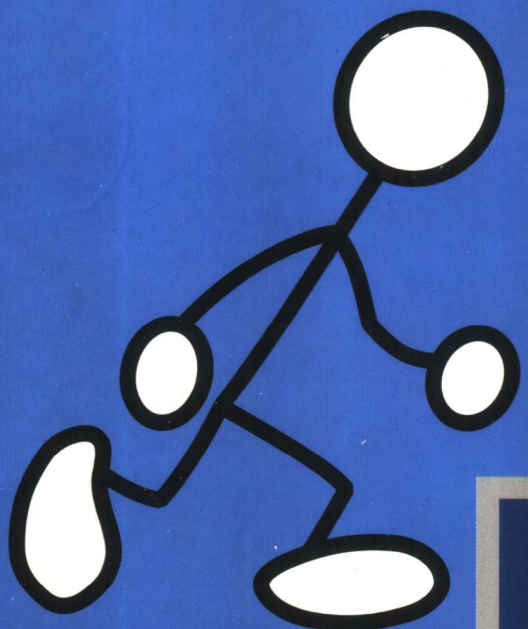
新大纲，新考点，新法宝
——助您顺利过关，取证

最新计算机等级考试教程

三级A类（下册）



陶宏才　吴学畅
冉蜀阳　张丽梅　编著

机械工业出版社
China Machine Press

最新计算机等级考试教程

三级A类

（下册）

陶宏才 吴学畅 冉蜀阳 张丽梅 编著

机械工业出版社
China Machine Press

本书根据国家教育部制定的《全国计算机等级考试三级A类考试大纲》的要求编写，可作为全国计算机等级考试三级A类教材。

本书分上、下两册，上册主要介绍计算机基础知识、操作系统及软件基础、数据结构与算法、微机组成原理与接口技术；下册主要介绍汇编语言程序设计、计算机测控技术、计算机网络与数据通信基础和计算机基本操作知识。为帮助读者掌握所学内容，每章后均附有习题。

本书可供计算机硬件应用人员使用，更是参加等级考试人员的必备参考书。

图书在版编目 (CIP) 数据

最新计算机等级考试教程. 三级A类(下册)/陶宏才等编著. - 北京：机械工业出版社，2000.1

ISBN 7-111-07675-3

Ⅰ.最… Ⅱ.陶… Ⅲ.电子计算机 - 水平考试-教材，Ⅳ.TP3

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：卢志坚

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2000年1月第1版 · 2001年3月第4次印刷

787mm × 1092 mm 1/16 · 15印张

印数：11 001-14 000册

定价：25.00元（下册）

《最新计算机等级考试教程》编委会

序　言

计算机技术是近20年来发展最迅猛、应用最广泛的现代科学技术之一，也是20世纪信息技术产业的关键核心技术，是21世纪知识经济时代的重要基础技术。在当今社会中，人人都应当掌握这一基本的生存技能与必备的文化素质。因此，在全民中普及推广和开发应用计算机技术，对国家的生存和发展，无疑具有不可低估的社会经济价值和历史进步意义。

当今在全国开展的“全国计算机等级考试”，对推动全社会学习计算机技术具有不容轻视的影响。计算机技术的开发与应用，说到底，无非是在普及计算机基础知识和基本操作的前提下，进行计算机程序的设计与运用。因此，“如何使学生轻松、愉快地掌握计算机基础知识和基本操作以及学好计算机程序设计，怎样让教师愉快、轻松地教好计算机基础知识和计算机程序设计”，自然地成为“全国计算机等级考试”教材要实现的重要目标，是值得探索与实践的重大课题。

为此，我们根据教育部考试中心1998年新颁布的“全国计算机等级考试大纲”，组织富有教学经验的计算机基础课教师编写了这一套系列教材，作为全国计算机等级考试的教学(包括自学)用书。本系列教材深入浅出地讲明了大纲规定的一、二、三级的应知应会内容；同时，编者通过各种类型的模拟试题，立足根本，引导读者掌握审题技术，揭示解题思路，着力于帮助读者逐步提高解题能力和应试技巧。希望读者在学习这些教材时，不要死记硬背，多从理解上下功夫，以达到融会贯通。我们衷心希望这些教材能成为广大考生和计算机爱好者的良师益友，在教学中收到事半功倍的效果。

史济民

1999年2月

前　言

本教材系按国家教育部考试中心1998年最新修订的《全国计算机等级考试三级A类考试大纲》，由高等院校多年从事计算机教育的第一线专家教授组成的编写委员会编审出版。全书分为下、下两册。在编写出版过程中，得到了机械工业出版社的大力支持与合作，在此表示感谢。

全国计算机等级考试三级A类侧重点是面向计算机硬件方面的应用人员。要求应试人员全面掌握计算机硬件基础、软件基础、数据结构与算法、微机组成原理与接口技术，汇编语言程序设计、微机测探技术和计算机网络方面的知识和应用能力。本教材正是按这一主线来编排的。

本教材内容丰富、实用，叙述条理清楚、循序渐进，整体结构紧凑，风格前后一致，每一章后均配有丰富的习题和答案，便于自学。

本教材上册由冯军焕编写第1章，李有梅编写第2章，杨燕编写第3章，文登敏编写第4章；下册由吴学畅编写第1章，冉蜀阳编写第2章，陶宏才编写第3章，张丽梅编写第4章。全书上册由冯军焕、下册由陶宏才统稿并审定。

由于计算机技术是一门迅速发展的学科，涉及面广泛、内容新颖，作者水平所限，这套教材肯定会有很多不足之处，殷切希望广大读者批评指正。

编　者

1999年10月

目　录

第1章 汇编语言程序设计

1.1 汇编语言基础知识

汇编语言与高级语言不同，前者面向计算机，后者面向问题。汇编语言程序执行速度快，所占内存少。因此尽管汇编语言编程和调试时间长，程序设计技巧性强，但作为一个软件工作者，特别是系统软件和实时控制的软件程序设计者，应能熟练地使用汇编语言编程和调试。

1.1.1 汇编语言的功能特点

汇编语言是面向计算机的程序设计语言，与具体的计算机硬件有着密切的关系。因此，用它们编写出的程序只适用于某一个系列的计算机，可移植性差。但由于汇编指令与计算机语言指令一一对应，即一条汇编语言的可执行指令对应着一条计算机语言指令。因此，汇编语言可直接利用计算机硬件系统的许多特性，如寄存器、标志位以及一些特殊指令等，执行速度快、占用内存少。由于汇编语言要求对计算机的组成部分进行操作，因此学习汇编语言程序设计除了要掌握计算机的指令系统外，还要熟悉计算机的内部结构，特别是中央处理器(CPU)和存储器的结构，以及与编程有关的其他部分(芯片)的结构，如中断系统、视频显示、键盘接收、定时功能、通信等。

高级语言(如Paascal、FORTRAN、Basic)是面向问题的，它与计算机的硬件无关，可以在各种不同的计算机上运行。因此可移植性好。但是用高级语言编写的程序是不能直接执行的，需要由编译程序或解释程序将它们翻译成对应的目标程序，才能被计算机所接受。由于高级语言指令与计算机语言指令不是一一对应的，往往一条高级语言指令要对应着多条计算机语言指令。因此，这个翻译过程要比将汇编源程序翻译成目标程序花费的时间长得多，产生的目标程序也较冗长，占用存储空间大，执行的速度也慢。虽然采用高级语言编写程序可以节省软件开发的时间，但它不允许程序员直接利用寄存器、标志位等计算机硬件，因而影响了许多程序设计技巧的发挥。

是否采用汇编语言编写程序，要视具体的应用场合而定。一般来说，某些对时间和存储器容量要求较高的程序用汇编语言来书写，如系统软件、实时控制系统、智能化仪器仪表软件等。

1.1.2 计算机指令与汇编语言

1. 基本概念

(1) 指令

指令是计算机所能够接受电脑软件工作者的命令的最小工作单位。它最终是由机内电气元件的状态来体现的。这个状态被译码器所识别，并经过一定的时间周期付诸实现。由此便完成了指令所规定的操作，我们称这种指令为计算机指令。

(2) 代码指令

代码指令是计算机指令的一种数据表示形式，通常用二进制、八进制或十六进制表示。例

如8086/8088微处理器有一个状态标志寄存器，其中有一位是进位标志，可以通过一些计算机指令使其置“1”或置“0”，其中一条二进制代码的形式是:

11111000

它的操作结果使进位标志置“0”。

我们称11111000为该计算机的二进制代码指令，而称相应的八进制数370Q为八进制代码指令，称相应的十六进制数F8H为十六进制代码指令。以下在不引起混淆的情况下，将计算机指令和代码指令混用，不加区分，这一点务请读者注意。

(3) 计算机指令程序

完成某一个特定的计算或一系列操作的代码指令的有序集合称为代码程序或计算机指令程序。计算机指令程序必须预置在内存储器的某一个部分，并指出其起始位置后才能予以执行。无论何种机型采用何种方式工作，其作为一个完整的程序，总是存在着一个隐含的规则，就是在通常的情况下总是按照指令存放的顺序由低地址向高地址逐条予以执行，直到遇到转移指令才转向新的地址顺序执行，或是遇到停机指令才终止程序的运行。这个起始位置和执行顺序的自动递增和改变，在8086/8088微处理器中是通过被称作IP的指令指针计数器实现的。IBM-PC机上一个简单计算机指令程序如例1-1 所示。

例1-1 8088计算机指令程序为

```
1D210H:    B233H
1D212H:    B402H
1D214H:    CD21H
1D216H:    CD20H
```

其中，冒号左边表示单元地址，冒号右边是单元的内容。例中共4条指令，按1D210H地址启动，其结果在PC机上输出一个字符“3”。

(4) 汇编指令

汇编指令是指用助记符表示相应计算机指令的操作码和操作数，按照一定的格式书写的一种面向计算机的指令形式，又称符号指令。早期的计算机只能执行计算机指令程序，这给程序设计者带来极大的不便，即需要记忆或不断地查找上百条乃至上千条指令。高级语言则不同，它类似自然语言而又区别于自然语言，在词法、句法和文法方面有着严格的要求。然而高级语言并不能够直接为计算机所认识，一个计算机要运行高级语言，必须首先用翻译程序将其翻译成计算机语言程序。该翻译程序称为解释程序或编译程序，它是一个可以执行的计算机语言程序，其用途是实现高级语言向计算机语言的转化。汇编指令也必须经过翻译程序（汇编程序）翻译后才能变成计算机指令。

(5) 汇编指令源程序

汇编语言源程序又称汇编源程序，是由汇编指令按照一定的语法规则书写而成的。在汇编源程序中，除了上述的汇编指令外，还有必要的数据及其结构的描述，以及源程序向汇编程序(即翻译程序)提供的一些必要的信息，这些是通过被称作伪汇编的指令(伪指令)实现的。伪指令是相对于汇编指令而言的，它并不像汇编指令那样一对一地被汇编(翻译)成代码指令。一个简单的汇编语言程序见例1-2。

例1-2 输出字符'3'的汇编语言程序为

```
MOV     DL,     '3'
MOV     AH,     2
INT     21H
INT     20H
```

2. 80X86上的汇编语言

80X86上运行汇编语言的汇编程序。目前在DOS操作系统一般用MASM宏汇编，相应的调试程序是DEBUG或TDEBUG。一个存放汇编语言源程序的盘文件应取扩展名.ASM，经汇编后可有选择地产生扩展名为.LST的源程序清单(即列表)文件以及.OBJ为扩展名的浮动二进制代码文件(目标文件)。此外还可以有选择地产生.REF符号索引文件。汇编以后的浮动二进制代码文件.OBJ是不能用来调试和执行的，应当使用连接程序将其连接后产生一个扩展名为.EXE的文件才能执行和调试。

1.1.3 汇编语言中的数据表示及数据类型

本节对上册书中介绍过的内容再做一些简单的回顾。读者如果对这些内容已经熟悉，可跳过这一节。

1. 二进制、八进制与十六进制数的表示

任意一个十进制数上的每一位数。都可以取0~9这十个数码中的一个，并且是“逢十进一”的。而二进制数每个数位只可能取两个不同的数码“0”或“1”，并且是“逢二进一”的。

一个十进制数2345可表示成如下以10为基数的形式:

$$2345=2\times10^3+3\times10^2+4\times10^1+5\times10^0$$

一个二进制数1010可表示成如下以2为基数的形式:

$$(1010)=1\times2^3+0\times2^2+1\times2^1+0\times2^0$$

2^i(i为正整数)称为每一位数字对应的权。所以二进制数1010自左至右每一位数字对应的权分别为2^3、2^2、2^1、2^0。

为了与十进制数相区别，常在二进制数后面写一个字母B。例如：1010B，而在十进制数后面写上字母D，例如：2345D。但一般十进制数，都省略字母D。

由于二进制数书写起来很长，不易看懂且记忆不方便，因此常用八进制或十六进制来表示。为区别起见，八进制数后面用字母“Q”做标记，而十六进制数后面用字母“H”做标记。

八进制数一位对应三位二进制数，用0~7来表示；十六进制数一位对应四位二进制数，以16为基数，每一个数位都有16种状态，即用数码0~9和A~F来表示，其中A~F表示十进制的10~15，并且是“逢十六进一”。

例如，十六进制数308BH可表示成如下形式:

$$308BH=3\times16^3+0\times16^2+8\times16^1+B\times16^0$$

表1-1列出了十进制、二进制和十六进制之间的相互关系。

表1-1 十进制、二进制与十六进制数字对照表

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0000	00H	8	1000	08H
1	0001	01H	9	1001	09H
2	0010	02H	10	1010	0AH
3	0011	03H	11	1011	0BH
4	0100	04H	12	1100	0CH
5	0101	05H	13	1101	0DH
6	0110	06H	14	1110	0EH
7	0111	07H	15	1111	0FH

有一点请注意：在汇编语言中，凡是以字母A~F开头的十六进制数，都要以数字“0”作

为开头，以避免与标识符相混淆。例如：十六进制数B3H应写成0B3H。

2. 带符号二进制数的表示

前面所说的二进制数均为无正、负符号的数，不管这个数含有多少个二进制数位，这些数位都是该数中的一个组成部分。例如，8位无符号数的形式为$D_7D_6\cdots D_0$，其数值范围为00~0FFH，即0~255。16位无符号数的形式为$D_{15}D_{14}\cdots D_0$，其数值范围的0000H~0FFFFH，即0~65535。

由于二进制数上的每一个数位只有“0”和“1”两种状态，因此，对于一个带有正号“+”或负号“-”的二进制数的符号来说，也只能用这两种不同的状态来区别。在计算机内部，符号实际上是被“数码化”了。

在计算机中，带符号的数通常是利用该数的最高有效二进制位来作为符号位(S)，S为0则表示该数为正数，S为1则表示该数为负数。由于符号位占据了一个二进制位，所以，这样的二进制数比相对应位数的无符号二进制数来说，其数值范围减少了一半。例如：8位有符号数表示形式为$SD_6D_5\cdots D_0$，其数值范围为80H~7FH，即-128(80H)~+127(7FH)。

对任意一个带符号的负数来说，计算机都是按补码的形式来存储和处理的。

3. 数的补码表示

一个负数数值的补码形式是先写出该负数所对应的正数的二进制形式，对该正数的每一个二进制位取反(即0变1，1变0)，然后再将取反后的结果加1，最后得到的就是这个负数的补码形式。简单地说就是：取反加一。

例如，对于-5来说，其所对应的正数的二进制形式为00000101B，每一位取反后为：11111010B，取反后的结果加1即是-5的补码形式11111011B。

因此，在计算机内部，-5是以11111011B(或0FBH)的形式存储的。

负数参加运算时，也是以其补码来进行的。例如：

8-2=8+(-2)=00001000B+11111110B=00000110B=6

4. 字节(BYTE)、字(WORD)、双字(DWORD)、四字(QWORD)和十字节(TBYTB)

(1) 字节

8位二进制数组成一个字节。一个存储单元就是由八个连续二进制位组成的，也就是一个字节。在存储器的每个存储单元中，都可存储诸如字符、数字、计算机指令或另一个存储单元的地址等信息。

字节的最低位一般称为第0位，最高位称为第7位，8个二进制位可产生2^8(即256)个状态。因此，一个字节存储单元内可以是0~255之间的任意一个正整数。

(2) 字

2个字节(即16个二进制位)组成一个字，因此，一个字可有2^{16}(即65536)个状态。其中高8位称为高字节，低8位称为低字节。存储单元按字节编址，而大部分的数据又都是以字为单位表示的。那么一个字是怎样存入存储器的呢？当把一个整数字保存在存储器中时，两个字节的存放顺序是相反的，即高字节存储在高地址部分，低字节存储在低地址部分。例如：把25B0H这个数存放在存储器中时，其存放顺序为：

存储器地址	数值	
1000H	0B0H	低字节
1001H	25H	高字节

注意，同一个地址既可看作字节单元的地址也可看作字单元的地址。例如，把1000H号单元看作字节单元的地址时，它的内容为0B0H；看作字单元的地址时，它的内容为25B0H。当给定一个地址后，用什么办法来区分它是一个字节单元的地址，还是字单元的地址呢？形式上

不好区分，只有根据使用情况在编写程序时予以确定。另外，字单元的地址可以是偶数编址，也可以是奇数编址。一般情况下字单元的地址不以奇数编址，这是因为计算机是以偶地址访问存储器的。因此，当以奇地址作为字单元的地址访问时，计算机必须经过两次访问才能实现，这样做当然要花费较多的时间。

(3) 双字

双字由2个连续存放的相邻的字(即4个字节)组成，其宽度为32位。双字的存储顺序与字类似，即低字存在低地址部分，高字存在高地址部分。利用双字进行算术运算，可以提高运算的精度。

(4) 四字

四字由4个连续字组成，其宽度为64位。它可解决双字满足不了的精度问题。四字的存储与上述一致，即最低字存在低地址的存储单元，而最高字存储在高地址存储单元。

(5) 十字节

十字节如同其名字一样，由十个字节组成，其宽度为80个二进制位。它可以存储极大的数字或字符串。它的存储结构也与上述方法一致，这里不再赘述。

5. ASCII码

一个字节中的8位并不总是代表着数字值。我们知道，每一台计算机都配有键盘，上面有数字、字母和一些特殊字符的相应键。在一定条件下，每按下一个键，送入到计算机中的这个键所表示的内容就被转化为一个八位的二进制数，在这里，称这个八位的二进制数为字符。ASCII码(或字符)就是能够表示字母、数字、专用字符和控制字符的7位编码，它能表示128个符号和代码。例如“A”的ASCII码是41H，最高位(第7位)有时用做数据传输和检索错误的检测代码。有些生产字符ROM芯片的厂家将第7位用于存储扩展的字符集，这样可使表示符号(字符)的数目比原来增加一倍，达到256个。用扩展的ASCII字符可表示某些特殊的外语符号、数学符号以及一些非常有用的图形符号。

6. BCD码

BCD码也称作无符号十进制数，是用二进制编码的十进制数。

BCD码有两种：压缩的BCD码(或无符号压缩十进制数)和非压缩的BCD码(或无符号非压缩十进制数)。

(1) 压缩的BCD码

压缩的BCD码是用一个字节来表示两位二进制编码的十进制数，而每位十进制数用4位二进制数来表示，高4位表示十进制数的十位数字，低4位表示十进制数的个位数字，即0000B~1001B表示十进制数中的0~9。显然4位二进制数中只有0~9是有效的。

例如，十进制数56用压缩BCD码表示为：

0101 0110

5 6

(2) 非压缩的 BCD码

非压缩的BCD码是用一个字节来表示一位二进制编码的十进制数，数值放在低4位中，高4位对乘除法运算必须是零，而对加减法运算可为任意值。两个字节的非压缩BCD码可表示的十进制数范围同样也是0~99。

例如，56的非压缩BCD码的形式为：

0000 0101 0000 0110

5 6

虽然两个十进制数的运算可以用相应二进制数的运算得到，但是由于受计算机字长的限制，

当对于占用较多字节的数进行运算时，这种方法就会受到限制，甚至不能进行，这时就要采用BCD码进行运算，以避免多字节数据运算时所受的限制。

7. 80X86处理的数据类型

80X86支持大多数高级语言所具有的如下基本数据类型:

1) 无符号二进制数。

2) 有符号二进制数　80X86均支持8位、16位的有符号数。80386还支持32位的有符号数。有符号数的负数在机内是以补码形式存放和处理的。

3) BCD 码(二进制编码的十进制数)。

4) ASCII字符。

5) 逻辑地址。

由于80X86对存储空间采用了分段结构，因此，为了正确描述一个存储空间的地址，使用了逻辑地址的概念。逻辑地址包括两个部分：一个是对存储器分段的段地址，另一个是在某个段中查找目的地址的偏移地址。每个段地址和偏移地址均是16位的无符号数。

6) 位　单一的一个二进制位。

7) 串　由多个字节数据组成的一个完整数据。例如，“This is a string”是一个字符串，而12345678H是一个4字节的正数。

1.2 80X86结构

本节的内容在上册书中已作过介绍，但考虑到本章内容的系统性，从汇编语言程序设计的角度对有些内容再做一些强调。对微机原理掌握得较好的读者可跳过本节内容。

本节将首先介绍8086/8088的存储器和中央处理器的内部结构，作为学习汇编语言程序设计的基础知识。

1.2.1 8086/8088CPU、存储器和寄存器

1. 存储器的有关概念

存储器可以视为一个存放信息的大仓库，而一个仓库又分成若干个，成千上万个房间。我们把每一个房间称为一个存储单元。为了能区分出不同的存储单元，应把全部存储单元按照一定的顺序编号，每个存储单元的编号就称为该单元的地址。不同的存储单元具有不同的地址。存储单元内存放的信息代码称为单元的内容，每个存储单元由固定个数的位组成，这个固定个数的位称作单元的字长，也叫计算机的字长。字长是随计算机的大小和用途的不同而有所区别的。从上面的介绍可以看出，一个存储单元有两个重要的属性:

1) 地址——该单元在存储器中的相对位置。

2) 内容——存储在该单元中的信息代码。

为了能区分出存储单元的这两个属性，通常使用一个符号，例如A，表示某个单元的地址，则记号(A)就表示单元A中的内容。假如A单元中放着B，即B=(A)；而B又是一个地址，则可用(B)=((A))来表示B单元的内容。反之，有时也用记号“< >”表示内容所在的地址。即由内容指出地址。例如：<a>=A表示信息代码a存放在地址为A的单元中。

一个存储器的单元总数称为该存储器的容量。在计算机里，存储器通常是以KB为单位描述其存储空间的容量。1KB=2^{10}Byte，即1KB是指1024Byte存储单元。

另外需要注意的是，存储器(ROM除外)有这样的特性：从某个单元取出其内容后，该单元仍然保存着原来的内容不变，以后可继续使用；反之，若往某个单元存入新的信息时，那么原

来的内容就被冲掉，新的信息就自动地取代了原来的内容。如果原来的内容以后还有用，一定要在存入新的信息之前，保存一个副本。

2. 存储器结构

计算机存储信息的最基本单位是一个二进制位(bit)。一位可存储一个二进制数：0或1。由于一位二进制位表示的信息量太少，所以在计算机的存储器里实际存储信息的单位是字节(Byte)。一个字节是由8位二进制位组成的，每个字节的8位从右到左编号为0~7。

计算机中常用的字符就是用字节(B)表示的。因为8086的字长为16位，所以一个字由两个字节组成，每个字的16位从右到左编号为0~15，其中0~7位称为低位字节，8~15位称为高位字节。

由于8086以字节为单位存储信息，为了能正确地存放和取得信息，存储器按字节编址。地址从0开始编号，顺序地每次加1。在计算机里，地址是用二进制数表示的。地址是无符号的整数，为了读写的方便，书写格式采用十六进制数，如图1-1所示。

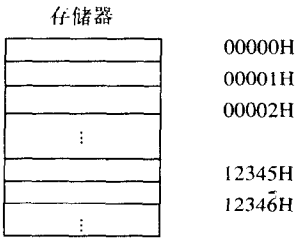


图1-1 存储器组织

3. 中央处理器

中央处理器(CPU)是分析、控制并执行指令的部件，它由算术逻辑部件、控制逻辑部件和工作寄存器组成。在此着重介绍在汇编语言程序设计中经常用到的工作寄存器。

工作寄存器是CPU内部用来存放程序执行过程中所需要的或所得到的各种信息，包括指令地址、操作数地址、操作数以及运算的中间结果、运算结果等等，它在计算机中起着重要作用。每一个工作寄存器相当于存储中的一个存储单元，但它的存取速度比存储单元要快得多。

在8086/8088中，一共有14个16位的工作寄存器，它们是：4个数据寄存器、2个指针寄存器、2个变址寄存器、4个段寄存器和2个控制寄存器。工作寄存器结构如图1-2 所示。

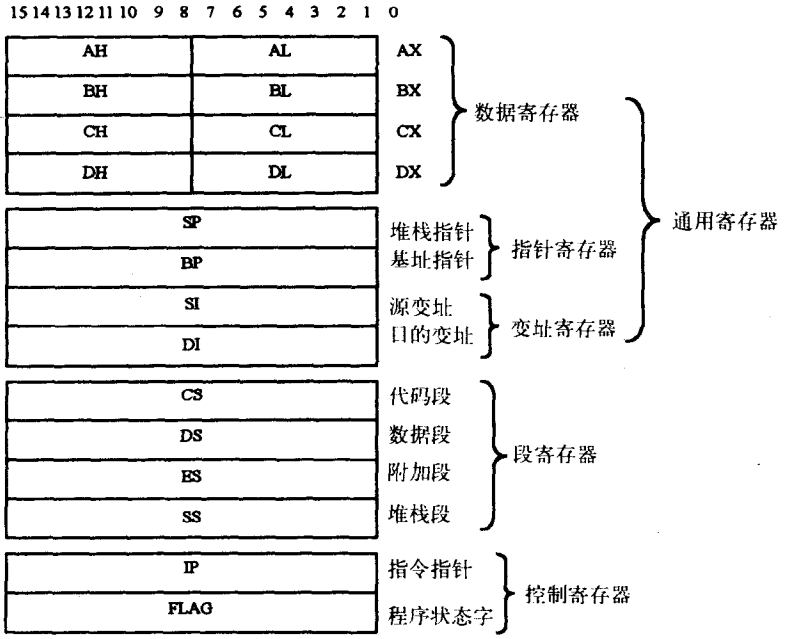


图1-2 工作寄存器结构

(1) 数据寄存器

数据寄存器有4个，包括AX、BX、CX和DX。

1) AX(Accumulator)累加寄存器 I/O指令使用这个寄存器与外部设备传送信息。另外，一些串处理指令和算术运算指令也需要使用这个寄存器。

2) BX(Base)基址寄存器 在生成20位物理地址时经常使用这个寄存器，并使用DS寄存器作为默认的段寄存器。

3) CX(Count)计数寄存器 在循环和串处理指令中用作计数器，以控制重复次数，也用在多位移位操作上。

4) DX(Data)数据寄存器 在作字乘法或字除法时，把DX和AX组合在一起存放一个双字长数，DX用来存放高位字。另外，在I/O指令中，DX可用来存放I/O的端口地址。

以上四个数据寄存器都作为通用寄存器使用，用来暂存计算过程中所用到的操作数、中间结果、运算结果或其他信息。另外，这四个寄存器中的每一个都可单独作为16位的寄存器使用，而且每一个寄存器还可以作为两个8位的寄存器使用。例如AX的高8位为AH寄存器，低8位为AL寄存器。类似的还有BH、BL、CH、CL、DH和DL寄存器。

在8086/8088汇编语言中，一些汇编指令虽然在书写指令时没有写出相应寄存器的名字，但该指令本身就决定了使用某一个指定的寄存器，称其为隐含寄存器。另外，一些寄存器虽然在一般情况下可互换使用，但是作为一些特殊用途使用时，则必须使用其中的某些寄存器，这些请读者必须予以足够的重视。

(2) 指针寄存器

指针寄存器多用于访问堆栈段中的数据。

1) 堆栈指针(SP)寄存器 用来指示栈顶的偏移地址。

2) 基址指针(BP)寄存器 可作为堆栈区中的一个基地址，以此来访问堆栈中的其他信息。常用于访问通过堆栈传递的参数。

以上两个指针寄存器都可以与堆栈段(SS)寄存器联用，确定堆栈段中某个存储单元的地址。

(3) 变址寄存器

变址寄存器分为源变址(SI)寄存器和目的变址(DI)寄存器两种。它们一般与数据段(DS)寄存器联用，用来确定数据段中某个存储单元的地址。在串处理指令中，SI和DI作为隐含的源变址和目的变址寄存器时，SI和DS段寄存器联用，DI和ES段寄存器联用，分别达到在数据段和附加段中寻址的目的。

上述指针和变址寄存器只能作为16位寄存器使用。除特殊用途外，它们还可以像数据寄存器一样在运算过程中存放操作数。

(4) 段寄存器

包括CS、DS、SS和ES四个段寄存器，用于存放相应段的起始地址。

(5) 控制寄存器

① 指令指针(IP)寄存器

它用来存放代码段中的偏移地址，与代码段(CS)寄存器联用，确定下一条指令的物理地址。在程序的执行过程中，控制器根据CS和IP的内容从存储器中取得下一条要执行的指令。而控制器中取得这条指令后马上自动修改IP的内容，使它又指向下一条待执行的指令。如此周而复始，直到程序结束。因此IP寄存器是计算机中很重要的一个控制寄存器。在计算机里，程序之所以能自动运行，靠的就是这个IP寄存器。

② 标志寄存器(FLAG)

也称程序状态字(Program Status Word，缩写为PSW)寄存器。这是一个16位寄存器，但仅占用了9位。由6位条件码标志和3位控制标志构成，还有7位保留位，它们一般为0。标志寄存器各位含义如图1-3 所示。

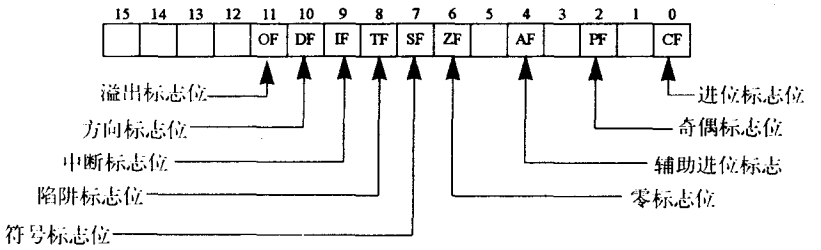


图1-3 标志寄存器结构

条件标志是用来记录程序中运算结果的状态信息。由于这些状态信息往往是作为后继条件转移指令的转移控制条件，所以称为条件码。其中：

1) CF进位位 记录运算时从最高有效位产生的进位或借位值。当有进位或借位时，CF=1，否则CF=0。

2) PF奇偶位 记录运算结果操作数中“1”的个数是偶数还是奇数的情况。当结果操作数中“1”的个数为偶数时，PF=1，否则PF=0。此位可用于检查在数据传送过程中是否发生错误。

3) AF 辅助进位位 记录运算时第3位(字节操作)或第7位(字操作)向高半字节或高位字节进位或借位的情况。当有进位或借位时，AF=1，否则AF=0。

4) ZF 零标志位 当运算结果为0时，ZF=1，否则ZF=0。

5) SF 符号位 记录运算结果的符号情况。当运算结果为负时，SF=1，否则SF=0。

6) OF溢出位 记录运算结果是否超出了计算机所能表示数的范围的情况。若结果操作数超出了计算机所能表示的范围称为溢出。当产生溢出时，OF=1，否则OF=0。

控制标志位的三位是：

1) TF陷阱位 用于单步操作方式，是调试程序的一种手段。当 TF=1时，CPU在每条指令执行完后便产生陷阱(内部中断)，以供检查；否则当TF=0时，计算机正常运行不产生陷阱。

2) IF中断位 当IF=1时，表示允许中断；当IF=0时，则屏蔽中断。

3) DF方向位 用于在串处理指令中控制处理信息的方向。当DF=1时，每次操作后使变址寄存器SI和DI的内容减量，使串处理从高地址向低地址方向处理；当 DF=0时，则使SI和DI的内容增量，使串处理从低地址向高地址方向处理。

在上述标志 FLAG寄存器的各标志位中，条件码的状态信息一般是由 CPU根据运算结果自动设置的；而控制标志位的状态信息则是由系统程序或用户程序根据需要用指令来设置的。为了方便起见，8086/8088提供了一些设置标志位的指令，以便程序员根据需要使用这些指令建立标志状态信息。

调试程序DEBUG提供了测试标志位的手段，即用符号表示标志位的值。表1-2说明了各种标志位(因DEBUG中没有提供TF的符号，所以表格中未包括TF位在内)的符号表示。

表1-2 FLAG 中标志位的符号表示

标志位	标志位为1	标志位为0
CF位	CY	NC
PF位	PE	PO
AF位	AC	NA
ZF位	ZR	NZ
SF位	NG	PL
IF位	EI	DI
DF位	DN	UP
OF位	OV	NV