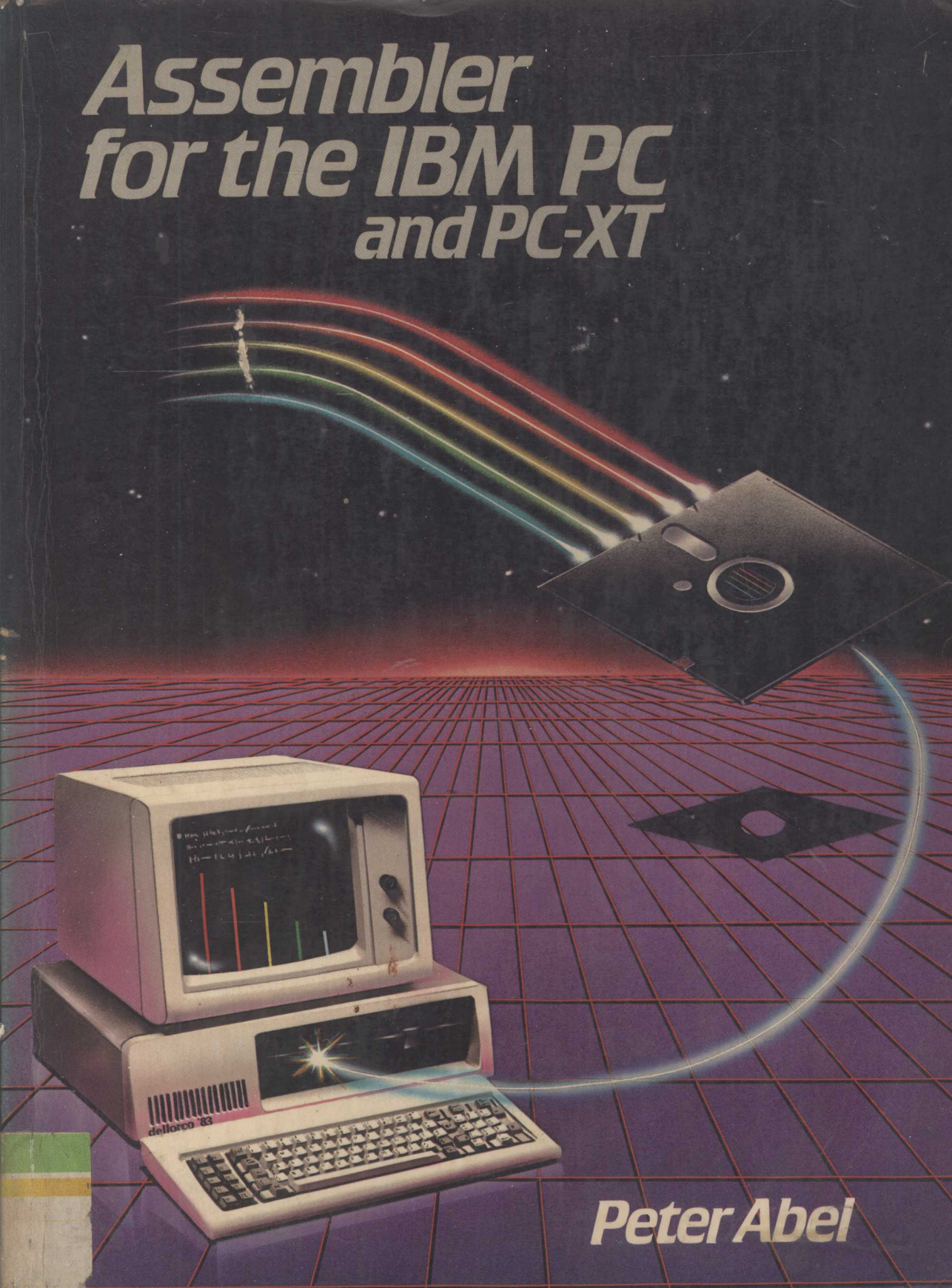
Assembler for the IBM PC and PC-XT



Peter Abel

PETER
ABEL

ASSEMBLER FOR THE IBM PC AND PC-XT

a *Reston Computer Group* book
Reston Publishing Company, Inc.
a *Prentice-Hall* company
Reston, Virginia

Library of Congress Cataloging in Publication Data

Abel, Peter
Assembler for the IBM/PC and PC-XT.

"A Reston Computer Group book."
1. IBM Personal Computer–Programming. 2. IBM Personal Computer XT–Programming. 3. Assembler language (Computer program language) I. Title. II. Title: Assembler for the I.B.M./P.C. and P.C.-X.T.
QA76.8.I2594A23 1983 001.64'2 83-16057
ISBN 0-8359-0153-X (pbk)
0-8359-0110-6 (case)

A Prentice Hall Company
Reston, Virginia 22090

10 9 8 7 6 5 4 3 2

Interior design and production: Jack Zibulsky

The final typeset pages composed in Melior were produced on a TyXSET 1000 system in Reston, Virginia, using a Mergenthaler Omnitech/2100. The page proofs were produced using a Canon LBP-10 Laser Printer. TyXSET 1000 is a trademark of TyX Corp.

Printed in the United States of America

ASSEMBLER FOR THE IBM PC AND PC-XT

PREFACE

The origin of the microprocessor goes back to the 1960s when research designers devised the integrated circuit (IC). The designers combined various electronic components (such as capacitors and transistors) into a single component on a silicon "chip." The manufacturers set this tiny chip into a device resembling a centipede, and connected it into a functioning system. As the technology advanced in the early 1970's, designers combined components onto a single chip. The introduction of the Intel 8008 chip in a computer terminal ushered in the first generation of microprocessors.

By 1974, the 8008 had evolved into a second generation microprocessor, the 8080, offering general-purpose utility. Its success prompted other companies to manufacture 8080 processors or variations such as the Zilog Z-80.

In 1978, Intel produced the third generation of microprocessors, the 8086 processor, which not only provided some compatibility with the 8080 but also significantly advanced the design. The 8088 processor was developed as a variation of the 8086 to provide a slightly simpler design and compatibility with current input/output devices. It is the 8088 that IBM® selected in 1981 as the processor for its *personal computer*, and in 1983 for the PC-XT.®

At that time, two or three manufacturers dominated the market for microcomputers. Who would have expected that the introduction of yet another "micro" to this market would have such immediate success? Perhaps the name *IBM* has had much to do with this success, and yet the IBM Personal Computer® is a powerful, effective device. Within a year, a number of other manufacturers provided similar microcomputers that share compatiblity with the IBM PC, no doubt as an effort to share in this success. These manufacturers include Commodore, Corona, DEC, Grid, Northstar, Texas Instruments, Toshiba, Vector Graphic, Victor, Wang, and Zenith.

Intel has been further developing its microprocessors. Two coprocessors that can operate with the 8086/8088 are the 8087 Numeric Data Processor (for high-speed, high-precision scientific computations, and the 8089 Input/Output Processor (for interleaved input/output operations). A souped-up version of the 8086 is the Intel 186 that provides additional operations to the 8086/8088 instruction set. At a higher level, the Intel 286 features built in memory management. You may also see these processors referred to as iAPX 86 (8086), iAPX 88 (8088), iAPX 186, and iAPX 286, where iAPX means Intel Advanced Processor Architecture.

The amazing spread of microcomputers has also brought about a renewed interest in Assembler language. There are two main reasons for this interest. The first reason is practical: a program written in Assembler language typically requires less memory space and less execution time. The second reason is academic: a knowledge of Assembler language and its resulting machine code provides an understanding of machine architecture that no "high-level" language can possibly provide.

Everyone these days has heard of such high-level languages such as BASIC, COBOL, and FORTRAN. These languages were designed to eliminate the technicalities of a particular computer. An Assembler language however is designed for a specific computer, or perhaps more accurately, *microprocessor*. As a consequence, in order to write a program in Assembler language for your own computer, you have to know something about the computer's architecture. But don't be alarmed! This book supplies all the basic material that is required. Those requiring even more advanced material can refer to the IBM Personal Computer Technical Reference Manual.

Among the material and knowledge required for this topic are the following:

- Access to an IBM Personal Computer or an equivalent microcomputer with compatible 8086 or 8088 architecture. The computer should have a minimum of 64K memory, and at least one diskette drive. Nice to have but not essential would be an 80-column screen, an additional 32K memory, and a second diskette drive.

- Familiarity with the IBM Guide to Operations Manual, especially with the description of the keyboard.
- A diskette containing the Assembler language translator.

Knowledge not required for this topic is as follows:

- A programming language. Although such knowledge may help you grasp some programming concepts more readily, it is by no means essential.
- Prior knowledge of electronics or circuitry. This book provides the necessary information about the Intel 8086/8088 PC architecture that you will require for Assembler programming.

What you can do once you have completed this book:

- Understand the hardware of the Personal Computer.
- Understand machine language code and hexadecimal format.
- Write programs in Assembler language to handle the screen, perform arithmetic, convert between ASCII and binary formats, perform table searches and sorts, and perform disk input and output.
- Trace machine execution as an aid in debugging.
- Write your own macro-instructions.
- Link together separately assembled programs.
- Understand the steps involved in assembly, link, and execute.

OPERATING SYSTEMS

The two major operating systems available on the IBM PC and other similar 8088-based microcomputers are MS-DOS from MicroSoft (known as PC-DOS on the IBM PC) and CP/M-86 from Digital Research. Both systems have their merits and adherents. Since DOS came with the system that I and most of my acquaintances bought, I have used that system as the primary vehicle for this book.

The Assembler instruction sets for both systems are virtually identical. However, there are three main areas in which DOS and CP/M differ:

1. Operating system commands such as "assemble" a program and "link" a program, and the support programs for linking and debugging.
2. The special commands to the Assembler program that define special features for the language.
3. The interface between the Assembler program and the input/output system.

I have tried to point out where the two operating systems differ. But since the suppliers are continually upgrading their versions, you should consider their manuals as the final authority.

THE APPROACH TO TAKE

This book is intended as both a tutorial and as a permanent reference. To make the most effective use of your investment in a microcomputer and software, your best approach is to work through each chapter carefully, and reread any material that is not immediately clear. Key in the example programs, assemble them, and use the DOS DEBUG program (or CP/M DDT-86) to trace execution. Work through the exercises that each chapter provides.

The first six chapters furnish the foundation material for the book and indeed for the Assembler language. After these chapters, you can begin with any of these chapters: 8, 9, 10, 12, 13, or 15. Related chapters are 6/7, 10/11, and 13/14. Chapters 18, 19, and 20 are intended as reference.

Learning Assembler and getting your programs to work is an exciting experience. You'll spend a lot of time and effort, but the rewards are sure to be great. Good luck in your efforts!

OTHER REFERENCES

Although this book is intended to stand alone, some readers may want to investigate machine language and 8086/8088 system architecture in more detail. The following books are recommended:

International Business Machines Corporation, Macro Assembler Manual, IBM Corp., Personal Computer, P.O. Box 1328, Boca Raton, Florida 33432, 1981. Provides a description of each instruction and a list of Assembler error messages. The Preface says that the manual "is a reference for experienced assembler programmers, like yourself (sic), who use the IBM Personal Computer MACRO Assembler."

_____, *Technical Reference Manual*, 1982. A stack of material on the IBM PC hardware and input/output devices. A useful reference for those inclined towards technical material.

Rector, Russell and George Alexy, *The 8086 Book*, Osborne/McGraw-Hill, Berkeley, CA, 1980. Provides material on the execution logic of each instruction and a detailed multiprocessor description.

Morse, Stephen P., *The 8086 Primer*, Hayden Book Company, Inc., Rochelle Park, New Jersey, 1980. The author was involved in the design of the 8086 and presents very readable material on why certain operations work the way they do, plus machine organization and 8086 system design.

Osborne, Adam, *An Introduction to Microcomputers*, Volume I, Osborne/McGraw-Hill, 1980. An excellent, readable introduction to the subject.

ACKNOWLEDGEMENTS

The author is grateful for the assistance and cooperation of all those who contributed suggestions and reviews, especially Sean Nelson of Easyware System Builders for technical advice. Thanks also to IBM for permission to reproduce with modifications Table B-1 in Appendix B from a publication copyrighted in 1977 by International Business Machines as IBM form number GC20-1684.

CONTENTS

PREFACE, xi

ACKNOWLEDGEMENTS, xv

CHAPTER

1 **INTRODUCTION TO THE IBM PERSONAL COMPUTER, 1**
Introduction, 1
Bits'n'Bytes, 2
ASCII Code, 2
Binary Numbers, 3
Hexadecimal Representation, 6
Segments, 8
Registers, 9
PC Architecture, 13
Key Points to Remember, 17
Questions, 18

2 **MACHINE EXECUTION, 19**
Introduction, 19
Machine Language Example I: Immediate Data, 20
CP/M Differences, 24
Machine Language Example II: Defined Data, 25
Machine Addressing, 28

Machine Language Example III: Memory Size Determination, 30
Key Points to Remember, 31
Questions, 32

3 **ASSEMBLY LANGUAGE REQUIREMENTS, 35**
Introduction, 35
Assembler Comments, 36
Coding Format, 36
Pseudo-operations, 38
Listing Pseudo-operations: PAGE and TITLE, 38
SEGMENT Pseudo-operation, 39
PROC Pseudo-operation, 40
ASSUME Pseudo-operation, 41
END Pseudo-operation, 41
Program Initialization, 41
Example Source Program I, 43
Keying in the Program, 44
Preparing a Program for Execution, 45
Assembling the Program, 46
Linking the Program, 50
Executing the Program, 51
Example Source Program II, 53
Cross-Reference File, 56
CP/M Differences, 56
Key Points to Remember, 58
Questions, 60

4 **DATA DEFINTION, 63**
Introduction, 63
Data Definition Pseudo-operation, 64
Define Byte-DB, 66
Define Word-DW, 66
Define Doubleword-DD, 68
Define Quadword-DQ, 69
Define Tenbytes-DT, 69
Immediate Operands, 70
EQU Pseudo-operation, 71
CP/M-86 Differences, 72
Key Points to Remember, 74
Questions, 74

5 **PROGRAM LOGIC, 77**
Introduction, 77
The Unconditional Jump: JMP, 78
The LOOP Instruction, 80
Flags Register, 81
Conditional Jump Instructions, 82
CALL and Procedures, 85
Stack Segment, 87
Program: Extended Move Operations, 89
Boolean Operations: AND, OR, XOR, TEST, 90
Program: Changing Lowercase to Uppercase, 92

Shifting and Rotating, 92
CP/M-86 Differences, 95
Program Organization, 95
Key Points to Remember, 97
Questions, 97

6 **SCREEN PROCESSING I: BASIC FEATURES, 99**
Introduction, 99
The Interrupt Instruction-INT, 100
Setting the Cursor, 100
Clearing the Screen, 101
Displaying on the Screen, 102
Program: Displaying the ASCII Character Set, 102
Accepting Input from the Keyboard, 104
Program: Accepting and Displaying Names, 108
CP/M-86 Differences, 110
Key Points to Remember, 111
Questions, 112

7 **SCREEN PROCESSING II: ADVANCED FEATURES, 113**
Introduction, 113
Attribute Byte, 114
BIOS Interrupt, 10,115
Program: Blinking, Reverse Video, and Scrolling, 119
Comments on Displaying, 122
Color/Graphics, 123
Text (Alphanumeric) Mode, 124
Graphics Mode, 125
Medium-Resolution Mode, 126
Key Points to Remember, 128
Questions, 129

8 **PRINTING, 131**
Introduction, 131
Print Control Characters, 132
Printing Using DOS INT 21H, 133
Program: Printing With Page Overflow and Headings, 134
Printing Using BIOS INT 17H, 138
Program: Printing with BIOS, 139
Key Points to Remember, 141
Questions, 142

9 **STRING INSTRUCTIONS, 143**
Introduction, 143
Features of String Operations, 143
REP: Repeat String Prefix, 144
MOVS: Move String, 145
LODS: Load String, 146
STOS: Store String, 148
CMPS: Compare String, 149
SCAS: Scan String, 150
Scan and Replace, 150

Alternate Coding, 151
Duplicating a Pattern, 151
Program: Right Adjusting the Screen, 153
Key Points to Remember, 155
Questions, 155

10 **ARITHMETIC I: PROCESSING BINARY DATA, 157**
Introduction, 157
Addition and Subtraction, 158
Unsigned and Signed Data, 162
Multiplication, 163
Shifting the DX:AX Registers, 170
Division, 170
Intel 8087 Numeric Data Processor, 175
Key Points to Remember, 176
Questions, 176

11 **ARITHMETIC II: ASCII AND BCD DATA, 179**
Introduction, 179
ASCII Format, 180
Binary Coded Decimal (BCD) Format, 185
Conversion of ASCII to Binary Format, 189
Conversion of Binary to ASCII Format, 191
Shifting and Rounding, 192
Program: Converting Hours and Rate for Calculating Wage, 193
Key Points to Remember, 198
Questions, 199

12 **TABLE PROCESSING, 201**
Introduction, 201
Defining Tables, 202
Direct Table Accessing, 202
Table Searching, 204
Table Searching Using String Compares, 208
The Translate (XLAT) Instruction, 210
Program: Displaying Hex and ASCII, 212
Program: Sorting Table Entries, 214
TYPE, LENGTH, and SIZE, 218
Key Points to Remember, 218
Questions, 219

13 **DISK PROCESSING I: INTRODUCTION, 221**
Introduction, 221
The Directory, 224
File Control Block: FCB, 225
Creating a Disk File Under DOS, 227
Program: Creating a Disk File, 228
Sequential Reading of a DOS Disk File, 232
Program: Reading a Disk File, 234
File Allocation Table (FAT), 237
DOS Files, 238
Absolute Disk I/O Under DOS, 242

BIOS Disk I/O, 243
Program Example Using BIOS, 245
Summary, 248
Key Points to Remember, 249
Questions, 250

14 **DISK PROCESSING II: ADVANCED, 251**
Introduction, 251
Random Reading, 252
Random Writing, 253
Program: Reading a Disk File Randomly, 254
Random Block Processing, 257
Program: Reading a Random Block, 259
Indexes, 261
Program: Disk Processing Using an Index, 263
Key Points to Remember, 266
Questions, 267

15 **MACRO WRITING, 269**
Introduction, 269
A Simple Macro Definition, 270
Use of Parameters in Macros, 272
Comments, 274
The LOCAL Pseudo-op, 276
Includes from a Macro Library, 276
Concatenation (&), 280
Repetition: REPT, IRP, and IRPC, 280
Conditional Pseudo-operations, 282
The EXITM Pseudo-op, 283
Macro Using IF and IFNDEF Conditions, 284
Macro Using IFIDN Condition, 286
Key Points to Remember, 288
Questions, 288

16 **LINKING TO SUBPROGRAMS, 291**
Introduction, 291
Intersegment Calls, 293
The EXTRN and PUBLIC Attributes, 294
Program: Use of EXTRN and PUBLIC for a Label, 295
Program: Use of PUBLIC in the Code Segment, 298
Program: Common Data in Subprograms, 301
Passing Parameters, 304
Linking BASIC and Assembler, 307
Linking PASCAL to Assembler, 312
Key Points to Remember, 315
Questions, 316

17 **BIOS: BASIC INPUT/OUTPUT SYSTEM, 317**
Introduction, 317
BIOS Interrupts, 319
DOS Interrupts, 323
Ports, 325
Generating Sound, 326

18 **ASSEMBLER PSEUDO-OP REFERENCE, 329**
Introduction, 329
Indexed Memory, 329
Assembler Operators, 330
LENGTH, 331
OFFSET, 331
PTR, 331
SEG, 332
SHORT, 332
SIZE, 333
TYPE, 333
Assembler Pseudo-Operations, 334
ASSUME, 334
EXTRN, 334
GROUP, 335
INCLUDE, 335
LABEL, 336
NAME, 336
ORG, 337
PROC, 338
PUBLIC, 338
RECORD, 339
WIDTH, 340
MASK, 342
SEGMENT, 342
STRUC, 344

19 **DOS PROGRAM LOADER, 349**
Introduction, 349
COMMAND.COM, 350
Program Segment Prefix, 352
Loading a Program, 352
Example Executable Program, 354

20 **INSTRUCTION REFERENCE, 359**
Introduction, 359
Register Notation, 360
Addressing Mode Byte, 361
Example Two-byte Instructions, 362
Example Three-byte Instructions, 363
Example Four-byte Instructions, 363
Instructions in Alphabetical/Sequence, 364

SOLUTIONS TO SELECTED QUESTIONS, 397

APPENDIXES

A **ASCII CHARACTER CODES, 391**

B **HEXADECIMAL/DECIMAL CONVERSION, 393**

INDEX, 409

1

INTRODUCTION TO THE IBM PERSONAL COMPUTER

Objective:
To explain features of microcomputer hardware and program organization for the Assembler programmer.

INTRODUCTION

If you have not digested the material in the IBM Guide to Operations, then now is the time to do so. Otherwise, let's get on with the project! There are some fundamentals that you must master before progressing to Chapter 2. This material involves the organization of the computer system. The fundamental building blocks of a computer are the *bit* and the *byte*. These supply the means by which your computer can represent data and instructions in memory.

A program in machine code consists of different *Segments* for defining data, for machine instructions, and a Segment named the *Stack* that contains stored addresses. To handle arithmetic, data movement, and addressing, the computer has a number of *registers*. This chapter covers all this material so that you can get going right away in Chapter 2 on your first machine language program.

BITS'N'BYTES

The smallest unit of data in a computer is a *bit*. A bit may be magnetized as *off* so that its value is zero, or as *on* so that its value is one. A single bit doesn't provide much information, but it is surprising what a bunch of them can do!

A group of nine bits represents a *byte*, eight bits for data and one bit for "parity." The eight bits provide the basis for representing characters such as the letter "A" and the asterisk, and for binary arithmetic. For example, a representation of the on and off bits for the letter "A" is 01000001 and for the asterisk is 00101010 (you don't have to memorize such facts).

Note on parity: The parity bit assumes that the "on" bits for a byte are always an odd number. The parity bit for the letter "A" would be on and for the asterisk would be off. When an instruction references a byte in storage, the computer checks its parity. If parity is even, a bit is assumed to be "lost" and the system displays an error message. A parity error may be a result of a hardware fault or it may be nonrecurring; either way, it is a rare event. Thankfully, this is all you need to know about parity.

You may have wondered how a computer "knows" that a bit value 01000001 represents the letter "A". When you key in an "A" on the keyboard, the system accepts a signal from that particular key into a byte in memory that sets the bits to 01000001. You can move this byte about in memory as you will, and that particular value when sent to the screen or printer generates the letter "A".

For reference purposes, the bits in a byte are numbered 0 to 7 from right to left as shown for the letter "A" below:

```
Bit number:   7 6 5 4 3 2 1 0
Bit contents: 0 1 0 0 0 0 0 1
```

The number 2^{10} equals 1024, which happens to be the value "K". For example, a computer with 64K memory has 64 × 1024 bytes, or 65,536.

Since the 8088 processor in the IBM PC uses 16-bit architecture, it can access 16-bit values in both memory and its registers. A 16-bit (2-byte) field is known as a *word*. The bits in a word are numbered 0 through 15 from right to left as shown for the letters "PC" below:

```
Bit number:   15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0
Bit contents:  0  1  0  1  0  0 0 0 | 0 1 0 0 0 0 1 1
```

ASCII CODE

The eight data bits enable 2^8 (256) possible combinations, from all bits off, 00000000, through all bits on, 11111111. There is no requirement that