INTRODUCTION TO FORTRAN IV

SECOND EDITION

ROBERT H. HAMMOND

WILLIAM B. ROGERS

BYARD HOUCK, Jr.

INTRODUCTION TO FORTRAN IV

SECOND EDITION

ROBERT H. HAMMOND
Associate Professor of Engineering
Director, Freshman Engineering
and Student Services Division
North Carolina State University

WILLIAM B. ROGERS
Professor of Engineering Fundamentals
Virginia Polytechnic Institute
and State University

BYARD HOUCK, Jr.
Senior Advisor in Engineering
North Carolina State University

McGRAW-HILL BOOK COMPANY

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

INTRODUCTION TO FORTRAN IV

2 3 4 5 6 7 8 9 0 DODO 7 8 3 2 1 0 9 8

Library of Congress Cataloging in Publication Data

Hammond, Robert H.
Introduction to FORTRAN IV.

Includes index.
1. FORTRAN (Computer program language) I. Rogers, William B., date joint author. III. Houck, Byard, joint author.
QA76.73.F25H36 1978 001.6'424 77-6430
ISBN 0-07-025897-X

This book was set in Univers Medium by Hemisphere Publishing Corporation.
The editor was B. J. Clark;
the cover was designed by Scott Chelius;
the production supervisor was Donna Piligra.
The printer and binder was R. R. Donnelley & Sons Company.

PREFACE TO THE SECOND EDITION

The ink is scarcely dry on the pages of a new textbook before the author's telephone begins to ring. Congratulations on the new publication? Perhaps, but more than likely it is a gleeful colleague with the good news, "Say, I found a mistake in your book." Mistakes in typography and other minor aberrations overlooked in the final proofreading are an annoying and embarrassing, but inevitable, fact of life. One can only hope to reduce such blunders to an acceptable minimum and strive to locate and correct as many as possible in a second or third printing.

When the availability of a textbook in the used book market significantly affects new book sales, a new edition is indicated. There are, of course, other cogent reasons, more pedagogic than economic, for a complete revision. Among these reasons are obsolescence and the availability of new information or data; the discrediting of old and the accepting of new hypotheses; new discoveries in the state of an art or craft; and the need indicated by users to expand existing information, add new topic areas, and improve and clarify the presentation of existing material. It is this last reason that has persuaded the authors and publisher to undertake the substantial task of preparing a second edition. Students have convinced us that some explanations were less than crystal clear or stopped short of answering all the obvious questions. Instructors have expressed a need for material not included, more detailed coverage in certain areas, and more illustrative examples and problems.

Specifically, some subject areas have been rearranged or combined under a single chapter heading in a more logical progression or grouping. We have added computed GO TO, character output using the PRINT statement, character input with the DATA statement and character manipulation, and a more detailed explanation of paper control. Discussion of arrays and subscripted variables has been expanded. More illustrations and sample problems have

been added, and the number of problems in each chapter has, in most cases, been almost tripled. We hope that we have not done too much, that we have not defeated our original purpose of presenting the elements of the FORTRAN language in a simple, straightforward, and easy-to-understand fashion. The authors realize that all instructors may not want to include every page, section, or paragraph in their initial assignments. But when mistakes and unexpected results occur in programming, specific referral can be made to explain what happened and how to correct the situation.

We have carefully considered and greatly appreciate the suggestions for improvement made by our colleagues and students, and hope that we have adequately met the requirements of the majority of the users in this second edition. In particular, we thank Professor Parriz F. Rad, Clemson University, and Professor Robert D. LaRue, The Ohio State University, for their painstaking review of the manuscript and their valuable comments. To both new and old users who stand ready with reading glass and dictionary, good hunting.

Robert H. Hammond
William B. Rogers
Byard Houck, Jr.

PREFACE TO THE FIRST EDITION

The publication of a new textbook dealing with the FORTRAN computer language might aptly be compared with the arrival of another starling on the horizon to join the flock already roosting in the town park—and greeted with about the same interest and enthusiasm! Professors write textbooks for a variety of reasons: to supplement their salaries; to satisfy a pressing institutional requirement to publish or perish; to impress other professors with their erudition; to gratify their egos; and perhaps once in a great while to help a few less-than-brilliant students to improve their understanding of the rudiments of a specific subject area. The authors plead nolo contendere to any and all of these spurious incentives but sincerely hope that their primary motivation has been to provide a usable textbook which will simplify the fundamentals of FORTRAN for the average college student.

The authors do not claim to be authorities in the field of computer programming or even experts in the use of the FORTRAN language. This brief exposition is not a definitive text, nor is it a comprehensive reference book. In fact, the authors have not even discussed the subject to the extent of their own limited knowledge. The most obvious fault to those real or self-styled experts who take the trouble to critically examine what we have written will be oversimplifications in some areas and significant (by some opinion) omissions in others. To these oversimplifications and omissions, we plead guilty.

Why do we presume to drench an already saturated area with still more verbiage? Because we believe we have said it so the beginning student can understand it. For many years the authors have taught elementary engineering subjects, including FORTRAN programming, to thousands of freshmen at the United States Military Academy, North Carolina State University, and the Virginia Polytechnic Institute and State University. At the same time, we have taught many novice professors the techniques of classroom presentation and

have supervised their subsequent performance. If there is any field in which the authors may modestly claim expertise, it is in the forced-feeding of required technical foundation subject matter to large numbers of generally indifferent beginners.

It is for this mass audience of indifferent beginners that this textbook is written. The authors have attempted to reduce the often mysterious and seemingly complex world of the computer to its simplest elements. The terminology has been purged of unnecessary "computerese." Both the language and the mathematics should be readily understood by anyone with a reasonable high school education and average verbal comprehension.

No attempt has been made to present everything there is to know about FORTRAN. Enough is covered to permit the student to understand the fundamental techniques of the language and, with practice, to write meaningful programs. In general, only one way of accomplishing a desired result is discussed. Options, exceptions, shortcuts, and exotic statements or routines not likely to be encountered by the beginner have been avoided. The text material has been implemented by illustrations which provide a brief graphic summary of fundamental instructions. Flow diagrams and computer printouts illustrate applications and the results obtained from each routine discussed. Illustrative problems previously introduced are again used in succeeding chapters so that the student can concentrate on the new FORTRAN instruction being discussed without also worrying about the concepts in a new problem. The problems given at the end of the various chapters are basic in concepts so that the student can concentrate on applying the FORTRAN principles and not be overcome by the concepts involved in the problem. When a student has demonstrated an understanding of FORTRAN, the teacher may assign more advanced problems that involve the teacher's interests and objectives.

This textbook is written primarily for students at institutions where the WATFIV compiler for FORTRAN IV is used. There are a few discussions that apply only to WATFIV users. However, if WATFIV is not the compiler used, the teacher can eliminate those discussions and adapt the other discussions to fit other compilers.

The authors have enjoyed some success in their classes with this material in the form of handouts, projecturals, and chalkboard sketches. We now offer it to you, the average beginning student, with the hope that it will ease somewhat the process of learning this fascinating FORTRAN.

And, to you, professor: There should be sufficient material in this textbook to keep both you and your students gainfully occupied for one full semester. Or, if you talk fast, it can be covered in one quarter.

Robert H. Hammond
William B. Rogers
Byard Houck, Jr.

CONTENTS

1 WHAT IS A DIGITAL COMPUTER?

1-1 INTRODUCTION

There are two types of computers in wide use today: digital and analog computers. The *digital computer* is essentially a counting device and operates with numbers represented by a finite sequence of digits. The *analog computer* operates by measuring the magnitudes of the quantities in an electric circuit which is set up to parallel (or be analogous to) the equation of the phenomenon being investigated. Analog-computer results are usually displayed as a curve on a cathode-ray tube, and numerical quantities are not shown.

The needs of modern-day computations have led to the increasing use of a combination of digital and analog computers. This combination is known as a *hybrid computer.* A detailed discussion of the use of hybrid computers, as well as any discussion of analog computers, is outside the scope of this text, but all students should be aware of their existence.

1-2 THE BASIC COMPONENTS OF DIGITAL COMPUTERS

A general-purpose digital computer consists basically of five components or functional units: input, storage, arithmetic, control, and output units. The relationship of these functional units is represented by the block diagram in Fig. 1-1. Information is interchanged between these units as indicated by the arrows.

a The Input Unit The *input unit* places the desired instructions and data into the storage unit. A commonly used input device is the *card reader*, illustrated in Fig. 1-2. A card reader senses the holes punched in a computer card and transmits the information punched in the cards to the storage unit. This is the

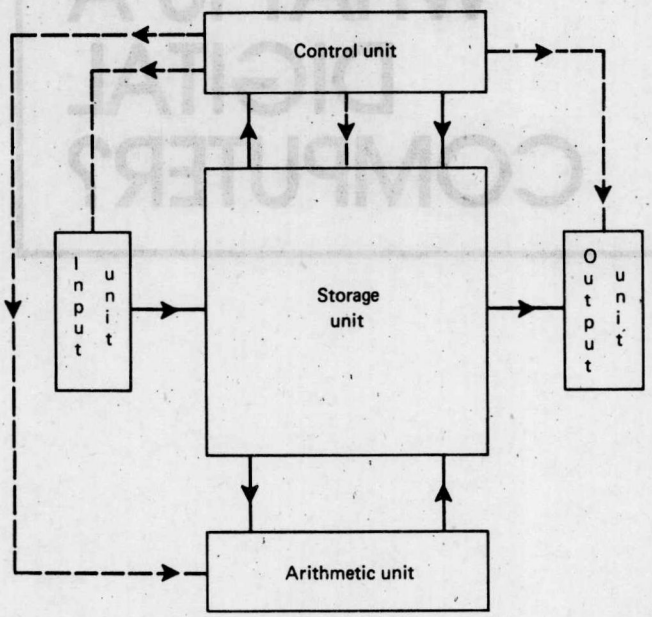

FIG. 1-1
Basic components, or functional units, of a digital computer. Solid lines represent flow of information (data), dashed lines represent flow of control signals.

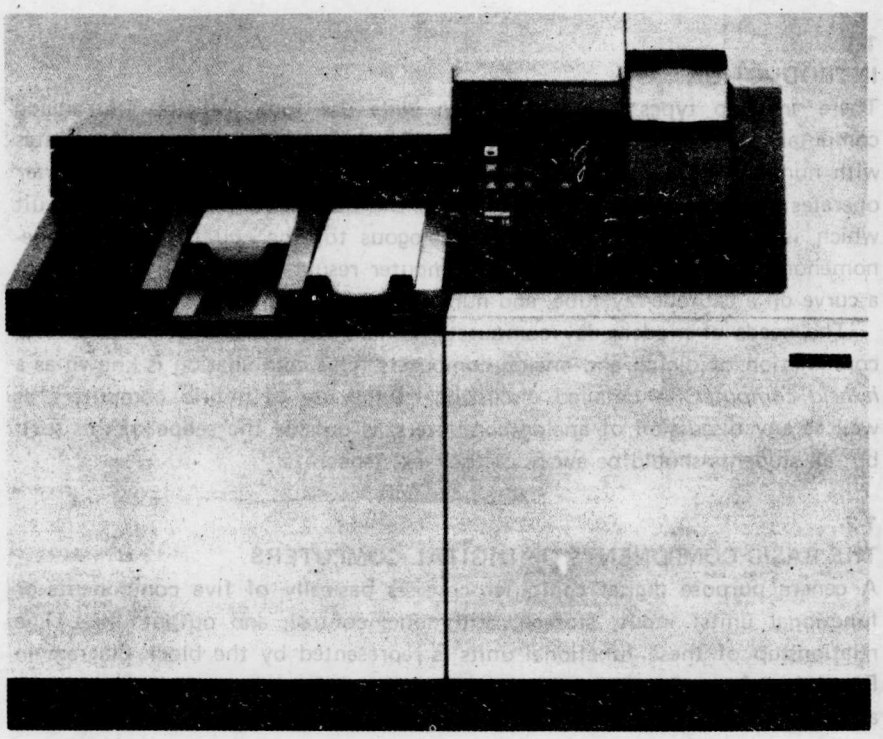

FIG. 1-2
IBM 3505 card reader. (*Courtesy of IBM.*)

only input device that will be discussed in this text. However, other input devices that may be available include the typewriter, the tape drive, and punched paper tape. The input unit desired is specified by an appropriate code number in the input instructions.

FIG. 1-3
IBM 029 card punch. (*Courtesy of IBM.*)

Before using the card reader, blank computer cards must first be punched on a machine called a *card punch*, illustrated in Fig. 1-3. Brief instructions covering the basic operation of the IBM 029 card punch are given in Appendix B. The keys of the card punch are similar to the keys of a standard typewriter; however, pressing them causes one or more holes to be punched in the card. The rectangular hole or pattern of holes represents the desired character.

b The Storage Unit The *storage unit* consists of many *core planes* (see Fig. 1-4). Each core plane is made up of a number of ferrite cores (or rings) strung on hair-thin wires. Dependent upon how current is passed through these wires, each ring can be magnetized with either a clockwise or a counterclockwise magnetic field. The rings are divided into groups of rings, with each group known as a "word." Each word has its own unique address which the computer knows. The size of a word can vary from a 1- to 10-digit number (or more), depending upon the capability of the individual computer. Computer users must always determine the word size of the particular computer that they will be using.

A distinctive feature of the storage unit of a general-purpose computer is destructive *read-in* and nondestructive *read-out.* Each time information is stored in a given word, the previous contents of that word are erased as the new data are read in. However, the contents of that word can be moved to another word with a different address without changing the contents of the original word. This feature is important to remember in preparing instructions

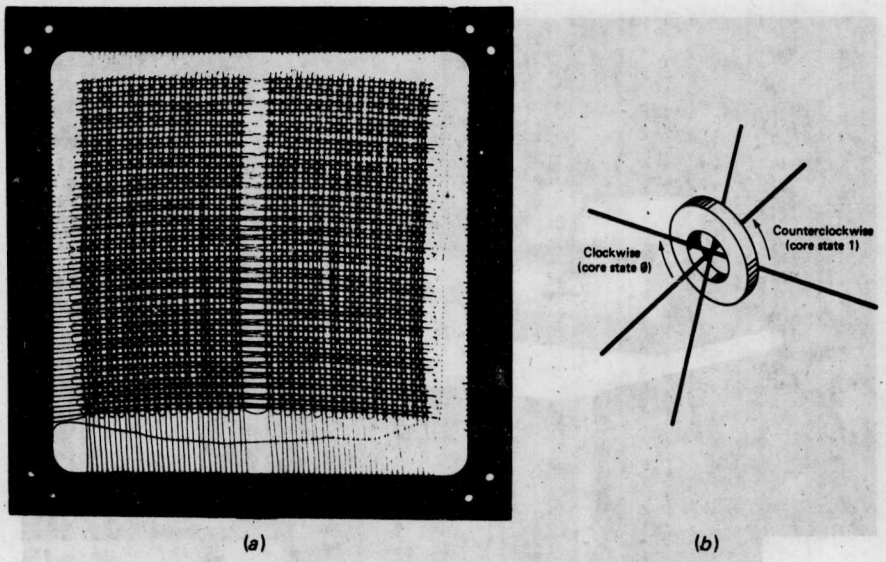

FIG. 1-4
(*a*) Core plane with ferrite cores visible on intersecting conductors. (***Courtesy of IBM.***) (*b*) Enlarged drawing of a core.

for the computer. A second characteristic of this type of storage unit is called *random access*, which means that any word address in the storage unit is as easy to find as any other address and takes the same amount of time.

Auxiliary storage capacity may be added by using magnetic tapes or discs. Access to information stored on tape is sequential. To retrieve a given item, the entire tape must be examined from some starting point to the location of the desired information. Thus, it may take more time to find one word than another. The magnetic disc is a cross between random and sequential access and requires less time to search than the tape. The discussion in this text will be confined to the basic random-access storage unit. When students advance to the point where auxiliary storage capacity is required for their programs, they should consult the systems manuals on file at the local computer center or refer to a more comprehensive computer textbook.

The storage unit is also known as the *memory unit.* The term *memory* is not used in this text because it implies the human capability to remember. To those unfamiliar with the computer, it also implies the ability to think. Both these implications are misleading. A computer does not remember, nor does it think for itself. The computer does precisely what it is instructed to do—nothing more, nothing less. The computer's sequence of actions and the results thereof depend solely upon the instructions it receives from a human programmer. An often used expression among computer people is "garbage in—garbage out," which means that unless the logic of the program is correctly planned to perform the desired calculations, the output will be meaningless or misleading.

c The Arithmetic Unit The *arithmetic unit* is a portion of the computer set aside for performing the basic arithmetic operations: addition, subtraction, multiplication, and division. It also provides temporary storage for holding the

results of these operations. This small storage unit is known as the *accumulator.*

d The Control Unit The *control unit* is the heart of the modern digital computer. It selects an instruction and causes the computer to obey that instruction whether it be to read a data card, perform some arithmetic operation, compare two values, or print results. The control unit consists primarily of two parts: a small storage unit known as the *instruction register* (IR) and a device called the *instruction counter* (IC).

e The Output Unit Printed results from the computer may be obtained from the *printer.* This machine prints the pages of results commonly associated with computer systems and is illustrated in Fig. 1-5. Other output units punch cards, store information on magnetic or punched paper tape, utilize the typewriter on the computer console or remote terminal, or display results on a cathode-ray tube (CRT). The output unit desired is specified by an appropriate code number in the output instructions.

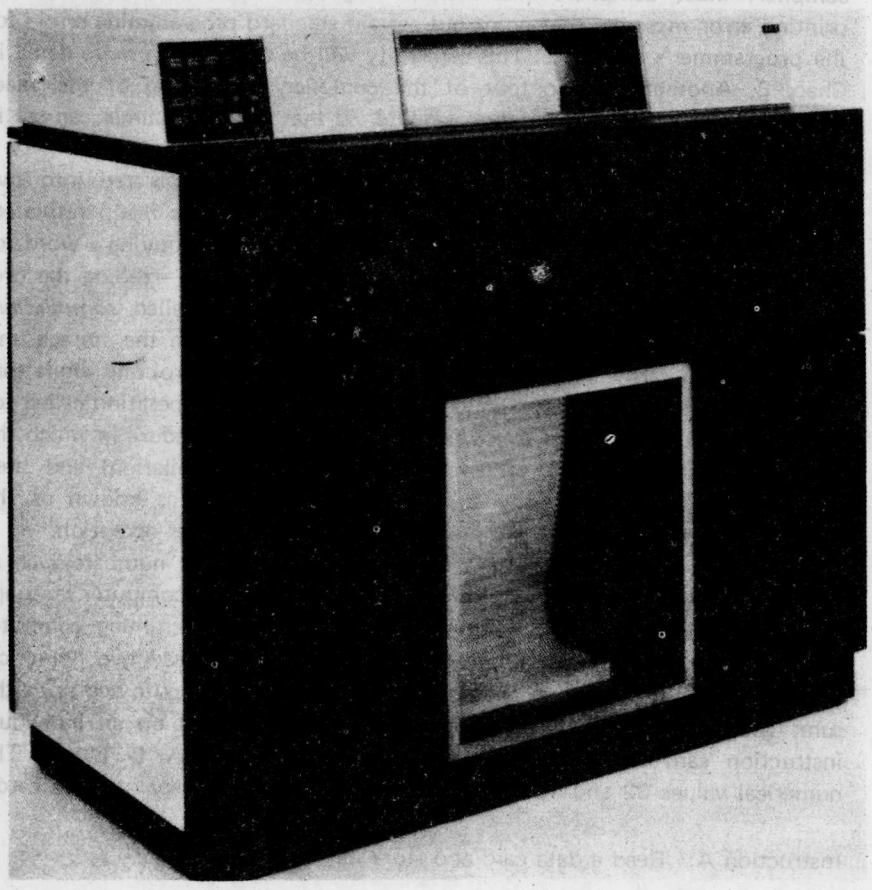

FIG. 1-5
IBM 3211 line printer. (*Courtesy of IBM.*)

1-3 THE STORED-PROGRAM CONCEPT

Before any problem can be solved on the computer, a set of step-by-step instructions must be written which precisely describe how to solve the problem. This set of instructions is called a *program*. The program must be written in a *language* which the computer can understand. Each step of the program is called a *statement*. Usually each statement is punched on a separate card called an *instruction card*. After all instruction cards have been punched and arranged in order, each item of numerical data is punched on a card. These cards are called *data cards* and follow the last instruction card. Depending upon how the program is written, each item of data may be punched on a separate card or many items of data may be punched on the same card.

In the preceding paragraph we stated that the program must be written in a language which the computer can understand. Actually, the computer can understand only *machine language*, a language which is complex and lengthy. The computer cannot understand directly the user-oriented FORTRAN language. Instead an intermediate program must be employed which can be stored in the computer and translate the user-oriented FORTRAN into the machine-oriented language of the computer. Such a program is called a *compiler*. Most compilers also include diagnostic routines which result in printing error messages that point out certain standard programming errors for the programmer's assistance. This capability will be discussed in more detail in Chap. 8. Another usual output of the compiler is a listing of the exact FORTRAN program read by the computer so that the programmer can see in printed form what was actually read from the punched cards.

The computer solution to a particular problem is separated into two phases. First, the entire set of instructions (or program) is read, translated, and filed in the storage unit, each individual instruction occupying a word (or words as required) with a distinct address; this first phase—reading the program, translating, and filing it in the storage unit—is called *compilation*. Second, each separate instruction is called sequentially from the storage unit and held temporarily in the instruction register of the control unit while that instruction is executed; this second phase—performing the operation called for in the instruction—is called *execution*. This two-phase procedure in which the computer first stores the program in its entirety (compilation) and then automatically and sequentially follows those instructions is known as the *stored-program concept* and is the essence of digital-computer operation.

Consider a simple arithmetic problem: the sum of two numbers such as 32 + 18 = 5Ø. Without attempting to simulate any actual computer language but using instructions understandable to the reader and assuming computer acceptability, the following *program* has been written to read two numerical values, 32 and 18, punched on two data cards, and to compute and print the sum. (Each instruction is assumed to have been punched on an individual instruction card and identified alphabetically by letters A, B, C, etc. The numerical values 32 and 18 have each been punched on a separate data card.)

Instruction A: Read a data card and store its value in address 2-1

Instruction B: Read the next data card and store its value in address 2-2

Instruction C: Copy the value in address 2-1 into the accumulator

Instruction D: Add the value in address 2-2 to the value in the accumulator

Instruction E: Store the value in the accumulator in address 2-3

Instruction F: Print the value in address 2-3

(End of Instructions)

Data: 32

Data: 18

Figure 1-6 represents a graphical simulation of the five functional units of the computer as described in Sec. 1-2. The storage unit provides for 30 "words" whose locations are specified by a two-digit numerical "address" identifying the row and column, respectively. (For example, the address of the bottom right space or word is 5-6, row 5, column 6.) The XX's in the storage unit, the instruction register (IR), instruction counter (IC), and the accumulator (Fig. 1-6*a*) represent miscellaneous information remaining in storage after completion of the previous program. When current information is read into storage, previously stored information will be destroyed. To start the se-

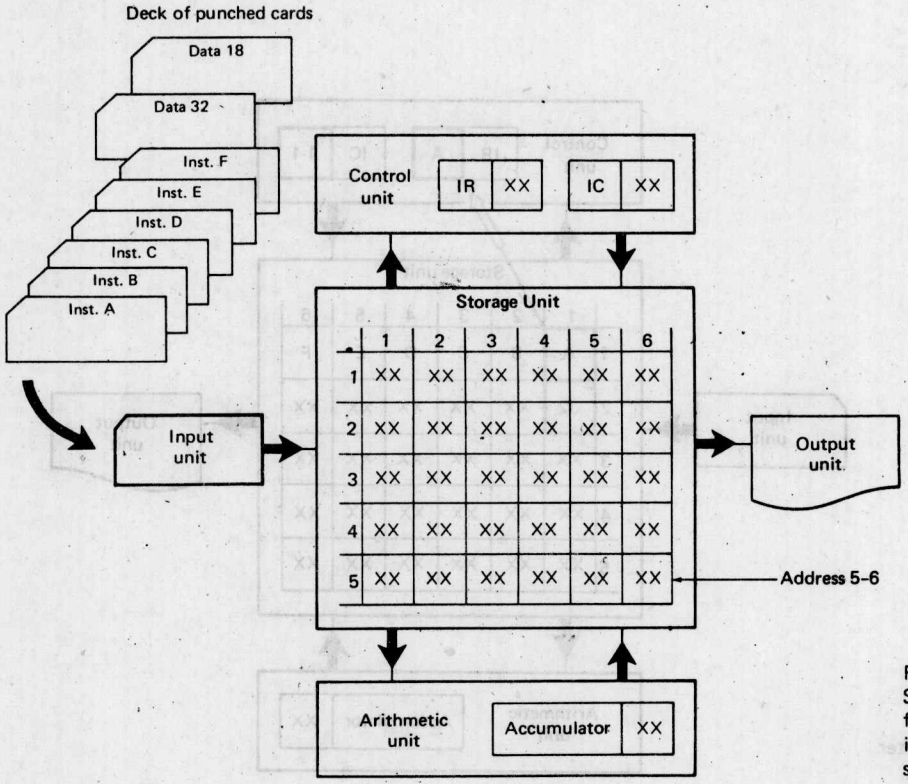

FIG. 1-6*a*
Simulation of computer ready for input of instructions. (IR, instruction register; IC, instruction counter.)

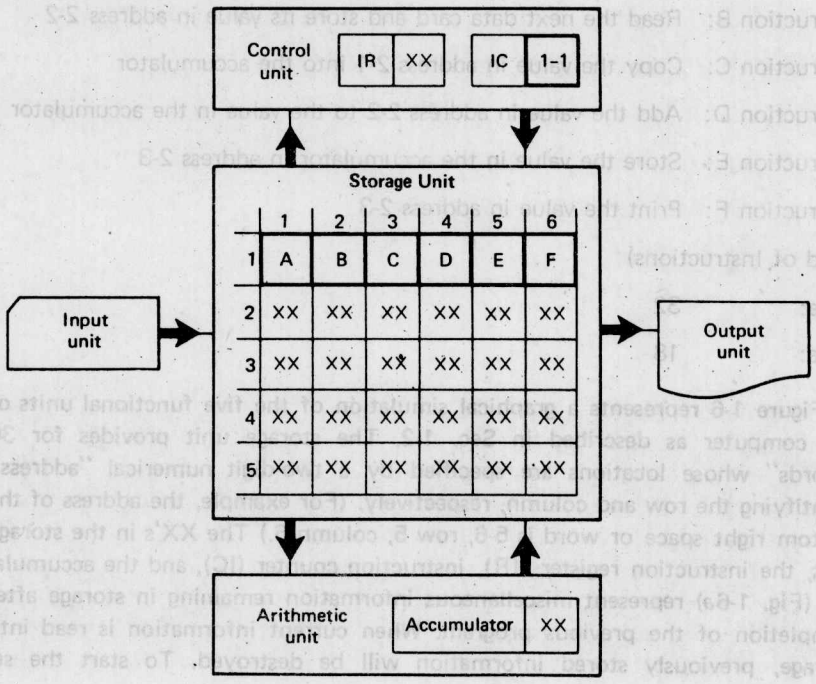

FIG. 1-6*b*
Simulation of computer ready for execution.

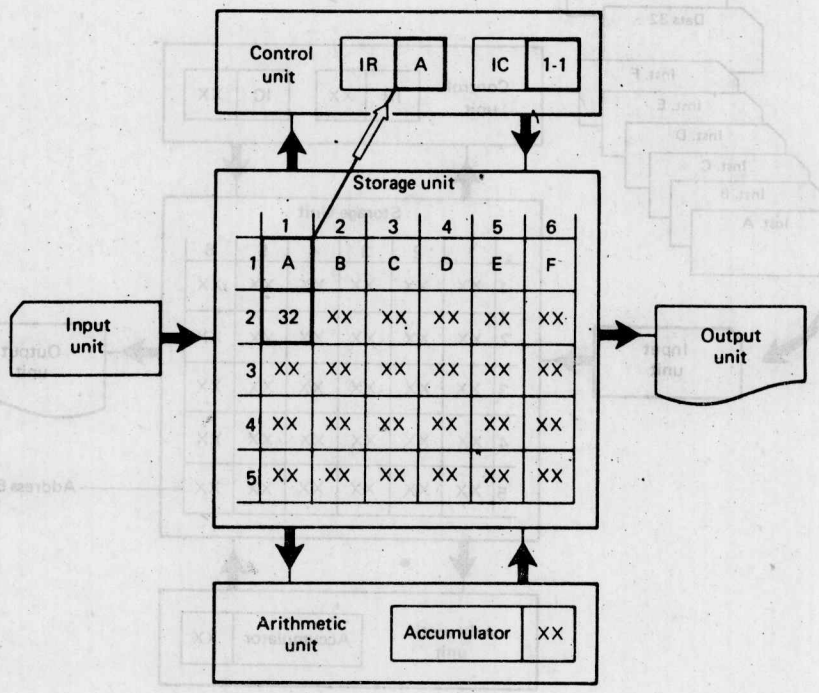

FIG. 1-6*c*
Simulation of computer after execution of instruction A.

quence, the card deck is placed in the card reader (as indicated in Fig. 1-6*a*), the "start" button is pressed, and action begins.

The condition of the computer after all instructions have been read in (compilation is complete) is shown in Fig. 1-6*b*. Instructions A, B, C, D, E, and F are stored sequentially in words with addresses 1-1, 1-2, 1-3, 1-4, 1-5, and 1-6, respectively. The information remaining from previous calculations (XX) has been destroyed in addresses 1-1 to 1-6 and replaced by the new and relevant information. Note that the instruction counter (IC) has been automatically set to the address of the first instruction (1-1). Execution of the program begins.

In Fig. 1-6*c*, the first instruction (instruction A in address 1-1) has been "copied" into the instruction register (IR), destroying the previous instruction (XX) but leaving instruction A unchanged in address 1-1. The computer then executes instruction A:

Read a data card and store its value in address 2-1.

The first data card in sequence is read, and the numerical value punched therein (32) is stored in word address 2-1. The IC automatically increments to the next address in sequence (1-2).

Figure 1-6*d* illustrates the execution of instruction B. The IC calls for the instruction in address 1-2 (instruction B) to be copied into the IR, destroying the previous instruction (instruction A). Note that instruction B remains unchanged in address 1-2. The computer then executes instruction B:

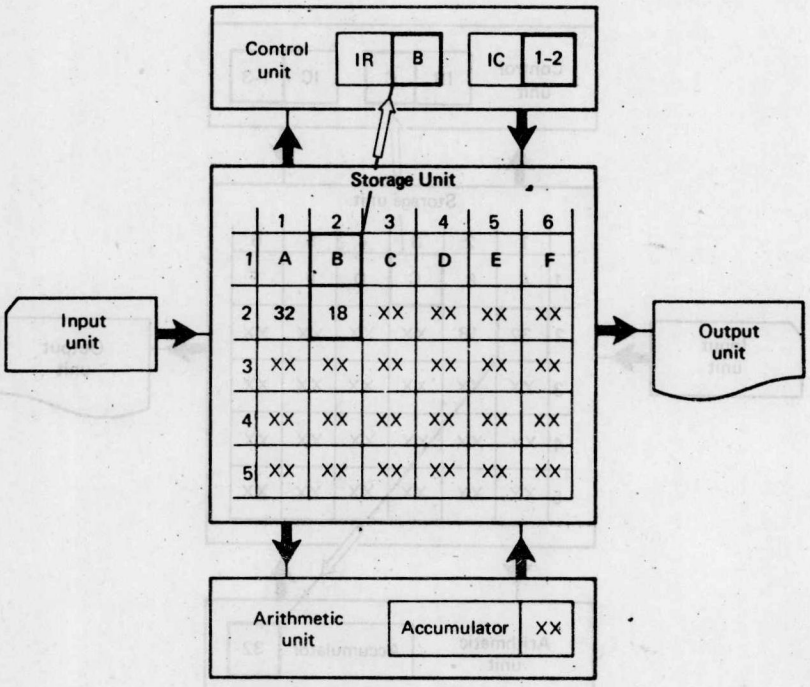

FIG. 1-6*d*
Simulation of computer after execution of instruction B.