STRUCTURED COMPUTER ORGANIZATION

ANDREW S. TANENBAUM

Prentice-Hall Series in Automatic Computation

STRUCTURED COMPUTER ORGANIZATION

ANDREW S. TANENBAUM

Vrije Universiteit
Amsterdam, The Netherlands

PRENTICE-HALL, INC.

ENGLEWOOD CLIFFS, NEW JERSEY

Library of Congress Cataloging in Publication Data

TANENBAUM, ANDREW S. (date)
Structured computer organization.

Bibliography
1. Electronic digital computers—Programming.
I. Title.
QA76.6.T38 001.6′42 74-30322
ISBN 0-13-854505-7

10 9 8 7 6

Printed in the United States of America

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA, PTY. LTD., *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA (PTE.) LTD., *Singapore*

STRUCTURED
COMPUTER ORGANIZATION

Prentice-Hall
Series in Automatic Computation

AHO, ed., *Currents in the Theory of Computing*
AHO AND ULLMAN, *The Theory of Parsing, Translation, and Compiling,*
Volume I: *Parsing;* Volume II: *Compiling*
ANDREE, *Computer Programming: Techniques, Analysis, and Mathematics*
ANSELONE, *Collectively Compact Operator Approximation Theory and Applications to Integral Equations*
BATES AND DOUGLAS, *Programming Language/One*, 2nd ed.
BLUMENTHAL, *Management Information Systems*
BRENT, *Algorithms for Minimization without Derivatives*
BRINCH HANSEN, *Operating System Principles*
COFFMAN AND DENNING, *Operating Systems Theory*
CRESS, et al., *FORTRAN IV with WATFOR and WATFIV*
DAHLQUIST, BJÖRCK, AND ANDERSON, *Numerical Methods*
DANIEL, *The Approximate Minimization of Functionals*
DEO, *Graph Theory with Applications to Engineering and Computer Science*
DESMONDE, *Computers and Their Uses*, 2nd ed.
DRUMMOND, *Evaluation and Measurement Techniques for Digital Computer Systems*
ECKHOUSE, *Minicomputer Systems: Organization and Programming (PDP-11)*
FIKE, *Computer Evaluation of Mathematical Functions*
FIKE, *PL/1 for Scientific Programmers*
FORSYTHE AND MOLER, *Computer Solution of Linear Algebraic Systems*
GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*
GORDON, *System Simulation*
GRISWOLD, *String and List Processing in SNOBOL4: Techniques and Applications*
HANSEN, *A Table of Series and Products*
HARTMANIS AND STEARNS, *Algebraic Structure Theory of Sequential Machines*
JACOBY, et al., *Iterative Methods for Nonlinear Optimization Problems*
JOHNSON, *System Structure in Data, Programs, and Computers*
KIVIAT, et al., *The SIMSCRIPT II Programming Language*
LAWSON AND HANSON, *Solving Least Squares Problems*
LORIN, *Parallelism in Hardware and Software: Real and Apparent Concurrency*
LOUDEN AND LEDIN, *Programming the IBM 1130*, 2nd ed.
MARTIN, *Computer Data-Base Organization*
MARTIN, *Design of Man-Computer Dialogues*
MARTIN, *Design of Real-Time Computer Systems*
MARTIN, *Future Developments in Telecommunications*
MARTIN, *Programming Real-Time Computing Systems*
MARTIN, *Security, Accuracy, and Privacy in Computer Systems*
MARTIN, *Systems Analysis for Data Transmission*
MARTIN, *Telecommunications and the Computer*

MARTIN, *Teleprocessing Network Organization*
MARTIN AND NORMAN, *The Computerized Society*
MCKEEMAN, et al., *A Compiler Generator*
MEYERS, *Time-Sharing Computation in the Social Sciences*
MINSKY, *Computation: Finite and Infinite Machines*
NIEVERGELT, et al., *Computer Approaches to Mathematical Problems*
PLANE AND MCMILLAN, *Discrete Optimization: Integer Programming and Network Analysis for Management Decisions*
POLIVKA AND PAKIN, *APL: The Language and Its Usage*
PRITSKER AND KIVIAT, *Simulation with GASP II: A FORTRAN-based Simulation Language*
PYLYSHYN, ed., *Perspectives on the Computer Revolution*
RICH, *Internal Sorting Methods Illustrated with PL/1 Programs*
SACKMAN AND CITRENBAUM, eds., *On-Line Planning: Towards Creative Problem-Solving*
SALTON, ed., *The SMART Retrieval System: Experiments in Automatic Document Processing*
SAMMET, *Programming Languages: History and Fundamentals*
SCHAEFER, *A Mathematical Theory of Global Program Optimization*
SCHULTZ, *Spline Analysis*
SCHWARZ, et al., *Numerical Analysis of Symmetric Matrices*
SHAH, *Engineering Simulation Using Small Scientific Computers*
SHAW, *The Logical Design of Operating Systems*
SHERMAN, *Techniques in Computer Programming*
SIMON AND SIKLOSSY, eds., *Representation and Meaning: Experiments with Information Processing Systems*
STERBENZ, *Floating-Point Computation*
STOUTEMYER, *PL/1 Programming for Engineering and Science*
STRANG AND FIX, *An Analysis of the Finite Element Method*
STROUD, *Approximate Calculation of Multiple Integrals*
TANENBAUM, *Structured Computer Organization*
TAVISS, ed., *The Computer Impact*
UHR, *Pattern Recognition, Learning, and Thought: Computer-Programmed Models of Higher Mental Processes*
VAN TASSEL, *Computer Security Management*
VARGA, *Matrix Iterative Analysis*
WAITE, *Implementing Software for Non-Numeric Application*
WILKINSON, *Rounding Errors in Algebraic Processes*
WIRTH, *Systematic Programming: An Introduction*
YEH, ed., *Applied Computation Theory: Analysis, Design, Modeling*

To Suzanne, Pluis, and Koe

PREFACE

Once upon a time computers were very simple. These early machines executed a small number of elementary instructions, and the user wrote his programs directly using these primitive instructions. Those days are long gone. A modern computer is a far more complicated entity, often consisting of a half dozen or more distinct levels at which it can be programmed and studied. In fact, nowadays it is often difficult to tell where the "machine" ends and where the software begins.

Many universities have a course in assembly language programming and computer organization early in the curriculum. There was a time when teaching the students how to program one specific computer in assembly language was considered adequate. Those days are also long gone. These courses must keep pace with a rapidly changing subject, and must now provide an introduction to a wide range of topics related to computer organization, many of which were barely known outside the research laboratories only a few years ago. Segmented virtual memories, parallel processing, race conditions, microprogramming, variable architecture machines, store and forward networks, self-virtualizing machines, and cache memories are just a few examples of these varied topics.

This book is intended as a textbook for such an introductory course on assembly language programming and computer organization. The only prerequisite is an introductory course in computer science (or equivalent practical experience) including a little knowledge of programming in a high-level language such as FORTRAN, COBOL, or PL/I. No mathematical or engineering background is needed, making the book suitable for use even at the sophomore level. Nearly all of the material of the B2 course in the ACM Curriculum 68 is covered, as well as similar material that has become important since that proposal was written. The book can also be used for the proposed COSINE course on "Machine Structure and Machine Language Programming." The chapters are self-contained, allowing them to be used as references on specific subjects.

The structured organization of computers, as a hierarchy of levels, is the theme of this book. At the bottom level is the true hardware, whose function is to execute interpreters called microprograms. The language interpreted by the microprograms is the one most people think of as the "machine language" and is the one described in the manufacturer's machine reference manual. Thus, this "machine language" is already one level removed from the language directly executed by the electronic circuits. We call the lowest level "the microprogramming level" (Chapter 4) and the level it supports "the conventional machine level" (Chapter 3).

Most computers have an operating system that runs on the conventional machine level. The operating system provides its users with an "extended" or "virtual" machine that has instructions and facilities not present at the conventional machine level. These include file manipulation instructions, instructions for parallel processing (multitasking) and virtual memory. The set of features and instructions available to the user of the operating system can be regarded as defining a new level, "the operating system machine level" (Chapter 5).

The fourth level is the symbolic assembly language level. Unlike levels 2 and 3, which are supported by interpretation, this level is implemented by a program called an assembler (Chapter 6), which translates level 4 programs to one of the levels below it. Still higher levels, not covered in this book, are the levels defined by problem-oriented languages.

Chapter 7 examines the construction and applications of multilevel computers as a whole, in contrast to Chapters 3–6, each of which deals only with a single level. Chapter 8 is a guide to further study.

The IBM 370, CDC Cyber 70, and DEC PDP-11 computers are used as running examples throughout the text. These machines are used for illustrative purposes only, and no prior experience with any of them is expected.

There are a few algorithms written in a simple subset of PL/I, but these should be immediately understandable to anyone familiar with FORTRAN or ALGOL. PL/I was chosen as a compromise, since it is both widely used and suitable for structured programming. My own first choice was ALGOL 68, but few sophomores know it, unfortunately.

Many people have made contributions to this book. At times I have felt more like an editor than an author. Two people stand out above the others. Jack Alanen read the first handwritten draft in its entirety, and kept many bad ideas from even getting as far as the typewriter. He also made numerous suggestions for improving both the content and presentation. I especially thank Jack for teaching me a lot about teaching. Kim Gostelow went over a later version with a fine-tooth comb, commenting extensively, correcting errors, and trying to turn my prose into something resembling English. Nearly every single paragraph in the entire manuscript was marked with some suggestion for improvement. I am deeply indebted to both of them.

Reind van de Riet and Mitchell Tanenbaum also read and criticized the complete manuscript, providing me with different points of view. John W. Carr III, Arnie Falick, Dick Grune, Jim van Keulen, Bob Rosin, Carel Stillebroer and Wayne Wilner each offered suggestions and help with specific sections. I would also like to thank my students, especially Arie de Bruin, Wim Harmsen, Ad König, Sape Mullender, Johan

Stevenson, and Hans van Vliet for providing feedback. Mrs. Homburg-Knieper did a fine job of typing several versions of the manuscript.

I would like to express my appreciation to the International Business Machines Corporation, the Control Data Corporation, the Digital Equipment Corporation and the Burroughs Corporation for permission to adapt material from their copyrighted publications as follows: IBM—"IBM System/370 Principles of Operation Manual" and engineering drawings for the IBM 3125 CPU; CDC—"Control Data Cyber 70 Models 72, 73, 74 Computer Systems Reference Manual"; DEC—"PDP-11/45 Processor Handbook" and PDP-11/40 engineering drawings; Burroughs—"B1700 Systems Reference Manual." Any errors in the descriptions of these computers are my responsibility.

Finally, I want to thank Suzanne for encouragement, support, help, and especially patience throughout my long period of self-imposed exile from the human race during the preparation of this book. I also would like to thank her for breaking up the long hours with an *uitgeperste sinaasappel.*

ANDREW S. TANENBAUM

CONTENTS

3 THE CONVENTIONAL MACHINE LEVEL

1 INTRODUCTION

A digital computer is a machine that can solve problems for people by carrying out instructions given to it. A sequence of instructions describing how to perform a certain task is called a program. The electronic circuits of each computer can recognize and directly execute a limited set of simple instructions into which all its programs must be converted before they can be executed. These basic instructions are rarely much more complicated than

add two numbers

check a number to see if it is zero

move a piece of data from one part of the computer's memory to another

Together, a computer's primitive instructions form a language in which it is possible for people to communicate with the computer. Such a language is called a **machine language.**

The people designing a new computer must decide what instructions to include in its machine language. Usually they try to make the primitive instructions as simple as possible, consistent with the computer's intended use and performance requirements, in order to reduce the complexity and cost of the electronics needed. Because most machine languages are so simple, it is difficult and tedious for people to use them.

There are two principal ways to attack this problem; both involve designing a new set of instructions that is more convenient for people to use than the set of built-in machine instructions. Taken together, these new instructions also form a