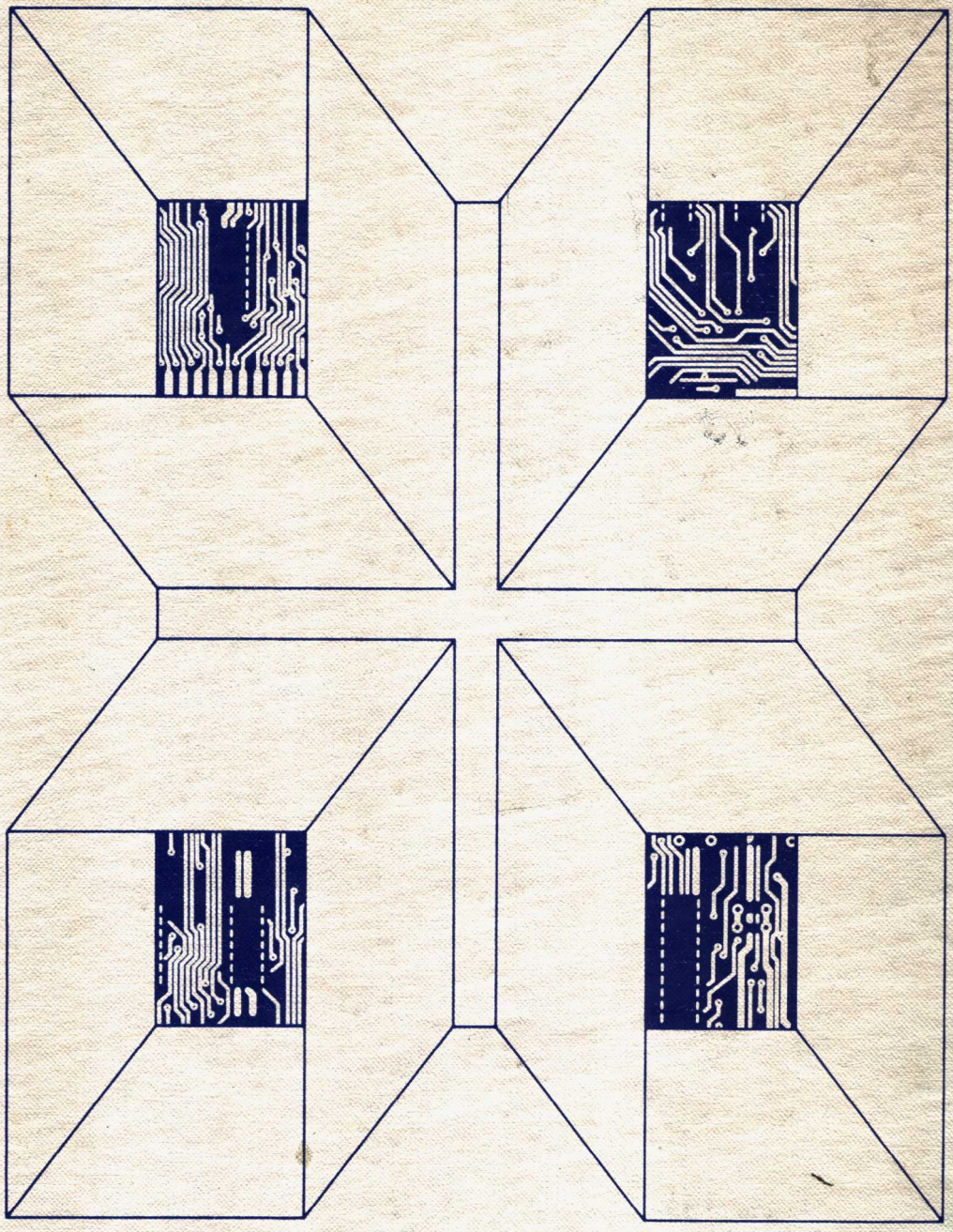


computer organization and assembly language programming

james l. peterson

COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE PROGRAMMING

JAMES L. PETERSON
UNIVERSITY OF TEXAS AT AUSTIN

ACADEMIC PRESS
NEW YORK SAN FRANCISCO LONDON
A Subsidiary of Harcourt Brace Jovanovich, Publishers

ACADEMIC PRESS, INC.
111 FIFTH AVENUE, NEW YORK, NEW YORK 10003

UNITED KINGDOM EDITION PUBLISHED BY
ACADEMIC PRESS, INC. (LONDON) LTD.
24/28 OVAL ROAD, LONDON NW1

ISBN: 0-12-552250-9
LIBRARY OF CONGRESS CATALOG CARD NUMBER: 77-91331

PRINTED IN THE UNITED STATES OF AMERICA

COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE PROGRAMMING

This is a volume in
COMPUTER SCIENCE AND APPLIED MATHEMATICS
A Series of Monographs and Textbooks
EDITOR: WERNER RHEINBOLDT

PREFACE

This book has been designed and used as a text for a second course in computer programming. It has developed from class notes for a course offered at the University of Texas at Austin to undergraduate students. These students have had one previous programming course and should know, from that first course, the basic operation of computers, in general, and have some basic skills in converting problem statements into programs in a higher level language, such as Fortran. The second course, and this text, assumes that the student knows how to program, i.e., how to find an algorithm to solve a problem and convert that algorithm into a program.

The purpose of this book is to teach the student about lower level computer programming: machine language and assembly language, and how these languages are used in the typical computer system. This is meant to give the student a basic understanding of the fundamental concepts of the organization and operation of a computer. Even if the student never again programs in assembly language (and we would hope that they never have to!) it is important that they understand what the computer is doing at the machine language level. A good understanding of computer organization translates into a better understanding

of the features and limitations of all computer facilities, since all systems must eventually rest on the underlying hardware machine.

The content of this text follows the recommendations of the ACM Curriculum 68 for Course B2 "Computers and Programming." After a brief review of the general concepts of computers in Chapter 1, the remainder of the text uses the MIX computer to provide an example machine for illustrating computer organization and programming. Chapter 2 and Chapter 3 present the architecture of the MIX computer, its machine language and the MIXAL assembly language. Programming techniques in assembly language are covered in Chapters 4 and 5 with Chapter 5 concentrating mostly on input/output programming. The use and implementation of the subroutine concept is investigated in Chapter 6.

Systems programs are considered in the next three chapters. Chapter 7 explores loaders, while Chapter 8 discusses assemblers. In Chapter 7 the code for a simple, but real, absolute loader is given; in Chapter 8, the code for a MIXAL assembler is given. These two programs provide an opportunity for the student to see and study a real loader and assembler, and not simply the concepts in the abstract. Chapter 9 briefly discusses other system programs, macro assemblers, compilers, interpreters, and operating systems.

From these chapters, the basic concepts of assembly language programming and programs should be evident to the student. Chapter 10 then proceeds to present a brief description of several other computers, to introduce the student to both the similarities and differences among computer systems.

In our one-semester course, these concepts are reinforced by numerous programming assignments. The early assignments emphasize basic programming techniques such as simple arithmetic, input/output, character manipulation and array handling. The later assignments have included writing either a relocatable loader and two-pass assembler for a subset of the MIX computer, or writing an interpreter and one-pass load-and-go assembler for a simple minicomputer (16 instructions, four general registers, etc.). All of these assignments are programmed in MIXAL. The last assignment is to write a simple program in the new assembly language of their own assembler. Thus, students should see that they know how to program in assembly language, in general, and not simply in MIXAL.

The major question in your mind now is undoubtedly: Why MIX? MIX is a pseudo computer, not a real one. This is at once both its major drawback and its major advantage. The major drawback to MIX is, of course, that it is not real; this implies that the use and programming of the MIX computer will include a certain air of artificiality which may annoy and confuse some students.

However, from an educational point of view, MIX is ideal. It is simple, easy to understand, and yet typical of many computers. Machine and assembly languages are different for each computer. However, the techniques of assembly language programming are largely machine independent. Thus, learning one assembly language provides the basis for quickly and easily learning any other

assembly language. This is emphasized by the descriptions of other computers in Chapter 10.

Also consider the alternative to teaching MIX: teaching the structure and language of a real computer. As Knuth has written, in the Preface to Volume 1 of *The Art of Computer Programming* (Addison-Wesley, Reading, Mass., 1973),

> "Given the decision to use a machine-oriented language, which language should be used? I could have chosen the language of a particular machine X, but then those people who do not possess machine X would think this book is only for X-people. Furthermore, machine X probably has a lot of idiosyncrasies which are completely irrelevant to the material in this book, yet which must be explained; and in two years the manufacturer of machine X will put out machine X + 1 or machine 10X, and machine X will no longer be of interest to anyone".

Knuth continues that it is very unlikely that programmers will only use one computer in their life. Each new machine can be easily learned once the first machine is understood, but the ability to change smoothly from one computer to another is an important skill for a programmer. Thus, teaching first MIX and then another, real, computer is preferable, since it immediately forces the student to understand how to move from machine to machine. In my own, so far short career, I have programmed on several different computers (IBM 1620, CDC 3600, CDC 6500, PDP-11/20, HP 2116, CDC 1700, IBM 360/370, DEC-10, SDS Sigma 5, Nova 3/D).

From an economic viewpoint also, the MIX machine is an advantage over a real computer. It is often said that a simulated machine is much more expensive than a real machine, and for production computation this is undeniably true. However, for a student environment, most of the computer time is in assembly and debugging, *not* execution. The simple MIXAL assembler, written as a cross assembler for the machine at hand, generating load-and-go code for a simulator with good trace, dump, and error detection facilities will provide a much better instructional tool at a lower price than most real assemblers with their extensive pseudo instructions, macros, relocatable code, and operating system input/output, most of which cannot and need not be used in an introductory course.

The construction of a MIXAL assembler/simulator is, although nontrivial, within the range of a senior year or early graduate student project. The complexities are derived mainly from the need for an event driven simulator to allow CPU and I/O overlap, and the need to provide the best possible debugging facilities. Properly written, the design and code for these systems would be easily transported over the years to new computer systems.

One further benefit from the use of MIX is the ability to easily pick up and use *The Art of Computer Programming* books by D. E. Knuth. These are very handy in later courses as references and texts. More can be learned from them with a good knowledge of MIX.

In summary, we feel that MIX is preferable to any real machine for teaching a beginning course in machine language, assembly language and computer organization. The major problem we have faced in using MIX has been the lack of an adequate text, a problem which we hope has now been solved.

I would especially like to express my gratitude to the reviewers—Stan Benton, Montclair State College; Michel Boillot, Pensacola Junior College; Werner Rheinboldt, University of Maryland; and Robert C. Uzgalis, University of California—Los Angeles; whose comments and suggestions helped greatly in guiding the manuscript to its final form.

Austin, Texas
August 11, 1977

James Peterson

CONTENTS

1
BASIC COMPUTER ORGANIZATION 1

1.1 The Memory Unit 3
1.2 The Computation Unit 11
1.3 The Input/Output System 41
1.4 The Control Unit 62
1.5 Summary 65

2
THE MIX COMPUTER SYSTEM 67

2.1 The MIX Computer Architecture 68
2.2 Machine Language 75
2.3 Introduction to Assembly Language 81
2.4 MIXAL: MIX Assembly Language 84
2.5 Summary 100

3
A DETAILED DESCRIPTION OF THE MIX COMPUTER 104

3.1 Instruction Interpretation and Execution 104
3.2 Effective Operand Calculation 108
3.3 Partial Field Specifications 114
3.4 Loading Operators 117
3.5 Storing Operators 118
3.6 Integer Arithmetic Instructions 119

3.7 Floating Point Arithmetic Instructions 120
3.8 Comparison Operators 121
3.9 Jumps 122
3.10 Immediate Operators 123
3.11 Input/Output Instructions 125
3.12 Shift Instructions 126
3.13 Miscellaneous Commands 127
3.14 Binary Instructions 128
3.15 Instruction Execution Times 129
3.16 Summary 130

4
ASSEMBLY LANGUAGE PROGRAMMING TECHNIQUES 132
4.1 Arithmetic 134
4.2 Jumps 139
4.3 Loops 140
4.4 Arrays 142
4.5 Stacks 148
4.6 Character Manipulation 149
4.7 Lexical Scanning 157
4.8 Summary 159

5
INPUT/OUTPUT PROGRAMMING 161
5.1 Basic I/O Programming Concepts 161
5.2 Programming MIX I/O Devices 165
5.3 A Simple I/O Program 173
5.4 Overlapped I/O 177
5.5 Blocking 187
5.6 Summary 193

6
SUBROUTINES AND PARAMETERS 196
6.1 Subroutine Structure 198
6.2 Parameters 203
6.3 Call by Value, Reference, or Name 214

6.4 The Cost of Subroutines 222
6.5 Other Topics about Subroutines 223
6.6 Summary 229

7
LOADERS AND LINKERS 232
7.1 Absolute Loaders 234
7.2 Relocatable Loaders 244
7.3 Variations of Loaders 259
7.4 Summary 263

8
ASSEMBLERS 266
8.1 Data Structures 267
8.2 General Flow of an Assembler 273
8.3 An Example Assembler 283
8.4 Summary 322

9
SYSTEMS PROGRAMS 325
9.1 Macro Assemblers 326
9.2 Conditional Assembly 336
9.3 Compilers and Higher Level Languages 341
9.4 Interpreters 343
9.5 Operating Systems 344
9.6 Other Systems Programs 346
9.7 Systems Programming Languages 346
9.8 Summary 348

10
SOME COMPUTER ARCHITECTURES 350
10.1 A History of Computers in the United States 351
10.2 The PDP-8 355
10.3 The HP 2100 369
10.4 The PDP-11 378
10.5 The IBM SYSTEM 360 and SYSTEM 370 387
10.6 The Burroughs B5500 396

10.7 The CDC 6600 403
10.8 The INTEL 8080 411
10.9 Summary 419

Appendix A. References 423

Appendix B. The MIX Instruction Set 426

Appendix C. MIX Symbolic Opcodes—Alphabetic Order 431

Appendix D. MIX Symbolic Opcodes—Numeric Order 436

Index 441

1

BASIC COMPUTER ORGANIZATION

Computers, like automobiles, television, and telephones, are becoming more and more an integral part of our society. As such, more and more people come into regular contact with computers. Most of these people know how to use their computers only for specific purposes, although an increasing number of people know how to program them, and hence are able to solve new problems by using computers. Most of these people know how to program in a higher-level language, such as Basic, Fortran, Cobol, Algol, or PL/I. We assume that you know how to program in one of these languages.

However, although many people know how to use automobiles, televisions, telephones, and now computers, few people really understand how they work internally. Thus, there is a need for automotive engineers, electronics specialists, and assembly language programmers. There is a need for people who understand how a computer system works, and why it is designed the way that it is. In the case of computers, there are two major components to understand: the hardware (electronics), and the software (programs). It is the latter, the software, that we are mainly concerned with in this book. However, we also consider how the hardware operates, from a programmer's point of view, to make clear how it influences the software.

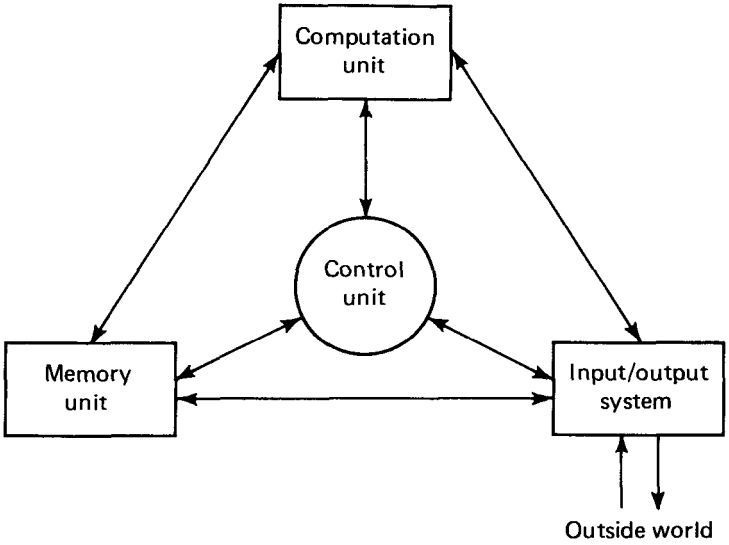


FIGURE 1.1 Basic components of a computer.

This chapter reviews the basic organization of computers, to provide a background for the software considerations of later chapters. You should be familiar with most of this material already, but this review will assure that we all agree on the basic information.

We can understand a computer by studying each of its components separately and by examining how they fit together to form a computing system. The four basic components of a computer are shown in Figure 1.1. These four elements are: (a) *a memory unit*, (b) *a computation unit*, (c) *an input and output system*, and (d) *a control unit*. The arrows between the different components in Figure 1.1 illustrate that information *may* travel freely between any two components of the computer. Some information paths may be missing in some computers. For example, in many systems there is no direct connection between the computation unit and the input/output system.

The memory unit functions as a storage device in the computer, storing data and programs for the computer. The computation unit does the actual computing of the computer—the additions, subtractions, multiplications, divisions, comparisons, and so on. The input/output (I/O) system allows the computer to communicate with the outside world, to accept new data and programs and to deliver the results of the computer's work back to the outside world. The control unit uses the programs stored in the computer's memory to determine the proper sequence of operations to be performed by the computer. The control unit issues commands to the other components of the system directing the memory unit to pass information to and from the computation unit and the I/O system, telling the computation unit what operation to perform and where to put the results, and so forth.

Each of these components is discussed in more detail below. Every computer must have these four basic components, although the organization of a specific computer may structure and utilize them in its own manner. We are therefore presenting *general* organizational features of computers, at the moment. In Chapters 2, 3, and 10 we consider specific computers and their organization.

1.1 THE MEMORY UNIT

A very necessary capability for a computer is the ability to store, and retrieve, information. Memory size and speed are often the limiting factors in the operation of modern computers. For many of today's computing problems it is essential that the computer be able to quickly access large amounts of data stored in memory.

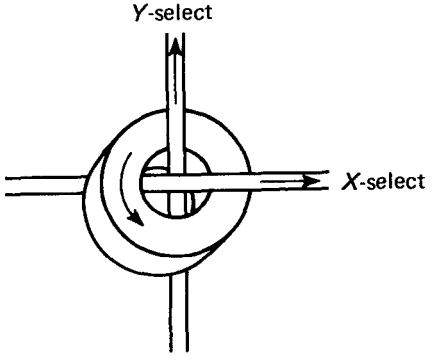
We consider the memory unit from two different points of view. We first consider the *physical* organization of a memory unit. This will give us a foundation from which we can investigate the *logical* organization of the memory unit.

Physical organization of computer memory

For the past twenty years, the *magnetic core* has been the major form of computer memory. More recently, *semiconductor memories* have been developed to the point that most new computer memories are likely to be semiconductor memories rather than core memories. The major deciding factors between the two have been speed and cost. Semiconductor memories are undeniably faster, but until recently have also been more expensive.

Core memories have been used for many years and will undoubtedly continue to be used widely. They have been the main form of computer main memory for almost twenty years. Since semiconductor memories have been trying to replace core memories, they have been built to look very much like core

FIGURE 1.2 A magnetic core (much enlarged).



memories, from a functional point of view. Therefore, we present some basic aspects of computer memories in terms of core memories first, and then we consider semiconductor memories.

Core memories

Figure 1.2 is a drawing of a magnetic core. Cores are very small (from 0.02 to 0.05 inches in diameter) doughnut-shaped pieces of metallic ferrite materials which are mainly iron oxide. They have the useful property of being able to be easily magnetized in either of two directions: clockwise or counterclockwise. A core can be magnetized by passing an electrical current along the wire through the hole in the center of the core for a short time. Current in one direction (+) will magnetize the core in one direction (clockwise), and current in the opposite direction (−) will magnetize the core in the opposite direction (counterclockwise). Once the core has been magnetized in a given direction, it will remain magnetized in that direction for an indefinitely long time (unless it is deliberately remagnetized in the opposite direction, of course). This allows a core to store either of two *states*, which can be arbitrarily assigned the symbols 0 and 1 (or + and −, or A and B, etc.).

FIGURE 1.3 Core plane showing X-select and Y-select wires and sense wire.