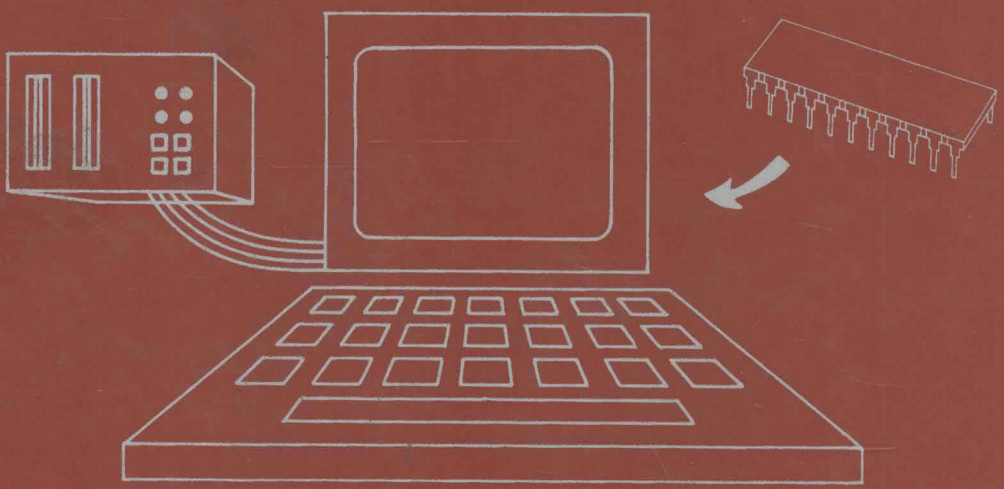
COMPUTING SYSTEM FUNDAMENTALS

an approach based on microcomputers

Kenneth J. Danhof
Carol L. Smith



COMPUTING SYSTEM FUNDAMENTALS

An Approach Based on Microcomputers

KENNETH J. DANHOF
CAROL L. SMITH
Southern Illinois University at Carbondale

ADDISON-WESLEY PUBLISHING COMPANY

Reading, Massachusetts • Menlo Park, California
London • Amsterdam • Don Mills, Ontario • Sydney

This book is in the
ADDISON-WESLEY SERIES IN COMPUTER SCIENCE

Consulting Editor: Michael A. Harrison

Library of Congress Cataloging in Publication Data

Danhof, Kenneth J
Computing system fundamentals.

Includes bibliographical references and index.
1. Electronic digital computers. 2. Microcomputers.
I. Smith, Carol L., joint author. II. Title.
QA76.5.D252 001.6'4 79-14933
ISBN 0-201-01298-7

ISBN 0-201-01298-7
ABCDEFGH-MA-89876543210

COMPUTING SYSTEM FUNDAMENTALS

An Approach Based on Microcomputers

PREFACE

This book is intended as a textbook for an introductory course on computer organization and systems at the sophomore or junior level. As such it corresponds to the CS3 course described in the ACM 1978 curriculum guidelines. The primary prerequisite is a first course in programming with a high-level language such as FORTRAN or PL/1.

There are two salient features in the approach taken in this textbook. First, there is the matter of teaching the fundamentals of computer hardware and software (and the interplay between the two) and doing this in an unrestricted hands-on environment. The second feature is the incorporation of the most recent development of computer technology, the microprocessor, into the computer science curriculum.

Relative to the first issue, it is generally recognized that the future computer science or computer engineering student must understand both the hardware and software aspects of computers. Although in the past computer science curricula have tended to be either heavily software- or heavily hardware-oriented, the most recent curriculum specifications suggest a greater unification of hardware and software within one discipline. In fact, both the recent ACM and IEEE curriculum guidelines suggest the use of minicomputers or microcomputers in a hands-on laboratory situation as a means of achieving this goal. This text constitutes a realization of this approach.

Returning to the second basic feature of the book, we believe that, while the microcomputer revolution is currently spreading through various application areas, it has not really affected the computer science curriculum itself to a significant extent. This book is designed to correct the situation. It is apparent that not only should the microcomputer be clearly understood by computer science students, but that it is at precisely this level that one can best view and study the entire system. The course is constructed to utilize the microcomputer both as an object of study and as a laboratory tool. The low cost of microcom-

puters makes it possible to give each student essentially unlimited hands-on access to a microcomputer.

The course is structured in terms of programming "levels" and four "modules" support the development. Module I (Chapters 1–5) introduces the specific microcomputers to be studied—the Motorola 6800 and the Intel 8085. In this module, the student is taken from the machine language level to the assembly language level. Chapters 2 and 3 are devoted to the instruction sets of the two microprocessors. Chapter 4 introduces the basic programming constructs and compares the two instruction sets. Chapter 5 covers basic hardware components and discusses system architecture for the two microcomputers.

Module II (Chapters 6 and 7) introduces the students to system I/O programming. Chapter 6 covers typical cassette punch and load routines and Chapter 7 describes more general loaders and a simple text editor.

In Module III (Chapters 8, 9, 10), the students are introduced to a high-level programming language—PL/M. Chapter 8 describes PL/M and the corresponding structured-programming concepts. Chapter 9 covers the concepts involved in writing a two-pass assembler and thereby prepares the student for the task of producing a small resident assembler on the microcomputer. The last chapter in this module, Chapter 10, describes other software support packages that are typically found in a resident system and explores the microcode support for the machine-language level of the machine.

Module IV (Chapters 11 and 12) contains a discussion of interrupt I/O and some of the many application areas of microprocessors. Following a consideration of interrupt and parallel I/O support on each of the two systems, case studies are made of particular applications.

While moving through these four modules, the student transforms her or his own microcomputer from an initial naked piece of hardware into a functioning small computer system containing a loader, a text editor, and a resident assembler. Moreover, the student has written the programs that support this system.

The course presupposes a rather modest laboratory, which includes the microcomputers themselves (at least one of which has an extended memory) and a terminal. In addition, a support system centered around a computer sufficiently large to handle the necessary software development packages and a link between the support system and the microcomputers is needed. It should be noted that the ACM curriculum 1978 specifications recommend a laboratory of just this sort for computer science departments.

Although it is possible to cover most of the material in a three-hour, one-semester course, it may be desirable to omit some of the material in a given situation. It is possible, for example, to consider only one of the two microcomputers, or to provide the students with some of the programs (such as the text editor) which they might otherwise write. Moreover, various sections of the text (such as parts of Chapter 10 and much of Module IV) might be

regarded as optional material. An Instructor's Manual, which includes solutions to selected exercises and a description of a possible supporting microcomputer laboratory, is available. In addition, supporting software packages are available from the Computer Science Department, Southern Illinois University at Carbondale. These include the PL/M STAR Compiler and associated cross assemblers for the M6800 and I8085, as well as linkers and simulators for the two machines.

The authors wish to acknowledge Clovis Tondo, Debra May, Linda Foran, and Jeff Marks for their assistance in the development of the software support packages and David Alvin for his considerable help in developing the microcomputer laboratory and its support system. We are further indebted to the CS 306 students who took the course on an "experimental" basis and provided many helpful comments. Finally, we wish to thank our typists, June Blonde and Glenda Russell, who suffered through the several versions of the manuscript.

The course was developed with support from the National Science Foundation grant number SER77-02523.

Carbondale, Illinois K.J.D.
July, 1980 C.L.S.

CONTENTS

MODULE I WORKING WITH THE BASIC MACHINE

Chapter 1	**An Overview of Computer Organization**	**1**
1.1	Computing systems in general	1
1.2	The basic hardware components	6
1.3	Instruction formats	14
1.4	Addressing modes	20
1.5	Maxis, minis, and micros	24
Chapter 2	**The Motorola M6800 Instruction Set**	**27**
2.1	Chapter overview	27
2.2	The microprocessor unit	27
2.3	Accumulator and memory reference instructions	31
2.4	Index register and stack pointer instructions	41
2.5	Jump and branch instructions	44
2.6	Condition code register instructions	49
Chapter 3	**The Intel 8085 Instruction Set**	**51**
3.1	Chapter overview	51
3.2	The microprocessor unit	51
3.3	Data transfer instructions	54
3.4	Arithmetic instructions	58
3.5	Logical instructions	60
3.6	Branch control instructions	63
3.7	Stack, input/output, and machine control instructions	65

Chapter 4	**Programming with the M6800 and the I8085**	**69**
4.1	Programming techniques	69
4.2	Comparison of the two instruction sets	91
4.3	Program development techniques	93
Chapter 5	**Introduction to Microcomputer Architecture**	**97**
5.1	Basic circuits	97
5.2	Microprocessors and associated components	102
5.3	Hardware support for input/output functions	108
MODULE II	**INTERFACING WITH OTHER SYSTEMS**	
Chapter 6	**Automating the Loading Process**	**121**
6.1	Microcomputer development systems and systems programs	121
6.2	A typical cassette interface	122
Chapter 7	**System Interface Programs**	**137**
7.1	Bootstrap and absolute loaders	137
7.2	Relocating loaders	140
7.3	Linking loaders	143
7.4	Text editors	149
MODULE III	**BUILDING A RESIDENT SOFTWARE SYSTEM**	
Chapter 8	**High-Level Language Programming**	**153**
8.1	The PL/M programming language	153
8.2	Structured programming and software engineering	180
8.3	High-level versus assembly-language versus conventional machine-language programming	190
Chapter 9	**Assembly Language Support**	**195**
9.1	The assembler function	195
9.2	Macro assemblers	212
9.3	Conditional assembly	222
9.4	Cross, resident, and self-assemblers	225
Chapter 10	**Advanced System Software Support**	**227**
10.1	Compilers	227
10.2	Simulators and debuggers	241
10.3	Traversing the levels	247

MODULE IV MICROPROCESSOR APPLICATIONS AND INTERRUPT I/O

Chapter 11 Microprocessor Interrupt Systems 253
11.1 Interrupt-driven I/O 253
11.2 An example using interrupt I/O 263

Chapter 12 Applications and Future Directions 267
12.1 Utilizing microprocessors in dedicated systems 267
12.2 Interprocessor communication 270
12.3 Future directions 274

Appendixes

A ASCII Code 277
B M6800 Instructions Listed Numerically by Opcode 281
C I8085 Instructions Listed Numerically by Opcode 285
D Motorola M6800 and Assembler Conventions 289
E Intel 8085 Assembler Conventions 299
F PL/M STAR Syntax Specification 309
G PL/M STAR Built-in Functions 317

Bibliography 321

Index 325

CHAPTER ONE

AN OVERVIEW OF COMPUTER ORGANIZATION

1.1 COMPUTING SYSTEMS IN GENERAL

In framing a definition of what constitutes a computing system, we tend to be heavily biased by the types of interaction we have had with such a system. For the most part, modern computing systems are viewed as mysterious machines that are capable of performing wondrous tasks, such as computing space-flight paths or predicting election results, and/or making disastrous mistakes, such as refusing to acknowledge that you paid last month's bill.

The mystique about the nature of computing systems has been reinforced by the lack of physical contact that is permitted with the machine. This situation is common even within a computer-science curriculum. Although beginning students in computer science are exposed to the process of programming (giving instructions to) computing systems, this exposure tends to reinforce the mystical illusion rather than dispel it. These students normally deal with a machine that seems to require meaningless commands and all too often returns meaningless messages. It is rare that beginning students even get to see the machine, which is usually locked away in a secure room and attended by the high priests of the field, i.e., computer operators, programmers, and technicians.

The objective of this text is to unravel the mysteries surrounding computers and thereby derive an accurate definition for a computing system. Our premise is that the best way to accomplish this is to encourage close interaction with these systems. For this reason our text uses microcomputer systems as case studies. For the time being we define microcomputers as small, inexpensive computers. These two attributes—i.e., small physical size and low cost—make these systems ideal for our use.

One point that should be stressed is that, while we are using microcomputers as case studies, we are really investigating computing system concepts that are common to all computing systems, even the large expensive ones that are locked away in secure rooms. Although microcomputers are currently being sold in neighborhood electronics shops and used in computer games, they are

not toy systems. This fact is emphasized by the comparison figures shown in Table 1.1. In this table we compare the IBM 7090 computer, a very popular machine manufactured by International Business Machines in 1960, with a typical microcomputer system based on the M6800 microprocessor developed by Motorola in 1970. The figures in this table highlight the fact that, despite the small cost of microcomputers, the computing power of these machines is significant. The cycle time listed in the table is a measure of the amount of time needed to transfer information in the computer. The cycle time for the M6800 is less than the cycle time for the IBM 7090, but the former has a smaller information unit.

The fact that microprocessors are used in an ever increasing number of applications, including automotive parts, point-of-sale terminals, and games, is a result of the low cost and small size of these components. We should not overlook the fact that they are also powerful computing machines. Since we are stressing the power of microprocessors and emphasizing the fact that we'll be studying computing concepts that are relevant for all computers, an obvious question is "Why do some computing systems sell for over a million dollars while others are marketed for under 300 dollars?" The distinction between large machines (maxis), medium-sized machines (minis) and small machines (micros) will be discussed in Sec. 1.5, after we have introduced enough computer terminology to make the discussion meaningful.

Table 1.1
IBM 7090/M6800 Comparisons

	IBM 7090	M6800
Cost	\$3,000,000.00	\$3,000.00
Cycle time	2.18×10^{-6} sec	1.0×10^{-6} sec
Memory	32768 units unit width: 36	65536 units unit width: 8

1.1.1 The Components of a Computing System

Computing systems consist of several components, not just the electronic circuits that are called the *hardware* components of the system. In a typical computing system we'll find a blend of hardware, software, and firmware components. The term *software* is used to refer to a set of programs or instructions that are executed by the hardware. Hardware can be viewed as a very inflexible part of a computing system; i.e., the electronic components are fixed and can be changed only with great difficulty. Software, on the other hand, is quite flexible since it is relatively easy to change one or more instructions. In terms of flexibility, firmware lies somewhere between hardware and software. *Firm-*

ware is a program or set of instructions, but it is normally encoded in a medium that makes it relatively difficult to change. The use of firmware is advantageous for programs that are never (or seldom) altered since firmware media are generally faster and less expensive than typical software media.

In studying computing systems, we must discuss hardware, software, firmware, and the interaction between these components. When viewed as a whole, computing systems are quite complex and seem to defy total comprehension. To overcome the complexity factor, we will investigate these systems in terms of a set of levels, moving to a new level only when we completely understand the preceding levels. The analysis of computing systems in terms of levels has been described in Tanenbaum (1976). We briefly review this approach in the following paragraphs.

1.1.2 Levels of Computing Systems

While computing systems taken in totality are very complex, the basic computer is actually a very simplistic device when viewed at its lowest level. At this level computers are viewed as machines that can only execute (directly in the hardware) instructions encoded in a language consisting entirely of 0's and 1's, i.e., the computer's machine language. Let us say that this language is at level L1.

Even though we can write programs in L1, it soon becomes very tedious and difficult. To overcome this problem, the computing system designer usually develops a new set of instructions that are more convenient for humans to use. These instructions form a new language, e.g., L2. Now the designer must specify a way of running L2 programs on the original machine. There are two ways of attacking this problem—*translation* and *interpretation.*

In the case of *translation,* an L1 program called a *translator* is used to replace each instruction of an L2 program by L1 instructions that perform the same function. Thus the translator accepts an L2 program as input and outputs an equivalent L1 program, which may then be executed on the machine.

Interpretation involves writing an L1 program (an *interpreter*) that will again accept L2 programs as input data. The interpreter decodes each L2 instruction and immediately executes an equivalent set of L1 instructions. Note that an equivalent L1 program is not created, i.e., each L2 instruction is decoded and directly executed. Thus two processes (interpretation and execution of the L2 program) are occurring simultaneously.

Translation and interpretation are methods of moving from one level (L2) to another (L1). Ordinary users are unaware of level transversal; e.g., FORTRAN programmers might feel that they are using a FORTRAN machine, just as the L2 and L1 programmers feel that they are using L2 and L1 machines, respectively. However, the L1 programmer is using a *real machine* while the L2 and FORTRAN programmers are using *virtual machines.* The L1 machine is real because L1 programs are directly executed in the hardware. L2

is a virtual machine since L2 programs must be either translated or interpreted to the L1 level before they are executed.

Early computers had only one level—the conventional machine language (CML) level. Modern computing systems are organized in terms of several levels. The notion of a two-level system was suggested by M. Wilkes in England (Wilkes, 1951). This machine was designed to have a built-in (unchangeable) interpreter, which would execute machine-language programs. In this approach, machine-language instructions were simulated, by the interpreter, with "microinstructions." At present, the micro level is common on most machines. The reader should be warned at this point not to confuse the use of the term micro in this context with the use of micro in microcomputer or microprocessor. The built-in unchangeable interpreter mentioned here is a firmware component of the system, which is used to define the control component of a computer. The instructions found in the interpreter were termed microinstructions because basic CML-level instructions were defined in terms of these lower-level instructions. In practice, the term *microprogramming* refers to the programming of the control function of a system; it does not refer to writing programs for microcomputers. A microcomputer may have a micro level also; i.e., its control function may be defined by a set of microprograms.

In the 1950's software components called *assemblers* and *compilers* were developed, and they added additional levels to the system. An assembler is a translator that supports an *assembly language,* i.e., a language that is very close to the CML-level instructions, but it is encoded in terms of mnemonic instructions and symbolic names. Compilers, which are also translators, support the so-called *high-level languages* (FORTRAN, PL/1, or PASCAL), which contain constructs that are quite dissimilar from the CML level and are designed to make programming much easier.

A final level was added in the 1960's when major advances were made in operating systems. *Operating systems* (OS) are software components that control the use of system resources and provide access to other software or hardware functions.

Figure 1.1 shows the levels available on most modern computing systems, together with the normal methods of level transversal. Note that two arrows emanate from the OS level, one to the CML level and one to the MP level. This is because the OS level actually consists of a mixture of OS-level and CML-level instructions. The OS instructions require interpretation down to the CML level, while the CML instructions can be passed directly to the MP level.

The five-level structure of Fig. 1.1 will be present in the computing systems we will examine in this text. In particular, we will develop a microcomputer system in the context of these levels.

Within the context of the hardware, software, and firmware components of a system, we should note that there are no rules that state at what level a function should be implemented. In some machines, for example, a multiply instruction is directly supported by the hardware while, in others, it must be

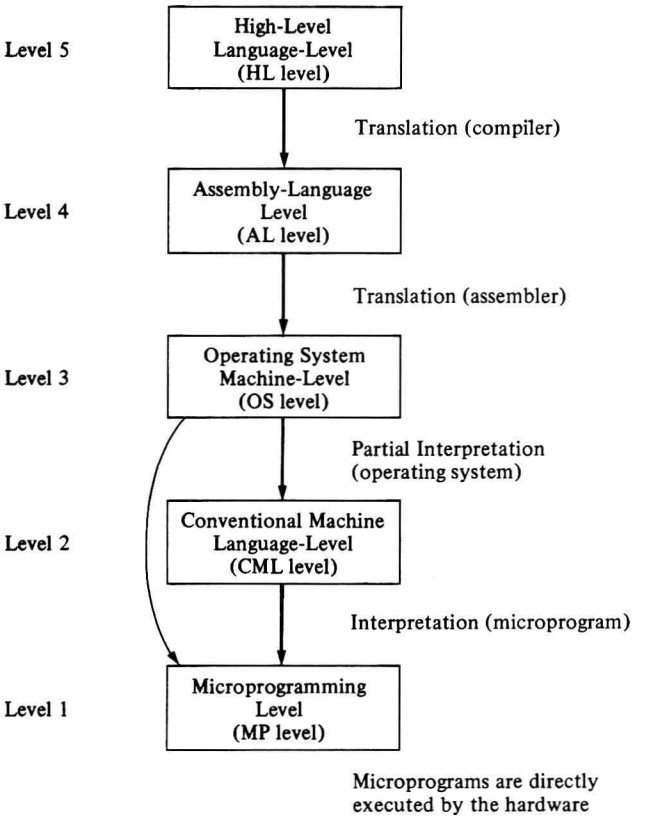


Fig. 1.1 Computing system levels.

simulated in the software. This is true of the more complex functions also. For instance, while we defined translators as software components, this definition is only generally, but not always, true. For example, on the SYMBOL-2R system, the translator is a hardware component (Anderberg, Smith, 1973). Hardware and software are functionally equivalent; i.e., any operation performed by software can be built directly into the hardware, and any instruction executed by the hardware can be simulated in the software. This observation applies to firmware also. In this text we will refer to functions as existing in the form in which they are typically implemented. The decision as to whether a function should be implemented in hardware, software, or firmware is usually dictated by economics rather than feasibility.

We begin our investigation of computer systems at the CML level. Although not the lowest level in modern machines, this is the level described in manufacturers' "Principles of Operation" manuals, and it was historically the lowest level.

1.2 THE BASIC HARDWARE COMPONENTS

A minimal computing-system configuration requires at least the following hardware components: Central processor unit (CPU), Memory, and input/output (I/O) interfaces. These might be arranged as shown in Fig. 1.2. The buses serve as links between the various hardware components and are generally bidirectional. The number of lines in (or width of) these buses varies depending on the particular bus and the system.

The CPU—often called the *microprocessor* (μP) in the case of microcomputers—is the core of any computing system. It includes a *control unit* (CU), an *arithmetic logic unit* (ALU), and various *registers* (Fig. 1.3).

Among the registers, we would typically expect to find the following:

1. *Accumulators* (one or more), which generally hold the results of the various operations being performed on the data. These registers would normally be of the same width (be composed of the same number of bits) as the data bus.

2. A *program counter* (PC), which at any time contains the address of the next instruction to be executed. This would normally be of the same width as the address bus.

3. A *condition code* or flag register, which consists of various flags indicating conditions such as arithmetic overflow or carry, a result "zero," etc.

Finally, we would expect to find other special-purpose registers that could be used to facilitate the accessing of memory. Registers such as index registers and stack pointers would fit into this category.

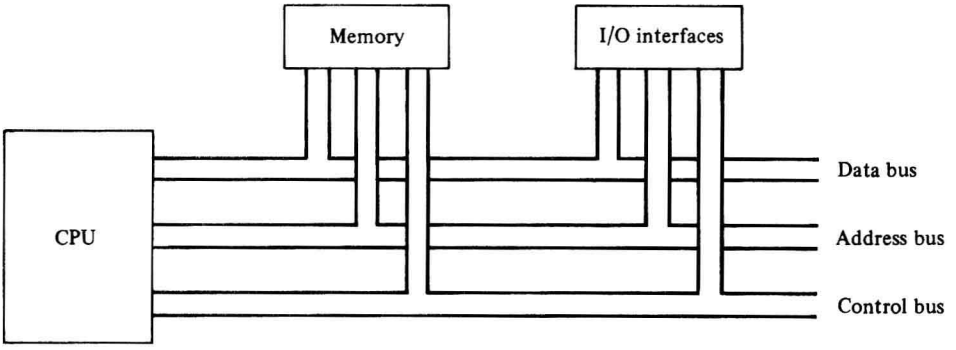


Fig. 1.2 Minimal computing system.

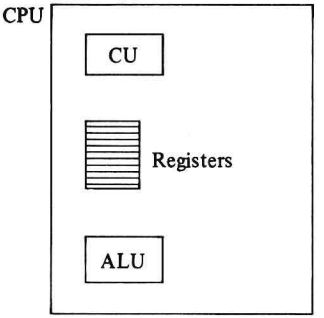


Fig. 1.3 Central processing unit.

In the basic operation of the computer, the following sequence of steps is taken:

1. The CPU fetches a machine-language instruction from the memory location indicated by the PC.
2. The PC is then updated to point to the next instruction.
3. The fetched instruction is decoded and executed by the CU.
4. Go to step (1).

The ALU facilitates those instructions calling for arithmetic or logical operations.

1.2.1 Number Systems

As we shall see later, a digital computer is constructed from two-state binary devices that can be either on or off. We shall refer to these states as 1 and 0, respectively. Such an individual binary device, or more precisely, the information it contains, is referred to as a *bit* (BInary digiT). The registers and memory units mentioned above might be regarded as groupings of these binary devices. Generally however, access to registers would be somewhat easier and faster than access to memory.

Since computers are constructed from binary devices, it is natural to utilize the binary (base 2) number system for the internal representation of numbers. Binary numbers involve only the two digits 0 and 1; and in the binary representation of an integer, each digit is weighted according to its position. Thus the value of the six-bit binary number $b_5b_4b_3b_2b_1b_0$ is

$$b_5 \cdot 2^5 + b_4 \cdot 2^4 + b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0.$$

Note that this is consistent with the usual interpretation of the digits comprising an integer. For example, the decimal (base 10) integer 324 is evaluated as $3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0$.