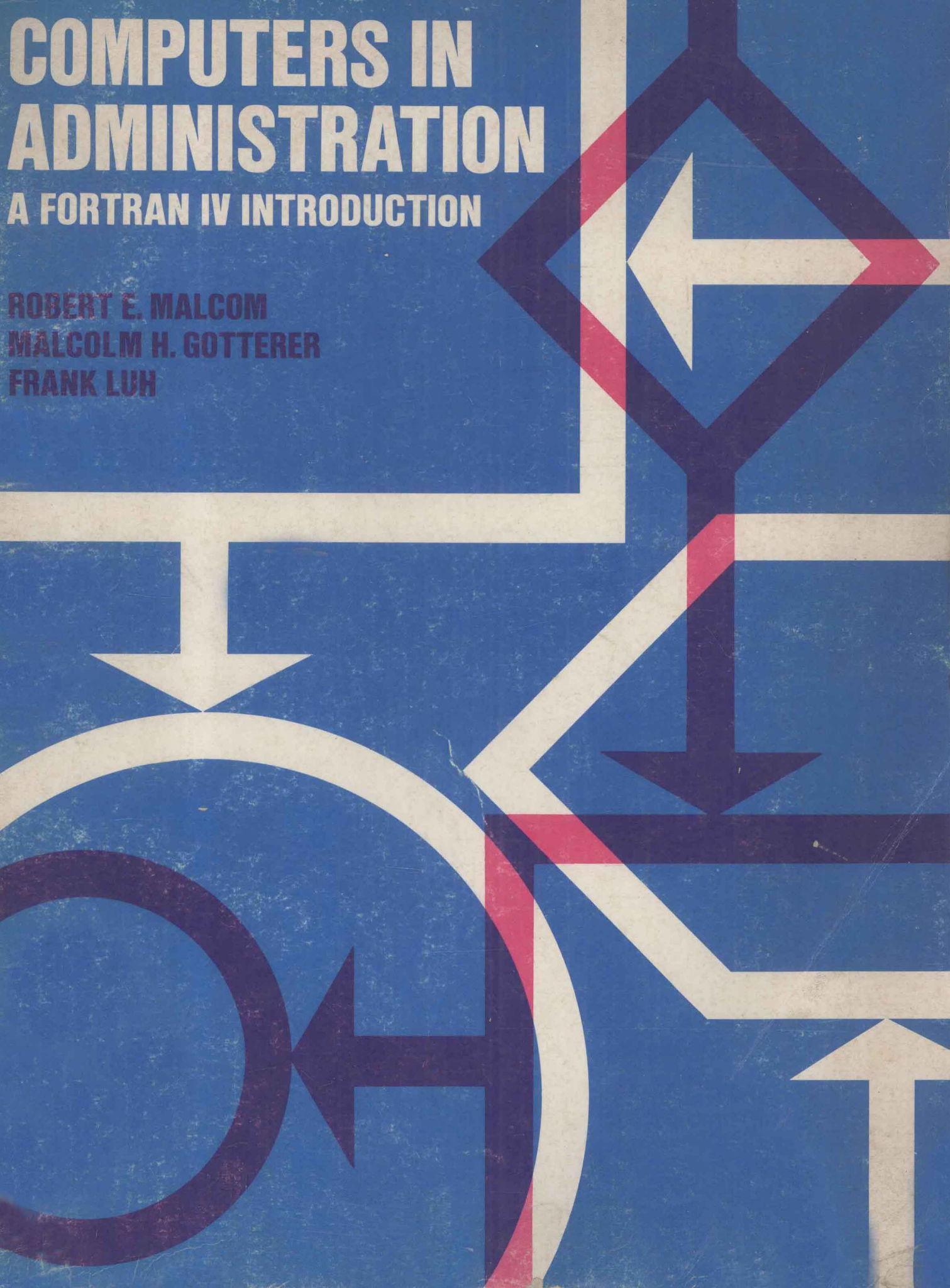
COMPUTERS IN ADMINISTRATION

A FORTRAN IV INTRODUCTION

ROBERT E. MALCOM
MALCOLM H. GOTTERER
FRANK LUH



COMPUTERS IN ADMINISTRATION

A Fortran IV Introduction

ROBERT E. MALCOM
The Pennsylvania State University

MALCOLM H. GOTTERER
The Pennsylvania State University

FRANK LUH
Lehigh University

INTEXT EDUCATIONAL PUBLISHERS *New York / London*

Library of Congress Cataloging in Publication Data

Malcom, Robert E
Computers in administration.
(Intext series in general business)
An updated version of Computers in business, published in 1968.
1. FORTRAN (Computer program language)
I. Luh, Frank, joint author. II. Gotterer, Malcolm H., 1924- joint author. III. Title.
HF5548.5.F2M3 1973 651.8'02'4658 73-749
ISBN 0-7002-2426-2

INTEXT EDUCATIONAL PUBLISHERS
257 Park Avenue South
New York, New York 10010

COMPUTERS IN ADMINISTRATION

A Fortran IV Introduction

Preface

It is the authors' belief that all persons preparing for a career in administration—whether in the private or public sector—should have some understanding of the capabilities and usefulness of computers in their chosen fields. A computer is more than a glorified calculating device. New applications are being discovered every day, and the uses of computers are limited only by the imagination of users. Therefore, the administrator must learn how the computer may work for him, rather than how computers are being used at present or what the capabilities of certain systems are.

This does not mean that he has to become an expert programmer, but it does mean that he should have some basic programming knowledge. Computer programming requires a rigorous analysis and structuring of the problem to be solved, which leads to an enhanced ability to recognize the type of problem that may be solved by computer. We have emphasized two aspects of programming that usually receive less attention in more scientifically oriented tests.

(1) There is stronger emphasis on program documentation. The flowchart especially is stressed as a major tool for both analysis and documentation. In scientific applications, a few mathematical formulas may concisely explain a program. In data processing, a detailed explanation of many steps is more typical, and is necessary for program control.

(2) There is also greater emphasis on programmed checks to assure validity of results. In manual data processing equipment, many data errors are recognized and eliminated on sight. In automatic data processing, possible data errors must be anticipated by the programmer, who must then provide for their elimination from

the system. Such controls are especially necessary when large volumes of data are to be processed.

The organization of this text differs considerably from many general purpose Fortran introductions. The typical arrangement is to have chapters on the basic functions—flowcharting, arithmetic instructions, input-output instructions, control instructions, specification instructions, and subprograms. We have tried to integrate most of these functions and to present combined material in order of relative increasing difficulty.

The actual running of problems on the computer is highly desirable, but very time consuming for both students and instructors. Programming courses often include a laboratory period to allow for problem programming activity. The authors' preference is that no regular laboratory be scheduled, but that released time for individual consultation be made available as it is needed. It is recommended that a problem from each chapter be *successfully* programmed by each student.

The illustrative examples and suggested problems are mostly drawn from data processing applications. Both the examples and problems are intended to illustrate the various programming concepts under discussion at each stage of development, and they should not be interpreted as models of the most appropriate procedure possible. Some background in introductory accounting, algebra, and statistics will be helpful but is not necessary. Many of the suggested problems intentionally do not have suggested data. This has been done because development of data to test a program's correctness is considered a basic part of programming skill.

Some users may wish to study library functions before they are covered in the text. Library functions are discussed in the first section of Chapter 5. This section can be understood without too much difficulty after Chapter 2 and with ease after the first half of Chapter 3. With this exception, it will generally prove necessary to follow the regular order of presentation for full comprehension. It is desirable, but not necessary, for each student to have a reference manual of the Fortran system on which any programs are to be run. The text is not intended to be a reference manual, and only the most general instructions are included from the dozens of Fortran dialects available. Instructions which do not conform to U.S.A. Standard Fortran are relegated to appendixes, with the exception of the popular single quote for Hollerith messages.

In many instances, specific instruction limitations for many computers are less restrictive than those detailed in this text. It is recommended that the given text restrictions be followed if there is any possibility that problems written may later be run on differing computer systems. (The biggest differences will be found between Fortran systems for large- and small-scale computers and between computer systems of various manufacturers.) Standard usage is generally the best policy.

A variety of more advanced and peripheral topics has not been included, and a user may supplement the text in many ways. Discussions

of numerical error analysis, alternative number systems, magnetic tape handling, disk-type storage devices, real-time processing, decision tables, symbolic and absolute machine languages, feasibility studies, information retrieval, typical hardware and software offerings from manufacturers, symbolic logic, monitor systems, sorting and merging methods, report generators, and overall system studies have not been included. It would be impossible to cover all of these topics in meaningful depths in a programming course, and the authors would be interested in reactions from users as to what additions might be most helpful. We have included a chapter on simulation as a basic computer-dependent tool in administrative analysis. No new instructions are contained in it, however, so instructors may consider this material optional.

The material in this text has been freely adapted from an earlier book, Malcom and Gotterer: *Computers in Business* (International Textbook Company, 1967. We feel that this book benefits from the undergraduate and graduate courses we have taught at The Pennsylvania State University and Lehigh University during the intervening years. Acknowledgment is due many persons for making the courses and this text possible. While it is impossible to name all those who have played a part in the development of this book, we would like to thank the following members of The Pennsylvania State University staff: Professor Preston C. Hammer, Head of the Department of Computer Science; Ossian R. MacKenzie, Dean of the College of Business Administration; Professor G. Kenneth Nelson, Head of the Department of Accounting and Management Information Systems; and Professor Donald T. Laird, Director of the Computation Center. Members of the Lehigh University staff who made important contributions are Professor Robert H. Mills, Chairman of the Department of Accounting and Robert E. Pfenning, Lecturer in the Department of Accounting.

Thanks are also due to several graduate assistants who have contributed to the exercise and problem material and to colleagues and reviewers who have offered many helpful suggestions. Any shortcomings, of course, are entirely the responsibility of the authors. Comments and suggestions for improvements are welcomed from readers.

Contents

1 Introduction

> If there is one idea which I would like to leave with you today, it is that the computer represents an important extension of the powers of the mind of man. When the history of our age is written, I think it will record three profoundly important technological developments: *Nuclear energy,* which tremendously increases the amount of *energy* available to do the world's work; *Automation,* which greatly increases man's ability to use *tools*; and *computers,* which multiply man's ability to do mental work. Some of our engineers believe that of these three, the computer will bring the greatest benefit to man.[1]

As with all prophecies, only time will tell whether or not Mr. Cordiner was right. Some of his contemporaries thought that half a dozen or so computers would supply all the computing power the United States could ever use, whereas others foresaw that, within a decade, all large American businesses would be run virtually automatically by computers. Already, both viewpoints are in error. Nevertheless, the computer industry is one of the most dynamic in the world, and if the first decade was too short a time to do away with business middle management, each succeeding year has seen a dramatic change in the uses to which computers are being put, in both business and science.

Basically, a computer may be thought of as a high-powered calculating machine, but, as the term is most commonly used, a computer has three distinguishing characteristics. First, it has the ability to compare values and thus to make decisions. Second, it has an internal memory device to store both data and a program—a series of instructions about what to do with the data. Third, it operates electronically at a very high rate of speed. Unless the term is qualified, computers are also assumed to

[1] Testimony of Ralph J. Cordiner, then President of the General Electric Company, before the Subcommittee on Economic Stabilization, U.S. Congress Joint Committee on the Economic Report, *Automation and Technological Change.* Hearings before the 84th Congress, 1955, p. 444.

be digital (as opposed to analog, which work with continuous rather than discrete values) and general purpose (as opposed to special purpose machines, which can do only restricted jobs, such as handling inventory counts).

The first machine to meet the above criteria is generally considered the Eniac (meaning electronic numerical integrator and computer), announced by the U.S. Army on Feb. 16, 1946. Designed and constructed the previous year, the machine was the development of J. Presper Eckert, an electronics engineer, and John W. Mauchly, a mathematician, both of the University of Pennsylvania. The computer really has no inventor, as such, for it evolved from a series of contributions by several persons, over a period of years.

The Eniac was built to compute firing tables for artillery pieces, a job requiring hundreds of people working several months on desk calculators. With the Eniac, 30 seconds were needed to compute the trajectory of a shell, a computation requiring 20 hours under the former method.

The Eniac led to the development of the Univac (meaning universal automatic computer). This was the first commercially available general purpose machine and, for a time, was synonymous with the word *computer*. The first Univac was delivered to the United States Bureau of the Census in April 1951, for use in tabulating population data. Subsequent machines were taken by the Air Force for budgeting purposes, by the Army Map Service for transferring survey control points from a military system to the universal earth-coordinate system, and by the Atomic Energy Commission for scientific work.

A significant milestone in commercial data processing was reached by General Electric on October 22, 1954, when it began payroll processing on a Univac computer. Although insurance companies had made earlier use of computers, this was the first application to daily business operations with repetitive deadlines to be met.

By today's standards these computers were quite slow. The early machines relied upon thousands of vacuum tubes in their electronic circuitry, and sometimes upon semimechanical devices for their memories. Typical instruction times were measured in thousandths of a second, called milliseconds (msec). Memory size was measured in thousands of characters. In retrospect, they are referred to as first generation computers.

A most significant improvement came with the development of transistors to replace the vacuum tube circuits. At the same time, magnetic core memories, also solid state in nature, were produced. Both of these developments made computers faster and more reliable, and typical second generation operating speeds were measured in millionths of a second, called microseconds (μsec). Memory units generally accomodated several hundred thousand characters.

Currently, computers are in the third generation, and fourth generation machines are being developed. Transistors have been replaced by

integrated circuits, an outgrowth of space research activities. Operating speeds of the large machines are now measured in billionths of a second, called nanoseconds (nsec). Memory units may contain millions of characters. Whereas each computer manufacturer formerly had a line of unrelated machines for large- and small-scale, business, and scientific use, the new lines are interrelated and have similar instructions and operating characteristics. This permits several machines to be more easily interconnected into a large system, so computer systems can be enlarged by adding more units rather than a completely different, larger machine (a building block approach). More complex controls also allow priorities to be assigned to different operations; thus computers may serve many users simultaneously, by a process called time-sharing. Perhaps of even more significance is the fact that as computers have been improved in design, the cost of computation has declined sharply. Thus the use of computers continues to grow at a very rapid rate.

A diagram of a computer memory unit is shown in Exhibit 1-1. The unit is divided into many locations, each with a numeric label or address. An analogy is often made between computer memory units and numbered post office boxes serving as receptacles for letters. The number of the post office box corresponds to the memory address, and the letters inside represent the stored data. Although each location has a numeric address, some other arbitrary label is often used for the convenience of programmers. Thus, in the diagram, location 1000 is arbitrarily assigned the label *A*, and location 1005 is assigned the label *X*. These arbitrary assignments can be made by the computer itself, so the programmer can refer to locations in memory by a symbolic name, without ever knowing the absolute numeric location used by the machine.

As shown in Exhibit 1-1, the instruction in the location labeled *A* is "Read a number from a card and store it in a location labeled *X*." The next instruction says to read another number and store it in a location labeled *Y*. Then subsequent instructions say to add these two numbers together, label the answer *Z*, and print the answer on a paper form. These instructions, as a group, constitute an example of a program.

Of course, a computer has more than just a memory unit. Logically, computers usually consist of five elements: (1) an input section to get data and instructions from external sources into the memory unit; (2) a control unit to interpret the meanings of the instructions; (3) an arithmetic device to carry out the instructions; (4) a main memory unit; and (5) an output unit to communicate the answers from the machine to humans.

Another special name is used for the control, arithmetic, and main memory units. Collectively they are referred to as the *central processing unit,* or CPU. This unit is diagrammed in Exhibit 1-2. (We are distinguishing here the CPU memory unit from other input/output devices, which in fact also store data, and are therefore *memories.*)

All of the units just described are referred to as computer *hardware,*

Exhibit 1-1 *A Computer Memory Unit*

	1	
	2	
	3	
	...	
	...	
A	1000	"Read a number from a card and store it in a location labeled X"
B	1001	"Read another number and store it in a location labeled Y"
C	1002	"Add the number labeled X to the number labeled Y"
D	1003	"Put the above answer in a location labeled Z"
E	1004	"Print out the number in the location labeled Z"
X	1005	"5"
Y	1006	"3"
Z	1007	"8"
	...	
	...	
	2000	
	2001	
	2002	
	...	
	...	

Programmer Labels for Memory Locations (Arbitrary Choice)

Computer Labels for Memory Locations

Symbols in Each Memory Location Which Can Represent Instructions To Be Performed or Data To Be Manipulated

as they are physical entities. Computer hardware, however, does not process data by itself. Complex sets of instructions or programs are required to make it go. These programs are referred to as *software.* Much software is furnished by the hardware manufacturers, but additional software is also produced by the organization using the computer.

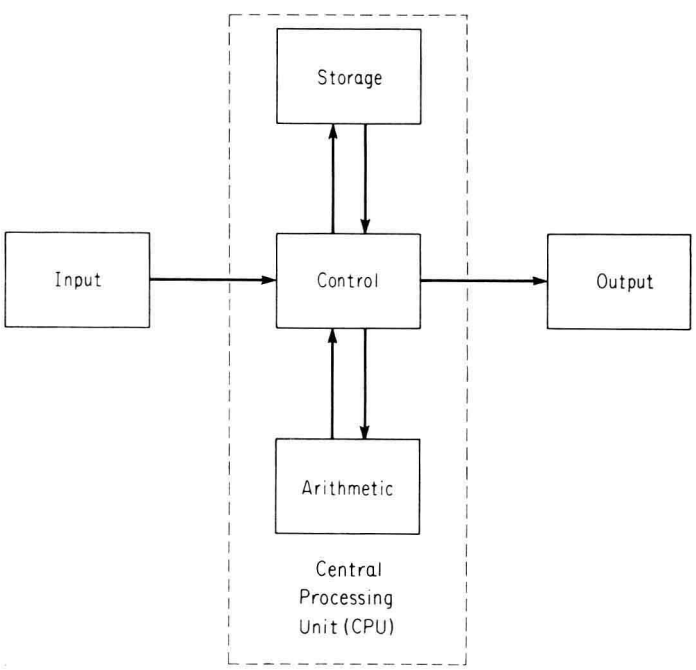
The computer component can perhaps be better understood by using a common analogy. Consider a clerk using a calculating machine to prepare payroll checks. Input data might be in the form of clock cards

indicating the time worked, the pay rate, and the foreman's approval. The control unit would be the clerk himself, interpreting which data were hours and which were rates and what should be multiplied and what should be added. The arithmetic unit would be the calculating mechanism, performing additions, subtractions, multiplications, and divisions. To add two numbers, the clerk would decide which buttons to operate (control), and then the machine would do the work (arithmetic). The output might be in the form of written checks and a log of the work performed. The internal memory unit would be the keyboard and dials of the machine (from which input data and answers were read), and the brain of the clerk (in which the instructions were remembered).

1-1 DATA REPRESENTATION

Whereas, with the brain, humans can remember hundreds of different characters or symbols and what they represent, computers understand only two. For example, our decimal system uses 10 characters, called digits, which are 0 1 2 3 4 5 6 7 8 9. The English alphabet uses 26 characters, or letters, which are A B C D E . . . X Y Z. We also use many special characters, such as + − , . * ( ) $ = ' b / (where b indicates a blank space). All the characters just specified in fact constitute the character set

Exhibit 1-2 *Computer Organization*



used in the Fortran IV computer language.[2] The computer itself recognizes only 1 and 0.

The unit which stores the 1 or the 0 in a computer is referred to as a *bit* (for binary digit). It is the smallest level of data representation. Generally, bits by themselves are not very useful, although in symbolic logic problems the 1 may represent a true condition and the 0 a false condition, and, for data representation, this is sufficient. Groups of bits, however, may be developed into elaborate coding systems, so that as many different characters as desired can be represented.

One widely used code is the Extended Binary Coded Decimal Interchange Code (EBCDIC). Selected character representations for this code are shown in Exhibit 1-3.

The parity bit is not part of code, but a special bit used to check computer accuracy. In Exhibit 1-3 each character always has an odd number of 1s, or odd parity. Even parity is also used in some systems.

Note that with one bit only two characters can be represented. With two bits there are 2^2, or 4, possibilities—00, 01, 10, 11. In the Fortran IV character set of 10 digits, 26 letters, and 12 special characters, 48 bit combinations are needed, and the smallest power of 2 to handle this is 2^6, or 64 (as 2^5 is only 32, and not sufficient). A 6-bit binary coded decimal notation (often abbreviated BCD) was in fact used with the first Fortran language. The 8-bit code allows for encoding small as well as capital

Exhibit 1-3 *Selected Character Representations for EBCDIC*

		Bit Number							
Character	*Parity*	0	1	2	3	4	5	6	7
A	0	1	1	0	0	0	0	0	1
B	0	1	1	0	0	0	0	1	0
C	1	1	1	0	0	0	0	1	1
D	0	1	1	0	0	0	1	0	0
. . .									
X	1	1	1	1	0	0	1	1	1
Y	1	1	1	1	0	1	0	0	0
Z	0	1	1	1	0	1	0	0	1
0	1	1	1	1	1	0	0	0	0
1	0	1	1	1	1	0	0	0	1
2	0	1	1	1	1	0	0	1	0
. . .									
8	0	1	1	1	1	1	0	0	0
9	1	1	1	1	1	1	0	0	1
+	1	0	1	0	0	1	1	1	0
−	1	0	1	1	0	0	0	0	0
b	1	0	0	0	0	0	0	0	0
. . .									

[2] In some Fortran dialects the $ is arbitrarily treated as a letter rather than as a special character. It is always permissible to use the $ as a special character, but it is not always permissible to use it as a letter. It will be treated in this text as a special character.

letters, as well as many other special characters. It also has unassigned spaces for future expansion. Another widely cited 8-bit code is the U.S.A. Standard Code for Information Interchange-8 (USASCII-8).[3]

Many other arbitrary coding schemes have been devised. Computers sometimes do calculations with digits represented in some BCD form, but often another representation is used for digits only. This involves changing number representations from the decimal, or base-10, system, to the binary, or base-2, system. This is illustrated in Exhibit 1-4.

Calculations can be done much faster in pure binary form, but the data going into and out of the machine must be translated to decimal form for most people to read. Decimal processing has therefore been most widely used in business, where input and output are relatively large. Binary processing has been used most widely for scientific calculations, where input and output are relatively small. Many computers will operate

Exhibit 1-4 *Number Representations*

Decimal

Thousands = 10^3	Hundreds = 10^2	Tens = 10^1	Units = 10^0
1	2	0	5
5 X 1	=		5
0 X 10	=		00
2 X 100	=		200
1 X 1000	=		1000
			1205

Binary

One-thousand-twenty-fours = 2^{10}	Five-hundred-twelves = 2^9	Two-hundred-fifty-sixes = 2^8	One-hundred-twenty-eights = 2^7	Sixty-fours = 2^6	Thirty-twos = 2^5	Sixteens = 2^4	Eights = 2^3	Fours = 2^2	Twos = 2^1	Units = 2^0
1	0	0	1	0	1	1	0	1	0	1

1 X 1	=	1
0 X 2	=	0
1 X 4	=	4
0 X 8	=	0
1 X 16	=	16
1 X 32	=	32
0 X 64	=	00
1 X 128	=	128
0 X 256	=	000
0 X 512	=	000
1 X 1024	=	1024
		1205

[3]The American National Standards Institute (ANSI) is an association promoting design uniformity in many fields. It has formerly been known as the U.S.A. Standards Institute and the American Standards Association (ASA).

on data in either form, and even binary computers can handle nonnumeric data in coded form.

We have now compared information representation at two levels (bit and character) for two media—the English language and the computer. Let us now consider a third widely used medium in data processing—the punched card. Several such cards are illustrated in Exhibit 1-5. They are frequently called IBM cards or EAM cards. IBM refers to International Business Machines Corporation, whose coding system is the most widely used, and EAM to electric accounting machine, a noncommercial synonym promoted by the federal government. Yet another name is Hollerith card, after the originator, Herman Hollerith.

Punched cards are divided horizontally into 80 columns and vertically into 12 rows, where holes may be punched to indicate information. The 10 bottom rows are used to indicate the digits 0 through 9. The top row is called the 12 or Y row, and the second row is called the 11 or X row.

Numbers are represented in any column by punching out the hole in the row of the desired value. Alphabetic characters are formed, in code, by punching out two holes at a time in the same column. One hole is always from the top three rows, sometimes called zone punches. Special characters are represented by other combinations of punches, which are not now completely standardized.

By taking a hole to represent a 1 and no hole to represent a 0, punched cards are seen as also coding data from binary bits. Thus, a decimal 1 is represented in card coded binary (from top to bottom) by 000100000000, and a decimal 2 is represented by 000010000000. An *A* is represented by 100100000000 (or 12 and 1), *B* by 100010000000 (or 12 and 2), *L* by 010001000000 (or 11 and 3), *S* by 001010000000 (or 0 and 2), and *Z* by 001000000001 (or 0 and 9).

Card columns can be arbitrarily grouped together to form what are called *fields*. Numbers or characters within a common field are treated as a single unit. Field sizes may vary from a column to the entire card. In Exhibit 1-5, the second card is split into five name and address fields—a name field of columns 1 through 20, a street address field of columns 21 through 40, a city field of columns 41 through 60, a state field of columns 61 through 74, and a zip code field of columns 75 through 80. The third card is split into eight numeric fields of 10 columns each. As can be seen, fields 4, 5, and 6 run together. As long as it is precisely known where fields begin and end, fields can be made continuous, without blank spaces between them.

In the English language, we also group together related characters, but the term *word* is used rather than *field*, and words are separated by blanks and punctuation rather than being allotted a certain physical space. Note that English words and punched card fields do not necessarily have exact correspondence. The last name, first name, and middle initial are treated as separate words in the English language, but they comprise a single unit name field on the punched card example.