William Ralph Bennett, Jr.

Introduction to computer applications for non-science students (BASIC)

ABCDEFGHIJKLMNOPQRSTUVWXYZ Space

TP39
B472

William Ralph Bennett, Jr.

*Charles Baldwin Sawyer Professor
of Engineering and Applied Science,
and Professor of Physics
Yale University*

Introduction to computer applications for non-science students (BASIC)

Prentice-Hall, Inc. Englewood Cliffs, New Jersey

Library of Congress Cataloging in Publication Data

BENNETT, WILLIAM RALPH, JR.
Introduction to computer applications for non-science students (BASIC).

Includes bibliographies and index.
1. Electronic digital computers—Programming.
2. Basic (Computer program language) I. Title.
QA76.6.B44 001.6'42 75-30918
ISBN 0-13-480061-3

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA, PTY. LIMITED, *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PRIVATE LIMITED, *Singapore*

To Fran

Preface

Three years ago I was asked by the chairman of the Engineering and Applied Science Department at Yale University to develop an introductory computer-applications course that would have broad appeal to students in both the humanities and physical sciences. The present volume, which emphasizes material of general interest, is one result. A one-term course incorporating the present material has been given for three years under the title "The Computer as a Research Tool" and has been taken by students ranging from freshman English majors to graduate students in chemistry.† Much to the author's surprise and personal gratification, the course made the "Ten Best" list twice at Yale during this period.‡ Most students taking the course had had (or were taking concurrently) at least one term of calculus. Beyond that, there was no real common denominator. It should be emphasized, however, that a prior knowledge of calculus is not a prerequisite for the present material and that nearly everything in the present book should be fully comprehensible to students with a mathematics background consisting only of high school algebra and trigonometry. The more difficult mathematical sections occur near the ends of Chapters 2 and 3, are clearly marked, and may easily be omitted. Both programming and conceptual difficulties increase gradually within chapters and from one chapter to the next. Chapters 1 through 3 are intended as introductory background material for the main part of the course, which is contained in Chapter 4.

† The course at Yale also contained material on dynamics, random processes, Fourier series, and electronics that is available (together with the present chapters) in a more complete book by the author, *Scientific and Engineering Problem-Solving with the Computer*, (published by Prentice-Hall, Inc.). Of necessity, the Yale course covered the present material in highly abbreviated form. However, the present book can be the basis of a very substantial introductory one-term course for humanities students.

‡ *Yale Course Critique* (published by the *Yale Daily News*, New Haven, Conn.): 1974 edition, pp. 7. 43; 1975 edition pp. 13, 71.

Although the chapters are labeled according to subject matter, they are also organized according to programming technique:

Chapter	Techniques Emphasized
1	Elementary programming in BASIC
2	Series summation and matrix operations
3	Teletype plotting and graphic displays
4	Character coding and printing

The structure of the book is also based on the belief that the best way to teach students computational methods is to give them lots of interesting problems of gradually increasing difficulty. I chose BASIC as the main programming language because it is rapidly learned and "conversational." It is also efficient enough to work effectively within minicomputers. Students who have never touched a computer terminal can start right off doing meaningful problems in BASIC, and the frustrations associated with batch processing are totally avoided. Under these conditions, the students teach themselves to a large extent and one merely has to be available to give general guidance and help them out of occasional pitfalls.

The problems are given immediately after the relevant discussions in the text and where possible have been based on unusual events of general and social interest. An effort was also made to select nontrivial problems. Apart from introductory examples in each chapter, the remaining problems are of a type that would either be impossible or prohibitive to solve without a computer. For the same reason, most topics and problems discussed in the present book are rarely treated quantitatively in the normal curriculum.

An attempt has been made to try to choose material that would alternate in appeal between humanities and science students. Although it is very hard to please both groups simultaneously, one can play a game in which important methods in science are applied to problems of social importance. Also, the author has tried throughout the manuscipt to provide occasional bits of comic relief. However, what sometimes seems hilariously amusing late at night does not always withstand the cold light of dawn.

Credit for the course at Yale was based entirely on the completion of assigned problems (about two or three a week) and a term research project. For the latter reason, Research Problems are occasionally suggested throughout the book with the object of stimulating further independent work on the part of the student. The standard problems are presented to illustrate what can easily be accomplished in a specific area in one afternoon through the expenditure of a few dollars in computer time (if you have to pay for it). Nearly all of them can be done on a minicomputer with a memory of only 16,000 16-bit words. However, the more meaningful ones are much too difficult for use on examinations, and the whole concept of an examination seemed pointless. I think the ability of a student to solve 30 formidable problems over a period of one term is the most important measure of his or her grasp of the subject. However, tastes vary on this point and the above approach to the grading question involves a massive investment in time spent examining program listings and problem results handed in by students. In this connection it helps to encourage the students to document their programs right from the start. One does not usually need formal flow charts, but an occasional comment in the margin of a program becomes very helpful. It is also desirable to encourage students to think through their problems carefully before going near a terminal.

There are various ways in which the material can be emphasized and presented within a one-term course, and abstracts have been included at the start of each chapter to aid in this process. Most students I have encountered had never done any real programming, and I constructed the first three chapters as an introductory technical background. Chapter 1 is for students without previous exposure to computers and could easily be skipped (or covered very rapidly) by students with prior computing experience. I have usually covered this material in one week of lectures and left most of it to be read by beginning students as needed and learned through the mechanism of problems assigned the first day. (An immediate assignment of four or five problems also serves the useful purpose of encouraging students who do not want to do any work to drop the course.) The material by Kemeny and Kurtz written on BASIC (see References to Chapter 1) is very useful at this point, if available. Also, it is assumed that a manual on BASIC provided by the individual computer service will be available for occasional reference. However, a primary objective was to make the present book self-contained. (It seems unfair to expect students to have to buy more than one book for a course.)

Chapter 2 covers a number of routine programming techniques and reviews some aspects of high school algebra and introductory calculus (particularly derivatives and Taylor series) which provide useful insight regarding the function statements built into the BASIC language. It is also desirable to have students start getting used to vectors and matrices as early as possible so that the mechanics of using them does not become a stumbling block in the later applications involving more difficult subject matter. For this reason, I have chosen to sneak up on the MAT commands in BASIC by first using them in the Ramanujan problem to store and manipulate data, and then to gradually introduce more complicated MAT operations in succeeding sections. By the time the student gets to the discussion of the input–output theory in economics, he should not only be used to using matrices but also have a genuine appreciation for the tremendous power of MAT commands.

Chapter 3 in its entirety is not essential to the rest of the book. However, most students love plotting things, and such techniques can provide helpful insight to the solution of difficult problems later. The chapter presents a survey of representative methods and devices that are currently available. The material on teletype plotting can be covered within about two weeks, and this material (or its equivalent, using high-resolution displays) is all that is really required for the rest of the book. What one specifically does beyond teletype plotting will be limited by available hardware. However, computer-controlled high-resolution display devices are becoming increasingly available, and it seemed clear that at least some discussion of the use of representative hardware should be included. The minicomputer owner, in particular, is in a position nowadays where such things may be implemented cheaply.

Many of the problems depend on substantial blocks of data being available in the BASIC DATA format. Because little educational benefit results from typing in these huge blocks of data by hand and needless amounts of terminal time get wasted in the process, it is desirable to make these DATA statements available to the students on punched tape or within disc files. To facilitate this process, the author will try to provide, at a reasonable cost, punched tape listings of such BASIC DATA statements in ASCII code. However, this offer is made subject to the condition that it may be terminated at any time should the process become impractical. Further, occasional errors made in the mechanical punching process will have to be corrected by the purchaser (for example, by checking against listings illustrated in the present book). For further information regarding this question, write directly to the author, Department of Engineering and Applied Science, Yale University, New Haven, Connecticut 06520.

William Ralph Bennett, Jr.

Acknowledgments

I would like to thank Professor Robert G. Wheeler of Yale University for suggesting and encouraging this project. Although the project has taken a lot more work than I initially expected, it has also been a lot of fun. I should also like to thank Professor Charles A. Walker (currently chairman of the Engineering and Applied Science Department) for his sustaining support. Obviously, the whole thing could not have been accomplished while carrying a heavy teaching load. I am particularly indebted to the Hewlett-Packard Corporation for the donation of a 2116B computer which was used in the course and to the Moore Fund at Yale for a grant to purchase additional equipment.

The following people warrant special thanks: Dr. John W. Knutson, Jr., for his assistance in presenting the course initially; William C. Campbell, Donald R. Carlin, Robert M. Fleischman, H. Thomas Hunt, and Jeffrey J. Korman for expert teaching assistance and machine-language programming help; and Marjorie G. Wynne, Edwin J. Beinecke Research Librarian at the Beinecke Rare Book and Manuscript Library for her help in tracking down several obscure references. In addition, I am indebted to all the students at Yale University who have taken the course for their patience in letting me try this material out on them, for their many suggestions, and for their permission to reproduce some of the original data quoted the text. I should also like to thank Professor Jack K. Cohen of the University of Denver (Colorado) for a careful reading of the manuscript and numerous helpful suggestions. Special thanks also go to Phyllis Springmeyer of the College Book Editorial Department at Prentice-Hall for her invaluable help.

I should also like to thank the various people and organizations specifically cited in the figure captions for their permission to reproduce figures. In addition, I would like to thank my father, William Ralph Bennett, for several helpful background discussions on communications problems. Finally, I am especially indebted to my wife, Frances Commins Bennett, for numerous helpful discussions, for editing and typing the original manuscript, and for putting up with some of the more obscure conversational topics that this manuscript has stimulated.

Contents

1 Introduction

This chapter is intended for people who have never seen, used, or touched a computer terminal. Sections 1.1–1.3 discuss background concepts and define some of the basic jargon of the computer-science world. Rudimentary programming operations in BASIC are started in Section 1.4 and gradually increase in difficulty throughout the remainder of the chapter. Problems of practical importance are emphasized in these examples. The main object of the chapter is to get the beginner's feet wet as soon as possible and to work up to a few nontrivial problems by the end of the chapter. Readers with previous experience at programming should start by having a look at the problems toward the end of the chapter. If these problems are too easy, go immediately to Chapter 2. Note that the standard commands in BASIC are listed in the index at the end of the book with page references to representative examples of their use and definitions of their meanings.

Most electronic computers count, add, subtract, multiply, divide, and do logic operations in base 2. The reason is fairly obvious: The binary numbers 0 or 1 can be very naturally represented by opening or closing a switch. In practice, computers use electronic circuits that have two stable operating conditions to represent such binary numbers. The transition from one stable point to the other is induced electronically and is analogous to changing the state of a two-position switch. This type of electronic switching can be accomplished in astonishingly short time intervals ($\approx 10^{-9}$ second at the present state of the art) and is the primary technology upon which contemporary high-speed digital computing rests.

If you have a sufficiently long row of ON/OFF switches, you can use them to represent any specified binary integer. For example, the binary number 1010 corresponds to the number

$$1\times 2^3+0\times 2^2+1\times 2^1+0\times 2^0=8+2=10$$

in base 10 and may be represented by four ON/OFF switches. The process of handling numbers that are not integers involves utilizing negative powers of 2 (i.e., those which could be stored to the right of the *binary point*) in much the same way that decimal numbers are normally handled in base 10.

Processes in binary arithmetic can be performed by simple electronic circuits in which the output voltage is a two-valued function of two separate input voltages. For example, adding two binary integers together involves applying the following rule to the successively higher, corresponding pairs of digits:

$$\begin{aligned}&0+0=0\\&0+1=1\\&1+0=1\\&1+1=0 \qquad \text{but carry 1 to the next-higher digit}\end{aligned}$$

This fundamental rule in binary addition can be accomplished with a circuit that is turned ON when either input voltage (digit) is ON separately, but which is otherwise turned to the OFF position. (People who have done counting experiments will recognize this type of circuit as an *anticoincidence circuit.*) The process of carrying 1 to the next-higher digit (where $1+1=0$) can be effected with a circuit that is normally in the OFF position, unless both inputs are ON. (People used to counting experiments will recognize the latter as a *coincidence circuit.*) The other binary arithmetic operations (subtraction, multiplication, etc.) can be performed in similar fashion.

The number of operations done in one step (or how long a given binary arithmetic operation takes) is largely a function of the complexity of the circuitry in an individual computer. For example, adding N to a given binary number can be accomplished directly or by adding 1 N times; shifting a binary number N places to the left (which is equivalent to multiplying by 2^N) can be accomplished directly in one step or by shifting by one digit N times; and so on. Hence the inherent speed of a particular computer is extremely dependent on the actual circuitry used.

The invention of binary codes and binary logic operations is very old. Francis Bacon (1561–1626) used binary codes to transmit secret diplomatic messages. Joseph Marie Jacquard (1752–1834) used binary-coded punchcards to operate looms with such success that about 11,000 of them were in use throughout France by 1812. George Boole (1815–1864) developed a mathematical theory of binary logic during the nineteenth century. Hence the mathematical background for most of the binary operations used in modern digital computing was established long before the first digital computer of any consequence had been built.

Although a mechanical desk calculator that could add, subtract, multiply, and divide had been built as early as 1623—by Wilhelm Schikard—the modern digital computer was largely a post–World War II development. Prior to that time, most of the emphasis on electronic machines had been based upon analog devices—for example, those used during World War II to permit RADAR control of antiaircraft guns.

The very first digital computers were extremely sluggish, cumbersome things in which the bistable "electronic" circuits were made up from mechanical relays. Indeed, one such machine developed at the Bell Laboratories in 1944 contained over 9000 telephone relays, covered a floor space of about 1000 square feet, and weighed about 10 tons!

One of the earliest high-speed electronic digital computers was that developed under the direction of John von Neumann at the Institute for Advanced Study in Princeton, New Jersey, during the period 1946–1952. This device contained several thousand vacuum tubes and used a memory based upon the continuous rejuvenation of arrays of binary digits which were stored electrostatically in a large bank of cathode-ray tubes (see Fig. 1-1). Because the

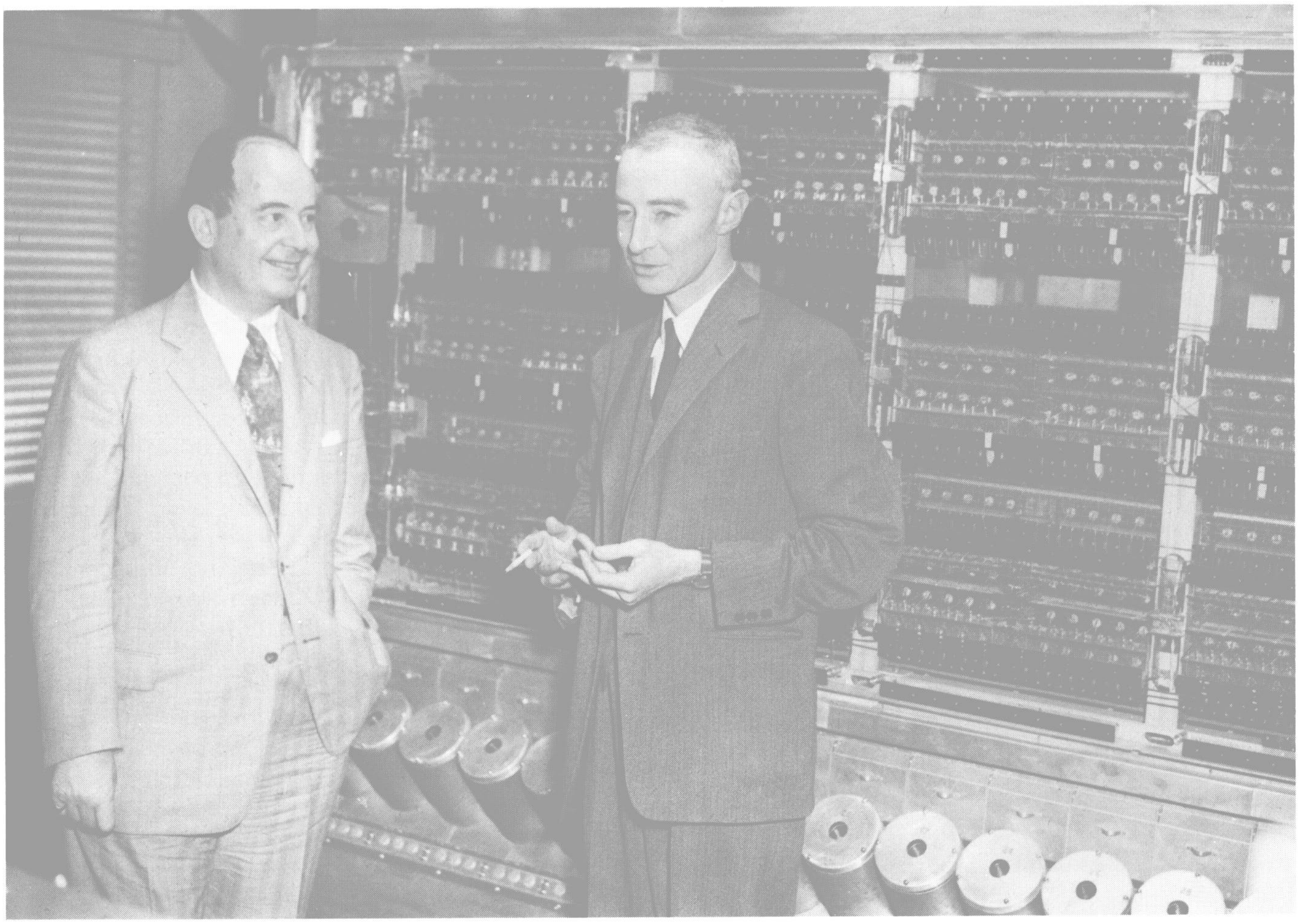

Fig. 1-1. von Neumann's computer. (John von Neumann is at the left and J. Robert Oppenheimer is at the right.) Note the bank of cathode-ray-tube storage elements across the bottom of the picture (within the cylindrical cans); these were used for the computer memory. Each tube contained a square display of $32 \times 32 = 1024$ binary storage bits that were periodically regenerated at about 1000 times per second. A section from this computer is currently on display at the Smithsonian Institution in Washington, D.C. (*Courtesy* of the Institute for Advanced Study, Princeton, New Jersey.)

lifetimes of vacuum tubes were typically about 1000 hours, the electrostatic storage technique was tricky at best, and the entire contents of the memory in the von Neumann computer had to be re-stored about 1000 times per second, it is rather amazing that the device could be made to function reliably at all over long periods of time. Nevertheless, the machine served as an effective laboratory to test many of the notions of programming and coding used in contemporary large-scale, high-speed digital computers. Prior to the design of the von Neumann machine, it was argued that decimal systems were the most appropriate for computers because of the formidable problems in decimal-to-binary conversion. One early accomplishment of the von Neumann project was the demonstration that such conversions could be accomplished with a fairly small number of machine operations (approximately 47 steps) and in time intervals of only a few milliseconds (using the circuitry of that period). The machine was also used to solve some very substantial numerical problems connected with the development of the hydrogen bomb[1]—especially problems involving the inversion of high-order matrices. [See the discussion of the von Neumann machine by Goldstine (1972). Goldstine worked on many of the fundamental mathematical and programming problems encountered with this machine, and his book contains a very interesting account of the techniques adopted, in addition to a comprehensive description of the historical background.]

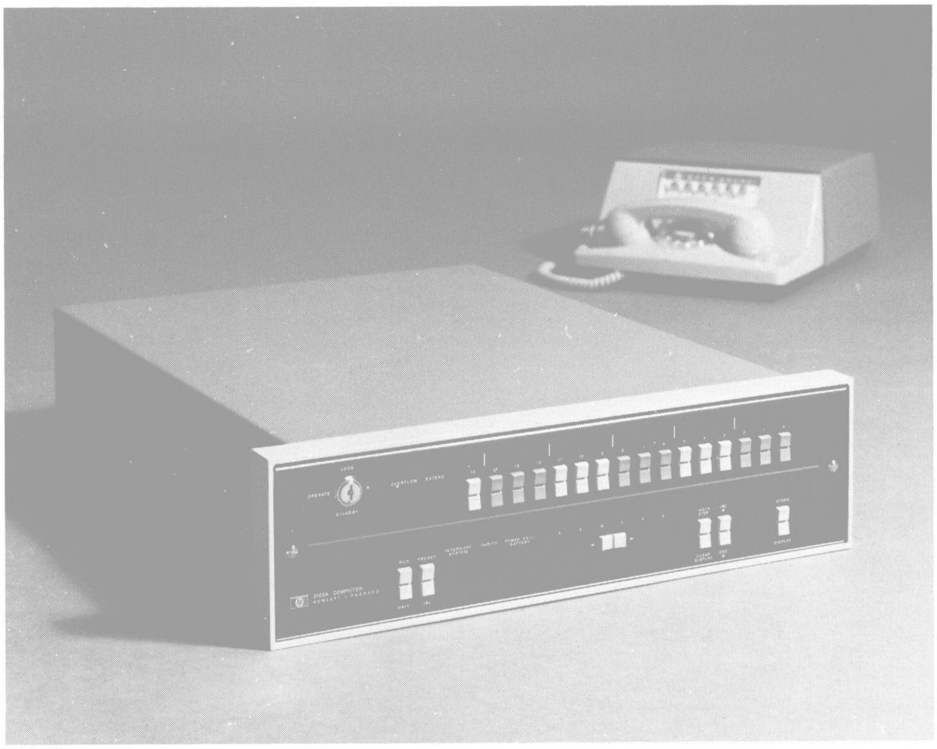

Fig. 1-2. Representative, contemporary minicomputer capable of storing up to 32,000 separate 16-bit numbers in its memory. (*Courtesy* of the Hewlett-Packard Co.)

[1] It is, in fact, rather ironic that J. Robert Oppenheimer, who was the Director of the Institute for Advanced Study during most of that period and a strong proponent of the von Neumann project, was crucified shortly thereafter for his initial stand against the development of the H-bomb. [See, for example, von Neumann's testimony in support of Oppenheimer reproduced by the U.S. Atomic Energy Commission (1971, p. 655), regarding the relevance of the von Neumann computer to the H-bomb project.]

The development and practical availability of transistors (with nearly infinite lifetimes) and reliable ferrite core memories (which do not require periodic rejuvenation) had a massive effect on the computer field during the next two decades. The exponential growth in this field has continued well into the present decade, as the effects of integrated circuits, circuit-chip technology, and semiconductor memories have become felt. Not only has the capability, speed, and reliability increased considerably; the physical size and cost of electronic digital computers has decreased by orders of magnitude during the past decade. The latter phenomenon is especially heartening in an age of constant inflation in the price of nearly every other type of commodity. One can now purchase for a few thousand dollars a small digital computer (or *minicomputer*) which is about the same size as a "hi-fi" set (see Fig. 1-2) and which is enormously more powerful than the early von Neumann machine (which itself cost many hundreds of thousands of dollars and occupied a small building). Hence it seems likely that we are on the threshold of an age when the "family computer" will be as realistic and useful an item as the family automobile has become in American life. At a further extreme, the use of circuit chips and small memories has permitted the development of pocket-size, battery-operated computers (which now sell for a few hundred dollars) which use full-scale digital programming techniques to calculate series solutions for the transcendental functions to high accuracy, together with the more usual arithmetic operations. Indeed, at least one of the currently available pocket computers is capable of retaining as many as 100 fully alterable program steps (see Fig. 1-3). Thus we are already well into an age where small computers can extend the mathematical ability of the human brain in much the same way that hearing aids and electronic guidance devices can extend man's other perception capabilities. Immediate access to these powerful computational aids opens up for solution a new domain of important problems in the same way that a good pair of eyeglasses can help a near-sighted individual see a larger portion of the landscape. In fact, one could make a strong argument that access to both small and large digital computers should be regarded as a fundamental individual right in our society, much like those guaranteed by the Constitution. The point here is that society is presently facing such complex problems as to warrant increased reliance on computer-simulated models for solution. Those people who do not have access to computers to investigate alternative solutions to communal problems will, in some sense, have lost their ability to participate in the democratic process. In any event, digital computers have become easily available to a large fraction of the population, and there is every indication at the present moment that this trend will continue.

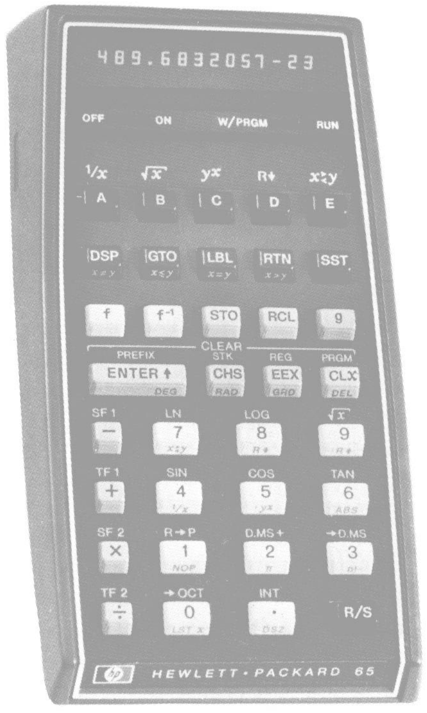

Fig. 1-3. Pocket-sized battery-operated digital computer capable of holding 100 reprogrammable statements. (*Courtesy* of the Hewlett-Packard Co.)

1.2 Machine Language

Most currently available digital computers operate in a manner that is logically similar to the method devised in the von Neumann machine. A certain set of possible binary logic operations (typically about 100) is built into the electronic circuitry of the machine. As part of man's never-ending desire to attribute human qualities to electromechanical devices, the convention by which different sequences of instructions may be fed into the computer is known as a *programming language*. (In a similar vein, one finds computer scientists referring to the rules for applying such languages as *grammar*; the storing capacity of the computer as *memory*; the various multiple-bit numbers stored in the memory as *words*; and so on.) Access to the rudimentary set of binary logic operations wired into the computer is obtained through something known as *machine language*. A *program* in machine language just consists of a sequence of large binary numbers, which are consulted in order when the computer runs. Each memory location in the computer can store a word containing a large number of binary digits, or *bits*. In the section of the computer memory used to store the machine-language program, part of each word is used to code the machine-language instruction; the other part contains the memory location upon which the instruction is to operate. In addition, the computer contains entities known as *registers* within which the various allowed binary logic operations are performed. One of these registers is set aside just to keep track of which memory location contains the next machine-language instruction to be executed.

Thus a machine-language program consists of an ordered set of instructions that is sequentially executed, starting at a specified memory (or *core*) location. Such programs tend to be exceedingly tedious affairs in which the computer is led by the hand through every single operation required. For example, the first instruction might be to take a 16-bit binary number out of one specific memory location and store it in the A register; the next instruction might be to add the contents of another memory location to the contents of the A register; the third instruction might be to store the result (now in the A register) in some other memory location (whose *address* might, in turn, have been computed in still another register); and so on. Generally, an enormous number of machine-language operations have to be performed before anything very useful is accomplished. In addition, all these machine-language instructions have to be entered by some means in the computer memory. (At the most rudimentary extreme, such sequences of binary numbers can be entered by hand using a long row of toggle switches to set up each required multiple-bit binary number.) The great virtue of the computer is that once these instructions have been entered in the memory, long sequences of them can be done over and over again with great speed (these days, anywhere from about 30 nanoseconds to 2 microseconds per machine operation, depending on the particular computer).

1.3 More Advanced Languages: Compilers and Interpreters (Why BASIC?)

Although the early programmers were forced to write their programs directly in machine language, most people will find that practice exceedingly tedious. This is especially true in the routine conversions that come up over and over again in going back and forth between base 10 and base 2 arithmetic.

Fortunately, some dedicated souls have spent their lives devising machine-language programs that do all these routine operations for us. Thus higher-level programming languages, such as FORTRAN, ALGOL, and BASIC, have been developed that translate standard arithmetic operations back and forth to machine-language operations in base 2. In addition, these higher-level languages generally have some standard set of options for getting data in and out of the computer and a set of *subroutines* (small, specialized programs that can be used over and over again at different points in a large program) built in for the purpose of computing common mathematical operations and functions.

As a specific illustration, the two statements

```
1  INPUT X,Y
2  PRINT X+Y
```

written in BASIC, are fairly self-evident even to someone who has never used a computer before. The statements have the meaning that when the program is run in BASIC, two general numbers (X and Y) are entered from the keyboard of a teletype terminal and the computer then prints the sum of the two numbers back on the same terminal. In a representative computer (containing no "hard-wired" arithmetic capability), this relatively harmless-looking two-line program in base 10 arithmetic takes hundreds of separate machine-language steps involving logic operations in base 2. Because the computation would take less than 1 millisecond on even the slowest contemporary computers, the terminal operator is shielded from all the behind-the-scenes effort that went into the calculation.