,此时地址指针减 1 或 2。

1. 串传送指令(MOVS)

格式:MOVS DST,SRC (该指令应在操作数中注明传送的是字节、字或双字)

MOVSB (字节)

MOVSW(字)

MOVSD(双字)

功能:该指令是把指针"DS:SI"所指向的字节、字或双字传送给指针"ES:DI"所指向的内存单元,并根据方向标志位 DF 的值对寄存器 DI 和 SI 作相应增减。

注意以下几点:

- 该指令的执行不影响任何标志位。
- 除了指示源串和目的串的地址外,还需指出操作是字节传送还是字传送的。

【例 3-4】 将数据段 Source 指示的 100 个字节数传送到附加数据段 Destination 指示的主存区。

程序段如下:

```
    MOV   SI,OFFSET Source        ;把数据段 Source 首地址送到 SI 中
    MOV   DI,OFFSET Destination   ;把附加数据段 Destination 首地址送到 DI 中
    MOV   CX,100                  ;循环 100 次,传送 100 个字节数
A:  MOVSB                         ;按字节传送,每传送一次,SI 和 DI 的值增 1
    DEC CX                        ;每传送一次,循环次数 CX 值减 1
    JNZ A                         ;CX 值不等于 0,跳转到 A 继续执行
```

2. 串存储指令(STOS)

格式:STOS DST

STOSB (字节)

STOSW(字)

STOSD(双字)

功能:该指令是把寄存器 AL、AX 或 EAX 中的值存到以指针"ES:DI"所指向内存单元

为起始的一片存储单元里，并根据方向标志位 DF 的值对寄存器 DI 作相应增减。

注意以下几点：

- 该指令不影响任何标志位。
- 除了指出字符串的当前地址外，目的操作数还指示了从相应累加器中移出的是字节还是字或双字。

【例 3-5】 将附加段 64 KB 主存区全部设置为 0。

程序段如下：

```
    MOV    AX,0                ;AX 值初始为 0
    MOV    DI,0                ;附加段偏移首地址为 0
    MOV    CX,8000H            ;设置循环次数
    CLD                        ;指针寄存器 DI 地址自增
A:  STOSW                      ;按字传送
    DEC    CX                  ;每传送一次，循环次数 CX 值减 1
    JNZ    A                   ;CX 值不等于 0，跳转到 A 继续执行
```

3. 串载入指令(LODS)

格式：LODS　DST

LODSB(字节)

LODSW(字)

LODSD(双字)

功能：从指针“DS:SI”所指向的内存单元开始，取一个字节、字或双字进入 AL、AX 或 EAX 中，并根据方向标志位 DF 的值对寄存器 SI 作相应增减。

注意以下几点：

- 该指令的执行不影响任何标志位。
- 在指令 LODS 中，它会根据其地址表达式的属性来决定读取一个字节、字或双字，即当该地址表达式的属性为字节、字或双字时，将从指针“DS:SI”处读一个字节到 AL 中，或读一个字到 AX，或读一个双字到 EAX 中，与此同时，SI 还将分别增减 1，2 或 4。
- 除了指出字符串的当前地址外，源操作数还指出了送入累加器的是字节还是字或双字。

4. 串比较指令(CMPS)

格式：CMPS　DST,SRC

CMPSB(字节)

CMPSW(字)

CMPSD(双字)

功能：该指令是把指针“DS:SI”和“ES:DI”所指向字节、字或双字的值相减，并用所得到的差来设置有关的标志位。与此同时，变址寄存器 SI 和 DI 也将根据方向标志位 DF 的值作相应增减。

注意以下几点：

- 受影响的标志位：AF、CF、OF、PF、SF 和 ZF。
- 除了指示源串和目的串的地址外，操作数还指出操作是字节比较还是字比较或双字

比较。

5.串扫描指令(SCAS)

格式:SCAS DST
SCASB(字节)
SCASW(字)
SCASD(双字)

功能:该指令是用指针"ES:DI"所指向字节、字或双字的值与相应的 AL、AX 或 EAX 的值相减,用所得到的差来设置有关标志位。与此同时,变址寄存器 DI 还将根据标志位 DF 的值进行增减。

注意以下几点:

- 受影响的标志位:AF、CF、OF、PF、SF 和 ZF。
- 除了指示源串和目的串的地址外,操作数还指出操作是字节扫描还是字扫描或双字扫描。

6.重复前缀指令

格式:REP LODS/LODSB/LODSW/LODSD
REP STOS/STOSB/STOSW/STOSD
REP MOVS/MOVSB/MOVSW/MOVSD

功能:重复前缀指令是重复其后的字符串操作指令,重复的次数由 CX 来决定。每执行一次操作,CX 减 1,直到 CX 等于 0 为止。

执行步骤如下:

(1)判断:CX=0。

(2)如果 CX=0,则结束重复操作,执行程序中的下一条指令。

(3)否则,CX=CX-1(不影响有关标志位),并执行其后的字符串操作指令,在该指令执行完后,再转到步骤(1)。

7.条件重复前缀指令

条件重复前缀指令与重复前缀指令(REP)功能相类似,所不同的是:其重复次数不仅由 CX 来决定,还会由标志位 ZF 来决定。根据 ZF 所起的作用又分为:相等重复前缀指令 REPE/REPZ 和不等重复前缀指令 REPNE/REPNZ。

1)相等重复前缀指令

格式:REPE/REPZ SCAS/SCASB/SCASW/SCASD
REPE/REPZ CMPS/CMPSB/CMPSW/CMPSD

执行步骤如下:

(1)判断条件:CX≠0 且 ZF=1。

(2)如果条件不成立,则结束重复操作,执行程序中的下一条指令。

(3)否则,CX=CX-1(不影响有关标志位),并执行其后的字符串操作指令,在该指令执行完后,再转到步骤(1)。

2)不等重复前缀指令

格式为:REPNE/REPNZ SCAS/SCASB/SCASW/SCASD
REPNE/REPNZ CMPS/CMPSB/CMPSW/CMPSD

执行步骤如下：

(1)判断条件：CX≠0 且 ZF＝0。

(2)如果条件不成立，则结束重复操作，执行程序中的下一条指令。

(3)否则，CX＝CX－1(不影响有关标志位)，并执行其后的字符串操作指令，在该指令执行完后，再转到步骤(1)。

3.2.6 控制转移指令

在 80x86 中，程序的执行序列是由代码段寄存器 CS 和指令指针 IP 确定的。CS 包含当前指令在代码段的地址，IP 则是要执行的下一条指令的偏移地址。程序的执行一般是依指令序列顺序执行，但有时需要改变程序的流程。控制转移类指令通过修改 CS 和 IP 的值来改变程序的执行顺序，包括 5 组指令：无条件转移指令、条件转移指令、循环控制指令、子程序调用指令和中断指令。

1.无条件转移指令

无条件转移指令将无条件跳转到指定的地址去执行从该地址开始的指令。总体来说，转移指令可以分为两大类：段内转移和段间转移。段内转移是指在同一段的范围内进行的转移，此时只需要以改变 IP 或 EIP 寄存器的内容，即用新的目标转移地址代替原有的 IP 或 EIP 的值就可达到转移的目的。段间转移则是要转移到另一段去执行程序，此时不仅要修改 IP 或 EIP 寄存器内容，还需要修改 CS 段寄存器的内容才能达到目的。因此，此时的目标转移地址应由新的段地址和偏移地址两部分组成。

1)段内直接转移

格式：JMP　SHORT OPR　　；短转移

　　JMP　NEAR PTR OPR　　；近程转移

功能：采用相对寻址将当前 IP 值与 JMP 指令中给出的偏移量之和送 IP 中。

操作：(IP)←(IP)＋8 位位移量

　　(IP)←(IP)＋16 位位移量

注意以下几点：

- 段内短转移(SHORT)指令偏移量为 8 位，允许转移的偏移值范围为－128～＋127。
- 段内近程转移(NEAR)指令在 16 位指令模式下，偏移量为 16 位，允许转移的偏移值范围为-2^{15}～$2^{15}-1$。
- 在 32 位指令模式下，偏移值范围为-2^{31}～$2^{31}-1$。

这两条指令可以直接写成“JMP　OPR”，在这种格式中，用户不需要考虑是短转移还是近程转移，在汇编时汇编程序会计算出 JMP 下一条指令与目标地址间的相对偏移量，若偏移量在－128～127 字节之间，产生一个短转移，否则产生一个在－32 768～32 767 内的近程转移。

比如：

设 DS＝1000H，EBX＝00002000H。

```
JMP     BX              ;将 2000H 送 IP
JMP     NEAR PTR [BX]   ;将地址 1000H:2000H 单元存放的一个字送 IP
JMP     NEAR PTR [EBX]  ;将段选择符为 1000H,偏移地址为 00002000H 单元
                        ;存放的双字送 EIP
```

2)段内间接转移

格式:JMP　WORD PTR OPR

功能:段内间接转移,该指令地址在存储器中,默认段寄存器根据参与寻址的通用寄存器来确定,将指定存储单元的字取出直接送 IP 实现程序转移。

操作:(IP)←(EA)

注意以下几点:

- EA 值由 OPR 的寻址方式确定,但不能为立即数。
- 如果指定的是寄存器,则把寄存器的内容送到 IP 或 EIP 寄存器中。
- 如果指定的是存储器中的一个字或双字,则把存储单元对应的内容送到 IP 或 EIP 寄存器。

例如:

```
JMP [BX]
```

执行前:(BX)=2010H,(2010H)=0FA34H;执行后:(IP)=0FA34H。

3)段间直接转移

格式:JMP　FAR PTR OPR

功能:段间直接转移,FAR PTR 说明标号 OPR 具有远程属性。将指令中由 OPR 指定的段值送 CS,偏移地址送 IP。

操作:(IP)←OPR 的段内偏移地址

(CS)←OPR 所在的段的段基址

注意以下几点:

- 直接寻址方式,指令转移的段基址和偏移地址直接存放在指令的操作数字段中,所以只要用其中的段基址和偏移地址分别取代 CS 和 IP 寄存器中的内容就能实现转移。
- 在汇编格式中,OPR 可以使用符号地址,在机器语言中则要指定段基址和偏移地址。

4)段间间接转移

格式:JMP　DWORD PTR OPR

功能:段间间接转移,从存储单元取双字作为要转移的目标地址。

操作:(IP)←(EA)

(CS)←(EA+2)

注意以下几点:

- EA 由 OPR 的寻址方式确定,它可以是除立即数和寄存器外的任何存储器寻址方式。
- 目标地址存放在主存中连续的两个字单元中,低位字送到 IP 中,高位字送到 CS 中。

例如:

```
JMP  DWORD PTR BUF[BX]
```

执行前:(DS)=3000H,(BX)=0020H,(BUF)=0100H,(30120H)=1234H,(30122H)=5678H,(EA)=30 000H+0100H+0020H=30120H;执行后:(CS)=5678H,(IP)=1234H。

2.条件转移指令

该类指令是根据上一条指令对标志寄存器中标志位的影响来决定程序执行的流程,若满足指令规定的条件,则程序转移;否则程序顺序执行。

条件转移指令的转移范围为段内短转移或段内近程转移,不允许段间转移。段内短转

移(short)的转移偏移值范围为－128～＋127。

条件转移指令包括4类:单标志位条件转移、无符号数比较条件转移、带符号数比较条件转移和测试CX条件转移。

1)单标志位条件转移

(1)JZ(或JE)。

格式:JZ(或JE) OPR

功能:结果为0或相等则转移。

条件:ZF＝1。

(2)JNZ(或JNE)。

格式:JNZ(或JNZ) OPR

功能:结果不为0或不相等则转移。

条件:ZF＝0。

(3)JS。

格式:JS OPR

功能:结果为负则转移。

条件:SF＝1。

(4)JNS。

格式:JNS OPR

功能:结果为正则转移。

条件:SF＝0。

(5)JO。

格式:JO OPR

功能:溢出则转移。

条件:OF＝1。

(6)JNO。

格式:JNO OPR

功能:不溢出则转移。

条件:OF＝0。

(7)JP。

格式:JP OPR

功能:奇偶校验位为1则转移。

条件:PF＝1。

(8)JNP。

格式:JNP OPR

功能:奇偶校验位为0则转移。

条件:PF＝0。

(9)JB(或JC)。

格式:JB(或JC) OPR

功能:低于或不高于,或等于,或进位为1则转移。

条件:CF＝1。

(10)JNB(或 JNC)。

格式:JNB(或 JNC)　OPR

功能:不低于或高于,或等于,或进位为 0 则转移。

条件:CF=0。

2)无符号数比较条件转移

(1)JBE(或 JNA)。

格式:JBE(或 JNA)　OPR

功能:低于或等于,即不高于则转移。

条件:CF=1 或 ZF=1。

(2)JNB(或 JAE)。

格式:JNB(或 JAE)　OPR

功能:不低于,即高于或等于则转移。

条件:CF=0 且 ZF=0。

3)带符号数比较条件转移

(1)JL(或 JNGE)。

格式:JL(或 JNGE)　OPR

功能:小于,即不大于且不等于则转移。

条件:SF=1 或 OF=1。

(2)JNL(或 JGE)。

格式:JNL(或 JGE)　OPR

功能:不小于,即大于或等于则转移。

条件:SF=0 或 OF=0。

(3)JLE(或 JNG)。

格式:JLE(或 JNG)　OPR

功能:小于或等于,即不大于则转移。

条件:SF 和 OF 异号或 ZF=1。

(4)JNLE(或 JG)。

格式:JNLE(或 JG)　OPR

功能:大于,即不小于且不等于则转移。

条件:SF 和 OF 同号且 ZF=0。

4)测试 CX 条件转移

格式:JCXA

功能:CX 寄存器的内容为 0 则转移。

条件:(CX)=0。

3.其他控制指令

1)循环控制指令

格式:LOOP　　CC TARGET

功能:将(E)CX 内容减 1,不影响标志位,若(E)CX 不等于 0,且测试条件 CC 成立,则转移到目标地址 TARGET 处执行程序。转移范围为-128～+127。循环控制指令如表 3-1 所示。

表 3-1 循环控制指令

助 记 符	循环条件	说 明
LOOP	CX≠0	CX←CX－1,若 CX≠0 则循环
LOOPW	CX≠0	CX←CX－1,若 CX≠0 则循环
LOOPD	ECX≠0	ECX←ECX－1,若 ECX≠0 则循环
LOOPE/LOOPZ	CX≠0 且 ZF＝1	CX←CX－1,若 CX≠0 且 ZF＝1,则循环
LOOPEW/LOOPZW	ECX≠0 且 ZF＝1	CX←CX－1,若 CX≠0 且 ZF＝1,则循环
LOOPED/LOOPZD	ECX≠0 且 ZF＝1	ECX←ECX－1,若 ECX≠0 且 ZF＝1,则循环
LOOPNE/LOOPNZ	CX≠0 且 ZF＝0	CX←CX－1,若 CX≠0 且 ZF＝0,则循环
LOOPNEW/LOOPNZW	CX≠0 且 ZF＝0	CX←CX－1,若 CX≠0 且 ZF＝0,则循环
LOOPNED/LOOPNZD	ECX≠0 且 ZF＝0	ECX←ECX－1,若 ECX≠0 且 ZF＝0 则循环

2)子程序调用指令

子程序的调用和返回是一对互逆操作,也是一种特殊的转移操作。一方面,当调用一个子程序时,程序的执行顺序被改变,CPU 将转而执行子程序中的指令序列,当子程序执行完后,还要求 CPU 能转而执行主调用指令下面的指令,它是一种“有去有回”的操作。为了满足子程序调用和返回的操作的特殊性,在指令系统中设置了相应的调用和返回的特定指令。

(1)调用指令(CALL)。

格式:CALL OPRD

功能:程序转到 OPRD 地址处执行。

过程调用可分为段内调用和段间调用。寻址方式也可以分为直接寻址和间接寻址。本指令不影响标志位。

①段内直接调用。

格式:CALL NEAR 类型的过程名

功能:每一个过程在定义时,应指定它是近类型(NEAR),还是远类型(FAR)。本指令是段内直接调用,因而过程与调用指令同处在一个代码段内。在执行该调用指令时,首先将 IP 的原内容入栈保护,然后由指令代码给出的目的地址的段内偏移量送入 IP,从而实现过程调用,将程序转至过程入口。

②段内间接调用。

格式:CALL OPRD

功能:OPRD 为 16 位通用寄存器或存储器数。本指令执行时,首先将 IP 的内容入栈保护,然后将目的地址的段内偏移量由指定的 16 位寄存器或存储器送至 IP 中,从而实现过程调用。

注意:寄存器间接调用时,寄存器不用方括号括起来。如果用方括号,则为存储器操作数间接调用。

③段间直接调用。

格式:CALL FAR 类型的过程名

功能:段间直接调用时指令执行过程中应同时将当前的 CS 及 IP 的值入栈保护,然后将 FAR 类型的过程名所在的段基址和段内偏移值送 CS 及 IP,从而实现过程调用。

④段间间接调用。

格式：CALL　　DWOPRD

功能：DWOPRD 为存储器操作数，段间间接调用只能通过存储器双字进行。本指令执行时，首先将当前的 CS 及 IP 的值入栈保护，然后将存储器双字操作数的第一个字的内容送 IP，第二个字的内容送 CS，以实现段间调用。

(2)返回指令(RET)。

当子程序执行完时，需要返回到调用它的程序之中。为实现此功能，指令系统提供了一条专用的返回指令。

格式：RET/RETN/RETF [Imm]

功能：从堆栈中取出断点地址，送给 PC，并从断点处开始继续执行原程序。RET 一般应放在子程序的末尾。

3.2.7　处理机控制指令

1.标志处理指令

这一组指令用来设置或清除标志位，它们只影响本指令指定的标志位，而不影响其他标志位。

- CLC(clear carry)：进位位清 0 指令，执行操作后，CF＝0。
- STC(set carry)：进位位置 1 指令，执行操作后，CF＝1。
- CMC(complement carry)：进位位求反指令，执行操作后，CF＝!CF。
- CLD(clear direction)：方向标志位清 0 指令，执行后 DF＝0。
- STD(set direction)：方向标志位置 1 指令，执行后 DF＝1。
- CLI(clear interrupt)：中断标志清 0 指令，执行后 IF＝0。
- STI(set interrupt)：中断标志置 1 指令，执行后 IF＝1。

2.其他指令

1)NOP (no operation) 无操作指令

格式：NOP

功能：该指令不执行任何操作，其机器码占有一个字节单元，在调试程序时往往用这条指令占用一定的存储单元，以便在正式运行时用其他指令取代。

2)HLT (halt) 停机指令

格式：HLT

功能：该条指令可使计算机暂停工作，使处理机处于停机状态以等待下一次外部中断的到来，中断结束后可以继续执行下面的指令。

3)ESC (escape) 换码指令

格式：ESC　op，reg/mem

功能：这条指令在使用协处理机(coprocessor)时，可以指定由协处理机执行的指令。指令的第一个操作数即指定其操作码，第二个操作数即指定其操作数。协处理机(如 8087、80287、80387)是为了提高 CPU 速度而选配的硬件。

4)WAIT (wait) 等待指令

格式：WAIT

功能:该指令使处理机处于空转状态,它也可以用来等待外部中断发生,但中断结束后仍返回 WAIT 指令继续等待。它也可以与 ESC 指令配合等待协处理机的执行结果。

5)LOCK (lock) 封锁指令

格式:LOCK

功能:该指令是一种前缀,它可以与其他指令联合,用来维持总线的锁存信号直到与其联合的指令执行完为止。当 CPU 与其他处理机协同工作时,该指令可避免破坏有用信息。它可与下列指令联合使用:

```
BST/BTR/BTC                    mem,reg/imm
XCHG                           reg,mem
XCHG                           mem,reg
ADD/OR/ADC/SBB/AND/SUB/XOR     mem,reg/imm
NOT,NEG,INC,DEC                mem
CMPXCHG/XADD
```

6)BOUND (bound) 界限指令(286 及其后继机型可用)

格式:BOUND reg,mem

BOUND 指令检查给出的数组下标是否在规定的上下界之内。如在上下界之内,则执行下一条指令;如果超过了上下界范围,则产生内部 5 号中断。如发生中断,则中断返回时返回地址仍指向 BOUND 指令,而不是其下一条指令。指令的第一个操作数必须使用寄存器寻址方式,用来存放当前数组下标。第二个操作数必须使用存储器寻址方式,该操作数用来存放数组下标的上下界,下界存放在低地址指定的单元中,上界存放在高地址所指定的单元中。

例如:

```
BOUND       SI,DATA
MOV         AX,ARRAY[SI]
```

在 DATA 单元中,存放数组 ARRAY 的起始下标,DATA+2 单元中存放 ARRAY 的末尾下标,SI 中存放当前要访问数组元素的下标。

7)ENTER (enter) 建立堆栈帧指令(286 及其后继机型可用)

格式:ENTER imm16,imm8

指令中所给出的两个操作数均为操作数。第一个操作数为 16 位立即数,由其指定堆栈帧的大小,即其所占据的字节数。第二个操作数为 8 位立即数,它给出过程嵌套层数,此数的范围应为 0~31。ENTER 用于过程调用时建立堆栈帧,为过程间传递参数所用。堆栈帧是一个动态存储区。

8)LEAVE (leave) 释放堆栈帧指令(286 及其后继机型可用)

格式:LEAVE

LEAVE 指令在程序中位于 RET 指令之前,用来释放由 ENTER 指令建立的堆栈帧存储区。

操作:(SP)←(BP)或(ESP)←(EBP)

(BP)←POP()或(EBP)←POP()

本章小结

本章主要介绍了80x86的寻址方式和指令系统。

寻址方式和指令系统是汇编语言的基本知识，是学习汇编编程的基础。学习过程中要求读者理解为什么要寻址，通过相关实例重点掌握算术运算类指令、逻辑运算指令、数据传送指令、控制转移指令等类型指令的功能和应用方法，为学习第4章汇编语言程序设计打好基础。

习 题 3

1. 说明下列指令中源操作数的寻址方式。如果BX=2000H，DI=40H，给出DX的值或有效地址EA的值。

(1)MOV DX,[1234H]
(2)MOV DX,1234H
(3)MOV DX,BX
(4)MOV DX,[BX]
(5)MOV DX,[BX+1234H]
(6)MOV DX,[BX+DI]
(7)MOV DX,[BX+DI+1234H]

2. 指出下列指令的错误原因。

(1)AND AX,DL
(2)AND CS,DX
(3)MOV AX,IP
(4)MOV [BP][DI],[SI]
(5)SUB [BP][SI],ES:DX
(6)XCHG AL,[SI][DI]
(7)JGE AX
(8)PUSH DL

3. 判断下述各指令的正误。

(1)ADD [SI],[BX]
(2)ADD AX,100
(3)ADD AX,BL
(4)ADD [SI],100
(5)MUL 5
(6)DIV 5
(7)SHR AX,CX
(8)SHR AX,3

4. 给出下列程序段中各条指令执行后 AL 的值，以及 CF、ZF、SF、OF 和 PF 的状态。

```
MOV    AL,89H
ADD    AL,AL
ADD    AL,9DH
CMP    AL,0BCH
SUB    AL,AL
DEC    AL
INC    AL
```

5. 写出实现下列功能的指令序列。

(1)将 AL 与 DX 中的两个无符号数相加，结果放入 AX。

(2)将 AL 与 DX 中的两个带符号数相加，结果放入 AX。

6. 已知程序段如下：

```
MOV    AX,1234H
MOV    CL,4
ROL    AX,CL
DEC    AX
MOV    CX,4
MUL    CX
INT    20H
```

试问：

(1)每条指令执行完后，AX 寄存器的内容是什么？

(2)每条指令执行完后，CF、SF 的值各是什么？

(3)程序结束时，AX 和 DX 的内容是什么？

7. 用其他指令完成和下列指令一样的功能。

(1)REP　MOVSB

(2)REP　STOSB

(3)REPNE　SCASB

(4)REP　LODSB

(5)REPE　CMPSB

8. 判断下面每条指令的正误。

(1)REP　MOVS

(2)MOVSB

(3)REP　LODSB

(4)CMPS

(5)REPZ　SCASW

(6)REPE　CMPSB