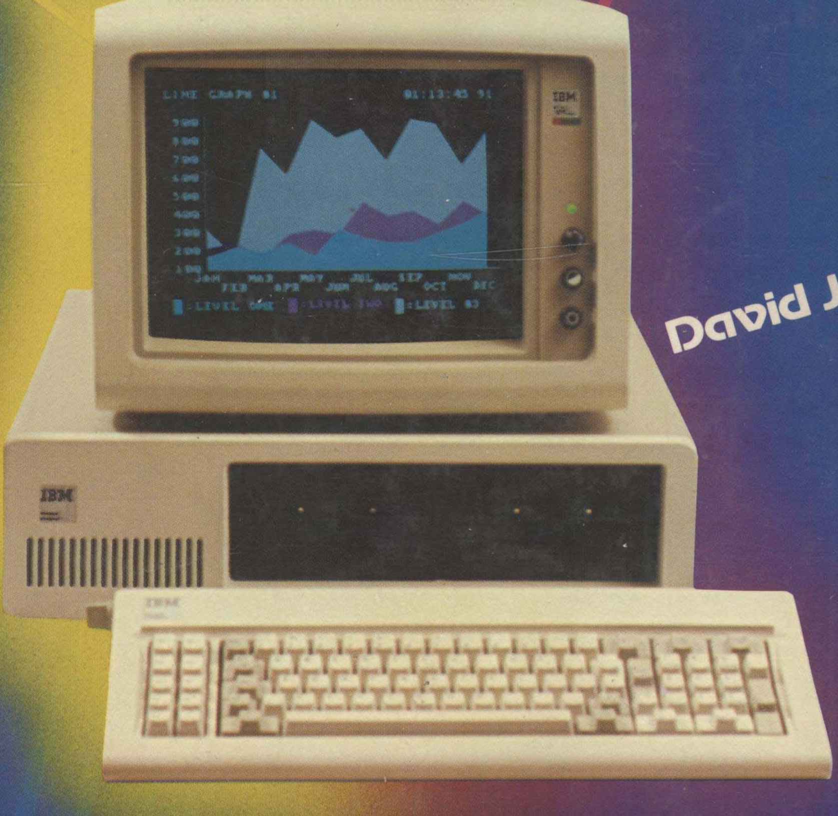
ASSEMBLY LANGUAGE PROGRAMMING FOR THE IBM PERSONAL COMPUTER

David J. Bradley



David J. Bradley was a member of the team that designed and built the IBM Personal Computer.

Assembly Language Programming for the IBM Personal Computer

DAVID J. BRADLEY

IBM Corporation

PRENTICE-HALL, INC. Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

BRADLEY, DAVID J. (date)
Assembly language programming for the IBM Personal Computer.

Includes index.
1. IBM Personal Computer—Programming. 2. Assembler language (Computer program language) I. Title.
QA76.8.I2594B7 1984 001.64'2 83-8638
ISBN 0-13-049189-6
ISBN 0-13-049171-3 (pbk.)

Editorial/production supervision
and interior design: Kathryn Gollin Marshak
Cover design: Photo Plus Art—Celine Brandes
Manufacturing buyer: Gordon Osbourne

Printed in the United States of America

10 9 8 7 6 5 4

ISBN 0-13-049171-3 {P}
ISBN 0-13-049189-6 {C}

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*

Preface

This book teaches you how to write assembly language programs for the IBM Personal Computer. It also explains how to use the IBM Personal Computer to write those programs. Finally, it shows you ways to use assembly language programs with the rest of the system.

This is a first book in assembly language programming. Previous programming experience, in a high-level language, is desirable for the use of the book. There is no discussion of algorithm design and programming techniques. It may be difficult for you to use this book if you have had no previous experience in writing BASIC or Pascal or similar programs. However, the text presents material in a way that should allow you to begin assembly programming even if you don't know anything about the internal workings of a computer.

The first section, comprising the first two chapters, teaches the fundamentals of computer operation. It emphasizes those functions of the computer that aren't readily apparent when using a high-level language. This includes a discussion of binary arithmetic and data representation. The section also deals with the general operation of the assembler. This section may be skipped by programmers with assembly language experience.

The second portion of the book, consisting of Chapters 3, 4, and 7, is an explanation of the processor used in the IBM Personal Computer, the Intel 8088. This includes a description of the 8088, its registers, and its addressing modes. The book presents the instruction set of the 8088 together with examples showing the use of most of the instructions. Chapter 7 is devoted to the 8087 Numeric Data

Processor. It describes the additional data types and instructions made available through the 8087. The chapter uses several examples to show the operation of the numeric coprocessor.

Where the preceding section was sufficiently broad to cover nearly all systems using the 8086/8088 processor family, the final sections are linked directly to the IBM Personal Computer. Chapters 5 and 6 deal with the creation and use of programs on the IBM machine. You'll learn how to use the assembly language "tools." These include the line editor, the assembler, and the linker, as well as an overview description of the Disk Operating System. This section describes the Disk Operating System as a program environment. Chapter 6 describes some of the special tools available as part of the Macro Assembler. This includes not only macros, but also the special data definition tools that are essential for assembly language programming.

Chapters 8, 9, and 10 cover the hardware and "microcode" portions of the IBM Personal Computer. This section emphasizes the unique aspects of the IBM Personal Computer and the application of assembly language programming to it. Of particular interest is Chapter 10, which describes the techniques of linking an assembly language program to another program or system. This chapter contains several different methods of fitting an assembly language program with another program, or making the program a permanent addition to the system.

The author was a member of the team that designed and built the IBM Personal Computer. I would like to thank all of the people in the Personal Computer organization who have helped me during the preparation of this text. Special thanks go to Dave O'Connor and Jud McCarthy, my managers during this period. Most important, I would like to thank my wife, Cynthia, for her help and encouragement.

David J. Bradley

Contents

Preface *xi*

Chapter 1 INTRODUCTION *1*

Assembly Language Programming 1
IBM Personal Computer 3
This Book 3

Chapter 2 COMPUTER FUNDAMENTALS *5*

Binary Arithmetic 5
Two's Complement 7
Hexadecimal Representation 8
Machine Language and Assembly Language 9
Assembly Language Syntax 11
Assembler Operation 12
Bits, Bytes, and Words 13
Bit Numbering 16
Character Set 17
Computer Operation 19
Subroutines 21
Stack 22
Interrupts 25

Chapter 3 THE 8088 MICROPROCESSOR 28

8088 Programming Model 28
General Registers 29
Addressing Registers 30
Direct Addressing 31
Address Calculation 32
Base + Displacement Addressing 33
Base + Index + Displacement 33
Mod-r/m Byte 35
Physical Addressing 36
Segment Registers 37
Segment Overrides 38
SEGMENT Statement 39
ASSUME Statement 40
Control Registers 42

Instruction Pointer, 42
Stack Pointer, 43
Flag Register, 44
Sign Flag, 45
Zero Flag, 45
Parity Flag, 45
Carry Flag, 45
Auxiliary Carry Flag, 47
Overflow Flag, 49
Trap Flag, 50
Interrupt Flag, 50
Direction Flag, 51

Interrupt Vectors 52

Chapter 4 THE 8088 INSTRUCTION SET 54

Data Movement 54

Move, 55
Exchange, 58

Input and Output, 58
Load Effective Address, 59
Load Pointer, 60
Flag Transfer, 60
Translate, 61

Stack Operations 63

Parameter Passing 65

Arithmetic Instructions 67

Addition, 67
Subtraction, 70
Single-Operand Arithmetic, 70
Compare, 71
Decimal Adjust, 72
ASCII Adjust: Addition and Subtraction, 73
Multiply, 74
ASCII Adjust: Multiply, 75
Divide, 76
ASCII Adjust: Division, 78
Convert, 79

Arithmetic Example 79

Logical Instructions 80

Shift and Rotate Instructions 83

String Instructions 87

Load and Store, 88
REP Prefix, 89
Move String, 90
Scan and Compare, 91

Transfer of Control Instructions 93

NEAR and FAR, 94
Jump Addressing, 94
Unconditional Transfers, 95
Conditional Jumps, 97
Condition Code Tests, 98
Loop Control, 102

Processor Control Instructions 104

Flag Setting, 104
Special Instructions, 105

Chapter 5 USING DOS AND THE ASSEMBLER 108

Disk Operating System 108

File System 109

File Names 110

Directory 111

Command Processor 111

DOS Functions 115

File Control Block 118

.COM and .EXE Files 124

Creating an Assembly Language Program 126

DOS Line Editor 127

Assembler and Macro Assembler 130

Symbol Table, 132
Cross-Reference, 133

Linker 135

Multiple Modules, 135
EXTRN and PUBLIC, 136
Link Operation, 139
Link Map, 140

DEBUG 141

Converting from .EXE to .COM 146

Chapter 6 FEATURES OF THE MACRO ASSEMBLER 150

Macros 150

Macro Arguments 154

Conditional Assembly 156

Repeat Macros 161

Macro Operators 162

INCLUDE Statement 164

Segments 165

Structures 171

Records 175

Chapter 7 THE 8087 NUMERIC DATA PROCESSOR 181

8087 Operation 181

8087 Data Types 183

Floating-Point Data Representation 186

8087 Real Data Formats 190

Real Number Data Definition 192

8087 Programming Model 193

Register Stack, 193
Control Word, 195
Status Word, 197

8087 Instruction Set 198

Data Transfer Instructions, 199
Control Instructions, 203
Arithmetic Instructions, 206
Comparison Instructions, 209
Functions and Transcendentals, 212

Examples 215

Powers of 10, 216
Ten to the X, 218
Floating-Point Display, 220
Quadratic Equation, 222
SIN of an Angle 224

Debugging with the 8087 227

Chapter 8 THE IBM PERSONAL COMPUTER 230

System Hardware 231

Speaker 232

Keyboard 235

Time of Day 239

System Features 245

Display Adapters 245

Monochrome Display and Printer Adapter, 245
Color/Graphics Monitor Adapter, 248
Text Mode, 248
Graphics Mode, 255
Color in 320 × 200 APA Mode, 256
High-Resolution Graphics, 256

Parallel Printer Adapter 257

Asynchronous Communications Adapter 258

Game Control Adapter 261

Diskette Drive Adapter 263

Direct Memory Access 265

Chapter 9 THE ROM BIOS ***268***

Notes on the ROM BIOS Listing 270

Power-on Self-Test 271

ROM BIOS Interrupts 273

Device Drivers 273

User Routines, 274
Parameter Blocks, 275

ROM BIOS Data Area 275

Device Driver Routines 276

System Service 276

Printer and Asynchronous Communications 278

Keyboard 280

Keyboard Data, 280
Inside the Keyboard BIOS, 282

Cassette 283

Diskette 285

Diskette Data Areas, 285
Read and Write Commands, 286

Verify Command, 287
Format Command, 287

Video 290

Video Data Areas, 290
Video BIOS Functions, 290
Mode Setting, 293
Scrolling, 293
Character Read and Write, 295
Text in Graphics Modes, 295
Graphics, 297
Write Teletype, 298

Chapter 10 ASSEMBLY LANGUAGE EXTENSIONS AND SUBROUTINES 299

BIOS Extensions 300

DOS Exit and Stay Resident 300

High-RAM Load 307

Assembly Language Subroutines 314

BASIC BLOAD Assembly Routine 314

Embedded Short Program 319

Compiled High-Level Languages 322

Summary 325

Appendix A 8088 INSTRUCTION SET 326

Appendix B 8087 INSTRUCTION SET 439

Bibliography 333

Index 335

CHAPTER 1

Introduction

Welcome to assembly language programming for the IBM Personal Computer. This book teaches you how to write assembly language programs, and it uses the IBM Personal Computer as the teaching vehicle.

ASSEMBLY LANGUAGE PROGRAMMING

Why should you be interested in assembly language programming? High-level languages such as BASIC, FORTRAN, and Pascal are commonly used today. You are probably familiar with at least one high-level language. If you're currently using an IBM Personal Computer, you know that the BASIC interpreter is part of the system unit. Why should you bother with another programming language, especially one that's going to be difficult at times? Even with today's high-powered languages, you still need assembly language because of its power and precision.

Assembly programs can be very powerful. Given programmers with equal skills and abilities, the assembly language program will be smaller in size and faster in execution than the same program written in a high-level language. This holds true for virtually all small and medium-size programs. Unfortunately, assembly language programs lose some of their power as the program size increases. That's because of the attention to detail that's necessary for an assembly program. As you'll see throughout this text, assembly language requires you to decide each and every action of the computer. For a small program, this lets you optimize the program to work efficiently with the computer hardware. For a larger program, the myriad details may prevent you from doing an efficient job of the entire program, even though some individual components of the program will be very good. Clearly, assembly language programming is not the answer for all programs.

Assembly language programs are also very precise. Because assembly language allows the programmer to deal directly with the hardware, assembly programs can do things that no other program can. Certainly for I/O device programming, where the program requires control of the individual I/O device bits, assembly language programming is the only appropriate choice.

Clearly, the power and precision of assembly language programming offers advantages. But its attention to detail also causes problems. When is the correct time to use assembly language programming?

Certainly, you must use assembly language programs when there's no other way to write the program. For example, IBM programmers wrote all of the I/O device control programs for the IBM Personal Computer using assembly language routines. IBM needed the precision of assembly language to control the I/O devices and the interrupt system, where no other programming language would work. Similarly, IBM wrote the diagnostic routines, which must check every detail of the hardware, in assembly language.

You should also use assembly language when performance is a major concern. This performance may be either the execution time of a program or its final size. The FORTRAN mathematics subroutine library is an example of a program that requires good performance, both in time and space. The math routines are a part of every FORTRAN program, so they should be as small as possible. Also, those routines handle all of the math functions in a FORTRAN program, so they are used frequently. Therefore, they should execute rapidly.

What programs aren't candidates for assembly language? Well, you can write any program in assembly language, but a program of significant size is better handled in a higher-level language, such as BASIC or Pascal. These languages let you concentrate on the problem. You don't have to deal directly with the details of the hardware and the processor. A high-level language lets you step back and see the forest rather than the trees.

Obviously, then, you need to mix assembly language programs with high-level language programs. This text will concentrate on assembly language programming for those tasks for which it is well suited, such as I/O control. In fact, the final chapter deals directly with the problem of linking assembly language programs with other programming languages. These methods give you the best of both worlds. You can use assembly routines for the precise control and power when you need it, and high-level routines for the overall program. All you have to do is hook them together to make it happen.

There's a final reason for learning assembly language programming. Only by writing programs at this level of detail can you learn how the machine works at its lowest levels. If you want to know everything there is to know about a computer, you have to be familiar with its assembly language. The only way to do that is to write programs in that language. Reading the manual alone won't do the job.

IBM PERSONAL COMPUTER

Why does this text use the IBM Personal Computer as the basis for learning assembly language programming? There are several reasons. First, the IBM Personal Computer is new and powerful. As a personal computer, the IBM machine has extended capabilities, beyond those of earlier personal computers. As you'll see in more detail later, the Personal Computer uses the Intel 8088 microprocessor. This processor can do 16-bit arithmetic and address over 1 million characters of storage. These capabilities give it a power much closer to large computers than to the earlier personal computers.

Second, the IBM Personal Computer has all the development tools that you'll need to do assembly programming. Besides the assembler, IBM provides an editor, linker, and disk operating system to put it all together. There's even a debugger to help you take it all apart and then put it back together right.

Finally, the IBM Personal Computer is a good system on which to learn assembly language programming because of its availability. It is an inexpensive machine, yet still offers all the capabilities that assembly language programming requires. Even more, as a "personal" computer, the machine belongs to you, at least while you're executing your program. This means that you can try out things that you couldn't on a larger machine that you're sharing with others. You can take over the I/O attachments and program them to do interesting things. You can do anything you want with any part of the system. You can do this even if it brings the system "down." Since it's a personal machine, when there's a problem, you just turn the machine off and start all over. The only person you can interfere with is yourself. As a personal machine, it makes a great development environment.

THIS BOOK

This book introduces you to the IBM Personal Computer and its assembly language. Although the major concentration is on assembly programming, this text describes the programming aspects of the major hardware features of the machine. You'll learn how the I/O devices work and how the programs make them work correctly. You'll also learn how to write your own assembly language programs using the IBM Personal Computer. After you write these programs, this book shows you how to link them into a high-level language program or build your program into the system.

Assembly language programming is a fascinating experience, but often frustrating. This book uses examples to show you some programs that do work. These examples should get you started. But the only way you can learn about programming is to do the programs yourself. You have to make your own mistakes to learn. Good luck, and have fun.

CHAPTER

2

Computer Fundamentals

This chapter will explain the characteristics of computers. It will tell you how computers work and why they do things the way they do. Some of the concepts may be familiar to you. If you have had no previous experience with assembly language programming, many of these operations will be new.

BINARY ARITHMETIC

All computers store information using the binary system. This means that every item of information stored by the computer has only two choices. These choices are labeled "on" and "off," "true" and "false," or "1" and "0." The computer stores these values as voltage levels. Fortunately, we don't need to be concerned with the voltages, only the numbers, when writing programs. From the simple numbers 0 and 1, we can do very complicated arithmetic.

Because of the binary representation of data, computers use base 2 arithmetic to perform their computations. Base 2 arithmetic uses only the two numbers, 0 and 1. We normally use base 10, or decimal arithmetic. In base 10 arithmetic, there are 10 different numerals used, 0 through 9. Base 2 arithmetic can be thought of as the system for people with two fingers.

The limitation of only 10 numerals in base 10 arithmetic does not prevent us from representing larger numbers. We use multiple-digit numbers, where each position within the number is a different power of 10. The rightmost digit of any