FORTRAN IV
FORTRAN IV
FORTRAN IV
FORTRAN IV

An Introduction To Fortran IV Programming A General Approach

Second Edition

Paul W. Murrill & Cecil L. Smith

FORTRAN IV
FORTRAN IV
FORTRAN IV
FORTRAN IV

An Introduction to FORTRAN IV Programming

A General Approach

Second Edition

PAUL W. MURRILL and CECIL L. SMITH
Louisiana State University

(includes appropriate implementation information and error codes for the WATFIV *compiler)*

INTEXT EDUCATIONAL PUBLISHERS
New York

To **WHIT**

Fourth Printing

Library of Congress Cataloging in Publication Data

Murrill, Paul W.
An introduction to FORTRAN IV programming.
"Includes appropriate implementation information and error codes for the WATFIV compiler."
Includes index.
1. FORTRAN (Computer program language) 2. Electronic digital computers—Programming. I. Smith, Cecil L., joint author. II. Title.
QA76.73.F25M88 1975 001.6'424 75-1284
ISBN 0-7002-2469-6

Intext Educational Publishers
666 Fifth Avenue
New York, New York 10019

Manufactured in the United States of America

An Introduction to FORTRAN IV Programming

Preface
to the Second Edition

The broad and significant acceptance of the First Edition has been, of course, a matter of deep satisfaction to all of those who had a role in its creation. Since its publication, however, there have been many changes both within the computing world and in the manner in which FORTRAN instruction is done. These, plus the experience with the First Edition, have convinced us that there is a need to publish this Second Edition, which we think has the following advantages over the previous edition:

1 Material describing modern developments in computer technology and usage are reflected in the text material. Since time-sharing systems are becoming more popular, discussion of their features is included, especially in Chapters 1 and 3. The instructor will need to supplement the text with material appropriate to any specific time-sharing terminal if one is used, however, since these systems do not have the degree of standardization that is present in conventional card-oriented systems.

2 This text strongly reflects the conviction that students should start to write short, simple programs at a very early stage of their learning experience.

The First Edition was oriented this way and this edition is even more slanted in this direction. Format-free input-output similar to that found in WATFIV is introduced so that the first programs can be written without the trauma of format. Concurrent presentation of format input-output is included, of course, for those systems that require its use. We like to allow students to write three or four programs without format and then to require it thereafter.

3 Material has been included on the WATFIV in-core compiler and its error messages are listed in an appendix. The text is, of course, not dependent on WATFIV; it is present only for student convenience.

4 The number of exercises—which was generous in the First Edition—has been increased and significantly revised.

5 Functions and subordinates were not treated in an especially effective manner in the First Edition; but we think in the current version the exercises themselves have been improved.

6 The treatment of input-output operations has an in-depth coverage in a later chapter (plus, of course, elementary presentations in early chapters), and discussion of handling character data is more useful.

7 Compilers change—one example is the manner in which they handle mixed-mode arithmetic. The First Edition implied that mixed-modes were taboo. Students often quickly learned, however, that the particular compiler they were using could accept mixed-modes, and they would then use such operations without understanding the potential hazards. This Second Edition handles this in a much more direct fashion.

8 The entire text offers a better treatment of such things as compilation, operating systems, computer characteristics, and FORTRAN itself.

We want to express our sincere thanks and appreciation to all those who helped us in this revision—especially to those who have sent us suggestions from colleges and universities throughout the United States. Such assistance, when coupled with the thoughtful advice and aid given to us by our own students and colleagues, make us hope this edition is a significant improvement over the First Edition.

Contents

1

INTRODUCTION TO DIGITAL COMPUTERS

The steam engine and other devices for doing work gave man an extension of his physical capabilities and brought about the Industrial Revolution. In a very similar manner, electronic computers are providing man with tools with which he can process quantities of information and solve problems that otherwise would be impossible to handle. These computers are producing an *informational revolution* that will have more impact on each of our everyday lives than any other aspect of modern technology—even atomic energy. The purpose of this book is to assist you, as a student, in learning to utilize these computers in your day-to-day work.

1-1. DIGITAL-COMPUTER CHARACTERISTICS

Modern electronic computers are of two basic types—digital and analog. The entire content of this book is directed toward understanding and programming digital computers, and no attention is devoted to the study of analog computers or combinations of analog and digital computers (hybrid computers).

Digital computers can be appreciated best by first considering some of their characteristics. Understanding these characteristics will help us to appreciate their usefulness.

One of the most prominent characteristics of digital computers is their truly incredible *speed.* Although they only work one step at a time, i.e., *sequentially,* they perform their tasks at rates that are beyond the comprehension of the novice. As an example, some large machines are capable of adding together several hundred thousand 16-digit numbers in less than a second. These tremendous speeds make it possible for the machine to do work in a few minutes that might otherwise require years of time.

Not only is the digital computer capable of working very rapidly, but it also has a perfect *memory.* It has virtually instantaneous "recall" of both data and instructions that are stored inside, and it never forgets or loses the accuracy of the information which it has within its memory.

A digital computer is an extremely *accurate* device. In most machines numbers are handled with seven, eight, or nine significant digits, and twice this accuracy can usually be obtained by the programmer. This means that a machine would have no difficulty multiplying 2782.4362 times 40.127896 and obtaining the product correct to 8 or 16 significant figures.

Coupled with the significant characteristics already listed, the digital computer does its work *automatically.* It can accept instructions from its operator, and then execute these instructions without need for human intervention. This implies that the machine can be given a problem; then while you attend a movie, it will do your work with incredible accuracy and at fantastic speeds. Learning to use such a tool should require no further motivation.

Additional characteristics of the digital computer will be noted later, but for the present it will be more advantageous to see how the machine works.

1-2. HOW THE DIGITAL COMPUTER WORKS

The digital computer is basically a device to accept *data* and a set of instructions as to how to manipulate these data in order to produce a set of answers. The set of instructions is called the *program,* and these are prepared by a *programmer* (you). (See Figure 1-1.) This book is primarily concerned with the preparation of programs. Sometimes

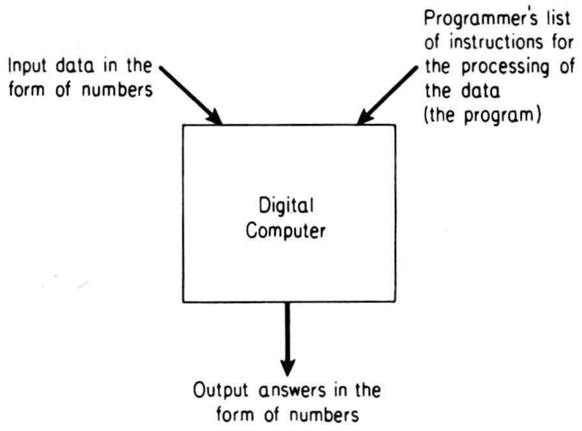

Input data in the form of numbers
Programmer's list of instructions for the processing of the data (the program)
Digital Computer
Output answers in the form of numbers

Figure 1-1. Functional role of the digital computer.

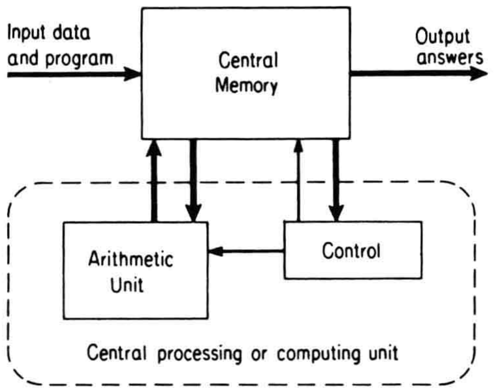


Figure 1-2. Relation of memory unit, arithmetic unit, and control unit (solid arrows represent information flow, dashed arrows represent control signals).

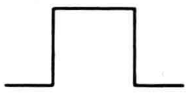
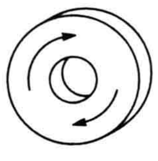
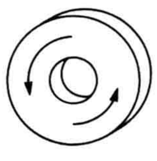
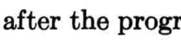
Device	"0" State	"1" State
Current pulse on a wire	No pulse	 Pulse
Magnetic field in a magnetic core	 Clockwise	 Counterclockwise
Switch	 Open	Closed

Figure 1-3. Examples of binary devices.

the data may be contained within the program; but more often the data are entered into the computer after the program.

In general, the computer may be thought of as being composed of three main sections: the memory, the central processing unit (CPU), and the input/output processor. The computer *memory* is used for storing data, instructions, intermediate results, and final answers; the *central processing unit* performs all the necessary manipulations of the data; and the *input/output processor* communicates with the outside world. (See Figure 1-2.) Transfer of instructions and data among these units takes very little time—in some machines less than one millionth of a second.

All information and signals in transit inside the computer are handled as electrical signals (usually pulses), and in memory this information is stored in magnetic cores, switches (flip-flops), and/or as magnetized spaces on drums, discs, and tapes. All of these devices are designed to exist in only one of two states which we may associate with the symbols 0 and 1. (See Figure 1-3.) These two states may be considered as *binary digits* or *bits* (a contraction of "*bi*nary dig*its*") and are used to represent information. Thus the number system employed is basically binary; but it is usually more convenient for instructions and addresses to be "represented" in the octal or hexadecimal number systems. Nonnumeric information (alphabetic and special symbols) in a computer is represented in a binary code, and numbers are represented in one of two ways: In a *binary-coded decimal system* (each digit coded in a fixed number of bits) or the decimal numbers are converted into the *binary number system* (used in most computers primarily designed for scientific work).

1-3. CONTROL AND OPERATION OF THE COMPUTER

While it is not necessary to understand such topics as binary arithmetic, the electronics of digital circuits, or other topics fundamental to the design of digital computers in order to learn to program in FORTRAN, a superficial understanding of the general operation of digital computers will easily reveal the origins of certain rules and conventions incorporated into the FORTRAN language. In reality, FORTRAN reflects basic machine characteristics to a greater extent than most other computer languages.

The general schematic diagram of the computing system in Figure 1-2 is shown in a little more detail in Figure 1-4. As pointed out in the last section, this system is broadly divided into three units: central processing unit, memory, and input/output processor.

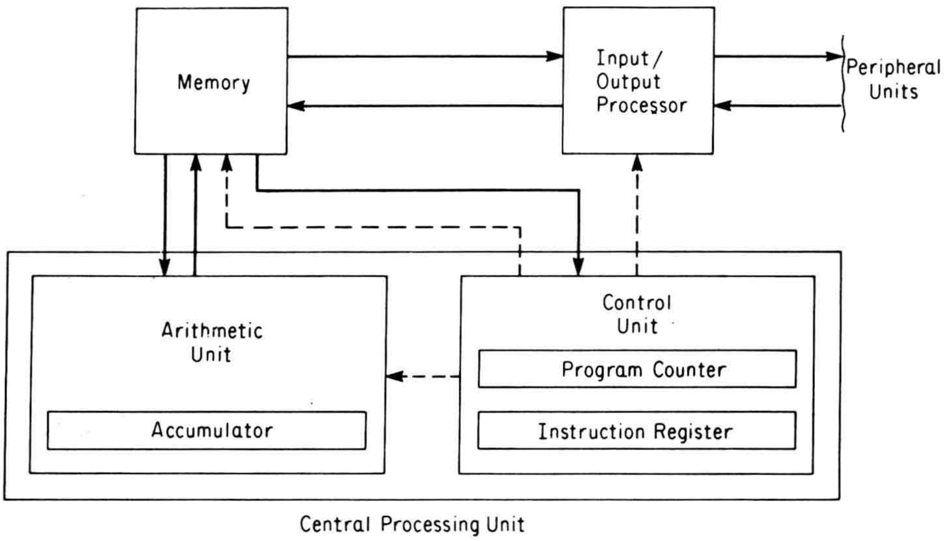


Figure 1-4. Schematic diagram of a simple computer (solid arrows represent information flow; dashed arrows represent control signals).

The central processing unit is further divided into two subunits: the arithmetic unit and the control unit. The *arithmetic unit* is responsible for performing operations such as additions, comparisons, etc., on the information in memory. The *control unit* is responsible for interpreting the instructions sequentially in memory and directing the arithmetic unit and input/output processor to perform the appropriate operations.

The concept of storing both the instructions (i.e., the program) and the data in the same memory unit has been of utmost importance in the development of computing machines. The very earliest computers employed hand-wired programs which made them inconvenient to use. The brilliant mathematician John von Neumann proposed the stored program concept which, coupled with the remarkable advances in electronics technology, led directly to computing machines as we know them today.

Since the central memory plays such an important role in the operation of the computer, a clear view of the organization of this memory is essential. Perhaps the most vivid way of visualizing the memory of the computer is as a set of mailboxes called memory cells, memory locations, or storage locations. This analogy is quite appropriate. In each individual memory cell only one word of information may be stored at any one time. This word of information may be either data (numerical or nonnumerical) or computer instructions. Each memory cell has its own individual address, and it is common to refer to memory cells by their addresses. The word contained in the memory cell appears as a binary number, and by superficial inspection there is no way to identify whether this word is data or whether it is an instruction. The computer must be told explicitly which memory cells contain instructions and which contain data. The control unit of the computer will treat the contents of a memory cell (a word) as though it were an instruction, and the arithmetic unit of the computer will treat the contents of a memory cell as though it were data. Each individual memory cell (or word) contains a fixed, preset number of digits, and that number of digits will limit the amount of significant information that can be stored in that memory cell. The instructions that are used by the computer for the processing of information are constructed so that they deal with memory-cell addresses.

As pointed out in the first section of this book, the digital computer is a sequential machine. Its operation is a sequence of cycles, each of which consists of two phases: a *fetch* phase and an *execute* phase. The fetch phase uses two registers in the control unit: the *program counter* and the *instruction register*. The program counter is often referred to by the more descriptive name of *instruction address register*, and it always contains the address of the next instruction to be executed. At the beginning of the fetch phase, the contents of the location in memory whose address is currently in the program counter is loaded into the instruction register. Therefore, the contents of this memory location will be treated as an instruction. At the completion of the fetch phase the program counter is incremented by one, so that it now points to the next instruction to be retrieved from memory.

At the start of the execution phase, the control unit decodes the instruction and issues specific commands to the various elements of the arithmetic unit. In performing its operations, the arithmetic unit utilizes a register called the accumulator to contain the data on which it is to operate. For example, a typical instruction might be to add the contents of a specific storage location in memory to the current contents of the accumulator. The instruction itself contains the address of the memory location involved and a group of bits (called the *operation code*) whose pattern indicates that addition is to be performed. The control unit relays the address to the memory addressing circuits in order to retrieve the contents of the storage location, and activates the "add" circuit in the arithmetic unit to achieve the desired result.

To illustrate the sequence of operations, suppose we examine the instructions required to add the contents of two storage locations (specifically, at addresses 2749 and 1398) and store the result in a third storage location (specifically, at address 1972). Three instructions are required:

Instruction	*Explanation*
LW, 2749	"Load Word" copies the contents of the storage location at address 2749 into the accumulator.
AW, 1398	"Add Word" adds the contents of the storage location at address 1398 to the current contents of the accumulator.
STW, 1972	"STore Word" copies the contents of the accumulator into the storage location at address 1972.

In the above explanations, note the use of the word "copies." The instruction LW, 2749 in no way alters the contents of the storage location at address 2749. Similarly, the instruction STW, 1972 does not alter the contents of the accumulator, but does obliterate whatever was previously contained in the storage location at address 1972. Although not specifically mentioned, the instruction AW, 1398 does not alter the contents of the storage location at address 1398. These points can be summarized by the following rule: Read operations on memory are nondestructive; write operations on memory are destructive. FORTRAN follows this rule exactly.

To further illustrate the sequence of operations, suppose the three instructions are stored in the memory locations at addresses 1027, 1028, and 1029. If the program counter initially contains 1027, the sequence of operations is as follows:

1. The contents of the storage location at address 1027 are copied into the instruction register.
2. The program counter is incremented by 1, giving 1028.
3. The contents of the storage location at address 2749 are copied into the accumulator.
4. The contents of the storage location at address 1028 are copied into the instruction register.
5. The program counter is incremented by 1, giving 1029.
6. The contents of the storage location at address 1398 are added to the contents of the accumulator.
7. The contents of the storage location at address 1029 are copied into the instruction register.
8. The program counter is incremented by 1, giving 1030.
9. The contents of the accumulator are copied into the storage location at address 1972.

Many current computers could perform all these operations in less than ten millionths of a second.

Of course, current computers offer far more features than illustrated by the previous example, but their operations are basically straightforward. The examination of these other features is inappropriate for an introductory manual on FORTRAN.

1-4. PROGRAMMING LANGUAGES

In the previous section we discussed how the computer would execute a program: In this section we want to examine the preparation of a program.

Writing a program directly in instructions, as described in the previous section, is said to be programming either in assembly language or in machine language. While this approach is relatively straightforward, it becomes tedious, especially for large programs. In essence the available instructions comprise the computer's language, which we could learn to speak, but would rather not. Of course, the best solution would be for the computer to speak our native language, which for most readers of this book would be English. Unfortunately, this goal has not yet been achieved, although progress is being made.

The current solution is to use an intermediate language that has some of the characteristics in which problems are naturally expressed, but a language that is sufficiently rigorous to permit the computer to perform the translation from the program written in the programming language into the instructions that comprise the computer's natural language. This situation is illustrated in Figure 1-5. The programmer must translate the statement of the problem into statements in the programming language. Using a program known as a *compiler*, the computer translates the statements in the programming language into machine-executable instructions, a process referred to as *compilation*.

Since the statement of problems tends to differ from discipline to discipline, several different programming languages have appeared, each with special characteristics that make one language more attractive to some fields and disciplines than to others. In the business field, COBOL (COmmon Business Oriented Language) has dominated, primarily because its features enable large files of data to be manipulated readily. In science areas, FORTRAN (FORmula TRANslator) has dominated, primarily because algebraic expressions can be readily implemented. However, FORTRAN has enjoyed some use in business circles. The last decade saw the introduction of several new languages, some

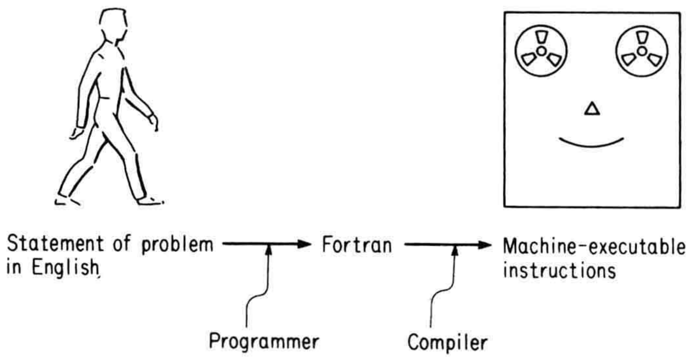

Statement of problem in English
Fortran
Machine-executable instructions
Programmer
Compiler

Figure 1-5. Role of FORTRAN.