Modern X86 Assembly Language Programming

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

Daniel Kusswurm

Apress®

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

ISBN-13 (pbk): 978-1-4842-0065-0

ISBN-13 (electronic): 978-1-4842-0064-3

Managing Director: Welmoed Spahr
Lead Editors: Steve Weiss (Apress); Patrick Hauke (Intel)
Technical Reviewer: Paul Cohen
Editorial Board: Steve Anglin, Gary Cornell, Louise Corrigan, James T. DeWolf, Jonathan Gennick, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing, Matt Wade, Steve Weiss
Coordinating Editor: Melissa Maldonado
Copy Editor: Kezia Endsley
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

This book is dedicated to those individuals who suffer the ravages of Alzheimer's disease and their unsung compassionate caregivers.

Contents at a Glance

Contents

About the Author

Daniel Kusswurm has over 30 years of professional experience as a software developer and computer scientist. During his career, he has developed innovative software for medical devices, scientific instruments, and image processing applications. On many of these projects, he successfully employed x86 assembly language to significantly improve the performance of computationally-intense algorithms or solve unique programming challenges. His educational background includes a BS in Electrical Engineering Technology from Northern Illinois University along with an MS and PhD in Computer Science from DePaul University.

About the Technical Reviewer

Paul Cohen joined Intel Corporation during the very early days of the x86 architecture, starting with the 8086, and retired from Intel after 26 years in sales/marketing/management. He is currently partnered with Douglas Technology Group, focusing on the creation of technology books on behalf of Intel and other corporations. Paul also teaches a class that transforms middle and high school students into real, confident entrepreneurs, in conjunction with the Young Entrepreneurs Academy (YEA) and is a Traffic Commissioner for the City of Beaverton, Oregon and on the Board of Directors of multiple non-profit organizations.

Acknowledgments

The production of a motion picture and the publication of a book are somewhat analogous. Movie trailers extol the performances of the lead actors. The front cover of a book trumpets the authors' names. Actors and authors ultimately receive public acclamation for their efforts. It is, however, impossible to produce a movie or publish a book without the dedication, expertise, and creativity of a professional behind-the-scenes team. This book is no exception.

I would like to thank Patrick Hauke for his valuable advice and championing of the book project during its conceptual stage. I am indebted to Steve Weiss for his editorial savvy and guidance through the book publishing jungle. I am extremely appreciative of Melissa Maldonado's efforts to keep me and everyone else focused and on schedule. Paul Cohen deserves kudos for his meticulous technical review and practical suggestions. Copy editor Kezia Endsley and proofreader Ed Kusswurm merit applause and recognition for their hard work and constructive feedback. I accept full responsibility for any remaining imperfections.

I would also like to thank Dhaneesh Kumar and the entire production staff at Apress for their contributions, Vyacheslav Klochkov and Mitch Bodart for their help in clarifying how to effectively use the FMA instructions, and my professional colleagues for their support and encouragement. Finally, I would like to recognize parental nodes Armin (RIP) and Mary along with sibling nodes Mary, Tom, Ed, and John for their inspiration during the writing of this book

Introduction

Since the invention of the personal computer, software developers have used assembly language to create innovative solutions for a wide variety of algorithmic challenges. During the early days of the PC era, it was common practice to code large portions of a program or complete applications using x86 assembly language. Even as the use of high-level languages such as C, C++, and C# became more prevalent, many software developers continued to employ assembly language to code performance-critical sections of their programs. And while compilers have improved remarkably over the years in terms of generating machine code that is both spatially and temporally efficient, situations still exist where it makes sense for software developers to exploit the benefits of assembly language programming.

The inclusion of single-instruction multiple-data (SIMD) architectures in modern x86 processors provides another reason for the continued interest in assembly language programming. A SIMD-capable processor includes computational resources that facilitate concurrent calculations using multiple data values, which can significantly improve the performance of applications that must deliver real-time responsiveness. SIMD architectures are also well-suited for computationally-intense problem domains such as image processing, audio and video encoding, computer-aided design, computer graphics, and data mining. Unfortunately, many high-level languages and development tools are unable to fully (or even partially) exploit the SIMD capabilities of a modern x86 processor. Assembly language, on the other hand, enables the software developer to take full advantage of a processor's entire computational resource suite.

Modern X86 Assembly Language Programming

Modern X86 Assembly Language Programming is an edifying text on the subject of x86 assembly language programming. Its primary purpose is to teach you how to code functions using x86 assembly language that can be invoked from a high-level language. The book includes informative material that explains the internal architecture of an x86 processor as viewed from the perspective of an application program. It also contains an abundance of sample code that is structured to help you quickly understand x86 assembly language programming and the computational resources of the x86 platform. Major topics of the book include the following:

- X86 32-bit core architecture, data types, internal registers, memory addressing modes, and the basic instruction set
- X87 core architecture, register stack, special purpose registers, floating-point encodings, and instruction set

- MMX technology and the fundamentals of packed integer arithmetic
- Streaming SIMD extensions (SSE) and Advanced Vector Extensions (AVX), including internal registers, packed integer and floating-point arithmetic, and associated instruction sets
- X86 64-bit core architecture, data types, internal registers, memory addressing modes, and the basic instruction set
- 64-bit extensions to SSE and AVX technologies
- X86 microarchitecture and assembly language optimization techniques

Before proceeding I should also explicitly mention some of the topics that are not covered. This book does not examine legacy aspects of x86 assembly language programming such as 16-bit real-mode applications or segmented memory models. Except for a few historical observations and comparisons, all of the discussions and sample code emphasize x86 protected-mode programming using a flat linear memory model. This book does not discuss x86 instructions or architectural features that are managed by operating systems or require elevated privileges. It also doesn't explore how to use x86 assembly language to develop software that is intended for operating systems or device drivers. However, if your ultimate goal is to use x86 assembly language to create software for one of these environments, you will need to thoroughly understand the material presented in this book.

While it is still theoretically possible to write an entire application program using assembly language, the demanding requirements of contemporary software development make such an approach impractical and ill advised. Instead, this book concentrates on creating x86 assembly language modules and functions that are callable from C++. All of the sample code and programing examples presented in this book use Microsoft Visual C++ and Microsoft Macro Assembler. Both of these tools are included with Microsoft's Visual Studio development tool.

Target Audience

The target audience for this book is software developers, including:

- Software developers who are creating application programs for Windows-based platforms and want to learn how to write performance-enhancing algorithms and functions using x86 assembly language.
- Software developers who are creating application programs for non-Windows environments and want to learn x86 assembly language programming.

- Software developers who have a basic understanding of x86 assembly language programming and want to learn how to use the x86's SSE and AVX instruction sets.
- Software developers and computer science students who want or need to gain a better understanding of the x86 platform, including its internal architecture and instruction sets.

The principal audience for Modern X86 *Assembly Language Programming* is Windows software developers since the sample code uses Visual C++ and Microsoft Macro Assembler. It is important to note, however, that this is not a book on how to use the Microsoft development tools. Software developers who are targeting non-Windows platforms also can learn from the book since most of the informative content is organized and communicated independent of any specific operating system. In order to understand the book's subject material and sample code, a background that includes some programming experience using C or C++ will be helpful. Prior experience with Visual Studio or knowledge of a particular Windows API is not a prerequisite to benefit from the book.

Outline of Book

The primary objective of this book is to help you learn x86 assembly language programming. In order to achieve this goal, you must also thoroughly understand the internal architecture and execution environment of an x86 processor. The book's chapters and content are organized with this in mind. The following paragraphs summarize the book's major topics and each chapter's content.

X86-32 Core Architecture—Chapter 1 covers the core architecture of the x86-32 platform. It includes a discussion of the platform's fundamental data types, internal architecture, instruction operands, and memory addressing modes. This chapter also presents an overview of the core x86-32 instruction set. Chapter 2 explains the fundamentals of x86-32 assembly language programming using the core x86-32 instruction set and common programming constructs. All of the sample code discussed in Chapter 2 (and subsequent chapters) is packaged as working programs, which means that you can run, modify, or otherwise experiment with the code in order to enhance your learning experience.

X87 Floating-Point Unit—Chapter 3 surveys the architecture of the x87 floating-point unit (FPU) and includes operational descriptions of the x87 FPU's register stack, control word register, status word register, and instruction set. This chapter also delves into the binary encodings that are used to represent floating-point numbers and certain special values. Chapter 4 contains an assortment of sample code that demonstrates how to perform floating-point calculations using the x87 FPU instruction set. Readers who need to maintain an existing x87 FPU code base or are targeting processors that lack the scalar floating-point capabilities of x86-SSE and x86-AVX (e.g., Intel's Quark) will benefit the most from this chapter.

MMX Technology—Chapter 5 describes the x86's first SIMD extension, which is called MMX technology. It examines the architecture of MMX technology including its register set, operand types, and instruction set. This chapter also discusses a number of related topics, including SIMD processing concepts and the mechanics of packed-

integer arithmetic. Chapter 6 includes sample code that illustrates basic MMX operations, including packed-integer arithmetic (both wraparound and saturated), integer array processing, and how to properly handle transitions between MMX and x87 FPU code.

Streaming SIMD Extensions—Chapter 7 focuses on the architecture of Streaming SIMD Extensions (SSE). X86-SSE adds a new set of 128-bit wide registers to the x86 platform and incorporates several instruction set additions that support computations using packed integers, packed floating-point (both single and double precision), and text strings. Chapter 7 also discusses the scalar floating-point capabilities of x86-SSE, which can be used to both simplify and improve the performance of algorithms that require scalar floating-point arithmetic. Chapters 8 - 11 contain an extensive collection of sample code that highlights use of the x86-SSE instruction set. Included in this chapter are several examples that demonstrate using the packed-integer capabilities of x86-SSE to perform common image-processing tasks, such as histogram construction and pixel thresholding. These chapters also include sample code that illustrates how to use the packed floating-point, scalar floating-point, and text string-processing instructions of x86-SSE.

Advanced Vector Extensions—Chapter 12 explores the x86's most recent SIMD extension, which is called Advanced Vector Extensions (AVX). This chapter explains the x86-AVX execution environment, its data types and register sets, and the new three-operand instruction syntax. It also discusses the data broadcast, gather, and permute capabilities of x86-AVX along with several x86-AVX concomitant extensions, including fused-multiply-add (FMA), half-precision floating-point, and new general-purpose register instructions. Chapters 13 - 16 contain sample code that depicts use of the various x86-AVX computational resources. Examples include using the x86-AVX instruction set with packed integers, packed floating-point, and scalar floating-point operands. These chapters also contain sample code that explicates use of the data broadcast, gather, permute, and FMA instructions.

X86-64 Core Architecture—Chapter 17 peruses the x86-64 platform and includes a discussion of the platform's core architecture, supported data types, general purpose registers, and status flags. It also explains the enhancements made to the x86-32 platform in order to support 64-bit operands and memory addressing. The chapter concludes with a discussion of the x86-64 instruction set, including those instructions that have been deprecated or are no longer available. Chapter 18 explores the fundamentals x86-64 assembly language programming using a variety of sample code. Examples include how to perform integer calculations using operands of various sizes, memory addressing modes, scalar floating-point arithmetic, and common programming constructs. Chapter 18 also explains the calling convention that must be observed in order to invoke an x86-64 assembly language function from C++.

X86-64 SSE and AVX—Chapter 19 describes the enhancements to x86-SSE and x86-AVX that are available on the x86-64 platform. This includes a discussion of the respective execution environments and extended data register sets. Chapter 20 contains sample code that highlights use of the x86-SSE and x86-AVX instruction sets with the x86-64 core architecture.

Advanced Topics—The last two chapters of this book consider advanced topics and optimization techniques related to x86 assembly language programming. Chapter 21 examines key elements of an x86 processor's microarchitecture, including its front-end pipelines, out-of-order execution model, and internal execution units. It also includes a discussion of programming techniques that you can employ to write x86 assembly

language code that is both spatially and temporally efficient. Chapter 22 contains sample code that illustrates several advanced assembly language programming techniques.

Appendices—The final section of the book includes several appendices. Appendix A contains a brief tutorial on how to use Microsoft's Visual C++ and Macro Assembler. Appendix B summarizes the x86-32 and x86-64 calling conventions that assembly language functions must observe in order to be invoked from a Visual C++ function. Appendix C contains a list of references and resources that you can consult for more information about x86 assembly language programming.

Sample Code Requirements

You can download the sample code for this book from the Apress website at http://www.apress.com/9781484200650. The following hardware and software is required to build and run the sample code:

- A PC with an x86 processor that is based on a recent microarchitecture. All of the x86-32, x87 FPU, MMX, and x86-SSE sample code can be executed using a processor based on the Nehalem (or later) microarchitecture. PCs with processors based on earlier microarchitectures also can be used to run many of the sample code programs. The AVX and AXV2 sample code requires a processor based on the Sandy Bridge or Haswell microarchitecture, respectively.
- Microsoft Windows 8.x or Windows 7 with Service Pack 1. A 64-bit version of Windows is required to run the x86-64 sample code.
- Visual Studio Professional 2013 or Visual Studio Express 2013 for Windows Desktop. The Express edition can be freely downloaded from the following Microsoft website: http://msdn.microsoft.com/en-us/vstudio. Update 3 is recommended for both Visual Studio editions.

Caution The primary purpose of the sample code is to elucidate the topics and technologies presented in this book. Minimal attention is given to important software engineering concerns such as robust error handling, security risks, numerical stability, rounding errors, or ill-conditioned functions. You are responsible for addressing these issues should you decide to use any of the sample code in your own programs.

Terminology and Conventions

The following paragraphs define the meaning of common terms and expressions used throughout this book. A *function, subroutine,* or *procedure* is a self-contained unit of executable code that accepts zero or more arguments, performs an operation, and optionally returns a value. Functions are typically invoked using the processor's call instruction. A *thread* is the smallest unit of execution that is managed and scheduled by an operating system. A *task* or *process* is a collection of one or more threads that share the same logical memory space. An *application* or *program* is a complete software package that contains at least one task.

The terms *x86-32* and *x86-64* are used respectively to describe 32-bit and 64-bit aspects, resources, or capabilities of a processor; *x86* is employed for features that are common to both 32-bit and 64-bit architectures. The expressions *x86-32 mode* and *x86-64 mode* denote a specific processor execution environment with the primary difference being the latter mode's support of 64-bit registers, operands, and memory addressing. Common capabilities of the x86's SIMD extensions are described using the terms *x86-SSE* for Streaming SIMD Extensions or *x86-AVX* for Advanced Vector Extensions. When discussing aspects or instructions of a specific SIMD enhancement, the original acronyms (e.g., SSE, SSE2, SSE3, SSSE3, SSE4, AVX, and AVX2) are used.

Additional Resources

An extensive set of x86-related documentation is available from both Intel and AMD. Appendix C lists a number of resources that both aspiring and experienced x86 assembly language programmers will find useful. Of all the resources listed Appendix C, the most important tome is Volume 2 of the reference manual entitled *Intel 64 and IA-32 Architectures Software Developer's Manual—Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C (Order Number: 325462)*. This volume contains comprehensive information for each processor instruction, including detailed operational descriptions, lists of valid operands, affected status flags, and potential exceptions. You are strongly encouraged to consult this documentation when developing your own x86 assembly language functions in order to verify correct instruction usage.