THE PRINCIPLES OF COMPUTER ORGANIZATION

G. MICHAEL SCHNEIDER

THE PRINCIPLES OF COMPUTER ORGANIZATION

G. MICHAEL SCHNEIDER
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE
MACALESTER COLLEGE

EW YORK CHICHESTER BRISBANE TORONTO SINGAPORE

Production supervised by Ellen C. Baron
Cover and Text design by Karin Gerdes Kincheloe
Cover illustration by Steve Jenkins
Manuscript edited by Brenda Griffing under the supervision of Martha Cooley

Library of Congress Cataloging in Publication Data:

Schneider, G. Michael.
The principles of computer organization.

Includes Index
1. Computer architecture. 2. Assembler language (Computer program language) I. Title.
QA76.9.A73S36 1985 001.64 84-20853
ISBN 0-471-88552-5

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PREFACE

This book is intended as a one-semester text for the course entitled CS3, "An Introduction to Computer Systems," as described in Curriculum '78 of the Association for Computing Machinery (ACM). It also contains material from course CS4, "Introduction to Computer Organization," and it would be appropriate for course CS/IS 3, "Computer Organization and Assembly Language Programming," described in the recent ACM Report on Small College Computer Science Curricula.

A course in computer organization has two quite different and distinct goals:

- To study broad, general concepts related to the structure and organization of all computer systems
- To learn the assembly language and architecture of one specific system.

The main focus of such a course must be the first of these two points: the study of machine-independent concepts in computer organization. Technology is fleeting, and today's state-of-the-art computer system will be tomorrow's antique. During their professional careers, today's students will encounter computers from many different manufacturers, with quite different architectures and instruction sets. Concentrating too early on the unique hardware capabilities and ideosynchrasies of one machine can produce students with "tunnel vision" who feel comfortable with only one approach to computer design and are either uncomfortable with or ignorant of other structures. The ACM Report on Small College Computing Curricula alluded to this problem when describing the course entitled "Computer Organization and Assembly Language Programming." The report warned that "caution must be taken to avoid overemphasizing the particular hardware in use." Similarly, Curriculum '78, in its description of course CS3, stated that " . . . concepts and techniques that apply to a broad range of computers should be emphasized."

In light of this concern, Parts I and II of this text introduce the topics of information representation and computer organization in a machine-independent fashion.

AJ298/08

Part I (Chapters 2–5), "The Representation of Information," discusses approaches for representing the four basic data types that exist on computer systems: unsigned binary, signed integers, characters, and floating point. The discussion includes a survey of a number of possible representational techniques, their theoretical underpinnings, and their advantages and disadvantages from the point of view of efficiency, accuracy, and ease of implementation. It concludes with a discussion of how higher-level data structures found in languages such as FORTRAN, COBOL, and Pascal can be realized in terms of these more elementary hardware data types.

Part II (Chapters 6–9), "The Design of an Idealized Computer," develops in step-by-step fashion the organization and structure of the well known and almost universally used Von Neumann architecture. Each large-scale functional component—memory, ALU, buses, I/O, and processor—is separately introduced along with important general concepts and principles associated with that functional unit. At the conclusion of this section, these separate pieces are combined and integrated into an idealized model of a Von Neumann computer of the type diagrammed in Figure 9-18. The text then introduces a range of different instruction formats and presents some typical instruction sets of the type that students are most likely to encounter. Finally, the overall fetch/execute instruction cycle of this idealized computer is described along with its realization in both hardware and microcode.

By the end of Parts I and II of this text, the student will have a solid grounding in the fundamental principles of information representation and computer structures as they apply to a wide range of different systems. However, computer organization is also an applied discipline, and it is important that students gain some experience in programming a real, rather than an idealized, computer system. This system will preferably have an interesting architecture and will be representative of what is currently available in the marketplace. For this text we have chosen the PDP-11 family of computers manufactured by the Digital Equipment Corporation. The PDP-11 is by far the most widely used minicomputer in the world today, and its architecture has influenced the design of a number of other systems. Its modest yet powerful basic instruction set is relatively easy for a student to learn and master in a one-semester course in assembly language. In addition, the PDP-11 instruction set can be executed by all members of the VAX-11 family of computers, using what is called "compatibility mode." The VAX-11 is one of the most popular of the group of "supermini" computers and is widely available in the computer science departments of colleges and universities. Therefore, by choosing the PDP-11, we have selected a computer that is interesting, important, and available to the widest possible range of students.

Part III (Chapters 10–18), "The Structure and Organization of an Actual Computer System," introduces the student to the PDP-11 and its assembly language, MACRO-11. These chapters cover virtually all MACRO-11 features, including the 90 or so instructions in the basic instruction set, all 12 addressing modes, interrupt-driven I/O (Chapter 15), subroutines and parameters (Chapter 16), macros and conditional assembly (Chapter 17), and the floating point instruction set (Chapter 18). The text contains a number of MACRO-11 programs and program fragments that illustrate the language feature under discussion. We have chosen to treat MACRO-

11 as a systems-oriented language rather than an applications programming language. We feel that this approach is more representative of the actual uses of assembly language and can better exemplify the capabilities of MACRO-11. Therefore, the examples presented in this section are primarily system-related and include such tasks as normalizing floating point numbers, multibuffering input/output, generating correct parity bits, performing extended precision arithmetic, managing a real-time clock, and producing a symbolic dump. In order to simplify input and output, which in assembly language can be quite complex, we have provided (in Appendix E) a set of prewritten MACRO-11 I/O routines that students may use until they are able to write their own I/O code.

Part IV (Chapters 19–20) is entitled "An Introduction to System Software." As everyone in computer science is well aware, an in-depth understanding of computer systems requires a knowledge of both hardware and software and the close relationship of the two. Of course, a thorough treatment of this complex topic is well beyond the scope of a single course in computer organization, and it will be the subject of a number of succeeding courses in computer science. In Chapters 19 and 20 we provide an introduction to some of the most common system software components that students have encountered, including scanners, one and two pass assemblers, linkers, and absolute and relocatable loaders. This material serves as both an introduction to the general concepts of system software and as motivation for material that will be presented in future courses, such as operating systems and compiler design. It also allows us to end the text by summarizing the life cycle of a process on a typical Von Neumann computer—from translation through linking, loading, and execution using the fetch/execute instruction cycle described in Part II. This discussion should clarify the close working relationship of hardware and software as well as demonstrate the steps needed to run computer programs.

There are essentially two different ways to approach a course in computer organization; this text will easily adapt to either one. The instructor who wishes to concentrate more on the general principles of computer organization and somewhat less on the assembly language of a specific system will want to spend more time on Chapters 1 to 9 and 19 to 20, and less time on the MACRO-11 material in Chapters 10 to 18. Because less time will be available to cover the language, the instructor may choose to omit some of the more advanced features of MACRO-11 such as conditional assembly, traps, or floating point instructions.

Conversely, the instructor who wants to concentrate more on assembly language programming may present the material in Chapters 2 through 9 more quickly so that time is available for a thorough presentation of all the features of MACRO-11. This instructor should find Chapters 10 to 18 a complete and self-contained introduction to assembly language programming in MACRO-11. This approach will allow the students to get into assembly language earlier and write more, and more complex, programs. However, we hope that adequate time will still be spent on the general concepts described in Parts I and II in order to give the student a firm grounding in the essential principles of computer organization.

I would like to thank a number of people who helped me in the preparation of this text. The faculty and staff of the Computer Science Department of the Hebrew University of Jerusalem provided me with both office and computer facilities during the writing of the original manuscript. Mr. Mark Wickham painstakingly and accurately coded, ran, and tested all of the MACRO-11 examples presented in the book. My students in the course CS30, "Computer Systems Organization," at Macalester College were the guinea pigs upon whom I tested the ideas contained in this book. A most important thank-you goes to Ms. Sandy Whelan, who edited, typed, and revised the manuscript and whose contribution to the success of this project was invaluable.

Finally, I wish to thank all of the wonderful people with whom I worked at John Wiley: Carol Beasley, Gene Davenport, Richard Bonacci, Lorraine Mellon, and Elaine Rauschal. This is the fourth book that I have completed for Wiley, and I have truly enjoyed every one.

G. Michael Schneider

Minneapolis, Minnesota

CONTENTS

CHAPTER 1

AN INTRODUCTION TO COMPUTER SYSTEMS

1.1 Introduction

In this text we will be studying the design, structure, and internal organization of computer systems. In this respect a computer is quite similar to a number of other large, complex systems, such as an automobile, a guided missile, or a telephone network. It can be studied at many levels of abstraction. People with different backgrounds and goals will examine a system in completely different ways and decompose it into totally different "building blocks."

We can clarify the last point by listing some of the ways that people view one well-known and widely used system—the automobile. To the casual *driver*, an automobile is simply a means of transportation. It can be driven from here to there at a certain speed, at a certain cost, and with a particular level of convenience. Technical questions about what is happening under the hood are irrelevant at this level of abstraction; the only things that matter are how to start and stop it and how to drive it.

From the viewpoint of an *automobile mechanic*, an automobile is a collection of major subsystems (e.g., engine, transmission, electrical, exhaust) that must function and interact properly to provide acceptable service.

The *automobile designer* is interested in the same collection of subsystems, but is also concerned about how to design them and put them together so that the finished product will meet the needs and desires of the user. The designer is concerned with marketplace demands, aesthetic appeal, convenience, reliability, and manufacturing costs.

An *automotive engineer* is concerned not only with these major functional subsystems, but also studies the design of more primitive and basic automotive building blocks such as valves, cams, gears, gaskets, and seals. The parameters of interest at this level may include stress, strain, tolerance, and strength.

Finally, a *chemist* or *metallurgist* considers a car at the lowest level of abstraction—as simply a collection of elements and materials such as iron, steel, aluminum, rubber, and glass.

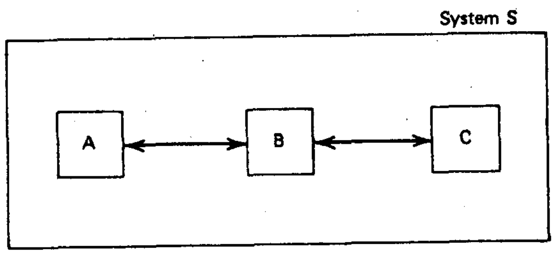
This extended analogy should make you realize that when you study any large complex system, whether it be an automobile or a computer, the "pieces" you see and the approach you choose depend entirely on your purposes in studying the system, and the questions you wish to answer.

The viewpoint from which you decompose and study a system is called an *abstraction level,* and the components you study at that level are typically called the *building blocks* or *primitives* of the abstraction. At any given level of abstraction, the internal workings of the primitives at that level are hidden from view, and you see these building blocks only as indivisible entities that perform a given function. Likewise, any relationships that exist between these inner subcomponents are invisible, and the building blocks are viewed solely as "black boxes."

For example, in Figure 1-1a, you would study system S by studying its three components, A, B, and C, and the interactions between them, without concern for how A, B, and C themselves are constructed. If the inner workings of components A, B, and C were truly important to an understanding of system S, this level of abstraction would be inappropriate and you would have to choose a different abstraction level, such as the one shown in Figure 1-1b on the next page. Now you can view system S as being composed of components A, B, and C, as well as subcomponents A1, A2, B1, B2, B3, C1, and their interrelationships. You have gained some additional information about the internal construction of system S. However, you may also become enmeshed in a great deal of low-level detail that could inhibit your understanding of the operation of the overall system. Whether Figure 1–1a or 1–1b is the better way to view system S depends on what questions you wish to answer and what relationships you want to study.

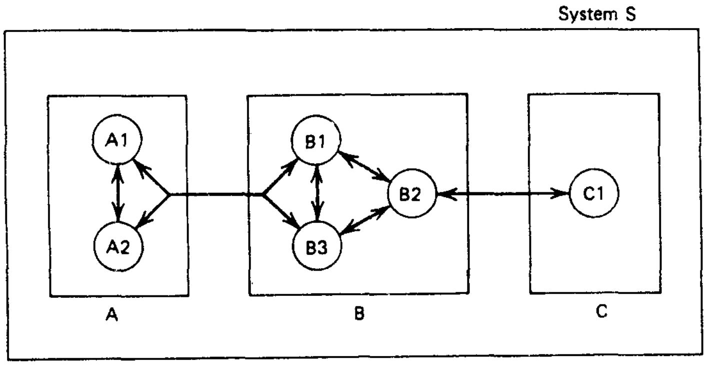
In the next section, we show specifically how this generalized discussion of abstraction levels applies to the study of computers and computer systems. This allows us

FIGURE 1-1a Abstraction Level 1.



(a)

FIGURE 1-1b Abstraction Level 2.



(b)

to explain in detail the material that is presented in the remainder of the text, and to explain the viewpoint from which we will be examining computer systems. This viewpoint will be quite different from the framework you may now be using.

1.2 The Hierarchy of Abstractions

There is no single, uniform way to classify the different ways of studying a computer system. Similarly, the terminology associated with these differing viewpoints is not totally standardized. Figure 1-2 presents a quite common and very popular set of abstractions of a computer system.

The very highest level of abstraction (level 1) is typically called the *operating systems level* or, more simply, the *systems level.* At this level of abstraction, you view a computer (or possibly a network of computers) as a "black box" that solves problems, much as the casual driver views a car merely as a means of transportation, without concern for the technical aspects of automotive engineering. At this level, you simply ask the computer to execute tasks, such as statistical packages, text processing, or graphic displays. You communicate with the computer through a high-level command language, which may be part of a program called the operating system or some other software system. This program interprets your request and either performs the desired task or schedules another program to execute it. Your interaction with the computer system is limited to entering these commands, providing data, and viewing the results. Typical primitives at this level include:

Log on/log off procedures
Account numbers, passwords, and billing procedures