Assembly Language for x86 Processors

Eighth Edition

KIP R. IRVINE
Florida International University
School of Computing and Information Sciences

Pearson
To Jack and Candy Irvine
CONTENTS

Preface xxi

1 Basic Concepts 1
1.1 Welcome to Assembly Language 1
  1.1.1 Questions You Might Ask 2
  1.1.2 Assembly Language Applications 5
  1.1.3 Section Review 6

1.2 Virtual Machine Concept 7
  1.2.1 Section Review 9

1.3 Data Representation 10
  1.3.1 Binary Integers 10
  1.3.2 Binary Addition 12
  1.3.3 Integer Storage Sizes 13
  1.3.4 Hexadecimal Integers 14
  1.3.5 Hexadecimal Addition 16
  1.3.6 Signed Binary Integers 16
  1.3.7 Binary Subtraction 19
  1.3.8 Character Storage 20
  1.3.9 Binary-Coded Decimal (BCD) Numbers 22
  1.3.10 Section Review 22

1.4 Boolean Expressions 23
  1.4.1 Truth Tables for Boolean Functions 26
  1.4.2 Section Review 27

1.5 Chapter Summary 27

1.6 Key Terms 28

1.7 Review Questions and Exercises 29
  1.7.1 Short Answer 29
  1.7.2 Algorithm Workbench 31

2 x86 Processor Architecture 33
2.1 General Concepts 34
  2.1.1 Basic Microcomputer Design 34
  2.1.2 Instruction Execution Cycle 35
  2.1.3 Reading from Memory 36
2.1.4 Loading and Executing a Program 37
2.1.5 Section Review 38
2.2 32-Bit x86 Processors 39
  2.2.1 Modes of Operation 39
  2.2.2 Basic Execution Environment 39
  2.2.3 x86 Memory Management 43
  2.2.4 Section Review 44
2.3 64-Bit x86-64 Processors 44
  2.3.1 64-Bit Operation Modes 45
  2.3.2 Basic 64-Bit Execution Environment 45
  2.3.3 Section Review 46
2.4 Components of a Typical x86 Computer 47
  2.4.1 Motherboard 47
  2.4.2 Memory 49
  2.4.3 Section Review 49
2.5 Input-Output System 50
  2.5.1 Levels of I/O Access 50
  2.5.2 Section Review 52
2.6 Chapter Summary 53
2.7 Key Terms 54
2.8 Review Questions 55

3 Assembly Language Fundamentals 57
3.1 Basic Language Elements 58
  3.1.1 First Assembly Language Program 58
  3.1.2 Integer Literals 59
  3.1.3 Constant Integer Expressions 60
  3.1.4 Real Number Literals 61
  3.1.5 Character Literals 61
  3.1.6 String Literals 62
  3.1.7 Reserved Words 62
  3.1.8 Identifiers 62
  3.1.9 Directives 63
  3.1.10 Instructions 63
  3.1.11 Section Review 67
3.2 Example: Adding and Subtracting Integers 67
  3.2.1 The AddTwo Program 67
  3.2.2 Running and Debugging the AddTwo Program 69
  3.2.3 Program Template 73
  3.2.4 Section Review 74
3.3 Assembling, Linking, and Running Programs 75
3.3.1 The Assemble-Link-Execute Cycle 75
3.3.2 Listing File 76
3.3.3 Section Review 78

3.4 Defining Data 78
3.4.1 Intrinsic Data Types 78
3.4.2 Data Definition Statement 79
3.4.3 Adding a Variable to the AddTwo Program 80
3.4.4 Defining BYTE and SBYTE Data 81
3.4.5 Defining WORD and SWORD Data 83
3.4.6 Defining DWORD and SDWORD Data 84
3.4.7 Defining QWORD Data 84
3.4.8 Defining Packed BCD (TBYTE) Data 85
3.4.9 Defining Floating-Point Types 85
3.4.10 A Program That Adds Variables 86
3.4.11 Little-Endian Order 87
3.4.12 Declaring Uninitialized Data 88
3.4.13 Section Review 88

3.5 Symbolic Constants 90
3.5.1 Equal-Sign Directive 90
3.5.2 Calculating the Sizes of Arrays and Strings 91
3.5.3 EQU Directive 92
3.5.4 TEXTEQU Directive 93
3.5.5 Section Review 94

3.6 Introducing 64-Bit Programming 95

3.7 Chapter Summary 96

3.8 Key Terms 98
3.8.1 Terms 98
3.8.2 Instructions, Operators, and Directives 98

3.9 Review Questions and Exercises 99
3.9.1 Short Answer 99
3.9.2 Algorithm Workbench 100

3.10 Programming Exercises 100

4 Data Transfers, Addressing, and Arithmetic 102
4.1 Data Transfer Instructions 103
4.1.1 Introduction 103
4.1.2 Operand Types 103
4.1.3 Direct Memory Operands 103
4.1.4 MOV Instruction 105
4.1.5 Zero/Sign Extension of Integers 106
4.1.6 LAHF and SAHF Instructions 108
4.1.7 XCHG Instruction 109
4.1.8 Direct-Offset Operands 109
4.1.9 Examples of Moving Data 110
4.1.10 Section Review 113

4.2 Addition and Subtraction 114
4.2.1 INC and DEC Instructions 114
4.2.2 ADD Instruction 114
4.2.3 SUB Instruction 115
4.2.4 NEG Instruction 115
4.2.5 Implementing Arithmetic Expressions 115
4.2.6 Flags Affected by Addition and Subtraction 116
4.2.7 Example Program (AddSubText) 120
4.2.8 Section Review 121

4.3 Data-Related Operators and Directives 122
4.3.1 OFFSET Operator 122
4.3.2 ALIGN Directive 123
4.3.3 PTR Operator 124
4.3.4 TYPE Operator 125
4.3.5 LENGTHOF Operator 125
4.3.6 SIZEOF Operator 126
4.3.7 LABEL Directive 126
4.3.8 Section Review 127

4.4 Indirect Addressing 127
4.4.1 Indirect Operands 127
4.4.2 Arrays 128
4.4.3 Indexed Operands 129
4.4.4 Pointers 131
4.4.5 Section Review 132

4.5 JMP and LOOP Instructions 134
4.5.1 JMP Instruction 134
4.5.2 LOOP Instruction 135
4.5.3 Displaying an Array in the Visual Studio Debugger 136
4.5.4 Summing an Integer Array 137
4.5.5 Copying a String 138
4.5.6 Section Review 139

4.6 64-Bit Programming 140
4.6.1 MOV Instruction 140
4.6.2 64-Bit Version of SumArray 141
4.6.3 Addition and Subtraction 142
4.6.4 Section Review 143
5.7 Key Terms 199
  5.7.1 Terms 199
  5.7.2 Instructions, Operators, and Directives 199

5.8 Review Questions and Exercises 199
  5.8.1 Short Answer 199
  5.8.2 Algorithm Workbench 202

5.9 Programming Exercises 203

6 Conditional Processing 205

6.1 Boolean and Comparison Instructions 206
  6.1.1 The CPU Status Flags 206
  6.1.2 AND Instruction 207
  6.1.3 OR Instruction 208
  6.1.4 Bit-Mapped Sets 209
  6.1.5 XOR Instruction 211
  6.1.6 NOT Instruction 212
  6.1.7 TEST Instruction 212
  6.1.8 CMP Instruction 213
  6.1.9 Setting and Clearing Individual CPU Flags 214
  6.1.10 Boolean Instructions in 64-Bit Mode 214
  6.1.11 Section Review 215

6.2 Conditional Jumps 216
  6.2.1 Conditional Structures 216
  6.2.2 Jcond Instruction 217
  6.2.3 Types of Conditional Jump Instructions 217
  6.2.4 Conditional Jump Applications 221
  6.2.5 Section Review 225

6.3 Conditional Loop Instructions 226
  6.3.1 LOOPZ and LOOPE Instructions 226
  6.3.2 LOOPNZ and LOOPNE Instructions 227
  6.3.3 Section Review 227

6.4 Conditional Structures 228
  6.4.1 Block-Structured IF Statements 228
  6.4.2 Compound Expressions 233
  6.4.3 WHILE Loops 234
  6.4.4 Table-Driven Selection 237
  6.4.5 Section Review 239

6.5 Application: Finite-State Machines 240
  6.5.1 Validating an Input String 240
  6.5.2 Validating a Signed Integer 241
  6.5.3 Section Review 245
6.6 Conditional Control Flow Directives (Optional topic) 246
   6.6.1 Creating IF Statements 246
   6.6.2 Signed and Unsigned Comparisons 249
   6.6.3 Compound Expressions 250
   6.6.4 Creating Loops with .REPEAT and .WHILE 253

6.7 Chapter Summary 254

6.8 Key Terms 255
   6.8.1 Terms 255
   6.8.2 Instructions, Operators, and Directives 255

6.9 Review Questions and Exercises 256
   6.9.1 Short Answer 256
   6.9.2 Algorithm Workbench 258

6.10 Programming Exercises 259
   6.10.1 Suggestions for Testing Your Code 259
   6.10.2 Exercise Descriptions 260

7 Integer Arithmetic 263

7.1 Shift and Rotate Instructions 264
   7.1.1 Logical Shifts and Arithmetic Shifts 264
   7.1.2 SHL Instruction 265
   7.1.3 SHR Instruction 266
   7.1.4 SAL and SAR Instructions 267
   7.1.5 ROL Instruction 268
   7.1.6 ROR Instruction 269
   7.1.7 RCL and RCR Instructions 269
   7.1.8 Signed Overflow 270
   7.1.9 SHLD/SHRD Instructions 270
   7.1.10 Section Review 272

7.2 Shift and Rotate Applications 274
   7.2.1 Shifting Multiple Doublewords 274
   7.2.2 Multiplication by Shifting Bits 275
   7.2.3 Displaying Binary Bits 276
   7.2.4 Extracting File Date Fields 276
   7.2.5 Section Review 277

7.3 Multiplication and Division Instructions 279
   7.3.1 Unsigned Integer Multiplication (MUL) 279
   7.3.2 Signed Integer Multiplication (IMUL) 281
   7.3.3 Measuring Program Execution Times 284
   7.3.4 Unsigned Integer Division (DIV) 286
   7.3.5 Signed Integer Division (IDIV) 288
# Contents

8.3 **Recursion** 331  
  8.3.1 Recursively Calculating a Sum 331  
  8.3.2 Calculating a Factorial 333  
  8.3.3 Section Review 339  

8.4 **INVOKE, ADDR, PROC, and PROTO** 340  
  8.4.1 **INVOKE** Directive 340  
  8.4.2 ADDR Operator 341  
  8.4.3 PROC Directive 342  
  8.4.4 PROTO Directive 345  
  8.4.5 Parameter Classifications 348  
  8.4.6 Example: Exchanging Two Integers 349  
  8.4.7 Debugging Tips 350  
  8.4.8 WriteStackFrame Procedure 351  
  8.4.9 Section Review 352  

8.5 **Creating Multimodule Programs** 352  
  8.5.1 Hiding and Exporting Procedure Names 352  
  8.5.2 Calling External Procedures 353  
  8.5.3 Using Variables and Symbols across Module Boundaries 354  
  8.5.4 Example: ArraySum Program 355  
  8.5.5 Creating the Modules Using Extern 356  
  8.5.6 Creating the Modules Using **INVOKE** and **PROTO** 359  
  8.5.7 Section Review 362  

8.6 **Advanced Use of Parameters (Optional Topic)** 363  
  8.6.1 Stack Affected by the **USES** Operator 363  
  8.6.2 Passing 8-Bit and 16-Bit Arguments on the Stack 364  
  8.6.3 Passing 64-Bit Arguments 366  
  8.6.4 Non-Doubleword Local Variables 366  

8.7 **Java Bytecodes (Optional Topic)** 369  
  8.7.1 Java Virtual Machine 369  
  8.7.2 Instruction Set 370  
  8.7.3 Java Disassembly Examples 371  
  8.7.4 Example: Conditional Branch 374  

8.8 **Chapter Summary** 376  

8.9 **Key Terms** 377  
  8.9.1 Terms 377  
  8.9.2 Instructions, Operators, and Directives 377  

8.10 **Review Questions and Exercises** 377  
  8.10.1 Short Answer 377  
  8.10.2 Algorithm Workbench 378  

8.11 **Programming Exercises** 378
9 Strings and Arrays 381

9.1 Introduction 381

9.2 String Primitive Instructions 382
  9.2.1 MOVSB, MOVSW, and MOVSD 383
  9.2.2 CMPSB, CMPSW, and CMPSD 384
  9.2.3 SCASB, SCASW, and SCASD 385
  9.2.4 STOSB, STOSW, and STOSD 385
  9.2.5 LODSB, LODSW, and LODSD 385
  9.2.6 Section Review 386

9.3 Selected String Procedures 387
  9.3.1 Str_compare Procedure 388
  9.3.2 Str_length Procedure 389
  9.3.3 Str_copy Procedure 389
  9.3.4 Str_trim Procedure 390
  9.3.5 Str_ucase Procedure 393
  9.3.6 String Library Demo Program 393
  9.3.7 String Procedures in the Irvine64 Library 395
  9.3.8 Section Review 398

9.4 Two-Dimensional Arrays 399
  9.4.1 Ordering of Rows and Columns 399
  9.4.2 Base-Index Operands 399
  9.4.3 Base-Index-Displacement Operands 401
  9.4.4 Base-Index Operands in 64-Bit Mode 402
  9.4.5 Section Review 403

9.5 Searching and Sorting Integer Arrays 404
  9.5.1 Bubble Sort 404
  9.5.2 Binary Search 406
  9.5.3 Section Review 412

9.6 Java Bytecodes: String Processing (Optional Topic) 413

9.7 Chapter Summary 414

9.8 Key Terms and Instructions 415

9.9 Review Questions and Exercises 415
  9.9.1 Short Answer 415
  9.9.2 Algorithm Workbench 416

9.10 Programming Exercises 416
10 Structures and Macros  421

10.1 Structures  421
  10.1.1 Defining Structures  422
  10.1.2 Declaring Structure Objects  424
  10.1.3 Referencing Structure Objects  425
  10.1.4 Example: Displaying the System Time  428
  10.1.5 Structures Containing Structures  430
  10.1.6 Example: Drunkard’s Walk  430
  10.1.7 Declaring and Using Unions  434
  10.1.8 Section Review  436

10.2 Macros  437
  10.2.1 Overview  437
  10.2.2 Defining Macros  438
  10.2.3 Invoking Macros  439
  10.2.4 Additional Macro Features  440
  10.2.5 Using Our Macro Library (32-Bit Mode Only)  444
  10.2.6 Example Program: Wrappers  451
  10.2.7 Section Review  452

10.3 Conditional-Assembly Directives  453
  10.3.1 Checking for Missing Arguments  454
  10.3.2 Default Argument Initializers  455
  10.3.3 Boolean Expressions  456
  10.3.4 IF, ELSE, and ENDF Directives  456
  10.3.5 The IFIDN and IFIDNI Directives  457
  10.3.6 Example: Summing a Matrix Row  458
  10.3.7 Special Operators  461
  10.3.8 Macro Functions  464
  10.3.9 Section Review  466

10.4 Defining Repeat Blocks  467
  10.4.1 WHILE Directive  467
  10.4.2 REPEAT Directive  468
  10.4.3 FOR Directive  468
  10.4.4 FORC Directive  469
  10.4.5 Example: Linked List  470
  10.4.6 Section Review  471

10.5 Chapter Summary  473

10.6 Key Terms  474
  10.6.1 Terms  474
  10.6.2 Operators and Directives  474
10.7 Review Questions and Exercises 475
  10.7.1 Short Answer 475
  10.7.2 Algorithm Workbench 475

10.8 Programming Exercises 477

11 MS-Windows Programming 480
11.1 Win32 Console Programming 480
  11.1.1 Background Information 481
  11.1.2 Win32 Console Functions 485
  11.1.3 Displaying a Message Box 487
  11.1.4 Console Input 490
  11.1.5 Console Output 496
  11.1.6 Reading and Writing Files 498
  11.1.7 File I/O in the Irvine32 Library 502
  11.1.8 Testing the File I/O Procedures 504
  11.1.9 Console Window Manipulation 507
  11.1.10 Controlling the Cursor 510
  11.1.11 Controlling the Text Color 511
  11.1.12 Time and Date Functions 513
  11.1.13 Using the 64-Bit Windows API 517
  11.1.14 Section Review 518

11.2 Writing a Graphical Windows Application 519
  11.2.1 Necessary Structures 519
  11.2.2 The MessageBox Function 521
  11.2.3 The WinMain Procedure 521
  11.2.4 The WinProc Procedure 522
  11.2.5 The ErrorHandler Procedure 523
  11.2.6 Program Listing 523
  11.2.7 Section Review 527

11.3 Dynamic Memory Allocation 528
  11.3.1 HeapTest Programs 531
  11.3.2 Section Review 535

11.4 32-bit x86 Memory Management 535
  11.4.1 Linear Addresses 536
  11.4.2 Page Translation 539
  11.4.3 Section Review 541

11.5 Chapter Summary 541

11.6 Key Terms 543

11.7 Review Questions and Exercises 543
  11.7.1 Short Answer 543
  11.7.2 Algorithm Workbench 544

11.8 Programming Exercises 544
12 Floating-Point Processing and Instruction Encoding 547

12.1 Floating-Point Binary Representation 547
  12.1.1 IEEE Binary Floating-Point Representation 548
  12.1.2 The Exponent 550
  12.1.3 Normalized Binary Floating-Point Numbers 550
  12.1.4 Creating the IEEE Representation 550
  12.1.5 Converting Decimal Fractions to Binary Reals 552
  12.1.6 Section Review 554

12.2 Floating-Point Unit 555
  12.2.1 FPU Register Stack 555
  12.2.2 Rounding 558
  12.2.3 Floating-Point Exceptions 560
  12.2.4 Floating-Point Instruction Set 560
  12.2.5 Arithmetic Instructions 563
  12.2.6 Comparing Floating-Point Values 567
  12.2.7 Reading and Writing Floating-Point Values 570
  12.2.8 Exception Synchronization 571
  12.2.9 Code Examples 572
  12.2.10 Mixed-Mode Arithmetic 574
  12.2.11 Masking and Unmasking Exceptions 575
  12.2.12 Section Review 576

12.3 x86 Instruction Encoding 577
  12.3.1 Instruction Format 577
  12.3.2 Single-Byte Instructions 578
  12.3.3 Move Immediate to Register 579
  12.3.4 Register-Mode Instructions 580
  12.3.5 Processor Operand-Size Prefix 581
  12.3.6 Memory-Mode Instructions 582
  12.3.7 Section Review 585

12.4 Chapter Summary 585

12.5 Key Terms 587

12.6 Review Questions and Exercises 587
  12.6.1 Short Answer 587
  12.6.2 Algorithm Workbench 588

12.7 Programming Exercises 589
13 High-Level Language Interface  593

13.1 Introduction  593
  13.1.1 General Conventions  593
  13.1.2 .MODEL Directive  595
  13.1.3 Examining Compiler-Generated Code  597
  13.1.4 Section Review  602

13.2 Inline Assembly Code  602
  13.2.1 __asm Directive in Visual C++  602
  13.2.2 File Encryption Example  605
  13.2.3 Section Review  608

13.3 Linking 32-Bit Assembly Language Code to C/C++  609
  13.3.1 IndexOf Example  610
  13.3.2 Calling C and C++ Functions  613
  13.3.3 Multiplication Table Example  615
  13.3.4 Section Review  618

13.4 Chapter Summary  619

13.5 Key Terms  620

13.6 Review Questions  620

13.7 Programming Exercises  620

14 16-Bit MS-DOS Programming  622

14.1 MS-DOS and the IBM-PC  622
  14.1.1 Memory Organization  623
  14.1.2 Redirecting Input-Output  624
  14.1.3 Software Interrupts  625
  14.1.4 INT Instruction  626
  14.1.5 Coding for 16-Bit Programs  627
  14.1.6 Section Review  628

14.2 MS-DOS Function Calls (INT 21h)  628
  14.2.1 Selected Output Functions  630
  14.2.2 Hello World Program Example  632
  14.2.3 Selected Input Functions  633
  14.2.4 Date/Time Functions  637
  14.2.5 Section Review  641

14.3 Standard MS-DOS File I/O Services  641
  14.3.1 Create or Open File (716Ch)  643
  14.3.2 Close File Handle (3Eh)  644
  14.3.3 Move File Pointer (42h)  644
  14.3.4 Get File Creation Date and Time  645
16 BIOS-Level Programming 689

16.1 Introduction 689
  16.1.1 BIOS Data Area 690

16.2 Keyboard Input with INT 16h 691
  16.2.1 How the Keyboard Works 691
  16.2.2 INT 16h Functions 692
  16.2.3 Section Review 696

16.3 Video Programming with INT 10h 697
  16.3.1 Basic Background 697
  16.3.2 Controlling the Color 699
  16.3.3 INT 10h Video Functions 701
  16.3.4 Library Procedure Examples 713
  16.3.5 Section Review 713

16.4 Drawing Graphics Using INT 10h 714
  16.4.1 INT 10h Pixel-Related Functions 715
  16.4.2 DrawLine Program 716
  16.4.3 Cartesian Coordinates Program 718
  16.4.4 Converting Cartesian Coordinates to Screen Coordinates 720
  16.4.5 Section Review 721

16.5 Memory-Mapped Graphics 722
  16.5.1 Mode 13h: 320 * 200, 256 Colors 722
  16.5.2 Memory-Mapped Graphics Program 724
  16.5.3 Section Review 727

16.6 Mouse Programming 727
  16.6.1 Mouse INT 33h Functions 727
  16.6.2 Mouse Tracking Program 732
  16.6.3 Section Review 737

16.7 Chapter Summary 738

16.8 Programming Exercises 739

A MASM Reference 741
B The x86 Instruction Set 763
C BIOS and MS-DOS Interrupts 797
D Answers to Review Questions (Chapters 14–16) 807

Glossary 816
Index 828
Preface

Assembly Language for x86 Processors, Eighth Edition, teaches assembly language programming and architecture for x86 and Intel64 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 recent versions of Microsoft Windows. 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 numerous languages.

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 x86 and Intel64 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 Eighth Edition

This edition represents this book’s transition into the world of interactive electronic textbooks. We’re very excited about this innovative concept, because for the first time readers will be able to experiment and interact with review questions, code animations, tutorial videos, and multiple-input exercises.

- **All section reviews** in the chapters have been rewritten as interactive questions, giving the reader immediate feedback on their answers. New questions were added, others removed, and many revised.
- **Code animations** allow the reader to step through program code and view both variable values and comments about the code. Readers no longer have to visually jump back and forth between program code and text explanations on the next page.
- **Links to timely tutorial videos** have been inserted in the text, so readers can receive tutoring on topics as they encounter them in the text. Previously, readers would need to purchase a separate subscription to gain access to the entire set of videos, presented as a list. In this edition, videos are free.
- **Multiple-input exercises** allow readers to browse a program listing and insert variable values into boxes next to the code. They receive immediate colorized feedback, giving them the opportunity to experiment until all input values are correct.
- **Hypertexted definitions of key terms** are placed throughout the text, connected to an online glossary.

In short, we have taken the successful content of this book (refined through many editions) and brought it into the interactive electronic textbook world.

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 confidence to work in any OS/machine-oriented environment.

Protected mode programming is entirely the focus of chapters 1 through 13. As such, students can create 32-bit and 64-bit programs that run under the most recent versions of Microsoft Windows. The remaining three legacy chapters cover 16-bit programming. These chapters cover BIOS programming, MS-DOS services, keyboard and mouse input, dist storage fundamentals, video programming, and graphics.

**Subroutine Libraries** We supply three versions of the subroutine library that students use for basic input/output, simulations, timing, and other useful tasks. The Irvine32 and Irvine64 libraries run in protected mode. The 16-bit version (Irvine16.lib) runs in real-address mode and is used only by Chapter 14 through Chapter 16. Full source code for the libraries is supplied on the companion website. 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, running in a recent version of Microsoft Visual Studio. In addition, batch files are supplied that permit students to assemble and run applications from the Windows command prompt. Information
Updates and corrections to this book may be found at the Companion website, 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 author’s website contains supplemental learning materials, study guides, and all the source code from the book’s examples. Two link libraries (32-bit and 64-bit) are supplied with the book, containing more than 40 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.

**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. More details can be found on Page xxvi.

**VideoNotes**  VideoNotes are Pearson’s visual tool designed to teach students key programming concepts and techniques. These short step-by-step videos demonstrate basic assembly language concepts. VideoNotes allow for self-paced instruction with easy navigation including the ability to select, play, rewind, fast-forward, and stop within each VideoNote exercise. Details below.

**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, Intel64 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 32-bit floating-point unit. Understanding the encoding of 32-bit 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**: Memory organization, interrupts, function calls, and standard MS-DOS file I/O services.
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.
   - Appendix A: MASM Reference
   - Appendix B: The x86 Instruction Set
   - Appendix C: BIOS and MS-DOS Interrupts
   - Appendix D: Answers to Review Questions ( Chapters 14–16)

Instructor and Student Resources

Instructor Resource Materials
The following protected instructor material is available on pearson.com
For username and password information, please contact your Pearson Representative.
   - Lecture PowerPoint Slides
   - Instructor Solutions Manual

Student Resource Materials
The following useful materials are located at www.asmirvine.com:
   - Getting Started, a comprehensive step-by-step tutorial that helps students customize Visual Studio for assembly language programming.
   - Corrections to errors found in the book.
   - Supplementary articles on assembly language programming topics.
   - Required support files for assembling and linking your programs, complete source code for all example programs in the book, and complete 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.
   - Debugging Tools: Tutorials on using the Microsoft Visual Studio debugger.

Acknowledgments
Many thanks are due to Tracy Johnson, Portfolio Manager for Computer Science at Pearson Education, who has provided friendly, helpful guidance for many years. Vanitha Puela of SPi Global did an excellent job on the book production, along with Amanda Brands as the Content Producer at Pearson.

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
About the Author

Kip Irvine has written five computer programming textbooks, for Intel Assembly Language, C++, Visual Basic (beginning and advanced), and COBOL. His book Assembly Language for Intel-Based Computers has been translated into six languages. His first college degrees (B.M., M.M., and doctorate) were in Music Composition, at University of Hawaii and University of Miami. He began programming computers for music synthesis around 1982 and taught programming at Miami-Dade Community College for 17 years. He earned an M.S. degree in Computer Science from the University of Miami, and taught computer programming in the School of Computing and Information Sciences at Florida International University for 18 years.