IBM PC汇编语言程序设计和接口技术

王永山

IBM PC 汇编语言程序设计和接口技术

王永山 编

西安电子科技大学出版社
1991

内 容 简 介

本书内容分两大部分．前三章是 IBM PC／XT 微机系统的汇编语言编程方法．在介绍了 8086／8088 微处理器指令系统基础上，讨论了单模块程序结构的特点和多模块程序编程的特殊问题；紧密结合 IBM PC／XT 微机系统特点，讨论了汇编语言编程时，调用 BIOS 和 DOS 系统子程序的方法．后七章详细讨论了 IBM PC／XT 机的接口技术．讨论侧重两个方面：在汇编语言(有些部分还介绍了 BASIC 语言有关语句)级上如何利用系统已有的接口资源和如何开发出特殊需要的新的接口．

在编写指导思想上，编者特别重视书的工程实用性，力求叙述通俗易懂．本书是在《微机在数据采集和控制系统中的应用技术》课(为非计算机专业研究生开设)的讲稿基础上整理编成的．适用于已有微机基本知识的高年级大学生、研究生，亦可供使用 IBM PC／XT 机的工程技术人员学习和参考．

IBM PC 汇编语言程序设计
和接口技术

王永山　编

责任编辑　殷咸安

西安电子科技大学出版社出版发行

西安电子科技大学印刷厂印刷

新华书店经销

开本 787×1092　1／16　印张 16.8／16　字数 391 千字

1989 年 6 月第 1 版　1991 年 5 月第 3 次印刷　印数 7 001—12 000

ISBN 7－5606－0105－7／TP・0039(课)　　定价：1.35 元

前　言

微型计算机用在数据采集或控制系统中的时候，必须解决接口问题，而且编应用程序时，总是部分地或全部地使用汇编语言．本书的目的是提供在汇编语言级上利用和开发IBM PC(或它的兼容机)微机输入输出接口的知识和技术．本书是在为电子学学科非计算机专业研究生开设的《微机在数据采集和控制系统中的应用技术》课的讲稿基础上整理编写而成的．所以，学习本书的读者需要有一定的微机基本知识．

在指导思想上，编者特别重视两点：一点是它的实用性，希望它起到"手册"或"指南"的作用；另一点是便于自学，编者在这方面所做的努力是把广泛流传的但难以理解的译文资料，尽量找到原文，并根据自己的理解进行叙述．例如，把 DOS 功能调用根据用途分类，分散在内容相关的章节讨论．当然，这是冒风险的，因为自己的理解和叙述难免不准确，甚至是错误的．

从内容上本书包括两大部分：前三章是关于汇编语言程序设计，既是独立部分，又为后面部分打基础．后七章是关于输入输出接口的开发技术．

第一章从汇编语言程序设计的需要出发，简要介绍 8088／8086 微处理器，并且尽早地概括出这种微处理器汇编语言程序结构的基本特点．为此，介绍了某些关键性的伪指令，为第二、三章讨论扫清障碍．

第二章除简要叙述 8088／8086 的指令系统外，还给出了足以理解程序结构和掌握编程方法的例子．把系统软件提供给用户的输入输出子程序调用方法放在本章讨论，不仅能早些触及包括输入输出功能的编程方法，而且能尽早上机实习．

第三章把编程扩展到多模块的范围．其关键问题是模块之间的连接和交叉访问．从实用出发，本章介绍了 BASIC 语言程序调用机器码子程序的方法．除本章之外，第一、六、八章也涉及到 BASIC 语言的编程问题．

第四章开始讨论输入输出接口开发技术．第四章讨论了与后几章都有关的问题，例如系统级输入输出总线和地址的分配与译码等问题．

从第五章到第九章分别讨论了最常使用的并行接口、串行接口、定时／计数器应用、外中断控制系统和 DMA 传送控制系统．对它们的讨论都着眼于两个方面：如何利用系统提供的接口资源和如何开发出新的接口．编者在这些内容上下了较多的功夫．

第十章是关于用汇编语言编程控制磁盘操作，当然不是指机械操作，而是关系到磁盘文件的各种问题．

本书适用于正在从事 IBM PC 系统接口和应用软件开发的高年级大学生、研究生和科技工作者．

由于编者水平的限制，加之时间仓促，定会有不少缺点和错误，敬请读者批评指正．

编　者

198-

目　录

第一章　8088／8086 编程模型和汇编语言基础

8088 与 8086 有相同的指令系统，在软件上是兼容的。在内部结构上，从本质上说也是相同的，只是在与外界联系的引脚安排上稍有差别：8086 有 16 位数据总线，与地址总线中的 A_0～A_{15} 兼用，所以是 16 位微处理器；而 8088 只有 8 位数据总线，与地址总线的 A_0～A_7 兼用，所以是 8 位微处理器。此外，这两种微处理器的引脚 28 和 34 上的信号形式和功能，也稍有不同，对其感兴趣的读者可参阅有关手册。两种微处理器内部寄存器的位数相同，许多操作都是 16 位的。有些指令需要在数据总线上传送 16 位数，8086 一次完成，8088 两次完成，但仍是一条指令，所以并没有因为数据总线的位数不同影响它们的指令系统。

8088／8086 与 Z80 等微处理器相比，先进之处之一是取指令操作和执行指令操作可以并行进行。就是说，如果指令规定的操作不占用数据总线，这时就可以通过数据总线从存贮器中依次取出下条指令的指令码。与这种功能相适应，8088／8086 内部有按先进先出规则排列起来的多个字节的指令缓冲寄存器(称为指令栈)，代替 Z80 等微处理器中单字节的指令寄存器。8086 中这个指令缓冲寄存器有 6 个字节，8088 的是 4 个字节，这是它们的差别之一。

本章的内容实际上是第二、三章的预备知识。§ 1.1 节介绍 8088／8086 的硬件背景。§ 1.2 节说明 8088／8086 与其它 8 位微处理器的一个突出差别，即存贮器空间和 I／O 地址空间。8088／8086 汇编语言与其它 8 位微处理器汇编语言的许多差别都来源于存贮器的编址方式不同。§ 1.3 节介绍的 MASM 汇编语言基础是非常重要的一节。本节对 MASM 汇编语言程序的结构，编程中最常用到的变量、标号及有关的伪指令、操作符做了介绍，为第二、三章的学习扫清了障碍。建议读者在以下各章学习过程中能经常复习本章。

§ 1.1　8088／8086 的程序设计模型

一、8088／8086 的寄存器结构

图 1.1 是 8088 的内部结构框图。从图中可以看出，其它微处理器(例如 Z80)中的指令寄存器在 8088 中被一个由 4 个字节(8086 是 6 个字节)组成的指令栈(先进先出结构)所代替，从而大大提高了程序的运行速度。这是因为，在 8088／8086 内部，指令的执行部分和总线接口部分是并行工作的。总线接口部分负责控制存贮器的读／写。并行工作方式使得在执行指令的同时(只要不是访问存贮器和 I／O 端口的指令)，总线接口部分可从存贮器依次取指令码入指令栈。这样一来，在大多数情况下，指令执行部分就不再花费时间等待取指令的操作。

除了指令栈之外，8088／8086 内起控制和工作寄存器作用的寄存器可以按功能分为三组。

1. 数据寄存器组　它包括 AX、BX、CX 和 DX，它们都是 16 位的，在指令执行过

程中，既可以用来寄存操作数，也可以用来存放操作的结果。在多数情况下，特别是在算术、逻辑指令中，它们没有固定的应用，但在有些指令中，要求某个或某几个寄存器有特定的应用。例如，在LOOP(循环)指令中，CX寄存器固定用于循环次数控制，每执行一次LOOP指令，CX内容减1，如果差值不为0，则循环；反之，差值为0，则执行相邻的下条指令。此外，在乘、除指令和字符串操作指令中，还规定了几个寄存器的特定作用。

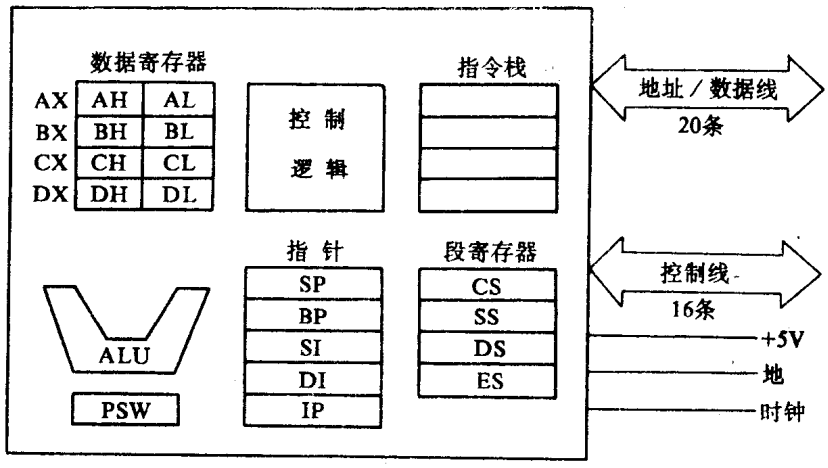

图 1.1　8088 内部结构框图

这四个寄存器中的每一个，既可以作为16位寄存器使用，也可以作为两个8位寄存器使用。这样就可以看作是八个独立的8位寄存器。AH、BH、CH、DH和AL、BL、CL、DL分别是寄存器AX、BX、CX和DX的高8位和低8位组成的独立寄存器。

2. 指针和变址寄存器组　它包括IP、SP、BP、SI和DI。这组寄存器在功能上的共同点是，在访问内存时，用于形成20位物理地址码的组成部分。在任何情况下，它们都不能独立地形成访问内存的地址码，因为它们都只有16位。要形成20位的地址码，它们必须和一个段寄存器的内容相加。后面我们将说明段寄存器和地址形成的方法。

IP(Instruction Pointer)指令指针。其作用与其它微处理器中的程序计数器PC相似。不同的是，它的内容要和代码段寄存器CS的内容相加才能形成取指令的地址。

SP(Stack Pointer)堆栈指针。其作用是，它的内容与堆栈段寄存器SS的内容相加提供堆栈操作(压入或弹出)的地址。

BP(Base Pointer)基址指针。在某些寻址方式中，BP的内容构成段内偏移地址的一部分，后面讨论寻址方式时将进一步说明。特别值得注意的是，凡偏移地址中含有BP内容时，如果不加特别说明，其段地址由堆栈段寄存器SS提供。就是说，这条指令是对堆栈区的存贮单元操作。

SI(Source Index)和DI(Destination Index)是两个变址寄存器。在有变址的寻址方式中，其内容作为段内偏移地址的组成部分。两个寄存器的第一个字母S和D分别含有源和目的的意思。在某些指令中，包含两个地址，有源地址和目的地址之分。这两个寄存器常常被指令隐含指定为提供源地址和目的地址的偏移量。也有些指令，虽然只含一个地址，但却隐含规定这两个寄存器之一提供偏移地址。这种情况多数出现在字符串处理指令中。采用变址寻址方式，且自动修改变址寄存器内容，可以很方便地访问地址连续的存贮

单元。例如，指令 STOS(存贮字符串)，源地址(字操作时为 AX，字节操作时为 AL)和目的地址(存贮单元)都是隐含的。其操作是，AX(字操作)或 AL(字节操作)的内容存贮于由 DI 的内容作为段内偏移地址、由 ES(附加段寄存器)内容作为段地址形成的存贮器地址单元中(字节操作时为一个单元，字操作时为相邻两个单元)，并且自动修改 DI 的内容——字节操作时，DI+1→DI；字操作时，DI+2→DI。自动修改变址寄存器内容，为重复执行 STOS 指令提供了方便的寻址方式。

3. 段寄存器组　设置段寄存器的必要性来自于 8088／8086 可访问的存贮器空间为 1 兆字节。1 兆字节存贮单元要由 20 位地址码编址，但前面介绍的指针或变址寄存器只有 16 位，如何形成 20 位物理地址呢？任何访问存贮器所需要的 20 位地址，都是由图 1.2 所示意的操作提供的。图中的段地址就是由段寄存器提供的。有四个段寄存器，它们是 CS(Code Segment)代码段寄存器、SS(Stack Segment)堆栈段寄存器、DS(Data Segment)数据段寄存器和 ES(Extra Segment)附加段寄存器。各段寄存器的用途在§1.2 节讨论。

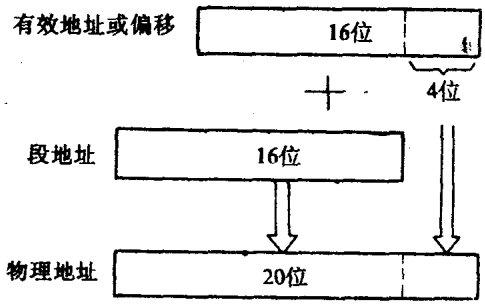

图 1.2　20 位物理地址形成示意图

其它微处理器中有状态寄存器，而在 8088／8086 中起同样作用的寄存器称为标志寄存器或处理器状态字(PSW)寄存器。它也是 16 位的，但只利用了其中的 9 位。其中的每一位都称为一种标志位，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

这些标志位按功能可分为两类：状态标志，反映刚刚完成的操作结果情况；控制标志，在某些指令操作时起控制作用。

属于状态标志位的有：

SF(Sign Flag)，它总是与操作结果的最高位相同，因为在补码运算时最高位是符号位：为 1，代表结果为负；为 0，代表结果为正。

ZF(Zero Flag)，结果各位都为 0，则该标志位被置为 1；否则为 0。

PF(Parity Flag)，如果结果的低 8 位中 1 的个数为偶数，则该标志位为 1；否则为 0。

CF(Carry Flag)，加法时最高位有进位输出或减法时最高位需要借位，该标志位置 1；否则置 0。

AF(Auxiliary Carry Flag)，加法时位 3 有进位加入位 4 或减法时位 3 需从位 4 借位，则该标志位置 1；否则置 0。这个标志位用于实现 BCD 算术运算结果的调整。

OF(Overflow Flag)，溢出标志位，与其它微处理器中的溢出标志位相同。在加或减运算的结果超过了寄存器所能表示的数值范围时，出现溢出。具体说，对于加运算，如果次高位形成了进位加入最高位，而最高位相加时(包括次高位的进位)却没有进位输出；或者反过来，次高位没有进位加入最高位，但最高位却有进位输出，都将使 OF 位置 1。因为这两种情况分别是：两个正数相加，结果的正数和超过范围(形式上变成了负数)，两个

负数相加，结果的负数和超过范围(形式上变成了正数)。对于减运算，当次高位不需从最高位借位，但最高位却需借位(正数减去负数，差超过范围)；或反过来，次高位需从最高位借位，但最高位不需借位(负数减去正数，差超过范围)，也会使 OF 位置 1。

以上所述各状态标志位的置 1 置 0 条件是一般规则，各种指令对状态标志位的影响情况请参阅 § 2.1 节对各种指令的说明。

属于控制标志位的有：

DF(Direction Flag)，字符串处理指令执行时受它的控制：如果该位已先置 0，则指令执行时存贮器的地址(该类指令都含有访问存贮器连续地址的操作)是从低地址向高地址方向发展；反之，如果该位先置 1，则从高地址向低地址方向发展。

IF(Interrupt Enable Flag)，如果该位已被置 1，处理器能响应可屏蔽中断请求；否则，不能响应可屏蔽中断请求。

TF(Trap Flag)，该位被置 1，使处理器进入单指令工作方式。每条指令开始之前总要测试 TF 位是否为 1。如果为 1，不是立即，而是在本指令执行之后，自动进入类型 1 中断，即转而执行陷阱服务子程序。该子程序的开始地址由 00004H～00007H 四个存贮单元提供。由于进入中断时的操作序列包括把处理器状态字 PSW 压入堆栈，继而又清除 TF 位，所以在执行陷阱服务子程序期间不会再进入类型 1 中断，必须在陷阱子程序的最后一条指令 IRET 执行之后，从堆栈中弹出 PSW，使 TF 恢复为 1，下一条指令之后才能再次进入陷阱中断。可以说陷阱中断是处理器硬件的功能，但服务子程序，对于不同的微机系统可以是不同的。在 IBM PC 系统中，用系统软件 DEBUG 调试程序时，T 命令的执行就是利用了陷阱中断，服务子程序的功能是显示所有寄存器的当前值和将要执行的下一条指令。

控制标志位有一个共同的特点，就是对每一位都有独立的置 0 或置 1 指令。

标志寄存器可以作为一个整体(16 位)压入堆栈或从堆栈中弹出。如果根据需要选定各位的值先置于某个寄存器，然后压入堆栈，最后再从堆栈中弹出到 PSW 寄存器中，用这样的方法可以任意重新设置 PSW 各位的值。PSW 寄存器的低 8 位(包括 SF、ZF、AF、PF 和 CF)还可以用指令与 AH 互相传送。显然，使用这种传送指令也可测试或设置这些位的值。

二、8088 的输入输出引脚

图 1.3 是 8088 与外界连接的引脚图。40 条输入输出引线按功能可分为地址线、数据线、控制和状态线、定时信号、电源及地线。AD_0～AD_7 经过图 1.4 所示的逻辑控制关系，既形成地址总线的低 8 位 A_0～A_7，又形成 8 位数据总线 D_0～D_7。实现这一点的基础是在时间上地址码的低 8 位出现在前，而数据出现在后。A_8～A_{19} 是地址总线的高 12 位。

控制信号线中，IO／$\overline{M}$是输出信号，用来区别操作是访问存贮器单元(信号为低)还是访问输入输出端口(信号为高)。它与$\overline{RD}$、$\overline{WR}$相配合，控制着存贮器读写和 I／O 端口读写。DT／$\overline{R}$也是输出信号，在执行使数据从处理器向外传送的操作时，该信号为高电平输出；反之，执行处理器从外界接收数据的操作时，该信号为低电平输出。显然，这个信号用于加在总线上的三态缓冲器上，控制传送方向。$\overline{DEN}$也是输出信号，用于加在总线

上的三态缓冲器上，控制总线的接通(低电平)和断开(高电平)。因此，数据出现在数据总线上期间，$\overline{DEN}$输出低电平。ALE 信号的功能可从图 1.4 中看出，它出现期间恰好能保证把 AD_0～AD_7 的地址码锁存于地址锁存器中。INTR 是可屏蔽中断请求输入端。NMI 是非可屏蔽中断请求输入端。$\overline{INTA}$是处理器发向中断控制器的中断响应信号，控制中断周期的操作执行。HOLD 和 HLDA 用来控制 DMA 传送，HOLD 是 DMA 请求输入端，HLDA 是处理器允许 DMA 传送的响应信号输出端。$\overline{SSO}$是状态输出引端，它与 IO／$\overline{M}$、DT／$\overline{R}$上的信号组合起来表明总线所处的操作状态，如表 1.1 所示。RESET 信号用于系统的启动和再启动。外逻辑电路产生的启动信号一方面经过这一引端加入处理器，另一方面加到外围设备。在处理器内部，RESET 信号使 DS、ES、SS、IP 和 PSW

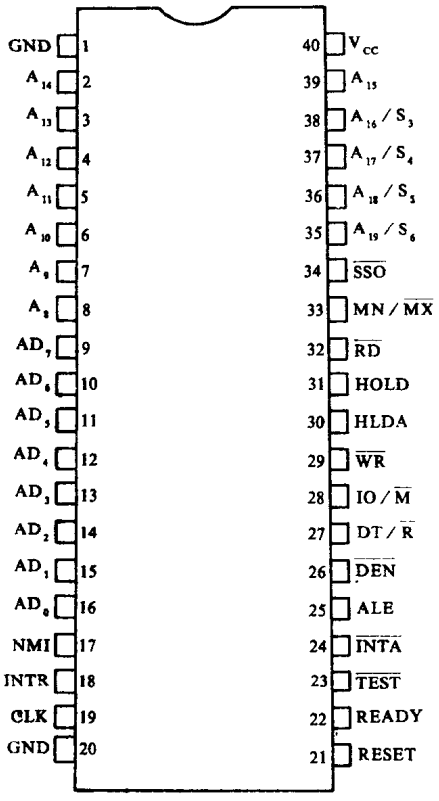

图 1.3　8088 输入输出引脚图

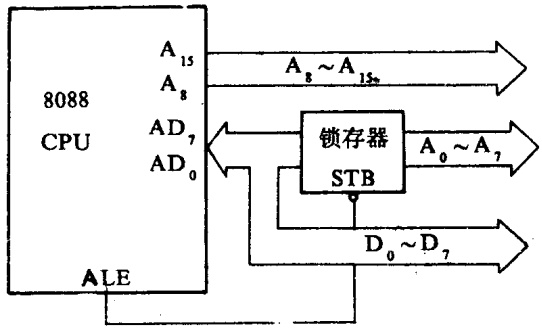

图 1.4　按时间顺序把地址和数据线分开

表 1.1　IO／$\overline{M}$、DT／$\overline{R}$和$\overline{SSO}$组合规定的状态

IO／$\overline{M}$	DT／$\overline{R}$	$\overline{SSO}$	操作状态
1	0	0	中断响应
1	0	1	I／O 端口读
1	1	0	I／O 端口写
1	1	1	暂停
0	0	0	取指令码
0	0	1	存贮器读
0	1	0	存贮器写
0	1	1	无作用

寄存器清除为 0，使 CS 寄存器置入 FFFFH。READY 是输入信号端，当加入低电平时，可以增加机器周期内的等待状态个数，达到延长时间目的，所以它可用作慢速存贮器或 I／O 设备与处理器同步的控制端。$\overline{TEST}$也是用于使处理器与外围设备同步操作的控制端，但它必须与 WAIT 指令配合工作：当处理器执行 WAIT 指令后，处理器便处于等待状态，直到外部在$\overline{TEST}$端加入低电平，处理器才继续执行下条指令。

定时信号只有 CLK 一个输入引端，用于输入系统时钟信号。IBM PC 系统中这一引

端输入 4.77 MHz 脉冲信号。

V_{cc} 是唯一的+5 V 电源输入端。两个 GND 是接地引端。

§ 1.2 8088 / 8086 的存贮器地址空间、I / O 地址空间和寻址方式

对于只学过 Z80 或 6502 微处理器的读者来说，8088 / 8086 操作涉及的存贮器的段是一个全新的概念。它的指令系统和汇编语言程序设计的许多特点都是根据存贮器段的划分而产生的，所以必须对存贮器的段、段地址及段内偏移地址建立清楚的概念，才能理解各种寻址方式。

一、存贮器的段、段地址和段内偏移地址

如图 1.2 所示，无论是为取指令还是为执行指令而访问内存时所需要的 20 位物理地址，都是由段地址和段内偏移地址相加产生的。分析图 1.2 我们可以得到以下几点：

1. 如果偏移地址为 0，这时的 20 位物理地址便是一个段的开始地址。段的开始地址的特点是一定能被 16 整除，因为它实际上是段寄存器的 16 位内容左移 4 位(即乘以 16)形成的。今后，我们把 20 位的段开始地址除以 16 得到的 16 位数，即段寄存器内容称为段地址。显然，在 1 兆字节的存贮器空间中，可以有 2^{16} 个段地址，任意相邻的两个段地址相距 16 个存贮单元。

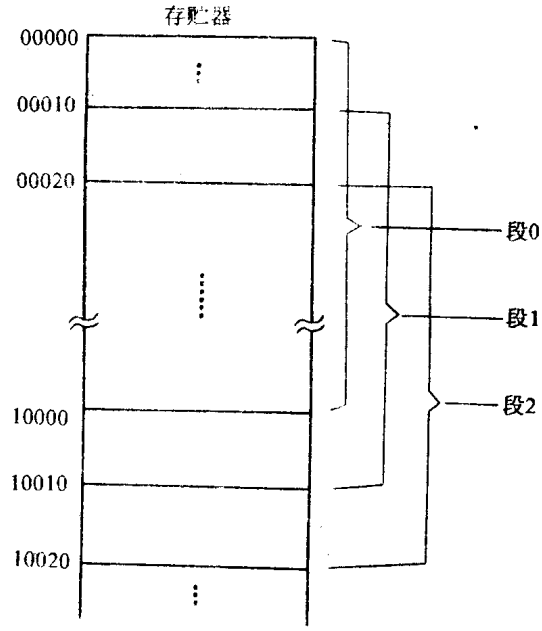

图 1.5 存贮器段的覆盖示意图

2. 任一段内的偏移地址都是 16 位的，所以最大可以包括一个 64 k 单元的存贮器空间。由于相邻两段地址只相距 16 个单元而不是 64 k 单元，所以段与段之间是互相覆盖的，如图 1.5 所示。

3. 段地址总是由段寄存器提供的，而偏移地址，有时也称有效地址(EA)，是由指令指针 IP(在取指操作时)或指令中规定的寻址方式(执行指令时)提供的。

二、信息的分段存贮与段寄存器的关系

段寄存器的利用不仅使存贮器地址空间扩大到 1 兆字节，而且为信息按特征分段存贮带来了方便。存贮器中的信息可以分为程序、数据和计算机的状态等信息。为了操作的方便，存贮器可以相应地划分为：程序区，该区存贮程序的指令代码；数据区，它存贮原始数据、中间结果和最后结果；堆栈区，用以存贮需要压入堆栈的数据或状态信息。段寄存

器的分工是：代码段寄存器 CS 划定并控制着程序区；数据段寄存器 DS 和附加段寄存器 ES 控制着数据区；而堆栈段寄存器 SS 对应着堆栈存贮区。表 1.2 列出了各种类型访问存贮器时所要使用的段寄存器和段内偏移地址的来源，它规定了为各种目的访问存贮器时形成 20 位物理地址的大的原则。这个表的要点如下：

表 1.2　各种类型访问存贮器时的地址成分

访问存贮器类型	默认段地址	可指定段地址	段内偏移地址来源
取指令码	CS	无	IP
堆栈操作	SS	无	SP
字符串操作源地址	DS	CS、ES、SS	SI
字符串操作目的地址	ES	无	DI
BP 用作基址寄存器时	SS	CS、DS、ES	依寻址方式求得有效地址
一般数据存取	DS	CS、ES、SS	依寻址方式求得有效地址

1. 任何类型访问存贮器时，其段地址要么由"默认"段寄存器提供，要么由"指定"的段寄存器提供。所谓默认段寄存器是指在指令中不用专门的信息指定另外一个段寄存器的情况，这时就由默认段寄存器提供访问内存的段地址。实际程序设计时，绝大多数属于这种情况，因此要熟记各种类型访问内存时的默认段寄存器。有几种类型访问存贮器允许指定另外的段寄存器，这为访问不同的存贮器段提供了灵活性。段寄存器的指定是靠在指令码中增加一个字节的前缀实现的。有些类型访问存贮器不允许指定另外的段寄存器，它们是：为取指令码访问内存时，段寄存器一定是 CS；堆栈操作时，段寄存器一定是 SS；字符串处理指令的目的地址，其段地址寄存器一定是 ES。

2. 段寄存器 DS、ES 和 SS 的内容是用传送指令置入的，但任何传送型指令不能向段寄存器 CS 中置入数。后面将讲到，ASSUME 伪指令及 JMP、CALL、RET、INT 和 IRET 等指令可以设置和影响 CS 的内容。更改段寄存器的内容意味着存贮区的移动。这说明无论程序区、数据区还是堆栈区都可以不限于 64 k 字节的容量，都可以利用重置段寄存器内容的方法予以扩大，而且各存贮区都可在存贮器中浮动。

3. 表中"段内偏移地址来源"一栏指明，除了两种类型访问存贮器是"依寻址方式求得有效地址"外，其它都指明了一个 16 位的指针寄存器或变址寄存器。如取指令访问内存时，段内偏移地址只由指令指针 IP 提供；堆栈的压入、弹出操作时，段内偏移地址只由 SP 提供；字符串操作时，源地址和目的地址中的段内偏移地址分别由 SI 和 DI 提供。除此以外的为存取操作数而访问内存时，段内偏移地址将由下面讨论的指令码规定的寻址方式求得。

三、访问存贮器的寻址方式

表 1.2 中虽然只有两项其段内偏移地址是依寻址方式求得有效地址，但在实际程序设计时，这却是多数情况。8088 / 8086 的寻址方式分两种情况：为获得操作数的寻址方式和为获得转移地址的寻址方式。

1. 获得操作数的寻址方式

(1) 立即数　操作数就在指令码之内，可能是一个字节，也可能是两个字节.

(2) 直接寻址　16 位有效地址码，即段内偏移地址在指令码之内，占两个字节.

(3) 寄存器寻址　操作数在一个寄存器之内，而这个寄存器以 3 位代码形式包含在指令码之中.

(4) 寄存器间接寻址　操作数在存贮单元中，而存贮单元的段内偏移地址在基址寄存器 BX 或变址寄存器 SI 或 DI 中，指令码中要指明 BX、DI 或 SI. 也就是说有效地址 EA 可能有三种情况:

$$EA=\begin{bmatrix}(BX)\\(SI)\\(DI)\end{bmatrix}$$

(5) 寄存器相对寻址　操作数在存贮单元中，其有效地址是一个 8 位的或 16 位的位移量(以后都用 disp 表示)与一个基址寄存器或变址寄存器的内容之和. 位移量 disp 和这个寄存器要在指令码中给出. 可表示为

$$EA=\begin{bmatrix}(BX)\\(BP)\\(SI)\\(DI)\end{bmatrix}+\begin{bmatrix}8位disp\\16位disp\end{bmatrix}$$

(6) 基址且变址寻址　操作数的有效地址是一个基址寄存器和一个变址寄存器的内容之和. 基址寄存器和变址寄存器要在指令码中指明. 可表示为

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

(7) 基址、变址且相对寻址　操作数的有效地址是一个 8 位或 16 位的位移量 disp、一个基址寄存器内容和一个变址寄存器内容三部分之和，即

$$EA=\begin{bmatrix}(BX)\\(BP)\end{bmatrix}+\begin{bmatrix}(SI)\\(DI)\end{bmatrix}+\begin{bmatrix}8位disp\\16位disp\end{bmatrix}$$

(8) 隐含寻址　有些指令的指令码中不包含指明操作数地址的部分，而其操作码本身隐含指明了操作数地址. 例如，乘法指令 MUL 的指令码中只需指明一个乘数的地址，另一个乘数和积的地址是隐含固定的. 除法指令 DIV 和其它指令也有这种情况.

上述寻址方式各适用于哪些指令，它们在指令码中是如何编码的，请读者参阅 § 2.1 节说明和附录 A.

在汇编语言编程时，重要的是这些寻址方式如何书写. 纵观各类指令，有单操作数指令和双操作数指令两种. 双操作数指令又分下列情况:

(1) 一个是立即数，另一个是寄存器或存贮单元;

(2) 一个是寄存器，另一个是寄存器或存贮单元. 既然是两个操作数，就包含两种寻址方式，但不可能在同一条指令中包含两个都指明存贮单元的寻址方式(这里不包括表 1.2

中给出的隐含两个存贮器地址的情况)。有些两个操作数指令中，其中一个是寄存器，是隐含的，书写时就与只有单操作数指令一样。这样一来，就书写而言，可简化为如何书写立即数和位移量、寄存器寻址及存贮单元的寻址问题了。

立即数和位移量都可用数字、已赋值(用 EQU 或 = 伪指令赋值)的变量名或表达式表示。写数字时，以 B 结尾表示二进制数，以 H 结尾表示十六进制数，无字母结尾表示十进制数。

下面是各种寻址方式时操作数的书写方法。

(1) 立即数　如:

```
00101010B
539
0A9H
'AB'; 表示字母 A 和 B 的 ASCII 码
XX-YY+5; XX、YY 都是已赋值的常数
```

表示十六进制数时，若最高有效位是 A～F，书写时需在前面加一个 0，以免当作字符串变量。

(2) 直接寻址　书写成变量名或包括变量名的表达式。如:

```
WGT
CNT-5
ARRY+5
```

变量名，如 WGT、CNT 和 ARRY 必须是用数据定义伪指令(DB、DW 或 DD)定义过的，除指明了操作数是字型还是字节型的类型信息之外，还指明了操作数的地址(在汇编过程中可以计算出来)。

(3) 寄存器寻址　写出寄存器相应的代表字符。如:

```
AX
DH
```

(4) 寄存器间接寻址　书写时在括号内写入寄存器代表字符。如:

```
[BX]
```

(5) 寄存器相对寻址　书写时可以是

变量名[寄存器符 ± 常数表达式]

或　　[寄存器符 ± 常数表达式]

如:

```
VAR _X[ BX ]
VAR _X[ SI+10H ]
[ BX-1 ]
```

其中 VAR _ X 是变量名，汇编过程中所算得它在定义段内的偏移地址可以看作常数，与另外常数，如上例中的 10 H 相加产生一个新的常数，这就是指令码中的 disp 部分。

(6) 基址且变址　书写为

[基址寄存器][变址寄存器]

如:

[BP][SI]

(7) 基址变址且相对　书写为

变量名[基址寄存器±常数][变址寄存器±常数]

或　　[基址寄存器±常数][变址寄存器±常数]

如:

VAR_Y[BX+5][SI-2]

VAR_Y[BX][SI]

[BP+2][DI-9]

汇编后变量在定义段内的偏移地址作为常数与其它常数相加，作为指令码中的 disp 部分。

2. 获得转移地址的寻址方式

(1) 段内相对寻址　指令码中包括一个位移量 disp，转移的有效地址为(IP)+disp. disp 可以是 16 位的，也可以是 8 位的. 如果 disp 是 8 位的，称为段内短相对寻址。无论是 8 位还是 16 位，disp 在指令码中都是用补码表示的有正负符号的数.

(2) 段间直接寻址　指令码中直接给出 16 位的段地址和 16 位的段内偏移地址。

(3) 段内间接寻址　在同一代码段内，要转移到的地址的 16 位段内偏移地址要么在一个 16 位寄存器里，要么在存贮器相邻两个单元中. 这个寄存器或相邻单元的第一个单元的地址，是在指令码中以上面讨论的存取操作数的寻址方式给出的，只不过寻址方式决定的地址里存的不是操作数而是转移的地址.

(4) 段间间接寻址　和段内间接寻址相似，但不可能有寄存器间接寻址了，因为要求得的转移地址是 32 位. 指令中一定给出某种访问内存单元的寻址方式. 用这种寻址方式计算出的存贮单元地址开始的连续四个单元，其中的内容就是要转移到的地址. 其中前两个单元内的 16 位值是段内偏移地址，后两个单元内的 16 位值是段地址.

属于转移类指令的有转子程序指令 CALL、无条件转移指令 JMP 和多种条件转移指令. 并不是每种指令都具有上述四种寻址方式. 各种指令有哪些寻址方式和如何书写，将在§2.1 节结合指令功能说明.

中断指令 INT 的寻址方式很特殊，将在§2.2 节说明.

四、I/O 地址空间和寻址方式

I/O 指令访问的是 I/O 地址，每一个 I/O 地址又称为一个 I/O 端口. 根据需要，一个外部设备可能占用一个或多个 I/O 端口. 8088 微处理器中，虽然 I/O 地址线和存贮器地址线是公用的，但如表 1.1 所示，IO/$\overline{M}$、DT/$\overline{R}$及$\overline{SSO}$的不同组合可以区别开是访问存贮器还是访问 I/O 端口. 再加上$\overline{WR}$、$\overline{RD}$等信号经过译码，在系统总线上形成存贮器写$\overline{MEMW}$、存贮器读$\overline{MEMR}$、输入输出写$\overline{IOW}$和输入输出读$\overline{IOR}$信号，控制存贮器读写和 I/O 端口读写. 本来 8088 地址总线的低 16 位都可以用作 I/O 端口地址线，即最多可有 2^{16} 个端口地址，但 IBM PC 系统只使用 $A_9 \sim A_0$ 十条地址线为 I/O 端口编址，所以最多有 $2^{10}=1024$ 个端口地址.

I/O 端口的寻址有直接和间接两种方式.

1. 端口的直接寻址　当端口地址数在 0～255 范围内时可以利用这种方式. 书写格式

为:

```
IN  AL，常数表达式
OUT 常数表达式，AL
```

其中常数表达式为端口地址数．例如:

```
IN   AL，PORT1
OUT  PORT1＋1，AL
```

这里的 PORT1 是用 EQU 或＝伪指令赋值的常数．当然也可以把端口地址直接写为数字常数．例如:

```
IN   AL，200
OUT  200，AL
```

汇编后的指令码中，端口地址占一个字节，所以当端口地址大于 255 时，不能用直接寻址方式，而必须用寄存器间接寻址方式．

2. 端口的寄存器间接寻址　间址寄存器固定用 DX．书写格式为:

```
IN   AL，DX
OUT  DX，AL
```

显然，在输入输出指令之前，必须把端口地址送入 DX．例如:

```
MOV  DX，PORT1
IN   AL，DX
MOV  DX，PORT2
OUT  DX，AL
```

其中 PORT1 和 PORT2 是表示端口地址的常数．

I／O 指令的另一个操作数地址是寄存器 AL 或 AX．在输入输出 8 位数时用 AL；输入输出 16 位数时用 AX．16 位数的输入输出是靠两次 8 位数传送实现的：第一次传送低 8 位，是指令中指明的端口地址；第二次传送高 8 位，是相邻的下一个(原地址数加 1)端口地址．

五、BASIC 语言对存贮单元和 I／O 端口访问语句

在这里顺便说明一下 IBM PC 系统的 BASIC 语言对存贮单元和 I／O 端口访问语句是有实用意义的．

访问存贮单元的 BASIC 语句有:

(1) DEF SEG 语句

用途：定义段地址．把 BASIC(或 BASICA)语言调入内存后，DS、CS、SS 都置入相同的值．这样一来，整个 BASIC 语言的工作，包括用户程序区、数据区等都在同一个 64 k 存贮区之内．如若对其它存贮区进行访问，必须先用 DEF SEG 语句定义指向希望访问的段地址．在这个语句之后，才可设置访问存贮单元的语句．

格式：DEF SEG＝段地址

其中段地址可以是表示为&H0～&HFFFF 的十六进制数，也可写成 0～65535 之内的十进制数．这个数即是定义的段地址．如果只写 DEF SEG，这个语句也是合法的，表示恢