Preface

Assembly Language for x86 Processors, Sixth Edition, teaches assembly language programming and architecture for Intel and AMD processors. It is an appropriate text for the following types of college courses:

- Assembly Language Programming
- Fundamentals of Computer Systems
- Fundamentals of Computer Architecture

Students use Intel or AMD processors and program with **Microsoft Macro Assembler (MASM),** running on Windows 98, XP, Vista, and Windows 7. Although this book was originally designed as a programming textbook for college students, it serves as an effective supplement to computer architecture courses. As a testament to its popularity, previous editions have been translated into Spanish, Korean, Chinese, French, Russian, and Polish.

Emphasis of Topics This edition includes topics that lead naturally into subsequent courses in computer architecture, operating systems, and compiler writing:

- Virtual machine concept
- Instruction set architecture
- Elementary Boolean operations
- Instruction execution cycle
- Memory access and handshaking
- Interrupts and polling
- Hardware-based I/O
- Floating-point binary representation

Other topics relate specially to Intel and AMD architecture:

- Protected memory and paging
- Memory segmentation in real-address mode
- 16-bit interrupt handling
- MS-DOS and BIOS system calls (interrupts)
- Floating-point unit architecture and programming
- Instruction encoding

Certain examples presented in the book lend themselves to courses that occur later in a computer science curriculum:

- Searching and sorting algorithms
- High-level language structures

- Finite-state machines
- Code optimization examples

What's New in the Sixth Edition

In this revision, we have placed a strong emphasis on improving the descriptions of important programming concepts and relevant program examples.

- We have added numerous step-by-step descriptions of sample programs, particularly in Chapters 1–8.
- Many new illustrations have been inserted into the chapters to improve student comprehension of concepts and details.
- **Java Bytecodes:** The Java Virtual Machine (JVM) provides an excellent real-life example of a stack-oriented architecture. It provides an excellent contrast to x86 architecture. Therefore, in Chapters 8 and 9, the author explains the basic operation of Java bytecodes with short illustrative examples. Numerous short examples are shown in disassembled bytecode format, followed by detailed step-by-step explanations.
- Selected programming exercises have been replaced in the first 8 chapters. Programming exercises are now assigned stars to indicate their difficulty. One star is the easiest, four stars indicate the most difficult level.
- Tutorial videos by the author are available on the Companion Web site (www.pearsonhighered.com/irvine) to explain worked-out programming exercises.
- The order of chapters in the second half of the book has been revised to form a more logical sequence of topics, and selected chapters are supplied in electronic form for easy searching.

This book is still focused on its primary goal, to teach students how to write and debug programs at the machine level. It will never replace a complete book on computer architecture, but it does give students the first-hand experience of writing software in an environment that teaches them how a computer works. Our premise is that students retain knowledge better when theory is combined with experience. In an engineering course, students construct prototypes; in a computer architecture course, students should write machine-level programs. In both cases, they have a memorable experience that gives them the confidence to work in any OS/machine-oriented environment.

Real Mode and Protected Mode This edition emphasizes 32-bit protected mode, but it still has three electronic chapters devoted to real-mode programming. For example, there is an entire chapter on BIOS programming for the keyboard, video display (including graphics), and mouse. Another chapter covers MS-DOS programming using interrupts (system calls). Students can benefit from programming directly to hardware and the BIOS.

The examples in the first half of the book are nearly all presented as 32-bit text-oriented applications running in protected mode using the flat memory model. This approach is wonderfully simple because it avoids the complications of segment-offset addressing. Specially marked paragraphs and popup boxes point out occasional differences between protected mode and real-mode programming. Most differences are abstracted by the book's parallel link libraries for real-mode and protected mode programming.

Link Libraries We supply two versions of the link library that students use for basic input-output, simulations, timing, and other useful stuff. The 32-bit version (Irvine32.lib) runs in protected mode, sending its output to the Win32 console. The 16-bit version (Irvine16.lib) runs in real-address mode. Full source code for the libraries is supplied on the Companion Web site. The link libraries are available only for convenience, not to prevent students from learning how to program input-output themselves. Students are encouraged to create their own libraries.

Included Software and Examples All the example programs were tested with Microsoft Macro Assembler Version 10.0, running in Microsoft Visual Studio 2010. In addition, batch files are supplied that permit students to assemble and run applications from the Windows command prompt. The 32-bit C++ applications in Chapter 14 were tested with Microsoft Visual C++ .NET.

Web Site Information Updates and corrections to this book may be found at the Companion Web site, including additional programming projects for instructors to assign at the ends of chapters.

Overall Goals

The following goals of this book are designed to broaden the student's interest and knowledge in topics related to assembly language:

- Intel and AMD processor architecture and programming
- Real-address mode and protected mode programming
- Assembly language directives, macros, operators, and program structure
- Programming methodology, showing how to use assembly language to create system-level software tools and application programs
- Computer hardware manipulation
- Interaction between assembly language programs, the operating system, and other application programs

One of our goals is to help students approach programming problems with a machine-level mind set. It is important to think of the CPU as an interactive tool, and to learn to monitor its operation as directly as possible. A debugger is a programmer's best friend, not only for catching errors, but as an educational tool that teaches about the CPU and operating system. We encourage students to look beneath the surface of high-level languages and to realize that most programming languages are designed to be portable and, therefore, independent of their host machines. In addition to the short examples, this book contains hundreds of ready-to-run programs that demonstrate instructions or ideas as they are presented in the text. Reference materials, such as guides to MS-DOS interrupts and instruction mnemonics, are available at the end of the book.

Required Background The reader should already be able to program confidently in at least one high-level programming language such as Python, Java, C, or C++. One chapter covers C++ interfacing, so it is very helpful to have a compiler on hand. I have used this book in the classroom with majors in both computer science and management information systems, and it has been used elsewhere in engineering courses.

Features

Complete Program Listings The Companion Web site contains supplemental learning materials, study guides, and all the source code from the book's examples. An extensive link library

is supplied with the book, containing more than 30 procedures that simplify user input-output, numeric processing, disk and file handling, and string handling. In the beginning stages of the course, students can use this library to enhance their programs. Later, they can create their own procedures and add them to the library.

Programming Logic Two chapters emphasize Boolean logic and bit-level manipulation. A conscious attempt is made to relate high-level programming logic to the low-level details of the machine. This approach helps students to create more efficient implementations and to better understand how compilers generate object code.

Hardware and Operating System Concepts The first two chapters introduce basic hardware and data representation concepts, including binary numbers, CPU architecture, status flags, and memory mapping. A survey of the computer's hardware and a historical perspective of the Intel processor family helps students to better understand their target computer system.

Structured Programming Approach Beginning with Chapter 5, procedures and functional decomposition are emphasized. Students are given more complex programming exercises, requiring them to focus on design before starting to write code.

Java Bytecodes and the Java Virtual Machine In Chapters 8 and 9, the author explains the basic operation of Java bytecodes with short illustrative examples. Numerous short examples are shown in disassembled bytecode format, followed by detailed step-by-step explanations.

Disk Storage Concepts Students learn the fundamental principles behind the disk storage system on MS-Windows–based systems from hardware and software points of view.

Creating Link Libraries Students are free to add their own procedures to the book's link library and create new libraries. They learn to use a toolbox approach to programming and to write code that is useful in more than one program.

Macros and Structures A chapter is devoted to creating structures, unions, and macros, which are essential in assembly language and systems programming. Conditional macros with advanced operators serve to make the macros more professional.

Interfacing to High-Level Languages A chapter is devoted to interfacing assembly language to C and C++. This is an important job skill for students who are likely to find jobs programming in high-level languages. They can learn to optimize their code and see examples of how C++ compilers optimize code.

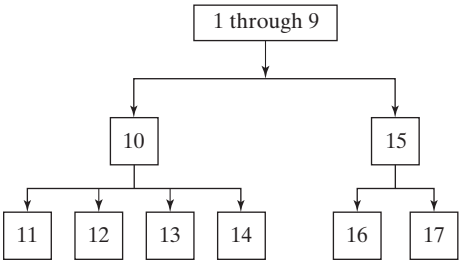
Instructional Aids All the program listings are available on the Web. Instructors are provided a test bank, answers to review questions, solutions to programming exercises, and a Microsoft PowerPoint slide presentation for each chapter.

VideoNotes VideoNotes are Pearson's new visual tool designed to teach students key programming concepts and techniques. These short step-by-step videos demonstrate how to solve problems from design through coding. VideoNotes allow for self-paced instruction with easy navigation including the ability to select, play, rewind, fast-forward, and stop within each VideoNote exercise. A note appears within the text to designate that a VideoNote is available.

VideoNotes are free with the purchase of a new textbook. To *purchase* access to VideoNotes, go to www.pearsonhighered.com/irvine and click on the VideoNotes under *Student Resources*.

Chapter Descriptions

Chapters 1 to 8 contain core concepts of assembly language and should be covered in sequence. After that, you have a fair amount of freedom. The following chapter dependency graph shows how later chapters depend on knowledge gained from other chapters.



1. **Basic Concepts:** Applications of assembly language, basic concepts, machine language, and data representation.
2. **x86 Processor Architecture:** Basic microcomputer design, instruction execution cycle, x86 processor architecture, x86 memory management, components of a microcomputer, and the input-output system.
3. **Assembly Language Fundamentals:** Introduction to assembly language, linking and debugging, and defining constants and variables.
4. **Data Transfers, Addressing, and Arithmetic:** Simple data transfer and arithmetic instructions, assemble-link-execute cycle, operators, directives, expressions, JMP and LOOP instructions, and indirect addressing.
5. **Procedures:** Linking to an external library, description of the book's link library, stack operations, defining and using procedures, flowcharts, and top-down structured design.
6. **Conditional Processing:** Boolean and comparison instructions, conditional jumps and loops, high-level logic structures, and finite-state machines.
7. **Integer Arithmetic:** Shift and rotate instructions with useful applications, multiplication and division, extended addition and subtraction, and ASCII and packed decimal arithmetic.
8. **Advanced Procedures:** Stack parameters, local variables, advanced PROC and INVOKE directives, and recursion.
9. **Strings and Arrays:** String primitives, manipulating arrays of characters and integers, two-dimensional arrays, sorting, and searching.
10. **Structures and Macros:** Structures, macros, conditional assembly directives, and defining repeat blocks.
11. **MS-Windows Programming:** Protected mode memory management concepts, using the Microsoft-Windows API to display text and colors, and dynamic memory allocation.
12. **Floating-Point Processing and Instruction Encoding:** Floating-point binary representation and floating-point arithmetic. Learning to program the IA-32 floating-point unit. Understanding the encoding of IA-32 machine instructions.

13. **High-Level Language Interface:** Parameter passing conventions, inline assembly code, and linking assembly language modules to C and C++ programs.
14. **16-Bit MS-DOS Programming:** Calling MS-DOS interrupts for console and file input-output.

- **Appendix A:** MASM Reference
- **Appendix B:** The x86 Instruction Set
- **Appendix C:** Answers to Review Questions

The following chapters and appendices are supplied online at the Companion Web site:

15. **Disk Fundamentals:** Disk storage systems, sectors, clusters, directories, file allocation tables, handling MS-DOS error codes, and drive and directory manipulation.
16. **BIOS-Level Programming:** Keyboard input, video text, graphics, and mouse programming.
17. **Expert MS-DOS Programming:** Custom-designed segments, runtime program structure, and Interrupt handling. Hardware control using I/O ports.

- **Appendix D:** BIOS and MS-DOS Interrupts
- **Appendix E:** Answers to Review Questions (Chapters 15–17)

Instructor and Student Resources

Instructor Resource Materials

The following protected instructor material is available on the Companion Web site:

www.pearsonhighered.com/irvine

For username and password information, please contact your Pearson Representative.

- Lecture PowerPoint Slides
- Instructor Solutions Manual

Student Resource Materials

The student resource materials can be accessed through the publisher's Web site located at www.pearsonhighered.com/irvine. These resources include:

- VideoNotes
- Online Chapters and Appendices
  - Chapter 15: *Disk Fundamentals*
  - Chapter 16: *BIOS-Level Programming*
  - Chapter 17: *Expert MS-DOS Programming*
  - Appendix D: *BIOS and MS-DOS Interrupts*
  - Appendix E: *Answers to Review Questions (Chapters 15–17)*

Students must use the access card located in the front of the book to register and access the online chapters and VideoNotes. If there is no access card in the front of this textbook, students can purchase access by going to www.pearsonhighered.com/irvine and selecting "purchase access to premium content." Instructors must also register on the site to access this material. Students will also find a link to the author's Web site. An access card is not required for the following materials:

- *Getting Started*, a comprehensive step-by-step tutorial that helps students customize Visual Studio for assembly language programming.
- Supplementary articles on assembly language programming topics.

- Complete source code for all example programs in the book, as well as the source code for the author's supplementary library.
- *Assembly Language Workbook*, an interactive workbook covering number conversions, addressing modes, register usage, debug programming, and floating-point binary numbers. Content pages are HTML documents to allow for customization. Help File in Windows Help Format.
- Debugging Tools: Tutorials on using Microsoft CodeView, MS-DOS Debug, and Microsoft Visual Studio.

Acknowledgments

Many thanks are due to Tracy Dunkelberger, Executive Editor for Computer Science at Pearson Education, who has provided friendly, helpful guidance over the past few years. Maheswari Pon-Saravanan of TexTech International did an excellent job on the book production, along with Jane Bonnell as the production editor at Pearson. Many thanks to Scott Disanno, the book's managing editor, and Melinda Haggerty, the assistant editor.

Sixth Edition

Many thanks are due to Professor James Brink of Pacific Lutheran University, Professor David Topham of Ohlone College, and Professor W. A. Barrett of San Jose State University. All have contributed excellent code examples and debugging suggestions to this book. In addition, I give grateful acknowledgment to the reviewers of the Sixth edition:

- Hisham Al-Mubaid, University of Houston, Clearlake
- John-Thones Amenyo, York College of CUNY
- John F. Doyle, Indiana University, Southeast
- Nicole Jiao, South Texas Community College
- Remzi Seker, University of Arkansas, Little Rock

Previous Editions

I offer my special thanks to the following individuals who were most helpful during the development of earlier editions of this book:

- William Barrett, San Jose State University
- Scott Blackledge
- James Brink, Pacific Lutheran University
- Gerald Cahill, Antelope Valley College
- John Taylor