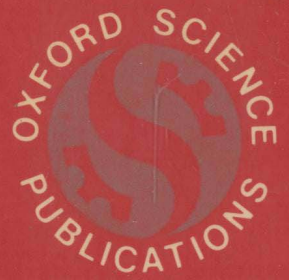


EFFECTIVE FORTRAN 77

Michael Metcalf

Effective FORTRAN 77

Michael Metcalf
CERN, Geneva, Switzerland

CLARENDON PRESS · OXFORD
1985

Oxford University Press, Walton Street, Oxford OX2 6DP
London New York Toronto
Delhi Bombay Calcutta Madras Karachi
Kuala Lumpur Singapore Hong Kong Tokyo
Nairobi Dar es Salaam Cape Town
Melbourne Auckland
and associated companies in
Beirut Berlin Ibadan Mexico City Nicosia

OXFORD is a trade mark of Oxford University Press

Published in the United States
by Oxford University Press, New York

British Library Cataloguing in Publication Data

Metcalf, Michael
Effective FORTRAN 77.
1. FORTRAN (Computer program language)
2. Microcomputer–Programming
I. Title
001.64'24 QA76.73.F25
ISBN 0-19-853709-3

Library of Congress Cataloguing in Publication Data

Metcalf, Michael.
Effective FORTRAN 77.

Bibliography: p.
Includes index.
1. FORTRAN (Computer program language) I. Title.
QA76.73.F25M478 1985 001.64'24 84-28522
ISBN 0-19-853709-3

Printed in Great Britain by
St Edmundsbury Press Ltd
Bury St Edmunds, Suffolk

Effective FORTRAN 77

PREFACE

FORTRAN 77 is now established as the dominant version of this programming language, relegating FORTRAN 66 to the position of a computing relic. The introduction of the new standard in the late 1970s led to the publication of a spate of excellent text books, but these are usually intended for beginners – those who have no experience, either in FORTRAN or in programming as such. This means that more experienced FORTRAN programmers are often left to their own devices, as there are very few texts which enable them to revise and develop their skills. At the same time, there are very many programmers who begin their training in BASIC or in a teaching language such as PASCAL, and are confronted with FORTRAN only later when they embark on their careers in research or industry.

The purpose of this book is to provide a complete but concise review of the FORTRAN 77 programming language, to serve both as a reference for the intermediate or advanced programmer, and as a means whereby programmers in other languages can rapidly acquire a knowledge of FORTRAN 77, without becoming bogged down in expanses of explanatory text more suitable for absolute beginners. The book should similarly be useful to FORTRAN 66 programmers wishing to convert to the new standard, and to those beginners who may prefer a faster approach to the topic. However, the book goes beyond a bare exposition of the language, presenting other material which helps in an understanding of its history and development, as well as giving detailed advice on how FORTRAN should be used wisely within the boundaries of its inherent limitations. Since FORTRAN, the first high-level programming language, is a far from perfect tool, it is even more important that it be used carefully, lacking as it does much of the built-in protection contained in more recent languages.

Chapter 1 gives a brief introduction to computers and a short review of FORTRAN's relatively long and successful history, as well as indicating likely developments in the future. The following six chapters are devoted to a complete description of the language. They present the version defined by the 1978 ANSI standard, the reader being taken step-by-step through each of its features, which are explained in sufficient but not excessive detail.

At a time when the interchange of software is becoming ever more important and computer systems ever more varied, it is essential to ensure that programs written and tested on one system will run with little or no change on another. Chapter 8 provides a list of recommendations which should help in the production of portable code. Writing portable code from the outset can often save costly re-programming at a later stage, and is a technique which needs to be absorbed when learning the language itself.

The topics of style and good programming practice are further developed in Chapter 9, and the question of program design is tackled in Chapter 10, which shows how the principles of good program design can be applied in the context of FORTRAN programming.

One of FORTRAN's prime advantages has always been its emphasis on efficient object program execution, and some advice on how this might be achieved is given in Chapter 11. As modern computers become more powerful, the problems they help to solve become more complex, and it remains vital to ensure that programs remain efficient, even though computer hardware is ever cheaper.

The final chapter, 12, deals briefly with the topics of program testing and documentation, two areas neglected in many texts, but ones which are important in program development and maintenance.

FORTRAN has had a long and successful history and, in spite of the development of new languages, it still remains the most widely used in large-scale scientific computation. This book does not set out to demonstrate that it is a perfect tool for that purpose, but rather tries to ensure that it is used to the best possible effect in those areas in which it still has no obvious contender.

ACKNOWLEDGEMENTS

A book such as this contains a great deal of detailed information which requires extensive and careful checking. I have been greatly helped in this task by several friends and colleagues, and I am particularly grateful to J. M. Gerard and R. Matthews of CERN and to J. D. Wilson of Leicester University for their critical comments on drafts of the text. The responsibility for any remaining errors or omissions is entirely my own.

Much of the advice in the latter half of the book has been collected from diverse sources, and I acknowledge particularly the work of Mme. F. Vapne of the Electricite de France, which provided the basis of Chapters 8 and 9. I thank also the University of Leicester for permission to reproduce Appendix B.

It is a pleasure to thank the CERN management, and especially P. Zanella, for encouraging me to undertake this work, and for providing the necessary resources for its realisation.

My final thanks go to A. Berglund and D. Stungo for their helpful cooperation in the preparation of the final camera-ready copy.

CONTENTS

1 INTRODUCING FORTRAN

This book is concerned with the FORTRAN programming language. It sets out not only to offer a complete and relatively concise description of the whole language, but seeks also to emphasise those language features which are considered to be consistent with good programming practice. Features which are less desirable are given less prominence. The language description occupies Chapters 2 to 7, which are written in such a way that simple programs can already be coded after the first three of these chapters have been read. Successively more complex programs can be written as the information in each subsequent chapter is absorbed. Chapter 7 covers the whole of the input/output features in a manner which confines the more advanced features to its end, so that the reader can approach this more difficult area feature by feature, but always with a useful subset behind him.

The remainder of the book is concerned with the effective use of the language. It sets out to provide guidance which goes beyond the coding of syntactically correct programs, describing how programs can also be made portable, neat, efficient, well documented and tested. Although these chapters may often read as long lists of do's and don'ts, they provide the material necessary to progress from a mere formal command of the language to the ability to wield it as a well mastered tool.

This present chapter has the task of setting the scene for those which follow. For those who are unfamiliar with any notions of computers or computing, it contains a very brief and highly simplified description of the basic hardware elements out of which computers are constructed, and goes on to present some basic concepts of computer programs and programming. Since this book is aimed at readers who typically already have some familiarity with programming, these two sections may be skipped by such people without

loss.

The remaining three sections of this introductory chapter present the FORTRAN language as such. FORTRAN has evolved considerably since it was first introduced about thirty years ago, and these sections describe its history, current status and likely future, this last section being inevitably somewhat speculative.

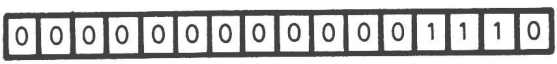
1. Computer Hardware

A large computer is one of the most complex and ingenious pieces of machinery ever devised by man. Fortunately, the typical programmer has to understand rather little about its inner workings, and can often write small-scale programs without great concern for what goes on behind the scenes, although most large-scale applications require more detailed knowledge of the capabilities and limitations of the hardware.

1.1 Storage

Computing is concerned with the manipulation or processing of *data.* These data may be lists of numbers, school records, company accounts, airline schedules, or any other kind of information suitable for automatic processing. In order to have the data available in a form in which it may be readily accessed, every computer has a *main memory* or *store.* A store is normally divided into *words,* each word containing one item of information, as shown in Fig. 1. The number of words in a store varies from several thousand on very small microprocessors, to several hundred million on large super-computers.

Each word in the store is composed of more elementary units known as *bits.* A bit can have only one of two values, 0 or 1, often represented physically by the on or off state of a minute electronic switch. If a word contains m bits, then the word itself can be in any of 2^m different states. The actual meaning assigned to a particular pattern of 0's and 1's in a word depends on the individual computer model, but almost all computers store the positive integers, 0, 1, 2, 3 *etc.* in a way whereby each bit which is set to 1 in the word represents a value which is $2^{(n-1)}$, where n is the position of the bit within the word counting from the right. The number 14 can thus be represented by the bit string 1110, *i.e.* the sum of $2^{(2-1)}$, $2^{(3-1)}$ and $2^{(4-1)}$, or pictorially:



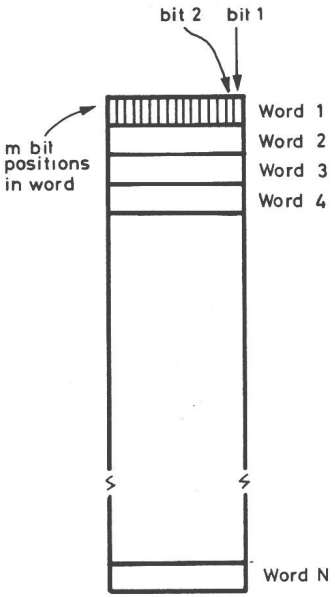


Fig. 1 A computer store

Many present day computers have stores with 32 bits to a word, with an intermediate division of the word into four groups of eight contiguous bits. Such a group of eight bits is known as a *byte*. A byte has 2^8 or 256 different states, and may be used to store, for instance, the different representations of the characters in a character set of upper and lower case letters, numbers, punctuation marks *etc.*

Access to words in main memory is fast, but the memory itself is expensive. Since the amount of data a computer is expected to process often far exceeds the capacity of the main memory to store it, there is usually another level of storage which has a much larger capacity, but is less expensive per unit of storage. This is known as a backing store, and on most modern computer systems consists of various types of rotating disc packs. This type of storage has the drawback that the access to the data is slower than for main memory.

A further level of storage is provided by magnetic tapes, which have to be physically mounted onto units before they can be written or read, but which are very cheap, and each can hold quantities of data exceeding thousands of millions of bits.

We thus see that there is a hierarchy of storage, from small, fast but expensive main memory, to large, slow but cheap dismountable tapes. A further level on many computers is a small,

high-speed memory known as a *cache memory* which acts as a buffer between the rest of the storage system and the device which exploits it, the central processing unit.

1.2 Central processing unit

The heart of a computer is its central processing unit (CPU). This is the device which performs the actual processing of the data. For instance, it may take the contents of two specified locations in memory, add them together and return the result to a third location. To be able to perform this operation, the operands will be fetched from memory and placed in high-speed *registers* in the CPU. These are a small store of usually a few dozen words inside the CPU itself, to which access is extremely fast, allowing the CPU to operate on their contents at its full speed, without having to wait for operands to be fetched from main memory, once they have been placed in the registers.

The main part of the CPU is the arithmetic and logic unit (or units), which is a device capable of performing various *operations* on *operands* held in the registers. These operations are the familiar arithmetic operations of add, multiply, subtract and divide, as well as others which, for example, allow the bits of a word to be shifted in position to the left or right, to be complemented, masked or to be logically combined with the bits of another word. A CPU and its registers are shown diagrammatically in Fig. 2, and a fuller description of several actual models can be found in Metcalf (1982).

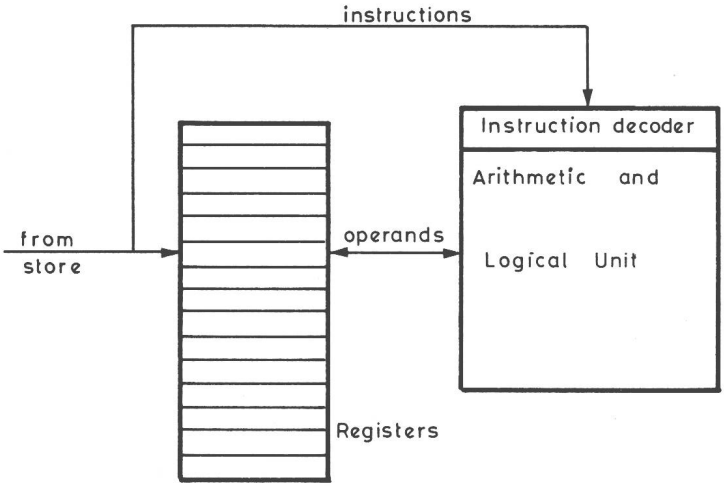


Fig. 2 A central processing unit

1.3 Instructions

A CPU has not only to be supplied with data, but also requires the necessary orders on how the data are to be manipulated. These orders are contained in *instructions* which consist of coded information about which operands are to be operated on, and by which operators. The instructions control directly the movement of data words between the main memory and the registers in the CPU.

The instructions themselves are stored just like data words in the main memory. In many computers, a special register points to the location in memory which contains the next instruction to be executed. When the current instruction has been carried out, the next instruction word is fetched from memory into a special part of the CPU, where it is first decoded, and the orders it contains then executed. This is also shown in Fig. 2. Since at the level of the storage an instruction word is treated just like a data word, it is perfectly possible for the CPU to modify instructions stored in memory.

1.4 Access to a computer

Nowadays there are two principal ways in which computers are used and accessed. In the first, the computer is relatively small, and sits on a desk top or in a corner of an office, and is operated interactively *via* a *visual display unit* or *terminal.* This consists of a keyboard which may be used to send *commands* to the computer, and a screen on which the computer displays requests for commands and the responses to those requests. The operation of the terminal may be under direct control of the CPU of the computer to which it is connected, or be controlled indirectly by other hardware.

The second method of use is once again *via* a terminal, but in this case the computer to which it is connected is located remotely, possibly in the same building but perhaps many kilometers away. Here, the computer in question will be larger, and there may be as many as several hundred people using it simultaneously. The connection between the terminals and the large computer will often be under control of smaller computers. In many cases, a number of large computers will be linked together in a *network,* or be connected *via* telephone lines, and it will be possible for a user sitting at his terminal to connect from one computer to another, and even to use a computer situated in another country or continent.

2. Computer Software

The computer hardware which has just been outlined operates under the control of sequences of instructions. These instructions are stored in memory, and fetched by the CPU for decoding and execution. A long sequence of instructions might be only partially stored in main memory, the remainder being kept in the backing store until needed. A sequence of instructions to perform a defined task is known as a *program*. A collection of programs to control the operation of the hardware of a computer and of the application programs submitted by users is known as an *operating system*.

Programs are written in a precisely defined code, or *programming language*. These exist in various forms. Those languages which require a detailed knowledge of the hardware of a specific computer are known as *low-level languages*. Others written in a more abstract way requiring little or no such knowledge are called *high-level languages*. FORTRAN is such a language. It is written in a manner akin to mathematical formulae, and is translated into the instructions required to drive the CPU by another program which is part of the operating system and known as a *compiler*. The compiler reads the FORTRAN *source code,* checks that the syntax is correct, *i.e.* that it conforms to the rules of FORTRAN grammar, and generates the instructions in the form of a so-called *object code*. In order for this code to be executed, it must first be correctly placed in the computer's memory and the execution initiated. This is the task of other programs, which are also part of the operating system and known as *loaders* and *linkage-editors*.

Programs are collectively known as computer software. A given piece of hardware – or computer – can in principle be operated and used by many different suites of software. In practice, very few different operating systems exist for a given type of computer, as it requires a large investment in manpower and hence money to write such a system. Application programs to solve a small problem, on the other hand, may often be written in an afternoon by one person, although very large application programs are also written, comparable in size and complexity to an operating system. In this book, we shall be concerned with just one language used to write application programs varying in size from a few lines to hundreds of thousands of lines.