FORTRAN IV FOR BUSINESS AND GENERAL APPLICATIONS

FORTRAN IV FOR BUSINESS AND GENERAL APPLICATIONS

HARICE L. SEEDS
Los Angeles City College

JOHN WILEY & SONS, INC.
NEW YORK
LONDON
SYDNEY
TORONTO

This book was set in Times Roman by Graphic Arts Composition, Inc. It was printed and bound by Hamilton Printing Company. The text and cover designer was Jerome B. Wilke. The drawings were designed and executed by John Balbalis with the assistance of the Wiley Illustration Department. Photographs courtesy of Alice Belkin, Burroughs, Computer Machinery Corporation, International Business Machines, National Cash Register, Trendata, and Xerox Corporation. Regina R. Malone supervised production.

Library of Congress Cataloging in Publication Data:

Seeds, Harice I.

FORTRAN IV for business and general applications.

Includes index.

1. FORTRAN (Computer program language) 2. Electronic data processing—Business.

I. Title.

HF5548.5.F2S344 001.6'424 74-34058

ISBN 0-471-77109-0

Printed in the United States of America

10-9 8 7 6 5 4 3 2 1

Preface

The FORTRAN language is a tool for solving problems. Its versatility, especially as developed in the version known as FORTRAN IV, is limited only by the imagination of its users. Many computer languages have been developed in recent years for special purposes, but FORTRAN IV remains one that is applicable to a wide variety of purposes. It is also the most common computer language and is found on almost all digital computers.

Manufacturers, software companies, and other businesses are becoming more and more aware of FORTRAN's usefulness. They see advantages in the comparatively short initial learning period required for programmers. They also see that less time is needed for developing individual programs because less coding and fewer commands are required than in the traditional business languages.

This book deals mainly with business problems, but students in the social sciences, humanities, and other nonmathematical disciplines will find it useful and easy to understand. Laymen's terms explain concepts that are too often hidden in technical jargon. Technical terms are introduced and explained as they are used so that students will become familiar with them. Each new concept is presented as simply as possible, both as a separate development and in relation to its use with other FORTRAN capabilities.

This is accomplished partly by presenting a programming problem that begins as a card-to-printer listing and develops through several additions into various billing programs for a business firm. The programs incorporate most of the features of business data processing and of FORTRAN IV. Although other applications are also developed, the concentration on one basic program enables the student to see the new, increased capabilities most advantageously. It also enables him to compare and contrast FORTRAN IV techniques by seeing different solutions to the same basic problem.

Another unique feature of program development is the consistent use of card layout forms, printer spacing charts, and flowcharts. These programmer aids show the student the various considerations involved in commercial programming and the necessity for complete documentation.

Several programming assignments follow each programming chapter. All assignments demonstrate the instruction being studied as it is encountered in actual programming, and they were carefully developed to require only the FORTRAN IV capabilities already studied. Although

the assignments are realistic, they are purposely simple so that the student can develop and execute several programs during the course.

A list of chapter objectives introduces each chapter. Review questions are interspersed throughout the chapters for study checkpoints. At the end of each chapter, just before the programming assignments, students are asked to review the chapter objectives. Thus the objectives not only introduce the FORTRAN capability but also are a chapter summary and an aid for review.

When the FORTRAN IV language is studied concurrently with, or follows, a general computer science or business data processing course, Chapters 1 and 2, which are introductory, can be skipped and programming can be introduced immediately. When the instructor introduces the FORTRAN language, he can start with either Chapter 3 or Chapter 5. Chapter 3 explains and develops a short, complete FORTRAN IV program to input and output data. Chapter 4 extends this program so that unlimited amounts of data can be read and printed. Chapter 5, on the other hand, concentrates on literals in printed output.

The study of Chapter 5 before Chapters 3 and 4 enables the student to write programs at the first class meeting. He becomes familiar with computer output before encountering the intricacies of field specifications. Thus he has high enthusiasm from seeing the results of his programming efforts so quickly, and this interest extends into the study of the more complicated concepts of Chapter 3. The student also learns many aspects of formatting while at the height of his enthusiasm. This is a distinct advantage because report form is important in business data processing. The book's flexibility allows the instructor to choose the approaches that he prefers.

The arrangement of FORTRAN IV instructions and the importance placed on actual programming exercises give students a usable tool no matter how far the course extends. For business survey courses or for general education classes, this is particularly valuable. By the time the student has completed Chapter 11, he can program any FORTRAN IV problem, although the program may be longer than it will be when all of the language capabilities are learned. He also knows what tasks computers can perform. This is an educational requirement not only for businessmen but for all laymen in the 1970s.

Harlce L. Seeds

Contents

1

Introduction to Programming

FORTRAN is a language that requires very little knowledge of the inner manipulations of a computer. The FORTRAN language itself takes care of most of the details a computer requires. Some computer knowledge helps, however, in understanding terms and procedures, and most people who are interested in programming want to know how the computer accomplishes its amazing feats. Therefore this introductory chapter presents an overview of (1) the computer, (2) data as stored and used by computers, and (3) the development of computers and computer programming languages. Each of these topics is presented briefly to introduce the reader to basic concepts of computer development and to initiate him to computer-oriented vocabulary terms. Each computer concept has many aspects. Books ranging from easily understood laymen's language to highly technical terminology are available if the reader wishes to explore these aspects more fully.

Computer Hardware

In computer terminology, the physical equipment or devices of a computer are called *hardware*. Computer programs of instructions are called *software*. A quick survey of computer hardware and software provides background for a study of the FORTRAN language.

FORTRAN IV is the most modern version of the FORTRAN language. Programs in FORTRAN IV can be processed by virtually all modern computers, such as IBM S/360, S/370, 1130, and System 3 computers, Xerox, Burroughs, CDC, and Univac computers. These are

virtually all third- and fourth-generation computers that can perform intricate manipulations unknown to earlier computers.

The word computer gives the impression of one machine. Actually, *computer system* is a more accurate term because computers consist of several devices. Each device is a machine consisting of its own electronic circuitry but connected to the other computer equipment.

Computer systems vary in size according to manufacturer, series, model, and amount of storage. Their physical shapes differ, depending upon which equipment is utilized. Some devices may not even be physically in the same area as the rest of the system. They may be a few feet away (in the next office or upstairs, for example) or they may be hundreds of miles away from the central system.

REVIEW QUESTIONS

1. In computer terminology, what is computer equipment called?
2. Why is the term "computer system" more accurate than the word computer?
3. What factors determine the physical appearance of a computer?

Central Processing Unit

Because physical appearance can differ so widely, Figure 1 demonstrates the way computers function regardless of specific devices.

Of all parts of a computer, the *Central Processing Unit* (CPU) performs the greatest number and the greatest variety of tasks. It is one

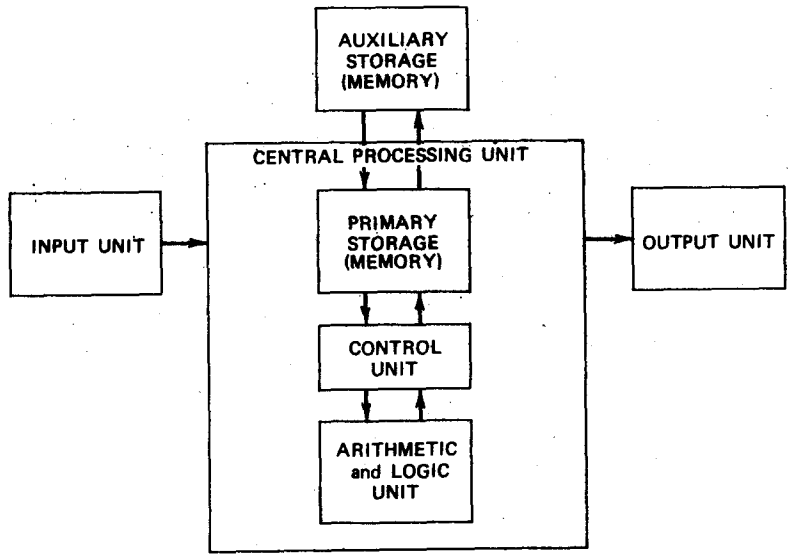

Figure 1 Schematic diagram of computer systems.

piece of equipment, but it contains the Arithmetic-Logic Unit, the Control Unit, and primary storage. The "face" or front of the CPU is a visual display unit called the *console*. By its lights, the console gives information to the operator about operations being performed by the computer. Many of the dials, switches, and keys on the console can be used by the operator to give operational instructions to the CPU itself. Thus, in a special way, the Central Processing Unit is also an input/output device.

The most essential parts of the CPU, however, are within it. The *arithmetic-logic* section performs the basic arithmetic processes of addition and subtraction, multiplication and division. (Technically, multiplication and division are variations of addition and subtraction.) No matter how complicated mathematical calculations are, computers break them down to these arithmetic processes. The Arithmetic-Logic Unit of the CPU also stores (briefly), shifts, and transfers data. It compares data and makes decisions that depend upon whether the two compared items are equal or unequal. This last process of comparing and branching is the essence of "logical" operations.

The *control section* of the CPU is the computer's "traffic manager." The Control Unit coordinates the interaction between units of the computer. It directs the order by which operations will be performed. It also determines when input/output devices will transmit and when data will be transferred to and from storage.

Storage in the CPU (and on disk in fourth-generation virtual storage computers) is often termed "main" or "primary" memory. Until the late 1960s, primary storage usually consisted of magnetic cores. These are tiny iron rings (Figure 2). They can be magnetized in millionths of a second, and they keep the magnetism until it is deliberately changed. The cores are either "on" (magnetized clockwise usually) or "off" (magnetized counterclockwise usually).

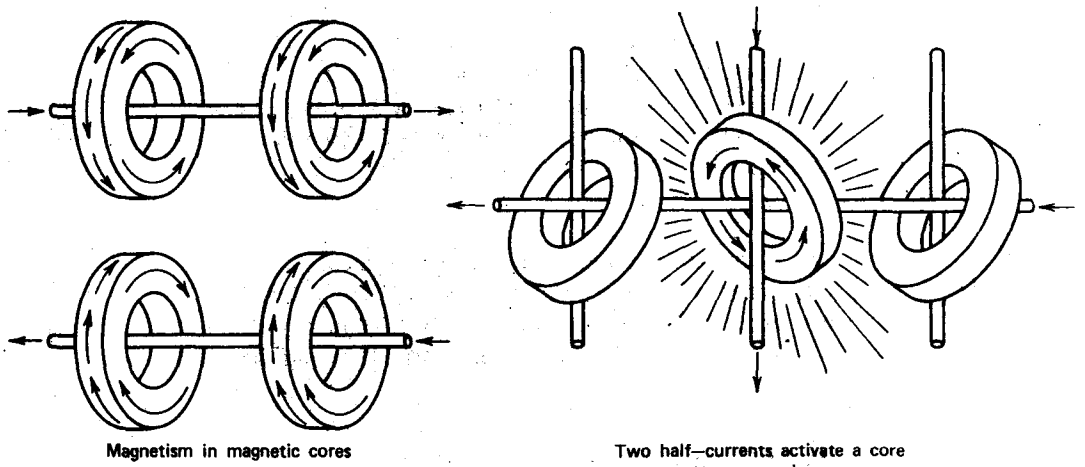

Figure 2 Binary states of magnetic core.

These states of magnetic core are binary because only two conditions are possible: clockwise or counterclockwise, "on" or "off," 1 or 0. Thus the clockwise/counterclockwise or on/off state of a core in memory is represented numerically by a binary digit, either 0 or 1. In discussing computer storage, 0 represents the off condition, and 1 represents the on condition. Such discussions frequently refer to binary digits as "bits."

Today most primary storage is composed of miniaturized circuitry. Although more difficult to visualize than core, its components also utilize the principle of binary conditions and are represented by binary digits.

REVIEW QUESTIONS

4. Which part of a computer performs the greatest number and greatest variety of tasks?
5. What is the "face" or front of the CPU called?
6. What does the control unit coordinate?
7. Which part of the CPU performs addition, subtraction, multiplication and division?

Auxiliary Storage Units

Additional storage units increase the memory capacity of a computer. They usually are called *auxiliary* or *secondary storage*. Storage that is separate from the CPU takes many forms, but each uses the binary principle. That is, a spot either is or is not magnetized; a punch either is or is not present. Punched paper tape, magnetic tape, magnetic disk, magnetic drum, and magnetic strips in data cells are all forms of auxiliary storage. So are punched data cards. Each form needs its own special "reader" or drive. However, an installation can use different devices in any combination that is deemed best for its own jobs.

Before information in auxiliary storage can be utilized, however, it must be transferred into the primary storage area of the Central Processing Unit. Only in the CPU can any processing except input/output occur. Figure 3 shows the devices and media that are most frequently used for auxiliary storage.

REVIEW QUESTIONS

8. Name three types of auxiliary storage.
9. What is the meaning of a "binary" condition?
10. How is all computer storage a form of binary states or conditions?
11. Where must data be stored before they can be processed?

Input/Output Devices

Input/output devices all have the same purpose: to get information into or out of the Central Processing Unit. Computer programs fre-

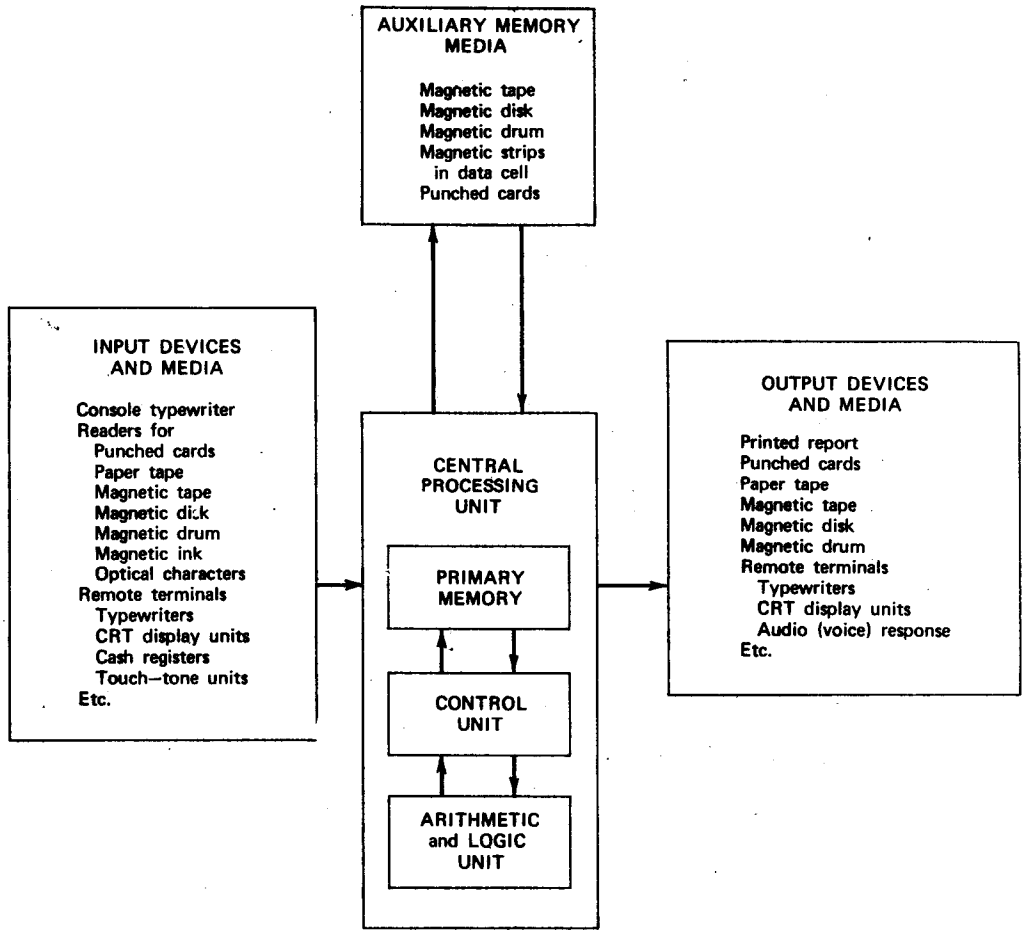

Figure 3 Devices and media in computer systems.

quently print reports, but computer output is increasingly intended for later use as input for another computer program. The most common medium that can be both input and output is still punched cards. The uses of magnetic tape and magnetic disks are growing, however, because they require little handling and are fast in operation. Storage on magnetic tapes, magnetic disks and drums, and in data cells also takes far less space than card storage does.

The most dramatic advances in recent computer development involve such input/output devices as voice recognition units, optical character scanners for handwritten or printed characters, microfilm printers, and cathode ray tube (CRT) display units that look like small television screens. No matter what medium is used, however, every input device must perform the task of transmitting data that the CPU can utilize finally in machine-understandable (binary) form, and every out-

put device must transmit from binary form into human-understandable or machine-understandable form.

Many input/output media, such as punched cards or magnetic tape, are auxiliary storage until the data records are required for input.

Characters, Fields, Records, and Files

The basic element of computer input or output is called a *record*. It is all the information related to one subject. That subject can be an inventory item, a customer, or an employee, for example. A record can be a punched card report of a purchased item, a line of print in a payroll report, or data input or output on a magnetic tape or disk. Minimum and maximum record lengths depend on the input or output medium (card, tape, disk, etc.) transmitting the data.

A record is made up of one or more pieces of information called data fields or simply *fields*. A punched card record, for instance, could contain the identifying number, price, and quantity of an item for sale. Each of these three units of information related to the sale item is a numeric field. *Numeric fields* contain only numeric characters. That is, they can contain only numbers, plus or minus signs, and decimal points. Dollar signs and commas cannot be part of a FORTRAN numeric field. If a sign is not part of a numeric field, the field is assumed to be positive.

If the name of an item is present, that field is *alphameric*. This means that its characters can be alphabetic, numeric or special characters. FORTRAN alphabetic characters include not only all the letters of the English alphabet, but also the dollar sign ($). *Special characters* include everything else in the computer printer set of characters, such as punctuation marks, symbols like the percentage sign (%) and ampersand (&), and the blank (usually signified in writing as ␣). Alphameric fields have many special capabilities as well as special rules in FORTRAN. These are discussed and demonstrated at considerable length throughout the text.

Fields are always part of a record. A collection of records is called a *file*. Time cards for all employees form a time card file. A magnetic tape or disk with all customers' names and addresses is a file. A printed report is an output file to the computer. Ordinarily, a business program processes one complete file of records.

REVIEW QUESTIONS

12. What are the basic purposes of input/output devices?
13. Why are magnetic tapes and magnetic disks used increasingly?
14. What is a record?
15. What is a field?
16. What is the difference between a numeric and an alphameric field?
17. What is a file?

Bytes

A number, letter, or special character ordinarily takes one column in a punched card or one print position on a printed report. In IBM S/360 and S/370 and System 3 computers, and in other third- and fourth-generation computers, a character occupies one byte of computer storage. These computers are often called "8-bit byte" computers because each character is represented by 1 byte consisting of 8 bits or binary digits.[1] These bits are the numeric representations of magnetic core or other units of computer storage that have only 2 possible states: "off" (0) or "on" (1). Fortunately for the FORTRAN programmer he rarely needs to be aware of these internal storage details. He should know that the storage unit for a character is a byte since the term is often used in discussions.

Software

In order to utilize the capabilities of a computer, a person must analyze his problem to see how a computer can solve it. Then he must instruct the machine to do the required processing. At first, such instructions were wired into a computer, and only data were "read" by an input device so that it could be stored and processed.

Not long after electronic computers were first in operation, however, the great mathematician John von Neumann observed that computer memories could store not only data but programs of instructions as well. His "stored program concept" developed into series of instructions stored in computer memory to manipulate data also stored in the computer. Each group of instructions is called a computer program, and each program normally processes a data file.[2] Thus two separate kinds of input are entered into computer storage: (1) instructions to the computer and (2) data that the computer will process by directions given in the program of instructions.

Today these programs can be very complex. They can be written by the computer installation personnel or a programming service bureau, or they can be supplied by the computer manufacturer. The computer's own programs of operating instructions and procedures are usually called the *supervisor* or *nucleus*. These programs ordinarily are supplied by computer manufacturers or software specialists.

Although industrywide agreement has not been reached about

[1]Nine bits actually make up each byte, but the "parity" or self-checking bit is known only to the computer. So programmers refer only to the 8 bits with which they work.

[2]Such mathematical programs as creating random numbers or searching for prime numbers are usually performed without input data, however.

whether or not to include user application programs, the term software generally refers to professionally prepared stored programs of instructions.

Machine Language

Instructions in the early stored programs were given in numeric form. The programmer gave decimal number instructions that the computer easily translated into usable binary form. An instruction usually consisted of two or three numeric parts, representing an arithmetic or manipulating operation and one or two addresses in computer storage. In this "machine language," which was different for almost every kind of computer, the number 4 might represent the instruction to add one field to another field. The complete instruction could be:

```
4    105    207
```

This machine language instruction directs the computer to add (4) data beginning at position 105 of computer storage to data beginning in storage position 207.

The machine languages were a big step above wired instructions, but every tiny detail had to be programmed, the instructions were difficult to remember, and the language varied with each computer.

Assembler Languages

Computers respond quickly to numeric commands, but human beings have difficulty remembering the numeric symbols. This situation was eased when the programmers writing instructions for the computer substituted a letter for the numeric operation code in an instruction. For instance, instead of

```
4    105    207
```

the instruction could be

```
A    105    207
```

This was a big advance in programming because letters meaningful to humans, such as A for *add*, are more easily remembered than numbers. For the computer, the translation from A to 4 to the binary digits required only a little extra preliminary instructions.

Next came the realization that alphabetic characters can represent more than just operation codes. Instead of being limited to the actual numeric storage addresses, programmers can use names or labels as addresses in storage. Data fields can be referred to by such humanly

meaningful names as BONUS or PAY instead of being addressed by their numeric locations in storage. For example, the instruction

```
A    BONUS    PAY
```

adds data from the two storage locations labeled BONUS and PAY.

In order to convert the alphabetic operation codes and data field labels to their numeric addresses, the computer must already have stored instructions on how to perform the conversions. A large program, called an *assembler,* translates or converts the operation codes and labeled fields into the numeric commands required by the computer.

Assembly language instructions are in a 1-to-1 ratio with machine language commands. This means that one assembly language command translates into one machine language command. Assembly languages were an important step in the development of computer programming, but their detailed instructions and rigid requirements demand great time and care in both logic development and instruction writing.

REVIEW QUESTIONS

18. What is a byte?
19. What is computer software?
20. Why do programmers rarely write in machine languages?
21. What is meant by the statement that machine and assembly language commands have a 1-to-1 ratio?

Compiler Languages

After assembly (or symbolic programming) languages were established, programmers began to develop highly sophisticated sets of commands for converting instructions. These conversion programs usually are called *compilers*, although there are also generators, which have certain technical differences. Like assemblers, compilers convert or translate statements understood by humans into machine commands that a computer can execute. Compiler language instructions, however, are closer to the English language then to machine language. This is largely possible because compiler language instructions are not on a 1-to-1 ratio with machine language commands. One compiler language instruction generates several machine language commands, so that there is a 1-to-several ratio. This also means that the programmer need not worry about most of the internal computer movements since the compiler is assuming those details. Therefore compiler language programming is much easier and faster than assembler or machine language programming.

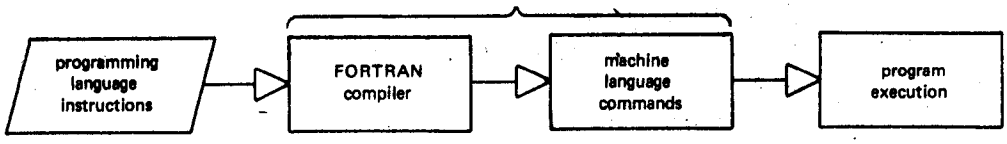

Figure 4

Because new tasks are constantly being found for computers, and because people differ about the best way to develop computer languages, many different compilers have been designed and are in use today. Some compiler languages are designed primarily for scientific use, others mainly for business reports. Some languages are developed for general computer usage; others are meant for limited, special computer tasks. Almost all are written so that they can be used on many different computers, regardless of the specific machine language of the computer. The most common, and therefore best known, compiler languages are FORTRAN, COBOL, ALGOL, PL/I, RPG (with a generator rather than a compiler), APL, and BASIC. They all have elaborate sets of instructions so that the programmer can easily utilize a computer's power and versatility.

Nonmachine languages, particularly compiler and generator languages, are often referred to as "source" languages. Source languages, except for assembly languages, frequently are divided into two categories: problem-oriented languages and procedure-oriented languages. In *procedure-oriented languages,* the programmer develops his program according to the types of tasks that the computer is to perform. For instance, the first instructions might describe all the forms of data to be fed to the computer by punched cards, paper tape, magnetic tape, or magnetic disk, during any part of the program. These are all covered by the general term "input." The next instructions detail all calculations that are to be performed on any of the data fields during any part of the program. Last might be specifications for all data printed, punched, or otherwise output by a device of the computer. All instructions are grouped by procedural divisions of input, calculations, and output. The compiler or generator translates these procedure-oriented instructions into machine commands that are to be executed in the computer's actual order of procedures. COBOL and RPG are both procedure-oriented languages.

In *problem-oriented languages,* such as FORTRAN, the programmer arranges instructions basically in the order by which they are to be executed. An instruction for reading occurs when the program needs input information. Calculations are described when the logic of the problem requires these calculations. Output instructions are given when information is to be printed, punched, or otherwise output. Ordinarily input, output, and calculation instructions are interspersed throughout the entire problem-oriented language program.

REVIEW QUESTIONS

22. What is a source language?
23. What are the advantages of compiler languages over machine or assembler languages?
24. What are the two basic categories of compiler or generator languages?
25. Name three compiler languages.