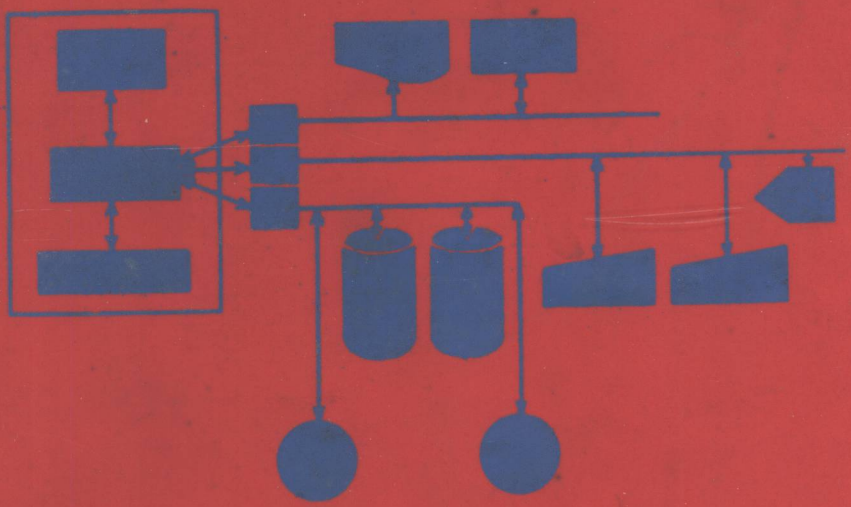
Digital Computer System Principles

HERBERT HELLERMAN



T M H EDITION

SECOND EDITION

Digital Computer System Principles

Herbert Hellerman

School of Advanced Technology
State University of New York at Binghamton

TATA McGRAW-HILL PUBLISHING COMPANY LTD.
New Delhi

Digital Computer System Principles

T M H Edition 1974

Reprinted in India by arrangement with McGraw-Hill, Inc.
New York.

This edition can be exported from India only by the Publishers,
Tata McGraw-Hill Publishing Company Ltd.

Published by Tata McGraw-Hill Publishing Company Limited and
Printed by Mohan Makhijani at Rekha Printers Private Limited, New Delhi-110015.

Monadic form f B			f	Dyadic form A f B	
Definition or example		Name		Name	Definition or example
+B ↔ 0+B		Plus	+	Plus	2+3.2 ↔ 5.2
-B ↔ 0-B		Negative	-	Minus	2-3.2 ↔ ¯1.2
×B ↔ (B>0)-(B<0)		Signum	×	Times	2×3.2 ↔ 6.4
÷B ↔ 1÷B		Reciprocal	÷	Divide	2÷3.2 ↔ 0.625
B	⌈B / ⌊B	Ceiling	⌈	Maximum	3⌈7 ↔ 7
¯3.14	¯3 / ¯4	Floor	⌊	Minimum	3⌊7 ↔ 3
⋆B ↔ (2.71828...)⋆B		Exponential	⋆	Power	2⋆3 ↔ 8
⍟⋆N ↔ N ↔ ⋆⍟N		Natural logarithm	⍟	Logarithm	A⍟B ↔ log B base A 'A⍟B ↔ (⍟B)÷⍟A
|¯3.14 ↔ 3.14		Magnitude	|	Residue	Case: A|B A=0: B-(|A)×⌊B÷|A A=0, B≥0: B A=0, B<0: DOMAIN ERROR
!0 ↔ 1 !B ↔ B×!B-1 or !B ↔ gamma (B+1)		Factorial	!	Binomial coefficient	A!B ↔ (!B)÷(!A)×!B-A 2!5 ↔ 10 3!5 ↔ 10
?B ↔ random choice from ⍳B		Roll	?	Deal	A mixed function (see Fig. 2.18.2)
○B ↔ B×3.14159...		Pi times	○	Circular	See table at left
~1 ↔ 0 ~0 ↔1		NOT	~		
			∧	AND	A: 0 0 1 1; B: 0 1 0 1; A∧B: 0 0 0 1; A∨B: 0 1 1 1; A⍲B: 1 1 1 0; A⍱B: 1 0 0 0
			∨	OR	
			⍲	NAND	
			⍱	NOR	
			<	Less	Relations
			≤	Not greater	Result is 1 if the
			=	Equal	relation holds, 0
			≥	Not less	if it does not:
			>	Greater	3≤7 ↔ 1
			≠	Not Equal	7≤3 ↔ 0

Table of dyadic ○ functions

(-A)○B	A	A○B
(1-B⋆2)⋆.5	0	(1-B⋆2)⋆.5
Arcsin B	1	Sine B
Arccos B	2	Cosine B
Arctan B	3	Tangent B
(¯1+B⋆2)⋆.5	4	(1+B⋆2)⋆.5
Arcsinh B	5	Sinh B
Arccosh B	6	Cosh B
Arctanh B	7	Tanh B

Fig. 2.6.1. Primitive scalar functions (operators).

To my wife Elaine

Preface

The rapid progress in the design and application of general-purpose digital computers has tended to outdistance systematic treatment of the field as a whole. A major step in this treatment is to recognize information processing as a subject in its own right and not as just a tool for other disciplines. This recognition is justified by the large and growing body of techniques and knowledge derived from the many designs and widespread use of computer systems, as well as the influence of such systems on our way of thinking about problems.

This second edition is an extensively revised version of the first edition, but it has the same basic purpose: to give a college-level treatment of the important principles of digital computer systems. The viewpoint is primarily tutorial rather than encyclopedic in that it is intended to impart skills and to encourage a critical and analytic spirit. Much attention is given to those ideas that are common to many aspects of computer systems so that the reader may not only learn specific practices, but also learn *how* to learn about them to develop the awareness and confidence needed to tackle new problems. To help accomplish this, categories of structure, alternative considerations, and summary information are highlighted in charts which constitute many of the figures. Also, no opportunity is lost to repeat unifying techniques and ideas (like space-time trade-offs and the finite-state-machine model) in several different contexts.

Major changes from the first edition include: use of the APL language rather than its predecessor, the Iverson language; more emphasis on statistics and string processing in programming examples; expanded treatment of program translation; extensive revision of the chapter on finite-state and Turing machine

models; statements on the principles and rationale of microprogramming; and inclusion of architectural features of the IBM system/370 as well as some detail of the buffer store organization of the Model 155.

The reader is assumed to be comfortable with mathematics through college algebra. In a few cases, higher mathematics is used, but these are infrequent and may be skipped without losing continuity. Although the text is self-contained and hence suitable to novices, it is likely to be best appreciated by those who are not encountering the subject of programming for the very first time. Those with some computer experience may also welcome a systematic coverage of material they may have acquired only piecemeal.

A key notion in all of computer science is the sequential process. Examples include methods of solving linear equations, evaluating formulas, or determining the internal switching operations to implement a single computer instruction. Programming in this broad sense is a major theme of this book. Topics in logical design, machine description, numerical analysis, and program translation are discussed from this viewpoint.

Chapter 1 gives a brief historical perspective and an informal overview of important ideas of machine organization and programming.

Chapter 2 introduces the APL programming language used throughout the book. Most of the examples and exercises in this chapter are taken from elementary numerical mathematics, statistics, and string processing.

Chapter 3 discusses several topics of language description and translation, including techniques used in compilers and interpreters.

Chapter 4 is an introductory treatment of storage organization, including descriptions of selected devices and algorithms for list maintenance, sorting, and searching.

Chapters 5, 6, and 7 are hardware-oriented and treat combinational circuits, bussing and magnetic-core storage, and sequential circuits, respectively.

Chapter 8 is concerned with the detailed representation and manipulation of information and includes arithmetic operations and coding schemes.

Chapter 9 is concerned with the architecture of computer equipment, i.e., the alternatiaves available to designers in selecting addressing, instruction sequencing, input/output control, and privileged-mode features. It also includes a detailed discussion of the rationale and technique of microprogramming and a simple machine example.

Chapter 10 is a description of the architecture of the IBM system /360 and system/370. Although the emphasis is on the appearance of the system to its machine-language users, some implementation topics are also included, especially the buffer-storage organization of the Model 155.

Chapter 11 is an introduction to reliability theory and some of its elementary applications to computer systems.

As indicated earlier, programming forms a major theme of the book. The APL language was selected to meet the need of a single comprehensive method of

describing sequential processes as programs. It was chosen because its extensive set of operators and its ability to directly specify operations on arrays permit concise descriptions without requiring unessential detail. Algorithms for internal machine operations, programming systems, and problem solutions are presented as programs or statements in a single notation. Most programs are augmented by word descriptions (sometimes line-by-line). Since the first edition of this book, which used the Iverson language (an early form of APL), machine implementations of APL have become available on several computers. For those readers with access to such a system, the programs in the text may be entered and run from timesharing terminals. If an APL system is not available, the programs may still be used as a powerful means of description and may also be transcribed to other machine-executable languages. For convenience in reference and to encourage self-study, most of the APL operators (including examples of use) are summarized in a few charts.

To highlight general applicability of results and techniques, topics are often treated abstractly, but are illustrated with concrete examples. These are taken from several systems, especially IBM systems. This is due in part to my own experience but also to the fact that most of the world's computers are manufactured by IBM. Of course many features of IBM systems originated in equipment of other manufacturers or in research groups and also appear elsewhere.

Like the first edition, this book is intended for colleges and college-level courses in industry as well as for self-study. A one-semester introductory course at the senior-first-year graduate level in electrical engineering may use most of Chapters 1, 2, 5, 6, 7, and 8. A course in programming, using APL as the major language, can use Chapters 1, 2, 3, 4, etc., to introduce a simple Assembler language using part of Chapter 9. A two-semester course in introductory computer science or computer engineering could use the book in cover-to-cover fashion.

I am indebted to Mr. John McPherson and Dr. Frank Beckman of the IBM Systems Research Institute for the encouragement that led to the first edition. Others who contributed with valuable criticisms include Mr. C. L. Gold, Mr. John McKeehan, Miss Barbara White, and Dr. G. M. Weinberg. My indebtedness to Dr. K. E. Iverson, the principal architect of the APL language, is indicated by the use of the language throughout the text. The diligent work of Mr. Gary Rogers of the State University of New York, Binghamton, in checking out several of the APL programs and reading proof was most valuable. Finally, thanks are due to Mrs. Shanna McGoff for her help in typing the manuscript.

Herbert Hellerman

Contents

1

Automatic Computer Systems

The modern general-purpose digital computer system, which is the subject of this book, is the most versatile and complex creation of mankind. Its versatility follows from its applicability to a very wide range of problems, limited only by human ability to give definite directions for solving a problem. A *program* gives such directions in the form of a precise, highly stylized sequence of statements detailing a problem-solution procedure. A computer system's job is to reliably and rapidly execute programs. Present speeds are indicated by the rates of arithmetic operations such as addition, subtraction, and comparison, which lie in the range of about 100,000 to 10,000,000 instructions per second, depending on the size and cost of the machine. In only a few hours, a modern large computer can do more information processing than was done by all of mankind before the electronic age, which began about 1950! It is no wonder that this tremendous amplification of human information-processing capability is precipitating a new revolution.

To most people, the words "computer" and "computer system" are probably synonymous and refer to the physical equipment, such as the Central Processing Unit, console, tapes, disks, card reader, and printers visible to anyone visiting a computer room. Although these devices are essential, they make up only the visible "tip of the iceberg." As soon as we start to use a modern computer system, we are confronted not by the machine directly but by sets of rules called *programming languages* in which we must express whatever it is we want to do. The central importance of programming language is indicated by the fact that even the physical computer may be understood as a hardware interpreter of one particular language called the *machine language.* Machine languages are designed

for machine efficiency, which is somewhat dichotomous with human convenience. Most users are shielded from the inconveniences of the machine by one or more languages designed for good man-machine communication. The versatility of the computer is illustrated by the fact that it can execute translator programs (called generically *compilers* or *interpreters*) to transform programs from user-oriented languages into machine-language form.

It should be clear from the discussion thus far that a computer system consists of a computer machine, which is a collection of physical equipment, and also programs, including those that translate user programs from any of several languages into machine language. Most of this book is devoted to examining in some detail theories and practices in the two great themes of computer systems: equipment (hardware) and programming (software). It is appropriate to begin, in the next section, by establishing a historical perspective.

1.1 HISTORICAL PERSPECTIVE

Mechanical aids to counting and calculating were known in antiquity. One of many ancient devices, the abacus, survives today as a simple practical tool in many parts of the world, especially the East, for business and even scientific calculations. (A form of the abacus was probably used by the ancient Egyptians, and it was known in China as early as the sixth century B.C.) In the hands of a skilled operator, the abacus can be a powerful adjunct to hand calculations. There are several forms of abacus; they all depend upon a positional notation for representing numbers and an arrangement of movable beads, or similar simple objects, to represent each digit. By moving beads, numbers are entered, added, and subtracted to produce an updated result. Multiplication and division are done by sequences of additions and subtractions.

Although the need to mechanize the arithmetic operations received most of the attention in early devices, storage of intermediate results was at least as important. Most devices, like the abacus, stored only the simple current result. Other storage was usually of the same type as used for any written material, e.g., clay tablets and later paper. As long as the speed of operations was modest and the use of storage also slow, there was little impetus to seek mechanization of the *control* of sequences of operations. Yet forerunners of such control did appear in somewhat different contexts, e.g., the Jacquard loom exhibited in 1801 used perforated (punched) cards to control patterns for weaving.

Charles Babbage (1792–1871) was probably the first to conceive of the essence of the general-purpose computer. Although he was very versatile, accomplished both as a mathematician and as an engineer, his lifework was his computing machines. It is worth noting that Babbage was first stimulated in this direction because of the unreliability of manual computation, *not* by its slow speed. In

particular, he found several errors in certain astronomy tables. In determining the causes, he became convinced that error-free tables could be produced only by a machine that would accept a description of the computation by a human being but, once set up, would compute the tables and print them—all without human intervention. Babbage's culminating idea, which he proposed in great detail, was his Analytic Engine, which would have been the first general-purpose computer. It was not completed because he was unable to obtain sufficient financial support.

As Western industrial civilization developed, the need for mechanized computation grew. As the 1890 census approached in the United States, it became clear that if new processes were not developed, the reduction of the data from one census would not be complete before it was time for the next one. Dr. Herman Hollerith applied punched cards and simple machines for processing them in the 1890 census. Thereafter, punched-card machines gained wide acceptance in business and government.

The first third of the twentieth century saw the gradual development and use of many calculating devices. A highly significant contribution was made by the mathematician Alan Turing in 1937, when he published a clear and profound theory of the nature of a general-purpose computing scheme. His results were expressed in terms of a hypothetical "machine" of remarkable simplicity, which he indicated had all the necessary attributes of a general-purpose computer. Although Turing's machine was only a theoretical construct and was never seriously considered as economically feasible (it would be intolerably slow), it drew the attention of several talented people to the feasibility of a general-purpose computer.

World War II gave great stimulus to improvement and invention of computing devices and the technologies necessary to them. Howard Aiken and an IBM team completed the Harvard Mark I electric computer (using relay logic) in 1944. J. P. Eckert and J. W. Mauchly developed ENIAC, an electronic computer using vacuum tubes in 1946. Both these machines were developed with scientific calculations in mind. The first generation of computer technology began to be mass-produced with the appearance of the UNIVAC I in 1951. The term "first generation" is associated with the use of vacuum tubes as the major component of logical circuitry, but it included a large variety of memory devices such as mercury delay lines, storage tubes, drums, and magnetic cores, to name a few.

The second generation of hardware featured the transistor (invented in 1948) in place of the vacuum tube. The solid-state transistor is far more efficient than the vacuum tube partly because it requires no energy for heating a source of electrons. Just as important, the transistor, unlike the vacuum tube, has almost unlimited life and reliability and can be manufactured at much lower cost. Second-generation equipment, which appeared about 1960, saw the widespread installation and use of general-purpose computers. The third and fourth

generations of computer technology (about 1964 and 1970) mark the increasing use of integrated fabrication techniques, moving to the goal of manufacturing most of a computer in one automatic continuous process without manual intervention. Although this goal is not quite met even by existing fourth-generation technology, costs are sharply down, reliability has increased, and miniaturization improved. Miniaturization is essential for high speed because electric signals must travel from point to point within the computer. Since the maximum propagation speed is limited to the speed of light, minimum delays require the shortest possible path lengths, obtainable by fabricating circuit components and their interconnections as small as possible.

Hardware developments were roughly paralleled by progress in programming, which is, however, more difficult to document. An early important development, usually credited to Grace Hopper, is the symbolic machine language which relieves the programmer from many exceedingly tedious and error-prone tasks. Another milestone was FORTRAN (about 1955), the first widely used *high-level language*, which included many elements of algebraic notation, like indexed variables and mathematical expressions of arbitrary extent. Since FORTRAN was developed by IBM, whose machines were most numerous, FORTRAN quickly became pervasive and, after several versions, remains today a very widely used language.

Other languages were invented to satisfy the needs of different classes of computer use. Among the most important are COBOL, for business-oriented data processing; ALGOL, probably the most widely accepted language in the international community, particularly among mathematicians and scientists; and PL/I developed by IBM and introduced in 1965 as a single language capable of satisfying the needs of scientific, commercial, and system programming. Finally, in this brief and incomplete listing we must mention APL, the language developed chiefly by K. E. Iverson, which is used throughout this book. APL is in many ways the most sophisticated of existing languages and first became widely available in 1966.

Along with the introduction and improvements of computer languages, there was a corresponding development of programming technology, i.e., the methods of producing the compiler and interpreter translators and other aids for the programmer. A very significant idea that has undergone intensive development is the *operating system*, which is a collection of programs responsible for monitoring and allocating all systems resources in response to user requests in a way that reflects certain efficiency objectives. By 1966 or so, almost all medium to large computers ran under an operating system. Jobs were typically submitted by users as decks of punched cards, either to the computer room or by *remote-job-entry* (RJE) terminals, i.e., card reader and printer equipment connected by telephone lines to the computer. In either case, once a job was received by the computer, the operating system made almost all the scheduling

decisions. A large computer could run several hundred or even thousands of jobs per 24-hour day with only one or two professional operators in the machine room.

The 1960s saw a great intensification of the symbiosis of the computer and the telephone system (*teleprocessing*). Much of this was RJE and routine non-general-purpose use, such as airline reservation systems. Considerable success was also achieved in bringing the generality and excitement of a general-purpose computer system to individual people through the use of *timesharing* systems. Here, an appropriate operating-system program interleaves the requests of several human users who may be remotely located and communicating over telephone lines using such devices as a teletype or typewriter terminal. Because of high computer speed relative to human "think" time, a single system could comfortably service 50 to 100 (or more) users, with each having the "feel" of his own private computer. The timesharing system, by bringing people closest to the computer, seems to have very great potential, as yet largely unexplored, for amplifying human creativity.

1.2 A CLASSIFICATION OF AUTOMATIC COMPUTERS

Automatic computers may be broadly classified as analog or digital (Fig. 1.2.1). Analog computers make use of the analogy between the values assumed by some physical quantity, such as shaft rotation, distance, or electric voltage, and a variable in the problem of interest. Digital computers in principle manipulate numbers directly. In a sense all computers have an analog quality since a physical representation must be used for the abstraction that is a number. In the digital computer, the analogy is minimal, while the analog computer exploits it to a very great extent.

Both analog and digital computers include a subclass of rather simple machines that mechanize only specific simple operations. For example, the slide rule is an analog computer that represents numbers as distances on a logarithmic scale. Multiplication, division, finding roots of numbers, and other operations are done by adding and subtracting lengths. Examples of operation-only machines of the digital type include adding machines and desk calculators.

A second class, more sophisticated than operation-only machines, may be termed *problem-setup* machines. In addition to performing arithmetic operations they can accept a description of a procedure to link operations in sequence to solve a problem. The specification of the procedure may be built into the machine's controls, as in certain special-purpose machines, or a plugboard arrangement may be supplied for specifying the desired sequence of operations. The main idea is that the problem-solution procedure is entered in one distinct operation, and thereafter the entire execution of the work on the problem is automatic.