# INTEL8086程序设计基础

张开善　熊第衡　葛本修
徐爱卿　朱淑桃　裴　珉　编

電子工業出版社

## 内 容 简 介

本书对8086微型计算机系统的硬件和软件作了较全面地分析。着重阐述了8086汇编语言程序设计的方法。对SDK—86（国产型号TP—86）单板机及其监控程序进行了分析，并对8086宏处理语言作了简要介绍。

本书可供从事微型计算机科研、生产和应用的工程技术人员参考，亦可作为应用8086微型计算机系统的工程技术人员，有关院校及培训班的教材或参考书。


INTEL 8086程序设计基础

张开普 熊第衡 葛本修
徐爱卿 朱淑桃 裴 珉 编

责任编辑：王惠民

电子工业出版社出版（北京市万寿路）
新华书店北京发行所发行 各地新华书店经售
北京市密云县印刷厂印刷

开本：787×1092 1/16 印张：22.125 字数：510千字
1987年8月第1版 1987年8月第1次印刷
印数：6,000 定价：4.70元
统一书号：15290·433

# 目　录

# 第一章　8086的结构

计算机的结构就是指组成这台计算机的功能部件以及它们之间的配合和联系。它所涉及的问题是：在计算机中有多少寄存器；这些寄存器具备哪些功能；能连接多少存储器；存储器如何寻址，以及能使用哪些类型的输入／输出设备。

8086是一个单片处理器，它具有四组寄存器。第一组是通用寄存器，它们是用来存放中间结果的；第二组是指针和变址寄存器，它们用来在存储器的规定段区中给信息定位；第三组是段寄存器，它们用来确定存储器的段区；第四组是指令指针。在8086中还有九个标志位，这些标志位用来记录处理器的状态和控制它的操作。8086可以访问1兆字节的存储器和65,000个输入或输出端口。本章的前半部分将对这些特性作详细地说明。

典型的计算机指令需要对被处理的操作数（Operands），即被处理的数据进行定位，并对这些操作数的数值执行规定的操作，还要将结果存放到某个规定的结果单元。操作数和结果的存放单元可以由指令指定在存储器或寄存器中。通过指令的说明，确定并找到这些存储单元有效地址的方法称为计算机的操作数寻址方式（Operand-Addressing Modes）。8086操作数的寻址方式将在本章的后半部分介绍。有关对不同操作数进行处理的实际指令将在第二章阐述。

## 第一节　存储器

### 一、存储器的结构

存储器的每一个存储单元的长度为8位二进制代码，也就是一个字节(Byte)；8086系统的存储器可以具有$2^{20}$（近似于1,000,000）个字节的容量。每个字节指定一个唯一的地址（无符号数），因此它的寻址范围是从0至$2^{20}-1$(二进制表示为00000000000000000000至11111111111111111111；十六进制表示为00000H至FFFFFH)①，如图1-1所示。

存储器中任何两个相邻的字节中存放一个16位的字（Word），在一个字中的每个字节有一个字节地址；这两个地址中数值较小的那个作为这个字的地址。有关字的例子如图1-2所示。

一个字包含16位。其中的低8位放在较低位的地址中，该地址加1后的单元存放这个字的高8位；如果在存储器中存放一个字节的序列，那么最低8位放在地址较低的单元中，以后依次存放，最高的8位则应放在最高位的地址中。这就是说8086存储器中字

① 以后我们用数字后面加字母B(Binary)表示二进制数；以字母Q（Octal）表示八进制；以字母D（Decimal）或不加字母表示十进制数，用字母H（Hexadecimal）表示十六进制数。

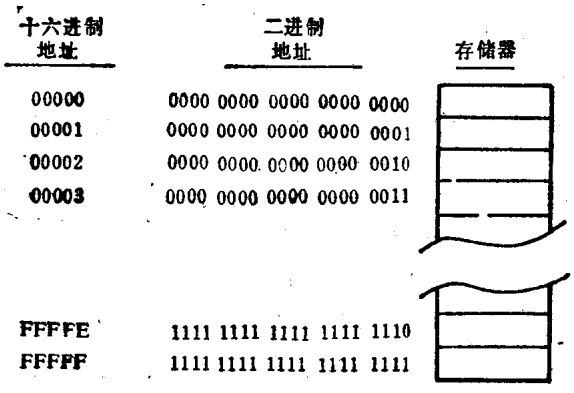


图1-1 存储器地址

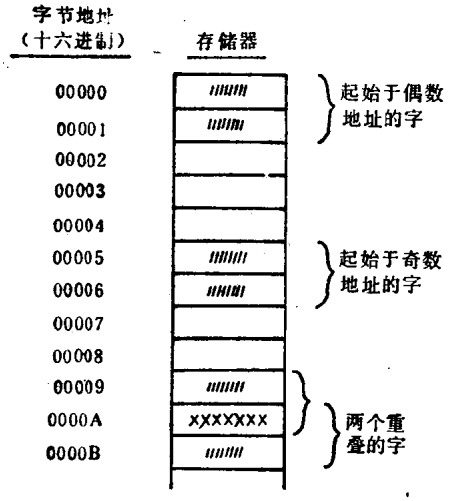


图1-2 存储器中字的例子

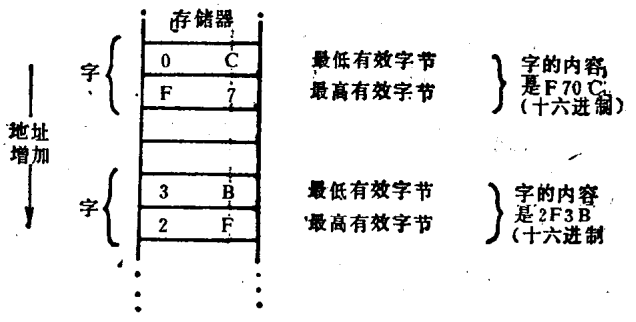


图1-3 存储器中"自下而上"存储的例子

的存放是反向的（或称自下而上存放），如图1-3所示。

8086访问内存的指令有两种方式，一种是字节访问指令；另一种是字访问指令，然而8086每次访问内存储器的信息量总是16位。在执行字节访问指令时，16位信息中仅有8位是有用的，可以是第一个字节，也可以是第二个字节，而其他的8位则视为无效，如图1-4(a)、(b)所示。这16位字在存储器中总是放在以偶数地址起始的两个相邻的字节中，这样在读、写一个以偶数为起始地址的字时，只需访问一次存储器；然而，对于以奇数为起始地址的字，就必须两次访问存储器，读、写两个以偶数为起始地址的两个字。忽略每个字中所不需要的那个字节，并对所须的两个字节进行字节调整，如图1-4(c)(d)所示。但是，它从存储器取出或存入的信息量每次总是16位。在字节指令的情况下，这些位数仅有8位是有用的，而其他的8位则为无效。16位字在存储器中总是存放在以偶数地址起始的两个相邻字节中，这就是说，读或写一个以偶数为起始地址的字的指令，只须访问一次存储器；然而，对于以奇数为起始地址的字的指令，就必须两次访问存储器中的两个偶数地址的字，忽视每个字中所不需要的那半个字，并对剩下的那两个半字进行某种字节调整，各种字节和字的读操作例子如图1-4所示。

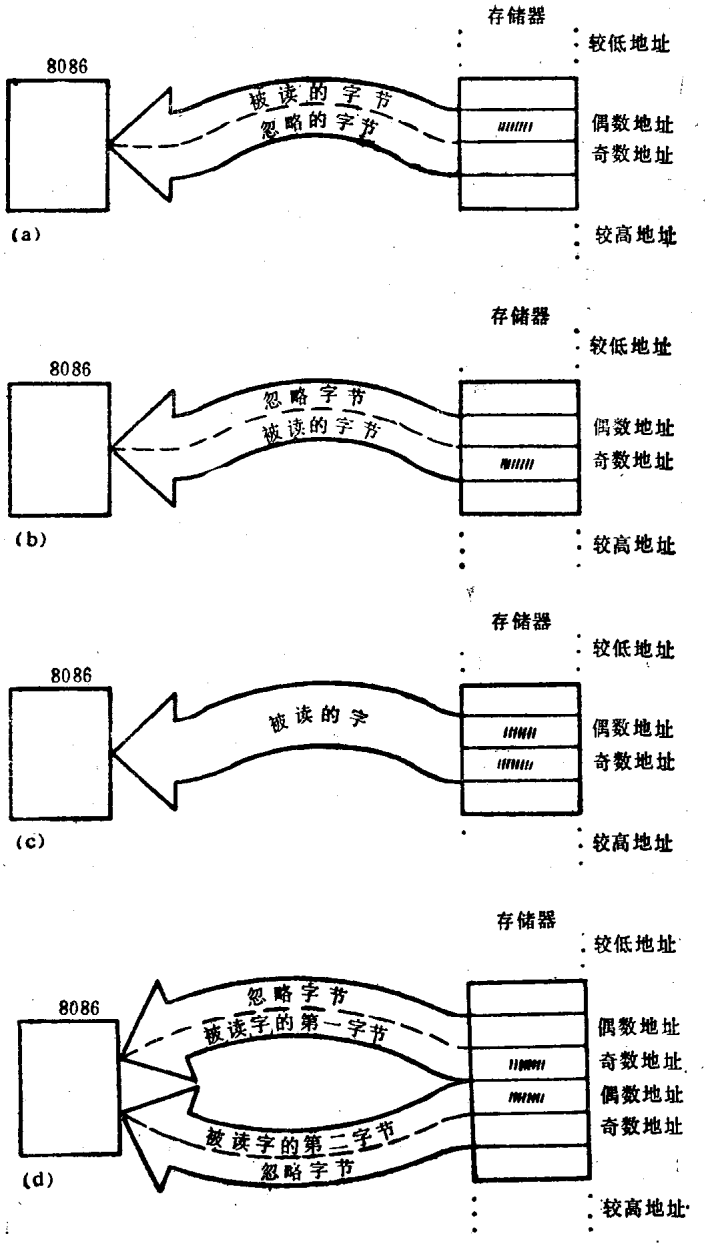


图1-4　从8086存储器的偶数和奇数地址读字节和字。
(a) 读偶数地址中的字节；(b) 读奇数地址中的字节；(c) 读偶数地址中的字；(d) 读奇数地址中的字，要求两次存储器访问

在8086（或8088）程序中，指令仅要求指出对某个字节或字进行访问，而对存储器访问的方式不必说明，无论执行哪种访问，都是由处理器来完成的。

## 二、存储器的分段

因为8086可以寻址存储器的容量为$2^{20}$个字节。这就是说，在8086处理器中，字节和字的地址必须由20位来表示。但，8086是字长为16位的机器，对地址的运算也只能有

16位长。因此，要求有一个附加的装置进行地址计算。

我们可以将1兆字节的存储器分成若干段；每段包括$2^{16}$（近似65，000）个字节。每段的首地址都是一个能被16整除的数字（即字节地址的最低4位都是0）。在任何给定的时刻，程序能够立即访问这样四个段中的内容。这四个段称为：现行代码段（Current Code Segment）、现行数据段（Current Data Segment）、现行堆栈段（Current Stack Segment）和现行附加段（Current Extra Segment），附加段是一个通用区，通常作为一个附加的数据段来看待。我们将这四个现行段的第一个字节的地址的16位最高有效位放入四个专用寄存器中的一个，作为这个现行段的标记。这些寄存器分别称为分段寄存器（Segment Registers）。CS:代码段寄存器；DS:数据段寄存器；SS:堆栈段寄存器；ES:附加段寄存器。各个段不需相等，而且它们可以重叠。有关段的例子如图1-5所示。

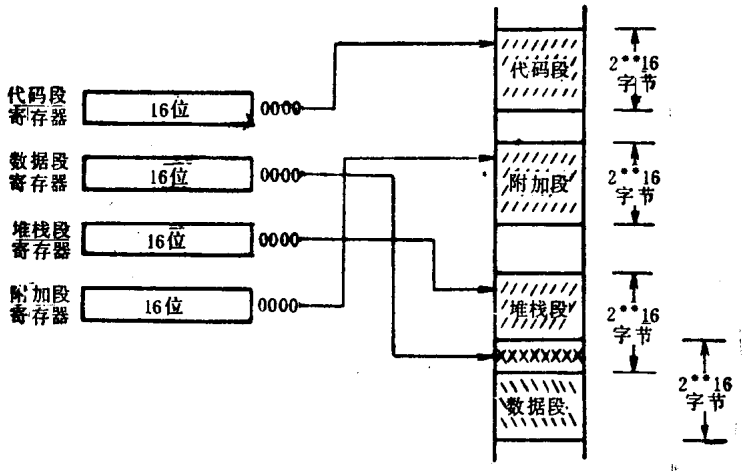


图1-5 段的例子。注意在这个例子中堆栈段和数据段重叠

例如，假设16位的代码分段寄存器包含的数值是C018H，这样代码段的起始字节地址是C018H，而段长度最大可达$2^{16}$（10000H）个字节。因此，该代码段的最后字节地址是D017FH。在一个段内，地址则可以由一个16位的偏置量指出。处理器将这个16位偏置地址与分段寄存器的内容相加，并在它的低四位添加四个零后，与它原来的数值相加，构成20位字节或字地址，如图1-6所示。

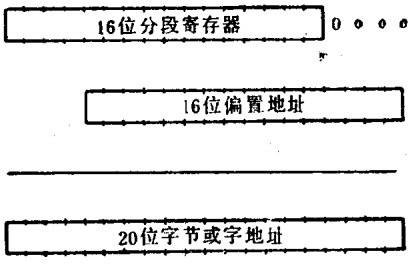


图1-6 字节或字地址结构

因此，在上例中，这个字节在现行代码段中的字节地址是CFFFFH，它在代码段中有一个FE7FH的偏置地址（CFFFFH—C0180H），如图1-7所示。

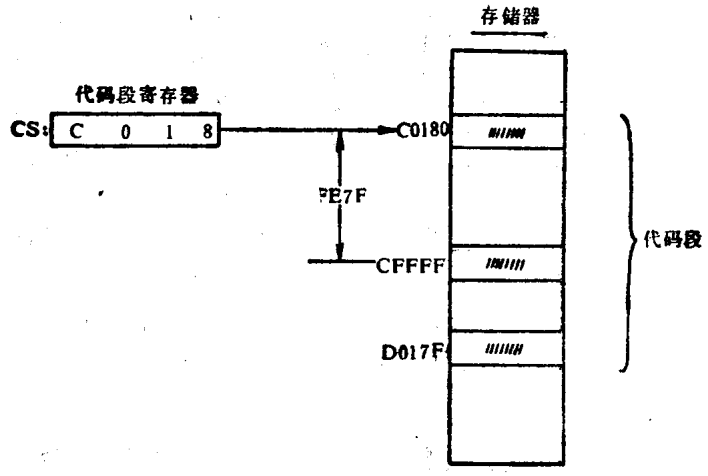


图1-7 字节地址结构的例子

## 第二节 输入/输出的结构

8086系统与外部环境连接的装置称为端口（Port）。通过这些端口，8086可以接收外部的信息，并且可以发出信号来控制其他的过程。

8086可以访问类似于存储器字节的$2^{16}$（近似于65,000）个8位端口，每个8位端口由不同的地址确定，它的范围是从0至$2^{16}-1$。任何两个相邻的8位端口可以作为一个相当于存储器字的16位端口来处理，16位奇数地址端口（对于8088则是所有16位端口）则要求进行两次访问。事实上，除了它们没有端口分段寄存器以外，端口寻址与存储器字节或字的寻址方式是相同的；换句话说，全部端口可以认为是在一个段中。

## 第三节 寄存器结构

8086处理器包括十三个16位寄存器和9位标志位。如前所述，这些寄存器可分为四组。其中三组，每组包括四个寄存器，第十三个寄存器称为指令指针，它不能由程序员直接使用，而是由它自身进行置位。8086的寄存器和标志位如图1-8所示。

三组可编程的寄存器是通用寄存器，指针和变址寄存器以及分段寄存器。通用寄存器主要是用来保存算术和逻辑运算的操作数；指针和变址寄存器用来保存段内的偏置地址；分段寄存器用来确定段的起始地址。

### 一、通用寄存器

处理器中，如果没有通用寄存器，则各个指令就必须从存储器取出它的操作数并将运算结果存入存储器。但是，存储器访问比较费时，若将经常使用的操作数和结果保存在一个能快速存取的地方，就可减少这个存取时间。为此，8086在处理器中设有四个通用寄存器。

8086的16位通用寄存器是AX，BX，CX和DX。每个通用寄存器可分为两个8位寄

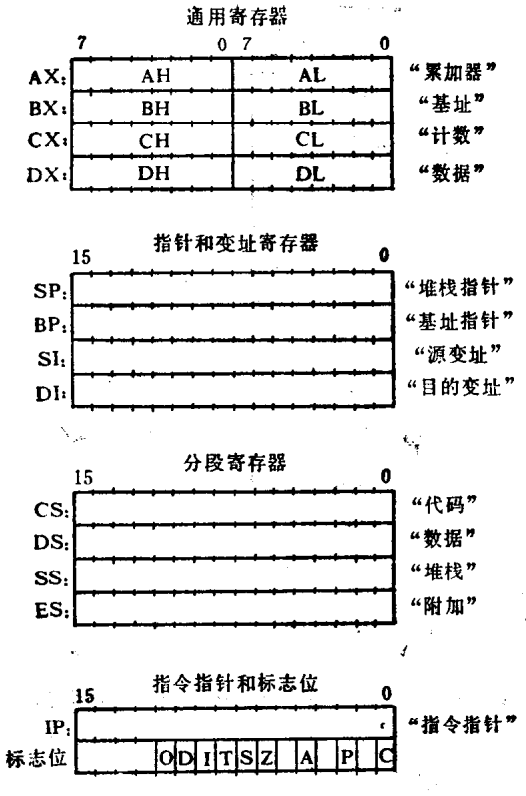


图1-8 8086的寄存器和标志位

存器；也可以将这两部分组合成为一个16位寄存器。因此，每个通用寄存器的两个 8 位寄存器给定了它自己的名称：其低位有效部分的名称分别为AL，BL，CL和 DL；而 高位有效部分的名称是AH，BH，CH和DH。这些寄存器的双重性使得它们很容易地处理字节和字的信息。

通常，通用寄存器的内容可以交互地参与8086的算术和逻辑运算。例如，加法指令（ADD)可以是两个 具有相同位数的通用寄存器的内容进行相加，而将其结果存放在其中的某个寄存器中。但是，某些指令规定某个通用寄存器有某种专门的用途。例如，在串操作指令中要求 CX 寄存器用来作为计数器，记录数据串中的元素的数目。而AX，BX或DX寄存器 都 不能用于这个目的。CX 寄存器的 这种特殊用途就可以把它称为计数（Count）寄存器，而对于 AX，BX和DX寄存器可以分别称为累 加 器（Accumulator)，基址（Base）和数据（Data）寄 存 器。

通用寄存器的这些特殊规则在程序较长的情况下，可以使程序缩短。在编程的过程中要将特殊类型的数据放在特殊的寄存器中，如果我们选择CX寄存器来记录数 据 串中元素的数目；那么这个数据就总存放在其中，不必将数据串的大小传送到CX。如 果 不设计这种特殊用途，则各通用寄存器的功能是相同的，每个字符串指令也就必须指出该特殊字段存放在哪个寄存器中，这就使得每条串指令变长，由一字节指令变为两字节指令；或者需要采用多字节的串指令，这都使得程序变长. 因此对某些指令要使用这些专用寄存器，这在减少程序的长度方面是具有切实效果的。

## 二、指针和变址寄存器

在一条访问存储器单元的指令中，可以直接指出这个单元的地址。这个地址占据指令的一定空间，因而增加了代码的长度。如果将常用单元的地址存放在特定 的 寄 存 器中，那么访问这些单元的指令只需指出该寄存器的地址即可。这可以大大缩短指令的长度。这种寄存器往往称为指针或变址寄存器。

8086的指针和变址寄存器由四个16位寄存器SP，BP，SI和DI组成。对于 在一 个 段内的寻址，这些寄存器通常存放的是偏置地址。例如，一条加法指令（ADD），可以 规定该指令的一个操作数的地址是在存贮器的现行数据段中，而其段内的偏置包含在指定的指针或变址寄存器（譬如SI）中。

此外，指针和变址寄存器提供另外的（或者说更重要的）功能，即减小指令的长度。当程序正在运行过程中，允许指令访问的单元，它们的偏置地址正是前面指令的计算结果。尤其在高级语言程序中，为了建立可变的偏置地址，需要执行这种计算。这些计算可以在通用寄存器中执行，而将结果作为一个偏置值传送到指针或变址寄存器中（这种传送又将占用指令，这就增加了程序的长度）。指针和变址寄存器中的内容允许参与16位通用寄存器的算术和逻辑运算。因此，上述加法指令的另一个操作数可以是DI寄存器中的内容。

这四个寄存器分成指针寄存器SP和BP；以及变址寄存器SI和DI。指针寄存器为现行堆栈段的访问提供了一种方便的方式。对于实现高级语言来说，堆栈段的用途就象一个“数据区”（这将在本章的结尾论述）。因此，如果不特别指明是某个段，则在指针寄存器中存放的偏置值被认为是在现行堆栈段中；反之，包含在变址寄存器中的偏置值通常被认为是在现行数据段。例如，如果加法指令（ADD）规定：SI包含该指令的一个操作数的偏置值，则那个操作数就可以认为在现行数据段中，除非ADD指令明确地规定了某些其他段。

某些指令区分两个指针寄存器SP和BP。PUSH（推入）和POP（弹出）指令从SP寄存器可以得到栈顶位置的偏置量。因此，这个寄存器称为堆栈指针（Stack Pointer）。BP寄存器不能用于这个目的，它使BP寄存器可以任意存放在堆栈段中数据区的“基址”的偏置量。因此，称BP寄存器为基址指针（Base Pointer）。

此外，串操作指令使两个变址寄存器SI和DI之间有所区别。这些串操作指令要求一个源操作数和一个目的操作数，源操作数的偏置地址从SI得到。同样，DI中存放着目的操作数的偏置地址，它们称为源变址器（Source Index）和目的变址器（Destination Index），对于这些串操作指令，SI和DI的作用不能互换。例如，数据串传送指令要求把在现行数据段起始的，以SI中的内容为偏置的数据串传送到由现行附加段重定位的，以DI内容为偏置的单元中，SI和DI寄存器不必在串传送指令中明确地说明。在附加段中的目的串偶尔可以代替在数据段中的目的串。因此，每个数据串将有它自己的一个段，且这个段可以达到$2^{16}$字节长。

## 三、分段寄存器

如前所述，8086具有1兆字节的存储器，但是，包括在指令中的以及在指针和变址寄存器中的地址仅有16位长。这种地址不可能在1兆字节的存储器中进行寻址，因为该地址只能是某个特定的65,000字节段中的偏置量。

8086的分段寄存器都是16位寄存器，它们是CS，DS，SS和ES。这些寄存器指明一个特定现行段，并且彼此不能互换使用。CS确定为现行代码段寄存器，DS为现行数据段寄存器，SS为现行堆栈段寄存器，ES为现行附加段寄存器。

某条指令在某个段中指定一个偏置。而分段寄存器确定我们可以使用的四个段。选用哪一个段，这要取决于偏置量的意义。偏置可以确定下一条执行的指令，或者可以确定一条指令的操作数。

所有指令都是从现行代码段取出的。因此，需要一个寄存器，它可存放下一条要执行的指令在现行代码段中的偏置，这就是IP的作用，IP称为指令指针（Instruction

Pointer)。例如，如果CS包含数值1FF7H，而IP包含003AH。那么，下一条指令将从存储器的IFFAAH单元取出。因为

| | |
|---|---|
| 1FF70 H | 代码段起始地址 |
| + 003A H | 包含在IP中的偏置 |
| 1FFAA H | 下一条指令的存储单元地址 |

（由图1-5中可以看出，十六进制的“0”值是在构成存储单元地址时，在分段寄存器中附加上的值）。

对于取操作数的分段，通常可以在指令的前面带上一个特定的1字节前缀来区分。这个前缀确定四个现行段；操作数从四个现行段中的任何一个取出。在没有这种前缀的情况下，操作数是从现行数据段取出；除非偏置地址是从指令寄存器的内容计算出来的，在这种情况下则使用现行堆栈段；或者该操作数是一条串操作指令的目的操作数，在这种情况下，则使用现行附加段（对于这两个例外的原因，已在前节提及）。

例如，研究一个加法指令（ADD），它的一个操作数在数据段，其偏置包含在SI中，指令将在它的操作数字段指明SI，但是，没有提及DS。当执行这条指令时，为了找到操作数地址，处理器可以与使用IS的内容一起使用DS的内容。其次，如果有一条操作数在代码段中，而偏置包含在SI中的加法指令（与在ROM中驻有常数的情况一样）。如前所述，这条加法指令将在它的操作数字段指定SI。此外，该指令将由前缀字节来确定CS。

## 四、标志寄存器

8086包括九个标志位，它们用来记录处理器的状态信息（状态标志位Status Flags）或者控制处理器的操作（控制标志位Control Flags）。状态标志位通常是在执行算术或逻辑指令之后置位，它反映这些操作结果的某些特征。这些标志位是进位标志（CF），它表示指令在最高位产生一个进位；辅助进位标志（AF），它表示指令最低的第四位产生进位；溢出标志（OF），它表示指令执行产生一个带符号的结果，它超出了所能表示的范围；零标志（ZF），它表示指令产生了一个为零的结果；符号标志（SF），表示指令产生一个负的结果；奇偶标志（PF），表示指令产生的结果具有偶数个“1”位。

控制标志位是方向标志（DF），它控制串操作指令的方向；中断允许标志（IF），允许或者不允许外部中断；陷阱标志（TF），为了程序调试，它迫使处理器进入单步方式。

在第二章中将对这些标志位进行详细说明，在这章的最后一节将对这些标志位的性能进行总结。

## 五、8086内部结构的特点

8086处理器可以处理16位数据，也可以处理8位数据。它除了能执行整套8080/8085的指令系统外，还增加了很多新的16位操作指令（包括乘法和除法）及按位处理指令。此外，它还具有一些小型机所具备的功能，如重新进入码的操作以及采用动态可浮动程序。

8086与8080寄存器的相当部分在图1-9中用斜线表示出来。

8086处理器的功能框图见图1-10。其中包括两大部分：总线接口部件BIU与执行/控制部件EU。EU包括数据寄存器和算术逻辑运算部件，它用来执行基本的处理功能。它从总线接口部件BIU接受预先取出的指令，并把操作数地址送回到BIU。然后，它再通过BIU接收操作数，对它们进行处理并把处理结果送到BIU，再将其存储起来。

总线接口部件BIU保持着6个字节的预先取出的排队指令流，用来执行取指令、读操作数和写结果，并实现地址浮动。BIU和EU可分别独立地控制，以使指令的执行和取出能重叠进行（当EU执行指令时，BIU已经把下面要执行的指令取出来）。在系统的其他部件不使用存储周期时，总线接口部件则占用存储器的总线，使其不断"忙于"按指令顺序预取以后的指令字。

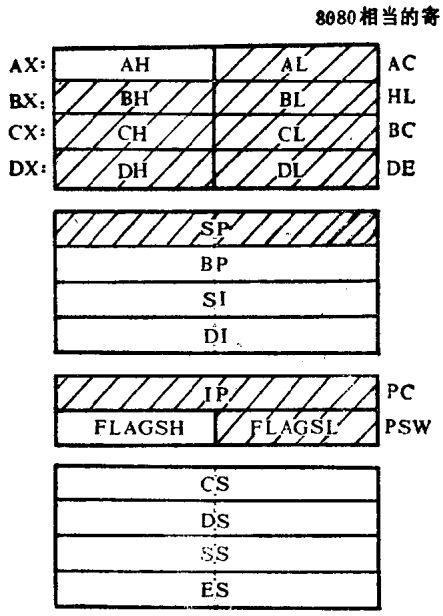


图1-9　8080寄存器可以看成是8086寄存器的一个子组

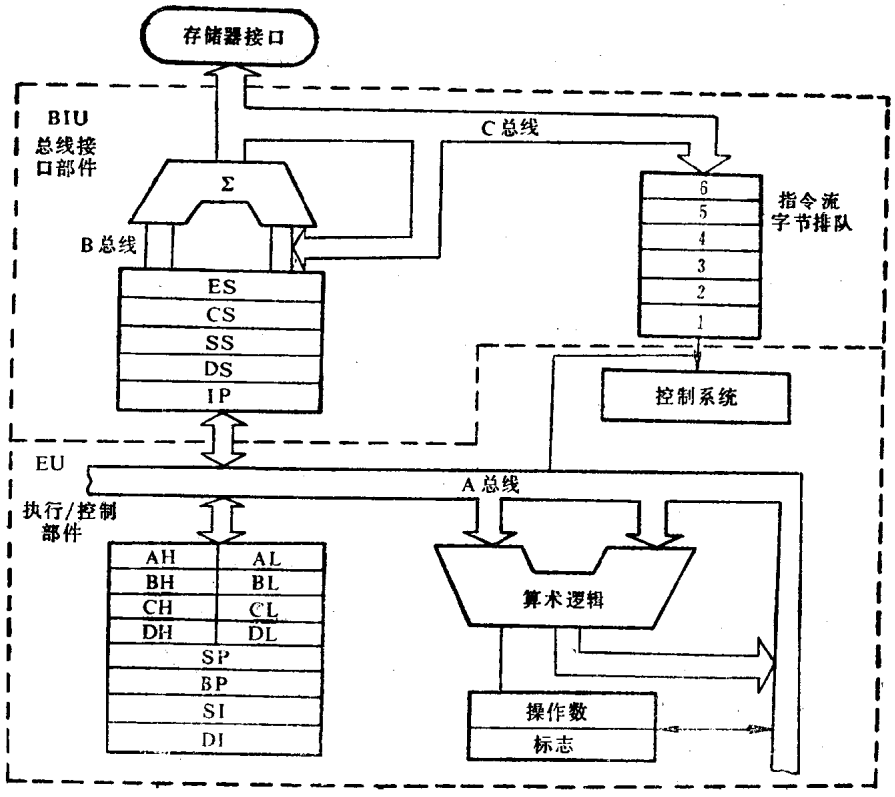


图1-10　8086处理器功能框图