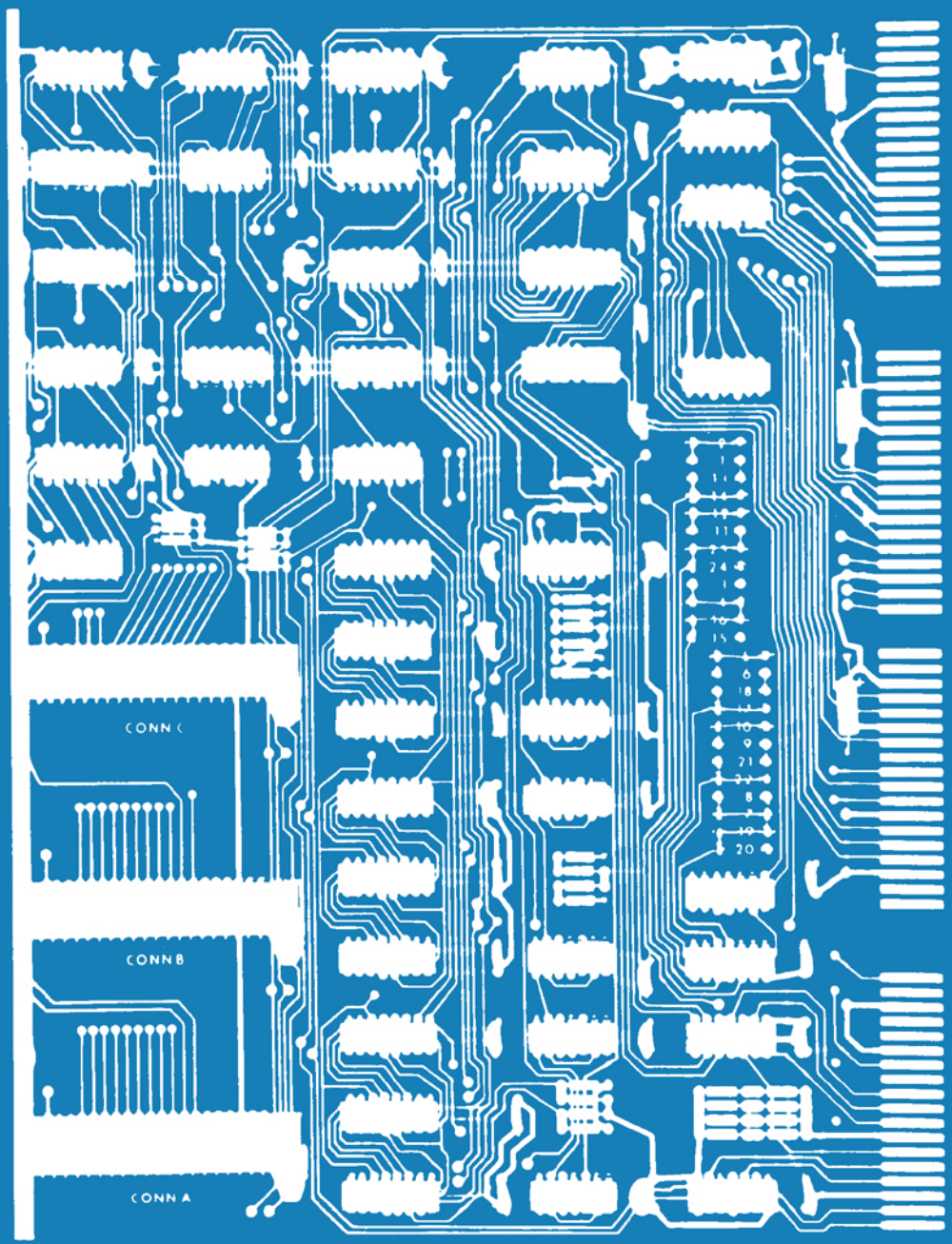
COMPUTER ENGINEERING

A DEC VIEW OF HARDWARE SYSTEMS DESIGN



C. GORDON BELL · J. CRAIG MUDGE · JOHN E. McNAMARA

COMPUTER ENGINEERING

A DEC VIEW OF HARDWARE SYSTEMS DESIGN

C. GORDON BELL · J. CRAIG MUDGE · JOHN E. McNAMARA

DIGITAL PRESS

Printed in U.S.A.

1st Printing, September 1978
2nd Printing, December 1978
3rd Printing, January 1979
4th Printing, August 1979
Documentation Number JB066-A
Library of Congress Catalog Card Number 77-91677
ISBN 0-932376-00-2

The manuscript was created on a DEC Word Processing System and, via a translation program, was typeset on Digital's DECset-8000 Typesetting System.

Cover and display pages designed by Elliott N. Hendrickson.

To the people at Digital, especially
the engineers, and Ben

This page intentionally left blank

FOREWORD

The progress which has brought the number of computers in use in the world from dozens to millions within a generation has not been the result of a single discovery or the work of a single inventor or company. Rather, men and women from fields as diverse as semiconductor physics and mechanical engineering have studied long hours and worked with various measures of inspiration and perspiration to make the discoveries and develop the technologies needed to advance the state of the art in computer technology.

There are several aspects of the progress in computer technology which have made it an exceptionally exciting and rewarding field for the people involved. First of all, a great many of the major steps forward, such as the invention of the transistor, have taken place within our lifetimes. Secondly, there has been an opportunity to associate with many fine colleagues whose brilliance, courage of conviction, and capacity for endless work have been a great inspiration. Finally, there has been the great promise of computers – their ability to free men's minds of repetitive and boring tasks, their ability to reduce the cost of producing goods, their ability to improve the lives of so many people in so many ways – and the fun and excitement of working with them.

In the chapters of this book, various authors relate some of their experiences in the past twenty years, draw some conclusions about how computer technology got to where it is, and project into the future from some of the trends they have seen. While it is impossible in a single book to capture all of the excitement and challenge of these years, they have done an admirable job for which they are to be commended. Hopefully, this glimpse into the past and present will encourage the students of the future to enter the computer engineering field and bring with them ideas, ambition, and courage.

Kenneth H. Olsen
President
Digital Equipment Corporation

This page intentionally left blank

PREFACE

This book has been written for practicing computer designers, whether their domain is microcomputers, minicomputers, or large computers, and for those who by their contact with computer are students of design – users, programmers, designers of peripherals and memories, and students of computer engineering and computer science.

Computer engineering is a collage of different activities and disciplines, only one of which – the technical aspects (multiplier design, the behavior of synchronizer circuits, and series/parallel tradeoffs, for example) – is covered by conventional texts. This book uses the case study method to show how all the different factors (technology push, the marketplace, manufacturing, etc.) form the real-world constraints and opportunities which influence computer engineering.

Computer engineering can be thought of as a multivariable mathematical problem in which the engineer searches for an optimum within certain constraints. Unfortunately, an optimum in one variable is rarely an optimum in another, and thus a major portion of computer engineering is the search for reasonable compromises. A common method used to aid the search is to assign weights to various system variables and to seek a weighted optimum. The weights vary with the intended application. In one situation, speed might receive the maximum weight; in another, instruction set compatibility might be the most important; and in yet another, reliability might be paramount. The number of dimensions to the problem is large, and the meaningful measures for them are few. For example, the cost variable is multidimensional and includes manufacturing, development, and field support costs. In addition, there are numerous interdependencies among the variables such as the relationships between instruction set, machine organization, logic design, and circuit design. These relationships and the contraints that control the weighting of the variables change with time. For example, the cost function changes when different subsystems use different technologies, and this influences the relationships. In addition, constraints such as maintainability and

compatibility vary in importance from year to year. Finally, while some of the relationships, such as the time-space tradeoff in adder design, are well understood, others, particularly those involving marketing factors, are not.

Because no theory exists to undergird this multidimensional design problem, we believe that there is no substitute for an extensive, critical understanding of the existing examples of designed and marketed systems. Therefore, this book uses the case study approach. For examples, we have used the thirty DEC computers that have been built over the twenty years that the company has existed, plus some PDP-11-based machines built at Carnegie-Mellon University. Carnegie-Mellon's machines explore interconnect structures that we feel will form the basis of future generations.

The association between DEC and Carnegie-Mellon has produced not only some interesting machines to examine but also some of the written material for this book. People in universities can and do write, whereas engineers directly involved in design work are less inclined or encouraged to publish their work.

A substantial portion of the material contributed by DEC authors is historical. We strongly believe that historical information is worth the expense in terms of writing, reading, and learning; machine design principles and techniques change slowly. In fact, the machines currently being designed are based on principles that have been understood and used for years, and we are often asked, "Are we running out of design issues?" Yes, we feel technology provides the forcing function for new designs, not new principles.

Learning about design is always important. Although new designs often appear to be a reapplication of old principles, in the process of being reapplied they change and go beyond their first application. Design is learned by examining and emulating previous designs plus incorporating general principles, new use, and new technology. Indeed, the microcomputer developments draw (or should draw) extensively from the minicomputers. As we build new structures, we should be able to avoid the pitfalls of the immediate past design.

We have intentionally restricted our scope to DEC computers. The reason is obvious: we can speak with first-hand knowledge. If we had used other companies' designs, our data would have been less accurate, and some factors, e.g., design styles, would have been omitted. The main reason, however, is a key part of the philosophy of the book. To understand machine design evolution, the effects of changes in the underlying technologies, and time-invariant principles, we must analyze a family beginning at birth and follow it over several generations of technology. Four series of DEC computers allow such an analysis. DEC computers also provide an opportunity to study another dimension of computer engineering – the coexistence of complementary (and sometimes competing) products. Particular design efforts must compete for resources (design talent, manufacturing-plant capacity, and software, marketing, and sales support). DEC computers have, in general, been designed to be complementary and to avoid overlapping or redundant products. Thus, another set of constraints can be seen at work in the design space.

The book concerns itself with general purpose computers which are intended to be widely available commercially. The engineering of computers for highly specialized applications, for which only a few copies are built, is not treated. Moreover, because not all major principles of computer architecture and computer engineering are embodied in the DEC computers, the reader may want to examine other designs, as well. For example, the reader cannot learn about descriptor architectures, array processors, list-processing machines, or general purpose emulators from this book.

At one time consideration was given to postponing the publication of a book until 1982, at which time DEC will celebrate its twenty-fifth anniversary. This idea was rejected because another five years would further impede the collection of data about the early machines. More importantly, the twenty-year period of DEC modules and computers (1957–1977) has extended from the early second generation to the fourth generation. Today, the processor of several DEC computers occupies a single large-scale integrated circuit consisting of several thousand transistors, whereas in 1957 only one transistor could be fabricated on a single piece of germanium. In another five years, the design, manufacture, and distribution of computers will be radically different – so much so as to merit a new book.

We expect an increasingly larger number of people to be involved in computer engineering and hence students of this material, because we expect computers as we know them today will disappear within ten years! With the processor-on-a-chip, the number of computer *systems* designers (users) has risen by several orders of magnitude.

In the area of large computer systems, the buyers and users are also clearly the computer designers: they select components (from the set of available components) and interconnect them to form specific structures. It is essential for us all to have a model of the price, performance, and reliability parameters and how they vary with time. Previous generations have focused first on the invention of the computer, next on the understanding of price/performance tradeoffs, and most recently on manufacturing – especially the fabrication of the semiconductors that now drive computer evolution. In the next five years, design will focus on applications: conventional applications will be more efficient, computers will be extended to reach new applications, and life-cycle costs will receive more attention. For the computer engineer, the evolution of DEC machines provides an excellent perspective on the influence of applications on design. For those of us who must deal with design goals, constraints, and objective functions to improve reliability, availability and maintainability, it is imperative that we first clearly understand previous design problems.

For the programmers who use computers and are a part of the computer design process, understanding this material is mandatory in order to know the rules of the game. We say comparatively little about software, other than how it has influenced hardware design. The increasing role of software functions in the hardware domain is a clear process that has allowed (and forced) computer architecture to change. The engineering of DEC software will be treated in subsequent

volumes, perhaps one on language translators and one on operating systems. We hope also that future volumes will be devoted to mass storage devices, terminals, and applications.

Two notations, ISP and PMS, were introduced in the book, *Computer Structures* [Bell and Newell, 1971]. We continue to use them in this book, especially since they have left the realm of notations and have become working design tools. ISP was introduced to describe the instruction set processor of a computer – the machine seen by the program (and programmer). ISP is now used for machine description, simulation, verification of diagnostics, microprogramming, automatic assembler generation, and the comparison of computer architectures. The evolution and improvement of ISP is principally due to needs of the Army/Navy Computer Family Architecture (CFA) project and the work of Mario Barbacci. The latest version, ISPS, is being used within DEC for implementing processors, simulators, etc. ISPS language descriptions of current DEC machines (PDP-8, PDP-10, PDP-11, VAX-11) and several terminals have been made. We hope that these will be made widely available and so further stimulate the use of machine-description languages. The widespread application of good languages would help alleviate two current design problems: first, that of hand-crafted design tooling keeping up with the rate of introduction of new technologies and second, the problem of managing the ever-increasing complexity of computer structures. The PDP-8 description presented in Appendix 1 has been verified by machine diagnostics, in contrast to conventional descriptions.

PMS (processor-memory-switch) notation (given in Appendix 2) has not yet been widely used in formal methods to aid design. It has, however, been used extensively to describe computer structures. A prototype system which recognizes PMS and performs several performance analysis functions was constructed by Knudsen [1972]. Currently, ISPS is being extended to include the interconnection of computational blocks so that PMS and ISPS form a single system describing structure and behavior. In this book, we use PMS to describe functional blocks. However, all PMS components are enclosed to form a block diagram, unlike the original stick notation.

The book begins with three introductory chapters. The first presents the major themes to be illustrated by the book. We show that computer evolution has been based primarily on semiconductor and magnetic recording technologies. These technologies determine costs, and therefore price, performance, reliability, size, weight, power, and other dimensions which constitute the physical characteristics of the machines. The major theme of the book is that technology has enabled (or forced) three types of computers to be built:

1. Machines with constant performance and decreasing cost.
2. Machines with contant cost and increasing performance.
3. Radically new (large or small) structures, often research machines, which create new computer classes outside the evolution possibilities.

Chapter 2 traces the evolution of memory and logic technology. Engineering is firmly rooted in economics and inherently practical. Packaging (including component interconnections) is covered in Chapter 3 for a very pragmatic reason: of the total product cost of a small computer system, 50 percent is due to packaging and power, and these costs are rising. To further emphasize the practical aspects of engineering in Chapter 3, a section on high-volume manufacturing is included; the result of a designer's creativity must not only work but be buildable by production-line methods.

Following the introductory chapters are five parts:

I. In the Beginning

II. Beginning of the Minicomputer

III. The PDP-11 Family

IV. The Evolution of Computer Building Blocks

V. The PDP-10 Family

The introductions to each part describe what to look for in the evolution of each machine: its interaction with designers, technology, and use (marketplace). More importantly, we have tried to point out the classic (timeless – so far) design principles. Data that has become available since the original papers were published is also included.

Part I describes modules, the product on which DEC was initially founded. Chapter 5 shows how modules evolved and assimilated semiconductor technology in order to build computers.

The PDP-1 and other 18-bit machines and the PDP-8 began the minicomputer phenomenon as described in Part II. Although six computers form the 18-bit family, there is only one chapter devoted to them, primarily because there has been a dearth of written papers; this chapter was written for *Computer Engineering*. Chapter 7 shows the historical development of the 12-bit machines, and Chapter 8 explores the structure of the PDP-8 in detail.

Part III, nearly two-thirds of the book, is based on the PDP-11. The PDP-11 has been implemented with multiple technologies and multiple design goals at a given time, i.e., a set of machines to span a performance range. Because of cost and performance goals, a number of problems have had to be solved to permit subsetting (for the LSI-11) and supersetting (for the larger memory PDP-11/70 and for VAX-11).

Part IV is devoted to module set evolution. Chapter 18 describes the Register Transfer Modules (RTMs, also called PDP-16), a set of modules for building

digital systems. Although these modules were unsuccessful in the marketplace, they were the forerunner of the bit-slice approach now widely used for implementing mid-range processors and special-purpose digital systems. Chapter 20 describes a set of modules based on the PDP-11 computer, called Computer Modules, which grew out of the original RTM research and were used to construct Cm*, a multi-microprocessor system.

Part V covers the PDP-10. Prior to the publication of the paper reproduced here as Chapter 21, very little had been published at the engineering level. The published literature had emphasized operating systems, languages, networks, and applications.

Computer Engineering is modeled after *Computer Structures* [Bell and Newell, 1971] and is intended to complement the subject matter therein. *Computer Structures* treats the design of instruction set architectures; *Computer Engineering* treats the design of machines which *implement* instruction sets. *Computer Structures* covers a broad range of ISP structures and PMS structures, from early stack machines and bit-serial machines, through list processors and higher level language machines, to supercomputers. By giving the seminal Burks, Goldstine, and von Neumann paper and the Whirlwind paper, it reaches far back into history. *Computer Engineering* on the other hand, takes a much narrower set of ISPs (four) and examines their implementations in detail. Instruction set design is mentioned only as it interacts with implementation. We focus on four computer families from both the designer and the historical viewpoint. In particular, we emphasize the lower level technological, economic, organizational, and environmental forces affecting the evolution of DEC computer families.

Although this book is principally for designers and students, it will also be of interest (as an historical record) to DEC employees who have been involved in the design, manufacture, distribution, and servicing of the computers.

Our recommendations for the use of this text in university curricula are based on teaching experience, requests from academic colleagues for material to teach design, and our participation in curriculum development. The book directly addresses the philosophy of the IEEE Computer Society Task Force on Computer Architecture [Rossman *et al.*, 1975]: "To appreciate how the architectures of computer systems develop, one must analyze complete systems." As such, *Computer Engineering* serves to complement Buchholz [1962], Bell and Newell [1971], and Blaauw and Brooks [in preparation] in a course on computer architecture, for example, IEEE course CO-3.*

For undergraduate courses on computer organization, such as IEEE CO-1* and the ACM courses I3 and A2†, we believe that the book could be used as a supplementary text. In a course on computer engineering, using the style given in

*"A Curriculum in Computer Science and Engineering-Committee Report," Model Curricula Subcommittee, IEEE Computer Society, EH0119-8, January 1977.

†"Curriculum 68," Commun. ACM, *11*, 3, pp. 151–197, March 1968.

the syllabus of CO-2* (I/O and Memory Systems) as a model, this could be a primary text, provided that material on other manufacturers' computers is made available to show different viewpoints.

ACKNOWLEDGEMENTS

We gratefully acknowledge our contributing authors, whose insights have greatly enhanced the scope of this book, and our colleagues at DEC, who assembled information, and provided subject matter expertise and advice.

We would like to thank R. Eckhouse, R. Glorioso, S. Fuller, J. Lipovski, and P. Jessel whose critiques of the preliminary drafts of the introductory chapters and book outline proved very helpful. We would also like to thank J. Cudmore, R. Doane, R. Elia-Shaoul, S. Fuller, L. Gale, L. Hughes, R. Peyton, and S. Teicher, who provided data for Chapter 2 and valuable critiques of earlier drafts. We also acknowledge the reviewers of the second draft of the manuscript, to whose criticisms we have especially tried to respond. We received instructive comments and evaluations from D. Aspinall, G. Blaauw, R. Clayton, D. Cox, J. Dennis, P. Enslow, D. Freeman, J. Grason, J. Gray, W. Heller, G. Korn, J. Lipcon, J. Marshall, E. McCluskey, C. Minter, M. Moshell, E. Organick, W. Schmitt, B. Schunck, I. Sutherland, J. Wakerly, and J. Wipfli. We would like to extend special thanks to H. Stone for his extensive and particularly useful review comments.

We are also indebted to many for their support in producing *Computer Engineering*. We are particularly indebted to Heidi Baldus of Digital Press who coordinated the production of *Computer Engineering* and whose encouragement kept us going through a number of difficult times. For their expertise and patience, we thank the Technical Documentation group, especially Denise Peters. We also thank Mary Jane Forbes and Louise Principe for their constant support in the course of this book's development and production. The manuscript creation and preparation on the DEC Word Processing System, followed by transmission to the DECset-8000 Typesetting System, permitted numerous drafts and rapid creation of the final typeset material.

C.G.B.
J.C.M.
J.E.M.

August 1978

This page intentionally left blank

ACKNOWLEDGEMENTS

C.G. Bell, J.C. Mudge, and J.E. McNamara: Seven Views of Computer Systems. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; J.C. Mudge and J.E. McNamara, Digital Equipment Corporation.

C.G. Bell, J.C. Mudge, and J.E. McNamara: Technology Progress in Logic and Memories. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; J.C. Mudge and J.E. McNamara, Digital Equipment Corporation.

C.G. Bell, J.C. Mudge, and J.E. McNamara: Packaging and Manufacturing. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; J.C. Mudge and J.E. McNamara, Digital Equipment Corporation.

K.H. Olsen: Transistor Circuitry in the Lincoln TX-2. Copyright © 1957 by AFIPS. Reprinted, with permission, from the *Proceedings of the Western Computer Conference,* 1957, pp. 167–171. This work was supported jointly by the U.S. Army, Navy, and Air Force under contract with M.I.T. K.H. Olsen, Lincoln Laboratory M.I.T. (currently with Digital Equipment Corporation).

R.L. Best, R.C. Doane, and J.E. McNamara: Digital Modules, the Basis for Computers. R.L. Best, R.C. Doane, and J.E. McNamara, Digital Equipment Corporation.

C.G. Bell, G. Butler, R. Gray, J.E. McNamara, D. Vonada, and R. Wilson: The PDP-1 and Other 18-Bit Computers. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; G. Butler *et al.,* Digital Equipment Corporation.

C.G. Bell and J.E. McNamara: The PDP-8 and Other 12-Bit Computers. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; J.E. McNamara, Digital Equipment Corporation.

C.G. Bell, A. Newell and D.P. Siewiorek: Structural Levels of the PDP-8. Revised and updated version of Chapter 5, "The DEC PDP-8," *Computer Structures: Reading and Examples,* C.G. Bell and A. Newell, McGraw-Hill Book Co., New York, 1971. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; A. Newell and D.P. Siewiorek, Carnegie-Mellon University.

C.G. Bell *et al.:* A New Architecture for Minicomputers – The DEC PDP-11. Copyright © 1970 by AFIPS. Reprinted, with permission, from the *Proceedings of the Spring Joint Computer Conference,* 1970, pp. 657–675. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University. Those who have contributed subject matter expertise include R. Cady, H. McFarland, B.A. Delagi, J.F. O'Loughlin, R. Noonan, and W.A. Wulf.

W.D. Strecker: Cache Memories for PDP-11 Family Computers. Copyright © 1976 by the Institute of Electrical and Electronics Engineers, Inc. Reprinted, with permission, from the *Proceedings of the 3rd Annual Symposium on Computer Architecture,* 1976, pp. 155–158. W.D. Strecker, Digital Equipment Corporation.

J.V. Levy: Buses, The Skeleton of Computer Structures. J.V. Levy, Digital Equipment Corporation (currently with Tandem Computers, Inc.).

M.J. Sebern: A Minicomputer-Compatible Microcomputer System: The DEC LSI-11. Copyright © 1976 by the Institute of Electrical and Electronics Engineers, Inc. Reprinted, with permission, from the *Proceedings of the IEEE,* June 1976, Vol. 64, No. 6. Manuscript received by IEEE on October 10, 1975; revised December 12, 1975. M.J. Sebern, Digital Equipment Corpoation (currently with Sebern Engineering, Inc.).

J.C. Mudge: Design Decisions for the PDP-11/60 Mid-Range Minicomputer. Copyright © 1977 by the Computer Design Publishing Corp. Reprinted, with permission, from *Computer Design,* August 1977, pp. 87–95. Appears under title "Design Decisions Achieve Price/Performance Balance in Mid-Range Minicomputers" in *Computer Design* issue. J.C. Mudge, Digital Equipment Corporation.

E.A. Snow and D.P. Siewiorek: Impact of Implementation Design Tradeoffs on Performance: The PDP-11, A Case Study. Copyright © 1978 by Edward A. Snow and Daniel P. Siewiorek. This research was supported in part by the National Science Foundation under grant GJ-32758X and by an IBM fellowship. Engineering documentation was supplied by the Digital Equipment Corporation. E.A. Snow (currently with Intel Corp.) and D.P. Siewiorek, Carnegie-Mellon University.

R.F. Brender: Turning Cousins into Sisters: An Example of Software Smoothing of Hardware Differences. R.F. Brender, Digital Equipment Corporation.

C.G. Bell and J.C. Mudge: The Evolution of the PDP-11. Chapter includes material from "What Have We Learned From the PDP-11?" by C.G. Bell, in *Perspectives on Computer Science:* From the 10th University Symposium at the Computer Science Department, Carnegie-Mellon University, A. Jones (Ed.), Academic Press, Inc., 1978. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; J.C. Mudge, Digital Equipment Corporation.

W.D. Strecker: VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family. Copyright © 1978 by American Federation of Information Processing Societies, Inc. Reprinted, with permission, from the *Proceedings of the National Computer Conference,* June 1978, pp. 967–980. W.D. Strecker, Digital Equipment Corporation.

C.G. Bell, J. Eggert, J. Grason, and P. Williams: The Description and Use of Register Transfer Modules (RTMs). Copyright © 1972 by the Institute of Electrical and Electronics Engineers, Inc. Reprinted, with permission, from the *IEEE Transactions on Computers,* May 1972, Vol. C-21, No. 5, pp. 495–500. Manuscript received by IEEE February 19, 1971; revised May 11, 1971. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; J. Eggert, Digital Equipment Corporation (currently with Eggert Engineering); J. Grason, Carnegie-Mellon University (currently with Bell Laboratories); P. Williams, Digital Equipment Corporation (currently with Data Terminal Systems, Inc.).

T.M. McWilliams, S.H. Fuller, and W.H. Sherwood: Using LSI Processor Bit-Slices to Build a PDP-11 – A Case Study in Microcomputer Design. Copyright © 1977 by AFIPS. Reprinted, with permission, from the *Proceedings of the National Computer Conference,* 1977, pp. 243–253. This work was partially supported by the Advanced Research Projects Agency (ARPA) of the Department of Defense under contract F44620-73-C-0074, monitored by the Air Force Office of Scientific Research. T.M. McWilliams, Carnegie-Mellon University (currently with Stanford University and Lawrence Livermore Laboratory, University of California); S.H. Fuller, Carnegie-Mellon University (currently with Digital Equipment Corporation); W.H. Sherwood, Carnegie-Mellon University (currently with Digital Equipment Corporation).

S.H. Fuller, J.K. Ousterhout, L. Raskin, P. Rubinfeld, P.S. Sindhu, and R.J. Swan: Multi-Microprocessors: An Overview and Working Example. Copyright © 1978 by Institute of Electrical and Electronics Engineers, Inc. Reprinted, with permission, from the *Proceedings of the IEEE,* February 1978, Vol. 61, No. 2, pp. 216–228. Manuscript received by IEEE November 11, 1977. This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract F44620-73-C-0074, which is monitored by the Air Force Office of Scientific Research, and in

part by the National Science Foundation under Grant GJ 32758X. The LSI-11s and related equipment were supplied by Digital Equipment Corporation. S.H. Fuller, Carnegie-Mellon University (currently with Digital Equipment Corporation); J.K. Ousterhout *et al.*, Carnegie-Mellon University.

C.G. Bell, A. Kotok, T.N. Hastings, and R. Hill: The Evolution of the DECsystem-10. Copyright © 1978 by the Association for Computing Machinery. Reprinted, with permission, from the *Communications of the ACM*, January 1978, Vol. 21, No. 1, pp. 44–63. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; A. Kotok, T.N. Hastings, and R. Hill, Digital Equipment Corporation.

M. Barbacci: Appendix 1 – An ISPS Primer for the Instruction Set Processor. M. Barbacci, Carnegie-Mellon University.

J.C. Mudge: Appendix 2 – The PMS Notation. J.C. Mudge, Digital Equipment Corporation.

C.G. Bell, J.C. Mudge, and J.E. McNamara: Appendix 3 – Performance. C.G. Bell, Digital Equipment Corporation and Carnegie-Mellon University; J.C. Mudge and J.E. McNamara, Digital Equipment Corporation.

TRADEMARKS

The following trademarks appear in *Computer Engineering: A DEC View of Hardware Systems Design.*

Company	Trademark	
Computer Automation Corporation	Naked Mini	
Digital Equipment Corporation	DEC	DECsystem-10
	DECSYSTEM-20	DECtape
	DECUS	DDT
	DIBOL	DIGITAL
	Fastbus	Flip Chip
	FOCAL	LSI-11
	Massbus	PDP
	RSTS	RSX
	TOPS-10	TOPS-20
	Unibus	
Fairchild Camera and Instrument Corporation	Macrologic	
Friden Company – A Division of Singer Company	Flexowriter	
Gardner-Denver Company	Wire-wrap	
Teletype Corporation	Teletype	
Xerox Corporation	Xerox 6500 color graphics printer	

CONTENTS

1

Seven Views of Computer Systems

C. GORDON BELL, J. CRAIG MUDGE,
and JOHN E. McNAMARA

A computer is determined by many factors, including architecture, structural properties, the technological environment, and the human aspects of the environment in which it was designed and built. In this book various authors reflect on these factors for a wide range of DEC computers – their goals, their architectures, their various implementations and realizations, and occasionally on the people who designed them.

Computer engineering is the complete set of activities, including the use of taxonomies, theories, models, and heuristics, associated with the design and construction of computers. It is like other engineering, and the definition that Richard Hamming (then at Bell Laboratories) gave is especially appropriate: engineers first turn to science for answers and help, then to mathematics for models and intuition, and finally to the seat of their pants.

In the few decades since computers were first conceived and built, computer engineering has come from a set of design activities that were mostly seat-of-the-pants based to a point where some parts are quite well understood and based on good models and rules of thumb, such as technology models, and other parts are completely understood and employ useful theories such as circuit minimization.

In this chapter, seven views are presented that the authors have found useful in thinking about computers and the process that molds their form and function. They are intentionally independent; each is a different way of looking at a computer. A computer scientist or mathematician sees a computer as levels-of-interpreters. An engineer sees the computer on a structural basis, with particular emphasis on the logic design of the structure. The view most often taken by a buyer is a marketplace view. While these people each favor a particular view of computers, each typically understands certain aspects of the other views. The goals of Chapter 1 are to increase this understanding of other views and to increase the number of representations used to describe the object of study and, hence, improve on its exposition. Thus, "The Seven Views of Computer Systems" forms a useful background for the subsequent chapters on past, present, and future computers.

VIEW 1: STRUCTURAL LEVELS OF A COMPUTER SYSTEM

In *Computer Stuctures* [Bell and Newell, 1971], a set of conceptual levels for describing, understanding, analyzing, designing, and using computer systems was postulated. The model has survived major changes in technology, such as the fabrication of a complete computer on a single silicon chip, and changes in architecture, such as the addition of vector and array datatypes.

As shown in Figure 1, there are at least five levels of system description that can be used to describe a computer. Each level is characterized by a distinct language for representing the components associated with that level, their modes of combination, and their laws of behavior. Within each level there exists a whole hierarchy of systems and subsystems, but as long as these are all described in the same language, they do not constitute separate levels. With this general view, one can work up through the levels of computer systems, starting at the bottom.

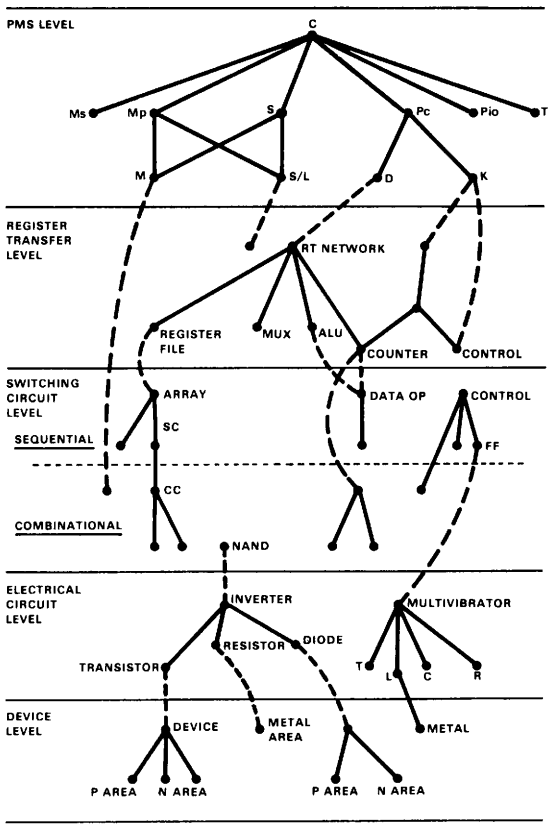


Figure 1. Hierarchy of computer levels, adapted from Bell and Newell [1971].

The lowest level in Figure 1 is the device level. Here the components are p-type and n-type semiconductor materials, dielectric materials, and metal formed in various ways. The behavior of the components is described in the languages of semiconductor physics and materials science.

The next level is the circuit level. Here the components are resistors, inductors, capacitors, voltage sources, and nonlinear devices. The behavior of the system is measured in terms of voltage, current, and magnetic flux. These are continuously varying quantities associated with various components; hence, there is continuous behavior through time, and equations (including differential equations) can be written to describe the behavior of the variables. The components have a discrete number of terminals whereby they can be connected to other components.

Above the circuit level is the switching circuit or logic level. While the circuit level in digital technology is very similar to the rest of electrical engineering, the logic level is the point at which digital technology diverges from electrical engineering. The behavior of a system is now described by discrete variables which take on only two values, called 0 and 1 (or + and −, true and false, high and low). The components perform logic functions called AND, OR, NAND, NOR, and NOT. Systems are constructed in the same way as at the circuit level, by connecting the terminals of components, which thereby identify their behavioral values.

After a system has been so constructed, the laws of Boolean algebra can be used to compute the behavior of the system from the behavior and properties of its components.

In addition to combinational logic circuits, whose outputs are directly related to the inputs at any instant of time, there are sequential logic circuits which have the ability to hold values over time and thus store information. The problem that the combinational level analysis solves is the production of a set of outputs at time t as a function of a number of inputs at the same time t. The representation of a sequential switching circuit is basically the same as that of a combinational switching circuit, although one needs to add memory components. The equations that specify sequential logic circuit structure must be difference equations involving time, rather than the simple Boolean algebra equations which describe purely combinational logic circuits.

The level above the switching circuit level is called the register transfer (RT) level. The components of the register transfer level are registers and the functional transfers between those registers. The functional transfers occur as the system undergoes discrete operations, whereby the values of various registers are combined according to some rule and are then stored (transferred) into another register. The rule, or law, of combination may be almost anything, from the simple unmodified transfer (A $\leftarrow$ B) to logical combination (A $\leftarrow$ B $\wedge$ (AND) C) or arithmetic combination (A $\leftarrow$ B + (PLUS) C). Thus, a specification of the behavior, equivalent to the Boolean equations of sequential circuits or to the differential equations of the circuit level, is a set of expressions (often called productions) that give the conditions under which such transfers will be made.

The fifth and last level in Figure 1 is called the processor-memory-switch (PMS) level. This level, which gives only the most aggregate behavior of a computer system, consists of central processors, core memories, tapes, disks, input/output processors, communications lines, printers, tape controllers, buses, teleprinters, scopes, etc. The computer system is viewed as processing a medium, information, which can be measured in bits (or digits, characters, words, etc.). Thus, the components have capacities and flow rates as their operating characteristics.

The program level from the original set of levels shown in Bell and Newell has been dropped because it is a functional rather than a structural level.

Many notations are used at each of the five structural levels. Two of the less common ones are the processor-memory-switch (PMS) and instruction set processor (ISP) notations. A complete description of these notations is given in Bell and Newell [1971: Chapter 2]. Those aspects of PMS that are used in this book are described in Appendix 2. The ISP notation has evolved to the ISPS language, which is described in Appendix 1.

VIEW 2: LEVY'S LEVELS-OF-INTERPRETERS

In contrast to the Structural View, this view is functional. According to this view, presented by John Levy [1974], a computer system consists of layers of interpreters, much like the layers of an onion.

An interpreter is a processing system that is driven by instructions and operates upon state information. The basic interpretive loop, shown in Figure 2, is most familiar at the machine language level but also exists at several other levels.

To formalize the notion of Levels-of-Interpretation, one can represent a processing system by the diagram in Figure 3.

The state information operated on by an interpreter is either internal or external. This can best be understood by considering the "onion skin" levels of the five processing systems that form a typical airline reservation system. These levels are listed in Table 1.

The Level 0 system is the logic that sequences the Level 1 micromachine. The Level 1 system is a microprogrammed processor implemented in real hardware. It is the machine seen by the logic designer. The Level 2 system is the central processing unit (CPU). It is the machine seen by the machine language programmer. The Level 3 system shown here is a FORTRAN language processing system. The Level 4 system is an airline reservation system. Four of these five systems form the hierarchy shown in Figure 4, where each system is an interpreter that sequences through multiple steps in order to perform a single operation for the next level interpreter. The highest level system, the airline reservation system, is an interpreter operating on messages received from outside of the system. It tests and modifies states and generates messages to send back outside the system, thus performing a single operation for the outermost interpreter.

In practice, few systems are levels of pure interpreters, although layers are present. Deviations from the model have occurred for both hardware and software reasons. In the hardware deviation case, the micromachine shown in Level 1 is often not present, but rather the Level 2 central processing unit is implemented directly using Level 0 sequential controllers. This practice of skipping Level 1 was initially due to the lack of adequate read-only memories but is now generally limited to the case of very high speed machines such as the Cray 1 and the Amdahl V6 which cannot tolerate the fetch and execute cycle times associated with a control store.

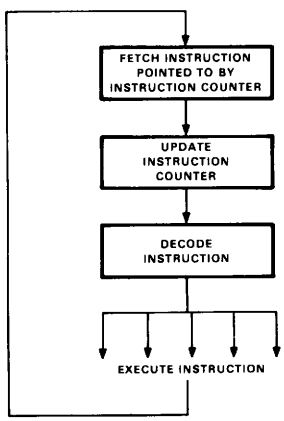


Figure 2. The basic interpretive loop [Levy, 1974].

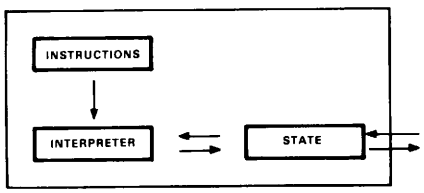


Figure 3. A processing system [Levy, 1974].

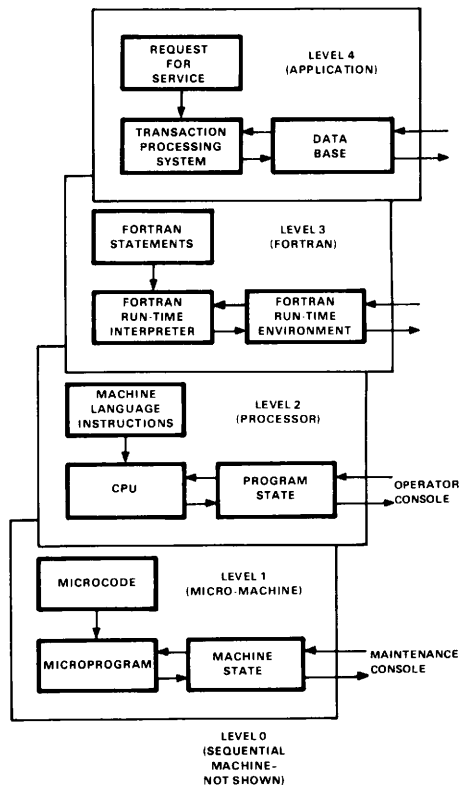


Figure 4. A hierarchy of interpreters [Levy, 1974].

Table 1. Five Levels-of-Interpreters for an Airline Reservation System [Levy, 1974]

Level 4	Instruction:	Seat allocation request message
	Interpreter:	Airline reservation system
	Internal state:	Number of requests pending at this moment Location of passenger list on a disk file Number of lines connected to system
	External state:	Number of reserved seats on a given flight Airline name for a given flight
Level 3	Instructions:	FORTRAN statement codes
	Interpreter:	FORTRAN execution system
	Internal state:	Memory management parameters User name Main storage size Location of disk files Interrupt enable bits Expression evaluation stack Dimensions of arrays
	External state:	Subroutine names Values of data in arrays Statement number Program size Value of an expression DO-loop variable value Printed characters on line printer
Level 2	Instructions:	Machine language instructions
	Interpreter:	Processor
	Internal state:	Program registers Condition codes Program counter
	External state:	Data in main memory Disk controller registers
Level 1	Instructions:	Microcode
	Interpreter:	Micromachine
	Internal state:	Instruction register Flip-flops holding error status Stack of microprogram subroutine links
	External state:	Program registers Condition codes Program counter
Level 0	Instructions:	Hardwired combinational network
	Interpreter:	Sequential machine controlling the micromachine
	Internal state:	Clock, counters, etc., controlling micromachine timing
	External state:	Micromachine, console

There are two primary software driven departures from the pure interpreter model: (1) high level languages are usually executed by a compiler rather than by an interpreter, and (2) some layers are bypassed when more ideal primitives exist at deeper levels. Figure 5 illustrates this bypassing process. A pure interpreter implementation of FORTRAN would use an object time system (OTS) for all FORTRAN *C* operations designated in the figure. The object time system would require an operating system (OPSYS) for the interpretation of some of its operations, and the operating system in turn would be interpreted by the instruction set interpreter (ISP interpreter). However, the *A* operations in the figure would be directly interpreted by the instruction set interpreter.

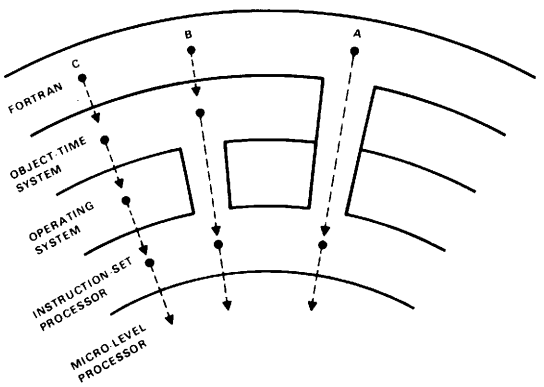


Figure 5. Levels-of-interpreters with "pipes" that bypass levels. FORTRAN operation *C* is interpreted by an OTS function which in turn is interpreted by the operating system which is interpreted by the ISP. FORTRAN operation *A* has a pipe directly to the ISP interpreter.

In the final analysis, the number of levels is just another tradeoff. Performance considerations lead to the deletion of levels; complexity leads to the addition of levels. Having presented the pure interpreter model, one can now return to the Onion-Skin-Layered Model to better understand how the different layers relate.

The macromachine hardware can be thought of as a base level interpreter. It is most often extended upward with an operating system. There may be several operating system levels so that the machine can be built up in an orderly fashion. A kernel machine might manage and diagnose the hardware components (disks, terminals) and provide synchronizing operations so that the multiple processes controlling the physical hardware can operate concurrently. Next, more complex operations such as the file system and basic utilities are added, followed by policy elements such as facilities resource management and accounting. As viewed through the operating system, one sees a much different machine than that provided by the basic instruction set architecture. In fact, the resultant machine is hardly recognizable as the architecture most usually given by a symbolic assembler. It includes the basic machine but has more capable I/O and often the ability to be shared by many programs (or tasks).

Operating systems designers believe all these facilities are necessary in order to implement the next higher level interpreter – the standard language. The language level may include interpreters or compilers to translate back to the machine architecture for ALGOL, BASIC, COBOL, FORTRAN, or any of the other standard languages and their dialects.

VIEW 3: PACKAGING LEVELS-OF-INTEGRATION

This is a structural view that packages the various components (hardware and software) into levels. The levels for DEC computers in 1978 were as follows:

9 Applications
8 Applications components
7 Special languages
6 Standard languages

5 Operating systems
4 Cabinets (to hold complete hardware systems)
3 Boxes
2 Modules (printed circuit boards)
1 Integrated circuits

This view is the most important in the book, because it shows how computer systems are actually structured and, hence, how their costs are structured. As a structural view of the object being sold, however, it is completely a function of the technology, the organization building the system, and the marketplace, all of which are changing so rapidly that the view could better be titled "Dynamic Levels-of-Integration." There are three major changes taking place:

1. Changes in the hardware levels, where the shrinking in physical size of functions has three effects:
   a. Lower levels subsume higher levels.
   b. The semiconductor component supplier is forced to assume higher and higher level design responsibilities.
   c. Levels disappear.
2. Changes in the software levels, again with three effects:
   a. Each level grows in size as more functionality is added over time.
   b. More levels are added as minicomputers are applied to a broader range of applications.
   c. Functions migrate downward from level to level.
3. Changes in the hardware/software interface, where software functions migrate into hardware for higher performance.

For the first of these areas of change, hardware levels, it is interesting to note that interconnection and packaging now constrain and limit design more than any other factor, excluding the basic lowest level component (semiconductor) technology.

The constraint caused by the interconnection and packaging takes place because most manufacturing costs are associated with the physical structure. As interconnection levels must be introduced to build complex structures, many usually undesirable side effects occur. Electrical interconnection requires cables which require space and interfere with cooling airflow. Long interconnections increase signal transmission delays, and these reduce performance. Signal transmission not only makes the computer susceptible to electromechanical interference but also may radiate electromagnetic waves that need to be controlled.

Figure 6 shows the costs of various levels-of-integration versus time for small computers. The cost depends partly on implementation and architecture word length. As the word length is made shorter, there are some savings, particularly for very small computers, because some levels-of-integration cease to exist. For example, most hand-held calculators are implemented using 4-bit, stored program computers with fixed programs that occupy a single integrated circuit. There are associated modules, backplanes, boxes, and cabinets – but all are contained in a single package that fits in the hand.

Semiconductors, the lowest level of technology, have had the greatest price decline (Figure 6). Modules have a lesser price decline because they are a mix of integrated circuits, printed circuit boards, component insertion labor, and testing labor. The price decline for the integrated circuit portion of the module cost is moderated by the labor-intensive nature of module fabrication, thus producing a price decline for modules that is markedly less than that for integrated circuits. At the box level-of-integration, power supplies and metal or plastic boxes are also labor-intensive and further moderate the price decline provided by the integrated circuits. Finally, as boxes are integrated (by people) and applied at a system

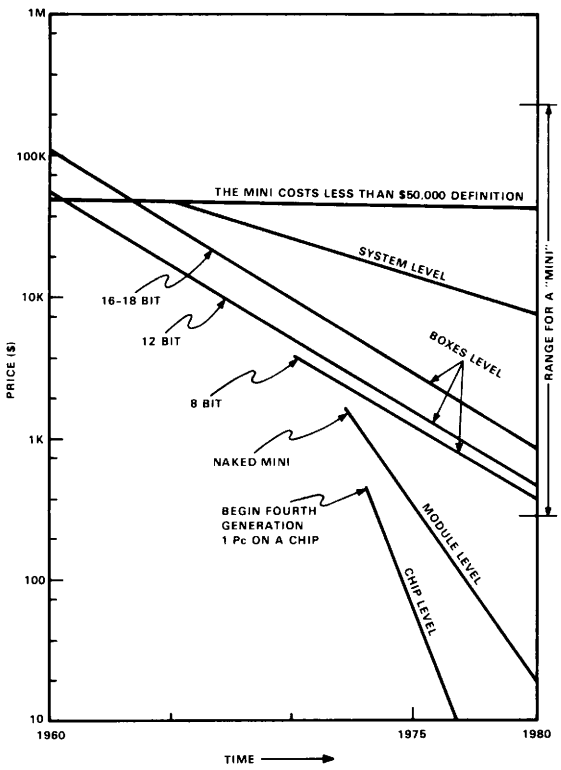


Figure 6. Machine price for various levels-of-integration versus time.

level (by people), the price decline almost disappears.

Many of the cost improvements brought about by new technology are derivative. They are by-products of using less power and less space, thus avoiding the labor-intensive levels of packaging integration.

An astute marketing-oriented person might ask, "How, with all the technology, can we do something unique so that we can maximize the benefit from the technology without having to pay so much for labor-intensive items such as packaging?" One answer: "Reduce prices by not providing a power supply and mounting hardware. Let the user provide all added-on parts and mount the computer as needed. In this way, the price, though not necessarily the total cost to the user, is reduced. We'll sell at the board level." Computer Automation followed this philosophy when it introduced the Naked Mini so that users could supply more added value (packaging and power technology).

A similar effect can be seen in the PDP-11 series since the PDP-11/20's introduction in 1970. At that time, the 4,096-word PDP-11/20 (mounted in a box) sold for $9,300. In 1976, the boxed version of an LSI-11 cost $1,995, reflecting a factor of 4.7 improvement over the PDP-11/20. The 4,096-word core memory module used in the PDP-11/20 sold for $3,500, while a 16,384-word metal-oxide semiconductor (MOS) memory module for an LSI-11 sold for $1,800, reflecting a factor of 7.8 improvement.

The changing levels-of-integration have also changed the domain of the semiconductor suppliers. In the early 1970s, Intel, North American Rockwell, and other semiconductor companies began to use the higher semiconductor densities to reduce the number of levels-of-integration by packaging a complete processor-on-a-chip. These organizations had assimilated logic design, but were frustrated because their customers could really not identify higher functionality units (beyond memory) requiring on the order of 1,000 gates on a chip. Also, the speed of these high density units was quite low.

They discovered that the best finite state machine to make was just a simple computer, because it provided the finite state machine plus the useful functions that were not covered by switching circuit theory. It was "simply a small matter of programming" to do something useful. Whereas programs for these simple computers cost $1 to $100 per instruction to write, the prices for processors-on-a-chip have followed a very steep decline of up to 50 percent price reduction per year.

Robert Noyce of Intel developed Figure 7 in October 1975. It illustrates what has been happening in the semiconductor industry and has been modified slightly to show the technology