COMPUTER ARCHITECTURE: A MODERN SYNTHESIS

VOLUME 1: FOUNDATIONS

COMPUTER ARCHITECTURE: A MODERN SYNTHESIS

VOLUME 1: FOUNDATIONS

Subrata Dasgupta
Edmiston Professor of Computer Science
University of Southwestern Louisiana
Lafayette, Louisiana

John Wiley & Sons, Inc.
New York Chichester Brisbane Toronto Singapore

Library of Congress Cataloging-in-Publication Data:

Dasgupta, Subrata.
Computer architecture : a modern synthesis / Subrata Dasgupta.
p. cm.
References: v. 1, p. 331-366
Includes index.
Contents: v. 1. Foundations.
ISBN 0-471-82310-4 (v. 1)
1. Computer architecture. I. Title.
QA76.9.A73D35 1989 87-37188
004.2'2—dc19 CIP

Printed in the United States of America

10 9 8 7 6 5 4 3 2

PREFACE

The casual reader turning the pages of a comprehensive text or survey article on computers will be struck by the *diversity* of computers that have emerged over the past four decades.

This dive.sity manifests itself across several dimensions. One of these is the *technological* dimension—that is, the physical basis for their construction and implementation. Most strikingly, however, other sources of computer diversity lie in their *functional* characteristics—their externally observable behavior, properties, and capabilities—and in their *internal structure and organization.* Collectively, these essentially abstract properties define what has come to be known as *computer architecture* and constitute the architectural dimensions of computer diversity.

Thus, one view of computer architecture sees it as the study of such abstract characteristics of computers and their interrelationships that have led to the "evolution" and proliferation of computer species. The analogy between computer architecture and the anatomical and evolutionary studies in biology is, in some sense, evident.

Unlike organisms, however, computers are designed and built entities; they are *artifacts.* Thus, from another viewpoint computer architecture is seen as a *design discipline* concerned with the design, development, description, and verification of computer *architectures.* Here the appropriate analogy is perhaps with *building architecture.* Indeed, in the latter context we use the term "architecture" both in the declarative sense as defining some set of abstract properties, plan, or theme that a building exhibits as well as in the procedural sense of a design discipline concerned with the process of producing these abstract properties.

The analogy between computer architecture and building architecture is really striking in this respect: they are both concerned with abstract properties that ultimately depend on appropriate technologies for their effective implementation; yet each has its own autonomous vocabulary, heuristic rules, and scientific principles—constituting a universe of discourse—that are distinct from the vocabulary, rules and principles of its underlying technology; the efficacy of both architectural disciplines and principles are ultimately tested in social (i.e., human) environments; and finally, they are both, again as disciplines, preoccupied with the problem of organization.

However, this text and its companion volume are not about these analogies, fascinating though they are. In these two volumes I have attempted to deal simultaneously with the two faces of computer architecture—its compendium of architectural *principles* and the issues related to architectural *design.* The first is concerned with parts or subsystems viewed in convenient isolation. The

AJS386/01

second with the design of the whole. The "synthesis" in the title refers, first, to the unification of these two facets and, second, to my attempt to organize the diversity of architectural principles within a single integrated framework.

The two texts are intended to form a coherent whole. However, they have been written and organized so that each volume can be read, studied, and used independently of the other. The present volume, *Foundations,* is intended for senior undergraduate students of computer science and engineering and consists of a set of topics that can be covered in a one-semester senior undergraduate course in computer architecture. In contrast, the companion volume, *Advanced Topics,* is suitable as a graduate-level text. There are, however, two aspects common to the two books: shared References and an identical first chapter, which establishes a common framework and terminology for both *Foundations* and *Advanced Topics.*

Volume 1 is organized into two parts. Part I deals with issues common to all computer architecture. In particular, Chapter 1 introduces the important idea of *architectural levels*—a notion that pervades the entire book—and establishes the connections between architecture on the one hand and design methodology, compilers, microprogramming, software technology, and implementation technology on the other. Chapters 2 and 3 discuss, respectively and in more detail, the issues of implementation technology and design methodology as they relate to architecture.

Part II discusses various facets of the architecture of uniprocessors. At its heart is the register machine, which is discussed at different abstraction levels in Chapters 4 and 5. Since the register machine is historically tied to the emergence and development of microprogramming, a substantial part of Chapter 5 is devoted to this topic.

Chapters 6 and 7 present two contrasting approaches to the development of "language directed architectures"—alternatives to the register machine style. The first is the (now almost "classical") stack machine approach; the second is the very recently developed idea of reduced instruction set computers, much in vogue at this time.

Finally, Chapter 8, discusses various fundamental and important issues related to the memory aspects of architecture.

In concluding, I would like to note an important feature of the problem sets at the end of each chapter: and that is (in keeping with the view of computer architecture as a design discipline), the emphasis on *design problems.* Some of these are suitable for solution as normal "assignments" whereas others are more in the nature of "projects" that can be pursued in the course of an entire semester.

Lafayette, Louisiana
July 31, 1987

Subrata Dasgupta

ACKNOWLEDGMENTS

A project of this kind can only be conducted in an atmosphere in which one's research, teaching, and writing can be carried out in symbiotic harmony and where one is relatively free of administrative duties. I have been fortunate in enjoying, for several years, precisely such a climate at the Center for Advanced Computer Studies of the University of Southwestern Louisiana. Its Director, Terry Walker, has had much to do with this pleasant state of affairs. I thank him for the environment and his support.

I also must thank the National Science Foundation for supporting my recent studies of computer architecture as a design discipline. The fruits of these studies have advertently and inadvertently influenced several parts of this book. These same researches formed the basis of continuing discussions of many aspects of architecture with my students (and collaborators), notably Philip Wilsey, Ulises Aguero, Alan Hooton, Cy Ardoin, and Sukhesh Patel. My ideas were also formed under the influence of an ongoing transatlantic dialogue (stretching over 6 years) with Werner Damm formerly of Aachen Technical University and now with the University of Oldenburg, West Germany. I thank them all for the pleasure of their intellectual company.

A part of this work was done while I was a visiting fellow at Wolfson College, Oxford, and the Oxford University Computing Laboratory in the summer of 1986. I am very grateful to Professor C. A. R. Hoare and to Sir Raymond Hoffenberg, President of Wolfson College, for providing me the facilities and an enchanting physical environment in which to work.

Several persons were kind enough to read and review selected chapters from the two volumes. I thank, in particular, Laxmi Bhuyan, Dipak Ghosal, John Gurd, Steven Landry and Robert Mueller for their many helpful comments. I also thank William M. Lively and James L. Beug who provided very useful reviews of the manuscripts. For any residual errors that may be found in the text, may I say, in the time-honored tradition: *mea culpa!*

In the course of this work I received enormous logistical help from several persons. In particular:

Cathy Pomier, who typed the manuscripts and undertook their many revisions with invariable and sustained good humor. For her, the adjective "unflappable" is truly apt.

Nancy Pellegran, who typed the problem sets and the index and helped put the finishing touches to the manuscript.

Philip Wilsey and Ulises Aguero who exercised their computational ingenuity in producing the computer generated diagrams that appear in the text, and who assisted me in innumerable ways.

Richard Bonacci, my original editor at Wiley, who warmly supported this project.

Gene Davenport, Senior Editor at Wiley, who provided firm, wise, and invaluable advice during the later, crucial stages of my writing.

Joe Dougherty, Editor, Gilda Stahl, Senior Copy Editor, and Dawn Reitz, Senior Production Supervisor at Wiley, for their assistance during the physical production of this book.

I am grateful to each one of these individuals for their help.

My thanks to Robert Mueller, Werner Damm, Gert Dohmen, Philip Wilsey, the Institute of Electrical and Electronic Engineers, Intermetrics, Inc., the Association for Computing Machinery, Digital Equipment Corporation, John Wiley & Sons, Academic Press, McGraw-Hill, and MIT Press for granting me permission to reproduce diagrams and excerpts from their publications. I also thank George Spix and Linda Turpin of Cray Research for providing information on some recent Cray systems.

Finally, a note of gratitude to my wife, Sarmistha, and sons, Jaideep and Monish, for living patiently with this, seemingly interminable, project and for their love and support.

CONTENTS

PART ONE

INTRODUCTION AND BACKGROUND

CHAPTER 1

THE SCOPE OF COMPUTER ARCHITECTURE

In this book the term *computer architecture* will be used in two complementary ways. It will refer to certain *logical* and *abstract properties* of computers, the nature of which will be described herein. The term will also be used to denote the art, craft, and science—or more generally, the discipline—involved in *designing* these same logical and abstract properties. Thus, computer architecture (or more simply, when there is no room for ambiguity, architecture) refers both to certain characteristics of computers and to the design methods used in realizing these characteristics.

1.1 EXO-ARCHITECTURE

What are these logical and abstract properties that are of interest to the computer architect? There are first the *functional* characteristics of computers: their externally observable behavior, properties, and capabilities that are of fundamental interest to a certain group of users. These users include, in particular, system programmers responsible for the construction of operating systems and compilers for a given computer and the applications programmers involved in writing programs in the computer's assembly language.

The collection of externally observable behavior, properties, and capabilities goes by several names in the architectural literature, including, simply, computer *architecture* (Myers, 1982), the *instruction set processor* (ISP) level (Siewiorek, Bell, and Newell, 1982), the *conventional machine level* (Tanenbaum, 1984), and *exo-architecture* (Dasgupta, 1984). I will employ this last term in this book to remind you that these properties reflect the *external* functional and logical features of computers.

The primary components of a computer's exo-architecture are

1. The organization of programmable storage.
2. Data types and data structures, their encoding and representation.
3. Instruction formats.
4. The instruction (or operation code) set.
5. The modes of addressing and accessing data items and instructions.
6. Exception conditions.

A computer's exo-architecture represents a particular abstraction level at which we may choose to view it. An *abstraction* is a simplified or selective description of a system that highlights some of the system properties while suppressing others. In the case of complex systems, we may need to perform *different kinds* of abstraction depending on the purpose at hand. Furthermore, these different kinds of abstractions may be so selected as to form a hierarchic relationship with one another. In that case, we talk of the existence of different *abstraction levels.*

The abstraction level of a computer that we call exo-architecture defines the interface between the physical machine and any software that may be superimposed on it (Fig. 1.1). Indeed, the establishment of such a user interface is the *purpose* of this abstraction level. The "users" of this interface are the operating system and compiler writers and, generally, those who wish to program in assembly language.

It is important to note that abstractions and abstraction levels are *artifacts.* We invent them so that we have a means for organizing and understanding complex phenomena, but there is nothing sacrosanct about them. Thus, two different designers of an exo-architecture may choose and define two very distinct sets of functional capabilities, depending on what they consider to be useful for the "user."

Example 1.1

For most conventional single-processor systems the exo-architecture will consist of the features cited earlier; namely, the instruction set, operand addressing modes, the word length, the number of words (or bytes) of available main memory, the number and types of high-speed programmable registers, and so on. The user may never need to know such "internal" details as the precise mechanisms by which instructions are interpreted by the hardware or whether, for example, instructions are "pipelined."

In contrast, the user of a *vector processor* may well have to know some details of its internal processor organization in order to effectively exploit the potential parallelism that such processors offer. The exo-architecture of these machines may then be defined to reveal such details rather than to hide them as in conventional processors. ■

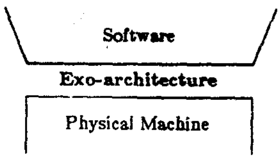

FIGURE 1.1 Exo-architecture: The interface between software and physical machine.

1.2 ENDO-ARCHITECTURE

An exo-architecture is realized by mechanisms implemented in hardware and microcode (or firmware). We can, in fact, describe these mechanisms and their interactions at various levels—for example, the circuit, logic (gate), or register transfer levels. However, important as these levels are, for many purposes they are too detailed—they contain too much information. To understand how the hardware/firmware complex realizes an exo-architecture may require us to abstract from the details of logic or even register transfer levels. This abstraction of the hardware/firmware details has been given several names in the literature, including *processor architecture* (Myers, 1982), *computer organization* (Hayes, 1978), and *endo-architecture* (Dasgupta, 1984). I will use this last term in this book to emphasize that these characteristics constitute a description of a computer's *internal* organization.

Basically, a computer's endo-architecture consists of the following descriptions.

1. The capabilities and performance characteristics of its principal functional components.
2. The ways in which these components are interconnected.
3. The nature of information flow between components.
4. The logic and means by which such information flow is controlled.

It is important to realize that the purpose of this abstraction level is really to aid *understandability*. This abstraction is necessary not only for the "reader" of the design but also for the designer so that he or she need not have to manage and master too many "low-level" details.

The relationship between exo-architecture, endo-architecture, and the next lower (e.g., register-transfer) level representation of the circuits that interpret and realize these architectural levels is illustrated in Figure 1.2.

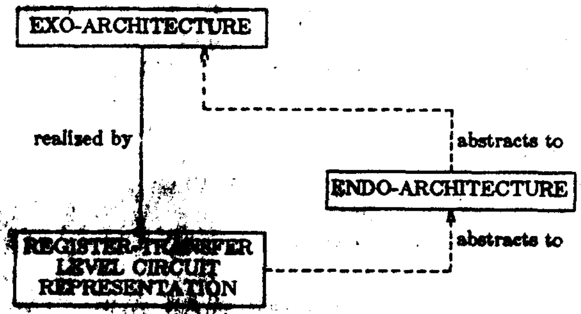

FIGURE 1.2 The relationships between exo-architecture, endo-architecture, and the register-transfer level.

1.3 MICRO-ARCHITECTURE

As described in the foregoing sections, a processor's endo-architecture is an abstracted view of its internal hardware organization. However, architects in practice may exercise considerable freedom in deciding how detailed the endo-architectural design and description should be. A very special situation arises in the cases of *microprogrammed* and *user microprogrammable* computers because, for these machine classes, the architect specifies the endo-architecture at a level of detail necessary for the microprogrammer to write and implement the microcode for such machines.

I will reserve the term *micro-architecture* to denote the internal architecture —the logical structure and functional capabilities—of a computer as seen by the microprogrammer.

Remarks

Several points about micro-architecture are worth noting.

1. The *purpose* of micro-architecture as a distinct abstraction level is to establish and define *the interface between the hardware* and the *superimposed* firmware (microcode). Thus, micro-architecture is to the microprogrammer what exo-architecture is to the (assembly language) programmer.
2. Extending this parallel, and given the recent trend toward the use of high-level microprogramming languages (HLMLs) and their compilers, the micro-architecture of a processor defines those aspects of the hardware system required either by the microprogrammer or by the HLML compiler writer.
3. Although a micro-architecture may be viewed as a special version of a machine's endo-architecture, it is important to keep in mind that the latter may be defined independent of (a) whether microprogramming or hardwired logic is used to implement the processor or (b) the precise style, logic, and organization of the control unit. In other words, a given computer may be designed and described meaningfully in terms of its exo-architecture, its micro-architecture on which the microprogram is run so as to realize the exo-architecture, and an endo-architecture that is an abstraction of the micro-architecture/microprogram complex. The relationship between these levels is shown in Figure 1.3.
4. The micro-architecture of a processor, depending on how detailed it is, may or may not coincide with the *register-transfer level* description. At the latter level, computer structures are described in terms of such primitives as terminals, registers, delays, counters, clocks, memories, and combinational circuits. The primitives from which such a description is composed bear obvious one-to-one correspondences with common medium-scale integration (MSI) logic circuits. Generally speaking, the register-transfer level description will contain more information than the microprogrammer needs to know, hence the micro-architecture abstracts somewhat from this level (Fig. 1.3).