教育部高职高专规划教材

高职高专现代信息技术系列教材

汇编语言程序设计

梁发寅　宗大华　编著

人 民 邮 电 出 版 社

高职高专现代信息技术系列教材

编 委 会 名 单

图书在版编目（CIP）数据

汇编语言程序设计/梁发寅，宗大华编著. —北京：人民邮电出版社，2004.3
（高职高专现代信息技术系列教材）
ISBN 7-115-12060-9

I. **汇**… II. **①梁**… **②宗**… III. **汇编语言—程序设计—高等学校：技术学校—教材**
IV. TP313

中国版本图书馆 CIP **数据核字（**2004**）第** 008344 **号**

内 容 提 要

汇编语言是一种面向机器的符号式程序设计语言。汇编语言具有编程质量高、执行速度快、占用存储空间少、易记、易修改等优点。

本书以 8086/8088 **汇编语言格式的指令为出发点，分** 7 **章进行讲述。本书认为寻址方式是学习汇编语言的基础，学习中应强调程序设计的各种结构，并认真做好上机实践。**

本书适用于高职高专计算机及相关专业的学生。编写时，力求做到突出基础知识和必备知识；由浅入深地安排全书内容；多举实例，用例子说明概念及各种编程方法；每章最后附有习题，帮助理解和巩固所学内容。

相信本书对入门、理解、初步掌握汇编语言以及学习汇编语言的编程方法，都会有所裨益。

教育部高职高专规划教材

高职高专现代信息技术系列教材

汇编语言程序设计

◆ **编　　著　梁发寅　宗大华**
责任编辑　潘春燕

◆ **人民邮电出版社出版发行**　　**北京市崇文区夕照寺街** 14 **号**
邮编　100061　　**电子函件**　315@ptpress.com.cn
网址　http://www. ptpress.com.cn
读者热线：010-67194042
北京汉魂图文设计有限公司制作
北京隆昌伟业印刷有限公司印刷
新华书店总店北京发行所经销

◆ **开本：**787×1092　1/16
印张：12.25
字数：287 **千字**　　2004 **年** 3 **月第** 1 **版**
印数：1 – 5 000 **册**　　2004 **年** 3 **月北京第** 1 **次印刷**

ISBN 7-115-12060-9/ TP · 3821

定价：17.00 **元**

本书如有印装质量问题，请与本社联系　电话：（010）67129223

关于本书

汇编语言是一种面向机器的程序设计语言，是人与计算机之间交换信息的有力工具。汇编语言一方面保持与机器指令一一对应；另一方面用助忆符代替机器指令中的操作码，用符号地址代替机器指令中的地址码。因此，使用汇编语言编写程序，不仅继承了用机器语言编写程序时具有的编程质量高、执行速度快和占用存储空间少的优点，而且还克服了机器语言程序直观性差、难学、难记、难检查及难修改等缺点。

本书以汇编格式的 8086/8088 指令为出发点，共分 7 章进行讲述。第 1 章由两个部分内容组成：首先介绍与汇编语言有关的计算机基础知识，然后介绍汇编语言的各种寻址方式。寻址方式是学习汇编语言的基础。第 2 章把 8086/8088 的指令划分成 6 组，并按组对它们的格式、功能做了介绍。第 3 章讲述汇编语言各种语句的编写格式和程序的编写格式。第 4 章介绍程序设计中的子程序结构，突出返回地址的保存、现场保护和现场恢复等程序设计方面的安排。第 5 章介绍程序设计中的循环结构，重点为循环控制条件的各种构成方法。第 6 章介绍程序设计中的分支结构，重点为如何按条件对分支进行判断。第 7 章为上机实践，介绍汇编语言的工作环境以及编写源程序、汇编、连接、调试以及运行的全过程。

本书是为高职高专计算机及相关专业学生编写的汇编语言教材，因此在编写时，力求做到如下几点：

1．突出对基础知识、必备知识的介绍，避免面面俱到；

2．以由浅入深、螺旋式上升的方式安排全书内容，把各种知识一步步地灌输给读者；

3．多举实例，用例子说明概念、说明各种编程的方法；

4．每章最后附有一定数量的习题，以帮助读者理解和巩固所学的内容。

在本书的编写过程中，陈吉人、沈寄云和宗涛提供了很多实例，为每章所附习题的收集、调试做了很多的工作，在此表示诚挚的谢意！由于编者水平所限，书中难免出现谬误或不当之处，在此恳请读者不吝批评、指正！

编者

2004 年 3 月于北京

丛 书 前 言

江泽民总书记早在十五大报告中提出了培养数以亿计高素质的劳动者和数以千万计专门人才的要求，指明了高等教育的发展方向。只有培养出大量高素质的劳动者，才能把我国的人数优势转化为人才优势，提高全民族的竞争力。因此，我国近年来十分重视高等职业教育，把高等职业教育作为高等教育的重要组成部分，并以法律形式加以约束与保证。高等职业教育由此进入了蓬勃发展时期，驶入了高速发展的快车道。

高等职业教育有其自身的特点。正如教育部“面向 21 世纪教育振兴行动计划”所指出的那样，“高等职业教育必须面向地区经济建设和社会发展，适应就业市场的实际需要，培养生产、管理、服务第一线需要的实用人才，真正办出特色。”因此，不能以本科压缩和变形的形式组织高等职业教育，必须按照高等职业教育的自身规律组织教学体系。为此，我们根据高等职业教育的特点及社会对教材的普遍需求，组织高等职业学校有丰富教学经验的老师，编写了这套《高职高专现代信息技术系列教材》。

本套教材充分考虑了高等职业教育的培养目标、教学现状和发展方向，在编写中突出了实用性。本套教材重点讲述目前在信息技术行业实践中不可缺少的、广泛使用的、从业人员必须掌握的实用技术。即便是必要的理论基础，也从实用的角度、结合具体实践加以讲述。大量具体的操作步骤、许多实践应用技巧、接近实际的实训材料保证了本套教材的实用性。

在本套教材编写大纲的制定过程中，广泛收集了高等职业学院的教学计划，调研了多个省市高等职业教育的实际，反复讨论和修改，使得编写大纲能最大限度地符合我国高等职业教育的要求，切合高等职业教育实际。

在选择作者时，我们特意挑选了在高等职业教育一线的优秀骨干教师。他们熟悉高等职业教育的教学实际，并有多年的教学经验；其中许多是“双师型”教师，既是教授、副教授，同时又是高级工程师、认证高级设计师；他们既有坚实的理论知识，很强的实践能力，又有较多的写作经验及较好的文字水平。

目前我国许多行业开始实行劳动准入制度和职业资格制度，为此，本套教材也兼顾了一些证书考试（如计算机等级考试），并提供了一些具有较强针对性的训练题目。

对于本套教材我们将提供教学支持（如提供电子教案等），同时注意收集本套教材的使用情况，不断修改和完善。

本套教材是高等职业学院、高等技术学院、高等专科学院教材。适用于信息技术的相关专业，如计算机应用、计算机网络、信息管理、电子商务、计算机科学技术、会计电算化等。也可供优秀职高学校选作教材。对于那些要提高自己应用技能或参加一些证书考试的读者，本套教材也不失为一套较好的参考书。

最后，恳请广大读者将本套教材的使用情况及各种意见、建议及时反馈给我们，以便我们在今后的工作中，不断改进和完善。

目　　录

第 1 章　汇编语言基础知识

汇编语言是一种符号语言，它不仅便于记忆，而且能够直接与计算机硬件对话。本章讲述学习汇编语言之前应该具备的基础知识。首先假定你已经具有了二进制数的概念，即知道有关数的进制问题，知道二进制数与十进制数之间如何转换，知道在计算机中为什么要引入八进制和十六进制，知道二进制数与八进制数、十六进制数之间如何转换，等等。从这些知识出发，本章将主要涉及以下 5 个方面的内容：

（1）在计算机中，有符号数是以补码的形式、字符是以 ASCII 码的形式存放在存储器里的。

（2）存储器单元的内容是用存储器单元的地址来表示的。

（3）在字长为 16 位的计算机中，是通过对存储器地址分段的方法，用“段地址：偏移地址”这种方式，表示 20 位的存储器地址的。

（4）介绍在 CPU 中数据寄存器、段寄存器、指针及变址寄存器的功能，以及它们在编写汇编语言源程序时的用途。

（5）介绍 8086/8088 的寻址方式，着重阐明与数据有关的 7 种寻址方式。

1.1　有符号数和无符号数

1.1.1　有符号数的补码表示

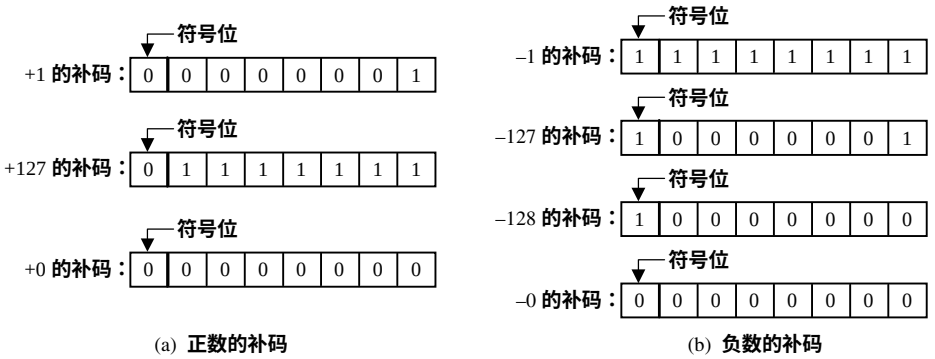
计算机中的数是用二进制来表示的，其正、负号也用二进制数来表示。一般用最高有效位来表示数的符号，即用 0 表示正号，用 1 表示负号。把一个数的符号加以数值化，称这样的数为机器数。在计算机内，可以用不同的码制来表示一个机器数，常见的有原码、反码和补码三种表示方法。但大多数计算机都采用补码表示法。

在补码表示法中，正数 x 的补码就是 x 的机器数。比如，当字长为 8 个二进制位时，用最高位取值为 0 表示正号，后面七位表示数值，就构成了正数 x 的补码。这样，$[+1]_{补}=0000001$，$[+127]_{补}=01111111$，$[+0]_{补}=00000000$，如图 1-1（a）所示。

负数 x 的补码由 $(2n-|x|)$ 表示。比如，当字长为 8 个二进制位时，–1 的补码应该表示成 (2^8-1)（这里，1 是–1 的绝对值），即 256–1=255。由于 255 用 8 个二进制位表示时，是 11111111，因此，$[-1]_{补}=11111111$。同样地，$[-127]_{补}=2^8-127=256-127=10000001$。这样，$[-128]_{补}=2^8-128=256-128=10000000$，$[-0]_{补}=2^8=00000000$。所以，10000000 这个数，在补码表示法中被定义为–128。如图 1-1（b）所示。这样，8 位补码能表示数的范围是：

$$-128 \leq N \leq +127$$

这个范围用 16 进制数表示时是：80H≤N≤7FH。



(a) 正数的补码　　(b) 负数的补码

图 1-1　字长为 8 个二进制位时数的补码表示

类似地，当字长 n=16 时，$[+1]_{补}$=0000000000000001，$[-1]_{补}$=1111111111111111。由此可以看出，将一个数的补码从 8 位扩展到 16 位（进而从 16 位扩展到 32 位）时，只需将符号位扩展便可。正数的符号位扩展应在前面补 0，负数的符号位扩展应在前面补 1。因此，16 位补码能表示的数的范围应该是：

$$-32768 \leq N \leq +32767$$

用 16 进制数表示时是：8000H≤N≤7FFFH。一般地，n 位补码能表示的数的范围是：

$$2^{n-1} \leq N \leq 2^{n-1}-1$$

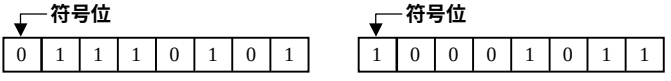
表 1-1 列出了当 n=8 和 n=16 时，以补码形式表示的数的范围。

表 1-1　　补码形式下数的表示范围

十进制	二进制	十六进制	十进制	十六进制
n=8			n=16	
+127	01111111	7F	+32767	7FFF
+126	01111110	7E	+32766	7FFE
…	…	…	…	…
+2	00000010	02	+2	0002
+1	00000001	01	+1	0001
0	00000000	00	0	0000
–1	11111111	FF	–1	FFFF
–2	11111110	FE	–2	FFFE
…	…	…	…	…
–126	10000010	82	–32766	8002
–127	10000001	81	–32767	8001
–128	10000000	80	–32768	8000

例 1-1　求 117 和–117 的 8 位补码。

解：117 是一个正数，它的数值用 7 个二进制位表示时，应是 1110101。所以$[+117]_{补}$=01110101，如图 1-2（a）所示。$[-117]_{补}$=256–117=139，它的二进制表示为 10001011，所以，$[-117]_{补}$应该是 10001011，如图 1-2（b）所示。



(a) 117 的补码形式　　(b) –117 的补码形式

图 1-2　117 和–117 的补码形式

求负数的补码，也可以用下述的简便方法：先写出与其绝对值相等的正数的补码，然后按位求反（1 变 0，0 变 1），最后再在末位加 1。比如，求–117 的补码，步骤是：

先求正数 117 的补码为：　01110101
然后按位求反，结果为：　10001010
最后末位加 1，结果为：　10001011

可以看出，10001011 正好就是–117 的补码。

对任何一个二进制数，按位求反后末位加 1 的运算称为求补运算。可以证明，以补码表示的数具有以下性质：

性质 1： $[+X]_{补} \xrightarrow{求补运算} [-X]_{补} \xrightarrow{求补运算} [+X]_{补}$

性质 2： $[X+Y]_{补}=[X]_{补}+[Y]_{补}$

性质 3： $[X-Y]_{补}=[X]_{补}-[Y]_{补}$

性质 1 表示对一个正数的补码执行求补运算，其结果恰是与其绝对值相同的负数的补码。同样地，对一个负数的补码执行求补运算，其结果恰是与其绝对值相同的正数的补码。性质 2 表示两个数和的补码，等于这两个数补码的和。性质 3 表示两个数差的补码，等于这两个数补码的差。

要从负数的补码中找到它所对应的十进制数，往往是利用性质 1 对负数的补码执行求补运算，得到正数的补码后，再计算出所要求的十进制数。

1.1.2　补码的加法和减法

在计算机中采用补码表示数，会使计算机的加、减运算非常简便。对于加法，不必去判断数的正负号，将符号位一起参加运算就可以得到正确的结果；对于减法，可将减数求补后变成加法进行运算，同样得到正确的结果。下面举例说明。

例 1-2　求 5–3。

解：这是减法，根据上面所述，可将减数–3 **求补后，做加法运算，即：**

	$[+5]_{补}+[-3]_{补}$
5	5 的补码：　00000101
– 3	– 3 的补码：　+ 11111101
2	计算结果：1← 00000010 (+2 的补码)

例 1-3　求 25–32。

解：

	$[+25]_{补}-[+32]_{补}$	$[+25]_{补}+[-32]_{补}$
25	00011001	00011001
– 32	– 00100000	+ 11100000
–7	1→11111001	11111001 (–7 的补码)

1←表示进位（从最高有效位向高位的进位），1→表示借位。由于机器字长的限制，这种进位会自动丢失，借位会自动进行，不会影响到运算结果的正确性。计算机自动把每次运算所产生的进位或借位保存在特定的地方，以便于检查或由程序进行判断。

注意：由于受计算机字长的限制，当补码的加、减运算所得的结果超出原字长的表示范围时，即产生溢出，这时结果是错误的。

例 1-4　求 98+117。

解：

	$[+98]_{补}+[117]_{补}$
98	01100010
+ 117	+ 01110101
215	11010111 (**负数**)

这里，两个正数相加应是正数，但结果是负数，结果错。之所以会这样，是因为 215 已超出 8 位数补码的表示范围：$-128 \le N \le +127$。

例 1-5　求(–20)+(–117)

解：

	$[-20]_{补}+[-117]_{补}$
–20	11101100
+) –117	+ 10001011
–137	1← 01110111 (**正数**)

这里，两个负数相加应是负数，但结果是正数，结果错。之所以会这样，是因为–137 已超出 8 位数补码的表示范围：$-128 \le N \le +127$。

运算结果超出规定字长的补码所能表示的范围，我们称其为溢出。补码的加、减运算产生溢出时，结果都是错误的。注意：这种错误是因为受字长的限制所引起的，并不是加减运算的法则不对。如果我们将例 1-4 中表示数的位数扩展到 16 位，则有：

```
  0000000001100010
+ 0000000001110101
  0000000011010111 (00D7H)
```

正数 00D7H 正好是 215 的补码，结果是正确的。因此，当不能确定运算是否会产生溢出时，要选择适当的字长来扩大数的表示范围，以保证运算结果正确。比如，如果一个字长是 16 位，那么也可以用两个字长（32 位）来表示一个较大的数，称其为双字长数或双精度数，其格式如图 1-3 所示。

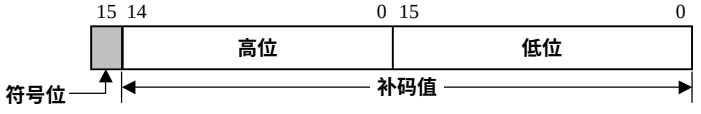


图 1-3　32 位双精度数的格式

其中高位字的最高有效位为符号位，其余的 31 位表示数值。也就是说，双精度数的低位字是无符号数。

1.1.3　无符号数

在某些情况下，如果要处理的数全是正数，那么再保留符号位就没有意义。如果利用它，就可以改变所表示数的范围。把最高位也当作数值处理，这种数称为无符号数。8 位无符号数表示的范围是：$0 \le N \le 255$。显然，它与有符号数时表示的范围 $-128 \le N \le 127$ 不同了。不难看出，16 位无符号数的表示范围应该是：$0 \le N \le 65535$，用十六进制数表示时为 0000H≤*N*≤FFFFH。

在计算机里，把 $2^{10}=1024$ 称为 1K，把 1024K 称为 1M（1 兆），因此，很容易得出：

$2^{16}=2^6\times2^{10}=64\times1024=64K$（65536），$2^{20}=2^{10}\times2^{10}=1024K=1M$（1048576）。

用十六进制数表示：1K=400H，1M=100000H。

在计算机中，存储器的地址是用无符号数表示的。上面提及的 32 位双精度数的低位字也是无符号数。把一个正数的最高位的 0 看成是正号，或把它看作是数值 0，这对于这个正数的值不会产生什么影响。因此，在某些情况下，正数也可以当作无符号数来处理。两个无符号数相加，如果最高位有进位，表示无符号数溢出。产生溢出，计算结果当然是不对的，引起这个错误的原因，上面已经给出了解释。例如：

```
  160        无符号数 160：    10100000
+ 117        无符号数 117：  + 01110101
  267        计算结果：      1←00010101（15）
```

此时，因为 267 已超出 8 位无符号数表示的范围：0～255。同样，如果扩展到 16 位进行运算，则结果是正确的。

在多数情况下，对有符号数和无符号数的处理是不一样的。

1.1.4　字符的 ASCII 码

计算机从键盘接收的信息，以及显示或打印输出的信息，都是以字符方式输入输出的。一个字符，在计算机内部用 8 个二进制位来表示，其中低 7 位是该字符对应的编码值，最高位用作校验。目前，最常用的一种字符编码称为字符的 ASCII 码，它是美国信息交换标准代码：American Standard Code for Information Interchange 的缩写。表 1-2 列出了在汇编语言中常用字符的 7 位 ASCII 码值（16 进制数表示）。

表 1-2　　常用字符的 7 位 ASCII 码（16 进制数表示）

字符	ASCII	字符	ASCII	字符	ASCII	字符	ASCII
响铃	07	0	30	A	41	a	61
换行	0A	1	31	B	42	b	62
回车	0D	⋮		⋮		⋮	
空格	20	8	38	Y	5A	y	7A
$	24	9	39	Z	5B	z	7B

图 1-4 给出了字符 A，a，0 和$的 ASCII 码值在内存中的表示形式。

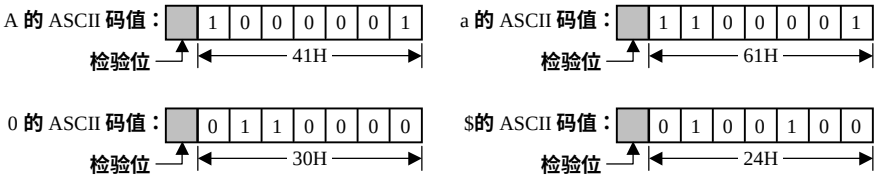


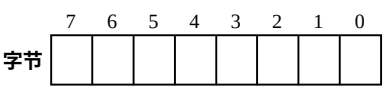
图 1-4　字符 A，a，0 和$的 ASCII 码值在内存中的表示形式

1.2 存　储　器

Intel 8086/8088 是两种第三代微处理器。本书的汇编语言是针对这两种微处理器的计算机编写的。这两种计算机的组成基本相同，由中央处理器 CPU、存储器和输入输出子系统 3 个主要部分组成，通过系统总线连接在一起。系统总线又称外部总线，包括数据总线、地址总线和控制总线。8086 的外部数据总线是 16 位，8088 的外部数据总线是 8 位，两者的区别仅此而已。它们的地址总线都是 20 位，可访问（取数或存数）内存储器的最大容量都是 1M 字节。注意：1M=1024K=2^{20}。

1.2.1 存储单元的地址和内容

在计算机中，每 8 个二进制位组成一个字节，每个二进制位给予一个编号，位编号如下图所示。



有时又把位编号由低到高用 D0，D1，D2，D3，D4，D5，D6 和 D7 位来表示。

存储器是以字节为单位存取信息的，存储器中的一个字节称为一个单元。为了正确地存放和读取信息，每一个字节单元都有一个地址，称为存储器的地址。地址从 0 开始编号，顺序地每次加 1。不难理解，地址是一个无符号整数。在书写单元的地址时，如果用 2 进制数表示，肯定很长。因此，通常都用十六进制数来书写。由于：

$$2^{16}=2^6\times2^{10}=64\times1024=64K$$

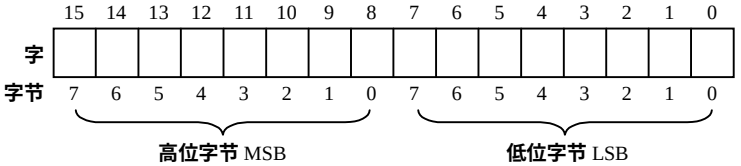
因此，16 位二进制数可表示 64K 个字节单元的地址，其地址编号的范围用十六进制数表示为

0000H～FFFFH

一个存储单元中存放的信息称为该存储单元的内容。图 1-5 所示为存储器里存放信息的情况。0004H 单元存放的信息为 34H，也就是说 0004H 单元的内容为 34H，表示为：

(0004H)=34H

前面已经提及，一个字节含 8 个二进制位。在 16 位字长的计算机中，大部分数据都是以字为单位表示的。一个字由两个字节组成，位编号如下图所示。



因此，一个字存入存储器时，要占用相继的两个存储单元。存放的原则是：低位字节存入低地址单元，高位字节存入高地址单元。也就是说，数据存放的次序和书写次序是相反的。存储器中两个字节单元组成一个字单元，字单元的地址采用它的低位字节单元的地址来表示。若某个字单元的两个字节单元的地址是 X 和 $X+1$，那么，这个字单元的地址应该是 X，而不是 $X+1$。这个字单元的内容应该写成(X)，它的是两个字节单元的内容应该写成$(X+1)(X)$，而不是$(X)(X-1)$。

如果把图 1-5 所示的 0004H～0005H 两个字节单元视为一个字单元，要把 1234H 存放在里面，那么在地址为 0004H 的单元里，存放的应该是低字节 34H；地址为 0005H 的单元里，存放的应该是高字节 12H。这个字单元的地址应该是 0004H，这个字单元的内容应该表示为

（0004H）=1234H

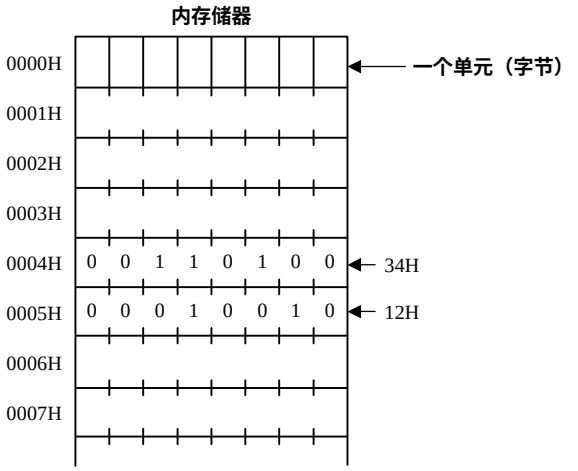


图 1-5　存储单元的地址和内容

由此可见，对于存储器，同一个地址既可以看成是字节单元的地址，也可以看成是一个字单元的地址。一般来说，字单元的地址可以是偶数，也可以是奇数。在 8086 处理器的计算机中（16 位外部数据总线），是以偶地址访问存储器的，所以要取一个奇地址中的数据，CPU 要访问两次存储器。

综上所述，下面 3 点是汇编语言的基础，必须牢记。

（1）用存储器单元的地址外面加括号的方法，来表示一个存储单元的内容。例如：(0004H)=34H，或(0004H)=1234H。

（2）一个字存入存储器要占用相继的两个存储单元，存放时，低位字节存入低地址单元，高位字节存入高地址单元。

（3）同一个存储单元的地址既可看成是字节单元的地址，又可看成是字单元的地址。字单元的地址采用它的低字节单元的地址来表示。

1.2.2　存储器地址的分段

在 8086/8088 处理器的计算机中，内存储器的最大容量为 1M 字节。由于：

$$1M=1024K=2^{20}$$

因此，在 1M 字节的存储器中，每个存储器单元的地址应该用 20 个二进制位来表示。若用十六进制数来表示这 1M 字节的地址范围，那么应该是 00000H～FFFFFH。这就是说，每个存储器单元有一个惟一的 20 位地址，称为该存储器单元的物理地址。CPU 是用物理地址对存储器进行访问的。

前面已经说过，16 位二进制数所能表示的地址范围是 0000H～FFFFH，而 1M 字节的存储器地址范围是 00000H～FFFFFH。那么在 16 位字长的计算机中，用什么方法来表示 20 位地址呢？这就要借助于存储器地址分段来解决。具体的做法如下。

在 1M 字节的内存地址范围内，从 00000H 地址开始，每 16 个字节划分成为一小段。那么，在 00000H～FFFFFH 地址范围内，这样的小段总共有 64K 个。各小段的地址分布如下：

00000H，00001H，00002H，……，0000EH，0000FH
00010H，00011H，00012H，……，0001EH，0001FH
00020H，00021H，00022H，……，0002EH，0002FH
⋮
FFFE0H，FFFE1H，FFFE2H，……，FFFEEH，FFFEFH
FFFF0H，FFFF1H，FFFF2H，……，FFFFEH，FFFFFH

其中，第一列的地址便是每个小段的首地址。分析这些首地址，可以看出它们有这样的共同特点：十六进制数表示的地址中，最低一位是 0。如果把它们展开成二进制表示，意味着这些 20 位地址中的最低 4 个二进制位为 0000。

编写汇编语言程序时，根据需要可以将 1M 字节的地址分为若干个段，规定每个段的起始地址必须从任意一小段的首地址开始，每个段的大小可达 64K。注意：64K 这个范围内的地址，正好可以用 16 位二进制数来表示。由于段的起始的地址最低 4 位为 0，于是可以用一个字来存放它的高 16 位，暗中记住它的最低 4 位是 0，我们把段起始地址的高 16 位称为段地址；把段内各个单元相对于段起始地址的偏移值的低 16 位(高 4 位为 0)称为该单元的偏移地址。这样，20 位的物理地址可由 16 位的段地址和 16 位的偏移地址组成。物理地址的计算方法如图 1-6 所示。

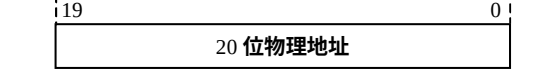


图 1-6　由两个 16 位的字拼装成一个物理地址

也就是说，把 16 位的段地址左移 4 位，再加上 16 位的偏移地址值就形成物理地址。即

10H×段地址+偏移地址=物理地址

因此，通常把物理地址用段地址和偏移地址来表示，写成：

段地址：偏移地址

用这种形式表示的地址称为逻辑地址，它与物理地址是一一对应的。

例 1-6　已知段地址和偏移地址为 301AH：008AH，试问它的物理地址是什么？

解：16 位的段地址 301AH 左移 4 位（也就是乘 16），得到 301A0H。加偏移地址 008AH，

即：

$$\begin{array}{r} 301A0H \\ +\quad 008AH \\ \hline 3022AH \end{array}$$

于是，它对应的物理地址是 3022AH。

编写汇编语言程序时，通常将程序所需要的存储空间分成代码段、数据段、堆栈段和附加段 4 种类型。代码段存放当前正在运行的程序；数据段存放当前运行程序所需要使用的数据；堆栈段给出了堆栈所在的区域（“堆栈”是一种特殊的存储区，要以先进后出的方式来访问该区域里的数据）；附加段是附加的数据段，是一个辅助的数据区。

在 8086/8088 处理器中，有 4 个专门用来存放段地址的寄存器，它们的名字分别是 CS（代码段）、DS（数据段）、SS（堆栈段）和 ES（附加段）寄存器。每个段可以独立地占用 64K 存储区，各段存储区也可以重叠，即每个段的大小允许根据需要来分配，不一定要占 64K。当然，每个存储单元的内容决不能冲突。一般情况下，各段的段地址是由操作系统负责分配的。

例 1-7　若用户程序的分段情况为：代码段需要 8K(2000H)存储区，数据段需要 2K(800H)存储区，堆栈段需要 256 个字节的存储区。假定操作系统从 02000H 单元开始分配存储区，则此时分段情况如图 1-7 所示。

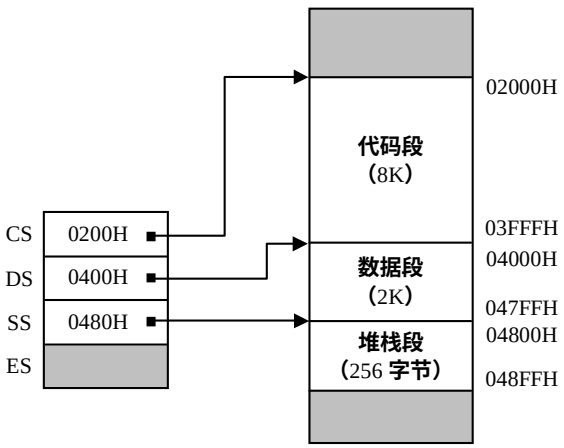


图 1-7　例 1-7 的分段情况图

解：由于代码段的段起始地址是 02000H，因此 CS=0200H，8K 存储区的字节单元的个数是 2000H 个，因此，代码段的地址范围是 02000H～03FFFH。代码段结束后的第一个小段的首地址 04000H 就作为数据段的起始地址，因此 DS=0400H，2K(800H)的数据段的地址范围是 04000H～047FFH。同理，SS=0480H，堆栈段的地址范围是 04800～048FFH。这样，代码段、数据段和堆栈段重叠在一起了（共在 64K 段的范围内）。注意：每个存储单元的内容是不允许发生冲突的。例如，当向数据段存放数据时，指定存入的偏移地址不能超过 47FFH，否则会将数据存到堆栈段去，这是绝对不允许的。

图 1-8 给出了各存储段中物理地址和偏移地址之间的对应关系。从图 1-8 中可以看出，各段的段地址确定后，段内各存储器单元的物理地址与偏移地址就一一对应了，而且各段的偏移地址都是从 0000H 单元开始的。从代码段、数据段等的偏移地址可以看出，有部分偏移