高等院校计算机专业系列教材

高永慧　王　彬 编著

宏汇编语言程序设计

自学与提高的捷径

科学出版社

宏汇编语言程序设计

高等院校计算机专业系列教材

宏汇编语言程序设计

自学与提高的捷径

高永慧　王　彬　　编著

科学出版社

1994

（京）新登字 092 号

内容简介

本书是“高等院校计算机专业系列教材”中的一种，用作计算机专业及有关专业的“汇编语言程序设计”课程的教材. 本书主要介绍 Intel 8086/8088，80X86 汇编语言程序设计的方法和技术. 全书共分八章. 第一章叙述与编程密切相关的 8086 的结构；第二章给出几个简单的实例，以帮助读者尽快了解汇编语言；第三章介绍 8086 的指令系统；第四章叙述分支、循环、子程序的基本结构以及程序设计的基本方法和技术；第五章介绍宏和条件汇编技术；第六章总结性地说明伪指令的使用；第七章简单介绍输入/输出和中断；第八章是汇编语言在磁盘文件操作、发声、绘图等方面的应用以及 MASM5.0 与 MS-C 语言的接口. 另外，在附录中列出了 MASM 和 DEBUG 的使用以及用 80286 和 80386 编程的有关资料.

本书适宜作高等院校教材，并适于初学者使用. 熟习高级语言程序设计的读者，可以通过学习本书迅速掌握汇编语言程序设计技术. 另外，本书也可以供使用汇编语言技术的工程技术人员参考.

高等院校计算机专业系列教材

宏汇编语言程序设计

自学与提高的捷径

高力毅　王　彬　编著

责任编辑　王春晖

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100717

北京黄坎印刷厂　印刷

新华书店北京发行所发行　各地新华书店经售

*

1994 年 9 月　第 一 版　开本：787×1092　1/16

1994 年 9 月第一次印刷　印张：19¼

印数：1—3 200　字数：439 000

ISBN7-03-004271-9/TP・388

定价：22.80 元

前　言

当代新技术革命的蓬勃发展，带来社会生产力新的飞跃，引起整个社会的巨大变革，电子计算机技术是新技术革命中最活跃的核心技术，在工农业生产、流通领域、国防建设和科学研究方面得到越来越广泛的应用.

党的十一届三中全会以来，我国计算机应用事业的发展相当迅速. 到目前为止，全国装机量已突破数十万台，16 位和 32 位以下微型计算机已经形成产业和市场规模，举国上下在计算机科研、开发、生产、应用、经营等方面取得了优异的成绩，创造了显著的经济效益和社会效益，并在商业、城建、金融、科技、文教、卫生、公安等广阔的领域中积极开发应用计算机技术，为开拓计算机应用的新局面作出了重要贡献. 实践证明，人才是计算机开发应用的中心环节，我们必须把计算机专业人才的开发与培养放在计算机应用事业的首位，要坚持不懈地抓住人才培养这个关键.

北京工业大学计算机学院（原北京计算机学院）建院 15 年来，拥有一批以计算机专家、教授为骨干的教学、科研师资队伍，培养了数千名计算机专业技术人才，在多年的教学科研工作中，积累了许多经验. 承蒙科学出版社的大力支持，教师们决心将多年总结的教学经验编写出来，出版一套计算机专业的系列教材. 本套教材在内容上注重科学性，工程性和实用性，具有简明清晰，通俗易懂，方便教学，易于自学等特点. 可供大专院校的师生学习计算机专业知识使用，或作为计算机专业的技术人员的参考资料，也可以提供给有志于学习和使用计算机的人员入门与提高使用. 出版本套教材是人才培养和开发方面的一件很有意义的工作.

这套系列教材包括《计算机导论》、《计算机组成原理》、《数字系统逻辑设计》、《微型计算机接口技术》、《数据库技术与应用》、《宏汇编语言程序设计》、《计算机操作系统原理》、《编译原理》、《数据库技术与应用》和《数据安全与软件加密》等 10 本. 自 1994 年 9 月开始上述各教材将陆续出版.

汇编语言是一种自身靠近硬件的程序设计语言，因此是一种比较难学，尤为难以自学的语言. 当用户要编写运行速度快、占用空间少、特别是直接控制硬件的程序时，却离不开汇编语言. 目前，使用汇编语言进行程序设计的需求日益增加，计算机专业的学生必须掌握这一技术. 根据这一需求我们编写了这本教材.

本书具有以下特点：

1）叙述简明，由浅入深，通俗易懂，适于自学.

本书主要是为初学汇编语言者编写的，因而全书注意按语言的规律，由浅入深地介绍汇编语言. 为使读者能够边阅读边上机实习，从而逐步掌握有关内容，我们在保持全书的连贯性、论述体系的一致性和科学性的前提下，对讲述的内容进行了合理组织，力求逐步增加内容，引申概念.

汇编语言程序设计是一门实践性极强的课程，所以书中尽量避免用大篇的文字来论述

概念，而是采用大量的图解和例题，并加以注释．读者通过例题可以一目了然地理解和掌握汇编语言程序设计的方法．本书中所有例题均在 IBM-PC/AT 机上用 MASM5．0 汇编、用 LINK 连接，并运行通过．读者在学习中可以在机器上键入这些例题，在 DEBUG 下或屏幕上看运行结果．

2）始终贯穿结构化程序设计方法．

由于汇编语言本身的特点使得写出的程序难读、难懂、容易出错，因此，在编写这本教材的过程中，我们特别注意用结构化程序设计的思想和方法，引导读者学习汇编语言程序设计的基本方法，尽量使读者养成良好的结构化程序设计风格．

3）与高级语言对比介绍．

鉴于读者大都熟悉高级语言程序设计，因此，每当引入新概念时，常从高级语言的相应概念出发，使读者借助于已有的知识理解汇编语言，这样可以取得事半功倍的效果．

4）实用性强．

MASM5．0 版增加了一些新的功能，如段的简化定义、增强的出错处理、新的汇编任选项等．所有这些在本书中都有所体现．充分利用宏功能是减少编程错误和提高程序可读性的可行途径，本书较深入地介绍了宏和条件汇编．另外，对汇编语言与 C 语言的接口、程序段前缀的利用等很实用的内容也做了详细的讲解．因此，本书也可作为计算机软件开发的科技人员的提高用书．

关于本书的内容，有三点需要向读者说明：第一，本书没有涉及作为汇编语言基础知识的进位计数制及不同基数的数之间的转换、数和字符在计算机内的表示等．第二，本书只在第七章简要地介绍输入输出和中断理论．这是因为前者一般在《计算机导论》或《计算机组成原理》等课程中讲解，而后者在《微型计算机接口技术》课程中做系统的介绍．第三，由于汇编语言的不可移植性（同一系列机上可向上兼容），我们总得选择一种样板机，于是我们选择了目前国内用户最多的 Intel 80X86 系列．在该系列中，我们介绍 8086/8088 的宏汇编．实际上 80286 和 80386 等在实模式下的宏汇编也就是 8086 的宏汇编．附录 F 中给出了 80286 和 80386 等扩充的指令．8086 和 8088 的编程是没有区别的，它们的区别在于：当存取（或 I/O）一个字数据时，8088 是二次完成的，而 8086 是一次完成的．因此，本书中我们只介绍 8086 的宏汇编．

为了方便读者，本书所有例题都已录制成盘，盘中并有与本书配套的习题集，欢迎复制使用．

为了使有一定汇编语言程序设计基础的读者，在实际编程中遇到问题时能够方便地查阅资料，本书把一些在编程工作中经常需要查阅的内容归纳在附录中．

本书根据高永慧多年教授“汇编语言程序设计”课程的讲稿整理而成．初稿由王彬完成，后经高永慧反复修改成篇．本院软件工程系王燕兴审阅了全书，院计算中心李萌帮助我们将部分附录输入机内．对于他们付出的辛勤劳动，我们表示诚挚的谢意．

由于我们水平有限，书中难免存在缺点和错误，恳请读者批评、指正．

编著者

1993．11

目　录

第一章　8086 微处理器的结构

1.1　引言

汇编语言是一种面向计算机的程序设计语言，是机器语言的符号表示，是较低级的语言．可执行的汇编语句与机器的指令码具有一一对应的关系，用汇编语言编写的程序称为汇编语言源程序或汇编语言程序．汇编语言与机器硬件结构密切相关，它通常为特定的计算机系统设计．因此，汇编语言程序设计者要对相应计算机的组织结构有较多的了解，如：存储器分段操作、数据的大小与类型、寄存器的操作，各种设备的端口地址及其操作方式等．所以，编写汇编语言源程序比编高级语言源程序要困难得多．为了对汇编语言有个初步认识，让我们先来看下面的实现 3 个数相加（23H＋0AH＋10H）的源程序．它由栈段和代码段两段组成：

```
; 定义名为 STACK 的栈段
STACK   SEGMENT   STACK                ; 栈段开始
    DB  16  DUP ('MYSTACK  ')          ; 定义存储空间为 80H 个字节的栈
STACK   ENDS                           ; 栈段结束
; 定义名为 MYCODE 的代码段
MYCODE  SEGMENT                        ; 代码段开始
  ASSUME  CS:MYCODE,SS:STACK           ; 指明 MYCODE 为现行代码段，STACK 为现
                                       ; 行栈段
  MYPROC  PROC  FAR                    ; 定义名为 MYPROC 的远子程序
          PUSH  DS                     ; DS 的内容进栈
          SUB   AX,  AX                ; 0→AX
          PUSH  AX                     ; AX 内容进栈
          MOV   AL,  23H               ; 23H→AL
          ADD   AL,  0AH               ; 23H＋0AH→AL
          ADD   AL,  10H               ; 23H＋0AH＋10H→AL
          RET                          ; 返回
  MYPROC  ENDP                         ; 子程序结束
MYCODE  ENDS                           ; 代码段结束
    END   MYPROC                       ; 源程序结束，其中 MYPROC 指明程序运行
                                       ; 时的启动位置
```

上述源程序的子程序 MYPROC 体中有 7 条指令，其中

```
PUSH  DS
SUB   AX,  AX
PUSH  AX
```

是为使该程序运行后能返回操作系统做的准备工作．下面的 3 条指令：

```
MOV  AL，23H
ADD  AL，0AH
ADD  AL，10H
```

完成23H＋0AH＋10H，结果在累加器AL中．这个过程就像用算盘求若干数之和一样，这里AL是我们的算盘，计算过程是，先把数23H打在算盘上，然后将0AH加上去，最后再把10H加上去，最终结果留在算盘上．最后一条指令RET实现子程序返回．

从上述程序可以看出，用低级语言编写程序，是很繁琐的，它不能像高级语言编程序那样只指明“做什么”，而必须详细描述“如何做”．这就是用汇编这样的低级语言编程的困难所在．汇编语言是介于计算机能够理解的代码语言（即机器语言）与使用者容易理解的高级语言之间的一种语言．它以操作符代替二进制码帮助使用者记忆和使用代码语言，所以又称为符号语言．用汇编语言编出的程序与高级语言源程序比较，难读，容易出错，而且因为每种系列机的低级语言均不相同，所以汇编语言程序在不同的系列机中不能移植，在同一系列机中也只能向上兼容．虽然汇编语言源程序有上述缺点，但它也有高级语言程序所不具备的诸多长处．由于汇编语言比高级语言更接近于机器语言，因而能透彻地反映计算机硬件的功能与特点，便于使用者灵活地根据自己的要求编制最为经济（省时间，省内存）的程序，而且汇编语言能直接控制计算机硬件（如显示器、打印机、存储器、磁盘等）的操作，以及可直接与操作系统进行接口，等等，因此，汇编语言能编出高级语言无法实现的程序．所以，用汇编语言编写程序是程序员的一项基本功，必须很好地掌握．

学习汇编语言较为困难，那么，我们应该如何学习它呢？实际上，汇编语言和计算机高级语言甚至于自然语言一样，都是“语言”，只不过用计算机语言写出来的“文章”（源程序）用于人和计算机交流信息罢了．所以，学汇编语言时，只要抓住语言的特点，就不会觉得困难了．具体地说，首先要掌握组成语言的基本符号（字符集），然后认识这种语言里的词（如操作符，操作数……），接着要知道如何用这些词组织成各种语句（指令），并且在明确语句的语法的同时熟习它的语义，最后要通晓整个文章的结构，并学会用一个个的语句‘描述’向计算机传递的信息，即写出一篇文章．当然，要写出“好”的文章，事先要经过严密的构思，也就是要贯彻结构程序设计方法．只要读者掌握了这种学习规律，所谓的困难，就迎刃而解了．

与任何一种高级语言源程序一样，汇编语言源程序也是硬件不认识的，必须把它翻译成在逻辑上等价的、叫做目标程序的机器语言程序，才能在硬件上运行．完成这一翻译功能的是汇编程序．如前面的源程序，经汇编后所得汇编列表（左边是翻译结果：用十六进制表示的机器指令代码的存放位置及指令代码，右边是对应的汇编源程序）是

```
0000                              STACK  SEGMENT  STACK
0000  10[                           DB  16  DUP ('MYSTACK  ')
        4D  59  53  54
        41  43  4B  20
                    ]
0080                              STACK  ENDS
0000                              MYCODE  SEGMENT
0000                                ASSUME  CS：MYCODE，SS：STACK
```

```
0000                    MYPROC  PROC  FAR
0000  1E                PUSH  DS
0001  2B  C0            SUB  AX, AX
0003  50                PUSH   AX
0004  B0  23            MOV  AL, 23H
0006  04  0A            ADD  AL, 0AH
0008  04  10            ADD  AL, 10H
000A  CB                RET
000B                    MYPROC  ENDP
000B                    MYCODE  ENDS
                            END  MYPROC
```

从上述汇编列表中可以看到，汇编程序主要做了两件事：一是代真，即将符号指令翻译成了机器代码指令；二是存储分配，即给每个数据或指令分配了一个存储位置.

汇编程序是个“翻译”，可以从不同角度对其分类．一种分法是将它们分成基本汇编程序和宏汇编程序．前者把一条汇编指令翻译成一条机器指令；而后者允许程序员把一个指令序列定义成一条宏指令，在源程序中写宏调用．宏汇编程序处理宏调用指令时，将宏体嵌在调用处，即一条宏调用指令被翻译成若干条机器指令．也可以从汇编程序是否驻留在目标程序执行的机器上将它分成驻留汇编程序和交叉汇编程序两种，前者是指汇编程序赖以运行的机器就是执行目标程序的机器；而后者是指汇编程序所生成的目标程序在另一型号的机器上运行.

1.2 存储器和 I/O 的结构

编写汇编语言程序必须对计算机的硬件结构有所了解，所谓计算机的结构是指组成它的功能部件及它们之间的配合和联系，即：有多少寄存器，它们的功能如何；有多少存储器，如何寻址；能使用哪些 I/O 设备．本章简要介绍上述内容.

1.2.1 存储器结构

存储器用来存放数据和指令代码，它就像一个大饭店，有许多房间，每个房间有一个表示其“地址”的号码，数据和指令是住在房间里的客人.

8086 的存储单元个数为 $2^{20}=1048576=1M$．地址编码从 00000H～0FFFFFH，即地址码用 20 位二进制数表示.

存储单元长度为 8bit，即以字节为单位寻址，每个字节有一个地址码．也允许以字为单位编址，一个字由 2 个连续的字节组成，于是一个字有 2 个地址码，约定以低字节的地址为字地址．存放在字中的数据，应将数的高位放在高字节，低位放在低字节.

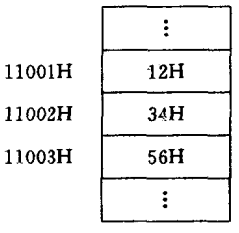

图 1.1 字和字节的内容

设 X 为存储单元的名字，约定，以后用 (X) 表示 X 所代表的存储单元中的内容，<X>表示 X 的地址．例如：

(11001H) =3412H

(11002H) =5634H

显然，因为 4 位十六进制要展开成 16 位二进制，所以这里的 11001H 和 11002H 都是字地址，此时内存的格局见图 1.1.

8086 仅允许以字节为单位或以字为单位访问存储器.

1.2.2 存储器的分段

8086 是 16 位机，参与运算的数据不能超过 16 位，但其地址码是 20 位的，为了能在 16 位机上实现 20 位的地址运算，需要一个计算地址的附加装置．为了计算物理地址，可把内存分成若干段，每段最长为 2^{16} 个字节（64K）．各段不需相等，并且可以重叠．每段有个段头地址，段内每个字节以段头为基址相对编址，叫偏移地址．显然，一个段的最大偏移地址为 0FFFFH，即不超过 16 位二进制．这样，可得如下的计算物理地址的公式：

物理地址＝段头地址＋偏移地址

硬件规定，段不能起始于任意地址，而必须从一个小段的首地址开始．从 0 地址开始，每 16 个字节为一个小段，因此在十六进制表示中，每个小段的首地址最低位为 0，即 20 位地址的低 4 位为 0．因此，段始地址只需保存高 16 位即可.

一个程序在运行的任意时可访问如下 4 种段：

1）现行代码段，段寄存器为 CS.

2）现行数据段，段寄存器为 DS.

3）现行堆栈段，段寄存器为 SS.

4）现行附加段，段寄存器为 ES.

每种段的段始地址的高 16 位存放于相应的段寄存器中．计算物理地址的公式为

物理地址＝（段寄存器）×16＋偏移地址

在程序中，指令的操作数若为存储器，表示存储器的总是其偏移地址，计算物理地址的功能由硬件实现．例如：设（DS）=0AB18H，则数据段的最大物理地址为

$$\begin{array}{r} 0AB180H \\ +\quad 0FFFFH \\ \hline 0BB17FH \end{array}$$

偏移地址 0FE7FH 的物理地址为

$$\begin{array}{r} 0AB180H \\ +\quad 0FE7FH \\ \hline 0BAFFFH \end{array}$$

1.2.3 I/O 的结构

8086 系统与外部环境连接的装置叫端口．通过它们 8086 可接收外部信息，并可发出信号来控制其他过程．8086 可访问 2^{16} 个 8 位的 I/O 端口．每个端口由一个地址确定．两个地址相邻的 8 位端口可做为 16 位端口使用．I/O 端口类似存储器，只是没分段，地址用 16 位二进制表示，可认为它们在同一段中.

1.3 寄存器结构

访问存储器需要计算物理地址，这是要花费机器时间的. 因此，使用频率较高的存储地址、数据以及运算的中间结果，在可能的情况下，应存放在寄存器中. 每个寄存器都有一个名字，访问时直接写它的名字即可.

Intel 8086/8088 共有 4 组计 14 个 16 位的寄存器.

1.3.1 通用寄存器

通用寄存器共有 4 个，见图 1.2. 寄存器 AX，BX，CX，DX 都是 16 位的，但每个都可分成两个 8 位寄存器使用，其名字见图 1.2. 这种双重性可使 8086 方便地处理字和字节信息.

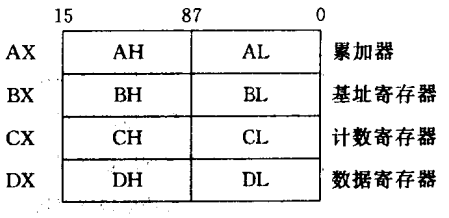

图 1.2 通用寄存器

我们称它们为通用的，是因为它们都能用于算术运算和逻辑运算，而且通常不需强调哪个能做目的操作数，哪个能做源操作数，但它们必竟是 4 个不同的寄存器. AX 又称累加器，这是因为硬件要求乘、除指令中，被乘数、被除数一定要放在累加器中；输入/输出指令中待传输的数据必须放在累加器中；还有，字节换码指令中，待换码的数据必须在 AL 中等等. CX 被称为计数寄存器，是因为串操作指令、循环指令等是把 CX 做为计数器使用的. DX 叫做数据寄存器，它在输入/ 输出指令和乘除指令中被派做特殊用途. BX 是基址寄存器，常将数组、结构等数据结构的起点的偏移地址放在 BX 中，以便于访问数组元素和结构的域.

1.3.2 指针和变址寄存器

有 4 个指针和变址寄存器，见图 1.3. 指针和变址寄存器可参与运算（所以本书后面将它们和 AX，BX，CX，DX 统称为通用寄存器），但通常情况下它们的内容是偏移地址.

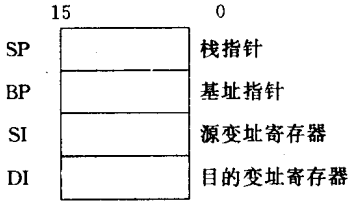

图 1.3 指针和变址寄存器

使用指针或变址寄存器可以缩短指令，因为若将偏移地址放在此 4 个指针寄存器中，然后把寄存器做为间接地址写在指令中，只要有 3 位二进制就可以表示操作数地址了，而在指令中直接写偏移地址本身则要占用 16 位二进制；另外将地址放在这 4 个寄存器中进行运算，并存放，比在通用寄存器中运算然后再传送到指针或变址寄存器中可减少传送指令，从而缩短程序.

如无特殊声明，SP，BP 内隐含栈段偏移地址；SI，DI 隐含数据段偏移地址；否则应加段前缀. 比如：

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

用显式指明 BP 中含有数据段的偏移地址.

SP 和 BP 虽然都是指向堆栈段，但它们是有区别的. SP 总是指向栈顶，BP 可以指向栈内任何地址. 关于 SP，我们有必要提醒读者注意：虽然硬件允许 SP 与其他通用寄存器一样参与运算，但在编程时，只能令 SP 指向栈顶. 所以程序员只能给 SP 赋初值或调整栈指针，以及通过写 PUSH（压栈）、POP（弹栈）、CALL（转子）、RET（返回）等指令来改变 SP 的值. 编程时 SP 通常不应参与其它运算. 通过使用 BP 来访问存放在堆栈段的数组或结构是很方便的.

SI 和 DI 也是有区别的，硬件规定：串操作时，SI 一定指向源串，DI 一定指向目的串，并且，源串在数据段，目的串在附加段.

BX 中的偏移地址，如果没有特殊声明，应在数据段.

1.3.3 段寄存器

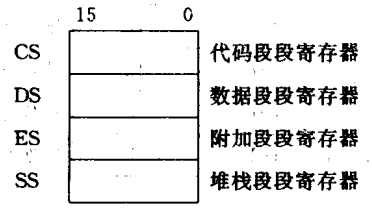

图 1.4 段寄存器

段寄存器有 4 个，见图 1.4.

段寄存器的用途前面已经介绍过了，这里仅提醒读者注意，CS 的加载是由连接程序（LINK）完成的，用户程序不能对 CS 赋值；而 DS、ES 的值必须由用户程序加载. SS 可由用户加载，或当用户指定段的连接类型为 STACK 时，由连接程序加载.

1.3.4 控制寄存器

控制寄存器包括指令指针寄存器 IP 和标志位寄存器 FLAGS.

IP 用来存放代码段中的偏移地址. 在程序运行的过程中，它始终指向下一条要执行的指令，它与 CS 寄存器联合确定下一条要执行的指令的位置，而控制器一旦取得这条指令就马上修改 IP 的内容，使它包含下一条指令所在的偏移地址. 可见，计算机就是用 IP 寄存器和段寄存器 CS 来控制指令序列的执行流程的.

FLAGS 由 6 个条件码标志和 3 个控制标志组成，它们在 FLAGS 占据的二进制位如图 1.5 所示，其中的空白位尚未占用.

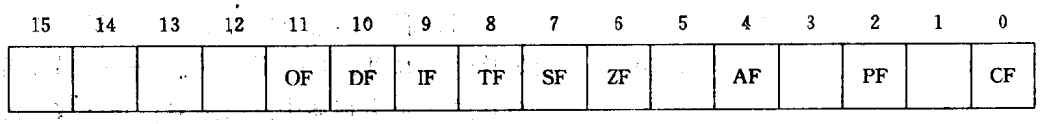

图 1.5 标志位寄存器

FLAGS 中条件码标志用来记录程序运行结果的状态信息. 由于这些状态信息往往作为后续的条件转移指令的转移条件，所以称为条件码. 有关条件码的详细介绍见表 1.1.

表 1.1 FLAGS 中的条件码

标志	名称	值为“1”的条件	常见的用途	标志为 1 的符号	标志为 0 的符号
CF	进位标志	运算时，最高有效位有进位	测试无符号数加法溢出，或减法不够减；或移位指令移出的值是什么	CY	NC
AF	半进位标志	低 4 位向高 4 位有进位	CPU 测试十进制数运算时，低位是否向高位有进位	AC	NA
PF	奇偶标志	结果低 8 位有偶数个“1”	核对 ASCII 码的奇偶性	PE	PO
ZF	零标志	结果为“0”	测试结果是否为“0”	ZR	NZ
SF	符号标志	结果最高位为“1”	测试有符号数运算结果是否为负；或无符号数运算结果的最高位是否为“1”	NG	PL
OF	溢出标志	运算过程中，操作数超出了机器所能表示的范围	有符号数运算结果是否溢出	OV	NV

表 1.1 中标志为“1”的符号、标志为“0”的符号出现在调试程序 DEBUG 中. 它用符号表示标志位的值. 状态信息是 CPU 根据计算结果自动设置的，但是，在必要时，程序员可用指令设置 CF 的值.

FLAGS 中的控制标志是 IF，DF 和 TF.

IF 叫中断标志，当 IF 为 1 时，CPU 允许中断，当 IF 为 0 时，关闭中断. DEBUG 用符号 EI 和 DI 表示 IF 的值为 1 和 0.

DF 叫方向标志，在串操作指令中用 DF 控制处理信息的方向. 当 DF 为 1 时，表示逆向操作，即每次操作后使变址寄存器 SI 和 DI 减量，这样就使串操作从高地址向低地址进行. 当 DF 为 0 时，表示正向操作，即操作后 SI 和 DI 增量，于是串操作从低地址向高地址进行. 表示 DF 为 1 和为 0 的符号分别是 DN 和 UP.

TF 叫陷井标志，用于单步方式操作. 当 TF 为 1 时，每条指令执行完后产生陷井，由系统控制计算机；当 TF 为 0 时，CPU 正常工作不产生陷井，即指令连续执行.

IF 和 DF 的值由程序员用指令设置，TF 的值由调试程序 DEBUG 设置.

1.3.5 8086 的内部结构

8086 微处理器由两部分组成，见图 1.6，一部分是执行单元 EU，它负责执行指令；另一部分是总线接口单元 BIU，它负责提供指令及数据，并负责计算物理地址和执行总线周期. BIU 设立了一个指令队列排队器，可先行取 6 个字节的指令. BIU 和 EU 可分别独立控制，以使指令的执行和取出重叠进行. 在系统其他部件不占用存储周期时，BIU 则占用存储器总线，使其不断“忙于”按指令顺序予取以后的指令字. 所以，当 EU 执行指令时，BIU 已把下条指令取出. 这样可以提高 CPU 的速度.