TURING 图灵程序设计丛书

Practical Reverse Engineering

x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation

逆向工程实战

【美】Bruce Dang【法】Alexandre Gazet
【美】Elias Bachaalany【法】Sébastien Josse 著
单业 译

逆向工程领域先驱Rolf Rolles审校并鼎力推荐

包含针对真实病毒和后门程序的练习和实验

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Practical Reverse Engineering

x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation

逆向工程实战

【美】Bruce Dang【法】Alexandre Gazet
【美】Elias Bachaalany【法】Sébastien Josse 著
单业 译

人民邮电出版社
北京

图书在版编目（CIP）数据

逆向工程实战 /（美）邓（Dang, B.）等著 ；单业译 .
-- 北京 ：人民邮电出版社，2015.8
（图灵程序设计丛书）
ISBN 978-7-115-38893-3

Ⅰ. ①逆… Ⅱ. ①邓… ②单… Ⅲ. ①软件工程
Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2015)第063536号

内 容 提 要

本书是一本涵盖 x86、x64 和 ARM 操作系统的逆向工程类图书，由浅入深地讲解了包括 Windows 内核模式代码的恶意软件和驱动程序、虚拟机保护技术等内容。作者通过大量真实案例和示例，提供了系统化的解决方案。

本书适合所有程序员和想要开始学习逆向工程的读者阅读。

◆ 著　[美] Bruce Dang [法] Alexandre Gazet
[美] Elias Bachaalany [法] Sébastien Josse
译　单 业
责任编辑　朱 巍
执行编辑　李岩俨
责任印制　杨林杰

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编 100164　电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京天宇星印刷厂印刷

◆ 开本：800×1000 1/16
印张：18.25
字数：431千字　2015年8月第1版
印数：1－4 000册　2015年8月北京第1次印刷
著作权合同登记号 图字：01-2014-4712号

定价：59.00元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

版权声明

本书献给那些对知识不懈追求并无私分享的人们。

引　言

逆向工程的学习过程类似于成年人学习外语的过程。学习外语的第一个阶段是学字母表，这些字母构造了具有明确语义的单词。接下来的阶段涉及对语法规则的理解，语法规则决定了如何把单词连接起来组成正确的句子。一个人掌握了这些规则之后，就开始学习如何把多个句子组织起来，清晰表达复杂的思想。最终，学习者就能够阅读大部头书籍，即使这些书籍风格不同，他仍然能够掌握其中的思想。到达这个阶段之后，学习者就可以阅读关于这门语言更深奥主题（如历史句法学、音韵学等方面）的参考书了。

在逆向工程领域中，“语言”就是体系结构和汇编语言，“单词”则是一条条汇编指令，“段落”即是汇编指令序列，一本“书”就是一个程序。要完全理解一本书，读者不仅仅要掌握单词和语法，还应了解叙事风格和结构，以及不成文的写作惯例等因素。同样，理解计算机程序需要掌握的概念也不仅仅是汇编指令。

某种程度上说，开始从一本书去学习一个全新的技术主题有点让人望而却步。但如果我们因此声称逆向工程的学习过程很简单，可以通过阅读本书完全掌握，则是在误导读者。实际上这个学习过程是非常复杂的，因为这项技术涉及几个分散的知识领域。比如，一个高效的逆向工程师需要对计算机体系结构、系统编程、操作系统、编译器等技术都有所了解；对于某些领域来说，还需要具备较强的数学背景。那么如何确定从哪里开始学起呢？这取决于各人的经验和技能。而我们无法适应每个人的背景条件，因此在这个简介中为那些没有任何编程背景的读者列出了学习和阅读的方法。读者应该可以从中找到自己的位置，并从那里开始学习。

为了讨论方便，这里把逆向工程不严格地定义为对一个系统的理解过程。这是一个解决问题的过程。一个系统可能是一个硬件设备、一个软件程序、一个物理或化学过程等。对于本书来说，这个系统就是指一个软件程序。要理解一个程序，首先必须理解软件是如何编写的。因此，第一个要求是了解如何通过某种语言（比如 C、C++、Java 等）为计算机编写程序。由于 C 语言的简洁高效及其用途的广泛性，我们建议从它开始学起。这里给出一些值得考虑的优秀参考书，包括 Brian Kernighan 和 Dennis Ritchie 所著的《C 程序设计语言》以及 Samuel Harbison 所著的《C 语言参考手册》。在熟悉了编写、编译和调试简单程序的过程后，就可以考虑阅读 Peter Van Der Linden 所著的《C 专家编程》。到了这个阶段，你应该已经熟悉了变量、作用域、函数、指针、条件式、循环、调用栈、库等高级概念。对于栈、队列、链表和树等数据结构的了解可能是有所帮助的，不过就目前阶段来看还不是必需的。除此以外，你还可以简单阅读下面这些书籍，以求更好地了解程序是如何真正被连接到一起的：Alfred Aho、Ravi Sethi 和 Jeffrey Ullman 所著的《编译原理》

以及 John Levine 的《链接器和加载器》。阅读这些书的主要目的是了解基本概念，目前还不需要完全理解全部内容（关于这一点，稍后会加以详述）。学有余力的读者还可以考虑阅读 Steven Muchnick 所著的《高级编译器设计与实现》。

一旦对程序的编写、运行和调试过程有了充分的理解，就可以开始探索程序的执行环境了，这包括进程和操作系统。我们建议首先通过从英特尔公司的《Intel 64 和 IA-32 体系结构软件开发人员手册（卷 1）：基本体系结构》来学习 Intel 处理器，特别要注意其中的 2 ~ 7 章。这些章节解释了现代计算机的基本组成部分。对 ARM 体系结构感兴趣的读者可以阅读 ARM 的 *Cortex-A Series Programmer's Guide* 和 *ARM Architecture Reference Manual ARMv7-A and ARMv7-R Edition*。尽管我们这本书讲述了 x86/x64/ARM 体系结构，但并不会讨论这些体系结构的全部细节。（我们假定读者会根据需要来参考以上介绍的这些手册。）通过浏览这些手册，读者应该能够对计算系统的技术构建模块有基本的识别和判断。如果要对概念有更深入的理解，可以阅读 Andrew Tanenbaum 所著的《计算机组成：结构化方法》一书。另外，所有读者都应该查阅 *Microsoft PE and COFF Specification*。到了这一阶段，读者应该已经具备了阅读并理解第 1 章和第 2 章所需的所有背景知识。

接下来就要探索操作系统了。尽管操作系统的种类很多，但多数概念是共通的，包括进程、线程、虚拟内存、特权隔离、多任务等。要理解这些概念，最好的方法就是阅读 Andrew Tanenbaum 所著的《现代操作系统》一书。尽管该书对于理解概念很有帮助，但其中并没有讨论实际操作系统中的很多重要技术实现细节。对于 Windows 来说，可以考虑阅读 Peter Viscarola 和 Anthony Mason 所著的 *Windows NT Device Driver Development*。尽管这是一本关于驱动程序开发的书，但其中的背景知识章节对 Windows 操作系统做了完美而具体的介绍。（这本书也是本书第 3 章很好的补充材料。）另外一本很有启发性、也是关于 Windows 内存管理的优秀著作是 Enrico Martignetti 所著的 *What Makes It Page? The Windows 7 （x64） Virtual Memory Manager*。

到了这一阶段，读者应该已经具备了阅读本书第 3 章所需的所有背景知识。可以考虑开始学习 Win32 编程了。Johnson Hart 所著的 *Windows System Programming , 4th Edition* 以及 Jeffrey Richter 和 Christophe Nasarre 所著的《Windows 核心编程》都是非常优秀的参考书。

针对于第 4 章调试与自动化的内容可以考虑的参考书是 Tarik Soulami 所著的 *Windows Debugging: A Practical Guide to Debugging and Tracing Strategies in Windows*，以及 Mario Hewardt 和 Daniel Pravat 所著的《Windows 高级调试》。

阅读第 5 章则需要对汇编语言有充分的理解，应该在阅读 x86/x64/ARM 章节之后开始阅读。相关背景知识可以通过阅读 Christian Collberg 和 Jasvir Nagra 所著的《软件加密与解密》一书获得。

注意　本书包含了针对真实病毒和后门程序的练习和实验。我们有意这么做是为了确保读者能够立即把新学的技术应用于实际。这些恶意代码示例用字母序来指代（如示例 A、B、C 等），可以在附录中找到其对应的 SHA1 散列值。出于法律方面的顾虑，我们决定不随本书发布这些示例。但是读者可以通过搜索各种恶意代码库比如 www.malware.lu 来下载，或者在 http://kernelmode.info 这样的论坛上请求得到这些示例。这些示例中有许多是来自于引发全球性新闻的著名黑客事件，因此是很有趣的。也可能会有热心的读者搜集所有这些示例并打包通过 BitTorrent 发布。如果前面这些方法都行不通，请尽管给本书作者发邮件。分析这些示例的时候，需要确保你是在安全的环境中，以免不小心感染了这些病毒。

另外，为了帮助读者熟悉 Metasm，我们准备了两个练习脚本：symbolic-execution-lvl1.rb 和 symbolic-execution-lvl2.rb。这些问题的解答过程会带你进入深入了解 Metasm 的旅程。可以在 www.wiley.com/go/practicalreverseengineering 找到这些脚本。

一定要意识到，这些练习是本书至关重要的部分。本书的编写模式就是按照这种思路来的。如果你仅仅是阅读而不做练习的话，不会有深入的理解和太多收获。读者尽可以写出对这些问题的解答，或发布在自己的博客上，这样其他人可以从中学习；你也可以把答案发布在逆向工程 reddit （www.reddit.com/r/ReverseEngineering），并得到社区（也许就是本书作者）的反馈。如果你成功完成了所有练习，那一定要好好奖励一下自己，然后请把你的简历发给作者。

要成为高效的逆向工程师，学习的旅程是漫长又耗时的，需要耐心和坚持。在学习的道路上你可能会多次失败（比如无法理解书中概念或者无法完成本书习题），但是不要放弃。记住，失败是成功的一部分。通过这个引言及后续的章节，你应该已经准备好了踏上学习的征程。

我们非常希望了解读者的学习经验，并据此进一步调整本书内容，改善本书质量。读者的反馈对于我们及未来可能的新版本是无比宝贵的。反馈意见和问题可发送给 Bruce Dang（bruce.dang@gmail.com）、Alexandre Gazet（agazet@quarkslab.com）或 Elias Bachaalany（elias. bachaalany@gmail.com）。

致　谢

撰写此书是我们经历过的最有趣也是最耗时的工作。这本书呈现了当年我们开始学习逆向工程时希望了解的东西，那时候，书籍和在线资源都极度缺乏（那时候还没有博客这种东西），我们主要是通过朋友和独立的试错实验来学习这门技术。那时信息安全“产业”也还不存在。而现在，一切都已经不一样了，我们有反编译器、Web 扫描器、静态源码扫描器、云（？）和 APT（无法想象！），还有无数博客、论坛、书籍和讲授逆向工程技术的实体课堂。这些资源的质量参差不齐，有些水平很低，却厚颜出版或提供出来，为的就是利用计算机安全领域上升的需求市场来牟利；有些质量非常高，却没有得到广泛的注意和阅读，只因为缺少宣传，或只是因为它们过于难懂。对于逆向工程的学习没有统一的基础资源可以利用。我们希望这本书可以为读者提供这个基础。

现在是本节的主体部分——感谢耐心帮助我们走到现在的人们。所有作者都感谢 Rolf Rolles 对第 5 章的贡献。Rolf 是逆向工程领域的一位先驱者。他对虚拟机解混淆、逆向工程应用程序分析以及二进制分析的开创性工作，影响并激励了新一代逆向工程师。我们期待他能够继续对本领域作出贡献并激励其他人这么做。另外，还要感谢我们的哥们 Matt Miller 作为技术审阅者为本书作出的贡献。Matt 是我们领域的又一位先驱者，他对 Windows 中的攻击缓解做出了开创性的工作。他致力于研究细节，并且乐于帮助他人学习，是所有人的榜样。最后，感谢 Carol Long、John Sleeva、Luann Rouff 和 John Wiley & Sons 全体员工在整个出版过程中所付出的努力。

——全体作者

我要感谢我的父母为我提供更好的生活所作出的牺牲；感谢我的姐姐 Ivy Dang 和哥哥 Donald Dang 的支持和鼓励；另外，感谢多年来的朋友 Rolf Rolles，他也是我的理性之源。我的榜样并不多，但我的世界观的形成离不开以下朋友的影响：Le Thanh Sang、Vint Cerf 和 Douglas Comer。在大学里，我从 David Knetchges 那体会了中国文学的乐趣，从 Kyoko Tokuno 那里学习了佛法知识，从 Richard Salomon（他从岩石和硬币中获得了那么多的思考）那里学习了印度历史，跟随 Daniel Waugh 学习了中亚历史，跟随 Nyan-Ping Bi 学习了中文。虽然他们不是逆向工程师，但是他们的热忱和奉献精神永远激励我，使我成为一个更好的人，更好的工程师。如果我能够更早遇到他们，我的职业生涯可能会有很大不同。

在我的整个职业生涯中，我非常幸运遇到了影响我的睿智的人（以下排名没有特别顺序）：Alex Carp、rebel、Navin Pai、Jonathan Ness、Felix Domke、Karl J.、Julien Tinnes、Josh Phillips、

Daniel Radu、Maarten Boone、Yoann Guillot、Ivanlef0u（感谢您对我们的款待）、Richard van Eeden、Dan Ho、Andy Renk、Elia Florio、Ilfak Guilfanov、Matt Miller、David Probert、Damian Hasse、Matt Thomlinson、Shawn Hoffman、David Dittrich、Eloi Vanderbeken、LMH、Ali Rahbar、Fermin Serna、Otto Kivling、Damien Aumaitre、Tavis Ormandy、Ali Pezeshk、Gynvael Coldwind、Anakata（罕见的奇才）、Richard van Eeden、Noah W.、Ken Johnson、Chengyun Yu、Elias Bachaalany、Felix von Leitner、Michal Chmielewski、sectorx、Son Pho Nguyen、Nicolas Pouvesle、Kostya Kortchinsky、Peter Viscerola、Torbjorn L.、Gustavo di Scotti、Sergiusz Fonrobert、Peter W.、Ilja van Sprundel、Brian Cavenah、upb、Maarten Van Horenbeeck、Robert Hensing、Cristian Craioveanu、Claes Nyberg、Igor Skorchinsky、John Lambert、Mark Wodrich（模范佛教徒）、David Midturi、Gavin Thomas、Sebastian Porst、Peter Vel、Kevin Broas、Michael Sandy、Christer Oberg、Mateusz“j00ru”Jurczyk、David Ross 和 Raphael Rigo。Jonathan Ness 和 Damian Hasse 一直赞同我在行事上的与众不同，并给予我机会去探索（无论成功还是失败）。如果这里遗漏了谁，敬请原谅。

下面的人对本书初稿提供了反馈和改进意见：Michal Chmielewski、Shawn Hoffman、Nicolas Pouvesle、Matt Miller、Alex Ionescu、Mark Wodrich、Ben Byer、Felix Domke、Ange Albertini、Igor Skorchinsky、Peter Ferrie、Lien Duong、iZsh、Frank Boldewin、Michael Hale Ligh、Sebastien Renaud、Billy McCourt、Peter Viscerola、Dennis Elser、Thai Duong、Eloi Vanderbeken、Raphael Rigo、Peter Vel 以及 Bradley Spengler（真正的杰出人士）。没有他们富有洞察力的评论和建议，本书绝大部分内容都将不忍卒读。当然，遗留的错误由我完全负责。

还有无数不知名的朋友帮助形成了我的知识领域，间接促成了这本书的诞生。

我还要感谢 Omni Group 的 Molly Reed 和 Tami Needham 为我们提供了 OmniGraffle 的许可证，用于在早期稿件中创建草图。

最后，我要感谢 Alex、Elias 和 Sébastien 帮助我编写了本书。没有他们，这本书不可能完成。

——Bruce

首先，我要感谢 Bruce Dang 邀请我参与到这个杰出的项目中。这是一段漫长而又精妙的旅程。Rolf Rolles 最开始就参与了这个项目，感谢他花费了无数时间与我一起设想第 5 章的内容，并搜集材料。然后 Sébastien Josse 同意加入我们，他的贡献是无价的，如果没有他，这本书绝不会是这样。谢谢你，Seb。

我还要感谢我的朋友 Fabrice Desclaux、Yoann Guillot 和 Jean-Philippe Luyten，感谢他们宝贵的反馈意见。

最后，感谢 Carol Long 使这本书得以出版，感谢 John Sleeva 帮助我们保持正轨。

——Alexandre

首先我要感谢我的朋友和同事 Bruce Dang，给我这个机会参与到这个项目中。我还要感谢所有朋友和同事的帮助和支持。特别是，我要感谢 Daniel Pistelli（Cerbero GmbH 的 CEO）、Michal Chmielewski、Swamy Shivaganga Nagaraju 和 Alexandre Gazet 在我编写此书过程中提供的技术支

持和反馈意见。

我还要感谢 Ilfak Guilfanov（Hex-Rays SA 的 CEO）先生。在 Hex-Rays 工作期间我从他身上学到了很多。他创造 IDA Pro 时的努力工作，耐心和坚持不懈永远激励着我。

还要多谢 John Wiley & Sons 出版社能给我们这个机会出版此书。感谢策划编辑 Carol Long 的督促和职业帮助，感谢项目编辑 John Sleeva 和文字编辑 Luann Rouff 付出的精力、耐心和努力。

——Elias

我要感谢 Alexandre、Elias 和 Bruce 给我这个机会参与此书。我还要感谢 Jean-Philippe Luyten 使我和他们三人保持联系。最后感谢编辑 Carol Long 和 John Sleeva 在项目实施过程中所提供的帮助以及所表现出的职业精神。

——Sébastien

目　录

第1章 x86与x64

x86是基于Intel 8086处理器的小端（little-endian）体系结构。本书中讨论x86是指《Intel 64和IA-32体系结构软件开发人员手册》中定义的Intel体系结构（IA-32）的32位实现。一般而言，它可以在两种操作模式下运行：*实模式*和*保护模式*。实模式是指处理器刚刚上电后只支持16位指令集的状态。保护模式是指处理器支持虚拟内存、分页以及其他功能的状态，也是运行当代操作系统的状态。这个体系结构的64位扩展称为x64或x86-64。这一章中讨论的是运行于保护模式下的x86体系结构。

x86通过一种称为*环级别*（ring level）的抽象来支持特权隔离（privilege separation）。处理器支持4种特权级别，编号为0到3（通常不使用ring1和ring2，所以这里不予讨论）。ring0具有最高特权级别，可以修改所有的系统设置。而ring3的特权级别最低，只能读取/修改部分系统设置。因此现代操作系统通常将用户模式应用程序运行在ring3级别，将内核运行于ring0级别，以此实现用户/内核特权隔离。ring级别编码于CS寄存器中，在官方文档中有时被称为当前特权级别（Current Privilege Level，CPL）。

本章讨论定义于《Intel 64和IA-32体系结构软件开发人员手册》卷1~3中的x86/IA-32体系结构（www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html）。

1.1 寄存器组与数据类型

运行于保护模式下的x86体系结构有8个32位通用寄存器（General Purpose Registers，GPR）：EAX、EBX、ECX、EDX、EDI、ESI、EBP、ESP。这些寄存器中有一部分还可以进一步分为8位和16位寄存器。指令指针存储在EIP寄存器中。图1-1中展示了这个寄存器组。表1-1介绍了部分GPR的用途。

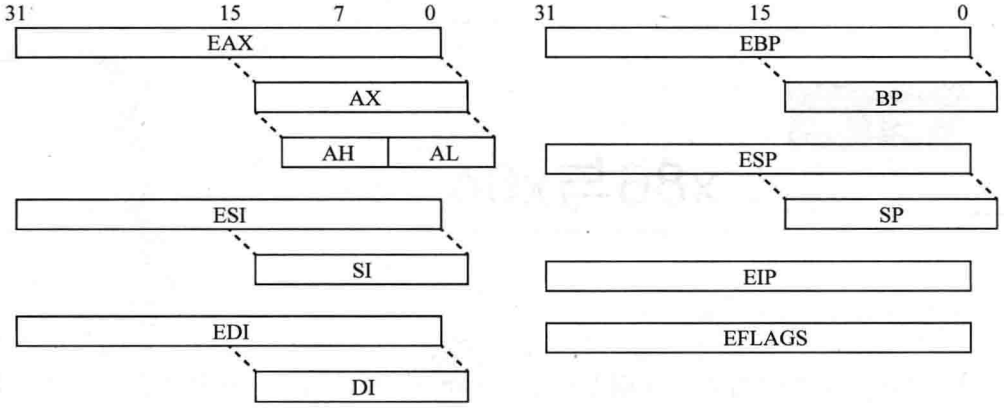

图　1-1

表1-1　部分GPR及其用途

寄存器	用　途
ECX	循环计数
ESI	字符串/内存操作的源
EDI	字符串/内存操作中的目标
EBP	帧基指针
ESP	栈指针

常用的数据类型有以下几种。

- 字节（Byte）：8位，比如AL、BL、CL。
- 字（Word）：16位，比如AX、BX、CX。
- 双字（Double Word）：32位，比如EAX、EBX、ECX。
- 四字（Quad Word）：64位。虽然x86并不支持64位GPR，但是在某些场景下可以把两个寄存器（通常是EDA:EAX）的内容合并起来当作64位值。比如，RDTSC指令会把一个64位值写入EDX:EAX寄存器。

32位寄存器EFLAGS用于存储算术运算状态以及其他运行状态（比如陷阱标志位）。举例来说，如果前一个ADD运算的结果是0，那么ZF标志位就会被设置为1。最初EFLAGS中的标志位用于实现条件分支跳转。

除了通用寄存器、EIP和EFLAGS，还有一些寄存器用于控制重要的底层系统机制，比如虚拟内存、终端和调试等。例如，CR0寄存器控制分页机制的开关，CR2寄存器中保存着导致缺页（page fault）异常发生的线性地址，CR3是分页数据结构的基地址，CR4控制硬件虚拟化设置。DR0~DR7寄存器用于设置内存断点。在1.5节中会详细介绍这些寄存器。

注意　尽管调试寄存器有8个，但系统只支持4个内存断点（DR0~DR3），其余寄存器用于保存状态。

还有一些特定于模型的寄存器（Model-Specific Register，MSR）。正如其名字所暗示的，Intel和AMD的不同处理器中这些寄存器也会有所不同。每个MSR用名字和一个32位数字标识，通过RDMSR/WRMSR指令来读写。只有运行于ring0级别的代码才能够访问这类寄存器；它们通常用于存储特殊计数或者实现底层功能。举例来说，SYSENTER指令会把执行跳转到IA32_SYSENTER_EIP MSR（0x176）存储的地址处，通常这里是操作系统的系统调用处理函数。本书中会在遇到的地方对具体MSR加以讨论。

1.2 指令集

x86指令集为寄存器和内存之间的数据移动提供了很大的灵活性。数据移动可以分为5种方式：

- 立即数到寄存器
- 寄存器到寄存器
- 立即数到内存
- 寄存器到内存，或反方向
- 内存到内存

前面4种方式是所有现代体系结构都支持的，而最后一种则是x86所独有的。像ARM这样的经典RISC体系结构只支持通过加载/存储指令（LDR/STR）从内存读出或向内存写入数据。比如递增内存中数据值需要执行3条指令。

(1) 把数据从内存读入到寄存器中（LDR）。

(2) 寄存器加1（ADD）。

(3) 把寄存器值写回内存（STR）。

而对于x86来说，因为可以直接访问内存，这样的操作只需要一条指令（INC或ADD）。MOVS指令可以同时读写内存。

ARM

```
01: 1B 68        LDR     R3, [R3]
 ; 读入地址R3处的值并保存在R2中
02: 5A 1C        ADDS    R2, R3, #1
 ; 加1
03: 1A 60        STR     R2, [R3]
 ; 把更新后的值写回地址R3处
```

x86

```
01: FF 00        inc     dword ptr [eax]
; 直接递增地址EAX处的值
```

x86的另一个重要特性是使用了变长指令——指令的长度从1到15字节不等。而在ARM上，指令长度只能是2字节或4字节。

1.2.1 语法

根据汇编器/反汇编器的不同，x86汇编代码有两种语法记法：Intel和AT&T。

Intel

```
mov ecx, AABBCCDDh
mov ecx, [eax]
mov ecx, eax
```

AT&T

```
movl $0xAABBCCDD, %ecx
movl (%eax), %ecx
movl %eax, %ecx
```

注意上面两套指令是相同的，只是写法不同。Intel和AT&T记法有若干不同之处，最重要的是以下几点。

- AT&T记法在寄存器名前加前缀%，立即数前加$。Intel记法不加前缀。
- AT&T记法加入了指示指令宽度的后缀，比如MOVL（长整型）、MOVB（字节）等。而Intel记法没有这种标识。
- AT&T记法把源操作数放在目标操作数之前。而Intel记法与之相反。

Windows系统上的反汇编器/汇编器以及其他的逆向工程工具（IDA Pro、OllyDbg、MASM等）通常使用Intel记法，而UNIX系统上的工具（GCC）通常遵从AT&T记法。Intel记法在实践中占据统治地位，也是本书中使用的记法。

1.2.2 数据移动

指令用于操作来自寄存器或主内存中的数据。移动数据最常用的指令是MOV，它最简单的用途就是把寄存器中的值或立即数移动到另一个寄存器中。比如：

```
01: BE 3F 00 0F 00      mov   esi, 0F003Fh ; 设置ESI = 0xF003
02: 8B F1               mov   esi, ecx     ; 设置ESI = ECX
```

MOV指令的另一常见用法是从内存读出/写入数据。与其他汇编语言惯用法一样，x86使用方括号[]标识内存地址。（唯一的例外是LEA指令，它也使用[]，但不一定是指内存。）内存访问有几种不同的方法，我们从最简单的开始。

汇编代码

```
01: C7 00 01 00 00+     mov   dword ptr [eax], 1
 ; 把地址EAX处的内存设置为1
02: 8B 08               mov   ecx, [eax]
; 把ECX写为地址EAX处的值
03: 89 18               mov   [eax], ebx
; 把EBX值写入地址为EAX的内存
04: 89 46 34            mov   [esi+34h], eax
; 把EAX值写入地址为 (ESI+0x34)的内存
05: 8B 46 34            mov   eax, [esi+34h]
; 把地址(ESI+0x34)处的值写入EAX
06: 8B 14 01            mov   edx, [ecx+eax]
; 把地址(ECX+EAX)的值写入EDX
```