前　言

汇编语言是一门面向机器的语言，它能够利用计算机所有硬件特性并能直接控制硬件。在许多对程序运行速度要求的场合以及很多需要直接控制硬件的应用场合；汇编语言是不可少的工具，正因为如此，人们需要了解和掌握汇编语言的知识，并具有程序设计的能力。

“汇编语言程序设计”是大专计算机专业学生必修的一门重要的专业基础课。是从事计算机研究和应用，特别是软件研究的基础。它不仅是计算机原理、操作系统等核心课的必要先修课程，而且对训练学生掌握程序设计技术，熟练上机操作和程序调试都有很重要的作用。

本书根据大专计算机专业教学大纲编写。以 IBM－PC 机为目标机器，较全面介绍了汇编语言的基本概念，基本原理及程序设计的常用方法和技术。本书是在参考了许多有关教材的基础上，结合几年来的教学实践编写而成的。它不仅适用于高等院校大专计算机专业和本科非计算机专业，而且可作为中专学校计算机专业教材。同时也可供自学计算机的读者使用。

本书共分九章，第一章～第三章全面介绍了汇编语言程序设计必要的基础知识，包括 CPU 结构，寻址方式，指令系统和程序格式。

第四章重点介绍了汇编语言程序设计的常用方法与技术，包括分支，循环，子程序和模块化程序设计。

第五章讨论了输入/输出程序设计，包括输入/输出概念，中断及中断程序设计的概念，ROM－BIOS 中断调用方法等。

第六章阐述了高级宏汇编技术，包括结构，记录，宏汇编，重复及条件汇编知识。

第七章介绍了 80486/80586 常用指令，包括原指令加强的功能和新增的指令。

第八章叙述了上机实验的方法和内容。

在书中给出了第四章的习题及第五章的部分习题的参考答案，在附录中给出了汇编出错信息。

本书由李革新、陈建新、陈佛敏任主编并审校，丁焕峰、郭自龙、胡昌杰任副主编，胡红、吴进波任编委。在本书的编写过程中，兄弟学校的有关领导和老师给予了大力支持和帮助。在此向为本书编写给予支持和帮助的领导、专家、友人及参考文献的作者表示衷心的感谢！

由于作者水平有限，时间紧，不妥或错误之处在所难免，恳请读者批评指正！

编　者

一九九八年七月

目　录

第一章 绪 论

在计算机程序设计中，使用较多的是采用高级语言编写程序，但用高级语言编写的程序，机器不能直接执行，需要由编译程序或解释程序将它翻译成对应的机器语言程序，机器才能接受。这样的高级语言程序进行编译或解释生成的机器语言程序占用存储空间较大，执行起来速度较慢。汇编语言是唯一能够充分利用计算机硬件特性的一种面向机器的低级语言，它随机器的结构的不同而不同，它直接利用机器提供的指令系统编写程序，与机器语言程序一一对应，因此占用存储空间少，执行速度快。学习使用计算机语言编程，除使用高级语言编写管理类的软件外，可以使用汇编语言开发出更有效的计算机应用系统，尤其是实时控制系统，因此学习和掌握汇编语言是非常必要的。

第一节 汇编语言的基本概念

一、机器语言

计算机执行的每一个有意义的操作都是由使用计算机的人事先安排好的，就是用特定的语言编写程序。编写程序的工作称为程序设计，程序设计所用的语言称为程序设计语言，亦称计算机语言。由于程序是人们为计算机安排的工作步骤，所以程序必须使人和计算机都能够理解，因此，程序设计时就必须选择合适的语言，以做到既便于人编写程序，又便于计算机执行。对于计算机硬件来说，它能“懂”的唯一语言就是机器指令代码，所以指令代码被称为“机器语言”，最初的程序语言就是机器语言。

机器语言是为指挥计算机完成某一基本操作的命令，它由一组二进制代码组成，以指出计算机所要进行的操作及其操作对象，经指令译码器译码然后完成规定的操作，每一组代码构成一个机器指令。

机器指令的一般形式为：

操作码	地址码

操作码指出该指令的功能以及所要完成的操作，即运算的种类，如：加、减、传送、移位等最基本的操作。地址码指出参与运算的操作数和运算结果存放的位置，它们分别用二进制代码表示。一般机器指令的长度为单字节或多字节。每个指令具体占几个字节，系统对每条指令均有规定。

要想使计算机解决一个实际问题，必须要完成一系列有意义的操作，不同的操作对应不同的机器指令，因此要解决一个实际问题，就要编写出一系列机器指令的集合，交由计算机去执行，以完成所要求的操作。

机器指令是面向机器的，即每种型号的计算机都规定自己所特有的，一定数量的基本指令，这批指令的全体即为计算机的指令系统，这种机器指令的集合就是机器语言。

由于机器指令都是二进制的代码组，因此编写机器语言程序极其繁琐，很难记忆，且语言本身与人们的习惯相差甚远，容易写错，编出的机器语言程序也很难看懂，难于交流，所以实际使用很困难，因而使用受到限制。不同机种的机器指令系统不同，在某一机种上编写的机器语言程序，不能在另一机种上使用，必须按照所用机器的指令系统重新编写程序。

机器语言程序也有它的优点，这种程序可直接在计算机上运行，效率较高，占用主机时间少，但仅限于使用机器语言来编写程序的机器，现在只是在简单的计算机系统中或对计算机进行检测时才使用它。

二、汇编语言

在用机器语言编写程序的过程中，人们考虑到计算机能自动迅速地完成许多工作，那么编写程序的工作本身能否部分地交给计算机去做呢？这就是程序设计自动化的思想。

用机器语言编写程序的一个最大弱点，是用数码表示操作和地址。数码含义不直观，难记忆，写出的程序也难以阅读和调试。为克服这些缺点，人们就想出了用记忆符号来表示指令，称为汇编指令。例如：用助记符表示机器指令的操作码，用变量代替操作数的存放地址等。这样每条机器指令对应一条汇编指令，所有汇编指令的集合就构成了计算机的汇编指令系统，汇编指令又称符号指令，主要操作与机器指令一一对应的，这种用符号编写的，并遵循一定的语法规则的计算机语言，就是汇编语言，因此汇编语言也是面向机器的语言。

汇编语言由于采用一些助记符号来表示机器指令中的操作码，对用户来说虽然易写易读，但用它编写的源程序并不能直接被计算机识别，所以要让计算机执行汇编语言程序，还得配一个翻译程序——汇编程序。汇编程序将用汇编语言编写的源程序经过翻译转换成机器语言程序（亦称目标程序），目标程序中的二进制代码（即机器指令）称为目标代码。用汇编程序翻译源程序生成目标代码的过程称为汇编。由此可看出，汇编程序相当于一个翻译器，它加工的对象是汇编源程序，其所得结果为目标程序。

从语言的角度来说，汇编指令就是汇编语言的指令语句，伪指令（汇编控制指令）是汇编语言的指示语句。指令语句解决操作问题（操作码），指示语句就指挥汇编程序正确完成翻译工作，如源程序应从什么位置开始安放，汇编到什么位置结束，留多少存储单元作临时存储区，数据的类型是什么，数据应放在什么位置等。这样汇编语言的内容应由指令助记符，语句标号、数据变量、伪指令以及它们的使用规则等构成。

由于汇编语言指令基本上与机器语言对应，指令语句实现的都是一些简单的操作，如累加器内容的移位，数据在存储器与寄存器间的传送，寄存器内容的相加等，因而其程序的编写也是相当麻烦的。要完成复杂的操作，必须由程序设计人员把这些操作转为一系列汇编语言的指令语句。这种转换工作是有一定的困难和耗费精力的。为了提高编程效率，现代计算机系统一般都提供了宏汇编指令。所谓宏指令就是用一个名字来代替程序中重复出现的一组语句，在程序中凡是所需要的地方就使用宏指令名字及不同参数进行宏调用，从而使汇编程序清晰，简洁，有利于阅读修改与调试，同时也加速了程序的编写进程。

三、高级语言

汇编语言与机器语言相比，虽然有很大的进步，但仍然存在不少缺点，譬如：汇编语

言与机器语言一样，也是面向机器的，与自然语言以及数学公式差异仍很大，用它编写程序繁琐，且缺乏通用性。这就是到50代中期出现各种乐意为人们所接受的高级语言的原因。

高级语言也称算法语言，它是面向问题或过程的语言，即用高级语言编写的程序与所解决的问题及计算过程有关，一般与计算机的内部逻辑结构无关，这种语言也接近于自然语言和数学语言。因此，它更符合人们的思维，更易为人们所理解、学习。用高级语言编写程序，可使人们摆脱机器内部的逻辑结构，而集中精力于问题本身。高级语言的通用性强，用它编写的程序原则上可在各种型号的机器上运行，现在使用的大部分应用软件都是用高级语言编写的。用户用某种高级语言编写程序，只要所使用的机器上具有这种语言的编译程序就行。

目前广泛使用的高级语言都是“过程性”语言，用它编程序必须写出每一步如何进行的

全过程，程序设计者必须具体指出执行的每一个细节（例如：输入一个数给某一变量，进行某一公式的运算，进行条件判断，执行多少次循环等等）这要求程序设计人员考虑得十分周到，一处小的错误将导致程序运行失败。近年来已出现了比“过程性语言”更高一级的“描述性”语言。描述性语言比过程性语言更接近于人的思维，用它来编写程序，一般不需要告诉计算机“怎么做”，而只需要告诉计算机“做什么”，这就更加方便了编程，现在也称这种描述性语言为“非过程化语言”或称“第四代语言”。

四、三种语言的使用比较

高级语言有许多优点，但它还不能完全取代汇编语言和机器语言，这是因为汇编语言和机器语言还有自己独特的优点。

机器语言使用麻烦，难于掌握，编程困难且不好检查和修改，同时程序的通用性也差。但机器语言执行最快，最直接，而且占内存空间最少，因为它不用翻译。

高级语言易学易懂，使用方便，通用性强，便于推广与交流，且它有文件操作功能，可以将有关数据以文件方式存入外存储器中。但高级语言程序机器不能直接执行，需借助于编译（或解释）程序，将高级语言编写的源程序翻译成机器语言，我们把完成这一任务的翻译程序称为编译程序（或解释程序）。由它将高级语言程序翻译成用机器指令表示的目标程序（二进制代码模块），目标程序才能为计算机所执行。由于高级语言是面向过程的，主要考虑怎样容易编写程序，因此在编译生成目标代码时不可避免地出现所编译后的目标程序不精炼，过于冗长，执行速度慢等特点。

汇编语言是机器语言与高级语言之间较好的折衷，它的执行速度与机器语言接近，同时由于汇编程序要比编译程序短得多，所以占内存较少，高级语言并不能有效地产生机器语言程序，而汇编语言恰好能弥补高级语言的不足，它的指令语句与机器指令一一对应，编写的程序较为精练，使用效率较高。

目前，还有一些系统程序和应用程序仍需要用汇编语言来编写，实用中还有大量用汇编语言编写的系统程序（如各种编译程序，服务性程序、操作系统、数据库管理系统等）和应用程序（如实时控制和专用微机系统等）。几乎每个计算机系统都把汇编语言作为系统的基本配置，汇编程序成为系统软件的核心成份之一。汇编语言程序设计是从事软件研究的

基础，对于从事软件开发等应用的工作人员来讲，掌握汇编语言及其程序设计技术是非常重要的。三种语言各有所长，一般说来，高级语言适合于数值计算和数据处理以及人工智能等。机器语言，汇编语言则适合于那些对于时间和空间要求特别高的问题，如实时控制等。在一些大型程序系统中，也有把这三种语言结合使用的情形，即主程序用高级语言编写，而把一些对时间和空间要求较高的子程序用机器语言或汇编语言来编写，这样可充分发挥各种语言的长处，提高系统的质量和效率。

五、关于书中使用符号的说明

在本书后面的学习中，需要引用以下一些符号：

1. PA　表示某一存储单元的物理地址；
   EA　表示某一存储单元的有效地址（或称偏移地址）：指某一存储单元到它所在段的段首址的字节距离。
2. (…)　表示地址“…”中的内容。
   例如，有效地址 EA＝100 的存储单元的内容为 50H，则表示为（100）＝50H；
   寄存器 AX 的内容为 0FAF4H，则可表示为（AX）＝0FAF4H。
3. […] 表示以地址“…”中的内容为偏移地址。
   例如，若 (BX) = 0400H，而 (0400H) = 384，则 ([BX]) = 384。
4. DEST 表示目的地址，即目的操作数存放的偏移地址。
   SOUR 表示源地址，即源操作数存放的偏移地址。
5. → 表示传送。
   例如，(AX) → BX 表示把 AX 的内容传送到 BX 中。
6. ∧ 表示逻辑与
   ∨ 表示逻辑或
   ⊕ 表示逻辑异或
   —— 上划线表示逻辑非
7. _ 下划线表示从键盘输入
   / 表示或者
8. Flags 表示状态标志寄存器

第二节　8088/8086 CPU 的功能结构

汇编语言是面向机器的语言。汇编语言程序设计与计算机硬件有密切的关系，因此为了掌握汇编语言程序设计，特别是输入、输出和中断的程序设计，必须要较好地了解微处理器的结构、性能及有关部件的功能。如果要与外部设备进行数据传输，还要了解有关接口电路的功能。如果不了解计算机的硬件和内部结构，可以说是很难进行汇编语言程序设计的。

8088/8086 CPU 就功能而言可分成两大部分：总线接口单元 BIU（Bus Interface Unit）和执行单元 EU（Execution Unit），下面分别介绍这两个部件的基本结构与功能以及寄存器的结构。

一、执行单元EU与总线接口单元BIU

首先介绍8088/8086 CPU的内部结构和存储器结构，作为学习汇编语言程序设计的基础。

Intel 8088/8086是两种第三代微处理器。第一代微处理器是1971年推出的4044微处理器，仅有4位字长且仅有最基本的指令组，功能很有限。但它的成功，是计算机科学领域的一个重要的里程碑，开创了微型计算机这一计算技术的新领域。第二代微处理器是1973年研制成功的8位微处理器8080，它比第一代微处理器的功能强，可访问64K的地址，它的改进型为8085，它们虽比第一代微处理器功能强，但具体功能仍然有限。如：精度低，速度慢。Intel公司在1977年又推出16位微处理器产品8086，功能大大加强，其速度比8位机快得多。在汇编语言一级，它们与8080/8085是兼容的。它们有20条地址线，直接寻址能力为1MB。8086为16位的数据通道，因此它需要16位的存储器，16位的数据总线及16位的外部设备。但8位机使用的时间已相当长，很多价格合理的外设都是8位的形式，为解决这个矛盾，Intel公司推出了8088微处理器。它具有8位数据通道，可以与存储器或输入输出交换信息。8088与8086有相同的内部结构，相同的16位寄存器，区别仅在于数据线的根数不一样，由于8088只有8根数据线，故一次存取1个字节。8086有16根数据线，一次存取两个字节。对于8088来说，如果需要16位数据，则它必须分两次从连续的两个字节中取出数据将它们拼成一个16位的二进制数，从而实现对16位数据的操作。在其它方面，8086/8088都是相同的。为其中一个CPU编写的软件，可以不需修改的在另一个CPU上执行，早期IBM PC系列机及兼容机上广泛地采用了8088微机处理器。

8088微处理器的执行部件与总线接口部件其内部结构如图1.2.1所示。

1. 执行部件（EU）

EU用来完成指令的执行，并进行算术逻辑运算等。它主要由标志寄存器Flags、算术逻辑

单元ALU（Arithmatic Logical Unit）、通用寄存器和EU控制单元组成。在8086/8088中，由于取指令部分和执行指令部分是分开的，所以在一条指令的执行过程中，就可以取出一条指令或多条指令，放在指令流队列中排队，当一条指令执行完以后，就可以立即执行下一条指令，减少了CPU为取指令而等待的时间，提高了CPU的利用率，加快了系统的运行速度。另一方面又降低了与之配合的存储器的存取速度的要求。

EU主要实现两种操作，一是根据指令进行算术/逻辑运算，二是由EU计算出指令要求寻址单元地址的偏移量，以形成一个20位的物理地址，去存储器存取所要求的操作数。

2. 总线接口部件（BIU）

BIU负责8086/8088 CPU与存储器和外设之间的信息传送。BIU用来实现EU的所有总线操作。它由地址加法器、段寄存器（CS，DS，ES，SS）、指令指针IP、指令队列和总线控制逻辑组成。在EU执行指令的过程中，BIU负责从内存指定部分取出指令送至指令流排队机构中排队。在执行指令时，所需要的操作数也是由BIU从内存的指定区域取出传送给EU部分去执行。

8088的指令队列中最多可同时存放指令的长度为4个字节，当指令队列空的时候，BIU自动执行总线操作，取指令存入指令队列。

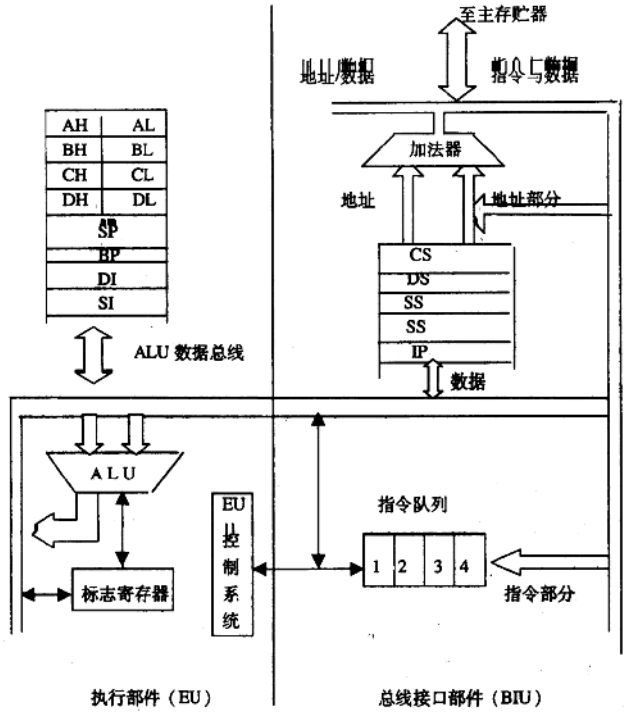

图 1.2.1　8088 CPU 结构

在BIU 中，有一个指示器 IP，它总是保存着下一次将要从主存中取出的指令的偏移地址，这一偏移地址的值为该指令到所在段段首址的字节距离。BIU 根据 IP 的内容和段寄存器 CS 的内容形成指令的物理地址（20 位），并根据该地址从主存中取出指令，送入指令队列机构中排队，然后 IP 增量，形成下一字节的偏移地址。如果碰到要改变 IP 的指令（如转移指令），原指令队列中的指令已不能使用，这时将按 IP 的新内容与CS 段寄存器的内容形成指令的物理地址，再从主存中取出指令，立即送 EU 执行，然后再从该地址开始，继续重新取指令填满指令队列。

指令指针 IP 和段寄存器 CS 或将 EU 送来的偏移量与段寄存器 DS 经地址加法器形成一个 20 位的物理地址，从存储器中取出指令或数据。

为提高 CPU 的运行速度，8086/8088 系统设计为并行工作方式，即指令与数据的存取电路和指令的执行电路并行工作。其执行过程如图 1.2.2 所示：

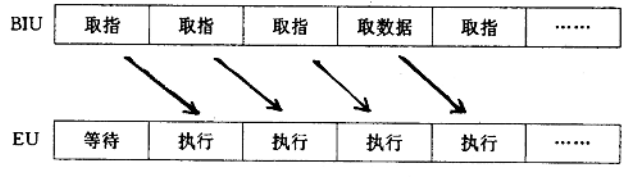

图 1.2.2　BIU 与 EU 并行工作情形

假设程序的指令代码预先已存放在存储器中，为了执行程序，CPU 依据时钟节拍，产生一系列的控制信号，有规则地重复执行取指、执行……操作的过程。在执行指令的过程中，利用 EU 分析操作码和执行指令不占用总线时间这一特点，由 BIU 自动地通过总线取存储器指令到指令队列中。

由于指令的执行单元 EU 与总线接口单元 BIU 的操作是独立进行的，因此可以并行工作。在 EU 执行指令的过程中，BIU 就可以取出指令存放在指令队列中。当 EU 执行完一条指令就可以立刻到指令队列中去取下一条要执行的指令继续执行。这样减少了 CPU 为从内存取指令而等待的时间，提高了 CPU 的利用率，也提高了整个系统的运行速度。

二、8085/8088 CPU 寄存器的结构

1. 通用寄存器

1）数据寄存器

AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX

2）指针/变址寄存器

SP	堆栈指针寄存器
BP	基址指针寄存器
SI	源变址寄存器
DI	目的变址寄存器

2. 专用寄存器

1）控制寄存器

IP		指令指针寄存器
FLAGH	FLAGL	状态标志寄存器

2）段寄存器

CS	代码段寄存器
DS	数据段寄存器
SS	堆栈段寄存器
ES	附加段寄存器

8086/8088 共有 14 个寄存器供系统使用，在汇编语言程序设计中要经常用到这些寄存器，因此，对它们的了解也是十分必要的。

（1）通用寄存器组

通用寄存器包括 AX、BX、DX、SI、DI、SP、BP 等 8 个寄存器。8086/8088 也能处理 8 位数，上图中 4 个 16 位数据寄存器也可以作为 8 个 8 位寄存器使用。

8 个通用寄存器的通常用途见表 1-1。

对于 4 个通用寄存器可任意参加算术运算或逻辑运算，它们有自己专用的名称，对有些指令有特定的功能。AX（Accumulater）在指令中使用最多，被称为累加器。BX（Base）称为基址寄存器，CX（Count）称为计数寄存器，DX（Data）称为数据寄存器，源变址（Source Index）寄存器 SI，和目的变址（Destination Index）寄存器 DI。它们不但具有一般通用寄存器的特性，还可以在串操作指令中作专用寄存器使用。进行数据串操作时，SI 用来存放源数 据串的首地址，DI 用来存放目的数据串的首地址，进行数据串操作时，

CPU 会自动地使用 SI 和 DI 变址寄存器。

表 1-1　8 个通用寄存器的通常用途

寄存器	通 常 用 途	寄存器	通 常 用 途
AX	字乘法，字除法，字 I/O	CL	变量移位或循环
AL	字节乘、除法、I/O、转	DX	字乘法、字除法、及间接 I/O
	换、十进制算术运算	BP	基地址堆栈操作
AH	字节乘法，字节除法	SP	堆栈操作
BX	转移	SI	数据串操作（源址）
CX	串操作，循环次数	DI	数据串操作（目的址）

在计算机中，通常在内存中开辟一个专用的数据存储区，这片存储区采用的存储方式是一端固定，另一端活动，即只允许在一端插入或删除数据，我们称这一片存储区为堆栈。它具有“后进先出”的存储特性，主要用来在子程序调用和中断操作时保护程序的现场或断点。

从硬件的观点看，堆栈由一片存储单元与一个指示器组成，固定端称为栈底，栈指针用来指示栈元素进栈和出栈时偏移地址的变化，指针所指示的最后存入数据的单元叫栈顶，所有信息的存取都在栈顶进行，因而栈指针 SP 总是指向栈顶的。在堆栈指令操作时，由 SP 给出入栈或出栈的数据在栈中的地址，但要确定堆栈的物理地址，SP 必须与 SS（堆栈段寄存器）相结合（因为 SP 的内容为栈顶相对于 SS 的偏移地址）。空栈时，SP 指向堆栈段的栈底（即最高地址），存入时栈顶由高地址向低地址变化。

BP（Base Pointer）是基址指针，它也可以用来对堆栈中的数据进行操作，使用 BP 可以对堆栈中任意位置的数据进行操作，但 BP 它不具有 SP 始终指向栈顶的含义，与 SP 类似，它必须与 SS 相结合才能确定数据在堆栈中的实际物理地址。

（2）指令指针寄存器与标志寄存器

指令指针寄存器 IP（Instruction Pointer）它是用来存放将要取出指令的偏移地址的，但要形成指向指令的真正的物理地址，必须与 CS（代码段寄存器）相结合。

Flags 是标志寄存器，它反映系统的状态及运算结果的特点，占 2 个字节，共 9 个标志位。

Flags 寄存器的结构如下：

D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
				O	D	I	T	S	Z		A		P		C

下面分别叙述各标志位的作用：

辅助进位标志 AF（Auxitiary Carry Flag）：在字节操作时若低半字节（一个字节的低 4 位）向高半字节有进位或借位；字操作时，低位字节向高位字节有进位或借位，则 AF＝1，否则 AF＝0。这个标志用于十进制算术运算指令中，即通过二进制数运算调整为十进制数表示的结果。

进位标志 CF (Carry Flag)：当结果的最高位（字节操作时的 D7 或字操作时的 D15）产生一个进位或借位，CF＝1，否则 CF＝0。这个标志主要用于多字节数的加、减法运算。移位或循环移位指令，也能够把存储器或寄存器中的最高位（左移时）或最低位（右移时）放入标志 CF 中。程序中也可以根据 CF 标志位状态决定程序是否转移。

溢出标志 OF (Overflow Flag)：有符号数运算时，当其运算结果超出了 8 位或 16 位有符号数所能表达的范围时，(8 位大于 127 小于－128，16 位大于 32767 小于－32768 时）将产生溢出，置 OF 为 1，否则置 OF＝0。在溢出情况下能产生中断。溢出与进位是两个不同性质的标志，千万不能混淆。例如在字节运算时

```
MOV     AL,     64H
ADD     AL,     64H
```

即 01100100＋01100100＝11001000

D7 位向前无进位，故运算后 CF＝0，但运算结果超过了＋127，此时溢出标志 OF＝1。

符号标志 SF (Sign Flag)：它的值与运算结果的最高位相同。即结果的最高位（字节操作为 D7，字操作时为 D15）为 1，SF＝1，否则 SF＝0。

本标志主要用于表示有符号数运算结果的正负，当 SF＝0 时，表示结果为正数，当 SF＝1 时，表示结果为负数。

奇偶标志 PF (Parity Flag)：当执行结果使 1 的个数为偶数时，置 PF 为 1，否则表明 1 的个数为奇数，PF 为 0，此标志主要用于逻辑运算中。在数据的传输过程中可通过检查本标志位，判定是否产生数据传输错误。

零标志 ZF (Zero Flag)：当运算结果为零时，置 ZF 为 1，否则置 ZF 为 0。本标志常用于分支程序或循环程序的转移控制中，可使程序根据运算结果是否为 0 决定程序的流向。

上述 6 个标志为状态标志位。

8086/8088 还提供了三个控制标志，它们能由程序来置位或复位，以变更处理器的操作。

方向标志 DF (Direction Flag)：在字符串操作指令中，字符串操作可为自动增址或减址（即由低地址向高地址方向变化或从高地址向低地址方向变化）当 DF＝1，串操作为自动减址，当 DF＝0，串操作为自动增址。

中断允许标志 IF (Interrupt－enable Flag)：若指令中置 IF 为 1，则允许 CPU 去接收外部的可屏蔽中断请求，此时 CPU 为开中断。若使 IF 为 0，则屏蔽外部中断请求，此时 CPU 为关中断。对内部产生的中断不起作用。

追踪标志 TF (Trap Flag)：置 TF 为 1，使处理进入单步方式，以便于调试，在这个方式中，CPU 在每条指令执行以后产生一个内部中断，允许程序在每条指令执行以后进行检查。当 TF＝0 时，CPU 正常执行操作。

3. 段寄存器

8086/8088 有四个段寄存器，分别称为代码段 CS (Code Segment)，数据段 DS (Data Segment)，堆栈段 SS (Stack Segment) 和附加段 ES (Extra Segment) 寄存器。四个段寄存器都在 BIU 中，使用四个段寄存器可以存放各段的基本地址。其中：CS 寄存器指示当前代码段，它规定了现行程序所在的存储区的首址，CPU 执行的指令是从 CS 段中取得的。ES 寄存器指示当前堆栈段，堆栈所进行的操作将在这个区段中。DS 寄存器指示当前数据段，

ES 指示当前附加数据段，即 DS 和 ES 规定了现行程序中所用数据的存储区的首址，程序中的变量和数据存放在这个段中。

第三节　存储器与物理地址的形成。

一、存储器

存储器是用来存放程序和数据的，它能准确地记录指定的信息并保存以供用户选用，对不需要的信息，可用新的内容来取代或将其抹去。它的基本存储单位是位，能容纳二进制信息 0 或 1。整个存储器由许多存储位构成，这些存储位每 8 位组合成一个字节，每相邻的两个字节又可组成一个字。为了区别不同的存储单元，系统为每个单元都指定一个编号，称为此单元的物理地址。存放在存储器中的程序和数据一律按物理地址存取。

8086/8088 有 20 条地址线，它的直接寻址能力为 1Mb（2 的 20 次方），所以在一个系统中，可以有多达 1Mb 的存储器，地址从 00000H 到 FFFFFH，CPU 与存储器交换信息必须使用这种 20 位的物理地址。任意给定一个 20 位的地址，就可以从这 1Mb 范围中找到相应的单元，取出所需要的指令和操作数。

二、存储器物理地址的形成

8086/8088 CPU 内部的 ALU 只能进行 16 位运算，与地址有关的寄存器如 SP，IP 以及 BP、SI、DI 等也是 16 位的，因而对地址的运算也是 16 位的，也就是说，对于 8086/8088 来说，各种寻址方式寻找操作数的范围最多可以是 64kb。为了能表示 20 位的物理地址，将 1Mb 字节的存储器按 64KB 分段，有四个段寄存器 CS，DS，SS，ES 保存当前可使用段的段首址，即有了段寄存器，就有了逻辑段的开始地址，此时相对于段首址而言的指令或数据的地址就是段内的偏移量。这时，指令或数据真正的物理地址应为段寄存器的内容加上偏移量。如果使各段的段首址都从能被 16 整除的地址开始，这些段首址的低 4 位都是 0，若忽略这些 0，则段首址的高 16 位正好装入一个段寄存器中。作存取访问，寻址一个具体的物理单元（存储器）时，CPU 根据操作要求，它必须由一个基本地址再加上由 SP 或 IP 或 BP 或 SI 或 DI 等可由 CPU 处理的 16 位偏移量来形成 20 位的物理地址。这个基本地址就是由 8086/8088 中的寄存器 CS，SS，DS，ES 中的一个来形成的。当选择某一段寄存器，将其中的内容左移 4 位，即在最后补入 4 个 0，再与该段中待访问的存储单元的偏移地址相加，即得到该单元的 20 位物理地址。

当要取指令时，则由 IP 所决定的 16 位偏移量与自动选择的代码段寄存器 CS 内容相加形成取指令的 20 位物理地址。要取指令执行时，它的物理地址 PA 应为：

PA＝（CS）左移 4 位＋（IP）。

当涉及一个堆栈操作时，则自动选择堆栈段寄存器 SS，再加上由 SP 所决定的 16 位偏移量，计算得到堆栈操作所需要的 20 位物理地址。用户一旦定义好了堆栈段，系统则自动以 SP 为指针，并设置好指针的位置，往堆栈中压入数据和从栈中弹出数据时，只能使用 PUSH 和 POP 命令，这时栈顶的物理地址为：

PA＝（SS）左移 4 位＋（SP）

当其它指令要访问堆栈中某一存储单元时，必须通过基地址寄存器BP进行，即将该存储单元的偏移地址置入BP中，这时它的物理地址应为：

PA＝（SS）左移4位＋（BP）

PC机允许用户建立自己的字堆栈，最大可达64KB，其存储区由堆栈段寄存器SS给定，并固定采用作指针，即SP的内容为栈顶相对于SS的偏移地址。空栈时，SP指向堆栈段的最高地址即栈底。存入时，栈顶均由高地址向低地址变化。

当要存取操作数时，则自动选择数据段寄存器DS或附加段寄存器ES，再加上一个16位偏移量，计算得到20位的物理地址。所给出的16位偏移量，可以是包含在指令中的直接地址，也可以是某一个16位地址寄存器的值，也可以是指令中的偏移量加上16位地址寄存器中的值等等，主要取决于指令的寻址方式。当访问数据段中某一变量时，该变量的物理地址为：

PA＝（DS或ES）左移4位＋该变量的偏移地址

这种对存储器分段的方法，在不改变段寄存器值的情况下，寻址的最大范围是64KB。现有一个任务，若其程序长度、堆栈长度，以及数据区的长度都不超过64KB的话，则可在程序开始时，分别给DS，SS，CS置值，然后在程序中可以不再考虑这些寄存器，程序就可以在各自的区域中正常地进行工作。若对于一个程序中要用的数据区超过64KB，或要求从两个或多个不同的区域中存取操作数时，只要在取操作数以前，用指令给数据段寄存器重新赋值就可以了。若程序的总长（包括数据段和堆栈段）不超过64KB时，只要使CS，DS，SS相同就可以在64KB区域中工作了。这种将存储器分段的方法，对于要求在程序区、堆栈区、数据区之间隔离是非常方便的。尽管CPU在某一时刻最多只能访问4个段，但它并不限制程序中只能定义4个段，如果CPU需要访问4个段以外的存储区，只要改变相应段寄存器的内容即可。这种分段方法也适用于程序的再定位要求，在不少情况下要求同一个程序能在内存的不同区域中运行而不必修改程序中的任何代码，只要使程序中的转移指令都为相对转移指令，而在运行该程序前设法改变各个段寄存器的值即可。

第四节　计算机中数据的表示

计算机只能处理由0和1组成的信息，我们平时处理的十进制数，在计算机中是用二进制来表示的。把一个数连同其符号在机器中的表示加以数值化，这样的数称为机器数。一般用最高有效位表示符号位（正数用0表示，负数用1表示）。机器数可以用不同的码制来表示，常用的有原码、反码和补码，IBM－PC机的整数用补码表示。下面仅介绍数的补码表示法。

一、数的补码表示

对于有符号的数一律采用n位二进制补码表示，n可以是16位，也可以是8位。采用补码，可以在运算中把负数化为正数形式使得减法运算化为加法运算。在补码表示中，正整数采用符号——绝对值表示法，即数的最高有效位为0，表示符号为正，数的最高有效位为1，则表示符号为负，数的其余部分则表示为数的绝对值。

假设机器字长为8位时，［＋1］补＝00000001