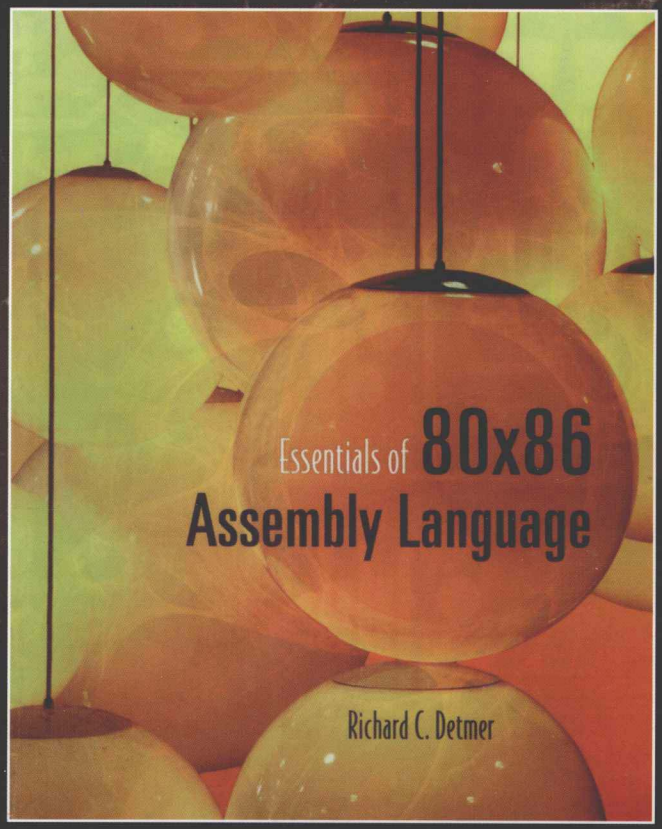
计 算 机 科 学 丛 书

HZ BOOKS
华章教育

Jones and Bartlett

# 80x86汇编语言基础教程

(美) Richard C. Detmer 著 郑红 陈丽琼 译



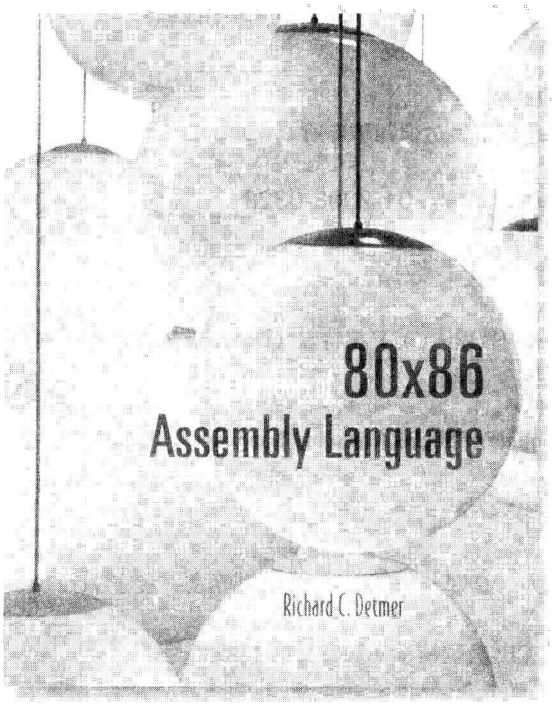

## Essentials of 80x86 Assembly Language

机械工业出版社
China Machine Press

计　　算　　机　　书

# 80x86汇编语言基础教程

(美) Richard C. Detmer 著　郑红 陈丽琼 译




**Essentials of 80x86 Assembly Language**

机械工业出版社
China Machine Press

本书主要针对 Intel 80x86 体系结构介绍汇编语言知识，强调基本的 80x86 整型指令，同时也介绍了浮点型结构，内容基本而完整。通过本书的学习，学生可以使用微软的 MASM 汇编器汇编 32 位平面存储模式程序，并在微软的 Windbg 调试器控制下跟踪程序指令的执行，了解计算机内部存储器和寄存器内容的变化。本书适合用作汇编语言课程的教材，同时也是计算机组成和体系结构课程的很好的补充教材。



**本书版权登记号：图字：01-2008-0328**

**图书在版编目（CIP）数据**

80x86 汇编语言基础教程 /（美）德特默（Detmer，R.C.）著；郑红，陈丽琼译．—北京：机械工业出版社，2009.1
（计算机科学丛书）
书名原文：Essentials of 80x86 Assembly Language

ISBN 978-7-111-25382-2

Ⅰ.8… Ⅱ.①德… ②郑… ③陈… Ⅲ.汇编语言－程序设计－教材 Ⅳ.TP313

中国版本图书馆 CIP 数据核字（2008）第 179283 号

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）
责任编辑：刘立卿
北京慧美印刷有限公司印刷
2009年3月第1版第1次印刷
184mm×260mm • 13.75印张
标准书号：ISBN 978-7-111-25382- 2
ISBN 978-7-89482-851-4（光盘）
定价：35.00元（含光盘）

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
本社购书热线：（010）68326294

# 出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章分社较早意识到“出版要为教育服务”。自1998年开始，华章分社就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章分社欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：


华章网站：www.hzbook.com

电子邮件：hzjsj@hzbook.com

联系电话：（010）88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037



华章教育

# 前 言

许多计算机组成原理或计算机体系结构的书都提供一些通用的知识，但很少或几乎没有涉及亲身实践一个具体的计算机体系结构。本书对于那些希望为学生提供实际操作 Intel 80x86 体系结构经验的老师来说，是一本很好的补充教材。通过本书的学习，学生能够使用微软的 MASM 汇编器（本书附带）汇编 32 位、平面存储模式的程序。本书也可单独作为汇编语言课程的教科书。学生可以在微软的 Windbg 调试器（本书附带）控制下执行程序，通过跟踪程序指令的执行，透视计算机内部来观察存储器和寄存器内容的变化。

本书强调基本的 80x86 整型指令，但是也介绍了浮点型结构。本书将涉及以下主题：

- 80x86 整型数的表示
- 80x86 内存寻址
- 80x86 寄存器
- 汇编语言的语法
- 操作码和指令格式
- 在 Windbg 下汇编和运行程序
- 数据复制指令
- 整型数的加法指令和减法指令
- 整型数的乘法指令
- 整型数的除法指令
- “与”、“或”及“异或”指令
- 移位指令和循环指令
- 无条件转移指令和条件转移指令
- 80x86 堆栈，压入指令和弹出指令
- 子程序包，调用指令和返回指令
- 80x86 浮点数的表示
- 80x86 浮点寄存器
- 部分 80x86 浮点数指令

## 风格和教学

本书主要通过示例教学。早在第 2 章，本书就给出了一个完整的汇编语言程序，并且在学生能够理解的层次上，仔细地考查了程序的各个部分。随后的章节包含了许多汇编语言代码的例子，同时，对一些新的或者难理解的概念给出了恰当的解释。

本书使用了大量的图表和例子。给出许多“指令执行前”和“指令执行后”的例子来讲解指令。每章节的后面有练习，简答题加深了对所学内容的理解，编程题让学生有机会将书中的内容运用到汇编语言编程中。

## 软件环境

“标准”80x86 汇编器是微软的宏汇编器（MASM），版本为 6.11。尽管该汇编器生成的代码用于 32 位的平面存储模式编程，非常适合 Windows 95、Windows NT 或者 32 位的微软操作系统环境，但是，与该软件包对应的链接器和调试程序并不适合在这样的系统环境中使用。本书附带一张光盘，包含 MASM（ML）的汇编程序、最新的微软链接器、32 位的全屏调试程序 Windbg（也来自于微软）以及必要的支持文件。该软件包为生成和调试控制台应用程序提供了一个良好的环境。

## 学习指南

本书的补充内容包括一个教师指南，该指南提供了一些教学提示和许多习题的答案。采用本书作为教材的老师，可向出版中文版的出版社提供申请，索取该教师指南。另外，如果有问题或者建议，可通过 rdetmer@mtsu.edu 联系本书的作者。

# 目 录

# 第 1 章　计算机中数的表示

用 Java 或 C++ 等高级语言编程时，要用到许多不同类型的变量（比如整型、浮点型或者字符型），变量一旦声明，就不需要考虑数据在计算机中是如何表示的。然而，用机器语言编程时，就必须考虑数据存储的位置和方式。本章将讨论 80x86 微处理器数据存储和处理的位置，以及常用的数据表示方式。

## 1.1　二进制数和十六进制数

计算机用**位**（bit）（二进制数制用 0 或 1 表示不同的电子状态）来表示数值。以 2 为基数，数字 0 和 1 表示**二进制**数。二进制数跟十进制数很相似，但二进制相应的权（从右到左）依次为 1、2、4、8、16 和 2 的更高次幂等，而十进制相应的权为 1、10、100、1000、10000 和 10 的更高次幂等。例如，二进制数 1101 可表示十进制数 13。

| 1 | | 1 | | 0 | | 1 | | |
|---|---|---|---|---|---|---|---|---|
| one 8 | + | one 4 | + | no 2 | + | one 1 | = | 13 |

二进制数很长，在读写时很不方便，比如，八位二进制数 11111010 表示十进制数 250，15 位二进制数 111010100110000 表示十进制数 30 000。而用**十六进制**（基数 16）表示时，大约只需要用到相应二进制表示的四分之一长度的位数。十六进制与二进制的转换很容易，因此，十六进制的表示可以简化二进制的表示。十六进制需要 16 个数字：其中 0、1、2、3、4、5、6、7、8 和 9 与十进制数相同；A、B、C、D、E 和 F 等同于十进制的 10、11、12、13、14 和 15。另外，这几个字母不论是小写还是大写都可用于表示数。

十六进制数中的权值对应 16 的幂，权值从右到左依次是 1、16、256 等等。十六进制数 9D7A 按如下方法计算：

```
   9    ×    4096      36864    [ 4096 = 16^3 ]
+ 13    ×    256        3328    [ D is 13, 256 = 16^2 ]
+  7    ×    16          112
+ 10    ×    1            10    [ A is 10 ]
= 40314
```

得出表示的是十进制的 40314。

图 1-1 列举了一些二进制、十六进制和十进制表示的数。读者应该熟记这些数，或者能够很快地推导出来。

上面的例子给出了如何将二进制数或十六进制数转换为十进制数。那么如何将十进制数转换成二进制数或者十六进制数呢？以及如何将二进制数转换为十六进制数呢？下面将介绍如何手工实现转换。通常情况下，可用一个具有二进制、十进制和十六进制转换功能的计算器很容

| 十进制 | 十六进制 | 二进制 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 10 |
| 3 | 3 | 11 |
| 4 | 4 | 100 |
| 5 | 5 | 101 |
| 6 | 6 | 110 |
| 7 | 7 | 111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| 10 | A | 1010 |
| 11 | B | 1011 |
| 12 | C | 1100 |
| 13 | D | 1101 |
| 14 | E | 1110 |
| 15 | F | 1111 |

图 1-1 十进制、十六进制和二进制数

易实现转换，这样，数制转换只不过是按一两个键而已。这种计算器可以像十进制那样进行二进制和十六进制的数学运算，而且具有很多其他的用途。注意：这种计算器用七个显示段来显示一个数字。比如，显示小写字母 b 时，看起来像数字 6，其他有些字符也可能难以辨认。

计算器不需要把一个十六进制数转换为对应的二进制形式。事实上，许多二进制数太长，一般的计算器不易显示，因此，每四位二进制数用一个十六进制数表示。其对应的二进制位如图 1-1 的第 3 列所示。如果位数不够四位，必要的时候前面用 0 补充。例如：

$3E8E2_{16}$=11 1011 1000 1110 $0010_2$

转换数字下标处的 16 和 2 表示基数。如果不易混淆，则这些下标处的数字可以忽略。二进制数补齐位数是为了增强可读性，如十六进制的数字 2 转换为二进制时，最前面用 0 补齐得到 0010。但由于二进制数前面的零不会改变该二进制数的值，所以上例中十六进制数的最高位 3 不需要转换为 0011。

把二进制数转换为十六进制数格式，则与上面的步骤正好相反。把二进制数从右向左每四位进行分隔，每四位二进制数用对应的十六进制数表示，例如：

1011011101001101111 = 101 1011 1010 0110 1111 = 5BA6F

前面介绍了如何将二进制数转换为十进制数。通常，一个很长的二进制数直接转换为十进制数并不采用这种方法。更快的方法是先将二进制数转换为十六进制数，再将该十六进制数转换为十进制数。以上面的 19 位二进制数为例：

$1011011101001101111_2$
$= 101\ 1011\ 1010\ 0110\ 1111$
$= 5BA6F_{16}$
$= 5\times65536+11\times4096+10\times256+6\times16+15\times1$
$= 375407_{10}$

下面给出十进制数转换为十六进制数的算法，该算法从右到左依次生成十六进制数位。该

算法用伪代码来描述，本书中其他的所有算法和程序都将采用伪代码描述。

```
until DecimalNumber = 0 loop
    divide DecimalNumber by 16, getting Quotient and Remainder;
    Remainder (in hex) is the next digit (right to left);
    DecimalNumber := Quotient;
end until;
```

**示例**

以十进制数 5876 转换为十六进制数的过程为例：

- 因为这是一个 until 循环，当第一次执行程序体的时候就进行循环控制条件检查。（为什么不用 while 循环？）
- 16 整除 5876（十进制数）

```
   367      商   新的十进制数的值
16|5876
  5872
  ----
     4      余数   最右边的十六进制数位
```

当前结果：4

- 367 不等于 0，再用 16 整除。

```
    22      商   新的十进制数的值
16|367
  352
  ---
   15       余数   生成的第 2 个十六进制数位
```

当前结果：F4

- 22 不等于 0，用 16 整除。

```
     1      商   新的十进制数的值
16|22
  16
  --
   6        余数   生成的下一个十六进制数位
```

当前结果 6F4

- 1 不等于 0，用 16 整除。

```
     0      商   新的十进制数的值
16|1
  0
  -
  1         余数   生成的下一个十六进制数
```

当前结果 16F4

- 当前的十进制数为 0，循环终止。最后结果为 $16F4_{16}$

**练习 1.1**

请根据下面给出的数字将表中空白的另外两种进制形式补充完整。

| | 二进制 | 十六进制 | 十进制 |
|---|---|---|---|
| 1. | 100 | _____ | _____ |
| 2. | 10101101 | _____ | _____ |
| 3. | 1101110101 | _____ | _____ |

| | | | |
|---|---|---|---|
| 4. | 11111011110 | _____ | _____ |
| 5. | 1000000001 | _____ | _____ |
| 6. | _____ | 8EF | _____ |
| 7. | _____ | 10 | _____ |
| 8. | _____ | A52E | _____ |
| 9. | _____ | 70C | _____ |
| 10. | _____ | 6BD3 | _____ |
| 11. | _____ | _____ | 100 |
| 12. | _____ | _____ | 527 |
| 13. | _____ | _____ | 4128 |
| 14. | _____ | _____ | 11947 |
| 15. | _____ | _____ | 59020 |

## 1.2 80x86 存储器

80x86 微机上的**随机存储器**（RAM）逻辑上可以看作是一个“条板单元”的集合，每个单元能存储一个**字节**（8 位）的指令或者数据。每个存储器字节都有一个 32 位的助记符，称为**物理地址**。一个物理地址通常用一个 8 位的十六进制数表示。第一个地址为 $00000000_{16}$，最后一个地址为无符号数的 $FFFFFFFF_{16}$。图 1-2 给出了一台 PC 中可能的存储器的逻辑图。由 $FFFFFFFF_{16} = 4\ 294\ 967\ 295$ 可知，PC 微机的存储器字节数可达到 4 294 967 296，即 4GB。

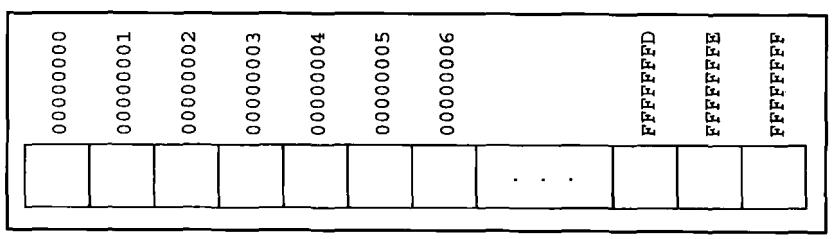



图 1-2 PC 存储器的逻辑图

在 80386 之前，Intel 80x86 处理器系列仅仅能够直接寻址的存储器为 $2^{20}$ 字节，使用 20 位物理地址，通常用 5 个十六进制数表示，范围从 00000 ～ FFFFF。

本书中的汇编语言程序使用**平面存储**模式。这意味着，逻辑上指向存储数据和指令的存储单元的地址实际上是用 32 位的地址编码的。

Intel 80x86 体系结构还提供了**分段存储**模式，早期的 8086/8088 CPU 只用这种模式。在 8086 / 8088 中，PC 存储器可看作是段的集合，每个段是 64KB，以 16 的倍数作为一个段的开始地址。也就是说，如果一个段的起始地址是 00000，另一个段（重叠第一个段）的起始地址就是 16（$00010_{16}$），下一个段的起始地址是 32（$00020_{16}$），其他段的起始地址依此类推。一个段的**段号**由其物理地址的前 4 个十六进制数组成。8086/8088 微机中的程序并不能使用 5 位的十六进制数地址，事实上，每个存储单元的定位取决于段号和从该段开始处的一个 16 位的**偏移量**。通常，程序只写出偏移量，段号可通过上下文来推断。偏移量是指从段的第一个字节到要定位地址的距离。在十六进制中，偏移量大小从 0000 到 $FFFF_{16}$。

从 80386 开始，80x86 系列处理器既有 16 位也有 32 位的分段存储模式。段号仍是 16 位长，

但不直接指向存储器中的一个段。事实上，段号仅仅是包含真正 32 位段的起始地址的表中的索引。在 32 位分段模式中，32 位的偏移量加上该段的起始地址可得出内存操作数的实际地址。对编程人员而言，段逻辑上是很有用的：在 Intel 的分段模式下，编程人员通常为代码、数据和系统堆栈分配不同的内存段。80x86 平面存储模式是真正的 32 位分段模式，所有的段寄存器包含相同的值。

事实上，当程序执行时，由程序产生的 32 位地址不一定是某个操作数存储的物理地址，操作系统和 Intel 80x86 CPU 有另外的存储管理层。**分页**（paging）机制用于将程序的 32 位地址映射成物理地址，当程序所产生的逻辑地址超过计算机实际的物理内存空间时，分页机制就非常有用了。如果程序太大而不能全部装入物理内存时，分页机制也可用于将部分程序在需要的时候从磁盘交换到内存中。当用汇编语言编程时，分页机制对用户是透明的。

**练习 1.2**

1. 假定微机的 RAM 是 256MB，则最后一个字节的 8 位十六进制数的地址是多少？
2. 假定一个微机的视频适配器预定的 RAM 地址是从 000C0000 到 000C7FFF，则其存储空间的字节数是多少？

## 1.3 80x86 寄存器

早期的 8086/8088 CPU 能够执行 200 多种不同指令。随着 80x86 系列扩展到 80286、80386、80486 以及 Pentium 处理器，CPU 可执行更多的指令。本书探讨如何用这些指令执行程序，从而理解机器级的计算机的性能。其他生产商生产的 CPU 也执行基本相同的指令集，因此，在 80x86 上编写的程序在这些 CPU 上不用改变也可运行。

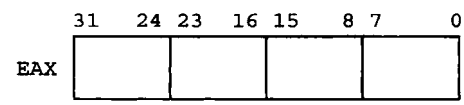
CPU 包含许多**寄存器**。访问每个内部寄存器要比访问 RAM 快得多。应用寄存器主要跟编程人员有关。一个 80x86 CPU（从 80386 开始）有 16 个应用寄存器。常用的指令可在寄存器与存储器间传输数据，也可对存储在寄存器或者存储器中的数据进行操作。所有的寄存器都有名字，一些寄存器有着特定的用途。下面将给出这些寄存器的名字，并详述它们的用途。

寄存器 EAX、EBX、ECX 和 EDX 称为**数据寄存器**或者**通用寄存器**。EAX 有时也是**累加器**，因为它用于存储许多计算的结果。下面是一条使用 EAX 寄存器的指令的例子：

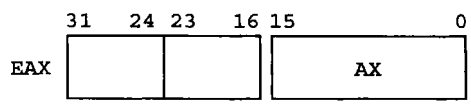
```
add eax , 158
```

将十进制数 158 与 EAX 中已有的数相加，用相加的和取代 EAX 中原来的数。（加法指令和下面提到的其他指令将在第 3 章详细讨论。）

寄存器 EAX、EBX、ECX 和 EDX 都是 32 位长，Intel 转换是以低位的 0 开始，从右向左，对数位转换。因此，如果把每个寄存器看成四个字节，则这些位用数表示为：




寄存器 EAX 整体上按照地址可分成若干部分。低位字位数从 0 ～ 15，就是常用的 AX 寄存器。

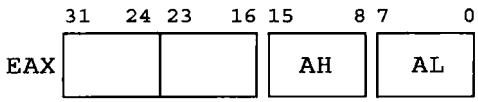


指令

```
sub  ax , 10
```

表示从存储在AX寄存器中的数中减去10，而EAX寄存器的高位数（16～31）没有任何改变。

同样，AX寄存器的低位字节（0～7位）和高位字节（8～15位）分别就是通常所说的AL和AH。




指令

```
mov  ah , 02ah
```

复制2A到8～15位，不改变EAX的其他的任何位。这里的操作数是十六进制而不是十进制。

寄存器EBX、ECX和EDX也有低16位的BX、CX和DX，它们又可按照高位和低位字节分别划分为BH和BL、CH和CL、DH和DL。BH、BL、CH、CL、DH和DL的每位的改变不会改变相应寄存器的其他位。要注意的是EAX、EBX、ECX、EDX的高位字并没有相应的名字——不能只使用名字引用16～31位。

8086到80286处理器有4个16位的通用寄存器，称之为AX、BX、CX和DX。增加“E”后表示“扩展”，将16位扩展成32位的80386寄存器。但是，80386以及后来的体系结构都有效地包含了以前的16位的体系结构。

另外还有4个32位的通用寄存器：ESI、EDI、ESP和EBP。事实上，可以用这些寄存器做算术之类的操作，但通常必须保留它们，用于特定的用途。ESI和EDI寄存器是**索引寄存器**，其中SI代表源索引，DI代表目的索引。它们可用于实现数组索引，并且在某些字符串操作中必须使用，但本书不讨论这些内容。

ESP寄存器是系统栈（内存中保留域）的**栈指针**，它很少直接通过程序来改变，但当数据入栈或者出栈时会改变。ESP寄存器在堆栈的用途之一就是过程（子例程）调用。过程调用指令地址紧跟在存储于栈中的过程调用指令之后，当调用返回时，这条指令地址就可从堆栈中取出。第5章将会详细探讨堆栈以及栈指针寄存器。

EBP寄存器是**基址寄存器**。通常，堆栈中被存取的数据项仅仅是存放在栈顶的数据项。然而，EBP寄存器除了标识栈顶位置外，也经常用于标识栈中的某一个固定位置，因此，在这个固定位置附近的数据可被访问。EBP也用于过程调用，尤其是带有参数的过程调用。

还有6个16位的**段寄存器**：CS、DS、ES、FS、GS和SS。在以前的16位分段存储器模式中，CS寄存器包含有代码段的段号，即当前正在执行的指令所存储的存储器区域。由于一个段是64K长，一个程序的指令集通常在64K的范围内；如果一个程序长度超过64K，则该程序在运行时，需要改变CS的值。同样，DS包含数据段的段号，即数据存储在存储器中的区域。SS寄存器包含有堆栈段的段号，即保留的栈。ES寄存器包含有可用于乘法运算的附加数据段的段号。FS和GS是80386增加的，它们便于访问两个附加数据段。

在平面32位存储器模式中，编程人员不太考虑段寄存器。操作系统给每个CS、DS、ES和SS相同的值。回想一下，这是一个表的入口指针，该表包含段的实际起始地址，也包含程序的大小。因此，当程序随意或者故意对另一个区域进行写操作时，操作系统可能提示错误。但是

这些对编程人员都没有什么关系，编程人员只根据 32 位地址来考虑。

32 位**指令指针**，或称为 EIP 寄存器，汇编语言的编程人员是不能直接对其进行访问的，CPU 必须从存储器中取出要执行的指令，并且，EIP 跟踪下一条待取指令的地址。如果是比较老式的、简单的计算机体系结构，下一条待取指令可能也是下一条将要执行的指令。事实上，80x86 CPU 在执行前一条指令时，它就取随后要执行的指令了。假设（通常是正确的）：下一次将执行的指令在存储器中是有序紧随（上一条指令）的。如果这种假设证明是错误的，例如，如果执行一个过程调用，则 CPU 取出存储的指令，设置 EIP 包含这个过程的偏移量，并且从新的地址取下一条指令。

另外，对于预取指令，80x86 CPU 在完成前一条指令的执行前，实际上它就开始执行这个预取指令了。**流水线**（pipelining）使用了这种方法，加快了处理器的有效速度。

最后的寄存器称之为**标志寄存器**（flag register），名为 EFLAGS 指的就是这种寄存器，但是指令中不使用这个助记符。32 位的一些位用于设置 80x86 处理器的某些特征，其他的位，称为**状态标志位**，表示指令执行的结果，常用的标志寄存器的 32 位中的有些位的名字在图 1-3 中给出。标志位的改变取决于所执行的指令。例如，第 6 位（零标志位 ZF）设定为 1，因为相加的和为 0。其他的标志将在下一节详细讨论。

| 位 | 助记符 | 作用 |
|---|---|---|
| 0 | CF | 进位标志位 |
| 2 | PF | 奇偶校验标志位 |
| 6 | ZF | 全零标志位 |
| 7 | SF | 符号标志位 |
| 11 | OF | 溢出标志位 |

图 1-3　部分 EFLAGS 位

总之，80x86 CPU 使用 16 个内部寄存器存储操作数和运算结果，并且跟踪段选择器和段地址，可执行很多指令，图 1-4 对这些寄存器的使用进行了总结。

| 名字 | 长度（位） | 使用 / 说明 |
|---|---|---|
| EAX | 32 | 累加器，通用；低位字 16 位 AX，可分为 AH 和 AL |
| EBX | 32 | 通用；低位字 16 位 BX，可分为 BH 和 BL |
| ECX | 32 | 通用；低位字 16 位 CX，可分为 CH 和 CL |
| EDX | 32 | 通用；低位字 16 位 DX，可分为 DH 和 DL |
| ESI | 32 | 源索引；串复制源地址，数组索引 |
| EDI | 32 | 目的索引；目的地址，数组索引 |
| ESP | 32 | 栈指针；栈顶指针 |
| EBP | 32 | 基址指针；栈中引用位置的地址 |
| CS | 16 | 代码段选择器 |

图 1-4　80x86 寄存器

| 名字 | 长度（位） | 使用 / 说明 |
|---|---|---|
| DS | 16 | 数据段选择器 |
| ES | 16 | 附加段选择器 |
| SS | 16 | 栈段选择器 |
| FS | 16 | 附加段选择器 |
| GS | 16 | 附加段选择器 |
| EIP | 32 | 指令指针：下一条待取指令的地址 |
| EFLAGS | 32 | 标志集：或者状态位 |

图 1-4 （续）

**练习 1.3**

1. 画图说明 ECX、CX、CH 和 CL 之间的关系。
2. 标志位 PF 是奇偶校验标志位，计算结果的低位字节的值如果为 1 表示计算结果是偶数，为 0 表示计算结果是奇数。如果计算结果的低字节是 $4E_{16}$，PF 是什么？

## 1.4 字符编码

字母、数字、标点符号等各种字符在计算机中都可用一个特定的数值来表示。字符编码方式有很多，微机中普遍采用的一种字符编码是美国信息交换标准代码（简称为 ASCII）。

ASCII 用 7 位表示字符，数值从 0000000 到 1111111，它有 128 种组合，可以表示 128 种字符。也可用十六进制数 00 ～ 7F 或者十进制数 0 ～ 127 表示[⊖]。附录 A 给出了 ASCII 的详细列表，在表中可以查到“Computers are fun .”用十六进制表示的 ASCII 码值：

| 43 | 6F | 6D | 70 | 75 | 74 | 65 | 72 | 73 | 20 | 61 | 72 | 65 | 20 | 66 | 75 | 6E | 2E |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | o | m | p | u | t | e | r | s | | a | r | e | | f | u | n | . |

注意：尽管空格字符不可见，但仍然有一个字符编码（十六进制数 20H）。

数字也可以用字符编码表示，例如用 ASCII 表示日期“October 23, 1970”：

| 4F | 63 | 74 | 6F | 62 | 65 | 72 | 20 | 32 | 33 | 2C | 20 | 31 | 39 | 37 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| O | c | t | o | b | e | r | | 2 | 3 | , | | 1 | 9 | 7 | 0 |

其中，日期中的数字字符 23 用 ASCII 码值 32 33 表示，1970 用 31 39 37 30 表示，这与 1.1 节所讲的二进制表示有所不同，上节中 $23_{10}=10111_2$，$1970_{10}=11110110010_2$。计算机使用这两种数字表示方法，其中 ASCII 表示法用于外设输入输出，二进制表示法用于计算机内部计算。

ASCII 的代码看起来似乎是任意指定的，但事实上是遵循某些规范的。大写字母的 ASCII

⊖ 许多计算机使用扩展字符集，它另外增加了十六进制字符 80 ～ FF（十进制 128 ～ 255）。本书不使用扩展字符集。

码值是相邻的，小写字母的ASCII码值也是如此。大写字母的编码与其相对应的小写字母的编码仅仅有一位不同，大写字母的第5位是0，而小写字母的第5位是1，其他各位都相同。（通常计算机用位来表示数时，从右到左，最右边的位以第0位开始。）例如，

大写字母M的编码为$4D_{16}=1001101_2$

小写字母m的编码为$6D_{16}=1101101_2$

**打印输出字符**从$20_{16}$～$7E_{16}$。（空格字符也是打印输出字符。）数字0，1，…，9的ASCII值分别为$30_{16}$，$31_{16}$，…，$39_{16}$。

ASCII码值从$00_{16}$到$1F_{16}$以及$7F_{16}$都是**控制字符**，例如，ASCII键盘上的ESC键的ASCII码值是$1B_{16}$，简称ESC，表示特殊服务控制，但经常被认为是“escape”的含义。ESC字符经常与其他字符一起传给外部设备，比如，传给一台打印机，让它执行某种指定的操作。因为这样的字符序列没有标准化，所以本书将不作讨论。

CR和LF是两个非常常见的ASCII控制字符，CR（$0D_{16}$）表示返回，LF（$0A_{16}$）表示换行。当按下ASCII键盘的Return或Enter键时，就会产生编码$0D_{16}$，如果该编码送到ASCII显示器，则使光标移到当前行的开始处而不是到新的一行；如果该编码送到ASCII打印机，则会使打印头移到当前行的开始处。换行码$0A_{16}$在ASCII显示器上会使光标垂直移到下一行或者使打印机将纸向上滚动一行。

使用较少的控制字符有“Form Feed”（$0C_{16}$），该字符使打印机换页；控制字符“Horizontal Tab”（$09_{16}$）在按下键盘的Tab键时生成；“Backspace”（$08_{16}$）在按下键盘的Backspace键时生成；“Delete”（$7F_{16}$）在按下键盘的Delete键时生成。注意：Delete键和Backspace键生成的代码不同。响铃（Bell）字符（$07_{16}$）输出到显示器时会听见响铃声。

许多大型计算机用“扩展二进制编码－十进制信息编码”（简称EBCDIC）。本书不讨论EBCDIC编码。

### 练习 1.4

1. 每个十六进制数可表示为一个十进制数或两个字符的ASCII值，请写出这两种表示。
   a. 2A45
   b. 7352
   c. 2036
   d. 106E
2. 写出下列字符串的ASCII值，不要忘记空格和标点符号。返回和换行字符用CR和LF表示，如果写在一起CRLF（中间没有空格）表示回车换行功能。
   a. January1 is New Year's Day. CRLF
   b. George said, “Ouch!”
   c. R2D2 was C3P0's friend.CRLF[0是数字0]
   d. Your name? [问号后输入两个空格]
   e. Enter value: [冒号后输入两个空格]
3. 将下列ASCII序列输出到计算机显示器，会显示什么？
   a. 62 6C 6F 6F 64 2C 20 73 77 65 61 74 20 61 6E 64 20 74 65 61 72 73

b. 6E 61 6D 65 0D 0A 61 64 64 72 65 73 73 0D 0A 63 69 74 79 0D 0A

c. 4A 75 6E 65 20 31 31 2C 20 31 39 34 37 0D 0A

d. 24 33 38 39 2E 34 35

e. 49 44 23 3A 20 20 31 32 33 2D 34 35 2D 36 37 38 39

## 1.5 有符号整数的二进制补码表示

本节详细探讨计算机中数的表示。前面已经介绍了两种表示数的方法，一种是用二进制表示（通常用十六进制表示），另一种是用 ASCII 码表示，但是这两种表示法有两个问题：如何表示一个负数；表示数的有效位是有限的。

假定内存中的数用 ASCII 码存储，一个 ASCII 码通常用一个字节存储，而 ASCII 码长度是 7 位，附加位（左侧或最高位）置 0。为了解决上述表示法中的第一个问题，可在编码中包含一个负数符号。例如：四个字符的 −817 的 ASCII 编码表示就是 2D，38，31 和 37。为解决第二个问题，通常用固定长度的若干字节数，位数不足时，在 ASCII 码的左边用 0 或者空格补充；也可以使用一个长度可变的字节数，但必须规定要表示的数的最后一个数位以 ASCII 码结束，也就是说用一个非数字字符结束。通常，80x86 结构没有什么指令用来处理 ASCII 格式的数字，即使有这样的指令，也很少使用。

计算机内部使用二进制表示数时，80x86 和多数计算机选用固定长度的二进制数运算来解决长度问题，Intel 80x86 系列可用的长度有 8 位（1 字节）、16 位（1 **个字**）[⊖]、32 位（**双字**）和 64 位（**四字**）。

例如，697 的双字长二进制表示为：

$$697_{10} = 1010111001_2 = 00000000000000000000001010111001_2$$

二进制数表示的前面添加 0 是为了使长度是 32 位，该二进制数用十六进制数表示如下：

| 00 | 00 | 02 | B9 |
|---|---|---|---|

前面介绍的表示法能够很好地表示非负数和**无符号**数，但不能表示负数。同时，对于任意给定长度都可表示一个最大无符号数，例如一个字节长度，表示的最大无符号数为 $FF_{16}$ 或者 $255_{10}$。

**二进制补码**表示法与前面所讲的无符号数表示法很相似，但前者可以表示负数。二进制补码表示数时，应该指定其长度，所以可用“双字长的二进制补码表示法”表示一个数。二进制补码表示非负数与表示无符号数大致相同；也就是说，二进制补码表示数时，前面需要补充很多 0 来确保定长。只有一个限制：对于正数的表示有一个附加位，最左边的一位置 0。如：用字长二进制补码表示最大的正数就是 $0111111111111111_2$，即 $7FFFF_{16}$ 或者 $32767_{10}$。

如果根据正数的表示是最左边一位置 0，就推测负数的表示是将最左边一位置 1，其他各位跟对应正数的表示完全相同，那就大错特错了。因为负数的表示要比正数的表示复杂得多，所以不能简单地将最左边的位由 0 改为 1 来表示负数。

一个十六进制的计算器很容易将一个负的十进制数转换为二进制补码表示。例如，计算器

⊖ 其他计算机体系结构一个字的长度可能不是 16 位。