HZ BOOKS
华章教育

计 算 机 科 学 丛 书

Apress®

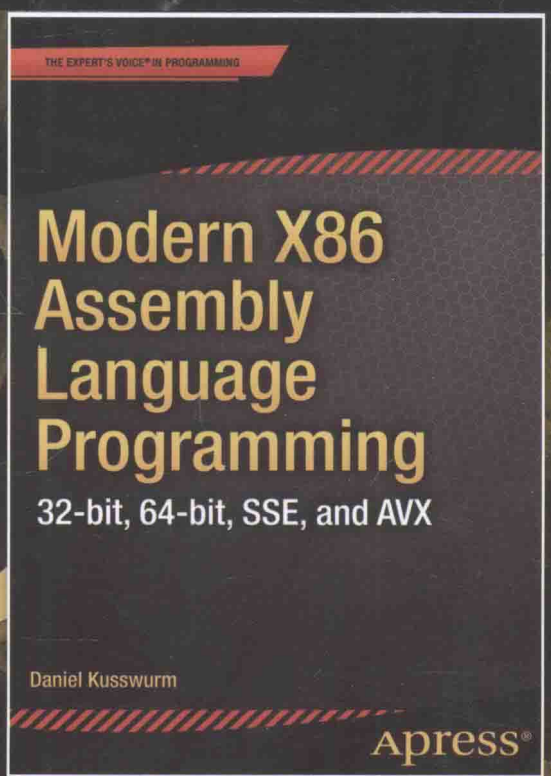
现代x86汇编语言程序设计

[美] 丹尼尔·卡斯沃姆（Daniel Kusswurm）著

张银奎 罗冰 宋维 张佩 等译

Modern X86 Assembly Language Programming

32-bit, 64-bit, SSE, and AVX



机械工业出版社
China Machine Press

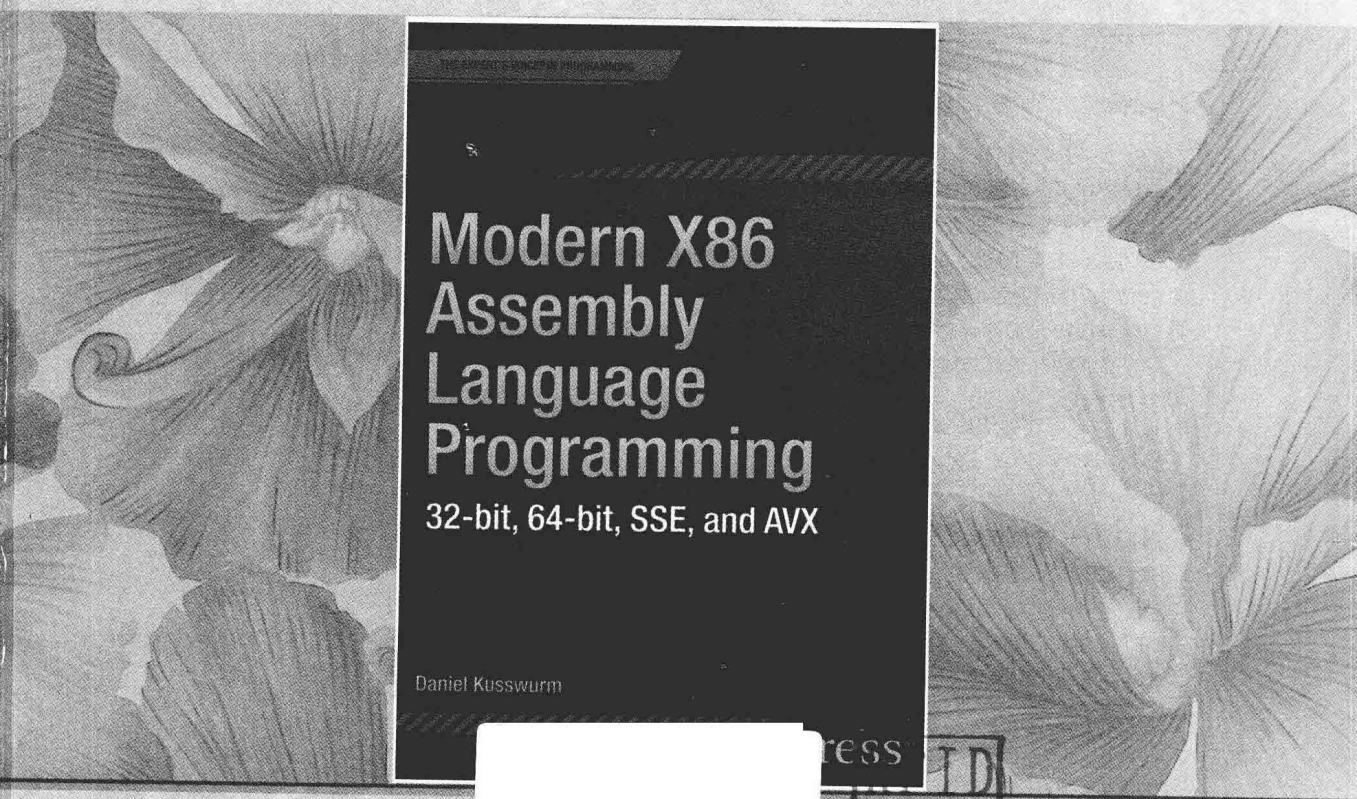
计 [redacted] 丛 书

现代x86汇编语言程序设计

[美] 丹尼尔·卡斯沃姆（Daniel Kusswurm）著
张银奎 罗冰 宋维 张佩 等译

Modern X86 Assembly Language Programming

32-bit, 64-bit, SSE, and AVX



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

现代 x86 汇编语言程序设计 /（美）丹尼尔·卡斯沃姆（Daniel Kusswurm）著；张银奎等译 . —北京：机械工业出版社，2016.7
（计算机科学丛书）
书名原文：Modern X86 Assembly Language Programming: 32-bit, 64-bit, SSE, and AVX

ISBN 978-7-111-54278-0

I. 现… II. ①丹… ②张… III. 汇编语言－程序设计 IV. TP313

中国版本图书馆 CIP 数据核字（2016）第 163448 号

本书版权登记号：图字：01-2015-6379

本书从应用编程的角度解释 x86 处理器的内部架构和执行环境，全面介绍如何用 x86 汇编语言编写可被高级语言调用的函数。主要内容包括：x86-32 核心架构（第 1 章和第 2 章），x87 浮点单元（第 3 章和第 4 章），MMX 技术（第 5 章和第 6 章），流式 SIMD 扩展（第 7 章至第 11 章），高级向量扩展（第 12 章至第 16 章），x86-64 核心架构（第 17 章和第 18 章），x86-64 SSE 和 AVX（第 19 章和第 20 章），高级主题（第 21 章和第 22 章）。书中包含了大量的示例代码，以帮助读者快速理解 x86 汇编语言编程和 x86 平台的计算资源。

本书可作为高等院校计算机及相关专业学生的教材，也可供想要学习 x86 汇编语言编程的软件开发者使用。

出版发行：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：迟振春　　责任校对：殷　虹
印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2016 年 10 月第 1 版第 1 次印刷
开　　本：185mm×260mm　1/16　　印　　张：30.75
书　　号：ISBN 978-7-111-54278-0　　定　　价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

文艺复兴以来，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；也正是这样的优势，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公司较早意识到“出版要为教育服务”。自 1998 年开始，我们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，Brian 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相助，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，有的还专门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。华章公司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
华章科技图书出版中心

译者序

Modern X86 Assembly Language Programming: 32-bit, 64-bit, SSE, and AVX

尼采曾经说过："没有可怕的深度，就没有美丽的水面。"这句话或许可以解释我们为什么要翻译这本书。

这本书确实是讨论汇编语言编程的。在这样一个各种脚本语言大行其道的"速成"时代，是否还有必要学习汇编语言呢？对于这个问题，简单回答是或否都可能失于片面。

不妨先讲个小故事。我曾经编写过一个 C++ 程序，以多线程的方式计算 Mercator 级数。调试版本工作正常之后测试发布版本时发现了一个大问题：负责具体计算任务的工作线程纷纷完成计算并退出后，负责汇总结果的主线程却迟迟不给出结果，而且占用 CPU 很高，上调试器一看，CPU 在以下 while 语句不停地循环。

```
while (pBoss->m_dwThreadCount != pBoss->m_dwCalcThreads)
{
    ///Sleep(1);
}   // busy wait until all threads are done with computation of partial sums
```

观察 while 语句中的两个变量，它们显然已经相等了：

```
+0x048 m_dwThreadCount  : 0xa
+0x04c m_dwCalcThreads  : 0xa
```

尝试单步跟踪，CPU 始终不离开这行语句，仿佛被黏在这里一样。观察程序指针对应的汇编指令，我不禁愕然：

```
004012f0 ebfe            jmp     MulThrds!CBoxBoss::MasterThreadProc+0x40 (004012f0)
```

这是一条无条件跳转语句，跳转的目标地址居然就是同一条指令的地址。看来 CPU 在原地打转，根本没有判断两个变量是否相等。但这样说其实不准确，观察当前指令前面的指令，CPU 曾经做过比较和判断：

```
004012e6 8b4e4c          mov     ecx,dword ptr [esi+4Ch]
004012e9 8b4648          mov     eax,dword ptr [esi+48h]
004012ec 3bc1            cmp     eax,ecx
004012ee 7402            je      MulThrds!CBoxBoss::MasterThreadProc+0x42 (004012f2)
```

问题的关键是 while 循环内部没有改变要判断的两个变量，而且定义这两个变量时又忘记使用 volatile 关键字来修饰，于是编译器在编译发布版本、做自动优化时，为了避免反复读取内存（内存在 CPU 外部，相对于读寄存器来说访问内存是比较大的开销），便只读取变量一次，也只比较一次，之后便只做跳转而不读取和比较了。增加 volatile 关键字后这个问题就解决了。

这个小故事说明了两个道理。

第一，汇编是 CPU 的语言，学会汇编可以为我们打开一扇窗，透过这扇窗我们可以穿越层层阻隔，探视深邃微妙的底层世界，理解 CPU 的行为，看它是在那个世界里欢快地奔跑还是遭遇陷阱停滞不前。

第二，编译器所做的自动优化有时是出乎我们预料的，可能让我们惊喜，也可能让我们沮丧。要理解编译器的优化行为，懂得汇编语言常常是必需的。

学会汇编语言的另一个好处是当我们对编译器自动优化所达到的效果不满意时，可以直接使用汇编语言编写代码，把性能提高到极致。一个典型的例子就是使用强大的 SIMD 指令（SSE 和 AVX）来优化各种图形图像应用的性能，比如视频编解码和各种机器学习算法。这本书很全面地介绍了 x86 CPU 的各种浮点计算技术，包括经典的 x87 指令，以及最近一些年流行的 SSE 指令和 AVX 指令。单凭这些内容，这本书便不愧“摩登”（Modern）之名。

尼采还说过一句我不愿意接受的话：“书的时代结束了，演员的时代开启了。”在眼下这样一个不是书的时代里，无论是写一本书还是翻译一本书都很艰难。太多干扰让我们无法安静地思考和遣词造句。我的多位格友（格物之友，搜索“格友”公众号可以了解更多）与我一起翻译了这本书，为了能有大块的时间专注于翻译并相互切磋，我们曾两度集结到苏州。第一次住在灵岩山下的木渎古镇。白天在灵岩山下的一个茶馆里挥汗如雨，晚上继续在宾馆里挑灯夜战。中午登上灵岩山，到灵岩寺里吃素面。虽然辛苦，但很充实和快乐。“无图无真相”，贴一张当时的照片吧！

照片中从左至右依次为：罗冰、高异明、宋维、张银奎、崔燚、王卫汉、王科平、张佩。以上格友承担了本书的主要翻译任务，此外，何旻、黎水芬和王建荣也参与了本书的少量翻译和校对工作，在此表示感谢。由于我们的水平有限，难免有翻译不当之处，希望各位读者批评指正。

x86 架构的第一代产品 8086 是从 1976 年开始研发的，距今已有 40 个年头。这 40 年中，基于 x86 CPU 开发的各种产品难以计数，这些产品让这个经典架构传遍了整个世界。学习 x86 汇编语言是了解这一经典架构的最好方式，当我们把一条条指令交给 x86 CPU 执行时，其实就是在和这个经典架构直接“交谈”。在这个交谈过程中，我们可以体会到其中所蕴藏着的智慧，感受到“软件的大美”！

张银奎

2016 年 8 月 16 日于格盡轩

前　言

Modern X86 Assembly Language Programming: 32-bit, 64-bit, SSE, and AVX

从个人电脑发明那一天起，很多软件开发者就使用汇编语言编程，以解决各种各样的难题。在 PC 时代的早期，用 x86 汇编语言编写大段的程序或整个应用是很普遍的。即便是在 C、C++ 和 C# 等高级语言越来越流行的今天，许多软件开发者也仍然使用汇编语言来编写性能攸关的代码。虽然近些年编译器进步很快，编译出来的机器码变得更短、更快，但在某些情况下，软件开发者还是需要努力发挥汇编语言编程的优势。

现代 x86 处理器包含单指令多数据（SIMD）架构，这给我们提供了另一个持续关注汇编语言编程的原因。SIMD 架构的处理器可以同时计算多个数据，这可以显著提高那些需要实时响应的应用软件的性能。SIMD 架构也非常适合那些计算密集型的领域，比如图像处理、音视频编码、计算机辅助设计、计算机图形学和数据挖掘等。遗憾的是许多高级语言和开发工具不能完全发挥现代 x86 处理器的 SIMD 能力。而汇编语言恰恰可以让软件开发者充分利用处理器的全部计算资源。

现代 x86 汇编语言编程

本书是专门针对 x86 汇编语言编程的一本启发性教材，其主要目的是教你如何用 x86 汇编语言编写可被高级语言调用的函数。本书从应用程序编程的角度来解释 x86 处理器的内部架构。书中包含了非常多的示例代码，帮助你快速理解 x86 汇编语言编程和 x86 平台的计算资源。这本书的主要议题包括：

- x86-32 核心架构、数据类型、内部寄存器、内存寻址模式和基本指令集。
- x87 核心架构、寄存器栈、特殊寄存器、浮点编码和指令集。
- MMX 技术和对组合整数进行计算。
- 流式 SIMD 扩展（SSE）和高级向量扩展（AVX），包括内部寄存器、组合整型和浮点运算以及相关指令集。
- x86-64 核心架构、数据类型、内部寄存器、内存寻址模式和基本指令集。
- SSE 和 AVX 技术的 64 位扩展。
- x86 微架构和汇编语言优化技术。

在讨论其他内容之前，我想特别声明一下本书没有覆盖到的内容。本书没有介绍 x86 汇编语言的传统内容，比如 16 位实模式应用和分段内存模型。除了几处历史性的回顾和比较外，所有其他讨论和示例代码都是假定处于 x86 保护模式和平坦线性内存模型下。本书没有讨论 x86 的特权指令和用以支持开发操作系统内核的 CPU 功能，也没有介绍如何用 x86 汇编语言去开发操作系统或者设备驱动程序。不过，如果你真的想用 x86 汇编语言去开发那些系统软件，那么需要先充分理解这本书的内容。

虽然理论上仍然可以完全用汇编语言开发一个应用程序，但是现实中的各种需求使得这种方法很难实行。所以本书重点关注如何创建可被 C++ 调用的 x86 汇编语言模块和函数。本书中的所有示例代码和示例程序都是用微软的 Visual C++ 工具编写并使用微软的宏汇编器编译的。这两个工具都包含在微软的 Visual Studio 开发工具集里面。

目标读者

本书是针对下面几类软件开发者而编写的：

- 在 Windows 平台下开发应用程序并想用 x86 汇编语言提高程序性能的软件开发者。
- 在非 Windows 环境下开发应用程序并想要学习 x86 汇编语言编程的软件开发者。
- 对 x86 汇编语言编程有基本了解，想要学习 x86 的 SSE 和 AVX 指令集的软件开发者。
- 想要或需要更好理解 x86 平台（包括其内部架构和指令集）的软件开发者和计算机学院的学生。

本书主要是针对 Windows 平台上的软件开发者编写的，因为示例代码采用了 Visual C++ 和微软宏汇编编译器。但是，本书并不是一本介绍如何使用微软开发工具的书，非 Windows 平台开发者也可以从本书获益，因为大多数内容的编写和介绍并不依赖任何特别的操作系统。具有 C 和 C++ 编程经验有助于读懂本书的内容和示例代码，但是并不需要读者事先具有 Visual Studio 使用经验，也不需要先学习 Windows API。

内容概要

本书的主要目的是帮助你学习 x86 汇编语言编程。为了达到这个目的，你需要全面理解 x86 处理器的内部架构和执行环境。本书的章节和内容是按照这样的思路规划的。下面简要介绍一下本书的主要议题和各章节的内容。

x86-32 核心架构——第 1 章涵盖了 x86-32 平台的核心架构，讨论了这个平台的基本数据类型、内部架构、指令操作数和内存寻址模式。这一章也简要介绍了 x86-32 的核心指令集。第 2 章讲解了利用 x86-32 核心指令集和常用编程结构编写 x86-32 汇编语言程序的基础知识。第 2 章及其后章节讨论的示例代码都是可以独立运行的程序，这意味着你可以运行、修改或者用这些代码做一些实验来提高学习效果。

x87 浮点单元——第 3 章探讨 x87 浮点单元（FPU）的架构，描述了 x87 FPU 的寄存器栈、控制字寄存器、状态字寄存器和指令集。这一章还深入探讨了用于表达浮点数和某些特殊值的二进制编码方案。第 4 章包含了一些示例，用以演示如何用 x87 FPU 指令集进行浮点运算。对于那些需要维护 x87 FPU 代码或者要在不具有 x86-SSE 和 x86-AVX 的处理器（比如 Intel 的 Quark）上工作的读者来说，本章的内容是最适用的。

MMX 技术——第 5 章描述了 x86 的第一个 SIMD 扩展，即 MMX 技术。它分析了 MMX 技术的架构，包括它的寄存器组、操作数类型和指令集。这一章也讨论了一些相关课题，包括 SIMD 处理概念和组合整型运算。第 6 章包含了用以演示基本 MMX 运算的示例代码，包括组合整型运算（回绕方式和饱和方式）、整数矩阵处理和如何正确地在 MMX 和 x87 FPU 代码之间切换。

流式 SIMD 扩展——第 7 章的焦点是流式 SIMD 扩展（SSE）。x86-SSE 为 x86 平台新增了一组 128 位的寄存器，并增加了一系列指令，用以支持不同的数据类型，包括组合整型、组合浮点数（单双精度）和字符串类型的数据。第 7 章还讨论了 x86-SSE 的标量浮点运算功能，对于那些需要进行标量浮点计算的应用程序来说，这个功能可以大大简化算法并提高性能。第 8 章到第 11 章包含了一系列使用 x86-SSE 指令集的示例代码。比如用 x86-SSE 的组合整型去进行图像处理，例如直方图构建和像素阈值化。这些章节也包含了示例代码来演示如何用 x86-SSE 对组合浮点数、标量浮点数和字符串进行计算和处理。

高级向量扩展——第 12 章探讨 x86 最新的 SIMD 扩展——高级向量扩展（AVX）。该章解释了 x86-AVX 执行环境、数据类型和寄存器组以及最新的三目指令格式。同时也讨论了 x86-AVX 的数据广播、收集、排列（permute）功能以及与 x86-AVX 一起引入的扩展，包括融合乘加（fused-multiply-add，FMA）、半精度浮点和新的通用寄存器指令。第 13 章到第 16 章包含了一系列示例代码来演示如何使用 x86-AVX 的各种计算资源，包括对组合整型、组合浮点和标量浮点操作数进行计算的 x86-AVX 指令。这些章节还包含了例子来演示数据广播、收集、排列和 FMA 指令的用法。

x86-64 核心架构——第 17 章探讨的是 x86-64 平台，包括这个平台的核心架构、所支持的数据类型、通用寄存器和状态标志，也解释了为支持 64 位操作数和内存寻址而对 x86-32 平台所做的扩展。这一章最后讨论了 x86-64 指令集，包括那些不再支持的指令。第 18 章用很多示例代码演示了 x86-64 汇编语言编程的基本知识。示例代码包括如何用不同大小的操作数进行整型运算、内存寻址模式、标量浮点运算以及常用的编程结构。第 18 章还介绍了 C++ 调用 x86-64 汇编语言函数的调用约定。

x86-64 SSE 和 AVX——第 19 章描述了如何在 x86-64 平台上使用增强的 x86-SSE 和 x86-AVX，讨论了各自的执行环境和扩展的数据寄存器组。第 20 章包含了在 x86-64 核心架构中使用 x86-SSE 和 x86-AVX 指令集的示例代码。

高级主题——本书最后两章讨论的是与 x86 汇编语言编程相关的高级内容和优化技术。第 21 章讨论了 x86 处理器微架构的关键点，包括它的前端流水线、乱序执行模型和内部执行单元。这一章还讨论了一些编程技术，让你的 x86 汇编语言程序在时间和空间上都很高效。第 22 章包含了几个展现高级汇编语言技术的示例代码。

附录——本书包括三个附录（可从下面的 Apress 网站或华章网站 www.hzbook.com 下载）。附录 A 介绍了使用微软的 Visual C++ 和宏汇编器的简要教程。附录 B 总结了 x86-32 和 x86-64 的调用约定，使用汇编语言编写的函数必须遵守这些约定才可以被 Visual C++ 函数所调用。附录 C 列出了关于 x86 汇编语言编程的参考文献和更多资源。

示例代码要求

可以从 Apress 网站 http://www.apress.com/9781484200650 下载本书的示例代码。编译和运行示例代码的软硬件要求如下：

- 一台包含新近微架构 x86 处理器的个人电脑。所有 x86-32、x87 FPU、MMX 和 x86-SSE 示例代码都可以运行在 Nehalem 或其以后的微架构处理器上。很多示例程序也可以在更老的微架构处理器上执行。AVX 和 AXV2 示例代码分别要求 Sandy Bridge 和 Haswell 微架构处理器。
- 微软 Windows 8.x 或者 Windows 7（Service Pack 1）。x86-64 示例代码需要运行在 64 位 Windows 操作系统上。
- Visual Studio Professional 2013 或者 Visual Studio Express 2013 for Windows Desktop。Express 版本可以从微软的网站 http://msdn.microsoft.com/en-us/vstudio 上免费下载。推荐使用 Update 3 版本。

警告 本书所有示例代码的主要目的是解释这本书中的议题和技术，很少考虑软件工程中的一些重要问题，比如鲁棒的错误处理、安全性、稳定性、舍入误差等。若你决定把这些示例代码用在自己的程序里，则应该考虑上述问题。

术语和惯例

这里给本书中使用的术语下个定义。*函数*、*子程序*或者*过程*是一段独立的可执行代码，接受 0 个或更多参数，完成一个操作，并且可以选择性地返回一个值。通常函数被处理器的调用指令所调用。*线程*是可以被操作系统管理和调度的最小执行单元。*任务*或者*进程*包含一个或多个共享同一逻辑内存空间的线程。*应用*或*程序*是包含至少一个任务的一个完整软件包。

术语 x86-32 和 x86-64 分别用来描述 x86 处理器的 32 位和 64 位特征、资源以及处理能力。x86 用以代表 32 位和 64 位架构的共有特征。x86-32 模式和 x86-64 模式表示处理器的特定执行环境，它们之间的主要不同是，x86-64 模式支持 64 位寄存器、操作数和内存寻址。x86-SSE 表示流式 SIMD 扩展，x86-AVX 表示高级向量扩展。当讨论特定 SIMD 增强指令时，会使用 SSE、SSE2、SSE3、SSSE3、SSE4、AVX 和 AVX2 这样的英文简称。

其他资源

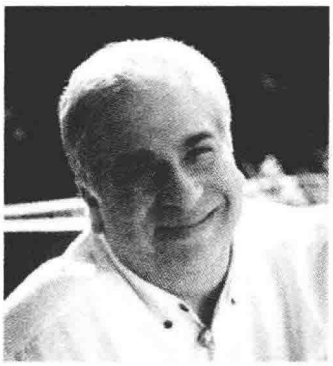
Intel 和 AMD 提供了大量与 x86 有关的文档。附录 C 列出了许多对初学者和有经验的 x86 汇编语言程序员有用的资源。其中《Intel 64 and IA-32 Architectures Software Developer's Manual-Combined Volumes: 1, 2A, 2B 2C, 3A, 3B and 3C (Order Number: 325462)》的第二卷最为重要。这一卷中对每一条处理器指令都给出了非常全面的信息，包括详细的操作过程、使用的所有操作数、会影响到的状态标志和可能导致的异常。当你开发自己的 x86 汇编语言函数的时候，强烈建议你查阅这个文档来核对指令的用法。

致谢

出版一本书和拍一部电影有很多相似之处。电影的预告片赞美主角的出色表演，书的封面要“鼓吹”一下作者。不论是演员还是作者，他们的努力最终都要接受大众的检阅。而且，无论是出品一部电影还是出版一本书，都离不开执着的精神、高超的技能和富有创造力的专业幕后团队。本书也不例外。

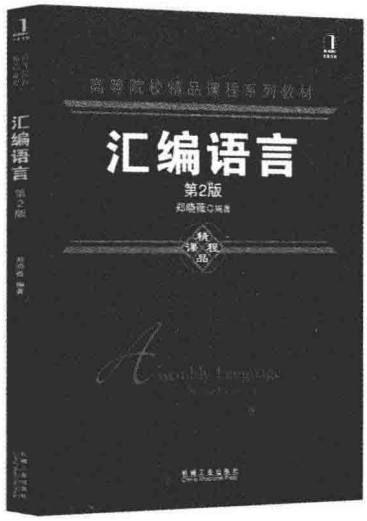
首先我要感谢 Patrick Hauke，在本书的孕育阶段，他便积极支持这个项目并给我非常有价值的建议。还要感谢 Steve Weiss，他的丰富编辑经验指引我们克服了出版中的各种难题。我非常感谢 Melissa Maldonado 的努力，让我和其他每个人专注于本书的编写。我要感谢 Paul Cohen 细致的技术审查和很多切实可行的建议。文字编辑 Kezia Endsley 和校对 Ed Kusswurm 的辛勤工作和富有建设性的反馈都值得鼓掌和表扬。剩下的任何不完美之处我愿意负全责。

我还要感谢 Dhaneesh Kumar 和整个 Apress 出版社团队的奉献；感谢 Vyacheslav Klochkov 和 Mitch Bodart 指导我如何高效使用 FMA 指令，也要感谢我单位同事的支持和关注。最后，我要感谢我的父母 Armin 和 Mary 以及我的兄弟姐妹 Mary、Tom、Ed 和 John，感谢他们在我写这本书的时候给我很多鼓励。



保罗·科恩（Paul Cohen） 在x86架构的“孩提”时代（从8086开始）便加入了英特尔公司，工作26年后，从市场营销管理岗位上退休。目前他正与道格拉斯科技集团合作，代表英特尔和其他公司开发技术书籍。保罗还与青年企业家学院（YEA）合作，给初高中学生上课，教他们如何成为一个有自信心的真正企业家。同时，他也是俄勒冈州比佛顿市的交通专员和多个非营利组织的董事。

推荐阅读



汇编语言：基于x86处理器（原书第7版）

作者：基普·欧文 译者：贺莲 龚奕利 ISBN：978-7-111-53036-7 定价：99.00元

本书全面细致地讲述了汇编语言程序设计的各个方面，不仅是汇编语言本科课程的经典教材，也可作为计算机系统和体系结构的入门教材。本书专门为32位和64位Intel/Windows平台编写，用通俗易懂的语言描述学生需要掌握的核心概念，首要目标是教授学生编写并调试机器级程序，并帮助他们自然过渡到后续关于计算机体系结构和操作系统的高级课程。本书作者的网站提供了更多的资源和工具，教师和学生可以访问章目标、调试工具、补充文件，以及MASM和Visual Studio 2012的入门教程等。

主要特色

教授有效的设计技巧：自上而下的程序设计演示和讲解，让学生将技术应用于多个编程课程。

理论联系实践：学生将在机器级水平编写软件，为以后在任何操作系统/面向机器的环境中工作做好准备。

灵活的课程安排：教师可结合具体情况以不同的顺序和深度覆盖可选章节主题。

汇编语言（第2版）

作者：郑晓薇 编著 ISBN：978-7-111-44450-3 定价：39.00元

本书作者根据多年讲授汇编语言课程的教学经验以及对汇编语言课程的教学改革，以现代教育理论为基础，精心设计了本书的结构。全书以80X86系列微型计算机为基础，以MASM5.0为汇编上机实验环境，重点介绍Intel8086指令系统。第2版在上版的基础上修订了部分内容，特别是对实验内容进行了改进，增加了两节新的实验，以便适应更多的应用。

本书特色包括：（1）以实例驱动教学。（2）启发式设问引导教学。（3）构造学习框架。（4）实验训练贯穿始终。

x86-32 核心架构

本章将从应用程序的角度解析 x86-32 核心架构。我们会从 x86 平台的简要历史开始介绍，以便为后续的讨论提供一个参考框架。接下来会回顾 x86 的数据类型，包括基本类型、数字类型和组合类型。而后，我们将深入挖掘 x86-32 的内部架构细节，包括执行单元、通用寄存器、状态标志、指令操作数和内存寻址模式。本章的最后一部分是对 x86-32 指令集的概览。

与 C 和 C++ 这样的高级语言不同，汇编语言编程需要软件开发者在写代码之前比较全面地理解目标处理器的架构特征。这一章的内容将满足大家的这一需要，并为理解第 2 章中的示例代码奠定基础。本章也为理解第 17 章将讨论的 x86-64 核心架构准备了一些基础。

1.1 简史

在深入解析 x86-32 平台的技术细节之前，让我们先来看一下它的简要历史，这将有助于理解这个架构是如何随着时间而演变的。在下面将要介绍的回顾内容中，我们将把主要精力放在那些对使用 x86 汇编语言的软件开发者产生重大影响的处理器产品和架构变化上。需要对 x86 产品线有更全面理解的读者请参考附录 C 所列出的资源。

x86-32 平台的第一个版本是 1985 年问世的英特尔 80386 微处理器。80386 扩展了其 16 位前身的架构，新增的特性包括 32 位宽的寄存器和数据类型、平坦地址模型、4GB 的逻辑地址空间以及分页虚拟内存。80486 处理器通过增加芯片上高速缓存（cache）和指令优化提高了性能。另外，大多数版本的 80486 CPU 都包含集成的 x87 浮点单元（FPU），不再像 80386 那样需要和分离的 80387 浮点单元协同工作。

1993 年推出的奔腾处理器进一步扩展了 x86-32 架构，开创了被称为 P5 的微架构。微架构定义了处理器内部单元的组织结构，包括寄存器文件、执行单元、指令流水线、数据总线和高速缓存。一种微架构常常被多个产品线的处理器所使用。P5 微架构在性能方面的增强包括并行的（两条）指令执行流水线、64 位外部数据总线，并把芯片上高速缓存分成指令缓存和数据缓存两种。P5 微架构的后续版本包含了一种新的计算资源，称为 MMX（1997 年）。MMX 技术支持使用 64 位宽的寄存器对组合整型做单指令多数据（SIMD）运算。

1995 年推出的奔腾 Pro 处理器和 1997 年推出的奔腾 II 处理器都使用了 P6 微结构，它所采用的三路超标量设计进一步扩展了 x86-32 平台。这种设计意味着可以在一个时钟周期内解码、分发和执行三条不同的指令（平均）。P6 微架构的其他改变包括支持指令的乱序执行、改进的分支预测算法以及投机执行。也是基于 P6 微架构的奔腾 III 处理器是在 1999 年发布的。它包含了一种新的 SIMD 技术，被称为流式 SIMD 扩展（streaming SIMD extension，SSE）。SSE 为 x86-32 平台增加了 8 个 128 位宽的寄存器以及支持组合单精度（32 位）浮点运算的指令。

2000 年，英特尔推出了一种名为 Netburst 的新型微架构，它包含了扩展 SSE 浮点能