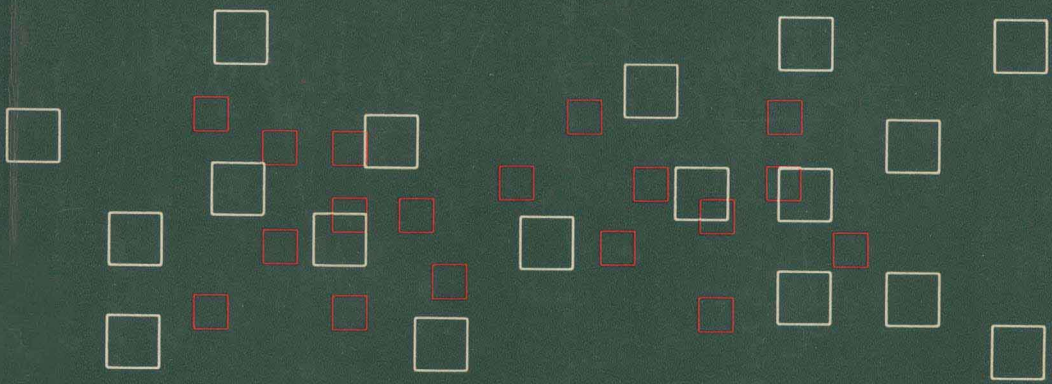
FUNDAMENTALS OF MICROCOMPUTER PROGRAMMING INCLUDING PASCAL



Daniel R. McGlynn

Fundamentals of Microcomputer Programming, Including Pascal

DANIEL R. McGLYNN

A Wiley-Interscience Publication
JOHN WILEY & SONS
New York Chichester Brisbane Toronto Singapore

Library of Congress Cataloging in Publication Data:

McGlynn, Daniel R.
Fundamentals of microcomputer programming, including Pascal.

"A Wiley-Interscience publication."
Bibliography: p.
Includes indexes.
1. Microcomputers—Programming. 2. Pascal (Computer program language) I. Title.

QA76.6.M4 1982 001.64'2 82-8645
ISBN 0-471-08769-6 AACR2
ISBN 0-471-08769-6 (pbk)

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Fundamentals of Microcomputer Programming, Including Pascal

Preface

This book is an introduction to some of the fundamental concepts in computer programming languages, in particular, the language Pascal, for microcomputers. In the last decade, advances in semiconductor technology have reduced the size and cost of computers to the point where desktop units are now readily accessible to consumers, small businesses, and educational institutions. Although the available computer hardware offers a wide variety of capabilities for all applications, the software—or computer programs—for such microcomputers remains a major hurdle for many users.

One relatively new language that offers considerable potential for a wide variety of microcomputer users is Pascal. It is widely used in colleges and universities for teaching programming languages, and increasing numbers of mini- and microcomputers now offer Pascal software.

However, Pascal is a relatively sophisticated programming language and is usually learned by those who already know another programming language like BASIC or FORTRAN. There are many who argue that Pascal should be taught as the first programming language so that the user's thought processes are not affected by the "bad habits" found in languages like BASIC and FORTRAN. This book offers one approach to Pascal as a first programming language.

The teaching of computer programming and programming languages has traditionally been from an applications-oriented perspective: computer programming for business majors, computer programming for engineers, and so on. This book approaches the teaching of computer programming from a computer science perspective, that is, computer programming as it is related to computer hardware and computer linguistics.

This book is written for the person who wants a thorough understanding of modern concepts in computer programming and programming languages and wants to apply that understanding in using Pascal or other structured programming languages. Since it assumes no previous background in programming, this book is suitable for introductory courses in programming or computer science at the college level, or for professional development courses.

It is not, however, intended as another book in the spirit of "how to program in Pascal." The book is intended to give the reader a greater appreciation of programming and computer software and, within that context, to present the fundamentals of the Pascal programming language. It could be said that the intent of the book is analogous to a course in art appreciation rather than in painting.

A revolution is taking place in education, based partly upon the availability of inexpensive personal computers for home and classroom use, and partly upon the development and widespread acceptance of prepackaged software and simple, easy-to-learn programming languages like BASIC. It is pertinent to ask what impact these developments can have on computer education. Other academic fields have had their revolutions—consider the "new math" as a case in point. It took only a few years of instruction in abstract concepts of sets and mappings before the books entitled "Why Johnny Can't Add" began appearing in bookstores (which was, incidentally, long before the $10 electronic calculator was on the market).

It is therefore pertinent to at least pose the question about the impact of prepackaged software on the next generation of programmers. Is the specter of "Why Johnny Can't Program" a real one? In raising such a question, we are not concerned with the students who have really mastered a high-level language like Pascal and understand programming, any more than the critics of the new math were concerned with students who actually learned arithmetic, either with the assistance of, or in spite of, the new math. Instead, we are concerned that the users of microcomputers should have a greater appreciation and understanding of computer programming.

Chapter 1 gives the reader an overview of computer hardware and software from the perspective of modern mini- and microcomputers. Chapter 2 presents a summary of the field of computer linguistics, in order to give an appreciation of how computer programming languages are related to natural languages, and how the field of computer science characterizes such languages. Chapter 3 presents the different types of computer programming languages on a conceptual level—from low level to high level languages, from sequential to concurrent processing. An introduction to the important concept of data flow languages is also presented. Chapters 4 and 5 tackle the subject of programming, from problem definition to a consideration of programming costs. Chapters 6 through 8 present the Pascal language in a fairly thorough manner, while Chapter 9 contains a number of simple programs in Pascal for different applications.

As the title of this book suggests, the microcomputer implementations of Pascal are probably of most interest, particularly for the first-time programmer. Chapter 10 describes the features and differences between a number of the most popular microcomputer implementations of the language. Chapter

11 describes in detail the widespread and popular UCSD Pascal used in a number of microcomputer implementations.

Finally, since Pascal is only one step in the evolution of programming languages, Chapter 12 briefly describes two languages—Modula-2 and Ada—which are evolutionary descendants of Pascal.

The author is grateful for the assistance of the Pascal Users' Group for use of the ISO Draft Standard on Pascal, and David V. Moffat for the use of his extensive bibliography.

DANIEL R. MCGLYNN

Anaheim, California
September 1982

Fundamentals of Microcomputer Programming, Including Pascal

Contents

1

Communicating with the Microcomputer

. . . [T]he next postindustrial revolution would be the silicon revolution of information processing and computers that would . . . change the face of the earth.

Jean-Jacques Servan-Schrieber,
Le Défi Mondial (1980)

When the microprocessor or "computer-on-a-chip" was introduced in the early 1970s, one of the major challenges facing the prospective users was how to communicate with it. Semiconductor technology had advanced much more rapidly than computer programming and computer languages, and the full capability of the early microprocessors was not realized. Today microcomputers are accessible to consumers, small businesses, and educational institutions, and the software—though much more highly developed than in the mid-1970s—still remains a major hurdle for most users.

Microcomputer programming and programming languages are basically concerned with communicating with a microcomputer. The present-day microcomputer users are, however, different from the original users of microprocessors in the early 1970s, and even different from the computer hobbyists who assembled computer kits in the mid-1970s and "programmed" such computers by entering instructions bit by bit on front panel sense switches. The present-day microcomputer user approaches the computer from an applications perspective—thinking in concepts of storing customer names in one file, and zip codes in another file.

Although it is instructive to consider the position taken by some that computers are intended to solve real-world problems expressed by users in such an applications-oriented perspective, the fact is that programming languages are adapted to computer hardware that exists, rather than computer hardware being designed to execute specific programming languages. Until such time that special-purpose, dedicated computer hardware is designed to execute specific languages or solve specific applications, in order to understand the task of communicating with a microcomputer, one must begin with the structures and elements of the microcomputer which participate in such communication.

With such a viewpoint explicitly stated, we consider the following basic issues:

computer architecture
hardware/software interface
types of computer programs
information theory

COMPUTER ARCHITECTURE TYPES

There are a number of different ways to classify computer architectures, depending whether one is looking at the computer from a hardware, software, or operational perspective. Since this book is concerned with computer programming and software, it is appropriate to consider a classification based upon software, or more particularly, the proximity of the user programming language and the language actually executed by the machine. The classification is:

von Neumann architecture
syntax-oriented architecture
indirect execution architecture
direct execution architecture

In studying computer programming and software, it is important that the reader keep in mind the basic relationship between the software and the type of computer architecture. The capabilities and limitations of a particular computer language, or of a particular program, are closely related to the type of computer architecture on which the language or program is implemented. Some computer languages operate much more efficiently with a

given computer architecture than with others; similarly, certain programs are executed much more efficiently on certain types of computer architectures.

Since von Neumann architecture is the most commonly implemented architecture, and the typical computer programmer will rarely encounter a non-von Neumann computer, the importance of the relationship of computer language and computer architecture is not particularly relevant from an immediate, practical viewpoint. However, it must be realized that as semiconductor and microprocessor technology advances, it will become more and more feasible to implement non-von Neumann architectures, and at such time the relationship between language and architecture will take on a new relevancy.

Microcomputer Architectures

A microcomputer is a computer that incorporates a microprocessor as the central processing unit. Since almost all commercial microprocessors are designed with a von Neumann architecture, the broad categories of computer architectures discussed previously are not as relevant as a discussion of the system architectures implemented using microprocessors. System architecture refers not to the specific operations of the memories, registers, or control units that distinguish the broad categories of computer architectures, but to the interrelation of system components such as the central processor, peripheral processors, interfaces, operating systems, and similar components.

Microcomputers can be generally categorized according to their system architectures into five distinct categories:

microcontrollers
microcomputers
micromidis
micromaxis
micromainframes

The microcontroller is a relatively simple microprocessor-based computer typically directed to routine industrial control applications. The microprocessor employed is usually a "low end" microprocessor such as the Intel 8048 or 8022, the Zilog Z8, or the Mostek 3870, or similar 8-bit microprocessors.

The microcomputer is a term for a general-purpose microprocessor-based computer system. When used in the context of system architecture classification, the term microcomputer refers to a midrange 8- or 16-bit microproces-

sor-based system, such as an Intel 8080 or 8085, or even the 16-bit Intel 8086 or Zilog Z8000, in simple system configurations.

The micromidi is a 16-bit microprocessor-based computer system with the characteristics of a "midicomputer," a term that refers to a computer having capabilities greater than a minicomputer but less than a maxicomputer or mainframe. An example of a midicomputer could be the IBM System/32. The type of microprocessor used to implement a micromidi would be a 16-bit microprocessor such as the Intel 8086 or Zilog Z8000 in an extended configuration. Micromidi systems are only now just becoming feasible to implement with the commercial availability of such microprocessors and peripheral processors.

The micromaxi and micromainframe are two microprocessor system architectures that are expected to be made possible with the next generation of 16-bit and 32-bit microprocessors. Such systems refer to computers having the architecture of the present-day commercially available maxicomputers or mainframe computers. The next generation of microprocessors which will make these systems possible is expected to include a pseudo-32 bit processor (i.e., 32-bit internal architecture with 16-bit external buses), as well as, eventually, actual 32-bit processors.

Associated with these levels of system architectures are corresponding levels of software. One proposed system architecture for future microcomputer systems contemplates the use of modularized programs which can be used as building blocks for more complex systems. Some of the most important modular programs to be developed are various operating system kernels. By selecting and combining appropriate operating system kernels, the user will be able to define different software levels and capabilities.

User-Level Architecture

The discussion of computer and microcomputer architectures in the preceding sections is, practically speaking, of little relevance to the computer user. Although a knowledge of machine architecture is useful for fully understanding the operation of the computer, as well as its capabilities and limitations, the basic machine architecture generally cannot be easily changed by the user. In many instances the particular type of machine architecture is not even important to the user's application.

However, certain aspects of machine architecture are visible and even important to the user. We call such aspects of the machine architecture which are apparent to the user or programmer the "user-level architecture." All other features of the machine—whether hardware or software—which the user is not aware of are called "transparent" features.

The concept of different levels of computer architecture will be developed in greater detail in the next two sections on the hardware/software interface, and in a discussion of technological hierarchies. It is merely important to note at this point that although two different computers may appear the same, operationally, from the perspective of the user, the machines may have completely different architectures.

User-level architecture is a relative term depending upon the particular user, rather than the particular hardware configuration. One user sitting at the console of the computer may communicate with certain features of the computer in a completely different manner than another user sitting at a remote location and communicating with the computer via a terminal. Although both users have their instructions executed by the same hardware, the nature of their interaction or "interfacing" with the computer is different. In effect, the two different users are interacting with two different user-level architectures.

The user-level architecture is principally defined by software. Although computer architecture basically describes the computer hardware, computer software is a very important component of a system's architecture. The architecture that is apparent to the user is a combination of hardware and software features that dynamically interact. The nature of this hardware/software interface and interaction, and the relationship of the user to this interface are described in the next section.

HARDWARE/SOFTWARE INTERFACE

The preceding discussion of computer architecture focused on the hardware aspects of computer design. Another important aspect of the computer design is the role of computer software at all levels of the architectures, and the interface between such software and the computer hardware.

In order to place the hardware/software interface in perspective, we consider the following issues in the present section:

basic definitions
language levels

Basic Definitions

Before turning to the discussion of the hardware/software interface, it is important to establish definitions for some of the terminology. Some key terms are:

programming language
grammar
language level
metalanguage
string
instruction
data

Simply defined, a programming language is the means used for communicating instructions to the computer. Computer languages will be described in much greater detail and much more precisely in the following two chapters; however, for our purposes here, such an inferential definition is adequate.

We already mentioned various programming languages such as FORTRAN, COBOL, Pascal, and ALGOL. It must be emphasized that these are programming languages employed by a programmer of a general purpose computer to communicate with it.

A grammar is a set of rules that describe or define the programming language. Grammars will also be described in greater detail in the next chapter.

The language level is an informal designation of the degree of complexity of the operations described by a single communication in a language. These levels are listed in Table 1.1.

A metalanguage is a language used to describe a programming language.

A string is an ordered sequence of characters in a programming language, and may represent a statement, a computation sequence, or a data element to be processed.

An instruction is an element of a programming language which defines a specific operation or sequence of operations to be performed.

Data is any information other than instructions or control operations handled by the processor.

Language Levels

Just as there are different levels of architecture from user architecture to machine architecture, there are different levels of languages associated with the computer. Another way to describe a language level is as the relative degree of complexity of the operation described by a single communication at that level.

Table 1.1 lists five language levels from the simplest and most closely related to the machine hardware, to the most complex and most closely related to the user.