IBM PC宏汇编语言程序

设计及应用

IBM PC
宏汇编语言程序设计及应用

王 保 恒 编

国防科技大学出版社

[湘] 新登字 009 号

内 容 简 介

本书以国内广泛使用的 IBM PC 系列微机为背景，重点讲述 IBM PC 微机的宏汇编语言 MASM。在系统介绍 8086/8088 和 8087 微处理器的结构组成、数据表示及指令系统基础上，详细讨论汇编语言程序设计的基本概念、基本方法和设计技巧。此外，还面向较高水平程序员，从开发应用的角度出发，讲述 DOS 系统功能调用、ROM BIOS 中断调用和用户程序驻留内存问题，特别对中断服务程序和设备驱动程序的设计进行了深入的讨论和研究。最后，还对 NDP 机的编程和 80286 的扩充功能进行了分析和介绍。

全书共八章九个附录，每章均配有例题和习题，重点突出，内容丰富，本书可作为高等院校计算机及有关专业的教材，也可供从事微机科研生产及开发应用的科技人员自学或参考。

IBM PC 宏汇编语言程序设计及应用

王保恒 编

责任编辑：宋焕章 朱淑娥

*

国防科技大学出版社出版发行

（湖南长沙砚瓦池正街 47 号 邮码 410073）

国防科技大学印刷厂印装

新华书店总店科技发行所经销

*

开本：787×1092 1/16 印张：19.75 字数：459 千字

1992 年 8 月第 1 版第 1 次印刷 印数：1—11000 册

ISBN 7—81024—183—4

TP·33 定价：9.95 元

前　言

程序设计是计算机各专业重要的技术基础课，也是试图用计算机解决和处理问题的科技人员的必修课。

汇编语言是介于计算机能直接理解的机器语言与使用者容易理解的高级语言之间的一种语言。它除有与代码指令一一对应的符号指令外，还增加了专用于定义变量、常量、符号、过程、分配存储空间、定位程序起始地址等一系列称之为伪指令的符号指令。同高级语言比较，它更接近机器语言，更能全面地反映计算机硬件的功能与特点。同机器语言比较，它更易于阅读，编写和修改程序。因此，使用汇编 语言可编写出运行速度快、占存储空间少、能充分利用硬 件资源、发挥计算机效能并能进行精确控制的程序。汇编语言广泛应用于如下方面：高级语言编译程序的编制；编辑、调试、链接装配、磁盘驱动和磁盘读写等实用程序的编制；控制、指挥和监测等实时处理程序的编制；计算机系统的开发。尽管目前存在着独立于计算机的上千种高级语言，汇编语言仍作为一种强有力的软件工具显示出它的重要性。

汇编语言将因计算机的不同而异。因此，亦称它为面向机器的语言。本书是以 IBM PC 系列微机作为模型机编写的教材，因此必然带有该系列微机的浓重色彩。但实践和经验证明，这是无关大局的。因用汇编语言进行程序设计的基本概念、基本技巧和基本方法是普遍适用的，掌握一种系列或型号计算机的汇编语言程序设计，其它便可触类旁通。

本书共八章九个附录。第一章简单介绍 IBM PC 微机的硬件结构，第二章系统介绍 IBM PC 系列微机的简单指令。第三章详细描述 IBM PC 系列微机宏汇编语言的语句结构、语法规则和各种伪指令的功能、使用方法与使用约定。第四章讨论汇编语言程序设计的直接、分枝、循环和子程序设计技术，并介绍与这些设计技术密切相关的复杂或专用的 IBM PC 系列微机的指令。第五章介绍宏汇编语言为用户提供的两个有用工具——宏处理与条件汇编。第六、七两章面向较高水平程序员。从开发应用的角度出发详细阐述宏汇编语言与系统软件 DOS、ROM BIOS 的程序接口；讨论中断服务程序的设计与驻留内存的方法；介绍可安装的设备驱动程序的设计方法；分析协处理器 8087 的硬件结构、指令系统、数据表示格式以及 NDP 机的汇编语言程序设计方法。第八章介绍 80286 微处理器的组成和 80286 扩充及增加的指令。对支持保护虚地址操作方式的描述符、选择符、描述符表、特权级等问题也给予较详细的说明。

阅读本书之前至少应学过一门高级语言，了解计算机的结构组成及其功能，掌握数制码制及其转换。

本书带 * 章节有一定的难度，前五章不带 * 章节是基础部分，可作为一般院 校计算机专业的教材，讲授 60 学时。对于重点院校，则应讲述除 6.5 节外的前六章。5.4 节、第六、七两章可作为研究生开发应用微机的参考书或教科书。

本书有例题，思考题和练习题近 200 个。其中有的是为理解基本概念和解释语句的功能而设置，有的是为应用提出，也有难度较大实践性很强的开发性题，书中非示意性例 题都 经上机实践通过。

本书是编者在多年为计算机专业讲授汇编语言程序设计课所编讲义的基础上修改而成，其中 4.2 节和附录七由刘真同志编写。

在本书的编写过程中，得到齐治昌、王凤学、王广芳和郭浩志教授，陈怀义、窦文华、马英仲副教授的热情指导和帮助。郭浩 志、陈怀义还审阅了部分书稿。六〇一教研室实验室的同志们为该书的实践提供了大量的机时，谨在此表示衷心的感谢。

对本书错误和不当之处，敬请专家、同行及广大读者批评指正。

编　者

1991 年 6 月

目　　录

第七章　NDP 机汇编语言程序设计

第八章　80286 微处理机

第 一 章

IBM PC 微机结构组成

任一计算机的硬件系统，都由五个必不可少的基本部分组成。它们是输入、输出、运算器、控制器和内存储器。IBM PC 微型计算机硬件系统当然也不例外，其结构如图 1.1 所示。

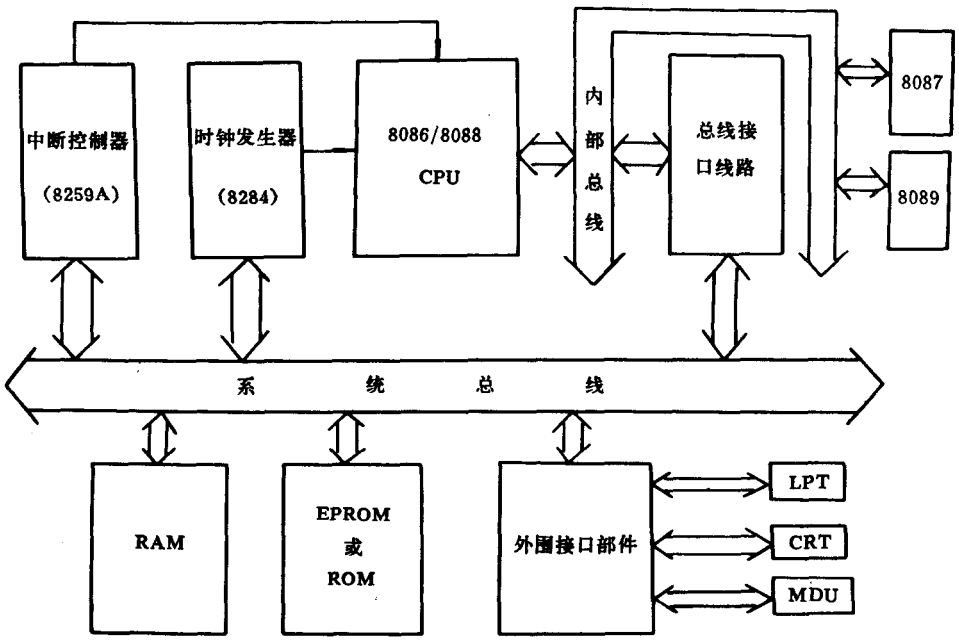


图 1.1 IBM PC 微型机系统

IBM PC 微型计算机硬件系统最大特点是模块结构，组成的系统可大可小，通过系统总线和内部总线连接各部件。图 1.1 中的中断控制器、时钟发生器、8088 或 8086CPU、内部总线和总线接口线路构成运算器和控制器。RAM 和 EPROM（或 ROM）组成内存储器，外围接口部件、键盘显示终端 CRT、行式打印机 LPT 和硬磁盘或软磁盘 MDU 组成输入输

出，由图 1.1 可以看出，这些部件都通过内部总线和系统总线连接成系统，其中 8087 和 8089 是可选 的协处理器。

输入输出的功能是实现人机间通信和信息交流。内存储器的功能是存储程序和数据。运算器的功能是完成算术运算和逻辑操作。控制器的功能是控制协调全机工作。

1.1 8086/8088CPU

8088CPU 具有八位数据通道，可以与存储器或输入输出交换信息，而 8086CPU 则为十六位通道 。8086CPU 具有 6 字节的指令流队列，8088CPU 则是 4 个字节。其它方面，两个处理器是相同的。为其中一个 CPU 编写的软件可以不修 改地在另一个 CPU 上运行。IBM PC 系列机及其兼容机广泛采用了 8088CPU。

图 1.2 为 8086/8088CPU 结构示意图。图中总线接口线路和系统总线并不属于 8086/8088CPU，是为使读者明了 8086/8088CPU 与系统如何连接而加画上去的，图中虚线表示控制线。

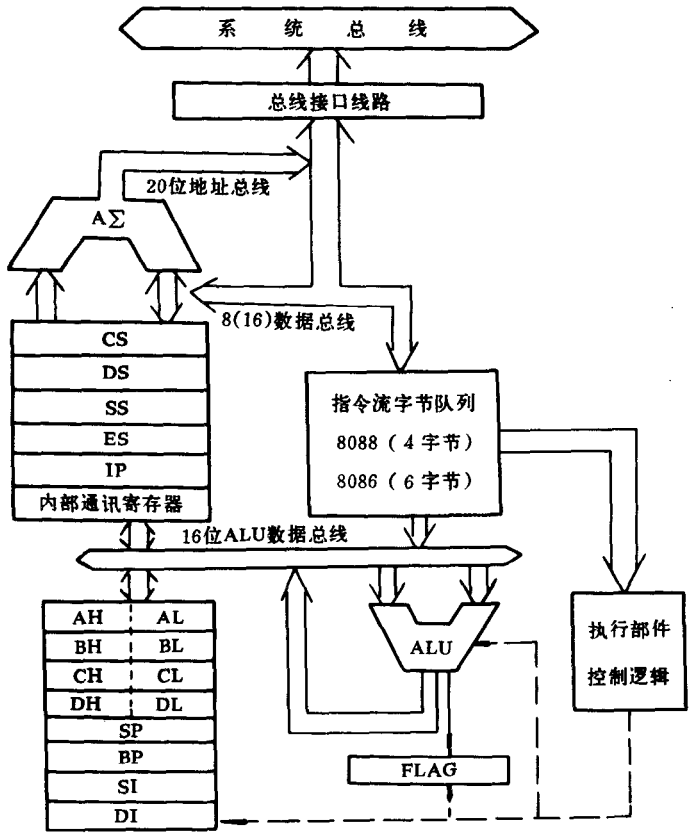


图 1.2 8086/8088CPU

8086/8088CPU 可分为两大组成部分。以 16 位 ALU 数据总线为界，下面的部分为执

行部件 EU（Execution Unit），上面的部分为总线接口部件 BIU（Bus Interface Unit）。

BIU 负责 8086/8088CPU 与存储器和 I/O 设备之间的信息传送，具体地说，BIU 负责从内存指定地址取出指令，并送指令流字节队列去排队。在执行指令时，所需的存储器操作数也由 BIU 从内存指定单元取出，传送给 EU 去执行。该部件中的四个段寄存器 CS、DS、SS 和 ES，指令指针 IP 以及地址加法器产生存取数据或指令的 20 位地址。指令流字节队列寄存取出来的指令。内部通信寄存器则是 I/O、MEM 与 EU 部件交换数据的暂存寄存器。

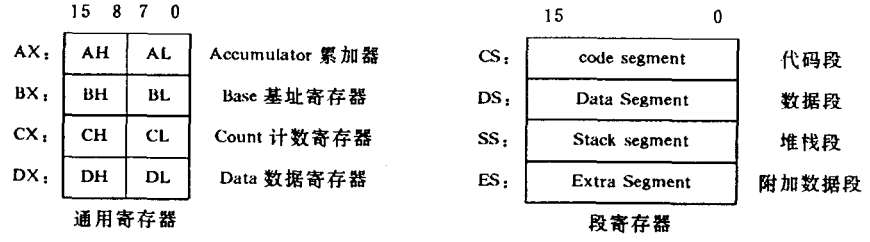
EU 控制和执行指令。它由 8 个 16 位寄存器、EU 控制部件、标志寄存器 FLAC 以及算逻运算部件 ALU 构成。ALU 可实现 16 位的算逻运算，FLAG 寄存执行结果的状态信息 8 个 16 位寄存器的功能见 1.2 节。由于取指部件和执行指令部件是分开的，所以在一条指令的执行过程 中，就可以取出下一条或多条指令，并放到指令流字节队列中去排队。当一条指令执行完以后立即执行下一条指令。减少了 CPU 为取指令而等待的时间，提高了 CPU 的效率，加快了系统的运行速度。

1.2 8086/8088CPU 寄存器

这里介绍的寄存器是 IBM PC 宏汇编的指令性语句直接涉及到的寄存器，不包括内部的数据暂存器，它可分为四组如图 1.3 所示。

一、通用寄存器组

它包括 AX、BX、CX 和 DX 四个 16 位寄存器。每个寄存器的高 8 位和低 8 位可以分开作为两个 8 位寄存器，或在一起作为 16 位寄存器使用。因此，引用寄存器的高、低 8 位都有自己的名字，低 8 位分别被命名为 AL、BL、CL 和 DL，高 8 位被命名为 AH、BH、CH 和 DH。通用寄存器的双重性，使它们在处理字信息（16 位）和字节信息都 一样灵活方便。这些通用寄存器的内容可以互换地参于算术运算和逻辑操作，都可以作为累加器。例 如减法指令 SUB 可把任何 8 位或 16 位通用寄存器的内容，同其它通用寄存器的内容相减，并把结果送入这两个寄存器中的任意一个，有少量指令把某些通用寄存器作为专用。例如，串操 作指令需要 CX 寄存器隐含寄存串中元素个数的计数，而 AX、BX 和 DX 却不能用于这个目的，又如可以用 DX 寄存器间接指出 I/O 设备的端口地址，其它寄存器则无此功能。由于 CX、DX 的这种特殊用法，故把 COUNT、DATA 的缩写作为这两个寄存器的名称，同理 AX、BX 寄存器则用 ACCUMULATOR、BASE 的缩写来命 名，通用



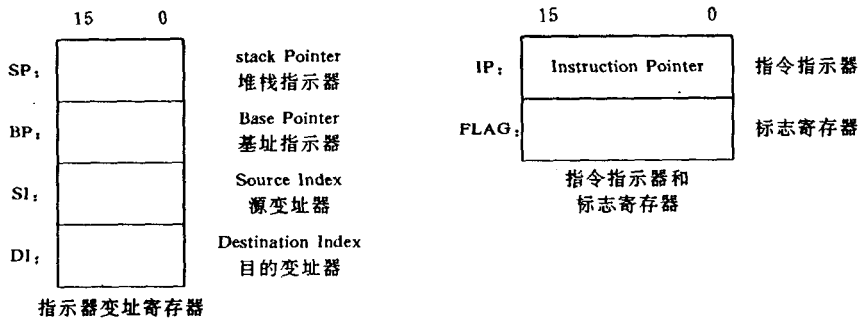


图 1.3 8086/8088CPU 寄存器

寄存器的隐含功能见表 1.1.

表 1.1 通用寄存器的隐含功能

寄存器名称	涉 及 的 操 作
AX	字乘法，字除法和字 I/O（字乘、字除中为被乘数，被除数及乘积的低 16 位、高 16 位在 DX 中）
AL	字节乘、字节除、字节 I/O，换码和十进制算术运算（被乘数、被除数及乘积的低 8 位，高 8 位在 AH 中）
AH	字节乘、字节除（积的高 8 位和余数）
BX	换码指令 XLAT 中的换码表基地址
CX	串操作和循环指令的计数
CL	作为移位计数器，用于移位次数大于 1 的移位指令
DX	字乘法积的高 16 位，字除法被除数的高 16 位，间接 I/O 的端口地址

二、指示器和变址寄存器组

对于访问存储器的指令，可以在指令中直接给出该单元的地址，但这会增加指令代码的长度，如果把经常使用单元的地址存放在特定寄存器中，访问这些存储单元的指令，就不必再含有该地址，而只要指定该寄存器即可，通常称这种寄存器为指示器或变址寄存器。

本组由四个 16 位寄存器组成，它们是串源变址寄存器 SI，串目的变址寄存器 DI，堆栈指示器 SP 及基址指示器 BP. 这些寄存器通常含有在某一段内寻址的偏移地址。

使用指示器和变址寄存器，除减少指令长度外，更重要的作用是：使指令访问的偏移地址是程序运算的结果。这个结果是指示器或变址寄存器与 16 位通用寄存器进行算术和逻辑运算得到的。

因四个寄存器在使用中的差别而把它们分为指示器 SP、BP 与变址寄存器 SI、DI.

SP 和段寄存器 SS 结合在 RAM 中建立堆栈，并通过栈顶来存取堆栈中的数据，而 BP 和 SS 结合则不通过栈顶存取堆 栈中的数据。通常 SI、DI 与段寄存器 DS 结合实现对数据段的存取，仅串操作指令时，DI 才与段寄存器 ES 结合实现对附加段的访问。

三、段寄存器组

本组有四个 16 位的段寄存器 CS、DS、SS 和 ES. 用来标识当前代码段、数据段、堆

栈段和附加段。其主要功能是使 8086/8088CPU 对 IBM PC 内存的不同段进行读写。

四、指令指示器 IP 和标志寄存器 FLAG

指令指示器亦称指令计数器或程序计数器，是 16 位寄存器。CPU 从硬 件上保证它总是自动加 1，这就使得指令能按顺序一条条地执行下去。当需要分枝或调 用时，要用专门的转移或转子指令修 改 IP，从而达到分枝或调 用的目的，由于 8086/8088 是分段存储程序和数据，IP 总是含 有当前代码段的偏移量，故依此偏移量可从代码段中取出指令送指令流字节队列以便执行。

标志寄存器 FLAG 也是 16 位的寄存器，但只用九位，其余位空闲。九个标志位分成控制标志位和状态标志位两组，见图 1.4. 状态标志位寄存着指令执行结果的状态信息，在任何给定的时刻，FLAG 可以向关心 8086/8088 结果状态的用户提供必要的信息。如运算结果的正负，等于零否、是否溢出，有否进位或借位等。这些状态信息给人们提供了处理问题的途径，控制标志位则对 CPU 的某些操作进行干预。

1. 状态标志位

系统执行算术运算、逻辑运算、移位、测试和比较指令时会影响状态标志位。而在执行数据传送和程序控制类指令时不影响状态标志位，下面仅就一般情况说明指令对状态标志位的影响，至于某一具体指令对状态标志位影响详见附录一.

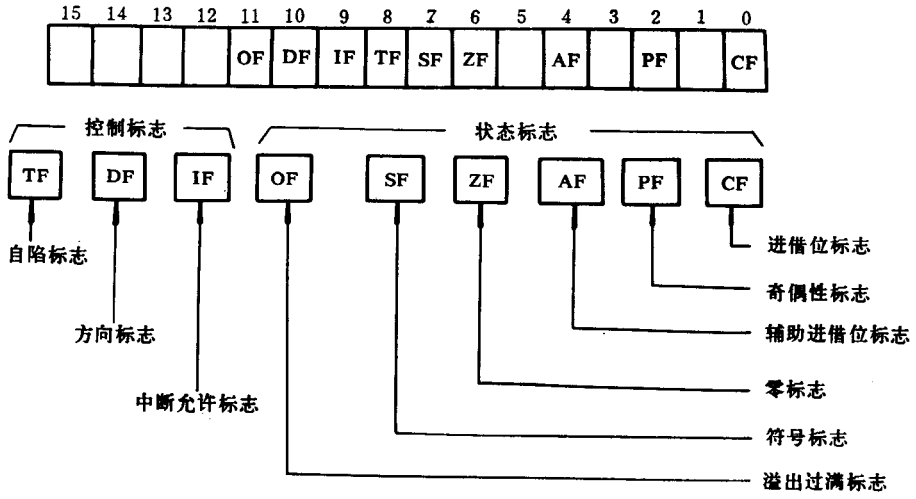


图 1.4 8086/8088 的标志寄存器

1）辅助进借位标志 AF（Auxiliary Carry Flag）

在字节操作由低半字节向高半字节进位或借位，或字操作低位字节向高位字节进位或借位时，AF＝1，否则 AF＝0. 这个标志位用于十进制调整指令。

2）进借位标志 CF（Carry Flag）

当结果的最高位（字节操作时的第七位或字操作时的第十五位）产生进位或借位时，CF＝1，否则 CF＝0. 该标志主要用于多字节数的加减运算。移位和循环移位指令也影响 CF，它们把操作数最后移出的位放入 CF 中，对于 AND、OR 和 XOR 逻辑操作，无条件

地将 CF 置 “0”.

CF 标志除了受指令执行结果的影响外，8086/8088CPU 还专门设置了三条指令改变 CF 的状态。它们是：

CLC (Clear Carry Flag)，即 CF←0.

STC (Set Carry Flag)，即 CF←1.

CMC (Complement Carry Flag) 即 CF 取反，若 CF＝1，则 CF＝0，否则 CF←1.

3) 溢出标志 OF (Overflow Flag)

对于有符号数的算术运算，若结果超出相应数据的范围，则置 OF＝1，表示溢出。以有符号整数为例，字节运算结果B不满足 $-128 \leqslant B \leqslant +127$，字运算结果W不满足 $-32768 \leqslant W \leqslant +32767$ 时，则 OF＝1，否则 OF＝0.

溢出和进位是两个不同性质的标志，千万不能混淆。

对于有符号数的算术运算，必须根据 OF 是 “1” 还是 “0” 决定溢出与否；而对于无符号数的算术运算，必须根据 CF 是 “1” 还是 “0” 决定是否溢出，此时，OF 不一定为 1，反之亦然。例如：字节 (−1) ＋ (−1) ＝−2.

```
        11 11 11 11     ; IBM PC 中负数是用补码表示的。
+       11 11 11 11
-----------------------
有进位→ [1] 11 11 11 10
```

第 7 位 (以后均简写为 D_7) 有进位，故运算后 CF＝1；因运算结果未超过字节有符号数的表示范围−128～+127，所以 OF＝0，这是 CF＝1 而 OF 不为 1 的情况。若把−1 看成无符号数 (255) 时，则就产生溢出了。又如 (+100) ＋ (+100) ＝+200.

```
  0110 0100
+ 0110 0100
-----------
 [0] 1100 1000
```

D_7 位无进位 CF＝0，但运算结果超过了+127，此时，OF＝1，这是 CF＝0 而 OF＝1 的情况。若要把上述两加数都看成无符号数 100 时，则就不产生溢出了 (因 CF＝0)．不管是有符号数还是无符号数，它们的运算结果都会同时去影响 OF 和 CF，因为 ALU 不会识别是有符号数运算还是无符号数运算，程序设计者自己应当清楚是哪种数据的运算，从而由 CF 或 OF 标志位去判断是否溢出。

4) 符号标志 SF (Sign Flag)

符号标志位与运算结果的最高位始终保持一致。即结果的最高位 (字节为 D_7，字为 D_{15}) 为 1 则 SF＝1，否则 SF＝0. 由于 IBM PC 系统中，有符号数是用补码表示的，所以 SF＝0 表示正数，SF＝1 表示负数，这就是称 SF 为符号标志位的原因。

5) 奇偶性标志 PF (Parity Flag)

若操作结果中二进制位 “1” 的个数为偶数，则 PF＝1，否则 PF＝0，这个标志通常用来检查在数据传送过程中是否发生错误。

6) 零标志 ZF (Zero Flag)

若运算或操作的结果为零则 ZF＝1，否则 ZF＝0.

2. 控制标志位

1）方向标志 DF（Direction Flag）

该标志位是通过两条专门的指令来确定其状态。若将 DF 置“1”，则使得串操作指令的源、目的地址自动减 1（字节串）或减 2（字串），也就是从高地址到低地址处理字符串；若 DF 为零，则串操作指令的源、目的地址自动增 1（字节串）或增 2（字串）。将 DF 置“1”的符号指令为 STD (Set Direction Flag)，将 DF 清“0”的符号指令为 CLD (Clear Direction Flag)。

2）中断允许标志 IF（Interrrupt enable Flag）

该标志是通过两条专门的指令确定其状态。若 IF＝1，则允许 CPU 去接受外部的可屏蔽中断请求，否则屏蔽即不接收上述的中断请求。将 IF 置“1”的符号指令为 STI (Set Interrupt enable Flag)。将 IF 清零的符号指令为 CLI（Clear Interrupt enable Flag）

3）自陷（追踪）标志 TF（Trap Flag）

若 TF＝1，使处理机进入单步执行方式，以便调试程序。在该方式下，CPU 每执行一条指令，产生一个内部中断，允许程序员检查执行的结果。该标志没有专门的指令直接改变它，要想改变它，只有借助于堆栈修改 TF，详见 2.3 节例 7.

下面介绍两条标志寄存器传送指令。这两条指令只涉及到标志寄存器的低 8 位。

LAHF（Load AH With Flags）

SAHF（Store AH into Flags）

前者的指令意义是将标志寄存器的低 8 位传送到 AH，后者的指令意义正好与前者相反，是将 AH 的内容传送到标志寄存器的低 8 位。

1.3 IBM PC 微机堆栈和内存储器

一、堆栈

IBM PC 微机的堆栈是在内存 RAM 中开辟的一端固定一端活动的存储空间。称活动端为栈顶，称固定端为栈底。当堆栈为空时，栈顶和栈底重合。堆栈中的数据遵循先进后出（后进先出）的原则存取。在堆栈存入或从堆栈取出数据一般是通过栈顶实现。堆栈指示器 SP 始终指向栈顶，且通过 SP 实现从栈顶存取数据。堆栈中的数据也可通过 BP 基址指示器进行存取，在这种情况下，则不是通过栈顶存取数据。

IBM PC 微机的堆栈结构如图 1.5 所示。栈的伸展方向是从大地址向小地址。进栈时 SP 的变化顺序 SP－1、SP－2、SP－3、……、SP－I、出栈时 SP 的变化顺序是 SP、SP＋1、SP＋2、……、SP＋I.

堆栈的设置主要用来解决多级中断、子程序嵌套和递归等程序设计中难以处理的实际问题。还可用来保存现场、寄存中间结果，并为主、子程序的转返提供强有力的依托。

二、内存储器

内存储器由 RAM (Random Access Memory) 和 EPROM (Erasible Programmable Read Only Memory) 或 ROM（Read Only Memory）构成，由于 8086/8088 有 20 条地址线，且存储

器是以字节为单位，故其直接寻址能力可达 1MB（2^{20}字节）。因此在 IBM PC 微机系统中，可以有多达 1MB 的存储器，地址从 00000H 到 FFFFFH。任意给定一个 20 位地址，就可以从存储器相应单元中取出所需要的指令或数据。如前所述，8086/8088CPU 内部的 EU 只能进行 16 位运算，与地址有关的寄存器 IP、SP、BP、SI、DI 等也都 是 16 位的。因而对地址的运算也只能是 16 位，即只能得到 16 位地址，而要访问存储器，就必须给出 20 位地址。如何由 16 位地址变换成 20 位地址呢？这就涉及到 16 位地址和 20 位地址间的转换，涉及到 IBM PC 内存储器的分段技术。

1. 为什么要对存储器进行分段

EU 只提供 16 位二进制数据作为地址，而存储器却需要 20 位二进制数据作为地址。为解决这一矛盾而提出存储器的分段技术。

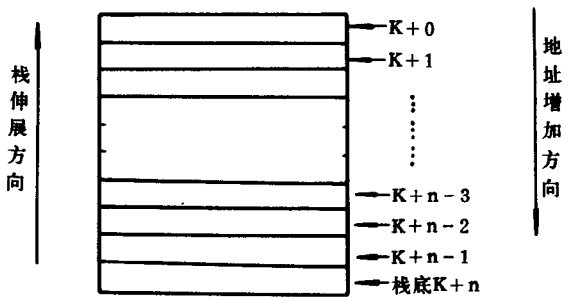


图 1.5　IBM PC 微机堆栈

2. 逻辑分段、物理分段及实现方法

依逻辑关系对源程序进行分段叫逻辑分段。对源程序可作如下形式的逻辑分段：存放全局数据的段；存放局部数据的段；存放特殊数据的段；作为堆栈的段；存放主程序的段；存放公共子程序的段；存放中断服务程序的段等等。依段寄存器对存储器进行分段叫物理分段。逻辑分段由段定义伪指令实现；物理分段是通过往 CS、DS、ES 和 SS 送段地址实现。物理分段的依据是逻辑分段。因物理段不得超过 64KB，故当逻辑段大于 64KB 时，必须分成两个逻辑段。段可分为四类：代码段，主要用来存放程序；堆栈段，专用来作为堆栈；数据段和附加段，均用来存放程序的各种数据。

3. 物理地址，逻辑地址及物理地址的形成

IBM PC 微机内存储器的每一存储单元都具有两种类型的地址，一个是物理地址，另一个是逻辑地址。所谓物理地址指的是 1MB 存储空间的某一实际单元的地址编号，用 20 位二进制数据表示。它是 0H～FFFFFH 之间的任一数值。当 CPU 和内存储器间交换数据时，采用的就是物理地址，物理地址和存储单元是一一对应的。所谓逻辑地址指的是编写源程序时使用的地址，它由段地址和偏移地址两部分构成，对于任一给定的单元（20 位二进制数作为地址），段地址用来指出包含该单元的段的首地址；偏移地址，亦称偏移量，它是一个无符号 16 位二进制表示的数，它指出由段首地址到该单元的距离，即该单元的物理地址与段首地址之差。如某存储单元的物理地址为 1FFAAH，段的起始地址为 1FF70H（段寄存器的内容应为 1FF7H，因为系统规定段寄存器的单位是 2^4），则偏移地址为：

```
  1 F F A A H
-1 F F 7 0 H
-------------
  0 0 0 3 A H
```

取 16 位二进制数且用十六进制表示时为 003AH，通常，都是知道逻辑地址，即段地址和偏移地址，再由逻辑地址求得物理地址：

物理地址＝段地址＋偏移地址

从逻辑地址到物理地址的转换是由 BIU 部件实现的，如图 1.6 所示。

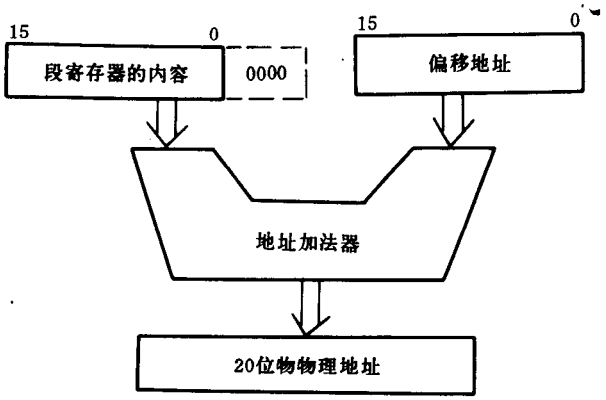


图 1.6 物理地址的形成

物理地址是逻辑地址经运算而得到的，因此弄清逻辑地址的来源是必要的。访问存储器的操作类型不同，逻辑地址的来源也不同。为了解逻辑地址的来源，先介绍当前段。在程序运行的任一时刻，可立即访问四个不同段：当前代码段，当前堆栈段，当前数据段和当前附加段。在通常情况下，当前代码段寄存着正在执行的程序。当前数据段寄存着当前代码段中的程序所需的数据，并为当前代码段中的程序准备了存结果的单元。当前堆栈段为当前代码段准备了堆栈操作所需的存储空间。当前附加段是通用区域，经常作为一个附加的数据段，为当前代码段寄存着某些特殊数据或准备着存某些特殊数据的单元。段寄存器和当前段的对应关系见表 1.2，逻辑地址的来源见表 1.3.

表 1.2 段寄存器和当前段的对应关系

段寄存器	含　义	隐含的当前段
CS	代码段寄存器	当前代码段
DS	数码段寄存器	当前数据段
SS	堆栈段寄存器	当前堆栈段
ES	附加段寄存器	当前附加段

指令总是取自当前代码段 CS，IP 作为偏移地址，即取指令时，逻辑地址为 CS 和 IP，且不允许使用段取代前缀操作。

堆栈指令总隐含着一个堆栈操作数，该操作数总被 存放在当前堆栈段，SP 的内容作为偏移地址，即堆栈操作数的逻辑地址为 SS 和 SP，且不允许使用段取代前缀操作。

串操作指令的目的串操作数必须存放在当前附加段，且由 DI 指出偏移地址。它也不允许使用段取代前缀操作，它的逻辑地址为 ES 和 DI.

串操作指令的源 串操作数通常要求存放到当前数据段，且由 SI 指出偏移地址。它允许用户将它存放在其它当前段，但使用时必须通过段取代前缀给于说明，否则其将仍然