INTRODUCTION TO FORTRAN IV

Third Edition

ROBERT H. HAMMOND

WILLIAM B. ROGERS

INTRODUCTION TO FORTRAN IV

Third Edition

ROBERT H. HAMMOND
Associate Professor Emeritus of Engineering
North Carolina State University

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

McGraw-Hill Book Company

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

INTRODUCTION TO FORTRAN IV

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

ISBN 0-07-025908-9

This book was set in Univers by Automated Composition Service, Inc.
The editors were Julienne V. Brown, Kiran Verma, and Madelaine Eichberg;
the cover was designed by Joseph Gillians;
the production supervisor was Phil Galea.
New drawings were done by J & R Services, Inc.
R. R. Donnelley & Sons Company was printer and binder.

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 II. Title.
QA76.73.F25H36 1983 001.64'24 82-10028
ISBN 0-07-025908-9 AACR2

PREFACE

Anyone presumptuous enough to write a textbook must have attained a knowledge of the subject matter significantly greater than the students for whom the textbook is written and should be both able and willing to organize and enunciate a limited amount of that knowledge in a manner readily comprehendible to those students. Unfortunately, increase in knowledge is often accompanied by a proportionate decrease in the author's ability (or willingness) to present the subject simply and directly in terms that are easily understood. This paradox often results in a poor textbook, which is more a showcase for the author's erudition, written to impress colleagues, than an effective instructional manual dedicated to explaining and clarifying the details of the subject for the uninitiated.

The authors of this text have certainly increased their individual and collective knowledge of FORTRAN since the publication of the earlier editions, and the temptation to display that new knowledge is great. We have also tried to remember that each beginning student opens the game at the same square one where all the preceeding students started and advances one square at a time toward the goal. Some, with prior knowledge or an aptitude for the game, learn rapidly and are able to advance faster, but this cannot be assumed. A beginning text must address the person meeting the subject for the first time in terms that are understandable to that person. Every square must be touched, and in some cases it is necessary to go back a few squares and repeat the moves.

A conscious and determined effort has been made in writing this book to organize and cross-reference the material so that the student is never left mystified by an unexplained term or concept. Many example programs are included. Some explanations, rules, and cautions are repeated to the point of redundancy. Experience with beginning students has demonstrated the need for such repetition.

This third edition contains virtually all the text material and illustrations from the first two editions, but most of the text has been reorganized and rewritten in what the authors believe to be a more easily learned and logical sequence. The essence of digital-computer operations and the essentials of FORTRAN programming are presented in the first two chapters. The chapters which follow build on the fundamentals established in Chaps. 1 and 2. Simple programming can begin with Chap. 2. The mechanics of card punching have been moved to Appendixes A and B. A separate chapter on the flowchart has been eliminated. The structure of the FORTRAN language is discussed in detail but within the context of the accompanying logic. The algorithm (flowchart) is not considered as a separate entity but parallel with the related FORTRAN language program. Each new routine is discussed both as a logical procedure and a program segment implementing that logic. Specific system requirements which are not generally applicable have been avoided.

Additions to the third edition include an explanation of the use of inverse trigonometric functions in obtaining true values for azimuths and other angles which may exceed

90°, expanded discussions of arrays and double-precision calculations, calculations with complex numbers, and the use of the T and G specifications in the FORMAT statement.

Most of the chapters include problems that test the techniques presented. Some problems are short and simple, emphasizing a programming technique. Others are longer and more complicated, requiring both application of the technique and a careful analysis of the logic of the solution. A few are relatively difficult. Most problems involve only the relationships of mathematics and science with which a freshman student should be familiar. A few problems require more advanced knowledge, and some research and/or instructor assistance may be needed. There should be sufficient problem material to keep the best students challenged for a semester or quarter course.

The authors acknowledge the valuable suggestions and criticisms provided by their many colleagues and students who have used the earlier editions of this textbook and the reviewers of the manuscript for this edition. Most of these reviewers are unknown to the authors, but to one, Dr. Leendert Kersten of the University of Nebraska, special thanks are due for his suggestions, which are embodied in Chaps. 4 and 10. Much credit for the timely completion of the manuscript is due to Mrs. Mary Cunningham, of North Carolina State University at Raleigh, who patiently typed and retyped the many drafts from the authors' notes.

Robert H. Hammond
William B. Rogers

CONTENTS

1 THE DIGITAL COMPUTER: BASIC CONCEPTS

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 discrete numerical quantities 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 often displayed as a curve on a cathode-ray tube with no discrete numerical quantities shown. The needs of modern computations have led to the increasing use of a combination of digital and analog computers, known as a *hybrid computer*. Analog and hybrid computers are not discussed in this text, which is limited to the general-purpose digital computer.

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* puts instructions and data into the storage unit. A common 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 coded information punched in the cards to the storage unit. Other input devices that may be available include the keyboard, the tape drive, and punched paper tape. The choice of input unit is specified by an appropriate code number in the input instructions.

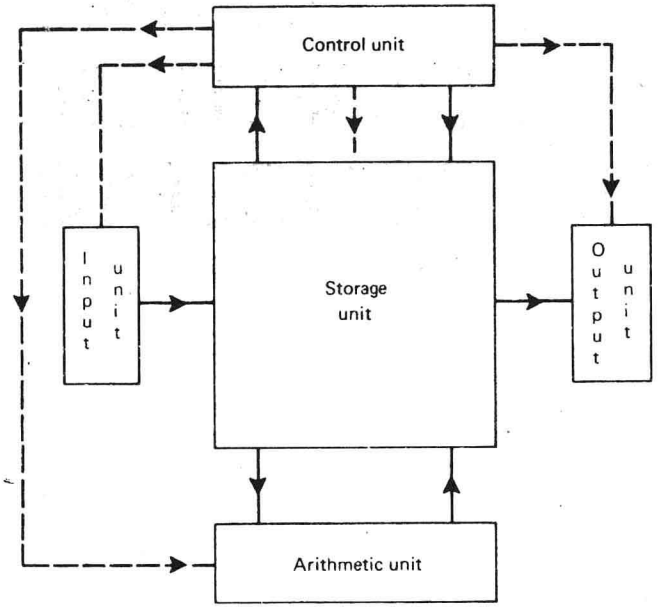


FIG. 1-1
Basic components, or functional units, of a digital computer. Solid lines represent flow of information (data); dashed lines represent flow of control signals.

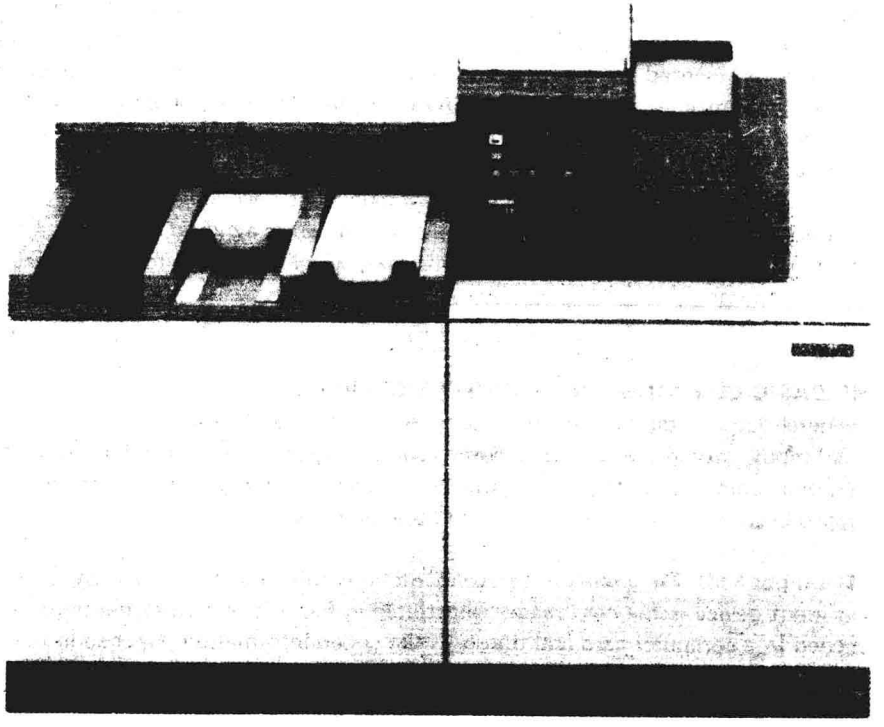


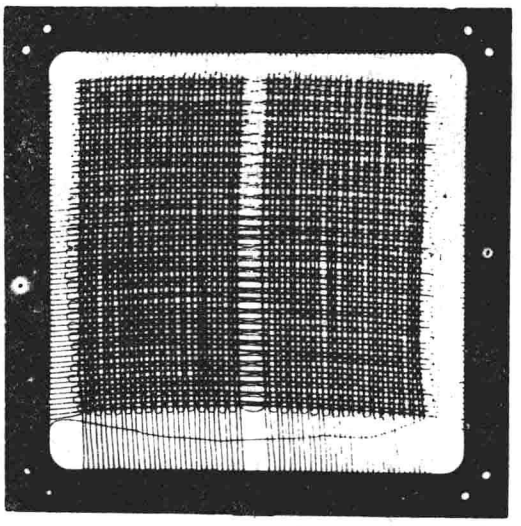
FIG. 1-2
IBM 35Ø5 card reader. (*IBM.*)

FIG. 1-3
IBM Ø29 card punch. (*IBM.*)

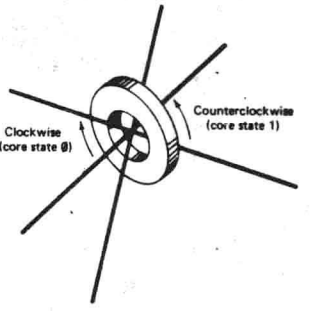
Before being placed in the card reader, blank computer cards must be punched on a machine called a *card punch* (Fig. 1-3). Depressing the character keys of the card punch causes one or more rectangular holes to be punched in the card. (The character punched may also be printed at the top edge of the card.) Each hole or column of holes represents a specific character to the card reader. Brief instructions covering the manual operation of the IBM Ø29 Card Punch are provided in Appendix A.

b The Storage Unit The storage unit of the early digital computers consisted of many core planes (Fig. 1-4), each made up of a number of ferrite cores (or rings) strung on hair-thin wires. Depending on how current was passed through those wires, each ring could be magnetized with either a clockwise or counterclockwise magnetic field. As computers increased in storage capacity, more space was required to house these core planes and size became a limiting factor. This problem of expanding size has been overcome by using ultraminiature electronic components; the storage unit of most present-generation computers consists of solid-state electronic circuitry instead of ferrite rings. But whether a ferrite ring is magnetized with a given polarity or an electronic gate is open or closed, the storage unit of any computer still functions as an integrated assembly of many two-state, or *binary*, devices. In an ON or OFF position, these binary devices may represent, respectively, YES or NO, PLUS or MINUS, ONE or ZERO, etc. A group of binary devices is referred to as a WORD,† each WORD having a unique retrievable *address*.

†In this text WORD stands for a binary word, as distinguished from an ordinary spoken or written (language) word.



(a)



(b)

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

Distinctive features of the storage unit of a general-purpose computer are *destructive read-in* and *nondestructive readout*. 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 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 can be added by using magnetic tapes or discs. Access to information stored on tape is *sequential*. This means that 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. Students who have advanced to the point where auxiliary storage capacity is required should consult the systems manual on file at the local computing center for additional information.

The storage unit is also known as the *memory unit*. The term "memory" is avoided 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 operations and the results thereof depend solely upon the instructions it receives from a human programmer. A common 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 is 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 can be obtained from the *printer* (Fig. 1-5). This machine prints the pages of results commonly associated with computer systems. Other output units punch cards, store information on magnetic or punched paper tape, use the typewriter on the computer console or remote terminal, or display results on a cathode-ray tube (CRT). The choice of output unit is specified by an appropriate code number in the output instructions.

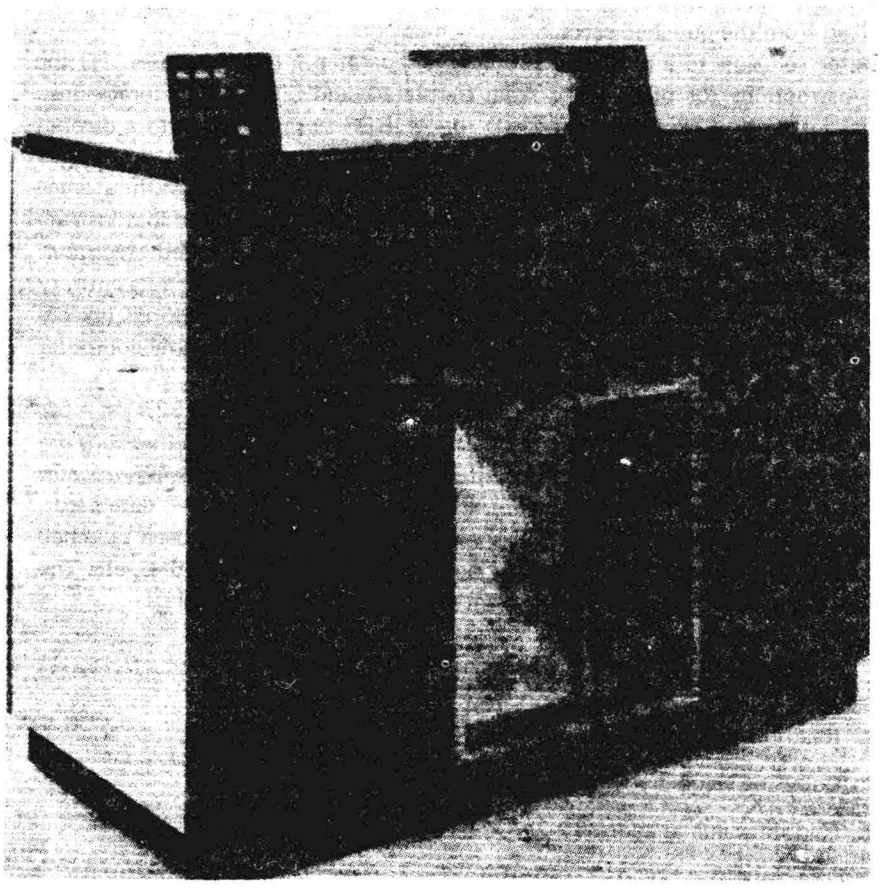


FIG. 1-5
IBM 3211 line printer. (*IBM.*)

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 state precisely 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*. For card input, 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, called *data cards*, 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 directly interpret the user-oriented language. A prestored intermediate program translates the user-oriented language (FORTRAN) into the machine-oriented language of the computer. This prestored translator program is called a *compiler*. Most compilers also include diagnostic routines which print ERROR messages identifying predictable programming errors. This capability will be discussed in more detail in Chap. 7. Another usual output of the compiler is a list of the statements read 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: (1) 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*. (2) 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 specified operation) 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 (execution), 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 described in Sec. 1-2. The storage unit provides for 3Ø 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 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 sequence, the card deck is placed in the card reader (as indicated in Fig. 1-6), the START button is pressed, and action begins.

The condition of the computer after all instructions have been read in (compilation is

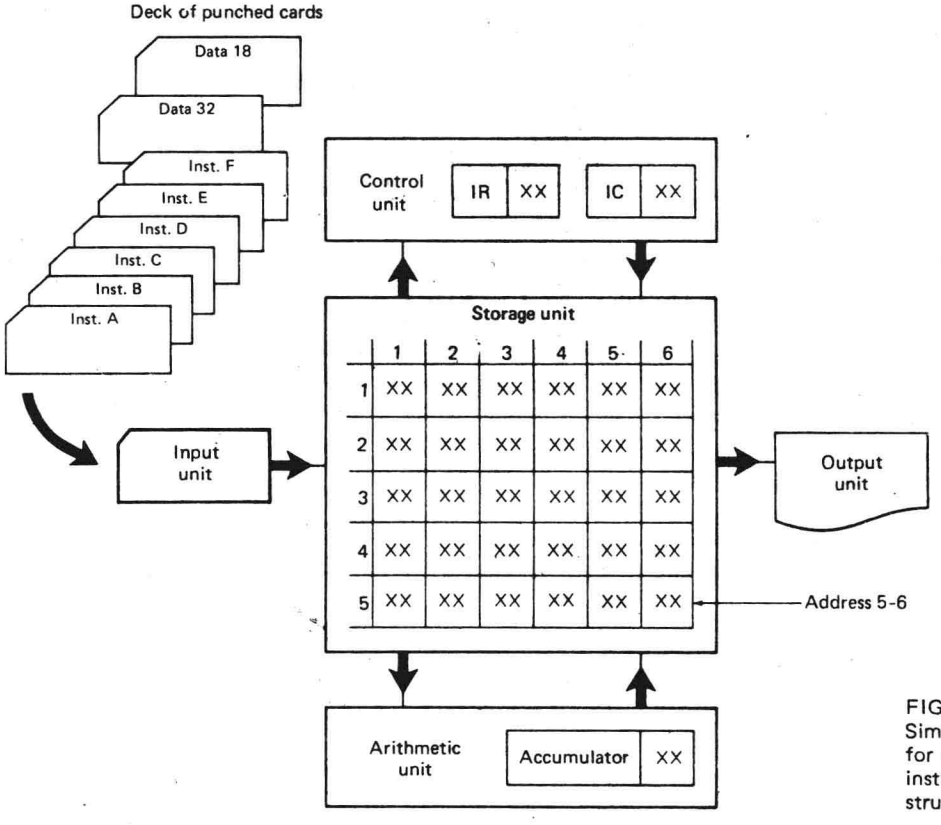


FIG. 1-6
Simulation of computer ready for input of instructions (IR = instruction register; IC = instruction counter).

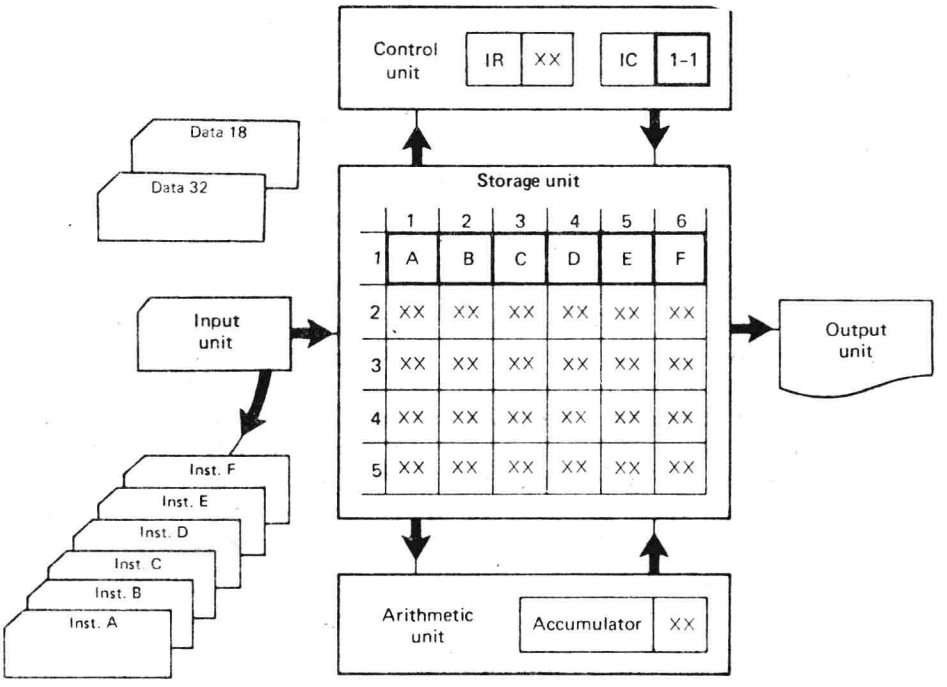


FIG. 1-7
Simulation of computer ready for execution.

complete) is shown in Fig. 1-7. 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. 1-8 the first instruction (instruction A in address 1-1) has been "copied" into the instruction register, 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 on it (32) is stored in address 2-1. The IC automatically increments to the next address in sequence (1-2).

Figure 1-9 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:

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

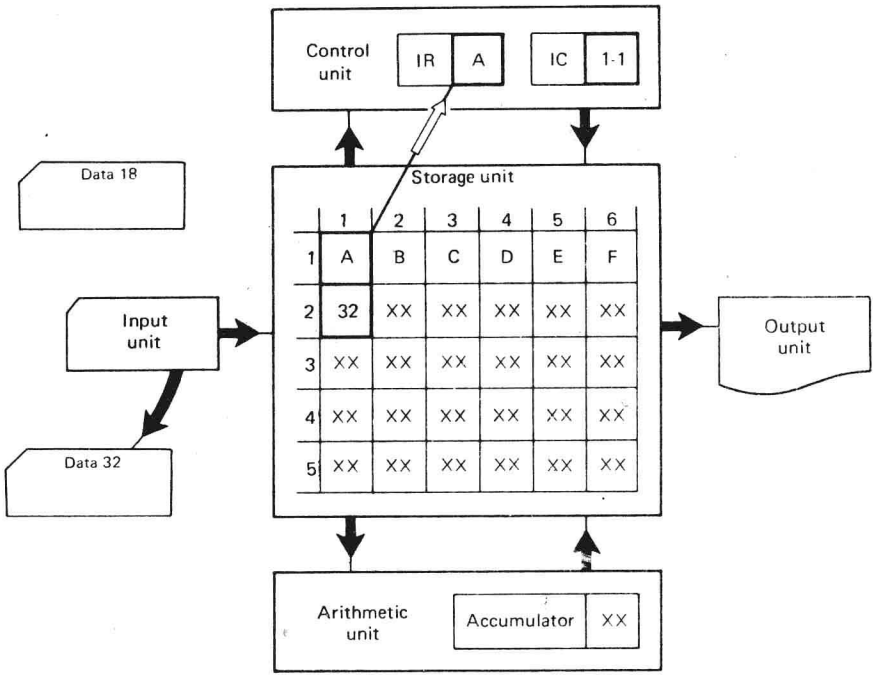


FIG. 1-8
Simulation of computer after execution of instruction A: READ A DATA CARD AND STORE ITS VALUE IN ADDRESS 2-1.

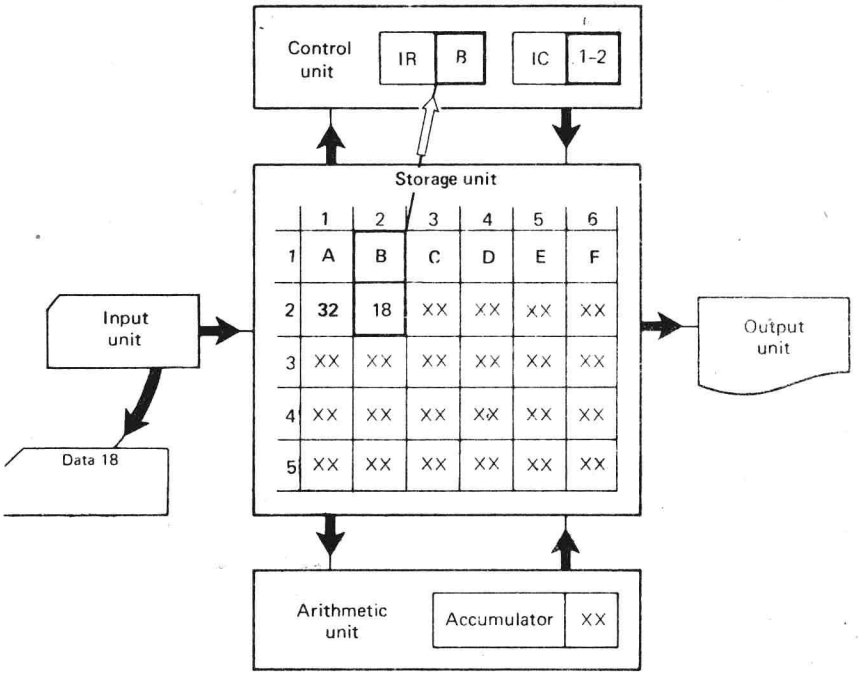


FIG. 1-9
Simulation of computer after execution of instruction B: READ A DATA CARD AND STORE ITS VALUE IN ADDRESS 2-2.

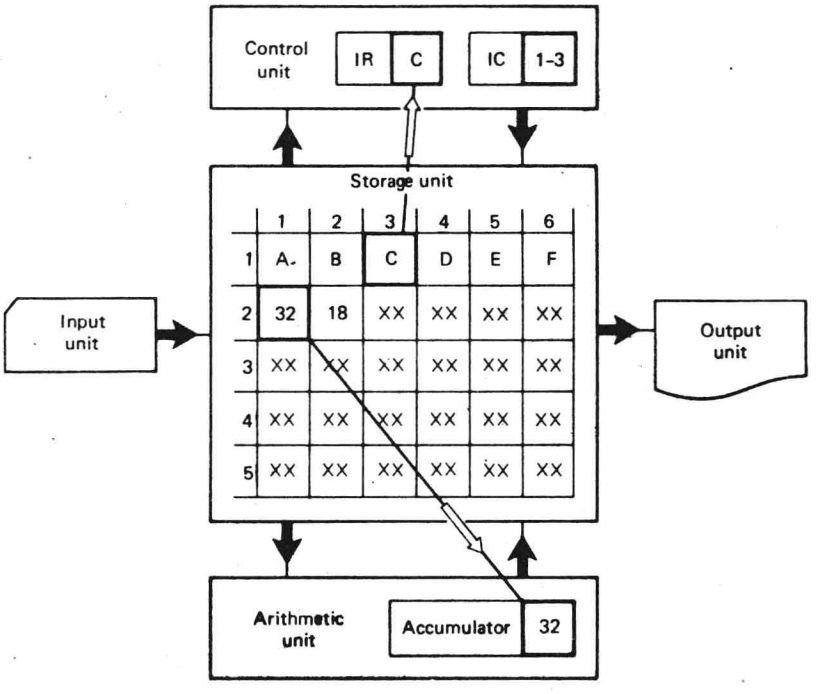


FIG. 1-10
Simulation of computer after execution of instruction C: COPY THE VALUE IN ADDRESS 2-1 INTO THE ACCUMULATOR.

The next data card in sequence is read, and the numerical value punched on it (18) is stored in address 2-2. The IC automatically increments to the next address in sequence (1-3).

In Fig. 1-10 the instruction in address 1-3 (instruction C) has been copied into the IR. Instruction C is executed:

Copy the value in address 2-1 into the accumulator.

The value in address 2-1 (32) is copied into the accumulator, where arithmetic operations are performed. The previous value in the accumulator has been erased. The IC now increments to 1-4.

The instruction in address 1-4 (instruction D) is copied into the IR (Fig. 1-11). Instruction D is executed:

Add the value in address 2-2 to the value in the accumulator.

The value in address 2-2 (18) is added to the value in the accumulator (32). The sum (5Ø) replaces the previous value (32) in the accumulator. The IC increments to 1-5.

The instruction in address 1-5 (instruction E) is copied into the IR (Fig. 1-12). Instruction E is executed:

Store the value in the accumulator in address 2-3.

The value in the accumulator (5Ø) is copied into address 2-3. The IC increments to 1-6.

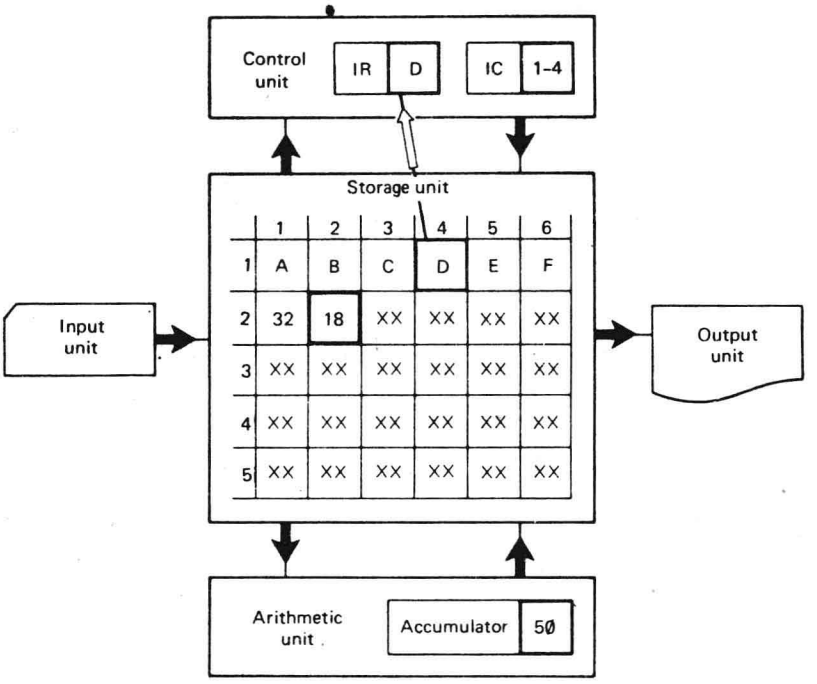


FIG. 1-11
Simulation of computer after execution of instruction D: ADD THE VALUE IN ADDRESS 2-2 TO THE VALUE IN THE ACCUMULATOR.

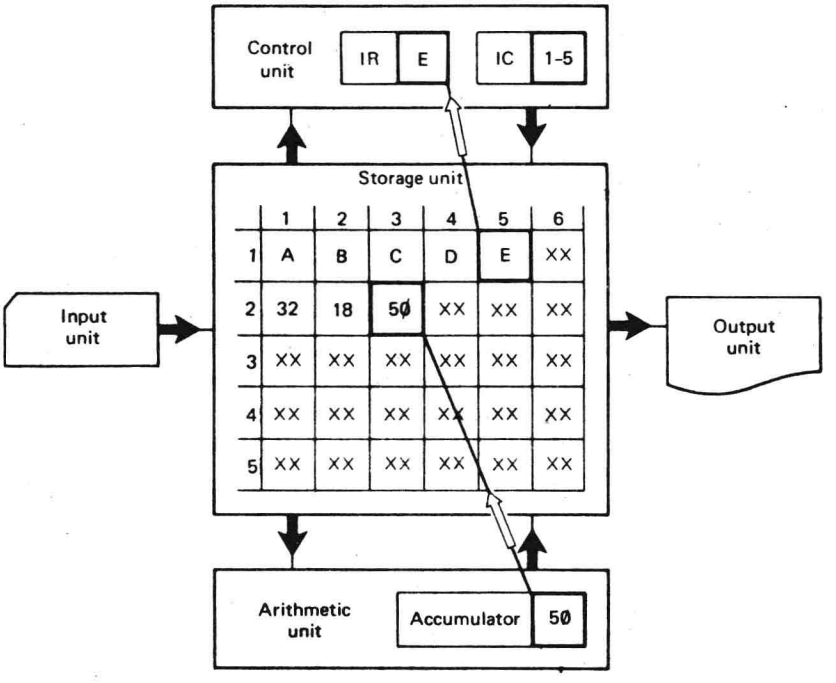


FIG. 1-12
Simulation of computer after execution of instruction E: STORE THE VALUE IN THE ACCUMULATOR IN ADDRESS 2-3.