Pentium/80486

实用汇编语言程序设计

艾德才等　编著

清 华 大 学 出 版 社

(京)新登字 158 号

内 容 简 介

本书系统地介绍了 Pentium/80486 微处理机汇编语言程序设计的概念、方法和技巧；介绍了中断与输入/输出、BIOS 功能调用及其相关程序设计；介绍了汇编语言程序与高级语言程序的接口技术；书末给出了 Pentium/80486 全部指令。

本书强调实用性与先进性。内容安排由浅入深，便于自学。书中列举大量实例，简明易懂，且均上机调试过。

本书可供初级以上的计算机应用人员作为教科书及实用参考书，也可用作为大、中专院校有关专业的教材或教学参考书。

图书在版编目(CIP)数据

Pentium/80486 实用汇编语言程序设计/艾德才等编著 . 北京：清华大学出版社，1997
ISBN 7-302-02684-X

Ⅰ.P… Ⅱ.艾… Ⅲ.汇编语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(97)第 21505 号

出版者：清华大学出版社(北京清华大学校内，邮编 100084)
Internet 网址 www.tup.tsinghua.edu.cn
印刷者：昌平环球印刷厂
发行者：新华书店总店北京科技发行所
开 本：787× 1092 1/16 印张：25 字数：442 千字
版 次：1997 年 11 月第 1 版 1997 年 11 月第 1 次印刷
书 号：ISBN 7-302-02684-X/TP・1388
印 数：0001～6000
定 价：26.00 元

前　言

汇编语言是一种功能很强的程序设计语言,是计算机能提供给用户的最快的而又最有效的语言,也是能够利用计算机所有硬件特征并能直接用来控制硬件的唯一的一种程序设计语言,它给程序设计人员一种对计算机的绝对控制手段。若用户对程序空间和时间要求很高,对数据采集和处理速度要求很快,用高级语言编制程序又无能为力时,就只能用汇编语言来解决这一难题。

汇编语言有两个突出特点:一是汇编语言的语句与机器指令一一对应,所以用汇编语言进行程序设计不仅可以从根本上认识理解计算机工作过程,而且程序设计人员能够充分利用计算机的硬件资源。二是用汇编语言编写的程序目标代码短,运行速度快。

近年来,微处理机技术飞快发展,32 位微处理机的杰出代表——80486 直至 Pentium,业已进入工厂、机关、团体、学校,进入了各行各业,甚至家庭。但是目前市场上介绍 Pentium/80486 硬件基本知识及其汇编语言程序设计方面的书较少。本书的目的就是为广大 Pentium/80486 微处理机用户提供一本既可用作基本教材,又可用作实用参考的图书。

书中内容安排由浅入深,从易到难、循序渐进。首先通过介绍汇编语言基本知识、Pentium/80486 基本结构,引导读者一步步进入 Pentium/80486 汇编语言程序设计世界,然后通过实例告诉读者怎样用 Pentium/80486 汇编语言编写比较简单的汇编程序,再逐步过渡到利用某些算法来编制比较高级的汇编语言程序。书中对程序的每一条语句都用中文给以解释,便于读者理解。书中列举了大量实用汇编语言程序,读者可以在实践中直接引用。因此本书可作为程序设计的工具书。

本书第一、二、三、四、五、七章由艾德才编写,第九、十、十三、十五章由张桦编写,第六、八、十一、十二章由陆明编写,第十四章、第十六章由吴奇编写,第十七章由胡敏编写,全书由艾德才组织编写、审阅定稿。

参加本书编写工作的还有潘琴、刘文丽、胡琳、高华芬、艾菲、卢弘岩、于宝龙、蔡欣琪。

由于编者水平有限,新技术新词汇尚未规范,书中不足和谬误在所难免,敬请计算界前辈、同仁及广大读者不吝指正。

编　者

1997 年 3 月于天津大学

目　　录

第一章　汇编语言程序设计基础

第一节　基 本 知 识

本节首先介绍汇编语言的基本知识,如字符、字节、双字、四字、10 字节等。

80486 除支持由 8088、8086、80286 所支持的数据类型外,还支持 32 位带符号和不带符号的整数,以及从 1 位到 32 位这么大范围的位字段(Bit Field)操作。而由 8088、8086 和 80286 所支持的指针类型也被扩展成仅有偏移量的指针和整个 48 位的指针。

一、字节(Byte)

众所周知,计算机绝对不允许杂乱无章地存放各种长度的二进制数或若干二进制位。在 80486 体系结构内,所谓一个单元就是 8 个连续的二进制位序列,就是字节。

字节中的每一位都有编号,最低位 LSB 编号为 0,最高位 MSB 编号为 7。图 1.1 显示出若干连续存储单元,每一个存储单元内都保存一字节信息。这些信息可能是一条机器指令,或是另一个存储单元的地址,也可能是数据或字符数据。

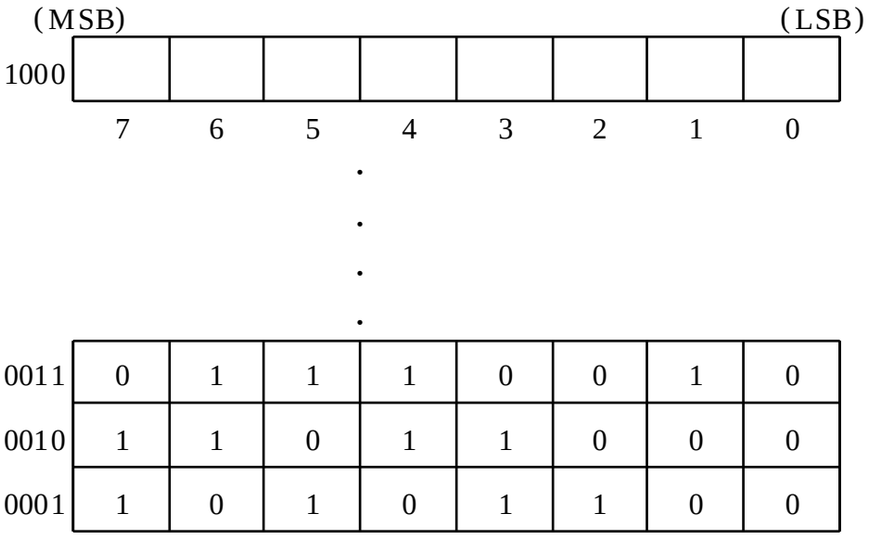

图 1.1　连续存储单元示例

8 位可以产生 256 个状态。一个存储单元表示的正整数可从 0 到 255。在一个存储单元内也可保存一个 ASCII 字符。“字”是一个专业术语,指的是在单个存储单元内存放的位数。在早期的微处理机中一个存储单元只能存放 4 位。而现在某些计算机一个存储单元可存放 64 位。在这种情况下,所谓字,指的就是一个 64 位字串。按照 80486 的结构,一个字是 32 位。一个 8 位的字节(BYTE)还可以再分成 2 个四位组,叫半字节。一个半字节刚好可以表示一个 16 进制的一位数。

二、字符(Characters)

字节中的8位也可表示一个字符的代码。ASCII码实际上是7位代码,它可以表示字母字符和数值字符。ASCII编码给每一个字母、数字、专用字符、专用控制字符都赋予一个二进制代码。

7位编码能表示128个符号和代码。第8位有时用作数据传输和检索错误的检测代码。有些字符ROM芯片的制造厂家使用第8位去访问一个扩充的字符集。有时把第8位作为附加位使表示的符号加倍而达到256个。此举用来表示某些专用外语符号、数学符号和那些非常有用而又重要的图形符号。

三、字

一个由8个二进制的位组成的字节不能方便地处理所有汇编语言程序设计问题。这样在80486上就特别需要另外一种组织位的方法,即字。

80486是32位机,既可以使用8位和16位整型数据,也可使用由两个字节组合成的一个字,如图1.2所示。字允许表示的不带符号的整数范围超出255,最大可到65 535(16进制的00FF到FFFF)。而实际上汇编程序在存储和操作时都把16位的整数当成字对待。对于存放字符串数据和整型变量的存储单元,都由16位的地址指针保持跟踪,其范围从0000H到FFFFH。

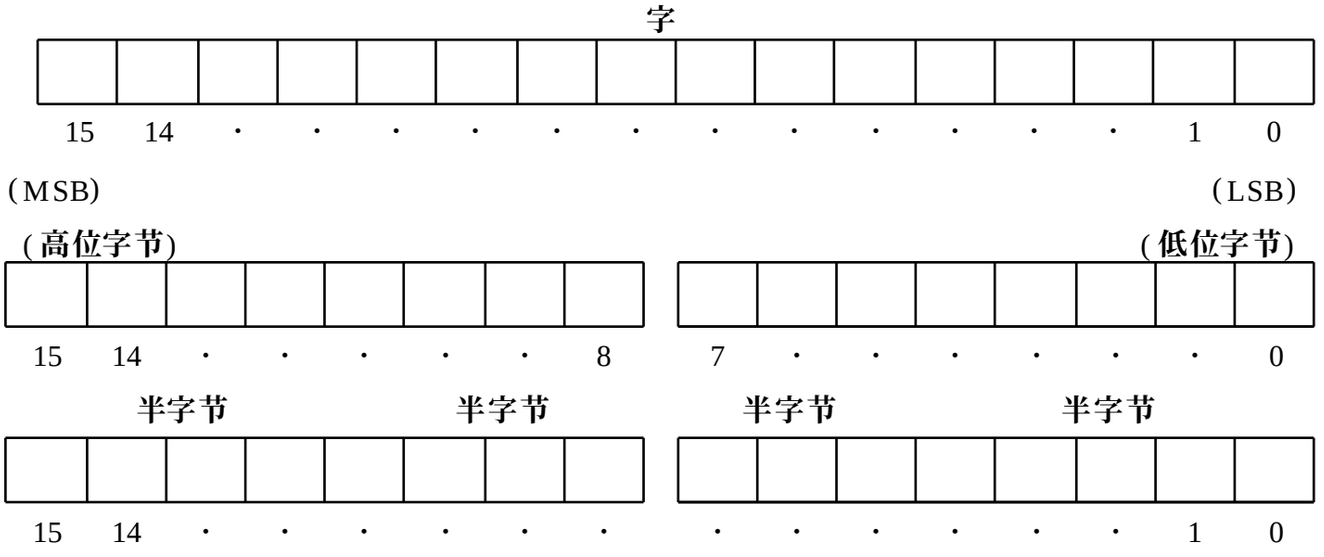

图1.2 字和字节

当把一个整数字存放在存储器中,这两个字节是以反向顺序形式存放的。即先放低序位字符LSB,再放高序位字节MSB。也就是说,高序位字节的地址编号比低序位字节的地址编号大。为便于记忆,可认为低序位字节存在低地址,高序位字节存在高地址。例如,把3546H这个数存放在存储器内,其存放次序应为:

存储地址	值
00000001	35H
00000000	46H

在绝大多数情况下,汇编程序设计人员没有必要因字的存储方案而感到烦恼。因为存储器的指令非常理解存储方式,并且在存储时都要进行必要的转换。但设计人员也应该知道存储方法。因为在存储跟踪以及在转储时是有用的。就微处理机而论,说到字,就是16位的字,而不是8位的字节。

四、双字(Double Word)

双字,就是两个字。它是由连续存放的两个相邻的字构成,其宽度为32位,如图1.3所示。这是一种非常重要的数据形式,汇编语言程序设计人员时常需要使用32位的双字。利用双字的字段宽度,可使用32位的双字进行算术运算,以提高运算精度。32位数据可以以浮点形式和整数形式表示非常大和非常小的数。

双字在存储时的排列顺序类似于字的排列顺序。在这种情况下,低序位的字存储在地址序号低的存储单元,而高序位字则存储在紧接高序位地址存储单元。32位双字是作为一个四字节串存储的,以低序位字节LSB开始到高序位字节MSB结束。数12345678H在存储器存放时的顺序如图1.3所示:

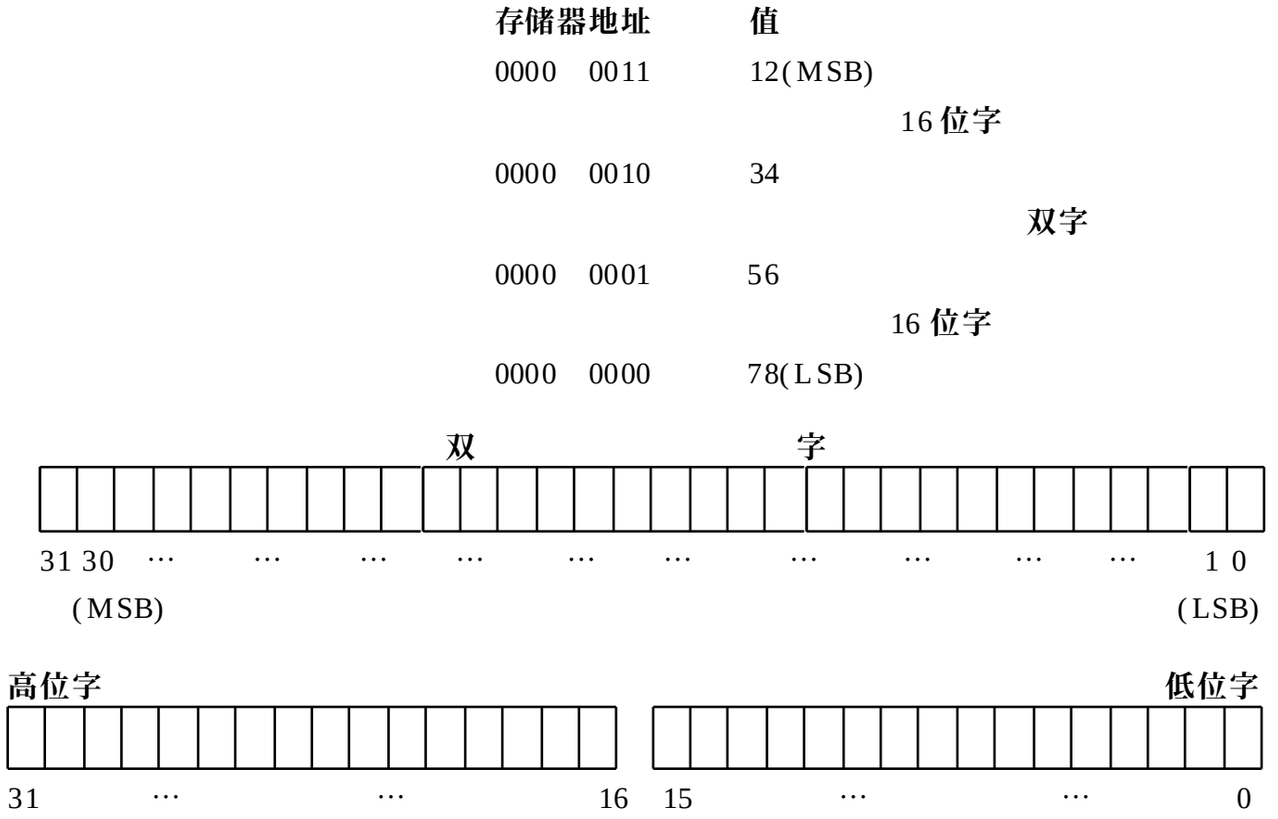

图1.3 双字中的字节和字

五、四字(Quad Word)

如果双字还不能满足数字精度的话,可用四字。所谓四字就是连续的四个字作为表示信息的一个单位,四字能够存放非常大的数或字符串。四字存储方案与以前的例子完全一致。低位字存在序号低的存储单元,而高位字存在序号高的存储单元。例如数值1234567890ABCDEF在存储器内的存放顺序如图1.4所示。

存储器地址	值	
00000111	12H(MSB)	
00000110	34H	32 位双字
00000101	56H	
00000100	78H	
00000011	90H	
00000010	ABH	32 位双字
00000001	CDH	
00000000	FFH(LSB)	

图 1.4　四字存储示意图

六、10 字节

10 字节,顾名思义,就是用 10 个字节表示一个信息单位。它是一个 80 位的值,因而能够存储极大的数值或字符数据。它是 80486 微处理机定义的最大的数据类型。10 字节的存储方案在结构上与四字和二字相同,即低位字节存放在序号低的存储单元,而高位字节则存放在连续的高序号存储单元。

七、非标准的位字段

汇编程序设计人员所遇到的大部分数据都可以归并到以前定义的数据类型中。但也有一个例外,例如,PC-DOS 的设计者为提高存储器的利用率,决定在一个单值特定结构内存放日期标记。由于需要检验年、月、日字段值的范围,完全可以仅用一个 16 位的字表示日期标记。设计者采用7位表示年字段。从理论上说,这样安排的表示范围为从0到127。而实际上,所表示的范围却被限制在从0到119之间,而且还要在这个值上加上基数1980。因为一年之内仅有12个月,用4位表示月字段就足够了。考虑到一月之内最多31天,用5位来表示日字段满富裕,因为2的5次方等于32,5位表示范围即为从0到31。

另外非标志位部分包括计算机在图形学上的应用。在监视器屏幕上,单独一位可表示一个点。如果这位为 0,没有光点;如果这位为 1,则有光点。

光点映像能从一个系统向另一个系统变化。例如,当有光点时,若要给光点一种颜色,那就必须给光点分配两位,用来描述在光点条件允许之下,用两种不同的颜色描述亮还是不亮。

所有这些专用的非标准位结构在存储、访问和控制操作时,都可把它们当成二进制数据。在解释存储器和存储器信息转储时务必注意。

八、二进制操作

微处理机在执行最简单的加法和减法操作时,对微处理机进行的仔细检查表明,它仅是一次反复的位串比较。CPU 内有各种各样加法电路。但是,最后这些电路还是依靠逐位比较进行工作。

逐位比较包含一个非常简单的处理过程,因为只能有三种可能的位对组合。或者二者全为 0,或者二者中有一个为 1,或者二者全为 1。不论执行的是加法、减法、乘法或除法操作,根据以上三种位对组合,都要实施简单位串比较。

在做加法时,从低位开始使用下面几项规则:

(1) 两 0 位相加,把和标志位置 0;

(2) 如果有 1 位为 1,则和标志位置 1;

(3) 如果这两位都为 1,则和标志位置 0,同时把进位标志置 1。

这个处理过程在每个顺序位上重复,连同操作符一起还要顾及到以前的进位。CPU 还保持对存储器区域中每次每个位比较的跟踪,其结果看起来很像二进制加法。而实际上,这些处理过程不会有什么操作会超过重复的位比较。

虽然 CPU 用硬件实现了加、减、乘和除操作,程序人员还必须使用其它的位比较操作。这些操作包括逻辑 AND, OR, XOR, SHIFT LEFT, SHIFT RIGHT, ROTATE LEFT, ROTATE RIGHT 和 COMPLEMENT。随着逻辑 AND, OR 和 COMPLEMENT 操作,观察位作为逻辑上的真(True)或假(False)标志,而不作为数值,是有帮助的。

在进行逻辑 AND 操作时,要进行两位的比较。如果二者均为 1,其结果为 1。请注意,和二进制加法不同,在二进制加法时,两个均为 1 的位进行比较,把和标志位置 0,而把进位标志置 1。逻辑 AND 操作结果如下:

	逻辑 AND			
	0	0	1	1
AND)	0	1	0	1
结果)	0	0	0	1

经常使用 AND 操作去挑选屏蔽某一位。例如,在执行十进制的乘法或除法之前,需要清掉一个未压缩的十进制数中最高四位。这可以用 00001111 与那个未压缩的十进制数进行 AND 操作来完成。例如:

	1	0	1	0	0	0	1	1
AND)	0	0	0	0	1	1	1	1
	0	0	0	0	0	0	1	1

对逻辑 OR 操作,只要两位中有一位为 1 或者两者都为 1,则结果才为 1。OR 操作对于设置专用位特别有用。OR 操作结果如下:

	逻辑 OR 操作			
	0	0	1	1
OR)	0	1	0	1
结果)	0	1	1	1

例如,用 10000000 通过 OR 操作可把一个 8 位数中的最高位 MSB 置 1。

```
        0  0  0  1  1  0  1  0
OR)     1  0  0  0  0  0  0  0
        ------------------------
        1  0  0  1  1  0  1  0
```

逻辑 EXCLUSIVE OR(异或)操作也是比较两位。只有当两位的值不同时,结果才为1。逻辑异或操作在把指定位变反时非常有用。异或操作多用于图形学。逻辑异或操作结果如下:

```
        逻辑 EXCLUSIVE OR
           0  0  1  1
异或)      0  1  0  1
        ---------------
结果)      0  1  1  0
```

下面例子中,是用 00111100 与另一个八位数进行异或操作,而把其中间的四位变成补码。

```
           1  0  1  0  0  1  1  0
EX OR)     0  0  1  1  1  1  0  0
           ------------------------
           1  0  0  1  1  0  1  0
```

SHIFT LEFT/RIGHT, ROTATE LEFT/RIGHT 和 COMPLEMENT 都是对单独一个操作数操作。SHIFT 指令特别适用于对给定数值进行乘 2 或除 2 操作。使用这个指令完成乘 2、除 2 操作,比乘法及除法所需的时钟数、字节数少得多。

不带符号的数,左移一位,最低位用 0 补齐,就完成对这个数的二倍运算。

```
SHL)  0  1  0  0  0  0  0  1  (十进制的 65)
      ------------------------------------
      1  0  0  0  0  0  1  0  (十进制的 130)
```

把一个不带符号的数缩小一半,只需把这个数右移一位,而最高位用 0 补齐即可。

```
SHR)  0  0  0  0  1  0  1  0  (十进制的 10)
      ------------------------------------
      0  0  0  0  0  1  0  1  (十进制的 5)
```

ROTATE 指令提供了对一个数进行重新排列的能力。这条指令与 SHIFT 指令相似之处是,一次可向左或向右移动一位。不同之处在于,这条指令在左移或右移时并不丢失移出的那一位。而是把移出的那一位再补到另一端腾空的那一位上。例如:

```
ROR)  0  0  0  0  1  1  1  1  (右环移)
      ------------------------------
      1  0  0  0  0  1  1  1

ROL)  0  0  0  0  1  1  1  1  (左环移)
      ------------------------------
      0  0  0  1  1  1  1  0
```

第二节　寻 址 方 式

一条 80486 指令不仅含有完成这条指令所需的特殊操作信息,而且还包含要处理的操作数的类型以及操作数的存储单元。共有八种主要寻址类型,分别介绍如下:

一、立即寻址

微处理机使用哪一种寻址方式要由操作语法决定。例如,一条指令是这样写的:

```
MOV     AH，00
MOV     AL，04
```

该指令包含操作数的值。这时 AH 寄存器被清 0,而 AL 寄存器内装入的是二进制的 00000100。在此尚需强调,80486 主要接收 32 位操作数。下例是把 16 位的源操作数传送到 AX 寄存器内:

```
MOV   AX, 0FFFFH(任何 16 进制数前必须写 0)
```

而指令

```
MOV   EAX, 0ABCD1234H
```

的作用则是把一个 32 位的操作数装到 EAX 寄存器内。

在利用立即寻址方式时,所有操作数的值都需要扩充符号。也就是说,操作数的最高位的值都要重复,以便完成目的操作数的位宽度。例如,

```
MOV   AX, 302
```

这条指令就用了与 302 等价的 10 位二进制数 0100101110,必须把它扩充成 16 位的目的操作数,其办法是通过把 0 符号位重复几次,把它变成 AX 寄存器中的高位字段。经过这么处理 AX 寄存器中的内容变成 0000000100101110。

再举一个例子,

```
MOV   AL,- 40
```

也把符号扩充为 8 位的源操作数和目的操作数。- 40 用 7 位(1011000)表示,就要把它扩充到八位,变成 11011000。

二、寄存器寻址

所谓寄存器寻址,就是源操作数的值已经存放在 80486 的一个内部寄存器内。这个值可以是 8 位的,也可以是 16 位的,还可以是 32 位的。微处理机是用寄存器的名字说明操作数的宽度。例如,

```
MOV   EDX, EAX
```

这条指令的功能是把 EAX 寄存器内的 32 位内容送至 EDX 寄存器。而下一条寄存器寻

址的指令

```
MOV  DX, AX
```

的作用是指示处理机到 AX 寄存器取 16 位的源操作数,同时把这个值传送到 16 位的 DS 寄存器。寄存器寻址方式也可以使用 8 位的源寄存器和 8 位的目的寄存器。例如,

```
MOV  DL, AL
```

在八种主要寻址方式中,立即寻址和寄存器寻址两种方式所需的时钟最少。因为操作数的数据已经包含在指令之内,操作数也已在内部放好,所以省去了访问外部寄存器或外部设备所需的时间。

另外六种寻址方式的执行时间长一些。这是因为微处理机必须根据段地址、段偏移量、基址寄存器或者变址寄存器的内容计算操作数的地址。计算出来的操作数地址就是有效地址,或叫 EA。

三、直接寻址

在 16 位操作数情况下,所谓直接寻址,就是操作数的 16 位段偏移量已包含在指令之内。偏移量加上移动后的数据段 DS 寄存器的内容就返回一个 20 位的 EA。它就是物理地址。通常,直接寻址的操作数是一个标号。

举例来说,图 1.5 中的指令是把由标号为 MYDATA 的存储单元的内容装到 AX 寄存器中去。MYDATA 是由两个相邻存储单元组合而成。请注意在序号低的存储单元内存放操作数的低位,在序号高的存储单元内存放高位。

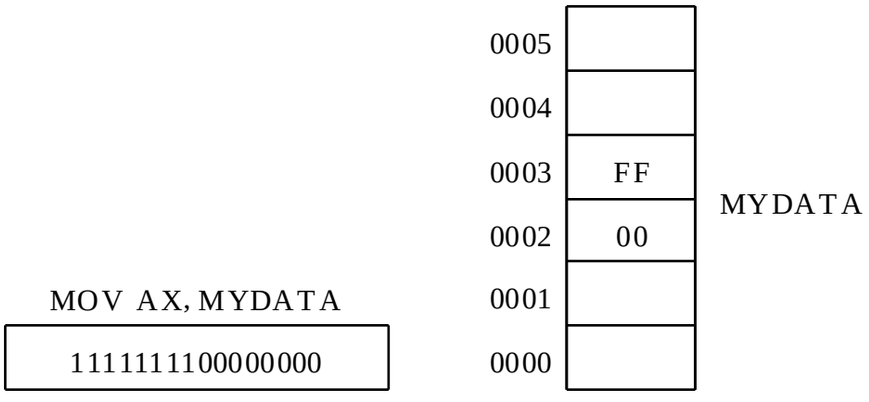

图 1.5　直接寻址

四、寄存器间接寻址

所谓寄存器间接寻址,就是由一个标号代替了要访问的源操作数的地址。操作数的值是由保存在以下几个寄存器内的偏移量地址指出。这些寄存器是源变址寄存器 ESI、目的变址寄存器 EDI、基址寄存器 EBX 或基址指针 EBP。

微处理机是通过指令的语法来识别寄存器间接寻址的。源操作数的标志符用方括号[　]括起。在图 1.6 中,只有当寄存器 BX 内装入 MYDATA 的偏移量地址时才能进行,且还要使用 OFFSET 操作符或使用装入有效地址指令 LEA 才能完成操作。如:

```
MOV   BX,OFFSET MYDATA
LEA   BX,MYDATA
```

用寄存器间接寻址可对以表格形式存放的数据进行访问。这样,在访问某个单个值时就变得更加有效。虽然由于使用基址寄存器和访问存储单元增加了一些有效存取周期,但不必因为再从存储器内取一个地址,然后再访问这个源操作数而花费更多的周期。

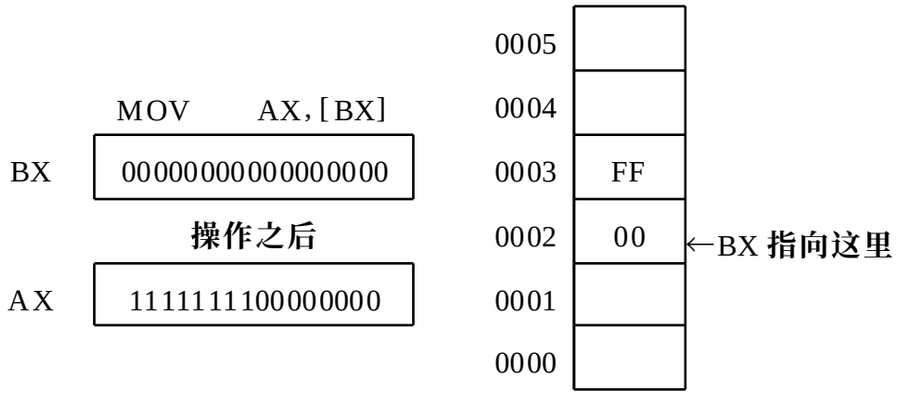

图 1.6　寄存器间接寻址

五、基址相对寻址

使用基址相对寻址得到操作数的有效地址,是由相对于被选中段的基址寄存器(或者 BX,或者 BP)的内容与位移量求和获得的。基址相对寻址方式在访问复杂数据结构的记录时经常使用。基址寄存器指向数据结构的基址,而位移量所选中的则是它的一个特殊字段。改变位移量的值就可访问记录内的不同字段。而访问不同记录内的同一字段,只须改变基址寄存器的内容。

图 1.7 中部分代码 MESGE1 内含有一个字符串。LEA 指令把偏移量地址装进 BX 寄存器。若欲访问 MESGE1 内的第五个元素,则可以通过把 MESGE1 中的基址(BX)与字符串的位移量+ 4 相加,完成对 MESGE1 中第五个元素访问的目的。

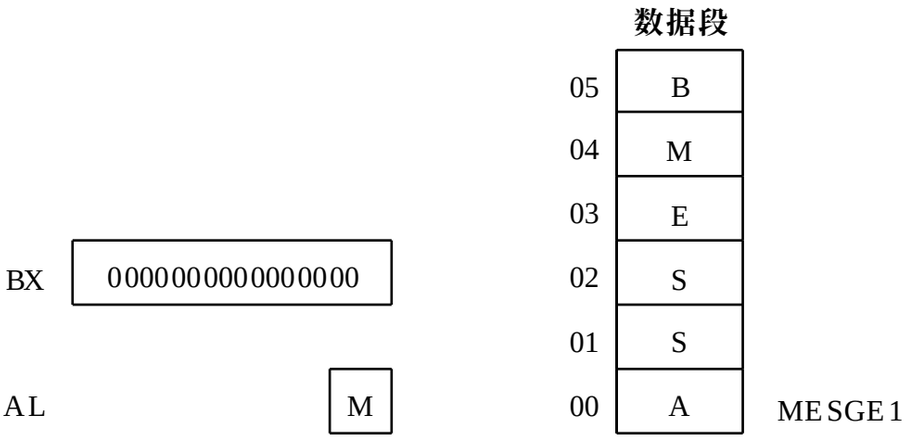

图 1.7　基址相对寻址

汇编程序可以识别下面三种基址相对寻址的表示方法:

```
LEA   [BX]+ 4
LEA   4[BX]
```

```
LEA   [BX+ 4]
```

其中第一种是经常使用的语法,而且位移量可以在基址寄存器的前头,或者把位移量括在方括号[ ]之内。给位移量增值经过 MESGE1 传送也能完成,或者使用下面这条指令:

```
LEA   BX, MESGE1
```

改变基址[BX]的值,达到改变所访问的信息的目的。

六、基址变址寻址

所谓基址变址寻址,就是操作数位于所选的段内,其偏移量为下列内容之和:基址寄存器的内容加变址寄存器的内容,并且再加上一个可供选择的位移量。如果位移量没有包括在内,基址变址寻址方式常用来访问一个动态数组(即数组基址在程序执行期间可以改变)内的元素。如果位移量包括在内则将允许访问数组中的各个元素,而这个数组则是结构中的一个段,例如一个记录。

如图 1.8 所示,基址寄存器指向记录结构的基址,而存放在 DI 寄存器中的位移量包含记录开始点到数组字段起始点的距离,而且元素位移量应该包含在 ELEMENT 变量内。假设在变量 ELEMENT 内包含有偏移量为 02H,图 1.8 中展示出了在指针情况下的基址变址寻址过程。

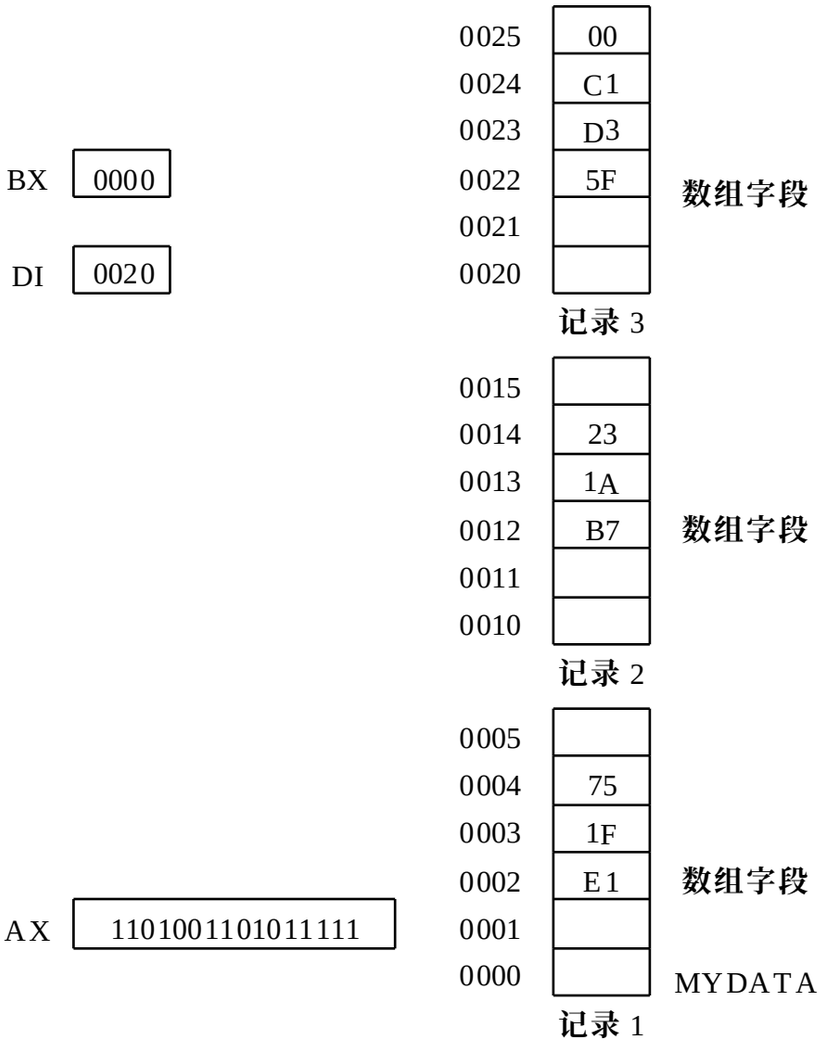

图 1.8　基址变址寻址

在图 1.8 内,记录结构的基址存放在寄存器 BX 内,其值为 0000。第三个记录的数组字段含有 0020 的位移量,并且存放在 DI 之内。用含有 ELEMENT 中的初始值位移量访问数组字段中的第三个元素。

七、直接变址寻址

在直接变址寻址时,操作数的偏移量地址是将位移量与所选中段内的变址寄存器(SI 或 DI)的内容相加得到的。直接变址寻址常用于访问静态数组元素。把位移量的值固定在数组的开始,而变址寄存器则选择此结构中的一个单元素。不像记录那样各字段的宽度和类型可以变化,而数组中的数组元素则是同一类型的。因为数组元素的数据类型和大小规模都相同,所有数组的传送问题变成有规律地增加或减少位移量的问题。例如,

```
MOV   SI, 2
MOV   AL, ARAY1[SI]
```

根据数组元素的数据类型必须仔细地选择合适的位移量的值。上例就是把 ARAY1 中的第三个值装到 AL 寄存器中。又如:

```
MOV   AX, ARAY1[SI]
```

如图 1.9 所示,以上这条语句的作用是把一个 16 位值装到 AX 寄存器内。根据如图 1.9 所示的数组的数据类型而采用的操作方式。

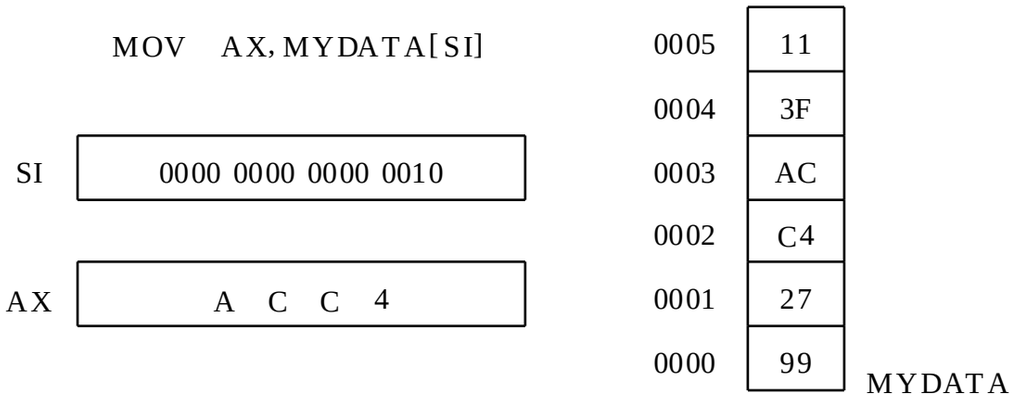

图 1.9　直接变址寻址

八、扩充寻址方式

32 位寻址方式被扩充成准许把任何一个寄存器作为基址寄存器或变址寄存器。32 位寻址方式要求,如果使用基址和变址寄存器,其内保存的必定是有效的 32 位值,任何一个 16 位寻址方式的指令都截断 32 位寄存器的内容。所以凡超过 16 位的都被忽略掉了。

第三节　程序设计风格

一个用汇编语言编写的程序实际上就是一个可执行的语句序列。语句序列告诉汇编程序要完成的操作。这个语句序列就是源代码。像其它语言一样,汇编语言也有语法。

每个汇编语言语句都由四个字段构成。它们是:名字字段(Name Field)、操作字段