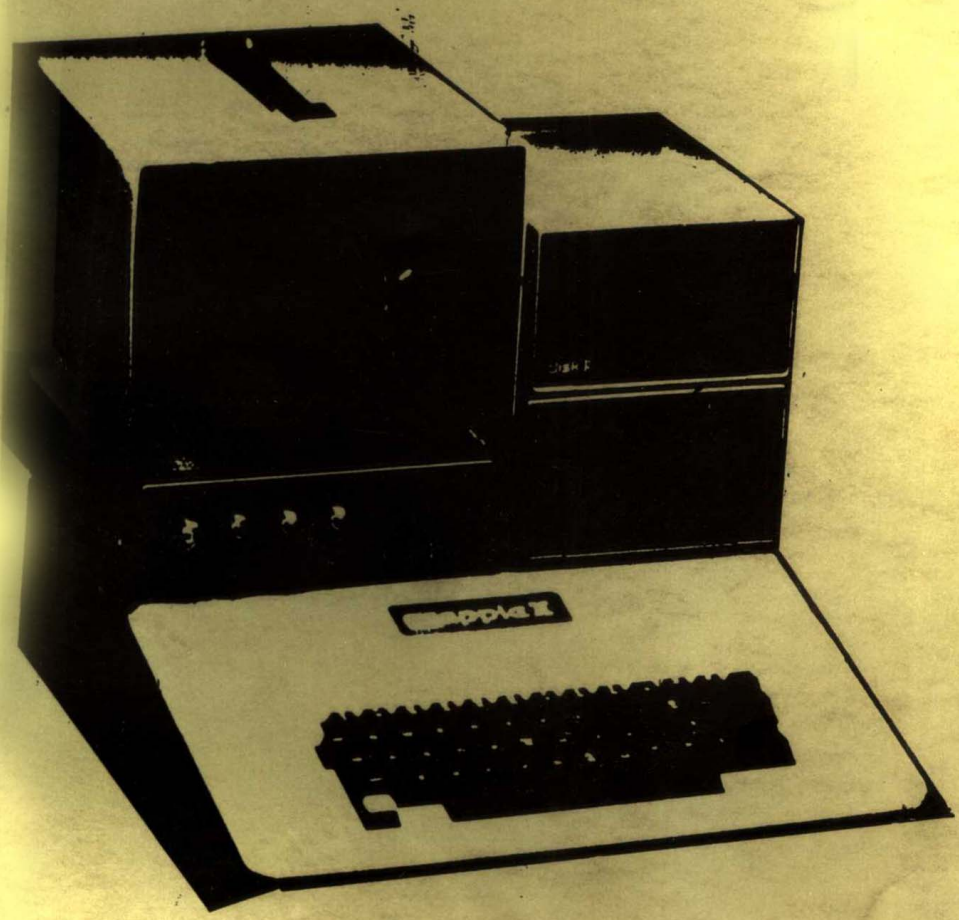
A DICTIONARY OF MINICOMPUTING AND MICROCOMPUTING



PHILIP E. BURTON

A Dictionary of Minicomputing and Microcomputing

Philip E. Burton

Garland STPM Press
New York & London

Library of Congress Cataloging in Publication Data

Burton, Philip E
A dictionary of minicomputing and microcomputing

Bibliography p
1 Minicomputers Dictionaries 2 Microcomputers Dictionaries I Title
QA76 5 B854 001 64 04 0321 80-28272
ISBN 0-8240-7263-4 AACR1

Published by Garland STPM Press
136 Madison Avenue, New York New York 10016

Printed in the United States of America

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Cover photo courtesy of Apple Computer Inc

Preface

"Computing plays such a crucial role in everyday life and in the technological future of this nation that the general public's ignorance of the subject constitutes a national crisis." So said Arthur Luerhmann, director of computer research at the Lawrence Hall of Science of the University of California at Berkeley, in an article entitled "Computer Illiteracy—A National Crisis and a Solution for It," which appeared in the July 1980 issue of *Byte* magazine, the small-systems journal. This article should be reprinted on the editorial pages of every newspaper in the country. He continued: "The ability to use computers is as basic and necessary to a person's normal education as reading, writing, and arithmetic. As jobs become increasingly oriented toward the use of information, society demands and rewards individuals who know how to use information systems. The American computer industry, which leads the world today, depends for its future upon a mass market of computer-literate workers and consumers."

What are the attributes of a computer-literate person?

He feels as comfortable at the keyboard of a computer as he does at a typewriter or at the steering wheel of his car.

He has acquired the basic skills of operating a computer system.

He writes programs in one or more computer languages.

He can express his desires and solve his problems by resolving them into simple sequential steps a computer can comprehend and execute.

"Yet, despite computing's critical importance today, the overwhelming majority of this country's general public is woefully ill-prepared to live and work in the Age of Information, as some have called it."

Why does computer illiteracy constitute a national crisis?

The computer promises a major increase in American productivity. American computer technology is years ahead of those in the rest of the world, at least for now. The greatest market for the computer industry today does not involve merely the demand for stand-alone computers per se. The larger potential lies in the so-called dedicated applications in which integration of one or more computers into a larger machine will revolutionize the way we do our daily work, producing better products at lower cost. Two prime examples:

1. The computer will soon become the standard instrument for written communications. Most offices and most professional writers already use word processors, and their ranks are joined daily by lawyers, architects, consulting engineers, and many other professionals. One encounter with the joys of electronic text editing, and suddenly the old way is unacceptable. A well-designed word processor makes editing a quick, clean, and remarkably easy process; the task may be completed in a fraction of the time older methods required. The quality of the composition is higher because substitutions and deletions of words, sentences, and whole paragraphs on the video monitor screen are so simple and rapid; many of these changes would never be attempted with pencil or ordinary typewriter. The production of the finished typed page is highly cost-effective. One high-speed printer can turn out perfect copies of the entire typing work load for a large office in a few minutes. Word processors are making typists obsolete: people edit, but the computer does the typing. Despite the American lead in computer technology, the potential of the computer and the markets it is generating are clearly recognized by the rest of the world. Foreign governments are spending millions of dollars in attempts to leapfrog the United States in computer sales and applications—military, industrial, and commercial. Although we have a considerable positive trade balance in computers and their peripherals, imports of word-processing equipment already exceed exports by an alarming margin, most from Japan and Italy.

2. The computer has sired another growth industry that is beginning to have profound technological and social impact on world productivity. Sales of robots have already passed the mark of $100 million per year. Japan and West Germany, like the United States, are using robots in many types of production assembly lines, particularly in areas that are hazardous or unpleasant for humans. Robot requirements are spurring basic research in artificial intelligence, "learning" computers, speech recognition, artificial "sight" (video recognition of objects), and many robot-related disciplines.

As the demand for all kinds of computers increases, so does the shortage of computer-literate people to program them. The numbers of scientists and engineers needed to devise new applications for computers and the numbers of technicians required to service them and keep them running are growing exponentially. If we continue to produce trained and capable software authors

at our current rate, the full potential of the computer will never be realized. Hundreds of thousands of challenging new jobs will go begging. America has lost its dominance in automobiles and steel. Will computers and computer-based products be next?

In the February 28, 1980, issue of *Electronics* magazine, in an article entitled "Intel Takes Aim at the 80's," Andrew S. Grove, president and chief operating officer of Intel Corporation, Santa Clara, California, one of the nation's largest producers of microprocessors and microcomputers, quantified the scale of the situation. On the assumption that in the next decade the number of microcomputer designs will increase by 30% yearly and the implementation effort per design will double, Grove came up with what he called an absolutely mind-boggling requirement: "We will need in excess of 1 million software engineers by 1990." Because the electrical engineering graduates in the United States each year currently number in the tens of thousands, Grove called the situation a "programmer catastrophe."

Grove was considering only those software engineers involved in the design of new products and systems. Add to that the numbers of computer-literate people required in business and industry to program and to maintain, update, and modify existing software programs and the shortage takes on even greater dimensions. If these requirements for software authors are not met, they will lead to a national economic and military crisis. Unless we do something to train the people we need, our major competitors will soon be reducing the American share of the lucrative markets for microcomputers and microprocessor-based "smart" products.

The numbers of doctors, lawyers, architects, insurance agents, auto-parts dealers, storekeepers, and owners of small businesses of all sorts who own their own computers are expanding rapidly as the prices of powerful systems with large disk storage continue to fall. Because of the rapid proliferation of these machines, there is a growing demand for people to write programs for them, to modify existing software, and to run and maintain the equipment. Applications in process control and other industrial plant operations are multiplying. The small computer has increased industrial acceptance of computer control by making distributed, redundant systems economically feasible, thus making industrial processes less vulnerable to failure of a single central computer.

Why is computer literacy so important to the individual?

To quote again from Arthur Luerhmann: "People with programming skills command jobs with far higher salaries than are average for people with similar education; and word-processing specialists earn more than clerk-typists. Corporate profitability incentives are equally strong, because the success of businesses depends as much on the creative use of information as upon the efficient use of material and energy resources. Personal incentives are also great. The children and adults I see learning to use computers display an unusual intensity of concentration and evident satisfaction with their results."

What must we do to achieve a computer-literate society?

This is a complex question about which many books are now being written. The scope of the subject extends well beyond what can be printed in the preface to a dictionary of microcomputing. The question can be divided into three topics for discussion:

1. How should we introduce our infants and preschool children to the computer, and what role should the computer play in a child's learning process?

2. How do we enable our children who are already in school to develop a comfortable relationship to computers.

3. How are those who have left school to be educated?

In his article "New Cultures from New Technologies," which appeared in the September 1980 issue of *Byte* magazine, Seymour Papert presented some prophetic and provocative new ideas. Papert's "vision of a few ways in which computers might affect how children learn" is not the idle speculation of a hyperactive computer hobbyist; he is associated with project LOGO, the Massachusetts Institute of Technology Artificial Intelligence Laboratory. The concepts in the *Byte* article came from his book *Mindstorms, Children, Computers and Powerful Ideas* (New York, Basic Books, 1980, ISBN 0-465-04627-4). Papert stated that computers will become "so integrated into new ways to think about ourselves and about the subject matters we learn that the nature of learning itself will be transformed." Papert's book should be read by everyone interested in the education of children, for therein lies the future of our country. "It is time we learned to think in terms of a computer for every child, and we should think about children having access to computers from infancy." Children should be learning to use computer terminals at the same time they are learning to talk.

The role of today's low-cost computers in the educational process is being debated with as much intensity, apprehension, and misinformation as was Darwin's theory of evolution. In an article in the *New York Times* on July 22, 1980, entitled "About Education—Debating the Role of Computers," Fred M. Hechinger wrote that "the school people observed at these workshops appeared to be in two categories: those who looked forward to having a microcomputer on every desk, with great expectations for more effective learning at a much lower cost; and those who feared a '1984' world of automated, dehumanized, fragmenting of information. Proponents of computer-assisted teaching make three points. First, they say that the computer is an invaluable help to the teacher. Second, they warn that inflation-shrunken budgets make drastically cheaper teaching devices crucial for the public schools' survival. Third, they say schools must adopt microcomputers before the home, or risk a parental takover of the basic learning functions." Whether or not the preceding analysis is correct (and I believe it to be so), the article fails to address the necessity for bringing computer literacy to every student at the earliest possible age. As Arthur Luehrmann said in his *Byte* magazine article, "all future students should

acquire basic skills in computer use, including hands-on operation, programming, and problem solving, during their early secondary school years. They should also make further use of these skills in other courses in mathematics, science, language, etc., and in vocationally oriented courses in word processing, accounting, and the like."

Most of the new words defined in this edition are from the software world, largely because of the economics of the computer world. The power and complexity of microprocessor chips have grown as fast as the numbers sold, but the price of the hardware continues to be almost negligible when compared with the software costs. The expenses involved in software authorship are riding upward with inflation. In the past few years, extensive efforts have been made to reverse this trend. The approach has been twofold: (a) to minimize software costs, new hardware chips have been specifically designed to be compatible with high-level languages; (b) the use of structured, self-documenting languages (PASCAL, ADA, COBOL, and others) has been facilitated by the design of chip architecture and the development of associated compilers that convert these high-level languages into codes that the machines can understand.

Efforts to reduce the high cost of software are under way, with heavy emphasis on top-down modular design and structured programming. The term "structured programming" has been used for many years to describe an overall orderly design process of which "structured coding" in an integral part. The discipline of structured coding began as far back as the late 1950s, emerging principally from the work of Dijkstra, Jacopini, Böhm, and Warnier. The original concept of structured coding grew out of a 1966 article by Böhm and Jacopini, who proved that any program could be built up from three simple constructs now called the Böhm-Jacopini constructs: PROCESS, IF-THEN-ELSE, and DO-WHILE. Some were not content to leave things that simple, and so the "extended constructs" were added: CASE, REPEAT-UNTIL, and DO-WHILE WITH BREAK. Structured programming has contributed so significantly to the literature that an entire appendix containing much of the special vocabulary associated with it has been included (Appendix A).

Although there are other contenders for preeminence (such as ADA, the high-level language the Department of Defense is subsidizing), many people think that PASCAL, with extensions, may well become as nearly universal a language as any for some time to come. PASCAL has a vocabulary all its own; Appendix B provides a glossary of PASCAL terms.

The other appendixes, Magnetic-Bubble Memory Technology (Appendix C), Printers (Appendix D), Automatic Control (Appendix E), Multiprocessing (Appendix F), Data Communications (Appendix G), and Magnetic Recording and Storage Technology (Appendix H) all cover what I consider to be the most important and most dynamic aspects of computing at the present time, from the point of view of the user. The reader should find it timesaving and beneficial to have these subjects separated into individual appendixes for easy reference.

Acknowledgments

I am deeply indebted and grateful to the following people who helped to make this volume possible: Dr. John Steinhoff, Computing Sciences Branch, Research Department, Grumman Aerospace Corporation, who read the manuscript for the first edition and made helpful criticisms, suggestions, and comments; Dr. A. R. Palmer, former head of the Department of Earth Sciences, State University of New York at Stony Brook, for his advice and encouragement; my twin brother, James R. Burton, General Electric Company, Syracuse, New York, who reviewed the manuscript for the second edition, for his eagle eye and professional recommendations; Diane Carter, who typed the manuscript and lost a considerable amount of sleep catering to my capricious last-minute changes without rapping my skull, and all the other Carters who also helped; Susan Feeney, who also typed a portion of the second edition; Nicole Scholten, and Kathy and Karen, who performed some boring editorial chores.

How to Use This Book

This book is organized as an alphabetically ordered dictionary, followed by nine separate appendixes. Appendixes A through H are devoted to one specialized aspect of computing that I believe to be particularly dynamic and relevant at the present time: Structured Programming, PASCAL, Magnetic-Bubble Memory Technology, Printers, Automatic Control, Multiprocessing, Data Communication, and Magnetic Recording and Storage Technology. Appendix I contains Tables of Powers of 2 and Hexadecimal Arithmetic. Because the primary purpose of the book is to inform and explain, it seemed to me that the reader would appreciate having other related terms in one area.

Many of the terms used in any rapidly evolving technology do not have unique definitions; they may be used in different ways in different situations, and the use of a term can differ from one manufacturer to another. I have tried to list all the meanings I have encountered in my work and study.

All definitions that appear in the various appendixes are cross-referenced in the main dictionary. I shall probably be chastised by formal lexicographers for listing terms of more than one word alphabetically in the order in which they are used. For example, "source code" appears under the letter S, and it is not cross-referenced as "code, source" under the letter C. (Dictionaries are expensive enough as it is.) For the same reason, terms are defined as concisely as possible for the layman and novice. Many terms are illustrated with figures, particularly in cases in which a picture is worth the proverbial thousand words. Where appropriate, examples are given to clarify a concept or meaning. An abbreviation will appear in alphabetical order, not at the beginning of the listings for its first letter.

Words that have the same meaning in the computer world and elsewhere usually are not listed. In writing the book and in determining when it was complete, I attempted to "define in clear, easy-to-understand terms all the essential vocabulary the student or layman needs to read and understand today's burgeoning computer literature." I believe the book meets that goal.

Contents

A

abbreviated addressing: a modification of the direct addressing mode. It uses only a part of the full address to increase execution speed because of the shorter code, which allows direct addressing.

abort: 1: to terminate the execution of a program before it finishes its normal run. 2: a procedure to abort when an irrecoverable error, malfunction, etc., occurs.

absolute address: 1: an address specified by its actual numbered location in memory, as opposed to one identified by a label or one calculated by adding or subtracting a displacement value from the contents of the program counter or an index register. 2: a short string of characters that identifies a memory location without modification. 3: synonymous with machine address and specific address.

absolute addressing mode: an address mode in which the effective address of the operand is the next word (absolute short) or the next two words (absolute long) following the opcode word of a computer instruction in the program listing—Motorola 68000. In the 6800, the same mode is called direct addressing.

absolute magnitude: *See* **absolute value**

absolute value: the value of a given quantity without regard to its sign (i.e., whether positive or negative). Most computer high level languages have an absolute-value function, ABS(*X*), that returns the absolute value of the argument *X* when it executes. For example, if *X* = −5.05, ABS (*X*) = 5.05. Synonymous with absolute magnitude.

access: 1: the process of obtaining data from storage or putting data into storage. 2: to attempt to retrieve data from or store data in a storage device. *See also* **direct memory access**.

access arm: *See* Appendix H.

access memory: to fetch a word from memory and store it in a CPU register. The time required is called memory access time and is a criterion of computer speed.

access mode: a COBOL programming technique used to obtain a specific record from a file residing in a storage device or to store a specific record in that file.

access time: 1: the interval between the instant the address of a memory cell is stable on the address bus and the instant the requested data are stable on the data bus. 2: the time required for any device to receive or transmit data after the associated command is received.

access violation: an attempt to access an illegal memory address, or an attempt to access a location outside of the allocated limits for a memory segment.

accumulator: one or more registers that temporarily store sums and other arithmetic or logic results in the arithmetic/logic unit (ALU) of a CPU.

accumulator addressing mode: an addressing mode that involves one or more accumulators and therefore inherently contains all the addressing information required.

Examples:

CLRB: clear accumulator B. ABA: add the contents of accumulator A to accumulator B and place the results in accumulator A.

accuracy: freedom from error, as opposed to precision, the number of significant digits used to express a quantity. For example, a five-place error-free sine table is accurate; a seven-place table with errors is more precise, but inaccurate.

ACIA: acronym for asynchronous communications interface adapter, a Motorola device for use with the MC6800 family of microprocessors. It converts 8-bit parallel outputs from the CPU into serial characters for transmission to a teletype (TTY) or cathode-ray-tube (CRT) terminal. Likewise, it converts serial data from a peripheral device to 8-bit parallel form for CPU input. The ACIA can interface the CPU with a low-speed modem, enabling direct microprocessor-to-microprocessor communication over ordinary telephone lines, using acoustic couplers.

acknowledge signal: a pulse or voltage level sent by a receiving device to indicate receipt of a sender's transmission.

ACL: appliance computer language, the hypothetical future high-level programming language for the personal (appliance) computer.

acoustic coupler: an adapter for transmitting digital data over ordinary voice-quality telephone lines. After the receiving number is dialed, the telephone handset is placed in cushioned muffs. Serial data inputs are converted by a modem into tones that can be sent over the telephone lines. A similar or identical modem on the receiving end changes the tones back into digital data.

activate: a PASCAL programming-language term. *See* Appendix B.

active device: an electronic circuit that contains an amplifier providing gain. A passive device contains only resistors, capacitors, diodes, etc.

active state: the digital state that causes a given action to occur. It may be either the high state or the low state, depending on the circuit and pin in question.

actual parameter: *See* Appendix B.

actuator: an electromechanical, hydraulic, or pneumatic power device that changes the linear or rotational position and/or speed of an object in response to weaker control signals.

ADA: a candidate to fulfil the Department of Defense desire for a universal programming language, at least for the military. ADA's heritage is largely PASCAL. It is now undergoing tests and evaluation that should be completed by the mid-1980's, after which implementation can proceed in earnest. ADA will figure prominently in real-time multitask applications, especially process control.

A/D, ADC: abbreviations for analog-to-digital converter.

adapter: any device used to create compatibility between parts, equipments, subsystems, or whole systems; an interfacing unit.

adaptive process: *See* Appendix E.

adaptive tuning. See Appendix E.

addend: an operand of the addition operation; the number added to the augend to form a sum.

adder: an electronic circuit that forms the sum of two or more quantities. *See* **quarter adder, half adder,** and **full adder**.

adder-subtracter: a digital arithmetic element that performs either addition or subtraction, depending on the state of a control "switch." The subtraction path has a complementer circuit that forms the 2's complement of the subtrahend and then adds it to the minuend in a conventional adder circuit.

add instruction: a computer instruction that performs an arithmetic addition of two numbers. Usually the augend must be put into the accumulator first. The addend may be in another CPU register or may be fetched from a memory location. After the add instruction is executed, the result is in the accumulator.

address: 1: a binary word used by a programmer to designate a particular memory location or input output (I O) unit. The computer places the word on its address bus, uniquely selecting the desired memory cell or I/O device. 2: a number, name, or label uniquely identifying a location or a device where data are stored.

address bus: the bus lines that transmit the address word from the CPU to the computer memory and all I/O peripheral devices. One unique location is selected and conditioned to send its data to the CPU or receive data from it.

address field: that part of an instruction-word format reserved for an address code.

address offset: 1: a number added to a page number to address a memory location within the page boundaries. 2: a number added to the program counter during a branch instruction to determine a memory address relative to the program counter. Also called relative offset or REL.

address register: a special register used by the CPU to store an address. It may be in the CPU itself to address memory or in an I/O unit, where it stores the

address word unique to that device. The address word is then decoded to wake up the device and make it do its thing.

address space: the number of words or bytes of main memory (not bulk or secondary) that can be addressed by the CPU without additional external logic. One 16-bit microprocessor has an addressing capability of 16,777,216 bytes; 8-bit CPUs are usually limited to 65,536 bytes or less.

address translation: a memory-management technique. Memory addresses generated by the program are treated as logic addresses, to be interpreted or translated into real-world physical memory locations by the memory-management system before the processor sends the memory access request to the memory system.

add time: the time required by a given microprocessor to add two multidigit decimal numbers. For example, for two 16-digit numbers, the computer performs the addition by using 2's complement binary numbers. Microcomputers often are rated by comparing their add times, a criterion of relative speed.

adjacent-channel interference: unwanted leakage of the transmission of one channel into another channel next to it.

ADP: automatic data processing.

AED: automatic engineering design, an MIT-developed extension of ALGOL.

ALGOL: algorithmic language, a high-level programming language designed to facilitate the proper writing and documentation of numerical and other algorithms in a nearly standard, essentially machine-independent form. The language was invented to promote clearer communications between individuals, and it remains one of the better (and still surviving) attempts at a common programming language. ALGOL was developed by international cooperation to obtain a standardized algorithmic language, and is widely employed in Europe.

algorithm: a procedure made up of mathematical and/or logic operations that achieves a desired result when followed. *Examples:* Booth's algorithm for multiplication; a square-root algorithm.