第一章　基础知识

第一节　为什么要用汇编语言编写程序

采用高级语言（例如 BASIC,Pascal,FORTRAN 等）编写的程序，机器是不能直接执行的，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。通过编译或解释程序生成的机器语言程序往往比较冗长，占用存贮空间较大，执行起来速度慢。高级语言的程序员无法直接利用机器硬件系统的许多特性，例如寄存器、标志位以及一些特殊指令等，影响许多程序设计技巧的发挥。而汇编语言程序是直接利用机器提供的指令系统编写的程序，它与机器语言程序一一对应，因此占用存贮空间少，执行速度快。

用汇编语言编程可以充分发挥机器硬件的功能并提高编程的质量。为此学习汇编语言程序设计首先应该熟悉机器的指令系统。而指令系统又是与具体机器的内部结构密切相关的，因此要熟悉机器的内部结构，特别是中央处理器（ CPU ）和存贮器的结构。还应熟悉机器中与编程有关的其它部分的结构，例如中断系统、视频显示、键盘接收、定时功能等等。另外我们还必须知道系统为我们提供的软件环境，如放在ROM 中的监控程序、存在磁盘上的操作系统、汇编程序、调试程序等。我们可以充分利用它们的支持，直接调用其中的子程序等等。

是否采用汇编语言编写程序，要看具体的应用场合，在软件的开发时间及软件的质量方面进行权衡和抉择。一般来说某些对执行时间和存贮器容量要求较高的程序采用汇编语言编写，如实时控制系统、智能化仪器仪表及高性能软件等方面。

第二节　8086/8088 CPU 的功能结构

本章我们将首先介绍 8086/8088 CPU的内部结构和存贮器结构，作为学习汇编语言程序设计的基础知识。

Intel 8086/8088是两种第三代微处理器。在汇编语言一级，它们与 8080/8085 是兼容的。它们有 20 条地址线，直接寻址能力达 1MB。8088 具有 8 位数据通道可以与存贮器或输入输出交换信息。而 8086 则为 16 位数据通道。其它方面，两个处理器都是相同的，为其中一个 CPU 写的软件可以不需修改地在另一个 CPU 上执行。 IBM PC系列机及兼容机上广泛地采用了 8088。

8086/8088 CPU 就功能而言分成两大部分：总线接口单元 BIU（ Bus Interface Unit）和执行单元 EU（ Execution Unit ）。如图 1-1 所示。

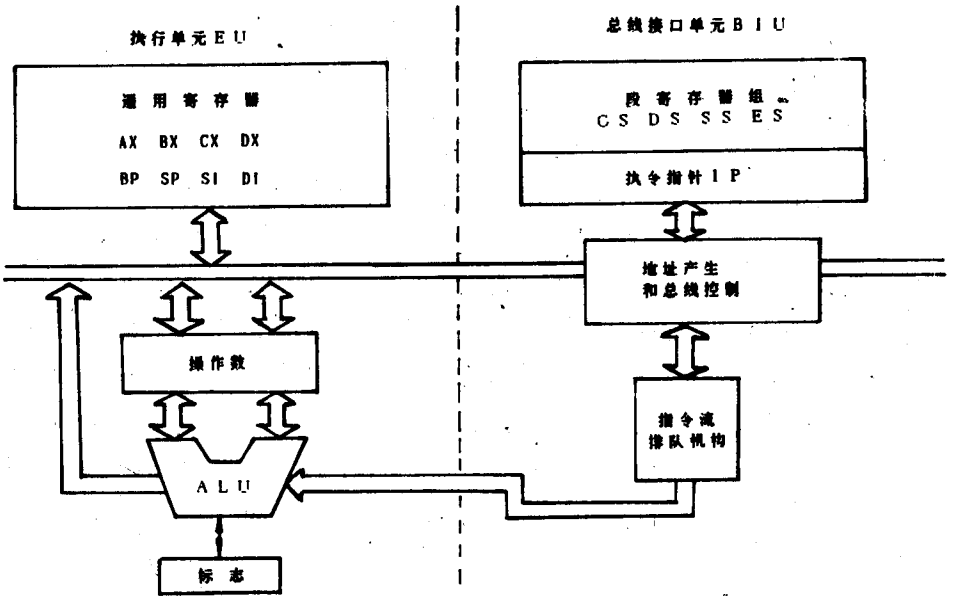

图 1-1 8086/8088 CPU的功能结构

BIU负责 8086/8088 CPU 与存贮器和外部设备之间的信息传送。具体地说，即 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，所需的操作数，也由 BIU 从内存的指定区域取出，传送给 EU 部分去指行。

EU负责指令的执行，并进行算术逻辑运算等。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（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贮器的存取速度的要求。

如前所述，在 8080/8085 等标准的 8 位微处理器中，程序的执行顺序为取第一条指令，执行第一条指令；取第二条指令，执行第二条指令；・・・ 直至取最后一条指令，执行最后一条指令。这样，在每一条指令执行完以后，CPU 必须等到下一条指令取出来以后才能执行。它的工作顺序如图 1-2 所示。

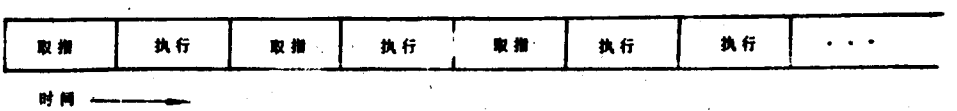

图 1-2 8 位微处理器的程序执行过程

但在 8086/8088 中，由于 BIU 和 EU 分开，所以，取指和执行可以重迭进行，它的执行顺序如图 1-3 所示。

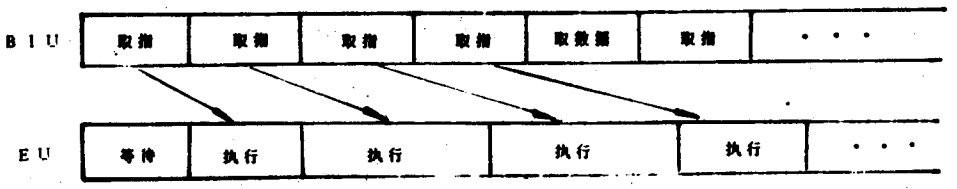

图 1-3 8086/8088 程序的执行过程

第三节 8086/8088 CPU 的寄存器结构

8086/8088 的寄存器结构如图 1-4 所示。

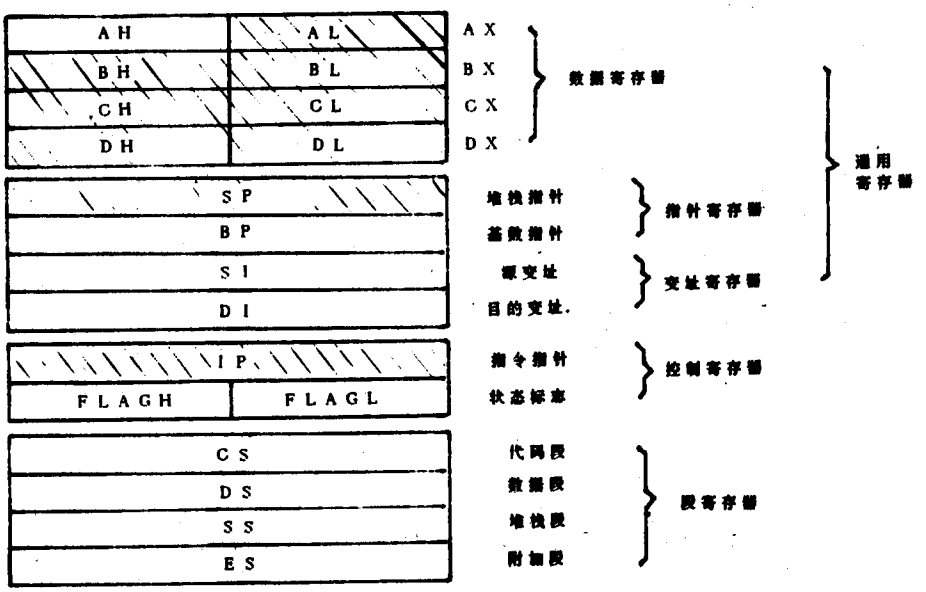

图 1-4 8086/8088 的寄存器结构

AX、BX、CX、DX 4个数据寄存器是 16 位的，其中 AX 叫累加器，它们的通常用途可以用表 1-1 说明。

表 1-1

寄存器	通 常 用 途
AX	字乘法、字除法、字 I/O
AL	字节乘法 字节除法 字节 I/O 转移 十进制算术运算
AH	字节乘法 字节除法
BX	转移
CX	串操作 循环次数
CL	变量移位或循环
DX	字乘法 字除法 间接 I/O

8086/8088 也能处理 8 位数，图 1-4 中的 4 个 16 位数据寄存器也可以作为 8个 8 位寄存器使用，图中打斜线的部分即相当于 8080/8085 中的通用寄存器。

8086/8088 中的堆栈指针 SP (Stack Pointer)类似于 8080 和 8085 中的堆栈指针，用于确定在堆栈操作时，堆栈在内存中的位置。但在 8086/8088 中 SP 还必须与 SS（堆栈段寄存器）一起才能确定堆栈的实际位置。

在 8086/8088 中增加了三个 16 位寄存器，即基数指针寄存器 BP（Base Pointer Register）、源变址寄存器 SI（Source Index Register）和目的变址寄存器 DI（Destination Index Register），还增加了几种寻址方式，从而能更灵活地寻找操作数。

8086/8088 中的指令指针 IP（Instruction Pointer）类似于 8080/8085中的程序计数器 PC。但是它们也略有不同 8080/8085 中的 PC 指向下一条即将要执行的指令，而在 8086/8088 中 IP 则指向下一次要取出的指令，另一方面，IP 要与 CS 寄存器相配合才能形成真正的物理地址。

8086/8088 中的标志寄存器占用两个字节，共有九个标志位（保留了 8080/8085 中的 5 个标志）。

此外 8086/8088 中还有 4 个 16 位的段寄存器 CS、DS、SS、ES，使 8086/8088 能在 1MB 的范围内对内存进行寻址。

有关本节所涉及到的寄存器及其使用，分别在以后有关章节中说明。

第四节 堆栈与存贮器结构

一.堆栈

堆栈是在内存RAM 中开辟的一端固定一端活动的存贮空间。活动端叫栈顶，固定端叫栈底。数据遵从先进后出的原则。所有信息的存入和取出都从栈顶进行，CPU 中有一个栈指针寄存器 SP，它始终指向堆栈的顶部(即栈顶的地址)。SP 的初值(即栈底)的设置可以由指令 MOV SP，im 来实现（im 是 16 位立即数)。堆栈主要用来进行现场数据保护，子程序与中断服务程序的调用返回等。

堆栈操作的指令分为两类，即数据进栈指令 PUSH 和出栈指令 POP。

(1) 进栈指令 PUSH

PUSH 指令可以将通用寄存器、段寄存器中的一个字推进栈顶。例如：

PUSH AX 与 PUSH BX

指令的执行分为两步：第一步先 SP-1→SP，然后把寄存器中的高位字节（如 AH）送至 SP 所指的单元。第二步再次使 SP-1→SP，把寄存器的低位字节（如 AL）送至SP所在单元。如图 1-5 所示。

随着推入内容的增加，堆栈扩展，SP 值减小，但每次操作完 SP 总指向栈顶。

(2) 出栈指令 POP

出栈指令 POP 将现行 SP 所指的栈顶的一个字传送至段寄存器或通用寄存器。

例如： POP AX

它的操作步骤是先将栈顶内容送入 AX寄存器的低位字节，再 SP+1→SP，然后再将栈顶内容送入 AX 寄存器的高位字节，再 SP+1→SP。

二．存贮器结构

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 的存贮器，地址从 00000H 到 FFFFFH。任意给定的一个 20 位地址，就可以从这 1MB 中取出所需要的指令和操作数。如前所述，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 等也都是 16 位的。因而对地址的运算也只能是 16 位。这就是说，对于 8086/8088来说，各种寻址方式，寻找操作数的范围最多可能是 64KB 。那么，它的 20位地址又是如何形成的呢？它是将整个 1MB存贮器以64KB为范围分为若干段。在寻址一个具体物理单元时，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 位偏移量来形成实际的 20 位物理地址。这个基本地址就是由 8086/8088 中的段寄存器，即代码段寄存器 CS、堆栈段寄存器 SS、数据段寄存器 DS 以及附加段寄存器 ES 中的一个来形成的，在形成 20位物理地址时，段寄存器中的 16 位数会自动左移 4 位，然后与 16位偏移量相加。如图 1-6 所示。

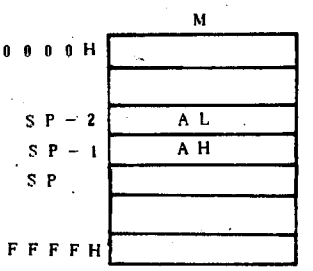

图 1-5 堆栈操作示意图

每次需要产生一个 20 位地址的时候，一个段寄存器会自动被选择。且能自动左移4位再与一个16 位地址偏移量相加，产生所需要的 20 位物理地址。

当取指令的时候，则自动选择代码段寄存器 CS ，再加上由 IP 所决定的 16 位偏移量，计算得到要取的指令的物理地址。

当涉及到一个堆栈操作时，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。

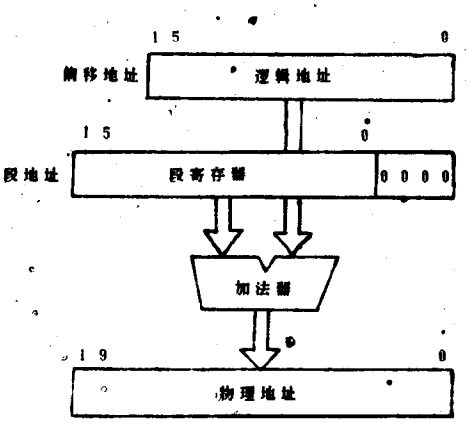

图 1-6 8086/8088 中物理地址的形成

当涉及到一个操作数时，则自动选择数据段寄存器 DS 或附加段寄存器 ES ，再加上 16 位偏移量，计算得到操作数的 20 位物理地址。 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，取决于指令的寻址方式。

在 8086/8088 系统中，存贮器的访问，如图 1-7 所示。

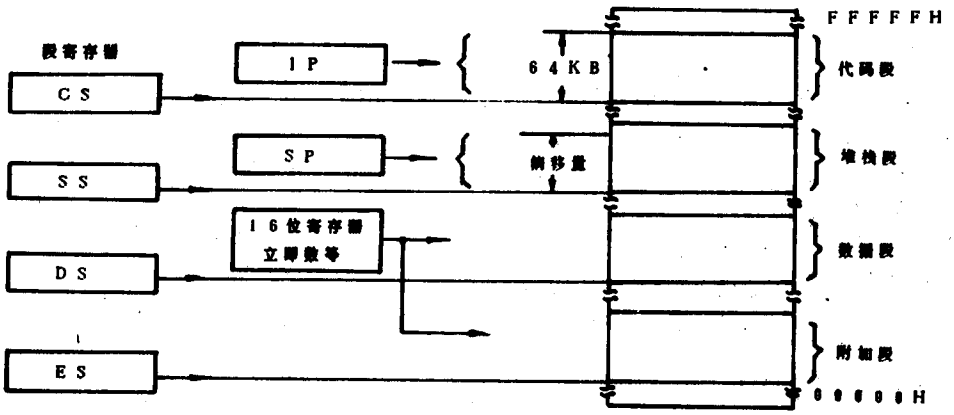

图 1-7 8086/8088 存贮器结构

所以，在不改变段寄存器值的情况下，寻址的最大范围是 64KB ，若有一个任务，它的程序长度、堆栈长度，以及数据区长度都不超过 64KB 的话，则可在程序开始时，分别给 DS、SS、CS 置值，然后在程序中就可以不再考虑这些段寄存器，程序就可以在各自的区域中正常地进行工作。若某一个任务所需的总的存贮器长度（包括程序长度、堆栈长度和数据长度等）不超过 64KB ，则可在程序开始时使 CX、SS、DS 相等，程序也能正常工作。

这种存贮器分段的方法，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。

上述的存贮器分段方法，对于要求在程序区、堆栈区和数据区之间隔离时是非常方便的。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，而不改变程序本身，这在 8086/8088中是可行的。只要使程序中的转移指令都为相对转移指令，而在运行这个程序前设法改变各个段寄存器的值就可以了。

第五节 数字、字符编码

一.ASCII 码

在计算机中，数字是用二进制表示的。而计算机处理的问题中不仅仅是数字，还包括各种字符， 如大小写英文字母、标点符号、运算符号等等。 为了计算机能识别和处理它们，这些字符应如何表示呢？由于计算机中的基本物理器件是具有两个状态的器件，所以各种字符只能按特定的规则用若干位二进制码的组合来表示。编码可以有各种方式（即规定），但要考虑通用问题。目前在微型计算机中普遍采用的是 ASCII 码（AmericanStandard Code for Information Interchange 美国标准信息交换码），IBM PC ASCII码字符

表见附录 B。

它是用八位二进制数编码，故可以表示 256 个字符。表中最上两行是位 7 到位 4的代码。最左边的两列是位 3 到位 0 的代码。用此表，可实现字符与其 ASCII码的互查。从表中可以看出数字 0 到 9 对应的 ASCII 码是 30H 到 39H，英文大写字母 A～Z 对应的 ASCII 码是 41H 到 5AH ，英文小写字母 a～ z 的 ASCII 码是 61H 到7AH 。整个表以中间为界分为左、右两部分。左边 128 个字符的 ASCII码值的最高位为 0，右边128个字符的 ASCII 码值的最高位为 1 。256 种字符大体可以分为如下几类 ：

- 16 个专用的游戏符号
- 15 个用于文字处理编辑的符号
- 96 个常用 ASCII 字符
- 48 个外语字符
- 48 个商用图形符号
- 16 个常用希腊字母
- 15 个常用科学符号

二．BCD 码

虽然二进制数实现容易、可靠，二进制运算规律十分简单，但是二进制数不直观、书写麻烦、易错。于是在计算机输入输出时通常还是采用人们习惯的十进制数表示。这就产生了二进制编码的十进制数，称为 BCD 码（ Binary Code Decimal ）。

一位十进制数用四位二进制数编码表示，表示的方法可多种，通常用的是 8421 BCD 码。其特点是：

(1) 8421 BCD 码的十个十进制数字 0 到 9 分别用四位二进制数表示。四位二进制数有 16 种组合，取前十种，即用 0000、0001、・・・ 、1001 分别表示 0 到 9。

(2) 每组四位二进制数之间是二进制的 ，而组与组之间（即 BCD 码的十进制数之间）是十进制的。

例如：

98D=(1001 1000)BCD

16D=(0001 0000)B=(0001 0110)BCD

可以从 (0100 1001.0110 1001)BCD 方便地认出是十进制数的 49.69。很容易实现十进制数与 BCD 码的转换。关于 BCD 码与二进制间的转换，手算比较费事，可以由专门的指令或程序实现。

习题一

1．8086/8088 CPU 中有哪些寄存器？如何分组？各有什么用途？

2．设堆栈指针 SP 的初值为 2000H，AX=3000H，BX=5000H，试问：

(1) 执行指令 PUSH AX 后，SP=?

(2) 再执行 PUSH BX 及 POP AX 后，SP=? AX=? BX=?

并画出堆栈变化示意图。

3. 8086/8088系统中，存贮器的物理地址由哪两部分组成？每一个段与寄存器之间有何对应要求？

第二章　8086/8088 的指令系统

本章介绍8086/8088 的部分指令、寻址方式和有关的标志寄存器，以便进一步学习程序设计的方法。

第一节　寻址方式

寻址方式就是指令中用于说明操作数所在地址的方法。8086/8088 的基本寻址方式有六种。

一.立即寻址（Immediate Addressing）

这种寻址方式所提供的操作数直接包含在指令中。它紧跟在操作码的后面，与操作码一起放在代码段区域中。如图 2-1 所示。

例：MOV AX，im

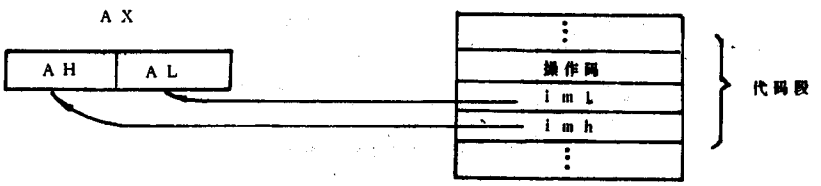

图 2-1 立即寻址示意图

立即数 im 可以是 8 位的，也可以是 16 位的。若是 16 位的，则 imL 在低地址字节，imH 在高地址字节 。若是字操作数 ，而且它的高位字节是由低位字节符号扩展的，则在指令中的立即数，只具有低位字节。

立即寻址主要是用来给寄存器或存贮器赋初值。

二.直接寻址（Direct Addressing）

操作数地址的16位偏移量直接包含在指令中，它与操作码一起存放在代码段区域。操作数一般在数据段区域中，它的地址为数据段寄存器 DS 加上这 16 位地址偏移量。如图 2-2 所示。

例如：　MOV　AX，DS:[2000H]

指令中的 16 位地址偏移量是低位字节在前，高位字节在后。

这种寻址方法，是以数据段的地址为基础，故可在多达64KB的范围内寻找操作数。

在 8086/8088中允许段超越。即对于寻找操作数来说，还允许操作数在以代码段、堆栈段或附加段为基准的区域中，即只要在指令中指明是段超越（详见第三章）的，则16位

地址偏移量可以与 CS 或 SS 或 ES 相加，作为操作数的地址。

MOV AX，DS:[2000H]

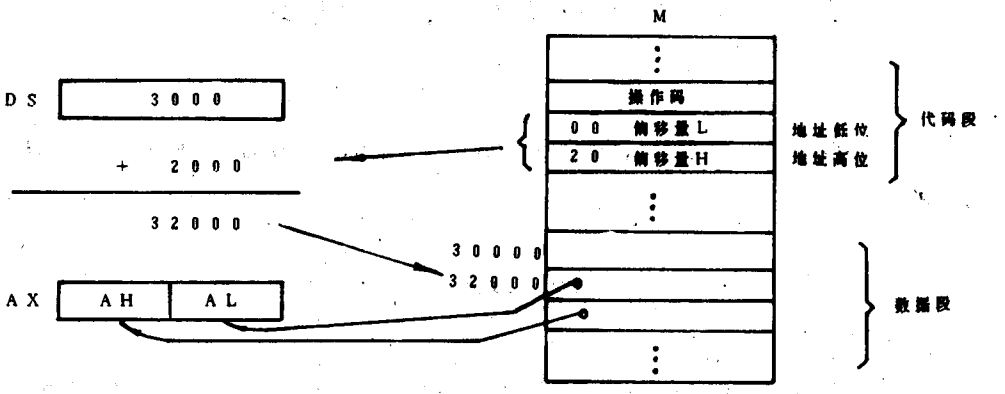

图 2-2 直接寻址示意图

三.寄存器寻址 (Register Addressing)

操作数包含在 CPU 的内部寄存器中，例如寄存器 AX、BX、CX、DX等，如图 2-3 所示。

例: MOV DS，AX

图 2-3 寄存器寻址方式示意图

虽然操作数可在CPU 内部通用寄存器的任意一个中，且它们都能参与算术或逻辑运算和存放运算结果，但是， AX 是累加器，若结果是存放在 AX 中的话，则通常指令执行时间要短些。

四.寄存器间接寻址 (Register Indirect Addressing)

在这种寻址方式中，操作数是在存贮器中。但是，操作数地址的 16 位偏移量包含在以下四个寄存器 SI、DI、BP、BX 之一中。这又可以分成两种情况:

1.若以 CI、DI、BX 间接寻址，则类似于 8080 中的 HL 间接寻址，通常操作数在现行数据段区域中，即数据段寄存器 DS 加上 SI、DI、BX 中的 16 位偏移量，为操作数的地址，如图 2-4 所示。

例: MOV AX，[SI]

2.若是以寄存器 BP 间接寻址，则操作数在堆栈段区域中，即堆栈段寄存器 SS 与 BP相加作为操作数的地址，如图 2-5 所示。

例 : MOV AX，[BP]

若在指令中规定是段超越的，则 BP 也可以与其它的段寄存器相加，形成操作数地址。

MOV AX, [SI]

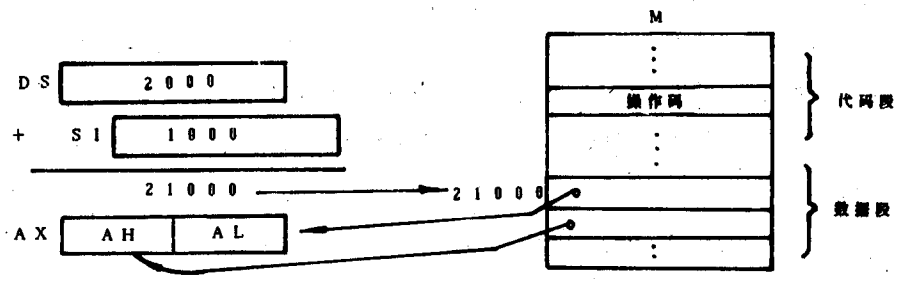

图 2-4 寄存器间接寻址示意图

MOV AX, [BP]

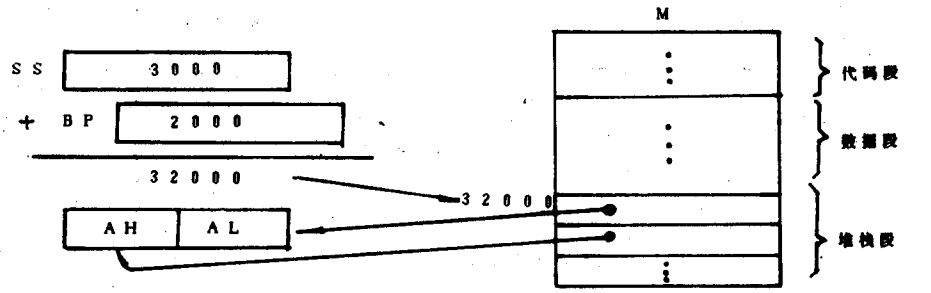

图 2-5 以 BP 作为间接寻址示意图

五.变址寻址 (Index Addressing)

所谓变址寻址就是由指定的寄存器内容，加上指令中给定的 8 位或 16 位偏移量（当然要由一个段寄存器作为地址基准)作为操作数的地址。

上述可以作为寄存器间接寻址的四个寄存器SI、DI、BX、BP也可以作为变址寻址。如图 2-6 所示。

例: MOV AX，COUNT [SI]

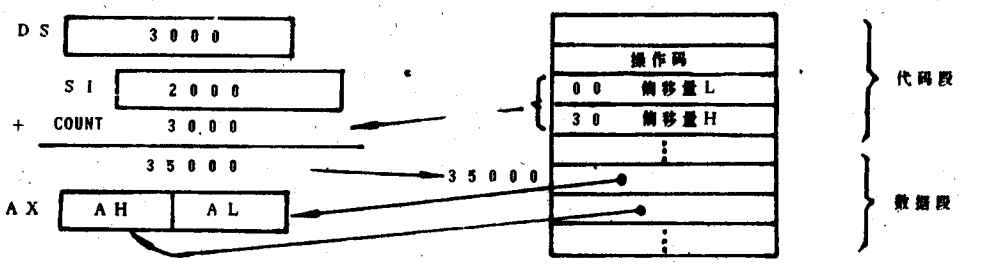

图 2-6 变址寻址示意图

在正常情况下，若用 SI、DI 和 BX 作为变址，则与数据段寄存器相加，形成操作数的地址；若用 BP 变址，则与堆栈段寄存器相加，形成操作数的地址。

但是，只要在指令中指定是段超越的，则也可以用别的段寄存器作为地址基准。

六.基址加变址寻址

在 8086/8088 中，通常把 BX和 BP看作是基址寄存器，把 SI、DI看作变址寄存器，可以把这两种寻址方式组合起来形成一种新的寻址方式。这种寻址方式是把一个基址寄存器（BX 或 BP）的内容加上一个变址寄存器（SI 或 DI）的内容，再加上指令中指定的 8 位或 16 位偏移量（ 当然要以一个段寄存器作为地址基准 ）作为操作数的地址，如图2-7 所示。

例如： MOV AX，MASK [BX] [SI]

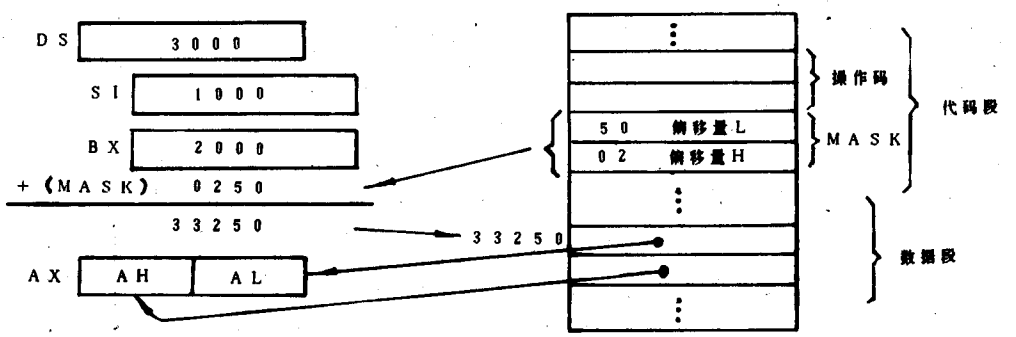

图 2-7 基址加变址寻址示意图

在正常情况下，由基地址决定哪一个段寄存器作为地址指针。即若用BX作为基地址，则操作数在数据段区域中；若用 BP 作为基地址，则操作数在堆栈段区域中，但若在指令中规定段是超越的，则可用其它段寄存器作为地址基准。

如上所述，8086/8088 中的存贮器是分段的，我们寻找一个内存操作数，只能在某一个段的 64KB 范围内寻找，以什么寄存器间址、变址与基址加变址，则操作数在什么段区域中，在 8086/8088 中有一个基本约定。 只要在指令中不特别说明要超越这个约定，则正常情况就按这个基本约定来寻找操作数，这就是所谓的 Default(默认)状态。这些基本约定和允许超越的情况如表 2-1 所示。

表 2-1

存贮器存取方式	约定段基数	可修改的段基数	逻辑地址
取指令	CS	无	IP
堆栈操作	SS	无	SP
源串	DS	CS、ES、SS	SI
目的串	ES	无	DI
用 BP 作为基寄存器	SS	CS、DS、ES	有效地址
通用数据读写	DS	CS、ES、SS	有效地址

第二节　标志寄存器

标志寄存器反映系统的状态及运算结果的特点。8086/8088 中有一个标志寄存器，占用 2 个字节，共有 9 个标志位，如图 2-8 所示。

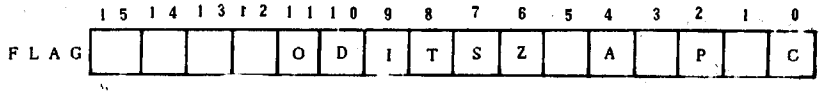

图 2-8 8086/8088 中的标志位

一.标志位

各个标志位的功能分述如下：

1.辅助进位标志 AF (Auxitiary Carry Flag)

在字节操作时若由低半字节(一个字节的低 4 位) 向高半字节有进位或借位；在字操作时，低位字节向高位字节有进位或借位，则 AF=1 ，否则为 0。这个标志用于十进制算述运算指令中。

2.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，则CF=1，否则为 0。这个标志主要用于多字节数的加、减法运算。移位和循环移位指令也能够把存贮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。

3.溢出标志 OF (Overflow Flag)

在算术运算中，带符号数的运算结果超出了8 位或 16 位带符号数能表达的范围，即在字节运算时 >+127 或 <-128，在字运算时>32767 或 <-32768，此标志置位。一个任选的溢出中断指令，在溢出情况下能产生中断。

溢出和进位是两个不同性质的标志，千万不能混淆了。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。

又例如，在字节运算时：

```
        MOV AL，0ABH
        ADD AL，0FFH
```

即:

```
    10101011   (-85)
+   11111111   (-1)
-------------------
  1 10101010
```

D7位向前有进位，故运算后 CF=1 ；但运算的结果又不小于 -128 ，此时，溢出标志 OF=0。

在字运算时，如有

```
        MOV   AX,0064H
        ADD   AX,0064H
```

即：

```
    00000000 01100100
+   00000000 01100100
-----------------------
  00000000 11001000
```

D15 位未产生进位，故 CF=0；运算结果显然未超 +32767 故 OF=0。

但若有

```
        MOV AX,6400H
        ADD AX,6400H
```

即：

```
    01100100 00000000
+   01100100 00000000
-----------------------
  11001000 00000000
```

D15 位未产生进位，故 CF=0。但运算结果不小于 +32767 溢出标志 OF=1。

又例如：

```
        MOV AX,0AB00H
        ADD AX,0FFFFH
```

即：

```
    10101011 00000000
+   11111111 11111111
-----------------------
  110101010 11111111
```

D15 位产生进位，故 CF=1，但运算结果不小于 -32768，故 OF=0。

4.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（字节操作时为 D7，字操作时为 D15）为 1，则 SF=1；否则 SF=0。

由于在 8086/8088 中符号数是用补码表示的，所以 SF 表示了结果的符号，SF=0 为正，SF=1 为负。

5.奇偶标志 PF (Parity Flag)

若操作结果中"1"的个数为偶数，则 PF=1，否则 PF=0。这个标志可用于检查在数据传送过程中是否发生错误。

6.零标志 ZF (Zero Flag)

若运算的结果为 0，则 ZF=1，否则 ZF=0。

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

7.方向标志 DF (Direction Flag)

若用指令置DF=1，则引起串操作指令为自动减量指令，也就是从高地址到低地址处理字符串；若使 DF=0，则串操作指令就为自动增量指令。

8.中断允许标志 IF (Interrupt-enable Flag)

若指令中置 IF=1，则允许 CPU 去接收外部的可屏蔽中断请求；若使IF=0，则屏蔽上述的中断请求；对内部产生的中断不起作用。

9.追踪标志 TF (Trap Flag)

置 TF 标志，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。

二.标志操作指令

8086/8088 中有一部分指令是专门对标志寄存器或标志位进行操作的。包括四条标志寄存器传送指令和标志位操作指令。

1.标志寄存器传送指令

(1) LAHF (Load AH with flags)

把标志寄存器的低 8 位（包括符号标志 SF、零标志 ZF、辅助进位标志 AF、奇偶标志 PF 和进位标志CF）传送至 AH 的指定位，即相应地传送至位 7、6、4、2 和 0。（位 5、3、1 的内容没有定义），如图 2-9 所示。

这条指令本身不影响这些标志位。

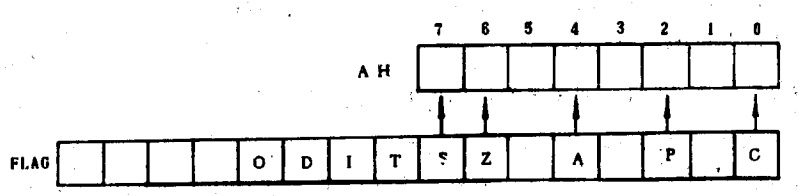

图 2-9 LAHF 指令示意图

(2) SAHF (Store AH into Flags)

这条指令与上一条的操作刚好相反，它是把寄存器 AH 的指定位，传送至标志寄存器的低 8 位的 SF、ZF、AF、PF 和 CF 标志。因而这些标志的内容就要受到影响，这取决于AH中相应位的状态，但这条指令并不影响溢出标志 OF、方向标志 DF、中断屏蔽标志IF 和追踪标志 TF，即不影响标志寄存器的高位字节。

(3) PUSHF (Push Flags)

把整个标志寄存器（包括全部九个标志）推入至堆栈指针所指的堆栈的顶部，同时修改堆栈指针，即 SP-2→SP。

这条指令不影响标志位。

(4) POPF (POP Flags)

这条指令把现行堆栈指针所指的一个字，传送给标志寄存器，同时相应地修改堆栈指针，即 SP+2→SP。

这条指令执行后，8086/8088 的标志位就取决于原堆栈顶部的内容。

PUSHF 和 POPF 这两条指令可以保存和恢复标志寄存器。在子程序调用和中断服务中可利用这两条指令来保护和恢复标志位。

另外，这两条指令也可以用来改变追踪标志TF。在8086/8088 的指令系统中，没有直接能改变TF标志的指令，故若要改变 TF标志，先用 PUSHF 指令把标志位入栈，然后设法改变栈顶存贮单元的 D8 位（把整个标志看成一个字）再用 POPF 指令恢复，这样其余的标志不受影响，而只有 TF 标志按需要改变了。

2.标志位操作指令

8086/8088 有七条直接对标志单独进行操作的指令。其中有三条是针对进位标志 CF 的，有两条是针对标志 DF 的，有两条是针对中断标志 IF 的。

(1) CLC (Clear Carry Flags)

此指令使标志 CF=0

(2) CMC (Complement Carry Flags)

此指令使标志 CF 取反，即若 CF=0，则 1→CF；若CF=1，则 0→CF。

(3) STC (Set Carry Flags)

此指令使标志 CF=1

(4) CLD (Clear direction Flag)

此指令使标志 DF=0，则在串操作指令时，使地址增量。

(5) STD (Set Direction Flag)

此指令使标志 DF=1。则在串操作指令时，使地址减量。

(6) CLI(Clear Interrupt enable Flag)

此指令使中断允许标志 IF=0，于是在 8086/8088 系统中，外部装置送至可屏蔽中断 INTR 引线上的中断请求，CPU 就不予以响应---也即中断屏蔽。但此标志对于非屏蔽中断 NMI 引线上的请求，以及软件中断都没有影响。

(7) STI(Set Interrupt-enable Flag)

此指令使标志 IF=1，则 CPU 就可以响应出现在 INTR 引线上的外部中断请求。

上述 7 条指令除对指定的标志位进行操作外，对其它标志位皆无影响。

第三节　　指令系统

8086/8088 的指令系统可以分为以下六个功能组。

1.数据传送 (Data Transter)

2.算术运算 (Arithmetic)

3.逻辑运算 (Logic)

4.串操作 (String Menipulation)

5.程序控制 (Program Control)

6.处理器控制 (Processor Control)

本节中我们就其中的一部分指令加以介绍，其余部分将结合程序设计方法的讲解分散到各章节中去介绍。为查阅指令方便，书后有附录 A（指令系统综述与查阅表 ），可由此找到各类指令的详述。

一.数据传送指令：

这里主要介绍 MOV、XCHG 和地址传送指令。

1. MOV OPRD1,OPRD2

MOV 是操作码，OPRD1 和OPRD2 分别是目的操作数和源操作数。此指令把一个字节或一个字操作数从源传送至目的。

源操作数可以是累加器、寄存器、存贮器以及立即操作数，而目的操作数可以是累加器、寄存器和存贮器。

数据传送方向的示意图，如图 2-10 所示。

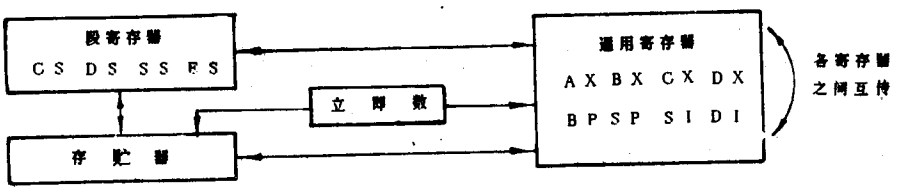

图 2-10 数据传送方向示意图

具体来说，一条数据传送指令能实现：

(1) CPU 内部寄存器之间数据的任意传送（除了代码段寄存器 CS 和指令指针 IP 以外)。例如：

```
MOV AL,BL
MOV DL,CH
MOV AX,DX
MOV CX,BX
MOV DS,BX
```