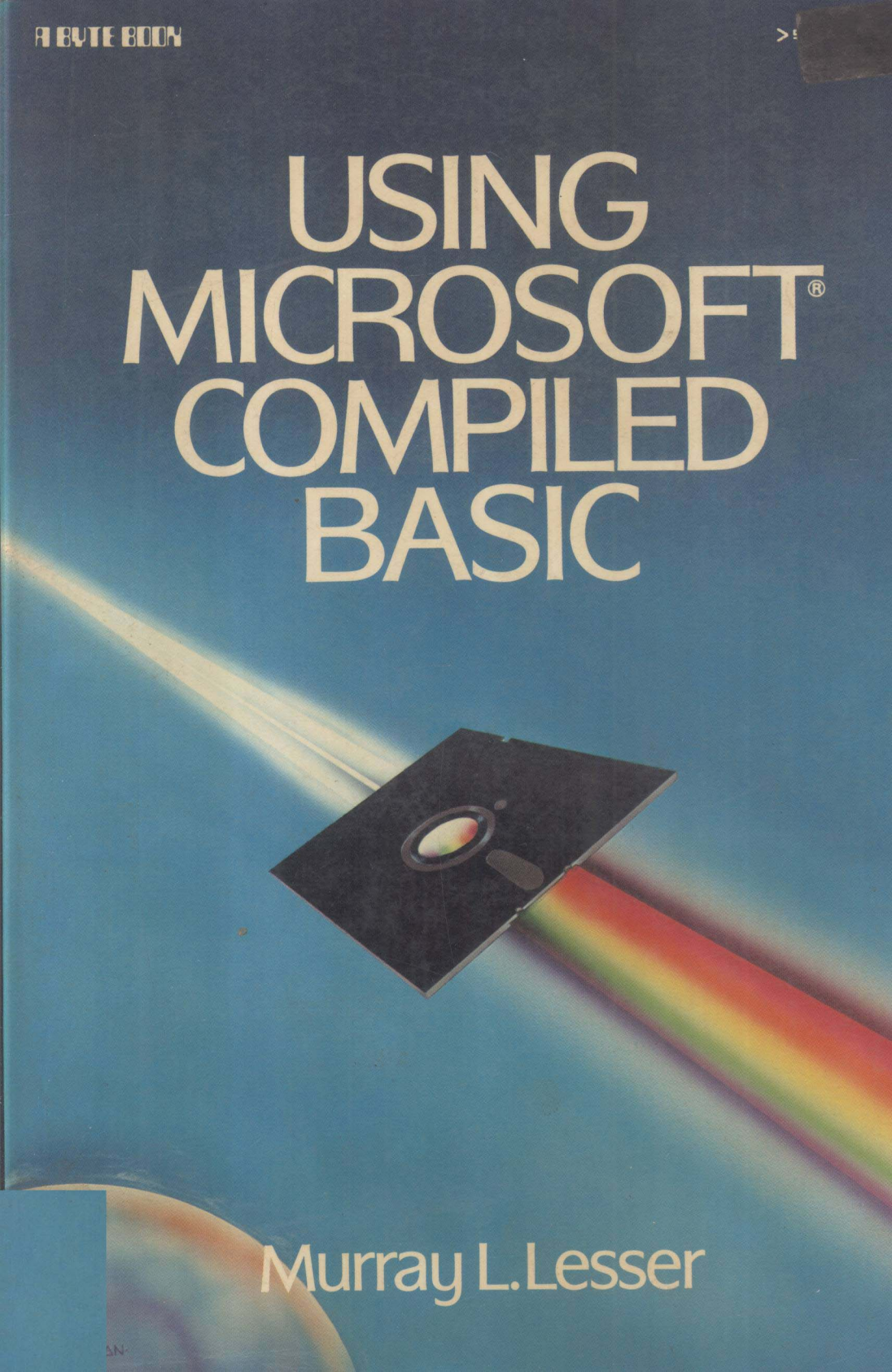
A BYTE BOOK

USING MICROSOFT® COMPILED BASIC



Murray L. Lesser

Using Microsoft® Compiled BASIC

Murray L. Lesser

McGraw-Hill Book Company

New York / St. Louis / San Francisco / Auckland
Bogotá / Hamburg / Johannesburg / London / Madrid
Mexico / Montreal / New Delhi / Panama / Paris
São Paulo / Singapore / Sydney / Tokyo / Toronto

Library of Congress Cataloging in Publication Data

Lesser, Murray L.
Using Microsoft Compiled BASIC.

Bibliography: p.
Includes index.
1.Basic (Computer program language) 2.CP/M-80 (Computer operating system) I.Title. II.Title: Using Microsoft Compiled B.A.S.I.C.
QA76.73.B3L46 1985 001.64 84-3898
ISBN 0-07-037302-7

1234567890 DOC/DOC 8987654

ISBN 0-07-037302-7

The editors for this book were Stephen G. Guty and Esther Gelatt, the designer was Mark E. Safran, and the production supervisor was Thomas G. Kowalczyk. It was set in Century Schoolbook by University Graphics, Inc.

Printed and bound by R. R. Donnelley & Sons Company.

Using Microsoft® Compiled BASIC

For Jean

Preface

When I bought my first microcomputer in 1979, the only software included in the price was the manufacturer's version of CP/M 1.4 (since replaced by version 2.2). As I intended to use the machine for writing and record keeping, I purchased the "word-processing" program recommended by my dealer (since abandoned by both of us because the publisher refused to fix the bugs) and an early version of the Microsoft BASIC Compiler.

Although I had been associated with the use or design of digital computers for 30 years, I hadn't programmed professionally since 1954 and had never used a compiler or macro assembler. So, I spent the next few years learning to use both properly.

A little was learned by carefully reading between the lines of the manuals and the few pertinent articles in the small-computer magazines. Most of what I learned, I learned from running experiments on the system—trying things and seeing what would happen—and generalizing from the results.

The most important thing I learned was not in the literature: The best way to use the compiler is to take advantage of the *differences* (rather than the similarities) between the compiled and interpreted versions of Microsoft's BASIC-80. Unlike the interpreted version, compiled BASIC is an excellent language in which to write well-structured, high-performing, useful data-processing programs that are easy to read, easy to debug, and easy to modify later.

This is the book I wish had been available when I began using the BASIC compiler. Since there isn't room for everything, I have concentrated on the most important aspects of using compiled BASIC under CP/M. I have omitted details (both of the language and of CP/M) that are explained fairly clearly in the manuals or are of very limited interest. If you can't find what you need in this text, read the appropriate manual. If it isn't there, you will have to run your own experiments.

I wish to thank *Byte* magazine, the Microsoft Corporation, and Digital Research, Inc., for permission to reprint portions of their copyrighted materials. Neither Microsoft nor Digital Research has reviewed the context in which I have used their material, and neither is responsible for any errors I may have committed.

My litigious friend tells me I should warn you that although permission is freely granted for noncommercial use of the illustrative programs listed herein, you use them at your own risk. Neither the author nor the publisher makes any warranties concerning the program listings and neither assumes any responsibility or liability of any kind for errors or for the consequences of any such errors.

MURRAY L. LESSER

Contents

1

Introduction

There are many books for beginners in computer programming. This is not one of them. It is an intermediate text dealing with writing programs with BASCOM (Microsoft's compiled BASIC) using some of the innards of CP/M. It tells you how to make temporary minor modifications to your CP/M system, under program control, so you can do things you would like to do but can't under the normal CP/M interface protocols. But it won't tell you how to make major modifications to your system or how to get rich by writing a best-seller program. Rather, it describes a set of techniques to make the programs you write for your own use easier to write, easier to debug, easier to use, and easier to modify when the urge arises.

About This Book

This book is for people who have some background in programming with some version of BASIC but not enough to have developed too many bad habits. It helps if you can write and run simple assembly-language programs under CP/M, as one of the strengths of BASCOM lies in the ability to extend the language easily by adding called assembly-language subroutines. If you haven't had any such experience, you may find this book hard going, but you can manage with the aid of some elementary texts I will list later.

In writing the book, I have assumed you have some intellectual curiosity and are not interested solely in a *how-to* book that doesn't contain some *why-to* as well. I, for one, can't remember a set of complicated rules without a structure to hang them on.

For programming, the structure is a conceptual model of the

entire system as seen by the programmer, which includes the operating system interface and even—to some extent—the hardware. Thus, there are several tutorials giving my conceptual model of the important parts of the structure of BASCOM and CP/M, with examples of BASCOM and assembly-language programs I have found useful in my own programming activities. This model is highly oversimplified and may be wrong in some details. However, I have found it sufficient unto the need.

I haven't written a programming manual for BASCOM and CP/M. I am assuming that you have at least skimmed over the manuals that came with your distribution diskettes; I don't intend to repeat much of the detail. In the long run, those manuals will become very familiar to you, and you will figure out their idiosyncrasies as you practice using the language and the system. However, I will try to clarify some of the more obscure points in them and correct the errors I know about.

Since it is impossible to discuss the various facets of using BASCOM and CP/M in a completely sequential manner, there will be occasional real or implied forward references—at least to the extent that I will sometimes use programming constructs that won't be discussed until later in the book. Please be assured that when I do so, the programming manual references to those constructs are correct. You should refer to the applicable manual if you need more information to understand the usage.

Although this book deals largely with developing useful BASCOM programs running under CP/M, much of the material is applicable (with modification) to programs written in MBASIC (the interpreter version of BASIC-80). There is great emphasis on assembly-language subroutines, which are easily called by programs written in BASCOM. Such subroutines can also be called (but not as easily) from MBASIC, and most of the subroutines given in this book can be modified for that use. Of course, there is no way to get BASCOM's performance, program integrity, or program readability when using MBASIC.

Conventions Used in the Text

I dislike rewriting computer history to fit modern typographic styling. Thus, language and proprietary program names whose inventors wrote them in all caps are left in their original form (e.g., FORTRAN, BASIC). Names that were not so written by their inventors are given as proper nouns (e.g., Pascal, Ratfor).

When referring to the various aliases of Microsoft's BASIC-80 version 5, I will use *BASIC-80* for the general case, *BASCOM* if the reference holds only for the compiler version, and *MBASIC* if I mean the interpreter version. Plain *BASIC* will refer to the language in general, while

Basic means what it means in English (e.g., CP/M's Basic Disk Operating System has nothing to do with the BASIC programming language).

Generic terms used in the programming art, particularly those associated with file controls, are given normal lower-case spelling (e.g., open, close). However, commands in BASIC-80 that may be spelled the same but have content well beyond the generic meaning are given in all caps (e.g., OPEN, CLOSE).

Since much of the material deals with subroutines written in assembly language and called by BASIC-80 programs, I will use the term *called subroutines* for these, to distinguish them from *subroutines* written in BASIC-80 and reached through the GOSUB command.

Superscript numbers in the text refer to notes at the end of each chapter.[1] Listings are found at the end of the chapter in which they are cited, following the "Notes" section.

The Art of Programming

Programming is not a science, not even a branch of mathematics. Programming is an intellectual art form somewhat akin to writing essays. The major requirements in either case are to think in logical progression and to be able to convey your thoughts, unambiguously, in a written language. Fortunately, it is easier to convey unambiguous thoughts to machines than it is to people.

If you didn't think the problem through completely and unambiguously, the program will have *bugs* in it. When a running computer runs into a bug, the results are likely to be highly surprising; the machine does what you told it to do not what you meant it to do.

Since a large computer program is probably the most complex structure made by human beings, it is practically impossible to guarantee that such a program is bug-free. Thus, almost all proprietary software has bugs in it. Most of your and my programs that cannot be listed on one page also contain bugs. This is the reason for structured programming: To attempt to reduce a complex program into many simple programs that can be written and tested independently and then linked together to make up the final program.[2]

The obvious bugs in proprietary programs were removed by conscientious testing before putting the software into service. Successful testing requires making test runs at what Kernighan and Plauger call the "boundaries . . . of program operations,"[3] where there are changes in control due to a change in variable.

But some bugs are very subtle; they may be value-dependent in a not obvious manner. These show up rarely and are found only when lots of

users have used a program lots of times. When enough users have found and reported serious bugs, the publisher of the program permits the programmers to put out an update.[4]

One hallmark of a good program is that the listings are intelligible to human readers. The following factors lead to readability:

- The program is organized properly (structured).
- The terms used in the language itself remind the reader of their operation.
- The variable names are properly chosen to have meaning.
- The program listing is neatly arranged on the sheet of paper.
- The program is properly commented.

Six months after you have written the program, you will want to revise it. Unless you can read it, you will never be able to remember how that clever programming works! I am not going into great detail on how to write programs for human consumption. You will have to learn from example.

Another hallmark of a good computer program is that the input and output formats are arranged for the convenience of the users of the information not for the convenience of the programmer. This is the most important factor to keep in mind when writing a program, since the only purpose served by a computer program is to change information from one format to another.[5] The input format (along with information contained in internal intermediate files) is changed to the desired format for output, and the internal files are updated accordingly. Since much of this format manipulation is string handling, BASIC-80 is a very convenient language for most data-processing applications. It contains a plethora of string-handling and input/output commands and functions.[6]

Programming Manuals

All proprietary (purchased) software comes with some form of documentation, commonly referred to as the manual. Programming manuals vary from a single page to an elaborate book in a fancy binder. All suffer from the same disease. They give the information somebody thought you wanted in order to run the software; they do not give the information you need. One reason for this sad state of affairs is that individual needs vary. If you are a beginner, you want a manual that goes into excruciating detail about how to use the program. As you gain experience, you prefer what programmers call a reference manual.

A good manual is well supplied with listings showing usage examples. Learn to read and understand those listings; sometimes they contain more information than does the text.

Besides being excellent examples of poor exposition, most programming manuals contain erroneous vestiges left over from earlier editions of either the program or the manual. Sometimes program changes are running ahead of manual updates. The easiest way to resolve differences between your understanding of the manual and the way the program actually works is by direct experimentation. It is also the easiest way to learn how to use the program or language.

Hardware/Software Considerations

According to the Microsoft manuals, BASCOM will run in a usable memory as small as 48K. But more memory will allow the BASCOM compiler to do all the optimizing of which it is capable.

My microcomputer is a 64K Z-80 system with two 8-inch diskette drives. All programs listed have been tested under my hardware vendor's version of CP/M 2.2 and (unless otherwise stated) his version of CP/M 1.4. However, not all CP/Ms are created equal, and some of the programs may give strange results on your system. If this happens to you, you will have to run your own experiments to find and fix the bug—whether it is mine or is in your version of CP/M. At least you will have the listings for mine.

My terminal is a Perkin-Elmer 550, and my printer is a Qume Sprint 5. I have carefully segregated the peripheral-hardware-dependent programming into called subroutines. Your first task will be to rewrite them to fit your own configuration.

Almost all assembly-language listings are in Z-80 mnemonics. The exceptions are a few 8080 macros that simulate needed Z-80 convenience functions—for those of you who are using an 8080 or 8085 system. If you own a Z-80 system and are still programming in 8080 mnemonics, you are working too hard.

You cannot link assembly-language programs to compiled BASCOM programs unless you use an assembler that produces Microsoft *.REL* files. Thus, you cannot use Digital Research's ASM or MAC assemblers. You might as well use the Microsoft MACRO-80 assembler that came bundled in with your compiler, along with Microsoft's linker (LINK-80).

Remember that CP/M is an 8080 operating system, even if you are running it on Z-80 hardware. Hence, the default mode of MACRO-80 running under CP/M is 8080 mnemonics. You must tell the assembler, each time, if you want it to understand Z-80.