软件工程

第一节 软件与软件工程

一、软件的定义与特点

（一）软件的定义

计算机系统是通过运行程序来实现各种不同的应用。把各种不同功能的程序，包括用户为自己的特定目的编写的应用程序、检查和诊断机器系统的程序、支持用户应用程序运行的系统程序、管理和控制机器系统资源的程序等通常称为软件。

（二）软件的组成

在运行中能提供所希望的功能和性能的指令集（即程序）；使程序能够正确运行的数据结构；描述程序研制过程、方法所用的文档。

（三）软件的特点

主要特点	具体描述
软件具有抽象特征	软件是一种逻辑实体，而不是具体的物理实体，因而它具有抽象性
软件是智力劳动的结果	软件是通过人们的智力活动，把知识与技术转化成信息的一种产品，是在研制、开发中被创造出来的
软件无备件特征	在软件的运行和使用期间，没有硬件那样的机械磨损、老化问题
软件对硬件有依存性	软件的开发和运行经常受到计算机系统的限制，对计算机系统有着不同程度的依赖性。在软件的开发和运行中必须以硬件提供的条件为基础
软件的研发主要由人工完成	软件的开发至今尚未完全摆脱手工的开发方式
软件无明显的制造过程	软件的开发费用越来越高，成本相当昂贵

二、软件工程

（一）软件危机

软件危机指的是软件开发和维护过程中遇到的一系列严重问题。

（二）软件工程

软件工程是指导计算机软件开发和维护的工程学科。采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，这就是软件工程。

软件工程是 1968 年在德国的 NATO 会议上提出的，希望用工程化的原则和方法来克服软件危机；而软件危机就是软件开发和维护过程中的各种问题，由于软件开发阶段缺乏好的方法的指导和好的工具的辅助，而且缺少有关的文档，使得大量的软件难以维护。

软件工程包括三个要素：方法、工具和过程。

（三）软件生存周期

一个软件从定义到开发、使用和维护，直到最终被废弃，要经历一个漫长的时期，通常把软件经历的这个漫长的时期称为生存周期。软件生存周期就是从提出软件产品开始，直到该软件产品被淘汰的全

过程。

软件生存周期（Software Life Cycle）又称为软件生命期、生存期。是指从形成开发软件概念起，所开发的软件使用以后，直到失去使用价值消亡为止的整个过程。

一般来说，整个生存周期包括计划（定义）、开发、运行（维护）三个时期，每一个时期又划分为若干阶段。每个阶段有明确的任务，这样使规模大、结构复杂和管理复杂的软件开发变得容易控制和管理。

软件生存周期的六个阶段是：定义及规划、需求分析、软件设计、程序编码、软件测试、运行维护。

软件生存期也可以分为三个大的阶段：计划阶段、开发阶段和维护阶段。

下表归纳了软件生存周期各个阶段的任务、参与人员和产生文档。

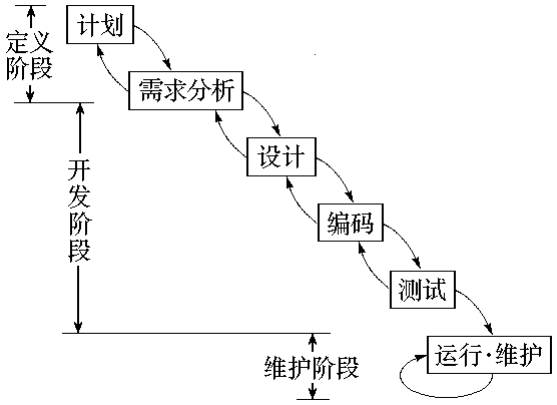
阶段		任务	参与人员	产生文档
软件计划阶段——待开发软件要“做什么”				
问题定义		确定要开发软件的目标	用户、项目负责人、系统分析员	可合并项目计划书中
可行性研究		研究软件开发的可行性	用户、项目负责人、系统分析员	可行性分析报告、项目计划书
需求分析		确定待开发软件的功能、性能、界面等要求，从而确定系统的逻辑模型	用户、项目负责人、系统分析员	需求规格说明书
软件开发阶段——待开发软件“怎么做”				
软件设计	概要设计	模块分解，确定软件的结构，模块的功能和模块间的接口，以及全局数据结构的设计	系统分析员、高级程序员	设计说明书、数据说明书、模块开发卷宗
	详细设计	设计每个模块的实现细节和局部数据结构的设计	高级程序员、程序员	
编码		用某种程序语言为每个模块编写程序	高级程序员、程序员	程序清单
软件测试		发现软件中的错误，并加以纠正	高级程序员或系统分析员（另一部门或单位）	软件测试计划、软件测试用例说明，软件测试报告
软件维护阶段—开发后交付使用的软件的维护				
软件维护		使软件适应外界环境的变化、实现功能的扩充和质量的改善而修改软件	维护人员	维护计划、维护报告

三、软件开发模型

为了反映软件生存周期内各种工作应如何组织及软件生存周期各个阶段应如何衔接，需要用软件开发模型给出直观的图示表达。软件开发模型是软件工程思想的具体化，是实施于过程模型中的软件开发方法和工具，是在软件开发实践中总结出来的软件开发方法和步骤。总的说来，软件开发模型是跨越整个软件生存周期的系统开发、运行、维护所实施的全部工作和任务的结构框架。常见的软件开发模型有瀑布模型、演化模型、螺旋模型和喷泉模型等。

（一）瀑布模型

瀑布模型：瀑布模型规定了各项软件工程活动，包括：制定开发计划，进行需求分析和说明，软件设计，程序编码。测试及运行维护。并且规定了它们自上而下，相互衔接的固定次序，如同瀑布流水，逐级下落。



（二）螺旋模型

对于复杂的大型软件，开发一个原型往往达不到要求。螺旋模型将瀑布模型与演化模型结合起来，并且加入两种模型均忽略了的风险分析。螺旋模型沿着螺线旋转，如图所示，在笛卡尔坐标的四个象限上分别表达了四个方面的活动，即：

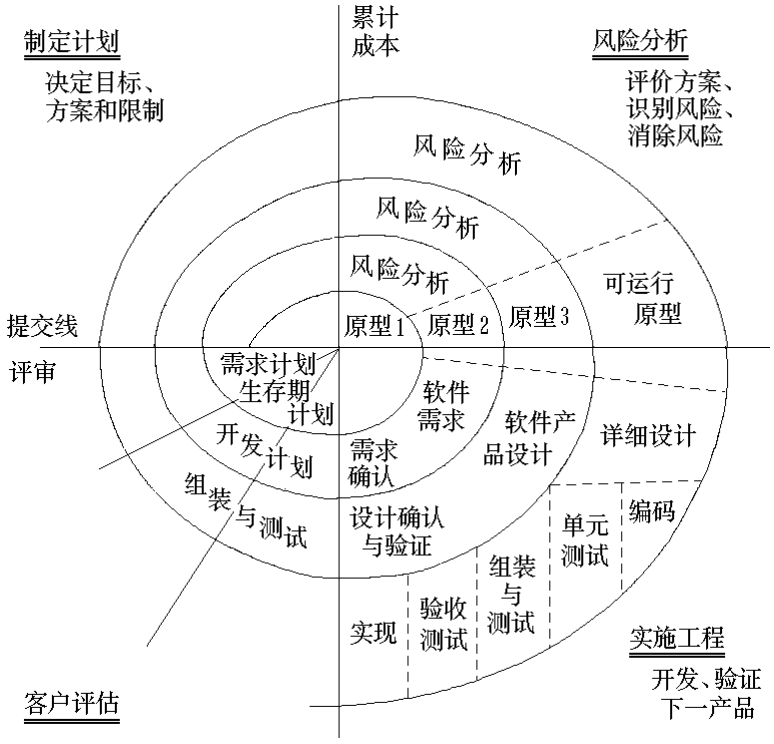
1.制订计划——确定软件目标，选定实施方案，弄清项目开发的限制条件；

2.风险分析——分析所选方案，考虑如何识别和消除风险；

3.实施工程——实施软件开发；

4.客户评估——评价开发工作，提出修正建议。

沿螺线自内向外每旋转一圈便开发出更为完善的一个新的软件版本。



（三）原型模型

又称快速成型模型，快速原型模型正是为了克服瀑布模型的缺点而提出来的。一般用于最终系统的早期用户评价，开发工期短，质量有保证。其本质是“快速”，开发人员应该尽可能快地建造出原型系统，以加速软件开发过程，节约软件开发成本。原型的用途是获知用户的真正需求，一旦需求确定了，原型将被抛弃。

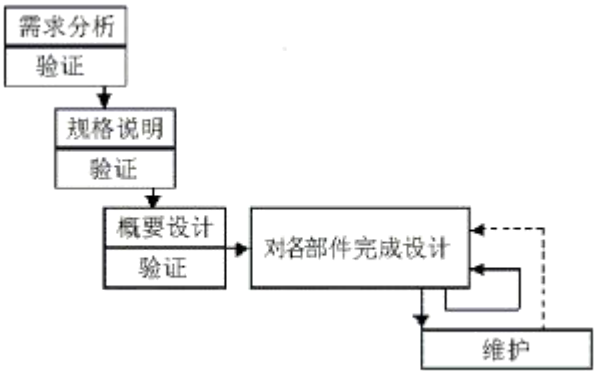
主要优点：

1.使用这种软件过程开发出的软件产品通常能满足用户的真实的需求；

2.软件产品的开发过程基本上是线性顺序过程。

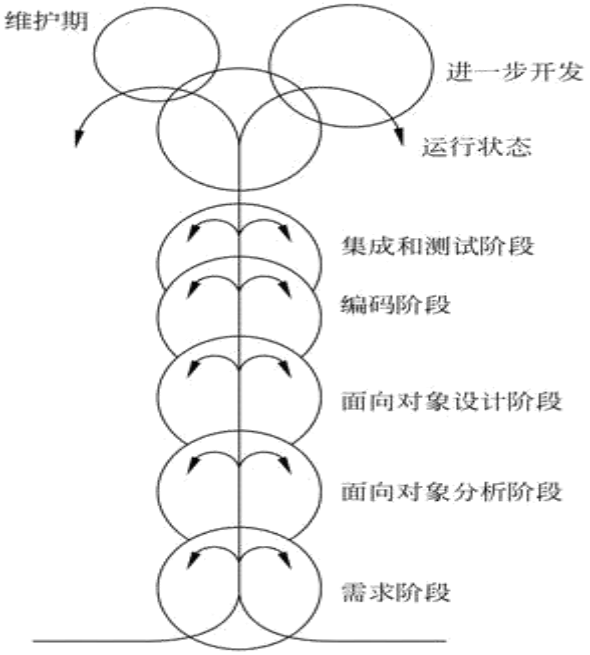
（四）增量模型

增量模型也称为渐增模型，使用增量模型开发软件是地，把软件产品作为一系列的增量构件来设计、编码、集成和测试。增量模型能较短时间内提交可完成部分工作的产品；可以使用户有充裕的时间学习和适应新产品。



（五）喷泉模型

典型的面向对象软件开发过程模型之一。喷泉模型能保证分析工作中得到的信息不会丢失或改变整个开发过程都是吻合一致的，或者说是“无缝”连接的。



四、CMMI 能力成熟度模型

CMMI 全称是 Capability Maturity Model Integration，即软件能力成熟度模型集成（也有称为软件能力成熟度集成模型），其目的是帮助软件企业对软件工程过程进行管理和改进，增强开发与改进能力，从而能按时地、不超预算地开发出高质量的软件。CMMI 主要有五个层次。

层次	特点
初始级	软件过程是无序的，有时甚至是混乱的，对过程几乎没有定义，成功取决于个人努力。管理是反应式的。
可重复级	建立了基本的项目管理过程来跟踪费用、进度和功能特性。制定了必要的过程纪律，能重复早先类似应用项目取得的成功经验。
已定义级	已将软件管理和工程两方面的过程文档化、标准化，并综合成该组织的标准软件过程。所有项目均使用经批准、剪裁的标准软件过程来开发和维护软件，软件产品的生产在整个软件过程是可见的。
量化管理级	分析对软件过程和产品质量的详细度量数据，对软件过程和产品都有定量的理解与控制。管理有一个作出结论的客观依据，管理能够在定量的范围内预测性能。
优化管理级	过程的量化反馈和先进的新思想、新技术促使过程持续不断改进。

第二节 软件计划阶段

一、可行性研究

（一）目的

可行性研究的目的是用最小的代价在尽可能短的时间内确定问题是否能够解决。也就是说可行性研究的目的不是解决问题，而是确定问题是否值得去解，研究在当前的具体条件下，开发新系统是否具备必要的资源和其他条件。

一般说来，应从经济可行性、技术可行性、运行可行性、法律可行性和开发方案的选择等方面研究可行性。

（二）工具

系统流程图是描绘物理系统的传统工具。它的基本思想是用图形符号以黑盒子形式描绘系统里面的每个部件（程序、数据库、图表、人工处理等）。系统流程图不同于程序流程图。

二、需求分析

（一）需求分析任务

需求分析是软件定义时期的最后一个阶段，它的基本任务是准确地回答“系统必须做什么？”这个问题。需求分析所要做的工作是深入描述软件的功能和性能，确定软件设计的限制和软件同其他系统元素的接口细节，定义软件的其他有效性需求。

通常软件开发项目是要实现目标系统的物理模型，即确定待开发软件系统的系统元素，并将功能和数据结构分配到这些系统元素中。它是软件实现的基础。

需求分析的任务不是确定系统如何完成它的工作，而是确定系统必须完成哪些工作，也就是对目标系统提出完整、准确、清晰、具体的要求。

（二）需求分析的工具

主要是 Word、PowerPoint、Visio、ProntPage、Excel 等工具。

第三节 软件开发阶段

一、概要设计

（一）基本概念

概要设计是一个设计师根据用户交互过程和用户需求来形成交互框架和视觉框架的过程，其结果往往以反映交互控件布置、界面元素分组以及界面整体板式的页面框架图的形式来呈现。这是一个在用户研究和设计之间架起桥梁，使用户研究和设计无缝结合，将对用户目标与需求转换成具体界面设计解决方案的重要阶段。

（二）主要任务

概要设计的主要任务是把需求分析得到的系统扩展用例图转换为软件结构和数据结构。设计软件结构的具体任务是：将一个复杂系统按功能进行模块划分、建立模块的层次结构及调用关系、确定模块间的接口及人机界面等。数据结构设计包括数据特征的描述、确定数据的结构特性，以及数据库的设计。显然，概要设计建立的是目标系统的逻辑模型，与计算机无关。

（三）设计原则

软件设计的原则对提高软件的设计质量有很大的帮助，主要有以下几点：

设计原则	功能
抽象	抽象是指忽视一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面。过程抽象和数据抽象是常用的两种主要抽象手段。
模块化	模块化是指将一个待开发的软件分解成若干个小的简单的部分——模块，每个模块可独立地开发、测试、最后组装成完整的软件。这是一种复杂问题的“分而治之”的原则。
信息隐蔽	开发整体程序结构时使用的法则，即将每个程序的成分隐蔽或封装在一个单一的设计模块中，定义每一个模块时尽可能少地显露其内部的处理。信息隐蔽原则对提高软件的可修改性、可测试性和可移植性都有重要的作用。
模块独立	模块独立是指每个模块完成一个相对独立的子功能，并且与其他模块之间的联系简单。衡量模块独立程度的度量标准有两个：耦合（耦合是指模块之间联系的紧密程度）和内聚（模块内部各元素之间联系的紧密程度）。

（四）概要设计的方法

方法	特点
功能层次模型	一般来讲就是系统的功能图，模块分布图等描述整个系统的功能的分布和功能的层次结构
数据流模型	以数据流为着眼点的分析方法得到的模型，主要通过数据在整个系统的流动情况来确定系统的主要功能主线和流程
控制流模型	通过了解和界定系统中控制线，通过控制流的走向和控制的对象来确定系统的功能分布和控制与被控制的关系

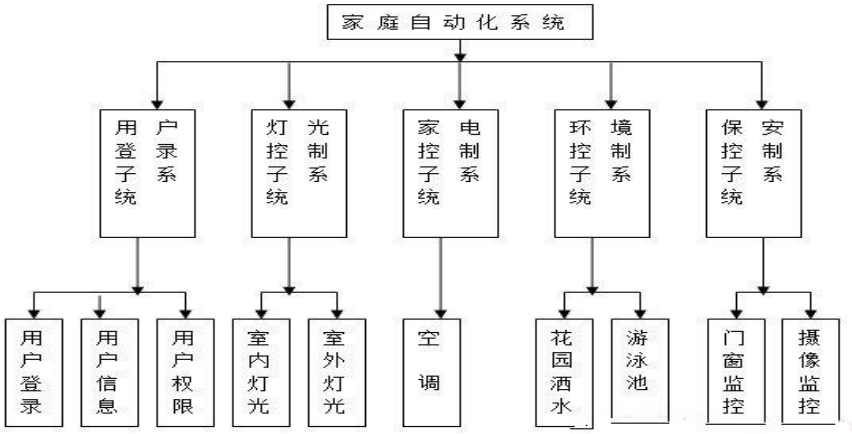
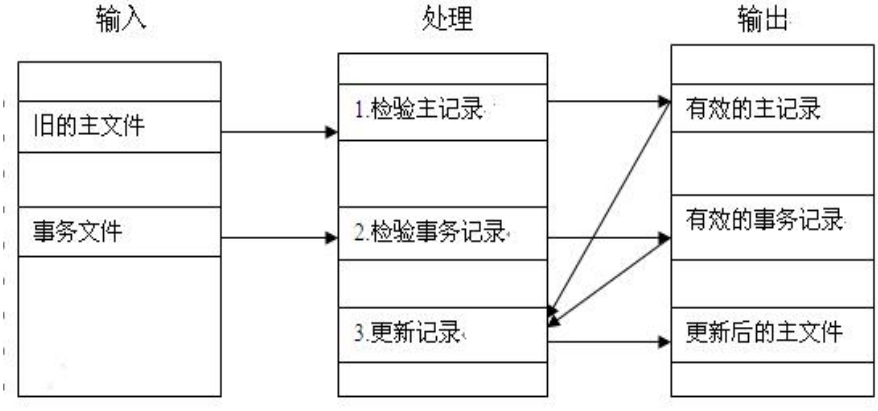
结构化分析（SA）方法是一种面向数据流的需求分析方法，它适用于分析大型数据处理系统。结构化分析方法的基本思想是自顶向下逐层分解，这样做可以把一个大问题分解成若干个小问题，经过多次

逐层分解，每个最底层的问题都是足够简单、容易解决的，这个过程就是分解的过程。

结构化方法的分析结果由数据流图 DFD、数据词典和加工逻辑说明几个部分组成。其中，DFD 的基本成分有数据流（data flow）、加工（process）、文件（file）和源/宿（source/sink）。

（五）设计工具

使用的图形工具主要有层次方框图、Warnier 图、IPO 图。

工具	图示
层次方框图	
Warnier 图	
IPO 图	

（六）产生的文档

1.概要设计说明书。2.数据库设计说明书。3.用户手册。4.修订测试计划。

（七）概要设计文档书写格式

概要设计的格式如下：

1 引言

1.1 编写目的

1.2 背景

1.3 定义

1.4 参考资料

2 总体设计

2.1 需求规定

2.2 运行环境

2.3 基本设计概念和处理流程

2.4 结构

2.5 功能需求与程序的关系

2.6 人工处理过程

2.7 尚未解决的问题

3 接口设计

3.1 用户接口

3.2 外部接口

3.3 内部接口

4 运行设计

4.1 运行模块组合

4.2 运行控制

4.3 运行时间

5 系统数据结构设计

5.1 逻辑结构设计要点

5.2 物理结构设计要点

5.3 数据结构与程序的关系

6 系统出错处理设计

6.1 出错信息

6.2 补救措施

6.3 系统维护设计

（八）概要设计实例

以网上书店为例，重点介绍概要设计阶段中的需求规定部分。网上书店分为 5 个模块。

1.登录模块

功能编号	1	功能名称	登录
功能描述	1.前台会员登录：会员输入正确的用户名、密码后成功登录前台系统并记住会员的用户名和用户的类型。 2.后台管理员登录:管理员输入的用户名、密码和验证码，成功的登录后台，并旦根据不同的角色分配相应的权限。		
输入项	用户名、验证码、密码		

处理描述	1.会员登录：若数据库用户信息表中存在此用户，用户就可以进入系绞前台进行会员的操作。 2.后台管理员的登录：若教据库的管理信息表中存在此用户、密码相符并且验证码正确.此时系统就转入后台管理界面.并且系统根据角色给管理员分配相应的权限。
输出项	1.前台相应的页面 2.后台相应的页面 3.出错信息
界面要求	图形化用户界面

2.注册模块

功能编号	2	**功能名称**	**注册**
功能描述	如果用户未注册,用户点击注册按钮可以进入注册页面，在注册页面中用户按要求填写注册信息，注册成功后用户就拥有会员的所有权限。		
输入项	系统要求的用户注册信息		
处理描述	系统首先判断用户信息表中该用户名是否存在，若不存在则判断用户的注册信息是否合法，如果合法则注册成功。否则，注册失败。		
输出项	1.注册成功页面 2.注册失败页面 3.出错信息		
界面要求	图形化用户界面		

3.普通用户模块

功能编号	3	**功能模块**	**非会员界面**
功能描述	1.图书显示：系统按图书类别、最新上架图书和最畅销图书来显示图书信息 2.查询图书：用户可以根据书名、作者、出版社等条件进行图书查询。 3.注册（参见注册祺块> 4. 帮 助 中 心 ：为该系统的用户提供帮助.在帮助中心我们可以了解到 会员的积分规则和会员购书琉程等。		
输入项	图书查询条件		

处理描述	1.用户进入系统以后，系统自动从数据库中读取信息，并且在图书显示页面上按图书类别、最新上架图书和最畅销图书来显示图书信息。 2.系统可以根据用户输入的查询条件在数据库中查询，并且把查询结果显示出来。 3.注册
输出项	1.图书信息 2.查询的结果 3.错误信息
界面要求	图形化界面

4.会员模块

功能编号	**4**	**功能模块**	**会员界面**
功能描述	1.在线购买图书 2.发表评论：用户只能对自己已购买的图书发表评论。 3.会员自助服务管理：会员注册信息维护、找回密码、我的暑假和查看购书记录 4.其他普通用户所具有的功能		
输入项	会员登录以后才能进入此模块		
处理描述	1.系统把用户要购买的图书放入购物车中，当用户点击购买，并且填 写好配送信息时，系统自动生成购物订单。 2.当用户对他已购买的图书发表评论时，系统会把用户的评论存入数据库中，以 便于其他用户可以浏览此评论。 3.会员自助服务管理： （1）用户可以对自己的用户信息进行维护，系统按用户的需要对用户的注册信息进行维护。 （2）如果会员忘记了自己的密码，可通过注册邮箱取回密码。系统从数据库中检测会员出书的邮箱和密码，并把密码发到会员的注册邮箱中，否则产生错误信息。 （3）会员可以对自己的购书记录进行查看。 4.普通用户的功能。		
输出项	1.订单的生成 2.用户找回的密码 3.用户的评论 4.错误信息		
界面要求	图形化界面		

5.管理员界面

功能编号	**5**	**功能模块**	**后台管理界面**

功能描述	1.会员管理：完成会员的编辑和删除等操作 2.评论管理：充成用户评论的谄辑和删除 3.角色管理：充成角色的添加、删除、编辑等操作 4.图书管理：完成图书信息的添加，图书信息的编辑：图书信息的删除和图书分类管理等操作 5.订单管理：充成订单的维护和订单的查询 6.数据备份还原：完成数据的备份和还原 7.销售绕计查询（由张爱玲完成）：总体销售统计、销售额统计、图 书销售额排名、图书访问购买率、图书类别销售排名、会员订单量排名、会员购物额排名。
输入项	通过登录模块的管理员登录以后才能进入后台管理界面
处理描述	系统根据登录管理员的角色来给管理员分配权限.不同的角色可进行不同的操作。
输出项	1.查询、增加、删除、修改后的信息 2.销售统计信息 3.备份还原的数据 4.错误信息
界面要求	图形化用户界面

二、详细设计

（一）基本概念

详细设计是软件工程中软件开发的一个步骤就是对概要设计的一个细化就是详细设计每个模块实现算法所需的局部结构。

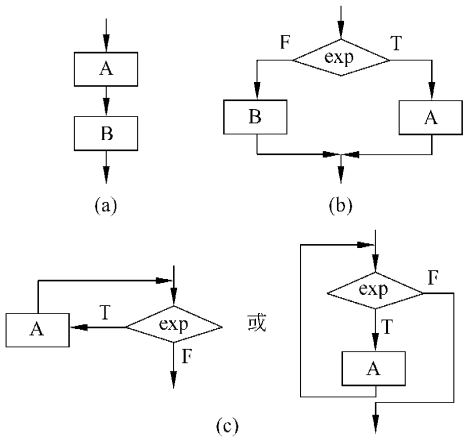
（二）主要任务

详细设计的主要任务是设计每个模块的实现算法所需的局部数据结构详细设计的目标有两个实现模块功能的算法要逻辑上正确和算法描述要简明易懂。

（三）设计方法

传统软件开发方法的详细设计主要是用结构化程序设计法。

结构程序设计的概念最早由 E.W.Dijkstra 提出。1965 年他在一次会议上指出：“可以从高级语言中取消 GO TO 语句”。1966 年 Bohm 和 Jacopini 证明了，只用 3 种基本的控制结构就能实现任何单入口单出口的程序。即：顺序、选择和循环。



（四）设计工具

常用的图形描述工具有程序流程图、盒图（NS 图）和问题分析图（PAD）。典型的语言描述工具是 PDL（program design language）。典型的表格描述工具是判定表和判定树。

（五）产生的文档

详细设计说明书。

三、软件编码

根据详细设计说明书编写程序，为开发项目选择程序设计语言需要考虑的因素有应用领域、算法和计算的复杂性、软件运行环境、用户需求、数据结构和开发人员的水平。软件的设计质量与程序设计语言的技术性能无关，但在程序设计转向程序代码时，转化的质量受语言性能的影响。

好的程序应该具有模块化结构，系统应该有较高的模块独立性。

从应用领域看，COBOL 适合商业领域；FORTRAN 适合科学计算；PROLOG 和 LISP 适合人工智能领域；SMALLTALK、C++、JAVA 是面向对象语言；C 是开发系统的程序设计语言。

此阶段是将软件设计的结果转化为计算机可运行的程序代码。在程序编码中必定要制定统一、符合标准的编写规范。以保证程序的可读性、易维护性，提高程序的运行效率。

四、软件测试

（一）测试的概念

1.软件测试

软件测试是对软件计划、软件设计、软件编码进行查错和纠错的活动（包括代码执行活动与人工活动）。

2.程序测试

程序测试是早已流行的概念。它是对编码阶段的语法错、语义错、运行错进行查找的编码执行活动。找出编码中错误的代码执行活动称程序测试。纠正编码中的错误的执行活动称程序调试。通过查找编码错与纠正编码错来保证算法的正确实现。

3.软件确认与程序确认

软件确认是广义上的软件测试，它是企图证明程序软件在给定的外部环境中的逻辑正确性的一系列活动和过程，指需求说明书的确认，程序的确认。程序确认又分成静态确认与动态确认。静态确认包括，正确性证明，人工分析，静态分析。动态分析包括动态确认与动态测试。

（二）测试的目的

对源程序最基本的质量要求是正确性和可靠性，此外还很注重软件的易使用性、易维护性和易移植性。软件测试的工作量约占软件开发总工作量的 40%以上，其目的是尽可能多的发现软件产品（主要是指程序）中的错误和缺陷。

测试的目的是确保软件的质量，尽量找出软件错误并加以纠正，而不是证明软件没有错。测试的范围是整个软件的生存周期，而不限于程序编码阶段。

（三）测试的原则

1. 测试前要认定被测试软件有错，不要认为软件设有错。

2.要预先确定被测试软件的测试结果。

3.要尽量避免测试自己编写的程序。

4.测试要兼顾合理输入与不合理输入数据。

5.测试要以软件需求规格说明书为标准。

6.要明确找到的新错与已找到的旧错成正比。

7.测试是相对的，不能穷尽所有的测试，要据人力物力安排测试，并选择好测试用例与测试方法。

8.测试用例留作测试报告与以后的反复测试用，重新验证纠错的程序是否有错。

（四）测试的方法

类型	功能	方法
白盒测试	根据程序的内部逻辑来设计测试用例	逻辑覆盖法（语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径覆盖）
黑盒测试	根据规格说明所规定的功能来设计测试用例，它不考虑程序的内部结构和处理过程	等价类划分、边值划分、错误猜测

白盒测试又称结构测试、透明盒测试、逻辑驱动测试或基于代码的测试。

常见的四种黑盒测试用例的方法。

（五）软件测试过程

测试过程	功能	方法
单元测试	主要用来发现编码和详细设计中产生的错误，一般在编码阶段	白盒测试
集成测试（组装测试）	主要用来发现设计阶段产生的错误，是对各模块组装而成的程序进行测试，主要检查模块间的接口和通信	黑盒测试
确认测试	检查软件的功能、性能和其他特征是否与用户需求一致，它以需求规格说明书作测试为依据	黑盒测试 Alpha 测试是在开发者的现场由客户来实施的，从用户角度和环境下进行； Beta 测试是在开发者不在现场下测试，由软件最终用户实施

系统测试	①恢复测试、②安全测试、③强度测试、④性能测试。	

软件测试阶段的任务是在软件设计完成之后要进行严密的测试，一发现软件在整个软件设计过程中存在的问题并加以纠正。

第四节 软件维护阶段

一、基本概念

软件维护阶段是指从软件交付使用到软件被淘汰为止的整个时期，它是在软件交付使用后，为了改正软件中隐藏的错误，或者为了使软件适应新的环境，或者为了扩充和完善软件的功能或性能而修改软件的过程。

软件维护阶段的主要任务是，软件将在运行使用中不断地被维护，根据新提出的需求进行必要而且可能的扩充和删改、更新和升级。

二、软件维护的分类

根据引起软件维护的原因，软件维护通常可分成改正性维护、适应性维护、完善性维护、预防性维护。

类型	定义
改正性维护	诊断和改正错误的过程
适应性维护	为了使软件适应这些环境变化而修改软件的过程
完善性维护	在软件投入使用过程中，为了满足用户可能还会有新的功能和性能要求，可能会提出增加新功能、修改现有功能等要求而进行的维护
预防性维护	为了改进软件未来的可维护性或可靠性，或者为了给未来的改进奠定更好的基础而进行的修改

三、软件质量

软件质量是指反映软件系统或软件产品满足规定或隐含需求的能力的特征和特性全体。下面从管理的角度列出了影响软件质量的主要因素。

阶段	质量因素	定义
产品运行	正确性	系统满足规格说明和用户目标的程序，即在预定环境下能正确的完成预期功能的程序
	健壮性	在硬件发生故障、输入的数据无效或操作错误等意外环境下，系统能做出适当响应的程序
	效率	为了完成预定的功能，系统需要的计算资源的多少
	完整性（安全性）	对未经授权的人使用软件或数据的企图，系统能够控制（禁止）的程序
	可用性	系统在完成预定应该完成的功能时令人满意的程度
	风险	按预定的成本和进度将系统开发处理，并且为用户满意的概率
产品修改	可理解性	理解和使用该系统的容易程度
	可维修性	诊断和改正在运行现场发现的错误所需要的工作量的多少
	灵活性（适应性）	修改或改进正在运行的系统需要的工作量的多少
	可测试性	软件容易测试的程度
产品转移	可移植性	把程序从一种硬件配置和（或）软件系统环境转移到另一种配置和环境时，需要的工作量多少
	可再用性	在其他应用中该程序可以被再次使用的程度（或范围）
	互运行性	把该系统和另一个系统结合起来需要的工作量的多少

第九章 汇编语言

第一节 8086 的寄存器、存储器和系统功能调用

8086 是美国 Intel 公司生产的微处理器，其主要部件包括：运算器、控制器和寄存器组。

一、寄存器 8086 寄存器组

寄存器组 8086 寄存器组包括通用寄存器和专用寄存器。

（一）通用寄存器

1.数据寄存器：AX、BX、CX、DX，四个 16 位通用寄存器，用来暂时存放计算过程中所用到的操作数、结果和其他信息。既能以字形式（如 AX）也能以字节形式（如 AH、AL）访问。

（1）AX（accumulator）：累加器，算术运算的主要寄存器。所有的 I/O 指令都使用这一寄存器与外部设备传送信息。

（2）BX（base）：通用寄存器，在计算存储器地址时常用作基址寄存器。

（3）CX（count）：通用寄存器，在循环和串操作指令中用作隐含的计数器。

（4）DX（data）：通用寄存器，在作双字长运算时把 DX 和 AX 合在一起存放一个双字长数，DX 用来存放高位字。对某些 I/O 操作，DX 用于对 I/O 端口的寄存器间接寻址。

2.指针及变址寄存器：SP、BP、SI、DI，四个 16 位寄存器。

（1）SP：堆栈指针寄存器。

（2）BP：基址指针寄存器。

（3）SI：源变址寄存器。

（4）DI：目的变址寄存器。

（二）专用寄存器：IP、SP、FLAGS，3 个 16 位寄存器

1.IP：指令指针寄存器。存放代码段中的偏移地址。

2.SP：堆栈指针寄存器，指示栈顶的偏移地址。

3.FLAGS：标志寄存器，又称为程序状态字寄存器（programstatusword，PSW）。由条件码标志（flag）、控制标志和系统标志构成。8086 的 FLAGS 如下所示：

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

（1）条件码：用来记录程序中运行结果的状态信息，它们是根据有关指令的运行结果由 CPU 自动设置的。由于这些状态信息往往作为后续条件转移指令的转移控制条件，所以称为条件码。

①OF：溢出标志（overflowflag）。结果溢出 OF=1，否则 OF=0。

②SF：符号标志（signflag）。结果为负 SF=1，否则 SF=0。

③ZF：零标志（zeroflag）。结果为 0 时 ZF=1，否则 ZF=0。

④CF：进位标志（carryflag）。最高有效位有进位或借位 CF=1，否则 CF=0。

⑤AF：辅助进位标志或半进位标志（auxiliarycarryflag）。半个字节产生的进位或借位时 AF=1，否则 AF=0。

⑥PF：奇偶标志（parityflag）。结果低 8 位中所含的 1 的个数为偶数时 PF=1，否则 PF=0。

（2）控制标志位：为方向标志（directionflag，DF），在串处理指令中控制处理信息的方向用。

①DF=1：变址寄存器 SI 和 DI 减量，使串处理从高地址向低地址方向处理。

②DF=0：变址寄存器 SI 和 DI 增量，使串处理从低地址向高地址方向处理。

（3）系统标志位：可以用于 I/O、可屏蔽中断、程序调试、任务切换和系统工作方式等的控制。

①TF：陷阱标志（trapflag，TF），又称单步标志。用于调试时的单步方式操作。TF=1 时每条指令执行完后产生陷阱（中断），TF=0 时不产生陷阱。

②IF：中断标志（interruptflag，IF）。IF=1 时允许可屏蔽中断请求，IF=0 时禁止可屏蔽中断。

（三）段寄存器

段寄存器是一种专用寄存器，它们专用于存储器寻址，用来直接或间接地存放段地址。在 8086 的处理器中只有 CS、DS、SS、ES 四个 16 位寄存器。

1.代码段 CS：存放当前正在运行的程序。

2.数据段 DS：存放当前运行程序所用的数据。

3.堆栈段 SS：定义堆栈的所在区域。

4.附加段 ES：附加的数据段。

除非专门指定，一般情况下，各段在存储器中的分配是由操作系统负责的。而且允许段重叠。如果段超过 64KB 则将其作为 2 个以上的段，并动态修改段地址。

二、存储器

（一）存储单元的地址和内容：

1.字节：每 8 位二进制数组成一个字节（Byte）。在存储器里是以字节为单位存储信息。

2.字：16 位二进制数（2 个字节）组成一个字。分成低位字节和高位字节存储。

3.存储单元的内容：该存储单元存放的信息。存储器的内容取之不尽。

（1）字节的存储：直接存入某存储单元。

（2）字的存储：一个字存入存储器要占用两个单元。存放时，低位字节存入低地址单元，高位字节存入高位地址单元。字单元的地址用它的低地址表示，应尽量为偶数地址。

（3）如果用 X 表示某存储单元的地址，则 X 单元的内容可以表示为（X）；假如 X 单元中存放着 Y，而 Y 又是一个地址，则可用（Y）=（（X））来表示 Y 单元的内容。

（二）实模式存储器寻址

8086 只能在实模式下工作。实模式就是为 8086 而设计的工作方式，实模式下允许的最大寻址空间为 1MB。它要解决在 16 位字长的机器里怎么提供 20 位地址的问题，而解决的办法是采用存储器地址分段的方法。存储器地址的分段

（1）段：程序员在编制程序时要把存储器划分成段，在每个段内地址空间是线性增长的。段的最大长度为 64KB，以便能用 16 位二进制数表示段内地址。

①小段：从 0 地址开始，每 16 字节为一小段。小段的首地址用 16 进制数表示时其最低位为 0。

②段的起始地址：段不能起始于任意地址，而必须从任一小段的首地址开始。

（2）物理地址、段地址、偏移地址

①物理地址（PA）：在 1MB 的存储器里，每一个存储单元都有一个唯一的 20 位地址，称为该存储单元的物理地址（20 位）。20 位物理地址由 16 位段地址和 16 位偏移地址组成。

②段地址：段起始地址的高 16 位值（16 位），低 4 位固定为 0 被省去。

③偏移地址（偏移量 EA）：指在段内相对于段起始地址的偏移值（16 位）。

④物理地址=段地址×10H+偏移地址。用段地址：偏移地址表示。

三、外部设备的功能调用

（一）DOS 功能调用

DOS 功能调用是 DOS 的一个组成部分，在开机时由操作系统从系统磁盘装入存储器。为了使用方便，将 DOS 层功能模块所提供的 88 个子程序统一顺序编号从 00H 到 57H。DOS 系统功能调用方式如：

1.置入口参数；

2.将子程序编号送入 AH 寄存器；

3.执行中断指令：INT21H。有的子程序不需入口参数，但大部分需要将参数送入指定地点。程序员只须给出这三个方面的信息，不必关心具体程序如何，在内存中的存放地址如何，DOS 根据所给的信息，自动转入相应的子程序去执行。调用结束后有出口参数时一般在寄存器中，有些子程序调用结束时会在屏幕上看到结果。常用的有 01、02、09 和 4C。

（1）键盘输入并回显（01 号功能调用）

格式：MOV AH，01H INT 21H 功能：键盘输入一个字符把其 ASCII 放入 AL 并在屏幕上显示输出。

（2）屏幕显示一个字符（02 号功能调用）

格式：MOV DL，'字符'MOV AH，02H INT 21H 功能：将置入 DL 寄存器中的字符在屏幕上显示输出。

（3）屏幕显示字符串（09 号功能调用）

格式：MOV DX，字符串的偏移地址 MOV AH，09H INT 21H 功能：在屏幕上显示字符串。在使用 9 号功能调用时，应当注意以下问题。

①待显示的字符串必须先放在内存一数据区（DS 段）中，且以‘$’符号作为结束标志。

②应当将字符串首地址的段基址和偏移地址分别存入 DS 和 DX 寄存器中。

（4）返回 DOS 操作系统（4CH 号功能调用）

格式：MOV AH，4CH INT 21H 功能：终止当前程序的运行，并把控制权交给调用的程序，即返回 DOS 系统，屏幕出现 DOS 提示符，如”C:/>”，等待 DOS 命令。

（二）BIOS 功能调用

BIOS 存放在机器的 ROM 中，比 DOS 的层次还要低。BIOS 是固化在只读存储器 ROM 中的一系列输入/输出服务程序，它存放于内存的高地址区域内，除负责处理系统中的全部内部中断外，还提供对主要 I/O 接口的控制功能，如键盘、显示器、磁盘、打印、日期与时间等。BIOS 采用模块化结构，每个功能模块的入口地址都存于中断向量表中。对这些中断调用是通过软中断指令 INT n 来实现的，中断指令中的操作数 n 即为中断类型码。BIOS 的调用方法与 DOS 系统功能调用方法类似：

①置功能号于 AH 中。

②置入口参数。

③执行 INT n。

④分析出口参数及状态。

例如：键盘 I/O 中断调用（INT16H）键盘 I/O 中断调用有三个功能，功能号为 0，1，2，且必须把功能号放在 AH 中。

（1）0 号功能调用

格式：MOV AH，0 INT 16H

功能：从键盘读入字符送 AL 寄存器。执行时，等待键盘输入，一旦输入，字符的 ASCII 码放入 AL 中。若 AL＝0，则 AH 为输入的扩展码。

（2）01 号功能调用

格式：MOV AH，01H INT 16H

功能：用来查询键盘缓冲区，对键盘扫描但不等待，并设置 ZF 标志。若有按键操作，则 ZF＝0，AL 中存放的是输入的 ASCII 码，AH 中存放输入字符的扩展码。若无键按下，则标志位 ZF＝1。

（3）02 号功能调用

格式：MOV AH，02H INT 16H

功能：检查键盘上各特殊功能键的状态。执行后，各种特殊功能键的状态放入 AL 寄存器中。

这个状态字记录在内存 0040H：0017H 单元中，若对应位为“1”，表示该键状态为“ON”，处于按下状态；若对应位为“0”，表示该键状态为“OFF”，处于断开状态。

第二节 8086 的寻址方式和指令系统

说明：

1.指令系统：计算机能够提供给用户的一组指令集即为该计算机的指令系统。

2.指令的组成：计算机中的指令由操作码字段和操作数字段组成。

（1）操作码字段：指示计算机所要执行的操作。

（2）操作数字段：指出在指令执行操作的过程中所需要的操作数。

3.指令的格式：操作码[操作数[，操作数][，操作数]]。8086 CPU 规定操作数不能超过两个。源操作数和目的操作数：使用两地址指令的两个操作数分别称为源操作数和目的操作数。

4.操作数的表示方法使用的是寻址方式。寻址方式是规定寻找操作数的方法。

5.汇编语言：符号语言。用助记符来表示操作码，用符号或符号地址来表示操作数或操作数地址。它与机器指令一一对应。

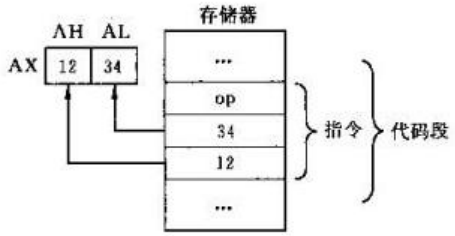
一、8086 的寻址方式

（一）与数据有关的寻址方式

这种寻址方式用来确定操作数地址从而找到操作数。

1.立即数寻址方式：操作数直接存放在指令中，紧跟在操作码之后，这种操作数称为立即数。它作为指令的一部分存放在代码段里。

例如：MOV AX，1234H



2.寄存器寻址方式：操作数在寄存器中，指令指定寄存器号。如：MOV BX，AX

3.直接寻址方式：操作数的有效地址只包含位移量一种成份，其值就存放在代码段中的指令的操作码之后。位移量的值即操作数的有效地址 EA。

（1）由此及往下的各种寻址方式的操作数都在除代码段以外的存储区中。用方括号[]括起来的为存储器操作数。寄存器名称外加小括号”（）”表示是该寄存器的内容。

（2）有效地址（EA）：即操作数的偏移地址。自此开始的寻址方式即为求得有效地址（EA）的不同途径。有效地址的计算可以用下式表示：

EA＝基址＋变址＋位移量

有效地址可以由以下三种成分组成：

①位移量是存放在指令中的一个 8 位、16 位的数，但它不是立即数，而是一个地址。

②基址是存放在基址寄存器中的内容。它是有效地址中的基址部分，通常用来指向数据段中数组或字符串的首地址。

③变址是存放在变址寄存器中的内容。它通常用来访问数组中的某个元素或字符串中的某个字符。

（3）段跨越前缀：8086 允许数据存放在数据段以外的段中，应在指令中用该段寄存器加冒号（":"）即段跨越前缀来指定该段。只要有 BP 则隐含的段寄存器为 SS。否则隐含的段寄存器为 DS。如：MOV AX，ES:VALUE。但是在以下三种情况下，不允许使用段跨越前缀，它们是：

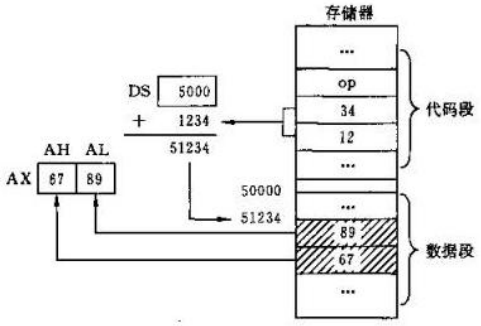
①串处理指令的目的串必须使用 ES 段；

②PUSH 指令的目的和 POP 指令的源必须用 SS 段；

③指令代码必须存放在 CS 段中。

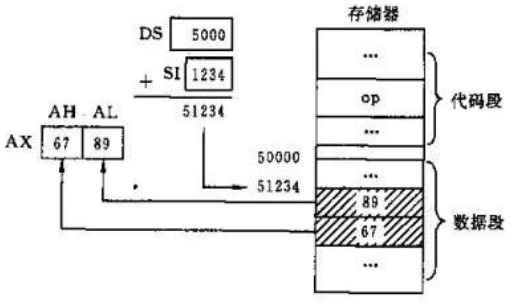
（4）8086 CPU 中为了使指令字不要过长，规定双操作数指令的两个操作数中，只能有一个使用存储器寻址方式，这就是一个变量常常先要送到寄存器的原因。

例如：假设数据段寄存器 DS 的内容为 50000H，地址为 51234H 字存储单元中的内容为 6789H，那么执行指令 MOV AX，[1234H]后寄存器 AX 的内容为 6789H



4.寄存器间接寻址方式：操作数的有效地址只包含基址寄存器或变址寄存器内容一种成份。有效地址就在某个寄存器中，而操作数则在存储器中。可使用段跨越前缀。如：MOV AX，ES：[BX]

例如：MOV AX，[SI]，假设：（DS）=50000H，（SI）=1234H

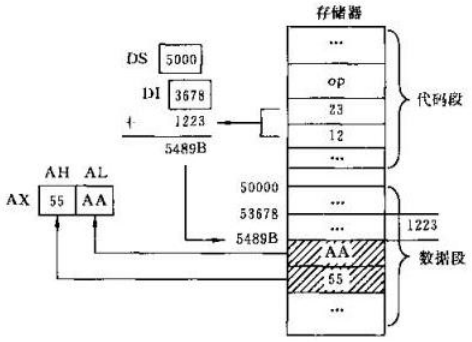


5.寄存器相对寻址方式（或称直接变址寻址方式）：操作数的有效地址为基址寄存器或变址寄存器的内容和指令中指定的位移量之和，有效地址由两部分组成。可使用段跨越前缀（又称为段超越）。如：MOV ES：STRING[SI]，DL

$$EA=\begin{Bmatrix}(BX)\\(BP)\\(DI)\\(SI)\end{Bmatrix}+\begin{Bmatrix}8位位移量\\16位位移量\end{Bmatrix}$$

例子：MOV AX，[DI+1223H]等价于 MOV AX，1223H[DI]

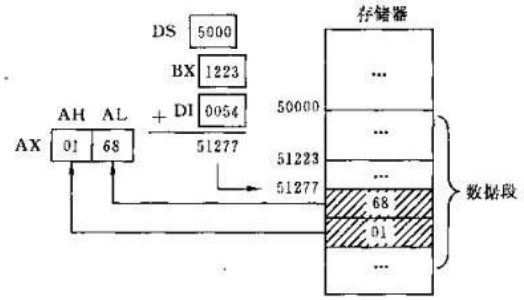
假设：（DS）=50000H，（DI）=3678H



6.基址变址寻址方式：操作数的有效地址 EA 是一个基址寄存器和一个变址寄存器的内容之和。可使用段跨越前缀。如：MOV AX，ES：[BX][SI]

$$EA=\begin{Bmatrix}(BX)\\(BP)\end{Bmatrix}+\begin{Bmatrix}(SI)\\(DI)\end{Bmatrix}$$

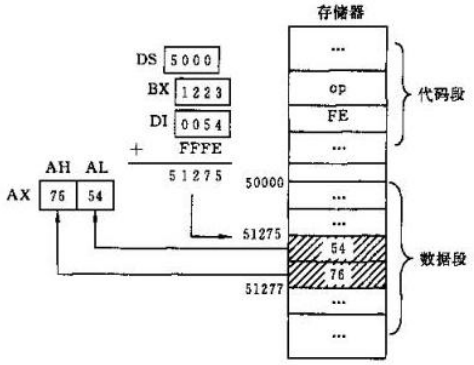
例子：MOV AX，[BX+DI]等价于 MOV AX，[BX][DI]



7.相对基址变址寻址方式：操作数的有效地址 EA 是一个基址寄存器与一个变址寄存器的内容和指令中指定的位移量之和。可使用段跨越前缀。如：MOV ES：MASK[BX][SI]，AX

$$EA=\begin{Bmatrix}(BX)\\(BP)\end{Bmatrix}+\begin{Bmatrix}(SI)\\(DI)\end{Bmatrix}+\begin{Bmatrix}8位位移量\\16位位移量\end{Bmatrix}$$

例子：MOV AX，[BX+DI-2]



（二）与转移地址有关的寻址方式

这种寻址方式用来确定转移指令及 CALL 指令的转向地址。

1.段内直接寻址：转向的有效地址是当前 IP 内容和指令中指定的 8 位或 16 位位移量之和。（操作数 OPR 采用相对寻址方式。）当它用于条件转移指令时，位移量只能是 8 位。

$$(IP) \leftarrow EA = (IP) + \begin{cases} 8\text{位位移量，如：JMP SHORT OPR} & \text{；段内直接短程转移} \\ 16\text{位位移量，如：JMP NEAR PTR OPR} & \text{；段内直接近程转移} \end{cases}$$

2.段内间接寻址：转向的有效地址是一个寄存器或是一个存储单元的内容。（操作数 OPR 采用除立即数以外的任一数据寻址方式。）

$$(IP) \leftarrow EA = \begin{cases} \text{寄存器内容，如：JMP BX 或 JMP ECX} & \text{；OPR为BX或ECX} \\ \text{存储器内容，如：JMP WORD PTR [存储器寻址方式]} & \text{；OPR为存储器} \end{cases}$$

3.段间直接寻址：指令中直接提供了转向的段地址和偏移地址。（操作数 OPR 采用立即数寻址方式。）

（IP）←EA=OFFSETOPR；OPR 的偏移地址→（IP）

（CS）←SEGOPR；OPR 的段地址→（CS）

4.段间间接寻址：用存储器中两个相连字来取代 IP 和 CS 的内容。（操作数 OPR 采用存储器寻址方式。）

（IP）←EA=存储器中双字单元的低字内容

（CS）←EA+2=存储器中双字单元的高位字内容

（三）经常用到的操作符

①SHORT：属性操作符，表示段内短程转移。

②PTR：属性操作符，建立一个符号地址（取后面内容的地址）。

③NEAR：类型操作符，距离类型，段内近程。

④FAR：类型操作符，距离类型，段间远程。

⑤WORD：类型操作符，数据类型，字。

⑥DWORD：类型操作符，数据类型，双字。

⑦NEARPTR：取段内近程地址值操作符。

⑧FARPTR：取段间远程地址值操作符。

⑨WORDPTR：取字长地址值操作符。

⑩DWORDPTR：取双字长地址值操作符。

二、8086 的指令系统

（一）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。

1.通用数据传送指令

（1）MOV——传送指令

指令格式：MOV DST，SRC　　；（DST）←（SRC）。DST 表示目的操作数，SRC 表示源操作数

说明：

①DST 为除 CS 外的各寄存器寻址方式或任意存储器寻址方式。SRC 为任意数据寻址方式。

②DST、SRC 不能同时为存储器寻址方式，也不能同时为段寄存器寻址方式，而且在 DST 为段寄存器时，SRC 不能为立即数。

③MOV 指令不影响标志位。

（2）PUSH——进栈指令

指令格式：PUSH SRC；16 位指令：（SP）←（SP）–2（（SP）+1，（SP））←（SRC）

说明：

①堆栈：计算机开辟的以”后进先出”方式工作的存储区。它必须存在于堆栈段中，只有一个出入口，所以只有一个堆栈指针 SP。SP 的内容在任何时候都指向当前的栈顶。

②8086 中的 SRC 不能为立即数寻址方式。

③PUSH 指令不影响标志位。

（3）POP——出栈指令

指令格式：POP DST；16 位指令：（DST）←（（SP）+1，（SP））（SP）←（SP）+2

说明：

①DST 为除立即数及 CS 寄存器以外的任意数据寻址方式。

②POP 指令不影响标志位。

（4）PUSH A——寄存器进栈指令

指令格式：PUSH A；16 位通用寄存器依次进栈，进栈次序为：AX、CX、DX、

BX、指令执行前的 SP、BP、SI、DI。指令执行后（SP）←（SP）–16 仍指向栈顶。

（5）POP A——寄存器出栈指令

指令格式：POPA；16 位通用寄存器依次出栈，出栈次序为：DI、SI、BP、

SP、BX、DX、CX、AX。指令执行后（SP）←（SP）+16 仍指向栈顶。注意 SP 内容并未恢复。

（6）XCHG——交换指令

指令格式：XCHGOPR1，OPR2；（OPR1）←→（OPR2）。其中 OPR 表示操作数

说明：

①OPR1、OPR2 为除段寄存器以外的各寄存器寻址方式或任意存储器寻址方式。

②OPR1、OPR2 不能同时为存储器寻址方式。

③XCHG 指令不影响标志位。

2.累加器专用传送指令

（1）IN——输入指令

长格式为：IN AL，PORT（字节）　；（AL）←（PORT）（字节）

IN AX，PORT（字）　　；（AX）←（PORT+1，PORT）（字）

短格式为：IN AL，DX（字节）　　；（AL）←（(DX)）（字节）

IN AX，DX（字）　　；（AX）←（(DX）+1，(DX)）（字）

说明：①.8086CPU 规定只能用低 16 位地址总线（A15～A0）来寻址外部设备，因此外部设备最多可有 65536 个 I/O 端口，端口地址为（0～FFFFH）。

②长格式只适用于端口（PORT）号≤255（FFH）。

③短格式适用于任意端口号（0～FFFFH）。但只能用 DX 寄存器对端口地址进行间接寻址。

④IN 指令不影响标志位。

（2）OUT——输出指令

长格式为：OUT PORT，AL（字节）　　；（PORT）←（AL）（字节）

OUT PORT，AX（字）　　；（PORT+1，PORT）←（AX）（字）

短格式为：OUT DX，AL（字节）　；((DX))←（AL）（字节）

OUT DX，AX（字）　　；((DX）+1，(DX）←（AX）（字）

说明：

①长格式和短格式的规定与 IN 指令相同。

②OUT 指令不影响标志位。

（3）XLAT——换码指令

指令格式：XLAT OPR；16 位指令：（AL）←((BX）+（AL))

XLAT；上式的简写，OPR 为阅读程序用的表格首地址。

说明：

①在使用这条指令前，应先建立一个字节表格，表格的首地址应提前存入 BX 寄存器，需要转换的代码应该是相对于表格首地址的位移量也应提前存入 AL 寄存器中。表格的内容则是所要换取的代码，该指令执行后就可在 AL 中得到转换后的代码。

②XLAT 指令不影响标志位。

3.地址传送指令

（1）LEA——有效地址（EA）送寄存器指令

指令格式：LEA REG，SRC；（REG）←SRC

说明：

①指令把源操作数（只能是存储器寻址方式）指定的有效地址送到指令指定的 16 位或 32 位寄存器（REG）中（但不能是段寄存器）。

②LEA 指令不影响标志位。

（2）LDS、LES、LFS、LGS、LSS——地址指针送寄存器和相应段寄存器指令，以 LDS 为例

指令格式：LDS REG，SRC；（REG）←（SRC），（DS）←（SRC+2）或（DS）←（SRC+4）

说明：

①该组指令的源操作数只能用存储器寻址方式，根据任一种存储器寻址方式找到一个存储单元。

②该组指令不影响标志位。

4.标志寄存器传送指令

（1）LAHF——标志送 AH 指令

指令格式：LAHF；（AH）←（FLAGS 的低位字节）

（2）SAHF——AH 送标志寄存器指令

指令格式：SAHF；（FLAGS 的低位字节）←（AH）

（3）PUSHF——标志进栈指令

指令格式：PUSHF；（SP）←（SP）-2，((SP）+1，（SP））←（FLAGS）

（4）POPF——标志出栈指令

指令格式：POPF；（FLAGS）←((SP）+1，（SP)),（SP）←（SP）+2

说明：这组指令中 LAHF、PUSHF 不影响标志位。

5.类型转换指令

（1）CBW——字节转换为字指令

指令格式：CBW；（AH）←AL 内容的符号位，形成 AX 中的字。

（2）CWD——字转换为双字指令

指令格式：CWD；（DX）←AX 内容的符号位，形成 DX：AX 中的双字。.

（二）算术指令

8086 的算术运算指令包括二进制运算及十进制运算指令。算术指令用来执行算术运算，它们中有双操作数指令，也有单操作数指令。双操作数指令的两个操作数中除源操作数为立即数的情况外，必须有一个操作数在寄存器中。单操作数指令不允许使用立即数寻址方式。

1.加法指令

（1）ADD——加法指令

指令格式：ADDDST，SRC　　；（DST）←（DST）+（SRC）DST 表示目的操作数，SRC 表示源操作数

（2）ADC——带进位加法指令

指令格式：ADCDST，SRC　　；（DST）←（DST）+（SRC）+CF

（3）INC——加 1 指令

指令格式：INCOPR　　；（OPR）←（OPR）+1

说明：①.以上指令除 INC 不影响 CF 标志外，它们都影响条件标志位。

②.OF 是有符号数的溢出，CF 是无符号数的溢出。但 CF 可作为多位运算的进位标志。

2.减法指令

（1）SUB——减法指令

指令格式：SUB DST，SRC；（DST）←（DST）-（SRC）

（2）SBB——带借位减法指令

指令格式：SBB DST，SRC；（DST）←（DST）-（SRC）-CF

（3）DEC——减 1 指令

指令格式：DEC OPR；（OPR）←（OPR）-1

（4）NEG——求补指令

指令格式：NEG OPR；（OPR）←-（OPR）即（OPR）←0-（OPR）

（5）CMP——比较指令

指令格式：CMP OPR1，OPR2；（OPR1）-（OPR2），运算后根据结果影响标志

说明：

①以上指令除 DEC 不影响 CF 标志外，它们都影响条件标志位。

②OF 是有符号数的溢出，CF 是无符号数的溢出。但 CF 可作为多位运算的借位标志。

3.乘法指令

（1）MUL——无符号数乘法指令

指令格式：MUL SRC；

字节操作：（AX）←（AL）×（SRC）

字操作：（DX，AX）←（AX）×（SRC）

双字操作：（EDX，EAX）←（EAX）×（SRC）

4.除法指令

（1）DIV——无符号数除法指令

指令格式：DIV SRC；

字节操作：（AL）←（AX）/（SRC），（AH）←（AX）%（SRC）

字操作：（AX）←（DX，AX）/（SRC），（DX）←（DX，AX）%（SRC）

5.十进制调整指令

（1）BCD 码概述

①BCD 码（BinaryCodedDecimal）：用 4 位二进制数表示一位十进制数的编码方法。此处的 BCD 码当作无符号数计算（它可用一个单独的字节作为符号位）。

②压缩 BCD 码（packedBCDformat）：用一个字节 8 位的二进制数表示 2 个 BCD 码。又称为组合 BCD 码。

③非压缩 BCD 码（unpackedBCDformat）：用一个字节低 4 位表示 1 个 BCD 码，高 4 位没有意义。又叫扩展 BCD 码。因此数字的 ASCII 码是一种非压缩 BCD 码。

（2）压缩的 BCD 码调整指令

①DAA——加法的十进制调整指令：紧跟在 ADD 或 ADC 指令之后进行调整。

指令格式：DAA ；若（（（（AL）AND0FH）>9）OR（（AF）=1））则（AL）←（AL）+6，（AF）←1；

若（（（AL）>9FH）OR（（CF）=1））则（AL）←（AL）+60H，（CF）←1

②DAS——减法的十进制调整指令：紧跟在 SUB 或 SBB 指令之后进行调整。

指令格式：DAS ；若（（（（AL）AND0FH）>9）OR（（AF）=1））则（AL）←（AL）

–6，（AF）←1；

若（((AL）>9FH）OR（(CF）=1)）则（AL）←（AL）–60H，（CF）←1

说明：

①参加加、减运算的两个数一定为压缩 BCD 码，且加减法指令的目的操作数为 AL。

②除对 OF 标志位无定义外，影响其它所有条件码。

（3）非压缩的 BCD 码调整指令

①AAA——加法的 ASCII 调整指令：紧跟在 ADD 或 ADC 指令之后进行调整。

指令格式：AAA　　；若（(((AL）AND0FH）>9）OR（(AF）=1)）则（AL）←（AL）+6，（AH）←（AH）+1，

（AF）←1，（CF）←（AF），（AL）←（(AL）AND0FH)；否则（AL）←（(AL）AND0FH）

②AAS——减法的 ASCII 调整指令：紧跟在 SUB 或 SBB 指令之后进行调整。

指令格式：AAS　　；若（(((AL）AND0FH）>9）OR（(AF）=1)）则（AL）←（AL）–6，（AH）←（AH）–1，

（AF）←1，（CF）←（AF），（AL）←（(AL）AND0FH)；否则（AL）←（(AL）AND0FH）

说明：

①参加加、减运算的两个数一定为非压缩的 BCD 码，且加减法指令的目的操作数为 AL。

②影响 AF、CF，其余标志位无定义。

③AAM——乘法的 ASCII 调整指令：紧跟在 MUL 指令之后进行调整。

指令格式：AAM　　；（AH）←（AL）/0AH，（AL）←（AL）%0AH

说明：

①参加乘法运算的两数一定为高 4 位为 0 的非压缩 BCD 码，乘积在 AL 寄存器中。

②影响 SF、ZF、PF，其余标志位无定义。

③调整方法为：把 AL 寄存器的内容除以 0AH，商放在 AH 中，余数放在 AL 中。

④AAD——除法的 ASCII 调整指令：在 DIV 指令之前调整。在 DIV 指令之后再用 AAM 调整。

指令格式：AAD;（AL）←10×（AH）+（AL），（AH）←0

说明：

①被除数是存放在 AX 中的两位高 4 位为 0 的非压缩 BCD 码，除数也为高 4 位为 0 的非压缩 BCD 码。

②影响 SF、ZF、PF，其余标志位无定义。

③调整方法为：在 DIV 指令之前把两位高 4 位为 0 的非压缩 BCD 码的被除数调整为二进制数，再运行 DIV 指令，在 DIV 指令之后再用 AAM 指令将商调整为高 4 位为 0 的非压缩 BCD 码。

（三）逻辑指令

1.逻辑运算指令：可以对双字、字或字节执行按位的逻辑运算。

（1）AND——逻辑与指令

指令格式：ANDDST，SRC　　；（DST）←（DST）∧（SRC）

（2）OR——逻辑或指令

指令格式：ORDST，SRC　　；（DST）←（DST）∨（SRC）

（3）NOT——逻辑非指令

指令格式：NOTOPR　　；（OPR）←（$\overline{OPR}$）

（4）XOR——逻辑异或指令

指令格式：XORDST，SRC　　；（DST）←（DST）⊕（SRC）

（5）TEST——测试指令

指令格式：TEST OPR1，OPR2；（OPR1）∧（OPR2）

说明：

①DST、OPR、OPR1 不允许使用立即数寻址方式。

②DST 与 SRC 及 OPR1 与 OPR2 的双操作数指令不能同时是存储器操作数。

③NOT 指令不影响标志位。其它四条指令使 CF=OF=0，AF 无定义，SF、ZF、PF 则根据运算结果设置。

2.移位指令

（1）移位指令

①SHL——逻辑左移指令

指令格式：SHL OPR，CNT；

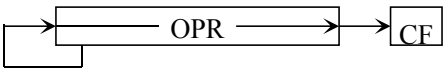
②SAL——算术左移指令

指令格式：SAL OPR，CNT；同上，算术左移是针对带符号的，逻辑左移是针对不带符号的

③SHR——逻辑右移指令

指令格式：SHR OPR，CNT；

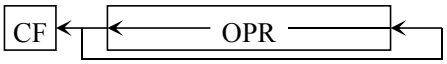
④SAR——算术右移指令



指令格式：SAR OPR，CNT；

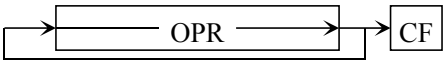
（2）循环移位指令

①ROL——循环左移指令



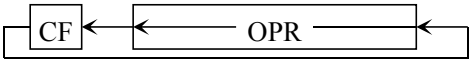
指令格式：ROL OPR，CNT；

②ROR——循环右移指令



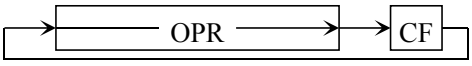
指令格式：ROR OPR，CNT；

③RCL——带进位位循环左移指令



指令格式：RCL OPR，CNT；

④RCR——带进位位循环右移指令



指令格式：RCR OPR，CNT；

说明：

①OPR 为除立即数以外的任意寻址方式。移位次数由 CNT 决定，CNT=1 只移位 1 次；若移位次数超过 1 次，在 8086 中则 CNT 必须用 CL 代替。而在其他机型中也可用 8 位立即数指定范围从 1～31 的移位次数。

②CF 位已在指令中给出其影响情况。OF 位只有在 CNT=1 时有效，一次移位前后的最高有效位（符

号位）发生变化则 OF=1，否则 OF=0。循环指令不影响其它条件。移位指令由结果影响 SF、ZF、PF，而 AF 无定义。

（四）串处理指令

用一条指令实现对一串字符或数据的操作。

1.与 REP 相配合工作的 MOVS、STOS、LODS、INS 和 OUTS 指令

（1）REP 重复串操作直到计数寄存器 CountReg（CX）的内容为 0 为止

指令格式：REPstringprimitive　；其中 stringprimitive 可为 MOVS，STOS，LODS，INS 或 OUTS

执行操作：

①如（CountReg）=0，则退出 REP，否则往下执行；

②（CountReg）←（CountReg）-1

③执行其后的串指令

④重复①～③

（2）MOVS——串传送指令

指令格式：

[REP]MOVS[ES：]DST，[Sreg：]SRC　；[Sreg：]为段跨越前缀。“[]”中为可选项。

[REP]MOVSB　；字节

[REP]MOVSW　；字

执行操作：((ES)：(Destination-index))←((Sreg)：(Source-index))。Sreg 缺省时为 DS。指针修改为：

字节操作：(Source-index)←(Source-index)±1，(Destination-index)←(Destination-index)±1

字操作：(Source-index)←(Source-index)±2，(Destination-index)←(Destination-index)±2

说明：

①在上述操作中，当方向标志 DF=0 时用”＋”，DF=1 时用”－”。可以使用指令 CLD 使 DF=0，STD 使 DF=1。

②Source-index 为源变址寄存器，当其地址长度为 16 位时用 SI 寄存器。

③Destination-index 为目的变址寄存器，当其地址长度为 16 位时用 DI 寄存器。以上 3 条说明适用于所有的串操作指令。

④该指令不影响条件码。

补充说明：在执行串操作指令之前，应该先做好以下准备工作：

①把存放在数据段中的源串首地址（如反向传送则为末地址）放入源变址寄存器中；

②把将要存放数据串的附加段中的目的串首地址(或反向传送时的末地址)放入目的变址寄存器中；

③把数据串长度放入计数寄存器（CX)；

④建立方向标志。在完成这些准备工作后，就可以使用串指令传送信息了。

（3）STOS——存入串指令

指令格式：

[REP]STOS[ES：]DST

[REP]STOSB ；字节

[REP]STOSW ；字

字节操作：((Destination-index))←（AL），（Destination-index）←（Destination-index）±1

字操作：((Destination-index))←（AX），（Destination-index）←（Destination-index）±2

（4）LODS——从串取指令

指令格式：

LODS[Sreg：]SRC ；该指令与 REP 连用无多大意义

LODSB ；字节

LODSW ；字

字节操作：（AL）←（Source-index），（Source-index）←（Source-index）±1

字操作：（AX）←（Source-index），（Source-index）←（Source-index）±2

（5）INS——串输入指令

指令格式：

[REP]INS[ES：]DST，DX

[REP]INSB ；字节

[REP]INSW ；字

字节操作：((Destination-index))←((DX))（字节），（Destination-index）←（Destination-index）±1

字操作：((Destination-index))←((DX))（字），（Destination-index）←（Destination-index）±2

（6）OUTS——串输出指令

指令格式：

[REP]OUTSDX，[Sreg：]SRC

[REP]OUTSB ；字节

[REP]OUTSW ；字

字节操作：((DX))←（Source-index）（字节），（Source-index）←（Source-index）±1

字操作：((DX))←（Source-index）（字），（Source-index）←（Source-index）±2

2.与 REPE/REPZ 和 REPNE/REPNZ 联合工作的 CMPS 和 SCAS 指令

（1）REPE/REPZ 当相等/为零时重复串操作

指令格式：REPE（或 REPZ）stringprimitive；其中 stringprimitive 可为 CMPS 或 SCAS

执行操作：

①如（CountReg）=0 或 ZF=0（即某次比较的结果两个操作数不等）时退出，否则往下执行；

②（CountReg）←（CountReg）-1

③执行其后的串指令

④重复①～③

（2）REPNE/REPNZ 当不相等/不为零时重复串操作

指令格式：REPNE（或 REPNZ）stringprimitive；其中 stringprimitive 可为 CMPS 或 SCAS

执行操作：

①如（CountReg）=0 或 ZF=1（即某次比较的结果两个操作数相等）时退出，否则往下执行；

②（CountReg）←（CountReg）-1

③执行其后的串指令

④重复①～③

（3）CMPS——串比较指令

指令格式：

[REPE/REPNE]CMPS[Sreg：]SRC，[ES：]DST

[REPE/REPNE]CMPSB　　；字节

[REPE/REPNE]CMPSW　　；字

执行操作：((Sreg)：(Source-index)) –((ES)：(Destination-index))。Sreg 缺省时为 DS。指针修改为：

字节操作：（Source-index）←（Source-index）±1，（Destination-index）←（Destination-index）±1

字操作：（Source-index）←（Source-index）±2，（Destination-index）←（Destination-index）±2

（4）SCAS——串扫描指令

指令格式：

[REPE/REPNE]SCAS[ES：]DST

[REPE/REPNE]SCASB　　；字节

[REPE/REPNE]SCASW　　；字

字节操作：（AL）－（Destination-index），（Destination-index）←（Destination-index）±1

字操作：（AX）－（Destination-index），（Destination-index）←（Destination-index）±2

说明：

①串处理指令在不同的段之间传送或比较数据，如果需要在同一段内处理数据，可以在 DS 和 ES 中设置同样的地址，或者在源操作数字段使用段跨越前缀来实现。

②当使用重复前缀时（CountReg）是每次减 1 的，因此对于字或双字指令来说，预置时设置的值应该是字或双字的个数而不是字节数。

（五）控制转移指令

1.无条件转移指令

JMP——跳转指令，无条件地转移到指令指定的地址去执行从该地址开始的指令。分为段内转移（在同一段的范围内进行转移）和段间转移（转到另一个段去执行程序）。

（1）段内直接短转移

指令格式：JMP SHORT OPR

执行操作：（IP）←（IP）+8 位位移量

说明：转移的目标地址 OPR 可直接使用符号地址（一个短标号），而在机器执行时则是当前的 IP 的值（即 JMP 指令的下一条指令的地址）与指令中指定的 8 位位移量之和。相对位移量为 8 位（只需在–128～+127 字节范围内转移）。（条件转移只能用此方式。）

（2）段内直接近转移

指令格式：JMP NEAR PTR OPR

执行操作：（IP）←（IP）+16 位位移量

说明：指令中给出一个相对位移量（实际是一个近标号），这样有效转移地址为 IP 的当前值再加上一个 16 位的位移量。OPR 可直接使用符号地址。

（3）段内间接近转移

指令格式：JMP WORD PTR OPR

执行操作：（IP）←（EA）

说明：有效地址 EA 值由 OPR 的寻址方式确定。它可以使用除立即数方式以外的任一种寻址方式。如果指定的是寄存器，则把寄存器的内容送到 IP 寄存器中；如果指定的是存储器中的一个字，则把该存储单元的内容送到 IP 寄存器中。

（4）段间直接远转移

指令格式：JMP FAR PTR OPR

执行操作：（IP）←OPR 的段内偏移地址，（CS）←OPR 所在段的段地址

说明：指令的操作数是一个远标号，该标号在另一个代码段内。指令的操作是将标号的偏移地址送 IP，段地址送 CS。

（5）段间间接转移：指令的操作数为一个 32 位的存储器地址。指令的操作是将存储器的前两个字节送 IP，后两个字节送 CS，以实现到另一个代码段的转移。

指令格式：JMP DWORD PTR OPR

执行操作：（IP）←（EA），（CS）←（EA+2）

说明：有效地址 EA 值由 OPR 的寻址方式确定。它只能使用任一种存储器寻址方式。根据寻址方式求出 EA 后，把指定存储单元的字内容送到 IP 寄存器，并把下一个字的内容送到 CS 寄存器，这样就实现了段间跳转。

2.条件转移指令

以某一个标志位的值或者某几个标志位的值作为判断是否转移的依据。如果满足指令中所要求的条件，则产生转移；否则往下执行排在条件转移指令后面的一条指令。只能使用段内直接短转移的寻址方式。

指令格式：Jcc short_label；cc 表示条件，具体分为如下 4 组：

（1）根据单个条件标志的设置情况转移。这组包括 10 种指令。它们一般适用于测试某一次运算的结果并根据其不同特征产生程序分支作不同处理的情况。

①JZ（或 JE）——结果为零（或相等）则转移

指令格式：JZ（或 JE）OPR

测试条件：ZF=1

②JNZ（或 JNE）——结果不为零（或不相等）则转移

指令格式：JNZ（或 JNE）OPR

测试条件：ZF=0

③JS——结果为负则转移

指令格式：JS OPR

测试条件：SF=1

④JNS——结果不为负则转移

指令格式：JNS OPR

测试条件：SF=0

⑤JO——结果溢出则转移

指令格式：JO OPR

测试条件：OF=1

⑥JNO——结果不溢出则转移

指令格式：JNO OPR

测试条件：OF=0

⑦JP（或 JPE）——奇偶位为 1（偶数个 1）则转移

指令格式：JP（或 JPE）OPR

测试条件：PF=1

⑧JNP（或 JPO）——奇偶位为 0（奇数个 1）则转移

指令格式：JNP（或 JPO）OPR

测试条件：PF=0

⑨JB（或 JNAE，或 JC）——低于，或者不高于或等于，或进位为 1 则转移

指令格式：JB（或 JNAE，或 JC）OPR

测试条件：CF=1

⑩JNB（或 JAE，或 JNC）——不低于，或者高于或等于，或进位为 0 则转移

指令格式：JNB（或 JAE，或 JNC）OPR

测试条件：CF=0

（2）比较两个无符号数，并根据比较的结果转移

①JB（或 JNAE，或 JC）——低于，或者不高于或等于，或进位为 1 则转移（即“＜”）

指令格式：JB（或 JNAE，或 JC）OPR

测试条件：CF=1

②JNB（或 JAE，或 JNC）——不低于，或者高于或等于，或进位为 0 则转移（即“≥”）

指令格式：JNB（或 JAE，或 JNC）OPR

测试条件：CF=0

③JBE（或 JNA）——低于或等于，或不高于则转移（即”≤”）

指令格式：JBE（或 JNA）OPR

测试条件：CF∨ZF=1

④JNBE（或 JA）——不低于或等于，或者高于则转移（即”＞”）

指令格式：JNBE（或 JA）OPR

测试条件：CF∨ZF=0

（3）比较两个带符号数，并根据比较的结果转移

①JL（或 JNGE）——小于，或者不大于或等于则转移（即“＜”）

指令格式：JL（或 JNGE）OPR

测试条件：SF⊕OF=1

②JNL（或 JGE）——不小于，或者大于或等于则转移（即“≥”）

指令格式：JNL（或 JGE）OPR

测试条件：SF⊕OF=0

③JLE（或 JNG）——小于或等于，或不大于则转移（即“≤”）

指令格式：JLE（或 JNG）OPR

测试条件：（SF⊕OF）∨ZF=1

④JNLE（或 JG）——不小于或等于，或者大于则转移（即“＞”）

指令格式：JNLE（或 JG）OPR

测试条件：（SF⊕OF）∨ZF=0

（4）测试 CX 的值为 0 则转移指令

JCXZ——CX 寄存器的内容为 0 则转移

指令格式：JCXZOPR

测试条件：（CX）=0

3.循环指令

（1）LOOP——循环指令

指令格式：LOOP OPR

测试条件：（CountReg）≠0

（2）LOOPZ/LOOPE——当为 0 或相等时循环指令

指令格式：LOOPZ（或 LOOPE）OPR

测试条件：（CountReg）≠0 且（ZF）=1

（3）LOOPNZ/LOOPNE——当不为 0 或不相等时循环指令

指令格式：LOOPNZ（或 LOOPNE）OPR

测试条件：（CountReg）≠0 且（ZF）=0

说明：

这三条指令的执行步骤是：

①（CountReg）←（CountReg）–1

②检查是否满足测试条件，如满足且操作数长度为 16 位，则（IP）←（IP）+D8 的符号扩展；

4.子程序

为便于模块化程序设计，往往把程序中某些具有独立功能的部分编写成独立的程序模块，称为子程序（也称为过程）。为此需要提供子程序（过程）的调用和返回指令。

（1）CALL——子程序（过程）调用指令

①段内直接近调用：指令操作数是一个近过程，该过程在本段内。

指令格式：CALL DST

执行操作：当操作数长度为 16 位时，Push（IP）

（IP）←（IP）+D16

说明：指令中 DST 给出转向地址（即子程序的入口地址），D16 即为机器指令中的位移量。

②段内间接近调用：指令的操作数是一个寄存器或存储器地址，其内容是一近过程的入口地址。

指令格式：CALL DST

执行操作：当操作数长度为 16 位时，Push（IP）

（IP）←（EA）

说明：指令中 DST 为除立即数以外的任一种寻址方式，由指定的寄存器或存储单元的内容给出转向地址。

③段间直接远调用：指令的操作数是一个远过程，该过程在另一个代码段内。

指令格式：CALL DST

执行操作：当操作数长度为 16 位时，Push（CS）

Push（IP）

（IP）←DST 指定的偏移地址

（CS）←DST 指定的段地址

④段间间接远调用：指令的操作数是一个存储器地址。

指令格式：CALLDST

执行操作：当操作数长度为 16 位时，Push（CS）

Push（IP）

（IP）←（EA）

（CS）←（EA+2）

说明：指令中 DST 为任一种存储器寻址方式，由指定的存储单元的内容给出转向地址。

（2）RET——子程序（过程）返回指令

RET 指令放在子程序的末尾，它使子程序在功能完成后返回调用程序继续执行，而返回地址是调用程序调用子程序（或称转子）时存放在堆栈中的，因此 RET 指令的操作是返回地址出栈送 IP 寄存器（段内或段间）和 CS 寄存器（段间）。

①段内近返回

指令格式：RET　　；DEBUG 反汇编为 RET，机器码为 C3H

执行操作：当操作数长度为 16 位时，（IP）←Pop（）

②段内带立即数近返回

指令格式：RETEXP　　；DEBUG 反汇编为 RETn，机器码为 C2xxxxH

执行操作：在完成与①的 RET 完全相同的操作后，还需要修改堆栈指针：

（SP）←（SP）+D16

说明：其中 EXP 是一个表达式，根据它的值计算出来的常数成为机器指令中的位移量 D16。主程

序通过压入堆栈操作将一定的参数或参数地址传递给子程序。子程序运行过程中，使用了这些参数或参数地址，子程序返回时没有必要再将这些参数或参数地址保留在堆栈中，因而，可以在返回指令后面加上参数 EXP，以腾出那些无用的参数或参数地址所占的单元。

③段间远返回

指令格式：RET ；DEBUG 反汇编为 RETF，机器码为 CBH

执行操作：当操作数长度为 16 位时，（IP）←Pop（）

（CS）←Pop（）

④段间带立即数远返回

指令格式：RETEXP ；DEBUG 反汇编为 RETFn，机器码为 CAxxxxH

执行操作：在完成与③的 RET 完全相同的操作后，还需要修改堆栈指针：

（SP）←（SP）+D16

说明：

①从近过程返回和从远过程返回的指令是一样的，但机器编码不一样。段内返回指令的代码为 C3H，段间返回指令的代码为 CBH。

②RETEXP 指令为带参数的返回指令；EXP 可为 0～FFFFH 范围中的任何一个偶数。

5.中断

（1）中断的概念

①中断：当系统运行或者程序运行期间遇到某些特殊情况时，需要计算机自动执行一组专门的服务程序来进行处理，这种情况称为中断。中断分为内部中断和外部中断。

A.内部中断：由中断指令或者是程序运行结果产生的中断称为内部中断。

B.外部中断：由于外部 I/O 设备随机请求而产生的中断称为外部中断。

②中断服务程序：在中断中运行的一组服务程序称为中断服务程序或中断子程序。

③中断向量：中断服务程序的入口地址。

④中断类型码 N：为了区分各种中断而给每个中断按序从 0～FFH 编的号码称为中断类型码，用 N 表示。

⑤中断向量表：8086/8088CPU 将所有的中断向量按中断类型码顺序存放在内存的起始 1KB 地址单元中，这 1KB 地址单元就称为中断向量表。

（2）INT——中断调用指令

指令格式：INT TYPE

或 INT ；TYPE=3 时，缺省

执行操作：Push（FLAGS）

IF←0

TF←0

AC←0

Push（CS）

Push（IP）

（IP）←（TYPE*4）（每个中断向量占 4 个字节）

（CS）←（TYPE*4+2）

说明：

①其中 TYPE 为类型号，它可以是常数或常数表达式，其值需在 0～255 范围内。

②类型 0 的中断称为除数为 0 中断，由 CPU 自动产生，不能用中断指令调用。

③类型 1 的中断称为单步中断，CPU 进入单步中断的依据为（TF）=1。不能用中断指令来调用。单步中断由调试程序 DEBUG 使用。

④类型 2 的中断称为非屏蔽中断，属硬件中断，紧急情况使用，不许用中断指令来调用。

⑤类型 3 的中断称为断点中断。用在调试程序中。INT 又称为断点中断指令，它是单字节指令。与其他 INT TYPE 不同，是双字节指令。

⑥类型 4 的中断称为溢出中断。有专门的溢出中断调用指令 INTO。无 INT4 指令。见下面。

（3）INTO——若溢出则中断指令

指令格式：INTO

执行操作：若 OF=1，则：Push（FLAGS）

IF←0

TF←0

AC←0

Push（CS）

Push（IP）

（IP）←（10H）

（CS）←（12H）

（4）IRET——从中断返回指令

指令格式：IRET　　；适用于操作数长度为 16 位的情况

执行操作：（IP）←Pop（）

（CS）←Pop（）

（FLAGS）←Pop（）

（六）处理器控制与杂项操作指令

1.标志处理指令

（1）CLC——清进位标志指令

指令格式：CLC　　；（CF）←0

（2）CMC——对进位标志求反指令

指令格式：CMC　　；（CF）←（$\overline{CF}$）

（3）STC——置进位标志指令

指令格式：STC　　；（CF）←1

（4）CLD——清方向标志指令

指令格式：CLD　　；（DF）←0

（5）STD——置方向标志指令

指令格式：STD　　；（DF）←1

（6）CLI——清中断允许标志指令

指令格式：CLI　　；（IF）←0

（7）STI——置中断允许标志指令

指令格式：STI　　；（IF）←1

（8）设置单步标志程序段：置（TF）=1 程序。

```
PUSHF
POP AX          ；（AX）←（FLAGS）
OR AX，0100H        ；（TF）←1
PUSH AX
POPF
```

（9）清除单步标志程序段：清（TF）=0 程序。

```
PUSHF
POP AX          ；（AX）←（FLAGS）
AND AX，0FEFFH      ；（TF）←0
PUSH AX
POPF
```

2.其它处理器控制与杂项操作指令

（1）NOP（Nooperation）——空操作指令：CPU 执行指令时不进行任何操作，但占用 3 个时钟周期和一个字节的空间，然后继续执行下一条指令。起延时作用或为其他指令保留存储空间。

指令格式：NOP　　；空操作，占用 3 个时钟周期

（2）HLT（Halt）——暂停指令：执行 HLT 指令后，CPU 进入暂停状态，CS 和 IP 指向 HLT 后面的一条指令的地址。外部中断或复位信号 RESET 可使 CPU 退出暂停状态。

指令格式：HLT　　；暂停指令的执行

（3）ESC（Escape）——交权指令

指令格式：ESCext_op，reg/mem　　；ext_op 是外操作码（协处理器的操作码）

（4）WAIT（WaitwhileTESTpinnotasserted）——等待指令：8086 在执行 WAIT 指令的过程中，不断检测 TEST 引腿上的信号;而协处理器在完成工作以后，会往 8086 的 TEST 引腿送入一个低电平信号。8086 检测到此信号以后，便退出等待状态。

指令格式：WAIT　　；等待协处理器操作结束

①外部中断可使 CPU 离开等待状态，但中断返回后又回到等待状态。

②一般在用 ESC 指令前先用一条 WAIT 指令，以防 8086 同时让协处理器干两件及两件以上的事。这是协处理器不允许的。

（5）LOCK（Lockbus）——总线封锁指令：指令前缀。

①LOCK 指令前缀是一个特殊的可以放在任何指令前面的单字节指令前缀。

②该指令前缀迫使 CPU 封锁总线，并在 $\overline{\text{LOCK}}$ 线输出低电平，直到执行完前缀后面的指令为止。外部硬件可接收这个 $\overline{\text{LOCK}}$ 信号，而无法得到总线控制权。

第三节 汇编语言程序格式和伪指令

一、汇编程序功能

（一）汇编程序（MASM）：把用户编写的汇编语言源程序翻译成机器语言目标程序的一种系统程序。

（二）汇编语言源程序：用汇编语言编写的程序称为汇编语言的源程序（扩展名为“.ASM”）。

（三）汇编程序的作用：把汇编语言源程序转换成用二进制代码表示的目标文件（称为“.OBJ”文件）。

（四）在计算机上运行汇编语言程序的步骤是

1.用编辑程序建立 ASM 源文件；

2.用 MASM 程序把 ASM 文件转换成 OBJ 文件；

3.用 LINK 程序把 OBJ 文件转换成 EXE 文件；

4.用 DOS 命令直接键入文件名就可执行该程序。

（五）汇编程序的主要功能

1.检查汇编语言源程序。

2.测出源程序中的语法错误，并给出出错信息。

3.产生源程序的目标程序，并给出列表文件。

4.展开宏指令。

二、伪操作

伪操作（伪指令）：用来为汇编程序提供某些信息，让汇编程序在汇编过程中执行某些特定功能的指令叫伪指令。它不是 CPU 指令系统中的指令。要注意的是：指令是指挥 CPU 执行什么操作，而伪操作是指挥 MASM 怎样工作。

（一）处理器选择伪操作

这一组伪操作的功能是要告诉汇编程序应该选择哪一种指令系统。

.8086　　；选择 8086 指令系统，缺省时的默认值即为此

（二）段定义伪操作

1.完整的段定义伪操作

（1）SEGMENT/ENDS——段定义伪操作：此对伪操作可以将汇编语言源程序分成几个段，通常为数据段、堆栈段、附加段和代码段。

伪操作格式：segnameSEGMENT[align_type][combine_type][use_type][‘class’]

┆

segnameENDS

说明：

①定位类型（align_type）：说明段的起始地址应有怎样的边界值。它们可以是：

PARA：指定段的起始地址必须从小段边界开始，即段地址必须能被 16 整除。这样起始偏移地址可

以从 0 开始。缺省时默认为 PARA。

BYTE：该段可以从任意地址开始。这样起始偏移地址可能不是 0。

WORD：该段必须从字的边界开始，即段地址必须为偶数。

DWORD：该段必须从双字的边界开始，即段地址必须能被 4 整除。

PAGE：该段必须从页的边界开始，即段地址必须能被 256 整除。

②组合类型（combine_type）：说明程序连接时的段合并方法。它们可以是：

PRIVATE：该段为私有段（默认），在连接时将不与其它模块中的同名分段合并。

PUBLIC：该段连接时将与有相同名字的其它分段连接在一起。连接次序由 LINK 指定。每一分段都从小段的边界开始，因此原有段之间有空隙。

COMMON：该段连接时与其它同名分段有相同的起始地址，所以会产生覆盖。COMMON 连接后的长度是各分段中长度最长的那个段的长度。

ATexpression：使段起始地址是表达式计算出来的 16 位值。不能指定代码段。

MEMORY：与 PUBLIC 同义。

STACK：把不同模块中的同名段组合而形成一个堆栈段。其长度为原有段的长度总和。栈顶可自动指向连接后形成的大堆栈段的栈顶。

③使用类型（use_type）：只适用于 386 及其后继机型

④类别（‘class’）：在引号中给出连接时用于组成段组的类型名。类别说明并不能把相同类别的段合并起来，但在连接后形成的装入模块中，可以把它们的位置靠在一起。

（2）ASSUME——段指定伪操作：告诉汇编程序，段和段寄存器的对应关系。

伪操作格式：ASSUME 分配（assignment），…，assignment

说明：其中 assignment 说明分配情况，其格式为：

段寄存器名：段名字[，段寄存器名：段名字[，……]]

ASSUMENOTHING 则可取消前面由 ASSUME 所指定的段寄存器。

2.存储模型与简化段定义伪操作

（1）MODEL——存储模型（memory_model）伪操作，即用来说明在存储器中是如何安放各个段的。

伪操作格式：.MODELmemory_model[，modeloptions]

说明：根据它们的不同组合，可以建立如下七种存储模型：

①Tiny 所有数据和代码都放在一个段内，其数据和代码都是近访问。

②Small 所有数据放在一个 64KB 的数据段内，所有代码放在另一个 64KB 的代码段内，数据和代码也都是近访问的。这是一般最常用的一种模型。

③Medium 代码使用多个段，一般一个模块一个段，而数据则合并成一个 64KB 的段组。这样，数据是近访问的，而代码则可远访问。

④Compact 所有代码都放在另一个 64KB 的代码段内，数据则可放在多个段内，形成代码是近访问的，而数据则可远访问的格式。

⑤Large 代码和数据都可用多个段，所以数据和代码都可以远访问。

⑥Huge 与 Large 模型相同，其差别是允许数据段的大小超过 64KB。

⑦Flat 允许用户用 32 位偏移量，但 DOS 下不允许使用这种模型，只能在 OS/2 下或其他保护模式的操作系统下使用。MASM5 不支持，但 MASM6 支持。

modeloptions 允许用户指定三种选项：高级语言接口、操作系统和堆栈距离。

①高级语言接口是指该汇编语言程序作为某一种高级语言程序的过程而为该高级语言程序调用时，应该用如 C，BASIC，FORTRAN，PASCAL 等来加以说明。

②操作系统是要说明程序运行于哪个操作系统之下，可用 OS_DOS 或 OS_OS2 来说明，默认项是 OS_DOS。

③堆栈距离可用 NEARSTACK 或 FARSTACK 来说明。其中 NEARSTACK 是指把堆栈段和数据段组合到一个 DGROUP 段中，DS 和 SS 均指向 DGROUP 段；FARSTACK 是指堆栈段和数据段并不合并。

（2）简化的段定义伪操作

①汇编程序给出的标准段有下列几种：这种分段方法把数据段分得更细：一是把常数段和数据段分开；二是把初始化数据段和未初始化数据段分开（其中初始化数据段是指程序中已指定初始值的数据）；三是把近和远的数据段分开。这样做的结果可便于与高级语言兼容。

Acode	代码段
Binitializeddata	初始化数据段
Cuninitializeddata	未初始化数据段
Dfarinitializeddata	远初始化数据段
Efaruninitializeddata	远未初始化数据段
Fconstants	常数段
Gstack	堆栈段

②对应以上的标准段，可有如下简化段伪操作

伪操作格式：.CODE[name]　　；对于一个代码段的模型，段名为可选项；

对于多个代码段的模型，则应为每个代码段指定段名

.DATA

.DATA?

.FARDATA[name] ；可指定段名。如不指定，则将以 FAR_DATA 命名。

.FARDATA?[name]；可指定段名。如不指定，则将以 FAR_BSS 命名。

.CONST

.STACK[size]　　；可指定堆栈段大小。如不指定，则默认值为 1KB。

说明：当使用简化段伪操作时，必须在这些简化段伪操作出现之前，即程序的一开始先用.MODEL 伪操作定义存储模型，然后再用简化段伪操作定义段。每一个新段的开始就是上一段的结束，而不必用 ENDS 作为段的结束符。

（2）与简化段定义有关的预定义符号：汇编程序给出了与简化段定义有关的一组预定义符号，它们可在程序中出现，并由汇编程序识别使用。如预定义符号@data 就给出了数据段的段名。另有一些预定义符号，也可与条件汇编伪操作相配合，以帮助用户编写一些较为复杂的代码。

3GROUP——段组定义伪操作：在各种存储模型中，汇编程序自动地把各数据段组成一个段组DGROUP，以便程序在访问各数据段时使用一个数据段寄存器 DS。GROUP 伪操作允许用户自行指定段组，其格式如下：

伪操作格式：grpnameGROUPsegname[，segname…]

说明：其中 grpname 为段组名，segname 则为段名。

（三）程序开始和结束伪操作

1.NAME——指定模块名伪操作

伪操作格式：NAMEmodule_name　；汇编程序将以给出的模块名作为该模块的名字

2.TITLE——指定打印标题伪操作：指定每一页上的打印标题。同时，如果程序中没有 NAME 伪操作，则汇编程序将用 text 中的前 6 个字符作为模块名。text 最多可有 60 个字符。如果程序中既没有 NAME 又没有 TITLE 伪操作，则将用源文件名作为模块名。所以 NAME 和 TITLE 伪操作并非必须的。

伪操作格式：TITLEtext

3.END——源程序结束伪操作

伪操作格式：END[label]　　；其中 label 指示程序开始执行的起始地址

说明：如果多个程序模块相连接，则只有主程序要使用标号，其他子程序模块则只用 END 而不必指定标号。

4.MASM6.0 版的汇编程序还增加了定义程序的入口点和出口点的伪操作

伪操作格式：.STARTUP　　；用来定义程序的初始入口点，并且产生设置 DS、SS 和 SP 的代码。

伪操作格式：.EXIT[return_value]　；用来产生退出程序并返回操作系统的代码，其中 return_value 为返回给操作系统的值。常用 0 作为返回值。

（四）数据定义及存储器分配伪操作

此类伪操作的通用格式为：

[Variable]MnemonicOperand[，Operand…][；Comments]

1.变量名（Variable）：可有可无，用符号地址表示。后面跟空格或 TAB 分隔符，不能跟”:”。其值等于第一个字节的偏移地址。

2.注释（Comments）：可有可无，用于说明该伪操作的功能。

3.助记符（Mnemonic）：说明所用伪操作的助记符名称同时也说明所定义的数据类型。通常为下面 6 种。

（1）DB（DefineByte）：定义字节伪操作，其后的每个操作数都占有一个字节（8 位）。

（2）DW（DefineWord）：定义字伪操作，其后的每个操作数都占有一个字（16 位）。

（3）DD（DefineDoubleWord）：定义双字伪操作，其后的每个操作数都占有一个双字（32 位）。

（4）DF（DefineFarPointer）：定义 6 个字节伪操作，用于存放远地址指针，其后的每个操作数都占有 6 个字节（48 位）。只能用于 386 及其后继机型。

（5）DQ（DefineQuadWord）：定义 4 字伪操作，其后的每个操作数都占有 4 个字（64 位）。

（6）DT（DefineTenBytes）：定义十字节伪操作，其后的每个操作数都占有十字节，形成压缩 BCD 码。

4.操作数（Operand）：一个常数、一个其值为常数的表达式，或是一个字符串常数。可以是问号”?”及重复操作符 DUP 等。

（1）”表达式”作为操作数（常数是一种特殊的表达式）：可用定义的变量或标号组成表达式。

①变量±常数表达式

②标号±常数表达式

（2）字符串作为操作数。

（3）“?”操作数：仅给变量保留相应的存储空间，而不赋给变量初值。

（4）重复操作符 DUP：用来指定某个（或某些）操作数重复的次数（还可以嵌套）。形式为：

repeat_countDUP（初值[，初值…]）；repeat_count 为重复的次数

（5）PTR 属性操作符：指定操作数的类型属性为 BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE 类型。

操作符格式：类型（Type）PTR 变量（Variable）±常数表达式（Constantexpression）

（6）LABEL——伪操作：指定一个操作数具有不同的类型属性。

伪操作格式：nameLABELType

说明：

①对于数据项，name 用变量名（variable_name）表示。类型为 BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE。

②对于可执行的程序代码，name 用标号名（label_name）表示。类型为 NEAR、FAR。

（五）表达式赋值伪操作 EQU、=：（又称为符号定义伪操作、符号赋值伪操作。）

有时程序中多次出现同一个表达式，为方便起见可以用赋值伪操作给表达式赋予一个名字。

1.EQU——表达式赋值伪操作：用来对一个名字进行赋值。但不能对一个已赋值的名字重新赋值。

伪操作格式：表达式名字 EQU 表达式　　；表达式可以为常数或者数据的地址

说明：PURGE 语句：用来解除对某一个名字的赋值。以后此名字才可由 EQU 重新赋值。

伪操作格式：PURGE 原名字　　；原名字，即前面已赋过值的名字

2.=——表达式赋值伪操作：”=”（等号）伪操作的功能与 EQU 伪操作基本相同，主要区别在于它可以对同一个名字重新赋值。

伪操作格式：名字=表达式　　；表达式可以为常数或者数据的地址

（六）地址计数器与对准伪操作

1.$——地址计数器：在汇编程序对源程序汇编的过程中，使用地址计数器（locationcounter）来保存当前正在汇编的指令的偏移地址。地址计数器的值可用$来表示，汇编语言允许用户直接用$来引用地址计数器的值。

（1）当$用在指令中时，它表示本条指令的第一个字节的地址。

（2）当$用在伪操作的参数字段时，则和它用在指令中的情况不同，它表示的是地址计数器的当前值。

2.ORG——地址对准伪操作：用来规定下一个字节的地址成为常数表达式的值。

伪操作格式：ORG 常数表达式（Constantexpression）

（七）基数控制伪操作

1.汇编程序默认的数为十进制数。二进制数要以字母 B 结尾；八进制数要以字母 O 或 Q 结尾；十进制数要以字母 D 结尾或缺省；十六进制数要以字母 H 结尾，当首字符为 A～F 时前面必须加 0。

2.RADIX——基数控制伪操作：可以把默认的基数改变为 2～16 范围内的任何基数。

伪操作格式：.RADIX 表达式（expression） ；表达式为十进制数，用来表示基数值

3.字符串可以看成串常数，要用成对的单引号或双引号作为定界符，得到的是字符串的 ASCII 码值。

三、汇编语言程序格式

汇编语言源程序中的每个字句可以由 4 项组成。格式如下：

[名字（name）]操作（operation）操作数（operand）[；注释（comment）]

（一）名字项

1.源程序中可用于名字的字符有：

（1）字母 A～Z，a～z。

（2）数字 0～9。数字不能用在名字的最前面。

（3）专用字符？、.、@、_、$。如果用到”.”则必须为名字的第一个字符。

2.名字项可以是标号或变量，用来表示本语句的符号地址。同一个名字不可重复定义。

（1）标号：在代码段中定义，后面必须跟冒号”:”。标号有三种属性：段、偏移及类型。

①段属性：定义标号的段起始地址，此值必须在一个段寄存器中，而标号的段总是在 CS 寄存器中。

②偏移属性：标号的偏移地址是从段起始地址到定义标号的位置之间的字节数。

③类型属性：用来指出该标号是在本段内（NEAR）引用，还是在其它段（FAR）引用。

（2）变量：在除代码段以外的其它段中定义，后面不能跟冒号。变量经常在操作数字段出现。变量也有三种属性：段、偏移及类型。

①段属性：定义变量的段起始地址，此值必须在一个段寄存器中。

②偏移属性：变量的偏移地址是从段起始地址到定义变量的位置之间的字节数。在当前段内给出变量的偏移值等于当前地址计数器的值，当前地址计数器的值可以用$来表示。

③类型属性：变量的类型属性定义该变量所保留的字节数。

（二）操作项

指令、伪操作或宏指令的助记符。

（三）操作数项

由一个或多个表达式组成，多个操作数项之间一般用逗号分开。操作数项可以是常数、寄存器、标号、变量或由表达式组成。表达式经汇编后有确定的结果，表达式是常数、寄存器、标号、变量与一些操作符相组合的序列，可以有数字表达式和地址表达式两种。用于组合表达式的操作符有：

1.算术操作符：+、-、*、/、MOD。MOD 为取余运算。地址表达式只能用+、-、*、/。

2.逻辑与移位操作符

（1）逻辑操作符：AND、OR、XOR、NOT。逻辑运算为按位运算，只能用于数字表达式中。

（2）移位操作符：汇编程序将 expression 左移或右移 numshift 位，如移位数大于 15，则结果为 0。

操作符格式：expressionSHL（或 SHR）numshift

3.关系操作符：EQ（相等）、NE（不等）、LT（小于）、GT（大于）、LE（小于或等于）、GE（大于

或等于）6 种。关系操作符的两个操作数必须都是数字或是同一段内的两个存储器地址。计算结果为逻辑值，结果为真，表示为 0FFFFH；结果为假，表示为 0。

4.数值回送操作符：TYPE、LENGTH、SIZE、OFFSET、SEG 等。

（1）TYPE：如果是变量，则汇编程序将回送该变量的以字节数表示的类型：DB 为 1，DW 为 2，DD 为 4，DF 为 6，DQ 为 8，DT 为 10。如果是标号，则汇编程序将回送代表该标号类型数值：NEAR 为-1，FAR 为-2。如果表达式为常数，则应回送 0。

操作符格式：TYPEexpression

（2）LENGTH：对于变量中使用 DUP 的情况，汇编程序将回送分配给该变量的单元数。否则回送 1。

操作符格式：LENGTH 变量（variable）

（3）SIZE：汇编程序将回送分配给该变量的字节数。但是此值是 LENGTH 值和 TYPE 值的乘积。

操作符格式：SIZE 变量（variable）

（4）OFFSET：汇编程序将回送变量或标号的偏移地址值。

操作符格式：OFFSET 变量（variable）或标号（label）

（5）SEG：汇编程序将回送变量或标号的段地址值。

操作符格式：SEG 变量（variable）或标号（label）

5.属性操作符：PTR、段操作符、SHORT、THIS、HIGH、LOW、HIGHWORD 和 LOWWORD 等。

（1）PTR：可用以指定存储器操作数的类型。通常和伪操作 BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE 等连起来使用。利用 PTR 运算符还可以建立一个新的存储器操作数，它与原来的同名操作数具有相同的段和偏移量，但可以有不同的类型。格式中的类型字段表示所赋予的新的类型属性，而表达式字段则是被取代类型的符号地址。

操作符格式：类型（type）PTR 表达式（expression）

（2）段操作符：用来表示一个标号、变量或地址表达式的段属性。段跨越前缀即是其中一种。

操作符格式：段寄存器名（或段名或组名）：地址表达式

（3）SHORT：用来修饰 JMP 指令中转向地址的属性，指出转向地址是在下一条指令地址的±127 个字节范围内。

（4）THIS：可以象 PTR 一样建立一个指定类型（BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE）的或指定距离（NEAR、FAR）的地址操作数。该操作数的段地址和偏移地址与下一个存储单元地址相同。

操作符格式：THIS 属性（attribute）或类型（type）

（5）HIGH 和 LOW——字节分离操作符：它接收一个数或地址表达式，HIGH 取其高位字节，LOW 取其低位字节。

（6）HIGHWORD 和 LOWWORD——字分离操作符：它接收一个数或地址表达式，HIGHWORD 取其高位字，LOWWORD 取其低位字。

（7）方括号”[]”操作符：经方括号”[]”括进去的内容表示存储器的地址。

6 表达式中的多个运算符的运算规则

（1）运算符的运算规则

①优先级高的先运算，优先级低的后运算。

②优先级相同时按表达式中从左到右的顺序运算。

③圆括号可提高运算符的优先级，圆括号内的运算总是在其任何相邻的运算之前进行。

（2）各种运算符的优先级如下表：

优先级		运算符
高	1	LENGTH，SIZE，WIDTH，MASK，()，[]，<>
	2	.（结构变量名后面的操作符）
	3	:（段跨越操作符）
	4	PTR，OFFSET，SEG，TYPE，THIS
	5	HIGH，LOW，HIGHWORD，LOWWORD
	6	+，–（一元运算符，即正、负）
	7	*，/，MOD，SHL，SHR
	8	+，–（二元运算符，即加、减）
	9	EQ，NE，LT，LE，GT，GE
	10	NOT
	11	AND
	12	OR，XOR
低	13	SHORT

（四）注释项（comment）

用来说明一段程序或一条或几条指令的功能，必须用分号“;”开始，它是可有可无的。注释应该写出本条（或本段）指令在程序中的功能和作用，而不应该只写指令的动作。

汇编语言源程序格式举例（1）

```
; 这里是程序标题（PROGRAMTITLEGOESHERE——）
; 后面有说明语句（FOLLOWEDBYDESCRIPTIVEPHRASES）
; 这里是说明语句（EQUSTATEMENTSGOHERE）
; ******************************************************
DATAREA    SEGMENT                          ; 定义数据段
; 这里是数据（DATAGOHERE）
DATAREA    ENDS
; ******************************************************
PROGNAM    SEGMENT                          ; 定义代码段
; ------------------------------------------------------------------------------
MAIN       PROC       FAR                   ; 主程序部分
ASSUME     CS:PROGNAM，DS:DATAREA
START:                                      ; 开始执行的地址
; 设置返回到 DOS（SETUPSTACKFORRETURN）。如省略则后面的 RET 改用 INT20H 等即可。
PUSH       DS                               ; 保存老的数据段地址
SUB        AX，AX                           ; AX 清 0
```

```
PUSH    AX                  ；此 3 条指令保护的是（DS）:0000，即 PSP+0 的地址
；设置 DS 指向当前数据段地址（SETDSREGISTERTOCURRENTDATASEGMENT）
MOV     AX，DATAREA         ；取数据段的段地址
MOV     DS，AX              ；保存到 DS 寄存器
；这里是程序的主要部分（MAINPARTOFPROGRAMGOESHERE）
RET                         ；返回到 DOS
MAIN    ENDP                ；主程序部分结束
；--------------------------------------------------------------------------------
SUB1    PROC    NEAR        ；定义子程序 1
；这里是子程序部分（SUBROUTINEGOESHERE）
SUB1    ENDP                ；子程序 1 结束
；--------------------------------------------------------------------------------
PROGNAM ENDS                ；代码段结束
END     START               ；源程序结束
```

汇编语言源程序格式举例（2）

```
.MODEL SMALL                ；DEFINEMEMORYMODEL
.STACK  100H                ；DEFINESTACKSEGMENT
.DATA                       ；DEFINEDATASEGMENT
；这里是数据（DATAGOESHERE）
.CODE                       ；DEFINECODESEGMENT
MAIN    PROC    FAR         ；主程序部分
START:                      ；开始执行的地址
MOV     AX，@DATA           ；取数据段的段地址
MOV     DS，AX              ；保存到 DS 寄存器
；这里是程序的主要部分（MAINPARTOFPROGRAMGOESHERE）
MOV     AX，4C00H
INT     21H                 ；返回到 DOS
MAIN    ENDP                ；主程序部分结束
END     START               ；源程序结束
```

四、汇编语言程序的上机过程

（一）建立汇编语言的工作环境

为运行汇编语言程序至少要在磁盘上建立以下文件：

1.EDIT.EXE　　全屏幕编辑程序，用于产生汇编语言源程序

2.MASM.EXE　　汇编程序

3.LINK.EXE　　连接程序

4.DEBUG.EXE　调试程序

（二）建立 ASM 文件

用全屏幕编辑程序建立汇编语言源程序（ex_movs.ASM）。

（三）用 MASM 程序产生 OBJ 文件

将源程序（ex_movs.ASM）汇编成机器语言目标程序（ex_movs.OBJ）。

1.操作命令

MASMex_movs;　　；最好在命令后加分号";"，这样不产生其它文件，速度快。

2.汇编程序还有一个重要功能：可以给出源程序中的错误信息。

（1）警告错误（WarningErrors）：指出汇编程序所认为的一般性错误。（不影响汇编结果。）

（2）严重错误（SevereErrors）：指出汇编程序认为已使汇编程序无法进行汇编的错误。并给出错误代号和信息。

（四）用 LINK 程序产生 EXE 文件

将目标程序（ex_movs.OBJ）连接成机器能执行的 ex_movs.EXE 文件。

LINKex_movs;　　；最好在命令后加分号";"这样只产生 EXE 文件，速度快。

（五）程序的执行和调试

1.在建立了 EXE 文件后，就可以直接从 DOS 执行程序，如下所示：

ex_movs↙

2.用 DEBUG 调试程序，如下所示：

DEBUGex_movs.exe↙

第四节 汇编语言程序设计

一、编制汇编语言程序

编制汇编程序的步骤如下：

（一）分析题意，确定算法。

（二）根据算法，画出程序框图。

（三）根据框图编写程序。

（四）上机调试程序。

二、程序有四种结构形式

程序的四种结构格式：顺序、分支、循环、子程序。

（一）顺序程序结构：完全按顺序逐条执行的指令序列

例如：编程将内存数据段字节单元 INDAT 存放的一个数 n（假设 0≤n≤9），以十进制形式在屏幕上显示出来。

DATA SEGMENT；定义数据段

INDAT DB 8；数据段内容

DATA ENDS；数据段定义结束

CODE SEGMENT；定义代码段

ASSUME CS：CODE，DS：DATA；指定段地址所在的段寄存器

START:；程序开始

MOV AX，DATA；数据段地址送 AX

MOV DS，AX；本指令执行后，DS 才有实际的数据段地址

MOV DL，INDAT；把 INDAT 中的数据送给 DL

ADD DL，30H；把数据转换为 ASCII 码

MOV DL，'D'；把要显示的数据的 ASCII 码送 DL

MOV AH，2；传递参数 2 号 DOS 功能调用，显示数据

INT 21H；调用中断

MOV AH，4CH；4C 号 DOS 功能调用

INT 21H；返回 DOS

CODE ENDS；代码段结束

END START；程序结束

（二）分支程序设计：分情况执行某段代码

例如：编写程序段，求 AX 中存放的带符号数的绝对值，结果存 RES 单元

DATA SEGMENT；定义数据段

NUM DW-8；数据段内容

RES DW?；数据段内容

DATA ENDS；数据段定义结束

CODE SEGMENT；定义代码段

ASSUME CS：CODE，DS：DATA；指定段地址所在的段寄存器

START:；程序开始

MOV AX，DATA；数据段地址送 AX

MOV DS，AX；本指令执行后，DS 才有实际的数据段地址

MOV AX，NUM；NUM 中的数据送 AX

CMP AX，0；与 0 进行比较判断

JGE ISPOSITIVE；大于等于 0 则转到 ISPOSITIVE

NEG AX；否则对 AX 求补

ISPOSITIVE:；程序标号

MOV RES，AX；把 AX 中数据送 RES

MOV AH，4CH；4C 号 DOS 功能调用

INT 21H；返回 DOS

CODE ENDS；代码段结束

END START；程序结束

（三）循环程序设计：重复执行某段代码

例如：数据段的 ARY 数组中存放有 10 个无符号数，试找出其中最大者送入 MAX 单元。

DATA SEGMENT；定义数据段

ARY DB17，5，40，0，67，12，34，78，32，10

MAX DB?；数据段内容

DATA ENDS；数据段定义结束

STACK1 SEGMENT PARA STACK；定义堆栈段

DW 20HDUP（0）；堆栈段内容

STACK1 ENDS；堆栈段定义结束

CODE SEGMENT；定义代码段

ASSUME CS：CODE，SS：STACK1，DS：DATA；指定段地址所在的段寄存器

BEGIN：MOV AX，DATA；程序开始，数据段地址送 AX

MOV DS，AX；本指令执行后，DS 才有实际的数据段地址

MOV SI，OFFSETARY；SI 指向 ARY 的第一个元素

MOV CX，9；CX 作循环次数计数

MOV AL，[SI]；取第一个元素到 AL

LOP：INCSI；SI 指向后一个元素

CMP AL，[SI]；比较两个数

JAE BIGER；前一个元素大于后一个元素时转移

MOV AL，[SI]；取较大数到 AL

BIGER：LOOP LOP；（CX）不等于 0 则转移

MOV MAX，AL；最大数送入 MAX 单元

MOV AH，4CH；4C 号 DOS 功能调用

INT 21H；返回 DOS

CODE ENDS；代码段结束

END BEGIN；程序结束

（四）子程序结构

子程序又称为过程，它相当于高级语言中的过程和函数。模块化程序设计是按照各部分程序所实现的不同功能把程序划分成多个模块，各个模块在明确各自的功能和相互的连接约定后，就可以分别编制和调试程序，再把它们连接起来，形成一个大程序。

1.子程序的设计方法

（1）过程定义伪操作：用在过程（子程序）的前后，使整个过程形成清晰的，具有特定功能的代码块。

伪操作格式：过程名（procedurename）PROC 属性（Attribute）

⋮

过程名（procedurename）ENDP

说明：

①其中过程名（procedurename）为标识符，它又是子程序入口的符号地址。

②属性（Attribute）是指类型属性，它可以是 NEAR 或 FAR。用户对过程属性的确定原则很简单：调用程序和过程在同一个代码段中则使用 NEAR 属性；不在同一个代码段中则使用 FAR 属性。主过程 MAIN 应定义为 FAR 属性。

2.子程序的调用和返回：8086 的 CALL 和 RET 指令完成的就是调用和返回功能。

3.保存和恢复寄存器：就是保护现场和恢复现场。在一进入子程序后，就应该把子程序所需要使用的寄存器内容保存在堆栈中；在退出子程序前把寄存器内容恢复原状。

4.子程序的参数传送：调用程序和子程序之间的信息传送称为参数传送（或称变量传递或过程通信）。

通过寄存器传送参数：这是最常用的使用最方便的一种方式，但参量很多时不能使用这种方法。

例如：将两个给定的二进制数（8 位和 16 位）转换为 ASCII 码字符串（通过寄存器传送参数）。

DATA SEGMENT；定义数据段

BIN1 DB 35H；数据段内容

BIN2 DW 0AB48H；数据段内容

ASCBUF DB 20H DUP（?）；数据段内容

DATA ENDS；数据段定义结束

STACK1 SEGMENT PARA STACK；定义堆栈段

DW 20H DUP（0）；堆栈段大小

STACK1 ENDS；堆栈段定义结束

CODE SEGMENT；定义代码段

ASSUME CS：CODE，DS：DATA，SS：STACK1；指定段地址所在的段寄存器

BEGIN：MOV AX，DATA；程序开始，数据段地址送 AX

MOV DS，AX；本指令执行后，DS 才有实际的数据段地址

XOR DX，DX；DX 清零

LEA DI，ASCBUF；存放 ASCII 码的单元首地址送 DI

MOV DH，BIN1；待转换的第一个数据送 DH

MOV AX，8；待转换的二进制数的位数送 AX

CALL BINASC；调用 BINASC 子程序

MOV DX，BIN2；待转换的第二个数据送 DH

MOV AX，16；待转换的二进制数的位数送 AX

LEA DI，ASCBUF；存放 ASCII 码的单元首地址送 DI

ADD DI，8；设置下一个数的存放首地址

CALL BINASC；调用 BINASC 子程序

MOV AH，4CH；4C 号 DOS 功能调用

INT 21H；返回 DOS

BINASC PROC；定义子程序 BINASC

MOV CX，AX；把 AX 中的数据送入 CX

LOP：ROL DX，1；最高位移入最低位

MOV AL，DL；把 DL 中的数据送入 AL

AND AL，1；保留最低位，屏蔽其他位

ADD AL，30H；把数据转为 ASCII 码

MOV [DI]，AL；存结果

INC DI；修改地址指针

LOOP LOP；跳转到 LOP

RET；子程序返回

BINASC ENDP；子程序 BINASC 结束

CODE ENDS；代码段结束

END BEGIN；程序结束