Computer Hardware and Organization: An Introduction

M. E. SLOAN

Computer Hardware and Organization: An Introduction

M. E. SLOAN
Michigan Technological University

（内部交流）

S R A
SCIENCE RESEARCH ASSOCIATES, INC.
Chicago, Palo Alto, Toronto
Henley-on-Thames, Sydney, Paris, Stuttgart
A Subsidiary of IBM

PREFACE

This text is intended for an introductory course in computer hardware and computer organization. Students are assumed to have taken the usual mathematics courses taught in American schools through at least the first two years of high school. The text is suited for college freshmen or sophomores but, with the optional sections, can be used for juniors and seniors.

At Michigan Technological University, this text is used in a one-term introductory course in computer engineering usually taken by three types of students. Electrical engineering students who intend to specialize in computer engineering take the course to survey areas of computer engineering that do not require a background in electronic circuits or in advanced mathematical techniques. Other electrical engineering and a few other engineering students who are not primarily interested in computer engineering take the course for an understanding of computer hardware. Computer science students also take the course as a computer hardware course. The text is supplemented by a laboratory program. Students design and construct several simple logic circuits and write a few machine language and assembly language programs.

The theme of the text is computer structure. The structure of a computer is examined at several progressive levels. Structures at one level become the components of a higher level. The text begins at the logic level, skipping over the electronic circuit level, a discussion of which would require the student to have a background in physics, circuit analysis, and electronics that is not needed for the rest of the book. Unlike most other introductory computer hardware texts, the text does not begin with number representation. This topic is covered in an appendix where it may serve as a review for students who are familiar with it; many students have learned binary arithmetic in high school or elsewhere. An instructor may choose to begin with this appendix. More advanced aspects of number representation for computers are treated in context as they are needed.

The main chapters are prefaced with an overview that allows the student to prepare a mental framework for the material of the chapter in accordance with the principles of cognitive learning theory. Each chapter ends with a summary, references, and problems. The section of the text needed for each problem is shown in parentheses to facilitate assigning problems. Answers for selected odd-numbered problems are given in appendix B. Since this is an introductory text,

most references are to other texts rather than to the original research papers.

Chapter 1 presents an overview of the structural levels of the computer. The logic level discussions (chapters 2 through 5) cover combinational logic circuit analysis and design; logic technologies and integrated circuit implementations of standard logic functions; flipflops and sequential circuits, concentrating on common computer circuits such as registers and counters; and construction of logic systems from register-transfer modules. Chapter 6 consolidates the material on the logic level by discussing the arithmetic circuits needed for a computer. The programming level is treated in chapters 7 through 8, which discuss the machine language and assembly language programming sublevels. Memories and microprogramming, input/output devices and interrupt handling, and the organization of minicomputers and microprocessors are developed in chapters 9, 10, and 11, respectively—the systems level. The back matter includes tables of instructions and codes for the PDP-8, a review of binary arithmetic, and a glossary.

The discussion in chapters 7 and 8 centers on minicomputers since we think students should have hands-on experience with a computer, and minicomputers are both relatively inexpensive and widely available for student use in most universities. The discussion of programming is made as general as possible, using the PDP-8 (the most widely used minicomputer) as a continuing example. Instruction sets of other popular minicomputers and microcomputers are also discussed. The instructor can readily supplement this material with information about any computer available for class use.

Many people helped in the development of this text. At Michigan Tech Arnold Lee reviewed an early version of the manuscript, and Ted Grzelak taught two classes from a later version and contributed many helpful suggestions. Harold Stone of the University of Massachusetts encouraged me throughout the developmental process. Dale Anderson (Iowa State University), Jay Bayne (California Polytechnic State University), John Keown (Southern Technical Institute, Marietta), Donald P. Leach (Foothill College), Robert J. Smith, II (Lawrence Livermore Laboratory), Michael G. Thomason (University of Tennessee, Knoxville), John Wakerly (Stanford University), and Gerald E. Williams (formerly of Riverside City College) also reviewed the manuscript and contributed many improvements. Also essential to the book's development was the careful typing of Ruth Tepsa and Phyllis Brumm.

CONTENTS

3 Combinational Logic Circuits and Logic Technologies

4 Sequential Logic

5 Register-Transfer Logic

6 Arithmetic Unit

PART II THE PROGRAMMING LEVEL

7 Machine Language Programming

8 Assembly Language Programming

1 Overview

What are you able to build with your blocks?
Castles and palaces, temples and docks.
Robert Louis Stevenson

The theme of this book is the diversity of levels in the structures of digital computers and other digital systems. This chapter defines the levels that we will study and provides an overview of the book.

1.1 INTRODUCTION

Modern digital computers are complex systems. To understand them, we will view them as a hierarchy of system levels, as first described by Bell and Newell (1971). We will consider four basic levels: electronic circuits, logic, programming, and computer systems. (The logic level is further divided into three sublevels: combinational logic, sequential logic, and register-transfer logic.) Since a thorough understanding of electronic circuits requires some advanced knowledge of physics and mathematics, we will concentrate on the other three levels, which are unique to computers and to digital technology.

The levels nest together in a hierarchy. At each level the system may be described in terms of appropriate components, structures, and system behavior. The components at a particular level obey certain physical or mathematical laws and combine to make structures. Together, the components and structures determine the behavior of the system at that level. Each level also has a distinct language or languages to describe components and their connections.

TABLE 1.1

HIERARCHY OF SYSTEM LEVELS WITH EXAMPLES OF STRUCTURES AND COMPONENTS

LEVEL	STRUCTURES	COMPONENTS
Computer systems	Computers, networks	Controls, processors, memories
Programming	Programs	Instructions, subroutines, memories, operators
Logic		
Register-transfer	Circuits—arithmetic unit	Registers, data operators
Sequential	Circuits—counters, registers	Flipflops—*RS, JK, T, D*
Combinational	Circuits—encoders, decoders	Gates—AND, OR, NOT, NOR, NAND
Electronic Circuit	Circuits—gates, multivibrators (flip-flops), amplifiers	Active—transistors, voltage sources Passive—resistors, capacitors, inductors, diodes

Table 1.1 shows the system levels with examples of their components and structures. Note that structures at one level can become components at another level. For example, gates and flipflops are both structures at the electronic circuit level, but gates are components at the combinational logic sublevel and flipflops are components at the sequential logic sublevel.

The purpose of this book is to study each computer level (except the electronic circuit level) briefly, yet in enough detail to show its characteristics and importance to computer systems. Because digital technology is rapidly changing, not all the levels we will study are well established or well understood. While the lowest levels have been organized or codified for several decades, the newer levels—particularly the register-transfer level and the computer systems level—change constantly. New levels may yet emerge. In addition, the levels described here cannot completely describe computer system behavior, since they exclude mechanical devices, such as card readers, teletype terminals, and line printers. We will study such devices briefly in chapter 10 to see how they interact with other structures and to improve our knowledge of computer hardware.

The study of computer systems is similar to the study of human systems. Humans, too, can be studied at many levels, for example the anatomical, the neurological, the biochemical, the psychological, the sociological, and the anthropological, to name just a few. An attempt to classify such a study would yield a hierarchy in which some structures at one level, say the biochemical, become components at another level, say the physiological. In other cases, the hierarchy is not clear. Just as there are careers in human systems that involve primarily one level, so there are careers in computer systems that concentrate on one level, such as programming or logic design. Yet a knowledge of the impact of one level on the entire system is important in order to analyze, design, and use systems successfully.

The rest of this chapter contains brief overviews of system levels. All the ideas introduced here will be discussed in more detail in later chapters. Some of the words and concepts may be unfamiliar to you now but will become familiar as you progress through the book. This chapter is intended to provide a framework for understanding the hierarchy of system levels; it is not intended to introduce any level rigorously.

1.2 ELECTRONIC CIRCUIT LEVEL

The lowest level shown in table 1.1 is the electronic circuit level. (A lower level might combine electromagnetic theory and quantum mechanics to explain the operation of the components at this level.) Its components can be passive (such as resistors, capacitors, inductors, and diodes) or active (such as voltage or current sources and transistors). Circuit behavior is characterized by continuous waveforms of voltage and current, which can be described by differential equations.

We will not consider the electronic circuit level in any detail since it requires a knowledge of circuit analysis and physics beyond that assumed for this book. However, we will consider how the technologies chosen to implement electronic circuit design, for example metal-oxide-semiconductor (MOS) technology, affect other levels of computer operation.

1.3 LOGIC LEVEL

The concept of logic level is unique to digital technology. The preceding electronic circuit level, in contrast, is useful for many technologies. The difference between the two levels is most obvious in circuit

behavior. At the logic level, circuit behavior is described by discrete binary variables that are called 0 and 1, or low and high, regardless of the exact voltages to which they correspond. At the electronic circuit level, circuit behavior is described by continuous waveforms that require analysis by differential equations, a far more complex form of mathematics.

If operation of a logic circuit depends only on the current values of the inputs, it is called *combinational logic.* The mathematics necessary to describe combinational logic is switching algebra. The components perform logical operations such as logical addition (OR) and logical multiplication (AND), which are analogous to conventional addition and multiplication. (We will define these operations precisely in chapter 2.) The components that implement the operations are connected at their terminals in much the same way that electronic components are connected. The signals are considered to be discrete 0s or 1s.

After learning to analyze and design combinational logic circuits with switching algebra, we will look at some simple map and table methods for combinational logic. We will also look at integrated circuits that have several combinational logic circuits on single chips of silicon. We will consider how integrated circuits have modified older methods of combinational logic design.

The *sequential logic* sublevel characterizes logic circuits whose behavior depends in part on the past history of the circuit. In addition to the logic components used at the combinational logic sublevel, the sequential logic sublevel requires memory or delay components, such as flipflops. A flipflop is a small storage unit whose output can be either 0 or 1. Difference equations, the discrete analog of the differential equations used at the electronic circuit level, describe circuit behavior.

We will study two basic types of sequential logic circuits—*registers* and *counters.* Ordinarily, computers and other digital systems operate with a basic unit of information called a *word.* For a given system a word has a standard number of bits, usually ranging from 4 to 64, where each bit can be either 0 or 1. Registers store information and usually hold one word, but they can be shorter or longer as needed. Counters, as their name implies, count computer operations or time intervals and are basic to the timing and control of a computer.

While both the combinational logic sublevel and the sequential logic sublevel have been standardized for nearly as long a time as electronic computers have existed, the last logic sublevel, the register-transfer level, is new and uncertain. Its components are registers

and transfer paths between registers. The behavior of a register-transfer system is represented by the sequence of bit values. Contents of the registers can be combined by logical operations, arithmetic operations, or simple transfers. A set of expressions specify the rules for register transfers. Register-transfer level descriptions of computers are becoming more common, and a number of register-transfer languages have been developed. Unfortunately, notation is not standard. Later we will look at one register-transfer language and at register-transfer hardware.

1.4 PROGRAMMING LEVEL

The programming level may be the most familiar to you since you probably have some experience in programming in a high-level language such as FORTRAN. This level is unique not only to digital technology, but even more narrowly, to digital devices with central processors that interpret a programming language. Some digital devices, such as simple digital voltmeters, do not have central processors; they have a logic level but not a programming level. As we will see in chapter 11, a major current trend is to replace digital systems based solely on logic with digital systems based on microprocessors, or small central processors.

We may regard instructions and subroutines as the basic components of the programming level since they are the components of programs. An alternate view is to consider programming level components to be memories and operations. *Memories* store *data structures* such as student records. *Operations* transform the data structures and produce new data structures. The behavior of the programming level is the sequence of data values. A *program* (that is, a sequence of instructions) prescribes the operations to be performed on given data structures. A control structure specifies the order of execution of instructions, allowing for program branching.

The programming level has a hierarchy of sublevels, as shown in table 1.2. The lowest sublevel, *microprogramming,* involves *microinstructions,* which are components of the basic instructions of a computer. *Machine language* programs are written with the basic instruction set of the computer and can be expressed as the actual bit patterns of 1s and 0s used by the computer hardware. Since bit patterns or numbers are difficult to remember, we often represent machine language instructions with symbols called *mnemonics.* For each instruction both an address and the numerical form of the instruction must be specified. *Assembly language* is shorthand notation for machine language that allows mnemonic instead of numeric

TABLE 1.2

HIERARCHY OF LANGUAGE SUBLEVELS OF PROGRAMMING LEVEL

LANGUAGE SUBLEVEL	EXAMPLE
Compiler languages	FORTRAN, PL/I, COBOL
Interpreter languages	BASIC, APL
Assembly language	Unique to each computer
Machine language	Unique to each computer
Microprogramming languages	Unique to each computer

instructions, and symbolic instead of numeric addresses. Assembly language programs are run with the help of a program, called an *assembler,* that performs many needed housekeeping functions for the programmer.

The three lowest programming sublevels—microprogramming, machine, and assembly language—are unique for each computer. In contrast the higher sublevels—*interpreter* and *compiler* languages—can be used on many computers. These two languages differ in the way they are implemented. Compilers usually translate a source program written in the compiler language, such as FORTRAN, directly into an equivalent machine language program which is executed. Interpreters usually translate and execute source language statements, written in an interpreter language such as BASIC, one at a time.

The programming level differs markedly from the logic level since it is linguistic rather than physical. It allows us to name things, to abbreviate, and to interpret instructions, none of which we could do at lower levels. Because the programming level differs so from other levels, people can become expert programmers with little or no understanding of logic or electronic circuit behavior.

Most of today's computer science focuses on the programming level. This level comprises a hierarchy of language levels and a diversity of languages that rivals the natural languages of the world. No universal programming language yet exists, although some notations for programming languages are becoming standard. This book will be concerned with programming primarily at the machine language and assembly language levels. We will illustrate machine and assembly language programming with programs for the PDP-8. a simple minicomputer.