HZ BOOKS

Professional

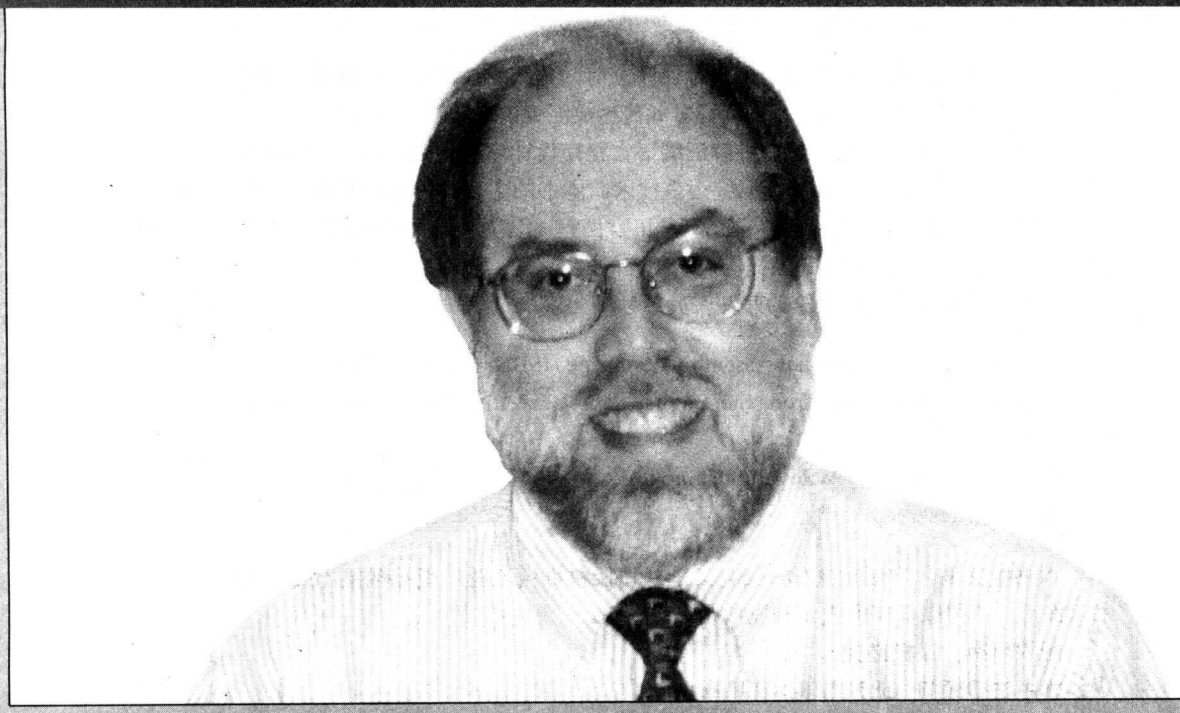
Assembly Language

汇编语言程序设计

（美） Richard Blum 著

马朝晖 等译

机械工业出版社
China Machine Press



Professional Assembly Language

汇编语言程序设计

（美） Richard Blum 著

马朝晖 等译

机械工业出版社
China Machine Press

每种高级语言程序在连接为可执行程序之前，都必须被编译为汇编语言程序，因此对于高级语言程序设计者来说，了解编译器如何生成汇编语言代码十分有用。

本书分为三部分。第一部分讲解汇编语言程序设计环境基础，第二部分研究汇编语言程序设计，最后一部分讲解高级汇编语言技术。本书的主要目的是向使用高级语言的程序员讲解编译器如何从C和C++程序创建汇编语言例程，以及编程人员应如何掌握生成的汇编语言代码，调整汇编语言例程以提高应用程序的性能。

本书适合有一定编程经验的开发人员参考。

Richard Blum: Professional Assembly Language (ISBN: 0-7645-7901-0).

Authorized translation from the English language edition published by John Wiley & Sons, Inc.

本书版权登记号：图字：01-2005-1622

图书在版编目（CIP）数据

汇编语言程序设计/（美）布鲁姆（Blum, R.）著；马朝晖等译. -北京：机械工业出版社，2006.1

书名原文：Professional Assembly Language

ISBN 7-111-17532-8

Ⅰ.汇 … Ⅱ.① 布 … ② 马 … Ⅲ.汇编语言-程序设计 Ⅳ.TP313

中国版本图书馆CIP数据核字（2005）第117128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：赵　康　刘立卿

北京瑞德印刷有限公司印刷 · 新华书店北京发行所发行

2006年1月第1版第1次印刷

787mm × 1092mm 1/16 · 26.5印张

印数：0 001- 4 000册

定价：48.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

前　言

在目前正在使用的程序设计语言之中，汇编语言是被误解得最深的一种。当提到“汇编语言”这个术语时，经常使人联想到低级的位移动和在长达数千页的指令手册中费力地查找正确的指令格式。随着各种出色的高级语言开发工具的快速发展，在各种程序设计新闻组中“汇编语言程序设计已经死亡了”这种评论并不少见。

但是，汇编语言程序设计远没有到死亡的时候。每种高级语言程序在能够连接为可执行程序之前都必须被编译为汇编语言程序。对于高级语言程序设计者来说，了解编译器如何生成汇编语言代码很有用处，这表现在使用汇编语言直接编写例程和了解编译器如何把高级语言转换为汇编语言方面。

本书目的

本书的主要目的是向使用高级语言的程序员讲解高级语言程序是如何被转换为汇编语言的，以及如何掌握生成的汇编语言代码。这就是说，本书的主要读者是已经熟悉高级语言（比如C、C++，甚至Java）的程序员。本书没有花时间讲解基本的程序设计原则。我们假设读者已经熟悉计算机程序设计的基础，并且有兴趣学习汇编语言以便了解程序运行的幕后发生了什么。

但是，如果你是程序设计的初学者并且把汇编语言程序设计作为起点，本书也没有完全忽略你的要求。可以从头到尾阅读各个章节获得如何进行汇编语言程序设计（和一般的程序设计）的基础知识。书中的每个主题都包括范例代码来演示汇编语言指令如何工作。如果你完全是程序设计新手，也可以从本书开始学习程序设计，进而学习本书其他高级主题。

本书范围

本书的主要目的是使C和C++程序员熟悉汇编语言，讲解编译器如何从C和C++程序创建汇编语言例程，并讲解如何整理生成的汇编语言例程以便提高应用程序的性能。

所有用高级语言（比如C和C++）编写的程序，在被连接为可执行程序之前，都会被编译器转换为汇编语言。编译器使用编译器的设计者定义的特定规则来确定如何正确地转换高级语言语句。很多程序员只是编写高级语言程序并且假设编译器会创建正确的可执行代码来实现程序。

但是，情况并非总是如此。当编译器把高级语言代码转换为汇编语言代码时，稀奇古怪的事情经常出现。另外，编译器往往遵循非常特别的转换规则，以至于不能在最终的汇编语言代码中发现节省时间的捷径，而对于编写不良的高级例程，它也不能加以改善。在这样的情况下，汇编语言代码的知识就有用武之地了。

本书正是讲解汇编语言的知识，描述在连接为可执行程序之前如何检查编译器生成的汇编

参与本书翻译工作的有：马朝晖、迟旭、陈美红、楼涵、裔彩霞、郑纪革、何运刚、张志刚、李晓东、曾明月、刘嘉。

语言代码，并发现可以修改何处代码来提高性能或者提供附加功能，帮助读者理解编译器的转换处理是如何影响高级语言例程的。

本书结构

本书分为三个部分。第一部分讲解汇编语言程序设计环境的基础。因为汇编语言在各种处理器和汇编器之间是不同的，所以必须选择常见的平台。本书使用运行在Intel处理器系列上的Linux操作系统。Linux环境提供丰富的程序开发工具，比如优化编译器、汇编器、连接器和调试器，它们的费用很低，或者是免费的。Linux环境之中这些丰富的开发工具使它非常适合把C程序剖析为汇编语言代码。

第一部分的各章如下:

第1章“什么是汇编语言”，一开始确保你确切地了解什么是汇编语言以及如何将它融入程序设计模型。这一章揭开了汇编语言的神秘面纱，并且提供了解如何把汇编语言和高级语言一起使用的基础知识。

第2章“IA-32平台”，提供对Intel奔腾处理器系列的简要介绍。当使用汇编语言时，了解底层的处理器和它如何处理程序是很重要的。但是这一章没有打算对IA-32平台的操作进行深入的分析，也没有提供在这个平台上进行程序设计涉及到的硬件和操作。

第3章“相关的工具”，讲解本书中使用的Linux开放源代码的开发工具。本书中使用GNU编译器、汇编器、连接器和调试器对程序进行编译、汇编、连接和调试。

第4章“汇编语言程序范例”，演示如何在Linux系统上使用GNU工具创建、汇编、连接和调试简单的汇编语言程序。这一章还演示如何在Linux系统上在汇编语言程序中使用C库函数为汇编语言应用程序添加额外的特性。

本书的第二部分研究汇编语言程序设计的基础。在能够分析编译器生成的汇编语言代码之前，必须了解汇编语言指令。这一部分的各章如下:

第5章“传送数据”，讲解在汇编语言程序中如何传送数据元素。讲解寄存器、内存位置和堆栈的概念，并且提供在它们之间传送数据的范例。

第6章“控制执行流程”，描述汇编语言程序中使用的分支指令。这可能是程序最为重要的特性之一，认识分支并且优化分支的能力对提高应用程序的性能是至关重要的。

第7章“使用数字”，讨论在汇编语言中如何使用不同的数字数据类型。能够在汇编语言程序内正确地处理整数和浮点值是很重要的。

第8章“基本数学功能”，讲解如何使用汇编语言指令实现基本的数学功能，比如加、减、乘和除。虽然这些通常是直接的功能，但是可以使用灵活的技巧提高这一领域工作的性能。

第9章“高级数学功能”，讨论IA-32浮点运算单元（Floating Point Unit，FPU），以及如何使用它处理复杂的浮点运算。浮点运算对于数据处理程序经常是至关重要的，了解它如何工作对高级语言程序员有非常大的好处。

第10章“处理字符串”，讲解汇编语言的各种字符串处理指令。字符数据是高级语言程序设计的另一个重要方面。在高级语言中处理字符串时，了解汇编语言层面上如何处理字符串能够提供深入的认识。

第11章“使用函数”，开始深入地讲解汇编语言程序设计。创建汇编语言函数去执行例程是汇编语言优化的核心。了解汇编语言函数的基础知识是有益的，因为编译器从高级语言代码生成汇编语言代码时经常会使用它们。

第12章“使用Linux系统调用”，这一章结束了这一部分，它演示使用已经创建的函数可以在汇编语言中执行多少高级功能。Linux系统提供很多高级功能，例如输出到显示器的功能。在汇编语言程序中经常可以利用它们。

第三部分讲解更加高级的汇编语言主题。因为本书的主要主题是如何在C或者C++代码之中并入汇编语言例程，所以最初的几章只是讲解如何这样做。其余的各章讲解一些更加高级的主题，圆满地完成读者对汇编语言程序设计的学习。这一部分包括下面这几章：

第13章“使用内联汇编”，讲解如何把汇编语言例程直接并入你的C或者C++语言程序中。内联汇编语言经常用于在C程序中“硬编码”快速例程，以便确保编译器为例程生成适当的汇编语言代码。

第14章“调用汇编库”，演示可以如何把汇编语言函数组合为库，供众多应用程序（包括汇编语言程序和高级语言程序）使用。能够把频繁使用的函数组合为C或者C++程序可以调用的单一库是非常节省时间的特性。

第15章“优化例程”，本书的核心部分：修改编译器生成的汇编语言代码以便满足特定要求。这一章讲解在汇编语言代码中究竟如何生成不同类型的C例程（比如if-then语句和for-next循环）。了解了汇编语言代码在做什么之后，就可以对它进行修改以便为特定的环境定制代码。

第16章“使用文件”，讲解汇编语言程序设计中最被忽视的功能之一。几乎所有应用程序都需要对系统进行某种类型的文件访问。汇编语言程序也不例外。这一章讲解如何使用Linux的文件处理系统调用读取、写入和修改系统中文件内的数据。

第17章“使用高级IA-32特性”，这一章结束了本书，它讲解高级的Intel的单指令多数据（Single Instruction Multiple Data，SIMD）技术。这种技术为程序员提供在单一指令之中执行多个运算操作的平台。在音频和视频数据处理的领域之中，这种技术变得非常重要。

使用本书的要求

本书中的所有范例都用汇编语言编写，并且运行在Linux操作系统和Intel处理器平台上。本书中广泛地使用开放源代码的GNU编译器（gcc）、汇编器（gas）、连接器（ld）和调试器（gdb）演示汇编语言特性。第4章专门讨论如何在Linux平台上使用这些工具创建、汇编、连接和调试汇编语言程序。如果读者没有安装Linux平台，第4章演示了如何使用可以直接从光盘引导的、无需修改工作站硬盘的Linux版本。无需在工作站上安装Linux，就可以使用本书中使用的所有GNU开发工具。

约定

为了帮助读者从书中得到最大的收益以及充分理解内容，我们在本书中使用了数种约定。

技巧、提示、诀窍和当前讨论之外的内容使用楷体排版。在代码范例中我们使用灰色背景突出显示新的和重要的代码，而当前上下文中不那么重要的代码以及以前显示过的代码将不用

灰色背景。

源代码

当你研究本书中的范例时，可以选择手工输入所有代码，或者从网上下载本书源代码文件。本书中用到的所有示例源代码都可以从www.wrox.com下载。访问这个站点时，只需查找本书的名称（可以使用Search文本框，也可以使用书名列表之一），然后在本书的具体介绍页面上点击Download Code链接即可获得本书的所有源代码。⊖

因为很多书籍的名称类似，所以查找书籍最简单的方式是通过ISBN查找；本书的ISBN是0-7645-7901-0。

下载代码后可以使用常用的压缩工具解压。或者可以到Wrox公司的代码下载页面（www.wrox.com/dynamic/books/download.aspx）查看本书和所有其他Wrox的书籍的代码。

勘误

我们尽了最大的努力来确保文本或者代码中没有错误。但是，没有人能够做到完美，错误在所难免。如果发现我们的书中有错误，比如拼写错误或者有错误的代码段，我们将非常高兴收到反馈。通过发送勘误表，可能避免另外一个读者数小时的迷惑，同时帮助我们提供品质更好的信息。

要查看本书的勘误表，可以访问www.wrox.com并且使用Search文本框或者书名列表之一查找书籍名称。然后，在书籍的具体介绍页面上点击Book Errata链接。在这个页面上，可以查看关于本书已经提交的和Wrox公司的编辑公布的所有勘误。完整的书籍列表，包括到每本书的勘误的链接，都可以在www.wrox.com/misc-pages/booklist.shtml页面上找到。

如果没有在Book Errata页面上找到“你的”错误，请访问www.wrox.com/ contact/techsupport.shtml并且完整填写这里的表单，把发现的错误发送给我们。我们会查看这些信息，并且，如果正确的话，会在书籍的勘误页面上公布消息并在书籍以后的版本中修正这些问题。

p2p.wrox.com

为了与作者和同行们进行讨论，请加入P2P论坛：p2p.wrox.com。这个论坛是基于Web的系统，你可以张贴关于Wrox公司书籍的和关于技术的消息，可以和其他读者以及技术用户进行交流。论坛提供订阅功能，当论坛上发布新帖子的时候，可以通过电子邮件发送你所选择的感兴趣的主题。Wrox公司的作者、编辑、其他业界专家以及本书读者都会出现在这些论坛上。

在http://p2p.wrox.com上，你会发现许多不同的论坛，在你阅读本书及开发自己的应用程序时，它们都能够给你提供帮助。可以按照下面的步骤加入论坛：

1) 访问p2p.wrox.com，点击Register链接。

2) 阅读使用条款并且点击Agree。

3) 填写必需的信息以及你希望提供的任何可选的信息，点击Submit。

⊖ 也可登录华章网站（www.hzbook.com）下载源代码。

4) 然后你会收到描述如何验证帐户和完成加入过程的电子邮件。

不加入P2P也可以阅读论坛的消息，但是要想发布自己的消息，就必须加入。

加入后，你可以发布新的消息并且回复其他用户发布的消息。任何时候都可以在Web上阅读消息。如果希望特定的论坛使用电子邮件通知新的消息，可以在论坛列表中点击这个论坛的Subscribe链接。

关于如何使用Wrox公司的P2P的更多信息，请阅读P2P FAQ，它们回答了关于论坛软件如何工作的问题，这些FAQ也包括关于P2P和Wrox公司书籍的很多常见问题。要阅读FAQ，可以在任何P2P页面上点击FAQ链接。

目 录

第二部分 汇编语言程序设计基础

第三部分 高级汇编语言技术

第一部分　汇编语言程序设计环境基础

第1章　什么是汇编语言

学习汇编语言的首要是了解什么是汇编语言。与其他程序设计语言不同的是，并不是所有汇编器都使用一种标准格式。不同的汇编器使用不同的语法编写程序语句。在试图掌握无数种汇编语言程序设计时，很多汇编语言程序设计的初学者都遇到了困难。

学习汇编语言程序设计的第一个步骤是，决定在现有环境中希望（或者需要）使用什么类型的汇编语言。一旦决定了要使用的汇编语言，开始学习并且在独立的和高级语言的程序中使用它就容易了。

本章首先讲解汇编语言是从哪里发展而来的，并且说明为什么要使用汇编语言进行程序设计。要了解汇编语言程序设计，必须首先了解它的最基本目的——使用处理器指令代码进行程序设计。接下来，这一章讲解编译器和连接器如何把高级语言转换为原始指令代码。学习了这些知识之后，就容易了解汇编语言程序和高级语言程序的区别在哪里，还有如何同时使用它们以相互补充。

1.1　处理器指令

在操作的最低层，所有计算机处理器（微型计算机、小型计算机和大型计算机）都按照制造厂商在处理器芯片内部定义的二进制代码操作数据。这些代码定义处理器应该利用程序员提供的数据完成什么功能。这些预置的代码被称为*指令码*（instruction code）。不同类型的处理器包含不同类型的指令码。通常按照处理器芯片支持的指令码的数量和类型对它们进行分类。

虽然不同类型的处理器可能包含不同类型的指令码，但是它们处理指令码程序的方式是类似的。这一小节讲解处理器如何处理指令，以及一个范例处理器芯片的指令码是什么样子。

1.1.1　指令码处理

当计算机处理器芯片运行时，它读取存储在内存中的指令码。每个指令码集合可能包含一个或者多个字节的信息，这些信息指示处理器完成特定的任务。每条指令码都是从内存读取的，指令码所需的数据也是存储在内存中并且从内存读取。包含指令码的内存字节和包含处理器使用的数据的字节没有区别。

为了区分数据和指令码，要使用专门的*指针*（pointer）帮助处理器跟踪数据和指令码存储在内存中的什么位置。这显示在图1-1中。

指令指针（instruction pointer）用于帮助处理器了解哪些指令码已经处理过了，以及接下来要处理的是哪条指令码。当然，有些专门的指令能够改变指令指针的位置，比如跳转到程序的

特定位置。

类似的，数据指针（data pointer）用于帮助处理器了解内存中数据区域的起始位置是哪里。这个区域称为堆栈（stack）。当新的数据元素被放入堆栈中时，指针在内存中“向下”移动。当数据被读取出堆栈时，指针在内存中“向上”移动。

每条指令码都包含一个或者多个字节的处理器要处理的信息。例如，下面这些指令码字节（十六进制格式）

```
C7 45 FC 01 00 00 00
```

通知Intel IA-32系列处理器把十进制值1加载到一个处理器寄存器定义的内存偏移位置。指令码包含若干信息片段（在后面的“操作码”小节中定义），它们明确地定义处理器要完成什么操作。处理器完成了一个指令码集合的处理之后，它读取内存中的下一个指令码集合（就是指令指针所指向的）。在内存中，指令必须按照正确的格式和顺序放置，使处理器能够正确地按顺序执行程序代码。

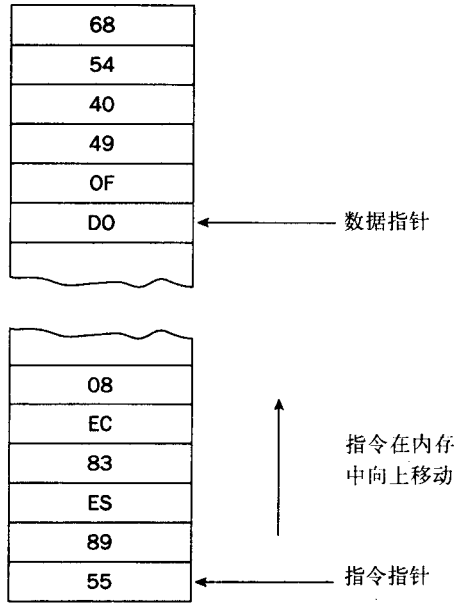


图 1-1

每条指令都必须至少包含1个字节的操作码（operation code，简写为opcode）。操作码定义处理器应该完成什么操作。每个处理器系列都具有其自己的预定义好的操作码，它们定义所有可用的功能。下一小节介绍Intel IA-32系列微处理器中使用的操作码是如何构成的。本书中的所有例子都使用这种类型处理器的操作码。

1.1.2 指令码格式

Intel IA-32系列微处理器包括现代IBM平台的微型计算机所使用的所有当前类型的微处理器（参见第2章），还包括流行的奔腾系列微处理器。IA-32系列的微处理器使用专门格式的指令码，了解这些指令的格式对汇编语言程序设计将有所帮助。IA-32指令码格式由四个主要部分构成：

- 可选的指令前缀
- 操作码（opcode）
- 可选的修饰符
- 可选的数据元素

图1-2是IA-32指令码格式的布局。

每个部分都用于完整地为处理器定义要执行的特定指令。下面的几个小节介绍指令码的每个部分和它们如何定义由处理器执行的指令。

Intel的奔腾处理器系列不是使用IA-32指令码格式的唯一处理器芯片系列。AMD公司生产的一系列芯片也完全兼容Intel IA-32指令码格式。

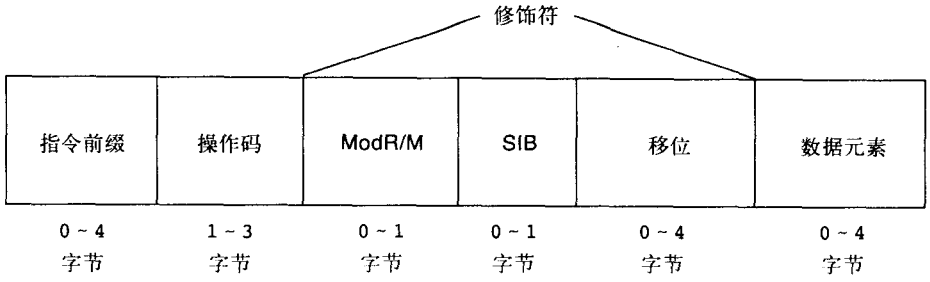


图 1-2

1. 操作码

如图1-2所示，IA-32指令码格式中唯一必须的部分是操作码。每个指令码都必须包含操作码，它定义由处理器执行的基本功能或者任务。

操作码的长度在1到3字节之间，它唯一地定义要执行的功能。例如，2字节的操作码OF A2定义IA-32 CPUID指令。当处理器执行这个指令码时，它返回不同寄存器中关于微处理器的特定信息。然后，程序员可以使用其他的指令码从处理器寄存器中提取信息，以便确定运行程序的微处理器的类型和型号。

寄存器是处理器芯片之内的组件，用于临时存储处理器正在处理的数据。寄存器的详细信息将在第2章“IA-32平台”中讲解。

2. 指令前缀

指令前缀可以包含1个到4个修改操作码行为的1字节前缀。按照前缀的功能，这些前缀被分为4个组。修改操作码时，每个组的前缀一次只能使用一个（因此最多有4个前缀字节）。这4个前缀组如下：

- 锁定前缀和重复前缀
- 段覆盖前缀和分支提示前缀
- 操作数长度覆盖前缀
- 地址长度覆盖前缀

锁定前缀表示指令将独占地使用共享内存区域。这对于多处理器和超线程系统非常重要。重复前缀用于表示重复的功能（常常在处理字符串时使用）。

段覆盖前缀定义可以覆盖定义了的段寄存器值的指令（这将在第2章中更加详细地讲解）。分支提示前缀尝试向处理器提供程序在条件跳转语句中最可能的路径的线索（这同预报分支的硬件一起使用）。

操作数长度覆盖前缀通知处理器，程序将在这个操作码之内切换16位和32位的操作数长度。这使程序可以在使用大长度的操作数时警告处理器，帮助加快对寄存器的数据赋值。

地址长度覆盖前缀通知处理器，程序将切换16位和32位的内存地址。这两种长度都可以被声明为程序的默认长度，这个前缀通知处理器程序将切换到另一种长度。

3. 修饰符

一些操作码需要另外的修饰符来定义执行的功能中涉及到什么寄存器和内存位置。修饰符