# Assembly Language for x86 Processors

**Seventh Edition**

KIP R. IRVINE

**Florida International University**
**School of Computing and Information Sciences**

PEARSON

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Vice President and Editorial Director, ECS: *Marcia Horton*
Executive Editor: *Tracy Johnson*
Executive Marketing Manager: *Tim Galligan*
Marketing Assistant: *Jon Bryant*
Program Management Team Lead: *Scott Disanno*
Program Manager: *Clare Romeo*
Project Manager: *Greg Dulles*
Senior Operations Specialist: *Nick Sklitsis*
Operations Specialist: *Linda Sager*
Permissions Project Manager: *Karen Sanatar*
Full-Service Project Management: *Pavithra Jayapaul, Jouve*
Printer/Binder: *Courier/Westford*
Typeface: *Times*



Previously published as *Assembly Language for Intel-Based Computers.*



**Library of Congress Cataloging-in-Publication Data**

Irvine, Kip R., 1951-
Assembly language for x86 processors / Kip R. Irvine, Florida International University, School of Computing and Information Sciences. — Seventh Edition.
pages cm
ISBN-13: 978-0-13-376940-1
ISBN-10: 0-13-376940-2
1. IBM microcomputers–Programming. 2. X86 assembly language (Computer program language) I. Title.
QA76.8.I77 2014
005.265—dc23 2013046432

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN-13: 978-0-13-376940-1
ISBN-10: 0-13-376940-2

*To Jack and Candy Irvine*

# Contents


## Preface xxiii

## 1 Basic Concepts 1

### 1.1 Welcome to Assembly Language 1
1.1.1 Questions You Might Ask 3
1.1.2 Assembly Language Applications 6
1.1.3 Section Review 6

### 1.2 Virtual Machine Concept 7
1.2.1 Section Review 9

### 1.3 Data Representation 9
1.3.1 Binary Integers 10
1.3.2 Binary Addition 12
1.3.3 Integer Storage Sizes 13
1.3.4 Hexadecimal Integers 13
1.3.5 Hexadecimal Addition 15
1.3.6 Signed Binary Integers 16
1.3.7 Binary Subtraction 18
1.3.8 Character Storage 19
1.3.9 Section Review 21

### 1.4 Boolean Expressions 22
1.4.1 Truth Tables for Boolean Functions 24
1.4.2 Section Review 26

### 1.5 Chapter Summary 26

### 1.6 Key Terms 27

### 1.7 Review Questions and Exercises 28
1.7.1 Short Answer 28
1.7.2 Algorithm Workbench 30

## 2 x86 Processor Architecture 32

### 2.1 General Concepts 33
2.1.1 Basic Microcomputer Design 33
2.1.2 Instruction Execution Cycle 34

2.1.3 Reading from Memory 36
2.1.4 Loading and Executing a Program 36
2.1.5 Section Review 37

**2.2 32-Bit x86 Processors 37**
2.2.1 Modes of Operation 37
2.2.2 Basic Execution Environment 38
2.2.3 x86 Memory Management 41
2.2.4 Section Review 42

**2.3 64-Bit x86-64 Processors 42**
2.3.1 64-Bit Operation Modes 43
2.3.2 Basic 64-Bit Execution Environment 43

**2.4 Components of a Typical x86 Computer 44**
2.4.1 Motherboard 44
2.4.2 Memory 46
2.4.3 Section Review 46

**2.5 Input–Output System 47**
2.5.1 Levels of I/O Access 47
2.5.2 Section Review 49

**2.6 Chapter Summary 50**

**2.7 Key Terms 51**

**2.8 Review Questions 52**

# 3 Assembly Language Fundamentals 53

**3.1 Basic Language Elements 54**
3.1.1 First Assembly Language Program 54
3.1.2 Integer Literals 55
3.1.3 Constant Integer Expressions 56
3.1.4 Real Number Literals 57
3.1.5 Character Literals 57
3.1.6 String Literals 58
3.1.7 Reserved Words 58
3.1.8 Identifiers 58
3.1.9 Directives 59
3.1.10 Instructions 60
3.1.11 Section Review 63

**3.2 Example: Adding and Subtracting Integers 63**
3.2.1 The *AddTwo* Program 63
3.2.2 Running and Debugging the AddTwo Program 65
3.2.3 Program Template 70
3.2.4 Section Review 70

### 3.3 Assembling, Linking, and Running Programs 71
3.3.1 The Assemble-Link-Execute Cycle 71
3.3.2 Listing File 71
3.3.3 Section Review 73

### 3.4 Defining Data 74
3.4.1 Intrinsic Data Types 74
3.4.2 Data Definition Statement 74
3.4.3 Adding a Variable to the AddTwo Program 75
3.4.4 Defining BYTE and SBYTE Data 76
3.4.5 Defining WORD and SWORD Data 78
3.4.6 Defining DWORD and SDWORD Data 79
3.4.7 Defining QWORD Data 79
3.4.8 Defining Packed BCD (TBYTE) Data 80
3.4.9 Defining Floating-Point Types 81
3.4.10 A Program That Adds Variables 81
3.4.11 Little-Endian Order 82
3.4.12 Declaring Uninitialized Data 83
3.4.13 Section Review 83

### 3.5 Symbolic Constants 84
3.5.1 Equal-Sign Directive 84
3.5.2 Calculating the Sizes of Arrays and Strings 85
3.5.3 EQU Directive 86
3.5.4 TEXTEQU Directive 87
3.5.5 Section Review 88

### 3.6 64-Bit Programming 88

### 3.7 Chapter Summary 90

### 3.8 Key Terms 91
3.8.1 Terms 91
3.8.2 Instructions, Operators, and Directives 92

### 3.9 Review Questions and Exercises 92
3.9.1 Short Answer 92
3.9.2 Algorithm Workbench 93

### 3.10 Programming Exercises 94

## 4 Data Transfers, Addressing, and Arithmetic 95

### 4.1 Data Transfer Instructions 96
4.1.1 Introduction 96
4.1.2 Operand Types 96
4.1.3 Direct Memory Operands 96

4.1.4 MOV Instruction 98
4.1.5 Zero/Sign Extension of Integers 99
4.1.6 LAHF and SAHF Instructions 101
4.1.7 XCHG Instruction 102
4.1.8 Direct-Offset Operands 102
4.1.9 Example Program (Moves) 103
4.1.10 Section Review 104

**4.2 Addition and Subtraction 105**
4.2.1 INC and DEC Instructions 105
4.2.2 ADD Instruction 105
4.2.3 SUB Instruction 106
4.2.4 NEG Instruction 106
4.2.5 Implementing Arithmetic Expressions 106
4.2.6 Flags Affected by Addition and Subtraction 107
4.2.7 Example Program (*AddSubTest*) 111
4.2.8 Section Review 112

**4.3 Data-Related Operators and Directives 112**
4.3.1 OFFSET Operator 112
4.3.2 ALIGN Directive 113
4.3.3 PTR Operator 114
4.3.4 TYPE Operator 115
4.3.5 LENGTHOF Operator 116
4.3.6 SIZEOF Operator 116
4.3.7 LABEL Directive 116
4.3.8 Section Review 117

**4.4 Indirect Addressing 117**
4.4.1 Indirect Operands 117
4.4.2 Arrays 118
4.4.3 Indexed Operands 119
4.4.4 Pointers 121
4.4.5 Section Review 122

**4.5 JMP and LOOP Instructions 123**
4.5.1 JMP Instruction 123
4.5.2 LOOP Instruction 124
4.5.3 Displaying an Array in the Visual Studio Debugger 125
4.5.4 Summing an Integer Array 126
4.5.5 Copying a String 127
4.5.6 Section Review 128

**4.6 64-Bit Programming 128**
4.6.1 MOV Instruction 128
4.6.2 64-Bit Version of SumArray 130
4.6.3 Addition and Subtraction 130
4.6.4 Section Review 131

### 4.7 Chapter Summary 132

### 4.8 Key Terms 133
4.8.1 Terms 133
4.8.2 Instructions, Operators, and Directives 133

### 4.9 Review Questions and Exercises 134
4.9.1 Short Answer 134
4.9.2 Algorithm Workbench 136

### 4.10 Programming Exercises 137

## 5 Procedures 139

### 5.1 Stack Operations 140
5.1.1 Runtime Stack (32-bit mode) 140
5.1.2 PUSH and POP Instructions 142
5.1.3 Section Review 145

### 5.2 Defining and Using Procedures 145
5.2.1 PROC Directive 145
5.2.2 CALL and RET Instructions 147
5.2.3 Nested Procedure Calls 148
5.2.4 Passing Register Arguments to Procedures 150
5.2.5 Example: Summing an Integer Array 150
5.2.6 Saving and Restoring Registers 152
5.2.7 Section Review 153

### 5.3 Linking to an External Library 153
5.3.1 Background Information 154
5.3.2 Section Review 155

### 5.4 The Irvine32 Library 155
5.4.1 Motivation for Creating the Library 155
5.4.2 Overview 157
5.4.3 Individual Procedure Descriptions 158
5.4.4 Library Test Programs 170
5.4.5 Section Review 178

### 5.5 64-Bit Assembly Programming 178
5.5.1 The *Irvine64* Library 178
5.5.2 Calling 64-Bit Subroutines 179
5.5.3 The x64 Calling Convention 179
5.5.4 Sample Program that Calls a Procedure 180

### 5.6 Chapter Summary 182

### 5.7 Key Terms 183
5.7.1 Terms 183
5.7.2 Instructions, Operators, and Directives 183

**5.8 Review Questions and Exercises 183**
5.8.1 Short Answer 183
5.8.2 Algorithm Workbench 186

**5.9 Programming Exercises 187**

# 6 Conditional Processing 189

**6.1 Conditional Branching 190**

**6.2 Boolean and Comparison Instructions 190**
6.2.1 The CPU Status Flags 191
6.2.2 AND Instruction 191
6.2.3 OR Instruction 192
6.2.4 Bit-Mapped Sets 194
6.2.5 XOR Instruction 195
6.2.6 NOT Instruction 196
6.2.7 TEST Instruction 196
6.2.8 CMP Instruction 197
6.2.9 Setting and Clearing Individual CPU Flags 198
6.2.10 Boolean Instructions in 64-Bit Mode 199
6.2.11 Section Review 199

**6.3 Conditional Jumps 199**
6.3.1 Conditional Structures 199
6.3.2 J*cond* Instruction 200
6.3.3 Types of Conditional Jump Instructions 201
6.3.4 Conditional Jump Applications 204
6.3.5 Section Review 208

**6.4 Conditional Loop Instructions 209**
6.4.1 LOOPZ and LOOPE Instructions 209
6.4.2 LOOPNZ and LOOPNE Instructions 209
6.4.3 Section Review 210

**6.5 Conditional Structures 210**
6.5.1 Block-Structured IF Statements 210
6.5.2 Compound Expressions 213
6.5.3 WHILE Loops 214
6.5.4 Table-Driven Selection 216
6.5.5 Section Review 219

**6.6 Application: Finite-State Machines 219**
6.6.1 Validating an Input String 219
6.6.2 Validating a Signed Integer 220
6.6.3 Section Review 224

**6.7 Conditional Control Flow Directives 225**
6.7.1 Creating IF Statements 226
6.7.2 Signed and Unsigned Comparisons 227
6.7.3 Compound Expressions 228
6.7.4 Creating Loops with .REPEAT and .WHILE 231

### 6.8 Chapter Summary 232

### 6.9 Key Terms 233
6.9.1 Terms 233
6.9.2 Instructions, Operators, and Directives 234

### 6.10 Review Questions and Exercises 234
6.10.1 Short Answer 234
6.10.2 Algorithm Workbench 236

### 6.11 Programming Exercises 237
6.11.1 Suggestions for Testing Your Code 237
6.11.2 Exercise Descriptions 238

## 7 Integer Arithmetic 242

### 7.1 Shift and Rotate Instructions 243
7.1.1 Logical Shifts and Arithmetic Shifts 243
7.1.2 SHL Instruction 244
7.1.3 SHR Instruction 245
7.1.4 SAL and SAR Instructions 246
7.1.5 ROL Instruction 247
7.1.6 ROR Instruction 247
7.1.7 RCL and RCR Instructions 248
7.1.8 Signed Overflow 249
7.1.9 SHLD/SHRD Instructions 249
7.1.10 Section Review 251

### 7.2 Shift and Rotate Applications 251
7.2.1 Shifting Multiple Doublewords 252
7.2.2 Binary Multiplication 253
7.2.3 Displaying Binary Bits 254
7.2.4 Extracting File Date Fields 254
7.2.5 Section Review 255

### 7.3 Multiplication and Division Instructions 255
7.3.1 MUL Instruction 255
7.3.2 IMUL Instruction 257
7.3.3 Measuring Program Execution Times 260
7.3.4 DIV Instruction 262
7.3.5 Signed Integer Division 264
7.3.6 Implementing Arithmetic Expressions 267
7.3.7 Section Review 269

### 7.4 Extended Addition and Subtraction 269
7.4.1 ADC Instruction 269
7.4.2 Extended Addition Example 270
7.4.3 SBB Instruction 272
7.4.4 Section Review 272

**7.5 ASCII and Unpacked Decimal Arithmetic 273**
7.5.1 AAA Instruction 274
7.5.2 AAS Instruction 276
7.5.3 AAM Instruction 276
7.5.4 AAD Instruction 276
7.5.5 Section Review 277

**7.6 Packed Decimal Arithmetic 277**
7.6.1 DAA Instruction 277
7.6.2 DAS Instruction 279
7.6.3 Section Review 279

**7.7 Chapter Summary 279**

**7.8 Key Terms 280**
7.8.1 Terms 280
7.8.2 Instructions, Operators, and Directives 280

**7.9 Review Questions and Exercises 281**
7.9.1 Short Answer 281
7.9.2 Algorithm Workbench 282

**7.10 Programming Exercises 284**

# 8 Advanced Procedures 286

**8.1 Introduction 287**

**8.2 Stack Frames 287**
8.2.1 Stack Parameters 288
8.2.2 Disadvantages of Register Parameters 288
8.2.3 Accessing Stack Parameters 290
8.2.4 32-Bit Calling Conventions 293
8.2.5 Local Variables 295
8.2.6 Reference Parameters 297
8.2.7 LEA Instruction 298
8.2.8 ENTER and LEAVE Instructions 298
8.2.9 LOCAL Directive 300
8.2.10 The Microsoft x64 Calling Convention 301
8.2.11 Section Review 302

**8.3 Recursion 302**
8.3.1 Recursively Calculating a Sum 303
8.3.2 Calculating a Factorial 304
8.3.3 Section Review 311

**8.4 INVOKE, ADDR, PROC, and PROTO 311**
8.4.1 INVOKE Directive 311
8.4.2 ADDR Operator 312
8.4.3 PROC Directive 313
8.4.4 PROTO Directive 316

8.4.5 Parameter Classifications 319
8.4.6 Example: Exchanging Two Integers 320
8.4.7 Debugging Tips 321
8.4.8 WriteStackFrame Procedure 322
8.4.9 Section Review 323

**8.5 Creating Multimodule Programs 323**
8.5.1 Hiding and Exporting Procedure Names 323
8.5.2 Calling External Procedures 324
8.5.3 Using Variables and Symbols across Module Boundaries 325
8.5.4 Example: ArraySum Program 326
8.5.5 Creating the Modules Using Extern 326
8.5.6 Creating the Modules Using INVOKE and PROTO 330
8.5.7 Section Review 333

**8.6 Advanced Use of Parameters (Optional Topic) 333**
8.6.1 Stack Affected by the USES Operator 333
8.6.2 Passing 8-Bit and 16-Bit Arguments on the Stack 335
8.6.3 Passing 64-Bit Arguments 336
8.6.4 Non-Doubleword Local Variables 337

**8.7 Java Bytecodes (Optional Topic) 339**
8.7.1 Java Virtual Machine 339
8.7.2 Instruction Set 340
8.7.3 Java Disassembly Examples 341
8.7.4 Example: Conditional Branch 344

**8.8 Chapter Summary 346**

**8.9 Key Terms 347**
8.9.1 Terms 347
8.9.2 Instructions, Operators, and Directives 348

**8.10 Review Questions and Exercises 348**
8.10.1 Short Answer 348
8.10.2 Algorithm Workbench 348

**8.11 Programming Exercises 349**

# 9 Strings and Arrays 352

**9.1 Introduction 352**

**9.2 String Primitive Instructions 353**
9.2.1 MOVSB, MOVSW, and MOVSD 354
9.2.2 CMPSB, CMPSW, and CMPSD 355
9.2.3 SCASB, SCASW, and SCASD 356
9.2.4 STOSB, STOSW, and STOSD 356
9.2.5 LODSB, LODSW, and LODSD 356
9.2.6 Section Review 357

### 9.3 Selected String Procedures 357
9.3.1 Str_compare Procedure 358
9.3.2 Str_length Procedure 359
9.3.3 Str_copy Procedure 359
9.3.4 Str_trim Procedure 360
9.3.5 Str_ucase Procedure 363
9.3.6 *String Library Demo* Program 364
9.3.7 String Procedures in the Irvine64 Library 365
9.3.8 Section Review 368

### 9.4 Two-Dimensional Arrays 368
9.4.1 Ordering of Rows and Columns 368
9.4.2 Base-Index Operands 369
9.4.3 Base-Index-Displacement Operands 371
9.4.4 Base-Index Operands in 64-Bit Mode 372
9.4.5 Section Review 373

### 9.5 Searching and Sorting Integer Arrays 373
9.5.1 Bubble Sort 373
9.5.2 Binary Search 375
9.5.3 Section Review 382

### 9.6 Java Bytecodes: String Processing (Optional Topic) 382

### 9.7 Chapter Summary 383

### 9.8 Key Terms and Instructions 384

### 9.9 Review Questions and Exercises 384
9.9.1 Short Answer 384
9.9.2 Algorithm Workbench 385

### 9.10 Programming Exercises 386

## 10 Structures and Macros 390

### 10.1 Structures 390
10.1.1 Defining Structures 391
10.1.2 Declaring Structure Variables 393
10.1.3 Referencing Structure Variables 394
10.1.4 Example: Displaying the System Time 397
10.1.5 Structures Containing Structures 399
10.1.6 Example: Drunkard's Walk 399
10.1.7 Declaring and Using Unions 403
10.1.8 Section Review 405

### 10.2 Macros 405
10.2.1 Overview 405
10.2.2 Defining Macros 406
10.2.3 Invoking Macros 407

10.2.4 Additional Macro Features 408
10.2.5 Using the Book's Macro Library (32-bit mode only) 412
10.2.6 Example Program: Wrappers 419
10.2.7 Section Review 420

**10.3 Conditional-Assembly Directives 420**
10.3.1 Checking for Missing Arguments 421
10.3.2 Default Argument Initializers 422
10.3.3 Boolean Expressions 423
10.3.4 IF, ELSE, and ENDIF Directives 423
10.3.5 The IFIDN and IFIDNI Directives 424
10.3.6 Example: Summing a Matrix Row 425
10.3.7 Special Operators 428
10.3.8 Macro Functions 431
10.3.9 Section Review 433

**10.4 Defining Repeat Blocks 433**
10.4.1 WHILE Directive 433
10.4.2 REPEAT Directive 434
10.4.3 FOR Directive 434
10.4.4 FORC Directive 435
10.4.5 Example: Linked List 436
10.4.6 Section Review 437

**10.5 Chapter Summary 438**

**10.6 Key Terms 439**
10.6.1 Terms 439
10.6.2 Operators and Directives 439

**10.7 Review Questions and Exercises 440**
10.7.1 Short Answer 440
10.7.2 Algorithm Workbench 440

**10.8 Programming Exercises 442**

# 11 MS-Windows Programming 445

**11.1 Win32 Console Programming 445**
11.1.1 Background Information 446
11.1.2 Win32 Console Functions 450
11.1.3 Displaying a Message Box 452
11.1.4 Console Input 455
11.1.5 Console Output 461
11.1.6 Reading and Writing Files 463
11.1.7 File I/O in the Irvine32 Library 468
11.1.8 Testing the File I/O Procedures 470
11.1.9 Console Window Manipulation 473
11.1.10 Controlling the Cursor 476

11.1.11 Controlling the Text Color 477
11.1.12 Time and Date Functions 479
11.1.13 Using the 64-Bit Windows API 482
11.1.14 Section Review 484

**11.2 Writing a Graphical Windows Application 484**
11.2.1 Necessary Structures 484
11.2.2 The MessageBox Function 486
11.2.3 The WinMain Procedure 486
11.2.4 The WinProc Procedure 487
11.2.5 The ErrorHandler Procedure 488
11.2.6 Program Listing 488
11.2.7 Section Review 492

**11.3 Dynamic Memory Allocation 492**
11.3.1 HeapTest Programs 496
11.3.2 Section Review 499

**11.4 x86 Memory Management 499**
11.4.1 Linear Addresses 500
11.4.2 Page Translation 503
11.4.3 Section Review 505

**11.5 Chapter Summary 505**

**11.6 Key Terms 507**

**11.7 Review Questions and Exercises 507**
11.7.1 Short Answer 507
11.7.2 Algorithm Workbench 508

**11.8 Programming Exercises 509**

# 12 Floating-Point Processing and Instruction Encoding 511

**12.1 Floating-Point Binary Representation 511**
12.1.1 IEEE Binary Floating-Point Representation 512
12.1.2 The Exponent 514
12.1.3 Normalized Binary Floating-Point Numbers 514
12.1.4 Creating the IEEE Representation 514
12.1.5 Converting Decimal Fractions to Binary Reals 516
12.1.6 Section Review 518

**12.2 Floating-Point Unit 518**
12.2.1 FPU Register Stack 519
12.2.2 Rounding 521
12.2.3 Floating-Point Exceptions 523
12.2.4 Floating-Point Instruction Set 523

12.2.5 Arithmetic Instructions 526
12.2.6 Comparing Floating-Point Values 530
12.2.7 Reading and Writing Floating-Point Values 533
12.2.8 Exception Synchronization 534
12.2.9 Code Examples 535
12.2.10 Mixed-Mode Arithmetic 537
12.2.11 Masking and Unmasking Exceptions 538
12.2.12 Section Review 539

**12.3 x86 Instruction Encoding 539**
12.3.1 Instruction Format 540
12.3.2 Single-Byte Instructions 541
12.3.3 Move Immediate to Register 541
12.3.4 Register-Mode Instructions 542
12.3.5 Processor Operand-Size Prefix 543
12.3.6 Memory-Mode Instructions 544
12.3.7 Section Review 547

**12.4 Chapter Summary 547**

**12.5 Key Terms 549**

**12.6 Review Questions and Exercises 549**
12.6.1 Short Answer 549
12.6.2 Algorithm Workbench 550

**12.7 Programming Exercises 551**

# 13 High-Level Language Interface 555

**13.1 Introduction 555**
13.1.1 General Conventions 556
13.1.2 .MODEL Directive 557
13.1.3 Examining Compiler-Generated Code 559
13.1.4 Section Review 564

**13.2 Inline Assembly Code 564**
13.2.1 __asm Directive in Visual C++ 564
13.2.2 File Encryption Example 566
13.2.3 Section Review 569

**13.3 Linking 32-Bit Assembly Language Code to C/C++ 570**
13.3.1 IndexOf Example 570
13.3.2 Calling C and C++ Functions 574
13.3.3 Multiplication Table Example 576
13.3.4 Calling C Library Functions 579
13.3.5 Directory Listing Program 582
13.3.6 Section Review 583

**13.4 Chapter Summary 583**

**13.5 Key Terms 584**

**13.6 Review Questions 584**

**13.7 Programming Exercises 585**

***Chapters are available from the Companion Web site***

# 14 16-Bit MS-DOS Programming 14.1

**14.1 MS-DOS and the IBM-PC 14.1**

14.1.1 Memory Organization 14.2
14.1.2 Redirecting Input-Output 14.3
14.1.3 Software Interrupts 14.4
14.1.4 INT Instruction 14.5
14.1.5 Coding for 16-Bit Programs 14.6
14.1.6 Section Review 14.7

**14.2 MS-DOS Function Calls (INT 21h) 14.7**

14.2.1 Selected Output Functions 14.9
14.2.2 Hello World Program Example 14.11
14.2.3 Selected Input Functions 14.12
14.2.4 Date/Time Functions 14.16
14.2.5 Section Review 14.20

**14.3 Standard MS-DOS File I/O Services 14.20**

14.3.1 Create or Open File (716Ch) 14.22
14.3.2 Close File Handle (3Eh) 14.23
14.3.3 Move File Pointer (42h) 14.23
14.3.4 Get File Creation Date and Time 14.24
14.3.5 Selected Library Procedures 14.24
14.3.6 Example: Read and Copy a Text File 14.25
14.3.7 Reading the MS-DOS Command Tail 14.27
14.3.8 Example: Creating a Binary File 14.30
14.3.9 Section Review 14.33

**14.4 Chapter Summary 14.33**

**14.5 Programming Exercises 14.35**

# 15 Disk Fundamentals 15.1

**15.1 Disk Storage Systems 15.1**

15.1.1 Tracks, Cylinders, and Sectors 15.2
15.1.2 Disk Partitions (Volumes) 15.4
15.1.3 Section Review 15.4

**15.2 File Systems 15.5**
15.2.1 FAT12 15.6
15.2.2 FAT16 15.6
15.2.3 FAT32 15.6
15.2.4 NTFS 15.7
15.2.5 Primary Disk Areas 15.7
15.2.6 Section Review 15.8

**15.3 Disk Directory 15.9**
15.3.1 MS-DOS Directory Structure 15.10
15.3.2 Long Filenames in MS-Windows 15.12
15.3.3 File Allocation Table (FAT) 15.14
15.3.4 Section Review 15.14

**15.4 Reading and Writing Disk Sectors 15.15**
15.4.1 Sector Display Program 15.16
15.4.2 Section Review 15.19

**15.5 System-Level File Functions 15.20**
15.5.1 Get Disk Free Space (7303h) 15.20
15.5.2 Create Subdirectory (39h) 15.23
15.5.3 Remove Subdirectory (3Ah) 15.23
15.5.4 Set Current Directory (3Bh) 15.23
15.5.5 Get Current Directory (47h) 15.24
15.5.6 Get and Set File Attributes (7143h) 15.24
15.5.7 Section Review 15.25

**15.6 Chapter Summary 15.25**

**15.7 Programming Exercises 15.26**

# 16 BIOS-Level Programming 16.1

**16.1 Introduction 16.1**
16.1.1 BIOS Data Area 16.2

**16.2 Keyboard Input with INT 16h 16.3**
16.2.1 How the Keyboard Works 16.3
16.2.2 INT 16h Functions 16.4
16.2.3 Section Review 16.8

**16.3 VIDEO Programming with INT 10h 16.8**
16.3.1 Basic Background 16.8
16.3.2 Controlling the Color 16.10
16.3.3 INT 10h Video Functions 16.12
16.3.4 Library Procedure Examples 16.22
16.3.5 Section Review 16.23

**16.4 Drawing Graphics Using INT 10h 16.23**
16.4.1 INT 10h Pixel-Related Functions 16.24
16.4.2 DrawLine Program 16.25
16.4.3 Cartesian Coordinates Program 16.27
16.4.4 Converting Cartesian Coordinates to Screen Coordinates 16.29
16.4.5 Section Review 16.30

**16.5 Memory-Mapped Graphics 16.30**
16.5.1 Mode 13h: 320 X 200, 256 Colors 16.30
16.5.2 Memory-Mapped Graphics Program 16.32
16.5.3 Section Review 16.34

**16.6 Mouse Programming 16.35**
16.6.1 Mouse INT 33h Functions 16.35
16.6.2 Mouse Tracking Program 16.40
16.6.3 Section Review 16.44

**16.7 Chapter Summary 16.45**

**16.8 Programming Exercises 16.46**

**17 Expert MS-DOS Programming 17.1**

**17.1 Introduction 17.1**

**17.2 Defining Segments 17.2**
17.2.1 Simplified Segment Directives 17.2
17.2.2 Explicit Segment Definitions 17.4
17.2.3 Segment Overrides 17.7
17.2.4 Combining Segments 17.7
17.2.5 Section Review 17.9

**17.3 Runtime Program Structure 17.9**
17.3.1 Program Segment Prefix 17.10
17.3.2 COM Programs 17.10
17.3.3 EXE Programs 17.11
17.3.4 Section Review 17.13

**17.4 Interrupt Handling 17.13**
17.4.1 Hardware Interrupts 17.14
17.4.2 Interrupt Control Instructions 17.16
17.4.3 Writing a Custom Interrupt Handler 17.16
17.4.4 Terminate and Stay Resident Programs 17.19
17.4.5 Application: The No_Reset Program 17.19
17.4.6 Section Review 17.23

**17.5 Hardware Control Using I/O Ports 17.23**
17.5.1 Input–Output Ports 17.24
17.5.2 PC Sound Program 17.24

**17.6 Chapter Summary 17.26**

**Appendix A MASM Reference 587**
**Appendix B The x86 Instruction Set 609**
**Appendix C Answers to Section Review Questions 644**

***Appendices are available from the Companion Web site***

**Appendix D BIOS and MS-DOS Interrupts D.1**
**Appendix E Answers to Review Questions (Chapters 14–17) E.1**

**Index 664**

# Preface

*Assembly Language for x86 Processors, Seventh 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 Seventh Edition

In this revision, we increased the discussions of program examples early in the book, added more supplemental review questions and key terms, introduced 64-bit programming, and reduced our dependence on the book's subroutine library. To be more specific, here are the details:

- Early chapters now include short sections that feature 64-bit CPU architecture and programming, and we have created a 64-bit version of the book's subroutine library named *Irvine64*.
- Many of the review questions and exercises have been modified, replaced, and moved from the middle of the chapter to the end of chapters, and divided into two sections: (1) Short answer questions, and (2) Algorithm workbench exercises. The latter exercises require the student to write a short amount of code to accomplish a goal.
- Each chapter now has a *Key Terms* section, listing new terms and concepts, as well as new MASM directives and Intel instructions.
- New programming exercises have been added, others removed, and a few existing exercises were modified.
- There is far less dependency on the author's subroutine libraries in this edition. Students are encouraged to call system functions themselves and use the Visual Studio debugger to step through the programs. The Irvine32 and Irvine64 libraries are available to help students handle input/output, but their use is not required.
- New tutorial videos covering essential content topics have been created by the author and added to the Pearson website.

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.

Protected mode programming is entirely the focus of the printed chapters (1 through 13). As such, students will create 32-bit and 64-bit programs that run under the most recent versions of Microsoft Windows. The remaining four chapters cover 16-bit programming, and are supplied in electronic form. These chapters cover BIOS programming, MS-DOS services, keyboard and mouse input, video programming, and graphics. One chapter covers disk storage fundamentals. Another chapter covers advanced DOS programming techniques.

*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 Chapters 14 through 17. 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 Version 11.0, running in Microsoft Visual Studio 2012. 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. 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.

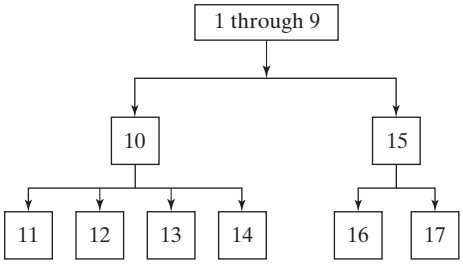
*Instructional Aids* All the program listings are available on the Web. Instructors are provided a test bank, answers to review questions, solutions to programming exercises, and a Microsoft PowerPoint slide presentation for each chapter.

*VideoNotes* VideoNotes are Pearson's new visual tool designed to teach students key programming concepts and techniques. These short step-by-step videos demonstrate 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.

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, 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 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.

- **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:

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.
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 14–17)

## 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 14: *16-Bit MS-DOS Programming*
  - 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 14–17)*

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 "*Video Notes and Web Chapters*." 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, located at *www.asmirvine.com*:

- *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 the Microsoft Visual Studio debugger.

## Acknowledgments

Many thanks are due to Tracy Johnson, Executive Editor for Computer Science at Pearson Education, who has provided friendly, helpful guidance over the past few years. Pavithra Jayapaul of Jouve did an excellent job on the book production, along with Greg Dulles as the production editor 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. Kip earned an M.S. degree in Computer Science from the University of Miami, and he has been a full-time member of the faculty in the School of Computing and Information Sciences at Florida International University since 2000.