THE DESIGN AND DESCRIPTION OF COMPUTER ARCHITECTURES

SUBRATA DASGUPTA

THE DESIGN AND DESCRIPTION OF COMPUTER ARCHITECTURES

SUBRATA DASGUPTA
University of Southwestern Louisiana
Lafayette, Louisiana

A Wiley-Interscience Publication
JOHN WILEY & SONS
New York Chichester Brisbane Toronto Singapore

UNIX is a trademark of Bell Laboratories.

ADA is a registered trademark of the U.S. Government ADA Joint Program Office.

Library of Congress Cataloging in Publication Data:

Dasgupta, Subrata.
The design and description of computer architectures.

"A Wiley-Interscience publication."
Bibliography: p.
Includes index.
1. Computer architecture. 2. System design.
I. Title.

QA76.9.A73D36 1984 621.3819'58 83-21826
ISBN 0-471-89616-0

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

THE DESIGN AND DESCRIPTION OF COMPUTER ARCHITECTURES

TO

MY PARENTS

AND

MY PARENTS-IN-LAW

PREFACE

The design of computer architectures, an aspect of the broader discipline of computer design, has traditionally suffered from two major drawbacks. First, the idea of storing the architectural design in some formal representation has remained largely outside the mainstream of architectural thought and practice; second, in the absence of a theoretical or formal framework, architectural designs have conventionally been evaluated with respect to both logical correctness and performance only after they have been implemented as physical systems.

These are major shortcomings for any field that lays some claim to being called a discipline. Their seriousness will be evident at one time or another to all those involved with computer architecture, whether as a designer, teacher, or theorist. For their consequences are a compendium of undesirable characteristics that are commonly encountered by all sectors of the architectural community:

1. An inability to communicate the design in a precise, unambiguous way.
2. The invariable presence in the design of features that are undefined and hence subject to erroneous interpretation.
3. An inability to understand the architecture, both from a systems point of view and at the level of minute detail.
4. A low level of confidence in the logical correctness and reliability of the design.
5. A basic inability to manipulate the design either for the purpose of exploring alternative choices for some part of the architecture or with the intent of extending or modifying an existing design.

Indeed, some or all items from this list will appear familiar to all those who are involved in the design of complex systems, such as the software de-

signer, urban planner, or (civil/structural) architect. In the area of software design, many of these characteristics were symptomatic of a fundamental weakness in our understanding of the nature and complexity of software in the late 1960s. This weakness prompted the development of a series of techniques, concepts, and ideas, largely pioneered by E. W. Dijkstra, C. A. R. Hoare, N. Wirth, and others, that in recent times have nucleated into a discipline dubbed "software engineering." In urban and environmental planning and architecture, the response to this problem of unmastered complexity is seen in the form of a plea for rational design, even a mathematization of design, notably exemplified by Christopher Alexander's influential monograph *Notes on the Synthesis of Form* (1964). More generally and in some ways more interestingly, with the recurring observation of the same problem of design in diverse fields, there has emerged from the 1960s onward a vigorous group of design theorists whose focus is the design process itself. For computer scientists, the pioneering and still most illuminating work in this respect must remain Herbert Simon's beautiful little book *The Sciences of the Artificial* (1981).

Computer architecture does not yet, apparently, face the kind of crisis that confronts software design or urban planning. Thus, the attempt to develop a theory or methodology of computer architecture design is less motivated by any perceived crises in this field than by a deep sense of dissatisfaction (on the part of this writer, at any rate) with the state of a design discipline that is ridden with flaws.

There are, however, indications that because of startling advances in technology, computer architects may well in the very near future face the problem of unmastered complexity so familiar to software designers. For instance, the development of large-scale integration (LSI) and (now) very large-scale integration (VLSI) semiconductor technology has resulted in a potential for processor chips of unprecedented gate complexity. Thus computer architects are already grappling with design problems that are quite different from those of a decade ago, simply because the technological constraints have changed so rapidly (Treleaven [1980]).

Furthermore, as an outcome of the large-scale availability of cost-effective microprocessors, the possibilities of distributed systems, networks, and multiprocessors are now widely recognized. Thus computer structures of unprecedented organizational complexity are being studied, designed, and experimented with.

Finally, these same technological changes have made the vertical migration of operating system functions and language primitives down to the microcode level a viable system implementation option. Thus microprograms are likely to become progressively larger and more complex, with an attendant increase in functionality and complexity of the overall architecture.

We are thus well advised to anticipate the emergence of far greater complexity in the architectural domain than might seem to exist at present.

This monograph is about the methodology of design of computer architecture. The term "methodology" is used here in its narrowest sense of "the study or description of the methods or procedures used in some activity" (Bullock and Stallybrass [1977], pp. 387–388)—in this case, computer architecture. In particular, this book is concerned with a particular approach to the design of architectures, based on the use of symbolic descriptions.

Style, according to H. A. Simon, is one way of doing things chosen from a number of alternative ways. It is normally identified by characteristics of the objects designed—as, for example, when we talk about a given computer processing a pipeline style of architecture on the basis of certain characteristics of information flow through the hardware during the course of a computation. As Simon has pointed out, it may also arise from differences in the design process. A good example from the software domain is provided by D. L. Parnas (1972) who, in a famous paper, showed that depending on the criteria used for decomposing systems into modules, one may obtain a software product organized in two quite different ways. The design process leaves its imprint on the final product; the alternative modularization techniques represent distinct design styles.

From this viewpoint it may be fair to say that the focus here is also on a particular style of architectural design, a style characterized mainly by the use of symbolic description languages for the specification of architecture at different, significant levels of abstraction.

Needless to say, it is not my place to advocate this approach to thinking about and designing computer architectures as *the* style to adopt. It is my objective, however, to promulgate the view that a sine qua non for the emergence of computer architecture as a design discipline is the elimination of the flaws described at the beginning of this preface. With a symbolic description language, the activities of designing, evaluating, modifying, and understanding an architecture can be conducted quite apart from, largely independent of, and preferably prior to the task of implementing the architecture in physical hardware. The proposed design style, as well as several other issues of architectural design discussed in this monograph, is intended as a contribution to such a discipline of computer architecture.

A few words must be said about the languages defined and used in this book. These languages (which, for reasons that will be evident later, form part of what I call the S* family) represent the outcome of work carried out over the past few years by this author and his students, on the development of a *family* of languages for the design, description, and verification of computer architectures and microprograms. Indeed, the present monograph is largely the culmination of this earlier work. Thus, a detailed discussion of the S* family as part of the present subject matter seemed both natural and necessary.

At the same time I must add a disclaimer. This monograph must not be interpreted as advocating these particular languages in preference to others that have been proposed in the literature. The rationale underlying the de-

sign of the S* family has been stated in earlier publications and is repeated here for the sake of completeness. It must remain for the reader to assess the family and decide on its usefulness relative to alternative proposals. Needless to say, as the history of programming languages amply illustrates, much experience must be gained with the application of design languages before any proper assessment can be achieved. At this time of writing, probably the only architecture description (or microprogramming) language that has been adopted to any degree by users outside the immediate locus of its invention is ISPS (Barbacci et al. [1978]). One of the reasons for devoting considerable space in this book to a detailed discussion of the S* family is the hope that others will seek to apply it, criticize it, and provide the necessary insight for the derivation of better and more elegant languages in the future.

SUBRATA DASGUPTA

Lafayette, Louisiana
November 11, 1983

ACKNOWLEDGMENTS

I am grateful for having had the opportunity of presenting some of the ideas discussed in this book in seminars at various universities and conferences over the last two years.

Parts of the manuscript were read by Werner Damm, Alan Wagner, Joseph Linn, Prasenjit Biswas, Harold Lorin, and Marius Olafsson. I thank them for their comments. I am also indebted to Marius Olafsson for allowing me to quote excerpts from his thesis.

At one time or another, I have had discussions with many colleagues and students on the topics of computer architecture, microprogramming, verification, and design methodology, many of which have influenced the shape of this book. In particular, I remember with pleasure conversations with Maurice Wilkes, Werner Damm, Michael Flynn, Alan Wagner, Marius Olafsson, Bruce Shriver, and Joseph Linn. Needless to say, none of the above are responsible for the opinions and ideas expressed in this book.

I am extremely grateful to Judith Abbott, who typed the major part of the book and to Cathy Pomier and Phil Wilsey for their assistance in completing the preparation of the manuscript. I must also thank James Gaughan and the staff of Wiley-Interscience for their invaluable editorial help.

My thanks to the Institute of Electrical and Electronics Engineers, the Association for Computing Machinery, AFIPS Press, John Wiley and Sons, Academic Press, and Springer Verlag for granting me permission to reproduce diagrams and excerpts from their publications.

Finally, a very personal note of gratitude to my parents, my parents-in-law, and to my wife for their love, support, and understanding over the years.

S.D.

THE DESIGN AND DESCRIPTION OF COMPUTER ARCHITECTURES

CONTENTS

ONE

WHAT IS COMPUTER ARCHITECTURE?

1.1 INTRODUCTION

Precisely what constitutes *computer architecture* has been a point of some debate in computer science. The crux of the problem appears to be the complex, hierarchic nature of computers. They are, first of all, complex in Simon's (1981) sense—they are composed of a *large number of parts* that interact in a nontrivial way; and hierarchy is a necessary way of organizing such a complex system. This is illustrated schematically by Fig. 1.1, where the individual system parts a through g are conceptually, logically, or physically (depending on the actual system) organized into two higher hierarchic levels. The relationship between adjacent levels is one of "consists of." Thus, α consists of A, B, and C; A consists of a, b, c, and so on.

In the case of computer systems, this kind of hierarchy is seen in the way we divide information into chunks when we design, describe, or understand such systems; thus, we talk of a system being composed of a processor, a memory, and a control unit. The processor in turn may consist of a local store, an arithmetic logic unit (ALU), multiplexers, and buses, while the control unit may be composed of a control memory, a set of specialized registers and a sequencer, and so on (Fig. 1.2).

A second kind of complexity arises from the many different *levels of description* that exist for computers. As pointed out by Bell and Newell (1971), these levels are not equivalent in the sense that anything said one way can be said another. On the contrary, each description level performs a

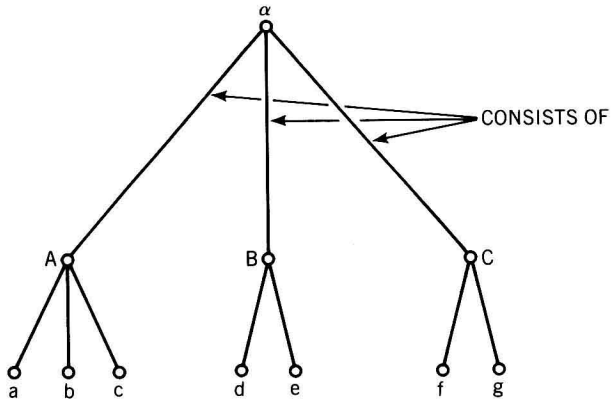

Figure 1.1 Hierarchy of several interacting components.

function that cannot be performed adequately by another. This is illustrated schematically by Fig. 1.3. The essential relationship between levels is one of abstraction or, conversely, refinement. Thus, level 1 is an abstraction of level 2; conversely, level 2 is a refinement of level 1.

A third kind of complexity arises when a system is *constructed* level by level. System complexity in this context appears—at least in some cases—to be the effect rather than the cause of multiple construction levels, since a system may have been designed or constructed in this form for reasons having nothing to do intrinsically with the management of complexity. The resulting system appears complex as a consequence of its many levels. Note that the relationship between levels here is one of "is implemented on" or "is constructed on" (Fig. 1.4).

A computer system, then, is complex and hierarchic in a number of different ways: it is composed of a large number of parts that interact in a nontrivial way; it is rich in the variety of levels at which it may be meaningfully described; and it may possibly be constructed as a many-layered system.

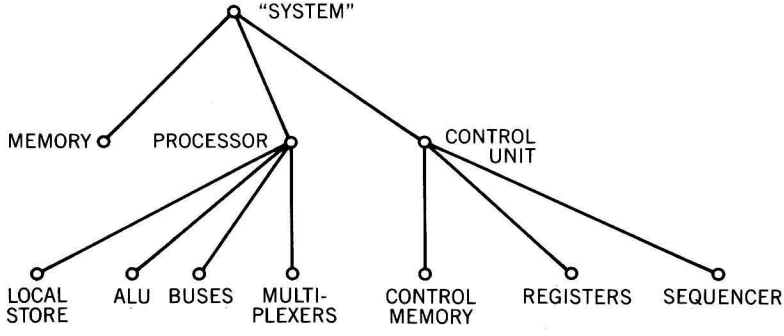

Figure 1.2 Hierarchy in a computer hardware system.