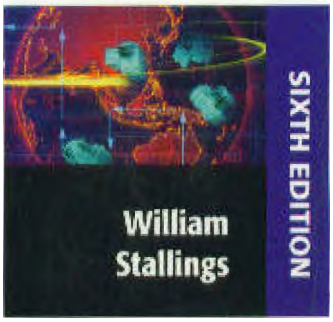
INTERNATIONAL EDITION

SIXTH EDITION

Computer Organization and Architecture

DESIGNING FOR PERFORMANCE

William Stallings



Computer Organization &Architecture

DESIGNING FOR PERFORMANCE

Williarn Stallings' book provides comprehensive and completely up-to-date coverage of computer organization and architecture including memory. VC], and parallel systems. The text covers leading-edge areas, including superscalar design, IA-54 design features, and parallel processor organization trends. It meets students' needs by addressing both the fundamental principles as well as the critical role of performance in driving computer design. Providing an unparalleled degree of instructor and student support, including supplements **and on-line** resources through the book's website, the sixth edition is in the forefront in its field.

NEW

- IA-64/Itanium architecture: chapter-length description and analysis that includes predicated execution and speculative loading.
- Cache memory: Cache memory is a central element in the design of high-performance processors. An entire chapter is devoted to this issue in the new edition.
- **Optical memory: expanded and updated.**
- **Advanced DRAM architecture;** more material has been added to cover this topic, including an updated discussion of SDRAM and RDRAM,
- **SMP's, dusters, and 141)MA systems; the** chapter on parallel organization has been expanded and updated.
- **Expanded instructor support: the book** now provides extensive support for projects with its new website,
- ***Pedagogy:** each chapter now includes **a** list of review questions (as well as homework problems) **and a** list of key words.

DISTINGUISHING KEY FEATURES

- **Running examples;** numerous concrete examples, especially Pentium 4 and Power PC G4.
- Bus **organization; detailed** treatment and evaluation of key design issues.
- **RISC broad, unified** presentation-
- ***Microprogrammed implementation:** lull treatment for a firm grasp.
- **I/O functions and structures:** provides full understanding and shows interaction of I/O modules with the outside world and the CPU.
- Unified **instructional approach:** enables student to evaluate instruction set design issues.
- **Instructors Resource CD-ROM: includes** solutions to homework problems, list of research project-S, list of simulation projects **plus student manual for both** SimpleScalar and SMPCache, and a list of suggested reading assignments.

THE AUTHOR'S WEBSITE:

http://www.WilliamStallings.comiC0A6e provides support for students, instructors and professionals:

- Links to important up-to-date site-related text **materials.**
- Provides transparency masters of figures and tables from the book in PDF format.
- Lists a set of course notes in PDP for handouts.
- Includes a set of PowerPoint slides for lecturing,

Pearson Education

Prentice Hall
Upper Saddle River
New Jersey 07458
wwwpreriKall.com

These are unabridged paperback reprints of established titles widely used by universities and colleges throughout the world. Pearson Education International publishes these lower-priced editions for the benefit of students. This edition rnay be sold only in th e countries to which it is consigned by Pearson Education International. It is not to be re-exported, and is not for sale **in the U.S.A., Mexico, or Canada.**

Prentice Hall International Editions

ISDN 0-13-04931:17-4

9 780130 49307 11 90000

THE WILLIAM STALLINGS BOOKS ON COMPUTE!?

DATA AND COMPUTER COMMUNICATIONS, SIXTH lannoN

A comprehensive survey that has become the standard in the field. covering (I) data communications, including transmission, media, signal encoding, link control, and multiplexing; (2) communication nchvorks. including circuit- and packet-switched, frame relay, ATM. and LANs: (3) the TCP/IP protocol suite. including IPv6, TCP. M1MF., and HTTP: as well as a detailed treatment of network security. **Received the 21111) Text and Academic Authors Association (TAA) award for tong-term excellence in a Computer Science Textbook.** ISBN 0-13-084370-9

CRYPTOGRAPHY AND NETWORK SECURITY, SECOND EDITION

A tutorial and survey on network security technology. Each of the basic building blocks of network security, including conventional and public-key cryptography, authentication, and digital signatures, are covered. The book covers important network security tools and applications, including S/MIME. IP Security, Kerberos, SSUmS. SET, and Vil19v3, In addition, methods for countering hackers and viruses are explored. **Received the TAA award for the best Computer Science and Engineering Textbook of 1999.**
ISBN 0-13 4;69017-0

OPERATING SYSTEMS, FOURTH EDITION

A state-of-the art survey of operating system principles. Covers fundamental technology as well as contemporary design issues, such as threads, mierokernels, SIVIPs, real-time systems, multiprocessor scheduling, distributed systems, clusters, security, and object-oriented design. **Third edition received the TAA award for the best Computer Science and Engineering Textbook of 1998.** ISBN 0-13-03199q41

HIGI I-SPEED NETWORKS AND INTERNETS, SECOND EDITION

A state-of-the art survey of high-speed networks, Topics covered include 'MP congestion control. ATM traffic management, internel traffic management. differentiated and integrated services, interne(routing protocols and multicast routing protocols, resource reservation and RSVP, and lossless and bossy compression. Examines important topic of self-similar data traffic. ISBN 0-13413221-0

Prentice Hall ww•.prenhall.com/stallings telephone: /00-52h-0485

AND DATA COMMUNICATIONS TECHNOLOGY

WIRELESS COMMUNICATIONS AND NETWORKS

A comprehensive, slate-of-the art survey. Covers fundamental wireless communications topics, including antennas Lind propagation. signal encoding techniques, spread spectrum, and error correclion techniques. Examines satellite, cellular, wireless local loop networks and wireless LANs, including Biuetooth and 144.12.1L Covers Mobile IP and 'NAP. ISBN 0-13-1:W864-6

LOCAL AND METROPOLITAN AREA NETWORKS, SIXTH EDITION

An in-depth presentation of I he technology and architecture or local and metropolitan area network, Covers topology, transinksion media, medium access control, standards, internctworking. Vinci notwork management. Provides an up-to-date coverage of LAN•MAN systems, including I.ast Ethernet. Fibre Channel, and wireless LANs, plus LAN QoS. **Received the 2001 TAA award tor long-term excellence in a Computer Science Texthimk.** ISBN 0-i:1-012939-9

ISDN AND BROADBAND ISDN, WITH FRAME RELAY AND ATM: FOURTH EDITION

An in-depth presentation of the technology and architecture of integrated services digital networks (ISDN). Covers the integrated digital network (I I)N), x.DSL, ISDN services and architecture. and signaling system no. 7 (SST) and provides detailed coverage of the Ill!-'1' protocol standards. Also provides detailed coverage of protocols **and** congestion control strategies for both frame relay and ATM. ISBN 0;13-973744-5

BUSINESS DATA COMMUNICATIONS, FOURTH EDITION

A comprehensive presentation of data communications and telecommunication from a business, perspective. Covers voice, data. image, and video communications and applications technology and includes a number of case studies. ISBN 0-1:;-088263-1

NETWORK SECURITY ESSENTIALS

A lulorial and survey on network security technology. The book covers important network security tools and applications, including WIMP, IP Security. Kerberos. SSL1TLS, SET, and X509v3. **In** addition, methods for countering hackers and viruses are explored. ISBN t1-13-016093-8

COMPUTER ORGANIZATION AND ARCHITECTURE

Designing fin' Performance

SIXTH EDITION

William Stallings

Prentice Hall

Pearson Education International

Th ,1411.1100 TtIny I Id ad rill i, III 11 0,:: LX1L111114::* to which !, C.:111!%1F.(1.Ak hy ..ki wnti.11) lintoTriA II{ OM I,
EI IS M 1 11 ;11111 II is 171II fir ....Ali. in I ho .{11;(),

Vice l[3]feuidciIt and Editorial Director. EC'S: !Weida .1. I f orton
k'tthli{li r: 1?4+0 X. Ape
Managa: d'ecke erreir
Ass[]Ui:Ue r 1.5. fledor
P.cli1031.11 Ass.iistanr:
Vice ['resident alud I.nrcci{rr 01 PT4FaLtr.'111111 4trid ManufacturingfESM: Derthic W. Ric:eon/4'
Executive Managing Editor: Vim O'Brien
Arir..istant Ma nafzing. Edite Ginn:21f. Tr p.macovw
PrOdiJeCiinl. Edilut: ROM' ***KPreleet?***
Dif OCLCIr {)[Cr eati'vo Services: *Parr! nelfarret*
C:reative **DifeCLOr: Curr)h.** Arisen
An Director: firithee CeirLyn'emeii
At Editor; *Greg Dudes.*
Coyer Designer: firohti.
Manutieci uring *Ttud y Pi.xciotti*
Mienurael wring Huiyer: *bokveU*
Senior Mu rk MOIINgC17 *iorteie*

7 224 8 •
C.' 642.sTri
2003

21103 1'5l.. Pearson Education, Enc.
Saddle River. New .Teirsq 1' 1715ff

All right reser...ell. Nei pan c.).1 this ha wk may be reprelalLiCefl, lorm or Fry :illy 11':ens.,
without permission in **Writing; 1** 1'151:US the publisher.

The I oLhof and pfiblis her of this hoc 5k have useit their best efloris in preparing this hoc 5k. '1 9h12S12 **1217urk** include the development. **fesuaieil, arid** I eriaing tof the thee rrit'S. and pre igra m s 1.4F determine Muir einem **ive** ness.. The fi utile ir and publisher makeato wa rfart .? cif am kind. expressed or implied, with repard in these prop uins or the documental 131)111 i cd rn (.his book. he autlfor.and **rlilh** I ISh12T Sh I] **DAt he liahl.12** SI y Yu'.11[I ot incidental 131 LUISS11.11.1ohE **iHrn0Pc**[4] ill cdn ncction with, ine 7rising uul of. the furnishing. performance, or use. of /hove pi ciAra ms.

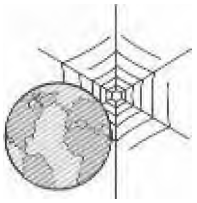
Prfniied in thy: /Jailed States cll America

10 7 6' 5 4 3. 2 I

ISBN I-13-17149307-4

PiYarSull EdLiciition
rcarson Education Australia **PIN.** Lirniied
Pearson Education Siiwapore. Pte. Lad.
}'Carson Educalroii North Asia Da
Peiirsein Eclucaiinr Cat c!a. Inc.
Pearson Ed ticaiefo n de Mexico, S.A. de C..V.
Yen rsem Pdilealion—,11;ipfin
Penrson Fclucatinn Malaysia, Pie. Ltd.
armin Eclucalion, tipiwiNewirdif. Nod! Lice,p..,tiy

As cilways
For A. T. S.



WEB SITE FOR COMPUTER ORGANIZATION AND ARCHITECTURE

Sixth Edition

The Web site at WilliamStallings.conA:0A6c.htral provides support for instructors and SALAtTILS using the book. It includes the elernunVi-

Course Support Materials

The course support materials inciudc

- Copies of figures from the book in PDF forma L
- Copies of tables from thu book in PDF format
- A set of PowerPoint slides for use as lecture aids
- A set of PDF course noit;.s suitable for student handout or for use 41.S viewgrLiphs
- Computer Science Student Resource Si1r2! contains a number of]links and documents chat students may find useNI in their ongoing computer science education. The site inciudesx review of basic_ rele '...ant mathematics; adviee on research, writing, and doing homework problems; links to computer science research resources, such as report repositories and bibliographies; and o[11,2r useful links
- An errata shed I:or the book, updated at most monthly

COA Courses

I he COA.5e Wet) s Le includes]links to Web sites for eours.cs mught using the book. These sites can provide lawful ideas about scheduling and topic ordering, as well a s a number of useful handouts and other matcr 41 Es.

Useful Web Sites

The COAfic Web itc includes links to relevant Web sites. **The** links cover a broad spectrum Of iopics and will enable students to explore timely issues in greater depth;

Internet Mailing List

An Internet mailing list is maintained so that instructors using this book can exchange information, sugge4...lions, and qUO;i1i0m. with each other and the ittLOOT. Subscription information is provided at the book's Web sill:.

Simulation Tools for CA Projects

'ihe Web site includes links to the SintpkSo kir and WPC:ache Web .rites. These are two software packages that **serve** is frame-works for project iniplerncrita1ion. Each site includes downloadable software and background information. See Appendix C for more inrOrMatiOn.

CONTENTS

Web Site for *Computer Organization and Architecture, Sixth Edition* vi
Preface xv
About the Author xxi

PART ONE OVERVIEW 1

CHAPTER 1 Introduction 3
1.1 Organizaticiii ind Arclitual:tre 4
1.2 Structure. and Function 5
1.3 Why study Computer OrAani.?..a.tion :and rch I e:et ate? 10
1.4 OLE.0110 ul 1i3C: Book LI
1.5 interne,t and Web Resourccs

CHAPTER 2 Computer Evolution and Performance 15
2.1 A Bri(.21 HiNuor.,! of Computers 1(i
2.2 Designing ror Perform 77
2.3 Pentium Rod PowerPC Evolinion 41
2.4 Rucommendcd Re.ading and Web Sits 44
2.5 K.c:y k.eview. Ouestion8, and Problems 45

PART TWO THE COMPUTER SYSTEM 47

CHAPTER 3 A '.17 op-Levei View of Computer Function and Interconnection 49
3.3 Computer Components 5(1
3.2 Computer Function 53
3.3 Interconnection Structiii.s
3.4 Bus Interconnection 69
3.3 P('1 79
3.6 Reconimendc4 Reading and Well Sites 89
3.7 Key Terms, Review Quic,tionz;., 4nd Problems 90
Appendix 3A: Timinr2. Diagrams 92

CHAPTER 4 Cache Memory 95
4.1 Computer Memory SysLarn Overvivw 96
4.2. Cache Memory Principles. 1{13
4.3 deE1enIN of CEIChC DOSigri. 106
4.4 Poruium 4 rind PowErPC Cache Organizations 121
4,5 Recommended Rc2adirtg 125
Kcy Terms, Review Questions. and Problcins 125
Appendix 4A; Puu1'orm41].}ce. Characteristics of Two-Le•el N1,-,..ral.nes 128

CHAPTER 3 Internal Memory 137

5,1 Semiconductor Main Memory 138
52 Error Correction 148
5.3 Advanced DRAM Organization 154
5,4 Recommended Reading and Web Sites 159
55 Kcy Terms, Review Questions. and 2m, 16)

CHAPTER 6 Exteenal **Memory 163**

6,1 Magnetic Disk 164
6.2 R,A.II3 174
6.3 Opticai Memory 184
6.4 Magnetic Tape 189
6.5 Reconirncnded Rcading and Web Sites 191
6:6 Key Terms, Review Questions, d Problems 192

CHAPTER 7 Input/Output 195

7.1 Exlernal Devices 197
110 Modules 201
7.3 Programmed I/O 204
7.4 Interrupt-Driven I.10 2.08
7.5 Direct Memory Aecuss 216
7.6 I/O Channels and Processors 220
7.7 The External Interface: FireWirc kind InfiniBand 223
7.8 Recommended Reading and Web Sites 233
7,9 Key '1'crTns, Review Questions, and Problems 233

CHAPTER 8 Operating System Support 237

8.1 Operating System Overview 28
8.2 Scheduling 250
8.3 Memory Management 256
8,4 Pentium 11 and PowerPC Memory Management **269**
8.5 Recommended Reading and Web Sites 277
8,6 Kcy 'Ferms, Revicw Questions, and Problems **278**

PART THREE THE CENTRAL PROCESSING UNIT 281

CHAPTER 9 Computer Arithmetic 283

9.1 The Ariihrne(ie and Logic Unit 284
2 Integer Representation 285
9.3 Integer Arithmetic 29]
9.4 Flooring-Pin Representation 307
9.5 Floating-Point Arithmetic 333
9.6 Recommended Reading and Web Sites 324
9.7 Key Terms, Review Questions, and Problems 325

CHAPTER 10 Instruction Sets: Characteristics and Functions 329

Machine Instruction Characteristics 330
10.2 Types or Operands 337
10.3 Pentium anti PocketPC Data Types 339
10.4 Types of Opor;.iiions 341
10.5 Pentium and PowerPC Operalion Types 355
10.6 Asscrnbty Language. 364
10.7 Recommended Reading 366
111.8 Key Terms, Review Questions, and Problems 360
Appendix I0A¸ Stacks 371
Appendix 10H.: Little-, Big-, and Bi-Eridian 376

CHAPTER 11 Instruction Sets: Addressing Modes and Formats 381

11.1 Addressing 382
11,2 Pentium 11˙1(1 PowerPC Addressing Modes 359
11.3 Instruction Formats 395
11.4 Pentium and PowerPC Instruction Formats 404
11.5 Recommended Rending 408
11.6 Key 'Terms, Review Questions, rand Problems 409

CHAPTER 12 CPU Structure and Function 411

12.1 Processor Organization 4[2
12.2 Register Organization 414
12.3 Instruction Cycle 420
12.4 Instruction Pipelining 424
12.5 'The Pentium Processor 440
12.6 The PowerPC Processor 430
12,7 Recommended Reading 457
12.H Key Terms. Review Questions, and Problems 458

CHAPTER 13 Reduced Instruction Set Computers 461

13.1 Instruction 1-:Necution Characteristics 463
13.2 The Use of a Largc Register Fite 467
13.3 Compiler-Based Register Optimization 473
13.4 Reduced Instruction Set Areill Cod 'Luc. 474
13.5 RISC Pipelining 482
13.6 MI PS 84000 486
13.7 SPARC 494
13.8 RISC versus C1SC: Controversy 500
13.9 Recommended Reading 501
13.10 Key Turns, Review Questions, and Problems 502

CHAPTER 14 Instruction-Level Parallelism and Superscalar Processors 5O

14.1 Overview 507
14.2 Design Issues 511
14.3 Pentium 4 520
14.4 PowerPC 527
14.5 Recommended Reading 535
14.6 Key Terms, Review Questions, and Problems 536

CHAPTER 15 The 1A-64 Architecture 541

15.1 Motivation 543
15.2 General Organization 544
15.3 Predication, Speculation, and Software Pipelining 546
15.4 1A-64 Instruction Set Architecture 563
15.5 Itanium Organization 568
15.6 Recommended Reading and Web Sites 569
15.7 Key Terms, Review Questions, and Problems 570

PART FOUR. THE CONTROL UNIT 573

CHAPTER 16 Control Unit Operation 575

161 Micro-Operations 577
16.2 Control of the Processor 583
16.3 Hardwired Implementation 594
16.4 Recommended Reading 597
16.5 Key Terms, Review Questions, and Problems 597

CHAPTER 17 Microprogrammed Control 599

17.1 Basic Concepts 600
17.2 Microinstruction Sequencing 609
17.3 Microinstruction Execution 615
17.4 1 1 SW} 627
17.5 Applications of Microprogramming 637
17.6 Recommended Reading 638
17.7 Key Terms. Review Questions. and Problems 639

PART FIVE PARALLEL ORGANIZATION 641

CHAPTER 18 Parallel Processing 643

18.1 rdultiple Processor Organizations 645
18.2 Symmetric Multiprocessors 647
18.3 Cache Coherence and the MESI Protocol 656
18.4 Clusters 663
18.5 Nonuniform Memory Access 670
18.6 Vector Computation 674
18.7 Recommended Reading 687
18,8 Key Terms, Review Questions, and Problems 688

APPENDICES

APPENDIX A Digita/ Logic 693

A.1 Boolean Algc4-Fr4i 694
A.2 Gates 696
.A1 Corn binational Circuits 699
A.4 Sequential Circuits 720
A,5 Problems 7:O

APPENDIX B Number **Systems 733**

B.I. The Decirrml Sp.licui 734
11,2 The Binary System 734
B..3 Converting between Binary and Deein-A 7.3.5
BA I lexackeirmil Notation 73
13.5 Problems 739

APPENDIX **C** Projects for Teaching Computer **Organization** and Architectute 741

C.1 Re.7',(2. 1rat Projects 742
C.2 Simulation Projects 742
C.3 Reading/Report Assignments **743**

GLOSSARY 745

REFERENCES 757

INDEX 773

PECEFACE

bOok is about the structure and function of computers. Its purpose is to present. as clearly and completely as possible, the nature and characteristics of modern-day computer systems.

rhis task is challenging for several re, sans. l first, there is a tremendous variety of products that can rightly claim the name of computer, from single-chip microprocessors costing a tcw dollars to supercomputers costing tens of millions of dollars. Variety is exhibited not only in cost, but in size. performance, and .application. Second, the rapid pace of change that has always characterized computer technology continues with no letup. These changes cover all aspects of computer technology, from the underlying integrated circuit technology used to construct computer components, to the increasing use of parallel organization concepts in combining those components.

In spite of the ', ariety and pace of change in the computer field. certain fundamental concepts apply consistently throughout. The application of these concepts depends on the current state of the technology and the pricelperformanc.e objectives of the designer. The intent of this hook is to provide a thorough discussion of the fundamentals of computer organization and architecture and to relate these to contemporary design issues.

The subtitle suggests the theme. and the approach taken in this book. It has always been important to design computer systems to achieve- high performance, but never has this requirement been stronger or more difficult to satisfy than today. All of the basic performance characteristics of computer systems, including processor speed, memory speed, memory capacity, and interconnection data rates, are increasing rapidly. Moreover• they are increasing ait different rates. This makes it difficult to desiv,rn a balanced system that maximizes the performance and utilization of all elements. 'Thus, computer design increasingly becomes a game of changing the structure or function in one area to compensate for a performance mismatch in another area. We will see this game played out in numerous design decisions through. out the book.

A computer system, like any syStem, consists of an interrelated set of components. The system is best characterized in terms of structure—the way in which components are interconnected—and function—the operation of the individual components. Furthermore, a computer's organization is hierarchical. Each major component can be further described by decomposing it into its major subcomponents and describing their structure and function. For clarity and ease of understanding, this hierarchical organization is described in this hook from the top down:

- **Computer System:** Major components are processor. memory. and 1/0.
- Processor. Major components are control unit. register, A1.1), and instruction execution unit.

  Control Unit: Major components are control memory, microinstruction sequencing logic, and registers.

The objective is to present the material in a fashion that keeps new material in a clear context. This should minimize the chance that the reader will get lost and should provide better motivation than a bottom-up approach.

Throughout the discussion, aspects of the system are viewed from the points of view of both architecture (those attributes of a system visible to a machine language programmer) and organization (the operational units and their interconnections that realize the architecture).

EXAMPLE SYSTEMS

'Phis hook uses examples from a number of different machines to clarify and reinforce the concepts being presented. Many, but by no means all, of the examples are drawn from two computer families: the Intel Pentium 4, and the IBWMotorola PowerPC. These two systems together encompass most of the current computer design trends. The Pentium 4 is essentially a complex instruction set computer (CISC) with some RISC Features. while the PowerPC is essentially a reduced instruction set computer (RISC). Both systems make use of superscalar design principles and both support multiple processor configurations.

PLAN OF THE TEXT

The book is organized into live parts:

Part One—Overview: This part provides a preview and context for **the** remainder of the book.

Part Two—The Computer System: A computer system consists of processor, memory, and 110 modules. plus the interconnections among these major components. With the exception of the processor, which is sufficiently complex to he explored in Part Three. this part examines each of these elements in turn.

Part Three—The **Central Processing Unit:** The CPU consists of a control unit, reaisters, the wirhrricikl and logic unit, the instruction execution unit, and the interconnections among these components. Architectural issues, such as i nstruelion sot design and data types, are covered, Part Three also]ooks at orLianiy.ational issues, such as pipelining.

Part Four—The Control Unit: The control unit is that part of the processor that aciivales the various components of the processor. This part looks at the functioning of the control unil and its implementation using microprogramming.

Part Five—Parallel Organization: This final part]ooks at some of the issues involved in rnuiiiple processor mid vector processing organizations.

The book also includes an extensive glossary. a list of frequcntiv used acronyms, and a bibliography. Each chapter includes homework problems, review questions, a list of key words, suggestions for further reading, and recommended Web sites.

A more detailed, chapter-by-chapter summary of each part appears at the beginning of lhaL part,

INTENDED AUDIENCE

The hook is intended for both an academic and a professional audience. As a textbook, it is intended as a one- or two-semester undergraduate course for computer science, computer engineering, and electrical engineering majors. It covers all the topic-5 in *CS 220 Computer .0 00f:titre, which* is one of the core. subject areas in the *EE ErA CM Cr ɨmputer Cr ricrila 2001 PTFOG*

For the professional interested in this field, the hook serves as a basic reference volume and is suitable for self-study.

INTERNET SERVICES FOR INSTRUCTORS AND STUDENTS

There is a Web site for this book that provides support for students and insiruetors. i he wile includes **links to** other relevant sites, copies of the figures and tables from the book in Pflb (Adobe Acrobat) format, and sign-up information for the book's Internet mailing list. The Web page is 11 WilliamS1allings,eonlICO Me.h1rnl: see the section, ¯Web Site for Computer 'Organizaticni and Architecture, Sixth Edition", preceding [his Preface, for more information. An Internet mailing list has been set up so that instructors using this book can exchange information, suggestions, and questions with each other and with the author. As soon as typos or olher errors are discovered. an errata list for this book will be available at hiamStallings.com . In addition, the Computer Science Student Resource site, at WiiliamStallings,corn/StudentSupport.htud, provides dociimun is, information, and useful links for computer science students and professiona]s.

PROJECTS FOR TEACHING COMPUTER ORGANIZATION AND ARCHITECTURE

For army instructors, an important component of a computer organization and architecture course is a project or set of projects by which the student gets hands-on experience to reinforce concepts from the text. This book provides an unparalleled degree of support for including a projects component in the course- The instructor's marmil not **only** includes guidance on how to assign and structure the projects, but also includes a set of suggested projects that covers a broad range of topics from the text:

- Research projects; The manual includes ri series of assignments that instruct the student to research a particular topic on the Web or in the literature, 4md write a report.
- **Simulation projects:** The manual provides support for the use of the two simulation packages: SimplcScalar can be used to explore computer organization and architecture design issues. SkIPCache provides a powerful educational tool for examining cache design issues for symmetric multiprocessors.
- **Rcadiogireport assignotents: The** manual includes a list of papers in the literature. one or more for each chapter, that can be assigned for the student to read and then write a **short** rcport'

See Appendix C for details.

WHAT'S NEW TN THE SIXTH EDITION

In the three years since the fifth edition of this book was published, the field has seen continued innovations and improvements. In this new edition, I try to capture these changes while maintaining li broad and comprehensive coverage of the entire field. To begin this process of revision, the fifth edition of this book was extensively reviewed by a number of professors **who** reach the ;40 0. In addition, a number of professionals working in the field reviewed individual chapters. The resell is that, in many plac.i.:27., the narrative has been clarified find tightened, and illustrations have been improved. Also, a number of new ''field-tested' problems have been added.

Beyond these refinements to improve pedagogy and **user** friendliness, there have been substantive changes throughout the book. Roughly the M.411TIC chapter organization has been retained, but much of the material has been revised and new material has been added. Some of the most noteworthy changes are the following:

- l**A-64/I11inium architecture: This** new architecture includes such important Concepts as predicated execution and speculal ive loading. 7 Ills edition features a chapter-length description and analysis.

- Cache memory': Cache memory is a central element in the design of high-performance processors, and cache **detiign** has become increasingly complex. An entire chapter is devoted to this issue in the new edition.
- Optical memory: 'the material on optical memory has been expanded and updated.
- Advanced I**MAM architecture: More** material has been added **to** cover this topic, including an updated discussion of SDRAM and RDRAM.
- **SMPK, clusters. and NUMA systems:** The chapter on parallel organization has been expanded and updated.
- **Expanded instructor support:** As mentioned previously, the book **now** provides extensive support for projects. Support provided by the book Web site has also been expanded,

ACKNOWLEDGMENTS

This new edition has benefited from review by a number ill people, **who** gave gen erously of their time and expertise. .1.'he following people reviewed all **Or** a large part of the manuscript: Willis King (University of I louston), Albert Heaney (California State University), A. S. Pandya (Florida Atlantic University). Yaser Khalifa (University of North Dakota), and Sanjecv Baskiyar (Auburn University).

Thanks also to the many people who provided detailed technical reviews of a single chapter: Nicole Kaiyan, Terje Mathisen, Daniel M. Pressel, Jeff Deifik, Bill Todd. Charlie Cassidy, Andy Isaacson, Alex Potemkin, Michael Spratte. Hatem Yassine. Grzegorz Mazur, Alan. Leholsky. Jonathan Hall. Sophie Wilson, Alan Alexander, David Vickers. Pete. Smoot, and Erik Seligman.

Professor Cindy Norris of Appalachian State University contributed some homework problems.

Professor Miguel Angel Vega Rodriguez, Prof, Dr. Juan Manuel SArichez P6re.e., and Prof. Dr. Juan Antonio Gomez Pulido, all of University of Extremadura, Spain prepared the SMPCache problems **in** the instructors manual and authored the SMPCache User's Guide.

Bezenek of the University of Wisconsin and Janes Stine of Lehigh University prepared I he SimpleScalar problems in the instructors manual, and Todd also authored the SimpleScalar User's Guide.

ABOUT THE AUTHOR

WI L..1..J.ANI ST.ALLINGS has made a unique contribution to understanding the broad sweep of Icchnictil developments in computer networking and computer architecture. l.ic has E3tithored 1.7 tiller, **Lin.d.eounting** revised edi-lions, a total of .35 books on various .:]ir pecits. ol' these subjc.ets..1-'or live vt.7,an, **ill ET row..** he has; been the•recipient of the award for the hest Computer Science and Prigineering t.c:01),..),..pk of the year from the Textbook **kind** Academic Authors Association.

In over 2 years in the field. Dr. Stallings lias heel' a technical contrib-LItor,,. technical **manager** and an executive. with several high-technology firms. He is an independcill cor.isuhant whose clients have included computer **Ei nd** networking rrizmufacturers and custorners.. s.oftm...;ii.c devel opmetit firm..., and leading.4.2dge government research institutions, He: created and maintains the Computer Scicficc Student 1-Z. ...source Site it

i l I La m St. a I tin gs.eorniS I 0 de n i S u pport .131m.l.

1)r, Stallings holds a Ph.D. from MIT in computer science. and a 1!-S. From Notrj.:: .r..)arn.p. in electrjçal engineer...

PART ONE Overview

ISSUES FOR PART ONE

The purpose of Part One is to provide a background and context for the remainder of this book, The fundamental concepts of comp u ter organization . and architecture are presented.

Chapter 1 Introduction

Chapter I introduces the concept of the computer as a hierarchical system. A computer can be viewed as a structure of components and its function described in terms of the collective function of its cooperating components. **Cach** componeni, in turn, can be described in terms of its internal structure and function. The major levels of this hierarchical view arc introduced. The remainder of the. book is organized, top down, using these levels,

Chapter 2 Computer Evolution and Performance

Chapter 2. serves two purposes. First, a discussion of the history of computer technology is an easy and interesting way of being introduced to the basic concepts of computer organization and architecture. The chapter also addresses the technology trends that have made performance the focus of computer system design and previews the various techniques and strategies that are used to achieve balanced, efficient performance.

CHAPTER

INTRODUCTION

1.1 Organization and Ambitii.eture

1.2 Structure and Function

Function
.StructuDe

1.3 Why Study Computer Organization and Architecture?

1.4 Outline AeBook

1.5 Internet and VVreb Resources

Web Sites rot This Book
Other Web Sites
US EN F.:f Newsgroups

This book is about the structure and function of computers. Its purpose is to present, as clearly and completely as possible, the nature and characteristics of modern-day computers. This task is a challenging one for two reasons.

First, there is a tremendous variety of products, from single-chip microcomputers costing a few dollars to supercomputers costing tens of millions of dollars, that can rightly claim the name *computer.* Variety is exhibited not only in cost, but also in size, performance, and application. Second. the rapid pace of change that has always characterized computer technology continues with no letup. These changes cover all aspects of computer technology, from the underlying integrated circuit technology used to construct computer components to the increasing use of parallel organization concepts in combining those components.

In spite of the variety and pace of change in the computer field. certain fundamental concepts apply consistently throughout. To be sure, the application of these concepts depends on the current state of technology and the price/performance objectives of the designer. The intent of this book is to provide a thorough discussion of the fundamentals of computer organization and architecture and to relate these to contemporary computer design issues. This chapter introduces the descriptive approach to be taken and provides an overview of the remainder of the book.

1.1 ORGANIZATION AND ARCHITECTURE

In describing computers, a distinction is often made between *computer architecture* and *computer organization.* Although it is difficult to give precise definitions for these terms. a consensus exists about the general areas covered by each (e.g., see [VRANNOI. [SIEW82], and IBELL78a]).

Computer architecture refers to those attributes of a system visible to a programmer or, put another way those attributes that have a direct impact on the logical execution of a program. Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory. Organizational attributes include those hardware details transparent to the programmer, such as control signals, interfaces between the computer and peripherals, and the memory technology used.

As an example. it is an architectural design issue whether a computer will have a multiply instruction. It is an organizational issue. whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system. The organizational decision may be based on the anticipated frequency of use of the multiply instruction, the relative speed of the two approaches, and the cost and physical size of a special multiply unit,

Historically, and still today, the distinction between architecture. and organization has been an important one, Many computer manufacturers offer a family of computer models, all with the same architecture but with differences in organization. Consequently, the different models in the family have different. price and performance characteristics. Furthermore, a particular architecture may span many years and encompass a number of different computer models. its organization

changing with changing technology. A prominent example of both these phenomena is the IBM System/370 architecture. This architecture first introduced in 1970 and included a number of models. The customer with modest requirements could buy a cheaper, slower model and, if demand increased, later upgrade to a more expensive. faster model without having to abandon software that had already been developed. Over the years, IBM has introduced many new models with improved technology to replace older models, offering the customer greater speed, lower cost, or both. These newer models retained the same architecture so that the customers software investment was protected. Remarkably. the Systemi370 architecture, with a few enhancements. has survived to this day as the architecture of IBM's mainframe product line.

In a class of computers called microcomputers, the relationship between architecture and organization is very close. Changes in technology not only influence organization but also result in the introduction of more powerful and more complex architectures. Generally, there is less of a requirement for generation-to-generation compatibility for these smaller machines. Thus, there is more interplay between organizational and architectural design decisions. An intriguing example of this is the reduced instruction set computer (RISO, which we examine in Chapter 12.

This book examines both computer organization and computer architecture_ The emphasis is perhaps more on the: side of organization. I lowever, because a computer organization must be designed to implement a particular architectural specification, a thorough treatment of organization requires a detailed examination of architecture as well

1.2 STRUCTURE AND FUNCTION

A computer is a complex system; contemporary computers contain millions of elementary electronic components. How, then. can one clearly describe them? The key is to recognize the hierarchical nature of most complex systems, including the computer [SIM069]. A hierarchical system is a set of interrelated subsystems, each of the latter, in turn, hierarchical in structure until we reach some lowest level of elementary subsystem.

The hierarchical nature of complex systems is essential to both their design and their description. The designer need only deal with a particular level of the system at a time. At each level, the system consists of a set of components **and** their interrelationships. The behavior at.each level depends only on a simplified, abstracted characterization of the system at the next lower level, Al each level, the designer is concerned with **Structure** and function:

- Structure: The way in which the components are interrelated
- **Function:** The operation of each individual component as part of the structure

In terms of description. we have two choices: starting at the bottom and building up to a complete description, or beginning with a top view and decomposing the system into its subparts. Evidence from a number of fields suggests that the top-down approach is the clearest and most effective [WEIN75].

The approach taken in this book follows from this viewpoint. The computer system be duscribed from the top down. We begin with the major components of a computeY, describing their structure and function, mid proceed to successively tower laycN of the hierarchy. The remainder of this fection provides a very brief overview of this plan of attack.

F unc tion

Roth the structure and functioning of a computer are, in essence, simple, Figure 1.1 depicts the basic functions thal a computer can perform. In general term, there are on4' four:

- Data processing
- Data storage
- Data movement
- C0111-rtg

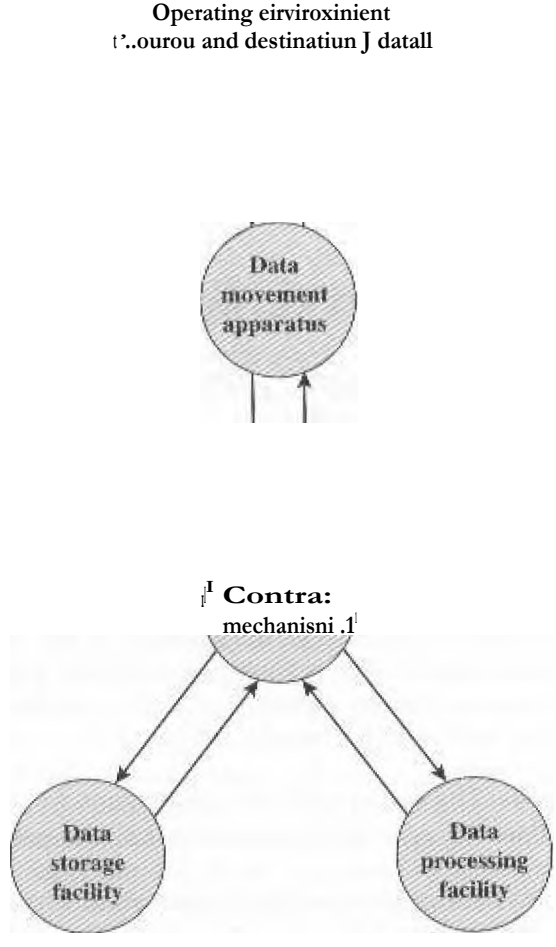


Figure Li A Functional Victv of thc. Computer

The computer, of course, must be able to *process data,* The data may take a wide variety of forms, and the range of processing requiretnents is broad. However, we shall see that there are Only a few fundamental methods or types of data processing.

It is also essential that *computer store do* ur. Even it' the computer is processing data on the fly data come in and get processed, and the results go out immediately), the computer must temporarily store at least those pieces of data that are being worked on at any given moment. Thus, there is at least a short-term data storage function. Equally important, the computer performs a long-term data storage function. Files of data are stored on the computer for subsequent retrieval and update.

The computer must he able to *move data* between itself and the outside world. The computer's operating environment consists of devices that serve as either sources or destinations of data. When data are received from or delivered to a device that is directly connected to the computer, the process is known *as inpur-ourPlit (1r'O). and* the device is referred to as *a perfpheral.* When data arc moved over longer distances, to or from a remote device, the process is known as *data commanications,*

Finally, there. must be *control* of these three functions. Ultimately, this control is exercised by the individual(s) who provides the computer with instructions. Within the computer, a control unit manages the computer's resources and orchestrates the performance of its functional parts in response to those instructions.

Al this general level of discussion, the number of possible operations that can be performed is few. Figure 1.2 depicts the four possible types of operations. The computer can function as a data movement device t Figure 1.2a), simply transferring data from one peripheral or communications line to another. It can also function as a data storage device (Figure 1.21)), with data transferred from the external environment to computer storage (read) and vice versa (write). The final two diagrams show operations involving data processing, on data either in storage (Figure 1.2e) or en route between storage and the external environment (Figure 1,2d),

The preceding discussion may seem absurdly generalized, it is certainly possible, even at a top level of computer structure, to differentiate a variety of functions, but, to quote 1SIEW821,

> There is remarkably little shaping of computer structure to fit the function to be performed. At the root of this lies **the** general-purpose nature of computers, in which all the functional specialization occurs at the time of programming and not at the time of design.

Structure

Figure 1.3 is the simplest possible depiction of a computer. The computer interacts in some fashion with its external environment, In general, all of its linkages to the external environment can he classified as peripheral devices or communication lines. We will have something to say about both types of linkages.

But of greater concern in this book is the internal structure of the computer itself, which is shown at a top level in Figure 1.4, There are four main structural components:

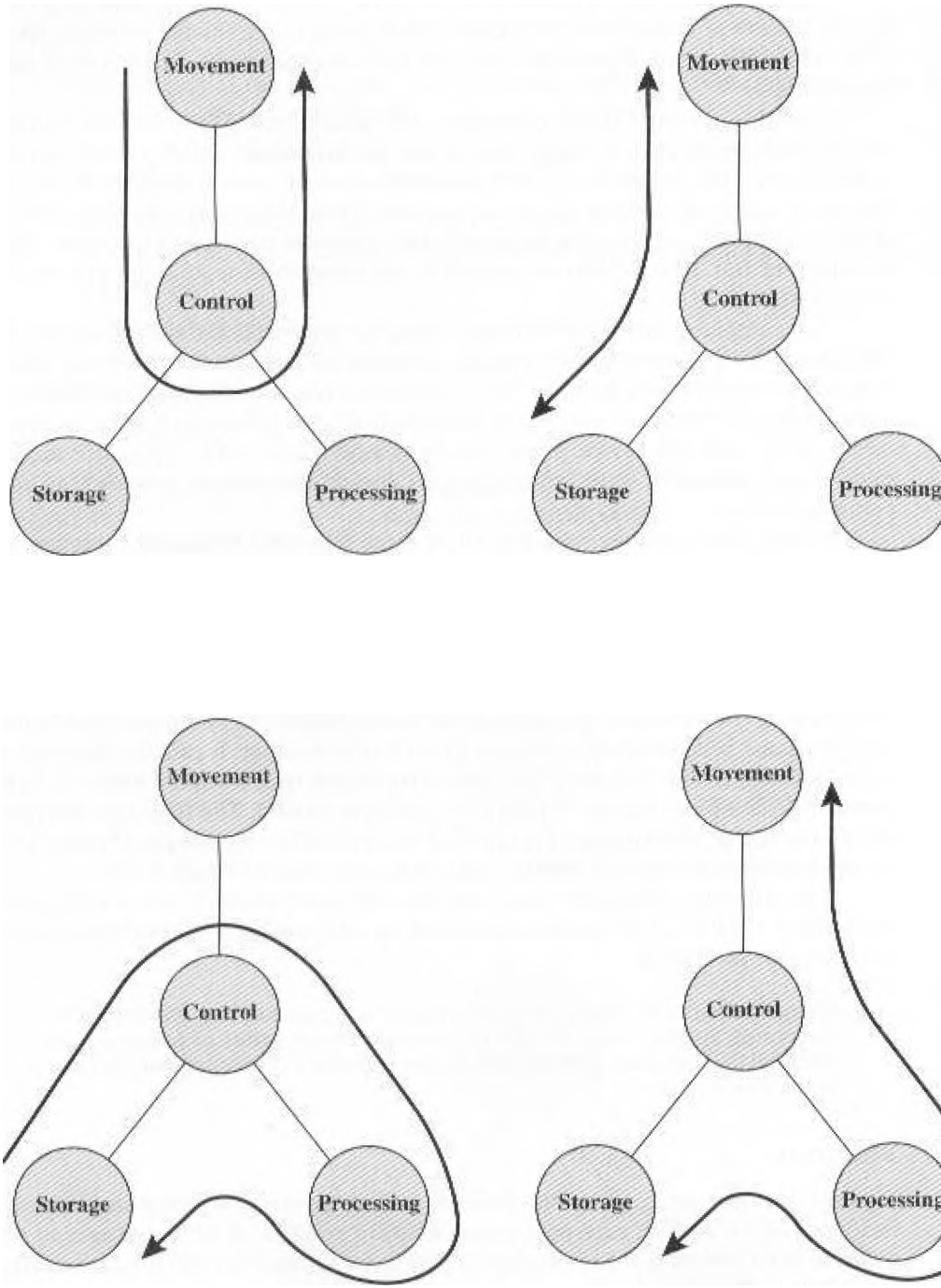
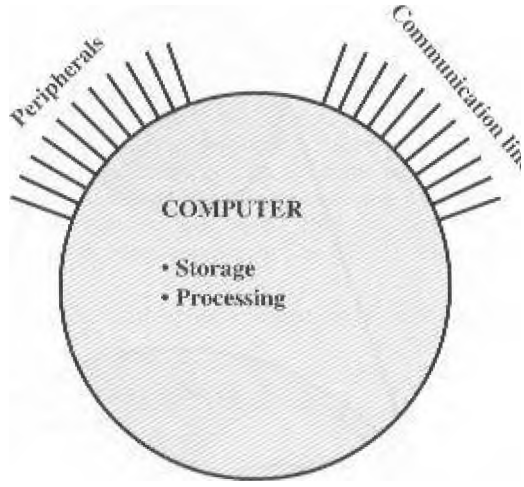


Figure 1.2 Possibl urpui r Op rtxfinns



Figark: 1.3 The Computer

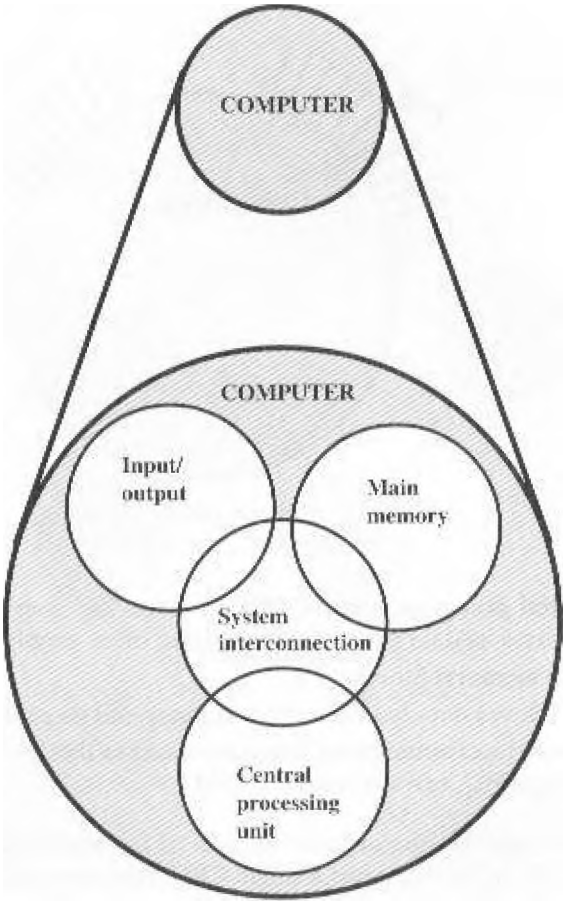
- Central processing unit (CPU): Coma)Is the operation oI the computer and performs iEs drug processing functions often simply referred lo as *procinaer*
- Main memory: Stores data
- 110: Moves daia between the computer and its external cnvironment
- System interconnection]: Some mechanism that provides for communication among CPU, rmin memory, and I/O

There may he one or more of each of the aforementioned components, Traditionally, there has htxn just a Singh: CPI:, In recent years, there has been increasing use of multiple processors in a single computer. Some design issues relating to multiple processors crop up and are discussed as the text proceeds: Part Fire focuses on such computers,

Each of these components will he examined in some detail in Pad Iwo. However, for our purposes, the most interesting and in mile ways the most complex **component** is the (:1^{3}1„: its structure is depicted in Figure 1_5. Its major structural components are as follows:

- **Control unit:** Controls the operation of the CPU and hence the computer
- **Arithmetic and logic unit (ALU): Performs the** computer's data processing funct ions
- Registers: Provides storage inl erna I to the CPU
- **CPU interconnection:** Some mechanism that provides for communication among the control unit, ALU, and registers

Each Of these components will be examined in some detail in Part Three. where we will see that complexib,,y is added by the use of parallel and pipeiined organizational techniques, Finally. l]iere arc several apprcmches to the implementation of the control unit, but the most common is a *microprogrammed* impiernentation• In essence,



Fivre 1.4 The Com1)111cr: rop-Lcvel Str LICt LL

microprogrammed ct **Introl** LIII t operates by executing microinstructions that tkriElo the functionalily of the control unit, With this approach, the structure of the control unit can be depicted as in Figure 1.6. This structure will be examined in Part Four.

1.3 WHY STUDY COMPUTER ORGANIZATION AND ARCHITECTURE?

ThQ1ELESACM Complier Curricula 200] Iii 1, prepared by the Joint Task Force on Computing Curricula of the lEEH (Institute of Electrical and Electronics Engineers) Computer Society and ACM (Association for Computing Machinery). lists computer architecture ari one of the core subjcet!i that should be in the curriculum {,r all students in computer science and computer engineering. The report says the following:

The. compuiler lies at the heart or compoling. Without it most Or the cornputjng disciplines today would be a branch of theorotical mathematics. To be a professional in any field of computing today. one sli ould not regard the computo as inst a black box That executes programs by magic. All students of computing should **acq** uire some and erstandin and appreciation of a cc.kmptiter s} stern's functional eumponents, their charact4:risties their perforinanm, and their interactions. 'Fhere are prnctical implications as well. Students aced tik understand computer iirehitecture irk order trk structure a program so that it runs moire efficiently on a real machine_ in selecting a system to usu.. Ilicy should is able. to Unilurstand the tradeoff aniung various componi nts. such as CPI! clock speed vs. ir n Mice.

lCLEN1001 givG5 the following examples w reasons for studying computer architecture:

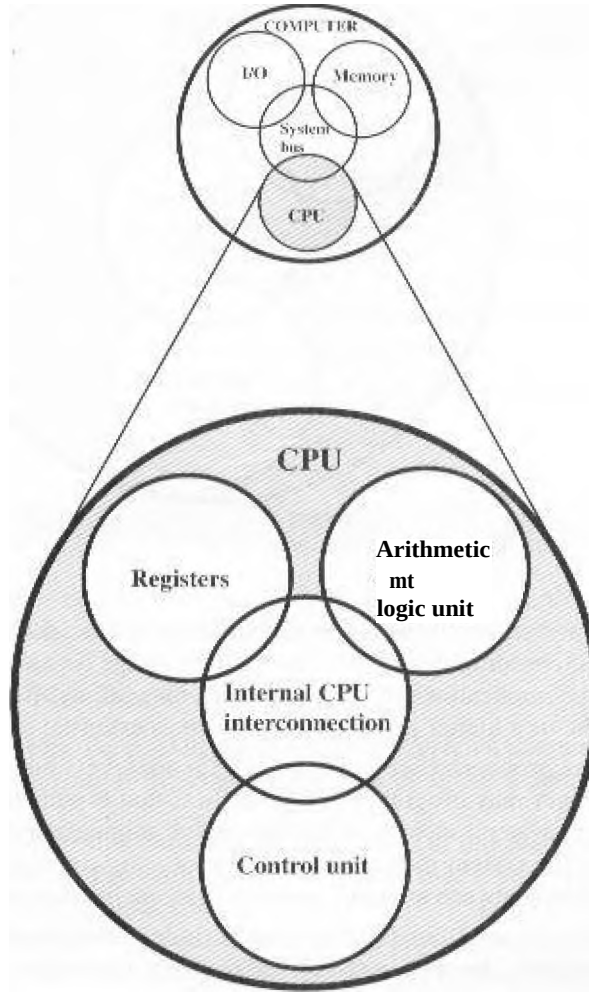


Figure 1-5 The Cendr.ill l'ruc.:c.ssing Unit (CPU)

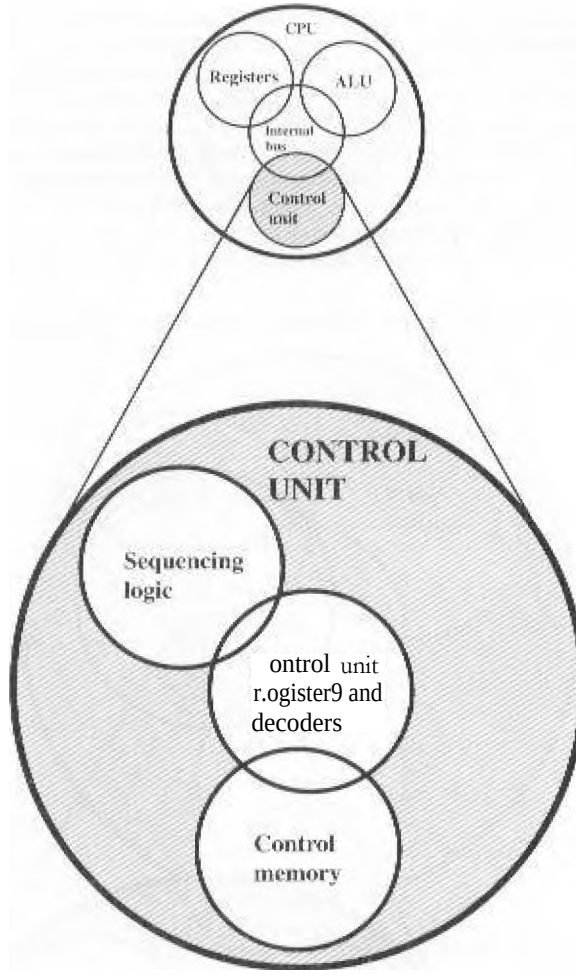


Figure l.6 Conirol Unit

1. Suppose a graduate enters the inclus..lry and is asked lo select the namt cost-effective computer for use throughout a large organization. An understanding ()I' the implications or spending more for various alternatives. such as a largo' cache or a higher processor clock rare, is essential to making the deciSion.
2. Many processors arc not used in PG or servers but in embedded systems, A designcr rria:%.' program it processor in C that is embedded in some real-time or larger system, such as 4i11 intelligent automobile electronics. controller. Debugging the system may require the use of a logic analyzer that displays tic..relationship between interrupt requests from engine sensors and machine-level code,
3. Concepts used in computer architecture find application in other courses. In particular, the way **in** which the computer provides architect ural support for programming languages and operating system facilities reinforces concepts Front those areas.

As can he seen by perusing the table of contents of this book, computer organization and architecture encompasses a broad range of design issues and concepts. A good overall understanding of these concepts will he useful both in other areas of study and in future work after graduation.

1.4 OUTLINE OF THE BOOK

The hook is organized into five parts:

Part One Provides an overview of computer organization and architecture and looks at how computer design has evolved_

Part Two: Examines the major components of a computer and their interconnections, both with each other and the outside world. This part also includes a detailed discussion of internal and external memory, and of I/O. Finally, the relationship between a computer's architecture and the operating system running on that architecture is examined.

Part Three; Examines the internal architecture and organization of the processor. This part begins with an extended discussion 0r computer arithmetic, Then we look at the instruction set architecture_ The remainder of the part deals with the structure and function **of** the processor, including a discussion of RISC and superscalar approaches, as well as a detailed look at the IA-64 architecture,

Part Four. Discusses the internal structure of the processor's control unit and the use of microprogramming.

Part Five: Deals with parallel organization, including symmetric multiprocessing and clusters_

1.3 INTERNEr AND WEB RESOURCES

There are a number of resources available on the Internet and the Web **to** support this book and to help one keep up with developments in this field.

Web Sites for This Book

A special Web page has been set up for this book at WilliarnStallings.comiCOAfie.html. See the two-page layout at the beginning of this hook for a detailed description of that site.

An errata list for this book will be maintained at the Web site and updated as needed. Please e-mail any errors that you spot to me. Errata .sheets for my other books are at WilliamStallings.com.

/ also maintain the Computer Science Student Resource Site. at WilliamStallings.comiStudentSupport,htmh. the purpose of this site is to provide documents, information, and useful links for computer science students and professionals, Links are organized into our categories:

- Math: Includes a basic math refresher, a queuing analysis primer. a number system primer, and links to useful math Web sites
- Flow-tu: Advice and guidance for solving homework problems, writing technical reports. and preparing technical presentations
- Research resources; Links to important collections of papers, technical reports, and bibliographies
- **Miscellaneous: A variety** of useful documents and links

Other Web Sites

There are numerous Web sites that provide information related la the topics of this book. In subsequent chapters. pointers to specific Web sites can be found in the "Recommended Reading and Web Sites' section. Because the URLs for Web sites tend to change frequently. I have not included these in the book. For all of the Web sites listed in the book. the appropriate link can be found at this book's Web site. Other links will be added when appropriate.

,----- The following are Web sites of general interest related to computer organization and architecture:

- **VielleVi Computer Architecture Home Page;** A comprehensive index to information relevant to computer architecture researchers, including architecture groups and projects, technical organizations, literature, employment, and commercial information
- **CPU Info Center:** Information on specific processors, including technical papers. product information, and latest announcements
- ACM Special **Interest Group on Computer Architecture: Information** on SI GA RCH activities and publications
- **IEEE Technical Committee on Computer** Architecture: Copies **of** TCAA newsletter

USENET Newsgroups

A number of US ✓NKT newsgroups are devoted to some aspect of computer organif.ation and architecture. As **with** virtually all USENET groups, there is a **high** noise to signal ratio, but it is worth experimenting to see if any meet **y**our needs. The most relevant are as follows:

- comp.arch.: A general newsgroup for discussion of computer arch iie.ctui Often quite good.
- **comp.arch.arithmetic: Discusses** computer arithmetic algorithms and standards.
- **comp.arch.storuge:** Discussion ranges from products to technology to practical usage issues
- **cump.parallcl:** Discusses parallel computers and applications.

CHAPTER

COMPUTER EVOLUTION AND PERFORMANCE

2.1 A Brief History of Computers

The First Gencraiion: Vacuum Tubes
The Second Generation: Transistors
The Third Generation: Integrated Circuits
Later Generations

2.2 Designing **for Performance**

MR ropro ce ssor Speed
Performance Balance

2.3 Pentium **and PowerPC Evolution**

PertEium
POWC: r

2.4 **Recommended Reading and Web Sites**

2$ Key Terms, Review Questions, and Problems

Key Terms
Review Questions
Pi **Phi e MS,**

KEY POINTS.

- The evolution of computers has been characterized by increasing processor speed, decreasing component size, increasin2.memory size, and increasing capacity and speed.
- One factor responsible for the great increase in proc-c.squi speed is the shrinking size of microprocessor components; this re•duces the distance between components and hence increases speed. However. the true gains in speed in recent years have come from the organization of the proeessor, including heavy use of pipelining and parallel execution techniques and the use of speculative execution techniques, which results in the tentative execution of future instructions that ruight he needed. All of these techniques arc designed to keep the processor busy as much of the time as possible.
- A critical issue. in Computer system design is balancing the performance of the various elements. so that gains in pernIrmance in one area arc not handicapped by a lag in other areas, In particular, processor speed has increased more rapidly than memory access time. A variety of techniques is used to compensate for this mismatch, including caches. wider data paths from memory to processor, zind more intelligent memory chips.

We begin our study of computers with a brief history. This history is itself interesting and also serves the purpose of providing an overview of computer structure and function. Next, we address the issue of performance, A consideration of the need for balanced utilization or computer resources provides a context that is useful throughout the hook. Finally, we look briefly at the evolution of the two systems that serve as key examples throughout the book: Pentium and Power PC,

2.1 A BRIEF HISTORY OF COMPUTERS

The First Generation: Vacuum Tubes

ENIAC

The ENIAC (Electronic Numerical Integrator And Computer), designed by and constructed under the supervision of John Mauchly and John Presper Eckert at the University Of Pennsylvania, was the world's first general-purpose electronic digital computer.

The project was a response to U.S. wartime needs during World War 11. The Army's Ballistics Research Laboratory (BRL), an agency responsible for developing range and trajectory tables for new weapons, was haying difficulty supplying these tables accurately and within a reasonable time frame. Without these firing tables, the new weapons and artillery were useless to gunners. The [IL employed

more than 200 people who, using desktop calculators, solved the necessary ballistics equations. Preparation of the tables for a single weapon would take one person many hours, even days.

Mauch a professor of electrical engineering at the[!niversity of Pennsylvania, and Eckert, one of his graduate students, proposed to build a general-purpose computer using vacuum tubes for the 13111..ˈs application. In 1943, the Army accepted this proposal, and work began on the ENIAC, The resulting machine was enormous, weighing 30 tons, occupying 1500 square feet of floor space, and containing more than 18.000 vacuum tubes_ When operating, it consumed 140 kilowatts of power. It was also substantially faster than any electromechanical computer, being capable of 5000 additions per second.

The ENIAC was a decimal rather than a binary machine. That is. numbers were represented in decimal form and arithmetic was performed in the decimal system. Its memory consisted or 20 "accumulators. ¯ each capable of holding a 10-digit decimal number. A ring of 10 vacuum tubes represented each digit. At any time, only one vacuum tube was in the ON slate, representing one of the **10** digiEs. The major drawback of the ENIAC was that it had to he programmed manually by setting switches and plugging and unplugging cables.

The ENIAC was completed in 1946, too late to he used in the war effort, Instead. its first task was to perform a series of complex calculations that were used to help determine the feasibility of the hydrogen bomb. The use of the ENIAC for a purpose other than that for which it was built demonstrated its general-purpose nature. The EN1AC continued to operate under BRL management until I 955, when it was disassembled.

The von Neumann Malithium

The task of entering and altering programs fur the EN lAC was extremely tedious. The programming process could be facilitated if the program could be represented in a form suitable for storing in memory alongside the data Then. a computer could get its instructions by reading them from memory, and a program could be set or altered by setting the values of a portion of memory.

This idea, known as the *wored prognan concept, is* usually attributed to the ENIAC designers. most notably the mathematician John von Neumann, who was a consultant on the ENIAC project, Alan Turing developed the idea at about the same time, The first publication of the idea was in a 1945 proposal by von Neumann for a new computer, the EDVAC (Electronic Discrete Variable Computer).

In 1946. von Neumann and his colleagues began the design of a new stored-program computer, referred to as the lAS computer, at the Princeton Institute for Advanced Studies. The 1.AS computer. although not completed until 1952. is the prototype of all subsequent general-purpose computers.

Figure 2.1 shows the general structure of the lAS computer. It consists of the following:

- A main memor!,/, which stores both data and instructions
- An arithmetic and logic unit (ALU) capable of operating on binary data
- A control unit, which interprets the instructions in memory and causes them to be exeeuled
- Input and output (110) equipment operated by the control unit

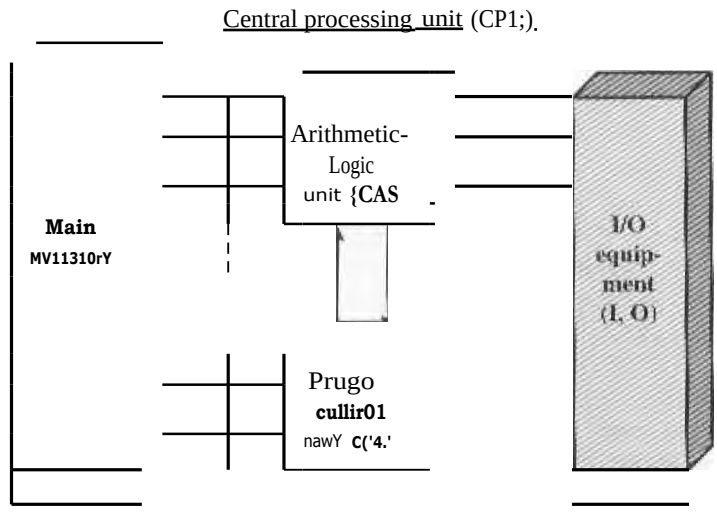


Figure 2.1 Structure of the I. XS Computer

This structure was outlined in von Ncurnann's earlier proposal, 'which k Wort h quotiny at this point IVONN431..

2.2 First Because the device is primarily a computer, it will have to perform the elementary operations of arithmetic most frequently. These arc addition. subtraction, multiplication and divisioi . it is therefore reasonable that it should contain specialized organs for 061 these operations,

It must 1 1' o1.1..R.rved, however, that while this principle as such is probably sound, the specific wo.y in which it is realized requires close scrutiny. , . At an rate a *4:cier1'al arithmetic:el* part of 4111: &vice will probably have to exist .and this constitutes *the first vecific pan:* (.r1-

2.3 Second: The logical control of the device. that is, the proper sequencing of its operations. can he most efficiently carried out by a central control organ. if the device is to be *elastic,* that is. as nearly as possible *eel.? peapose,* then a distinction must he made between the specific instructions given for and defining a particular T prob-tem, and the genefal control organs which see to it that these instructions—no matter what they are—are carried out. Tilt Cornier i rsusl he stored in some way., the latter arc represented by definite operating parts o[the dcvico, Ft the *cemtral control we* mean this latter function and the organs which perform it form the *second specific part: CC.*

2.4 'Third: Any due ice which is to carry out long and complicated sequences of operations (specifically of calculations) must have a con iderahlc memory .

(11) The instructions which govern a complicated problem may constitute considerable material- particularly so, if the code is circumstantial (which it is in most arrarigemcnt9, This inaledal must he remembered

At any rate the total *memory* constitutes *the third .specifiC pan of the device:* M. 2,6111.4..three specific part **s CA. CC** (together (), and **NI correspond** to the *fr.v.sociative* neurons in the human nervous system. It remains to discuss the equivalents of the *sensory* or *afferent* and the. *?no.ror or elfr.rent* **nctuxons.** These arc the *input* and *ow. pea organs rrf* the device ...

The device must be endowed with the al a v 10 maintain input and output (sensory and motor) contact with some specific medium of this type. The medium will he called the *outsides mytreling median of the* dm *is o:*

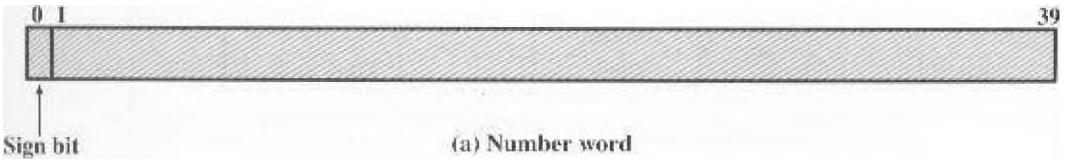
2.7 Fourth: The device must have organs to transfer ... information from R. into its specific parts C and M. These organs form its *input',* the *Pro.rth specific part: L It* will be seen that it is best to make all transfers from R (.ht into M and never directly from C

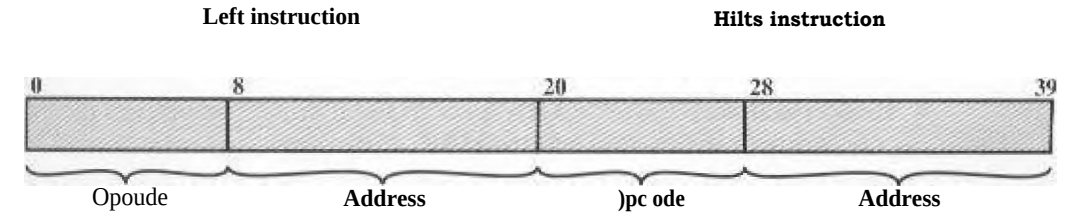
2,8 Fifth: The device **must have** organs to transfer ream its specific parts (.• and M into R. These organs form its *ortipar, the fifth specific part.: O. It* **will In.** seta, that it is again hest to make all transfers from M (by 0) into R. and never directly from C.

With rare exceptions, all of today's computers have this same general structure and function and are thus referred to as von Neumann machines. Thus, it is worthwhile at this point to describe briefly the operation of the !AS computer [BCRIc-Itii. Following [HAYE.98I_ the terminology and notation of von Neumann arc changed in the following to conform more closely lo modern usage the examples and illustrations accompanying this discussion are based on that latter text.

The mem ory of the lAS consists of 11 UU.1 storage locations, called twrds, of 40 binary digits (bits) each. Both data and instructions are stored there. Hence, numbers must be represented in binary form, and each instruction also has to be. a binary code. Figure 2.2 illustrates these formats_ Hach number is represented by a sign hit and a 39-bit value. A word may also contain two 20-Ftit instructions, with each instruction consisting of an s-hit operation code (opcode) specifying the operation to he performed and a 12-hit address designating one of the words in memory (numbered from 0 to 949.

The control unit operates the I AS by fetching instructions from memory and executing them one at a time. Fo explain this. a more detailed structure diagram is needed, as indicated in Figure 2.3. This figure reveals that both the control unit and the ALU contain storage locations, called *registers,* defined as follows:





(b) instruction word

Figure 2.2 lAS Memory Format s

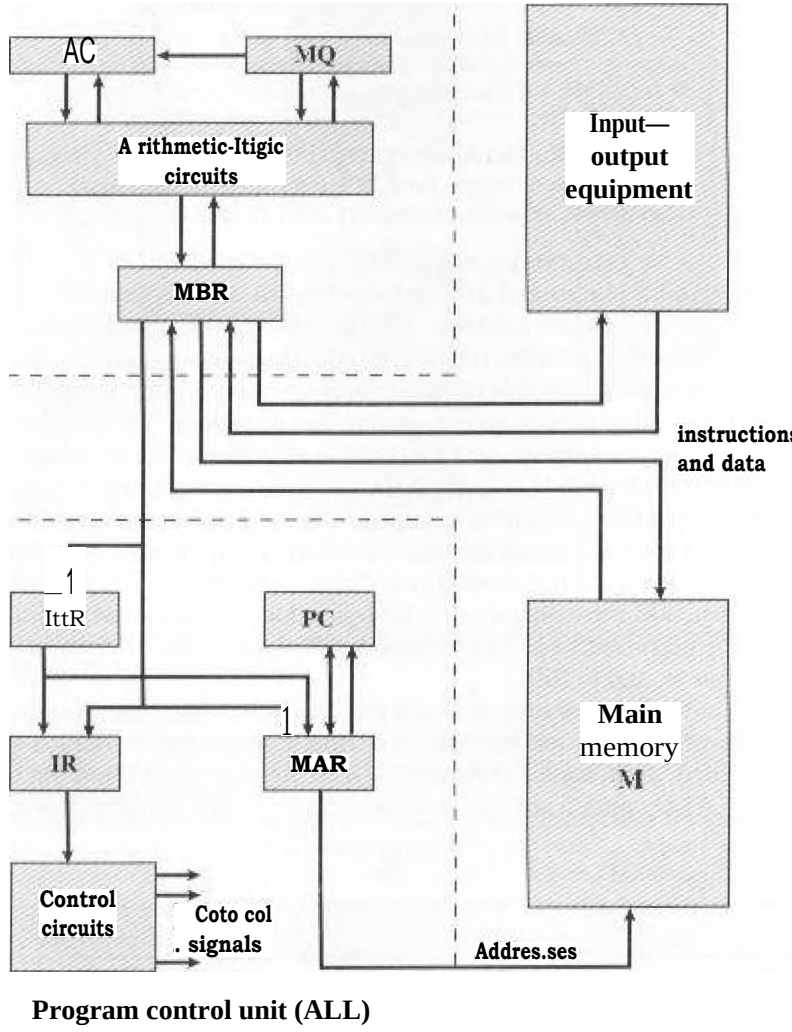
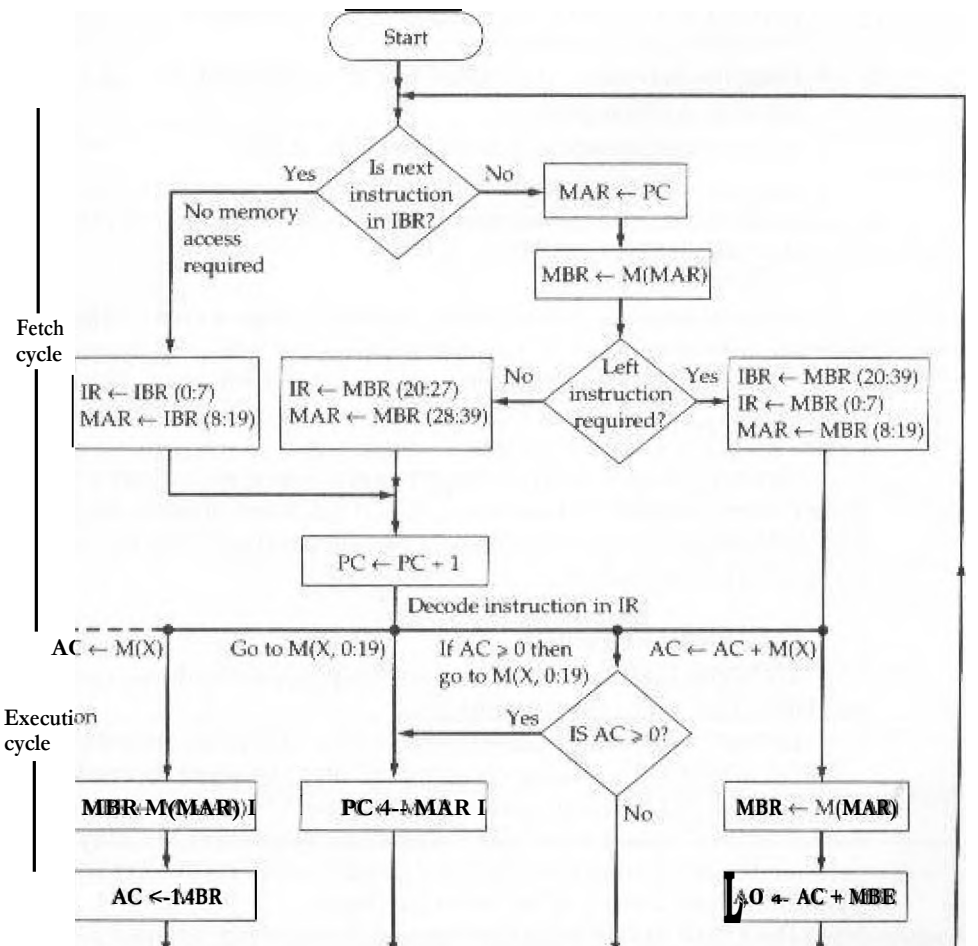


Figure 2.3 Expanded Structure of IAS Computer

- **Memory buffer register (MDR): Contains** a word to he. stored in memc.)ry. ar is used to receive a word from memory.
- **Memory address register (MAR):** Specifies the address **in memory of** the word to be written from or read into the NIBR.
- **Instrudion register (IR): Contains** the 8-hit op-code instruction being executed.
- **Instruction buffer register (lBR): Employed acs** hold temporarily the right-hand instruction from a word in memory.
- **Program Counter (PC): Contains the** address of the next instruction-pair to be fetched from memory.

- Accumulator (AC) and **multiplier quotient (MQ):** Employed to hold temporarily operands and results of AU! operations. For example. the result of multiplying two 40-hit numbers is an 80-hit number the **must** significant 40 hits are stored in the AC and the least significant in the MQ.

The IAS operates by repetitively performing an *instruction cycle,* as shown in Figure 2.4. Each instruction *cycle* conskis of two subcycles. During. the *fetch cycic,* the opcode of the next instruction is loaded inio the IR and the address portion is loaded into the. MAR. This instruction may be taken **from** the **1BR,** or it can be **obtained from memory by loading** 41 word into the MDR. and then down to the 1BR, l R, and MAR.



M(X) = contents of memory location whose address is X
(X = bits X through

Figure 2.4 Partial Flowchart of lAS Operation

Why the indirection? Those operations are controlled by electronic circuitry and result in the use of data paths. To simplify the electronics, there is only one register that is used to specify the address in memory for a read or write, and only one register to be used for the source or destination_

Once the opcode is in the IR, the *ex•cure cycle* is performed_ Control circuitry interprets the opcode and executes the instruction by sending out the appropriate control signals to cause data to be moved or an operation to be performed by the ALU.

The lAS computer had a total of 21 instructions, which are listed in Table 2.1. These can be grouped as follows:

- **Data transfer:** Move data between memory and ALL1 registers **or between Iwo** ALE: registers.
- **Unconditional branch:** Normally, the control unit executes instructions in sequence from memory. This s equence can be changed by a branch instruction. This facilitates repetitive operations.
- **Conditional branch: 'Fhe** branch can be made dependent on a condition. thus allowing decision points.
- **Arithmetic:** Operations performed by the ALU.
- **Address modify: Permits addresses to be** computed in the ALL' and then inserted into instructions stored in memory. 'F'his allows a program considerable addressing flexibility.

Table 2,1 presents instructions in a symbolic, easy-to-read form. Actually, each instruction must conform to the format of Figure 2.2b. The opcode portion (first 8 bits) specifies which of the 21 instructions is Lo he executed. The address portion (remaining 12 bits) specifies which of the 11.M.14.1 **memory** locations is **to** be involved in the. execution of the instruction.

Figure 2.4 shows several examples of instruction execution **by the control** unit. Note that each operation requires several steps. Some or these. arc quite elaborate. The multiplication operation requires 39 suboperations, one for each **bit position except** that of the sign bit!

Commercial Cons puters

The 1950s saw the birth of the computer industry with two companies, Sperry and IBM, dominating the marketplace.

In 1947. Eckert and Mauchly formed the **Eckert-Mauchly Computer** Corporation to manufacture computers commercially. Their first successful machine **was** the UNIVAC I (Universal Automatic Computer), which was commissioned by the Bureau of the Census for **the 1.950 calculations. The Eckert-Mauchly Computer Corporation** became part of the UNIVAC division of Sperry-Rand Corporation, which went on to build a series of successor machines.

'Die UNIVAC I was the first successful commercial computer. It was intended as the name implies, for both scientific **and** commercial applications. The **first paper describing** the system listed matrix algebraic computations. statistical problems, premium billings for a life insurance company, and logistical problems as **a sample of the tasks it could** perform.

Table 2.1 the I AS Iosiruction ScL

Instruction 7 .1'1¹e	Opcode	Symbolic Representation	Description
DErta transfer	00001010	LOAD VIO	fra usfer contents register MO to the accumulator AC
	04001001	LOAD MO.M X)	I'ratisler contents of *fneMOky* lot:11'60n X CO **MO**
	001 4111)t1	**STDR MiX** .)	**libra:lifer** mill En LS Ili elixir LIluk tor (0 Tn.:mew!: lomitjon X
	I MINK11:01	**LOAD** KT(X)	Tranhcr r'.1{ X) 1451 ho zieetirmi1m1or
	1:00:1(I1.)10	**LOAD M(X}**	**Transfer - M(X** Li} LhL accurnLIIntor
	IKAXIOI/11	**LOAD NUN)!**	**TTaa}..1C1** 4'·111AC' 01 ,WM to the au:LnntilE11.(ir
	CIODOO **t 0f)**	**LOAD** · I**M(X)1**	Tlonsfer —I47(X) to the :iccumulalui
Unconditional **branch**	(1)]01101	**JL**ᴵ **MP MPC,11:191**	THk ncNI insirlieriOn from left halt **of KIVX)**
	01X100 ¡ **10**	.1 1..ᴵ1',I1³ **M(X.20:39)**	**Take next irISITUCLICIa** ream rig1'll half of !irl(X)
Conditional branch	1)0001111	31..:MP+M(X.02191	11 **ninnber m** ihc: necumillMor is nOnriuFaiive, inSCuocii on frorn **left half** 01 **MIX** ,
	I')**00i0r.i.10**	*N•20:119)*	*11* r in **I hc aLLLI** nh **nhiI** oris aunn Livc. Lake nix I insl ruction **Cron, righi. half 1.)**
.Arithme tic	00000101	ADD M(X)	Add M(X1 to A (:; **put the remit in AC**
	01)001)11 /	*ADD* '11(X)1	**Add IM(X)1** 10 **AC; pi.v** ¡ **he result in AC**
	0fX1(JR1	**51(** }	subtrkirt m(x) rivn, **AC; put** the re41.1.11 in AC
	(01:)01.01)0'	Si R IM(X)	**SuhLrael. frorEi AC: TruL ih: miniinder 1a AC**
	40041(;1 I	**MU]_**	Multiply **M{X1 by MO: put inosi significant hitE of result in AC.** *put* **icant bib., in .M.Q**
	(IOW' LOL)	DJV MIX)	1)i %dile **hy M(X):: put** 1.11i **rollout in** "....1(} ;111c1 **the ra.maihder** ill **AC**
	00411i)(01./	**LSH**	**?v]al iply dEcumul Dior I)?..** 2 **0.0.. Shift !ell.** One hil !Imo **on1**
	0001.1111)1	KSI-[	Diuidc .112CLI **TUIFacIT by 2 fj.C%,** stLil **rich(** one position)
Alkiresr.. modify'	**1)0010(110**	**STDR** trli.X.5:19)	Retched **Icr1 L0dre.5 field at ?41(X:)** hti **ri eh i-rnoNt hots of AC**
	000/01)1 I	**STOR** M(X,2EI:119)	**Replace right addrccs i kI** 4tC M(H) **by [2 right-m(1EL bias arAC**

The UNIVAC II, which had greater memori, ' capacity and **higher TX!rfOrrilaitCC, than** the UNIVAC **I,** was delivered in the **laic** 1950s and **illustrates** several trends that have rornainc.d characcetistic of the computer indumry_ **adviuices in** technology allow Companies 1[3 conlnn1.1 t**o build lar**ger, **more** powerful **compu**ters. **Second_ each company tries to make its new machines** ***upward compatible with*** **the older**

machines. This means that the programs written for the older machines can be executed on the new machine. This strategy is adopted in the hopes of retaining the customer base: that is, when a customer decides to buy a newer machine, he or she is likely to get it from the same company to avoid losing the investment in programs.

'I]e UNIVAC division also began devclopment of the 1100 series of computers, which was to be its major source of revenue. This series illustrates a distinction that existed at one lime. The first model, the UNIVAC 1103, and its successors for many years were primarily intended for scientific applications, involving long and complex calculations. Other companies concentrated on business applications, which involved processing large amounts of text data. This split has largely disappeared, but it was evident for a number of years.

IBM, which was then the major manufacturer of punched-card processing equipment, delivered its first electronic stored-program computer, the 701, in 1953. The 701 was intended primarily for scientific applications IBAS11811. In 1955, 1BM introduced the companion 702 product, which had a number of hardware features that suited it to business applications. These were the film of a long series of 70017000 computers that established I HM as the overwhelmingly dominant computer manufacturer,

The Second Generation: Transistors

The first major change in the electronic computer came with the replacement of the vacuum tube by the transistor, The transistor is smaller, cheaper. and dissipates less heat than a vacuum tube but can be used in the same way as a vacuum tube to construct computers, Unlike the vacuum tube, which requires wires. metal plates, a glass capsule, and a vacuum, the transistor is a *solid .ware device,* made from silicon.

The transistor was invented at Bell Labs in 1947 and by the 1950s had launched an electronic revolution. It was not until the late 1950s, however, that fully transistorized computers were commercially available. IBM again was not the first company to deliver the new technology, NCR and, more successfully, RCA were the front-runners with some small transistor machines. IBM followed shortly with the 7000 series.

use of the transistor defines the *second Toleration* of computers. It has become widely accepted to classify computers into generations based on the fundamental hardware, technology employed (Table 2,2), Each new generation is characterized by greater processing performance, larger memory capacity, and smaller size than the previous one.

TOW 2.2 Computer **Generations**

Generation	Approximate Dates	Tedmology	'typical Speed (operations per second)
	1946-1957	Vacuum **tube**	40,000
		Transi&t.or	200.000
	1 1 **t65-1971**	**Small- and mediunt-scale integration**	1 ,(11:41,000
4	**1972 .1417**	**L.arge-geale intxparion**	1t1,000,000
S	**1975-**	**Very-large-si:ale integration**	**100.004000**

But there arc other changes as well. The second generation saw the introduction of more complex arithmetic and logic units and control units. the use of high-level programming languages, and the provision of *system sofiware* with the computer.

The second generation is noteworthy also for the appearance of the Digital Equipment Corporation (DEC). DEC was founded in 1957 and in that year, delivered its first computer, the PDP-1. This computer and this company began the minicomputer phenomenon that would become so prominent in the third generation.

The IBM 7094

From the introduction *of* the 700 series in 1952 to the introduction of the last member of the 7000 series in 1964, this IBM product line underwent an evolution that is typical of computer products. Successive members of the product line show increased performance. increased capacity. andior lower cost.

Table 2.3 illustrates this trend. The size of main memory. in multiples of 2^{1}" 36-bit words, grew from 2K (l K = 2 ') to 32K words, while the time to access one word of memory the *memory cycle time,* fell from 30 is to 1.4 1,1.s. The number of opcodes grew from a modest 24 to L5.

The final column indicates the relative execution speed of the central processing unit (CPU). Speed improvements are achieved by improved electronics *(e.g.,* a transistor implementation is faster than a vacuum tube implementation) and more complex circuitry. For,example, the IBM 7094 includes an lnstruetion Backup Register, used to buffer the next instruction. The control unit fetches two adjacent words from memory for an instruction fetch. Except for the occurrence of a branching instruction, which is typically infrequent, this means that the control unit has to access memory for an instruction on only half the instruction cycles. This prefetching significantly reduces the average instruction cycle lime.

The remainder of the columns of Table 2.3 will become. clear as the text proceeds.

Figure 2.5 shows a large (many peripherals) configuration for an IBM 7094. which is representative of second-generation computers IBELL71a]. Several differences from the 1AS computer are worth noting. The most important of these is the use of *data channels.* A data channel is an independent I/O module with its own processor and its own instruction set. In a computer system with such devices, the C.PU does not execute detailed I/O instructions. Such instructions are stored in a main memory to be executed by a special-purpose processor in the data channel itself. The CPU initiates an I/O transfer by sending a control signal to the data channel. instructing it to execute a sequence of instructions in memory. The data channel performs its task independently of the CPU and signals the CPU when the operation is complete. This arrangement relieves the CPU of a considerable processing burden.

Another new feature is the *multiplexor,* which is the central termination point for data channels, the 0^{3}1._, and memory. The multiplexor schedules access to the memory From the CPU and data channels, allowing these devices to act independently.

The Third Generation: Integrated Circuits

A single. self-contained transistor is called a *discrete componem.* Throughout the 1950s and early 1960s, electronic equipment was composed hugely of discrete components—transistors, resistors, capacitors. and so on. Discrete components were

Table 2.3 Example Mernhers of the BM 7.1.'0/70W Series

Model Number	Firs! Dvery	CPU Tech-	memae nology	(:yele Time(p)	Memory K)	Number w °prudes	Number of Index Register%	H ardwired Flua(hig Poin	.IJ O Overlap (CM n ne Is)	Instruction *I* Overlap	Speed (relative tu 701)
701	1952	vacuum tu	**Fled 10- NLaLit: LtibeS**		24	24		111	.1143	114	
744	1955	VaCUUM tubeS	Core		-1-32			!ies.	**rit)**	110	2.5
709	195g	**VACUUM .a>**	**(kirc**	12	32	140	3	**ycs.**	**Ye**[5]	(4(1	
TM]	1961'1	Transistor	**t:ore**	zh		169		yi2s	yes	11{1.	25
7094	1%2.	**TransisLor**	**Oafe**		32	15	7	ycs (010131111:: **PrCeiSiOn**	**ytY**	ycs	3D
71)94 11	'1 %4	Trans:ore!	UNIT:	1.4	32	, r		**yes** id4pubit **ivrccisiort)**	Yes	**yes**	50

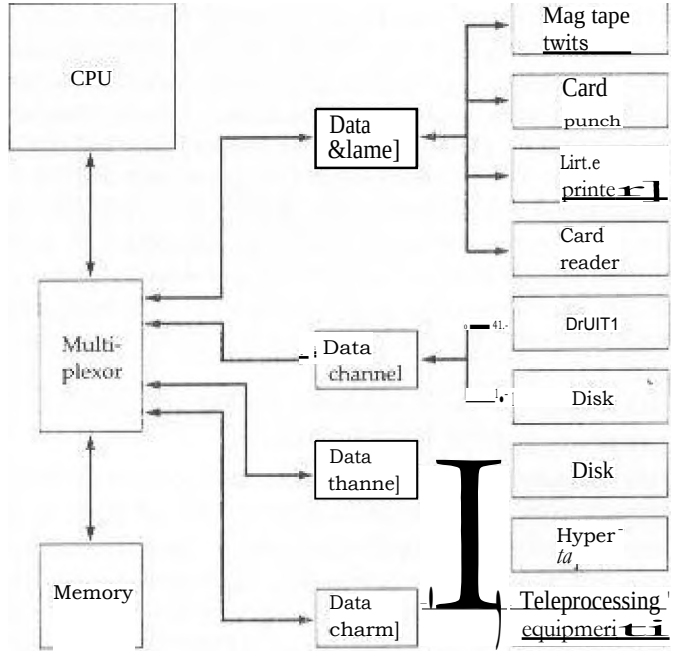


Figure 15 An 7094 Configuration

manufactured scparaicly, p4icloged in their own conlainen .. and soldered or wired together onto 1113Sonite-like circuit boards, which were then installed in computers, oscilloscopes. and other electronic equipment. Whenever an electronic device called fora transistor, a Little tube of medal containing a pinhead-sized piece of silicon had to be soldered to a circuit hoard. The entire mlinufactuting process. from transistor to circuit board, was expensive and cumbersome.

.11.ese facts of life were beginning to create problems in the computer industry. Early second-gcncra Lion comptilers con Lammed about 10.000 lransislors. This figure grew to the hundreds of thousands, making the inanufacture of ncwur,litirrc powerful machines increasingly difficult.

In 145K came ibe achievement [hat revolutionized eloetronics and started the era of microelectronics: the inveri1ion of the iriwgnited cirQui1. IL is. d,: iniegrated circuit that defines the third generation of computers. In this section we provide a brief introduction to the technology of integrated circuits. Then we look at perhaps the two most importni members of the I hird genera Lion. both cif which wcre inlroduced at the beginning of that era; the IBM System/360 and the DEC PDP-8.

MierOel ectrouies

Microelectronics means, Literally, "small electronics." Since the beginnings of electronics and the computer industry, there has been a persistent and consistent trend ii)wurLI Ilic reduction in size of digital electronic. circuits. Bc[orc exkirn-

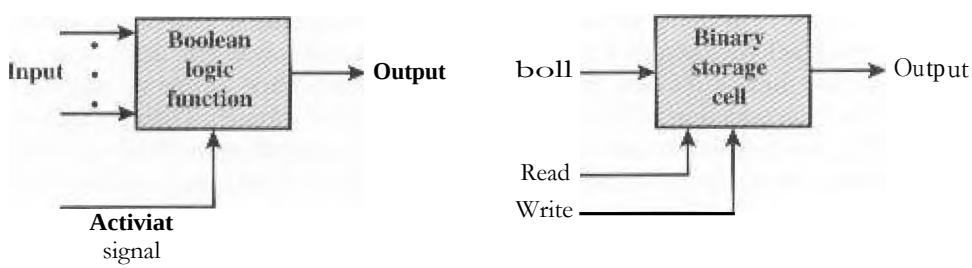
ining the implications and benefits of this trend, we need to say something about the nature of digital electronics, A more detailed discussion is found in Appendix A.

The basic elements of a digital computer, as we know, must perform. storage: rinavement, processing_ and control functions. Only two rundameni al types of components are required (FiRure 2.6): gates and memory eel's. A gate is a device that implements a simple 'Boolean or logical function. such as IF A AND *B* ARE TRUE 'TI FEN *C* IS TRUE (AN I) gate}. Such devices are called gales because they control data [low in much the same way that canal gates do. The memory cell is a device that can store one hit of data:. that is. the device can be in one of two stable states al, any time. By interconnecting large numbers of these fundamental devices, we ciin construe' a computer. We can relate this to our four basic functions as follows:

- **Data storage:** Provided by memory cells,
- **Data processing:** Provided by gates,
- **Data movement:** The paths between componvnis are used to move data from memory to memory and from memory through gates to memory.
- **Control;** The paths between components can carry control signals_ rim example, a gate will have one or two data inputs plus a control signal input that activates the. gate. When the control signal is ON, the gate performs its function on the data inputs and produces a data output, Similarly, the memory cell will store the bit that is on its input lead when the WRITE control signal is ON and will place the bit that is in the cell on Ifs output lead when the READ con-in)! signal is ON

Thus, a computer consists of gates, memory cells, and interconnections among these elements. The gates **and** memory cells are. in turn, constructed of simple digital electronic components,

The integrated circuit exploits the fact iliaL such components am transiswrs, resistors, and conductors can he fabricated froin a semiconductor such as silicon. It is merely an extension of the solid-state art to fabricate an entire circuit in a line piece of silicon rather than assemble discrete eomponenLs made from separate pieces of silicon into the:Name circuit. Many transistors can be produced **at** the mime



(a) Gate

(10} Merrior±r cLLl

Figure 2.6 Ikuliiiintental Computer Elements

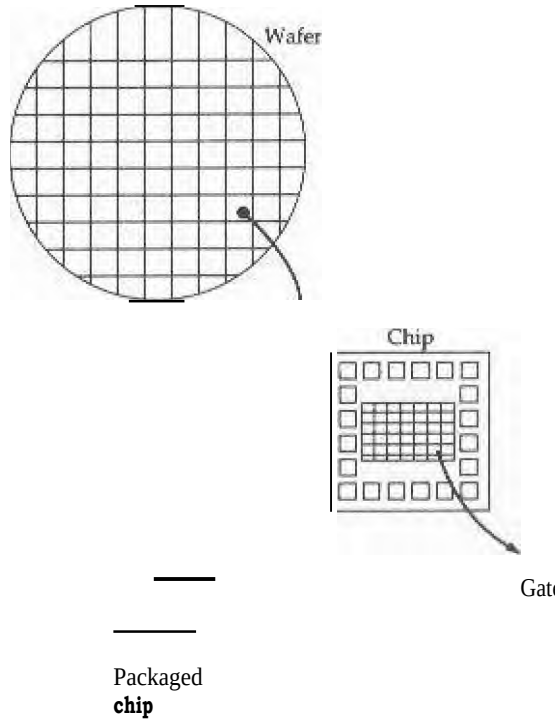


Figure 2.7 ReIntiorisltip between Wafer, Chip, and Oate

Rink on a single wafer of silicon. Equally important, these transistors can be connecti2d with a process of meiallization to form circuits,

Figure 2.7 depicts the key uoneepts in nn inteunted circuit. A thin ivafer *of* silicon is divided into a matrix of small areas, each a few millimeters square. The iden-**Lica**' circuit pattern i6 fabricated in each area, and the wafer is broken up into *claps.* Each chip consi.qs of many gates rind or memory cells plus a number of input and output attachment points. This chip k then packaged in housing that protects it and provides pins for attachment to devices beyond the chip. A number of these packages can then he interconnecied on a printed circuit board to produce larger and more complex circuits.

Initially, only a few ates or memory cells could be reliably manufactured and pact aged together. These early integrated circuits are referred to as *small-scale integration* (SSi). As time went on, it became po7'.si ble to pack more nand more components on the same chip. This growth in density is illustrated in Figure 2.8: it is ore of the most remarkable technological trends ever recorded. This figure reflects the famo **us NI** core's law, which propounded by (Jordon Moore, cofounder of in 1%5 IN'100R65j. Moore observed that the ntrrnber of trAnsiiitors that could he put on a single chip was doubling every year and correctly predicted that this pace would con *tin* ue into the near future. To the surprise of many, including Moore, the pace continued ycar after **War** mid decade 11 1 r decade. The pace siowod to to doubling every 18 months in the 19711s. but sustained that rate ever since.

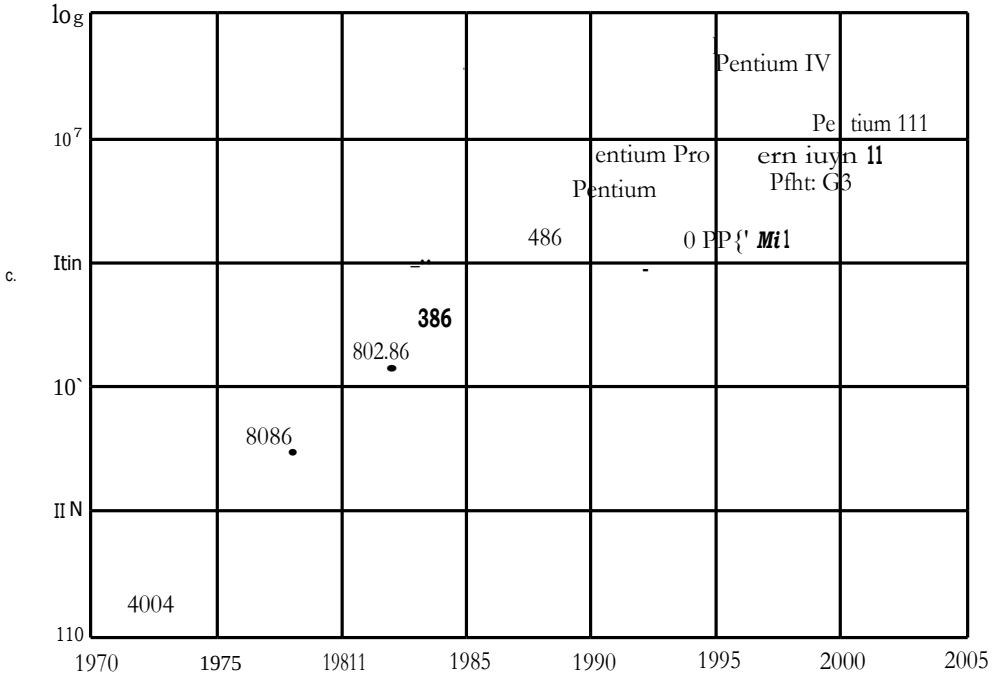


Figure 2.8 "rovv111 in CPI: Transistor Count

The consequences of Moore's law are profound:

1. The cosi ()I' a chip has remained virtually unchanged during this period of rapid growth in density. This means that the cosi of computer logic and memory circuitry has fallen at a dramatic rate,
2. Because logic and memory elements are placed closer together on more densely packed chips, the electrical path is shortened, increasing operating speed.
3. The computer becomeS smaller, making it more convenient to place in a variety of environments.
4. There is a reduction in power and cooling requirements.
5. The interconnections on the integrated circuit are much more reliable than solder connections. With more circuitry on each chip, there are fewer interchip connections.

1B A[System/360

Hy 1964, Ifi M had a firm grip on the computer market with its MOO series of machines. In that year. IBM announced the Systen060. a new family of computer products, Although the announcement itself was no surprise, it contained some unpleasant news for current IBM customers: The 360 product line was incompat-

Table 2.4 Key Characteristics of the Systerni360 Family

Characteristic	Model 30	Model 4(1	Model 50	Model 65	Model 75
Maximum mummy sizq. (bytes)	64K	256K	256K	5121;	512K
Data rate from memory (Mbyte.vs)	0.5	0.8	2.0	8.0	16.0
Processor cycle Lime (i.Ls)	1.11	0.625	0,[5]	0.25	0.2
Relative speed	1	:3.5	10	21	.50
Maximum number of data charmds	3	3	4	6	6
Maxi MUM data rate in uric channel (Khytesis)	250	4110	)i00	1 *2.5t)*	*1250*

ible with older IBM machines. Thus, the transition to the 360 would be difficult for the current customer base. This was a bold step by IBM, but one IBM felt was necessary to break out of sonic of the constraints of the 7000 architecture and to produce a system capable of evolving with the new integrated circuit technology [PADE8I, GIFTS?]. The strategy paid off both financially and technically, The 360 was the success of the decade and cemented IBM as the overwhelmingly dominant computer vendor, with a market share above 70%. And, with some modifications and extensions, the architecture of the 360 remains to this day the architecture Of IBM's mainframe' computers, Examples using this architecture can be found throughout this text.

The System1360 was the industry's first planned family of computers. The family covered a wide range of performance and cost. Table 2.4 indicates some of the key characteristics of the various models in **1965** (each member of the family is distinguished by a model number). The models were compatible in the sense that a program written for one model should be capable of being executed by another model in the series. with only a difference in the lime it takes to execute.

the concept of a family of compatible computers was both novel arid extremely successful. A customer with modest requirements and a budget to match could start with the relatively inexpensive Model 30. Later, if the customer's needs grew, it was possible to upgrade to a faster machine with more memory without sacrificing the investment in already-developed software. The characteristics of a family are as follows:

- **Similar or identical instruction set: In** many cases, the exact same set of machine instructions is supported on all members of the family. Thus, a propram that executes on one machine will also execute on any other. **in** some cases, the lower end of the family has an instruction set that is a subset of that **of** the top end of the family. This means that programs can move up but not down,

---

'The term *mainframe* is ti.scd for the. lareer, most powerful computers other than supercomptirers. Typical characteristics of a mainframe are that it supports a large database, has elaborate 110 hardware. and is used in a cuniral data processing

- **Mintier or identical operating system: The** same basic oparating system is available for al] family members. In some cases, additional features are added lo the higher-end members.
- **Increasing speed: The** rate of instruei ion execution increases in going from lower to higher family members.
- **Increasing number of 110 ports: In** going from lover to higher family members.
- **Increasing memory size;** In going from lower to higher family members.
- **increasing cos:** In going from lower to higher **family** members.

How could such a family concept be implemented? Differences were achieved based on three factors: basic speed, sire, and degree of simultaneity [STEV64]. For example, greater speed in the execution of a given in7iiruction could be gained by the **of** more complex circuitry in the AT allowing suboperations io be carried out in parallel. Another way of increasing speed was to increase the width of the data path between main memory and the. CPU. On. the Model 30, only 1 byte (8 bits) could be fetched from main memory al a time, whereas 8 bytes could be fetched at a time on the. Model 70.

The Systent1360 not only dictated the future course of IBM but also had a profound impel on the entire industry. Many of its features have become standard on other large computers.

DEC PUP-8

in the same year that IBM shipped its first Systerni.;60, another momcnious first shipment occurred: PDP- from Digital Equipment Corporation (DEC). At a time when the average computer required an air-conditioned room, the PDP-8 (dubbed a minicomputer by the industry, after the miniskirt of the day) **WRS** small enough Ihat it could be. placed on top of a **lab bench or be built into** other equipment. It could not do everything the mainframe could. but at $1 0,000. it was cheap enough for each lab technician to have one. In contrasl. the System/360 series of mainframe computers introduced just a Lew months before cost hundreds of thousands of dollars.

The [ow cost and small size of the PDP-8 enabled another manufacturer to purchase a PDP-K and integrate it **into** a total system fur resale. These other manufacturers came to be known as original equipment manufacturers OEMs), and the OEM market became and remains a major segmenl 0r the computer marketplace.

The FDP-8 was an immediate hit and made [)HC 's fortune. This machine and other members of the PDP-8 family that followed it (see Table 2.5) achieved a production status formerly reserved for IRM computers, with **about** 50,060 machines sold over the next dozen years, As DEC"s official history puts it, the PDP-8 "estah-iished the concept of minicomputers, leading the way to a multibillion dollar indus-LTV,' It also established DEC as the number one minicomputer vendor. and, by the time the PDP-8 had reached the end of its useful life, DEC was the number two computer manufacturer, behind IBM.

Table 2.5 Evolution *of* Ow PDF 8 I V(LL

.M4-xle.1	First Shipped	co51. of 12-bi(Words	– 4K IA Memory (5.10(105)	D i a Rah?. trom Memory Twords!,u.$).	(cubic feet)	Ionova t)r)s. ri nd I rripro mri eu [s
PDP-8	4;65	1E1.2		1.26	g.G.	A ttLornat:c.wi.rt..-wcapping prod ueziot
FL,N:5	91045	i.'74		D.08	3.2	Serial i nsiruc!icul ii npletrie isEat.on
	4:fifi	11.6		1.34	8.0	2.]odium. Scale integrated ci ro.iitis
f2DP-8.:L	ii6.5	7.0		1.2C.	2.0	Snrilk r Cc! b:nrc
PDP '8113	3:7 I	4.9')		1.32	2.2	Ow it'll/5
PDT KM.	6:72	169		1.52	L8	H ii1f-smt shined wil 1 i I t wer L...l k i tints Ni'E
	1.75	2./)		1.34	1.2	Sernicond uc;or trte:Ltury: CloH Ling-point rill)1:Lssor

In c.0111TaS1 to the central-switched architecture (Figure 2.3) ivied by IBM on its 700/7000 and 360 gystems, later models of [he MP-8 used a structure that is [low virtually universal for minicomputers and microcomputers: [lie bus structure, This is illustrated in Figure 2,9- The PDP-R bus, called the. Omnibus, consists of 96 separate sigma[paths. used to carry control, address, and data signals. BeckLIJSC all system components share a common set ot` i.2 .rta[pal hs, [heir use must be controlled by the CPU. This architecture is highly flexible., allowing modules to he plugged into [he bus to crea I e various contigura lions,

Later Generations

Beyond I he third generation there is less general agrE.crnent on defining generations of computers. Table 2,2 suggests that then: have been li fourth and a fifth aenend-li on, based on advances in integraled circuit technology. With the introduction of large'-scale integration (LSO. more than lOW components can he placed on li single integrated circuil chip, Very-Earge-scale integration (VLSI) achieved more than /0,000 components per chip, and current VLSI chips can contain more than 100,00 components.

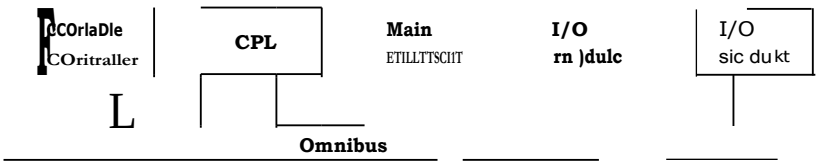


Figure 2.9 PD P-8 13 us Struci tire

With the rapid pace of technology, the high rate of introduction of new products. and the importance of software and communications as well as hardware. the classification by generation becomes less clear and less meaningful. It could be said that the commercial application of new developments resulted in a major change in the early .1970s and that the results of these changes are still being worked out. In this section, we mention two of the most important of these results.

Semiconductor Memory

Thc first application of integrated circuit technology to computers was eonstruction of the processor (the control unit and the arithmetic and logic unit) out of integrated circuit chips. But it was also found that this same technology could be used to construct memories,

in the 1950s and 1.960s, most computer memory was constructed from tiny rings of ferrom welie material. each about a sixteenth of an inch in diameter. These rings were strung up on grids 0r fine wires suspended on small screens inside the computer. Magnetized one way, a ring (called *a (.'ore)* represented a one: magnetized the other way, it stood for a zero. Magnetic-core memory was rather fast; it took as little as a millionth of a second to read a **bit** stored in memory. **But it** was expensive, bulky• and used destructive readout: The simple act of reading a core erased the data stored in it. It was therefore necessary to install circuits to restore the data as soon as it had been extracted.

Then, in 1970, Fairchild produced the first relatively capacious semiconductor memory. This chip, about the sic of a single core. could hold 256 hits of memory. It was nondestructive and much faster than core. It took only 70 billionths of a second to read a bit. However.. the cost per hit was higher than for that of core.

In 1974, a seminal event occurred: The price per bit of semiconductor memory dropped below the price per bit of core memory. Following this, there has been a continuing and rapid decline in memory cost accompanied by a corresponding increase in physical memory density. This has led the way to smaller, faster machines with mentor; sizes of larger and more expensive machines with a time lag of just a few years. Developments in memory technology, together with developments in processor technology to be discussed next. changed the nature of computers in less than a decade. Although bulky, expensive computers remain a part of the landscape, the computer has also been brought out to the "end user," with office machines and personal computers,

Since 1970, semiconductor memory has been through 11 generations: 1K, 4K, 16K. MK, 256K, 1M, 4M, 16M,)4M, 256M, and. as of this writing, 1 CT bits on a single chip (.1K = 2^{1}". 1 M = 10 = 2n. Each generation has provided four times the storage density of the previous generation, accompanied by declining cost per hit and declining access time.

Microprocessors

Just as the density of elements on memory chips has continued to rise, so has the density of elements on processor chips. As time went on, more and more elements were placed on each chip. so that fewer and fewer chips were needed to construct a single computer processor.

A breakthrough was achieved in 1971, when Intel developed its 4004. Ihe 4004 was the first chip to contain all of the components **of a CPU on a single chip:** The microprocessor was born.

The 4004 can add two 4-bit numbers and can multiply only by repeated addition. By today ˙s standards, (he 4004 is hopelessly primitive, **but it marked the beginning of a continuing evolution** of microprocessor capability and power.

This evolution can be seen most easily in the number of bits that the processor deals with at a time. There is no clear-cut measure of this, but perhaps the best measure is the data bus width: the number of bits of data that can be brought into or sent out of the processor at a time. Another measure is (he number of bits in the accumulator or in the set of general-purpose registers. Often, these measures coincide, but not always. For example, a number of microprocessors were developed **that operate on 16-bit numbers in registers but** can only read and write 8 bits at a time.

The next major step in the evolution of the microprocessor was the introduction in 1972 of the Intel 8008. This was the first 8-hit microprocessor and was almost twice as complex as the 404.

Neither **of these steps was to have the impact** of the next major event: the introduction in 1974 of the Intel 8080. This was the first general-purpose microprocessor. Whereas the 4004 **and** the 8008 had been designed for specific applications, the 8080 was designed to be the **CPU of a** general-purpose• microcomputer. Like the 8008, the 8080 is an 8-bit microprocessor. The 8080, however, is faster, has a richer instruction set. and has a large addressing capability.

About the same li me, 16-bit microprocessors began to he developed, However. it was not until the end of the 1970s **that** powerful, general-purpose 16-bit microprocessors appeared. One of these was the 8086. The next step in this trend occurred in 1481. **when both Bell Labs** and I It:Men - Pack ard developed 32 bit, single-chip microprocessors. **Intel introduced** its own 32-bit °processor, the 80386, in 1985 (Table. 2.6),

Table **2.6 Evolution** of Intel Microprocessors

fat 1970s Processors

	4004	8008	8080	8086	8088
Introduced	11:15:71.	4102	4:104	OW75	6i4:79
Clock speeds	ltN KHz	108 KHz	2 MEI./	5 MHz, S MHz, LO MHz	5 MHz, 8 MHz
Bus width	4 hits.	8 bits	8 bits,	16 bits	8 His
Number 41t transistors (ibierrins)	2300 (10)	3500	&KID (6)	29.0011 ci../	29 (%10 {3)
Addressable memory	640 bytes	I b kByLcs	64 K Bytes	1 MB	1 MB
Virtual memory	—	—			—

Table 2.6 ',coot *lituz4rti)*

111) **Milk Processors**

	1i02146	386111 **DX**	**386TM SX**	**486TM DX CPI;**
rothieetl.	211. tit	(l."l **7N5**	6:16111g	410189
Clock speeds	6 MHz 12.c MHz'	]6 MHz-33 1 11-17.	t6 MHz--33 '241-17.	25 MHz-51) MHz
rius width	16 hill	32 bits	16 bits	32 114;
Number of transistors (micronsl.	134,000 (1.9	275,000 {L)	275,000 (1-.)	1.2 rniliigh
Addressabk incrriory	16 rnetabytes	4 i4iti.ikbytes	4 gipbytes	4 7ip,abytes
Virtual rinerrsors..	1 t.6p.abyLe	61 terabytes	64 tcrabyles	64 liz.TH1':..Leg

(el 1991111s PrO M.SuirS

	486TM SX	**Pentium**	Pentium	**Pentium U**
In t rr Ouced	412219	3122;93	1 1.:01195	
Clock spec ds	14H7.— 133 MT-1	430 **MI 12.-** L61110142	150 MHz-. 200 MHz	2011 h1 Hz- 3(J MHz
Bus width	32 bits	32 bit5..	64 bitii	64 bits
Number of transistors (Microns)	1.1g5 **O.)**	:i.1 rrssllii 5rr (**A)**	5.5 (0.6)	milflic m (0.35)
Addr,7,155.21 c merrhiry	4 gi.githylMs	4 gigabys es	64 k6gabyLes	64 gabyt.:;:s
Virtual memory	64 ',erg 3.1v	64 terabytes	64 1 eraN. Les	Lerakaes

(id) Recent PmeesSors

	Pentium III	**Pentium 4**
luir.r5ctuceil	r)6:99	1.1.:2.01X1
Clack speeds	4.5D-660 MHz	1.3-1,8-1.31-1z
Bus wOlri	64 bits	64 bits
Number *61* SiOrS(microns.)	X15 millinn (0.19)	42 ru alio a
Addressable **MUrnofy**	64 14tgul',.01.1L.s	64 gtilabytes
Virtuill memory	ii4 terultytes	64 terabytes

11110; Ci'irp. **ILL T:1** www.j111.'11.rorrAnrOlilnuReuma **5** annilltiae. Brx n.htm

2.2 DESIGNING FOR PERFORMANCE,

Year by vear, the cost of computer systems continues to drop dramatically. white the performance find capacity of systems continue to rise equally dramatically. Al. 4i 104AI wm chOUSe club, you can pick up a personal computer for Less than 0f}0 that packs the wallop of an IBM rmiinfrome from 10 years ay.o. Inside that personal computer, including the microprocessor and memory and other chips, you get Ms of millions of transistors, You cannot buy [Iii) million of anything else for so little. That many sheets ,,r toilet paper would run more than $100,000.

Thus, rve have virtually "free" computer power. And this continuing technological revolution has enabled the development of applicalions of astounding complexity and poker- 1-err example, desktop 4pplicw ions that require the great power of today's microproce.ssor-based systems include

- image processing
- Speech recognition
- Videoconlereneing
- Multimedia authorina
- Voice ;'Ln d video annotation of riles
- Simulation modeling

'Workstation systems now support highly .,ophisi ieated engineering and scientific applications, as well as simulation systems, and ha '.•e the abilit!,' to support image and video applications. In addition. businesses are relying on increasingly powerful savers to handle. lransaction and &Lila base processing and to support nmssivc clientiserver networks that have replaced the huge mainframe computer centers of yesteryear.

What is fascinating abou I all this from the perspective of computer organization arid architecture is that, on 1hc *(me* hand. the basic buildin, Mocks for today's computer miracles are virtually the same as those of the IAS computer from over 50 years ago, while on the other hand. the techniques for squeezing the last iota of pCrIbrmnce IAA or I tic malt:11211s at hand have become increasingly sophisticated.

This observai ion serves as. a guiding principle ['or the presentation in this hook. As we progress through the various elements and Components of a computer, two objeetives are pursued, First, the book explains the fundamental f unetionali y in each area under consideration, and .second, the hook explores those techniques required to achieve maximum perfortnance. In the remainder of this section, we highlight some of the driving factors behind the need to design for performance.

cropro ces or Speed

What gives the Pentium or the PowerPC such mind-boggling power is the relenilc.ss pursuiI,)r speed by processor chip manufacturers. 'L'hc evolulion of these niklehi[ICS continues to hear out Moores Law. mentioned previously. So long as this law holds, chipmakers can unleash a new generation of chips every three years—with four times as many transistors. In mcrhory chips. this hum y uadruliled [he capacity if

dynamic random-access memory (DRAM), still the basic technology for computer main memory. every three years. In microprocessors, the addition of new circuits, and the speed boost that comes from reducing the distances between them, has improved performance four- or five fold every three years or so since Intel launched its x8ti family in 1978,

But the raw speed of the microprocessor will not achieve its potential unless it is fed a constant stream of work to do in the form of computer instructions. Anything that gets in the way of that smooth flow undermines the power of the processor. Accordingly, while the chipmakers have been busy learning how to fabricate chips of greater and greater density, the processor designers must come up with ever more elaborate techniques for feeding the monster. Among the techniques built into contemporary processors are the following:

- Branch prediction: The processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, arc likely to be processed next. If the processor guesses right most of the time, it can prefetch the correct instructions and buffer them so that the processor is kept busy, The more sophisticated examples of this strategy predict not just the next branch but multiple branches ahead, Thus, branch prediction increases the amount of work available For the processor to execute,
- Data now analysis: The processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions, In fact, instructions are scheduled to be executed when ready, independent of the original program order. This prevents unnecessary delay.
- Speculative execution: Using branch prediction and dal a flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution. holding the results in temporary locations. This enables the processor to keep its execution engines as busy as possible by executing instructions that are likely to be needed.

I'h csc and other sophisticated techniques are made necessary by the sheer power Of the processor. They make it possible to exploit the raw speed of the processor.

Performance Balance

While processor power has raced ahead at breakneck speed, other critical components of the computer have not kept up. The result is a need to look for performance balance: an adjusting of the organization and archil eel ore to compensate for the mismatch among the capabilities of the various components,

Nowhere is the problem created by such mismatches more critical than in the interface between processor and main memory: Consider the history depicted in Figure 2. ID. While processor speed and memory capacity have grown rapidly. the speed with whieh data can be transferred between main memory and the processor has lagged badly. The interface between processor and main memory is the most crucial pathway in the entire computer, because it is responsible for carrying a constant flow of program instructions and data between memory chips and the processor. If memory or the pathway (ails to keep pace with the processor's insistent demands. the processor stalls in a wait state, and valuable processing time is lost.

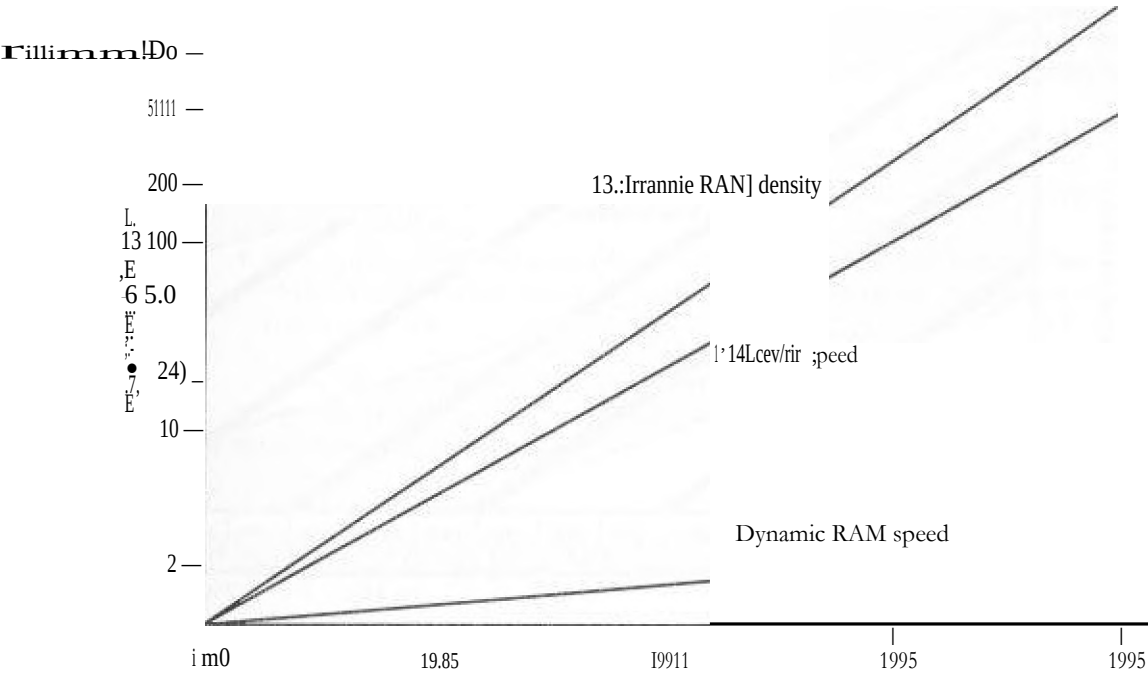
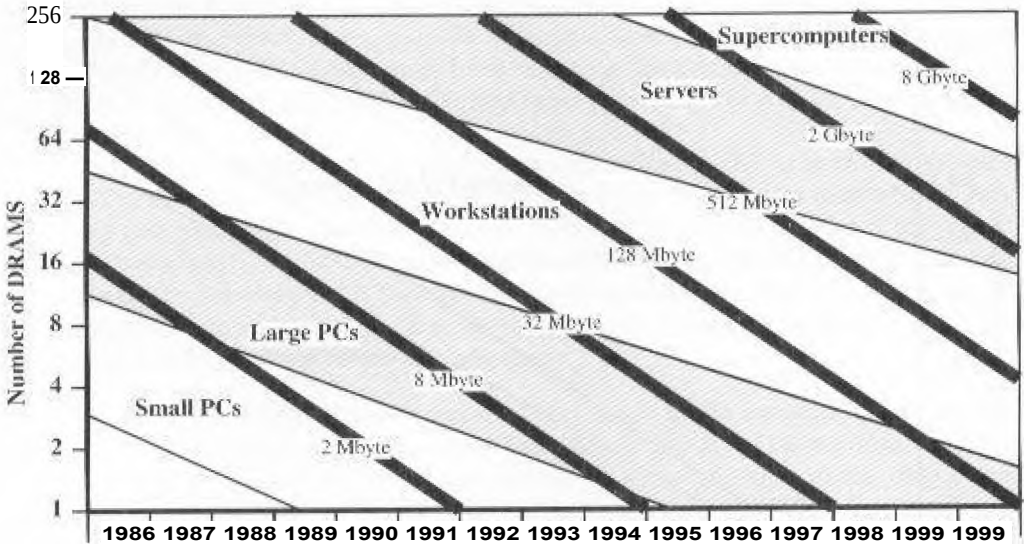


Figure 2.10 Evolution of DRAM and Processor Characteristics

The effects of these trends are shown vividly in Figure 2.11. The amount of main memory is going up, hill DRAM density is going up faster. The net result is that, on average, the number of DRAMs per system is going down. The solid black lines in the figure show that, for a fixed-size memory, the number of DRAMs needed is declining. But this has an effect on transfer rates, because with fewer DRAMs, there is less opportunity for parallel transfer of data, The shaded bands show that for a particular type of system, main memory size has slowly increased while the number of DRAMs has declined.

There are a number of ways that a system architect can attack this problem, all of which are reflected in contemporary computer designs. Examples include the following:

- Increase the number of bits that are retrieved at one time by making DRAMs "wider" rather than "deeper" and by using wide bus data paths,
- Change the DRAM interface to make it more efficient by including a cache or other buffering scheme on the DRAM chip.
- Reduce the frequency of memory access by incorporating increasingly complex and efficient cache structures between the processor and main memory. This includes the incorporation of one or more caches on the processor chip as well as on an off-chip cache close to the processor. chip.
- Increase the interconnect bandwidth between processors and memory by using higher-speed buses and by using a hierarchy of buses to buffer and structure data flow,



DRAM z	.411	l N	,fl	t Nll)	ii.	16 MN(	111"111	6.4I	1111-41	=thit
Predominant organizations		1M x 1 256K x 4		1M x 4		4M x4 2M x8		???		???

Figure 2.11 Trends in Drain Use [PRZY94]

Another area of design focus is the handling of I/O devices. As computers become faster and more capable, more sophisticated applications are developed that support the use of peripherals with intensive 110 demands. Table 2.7 gives some examples of typical peripheral devices in use on personal computers and workstations. 'These devices create tremendous data throughput demands. While the current generation of processors can handle the data pumped out by these devices, lhere remains the problem of getting that data moved between processor and peripheral. Strategies here include caching and buffering schemes plus *the* use of higher-speed interconnection buses and more elaborate structures of huses. In addition, the use of multiple-processor configurations can aid in satisfying 110 demands.

Table 2.7 Typical Bandwidth Requirements for Various Peripheral Technologies

Peripheral	Technology	Required Bandwidth l Mbytesis)
Graphics	**2d**. hit color	30
Local rirca	100 BASEX or FDD1	12
Disk controller	SCSI or P135'4	10
Full-tootion video	11.124 . 768@.30 fps	67-i.
110 peripherals	Othcr r31sc4.-l1aneous	5+

The key in all this is balance. Designers constantly strive to gallants the throughput and processing demands of the processor components. math memory, 110 devices, and the interconnection structures. This design must constantly be rethought to cope with two constantly evolving factors:

- The rate at which performance is changing in the various technology areas (processor, buse=s, memory, peripherals) differs greatly from one type of clement to another.
- New applications and new peripheral devices constantly change, the na.trri.• of the demand on the system in ierms of typical instruetion profile:and ihc data access patterns.

Thus, computer design is a constantly evolving art form. This book atturopts to present the fundamentals on which this an form is based and to present a suney ale current state of that art.

23 PENTIUM AND POWERPC EVOLUTION

Throughout this hook. we rely on many concrete examples of compuler design and implementation to illustrate concepts and to illuminate trade-offs. Most of the time, the book relies on examples from two computer families: the Intel Pentium and the ri5werPC. The Pentium represents the results of decades of design effort on complex instruction set computers (CI CO, Ii incorporates the sophisticated design principles once found only on mainframes and supercomputers and serves as an excellent example of CISC design. The PowerPC is a direct descendant of the first RISC system, the IBM S01. .11-ut is one of the most l:rowel-1111 and hest-designed RISC-based systems on the market,

In this section, we provide a brief overview of both systems.

Pentium

In terms of market share, Into] has ranked as the number one maker of microprocessors for decade t, a posilion it seems unlikely to yield. The evolution of its Hag.-ship microprocessor product serves as 21 good indicator of the evolution of computer technology in general.

Table 2.6 shows that evolution. Interestingly, as microprocessors have grown faster and much more eomplex, Inlcl has actually picked up the pace- In Lel used to develop microprocessors one after another, every four years. But Intel hopes to keep rivals at bay by trimming a year or two off this development time, and has done so with Wu most recent Pentium generalions.

It is worthwhile to list some of the highlights of the evolution of the. Intl product tine.

- 80S0: The world's first general-purpose microprocessor. This VC:IN machine, with an 8-bit data path to memory. The 8080 was used in the first personal computer, the Altair.

- 8086: A far more powerful, 16-bit machine_ In addition 10 a wider data path and larger registers, the 8086 sported an instruction cache, or queue, that prefetches a few instructions before they are executed. A variant of this processor, the 8088, was used in IBM's first personal computer. securing the success of Intel,
- 80286: This extension of the 8086 enabled addressing a I6-MByte memory instead of just 1 MByte,
- 80386: Intel's first 32-bit machine, and a major overhaul of the product. With a 32-bit architecture, the 80386 rivaled the complexity and power of minicomputers and mainframes introduced just a few years earlier, This was the first Intel processor **to** support multitasking, meaning **it** could run multiple programs at the same time.
- 80486: The 80486 introduced the use of much more sophisticated and powerful cache technology and sophisticated instruction pipelining. The 80486 also offered a built-in math coprocessor, offloading complex math operations from the. main CPU.
- Pentium: lAYith the Pentium, Intel introduced the use of supersca]ar techniques. which allow multiple instructions to execute in parallel.
- Pentium Pro: The Pentium Pro continued the move into superscalar organization begun with the Pentium, with aggressive use of register renaming. branch prediction, data flow analysis, and speculative execution.
- Pentium II: The Pentium 11 incorporated Intel MMX technology, which is designed specifically to process video, audio, and graphics data efficiently.
- **Pentium 111:** The Pentium III incorporates additional Floating-point instructions to support 3D graphics software.
- Pentium 4: The Pentium 4 includes additional floating-point and other enhancements for multime.dia.'
- **Hank=** This now generation of Intel processor makes use of a (4-hit organization with the IA-64 architecture, which is discussed in detail in Chapter 15.

PowerPC

In 1975, the 801 minicomputer project at IBM pioneered many of the architecture concepts used in RISC systems. The 801, together with the Berkeley RISC I processor, launched the RISC' movement, The 801. however, was simply a prototype intended to demonstrate design concepts. The success of the 801 project led IBM to develop a commercial RISC workstation product, the RT PC. The RT PC, introduced in 1986, adapted the architectural concepts of Lfic 801 to an actual product. The RT PC was not a commercial success, and it had many rivals with comparable or better performance. In 1990, IBM produced a third system, which built on the lessons of the 801 and the RT PC. The IBM RISC System/6000 was a RISC-like superscalar machine marketed as a high-performance workstation; shortly after its introduction, IBM began to refer to this as the POWER architecture.

'With ihc Pentium 4. Thiel switched from Roman numerals to Arabic numerali, for mixicl numbers.

For its next step, IBM entered into an alliance with Motorola, developer of the M000 series of microprocessors, and Apple, which used the Motorola chip in its Macintosh coroptilcrs. ˈiIte result is a series Or machines that implement the PowerPC architecture. •1 his architecture is derived from 1.he POWER arcbiteel Life, Changes were made to add key missing features and to enable more efficient implementation by eliminating some instructions and relaxing the specification to eliminate sonic troublesome speci411 eases. The resuhing PowerPC architecture is a superscalEu. RISC system, The PowerPC is used in millions of Applc Macintosh machines and in numerous embedded chip applications. An example of the latter is II- M' family of network management chips, which can be embedded in network equipment to provide eornmon Infinal,comeni .Leeos:s fOr users with rnultiVilndor platforms.

The Following are the principal members of the PowerPC family (Table 2.8):

- 601: The purpose of the 6.01 was to bring the PowerPC;.irehitcciurc. to the marketplace as quickly as possible. The 601 is a 32-bit r»achine.
- 603: Intended for low-end desktop and portable computers, II is also a 32-bit machine, comparable in performance with the 601, but with lower cost and a more efficient implementation.
- 604: Intended for desktop computers and low-cnd servers, Again, ihis is a 32-hit machine, but it uses much more advanced superscalar design techniques to achieve greater performance,
- 620: Intended for high-end servers. The first member *of* the PowerPC family to implement a full 64-hit architecture, including 64-bit registers and data paths.
- **740/750;** Also known as the (33 processor. This processor integrates two lcvelS of cache in the main processor chip, providing significant puforrriance improvement over a comparAle machine with off-chip cache organi2ation.
- G4: processor increases the parallelism and internal speed of rhe processor chip.

bihit 2.8 PowcrPC Processor Summary

	601	6031603e	6041604e	7401750 (G3)1	G4
Fast skup dace	1993	L994	1494	1997	199'9
Clock speeds (MEir/.)	50-120	30LI.	1.6.6-350	2.00-36.6	
1_I cach;.:		I[: 1Chv1 hist dmra	32 Khne inst 32 Kbytc dam	32 Kbyt,?.. insrr ?!2 Khym data	kbyre instr 32 gbycc data
1iricksi(1.2 1.2 cachc. support				256 Kbyte -] Mbyte	256 Kbyte••] Mbyte
Nurnher of I I H 11 SE.; I C) if-. I	2.8	I .6-2.t)		6.35	

2.4 RECOmMENDED READING. AND

A description of the IBM 7041)0 series can he found in IBELL714. 'There is good cowl ay. of the IBM 3W) in [SIEWEJ and of the PI)P x and other DEC machines in [BELL78a1. Thew. three hooks also contain numerous &talk:LI examples of other computers spanning the history of computers through the earls/ 1 .?:-;tis, A more recent book that includes an eNctlierit set of ease studies of historicat machines is [BLAA97]. A good hist my of the mieroprocessbr is [BETK971.

One of the best 1.rcaiinents of the Pentium is ISHA N981. The Intel, docunne.imation itself is alsu good [INTL(.111- IE fREV001 provides a good survey of the Intel microprocessor line, with emphasis on the ;i.2-bit machines.

LI1494] is a thorough treatment i he PowerPC architecture- ISHAN951 provides similar coverage. [WEI594.1 Ireats I'oth the POWER and Power.PC architectures.

For interesting discussions of Moore's law and its consequences, see •IU1C961. [SC'HA.97 J, and 1BOH R981.

HEI.1.31a. Bell, C.4 and Newq111,. A. *Cornpulep SiructRres.* ***Readings*** *and En.unpin* New York: MeCiraw-Ili11,

BELL78a Bell. Mlidge,l, and McNamara, J. *Engi.Fremringf A 1)EC Vim.s: of Hardware.* Bedford, MA: Digital Press. 1978.

IRETK97 113e.Lker, ['ern:Ando, J.. and %V W ell, S. "The **Hislory** the MieroprocesscPC *ne.211..ribs Irrilaricaf Journal, Autumn.* 199 l.

IILAA,97 Binauw, G., and Brooks.. P. *(..omiquer Architecture: Cord.cepti add Evethvion.* Reading- MA: Addison-Wesley, 1997.

1101-11198 Bohr, M. 'Silicon 'fiends and Limits for Adk.tiaxecd MieroprocesscFri.. *COM-MliniCaliONS of the ACM . 7litirch* 1998-

BRE'r1X1 *Titd: Intel .19 irpopp.r.rcmors:* 808.6.490.66, .0,786/801*88, 802.96, 80336. 80486, Pitp.inen, liotTheni Pro and PcrnrirzoF 11 Proccssars.* Uppcc Sttddte River, NJ;

FICTC96 Hutcheson and I Ititchesan..1. "'Technology and Economics in the Send-e.i .inductor Industry." *Scienrifk* Arrwricaid, January 19%.

11-1M94 International Busine.s,s Machines, Inc. The *Powe.,f'<' 41 chireertere. rl Spe,ci,itogoot for a Neg... Enmity 1?.M:' j" r494:ussom.* San Francisco.. CA,: 1organ Kaufmann", 1994.

INTE01 bid Corp. *Merl iirchitectrov ScOvar.e. Deueloper'..r Manual volumes).* Document 24.5470 and 14547.1. Aurora. CO. 2000.

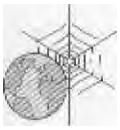
SCIIA97 Schaller. R. "Moores Law; Past, Ptesorrt, and Future."' *Sprvirten.* Jane *19[4]r1.*

SHAMS **Shaulcy, T.** *1'ovi...(71[1](.' Syvirrn t A n'hiter.e.rwe.* Reading. NIA.; Addison rWailey, 1095.

SHAN98 *Pro and Pentium 11 Sy y.rem ei.r.chire.crakre.* Rai:R.411g% MA: Addison-Wesiey. 1998.

SIEW82 Siewiorek. **D.; Boll. C.:** and A. *Crm puter Srrrrclures, PrinCiPleS and 1-..5rample.v. New* York: 1982.

WIL:IIS94:1 Weiss,-S-, cud Sruidi, J. POWER *road* Poli:erPC San PairicisN): rdorgail lCaral-rnann,



Recoinmentied Web Sites:

- **Intel Developer's ['loge:** Intel's]Ve1.3 page for developers! provides a starting point for accessing Pentium informarion. Also includes the Technology .Tournal.
- **PoTherPC: Two likb.** **tine by rvlo1orola kirril** one. by [BM. roc the PowerPC.
- **Top501111 Supercomputer Site:** Prcrvides brici description of architeclure and organization of current supercomputer products, plus comparisons.
- Charles BabliPsige PreivideE, 'Mks 10 a number of Web sites dealing with the hisiory of computers.

2.5 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Tering

accurnulaior (AC)	nisi ruction register (I1{)	OpcoJe
arithmetic and Logic unit (ALL)	inStrUctiC ' ll	ParikliBB equipinom manufacturer (0E:1/4,1)
chip	integrated eircuil. (IC) memory	program control unit
	I **Elenlciry a** ddress	1nro)2rant cottritcfr (PC)
execute cycle	register (MAR)	program coinputur
fetch cycle	rrrcrnc Fry btiffer	iipward **com** patible
(I/0)	re i.,tcc (MI3R)	111 Neltimnit machine
instructic *buffer*	microprocessor	water
register (111(?).	multiplexor	word
inurniction cycle		

Review Questions

2.1 What is a stored program computer?

2,2 What are the four main compunents ulrtiny general-purpose computer?

2.3 At the integrated circuit level. what are the1hree principal *constituen* **ts at a** computer syster0

2.4 Explain Moores law.

Lisl and 111; kcy chaructcrislics of a coniputor

2. What is the k.Ly distinguishing feature of a microprocessor?

Problems

11e1 A = All), A(2), A(.1.00'0) and P = B 1), B(2) B(10(f0) be two vector.; (one-ditmnsiontil arrays) 4.:41rnprising 1000 numbers each (hal aro to be added to form an array C such alai C(L) = A(1) + It I for = 1. 2 1.0%. Using the TA; instruclion sot, write a program For this problem.

2.2 In the **IBM** 3.60 Models 65 and 75, addresses arc striocred in two separate **ma** in memory units (e.g., all even-numbered words in onc unit and all odd-numbered words in kinolher). Vilhat might be the purpose Or this technique?

PART TWO

The Computer System

ISSUES FOR PART TWO

A cCaliputar syštetn COn'sists of processor, m eony, 170, miercon-nections among these major components, With I he t;'1.c't2.PkiUlli ut 111, proc.cN-sor. which is suFficienily compkx to (Acyclic Piti t. lho.c..to..it.$...s tidy, Vaal r.wo oft te.1;: Oip JiI for .A

40:4$44-4:414:C.

Chapter 3 A View of Computer **Function and Interconnection**

At a top level, a computer Con its of ü process6r. memory. .t..firrd il acornpo-'lents. The functional behavior of thc.ziystern con;,isrsof the ex.cillmge of data a rid c4 ini.F.o[si ot[s among ihesc componettts. To opal this cxchango, these components must be interconnected. C:hapter 3 1. pogins with a brief examination of the computer's components and their inpui-oniput re tirni nt, Thu chaplxr then look at key issues that affect interconnection design, especially the need to support intert.upts. The bulk of the chapictr is devoted *to* a siudy of the. tn95t co MD1011 approack tg. inte.r.connectign; the use. of A s,trg.e4tirr.: L2J.J$e• .;

CThapter 4 Cache Memory

Computer merriory exhibilIF. a wide range or type, 1cehno1ogy, organization, perform:la [tee. tau] cost. The typical computer system is equipped with a hierarchy of memory subsystems. sortie internal (dircei Ey ziccessi Mc by the procesNor) mid some (Nicrrial (acees..siblc by the proccs.sor ir'w an 110 modu[). Chapter 4 begins with an overview of this hiorarchy. Next, the ehapiet

deals in detail with the design of cache memory, including. separate. code and data caches and two-level caches.

Chapter 5 Internal Memory

The design of a main memory system is a never-ending battle among three computing design requirements: large storage capacity. rapid access lime, and low cost_ Asmemory technology evolves. each of these three characteristics is changing, so that the design de.cisions in orgzmizing main memory must he revisited anew with each new implementation. Chapter 5 focuses on design issues related to internal memory, First. the nature and organization of semiconductor main memory is examined. Then, recent advanced DRAM memory organizations are explored.

Chapter 6 External Memory

For truly large storage capacity and for more permanent stora2e than is available with main memory, an external memory organization is needed. The most widely used type- of mini memory is magnetic disk, anti much of Chapter 6 concentrates on this topic. first. we look at magnetic disk technology and design considerations_ Then_ we look at the use of RAID organization to improve disk mernory performance. Chapter 6 also examines optical and tape storage.

Chapter 7 Input/Output

110 modules arc interconnected with the processor and main memory, and each controls one or more external devices. Chapter 7 is devoted to the various aspects of organization, This is a complex area, and less well understood than other areas of computer system design in terms of meeting performance demands. Chapter 7 examines the mechanisms by which an 110 module. interacts with the rest of the computer system, using the techniques of programmed PO, interrupt 1/0. and direct memory access (DMA). The interface between an lit) module and oNlyrnal devices is also described.

Chapter 8 Operating System Support

A detailed examination of operating systems ((As) is beyond the scope of this book. However. it is important to understand the Nisk. [unctions of an operating system and how the OS exploits hardware to proVidt• the desired performance. Chapter describes the. basic principles of operating systems and discusses the specific design features in the computer hardware intended to provide support for the operating. system, The chapter begins with a brief history; which serves to **identify** the major types of Operating systems and to motivate their use Next, multiprogramming is explained by examining the long-term and short-term scheduling functions. Finally, an examination of memory management includes a discussion of segmentation, paging. and virtual memory.

CHAPTER

A TOP-LEVEL VIEW OF COMPUTER FUNCTION AND INTERCONNECTION

3.1 Computer Components

32 Computer Funetiiui

- Instruction Fetch and Execute
- Interrupts
- 110 Function

3.3 Interconnection Structures

3.4 Bus Interconnection

- Bus Structure
- Multiple-Bus Hierarchies
- Ylc **ren** ts of Bus Design

33 PCI

- Bus Structure
- PCI Commands
- Data Transfers.
- Arbitration

3.6 **Recommended Reading and Web Sites**

3.7 Key Terms, Review Quesiions, and Problems

- Key Icrms
- Review Questions
- Problems

Appendix 3A Timing Diagrams

KEY POINTS

- An instritelion cycle consists of art instruction fetch, followed by zero or snore operand fetches, followed by zero or more operand stores, followed by an interrupt check (if intermpts are enabled).
- The major computer VS Lern components (processor, main memory. 1../0 modules) need to be interconnected in order to exchange data litad control signals. 'Flee most popular means of interconnection is the use of a shared system bus consisting of multiple lines. In contemporary systems, there typically is a hierarchy of buses to improve performance..
- Key design el ements for buses include ii bitrat ion (whether permission to send signals on bus lines is Controlled centrally or in a distributed fashion); liming (whether signals on the bus are synchronized to a central clock or are sent 4iti ptchronously based on the most recent transmission); and width (number of address Lines and number of data lines).

At, top level, a computer consists of CPU (central processing unit), memory. iicl 1/0 components, with one or more modules of each type. These components are interconnected in some fashion to achieve the basic function of flit: Loin puter, which is to execute programs. Thus, at a top level. we can describe a computer system by (I) describing the em erna I behavior of each component. that is, the data and control signals that it exchanges with other components; and (2) describing the interconnection structure and the controls required to manage the use of the interconnection structure.

This top-level view of structure and function is important because of its explanatory power in understanding the nature of a computer. Equally important is its use to understand the increasingly complex issues of performance evaluation. A grasp of the top-level structure and function offers insight into system bottlenecks, alternate pathways. the magnitude of ',>.stem failures **if** a component fails, and the ease of adding performance enhancements. **In** many cases, requirements for treater system power and fail-safe capabilities are being met by changing the design rather than merely increasing the speed and reliability of individual components.

This chapter focuses on the basic structures used for computer component interconnection. As background, the chapter begins with a brief examination of the basic components and their interface requirements. Then a functional overview is provided. We are then prepared to examine the use of buses to interconnect system components.

3.1 COMPUTER COMPONENTS

As discussed in Chapter 2. virtually all contemporary computer designs are based on concepts developed by John von Neumann at the institute for Advanced Studies,

Princeton. Such a design is referred to as the *von N nano no a all i etcare* and is based on three key concepts:

- Data and instructions are stored in a single read—write memory,
- The contents of this memory are addressable by location, without regard to the type of data contained there.
- Execution occurs in a sequential fashion (unless explicitly modified) frailLone instruction to the next.

The reasoning behind these concepts was discussed *in* Chapter 2 but is worth summarizing here. 'There is a small set of basic logic components that can be combined in various ways to store binary data and to perform arithmetic and logical operations on that data. If there is a particular computation to be performed, a configuration of logic components designed specificativ for that computation could be constructed. We can think of the process of connecting the various components in the desired configuration as a farm of programming. The resulting "program" is in the form of hardware and is termed a *ha rilw red program.*

Now consider this alternative. Suppose we construct a general-purpose configuration of arithmetic and logic functions. This set of hardware will perform various functions on data depending on control signals applied to the hardware. In the original case of customized hardware, the system 4icuci-Pb.; data and produces results (Figure 3.1a). With general-purpose hardware, the system accepts data and control signals and produces results. Thus., instead of rewiring the hardware for each new program, I he programmer merely needs. to supply a new set of control signals.

Flow shall control signals be supplied? The answer is simple but subtle. The entire program is actually a sequence of steps. At each step, some arithmetic or logical operation is performed on some data. For each step, a new set of control signals is needed. Lc1 us provide a unique code for each possible set of control signals, and let us add to the general-purpose hardware a segment that can accept a code and generate control signals (Figure :Lib).

Programming is now much easier. Insiead of rewiring the hardware for each new program, ail we need to do is provide a new scqucncc of codes. Each code is, in effect, an instruction. and part of the hardware interprets each instruction and generates control signals. To distinguish this new method of programming. a sequence of codes or instructions is called Nernware..

Figure 3.1b indicates two major components of the system: an instruction interpreter and a module of general-purpose arithmetic and logic functions. These two Constitute the CPU. Several other components are needed to yield a functioning computer. Data and instructions must he put into the system. For this we need some sort of input module. This module contains basic components for accepting data and instructions in some form and converting them into an internal form of signals usable by the system. A means of reporting resulls is needed, and this is in the form of an output module. Taken together, these are referred to as //C) CO/Hp:Men M.

One more comroiwnt is needed - An input device will bring instructions and data in s.ccittenlially. But a program is not invariably executed sequentially; it may jump around (e.g., the 1AS jump instruction). Similarly, operatiouN on data may require access to more than just one element at a time in a predetermined sequence.

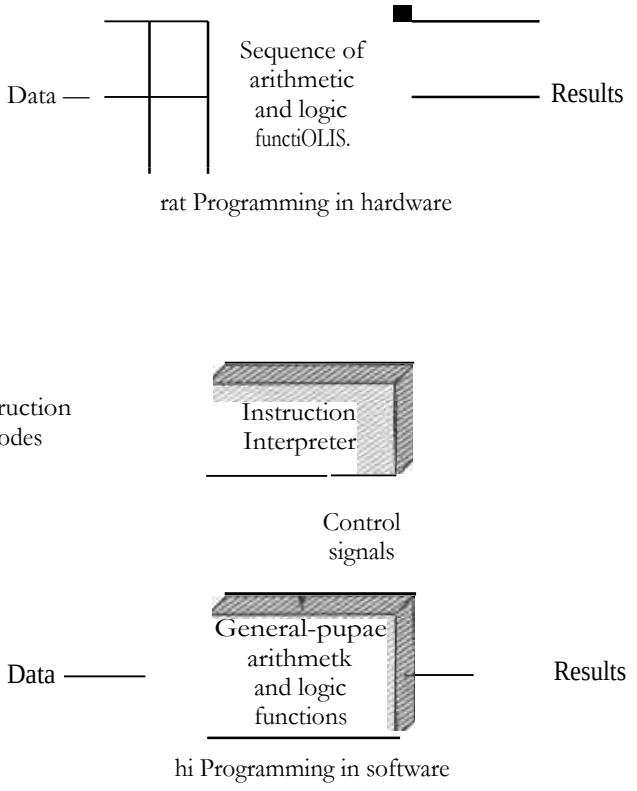


Figure 3.1 Hardware and Software Approaches

Thus, there must be a place to store temporarily both instructions and data. That module is called *memory,* or *main memory* to distinguish it from external storage or peripheral devices. on Neumann pointed out that the same memory could he used to store both instructions and data.

Figure 3.2 illustrates these top-level components and suggests the interactions among them. The CPU exchanges data with memory. For this purpose, it typically makes use of two internal (to the CPU) registers: a memory address register (MAR), which specifies the address in memory for the next read or write. and a memory buffer register (NeIBR), which contains the data to be written into memory or receives the data read from memory. Similarly, an 110 address register (I/0AR) specifies a particular device. An buffer register (.110BR) is used for the exchange of data between an module and the CPU.

A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data. Anil() module transfers data from external devices to CPU and memory, and vice versa. **It** contains internal buffers for temporarily holding these data until they can be sent on.

having looked briefly at these major components, we now turn.to an overview of how these components function together to execute programs.

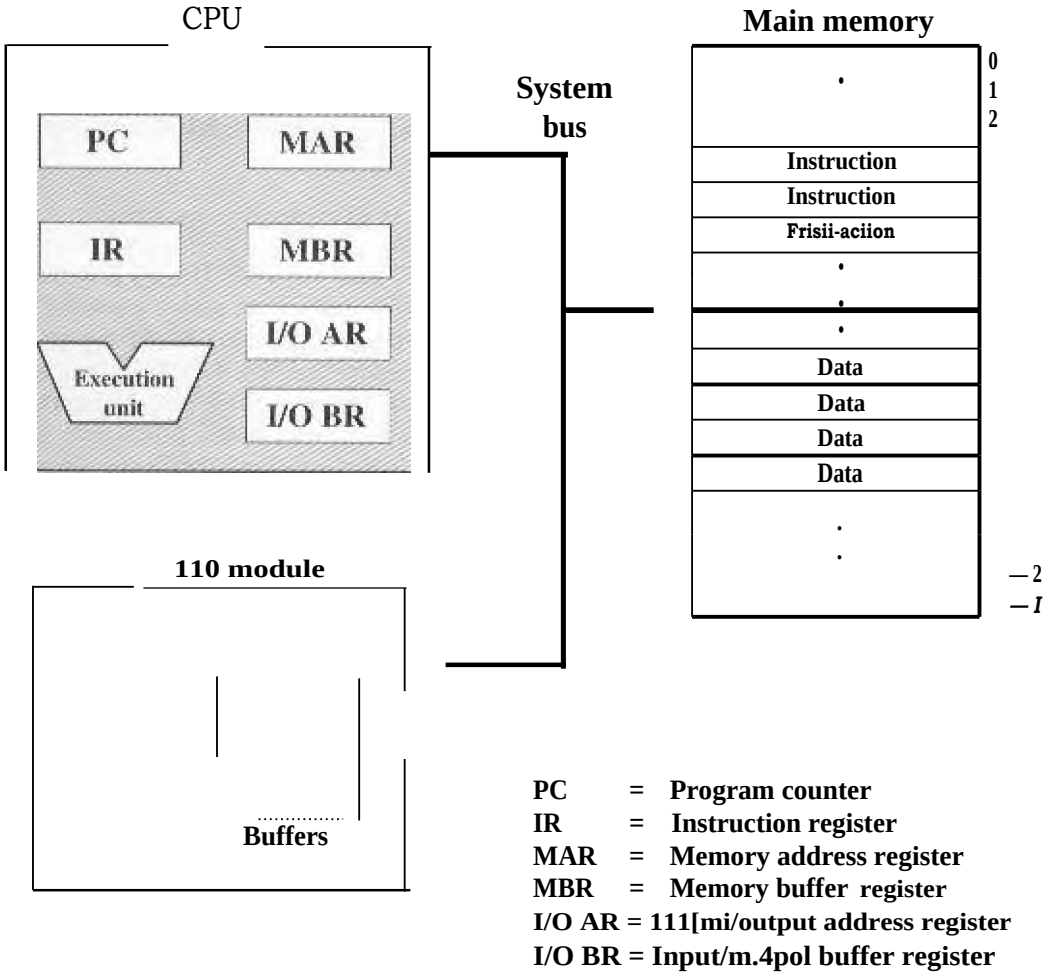


Figure 3.2 Compute.r C(irlivonc[thi; Fop-L4.2vel **Vi** l2 w

3.2 C6MPUTER

The bask function **performed by** a computer is execution of I, progrtml, **which consists** of a set of instructionz'. slorcd in memory. The processor does the actual work by executing instructions specified in the pmgram. This section provides an overview or the key elements of **program excuution. In its 7:limplc.s1** fc..1rni. instruction processing consists of two steps: The processor reads *(reicher.v)* instructions from memory one at a time and executes each instruction. Program execution consists of repeating **the prof.:(3ss** of instruction fetch and instruction execution, r]'he instruction execution may involve several operations and depends on **the** nature of the instruction (see, for example, the **iLhicr portion** of Figure 2.4).

Thu processing required for a single instruction is called an *instruction cycle.* Using the simplified two-step description given previously. the instruction cycle is depicted in Figure 3,3. The two step RTC referred to as the *fetch cycle* and the *execute cycle.* Program execution halts only if the machine is turned off, some sort of LUITCce verablQ. error occurs, or a program instrucl ion nal halts the computer is encountered,

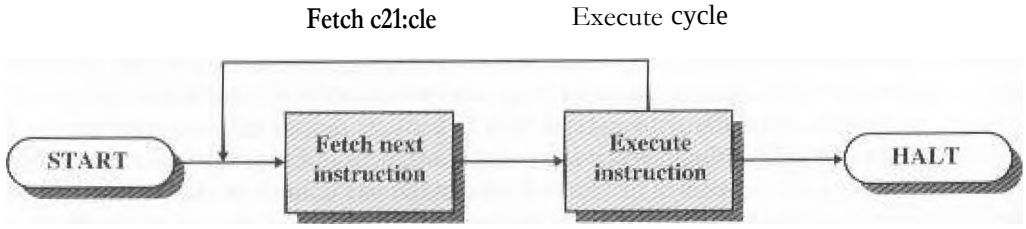
Instruction Fetch and Execute

Ai the beginning of each nisi ruction cycle, the processor fetches an instruction from memory, Ina typical processor. a resister **called** the **program** am-Later (PC) holds the address of the instruction to be fetched next. Unless told otherwise, the processor always increments the PC after each instruction fetch so that it will fetch lhe next instruction hi sequence (i.e., the instruction located at the next higher **memory** address). So. for example, consider a computer in which each instruction oecupies one 145-bil word of memory, Assume that the program counter is set to location 300. The processor will next fetch the instruction at location 300. On succeeding instruction cycles,. it will fetch instructions from locations 301. 302. 303, and so on. This sequence may be altered, as explained presently.

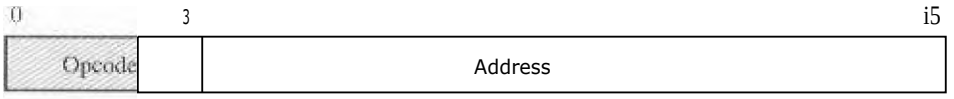
The fetched instruction is loaded into a register in the processor **known** as the instruction regisier **(I R).** The instruction contains bits that specify the action lhe processor is to take. The processor interprets the instruction and performs the required action. In general, these actions fall into four.eategorics:

- **Processor-meroory: Data** may be transferred from processor **to** memory or from memory to processor.
- **Proceskor4/0;** Data may he transferred to **or** from a peripheral device by transferring **between** the processor **and an 1 0 module.**
- **Data processing; The** processor may perform some arithmetic or logic operati on on **data,**
- **Control;** An instruction may specify that the sequence of execution be alicred. For example, the processor may fetch an instruction from loetition 149, which specifies that the next instruction be from location 182. The processor will **remember** this fact by setting the program counter to 182. Thus, on the next fetch cycle, the instruction will be fe1ched from location 182 rather than 150.

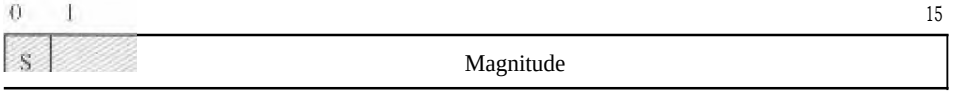
An instruction's execution may involve a combination of these actions.



Fipire **3,3 Basic hisEruction** Cycle



(a) Instruction format



(b) Integer format

Program counter CPC b = A dtlrexs cFt instruction
instruction m2 g (JR 1 = oNlroction being xcruted
Ac uniitialor (AC) = Temporary storage

Internal CPU registers

MD I = Load. AC From
= Siort AiC lo memory
0101 = Add to AC from memory

111) P'arnal of oracles

M C'haracieristics of a Hy poihetieal Machina

COnsickT a simple example using a hypothetical machine that includes the characteristics listed in Figure 3.4, The processor contains a single data register. called an accumulator (AC). Both instructions and data are 1.6 bile long- Thus, it is convenient to organize memory using 16-bit words. The instruction format provides 4 hits for the c)peode, so that there can be as many as 2^4 — lb different opcodes, and up to 2' = 4096 (4K). words or memory can be directly acldressed.

Figure 3.5 illustrates a partial program exeCuLion, showing ihu rcluvant portions or memory and processor reRisters. The program fragment shown adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter to Three instructions, which can be described as three fetch and three execute cycles, are required:

1. The PC contains 300 the address of 1he rinst instruction. This instruction (the value 1940 in hexadecimal) i loaded into the instruction register IN. and 1hc PC is incremented, Note that this process involves the use of a memory address register (Nel AR) and a memory buffer recister (MBR). For simplicity, these intermediate registers are ignored.

---

[1] HAkitlucirli;t11101:Jlion ik IHLN.I. in which cad] digit represents 4 bits. This is the most convenient notation for rcpmEenling the corslcitts of ttlemory kind rlfgird.vrs when Llic word length is a multiple Or 4. See Appendix ri rcircsiler on number Eystoras Wedmal. binary. hexadecimal).

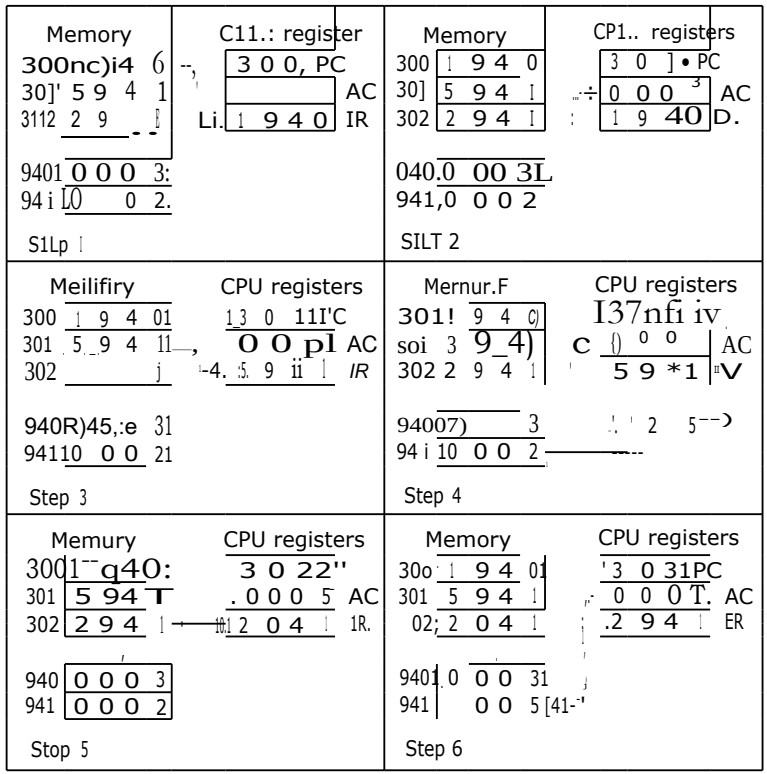


Figure 3.5 EXaillpL of Program Executinn (contents of memory and registers in Imaidecimal)

2. The first 4 bits (first hexadecimal digit} in the IR indicate tali the AC is to be loaded, The remaining 12 hits (three hexadecimal digits) specify the address (940) from which LI4.iia ate to.be loaded.
3. The next instruction (504 I) is fetched from location 301 and the PC is incremented.
.4. The old contents of the *AC* and the contents of location 941 are added and the result is stored in the AC.
5. The next instruction (2941) i fetched from location 302 and the PC is incremented.
6. The contents of the AC are stored in location 941.

In this example, three instruction cycles, eHch consisting of at fetch cycle and an execute. cycle. are needed to add the contents of location 940 to the contents of 941. With a more compiex set of insiructions. fewer eyeios would he needed. Some older processors. for example, included instructions that contain MOrC than one memory ztCidress. Thus the execution cycle for a particular instruction on such processors could involve more than one reference. to memory. Also. instead of memory referancg!,_ an instruction may sped[) an Li0 operation.

Fur example. the P1.)P-11 instruction expressed symbolically as ADD fi,A stores the sum of the contents of memory locations B and A into memory location A. A single inslruction cycle with the roilowina steps occurs:.

- Fetch the ADD instruction.
- Read the contents of memory location A into the. processor.
- Read the contents of memory location 13 inlo the processor. In order ihai contents of A are not lost. the processor must have at least two registers for storing memory values. rather than a single accumulator.
- Add the two values.
- Write the rctsutl, from the processor to memory location A.

Thus, the execution cycle for a particular instruction may involve more Hum one reference to memory. Also. instead of memory references, an instruction may specify an 110 operation. hcse ilddilional considerations in mind, Figure 3.6 provides a more detailed look at the basic instruction cycle of Figure. 3.3. MI; figure is in the form of a state diaaram. For any given instruction cycle, some states may be null and others may be visited more than once. The states can be described as follows;

- **Instruttion address calculation (lac):** Determine the address of the next instruction to be executed. Usually, th is invoivcs adding a fixed number 10 the address of the previous instruction. For example, if each instruction is I6i bits long wnd memory is organized **into** .16-bit words, then add 1 to the previous address. IL instead, memory **is** organized as individually addressable 8-bit bytes, then add 2 to the previous address,
- Instriroimi Fetch **(if):** Read instruction from its memory location into the processor.
- **iristruction operation decoding (100** Analyze instruction to determine type of operation to he performed and operand(s) to be used.
- **Operand address calculation (oac): If the** ope raLion involves 1112 le rencc to rill (Tumid in memory or available via UO, then determine the address of the operand.

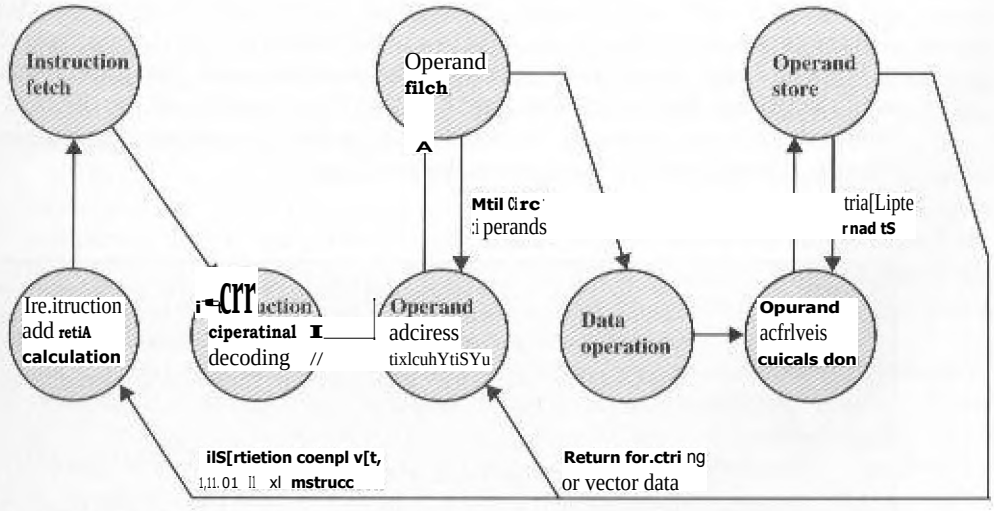


Figure 3.6 Insrruction (.9 yelc Statv. Di.4rrair

- Operand fetch (of): 1:etch the operand from memory or read it in from I/O,
- Data operation (do): Perform the operation indicated in the instruction.
- Operand store (os): Write the result into memory or out to If0.

Stales in the upper part of Figure 3.6 involve an exchange between the processor and either memory or an 110 module, States in the lower part of the diagram involve only internal processor operations. The oac slate appears twice, because an instruction may involve a read, a write, or both, However, the action performed during that state is fundamentally the same in both cases, and so only a single state identifier is needed.

Also note that the diagram allows for multiple operands and multiple results, because some instructions on some machines require this. For example, the P.DN- 1 1 instruction ADD A,B results in the following sequence of states: iac, if. iod, oac, of, oac, of. do, oac, os.

Finally. on some machines, a single instruction can specify an Operation to be performed on a vector (one-dimensional array) of numbers or a string (one-dimensional array) of characters. As Figure 16 indicates, this would involve repetitive operand fetch and/or store operations.

Interrupts

Virtually all computers provide a mechanism by which other modules (I/O. memory) may interrupt the normal processing of the processor, Table 3.1 lists the most common classes of interrupts. The specific nature of these interrupts is examined later in this book, especially in Chapters 7 and 12. However, we need to introduce the concept nov+. to understand more clearly the nature of the instruction cycle and the implications of interrupts on the interconnection structure. The reader need not he concerned at this stage about the details of the generation and processing of interrupts, but only focus on the communication between modules that results from interrupts.

Interrupts are provided primarily as a way to improve processing efficiency. For example, most external devices are much slower than the processor. Suppose that the processor is transferring data to a printer using the instruction cycle scheme of Figure 3.3. After each write operation, the processor must pause and remain idle until the printer catches up. The length of this pause may be on the order of many hundreds or even thousands *of* instruction cycles that do not involve memory. Clearly, this is a very wasteful use of the processor.

Table 11 Classes of Interrupts

PrOgralla	cknerauld try some condition that occurs a8 u mule of an instruction execution, such as arithim tic (werflo•, division by tern, attempt in exccole an illegal machine instrUclion, im reference outndk a ilSe.T'S irmnory space,
Timer	Gen orDi tl liy liTrH2r WlfltIn !he proveNsur. 0142 operating system to perform Certain Functions on a retrular
VO	Generated by an 110 controller, to signiii normal etimpletion nt tin Opc1aLiOn OT to qigi Lill. :1 calie rY I CrTor hdiLiCin..S.
Hardware failure	001Cla by a laiiuro Such in, power failure or mentor parity error.

Figure 3.7a illustrates this state of affairs. The user program performs a series of WRITE calls interleaved with processing. Code segments I. 2, and 3 refer to sequences of instructions that do not involve I/O. The WRIT I-i calk are to an program that is a system utility and that will perform the actual I/O operation · The 110 program consists of three sections:

- A sequence of instructions, labeled 4 in the figure, to prepare. for the actual l.l0 operation. This may include copying the data to he output into a special buffer and preparing the parameters for a device command.
- The actual I.0 command. Without the use of interrupts, once this command is issued, the program must wait for the I/0 device to perform the requested function (or periodically poll the device). The program might wait by simply repeatedly performing a test operation to determine if the 1/0 operation is done.
- A sequence of instructions, labeled 5 in the figure.. to complete. the operation. This may include setting a flag indicating the success or failure of the operation.

Because the 1/0 operation may take a relatively long time to complete, the 110 program is hung up waiting for the operation to complete; hence. the user program is stopped at the point or oh: WRITE call for sonic considerable period of time.

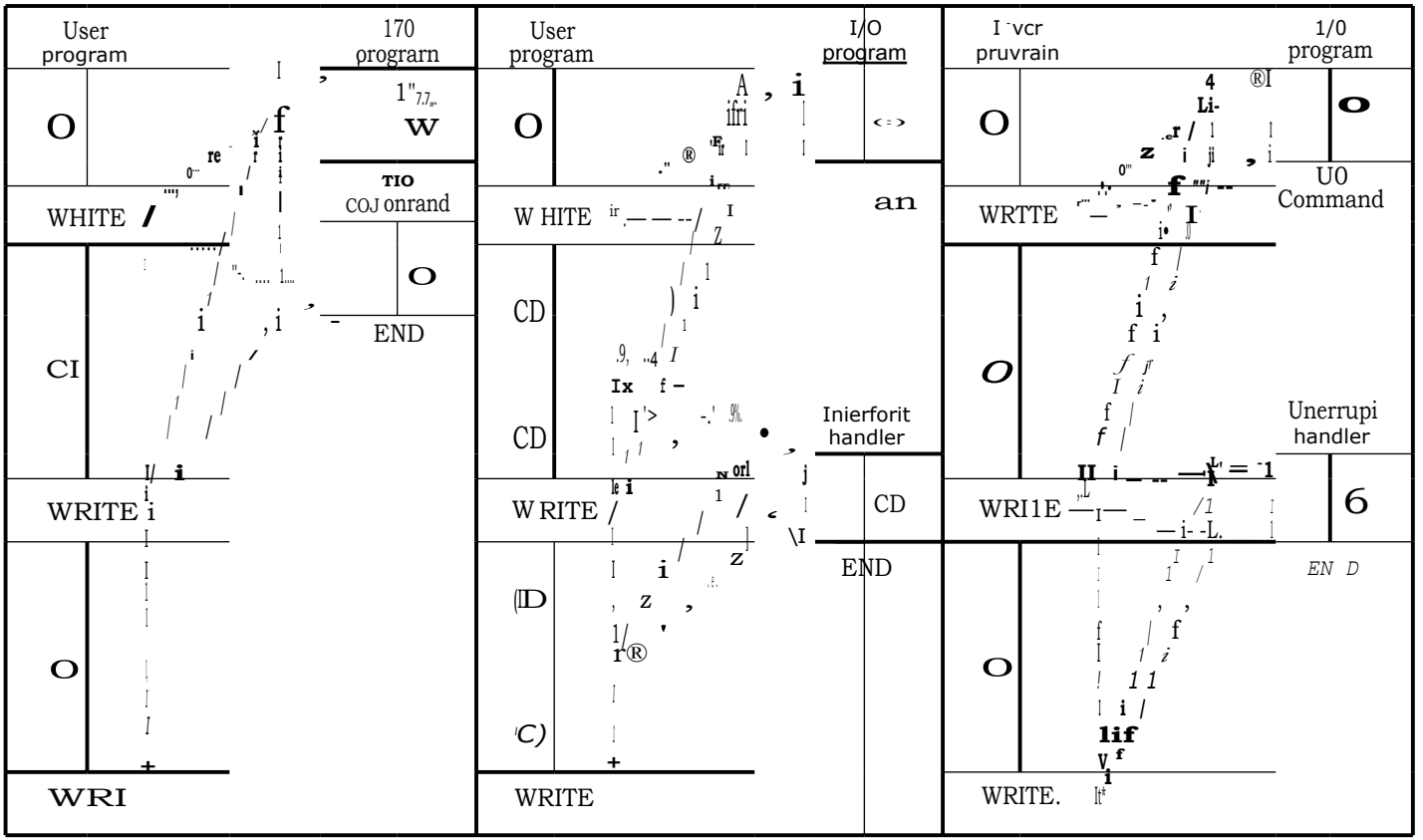
Interrupts and the Instruction cycle

With interrupts, the processor can be engaged in executing other instructions while an 1.0 operation is in progress, Consider the flow of control in Figure 3.7b. As before, the user program reaches a point at which it makes a system call in the form of a WRITE call. 'I program that is invoked in this case consists only of the preparation code and the act ind I/O command. After these few instructions have been executed, contral returns to the user program, Meanwhile, the external device is busy accepting data from computer memory and printing il. This 1/0 operation is conducted concurrently with the execution of instructions in the user program,

When the external device becomes ready to be serviced, that is, when it is ready to accept more data from the processor, the module for that external device sends an *interrupt request* signal to the processor, The processor responds by suspending operation of the current program, branching off to a program to service that particular 1.10 device, known as an interrupt handler, and resuming the original execution after the device is serviced. The points at which such interrupts occur are indicated by an asterisk in 1•igure 3.7b.

From I he point of view of the user program, an interrupt is just that: an interruption of the normal sequence of execution. When the interrupt processing is completed, execution resumes (Figure 3.g), Thus, the user program does not have to contain any special code to accommodate interrupts; the processor and the operating system are responsible for suspending the user program and then resuming it at the same point.

To accommodate interrupts, an i *nterrtpt cycle* is added to the instruction cycle, as shown in Figure 3.9. In the interrupt cycle. the processor checks to *see* if any interrupts have occurred. indicated by the presence of an interrupt signal. If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction of the current program. If an interrupt is pending, the processor does the following:



(a) No hive rru p Ls (h) Interrupts; shizirt UO wait 11..) Lauf·opts: lon2 1/0 wait

figure 3.7 Program Flow of Cuntli:11 withom and with Iniefropts

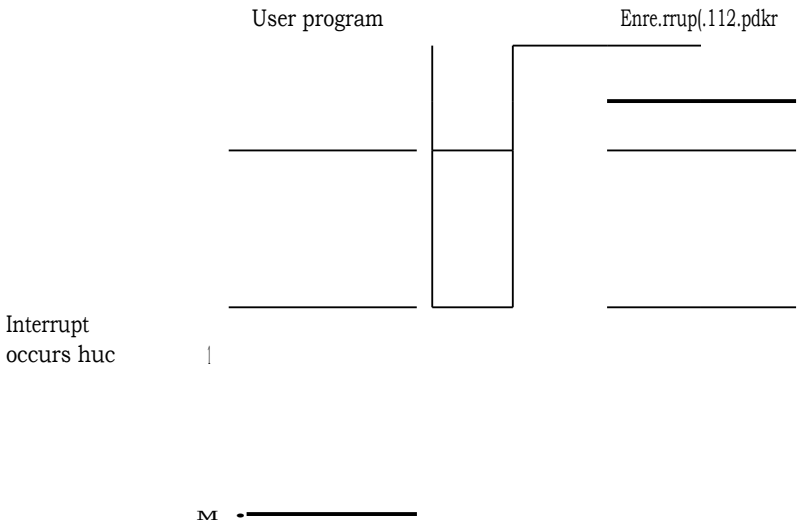


Figure 18 Transfer of C'ontroll. via Inte.rrupis

* It suspends execution of the current program being executed and saves its context. This niearm solving i he otidress 0r Ihe next inslruclion to be executed (current contents of the prognim counter) and any other data relevant to the processor's current activity.
- It sets the pro gram counter to the starting address of an i6'21**07010** *handler to*

The fare cc.ssor now proceeds lo lie fetch cycle and fetches Ihe first instruction in the interrupt handler program, which will service the inie mpt. The interrupt handler program is generally part of the operating system. Typically, this program determines the nature of the interrupt and performs whatever actions are needed. **In** th.c cxample 'se have been using, the handler determines which

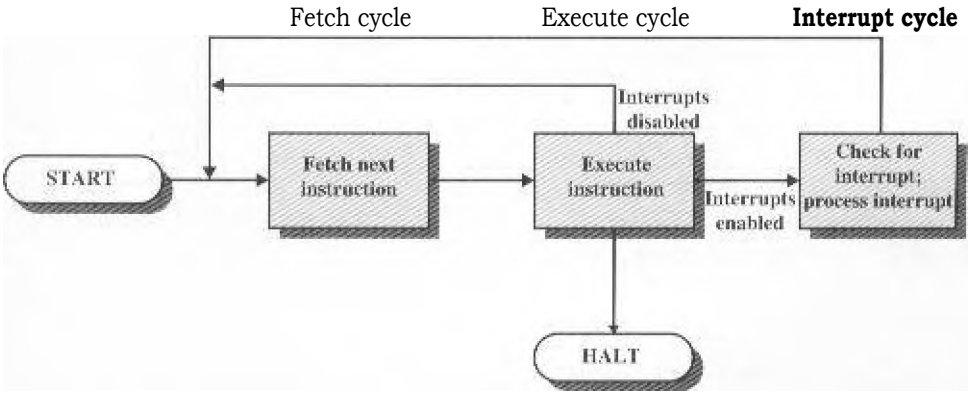


Figure 3.9 Instruction Cycle. with Interrupts

1;0 module generated the interrupt, and may branch to ,o program that will write more data out to that 1;0 module. When the interrupt handler routine is completed, the processor can resume execution of the user program al the point of inter r u pti

It is clear that there is some overhead involved in this process. Extra instructions must be executed (in the interrupt handler) to determine the nature of the interrupt kind to decide on the appropriate action. Nevertheless, because of the rel- large amount of time that would he wasted by simply waiting on an 110 operation, the processor can be employed much more efficiently with the use of interrupts.

To appreciate the gain in efficiency, consider Figure 3.10, which is a timing diagram based on the Mow of control in Figures 3.7a and 3.7b. Figures 3.7b and 310 assume that the lime required for the 110 operation is relatively Short. less than the time to complete the execution of instructions between write operations in the user program. The more typical case, especially for a slow device such as a printer, is that

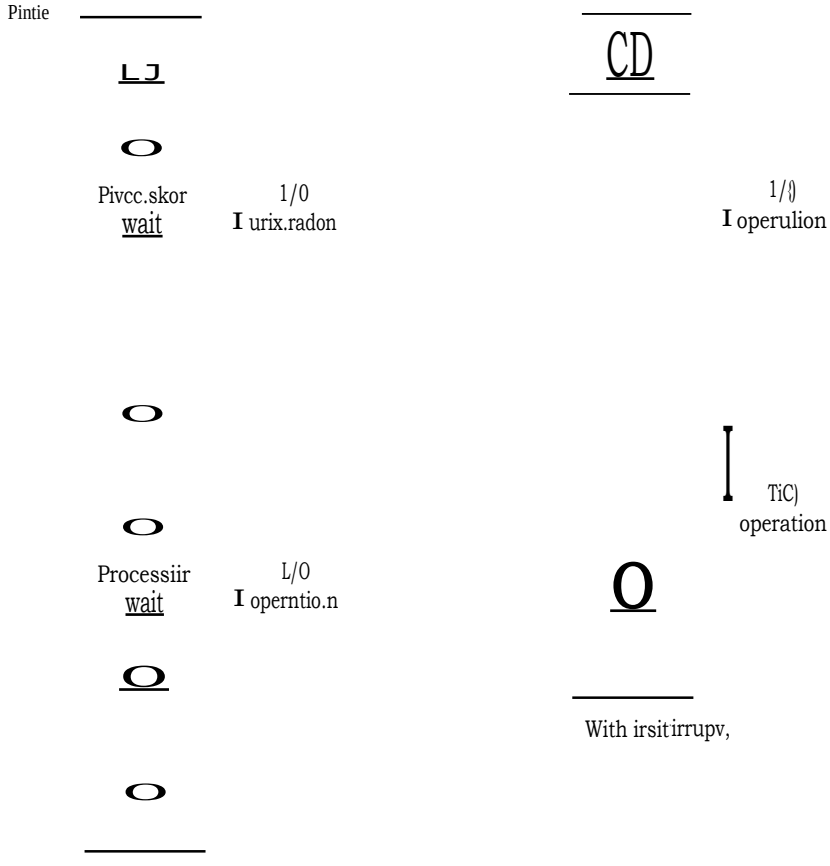


Figure 3.10 Progrnrn Timing... Short ,I+0 Wait

the 1/C) operation will take much more time than executing a sequence of user instructions. Figure 1.71: indicates this state of affairs. In this case, the user pro ram reaches the second WRITE call before !he I/O operation spawned by *the* first call is complete:. The result is that the user program is **hung** up at that point. When the pyeceding 1/0 operation is completed, this new WRITE. cal! may be processed, and a new 1.10 operation moy he started. Figure 3,11 shows the timing for this situation with and without the use of interrupt_ We can sec shat there is still a gain in efficiency because part of the time durin.g which the I/0 operation is underway overiaps with the execution or user instructions.

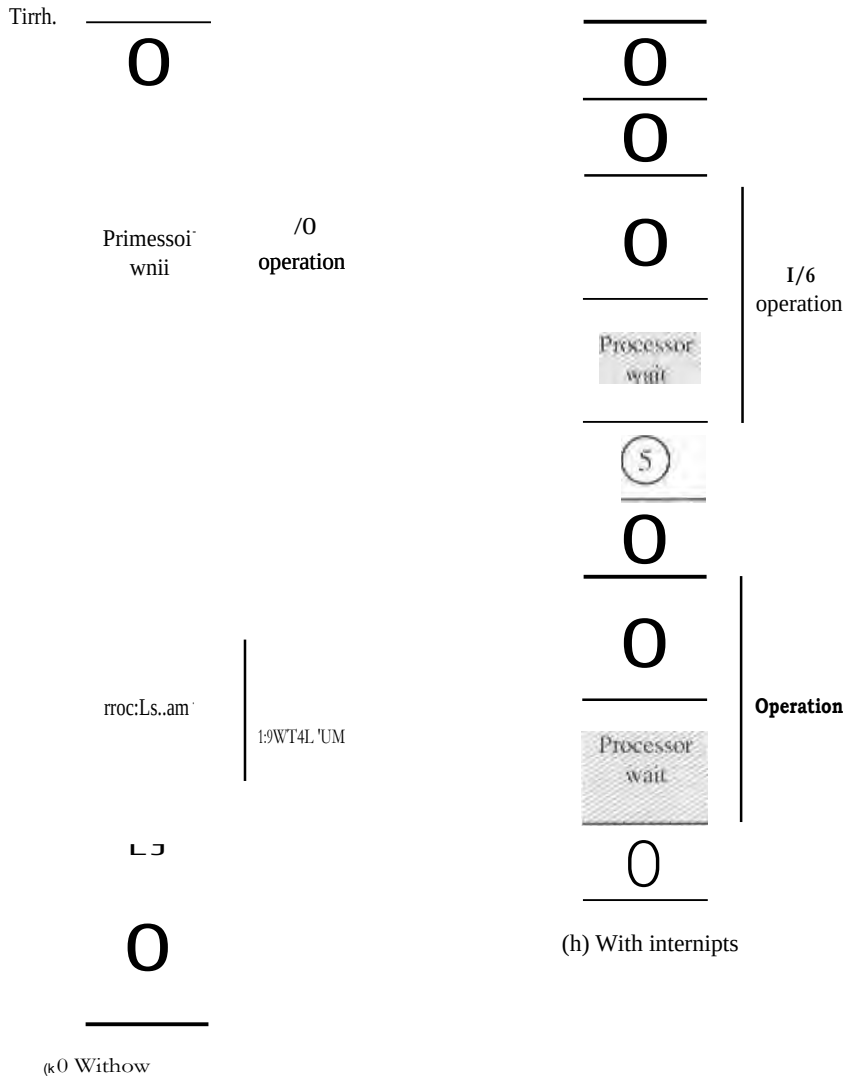


Figure 3.11 PI'1.1.4,10 in Timing: Long 1.0 Wait

Figure 3.12 shows a revised instruction cycle state diagram that includes interrupt cycle processing.

Multiple Interrupts

The discussion so far has focused only on the occurrence of a single interrupt_ Suppose, however, that multiple interrupts can occur, For example, a program may be receiving data from a communications line and printing results. The printer will generate an interrupt every time that it completes a print operation. The communication line controller will generate an interrupt every time a unit of data arrives. The unit could either be a single character or a block, depending on the nature of the communications discipline. In any case. it is possible for a communications interrupt to occur while a printer interrupt is being processed.

Two approaches can be taken to dealing with multiple interrupts. The first is to disable interrupts while an interrupt is being processed_ A *disabled interrupt simply* means that the processor can and will ignore that interrupt request signal. If an interrupt occurs during this time. it generally remains pending and will be checked by the processor after the processor has enabled interrupts_ 'Thus, when a user program is executing and an interrupt occurs, interrupts are disabled immediately. After the interrupt handler routine completes, interrupts are enabled before resuming the user program, and the processor checks to see il additional interrupts have occurred. This approach is nice and simple, as interrupts are handled in strict sequential order (Figure 3.13a).

The drawback to the preceding approach is that it does not take into account relative priority or time-critical needs. For example, when input arrives from the communications line, it may need to be absorbed rapidly to make room for more input_ If the first batch of input has not been processed before the second batch arrives, data may be lost_

A second approach is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to be itself interrupted (Figure 3.13b). As an example of this second approach, consider a system with three 110 devices: a printer, a disk, and a communications line, with increasing priorities of 2, 4, and 5, respective]y. Figure. 3_14 illustrates a possible sequence. A user program begins at r = 0. At t = 10, a printer interrupt occurs; user information is placed on the system stack and execution continues at the printer interrupt service routine (ISIS). While this routine is still executing, at t = 15, a communications interrupt occurs. Because the communications line has higher priority than the printer, the interrupt is honored. The printer ISR is interrupted, its stale is pushed onto the stack. and execution continues at the communications !SR_ While this routine is executing, a disk interrupt occurs (r = 20). Beeau',e this interrupt is of tower priority. it is simply held, and the communications ISR runs to completion.

When the communications 1SR is complete (t 25). the previous processor state is restored. which is the execution of the printer 'SR. However, before even a single instruction in that routine can be executed, the processor honors the higher-priority disk interrupt and control transfers to the disk ISR. Only when that routine is complete (t 35) is the printer ISR resumed_ When that routine completes (r — 40), control finally returns to the user program.

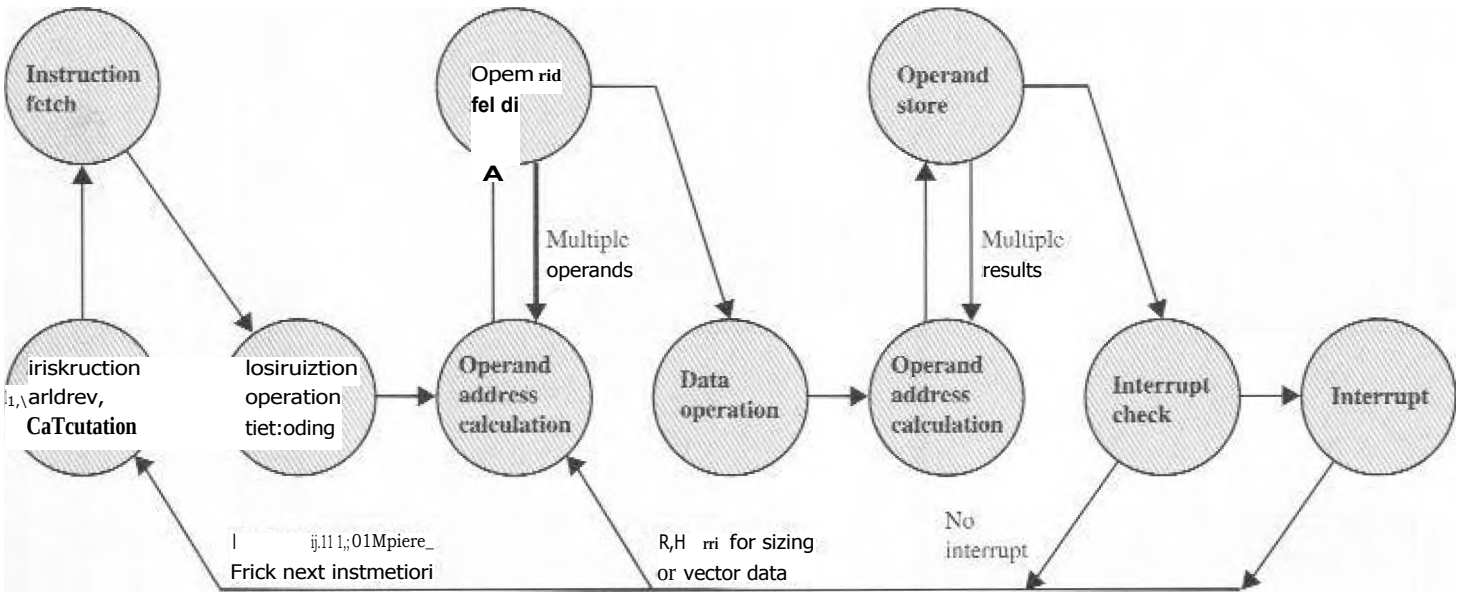
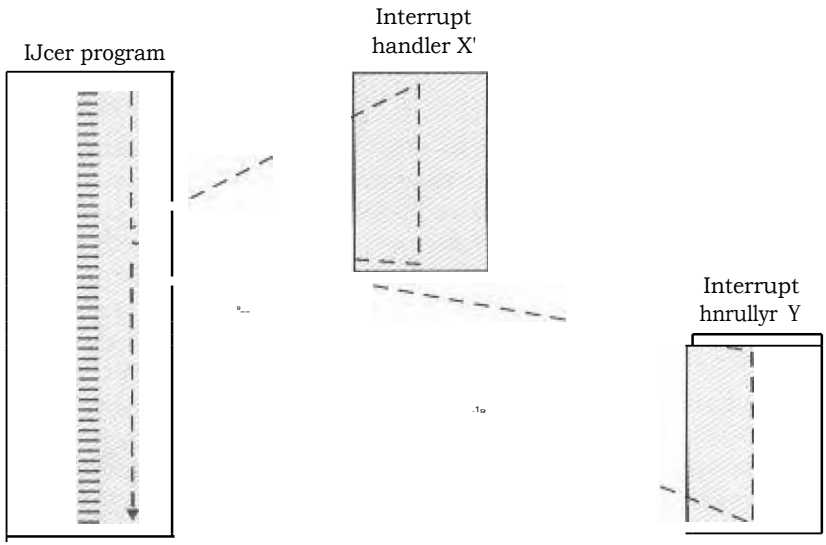
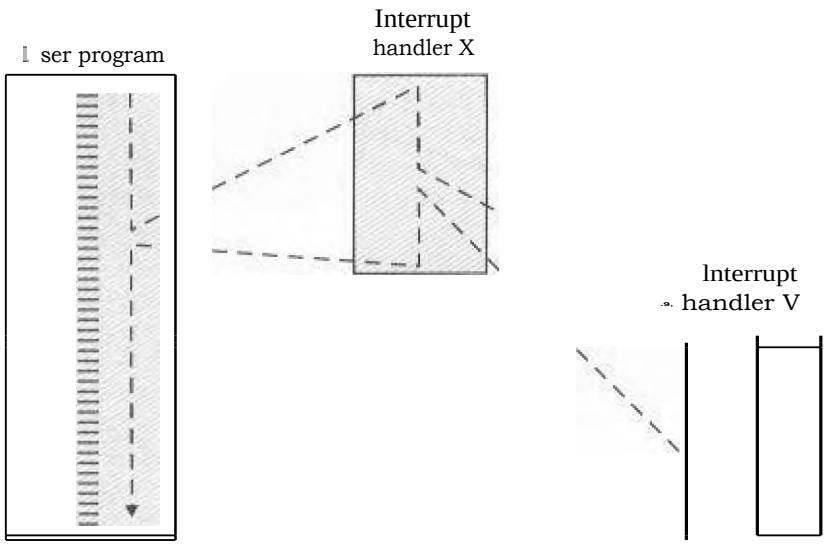


Figure 11-2 Distruetiod Cyck State Diagram, with Toternios



la I. Sequential inierrupt processing

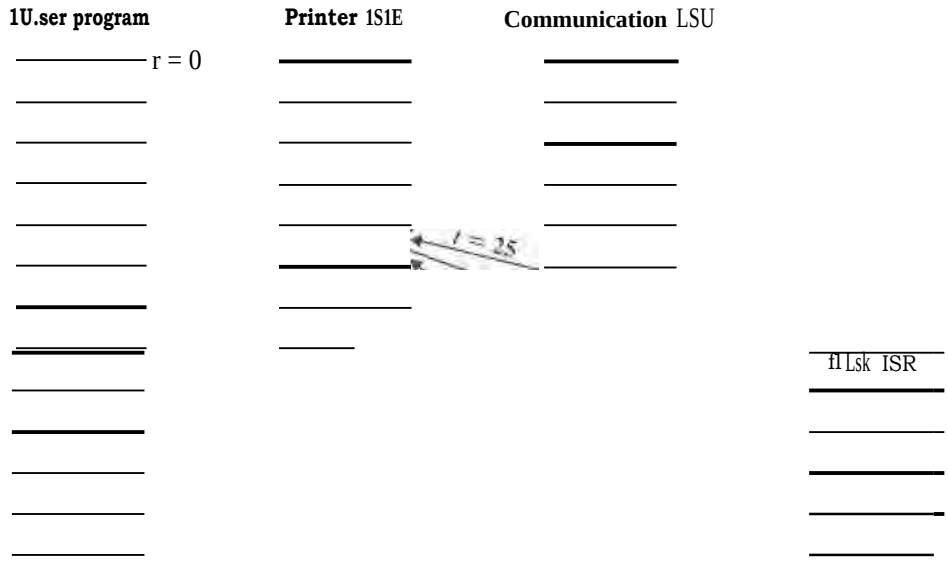


.1,11 Nested interrupt processing

Figure 3.13 'Transfer of Control with Multiple Intcrrupts

I/O Function

Thus far, wu have discussed the operation of the computer as controlled by the processor, and we havc hacked primarily at the interaction of processor and memory, The discussion has only alluded Its dill role of the I/O component. This ro[e.k discuss.cd in detail in Chapter 7, but a brief summary is in order hero.



Mime 1,14 **Example Time Sequenec. of Multipk. Interrupts ITANE90]**

An I/O module (e.g., i disk conl roller) can exchange data directly with the processor. Just as the processor can **initiate.a read or write with** memory, designating Ihe address of a specific location, the processor can also read data from or write data to an **1K) module. In Lhk 1+1t1t2r case,** lhe processor identifies a specific device that is controlled by a particular 110 **module, Thus, an instruction seque.ncc.** similar in form to that of Figure 3.5 could occur, with I/O instructions rather than memory-refe re (lc i **ti g inst** ructions.

In **sonic** cases, it is desirable to **allow IhO exchanges 10 occur directly** with memory. In such a case, the processor grants to an 110 module the authority to **rcmd from or** write to memory, so that the 1/0-memory transfer can occur without tying up the **processor.** During such *a* transfer, the module issues read or write commands to memory, relieving the processor or **rc',pomihility for the exchangc', operation is known as direct** memory access (DMA) and is examined Chapter 7.

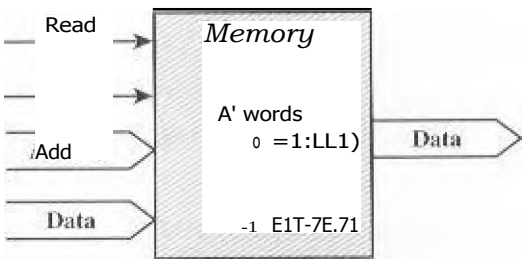
3d„ 4.f

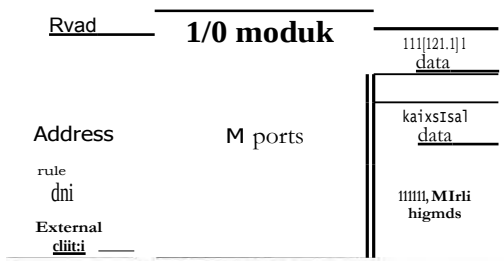
A computer Consists of a SUL Of COmponuilE; or modules of three basic types (processor, memory, I10) that communicate with each other. **In** effect, a computer is a nelwork of basic modules. Thus. there must be paths for connecting the modules.

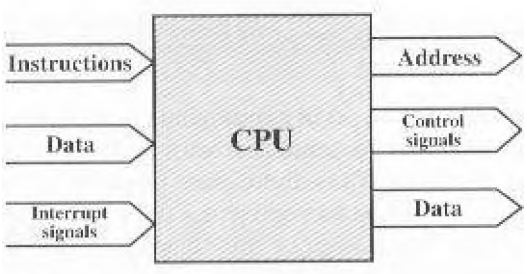
The collection or paths connecting the various modules is called the *inrercontrection srraiclurr*. **The design of this Alruclure will depend on the UNUhanges that** roust be **made between modules.**

Figure 115 suggc4;is the exchanges that are needed by indicating the major forms of input and output for each module type:

- Memor: 'Typically, a memory module will consist of N words of equal length. Lach word is assigned a unique numerical address (l1, I ..... $_{V}V - 1$),,A word of data can be read from or written into the memory. The nature of the operation is indicated by read and write wriirol signals. The location for the opermion is specified by an address.
- I/O module: From an internal f to the computer system) point of view, 1/0 is Itinoionallv similar to memory. ' l'hcre arc two operations, read and write. Further, an I/O module may control more than one external device. We can refer to each of I he interlaces to an external device as a *port* and give $_{eac}$h a unique address (e.g (I 1. l }, In *addition,* there are external data paths fur the input and outpul cif dada with an external dcvice. Finally, an 1/C) module malt. be able to send interrupt signals to the processor.







Figury 3.1.5 Computer hielodulcs

* Processor: The processor reads in instructions and data, writes out data alter processing, and uses control signals to control the overall operation of the sys tent It also receives interrupt signals.

The preceding list defines the data to he exchanged. The interconnection structure must support the following types of transfers:

- Memory to processor: The processor reads an instruction or a unit of data from memory.
- Processor to memory: The processor writes a unit of data to memory.
- I/O to processor: The processor reads data from an 110 device via an 110 module.
- Processor to **110:** The processor sends data to the 110 device.
- 1/0 to or from memory: For these two cases, an I/0 module is allowed to exchange data directly with memory. without going through the processor, using direct memory access (DMA).

Over the years., a number of interconnection structures have been tried. By far the most common is the bus and various multiple-bus structures. The remainder of this chapter is devoted to an assessment of bus structures.

3.4 BUS INTER. CONNECTION

A bus is a communication pathway connecting tvvo or more devices. A key characteristic of a bus is that it is a shared transmission medium. Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus. If two devices transmit during the same time period, their signals will overlap and become garbled. Thus. only one device at a ti me can successfully transmit.

Typically., a bus consists of multiple communication pathways. or lines. Each line is capable of transmitting signals representing binary 1 and binary U. Over time, a sequence of binary digits can be transmitted across a single line. Taken together, several lines of a bus can be used to transmit binary digits simultaneously (in parallel). For example, an 8-bit unit of data can be transmitted over eight bus lines.

Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy. A bus that connects major computer components (processor. memory, 110) is called a *system. bus.* The most common computer interconnection structures are based on the use of one or more system buses.

Bus Structure

A system bus consists, typically, of from about 50 to hundreds of separate lines_ Each line is assigned a particular meaning or function. Although there are many different bus designs, on any bus the lines can be classified into three functional groups

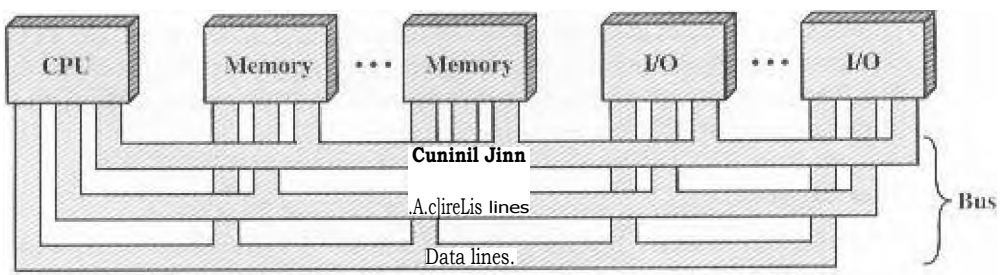
(Figure 3.16)7 data, address, and control **In** *addition,* there may be power distribution lines that supply power to the attached modules.

The *d'ena fines* provide 4 path for moving data between system modules. These lines, collectively , an called the *dear* bus, The data bus may consist of from 32 to hundreds of separate lines. the number of lines being referred to as the width of the data has. Because each line can carry only t bit at a time, ihe *number* of lines determines how many His can *he transf*erred at a time-The. width of the data bus is a key facl or in determining overall system performance. For example. ii the data bus is 8 bits wide and each instruction is 16 bits long, then the processor must access the memory module twice during each instruction cycle.

The *adiiresA lines* are used to designate I he source or destination of the data on the data bus. For example, if the processor wishes to read a word (S, **16,** or 32 bits) of data from memory, it puts the address of the desired word on the address lines. Clearly. the wichh of the address has determines ihe Maid rnum possible nicrnor!,' capacity of the system. Furthermore, the address lines are genera Ely also used to address 1.0 ports. Typically, the higher-order bits *are* used to select a particular module on the bus, and the Lower-order bits se]ecl a memory location or I/0 port within the module. For example, on an 8-hit address bus, address 011 HI 1 1 and below might reference locations in a memory module (module 0) with 128 words of memory, and address 10000000 and above refer to dev IL:es attached to an **mod-**ule (module

rile *control lines* are used to control the access to and the use of the data and address lines. Because the dal a and address lines are shared by a]1 components, there must be a means of controlling their use, Controi signals transmit berth command ;ind timing information between system modules. Timing signals indicate the of data and address information. Command sign Elk specify operations Lo he performed. Typical control lines include the following:

- **Memory write:** *Causes* data on the bus to be written into **the** addressed location_
- menrkor$_y$ **read:** Causes data from [he addressed location to be placed on the bus.
- **I/O write:** Causes data on t he huN to be output to the addressed VC) port,
- **I/O read:** Causes data from the addressed 110 port to be placed on *the* bus.
- **Transfer ACK:** indicates lhat data have been accepted from or placed on the bus.
- **Bus request:** Indicates that a module needs lo gain control of the bus.



1.i :Dirt. 3.16 Bus interconucction 5clrwrn±

- Bus grant: Indicates that a requesting module has been granted control of the bus.
- Interrupt request: Indicates that an interrupt is pending.
- Interrupt ACK: Acknowledges that the pending interrupt has been recognized.
- Clock: Used to synchronize operations.
- **Reset** Initializes a]] modules.

The operation of the bus is as follows. If one module wishes to send data to another, il must do two things: (I) Obtain the use of the bus, and **(2)** transfer dal a via the bus. If one module wishes to request data from another module. it must (i obtain the use of the bus. and (2) transfer a request to the other module over the appropriate control and address lines. It must then wait for that second module to send the data.

Physically, the system bus is actually a number of parallel electrical conductors. In the classic bus arrangement, these conductors are metal lines etched in a card or board (printed circuit board)..f'he bus extends across all of the system components, each of which taps into some or all of the bus lines. The classic •physical arrangement is depicted in Figure 3.17. In this example. the bus consists of two vertical columns of conductors. At regular intervals along the columns, there are attachment points in the form of slots that extend out horizontally to support a printed circuit board. Each of the major system components occupies one or more boards and plugs into the bus at these slots. The entire arrangement is housed in a chassis. This scheme can still he used **for** some of the buses associated with a computer system. However, modern systems tend to have all of the major components on the same board with inure elements on the same chip as the processor. Thus, an on-chip bus may connect the processor and cache memory. whereas an on-board bus may connect the processor to main memory and other components.

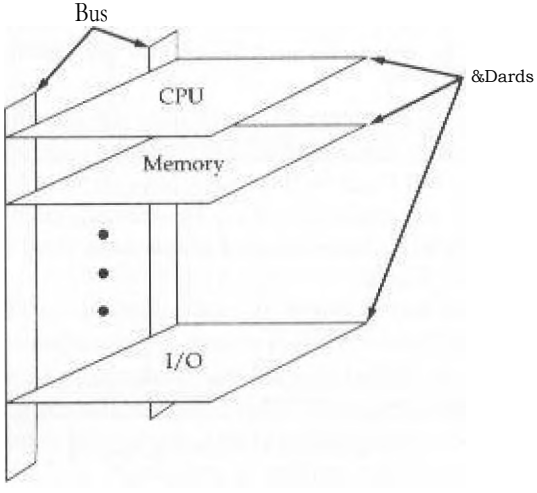


Figure **3.17** Typical Physical Realization of a Bus Architecture

This arrangement is **most** convenient. A small computer system may be acquired and then expanded later (more memory, more I/O) by adding more boards. if a component on a board fails, that board can easily be removed and replaced.

Multiple-Bus Hierarchies

if a •great number of devices arc connected to the bus, performance will suffer. There are two main causes:

I. In general, the.more devices attached to the bus, the greater the bus length and hence the greater the propagation delay. This delay determines the time it takes for devices to coordinate the use of the bus. When control of the bus passes from one device to another frequently, these propagation delays can noticeably affect performance.

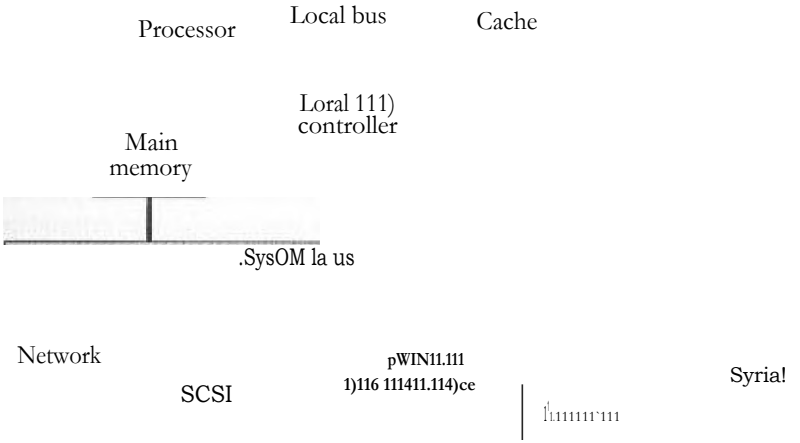
2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity tit the bus. This problem can be countered to some extent by increasing the data rate that the bus can carry and by using wider buses (e.g., increasing the data bus from 32 to 64 bits). l lowever, because the data rates generated by attached devices (e.g., graphics and video controllers, network interfaces) are growing rapidly, this is a race that a single bus is ultimately destined to lose..

Accordingly, most computer systems use multiple buses, generally laid out in a hierarchy. A typical traditional structure is shown in Figure 118a. There is a local bus that connects the processor to a cache memory and that may support one or more local devices. The cache memory controller connects the cache riot only to this local bus. but to a system bus to which arc attached all of the main memory modules. As will be discussed in Chapter **4,** the use of a cache structure insulates the processor from a requirement to access main memory frequently_ I knee, **main** memory can be moved off of the local bus onto a system bus, In this way, **trans-**fers to and from the main memory across the system bus do not interfere with the processor's activity.

It is possible to connect controllers directly onto the system bus. A more efficient solution is to make use of one or more expansion buses for this purpose, An expansion bus interface buffers data transfers between the system bus and the controllers on the expansion bus. This arrangement allows the system to support a wide variety of I/O devices and at the same time insulate. memory-to-processor traffic from 1.10 traffic.

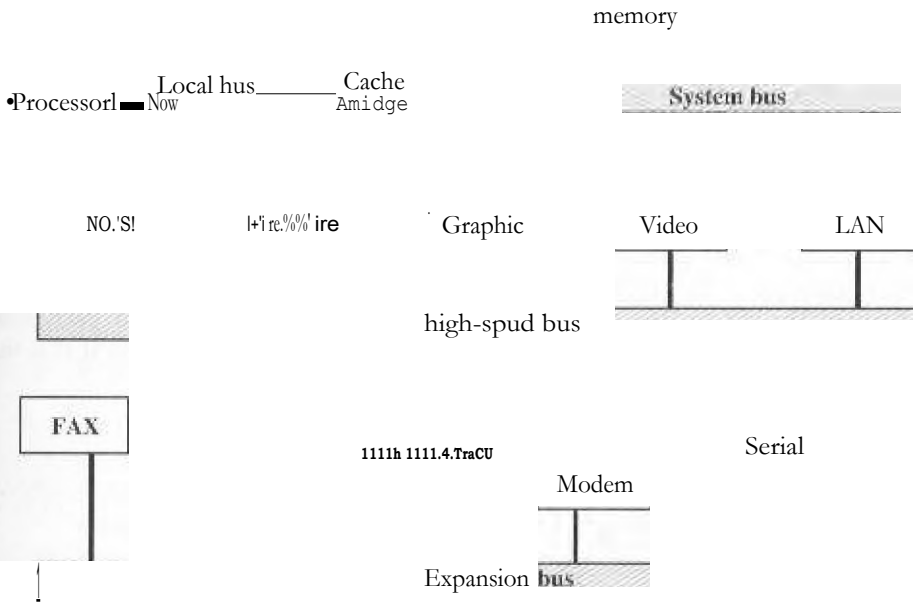
Figure 3.18a shows some typical examples of 110 devices that might be attached to the expansion bus. Network connections include local area networks (LANs) such as a 10-Mbps Ethernet and connections to wide area networks (WA Ns) such as a packet-switching network. SCSI (small computer system interface) is itself a type of bus used to support lucid disk drives and other peripherals_ A serial port could he used to support a printer or scanner_

This traditional bus architecture is reasonably efficient **but** begins to break down as higher and higher performance is seen in the 1/0 devices. In response to these growing demands. a common approach taken by industry is to build a high-



Exktin§km bus

la) Traditional bus architecture



thd. II U-perforinance architecture

Figure 3./8 Examplc Fie Configurations

speed bus that is closely integrated with the rest of the system, requiring only a bridge between the processor's bus and the high-speed bus. This arrangement is sometimes known as a mezzanine architecture.

Figure '_l lib shows a typical realization of this approach. Again, there is a local bus that connects the processor to a cache controller, which is in turn connected to a system bus that supports main memory. The cache controller is integrated into a bridge, or buffering device. that connects to the high-speed bus. This bus supports connections to high-speed LANs, such as Fast Ethernet at 11I(Mbps, video and graphics workstation controllers, as well as interface controllers to local peripheral buSes, including SCSI and FireWire. The latter is a high-speed bus arrangement specifically designed to support high-capacity I/O devices, Lower-speed devices are still supported off an expansion bus, with an interface buffering traffic bet wcen the expansion bus and the high-speed bus.

The advantage of this arrangement is that the high-speed bus brings high-demand devices into closer integration with the processor and at the same time is independent of the processor. Thus, differences in processor and high-speed bus speeds and signal line definitions are tolerated. Changes in processor architecture do not affect the high-speed bus, and vice versa,

Elements of Bus Design

Although a variety of different bus implementations exist, there are a few basic parameters or design elements that serve to classify and differentiate buses. Table 3.2 lists key elements.

Bus Types

Bus lines can be separated into two generic types: dedicated and multiplexed. A dedicated bus line is permanently assigned either to one function or to a physical subset of computer components.

An example of functional dedication is the use of separate dedicated address and data lines, which is common on many buses. However, it is not essential. For example, address and data information may be transmitted over the same set of lines using an Address Valid control line, At the beginning of a data transfer. the address is placed on the bus and the Address Valid line is activated. At this point, each module has a specified period of time to copy the address and determine if it is the

Table 3.2 Eicnivilts of Bus Design

Type	**Bum Width** **Addres3** Data
Method of Arhitrailoa	1) **ta Transfer 1'3.'pe** **Read**
l)istiihutucl 1 **buing,** Synchr4,511111.1.$ **Asynchronous**	**Read-noeliiy-uri** Lc **FilOCk**

addressed module. The address is Ihcn removed from the bus, and the same bus connections are used for 1,11c Submgiu rn read or write data transfer. This method of using the same lines for multiple purposes is known as *rime tradriplexing.*

The advantage of limo multiplexing is the use of fewer]ine s, which saves space. and, usually, cos. The disadvanl age is 1hat more complex cireuiiry is needed within each module. Also, there is potential reduction in performance because certain events that share the same lines cannot take place in parallel.

Physical dedicatim refers lo ihe use ot multiple buses, it4i.e.tt of which connects only a subset of modules. A typical example is the use of an IX) bus to interconnect all **110 modules** this bus is then connected to the main bus through some type of l/O adapter module, The potential advantage of physical dedication is high throughpul, because al L'n:. is less bus contention. A disadvantage is the increased the and cost of the system.

Method of Arbitration

In all but the simplest systems, more than one module may need control of the bus. For example, an 110 module may need 10 read or writc direeFly 10 memory, without sendimz the data to the processor. Because only one unit at a time can successfully transmit over the bus, *some* method of arbitration is needed. The various methods can be roughly classified as being either ccal tra izizd or distributed. In 1ra Hod scheme, a single I hrdwarc device, referred 10 ax a *controller or athifer,* is responsible for allocating time on the bus, The. device may be a separate module or part of the. processor. In a distributed scheme, there is no central controller- Rather, each module contains access control logic and the modulczi act together to share the bus. With both methods of arbitration, the purpose is to desianate one device. either the processor or an I/O module. as master. The master may then li me a data Iranster read or write). with some other device, which fie[s AS siave for this particular exchange.

Timing

Timing refers Lo the way in which events arc coordinated **oh** the bus. Buses use either synchronous timing or 4ts:..nchronouz, tin ing-

With **synchronous timing,** the occurrence of events on the bus is determined by a clock. The bus includes a clock line upon which a clock transmits a regular sequence Of alternating Is and tlx of equal duration. A single :L-0 transmission is referred to as a dock cycle or *bus. cycle* and defines a lime skit. **All** other devices on the bus can read the clock line. and all cYcrils sl ari a,L the be ;inning of a clock eyeie. Figure 3.19 shows a typical, hill simplified, tinning diagram for synchronous read and write operations (see Appendix 3A for a description of timing diagrams). Other bus signals may change at the leading edge of the clock signal (With a slight reaction delay), Most events occupy a single clock cycle,. In this simple example, the processor plaices a memory address on the address lines during the first clock cycle, and may assert various status]ines. Once the address Lines have siabilized, the processor issues an address unable signal. For a read operalion, the processor issues.a read command at the start of the second cycle. A memory module reco2-nizes the address and, after **a delay** of one cycle, places the data on the data

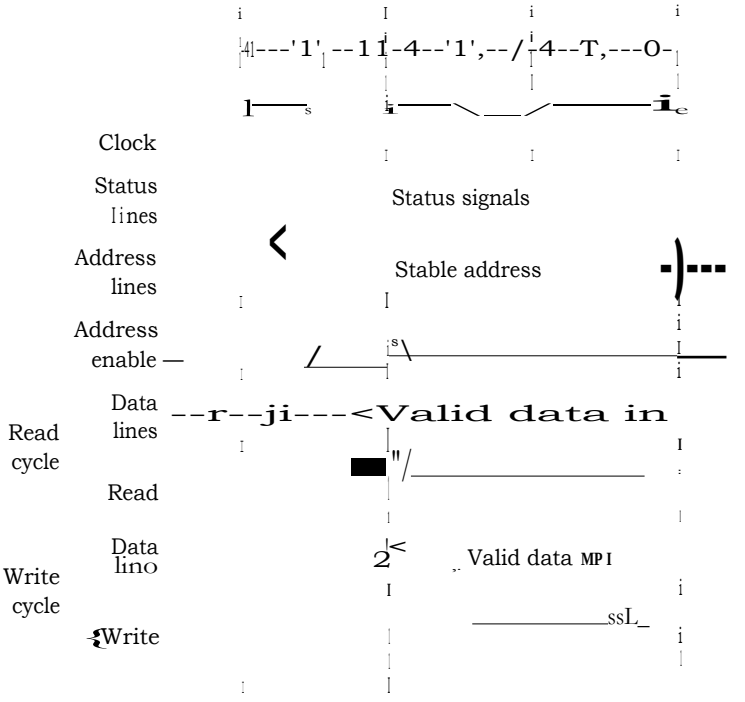
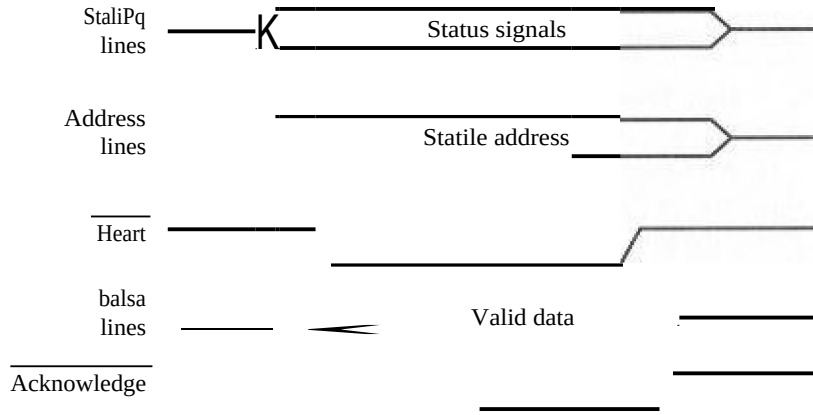


Figure 3.19 Timing of Synchronous Bus Operations

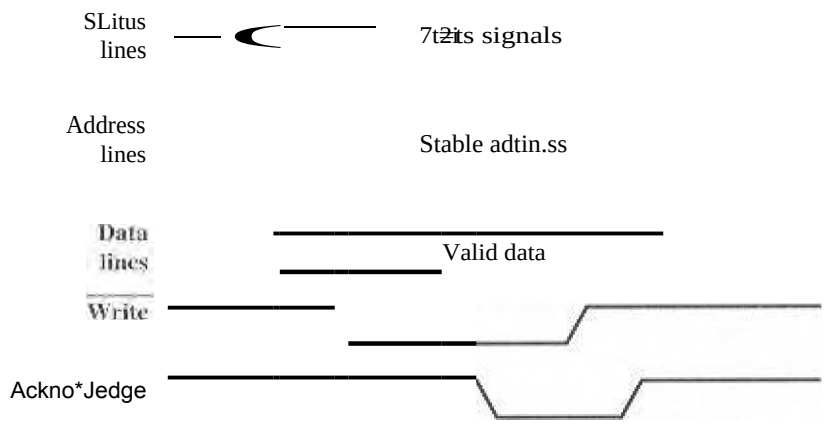
For a write operation, the processor puts the data on the data lines at the start of the second cycle, and issues a write command after the data lines have stabilized. The memory module copies the information from the data lines during the third clock cycle.

With asynchronous timing, the occurrence of one event on a bus follows and depends on the occurrence of a previous event. In the simple read example of Figure 3.20a, the processor places address and status signals on the bus. After pausing for these signals to stabilize, it issues a read command. indicating the presence of valid address and control signals. The appropriate memory decodes the address and responds by placing the data on the *data* line. Once the data lines have stabilized, the memory module asserts the acknowledged line to signal the processor that the data are available. Once the master has read the data from the data lines, it &asserts the read sienal. This causes the memory module to drop the data and acknowledge lines. Finally, once the acknowledge line is dropped, the master removes the address information.

Figure 3.20h shows a simple asynchronous write operation. In this case, the master places the data on the data line at the same lime that is puts signals on the status and address lines. The memory module responds to the write command by copying the data from the data lines and then asserting the acknowledge line. The master then drops the write signal and the memory module drops the acknowledge signal.



(a) System bus rend cynic



ill) System Luis **write** CYCle

3,2411 Timing c Aqynchronnus Bus Operations

Synchronous timing is z.;iinpler to implement and test. However, it is loss flexible than asynchronous timing, Because ME devices on a synchronous hus..urr2 tied to a fixed clock rate. the system cannot take advantage cyf advanen in device. perforIlliffiQc. **With** asynchronous liming, 41 mixture of alien and fast devices, using ()Eder and newer technology, can share a bus-

B us **Width**

We have already addressed the concept of his width. The widl.h of the data bus has an impact on system performance; The wider the data bus. the greater the number of hits transferred ;0 a ntic ti me. The width of the address bus has an impact

on system capacity: The wider the address bus, the greater the range of locations that can he referenced.

Data Transfer Type

Finally, a bus supports various data transfer types, as illustrated in Figure 3.21. Al] buses support both write (master to slave) and read (slave to master) transfers. In the case of a **multiplexed address/data bus. the bus is first** used for specifying the address and then for transferring t he data. For a read operation, there is typically a wait while the **data is being fetched from the** slave to be put on the bus. For either a read or a write, there **may also be a delay 11 11 is necessary** to go through arbitration to gain control of the bus for the remainder of the operation (i.e., seize the bus to request a read or write, then seize the bus again to perform a read or write).

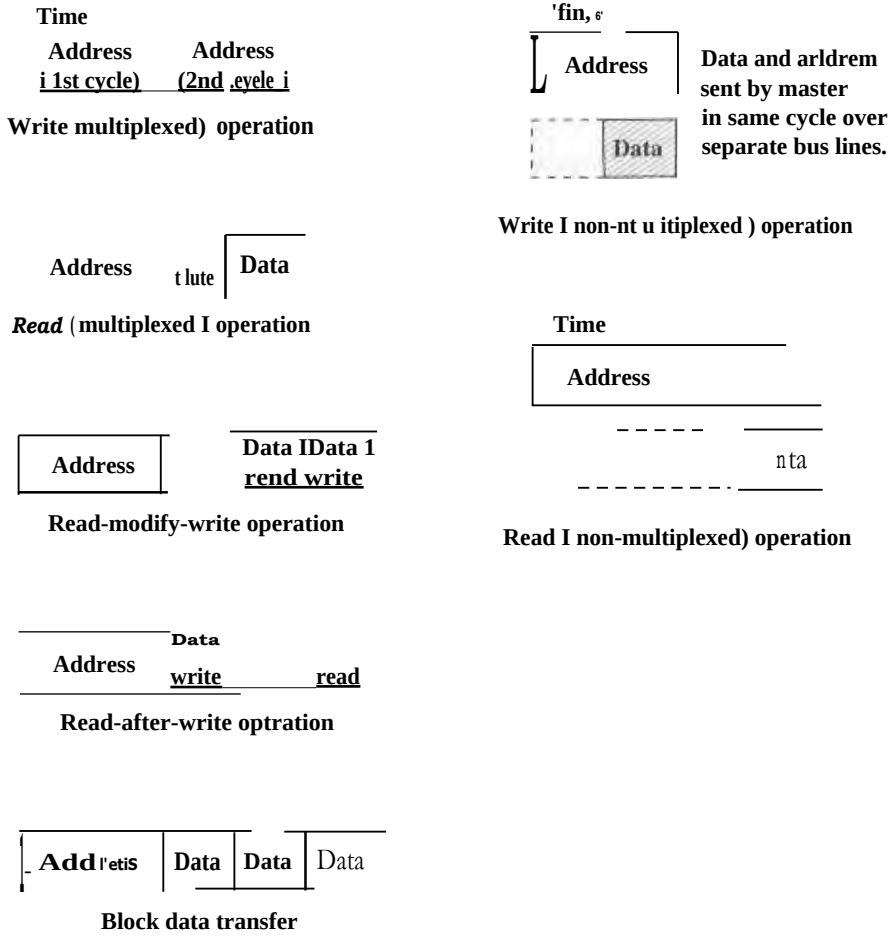


Figure 3.21 Bas Data Transfer Types ICIOOR89I

In the case of dedicated address and data buses. the address is put on the address bus and remains there while the data are put on the data bl/S. For a write. operation, the master puts the data onto the data bus as soon as the address has stabilized and the slave has had the opportunity to recognize its address, For a read operation, the slave puts the data onto the data bus as soon as it has recognized its address and has fetched the data.

There are also several combination operations that some buses allow. A read-modify-write operation is simply a read followed immediately by a write to the same address. The address is only broadcast once at the beginning of the operation. The whole operation is typically indivisible to prevent any access in the data element by other potential bus masters. The principal purpose of this capability is to protect shared memory resources in a multiprogramming system (see Chapter 8).

Read-after-write is an indivisible operation consisting of a write followed immediately by a read from the same address_ The road operation may be performed for checking purposes.

Some bus systems also support a block data transfer. In this ease. one address cycle is followed by n data cycles. The first data item is transferred to or from the specified address; the remaining data items are transferred to or from subsequent addresses.

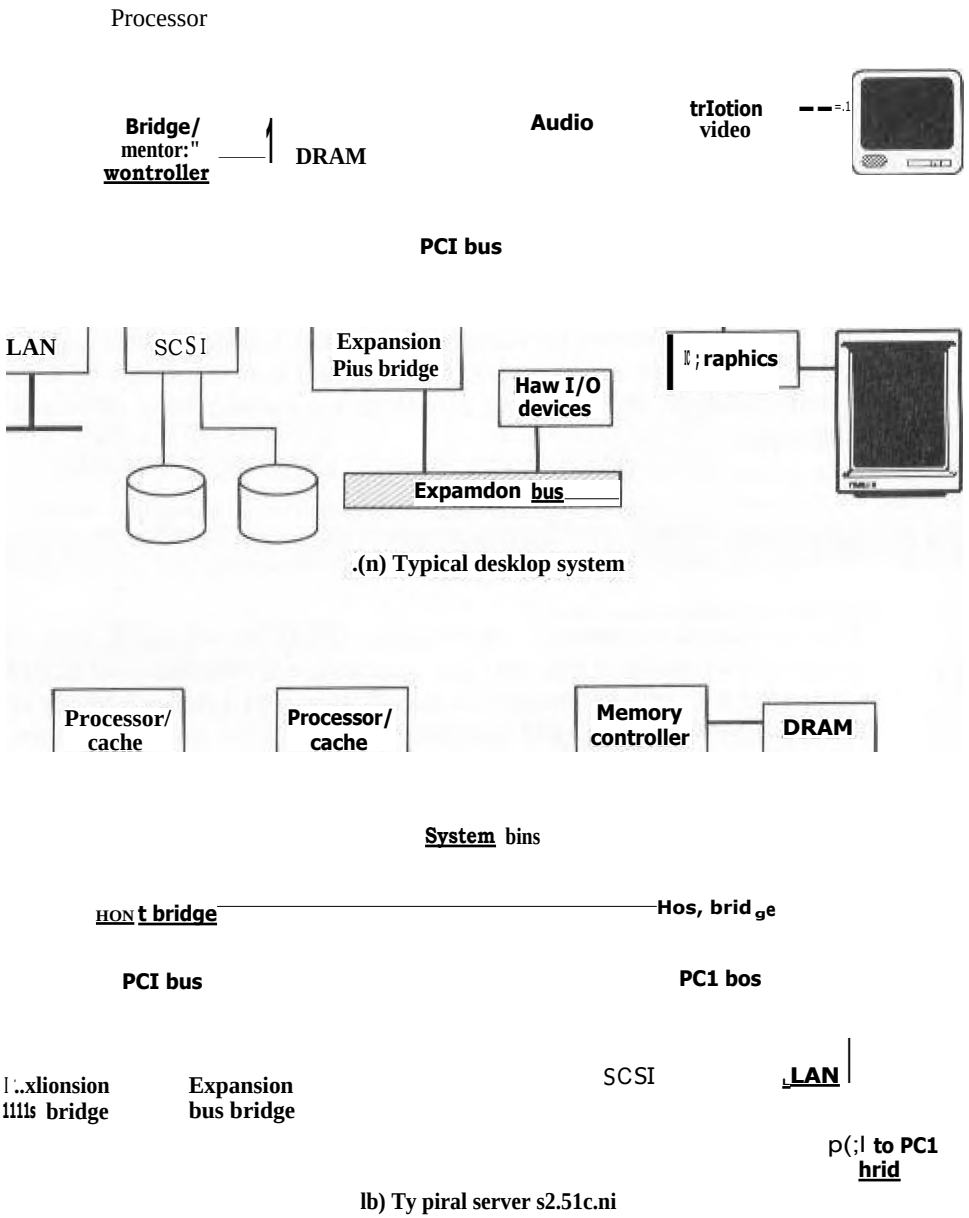
3.5 PCI

The peripheral component interconnect (PCI) is a popular high-bandwidth, processor-independent bus that can function as a mezzanine or peripheral bus. Compared with other common bus specifications, PCI delivers better system performance for high-speed 1.0 subsystems (e.g., graphic display adapters, network interface controllers, disk controllers, and so on), Thc current standard allows the use of up to 64 data lines al fifi MHz, for a raw transfer rate of 526 .MBytels, or 4.224 Gbps. But it is not just a high speed that makes PCI attractive_ PC1 is specifically designed to meet economically the 110 requirements of modern systems; it requires very few chips to implement and supports other buses attached to the PCI bus.

Intel began work on PC'i in 1990 for its Pentium-based systems. Intel soon released all the patents to the public domain and promoted the creation of an industry association, the PCI SW, to develop further and maintain the compatibility of the PC1 specifications_ The result is that PO has been widely adopted and is finding increasing use in personal computer, workstation, and server systems, As of this writing, the current version is PCI 2.2. Because the specification is in the public domain and is supported by a broad cross section of the microprocessor and peripheral industry. PCI products built by different vendors are compatible.

PCI is designed to support a variety of microprocessor-based configurations. including both single- and multiple-processor systems. Accordingly, it provides a general-purpose set of functions. It makes use of synchronous timing and a centralized arbitration scheme.

Figure 3.22a shows a typical use of PCI in a single-processor system. A combined DRAM controller and bridge to the PCI bus provides tight coupling with the



.(n) Typical desklop system

lb) Ty piral server s2.51c.ni

Figure 122 Example PCI Conagurations

processor and the ability to deliver data al high speedzs. The bridge aets as a data buffer so that the speed of the PC.I bus may differ from that of the processor's capability. In a multiprocessor system (Figure 3.22b), one or more PCI configurations rruiy be connected by bridges to the processor's system bus. The system bus supports only the processoricaehe units, main memory- and the PO bridges. Again. the use of bridges keeps the PCI independent of the processor speed yet provide.1 ', the a313iIi1y to receive and deliver data rapidly.

Bus Structure

PC:l may be configured its it 32- or 64-bin bus. Table 3.3 defines the 49 mandatory signul lines for PC'1. These are divided into the following functional groups:

- **System pins: Include** the clock **and** reset pins.
- **Address and** data **pins:** include 3.2 hoc'.! hat are 1 irne multiplexed fc.lr addresses and data. The other lines in this :group tux used to interpret and validate the **signal** Lines that carry the addresses and data.
- **Interface control** pins: Control the liming of *tr;irkwctionF.; anal* provide coorcli6: nation among initiators and targets.
- Arbitration pinN: Unlike **the** other PCI signal lines, these arc riot shared lines. Rather. each PCI master has its own pair of arhiira [ion lines I hat connect it directl!,' to the PCI bus arbiter.
- **Error reporting pins:** Used to report parity and other errors.

In addition. the PCI specification defines 51 optional sigitpl lines (fable 3,4), di iced in **10** the following functional groups:

- **Interrupt pins:** These are provided for PCI devices that must generate requests for service. AS with the arbitration pins, these are not shared lines. Rather, e;ic]1 PC[device has its own interrupt line or lines to an interrupt controller.
- Cache **support pins:** These pins are needed to support a memory on **PCI** that can lie ckiched in the processor or anol her &vice- These pins support snoopy cache protocols (see Chapter 18 for a discussion of such protocols).
- **64-bit bus extension pins: include** 32 lines that arc time multiplexed for addresses ;ind dah'i and than are combined with the mandatory address data lines to form a 64-hit address/data bus. 01 her lines iIa this group arc used to interpret and validate the signal lines that carry the addresses and data.
  there are two lines that enable two PCT devices to agree **to** the. use of the 64-bit capability.
- **JTAGibonndary scan pins:** These signal lines support testing proced u res defined in IEEE Standard 114Q].

PCI Commands

Bus activity occurs in the form of transactions between an initiator, or master. and a target. When a bus master' acquires control of the bus. it determines the type of

Table 3.3 Mandatory PC' Signal Lines

Designation	Type	Description
		System Pins
CIA	in	Provides timing for all transactions and is sampled by all inputs on the rising edge. Clock rates up to 33 MHz are supported_
RST#	in	Forces all E'Cl.specific registers. sequencers, and signals to an initialized state.
		Address and Data Pins
AD I: t :Al	tis	Multiplexed lines used for address and data.
OBE[3::010	[Is	Multiplexed bus command and byte enable signals. During the data phase. the lines indicate which of the four byte limes carry meaningful da ta_
PAR	Us	Provides even parity across AD and OSE lines one clock cycle later. The master drives PAR for address and write data phases: the target drive PAR for read data phases.
		interface Control Pins
FRAME*	sAis	Driven by current master to indicate the start and duration of a transaction. It is asserted at the start and deasserted when the initiator is ready to begin the final data phase.
1RDY,	sitis•	Initiator Ready. Driven by current bus master ;initiator of transaction), During a read. indicates that the master is prepared to accept data: during a write, indicateS that valid data are present on AD.
TRDY-FF	sits	Target Ready. Driven by the target (selected device). During a read, indicates I hat valid data are present on AD; during a write, indicates that target is ready to accept data_
STOPS	kik.'s	Indicates that current target wishes the initiator to stcup.the current transaction.
I DSEL	in	Inicialierition Device Select. Used as a chip select during configuration read and write transactions.
DEVSF.I ,t⁴	in	Device Select. Asserted by target when it has recognized its address. Indicates to current initiator whether any device has been selected.
		Arbitration Pins
REQo	L's	Indicates to the arbiter that this device requires use of the bus, This is a device-specific point•tO-prune line.
ONTA	tis	Indicates to the device that the arbiter has granted bus access. This is. ii device-. specific point-to-point line.
		Error Reporting Pius
PERR#	sills	Parity Error. Indicates a data parity error is detected by a target during a write data phase or by an initiator during a read data phase.
SERRff	old	System Error. May he pulsed by any device to report address parity errors and critical errors other than partly.

rabic 14 Optional Fti Signal Lints

Designation	1⁹t, pe	Description
Interrupt Pins		
TNTA#	(1..:a	used to n2CIL.LESI an interrupt.
I NTI:i	old	Used In TcgLicRI, ;in in V2Tr upl.: c nil y Iii 1114:216.11g un a inuttifuniction device.
I. TC',a	ad	Used to request an inierrupt: only has nicnn mg on ;..1 I'fl annel:Lan device.
INTD.:	419'11	Used to requerd an interrupt! only has meaning on inulltfunclion iin..ice..
Cache Support MIS		
SBON	infout	Snoop Back ell Indicalcs a hil to .3 rricidifiLd. line.
SDQNE	irIkrUL	Snoop Done. **Indicates** (lie 31.aras ot 11.1C Nnoop For she current acc&nt. Asseriled wile.'" snoop has been completed.
64-bit HIPS Extension Pins		
A11'[63;;34	L/.5	Multiplexed linuh used for kithirc v rind data I.0 enend hus LC. M bits.
Ci]3E[7ii4l	L.S	Multiplexed bus command and byte. c Ra IA 12 Niglvl I s. During the: rates phase. the lines provide additional bus commandri. During LFIE Haiti phase, the lines indicate wluch or the lour e.xtcu.de d hyl la ri.;:s cury 11.1CLInirpjul **data.**
REQ64	sills	Used to request 64-bit transfer.
ACk64.100	sith,	Midi cul6s l a rger is w:illinj.:: to perform **64-bit** transfer.
.PAR64	r/S	Provides .: ':.....:n 1'..irri:... ACTI MS i2xientlEd AD and OBE lines' one clock cycle liter,
ITAC/Boundury Simi Pins		
ICI(	in	Test Clod.... 1..:s42d Lci. dock ....f ..11.t. inionnation and i t data into aric.I out of Ulu du vicc durin F hcrundmy WWI.
MI	hi	1'.....st **Input.** Used to .scriiilly shill le.q. ILL a aitd insbuctions into the device.
TDO	out	Test Out put. L:sed to serially shift ic.sl. data and instructiuris out of the device.
MIS.	in	Test l'vlode Select. Used t CCI nisei! state of te51 a CCC35 pint control! LT:
TRSTfiF	in	Test Resut. I.IECti to ill iLiali2e: test access port c\5ittrall u•

in Inpuit-only signal

tiut Output-only signal

signal

SuLained id-state signal driven by only on. owne r

rr t1 Open dniiii: multiple destiCei to share a 3 a wire-OR

#l 0 i5.1! sI M occurs ak. Inw vollavg

transaction that will occur next. During the address phase of the transaction. the OBE lines are used to signal the transaction type. The commands arc

- Interrupt Acknowledge
- Special Cycle
- I/0 Read
- 110 Write
- Memory Read
- Memory Read Line
- Memory Read Multiple
- Memory Write
- Memory Write and Invalidate
- Configuration Read
- Configuration Nkrrite
- Dual Address Cycle

Interrupt Acknowledge is a read command intended for the device that functions as an interrupt controller on the PCI bus. The address lines are not used during the address phase- and the byte enable lines indicate the size of the interrupt identifier to be returned,

The Special Cycle command is used by the initiator to broadcast a message to one or more targets.

The Read and Write commands are used to transfer data between the initiator and an 110 controller. Each I/0 device has its own address space, and the address lines are used to indicate a particular device and to specify the data to be transferred to or from that device. The concept of I/O addresses is explored in Chapter 7.

The memory read and write commands are used to specify the transfer of a burst of data. occupying one or more clock cycles. The interpretation of these commands, depends on whether or not the memory controller on the PCI bus supports the PCI protocol for transfers between memory and cache. If so. the transfer of data to and from the memory is typically in terms of cache lines, or blocks,' The three memory read commands have the uses outlined in Table 3.5, The Memory Write command is used to transfer data in one or more data cycles to memory_ The Memory Write and Invalidate command transfers data in one or more cycles to memory. In addition, it guarantees that at feast one cache line is written. This command supports the cache function of writing back a line to memory.

The two configuration commands enable a master to read and update configuration parameters in a device connected to the PCI, Each PC1 device may include up to 25 ‧ internal registers that are used during system initialization to configure that device.

"The funda menial principles of cache memory are described in Chapter 4; bus based cache prolocols are described in Chapter 1K

Table 3.5 Interpretation of PCI Read Commands

held Command	For Cgichnhie manure	For Nonesichahle Memory
MC111CITV RCM'	unc-half or 1·2l-.ti fi C2Cht	Busting 2 <lath transfuc....clos IV. '.('S`;
Me111CITV Redd. Line	MCF1'12 tIrdu or:L.-11;1ft a tf,chr line 141 C411:E).2. Ines	Bursting 3 to 12 data srN nsirrw
Me11.1CITy Rcad. Multiple	Bursling rricivu Lachc hues	Burstiag more than 1,7 dm,' trmisfc.rs

The Dual Address Cycle command is used by an initiator to indicate that it is using 64-bit addressing.

Data Transfers

Every data transfer on the ['CI bus is n Irmisaell ion consisting of one address phase and one or inore ryhases. in this discussion, we illustrate iI typical read operation; a write operation proceeds similarly.

Figure 3.23 shows the timing of the read transacLlon.. All events are synchronized to [he falling transitions 01 the: clock, which occur in the middle of each clock cycle- Bus devici2S sample the bus lines on the rising edge at the beginning of a bus cycle. The followina are the significant events, labeled on !he diagram:

a. Once a bus master has gained control of the bus, it may begin the [tonsaction by asserting FRAME. This line remains Laili I the initiator is ready to complete the, last dota phase. '1 '111,2 initiator also puts the start address on the address bus, and the read command on the CIBE lines.
b. At the start of clock 2, the target device will recognize its .iddrcss on the AD lines.
c. The initiator ceases driving the AD bus. A turnaround cycle. Ondiented by the.two circular arrows) is required on all signal lines That 'nay be driven by more lhan one device, so that the dropping of the address signal will prepare the bus for use by the target device. The initiator chongcs the information on the CiBE lines to desigru which Al) lines are to be used for transfer for the currently addreAsed Clfi La to 4 bytes), iniLia tor also 4i;iscil:s 11Z17Y Lo indicate that it is ready for the first data hum.
d. selected target asserts DEVSEL to indicate that it has recognized its address and will respond. IL phIces the re vested data on the AD lines and asserts T'RDY to indicate that valid data is present on the bus.
e. The initiator reads the dab+ al the beginning of clock 4 and changes the byte enable lines m; ni;cdcd in preparation for the next read.

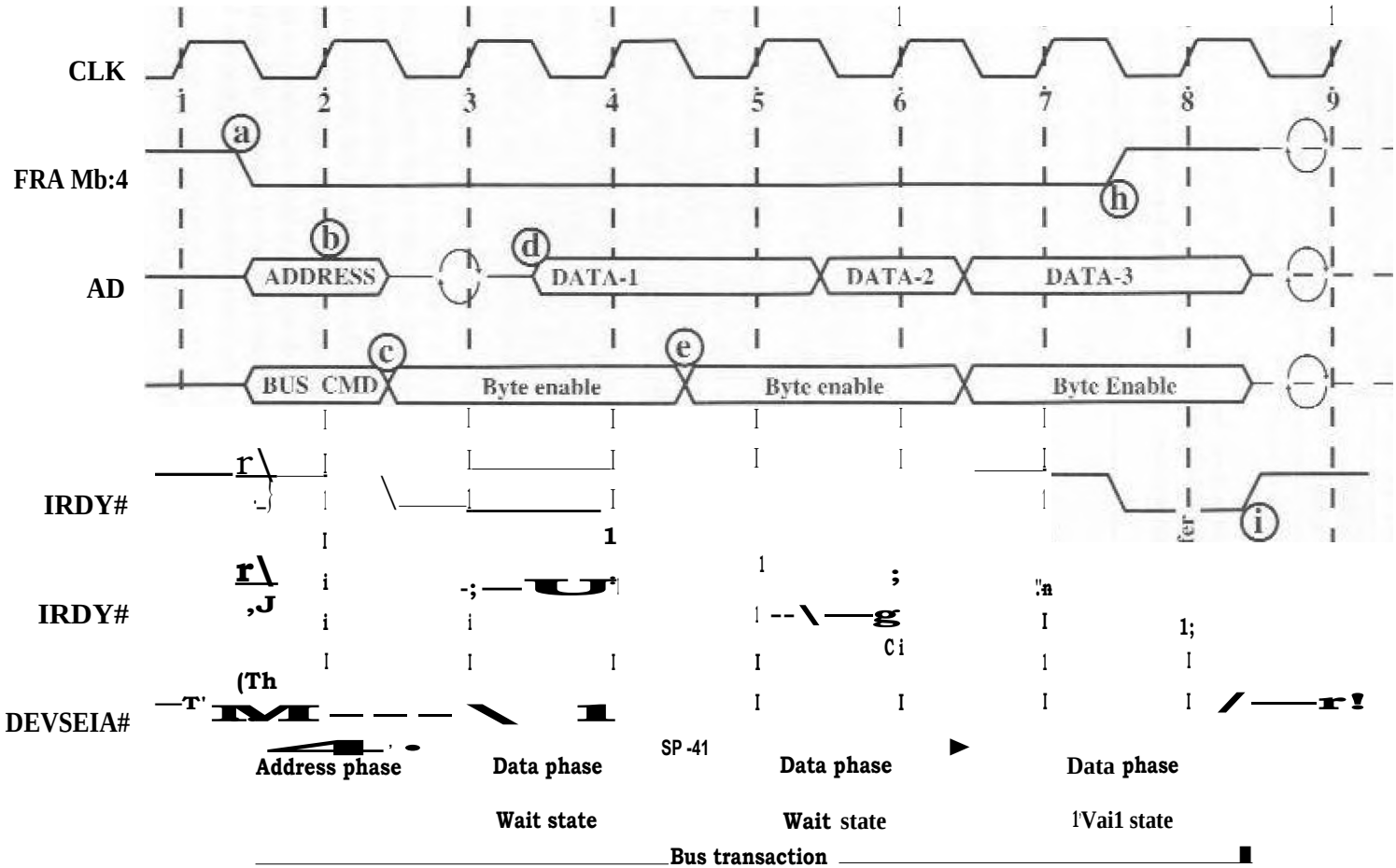


Figure 3.23 PC1 Read Operation

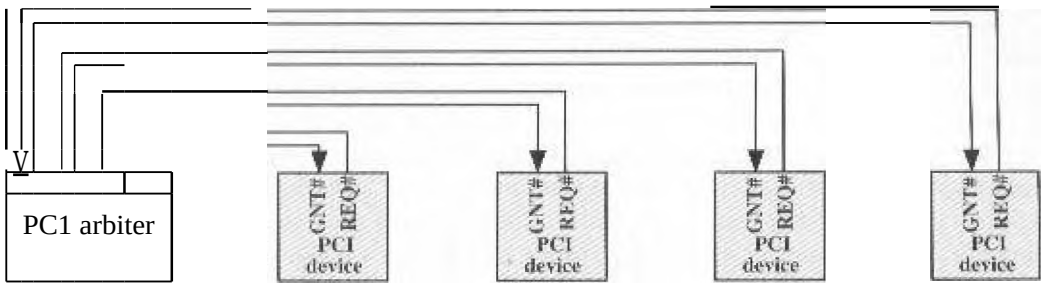


Figure 3.24 FC1 **Bus** Arbiter

f. **In** this example, the target needs some time to prepare the second block of data for transmission_ Therefore. it deasserts TRI)Y to signal the initiator that there will not be new data during the coming cycle. Accordingly, the initiator does not read the data lines at the beginning of the fifth clock cycle and does not change byte enable during that cycle. The block of data is read at beginning of clock 6.

g. During clock 0, the target places the third data item on the bus- However, in this example. the initiator is not yet ready to read the data item (e.g., it has a temporary buffer full condition). It therefore deasserts IRDY. This will cause the target to maintain the third data item on the bus for an extra clock cycle,

h. The initiator knows that the third data transfer is the last, and so it deasserts ll-Z.AME: to signal the target That I his is the last data transfer. It also asserts IRDY to signal that it is ready to complete that transfer.

i. The initiator deasserts fRDY, returning the bus to the idle state, and the target deasserts 'I'RDY and DEYSEL.

Arbitration

PC 1 makes use of a centralized, synchronous arbitration scheme in which each **master** has a unique request (REQ) and grant ((NT) signal. These signal lines **are** attached to a central arbiter (Figure 3,24) and a simple request-grant handshake is used to grant access to the bus.

The I'C'I specification does not dictate to particular arbitration algorithm. The arbiter can use a first-come-first-served approach, a round-robin approach, or some sort of priority scheme. A **PCI** master must arbitrate for each transaction that it wishes to perform, where a single transaction consists of an address phase followed by one Or more contiguous data phases.

Figure.3.25 is art example in which devices A and B are arbitrating for the bus. The following sequence occurs:

a. At some point prior to the start of clock 1, A has asserted its REQ The arbiter samples this signal at the beginning of clock cycle 1.

h. During **clock cycle 1. B** requests use of the bus by asserting its RI X) signal.

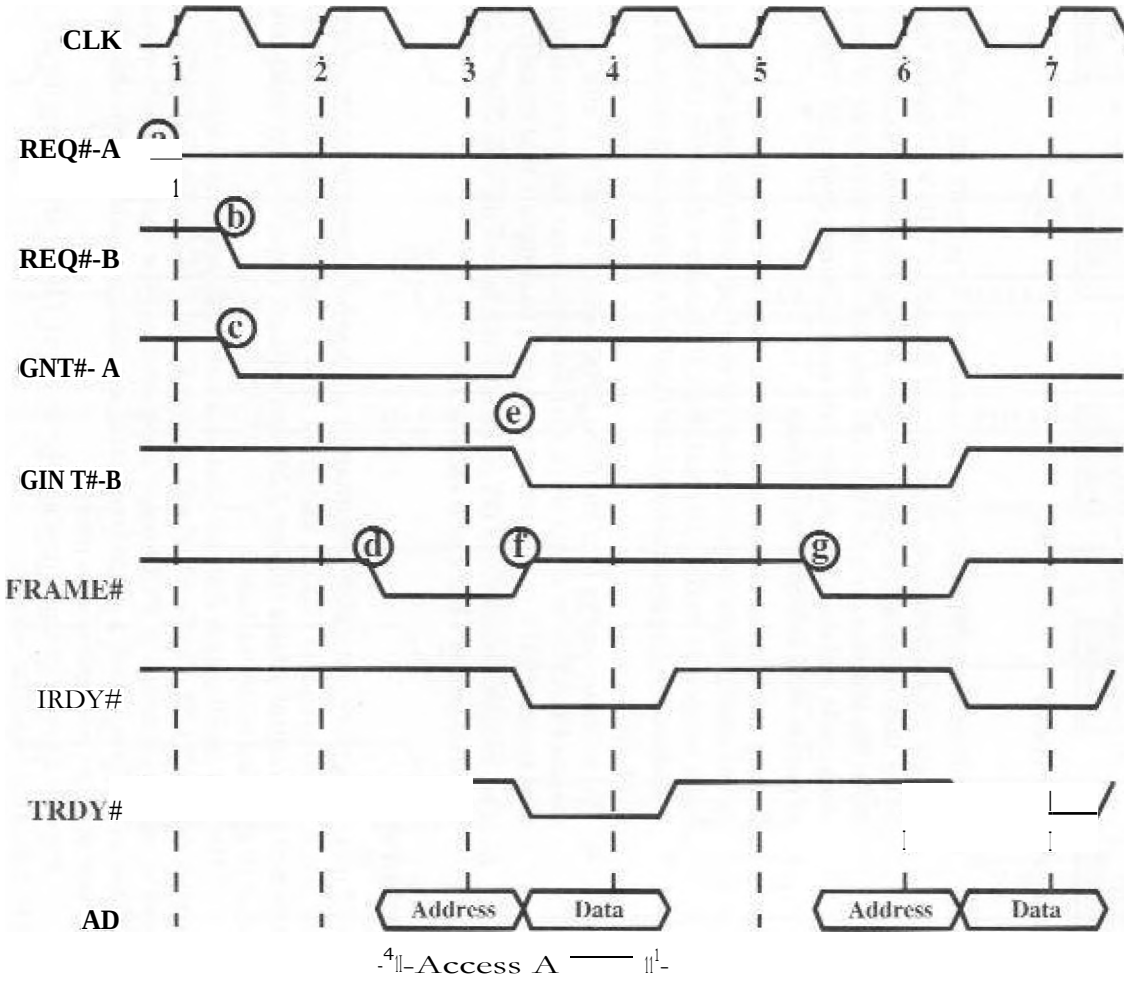


Figure 3.23 Ft:113m Arbitration between Two Masters

c. At the same time, the arbiter asserts **oNT-A** to grant bus access to A.

d. Bus master A samples CiNT-A at the beginning of clock 2 and learns that it has been granted bus access. It also finds IRI)Y and TROY deasserted, indicating that the bus is idle. Accordingly, it asserts FRAME and places the address information on the address bus and the command on the CBE bus (not shown). It also continues to assert RF.Q-A, because it has a second transaction to perform after this one.

e. The bus arbiter samples all REQ lines at the beginning of clock 3 and makes an arbitration decision to grant the bus to B for the next transaction. It then asserts GNT-B and deasserts CNT-A. B will not be able to use the bus until it returns to an idle state,

1. A deasserts FRAME to indicate that **the** last (and only) data transfer is in progress. It puts the data on the data bus and signals the target with **IRI)Y_** The target reads the data al the beginning of the next clock cycle.

g. At the beginning of clock 5, B finds IRI)Y and FRAME deassertect and so is able to take control of the. bus by asserting FRAME.. It also deasserts its REQ line, because it **army** wants to perform one transaction.

Subsequently, master A is granted access to the bus for its next transaction_

Notice that arbitration can take place at the same time that the current bus master is performing a data transfer. Therefore, no bus cycles are lost in performine arbitration. This is referred to as *hidden arbitration,*

3.6 RECOMMENDED READING AND WEB SITES

rhiL litL.rature on **buses and other interconnection structures is, surprisingly, not very extensive. ALE 93J includes an in-depth treatment of bus structures and bus transfer issues, including accounts of se v era specific** buses.

The clearest buck .lcrw ri **description of PCI is NIIAN951. IARBOGOI also contains a lot of solid information** on PCl.

ABB000 Abbot, D. PC! *Bros Demys.afied,* **Eagle Rock, VA: LI .Ft** Technology Pohlishing, 2000,

ALF.X93 Alexandridis, N. *Desiv? Micropwccs.vor-Based Systems.* **Englewood Cliffs, NJ: Prentice Hall, 1993,**

SIIAN95 Manley, and Anderson. D. *PC.1 Sysiemy Ayritifeerttre.* Richarclsou, TX: Mindshare Press, 1995.

Recornmoaded Web Sites:

- **PC Special Interest Group:** Informal ECM about I'C'I specifications and products.
- **PCI Pointers: I .inks to PCI vendors and other sources of** information.

3,7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS •

Key Terms

address bus
asynchronous Lintilig bus
bus 4ir bitratiim
bus width
CQ1111:111iZOCI arbitration
data bus
disablei.1 interrupt
distributed arhittation
instructiiin cycle.
4l 1ti1rLuction CkCCUre
ills !suction te' tcic
interrupi
inif rrupl. handler
intrrupt service routine
tiicri-lory address rt'E,rislxz (MAR)
memory butter rc.gister (AM).
r.riptseral. COInp47' 110E1 into rwikaect (PC1)
svmlironous
gystern bus

Review Questions

3.1 What general categories of function% are specified by computer instructions?

3,2 List and briefly define the possible stales I hat define an instruction execution.

3.3 List arid briefly define two .i ipproadies to dealing with multiple

3,4 What lypes of transfers must a computer's interauniection structure (e.g.. bus) sollilore

3,5 What is the benefit of using a multiple-bus architecture compared to a single-bus architecture"

3.6 List and brictly define ale functional groups 411: signal lines for P.C.1.

Problems

3.1 The hypothetical ma tine of Figure 3.4 also has two 1110 instructions;

0011 — Load AC from I.10
01 I I — Store AC to I/O

In these eases. the 12•bil address identifies a particular 110 device. Show tlic. program cxeciii lien (itsing the format of Figure 3.5) for the followinn program!

L Load AC from deviec 5.

1. Acid contents of IT1611143ty 14.1CatiCM 1440.

3. Store AC to device 6.

Assume that the next value retrieved from dkivice 5 is 3 and that location 940 contains a value of 2.

3.2 The proir,rain execution or Figure 3.5 is described in the text using six steps. Expand this description Lo show the use of the MAR and MBR.

3-3 Consider a hypothetical 32-bit microprocessor having instructions composed of two fields: 1'he first hvie contains the sirode and the remainder the humodiate operand or an operand address.

a, What is the maximum. directly addressable inerm iry capacity' in bytes)?

h. Disct-s file impact on the system speed if the microprocessor bus lac

1, a 32-bit local address bus and a 16i-bit local data bus, or

2, a 1.6-bit local address bus and a hit loon data buis.

c, How many hits are needed for the progr;ini countor and the instruction ree,ister?

,Sr drrce; [AIJ iX93]

3.4 Consider a hypothetical microprocessor generating a 16-bit adcirvis (foi' example, assume that the program counter and the address registers are l6 bits wide) and having a 16. bit data bus.

a. What is the maximum memory addross•space that the processor earl fi=ss directly if it is connected to a "16-bit memory"?

k What is the maximura memory address space that the processor can access directly if it is connt.cted to an "8-bit memory"?'

c. What architectural teal tires will allow this microprocessor to access a separate " apace."?

d. If an input and an output instruction can specify an ti-hit 1.0 port number, how many 8-bit PO ports can Ihe microprocessor support? 110Y.' inany 16-bit I/O Boris? Explain.

Source;

3.5 Consider a 32-bit microprocessor, with a 16-hit external data bus. tiriveil by an 8-.N.T1Le input clock. Assume that this microprocessor has a has cycle whose minimum duration equals four input clock cycles, What is the maximum clam 11;111nh2r **Tats** that this microprocessor can sustain? To increase its performance. would it lie beitr to make its eat:A˙nal data bus 32 hits or to double Ilitt eN1ernal clock ireqwency supplied to the microprocessor? State any other assumptions you make, and explain.

Source: 1AL EX931

3.6 Consider .k1 coniputer system that contains an PO module controlling a simple keyboardiprinler teletype. The following coritainod in the processor and wrinomd directly to the Aysleru bus:

11‘;Pk! Inpul Re.2,ister, f hiss;

OUTR: Output Register, 8 hits

Hic; Inpul Flag, 1 bit

MO: Output Flag, 1 bit

TEN: Interrupt Enable. I bit

Keystroke input front the (eietype and printer output to the teletype are controllcd by the 110 oic ulti le. '1 he ieletype is able to encode an alphanumeric symbol to an t hit word dad deeialc an 8-bit word into an alphanumeric symbol.

a. Describe how the processor. using.the first four registers listed in this problem, can achieve. TIO with the teletvp...

b. Descrik how the Function Can lie performod more efficiently by also e mploying TEN.

Figii ini11122HL.N rikiii.a] arbitration scheme that can be used with an obsu-i,th• as Midi thus T. Agents are daisy chained physically in priority wile'. left-most agent *in the* diagram receives a constant *but, prioriiy /it* (BPR\) signal indicating that no highcr-priorily agent desircs• the bus. If the agent does not wish the bus. it asserts *its bus priority* *BPRO) I.iu e.* At thm beginning of

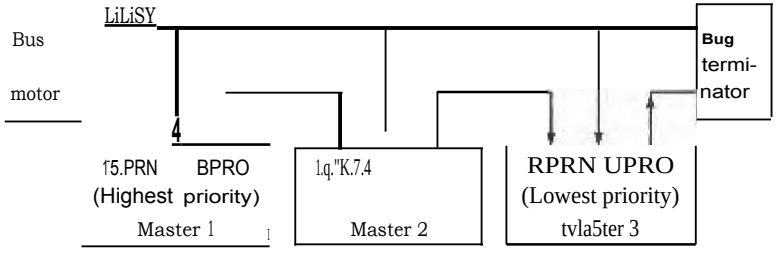


Figure 3.26 Mtillilnp' I 1) Jaributea Arbitration

a clock cycle. any agent can request control or the bus by lowering 131[1]R0 line. This lowers the BPRN line of the next agent in the chain, which is in turn required to lower its BPRO line. 'f'hus, the signal is propagated the length of the chain. At the end of this chain reaction, there should be only one agent whose BPRN is asserted and whose BPRO is not, This agent has priority. lf, the beginning of a bus cycle. the bus is not busy (BUSY inactive), the agent that has priority may seize control of the hus by asserting the BUSY line,

It lakes a certain amount of time for the BPR signal to propagate from the highest-priority agent to the lowest Must this time he less than the clock cycle? Explain,

3.? The VAX SIEff bus uses a distributed, synchronous arbitration scheme-. F:ach SBI device (i.e., processor, memory. 1.0 module) has a unique priority and is assigned a unique transfer request (TR) line. The SBI has 16 such lines (TRO, TR1, _ TR15), with TR(} having the highest priority. When a device wants to use the bus, it places a reservation for a future time slot by asserting its 'T R line during the current time slot, AL the end or the current time slot, each device with a pending reservation examines the TR lines; the highest-priority device with 7 reservation uses the next time slot,

A maximum of 17 devices can be attached to the bus. The device with priority 16 has no TR line. Why not?

3.9 Paradoxically, the lowest-priority device usually has the lowest average wait time. For this reason. the processor is usually given the lowest priority' on the SBI. Why does the priority 16 device usually have the lowest average wait time? Under what circumstances would this not he true'?

3.10 Draw and explain a timing diagram for a PCI write operation (similar to Figure 3.23),

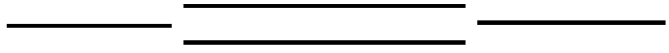
APPENDIX 3A TIMING DIAGRAMS

In this chapter. timing diagrams are used to illustrate sequences of events and dependencies among events. For the reader unfamiliar with timing diagrams, this appendix provides a brief explanation.

Communication among devices connected to a bus takes place along a set of lines capable of carrying signals. Two different signal levels (voltage levels), representing binary 0 and binary 1, may he transmitted. A timing diagram shows the signal level on a line as a function of time (Figure 3.27a). By convention, the binary I signal level is depicted as a higher level than that of binary 0. Usually, binary 0 is the default value. fhat is, if no data or other signal is being transmitted, then the level on a line is that which represents binary 0. A signal transition from 0 to 1 is frequently referred to as the signal's *leading edge:* a transition from l to 0 is referred to as *a trailing edge*_ Such transitions are not instantaneous, but this transition lime is usually small compared with the duration of a signal level. For clarity, the transition is usually depicted as an angled line that exaggerates the relative amount of time that the transition takes. Occasionally, you will see diagrams that use vertical lines, which incorrectly suggests that the transition is instantaneous. On a timing diagram, it may happen that a variable or at least irrelevant amount of time elapses between events of interest. This is depicted by a gap in the time.

Signals are sometimes represented in groups (Figure 3.27b). For example. if data are transferred a byte at a time, then eight lines are required. Generally, it is. not important to know the exact value being transferred on such a group, but rather whether signals are present or not

(a) Signal as a function of time

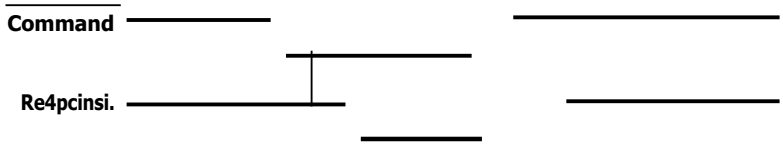


All lines at Ik

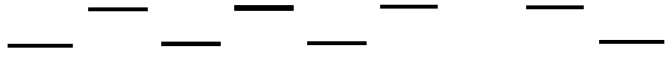
Each line may hi. 0 ɪᴘ r

All lines at 0

Croups (if line.,



{c) Cause-and-effect dependencies



Id) Cluck signal

Figure 3.27 liming Diagrams

A signal transition on one line may trigger an attached device k 'slake sinal changes on other lines. For example.. if ȃ memory module di,:tects **a read** control signal ((l or 1 traniLion), it will place data signals on 1he data lines. Such eau:94-41nd-effc.ei re14itionships produce. sequences of events. Arrows are used ɪ H1 **Inning** diagrams to show these dependencies (Figure 3.27c).

In Figure 3.27c, the overbar over the signal name indicates that the signal is active [ow ws **shown, For example, Command iS ,icEive, or** asserted, at 0 voIts. This means drat Command = is interpreted as logical 1, or true,

A clock line is often part of a system bus. An electronic clock is connected to the **clock Lint:.** and provides a repetitive, r ul,ir **sequence of** transitions (E"igure 3.27d). Other events may he synchronised to the clock signal.

CHAPTER 4

CACHE MEMORY

4.1 Computer Memory SyNtern Overview -

Systcms •

The Memory fiierarchy

4.2 Cache Memory Principles .

4.3 FIeiiiiit f Cache Design

Cache Size--7<1

Nlapping Function

RL..placerriunt

WriLe Pohey

Line Size.

Number of Cildic...

4.4 Pentium 4 and PowerPC Cache Organizations

Pentium 4 C...achc

PowcrPC CcIiE Organizatiot

4.5 Recommended Reading

4.6 Key Terms, Review Quetions, and Problerns

Term;

Rovic..w Ouestion

Probl

Appendix 4A Performance Characteristics of Two-Level Memories

LocEdily

operaLion of Two-Leel Memory

Pe.rformanco

KEY POINTS

- Computer memory is organized into a hie-ravehy. At the highest level (closest to the processor) are the processor registers. Next comes one or more levels of cache. When multiple leve15., are used, they are denoted LI. L2, etc. Next comes main memory, which is usually made out of dynamic random-access memory (DRAM). All of these a›re considered internal to the.computer system. The hierarchy continues with external memory, with the next level typically being a fixed hard disk, and one or more levels below that consisting of removal* media such as ZIP cartridges, optical disks. and tope.
- As one goes down the memory hierarchy, one finds, decreasing cosi/bit, increasing capacity, and slower access time. .11 would be nice to use only the fastest memory, but because that is the most expensive. memory, we trade off access time for cost by using more of the slower memory. The trick is to organize the data and programs in memory so that the memory words needed are usually in the faster memory.
- In general. it is likely that most future accesses to main memory by the processor will be to locations recently So the cache automatically retains a copy of some of the recently used words from the DRAM. If the cache is designed properly, then most of the time the processor will request memory words that ace already in.the cache.

Afthough seemingly simple in concept, computer memory exhibits perhaps the: widest range of type. technology, organization. performance, and cost of any feature of a computer system. No one technology is optimal in satisfying the memory requirements for a computer system. As a consequence, the typical computer system is equipped with a hierarchy of memory subsystems, some internal to the system (directly accessible by the processor) and some external (accessible by the processor via an 110 module).

'I bis chapter and the next focus on internal memory elements, while Chapter 6 is devoted to external memory. To begin, the first section examines key characteristics of computer memories. The remainder of the chapter examines an essential clement of al] modern computer systems: cache memory,

4.1 COMPUTER MEMORY SYSTEM OVERVIEW

Characteristics of Memory Systems

The. complex subject of computer memory is made more manageable if we classify memory systems according to their key characteristics. The most important of these are listed **in f** able 4.1,

Table 4.1 Key' Characteristiai of C:{riptitt..1 Mernoty Sysients

1.04.-ation	lierformitriee
Procussa.	Access time
fritt•rna.1	CyCie. tI me
ExcerItal (secorklar:Ii.)	'fraosfer rile
Capiicity	Physical **'type**
word size	SQmiconductor
Number of words	Magnetic
[Jai of Transfer	
	.M.HgooLo **-op/ i**
Block	Physical Characteristics
Access Method	lerolatileirionvolaLile
	ErasnbleinorLerusable
	tlrgaitizatiort
Rand noi t	

The term location in Table 4.1 refers to whether memory is internal and external to the computer. Internal memory is often equated with main memory_ But there are other l'orms of internal memory. The processor requires its own local memory, in the form of registers (e.g., see Figure 2.3). Further. as we shall see, the control unit portion of the processor may also require its own internal memory. We will defer discussion of these latter two types of internal memory to later chapters. Cache is another form of internal memory. External memory consists of peripheral storage devices, such as disk and tape, that arc accessible to the processor via I/O conirollus.

An obvious characteristic of memory is its **capacity.** For internal memory, this is typically expressed in terms of hyles (I byte- = ii bits) or words. Common word lengths are 8, 16, and 32 bits. External memory capacity is typically expressed in terms of bytes.

A related concept is the **unit of transfer,** For internal memory, the unit of transfer is equal to the number of data lines into and out of the memory module. This may be equal to the word length, but is often larger. such as 64. 128, or 256 bits. To clarify this point, consider three related concepts for internal memory:

- **Word:** The "natural" unit of organization of memory. The **size** of the word is typically equal to the number of bits used to represent a number and to the instruction length. Unfortunately. there are many exceptions. For example, the CRAY C90 has a 64-bit word length but uses a 46-bit integer representation. The VAX has a stupendous variety of instruction lengths, expressed as multiples of bytes. and a word size of 32 bits.
- **Addressable units:** In some systems, the addressable unit is the word. However, many systems allow addressing at the. byte level_ In any case. the relationship between the length in bits **A of an** address and the number N of addressable units is = N.

- Unit of transfer; For main memory, this is the number of hits read out of or written into memory at a time. The unit of transfer need not equal a word or an addressable unit. For external memory, data are often transferred in much larger units than a word, and these are referred to as blocks_

Another distinction among memory types is the **method of accessing** units of data. These include the following:

- **Sequential access:** Memory is organized into units of data, called records. Access must he made in a specific linear sequence. Stored addressing information is used to separate records and assist in the retrieval process. A shared read/write mechanism is used, and this must be moved from its current location to the desired location, passing and rejecting each intermediate record. Thus, the time to access an arbitrary record is highly variable. Tape units, discussed in Chapter 6, are sequential access,
- **Direct access:** As with sequential access, direct access involves a shared read—write mechanism_ However, individual blocks or records have a unique address based on physical location. Access is accomplished by direct access to reach a general vicinity plus sequential searching, counting. or waiting to reach the final location. Again, access time is variable. Disk units, discussed in Chapter 6. are direct access.
- **Random access:** Each addressable location in memory has a unique, physically wired-in addressing mechanism. The time to access a given location is independent of the sequence of prior accesses and is constant. Thus, any location can be selected at random and directly addressed and accessed. Main memory and some cache systems are random access.
- **Associative:** Ch is is a random-access type of memory that enables one to make a comparison of desired hit locations within a word for a specified match, and to do this for all words simultaneously. Thus, a word is retrie **ved** h **OD** a portion of its contents rather than its address. As with ordinary random-access memory. each location has its own addressing mechanism_ and retrieval lime is constant independent of location or prior access patterns. Cache memories may employ associative access.

From a user's point of view, the two most important characteristics of memory are capacity and **performance.** Three performance parameters arc used:

- Access **time (latency):** For random-access memory, this is the time it takes to perform a read or write operation. that is, the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use. For non-random-access memory, access time is the time it takes to position the read—write mechanism at the desired location.
- **rtie mo ry cycle time:** This concept is primarily applied to random-access memory and consists of the access time plus any additional time required before

a second access can commence_ This additional time may be required for transients to die out on signal lines or to regenerate data if they are read destructively. Now that memory cycle time is concerned with the system bus, not the processor_

Transfer rate: This is the rate at which data can he transferred into or out of a memory unit_ I tor random-access memory, it is equal to 11(cycle time).

For non-random-access memory, the following relationship holds;

$$= \frac{rV}{R}$$

where

$T.$ = Average time to read or write N bits
Average access time
N = Number of *hits*
R = Transfer rate, in bits per second (bps)

A variety of physical types of memory have been employed. The most common today are semiconductor memory. magnetic surface memory, used for disk and tape, and optical and magneto-optical.

Several physical characteristics of data storage are important. In a volatile memory, information decays naturally *or ii* lost when electrical power is switched off_ In a nonvolatile memory, information once recorded remains without deterioration until deliberately changed; no electrical power is needed to retain information_ Magnetic-surface memories are nonvolatile. Semiconductor memory may he either volatile or nonvolatile. Nonerasable memory cannot be altered, except by destroying the storage unit. Semiconductor memory of this type is known as *read-only memory* (ROM). Of necessity, a practical nonerasable memory must also he *n* onvolatil *e.*

For random-access memory, the organization is a key design issue. By *organizarion* is meant the physical arrangement of bits to form words, The obvious arrangement is not always used, as will be explained presently.

The Memory Hierarchy

The design constraints on a computer's memory can be summed up by three questions: How much? How fast? How expensive?

The question of how much is somewhat open ended. If the capacity is there. applications will likely be developed to use h. The question of how fast is, in a sense. easier to answer_ To achieve greatest performance, the memory must be able to keep up with the processor. That is, as the processor is executing instructions, we would not want it to have to pause waiting for instructions or operands. The final question must also be considered. For a practical sy stern , rhocost of memory must be reasonable in relationship to other components.

As might be expected. there is a trade-off among the three key characteristics of memory: namely, cost, capacity..and access time. At any given time, a variety of

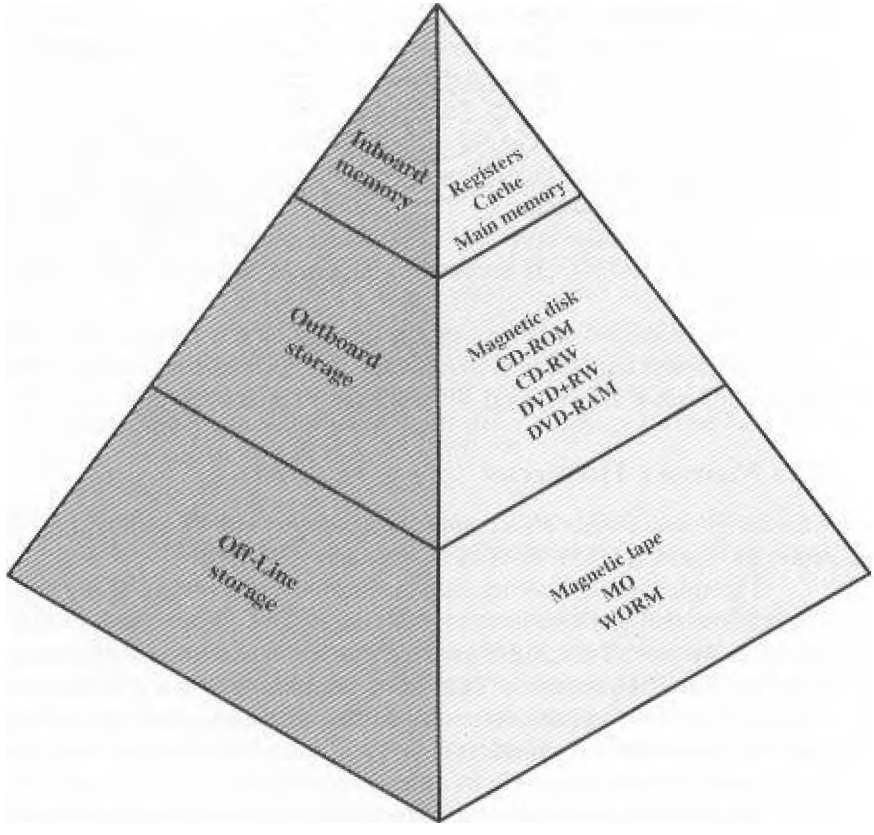
ieetinologies are. used to implerncut memory systems. Across this spectrum of kcch-riologies, the following relationships hold:

* FasLer access time, greater cost pi2r hit
* Oreatercapacity, smaller cost per hit
* Cireater capacity, s l ArkL # time

'Clic dilemma facing the designer is clear. The designer would like to use 111Qm-ory technologies abut provide for lare-capacit!,' mcmory. both because ihc capacity is. needed 4nd because the cosi per bit is low, How to meet periormance rcquirerri4,:nt, the designer neeth I o Lin expinisive, relatively lower-capacity memo $ with short access times.

The way out of this diluriima is riot to rely on a :iiagje memory compon'ent or technology, hui Lo employ a mpiory hierarchy. A typical hierarchy is illustrated in Figure 4.1...AN one goes down lhc hierateliv, the following occur!

41. Beercasing cost per bit
b. Increasing capacity



ligurc 4.1 The Mu.r[14..ir$' .1.1iCcarchy

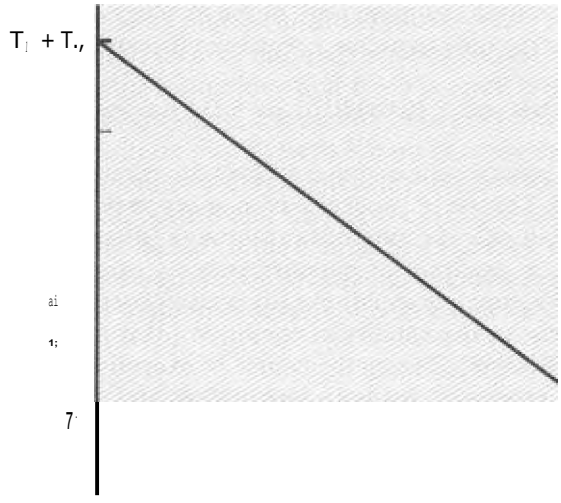


Figure 4.2 Performance of a Simple Two-Level Memory

C. Increasing access time

d. Decreasing frequency of access of the memory by Lift processor

Thus, smaller, more expensive., faster memories are supplemented by larger, cheaper, slower memories. The key to the sulLES!..ti of this organization is item (d): decreasing frequency of access. We examine this concept in greater detail when we discuss the cache. later in this chapter, and virtual memory in Chapter 8. A brief explamilicin is provilticAl a1 this pc..)inl.

Suppose.: that the processor has access to tw.o levels of MCMOly. Level 1 contains 1000 words and has an access time of 0.01 level 2 contains 100,000 words and has an access time of 11.1 Ills. Assume, that if a word to be accessed is in level L then the processor ac,cesses it directly- If is in level 2, then the word is first transferred to level 1 and then accessed by the processor. For simplicity, we ignore the ti me required for the processor to determine. whether the word is in level 1 or level 2. Figure 4.2 shows the general shape of the curve that covers this situation. The figure shows the average access time to a two-level memory as a function of the hit ratio *H,* where

1/ = fraction of all memory accesses that are found in the faster memory the cache)
1'| = access Lime 10 level
T, = aceem time to level 2

As can he seen, fur high percentages of level I access, the average total access time is much closer to that of level 1 than that of level 2.

In our example, suppose 95% of the memory accesses are found in the cache. Then the average time to access a word can be expressed as

(0,95) (0.01 p.$) + .(0.05) (0,01 µs [4]- 0.1 n.$) = 0,0095 — 0.0055 = 0,015 is

In this example, the average access time is much closer to 0.01 1.i.s than to 0.1 n.s, as desired, The use of two levels of memory to reduce average access time works in principle, but only if conditions (a) through (d) apply. By employing a variety of technologies, a spectrum of memory systems exists that satisfies conditions (a) through (c). Fortunately, condition (d) is also generally valid.

The basis for the validity of condition (d) is a principle known as locality of reference I1)ENN681. During the course of execution of a program, memory references hy the processor, for both instructions and data, tend to cluster, Programs typically contain a number of iterative loops and subroutines, Once a loop or subroutine is entered, there are repeated references to a small set of instructions. Similarly, operations on tables anti arrays involve access to a clustered set of data words. Over a long period of lime, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references.

Accordingly, it is possible to organize data across the hierarchy such that the percentage of accesses to each successively lower level is substantially less than that of the level above. Consider the two-level example already presented. Let level 2 memory contain all program instructions and data, The current clusters can be temporarily placed in level 1, From time to time, one of the clusters in level 1 will have to he swapped back to level 2 to make room for a new cluster coming in to level 1. On average, however. most references will be to instructions and data contained in level 1.

1 his principle can be applied across mote than two levels 0r memory, as suggested by the hierarchy shown in Figure 4.1. The fastest, smallest, and most expensive type of memory consists of the registers internal to the processor. Typically, a processor will contain a few dozen such registers, although some machines contain hundreds of registers. Skipping down two levels. main memory is the principal internal memory system of the computer. Each location in main memory has a unique address. Main memory is usually extended with a higher-speed, smaller cache. The cache is not usually visible to the programmer or, indeed, to the processor. It is a device for staging the movement of data between main memory and processor registers to improve performance.

The three forms of mernory just described are, typically, volatile and employ semiconductor technology. The use of three levels exploits the fact that semiconductor memory conics in a variety of types, which differ in speed and cost. Data are stored more permanently on external mass storage devices, of which the most common are hard disk and removable media, such as removable disk. tape, and optical storage. External. nonvolatile memory is also referred to as secondary or auxiliary memory, These are used to store program and data files and are usually visible to the programmer only in terms of files and records, as opposed to individual bytes or words, Disk is also used to provide an extension to main memory known as virtual memory, which is discussed in Chapter R.

Other forms of memory may be included in the hierarchy. For example, large IBM mainframes include a form of internal memory known as Expanded Storage. This uses a semiconductor technology that is slower and less expensive Than that of main memory. Strictly speaking, this memory does not fit into the hierarchy but is a side branch: Data can be moved between main memory and expanded storage but not between expanded storage and external memory. Other forms of secondary memory include optical and magneto-optical disks. Finally, additional levels can he effectively added to the hierarchy in software. A portion of main memory can he used as a buffer to hold data temporarily that is to be read out to disk. Such a technique_ sometimes referred to as a disk cache.' improves performance in two ways:

- Disk writes are clustered. Instead of many small transfers of data, we have a few large transfers of data. This improves disk performance and minimizes processor involvement.
- Some data destined for write-out may be referenced by a program before the next dump to disk. In that case, the data is retrieved rapidly from the software cache rather than slowly from the disk.

Appendix 4A ex-amines the performance implications of multilevel memory structures,

4.2 CACHE MEMORY PRINCIPLES

Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memories, The concept is illustrated in Figure 4,1 There is a relatively large and slow main memory together with a smaller, faster cache memory. The cache contains a copy Or portions of main memory. When

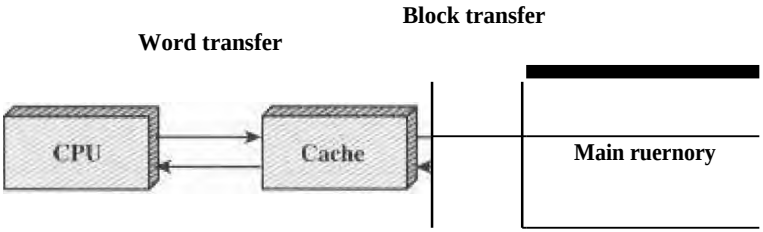


Figure 43 Cache. and Main Memory

'Disk cache is generally a purely software technique and is not examined in ihk ho4.51. :.. Sec ISTALUI] for a discussion

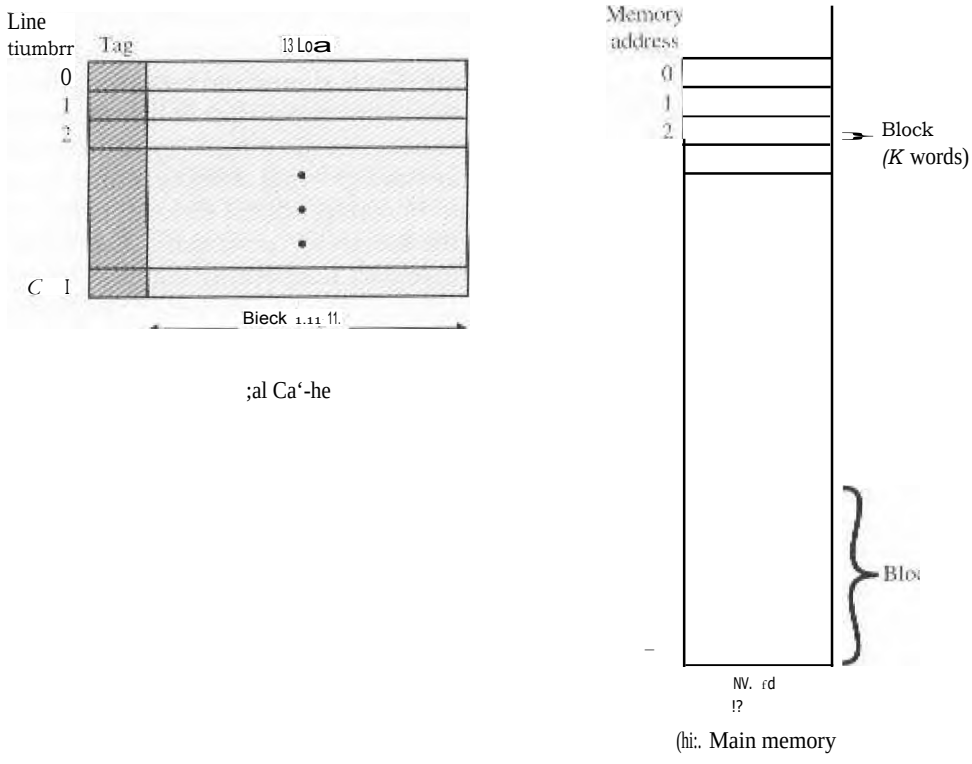


Figure 4A Caeheavlain **Memory** Structure

the processor attempts to read a word of memory, a check is made to determine if the word is in the cache. If so, the word is delivered to the processor. If not, a block of main memory, consisting of some fixed number.of words, is read ink) the cache and then the word is delivered to the processor. Because of the phenomenon of locality of reference., when a block 0r data is fetched into the cache to satisfy a single memory reference, it is likely that there will he future references to that same unemor?,/ location or to other words in)lie block.

Figure 4.4 depicts the structure of a cacheimain-memory system. Main memory consists of up to 2' addressable words, with each word having a unique n-hit address. For mapping purposes, this memory is considered to consist of a number of fixed-length blocks of K words each. That is, there are M = TIK blocks. Cache consists of *Clines* of K words each. and the number of lines is Considerably less than the number of main memory blocks $\{C \ll$ Ai). At any time, some subset of the blocks of memory resides in lines in the cache. if a word in a block of memory is read, that block is transferred to one of the lines of the cache. Because there are more blocks than lines, an individual line cannot be uniquely and permanently dedicated to a particular blmk. Thus, each line includes a **tag** that identifies which particular block is currently being stored_ The tag is usually a portion of the main memory address, as described later in this section.

Figure 4.5 illustrates the read operation. The processor generates the address, R A, of a word Io be read. if the word is contained in the cache, it is delivered to the processor. Otherwise, the block containing 1haR word k Ic..padcd into the cache:, and the word is delivered to the processor. Figure 4.: 7i shows these Last two operations occurring in parallel and reflects the organization shown in Figure 4.45, which is typical of contemporary cache organizations_ In this organization, the cache connects to the processor via data, control, and address lines. The data and address lines also attach to data and address buffers, which attach to a system bus from which main memory is reached. When j cache hit occurs, the data and **ii114.1ress** buffers are disabled and communication is, only h.clveccn pre cc5,scii- aril unche_ with no system bus traffic. When a cache miss occurs, the desired address is Loaded onto the system **hus.** and the data are returned through the data buffer to both the cache and the processor. In other organi fations, the cache is physically interposed between ihe processor and the main memory for all data, address, and control lines. In this latter case, for a cache miss, the desired word is first read into the cache and then trans, ferred from cache in processor.

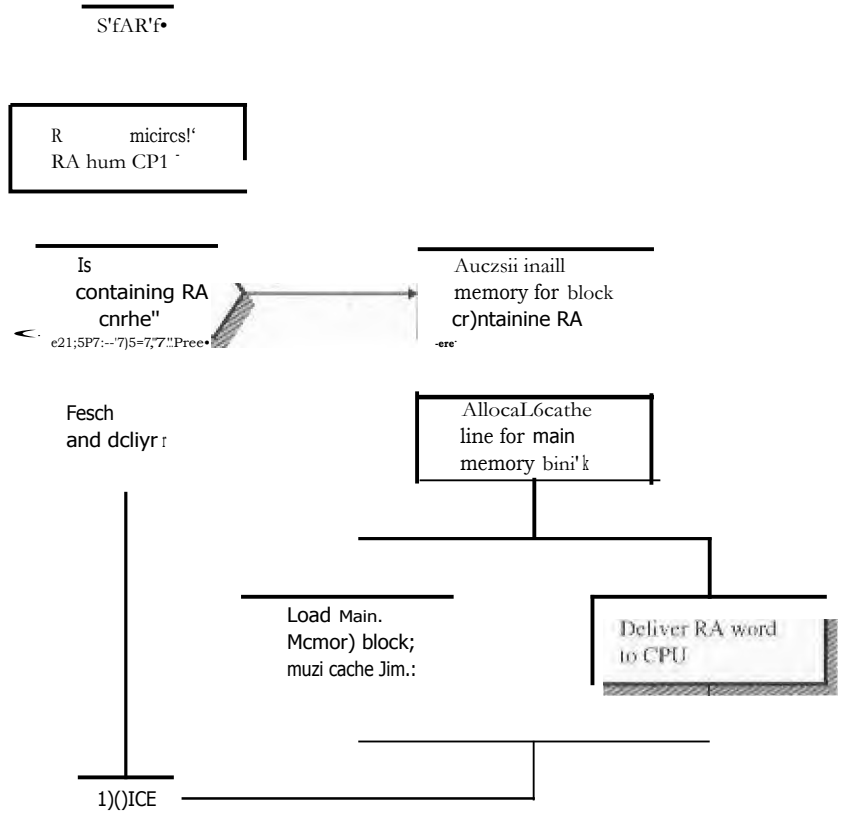
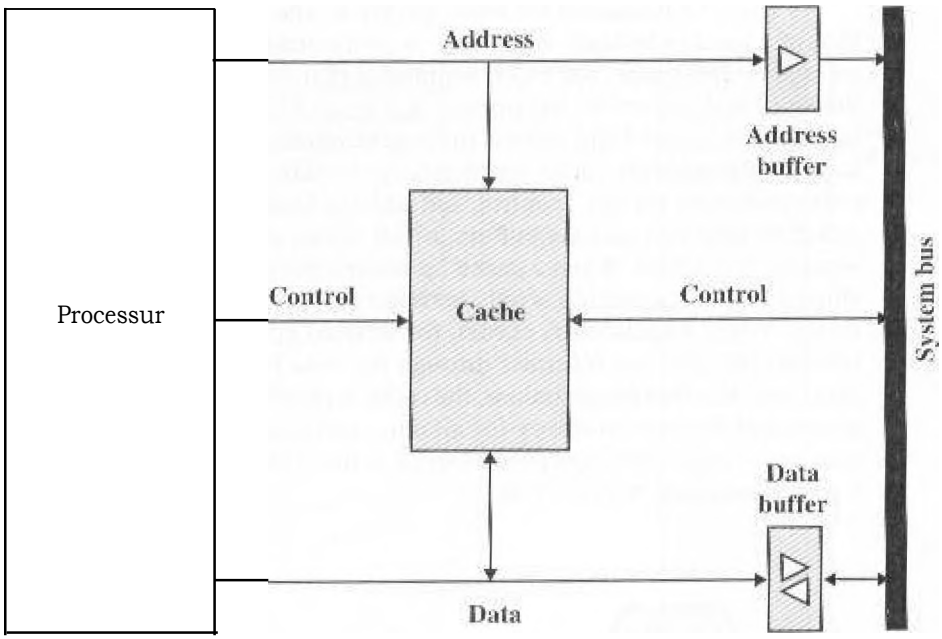


Figure 4.5 •;IL1142 1-Ze.041 O[vratin



lgure 4.6 Typical Cacho: ()rpm i7.2rLion

A discussion of i he performance parameters related to cache use iscontained in Appendix 4A,

4.3 FLENIENTS OF CACHE **DESIGN**

This section provides **an** overview of cache &sign parainetcrs and reports some typical resuli.s NT'Ve ocensionaJly refer to the **MSQ** of cadin in high-performance computing (l-IPC). HPC. deals with supereomputcrs and supercomputer SOftwEirc, especially for scientific applications that involve large amounts of data, vector and matrix computation, and the use of parallel o lgorithms. Coehe design for HPC is quite Jirroreat thFin for ol her hardware platforms mid applications. indeed., many researchers have found that MK:applicairions perform poorly on computer architectures Thai employ caches [RA IL931. Other researchers have since shown [h mclie hicrorchv can be useful in. improving performance **if** theapplication software is tuned to exploit the cache IWANC199, PRES011. [2]

Although [here are a Large number of cache implementations, thcre are a few basic design elements that SC.re to classify and differentiate cache architectures. fable 4,2 Lists key elements.

'Fo a ?erwral dsscussitni. of Hi-x7 1.1))WDLhil.

Tabl• 4.2 Eleineriis txf Cac-IIL. I] wtQil

Cache Size	Write Pohc
:Mapping Function	Wilie [hrouRb
Dircct	Write hack
Associ:k I i vc	Writc ono.:
rl ms,aicia.tik,,:%.	Line gixe
Replicernemt ANorithm	Number of emlies
roxritly uzie.4.1 a.Rif	S11101.1 iwo
First in first out (FIFO)	1 16ilied ur
ul.:1211;1.F1.:j	
Rilvidurn	

Cache Size

The first element, cache size, has already been discussed. We would like the size of the c4whe to be small enough so that the overall average cost per hit is close to that of main mentor). alone and large enough KO I hat the overall **access lime is** close to that of the cache alone. There are several other motivations for minimizing cache size, The larger the cache, the larger the number of gates involved in addressing the cache, 'Fite resu!L is Thal large caches Lend to be slightly slower than small ones—even when built with the same integrated circtti I Icchnoiogy and put in Ike same place on chip and circuit board, The available chip and board area also limits cache size. Because the performance of the cache is very sensitive lo the nature of the workload, n ix impossible **irrive** at singly •Loptimurn ¯ eaehe size. 'fable 4.,1. lists the cache sizes of some current and past processors.

Mapping Function

Because there are fewer cache lines.than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines. Further. a means is needed for deterrnininy, which main memory block **ei1 moLy** occupies as cache line. The choice of the mapping function dictates how the cache is organized. Three techniques can be used: direct. associative, and set associative. We examine each of these in turn. In each *(2;i2",* we Look at the general structure and then a specific example. For xII three cases_ the example includes the following elements:

- The cache can hold 64 '<Bytes.
- Data is Iransferred between main memory and the cache in blocks of 4 bytes each. This means that the cache is organiAed 161‹. = 2^{14} linos ol'4 hyLcs
- The main memory consists of If) Mbytes, with each byte directly addressable by a 24-bit address ($2'^{4}$ = lev1). Thus, for mapping purposes., we can consider math memory to consist of 4N1 blocks of 4 bytes each.

l he simplest technique, known as **direct mapping,** maps each block or main memory into only one possible cache line. Figure 4:7 illustrates the genera] mechanism. The mapping is expressed as

Table 43 Cache Sizes iJE Sonic. Processors

Processor	Type	Year of Introduction	L] cache'	cache	L3 cache
366t5	rame	L9M	to.32 KB		
PD1[3]-11170	MhucarripLiter	1975	1 KB		
VAX 11.7143	minicnnytorer	I91A	16KB		
111. kr 31)33	tramt2	1 L)7S	M K13		
LB .304KI	Mai n framc		128 so 2.56 KB		
Intel 9)486		1989	KB		
Pentium	N:	1993	KBIA KB.	256 Lo 512 KTh	
Powio.PC.60)	Pr:	190	32 KB		
Power PC 620	PC	1994	KEV32 KB		--
PowerPC 04	FCis.ETve	1999	KB.'32 K[3	2513 K13 ici I N113	2 NIB
113M S1391) ·C.34	Mairi[Tarn.;	19'7	32	256 KB	2 MB
IBM S3[1])0 G6	Mainframe	1.999	256 KB	S MEI	
'Pentium 4	PC-r5 mer	2u00	♯, KM'S KE-S	)5(3 KB	
]IBM SP	High-end lel r. supeLcompuier	200[1]	64 KB,62 KB	A MB	
RAY NITA"	PC/xurver	2001	16 K13d115 KB	9.6 KB	•I 1113
a Ili WU	POs,3rwcr	21101	[6 KB116 KB	96 kB	4 MB
SG1 Origin 201)1	Firah-end	20tH	32 K13.Y2 KB	MB	

N.. a sl.a0 !du)ng:.Tuv.itYr, and data cadIZE

'Run, ulchus ire inqruciii)n tsr1y ; IudeLawatiIin.

modulo fn

whrre

= cache Fine number

I .= main memory block number

Pk1 = DILISiber Or lines iii the cache

Th• mapping function easily impkcrdcntcd using the address. For purposes of cache access. ekieli main memory address can be viewed as consisting of three fields, The least significant iv bits identify i unique word Or byle within a block. of main memory; in most con Lanporar,' machine, the aldress is at the hyle Level The remaining s bits specify one, Or the blocks of Main 1I10mOry• The cache logic interprets these. s hits as a tag of s — r bits (most significant portion) and a line field of *r biLs.* This bitter field identifies one or the *on* = 2' !Ines of the cache. To summarize.

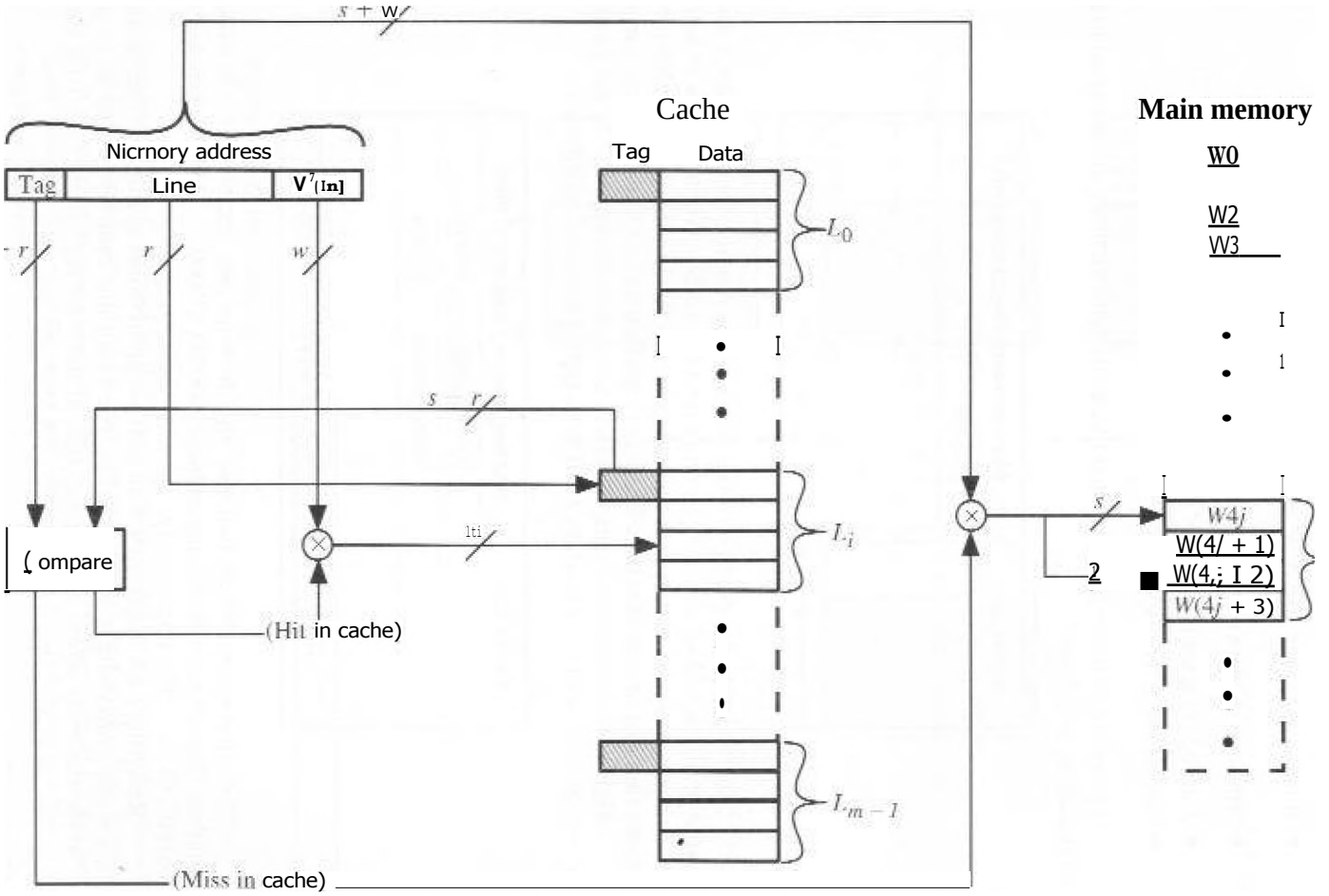


Figure 4.7 Direct-!Mapping Cache organitation [FEW AN931

- Address length = (s + w) bits
- Number of addressable units — words or hyl cw
- Block size = line size = 2 words or bytes
- Number of blocks in main memory ¯ 7,7,7 = 2'
- Number of lines in cache = 2'
- Size of lag = (5 • r) hill

The. effect of this mapping is that blocks 0r main memor!,' are assigned to lines of the cache as follows;

Cache line	Main memory blocks assigned
u	C yyr. 29.n.. .. 2' on
l	l . m -. l 2.117 - l . . 2` m • l
-	.
•	•
r00 -- I	Ai - I. 2x7•3 - I. 3m - I... . . 2'

Thus, the use of a portion of the address as a line number provides a unique mapping of each block of main memory into Inc cache- When a block is actually read into its assigned Line, it is nceessary to tag the data to distinguish ii from other blocks that can fit into that line. The most significant s r bits s.erve this purpose.

Figure 4.8 shows our example system using direct mapping,' In the example, **rrd =** 2." and i = modulo 2^4. The mapping becomes as follows;

Cache lime	Starting memory address of block
4	1}00/.300. 010000- . ..1-1^{2}001)(1
1	DO41]04. 010004 . . . FF001.4
;	•
2 - l	00FFFC.1.111-11-q."... .. FI:FFFC

Note that no Iwo blocks. I hill map into the same line number have Lhc same Lag number. Thus, blocks Willi' starting addresses 000000, 010000 FF01100 have tag 1111R1-hers 00, 01, ... FF. respectively.

Referring back to Figure 4.5, a read operation works as follows. The cache sys Lem i ptcwn14.H1 with a 24-hit address. The 14-hit line number is used as an index into the cache to access a particular line. if the s-bit lag number matches the tag

---

In khl and subsciimiik fieures. And m.r2111(3ry v:11L1122% Hit ru15rusatted h.i. lic!xadecirnal noEnlion. Sec A pp.! n fi5r.a basic rt.fresher on nunther systems (decimal, hinare.

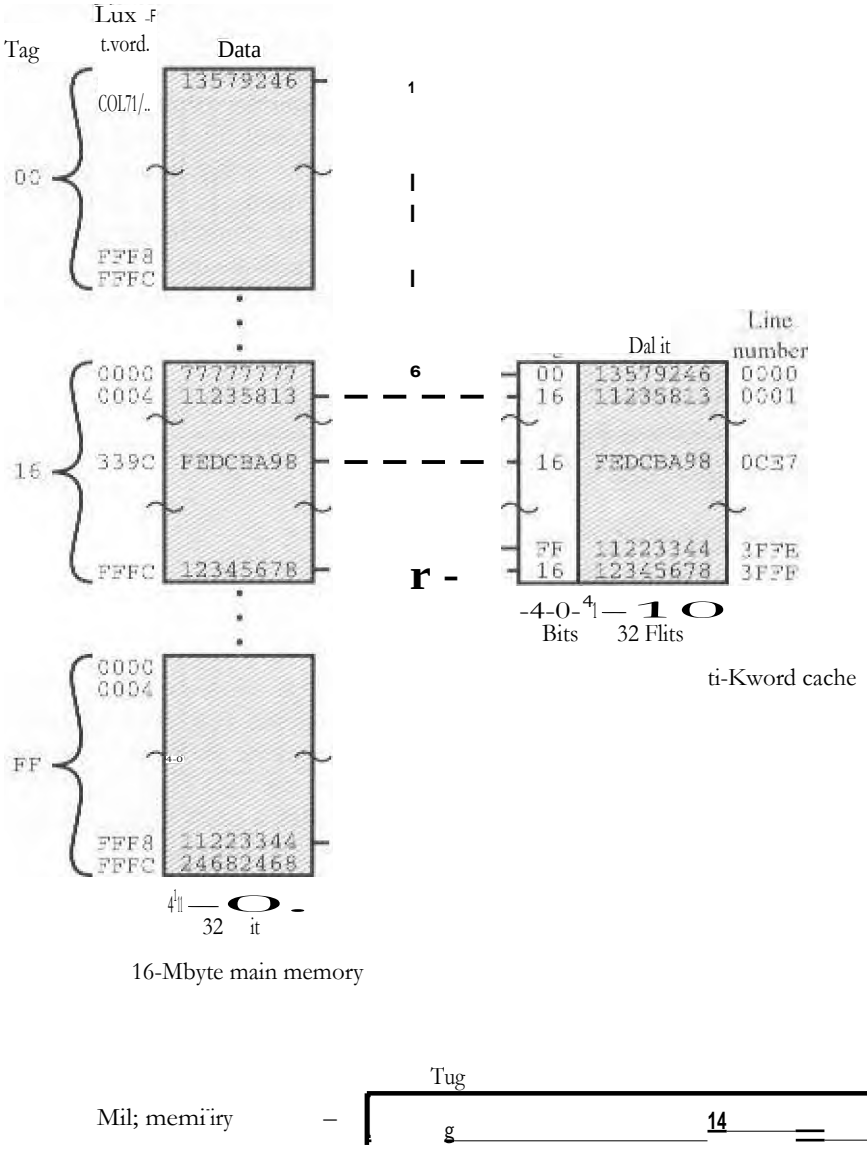


Figure 4.8 Direct Mapping Example

number currently stored in that hoe, then the 2-bit word number is used lo select one of the four bytes in that line. Otherwise. the 22-bit tag-plus-line field is owed feta block from main rnemod-. Tht: icidress that is used for the fetch is the 22-bit tag-plus-line concatenated with two El bits, so that 4 bytes are fetched starting on a block boundary.

the clirccl mapping icchniquc is simple and inoxpensive to implement. Its main disadvantage is that there is a fixed cache location for any given block, Thu s.

if a program happens lo reference words repeatedly from two different blocks that map into the same line, then the blocks will be continually swapped in the cache, and the **hit** ratio will be low (a phenomenon known as *thrashing),*

Associative mapping overcomes the disadvantage of direct **mapping by permitting** each main memory block **to** he loaded into any line of the cache. In this case, the cache control logic interprets a memory address simply as a tag and a word field. The tag field uniquely identifies a block of main memory_ To determine whether a block is in the cache. the cache control logic must simultaneousl!.,' examine every line's tag for a match. 1 [7]igure 4.9 illustrates the Logic. Note that no field in the address corresponds to line number, so that the number of lines in the cache **is not** determined by the address format. To summarize,

- Address length — w) bits
- Number of addressable units = 2 ·· words or bytes
- Block size. = line size = 2" words or bytes
- Number of blocks in main memory — =
- Number of lines in cache = undetermined
- Size of tag = *s* hits

Figure 4.1.0 shows our example using associative mapping. A main memory address consists of a 22-hit tag and a 2-bit byte number, 'Ite 22-bit tag must he stored with the 32-bit block of data for each line in the cache. Note that it is the leftmost (most significant) 22 bits of the address that form the tag.' Thus, the 24-bit hexadecimal address 16339C has the 22-bit tag 058CE.[7]. This is easily seen in binary notation:

memory address	0001	0110	0011	0011	1001	1100	(binary)
	1	6	3	3	9	C	(hex)
tag (leftmost 22 hits)	00	0101	1000	1100	1110	0111	(binary)
	0	5	8	C	E	7	(hex)

With associative mapping, there is flexibility as to which block to replace when a new block is read into the cache. Replacement algorithms, discussed later in this section, are designed to maximize the hit ratio. The principal disadvantage of assc.)-eiative mapping is the complex circuitry required to examine the tags of all cache lines in parallel.

Set associative mopping is a compromise that exhibits the strengths of both the **direct** and associative approaches while reducing their disadvantages. In this ease, the cache is divided into v sets, each of which consists of *k* lines. The relationships **arc**

•Irt Figure 4.11). the 2,2-bil 1.;15l is re.prochsc:d by a 6-digit licxadeciinal number. The most significant hexadecimal digit in fact ih milk. 2 hits in length.

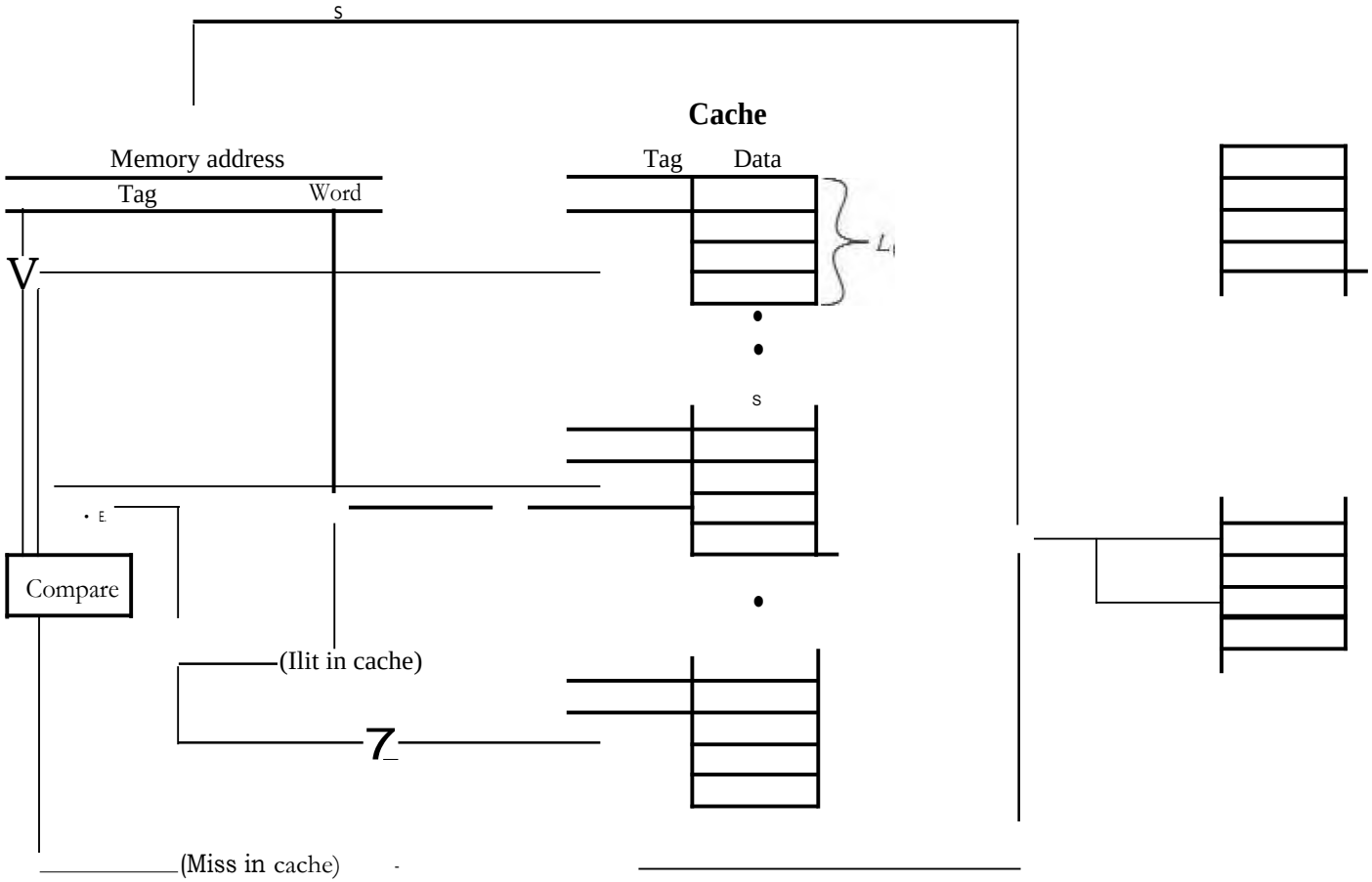


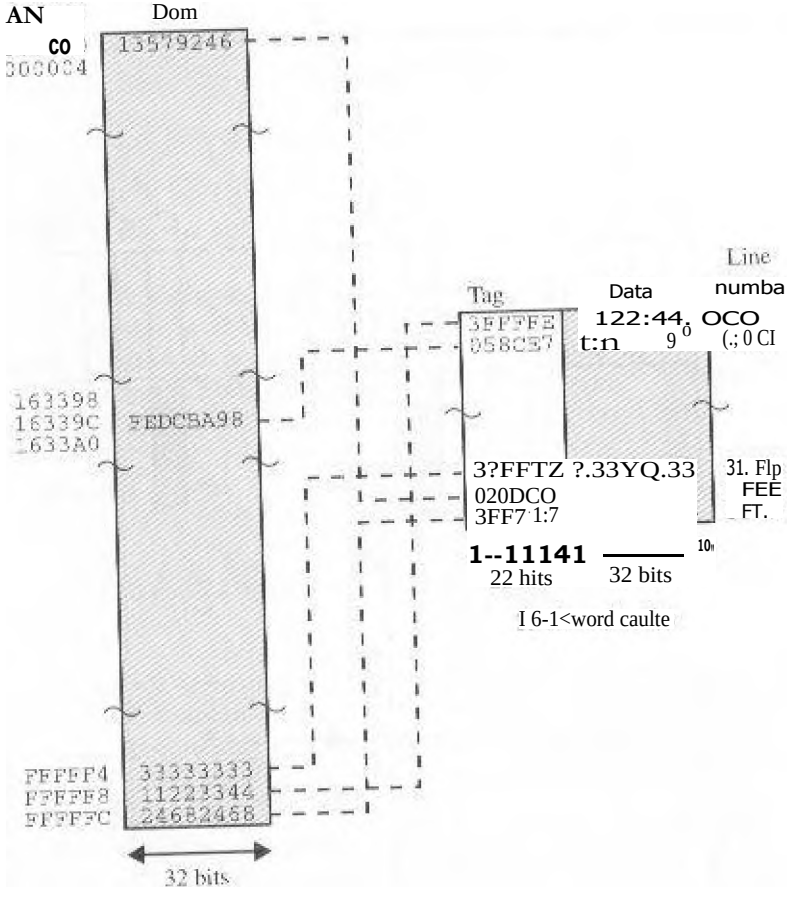
Figure 4.9 Fully Associative Cache Orwanization [I1WAN93]

$$tn \text{ —rxk}$$
$$j \text{ modulo i)}$$

whurc

= cachc set numbcr
j = rimin memory block number
m = number of lines in the cache

This is refured to as k-way set associative mapping,. With set associative mappinv, Hoch S_i cith be mapped into any of the lines of scf.i 1. in this ease, the cche control logic interprets a rnonory addre!4s simph,' us three fields: tag, set, an word, The d



1 fi-rilByte main InICTPRIty

Word

Main mrmory kuldress. — 22 12

Fi ore 4.)11. Aqqnciative Mapping

set bits specify one of if = 2^{1} sets. The s hits of the tag and set fields specify one of the 2' blocks of main memory. Figure 4.11 illustrates the cache control logic. With fully associative mapping. the lag in a memory address is quite large and must be compared to the tag of ever; line in the cache. With k.-way set associative mapping, the tag in a memory address is much smaller and is only compared to the *k tags* within a single set. To summarize,

- Address length = ⌈ iv) bits
- Number.of addressable units — 2' ·· ''' words or bytes
- Block size = line size = 2" words or bytes
- Number of blocks in main metnory= = 2'
- Number of lines in set = *k*
- Number of sets
- Number of lines in cache *k*
- Size of tag = *(s – d)* bits

Figure 4.12 shows our example using set associative mapping with two lines in each set, referred to as two-way set associative.' The 13-bit set number identifies a unique set of two lines within the cache. It also gives the number of the block in main memory, modulo 2". This determines the mapping of blocks into lines. Thus, blocks 000000. 008000 , FF8000 of main memory map into cache iset O. Any of those blocks can be loaded into either of the two lines in the set. Note that no two blocks that map into the same cache set have the same tag number. For a read operation. the 13-hit set number is used to determine which set of two lines is to be examined, Both lines in the set arc examined for a match with the tag number of the address to be accessed.

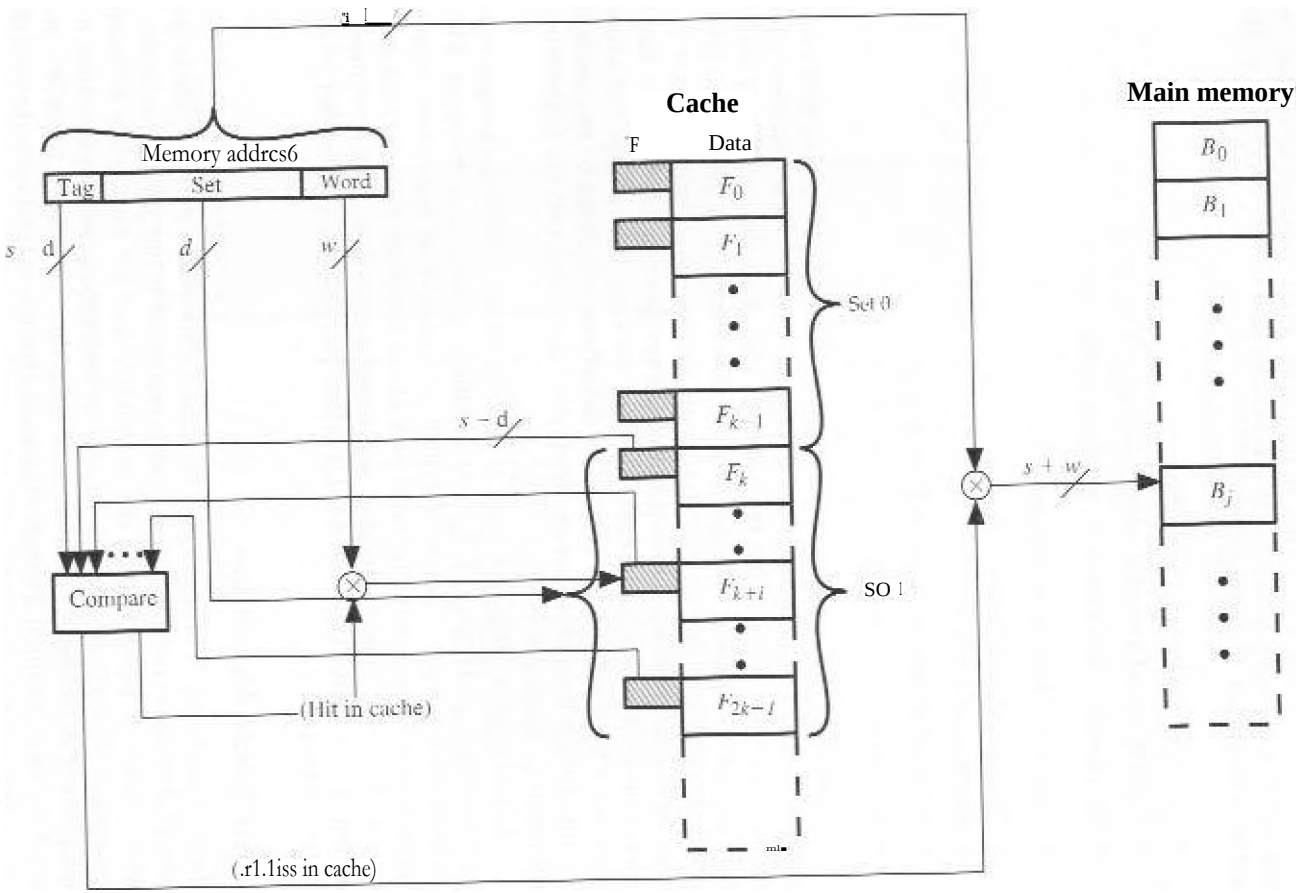
In the extreme case of r = m, *k* = 1, the set associative technique reduces to direct mapping, and for v = *k — in, it* reduces to associative mapping. The use of two lines per set (ly = *nr /2. k = 2) is the most common* sct associative organization. It significantly improves the hit ratio over direct mapping. Four-way set associative (1· = *4 - k = 4)* makes a modest additional improvement for a relatively small additional cost IMAY1384, HILL89]. Further increases in the number of lines per set have little effect.

Replacement Algorithms

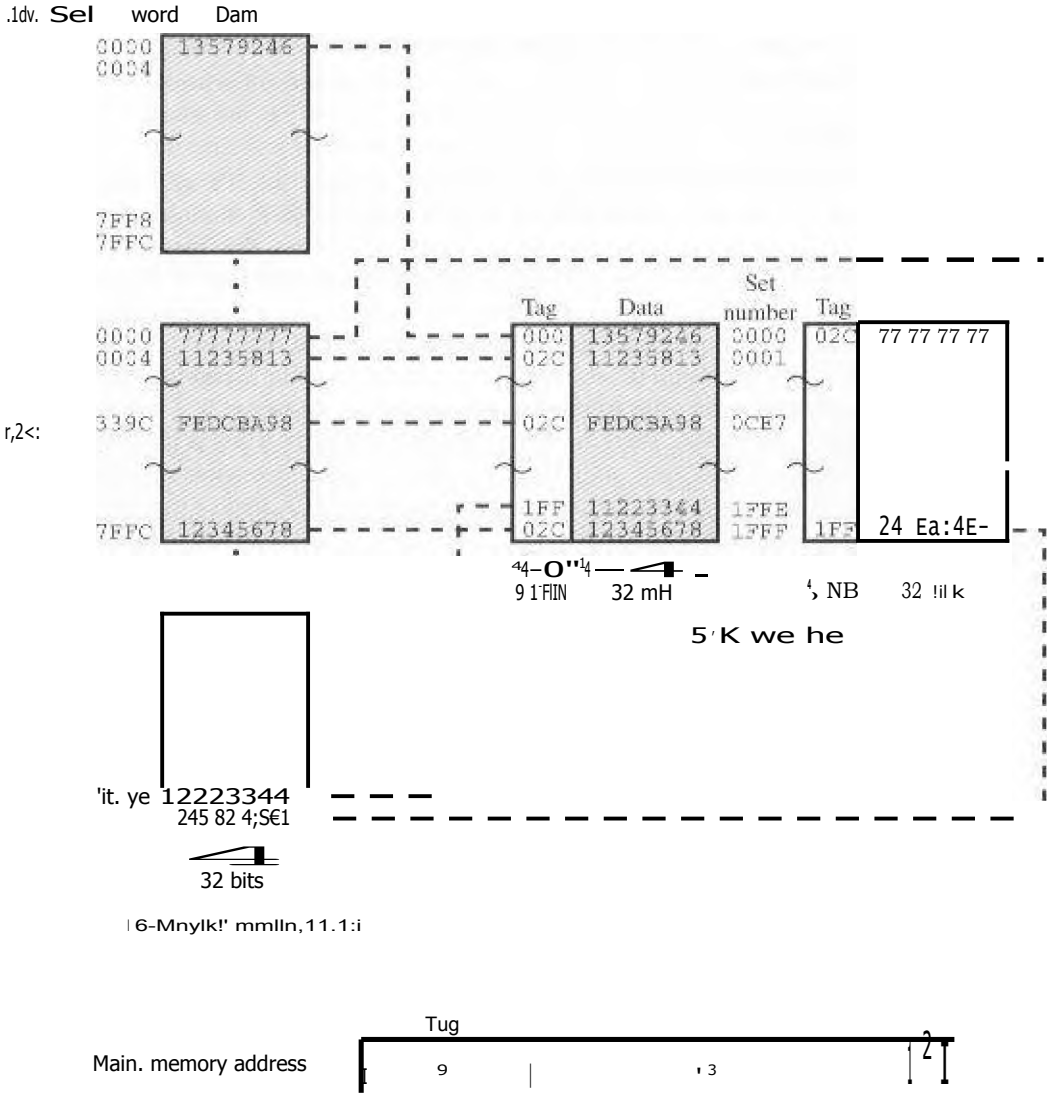
When a new block is brought into the cache, one of the existing blocks must be replaced. For direct mapping, there is only one possible line for any particular block, and no choice is possible. For the associative and set associative techniques, a replacement algorithm is needed. To achieve high speed, such an algorithm must be implemented in hardware. A number of algorithms have been tried: We mention four of the most common, Probably the most effective is least recently used (LAW): Replace that block in the set that has been in the cache longest with no reference to

---

Figure 4.1'2. the 9-bil sag is represented by a 3-digit hexadecimal numbor. The most significant hexadecimal digit in Fact is only I ail in lenglh,



lligore 4.11 k-Wav ktAssmiative Cache Organization



fi.gure 4.12 Two-Way SeL Ass6ciative Mapping Example

RBI two-way aSsoei.Nlivi,:, this is nisi ly itnplemented, Each line includes a USE HE When aHne§ ↑aecrc d, its IS hit is set to 1 and the USE bit of the other line in that set is set to.O. When a Hock is to be read into the set, th.E.iirte. whoNe USL 0 is used. Because we are assuming that morQ rQcently used memory locations are more likcly to referenced, LRI. J should give the best hit ratio. Another possibility is first-in-first-out (F[F0): Replace that block in the set ilia 1 ms been in Lhe c2iche longest FIFO c.asily implemQnicd as a round-robin or circularbuffer technique. Still another possibility is leastfrequently used (LFU): Replace that 1. ock in the set that has experienced the fewest references. LFU could bc iroplumcntc:d by associating a counter with each line. A techniqu not based on usage is to pick a line

at random from arriong the candidate lines. Simulation studies have shown that random replacement provides only slightly inferior performance to an aleorithm based on usage [SMITS2],

Write Policy

More a block that is resident in the cache can be replaced. it is necessary to consider whether it has been altered in the cache but not in main memory. if it has not, then the old block in the cache tnav be overwritten. If it has, that means that at least one write operation has been performed on a word in that line of the cache. and main memory must he updated accordingly. A variety of write policies, with performance and economic trade-offs. is possible. There are two problems to contend with. First, more than one device may have access to main memory. For example. an 110 module may he able to readlwrite directly to memory. If a word has been altered only in the cache. then the corresponding memory word is invalid. Further, if the 110 device has altered main memory, then the cache word is invalid. A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache. Then, if a word is altered in one cache. it could conceivably invalidate a word in other caches.

The simplest technique is called *write through.* Using this technique, all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid. Any other processor—cache module can monitor traffic to main memory to maintain consistency within its own cache. The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck. An alternative technique, known as *write back,* minimizes memory writes. 'Writ h write back, updates are made only in the cache. When an update occurs. an upDATE bit associated with the line is set. Then, when a block is replaced, it is written hack to main memory if and only if the UPDATE bit is set. The problem with write back is that portions of main memory are invalid, and hence accesses by 110 modules can be allowed only through the cache. This makes for complex circuitry and a potential bottleneck. Experience has shown that the percentage of memory references that are writes is on the. order of 15% [SMIT82]. However. for HPC applications, this number may approach 3 % (vector-vector multiplication) and can go as high as 5(t% (matrix transposition).

In a bus organization in which more than one device (typically a processor) has a cache and main memory is shared, a new problem is introduced. If data in one cache are altered, this invalidates not only the corresponding word in main memory, but also that same word in other caches (if any other cache happens to have that same word). Even if a write-through policy is used, the other caches may contain invalid data. A system that prevents this problem is said to maintain cache coherency. Possible approaches to *cache* coherency include. the. following:

- **Bus watching with write through:** Each cache controller monitors the address lines to detect write operations to memory by other bus masters. If another master writes k..) a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry. This strategy depends on the use of a write-through policy by all cache. controllers.
- **Hardware transparency:** Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches, 'l'hus. if one proces-

sor modifies a word in its cache. this update is written to main memory, In addition, any matching words in other caches are similarly updated.

- 1Noneacheithle memory: Only a portion of main memory is shared by more than one processor, and this is designated as noncacheable. In such a system, all accesses to shared memory are cache misses. because 1he shared memory is never copied into the cache. The noncacheable memory can he identified using chip-seleet logic or high-address bits.

Cache. coherency is an active field of research. This topic is explored further in Chapter 18.

Line Size

Another design element is the line size. When a block of data is retrieved and placed in the cache, not only the desired word but also some number of adjacent words are retrieved. As the block size increases from very small to larger sizes, the hit ratio will at first increase because. of the principle of loenlity, which states that data in the vicinity of a referenced word are likely to be referenced in the near future. As the block size increases, more useful data are brought into the cache. 'The hit ratio will begin to decrease, however, as the block becomes even bigger and the probability of using the newly fetched information becomes less, than the probability of reusing the information that has to be replaced. Two specific effects come into play:

- Larger blocks reduce the number of blocks that fit into a cache. Becauk each block fetch overwrites older cache contents, a small number of blocks results in data being overwritten shortly after they are fetched.
- As a block becomes larger, each additional word is farther from the requested word. and therefore less likely to be needed in the near future.

The relationship between block size and hit ratio is complex, depending on the locality characteristics of a particular program. and no definitive optimum value has been found. A size of from ti to 32 bytes seems reasonably close to optimum ISMIT87, PRZY88, PRZY9O. HAND98j. For TIPC systems. 64 and 128 byte cache line sizes are most frequently used.

Number of Caches

When caches were originally introduced, the typical system had a single cache. More recently, the use of multiple caches has become the norm, 'Iwo aspects of this design issue concern the number of levels of caches and the use of unified versus split caches.

Multilevel Caches

As logic density has increased, it has become possible to have a cache on the same chip as the processor: the on-chip cache. Compared with a cache reachable via an external bus, the on-chip cache reduces the processor",, external bus activity and therefore speeds up execution times and increases overall system performance. When the requested instruction or data is found in the on-chip cache, the bus access is eliminated. Because of the short data paths internal to the processor, compared

with bus lengths, on-chip cache accesses will complete appreciably faster than would even zero-wait state bus cycles. Furthermore, during this period the bus is free to support other transfers.

The inclusion of an on-chip cache leaves open the question of whether an off-chip. or external, cache is still desirable. Typically, the answer is yes. and most contemporary designs include both on-chip and external caches. The resulting organization is known as a two-level cache, with the internal cache designated as level 1 (1-1.) and the external cache designated as level 2 (L2). The reason for including an L2. cache is the. following. If there is no L2 cache and the processor makes an access request for a memory location not in the LI cache, then the processor must access []RAM or ROM memory across the bus. Due to the typically slow bus speed and stow memory access time, this results in poor performance. On the other hand, if an L2 SRAM (static RAM) cache is used. then frequently the missing information can he quickly retrieved. if the SRAM is fast enough to match the bus speed, then the data can be accessed using a zero-wait state transaction, the fastest type of bus transfer.

Two features ()I' contemporary cache design for multilevel caches are noteworthy. First, for en off-chip 1.2 cache, many designs do not use the system bus as the path for transfer between the L2 cache and the processor, but use a separate data path, so as to reduce the burden on the system bus. Second, with the continued shrinkage of processor components, a number of processors now incorporate the L2 cache on the processor chip, improving performance.

The potential savings due to the use of an 1..2 cache depends on the hit rates in both the Ll and 1...2 caches, Several studies have shown that, in general, the use of a second-level cache does improve perfOrmance (e.g., see [AZ1M92J, INOVI93]. IIIAND98]). I lowever, the use of multilevel caches does complicate all of the design issues related to caches, including size. replacement algorithm, and write policy; see [HAND981 and [PEIR99] for discussions.

Unified versus Split Caches

When the on-chip cache first made an appearance, many of the designs consisted of a single cache used to store rcicrences to both data and instructions. More recently, it has become common to split the cache into two; one dedicated to instrue-Lions and one dedicated to data.

There arc two potential advantages of a unified cache:

- For a given cache size, a unified cache has a higher hit rate than split caches because it balances the load between instruction and data fetches automatically. That is, if an execution pattern involves many more instruction fetches than data fetches, then the cache will tend to fill up with instructions, and if an execution pattern involves relatively more data fetches. the opposite will occur.
- Only one cache needs to be designed and implemented.

Despite these advantages, the trend is toward split caches, particularly for superscalar machines such as the Pentium and Poi.verPC., which emphasize parallel instruction execution and the prefetehing of predicted future instructions. The key advantage of the split cache design is that it eliminates contention for the cache between the instruction fetchfdccode unit and the execution unit. This is important in any design that relies on the pipelining of instructions, Typically, the processor

will fetch instructions ahead of time mid lilt a buffer. or pipeline, with instructions Lc) be executed. Suppose now that v•e have a unified instruction/data cache. When the execution unit performs a memory access to load and store data, the request is submitted to the unified cache. If. at the same„ time, the instruction prefetcher issues a read request to The cache for an instruction, lhril request will be temporarily blocked so that lite cache can service the execution unit first, enabling it to complete the currently CHeekiLing instruction. This cache contention can degrade performance by interfering with efficient use of the instruction pipeline. The split cache structure overcomes this difficulty.

4.4 PENTIUM 4 AND POWER.1[1]C CACHE ORGANIZATIONS

Pentium 4 Cache Organization

The evolution of cache organization is seen eleiirly in the evolution of Intel microprocessors. The S0386 does not include an on-chip cache. The 80486 includes a sin-gte On-chip cache of 8 Kilytes, using a line, size of 16 bytes and a four-way set associative organization. All of the Pentium processors include two on-chip 1.1 caches, one for data and one for instructions. For the Pentium 4, the Li akila cache is g ((Bytes, using a line size of 64 bytes and a four-way set associative. 0
The Pentium 4 instruction cache is described subsequently The Pentium 4 also includes an L2 cache that feeds both of the L1 caches. The L2 cache is eight-wav set associalive with a size of 256KB and a line size of l2K bytes.

Figure 4-].3 provides a simplified view of the Pentium 4 organization, highlighting the placement of the three caches- 'The processor core consists of four major components;

* **Fetch/decode unit;** Fetches program instructions in order from the *U* cache, decodes these into a series of micro-operations. and stores the results in the LI instruction cache.
- **Lii-of-order execution logic:** Schedules execution of the rnicto-operations subject to data dependencies and resource availability; thus, micro-operations may be scheduled for execution in a different order than they were fetched from the instruction stream. As time permits, this unit .schedulesspecuLtivc execution of micro-operations that may be required in the future.
- **Execution** units: These units executes micro-operations, fetching the required data from the LI data cache and temporarily storing **rCsul1r** in registers.
- **Memory subsystem: This** unit includes the L2 cache and the system bus, which is used hr access **main** memory when the LE and L2 caches have a cache miss, and to aceess the system resources.

Unlike the organization used in all previous Pentium modets. and in most other processors, the Pentium 4 instruction cache sits between the instruction decode logic and the execution core. The reasoning behind this design decision is as follows. **dis-**cussed more fully in Chapter 14, the Pentium process decodes, or translates, Pentium machine instructions into simple RISC-like instructions called micro-operations. The use of simple, fixed-length micro-operations enables the use of superscalar pipelining and scheduling techniques that enhance performance. However, the Pentium machine

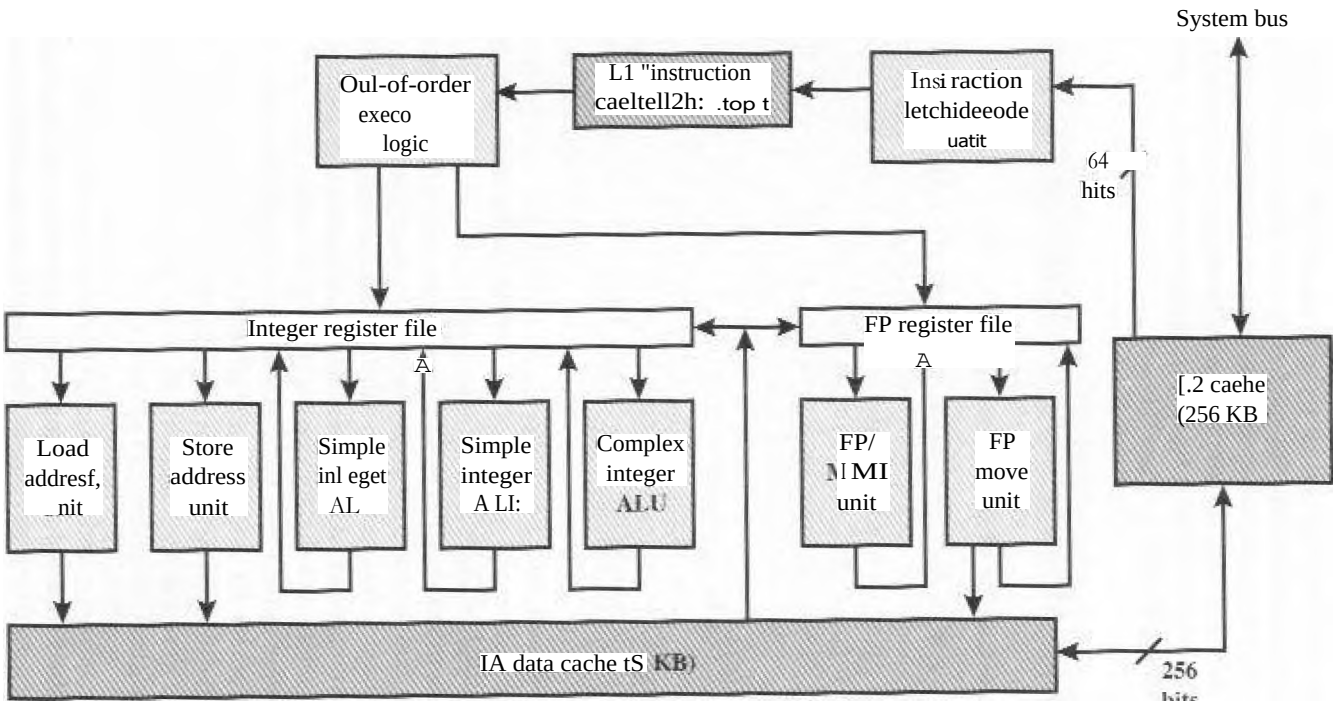


Figure 4.13 Pentium 4 Block Diagram

Table 4,4 I'...! ntiurn 4 C ai.Ji..(ɔn__berating IN.1(5LICS

Contra' Bits		operating ?dude		
	NW	Cache FilLs	Write Throughs	Invalidates
4l	4L	Enabled	Enabled	Enabled
1.	4'1	Disabled	Enabled	I:mobled
J.	L	riiiiiblud	Di,iubled	Disabled

C() = U: = L is.aq'suvsliJ coinhirlariori.

instructions are cumbersome to decode; they have a variable number of bytes and many differont c}plions. IL turns out that performance is enhanced if this decoding is done independently of the scheduling and pipelining [ogle. We return lo this topic in Chapter 14.

The data cache employs a write-back policy! Data are written to main memory only when they arc removed from the cache and there]ias been an update. The Pentium 4 processor can he dynamically eoririgurcd lo support write-through caching.

The LI data cache is controlled by two bits in one of the control registers, labeled the CD {cache disable} and NW (not write-through) bits (Table 4,4). There are also two Pentium 4 instructions that can be used to control the data cache: LNVD invalidates (flushes) the intern& cache rnaulory and signals nic external cache (if any) to invalidate. WB1NVD writes hack and invalidates internal cache, then writes hack and invalidates external cache.

PowerPC Cache Organization

The PowcrPC cache organization]ias evolved with the overall architecture of the PowerPC family, reflecting the relentless pursuit of performance that is the driving force for a]] microprocessor designers.

Table 495 shows this evolution. The original model, the 601, includes a single codcidain 32-k Byte cache that is eight-way set associative. The 603 employs a more sophisticated RISC: design but has a smaller cache: let KBytes divided into separate instruction and data caches, both using two-way set associative organiz,ation. The result is that the 603 gives approximately the same performance as the Mil at hrwer cost. The 64I4 and 620 each doubled the size of the caches from the preceding model. The *US* and G4 models has the same size 1,1 caches 4i2S the 620.

Figure 4.14 provides a simplified view of the PowerPC G4 organization, highlighting the plac.emen I of the two caches. The core execution units are two integer

Table 11.5 PowerPC Internal Caches

Model	Sixc	Bytes/Line	Organization
PowerPC: 61:11	1 32-KbytE	32	8-wily set associaLive
I·45werl*C 1503	2 S•Kbyte	32	2-way set rissociatik·e
PowerPC 604	2 1E.-Kby	32	4-way SQL assnclatiAv
PowerPC 620	2 32-KbyLe	64	8-9futi seL aV.LaCiaLi ...Lt.
PoW4211*C 03	2 32•KbyLe	64	-slioiy set iiifSociativo
PowerF'C.: G4	2 '32-E(1.v!elo	32	Sway set assmative

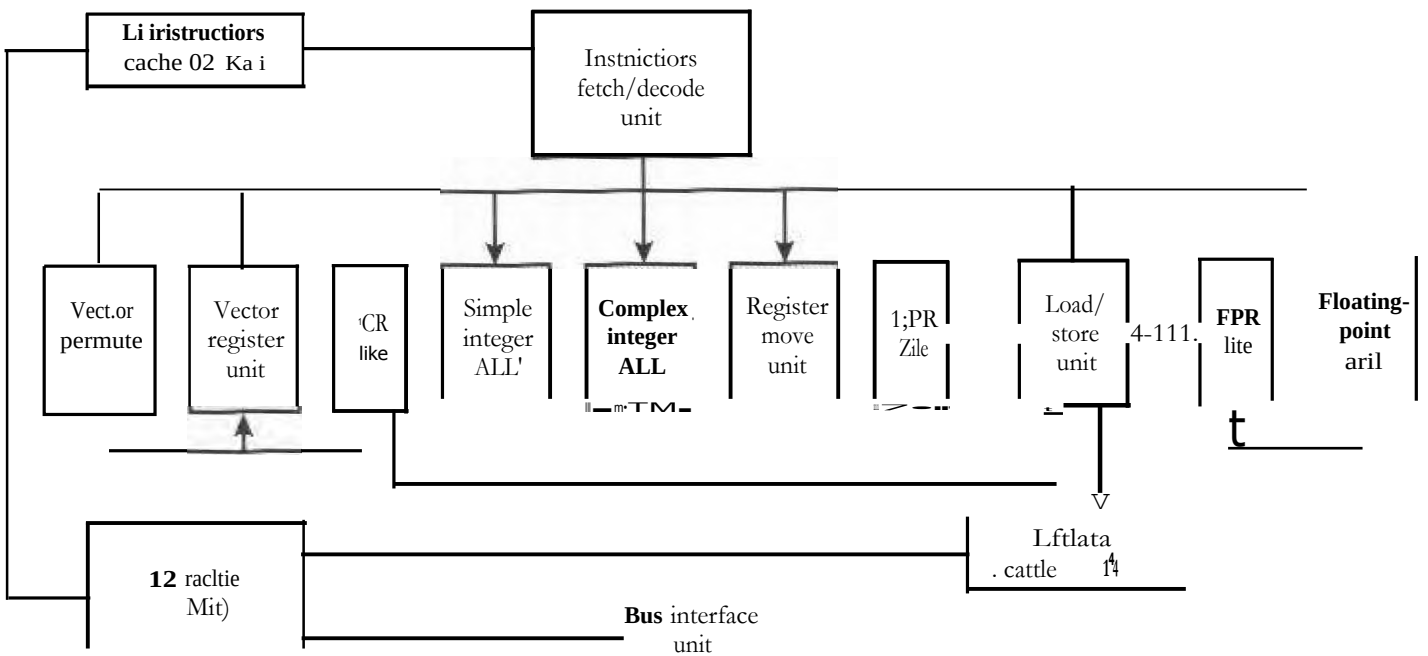


Figure 4.14 PowerPC G41 rock Diagram

arithmetic. and logic units, which can execute in parallel, and a floating-point unit with its own registers and its own multiply. add, and divide components. The data cache feeds both integer and floating-point operations via a loadIstore unit. The instruction cache, which is read only, feeds into an instruction unit, whose operation is discussed in Chapter 14.

The Li caches are eight-way set associative. The LZ cache is a two-way set associative cache with 256K. 5.12K, or I MB of tu mefy_

4.5 RECOMMENDED READING

A thorough treatment of cache design is to be found in [HAN' 1)94 A discussion of Pentium 4 cache organization can he found in ININT011 and of PowerPC 04 cache organization in IMO1 001.]. A classic paper that is still well worth reading is ISMIT821; it surveys the various elements of cache design and presents the results of an extensive set of analyses. [AGAR89] presents a detailed examination of a variety of cache. design issues related to multiprogratn-ming and multiprocessing, [HICiB901 provides as 4, 1 of simple formulas that can he used to estimate cache performance as a function of .011 kap, cache parameters.

AGA It$9 Agorwal. A. Anrrlt.cis *rif Cache Performance for Operating SySfellt% and Multi-progi.amn fing.* Boston: Kiuwer Academic Publishers, 1989,

lfl AND98 Handy. .1, *The Cache Memory Book. San .1)i* ego: •k`adeunie Press, 1993.

Highie. "Quick and Easy Cache Performanec. Analysis.' *•on2precer lecture* News, _him. 1990.

HINTO1 Hinton. 6., et al. The Mieroarcliiiecture of the Pentium 4 ProOssor.' *bad Technoloyy Journal, QI 2001. littplidevdkipc .lat* 4• [Int3.10gyAtil

M1)' 001 Motorola, *Powoe.PC* **M** U.' *RISC Mieroproiwssar lardware eeili•ii-fiCkil•,* Denver, CO: 2001. www,thotorold.eom

SMITA2 Smith, A. 'Cache Memories," r1, t'M *Omptaing Surve•s',* September 1992,

4.6 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

access time	hit ratio	sequential access
associative mpping	instruction cache.	set-asSOciative mapping
cache hit	1...1 cache	spatial locality
cache line	L2 cache	split cache
cache memory	L3 cache	tag
cache miss	locality	temporal locality
en elle Set	nieinory hierarchy	unified cache
data cache	Multilevel cache	write back
diteci access	random CCC517;	Write once
direct mapping	replacein el it algorithm	write through
high performance computing (HPC)		

Review Questions

4.1 What are the differences among sequential access. direct access, and random access?
4.2 What is the general relationship among access time, memory cost. and capacity?
4.3 How does the principle of locality relate to the use of multiple memory levels?
4.4 What are the differences among direct mapping, associative mapping, and set-associative mapping?
4.5 For a direct-mapped cache, a main memory address is viewed as consisting of three fields. List and define the three fields,
4.6 For an associative cache, a main memory address is viewed as consisting of two fields. List and define the two fields.
4.7 Fur a set-associative cache. a main memory address is viewed as consisting of three fields. List and define the three fields.
4.8 What is the distinction between spatial locality and temporal locality?
4.9 In general, what arethe strategics for exploiting spatial locality and temporal locality?

Problems

4.1 A set associative cache consists of 64 lines, or slots, divided into four-line seas. Main ',lemon contains 4K blocks of 128 words each. Show the format of main memory addresses,
4.2 For the hexadecimal main memory addresses 111111 .666666, BBBBBB. show the following information, in hexadecimal format:
   a. 'tag. Line, and Word values for a direct-mapped cache. using the format of Figure 4.8.
   b. Tag and Word values for an associative cache, using the format of Figure 4.10.
   c. Tag, Set, and Word values for a two-way set associative cache, using the format of Figure 412,
4.3 List the following values;
   a. For the direct cache example of Figure 4.8: address length, number of addressable units, block size. number of blocks in main memory, number of lines in cache. size of tag.
   b. For the associative cache example of Figure 4.10: address length, number of addressable units, block size, number of blocks in main memory, number of lines in cache, size of tag.
   e. For the two-way associative cache example of Figure 4.1.2: address length. number of addressable units, block size, number of blocks in main memory. number of lines in set, number ,±1 Sets, number of lines in cache. size of tag,
4.4 Consider a 32-bit mk:Hiproeessor that has an on-chip 16 KByte four-way set-associative cache. Assume that the cache has a tine size of four 32-hit words. Draw a block diagram of this *cache ~bowing* its organization and how the different address fields are used to determine a cache hitimiss. Where in the cache is the word from memory loca ' Lion ABM F.8F8 mapped?

   Source: lALEX931
4.5 (liven the following specifications for an external cache memory: four-way set associafive: line size of two I 6-bit words; able to accommodate a total of 4K 32-bit words from main memory; used with a 16-bit processor that issues 24-bit addresses. Design the cache structure with pertinent information and show how it interprets the processor's addresses.

   Source: [ALEX931
4.6 The Intel 80486 has an on-chip, unified cache, It contains 8 KBytes and has a four-way set associative organization and a block length of four 32-bit words. The cache is organized into 128 gets. There is a single "line valid bit" and three hits, BU. 131, and

B2 (the "LR1.7 bitS), per line, On a cache miss, the 80486 reads a 16-1100 line front main memory in a bus memory read burst, Draw a simplified diagram of the cache and show how the different fields of the address are interpreted.

*Soarcr: J*ALEX931

4,7 Consider a machine with a byte addressable main memory of 2' bytes and block size of S bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

a. !low is a 16-bit memory address divided into tag, line number, and byte number?
b. Into what line would bytes with each of the following addresses he stored'?

0001 0001 0001 1011
1100 0011 0011 0100
1101 0000 0001 1101
1010 1010 1010 1010

c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?
d. flow many total bytes of memory can be stored in the cache?
e. Why is the tag also stored in the cache

4.8 For its on-chip cache, the Intel 80486 uses a replacement algorithm referred to as pseudo least recently used. Associated with each of the 125 sets of four tines (labeled LO, 1.1.1,2, (3) are three bits Bit, B1, and B2. The replacement algorithm works as follows: When a line must be replaced, the cache will first determine whether the must recent use was front 1.0 and 1,1 or 12 and L3. Then the cache will determine which of the pair of blocks was least recently used and mark it for replacement. Figure 4,15 illustrates the logic.

a. Specify how the hits BO, BI, and B2 arc set and then describe in words how they are used in the replacement algorithm depicted in Figure. 4,15,
b. Show that the 80486 algorithm approximates a true algorithm. Hint: Omsider the case in whiCh the most recent order of usage is 1,0. L2. 13, Ll.
e. Demonstrate that a true UZI.: algorithm would require 6 hits per set.

4.9 A set associative cache has a block size of [our 16-bit words and a set size of 2. The cache can accommodate. a total of 4048 words. The main memory size that is cacheable is 64K X 32 hits, Design the cache structure and show how the processor's addresses are interpreted.

Source: [ALL 93]

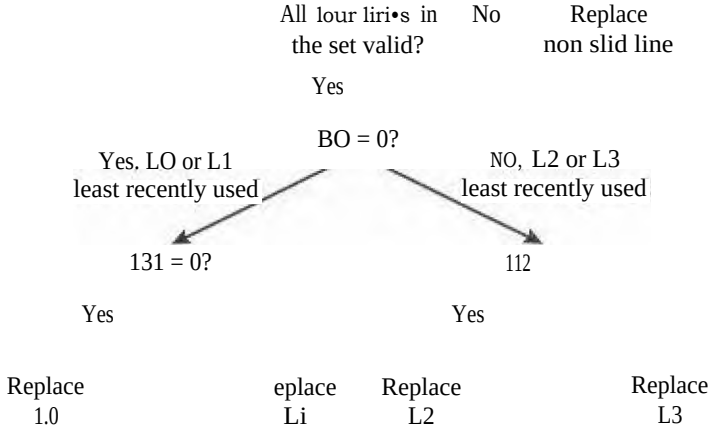


Figure 4.15 Intel 80456 On-Chip Cache Replacement Strategy

4.10 Consider a memory system that uses a 32-hit address to address at the byte level. plus a cache that uses a 64-byte line sin-.

a. Assume a direct mapped cache with a tag field in the address of 20 bits. Show the address format and determine the following parameters: number of addressable units, number of blocks in main memory, number of lines in cache, size of Lag.

h. Assume an associative cache. Show the address formal and determine the following parameters: number or addressable units, number of blocks in main memory, number of lines in cache, size of tag.

c. Assume a 4-way set associative cache with a lag field in the address of 9 hits. Show the address format and determine the following parameters; number of address• able units. number of blocks in main memory, number of lines in set. number of sets in cache. number of lines in cache. size of tag,

4.11 Describe a simple technique for implementing an LRli replacement algorithm in a four-way set associative cache.

4.12 Consider the following code:

```
for ii.    0; a.  < 20; L+-F)
    for (j - O; j <  :o
          =        *
```

a. Give one example of the spatial locality in the code,

b. Give one example of the temporal locality in the code,

4.13 Generalize Equations (4.1) and (42), in Appendix 4A, ill N'-level memory hierarchies.

4.14 A computer system contains a main memory of 32K 16-bit words. It also has a 4K-word cache divided into four-line sets with 64 words per line, Assume that the cache is initially empty. 'rite processor fetches words from location s fl, 1,2 ,,,,, 4351 in that order. It then repeats this fetch sequence nine more times. The cache. is 11) times faster than main memory. Estimate the improvement resulting from the. use of the cache. Assume an LA(policy for block replacement.

4.15 Consider n memory system with the following parameters:

$= 100$ as $= 0_01$ ebia

$T_{m} = 1,2(K)$ ns $C_{..} — 0.001$

a. What is the cost of I MO: of main memory?

IN What is the cost of I MByte of main memory using cache memory technology?

c. if the effective access time is 10% greater than the cache access time, what is the hit ratio *H?*

4.111 A computer has a cache, main memory. and a disk used for virtual memory. If a referenced word is in the cache, 20 ns are required to access it. If it is in main memory but not in the cache, 60 ns arc needed to load it into the cache, and then the reference is started again, If the word is not in main memory. 12 ms arc required to fetch the word front disk, followed by 60 ns to copy it to the cache. and then the reference- is started again. The cache hit ratio is 0.9 and the main memory hit ratio is 0.6_ What is the average time in its required to access a referenced word on this system?

APPENDIX 4A PERFORMANCE CHA_RACTE14§TICSP OF. TWO-LEVEL MEMORIES

in this chapter, reference is made to a cache that acts as a buffer between main memory and processor, creating a two-level internal memory. This two-level architecture provides improved performance over a comparable one-level memory, by exploiting a property known as locality, which is explored in this appendix.

Table 4.6 Characteristics of Two-Lc vc1 tarriorie.s

	Cache	Virtual Memory (Paging)	Disk Cache
'Typical access time ratios	40f1 (om-crtip cm:1w to main memory) 10.1 Ahep c **ache** to T(12i.11 111C.11107	14.00011 (main manory to disk)	Dapuoii (Inaiti rncrnory if.) disk)
NiErnory management System	truplermnicd by speci a l hardwom	CC)111h111a1L031 Ildrdwarc aged symluo scillwrirc	SVNILCM E(111WRTE
Typiwi Nock wire	4 u 126 bye s	().4 to 40% fn,..ics	64 to .111% bytils
Access of processor to irarowl level	Direct access	Indirect access	Indirect access

The main memory cache mechanism is part of the computer architecture. implemented in hardware and typically invisible to the operating system. There are two other instances of a two-level memory appro4ieh That also exploit lueah[y .and that are, at least parthi ly. implemented in the operating system: virtual memory and the disk cache (Table. 4.6). Virtual memory is explored in Chapter 8, disk cache is beyond the scope of this book but is examined in ISTAL01]. In this appendix, we look at some of the performance characteristics of tWO-level memories t hal are common to all throe approacho.7i.

Locary

f he basis fin- rho porlorninnea advantage or a two-level memory is a principle known as *locality of referene.-e* I)E 1,S I. This principle states that 111C1I101N references tend to cluster. Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references,

irrorn vin intuitive point of view, thEi principle of locality makes sense. Consider the following line of reasoning:

1. **Except** for hranch and call instructions, which constitute oitiv a small fraction of all program instructions, program execution is sequential. Hence. in most cases, the next instruction **to** be fetched immediately foliOW ', 1112 LISI instruction fetched.
2. ft is rare to have a **long** uninleiTupled sequence or procedure calls fonowed by the corresponding sequence of returns. Rather. a program remains confined to a rather narrow window of procedure-invocation depth. Thus. over a short period of time references to instructions lend to be localized to a kw **procciUre**
3. Most **iterative constructs consist of a relatively small** number of instructions repeated many times. For the duration of the iteration, computation is therefore COT1 lined to a **mall** contiguous portion of a, program.
4. In many programs, much of the computation involves processing data structures, such as arrays or sequences of records. In many cases, successive references to thc.e (Wu structures will be 10 closely located **claw items.**

Table 4.7 Relative Dynamic Frequency cif high-Level Language Operations

Study	[FlUcK831	[KAU" 1711	[PATTS21		[TANEN
Language	Pascal	FORTRAN	Pascal	C	SAL
Workload	Scientific	Student	System	System	System
Assign	74	67	45	38	42
Loop	4	3	5	3	4
Call	1	3	I5	12	12
IF	21)	II	29	43	36
COTO	2	9		3	
Other		7	6	1	b

This line of reasoning has been confirmed in many studies. With reference to point 1, a variety of studies have analyzed the behavior of high-level language programs_ 'T'able 4.7 includes key results, measuring the appearance of various statement types during execution, from the following studies. The earliest study of programming language behavior, performed by Knuth (KNUT711. examined a collection of FORTRAN programs used as student exercises, Tanenbaum [TANE78L published measurements collected from over 300 procedures used in operating-system programs and written in a language that supports structured programming (SAL). Patterson and Sequent. IPATTS2a] analyzed a set of measurements taken from compilers **and** programs for typesetting, computer-aided design (CAD), sorting, and file comparison. .1'he programming languages C and Pascal were studied. Huck [HUCK83] analyzed four programs intended to represent a mix of general-purpose scientific computing, including fast Fourier transform and the integration of systems of differential equations. There is good agreement in the results of this mixture of languages and applications that branching and call instructions represent onl!,' a fraction of statements executed during the lifetime of a program_ Thus, these studies confirm assertion I.

With respect to assertion 2. studies reported in [PATT85a I provide confirmation_ This is illustrated in Figure 4_16, which shows call-return behavior. Each call is represented by the line moving down and to the right, and each return by the line moving up and to the right. In the figure, a *window* with depth equal to 5 is defined, Only a sequence of calls and returns with a net movement of 6 in either direction causes the window to move. As can be seen. the executing program can remain within a stationary window for long periods of time. A study by the same analysts of Cr and Pascal programs showed that a window of depth 8 will need to shift only on less than 1% of the calls or returns [TAMI.83].

The principle of locality of reference continues to be validated in more recent studies. For example, Figure 4.17 **illustrates** the results **of a study of Web** page access patterns at a single site.

A distinction is made in the literature between spatial locality and temporal locality. **Spatial locality refers to** the tendency of execution lo involve a number of memory locations that arc clustered. This reflects the tendency of a processor to access instructions sequentially_ Spatial location also reflects the tendency of a program to access data locations sequentially, such a.s when processing a table of daiie **Temporal locality** refers to the tendency for a processor to access memory locations

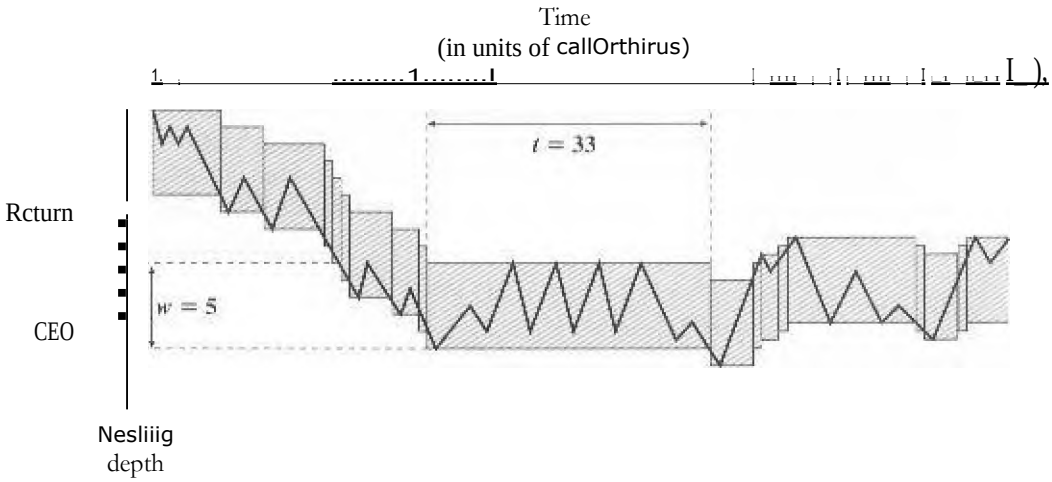


Figure 4.11; eAfirripl Call-R"(41111 11.1.:111'cLviOr Phpgram

that have been used recently. For example, when an iteration loop is executed, the processor executes the same set of instructions repeatedly.

Traditionally, temporal locality is exploited by keeping recently used instruction and data i4 Lies in cache incroory Fula by eNploiling a c;ichc. Iticlarch v. Spatial locality is generally exploited by using EarKer cache. blocks and by incorporating prefetching mechanisms (fetching items of anticipated use) into the eacR. control

Recently, Ihcre has been considerable res eareh on refining these techniques to achieve greater performance, but the basic strategies remain the same.

Operation of Two-Level Memory

The locality property can be exploited in the formation. of FI lmo-Few] rtioniory. The upper-level memory (Ml) is smaller, faster, and more expensive (per bit) than the lower-level mumory (M2). Pvil is used t temporary store for part *of* the contents

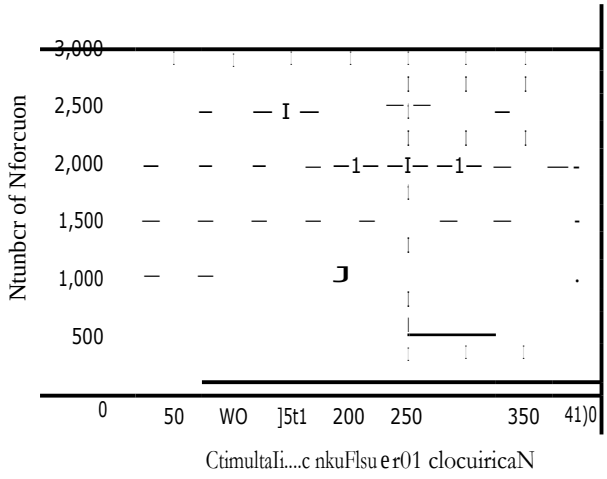


Figure 4.11 Locality' n 1-1eferenee for Pages IBAEN971

of the Larger M2. When a memory reference is made, an attempt is made to access the item in Ml. If this succeeds, then a quick access is made. If not, then a block of memory locations is copied from M2 to MI and the access then lakes place via Ml. Because of locality, once a block is brought into Ml, there should be a number of accesses to locations in that block. resulting in fast overall service.

To express the average time to access an item, we must consider not only the speeds of the two levels of memory, but also the probability that a given reference can he found in Mi. We have

$$\begin{aligned} & T_1 - (1 - H) \times (T_1 + T_2) \\ = T_1 \quad & (1 - H) \times T_2 \end{aligned} \tag{4.1}$$

where

T_1 — average (system) access time
T_1 = access time of MI (e.g., cache, disk cache)
T_2 = access time of M2 (e,g., main memory, disk)
H = hit ratio (fraction of time reference is found in M1)

Figure 4,2 shows average access time as a function of hit ratio. As can be seen, for a high percentage of hits, the average total access time is much closer to that of M•I than M2.

Performance

Let us look at some of the parameters relevant to an assessment of a vivo-level memory mechanism. First consider cost. We have

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \tag{4.2}$$

where

C_s average cost per bit for the combined two-level memory
C_1 = average cost per bit of upper-level memory M1
C_2 average cost per bit of lower-level memory M2
S_1 = size of MI
S_2 = size of M2

We would like $C_s \approx C_2$. Given that $C_1 >> C_2$. this requires $S_1 \ll S_2$, Figure 4.18 shows the relationship.

Next. consider access time. For a two-level memory to provide a significant performance improvement, we need to have T_1 approximately equal to T_1 (T_1 --- Given that T_2 is much less than T_1 ($T_2 \ll T_1$), a hit ratio of close to 1 is needed.

So we would like M1 to be small to hold down cost, and large to improve the hit ratio and therefore the performance. Is there a size of Mi that satisfies both requirements to a reasonable extent'? We can answer this question with a series of subquestions:

- What value of hit ratio is needed so that T_1
- What size of MI will assure the needed hit ratio?
- Does this size satisfy the cost requirement?

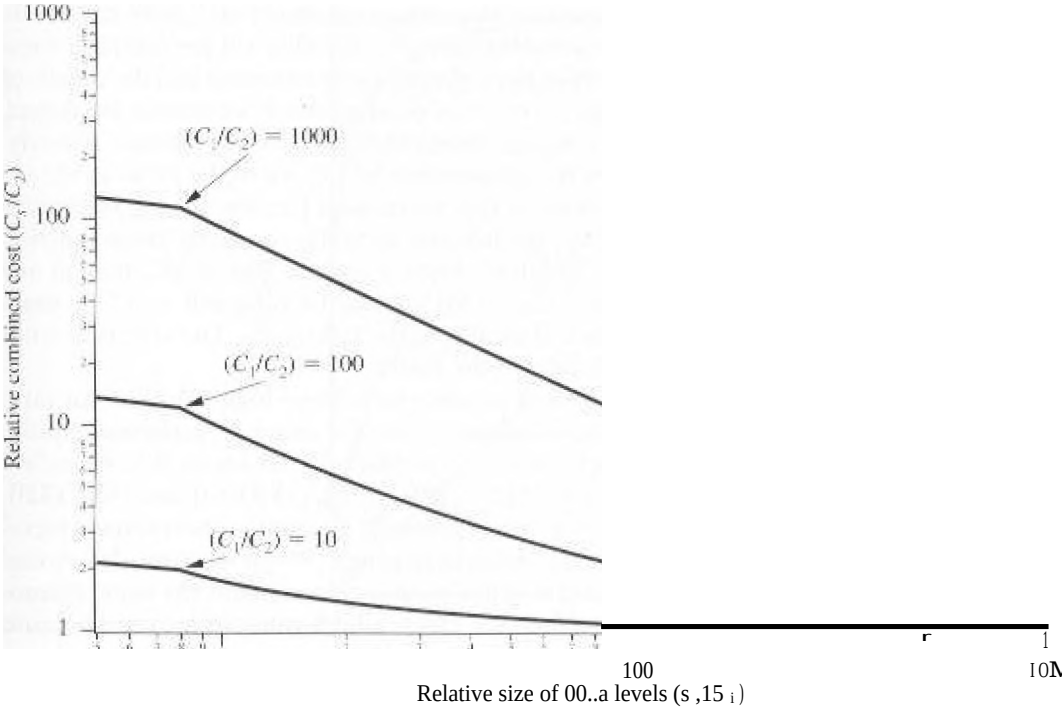


Figure 4.18 Relationship of Average Memory Cost to Relative Minatory Size l'or a Twii-LevelMemory

To get at this, consider the quantity 1', r $T_{,.}$ which is referred to as the *occesA efficiency,* It is a measure of how close average access time *(T)* is to MI access time ($T_{]}$). From Equation (4.1).

$$\frac{T_{]}}{T} \qquad (4.3)$$

In Figure 4. [9, we plot $T_{]}$ *1T,* as a function of the hit ratio *H.* with the quantity 1 I $T_{]}$ as a parameter. Typically, on-chip cache access time is about 25 to 50 times faster than main memory access time (i.e., 1 ' *IT,* is 5 to 10), off-chip cache access time is about 5 or 15 times faster than main memory access time (i.e., 1 ,/7 ' is 5 to 15).' and main memory access lime is about 1000 times faster than disk access time **(T2** 17] = **NM).** Thus, a hit ratio in the range of near 0.9 would seem to be needed to satisfy the performance requirement_

'For example. at I he time of [lib writing, for the Pentium 4, on-chip cache acces,i time is 1 ns fur data cache. 2 ns for instruction cache, and 3,5 its For L2 cache.; main memory access time is 3U ns. For the liatli.11M. **ch** ip cache a c cess **time** is 2 us tar LI cache and **6** ns Fro' 1.2 cad112.. of r-chip access time l'a.wr 1-3 cache is 21 ns: main memory *access* ti me is 5II ns-

Vtre can now phrase the question'about relative memory size more exactly. is a hit ratio of. say, 0,8 or better reasonable for $S_i << S_2$? This will depend on a number of factors, including the nature of the software being executed and the details of the design of the two-level memory. The main determinant is, of course. the degree of locality. Figure 420 suggests the effect thEli locality has on the hit ratio. Clearly. if MI is the same size as M2. I hen the hit ratio will he 1.0: All of the items in M2 are always stored also in **Ml.** Now suppose that there is no locality; that is, references are completely random. In that ease the hit ratio should be a strictly linear function of the relative memory size. For example, if M1 is half the size of M2, then at any time half of the items from M2 are also in Iv11 and the hit ratio will he 0..5. In practice, however, there is some degree of locality in the references, 'file effects of moderate and strong locality are indicated in the figure.

So if there is strong locality, it is possible to achieve high values of hit ratio even with relatively small upper-level memory size. For example, numerous studies have shown that rather small cache sizes will yield a hit ratio above 0.75 *regardless of the size of ;wait? inaruYry* [AGAR89], [PRZYKSI, [STRE83], and iSlylIT821). A cache in the range of lK to 128K words is generally adequate, whereas main memory is now typically in the multiple-mcgabyle range. When we consider virtual memory and disk cache.. we will cite other studies that confirm the same phenomenon, namely that a relatively small Ml yields a high value of hit raiio because of locality.

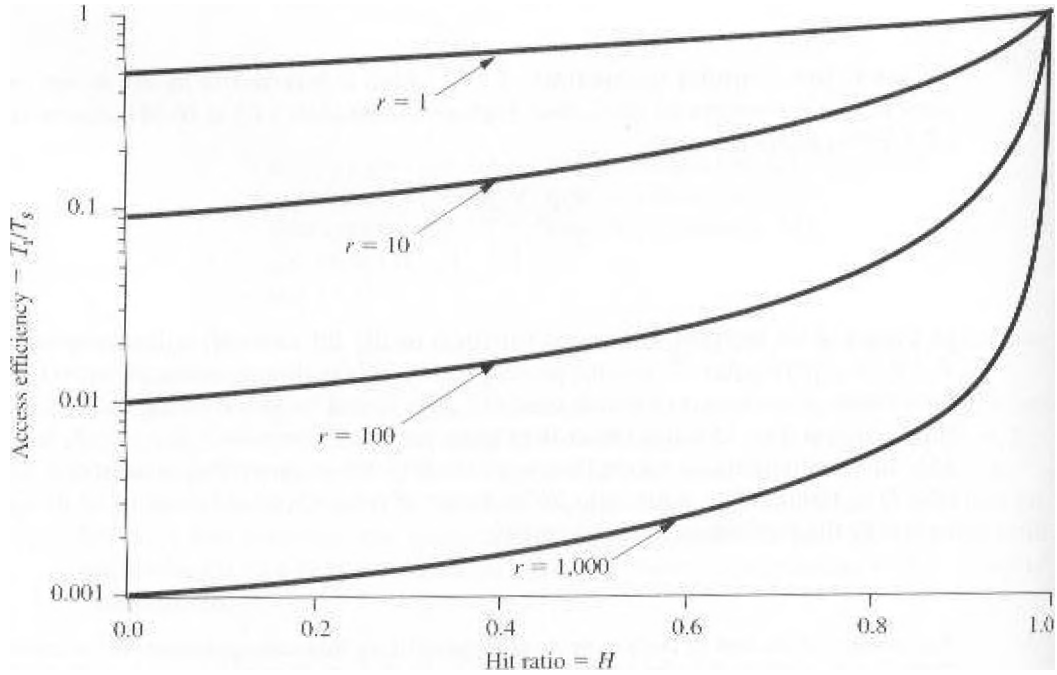


Figure 4.19 Access Efficiency as a Function of hit Rath.] v = T_1/T_2

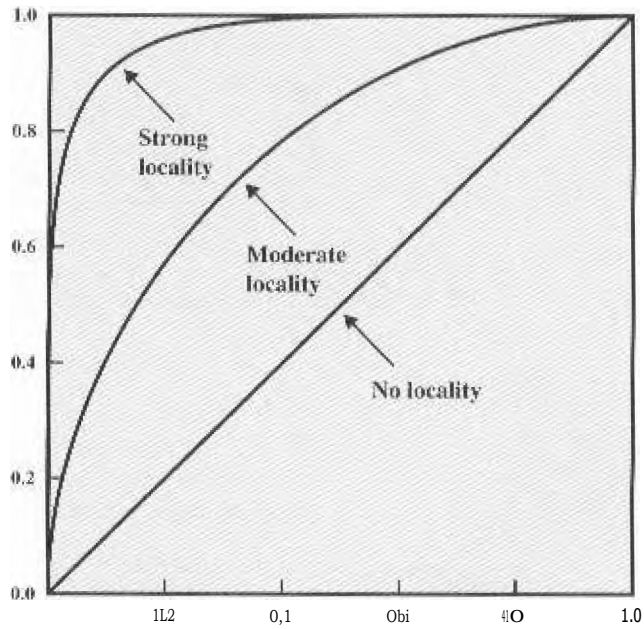


Figure **4.20** Hit Ratio as a Function of Relative Memory Size

This brings us, to the last question listed earlier: Does the relative size of the two memories satisfy the cost requirement? The answer is clearly yes. If we need only a relatively small upper-level memory to achieve good performance, then the average cost per hit of the two levels of memory will approach that of the cheaper lower-level memory.

Please note that with L2 cache, or even L2 and L3 caches, involved, analysis is much more complex_ See RTIR991 and [HAND98I for discussions.

CHAPTER 5

INTERNAL MEMORY

5.1 Semiconductor Main Memory

Organization
DRAM and SRAM
Type.!.. of NOM
Chip Logic
Chip Packaain.
Moduic. Organiz4ition.

5.2 Error Correction

5.3 Advanced DRAM Organization

SYJ IA) D °US I) R
R mhus DRAM ...
CiLac AVI

5.4 Recommended Reading and. Web Sitt8

55 Key Terms, Review Questions, um' Problems

Key ˙1**erms**
Revictw QUQ. stions
PtobNJW .

KEY POINTS

- The two basic forms of semiconductor random-access memory are dynamic RAM (DRAM) and static RAM (SRAM). SRAM is faster, more expensive, and less dense than DRAM, and is used for cache memory. DRAM is used for main metnor!,',
- Error correction techniques are commonly used in memory systems. These involve addino, redundant bits that are a (unction of the data bits to 1:01'311 are error-correcting code, If a bit error occurs, the code will detect and, usually, correct the error.
- To compensate for the relatively slow speed of DRAM. a number of advanced DRAM organizations have been introduced. The two most common are synchronous DRAM and Ronifius DRAM. Both of these involve using the system clock to provide. for the transfer or blocks of data.

This chapter begins with a survey of semiconductor main memory subsystems, including ROM, DRAM. and SRAM memories. Then we look at error control techniques used to enhance memory reliability. Following this, we look at more advanced DRAM architectures.

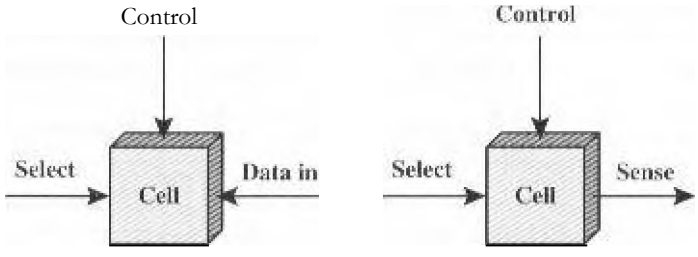
5.1 SEMICONDUCTOR MAIN MEMORY

In earlier compmers, the most common form of random-access enrage for computer main memory employed an array of doughnut-shaped ferromagnetic loops referred to as *cores.* Hence, main memory was often referred to as *core,* a term that persists to this day, The advent of, and advantages of, microelectronics has long since vanquished the magnetic core memory. Today, the use of semiconductor chips for main memory is almost universal. Key aspects of this technology are explored in this section,

Organization

The basic element of a semiconductor memory is the memory cell, Although a variety of electronic technologies arc used. all semiconductor memory cells share certain properties:

- They exhibit two stable (or semistablc) states, which can be used to represent binary l and O.
- They are capable of being written into (at least once), to set the state.
- They are capable or being read to sense the state.



(a) Write .11)i. Head

Figure 5.1 MOTH LPry r1.1 ll OpCi•kl i4J11

Figure 5.1 depicts the operation of a memory **Cc.11..Mosi** commonly, the cell has three functional terminals capabie of carrying an electrical signal. The select terminal, as the name suggests, selects a memory cell for a read or write operation. The control tevniinal indicates read or write. Far writing, the other terminal provides an electrica] signal that sets the state of the cell to 1 or 0. For reading. Out terminal is used for output of the cell's state. The. details of the internal organization, functi ming, and timing of the memory cell depend on the specific integrated circuit technology used and are beyond the scope 0.1' this book,4.:xcepi for **41 blia gli Tnnary.1 or** our purpa gcs. we will take it. is given t Ht individual cells can be selectci for reading and writing operations.

DRAM and SRAM

All of the memory types that we will explore in this chapter arc. random access. That is, individual words of memory are directly accessed through wired-in addressing logic,

Table 5.1 lists die major types of semiconductor memory. The most common is referred to as *random-access* ***Merflary*** (RAM). This is, of course. a misuse of the t**erm,** because all of the types listed in the table are random access. One distinguishing characteristic of RAM is that it is possible both to read data from the memory and to write new data into the memory easily and rapidly. Both the reading and writing are accomplished through the of electrical signals.

The other distinguishing characteristic of RAM is that it is volatile. A RAM must be provided with **a C.Dostni** power supply. 11 the power is interrupted, then the data are lost Thus, RAM can he used only as temporary storage. The two traditional forms of RAM used in computers are DRAM and SRAM,

Dynamic **RAM**

RAM technology is divided into two technoloyics: dynamic and static. A dynamic RAM (DRAM) is made with *cells* that store data as charge on capacitors. The presence or absence of charge on a capacitor is interpreted as a binary l or 6. Because capacitors have natural tendency to dimThargc, dynamic RAMS require periodic charge refreshing to maintain data storage. The term *elynamic* refers to this tendency of the stored charge to leak away. even with power continuously applied.

Table 5.1 Memory Types

Memory Type	iiry	Erasure	Write Media also'	Volt Iilit,
Rorsdom-access memory (RAM)	Read.write memory	FiccRrica I ty. h} to Ityr	Eloc(rieally	Nucsoola tilk
Rt'd d-wanly rriertkuty (ROM)	Remd-only critmor:y	Not possible	Masks	
Programmable ROM (PROM)			Electrically	
Erasable PROM {EPROM)	Readtncaly memory	chip level		
Elecirically Erasable FROM (EEPRDM}		Electrically, byte lcvd		
Flash memory		Elcpctri CH I I y. h1{14:k C'c1		

Figure 5,2a is a typical DRAM structure. for an indit,idual cell that stores one bit. The address line is activai ed when the hit value. from this cell is to be read or written. The transistor acts as 3 sgitch that is closed (allowing current to flow) if a voltage is ,ripplicd to the address line and open (no current fiows) if no voltage is present on the address line.

For the write operation, a vollagi2 signal is applied to the bit line; a high voltage represents 1, and a iow voltage represents 0, A signal is then applied to the addresf, line, allowing a charge to be transferred to the capacitor.

For the read operation, when the address line is selected. the transistor turns on and the charge stored cm the capacitor is fed out onto a hit line and to a sense amplifier. The sense amplifier compares the capacitor voltage to a reference value and determines if the cell contains a logic | a logic 0. The read out from **the** cell discharges the capacitor, which musk he restored to complete the operation.

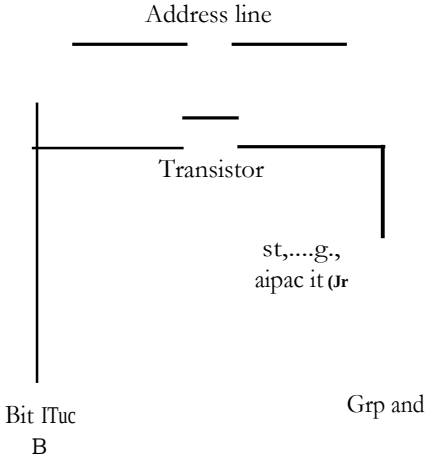
Although the DRAM cell is used to store a single bit (0 or 1), it is essentially an analog device. 'The capacitor can store any charge value within a ranee: a threshold value determines whether ale charge is interpreted as 1 or O.

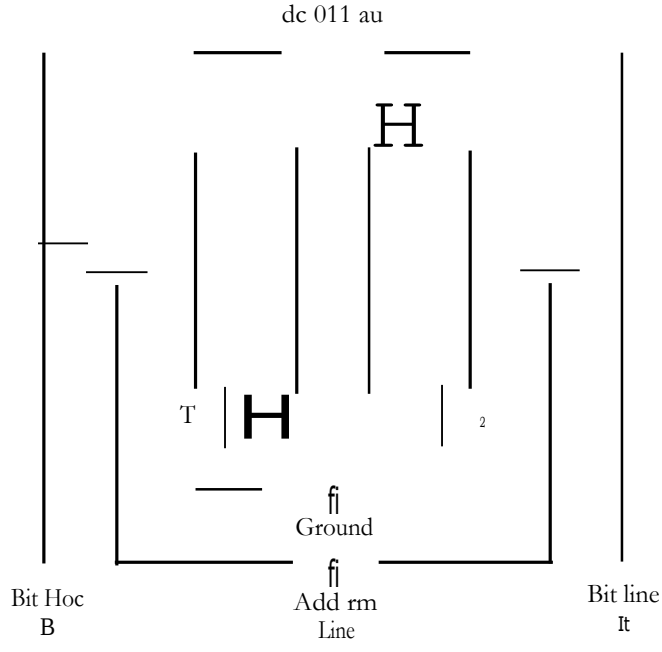
Static KAM

In contrast, a static RAM (SRAM) is a digital device, using the kaiTlle logic elements used in the processor. In a SRAM, binary values are siortx1 using traditional flip-flop logic-gate configurations (see Appendix A for a description of flip-flops). A static RAM will hold its data as long as power is supplied to it.

Figure 5.2b is a typical SRAM structure for an individual cell_ Four transistors T), 'U,. T_4) Are (TOSS' CA.}[Merted in an arrangement alai produces a stable logical state. In logic state 1. point C, is high and point C, i ioxy in this state, T, and ariz oft and T, and $T_{|}$ arc on.' **In** logic state 0. point $C_{|}$ is low and point C, is high; in

'The circles at the head ot T-, and 7^{4}_{1} indicate tii grkal ncgatiCni.





[a? Dynamic RAM IDRiVel) cell

0o) Static. Ft A EA (SRA1[0],1) cell

Figure 5.2 Typical Memory Cell Structures

this stale, **T.. and** '1:1 are on and T.-. and T_3 are off. Both states are stable as long as the direct current (de) voltage is applied. Unlike the DRAM, no refresh is needed to retain data.

As in the DRAM, the address line is used to open or close a switch. The address line controls two transistors (T, and TO. When a signal is applied to this line, the two transistors are switch on, allowing a read or write operation. For a write operation, the desired hit value is applied to line B, while its complement is applied to line B. This forces the four transislorS (T_1. $1'_2$, *13, .P_L)* into the proper state. For a read operation, the bit value is read from line **B.**

SRAM versus DRAM

Both static and d!.·natnic RAMs are volatile that is, power must he continuously supplied to the memory to preserve the bit 'raities, A dynamic memory cell is simpler and smaller than a stalie memory eel[. 'Thus, a DRAvt is more dense (smaller cells = more Cells per unit area) and less expensive than a corresponding SRAM. On the other hand, a DRAM requires the supporting refresh circuitry. For Larger memories. the fixed cost of the refresh circuitry is more than compensated for by the smEilier variable cost 0r DRAM cells. Thus, DRAMs tend to be favored [or Large memory requirements. A final point is **that** SRAMs are generally somewhat faster than DRAMs. Because of these relative characteristics, SRAM is used for cache memory (both on and off chip). and DRAM is used for **main. memory.**

Types of ROM

As the name suggests, a **read-only memory (ROM)** contains a permanent pattern of data that cannot be changed. A ROM is nonvolatile: that is, no power source is required to maintain the hit values in memory. While it is possible to read a ROM, it is not possible to write new data into it, An important application of ROMs is microprogramming, discussed in Part Four. 01 her potenliid applications **include**

- **Libniry** subroutines for frequently wanted functions
- System programs
- Function tables

For a modest-sized requirement, the advantage of ROM is that the data or program is permanently in main memory and need never be. loaded from a secondary storage device.

A ROM is created like any cal her inlCgraled circuit chip, with the data actually wired into the chip as part of the fabrication process. This presents two **problems:**

- **The data insertion** step includes a relatively large fixed cost, whether one or thousands of copies of a particular **ROM** arc rAticated_
  ']'here is no room **for** error. **If** one hit is wrong, the whole batch of ROMs must be thrown out.

When only a **smutl number of ROMs with a particular memory content** is needed. a Less expensive alternative is the **programuuable ROM (PROM),** i.ike the ROM, the PROM is nonvolatile and may be written into un(v once. For the PROM, the writing process is performed electrically and may be performed by a supplier or

c,:ustorner at a time Iaier than the original chip fabrication. Special equipment is required for the writing or "programming" process. PROMs provide flexibility and convenience. The ROM remains attractive for high-volume production runs.

Another variation on read-only memory is the read-mostly memory, which is UserLd COT applications in which read operations are far more frequent than write operations but for which nonvolatile storage is required. There are three common forms of read-mostly memory: EPROM, [EPROM, and flash memory.

optically crumble programmable read-only memory (EPROM) is read and written electrically, as with PROM. However, before a write operation, all the storage cells must be erased to the same initial state by exposure of the packaged chip to ultraviolet radiation. Erasure is performed by shining an intense ultraviolet light through a window that is designed into the. memory chip. This erasure process can be performed repeatedly; each erasure can take as much as 20 minutes to perform. Thus, the EPROM can be altered multiple times and, like the ROM and FROM, holds its data virtually indefinitely. For comparable amounts of storage. the EPROM is more evensive than PROM. Elul. it has the advantage of the multiple update. capability.

A more attractive form of read-mostly memory ix elOctrically erasable programmable read-only memory (EKPROM). This is a read-roost]!, ' memory that can be written into at any time without erasing prior contents: only the byte or bytes addressed are updated. The write operation takes considerably longer than the read operation. on the order of several hundred microseconds per byte. The LLPROM combines the advantage of nonvolatility with the flexibility of being updatable in place, using ordinary bus control, address. and data lines. EEPROM is More expensive than EPROM and also is less dense, supporting fewer hits per chip.

Another Form of semiconductor memory is flash memory named because of the speed with which it can he reprogrammed). First introduced in the mid-19f10s, flash memory is intermediate between EPROM and EEPROM *in* both cost and functionality. Like EEPROM, flash memory uses an electrical erasing technology. An entire flash memory can be erased in one Or a few seconds, which is much faster than EPROM. In addition, it is possible to 0122:.y iLLS1 blocks of memory rather than an entire chip. Flash memory gets its name because the microchip is organi t.cd so that a section of memory cells are erased in a single action or 'llash." I LowcYcr, flash rricniory does not provide byte-level erasure. Like EPROM, flash memory uses only one transistor per bit. and so achieves the high density (compared with EEPROM) of EPROM.

Chip Logic

As with other intet..rrared circuit products , semiconductor memory comes in packaged chips (Figure 2.7). Each chip contains an array of memory cells,

In the memory hierarchy as a whole, we saw dial there are trade-offs among speed, capacity, and cost. These trade-offs also exist when we consider the organization of memory cells and functional logic on a chip. For semiconductor memories, one of the key design issues is the number of bits Or data that may he readiwritte.n at a time. At one extreme is an organization in which the physical arrangement of cells in the array is the same as the logical arrangement as perceived by I he processor) of words in memory. The array is organized into W words of bits each. For

example, a 16-Mbit chip could be organized as 1M 16-bit words. At the other extreme is the so-called one-bit-per-chip organization. in which data is readiwritten one hit at a time, We will illustrate memory chip organization with a DRAM: ROM organization is similar. though simpler.

Figure 3.3 shows a typical organization of a 16-Mbit DRAM. In this case, 4 bits are read or written at a time. Logically, the memory array is organized as four square arrays of 2()48 by 2048 elements. Various physical arrangements are possible, In any case, the elements of the array are connected by both horizontal (row) and vertical (column) lines. Each horizontal line connects to the Select terminal ()leach cell in its row; each vertical line connects to the Data-ln/Sense terminal of each cell in its column.

Address lines supply the address of the word to be selected, A total of log, W lines are needed. In our example, 11 address lines are needed to select one of 2048 rows. These 11 lines are fed into a row decoder. which has II lines of input and 2045 lines for output. The logic of the decoder activates a single one of the 2048 outputs depending on the bit pattern on the 11 input lines ($2^{11} = 2048$).

An additional I I address lines select one of 2048 columns of 4 bits per column. Four data lines are used for the input and output of 4 hits to and from a data buffer_ On input (write), the bit driver of each bit line is activated for a 1 or 0 according to the value of the corresponding data line. On output (read), the value of each hit line is passed through a sense amplifier and presented to the data lines. The row line selects which row of cells is used for reading or writing.

Because only 4 bits are read/written to this DRAM, there must be multiple DRAMs connected to the memory controller to readlwrite a word of data to the bus.

Note that there are only 11 address lines (AO—A10), half the number you would expect for a 2048 x 2048 array. This is done to save on the number of pins. The 22 required address lines are passed through select logic external to the chip and multiplexed onto the 11 address lines. First, 11 address signals are passed to the chip to define the row address of the array, and then the other ii addreSs signals arc presented for the column address_ 'Mese signals are accompanied by row address select (RAS) and column address select (CAS) signals to provide timing to the chip.

The write enable (WE) and output enable (OE) pins determine whether write or read operation is performed. Two other pins_ not shown in Figure 5.3, arc ground (Vss) and a voltage source (Vcc).

As an aside, multiplexed addressing plus the use of square arrays result in a quadrupling of memory size with each new generation of memory chips_ One more pin devoted to addressing doubles the number of rows and columns, and so the size of the chip memory grows by a factor of 4.

Figure 5.3 also indicates the inclusion of refresh circuitry, All DRAMs require a refresh operation. A simple technique for refreshing is, in effect. to disable the I) RAM chip while all data cells are refreshed, The refresh counter steps through all of the row values. For each row. the output lines from the refresh counter are supplied to the row decoder and the RAS line is activated. The data are read out and written back into the same location, This causes each cell in the row to be refreshed,

Chip Packaging

As was mentioned in Chapter 2, an integrated circuit is mounted on a package that contains pins for connection to the outside world.

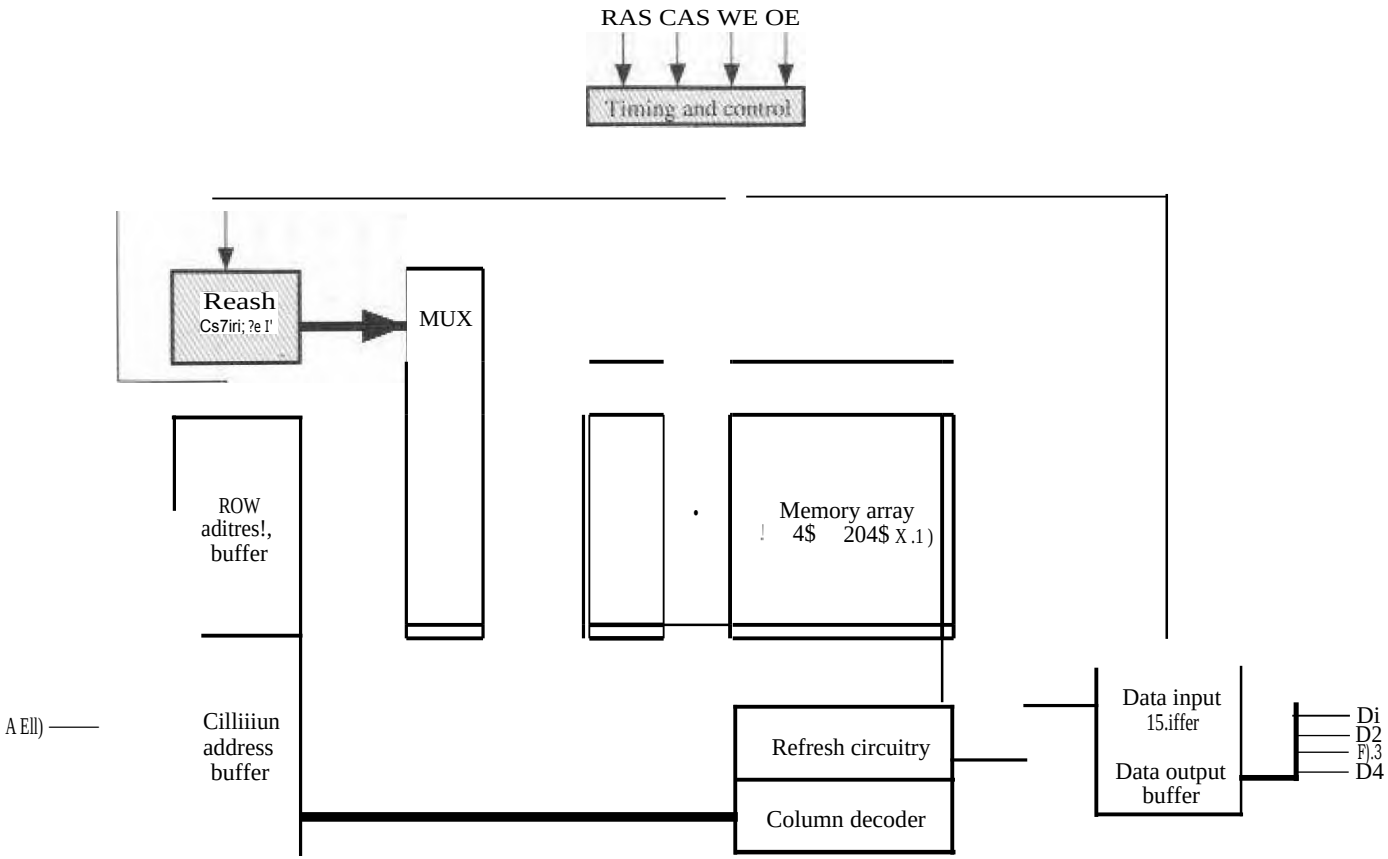
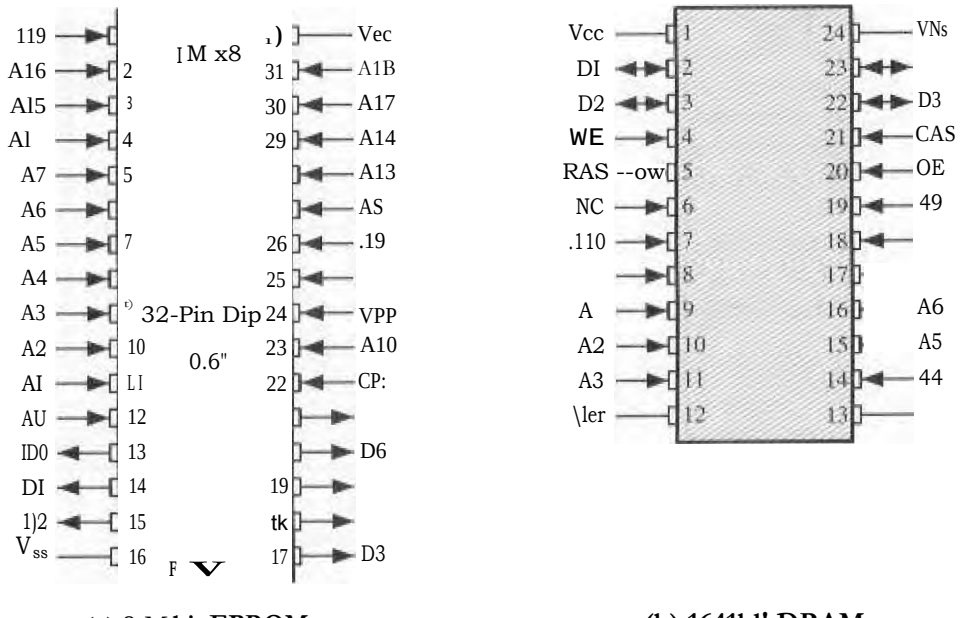


Figure 5.3 Typical 16 Mcgabit DR:• M {4M 41

Figure 5.4a shows an example EPROM package, which is an 8-Mbit chip organi4ed as 1M x 8. In this case, the organization is treated as a one-word-per-chip package. The package includes 32 pins. which is one of the standard chip package sizes_ The pins support the following signal lines:

- The address of the word being accessed. For I M words, a total of 20 ($2^{25} = 1M$) pins are needed (AU--A 19).
- The data to he read out, consisting of 8 lines (DO-D7).
- The power supply to the chip (Nice).
- A ground pin (Vss).
- A chip enable (CE) pin. Because there may he more than one memory chip. each of which is connected to the same address bus. the CE pin is used to indicate whether or not the address is valid for this chip. The CE pin is activated by logic connected to the higher-order bits of the address bus (i.e., address bits above A19). The use of this signal is illmtrated presently_
- A program voltage (Vpp) that is supplied during programming (write operations).

A typica] DRAM pin configuration is shown in Figure 5.4b. for a 16-Mhit chip organized as 4M X 4. There are several differences from a ROM chip- Because a RAM can he updated, the data pins arc inputioutput. The write enable (WE) and output enable (011) pins indicate whether this is a write or read operation. Because the DRAM is accessed by row and column, and the address is multiplexed, only 1 "address pins are needed to specify the 4M row/column combinations (2' I $\times 2^{11}$



(a) 8-M hit EPROM **(b) 1641bl' DRAM**

Figure 5.4 Typical Mowry Pack agc Pins and Signals

2^{22} = 4M). The functions of the row 4iddress select (RAS) and column address select (CAS) pins were discussed previoustv. Finally, the no connect (NC) pin is provided so that there arc. on CVCri number of pins.

Mo dul e Organization

If 41 RAM chip contains Drib/ 1 hit per word, then cicAv we will need at least al R.117-E3er of chips equal to the number of bits per word, As an example. Figure 5.5 shows how a memory module consisling of 256K 8-bit words could he orgy tiled, For 256K words, an [8-bit tic dress is needed and is supplied to the [nodule From sonic external source (e.g._ the address lines of zi bus to which the module is attached), The

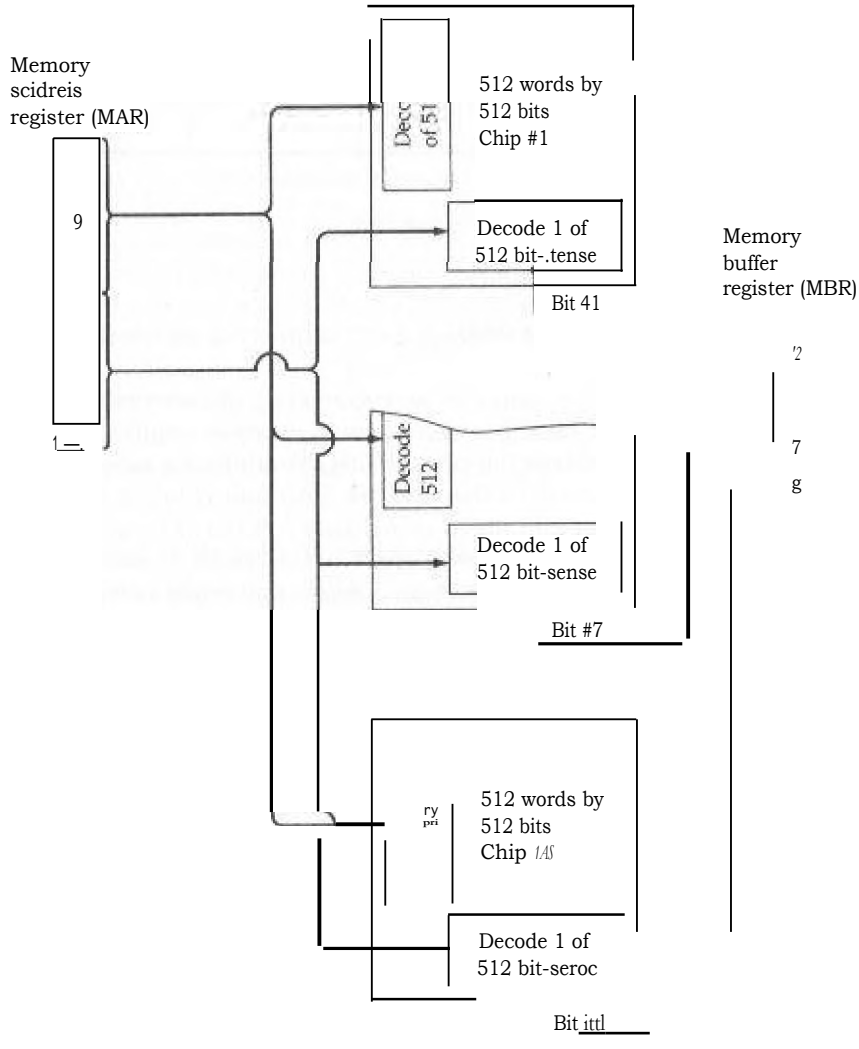


Figure 5.5 256-Kbyte Isrle nu] ry Organ tin n

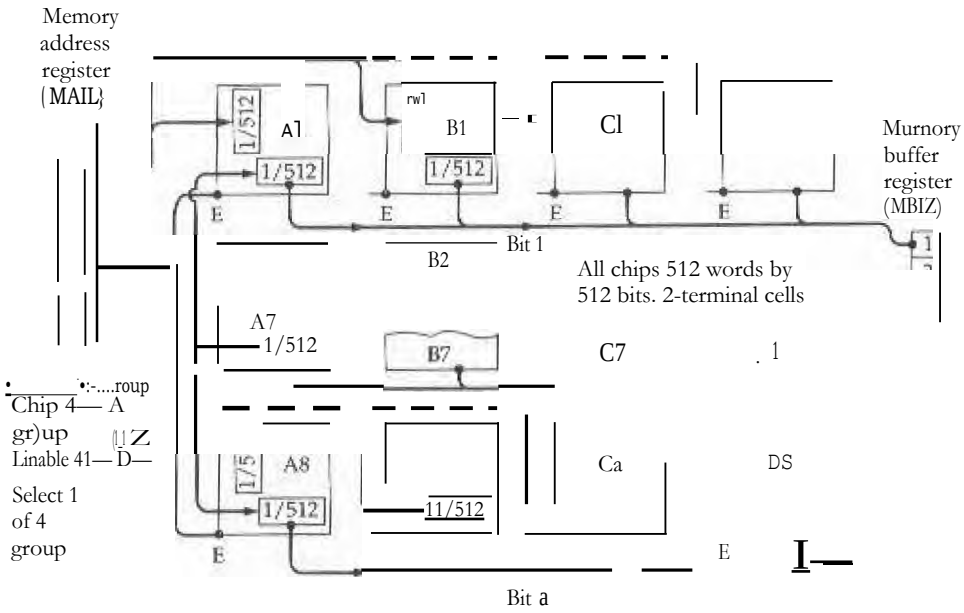


Figure. 5.6 Memory Organization

address is presented to 8 256K Y, 1-hit chips, each of which provides the input! output of I bit.

This organization works as long as ihe size of memory equals the number of bits per chip. In l he case in which larger memory is required, an array of chips is net4r.led. Figure 5.6 shows the possible organization of a memory consisting of llvl word by 8 bits per word. in this case, we have four columns of chips, each column containing 256K words arranged as in Figure 5.5, For '1 NI word, 20 address lines are needed. The ig least si gnificant bits are routed to ail 32 modules, The high-order 2 bits are input to a group select logic module thin sends a chip enable signal to one of the four **CIIjuntris** of modules,

5.2 ERROR ColtRECTION!·

A semiconductor memory system is subject to errors. These can be categorized as hard failures and soft C1T0E-r. A bard failure is a permauenl physical defect so that the memory cell or cells affected cannot reliably store data, but become stuck at 0 or 1 or switch erraticatt!,' between 0 and 1, Hard errors can be caused by harsh environmental abuse, manufacturing defects, and wear. A soil error is a random, nondestructive event that alters the contents 0r one or more memory cells, without damaging the memory. Soft errors can he caused by power supply problems or alpha particles. These particles result from radioactive decay and are distressingly common because radioactive nuclei are found in small quantities in nearly all materials,

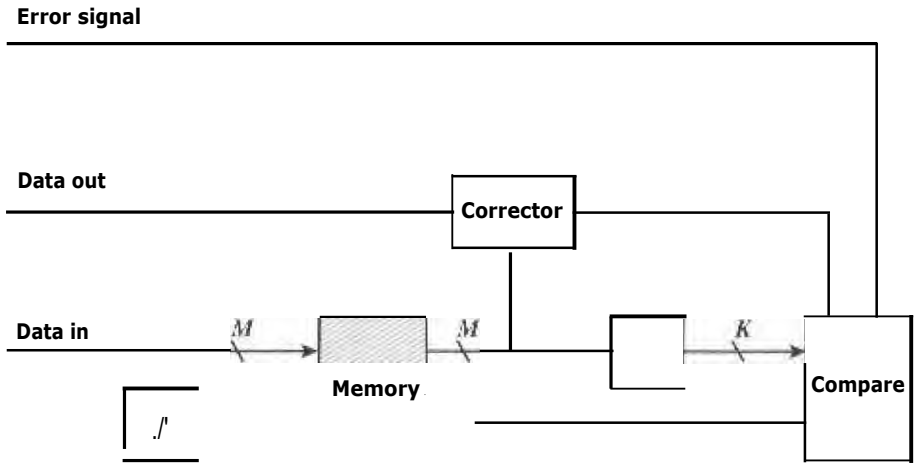


Figure 5.7 Error-Correcting C..ode Function

Both hard and soft errors are clearly undesirable, and most modern main memory systems include logic for both detecting and correcting **error:.**

Figure. 5.7 illustrates in general terms how the process is carried out. When data are to be read into memory, a calculation, depicted as a function *f,* is performed **on** the data to produce a code. Both the code and the data are stored. Thus, if an M-bit word of data is to be stored, and the code is of length *K* hits, then the actual size of the stored word is M + *K* bits.

When the previously stored word is read out. the code is used to detect and possibly correct errors. A new set of *K* code bits is generated from the M data bits and compared with the fetched code bits. The comparison yields one of three results:

- No errors are detected. The fetched data hits are sent out.
- An error is detected, and it is possible to correct the error. The data bits plus error correction hits are fed into a corrector, which produces a corrected set of :14 hits to be sent out.
- An error is detected, but it is not possible to correct it. This condition is reported_

Codes that operate in this fashion are referred to as *error·correcting codes.* A code is characterized by the number of hit errors in a word that it can correct and detect,

The simplest of the error-correcting codes is the *Hamming code* devised by Richard Hamming at Bell Laboratories. Figure 5,S uses Venn diagrams to illustrate the use of this code on 4-hit words *(214* = 4). With three intersecting circles, there are seven compartments, We assign the 4 data bits to the inner compartments (Figure 5.8a). The remaining compartments are filled with what are called *parity hits.* Each parity bit is chosen so that the total number of Is in its circle is even (Figure 5.8b), Thus, because circle A includes three data is, the parit!,' hit in that circle is set to I . Now, if an error changes one of the data bits (Figure 5,8c). it is easily found,

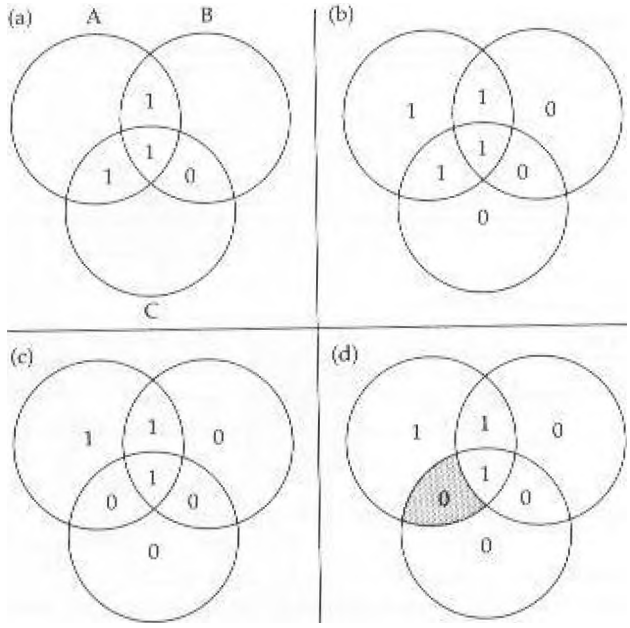


Figure 5.8 Hamming Error-C orrecting Coat

By checking the parity bits, discrepancies are found in circle A and circle C but not in **circle B.** Only one of the seven compartments is in A and C but not B. The error can therefore be corrected by changing that bit.

To clarify the concepts involved, we will develop a code that can detect and correct single-bit errors in 8-bit words,

To start, Let us determine how long the code must he Referring to Figure 5.7, the comparison logic receives as input two K-hit values. A bit-by-hit comparison is done by taking t he exclusive-or o1 i he two inputs. The result is called the syndrome *word.* Thus, each bit of the syndrome. is 0 or 1 according to if there is or •k, not a match in that hit position for the two inputs.

The syndrome word is therefore K bits wide **and** has a range between 0 and — I. The value 0 indicates that no error was detected, [caving 2^K — 1 values to indicate, if there is an error, which bit was in error. Now because an error could occur on any of the r l data hits or K check hits, we roast have

$$2^n - \qquad K$$

This ineguality gives the number of hits needed to correct a *Ingle bit error in a word containing Al data hits. For example. for a word 8 data hits ($M =$ we have

$K = 3$: — 1 < 8 + 3
• $K = 4$: 2^4 · 1 > 4 · 4

Thus, eight data bits require four check bits. The first three columns of Table 5.2 lists the number of check hits required for various data word lengths.

Table 5. Increase in Word Length with Error Correction

	Single-Error Correction		Single-Error Comc t ti hle-Error Dcteetion	
Data 1#11s	Check Bits	% Incr&19...	Check Bits	% Increase
	4	5(1	5	62.5
	5	31.25	6	37,5
32	fi	1S.7:r.	7	21.875
	7	I11.94	g	12.5
I2S	**8**	625	0?	7.11;
25:6		3..52	10	3.91

For convenience, we would like to gcncra I c a 4-bit syndrome for an K-hi l data word with the following characteristics:

- If the syndrome contains all Os, n0 error has been detected.
- I the syndrome contains one and only one hit **Set to I.** then an error has occurred in one of the 4 chock bits. No correction is reeved,
- If the syndrome ctmlaina more than one bit scr 10 I, then the numerical value of the syndrome indicate's the position of the data hit in error. This data bit is inverted for correction.

To achieve these characteristics, the data and check hits are arranged into a 12 N1 word as depicted in Figure 5.9, The bit positions are numbered from 1 to 12. Those bit positions whose position nurnbcrs are powers of 2 a ri2 designated as check hits. The check bits are calculated as follows, where the symbol ED designales the exel usivc-or operation!

$$C1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7$$
$$C'Z = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7$$
$$C4 = D2 \oplus D3 \oplus D4 \oplus D8$$
$$CS = D5 \oplus D6 \oplus D7 \oplus DK$$

Bit position	12	1 1 --1	10			7	fi	5	4	3	2	
Position number	I:1.00'	1.011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
P11ab4..._....PR Check hit		PI	_P6.	t35	:C8	I}	E1	P	.. C4		C2	T C t _1

Figure 5.9 1.,ayuut of **Data Flits acid** Check Bits

Each check bit operaits on every data bit whose position number contains al in the same bit position as the position number of that check bit- Thus, data hit positions 3, :9, 7, 9, and 11 (D1, D2, D4, 05. U.7) all contain a 1 in the least significant hit of their position number as does CL; bit positions 3, 6, 7, 10, and 11 all contain a 1 in the second bit pOsition, as does *C24* and so on. Looked at another way, bit posilion *e2* is checked by those bits C_j such that Si – Eor example, position 7 is checked by bits in position 4, 2, and I: and 7 = 4 1. 2 —

Let us verify that this scheme works with an example, Assume that the 8-bit input word is 00111001. with data bit D1 in the tightmosl position. The calculations are as follows:

= e o.e o 0 = 1
C:2 = 1 El) oelerie0-1
c4=0e0 e t O –
C8 = 1 El) 1 0 ED 0

Suppose now that data hit 3 sustains an error and is changed from 0 to 1. When the check bits are recalculated, we have

ci=1 eosie1 a)0=i
C2 =iete:1$11130-o
C:4 = 0 a:11 EF) 1 .2) 0 — 0
CS = eleoW0=0

When the new cheek bits are compared lo.rith the old check bits, the syndrome word is formed:

	C8	C4	C.2	C.J.
	0	l	1	1
B	0	0	0	1
	0	1	1	I

The result is 0110. indicating that bit position 6, Which contains data hit 3, is in error.

Figure 5,10 illustrates the preceding calculation. The data and check hits are positioned properly in the 12-bit word. Four of the data bits have a value 1 (shaded in the table), and their bit position values arc XORed to produce the Damming code 01.11, which forms the tour check digits, The entire block that is.siored is 001101001111. Suppose now that data bit 3, in bit position ñ, sustains an error and is changed flora 0 to L..1'hu resulting block is 003101101111. The resulting Hamming code is still 0111, An XOR of the I !attuning code and all of the bit position values for nonzero data bits resells in 0110. The nonzero result detects an error and indicates that the error is in bit position 6.

The code just described is known as a *single-error-correcting* (SEC) code_ More commonly, semiconductor memory is equipped with a single-error-correcting, double-error-detecting (SEC.-DEL)) code. As Table 5.2 shows, such codes requirc one mitlitional MI compared with SEC codes.

Bit position	12	11	10	I				5		3	2	1
Position number	1IMO	1011	1.010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Dula bit	DS	1)7	D6	D5			t.):";	L)2	D1			
Cheek bit									C4		C2	CI
Word stored R5	0.	0	1	1	0							
Of fetched as		0	1	L	0	1	a	0				
·.7411T24 pumficr	1100	10]	[1)10	10.01	1000	01[]	0[]0.	.0101	0100		0010	0001
CiTiefC fpFt					0						0	

Figure 5.10 Check Bit Calculation

Figure 5,11 illustrates how 7..41ch a eode works, again with a 4-bit data word. 'Mc sequence shows that if two errors occur (Figure 511 c), the chixking procedure goes astray (J) and worsens the problem by creating a third error (c). To overcome the problem, an eighth bit is added 'hal is set so that the total number of is in the diagram is even. The extra parity bit catches the. error (f).

An error-correcting code enhances the reiiability of the memory at the cost of added complexity. With a one-bit-per-chip organiza0on, an SEC-DED code is generally considered adequate, For example. the IBM 30xx implementations use an 8-bi1 SFC-DED wile 02Lch 64 bits of data in main memory. Thus, Lhc of main memory is actually about 12% larger lion i:l app4m:J11. to the user. The VAX computers use a 7-bit SEC-DED for each 32 hits s.)1 memory, for a 22% overhead. A number of contemporary DRAMs use 9 check bits for each 128 bii;s of c1a1H, for a 7% overhead ISIIA11971.

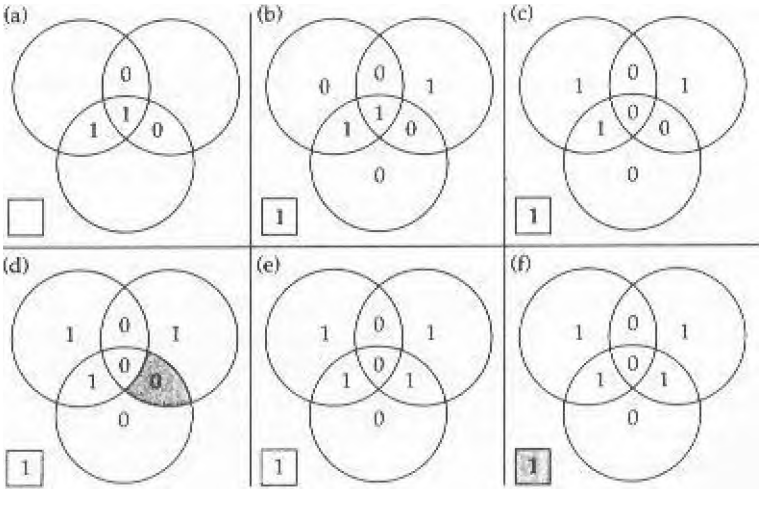


Figure 5.11 Hamming SEC-DEC Code.

5.3 ADVANCED DRAM ORGANIZATIO

As was discusscl in Chapter 2, one of the most critical system bottlenecks when using high-performance processors is the interface to main internal memory, This interface is the most important pathway in the entire computer system. Thu basic building Nock of main memory remains the DRAM chip, has for decades; until recently. there had been no signi ricanI changes in DRAM architecture since the early 1970s. The traditional DRAM chip is constrained both by its internal architecture and by its interface to the processor's memory bus.

We have seen that one attack on the performance problem of DRAM main memory has been to insert one or more levels of high-speed SRAM cache between the DRAM main memory and the processor. But SRAM is much costlier than DRAM. and expanding cache size beyond a certain puinl. yields diminishing returns.

In recent years, a number of cnhanceTnenis to the basic DRAM architecture have been explored, and some of these are now on the market. The Iwo schemes that currently dominate the market are SDRAM and RDRAM. RANI has also received considerable attention, We examine each of these approaches in this section.

Synchronous DRAM

One of the most widely used forms of DRAM is the synchri moos DRAM (SDRAM) NOCTL.941, Unlike the traditional DRAM, which is asynchronous, the SD RAM exchRT1gcs data with the processor synchronized to an external clock signal .and running at the full speed of the processorimemory bus without imposing wait states.

In a typical DRAM. the processor presents addresses and control levels to the memory, indicating that *a* set of data at a particular location in memory should be either read from or written into the DRAM. After a delay, the ti cC.css ti me, the DRAM either writes or reads the data, During the access-time delay, the DRAM performs various internal functions, such as *activat* ing (ilk: high capacitance of the row and column Imes. h031.7-.4 the data, and routing the. data out through the outpul buffers. The processor must simply wait through this delay. slowing system performance.

With synchronous access, the DRAM move:,1data in ztnd out under control of Lbc.,y,t,m ciock. Thu procesm)r or other lliasti2r issues the instruction and address information, which is latched by the DRAM. The DRAM thcn responds after a set number of clock cycles. Meanwhile, the master eau safely do other tasks while the SDR AM k processing Lhc requnt.

Figure fi.12 shoiA.s the internal logic of IBlyts 64 Mb SDRAM [11-1Mtil 1. which is typical of SDRAM organization, and Table 5.3 defines the various pin assignments. The SDRAM employs a burst mode io eliminate the address setup time and row and column line prechargc Lime aur the first access. In burst mode, a series of data bits can he clocked out rapidly after the first bit has been accessed. This mode is useful when all the bits to be accessed are in sequence and in the same row of the array as the initial In addiLion, the SDRAM has a multiple-bank internal architecture that improves opportunities for on-chip parallelism.

The mode register and associated control logic is another key feature differentiatin from convtmi iona I DRAMs. It provides a mechanism to cuslornur die SDRAf 1 10 suit specific !,ystern needs. The mode register specifics the

CKE **CKE 'Wafer**

Column decoder

1 Co1unn decoder

cLK —).Cl..1(buffer

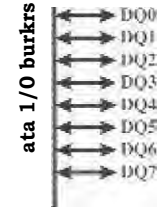
Cell array memory nani; (2Mb x DRAM

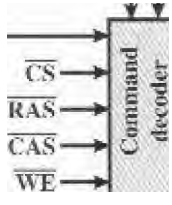
Cell array memory bank I i2 Mb x DRAM

Sense amplifiers

Sense amplifiers

4





Colmnn sikeolfer

Column decoder!

CAC = Column address counter

MR = regisin-

RC - Refresh counter

Ca array mernory bank 2 2 Mb 8) DRA'M

Cell array memory hal* 3 12 x DRAM

Sense amplifiers

Sense amplifiers

pw **5.I.2 Synchronous Dynamic I-Z AM OA).R.A M)**

TaIlk **5.3 SDRAM Pin Assignments**

Ail in Al.!.	AddIL inputs
CLK	Clock Input
CKE	Clock c:nallk'
C7.3	Chip select
RAS	ROW LICIIETOS}, S114.1:bl!
CAS	01113 M R OCIT2SU L11)1).'
WE	Write: 011E1111c
1)0') to D07	Date input:00Lp LIL
1.)'04"..1	Dater TrinSk

burst length, %vhich is the number of separate units of da I a synchronously fed onto the bus. The register also allows the programmer to adjust the latency between receipt of fl read request and the beginning of data transfer.

The SDRAM performs best when it is transferring large blocks of data serially, such as for applications like word processing, spreadsheets, and multimedia.

Figure 5.13 shows an example (4 SDRAM operation. In this case. the burst length is 4 arid Ihr latency is 2. The burst read command is initiated by having CS and CAS low while holding RAS and WE high at the rising edge of the clock. The address inputs determine the Martin column address For the burst, and the mode register sers the type of burst (sequential or interleave) and the burst Length (1, 2, 4, 8, full page). The delay from the start of the.cornmand to when the. data from the first cell appears on the outputs is equal to the valet.' of the CAS latency that is set in the mode register.

There is now an enhanced version. of SD RAM, known as double data rate SD RAM (DDR-SD RAM) that overcomes the once-per-cycle limitation. DDR-SDRAM can send data to the processor twice per clockcy cle_

Ratnbus DRAM

RDRAM, developed by Rambus [FARM92. CRIS97], has been adopted by Intel for its Pentium and Itanium procmors. It has become the main compelitur SDRA M. RDRAM chips are vertical packages, with al] pins on one.side, The chip eN.changes data with the processor over 28 wires no more than 12 centimeters long. The bus can address up to 320 RDRANI chips and is rated ac 1.6 Gaps.

The special RDRAM bus delivers address and control information using an zft.vnchronons Hoek-otiented protocol. After an initial 480 ns access time. this produces the 1.6 Gaps data rate. What makes this speed possible is the bus itself, which defines impedances, clocking, and signals very precisely. Rather than being controlled by the explicit RAS, (AS, R.PW, and CE signals used in conventional DRAMs, an RDRAM gets a memory request over the high-speed bus. This request contains the desircd address, the type of operation, and the number of bytes in the. operation,

Figure 5.l4 ill asi ra[e:':. the RD RAM layout. The. configuration eunsixis of a cum roller and a number of RDRAM modules connected together via a common

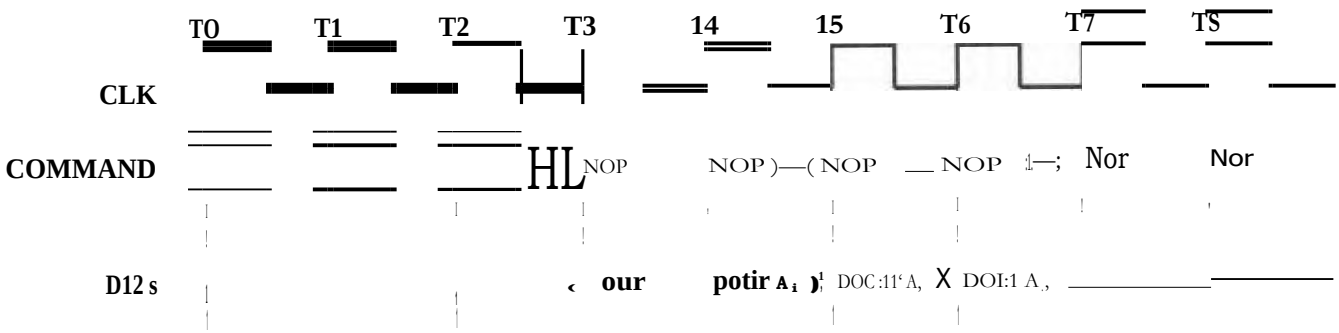


Figure 5.13 STYRAM Reid Timing (burst. leirgth = 4, CAS latency = 2)

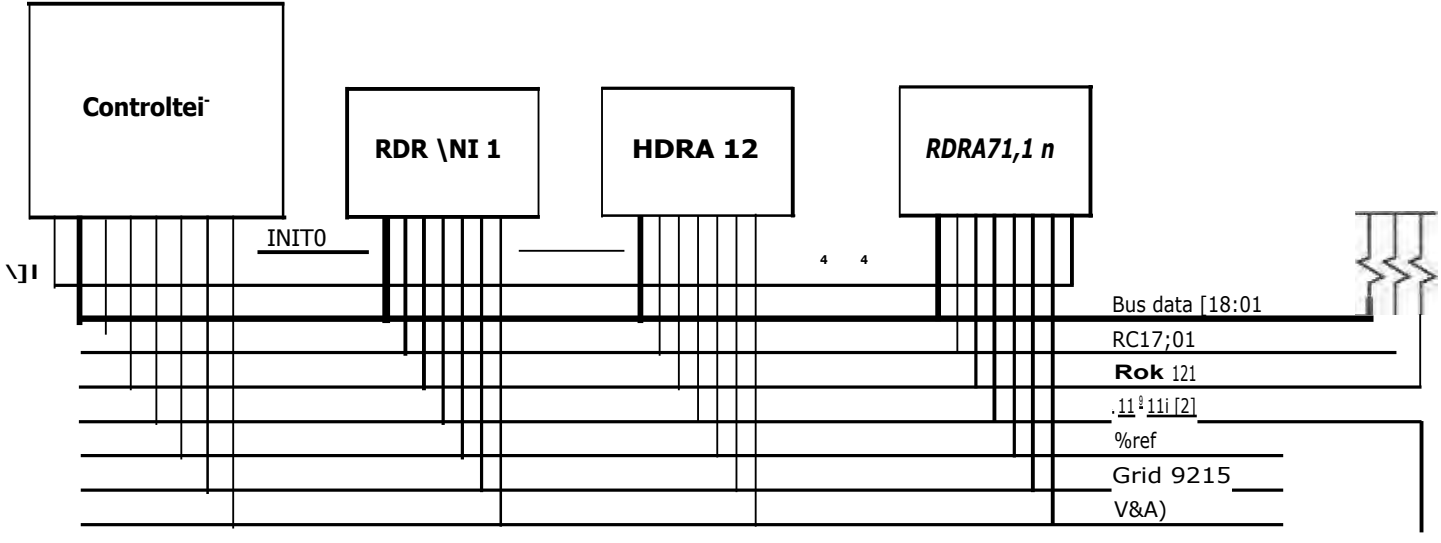


Figure 5.14 RDRAM SErtp. tum

bus. The controller is at one end of the configurai ion, and the far end of the bus is a parallel termination of the butt lines. The bus includes 1S data lines (16 actual data, two parity) cycLing twice the clock rate; that is one hit is tent at the leading and following edge of each clock signal. This resukEs in a signal rate on each data line of 800 Mhps. There is a separate set of 8 lines (RC) used for address and control signals. There is also a cloek signal that starts at the far end from the contrfillur propagates to the controller end and then loops bi]ck. A RDRAM module sends data to the controller synchronously to Ike clock to master, and the controller sends data to an RDRAIVI synchronously with the clock signal in the opposite direction. The remaining bus lines include a reference voltage, ground. and power source.

Cache DRAM

Cache DRA (CDRAM), developed by Mitsubishi [HI [)A]f}, ZHAN011, integrates a small SRAM cache 06 Kb) onto to gencrie DRAM chip.

The SRAM on the (DRAM can be used in two ways, First, i1 can be used as a true cache, consisting or a number of 64-bit lines. The cache mode of the CDRAM is effective for ordinary random access to memory.

The SRAM on the CDRAM can also he used as a buffer to support the serial access of a block of data, For example, to refresh a hit-mapped screen. the CDRAM can prefetch I he data from the DRAM into the SRAM buffer. subsequent accesses to Eh. ehip result in accesses solely to the SRAM.

5.4 RECOMMENDED READING AND WEB SITES

PR1N9 I provides a comprehensivc oi semiconductor memory tedmoiogies, including SRAM. DRAM, and flash inentorii..s. [SI EAR917] COVCTS the same maieriul, with more emphasis on testing and roliabi[i(y issues, l'fiR)N9r')1 focuses on advanced DRAM and SRAM architectures. For an in-depth look al IMAM., see IKEET011.

A good explanation ni error-correcting, codes ix conlain.cd in [MCELS5]. For a deeper s[oc]k, worthwhile book-lelligili treatments are IADAM91.] and [BLAII.831. [SHAR97] con-...rills S 50001 survey of codes used in contemporary main memories.

ADANT91 Ada mok, J. *Ftwo?dations of C'oding, New* YIN k %Vile:yr I i.19 I

BL A..1113.3 BUhiit, R. *Theory and Practice* (3.f Ert:or *cot rird* Re.ading. MA; Addison-1y'LL.sley..1083.

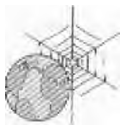
KEF.T01 Reeth, B. and Baker, R. *DRAM Circarif Dili got; A Tzar:47'411.* Piscataway, Nil: IEEE Press, NW.

ri4CF-1,85 McElicue, R. "i'he Reliability of Comprii4...r Memories." *Scientefie: American,* January 1985.

PR1E491 Ptiiice, 13. *Semicotsdurriw. .34,2rtiori.c.v.* New York: WiTcy, 1991.

PRINT) lifinc.d, H. *MR)? PertbrrPranCe Memories:: Neiv Arch &corn' nRA.Ms and SRAM's, .F.:volJaion axed Firneaon.* 'dew York; Wiley. 1999.

51-1.4297 Sharma, A. *Sm. iconducior Alenakries.: tryhnedogy, Tesritig. anei* New York: IEEE Press, 19447,



Recommended 1,Veh Sites:

- The RAM Guide: Good overview of RAM technology plus a number of useful links
- RamhusSite: Useful collection of documents and pointers to RDRAM vendors
- RDRAM: Another useful site for RDRAM information

5,5 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

key 'lertriS

cache DRAM (CDRAM)	Hamming code	single-error-correcting (SEC) code
dynamic RAM {DRAM)	hard failure	single-ern tr-correcting, do utile • error-detecting (SEC-)1^2,D) code
electrically crasablc programmable ROM (FE:PROM)	nonvolatile memory	soft error
erasable programmable (EPROM)	programmable ROM (PROM)	static RAM SRA M
error-corr..din code (EC(?)	R am Bus DRAM (RDRAls.1)	synchrontnis DRAM (SD RAM)
error correction	read 'nmstly memory	syndrome
flash memory	read-only memory (ROM)	volatile, memory
	semiconductor mennir	

Review Questions

5.1 What arc the key properties OF semiconductor memory?
5.2 What arc two senses in which the term *remr,om-access memory* is used?
5.3 What is the difference between DRAM and SRAM, in terms of application?
5.4 What is the difference between DRAM and SRAM, in terms of characteristics such as speed, size, and cu-5r?
5-5 Explain why one type of RAM is considered to he analog and the other digital.
5:6 What arc some applications for ROM?
5.7 What arc the differences among EPROM. F.F.PROM, and flash rnetnorv?
5.8 Explain the function of each pin in Figure 5.4b,
5.9 What is a parity bit?
5.10 How is the syndrome for the Hamming code interpreted?
5.11 How does SDRAM differ from ordinary DRAM?

Problems

Si Suggest reasons why RAMS traditionally have been organized as only one bit per chip whereas ROMs arc usually organiz.ed will' multiple bits per chip-

5.2 Consider a dynamic RAM that must be gi‘ren a refresh cycle 64 times per rns. Each refresh operation requires 150 ns; a memory cycle requires 250 ns. What percentage of the mernor,''s total operating time must be given to refreshes?

5.3 Design a 16-bit memory of total capacity 8192 hits using SRAM chips of size 44 hit. Give thi:. tray erpiillguration of the chips on the memory board showing all requited ioput and output signals for assigning this hicintiry to the lowest address sp4icc. The design should allow for both byte and 16-bit word accesses_
ʃ (.4l: E 93

5.4 For the Hamming code shown in Figure 5.10, show what happens when a chock bit rather than a data bit is in crror.

5.5 Suppose an data word stored in memory is 1101100111 Using the Hamming algorithm. determine what check bitg. would he stored in memory with the c..lara word. Show how you got your answer.

5.6 For the word 00111001, the check bits stored with it would be 1)1 I I. Suppose wizen the word is read from memory, the check hits arc calculated to he 1101. What is the data word that was road from memory?

5.7 How many check hits are needed if the Hamming error correction code is used to detect single bit errors in a 1024-1}ii daiii word?

5.8 Develop an SEC code for a 16-hit data word. Generate the code for the data word 0101 U011110001 I I WI.. Show [hat the code will correctly identity an error in data bit 5.
Source!: [ALEX931

CHAPTER 6

EXTERNAL MEMORY

6.1 Magnetic Disk

ni,1i1Tii 1iL Ruad anti NVI iie Meat isms
Dab, orgarriz.tion ;Ind 1"orinatting
Physical Charactoristics
Disk Performanc`-e PararnetersH?

6.2 RAID

RA [I) Love] 0
RA ID [.ove] 1
RAID Level 2
RAID Love] 3
RAID Love[4
RAID 1 .evel
RAID Level (i

6.3 Optical Memort

Compact Disk
Digital Versatile Disk

6.4 Magnetic Tape

6.5 Heunmniended Rending and Web Sites

6.6 Key Terms. Review Questions. and Problems

Kev Terms
Review OuosIliorts
Problems

.KEY POINTS

- Magnetic disks remain the most important component of external memor).. Both removable and fixed, or hard, disks are. used in systems ranging from personal computers to mainframes and supercomputers.
- To achieve greater performance and higher availability, a popular scheme on servers and larger systems is the RAID disk. technology. RAID refers to a family of techniques for using multiple disks as a parallel array of data storage devices, with redundancy built in k compensate for disk failure.
- Optical storage teennOlogy has become. increasingly important in all types of computer systems. While CD-ROM has been widely used for many years. more recent technologies, such as writable CD and DVI), are becoming. increasingly important,

Thi., chapter examines a range of external memory devices and systems. We ocon with the most important device, the magnetic disk. Magnetic disks are thE foundation of external memory on virtually a]1 computer systems, The ric xi section examines the use of disk arrays to achieve greater performance, looking specifically at the family of systems known as RAID (Redundant Array of independent Disks). An increasingly important component of many computer systems is external optical memory, and this is examined in the third section. Finally, magnetic tape is described.

6.1 MAGNETIC DISK

A disk is a circular platter constructed of nonmagnetic material, called the substrate, coated with a magnetizable material. Traditionally, the substrate has been an aluminum or aluminum alloy material. More recently, glass substrates have been introduced. The glass substrate has a number of benefits, including the following:

- Improvement in the uniformity of the magnetic film surface to increase disk reliability
- A significant reduction in overall surface defects to help reduce read—write errors
- Ability to support lower fly heights (described subsequently)
- Better stiffness to reduce disk dynamics
- Greater ability to withstand shock and damage

Magnetic Read and Write Mechanism

Data are recorded on and later retrieved from the disk via a conducting coil named the head; there are in many systems two heads. a read head and a write head. During a read or write operation. the head is stationary while the platter rotates beneath it.

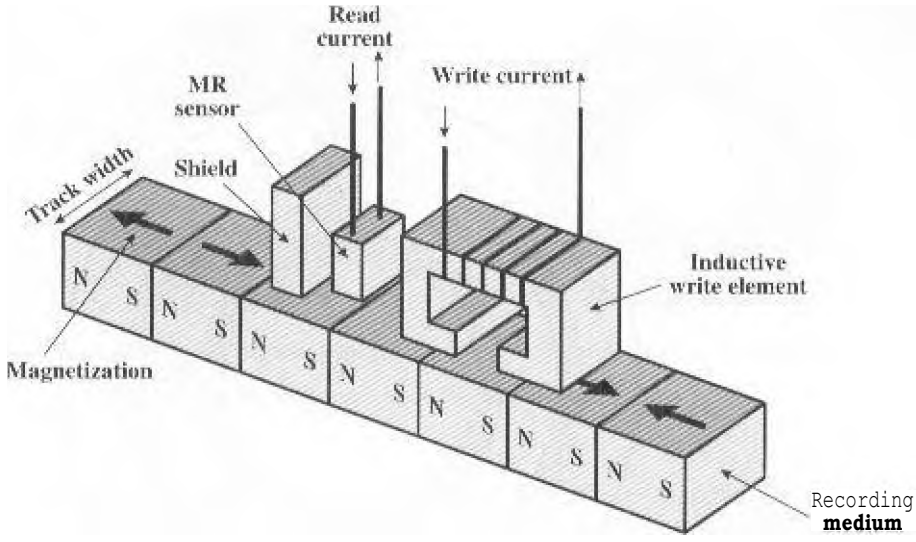


Figure GI Inductive Wrik,'MagncLorcsistivc Read F1 .ad

The write mechanism is based on the fact that electriCit V flowing through a coil **produces a magnetic field. Pulses arc sent to the write head, and magnetic patterns are recorded on the surface below, with different NUei-nz, Ivor** positive and negative **currunt:s. Pic write head itself is made of easily magnetizable material and is is the shape of a rechmgalat doughnut with a gap along one side and a** few lures **of conducting wire along the opposite side (Figure 6.1). An electric current in the wire induces a magnetic field across the gap, which in turn magnetizes a small area of the recording Tnedium. Reversing the direction of the current reverses the direction c. pf the magnetization on the** recording **medium.**

The traditional read mechanism is based on the fact that a magnetic field moving relative to a coil produces an electrical current in the coil. 'When the surface of the disk passes under the head, it generales a current of the S41111e polarily as the one already recorded. 'The structure of the head for reading is in this case essenLiAv **the same as for writing and therefore the same head can be used for both. Such single heads are used in floppy disk systems and in older rigid disk systems.**

Contemporary rigid disk systems use **a different read ineehanism, requiring a** separate read **head. positioned for convenience close to the write head. The read head consists of .a partially shielded magnetoresistive (MR) sensor. The MR material has an electrical** resistance that **depends on the direction of the magnetization of ihe medium moving under it. By passing a current through the MR sensor, resistance changes are detected as voltage signals. The MR design allows higher-frequency operation, which equaics to greater storage densities and operating speeds.**

Data Organization and Formatting

The head is a relatively small device capable of reading from or writing to a portion of the. piai ter rotating beneath it. This gives rise to the organization of data on the platter in a concentric set **of rings. called tracks. Each track is the same width as the head. There ;ire Ihuusands** of tracks per surface.

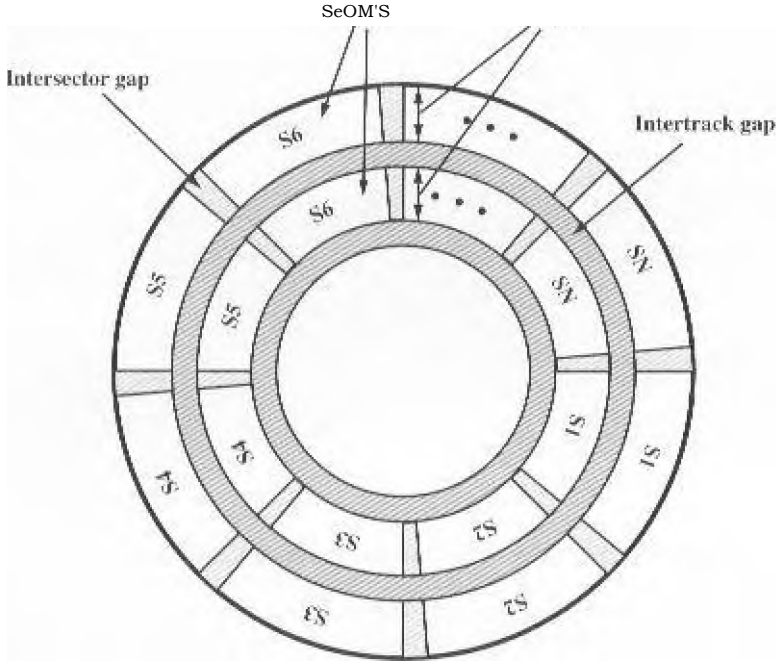
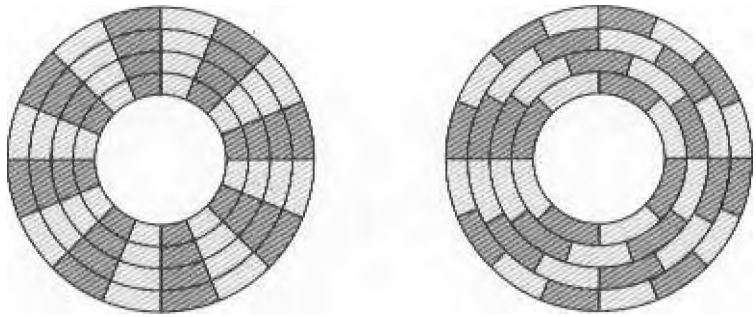


Figure 6.2 Disk Data Layout

Figure 6.2 depicts this data layout. Adjacent tracks are separated by gaps. This prevents, or at least minimizes, errors chic lo misalignment of the head or simply interference of magnetic fields.

Data are transferred to and from the disk in sectors (Figure 6,2). There are typically hundreds of sectors per track, and these may be of either fixed or variable length. In most **coniernponiry** fixed-length sectors are used. wish 512 bytes being the nearly universal sector size. To avoid imposing unreasonable precision requirements on the system, adjacent sectors are separated by intratrack {inter sector) gaps.

A bit near the center of a rotating disk travels past a fixed point (such as a read—write head) slower than a bit on the outside. Therefore, some way must be found to compensate for **the** variation in speed so that the head can read a]] the bits at the same rate. This cart be done by increasing the spacing between his of informalion recorded in segments of the disk. The in roemation can then *be* .scanned at **the same** rate by rotating the disk at a.fixed speed, known as the **constun€ angular velocity (CAV).** Figure 61a shows the layout of a disk using CAV, The disk is divided into a number of pie-shaped sectors and inter 21 series or concentric tracks. The **advan** tap of using CAV is that individual blocks of data can he directly addressed by track and sector. To move Lhe head from its current local ion to a specific address, it only takes a short movement of the head to a specific **track** and a short wait rot the proper sector to spin under the head. The disadvantage of C7AV is that the amount of dale that can be stored on the long outer tracks is the same as what can be stored on **the short** inner tracks_



(a) Constant angular velocity (b) Multiple zoned recording

Figure 6.3 Compariman of Disk Layeu(Meatocls

Because the density. in bits per linear inch, increases in moving from the outermost track Icy I hi: innermost tr,Hek. disk storage capacity in a straightforward CAV system is limited by the maximum recording density that van be **0Qhievcd** on the innermost track. To increase density, modern hard disk systems use a technique known as multiple zone recording, in which the surface is divided into a number of zones (l6 is typicA). Within a Aorie, the number of bits per track is consl ant, Zones farther from the center contain more bits (more sectors) than zones closer to the center. This allows for greater overall sloragc capacity at the expense of somewhat more. complex cireunry. As the disk head moves from one zone to Smother, **The** length (along the track) of individual bits changes, causing a change in the timing for reads and writes. Figure 6.3b suggests the naiurc of multiple zone recording in this illustration. each k only a single track wide.

Some means is needed to locate sector positions within a track. Clearly. there must be some starting point on the track and a way of identifying the start and end of each sector. These requirements are handled by means of control data recorded on the disk, Thus, the disk is formatted with some extra data used only by the disk drive and not accessible to the user.

An example of disk formatting is shown in Figure In this case, each track **conluins 30** fixed-length scctors of 600 bytes each. Each sector holds 51.2 bytes of data plus control information useful to the disk controller. The II) rick] is a unique identifier or address used to locate a particular sector. The SYNCH byte is a special bit pattern that delimits **the** beginning of the field. The track number identifies a track on a **soriux- the** hcrid number identifies a head, because this disk has multiple surfaces (explained presently). The ID and data fields each contain an error-dcteeting code.

Physical Characteristics

Tablc (ILl lists the major characteristics that differentiate among the various types of magnetic disks. First, the head may either be fixed or movallte with respect to the radial direction of the platter, In a fixed-head disk, there is one read-write head per track. All or the heads are mounted on a rigid arm that extends across all tracks; such systems are rare today. In a movable-head disk. there is only one read-write head. Again, the head is mounted on an arm. Became the head must be able to be positioned above any track, the arm can be extended or retracted for this purpose.

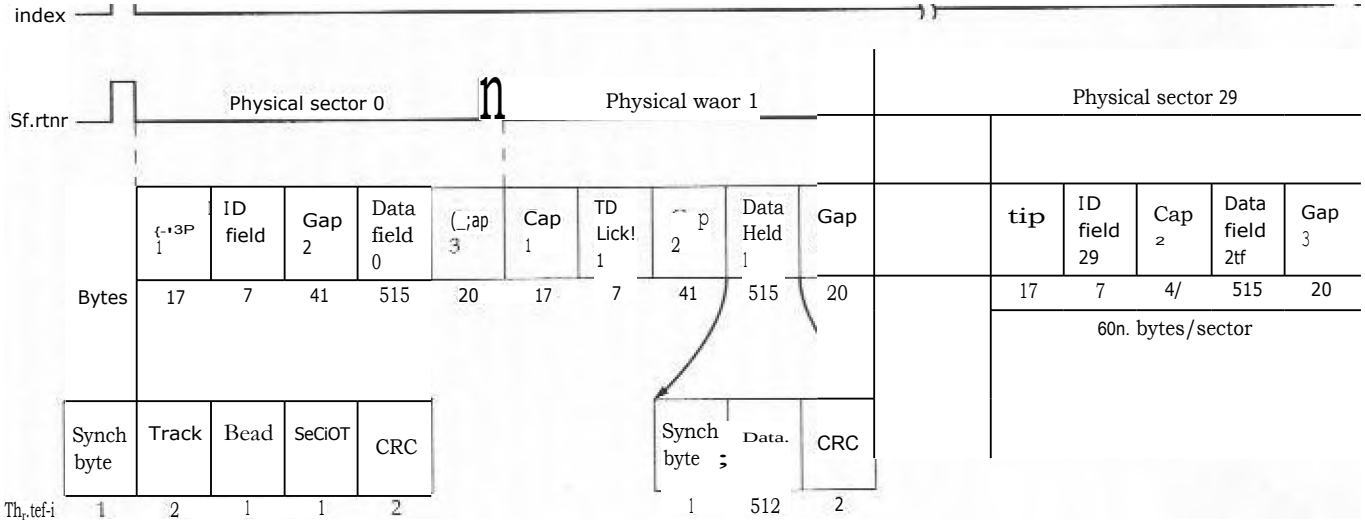


Figure 6.4 'IrVinchestur Disi. Back Format (Seagate ST506)

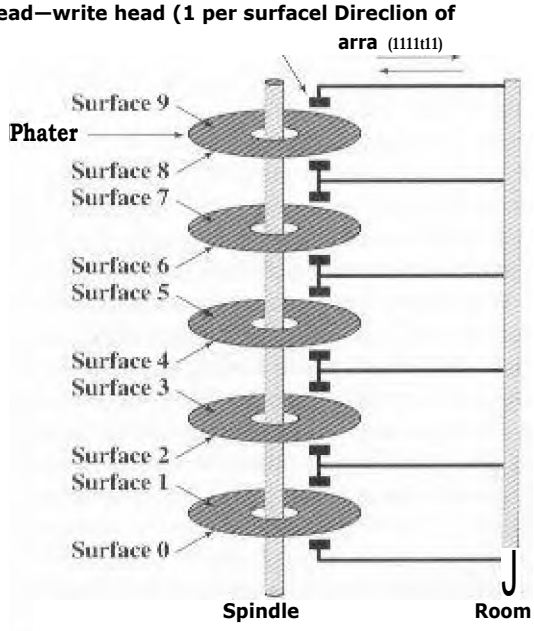
Table 6.1 **C** ıı**M** ·ıl **OriH tiCS iI I)i** sk System"'.

Head Moi ION	Phriten
11,2ad one pur Luacli;)	Sin.g[o p[;il IL r
Mow:11.th: twnd ir.:1C2	MultipIL pi:111.12r
Porlabitity	**Read Vitchanim**
Nourcincivkiblz ElksiE	C.batact .(tloppy)
Reinova bk. disk	Fixed gap
SicicA	Aerodynamic gap (Wirichi:;5100
Si riAl e	
I h.) It sIJLd	

The disk itself is mounted in a disk drive. which consists of the arm, Eishari that rotates the disk. and the electronics needed for inpui **and** output of binary data. A **non re rnovahl u disk is** petrnanentl!,' mounted in the disk drive.; the hard disk in a personal computer is a nonremovable disk. A removable disk c.In he **TUmoved and** replaced with another disk. The advantage. of the latter type is that unlimited amount...; or data arc. **available** with a limited number of disk systems. Furthermore, such a disk may he moved from one computer system to another, Floppy disks mid ZIP cartridge disks are examples of removable, disks.

For most disks, I he ıı ı ki**gnednible** coating is to both sides of the platter, which is then referred to as double sided. Sonic less expensive disk systems single sided **disks.**

Some disk drives accommodate multiple **philters** stacked vertically a fraction of in inch apart. Muttipie arms are provided (Figure 6.5). Multiple-platter disks



urc Compownts of a Disk Drivc.

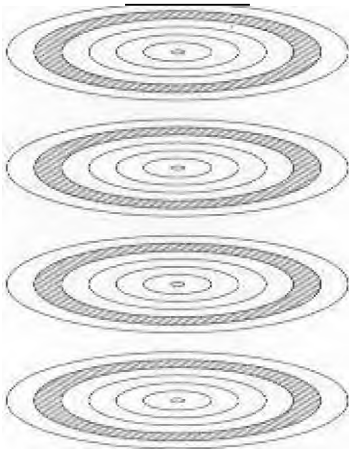


Figure 6.6 Tracks and Cylinders

einplo!,. a me val-FIc head. with one read-write head per platter surface. All of the heads are mechanically fixed so that **all** are at the same distance from the center of the disk and move together. Thus, at any time, a]] of the heads are posilioned over tracks that are of equal distance from the center of the disk, The set c..4 all i he tracks in the same relative position on the platter is referred to as a **cylinder.** For example. **all** of the shaded tracks in Figure 6.6 are part of one cylinder.

Firm I lv, the head mechanism provides a classification of disks into three types- Traditionally. the read-write head has been positioned fixed distance above the platter, allowing an **air** gap. At the other extreme is a head **mechanism that** actually conies ink) physical contact with the medium during a read En write operation. **This mechanism it used with** the '**hippy dirk,** which is a small, flcsible platter and the [east expensive type of disk.

To understand the third type of disk, we need to comment **on the relationship between** data density and the size (.1.1 the air gap, The head must gerieraiQ or weave art eiectromagnetic field of sufficient magnitude to write and read properly. The narrower the head is. the closer it must be to the platter surface to function. A narrower head means narrower tracks and therefore greater data density. which is desirable. However, the closer the head is to the disk. the greater the risk of error from impurities or imperfections. To push the technology further, the 'Winchester disk was developed, Winchester heads are used in sealed drive assemblies that are almoSt free or contaminan1s. They are designed to operate closer to the disk's surface than conventional rigid disk heads, thus allowing greater data density. The head is actually an aerodynamic foil that rests lightly on the platter's surface when the disk is motionless. '[']he pressure. generated By, a spinning disk is enough 10 make the rise aho the surface. The resulting noncontact system can be engineered to use narrower heads that operate closer to the platter's surface than conventional rigid disk heads.

As a m Alter of basl miva I i icrrn Wiriehesser wos Origi usod he ITN ,;.ts ri code niimc. I'm the 3MLI. disk model prior to its aruloWICerrtent. Thu 3:'AD WM it miiscavable ckir k pack with the heads sealed within she pack. The term is now applied to any sealed-unit disk drive with aerodynamic hend design, . I be Winchcsuo disk is curninonly found built in to pci-sprimicomputm and WI WkNtations, whom it is mlun-ed to k& hard disk.

!Able 6.2 Typical Hard Disk Drive Pararnotcrs

Charauleristies	Seagate Barracuda 180	seagate Cheelab X15-36LP	Seagate Barracuda 361-S	Toshiba HD D1242	IBM Mir odrive
Appkica bon	Hi 01-CZI prILI y SZTVer	Iiip.11-performance	Entry-11.:wc1 .dctskicip	Portable	Handheld devices
Capoi.-ity	Ikttb CJI	36.7 GB	1H.4 UTB	5 GB	I OB
Mintal track40-1Taek seek time	0. t? m s	4,1.3 rns	im		(.0 roS
2%VLYra e sock Li me	7.4 rits	3.6 ins	9.5 rrL!.	15 ms	
Sp.indlrz ETC2C.1	721K1 rpm	.151(it nt	7200	4200 rpm	301[rprn
Average Rst4k1tonalF delay	i.17 ms	2 roS	4-17 013	7.1.4 tris	S..33 a1.5
Maximum Lranac.r rate	160 :ABA	522 1.3 719 MI3:s	25 Ma:5	66 MBIs	13.3 MBI.s
By per sector	512	512	512	512	572
SW{ Fr per 'track	793	4g5	N11)		
Tracks per cylindu (riumhcr (31: plpito- $LITRICC.9	24			2	
of track s On km::: sidc p.inttur)	24.247	L8,479	2!4..•551	10,350	

Table 6.2 gives disk parameters for typical contemporari;' high-performance disks.

Disk Performance Parameters

The adu;i4 details of disk 1/0 operation depend **on** the computer system, the operating system, **and the** nature of the WO channel ;..ind disk controller hardware. A general timing diagram of disk I/O 1rnnsrei' i2.; shown in Figure 6.7.

When the **click** ffrive is operating, the.. disk is rotating at constant speed. To read or write, the head inust be positioned at the desired track and at the beginning of thil desired sector on that track. Track NeIceLion involves moving the hcad in a movable-head system or clixtronieolly selecting one head on a fixed-head 5:,' stern. On a movahle-head sy2tern ; the time it takes to position the head at the track k known as **seek time. In either case, (Alec** the track is selected, the c,lisk controller waits until the appropriatc scul car rotates to line **up** with the head, **The time** it takes [or the **beginning** of the sector to reach the heart is known as **rotational delay, or** rotational latency. The sum of the seek iirric, if any, and the rotational delay equals the access time, which is the time it takc, 10 get into position to read or write. Once the head is in position. the rend or write operation is then performed as the sector

moves under the head; this is the data transfer portion of the operation: the time required for the transfer is the **transfer time.**

In addition to the access time and iransier time, there are several queuing delays normally associated with a disk I/O operation. When a process issues an l.[1]0 request, it must first wait in a queue for the device to be availabk. AL that lime. ale device is assigned Lo the process. 11 the device shares a single I/0 channel or a set of 170 channels with other disk drives, then there may be an additional wait for the channel to he available. At that point, the seek is performed to begin **disk** access.

In some high•end sysurns for servers, a teehnique known as rotational positional sensing (RPS) is used This works as follows: When the seek command has been issued. the channel is released to handle other 1/0 operations. When the seek is completed. the device determines when the data will rotate under the head. As that sector approaches the head, the device tries to reestablish the communication path back to the host. 11 either the control **unit** or the channel is busy with another Lhcn Lhc recOnmxtion attempt fails and the device must rotate one whole revolution before it can attempt to reconnect, which is called an RPS miss. This is an extra delay element that must be added to the time line of Figure 6.7.

Seek Time

Seek time is the timei required to move the disk arm to the required track. It turns out that this is a difficult quantity to pin down. The seek lime consists of two key components= the initial startup time, **and** the time taken to traverse the tracks that have to be crossed once the.access arm **is up to** speed. Unfortunately, the traversal time iw not a linear function of Lhc number of tracks, but includes a startup time and a settling time (Hine **aiLer positioning** Lhc head over the tar .et track until track identification is confirmed).

Much improvement comes from smaller and lighter disk components. Serne **years ;Igo.** a typical disk was 14 inches {36 ern) in diameter, whereas the most common size today is 3.5 inches (8.9 cm), reducing the distance that the arm has to travel. A typical average seek time on contemporary hard disks is **under .L0 ms.**

HotAnna' Delay

Disks, oilier Ihan floppy disks, rotate at speeds ranging from 3600 rprn (for handheld devices such as digital carnera.$) up to, as of this writing. 15,000 **rpm at this** latter speed. there is one revolution per 4 nis. Thus, on the average, the rotational

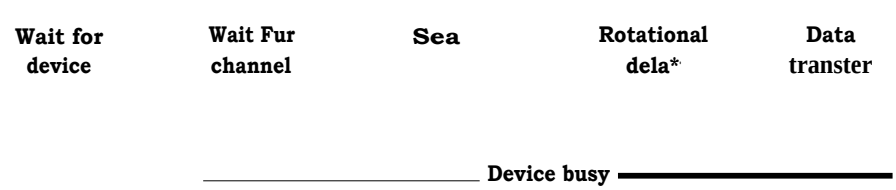


Figure 61.7 Tiiiiirpe of a Disk Transtur

delay will be 2 Ins. Flopp ..• disks typicall!,' rotate at between 300 and 61)1) rpm. Thus the average delay will be. between 100 and 50 ms.

Trunsier Time

The transfer time lo or from t he disk. depends on the rotation speed of the disk in the following fashion;

$$T = \frac{b}{rN}$$

where

_ tramrer time
= number of bytes to be transferred
N = number of bytes on Li track
r rotation speed, in revolutions per second

Thus the total average access time can be expressed as

$$T \quad \frac{1}{2r} + \frac{b}{rN}$$

where 7', js•the average seek time. Note that on a zoned drive, the number 0r bytes per track is variable, complicating the calculation.

A Timing Comparison

With the foregoing pararneiers defined, let us look at Iwo different P.O operations that illustrate the danger or relying on average values. Consider ui disk with an advertised average seek time of 4 ms, rotalion speed ci f1 ,00() and 512-byte sectors with 500 sectors per track. Suppose that we wish to read a file consisting of 2500 sectors for ik total of 1.2H Mbytes. We would like to estimate the tolal Lime for the transfer.

First. let us assume that the file is stored as compel ly as possible on the disk. That is the rile occupies all of the seekvs can 5 adjacent tracks (5 tracks 500 sectors.? track - 2M10 .sectors). This is known as *sequeraial organiz.rition,* Now, the time to read the first track is as follows:

.Average seek	4 ms
Rotational delay	4 ms
Read 500 sectors	ms
	16 iris

Suppose that the remaining tracks can now be read with essentially no seek time. That is, the 1/0 operation can keep up with the flow from the disk. Then, at most, we need to deal with rotational delay for each succeeding track. Thus each successive track is read in 4 -F 8 = 12 ins. To read the entire fill:,

Total time = l6 - 4 x 2 = 64 ms = 0.064 seconds

Now let us calculate the time required to read the same data using random access rather than sequential access: that is. accesses to the sectors arc distributed randomly over the disk. For each sector. we have

Average seek	4	ms
Rotational delay	4	ms
Read I sectors	0,016	ms
	.016	ms

Total time — 500 x 8,016 4008 ms = 4.008 seconds

It is clear that the order in which sectors arc read from the disk has a tremendous effect on I/O performance. In the case of file access in which multiple sectors arc read or written. we have some control over the way in which sectors of data arc deployed, and we shall have something to say on this subject in the next chapter. However, even in the case of a file access, in a multiprogramming environment, there will be requests competing for the same disk. Thus, it is worthwhile to examine ways in which the performance of disk I/0 can be improved over that achieved with purely random access to the disk. This leads to a consideration of disk scheduling algorithms, which is the province of the operating system and beyond the scope of this book (see [STAL0 I I for a discussion).

6.2 RATC[6] 4'W-F;

As discussed earlier_ the rile in improvement in secondary storage performance has been considerably less than the rate for processors and main memory. 'This mismatch has made the disk storage system perhaps the main focus of concern in improving overall computer system performance.

As in other areas of computer performance. disk storage designers recognize that if one component can only he pushed so far, additional gains in performance arc to be had by using multiple parallel components. In the case of disk storage. this leads to the development of arrays of disks that operate independently and in parallel. With multiple disks, separate 110 requests can be handled in parallel, as long as the data required reside on separate disks. Further, a single .1.10 request can be executed in parallel if the block of data to he accessed is distributed across multiple disks,

With the use of multiple disks, there is a wide variety of ways in which the data can be organized and in which redundancy can be added to improve reliability_ This could make it difficull to develop database schemes that are usable on a number of platforms and operating systems. Fortunately, industry has agreed on a standardized scheme for multiple-disk database design. known as RAID (Redundant Array of Independent Disks), The RAID scheme consists of seven levels. : zero through six.

---

[6] Additional levels have been defined by sonic resc,3rchers and some companies. but the seven levels &scribed in this section sire ihti ones universally aereed on.

1 hese icveis do not imply a hierarchical reialionship but designate different design architectures that share three common (,:haracieristics

1. RAID is a set of physical disk drives viewed by the operating system as a single logical drive.
2. Data are distributed across the physical drives of an array.
3. Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

The c,letai Is of the second and third characteristics differ ror the different RAID leycis. RAID 0 does not support the third uharacteristie,

The term *RAID* was originally coined in a paper by a group of re.!'.carChel's at the University or California; at Berkeley [PATTSS],'"Fhe paper outlined various RAID configurations and applications and introduced the definitions of the RAID levels that are still used. The R.A1D strategy replaces large-capacity disk drives with multiple smaller-capacity drives and distributes data in such a way as to enable si multaneous access to data from multiple; chives_ thereby improving 1.0 performance and allowing easier incremental increases in capacity.

The unique contribution of tlic RAID proposal is to address effectively the need for redundancy. Although allowing multiple heads and actuators to operate simull ancously achieves higher I/O and transfer rates, the use of multiple devices increases the probability of failure. To compensate for this decreased reliabiiity, RAID makes use of stored parii y information that enables the recovery of data lost due to a disk faiiure,

We now examine each of the RA 11.3 levels. Table 6.3 summarizes th e seven levels. Of these, levels 2 and 4 are not commercially offered and are not to achieve industry acceptance. Nevertheless, a description of these levels helps lo clarify the design choices in some of the other levels,

Figure 6.8 is an example that illustrates the use of the seven RAID schemes to support a data capaci1y requiring four disks not counting redundane.y. The figure highlights the **layout** of user data and redundant data and in di cates the. relative storage rctiuireritertts of the various levels. We refer Lo this figure throughout !he fok lowing discussion.

RAID **Level** 0

RAID level 0 is not 41 Irttu member of the RAID family. because it does not include~ redundancy to improw performance. However, there are a few applications, such as some on supercomputers in which performance and capacity are primary concerns and low cost is more important than improved reliability,

---

Fri lbw paper. the acrcrnyrn RAID stood for Redundant Array **Or ITIEXprUil'it DIAL The** ⇕UM *inPVprq*ı *vive* was used to contrast the small relatively inexpensive. disks in the. RAID array iu I he allernaLivti, {in glc Inre expenp.ivir disk (SLED}. The SLED is e3sK3utiallls. 9 I hing al the past, with similar 4.1iiik technology being tiled for both RAID and non-RA ED *CI* Fri orations. Accordingl'y, the industry has adopted 1bc term *ItidependeRr* to emphn2,ize that the RAID array maws sign' $ic,1 n 1 irer Eirrninnee and reliability eains.

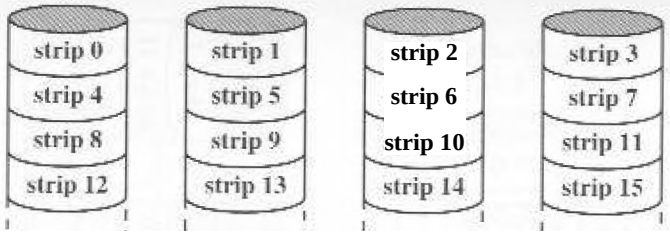
Table 6.3 RAID Levels

Category	Level	Description	Request Rate (Readfri/Vrite)	Data Transfur Rate (Read/Write)	Typieal Application
Striping	6	Koareliandant	I ,arge strips: Rxcellent	Small strips; Excelknt	Applications requiring high performance for noncritical data
Mirroring		MallMed	Good:fair	Fairifair	System drives: critical files
Parallel access	2	Redundant via Hamming code	Poor	tee llcnl	
	3	Bit-interleaved parity	Poor	ENcc.11erit	large 1:0 request size applications, such as imaging, CAD
[Tidcpundent 3CeCsS	4	Block-interleaved parity		Fa iripoor	
	5	Block-interleaved distributed Nrity	Exellentifair	Fairpoor	High request rate, read intermive:. data lookup
		Block-interleaved dual distributed parity	ŧAcciientipoor	Fair/poor	Applications requiring extremehi high avnilablity

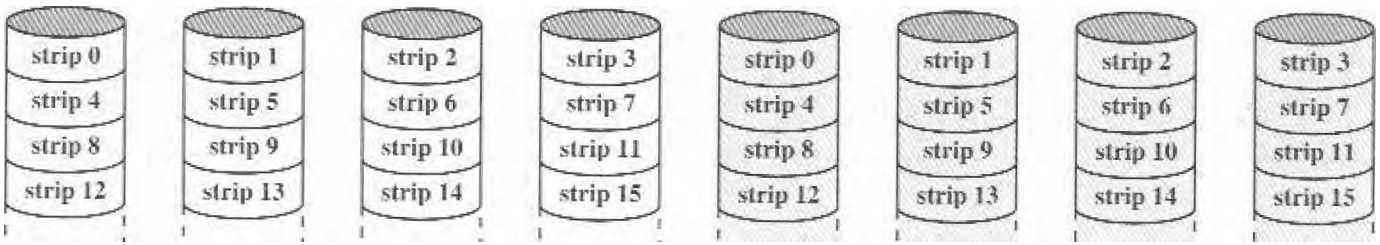
For RAID 0, the- user and system dal a are distributed across all of the disks in the array. This has a notable advantage over the use of a single large disk: If Iwo different 110 requests are pending for two different blocks of data, then there is a good chance that the requested blocks are on different disks, Thus, the two requests can he issued in parallel, reducing the 1!0 queuing time,

But RAID 0, as with all of the RAID levels, goes further than simply distributing the data across a disk array: The data *are striped* across the available disks. this is best understood by considering Figure 6,9. All of the user and system data are viewed as being stored on a logical disk. The disk is divided into strips; these strips may he physical blocks, sectors, or some other unit. The strips are mapped round robin to consecutive array members, A set of logically consecutive strips that maps exactly *one* strip to each array menthe]' is referred to as *a stripe,* In an n-disk array, the first *n* logical strips are physically stored as the first strip on each of the n disks, forming the first stripe; the second *n* strips are distributed as the second strips on each disk; and so on. The advantage of this layout is that if a single request consists of multiple logically contiguous strips. then up to *n* strips for that request can he handled in parallel, greatly reducing the I/O transfer time.

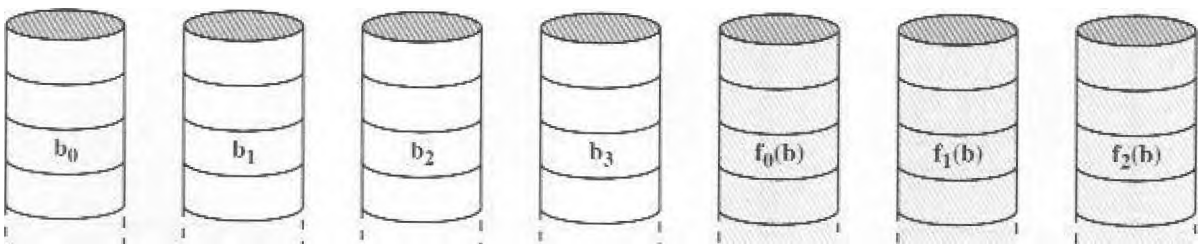
Figure 6,9 indicates the use of array management software to map between logical and physical disk space. This software may execute either in the disk subsystem or in a host computer.



(a) **RAID 0 (Nronredundunt)**

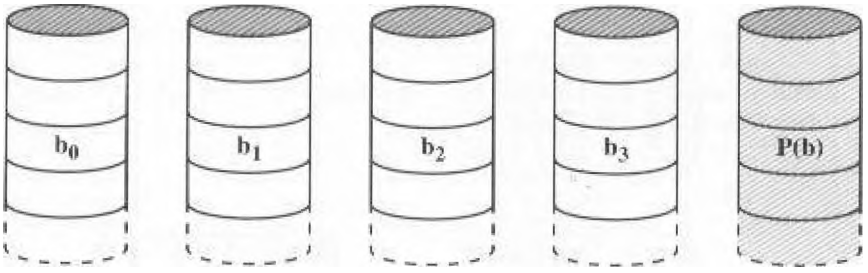


h) RAIL) I (Mirrored)

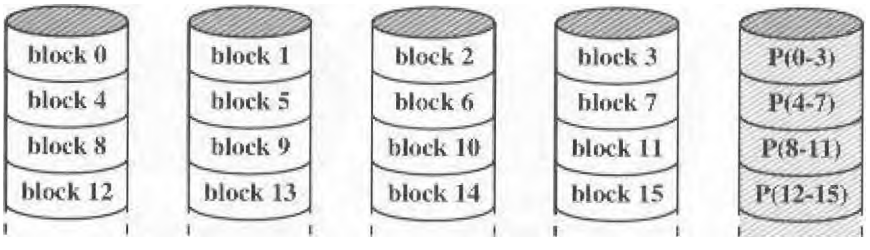


I i i It AID 2 !Redundancy through Ilarnming code)

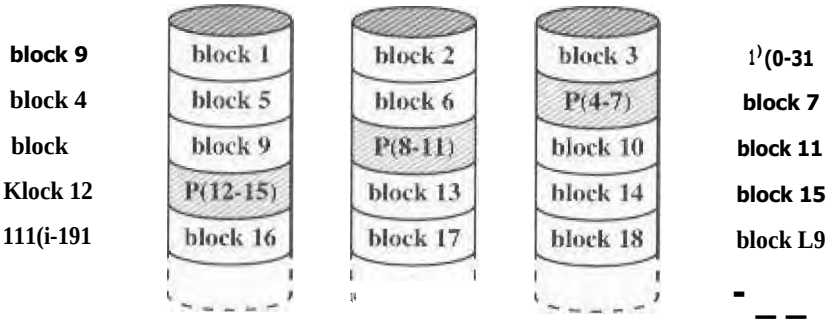
Figure 6.8 R. Lvyels (page 1 of 2)



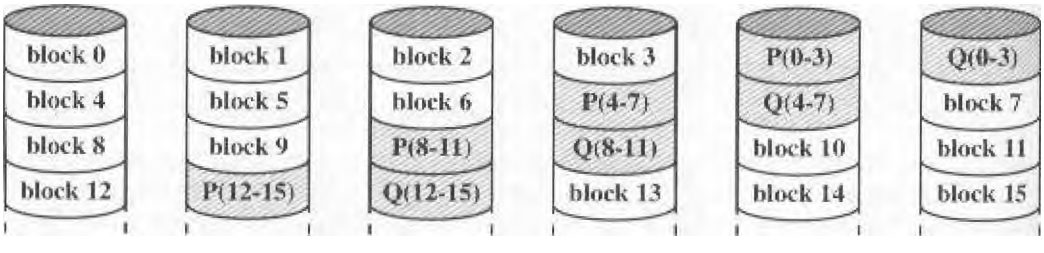
(d) RAID 3 (Rit-interleaved parity)



(e) RAID 4 (Block-level parity)

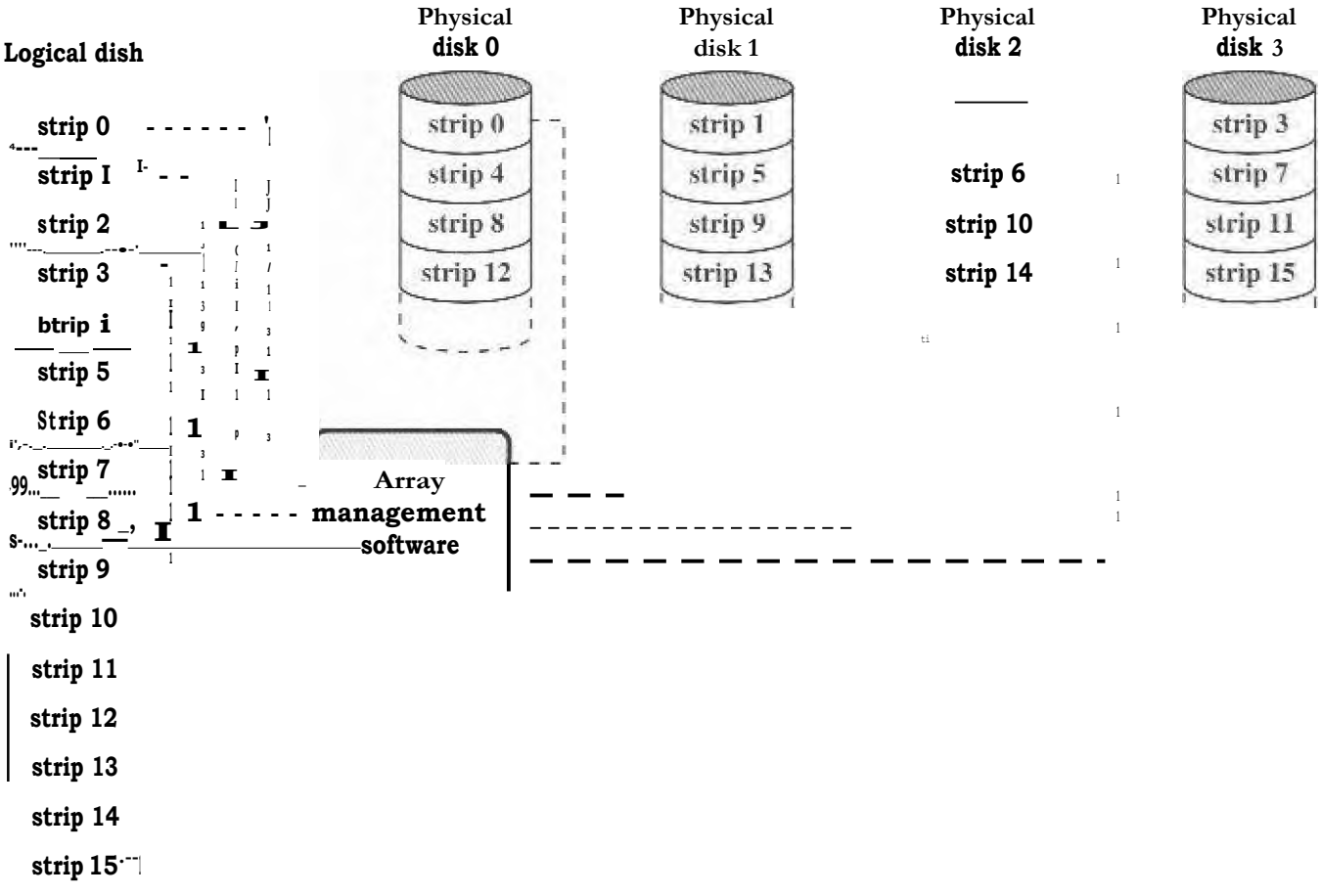


(D RAID 5 i Block-level distributed parity)



(g) RAID 6 (Dual redundancy)

Figure 6.8 RAID Levels (page 2 of 2)



6.9 Data Mapping for n RAID Lcycl 0 Array

RAID 0 for High Data Transfer Capacity

The performance of any of the RAID levels depends critically on the request patterns of the host system and on the layout of the data. These issues can he most clearly addressed in RAID 0, where the hupaci or redundancy does not interfere with the analysis. First, let us consider the use of RAID tl to achieve a high data transfer rate. For applications to see a high transfer rate, two requirements must be met. FirSt, a high transfer capacity must exist along the entire path between host memory and the individual disk drives. This includes internal controller buses. host system 110 buses, 110 adapters. and host memory buses.

The second requirement is that the application must make I/O requests that drive the disk array **efficiently.** This requirement is met if the typical request is for large amounts of logically contiguous data, compared to the size of a strip. In this case, a single I/O request involves the parallel transfer of data from multiple **disks.** increasing the effective transfer rate compared to a single-disk transfer.

RAID 41 For I ligh **I/O Request Rate**

In a transaction-oriented **environment.** the user is typically more concerned With response time than with transfer rate. For an individual request for a small amount of data, the **I/O time** is dominated by the motion of the disk heads (seek ti me.) and the movement of the disk (rotational latency).

In a transaction environment, there may he hundreds of I/O requests per second. A **disk** array can provide high **I/O** execution rates by **balancing the 1/0 load** across multiple disks. Effective load balancing is achieved only if there are typically multiple I/O requests outstanding. This, in turn, implies that there are multiple independent applications or a single transaction-oriented application that is capable of multiple asynchronous I/O requests. The performance will also be influenced by the strip size. If the strip size is relatively large, so that a single 1/0 request only involves a single disk access, then multiple waiting 1/0 requests can he handled in parallel, reducing the queuing time for each request.

RAID Level

RAID 1 differs from RAID levels 2 through C in the way in which redundancy is achieved. **In** these other RAID schemes, some form of parity calculation is used to introduce redundancy. whereas in RAID I. redundancy is achieved by the. simple expedient of duplicating all the data, As Figure 6.8b shows, data striping is used, as in RAID 0. But in this case, each logical strip is mapped to two separate physical disks so that every disk in the array has a mirror disk that contains the same data.

There arc a number of positive aspects to the RAID 1 organization:

L A read request can be serviced by either of the two disks that contains the requested data, whichever one involves the minimum seek time plus rotational **latency.**

2. A write request requires that both corresponding strips he updated, but this can be done in parallel. Thus, the write performance is dictated by the slower of **the** two writes (i.e., the one that involves the larger :seek time plus rota-

clonal latency). However, there is no "write pcmalt:y. ¯ .with RAID 1. RAID levels 2 through 6 involve the use of parity bits. Therefore, when a single strip is updated, thc array management software must first compute and update the parity bits as well as updating the actual strip in question.

3. Recovery from a failure is simple, When a drive fails. the data may still be accessed front the second drive.

The principal disadvantage of RAID 1 is the cost; it requires twice I he disk space of the logical disk that it supports. Because of I hal, a RAID configuration is likely to be limited to drives lhat store system software arid data and other highly critical files. 111 these cases, RAID l provides real-time backup of all data so that in the event of a disk failure, all of the critical data arc still immediately available.

In a transaction-oriented environment, RAID 1 can achieve high 110 request vales if the bulk 01 [he requests arc reads. In this situation, the performance of RAI F.) **1 can** approach double of that of RAID 0. However, if a substantial fraction of the I/O requests are write requests. [hen [here may be no ',i gnificant performance gain over RAID 0- RAID **1 may** also provide improved perf'ormance over RAID 0 for data transfer intensive applications with x high percentage of reads, improvement occurs if the application can split each read request so Thal both disk members participate.

RAID Level 2

RAID levels 2 and 3 make use of a parallel access technique- In a parallel access array, all member disks participate in I he execution or every 110 request. Typically. the ,pindles of the individual drives are synchronized so that each disk head is in same position on each disk at any given time.

As in the other RAID schemes, data striping is used. In the case of RAID 2 and 3. the strips are very small, often as small as a single byte or word. With RAJ!) 2, an error-corteciing code is calculated across corresponding hits on each data disk. and the bits of the. code are stored in the corresponding bit positions oil. multiple parity disks. Typically, a Hamming code is used. which is able to correct single-bit errors and detect double-bit errors.

Although RAID 2 requires fewer disks than RAID I. it is slill rather costly, The number of redundant disks is proportional to the log of the number of data disks. On a single read, all disks are simultaneously accessed. The requested data and the associated error-correcting code are delivered to the array controller, If there is a single-bit error. the controller can recognize and correct the error instantly, so that the read access lime is **not** slowed. On a single write, a]] data disks and parity disks **MUSI he** aectmed for the write operation.

RAID 2 would only be an effective choice. in an environment in which many disk errors occur. Given the high reliability of individual disks and disk drives, RAID 2 is overkill and is not iraplernurited.

RAID Level 3

RAIL) 3 is organized) in a simi kir Fashion to RAID 2. The difference is that RAID 3 requires only a single redundant disk. no matter how large the disk array- RAID 3

employs parallel access, with data distributed in small strips. Instead of an erropeor-reeling code. a simple parity bit is computed for the set of individual bits in the same position on all of the data disks.

Redundancy

In the event of a drive failure, the parity drive is accessed and data is reconstructed from the remaining devices. Once the failed drive is replaced, the missing data can be restored on the new drive and operation resumed.

The data reconstruction is quite simple, Consider an array of five drives in which X0 through X3 contain data and X4 is the parity disk. The parity for the *ith* bit is calculated as follows:

$$X4(i) = X3(0 \; 9 \; X2(i) \; Xl(i) \; ER \; WO$$

Suppose that drive XI has failed. If we add X4(i) e X1 *(i)* to both sides of the preceding equation, we gel

$$Xl(i) \quad X4(i) \; 4 \; X3(i) \; X2(i) \; XO(i)$$

Thus, the contents of each strip of data on X1 can be regenerated from the contents of the corresponding strips on the remaining disks in the array. This principle is true for RAID levels 3 through 6.

In the event of a disk failure, all of the data arc still available in what is referred to as reduced mode. In this mode, for reads, the missing data are regenerated on the fly using the exclusive-OR calculation. When data are written to a reduced RAID.3 array, consistency of the parity must be maintained for later regeneration. Return to full operation requires that the failed disk he replaced and the entire contents of the failed disk be regenerated on the new disk.

Performance

Because data are striped in very small strips, RAID 3 can achieve very high data transfer Tates. Any 110 request will involve the parallel transfer of data from all of the data disks. For large transfers, the performance improvement is especially noticeable. On the other hand, only one 1l**O** request can be executed at a time. Thus, in a transaction-oriented environment, performance suffers.

RAID Level 4

RAID levels 4 through 6 make use of an independent access technique. In an independent access array, each member disk operates independently, so that separate I/O requests can be satisfied in parallel. Because of this, independent access arrays are more suitable for applications that require high 110 request rates and are relatively less suited for applications that require high data transfer rates.

As in the other RAID schemes , data striping is used. In the case of RAID 4 through 6. the strips are relatively large, With RAID 4, a bit-by-hit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk.

RAID 4 involves write penalty when an I/O write request of small size is performed, Each time that a write•oocurs, the array management software must update not only the user data but also the corresponding parit:;,' bits. Consider an array five drives in which X0 ihrough X3 eontain data and X4 is the pari disk. Suppose That a write is performed that only involves a strip on disk Xt. Initially, for each bit *i,* we have the following relationship:

$$X4(i) = X3() \oplus X2(i) \oplus XI(i) \quad \text{WI)}$$

Mier th,2 upda le, with potentially altered bi1s indicated by a prime symbol,

$$\begin{aligned} &= X3(i) \oplus X2(i) \oplus XV(i) \oplus xo(i) \\ &\quad x3(i) \oplus x2(r) \oplus xr(i) \oplus Xt)(i) \text{ EP xi cty ox.1(i)} \\ &= x4(i) \oplus xi\ (i) \Rightarrow Xr(ii \end{aligned}$$

To calculate the new parity, Lhu array management software must read the old user strip and the old parity strip. Then it can update these two strips with the new data and the newly calculated parity. Thus. each strip write illiVO MS two reads and 1wo writes.

In the case of a larger sizc 1.0 write that involves strips on all disk drives, parity is easily computed by calculation using on]}' the new dal a hits. Thus, the parity drive on be updated in parallel with the data drives and there are no extra reads or writes.

In any case, every write operat i0 n must involve the parity c.ikk, which therefore can become a hot ill:neck.

RAID Level 5

RAID 5 is organiAed in a similar fashion to RAID 4. I he difference is that RAM 5 distributes the parity strips across all diski;. A iypical allocation is a round-robin scheme, as illustrated in Figurc. 6.81. For an n-disk array. the parity strip is on a different disk for the first *n s.tri F's,* and the pattern then repeats.

The di'dri bullion of parity strips across **all** drives avoids the potential 1.'0 bottleneck round in RAID 4.

RAID Level 6

RAI D 6 was introduced in a subsequent paper by the Berkeley researchers I I(ATZ89]. In the RAID 6 scheme, iwo different parity calculations are carried out and stored in suparale blocks on different disks. Thus, a RAID 6 array whose user data require N disks con.:-.isis of N -F 2 disks.

Figure 6.82 illustrates the :scheme. P **and Q** are two different data check al-2orithms. One of ihe two is the eXe[LLSive-OR calculation used in RAID 4 and 5. But the other is an independent data check algorithm. .rhis makes it possible to regenerate data even if two disks containing USer data [ail.

The advantage of RAID 6 is that it provides extremely high data avaiiabi]ity. Three dis.ks would have to fail within the M'1'1'1 (mean time to repair) interval to cause data to be lost. On the other **hand, RAID** 6 incurs a substantial write penafty, because each write affects two parity blocks.

6.3 OPTICAL MEMORY

In 1983, one of the most successful consumer products of all time was introduced: the compact disk (CD) digital audio system. The *CD* is a nonerasable disk that can store more than 60 minutes of audio information on one side, The huge commercial success E)1 the CD enabled the development of low-cost optical-disk storage technology that has revolutionized computer data storage. A variety of optical-disk systems have been introduced (Table 6,4). We briefly review each of these.

Compact Disk

CD-ROM

Both the audio CD and the CD-ROM (compact disk read-only memory) share a similar technology. The main difference is that CD-ROM players are more rugged and have error correction devices to ensure that data are properly transferred from disk to computer, Both types of disk arc made the same way. The disk is formed from a resin, such as polycarbonate. Digitally recorded information (either music or computer data) is imprinted as a .scries of microscopic pits on the surface of the poly. carbonate. This is done, first of all. with a finely focused, high-intensity laser to create a master disk. The master is used, in turn. to make a die to stamp out copies onto polycarbonLite. The pitted surface is then coated with a highly reflective surface, usually aluminum or gold. ' Ms shiny surface is protected against dust and scratches by a top coat of clear acrylic. Finally, a label can be silkscrcened onto the acrylic,.

Table 6.4 Optical Disk Products

CI)
Compact Disk. A none•asabli.: disk tkvat stores digitht. ,'d audio inlornsation..l'he standard system uses I 2'ern disks and call record more thall tininh-.trupted 1'iawiu tinK.

(1)'-ROM
Compact Disk Reud-Only MeinCrry , A rionefiSSIIne disk llSed for storinir computer data. The standard system uses .i2--•tts disks And con hold mom than 650 Mbytes.

C1)44
CD Recordable. Similar to a CD-ROM. Thti iisc-1 can write to the disk ord.!, {mice.

C.1.)-RW
C 0 Rewritahle. Similar **a CD-ROM. The user carrerase and rewrite to the disk multiple**

Diiaitil **Vidu.1. Disk. A ituchaolugly for producing digitilced. exinpri,.ksed ropresentation. or video tni•rrnation, fin well as large Yolumws cli ollim digital data. Roth'and 1.2.cm iliametcn; are mud. with a double-sided** capaciuy of up to .17 {.1hyi **basic 11 Y.0 is read-0311 F (DVD-ROM).**

Mit) Recordable. Similar to zk D-ROM. he 0W cun write. im the disk only otio3. Only one-sided disks can tic! used.

DVIto-RW
DVD kcwritable, Sitnilar to a D VD-ROM. The user can write to the disk multiple times. Only 0u-sick:a disks can Tv. used.

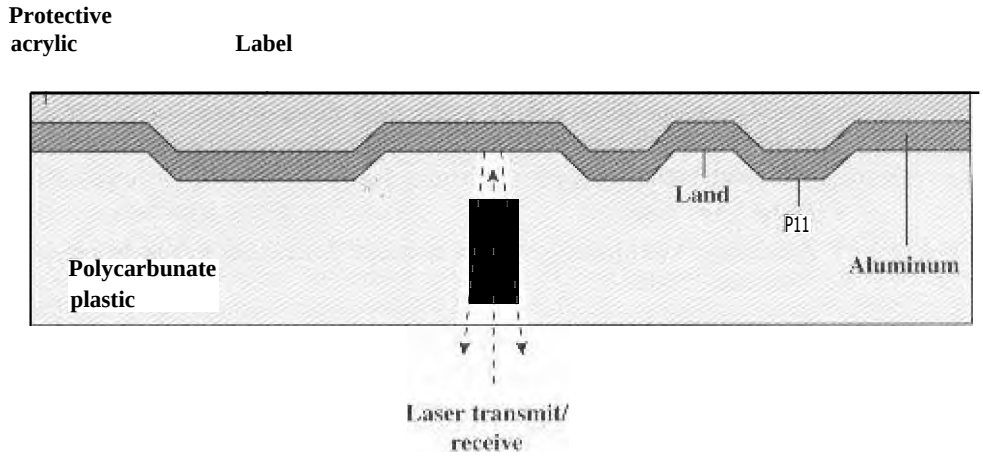


Figure &III CD Operation

Information is retrieved from a CD or CD-ROM by a low-powered laser housed in an optical-disk player, or drive unit. The laser shine!, through the clear **po[ycarbonkice** while a motor Tim [he disk **past it (Figure** 6.11)). The intensity of the reflected light of the laser changes as it encounters a pit. Specifically, if the laser beam falls on a pit. which has a somewhat rough surface, the lieu scatters **and a** low intensity is reflected back to t **he** source. 'Elie areas between pits are called */and,*. A land is a smooth surface, which reflects hack at higher intensity. The change ⌠14.2t **wc.Qn** pits and lands is detected by a photosensor and converted into a digital signal. The sensor tests the surface at regular intervals-'['he beginning or end of a pit represents a 1; when no 6ange in elevation occurs between intervals, a 0 is recorded.

Recall that on a magnetic disk, information **is** recorded in concentric tracks. With the simplest constant angular velocity (CAN') system, the number of bits per track is constant. An increase in density is achieved with multiple zoned recording, in which the surface is divided into a number of zones. with zones farther from the center containing more bits than zones closer to the center. Ali hough ihis technique increases capacity, it is still not oplimal.

To achieve greater capacity. CDs and CD-ROMs do not organize information on concentric tracks. Instead. the disk contains a single spiral track. beginning near the center and spiraling out to the outer edge **of** the disk- Sectors near the outside cif the disk are the same length as those near the inside. Thus, information is packed **cven Ey across the** disk **in segments t.if** the same size and these are scanned al 1he same rate by rotating the disk at a variable speed. The pill are then read by thi2 laser at a constant **linear velocity (CLV)-** The disk rotates more slowly for accesses near the outer edge than tor those near. the center. Thus, the capacity of a track and the rotational delay both increase for positions nearer the outer edge of the disk. The data capacity for a CD-ROM is ghoul 6180 **Mb.**

Data on the CD-ROM are organized as a sequence of blocks. A typical block **format** is shown in Figure 6.11. It consists of the following fields:

- Sync The sync field identifies the beginning of a block. It consists of a byte of all Os. 10 bytes of all Is. and a byte of all Os.
- Header The header contains the block address and the mode byte. Mode 0 specifies a blank data field; mode 1 specifies the use of an error-correcting code and 2048 bytes of data: mode 2 specifics 2336 bytes of user data with no error-correcting code.
- Data: User data.
- Auxiliary: Additional user data in mode In mode .1., this is a 288-byte error-correcting code.

With the use of CLV, random access becomes more difficult. Locating a specific address involves moving the head to the general area, **ji djasting** the rotation speed and reading the address, and then making minor adjustments to find and access the specific sector.

C[]-ROM is appropriate for the distribution of large amounts of data to a large number of users. Because of the expense of the initial writing process, it is not appropriate for individualized applications. Compared with traditional hard disks, the Cl)-ROM has two advantages!

- The optical disk together with the information stored on it can be mass replicated inexpensively—unlike a magnetic disk. The database on a magnetic disk has to be reproduced by copying one disk at a time using **1,WC)** disk drives.
- The optical disk is removable, allowing the disk itself to be used for archival storage. Most magnetic disks are nonremo'ahle. The information on nonremovable magnetic disks must first he copied to tape before the disk drive/disk can be used to store new information.

The disadvantages of CD-ROM are as follows:

- It is read-only and cannot be updated.
- It has an access time much longer than that of a magnetic diSk drive, as much as lin I r a second.

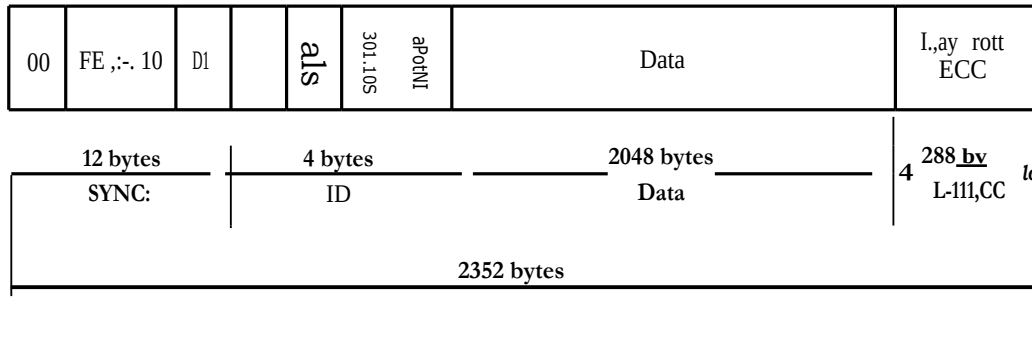


Figure 6.11 CD-ROM Block Formal

CD Recorcloble

To accommodate applie;itions in which only one or a small number of copies of a set of data is needed, the write-once read-many CD, known as the Cl) recordable (CD-R), has been developed. For CD-R, a disk is prepared in such a way that it can be subsequently written once with a laser beam of modest intensity, Thus, with a somewhat more expensive disk controller than for CD-ROM, the cuslomer can write once as well as read the disk.

The CD-R medium is similar to but not identical to that of a CD or CD-ROM. For CDs and CD-ROMs, information is recorded by Ihe pitting of the surface of the medium, which changes reflectivity. For a CD-R, the medium includes a dye laver. The dye is used to change reflectivity and is activated by a high-intensity laser. The resulting disk can be read on a CD-R drive or a CD-RO:VI drive.

The CD-R disk is attractive for archival 241 orage of documents and **il k!,** It provides a permanent record of large volumes of user data,

CD Rewritahle

1 he. CD-RW optical disk can be repeatedly written and overwritten, as with a magnetic disk. Although a number of approaches have been tried, the only pure optical approach that has proved attractive is called phase change. The phase change disk uses a material that has two significantly different reflectii,ities in iwt, different phase slates. There is an amorphous stale, in which the ' molecules exhibit a random orientation and which reflects light poorly: and a crystalline state, which has a smooth surface that reflects light well. A beam of laser light can change the material from one phase to the other. The primary disadvantage of phase change optiZ-2i1 disks is that the material eveniiiallv and permanently loses its desirable properties. Current materials can be used for between 500.000 and I ,000,000 erase cycles.

The CD, kW has the obvious advantage over CD-ROM and CD-R that it can be rewritten and thus used as a true secondary storage. As Such., it competes with Tnagnetic disk. A key advantage of the optical disk is that the engineering tolerances for optical disks arc much less severe than for high-capacity magnetic disks. Thus, they exhibit higher reliability and longer life.

Digital Versatile Disk

With the capacious digital versatile disk (DVD), the electronics industry has at last found an acceptable replacement for the analog VHS video tape. The DVD will replace the video tape used in video cassette recorders (VCRs) and, more important for this discussion, replace lhe CD-ROM in personal computers and servers. The DVD takes video into the digital a2e. It delivers movies with impressive picture quality, and it Call be randomly accessed like audio CDs, which 1.3V I) machines can also play. Vast volumes of (141ta can be crammed onto the disk, currently seven times as much as a CD-ROM. With DVD's huge storage. capacity and vivid quality, PC' games will become more realistic and educational software will incorporate more video. Following in the wake of these developments will he a new crest of traffic over the 'Memel and corporate intranets. as I his material is incorporated into Web sites.

'f'he DVD's greater capacity is due in three differences from CDs (Figure **6.12):**

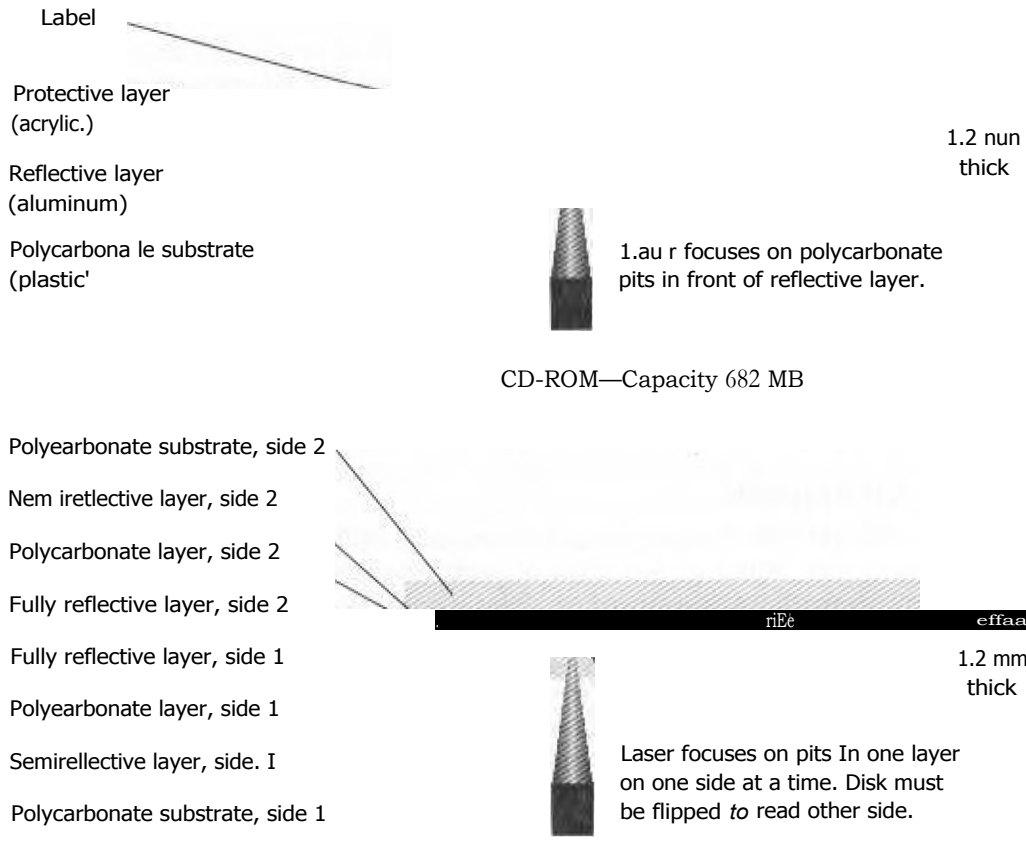


Figure 6.12 CD-ROM and DVD-ROM

1. Bits are packed more closely on a DVD, The spacing between loops of a spiral on a CD is 1.6 Arn and the minimum distance between pits along the spiral is 0.834 p.m. Thu 1)VD uses a laser with shorter wavelength and achieves a loop spacing of 0.74 p.m and a minimum distance between pits of 0.4 Am. The result of these two improvements is about a seven-fold increase in capacity, to about 4_7 GB.
2. The DVD employs a second layer of pits and lands on top of the first layer. A dual-layer DVD has a semiruflective layer on top of the reflective layer. and by adjusting focus, the lasers in DVD drives can read each layer separately. This technique almost doubles the capacity of the disk, to about 8.5 GB, The lower reflectivity of the second layer limits its storage capacity so that a full doubling is not achieved.
3. The DVD-ROM can be two sided whereas data is recorded on only one side of a CD. This brings total capacity up to 17 GB.

As with the CD, DVDs come in writeahic as well as read-only versions (Table 6.4).

6.4 MAGNETIC TAPE ,

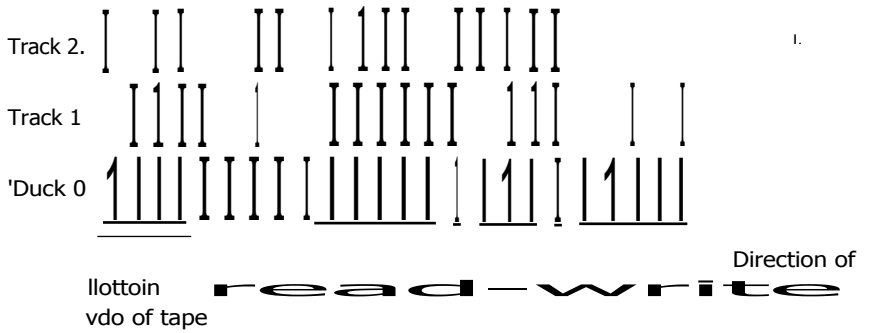
Tape systems use the same reading and recording techniques as disk systems. The medium is flexible polyester (similar to that used in some clothing) tape coated with magnetizable material. The coating may consist of particles of pure metal in special hinders or vapor-plated metal films. The tape and the tape drive are analogous to a home tape recorder system, Tape widths vary from 0,3S cm (0.15 inch) to 1.27 cm (0,5 inch), .tapes used to he. packaged as open reels that have to be threaded through a second spindle for use. Today, virtually all tapes are housed in cartridges.

Data on the tape are structured as a number of parallel tracks running lengthwise, Earlier tape systems typically used nine tracks, This made it possible to store data one byte. at a time, with an additional parity bit as the ninth track. This was followed by tape systems using 18 or :k6 tracks, corresponding to a digital word or double word. The recording of data in this form is referred to as **parallel recording.** Most modern systems instead use **serial recording,** in which data arc laid out as a sequence of hits along each track, as is done with magnetic disks. As with the disk, data are read and written in contiguous blocks, called *physical records,* on as tape. Blocks on the tape are separated by gaps referred to as *wrrecord* gaps. As with the disk, the **tape** is formatted to assist in locating physical records.

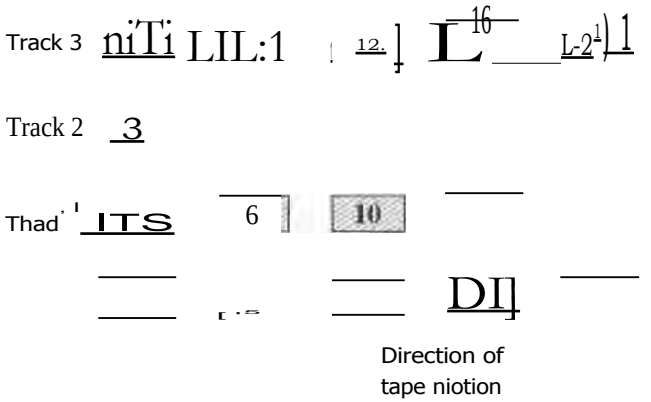
The typical recording technique used in serial tapes is referred **to** as **serpentine recording. in** this technique, when data are being recorded. the first set of bits is recorded along the whole length of the tape. When the end of the tape is reached, the heads are repositioned to record ii **new** ack, and the tape is again recorded on its whole length, this time in the opposite direction. That process continues, hack and forth. until the **Lap,: is full (Figure** 6.13a). To **increase** speed, the read-write head is capable of reading and writing a number of adjacent tracks simultaneously **(typically** 2 to 8 tracks). Data are still recorded serially along individual tracks, but blocks in .!',equence are stored on adjacent tracks, as suggested by Figure 6,13b. Table 6.5 shows parameters for **one** system. known as Dljnape,

Table 6.5 DLTiapo Urines

	DLT 4000	DLT WOO	SDLT 220
Capin:it)' ICB).	20	40	110
Bala rate INTRA)	1.5	6.0	I 1.0
Bit density (Kblart)	32.3	?1.6	51.6
Track densit). tifent)	1011	1(34	317
Media length IMO	549	549	5a9
Media width (ti-n)	1..27	1.2?	1,27
Number of tracks	I2		44I
Number of tracks read-write simultaneously	2	4	8



(a) Serpentine reading andliriting



(b) Block layout for system that reads-writes Nur tracks simultaiwotisly

Figure 6.13 Typical Magnetic Tap:. Pctaturcs

A tape drive is a *sequentivd- rcce.,s.s* device. If the tape hoad is positioned at record 1, then to read record N, ii is necessary to read physical records J. through A.' — 1, one at a time. if the. head is currently positioned beyond the desired record. it is necessary to rewind the tape a certain dirt nee and begin reading forward. Unlike the disk, the tape is in motion only daring a read or wrilc operation.

In con t rast to the tape, I he. disk drive is referred 10 as a *direct-access device.* A disk drive need not read all the sectors on a disk sequentially to get to the desired one, it must only wait for the intervening 2ic!.C1(11'5 within one track and can make. successive accesses to any traek.

Magnetic. tape was the first kind of secondary memory. It is still widely used as the loweAt-cast, slowest-speed member of the memory hierarchy.

6.5 RECOMMENDED READING AND WEB SITES

[M.F.E90a] provides a good survey 0r the underlyinlvt qii ding technoloy Ft tape systems. [MEE96b]focuses on the data storage techniipics for disk and tape systems. [COMEX)] is a short but instructive article on current treni Is in magnetic disk storage technology.

An excellent survey of RAID techni10,2, v,rit ten by the inventors of the RAID concept. is ICHEN94]. A more detailed disci•ion is published by the RAID Advisory Board, an association of suppliers and consumers or RAID-related products [MASS97]. A good recent paper is [FR 11 ‘.961.

[MARC901 an excellent overview of the optical storage field. A good survey of the underlying IL:cording and reading technology is [MANS97],

FROSC991 provides a comprehensive. overview of all types or external memory systems. with a modest amount or technical detail on each IKHUR011 is another good survey.

CIIEN94 Chen, P.; Lee, **E.; CiihsoR,** O.; Katz. R.; and Patterson, D. "RAID: High-Performance. Reliable Secondary Storage, r1 (31 *Computing Surwys,* June 1994.

COME00 Comerford, R. "Magnetic Storage: The Medium that Wouldn't .Die," *IFEE Spectrum,* **Dc.cember** 2000.

FRW96 Friedman, M. RAID Keeps Going and Going and *"IEEE Spectrum..* April 19%.

KIJUR01 Khtu shudov, A, *The Esseirtiol Guide to Computer flaw* Siotage. t Ipper Saddle River, NI Prentice Hall, 2001.

Al A NS97 Mansuripur,.M., and Sineerbox, 0, 'Principles and Techniques of Optical Data Storage." *P•ocerelin,t. or A- IEEE.* November 1997.

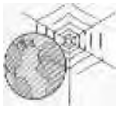
r+Ls.RC90 Marcham, A. *Optical .R{fearYfing.* Readin, MA: Addison-Wesley. 1990.

MASS97 Massiglia, P. *The RAID Book: A Srorage Syslern Tr:•hnology Ilemahook.* St. Peter, MN: The. Raid Advisory Board, 1997.

MEE96a Mee, C., and E. eds. *Magneik Recording Technology,* New York: McGraw-Hill, 1tt96.

ME .961) **Mee.** C., and Daniel. F. eds. Afrignetic Soyfogr *ilandbook.* New York: McGraw-Hill, 19%.

ROS094 Rosch, 'W. *Vt•ieur L. Ro.seh Kurth are Bible, Indianapolis,* IN: Sams, 1999.



Recommended Web Sites;

- **RAID Advisory Group:** RAID industry group. Information about RAID technology and products.
- **Optical Storage Technology Assodation:** Good source of information about optical storage technology and vendors, plus extensive list of relevant links.
- **DI,Ttapc:** *Good* collection or technical information and links to vendors,
- **Data Storage Magazine:** The magazine's Web site contains extensiv.2. information on data storage products and vendors.

6.6 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

access time
CL)
CD-ROM
CD- R
CD - RW
constan[ar riglllar velocity (CA V)
cOtiStaill linear velocity (CI.,V)
cylinder
DVD
DVD-ROM
DVD-R
DV D RW
fixed-head disk
floppy disk
gap
head
land
magnetic disk
maguerie tape
magnetoresistive
movable-head disk
multiple zoned recording
nonremovablc: disk
optical memory
pit
platter
RAID
removable disk
rotational del ʜy
sector
seek time
serperiiitte recording
striped data
substrate
track
transfer time

Review Questions

6.1 What are the advantages of using a glass substrate for a magnetic disk?
6.2 llow are data written onto a magnetic disk?
63 Haw are data read from a magnetic disk?
6.4 Explain the difference between a simple CAV system and a multiple zoned recording system.
6i Define the terms *track, cylinder,* and *sector.*
6.6 What is the typical disk sector size?
6.7 Define the terms *.seek* rime, *rearriiional rfetur. decess rime,* and *transfer time.*
(i.8 What common characteristics are shared by all RAID levels?
69 Briefly define the seven RAN) levels,
6,10 Explain the term *striped data.*
6.11 How is redundancy achieved in a RAID system?
6.12 In the context of RAID, what is the distinction between parallel access anti independent access?
6.13 What is the difference between CAV and CLV?
6.14 Whtat differences between a CD and a DVD account for the larger capacity of the latter?
6.15 Explain serpentine recording

Problems

6,1 Consider a disk with N tracks numbered from 0 to (A - 1) and assume that requested sectors are distributed randomly and evenly over the disk. We Want to calculate the average number of tracks traversed by a seek.

a. First, calculate the probability of a seek of length j when the head is currently posi• honed over track *t. Him:* this is a matter of determining the total number of combinations. reeogni2ing that all track positions for the destination of the seek are equally likely.

h. Next. calculate the probability of a seek of length K_ *Hint* this involves the summing over all possible combinations of movements of K tracks.

c. Calculate the average number of tracks traversed by a seek. using the formula for expected value

$$= Ei \quad -$$

Hint Use the equalities: $Ei \quad \frac{n(n — 1)}{2} \qquad n(n —)(2n + I)$

d. Show that for large values of N. the average number of tracks Traversed by a seek approaches

6.2 Define the following for a disk system:

t • seek time average time to position head over track
r — rotation speed of the dikk, in revolutions per second
n — number of hits per sector
N = capacity of a track,.in bits
= time to access a sector

Develop a formula for r_{n} as a function of the other parameters.

6.3 Assume a I0-drive RAID configuration, Fill in the following matrix, which compares the various RAID levels:

RAID Level	Storage Density	**Bandwidth Performance**	Tra **nsaction Performance**
1			
2			
3			
5			

Each parameter is normalized to the RAID level that delivers the best performance; therefore, the remaining numbers in the matrix should have a value between 0 and 1. Storage density refers to the fraction of disk storage available for user data. Bandwidth performance reflects how fast data can be transferred out of an array. Transaction performance measures how many operations per second an array can perform.

6.4 It should be clear that disk striping can improve data Transfer rate when the strip size is small compared to the 110 request size. Ii should also be clear that RAID II provides improved performance relative to a single large disk, because multiple 110 requests can be handled in parallel. However, in this latter case, is disk striping necessary? That is, does disk striping improve 11/0 request rate performance compared to a comparable disk array without striping?

CHAPTER

INPUT/OUTPUT

7.1 Lxternal Devices
Keyboard/ Monitor
Disk Drive

7.2 I/O Modules
Module **Function**
IjO Module Structure

7.3 Programmed VO
Overview
Commands
10 Instructions •

7.4 Interrupt-Driven I/O
Interrupt Procesing
Design Issues
Intel Ii 2C59A Interrupt Controller
The Intel g2C55A Programmable Peripheral Interface

7.5 Direct Memory Access
Drawbacks of Programmed and Interrupt-Driven [10
.D:VIA Function

7.15 **I/O Channels and Processors**
The Evolution of the 1/0 Function
Characteristics of Channels

7.7 The External Interface: FireWire and Infinitland
Types of interfaces
Point-to-Point and MuItipoint Configurations
FireWire Serial Bus
InfiniBand

7.8 Recommended Reading and Web Sites

7.9 Key Terms, Review Questions, and Problems
Key 'Ft:1ms
Review Questions
Problems

KEY Pf)INTS

- The computer system's 10 architecture is its imerfacc to the outside world. This architecture is designed to provide a systematic means of controlling interaction with the outside world and to provide the operating system with the information it needs to manage 1:'(.). activity effectively.
- The are three. principal I/O techniques: **progranmied I10, in** which I/O occurs under the direct and continuous control of the program requesting the I.10 operation; **interrupt-driven 1/0,** in which a program **issues** an 110 command and then continues to execute, until it is interrupted by the I/O hardware to signal the end of the operation: and **direct memory access (DMA),** in which a specialized [10 processor lakes over control of an 1/0 operation to move. a large block of data.

Two important examples of external 1/0 interfaces are **FireWire and lallnilmnd.**

In addition to the processor and 4i set of memory modules, the third key element 0r ,., computer system is a set of 110 modules, Each module interfaces to the system bus or central switch and controls. one or more peripheral devices. An 1/0 module is not simply a set of mechanical connectors **that wire a device into the system bus. Rather.** the. 110 module contains some "intelligence"; that is, it contains logic for performing a communication function between the peripheral and the bus.

The reader may wonder why one does not connect peripherals directly to the system **bus. The reasons** are as follows:

- There are a wide variety of peripherals with various methods of operation. It would he impractical to incorporate the necessary logic within the processor to control a range of devices.
- The data transfer rate of peripherals is often much slower than that ()I' the. memory or processor. Thus, it is impractical to use the high-Speed system bus to communicate directly with a peripheral,
- On the other hand, the data transfer rate of some peripherals is faster than that of the memory or processor. Again, the mismatch **would lead to inefficiencies** if not managed properly.
- Peripherals often use different data formats and word lengths than the computer to which they are attached.

Thus, an I/O module is **rcquired. This module** has two major **functions (Figure 7.1):**

* **Interface to** the processor and memory via Ihe system bus or central switch
- Interface to one or more peripheral devices by tailored data links

We begin this chapter **with a brief discussion of external devices, followed** by an overview of the structure and function of an Ii0 module, **Then we look at the** various ways in which the 110 function can be performed in cooperation **with**

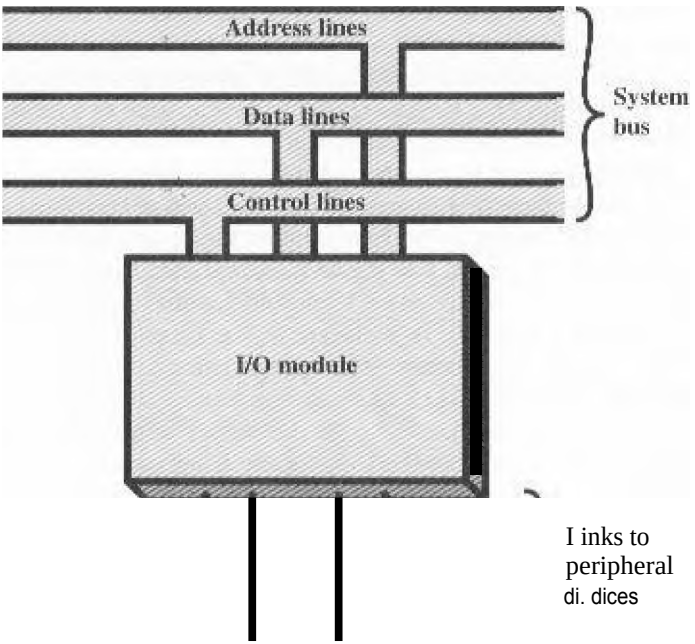


Figure 7.] Generic Model *of* an 110 Module

the processor and memory: the internal 110 interface- Finally. we 4: N.,d ine the external 110 interface. between the VO module and the outside world.

7 1 ,EXTERNAL DEVI

openations arc .1ccomplished through a wide assortment of external devices that provide a means of exchanging data between the external environrnc:mt and the computer. An external device attaches to the computer by a link to an 110 module (Figure 7.1). The link is used to exchange control, status, and data between the 110 module and the external device. An external device connected to an I/O module is often referred to as a *peripheral device* or. simply, a *pffipiwra*

We can broadly elassiry external devices into three categories

- **Human readable:** Suitable for communicating with i he computer user
- **Machine readable:** Suitable for communicating with equipment
- **Communication:** Suitable For Communicating with remote devices

Examples of human-readable devices are video display terminals (VDTs) and printers. Examples of machinc re;idable devices are magnetic disk and tape systems, and sensors and actuators, such as are used in a robotics application. Note **that** we are viewing disk and tape system s as I/O devices in this chapier, whereas in Chapter 6 we viewed them as memory devices. From a functional point of view. these devices are par1 of the memory hierarchy, and their use is appropriately discussed

in Chapter t5, From a structural point of view, these devices are controlled by LIO modules and are hence to be considered in this chapter.

Communication devices allow a computer to exchange data with a remote device, which may be a human-readable device. such as a terminal, a machine-readable device, or even another computer.

In very general terms, the nature of an external device is indicated in Figure 7.2. The interface to the 1/0 module is in the form of control, data, and status signals. *Control signals* determine the function that the device will perform, such ati send data to the I/O module (INPUT or READ), accept data from the I/0 module (OUTPUT or WRITE.), report status. or perform some control function particular to the device (e.g., position a disk head). *Data* are in the form of a set of hits to be sent to or received from the 110 module. *Sratery signet's* indicate the state of the device. Examples are READY/NOT-READY to show whether the device is ready for data transfer.

Control leqic associated with the device controls the device's operation in response to direction frorn the 110 module. The *transthicer* converts data from electrical to other forms of energy during output and from other forms to electrical during input. Typically. a buffer is associated with the transducer to temporarily hold data being transferred between the I/O module and the external environment; a buffer size of g to L6 bits is common.

The interface between the I/0 module and the external device will be examined in Section 7.7. 'The interface between the external device and the. environment is beyond the scope of this book, but several brief examples arc given here.

Keyboard/Monitor

The most common means of computer/user interaction is a keyboard/monitor arrangement. The user provides input through the keyboard. This input is then trans-

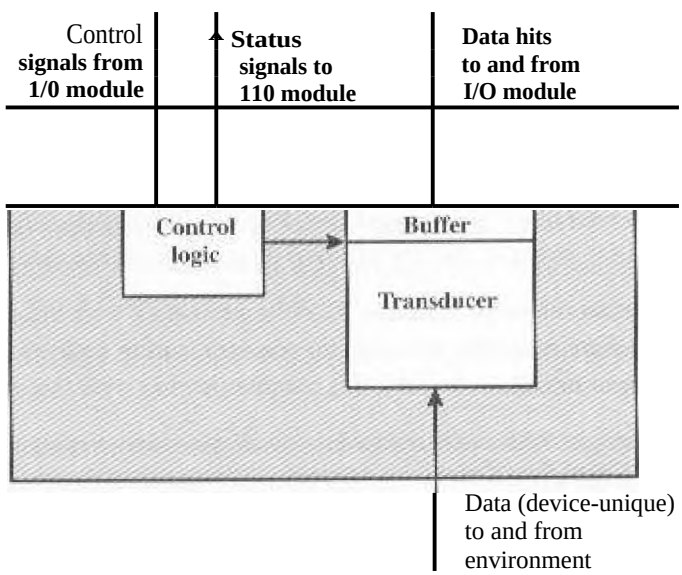


Figure 7.2 Block Diagram of an External Device

Table 7.1 'Iite International Reference Alphabet (IRA)

hii posilion															
	b,			#I	0	0	0	1	t	1	1				
	h,.			a	1l	1	1	L)	Il	1	1				
		b.		0	1	0	1	0	I	0	1				
b,1	h,	h.													
0	0			NI:I.	DLE.	SP	0	itF	P		P				
as	a	ii	I	SOH	DC1	!	1	A	O	:1	Li				
11	0	I	0	STX	DC2	9.	2	ii	R	II	r				
0	0	I	I	ETX	DO	is	2,	(9	S	c	s				
0				t'.QT	DC4	$	4	D	T	d	l				
	1			EICQ	NAK	%	5	E		e	11				
0	1	1	0	ACK	SYN	&	6	F	N·	f	v				
0	1	1	I	BEL	ETR		7	U	W	g	sii				
3	tj	li	0	BS	CANT	(	S	H	X	lt	x				
I	0	0		HT	EM	)	9	I	Y	i	y				
I	0	I	0	LF	SI YR		!	.1	Z	i	z				
I	0	I	I	V1	ESC	–		K	(	k					
I	I		il	FF	FS	9		L	.	I					
I	t	11	[	CR	CS	-	=	M	]	m					
I	L	I	0	SO	RS	.,		14;;	···	n	-				
I	1	I		S1	US	r	.,)	O	_	0	DEL				

milted to the computer and may also he displayed tin the monitor. In addition, the monitor displays data provided by the computer.

The basic unit of exchange is the character, Associated with each character is a code, typically 7 or 8 bits in length. The most commonly used text code is ihe International Reference Alphabet (IRA).' Each character in this code is represented by a unique 7-bit binary elide: thus, 128 different characters can be represented. Table 7.1 lists all of the code values. In the table, the bits of each character are labeled from b,. which is the most significant bit, to b, the least significant bit. ⁼ Characters are of two types: printable and control (Table 7.2). Printable characters are the alphabetic. numeric. and special characters that can be printed on paper or displayed on a screen. For example, the hit representation of the character "K" is b,b,b,b$_A$bJ),b, = 1001.011. Some of the control characters have to do with

---

'IRA]s defined in 1TU-T Recommendation T.50 and was formerly known as international Alphabet Number 5 (IA51. The U.S. national version of IRA is referred to as the American Standard **COLL** for Information Interchange (AKIO,

'IRA-uncodell characters are almost always stored and transmitted using' bits per character. The eighth hit is a parity bit used for error detection, The parity hit is the most significant hit and Is therefore labeled

Table 7,2 IRA Control Characters

Farm' Control

▪ (Back....:piEcc): Indic:11.es rno...ernenk of the printinE4 mechanism or display *rurm)r 1¯ raCk* afd one position.

(Horizontal Lab): Indicacm. movement **Or** the printing mechanism or displq W1'30.1' forward 10 the ne.xi prassigried ¯tab' or stopping juisilion.

IF (Line feed): Indicates rnovermi.nt of the printing nteehanisin cir display cursor to the start rat the lieu line.

ıyercieal Loh): Indicates Mos:emelt:I or the printing mothuniS.M. or display Qtn¯Nor LIle 'Lux!. 01 *a* Series preassigned printing lines.

FF (TormTe.ed.): Indicates Inovc.ment **{if** the priming mechanism or dis•plAy cursor to the starling position of the nE.N.1 page... Enrol Or Kre.2n.

CR (Currii4w 111ClicA.Ln. **LLSL.IVI2ITLeILL of aid** MCC h.ani Lan **{5r** cursor In !he trartin pOs ion. of 11.1e 5.41he line.

Trnnsrnissi**an Cord ra**

SOH (Stall or .h.:114,1ing.l: laud to indicate 1h' slut 1)1 ıı lwuding, which may eonlain **TrAiLLESA LEIfIMMOLtiCITI.**

SI'X{Start of Lem)! 1:sed to indicate the start of the Lext.and so also inthea.ces the.c.nd of the heading,

✓ iX ;Ella of text.): Used to terminate the le.xl 'that was started with SIX.

On (End ni minsmimion.): Indicates 1E1e end cif u IrAns **Ss** 11)11., may haw.: included am: or MOH! --, ems¯ with thi:ir heatfins..

LNQ (J¯ riguiry i: A leyuest for a **! N MI** rurrioLe station. ft may be used as ,1 ¯WHO ARE y0.¹1.;" request icir a station ¢O idenEiry

ACK (A cknowleilvy A charaemr LrunsMiitell by receiving device 4.6 **An** affirmation response Li) **H** IL is ur•ed us a positive **!VEP45r1IIC** 1.¹3 1 ¹**1-4Eing.**

NAK acknowledgment): Achnructer 1.37.1r"..niilied h'y a receiving dievice. as an negative response to n sender. It i used as i negniive response Lt.) messages.

S ! SViiehrOr§OLIS:K110: Used by a synchronous **to athieNCI N:!..114; WEICIL ncr da1.4.1.** arc being **Nerd., 1.1** cransrnis.sion syslent may send SIN diameters eonhnuOurdy.

ETR (End or iransmis:-iion Nock): Indicales Ilse end or a Hock of data For enntrnunication purpow.s, is used lot bloc.king data where the 11 s..D.Leau¯e is **1101 TIECEssaril'y** related co the procifs-sing Formes,

formation Separator

FS (File xriara tor)
G S (Group r`epiiraLor)
RS {Record separator)
US ? United sepurator)

lol ŧıI **Li.,** be used ill an oplionHI manner exccTit that their hierarchy **RhHil** he FS ()he room inclusive) to 1..:S Rim luat iriclusive/

Mistvlidne0116

NUL (Mill): No charactur. l.. st ci For ii ILLeg in Of Filling spac.r,. on ! **pC** ·.**WELC LI'.I** ıı2 re are no

B EL (1 LI1): 1:sed When there IL} c iIL **huirta ;:ittendoli.** It rna y c4.11¯ıXı¯ o] **ALLIrm err katanion devices.**

NI) (shirt out): intliudLos **that** the code **comibllati tans that** fallow shall be interrircle4 us outside of **the** standard character **SCL Hail** a SI character is reached.

SI (Shift in): Indicalei, the Ilae codo uornbinations rh.oc follow shill be interpreted according lo L1¯1l2 character se!.

oF,J,)icicic): Used **Lu** 4ibliwrate unwarticd characters: 11.)3 ·..**xarnple,** by ovcrwrilins.

(spoc.o: **A !Ion prmling character** Used in iicTiaraLe words, InLo iiLUVe the prinking much:mi.= or display c ursor lotw:ircl by c1.11.0

I MF (1)LiLki bark esca pc); A E ham ccer that sh II change rhC DIcflrum 5 of one or MOM con Liguc u folic i ng. eharaecers. It can provide supplementary con rols. or perm t5 the sanding of date characters **h.:Iving ;in** y bit Combination,

DCX DC4 (I¯MviLu controls): Chnrnetuni. fur the coairol of ancillary de icc s or 4peciiil c3rni **nal** auires.

CAN (Cancel): indicareA diet **Lire** data than prccede it **Ell H** mess *age* or block shoukl b. disregarded (tmually because. an caul likay been del.c..esed

• (**bid** of medium)! indicates the ph riica I end of tripe c.ir other MediuM, or LI¯iu end of Elle requarea us..7.(1 **porhan of** tEhr Lued

S :IS ; .S ubstit **u Lo). Su** bsti.ltkii.:41 liar a character that is io Li nd to h4 *c* tron eouS or invalid.

i:1⁹.NEVEJ: A eharaccer inl **ende.t1** Lu **Provide** eotic ex **Len** lion in thal ii **OVEN LI NI.veified** number DC continuously (olloveing characters en N.I.Lurnate,

controlling the printing or displaying of characters; an example is carriage return. Other control characters are concerned with communications procedures.

For keyboard input, when the user depresses a key_ this generates an electronic signal that is interpreted by the transducer in the keyboard.and translated into the hit pattern of the corresponding IRA code. This bit pattern is then transmitted to the 110 module in the computer. At the computer, the text can be stored in the same IRA code. On output, IRA code characters arc transmitted to an external device from the 110 module. The transducer at the device interprets this code and scuds the required electronic signals to the output device either to display the 'indicated charact er or perform the requested control function.

Disk Drive

A disk drive contains electronics for exchanging data, control, and status signals with an I/O module plus the electronics for controlling the disk read/write mechanism. In a fixed-head disk, the transducer is capable of converting between the magnetic patterns on the moving disk surface and bits in the devices buffer (Figure 72 A moving-head disk must also be able tc i **caLise** the disk arm to move radially in and out across the disk's surface.

7.2 110 MODULES

Module Function

The major functions or requirements for an I/O module fall into the following categories:

- (.7ontrol and timing
- **a Processor** communication
- Device communication
- Data buffering
- Error detection

During any period of time. the processor may communicate with one or more external devices in unpredictable patterns, depending on the program's need for I/O. The internal resources, such as main memory and the system bus, must be shared among a number of activities, including data 110. Thus, the 1/0 function includes a **control and timing** requirement. to coordinate the flow of traffic between internal resources and external devices. For example. the control of the transfer of data from an external device to the processor might involve the following sequence of steps:

1. The processor interrogates the I/O module to check the status of the attached device.
2. The 110 module returns the device status.
3. if the device is operational and ready to transmit, the processor requests the I transfer of data, by means of a command to the 110 module,
4. The I/O module obtains a unit of data (e.g,, 8 or 16 bits) from the external device.
5. The data are transferred from the l/O module to the processor.

11 !he sysWrn LTn.ploys a bus. then each of the interactions between the processor and the I/O module involves die or more bus arbitrations.

The preceding simplified scenario also illustrates that the I/O module musk communicate with the prcwe..4sof and with the external device. Prneessor committal• cation involves the following:

- Command deeoding: The I/O module accepts ec.mim;inds from the processor, typically seat as signals on the control bus. For example, an 1/0 module For disk drive might accept the following, commands: READ SECTOR, WRITE SECTOR, SEHK track number, and SCAN record III The latter two comininds each inciude a parameter that is sent on the data bus,
- **Data:** Data kire exchanged between I.hc processor and the I/O module owl' the data bus.
- Stsifin reporting: [[ecriuse peripherals are so slow, it is important to know the status of the 1/0 module, For example, if an **I/O** module is asked to send data to the processor (road). it may not he ready to do so because it is still working on the previous I/O comrmind. This fact can he reported with a status signal. C:ornmvn statuN ?ignals are BUSY and READY. There may also he signals to report various error conditions.
- * Address recognition: Just as each word of memory has an address. so does each Ii0 device, Thus, an [I0 module must recognize one unique address for each peripheral it controls.

On ghee olhcr sick. the I/O module must be able 10 perform device **COMM**. ['legion, This communication involves commands, status information, and data (Figure 7.2).

An essential task of an I/O module is **data buffering.** The need I'm- this fan c-lion is apparent from Figure 7.1 Whereas the transfer rate into **and** out of main memory or the processor is quite high, the 1'4 W is orders of magnitude lower for many peripheral devices and covers. a wide. range. Data coming from main memory are sent to an P.O moduli: in a rapid burst. The data are buffered in the I/O module and then sent to the peripheral device at its dala rate. In the opposite direction, dati are buffered so as not lo tic up the memory in a slow transfer operation. Thus, the I/O module moat he ihie to 011i:tate at both device and memory speeds. Similarly, it` the I/O device operates at a rate higher than the memory access rate. then the I10 [nodule performs the needed buffering operal ic n.

Finally, an 1/0 module is often responsible for **error detection and** for subsequently reporting Q!'rors to the processor. One class of errors includes mechanical and electrical inalfuncliom reported by the device (s,,g,, Nper jusr, had disk track). Another class consists of unintentional changes to the bit pattern as it is transmitled from device to I/0 module. Some form of error-detecting code is Often used t0 detect transmission errors. A simple example is the use of a parity hit on each character of data. For example, the IRA ch.:wader code occupies 7 bits of a by1e. The eighth hit k mi so Ebel Ihe total number of Is in the byte is even (even parity) or odd (odd pariiy). When a byte is received, the **I/O** module checks the parity to determine whether an error has occurred.

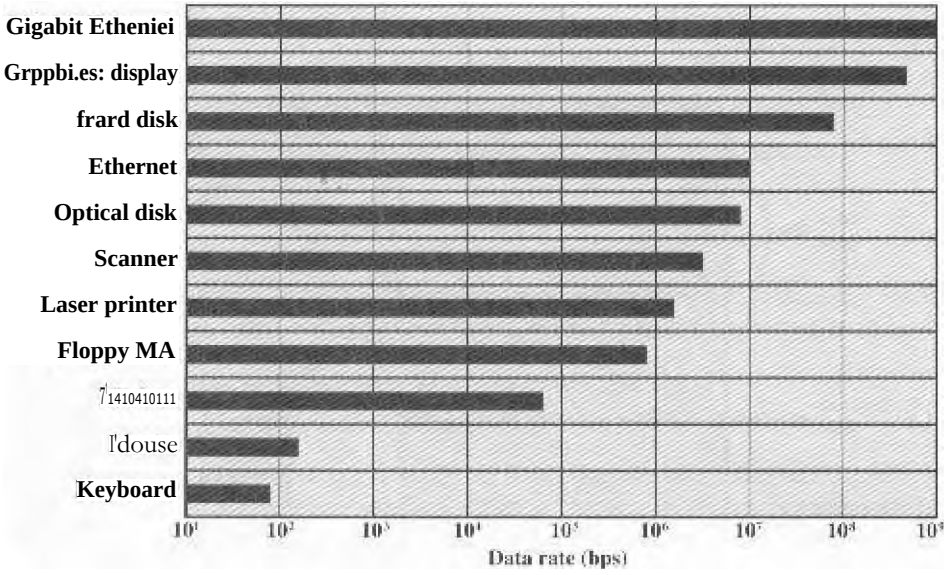


Figure 7,3 Typical I10 Device lilts. Rates

I/O Module Structure

1/0 modules vary considerabl!, ' in complexity and the number of external devices that dicy control. We will attempt only a very general description here. (One specific device. t he **inicl** 82C55A, is described in Section 7.4.) Figure 7.4 provides a general block diagram of an 110 module. The module connects **10** the rest of the computer through a set of signal lines (e.g., s!, ' stern bus lines). Data transferred to and from I he module are buffered in one or more data registers. There may also be one or more slaws regisicrs aro provide current status information. A status register may also function as a control register, to accept detailed control information from the processor. The logic within the module interacts with the processor via a set of control lines. '11 te processor uses the control lines 10 iSSI.PC commands to the 110 module. Sonic of the control lines may he used by the I/O moduic (e.g., for arbitration and status signals). The module must also be able to reco gnize and generate addresses associated with the devices it controls. Each 1/0 modu]e has a unique addres..s or, if it controls more than one external device, a unique set of addresses. Finally. the 1/0 module contains logic .specific to the interface with each device that it controls.

An 1/0 module Funclions to allow Ihe processor to view a wide range of devices in a simple-minded way. 'There is a spectrum of capabilities that may he provided. '['he I/O modu]e may hide the details of timing, formats, and the electromechanics of an external Llevice so that the processor can function in terms of simple read and write commands, and possibly open and close file cornmandq. In its simplest form, the I/0 module may still leave much of the work of controlling a device (e.g·. rewind a Tape) visible to the processor.

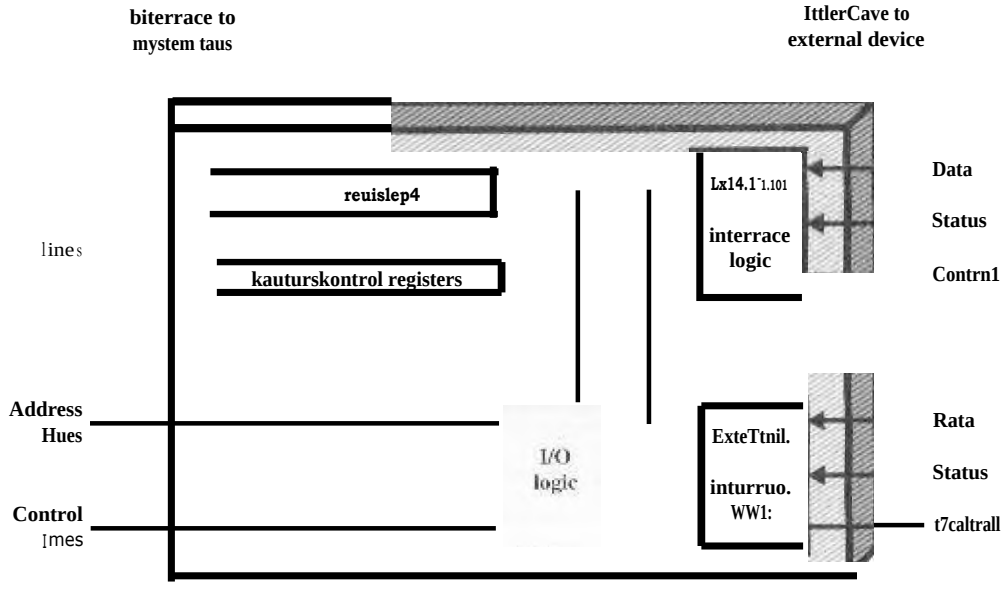


Figure 7.4 Block **Diagram of an 1/0 Moduk**

An 11 module that lakes on most of the detailed processing burden. pi esenting a high-level interface to the processor, is usuillly referred to as an *110 channel* or *1.10 processor,* An I10 module that is quite primitive and requires detailed control is usually referred to as an *110 controller* or *device coniroilcr. 1.10* controllers are commonly seen on microcomputers. whereas 110 channels are•used on mainframes.

In what follows, we will use the generic term *1/0 modale* when no confusion results and will use more specific terms where necessary.

7.3 PROGRAMMED I10

Three techniques are possible for 1/0 operations. With *programmed I/O,* data are exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the 1/0 operation, including sensing device status, sending a read or write command, and transferring the data. 'Mien the processor issues a command to the 1/0 module, it must wait until the 110 operation is complete. if the processor is lamer than the 110 module, this is wasteful of processor time. With *interrupi-driven 1/0,* the processor issues an **110** command, continues to execute other instructions, and is interrupted by the I/O module when the. latter has completed its work. With both programmed and interrupt 110, the processor is responsible for extracting data from main memory for output and storing data in main memory for input. The alternative is known as *direct memory access* (DMA), In this mode. the 110 module and main memory exchange data directly, without processor involvement.

Tokple 7.3 1.0 Techniquc.s

	No Interrupts	Use of Interrupts
1/0-111.-rnumipry Iran ter i hrough promisor	Prograrrima. 1.0	In **2rrupt-ciri VC)**
hired 1/0-to-Tnernne¢ transfer		Direa mcmory accoss {DMA)

Table 7.3 indicates the relationship among these three techniques. In [his section. we explore programmed 1/0. Interrupt 110 and DMA are explored in the following 1 wo sections, respectively.

Overview of Programmed I/O

When the proi:essor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module. With programmed I/O, the I/0 :nodule will perform the requested action and then set the appropriate bits in the 110 status register (Figure 7.4). The I10 module takes no further action to alert the processor. In particular, it does not interrupt the processor. Thus. it is the responsibility of the processor periodically to check the status of the I/O module until it finds that the operation is complete,

(·0 explain the programmed 110 technique, we view it first from the point of view of the I/O commands issued by the processor lo the PO module, **and then jrorio** the point of view of the I/O instructions execute'd.by the processor-

I/O Commands

To execute an I/0-related instruction. the processor issues an address, specifying the particular 110 module and external device, and an 110 command. There are four types of I/O commands [hal an I/O module may receive when it is addressed by a processor:

- Control: Ned to aeliv,a Le a peripheral and tell it what to do. For example, a magnetic-tape unit may he instructed to rewind or to move forward one record. These commands are tailored to the particular type of peripheral device.
- **Test: Used to test various sLittts C onditioos associated wit** h an I/O module and its peripherals. The processor will want to know that the peripheral of interest is powered on and available for use. It will also want to know if the most recent I/O operation is completed and if any errors occurred.
- Read: Causes the 1/0 module to obtain an item of data from the peripheral and plaice it in an internal buffer (depicted as a data register in Figure 7.4). The processor can [hen obtain the data item by requesting that the I/O module place it on the data bus.
- Write: Causes the 1.10 module **R**t take an item of data (byte or word) **From** lhe data bus and subsequently transmit that data item to the peripheral.

Figure 7.Sa gives an example of the use cif programmed I10 10 read in a block of data from a peripheral devic„e t **record from** tape) into memory. **Data are**

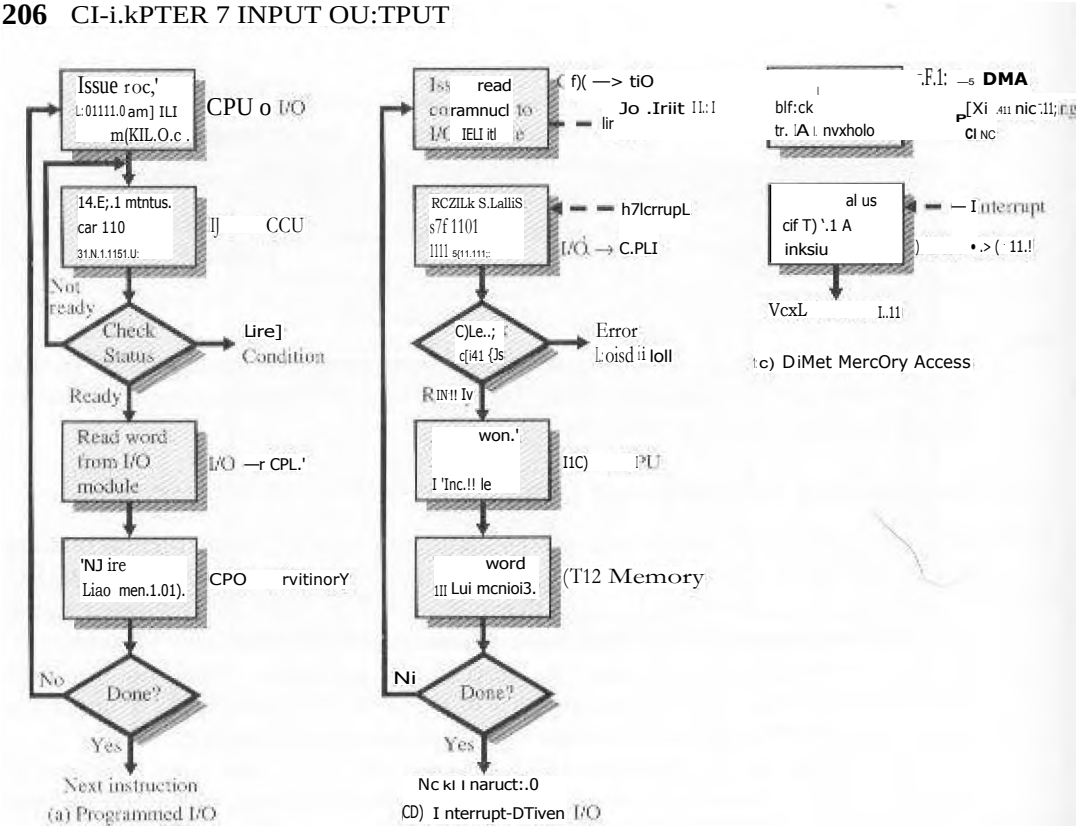


Figure **7,5** Three TechnicEtturi Input of a Block of DaLI

read in one word *(e.g..* 16 bits) at a tirnc. For each word that is read in, the processor nau;711 remain in a statils-chQ.eking cycle until it Lletermines dial the word is available in the I/O modules data register. flowchart highlights the main di--advantage of this technique: it is a time-consuming process Ihai keeps the processor busy needlessly.

I/O Instructions

With programmed 110, there is a close correspondence between the 1/0-related instructions that the proccssor fetches from memory and the I/O eornmands that the processor issues 10 an 110 module to execute the instructions. 'That is. the in.s1ruelions are emily mapped into 110 commands, and there is often a simp[c one-to-one relationship. The form of the instruction depends on the way in which external devices are addrf:Ased:

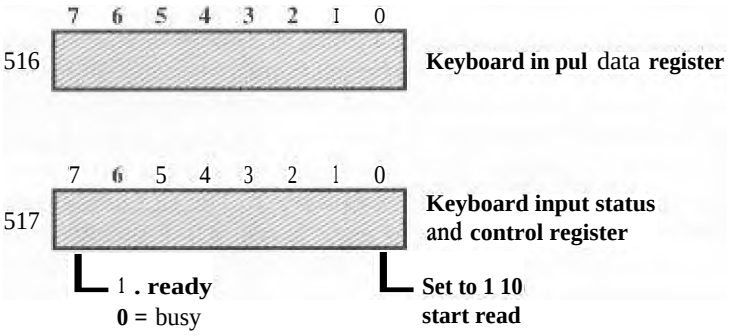
Typically, there. will be many [10 devices connected through 110 modules to the.vstern. Each device is given a unique idenlifier or address. When the processor issues an 110 commend. the cornmand Exn Wins the address or the desired th.vice. Thus, mach I/O (nodule must inLCrl3ret the address lines to determine if the command is [or itself.

When the processor, main memory, and 1.10 share a common bus, two modes of addressing re possible: memory mapped and isolated. With **memory-mapped**

I/0, there is a single address space for memory locations and I/0 devices, The processor treats the status and data registers of 1'(modules as memory locations and uses the same machine instructions to access both memory and 1.1r) deviee:71, So, for example, with 10 address lines, a combined total of 2'[11] =1024 memory locations and I.10 addresses can be supported, in any combination,

With memory-mapped 170, a single read line and a single. write line are needed on the bus. Alterriatiyel!,', the bus may be equipped with memory read and write. plus input and output command lines. Now, the corm nand line specifies whether the address refers to a memory location or an **device.** The full range of addresses may he available **for both.** Again, with IC) address lines, the system may now support both 1024 memory locations and 1024 LT_) addresses. Because. the. address space for 110 is isolated from that for memory, this is rererrc...d 10 as **isolated1/0.**

Figure 7,6 cc ritrasis these two programmed I/O techniques. Figure 7.6a shows how the interface for a simple input device such as a terminal keyboard mighi appear to a programmer using memory-mapped I/O. Assume a 10-bit address, with a



ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	1Aiad AC	,1,,	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	571	Co. slataf, byte
	Kranch if Sign i 0	202.	Loop until ready
	Load AC	516	Load data byte

la) Memory-mapped 110

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	I,nad I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Read*	201	Loop until complete
	In	5	Load dala byte

(b) Isolated I/O

Figure **7.6 Mc.rtiont-Maptx..1J** and 1'**..otaced**

512-bit memory (locations 0-511) and up to 512 I10 addresses (locations 512-10231. Two addresses are dedicated to keyboard input from a particular terminal, Addres 5l6 refers to the data register and address 517 refers to the status register, which also functions as a control register for receiving processor commands, The. program shown will read 1 byte of data from the keyboard into art accumulator register in the processor. Note that the processor loops„until the data byte is available.

With isolated I10 (Figure 7.617), the I/O ports are accessible only by special M commands, which activate the 1/0 command lines On the bus.

For most types of processors, there is a relatively large set .of different instructions for referencing memory, If isolated I/O is used, there are only a few I/O instructions. Thus. an advantage of memory-mapped 110 is that this large repertoire of instructions can he used, *allowing* more efficient programming. A disadvantage is that valuable memory address space is used up, Both memory-mapped and isolated are in common use.

7.4

The problem wit programmed 110 is that the processor has to wait a tong time For the I/O module of concern to be ready for either reception or transmission of data. The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result, the. level of the performance of the entire system is severely degraded.

An alternative **is** for the processor to issue an 110 command to a module and then ao on to do some- other useful work. The 110 module will then interrupt the processor to request service when it is ready to exchange data with the processor. The processor then executes the data transfer, as before, and then resumes its former processing.

Let us consider how this works, first from the point of view of the I/O module. For input, the module receives a READ command from the processor. The LO module then proceeds to read data in from an associated peripheral. Once the data are in the module's data register, the module signals an interrupt to the processor over a control line. 'the module then waits until its data are requested by the processor. When the request is made, the module places its data on the data bus and is then ready for another I10 operation.

From the processor's point of view, the action for input is as follows. The processor issues a READ command. lt then goes off and does something else (e.g., the processor may be working on several different programs at the same time). At the end of each instruction cycle, the processor checks for interrupts (Figure 3.9). When the interrupt from the 110 module occurs, the processor saves the contest (e.g.. program counter and processor registers) of the current program and processes the interrupt. In this ease., the processor reads the word of data from the. 110 module and stores it in memory. It then restores the context of the program it was working on (or some other program) and resumes execution.

Figure 7.5b shows the use of interrupt I/O for reading in a block of data. Compare this with Figure 7,5a. Interrupt 110 is more efficient than programmed **1/0** because it eliminates needless waiting. However, interrupt 1.0 still consumes a Jot

of processor Orne, 1˙pec,o LiSk, every word of data ihaL goes rrorn memory to I/O module or from IIC.) module to nicinory must pass through the processor..

Interrupt Processing

Let us consider the role of the processor in interrupt-driven DO in more detail. The occurrence of an interrupt triggers a number of events, both in the processor hardware and in soliware. Figure 7.7 shows a typical sequence. When an I/O device completes an 1.O operation. t he following secincore of hardware. weal* oeeurs:

11. The device issues an interrupt signal to the processor.
2. The processor finishes execurion of the current insiruction before responding to the interrupt, as indicated in Figure 3.9,

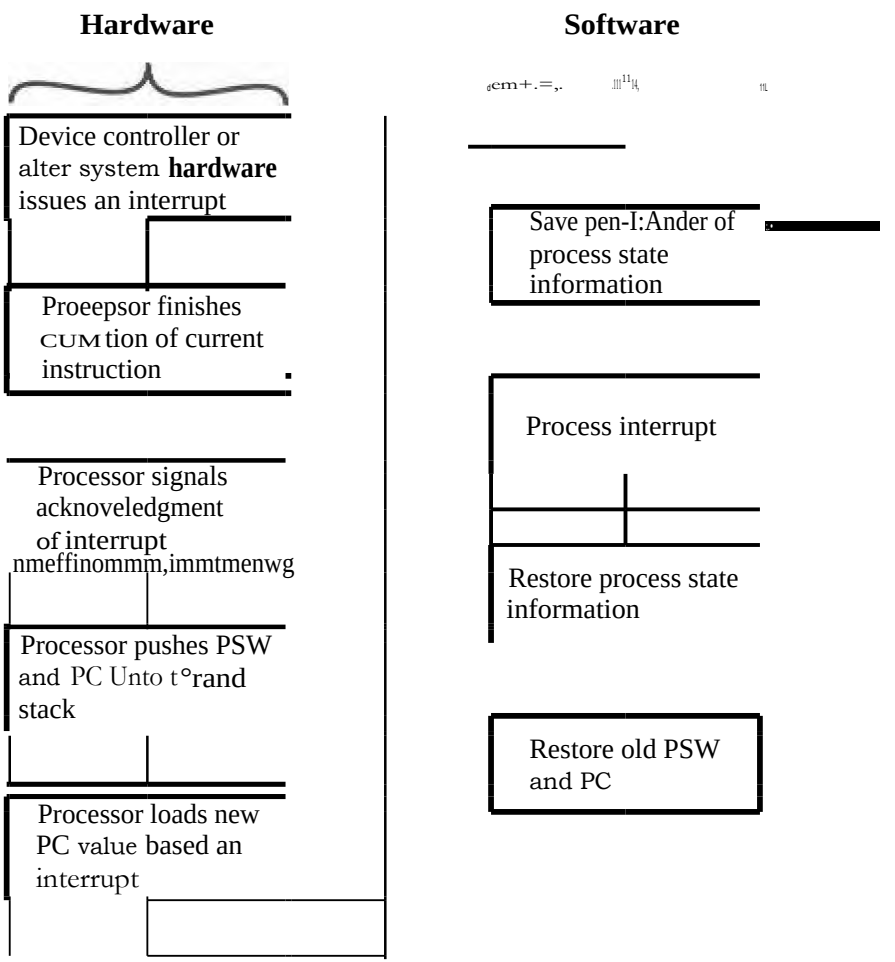


Figure 7.7 Simple Interrupt Processing

3. The processor tests for an interrupt. determines that there is one, and sends an acknowledgment signal to the device that issued the interrupt. The acknowledgment allows the device to remove its interrupt signal.
4. The processor now needs to prepare to transfer control to the interrupt routine_ To begin, it needs to save information needed to resume the current program at the point of interrupt. The, minimum information required is (a) the status of the processor, which is contained in a register called the program status word (PSW), and (b) the location of the next instruction to be executed, which is contained in the program counter. These can he pushed onto the system control stack.[1]
5. The processor now loads the program counter with the entry location ol' the interrupt-handling program that will respond to this interrupt. Depending on the computer architecture and operating system design, there may be a single program. one program for each type of interrupt, or one program for each device and each type of interrupt. if there is more than one interrupt-handting routine, the processor must determine which one to invoke. This information may have been included in the original interrupt signal. or the processor may have to issue a request to the device that issued the interrupt to get a response that contains the needed in formation.

Once the program counter has been loaded, the processor proceeds to the next instruction cycle. which begins with an instruction fetch. Because the instruction fetch is determined by the contents of the program counter. the result is that control is transferred to the interrupt-handler program. The execution of this program results in the following operations:

6. At this point, the program counter and PSW relating to the interrupted program have been saved on the system stack. However, there is other information that is considered part of the "state" of the executing program. In particular, the contents of the processor registers need to be saved. because these registers may be used by the interrupt handler. So, all of these values, plus any other state information, need to be saved. Typically, the interrupt handler will begin by saving the contents of all registers on the stack. Figure 7.6a shows a simple example. In this case, a user program is interrupted after the instruction at location N. The contents of all 0r the registers phis the address of the next instruction (N + 1) are pushed onto the stack. The stack pointer is updated to point to the new top of stack. and the program counter is updated to point to the beginning of the interrupt service routine.
7. The interrupt handler next processes the interrupt. This includes an examination of status information relating to the 1/0 operation or other event that caused an interrupt. It may also involve sending additional commands or acknowledgments to the I/O device.
8. When interrupt processing is complete, the saved register values are retrieved from the stack and restored to the registers te,,g_ see Figure 7.84

---

[1]See Appcntiir I OA For a discussion of sk Fick rspc Ntion.

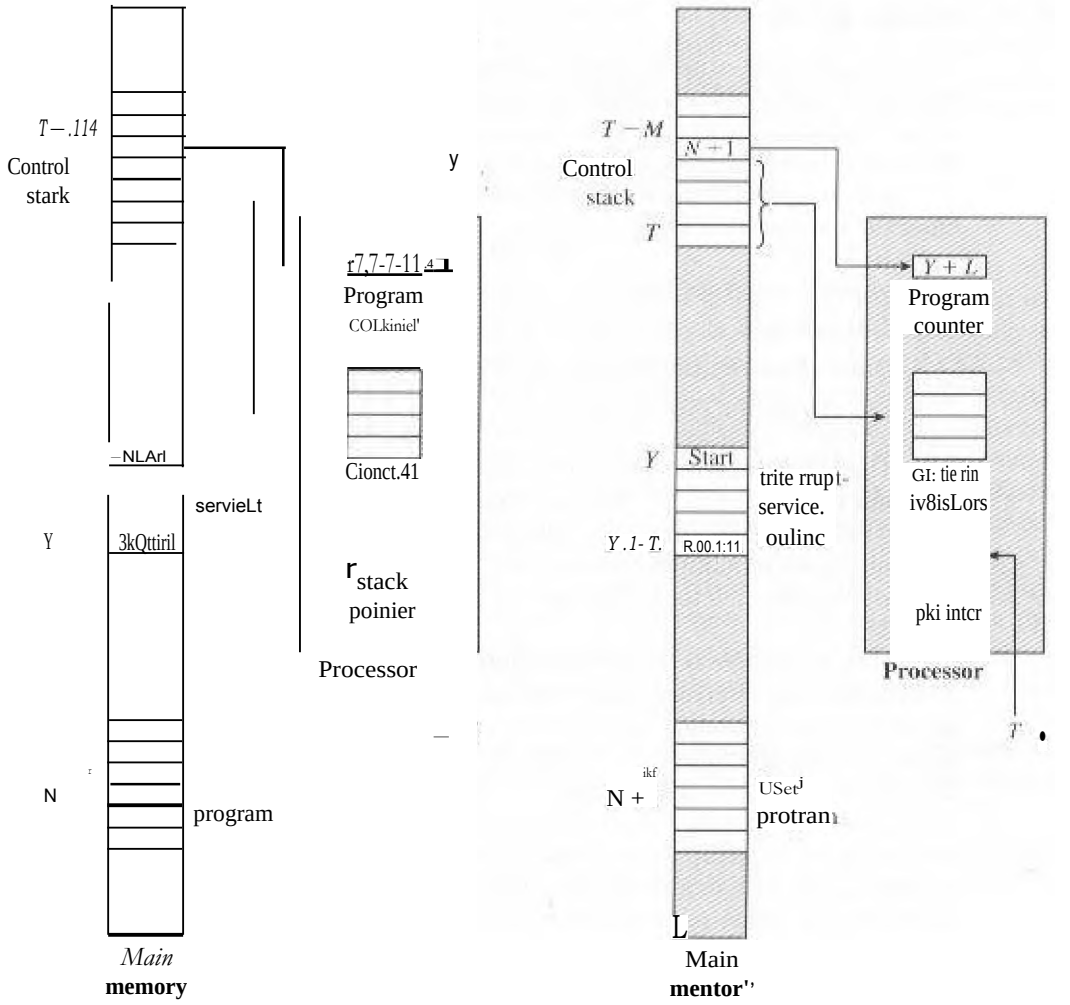


Figure. 7.8 Changes in Memory Find Registers fur an Interrupt

9. The final act is lo restore the PSW and program counter values from the stack. As a result. the next ingirnetion lo kpc. executed will be from the previously inierrtipted program.

Note th4i1ii is important to save all the state information about the interrupLea program for later resumption. 'rhis is because the. interrupt is not a routine called from the program. Rather, the interrupt can occur at any time and therefore at any point in the execution of a user program. Its oceurreiree is unpredietahle.lndecd. as we wi[[see in the next chapter. the two programs may not have anything in common and may belong to two different users,

Design Issues

Two design issues arise in implementing interrupt I10, First, because there will almost invariabl!,' be multiple I/0 modules, how does the processor determine which device issued the interrupt? And second, if multiple interrupts have occurred, *does* **the** processor decide which one tea process?

Let us consider deice identificatiOn first. Four general categories of techniques are in common use:

- Multiple interrupt lines
- Software Fail
- Daisy chain (hardware poll, vectored)
- Bus arbitration (vectored)

The most straightforward approach to the problem is to provide **multiple interrupt lines** between the processor and the modules. However, it is impractical to dedicate more than a few bus lines **or** processor pins to interrupt lines. Consequently. even if multiple lines are used, it is likely that each line will have multiple I/O modules attached to it. Thus, one of the other three techniques must be used on each line.

One alternative is **the software poll.** When the processor detects an interrupt, it branches to an interrupt-service routine whose job it is to polleach I/O module to determine which module caused the interrupt. The poll could be in the form of a separate command line (e.g., TEST110). In this case, the processor raises TEST110 and places the address of a particular 110 module on the address lines, The I/O. mod**ule** responds positively if it set the interrupt. Alternatively, each I/O module could contain an addressable status resister. The processor then reads the status register of each 1/0 module to identify the interrupting module. Once the correct module is identified, the processor branches to a device-service routine specific to that device,

The. disadvantage of the software poll is that it is time consuming. A more efficient technique is to use a **daisy chain,** which provides, in effect, a hardware poll. An example of a daisy-chain configuration is shown in Figure 3.25. For interrupts, all 110 modules share a common interrupt request line. The interrupt acknowledge line is daisy chained through the modules. When the processor senses an interrupt, it sends out an interrupt acknowledge_ This signal propagates through a series of ED modules until it gets to a requesting module. The requesting module typically responds by placing a word on the data lines. This word is referred to as a *vector and* is either the address of the 110 module or some other unique identifier. In either case, the processor uses the vector as a pointer to the appropriate device-service routine. This avoids the need to execute a general interrupt-service routine first. This technique is called a *vectored interrupt*

There is another technique that makes use of vectored interrupts, and that is bus arbitration. With hus arbitration. an 1i0 module must first gain control of the taus before it can raise the interrupt request line. Thus_ only one module can raise the li ne at a time, When the processor detects the interrupt, it responds on the interrupt acknowledge line. The requesting module then places its vector on the data lines.

The aforementioned techniques serve 10 identify the requesting module. They also provide a way of assigning priorities when more than one device is

requesting interrupt service, With multiple lines, the processor just picks the interrupt line with the highest priority. With software polling. the order in which modules are polled determines their priority- Similarly, the order of modules on a daisy chain determines their priority. Finally. bus arbitration can employ a priority scheme, as discussed in Section 3.4.

We now I urn to Iwo examples of interrupt strueLures.

Intel 82C59A Interrupt Controller

The Intel 80386 provides a single Interrupt .Request N' ER) and a single Interrupt Aek.nov,.ledge line. 'ro i]low the 80386 to handle a variety of devices and priority structures, it is usually configured with an external interrupt arbiter. the 82C:59A. External devices are connected to the 82C'59A, which in turn connects to the 80386.

Figure. 7,9 showf the use of the 82C.59A to connect. multiple 1.'O modules for the 80386. A single 8205)A can handle up to 8 modules. If control for more than modules is required, a cascade arrangement can he used to Lindlc up 64 modules.

The 82C9A s sole responsibility i.s the management of interrupts. it accepts interrupt requests from attached modules, determines which interrupt has the highest priority, and then signals the processor by raising the INIR line. rho processor acknowledges via the 1NTA line- This prompLs the 2C.5[1]A to place the appropriate vector ihrormai ion on the data bus. The processor can then proceed to process the interrupt and to communicate directly with the I10 module to road or write data,

The 82C59A is programmable. The 80386 determines the priority scheme to be used by setting a control word in the 82C59A. The following interrupl modes are possible:

- Fully nested: The interrupt requests are ordered in priority from 0 1RO.) through 7 (IR7).
- Rotating: In some applications a number of inierrupting devices are of equal priority. In this mode a Jeviec, after being serviced, receives the lowest priority in the group.
- Special musk: This allows the processor lo illhihji interrupts l'rom certain devices.

The Intel 82C55A Programmable Peripheral Interface

As an example of an Ii0 module used for programmed I/O and interrupt-driven 110. we consider the Inwl 82C55 A Programmable Peripheral Interlace. The 82(:35A is a single-chip, gLAteral-purpose I/O module designed for use with the Intel 80386 processor, Figure 7.10 shows a general block diagram plus the pin assignment for the 40-pin package in which it is houscd.

The right side of the block diagram is the external interface of the 82C55A, The 24 110 lines are programmable by the 80386 by means of the covil rol register. ' rhe 8038.fican set the value of the conlrul register to specify a variety of operating modes and configurations. The 24 lines are divided into three 8-bit groups (A, B, C). Each group can function as an 8-bit I/O port. In addition, group C' is subdivided into 4-bit groups (C', and C_H), which may he used in conjunction with the A and B lit) ports. Configured in this manner, they carry control and sLitus signals.

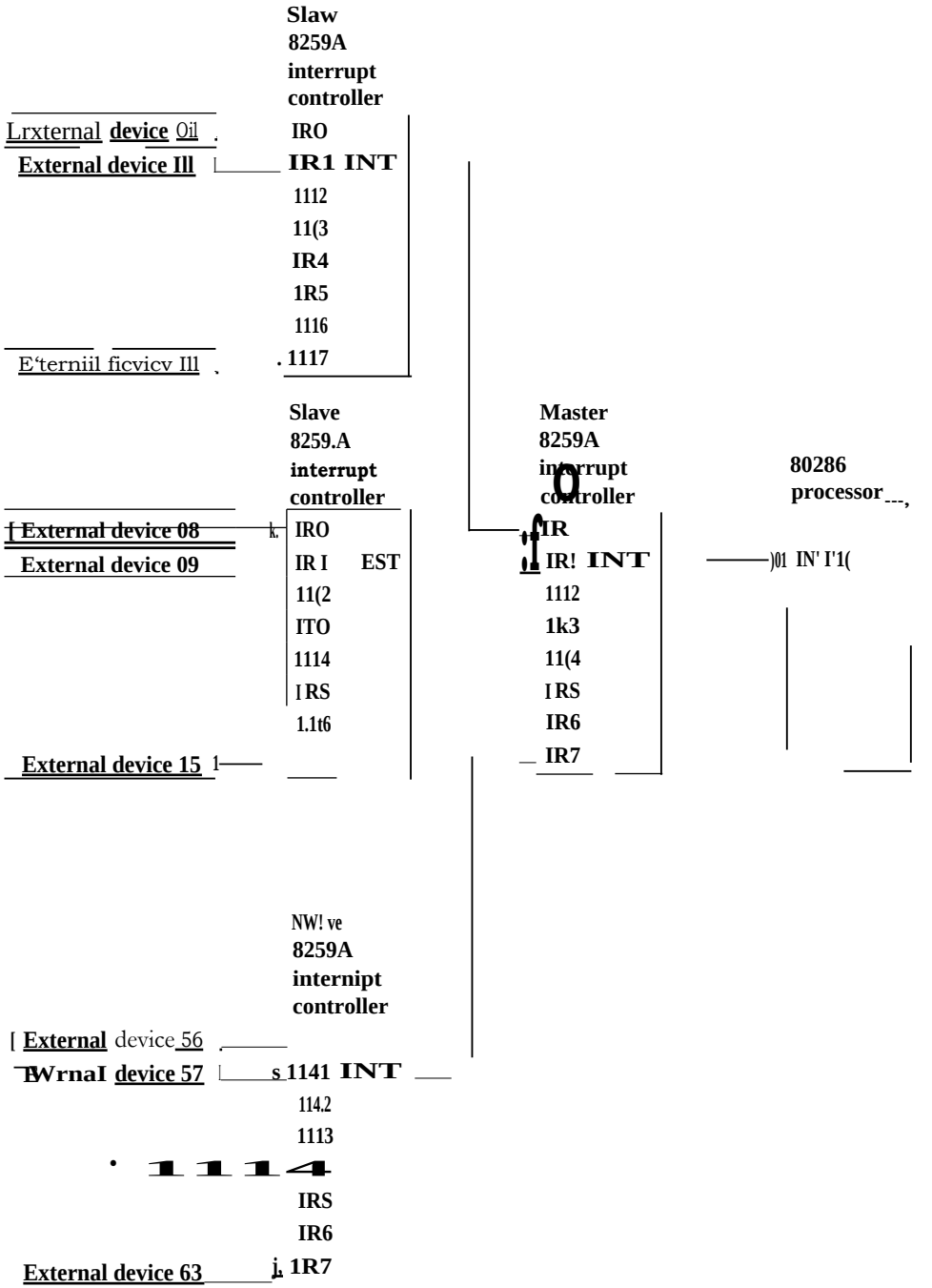
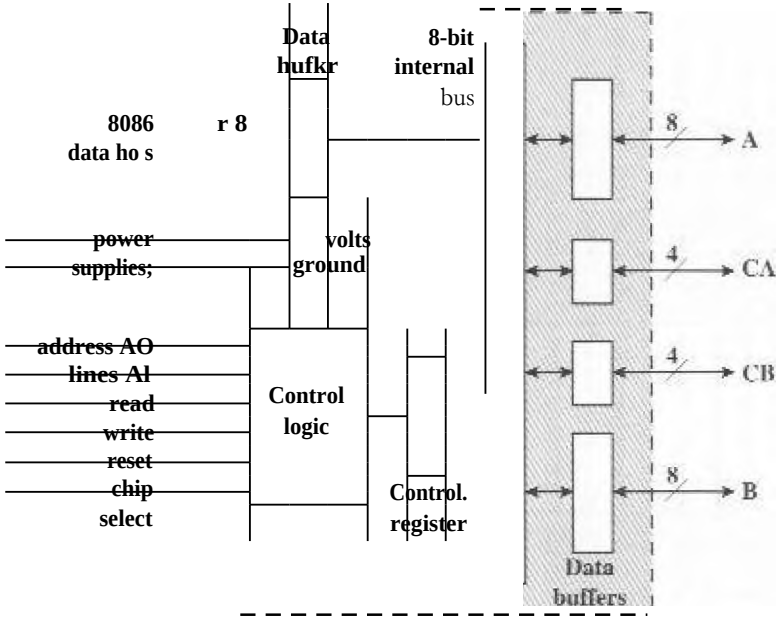
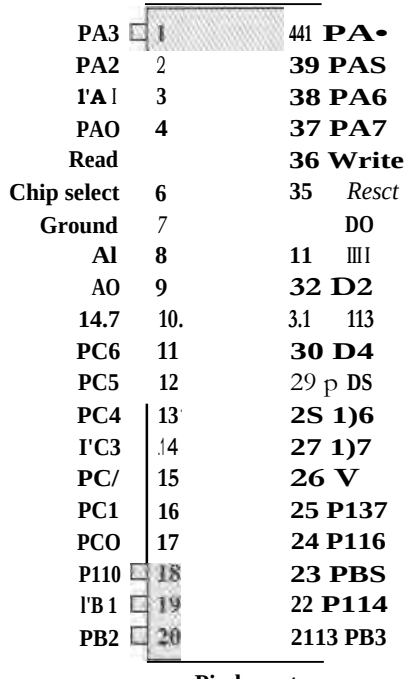


Figure 7.9 Use or the t32C59A Interrupt Controller



(al Block diugrarn



Pin layout

Figure 7.10 The Intel 82C55A ProgrammaNe Peripheral Interlace

The left side of the block diagram is the internal interface to the 80386 bus. It includes an 8-bit bidirectional data bus (DO through D7), used to transfer data to and from the 110 ports and to transfer control information to the control register, The two address lines specify one of the three 1/0 ports or I he control register. A transfer takes place when the CHIP SELECT line is enabled together With either the READ or WRITE line. The RESET, line is used to initialize the module.

The control register is loaded by the processor to control the [node of operation and to define signals, if any. In Mode 0 operation. the three groups of eight external lines function as three 8-hit 110 ports. Each port can he designated as input or output. Otherwise. 2,rour, A and B function as Ii0 ports. and the lines of group C serve as control lines for A and B. 'The control signals serve two principal purposes: ¯handshaking" and interrupt request. Handshaking is a simple liming mechanism. One control line is used by the sender as a DATA READY line, to indicate when the data are present on the 110 data lines, Another line is used by the receiver as an ACKNOWLEDGE, indicating that the data have been read and the data lines may he cleared. Another line may be designated as an INTERRUPT REQUEST line and tied back to the system bus.

Because the 82.(:55A is programmable- via the control register, it can be used to control a variety of simple peripheral devices. Figure 7.11 illustrates its use to control a keyboard/display terminal. The keyboard provides 8 hits of input. Two of these bits, SHIFT and CONTROL. have special meaning to the keyboard-handling program executing in the processor. However, this interpretation is transparent to the $2C55A. which simply accepts the 8 hits of data and presents them on the system data bus. Two handshaking control lines arc provided for use with the keyboard,

The display is also linked by an 8-bit data port. Again, two of the bits have special meanings that are ˙transparent to the 82C55A. In addition to Iwo handshaking lines, two lines provide additional control functions.

7.5 DIRECT MEMORY ACCESS

Drawbacks of Programmed and Interrupt-Driven I/O

Interrupt-driven 1/0. though more efficient than simple programmed I/O, requires the active intervention of the processor to transfer data between memory and an 1.10 module, and any data transfer must traverse a path through the processor. Thus, both these forms of 1/0 suffer from two inherent drawbacks:

1. The I/O transfer rate is limited by the speed with which the processor can test and service. a device.
2. 'file processor is tied up in managing an 110 transfer: a number of instructions must be executed for each I/O transfer (e_g.. Figure 7.3).

There is sornc•kkhat of a trade-off between these two drawbacks. Consider the transfer of a block of data. I.:sing simple programmed I/O, the processor is dedicated to the task of I/0 and can move data ul a rather high rate.. at the cost of doing nothing else. Interrupt 1/0 frees up the processor to some extent at the expense of the

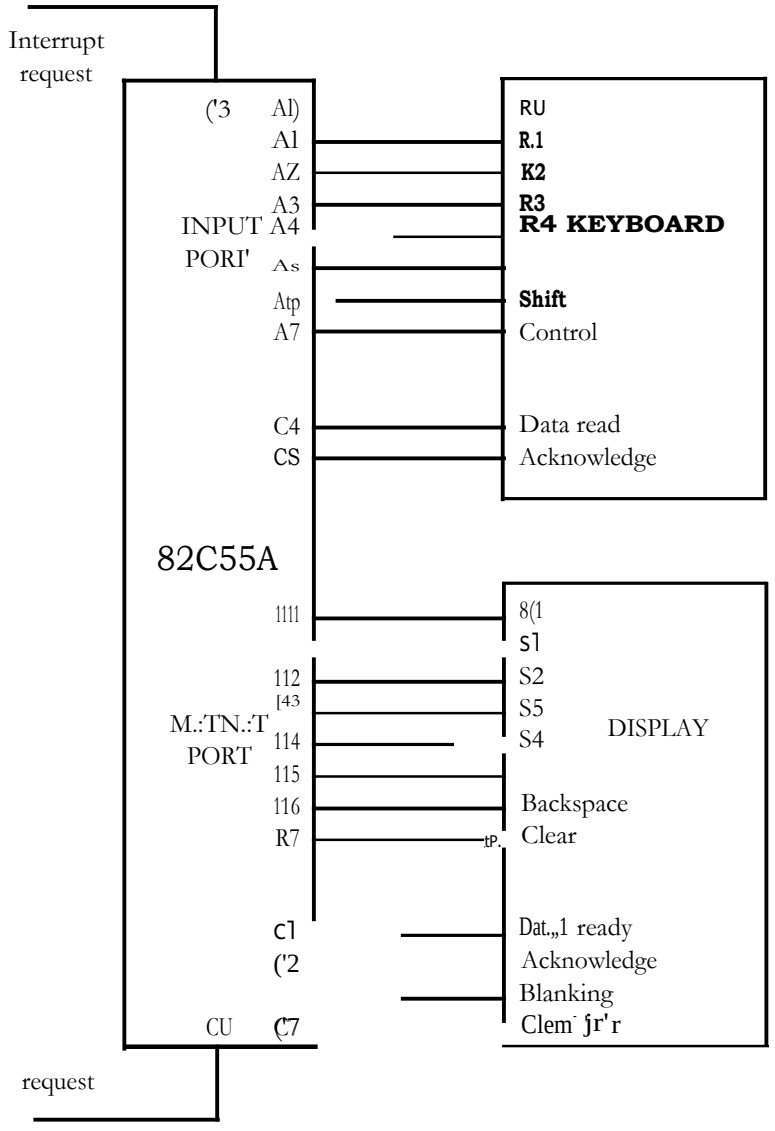


Figure 7.11 KcyboardiDit:play **.1rticrfaci** (0 82(..f).Li A

110 1ransil2r ratc. Nevertheless, both methods have an adverse impact on both processor activity and 1/0 transfer rale.

When 1arge. volunie:s cif dalu 2irc kr Ile moved, a more efficient k.%21iniquc is ruquitc:d: direct meinory access (DMA).

DMA Function

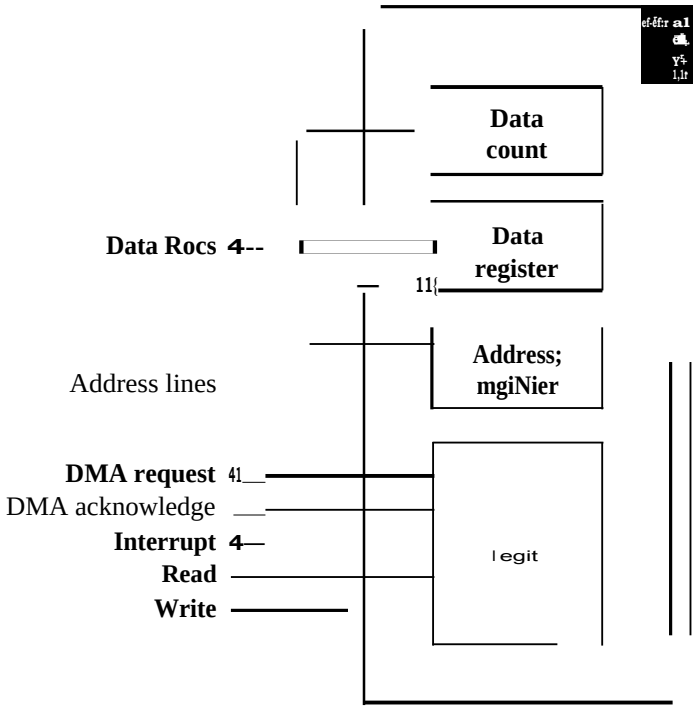
I)MA involves an additional module on the system bus, Iht: I)M A module (Figure 7.12) is capable of mimicking the processor and. indeed, of taking over control of

ll.ic systcrn 1 'rorn the processor. It needs to do this to transfer data to and from memory over the. system bus. For this purpose, the DMA module must use the bus only when the processor does not need it. or ii must force the processor to suspend operation lemporarily. 'tile fatter technique is more common and is referred to as *c yde Nieedirig,* because the DMA module in effect steals a bus cycle.

When the processor wishes to read or write. a Mock of data, it issues a command to the DMA module, by Ki,:ndirtg to the DMA module the following information;

- Whether a read or write is requesled, using the read or write control line between the processor mid the DMA module
- The address of the 110 device involved, communicated salt the data lines
- The starting location in mernor!,' to read from or write to, communicated on the data lines and stored by the DMA module in its addr4,:ss register
- The. number of words to he read or written, again communicated via the doto li nes and stored in the data count register

The processor then continues with other work. It has delegated this 110 opcc. ation to the DMA module. The DMA module transfers the entire block of data , one word at a time, directly to or from memory, withoin going through the processor. When the transfer is complcle. the DMA module sends an interrupt signal to she processor, **Thus.** the processor is involved only at the beginning and end of the trans. ter (Figure, 7,5c),



Figury 7.12 'typical DMA Block **Diagram**

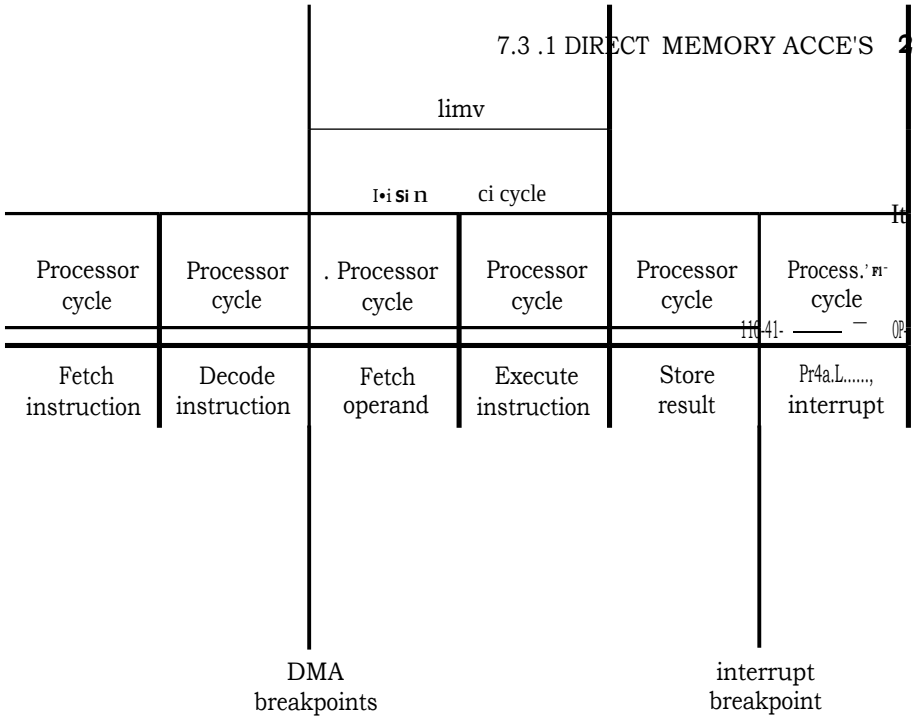
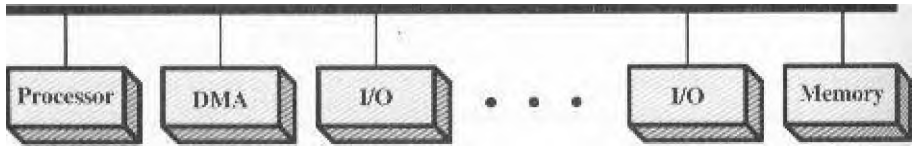


Figure 7.0 DMA and Interrupt Breakpoints during an Instruction Cycle

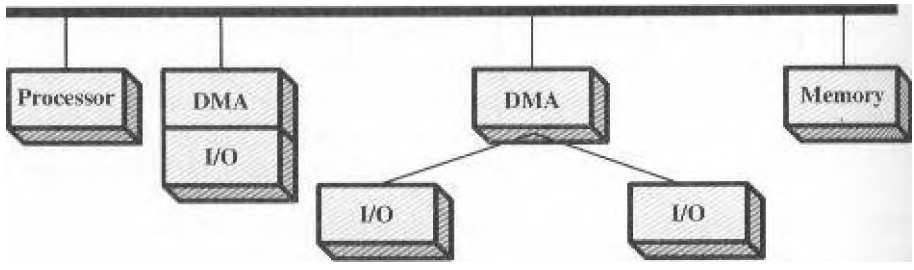
Figure 7.13 shows where in the instruction cycle the procesSor may **he** suspended. In each case, the processor is suspended just before it needs to use the bus, The DMA module then transfers one word and returns control to the processor, Note that this is nOt an interrupt; the processor does not save a context and do something else. Rather. the processor pauses for one bus cycle. The overall effect is to cause the processor to execute more slowly, Nevertheless, for a multiple-word 110 transfer, DMA is far more efficient than inte rrupt-dri yen or programmed 1/0,

The DMA mechanism can he configured in a variety of ways. Some possibilities are shown in Figure 7,14. 10 the first example, all modules share the same system bus_ The DMA module, acting as a surrogate processor, uses programmed I/O to exchange. data between memory and an 110 module through the DMA module, This **cuR** figuration, while it may be inexpensive, is dearly inefficient. As with processor-controlled programmed I/O, each transfer of a word consumes two bus cycles.

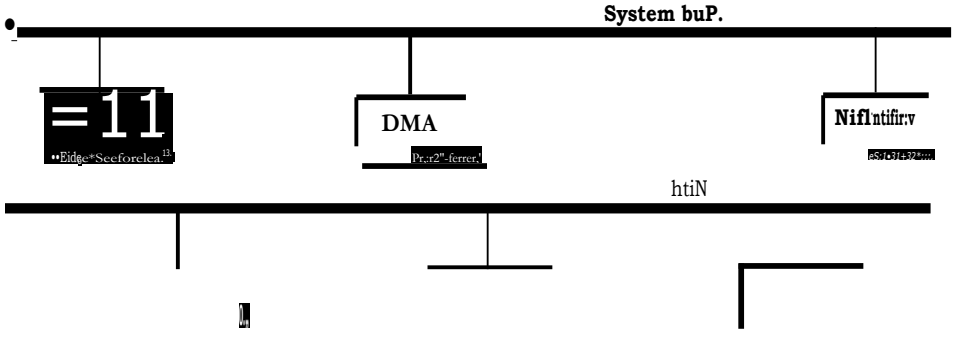
The number 0f required bus cycles can be cut substantially by integrating the DMA and 110 functions. As Figure 7.14b indicates, this means dial there is a path between the DMA module and one or more I/O modules that does not include the system bus. .1 he DMA logic may actually be a part of ;in I/O module, or it may be a separate module that controls one or more I/O modules. This concept can he taken one step further by connecting 110 modules to the DMA module using an I/O bus (Figure 7.14c). 'this reduces the number olliO interfaces in the DMA module to one and provide:, for **an easily** expandable configuration. In all of these cases (Figures 7.14b and c). the system bus that the DMA. module shares with the processor and memory is used by the DMA module only to exchange data with memory. The exchange of data between the DMA and modules takes place off the system bus.



(a) Single-Ims, **detactwd DMA**



(I)) Singie-hus, integrated DMA-1.10



(u) LIO hos

Figure 7.14 Alternative DMA Configurations

7.6 1/0 CHANNELS AND PROCESSORS

The Evolution of the 1/0 Function

computer '63'**gems he ve** evoived, there has been a pattern of in com-piexity and sop.histication of individual components. Nowhere is this **In ore** evident than in the 1.10 function. We] already wen pi.irt of thatevolution. The evoiu-nary steps can mma &dm folic-Avg:

I. The CPU directly controls a peripheral device. This is seen in simple microprocessor-controlled devices,

A. controller or I/O module is added. The CPU uses programmed I/O without interrupts. With this step, the CV(becomes somewhat divorced from the specific details of external device interfaces.

3, .l'he same configuration as in step 2 is used, but now inte]Tupnx arc employed. The CPU need not spend lime waiting for an 1/0 operation to be performed, increasing efficiency.

4, The I/O module is given direct aceesz., to memory via DMA. can now move a block of data ICT or from memory without involving the CPU, except at the beginning and end of the transfer.

5. The 1/0 module is enhanced to become a processor in its own right,.with a specialized instruction ;,1c1 tailored for 1.10. The CPU directs the processor to execute an I/O program in memory, The 110 processor fetches and executes these instructions without CPU intervention. This allows !he CPU to specify a sequence of LII) ad ivi ties and to be interrupted only when the entire sequence has been performed.

6. The I/O module has a local memory of its own and is, in fact. a computer in its own right, With this architecture., a large set of I/O devices can be controlled, with mipirnal CPU involvement. A common use for such an arch ileei tire has been to control communication with interactive terminals. Hie I/O processor takes care of most of the lacks; involved in controlling the terminals.

As one proceeds along this evoluiionary path , more and more of the I/O function is performed wilho w. CPU involvement. The CPI! is increasingly relieved of l.10-related improving performance. h the Last two steps (5-6), a major change occurs with the introduction of ihe 0.311.Cept of an I/O module capable of executing a program. For step 5, I he I/0 module is often referred Lo as an I/O *channel.* For step 6. the term PO *processor* is often used. However, both terms are Lm occasion applied toy both situations. In what follows, we will use the term /.. [1]O *channel,*

Characteristics of If0 Channels

the 1/0 channel represents an extension of the DMA concept. An I/O channel has the ability to execulc I10 instructions, which gives it complete control over I/O operations. In a computer s!,•stein with such devices, the CPU does not execute I/O instructions. Such instructions are stored in main memory to be executed by a speeial-pu•posc processor in the I/O channel itself. Thus, the CP(initiates an I/O transfer by instructing the I/O channel to execute a program in memory. The program will specify the device or devices, the area or areas of memory ['or storage, priority, and actions to be taken for [(Alain error conditions. the 1/0 channel follows these instructions and controls the data transfer.

Two types or I/O channels are common. as illustrated in Figure 7.13. A *seleeoof uheumel* controls multiple high-speed devices and. at any one Lime, is dedicated to the transfer of data with one of those devices, Thus, die I/O channel selects one device and effects. the data transfer. Each device. or a small set of devices, is handled by a *coyuroller,* or 1/0 module, that is much like the I10 modules we have been

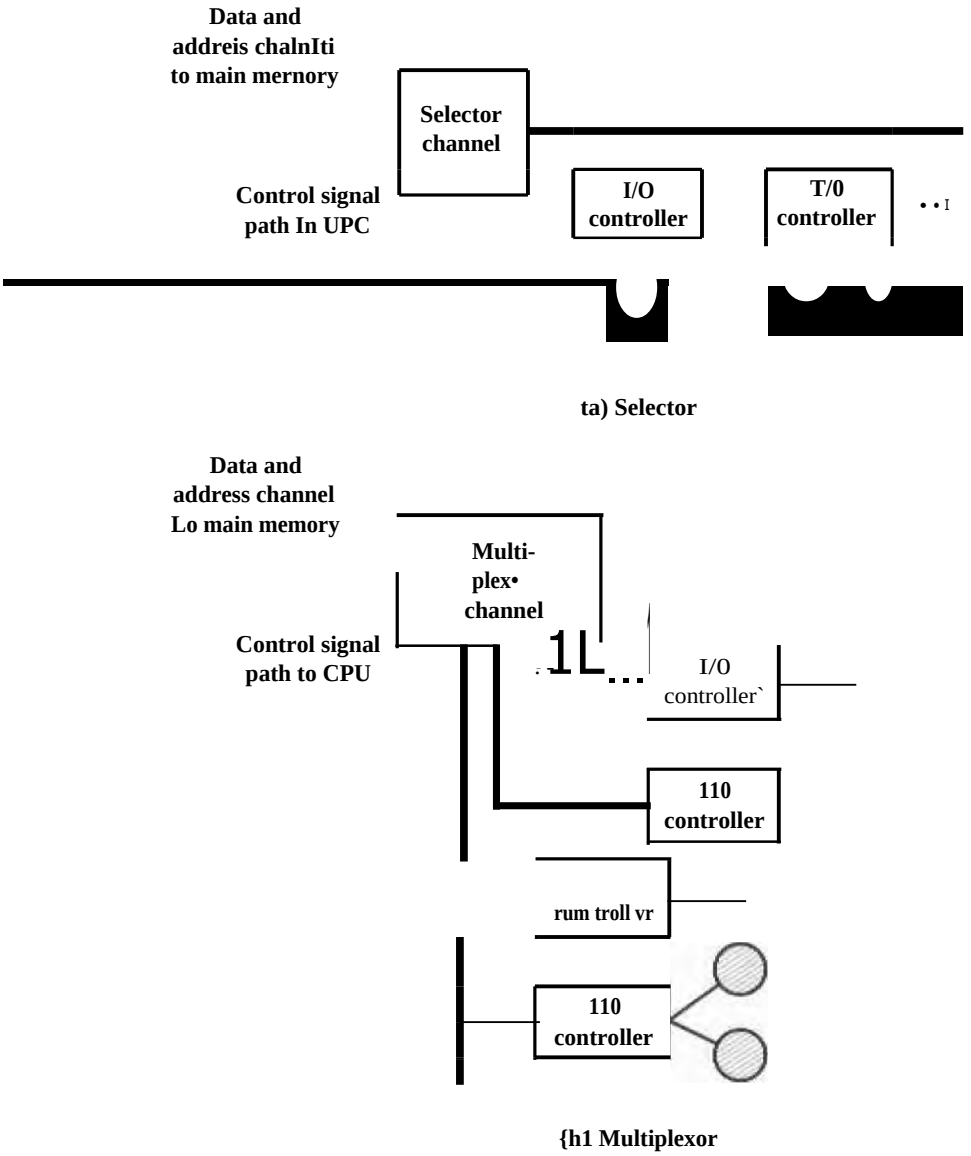


Figure 7.15 1/0 Channel Architecture

discussing. Thus, the 1rO channel serves in place of the CPU in controlling these ISO controllers, A *multiplexor channel* can handle 1.10 with multiple devices at the same time, For low-speed devices, a *byte multiplexor* accepts or transmits characters as fast as possible to multiple. devices.HFT example, the resultant character stream from three devices with different rates and individual streams A, $A_2A,A,$ $B_1B,13;14_4$ and $C_1C,C.,S'_4$ ... might be $A_1B,C,AC\text{-}A_3B,C,A_4$, and so on. For high-speed devices, a *block multiplexor* interleaves blocks of data from several devices,

7.7 THE EXTERNAL INTERFACE: FIREWIRE AND INFINIBAND

Types of Inter6ces

1'1Lc interface to a peripheral from an module must be tailored to the nature and operation of the peripheral. One, major characteristic of the. interface is whether it is serial or parallel (Figure 7.16). In a **parallel interface,** there are multiple lines connecting the I/O module and the peripheral, and multiple hits are transferred simultaneously. just as all of the bits of a word are transferred simultaneously over the data bus. In a serial interface, there is only one line used to transmit data, and bits must be transmitted one at a time. A parallel interface has traditionally been used for higher-speed peripherals, such as tape and disk, while the serial interlace has traditionally been used for printers and terminals. With a new generation of high-speed serial interfaces. parallel interfaces are becoming much less.common,

In either case, the module must engage in a dialogue with the peripheral. In general terms, the dialogue for a write operation is as follows:

1. The ID module sends a control signal requesting permi.ssion to send data.
2. The peripheral acknowledges the request.
3. I **he** module transfers data (one word or a block depending on the pc riphera I).
4. The peripheral acknowledges receipt of the data.

A read operation proceeds

Key to the operation of an I/O module is an internal hurler that can store data being passed between the peripheral and the rest of the system. This buffer allows

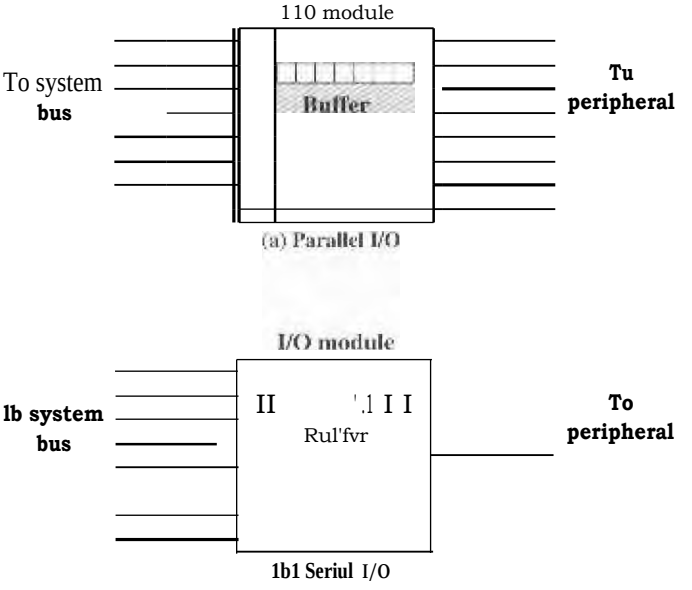


Figure 7.16 Parallel and Serial

the module to compensate for the differences in speed between the system but and its external lines.

Point-to-Point and Multipoint Configurations

The connection between an 1/0 module in a computer system and external deNims can be either point-to-point or multipoint. A point-to-point interface provides .. dedicated line between the I10 module and the external device. On small systems (VCs, workstations), typical point-to-point links include those to the keyboard. printer, and external modem. A typical exampIC of such an interface is the EiA-n specification (see [STAUXI] for ti description).

Of increasing importance are multipoint external inierfaces, used to supporl external mass storage devices (disk and tape drives) and multimedia devices (CD• ROMs, video, audio), These mullipoirit interfaces are in effect external bums. anc they exhibit the same type of logic as the buses discussed in Chapter 3. In this sec-Li on, we look at two key examples: Fire,Wire and I nfiniBand.

FireWire Serial Bus

With processor speeds reaching 0Hz range and storage devices holding multiple gigabits, the L1O demands for personal computers, workstations, and servers an formidable, Yet the high-speed channel technologies that have *been* developed for mainframe and supercomputer systems are loo expensive and **bulky for use** on these smaller systems. Accordingly, the has been great interest **in** developing a high-speed alternative lo SCSI and tither small-system LIO interfaces.. The result is the IEEE standard 1194, for a high-performance serial bus, commonly known as FireWire.

FireWire has a number of advantages over older I10 interfaces. It is very high speed,]ow cost. and easy to implement. In fact, FireWire is finding lavor not only for computer systems, but also in consumer electronics products, such as digital cameras. VC'Rs, and televisions. In these products. FireWire is used to transport video images, which are increasingly coming from digitized sources.

One of the sircngths of i.he FireVv'ire interface is that it uses serial transmission (hit at a lime) rather than parallel. Parallel inlerfaccs, such as SCSI, require more wires, which means wider, more expensive cables and wider, more elmsive connectors with more pins to bend or break. A cable with more wires requires shielding to prevent electrical interference between the wires. Also, with a parallel interface, synchronization between wires becomes a requirement, a problem that gets worse with increased cable length-

In addition, compuiers are **getting** physically smaller even As they expand in computing power and needs. Handheld and pocket-siv.c computers have little room for connectors yet need high data raLes Lo handle images and video,

The intent of FireWire is to provide a single I10 interface with a simple connector that can handle numerous devices through a single port, so that the mouse, laser printer, external disk drive, sound. and local area network hookups can be replaced with this single connector The connector is inspired by the one used in the Nintendo Gameboy. I i is so convenient that the user can reach behind the machine and plus it in without Looking.

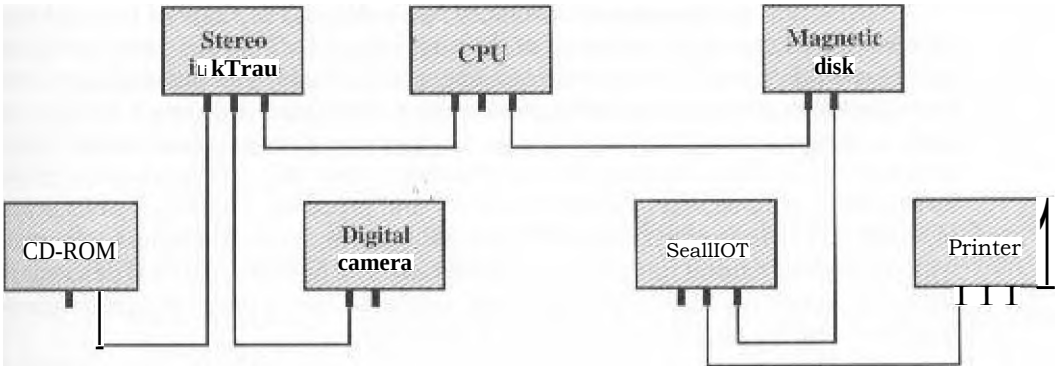


Figure 7.17 Simple EircWire Configuration

FireWire Configurations

FireWi re uses a daisy-chain configuration, with up to 63 devices connecLcd off a single port. Moreover, up log 1022 FireWire buses can he in1erconnucted using bridges, enabling a system to support as mani,. periphera Is as required.

FireWire provides for what is known as hoi plugging, which makes it possible to connect and disconnect periphern Is without having to power the computer system down or reconfigure the system, Also, FireWire provides for automatic configuration: it is not necessany. manually lo set device fas or to be concerned W ith the relative position of devices. Figure. 7. t 7 shows a simple FireWire configuration. With FireWire, there are no **tunurmi** kills. and the system automatimIly performs a configuration function Lo assign addresses. Also note lhal FircWire bus need not be a so-id **cLisy** chain. Rather, a tree-structured configuration is possible.

Au important feature of the FireWire standard is that it specifies a set of three layers of protocols to standardize the way in which the host system interacts with the peripheral devices over the serial bus. Figure 7.18 itlu.strates this stack. The three layers of the stack are as follows:

- **Physical layer:** Defines the transmission media that are permissible under FireWire and the electrical and **signaling** characteristics of each
- **Link** layer: Describes the transmission of data in the packets
- Transaction layer: Defines a request-response protocol that hides the lower-layer details of FireWire from applications

PhyNical Laker

The physical layer of FireWire specifies several alternative. transmission media and their connectors, with different physical and data transmission properties. Data rates from 25 to 400 lsilbps are defined. The physical laver converts binary Jain into electrical signals for various phy;,lical media. This layer also provides the arbitration service that guarantees that only one device at a time will transmit data

Two forms of arbitration are provided by FireWire, The simplest form is based on the tree-structured a **rrallRerenl** of the nodes on a FireWire bus, mentioned earlier. A special case of this structure is a linear daisy chain. The physical layer cap tains logic that allows all the attached devices to configure themselves so that one node. is designated as the root **of** the tree and other nodes are organized in a parent/child relationship forming the tree topolo*.r. Once this configuration is cgabfished, the root node acts as a central arbiter and processes requests for bus access in a first-conic-first-served fashion. In the case of simultaneous requests, the nede with the highest natural priority is granted access. The natural priority is determined by which competing node is closest to the root and. among those of equal distance from the root, which one has the lower ID number.

The itforementioned arbitration method is supplemented by two adlitional functions: fair arbitration and urgent arbitration. With fairness arbitration. time on the bus is organized into *fairness itervals.* Al the beginning of an interval. each node sets an arbitration_enable flag. During the interval, each node may compete for 1-Pus access. Once a node has gained access to the. bus, it resets its arbitration _enable tlag and may not again compete for fair access during this interval. This scheme makes the arbitration more fair. in that it prevents one or more busy high-priority devices from monopolizing the bus.

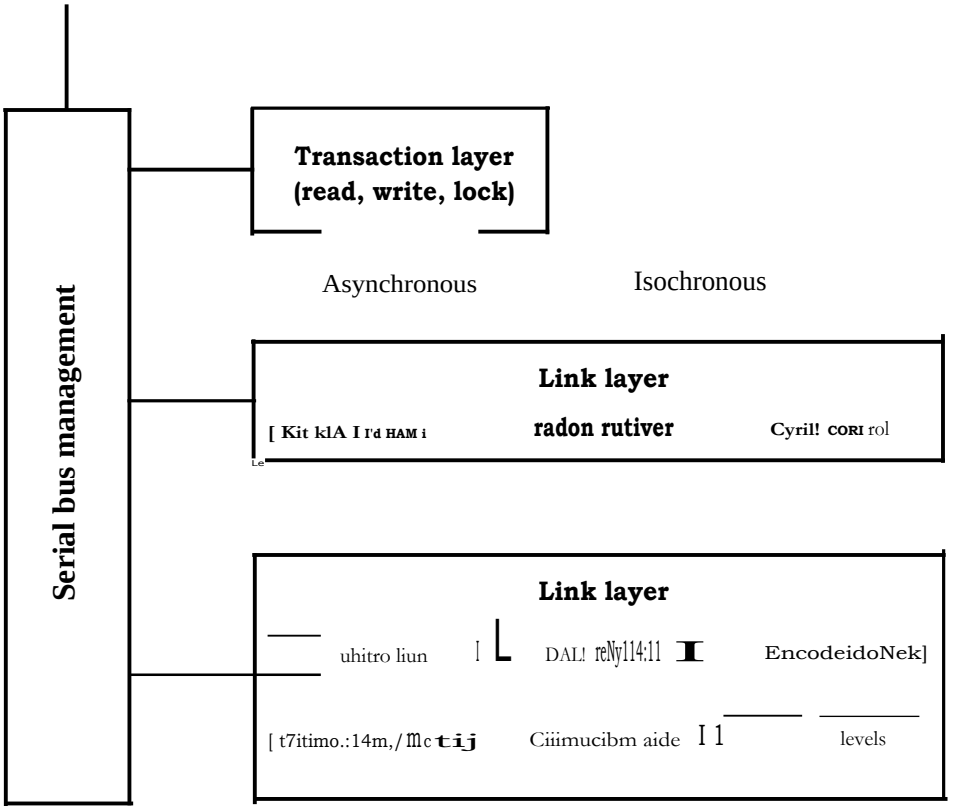


figure 7.18 1 •ii.c Wirc Protocol Stack

In addition to the fairness scheme, some LkviQc.s may be configured as having *urgent priority. Such* nodes may gain control of the bus multiple time during a fairness interval- In CNScnce, a counter is used at each high-priority node that enables the high-priority nodes to control 75% of the availabre bus time. For each packet that is transmitted as nonurgent, three packets may Inc transmitted a6. urgent.

Litrk Layer

The link layer defines the transmission of data in the form of packets. Two types of transmission are k.upported:

- Asynchronous: A variable amount of data and several bytes of transaction layer infOritlai t,re 1nm:slurred as a packet to an explicit address and an acknowledgment is returned,
- Isochronnus: A variable aMoulfil of data is transferred in a sequence of fixed-size. packets transmitted at regular intervals. This flPrin crf transmission uses simplified addressing and no acknowledgment.

Asynchronous transmission is used by data that have no fixed data rate requirements. Both the fiiirarfaiLTI [ion and urgent arbitration schemes may he used *for* asynchronous transmission. The default method is fair arbitration. Devices that desire a substaMial Fraction of the bus capacity or have severe .l4l.lxnry requirements use the urgent arbitration method. For example, a high-speed real-time data eoltection node may use urgent arbitration when critical data buffers are more than half full.

Figure 7.19a depicts a typic-at asynchronous transaction. The process of delivering a single packet is called a subaction. The subaction consists of *five* lime periods!

- Arbitration sequence: This is the enchange of signals required to give one device control of the bus.
- racket transmission: Every packet includes a header containing the source and desiinalion Ids. The header also contains packet type information, a CRC (cyclic redundancy check) checksum, and parameter information for the specific packet type. A packet may also include a data block consisting of user data and another CRC.
- Acknowledginent gap: l'his is the time delay for the destination to receive and decode a packet and generate an acknowkdgment,
- Acknowledgment; 'The recipient of the packet returns an acknowledgment packet with a code indicating the action taken by the recipient.
- Subaction gap: Thk is an enforced idle period to ensure that other nodes on the. bus do not begin arbitrating before the acknowledgment packet has been transmitt ed.

AL the time that the acknowledgment is sent, the acknowledging node is in control of the bus. Therefore, if the exchange is a request/response interaction between two nodes ; then the responding node can immediately transmit the response packet without going through an arbitration sequence (Figure 7. L9b).

For devices that regularly generate or consume data, such as digital sound or video, isochronoos access is provided. This method guarantees that data can be delivered within a specified latency with a guaranteed data rate.

To accommodate a mixed traffic load of isochronous and asynchronous data sources, one node is designated as *cycle master_* Periodically. the cycle master issues a cycle_start packet, This signals all other nodes that an isochronous cycle has begun. During this cvelc, Only isochronous packets may be sent (Figure 7.19c). Each isochronous data source arbitrates for bus access. The winning node immediately transmits a packet. There is no acknowledgment to this packet, and so other isochronous data sources immediately arbitrate for the bus after the previous isochronous packet is transmitted. The result is that there is a small gap between the transmission of one packet and the arbitration period for the next packet, dictated by delays on the bus. This delay, referred to as the isochronous gap, is smaller than a subaction gap.

After all isochronous sources have transmitted, the bus will remain idle long enough for a subaction gap to occur. This is the signal to the asynchronous sources that they may now compete for has access. Asynchronous sources may then use the. bus until the beginning of the next isochronous cycle.

isochronous packets are labeled with 8-hit channel numbers that are previously assigned by a dialogue between the two nodes that arc to exchange isochronous data. The header. which is shorter than that For asynchronous packets, also includes a data length field and a header CRC.

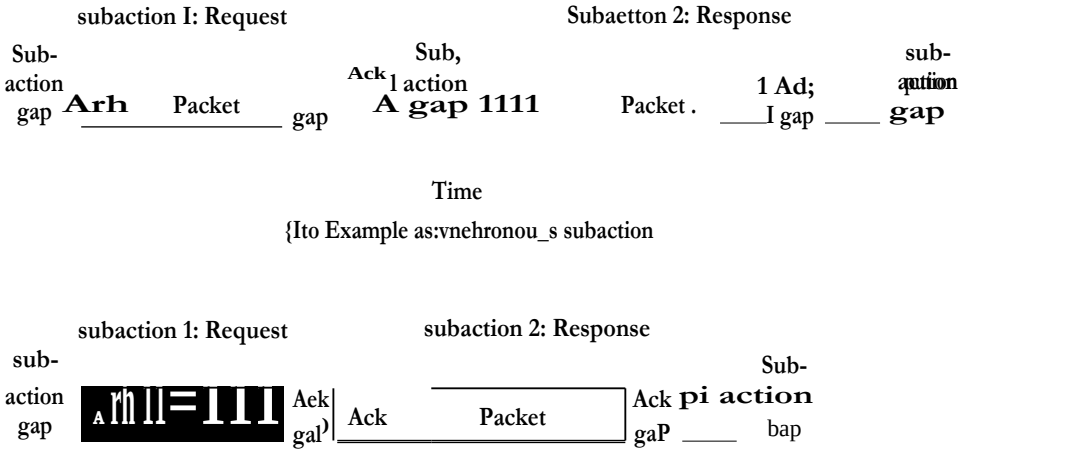


Figure 7.19 FireWire Subactions

lnfiniBand

InfiniBand is a recent 110 specification aimed at the high-end server market,' The first version of the specification was released in early 2.[)(1t and has attracted nurner-()UN vendors. The standard describes an architecture and specifications for data flow between processors **and** inteliigent I/O devices, InfinilIand is intended lo replace the ?CI bus **in** servers, to provide greater capacity, increased expandability, and enhanced flexibility in server design. **In** essence, InfiniBand enables servers, remote storage. and other network devices to be attached **in** a central fabric of switches and links, The switch-based architecture can connect up to 64,000 servers, storage systems, and networking devices,

Infiniband Architecture

Although PCI is a reliable interconnect method **and** continues to provide increased speeds, up to 1 Gbps, it is a limited architecture compared to In finiband. With InfiniBand, it is not necessary to have the basic I/O interface hardware inside the server chassis. With infinilIand, remote storage, Del working, and connections between servers arc accomplished by altaching all devices to a central fabric of switches **and**]links. Removing I/O from ihc server chassis allows greater server densitt allows for a more **11,:xibie arid scakiNe** data center, as independent nodes may be added as **necked,**

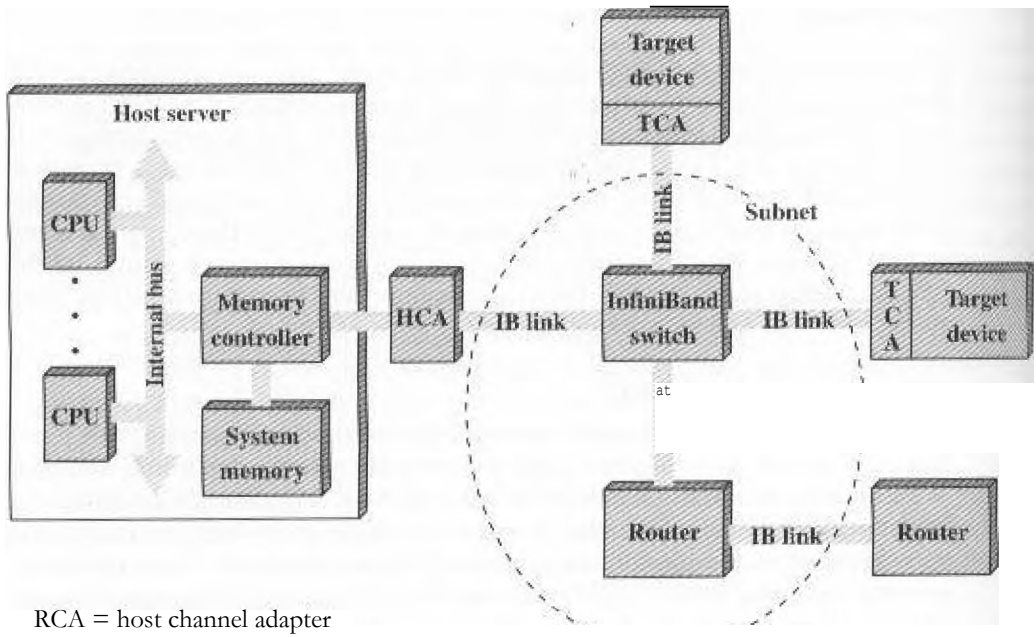
Lin likc Pek which measures distances from a CPU motherboard in centimeters, I afiniBand's channel design enables **1/0** devices to be placed up to 17 ln away from the server using copper. up to 31111 in using mid ti mode optical fiber. and up +to 10 **km** with single-mode optical fiber. Transmission rates has high as 30 Gbps can he achieved.

Figure 7,20 illustrates the InfiniBand architecture. The key elements are as follows;

* **Host channel adapter (HCA):** Instead of a number ,,r pfa slots. a typical server needs a single interface to an HCA that links the server to an InfiniBand switch, The HCA attaches lo the server at a memory controller, which has access to the system bus and controls traffic between the processor and memory and between the FICA arid memory. The !ICA uses direct-memory access (I)Iv(A) to read and write memory,
- **Target channel adapter (TCA):** A TCA is used to connect storage systems, routers. and other peripheral devices to an InfiniBand
- lnimiliond switch: A switch provides pains-to-point physical connections to a arteL of devices and switches **traffic** from one link to another. Servers and devices commliniCiJ Le ihrough their adapters. via the switch. The switch's intelligence manages the linkage without inlerruptirux the servers' operation.
- **Links: The** link between a switch and a channel adapter, or between. two switches-

---

4 1hilini.band is thc rusult col Lis roor r of two corr3pclin2 projucIF: Future U0 I h.rickQLI Ivy risco, HP. Compnti, and 1[310) art] Next CTCLIC78 100 1.O i*,..121oped by Intel and hacloz.d by vanisher 4.)C ocher cotnrtanisk



RCA = host channel adapter
'WA target channel adapter

Figure 7.20 infinitiand Switch Fabric

- **Subnet: A** subnet consists of one or more interconnected switches plus the links that connect other devices to those switches. Figure 7.20 shows a subnet with a single switch, but more complex subnets arc required when a large number of devices are to be interconnected. Subneis allow administrators to confine broadcast and multicast transmissions within the subnet.
- **Router:** Connects HIM Rand subnets, or connects an Infiniband switch to a network, such as a local area network, wide area network, or storage area network.

The channel adapters are intelligent devices that handle all **1/0** functions without the need to interrupt the server's processor. For example, there is a control protocol by which a switch discovers all I'CAs and FCAs in the fabric and assigns logical addresses to each. 'Ibis is done without processor involvement.

The I n finiband switch temporarily opens up channels between the processor and devices with which it is communicating. The devices do not have to share. a channel's capacity. as is the ease with a bus-based design such as PCI. which requires that devices arbitrate for access to the processor. Additional devices are added to the configuration by hooking up each device's TCA to the switch.

InfiniBand Operation

Each physical link between a switch and an attached interface (1-ICA or WA) can he support up to logical channels, called **virtual lanes.** One lane is reserved for fabric management and the other lanes for data transport, Data are sent in the

form a stream of packets, with each packet containing some portion of the total data to be transferred, plus addre ssing and control information. Thus, a set of communications protocols are used lu manage the transfer of data, A virtual lane is temporarily dedicated to the transfer of data from on,.. end node to another over the 1nCiniBand fabric. The InfiniBand swilch maps Inatic horn an incoming lane to an outgoing lane to route the LiM41 between the desired clad points.

Figure 721 indicates the logical structure used lo support exchanges over tnrinikand. Ter account for the fact that some deviccs can send data faster than 'temporarily buffers excess outbound and inbound data. The queues can tie located in the channel adapter or in I he attached deyice'.s memory. A separate pair of queues is used fot each virtual lane, The host Use.s these queues in the following fashion. The host places a transaction, called .a work queue entry (WOE) into either the send or receive queue. of the queue pair. The two most imporianl WQLs are.SEND and RECF I VE. Bear a SEND operation, the WOE specifies a Hock of data in the device's memory space for the hardware lo send to the destination. A RECH i'v WOE specifies where the hardware is ti place data received from rancrther dcviec when that cons.urner executes a SEND operation. 'The channel adapter processes each posted WOE in the proper prioritized order and gclierite, a completion queue entry (COE) to indicate the completion status.

Figure 7,21 also indicates that a layered protocol architecture is used, consisting ()I' four lavers!

- Physical: The physical-laver specification defines three]ink speeds (1X. 4X, and 12X) giving transmission rates of 25. I [1, and 30 (ihps, respectively {Tableh 7.4). The physical layer also defines the physical media, including copper and optical fiber,
- Link: This layer defines the basic packet slructure used to exchange data. including an addressing scheme that assigns a unique link address to every device in a subnet. This level includes the. Logic for setting up viritiai lanes and For .swi tching data through switches from source lo destination within a subnet. The packet structure includes an error de14,:ei ion code to provide reliability.
- Network: The network laver routes packets between different]nfiniBand subnets.
- Transport: The transport layer provides reliability mechanism for end-to-end transfer of packets across one nr more subnets.

Cable 7.4 iafiniR anti riks and Data Throughput Ratc.

Link	Sinai rate runidirtstionail	Usable capacity (80'...6 or si gnal rate)	Effective data throughput (scud + receive)'
1- wide	2.5 Gbps	2 (illps (250 !clips)	(250 + 251.1)
	lu Ghps	ii Gbps (1 Chips)	(1 F I) OBps
	30 °bps	2-1 tlims t3 GBps)	(3 -I • .3) Oho

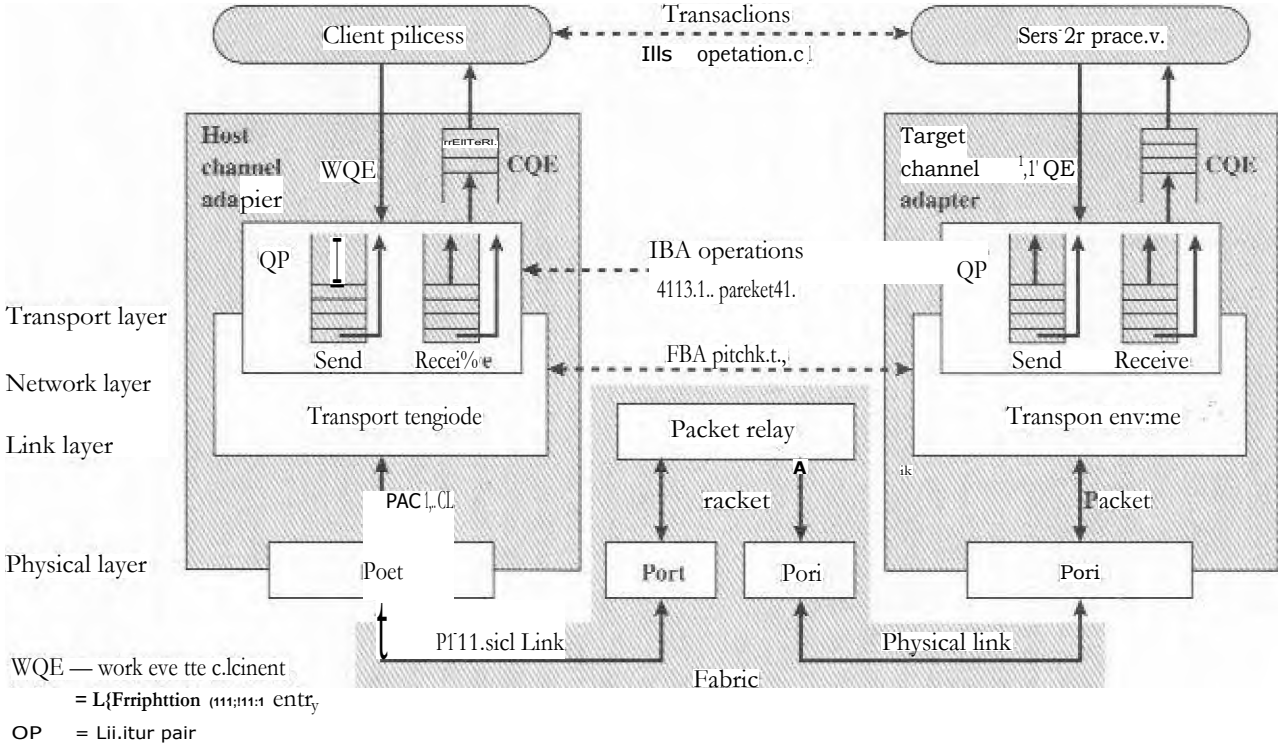


Figure 7.21 InfiniBand Communication Protocol Stack

RECOMMENDED READING AND WEB SITES

A:Pre 'O'O

A good discussion orI ntel 110 modules and arcliitt'ciu fQ, including the 82C5gA and 82C55A, can I7t [mind in I fiREV001.

HroWir,2 is covered in great &tail in [ANDE98]. [WICK97] and [11101\4001 provide a ciimeise twerviews of FireWirc. .

Inriniannd is covered in great de(ail in [I'7L]TR01 1.1 KAGA011providcs a concise overvic v.

ANDE98 Anderson, *FireWirr System A t•arlre.* Reading, MA: Addii.on-Wesley. 49'.K

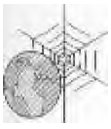
m0-x0n Brey, B. *The Iwell N't icro 808.0.18066, 8018618P I 88, M.F22.56. SOPA 80486, **Pen, hi.F?1, Pent PM** um? Prn r!0.1.2 Proc essom* Upper Saddle River, NJ; Prentice Halt. 2001.1,

FUTRO1 Futral. *W. IP.107 ifkold Arch:ow...rem Duvefopenc frx wmr e,ilro:inent.* Hillsbori OR: Intel Pntss, 2,1101.

KAGA.01 Kagan, M. ¯ infinif3arid; CPutsidi2 the lox Design." Communiciaimiq Symon .Suim:fni her 200 L I'vrorvr ,csdinag.com)

THO111011 niuMpSolk, D. "1 KEE 1394; Changing tIi Way We Do Multitnedia Comirmni-calitim," *itimedia,* April-June '2A)r).

WICK97 Wickelgren, 1. "The Facts About FircWii ." *IEEE Sp cc? Imo* .Apfil 1997.



Reu.otriniendd Web Sites:

- **TN** filmic Mt; ' is a Technical Corranilitl4 411 the National C'urninittte oii rnfor-Illation Technology Standards: .,.111E1 is responsible .14.1 lowv:r-lov61 interfaces. Its princi-pill work is the. Small Computer Sy.itein Int e rface (SCSI).
- 094 Trade Assodation: Includet' information and vendor pointers on FireWire.
- **Infinibnod Ira & Association:** Includes technical. information and vcndor pointers On Infiniband.

7.9 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

st{ta	1/0 cliarmoi	nuiliipti..2:tor.cha uM!
direct nicitiory cct!ss. I I }P,.1...\.)	110 coinniand	1/0
FircWire	1/0 !nodule	peripheral de.vicc.
	I/O proc,2;qor	p rogranarric.d I/O
	isolated I10	k.electoi chainiel
interrupt driven 110	;;emery-niappcd 110	serial 110

Review Questions

7.1 List three broad classifications of external. or peripheral, devices.

7.2 What is the International Reference. Alphabet?

73 What are the major functions of an module?

7.4 List and briefly define three techniques for performing I/O.

7.5 What is the difference ber.veen memory-mapped I/O and 'isolated IiO'l

7.6 When a device interrupt occurs, how does the processor determine which Linda issued the interrupt?

7.7 When a DMA module takes control of a bus, and while it retains control of the hi. What does the processor do?

Problems

7.1 In Section 7.3, one advantage and **One disadvantage** of memory-mapped I/O. comp:T ...I with isolated I/O. were listed. List two more advantages and two more disadvaara.o.

7.2 In virtually all systems that include DMA modules. DMA access to main memoiv given higher priority than CPI: access to main memory. Why?

7.3 Consider a disk system with 960 512-byte sectors per track and assume the disk at 3600 r pm. A processor reads one sector from the disk using interrupt-d riven with I'm:. interrupt per byte. If it takes 2.5 (I; to process each interrupt, what perccni age of the time will the processor spend handling 110 (disregard seek time)?

7.4 Repeat Problem 7.3 using DMA. and assume one interrupt per sector.

7.5 A DMA module is transferring characters to memory using cycle stealing, from a device transmitting at 9600 bps_ 'The processor is fetching instructions at the rate ci 1 million instructions per second (1 MIPS), By how much will the processor bestowed down due to the DMA activity?

7.6 A 32-bit computer has rwo selector channels and one multiplexor channel. Each selw for channel supports two magnetic disk and two magnetic tape units. The multiplexor channel has two lino printers. two card readers, and 10 VDT terminals connectedly it. Assume the following transfer rates:

Disk drive	800 '<Bytes's
Magnetic tape drive	200 KBylesis
Line printer	6.6 '<Bytes's
Card reader	1.2 KBytesis
VDT	1 I<Bylesis

Estimate the maximum aggregate **I70** transfer rate in this system.

13 A computer consists of a processor and an 110 device **D** connected to main **MI** ory M via a shared bus with a data bus width of one word. The processor can ex& cute ki maximum of 10' instructions per second. An average instruction requirofive machine cycles, three of which use the memory bus. A memory read or write operation uses one machine cycle. Suppose that the processor is continuously execunn3 "background" programs that require 95% of its instruction execution rate but not any instructions. Assume that one processor cycle equals one bus cycle. Now • • suppose the. device is to be used to transfer very large blocks of data betwan M and D,

a. if programmed is used and each one-word I'D transfer requires the promsw to execute two instructions. estimate the maximum I/O data-transfer rate, in wort per second, possible through **D.**

b. Estimate the same rate if DMA is *used.*

7.8 A data source produces 7-bit IRA characters, to each of which is appended a parity bit. Derive an expression feet the maximum effective data rale (rate of IRA data hits) over an k-bps line for the following:

a. Asynchronous transmission, with u I.5 unit stop bit

b. Bit-synchronous transmission, with a frame consisting of 48 control hits and 128 information hits

Same as (b), with a 1024—bit information field

d. Character synchronous. with 9 control characters per fraMe and I information character'

e. Same as (d), with 12 information characters

7.9 The following problem is based on a suggested illustration of 110 mechanisms in E CKE 90] (Figure 7.22):

Two boys are playing on either side of a high fence, One of the boys, named Apple-server. has a beautiful apple tree loaded with delicious apples growing on his side of the fence; he is happy to supply apples to the other boy whenever needed. The other boy, named Apple-eater, loves to ez1(apples but has none. In Fact. he must eat his apples at a fixed rate (an apple a da• k vups the doctor away). If he eats them faster than that rate. he will get sick. If he eats them slower, he will suffer malnutrition. Neither boy can talk. and so the problem i4 to get apples from Apple-server to Apples-eater at the comet. rate.

u. Assume that there is an alarm clock sitting on lop of the fence and that the clock can have multiple alarm settings. **How** can the clock he used to solve the problem? Draw a timing diagram to illustrate the solution.

b. Now assume that there is no alarm clock. Instead Apple-cater has a flag that he can wave whenever he needs an apple, Suggest a new solution. Would it he helpful for Apple-server also to have a flag? if so. incorporate this into the solution, Discuss the drawhacks of this approach.

e. Now lake away the flag and a'—unie the existence of a long piece of string. Suggest a solution that is superior to that of (Γ9 using the string.

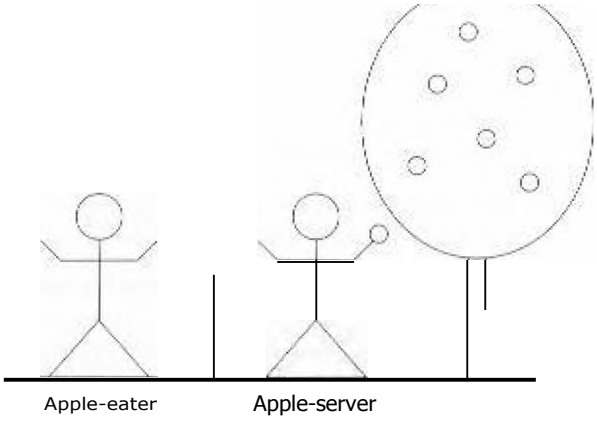


Figure 7.22 An Apple Problem

7.10 As:.4tinic that one 16-hit and two 8-bit microprocessors *M* to he interfaucci to a sys-
te i bus. The following details are given:

1. Ail microprocessors havc the hardware features necessary for any type of a transfer; prugraninied PC, interrupt-driven 110. and DrvIA.
2. Ail microprocessors bave 16-bit addrns bus.
1 'iwo memory boards, each of 64- KByt *Qapacity.* are interfaced with ilk': hus. designer wishes to use a shared mcmcir... that is as large as possible.
4. The system. bus supports a maximum of four interrupt lines *And* one Dlie1A

Make .kiny other assumptions nt...cessary, and

i ye the. Systrm bus specifications atems of number and types of lines.

b. Describe possible protocol for communicating on tha ns. i.e., read:vent intcrrupt, and DMA sequences.
c. Explain how the aforerrictitioned devices tr interfaccd Lo the s!..rste m bus.

SourcT; IA1.EX93]

CHAPTER

OPERATING SYSTEM SUPPORT

8.1 Operating System Overview

Operating System 0*.ctivcs **l'unciimis.**
'r!..Des of Operating Systems

Schedilliog

1,ong-Tcrin Scheduling
Mediuni-Terrn Scheduling
Short-Term Sched u ling

8.3 Memory Management

Swapping
Partitioning
Paging
Virtual Memory
Translation Lookaside Buffer
tnerimean

8.4 Pentium H and PowerPC Memory Management

Pentium II :Viernory Management kinrdwarc
PowurPC rvJemory-Pviana gurneTat **Hardware**

8.5 Recommended Reading and Wel) Sites

8.6 Key Terms, Review Questions, and Problems

Key Terms
Review Q e5;0.cyri.!.,
Ptoblems

KEN POINTS

- The operating system ((:).S) is the software that controls the execution of programs on a prOcussor and that manages the processor's resources. A number of the functions performed by the OS, including process scheduling anti memory management, can only he perfOrmed efficiently and rapidly if the proce ˙-sor hardware includes capabilities to support the OS. Virtually all processors include such elipa bilities to a greater or lesser extent. including virtual memory management hardware and process management hardware. The hardware includes special-purpose registers and buffers, as well as circuitry to perform, basic resource management tasks.
- One of the most important functions of the OS is the scheduling of processes, or tasks. The. OS determines which process should run at any given time. Typically, the hardware will interrupt a running process tone time to time to enable the OS to make a new scheduling decision so as to share processor time fairly among a number of processes.
- Another important OS function is Memory management. Most contemporary operating systems include a virtual memory capability, which has two benefits: {1) A process can run in main memory without all of the instructions and data for that program being present. in main memory at one lime, and (2) the total memory space available to a program may far exceed the actual main memory on the system, Although memory immagemern is performed in software-, the OS relies on hardware support in the processor, including paging and segmentation hardware.

Although the focus of this text is computer hardware, there is one area ofsoft-

»:nagesthe computer's opera ing ... teul The

. . .**Ot r** a ting system is a program that

vides services lon programmers, and schedules the execution of other programs. Some understanding of operating systems is essential to appreciate the mechanisms by which the CPU controls the computer system. In particular, explanations of the effect of interrupts and of the management of the memory hierarchy are hest explained in this context.

The chapter begins with an overview and brief history of operating systems. The hulk of the chapter looks al the Iwo operating system functions that are most relevant to the study of computer organization and architecture: scheduling and memory management.

8.1 OPERATING SYSTEM OVERVIEW

Operating System Objectives and Functions

An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware. It can he thought of as having two objectives:

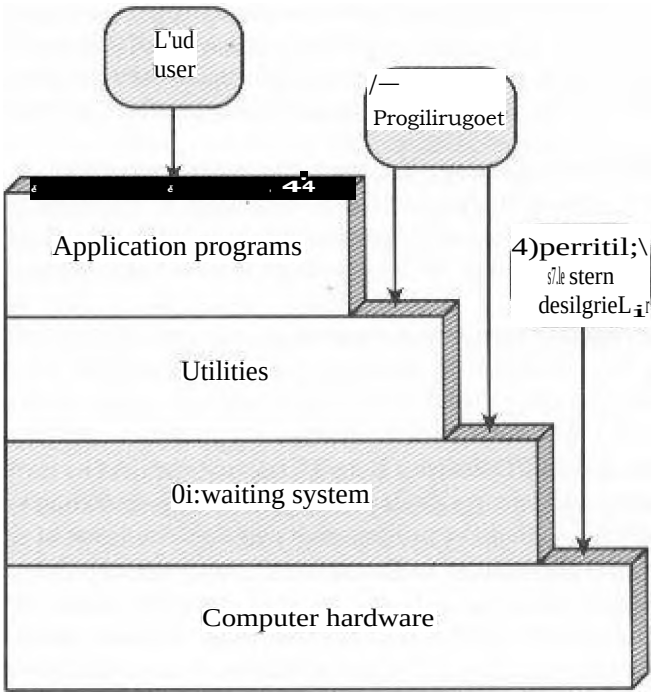


Figure S.1 I ,aver.3 and Views of Li Corripatcl · System

- Convenience: An operating system makes a computer more convenient to use.
- Efficiency: An operating system allows the computer qystem resources to be used in an efficient manner.

Let us examine these two aspects of an operating system in turn.

The Operating System *us* u UseriCtimputer **Interface**

The hardware and software used in providing applications to a user can he viewed in a layered or hierarchical Cashion, as depicted in Figure 8.1, The user or those applieal ions. the end user, generally is not concerned with the computer's 3relnieeture. Thus the end user views a **computer** system in Lerms of an application. That application can be expressed in a programming ianguage and is developed by an application programmer. IC one were to develop an application program as a set or ɾiToce,sor instructions that is completely responsible for writrolling the computer hardware, one would be faced with an overwhelmingly complex **task.** To ease this task, a **set of sysiems pre gnim:s** is provided. Some of these programs **are** recrrod to as utilities. 1 hesu implement frequently used func t ions that :mist in program creation, the management of files. and the control or 1. ¹0 devices. A programmer will make use of these facilities in developing an application, and the **application,** while it is running, will invoke L he utilities to perform certain functions. **The** most important system **program** is the operating system. The op erating system masks the details

of the hardware from the programmer and provides the programmer with a convenient interface For using the system. It acts as mediator, making it easier for Ihe pro• grammer and for application programs to access ancl I hose facilities and surviees.

Briefly, the operatic g system typically pr4 F:'i dCh SCTVI CCS in the following areas:

- **Program creation:** The operating system provides a variety of facilities and services. such as editors and debuggers, to assist the programmer in creating programs. Typically, these services are in the form **of** utility programs that ar e not actually part of the operating system but are acccssible through the uperating system.
- a **Program execution: A number** of tasks need to be performed to execute ¢F pro. gram. Instructions and data must **be** loaded ink) main memory, devices and files must be initialized, and other resources must be prepared. The oper• acting system handles all of this for the user.
- Access in 1.10 devices, kach I O device requires its own peculiar set of instrue• tions or control signals for operation. The operating system takes care **of** the. details so that the programmer can think in terms. of simple- **reads and** writes.
- g **Controlled access** *to* **files! In** the case of files, control must include an understanding of not only the nature of the **1/0** device (disk drive. tape drive) but also the file format on Ihe storage medium. Again, the **operating** system worries about Ihe details. **Farther,** in the a system with **multiple** simultaneous users, the operating system can provide protection mechanisms to control access to the files.
- **System** access: In the case of a shared or public system, the operating system controls access to the system as a whole and to specific system resources, The access function must provide protection of resources and data from unauthori•.cd users and roust resolve conflicts •or resource contention.
- **Error detection and response:** A variety of errors can occur while a computer system is running, These include internal and external hardware errors, such as a memory error, or a device failure or **malfunction;** and various software errors, such as arithmetic overflow, attempt to access forbidden memory loca• Lion, **and** inability of the operating system to grant the request or an applic,a• tion. In each cwic, the operating system must make the response that clears the error condition with the least impact on running applications. The response may range from ending **the program** that caused the error, co retrying the operation. to simply reporting the error to the application,
- ■ **Accounting: A good operating** system will collect usage statistics for various resources and monitor performance parameters such as response time. **On** anp system, this information is useful in anticipating the need for future enhancements **and in tuning** the system to improve performance. On a **multiuser** system. the information can be used for billing purposes.

The Operating System! as Resource Manager

A wmputer is a set of resources for the movement, storage, and processing of data **and for** the control of these functions. The operating system is responsible for managing these resources.

Can we say that it is the operating system that controls the movement, storage. and processing of data? From one point of view, the answer is yes: Hy managing the computer's resources, the operating system is in control of the computer's basic. *functions.* But this control is exercised in a curious way. Normally, we think *of* a control mechanism as something external to that which is controlled, or at least as something that is a distinct and separate part of that which is controlled. (For example, a residential heating system is controlled by a thermostat. which is completely distinct from the heat-generation and heat-distribution apparatus.) This is not the case with the operating system. which as a control mechanism is unusual in two respects;

- The operating system functions in the same way as ordinary computer software; that is. it is a program executed by the processor.
- The operating system frequently relinquishes control and must depend on the processor to allow it to regain control.

The operating system is, in Iaet, nothing more than a computer program. Like other computer programs, it provides instructions for the processor. The key difference is in the intent of the program. The operating system directs the processor in the use of the other system resourees'and in the timing of its execution of other programs. But in order for the processor to do any of these things, it must cease executing the operating system program and execute other programs. Thus, the operating system relinquishes control for the processor to do some "useful" work and then resumes control long enough to prepare the processor to do the next piece of work. The mechanisms involved in all this should become clear as the chapter proceeds.

Figure 8,2 suggests the main resources that are managed by the operating system. A portion of the operating system is in main memory. This includes the kernel, or nucleus, which contains the most frequently used functions in the operating system and, at a given time, other portions of the operating system currently in use. The remainder of main memory contains other user programs and data. The allocation of this resource (main memory) is controlled jointly by the operating system and memory-management hardware in the processor. as we shall see. The operating system decides when an 110 device can be used by a program in execution. and controls access to and use of files. The processor itself is a resource, and the operating system must determine how much processor time is to be devoted to the execution of a particular user program. In the case of a multiple-processor system, this decision must span all of the processors.

Types of Operating Systems

Certain key characteristics serve to differentiate various types of operating systems. The characteristics fall along two independent dimensions, The first dimension specifies whether the system is batch or interactive, In an *interactive* system, the useeprogrammer interacts directly with the computer, usually through a keyboardidisplay terminal, to request the execution of a job or to perform a transaction. Furthermore, the user may, depending on the nature of the application, communicate with the computer during the execution of the job. A *batch* system is the opposite of interactive. The user's program is batched together with programs

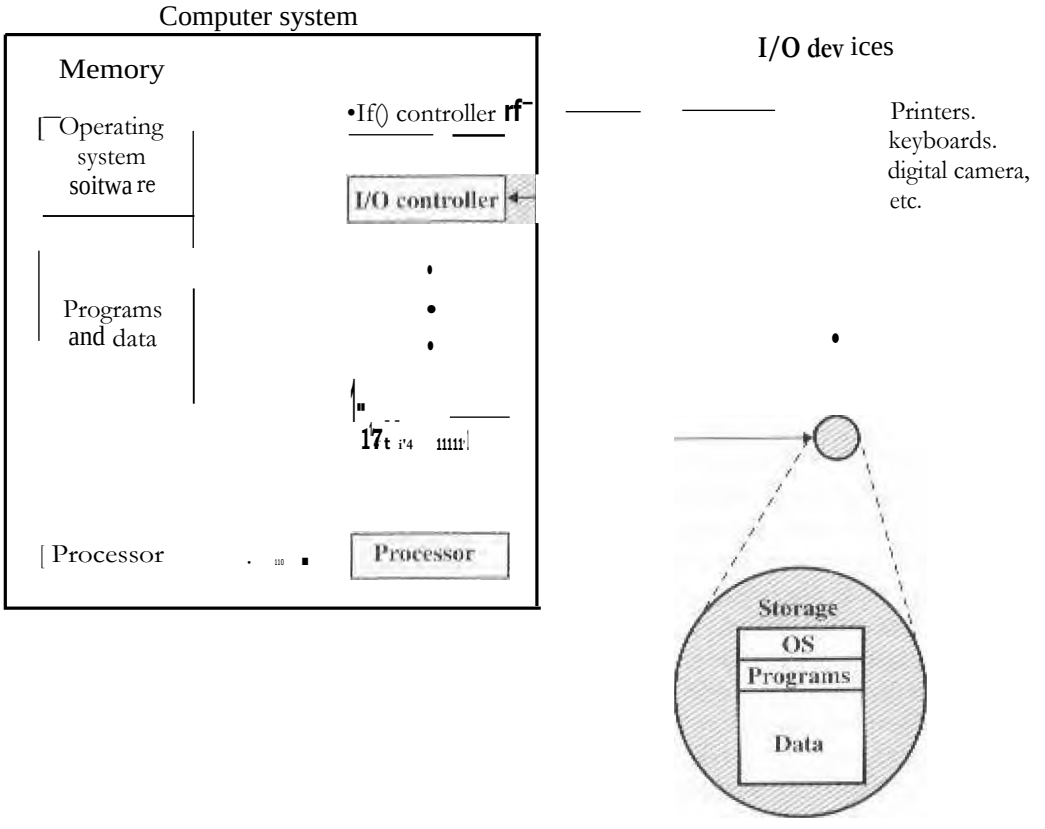


Figure K2 The Operating System as Resource Manager

from other users and submitted by a computer operator. After the program is completed, results are printed out for the user. Pure batch systems are rare today. However, it will be useful to the description of contemporary operating systems to examine batch systems briefly.

An independent dimension specifies whether the system employs *miA-programming* or not. With multiprogrammina, the attempt is made to keep the processor as busy as possible, bv having it work on more than one program at a time. Several programs are loaded into memory, and the processor switches rapidly among them. The alternative is a *uniprogramming* system that works only one program at a time.

Early Systems

With the earliest computers, from the late 1940s to the mid-1950s. the programmer interacted directly with the computer hardware; there was no operating system. These processors were run from a console, consisting of display lights, toggle switches, some form of input device, and *i* printer. Programs in pt ocessor code were loaded via the input device (e.g., a card reader). If an error halted the program , the

error condition was indicated by the lights. I he program In r could proceed to examine registers and main memory to determine the cause of the error. If he program proceeded to a normal completion, the output appeared on the printer.

These early systems presented Iwo main problems:

- **Scheduling:** Most installations used a sign-up sheet to reserve processor lirric. Typically, a user could 'sign up for a Nock of time in multiples of a half hour or so. A user might sign up for an hour and finish in 45 minutes, this would result in wasted computer idle time. On I he other hand, the user might run into problems, not finish in the allotted time, and be forced to stop before resolving the *probtem.*
- Setup time: A single program, celled *a* **job, could** involve loading the compiler plus the. high-level language program (source program) into memory. saving the compiled program (object program). and then loading and linking together the object program and common functions. Each of these steps could involve mounting or dismounting tapes, or setting up card decks_ lf an error occurred, the hapless user typically had to go back to the beginning of the setup sequence. Thus a considerable amouni of time was spent just in setting up the program to run.

This mode of operation could he termed serial processing, refleding the fact that users have access to the computer in series. Over time, various system software tools were developed to attempt 10 make serial processing more efficient. These include libraries of common functions . tinkers, loaders, debuggers, and 44river routines that were available as common software for alt users.

Simple Butch Systems

Early processors were very expensive, and therefore it was important to maximize processor utilization. The wasted time due to scheduling **and** setup time was unacceptable.

To improve utilization, simple batch operating systems were developed. With such a system, also called a *mi.pnimr,* the user no longer hAs direct access to the processor. Rather, the user submits the job on cards or tape to a computer operator, who *benches* the jobs together sequentially and places the entire **batch on an input** device, for use by the monitor.

To understand how this scheme works,]et us look al it from Iwo poiniw of view: that of the monilor and that of the processor. From the point of view of the monitor. it is the monitor that conlrols the sequence **of** events. For this to he so, much 0r the monitor must always be in main memory and avui lable for execution (Figure 83). That portion is referred to as the **resident monitor.** The rest of the monitor consists of utilities Lind common functions that are loaded as subroutines to the user program at the beginning of any job that requires them. The monitor reads in jobs one at a time from the input device (typically a card reader or magnetic tape drive). As it is read in, the eurren1 job is placed in the user program area, and control is passed to this job. When the lob is completed, it rei urns control to the monito•, which immediately reads in the next job. The results of each job arc printed out for delivery to the user,

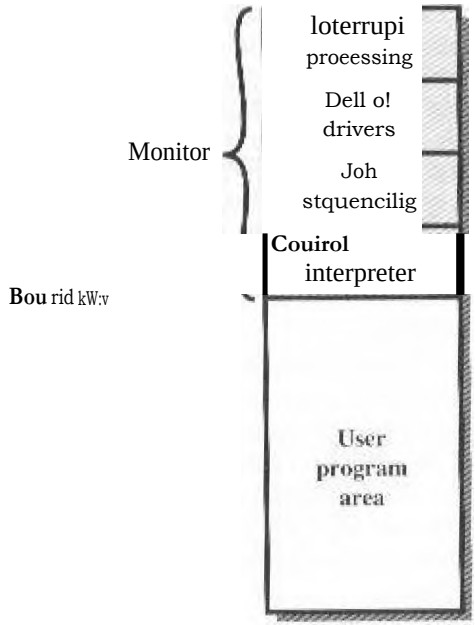


Figure 8J Memory Layout for li Resident Monitor

Now consider this sequence from the. point Of view of the processor, At a c.a. Lain poini in time, the processor k executing instructions from the portion of main memory containing the monitor. '1 hose instructions cause the next job to be mid m to another portion of main mentor/. Once a job has been read in, the proce;ssor encounter in the monitor a branch instruciion that instructs the processor to co. tin= execution 4it the start of the user pr(4,,'ram.'Vhe processor will then executt:. the instruction in the user's program until it encounters an ending or error conditioo, Eil **hcr** event causes the processor to fetch its next instruction from the monitor' program. Thus the phrase "control is passed to a job" simply means that ihe proccs• sor rs now fetching and executing instructions in a user program. and "control is returned to the monitor" mcins thus the processor is now retching and executing instructions from he monitor program,

It should bu clear that the monitor handle4 the scheduling problem. A batch of jobs is queued up. tine] jobs are executed as rapidly as possible, with Ito inlerven- ing idle time.

How about the job setup tune? The monilor handles this as welt With each job, instructions zkre included in a job control language (JCL). Phis is a specW txpe 01 programming language used to provide instructions to the monitor, A simply exam* is or a user submitting a program written in FOR'PRAN plus some data to be used by the program. Each FORTRAN instruction and each item of data is on a sep4irate punched card or a separate record on tape. in addition lo FOR. "I RAN and data lines. the job includes job control instructions. which are denc.iie by the beginning **The** overaii format of the job looks Like this:

```
$FTN
•
• FORTRAN instructions

$LOAD
5 RUN

• Data
•
```

To execute this job, the monitor reads the $1 'TN line and loads the appropriate compiler from its mass storage (usually tape). The compiler translates the user's program into object code, which is stored in memory or mass storage. **If** it is stored in memory, the operation is referred to as "compile, load, and go." If it is stored on tape: then the $LOAD instruction is required. This instruction is read by the monitor, which regains control after the compile operation. The monitor invokes the loader, which loads the object program into memory in place of the compiler and transfers control to it. In this manner, a large segment of main memory can be shared among different subsystems. although only one such subsystem could be resident and executing at a time.

We see that the monitor, or batch operating system. is simply a compujter program. It relies on the ability of the processor to fetch instructions from various portions of main memory in order to seize and relinquish control alternately. Certain other hardware features are also desirable:

- **Memory protection:** While the user program is executing, it must not alter the memory area containing the monitor. If such an attempt is made, the pruccssOr hardware should detect an error and transfer control to the monitor. The monitor would then abort the job, print out an error message, and load in the next job.
- **Timer** A timer is used to prevent a single job from monopolizing the system. **The** timer is set at the beginning of each job. If the timer expires. an interrupt occurs, and control returns to the monitor.
- **Privileged instructions:** Certain instructions are designated privileged and can he executed only by the monitor. If the processor encounters such an instruction while executing a user program, an error interrupt occurs_ Among the privileged instructions are instructions. so that the monitor retains control of all I/O devices. This prevents, for example, a user program from accidentally readin2 job control instructions from the next job_ If a user program wishes to perform I/O. it must request that the monitor perform the operation for it. If a privileged instruction is encountered by the processor while it is executin2 a user program, the processor hardware considers this an error and **transfers** control to the monitor.
- **Interrupts:** Early computer models did not have this capability. This feature gives the operating system more flexibility in relinquishing control to and regaining control from user programs.

Read one record From the	0.0015 seconds
Execute 1(X) instruction~	0.0001 seconds
Write one record to file	(I.0015 seconds
TOTAL	0.0031 seconds

Peri.:eist CPU utilizediott — $\frac{0.0(X.11}{0.003}$ — 012 2%

Figure SA System Oil ivation Example

Processor time alternates between execution of user programs and execution Of the monitor. There have been two sacrifices: Some main memory is now gisien over to the monitor and some processor time is consumed by the monitor. Both of these are forms of overhead. Even with this overhead, the simple batch system improves utilization of the computer.

Multiprogrammed Batch Systems

Even with the automatic job sequencing provided by ,9 simple batch operating system. the processor is often idle. The problem is that devices. are slow compared to the processor. Figure 8.4 details a representative calculation. The calculation concerns a program that processes a File of records and performs. on average. 100 processor instructions per record. In this example the computer spends over 96% of its time waiting for I/O devices to finish transferring data! Figure 8.5a illustrates this situation. The processor spends a certain amount of time executing, until it reaches an I/O instruction. it must then wait until that 1/0 instruction concludes before proceeding.

This inefficiency is not necessary. We know that there must he enough memory to hold the operating system (resident monitor) and one user program. Suppose that there is room for the operating system and two user programs. Now. when on job needs to wait for I/O. the processor can switch to the other job, which likely is not waiting for I/O (Figure 8.5b). Furthermore. we might expand memory to hok three., four, or more programs and switch among all of them (Figure 8.5c). The process is known as **multiprograttuning,** or **multitasking.** **It** is the central theme o! modern operating systems.

To illustiate the benefit of multiprogramming, let us take an example. Consider a computer with 256K words of available memory (not used by the operating system), a disk, a terminal, and a printer. Three programs, JOB1, JOB2, and JO1 .. are submitted for execution at the same time. with the attributes listed in Table 8.1. We assume minimal processor requirements for JOB2 and **.10B3** and continuous disk and printer use by JOB3. For a simple batch environment, these jobs will Ft executed in sequence. Thus. .10131 completes in 5 minutes. 3092 must wail until the 5 minutes is over, and then completes 15 minutes after that_ **.10B3** begins after 211 minutes and completes at **30** minutes from the time it was initially submitted, The

---

The term *+rrrrltitaskii* is sometimes reserved to mean multiple tasks within the same program that be handled concurrently by the operating system, in contrast to *niniiiiprughworung,* which would rci;: multiple processes (rum multiple programs, However, it is more common to equaic t he terms *ruatitasking* and *ennhiprograninung, as is* **dune in** most standards dictionaries (e,g., IEEE Sid 100-1992, **To** *New IEEE Standard Dientoktry of Pereira:al and Electroines Tema),*

C[00:1 op naill1.0SROneld L121100.0 riflOir 0_104 pug z1-1 01' Vulumssu) .rapid
-ll.103 OLI1 u! slo w ow gom 2ups !xch'yi @VI.[At glUjil Ern:111 MM ..q/g :ILL 1,14 un1u8.) gaup
its `S4.101 D1j1 troamiaq uopu altioa ooJTIOSa• SI gioqi as1ucoo2
INIIIIILLER0c..pid ginuz >3 131Purp unz axe sger ain Inn p5odcins mop.'
popncl own Di 11.1_qn1'}zu otp .] 7A0 pg213J0Au
soonosa[[p.I. 10j UOTUrilillallapUn SSOlf 2.MI11 pun luappLo 5; g9•8 ;),In'Tqd
In paymnsni[! 19.1 llowriipn ZID.Aap-eig-gZip&g(1 "Z•R alqi2j_ Jo utun/oD VuFLunini2oici
-!un. q1 LET usityLi 5; i1. sMU9 n SuodS01 pug **indOncu Li) 'u(livvin** 0.).111052.1 ,Y313.1aAle

;1Tfiwnig klual]eAoacillirmi **anau**

slutugiud Damp qiim VigutraimacudJuripsi (a)

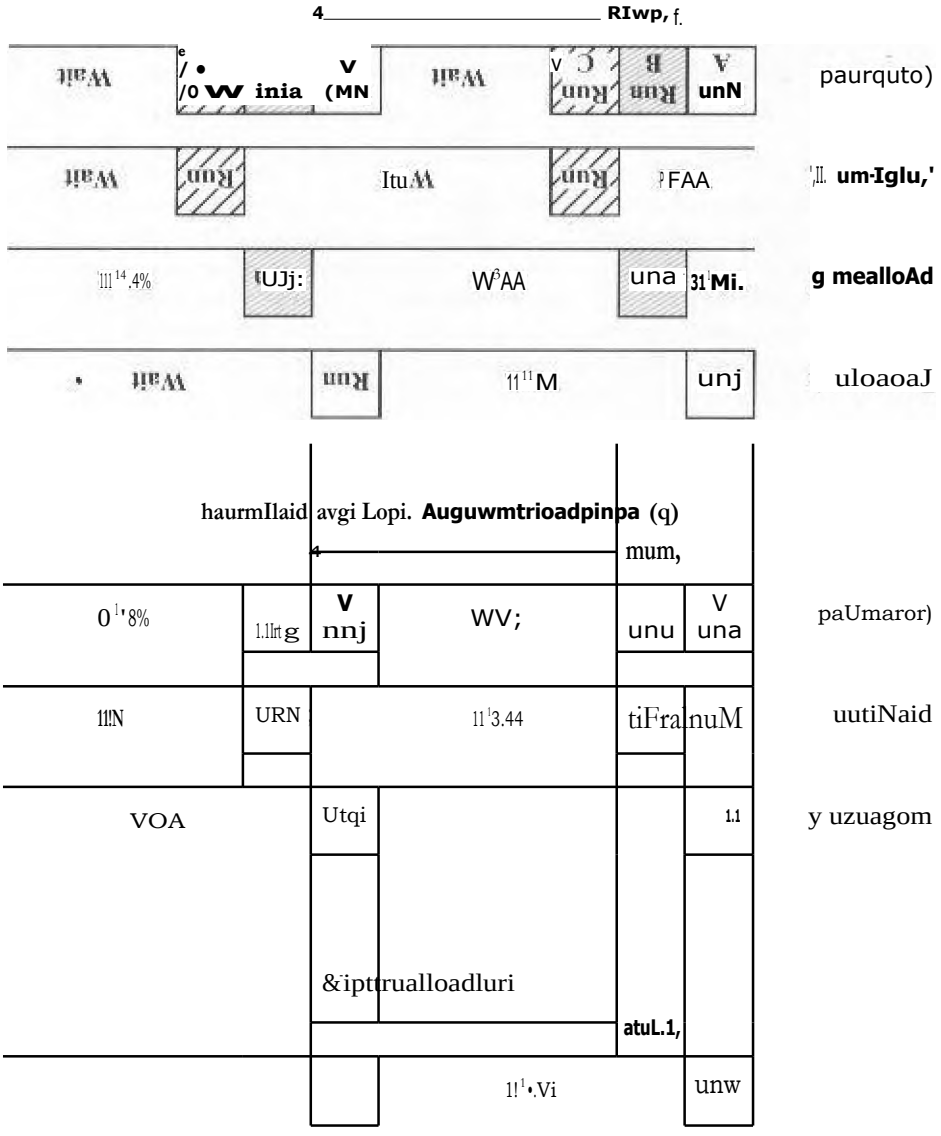


Table 8.1 Samplc. Program E..Necut.ion Attriboas

	.l 0111	;IOW	%10113
Type of job	I kavy compute	Heavy 1;0	Heavy 1;0
thiratiou	5 E'sin	15 min	L0 min
Memory ri.quired	51)1<	11.)0K.	KIK
Ne'ed ditili.?	Ni,	Eft}	9'4'S
Need lerminal?	No	Y4.'.4	No
Need printer?	Nfl	No	Y...:s

i heir input and output operations active). 10131 will still require 5 minutes to complete her Lit the end of that time, .10B2 will he one-third finished, ?l nd J0133 half finished. All three jobs will have finished within 15 minutes, .l tic improvement is evident when examining the multiprogramming column of 'l able 8,2, obtained from the histogram shown in Figure X.nh.

As with a simple batch system. a muitiprogramming batch system mull rely On certain comptiler hardware. features. Thc most notable additional feature that is useful for mull iprogramming is the hardware thai supports I/O interrupts and DMA. With interrupt-driven I/O or DMA, the processor can issue an I/O curnmanc.1 for one. joh and proceed with the excention of.another job while the I/O is carried out by the device controller. When the 1/0 operation is compicle, the processor is interroptal and control k passed to an interrupi-handling program in the operating system. The operas ink, system will then pass control to another job.

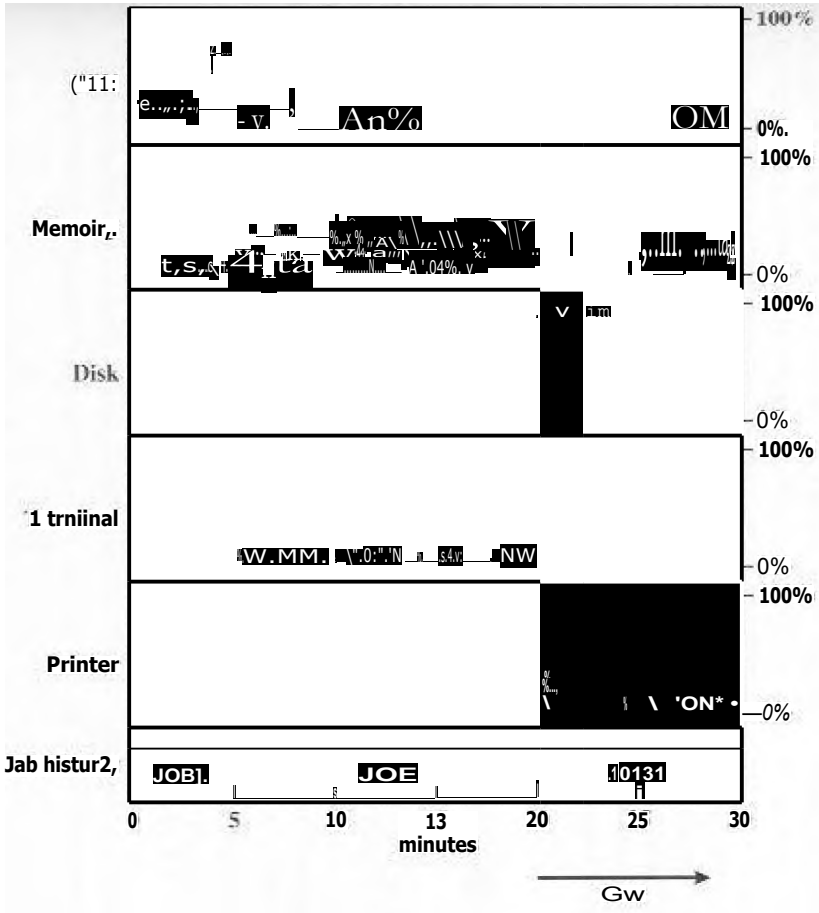
Multiprogramming operating systems are fairly sophisticated compared to single-prog,rmn, or uniprogramming.f.,ysiems. 'Vo have several jobs ready to run, the jobs must be kept in math mcmtiry , requiring some form of memory management, In addition, if several jobs arc ready to run, the processor must decide which one to run, which requires some algorithm for scheduling, Thu& euncepn.4. Lire discussed later in this chapter.

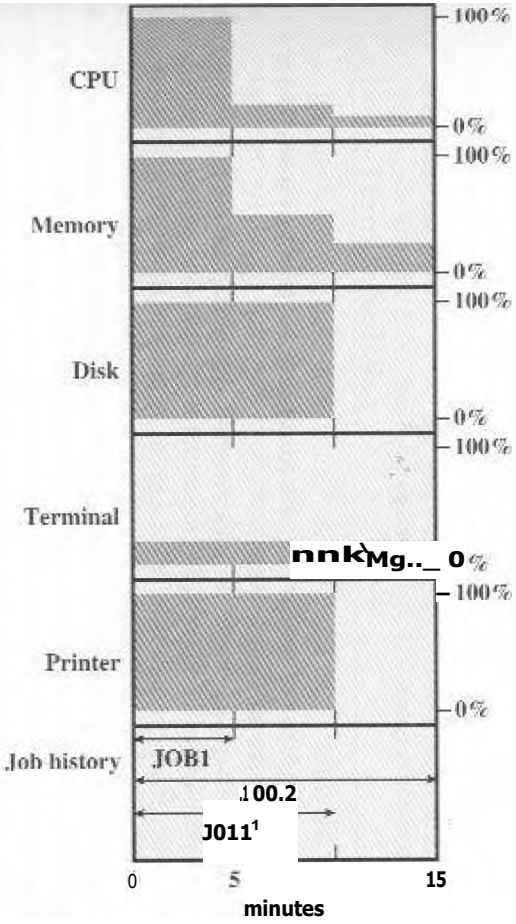
Time-Sharing Systems

With the use of mulliprogvarnming, hatch processing can he quite efficient. However, for many jobs. ii is dcsirabie to provide a mode in which the user inter.

Table 8.2 Mcts cif MulllirFrogramniing ors Resonter.,. Utilization

	Uniprogrumuning	**Mulliprogramming**
PICICeSSOr 'me	22%	43%
Memory we	'33 %	(7%
Disk use	33 :14.	(17'X.
Printer lime	33%	67%
Elapnd li **ME**	30 rain	1.5 min
'1'hroligliput rate	johEih	12 y.lbsth
Mean response **OW**	I}; rl.1111	10 min





(@ Uniprngrurnming

b) Multiprorraniming

Vigurc &A Utilization ilistoguains

Table /0 Batch Multiprograainning Time Shtiring

	Batch Multiprogramming	'fine Sharing
ohjecti.ve	Maxiinthe procoSSOV use	Minimize response Liiuo
Source of direct ive.s Lta oparasing **Hystem**	kb con **trot 19.DgU21.4** c.nrnin ids prtwilistil **With** the job	**GiDLETTjancl:i** &tiered nt the terminal

acts directly with the computer. Indeed, for some jobs, such as i msaction processing. an interactive mode is essential.

Today, the requirement for an interactive computing facility can be, and (Amn is, met by the use of a dedicated microcomputer. That option was not available in the 1960s, when most computers wcre big and costly. Instead time sharing wa;s develope6.

Just as multiprozramming allows the prmeMor to handle multiple batch ioby at a time. multiprogranuning can be used to handle multiple internaive jobs. En thi5. latter case. the technique is referred to as lime sharing, because the processor's time is shared among multiple users. In a time-sharing system, multiple Liscrs sinutltawr °ashy' access the system through terminals, with the operating sysiem interleaving the execution of each user program in a short burst or quantum of computation. Thus, if there arc fit users actively requcsting service at one time, each usei will only see On the average 1 *in* o[the effective uoinputer speed, nut counting operating s!istem overhead. However, riven the relatively slow human reaction time:. the response time on a properly desired system should he comparable to that on a ded icated computer.

Both hatch mulliproraniming and time sharing use multiprogramming. Tht key differences are listod in Table 8,3.

KZ SCHEDULING- IF

'Hie key to multiproaramming is scheduling, In Filet. four types of scheduling are typ• ically involved (Table. 8,4). We will explore these presently. Buff first, we introduce the concept of *procm. This lc* mi was first used by the designers of the [viultics opei. ating system in the I %Os. It is a somewhat more general term than *job.* Many 4.10fihitions have been given for the term *pmcess.* including

Table 8.4 Types of Sc liedulitty

1.011.E-mm	Tbe. decisici.n [15 **adC3 to** gr. pool 01 p rnce.ssos to he ckccu **tcd**
m6dium-Lean	**T** he docidon in add to LIN Tiumbof procvsys Chnt rLre poiliolly lir ft.114: it innin ineniuq
SiMet-LL 11 WIWI-10111.1g	rhe cktt i L9n a!, 1.4r which pvnilable process will t1C c.we-uted hy hc proee cr
PO **kuhodu4Nng**	The. LioLkik,n a **s 10** which process's pertainp.1.+0 requ **"I gl}211** cE6,;: d by an o};i i ibk 1.0 tii.ivice

- A program in execution
- The "animated spirit" of a program
- Thai entity to which a processor is assigned

This concept should become clearer as we proceed.

Long-Term Scheduling

The long-term scheduler determines which programs are admitted to the system for proce5sing. 'Chun, it controls the degree of multiprogramming (number of processes in memory), Once admitted, a job or user program becomes a process and is added to the queue for the shout-term scheduler. Ii some. systems, a newly created process begins in a swapped-out condition, in which case it is added to a queue for the medium-term scheduler.

In a batch system, or for the batch portion of a general-purpose operating system, **newly Kubmitici** jobs arc routed to disk and held in a hutch (ILLI21.1C. **The** long-term scheduler creates processes from the queue when it c2lli. There are two decisions involved here. First. the scheduler must decide that the operating system can take on one or more additional processes, Second , the scheduler must decide which job or jobs to accept and turn into processes. The criteria used may include priority, expected execution time, and I/O requirements,

For interactive programs in a time-sharing system, a process request generated when a user attempts to connect to the system. Time-sharing users **are** not simply queued up and kept waiting until the system can accept them. Rathe.r, the operating system will accept 11[1 horized comers until the system is sat tirffied. using some predefined measure of saturation. At that point, a connection request is met with a message indicating that the system is full and the user should try again later.

Medium-Term Scheduling

Medium-1cm scheduling is part *of* the swapping functiorli described in Section 8,3. Typically, the swapping-in decision is based on the need to manage the degree of muitiprogramming. On a system dial does not use virtual memory, memory management is also ;in issue. Thus, the swapping-in decision will consider the memory requirements of the swapped-out processes.

Short-Term Scheduling

The high-level seheduier executes relatively infrequently and makes the coarse-grained decision of whether or not to take on a new process, and which one to take. The short-term scheduler, also known as the *climwArcht.r,* executes frequently and makes the floc-grained decision of which job to execute next,

Proems States

undc•nitind the operation of the short-term scheduler, we need to consider the concept of a process state. During the lifetime of a process, its status will change a number of times. Its status at any point in time is referred to as a *stare.* The term *state* is used because id connotes that certain information exists ',hal defines the status at that point. At minimum. there arc five defined states for a process (Figure &Tr

* **New: A** program is admitted by the high-level scheduler but is not yet ready to execute. The- operating system will initialize the process. moving it to the ready state.
- **Ready:** The prc.}eess is ready to execute **and** is awaiting access to the processor
- **Runninic** The process is being executed by the processor.
- **Waiting:** The process is suspended from execution waiting for some system resource. such as W.
- **Halted:** The process has terminated and will be destroyed by the operating system.

For each process in the system, the operating system must maintain information indicating the state of the process and other information necessary for proeesi execution. For this purpose, each process is represented in the operating system by a *process control block* (Figure 8.8), which typically contains the following:

- **Identifier:** Each current process has a unique identifier.
- State: The current slate of the process (new. **ready, and** so **on),**
- **Priority:** Relative priority level.
- **Program counter:** The address of the next. instruction in the program to be executed.
- **Memory pointers:** The starting and ending locations of the process in ineinor.
- **Context data:** These are **data that are present in** registers in the processor while the process is executing, and they will be discussed in Part Three. Fat now. it is enough to say that **these** data represent the "context" of the process. The. context data plus the program counter are saved when the process leaves the ready state. They are retrieved by the processor when it resumes execu• Lion of the process.
- 110 **status information:IncludeNoutstanding1/0** requests, 1/0 devices (e.g.. tap drives) assigned to this process, a list of files assigned to the process, and soon.
- **Accounting information:** May include the amount of processor time and clock. time used time. limits, account numbers, and so on.

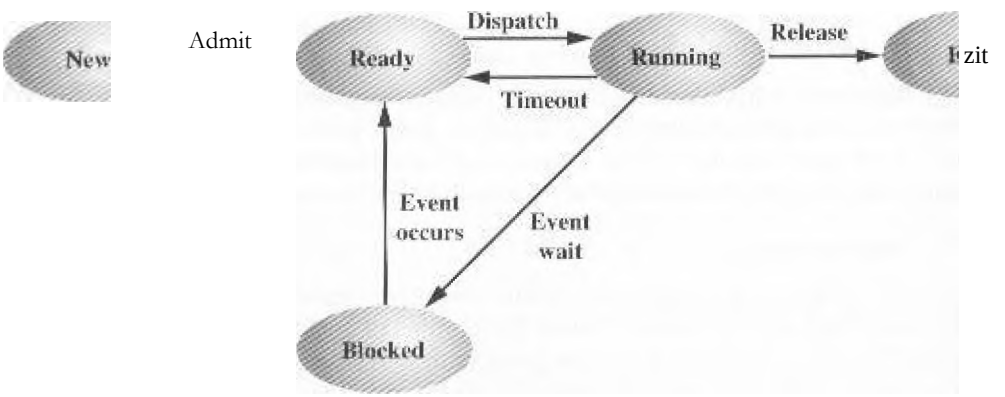


Figure 8.7 Five-State Process Model

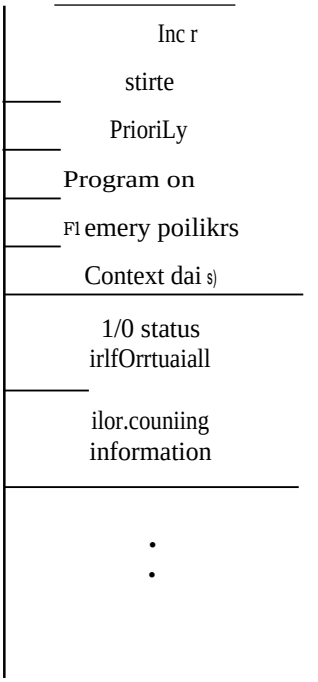


Figure 13.1'1 Process Control 131(Kk

When the scheduler accepts a new job or user request for execution, it creates a blank process control block and places the associated process in the new state. After the system has properly filled in the process control block, the process is transferred to the ready· stale.

Scheduling Techniques

To understand how the operating system manages the scheduling of the various jobs in memory, let us begin by considering the simple example in Figure S.9. The figure shows how main memory is partitioned at a given point in time. The kernel of the operating system is, of course. always resident. In addition, there are a number of active processes. including A and 1·3, each of which is allocated a portion of memory.

We begin at a point in time when process A is running. The processor is executing instructions from the program contained in A's memory partition. At some later point in time. the processor ceases to execute instructions in A and begins executing instructions in the operating system area. This will happen for one of three reasons:

1. Process A issues a service call (e.g., an 110 request) to the operating system. Execution of A is suspended until this call is satisfied by the operating system,
2. Process A causes an *interrupt.* An interrupt is a hardware-generated signal to the processor. When this signal is detected, the processor ceases to execute A and transfers to the interrupt handler in the operating system. A variety of

events related to A will cause an interrupt. One example is an error. such attempting to execute a privileged instruction. Another example is a timeout: to prevent any onc process from monopolizing the processor. each procc9s is only granted the processor fora short period at a time.

3. Some event unrelated to process A that requires attention causes an interrupt. An example is the completion of an I.O operation.

In any case, the result is the following. The processor saves the current context data and the program counter for A in A s process control block and then begins executing in the operating srstern. The operatinu swum. may perform sow work, such as initiating an NO operation. Then the short-term-scheduler portion of the operating system decides which process should he executed next. in this example, B is chosen. The operating s stern instructs the processor to restore 13's context data and proceed with the execution of B where it left off.

This simple example highlights the basic functioning of the short-term scheduler. Figure 8.1.0 shows the major elements of the operating system involved in the multiprogramming and scheduling of processes. The operating system receivo control of the processor at the interrupt handler if an interrupt occurs and at the service-call handler if a service call occurs. Once the. interrupt or service call is handled, ihe short-ierin scheduier is. invoked to pick a procesf, for execution.

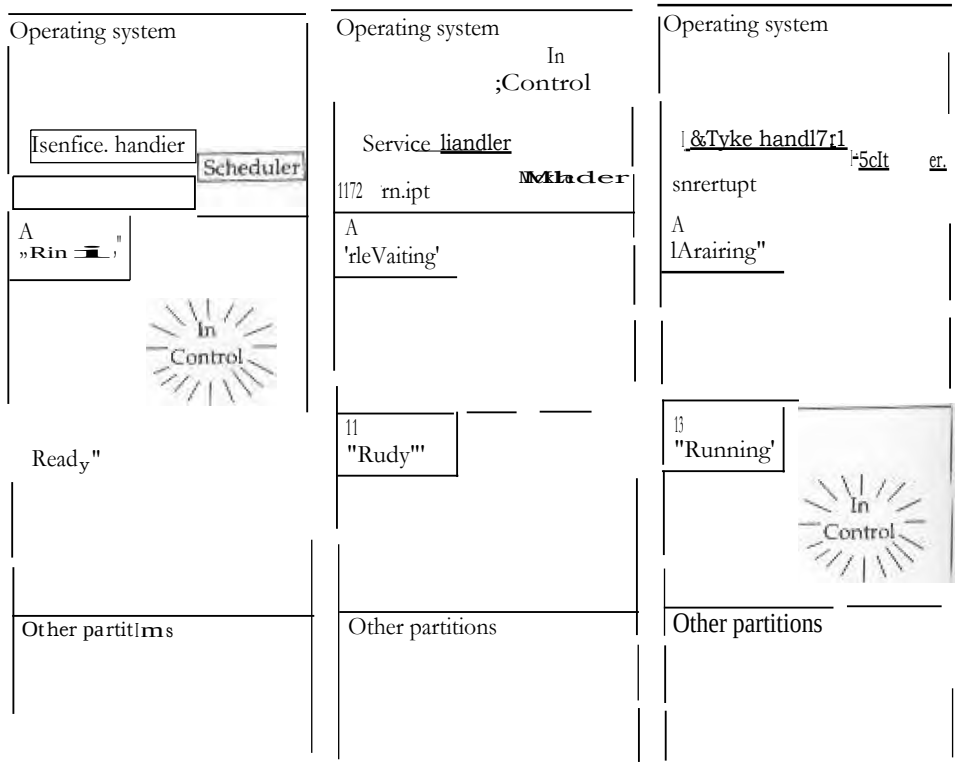


Figure 8.9 Rticduling Exarnpic

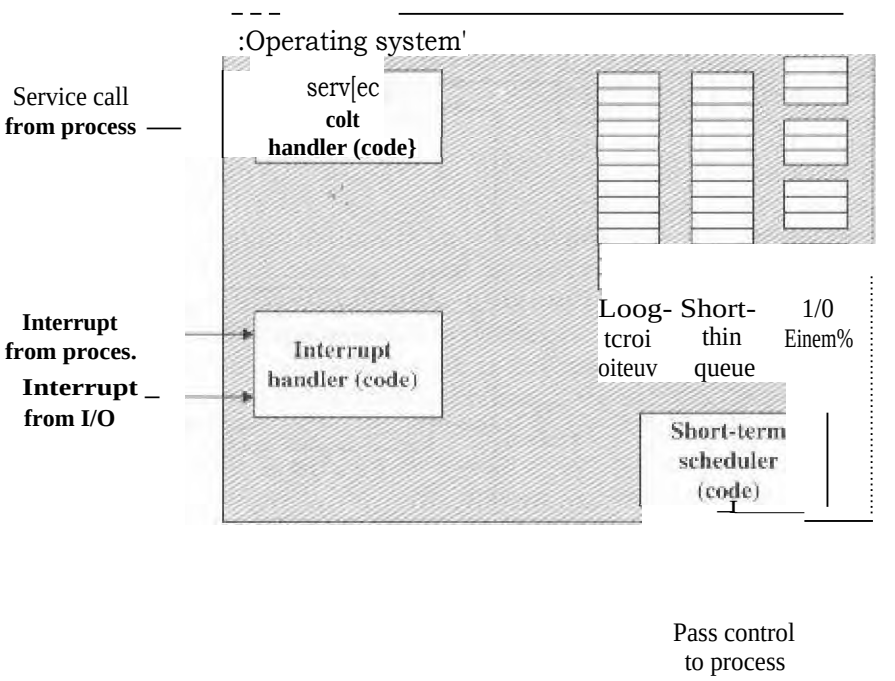


Figure 8.10 Rey Elements of tm Operating System for Muitiprog,ramming

To do its job, the operating system maintains a number of queues. Each queue is simply a waiting list of processes waiting for some resource. The *long, term queue is* a list of jobs waiting 10 use the system. As conditions permit. the high-level scheduler will allocate memory and create a process rOT 0111U cif the w ii1ing itcrns. *shori terrn queue* consists of all processes in the ready state. Any one of these processes could use the processor next. It is up to the short-term scheduler lo pick one. Generally, this is done with 4i round-robin algorithm, giving each process some time in turn.. Priority Levels may also be used there is an *//U geeveee* for each I/O device. More [Ion one process may request the. use of the same [10 device. All processes waiting to use. each device are lined up in that device`s

Figure 8.11 suggests how proCC,MS progress through the computer under the control of the operating system. Each process request (hatch job, user-defined inter-Active job) is placed in the long-term queue. As resources become available, a process request becomes a rarocvs's anti is then placed in the ready state and put in the short-term queue. The processor alternates between executing operating system instructions and executing user processes. While the operating system is in control, it decides which process in the short-term Llueue should be cxceutecl nexl. When the operating system has finished its immediate tasks. it turns the processor over to the ch.on.n process.

As w..as mcntiuncd carlicr, a process being executed may he suspended for a variety of reasons. If it is suspended becaun: the process i NuQsis 11(1 then it it placed in the appropriate queue. if it is suspended because of a timeout or

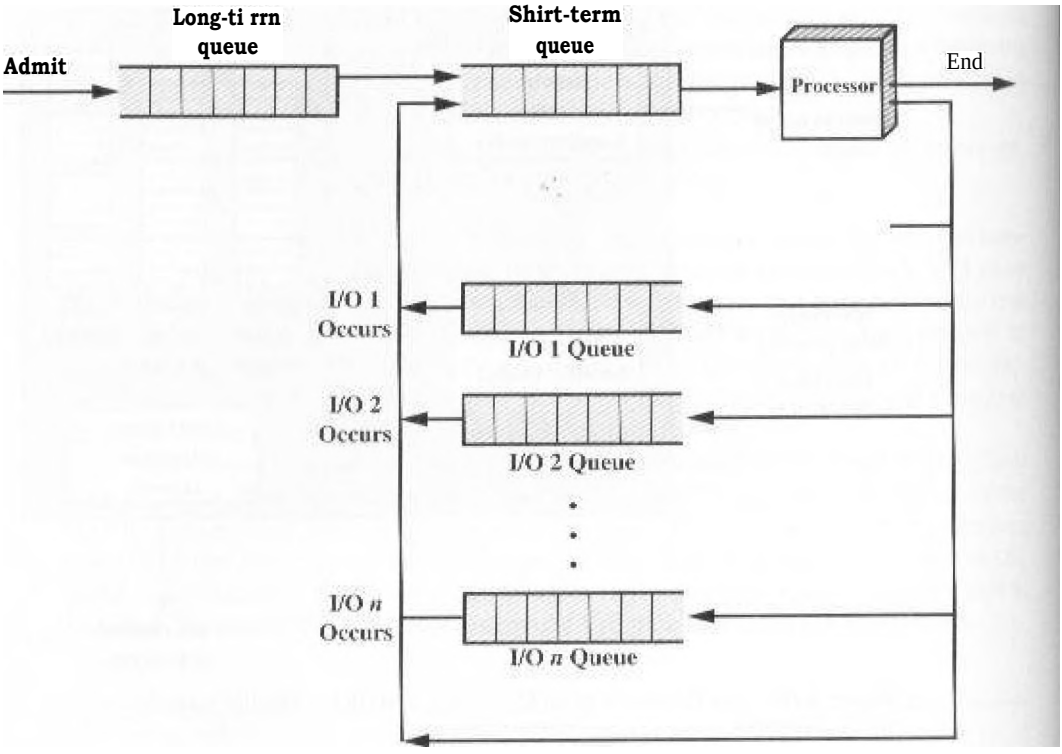


Figure 8.11 Queuing Diagram **Repres;2ritation of Processor Scheduling**

because the operating system must attend to **pressing business,** then it is placed in the ready f4 aW and put into the short-term **queue.**

we. meni ion that the operating system also manages the 1/0 queues, a n IIO **opermion is compicled,** the operating system removes **the** satisfied process from that I/O queLle a **nri** places iI **in** the short-lerm queue. It then selects **another t.441.ing process (if** any) and signals for the 1/0 device to satisfy that process's request.

$.3 MEMORY MANAGEMENT

In **uniprogramming sVstem,** main memory is divided into two parts: one part for I he operating system (resident monitor) and one part for the program currently being executed. In a multiprogramming system, the "user" part of memory is subdivided to accommodffie multiple processes. The task of subdivision is carried out dynamically by the operating system and is known as *memory mbrnagemmt,*

Effective memory management is vital in a multiprogramming s!....stain. II only a few processes are in mcmorv, then for much of Lhe time all of the processes will

be waiting for I.10 and the processor will he idle. Thus, memory needs to be allocated efficiently to pack as processes into memory as possible.

Swapping

Referring hack to Fiaire 8.11. we have discussed three types 01' w,.Le.L.Les: the long-term queue of requests for new processes, the short-1EY1T1 queue of processes ready to use the processor, and the various l/O.queues of prouesses that are not ready to use the processor_ Recall 1h.,r the reason for this elaborate machinery is that 11O activities arc much slower than computation and therefore 1he processor in a uniprogramming system is idle most of the time,

But the arrangement in Figure 8.11 does not entirely solve the problem. It is true that, in this case, memory holds multiple processes and that 11ic processor can move to another process when one process is waiting, But the processor is so much faster than 110 that it will be common *for all* the processes in memory to be waiting on I/0. Thus, even with multiprogramming, a processor could be idle mom of the lime.

What to do? Main memory could be expanded, and Mi3 he able to accommodate more processes. But there are two flaw4, in this approach. First, main memory is expensive, even today. Second, the appetite of programs for memory has grown as fast as the cost of memory has dropped. So larger memory results in larger processes, not more processes.

Another solution is *swapping,* depicted in Figme 8,12. We have a long-germ queue of process requests, typically stored on disk. These are broughl in, one al a time, as space becomes available. As processes are completed. they are moved out of main memory. Now the situation will arise that none of the processes in memory are in the ready state all are waiting on an I/O operation). Rather than remain idle, the processor Nwaps one of these processes back out to disk into an *iniermediute queue.* This is a queue of existing processes that have been temporarily kicked out of memory, The operating sysi cm then brings in another process from the inlet-mediate queue, or it honors a new process request from the long-term queue. Execution then continues with the newly arrived process.

Swapping. however, is an 1I0 operation, and therefore there is the potential/ for making the problem worse. not better. But because disk I/O is generally the fastest 110 on a system (e.g., compared with tape or printer I/O), swapping will usually enhance. performance. A more sophisticated scheme, involving virtual memory. improves performance over simple swapping. This will be discussed shortly. Bill first, we must prepare the ground by explaining partitioning and paging,

Partitioning

The simplest scheme for partitioning available memory is to use *fixed-size porritiom,* as shown in Figure &]3. Note I hat, although the partitions are of fixed size, they need not be of equal sixe. When *a process* is brought into memory, it is placed in the smallest available partition that will hold it.

Even with the use of unequal fixed-size partitions, there will be wasted memory. Jo most cases, a process will not require exactly as much memory as provided by the partition_ For example, a process that requires 3M bytes of memory would

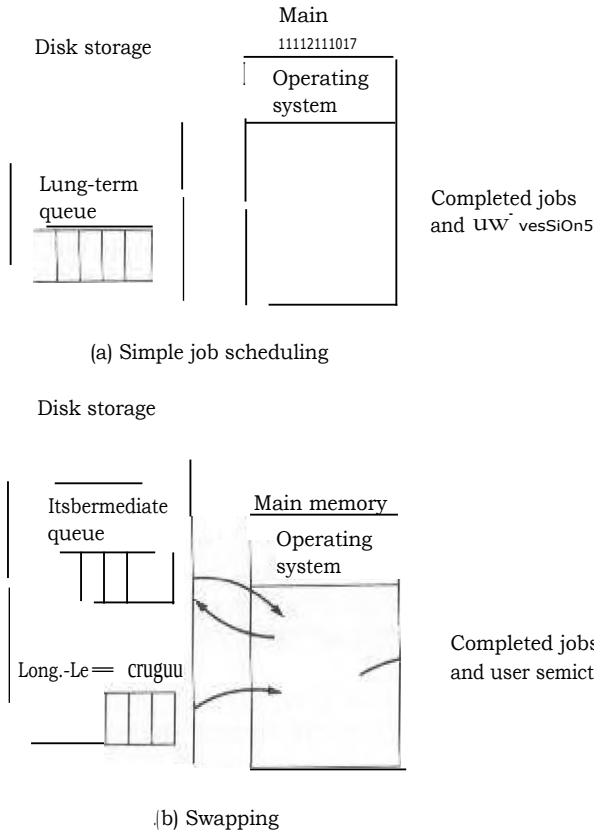


Figure 8.L2 The L:S6 of Swapping

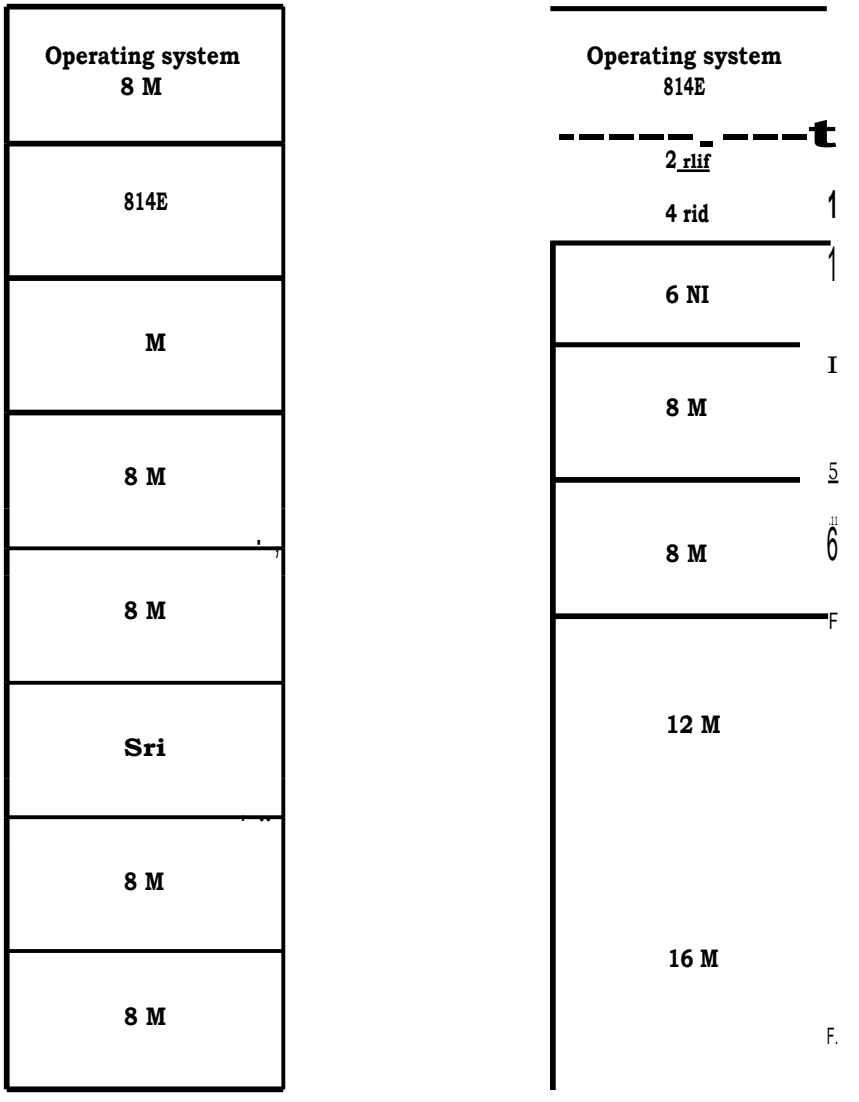
be placed in the **4M** partition of Figure 8.13b, wasting 1 M that could he used by another process,

A more efficient approach is to use *variable-,sire parritions,* When a process is brought into memory, is is allocated exactly EIS much memory as it requires and na more. An example, using 64 Mbytes of main memory, is shown in Figure 8.14.

main memory is emply% except for the operating system (a). The. first three. processes are itiaded in. startilig where the operating system ends and occupying just enough space for each process (b, c, d). This leaves a "hole" at the end of memory that is too small for a fourth process. At some point, none of the processes in him- or is ready. The operating system .iw4ips out process 2 (e), which leaves sufficient room to load a new process, process 4 (1). Because process 4 k smaller than proce 2. another small hole is created. Later, a point is reached at which none o the processes in main memory is ready, but process 2% in the Ready-Suspend state, is available. Because there is insufficient r00111 in memory for process 2, the operating system swaps process l out (g) and swaps process 2 back in (h)„ 'ks this example shows, this method starts out well, but eventually it leads to a situation in which there are a lot of small holes in memory. As time goes on. memory becomes more

and more. fragmented, and memory utilization declines: One technique for over-coining this problem is *conspaction:* From time to time. the operating system shifts the proccnlseF; in memory to place alt the lice memory together in one block. This is ti me-consuming procedure, wasteful of inocessor time.

Before we consider ways of dealing with the shortcomings of partitioning, we must clear up one loose end. If I he 17i..aLlor considers Figure 8.14 for a moment, it should become obvious that a process is not likely to he loaded into the same place in main memory each time it is swapped in. Furthermore, if compaction is used,



i A) Equal-size p irl itions

(I)) Unequai-size partitions

N. 1.3 H.xanipi.e. o1 Fiyed rtit i on i 40' a 64-Mbyte. Memory

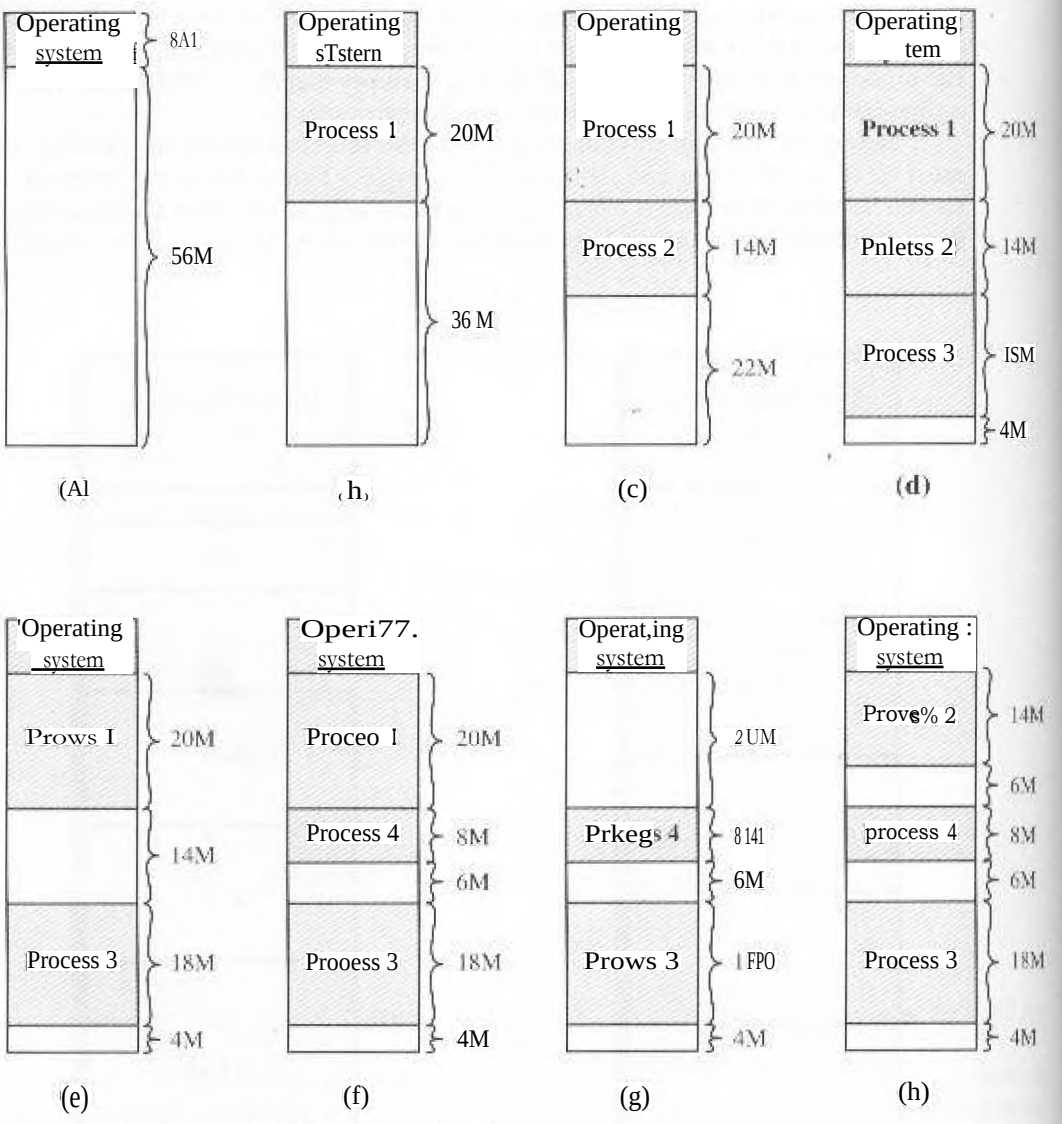


Figure 8.14 Th4.!. LITect of Dynamic Partitioning

process may be shifted while in main memory. A process in memory consists of instructions plus data. The instructions will contain addresses for memory locations of two types:

- Addresses of data items
- Addresses of instructions, used for branching instructions

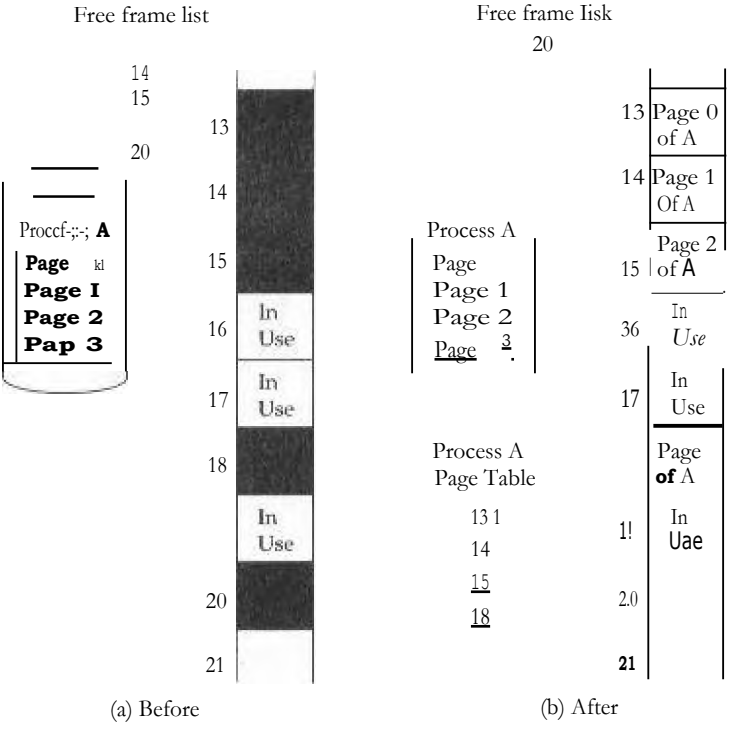
But these addresses are not fixed. They will change each time a process is swapped in. To solve this problem, a distinction is made between logical addresses

and physical addresses. A **logical address** is expressed as a location relative to the beginning of the program. Instructions in the program contain only logical addresses. A **phyNicall addrvis** is an actual iocatien in **main memory. When the processor** executes a process, it automatically converts from logical to physical address by adding the current starting location of the process, **called it base address, to** each logical addrcs.s..lhis is another exainpte of a processor hardware feature designed to meet an tepertrtiit syS.leitl requirement. The exact nature of this hardware feature depends on the memory management strategy in use. We will see several ex:In Epics later in this chapter.

Paging

Both unequal fixed-size and variable-sire purlilions are inefficient in the use of memory. Suppose, however, that memory is partitioned into equal fixed-size chunks that are relatively small, and that each process is also divided into small riNed-sive chunks of some size. Then the chunks of a program, known as *pages,* could he assigned L0 available chunks of memory. known *asframt!s,* **or** page frames. At most. then, the wasted space in rnemor,. for that process is a fraction of the last page.

Figure 8.15 shows an example of the use of pages **and** frames- At a given point in lime, some: of I ht frames in memory are in use and some are INe. The list of free frames is miiin tai ined l the operati[ts s!,.stetn, Process A, stored on disk, consists of



FiRtire R,1 Allocalion of Free Frarncs

four pages. When it comes time to load this process, t H ' **Pe rating** system finds four free frames and loads the tour pages of the process A into the four frames.

Now suppose, as in this exampie, that the,re ore nog sufficient unused corn12.u-ous frames to hold the process. Does this prevent the operating system frorn loading A'? The answer is no, because we can once again use the concept of logiol address. A simple base address will no langer suffice. **REli** her, the operating sri...:11 rnziiiitailiS a *page table* for each process. The pogo table shows the frame location for each page 61' the process. Within the program, each logicai address consists of [a] page number and a relative address within the page. Reeall that in the case of simple par. tiiioning, as logical address is the location of a word relative to the beginning u l the program; the processor translates ;hat into o physical address. With paging, the address translation is still done by processor hardware- The procc. sor must know how to .ievess the page table of the current process. Presented wi1h logical address (page number, relative address}, the processor uses the pa table ;o produce a physical address (frame number, relative address). An exampl 2 is shown in Figure 8.16,

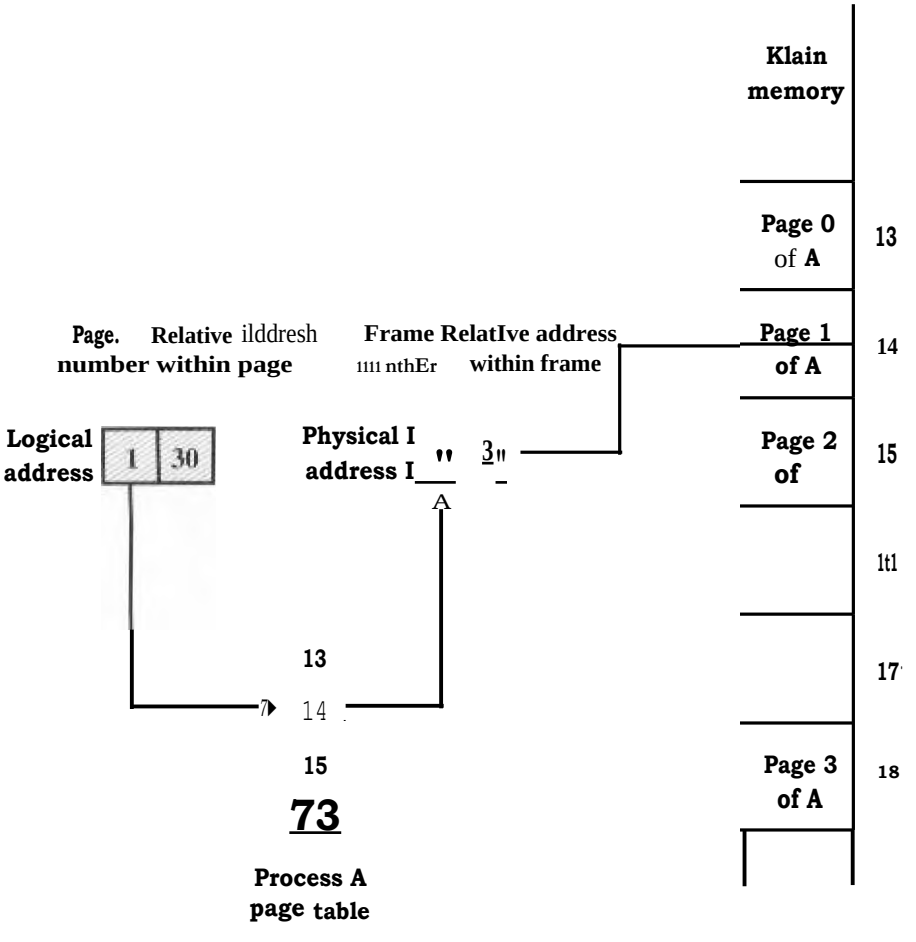


Figure 8.16 I,LOcal anti **Physical Addresses**

This approach solves the problems raised earlier. Main memory is divided into many small equal-size frames- Each process is divided into flume-size pages: Smaller processes require fewer pages, larger processes require more. When a process is brought in, its pages are loaded into available frames, and a page table is set up.

Virtual Memory

Demand Paging

With the use of paging, truly effective multiprogramming systems came into being. Furthermore, the simple tactic of breaking a process up into pages led to the development of another important concept; viii Wi I mil Wiry.

To understand virtual mcniory, we must add a refinement to the paging scheme just discussed. That refinement is *demand pix ing.* which simply means that each page of a process is brought in only when it is needed, that is, on demand.

Consider a large *process,* vork sisLing of a lone; program plus a number of arrays of data. Over any short period of time, execution may be confined to a small section of the program (e.g., a subroutine), and perhaps only one or 1 WO arrays of data are being used. This is I he principle cif Ideality, which we introduced in Appendix 4A. /1 would clearly be wasteful to load in dozens of pages for that process when only a few pages will be used before the program is suspended, We can make better use of memory by loading in just a few pages, '['hen, if the program branches to an instruction on a page not in main memory, or if the program references data on a page not in memory, a *page fault* is triggered. This tells the operating system to bring in the desired page.

Thus, at any one time, only a few pages of any given process are in memory, and therefore more processes can be maintained in memory. Furthermore, time is saved because unused pages are not swapped in and out of memory. I lowever, the operating system MUM be clever aboni how u rnanmus this scheme. When it brings one pagc in, it must throw another page out. If it throws out a page lust before i1 is about to be used, then it will just have to go get that page again a Imost immediately. Too much of this leads to a condition known aiti *thrashing:* The processor spends mosr 0r its time swapping pages rather than executing instructions. The avoidance of thrashing was a major research area in the 1970s and led to a variety 0r complex but effective algorithms. In essence, the operating system tries Lu guess, based on recent history', which pagc.s are least likely to be used in the near future.

With demand paging, it is not necessary to load an entire process into main memory. This fact has a remarkable consequence: lir Es *pos.vihie fw* IF *prmess* try *be larger than all of main memory.* One o1 the most fundamental restrictions in programming has been lifted. Without demand pagin g. a programmer must be acutely aware of how much memory is available. If the program being wri ll en is too large, the programmer must devise ways lo structure t he program into pieces that can be loaded *onc* at a time. With demand paging, that job is left to the operating system and the hardware. As far as the programmer is concerned, he or she is dealing with a huge memory, the size associated with disk storage_

Because a pawXSN L.NE:„Culi,!;; only in miin memory, that memory is referred to as real memory. But a programmer or user perceives a much larger memory—I hat v.rhich is allocated on the disk. This latter is therefore referred to as virtual memory.

Virtual memory allows for very effective multiprogramming *and* relieves the user of the unnecessarily tight constraints of main memory.

Page Table Structure

The basic mechanism for reading a word from memory involves the translation of a virtual, or logical, address. consisting of page number and offset, into a physical address, consisting of frame number and of fs;et, using a page table.. Because the page table is of variable length, depending on the size of the process, we cannot expect to hold it in registers. Instead, it must be in main memory to be accessed. Figure 8.16 suggests a hardware implementation of this scheme. When a particular process is running, a register holds the starting address of the page table for that process. The page number of a virtual address is used to index that table and look up the corresponding frame number. This is combined with the offset portion of the virtual address to produce the desired real address.

In most systems, there is one page table per process. But each process can occupy huge amounts of virtual memory. For example, in the VAX architecture. each process can have up to 2^{31} = 2 GBytes of virtual memory. Using 2^{9} = 512-byte pages, that means that as many as page table entries are required *per* VOL Clearly, the amount of memory devoted to page tables alone could be unacceptably high. To overcome this problem. most virtual memory schemes store page tables in virtual memory rather than *real* memory This means that page tables are subject to paging just as other pages are. When a process is running, at least a part of its page table must *be* in main memory, including the page table entry of the currently executing page. Some processors make use of a two-level scheme to organize large page tables. **In** this scheme, there is a page directory. in which each entry points to a page table. Thus, if the length of the page directory is *X,* and if the maximum length of a page table is Y, then a process can consist of up to *X* X Y pages. Typi cally, the maximum length of a page table is restricted to be equal to one page. We will see an example of this two-level approach when we consider the Pentium El later in this chapter.

An alternative approach to the use of one- or two-level page tables is Ihe usc of an inverted page table structure (Figure 8.17). This approach is used on IBM's AS141K) and on all of IBM's RISC products, including the PowerPC.

In this approach. the page number portion of a virtual address is mapped into a hash table using a simple hashing function. [a] The hash table contains a pointer to the inverted page table, which contains the page table entries. With this structure. there is one entry in the hash table and inverted page table for each real memory page rather than one per virtual page. Thus. a fixed proportion of real memory is required for the tables regardless of the number of processes or virtual pages sup-

---

[a] A hash ['unction maps numbers in 1112 range II through *M* into number:: in the 74ingc through A', who 6 > .V. The output of the hash function is used as an index into ihc hash table. Since more thar., input maps to the same output, it is passible for art iltpul item In map to a hash in ble entry that is ahvadv: occupied. In that case, the new item must *merthrw* ink})In ocher hash table location, 1 yriCtilly. the new item is placed in the firsl succeeding empty space. .and a pointer from the original location is provided to chain the entries together. Sec [STALL)! J for a more tleiailed discussion Or hash tables.

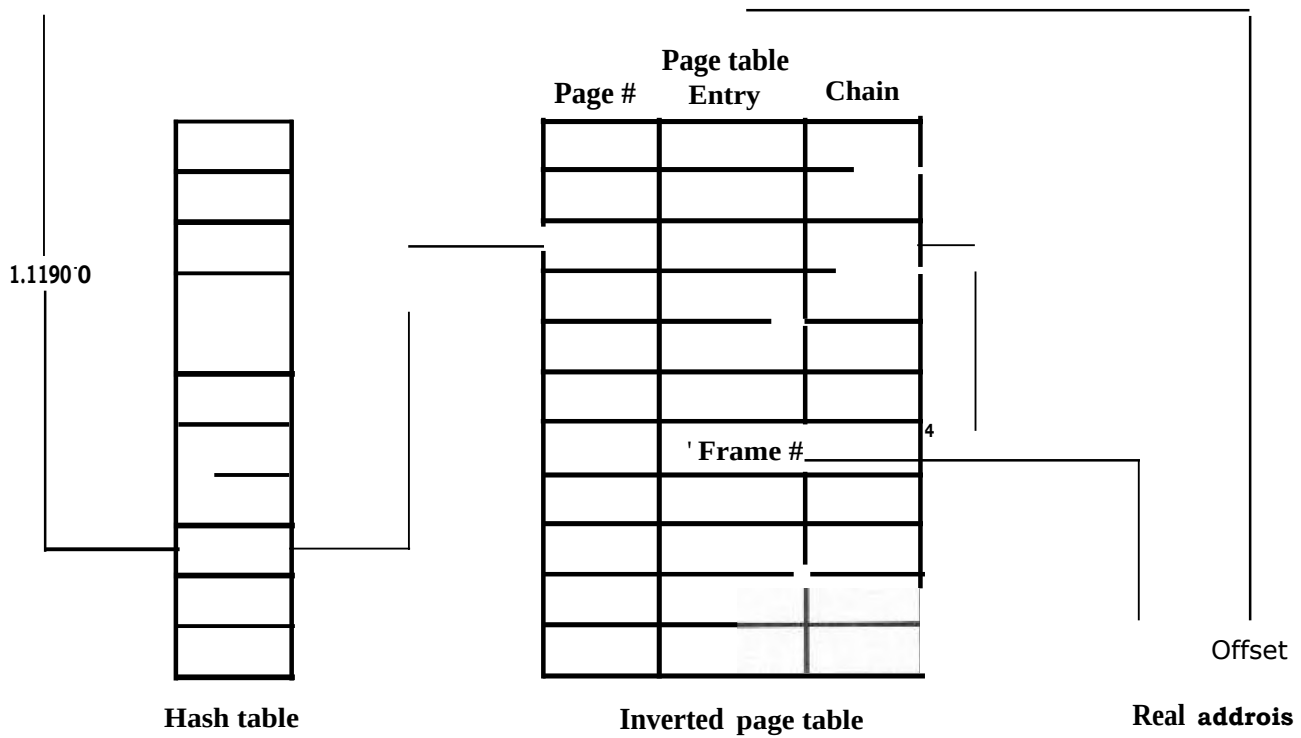


Figure 8.17 Inverted Page Table Structure

ported. Because more than one virtual address may map into the same hash table entry, a chaining technique is used for managing the overflow. The hashing. tech• nique results in chains that are typically short—either one or two entries,

Translation Lookaside Buffer

In principle, then, every virtual memory reference can cause two physical memory accesses one to fetch the appropriate page table entry, and one to fetch the cicwinJ data, Thus, a straightforward virtual memory scheme would have the effect of doubling the memory access time_ To overcome this problem, most virtual memory schemes make use of a special cache for page table entries. usually called a transla• lion lookaside buffer (TLB). This cache functions in the same way as a memory cache and contains those page table entries that have been most recently used. Figure 8.18 is a flowchart that shows the use of the TLB. By the principle of locality. most virtual memory references will be to locations in recently used pages. Therefore. most references will involve page •table entries in the cache. Studies of the VAX 11.13 have shown that this scheme can significantly improve perform= ICLAR85, SATYSI].

Note that the virtual memory mechanism must interact with the cache system (not the 'ILE; cache, but the main memory cache). 'Phis is illustrated in Figure 819. A virtual address will generally be in the form of a page number, offset. First, Ihe memory system consults the TLB to see if the matching page table entry is present, If it is. the real (physical) address is generated by combining the frame number with the offset. If not, the entry is accessed from a page table. Once the real address is generated, which is in the form of a tag and a remainder (see Figure 4.17), the cache is consulted to see if the block containing that word is present. If so, it is returned to the processor, If not, the word is retrieved from main memory.

The reader should be able to appreciate the complexity of the processor hardware involved in a single memory reference. The virtual address is translated into a real address. This involves reference to a page table. which may be in the TLB, in main memory, or on disk. The referenced word may be in cache, in main memory, or on disk. In the latter case, the page containing the word must be loaded into main memory and its Wick loaded into the cache. In addition, the page table entry for that page must be upda I ed.

Segmentation

There is another way in which addressable memory can be subdivided, known as *segmernathm.* Whereas paging is invisible to the programmer and serves the purpose of providing the programmer with a larger address space, segmentation is usually visible to the programmer and is provided as a convenience for organizing programs and data, and as a means for associating privilege and protection attributes with instructions and data.

Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments. Segments are of variable, indeed dynamic, size_ Typically, the programmer or the operating system will assign programs and data to clifferent segments, There may he a number of program segments for various types

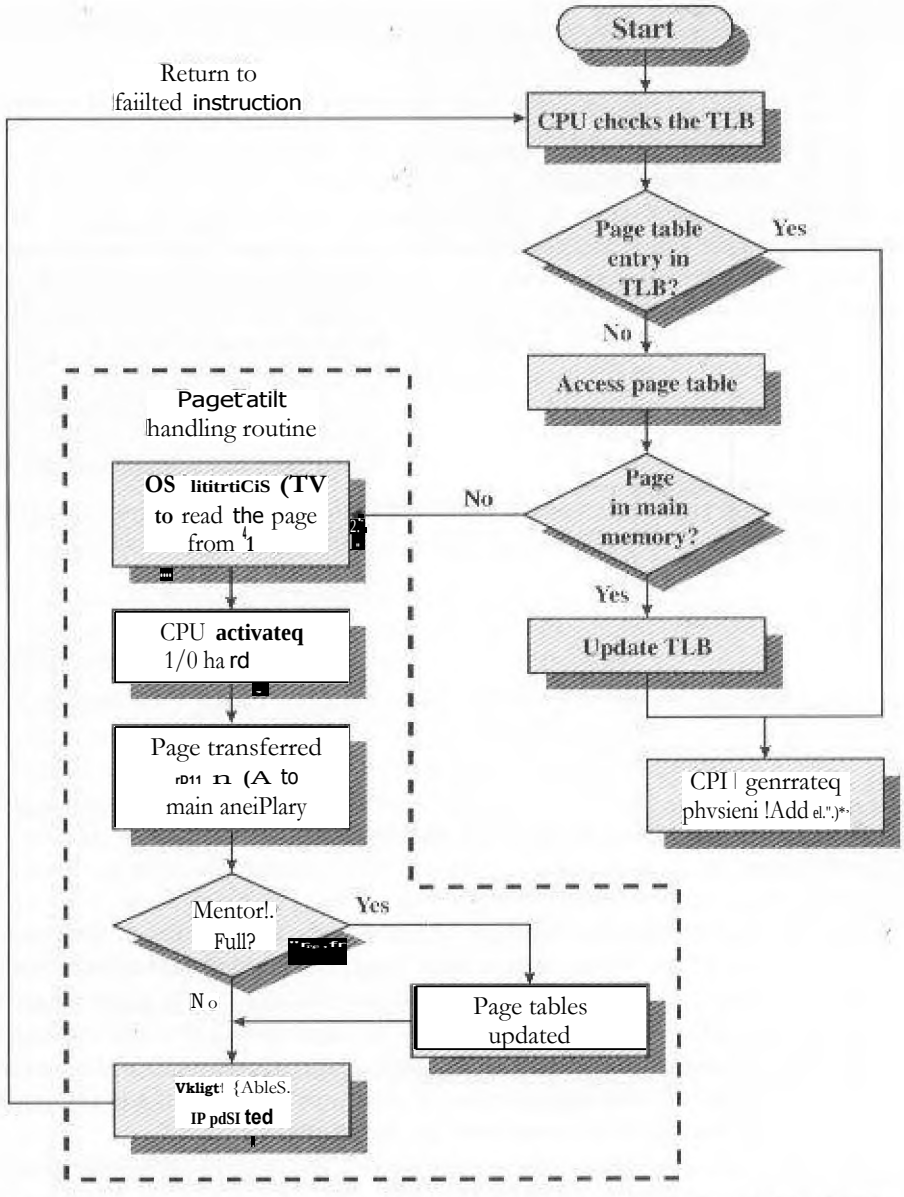


Figure 8.18 Operating of Pagi and R.. ion Lookaside Bufk.r (TLB) FUR1 1871

IA programs as ws n umber of data gegments. Each segment may be. ;.isNigned ,aastn d rights. Memory referencos consist of (segment umber, offset) form of address.

'Ellis organization has a riumbu Of ; Kiv;tii Lages to the proarammer over a non-segmented adfircs:i space:

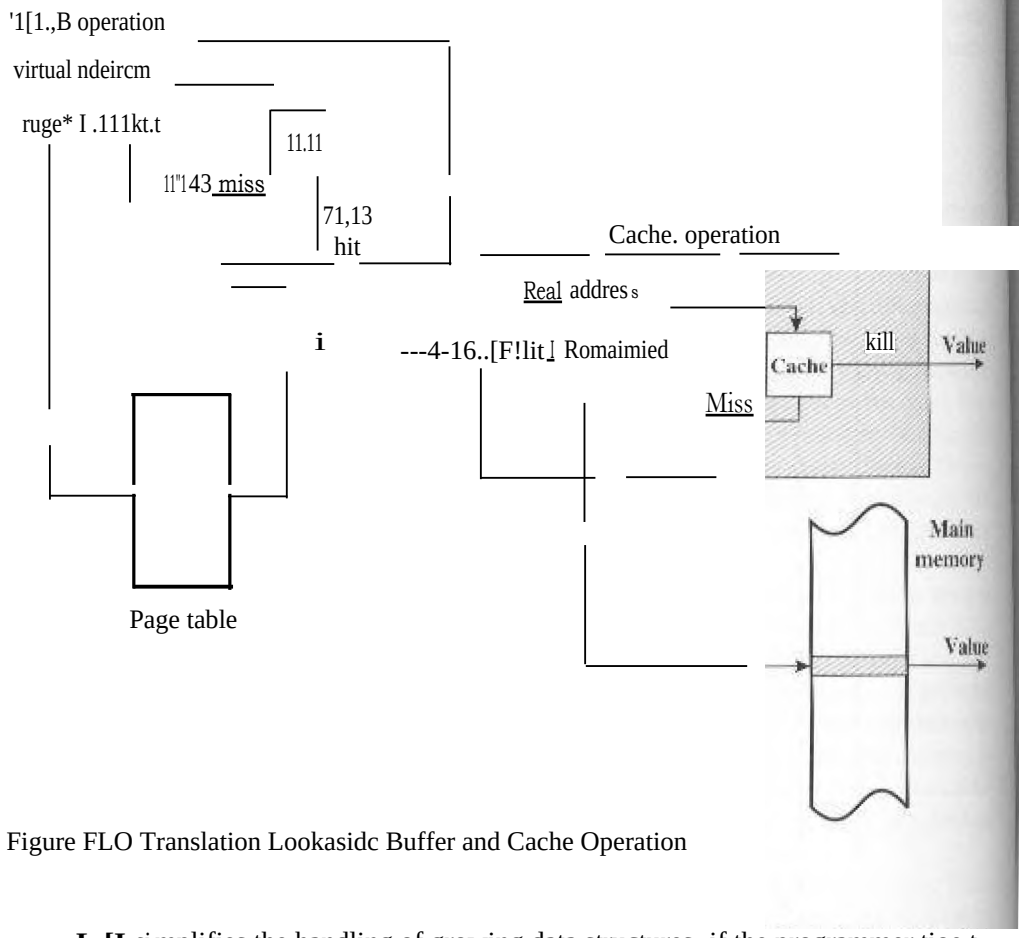


Figure FLO Translation Lookasidc Buffer and Cache Operation

L [I si mplifies the handling of growing data structures- if the programmer tioe.t. not know ahead of time how large a particular data structure will becOrnc,it, is not necesar!,' to guess- The. data structure can he assigned its own scgalept,:, and the operating system will expand or shrink the segment as needed.

2.. It allows programs to he altered and recompiled independently. without requiring that an entire !.51 of program* he relinked and reloaded. Again, this is accomplished using multiple segments.

3, It lends itself to sharing among processes. A programmer can place a utilit:.• program or a uscful table of data in a segtnent that can be addressed he other processes.

4. II leads itself I o protection. Reeause a segment can be constructed to contain a well-defined set of programs or data, the programmer or a system administrator can assign ac(xss privileges. in a convcnient fashion.

These advantages are not available with paging, which is imjiNible to Ilse programmer, On the other hand, we have seen that paging provides for an efficient form of memory management. combine the advantages of both, some systernh are equipped with the hardware and operating system software to provi& both.

8.4 PENTIUM II AND POWERPC MEMORY MANAGEMENT

Pentium II Memory Management Hardware

Since the introduction of the 32-bit • architecture, microprocessors have evolved sophisticated memory management schemes that build on the lessons learned with medium- and large-scale systems. In many cases, the microprocessor versions are superior to their larger-system antecedents. Because the schemes were developed by the microprocessor hardware vendor and may he employed with a variety of operating systems, they tend to he quite general purpose. A representative example is the scheme used on the Pentium 11. The Pentium 11 memory-management hardware is essentially the same as that used in the Intel 80386 and NO486 processors, with some refinements.

Address Spaces

The Pentium II includes hardware for both segmentation and paging Both mechanisms can be disabled, allowing the user to choose **from** four distinct views of memory:

- **Unsegmented unpaged memory:** In this ease, the virtual address is the same as the physical address. This is useful, for example, in low-complexity, high-performance controller applications.
- **Unsegmented paged memory:** Here memory is viewed as a paged linear address space. Protection and management of memory is done via paging. Thi, is favored by some operating systems (e.g., Berkeley. UNIX).
- **Segmented unpaged memory: Here memory** is viewed as a **collection of logical** address spaces. The advantage of **this** view over a **paged approach is that** it affords protection down to the level of a single byte. if necessary. Furthermore, unlike paging, it guarantees that the translation table needed (the segment table) is on-chip when the segment is in memory. Hence, segmented unpaged memory results in predictable access **times.**
- **Segmented paged memory:** Segmentation is used lo define logical memory partitiOns subject to access control, and paging is used to manage the allocation of memory within the partitions. Operating systems such as UNIX Systern V favor this view.

Segmentation

When segmentation is used, each virtual address (called a logical address in the Pentium II documentation) consists of a 6-bit segment reference and a 32-bit offset. Two hits of the segment reference deal with **the protection** mechanism, leaving 14 bits for specifying a particular segment. Thus, with unsegmented memory, the user's virtual memory is 2^{32} = 4 GBytes. With segmented memory, the total virtual memory Taco as seen by a user is 2" = 64 terabytes (TBytes). The physical address space employs a 32-bit address for a maximum of 4 Bytes.

The amount of virtual memory can actually be larger than the 6'4 ·Myles. This is because. the processor's interpretation of a virtual address depends on which process is currently active. Virtual address space is divided into two parts. One-half

of the virtual address space (8K segments X 4 CiBytes) is global. shared by all processes; the remainder is local and is distinct for each process.

Associated with each segment are two forms of protection: privilege level and access attribute. There are four privilege levels from most protected (level 0) to least protected (level 3), The privilege level associated with a data segment is its "classification"; the privilege level associated with a program segment is its "clearance." An executing program may only access data segments for which its clearance level is lower than (more privileged) or equal to (same privilege) the privilege level of the data **segment.**

The hardware does not dictate how these privilege levels are to he used; this depends on the operating **system** design and implementation, II was intended that privilege level I would be used for most of the operating system, and level (I would he used for that small portion of the operating system devoted to memory management, protection, and access control. This leaves two levels for applications. In many systems, applications will reside at level 3, with level 2 being unused. Specialized application subsystems that must be protected because they implement their own security mechanisms are good candidates for level 2. Some examples are data• base management systems, office automation systems, and software engineering environments.

In addition to regulating access to data segments, the privilege mechanism limits the use of certain instructions. Some instructions_ such as those dealing with memory-management registers, can only be executed in level O. 1/C) instructions can only be executed up to a certain level that is designated by the operating system: typically. this will be level 1.

The access attribute of a data segment specifies whether read—write or read-only accesses are permitted. For program segments. the access attribute specifies readlexecute or read-only access.

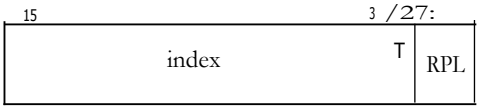
The address translation mechanism for segmentation involves mapping a virtual address into what is referred to as a linear address (Figure 8.20b). A virtual address consists of the 32-hit offset and a 16-hit segment selector (Figure 8,20a). The segment selector consists of the following fields:

- **Table Indicator (Ti): Indicates** whether the global segment table **or** a local segment table should be used for translation.
- **Segment Number;** The number of the segment. This serves as an index into the segment table.
- **Requested Privilege Level (RPL):** The privilege level requested for this access.

Each entry in a segment table consists of t54 bits, as shown in Figure 8.20c. The fields are defined in Table 8.5.

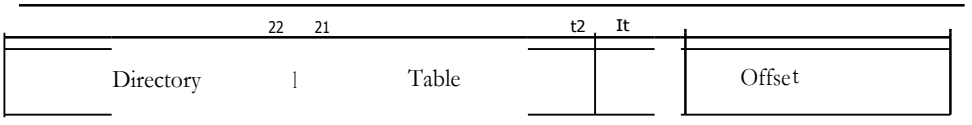
Paging

Segmentation is an optional feature and may be disabled. When segmentation is in use, addresses used in programs are virtual addresses and are converted into linear addresses, as just described. When segmentation is not in use, linear addresses are used in programs. In either case, the following step is to convert that linear address into a real 32-bit address.

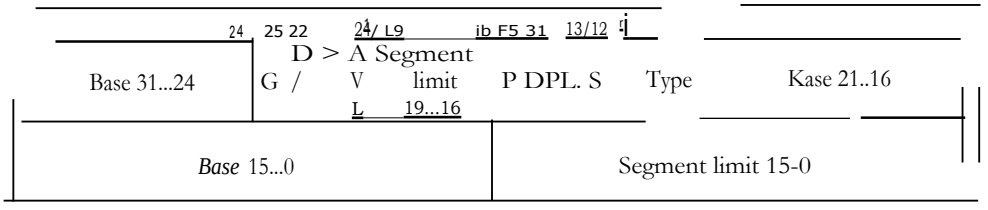


Il = Table indicator
RN. Recluestor Twiviiege level

(a) Segment selector



(b) Linear address

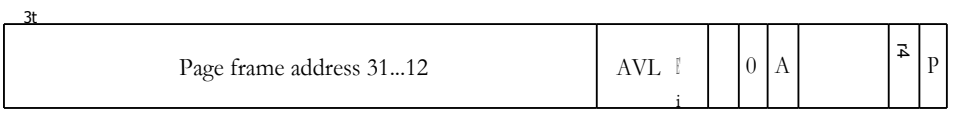


AVL Available for use by system software
Base Segment he address
D/B **De**f a ul t operation si2A
= Descriptor privilege size

G Granularity
Limit = Segment limit
P = Segment present
Type = Segment type
Descriptor type

Reserved

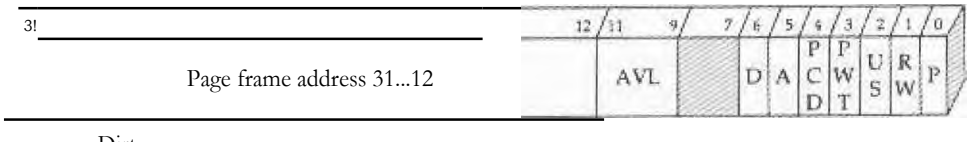
(e) Segment descriptor (segment table entry)



AVL = Available for systems programmer use
PS = Page *size*
A = Accessed
PCD = Cache disable

PWT Write through
US = User /supervisor
RW a Read—write
P = Present

(c1) Page directory entry



Dirty

(e) Page table entry

Figure 8.20 Pentium Memory-Management Formals

IahK 4A PL;ntiurtt TI Mcrnur• Mii gciir11 Pirarn1cr

Se t **De**st (S **mnt thk** Iitti).

I)e#roes the **LTI1l** idt sw i)1 t hL rci men witI] iu the 4-(1 (e huier Lddi i **t pa** rim ail

In :I)Lk **•efl nI,** tEii **j 'Jic U** bil art[] incFt tc. **11LE1e** 1,t **rijith** ac : dd **sirlg** nmI• a**t:** l **b** ut 2.

1}CSC[ig#nr **P** . ik **iI fl 11L)** 2',4 E%4

Spi i1iLx L FIL prvit c te J of Eh.seiner retCrie3to 1 thi **nioht dc.** r:

Cir^mlliprits bi

]raJicatt '}ih'i the Liti1 iicltt tctbant t diiiw3i h iii bti 4. Iyts.

L1Ih

L7(1fli Llsc ca**rp i rh inrI** The i **r** inEcrpretth [Lit fl ciLtic rtf **jI)**

the pymiithritv liii: in LLr11tS Oi ciiic **bl tc, up** Ri 11 I1E nkre tiri U L **MBi.** ι **r** in ki nil lit 4 K[ytc, itp Le0 i r C.L!1Th.L1 si 4 3 **iu1i4c'** t CT]vtci

hit

L)ctriiiittt \ h her U 5cgflWI1L **i.s a RyctcIll i tUeflt** 1 d **CL^4e** nr LIaF;L gLVLL.1I.

Segrneai **I'vc ni** bit (P}

] d fur ii JIpa2,c1 1 **jdica** bEIitr th• - l in iiiülti **mc** 1 uy. **Rr** itged **1tt]tI,. [hLs iHI** i yI l **Iii** I.

Tpe

LJi **guih** btwcLJJ arKItIc hi I] LI l **gmrits id** incl]c.ucr. the ccss Muihutc..

Pe l)trectory F.ntr hge Tthk **Entry**

c ii Mt {A

]Thk h[L i5 %**a L't h ih** r1,Lt **ur** in **h**a **t}i** Ie t is **4f** p; e **Lab.9** whi n ii.ad eFi **wrft.o opr:n'Ii t.'** th

• r iijULni page ocuii

1)irty blI **(1))**

This bits, i4 t ki t] **y Ili pr CY**3 **.ui]'** Ji ii w **rite pr:IIirI w t**ll c tii r pornJir, Iaal L O(GUr.

Fniuw A doTv.

[T{v id h phya. **ieii** I add **res3** (I **page iD** tl F1L1TY ii Ibe pfvECTI! hit k Sicc I ri (i

• nil 4K ti **iwdir.** tEIL I **i** 12 TiV **üi** 1Fd bits LSF bick **kdii.** Ur ElLr.

lit **a past** Jii tui, th 13 that)f.L]ca**ge 1thc.**

l? Cic1w **IW.ahlc bit {i(I1}**

IFLic11cs wIitIIic dmi **IrL)LEf rage m**a**y** Ei **cat1Qt;**

Pais SIebv (I'S)

Itsdicatcs whEllLf pa ge' size is 4 K1vLc or 4 Al yt;:. '

P1 1r WEIk• I'h uØi bit (I'W

ILi at!i whc **lhar arrit**e l **j J1** (i E{ Eia [+i,licw **wilt Fi** used fur daL W c] p4)E1dIig

rc*nt hit (P)

IidiCfflL whether 411c pig, ti 1e r

RiHI-WI11e **kilt (RW**

For t r-icc pa.'s, in3 wI]cihcr Iht kluge is cid-ciily acccsr t >j.d -wril int ut-[vçi

NGrT,

I r optxIF hit (U

InclLc **whether tw** vil1ik c **iiit FU** th c11urILr y4-**I**4 r **pi.riO** h V.1.F **UT** is **to**

1 *h **nil^c ILLIIri** S. Tll ;iid ILV)f **L'VLI}.**

To understand the structure of the linear address, you need to know that the Pentium II paging mechanism is actually a 1 'N4P-level table lookup operation. The first level is a page directory, which contains up to 1024 entries. This splits the 4-(i13!,'te linear memory space into 1024 page groups, each with its own page table. and each 4 MBytes in length, Each page table contains up to 1024 entries; each entry corresponds to a single 4-kByteyage. Memory management has the option of using one page directory for all processes, one page directory for each process, or some combination of the two. The page directory for the current task is always in main memory. Page tables may be in virtual memory.

Figure 8.20 shows the formats of entries in page directories and page tables, and the fields arc defined in Table 8.5. Note that access control mechanisms can be provided on a page or page group basis.

The Pentium H also makes use of a translation lookaside buffer. The buffer can hold 32 page table entries. Each time that the page directory is changed. the buffer is cleared.

Figure 8.21. illustrates the combination of segmentation and paging mechanisms. For clarity, the translation lookaside buffer and memory cache mechanisms are not shown.

Finally. the Pentium 11 includes a new extension not found on the 80386 or 80486. the provision for two page sizes. If the PSE (page size extension) bit in control register 4 is set 10 1, then the paging unit permits the operating system programmer to define a page as either 4 kByte or 4 MByte in size.

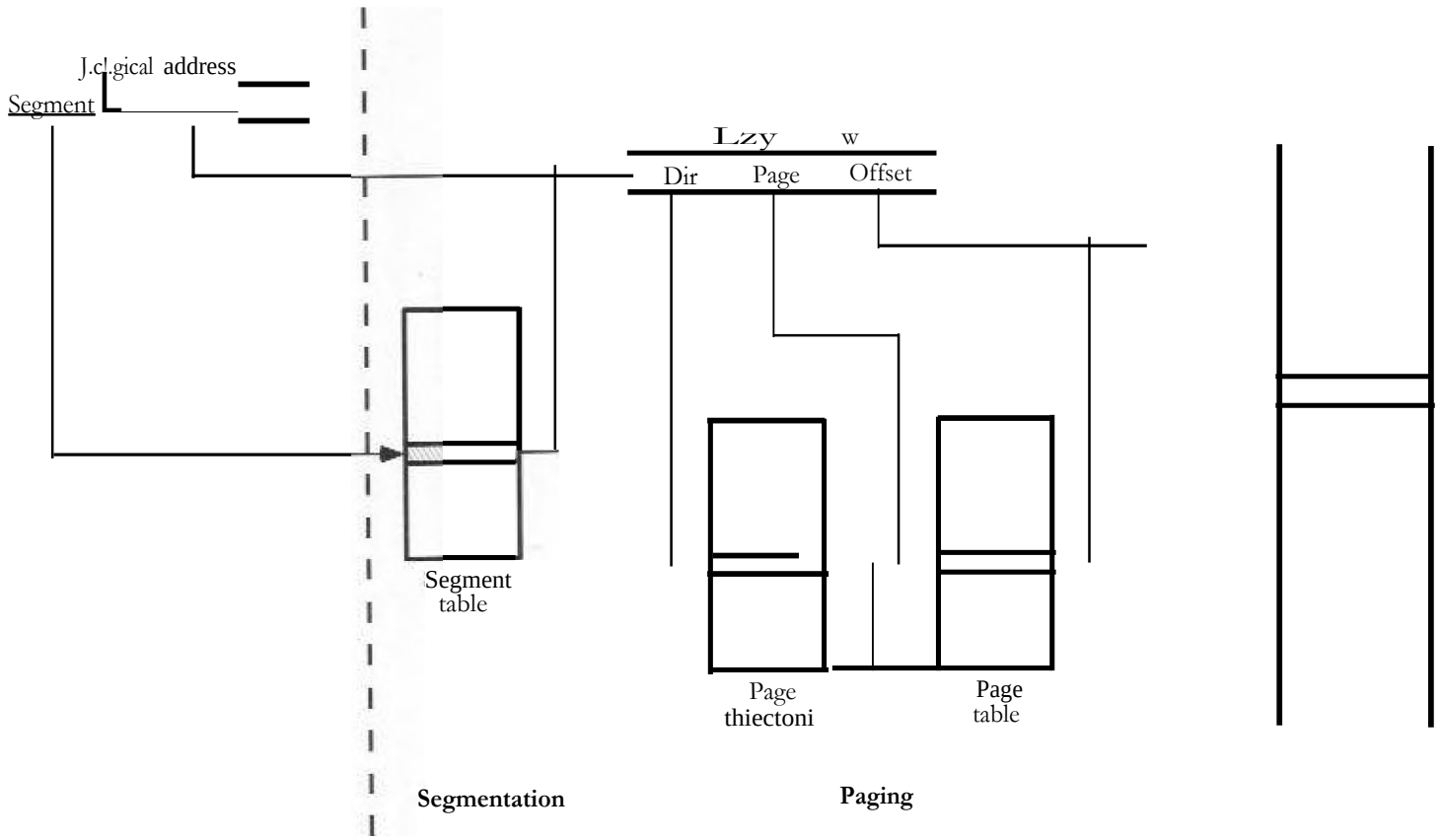
When 4-MByte pages are used, there is only one level of table lookup for pages. When the hardware accesses the page directory, the page directory entry (Figure-8,200 has the PS bit set to 1. In this case, bits 9 through 21 are ignored and hits 22 through 31 define the base address for a 4-MByte page in memory. Thus, there is a single page table.

The use of 4-MByte pages reduces the memory-management storage requirements for large main memories. With 4-KByte pages, a full 4-GByte main memory requires about 4 MBytes of memory just for the page tables. With 4-M Byte pages, a single table, 4 Bytes in length, is sufficient for page memory management.

PowerPC Memory-Management Hardware

The. PowerPC provides a comprehensive set of .addressing mechanisms. For 32-bit implementations of the architecture, a paging scheme with a simple segmentation mechanism is implemented. For 64-bit implementations, paging and *a* more powerful segmentation mechanism are supported. In addition, for both 32-bit and 64-hit processors there is an alternative hardware mechanism, known as block address translation. Briefly, the block addressing scheme is designed to address one drawback of paging mechanisms. With paging, a large number of pages may be frequently referenced by a program. For example, programs that use OS tables or graphics frame buffers may exhibit this behavior. The result may he that frequently used pages are constantly paged in and out. Block addressing enables the processor to map lour large blocks of instruction memory and four large blocks of data me.mory in a way that bypasses the paging mechanism.

A discussion of block addressing is beyond the scope of this chapter. In this subsection_ we concentrate on the paging and segmentation mechanisms of the 32-bit PowerPC. The 64-bit scheme is similar,



'Fngate M.21 Pentium Niemen, AdIdlre:.5s Translation Mccharn MT'S

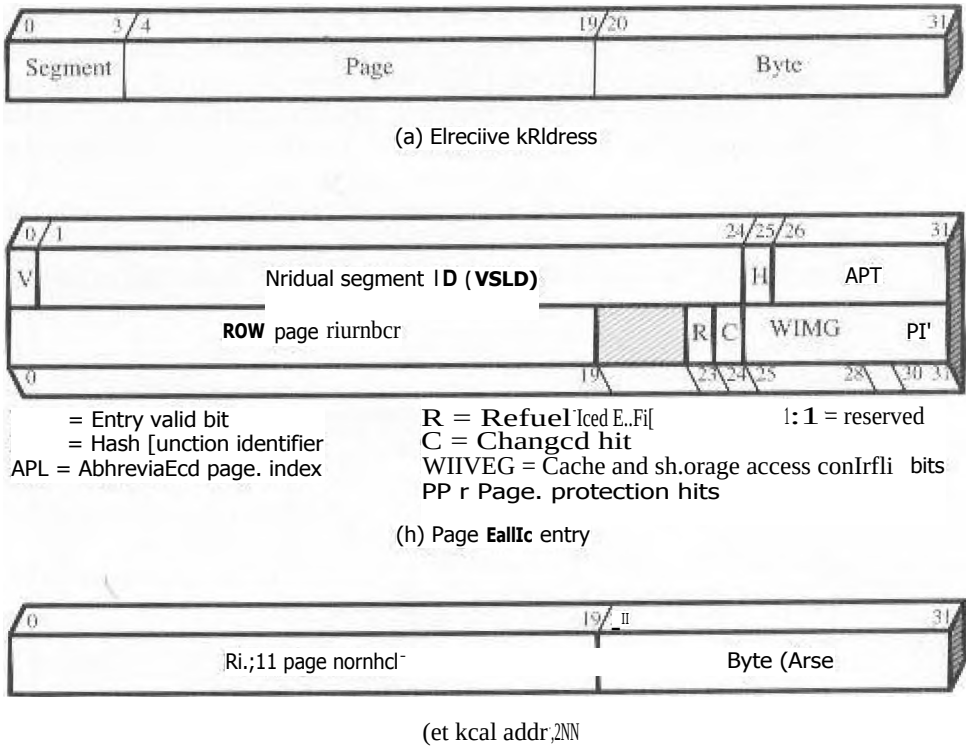


Figure 8.22 liovr.cr PC 32-Bit Nle.niory-Ivlarragerns?ht Formats

The 32-bit PowerP(' makes use of a 32-bii effective address (Figure. 8.22a), The address includes a 12-bit byte selector and a 16-bit page identifier. Thus, 2^{12} = 4 KByte pages are used. Up to 2^{16} = 64K paes per segment arc allowed. Four bits of the address are used to designate one of i 6 seRnient registers. The contents of these registers are controlled by the operating sysi ern. Each segment register includes access control hits and a 24-bit identifier, so that the 32-bit affective address maps into a f32-bit virtual Atiress (Figure

The PowerPC makes use of a single inverted page table. The virtual address is used to index into the page table in the following manner. First, a hash eode. is computed as racy's:

1-t(U , _18) SiD(5 23) e vpN(t)

The virtual page number in the virtual address is padded on the is [`t (most significant end) with three binary zeros to form a 19-bit number. Then a bii-by-bi I exclusive-or is calculated of that number and the 1.9 right-most bits of the virtual segment IL) to form 19-bit hash code. The 'able is organized as rt groups of 8 entries, From 10 to 19 hits of the hash code (depending on the size of the page table) arc used to select one of the groups in the table. The memory-management hardware then scans the eight entries of the group to test for a match with the **virtual address.**

.r() do the match, each page table entry includes the virl dal segment TI) and the left-most 6 bits of the. virtual page numbcr, called the abbreviated page index

(because at least 10 hits of the 6-bit virtual page number 411 w HyS participate in the hash to select a page iablc.t entry group, only an abbreviated form of the virtual pup number need be carried in the page table entry to match the virtual address). If here is a match, then the 20-bit real pge number from the addres.5 is concatenated with the lower 12 hits of the **drective** address to form Ihe 32-bit physical address to be accessed.

If there is no match, then the hash codex complemented to produce a new page table index that is in the some relative position at the opposite end of the table. This group is Ihcn scanned for a match. if no match is found, a page fault interrupt occurs.

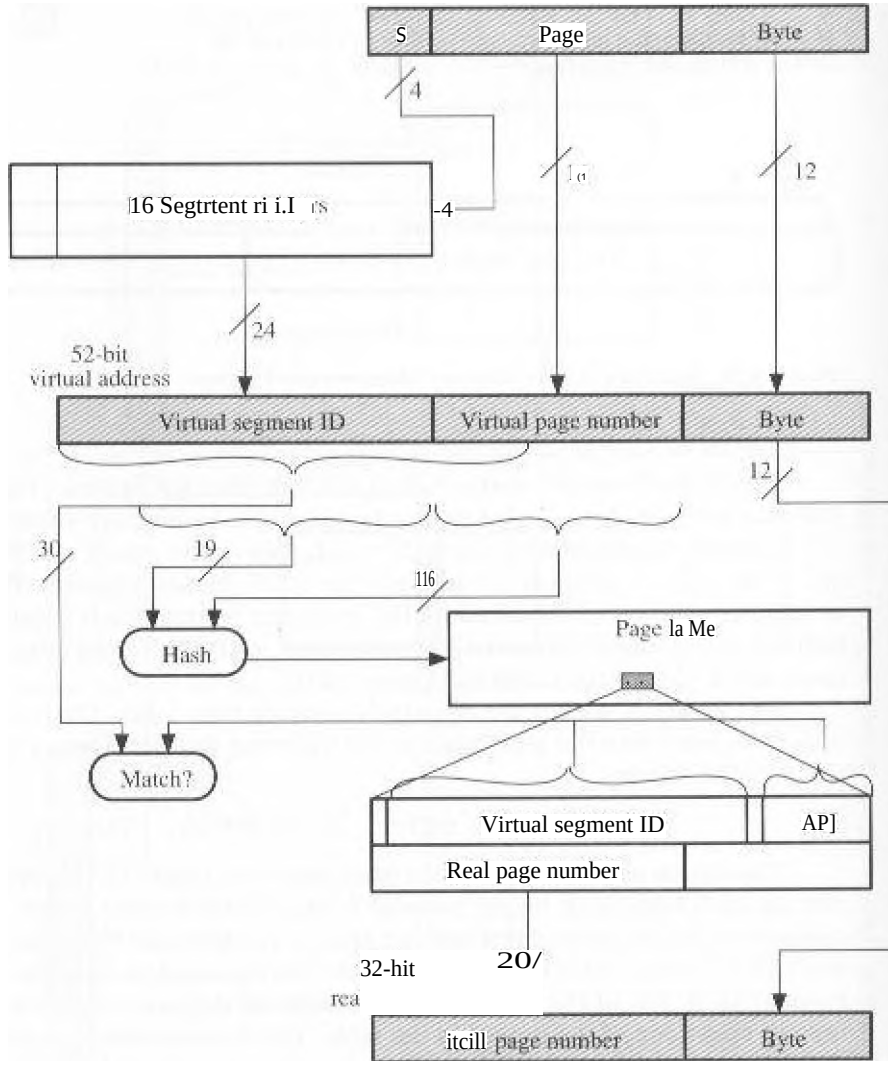


Figure 8.23 Pows:rPC 32-Lilt AtitIr Translation

Table 84 1.[3]43.tvel PC tyleinory Management Paranimors

Sexrnetit Table Entry	
Effeeki.te **Segment**	
I di:ases or, of E14.1+ 4tftctivu	useti iu J teriLlul ealty
Entry Valid (V) hit	
IndicasC5 wheshff this i,	111C,1-11(ity or E.:Orscirjr il.
Negnient Type (I) lest	
Indicaies whoslier	iN #i LswinUrs.. or E!) silginelit.
SapervIsor Key Ms)	
li;sed with tlio vatuail [raga n Limbo' to	tenni Ile trues ill page
Page Table Entry'	
Vrirry Valid. (1.[1]) Iiii	
1.1.1n2i her ih{214:' Is walivi data iik this	
1-40 Tikniiiier	
fridicHres w IIE111C1 LIVIA LS a]IriiIsa7y	
edge IT-Hfrx (API I	
Usxkl. to match 11 VIII U.,I iidd 1C.SS 1.11.16.1L1.21Y.	
Referenced (R) bit	
'Phis hit is we I Ii} I by 1.1112 procossrm when	read Or opErviLicin Lis Lhe.cisrvi..1)rsi:Lliir: Iiagi ticcurS.
('hinged **(C) bit**	
This hit is rL L to I by the prau2ssor when 3 write opratiors is i	corrusp.sriding (KC
I M4. spitz	
4G=-si: Usti Iv rice-back	use write-Enroll h
1.1.1: uiching next Ii hibiLed; [::I; cachininhibitcal	
M - not shared **shard memory.**	
O .0: rial guald.r.:4.1 memory; 6=1: gnarlwd	
P ›#i Prolottion. **(PP) bits**	
AIXIL'NS Ce.slI4t01 hft ILIStN.1 With K Nis Flom ss2gimui	y.L.F.3111311t .britrs.' Ir! define access rights.

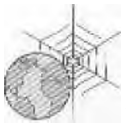
fip.ru K.22 shows the !oaic of the addresti Iranslation mechanism. and Figure 8.23 shows the formats of the effective address, page table entry, and real address. Finally. Table S,16 dctine ih pm-an-lea:Is in the page table entry.

The Memoty management scheme is designed k he dp rardly compatible vvith the 32-bit implementation. In eStiC Dec, all eficinive. addresses, general registers, and branch address registers.; are. **extended on the left to 64** bits.

8.5 RECOMMENDED READING !LIND WEB SITES

1st Auld covers Lhc. topics <rt this draw **ex in** cktail..

Sta..IIhtp, W. *Oppratin8 Systems, InEerntris* gad *PriLicipks,* 4th edition, Upper **Saddle River, NJ; Prentict. Half. 2Kl.**



Recommended Web sites:

- **Operating System Project Information:** Links to OS projects and research
- **ACM Special 6nerest** Group on Operating Systems: Information on **SIGOPS** publica. dons and conferences
- **IF:F.E Technical Committee on Operating Systems and Applications:** Includes an online newsletter and links to other sites
- **Review of Operating Systems:** Comprehensive review of commercial, free. research, and hi Fnt'ty Otis

8.6 KFY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

batch system.	oteinory protection	process state
demand paging	multiprograimning	real memory
interactive operating system	multitasking	resident monitor
interrupt	nucleus	segmentation
job control language	operating system (OS)	short-term scheduling
kernel	paging	swapping
logical address	page table	thrashing
long-term scheduling	partitioning	time-sharing; system
medium-te fin scheduling	physical address	translation lookaside buffer (T LB)
memory management	privileged instruction	**utility**
	priicess	v **irtual** memory
	process control block	

Review Questions

8.1 What is an operating system?
l.ist and briefly define the key services provided by an operating system.

8.3 List and brielly define the major types of **OS** scheduling.

8.4 What is the difference between a process and a program?

8,5 What is the purpose of swapping?

8.di If a process may **be** dynamically assigned to different locations in main memory, what is **the** implication for the addressing mechanism?

8.7 Is it necessary for all of the pages of a process to be in main memory while the process is executing?

81 Must the pages of a process in main memory be contiguous?

8.9 Is it necessary for the page's of a process in main memory to be in sequential order?

8.10 What is the purpose of a translation lookaside buffer?

Problems

8.1 Suppose that we have a multiprogratinned computer in which each job tiriil idequic.21. characteristics. In one iJortipuiaIiott period, *T.* far a job, half the lime is spent in 1.0 and the other half in proccy-.01....iclivity. Filch job runs for a total 01N ⟙wiods. Assume that a simple round-robin pliolity is used. and that I.[1]O opermioir, can overlap with processor operation. **De.figh...** We following quantities.;

- Tornaoloild time - actual time to complete a job
- liroughput – average number of jobs completed per time period *T*
- Processor utilization - percentage or time tllat the processor is active (not waiting).

Compute these quantities for orw, tv,o, and four simultaneous jobs. assuinifig that the period *T* is distributed in each of 1 he following ways;

A. 1..0 first half. processor second half

h. 110 first and fourth quarters, proixssur secoild and third quarters

8.2 An 110.bound program is ow I hai, if run alone, would spend more time waiting for I10 than using the prorx..4s1.1. A p3ticessor-bound program is the opposite. Suppose. a short-term scheduliri.1 iayon those programs that have used little processor time in the reo...ni psi. I why this algorithm favors PO-hound programs and yet does nut per denyprocessor time 10 processor-bound programs.

8.3 A program computes the row sums

$$C_{r} = \sum_{j=1}^{10} A_{r,j}$$

of an array A that is ID° by 10th Assume that the computer uses demand paging with a page size of Lon° ,vorth, and that the amount of main memory allotted for data is five page frames. Is there any difference in the page Fri ult rata if *A* were stored in virtual nu.tmory by rows or columns'? Explain.

8.4 Suppose the page table. for the process currently executing on the processor look; like the following. All numbers are derima], everything is numbered starting from zero, and all addresses are memory byte addresses. The page size is **1021** bytes.

Virtual Pap lumber	Valid hit	Reference hit	[nudity bill	frame number
0	I	I	0	4
1	I	1	1	1
	a	0	0	
	I	*b*	1)	2
4	El	D	0	—
5	I	I.)	I	cl

a- Describe exactly how. in genera]. a virtual address generated by the CPU is Mins-lated into a physical main memory address,

b. What physical address. it .kluy, would each or the following virtual addrows correspond to'? (Du not try 105 handle any page faults, if any.)

(i) 10.32.
2.22l
j41)9

Give reasons that the page size in a virtual memory system should be neither very small.. nor very large.

8.6 The following sequence of virtual page numbers is encountered in the course uf axe— cation on a competes' with virtual memory;

3 4 2 6 4 7 1 3 2 ri 3 5 L 2 3

Assume that a least recently used page replacemeni policy is adopted. Plot a graph of page hit ratio (fraction of page references in which the page is in math memory) as a • function Or main•memory page eapacily *n* ror 1 01 8. Assume that main memory is initially empty.

/4.7 in the VAX compute', lisor riage tables are 'located at virl Lid! addresses in the sys = space. 'i.Vhas is the advaniage of having user page tables in virtual rather than main memory? IrVfial is the disadvantage?

8,8 Consider a computer system with both segmentation and paging. When a segment is in memory, some words are wasted on the last page. In addition. for a segment sizes. and a page size p, there *are* *s/p* page 'able entries. The smaller the taupe size, the le s5 waste in the last page of the segment, but the larger the page table. What page Mire minimizes the total overhead?

8.9 A computer has a c.,ache. main memory, and a disk used ror virtual minor.... If a referenced word is in the cache, 20 ns are required to access ii. 1 r it is in inain memory but not ill the c Ht Feu, 60 ns are needed to ltlad it into Ihe Lathe, and then the reference i started nain. El the word is not in main memory, 12 ms are required to fetch the v,:iird four! ilkL. followed by 60 ns to copy it to the cache, and then the reference started again. The cache hit ratio is 0.9 and the main-memory hit ratio is 0.6. What is il.K1 average time in ns required to access a referenced word on this system?

8.10 Assume a task is divided into rout' equal-sized segments, and that the system builds an eight-entry page descriptor table ror each segment. Thus, the *sleet has a combination of segmentation and paging. Assume also that the page ske is 2 Kbytes.

B. **What is the** maximum size of each segment?

b. What is the maximum logical address space for the task?

e. Assume that an clement in physical location 00021ABC is accessed by this task. What is the format of the logical address that the task generates for it? What k the **Maximum.** physical address space for the system?

8.11 Assume a microprocessor capable oi' accessing up to 2' byi es of physical main memory. It implements ooe s• l: merited logical address space 471 maximum size 2 bytes. Each instructiori comairr-. IIre whole two-part addre.1... ternal memory management units (MMUs) art' **krnvi** whose management sehenie assigns contiguous blocks of physical memory c31 ked size 2 bytes to segments. .i'he starting physical address of a segthimt is always it isiltile by 1024. Show the delailed intereonnoction of the ester- "a1 mapping mechanism that converts logical addresses to physical addresses using appropriate number of MMUs. and show the detailed internal structure of an Mhil (assuming that each MMU contains a 121-entry directly mapped segment descriptor cache) and how each Mfy1l... is selected.

8.12 Consider a paged logical address space (composed of 32 pages of 2 Kbytes each) mapped into a 1-MByte physical memory space.

a, What is the format of the processor's logical address?

h. What is the length and width of the page table (disregarding the. "access rights"

e, What is the effect on the page Table it the physical mentors? •pac.e is reduced by half?

The Central Processing Unit

Up to this point, we have viewed rhe CPU as a "black box and examine have considered its interaction with 1/0 and memory. Part Three the. inlomai structure and function of the CPU, The CPU cofisists of a control unit registers, the arithmetic and iogic unit, the. instruction execution unit, and the interconnections among these components. Architectural issues, such as instruction sci design **arid** data types. ale. covered. The pari also looks at pipelining.

Chapter 9 Computer Arithmetic

Chapter 9 examines the funelinuality of the ALU and focuses on the rortsentiltion of num hers and techniques for implementing ,iiriihnictic operations. Processors typically support two types of arithmetic: integer, or fixed point, and floating point. For both c4isei;, the Chapter first examines the represnualion of numbers and i hen discusses arithmetic openaii pirs..1.hc important l 754 floating-point standard is examined in clutail.

Chapter 10 Instruction Sets: Characteristics and Functions

From a programmer's point of view, the best way to understand the op'] kiioa of ti processor is to learn the machine instruction set that it executes. The cc..1.1 npilL.7,; topic of instruc.1.tion set design occupies t'haptcs I() arid U. (hairier Eft focuses on the functional aspecisi of inistruction set design. The chapter examines the types of flinclions Ow are specified by complier instructions, and then lock!' **Spc c.ifiLLilly** at the types of operands (which specidata to he ciprated on) and the types of orxtralioll., (Which specify the

operations to be performned) commonly found in instruction sets. Then the rek:-tionship of processor instructions to assembly language is briefly explained.

Chapter 11 Instruction Sets: Addressing Modes and Formats

Whereas Chapter 10 can be viewed as dealing with the s,oniantics of instruction Chapter 11 is more concerned with the syntax of instruction sets. Specifically. CI: ter 11 looks at the way in which rn,mory icictrescoN ,art= srcified and at alp 01_ format of computer instructions.

Chapter 12 CPU Structure and Function

Chapter .12 is devoted to a discussion of the internal structure and function of thy processor. The chapter describes 1.13C use of registers as the CPU's internal memory. and then pulls together all of the material covered so far to provide. an overview of CPU structure and function. The overall organization (AEI:, control unit. register file) is reviewed. Then the organi'4ation of the register file is disciised. The rernitig-der of the chapter describes the functioning of the processor in executing nnichim; instructions. The instruction cycle is examined to show the function and inter-relationship of fetch, indirect, execute. and interrupt cycles. Finally. the use of pilw-lining to improve performance is explored in depth.

Chapter 13 Reduced Instruction Set Computers

The remainder of Pan Three looks in more detail at the key trends in CPU design. Chapter 13 describes the approach associated with the concept of a reduced instruc-tion set computer (RISC), which is one of the most significant innovations in corm puler organization and architecture in recent years. RISC architecture is a dramatic departure from the historical trend in processor architecture. An analysis of this approach brings into focus many of the important issues in computer oq!,aniz.ation and architecture. The chapter examines the motivation for the use of RISC design and then looks at the details of RISC instruction set design and RISC CPU architecture and compares RfSC with the complex instruction set computer (CISC) approach.

Chapter 14 Instruction_Level Parallelism and Superscalar Processors

Chapter 14 examines an even more recent and equally important design innova-tion: the superscalar processor. Although supersealar technology can be used on any processor, it is especially well suited to a RISC architecture. The chapter also looks at the general issue of instruction-level parallelism.

Chapter IS The IA_64 Architecture

The IA-64 instruction set architecture is a new approach to providing hardware sup-port for instruction-level parallelism and is significantly different from the approach taken in supersealor architectures. Chapter 1.5 begins with a discussion of the moti-vating factors for the new architecture. Net , the chapter looks at the general orga-nization to support the architecture. 'Me chapter then examines in some detail the kiev features of the IA-64 architecture that promote instruction-level parallelism,

CHAPTER

GO MPUTER ARITHMETIC

9.1 The Arithmetic and Logic Unii

92 Integer Representation

Sign•-Magnii tide Representation
l'Ivos Complcrnc.nt ReprQsen Lotion
Couverling howucn Different Bii Lengths
Fixed-Point Representation

93 integer Arithmetic

Negation
Addition **and** Subtraction
Mult ip.licut ion
Division

94 Eloating-Nint Repreit

Principles
rliF.F. Standard for Binary Fioating-Point Representation

95 Floating-Paint Arithmetic

Addition and Subtraction
:Multiplication and Division
Precision Consid CP1 tions
IEEE Stunc.hrd for l'Iinary Floating-Point Arithmetic

9.6 Recommended Heading and Vireb Site

9.7 Key Terms, Review .Questions, and Problems

Key Terms
KL-yi m Quesiions
Pnrilieins

KEY POINTS

- The two principal concerns rt.)1' co mputer arithmetic arc. the way in which liumbers arc represented (the binary format) and the algorithms used for the basic ariihmetie operations (add. subtract. multiply, divide). These two consideril• apply both to integer and floating-point arithmetic.
- Floating-point numbers are expressed as a number (sigoificand) multiplied by ui constant (base) raised to some integer power (exponent). Floating-pnini numbers can be used to represent very large and very small numbers.
* Most processors implement ihe. IEEE 754 standard for floating-point re„Px-sentation and floating-point arithmetic. lECE. 754 defines, both a 32-1111. acid a fro-hit format.

We begin our examination of the processor with an overview of the Faithie and logic unit (ALLT). The chapter then focuses on the most COM• Alex aspect of the ALU, computer arithmetic. The logic functions that iu't. part of the ALLT are described in Chapter 111 . and implementations of simple logic and arithmetic furictioro, in digital logic are described in Appendix A of this book.

Computer arithmetic is commonly performed on two very different types ut numbers: integer and floating point, In bg rt h eases, the representation chosen is a e ciai design issue and is treated •irst, followed by a discussion of ariihnielic opera:dm.

This eh:irate,- includes a number of examples, each of which is highlighted in a s box.

9.1 lig, WIWYJETic ANA,W(4,C, UNIT

The AIM is that part of the computer that actually perrorrns arithmetic and logical operations on data. All of the other elements of the computer system•control unit, registers, memory, I.10—are there mainly to bring data MI° the AL[; for it 10 process and then to take the results back out. We have, in a sense. reached the eon: or essence of a computer when we consider Ihe AUL

An ALL) and, indeed, all electronic components in the computer arc based on the use Of si mple digital logic deices that can store binary digits and perform si mple Boolean logic operations. For the interested reader, Appendix A cm km digital logic implementation.

Figure 9-1 indicates. in general terms, how the ALU is 411cl connected with the rest of the processor. Data are presented to the AU! in registers, and the results Qf an operation are stored in registers, These registers are temporary storage leentiost& within the processor that are connected by signal paths to the ALU (e.g., see Figure. 2.3). The AU; may also set flags as.the result of an operation. For example, an ovcr• flow flag is set to 1 if the result of a computation exceeds the length of the registu:

Figure 9,1 A LAJ Inputs Lind Outputs

into which it is lo be stored. The flag values are also stored in registers within the processor. Thc control unit provides signals that control the operation of Ilse ALU and the movement of **the** data into and out of the ALL],

9.2 INTEGER REPRESEN1ATION

In the binary **nurnbcr** system,' arbilrary' numbers can he represented with lust the digits **zero** and onc, the minus sign, and the period , or radix point,

—1101.0101 $_2$ = —1331?5 o

For purposes of computer storage and processing, however, we do not have the benefit of minus signs and periods. Onkv binary digits (0 and 1) may be used Lc; represent numbers. If we **are** li mited to nonnegative integers, the representation is straightforward.

An 8-bit word **can** represent the numbers from 0 to 255, including

00000000 0
(1(1000001 = I
00101.001 41
11000111[10 = 128
1].111111= 255

In general, if an n-bit sequence of binary digits $_t a_{n-}$, , $a_{.}$, *is* **interpreted as** an unsigned inte2er A, its value is

'Sec Appendix B For rI buhic rofnzsl m **flii m hcr s** :..sterns (dclzi mak, bin dry, h.z.x.ackcirnal).

A =

Sign-Magnitude Representation

There. are several alternative conventions used to represent negative as well at positive integers, all of which involve treating the most significant (leftmost} hit in the word as a sign bit, If the sign bit is 0, the number is positive: if the sign bit is t, the number is negative.

The simplest form of representation that employs a sign bit is the sign ' magnitude representation. In an n-bit word, the rightmost n — 1 bits hold the magnitude of the integer.

-18— 00010010
—1S— 1001.0010 (sign magnitude)

The general case can be expressed as follows:

Sign Magnitude A = $\begin{cases} \sum_{j=0}^{n-2} 2^j a_j & \text{if } a_{n-1} = 0 \\ -\sum_{j=0}^{n-2} 2^j a_j & \text{if } a_{n-1} = 1 \end{cases}$

There are several drawbacks to sign-magnitude representation. One is that addition and subtraction require a consideration of both the signs of the numbers and their relative magnitudes to carry out the required operation. This should become clear in the discussion in Section 9.3. Another drawback is that there are two representations of 0:

+ 0_{sm} = 000000po
o_{sm} = ib0000po (sign magnitude)

This is inconvenient. because it is slightly more difficult to test for 0 (an operation performed frequently on computers) than if there were a single representation.

Because of these drawbacks, sign-magnitude representation is rarely used in implementing the integer portion of the ALU, instead, the most common scheme is twos complement representation.'

Twos Complement Representation

Like sign magnitude, twos complement representation uses the most significant it as a sign bit. making it easy to test whether an integer is positive or negative. It dii-

In the literature, the terms rwo75 *complement* or 2'r *complement* are often used, Here we follow the practisn LiNed in standards documents and omit the apostrophe (e.g., IEEE Std 101}-1 903. The New ?FEE SwF. *elan! Dictionary of Elearicat and Electronics ferns).*

TatFle 91 CharaCtcriStics 1wL5s Complement Representation and Arillintetio.

Runge	through $2^{n-1} -$
umber of Rep-yen:notions' of zo1·o	One
7.* e- ntion	'rake 11·ic Flog5lean ccolipleinent of each bit of she corre.sponclin!F poi•riiive number. Chen add L to the resulking bit pa.tirn An unsigned
ExtwFFSiou 4).F Bil Length	Add hit FuritionE Lo Ike. lett and fill in with the xi:011C of origiiiiii Nip·3 hit.
Overflimo Rohl	II two numbers with the s.ime sign (both iac}3iti\:e Of both Ilega-Live) are added.. then (m1·flow occuts it and only if the resull has the opposite sign.
Siihtrurtion Rule	To subtract *B* from *A* lakc c he twa.i. complement irf 13 0Eld add ik to *A*

fors from the use of the sign-magnitude **representation** in the way thin the c Fi her hits ;Ire inlerprel Lit I 411)1e 9.1 highlights key characteristics of two:, compiement representation and ti'iLitlietiC, which are elaborated in this section and the next.

Most treatments of Twos complement representation focus on the rules for producing negative Mini bcrs, with no formal proof that the scheme ¯wt·nrk.s.. I nstead, our presentation of twos complement integers in this section and in Sectibn 9.3 is based on [DATT93], which suggests that twos complement representation is best understood by defining it in terms of a weighted sum of bits, as we RI previously for unsiEtned and sign-magnitude representations. The advantage of this treatment is that it does not leave any lingering doubt that the rules for arithmetic operations in twos complement notation may not work for some special f:Tises.

Consider ari a-hit integer, *A,* in twos complement repre71:.eriG)tion.. **1f** A is positive, then the sign bit, is zero. The remaining bits tc..precnit the magnitude of the. number in the same fashion as tot · sign magnitude:

$$A = \sum_{i=0}^{n-2} a_i 2^i \quad \text{for } A \geq 0$$

The number zero is identified as positive and therefore has a 0 sign bit and.a magnitride of **all OS, WC** can see that the range of positive integers that magi be represented is from 0 (all of the magnitude bits are 0) through $2^{n-1} - 1$ (a]1 of the magnitude bits are I). Any larger number **would** require more bits,

Now, for n 3ievrive. number A (A (1), the sign bit, *a,* is one. The remaining $n-1$ bits wan take on any one of 2' values. 'Fbewfore, the range of negative integers that can be represented **is** from —1 to -2^n I. We would Like to assign the **bit** values to negative integers in such **s** way that arithmetic can be handled in a straightforward fashion, similar to unsigned integer arithmetic- In unsigned integer representation, to compute the value of an **integer from the** bit representation, the **weight** of the most significant bit is $+2^{n-1}$ For a representation with a sign hit, it turns out

Ihe desired arithmetic properties are achieved, as we will see in Section 9.3, if

the weight of the most significant hit is -2" [1], This is the convention used in twos complement representation, yielding the following expression for negative n umbers:

Two Complement $$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

In the case of posiiive integers, $0.$ so the term $-2:" \text{ '0,, } , = 0.$ Therefore, Equation (92) defines the iwos complement representation for both positive and negative numbers,

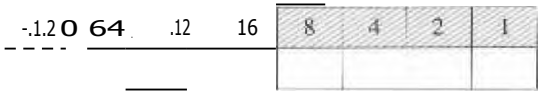
'Fable Q2 compares the sign-magnitude and twos complement representations for 4-hit integers. Although iwos complement is an awkward representation from the human point of view, we will see that it facilitates the most important arithmetic operations, addition and subtraction, For this reason, it is almost universally used as the processor representation for integers,

A useful illustration of the nature or twos complement representation is a value box, in which the value on the far right in the box is 1 (2^{u}) and each succeeding position to the left is double in v;iltie. until the leftmost position, which is nepta. As you can see in Figure 4,2a, the most negative twos complement number that can be represented is $-2!'$ if any of the hits other than the sign bit is one, it adds a positive amount to the **number.** Also, it is clear that a negative number must have a 1 at its leftmost position and u positive number must have a 0 in thai position. Thus, the largest positive number is a 0 followed by all ls, which equals 2" 1 .

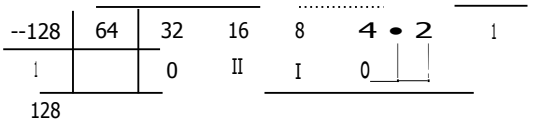
The rest of Figure 9.2 illustrates the use of 1he value box to convert from twos complement to decimal and from decimal to Twos complement.

Table 9.2 Alternative RwreseilLaticros for 4-Bit **Integers**

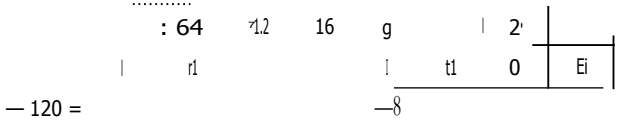
Decimal Rovesentation	Sign-Magnitude ICepreseniMion	'11441014 Complcrnent Hepresentation	Biased RepreRentation
-FI	—	—	1111
-7	0111	0111	11 W
-Ft)	0110	01.10	1101
+5	0101	01.01	L Ifx1
44	0100	0100	1011
+3	0011	0011	1010
-2	i.11(.0	0010	I W1
-1	;;o0I	0001	I WO
+0	r.,i.I0!!	WOO	0111
-4.1.	I.:11.10	—	
1	I1X11	1111	OLIO
-2	1010	1110	010.1
-.•	1011	3101	0100
-4	1100	1100	0011
-5	1103	1011	0010
-6	111.0	1010	I1(301.
-7	1111	WIJI	1:%101}
43	—	LOW	—



I:I(**cil.** 1n-po3lii.on twos cornpleinU111 **Value.** box



(b) Con vc rt binEiry I **MO(X) 1 1** to decimal



(c) Correcn dccinvil —120 co binary

Figure 9,2. of n 'Value Box **fur Co unieTSlO** hetwcen Twos Ci.prrirlinnwli Binary and Decimal

Converting between Different Bit Lengths

It is sometimes desirable to Lake an 02-bit integer and store it in m bits, where $m > n$. In sign-magnitude notation, this is easil!,' accomplished; Simply move the sign it to the nCvi leftmost position and **fill** in with zcros.

+18 —	00010010	(sign magnil tide, N hits)
	00.1100f1(0000 1100 I 0	(sign mtignilude. 16 bits)
-- 1•	.1 001.00 10	(sign magnitude. 8 bits)
	1000000000010010	(sign magnitude, 16 bits)

This procedure will not work for **twos complement** negative integers'. Using the same eumple,

-F tH	0€1010010	(twos complement., 8 bits)
+18	001)111 01101010010	(twos complement, 16 bits)
-•	11.101110	(I wos complement, 8 **hits}**
--32,658	1000000001101110	(twos complement, 16 bits)

'Me next to **last hue is** easily seen using, the box of Figure 9-2- The last line can be verified using ion (9.2) or a 16-bit value box.

lnstcad, the rule for twos complement integers is to move the sign hit to the new Leftmost position and fill in with copies **of the sign** For positive numbers,

fill in with zeros, and for negative numbers. fill in with ones. This is called sign e.xtc:nsion-

lg	11101110	(t.wm complement,	hits)
lg = 11111U		(twos complement,	.l6 bits)

To see why this rule wOrks, let us again consider an fl-bit sequence of binary digits $a_{n}, a_{n}, \ldots, a_{1}a$ interpreted as a twos complement integer A. so that its value is

$$A = -2' \; E\, 2'a,$$

HA is a positive number, the rule clearly works. Now, if A is negative and we want to construct an nt-bit representation, with ,n > *'Then*

$$A = \quad 'a_{n} - I- \quad ;-11$$

The two values must he equal:

$$- \quad + \quad = -2'' + \quad 2^{1}a:$$

$$2;a, -$$

$$2'' \quad \mathbf{I} \quad 2'a, =. \; 2'$$

$$4 \; E \quad 2'a_{;} . = i + \quad 2'$$

$$2' \bullet =. \; E \; 2'$$

$$2 - \quad - \quad - 7 = ra$$

In going from the first to the second equation, we require that the least significant $n - 1$ bits do not change between the two representations. Then we get to the next to last equation. which is only true it all of the bits in positions rr $- I$ through *fit* 2 are 1. Thus the sign-extension rule works.

Fixed-Point Representation

Finally, we mention that the representations discussed in this section are. sometimes referred to as fixed point. This is because the radix point (binary point) is fixed and assumed to be to the right of the rightmost digit. The programmer can use the same representation for binary fractions by scaling the numbers so that the binary point is implicitly positioned at some other location.

9.3 INTEGER ARITHMETI

This section examines common arithmetic functions on numbers in mos complement representation.

Negation

In sign-magnitude representation, the rule for forming the negation of an integer is simple; invert the sign bit. In twos complement notation, the negation of an integer can he formed with the following rules:

1. Take the Boolean complement of each bit of the integer (including the sign **bit).** That is. set each 1 to 0 and each 0 to 1.
2, Treating the result as an unsigned binary integer, add 1.

This Iwo-step process is referred to as the **twos complement operation,** or the taking of the twos complement of **an** integer,

+.1g =. 00010010 (twos comp iem ent)
bitwi se complement **11101101**
lli01110 **—18**

experied, the. negative of **the negative of that number** is itself:

1/01110 **(twi.3s complE..niv.ni)**
wise COOT[errtent = 00010001
Gan 0010= +18

We can demcinstraLe the validiiv ul the operation **just ie.scrilied** using the definition of the twos complement representation in Equation (9.2). Again, interpret an ;sequence of binary digits a, *a, a, as* **a twos-complement integer A, so that its value** is

$$= 2^{)} \quad | \quad 2`a;$$

Now form **the** bi1virise cornple,menl. and, treating Ibis is **an** unsigned integer, add 1. Finally, ihterprel the resulting n-hit scquentx of binary digits as a twos-complement integer ***B,*** so that its value is

$$B = -2^{"} \quad '0,, \quad 1 - F E.2.' a:$$

Now, we want A = — *B,* which means $A + B = 0$. This is easily shown to he true:

$$A \quad B = -(a_{n-1} - \quad .702'' \cdot + I - 1 \ (I2V \ ^{+1} \ a_f)$$

$$2'^{1} \cdot 1 + (24-^{1} - 1)$$

$$- \quad 4 \cdot 2' = \mathbf{O}$$

'1'he prml:..ding derivation assumes that we can first treat the bitwise eomplement A as an unsigned integer for the purpose of adding 1, and then treat the result aE a twos complement integer. There are two special cases in consider, First. considu A 0. **In that** case, for an 8-bit representation.

bitwise oomplement =	004100000 (tvitys complement) •IL1111111 toop00000 0	

There is *carry* out of Ihe most significant bit position, which is ignored. The result is that the negation of 0 is 0, as it should be,

The second special case is more of a problem- **If** we take the negation of the hit pattern of I followed by n — 1 zeros. we get bad the same number. For example. for 8-bit words,

	LZS — 100000E.10 (twos. complem en 19	
bitwise complement	011111.11	
	100000000 —128	

Some such anomaly is unavE_Iithible. The numbcr of different hit patterns in an el-bit word is 2', which is an even number, We wish to represent positive and ile.4-tivc integers and 0. H an equal number c.if positive and CLCE/iLiVe integers are repr• wonted (sign magnitudc): then there are two representations for EL If 'here is only one representation of 0 (twos complement), then there must be an unequal number or negative and positive numbers represented. In the case of LWOS complement. Cot iii n-bit length, there is a representation for —2' but not for +2''.

Addition and Subtraction

Addition in twos complement is illustrated in Figure 43. The first four examples illustrate successful operations, **If** the result of the operation is positive, we get 8 positive number in ordinary binary notation. II the result of the operation iE negative, we get a negative number in twos complement form. Note that, in soma

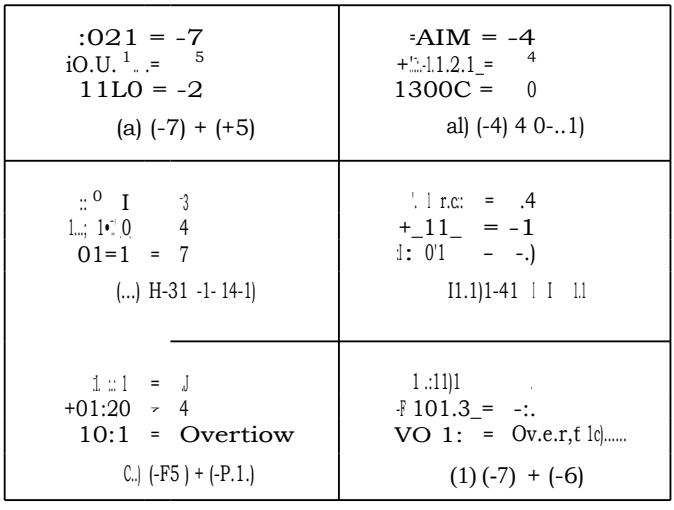


Figure 9.3 Addition or Numbers in Twos Coniplemetil Rciprt'sunta Lien

instances, there is a carry bit beys nd the end of the word (indicated by shading), which is ignored.

On any addition. the result may be larger than can be held in the word size being used, This condition is called overflove, When overflow occurs, the ALU must signal this fact so that no attempt is made to use the result. To detect overflow, the following. rule is observed: If two numbers are added, and they are hoh poitive *or* bOth negative, then overflow occurs if and only if the result has the opposite sign. Figures 13e and I' show examples of overflow, Note that overflow can occur whether or not there is a carry.

Subtraction is also easily handled with the following rule To subtract one number (subtrahend) from another (minuend), take the twos complement (negation) of the subtrahend and add it to the minuend. Thus, subtraction is achieved using addition. as illustrated in Figure 9.4. The last two examples demonstrate that the overflow rule still applies.

Some insight into twos complement addition and subtraction can be gained by looking at a geometric depiction [BENI-192]. as shown in Figure 9.5. '11 u.: circle in the upper half of each part of the figure is formed by selecting the appropriate segment of the number line and joining the endpoints. Note that when the numbers are *laid* out on a circle, the twos complement of any number is horizontally opposi e that number (indicated by dashed horizontal lines), Starling at any number on the Circle, we can add positive .1((or subtract negative *k)* to that number by moving *k* positions clockwise. and we can subtract positive *k* (or add negative *k)* from Ihai number by moving *k* positions counterclockwise. If an arithmetic operation results in traversal of the point where the endpoints **;ire** joined, an incofrect answer is given (overflow).

Ail *of* the examples of Figures 9.3 and 9.4 are easily traced in the circle of Figure 9.:5.

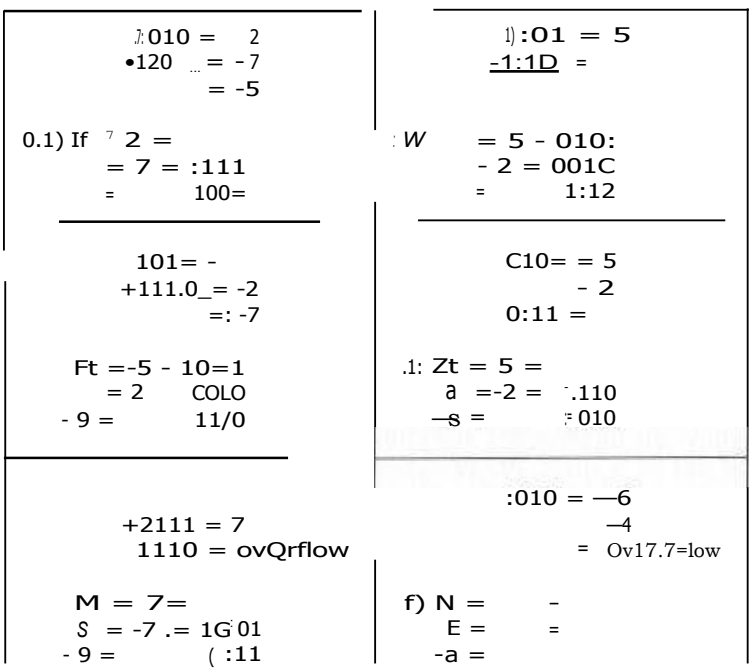
Figure 9.6 suggests the data pais and hardware elements needed to accornpiibh and subtraction. l he central element is a binary adder, which is presented two numbers for addition and produces a sum and an overflow indication. The binary adder ire...Nis the two numbers ais unsigned integers. (A logic implementation of an adder is given in Appendix A.) For addition, the two numbers arc prese.nli:d to the adder from two 1 c Ri!,ters, designated in this case as A and 13 registers. The result may be stored in one. of these registers or in a third. '1'he overflow indication is stored in a 1-bit overflow flag (0 — no overflow; 1 4 overflow). For huh. traction, the subtrahend (B register) is passed through a Lwos complemc.nter so that its twos complemeni is presented to the adder.

Multiplication

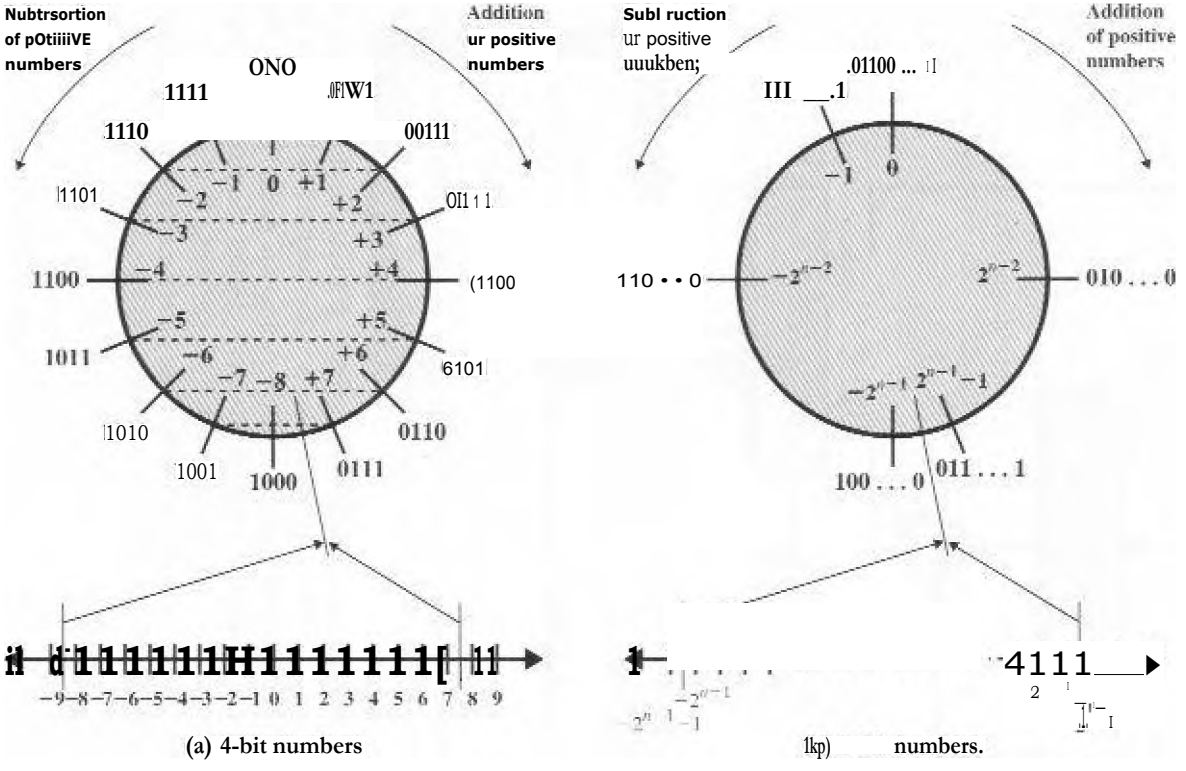
Compared with addition and subtraction, mulliplication is a complex operatitm, whether performed in hardware or software. A wide variety of algorithms have been used in v,Hrious computers. The purpose of this subseciion is to give the reader some feel for the type of approach typically taken, We begin with the simpler problem of multiplying two unsigned (nonnegative) integers_ and then we look at one of the most common iechniques for multiplication of numbers in twos complement representation,

Unsigned Integers

Figure 9.7 illustrates the multiplication of unsigned binary integers, a might he eta riled out using paper and pencil. Several important observations can be made;



Fiore 9.4 Subtraction of Numbers in Twos Complemmt Koprc8cmkation S)

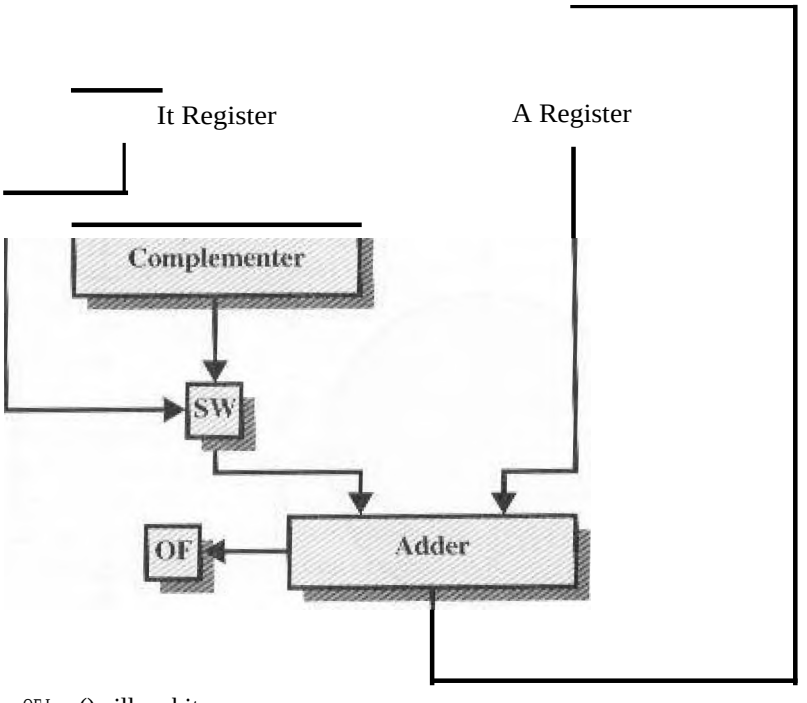


(a) 4-bit numbers

lkp) numbers.

Figure 9.5 Geometric. Depiction of Twos Compliment lntegen

tai



OF L..-- ()willow bit
SW = Switch t3elect Addition or subtraction)

Figure 9.l Block Diagram of Hardware for Addition and Subtraction

1. Multiplication involves t.hc generation of partial products. one for each digit in the multiplier. These partial products arc then summed to produce the final product.
2. The partial products arc easily defined. When the multiplier hit is O. the partial product is 0. When the multiplier is I, the partial product is the multiplicand.
3. The total product is produced by summing the partial products. For this operation, each successive partial product is shifted one position to the left relative to the preceding partial product.
4. The multiplication of two n-bit binary integers results in a product of up to 2, 'l bits in length (e.g., 11 X 11 = 1001 }.

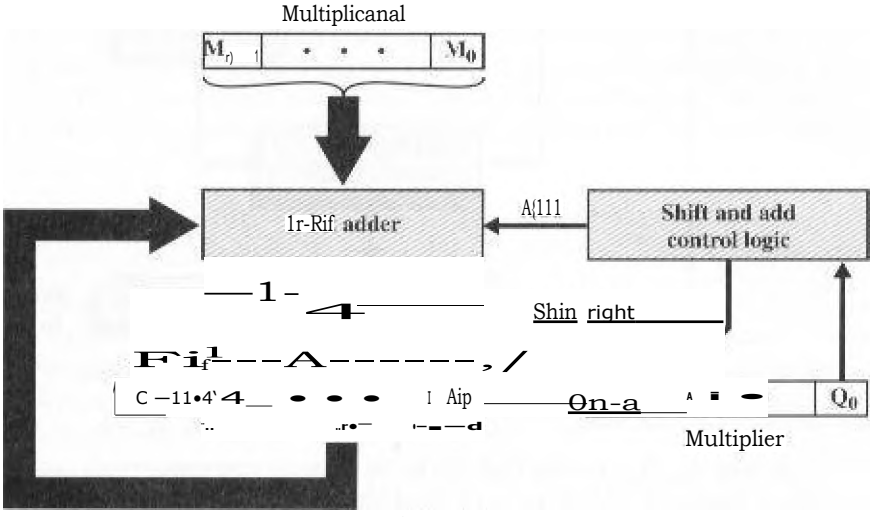
1 01 1	Multiplicand {11)
1 1 0 1	Multiplier {13)
1 0 11	
0 0	Partial products
1 0I 1	
10 1 1.	
10001111	Product (143)

Figure 9.7 Multiplication of Unsigned Binary Integers

Compared wilh 11-10 pencil-and-paper approach, there are several things we can do to make computerized mottiplication more efficient_ First, we can perform a running addition on the partial products rather than waiting until the end. This eliminates the need for storage of all the partial products; fewer registers are needed. Second, we can save some time on the generation or partial. products, For each I on the. multiplier, an add and a shift operation are required; but for each 0, only a shirt if required.

Figure 9.a shows a possible implementation emplovin.g these measures. The multiplier and multiplicand arc loaded into two registers {Q and MY A third register, the A register, is also needed and is initially sit 10 0, .There is also a 1-bit C: rcgistcr, initialized to 0, which holds a potential carry bit resulting from addition.

The operation of the multiplier is.as follows. Control logic reads the bits of the multiplier one at a time. if Q,, is 1, thin the multiplicand is added to the A reg-



ta) Block diagram

r					
	l: On	110.	1011	Initial va;ues	
	10 -1	1 ·.01	1011	Ada	Firs:
	;1:i	1110	10=1	S'nift	cycLe
		1111	10=1		Second cycle
	101	1111	:011	Ldd	Third
0.	0110	1111	1011	Sh:ft	cycle
1	O:)0_	11'L1	101:	Ada	Forth
0	1000	11L1	1C.1:	Sh 1ft	*cycle,*

{ht Example la-43 ni Figt.pnr. 9.7 (product in A, Q1

ngure 9.8 I I ii1·dwa rc I m plcmen latiun of UnNigncd Binary Multiplication

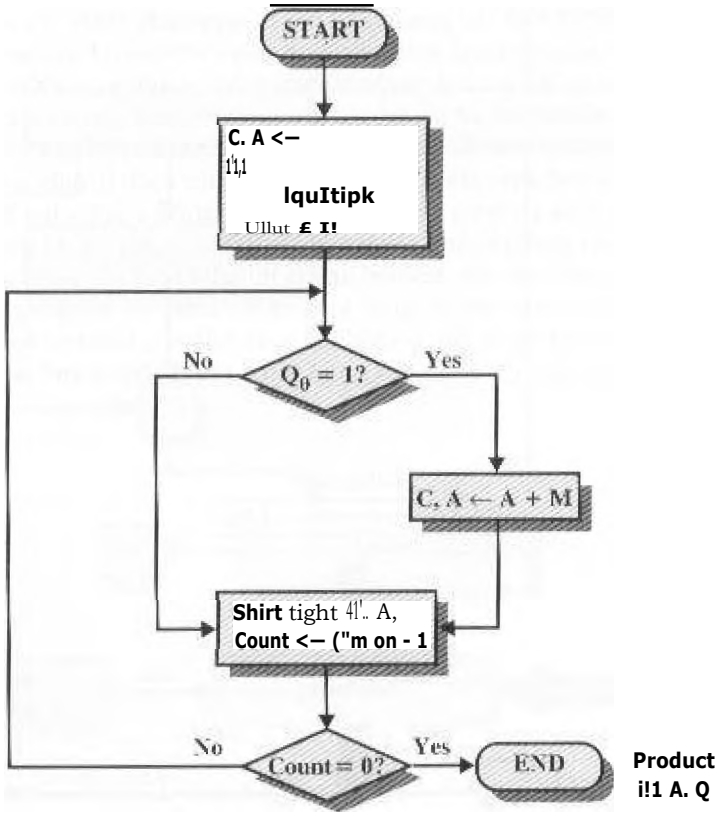


Figure 9.9 Flowchart for UnNi.pi<ld BirmrY

ister and the result IS siored iu the A register, with the C hit used for overflow. Then al] of the. bits of *the* C, A, and 0 registers are shifted to the right one hit, so Thal t he C bit goes into A,, goes into and 0,, is lost. llf 0,, is 0, then no addi tion is perCormed, jug the shift, 'this process, is repeated for each bit of the original multiplier, The resulting 24z-bit product is contained in the. A and 0 registers, A flowchart of the operation is shown in Figure 9.9. and an example is given in Figure 9.8h, Note that on the second cycle. when the tnultiplier bit is 0, there is no add operation.

Twos Complement Multiplication

We have seen that 1.cidition and subtraction can he performed on numbers in twos complement nOiaLian by tre.4i ling them as unsigned integers. **COnSicict**

L a C.1
– .3 C
1L.3

if **L. se** numbers are considered to be unsigned integers, then we ;ire. adding 9 (➲ plus 3 .({101.1), to gel 1 2 (1100)- **As** twos complement integers, we are adding —7 (1001) io 3 (0011) lo get —4 (1100),

```
        1:.01
-------------------
0002.10L1     1:1] x 1
              1211> O)42
0210:1:0      11)11   1 x2
0101:02 0     10=1 xl
-------------------
102011.11
```

Figure 9.10 Multiplication of Two Unsigned 4•Bit Intugcrs Yielding and 8-Bii

Unfortunately, this simple scheme will not work for multiplication. To see this. consider again Figure 93. We multiplied I I 0011) by 13 (1101) to. get 143 (10001111). If we inlcrpreL these as twos complement numbers. we have —5 (1011) ti mes —3 (1101) equals —113 (10001111). This example demonstrates that straightforward multiplication will not work if both the multiplicand and multiplier are negative. In fact, it will not work if dither the mul1ipliCand or the mulliplier is negative. To justify this statcment, we need to go hack to Figure 9.7 an.d explain what is being done in Terms of operations with powers of 2. Recall that any unsigned binary numbv.r can he expressed as a surn of powers of 2, Thus,

$$1101 = 2^2 + 0 \times 2^1 + 1 \quad 2^a$$

Further_ the multiplication of a binary number by 2' is accomplished by shifting dial number to the left *n* bits. With this in mind, Figure 9.10 recasts Figure 9.7 io make the gcrwration of partial products by multiplication e74plicit. The only difference in Figure 9.10 is that it recognizes that the parlia I products should **he** vie.o.red as 2n-bit numbers generated from the **multiplicand.**

Thus, as an unsigned integer, the 4-bit multiplimnd 1011 is stored in an 8-bit word as 00001011. Each parlial product (other 1h4in that for 2) consists of this number shifted to the left, with the unoccupied positions on the right filled with zeros (e.g., a shift to the left of two places yields 00101100).

Now we can demonstrate that straightforward multiplication will not work if the multiplicand is negative. The problem is that each contribui ion of the negative multiplicand as a partial producl must be a negative number on a 2n-hil field: the sign hits of the partial products must line up. This .k demonstrated in Hgure 9.

```
      1C.01                          1201   (-7)
    x0011   (3)                    x00:1   (3)
0 :0.7.1001   1•20:                 x      -
              1:::01  x  2          I' 1 ·.0010_ (-.7;  x  2  =  -141
OC.0110=1                          1::  011   (-21)
```

(a)1.1migneil integer.; 1:13) Twos cotapluitiont inieger!,

Figure 9.11 ComparKon of Multiplication of 1.:nsignEd and Twos (-...oirpletnent

which shows that multiplication of 1001 by 0011. If these are treated as unsigned integers. the multiplication of 9 x 3 proceeds simply. However, if 11)01 is interpreted as the twos complement value —7, then each partial product must be a negative twos complement number of 2n (8) bits. as shown in Figure 9.11b. Note that this is accomplished by padding out each partial product to the left with binary Is.

If the multiplier is negative, straightforward multiplication also will not work. The reason is that the bits or the multiplier no longer correspond to the shifts or multiplications that must take place. For example. the El-bit decimal number —3 is written 1101 in twos complement_ Il' we simply took partial products based on each hit position, we would have. the following correspondence:

$$1101 \qquad (1 \times 2^3 + 1 \times \quad + 0 \times 2^1 + 1 \times 2^0) \quad - (2^1 - \quad - 2^0)$$

In fact., what is desired is $(2^4 - 2^0)$, So this multiplier cannot be used directly in the manner we have been describing.

There are a number of ways out of this dilemma. One would be to convert both multiplier and multiplicand to positive numbers, perform the multiplication, and then take the twos complement of the result if and only if the sign of the two original numbers differed, Implementers have preferred to use techniques that do not require this final transformation step. One of the most common of these is Booth's algorithm. This algorithm also has the benefit of speeding up the multiplication process. relative to a more straightforward approach.

Booth's algorithm is depicted in Figure 9.12 and can he described as follows. As before, the multiplier and multiplicand are placed in the Q and Ni registers, respectively. There is also a 1-bit register placed logically to the right of the least significant bit (0,,) of the 0 register and designated 0 its use is explained shortly. The results of the multiplication will appear in the A and Q registers. A and 0_, are initialized to 0. As before. control logic scans the hits of the multiplier one at a time. Now, as each hit is examined, the bit to its right is also examined. If the two hits are the same (1-1 or (1-0), then all of the hits of the A, Q, and **0.** registers are shifted to the right 1 hit. If the Iwo hits differ, then the multiplicand is added to or subtracted from the A register, depending on whether the two hits are 0-1 or 1—). Following the addition or subtraction. the right shift occurs. In either case, the right shift is such that the leftmost hit of A. namely A,. ,_ not only is shilled into A but also remains in A„ ;. 'Ellis is required to preserve the sign of the number in A and 0. It is known as an **arithmetic shift,** because it preserves the sign bit.

Figure 9.13 shows the sequence of events in Booth's algorithm for the multiplication of 7 by 3. More compactly, the same operation is depicted in Figure 9.14a. The rest of Figure 9.14 gives other examples of the algorithm. As can he seen, it works with any combination of positive and negative numbers. Note also the efficiency of the algorithm. Blocks of Is or Os are skipped over, with an average of only one addition or subtraction per block.

Why does Booth's algorithm work? Consider first the ease of a positive multiplier. In particular, consider a positive multiplier consisting of one block of is surrounded by Os (for example. 90011 l 10), As we know. multiplication can be achieved by adding appropriately shifted copies of the multiplicand;

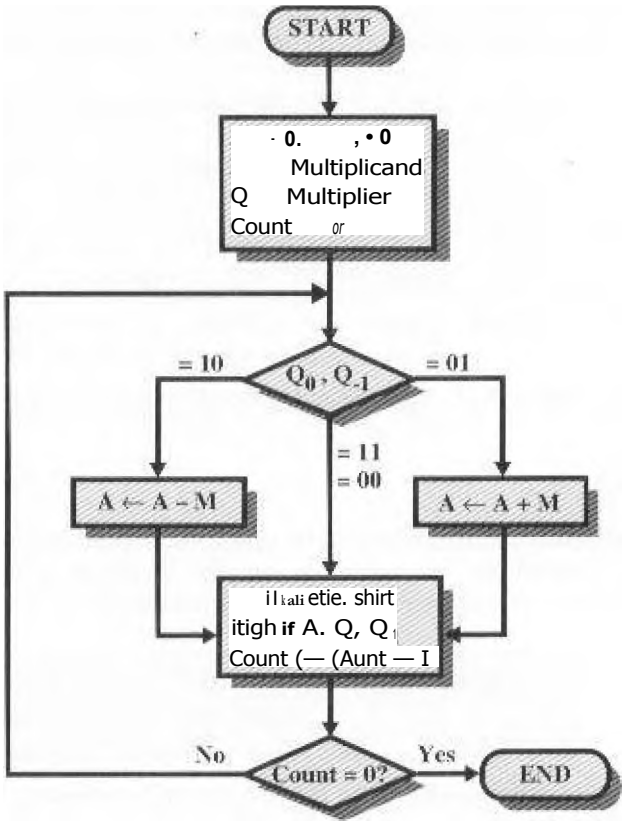


Figure 9,12 Booth's Algorithm lor Twos Complement Nioltiplica(ian

$$\begin{aligned} M \times (00011110) &= M \times (2^4 + 2^3 + 2^2 + 2^1) \\ &= M \times (16 + 8 + 4 + 2) \\ &= M \times 30 \end{aligned}$$

The number of such operations can I-'c reduced to two if w ()Nerve that

$$\begin{aligned} Mx\ 00011110) \qquad & (2' - 2^{1}) \\ &= M \times (32 - 2) \\ &= I \times 3U \end{aligned}$$

	02'11	0	0:11 '		
12.01	C011	C.	C.1=1	A •.— A r•f 1	First
11.20	100=	1	:111	Shit	cycle
1110	010r!	1	0=11	Shit:	
0:01	2120	,	(4:1	A - M 1	TILrd
0010	1010	0	:111	..hitt	cycl1
C0C.1	0:01	0	0:1:	Shi f.'T- }	Four:n

Figure 9.13 Example of Booth's A[goi-ithrn (7 X 3)

So the product can be generatced by one addition and one subtraction of he multi' plicand. This scheme extends to any number of blocks of ls in a multiplier% including the Qaz;• III which a single 1 is treated as a block.

$$M \times ((1IL\ L\ i010) = Sx\ (2'' - 1\%.1 \times (2 \quad 2^{1} - F.\ 2^{i}) \quad i- \quad \}$$

Booth's algorithm conforms to this scheme by performing a subtraction when titt first F of the block is Qncountered (1-11) and an addition when the end of the Hock is encountered ({1-1).

O:1, x0Gi11 (0 '. 1L1116.a1 1-0 0000002 1-1 .1 0:a1=1 0-1 207101 (_ +.2:) im) (7).x .11)= (2))	CcIll x1101 (C.) 111:1001 1-0 0000111 0-1 Ill D.. 1 1-0 1. 110:011 1.21:. 13) 0) X (.-3) = (-21)
1 021 X001 1:0) .2 0.00241= 1-.2 2.000020 , _ 111'.0 C-1 1 -12.1011 (-,•-•"_) (0 (-7) x (31= (-21)	10:.:1 x:.101_ .:0) 0000:111 1 -2. 11=1001 7-1 .0 02'11: 1-0 C0010101 (21) (d) (-'n x (-31= (2)1

Figure 9.14 Examples Using Booth's Algoriihm

To show that the .amc scheme works for a negative multiplier. we need to observe the following, I.& X be.s negative number in twos complement notation:

$$\text{Representation of } X = \{1x_{,,} \quad {}_{3}\bullet - .1\ {}_{1}ie_{f},'$$

Then the value of X can he expressed as follows:

$$X = \quad + (x_{,,} \times \quad {}^{2}) + \quad x \quad - ,, + \quad X\ 2^{1}) + (x_{,,} x\ 2') \qquad (9.4)$$

The reader can verify this by applying the algorithm to the numbers in ·rabic 9.2.

'fhe leftrniast hit of X is 1, because X is neptive. Assume. that I he leftmost 0 is in the kth position. Thus. X is of tic form

$$\text{Representation of } X = \quad I \ldots 10x_{,,} \quad ,, \qquad (9.5)$$

Then the value of X is

$$X = \quad I + \quad + \quad h" \text{-F.} \quad x21^{.1}) - \quad 4 \quad X\ 2') \qquad (9.6)$$

From 17i,ti uation (93), we can say that

Rearranging,

$$\text{-F} \quad + 2" \quad + ft^{441} = \qquad (9.7)$$

Substituting Equation (9.7) into Equation (9.6). we have

$$X \quad \mathbf{r'} \quad (x_{A}\ .\ ,\ X2\ ') \text{ -F} \quad (_{..}t_{n}\ X\ 20)$$

Al e?in return to Booth's algorithm. Remembering the representation of X [Eguntion (9.5)], it is clear that all of the hits from $x_{,,}$ up lu the leftmost 0 arc handled properly. because they produce all of the terms. in Eiluation (9,8) but) and thus are in the proper form. As the algorithm scans over the leftmost 0 te nd 4;Ticounti2IN the next 1 ($2^{k\ I}$), a 1-0 transiiion occurs and a subtraction takes place (-2' •). This is the remaining term in Equation (9.6).

As nn example. consider the multiplication oafsome multiplic-and by (-6). In twos complement representation, using an 8-hit word, (- () is represented as 11111010. By Equation (9.4), we. know that

$$-6 \quad + + \quad 2^{4}\ 2^{)} + 2'$$

which the reader can easily verify. Thus,

$$11{,}1\ x \quad 11.010) \quad hi \quad (-2^{7}\ 4\text{-}\ 2^{6}\ \mathbf{f}\ 2^{5} - h\ 2^{4} + 2^{3} + 2')$$

Using Equation (9.7),

$$M(11111010) = M \times (-2^3 \; 1\; 2')$$

which the reader can verify is still M v (--(). Finally. following our earlier line of reasoning,

$$\text{Ni} \quad \textbf{(111.11010)} \quad \times (-2` 4 \quad - 2')$$

We can see that Booth's algorithm conforms to this scheme. It performs a subtraction when the first 1 is encountered (1-0), an addition when (01) is encountered, and finally another subtraction when the first 1 of the next block **Of is** is encountered. Thus, Booth's algorithm performs fewer additions and subtractions than a mom straightforward algorithm.

Division

Division is somewhat more complex than **multiplication** but is based on the same general principles. As before, the basis for the algorithm is the paper-and-pencil approach, and the operation involves repetitive shifting and addition or subtraction.

Figure 9.15 shows an example of the long division of unsigned binary integers. It is instructive to describe the process in detail. First, the bits of the dividend examined from left to right, until the set of bits examined represents a number greater than or equal to the divisor; this is referred to as the divisor being able to divide the number. Cintil this event occurs. Os are placed in the quotient from left to right. When the event occurs, a 1 is placed in the quotient and the divisor is subtracted from the partial dividend. The result is referred to as **a** *partial remainder*. From this point on, the division follows a cyclic pattern. At each cycle, additional bits from the dividend are appended to the partial remainder until the result is greater than or equal to the divisor. As before, the divisor is subtracted from this number to produce a new partial remainder. The process continues until all the bits of the dividend arc exhausted.

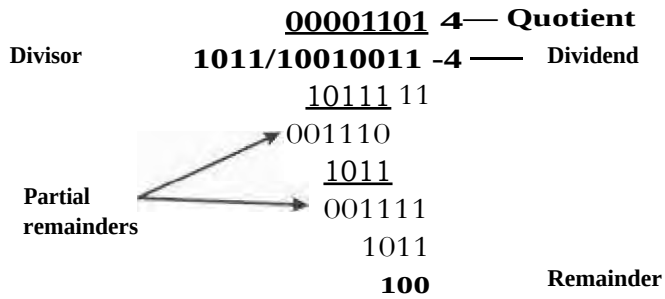


Figure 9.15 Example. **of** Division of Unsigned Binary Integers

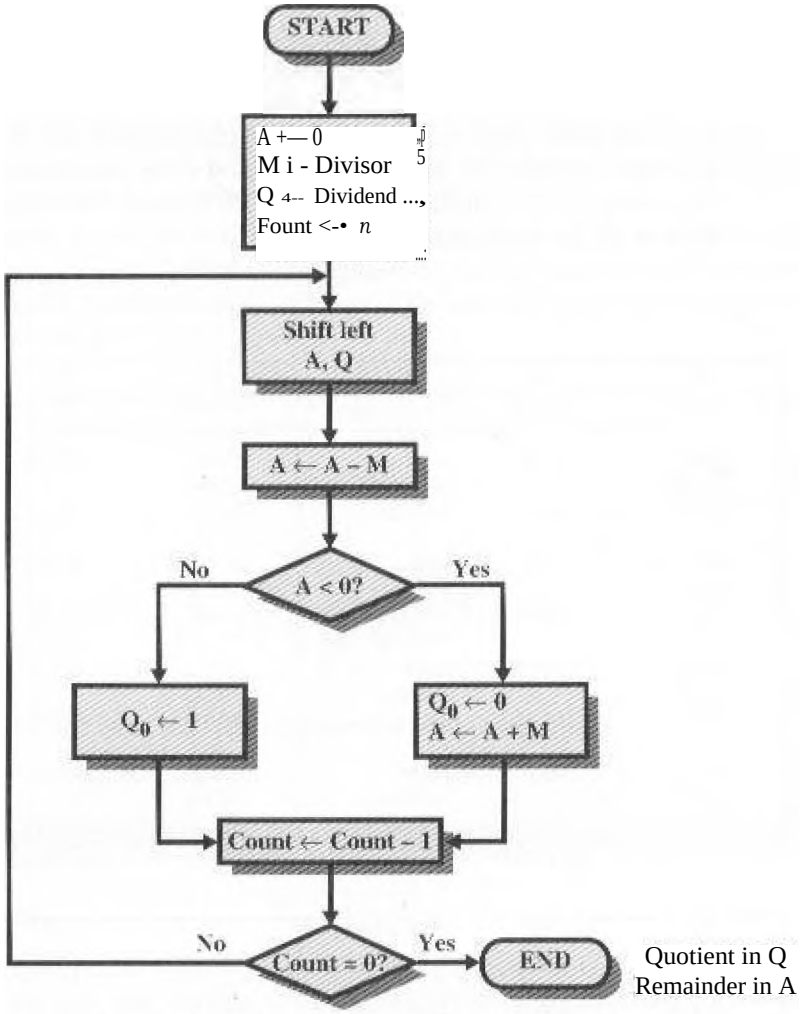


Figure 9.16 Flowchart for Unsigned Binary Divisilin

Figure 9,16 shows a machine algorithm that corresponds to the long division process. The divisor is placed in the M register. the dividend in the register. At each step, the. A and 0 registers together are shifted to the left I M is subtracted from A to determine whether A divides the partial remainder.' If it does, then gets a 1 hit. Otherwise, (...)„ gets a 0 bit and M must be added back to A to restore the previous value. The count is then decrernented. and the process continues for it steps. At the end, the quotient is in the register and the remainder is in the A register,

'This is subtraction of unsigned integers. A result I hat requires a borrow out of the most signilicant hit is a negative result.

This process can, with some difficulty, be extended to nepiive numbers. Vi41 give here out approach for twos core plement numbers. Several examples of Up approku:th are shown in Figure 9.17. 'The algorithm can be summarized as folEo95:'

1. Load lhe divisor into the M register and the dividend into the A. Q registers: The. dividend mist expressed as a 2n-his iwos complement number. Thui, ror example, the 4-bit 0111 becomes 000001E1, and 1.001 becomes 11111001.
2. Shift A, Q left 1 bit position.

A	Q	M=1)011	A	Q	M=110]
02 2	:)1.1:	initial value	00120	0111	1ri al vuluc
0000	1110	shift	2000	1110	hi ll
L1 21		subiract	1101		add
co D]	L1=0	restore.	0000	1112.	rolore
0:01	110:	shift	00..01	1100	shift
1110		mibtraLl	1 1I0		add
0001	1100	restore.	2001	11 c.. (':	restore
	102.0	4hi I.	2.01:	1202	shift
000.1:.		subtract	0700		
0206	100:	awl Q ..		1001	lieLQ) =1.
0001	0210	shift	.J 0::.1	'..: 0 1 Ci'	shill
:110		su kiwi	.110		add
0 001	;3 0:0	restore	000:	0 :1..':	res lore
	(a)17 VP)			OM 1711{ —3)	

A	Q	M=001L	A		
11=1	10:1	Initial value	:111	100:.	Initial value
111=	2010	.shift	1L11	0212i	shift
0: 1 .:.		add	001t7		810131 MCI
1111	0 2.1 r..'	rc.sr.orc	11:1	00=0	.N.store
11L0	0100	Nhi	1,	2100	shift
20 C.1		add	CiC 0:		subtraci
1112	:14 0	restore	1112.	C.	re.strare
1 1 0C'	1 J0	shift.	11:0	1.000	shift
1111		add	1 1:i		subtract
11:1	1001	set Q_0 = 1	L111	=0 01	seEQ.--1
111:	2010	shill	111:	0:1::	shill
0012		add	0212		solltract
1111	021:	re.store	1111	0010	restore
	{cu (_ 7]+'(3)			d} (-7)4-1)	

Figure 9.17 Examples of Twos Complement Division

3. if M and A have the same Sips. perform A A rsv1; otherwise, A A 1 M.
4. The preceding operation is successful if the sign of A is the same before and oparmion,
   a. If the operation is successful or.A = Cl, then set Q,, I.
   b. If Ihe operation is unsuccessful and A # 0, then set 0,, (— 0 and restore the previous value of A.
5. Repeat steps 2 ihrough 4 as many times as there are. hit positions in Q.
6. The remainder is in A. If the signs of the divisor and dividend were !he same, then the quotient is in otherwise. the correct quotient is the twos compiemen1 of Q.

The reader will note from Figure 9.17 Lhal (– 7) and (¯ 3) produce different remainders. This is because the remaindei i.w defined. by

$$D = \text{extfl}\ R$$

where

D = *dividend*
Q = quotient
V = divisor
R *runain.dur*

The rcmilts of Figure 9.17 are consistent with this formula.

s:

9.4 FLOATING-POINT REPRESENTATION

Principles

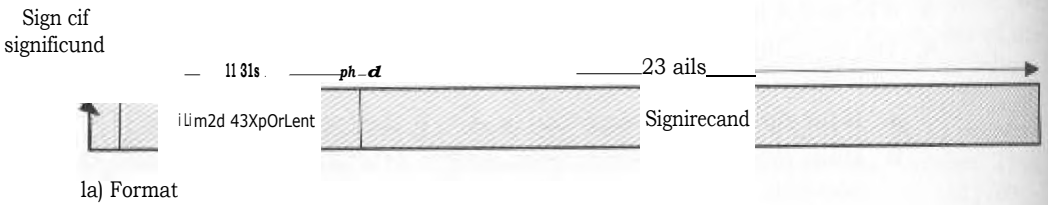
With a fixed-point notation (e.g., twos complement) it is possible Lo represent a range of positive and negative integers centered on 0. By assuming a fixed binary or radix pain ℃, 1h is formal allows the representation of numbers with a fractional component as welt.

This approach has limitations. Very large numbers cannot he represented, nor cat vin' small fractions. Furthermore. the fractional pan of the quotient in a division of two large numbers could he lust,

For decimal numbers, one gets around this limitation icy using scientific notation. Thus. 976.000,000,000,000 can be represented as 9.Th $\mathbf{10}^{14}$, and 0.0000tIi0000000976 can be represented as 9.76 10 What we have done. in effect. is dynamically to slide the decimal point Lo ri c.onvenien1 location Li nd usc the exponent of E0 to keep track of that decimal point. This allows a range of very large and very small numbers to be represented with only a few digits.

This same approCh can be taken with binary numbers, We can represent a number in the form

$$= S\ x$$



la) Format

	2 1c- 3	a 2	10017,01•_	1..7 10C0.1.COM)0200:..00C.02;7.0		1.C38L25	x 222
1.1D10.7,01	N 2 1-1 ''	= 1	100100:1		=	-1.639125	x
1 .Loicoo=	x 2 100	= 0	01:•12 0.	10:0.7.17.10.COON•JC.0200E.00C.	=	..635125	x 2 -2?
-1,1O1OK1	x 2 -111.3171	1	0 1 101011	12 10:;01:..:00CCPDV)::00::00 L C	=	-1.636125	x

1b) Hun ipIts

Figurc. tkig Typical 32-13it Floating-Point hon tat

'['his number can be stored in a binary word with three fields:

- Sign: plus or minus
- Significand S
- Exponent E

The base B is implicit and need not be stored because it **is** the same for all nurribeis. Typically, it is assumed that the radix point is to the right of the leftmost, or most significan1, hit of the signifieand. That is, there is one hit to the left of the radix point.

The: principles used in representing binary floating-point numbers are bes'. explained with an example.. Figure. 9,I 8a shows a typical 32-bit floating-point fur-mat. The leftmost hit stores ihe sign Or the number (0 — positive, 1 = negative). The exponent value is stored in the next 8 bits. The representation used is known as biased representation, A fixed value, called the bias, is subtracted from the field gel the true exponent value, Typically. the bias equals (2' 1 — 1), where *k* is the. number of bits in the binary exponent, in this case. the $.-bit field yid& the num• hers 0 thrortEh 2:5!5, With a is of 127, [he **Lillie** exponent values rare in Lk range —127 to + 128. **in** this example, the base is assumed to be 2.

Table 9.2 shows the biased representation for 4-bit integers. Note that whert: the bits of a biased representation arc treated as unsigned integers.lhe relative mak °nudes of the number!) do nor change. For example, in both biased and unsigned representations. the Largest number is 1111 and the smallest number is (1000. This ix not true of sign-magnitude= or twos **complement** representation, An advantage of biased represeniation is that nonnegative floating-point numbers eon be treated 4 integers for comparison intrpoKcc.

The final porlitin of the word (21 hits in this case) is the significan **d,** also [A]led the mantissa.

Any floating-point number can be expressed in **many** ways,

The fo[lowing are equivatent.where the significand isc:xpressed in binary' form.:

0.110 c 2
1.10 X $2^{=}$
0,0110 x

'Fo simplify operations on floating-point numbers. it is iypica I Iy required that they be normalized. A normalized number is one in which the most significant digit of the significand is nonzero. For base 2 representation, a normalized number is therefore one in which the most significant bit of the significant! is one. As was mentioned. the typical convention is that there is one bit to the kit of the radix point. Thus, a normalized nonzero number is one in the form

$$\pm \text{I .bbh} \qquad \text{b X 2-LE}$$

where b is either binary digit (0 or **l).** Because the most significant hit is always one, it is unnecessary to store this hit; rather, it is implicit Thus. the 23-bit field is used to store a 24-bit significand with a value in kite half open interval 11, 2). Given a number that is not normalized, the number may 1w normalized by shifting the radix point to the right of the leftmost bit and adjusting the exponent accordingly.

Figure 9,1XiD gives some examples of numbers stored in this format Note the following features

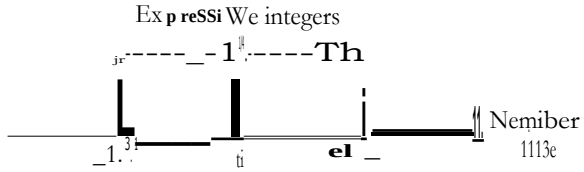
* The sign is stored in the first bit of the word,
- The first hit of the true significand is always 1 and need not be stored in the signilluind field.
- The value 127 is added to the true exponent to be stored in the exponent field.
- The base is 2.

With this representation, Figure Q19 indicates the range of numbers [hal can be represented in a 32-hit word. tising twos complement integer representation, all of the integers from -2^{31} to $2^{11} - 1$ can be represented, for a total of 2^{7} different numbers. With the example floating-point format of Figure 9.1S, the following ranges of numbers are possible:

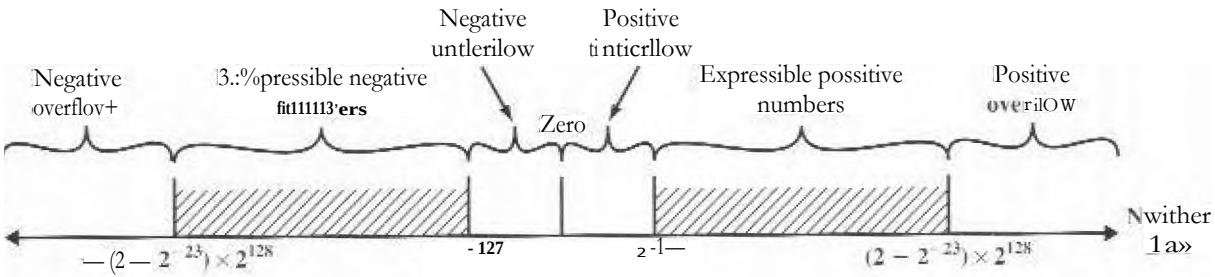
- Negative numbers between • (^{7} - 2 $^{.2}$?) X 2^{12A} and -2^{-127}
- Positive numbers between $2^{\cdot}$ and (^{7} - 2 ') X 2^{28}

Five regions on the number line are not included in these ranges:

* Negative numbers less than (2 - 2^{23}) X 2^{12}', called **negative overflow**
- Negaiive numbers greater than 2 • ', **called negative underflow**
- Zero
- Positive numbers less than 2^{7}. **called positive uWdetilow**
- Positive numbers greater **than (2** 2 ^{23}) x 2^{12}", called positive **overflow**



ia I Twos maple M ent integers



(13.1 Posting-point cumbers

Rpm 9.19EN]) ressi Mc. Num hers in T @x132 r rin ats

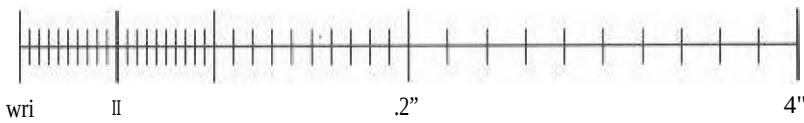


Figure 9.241 Density c.).1 I 1.5;:iling-Poini Numbers

'[hi representation as presented will not accommodate a value of 0. However, as we shall see, actual tloatinE-point represern al ions include a special bit pattern 10 designate zero. Overflow occurs when an arithmetic operation result in a magnitude greater Ilian can be expressed with an exponent of 128 (e.g., $2^{-2n} \times 2^{m} = 2^{:2n}$). UnLicrilow occurs when the fractional magnitude is. Loo small (e.g., 2" ""). Underflow is a less serious problem because the result can generally be satisfai:;-torily approximaled by

It is important to note that we are not representing more individual values with floating-point notation. The maximum number of different values that can he represented with 32. bits is still 2^{32}. What we have clone is to spread I.hose numbers out in two ranyxs, one positive and one negative.

Alsi.), note that the numbers represenled in floating-point notation are noL spaced evenly along the number line, as arc fixed-point numbers, The possible values get closer together near the origin and farther apart ass~ you move ;.iway, as shown in Figure 9.20. This is one of the trade-offs of noai ing-point math: Many calculations' produce results that are not exact and have to be rounded to the nearesi value that the notation can represent.

In the type of format depicted in Figure 9.18, there is a trade-off between range and precision. The example shows 8 bits devoted to the exponent and 23 to the significand. If we increase the number of bits in the exponent, we expand the range of expressible numbers. But because only a fixed number of different values can be expressed, we have reduced the density of those numbers and therefore the precision. The only way to increase both range and precision is lo use more Hts. Thus. mos1 compilers offer, at least, single-precision numbers and double-precision numbers. For example. a single-precision formal might be 32 bits, and a double-precision format 64 bits.

So there is a trade-off between the number of bits in the exponent and the number of bits in the significand. But it is even more complicated than that. The implied base of the exponent need not be 2. '1 he IBM S/390 architecture, for **exam**-plc, uses a base or 16 rAN DI.:67b I. The format consists of a 7-bil exponent and a 24-bit signific,7111(1,

In Qv: 1131v1 forEnat,

0. 11.01(k001 X 2 — 0.111040001 x 16"

and the exponent is stored to represent 5. rather than 20.

The advantage of using a larger exponent is that a greater range can be achieved for the same number of exponent hits. But remember, we have not increased the number of different values that can be represented. Thus. for a fixed format, a larger exponent base gives a greater range at the expense of less precision_

IEEE Standard for Binary Floating-Point Representation

The most important floating-point representation is defined in IEEE Standard 754 PEEE851. This standard was developed to facilitate the portability of programs from one processor to another and to encourage the development of sophisticated, numerically oriented programs. The standard has been widely adopted and is used on virtually all contemporary processors and arithmetic coprocessors.

The IEEE standard defines both a 32-bit single and a 64-bit double format (Figure 9.21), with g-hit and H -bit exponents, respectively. The implied base is 2. L. addition, the standard defines two extended Formats, single and double, whose exact format is implementation dependent. The extended formats include. additional hits in the exponent (extended range) and in the significand (extended precision). The extended formats arc to be used for intermediate calculations. With their greater precision, the extended formats lessen the chance of a final result that has been contaminated by excessive roundoff error; with their greater range, they also lessen the chance of an intermediate overflow aborting a computation whose result would have been representable in a basic format. An additional motivation for the single extended formal is that it affords some of the benefits of a double format without incurring the time penalty usually associated with higher precision. 'Table 9.3 stunmarizes the characteristics of the four formats.

Not all bit patterns in the IEEE Formats are interpreted in the usual way: instead, some hit patterns are used to represent special values. Table 9,4 indicates the values assigned to various bit patterns..l'he extreme exponent values of all zeros (0) and all ones (2515 in single format, 2047 in double format) define special values. The following classes of numbers are represented:

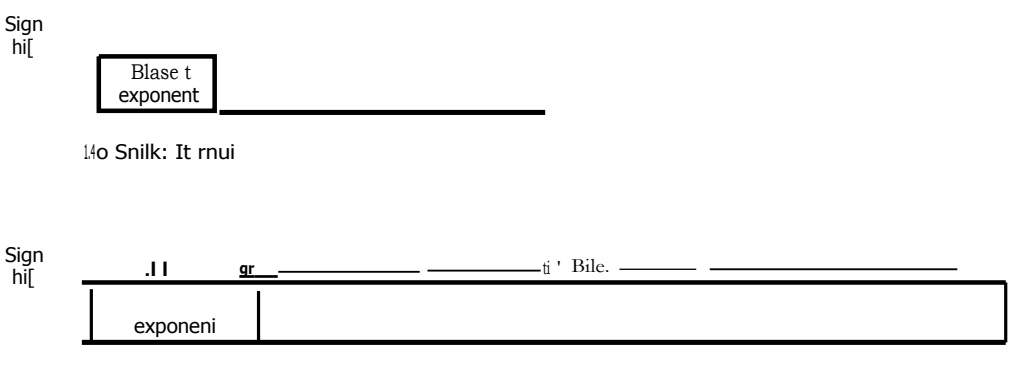


Figure 9.21 IEEE. 754 Fo rmat

Table 9.3 IEEE 754 Forma(['Ammeters

Parameter	ormat Single	Single Extended	Doable	Doable Extended
WORI. WI di Fli 4 !:)i Ls)	32	z-743	e.)-1	:-1.7.'4
Exponent width, (bits)	.\A	7-.1 i	t t	=' 15
Ex ponc n1 bloc	127	1..:Dspe6fiL.d	1023	Unspc..cifi.ed
I'1Q;5ximun exponunt	127	023	1022.	;-' 163.83
Min:EMIR') exponent	-12{-1	- -102..2	-L.02.2	-16382
Number Ent ge (bon 10)	t0 "',. 10' 1 '	Lin p.c:fiucl.	eg. lo•..1-	t.Insiwci.lic.il
Sii.milicand width. (hits)*	'2 1.	€:31	5 7	7- fi.3
7 unthur of eN:polienEs	.2.C4	Lilts pet iri...:d	2346	1.i nspi24:i lied
N1111113.21" (3r fractioas	2.3.1	Urn peci liod	2	Cnspcciricl.1
Numhcr of volui...	l,...l[# x 2."	t :3 0..peci Lied	1.49 x 2 5.	Unspeciikd

bil

- For cAponcrit vaiues in the range oft through 254 kw singie format and 1 through 2046 for double formm, normalized nonzero floating-point numbers arc represented, The exponent is biased, r;o that the range, of exponents is — L26 through -F127 for single torn it and —1022 through 1023. A nortnalized number requires a 1 bit to the left *of the* binary point; this bit is implied, giving an effective or 51-bit significand (called fraction in the st,andni ˙dj.
- An exponent of zero together with ;.1 fraction of zero represents positive or negative. zero, depending 0E1 the sign bit. As was mentio n it is useful to have an exact value of 0 represented.
- An exponent of all ones together with a friAlet ion of zero represents positive or negative infinity, depending on the sign bit. It is also useful to have a representation of infinity. This icinves it up to the user to decide. whether to treat overflow as an error condi I ion or to carry the value 05 and proceed with whatever program is being executed.
- An exponent of zero together with s nonzero fraction represents a (knot ¯mak ized number. In this c,rise, the bit to the left of [he binary ˙ point is zero ;Ind the true expoucnt is —126 or —1022. The.nunther is positive tw negative depending on the sign bit.
- An expon„:iii all ones together with ti nonzero fraction is given the value NaN, which means Not *a Number,* and is used to signal variom exception conditions.

The significance of denorinaliz ˙ed numbers and NaNs is discussed in Section 9,5.

9.5 FLOATING-POINT ARITHMETIC

'll ˙4i Mc 9.3 summarizes the batie operotions for floating-point arithmetic, For additartart subtraction, it is necessary to ensure that biitii operands have the same

'table 9.4 Interpretation of HEEL 754 Floating- **PI' ш**

	Single Precision (32 bit&)				Double Precision (64 bits)			
	Sign	Biased exponent	Fraction	Value	Sign	Biased exponent	Fraction	Value
Positive ZC111	0	C'	0	0	U	0	U	0
Ne20.1.ivc zero	1	0	0	0		0	0	−0
Plus infinity	0	25f. (ail 1	0	∞	0	2047 (a111.si	0	
MiLIU5 il) **ardty**	1.	255 (all Is)	0	∞	1	2047 (41 13)	0	
Ouici NaN	0 or 1	25ff• (all 19	≠ 0	NaN	0 or 1	2047 1211 1 s	≠ 0	'NaN
Signaling NaN	0 or 1	255 i a31 1s)	* 0	Nafq	.0 or 1	7.047 (all 1s)	≠ 0	**NN**
PosiLive m ced normeso	0	0 < c < 755	1	2^{c-127} (1.f)	U	e c < 2047	f	2 $^{c-127}$(.1.0
NeAauve normalized noazero	1	0 < c < **255**	f	2^{-117}(1.1)	1	(I <12 <2047	f	
¦SitiV¦! cicnon naliyd	0	U	f 0	2^{c-125}(0 o	0		f ≠ 0	2r ^{117}(0.11
NeatiVC **&rICiTmNFL4.1**	1	0	f *O	2^{c-126}(0.f)	1	0	f ≠ 0	−2c^{-}''''(01)

orlon nt value. This may 1 cquirr shining ihe radix point *on °nu* of the operands to achieve alignment. Multiplicaiion arid division are na rre straightforward.

A floating-point operation may produce One of these conditions:

- **Exponent overflow: A pogiLivc:** exponent exceeds the maximum possible exponent value. In some systems, this may be designated as I. **or**
- **Exponent underflow:** A negative exponent **is** less than the minimum possible exponent value (e.g. . -200 is less than **-in.** This means that IN number is too small to he represented. and it may be reported as 0,
- **Signifleand underflow:** Iii the process of aligning sio.nificands, digits may flow off the right end of the signiticand. As We shall discuss. some. form of rounding is required.
- **Signifleand overflow:** The addition of two significands of the same sign may result in a carry out of the most significant bit, This can **he** fixcLI rcalign- 4is cxplain_

Addition and Subtraction

In flo.ting-voint arinunctic, addilion ,rind subtraction are more complex than multiplication and division. This is because of the need for alignment. There are four basic phases of the algorithm for addition and subtraction;

1. Check for zeros.
2. Align the 6ignil9cands.
3. Add or subtract the significands.
4. Norrimlize the result-

A typical flowchart isshown in Figure 9-22. A step-by-step narrative highlights [he main functions **[or** Hoaling-point addition and subtraction. We assume a format similar to those of Figure 9.21. For the addition or subtraction operation, the two operands must be transferred to registers that will be used by the Al,(.1 If

Table 9.5 Floating-Point Numbers and ArithnwLicOperatioias

Floating Point Numbers	**Arithmetic Operation..4**
= X. x ..V•	y (x, x Hx: " - x .v - y = / B.'. - x ' X = (X, x X ‹ x

A i ! 1;1' · II.") ¦ O · 11. · 11;
11.2. :1%
X l •:11.3./.,:.2 ¦ x Ill: " _ • 1:1 i.llllll
x :···· ft: 7 ..5:. |

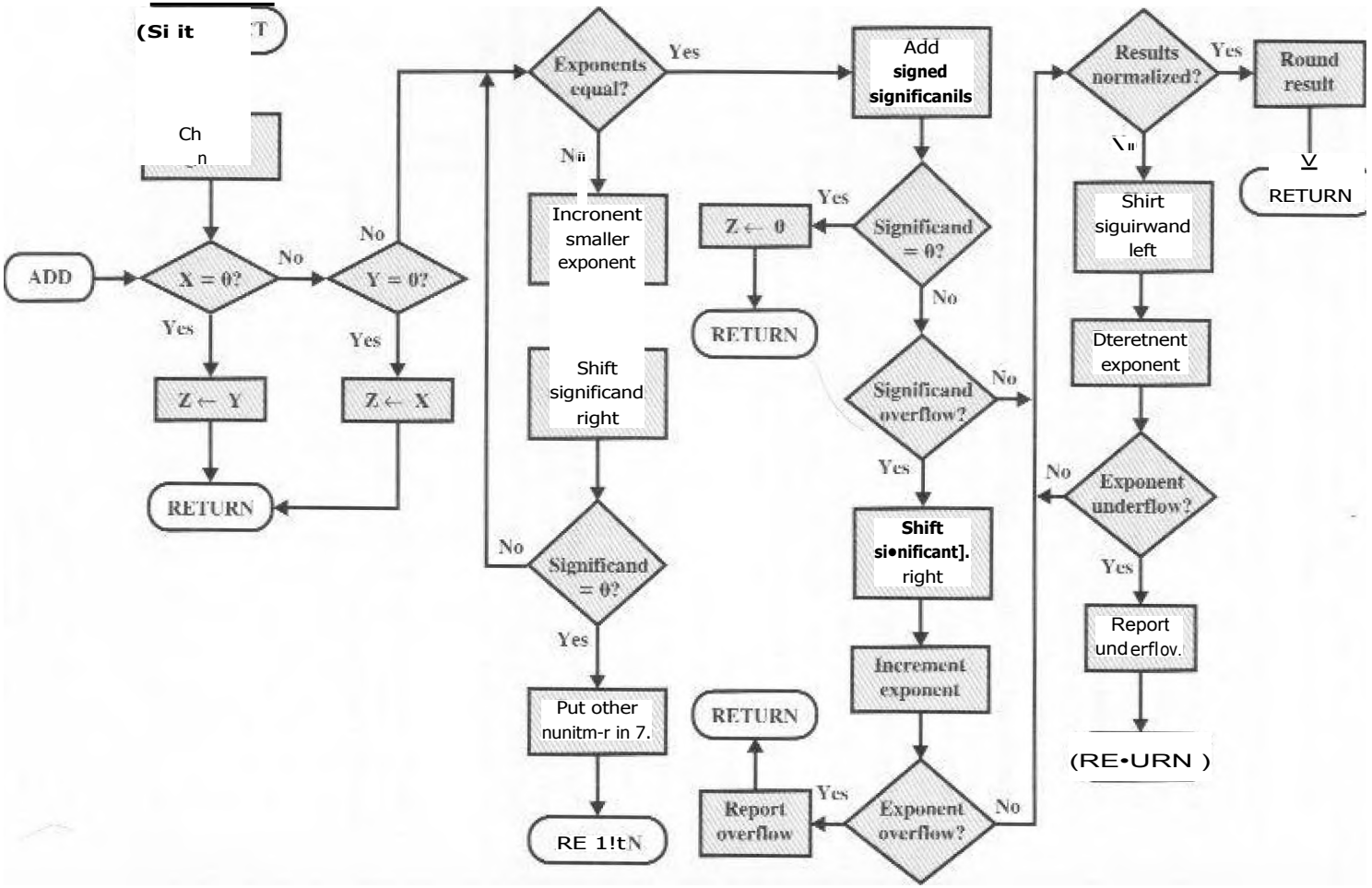


Figure 9.22 Floating-Point Addition anti Subtraction (Z ✕ Vj

the floating-point format includes an implicit significant' hit. that bit must be made explicit for the operation.

Phase 1: Zero check. Because addition and subtraction are identical except for a sign change, the process begins by changing the sign of the subtrahend if it is a subtract operation. Next, if either operand is O. the other is reported as the result.

Phase 2: Significand alignment. fhe.next phase is to manipulate the numbers so that the two exponents are equal.

To see the need for aligning exponents, consider the following decimal addition:

$$123 \times 1(Y^{1}) \text{ -I- } (456 \times 10^{2})$$

Clearly, we cannot just add the significant's. The digits must first he set into equivalent positions, that is. the 4 of the second number must be aligned with the lof the first. Under these conditions, the two exponents wilt be equal, which is the mathematical condition under which rwo numbers in this form can be added. Th us.

(123 x 10") — (456 X 10 (123 X It") (4.56. 0") 127.56 x 10'

Alignment may he achieved by shifting either the smaller number to the right (increasing its exponent) or shifting the larger number to the left. Bccau.sc either operation may result in the loss of digits, it is the smaller number that is shill ed; any digits that arc lost are therefore of relatively small significance. The alignment is achieved by repeatedly shifting the magnitude portion of the significand right t digit and incrementing the exponent until the Iwo exponents are equal. (Note that it the implied base is 16, a shift of 1 digit is a shift of 4 bits.) If this process results in a 0 value for the significand, then the other number is reported as the result. Thus. if two numbers have exponents that differ significantly, the lesser number is lost.

Phase 3: Addition. Next. the two significands are added together. Li king int o account their signs_ Because the signs may differ, the result may be 0. There is also the possibility of significant] overflow by I digit. II' so. the significand of the result is shifted right and the exponent is incremented. An exponent overflow could occur as a result: this would be reported and the operation halted.

Phase 4: Normalization. The final phase normalizes the result. Normalization consists of shifting significand digits left until the most significant digit (bit, or 4 bits for base-16 exponent) is nonzero. Loch shift causes a decrement of the exponent and thus could cause an exponent underfloor_ Finally, the result must be rounded off and then reported. We defer a discussion of rounding until after a discussion of multiplication and division.

Multiplication and Division

Floating-pain t multiplication and division are much simpler processes than addition and subtraction, as the following discussion indicates.

We first consider multiplication, illustrated in Figure 9.23. First. if either operand is (I, 0 is reported as the result. The next step is to add the exponents. It the exponents are stored in biased form. the exponent sum would have doubled

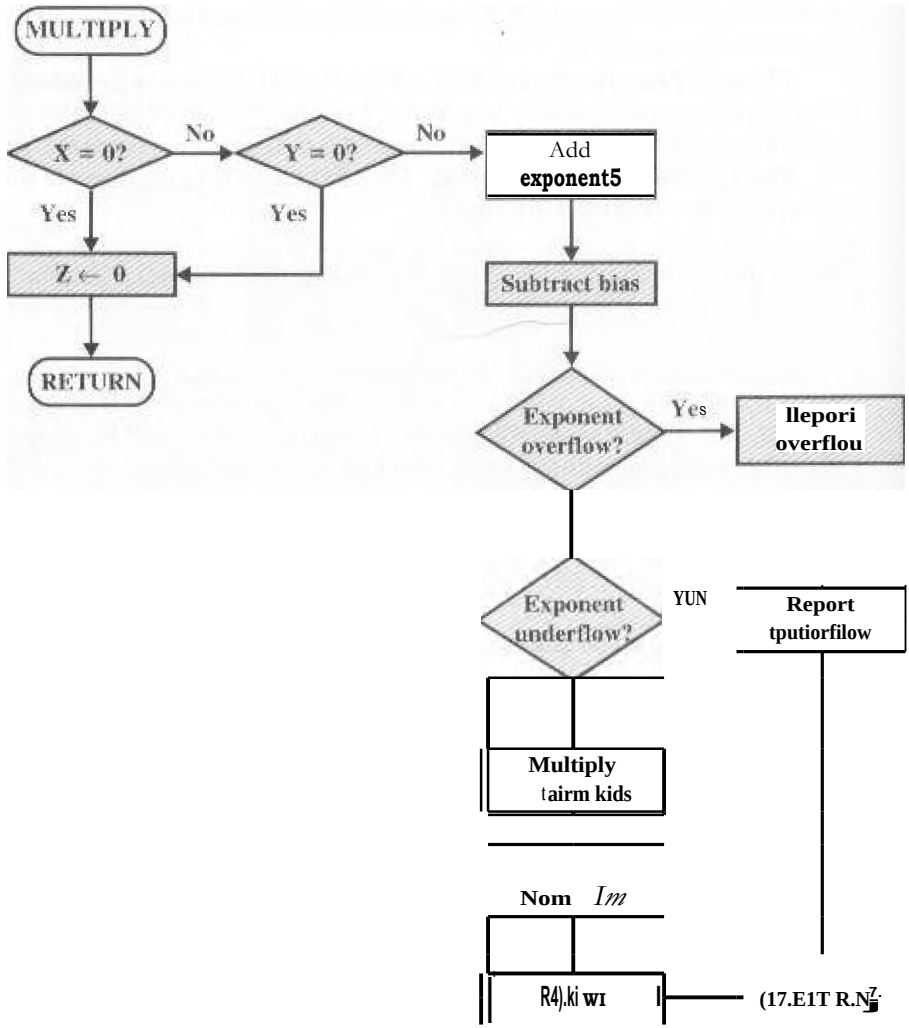


Figure 9.23 Flciating 'Peint Nluldrylicailen (Z 4— x x

the is l'hus, the bias value must liesubtracted from the sum, The result could he either an exponent overflow or underflow. which would bc reported, ending the algorithm.

If rh,: exponent of the product iswithin the proper range, thenext step is to mulliply the significands, la.king into account theft SiQ,J ⊐ Themultiplication is Nr. any in the same way as for integers. In this case, we are dealing.with a sign-magnitude repres.entation, but lite dEtails are similar lo those for twm.complerneal representation, The produet. will !De double the length of the multiplier and multiplicand. The extra bitx will be lost during rounding.

.After the product is calculated, the result is then normalized and rounded, as was done for addition and subtraction. Note that normalization could tcwit in expo-iw,n1 undcrflow.

FiIdly, |a 0s cormicricr rinwehm-1 Eor divihion depicted in Figure 9.24. Again, the first step is testing for 0. 1E the divisor is 0, an error report is issued, or the result is set to infinity, depending on the implementation. A dividend of I) Tenths in O. Next, IIi divisor eNponcni i.s NubLracted Iron, the dividend exponent. This removes the bias, which Tntiz.,1 hu added back in Tests are then made for exponent underfiow or overflow.

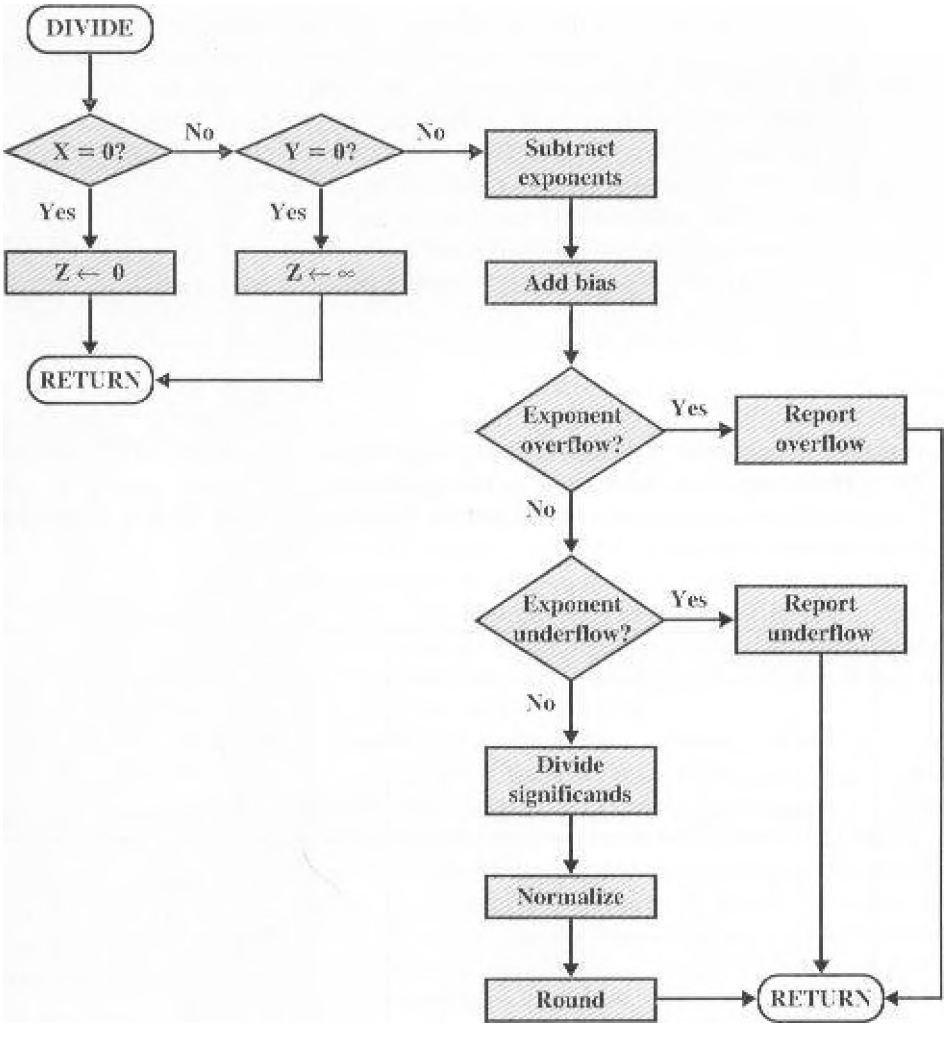


Figure 9.24 Floating-Point Division (Z<- **XIY)**

The next Kier) is to divide the significands. This is followed with the usual nor. ma]ization and rounding.

Precision Considerations

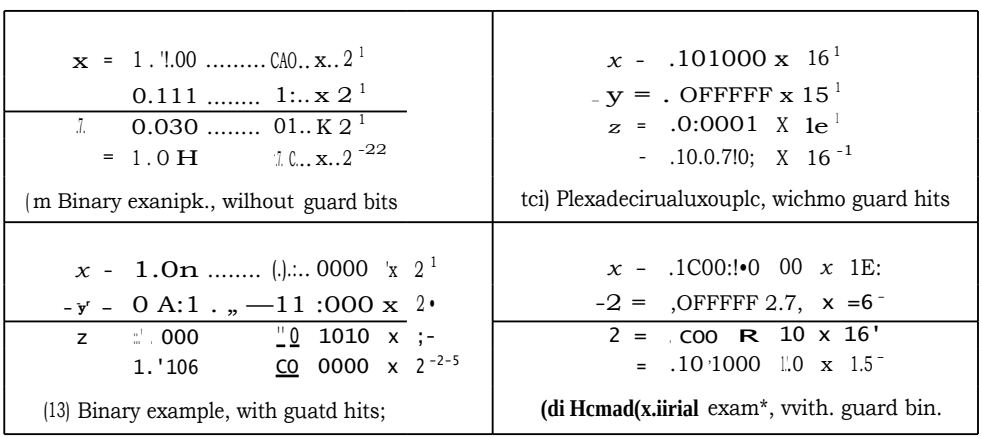
Guard Bits

We mentioned that, prior to a floating-point operation, the exponent and sip,- **nificzind** of each operand are Loaded into AU! registers, In the ease of flit' the length of the register is almost always greater than the length 01 the significand plus an implied register contains additional bits, called guard which are used to pad out the right end of the significand pith

The reason for the use of guard hits is illustrated iEt Figure 9,2.5. Consider numbers in the **IEEE** 10 rnat, which has a 24-bit significand, including an implied I hit Co the left of **Lhe** binary point. 'Iwo numbers **ihat** are ii. m.. close in value. are N , 1,00 . . . 00 X 2) and Y 1,11 . . . 11 X r, If the smaller number is to be suh- iracted from the larger. it must he **shiftcd right i hit wi** align the exponents, This is shown in Figure 9.25.a, In the process, V low..., 1 bit ofsigni [mance.; the re:ii.dt k 2^{-2}, The **same open li011 is rileateci in part 1:i with inc. miclition** of guard bils. Now the least significant hit is not **lost doe to alignment, and the result is 2 ', a** difference or a factor or 2 from the previous answer, When the radix is **Its,** the loss of **precision can he greater. As** l:'i gutes 9.25c and d show, the differtmce can be a factor of 16.

Rounding

Another detail that affects the precision of the result is the rounding policy. The result of any operation on the **signiticands is generally** stored in **a Longer register. When the result** is pui hack into the floating-point format, the extra bits must be disposed of.



kigure 9.25 'I'he USE: of Guard Hits

A number of lechniq LICS have been explored for performing roundinz In fact. the slandard lists four alternative approaches ;

- Round to nearest: The result is rounded to the nearest representable number.
- **Hound tfilikal'd M:** The result is rounded up toward plus infinity.
- Round toward The result is rounded flown toward negative
- hound toward 0: The result is rounded toward zero.

Let us consider each of these policies in turn. **Round to nearest is** the default rounding mode limed in the standard and is defined as follows: The representable value nearest to the infinitely precise result shall be delivered! if the two nearest representable values are equally near, 1he orN12 With its least significant bit 0 shall be delivered.

.....

If the extra bits, beyond the 23 bits that can be steered, are 10010, then the extra bits amount to more than.one-half of the last representable bit posil ion. in this case, the correct answer is to add hinat:;' 1O the last representable **round-**ing. up RP the neut representable number. Now consider that the extra bits are 01111. In this ease ; the extra bits amouni to less than one-half of the last representable bit position. TN'. correct ;.inswer is simply to drop the extra bits (LT uneate), which has the effect of rounding down to the next representable number.

The standard also addresses the :peciai ease of extra bits of the form [WOO Here the resull is exaetly halfway between the two possible representable valiws. One possible technique here would be to always fruneate, as this would be the simpiest operation. However, the difficulty with this simple approach is that it introduces a small but cumulative bias into a sequence of compu [a ions. What is required is an unbiased method of rounding. One possible approach would be. to round up or down on the basis of a random number so that, on average, the result would be unbiased. The argument agai nst his approach is that it does not produce predictable, deterministic results. **rim** melt taken by the IEEE standard is to force the result to be even : If the result of a computation is exactly midway between iwo representabie numbers, the value i5 rounded up if the last representable bit is currently 1 and not rounded up if it is currently O.

The next two options, rounding to plus **mid minus infinity,** are useful in implementing a technique known as interval arithmetic. InitLi-va I arithmetic provides an efficient method for monitoring and eoffiroiling err°, s in floating-point compui tons by producing. two values for each result. The two values correspond to the lower and upper endpoints of an interval that contains the true result,. The width of the interval, which is the difference between the upper and lower endpoints, indicates the accuracy of the result. if the erldpiru' of an interval are not represeniable, then the intemil endpoink are rounded down and up, respectively. Although the width of the interval may '...ary according to implementation. many algorithms have been desNned to produce narrow intervals. if the range between the upper and lower bounds i suffieienily narrow ; then a sufficiently accurate result has been obwined. It' not. at least we know this and can perform additional analysis,

The final technique specified in the standard is **round toward 'LIMP. This** is, ir, fact, simple. irunealion: The extra bits; are ignored. This is certainly the simplest tbt-nique. I iowever, the result is that the niagnitudc of the truncated value is alwaydw than or equal to the more precise original value, introducing a consistent bias tom! zero in the operation. This is a more serious bias than was discussed earlier, becatil this bias affixts every operation for which there are Dormer() extra bits.

IEEE Standard for Binary Floating-Point Arithmetic

IEEE 754 goes beyond the simple definition of a format to lay down specific prat. tices and procedures so that floating-point arithmetic produces uniform, predichibl .. results independent of the hardware platform. One aspect of this has already 1.26n discussed, namely rounding. This subsection looks at three oilier topics: MiNs. and denormalized numbers.

Infinity

Infinity arithmetic is treated as the limiting ease of real arithmetic, with th4 infinity values given the following interprciation:

$$-\infty < (\text{every finite number}) < +\infty$$

exception of the special cases discussed.subsecluently. any aritlun etic operation involving infinity yields the obvious result-

Fur example,

5 + (--.K)		.4. w.	5 (-c..3)	=	+0
5 -- (..rte;)	=		(±x) (--)		.)--^
5 -F (-3.2)	=	-x.	(–') – (..)	▾	-,x
3 – (–x)	=	+x	('- ') – (–')	▾	-Q.:.
5 X (-i-•;)		•F.:',c	(f ⁰ᵘ) — (—m)		-I-rk

Quiet and Signaling NaNs

A NaN is a **symbolic** entity encoded in floating-point formal. of which dim. arc two lypcs: ".i gnaling and quiet. A signaling NaN signals an invalid operation cxcepticm whenever it appears as an operand. Signaling NaNs afford values for uninitialized variables **and** arithmetic-like enhancements that are nul the subject of the standard, A quiet NaN propagates through almost every arithmetic operation without signaling an exception. Table 9.6 indicates operations that will pro. duce **a quiet** NaN.

Note that both types of NaNs have the same genera] formal (Table 9.4): an exponent of all tines and a nonzero fraction. The actual hit Rattern of the nonzero fraction is implementation dependent: the fraction values can be used to distinguish quiet NaNs from signaling NaNs and to specify **particular** exception conditium.

Denormalized Numbers

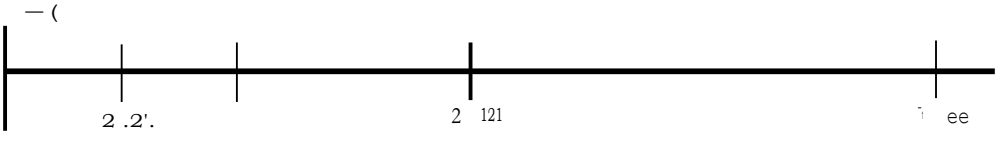
Denorrnalized numbers are **included** in **TEF:h** 754 to **handle** cases of exponent underflow. When the exponent of the result **Faccornes too small** {a negative. evo-

Table 9.6 Operationblhal Prodoce a Quiet NaN

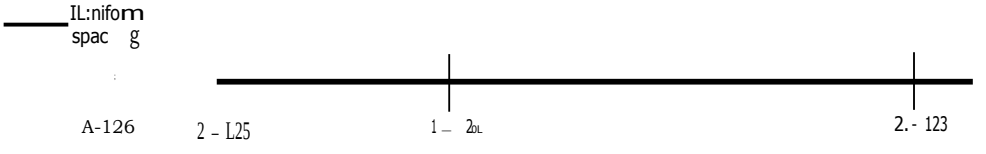
Operation	()Hid NAB **Produced by**
Any	An.....orie.ranon on a ....;;ignaling NaN
Add or ....;•Liblrod.	Magnitude subtraction of infinities: $(+\infty) + (-\infty)$ $(-\infty) + (+\infty)$ $(+\infty) - (+\infty)$ $(-\infty) - (-\infty)$
Multiply	$0 \times \infty$
Division	$\frac{0}{0}$ or $\frac{\infty}{\infty}$
Kernaindcr	*x* REM 0 or REM y
Square 1001	wh ---- I)

ni,:n1 with Re kirge a magnitude), the result is denormalized by right shifting the fraction and inc ; ernenting the exponent tor nch !Ihifl, until the exponent k wilhin a representable range.

Figure 916 illustnics the erfuet c)1 thu addition ofd northalized nurnbevi.. The TuprmIntable numbers can he grouped irito inten als of the form 1.2!', 2'1. Within each such interval, the exponent portion of 1 nurnber remains constant while the fraction varies, producing zi uniform spncing of representable. numbers; within



32-13;•it icFrinat » LESOUL deaorinalized mambas



• .; formt wish (k.normalizccInuniber...;

Figure 9.26 The Effect of **I EEE** 754 Denorrnalized Numbels

interval. As we get closer to zero, each successive interval is half the width of the. preceding interval but contains the same number of representable numbers. Hence the dunsity aJr reraresuniable numbers increases as we approach nett). liowevur. if only normalized numbers are used, shire is a gap hetwccn the smallest nornializd number and f). In the case of the 32-bit IEEE 754 format, there are representnhk numbers in each interval, and the smallesl representable positive number is With the Adition of dcnorrna in Hdditional 2^{22} numbers unifotmly added between. 0 and 2 - l

The use of denornialized numbers is referred to as *gradual underflow [COONO. Without* denormaiized numbers, i he gap between I he snialles1 representable nonzero number arid zero is rhua wider than the gap between the smollest representable nonzero number and the next larger number. Gradual underflow fills in lhal, gap and reduces the impact of exponent underflow to a level comparable with roundolf among the normalized numbers.

9.6 RECOMMENDED READING AND WEB SITES

[PARHOOf is an excelicnt ireatment of computer ari1111111 **giC,** covering all of the topics in this chapter in detail. 'FINN:it I is a useful discussion thnt focuses um practical design ;Ind mentatinn issues- For the serious student ofuomputer arithmetic, a very useful reference is the two-volume. I SWA R901. Volume I MIS originally. published in I 9R0 and provider- key papers (some very difficult to obtairi fltherwis.)4,11 wriputer arithmetic iundanrctak. Li me f I contains more recent papers. covering theoretical, design, and implementation aspects of computer arithmetic.

For floating-point arithmetic. riOLD91 1 is well named; "What Every Computer Scr enlist Should Know About Floating-Point Arithmetic." Another excellent treatment of the topic is CCPnlakined in KNUT981, which also covers integer computer arithmetic. The following more in-depth treatments arc also worthwhile; [OVER01. EVENDO, OBER97u, OBee.R`)71i. SOD[96].

1:5.1.11 '*,141describts the first IBM Si'39ltirrucemor too integrale radix-16 and IEEE 754 arithmetic in the same floating-point unii.

EVEN00 and. '011 ot I complinnt Flouiing•Puiiit Units." 1 *1=Lfs S rrN (}11 COehlphkrigOW,* May 2000.

FLYNN Flynn, M. and Oberman, S. A iliviayri *Derip.* New York; Wiley. 2001.

GOLD91 Goldberg, D. "What Every Computer Scientist Should Knew Abaut Flitial tug-Point Arithmetic." *ACM Cop.,puithg Surveys.* March 1.(191, do..ailutall; p;.:Nyxkx4,validgh.corn.

IKKU198 Kutti It, I.). *The Arr einem Prograrorming,* Voiron *re 2: Srminzertwrii fei* Rinclim;, MA; Addison-Wt.-ley, 199S,

1H3ER974 Oberman. S and l'Iyun, M. Design Issile3 Di 0ion acid Other Floating-Point Operations.' *farm Er CO,refingrri, I* **eI itch** y

OBER97b Oberman, aodFlyrin. \r-l. "Division Algirritlinis and 1mplcinenta **Lions."** *IEEE Transacsioni Compuree.v,* 19[illegible].

C1 `E Ovizmpu, M. *Alf oenprkiri CeJhriArliej c With 1E1; 6 NurilioR P49b11 ArithfnCii* Philadelphtk PA: Sociu tv for and Appkod Mathe.rilatics,:lrm1.

PARI100 Parl runi 5 B. *Comp,. Arilirnrcric.: r via?en:, nti a rdiwim* , Oxford! Oxford University Press, 20(X);

SCHWO9 Schwarz, E., and Krygowski, C. "The ,fiThi foriir- wd f *Rcsearch err DeL:4oprnem,* Septc.mbizr.iNcricrinheT 1999. (www.)

SODE.% sod,2r-quisi, *rind* Leeser. M. "Area 4111d Performanez Tradeoffs in Floativ Point Divide and Square-Root Irripkinentations." *ACM Computing Sti rye ys.* September 1996.

SWAR91) Swartzlande.r. E.., ed. *IC (iv- rirr4rrrLC lir. 1/rohen,r.s I (Mid 11..1..os* Alamitos. CA; IF.F.F, Computer Saciery Press:199G.

RCCOMITiell<led Web Si k..;

- IEEE *754:* The IEEE 754 documents. related pLiblications and papers. and a useful sot of links rekited computer arithmerie

9.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key 'Terms

Hritlintetic and logic unit (ALU)	guard bits	procliJci
arithmelie	minuend	
ha5e.	multiplicand	rmlix poi!
biased representation		rel II *ai* del"
denormalized number	negative. overflow	WV] ıding
dividend	negative underflow	sign bit
divisor	nortnalia4 number	si gni eicand
.Exponent	ones Ooiiipl4;11ii ill	signifirand i.PVC!rflow
ci.poners 0A:l2 0km	representation	significant 4fE1 dallow
xpoincnt underrlow		sign-magnitude TT..presentation
fixed-point rep reSC ntaii011	partial. prodocit	Si1lstr2abe1]cl
ra;prws4I11 .1tii1rt	posiriva overflow	twos ctniplc'.mcnl representation

Review Questions

9,1 Briefly explain. the following reptesait sign-rag ivi..os complement. biased.

41.2 Explain how lo determine if a number is Hui ralowing representations.: sign-rnquitukile.14tr.os uomplorunt.,

93 what is Itie sign-extensil rifle cur twos !:outpE'.inent numbers'?

9.4 I low L.iin you form the negation or an iriler:1 iii two.; colTLplkfient Npreseniation'?

In geoural lcrnik, when docis the twos compleirte at operation on an B-bit integer pro.. duet; ;III....Q.-C.'

9.6 What i8 1lie difkiE i*o..'5corn **pitmen(** representation of a numher and the. twos complemi..E1[

9.7 IC we. great 2 twos 00mph:int.:Ail numbers as unsigned integers 143r purposes of addition, the result is correct if interprels.A.1 as a twos complement number. This is nol true for nrlti1 iplication- Why?

'Whal arc the four essential elements of a number in floating-point notation?

9.9 What is the benc.fil *of* using biased represent al ion for the exponent portion of a flouting-point number?

9.10 What arc the differences among positive overflow, exponent overflow, and signih-cund overflow?

9.11 Whai are the basic eluments of Flom ing-point addition .l nd subtraction'?

9.12 C.3.iva a reason for the use of guard bits.

9.13 List four alternative methods of rounding Itn: re4tilt of a floating-point operation.

Problems

9.1 Another representation of integers lhat is sometimes encouni is **ones com• plcmwnr. Posiiiw** integers arc represcriled in the some wav **tide. A lug-** alive integer is represented by taking the 'Boolean complenion ilt k:;Icli hit of she; corresponding positive number.

a, provide a definition of **Ones** coinpiernent numbers u5fialg a Wi2ighted sum Of hill.. similar to Equations (9.1) and (9.2)-

b. What is the range of numbers that call be represented in ones vomplement?

c. Define an algorithm for performing addition in ones complement aril

9.2 Add columns 10 •Fable 9.1 for sign magnitude and ones eompleineill.

9.3 Considt:r ;lie following operation on a binary word. Start with the least signifieant. Copy all bits ihat are until the first hit is reached and cops: thrill [iii. too. Then IA: the complement of each bit therctifler. What is the result'?

9.4 l.ll SQction 9.3, the tWON4..oitiplernent operation is defined as follows. To find the tiwos complement of *K.* take Ow Boolean complement of each hit of X. and Olen add 1.

a. Show that the folli ming is an equivalent definit ion. For an n-hit integer X. the torus complement of k rimmed by treating *X* as an unsigned integer and ealculatin :: (2n - Xi.

b.. D4.2nsonstrate..1h.nt Figure 9.2 can be used lir support graphically the claim in pan a. by showing how a clockwise movernent is used to achieve subtraction.

9.5 Find the following differences using ivvos complement arithmetic!

&-	b.:11021_0D	c. 1111:.0C.O.L_11	**d.** 11 Ou011
·	**12.1110**	**-1 12201.1110::11**	**-11:01000**

9.6 Is the following a Valid alternative definition of overflow in twos complement arithmetic?

> **If** the exclusive-OR of the carry bits into and oul of the leftmost column is 1, then there is an 41',er11ow' condition. Otherwise, there is not.

9.7 Compare Figures 9.9 and 9.12.Vali. is the bit not used in the latter?

9.8 Given ; - ti 101 and v - 1010 in 14.k osi ornplement notation (Le_ x 4, v _ _ 6), env putt. the product $p - x$ Xy with Booill's algorithm.

9.9 prrive that the multiplication of two n-digit numbers in base B gives a product of no more than 2n digits.

9.10 Verify the validity of the unsigned binary division algorithm of Figure 9.16 by showing the steps involved in calculating the division depicted in Figure 9.15. Us: presentalion similar to alai of Figure.

9.11 The twos complelne.itt integer division algorithirt described in Section 9.3 is kno*rt as the restoring *method* because the value in the A register must be restored fa-

lowing unsuccessful subtraction. A slightly more complex approach. known as no n-restoring. avoids the unnecessary subtraction and addition. Propose an algorithm for 1his latter approach.

9.12 Under computer integer arithmetic, the quotient *.11K of two integers and K* is less than or equal to the usual quotient. True or false'?

9.13 Divide -145 [ix 13 in binary twos c ornpientent riotalion, using 12-hit words. Use the. algorithm described in Sect iiIn

9.14 Assume that ihc c.pi went c is constrainod tci lie: in the range 0 s *e* X. with a bias or I.?, rhat thv base is *b.* and thai 1.1w sign ilicarld is *p* kligits

a. What are the largest and smallesi positive value.s that can be written'

h. What are the largest and smallest positive values that can he written as normaLed floating-point numbers'?

9.15 Express the following numbers in 1 FEE .32-bit Floating-poini format:

a, -5 ie.. 1)16

h, -6 **d.** 384 **f. –**

9.16 Express the Ibllowing numbers in IBM's 32.bit floating-point format, which uses u7 bit exponent with an implied base of 16:

111.. 1.0	**i.** 1164	e. -15,0	**g. 7.2 X L C**
h. 0.5	**d. 0.0**	F. 5.4 X 1:0	

9.17 '1'4' tat would he the bias value for

a. A base 2 exponent (R. - 2) in a 6.bi1 field?

h. A 1-sale- t exponent (11 - 8) in a 7-bit field?

9,1H Draw a Ii irlF tkr hole io that in Figure 9.19k for the. float in;g-point format of 9..7.111.

9,19 Consider a floating-point formai with 8 hits for **die.** biased 1...,x.porkini and 2:1 hits for signilicand. tibinw the hit pattern for the following numbers in Ihis format:

a. 720

h. 0.645

9.20 V'hen people speak about inaccuracy in ihmtirig-point arithmetic. they often ascribe errors to cancellation that occurs during the subtraction of nearly equal quantities. Rut when X and Y are approximately equal. I he difference $X - Y$ is obtained exactly. with no error. What do these people really mean'?

9.21 Any ftonling-poii;1 represent@ lion used in ...sent only certain real nuinhers ;I:sac-ilk..., all oille.rs **rnml** appi4ixiimiteal. the stored value approximating the real value *.4,11scn* he !illative error. 0., is eNpi I as

/1 A'

A

Represent the decimal quantity I 0.4 in the following floating-point format: base - 2; exponent.: biased, 4 bits; significant]. **7** bits. What is the relative error?

9.22 Numerical kt**dus** A *and Beare* stored **in** the computer as approximations *A' raid* N4Jglecting any further truncation or. rOundoff errors, show that the relative error of **the** product **is** approximately t**he sum of [Eh'. TdatiV4. struts in** the laciors.

9,23 If A = 1.427, find di relative error if A Mrtincatod io 1.42 and if it is rounded to t.43-

9.24 One if the most serious errors in computer calculations occurs when two nearly equal numbers are subtracted. Consider A 0.222M and 0.22211. **The** cilinputer truncaLes all values to four decimal digits. Thus A: - 0.2228 *and 8'* - 0.2221.

Wi nd are the rolalive errors kir A' and

h. Vaal is the relaliye error for *C = - fr?*

9.15 Show how the folli r..ving floaiiim point additions are performed [where signifizaA are truncated L0 4 decimal digits)'.

a. 0.5566 X 111' .1- 0.7777 x 10' b. 03344 — 0.8877 x]0

9.21+ Show hoW the following floAtingrpoirit stibtractions to performed (where significa are truncmed to 4 decimal digits).

a. 0.7144 10 = — 11.(.60 X 10 h. 0.8844 X 10 ² **- 0.2233 X** LAY'

9.27 Show how the **10110W** irig 11 (tting-point calculations arc POr(ormc (".here significatith arc. truncated to 4 decimal digits}_

a. (0.2255 x 10^2) x (0.1234- 10) b. (0.8833 : 10^5)

9.28 Expross the octal numbers in htlxade.cimill notation:

a. 12 11), 5655 e. 25502145 **d.** :1726755

9.29 Prove that %wary r4A number with a terminating binary representation (finite ruun• ber .uf Lligits to the righ1tir the binar:L. point) also hits j terminating decimal reresea• tation (finite number of cligits to the right of the Llueimrdpoiru).

CHAPTER 10

INSTRUCTION SETS: CHARACTERISTICS AND FUNCTIONS

10.1. Machine litstruction Characteristics

10.2 Types of Operands

10.3 Pentium and PowerPC Data Types

10.4 Types of Operations

10,5 Pentium and PowerPC Operation Types

10.6 Assembly Language

10.7 Recommended Reading

10.8 Key Terms, Review Quemions, and Prot,'ems

Appendix 10A Stacks

Appendix 11}11 Little-, Big- and Bi-Endinn

KEY POINTS

- I he c:47;.ciiiial elements of a computer instruction are the opcodc, which spu...-ifies the operation to be performed; the. source kind destination operand references, which specify the input and output locations for the operation: and a next instruction reference. which is usually implicit.
- Op-codes specify operations in one of the following general categories: arithmetic and logic operations: movement of data between two registers, register and memory, or two memory locations: 110; and control.
- Operand references speci t).. a veRister or memory location of operand data. The type of data may be addresses, numbers.: characters. or logical data.
- A common architectural feature in processors is the use of a slack, which may or may not be visible. to the programmer. Stacks are used to manage procedure calls and returns and may be provided as an alternative form of addressing memory. The basic stack operations are PUSH. POP, and operations on the top one or iwo slack local ions. Stacks typically are implemented to grow from higher addresses to lower addresses,
- Processors may he categorized as big-endian, little-endian, or bi-radian. A multibyte numerical value. stored with the most significant byte in the lowest numerical address is stored in big-endian lash i4 in; if it is stored With the most significant byte in the highest numerical address, that is little-endian fashion, A bi-endian processor can handlL both styles.

Lich of what is discussed in this book is riot readily apparent to the user or programmer of a computer. If a programmer is using a high-level language, such as Pascal or Ada, very little of the architecture of the underlying machine is visible,

One boundary where the comput designer and the computer programmer can view the same machine is the machine instruction set. From the designer's point. of view. the machine instruction set provides the functional requirements for the (11:: Implementing the CPU is a task that in large part involves implementing the machine instruction set. From the user's side, the user who chooses to program. in machine language (actually, in assembly language; sec Section 10.6) becomes awire of the register and memory structure, the types of data directly supported by the machine, and the functioning of the AUL

A description of a computer's machine instruction set goes a long way toward explaining the computer's CPU. Accordingly, we focu, on machine instructions it this chapter and the next.

10.1 MACHINE INSTRUCTION CHARACTERISTICS

The operation of the determined by the instructions it executes, referred to as *machine instructions* or *computer insmalions_ The* collection of different instructions that the CPU can execute is referred to as the CPU's *thstractirm set.*

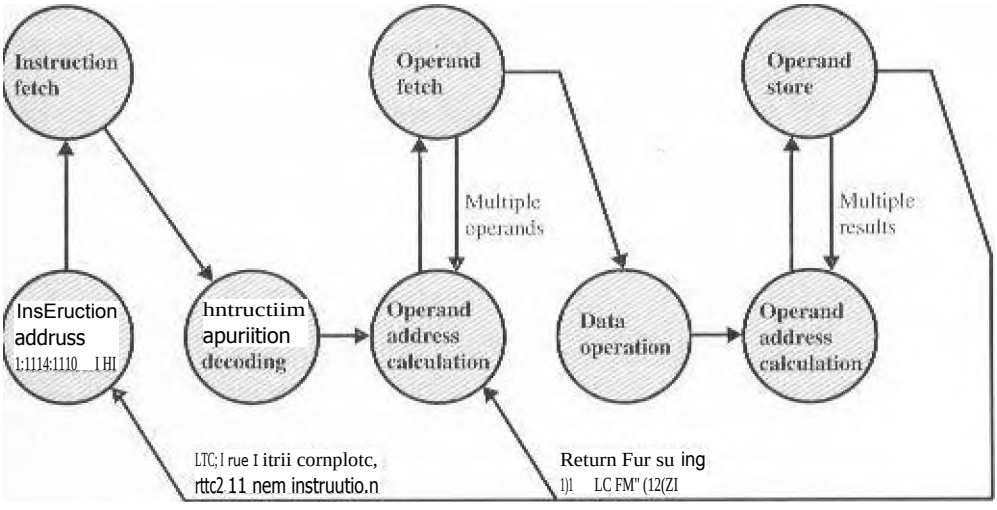


Figure UL Instrildion Cycle. Si nile I)iagrarri

Elements of a Machine Instruction

Each instruction must contain the information required by 1he CPU for execution. Figure 1111, which repeats Figure. 3.6. shows ll h,:2. !%.1.cpm in volved in instruction exeelation and. by implication, Llefines the elements of a machine. instruction. These elernerlt:5. are aix roillows:

- **Operation code:** Specifies the operation to be performed (e.g,, ADD, 110). The operation is specified by a binary code, known as. the operation code or opcode.
- **Source operand reference:** The operation may i vc one or mere source operands. that is, operands that are inputs for 1hs opuratit)11.
- **Result operand reference:** The operation mily produix a result.
- **Next ki4ruction reference:** This tells the CPU where to fetch Ihe ıɪʟ'xı instruction after the execution of this instruedim i cOropleLe.

The next instruction to be fetched is loo:- t(31 in main Memory or, in the case of a virtual memory !.3ys1i,:.m. in either main inemory or secondary memory (Ask), in most casi2s, the next instruction to be fetched immediately f.c.1.1lows the current instruction. In those cases, there is no explicit reference to the next instruction, When an explicit referenc• ix neetki, 1tien the Mill memory or virtual rilentory address mus; ti e foi ıʟ in which that address is supplied is cliseusscd III Chapter 11.

Source and result clperands. can bc in erne or 1 hri2c areas:.

- **kfitin or virtual memory: As** with net instrUCtiOn roferences. the main or virLLW rnetriOrsi address must be supplied.
- **CPU register.:** With rare excei:.ptioris, a CPU contains one or more registers that ɪMIN *be* referenced by machine instructions. If only one register exists, refer-

enee lo it may he iTnplicii. If more than one register exists, then each Tel Lista is assigned a unique number, and the instruction must contain the number of the desired register.

- 110 device: The instruction roust specify the I/0 module and device for the operation. If memory-mapped I/0 is used, this is just another main or virtual memory address.

Instruction Representation

Within the computer, each instruction is represented by a sequence of bits. The instruction is divided into fields, corresponding to the constituent elements of the instruction. A simple example of an instruction format is shown in Figure 10.2. As another example. the lAS instruction format is shown in Figure 2.1 With most instruy-tion sets, more than one format is used. During instruction execution. an instructia is read into ain in:A ruction register (IR) in the. CPU, The CPU must be able to extract [tic data from the various instruction fields in perform the required operation.

it is difficult for boih the prt Fgr am Mel' and the reader of textbooks to deal with binary representations of machine instructions. Thus., it has become common prat• lice Lo use a *.symbolk. represengaiion or machine* instructions, An example or this was used for the 'AS instruction set, in 'I'ablc 2.1.

Opcodes are represented by abbreviations, called *ne n* on/Ls, that indicate the operation, Common examples include.

ADD	Add
SUB	Subtract
Nun'	Multiply
DIV	Divide
LOAD	Load data rrom memory
STOR	Store data to memory

Operands are also represented symbolica/ly. For example. the instruction

AD R,

may mean add the value contained in data location Y to the contents of regisier R, In 1 his example- Y refers to the address of a location in memory, and R Terors to a particular register. Note that the operation is performed on the contents of a loca• lion, not on its address.

'Elms, it is possible to write a machine-language program in symbolic form. Each symbolic opcode has a fixed binary representation. and the programmer spec-

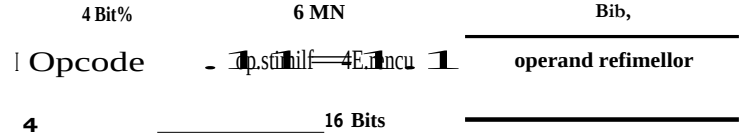


Figure 10,2 A Simple InsLirticl ion Format

ifies the location of each symbolic operand. For example. the programmer might begin with a list of definitions;

X - 513
Y 514

and so on. A simple program would accep11his symbolic input, convert opcodes and operand references to binary form, and construct binary machine instructions.

Machine-language programmers are rare to the point of nonexistence. host programs today are written in a high-level language or, failing that, assembly language, which is disens;.'.ed ;11 the end of this chapter. However, symbolic machine language remains a useful tool for describing machine instructions, and we will use it for that purpose..

Instruction Types

Consider a high-level language instruction that could be expressed in a language such as BASIC or FORTRAN. Fur uxamill4,.

x **I1Y**

This statement instructs the computer to add the value stored in Y to the value stcired in X and pm the result in X. might this be accomplished with machine instructions'? Let us assume that the variables X and Y correspond to locations 513 and 514. If we assume a simple set of machine instructions, this operal ion et iuld be accomplished with three instructions:

1. Load a register With the contents of memory location
2. Add the contents of memory Location 514 to the register.
3. Store the contents of the register in memory !mailer' 51i.

As can be seen. the single BASIC instruction may require three machine instructions- This is typical of I he relationship heiwern a high-level language and a machine ianguage. A high-level language expresses operations in a concise algebraic form, using variables. A machine language expresses operations in a basic form involving the movement of data to or from registers,

With 1h is. si mple example to guide us, *let* us consider the types of instructions that must be included in a practical computer. A computer should have a set of instructions that allows the user to formulate any date processing task. Another way to view it is to consider the capabili tics of a high-level programming language. Any program written in a high-level language must be translated into machine Language to be executed. Thus, the set of machine instructions musA stillieient to express any of the instructions from a high-level language. With this in mind we can categorize nislraelliOn types as fellows;

- Data iiroceming.. Arithmeiic and logic instructions
- Data storage.: !vtemory in!,IrLieLions
- Data movement I/O in.drucLioris
- C'untrol: lest and 13nind'i instructions

Arithmetic instructions provide computational capabilities for processing numeric data, *Logic* (Boolean) instructions operate on the bits of a word as bits rather than as numbers; thus, they provide capabilities for processing, any other type of data the. user may wish to employ. 'These operations are performed primarily on data in CPU registers. Therefore, there must be *memory* instructions for moving data between memory and the registers. *I/O* instructions are needed to transfer programs and data into memory and the results of computations hack out to the user. *Test* instructions are used to test the value of a data word or the status of a computation. *Branch* instructions are then used to branch to a different set of instructions depending on the decision made.

We will examine the various types of instructions in greater detail later in this chapter,

Number of Addresses

One of the traditional realys of describing processor architecture is in terms of the number of addresses contained in each instruction. This dimension has become less significant with the increasing complexity of CPU design. Nevertheless. it is useful at this point to draw and analyze this distinction.

What is the maximum number of addresses one might need in an instruction? Evidently, arithmetic and logic instructions will require the roost operands. Virtually all arithmetic and logic operations are either unary (one operand) or binary (two operands). Thus, we would need a maximum of two addresses to reference operands. The result of an operation must be stored, suggesting a third address- Finally, after completion or an instruction. the next instruction must be fetched, and its address is needed.

This line of reasoning suggests that an instruction could plausibly be required to contain four address references: two operands, one result. and the address of the next instruction. In practice, four-address instructions are extremely rare. Most instructions have one, two, or three operand addresses, with the address of the next instruction being implicit (obtained from the program counter).

Figure .10.3 compares typical one-, two-, and three-address instructions that could he used to compute Y = (A - 1-1) (C + D x E), With three addresses, each instruction specifies two operand locations and a result location. Because we would like to not alter the value of any of the operand locations, a temporary location, T, is used to store some intermediate results. Note that there are lour instructions and that the original expression had five operands.

Three-address instruction formats are not common. because they require a relatively long instruction format to hold the three address references. With two-address instructions, and for binary operations, one address must do double duty as both an operand and a result. Thus, the instruction St .'13 Y, B carries out the calculation Y B and stores the result in Y. The two-address format reduces the space requirement but also introduces some awkwardness. To avoid altering the value of an operand, a MOVE instruction is used to move one of the values to a result or temporary location before performing the operation. Our sample program expands to six instructions.

Simpler yet is the one-address instruction. For this to work, a second address must be implicit. This was common in earlier machines, with the implied address being a CPU register known as the *accumulator,* or AC. The accumulator contains

Instruction	Comment
SUB • A_4 B	Y.— A - B
MPY 1, D. E	T D x E
ADD T,	T•(—T+ C
DIV Y, Y,	Y <

(zE) Thico-;1411.3p.rs.s. ill %till

Instruct iiiiii	.iii ui (I i ela
1-DAD I)	Al.
MPY I	AC E
ADD C	(— + C
TOR Y	Y AC
LOAD .2\	AC: c• A
4E:13 13	AC •• AC . B
DIV V	AC (— AC
STOR Y	Y AC

(c) Orie-addrei...s instmetions

Instruction	Comme nt
MOVE V. A	Y <•-• A
SUB Y. B	Y Y' B
MOVE T. D	T •(— D
MPY' T. E	T•(—TxE
ADD T. C	T T C
DIV Y, '[	Y -F T

lb) i Ii!.;

Figure 111.3 Progrnms to Execute 11^7 — (A •- 131 (C + D.

one of the operands and is used to store the result. In our canipIc, eight instructions arc needed to accomplish the task,

It is, in fact, possible tiP rriai,:e do with /elm addresses for some instructions. e.roLaddress instruet ions an applicable to a special memory organization, called a sifrc:k. A stack is a last-in-first-out set of locations. The stack is in a known locatton and. often, at least the top two elements are in CPI) reyislers. Thus, zero-address instruction* would Eac.rcnue the top two stack elements. Stacks are deseribcd in Appendix ltIA. Their use is explored further later in this chapter and in Ch4ipter I L

Table 1(3.1 summarizes the interprctalions to be placed on instructions with zero. one, two, or three addremcs• In each ease in the table, it is assumed that the midresN or the nest instruction is implicit, and that one operation with two source operands and one result operand is to be performed.

The number of addresses per instruction is a basic design. decision. Fewer addresses per insirticlion result in n-l4 PPE' primitive instructions, which requires t1]ess complex, CPU, It z.ilso resuits in instructions of shorter length. On the Whey hand:

l'411)1'1. 10.1 or Instrueii.ori Addresses (Nonbranehing Instructioa6)

Number of Addresses	Symbolic preieuiatkn	Interpretation
3	OP A, B. C	A •(— B 0.k' C
1	OP A. B	A •(— A OE ki
1	OP A	AC t AC Or A
3	OP	T q-• (T •- I) OP T

▪ = ll CL:JITJU.U01⁴

l Hr .41;.0;

▪ II. I rr rip! rry UT C:.:111StOT 0.11)17%

r j – I s II Li uid111;: fit L: | S'HiCk

programs contain more total instructions, which in general results in longer neap tion times and longer, more complex programs. Also, there is an important threshold between one-address and multiple-address instructions_ With one-address instructions, the programmer generally has available only one general-purpose register. the accumulator. With multiple-address instructions, it is common to have multiple general-purpose registers, This allows some operations to he performcd solely on registers. Because register references are faster than memory references, this speeds up execution. For reasons of flexibility and ability to use multiple rogi,-ters, most contemporary machines employ a mixture oE two- and three-add TL:N· instructions.

The design trade-offs involved in choosing the number of addresses per instruction are complicated by other factors. There is the issue 0r whether an address references a memory location or a register. Because there are fewer registers, fewer hits are needed for a register reference. Also, as we shall see in the next chapter, a machine may offer a variety of addressing modes, and the specification of mode takes one or more bits. The result is that most CPU designs involve a variety of instruction formats.

Instruction Set Design

One of the most interesting, and most analyzed, aspects of computer design is instruction set design. The design of an instruction set is very complex, because it affects so many aspects of the computer system. The instruction set defines many of the functions performed by the CPI I and thus has a significant effect on the implcmentation of the CPU. The instruction set is the programmer's means of controlling the CPU. Thus, programmer requirements must be considered in desiti.ning the instruction set..

It may surprise you to know that some of the most fundamental issues Mating to the design of instruction sets remain in dispute. Indeed, in recent years, the level of disagreement concerning these fundamentals has actually grown. The nio!A important of these fundamental design issues include the following:

- **Operation repertoire:** !low many and which operations to provide, and how complex operations should be
- **Data types; ' l'he** various types of data upon which operations are performed
- **Instruction format:** Instruction length (in bits), number of addresses, size of various fields, and so on
- **Registers: Number** of CPI.1 registers that can be referenced by instructions, and their use
- **Addressing: The** mode or modes by which the address of an operand is specified

These issues are highly interrelated and must be considered together in designing an instruction set. This hook, of course. must consider them in some sequence, but an attempt is ma& to show the interrelationships.

Because of the importance of this topic, much of Part Three is devoted to instruction set design. Following this overview section, this chapter examines data

types **and** op.eration reperioire. Chapter 11 examines addressing modes (which includes a consideration of regiMers) and instruction formats, Chapter 13 examines the reduced instruction set computer (RISC). RISC archilcciurc calls into question many of the instruction sel design decisions made in many conti2inporary commercial computers.

10.2 TYPES OF OPERANDS

Machirs12.instructicins oncrate on data. The most important general categories of data arc

- Addresses
- Numbers
- Characterg
- Logical data

We will see, in discussing addressing modes in Chapter 1.1, ihat addrusscs arc. in fact, a form of data. In many cases, some ea lculal ion must he performed on the orwrand reference: in an instruction to determine the main or virtual memory address. In this context, addresses can be considered to be unsigned integers.

Other common data types are numbers. characters, and logical 61a, and each of these is briefly examined in this section. Beyond [hal. Nome machines define specialiw:ed daia types or data strueitire:i. For example. there may be machine operators that operate directly in a list or a string of characters.

Numbers

All machine Languages include numeric data types. Even in nonnumeric data processing, there is a need for numbers to act as counters, field widths, and so forth. An important distinction between numbers used in ordinary ni4,Lhcmaties ink] numbers stored in a computer is that the latter lirnittd. This is true in two senses. First, there is a li mit to the magnitude of numbers representable on a machine and second, in the case of floating-point numbers. a Limit to their preds.ion. 't nos, the programmer is faced with understanding the consequences of roundin, overflow, and undcrflow.

Three types of numerical data are common in computers:.

- Integer or fixed point
- Floaling point
- Decimal

We examined the first two in some detail in Chapter 9. It remains Id say a few words about decimal numbers.

Although all internal compuier opera[ions Lire binary in nature, the human *users* of the system deal with decimal numbers. Thus, there is a necessity lo converl from decimal to binary on input and from binary to decimal on output. For applications in which there is a great deal of 1/0 and comparatively little, comparatively

simple computation. it is preferable to store and operate on the numbers in decimal form. The most common representation fear this purpose is packed decimal.

With packed decimal, with decimal digit is represented by a 4-bit code, in the obvious way. Thus, (1 — 0000, 1 — 0001, S = 1000, and 9 = 1001. Note that this is a rather inefficient code because. only 10 of 16 possible 4-bit values arc used. To form numbers. 4-bit codes are strung together, usually in multiples of 8 bits. Thus, the code for 24f is 0000001001000110 This code is clearly less compact than a straight binary representation. but it avoids the conversion overhead. Negative numbers can be represented by including a 4-hit sign digit at either the left or right end of a string of packed decimal digits. For example. the code 1.111 might stand for the minus sign.

Many machines provide arithmetic instructions for performing operations directly on packed decimal numbers. The algorithms are quite similar to those described in Section 9.3 but must take into account the decimal carry operation.

Characters

A common form of data is text or character strings. While textual data are most core venient for human beings. they cannot, in character form, he easily stored or transmitted by data processing and communications systems. Such systems are designed for binary data. Thus, a number of codes have been devised by which characters are represented by a sequence of bits. Perhaps the earliest common example of this is the Morse code. Today, the most commonly used character code in the International Reference Alphabet (IRA), referred to in the United Slates as the American Standard Code *for* Information Interchange (ASCII; see Table 7.1). IRA is also widely used outside the United States. Each character in this code is represented by a unique 7-bit pattern: thus, 128 different characters can be represented. This is a larger number than is necessary 10 represent printable characters, and some of the patterns represent *control* characters. Some of these control characters have to do with controlling the printing of characters on a page. Others are concerned with communications procedures. IRA-encoded characters are almost always stored and transmitted using 8 bits per character. The eighth bit may be set too or used as a par. itv bit for error detection. In the latter case, the bit is set such that the total number of binary ls in each octet is always.odd (odd parity) or always even (even parity).

Note in Table 7.1 that for the IRA bit pattern 011XXXX. the digits ft through 9 are represented by their binary equivalents, 0000 through 1001, in the rightmost 4 hits. This is the sante code as packed decimal, '['his facilitates conversion between 7-hit IRA and 4-bit packed decimal representation.

Another code used to encode characters is the Extended Binary Coded Dec- ¡mal Interchange Code (EBCDIC). EBCDIC is used on 1BM 5/390 machines, It is an 8-bit code. As with IRA, EBCDIC is compatible with pocked decimal. In the case of EBCDIC. the codes 11110000 through 11111001 represent the digits 0 through 9.

Logical Data

Normally, each word or other addressable unit (byte, hal fword, and so on) is treated as a single unit of data. It is sometimes useful, however, to consider an n-hit unit as consisting of n L-hit items of data, each item having the value 11 or 1. When data are viewed this way, they arc considered to be *logical* data.

There are two advantages to the bit-oriented view. First, we may sometimes wish to store an array of Boolean or binary data items. in which each item can take on only the values 1 (true) and 0 (false). With logical data, memory can be used most efficiently for this storage.. Second, there are civeasions when we wish to manipulate the bits or a data item. Forexample, if floatin g-point operations are implemented in software, we need to be able to shift significant bits in some operations. Another example: To convert from IRA to packed decimal, we need to extract the rightmost 4 hits of each byte.

Note that, in the preceding examples, the same data are treated sometimes as logical and other times as numerical or text. The "type" of a unit of data is determined by the operation being performed on it. While this is not normally the case in high-level languages, it is almost always the case with machine language.

10.3 PENTIUM AND POWERPC DATA TYPES

Pentium Data Types

The Pentium can deal with data types of 8 (byte). 16 (word), 32 (doubleword). and 64 (quadword) bits in length. To allow maximum flexibility in data structures and efficient memory utilization, words need not he aligned at even- numbered addresses; doublewords need not be aligned at addresses evenly divisible by 4; and quadwords need not be aligned at addresses evenly divisible by 8. However, when data are accessed across a 32-bit bus, data transfers take place in units of doublewords, beginning at addresses divisible by 4. The processor ameris the request for misaligned

Table 111.2 Pentium **Data Types**

Data Type	1.1eNcripticin
CiuncrRI	13. 1.e, word (16 **bits), doulik.w4ird.t.12 hits). and quadvigird (#.1 1 bits) locations with arbitrary binary contents.**
Integer	**A signed binary value contained in a byte, word. or douhle•ord, using twos complement representation.**
Ordinal	**An unsigned integer contained in a byte, word, or doubleword.**
Unpacked binary coded decimal fEl.CD1	**A representation (rf a BCD digit in the range it Ihroueji 9, with *one* digit in each byte..**
Packed 13C0	**Packed byte representation of two liC1) digits: value in the range to 99.**
Near pointer	**A 32-hit effective aditivss that represents the oFfsct within a segment. Used lug all pointers in Al nonsegmented memory' and [car rofercnees within a set:mein in :3 segmented memory,**
Elit field	**A contiguous sequence of hits in which the position 01 each hit is considered as an independent unit, A hit. string can hL Cin at arty hit position of an' byte and can contain up to - 1 hi k.**
Byte strive	**A oontiguous sequence of bytes, words. or doublewords, on 11,111ring form zero to 2" - / bytes.**
Floating point	**See Figure 10.1.**

values into a sequence of requests for the has transfer. As with all of the Intel 807e.-. 6 machines, I he Pentium users the little-endian styie [kit is, the least significant byte iii stored in the lowest address (see A opc.ndix 1013 for a discussion of enclianness).

The Iwle, word. doubleword, and quadword are referred to as general datd types. In addi lion. the Pentium supports an impressive:array of specific data rypts that are recognized anal operated on by particular instructions. Table 10;2 stoma-Finis these types.

Figure KO 'illustrates the: Pentium nurnericat data types. The signed integers **are** in twos complement representation and may be RI. 32. or 04 bits ionR. The floating-

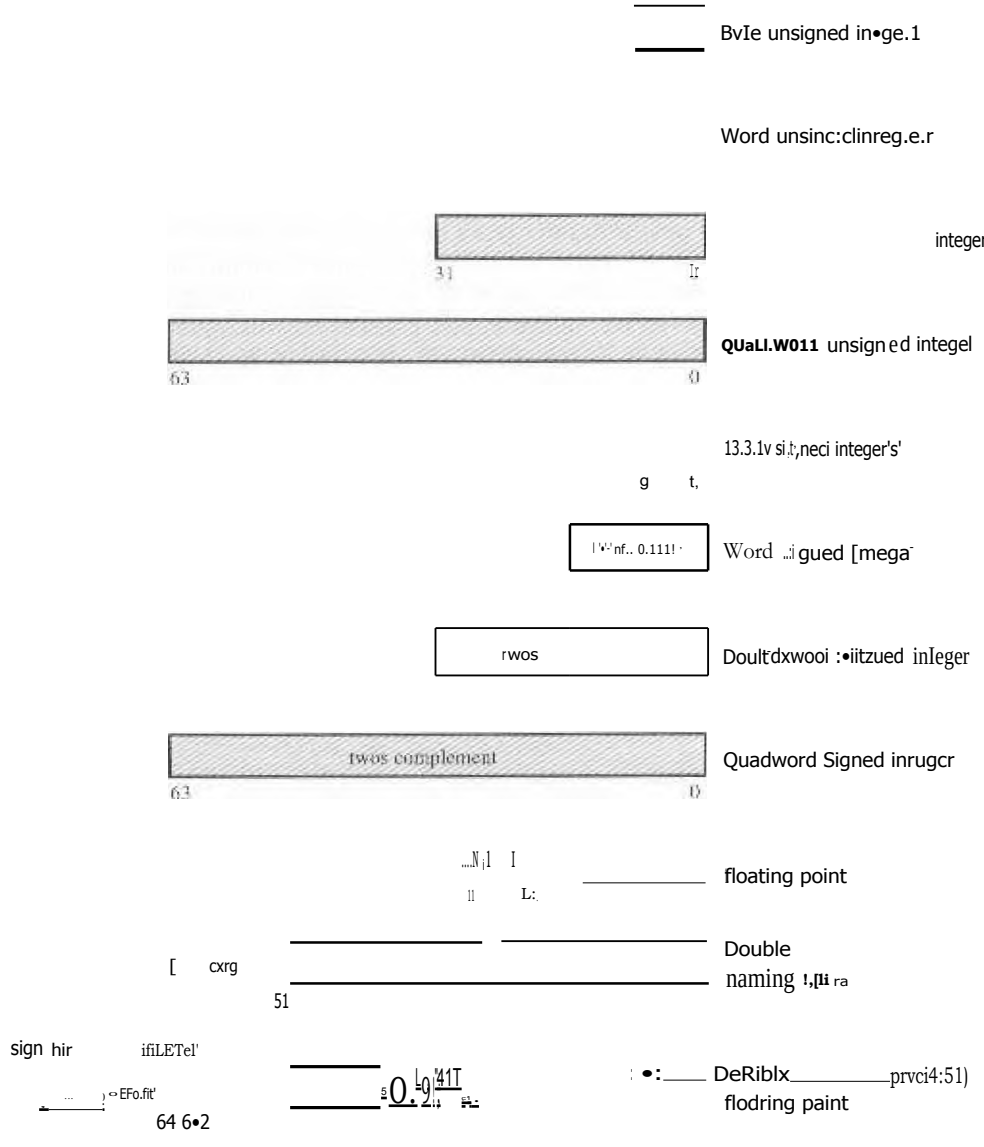


Figure WA Pentium Numeric Data HicruaLs

point type actually reIcrs Lo a st.!t of types that are used by the floating-point unit and operated on by floating-point instructions. The three floating-point representations conform to the IEEE. 754 standard.

PowerPC Data Types

'The PowerPC' can deal with data types of g (byte). 16 (ha I fword), 32 (word), and fib (doubleword) bits in length. Some instructions require that memory operands be aligned on a 32-bit boundary, In general, however, alignment is not required. One interesting feature of the PowerPC is that it can use either little-endian or big-cndian style: that is, the least significant byte is stored in the lowest or highest address (see Appendix 11111 for a discussion of endianness).

The byte, halfword, prd, and dOLINCward arc general data types. The processor interprets the contents of a given item of data cleNntling on the instruction. The fixed-point processor recognizes the following data types:

- **Unsigned byte:** Can he used for logical or intcgcr ariihmetic oper a tions- 11 is loaded from memory into a general register by zero extending on the]eft to the *full* register size.
- **Unsigned halfword:** As for unsigned byte, heal rot 16-bil quantities-
- * **Signed halfword:** Used for arithmetic operations: Eoaded into memory by sign extending on the *left* to full register size (i.e., the sign bit is ref iicated in al] vacant posi lion6).
- * **Unsigned word:** for logical operations and as an address pointer.
- ■ **Signed word:** Used for arithmetic ()pew ions.
- **Unsigned doubleword:** Used as an address pointer.
- **Byte string:** From 0 to 128 bytes in length.

In addition. the PowerPC supports the single- and double-precision floating-point dati types defined in IEEE 754.

firti.S

The number of different opeodes varies widely from machine to machine, 1Iowever, the same general type's Of operations are found on all machines. A useful and typical categorization is the following:

- ▪ Data transfer
- • Arithmetic
- • Logical
- ■ Conversion
- *
- ■ System control
- • Transfer of control

Table 10.3 (based *on* Il l Asr' HMI) lists common instruct ion types in each category. This section provides a brief survey of these various types of operations,

Table 10.3 Common, Instructiou Set OpratioirliF.

	Operation Name.	Dem....610 50n
Data sransfer	**MOVE ft NI 11 Sf LI)**	Transfer wont or hi SICk Horn SOW-CC 1.0 cichtination
	Su }re	TranSfer **Want** from processor LL.) memory
	1.oad	Trausfer wort! .rmiL (Nrnor!.. to proccsor
	•xchim.[412	cup Le n LP% of source and destma ti on
	'Cie rr (rOSCE)	Transfer wort] of Os to dcsl inH ri{
	&i!.	'1 runr442.] word of Is. t4 d esti ii i I is n
	Push	1 ran2...f121 word from souro2 1 c1 lop of :Ltack
	Pop	**TTH** nNtu r 9eOic.1 from top iii slack to destination
A1laure1ie.	Add	(:o m po le 'Loin Of LINO operand:,
	Surrirad.	**Caril riaLc** di nre31C.0 of two operands
	Multiply	pu e ptr.Fduct of rwo operHnds
	Divide	s111 pulC quaitlit cir two 4pervinOs
	A'osolute	,pperaliLt by its **9118**(51 IA IL valL1C5
	Ne.ga Le	Ch nge Or operand
	[ucre iilenl	Add t to cppetarid.
	Decrcinera	Sohttact] from opernmi
Logical	AND	
	OR	
	NOT	Perform the specified lo6iica1 ripe ratio!' hitwisc
	(Complement)	
	E)Zel usive-OR	
	Test	Test specified condition% sei) !lased on outcome
	Compare	Make or a rntrn;--I ic ea nsoi L of two Or more. opel m.d5! **NCI** Ila.p..(s1 based on (uncoil it!
	Set control vuriabl es	Claris inA ructions to KI controls fol prokeetspit purpose& **InlCirupL** ti mer con e Le.
	Shift	(rig{ 11 shift op.:mild, inLr{)d1C.1114 ODLiStanLS at end
	Rotate	O) 1.) stuit ilpernnci, with wraparound end
Transfer ryf control	Jun Lp flwari.ekt)	UncondiLiciloil 11'i:11131'er! LODA1 PC' %%II h !Teel fled address
	unip con dir.iona I	'I em. condi Llo n: either load PC' tvitkt specified addre..sg ar do no bins, In1s4 cm condition
	Jump Lo subroutinc	Place curl-42111 propdui control mfgrind Li on in known location; lump to specir led add rC%S
		Re Waco conwn Ls of PC and other register front known 1oca Lion
	F..x12ci.11.12	Fes ch. operlind from specified local ion **Hn** d wcectl Le as insl ruetign: di I not nincliry PC'
	Skip	ncremcni PC Lo hkits next instruction
	Skip condi knurl	feet pr2ci condition! etcher skip **or *d0*** nothin44 based Du condition
		Slop program cAocution
	'ciinr iL	rop pr igraul execution:. test specified 42....)n dition. repc re.stime execution when condition k satisfied
	.opt ration	fin opuru tic in is performed, but program execution is cOntirruiLd
In puLlous pu1	Input (mod)	'If aris Cor data fri5111 **SpLci** 'Jed 1..'0 Nil or rieviee 1.0 destinati.311 main memory **15r** proCeMol.' register)
	()ION!! (wrilc}	Transfer d al a from hpccifid source to port or 1.16.01C42:
	St;iri 1:4.)	**T** ransfer ins! tucl ir m s to 1.:0 processor I n inil in Le 170 operation
		I'ra sk.r claim i n LcFrrllutit7ll 110111 110 N'yStcrn Lci speuir3ed de:30111.MM
f;onveryi1 f1I	T r;1111111.1.12	T'ra nsiate. values in H {**A** memory based cl n a Lable.of corresipondences
		Convert 41.1c conients of **kl word CrOn1 011e. RIM** lo nni.)Lhe.i {e.g., packed decimaf to n u ry}

Table 104 CPU Actions for Vat. jou: I 'ypes of Operations

Data transfer	Transfer dal. n.c.trrl one location lCl am,' her 1 memory is ink.olved: Determine memory.address Perform virtual-In-aclual-memor:, address trawdorination (Meek (melte. Initiate memory
Arithmcite	May involve dam irxmr.sr, hcf Ore •andlor alter Ferfurat function in All) Si condition code: and flaFs
Logical	Sainc. as arithm:21EC
Conversion	Similar In ,;irithiii;•ifc r.irLd logical. May involve. special logic In 1et 1(1TM conversion
Transfer of control	Updritc program ciminer_ Fur stihruntine callrreIurn, manap:. FirCIrricWT posing and linkage
	Issue command to I/O module. If memory-mapped 1.'0, determine inentu3 y-rnapped kddrum

together with a brief discussion of the actions taken by the CPI I to execute a particular type of operation (summarized in Table l0.4). The lattei topic is examined in more detail in Chapter 12.

Data Transfer

The most fundamental type of machine instruction is the data transfer instruction. The data transfer instruction roust specify several things. First, the location of the source and destination operands must hu specified. Each location could be memory. a register. or the top of the stack. Second, the length of data to be transferred must be indicated, Third, as with all instructions with operands, the mode of addressing for each operand must be specified. This latter point is discussed in Chapter I t.

The choice of data transfer instructions to include in an instruction set exemplifies the kinds of trade-offs the designer must make. For example, the general location (memory or register) of an operand can he indicated in either the specification of the opcode or the operand. Table If15 shows examples of the most common IBM Si390 data transfer instructions. Note that there arc variants to indicate the amount of data to be transferred (8. 16,32, or 64 bits). Also, there are different instructions for register to register. register to memory. and memory to register transfers. In contrast, the VAX has a move (MOV) instruction with variants for different amounts of data to be moved. but it specifics whether an operand is register or memory as part of the operand. The VAX approach is somewhat easier for the programmer, who has fewer mnemonics to deal with. However, it is also somewhat less compact than the IBM S/390 approach, because the location (register versus memory) of each operand must be specified separately in the instruction. We will return to this distinction when we discuss instruction formats, in the next chapter,

In terms of CPU action, data transfer operations arc perhaps the simplest type. If both source and destination are registers, then the CPU simply causes data to be

l'oble 10.5 Eitaittpla:s of IBM SI391) **Data Transf.2: Operation**

Operatil n Mnemonic	Name	Nitwit:er of Rib Transferred	Description
]	Load	3 7	1 11111:,1..`1" i'morn murrLary in i Lgistu
I-I!	Loin] hall word	L6	Trail4fer :1.11m iise.irlors.. to c4.....sLee
I..R	I..oad	32	TraLlS1 01' Jrcini rEliSICT 1.0 Eckister
LER	Lor.id (5110s1)	32	Traw.:11.n Isom. floathig-poirst se.gisker io flo;ring-point R..gisic r
LE	Lokiil (short	32.	Tr.:Ansi:Qs Imin memory ;a) nou[ing•poiuti Tc..6.i.F.Lei
LDR	Load (long)	64	Transk.r l'i-orri flouting-point ft:lilt:La 10 fliiiiiii1F-point•rceisftr
[„n.	f..oad (toric)	4'4	Trall.510r From rnornory Ill 11i5;11....rol poln1 rqisr.er
ST	SIorl	32	Traitcfc 1'41111 1.2.0 ci c i n incurcH
NTH	S1CITE hall:word	1G	Tun...;I'ff I'r oriS reAisl.ci 10 rrienwry
STC	%Skil-12 chari:i.....tc r	g	Transtc-r rrilis, ii.. i.sl.....r 1c1 niumory
STE	Sufri: 1:€.1tort)	32	TrEmsfur from Lloalini-poinI. ropm...1.- 1.0 memory
STD	Su:Av. (long}	il..1	Tur1il)21' from Elry.iiirip-point reg1hi4.-r hi merao....y

translernal from one register to another: this is an cpperatiOn **inlernal lo** the CPU. If one or both operands **are** in memory, then the CPU must perform some (Pr all of the roliowing actions!

'L. Calculate the memory address, based on the addre: :s mode (discussed in Chapter 11),

1 If the addre.s:s. refers to viri ual memory, trail:date from virtual to actual memory address-

3, Determine whci her the addressed item is in cache.

4. If not, issue. I command hr the memory module,

Arithmetic

Most n1;ichilieS provide. the basic arithmetic opermitions of lidd, subtraei, **anti** divide. Thesc.. zlre invariably provided for signed integer (fixed-point) numbers. Often they are]so provided for floating-point and packed deciirral numbers.

Other possible operations **include a variQly** cif single-operand instructions: for example.

- AbwInte: Take 1.11 2 absolute value of the operand,
- Negate; Negate the operand.
- Increment: Add L hr the operkuld-
- Decrement; Subtract L from the operand.

The execution of an arithmetic instruction may involve data Irtii rer operaions to position operands for input to the ALL, and to deliver the output of the ALL'. **Figure** :4.5 illustratcs the movcmenis involved in both (Lila transfer and arithmetic operations. In addition, of course, the ALI! portion of the CPU performs thc desired operation.

Logical

Most machines also provide a variety of operations for manipulating individual bits of a word or other addressable units, often referred lo as "**bit** twiddling." They are based upon Boolean operations (see Appendix A).

Some of the basic logical operations that can be performed on Boolean or binary data are shown in 'Table 1011. The NtlY1 ' operation inverts a bit. AND, OR. and Exclusive-OR **(XOR)** are the most common logical functions with two operands. EQUAL is a useful binary test.

These logical operations can be applied bitwise to n-hit logical data units. Thus, if two registers contain the data

(R1) 10100101
(R2) 00001111

then

(RI) AND (R2) — 00000101

where the notation (X) means the contents of location X. Thus. the AND operation can be used as a *mask* that selects certain **bits** in **a** word and zeros out the remaining bits. As another example., if two registers contain

t111) = 10100101
(R2)= 11111Ill

then

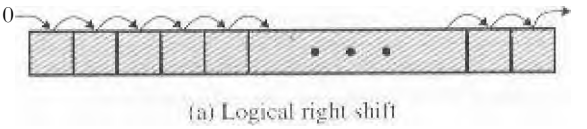
(RI) XOR (R2) = 01011010

With one word set **to** all 1s. I he XOR operation inverts all of the bits in the other word (ones complement).

In addition to bitwise logical operations, most machines provide a variety of shifting and relating functions. The most basic operations are illustrated in Figure 10.5. With a **logical shift, the** bits of a word are shifted left or right. On one end. the bit shifted out is lost. On the. other end, a 0 is shifted in. Logical shifts arc useful primarily for isolating fields within a word. The Os that are shifted into a word displace unwanted information that is shifted off the other end.

Table 10.6 Basic Logical Operations

P	Q	NOT P	P AND 0	P OR Q	P XOR Q	NO
	0	,	0	0	0	l
IF	l	l	0	l	I	0
I	0	0	0	l	l	0
l	1	0	1	l	0	l

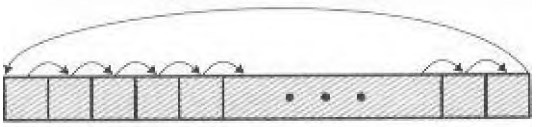


(a) Logical right shift

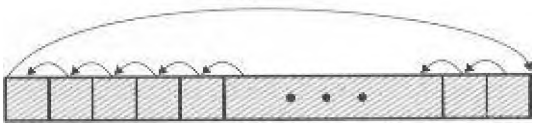
:tl.l. 1A3.g'k!al left shill.

I· I A ri thi:11 211.c. right shin

(d) Arithmetic left shift



(e) Right rotate



(1) Left rotaic

'Vire 10.5. Shill and Rotait. Operadoos

Af.,, art example, suppose we wish to transmit charvictcr.1 '. of data to an 110 device 1. characier at a (irnc. &emsh memory word is; [6 hits in length and contains two characters, wi2 mum *wzpack* the characters tic lore they can he. &M-To send the two chuirneten; ill word.

I. I Amd the word into a registi:r.
2. ANL) with the value [11[1.1 ! I (C)N COO. This masks ()unite character on the riht.
3. Shift to thenght eight kiines. This shifts the rem:lining character to thc right half of the rcgistur,
4. Pei Corm 110. The 110. module reads the lower-order 8 hih, from the data bus.

The preceding steps result in sending the left-hand character. To scrid the.right-hand eh ll'acter.

1. Load the word again into the register.
Z. AND with 0000000011111111.
Perform I/O.

The arithmetic shirt operation treats the data as a signed integer and does not shift the sign bit. On a right arithmetic shift, the sign hit is replicated into the bit position to its right. On a left arithmetic shift, a logical left shift is performed on all bits but the sign bit, which is retained. 'these operations can speed up certain arithmetic operations. With numbers in twos complement notation, a right arithmetic shift corresponds to a division by 2, with truncation for odd numbers. Bolt' an arithmetic left shift and a logical left shift correspond to a multiplication by 2 when there is no overflow- If overflow occurs, arithmetic and logical left shift operations produce different results, but the arithmetic left shift retains the sign of the number. Because of the potential for overflow. many processors do not include this instruction, including PowerPC and Itanium. Others, such .as the IBM S/390, do offer the instruction. Curiously, the Pentium architecture includes an arithmetic left shift but defines it to be identical to a logical left shift.

Rotate, or cyclic shift, operations preserve all of the bits being operated on One possible use of a rotate is to bring each fiit successively into the leftmost bit, where it can be identified by testing the sign of the data (treated as a number).

As with arithmetic operations. logical operations involve AI.0 activity and may involve data transfer operations. Table 1(1.7 gives examples of all of the shift and rotate operations discussed in this subsection.

Conversion

Conversion instructions are those that change the formal or operate on the format of data. An example is converting from decimal to binary. An example of a more complex editing instruction is the S/390 Translate (TR) instruction. This instruction can be used to convert from one 8-bit code to another, and it takes three operands:

TR RI, R2, L

he operand R2 contains the address of the start of a table of 8-bit codes. The. I.. bytes starting, at the address specified in RI are lranslated. each byte being replaced

Table 10.7 F...XaMpli:s of Shift and Rotate. Operations

Input	Operation	Resoll
10100110	Logical right shift (3 lit)	00(1111100
10100110	Logical tell shift (3 bits)	001 10000
10)001 ID	*Arithmetic* right slidt (3 bits)	1I1.1011X)
10100110	Arithrhoic left shill (3 bits)	101 moon
101001 lit	Right rotate (3 hits)	j 01(11)
l0l00110	Left rotate (3 hits)	00110101

by the contents of a table entry indexed by that 117. .1e. 1;or example, to translate from EBCDIC to IRA, we first create a 256-byte table in storage locations, say, 1000-WET hexadecimal. ' Hie table contains the characters of the IRA code in the sequence of the binary representation of the EBCDIC' code: that is, the IRA code is placed in the table at the relative location equal to the binary value of the HI3CDIC code of the same character. Thus. locations IWO through 10F9 will contain the value 30 through 39, because FO is the kBCDIC code for the digit 0. and 30 is the IRA code fot the digit 0, and so on through digit 9, Now suppose we have the EBCDIC for the digits 1984 starting at location 2100 and we wish to translate to IRA, Assume the followinw

- Locations 21.00 .2103 contain Fl F9 1-'8
- R1 contains 2100.
- R2 contains 1000:

Then, if we execute

TR R1, **R2,** 4

locations 210(1-2103 will contain 31 39 3S 34.

Input/Output

Input/output instructions were discussed in some detail in Chapter 7. As we saw. there are a variety of approaches taken, including isolated programmed 110. memory-mapped programmed 110, DMA, and the use of an 110 processor. Many implementations provide only a few **110** instructions, with the specific actions specified by parameters. codes, or command words.

System Control

System control instructions are those that can he executed only while the processor is in a certain privileged state or is executing a program in a special privileged area of memory. rypically, these instructions are reserved for the use of the operating system.

Some examples of systetn control operations are as follows, A system control instruction may read or alter a control register; we discuss control registers in Chapter 12. Another example is an instruction to read or modify a storage protection key, such as is used in the S/390 memory system. Another example is access to process control blocks in a multiprogramming system.

Transfer of Control

For all of the operation types discussed so far, the next instruction to be performed is the one that immediately follows, in memory, the current instruction, However, a significant fraction of the instructions ill any program have as their funelion changing the sequence of instruction execution. For those instructions, the operation performed by the ('Nt.' is to update the program counter to contain the address of some instruction ill memory.

There are a number of reasons why transfer-of-control operations are required, Among the most important are the following.:

1. In the practical use of computers, it is essential **Lo be** able to execute, each instruction more than once and perhaps many thousands of times. It may require thousands or perhaps millions of instructions to implement an application. This would be unthinkable if each instruction had so he, written out separately. Fla table or a list of items is to be prt locssed, a prowarn loop is needed. One sequence of instructions is executed repeatedly to process a]] the data.
2. Virtually all programs **involvc some** &vision making. We would like the computer Io do one thing if one condition holds, and another thing if another condition For example, a sequence of instructions computes the square root of a number. At the **suiri** of the sequence. the sign of the number is tested. If the number is negative, the cornpuiation **is not** performed. tru1 art error condition is reported.
3. To compose correctly tl 11.1 rgC **or even niedim n- ize** computer program is an exceedingly difficult task. It helps if there are mechanisms for breaking the task up into smaller pieces that can be worked on one at a time.

We now turn to a discussion of the most common transfer-of-control **operations** found in instruction sets: branch, skip, and procedure call.

linuich Instructions

A branch instruction. also called a jump instruction. has as one or its.operands the address. of the nexl instruction to be executed. Most often, the instruction is a *condirlondi t,Franch* instruction. '1'hat is, the brandh is made {update program counter to equal address specified in operand) only if a certain condition k met. 01herwise, the next instruction in sequence is executed (increment program counter as usual).

There are two common ways of generating the condition to be tested in a conditional branch instruction. First, most machines provide a l-bit or multiple-bit condition code that is set as the result.of some operations. This code can be I hough" *of* as a short user-visible register. As an example, an arithmetic operation (ADD, SUBTRACT, and so on) could set a 2-hit condition code with one of the following four values: 0, positive, negative, overflow. On such a machine, there could be lour differenI conditional branch instructions:

BR? X	Branch to location X if result is positive.
BRN X	Branch to location X if result is negative.
l-3 R/. X	Branch to location X if result is zero.
BO X	Branch 10 location X if overflow occurs.

In all of Ihese cases, 1he result referred to is the result of the most recent operation that set the **ci HILliti on code.**

Another approach that *can* be used with a threc-addressinstruction formal is to perform a comparison and specify a branch in the same instruction. For example,

1.314 **R I, R2. X**	Branch to X if contents of Rl = contents of R2.

Figure 10.6 shows examples of these operations. Note that a branch can he either forward (an instruction with a higher address) or *backward* { lower address). The example shows how an unconditional and a conditional branch can he used to create a repeating loop of instructions. The instructions in locations 202 through 21i) will he executed repeatedly until the result of subtracting Y from X is 0.

Skip Instructions

Another common form of transfer-of-control instruction is the skip inSITLIC' tion. The skip instruction includes an implied address. Typically, the skip implies that one instruction be. skipped, thus, the implied address equals the address of the next instruction plus one instruction-length.

Because the skip instruction does not require ai destination address field, it is free to do other things. A typical example is the increment-and-skip-if-zero (ISZ) instruction. Consider the following program fragment:

```
3 C 1
•
3011 r.sa R1
31C RR 3C.1
311
```

In this fragment, the two transfer-of-control instructions are used to implement an iterative loop, Rl is set with the negative of the number of iterations to be performed. At the end of the loop, **RI** is incremented. If it is not 0, the program branches back to the beginning of the loop. Otherwise, the branch is skipped. and the program continues with the next instruction after the end of the loop.

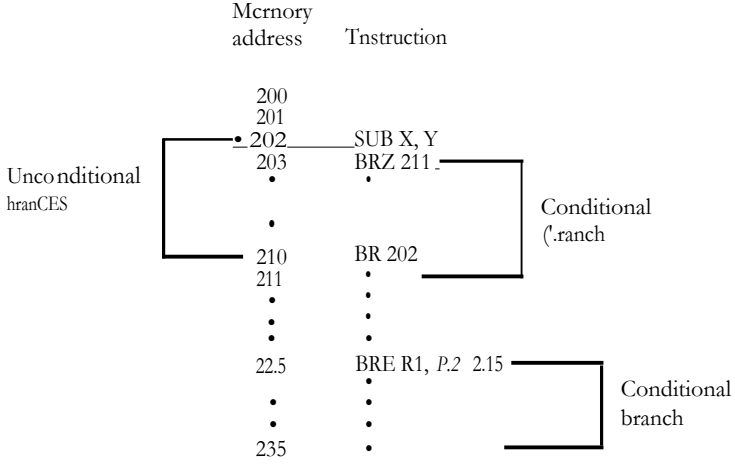


figure 10.6 Branch Instructions

Procedure eaIt Instructions

Perhaps the most important innovation in the development of programming ianguages is the *procedure.* A procedure is a self-contained computer program that is incorporated into a larger program. At any point in the program the procedure may be invoked, or *coffed.* The processor is instructed lo go and execute the entire procedure and then return to the point from which the call took place,

The two principal reasons for the use of procedures are economy and modularity. A procedure allows the same piece of code to be used many times. This is important for economy in programming effortand for making the most efficient use of storage space in the system (the program must he stored). Procedures also allow large programming tasks to be subdivided into smaller units. This use of *nu̇gdulariry* greatly eases the programming task.

The procedure mechanism involves two basic instructions: a call instruction that branches from the present location to the procedure. and a velum instruction that returns from the procedure to the place from which it was called. Both of these are forms of branching instructions.

Pigure 10,7a illustrates the use of procedures to construct a program- In this example, there is a main program starting al location 400(1. This program includes a call to procedure P ROC71, starting at location 4500. When this ca]] instruction is encountered, the CPI: suspends execution of the main program and begins execucion of PROC1 by fetching the next instruction from location 4500. Within PROC1, there are two calls to PROC2 at location 4800. In each case, the execution of PROC I is suspended and PROC2 is executed. The RE11.:RN statement causes the CPU to go back to the calling program and continue execution at the in:t1ruc-[ion after the corresponding CAUL instruction. This behavior is iilustrated in Figure 10.7b.

Several points are worth noting:

I. A procedure can be called from more than one location.
2. A procedure call can appear in a procedure. This allows the *ne.viiig* of procedures 10 an arbitrary depth.
3. l ath procedure ca]] is matched by a return in the called program.

Because we would like Lo he able to cal] a procedure from a variety of points. the CI-[U must somehow save the return address so that the return can take place appropriately. There are three common places for storing the return address:

- Register
- Start of called procedure
- Top of stack

consider a machine-language instruction CALL X, which stands for *COB procedure* ut *IOctifiw2* If the register approach is used, CALL X causes the following actions:

RN 4. PC -F
PC

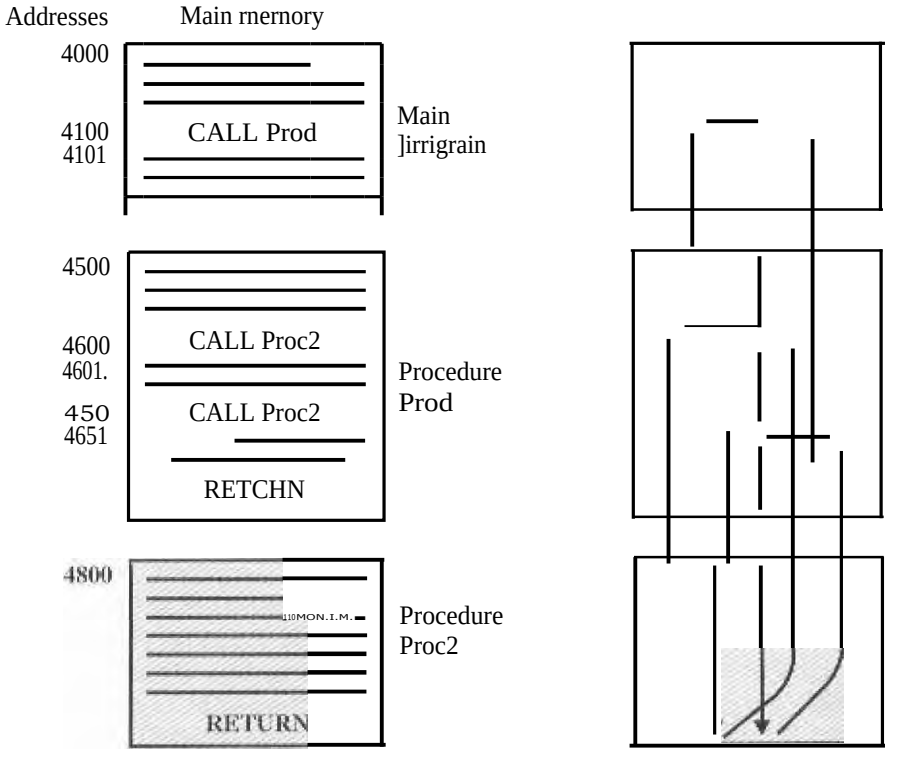


Figure 10,7 "s'e.cied Proccdur..2.s

irvhere RN is a register that is kilways used for this purpose. PC is the program' counter, and A is the instruction length. The called procedure can ntjw skive Ile con-Lents of RN to be used for lite later return.

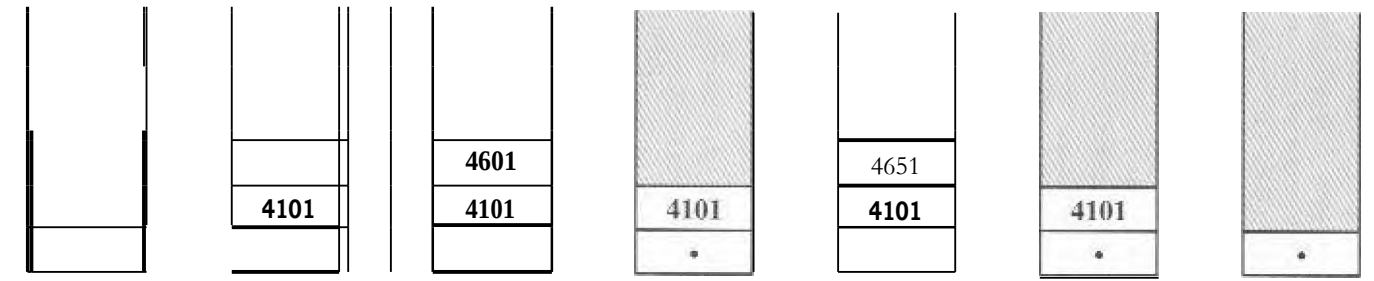
A second possibility is to store the return address at the start of the proceduru. In this case, CALL X causes

X PC — A
PC, t— X —1

This is quite bandy. I he return address h;is keen stored safely away.

Both if the preceding approaches work and have 1 icen used. The only I imi iation of these approaches is that they prevent the use of *reentrant* procedures, A reentrant procedure is one in whieh it is pOssible 10 lmeive several calls open to it 6 1U same the. A recursive procedure (one that calls 'Bell) is an example. or the: use of ibis feuture.

A more enerail and powtirfui approach is to use *a* stack (see Appendili 1,0A for a definition of the stack). When the CE(..) executes a call, il places the return address on !tic stack, When it executes return, it use!, the address on the slack. Figure ma illustrates the use of .the stack.



la qtat-1. leuntenis

I I)} Atter CALL Prue!

(e) CALL Fre.c2.

IA} After RETURN

(e) After CALL Prue2

tt) At a RETURN

Atter RETURN

Figure 10-S Ike of Stock to lEnplernunE Nestcd. Subroutines of Figuit 10.7

In addition to providing a return address, it is also often necessary to pass parameters with a procedure call. These tan he passed in registers. Another possibility is to store the parameters in memory atter the CALL instruction. In this case., the return must he to the location following the. parameters. Again, both of these approaches have drawbacks, **If** registers are used, the called program mid the calling program must be written to assure that the registers are used properly. The storing of parameters in memory makes it difficult to exchange a variable number of parameters. Roth approaches prevent the use of reentrant procedures.

A more flexible. approach to parameter passing is the stack. When the processor executes a call. it not only stacks the return address, it stacks parameters to be passed to the called procedure. The called procedure can access the parameters [torn the slack. Upon return, return parameter's can also be placed on the stack. The entire set of parameters, including return address, that is stored for a procedure invocation is referred to as a *stack frame.*

An example is provided in Figure 10.9. The example refers to procedure **P** in which the local variables .1.˙ 1 and x2 are declared, and procedure 0. which can be **called** by P and in which the local variables vi and y2 are declared. In this figure, the return point for each procedure is the Iirsi item stored in the corresponding stack frame. Next is stored a pointer to the beginning of the previous frame. This is needed if the number or length **of parameters to be slacked is variable.**

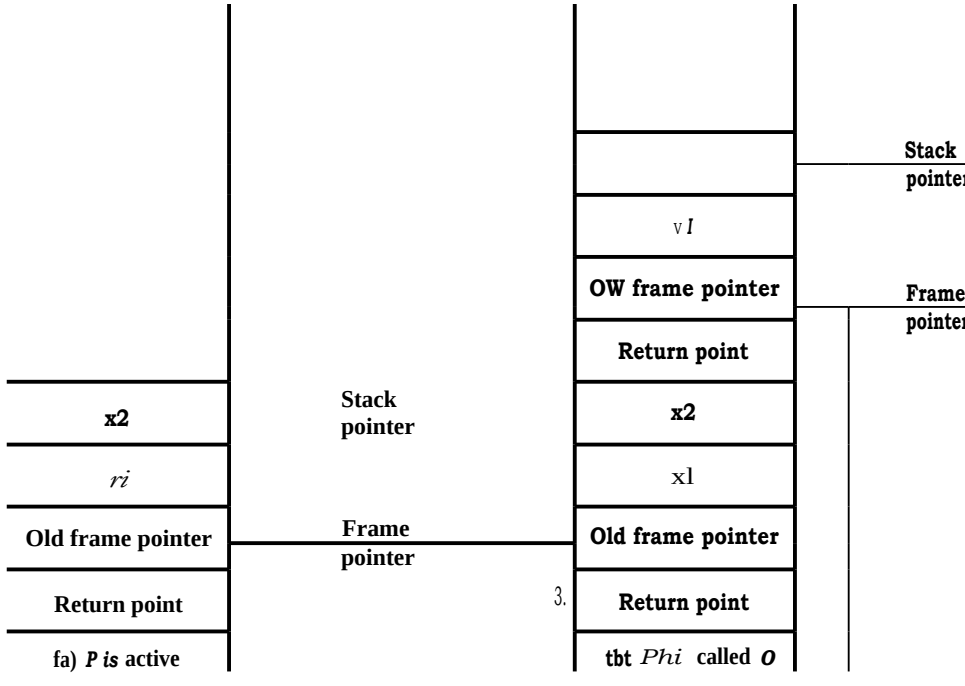


Figure 10.9 Stock Frame Growth Using Sample Procedures **P and**

10.5 PENTIUM AND POWERPC OPERATION TYPES

Pentium Operation Types

The Pentium provides a complex array of operation types, including a n umber of specialized instructions. The intent was to provide tools for the. compiler writer to produce optimind machine language translation of high-level language programs. Table 10.8 lists the types and gives examples of each. Most of these arc the conventional instructions found in most machine instruction sets, but several types of instructions arc tailored lo the 80x86/Pentium architecture and are of particular interest.

CA/Return Instructions

The Pentium provides four instructions to support procedure callireturn: CALL, ENTER. LEAVE, RETURN. It will he instructive to look at the support provided by these instructions. Recall from Figure 10.9 that a **common means of** implementing the procedure callireturn mechanism is via the use of stack frames. When a new procedure is called, the following must be performed upon entry to the new procedure:

- Push the return point on the stack.
- Push the current frame pointer on the stack.
- Copy the stack pointer as the new value of the frame pointer.
- Adjust the slack pointer to allocate a frame.

The CALL. instruction pushes the current instruction pointer value onto the stack and causes a jump lo the entry point of the procedure by placing the address of the entry point in the instruction pointer. **In** the 8O and 81J86 machines, the typical procedure began with the sequence

```
?USA    Ear
MOV          EEP
'1T71P  ESP, space, fox_loca=
```

where EBP is the frame pointer and ESP is the stack pointer. In the 80286 **and later machines, the ENTER instruction** performs all the aforementioned operations in a single instruction,

The ENTER instruction was added to the instruction set to provide direct support for the compiler. The instruction also includes a feature for support of what are called nested procedures in languages such as Pascal, C01-301., and Ada (not found in C or FORTRAN). It turns out that there are better ways of handling nested procedure calls for these languages, Furthermore, although the ENTER instruction saves a few bytes of memory compared with the PUSH, MOV, SUB sequence (4 bytes versus 6 bytes), it actually takes longer to execute **(10** clock cycles versus ti clOck cycles). Thus, although it may have seemed a good idea to the instruction set designers **to** add this feature, it complicates the implementation of the processor while providing little or no benefit. We will see that, in contrast. a RISC approach

Table 10.8 Pentium Operation Types (with EXarrip[CS Of Typical OpernitionS)

lostruclion	Descriplion
Data Movement	
MO's/	Move opernrid, bciwecn r<.i.ster5 cm beLwe.an rugistC1:
1[3] (.1 51-1	+Sh7 :•1)12 ra ad onto -.tack.
pii.!!-.1 I A	Push n11 rep.iste.is on muck.
MC 1[0]/!;X	hyte. word. dw.:5rd, sian extt.rided. **Movi...s:** [1]'yIC **LL word Dr a** vs.o.rd It 41 01.1.h1C wil **WIth iwoh-compleinent** E[En 42xL1231SirM.
I . EA	Lo.:Id CifoLtitiC a cldrexa. Loads 1.1112 01:1E1 of (FLO, 30 u m; operand. rather 1han its +**Jahn iii** the demination operand.
XI-AT	Tahle Replaces a byu.. in AL with al-11,W Ivorn riser-uodod irarodirLion. When XLAT is exc coLed. AL thould have Etn unsignctl index to the XLA•I• **uhkui Os** the ccm[ents o(AL Crom I h.: table index Lo the nble
IN. 91.11'	fnput. ouspul crrkrisnd From 1)0 spacc-
Arithmetic	
[6]1.1.11 D	Add operands.
51.:il	Su btract ornands.
Mill,	unsigned **ini.cgwr** multiplica.sion. with byte.. wo3d. or doublc or ap4m ands, and won.l. doolik.mnrd, of co..svd.Word YE9.1 IL.
11)**1** V	Signed 12ivide,
Logical	
AN L)	**AND operistids.**
KIN	Iii! t.s:..s l and set. Operistin on D hit CD.31.ci cspernn a. **The** inssrucl3on copie.s th42 current va I tic clt a bit in flug CF and hu lS. the orkgi MI I Ion 10 I.
13SF	**Bil** m:201 Corward. Scans n word cll. do ti bleword sot a 1 -b it And ...toms the nunsber ot' the l'iro. I. **bit** into kl reOster.
SF31_,...S HR	Shill **logipol** lull ur righ.1.
SAUSAR.	Shin. **arithmetic It** or sight.
RODTZ OR	Rotate loCI. (IT **Tight.**
SEIce	Sets a bytes to 1...e.ro ci f ono 11.12puii di sig t i n a1.113.. of L1112 16 c.ondiLions **cic.fir.566** by 5th bas flag
Control Transfer	
J it	L.i**ncood** [lion u I .i.0111
CALL	Trans **[el control** to al:v1111C' 14...c3tso n. elbro Insnt.fgr• the: aiLdiess o I she i L followirT Ihe CALL **k** placed nn lhe
JE1.11	**Juitip ii equLil!zern.**
L. O OPE'LOOPZ-	Loops if L'c1unlizer1 1. ' I17i iE ;1 cclndi4iOfLxh lu n ip using El val.41d litarud in rvgisIur ECX. Thu instrucLion First ducl.rernenir. ECX 1.'.4fore E CX for the branch condailm. 1n1crrupt!InteryupL it ovef Row. Traus fcr cc rol to an iL1 rup L c uLi Re.
String ()perallions	
MOVS	Movr2 word, dwuid This inR1ruction. Opc:TakeS 6L1 10E11.7:111 of a String. indc-xed reguilus ES1 and EDI. Amer each strine uperation. I he registers isre u toms' Incrum e n ted nr de.ere trse n ed to poml La El**k: ricx.i clancin of** the string.
CODS	Loud byte, word, Liwurd ni iLiir1
High-Level. Language SuPPrirt	
ENTER	**Creates Li iiiack** realm dull edit be. used Lu. the. rules DI a block-SIS LI **eturEd highlo•rd** hkngua
LEAVE	Roverses the aeition ot t he Ilrevious **ENTER.**

WA 10.8 *cantinued*

High-Level Language Support *conumeed*	
BOUND	Chuck army holinds Vertlies ttt3l the value in clitand 1 is within, lower and Upper Thu limits nru in Iwo *asliact:nt* memory locations referenced by opet and 2. AR interrupt occurs if 1.1u2 visiu.:2 is cl ue hounds. This instruction is used to check an array index.
Flag Control	
STC	Set Carty flag.
LAHF	Load A teOster from klgS- C4 Ti CS SE, 7F, AF, PF. find Cl bits into A register,
Segment Register	
LDS	Load pointer into D sc ncnt ru.eir41411". SySE ern .00(51.1
TILT	Holt.
LOCK	Asserts a hold on Shared rne.rilory Su thrdL Lhv Pentium hHs c.xclusi v2 I he. instruction that immediately Inflows the LOCK.
ESC	PIDCUNSCU C:)(1 enNiall escape. An escape code that indicates the suuxedins inNtructions arc to ht uatcuotcd by a numeric coprocessor that sUpperts hij2h-pruciiiiun hiLL!gur rind IIcFatin poen i caICSO I.
WAIT	Vir' nit until 1"11..:SYli negArt:d. kluspends PCULSi.LAM progam execution the proce:.:scir detect:, that the. 'RI :Ny pin is inmctivc, indicating that the nunicric 4oproce5;sor has linished eN.ceu Lion .
Protection	
	Stort. global dt.scriptar t
	Load sepnwra li mit. LCPLEIS a MUT"ST142C1110.4i Ngistcr with a .s.2gment limit. Vcri ry segincat fnr rue dinsiw ri Ling.
Cache Management	
INVE)	Flushet.; the internal cache. memory.
*BENVD	V1u511eS the internal cache ineinury aitur wi i Li ii i dirt!: limn Ina mcmory.
ItCVLPC,	invalidates s translation lookaside buffer (TLEI:112nlry.

to processor design would:avoid complex instructions such as ENTER and might prtxluoe a more efficient implementation with ;I Noqucncu elf simpler instructions.

Meinury NIanagetruent

Another set of specialized instructions deals with rricriitiry segnieniiition, These are privileged instructions that can only be executed from the opt:x..3111Th sp... tem. They allow loch and global segment tables (called descriptor tables) to be loaded and read. and for the privilege level or **lo he** checked and altered.

The special instructions for dealing v ilII the on-chip troche were dibous6cd in Chapter 4.

Condition Caries

We have trientiotte.ci thal. condition E.xules are bits in special registers that may be sct by certain operations and used in cOrniitional branch instructions. These conditions arc by arithmetic and compare operations. The compare operation in

most languages subtracts two operands, as does a subtract operation, The difference is that a compare operation only sets condition codes, whereas a subtract operation also stores the result of the subtraction in the destination operand.

Table 10.9 lists the condition codes used on the Pentium. Each condition, or combinations of these conditions, can be tested for a conditional jump. Table 10.10 shows the combinations of conditions for which conditional jump opcodes have been defined.

Several interesting observations can be made about this list, First, we may wish to test two operands to determine if one number is bigger than another. But this will depend on whether the numbers are signed or unsigned. For example. the 8-hit number 1111 Elll is bigger than 00000000 if the two numbers are interpreted as unsigned integers 1255 > but is less if they are considered as twos complement numbers (-- 1 0). Many assembly languages therefore introduce two s ets of terms to distinguish the two cases: If we are comparing two numbers as signed integers, we use the terms *Ic.iry than* and *greater than: if* we are comparing them as unsigned integers, we use the terms *beloiv* and *above.*

A second observation concerns the complexity of comparing signed integers. A signed result is greater than or equal to zero if (1) the sign bit is zero and there is no overflow (S = 0 AND 0 = 0). or (2) the sign hit is one and there is an overt'oe, A study of Figure 9.41 should convince you that the conditions tested for the various signed operations are appropriate (see Problem 10.14

Pentium MMX Instructions

In 1996, Intel introduced MMX technology int() its Pentium product line MMX is set of highly optimized instructions for multimedia tasks..l'here are *57* new instructions that treat data in a SIM D (single-instruction. multiple-data) fashion, which makes it possible to perform the same operation, such as addition or multiplication. on multiple data elements at once. hitch instruction typically takes a single clock cycle to execute. For the proper application. these fast parallel operations can yield a speedup of two to eight times over comparable algorithms that do not use the MMX instructions [AlK [961.

Table 10.9 Pcmium Condition Codes

Status Bit	Name	Description
	Carry	Indicates carrying or borrow mg into the leftinost hit posiiion Iollowtrig an **ki nthrnetic** operation, Also modified hr some of the shift and rotate oplobiets.
P	Parity	Parity or the result Or an aril h **!TIE VIC Or** lope opr'raIiuin. t ininalLeS even parity: Oindicatem othi parity.
A	Auxiliary **carry**	Represents carrying or hurrow•inti between half-bytes of an 8 ' hit arithmetic or logic operation using the Al. register.
	7./.115	1ndicaics ihat the result of an arithmetic or logic operation is. O.
S	Sign	Indicate the sign of the result of **'elfi nrithmgpc or** logic operatton
O	Overflow	indicates ariih ine tic owerflo,alter ;In addition or subtraction,

Table. 10.10 Pentium Condhlitmq for C'ormlitional Jump and Insrructions.

Symbol	Copridition Tested	Comment
A.	(7 1 AND	Abcrye; not below or equal (greater than, tirsined)
All:, NB. NC		Above or equal; not below (pea tel than or equ l. unsigni.•:(1): not carry
B. NAE. C	C .1	ItcIiw; not rl bOVE {ir equal (less than. unsi!ned): car ry
DE NA	C-1 OR Z-1	1312](1.41. ul CLIL1a1; 110L Fl Ur equal, LlnyirtEd)
L. Z		Equal: taro piped or unsigned)
0. NLE	[{S.1 AND 0.1) OR (SCI AND 0-0)3 AND 1 .7,=01	trcatcr than; not Jc4s titnn or .•,:(31.131 (signet)
GE. NL	(S-J AND 0 1) OR (S-l} AND 0 • 0)	ClreaLur ! hall (IT L'CIUal: not ICriki than (signal)
L. NGE	(S=1 AND 0 (1) OR (S=I AN]) (1=1)	Less than.; :sot gf L&2:1 than or equal (signed)
LE. NG	(S-L AND 0=01 OR (5=U AND 0.:.1) OR (Z-1,	9101r1 clr IAILLHI; nol greNiET than Nig•rwc•11.
	Z.=0	Not quall riot 2eto (sisne:ct or unsip.ned)
N(.)	0=0	N n (WC rilow
	S-0	Vol Rigel (nco
ICY. PO	P=1)	Not parity, parity odd
O	0=1	OVe alms:
P	P=L	Pnrityl parity cViall
	S-1	Sign i nr2 12a Live)

The focus of rkil MN is mithimedia programming, Video 4i nd audio data arc 1 ypically composed of large arrays of small data typcs. such as 8 or lib hits. whereas conventional instructions are tailored to operate on 32- or (,4-bit data. liere are some examples: In graphics. and video, a single scene consists of an array of pixels, and there arc 8 bits for e?ich pixel or 8 hill for each pixel color component (red, green, blue). Typical audio samples are quantized using 1•6 hits. Por some 3f) graphics aEgorithms. 32 bits are common for basic data types. 'PL) provide for parallel operation on these data 1E10 hs, three new data types are defined in MMX. Each gala type is 454 bits in [mall and consists of multiple srriller **each of which holds a** fixed-point integer. The types are as follows:

- Packet byte: Eight bytes packed into one 64-bit quantity
- Packed word: Four 16-bit words packed ink) 64 hits
- Packed doubtewordi Two 32-hit di aublewords packed into M hits

Table 10.11 lists the MMX instruclion set- Mosi of the instructions involve parallel operation on bytes, words, or douhiewords. I.LaT ox;irriple, I he P.SLI.Av

A pi r4I. or picture elennist, is the smallest element or a digital image lisac can he assigned a Li iry level. Equiykilently, u pixy] is an individual dpi in 1k (34)r rT)Hirj K NpTcsCrrta

Table 1.0,1. I MMX I ristr action Set

Category	111%truetiou	
Arithmetic	OD [IS. W. DI	Parallel add Cif packed four tC.hlt wort's, cit. two 32-hit douhlewords, with wiiip2TLY:111d
	PAL) D5 Ig, WI	Add with sritura Lion
	DDCS	Add tinNigned with Kit Lira tics!'
	PSUB [FL W. DI	Subtract will, wrapround
	PSI.II3S \VJ	51.31-iiracl with 2..aLtiration
	WJ	Stibtraei unNiglied w:th saturabon
	1[3]101.J1..HW	Pai.0c1 multiply cif lour 1 6-bit ..0.5:ds. with hi0-oriler 16 hits of -2-hit chosen
	PMULLW	Parallel mull 15e riiur t i pied 1.6..hiE words, with low-L5eder 1 h bits of 32 hit rgwult choser.
	DDWD	1111.1.1 li ply of ft[1] itgncd 16-hit rd:: adtl iouctlicr p;i1TS .41 32-bit result.;
(.43111r011.!.[4] 45n	PCMPEQ W, f)]	...oriip,rk: for ocfu LS] res'.311 is ID INk CH I S ir L BW (11 r1S. if FaL
	licriAPC.T [FL W, DI	Parallel 4_40inpare. For r:aser thou: re24111 is mask or is iC Lz Lk! Or 1.ti false
Conversion	PACKL SW13	Pack. wordhi into bytLs. inisign.ed
	PACKSS [W13, IDWI	Pack wordE into hyln...11:6 doubloNords into words. with signed. saLu rat i 0E1
	I)L...I N:Pt I< I T I B W, Wiz. DQ]	Pantile! unpack t inwrIcawed mem.e.) hi 17 ordcr Or d(111blewords from ItINIX rofLisi.c r
	PUNFCKL [IRV, WD, Dol	Parallcl unpark tinLerlealeed inergc) low-Di-din- bytes. words. (II ordk rro.rn TO8.iSter
Logical	PA N D	hi misu lagifial AND
	PNION	6a hi i bihkisci logical AND NOT
	Pnk	biiwise: logical OP
	PXOR	CR
Shift	PSLL [W, D: QI	Parallel loeical left shift DI packed words, doublewords. Or <I Li il dword by amount spixi.L'ikW in NIX ri3g.;ister ar immediate value
	1 S.R.L [W. D, Q]	Panillel logical right Nhifi of packed 4.'..ords, dmillie word R, or quadword
	PSRA 1W, D]	Parallel arithu,ci.it right shift of packed wordr, dou.hlewords. quadword
Data Transfer	Ni() [D, Qi	Movc closibleword or cll.i dword L?Frosrr hDetX rcgistc.t
Slate Mgt	Fmms	Empty Nal X slate (ernrty FP rogisiors tag hi's.'
N4:1e: L.: an ins Main:1. Apr; itidi.mod in h rackets.	1.111..1 Oo daca Lyixs [byte (15	in (WI, di:.11b1C:word ILL I. d types

performs a left logical shift separately on catch the four words in the packed word operand; the PA.1)01-linsIstiction Lakes packed byte operands as input and performs parallel additions on each byte position independently to produce a packed byte output.

One 41111YILLU I ft: MLin: 0f the new instruction set is the introduction of saturation arithnietic. With ordinar,' unsigned arithmetic, when am operation overflows (i.e., a carry out of the most significant bit). the extra bit is truncated. This is referred lo aS wraparound, hezotin ibe effect of the truncation can be. for example, to produce an addition result that is smaller than the two input operands. Considcr the addition of the two words, in hexadecimal, F000h and 300(1h. '1 he sum would be expressed as

```
 F000h = 1111 0003 OHO OCOO
+3000h =  0.011 0',:i00 000 0000
         10310 •3C0 00C3 '0000 = 2000h
```

If the two numbers represented image intensity. then I he resell of the addition is to make ihe combination of Iwo dark shades turn out to be lighter. 'This is typically not what is intended. With satunition arithmetic, if addition results in overflow or subtraction results in underf[ow. the result is set to the largest or smallest value representable. For the preceding example, with saturation arithmetic, we have

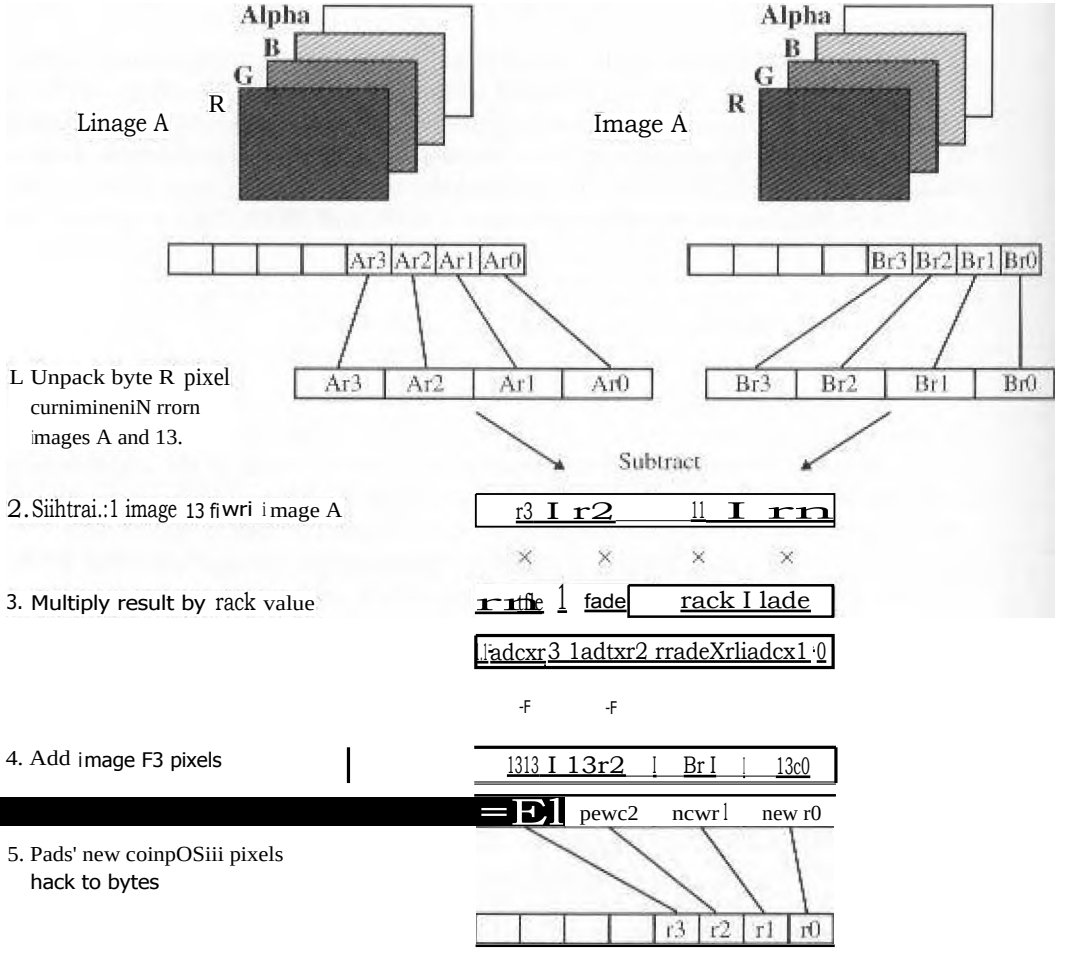
```
 FM.]Oh - =111 0303 000 000
+30C h -  0311 0303 00 0000
         1'0310 030 000 0000
          1 111 =111 :111  =111 = FFIPM
```

To provide. a feel for the use of MMX instructions, we look at an example, taken from ITELE971. A common video application is the fade-out. fade-in effect, in which one scene gradually dissolves into anol **hc.r. Two** images are combined with a weighlcd average:

$$\text{Result_pixel} = \text{fade} + \text{B_pixel} \times (1 - \text{fade})$$

This calcul;ition is performed on each pixel position in A and B. If a series of video frames is produced while gradually changing the fade value from 1 lo U (scded appropriately for an g-hit integer), the result is lo inde from image A to image B.

Figure 10,10 shows the sequence of stop, required for one set of pixels. The g-bit pixel components arc converted to 16-bit elements to accommodale the MMX 16-bit multiply capability. If these images use 640 480 resolution, and the dissolve technique uses all 255 possible value.; of the fade value, then the total number of instructions executed using M NIX is 535 million. The same calculation, performed without the IvIMX instructions. requires 1.4 billion instructions IINTE98].



MMXcrtcle serquence verforining this operation:

```
pmn          i1kr(1.7, I mri7     : m•ci ow 1111117
mo.srq             t'ad wal       ;l oad Caclu k altie repii.... oxI 4 iiincin
mcryl        rnm41, tMaleA        :I i.rad 4 red pixelconsrxiaciiss   i magi: A
inovd        intu], itrut&c.a           Tut plxct czancxynerns rrorn. image. B
punp.:khhx.  MEW), (01117         lunpadi. 4 riv:im 10 i6 hits
pun pckhlw mot', min?             :unpack 4 pixels      hiss
Emullw       mmo, rnm I           :sul.rtracE i mi4e E  Crum IJii igi A
             mn)(5,               ;multiply the 7.ubtracL re ull by radr
padddw       r um(, nim t         !Add requIL lo image B
puckumwb m int). mni7             ;pack        resulti [xick 1r, by tem
```

Figure 111.10 Image Corop-usihng *on (:7olor* Plane Represmiarion [PELI3V7 I

Table 10.12 PowOCPC **Operation** TypOsitwitli ExamplQs of Typical Operations)

Instruction	Description
	Bronch Oriented
h	Line° udisionai branch.
hl	Branch lo **iHrgcl address and place effective addrc.srt of imtructicia Col lovvin2** the branch into the Link Reuiiiter
bc	Branch conditional on Count Register a adi'or on bit in Condalion Register
st	System calkto invoke an ope:raring system si;:rviee
trap	**Compare two operands F›nd invoke system trap handler** if siwitiecl conditions are met
	I otuii.Slure
	Load word and zero extend to]eft: update source register
Id	**doublcworci**
Inve:	**Load** multiple word; load **consecutive** words into contiguous registers from the targei **rup.istl2r** through general-pin pose **fCt**i3ter 31.
	Load a string of bytes into registers beginning with target reiziaer:.4 **hytn: per register:** wrap around from register 31 so regisler
	Integer Arithmetic
add	Add **unelLtrItS** 011w1..1 tegisbLES **Eldtt phLee register**
;ubf	**SlullYLI'Lla** contents c.ir two registers and place in third register
	Multiply low-order contents 351F I WO **regiSLCTS a nd** place 64-hit product ill. third register
diva	**Divide 64-bit contents of L wcr rop.istvrs** kith] 1}laCC Erl tiLEOLieut **in** third register •
	Logical and Shift
crap	C:ornpnrc **two oFicrands and** sc:t Vour condildcm **hits in the spetifiea LOndiLi{n** register **field.**
crk:Ind	**CondiLiDn** register AND: t**wo bits of the** Condition Register are ANDc:d and I he re.still placed in **one** C)r **Ihe two** hit positions
and	AND contents of two regislers and plea: in third register
canal	COunt number of consecutive II **hits** starting ni bii ,eTi, in sl.}1.11TE **register** aJicl **c(Fura** ilt declination register
	Rot ale Icri double WOTC **I re AND** wilh mark, and **storc itt d s tinatiun** register
sly	Shift left hits in source register and store in destination register
	Routing Point
114	1.ciad **nurnbur from memory, LC:divert (L 64-** format, **and SLOTO** in flonLing-pciint **rqirdcl**
add	**Add** L'132)1.2.111.3 **ts.vo** registers and place lit third register
fmadd	Multiply contents of two registers. **add thG** ecintenis of a third, **and plow result** ill fourth register
fc:nirn	**(:cinipare twee Flogging_paint i5perHnds and set cortdirion** hits
	Csiehe
dcbf	Darn cache block flush; p-erfOrErflOOkUp in cache on spedficd target **acklress** perform flushing operation
ichi	**Imo-mum cache block invalidate**

PowerPC Operation Types

The PowcrE)C provides a large collection of operation types. Table 10.12 lists the t!,'pcs and gives exampLes of each. Several features are. worth noting.

Iiranch-Oriented inwtruction

The PosscrPC supports the usual unconditional and conditional branch cup. bilities. Conditional branch instructions test a single bi I of the condition register for true. false. or don't care and the con Len Is of the count register for zero, nonzero, or don't care. **Thu:s,** there are nine separate conditions that can be defined for the con• ditional branch instruction. if the count register is tested for zero or nonz .ero,lhen it is decremented by 1 prior to the test. This is convenient far Kciting up iteration loops.

Branch instructions can also indicate that the address of the location follow- ing thi branch is to be placed in the [ink register, described in Chapter 14. This fad• itates call/return processing.

Load/Store Insiructions

hi the PowerPC architecture, only load and store instructions 4L=:wdi mcrnor, locations: arithmetic and logical instructions are performed only on registers. This is characteristic of RISC design, and i1 is explored further in Chapter 13.

There **arc** two features that characterize the different]oadistore instructium

- **DIIIR size:** Data can be transferred in units of byte, hal fword, word, or dm.- bleword, **Instruction xrc** aim) [able for loading or storing a string of bytes into or from multiple registers.
- **Sign extension; For haliwnri and word** loads, the unused bits to the left in the 64-hit destination register are either filled with zeros or with the sign bit of the loaded quantity.

10.6 ASSEMBLY LANGUAGE

A CPI:: can understand and execute machine instructions. Such instructions arc simply binary numbers stored in the computer, If a progrimmer wished to program directly in machine language. then it would be necessar y to enter the program as **binary** daia.

Consider the simple BASIC statement

N=1-h,1-1(

Suppose we wished to program this statement in machine language and to initialize 1. ***J,*** and K to 2, 3, and 4, respectively, This is shown in Figure 10,11.3. The program starts in Location 101 (hexadecimal). Memory is reserved for the four variables dart- ing at location 201. The program consists of four instructions:

1. Load the conteras or location 201 into the AC.
2. Add the contents of location 202 to the AC.
3. Add the contents of location 203 to the AC.
4. Store the contents of the AC in location 204.

This is clearly a tedious and very error-prone process.

A slight improvement is to write the program in hexadecimal rather than binary notation (Figure 10.111)- We could write the program as a series of lines. Each line contains the address of a memory location and the hexadecimal code or the binary value to he stored in that location. 'Then we need a program that will accept this input, Iranslate each line ink) binary number, and store it in the specified Location.

For more improvement, we can make use of the symbolic name or inncmonic of each insiruction. This results in the *Nymbofic proKreon* shoves. in Figure 10.11c. 24ieh line of input still reprcNents one mentory location. Each tine consists of three fields. separated by spaces. The first field contains the address of a Location. For an instruction, the second field contains Ihe three-letter symbol for the opcode. It' it is. a memory-refcrencing instruction, then a third field contains the address. To store arbitrary data in a iocation. we invent a *pseudoinsrraction* with the symbol .0/kiT. This is merely an indication that the third field on the line von[ains hexadecimai number to be stored in the location specified in Llie fivsL field.

Add res',		CO titC/IN			Address	Instruction	
101	01.3 10	0010	0000	0001	101	LDA	2(11
102	0001	(10] (1	0000	0010	102	ADD	202
103	0001	(1010	0000	01111	103	ADD	203
104	0011	0010	000(1	0100	l 04	4'I- A	204
201	0[100	0000	0000	00 LO	201	DAT	2
202	0000	0000	0000	001.]	2(12	L)A1'	3
203	0000	ocom.	noon	01(10	203	DAT	4
204	0000	0000	0000	0000	2(14	D Al-	(1

(al Binary nrnaram (1].). Symbolic program

Andros	Contenis;	i .k.thel	Operation	Operand
1(11	22111	l-01041IL	LDA	
102	1202		ADD	.1
103	1203		**ADD**	K
104	32{14		STA	
2(11	0002	1	DATA	2
202	0003	1	DATA	3
201	(1004	K	DATA	4-
204	0000	N	DATA	(I

(C) flexadoc im al progiun Id) A3 Aenibl y program

Figure 10.11 Collimation of the rorrnuia N = I + 3 +

I-or this type of input wC need a slightly more complex program, The program accepts each line of input. generalcs a binary number based on the second and third (if present) fields, and stores it in the location specified by the first field.

The use of a symbolic program makes life much easier but is still awkward. In particular, we must give an absolute address l'ur each word. ihis means [hut the. program and data can be loaded into only one place in nicmury'. and we must know that place ahead of time. Worse, suppose we wish to change the program some day by adding or deleting a line, 'Ibis **will** change I he addresses 01 all subsequent words.

A much better system, and one commonly used, is to use symbolic addresses. [his iw illusmiled in Figure 10.11d. Each line still consists of three fields. The fir5t field is still fur the address, bin a symbol is used in ',l ead or an ab7;..olute numerical address. Some lines have no address, implying that the address of that line is one more than the address of the previous line. For memory-reference instructions, the third field also contains a symbolic address.

With this last refinement, we have. **an** *a.ssembly hinguive,* Programs written in assembly language (assembly programs) are translated into machine language by ao *ca.s.embkr.* This program mum not only do tl· e symbolic ironslaiion discussed **ear.** lier, but also assign some form of memory addresses to symbolic addresses.

The development of assembly language was a major milestone in the evolution of computer technology. It was the first step to the high-level languages in use today. Although few programmers use 4s:se•mbl:,.,j language, virtually t,11 machine provide one.. They are used, if at all, for systems programs such as compilers and 1l0 routines.

10.7 RECOMMENDED READING. -0³-WAtlr"·'*;(A?'

A ittirobr.lr of ttmbooks provide good coverage of machine language and instruction •;i Lltsign. including [PATT98], [TAN EN], and [HAYE98]. The Pentium instructicin silt is covtrtd by [1311-ENOIA. The PowtrPC. instruction seL is covered iti 11.13M)41 and IWEIS941.

RRF,Y00 Bre.y. B. *The. Imel ,144-roprour•veryt.y: Ai186..M. 06, 10118641188, 802S6,* 7ffk? f7, 80486, *Porgithol, NetiMnt Po.o Peeii* 2l.h.] *Proccsx*00.8.. 4·rpal· Rivt.I.T, NJ: Prentice 1-Tall, .2000.

HAYE98 lin yes. J. ComparoT Airbil•requre *road Organi7'.060.1, 5c.00.17.4 LefithYPJ,* Ntiv York! MeOraw-Hil I. I .998.

ll` **M+** International Business Machines, Inc. *The. PowarP(.7 ArOrifec· ta m· A Sfn.c.ifieerthhq for a New Parnifi DJ· RISC Pare -,slaty,* San Francisco, CA; Morgan Kalifrnarkti. 19[)4,

PATTI/8 Patterson. D., and Hennessy . .1. Comp *i.divr CJr ertaiErtrirrt, ae,id Desivl: The I-lard' ware/Sofrware Intryffer.e.* San **Mato**[). **CA: Morgan** Kaufmann, 1998.

TANE99 Tannbiliton, A. ***Sfrffi'M ri'd C(.1.tnpreiCr*** ***FlIOCIV{3431.1 C[i(fS,*** PrEntice Hill. 1 [)99.

Wiz-1594 Weiss, +1.110 Smitk. J. **PO.** '14[1**1#40** *P*ower IC, trancisco: NIDtpn15..0 1.kt- mann, 1994,

LOA KEY TERMS, REVIEW QUESTIONS, AND PIZOBLENI

Key Terms

aectimulator	jump	procedure call
addreNs	**Unit:** aldian	procedure return
.arithinetie shift	logical shill	push
hi-endian	machine instruction	reentrant procedure
frig endian	operaoil	reverse Polish notatima
branch	ri.er.iitioi	.1.148lq=
conditional branch	packed decimal	skip
instmetion set	pop	stack

Review Questions

10.1 What are the typical elements of a machine instruction?

10.2 What types of Locations can hold source and destination operands?

10.3 if an instruction contains four addresses, what might be the purpose of each address?

10.4 List and briefly explain five important instruction set design issues.

10.5 What types of operands arc typical in machine instruction sets?

10.6 What is the relationship between the IRA character code and **i[W** packed decimal representation?

10,7 What is the difference between au it li med ie shifi it logical shift?

1.0,X Why are traiisfer of cont fell i11s1ria114111N Jlvt!111,!1,1?

10.9 List and briefly explain two conunon ways of generating the condition to be tested in a conditional branch instruction.

10.10 W hat is meant by the term *nesting of procethries1*

10.11 List three possible places for staring the return address Err a procedure return.

10.12 What is a reentrant procedure?

10.13 What is the difference between amerribly language and machine language?

10.14 IN hat is reverse Polish notation'?

1015 What is the difference between big endian and little endian?

Problems

10.1 Madly (TIN provide logic for performing arithmetic on packed decimal numbers. Although the rules for decimal arithmetic are similar to those for binary operations. the decimal results may require some corrections to the individual digits if binary Logic is used.

Consider the decimal addition of two unsigned numbers. If each number consists of N digits, then there are 4N bits in each number. The two numbers .tire to he added using a binar:i.. adder. Suggest a simple rule for correcting the resull. Perform addition in this fashion on the numbers 1698 and 1786.

10.2 The tens complement of the decimal number .5(is defined to he 10 % X. whore N is the number of decimal digits in the number. Describe the use of ten's complement representation to perform decimal subtraction, illustrate the procedure by subtracting (0326) from (0736)L.

10.3 Compare zero-, one-. two-, and three-address machines by wri ring programs to compute

X — (A — B X CND - E

for each of [lee four machines. The instructions available for use are as follows!

0 Address	1 Address	2 Address	3 Address
PUSH M POP M ADD SI..18 m1:1 Div	LOAD M STORE. M ADD M SOB M m4:i m Div rs1	MOVE (X e• Y) ADD IX e— X 1 Y) klB IX <— X V) Nen .L (X i= X x Y) D1V (X t— XI)	MOVE (X t— *V)* ADD 0.<— Y + 7) SUB (X t— Y -- 7.1 Ml IT, Pc (— V \ 7.) Dry (.5,:: .— VIZ)

10,4 Consider a hypoinefical computer with an instruction 'Jel of only two or hit in. suructions..rk first hit specifies the opcode, and [Ile remaining It sl-PeCifY one of the — 1 D-bil words of main memory. The twci insiructions are

SUBS X Subtract the ountents•of location X from the accumulator, and store the result in location X and ace umttlator.

JUMP X Place address X in die program counter.

A word in main mentor!, may **Liint** either an instruction or a binary **mher** in two complerneni notation. Demonsiniie that this i **1Saufloii** repertoire is reasonably **COrm** pieta by specifying how the following operations can be programmed:

u. Data transfer: Location X to accumulator. accumulator to location X

b. Addition: Add contents **of** location X lo accumulator

e. Conditional branch

d. Logical OR

e. 110 Operations

10.5 Many instruction sets contain the instruction NOOP. meaning. no **ON rat** ion, which has no effect 4.311 the CPU state other than increinenlingihe program counter. Fuggesr some uses of this instruction,

10.6 In Section 10.4, it was stated I hat bolt an arithmetic [eft shift and a logical led shift correspond to a multiplication by 2 when there is no overflow, and if overflow occurs. arithmetic and logical left shift operations produce different resulES, but the arithmetic left shift retains the sign of the numl rer. Demonstrate that these statements arc true for 5 bit twos complement integers.

10.7 In what way are numbers rounded using an Ihrnelic right shift (c.g., round lowarcl +co, round toward – cc., toward zero. away from Or

10.8 Suppose a stack is to be used by lite CPU lo manage procedure calls and returns. Can the program counter he eliminated by using the top of the stack as a program counter?

10.9 Appendix 10A points not that there are no stack-oriented instruclions in an instruction set if the stack is to be used only by the CPU for such purposes as procedure handling. How can the CPU use a slack for any purpose without stack-oriented instructions?

10.10 Convert the following formulas from reverse **Polish** to infix_

a. AB — C D x

b. AR; CD.: I

C. ABCDE + *X* X

d. ABCDE + + —— X -F

10.11 Convert the following formulas from infix to reverse Polish:

a. rl BIC D•E

h. {A - B) >c (C I D) •I•

c. (A x 13) + (C x D} — E

d. (A — B) x ((fC: — 13 x 12:).:1)/(;) H

10.12 Convert the ex[resskin A — B C to postfix notation using Dijkstra's algorithm. Show the 81.0pS I LIWIM2d. is the result equivalent to (A I B) - C or A + — Cr? Doe% it matter?

10.13 The Pentium architecture includes an instruction called Decimal Adjust after Addition .(DAA). DAA performs the following 5u(iiience of instructiorm

```
if HAL AND CFH) >     0F(            then
     AL c    AL 1 Er
     AF      l!

     AF  I-  C:
andif.r.
if                        = l) then
     AL 4-- AL     6'.)l.
     CF t- 1;

     AP'
enaif-
```

H'" indicates hexadecimal. AL is an S-hit register that holds the result of addition *of* two unsigned 8•bit integers. AF is a flag lict if there is a carry fp coil **hit 3** io hit 4 in the result of an addition. CF is a flag set if there is a earry front bit 7 Et1 hi! ti. F,xplain the function performed by the DAA instruction.

10.14 The Pentium Compare instruction (CMV) subiracisilui **41]urci** nperand from the destination operand; it updates the sta1us flags (C. A. 7., 5, 05 but does not alter either of the operands. The CNN instruction. may 10110w...ill by a conditional Jump (,Ice) or Set Condition (SETec) instruction. where cc vefersio one of the 1.6 conditions listed in Table M. I . Dertionstrale that the conditions toted for a signed number compk ¯-ison are correct.

10,15 Nlipsi microprocessor instruction sets include an instruction that a condition and sets destination operand if the condition is true. Examples includi! 1111 5.(oo the Pentium, the Sec on the Motorola Ma8000. and the Sound 4111 **!Fit!** NHI ikPIIH I:\

a. There are a few differences among these instructions;

- SETce and Sec operate only on a byte. whereas Scond operates on byte. word. and doubleword operands.
- SETce and Scond set the operand hi integer one if true and to zero if false. Sec sets the byte to all binary ones if 1cue and all nips if false.

What are the rela1i've advaniages and disadvantages of these differences'?

h. None of these instructions set any of the condition code flags. and thus an explicit test of the result of the instruction is required to determine its value. Discuss whether condition codes should be .sei rstili of this instruction.

e. A simple IF stalcmcnt suit, ci **II** a I I i r.N can he implemented using a numerical reprc.sentation inethoil. I hat k, 1 **1n: 1:i** li g the Boolean value manifest, as opposed LO rt ***flow viconrioiniL***' *III ' 1.I,* **W1114'11 1** L"I) roen ts the valUe.of a Boolcan expression by a point reached in **t17e 31** 0171 ; mi. A iamipiler inight implement IF a > b TI LEN with the following g0X861 **LiIt** P:

```
SUB     CZ,            |              to
        A2, 2              ccnLcri,    Of 1:DCariDn T! to register  LK
C.Cd    A2_,  A     c.cltpare con:*3a1;8      :eoister  AX  and location A
.TL.E   T=5;7       j::urp if A
=NC                 ; add = to corten     of re7Later CX
TEST J)     OUT     ; ¯iJwc  Lf  con-e=lse  of  CX  ecual
THEN

OCT
```

The result of (A B) is *a* Boolean value held in a register and 'available later on, ovtside the context of the flow of code just shown. IL is convenient to use Pg. CX this, because many of the branch and loop cipcodes have a built-in test for C.

Show an alternative implementation using jnsi ruction that saves memory and execution time. (Him; No additional new :%8n instructions arc needed. other than the SE.Teti.)

d. Now consider the high-level language statement!

(13. C) OR (D — F)

A compiler might generate the following code:

```
    MOV    TAX, ri     ; Tr,ve 2on7.ents of l•catLon E
    CAp    TAX,  r ;cmpEra ronterta of regis=er EAX 3nd lOCSI7irM.
    MOV       eJ    ; 0 represemts false
    ME     Fl         ;j1:mp if 2
    MW        1       ;J. represents false
NI  t              C
    CMr    EA,
           bH, J
    JNE    !
    ECV EH,
N2  OR     EL, EH
```

Show an alternative impleniuntaLon using the SF IL instruction that saves 111M17.17 and execution Lime.

10.16 Using the algorithm for converting iufix to past ix defined in Appendix 10A, shove it k steps involved in converting the expression of Figure 10.15 into postilx. Use a presentation similar to Figure 10-L7-

1.0.17 show the calcu[at ion of the expression in Figure 10.17 ; using a presentation sim[Ear to Figuiv [(5.

10..111 Redraw the little-endian **layout** in Figure 10,18 so that the bytes appear as numbered in the big-endian layout. That is, show memory in 64-bit rows, with the bytes listed left to right. top to bottom.

10.19 For the following data structures, draw the big-endian and little-endian layouts, using the Lortnat of Figure 11.i. 8, Ind comment on the results.

a. strucz

```
    doub-E        {10x1=1213=41515171 8
sL;
```

b. struct.

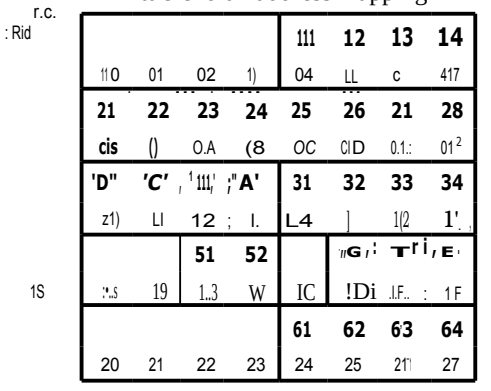
```
          ;       /./Oxl:1213:4
    int j         Zi0x]..161 71.8
s2
```

9tx".)E7t.

```
    short- L;     .r./Cx=112
    short 7::;
    short k;
    shark:  L;    ./.12x171a
```

111.20 The PowerPC architecture specification does not dictate: now a processor should implement little.cndian mode. It specifies only the view of memory a processor must have when operating in little-endian mode. When converting a data structure from big endian to little endian, processors are free to implement a true byte-swapping

Little-endian address mapping



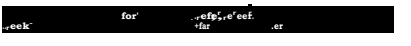
Figure; 1111,12 }'o Fort'[' Lit
Structure s in Memory

mucliunisni crc to use lornc sort of an address rrtudilleation mechanism, Current Plyomrt ³C. pruccbsurs arc al] ddatilt big-endian niatthincs and tisk.' **#Iddre 'is** in 'hi 1rcza data as little-endiall.

CI:Pnsidcr the slru lurt s dcinicd in Figure W.18. Thu layout in 11110 lower-right por- **HI ilk LA** ¹ **flw** lig.tire shows are structure: s suesi by Cho prock:ssor. In fact, it si meture. s llll little Lmlian its in memory is she in Figure 10.12. 1..% pialir [kw mapping chat is involvd. 1.11.scriln.iir1 easy way ll implement the map- k•ind discuss Ow elfcciivkInG..SS of this aprprnah,

10.21 Write a small program to determine the endianness °I:machine and report the results. Run the program on a comput...:x available to you and turn in the outpul.

APPENDIX 10A STACKS :



Stacks

A *,stack is an* ordered set of vic nwnts, only onc. of which can be accessed at a ti me. The point of access is called the top of the stack. The number of elements in the stack. or ire.ngth of the stack, is variable., items may only be El dded to or deleted from tlic is or the stack - 14 or this re..i.ison, a stack is also known as a *petvhchnvn Esq* or a *!am-in-lint-out (LIFO)*

Figure 10.13 shows the basic stack operations. We begin at some point in time when the stack contains some number of elements. A PUSH operation append., i pric new item to Lhc 1op or the stack. A Pop operal ion removes thu top item from the stack. In both cases, the top of the stack moves accordingly. Binary operations. which require two operands (e.g., multiply, divide. iLdd, subtract), is L1 top two stack items as operands, pop bath items, ;ind i1n.,11 the resuEt hack onto the stack. Unary operations, which require only one operand (e.g., logical NOT), use the item on the lop of the stack. All of these operations are summarized in Table 1. 13,

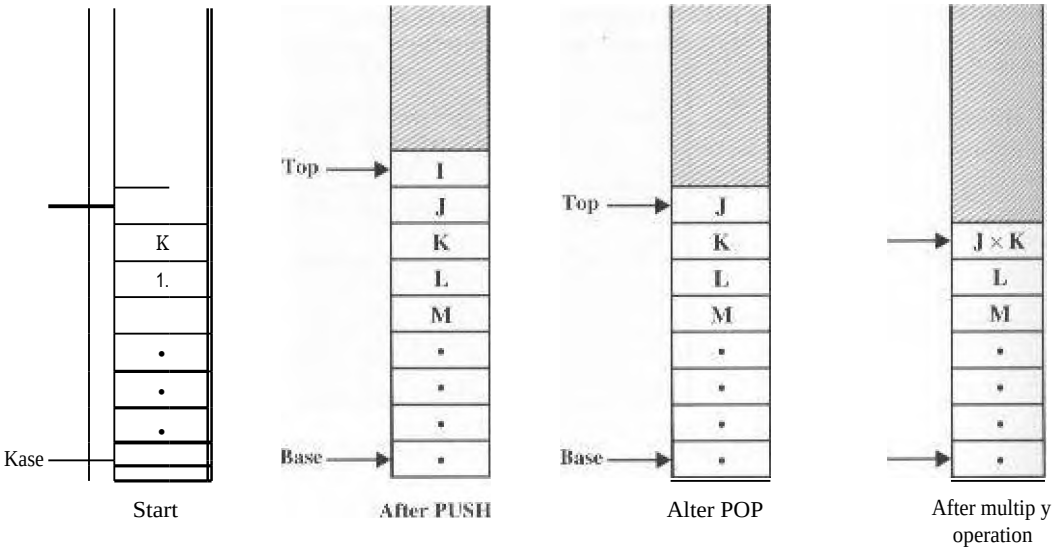


Figure 10.13 Basic Stack Operation

Stack Implementation

The stack is a useful structure to provide as part of a CPU implementation, One use, discussed in Section 10,4, is to manage procedure calls and returns. Stacks may also he useful to the programmer. An example of this is expression evaluation, discussed later in this section.

The implementation of a slack depends in part on its potential uses. if it is desired co make black operalions available to the programmer, then the instructiotl set will include stack-oriented operations, including PUSH, POP, and operations that use the top one or two stack elements as operands. Because all of these operations refer to a unique location, namely the top of the stack, the address of the operand or operands is implicit and need not he included in the instruction. HICSe are the zero-address instructions referred to in Section i0.1..

If the stack mechanism is to be used only by the CPU, for such purposes as procedure handling, then there will not he explicit stack-oriented insirmliOns in the

Table 10.13 Stack Oriented Operations

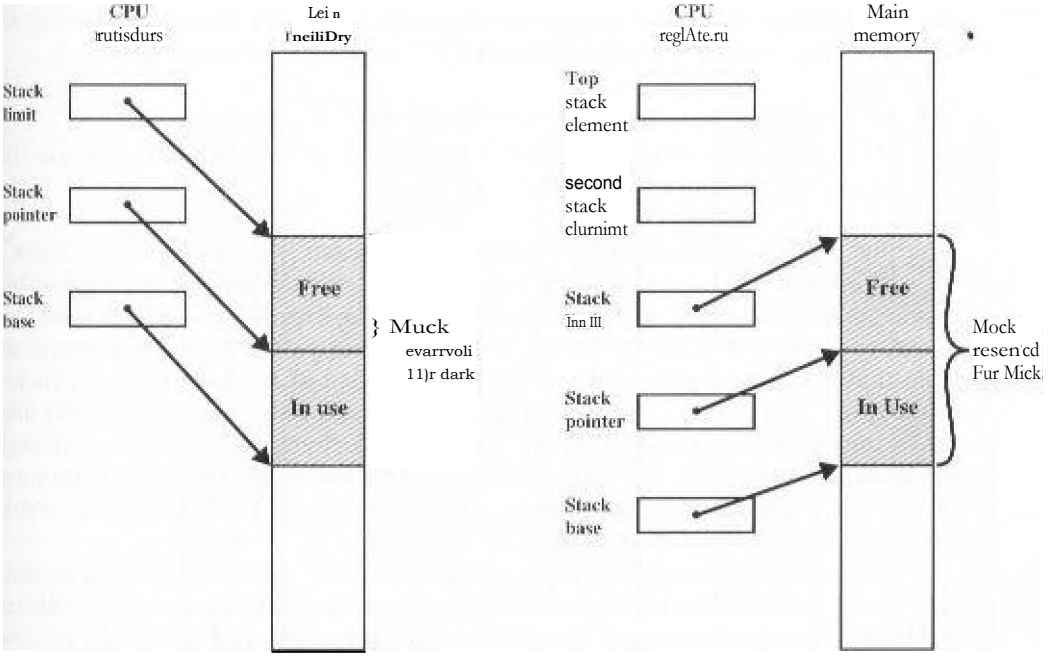
PUSH	Appetitt a new u142mtnt in [he top of **Ekle' Staek,**
POP	Delete the top el on en! iif [11
Unary operation	Perform operation on Lop elurnenL of ,auck. Rr..place Sep element **with** result.
13iliar1 operation	Perform operation on limp t**wo ctrrseral.; of** stack. I]cle[4:. Lop t**wo elements uF slid,** Place result or **cip riaLicln** on top of tack.

instruction set. In either case, the implementation of a stack requires that there he some sat of locations used to store the stack elements. A typical approach k illustrated in Figure 10.14a. A contiguous. block of locations is nerved in main mem-Ory (or virtual memory) *for* the stack. Niost of the time the block is partially filled with stack elements and the remainder is available for stack growth.

Three addresses are needed for proper operation. and these are often stored in CPU registers:

- Stuck pointer; Contains the address of thc top 0r the stack. If an item is appended to or deleted from the stack, the pointer is incremented or decremented to contain the address of the new top of the stack.
- Stack base: Contains the address of the hoLtom locaiion in the reserved Nock. If an attempt is made to POP when the stack is empty, an error is reported.
- Stack limit: Contains the address of the other end of the reserved block. alternpI is made to **PUSH** when 1hc block ix fully utili?..ed for the stack, an error is reported.

Troditionally, and on most machines today. the base of the stack is at the high-address end of the. reserved stack block, and the **limit** is at the low-address end. Thus, the stack grows from higher addresses to lower addresses.



(41 All 41C s.tmek in memory

(11.11 **two tap clemellix in** registur6

Figure 10,14 Typical Stack Organizations

To speed up stack operations, the top two stack elements are often stored in registers, at shown in Figure 10.14b. in this case, the stuck pointer contains the address of the third element of the stack.

Expression Evaluation

Niallic;.rnalieal formulas are usually expressed in what is known as *infix* notation. hi this Corm, a binary operation appears between the operands (e.g., a h). - or NM. plea expressions, parentheses are used to determine the order 0r evaluation of expressions. For example, a — (h x c) will yield a different result than (a b) c. To minimize the use of parentheses, operations have an implied precedence- Gen. erally, multiplication takes precedence over addition, so that a L. bxe is eiluiva• lent to a -I. (h x c).

An alternative technique is known as reverse *Pofi,ii,* or postfix, notation, In this notation, the operator follows its Iwo operands. For example,

a+ h	becomes a b —
a -F (h .x	becomes abcx
(a +b)x c	becomes a b.—

Note that, regardless of lhc complexity of an expression, no parentheses are required when using reverse Polish.

The advantage of postfix notation is that an expression in this form is easiiy evaluated using a stack. An expression in postfix notation is scanned from left to right. For each elemern of the expression, the following rules. arc

L If the element is a variable or constant, push it onto the stack.

2. if the element is an operator, pop the top two items of the stack, perform the operation, and push the restili.

After the entire expression has been scanned, the result is on the top of the slack.

The simplicity of this algorithm mikes it a convenient one for evaluating expressions. Accordingly, nviny compilers will take an expression in a high-Level language. convert it to postfix notation. and then generi Le the machine instructions Crum that notation. **Figure** 10,15 shows the sequence of machine instruction s for evaluating f = b)? (e -F d c) using stack-oriented instructions. The figure also shows the use of one-address and two-address instructions. Note that, even though the .stack-oriented rules were not used in the [ass two cases. the postfix notation served as a guide for generating the machine instructions. The sequence of events for the stack program is shown. in Figure 10.16,

The process of converting an infix expression to a postfix expression is itself most easily accomplished using a stack. The following algorithm is due.to Dijkstra [DIJK63]. The infix expression is scanned from left to right, and the post fix expres- **sion** is developed and output during the scan. The slops are as follows:

L Examine the next element in. the input.

/ If it is an operand, output it.

	Stack	General Registers	Single Register
	Push a Push b Subtract Push Push d Push Multiply Add Divide Pc»	Load a Subtract RI. Load R2. d hluUipiy R2; e Add k2. Divide R1, R2 Store Rh	Multiply c Add Sum:. Load a Subtract h Divide f Slum (
Number or instructions		7	
Memory access	l.0op + 6d	7 up — (id	iup—lid

Figure **10.15** Comparison of Three P`rogranis to Calculate f = b) (c • d

3. If it is an opening parenthesis, push it onto the stack.
4. tf it is an op.E.:ni tor, awn
   - If the top of the stack is an opening parenthesis, then push the operator.
   - If 4 has higher priority than the top i..)C thc stack (multiply .and diviii2 have higher priority than add :.)rd $.ubtract)_ then push the operator.
   - Else, pop operation from stack to output, and repeat Eltep 4.

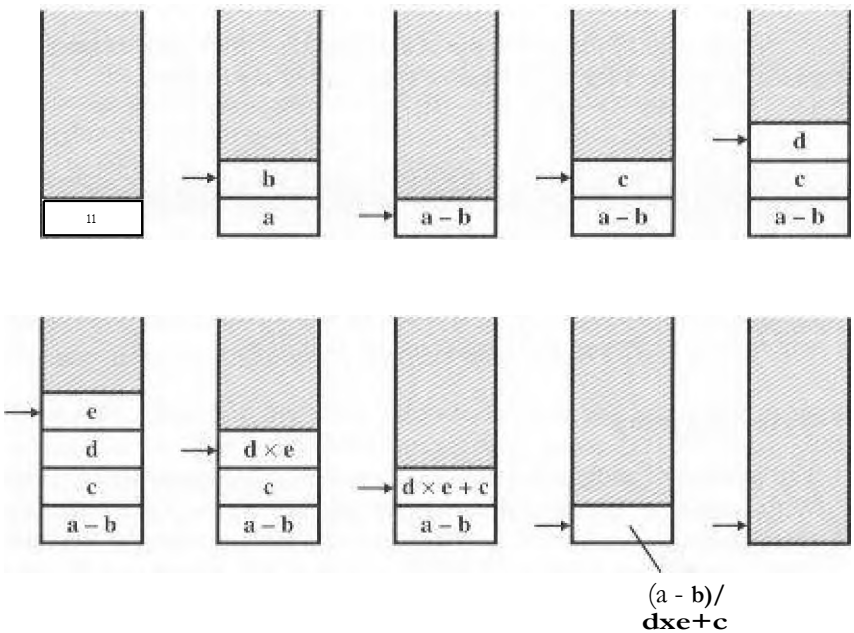


Figure 16.16 Use *or sloe':* to Compute f = (a • 13); (d e c).

Input	Output	Stack (top on right)
A+BxC+ (D - x F	empty	empty
+ C (D - E)	A	empty
x + (I) - E) X F	A	
x h)xF	AR	
C I (D-E)x.1	An	+x
I (D + E) x F	ABC	+
(D + E) x F	AB(.	
D + E) x F	A C x I	+
+ XF	A C x D	4. (
E) X F	A li C x -D	
) xF	ABCx - DE	-
x	A I-3 (7 x - D E +	
	ABC/ +DE-	
empty	A BC + F. + F	x
empty	ABCx+DF+F x	empty.

Figure 10.17 Conversion of an Exprcinion from Infix to Post rix Notation

5. If it is a closing parenthesis, pop operators to the output until an opening parenthesis is encountered. Pop and discard the opening parenthesis.
6. If there is more input, go to step 1.
7. **If** there is no more input, unstack the remaining operands.

Figure 10.17 illustrates the use of this algorithm. This example should give the reader some feel for the power of stack-based algorithms.

APPENDIX 10B LITTLE-, BIG- AND BI-ENDIAN

An annoying and curious phenomenon relates to how the byk.s within a word and the bits within a byte are both referenced and represented. We 14.)ok **first at** the problem of byte ordering, and Then consider that of **hits.**

Byte Ordering

The concept **of** endianness was first discussed in the literature by Cohen [COHE811, With respect to bytes, endianness has to do with the byte ordering of multibyte scalar values. The issue is best introduced with an example. Suppose we have the 32-bit hexadecimal value 173419678 and that it is stored **in** a 32-bit word in byte-addressable memory at b!, 'te location 1.84. 'nu value consists of four bytes, with the least significant byte containing **the** value 78 and the most significant byte containing the value l2. There are two ways to store this value:

AtItiros.;	Value
	12
185	34
E86	_56
t;`4[7]	

Address.	Value_
1.85	56
186	34
187	12

The mapping on the left stores the most significant byte in ihe lowest numerical byte addre ss; this is known as big endian and is equivalen1 Lo Ihe left-to-righttprder of writing in Western culture languages. The mapping on the right stores the least significant byte in the lowest numerical byte address; this is known as little endian and is reminiscent of the right-to-left order of arithmetic operations in arithmetic **Por** a given multibyte scalar value, big endian and little 4; ndion are byte-reversed mappings of each other.

The concept of endianness arises when it is necessary to treat a multiple-bylc en Lily as *a sins k* data item with a single address, even though it is composed of smaller addressable units. Some machines, such as the. Intel 80x86, Pentium, VAX. and Alpha. are tittle-endian machines, whereas others, such as the IBM System 3701340, thi.. Motorola 680x0, Sun SPARC, and most RISC machines, are big endian. This prestitui iwobleins when data are transferred from a machine of one endian type to the mile!, and when a programmer attempts to manipulate individual bytes or bits within a muitibvie scalar.

'File property of endionness does not c.xicnd beyond ɯn individual data unit. In any machine, aggregates such as files, data structures. and arrays are cornpoted of multiple data units, each with endianness. Thus, conversion of a block of memory from onc style or uicthihricss to the other rquires knowledge of the data structure.

figure 10.18 iliustrates how eadianness determines addressing and byte order, The C structure at the top contains a number of data types. The memory layoul in the lower left results from compilation of Ihat structure: fora big-endian machine, and that in the lower right lot a little-endian mach in each case, memory is depicted as a series of 64-bit rows. For the big endian case, memory typically is laid out left to right. top to bottom. whereas for the little-endian case. memory typically is laid out right to Left. topic bottom, Note alai these layouts are arbilrary_ sehenie could use either Lit to right or right to left within a row; this is a matter of depiction, not memory assignment. In fact, in looking at programmer manuals for a variety rriiehines, a bewildering collection of depictions is to be found, even within the same manual.

We can make several iihservations about this data structure:

- Each data item has the same address in both scheme s_ Poi- example, the address of the doubleworLi with hexadecimal value 2122232425262728 is 08.
- Within any given multibyte scalar value, the ordering of bytivs in the le- endian structure is the reverse of that for the big-erldimi structure.

'TELE terms 132' ,. *endian* and *endicrrr* come' tram **Part T, Chap ILr** 4 of **a StiwilCi** **Te7 •** *di*. They rofer (43 a religious **WAY** between. two Aro ups. one that hmaks ergs et the big *end* and the other that Frre4ts ego **HI I IN' Litt I** end,

```
st=zi.
   inz      a;      .!.:Cm11:17.    Lc:                    poor
   ins
                .b;          C.K2122_232.S_2326_2725
   0 718..f*            / Cx3132_"233.'
   cAar        71         A                                    irray
    'fLoriz             //0x5152
   int      fi          /71:1x6161_63;34
```

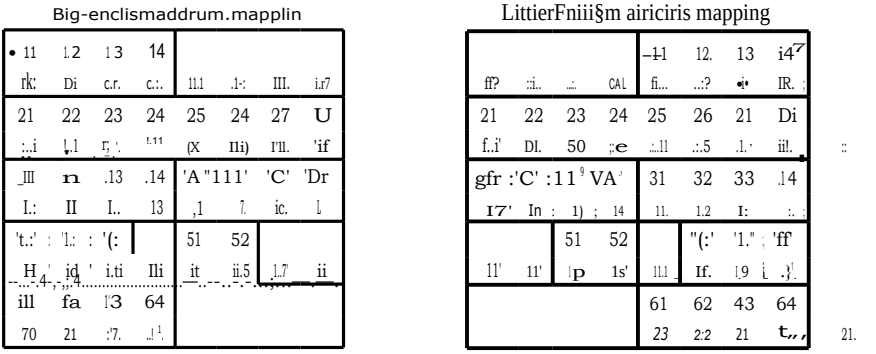


Figure 10.18 Example C Data Structurc and Its Encliffli Maps

■ Endianness does not affect the ordering of data item* within a structure. Thus. the four-character word c exhibits byte reversal. but the seven-character byte array d does **not.** Hence, the address of each individual element of d is the same in both structures.

The effect of endianhess i perfiaTis more dcnionstra1cd whigt we vive memory as a vertical array of bytes, as shown in Figure 10,19.

There is no general consensus as to which is the superior style of endianness.' The following points favor the big-endian style:

- Character-string sorting: A big-endian processor is faster in.comparing integraligncd character strings; the integer ALLT can compare multiple bytes in
- fleeirnal/IRA diurrapi; All values can be printed left to right withou1 causing confusion.
- Consistent order: Big-endian processors store their integers and character strings in the same order (most significant byte conies first},

The f011owing points favor the little-endian style:

* A big-endian processor has to perrorm addil ion when it convurts a 32-fait irate• ger ziddress to a 1h-bit integer address, to use the least significant bytes.

Ehe prophet revered by both groups in Clic- Endisn Wars *4sc (Mr Ay r's T11 Iv er lead Lhis* **La ray.** " All **FI.411CVC.IN 4halt** break their Eggs at the convenieriL Not much help!

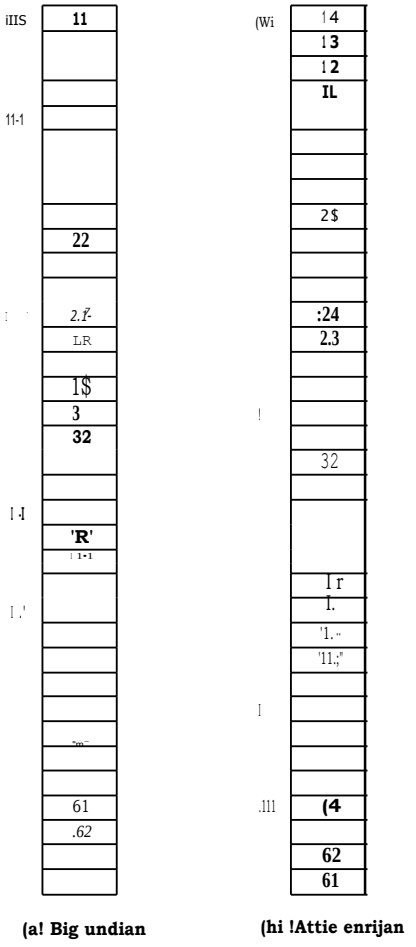


Figure PLO Anfither View of Figure 1(1.1.8

0 It is easier to perform higher-precision arithmetic with the little-endian style; you don't have to find the least-significant byte and move backward_

The differences are minor and the choke of endian style is often more a matter of accommodating previous machines. than ;.inytiling else.

The PowerPC is a biacintli;in processor that supports both big-ebdin and little-endian modes_ 'f'hu bi-endian architecture enables software developers to choose either mode when migrating operating sy'sl.cros and applications from other machines. The operating system cst4iblishes the endian mode in which processes execute. °nee a mode is selected, all subsequent memory loads arid stores are determined by the memory-addressing model (Cf that mode. To support this hardware feature, 2 bits arc maintained in the machine state register (MSR) maintained by the operating system as part of the process state. Ono bit specifies the endian

mode in which the kernel **runs;** the other specifies the processor's current operating,, mode. Thus. mode can he changed on a per-process basis.

Bit Ordering

In ordering the bits within a byte, we are immediately faced with two quemions:

1. Do you count the Cirst bit as bit zero or as bit one?
2.. Do you assign the lowest bit number to the byie's. Last significant bit Oink endian) or lea the bytes most significant bit (big endian)?

These questions are not answered in the same way on all machines. Indeed, on some machines, the answers are differen1 in different circumstances. Furthermore. the choice or big- or little-endian bit ordering within a byte is not always consistunt with big- or little-endian ordering of bytes within a rnultibyie scalar, The programmer needs to be concerned with these issues when miniraliaOng individual bits.

Another **area** of concern is when data are transmitted over a bit-serial Line. When an individual byte is transmitted. does the system transmit the must significant bit first or the least significant bit first? The designer must Tnake certain that incoming bits Arc handled properly- **For a discussion** of this issue, see 1JAME901.

CHAPTER 11

INSTRUCTION SETS: ADDRESSING MODES AND FORMATS

11.4 Addressing

Inimeaaie Addresing
• irect Addressing .;
:Indirect Acidi•oWng;:,
Register Addressing .",••••-
••11cgisteer Indirect Addresing;
Displacement Address*
Stack AddressitIg

11.2 Pentium and PotserPC Addreming Modes

. Pcntinni Addie:ssing Modes
.rowerPC Addressing Modes

11.3 Instruction FormaW

I nstruf.7tion Length
••110eLitiorl of Bits
ifiabic-Length Instructions

11.4 Pealing" and PoxverPC Instruction fiStritats

[claim 'Instruction Formats
Piya.erPC. instruction T.'orrnats

11.5 Recommended Reading

11,4 Key Terms, Review Questions. and Prol iiermi

Kiz.y Terms
Rvicw OtiOrli ons
ProbieMS

KEY POINTS

- An operand reference in n instruction either contains the actual value of the operand (irrurnedia re) or fl N.:fereEtce to the address of the operand. A wide variety of addressing modes is used in various instruction sets. These include. direct (operand address is in ad(1reti% field), indirect {address field points lo that con tAinq the (ye nd address), tegister, register indirect, .irid Var- ious forms of displacement, in which a register value is added to an address value to produce the operand address.
- instruction formai defines the layout fields in the instruction. Insmiction (01 trO1 design is complex undertaking, including such ecuv.iderili ions as instruction length, fixed or variable length, 131A mbel ;issillned to opcode and each operand reference, and how undressing mode is determined.

n Clapter 10, we focused on whar an instruction set does. Specifically, we examined the types of operands and operations shall n L;iy be specified by machine instructions, This chapter kurris to the. question of how to specify the operands 11141 operations of instruel ions. Two issues arise. First, how is the address of an operand specified, and second, how are the bits of an instruction organized toddle the operand addresses and operation of Thal instruction?

11.1 ADDRESSING

The address field or fields in a typical instruction format are relatively small, Wr.. would to he Able to reference a large range *of* locations in main memory or. for some systems, virtual memory. To achieve this objective, a variety of addressim techniques has been employed, They all involve 53ome trade-off between address range andlor addressing flexibility, on the one hand, and the number of memory rd. L.renccs and/or the complexity of address calculation, on the other. **In** this section, we examine the most common addressing techniques:

- Immediate
- Direct
- Indirect
- Resister
- Register indirect
- Displacement
- Stack

These modes are illustrated in Figure 11.1. In this section, we use the following notation:

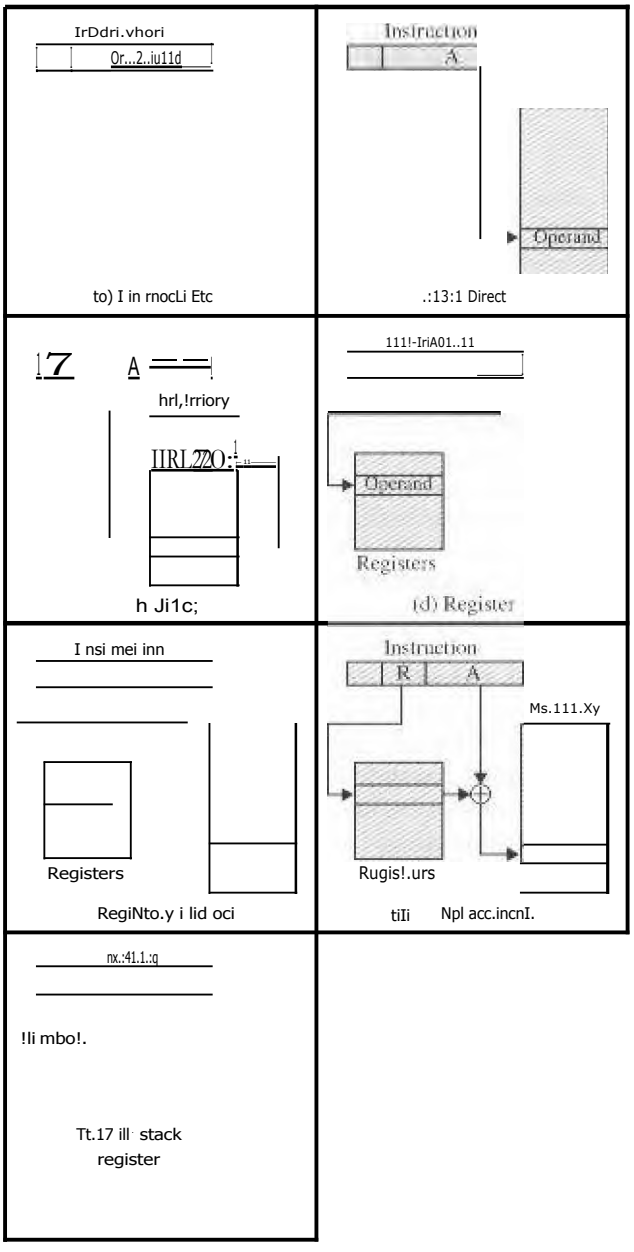


Figure 11..1 Ad d riLssing Moth:1i

A = contents of an address field in the instruction
R = contents of an address field in the instruction that rcfc.rs to a register
EA — Eictual (cfrcutive) nidrcss *of* thc location containing the referenced operand
(X) = contents of memory location X or register X

Table 11.1 indicates the address calculation performed for each addressing mode.

Table 11.1 Basic Addressing Mocks

Mt do	Algnrit li In	Principal Advantage	Principal Disathantage
Itnraediace	Operand , A	No memory ref..,=Nricc:	LimitEci opera rid magrkitude
Dirt	= A	Simplc	LindEEd address space
Indirect	EA -: (A)	Laise address space	Lapie noemory rcfGrc Rcc5
Register	EA = R	No .munory rcANnec:	ruitIrtss space
Rovistcr indirect	= (R)	84,1113-esh sprtee.	Urea raernoty reference
Dip aCerat n L	EA =A+ (R)	Flu xibi lity	Complexity
Stack	EA top of stack	No memory reference	Lintiitd applicahili ty

Before beginning this discussion, two comments need to he made. First, virtu, ally all computer architectures provide more than one of these addressing modes. The question arises 4$. to how the control unit Can determine which address mode is being used in a particular instruction. Several approaches are taken. Often, different opcodes will use different addressing modes. Also, one or more his in the instruction formal um tis.ud i1 a *mode field.* The valuo of the mode field deterniincs which addreissi lig mode is Lo he used.

The. second continent concerns the interpretation of the effective address (EA). In a system without virtual memory, the *effective address* will he either a main memory address or a register, In a virtual memory S'y'Siern., the effective address is a virtual address or a rLgister. The actual mapping to a physical address is a function of the paging mechanism and is invisible to the programmer,

Immediate Addressing

The simplest form of addressing is immediate addressing, in which the operand is actually present in the instruction:

$$\text{OPERAND} = \text{A}$$

This mode *can* be used to define and use constants or set initial values of variables_ Typically. the number will be stored in twos complement form; the leftmost hit of the operand Field is used as a sign bil. When the operand is loaded into a data register, the sign bit is extended to the left to the full data word size.

The advantage of immediate addressing is [hal no memory reference other Limn the instruction *fetch* is required Lu obtain the operand, thus saving one niernory of cache cycle in the instruction cycle, The disadvantage is that the size of the number is restricted to the size of the address field, which, in most instruction scts, is small compared with the word length.

Direct Addressing

A very simple form of c.iclresising ɑ red .idiressing_ in which the address field contains the effective address of the operand:

$$\text{EA} = \text{A}$$

The technique was common in earlier generations Of computers bul is not common on contemporary architectures..lt requires only one memory reference 4i nd no special calculation. The obvious, limitation is that it provides only a limited address space.

Indirect Addressing

With direct addressing. the length of the address. field is usually less than the word length, thus Hi rifling the address range. One solution is to have lht 4Kiducss field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is hnown as *indirect addressing:*

$$EA = (A)$$

As defined earlier, *the* parentheses are to he interpreted as meaning *contents of* The obvious advantage or this approach is that for a word length of N. an address spice of is now available. The disadvantage is Thal instruction execution requires two memory references to fetch the operand= one to geL its address and a second to 2et its value.

Although the number oi' words that can he addressed is now equal to the number of different effective addresses that may be referenced at any one time is limited to where. *K* is the length of the address field. Typically, this is not a burdensome .ICS triction, and it can be an asset. In a virtual memory environment, all the effective address locations can be confined to page 0 of any process. Because the address field of an instruction is sinali. it wilt naturally produce low-numbered direct addresses, which would appear in page 0. (The only restriction is that the page size must be greater than or equal to 2k,) When a process is active, there wi]l be repeated references to page O. causing it to remain in real memory. Thus, an indirect memory reference will involve, at most. one page faith rather than two.

A rarely used variant of indirect addressing is multilevel or cascaded indirect addressing;

$$EA — (... (A) ...)$$

In this caliC, one bit of a full-word address is an indirect [lag (I). if the I bit is 0, then the word contains the EA. IF the I bit is 1, then another level of indirection is invoked. There does not appear to *he* any particular advantage to this approach, and its disadvantage is that three or more memory references could be required to fetch an operand.

Register Addressing

Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address:

$$EA=R$$

Typically, an address field that references registers wi]l have from 3 to 5 bits, so that a total of from S to 32 general-purpose registers can he referenced.

The advantages of register addressing are that (1) only a small address field is needed in the instruction. and (2) no memory' references are required. As was discussed in Chapter 4, the memory access time for a register internal to the CPU is much less than that for a main memory address, The .disadvantage of register addressing is that the address space is very limited.

If register addressing is heavily used in an instruction set. this implies that the CPU registers will he heavily used. Because of the severely limited number of rep isters (compared with main memory locations), their use in this fashion makes sense only if they are employed efficiently. If every operand is brought into a register from main memory. operated on once, and then returned to main memory, then a waste• ful intermediate step has been added. If, instead, the operand in a register remains in use for multiple operations, then a real savings is achieved. An example is the intermediate result in a calculation. In particular. suppose that the algorithm for **twos** complement multiplication were to he implemented in son ware. The location labeled A in the flowchart (Figure 9.12) is referenced many times and should he implemented in a register **rather** than a main memory location.

It is up to the programmer to decide which values should remain in registers and which should he stored in main memory. Most modern CPUs employ multiple general-purpose registers. placing a burden for efficient execution on the assembly-language programmer (e.g., compiler writer).

Register Indirect Addressing

Just as register addressing is analogous to direct addressing, register indirect addressing is analogous to indirect addressing. In both cases, the only difference is whether the address field refers to a memory location tar a register. Thus, for register indirect address,

$$EA = (R)$$

The advantages and limitations of register indirect addressing are basically the same as for indirect addressing. In both cases, the address space limitation (limited range of addresses).of the address field is overcome by having that field refer to a word-length location containing an address. In addition, register indirect addressing uses one less memory reference than indirect addressing.

Displacement Addressing

A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing. It is known by a variety of names depending on the context of its use. but the basic mechanism is the same. We will refer to this as *dip placement addressing:*

$$EA = A + (R)$$

Displacement addressing requires that the instruction have two address fields. at least one of which is explicit. The value contained in one address field (value = A) is used directly. The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

We will describe three of the most common uses of displacement addressing:

- Relative addressing
- Base-register addressing
- Indexing

Relative Addressing

I;or relative addressing, the implicitly referenced register is the program counter (PC). That is. the current instruction address is added to the address field to produce the EA. Typically, the address field is treated as a twos complement number for this operation. Thus. the effective address is a displacement relative to the address of the instruction.

Relative addressing exploits the concept of locality that was discussed in Chapters 4 and 8. If most memory references arc relatively near to the instruction being executed, then the use of relative addressing saves address hits in the instruction.

Base-Register Addreming

For base-register addressing, the interpretation is the following: The referenced register contains a memory address, and the address field contains a displacement (usually an unsigned integer representation) from that address. The register reference may be explicit or implicit.

Base-register addressing also exploits the locality of memory references. It is a convenient means of implementing segmentation, which was discussed in Chapter 8. In some implementations, a single segment-base register is employed and is used implicitly. In others. the programmer may choose a register to hold the base address of a segment, and the instruction must reference it explicitly. In this latter case, if the length of the address field is K and the number of possible registers is N, then one instruction can reference any one of N areas of 2' words.

Indexing

For indexing, the interpretation is typically the following: The address field references a main memory address, and the referenced register contains a positive displacement from that address. Note that this usage is just the opposite of the interpretation for base-register addressing. Of course, it is more than just a matter of user interpretation. Because the address field k considered to be a memory address in indexing. it generally contains more bits than an address field in a comparable base-register instruction. Also, we shall see that there are some refinements to indexing that would not be as useful in Ihe base-register context, Nevertheless, the method of calculating the EA is the same for both base-register addressing and indexing, and in both cases the register reference is sometimes explicit and sometimes implicit (for different CPU I ypes).

An important use of indexing is to provide an efficient mechanism for performing iterative operations. Consider, for example. a list 0r numbers stored starting at location A. Suppose that we would like to add I to each element on the list. We need to fetch each value. add I to it, and store it back. The sequence of effective addresses that we need is A, A -h 1. A + 2, up to the last location on the

list. With indexing, this is easily done. The value A is stored in the instruction's address field. and the chosen register, called an *index register.* is initialind to 0. After each operation. the index register is incremented by I.

Because index registers are commonly used for such iterative !asks, it is typical that there is a need to increment or decrement the index register after each reference to it. Because this is such a common operation, some systems will automatically do this as part of the same instruction cycle. This is known as *auroindolug.* if certain registers are devoted exclusively to indexing, then autoindexing can be invoked implicitly and automatically. if general-purpose registers are used, the autoindex operation may need to be signaled by a hit in the instruction. Autoindexing using increment can be depicted as follows:

$$EA \; A + (R)$$
$$(R) \quad (R) \; 1$$

In some machines, both indirect addressing and indexing are provided, and it is possible to employ both in the same instruction. There are two possibilities: The indexing is performed either before or after the indirection.

If indexing is performed after the. indirection, it is termed *postindexing:*

$$EA = (A) \; I \; (R)$$

First, the contents of the address field are used to access a memory location containing a direct address. This address is then indexed by **the** register value. This technique is useful for accessing one of a number of blocks of data of a fixed format. For example, it was described in Chapter S that the operating system needs to employ a process control block for each process. The operations performed are the same regardless of which block is being manipulated. Thus, the addresses in the instruc• lions that reference the block could point to a location (value — A) containing a variable pointer to the start of a process control block, The index register contains the displacement within the block.

With *preindexing* the indexing is performed before the indirection:

$$EA = (A \quad .(R))$$

An address is calculated as with simple indexing. In this ease, however, the calculated address contains not the operand, but the address of the operand. An example of the use of this technique- is to construct a multiway branch **table. At** *a* particular point in a program. there may be a branch to one of a number of locations depending on conditions, A table of addresses can be set up starting at location *A.* By indexing into this table, **the** required location can be found.

Typically, an instruction set will not include both preindexing and postindexing.

Stack Addressing

The final addressing mode that we consider is stack addressing. As defined in Appendix 9A, a stack is a linear array of locations, It is sometimes referred to as a *pushdown list or lust-in-first-our queue. The* **stack** is a reserved block of locations. Items are appended to the top of the stack so that, at any given time, the block is

partially *f*illed- Associated with the stack is a pointer whose value the address of the top of the slack. Alternatively, the top two elements of the stack may he in CPU registers, in which case the stack pointer references the third element of the stack (Figure 10.14b). The stack pointer is maintained in a register, Thus. references to stack locations in memory are in fact registei indirect addresses.

The stack mode of 4iddrusr;ing is a form of implied addressing. The machine instructions need not include a me.mory referenec but implicitly operate on the top of the stack,

11.2 pENTtirm AND pOVVWC ADDRESSING iyipps

Pentium Addressing Modes

Recall from Figure 8.21 that the Pentium address translation mechanism produces an address, called a virtual or effective addrem,, 1114 4 is an offset into a segment. The sum of the starting address of the segment and the effective address produces a linear address. If paging is being used, this linear address must pass through a page-translation mechanism to produce 4i. physical address. in what follows, we ignore this last seep, because it is transparent to the instruction set and to the programmer,

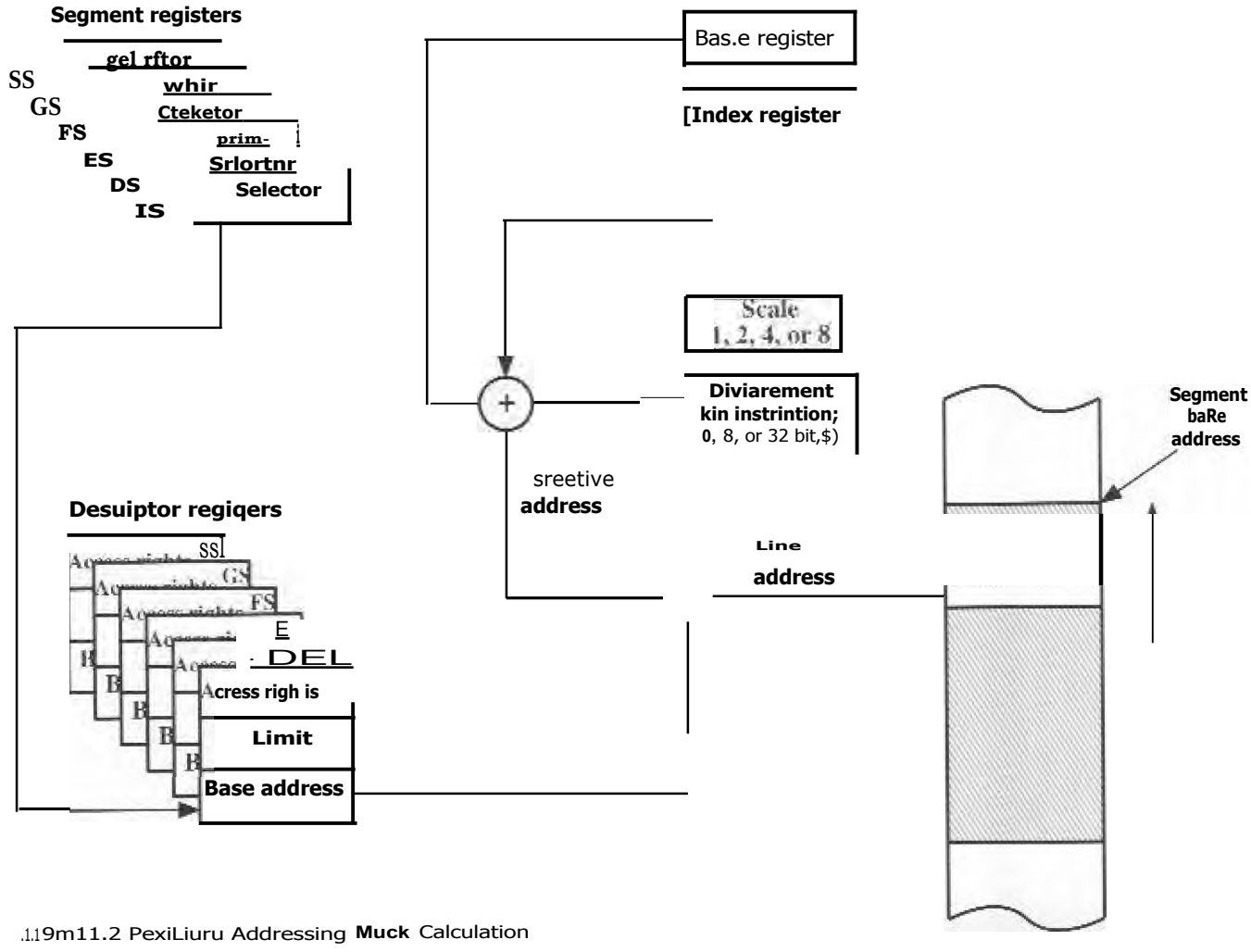
The Pentium is equipped with a variety of addressing modes intended to a]-low the efficient execution of high-level languages. Figure 1.1,2 indicates the logic involved. The segment register determines the segment that is the subject of the reference. There are six segment registers; the one being used fora particular reference depends on the context of execution and the instruction. Each segment 12g-isIer holds the starting address of the corresponding segment. Associated with each user-visible segment register is a segment descriptor register (not programmer visible), which records the access rights for the segment as well as the starting address and limit (Length) of the segment. In addition, there are two registers that may be used in constructing an address: the base register and the index register.

'Fable 11,2 lists the 12 Pentium addressing modes. Lei us consider each of these in turn.

For the immediate mode. the operand is included in the instruction. The operand can be a byte, word. or doubleword of data,

For **register operand mode, the** operand is located in a register. For gencra I instructions, such as data transfer, arithmetic, and logical instructions, the operand can be one of the 32-bit general registers (14AX, 17113X. E(-:X, EDX, ESI. **EDI. ESP, ERP),** one **of** the 16-bit general registers AX, BX, CX, DX. SI In SEJ. HP), or one of the 8-bit general registers **(AH, BH, CH, DH,** AL., BL. CL, DL). For floating-point operations, 64 - hit operands are formed by using two 32-bit registers as a pair. There are also some instructions that reference the segment registers (CS. DS, ES. SS. FS, GS).

The remaining addressing modes reference locations in memory. The memory location must be specified in terms of the segrneni containing the location and the offset from the beginning of the segment. In some cases, a segment is specified explicitly in others, the segment is specified by simple rules that assign a segment *by* default.



.1.19m11.2 PexiLiuru Addressing **Muck** Calculation

Table 11.2 Pentium II Addressing Modes

Mode	0 perwid A !withal
Frnmcdia[t	
RelOsler c•ixerand	LA — R
Dtsplacerncnt	LA = (SR) i •
Bk1 se.	I A = (SR) + (FS)
Base with clisplacurrirm	LA = (SR) + (13) - A
SO 11241 inMi. with displaceinem	LA= (SR) + (1) y. S — A
11.1 with incIEN Hnd displacQ.mant	LA= rsR) + 05)— A
Rost with scaleck i3lcieif and displaecTrizilL	LA = (SR) (1) x 5 — (FS} + A
Relarive.	LA (PC) .1- A

LA = Linear act:inns.
(X') = conceras
!.zracca rogislcr
k'i ' prt5graro counter
A – coraents of an addru.N field in the :Instruilzion
• - 1·QiNtor
• = hme](_ iti| ur
I = in tick 1 4$L.ri
S = SUN irle act')1.

In the displacement triode, the 4..Pperand · off et (the effective address of Figure 11.2) is contained as part of the instruction as an 16-, or n-bit displacement. With segmenlation• all addresses in instructions refer merely to an offset in a segment. The displacement addressing mode is found on few machines because, as mentioned earlier, it leads to long instructions. In the case of the Pentium, the disptaeernen I value can be as long as 32 bits, making for a 6-byte instruction. Displacement addressing can he useful for referencing global variables.

The remaining addressing modes are indirect, in the sense I h.i t the address portion of the instruction tells the processor where to Look to find the address. The base amide specifies that one of the 8-, 16-, or 32-bit registers contains the effective address. This is equivalent to whAll. we have referred to as register indirect addressing.

In the base with displacement mode, the instruction includes a displacement I he added to a base register, which may be any of the general-purpose registers. Examples of uses of this mode include. the following;

- l.hled by a compiler to point to the start of a local variable area. For example, the base register could point to the beginning of a stack frame, which contains the local variables for the corresponding procedure.
- Used lo index into an array when the element size is not 1, 2, **4,** or 8 bytes and which therefore cannot be indexed using an index register, in this case, the displacement points to the beginning of the array, and the base register holds the results of a calculation to determine the offset to a specific element within the array.
- Used to access a field of a record. The base register points to the beginning of the record. while the displacement is an offset to the field.

In the scaled index with displacement mode, the instruction includes a displacement to he added to a register, in this case called an index register. The index register may he any of the general-purpose registers except the one called ESP, which is generally used for stack processing. In calculating the effective address, the contents of the index register are multiplied by a scaling factor of 2, 4. or 8. and then added to a displacement. This mode is very convenient for indexing arrays. A scaling factor of 2 can he used for an array of to-hit integers. A scaling factor of 4 can he used for 32-bit integers or floating-point numbers. Finally, a scaling factor of 8 can be used for an array of double-precision floating-point numbers.

The base with index and displacement **mode** sums the contents of the base register, the index register, and a displacement to form the effective address. Again. the base register can he any general-purpose register and the index register can be any general-purpose.register except PSP. As an example, this addressing mode could be used for accessing a local array on a stack frame. This mode can also he used to support a two-dimensional array; in this case, the displacement points to the beginning of the array, and each register handles one dimension of the array.

The **based scaled index with displacement mode sums** the contents of the index register multiplied by a scaling factor. the contents of the base register, and the displacement. This is useful if an array is stored in a stack frame in this case, the array elements would be 2, 4, or 8 bytes each in length, This mode also provides efficient indexing of a two-dimensional array when the array elements are 2, 4. or 8 bytes in length.

Finally. **relative addressing** can be used in transfer-of-control instructions. A displacement is added to the value of the program counter, which points to the next instruction. **In** this case, the displacement is treated as a signed byte, word or doubleword value, and that value either increases or decreases the address in the rifogram counter.

PowerPC Addressing Modes

In common with most RISC machines, and unlike the Pentium and most CISC machines, the PowerPC uses a simple and relatively straightforward set of addressing modes: As Table 11.1 indicates, these modes are conveniently classified with respect to the type of instruction.

Load/Store Architecture

The PowerPC provides two alternative addressing modes for load/store instructions (Figure 11.3). With indirect addressing. the instruction includes a 16-bit displacement to be added to a base register, which may be. any of the general-purpose registers. In addition, the instruction may specify that the newly computed effective address is to he fed back to the base register, updating the current contents. The update option is useful for progressive indexing of arrays in loops,

The other addressing technique for loadistore instructions is **indirect indexed addressing.** In this case, the instruction references a base register and an index register, both of which may be any of the general-purpose registers. effective

Table 11.3 PowerPC' AdLiti.ssing Modes

Mode	Algorithm
	LoudiStore Addressing
Intli mct	**EA - i$R) - D**
Indirect indexed	**=11114.1.- (IR)**
	Brandt Addresing
Absolute	**EA = I**
Relative	**P.A = I Pt.') -** I
Indirect	**EA** _
	Fixed-Point Computation
Reg ter	**= Grit**
I mmediate	**Operand =** I
	ilomting-Point Computation
Reeister	**CA = FPR**

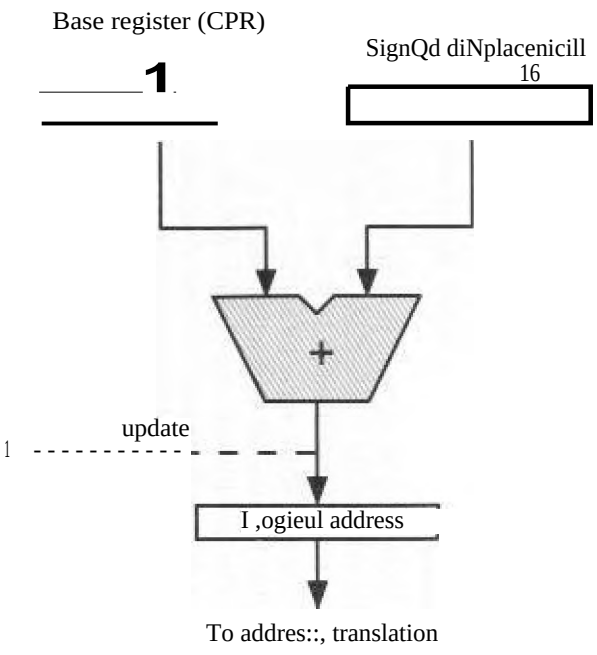
- addres6
t X1 = Lonients of X
BR = hase register
ilt = index rqi,ister
lit - link or count register
GIJR = **genet ill-purpose riLpisler**
= Floating-point i'clUslat
= diVlseemen1
I - immediatowitie
I't ' program ctiouthr

address is the sum of the contents of these two registers. Again, the update option causes **the base register to be updated to the new** effective address.

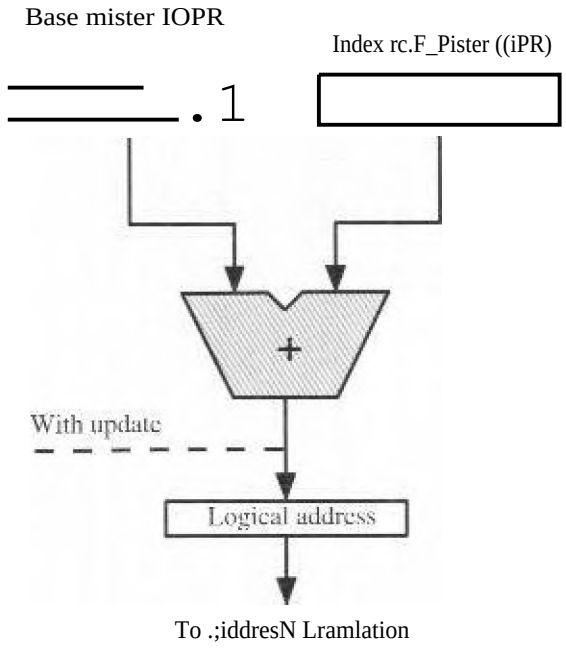
Branch Addressing

Three branch addressing modes are provided. When absolute addressing is used with unconditional branch instructions, the effective address of the next instruction is derived from a 24-hit immediate value within the instruction. The 24-hit **value is extended to a 32-bit value by adding two** zeros 10 its **least significant end (this is permissible because all instructions must occur on 32-bit boundaries) and sign extending.** For conditional **branch instructions, the effective address of the** next **instruction is derived from a** I 6-hit **immediate value within the instruction. The 16-bit value is extended to a 32-hit value by adding** two zeros to its least signif-icant **end and sign extending.**

With relative addressing. the 24-bit immediate value (unconditional branch instructions) or 14-bit immediate value (conditional branch instructions) **is extended as before. resulting** value **is then added to the program counter to define a location relative to the current instruction. The other conditional branch addressing mode is indirect addressing.** This mode **obtains the effective** address **of the** next **instruction from either the link register** or **the count register. Note that in this case**



) Indirect adrcssing



(b) Indirect indexed addressing

Figure 11.3 rCde RO7mia&ddressing Modes

the count register is used to hold the address for a branch instruction. This register may also be used to hold's count for tooping, as explained earlier.

Arithmetic I nstri 'dims

For integer arithmetic, al] operands must he contained either in regisicrs or as part of the instruction. With register addressing, a source or destination operand is specified as one of the general-purpose registers, With immediate addressing, a source operand appears as a I6-bit signed quantity in the instruction.

For floating-point arithmetic, all operands are in floating-point registers that is, only register aidie:y,higrw **taxed,**

3 INSTRUCTION FORMATS

An instruction format defines the la taut of the bits of an instruction, in terms of its constituent parts. An instruction format must include an opeode and, implicitly or explicitly. zero or more operands. Each explicit operand is referenced using one cif 1he addressing modes described in Section The format must. implicitly or explicitly, indicate1he addressing mode for each operand. IC)1' CLAtME insiruerion sets, more than one instruction Format is used.

The design of an instruction format is a complex art. and an amazing variety 0r designs have been implemented. We examine the key design issues, looking briefly at souse designs lo illustrate points, and then we examine the Pentium and PowerPC solutions in &Li i I.

Instruction Length

The most basic design issue to be faced is the instruction format length. This decision affects, and is affected by, **InCinrilry wire,** memory organization. bus structure. CPC complexity, and CPU speed. This decision determines the richness and flexibility of the machine as seen by the assembly-language programmer.

The most obvious trade-oft here is between the desire for a powerful instruction repertoire and a need to save space. Programmers want more opeodes, more operands, mdre addressing modes, and greater address raluze. More opeodcs and more operands make life **Casier** for I he programmer, because shorter programs can **he written** to accomplish given [asks. Similarly. more addressing modes give the programrru greater flexibility in implementing certain funclions, such as [able manipulations and multiple-way branching. And, of course. with the increase in main memory size and the increasing use of virtual memory. programmers want to he able to address larger memory ranges. All of these **things (oprodes,** operands. addressing **modes.** address range) require bits and push in the direclion or longer instruction lengths. But longer irril ruction length may be wasteful. A 64-bit instruclion occupies twice the space of a 32-bit instruction but is **probably** Jess than twice as useful.

Beyond this basic trade-off, there are other considerations. Either the instruclion length should be equal to the memory-transfer length (in a bus system. data-

bus length) or one should he a multiple of the other. Otherwise, we will not get an integral number Of instructions during a letch cvele. A related consideration is the memory transfer rate. **This** null.; has not kept up with increases in processor speed. Accordingly, memory can become a bottleneck if the processor can execute instructi ons faster than it can fetch them. One solution to this problem is to use cache meal. ory (see Section 4,3): another is Lo use shorter instructions. Thus, 16-bit instructions CM' be fetched at twice the rate of 32-bit instructions hul probably can be executed less than twice as fast.

A seemingly mundane but nevertheless important feature is that ihe instruction length should he 21 Mal Liple of the. character length, which is usually S bits, and of the length of fixed-point numbers, To see this, we **need** to make use of that unfortunately ill-defined word, woe/ [FRA183]..rhe weird length of memory is. in some Sense, the "natural ¯ unit of organiiiition. The size of a word usually del ermines the size of fixed-point numbers (usually the two are equal). Word site is also typically equal to, or at least integrally related lo. the memory transfer size. Because a common form of data is character data. we would like a word to store an integral 1lLl Mher 01' characters, Otherwise, there are wasted bits in each word when storing multiple characters, or a character will have to straddle a word boundary. The importance of this point is such thuti 1.1-1M, when it introduced the Systern1360 and wanted to employ S-bit rharactors, made the wreaching decision to move from the 345-bit architect Lire of the scientific members of the 70017000 series Lo a 32-bit architecture.

Allocation of Bits

We've looked r.I some of the factors that go into deciding t he length of the instruction format. An equally difficult issue is how to allocate the bits in that format. The trade-offs here are complex.

For a given instruction length, there is clearly a trade-off between the number of °Nodes and the pt awer of the addressing capability. More opcodes obviously mean more bits in the opcode field. For >in instruct ion format of a given length, this reduces the number of hits available for addressing, There is one interesting refinement 10 this trade-off, and that is the use of variable-lengih opeodes. In this approach, there ig a minimum opcode length hut, for some opcodes additional operations may be specified by using addii ional **hits in the** instruction.. For a fixed. length instruction. **his** leaves fewer bits for addressing. Thus. ads feature is used for those insiructions that require fewer operands andior les!, powerful addressing.

The following interrelated factors go into determining the use of the addressing bits:

- **Number of addressing modes;** Sometimes an addressing mode can be indicated implicitly. For example, certain opctide might always call for inile:';ing. **In** other cases, the addressing modes must be explicit. **and** one or more mode.bits will he needed.
- **Number of operands:** We have seen dial fewer addresses can make for longer, more awkward programs (e.g., Figure 103). Typical instruct ions on today's machines provide for two operands. Each operand address in the instruction might require its own mode indicator, or the use of a mode indicator could he limited to lust one of tfic addrcsg fields.

- Register versus memory: A machine must have registers so that data can be brought into the CPU for processing. With a single user-visible register (usually called the accumulator), one operand address is implicit and consumes no instruction bits. However, single-register programming is awkward and requires many instructions. Even with multiple registers, only a few bits are needed to specify the register. The more that registers can he used for operand references. the fewer bits are needed. A number of studies indicate that a total of 8 to 32 user-visible registers is desirable [LUND77, HUCK83]. Most contemporary architectures have at least 12 registers.
- **Number of register sets:** Most contemporary machines have one set of general-purpose registers, with typically 32 or more registers in the set. These registers can he used to store data and can be used to store addresses for displacement addressing. Some architectures, including that of the Pentium. have a collection of two or more specialized sets (such as data and displacement).. one advantage of this latter approach is that, for a fixed number of registers, a functional split requires fewer bits to be used in the instruction, For example, with two sets of eight registers, only 3 bits are required to identify a register: the opcode implicitly will determine which set of registers is being referenced.
- **Address range:** For addresses that reference memory, the range of addresses that can be referenced is related to the number of address hits. Because this imposes a severe limitation, direct addressing is rarely used. With displacement addressing, the range is opened up to the length of the address register. Even so, it is still convenient to allow rather large displacements from the resistel address, which requires a relatively large number of address hits in the instruction.
- **Address granularity;** For addresses that reference memory rather than registers, another factor is the granularity of addressing. In a system with 16- or 32-bit words, an address can reference a word or a byte at the designer's choice. Byte addressing is convenient for character manipulation but requires, for a fiXed-size memory, more address bits.

Thus. the designer is faced with a host of factors to consider and balance. Hoy,. critical the various choices are is not clear. As an example, we cite one study [CRAG79] that compared various instruction format approaches, including the use of a stack, general-purpose registers, an accumulator, and only memory-to-register approaches. Using a consistent set of assumptions, no significant difference in code space or execution time was observed.

Let us briefly look at how two historical machine designs balance these various factors.

PDP-S

One of the simplest instruction designs for a general-purpose computer was for the PDP-8 [BELL78b]. The PUP-8 uses 12-bit instructions and operates on 12-hit words. There is a single general-purpose register, the accumulator.

Despite the limitations of this design, the addressing is quite flexible. Each memory reference consists of 7 hits plus two 1-bit modifiers_ 'The memory is divided into fixed-length pages of 2^7 =, J.28 words each. Address calculation is based on

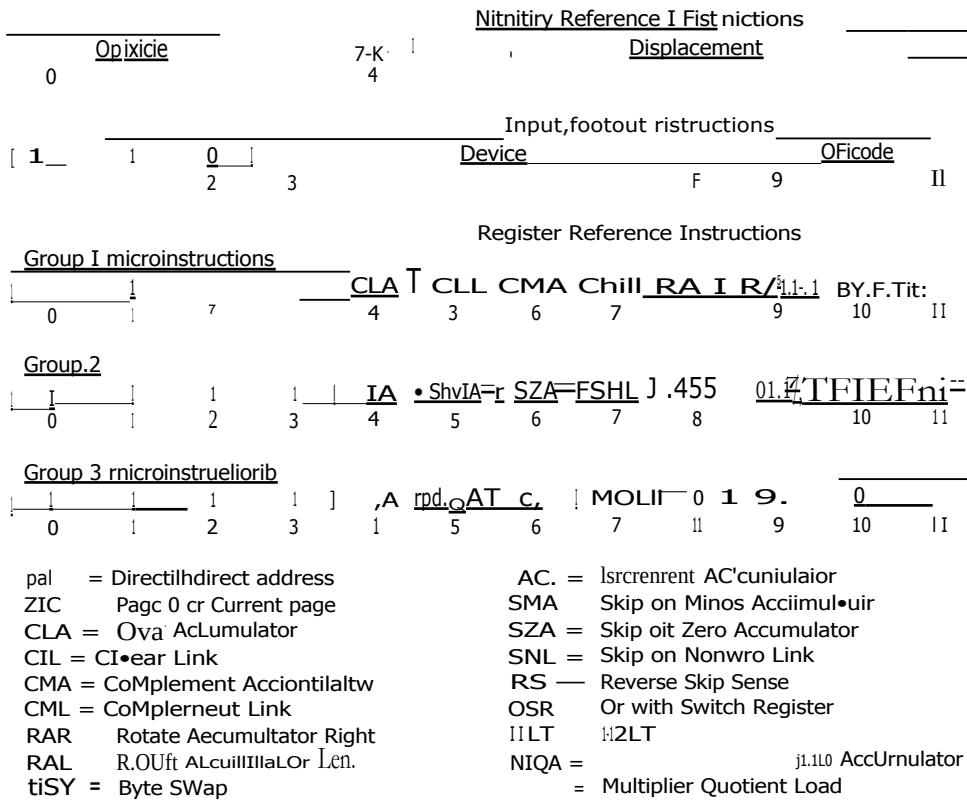


Figure **11.4** PDI³-8 tinsimetion Formats

references to pager 0 or the current page (page containing this instruction) as determined by the page bit. The second modifier bit indicates whether direct or indirect addressing is to be used. These two modes can be used in combinaiion, so ihrrt an indirect address is a 12-hil kiddrcss contained in a word of page 0 or the current page. In addition, S dedicated words on page 0 are atiloindes "registers.' When an indirect reference is made to one of these lomtions, preindexing occurs.

Figure 11,4 shows the PDP-8 instruction format. There M·V. a 3-bil opcode and three types ,,ſ instructions. For opcodes 0 through 5. the format is a single-address memory reference instruction including a page bit and an indirect hit. Thus, there are only six basic operations. 'I'o enlarge the group of operations. opcode 7 defines a register reference or *in ic roikul ruction .* in this format, the remaining bits are used lo encode additional operations. In general, each hit (lanes a specific operation (e.g.. clear accumulator), and thew oils can be combined in a single instruci ion. The microinstruction strategy was used as far back as the PDF-I by 1)1..C' and is. in a SQ:11.SC, a forerunner of today's rnicroprogrammed machines, to be discussed in Part Four. Opcode 0 is the 110 operation; 6 bits are used to select one of 64 devices, and 3 bits specify a particular PO command.

The FDP-8 instruction format is remarkably efficient, 11 supports indirect addressing, displacement addressing, and indexing. With the use of the uprotiC extension, it supports a total of approximately 35 instructions. Given the constraints of a 12-bit instruction length, the designers could hardly have done better.

PDP-111

A sharp contrast to the instruction set of the PDP-S is that **of** the PDP-10. The PUP-10 was designed to be a large-scale time-shared system, with an emphasis on making the system easy to program, even if additional hardware expense was involved.

Among the design principles that were employed in designing the instruction set were [BELL784

- **Orthogonality:** Orthogonality is a principle by which two variables are independent of each other. In the context of an instruction set, the term indicates that other elements of an instruction are independent of (not determined by) the opcode. The PUP-l0 designers use the term to describe the fact that an address is always computed in the same way, independent of the opcode. This is in contrast to many machines, where the address mode sometimes depends implicitly on the operator being used.
- **Completeness:** Each arithmetic data type (integer, fixed-point, real) should have a complete and identical set *of* operations.
- **Direct** addressing: Rase plus displacement addressing, which places a memory organization burden on the programmer, was avoided in Favor of direct addressing.

Each of these principles advances the main goal of ease of programming.

The PDP-10 has a 36-bit word length and a 36-bit instruction length. The fixed instruction format is shown in Figure 11.5. The opcode occupies 9 bits. allowing up to 512 operations. In fact, a total of 365 different instructions are defined, Most instructions have two **addresses,** one of which is one of 1.6 general-purpose registers. Thus. this operand reference occupies **4 bits.** The other operand reference starts with an 18-bit memory address field. This can be **used as an immediate operand or a memory address. In the latter** usage, both indexing and indirect addressing are allowed. The sanie general-purpose registers are also used as index registers.

A 36-bit instruction length is true luxury, There is no need to do clever things to get more opcodes; a 9-bit opcode field is more than adequate. Addressing is also straightforward. An 18-bit address Field makes direct addruSMTIg desirable, For memory sizes greater than 2^{18}. indirection is provided. For the ease of the pro-

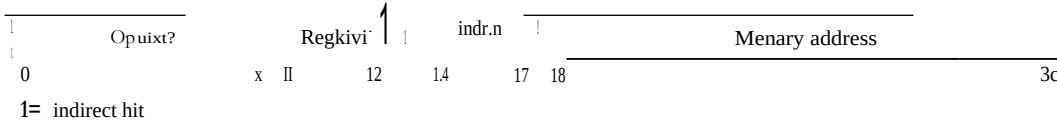


Figure 11.5 Pl)P-1O instruction Format

grammer, indexing is provided for table manipulation and iterative programs. Also. with an 18-bit operand field, immediate addressing becomes attractive..

The PDP-10 instruction set design does accomplish the objectives listed earlier TLL1ND711. The PDP-1O instruction set cases the task of the programmer or compiler at the expense of an inefficient utilization of space. This was a conscious choice made by the designers and therefore cannot be faulted as poor design.

Variable-Length Instructions

The examples we have looked at so far have used a single fixed instruction length. and we have implicitly discussed trade-offs in that context. But the designer rtm choose instead to provide a variety of instruction formats of different lengths. This tactic makes it easy to provide a large repertoire of opcodes, with different opcode lengths. Addressing can be more flexible. with various combinations of register and memory references plus addressing modes. With variable-length instructions, these many variations can be provided efficiently and compactly.

The principal price to pay for variable-length instructions is an increase in the complexity of the CPU, Falling hardware prices, the use of microprogramming (discussed in Part Four), and a general increase in understanding the principles of CPU design have all contributed to making this a small price to pay_ However. we will see that RISC and superscalar machines can exploit the use of fixed-length instructions to provide improved performance.

The use of variable-length instructions does not remove the desirability of making all of the instruction lengths integrally related to the word length. Because the CPU does not know the length of the next instruction to be fetched. a typical strategy is to fetch a number of bytes or words equal to at least the longest possible instruction. This means that sometimes multiple instructions arc fetched. However, as we shall see in Chapter 12, this is a good strategy to follow in any case.

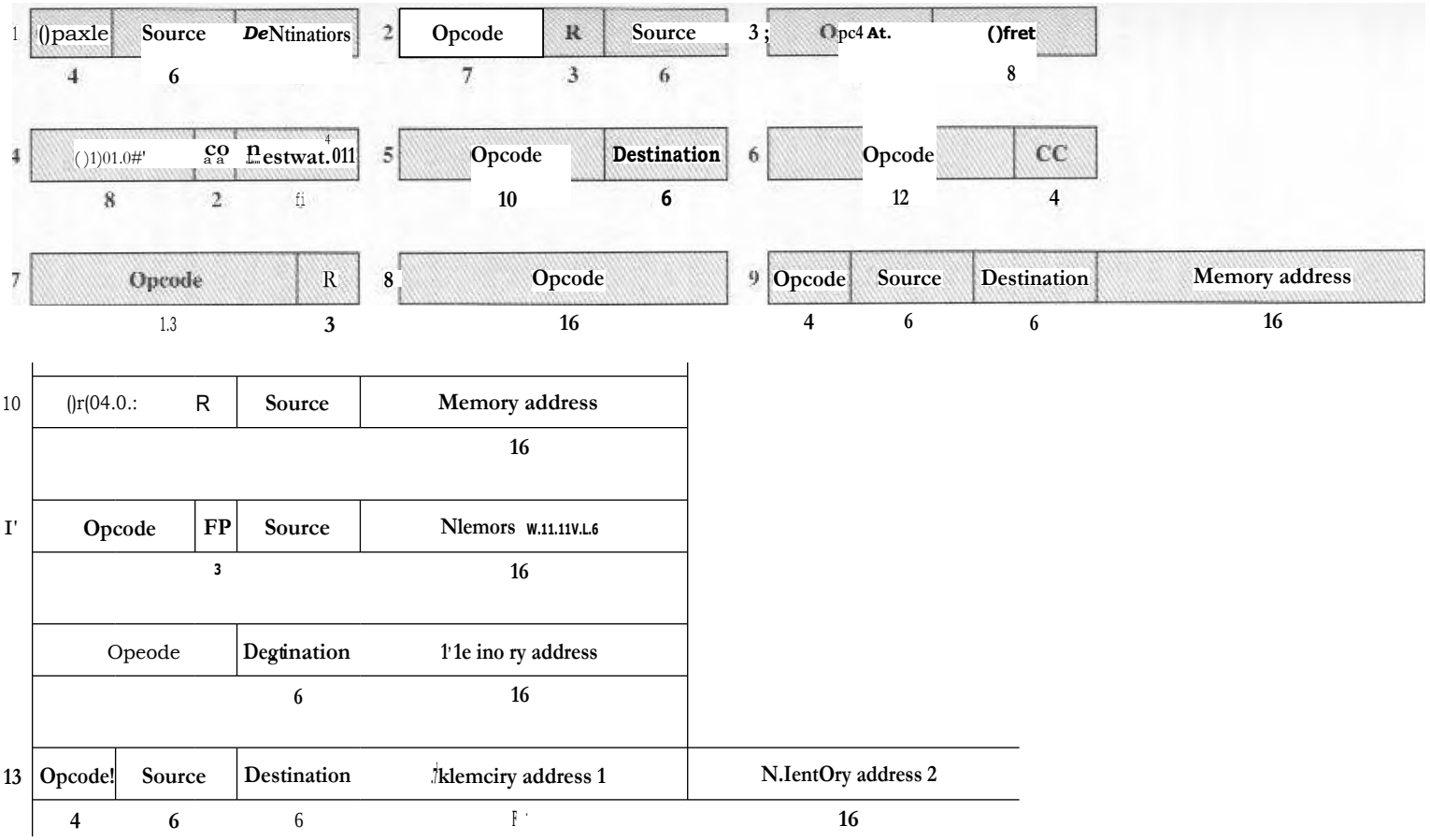
PDP-11

The PDP-I I was designed to provide a powerful and flexible instruction set within the constraints of a 16-bit minicomputer [BEU_701.

The PDP-11 employs a set of eight 16-bit general-purpose registers. Two of these registers have additional significance: One is used as a stack pointer for special-purpose stack operations, and one is used as the program counter, which contains the address of the next instruction.

Figure HA shows the PLOP-11 instruction formats. Thirteen different formats are used. encompassing zero-, one-, and two-address instruction types. The opcode can vary from 4 to 16 bits in length. Register references are 6 hits in length. Three bits identify the register, and the remaining 3 bits identify the addressing mode, The PDP-11 is endowed with a rich set of addressing modes. One advantage of linking the addressing mode to the operand rather than the opcode, as is sometimes done. is that any addressing mode can be used with any opcode, As was mentioned. this independence is referred to as *orthogonality*.

PDP-1 1 instructions are usually one word (16 hits) long. For some instructions, one or two memory addresses are appended, so that 32-hit and 48-bit instructions are, part of the repertoire. This provides for further flexibility in addressing.



Numbers below fields indicate bit length
Source and destination each contain a 3-bit addressing mode field and a 3-hit register number
Ft' indicates one of lour 11oating-p0int registers
R indiotes one of the general-purpose registers
CC is the condition code field

Figure 11.6 Instruction Formats for the PDP-1 f

The PI)1 -1.1 instruction set and addressing capability are complex. Thi' increases both hardware cost and programming cotnplexitv. The advantage is that more efficient or compact programs can be developed.

VAX

Most architectures provide a relatively small number of fixed instruction formats. This can came two problems for the progrannner. First_ addressing mode and opoode are not orthogonal_ for example_ for a given operation, one operand mug come from a register and another from memory, or both from registers, 2j11t1 so on. Second. only a limited number of operands can be viccommodated: typically up to two or I hux. WM some cjperations inherently require more operands, various NM' Legies MUSE be used to achieve the desired result using two or more insi ructions.

To avoid these problems, two criteria were used in designing the VAX instruction format iSTRE781:

1. All instructions should have the "natural" number of operands,
2. All operands should have the same generality in specification.

The result is a biddy 'variable instruel ion format. An instruction consists of a 1- or 2-byte opeodc followed by from zero to six operand specifiers, depending on the °Node. The minimal instruction length is 1. byte, and instructions op.to 37 bytes can be constructed. Figure 11.7 gives a few examples.

The VAX instruction begins with a I -byte opeode.'fhis suffices to handle most VAX instructions. However, as there are over 31 different instructions, t bits arc not enough. The hexadecimal codes FD and FF indicate an extended opcode, with the actual opcode being specified in the second byte.

The remainder of the instruction consists of up to six operand specifiers. An operand specifier is, at minimum, a 1-byte format in which the leftmost 4 hits are the address mode specifier. The only exception to this rifle is i he literal mode, which is signaled by the pattern 00 in the leftmost 2 hits, lvaving space for a 6-bit literal. Becmiz..e of this exception, a total of 12 different addressing modes can be specified.

An operand specifier often consists of just one byte, with the righirno!si 4 hits specifying one of 16 general-purpose registers- The length or I he operand specifier can he extended in one Of two ways. First, a constant value of one or more bytes may immediately follow the first byte. of the operand specifier. An example of this is the displacement mode, in which an 8-, Is-, or 3240 displacement is used. Second, an index mode of addressing may he used. ln this case. the first byte of the operand specifier consists of the 4-hit addressing mode code of 0100 and a 4-bit index register identifier. The remainder of the operand specifier consists of I he base address specifier, which may itself be one or more bytes in 'evil.

.rhe. reader tii viy be wondering. ns t he author did, what kind of instruction requires six operands. Surprisingly. the VAX has a number of such instructions. Consider

ADDP6 OP1, OP2, OP3, OP4, OP5, OPfi

This instruction adds two packed decimal numbers. OP1 and 0P2 specify the length and starting address of one decimal string; 0P3 and OP4 spc.teify a second string.

Hexadecimal FOrrnat	Explanation	Assembler Notation and Description
• 8 bits mrTi	Opcode for RSB	RSI3 Return from subroutine
D 4 5 9	01Kock. for CLRL Register R9	Cl,R I_ R9 Clear register R9
B 0 C 4 6 4 0 1 A I	Opeode for MOV W Vyrortl clisptac menI modc, Regime': R4 356 in he illecintal Byth displace meat mode, Regis.' ET R I I 25 in hexadecimal	MOV Vs(356(R4), 25 ' R I) Mow a word from ad d 'hat is 356 rl uw CI 5111.e.rils of R4 Li7 andrexs that 'IN 25 plus ciyntenis R11
C 1 0 5 O 4 F	Opcode for ADDL3 Short literal 5 Register mnt1i RO Index 111-r:Eix R2 l alireci word relative (clii,placement from PC) Arnoutu of displacement from PC mlniivc to Ideation A	ADD[..? 45, RO, Add 5 RI. a 32-bit integer in RO and store the result in loc:ItiDn whose ERItlicss is s.uin of A And 4 i Mu.% HIE conient3 of R2

Pi gime 1.13 1: ;iiiilrlc. of VAX Instructions

These Iwo strings are added and the result is stored in the deri s [ring whcnn2 length and Marting location are specified by 0P5 and OPti,

The VAX instruction !4(.1.1 provides for a wide variety of operationf; kind addressing modes. This gives a programinur_ suuh au a compiler writer, a very powerful and Clexible iool for developing proarams. In theory, this should Lead to efficient machine-laziguage conipili Lions 0f nigh-Level language programs and, in general, to effective and efficient use of CP1..: re.Nources. The penalty to *he l r,itl* for

benefits is the increased complexity of the C PL.t compared with a processor with a simpler instruction set and format.

We return to these matters in Chapter 13. where we examine t he case for very simple instruction sets.

11.4 PENTIUM AND POWERPC INSTRUCTION FORMATS

Pentium. Instruction Formats

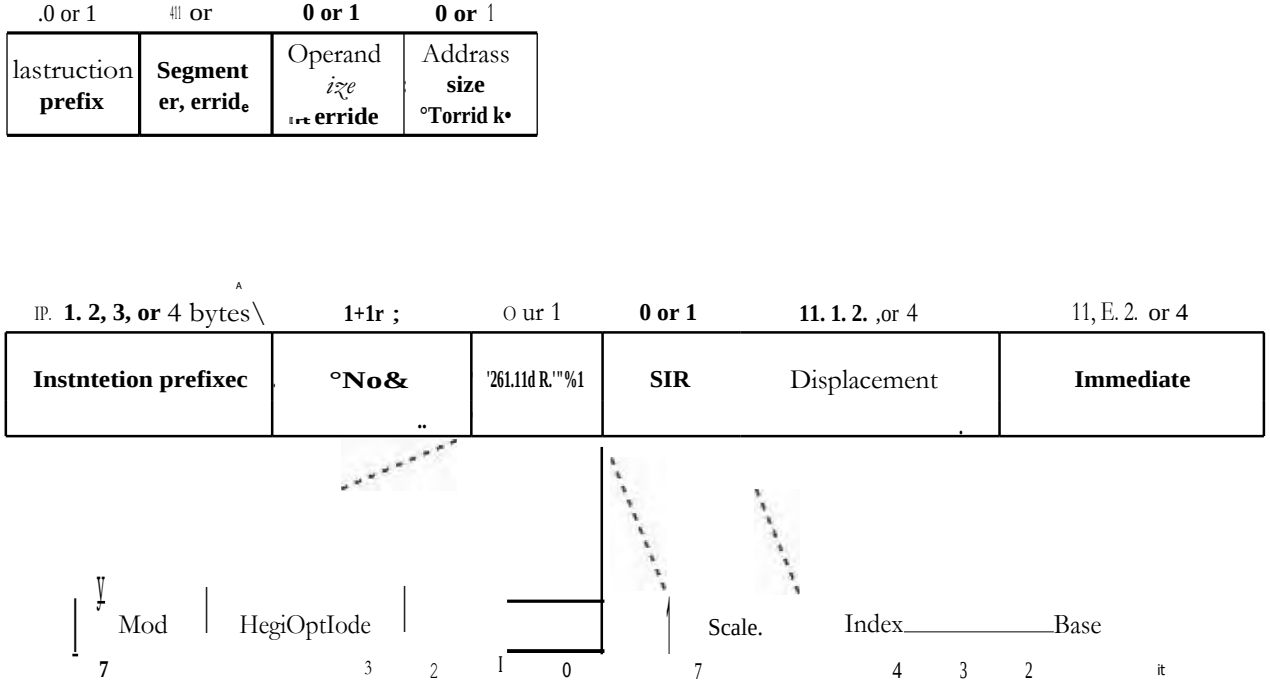
The Pentium is equipped with a variety of instruction formats. Of the elements described in this subsection, only the opcode field is always present_ Pigure 11.8 illustrates the general instruction format. Instructions are made up of from zero to four optional instruction prefixes, a 1- or 2-byte opcode. an optional address sped. Fier (which consists or the. ModFUm byte and the Scale Index byte), an optional displacement, and an optional immediate field.

Let us first consider the prefix bytes:

- **Instruction prefixes: The instruction prefix. if present,** consists of the LOCK **prefix or one of the repeat** prefixes. **The LOCK prefix is used to ensure exclusive use of** shared memory in multiprocessor environments. **The repeal prefixes** specify repeated operation of a string. which enables the Pentium to process strings much faster than with a regular software loop. There are five different repeat prefixes: REP. R1!PE, REPZ, REPN F., and RUPNZ. When the absolute RF.P prefix is present, the operation specified in the instruction is executed repeatedly on successive elements of the string; the number of **repetitions is** specified in register CX. The conditional R.EP prefix causes **the instruction to** repeat until the count in CX goes to zero or until the condition is met.
- Segment override: Hxplicitly specifies which segment register an instruction should use, overriding the default segment-register selection venerated by the Pentium for that instruction.
- **Address size: The processor can address memory using** either **16- or 32-bit addresses. The** address size determines the displacement size in instructions and the size 01 address offsets generated during effective address calculation. One of these wires is designated as default, and the address si4e prefix switches between 32-bit and 16-bit address **generation.**
- **Operand size:** An instruction has a **default** operand size of 16 or 32 bits, and the operand prefix switches between 32-bit **and** 16-bit operands.

The instruction itself includes the following fields:

- **Opcode: One- or two-byte opcode.** The opcode may also include hits that specify it data are byte- or full-size (16 or 32 bits **depending on context),** diree• lion of data operation (to or from memory). and whether an immediate data field must be sign extended.
- **ModRim: This byte, and the next, provide addressing information. The** mod Wm byte specifics whether an operand is **in** a **register or in memory; if it is in**



'Figure 11.8 Pentium. Ii truction Format

memory, then fields within the byte specify the addressing mode to he used. The ModRim byte consists of three fields: The Is.lod field (2 bits) combines with the rim field to form 32 possible values: 8 registers and 24 indexing modes; the RegiOpcode field (3 bits) specifies either a register number or three more bits of opcode inftnination; the rim field (3 hits) can specify a register as the location of an operand, or it can form part of the addressing-mode encoding in combination with the Mod field.

- SIB: Certain encoding of the Mod Rim byte specifies the inclusion of the SIB byte to specify fully the addressing mode. The SIB byte consists of three fields: The Scale field (2 bits) specifies the scale factor for scaled indexing; the Index field (3 bits) specifies the index register: the Base field (3 bits) specifies the base register.
- s **Displacement:** When the addressing-mode specifier indicates that a displacement is used, an 8-. 16-, or 32-bit signed integer displacement field is added,
- **Immediate: Provides the value of** an 8-, 16-, or 32-bit operand.

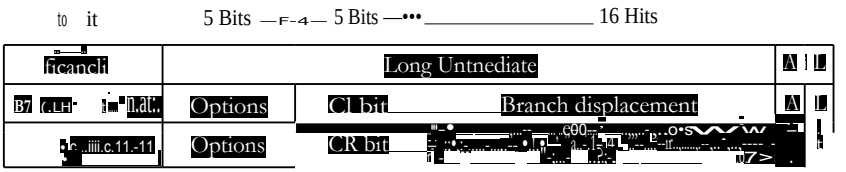
Several comparisons may be useful hero. In the Pentium format. the addressing mode is provided as part of the opeode sequence rather than with each operand, Because only one operand can have address-mode information, only one memory operand can be referenced in an instruction. In contrast, the VAX.carries the address-mode information with each operand, allowing memory-to-memory operations, The Pentium instructions are therefore more compact. However, if a memory-to-memory operation is required, the VAX can accomplish this in a single instruction.

The Pentium format allows the use of not oniv I-byte, but also 2-byte and 4-byte offsets for indexing. Although the use of the larger index offsets results in longer instructions, this feature provides needed flexibility. For example, it is useful in addressing large arrays or large stack frames. In contrast, the IBM 5.670 instruction format allows offsets no greater than 4K bytes (12 hits **or** offset information), and the offset must be positive. When a location is not in reach of this offset. the compiler must generate extra code to generate 1fie needed address. This problem is especially apparent in dealing with stack frames that have local variables occupying in excess of 4K bytes. As [MN/AM puts it. "generating code for the 370 is so painful as a result of that restriction that there have even **been compilers for the** 370 that simply chose to limit the size of the stack frame to **4K bytes."**

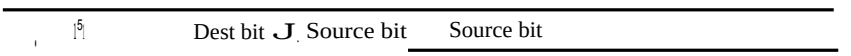
As can be seen. the encoding 0r the Pentium instruction set is very complex. This has to do partly with the need to be backward compatible with the 8086 machine and partly with a desire on the part of the designers to provide cvc.ly possible assistance to the compiler writer in producing efficient code. It is a matter of some debate whether an instruction set as complex as **this is preferable to the opposite** extreme of the RISC instruction sets.

PowerPC Instruction Formats

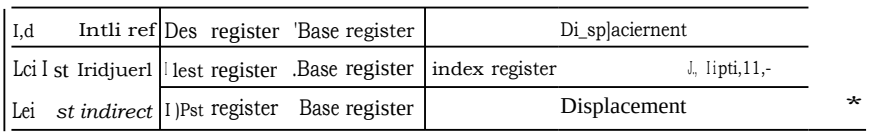
All instructions in the PowerPC are 32 bits long and follow a regular format. 'l'hc first 6 hits **of** an instruction specify the operation to be performed. In some cases. there is an extension to the opcode elsewhere in the instruction that specifies a particular subcase of an operation, In Figure 11.9, opcode hits are represented by the shaded portion of each format.



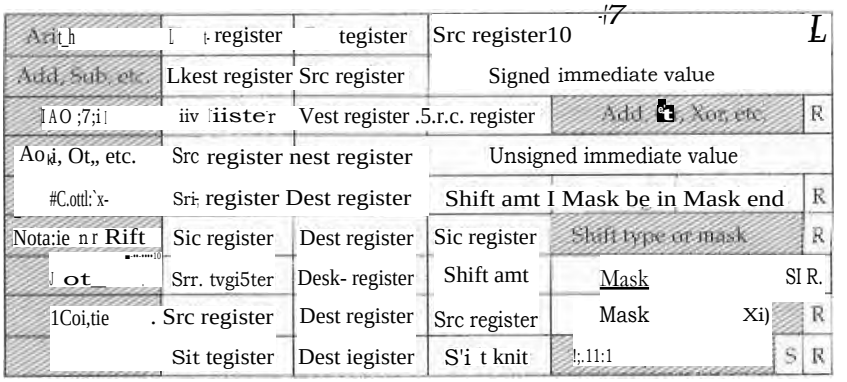
(a) Branch instructions



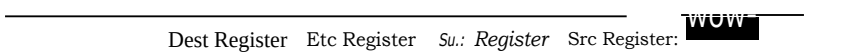
ondition register logical instructions



fc) Load /store instructions



(d) integer arithmetic, logical, and shittirotate instructions



(e) Floating-ixiint arithmetic instr uction.

A = Absolute or PC Relative
L = Link to Subroutine
O n Record Overflow in .XEK
R Record Conditions in CR1
= OpCode Extension
S = Part of .hi it Amount Field

b4-bit implementations nnly

Figure 11.9 Power PC Instruction Formats

Note the, regular structure Or the formats, which eascb the job of the instruction decode units. **For** H Il loadistore, arithmetic, and logical instructions, the opcode **is** followed by two 5-bit register references, cnabling 32 general-purpose registers to be used.

The branch instructions include. a link (Ll bit That indicates that the effeetiw addrcyi *of* the instruction following the branch instruction is to be placed in the link register. Two forms of the instruction also include a bit (A) that indicates whether the addressing mode is absolute or PC' relative. For.the conditional branch instruc. licon he CR bit field specifies the bit to tic [cm ed in the condition register. The option!, ficld specifies the conditions **under** which the branch is to bc **Liken**. The following conditions may be specified:

- Branch always.
- Branch if count 0 and condition is false.
- Branch if count 'L 0 and condition is true.
- Branch if count = 0 and condition is false,
- Branch if count = 0 and condition is true.
- Branch if count 7:- 0.
- **Branch** if count — 0.
- Branch if condition is false-
- Branch if condition is Intic,

Most instructions that result in a **0..irnpulaltion** (arithmetic. floating-point arith.-inetic, logical) include a bit that indicates whether the result oft **he** operation should he rccorded in the condition reaister. As will be shown, this feature is useful for branch prediction processing.

Floating-point instructions have fields for three source registers, In many cases. only two source regisicrs ato used. A few instructions involve multiplication of two source regisi Lis and then addition or subtraction of a third source reuistar.]'here composite instructions are included because of the frequency of **their** use. For example, the inner product that is pan of inan!,' matrix operations can be implemented using multiply-adds.

11.5 RECOMMENDED READING

Thu **1•43 Rio cliiipter 10 arc equally applicable io** *the* **material of this** Thal ter. **[BLAA971** 0.5111;111N ai 1.13d discussiun Or instructicm **formats and addressinF modes, In** adcJi(ion, the **Nader may wish to consult EFLYNK5j for >L discussion and anairiis cif instroctinn soi design issues, particularly ihusu** relating to rod rants.

BLAA97 **0., ;old Brooks, F.** *Corn pencr Aiviriwelare; Concepts Yazd Evofugueo. Re.adiftL MA_* **Addison•Weslev, 1997.**

FLYNS5 Flynn, M.7 **Johnson. J.; and Wakefield. S. "On Instruction sets and Their For: mats."** *1EEE Trior.vacti(ins on Compwc:r.s',* **March 1985-**

11.6 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terre

	immediate addressing	pteindeving
basc-register addres6ig	indexing	✓egister addressing
direct addressing	indirect addressing	register indirect ,tddressing
dhiplacorttoit addressing	instruction fitrinat	telativi: addressing
effective address	postindoxiiig,	word

Review Questions

IL] 131'i<!fly define immedkito addressina.

11.2 Brick define dirct addressing,

;nil; is2ci

11.4 Brikily kfin. register **addressing.**

11,5 Briefly define ti2gister indirect addressing.

11.6 Briefly define displacement addressing.

11.7 Briar define relative addressing.

11.8 What is the advantage of autoindexing3

11.9 What is the difference beiween postindexing and preindexing?

11.10 NA.'hat facts go into determining the use of thc addressing bits of an instruction?

11.11 What are the aclvantages and disadvantages of using a variable-length instruction formal?

Problems

11.1 Justify the assertion that a 32•bit instruction is probably much less than twiciz kIS Jul as a t6-bit instruction.

112 Given the following memory **vaihues (11141** as oiw-addre s s machine with an accumuLator. **w** hat values do the following instructions load into the accumulator?

- Word 20 contains 40.
- Word 30 contains 50.
- Word 40 contains
- Word 50 contains 70.

a. LOAD IMMEDIATE 20

b. LOAD DIRECT 20

c. LOAD INDIRECT 20

d. LOAD IMMEDIATE 30

e. LOAD DIRECT 30

C. **LOAD INDOZ.HCT .30**

11.3 La the address stored in thc program counter be designated by **the symbol x 1. Mc.** instruction stored in XI has an addross part (operand reference} V. The operand nftded to execute. the instructic.5n is stored in the memory word **Wii** 11 address X3. An index register contains the v;ilite X4. What is the relationship between these various quantitie4 if the ilicide of the instruction is (a) direct; (b) indirect: (c) PC relative; (d) inelexei I?

114 An address field in an instruction contains decimal value 14. Where is the corresponding operand located for:

a, immediate addressing?

h. direct addressine3

it. indirect addressirml

d. register addressing?

e. regiskr indirect addressing?

115 A PC-relative inode branch instruction is stored in memory at address 620 ₁₀, The branch is made to location 510„ . The address field in the instruction is 10 bits long. k the binary value in the instruction?

11.6 How many times does the CPU need to refer to memory when it fetches nerd execures an indirect-address-mode instruction if the-instruction is (a) a computation requiring a single operand; (b) a branch'?

11.7 The IE1M 37(1 does nor provide indirect addressing. Assume that the address of ao operand is in main memory. How would you access the operand?

118 Why was IBM's decision to move from 36 bits to 32 hits per word wrenching, and to w horn?

11,9 to he author proposes that the PC-relative addressing modes be climi• nated in favor of other modes, such as the use of a stack. What is the disadvantaee of this proposal?

11.10 Assume an instruction set that uses a fixed 16-hit instruction length. Operand spec-- fiers are 6 bits in length. There arc *K* two-operand instructions and *L* zero-operand instructions. What is the maximum number of orit-operand instructions that can be support ed?

11.11 Design a variable-length opcode to allow all of the following to be encoded in a 36-bit instruction:

- **ingtrUClions with Iwo 15-bit** a ddrc.ssis Ind one 3-bit register number
- instructions with one 15-hit address and uric 3-bit register number
- instructions with no addresses or registers

11.12 Consider the results of Problem 10.3. Assume that NI is a 16-bit memory address and that X, Y. and Z arc either 16-bit addresses or 4-hit register numbers. The one-adthes,i machine uses an accumulator, and the two- and three-address machines have 16 registers and instructions operating on all combinations of memory locations and reeis• ters. Assuming s-hit opcodes and instruction lengths that are multiples of 4 bits, how many bits does each machine need to compute X?

11.13 Is there any possible justification for an instruction with two opcodes?

11.14 The Pentium includes the following instruction;

```
ibEJL =1, op2, immediate
```

This instruction multiplies opt, which may he either register Or memory, by the immediate operand value. and places the result in op I , which must be. a register, There is no other three-operand instruction of this sort in the instruetion. set. What is the possible use of such an instruction? *Hine:* Consider indexing.

C11APTER 12

CPU STRUCTURE AND FUNCTION

12.1 Processor Organization

12.2 Register Organitatiou

I [se r-Vi si bit Registers
Control z•nd Status Registert4.
lixarripie Micerornet;,•5.s9i. Regi*.t Organizations

12.3 Instruction Cycle

[ii Indirect ()....cle
Dat flow

12.4 histruetion Pipelining •

Pipelining Straton•
.• Pipeline Performance •
Deating with BranchuR
Inte[8048 6 Pipelining

12.5 The Pentium Processor

Register Organization
• Interrupt Processing. ,,,

1.24 The PowerPC Procemor

Register Organization
1ntcrrupi Procumiing:.

12.7 Reconunended Rending

12.8 Key Terms, Review Questions, and Problems

Kcy Tin
Review Questions
Problems

KEY POINTS

- A processor includes both user-visible registers and controltstatus reOsters. The former may be referenced, implicitly or explicit Iv, in machine instruction's. User-visible registeu; may be general pin-pose or have a special u-se. such as fixed-point or flooi ing-point numbers, addresses. indexes, Find segment pointers. Control and status registers are used to control the operaiion of the CPU, One obvious example is the program ccatnter. Another important example is *a* program status word {J-SW} thait contains a variety of status and condition bits. I hest: include hits to reflect the result of the most recent arithmetic oper- ation, interrupt enable bits, and an indicator of whin her the. CPU is execuling in supervisor or user Mode,
- Processors make use of instruction pipelining to speed i.Lp execution. In essence, pipelining involves bre.nking up the instruction cycle into a n umhyr • of si,-..purate stages that occur in sequence. such as fetch instruction, decode in**st ruction,** deterniine operand addresses, fetch operands, uxueute iiistruction. and write operand result. Instructions move through these stages, a.s on an assembly tine. so that in principle, each stage can be working on a difl:crent instruction zii the _same. time. The occurrence of branches and dependencies between iw.tructions compiicates the design and use of pipelines.

This chapter discusses aspects of the processor riot vet covered in Part Thrce and ..el.s. the stage for the discussion of RISC and superscalar architecture in ('halters i3 and 14.

We begin with a summary of processor organization. Resisters, which form the internal memory of the processor, are then analyzed. We are then in a position to return lo the discussion (begun in Section 3.2) of the instruction cycle.. A doserip• tion of the instruction cycle and a common technique known *kL:i* instruction pipelining complete our description. The chapter concludes with an examination of sonie additional aspects. of the Pentium and PowerPC organizations.

12.1 PROCESSOR. ORGANTIZATIO.N

To understand the organizalion of the CPU, Let us consider the requirements placed on the CPU, the things that it must do:

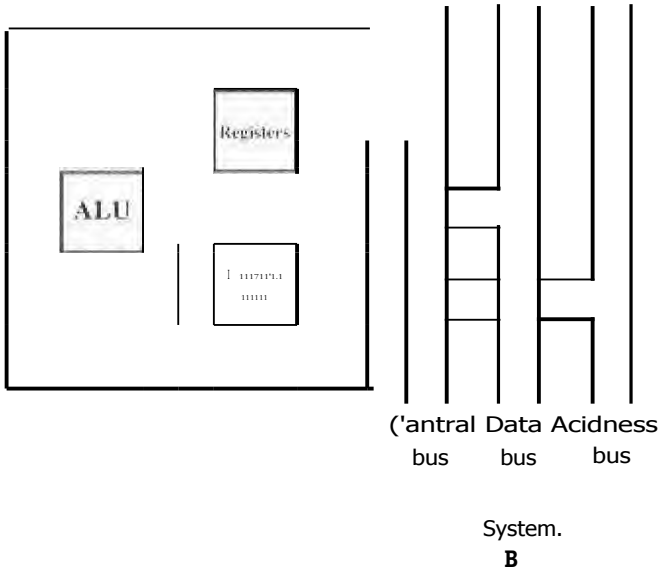
- **Fetch instruction: '1** reads an instruction from memory.
- Interpret instructiin: The instruction is decoded to determine what action k required.

- F'etch data The execution of an instruction may require reading elate from memory or an I/O module.
- Process data: The execution of an instruction may require. performing sc.inie arithmetic or logical operation on data.
- iNrite data: The results of an execution may require writing data Lo. nul.rtiory or an 110 module.

To do these things, it should be clear that the CPU needs to store some '111[1 temporarily. It must remember the location of the last instruction so [hat it in know where to get the next instruction. II needs to store instructions and data temporarily while an imtruetion is being executed- In other words. the CPU needs a small internal memory.

Figure12.1. is a simplified view of a CPU, indicating its conncetitan in the rest of the system via the symcmllii:i. A similar interface would be needed for any of Ihe interconnection structures described in Chapter 3. The reader will recall IhaL t he major components of the CPU are an *arithmetic (mil fogie;* (AM) and *a corgi-of taxi?' (CU), Thu* ALIJ does 01,2 actual wmputation or processing of data. The control unit controls the movement of data and instructions into and out of the CFU and controls the operation of the ALU. In addition. the figure show7:1 si minimal internal memory, consisting of a sel of storage loe4itions, Sul led *regiNgers.*

Figure L2.2 is a slightly more detailed view of the CPU. The data transfer and logic contrt31 paths are indicated, including an element labeled *internal CPU bus.* This element is needed to transfer data between the various registers and the



Hone 12,1 The CPU wiLli the Systcm Bus

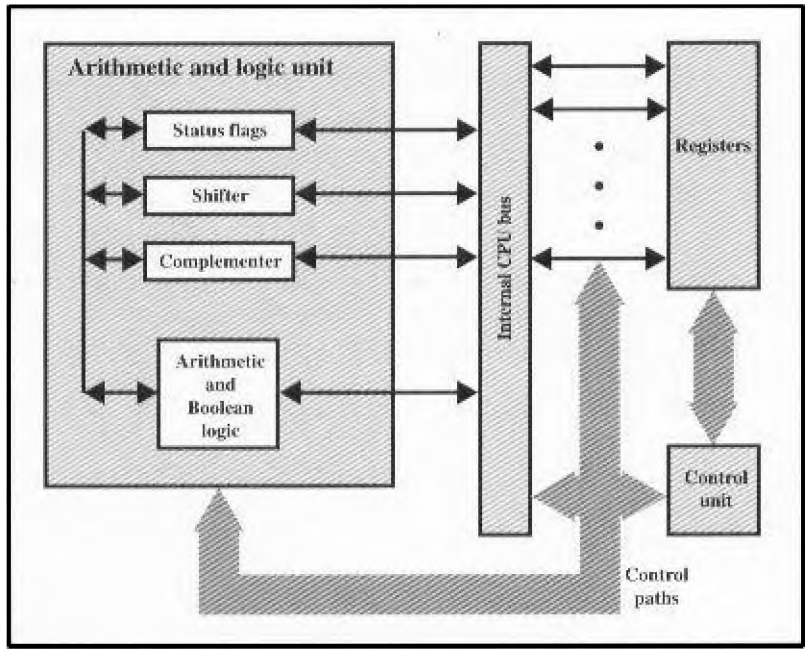


Figure 122 lRtursoaI Strnliire of kilt! cm.'

AU.;, because the ALU in fact operates only on data in the internal CPU memory. The figure also shows typical basic elements of the A UL Note the similarity between the internal structure of the computer as a whole and the internal structure of 1he. CPU, In both ca C there is a small collection of major elements (computer: CPU, $^{1/O.}$ memory; CPU: control unit, ALU. registers) connected hy data paths.

12.2 REGISTER ORGANIZATION

As we discussed in Chapter 4, a computer system employs a memory hierarchy. At higher levels of the hierarch y.mernory is fasta, s (nailer, and more expensive (per bit). Wiihin Llic CPU, there is ti set of registers that function as a level of memory ahtrvk:. main memory and cache in the hierarchy. The registers in the CPU perform two roles:

- User-visible registers: These enable the machine- or assembly-language programmer *to* minimize main memory references by optirnizin2 use of registers.
- Control :*Rd maths regigers:'Fhese are used by the control unit to control the operation of the. CPU and by privileged. operating system programs to controi the execution of programs.

There is not a clean separation of registers into these two categories. For example. on some machines the program counter is user visible (e.g., Pentium), but on many it is not (e.g.. PowerPC). For purposes of the following discussion, however, we will use these categories,

User—Visible Registers

A user-visible register is one that may be referenced by means of the machine language that the CPU executes. We can characterize these in the following categories:

- General purpose
- Data
- Address
- Condition codes

General-purpose registers can be assigned to a variety of functions by the programmer, Sometimes their use within the instruction set is orthogonal to the operation. That is, any general-purpose register can contain the operand for any opcode. This provides true general-purpose register use. Often, however, there are restricdons. For example. there may he dedicated registers for floating-point and stack operations,

In some cases, general-purpose registers can be used for addressing functions (e.g:, register indirect, displacement). In other cases, there is a partial or clean separation between data registers and address registers. **Data registers** may be used only to hold data and cannot be employed in the calculation of an operand address. **Address registers** may themselves be somewhat general purpose. or they may be devoted to a particular addressing mode. Examples include the following:

- **Segment pointers:** In a machine with segmented addressing (see Section 8.3), a segment register holds the address of the base of the segment. There may be multiple registers: for example. one for the operating system and one for the current process.
- Index registers: These are used for indexed addressing and may be autoindexed,
- **Stack pointer:** if there is user-visible stack addressing, then typically the stack is in memory and there is a dedicated register that points to the top of the stack. This allows implicit addressing; that is, push, pop, and other stack instructions need not contain an explicit stack operand.

There are several design issues to be addressed here. An important issue is whether to use completely general-purpose registers or to specialize their use. We have already touched on this issue in the preceding chapter. because it affects instruction set design, With the use of specialized registers, it can generally be implicit in the opcode which type of register a certain operand specifier refers to The operand specifier must only identify one of a set of specialized registers rather than one out of all the registers, thus saving bits. On the other hand, this specialization limits the programmer's flexibility.

Another design issue isihe number of registers, either general purpose or data plus address, to be provided. Again, this affects instruction set design because more registers require more operand specifier hits. As we previously discussed, somewhere between 8 and 32 registers appears optimum I LUND77], Fewer registers result in more memory references; more registers do not noticeably reduce memory references (e.g„ see I WILL90]). However, a new approach, which finds advantage in the use of hundreds of registers, is exhibited in some RISC systems and is discussed in Chapter 13.

Finally. there is the issue of register length. Registers that must hold addresses obviously must *be* at least long enough to hold the largest address. Data registers should he able to hold values of most data types. Some machines allow two contiguous registers to be used as one for holding double-length values.

A final category of registers, which is at least partially visible to the user, holds **condition cedes** (also referred to *as,flags).* Condition codes are bits set by the CPU hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero, or overflow result. In addition to the result itself being stored in a register or memory, a condition code is also set. The code may subsequently he tested as part of a conditional branch operation,

Condition code bits are collected into one or more registers. Usually, they form part of a control register. Generally, machine instructions allow these bits to be read by implicit reference, but the programmer cannot alter them.

In some machines, a subroutine call will result in the automatic saving of all user-visible registers, to be restored on return. The CPU performs the saving and restoring as part of the execution of call and return instructions. This allows each subroutine to use the user-visible registers independently. On other machines, it is the responsibility of the programmer to save the contents of the relevant user-visible registers prior to a subroutine call, by including instructions for this purpose in the program_

Control and Status Registers

There are a variety of CPU registers that arc employed to control the operation of the CPU. Most of these. on most machines, are not visible to the user· Some of them may be visible to machine instructions executed in a control or operating system mode-

Of course, different machines will have different register organizations and use different terminology. We list here a reasonably complete list of register types. with a brief deseription.

Four registers are essential to instruction execution:

- **Program counter (PC):** Contains the address of an instruction to be fetched,
- **Instruction register (1R):** Contains the instruction most recently fetched,
- **Memory address register (MAR);** Contains the address of a location in memory.
- **Memory buffer register (AMR):** Contains a word of data to be written to memory or the word most recently read.

Typically, the CPU updates the PC after each instruction fetch so that the PC always points to the next instruction to he executed. A branch or skip instruction

will also inodif!,' the contents of the PC. The fetched instruction is loaded into an IR, where the opcode and operand specifiers are analyzed. Lehi are exchanged with memory using the ".viAR and MBR. In 41 bu -urganixed System , the MAR connects directly to the 4iddress bux, a nc3 the MRR connects direct]to the data bus. User-visible registers, in turn, exchange data with the MBR.

The four registers just mentioned are used for he movcrnent of data between the CPU and mcmory. Within lite CPU, data must be presented to the ALU for processing. 'Ffie ALAI may have direct access to the NIBR and user-visible registers. Alternatively, there may be additional buffering registers kit !he boundary to the ALU: these registers serve as input and outpui registers for the ALL: and exchange data with the 7v111 El and user-visible registers.

Alt CPU designs include a register or set of registers, often known as the *program surgo* es *word* (PSW), that contain status information, 'De PSW typically contains condition codes plus other stitus in rorrnation. Common fields or flags include the following:

- Sign: Contains the sign bit of the result of the last arithmetic operation.
- Zero: Set when the result is O.
- Carry; Set if an °petal ion resulted in a carry (addition) into or borrow (subtraei ion) Out of a high-order bit. Used for multiword arithmetic operations,
- **Equal:Set** if a Logical compare result is equality,
- **Overflow:** Used to indicate 4irithmoik overflow.
- **Interrupt enable/disable:** Used to enable or disable interrupts.
- **Supervisor; Indicates whohcr** the. CPU is L.xectiting in supervisor or user mock Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

A number of other registers related to status and control might be Cound in a particular CPU design. In addition to the NW. there may be a pointer to a block of memory containing additional status information (e.g., process control blocks). In machines using vectored interrupts, an interrupt vector register may be provided. I a stack is used to implement certain functions (e.g., subroutine call). then a system stack pointer is needed. A page table pointer k uz,ed with a virtual memory system. Finally, registers. may be used in the control of I/O operations.

A number of factors go into the design of the control iind status register organization. One key issue is operating system support, Certain Iypes 0r control information are of specific utility to the operi Li ng system. If the CPU designer has a functional understanding of the operating system to he used, then the register organizatit'an can to some extent be tailored to the operating system,

Another key design decision is the alloeation of control information between registers and memorv, II is common to dedicate the first (lowest) few hundred or thousand words of memory for control purposes. The designer must decide how much control in formai ion Nh(alki be in registers and how much in memory. The usual trade-Of I of cosi speed arises.

Example Microprocessor Register Organizations

It is instructive to examine and compare the register organization of comparable systems. In this section, we look at two 16-bit microprocessors that were designed at about the same time: the Motorola [v1058000 ISTRI79] and the Intel 8086 1MORS781. Figures 12.3a and b depict the register organization of each; purely internal registers, such as a memory address register, are not shown.

The MC6S(1) partitions its 32-bit registers into eight data registers and nine address registers. The eight data registers are used primarily for data manipulation and are also used in addressing as index registers. The width of the registers allows 8-, 16-, and 32-bit data operations, determined by opcode, 'I he address registers con. lain 32-bit (no segmentation) addresses; two of these registers are also used as stack pointers, one for users and one for the operating system, depending on the current execution mode. Both registers are numbered 7, because only one can be used at a time. The MC68000 also includes a 32-bit program counter and a 16-bit status register.

The Motorola team wanted a very regular instruction set. with no special-purpose registers, A concern for code efficiency led them to divide the registers into two functional **components,** saving one bit on each register specifier. This seems a reasonable compromise between complete generality and code compaction.

The Intel 8086 takes a different approach to register organization, Every register is special purpose, although some registers are also usable as general purpose. The 8086 contains four 16-bit data registers that are addressable on a byte or 16-bit basis. and four 16-bit pointer and index registers. The data registers can be used as general purpose in some instructions. **In** others, the registers are used implicitly. For example, a multiply instruction always uses the accumulator. The four pointer reg. isters are also used implicitly in a number of operations; each contains a segment offset. There are also four 16-bit segment registers, ' l'hree of the four segment registers are used in a dedicated. implicit fashion, to point to the segment of the current instruction (useful for branch instructions), a segment containing data, and a segment containing a stack. respectively. These dedicated and implicit uses provide for compact encoding at the cost of reduced flexibility, The 8086 also includes an instruction pointer and a set of 1-bit status and control flags.

T'he point of this comparison should be clear. There is, as yet, no universally accepted philosophy concerning the best way to organizc.CPU registers [TOON811. As with overall instruction set design and so many other CPU design issues. it is still a matter of judgment and taste.

A second instructive point concerning register organization design is illustrated in Figure 12.3c. This figure shows the user-visible register organization for the Intel 803K6 ELAYS.51, which is a 32-bit microprocessor designed as an extension of the 8086.' The 80386 uses 32-bit registers. However. to provide upward compatibil• ity for programs written on the earlier machine, the S0386 retains the .original register organization embedded in the new organization. Given this design constraint, the architects of the 32-hit processors had limited flexibility in designing the register organization.

---

[1] BC,7.3 we the MC650(K) already uses 32•hit registers. the MCMO20 [IACCI&4]. which is H lull 32-hit archtexture. use's the Skime register organization.

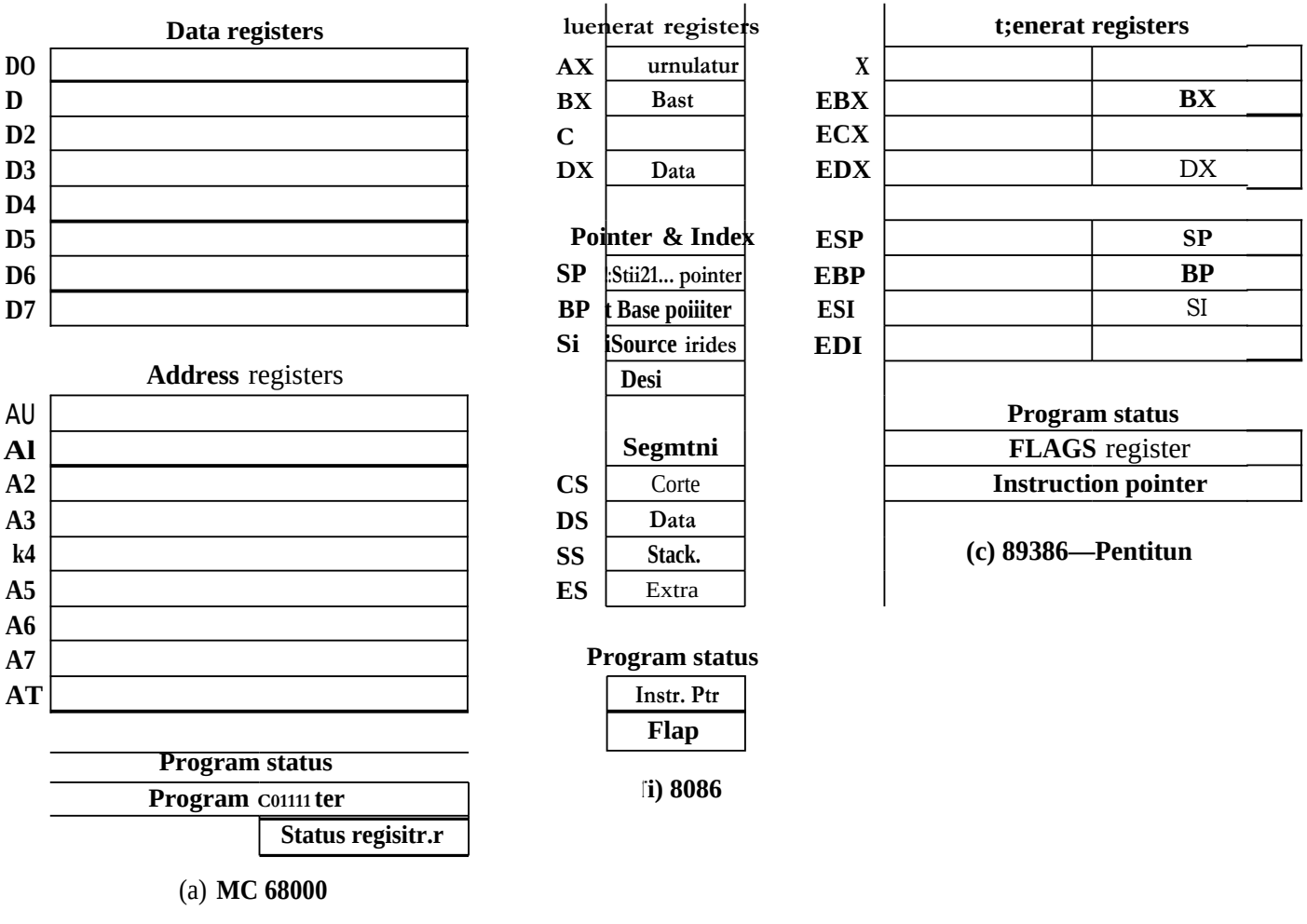


Figure 113 Example 'Microprocessor Register Organizaiticirls

12.3 INSTRUCTION CYCLE

In Section 3.2, we described the CPUs instruction cycle (Figure 3,9), To recall, an instruction cycle includes the following suhcycles:

- Fetch: Read the next instruction from memory into the CPU,
- Executer Interpret the opcode and perform the indicated operation.
- Interrupt: If' inierrupi,s4.ire enabled and an interrupt has occurred, save the current process state and service the interrupt.

We are now in a position to elaborate somewhat on the instruction cycle. First, we mint introduce one ,ridditional subcycie, known as the indirect eycic,

The Indirect Cycle

We have seen, in Chapter 11, that the execution of an insLruction may involve one ter more operands in memory, each of which requires a memory access. Further, if indirect addressing is used, [hen additional memory accesses are required,

We can think or the fetching of indirect addresses as one more insiruci ion sub. *cycle.* The result is shown in Figure 12,4. The main line of activity consists of alternating instruction fetch and instruction execution activiii.e. After an instruction is fetched, it is examined to determine if' any indirect addressing is involved. If so, the required operands are fetched using indirect addressing. Following execution, an interrupt may he processed before the next instruction fetch.

Another way to view this process is shown in Figure 12.5, which is a revised version of Figure 3.12, 'Ibis illustrates more correctly the nature of ihe instruction **cycle.** Once an instruction is fetched, its operand specifiers rriusi, be identified. Each

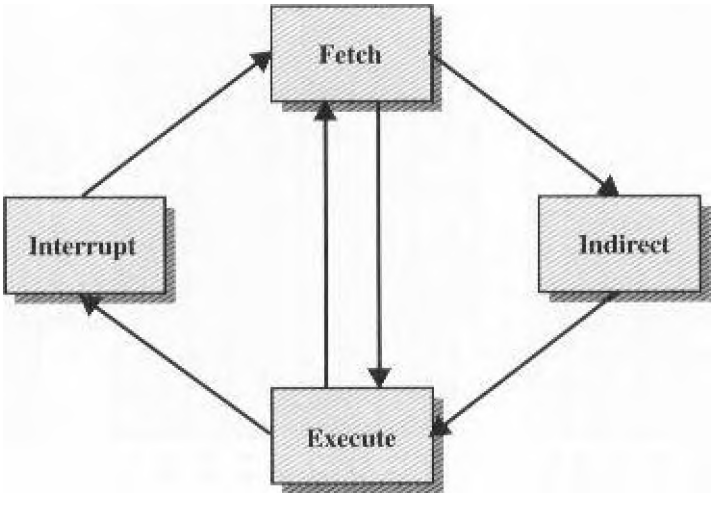


Figure 12.4 The Instruction Cycle

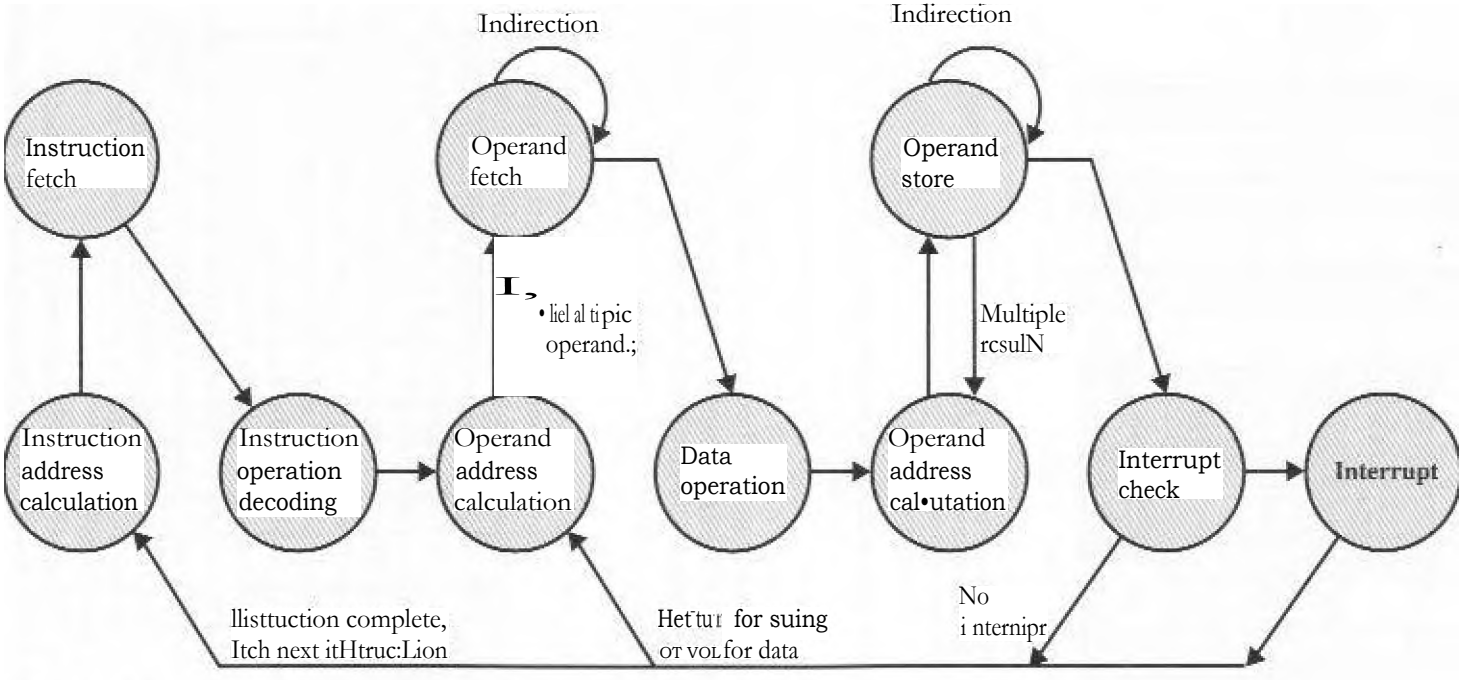


Figure 12.5 Instruction Cycle Stag Diagram

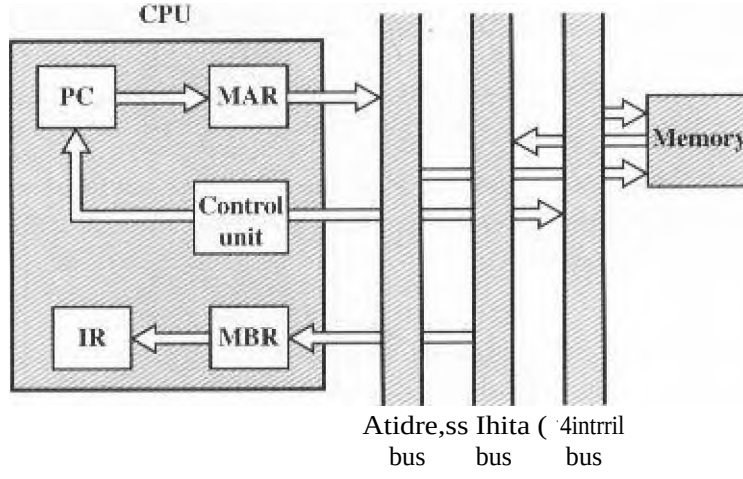


Figure 116 Data Flow. Fetch Cycle

input operand in memory is then fetched. and this process may require indirect addressing. Register-based operands need not he fetched. Once the oprode is executed, a similar process maybe. needed to store the result in main memory.

Data Flow

The exact sequence of events during an instruction cycle depends on the design of the CPU. Wu can, however, indicate in general terms what must happen. Lei us assume that a CPU that employs a memory address register (MAR), a merruiry buffer register {rvIBR}. a program counter (PC), and an instruction register (IR).

During the *filch cycle,* an instruction is read from memory. Figure 12,6 shows the flow of data during this cycle. The PC coniiiiris the address of the next instruction to be fetched. This address is moved to the MAR and placed on the address bus. The control unit requests a memory read, and the resell is piaced on the data bus and copied into the MBR and then moved to the 1R. Meanwhile, the PC is inuCtricn Led by 1, preparatory for the next retch,

Once the fetch cycle is over, the control unit examines the contents of the IR to determine if it contains an operand specifier using indirect addressing. If so, an *indire'c't cycle* is performed. As shown. in Figure 117, this is a si mple cycle. The righlmosi N' hits of the MBR, which contain the address reference, are transferred to the M.R. Then the control unit requests a memory read, to get the desired address of the operand into the MBR.

The fetch and indirect cycles are simple and predictable, The *execure cycle* takes many forms; the form depends on which of the various machine instructions

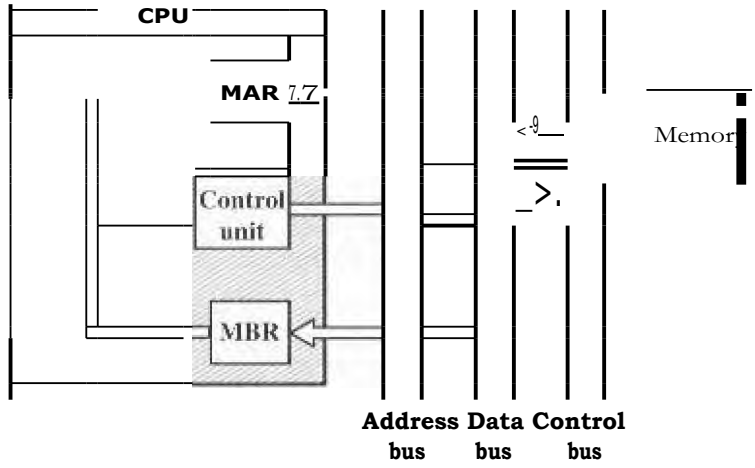


Figure 12.7 DatR Flow, I lidimet Cycle

is in the IR. This cycle may involve transferring data among registers_ rcad or write from memory or I/O, and/or I hi: **invocation or the All!.**

Like the fetch and indirect cycles, the *inierrapt cycle* is simple and predictable (Figure 12.8). The current contents of the PC must be saved so that ihc CPC eAn resnrw normal activity all L=r the interrupt_ Thus, the contents of the *PC.' arc* transferred to the MRR to lie written into memory. The special memory location reserved for this purpose is loaded into the MAR from the control unit, it might, for 'examplc, be a stack pointer. The PC is loaded with the atldro.s of *the* interrupt routine. As 2) result, the next instruction cycle **bcgin by** fetching the appropriate instruction.

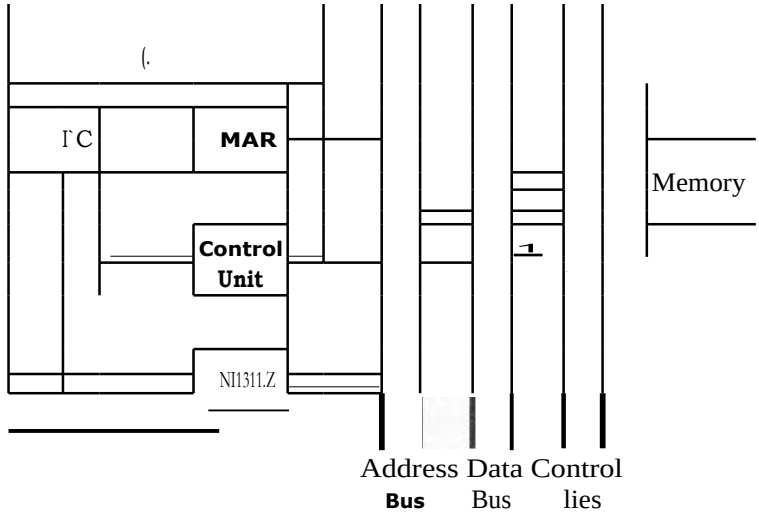


Figure 12.8 Data Flow, Interrupt Cycle

12.4 INSTRUCTION PIPELINING

As computer systems evolve, greater performance can be achieved by taking advantage of improvements in technology, such as faster circuitry. In addition, organizational enhancements to the CPU can improve performance, We have already seen some examples of this, such as the use of multiple registers rather than a single accumulator, and the use **of** a cache memory. Another organizational approach, which is quite common, is instruction pipelining,

Pipelining Strategy

Instruction pipelining is similar to the use of an assembly line in a manufacturing plant. An assembly line takes advantage of the fact that a product goes through various stages of production. By laying the production process out in an assembly line. products at various stages can be worked on simultaneously. This process is also referred to as *pipelining,* because, as in a pipeline, new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end.

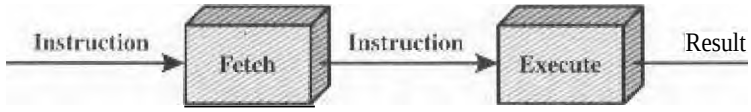
To apply this concept to instruction execution, we must recognize that, in fact, an instruction has a number of stages. Figure 12.5, for example, breaks the instruction cycle up into 10 tasks, which occur in sequence. Clearly, there should be some opportunity for pipelining,

As a simple approach, consider subdividing instruction processing into two stages; fetch instruction and execute instruction, 'Fficre are times during the execution of an instruction when main memory is not being accessed. This time could he used to fetch the next instruction in parallel with the execution of the current one, Figure 12,9a depicts this approach. The pipeline has two independent stages. The first stage fetches an instruction and **buffers** it. When the second stage is free, the first stage passes it the buffered instruction. While the second stage is executing the instruction. the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction. This. is called *imtruction prefeich* or *fetch (Pverifirp.*

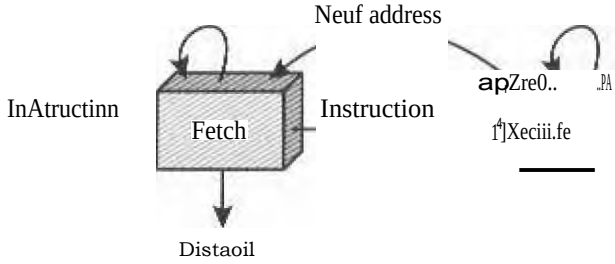
It should he clear that this process will speed up instruction•execution. If the fetch and execute stages were of equal duration, the instruction cycle time would be halved. However, if we look more, closely at this pipeline (Figure 12.9b), we will see that this doubling of execution rate is unlikely for two reasons:

1. The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some lime before it can empty its buffer.
2. A conditional branch instruction makes the address of the next instruction to be fetched unknown. Thus, the fetch stage must wait until it receives the next instruction address from the execute stage. The Qxecute stage may then have to wait while the next instruction is fetched.

Guessing can reduce the time loss from the second reason. A simple rule is the following; When a conditional branch instruction is passed on from the fetch to the



tko Simplified view



(to 14....xpnoded view

Figure 12.9 Two-Stage Insirticiion Pipeline

execute stage, the fetch stage fctches the next instruction in memor!,.. after the branch instruction. Then, if the branch is not taken, no time k lost. If the branch is taken. the fetched instruction room be discarded and a new instruction retched,

While these factors reduce the potential effectiveness of the two-stage piNline, some speedup oc.curs. ' l'o gain further speedup. the pipeline must have more stages. Let us consider the following decomposition of the instruction processing.

- Fetch instruction (FI): Read the nem. Opeeled instruction into a buffer.
- Decode instruction **OW** Determine the opcode arid the operand specifiers.
- Calculate operands (COO Calculate the effective address of each source operand. This may involve displacement. register indirect, indirect, or other forms of address Liileula Lion.
- Fetch operands (FO); Fetch each operand from memory. Operands in registers need riot be fetched.
- Execute instruction (ED: Perform the indicated operation and store the result, if any, in the specified destination operand loction.
- Write operand (WO): Store the result in memory.

With this decomposition, the various stages will be of more nearly equal duration. For the sake of ilhistration, let us assume equal duration. Using lhis assumption. Figure 12.10 shows that a six-stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.

Several comments are in order: The diagram assumes that each instruction goes through al] six stages of the pipeline. This will not always be the case. [or examIlle. a load instruction does not need the WO stage, However, to simplify the

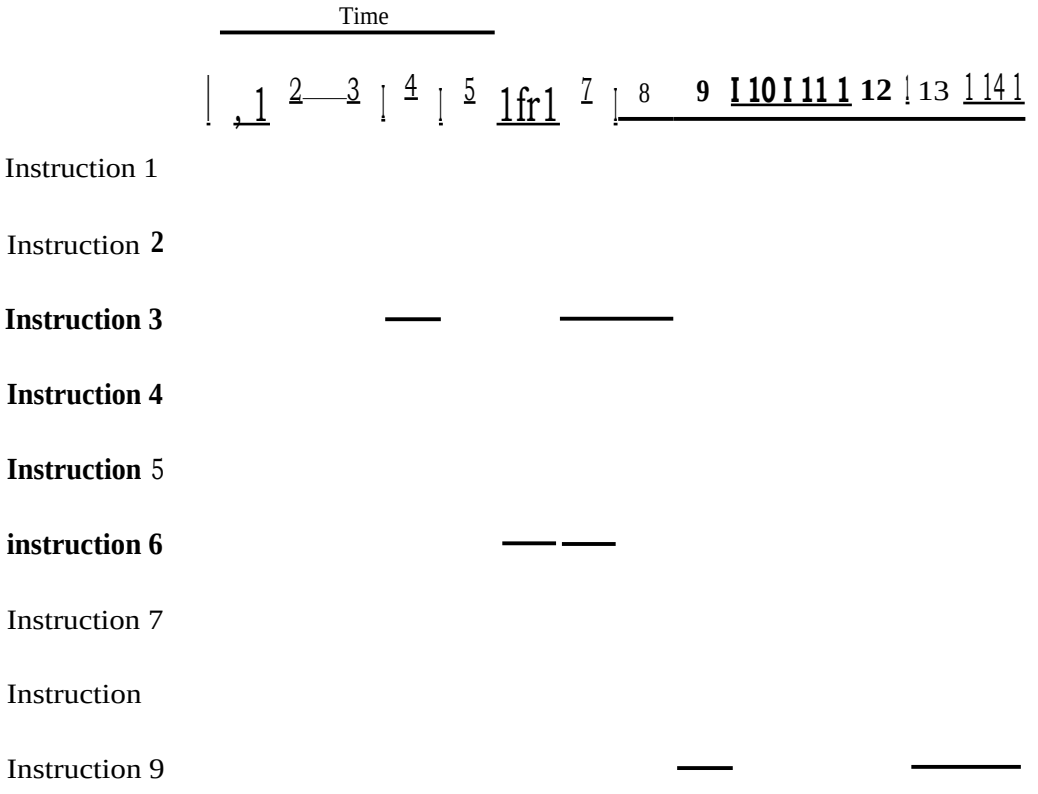


Figure 12,10 Tirninsz Diagram for Instruction Pipeline Opclatioti

pipeline. hardware, the timing is set up assuming that each insiruction requires all six stages. Also, the diagram assumes that all of the. stages can be performed in parallel. In particular, it is assumed that there are no memory conflicts. For cxamplu, the F1, FO, and WO stages involve a memory access. The diagram implies that all these accesses can occur simultaneously, Most memory systems will not permit that. However, the desired value may he in c,:iche, or the FO or It/V0 stage may be null. Thus, much of the lime., memory conflicts will not slow down the pipeline.

Sevtaa I odicr factors serve to limit the performance enhancement, if the six stages are not of equal duration, there will be some waiting involved at various pipeline stages, as discussed before for the two-stage pipeline. Another difficulty is the conditional branch instruction, which can invalidate several instruction retches. A similar unpredictable event is an interrupt. Figure 12,11. illustrates the effects of the. conditional branch. using the same program as Figure 12..10. Assume that instruction : is a conditional branch to instruction 15. Until the instruction is executed, there is no way of **knowing** which instruction will come next. The pipeline, in this example, simply loads the next instruction in sequence (instruction 4) and pro-

seeds. In Figure 12..10, the branch is not taken, and we get the full performance benefit of the enhancement. In.Fig.ure 1.2.11, the. branch is taken. This is not determined until the end of time unit 7, Al this point, the pipdiaw. must be cleared of instrue-

Lhat arc. not useful- During unit 8, insiruction 15 enters the pipeline. No instructions complete during time units 9 through 12.; this is the performance penalty incurred because. we could not anticipate the branch. Figure 12.12 indicates the lOgiC needed for pipc,lining to acuount l'orbmnulies and interrupts.

Other problems arise that did not appear in our simple two-stage organization. The CO stage may depend on the contents of a register that could be altered by a previous instruction that is still in the pipeline. Other such rcgisier and memory eunilicts could occur- The. Nys.11 cm must con Li fl logic to account for this type or conflict.

To clarify pipeline operation, it might be useful to look at an alternative depiction. Figures 1.2.111 and 12,11 show the progression of time horfc.orua I ly across the figures. with each row 1.,howing [hi: progress of Lin individual instruction. Figure 12.13 shows same sequence of events, with time progressing vertically down the figure.

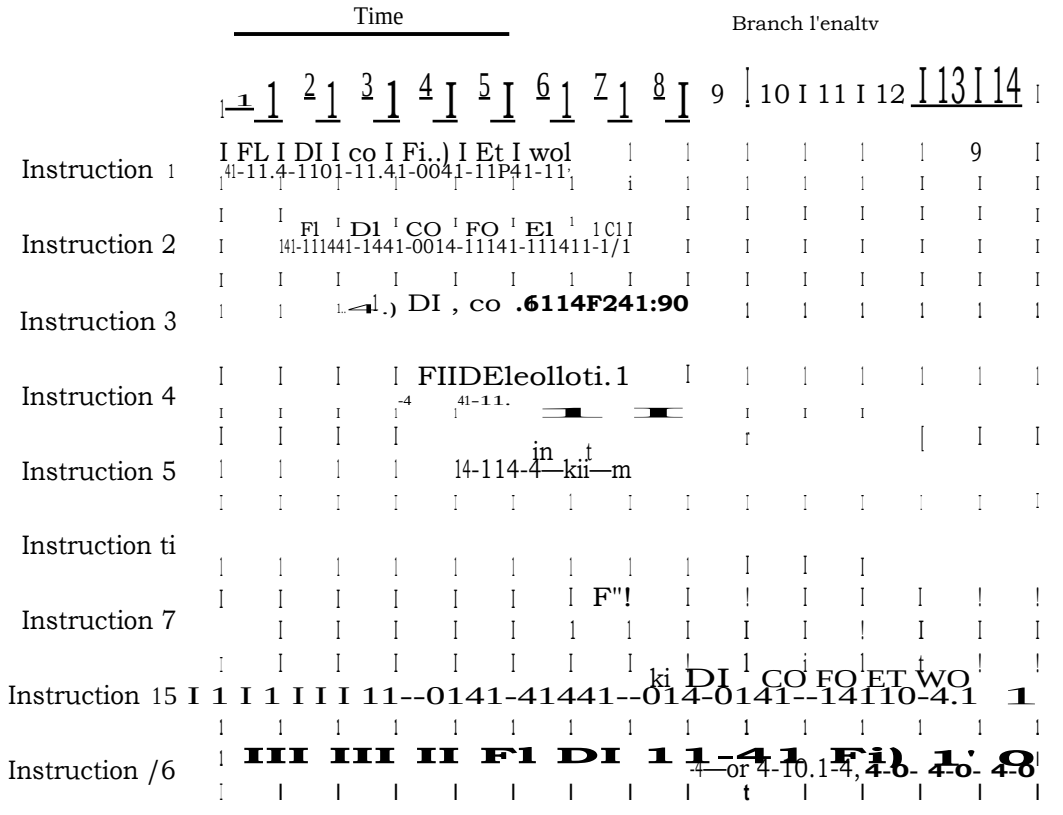


Figure 12.11 The Effoct of a CondiLional Branch on Instruclion Fiparic Operation

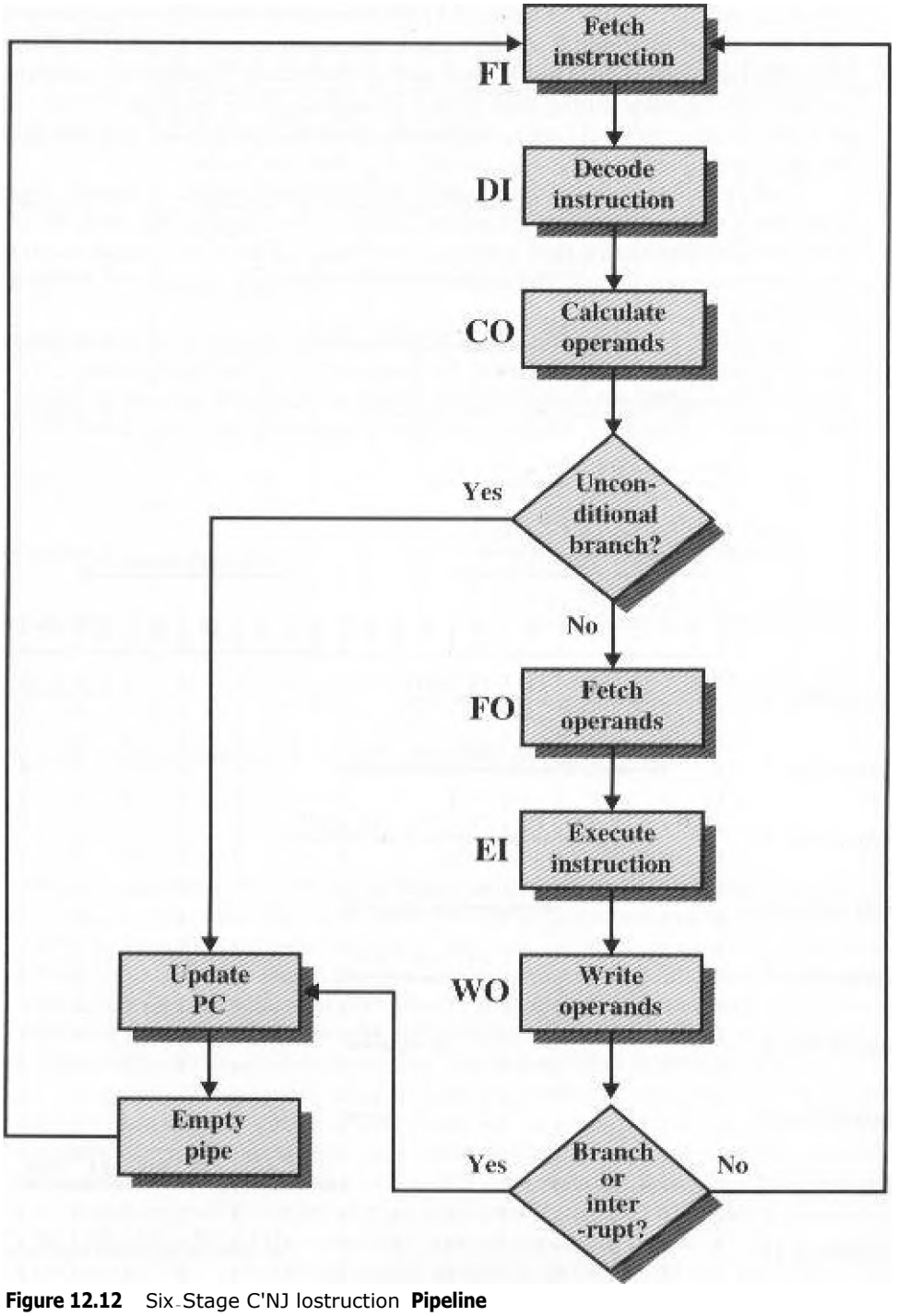


Figure 12.12 Six-Stage C'NJ lostruction **Pipeline**

and ciich row showing Lhc stoic of the pipeline at a given point in time. In Figure 3a (which corresponds to Figure 2,10). the pipeline is full at time 6. with 6 different instructions in various stages of execution. and remains full through time 9; we assume that instruction 1' is the Iasi instruction to he executed. In Figure 12.13n. (which corresponds lo Figure 12.11), the pipeline is full at times 6 and 7. At time 7,

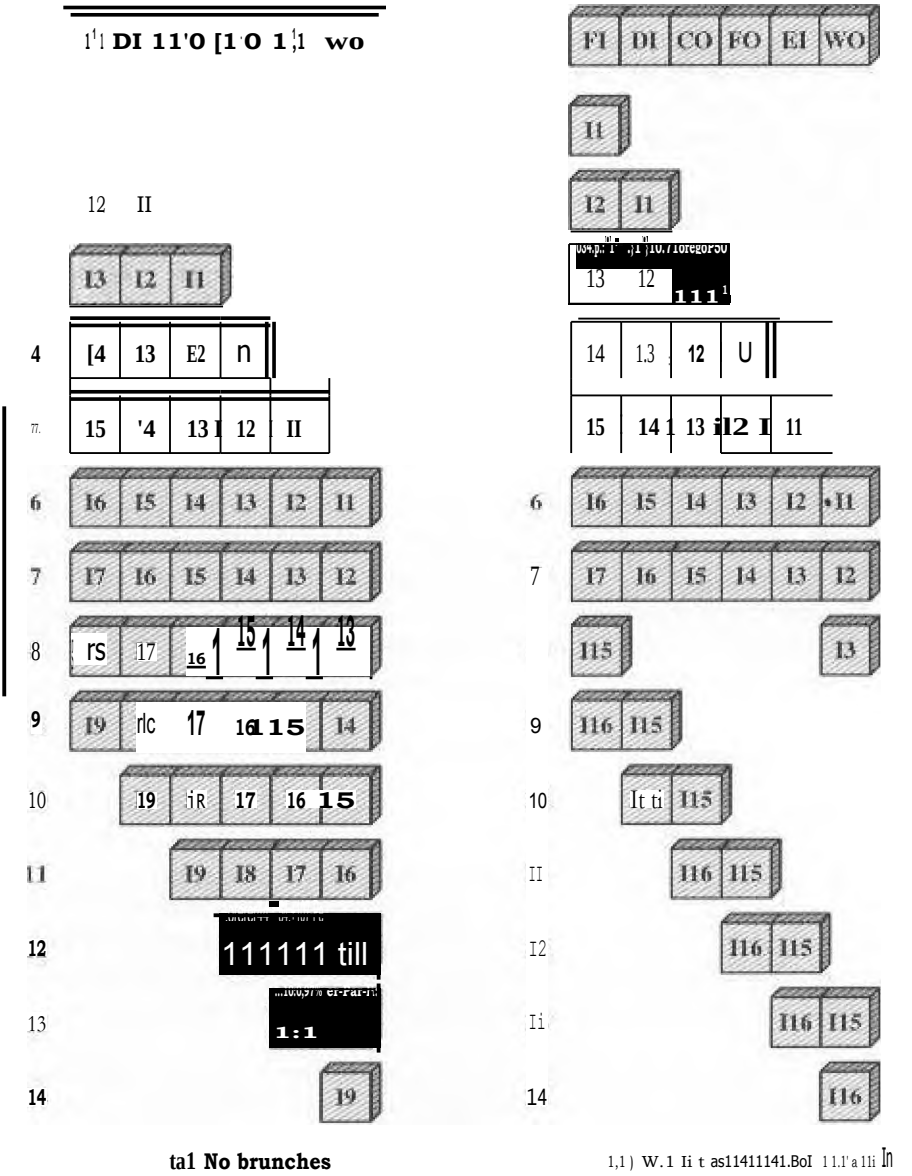


Figure 12.13 An Alternative Pipeline Depiction

instruction 3 is in the execute stage and executes a branch to instruction 15. At this point, instructions 14 through 17 are flushed from the pipeline, so that at time 8, only two instructions are in the pipeline, 13 and 115.

From the preceding discussion, it might appear that the greater the number of stages in the pipeline, the faster the execution rate. Some of the IBM &MO designers pointed out two factors that frustrate this seemingly simple pattern for high-performance design [ANDE67a], and they remain elements that designer must still consider:

1. At each stage of the pipeline., there is some overhead involved in moving data from buffer to buffer and in performing various preparation and delivery functions. This overhead can appreciably lengthen the total execution time of a single instruction. This is significant when sequential instructions are logically dependent, either through heavy use of branching or through memor!, ' access dependencies.
2. The. amount of control logic required to handle memory and register dependencies and to optimize the use of the pipeline increases enormously with the number of stages.. This can lead toy situation where the logic controlling the gating between stages is more complex than the stages being controlled.

Instruction pipelining is a powerful technique for enhancing performance but requires careful design to achieve optimum results with reasonable complexity.

Pipeline Performance

In this subsection, we develop sonic simple measures of pipeline performance and relative speedup (based on a discussion in IHWAN931). The cycle time -r of an instruction pipeline is the time needed to advance a set of instructions one stage through the pipeline; each column in Figures 12.10 and 12.11 represents one cycle time. The cycle time can he determined as

$$= mitx[rd + d = \quad d \quad \text{l ei} \quad k$$

where

T_{max} = maximum stage delay (delay through stage which experiences the largest delay)
k = number of stages in the instruction pipeline
d = time delay of a latch, needed to advance signals and data from one stage to the next

In general, the time delay d is equivalent to a clock pulse and $\tau_{max} >> d$. Now suppose that n instructions arc processed, with no branches. The total time required to execute all n instructions is

$$T_e = [k \quad (n - 1)] \quad \text{02.0}$$

A total of k cycles are required to complete the execution of the first insiruction, and the remaining n I instructions require $n - 1$ cvelus-[2] This equation k verified from Figure 12.1[1- The ninth instruct ion completes at time cycle]4:

$$14 = [6 - (9 - 1)j$$

The speedup factor for the instruction pipeline compared to execution withoul the pipeline is defined as

$$T_{(} \quad [k + fro \quad 1)]'r \quad k \quad \cdot I) \qquad (12.2)$$

Figure 12.14a plots the speedup *factor* as a function of the number of instructions that are executed withou1 a branch. As might be expected, at the limit ∞, wc have a k-fold speedup. Figure 12,14h shows the speedup factor as a function of the number of sta.:4es in the instruction pipeline.` Iii this ease, the speedup factor approaches the number of instructions that can be fed into the pipeline without branches. Thus, Ihe Larger the number of pipeline stages. the greater the potential for speedup. I Iowever. as a practical matter, the potential gains of additional pipeline, stages are aninIered by increases in cost, dc.]ays between stages, and the fact that branches will be encountered requiring the flushing of the pipeline,

Dealing with Branches

One of the major problems in designing an instruction pipeline is assuring a steady flow of instructions to the initial stages of the pipeline. The primary impediment, as we have seen, is the conditional branch instruction. Until the instruction is actuall!,. executed, it is impossible to determine whether the branch will he takes **or 1101,**

A variety of approaches have been taken for dealing with condition;i1 branches:

- Multiple streams
- Prefetch branch target
- Loop buffer
- Branch predictic.in
- Delayed branch

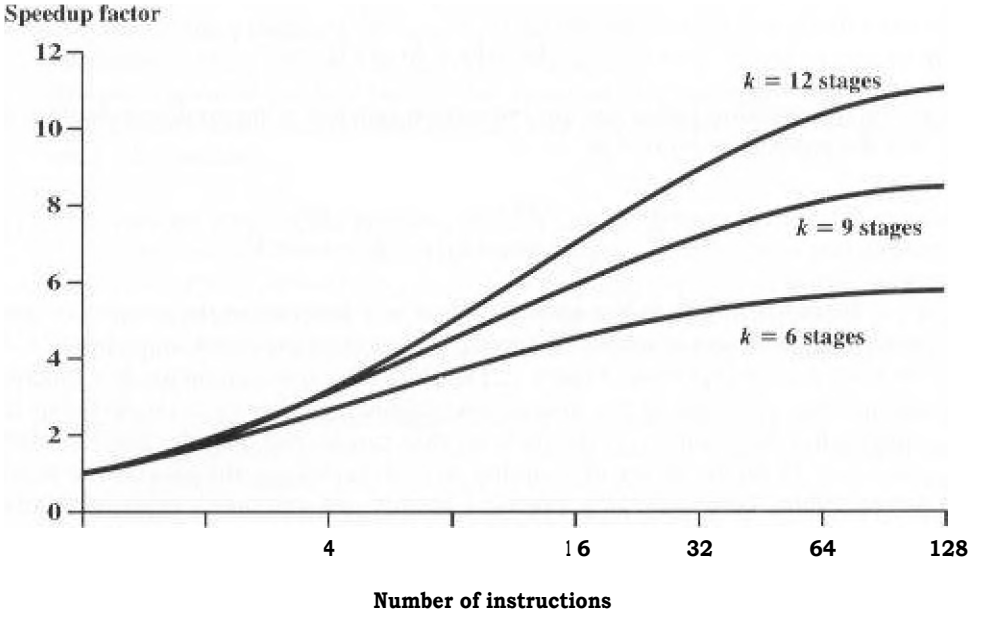
Multiple Streams

A simple pipeline suffers a penalty for a branch instruction because it must choose one of two instructions to feCeh next and may make the wrong choice. A brute-force approach is to replicate the initial portions of the pipeline and allow the

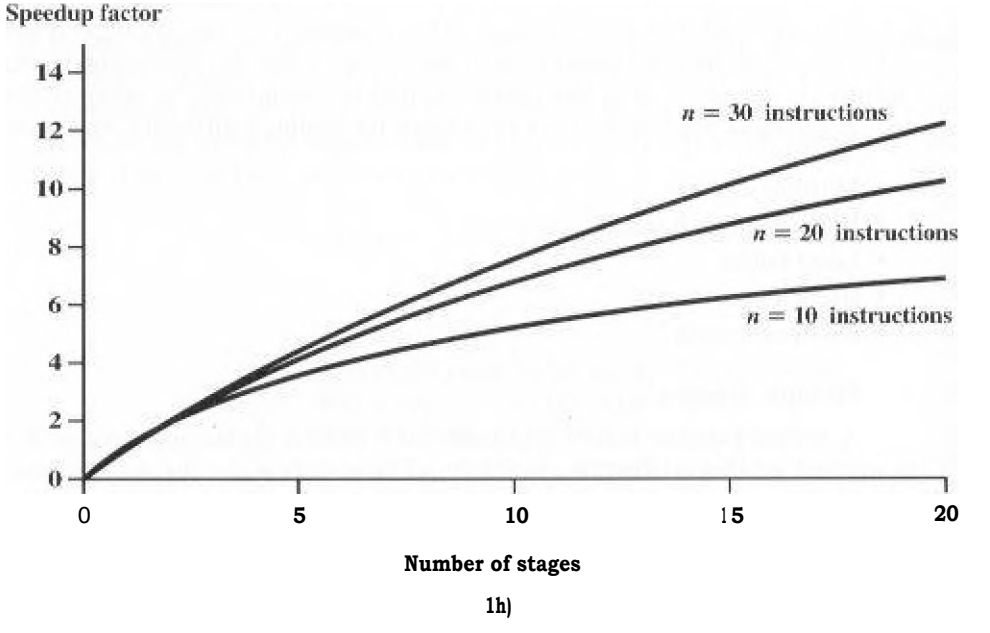
---

'We are being ki hie slopm: hem. The eyc10 circle only **equal** Ebc maximum vulva *it* 7 when all thd stuger. **FITE Full.** At **Etc bLprining,** cycle **Li mL inHy hex JeSS** Ear 1.11..1. first **Fri 1)1: lcw CyCl.eS.**

'Note that 1.11.e x-axis is logarithmic in Figurc 12.14a and linear in Figure 12.341'.



lal



1h)

Figure 12.14 Sptcdup Factors with Instruction

pipeline to fetch both insmictions, making use of two streams. There are two problems with this approach.:

- With multiple pipelines there are contention delays for access to the registers and to memory.
- Additional branch instructions may enter the pipeline (either stream) before the original branch decision is resolved. Each such instruction needs an additional stream.

Despite these drawbacks, this strategy can improve performance, Examples of machines with two or more pipeline streams are the IBM 370/168 and the IBM 3033.

Prefetch Branch Target

When a conditional branch is recognized. the target of the branch is prefetched, in addition to the instruction following the branch. This target is then saved until the branch instruction is executed. **If** the branch is taken, the target has already been prefetched.

The IBM 360191 uses this approach,

Loop Buffer

A loop buffer is a small, very-high-speed memory maintained by the instruction fetch stage of the pipeline and containing the n most recently fetched instructions, in sequence. If a branch is to be taken, the hardware first checks whether the **branch tarRet** is within the buffer. If so, the next instruction is fetched from the buffer. The loop buffer has three benefits;

1. With the use of prefetching, the loop buffer will contain some instruction sequentially ahead of the current instruction fetch address. Thus, instructions fetched in sequence will be available without the usual memory access time.
2. **If** a branch occurs to a target just a few locations ahead of the address of the branch instruction, the target will already be in the buffer, This is useful **for** the rather common. occurrence of IF—THEN and IF—THEN—ELSE. sequences.
3. This strategy is particularly well suited to dealing with loops. or iterations; hence the name *loop bffer. If the* loop buffer is large enough to contain all **the** instructions in a loop, then those instructions need to be fetched from memory only once, for the first iteration. For subsequent iterations, all the needed instructions are already in the buffer.

The loop buffer is similar in principle to a cache dedicated to instructions. The differences are that the loop buffer **only retains** instructions in sequence and is much smaller in size and hence lower in cost.

Figure 12.l5 gives an example. of 4 loop buffer. If the buffer contains 256 bytes, and byte addressing is used, then the least significant. 8 bits are used to index the buffer. The remaining most significant bits are checked to determine **if the branch target lies within** the environment captured by the buffer.

Among the machines using a loop huller are some 0f the CDC machines (Star-100, 6600, 7600) and the CRAY-1_ A specialized form 0f loop buffer is available on the Motorola 68010, for executing a three-instruction loop involving the DBcc (decrement and branch on condition) instruction (see Problem 12.6). A three-word buffer is maintained, and the processor executes these instructions repeatedly until the loop condition is satisfied.

Branch Prediction

Various techniques can be used to predict whether a branch will be taken. Among the more common are the following:

- Predict never taken
- Predict always taken
- Predict by opcode
- Taken/not taken switch
- Branch history able

The first three approaches are static: They do not depend on the execution history up to the time of the conditional branch instruction. The latter two approachcs are dynamic: They depend on the execution history.

The first two approaches arc the simplest. These either always assume that the branch will not be taken and continue to fetch instructions in sequence, or they always.assume that the branch will be taken and always fetch from the branch tar. get. The 68020 and the VAX 11/780 use the predict-never-taken approach. The VAX 111780 also includes a feature to minimize the effect of a wrong decision. If the fetch of the instruction after the branch will cause a page fault or protection violation, the proceswr halts its prefetching until it is sure that the instruction should be fetched.

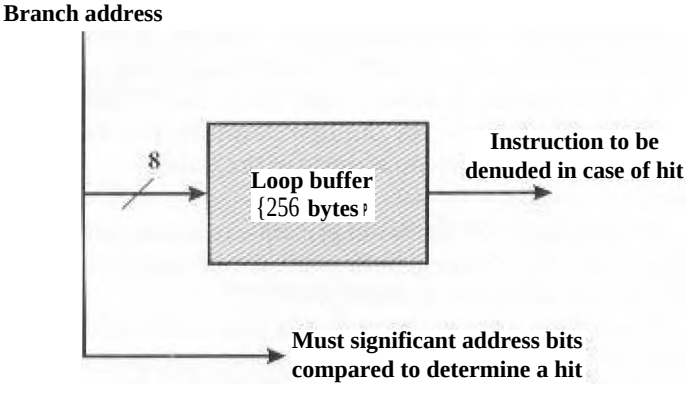


Figure 12.15 Loop Buffer

Studies analyzing program behavior have shown that conditional branches are taken more than 50% of the time [LILJ88], and so if the cost of prefetching from either path is the same, then always prefetching from the branch target address should give better performance than always prefetching from the sequential path. However, in a paged machine. prefetching the branch target is more likely to cause a page fault than prefetching the next instruction in sequence, and so this performance penalty should be taken into account, An avoidance mechanism may he employed to reduce this penally.

The final static approach makes the decision based on the opcode of the branch instruction. The processor assumes that the branch will be. taken for certain branch opcodes and not for others. fl,11-PiK1 reports success rates of greater than 75% with this strategy.

Dynamic branch strategies attempt to improve the accuracy of prediction by recording the history of conditional branch instructions in a program. For example, one or more bits can be associated with each conditional branch instruction that reflect the recent history of the instruction. These bits arc referred to as a taken/not taken switch that directs the processor to make a particular decision the next time the instruction is encountered. Typically. these history bits are not associated with the instruction in main memory. Rather, they are kept in temporary high-speed storage. One possibility is to associate these bits with any conditional branch instruction that is in a cache. When the instruction is replaced in the cache. its history is lost. Another possibility is to maintain a small table for recently executed branch instructions with one or more bits in each entry. The processor could access the table associatively, like a cache, or by using the low-order hits of the branch instruction's address.

With a single hit, all that can be recorded is whether the last execution of this instruction resulted in a branch or not. A shortcoming of using a single bit appears in the case of a conditional branch instruction that is almost always taken, such as a loop instruction. With only one bit of history, an error in prediction will occur twice for each use of the.loop: once on entering the loop, and once on exiting.

If two bits are used, they can be used to record the result of the last two instances of the execution of the associated instruction. or to record a state in some other fashion. Figure 12.16 shows a typical approach (see Problem 12.5 for other possibilities). **Assume** that the algorithm starts at the upper left-hand corner of the flowchart. As long as each succeeding conditional branch instruction that is encountered is taken, the decision process predicts that the next branch will be taken. If *a* single prediction is wrong, the algorithm continues to predict that the next branch is taken. Only if two successive branches are not taken does the algorithm shift to the right-hand side of the flowchart, Subsequently, the algorithm will predict that branches arc not taken until two branches in a row are taken. Thus, the algorithm requires two consecutive wrong predictions to change the prediction decision.

The decision process can be represented more compactly by a finite-state machine, shown in Figure 12.17. The finite-state machine representation is commonly used in the literature.

The use of history bits, as just described, has one drawback: if the decision is made to take the branch, the target instruction cannot be fetched until the target

address, which is an operand in the conditional branch instruction. is decoded. Greater efficiency could be achieved if the instruction fetch could he initiated as soon as the branch decision is made. For this purpose. more information must be saved, in what is known as a branch target buffer, or a branch history table.

The branch history table is a small cache memory associated with the instruction fetch stage of the pipeline. Each entry in the table consists of three elements: the address of a branch instruction, some number of hist ory bits that record the state of use of that instruction, and information about the target instruction. In most proposals and implementations, this third field contains the address of the target instruction. Another possibility is for the third field to actually contain the. target instruction. The trade-off is clear: Storing the target address yields a smaller table but a greater instruction fetch time compared with storing the target instruction IRECH981.

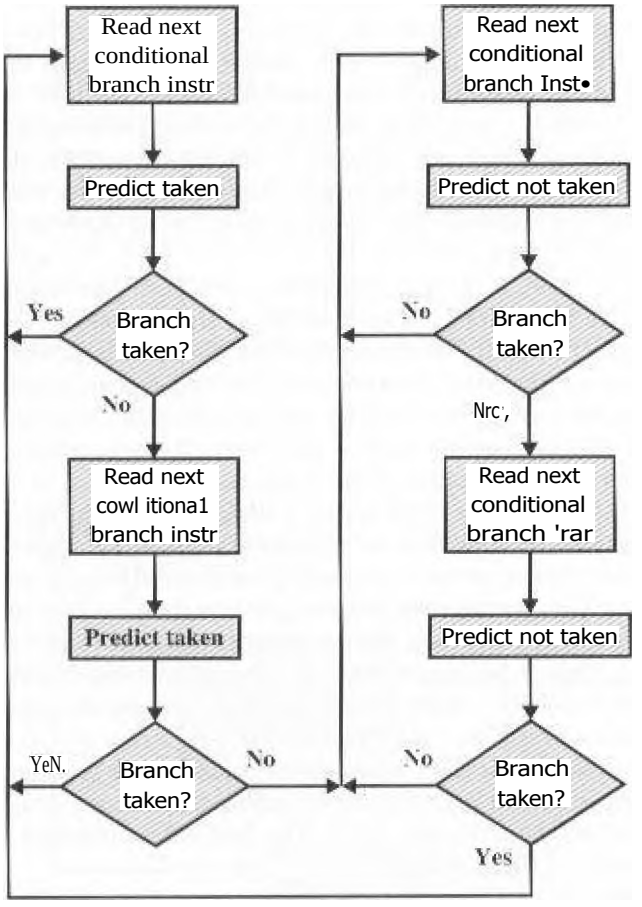


Figure 12.16 Branch Prediction Flowchart

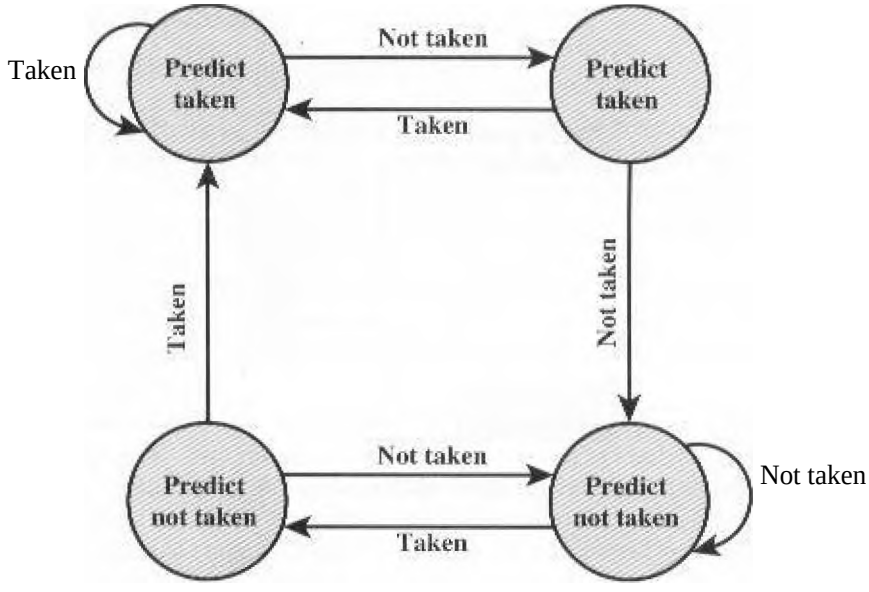


Figure 12.17 Branch Prediction State Diagram

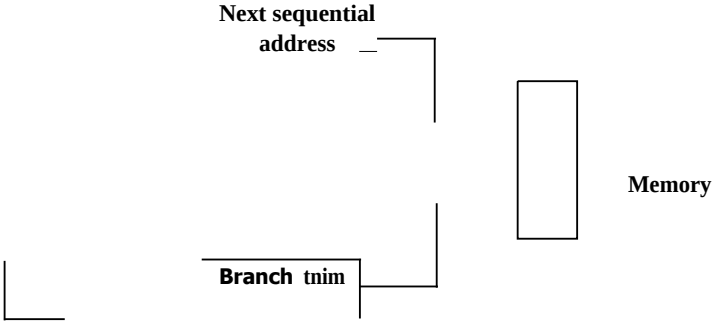
Figure 12.18 contrasts this scheme with a predict-never-taken strategy. with the former strategy, the instruction fetch stage always fetches the next sequential address. If a branch is taken. some logic in the processor detects this and instructs that the next instruction he fetched from the target address (in addition to flushing the pipeline). The branch history table is treated as a cache. Each prefetch to a lookup in the branch history table. If no match is found, the next sequential address is used for the fetch. if a match is found, a prediction is made based on the state of the instruction: Hither the next sequential address or the branch target address is fed to the select logic.

When the branch instruction is executed, the execute stage signals the branch history table logic with he result. The state of the instruction is updated to reflect a correct or incorrect prediction. lithe prediction is incorrect, the select logic is redirected to the correct address for the next fetch. When a conditional branch instruction is encountered that is not in the table, it is added lo the table and one of the existing entries is discarded, using one of the cache repl;icernent algorithms discussed in Chapter 4.

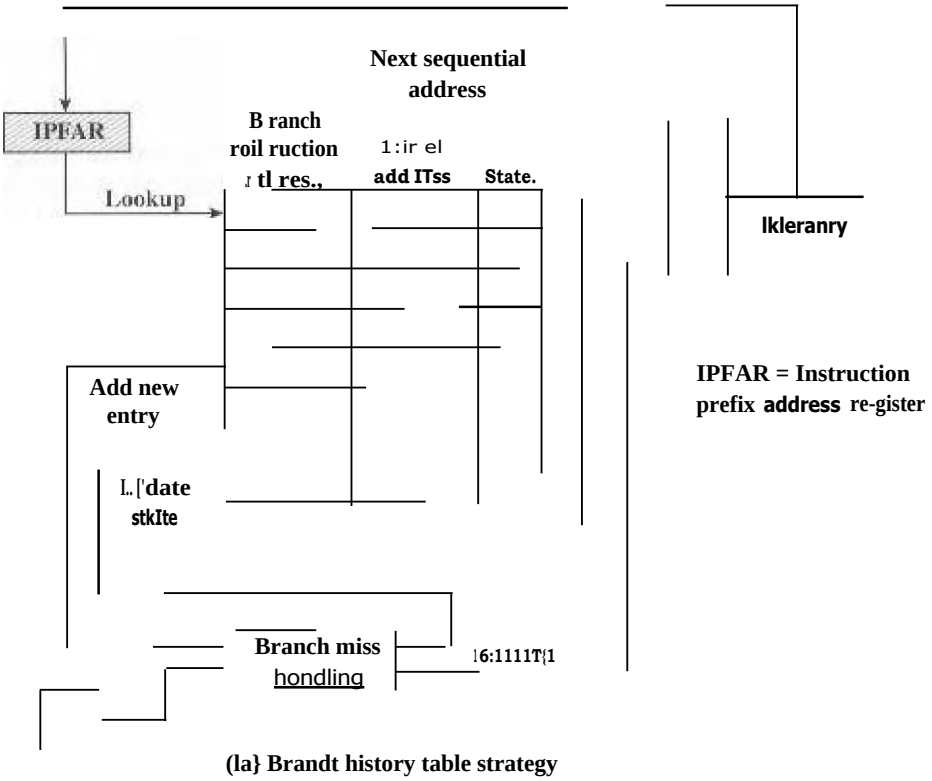
One example of an implementation of a branch history table is the Advanced Micro Device AMD,2[1])0911 microprocessor.

Delayed Branch

It is possible to improve pipeline performance by automatically rearranging instructions within a program, so that branch instructions occur later than actually desired. This intriguing approach is examined in Chapter 13.



(a) Predict never Ia ke.i strategy



(la} Brandt history table strategy

105 re. I2.18 Dealing with Brunches

Intel 80486 Pipelining

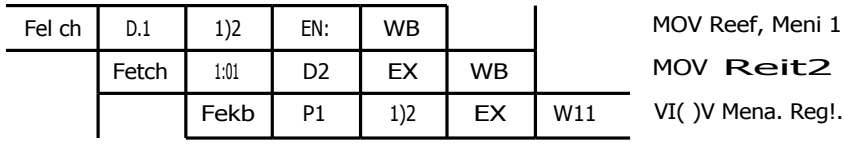
The 80486 implements a five-stage pipeline:

- Fetch: Instructions are fciched from the cache or from external memory and placed into one of the two 16-byte prefetch buffers- l'he objective of the fetch stage is to fill the prefetch buffers with new data as soon as the old data have been consumed by the instruction decoder. Because instructions are of variable length (from 1 to 11 bytes not counting prefixes). the status of the prefetoher relative to the other pipeline stages varies from instruction to instruction. On average, about five instructions are fetched with each 16-byte load [CRAW-)01. The **fetch stage** operates independently of the other stages to keep the prefetch buffers full.
- **Decode stage 1:** All c.)pcbde and addressing-mode information is decoded in the D I stage. The required in formation, as well as instruction-length information, is included in at most the first 3 bytes of the instruction. Hence. 3 bytes are passed to the DI stage from the prefetch buffers. The D1 decoder can then direct the D2 stage to capture the rest of the instruction (displacement and immediate data), which is not involved in the 1)1 **decoding.**
- Decode stage 2: 'File D2 stage expands each opcode into control signals for the It also controls the computation of the more complex addressing modes.
- **Execute:** This stage includes ALU operations. cache access, and register update.
- **Write back:** This stage, if needed, updates registers and status flags modified during the preceding execute stage. If the current instruction updates memory, the computed value is sent to the cache and to the bus-interface Write hurlers at the same time.

With the use of two decode stages, the pipeline can sustain a throughput of close to one instruction per clock cycle. Complex instructions and conditional branches can slow down this rate.

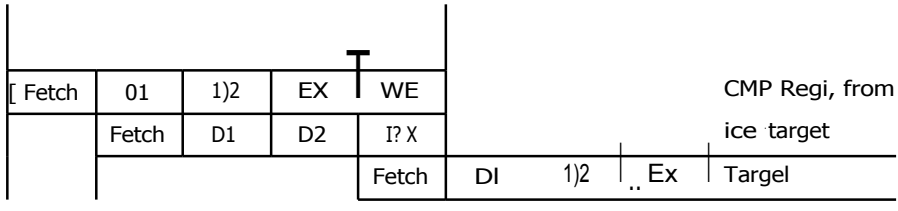
Figure 12.19 shows examples of **the** operation of the pipeline. Part a shows that there is no delay introduced into the pipeline when a memory access is required. However, as part h shows. there can be a delay for values used to compute memory addresses. That is, if a value is loaded from memory into a register and that register is then used as a base register in the next instruction, the processor will stall for one cycle. In this **example. the** processor accesses the cache in the EX stage of **the first instruction and** stores the value retrieved in the register during the WB stage. However, the next instruction needs this register in its D2 stage. When the D2 stage lines up with the WB stage of the previous instruction. bypass signal paths allow the D2 stage to have access to the same data being used by the WB stage for writing, saving **one pipeline** stage.

Figure 12,19c illustrates the timing of a branch instruction, assuming that the branch is taken. The compare instruction updates condition codes in the WB stage, and bypass paths make this available to the EX stage of the jump instruction at the same time. In parallel. the processor runs a speculative fetch cycle to the target of the jump during the EX stage of the jump instruction. If the processor determines



i a) No data load delay in the pipeline

b) Pointer load delii:1;



lei Branch instruction timing

'Figure 12,19 8C)486 Instruction Pipeline Examples;

a false branch condition, it discards this prefeteh and continues execution with the next sequential instruction (already fetched arid decoded).

12.5 THE PENTIUM PROCESSOR

An uliervim ur the Pentium 4 proi..,.:N...;or organization is depicted in Figure 4.13. In this section, we. examine some or Lift details.

Regist er Organization

The register organization includes the k.1I':)wing types of registers (Table 12.1):

- General: There arc eight 1[1].2-bit general-purpose registers (see Figure 2.30. Thew rnity be used for all types or Penli urn instructions they can also hold

operands for address calculations- In addition, some of these registers also serve special purposes:For example, string instructions use the contents of the. ECX, ESL and **EDI** registers as operands without having to reference theoe registers explicitly in the instruction, As a result, a number of instructions can be encoded more compactly.

- Segment: The six](-bit segment registers contain segment selectors, which index into segment tables, as discussed in Chapter 8. The code segment (CS) register references the segment containing the instruction being executed. 'OK stack segment (SS) register references the segment containing a user-visible stack, The remaining segment registers (DS. ES . FS, GS) enable the user to reference up to four separate data segments at as time,
- **rings: The** EFLAGS register contains condition codes and various mode bits.
- **Instruction pointer! C.'onlairff** the address of the cuiTent instruction.

There are also registers specifically devoted to the floating-point unit:

- **Numeric:** Each register holds an extended-precision 80-bit lioating-point number..J'here are eight registers that function as a stack, with push and pop op-erations in the instruction set.
- **Controi:** The 16-bit ennirol register contains bits **that** control the operation of the floating-point unit, including 1he type of rounding control, single, double, or extended precisiow and bits **to enkil plc** or disable various exception conditions.

'able **12.1** Pe.ratiurn. Processor gegiskers

(a) Integer Unit

	Number	Length (bits)	Purpose
GorterfiI		32	GeneraI-purpow user registers
Scgmerki		[6	Contain sr Jsyner31. sc.leck ors
Flap,		**32**	**Si Hlus** and contra! bitr
Instructio n poin ryr		3.	poixitcr

(b) Floating-Point Unit

Type	Iti^rrn **her**	Length (bitsi	PU tp tilsic
Numeric	S		Hold flokitin-point numbers
Control		16	Cond ea! bits
Status		16	Slaws hits
'rag wort]	1	ti	sKcificts contuyis)f mune ill: rcui)..tcr)..
Ins ıuıaı i on pointer	1	48	Fob ts to LEIS traction in tcrrup hV exception
D ⅃U [Join i			Vpirst5 to operand iilLel rup Lud by 12x.12c.pitiort

- **Status:** The 1.6-bit status register contains bits that reflect the current stale of the floating-point unit, including a 3-hit pointer to the top of the stack; condition codes reporting the outcome of the last operation: and exception flags,
- **Tag word:** This I6-bit register contains a 2-bit tag for each floating-point numeric register, which indicates the nature of the contents of the corresponding register. The four possible values are valid, zero, special (NaN, dcnormalized), and empty. These tags enable programs to check the contents of a numeric register without performing complex decoding of the actual data in the register. For example, when a context switch is made.. the processor need not save any floating-point registers that are empty.

The use of most of the aforementioned registers is easily understood. Let us elaborate briefly on several of the registers.

EFLAGS Register

The EFLAGS register (Figure .12.20) indicates the condition of the processor and helps to control its operation. It includes the six condition codes defined in Table 10.8 (carry. parity, auxiliary, zero, sign. overflow), which report the results of an integer operation. In addition, there are bits in the register that may he referred to as control bits:

- **Trap flag (TF):** When set, causes an interrupt after the execution of each instruction. This is used for debugging.
- **Interrupt enable flag (IF): When** set. the processor will recognize external interrupts.
- **Direction flag (DF):** Determines whether string processing instructions increment or decrement the 16-bit half-registers SI and DI (for 16-bit operations) or the 32-hit registers CSI and EDI (for 32-bit operations).
- **1/0 privilege flag (IOPL):** When set, causes the processor to generate an exception on all accesses to I/O devices during protected-mode operation.
- **Resume flag (0**[i]**):** Allows the programmer to disable debug exceptions so that the instruction can be restarted after a debug exception without immediately causing another debug exception.
- **Alignment cheek (AC):** Activates if a **word** or douhleword is addressed on a nonword i r nondoubleword boundary_
- **Identification flag (ID):** If this bit can be set and cleared. then this processor supports the (PhD instniet ion. This instruction provides information about the vendor, family, and model.

In addition, there are 4 bits that relate to operatin2. mode. The nested task (NT) flag indicates that the current task is nested within another task in protected-mode operation. The. virtual mode (VIA) hit allows the programmer to enable or disable 'virtual 8086 mode, which determines whether the processor runs as an 8086 machine. The virtual interrupt flag (VIF) and virtual interrupt pending **(VIP)** flag are used in a multitasking environment.

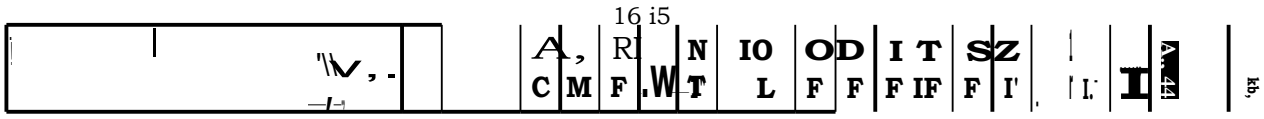


Figure 12.20 Peatitan II EFLAGS RyQC

Control Registers

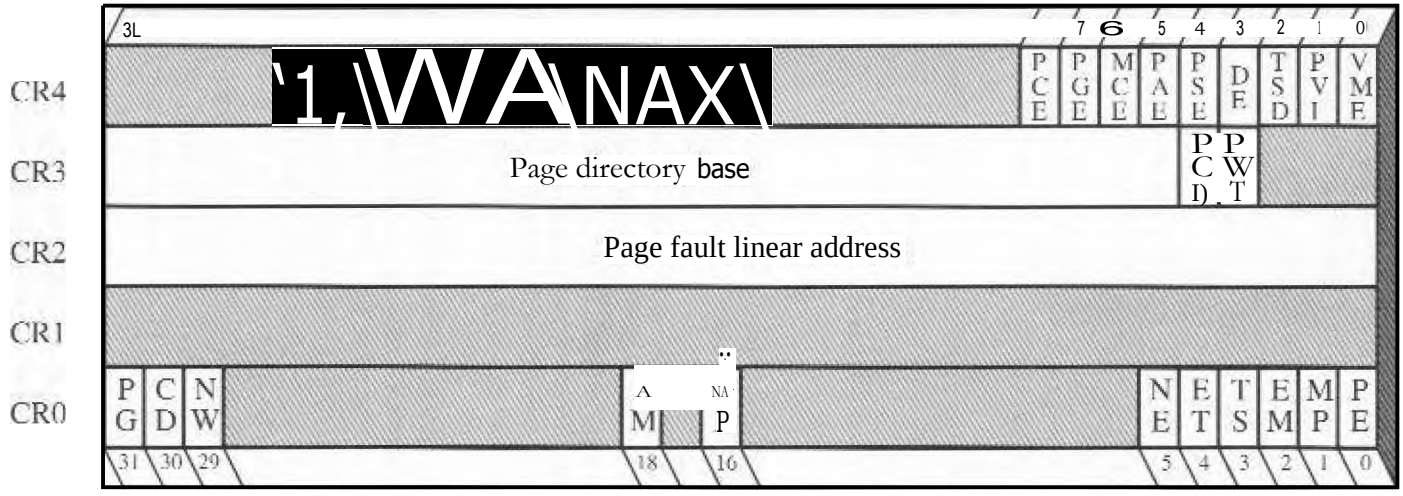
The Pentium employs four 32-hit control registers (register CR1 is unused) to control various aspects of processor operation (Figure 12.21). The CR0 register contains system control flags,.which control modes or indicate states thAt apply generally to the processor rather than to the execution of an individual task. The flags are as follows:

- **Protection enable (PE):** Enableidisable protected mode of Operation.
- rvlonitor **coprocessor (MP): Only** of inlerest when running programs from earlier machines on the ['curium:, it relates to the presence of an arithmetic coprocessor.
- **Emulation (EM):** Set when the processor does not have a floating-point unit, and causes an interrupt when an attempt is made to execute floating-point instructions.
- **Task** switched (TS): Indicates that the processor has switched tasks.
- Extension **type (ET):** Not used on the Pentium; used to indicate supporta math coprocessor instructions on earlier machines.
- **Numeric error (NE):** Enables the standard mechanism for reporting floating-point errors on external bus lines.
- **Write protect (WP):** When this bit is clear. read-only user level pages can be written by a supervisor process. This feature is useful for supporting process creation in some operating systems.
- **Alignment mask (AM):** Enables/disables alignment checking.
- **Not Write through (NW):** Selects mode of operation of the data cache. When this bit is set, the data cache is inhibited **from** cache write-through operations,
- **Cache disable (CD):** Enablesidisables the internal cache fill mechanism.
- **Paging (PG):** EnablesidiSables paging.

When paging is enabled, the CR2 and CR3 registers are valid. The CR2 register holds the 12-bit linear address of the Iasi page accessed before a page fault interrupt. The leftmost 20 bits of CR3 hold the 20 most significant bits of the base address of the page directory; the remainder of the address contains zeros. 'Two bits of CR3 are used to drive pins that control the operation of an external cache. The page-level cache disable (PC,D) enables or disables the external cache, and the page-level writes transparent (PWT) bit controls write through in the external cache.

Nine additional control bits are defined in CR4:

- **Virtual-8M mode extension (VME):** Enables support for **the** virtual interrupt flag in virtual-8086 mode,
- **protected-mode virtual Interrupts** (PVI): Priables support for the virtual interrupt flag in protected mode.
- Time **stamp disable (TSD):** Disables the read from lime stamp counter (RDTSC) instruction, which is used for debugging purposes.



PCE — Performance counter enable
PUE = Page global enable
MCE — Machine check enable
PALE = Physical address extension
PSE — Page size extensions
DE — Debug extensions
TSD = Time stamp disable.
PVT — Pageetctl mode virtual interrupt
VN1L = Virtual 8086 mode extensions
PCT) — gc-level cache disable
PWT = Page-level writes transparent

PO = Paging
CD — Cache disable
NW — Not write through
AM — Alignment mask
WP Write protect
NE Numeric error
ET Extension type
TS = Task switched
FM — Emulation
NIP = Monitor coprocessor
PE — Protection enable

Figure 12.21 Pentium II Control Registers

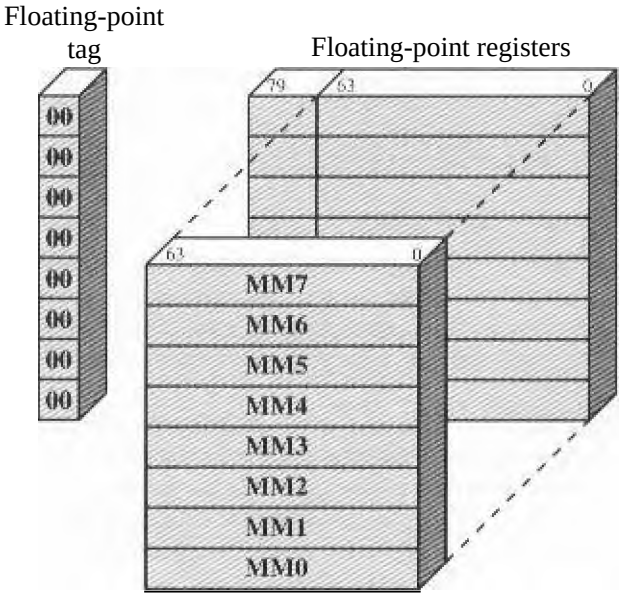
- **Debugging extensions (DE):** Enables 110 breakpoirn s; This allows the processor to interrupt on **110** reads and writes.
- **Page size extensions (PSE);** Enables the use of 4-Mbyte pages when set in the Pentium or 2M-byte pages when set in the Pentium Pro and Pentium,
- Physical **address extension (PAC): EnabIcs** address lines A35 through A32 whenever a special new addressing mode, **controlled** by the PSE, is enabled **for the** Pentium Pro and subsequenl Pentium architectures (Pentium ii through Pentium 4).
- **Machine check enable (NICE):** Enables the **machine** check interrupt, which occurs when a data parity error occurs during a read bus cycle **or** when a bus cycle **is** not successfully completed.
- **Page global enable (PG E);** Enables the use of global pages. **When** POE =1 and x task switch is performed. all of the. **TLB** entries are flushed with the exception of those marked global.
- **Performance cannier enable (PCE)i Enables the** execution of the RD.PMC (read performance **counter)** instruction at any privilege level. Two perfor-

manee counters are used to measure the duration of a specific event type and the number of occurrences of a specific event wpe.

MMX Registers

R.c.E.:4]I I from Section 10,3 **Lhai** the Pentium **MMX** capability makes use of several 64-bit **data types. The MMX** instructions make use of 3-bit register address fields, so that eight MMX registers are supported, In fact, the processor does not include specific **WAX** registers. Rather, **the** processor uses an aliasing technique (Figure 12.22). The existing floating-point registers are used to store MMX vmuss, Specifically, the low-order 64 bits (mantissa) **a** each floating-point register are used to form the eight MMX registers. Th115. the existing Pentium a rchitecture is easily extended to support the MMX eapabiliiy. Sonic key characteristics of the MMX use of these registers are as follows!

- Recall that the floating-point registers are treated as a stack for floating-point operations. For MMX operations, these same registers are accessed directly.
- The first time that an MMX instruction is cNeeuted after any floating-point operations. the FP tag word k marked vaiid. This reflects the change from stack operation to direct register addressing.
- The LMMS MMX State) instruction sets bits of the **FP** Lag word to indicate that till registers are empty. It is importanl that I be programmer insert this instruction al the end of an IvINIX code block so that subsequent floating-point operations function properly.
- When a value is written to an MMX register, bits [79:64] of the corresponding FP register (sign and exponent bits) are set **to al]** ones. This sets the value in the FP register to NaN (not a number) or infinity when viewed as 8 fl oating-point value. This ensures that an MMX data value will **not** look like a valid floating-point value.



Figurt. 12-22 klapping of MIX IZisicrs to Floating-Point Registers

Interrupt Processing

interrupt processing within a processor is a facility provided to support the operating system, Il, 4illows an application program to be suspended. in order that a variety of interrupt conditions can be serviced and later resumed.

Interrupts and Exceptions

Two classes of events cause the Pontiurri to suspend execution *of* Lilo current instruction stream and respond to the event: interrupts and exceptions. In both cases, the processor &Ives the context of the current process and transfers to a predefined routine to service the condition. An *interrupt* is generated by a signal from hardware, and it may occur at random times during the execution of a program. An *exception is* generated from software, and it is provoked by the execution of an instruction- There are two sources **of** interrupts and two sources of exceptions:

Irtl urtipIs

- Maskable interrupts: Received on the procxssor's INTR pin. 'ffie processor does not recognize a mask able interrupt unless the interrupt enable flag (IF) is set.

a **Nonmaskablc interrupts:** Received on the processor's NMI pin. Recognition of such interrupts cannot be prevented.

2. Exceptions

- **Processor-detected exceptions:** Results when the processor encounters an error while attempting to execute an instruction.
- **Programmed exceptions:** These are instructions that generate an exception (INTO, INT3. INT. and BOUND).

Interrupt Vector Table

Interrupt processing on the Pentium uses the interrupt vector table. Every type of interrupt is assigned a number, and this number is used to index into the interrupt vector table. This table contains 256 32-bit interrupt vectors, which is the address (segment and offset) of the interrupt service routine for that interrupt number.

Table 12.2 shows the assignment of numbers in the interrupt vector table; shaded entries represent interrupts, while nonshaded entries arc exceptions. The NMI hardware interrupt is type 2. CNITR hardware interrupts arc assigned numbers in the **range of 32 to 255; when an INTR interrupt is** generated, it must be accompanied on the bus with the interrupt vector number for this interrupt. The **remaining vector numbers are used for** exceptions.

If more than one exception or interrupt is pending, the processor **services them in a** predictable order. The location of vector numbers within the table does not reflect priority_ instead, priority among exceptions and interrupts is organized into five classes. In descending order of priority, these are

- **Class 1; Traps on** the previous instruction (vector number 1)
- **Class 2: External interrupts** (2. 32 255)
- **Class 3: Faults from fetching next instruction (3. 14)**
- **Class 4: Faults from decoding** the next instruction (6, 7)
- **Class 51** Faults on executing an instruction (a 4, 5, 8. 10-14, 16. 17)

Interrupt Handling

Just as with a transfer of execution using **a CALL instruction,** a transfer to an interrupt-handling routine uses the system stack to store the processor state. **When an interrupt occurs and is recognized by** the processor, a sequence of events takes place:

1. If the transfer involves a change of privilege level. then the current slack segment register and the. current extended stack pointer (ESP) register are pushed onto the **stack.**
2. **'Mc current value of the EFLAGS register is pushed onto the stack.**
3. **Both the interrupt (IF) and trap (TF)** flags are cleared. This disables INTR interrupts and the trap or single-step **feature.**
4. **The current code segment (CS) pointer and the current instruction pointer (IP or ELF) are pushed onto the stack,**

5. it the interrupt is accompanied by an error code, then the error code is pushed
6. onto the stack.

   The interrupl vector contents are fetched and /oaded ink) lhe CS and 11[3] or EIP regktcm Execution continues from the interrupt service routine.

To return from an interrupt, the inlerrupt service routine executes an **IRET** instruction..rhis causes all of the values saved on the stack to be rcslored; execution resumes from the point of the interrupt.

Table 12.2 Peritiuni Exception and Ink: rrupt Vector Table

'Vector Number	Demription
	Divide error. divi OverilLyw (11" CilVi Si{ In ti' /Cr{
l	Dul. ' ttg ukceptit)r); include.; v,lrinu 11111L5 rinil trap:' rL IHIL it 11 4|4•hugging
	NMI pin ittcv˜rupt:
	⌈5,.,;)kpilini; eau NC LI I N r ;I instruction, which is a 1-byte ins tilli:LiOrt Ur,811.11 for de htiy fan.g
4	INTO-fictEctc4 overflow:: occurs when the processui executes INTO with the OF !lap_ (LL
5	Ni) range exceeded; the SOUND inArUCtion CunIpqn˜cs regislr wilh atod-stored in memory and generates an interrupt if the uontcnts of the retrixtu˜ is ulit 4.)t' hounds-
6	Unacrined opcodc
7	Devi.= no available; asternpt to use ESC or WAIT iiistraLtion fails due to lack of external dOviCe.
	DClul ' lo fault; two inteirilptS fiCeLly durinti. the si.17112 ins!" uclicm nntt erinncil he handled serially
c)	Reserved
11)	Invalid task stale 2. grOcril; sezrunt descrilling a requested task is not inicialized or not valid
11	Segment not present; requircd xcgrnc.nl no! procns
12	Stack fault: limit c,1 stack 6L gmaint c r:ctcdcd pa˜ stack segment not present
13	General protecti4 prol..:.cLi on violation that does not cause another exception (e.g.. wrisisig Li) it rcad-cm1:!.. segrnv1t)
	Pauc fa ult
15	Rusurvcci
L6	Rod rl g-point error:, generated by a tloating-point aritUute tic instruction
17	Alignment check: access to a word scored at an odd hvtc uddress. DT a doublIvoiorcl slort:ci if an address not a i»ulLiple of 4
IS	VI:IChule check: model specific

vecioN, provided wlicis IN'ER signal is act i .16

t!=p,3˙.atiod;
Shatloct; interrtrpts

12.6 THE POWERPC PROCSSOR

An overview of the Powerl[3]C processor organization is depicted in Figure 4.14. In this section. we examine some of the details of the 64-bit implementation.

Regis ter Organization

Figure 12.23 depicts the user-visible registers for the PowerPC. The fixed-point unit includes

- **General: There** are thirty-two 64-bit general-purpose registers. These may be used to load, store, and manipulate data operands and may also he used for register indirect addressing. Register 0 is treated somewhat differently. For load and store operations and several of the add instructions, register 0 is treated as having a constant value ,!:ero regardless of its actual contents.
- **Exception register (XER):** Includes 3 bits that report exceptions in integer arithmetic operations. This register also includes a byte count field that is used as an operand for some string instructions (Figure 12.23a).

The floating-point unit contains additional user-visible registers:

- General: 'there are thirty-Iwo 64-bit general-purpose registers, used for all floating-point operations.
- **Floating-point status and control register (FPSCR):** This 32-hit register contains bits that control the operation of the floating-point unit and bits that record the status resulting from floating-point operations (Table 12.3).

The branch processing unit contains these user-visible registers:

- **Condition register:** Consists 0r eight 4-bit condition code fields (Figure 12.24b).
- **Link register:** The link register can he used in a conditional branch instruction for indirect addressing of the target address. This register is also used for call return behavior. If **the LK** bit in a conditional branch instruction is set, then the address following the branch instruction is placed in the link register, and it can be used for a later return.
- **Count: The** count register **can be used** to control an iteration loop, as explained in Chapter 10; the count register is decrernented each time it is tested in a conditional branch instruction. Another use for this registoi is indirect addressing of the target address in a **branch instruction.**

The fields of the condition register have a number of uses. The first 4 bits (CRO) are set **for all** integer arithmetic instructions for which the. Re **bit is** set. As 'Fable 12.4 shows, the field indicates whether the. result of the operation is positive. negative. or zero. The fourth bit is a copy of the summary overflow bit from the XER. The next field (CR1) is set for all floating-point arithmetic instructions for which the Re bit is set. In this case, the 4 hits are set equal to the first four hits of the. FPSCR (Table 12.3). Finally. the eight condition fields (CRO through CR) can

aid•
Exception
31
0
R31
R0
63
0
Link
Count
63
0
31
0
FPR31
FPSCR
31
0
63
0

IaI'Pie 123 PowizrPC 110k1iti n-Poinit Status and Cunirel Rc..!gista

Bit	Definition
4l	Exception summary. 54:1 i f ank 42xcupLiort occurs; remains sot until resuL **by** r1**ware.**
	Ertable-d exception 94Arn ni pry. Sct if uny enabEed exceptloo has occurred.
	Invalid operation xcer..riCS11 **summary.** Set if an operation cx..:;;:•rilion has occurred,
3	Overflow excepciOn. MrignitiRic tA rtsoif **ExcetC14** COO be repi·43s.:mi
4	Undutflow excepdon. Result is to sm.:01 to by n ni
5	Zero divide ex&eption.. Divisor is zero and divtdc.nd i5 finite Eionwro.
	| nEXNC1 exexplicin. Reloaded result ciiii4A ·sirorn tn1cgirrncd,a-L rum.di of an werflow occurs with 45vcr flow In.C.121}Lic)IL disabled.
	opera ticrn exception. 7!'signaling NaN; C; 9: i -:-•x•): la (1) 11: (..K X11): E2: **i:4.5(E)paviso)1** i vOlv big MN.
13	Fraction roulalud. Reatiadinp cif the Nstili ilicrcl ·ncnied the CI-adjust.
1.1	Fraction inexact. **ROuthie.d** tesulLcItaitc.s fraciion or an civerflOW occurs wit El overflow eNct.ption disabled.
I :Ii)	Rcstl11 flap, Five-bit code specifies less than. greater Ihan. equal, unordered, quiet NaN, ±norntoli.k•ed. Idenonualized. -ttt
20	kcJim'vetl.
1.-7;	| m'tiIti3 opm-altos Incuptilm. 21: sortwaro r equi3st; RILpiru **root i5t**' n number: 1nwii....r **k'45111VT'iL.1111111.151.1'111.E4** a number. an coccncIv, or **a** NECK
2.4	Envalid **vxccpLion.**
25	()willow .....xcc'pi **yin** citab1.3 . y.
26	Uudevilow excQp1inn 12nd ilk
27	Zero c
28•	Itiexa•a exceptiou
-W;31	Rounding con1J·t)1. Two-hit &:·!dc specifics to S10 itCSr, l.4 **P9r'Ard I·CMa.id** cc. k•Atril

IL nth l. .l

be used with a compare instruction; in **each case, the identity of lhe field is specified in the imItnicion** itself- pot **both fixed-point and floaling-point compare instructiorm, the firs.t 3 hits of the designated condition field record whether the lint operand is less than, greater than. or Lqun I to the second operand. 'the fourth hit is the summary overflow bit for a 1i:will-point compare. and an unordered indicator for a floating-point coraparc.**

In terrupt Processing

As with any processor, the PowerPC includes a bait): that enables the processor to interrupt the currently executing program to deal with an exception condition.

Types of Interrupts

lateimpts on a PowerPC are classified a those caused by sonic. system condition or event and those cauwd by the execution of an instruction. Table 12,5 lists the inierrupis recognii.ed *by* the PowerPC.

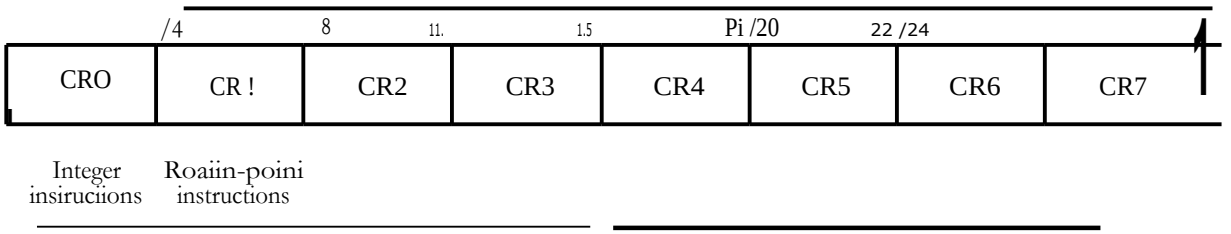
SO Summary overflow: set to **I to indicate that an** overflow occurred during the exection ol an instruction; remains 1 until reset by software

OV = Overflow: set to i to indicate that an overflow occuned during the exectiou of aninstruction; reset to 0 by next instruction if there is ilo overflow

CA Cagy: set to 1 co indicnie curry out of bit 0 during the execution of an instruction

Byte count = Specifics number of bytes to he transferred by Lii.a&Store String indexed instruction

(a) Fixed-point exception register (XER)



(b) Condition register

Figure 12.24 Po% eriir Register Formats

Thble 12.4 Interpretation of Bits in Condition Rogithtr

Bit position	t lto iintrger irmtrucliall milli Rr•=111	CR1 itluuthig•point instructiou with Re · 1)	riti (fixoct•point compare instradialli	CRi (floating-point C(mkipate insiructiou)
r	*result < CI*	Exception summary	op[---: 01)2	op I < or.,'
1: - I	teiult > 0	Enabled exception summary	opt •> up2	opl :--• opv
i - 2.	r sull = it	ILI:valid operaiiori excepti CM 2.1E1711 nary	op! = opa	opl= opt
i ' 3	Summary overilow	Ovifrilow exception	Summary uverflov	l :n45rdcred {one operand ig Fi Nnl\r)

mos!, or the interrupts listed in the table. are easily understood. A few warrant furthe I comment. The system reset interrupt happens at power on and when the reset button on the system unit is pressed, and it causes the system lo reboot. 'The machine check in lerrupt deals with certain anomalies, such as cache parit!,' error and referc nce to a no: Le X istent memory location, and ma}' Lh.i2 system to enter what is known as a checkstop state; this stale. SLINpc.nd!i processor execution and frc4ze.5 the contents of registersunlil a neberal _ rr`hc floating-point assist enables the proCCS-sor to invoke s.oftw.are routines to complete operations that cannot be handled directly by the floating-point unit. such as those involving denormaEized nunabcrs or unimplemented pooling-point opcocles.

Machine State Register

Fundarnenui I to the. interruption of a program is the Ability to recover the state of the prOfXS7:1{) I" at the ti me of the interrupt. This includes not only the contents of the various registers but also various control conditions reEating to execution. These conditions are conveniently summarized in I he. WISR (Table 12.4 Again, sevcrai of the bits in this registor vi4]rrant furtha comment.

When the privilege mode bit (bit 49) is set, the prucxssor .6; operating at a user privilege level. Only a subset of the i n7.1 ruction set is available. When thc hit is cleared. the processor operates at supervisor privilege lave!. This enables all of the instructions and provides access to certain system registers (such as the rvISR) not accessible from the user privilege level.

The values of the two floaling-point exception bits (bits 52 and 55) define the types of interrupts that the floating-point unit may generate. The interpretation iE as follows:

MO	FE1	Interrupts that will be recugnind
0	11	Nurse
0	I	Ira prc.cl4r2 TlinITZ'CC I'elL' Fel LJI.0
l	It	Imprecise recoverable
[	l	Precise

Table 12.S 1[3]owerPC Inierrupt Tab[c.

Entry Paint	Interrupt Type	Description
	R er..urved	
00 1.1)01i	Syqtern	**AL:wrtion of iho pfooesSor's hard or soft reset input si otAU by S** :Os!rnal Tone
C.C.21.klh	.kfachi chick	2!w!rtio.n. of l'EA# to the procesH3r it cliabEcAt elteuks
LXJ3U[1]Th	/..1 La s ctruge	Exampks: data page tau]}; i ielstti vicil id ion 15n load/store
.10400E1	InstrucLion Stara	Code page rata, attempted instruction fetch.rum 1.0 me t! 1.1:1:12NN old L i on
p 1.1-11.fr ex .err MV*EIE'r?rAeCrr .,errr		A&Rerti on **or** Llsc proce.ssor's vxr12r.nal intrrr jiL inert signal Ely external l I i. whoa exi...f? intiirrupt ...recognision
006001.E	A 'rn mutt	1..nsuceeskil mem pc (i access mentory due to rnis-2.1igncd operand
007001)	Prckgtarn	Floating-poilil illSerrupt; **user attempts** to encodiL puivil.c..aed **instruction: trap instruction executed with spccified xtidition met:** instruction
04.181.wa	Fl n;.7.-pc Fin 1 tt:ia	A1.1.011p I 10.•!xecuL4 IIcii L **R-poi instrucci on with 0.:inisg poi:ni unit (lisablo.1**
IX.190Qh		itX.11)1.43ginD Of the deererne tit reeiqe:r %visors external **recognttioii is** C.Liablat
IXIA01)h	Rewryucl	
0013	Reserv:L.d	
000.[Oki	System ca I I	Execution of a system call instruction
00D0111	TrHcu	Single-step or branch trai:e interrupt
I.E0011	**Fl ing- potet assist**	Atrempt Lo 1..xecuLe vety n fr.:(3u4231 I , complex 11oatirg-point ope ru Lion tc.g.. opurni ion on dcaor-inalized number)
DOE I Dh Lh rc Fttzh OCFFFh	**Psewr.:c**	
01010h through 02.FFFEk	impl cm En i s ' 111) speci fir)	

1:13Fluiicn! inc,frui,cx by insiriation cimlutiou
h ruo.76.1pt.5 1 cil mod 0.xecution

When the single-step trace Ht (bit 53) is set ; the processor branches to the trace interrupt handler after the successful completion ofc4ieh instruction, When the branch trace bit (bit 54) is set, the processor branches to the branch trace interrupt hondler after the successful completion ()I' each branch instruction. whether or not the branch wuslaken,

The instruction addre 'is Irlini7llation (bit 58) and data address translation (bit 59) determine whether real ridclressing is used or whether the memory-management unit performs address translation.

Table 1245 PowcrPC Machim St e Rogkter

Bit	
	ProunSLI i' mode
1.44	
4:5	Power rhartriol.rneln 41}1 ditlis4E)142:d .rfecr"
46	Inn l.eineututiou dependent
47	Define!, whether interrupt hand1:6 173: 0243004: ittle-endian mode
48	Exte.i.noi unnIlled;disabled
	iIcgc dlric pri Ante
50	Flouting-point unit avaiiahle.kina•vailatik
51	Machine check inwrrLtros
	FI no-9tine-r1cunL uxue NMI 111013e.
5.3	Sin0e.-scep trace eanblediclisablaci
54	Branch trncia cnnbl.cdirivRhled
	Ficmiin-poilltuxceptiun Tri.Odo l
5fi	Rum:pi:ea
	Most significant part of v c..;2ption odtlmsr.. is 0C0131FFF11
	iiistrucalun address li8n.561,1urs ctn. iff
5L)	[Mtn address LikSrimlutiOn.
60:61	**Re.siL.ro2c1**
62	insert:up t is remvc in hi cin onr=oviLrahl
63	Proc.osiiar is in hig nairiii!littl-endian mode

It I

StKio.I. :1;1.. or R I

Interrupt

When an interrupt occurs and is recognized by the processor, the following sequence of events. takes place. ;

1. The processor pla..2.; the address of the instruction to he excepted ne..xt in the Save/Restore Register 0 (SRRO). This is t he address of the currently executing instruction if the interrupt was caused a failed attempt to eWeLILL:: that instruction; otherwise, it is the address of the next instruction to be executed ;11' er t he uurrunt instruction.
2. The processor copies machine state COMatiOn from the N1SR to thc Sake, Restore Register 1 (51-t R1). The bits that are depicted as unshaded in Table i7.6 are copied. The retnaining bits of SRR I RTC. loaded with information specific to the interrupt type.
3. I'he :VI SR is set to a hardware-defined value specific to the interrupt type. For all interrupt types, address translation is turned off and external interrupts are d isabled,
4. The processor then transfers control to the appropriale interrupt handler. The addresses of the interrupt handlers are stored in the interrupt 'rabic!

(Table [2.5). The base address of that table is determined by bit 57 of the MSR.

To return from an in terrupi, 1hr,: interrupt se:rvicc routine executes an rfi (return from interrupt) instruction. This causes the bit i,ralties saved in SRR1 to he, restored to the MSR. Execution resumes at the localion slored in SR 12{1"

P f7, RR c ONIN4NPEP RE41),INg

EPA TTO1 I and EMOSHID1 1 provide excellent coverage of ih pipelining issues discussed in this chapter. [HENN91 .1 and [HWAN93] contain detailed discussions of pipelining. [50F119[1] provides an excellent. detailed discussion of the hardware design issues involved in an instruction pipeline,

[EN/ER 1.11 examines the evolution of branch prediction s(ralegies. [CRAC192] is A detailed 91.11 **4** of branch prediction in instruction pipelines. [LTL.:BE9 11 and [L.11181 examine various ',ranch prediction straicgies [hat can be used (o (111.e perforinn nee of [KAI:1,911 in:at:nines the dirficulvy introduced into branch prediction ini-r Viii (-**& targe** t addrelss is variable,.

Ite Intel 80486 insir uci in pipeline i described to D'AIRA911. [HREN't1{)[provides good coverage of intezmil I 1 ri IckIN!q] rig oo the P i11.IlLan, as does [S1-1AN93 I ror i11e PowerPC.

BREV(1111 43rd', B. *The In id ic pro ce News : 80186180M5, ti,.$643101 88 80.2 86, 80386, 80486. Pentium, Nt? neon Pro and Po?thi !I Proce* . Upper Saiddlc River, NJ: Peel ricu 2000,

C.B.A1U92. Cragoit., H. *Bram n Strate..1 1 To von' inn; and Perri e Jr1 n ono: ...14orii .4%.* ton. CA: 11 E.E. C,orripuicr Sock!' y PreNs.

Dubey, P.. and NI. "Branch Strategic: fkl()&1111g te nd {"}1 7tidr^i%Rtifoll, " Trawyrecrirypis cJeg IC:one *pilaffs,* Octoks 1991.

EVEHO1 Ei..ers. M., and Ych. T. "1..inderstanding Brandin: and Designing 13ranch PredicEors for High-Performanee Microproutssors." *Prom-dings* of *Ow IEEE'.* November 2001,

HENN91 Hennessy. .1,, and Jouppi, N. "Computer Technology and Architecture; Al! Eyolving interaction." *(. onp l ter,* September J 991,

HWAN93 'Hwang. K. *Advanced* coonpraeo. ArchMin•bere. New York: Nle0raw..1-lill, 1993.

KAE1.91 teach, D., and 1..innia. 11'. ¯Branch Hist:pry Predici ion of tvloving Target Branches Due lo Subroutine f..citrrus." *Proceeding:v., Men Annual Iniernationai Syn 1* **CC.Mptif2'1 ActlideeiraV,** 1 9'.}1,

1), '•Rv.i.titeirig the Branch **PEdlally** in Pipe.tisi.ed Processors." *Computer.* July

N10511101 Moshovos. A,, and S6hi, G. "Microlrhilucturtil 111novitions: Boasting Microprocessor Per forniHnce Beyond ScanicoudtietOr Techtiology Scaling," *Procere.elings o f the. 1 N wthcr* 2.0111.

PATTIll Pail. Y. "Reoirements, Bottlenecks, and Good Fortune; Agents for Microprocessor .EYoluilion." *Proctredingy or the EL E* Novontbor 200J.

S [110195 Stanley. T. *PoworPC' System Aie.hticcrurd,* Reading., Addison-W egey. 1995,

8011190 Sohi, 0. `instruction issue: Logic for High Performance Interruptabl. Multiple Functional 4.:nit, Pipelines Computers," *IEEE Transactions on C ynputer* March 1990.

T4BA91 Tabak, D. *Atir..anced ,4 kTop oce ..rsom.* New York: McGraw-Hill, 19'91.

12.8 KEY TERMS, REVIEW QUESTIONS, AlD PROBT EMS

Key Terms

branch pre.dictiou	flag	bl i u,Cf prefetch
condition code	instruction	program stand!. word PSW)
delayed branch	instruction pipeline	

Review Questions

1 **2.1** What general roies are performed by CPU regi8te1K?

121 Vvrhat categories of data are commonly suppcned by user-visible rcgis t ers'!'

12._3 What is ale. function CFI condition codes?

12A **Vaal is** a program status word?

J1,2.5 Why is a two-stage instruction pipeline unlikely It) cut the instruction cycle time in half- cotnpared with the use of no pipeline?

12.11 List and briefly explain ...aril Fus ti which an instruction pipeline can deal with conditional branch instruciions.

11.7 How are history bits used for branch prediction?

Problems

12.1 a. if the last operation performed on a computer with an xw w LI was an addition in which the two operands were 2 and 3. what would be the ',Ant. of i Ire °Flowing flan?

* Carry
* Zero
* Over]] ow
* Sign
* Even parity
* Half-carry

b. What if the operands were —1 (twos complement) and +1.?

12.2 Consider the tinning diagram *of* Figure 12.10. Assunio that there is only a two-stage (fetch, execute). Redraw the diagram 10 show how many time units are now n ceded Coe four instructions.

123! Consider an instruction sequence of length tt that is streaming through ille instruction pipeline. Let *p* be the probabithy of encountering a conditional Or unconditional branch instruction, and]et *q be* the probability that execution of a branch instruction 1 causes a jump to a nonconsecutive ddwss. Assume that each such jump requires the pipeline to be cleared, desiroying all ongoing instruction processing, whca I emerges from the]asl stage. Revise. Equations 12A and 12.2 to take these probabilities into account.

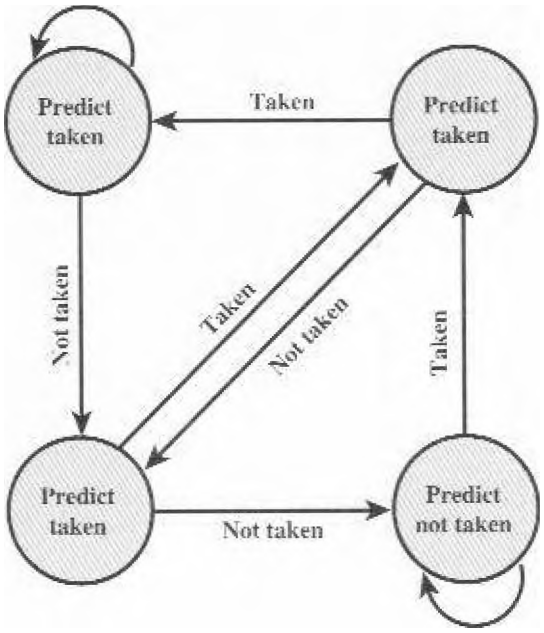
[2.4 One limitation of the multiple-stream approach to &Ming with branches in a pipeliiie is that additional branches will be encountered before the rirst branch is resolved. Suggcst two additional limitations or drawbacks.

115 Consider the state diagrams of 1 ·(pre 12.2S

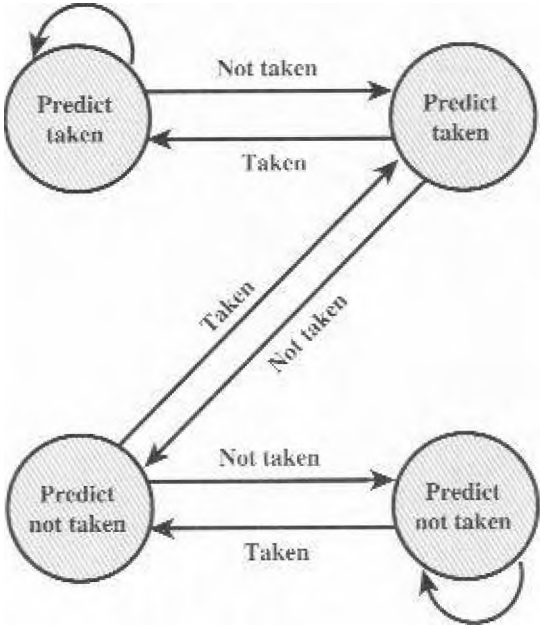
a. Describe the behavior or each.

b. Compare these with the branch prediction state diagram in Section 12.4, Discuss the re iat i L!..rits of each of the three approaches lo branch prediction,

laken



Not taken



N4 it taken

Figure 12.25 State Diaatam for. Problem I 2_5

12.6 The Motorola 68C4x0 machines include the inskructiou Decrement and Branch According w Condition, which has the follotiying farm

```
DEcu Dn,
```

where cc is one of the testable conditions, DTI is a general-purpose register. and displacement specifies the target addres!i relatNe lo the current address. Thu instruction can be defined as `kICOWS;`

```
if :cc
then begin
   Du := (Dr) - 1;
   if Dr * -1 then PC := {PC: + .5iZMIacerrien= end
ease FC :=       +
```

When the instruction is executed, the condition first tested to determine whether the iertniniltion coudition for the Loop is satisfied. if so, no operation is performed and execution curitinue.s ai 1he next instruction in sequence. ff the condition is false, the specified cla.1.a registE.r is decrement ell and checked to see if it is less than zero. II it is less than zero, the loop is terminated and execution continues at the next instructirm in sequence. Otherwise. the program branches to the specified 10ea.tion. Now consider the. following assembly. language program Iragmcni;

```
AGATH                    :Al
        u8xL     La, AGAIN
        NOP
```

Two strings addressed by Al) and AL are compared for equalitj..; the string pointers are incremented with each reference. DI initially contains the number of longwords (4 bytes) to be compared.

a. The initial contents of the reOsteN are AO = $00004 000, Al - S00005000, and 131 = $000000E1[7] (the 5 indiQtes hexadecimal notation). Memory between 540R) and $60(.110 is Loaded with words $AA A A. ff the foregoing program is run, speciflic the iminher of times the DiV,'41:: loop is executed and the contents 01 I [le Three re@-Niers when the NOP instruction is reached.

h. Repeat (a), but now assume that memory between $4000 acid 54FEE is loaded with S0000 and between $5000 and $6000 is loaded with $AA.A.

12.7 Redraw Figure 12.19c, assuming that the conditional branch is not taken.

CHAPTER 13

REDUCED INSTRUCTION SET COMPUTERS

13.1 Instruction Execution Characteristics

13.2 The Use of a Large Register File

133 Compiler.Based Register Optimization

13.4 Reduced Instruction Set Architecture

13.5 RISC Pipelining

13.6 MIPS R4110011

13.7 SPARC

13.5 RISC Versus CISC Controversy

13.9 Recommended Reading

1.3.10 Key Terms, Review Questions, and Problems

KEY POINTS

- Studies of the execution behavior of high-[eve] language proararns have provided guidance in designing :I flew typo prOccssor architecture! the reduced instruction set computer (RISC). Assignment statements predominate, suggesting that the simple movement of data should he >irtimizod. There ate also many IP and LOOP instructions, which suggesit. that the underlying7zeq 'Jena: control mechanism needs to ho permit cificieni pipelining. Studies of operand reference patterns sup.est that it should be possible *Lo* enhance performance hy keeping a moderate number of operands in risers

4 These studies have motivated thc key characieristics of RISC machiries 1 j a limited instruction set with a fixed format, (2) a]at ge number of registers or Lhe use a a compiler that optimizes register and (3) tin cruphl$is on optimizing the instruction pipe.]inc.

- The Nimple- instruction set of a RISC lends itself to efficient pipelining.becausc. there are fewer and more predictable cveralions peTFLirrutcd pet inslraction. A MSC' instruction set architecture. also lends itself to the. delayed branch tednique, in which branch instructions are rearranged with other instructions to improve pipeline efficiency.

irl.00 .the development of the stored-program computer around [9511. ill; n: have been remarkably few true innovations in the *areas of computer* orpiii. zation and architecture- The following are some of the major advances since II,: birth of the computer

- The family concept: Introduced by IHM with its System:160 in 1964, followed shorth,' thereafter by DEC. with its PDP-g. The family concept decouples the architecture of a machine from its implementation. A set of computers is offered, with different pricelperformance characteristics. that presents the same architecture to the user. The differences in price and performance are due to differenl implementations of the same architecture.
- **Microprogranuaed control unit:** Suggested hy Wilkes in 1951, and introduced by IBM on the S1360 line in 1964. Microprogramming eases the task of di:-signing and implementing the control unit and provides support for the fain ily concept.
- **Cache memory:** First introduced commercially on **IBM** S.f360 Model 85 in 1968. **The** insertion or this element into the memory hio Archv dramatically improves performance.
- **Pip-elining;** A means of introducing parallelism into the essentially sec.' ttentiA nature of a machine-instruction program. Examples are instruction pipelining and vector processing.
- Multiple processors: 'ibis category covers a number of different organizalions and ohjectiVes.

Table 13.1 Characteristics of Sonic CISCs, R1NC! .::., and Superscalar Procvssors

	Complex Instruction Si IT IS(:) Computer			Reduced Insi Niel ion Set (RISC) Computer		Suptrwalar		
Characteristic	37W L OR	VAX 11.•780	**hid** p0446	**SPARC**	MIPS R404 10	powc r PC	Chia **Sl'ARC**	**MIPS**
Year developed	**L973**	**1978**	**1989**	1987	**1991**	1993	**1996**	1 **Y9'**
NMWPM' of instru•tionN	**298**	303	235		94	225		
Instruction size Orocs)		2-57	1 _ 11	4	-1	4	**4**	
Addressing **moats**	**4**	**22**	I I	1	1	**2**	1	
Number of gcnend-purpose registers	**16**	**J6**		40- -520		**32**	**40-5'20**	**32**
Control memor, site Man	420	**48i)**	**246**					
Cache size ildrytcs)	**64**	**64**	1;	32	**125**	**16—'42**	**32**	

To this list must now be added one **of** the most interesting and, potentially, one of the most important innovations: reduced **instruction** set computer (RISC) architecture. The RiSC. architecture is a dramatic departure from the In.,' orical trend in processor architecture. An analysis of the RIS(.: architecture brings into locus many of the important issues in computer organization and architecture.

Although RISC systems **have** been defined and designed in a variety of ways by different groups, the key elements shared by most designs arc these:

- A large number of general-purpose registers, and/or the use of compiler technology to optimize register usage
- A limited and simple instruction set
- An emphasis on optimizing the instruction pipeline

Table I3,1 compares several RISC and non-RISC systems.

We begin this chapter with a brief **survey** of some results on instruction sets, and then examine each of the three topics just listed. This is followed by a description of two of the best-documented RISC designs.

13.I INSTRUCTION EXECUTION CHARACTERISTICS

One of the most visible forms of evolution associated with computers is that of programming languages. As the cost of hardware has dropped, the relative cost of software has risen. Along with that, &chronic shortage of **programmers** has driven up

software costs in absolute terms. Thus, the major cost in the life cycle of a system is software, not hardware. Adding to the cost, arid to the inconvenience, is the element of unreliability; It is common for programs. both syNiem irid application, to continue ;4.) exhibit' new hugs after years of operation.

The response from researchers and industry has been to develop ever more powerful and complex high-level prograin n; I anguages. These high-level languages (H [A.40 allow the programmer to express algorithms more concisely, take care or much of the detail, and often support naturally the use *of* structured programming or oblect-oriented design.

Alas. this solution gave rise to another problem, known as the *semantic sap,* the difference between the **op2nition8** provided in HLLs and those provided in computer architecture. Symptoms of this gap are alleged to include execution inefficiency, excessive machine program size, and compiler complexity. Designers responded with architectures intended to close this gap. Key features include Tarp instruction sets, dozens of addressing modes, and various FILL siaEernents implemented in hardware. An example of the latter is the CASE machine instruction on the VAX. Such complex instruction sets arc intended to

- Ease the task of the compiler wriler,
- Improve execution efficienc:,', because complex sequences of operations can he implemented in microcode.
- Provide support for even more complex reed sophisticated HLLs.

Meanwhile. a number of studies have been done over the years to determine the characteristics and patterns of execution of machine instructions generated from **1i** LL programs. The results of these studies inspired sonic researchers to look for a different approach: namely, to make. the architecture that supports the HLL simpler, rather than more complex.

To understand the line of reasoning of the RISC' advocates, we begin with a brief review of instruction execution characteristics. The aspects of cif imputation of interest are as follows:

- **Operations performed:** These determine the funeiions to he performed by the processor and its interaclion with memory.
- **Operands used:** The types of operands and the frequency of their use determine the memory organization for sioring them and the addressing modes for ".-essing them.
- **Execution sequencing:** This determines the control and pipeline organization.

In the remainder of this section, we summarize, the results of a number of studies of high-level-language programs. All of the results are based on dynamic 1TLe4• surernerns. That is, me urenients are collected by executing the program and counting the number of times some feature has appeared or ti particular property has held true, In contrast. static ITMISLITCMCFILs merely perform these counts OD the source Lcxt of a program. They give no useful information on performance, because they are not weighted relative. to the number of times each statement is executed.

Operations

A variety of studies have been made to analyze the behavior of HLL programs. Table 4.7, discussed in ChApter 4, includes key results from a number of studies.

There is quite good agreement in the results of this mixture of languages and applications. Assignment statements predominate. suggesting that the simple movemen of 6ta is of high importance. 'Fhere is 2ilso a preponderance of conditional statements (IF, LOOP). These statements are implemented in machine language with some sort of compare and branch instruction. This suggests ihat the sequence control mechanism of the instruction set is irroportanl.

These rt.uI1s arc instructive to the machine instruction set designer, indicating which types of statements occur most often and therefore should be supported in an "optimal ¯ fashion. However, these results do not reveal which statements use the most time in the execution of a typical program. That is, given a compiled mach ine-language program, which statements in the source laLiguage cause the execution of the most machine-langua ge instructions?

Co get at this underlying phenomenon, the Patterson programs [PAP ¯1·824 described in Appendix 4A, were compiled on the VAX, PDP-11, and Motorola 68000 to determine the average number of machine instructions and meinor!, ·' references per statement I ype. The second and third columns in Table 13-2 show 1hc relative frequency of occurrence of various HI,L instructions in a variety of progrants the data were obtained by obscrving the occurrences in running programs. rather than just the nwnber of times that statements occur in the source code. Hence these are dynamic frequency statistics. To obtain the data in columns four and five (machine instruction weighted), each value in the second and third columns is multiplied by the number cal' machine instructions produced by the. compiler. These results are then normalized so that columns four and five show the relative frequency of occurrence, weighted by the number of machine insiruclions per 111_1_ statement. Simiiar1y,ihe sixth and seventh eolumns by multiplying the frequency of occurrence of each statement type by the relative number of memory references caused by each statement. The data in columns four through seven provide surrogate measures of the actual time spent executing the various statement types. The results suggest that the procedure wiitireturn is the most time consuming operation in typical I'LL programs.

The reader should be clear on the significance of Table [3,2. This table indieatcs the relative significance of various statement types in an [ILL when that HELL is compiled for a typical contemporary instruction set architecture. Some other architecture could conceivably produce different results, However, this study produces resul ts ill iL141re representative for eon tem pora ry complex instruction set corn-

Table 13.2 Vir'cightcd Rclafivc Dynamic Fretpc.ncy ol:HLL Operatiuns [PATTS2a1

	Dynamic Occurrence		Machine-Instruction Weighted		NIerniT$_t$-liercrenet liveigbied	
	PKi .al		Pascal		Pascal	C
ASSICiN	45%	38%	r3%	13%	14"	
LC)C.)3)			42%	3.7 %	.1.1 %	
CALL	15%	l k.	31%	13%	44%	45 %
IF			%	21%		
0(.11'0						
OTRF R						

puter (CISC) architectures. 'i'hus, they can provide guidance to those looking for more efficient ways to support FILLS.

Operands

Much less work has been done on the occurrence of types of operands. despite the importance of this topic. There are several aspects the are significant.

The Patterson study already referenced [PATTS2a] also looked at the dynamic frequency of occurrence of classes of variables (Table 13.3). 'Fhe results, consistent between Pascal and C programs. show that the majority of references are to simple scalar variablcs. Further, more than 80'% of the scalars were local (to the procedure) variabies. In addition. references to arrarAtructures require it previous reference to their index or pointer, which again is usuall!L' a local scalar. 'finis, there is a preponderance of references to scalars, and these are highly localized.

The Patterson study examined the dynamic behavior of FILL programs. independent of the underl!,'ing. architecture, As discussed before, it is necessary to deal with actual architectures to examine program behavior more deeply. One study, [LUND77], examined 1]**EV- 10** instructions dynamically and found that each instruc lion on the average references 0.5 operand in memory and 1.4 registers., Similar results arc reported in IHUCK831 for C, Pascal, and FORTRAN programs on 5/370, PDP-11. and VAX- Of course. these figures depend highly on both the architecture and the compiler, hul they do iiiustrate the frequency of operand accessing.

'nose latter studies suggest the import awe of an architecture that lends itself to fast operand accessing. because this **IiperatiOn** is performed so frequently. The Patterson study suggesis that a prime candidate for optimisation is the mechanism for storing and accessing local scalar variables.

Procedure Calls

We have seen that procedure calks and returns are an important aspect of I'1LL programs, The evidence (Table 13.2) suggests that these arc i.hc most time-consuming operations in compiled HLL programs. Thus, it will be profitable to consider ways of implementing these operations efficiently. Two aspects are significant: the number of parameters and variables that a procedure deals with, and the depth of nesting.

Tanenbaum's study (TANEN' found tha1 of dynamically called prow' ,lures were passed fewer than six arguments, and that 92% of them used fewer than six local scalar variables. Similar results were. reported by the Herkeiey RISC team IKATE.S.1, as shown in Table 13.4. These results show that the number of words required per procedwe activation is not large. 'I.'he studies reported earlier indinted that a high proportion of operand references is to Local scalar valiables_'['he.!:.e studies show that those references are in fact confined to relatively few variables.

Table 113 Dynamic Pern[agtl, nt'Operanffi

	Vasa	C	Average
wgur vulva an I.	1656	23%	20%
Scalar vuriable		53%	55%
A rraylstructu re	26%	24%	.25%

Table 1.3.4 Procedure Argumenti and focal Scalar Variables

Percentage of Executed Procedure Calls With	Compiler, Interpreter, and Typesetter	Small Nona IMIleriC Programs
Hrtzurne.nts	0-7%	
romerus	0--3%	0%
wbrds of argumcnis and lucid scalars	—2(1%	
"12 words of arpuinunts a nd Local scalars	141%	3%

The same Berkeley group also looked at the pattern of procedure calls and returns in FILL programs. They found that it is rare to have a long uninterrupted sequence of procedure calls followed by the corresponding sequence of returns. Rather, they found that a program remains confined to a rather narrow window of procedure-invocation depth. This is illustrated in Figure 4.16, which was discussed in Chapter 4. These results reinforce the conclusion that operand references are highly localized,

Implications

A number of groups have looked at results such as those just reported and have concluded that the attempt to make the instruction set architecture close to H I.,1,s is not the most effective design strategy. Rather. the HLLs can best be supported b!,. optimizing performance of the most time-consuming features of lypieal HLL programs.

Generalizing from the work of a number of researchers, three elements emerge that. by and large. characterize RISC architectures. First, use. a large number of registers or use a compiler to optimize register usage. This is intended to optimize operand referencing. The studies. just discussed show that there are several references per I ILI.. instruction. and that there is a high proportion of move (assignment) statements. This. coupled with the locality and predominance of scalar references, suggests that performance can be improved by reducing memory references at the expense of more register references. Because of the locality of these references, an .expanded register set seems practical.

Second, careful attention needs 10 he paid to the design of instruction pipelines. Because of the high proportion of conditional branch and procedure call instructions, a straightforward instruction pipeline will be inefficient. This manifests itself as a high proportion Or instructions that are prefetchcd but never executed.

Finally, a simplified (reduced) instruction set is indicated. This point is not as obvious as the others. but should become clearer in the ensuing discussion.

6.2 firE CJSE OF AiIiiiGF:ILEGISYtitIPILE

Thg. results summarized in Section 13.1 point out the desirability of quick access to operands. We have seen that there is a large proportion of assignment statements in I Ili programs, and many of these are of the simple form A B. Also, there is a significant number of operand accesses per 1-ILL statement, It' we couple these

results with the fact that most accesses are to local scalars. heavy reliance on register storage is suggested,

The reason that register storage is indicated is that it is the fastest available storage device, faster than both main memory and cache. The register file is ph!, ' s- ically small, on the same chip as the AIA.1 and control unit, and employs much shorter addresses than addresses for cache and memory. Thus. a strategy is needed that will allow the most frequently accessed operands to be kept in registers and to minimize register-memory operations.

Two basic approaches are possible. one based on software and the other on hardware. The software approach is to rely on the compiler to maximize register usage. The compiler will attempt to allocate registers to those variables that will be used the most in a given time period. This approach requires the use of sophisticated program-analysis algorithms. The hardware approach is simply to use more registers so that more variables can he held in registers for longer periods of time.

In this section, we will discuss the hardware approach. This approach has been pioneered by the Berkeley RISC group [PATT824 was used in the first commercial RISC product, the Pyramid 1RAGA831: and is currently used in the popular SPARC architecture.

Register Windows

On the face of it, the use of a large set of registers should decrease the need to access memory. The design task is to organize the registers in such a fashion that this goal is realized.

Because most operand references are lo local scalars, the obvious approach is to store these in registers, with perhaps a few registers reserved for global variables. The problem is that the definition of *local* changes with each procedure call and return. operations that occur frequently. On every call. local variables must be saved from the registers into memory, so that the registers can be reused by the called program. Furthermore., parameters must be passed. On return, the variables of the parent program **must** he restored (loaded back into registers) and results must be passed back lo the parent program.

The solution is based on two other results reported in Section 13.1- First. a typical procedure employs only a few passed parameters and local variables (Table 13.4), Second, the depth of procedure activation fluctuates within a relatively narrow range (Figure 4.1.6). To exploit these properties. multiple small sets of registers are used, each assigned to a different procedure, A procedure call automatically switches the processor to use a different fixed-size window of registers, rather than saving registers in memory. Windows for adjacent procedures are overlapped to allow parameter passing.

The concept is illustrated in Figure 13.1. At any lime, only one window of reg. inters is visible and is addressable as if it were the only set of registers (e.g.. addresses 0 through N — I). The window is divided into three fixed-size areas. Parameter registers hold parameters passed down from the procedure that called the current procedure and hold results to be passed back up- Local registers are used for local variables, as assigned by the compiler. Temporary registers are used to exchange parameters and results with the next lower level (procedure called by current procedure). The temporary registers at one level are physically the same as the para.

meter registers at the next lower level- 'Fhis overlap permits parameters to be passed wilhout the actual movemeru of data.

To handle any possible pattern of calls arid returns, the number of rigis.ier windows would have to be unbounded. Instead, the regisler windows can be used to hold the Cew most recent procedure activations. Older acrivations must be saved in memory and later restored when the nesting depth decreases, Thus. the tel ual organization of the register file is as a circular buffer of overlapping windows. Two notable examples of this approach #arc Sun's SPA RC' architecture, described in Section 13.7, and the IA-64 architecture used in Inters I tanium processor, described in Chapter 15.

This organization is shown in Figure 13.2, which depicts a circular buffer of six windows, The buffer is filled lo a depth Or 4 (A eAled 13; B called C. C called ID) with procedure D active. The current-window pointer (OAT) points 10 the window of the currently active procedure. Register references by a machine instruction are ofCsel pt inier to delermine the itctwil physical register. The saved-window pointer identifies the window most recently saved in memory. If procedure D now calls procedure E. arguments for E are place.d in D's temporary registers (the overlap between w3 and w4) and the k advanced by orie window.

procedure then makes call to procedure F, the ca]] cannot be made with the current status of the buffer. This is because F's window overlaps window. 11 F begins to load its temporary registers. preparatory lo a call. ii will overwrite the parameler registers of A (,Ain}.'I1 ius., when CW1' is incremented (modulo 6) so that it becomes equal to SWP. an interrupt occurs. and As window is saved. Only the first two portions (A.in and Aloe) need be saved. Then, the. SWP ix increminted and the call to I proceeds. A 'Thtil:tr inl.errupl can occur on returns. **For** example, subsequent to the activation of I when B returns to A. CV IP is decremented and becomes equal to SWP. This causes an interrupt that results in i he reslOr'enion ref A's window.

From lhe preceding. it can be c.t!CII 111.;11 **ni** N-window register file can hold only N — I procedure aetivi ions. The value of N need not be large. As w'ElS mentioned in Appendix 4A. one study [TAM183] found that. with g windows, a save or resLore is needed on only I% Or the calls or TO urns. I lie Berkeley RISC computers use 8 windows of 16 registers each. Pyramid computer employs 16 windows of 32 registers each.

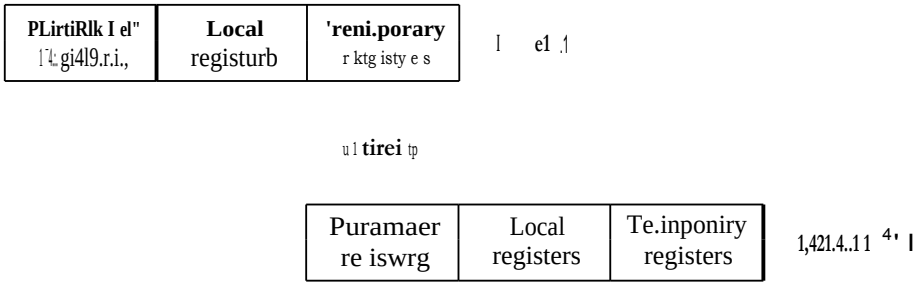


Figure 13.1 Overlapping Register

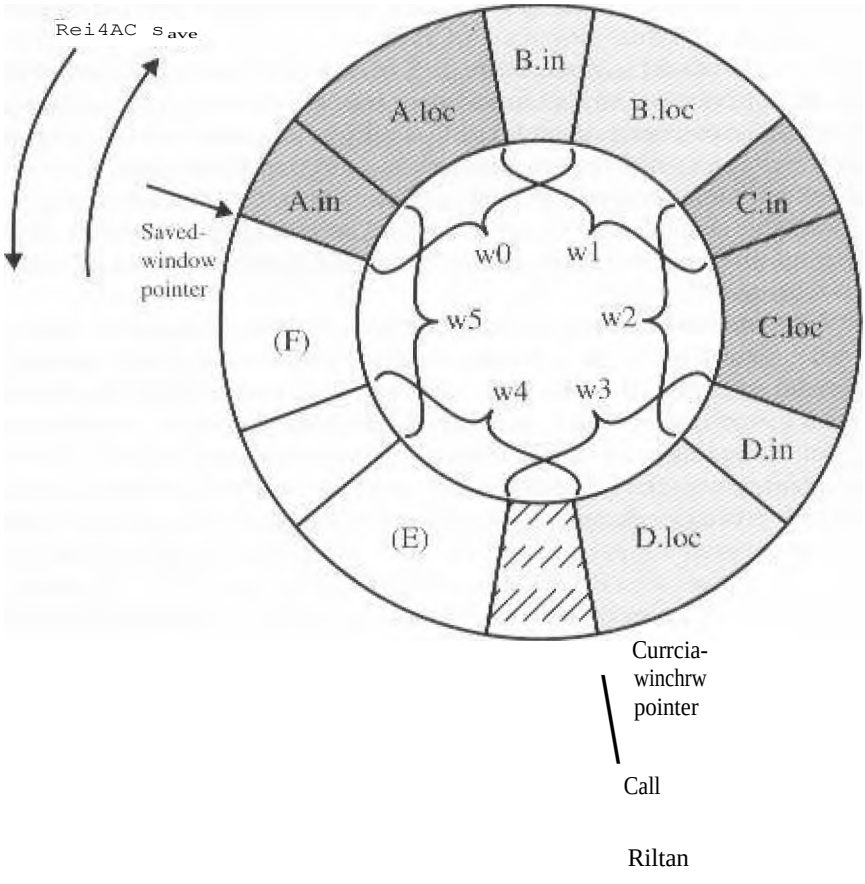


Figure 13.2 Circular-Butter Organization of Overlapped Windows

Global Variables

The window scheme just described provides an efficient organization for storing local scalar variables in registers. However, this scheme does not address the need to store global variables, those accessed by more than one procedure. **Two** options suggest themselves. **First,** variables declared as global in an FILL can be assigned memory locations by the compiler, and all machine instructions that reference these variables will use memory-reference operands. 'this is straightforward, from both the hardware and software (compiler) points of view. However, for frequently accessed global variables, this scheme is inefficient.

An alternative is to incorporate a set of global registers in the processor. These registers would he fixed in number and available to all procedures. A unified numbering scheme can he used to simplify the instruction format. For example, references to registers 0 through 7 could refer to unique global registers, and references to registers 8 through 31 could be offset to refer to physical registers in the current

window. There is an increased hardware burden to aecommod.ate the split in register addressing. In additiOn, the compiler must decide which global variables should be assigned to registers.

Large Register File versus Cache

'file register file, organized into windows, acts as a small, faSL buffer rot holding a subset or all variables that are likely to be used the most heavily. From this point of view, the register file ads much Eike a cache memory. although a much faster memory. The question therefore arkes ;is to whel her i1 would be simpler and better to use a cache and a small traditional renister file.

Table 13.5 compares characteristics of the two approaches. The window-based register file holds all the local scalar variables (except in the rare case of window overflow) of the most recent N — 1 proeedure activations. 'I'he cache holds a wleeflan of recenily used scalar variables. The register file should save time, because all local scalar variables are retained. On I he other hand, the cache may make. more efficient use of space, because it is reacting to Lhc ;...ittnuion dynamically, Furthermore, caches generally treat all memory references alike, including instructions and other types of **&Oa,** Thus, savings in these other areas are possible with a cache and not a reaister file.

A register tile may make inefficient use of space, because nol procedures will need the full window space allotted to them. On the other hand, the cache suffers from another sort of inefficiency: Data are read into the cache in blocks. Whereas the register file contains only those v4iri All in use, the cache reads in 41 Nock of Ili ta, some or much of which will not be used

The cache is capable of handling global as well as local variables. There are ustialEy many global scalarzt. but only a few of 1hern arc heavily used [KATE.3]. A cache will dynamically discover these variables and hold them. If the window-hascd register file is supplemented with global registers, it too can *hold* sonic glohril scalars. Elow.evcr, it is difficult for a compiler to determine which globa]s will be heavily used.

1he register file, the movement of data between registers and memory is determined by, the procedure nesting depth. Because this depth usually fluctuates within a narrow range, thc.use of memory is relatively infrequent, Most cache Meal-

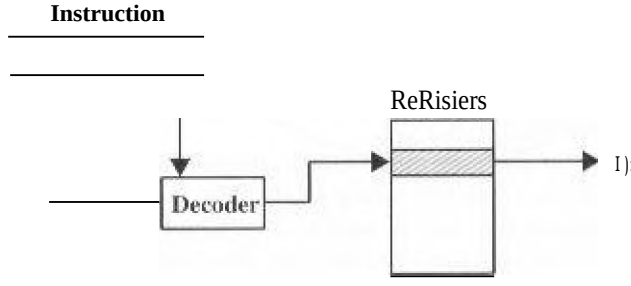
Table 13-5 Cliaroctoristic9 **of** Large-Register-File **and Cadic Organizations**

Large Register File		**Cache**	
]deal		**kLickintly**	
individual varial'les		**Blocks**	
Compiltt-kNFix:.n4:d g3cih;ri1 wariahks		ly	lobo! vniablc.5
Saw u, Tics' **ing**	INi3c-cl on proce.tlue		bki5s2.13 oil .;:aClic.
RI4r,it6r addressing		411:71.15ry : RI	

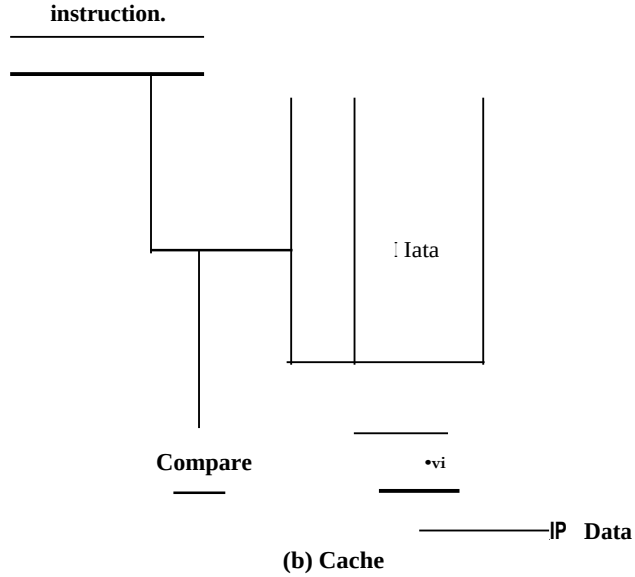
ories.are set associative with a small set silf.c. 'lhus, there is the danger That data or instructions will overwrite frequi.'2ntiy used variables.

l- sect on the discussion so far, the choice between a large window-based register file and a cache is not clear-cut. There is one characteristic_ however, in which the register approach is clearly :iuperior and which su ggests that a ea(,:he-based system will be noliceL ¯Fly sEcrwer. This distinction shows up in the amount of addressing overhead experienced by the two approaches.

Figure 13.3 illustrates the difference. To reference a local scalar in a window-based register rile. a 'virtual" register number and a window number are used. 'Ilicse can pass through a relatively simple decoder to select one of the physical registers. To reference. a memory location in cache, a full-width memory address must be generated. The conip]exi ty of c his operation depend s on the addressing *mode.* in **set** associative cliche, a portion of the. address is used to read a number of words



(:411 **Winnows-hosed register file**



(b) Cache

Figure 13.3 RcEurcncing a Sci'ithr

and tags equal to the set size. Another portion of the address is compared with the tags. and one of the words that were read is selected. It should he clear that even if the cache is as fast as the register file, the access time will be considerably longer. Thus, from the point of view of performanoc, the window-based register file is superior for local scalars. Further performance improvement could be achieved by the addition of a cache for instructions only,

13.3 COMPILER-BASED REGISTER OPTIMIZATION

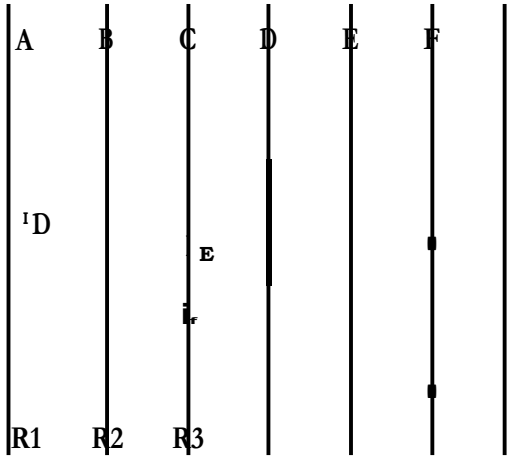
1..ei us assume now that only a small number (e.g., 16-32) of registers is available on the target RISC machine. In this case, opt imit.ed register usage is the responsibility of the compiler, A program written in a high-level language has, of course, no explicit references to registers, Rather, program quantities are referred to symbolically. The objective of the compiler is to keep the operands for as many computations as possible in registers rather than main memory, and to minimize load-and-store operations.

In general_ the approach taken is as follows. Each program quantity that is a candidate for residing in a register is assigned to a symbolic or virtual register, 'Fhe compiler then maps the unlimited number of symbolic registers into a fixed number of real registers. Symbolic registers whose usage does not overlap can share the same real register. if, in a particular portion of the program. there are more quantities to deal with than real registers, then some of the quantities are assigned to Memory locations_ Load-and-store instructions are used to posil ion quattities in registers temporarily for computational operations.

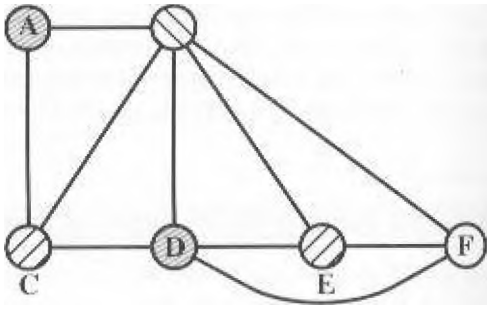
The essence of the optimization task is to decide which quantities are to he assigned to registers at any given point in the program, The technique most commonly used in RISC compilers is known as graph coloring, which is a technique borrowed from the discipline of topology [CHAI82. CHOW86. COU186. CHOW901,

The graph coloring problem is this_ Given a graph consisting of nodes and edges. assign colors to nodes such that adjacent nodes have different colors, and do this in such a way as to minimize the number of different colors. this problem is adapted lo the compiler problem in the: following way. First, the program is analyzed to build a register interference graph. The nodes of the graph are the symbolic registers. If two symbolic. registers are "live" during the same program fragment, then they arc joined by an edge to depict interference. An attempt is then made to color the graph with n colors, where *n is* the number of registers. Nodes that share the same color can he assign ed to the same register, I r this process does not fully succeed, then those nodes that Cannot be colored must be placed in memory, and loads and stores must he used to make space for the affected quantities when they are needed.

Figure L3.4 is a simple example of the process. Assume a program with six symbolic registers to he compiled into three actual registers. Figure I3.4a shows the ti me sequence of active use of each symbolic register. and part h shows the register interference graph (shading and cross-hatching are used instead of colors). A possible coloring with three colors is indicated. One symbolic register. F, is left uncolored and must be dealt with using loads and stores.



tat 'lime sew, *cove* of active use of registers



lb) KegiNter interference graph

Figure 13.4 I a 1)h Coloring Approach

In general. there is a trade-off between the use of a large set of registers and compiler-based register optimization. For example, I BRAD91 al reports on a study that modeled a RISC architecture with features similar to the Motorola 88000 and the ZIPS 82000. The researchers varied the number of registers from 16 to 128. and they considered both the use of all general-purpose registers and registers split between integer and floating-point use. Their study showed that with even simple register optimization. there is little benefit to the use of more than 64 registers. With reasonably sophisticated register optimization techniques. !here is only marginal performance improvement with more than 32 registers. Finally, they noted that with a small number of registers (e.g.. VI). a machine with a shared register organization executes faster than one with a split organization. Similar conclusions can be drawn from [HUGIA911. which reports on a study that is primarily concerned with optimizing the use of a small number of registers. rather than comparing the use Of large register sets with optimization efforts.

13.4 REDUCED INSTRUCTION SET ARCHITECTURE

In this section. we look at some of the general characteristics of and the motivation for a reduced instruction set architecture. Specific examples will be seen later in this chapter_ We begin with a discussion of motivations for contemporary complex **instruction** set architectures.

Why CISC

Vie have noted the trend to richer instruction sets, which include a larger number of instructions and more complex instructions. Two principal reasons have motivated this trend: a **desire** to simplify compilers and a desire to improve. perfor-

mance. Underlying both of these reasons was the shift to high-level languages (FILL) on the part of programmers architects attempted to design machines that provided better support fOr

It is not the intent of this chapter to say that the CISC designers took the wrong direction_ Indeed, because technology continues to evolve and because architectures exist along a spectrum rather than in two neat categories, a black-and-white assessment is unlikely ever to emerge. Thus:, the comments that follow are simply meant to point out some of the potential pitfalls in the CISC approach and to provide some understanding of the motivation of the RISC lidherents.

The first•of the reasons died, compiler simplification, weill obvious_ The task of the compiler writer is to generate a sequence of machine instructions for each HLL statement. If there are machine instructions that resemble HLL statements, this task is simplified. This reasoning has been disputed by the RISC' researchers IHNNS2.]. [RADIS31, [PA'1182b]). They have found that complex machine instructions are often hard to exploit because the compiler must find those cases that exactly fit the construct. 'Pie task of optimizing the generated code to minimize code size, reduce instruction execution count. and enhance pipelining is much more difficult with a complex instruction set. As evidence of this. studies cited earlier in this chapter indicate that most of the instructions in a compiled program are the relatively simple ones.

The other major reason cited is the. expectation that a CISC will yield smaller. faster programs. Let us examine both aspects of this assertion: that programs will be smaller and that they will execute faster.

There are two advantages to smaller programs_ First, beeause e program takes up less memory, there is a savings in that resource. With memory today being so inexpensive, this potential advantage is no longer compelling. More important, smaller programs should improve performance, and this will happen in two ways. Hrst, fewer instructions means fewer instruction bytes to be fetched. Second, in a paging.environment, smaller programs occupy fewer pages, reducing page faults.

The problem with this line of reasoning is that it is far from certain that a CISC program will be smaller than a corresponding RISC program. In many cases, the CISC program, expressed in symbolic machine language, may be *shorter (i.e.*. fewer instructions), but the number of bits of memory occupied may not be noticeably *smaller.* Table 3.fi shows results from three studies that compared the size of compiled C' programs on a variety of machines. including RISC which has a reduced

Table 13.6 Code Sise Resistive iii RISC I

	I PA'ITS2a1 11 C Programs	[KATE831 12 C Programs	111•ATS41 5 17 Programs
RISC I	I.0	1.0	**1.0**
VAX- 1 1178D	0.8	0.67	
M6tWG	1).9		0.9
78012	1.2		1.12
PDP-11170	.0.9	0,71	

instruction set architecture. Note thnt there is little or no savings using a GISC over a RISC. **it r iist.1 ;ni c.ro4Eing** to note that the VAX, which has 4l much more complex instruction set than the. PDP-11, achieves very [it * Savingz, over the biter. 'Mese results were confirmed by IBM researchers I RAD W I. who found that the I BM 801 (a RISC) produced code that was 0.9 times the size of code on an IBM St370. The cludy used a set of PL,II programs.

There are several reasons for the7se railicr surprising results. We have already noted that compilers on CISCs tend to favoi simpler instructions, so that the conciseness of inc. **complex** instructions seldom comes into pia **Y.** Also, because there are more instructions on a C.ISC, longer opcodes are required, producing longer instruelions. Finally, RISCs tend to emphasize register rather than memory references, and the former require fewer bits- An exampie of this last effect is discussed prez;.cittl±f..

So the expectation that a CISC. will produce smaller pa)grorns, with the alien' dant advantages, may not be realized, The second mc,Itivating factor for increasingly complex instruclion sets was that instruction execution would be faster, **it** seems to make. scnsc alai a complex FILL operation will execute more quickly as a Single machine instruction rather than as a series of more prirniiive instructions. However. because of the bias toward the use of 1[11.,fiL' simpier instructions. this may no.1 **be so. The entire control unit mull** he male more complex, anclior the microprogram control store must be made larger. to accommodate a richer instruction set. Either factor increases the execution time of the simple insi ructions.

in fact, sonic researchers have found that the speedup in the execution of oomplcx functions is due not so much to the power of the complex machine instructions as to their ✓esidence in high-speed control store [RADI.8.3], In effect, the control store acts as an instruction cache. Thus. the hardware archiLcet is in the position of trying to determine which subroutines or functionK will he used most frequently and assigning those to the control store by implementing them in microcode. The results have been less than unCOLINiging. On S/390 systems, instructions such **as** II·4l 74rib: and Extended-Precision• Floating-Point-Divide reside in high-speed storage, while the sequence involved in sot• brig up procedure calls or initiating an interrupt handler are in slower main memory,

Thu:5, it is far from dear that a trend to increasingly complex instruction sets is appropriate. This has led a number of groups io pursue the opposite path.

Characteristics of Reduced Instruction Set Architectures

Although a variety of diffcreni appro2iches to reduced instruction sal : ii thitecture have been Lakcn. certain characteristics are common to all of them:

- One instruction per cycle
- Register-to-register operations
- Simple aLldressing modes
- Simple instruction formats

Here, we provide a brief discussion of these characteristics, Specific examples are explored later in this chapter.

The first characteristic listed is that There is **one machine instruction per machine cycle.** A *machine cycle* is defined to he the time it takes to fetch two

operands from registers, perform an ALL' operation. and store the result in a register. Thus, RISC machine instructions should be no more complicated than, and execute about as fast as. microinstructions on CISC machines (discussed in Part Four). With simple, one-cycle instructions, there is little Or no need for microcode; the machine instructions can be hardwired. Such instructions should execute faster than comparable machine instructions on other machines. because it is not necessary to access a microprogram control store during instruction execution.

A second characteristic is that most operations should he register to register, with only simple LOAD and SIORE operations accessing memory. This design feature simplifies the instruction set and therefore the control unit. For example. a RISC instruction set may include only one or two ADD instructions (e.g., integer add, add with carry): the VAX has 25 different ADD instructions. Another benefit is that such an architecture encourages the optimization of register use, so that frequently accessed operands remain in high-speed storage.

This emphasis on register-to-register operations is notable for RISC designs. Contemporary C1SC machines provide such instructions but also include memory-10-memory and mixed registerimemory operations. Attempts to compare these approaches were made in the 1970s, before the appearance of R SCs. Figure 13,5a illustrates the approach taken. Hypothetical architectures were evaluated on program size. and the number of hits of memory traffic. Results such as this one led one researcher to suggest that future architectures should contain no registers at all [MYER781. One wonders what he would have thought, at the time, of the RISC machine once produced by Pyramid, which contained no less than 528 registers!

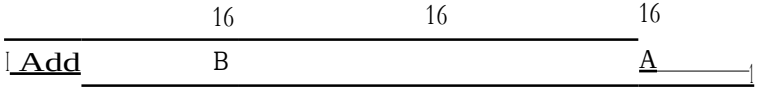
What was missing from those studies was a recognition of the fregtxnt access to a small number of local scalars and that, with a large bank of registers or an optimizing compiler. most operands could he kept in registers for long periods of time. Thus. Figure 13.5b may be a fairer comparison.

A third characteristic is the use of simple addressing modes. Almost all RISC instructions use simple register addressing. Several additional modes, such as displacement and PC-relative. may be included. Other, more complex modes can be synthesized in software from the simple ones. Again, this design feature simplifies the instruction set and the control unit_

A final common characteristic is the use of simple instruction formats. Generally. only one or a few formats arc used. Instruction length is fixed and aligned on word boundaries. Field locations, especially the opcode. are fixed. This design feature has a number of benefits. With fixed fields, opcode decoding and register operand accessing can occur simultaneously. Simplified formats simplify the control unit, Instruction fetching is optimized because word-length units are fetched. Alignment on a word boundary also means that a single instruction does not cross page boundaries.

Taken together, these characteristics can be assessed to determine the potential benefits of the RISC approach. These benefits fall into two main categories= those related to performance, and those related to VLSI implementation.

With respect to performance, a certain amount of ¯circumstantial evidence" can be presented. First. more effective optimizing compilers can be developed. With more-primitive instructions, there are more opportunities for moving functions out of loops, reorganizing code for efficiency. maximizing register utilization, and so forth. It is even possible to compute parts of complex instructions at compile time. I car example, I he



Memory-to-memory
1 = 56, D = 96, NI :--- 152

Load
Add rA r7.1
Siore A

Regisier-to-memory

(▮) A t—B+C

8	16	16	16
Add	B	C	A
Add	A	C	B
Sub	B	D	D

Memory-to-memory
1= **=** 288 M = 456

Register-to-memory
1 = 104 D = 96, M = 20(1

(13) A **B + C; A + C; D D _ B**

1 = Size of executed instructions
D Size of executed data
NI =I+ D = Total memory traffic

Figure .13.5 Two Comparisons of Register-to Regisier and Memory-to-Memory Approaches

5i390 Mi 'YE Characters (PvIVC.) instruction moves a string of characters from one location to another. Each lime it is executed, the move will depend on the length of the string, whether and in which direction the locations overlap, and what the alignment characteristics are. In most cases, these wilt al] he kilowli at compile time.Thus,ihe compiler could pnicitice ain optimized sequence of primitive instructions for this function.

A second point, ;.ilready nutted, i.s that most instructions generated by a compiler are relatively simple anyway. It would seem reasonable that control built sped liea rly for those instructions and usin@, little or no microcode could execute them faster than a comparable CISC.

A third point rclates to the use of instruction pipelining. RISC researchers feel that the instruction pipelining technique can he applied much more effeciively with a reduced instruction set. We examine this point in some detail presently,

A final, and somewhat less significant. point is that RISC processors are more responsive to interrupts because inierrupts are checked between rather elementary operations. Architectures with complex instructions .either restrict interrupts to instruction boundaries or must define specific interruptible points and implement mechanisms for restarting an instruction,

The case for improved performance for a reduced instruction set architecture is strong, bui one could perhaps still make an argument for CISC. A number of studies have been done but not on machines of comparable technology and power. Further, most studies have not attempted to separate the effects of a reduced instruction set and the effects of a large register file. The "circumstantial eviderirc," however, is suggestive-

111e second area of potential benefit, which is more dear-cat, relates to VLSI implementation. When VLSI i7,1te:le(1,1he design and implementation of thg processor are fundamentally changed. Traditional processor, such as the IBM S.1390 and the VAX, cAmsis1 of one or more printed circuit boards containing standardized 551 and MST packages, With Ihe advent of LSI and VLSI, it is possible to put an entire processor on a single chip. For a single-chip processor, there arc two motivations for following a RISC strategy. First, there is the issue of performance. On-chip delays are 0r much shorter duration than interchip delays. Thus, it makes sense to devote scarce chip real estate to those activities that occur frequently. We have seen that simple instructions and access to [twat •L7114]1 ˙5 411. ˙C, in fact, the most frequent activities, The Berkeley RISC chips were designed with this consideration in mind, Whereas a typical single-chip microprocessor dedicates about half of its area to the microcode control store. the RISC / chip devotes only about 6% of its area to the control unit [SHER84].

A second VLSI-related issue is design-and-implementation time. A VLSI processor is di fficuh to develop. Instead of relying on available SSUMSI parts, the designer must perform] circuit design, [avow, and modeling at the device level. With a reduced instruction set architecture, this process is far easier, as evidenced by Table 13,7 781] IL in addition, the performance of the RISC chip is equivalent to comparable MC. microprocessors. [hen the advantages of the RISC approach become. evident.

CISC versus RISC Characteristics

After the initial enthusiasm for RISC machines, there has been a growing *realization* that (1) RISC designs may benefit from the inclusion of some CISC' features

"table 13,7 1.) s.ign mid Layout Effort rcir Sonic. Microprocessors

CPU	Transistors (thousands)	1)4Aga (person-mouths)	Layout (pet'snu-monalm)
RISC1	44	15	12
RISC. II	41	Its	.12
M4581.100	(5b:	NO	70
7.0...10	1R	60	10
In iet iAPA-432	110	170	90

and that (2) C1 SC designs may benufit from the inclusion of some RISC features. The result is that the more recent RISC designs, notably the PowerPC, are nu longer "pure" RISC and the more recent CISC designs, notably the Pentium II **and later Pentiktm** models. do incorporate SOMe RISC' characteristics,

An interesting comparison in 'MASI 195J provides soma insight into this issue. Table 13.8 lists a number of processors and compares them across a number 0r chat. acteristics. For purposes of this comparison. the Lotlowing are **wrisitlenal** t!,, pieal of a classic RISC;

1. A single instruction size,
2. That size is typically 4 bytes.
3. A small number of data addressing modes, typically less than five, This parameter is difficult to pin down. In the table, register and Ulu& modes are not counted and different formats with different c **afrsei** sizes arc counted separately.
4. No indirect addressing that requires you to make one memory access In ga the address of another operand in memory,
5, No operations that combine load/store with arithmetic (e.g., add from memory, add to memory),
6. No more than one memory-addressed operand per instruction-
7. Does not support arbitrary alignment of data. for loadIstore operations.
K. Maximum number of uses of the memory m;inagemeni unit (MMI..) for a data **address** in an instruction,
9. Number of bits for inte ger register specifier equal Lo five or more. 'Phis means that al (east 32 integer registers, can he explicitly referenced at a time.
10. Number of bits for floating-point reaister specifier equal to four or more, This means that at least 16 floating-point registers can be Yxpl ly referenced at a time.

Items 1 through 3 are an indication of instruction decode complexity. Items 4 through S suggest the ease or difficulty of pipelining, especifllly **in** the presence of virtual memory recitnrcmen Ls. [terns 9 and 10 are related to the ability to take good advantage of compilers.

In the table, the first eight processors are clearly RISC' architectures, the next ruc are clearly CISC, and the last Iwo are processors often thought of as RISC that in fact hz•r..c many ('ISC characteristics,

Table 13.8 Character islics of Some Processors

Processor	Number of in.structinn sizes	Max instruction size in bytes	Number of addressing mz.Ries	Indirect addressing	Load/store combined with arithmetic	Max number of memory operands	Unaligned addressing allowed	Max nipmber of MMU uses	Number of bits for integer register specifier	Number of bits for FP register specifier
A N.11/29U0il		4		tiCt	110					3'
MIPS R201		4							S	4
SPARC				no	110		no			4
MC8S0011.		4		0	no		0(1	1	5	
TIP PA			e_r	ors	110		no			4
IBM RT.TC		4		110	no	1	0		-4	
IBM RSI61 10		4	4	no	00		yeS			
Inte41860		4		110	no	1	n	1	5	
IfIM .1041)				n(1'	yes		VCS	4		
Intel 80486	17.	12	1.5	110	yES.	2	yes			
NSC. 32016	71	21	23	yo.s	yes		_Yc5	4		3
`1068040		77	-14	yes	yts		ves		4	3
VAX	56	56	22	yEs.	es		ycs	24	4	
ClIpper	4"			no	nu			2	4	3'
Intel 80960		8"		110	TN)		yesz'		5	

Rim.. i sLi lc% FIJI conform to this di

11132 d0C-, ri i! confouri to thx_C Chal acr.s:TI,I lc.

13.5 RISC PIPELINING

Pipelining with Regular Instructions

As we discussed in Section 12,4. instruction pipelining is often used to enhance performance. Lei us reconsider this in the context of a RISC archiLecLutc, Most instructi ons are register to register, and an instruction cycle has the following two stages:

- I: Instruction fetch.
- E= ENCQUW. Performs an ALU operation with register input and output.

For load and store operations. three stages are required!

- I: Instruction fetch.
- E.! Execute. Calculates memory address-.
- D: Memory. Register-Io-memory or inemory-toiegistu operation.

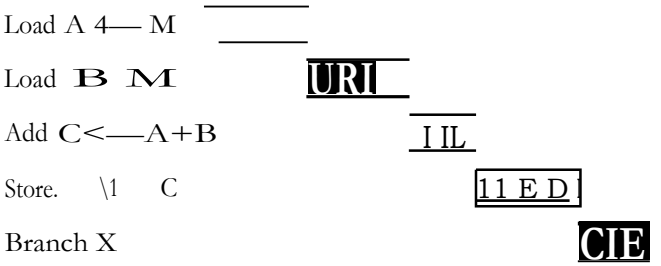
Figure 13.6a depicts the timing of a sequence of instructions using no pipelining. Clearly, [his is a wasteful process. Even very simple pipelining can substantially improve performance. Figure i3.01) ;,lhow!, a two-stage pipelining scheme, in which the I and E stages of two differcni. instructions are performed simultaneously. This scheme can yield up tie twice the execution rate of a serial scheme. Two problems prevent the maximum speedup from being achieved. First, we assume that a single-port memory is used and that only irnc memory access is possible per stage. This requires the insertion of a wait state in some instructions. Second, a branch instructi on interrupts the sequential flow of execution. To accommodate this with mini• mum circuitry, a NOOP instruction can b4 inserted into the instruction stream by, the compiler or assembler.

Pipelining can be improved further by permitting Iwo memory accesses per stage. This vicids the sequence, shown in Figure 13.6c. Now, up to three instructions can be overlapped. and the iniprovemcnl is as much as a factor of 3. Again, branuh instructions cause the speedup to fall short of the maximum possible. Also, note that data dependencies have an effect. If an instruction needs an operand that i8 altered by the preceding instruction, a delay is required. Again. this can be accomplished by a NOOV.

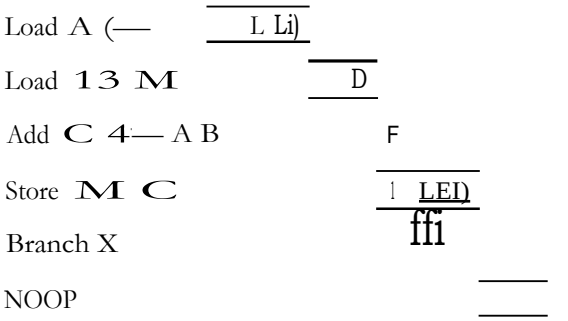
The pipelining discussed so far works best if the three stages are of approximatel!„y equal duration. Because the E stage Lain:illy involves an ALL operation, il may be longer. In this case, we can divide into two substages;

- E,1 Register file read
- E,: ALU operation and register write

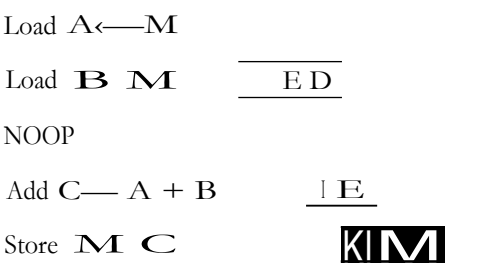
Because of the simpkity and regularity of a RISC instruction set, the design or the phasing into three or four stages is easily accomplished. Figure 13,6d shows the result with a four-stage pipeline-Up 111 four instructions at a time can be under way, and the maximum poLeTlial speedup is a factor of 4. Note again the use of NO0 Ps to account for *Li ma* and branch delays.



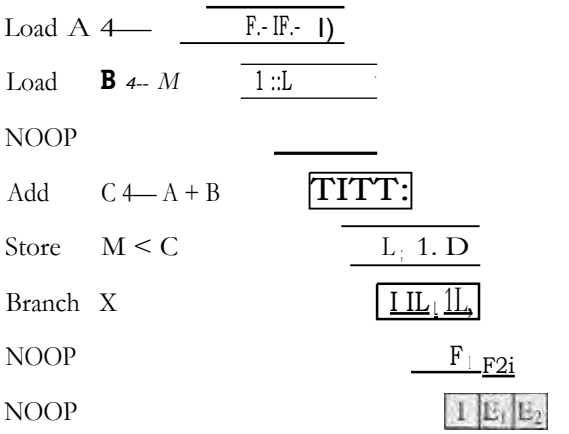
(a) Sequential execution



(h} Two-way pipelined timing



(c) Three-way pipelined timing



(d) Four-way pipdincd timing

Figure 13.6 The Fifects of Pipelining

Optimization of Pipelining

Because 0r the simple and regular nature of MSC instructions, pipelining schemes can be efficiently employed. There arc few variations in instruction execution duration, and the pipeline can be tailored to reflect this. However, we have seen that data and branch dependencies reduce the overall execution rate.

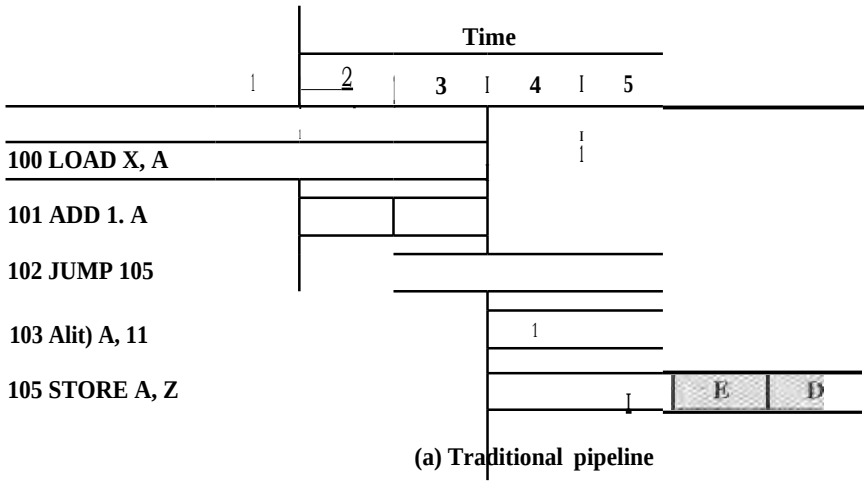
To compensate for these dependencies. code reorganization techniques have been developed. First. let in (.74insider branching instructions, *Delayed broach,* a way of increasing the efficiency of the pipeline, makes use of al-Franch that does noel Lake effect until after execution of the following instruction (hence the term *deiciye411.* 'I'he instruction location immediately following the. branch is referred to as the *defay Nlo.r.* This strange procedure is illustrated in Table 13.9, In the column labeled "normal branch, ¯ we see a normal symbolic instruction machine-language program. After 102 k executed , the next instruction to be executed is 105. To regularize the pipeline, a NOOP is inserted alter this branch. However. increased performance is achieved if the instructions at 101 and 102 are interchanged.

Figure 13.7 shows the result. Figure, l3- 7a shows the traditional approach to pipelining, of the type discussed in Chapter 12 (e.g., see Figures L2.11 1.riti 1 **112).** The .11:MP instruction is fetched al time At time 4. the JUMP instruction is executed at the same time thai instruction 103 (ADD insiruci ion) is fetched. Because a 31.11vIP occurs, which updates the program counier, the pipeline mull be cleared of instruction 1113: at time 5, instruction 1115, which is the target of the JUMP. is loaded. Figure 13.7b shows the same pipeline handled by a typical RISC organization. The timing is the same. However, because 0r the insertion of the NOOP insiruelion, we do not need special circuitry 1of clear the pipeline; he NOOP simply executes with no effect. Figure 13.7c shows the use of the delayed branch. The JIJMP instruction is fetched at lime 2, before the ADD instruction, which is fetched at time 3. Note, however, that the ADD instruction is fetched before the execution of the JUMP instruction has a chance to alter the program counter. Th ere fore., during time 4, the ADD instruction is executed at the same time lhaL instruction 105 is fetched. Thus, the original semantics of the program arc retained but one less clod cycle is **roc ired** For execution.

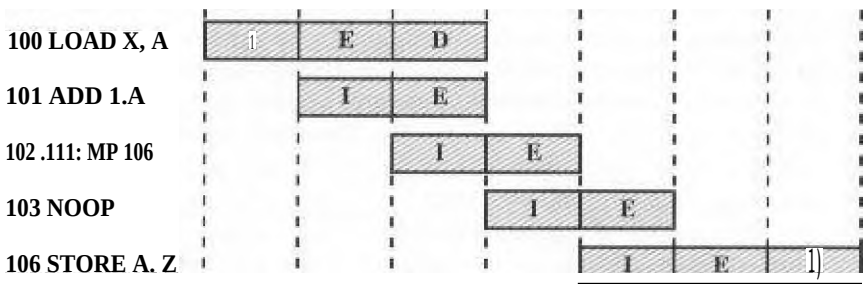
This interchange of instructions will work successfully for unconditional branches, calls. and returns. P'or conditional branches, this procedure cannot be

iropie **11.9** **Normal and** Delayud Branch

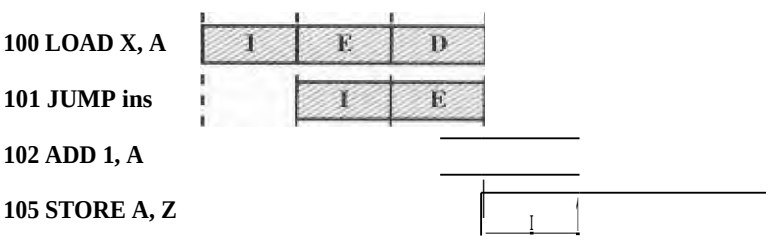
Address	Normal Branch		Delayed Branch		Optimized Delayed Branch	
Rio	**LOAD**	**X.A**	l**OAD**	X,A	LOAD	X,A
101	ADD	1: A	ADD	**L,A**	**.11!MP**	105
102	**JUMP**	105	JUMP	1U	ADD	LA
103	ADD	A.B	ls:OOP		ADD	.A,6
104	SUB	C..14	ADD	AJEL	SUFI	C.13
11)5	STORE	A.Z	SUB	C.B.	STORE	A,Z
106			S1 01'4.1¯	A.Z		



(a) Traditional pipeline



(11)1 RISC pipeline with inserted NOOP



(Eh Revemerl

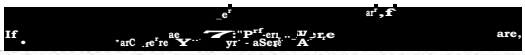
Figure 13.7 use of Lhc Delayed

blindly applied. If the condition that is tested kr the branch can be altered by the immediately **preceding** instruction. then the compiler must refrain from doing the interchange. and instead iris.121'1 **NOOP.** Otherwise, the compiler can seek to insert a useful instruction after the branch. The experience wish harsh the Berkeley RISC and IBM SO I systems is that the majority of conditional branch instructions can be optimized in this fashion ([1[3]AT1'82a], [RADI83]},

A similar sort of tactic, called the delayed had, can be used on LOAD instruc• tons. On LOAD insiructions, the register that is to be the target of the load is locked by the prouessor. The. processor then continues execution of the instruction stream until it reaches an instruction. requiring that register, at which point it idles until the load is compleic. If the compiler can rearrange instructions so that useful work can be done while the load is in the. pipeline, efficiency is increased.

Asa final note., we should point out than the (lesign of the instruction pipeline should not be carried out in isolation from other npl iini 7. a I. ion techniques applied to the system. For example, [BRAD9Ibi show.s that the scheduling of instructions for the pipeline and the dynamic allocation of registers should he considered together Lo achieve the greatest efficiency.

13.6 MIPS R401)(1



One of the first commercially available RISC chip sets was developed by MIPS 'Teehnology inc. The system was inspired by an experimental system, also using the name MIPS, developed at Stanford 1HENN84]. **In** this section we look at the MIPS 84000. It has substantially the same architecture and instruction seI of the earlier MIPS designs: the 82000 and R3000. The most significant difference is that the 84000 uses M rather than 32 bits for all internal and external data paths and for addresses, registers, and the ALL:.

The use of 64 hits has a number *of* advantages oi.rer a 32-bit architecture. Et allows a bigger address space—large enough for an operating system to map more than a terabyte of files directly into virtual memory for easy access. With 1-gigabyte. and larger disk drives now common, the 4-gigabyte address space of a 32-bit machine becomes Iiniiiing, Also, the 64-bit capacity allows the 840010 to process data **such** d4puble-precision floating-point numbers and character strings, up to eight chat deters in a single action.

The R40r)0 processor chip is partitioned into two sections, one vim aining the CPU and the other containing a coprocessor for memory management. The proces- sor has a very simple architecture. The intent was to design a system in which the instruction execution logic was as simple as possible, leaving space available for Logic to enhance performance (e.g., the entire memory-management unit).

The processor supports thirty-two 64-bit registers. It also provides for up to 128 Kbytes of high-speed cache, **hail each** for instructions and data. The relatively large. cache (the **IBM** 3090 provides .128 to 256 Kbytes of cache) enables the system to keep large sets of program code and data local to the processor, off-loading the main rrwmory bus and avoiding the need for a large rcgister file with the accompa• nying windowing logic.

Instruction Set

Table 13.10 lists the basic insiruetion set for all MIPS R series processors. Table 13.11 list the additional instructions implemented in the R4000. A]] processor instructions are encoded in a single 32-bit word format. All data operations are reg- ister to register; the only memory references are pure Load/store operations.

The R411(111 makes no ui.;;I: of condition codes. If in n instruction generates a condition, the corresponding fldgs are stored in a gcneral-purpose register, This avoids the nced for spocial togie to deal with condition codes as Li Icy rifted the pipelining mechanism and the reordering of instructions by 1he compiler_ instead. the mecharasms already implemented to deal with register-value dc1,)endencies are employed.

IA* 13-10 R-Serics Instruction St11

OP	Description	OP	
	Load/Store irattrutitans		Imitruehions
1.11	Load B yte	NI f LT	
	Load By[e Unsigmd	MULTU	Cnsigned
LE-f	E..oad ITO:01 1.1	DIV	Divide
1.1-11-J	LDUCJ Haliviord. Uih.i4Ined		1.:nhigncd
LW	Loud Word	:\0111-1 1	Move from HT
LWL	Load Word 1.4L	MTH!	Move to LEI
L'afrFt	Load Word Righ1	MELD	Move From LC)
.ST3	Store.	MTLO	4.11)VC' In LC)
.SH	S I orc Ha 1 fword	.'1 Li mp	and Etranch lastructionx
SW	wire Word	.1	Jump
SWL	Storu Vte'ord Leff	.IAI .	Jump and].ink
swR	Sion! Word 1R.;g1.0	J R	Jump to Rep.iriEur
	ArlthIllellic Instructions (All! Immediael	JALR	.Jump and Link Regisicr
Anal	Add lrffriwdiaLe	B EQ	Branch on Equal
ADDIU	Add Immediate Uns.ign.nd	BNE	Branch on Nol Equal
	Set on I .c NE Than firimediarc		Bra ach Than (5r Equal LC art)
	Sot o Leas Than Entmediati2 Unqgried	BGT7	Branch c,u GredLin- !hart Zero
ANDI	AND ImmcdiRre	BLTZ	Branch on Loss than 7.cro
ORE	f makudi II	BGEZ	Branch. on Crreutet Than or Equal. to Zero
XDRE	E lasive-OR	BLTZAL	Branch on Less than Z.c.rd. and Link
LL:1	Load Upper EnnoodiaLc.	BUEZAL	Branch oil &Hater Than or equal to Zero ii nd Link
	Arithmetic Insinktions {3-operand, R-typeji	Coprocessor in.struclinn%	
ADD.	Add	LWC:7.	Load Word I pTCK.018ar
ADD(J	Add Unsitmcd		OIL: WOId IA) C1.33113CCSS{ Er
SUB	Suhtract	WIC?:	M I lo Coprocessor
SUBL:	Sub1mcs Unsigned		.%/lovc Irani Coprocessor
	Set on Lcss `Chau	CTC7.	C:onirol lo Coproceswr
SLTU	Set on 1.,c83 Than UnAigncd.		rkloyc Control from Coprocenor
	AND	COP./	Coproi..emiir Oricra d on
OR	OR		Branch tan C7t)procchar E..True
XOR.	Excluivr-OR		krwric la on copt000sErn- z
NOR		Special ineitrutliMIS	
	lihifl I miracli.ons	SYSCALL	S:ystvm (Tall
SLL	Shift 1.1A 1..4>eical	TIRP.AI	Break
SRL	Shift .ogical.		
SRA	Shift RiOn. Aviihnicijc		
SLLV	5filk Left Login!! Iv'ariatd.e.		
S RLV	ShElk Right Lq.iciAl Variable		
SHAY	Shin Righi Arithmetic Variable		

Table 13.11 Additional 84000 InstruyLions

DP	Description	OP	Desffiption
	Load)Store Ingtxiictions		**Emption InstroctionN**
LL	Load Linked	TGE	Trap if GfeaLey Than or .1-..qual
	SLOW: 4.¯ontliLinnal	'EGEU	'Trap if CITC.1 L1' Than or Equal Unsigned.
	Sync	TLT	Trap if lac:NS Foto
	]um p sund **Branch iust1riiciions**	TL'I1.	Trap if LlrisignW
	BT;LE1C Il on. Equal Liktly	EQ	Trap if Ey UiLL
SNEL	liranch on Not Equal Likely	TNE	Trap Not Equal
EZL.	Branch on [..e.s Than or Equal IA Zero Likely	'1(31:1	Trap if Greater 'I..han or P.qual lunnwdinw
BCITZL	Branch on Greaser Than Ze..I0 Likely	'UHL!	Gremer Than Or ECIU a I 1.:ns.iersed Inuit di:pie
BLTZL	Branch on Les-; Itinn Zero L.ikely	TLTI	Trap it Less Than Immediate
BGEZL	Branch on {]realer Than or Equal Lu Zeso Likely	TI..TII:	Trap it Less Than 'Unsigned Im mcdi ate
FiLTZ.A.f.	Branch on Less Than Zero	'J'E.QI	Trap
L	Link LIM y		
BGEZ AL	Branch on Greater Than or Equal	TNE]	Ttap it Nol. Equal Inunediale
L	Lt Zero and Link Likely		
RC./.TL	Brandt on Coppocessor z True Likely		CIITED111.70.4.14 inStrUCtilDilS
(:L FL	DTA rich (in Cop' 5LT2r.SC1T t False	LI)C..e	Load I) 12 bproccmcii SEcire Double Coprocc.Fs.or

Further, conditions mapped onto lhe register files are subject to the same eornpile-ti me optimizations in allocation andl trctr,e t, otlICI' values stored in regbocts.

As with most RISC-hosed machines, the NIPS liSeb. n single 32-hit instruction Length. '1'his single instruction length simplifies inOrtii:lion l'clCII and decode, and it also simplifies the interaction of instruction fetch with the virtual memory manage-ment unit (i.e., instructions do not cross word or page boundaries). The three inslructiori formats, (Figure I 3.R)share common formatling opcoides and register references, simplifying instruction decode, Tlw effect Of more complex instructions can be. synthesized al compile LiInc-

Only [he simples[and most frequently used memory-addressing mode is imp]cryiented in hardware. Al] memory references consist 4.)1 a 1s-bit offset from fl 32-bit register. For example, the 'Load word' in.intrUCtiOni is of the form

```
1w r2,    (r3)    word n.e .ndd.ru!L.  12  ;if   rnn rs.gister 3 intc:   t.5tr. 2
```

Each of the 32 general-purpo5.e rcgiA.er'S can be used as [he bme. register- One reg-i5tcr, r0,..idways contains 0.

The compiler makes use of mu]tipio machine instructions to synthesise addressing modes in convention01 machines. Some examples are prodded in Table 13,12 [CH0W871. The table shows the USG of the instruction Lui (load upper immediate). This instruction loads [he upper hal f {,r a register with a 16-bil immediate value, setting the lower hair 10 zcro.

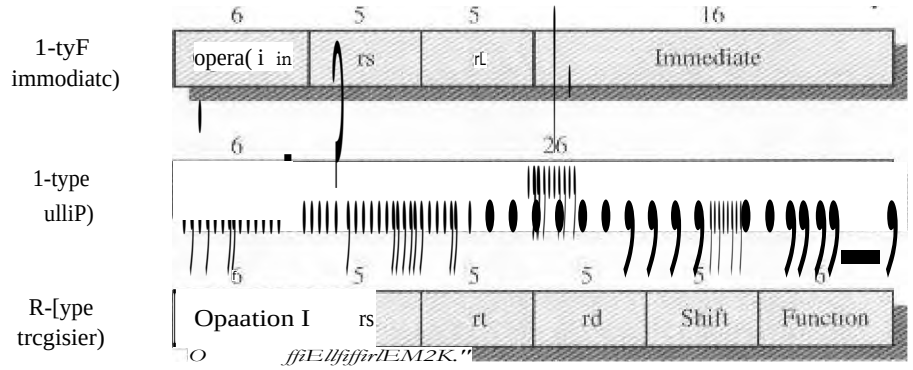
Instruction Pipeline

With its simplified instruction architecture,. the MIPS tit:Neve very efficient pipelining. Ii is instructive to look at the evolution of the MIPS pipeline, as it illustrates the evolution of RISC pipelining in general.

The initial experimental RISC systems and the generation Of commercial RISC prow.ssors achieve exccution spaeds that approach one instruction per system clock cycle. To improve oh this perforinance, two classes of processors have evolved to offer execution of multiple instructions per clock cycle: superscalar and superpipelined architectures. In essence, a superscalar archilect to ˙c replicates each of the pipeline stages so that two or more instructions al the same stage of the pipeline can be processed simultaneously„k superpipelined architecture is one that makes use of more, and more nne-grained, pipeline. slages. With more Aiges, more instructions cAri he in thc pipeline at th.c same lime, increiv,ing

Both approaches have limitations. With superscalar pipelining, dependencies between instructions in different pipelines **can** slow down the syslem, Also. overhead logic is recuired to coordimtc these dependencies, With super˙pipelining, there is overhead associated with transferring instructions from one stage to the next.

Chapter 14 is devoted to a study of superscalar architecture. The MIPS R40.10 is a good example of a RISC-based superpipe]line architecture,



() Nutt iori	01700Litill
rs	Source rcgister specifier
rl	Sourceidestination reis.ter speritit r
hininediat	immediate.. branch, or adcli.ss displacemem
Target	Jump target address
	Dcsiinolii spa iffier
Shift	Shift i ni unt
Function	runciiun spocifier

Figure 13.8 MIPS lrisiruclik˙rn Pm tls

Table 13.12 Synthesizing Other Addressing Modes with the MIPS Addre'ssing MOde

Apparent Instruction	Actual Instruelion
1w r2., ...iffsea>	lw r2, <16-hit (r())
lw 12, <:12-hit (Also}	Ku 1 1, <high 16 hits of Dfisei> 1w r2. <low l6 hits of c>ffset> 01)
1w r2. (r4)	lui r 1. <high IA hi addu r I. r]. r4 lW r2. • ow](.i ,tits of offset:. ir 1)

Figure 13.9a shows the instruction pipeline of she R3000. In the R3000, the pipeline advances once per clock cycle. The MIPS compiler is able to reorder instructions to fill delay slots with code 70 to 90% of the lime. All instructions follow the same sequence of five pipeline stages:

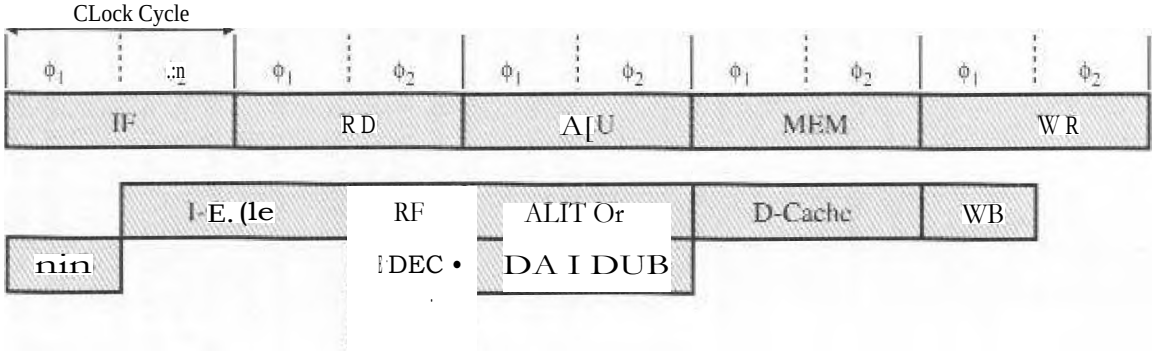
- Instruction fetch
- Source operand fetch from register file
- ALL-) operation or data operand address generation
- Data memory reference
- Write hack into register file

As illustrated in Figure 13.9a, there is not only parallelism due to pipelining but also parallelism within the execution of a single instruction. The 60-ns. clock cycle is divided into two 30-ns magus. The. external instruction and data access operations to the cache each require 60 as, as do the major internal operations (OP, DA, IA). Instruction decode is a simpler operation. requiring only a single 30-ns stage, overlapped with register fetch in the same instruction. Calculation of an address for a branch instruction also ovcrlapS instruction decode and register fetch. so that a branch at instruction *i* can address the !CACI 1E access of instruction *i* - 2. .Similarly, a load at instruction *i* fetches data that are immediately used by the OP of instruction *i* while- an ALA ./ski ft result gets passed directly into instruction 1 with no delay. This tight coupling between instructions makes for a highly efficient pipeline.

In detail. then, each clock cycle is divided into separate stages, denoted as 61 432. The functions performed in each stage are summarized in Table 13,1.3.

The 840011 incorporates a number of technical advances over the 83000. The use of more advanced technology allows the clock cycle lime to be cut in half, to 30 ns, and for the access time to the. register file to be cut in half. In addition, there is greater density on the chip, which enables the instruction and data caches to be incorporated on the chip. Before Looking at the final R4000 pipeline, let us consider how the 830001 pipeline can be modified to improve performance using R400-0 technology.

Figure 1 **3.9b** shows a first step. Remember that the cycles **in** I his figure are half as long as those in Figure 13.9a. Because they are on the same chip. the instruction



J Detailed R31)00 pipeline

(b) Modified R3000 pipeline with reduced latencie;-;

Cycle CyclIL. I Cycle R 2 de

7B R; ALE D-Cache C

c) Optimized R 31).E.X.) pipeline with parallel TLI ral. cache accesses

TF =r im&el)
RD = Read
MEM = Manor.: access
WB = W §jck
f-f";A:hcz = Instrucriou caerm acccss
RF =i»± oporand from regime,-
D-Cachc = Data caLlit
NIB = Instruction addr&•,..; 13 an...Aaiun
!DEC – Ins[ruLlion &code
IA = Compute mw iork .,ttIdrcss
DA = data virtual address
DTE.E1 = Dara address irdrislatiou
TC = Ddia HChe tag check

Figure 13.9 Fnhaircing the R3000 Pipelirrt

Table 13.13 83000 Pipulint. Siam;

Pipeline	Plum	Function
IF		using shu rilniikaw an instrircLion virCualaddross to Fl ph IITSI CH I .:1 11(1 Ll:Sti 4 I LiA" N 111W IlL11111 drCaSlprS i.
IF	1^{12}	thr cm I Hd them, Lk, LhE irisilLICLiira address.
RD	41	Return instruction from irisirugiion
		GOIllipare. sags and validity or luichcli
	$_{1}P^{2}$	Decade instruction.
		Rued register file.
		branch..calestlase braach target ackdress.
A1.1.!	.131 + .02	1.1 op ;ration, the arilhrrioic or Ingi CkI I Ope.raLi0(1 IS• perlDrrned.
A I	.131	11 a brunch, docide vel'i ther the branch is to in lakw, tar nut.
		mEniUTY rtft rcii.fiL (load cir store). calcutHtc data vi Luak
ALL.'		TI H mLnl nry rc 1trr n Ce. Ltamlate. data virtual address CkI I usi ng
MEM	4.11	Iry Inc wiry rLIcmhIEC. r412 n address Le•thiLa cache.
M EF I	02	IF H rLI't.rc nLL ru L u M data from data cache, and clicck
1413	ol	Write to regisler lilt ,

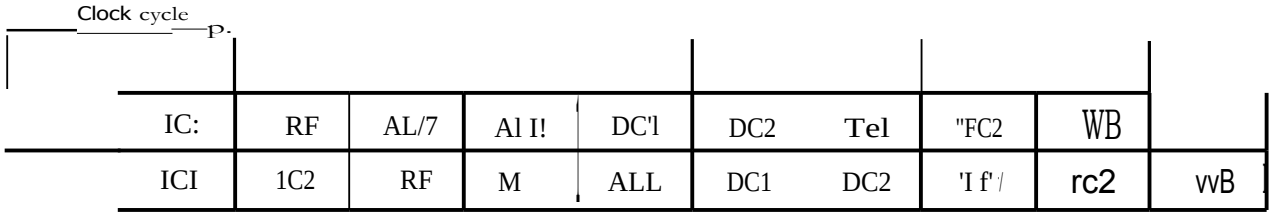
and•data cache stages take only halt as long so they still occupy only one clock cycle, Again, because of the speedup of ihe rc.gisLu file access, register read and wri1C still occupy only half of rr clock uycle.

1[-Ice; LiiL R4000 caches are on-chip, the virtual-to-physical address translation can delay the cache access. This delay is reduced by imptementin2 virtually indexed caches and going Lo a parallel cache access and address trandEll **Figure** L3,9c show the optimized fOtgX.) pipeline with this improvement. Hecause of the compression of eve 11 tS, 1.11 c data *cache* tag check is purfouncd separately on the next cycle after cache ziccess,

In f';u perpi pc. I i fled syz,teni, existing hardware is used several li mes per cycle by inserting pipeline registers to split up each pipe stage- kssentially, each super= pipeline stage operates at a mullipie of the base clock frequency, the multiple. depending on the degree of supcxpipelining. The R400() technology has the .petal and density to permit wperpipLdining of degree 2. Figure 13.10z1 shows Ole vd- mixed R3000 pipeline using this superpipelining. Note that this is essentially the same dynamic structure as Figure 13.9c.

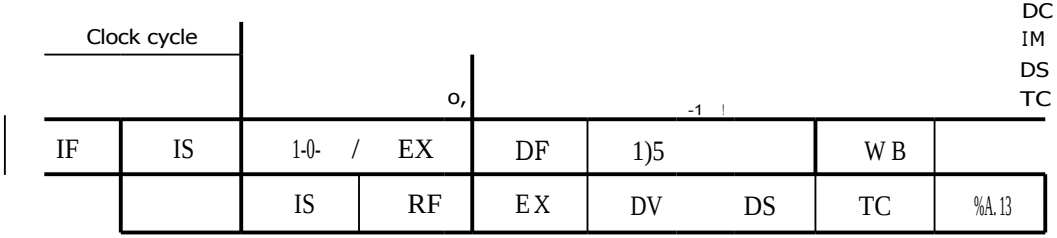
Further improvemcril's can he made, For the 84000, a much larger aid special lined vals designed. This makes it possible to execute ALI! operaticfns M twice the rate. Other improvements allow I hc QXC:CL11.100. Of loads and stores at 1vvicz the rate. The resulting pipeline is shown in Figure 13.10b.

Thu F 4 Il Hk hAs eight pipeline stages. meaning that ws rnany as eight instructions C4111 I)C: in the pipeline at the same time. The pipelinc .,liivarices at the: rate of two per dock cycle. The .eight pipeline stagQs are EN fO3IOWS1



(a) Su])...wipillineft irnplmentaion of the opti rnit.cti 83000 pipeline

IF = Instnicrion fetch first half
IS = Inslruction Ictch second half
RF = Fetch operands fet-gri reaisler
EX — Instruction cxccutc
= Insnitaino cache
DC = Data cache
IM = 1).am cadie aril. half
DS = Data cache second half
TC = Tabu clsca.



R4000 pipeline

Figure 13.10 ThoureLical R3017O and Actual 84000 Superpipclincs

- **instrilvtiun fetch first half:** Virtual address is presented to the instruction oche ;lnd **the** translation look aside buffer.'
- **'In traction fetch second half: Instruction cache. out** l ll **ui 1** Lic..on and the TLB generates the physical address,
- **Register Me:** Three 'activities occur in parallel!
  - Instruction is decoded and check made for inierloek conditions (i.e., this instruction depends on the result of a preceding instruction).
  - c Instruction cache tag check is made.
  - Operands are fetched from the register file.
- **Instruction execute:** One of three activities can occur:
  - c. If the instruction is a register-to-register operation, the ALU performs the arithmetic or logical opera tion.
  - O If **the imorto** ion is a load or store, the data virtual address is caleulated,
  - lf the instruction is a branch, the branch target virtual address is calculated and branch conditions arc checked.
- **Data cache first:** Virtual address is **presented** to the **data** cache and **TLB.**
- **Data cache second: Data** cache outputs the instruction, and the TLB generates the physical address.
- Tag check: Cache tag checks are performed for loads and stores.
- **Write buck;** Instruction 'vial! written back to register file.

13.7 SPARC

SPARC (Sealable Processor Architecture) refers to an architecture defined by Sun Microsystems. Sun developed its own SPARC' implementation but also licenses the architecture to other vendors to produce SPARC-compatible machines. **Tim. SPARC.** architecture is inspired by the Berkeley RISC I machine. and i1 instruction set and register organization is based closely on the BerkeIcy RISC mode].

SPARC Register Set

As with the Berkeley RISC. the SPARC makes use. of register windows. Each window consists of 24 registers. and the total number of windows is implementation dependent and names from 2 to 32 windows. Figure 13.1 I illu bates ail implementation that supports S windows, using a total of 136 physical registers; as the discussion in Section 112 indicates, this seems a reasonable number of windows. Physical registers 0 through 7 are global registers shared by all procedures. Each process sees logical registers 0 through 31.. **1,00w1** registers 24 through **31,** referred to as *jai,* are shared with the coiling (parent) procedure; and loaical registers 8 **through 15,** referred to as *outx,* are. shared with any called (child) procedure, These two portions c:pverlar with other windows. Logical registers lf) through *a* referred to aLS */0i:ids.* are not shared and donut overlap other windows. Again, as the discussion of Section 12.1 indicates, the availability of 8 registers **for parameter** passing should lie adequate in most cases see Table 13.4).

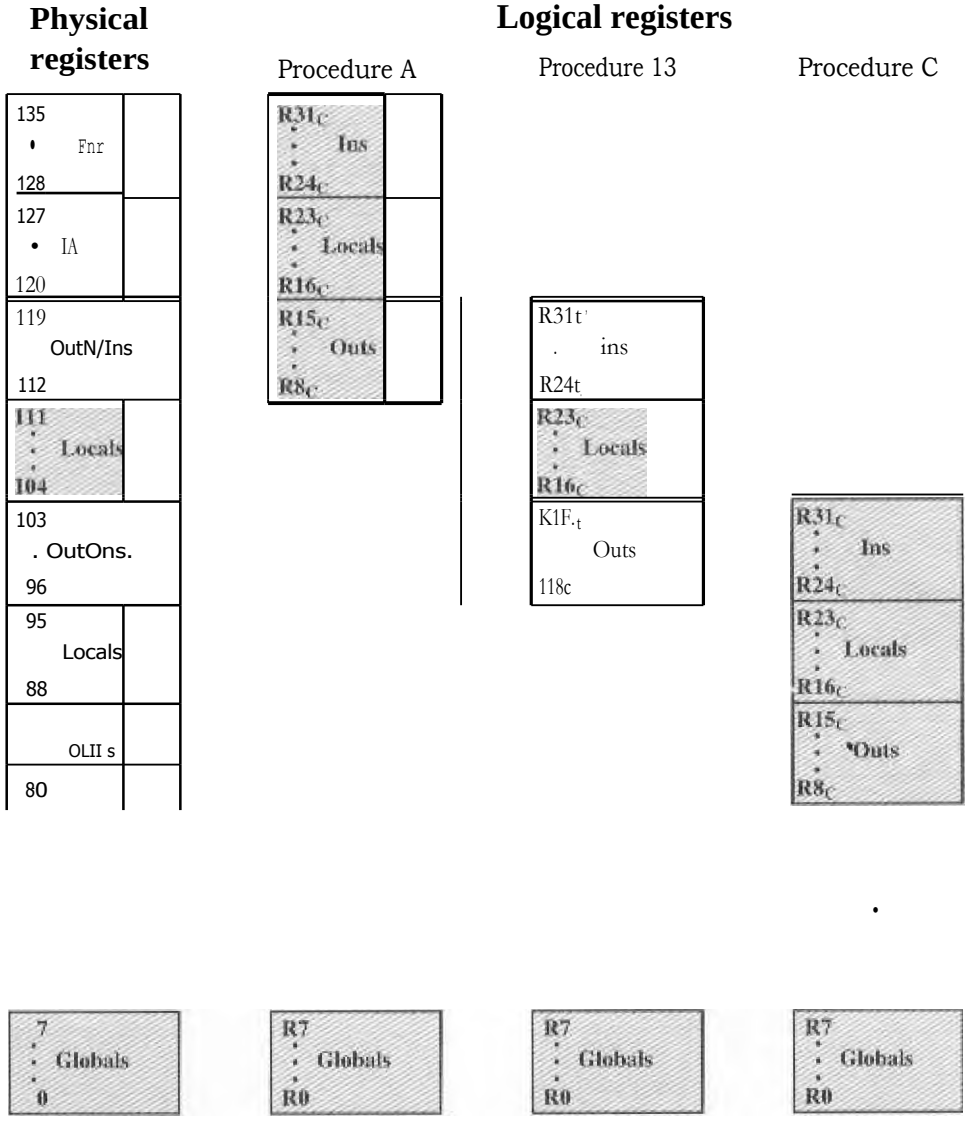


Figure 13.11 SPARC' Register Window **La**yout with Three Procedures

Figure 13.12 is another view of the register overlap. The calling procedure places any parameters to be passed in its out registers; the called procedure treats these same physical registers as it *ins* registers. The processor maintains a current window pointer (CW1 [3]). located in lhe processor status register (PSR), that points to the window of the currently executing procedure. Thc window invalid mask (WINI). also in the PSR, indicates which windows are invalid.

With the SPARC register architecture, it is usually not necessary to save and restore registers for a procedure call. The compiler is simplified because the corn-

piler need be concerned only with allocating the local register* for procedure in an efficient manner and need not be eonuCTni,:d with register allocation between procedures.

Instruction Set

Table 13.14 lists the instructions for the SPARC architecture. Most of the instructions reference only register operands. Register-to-register instructions 1 ⁻ ve three operands and can be expressed in the form

$$R_d \quad R_{si} \text{ op } S2$$

1 „ and it„„ to re register refereno....!i; S. can refer ei1 her to 4l rcgistir or to a 13-bit iiniiic-di2itc operand. Register *zero* (R,,) k hardwired with the value 0. This form is well suited to r!,'pieai pt.ograms. which have a high proportion of local scalars and vonstanis.

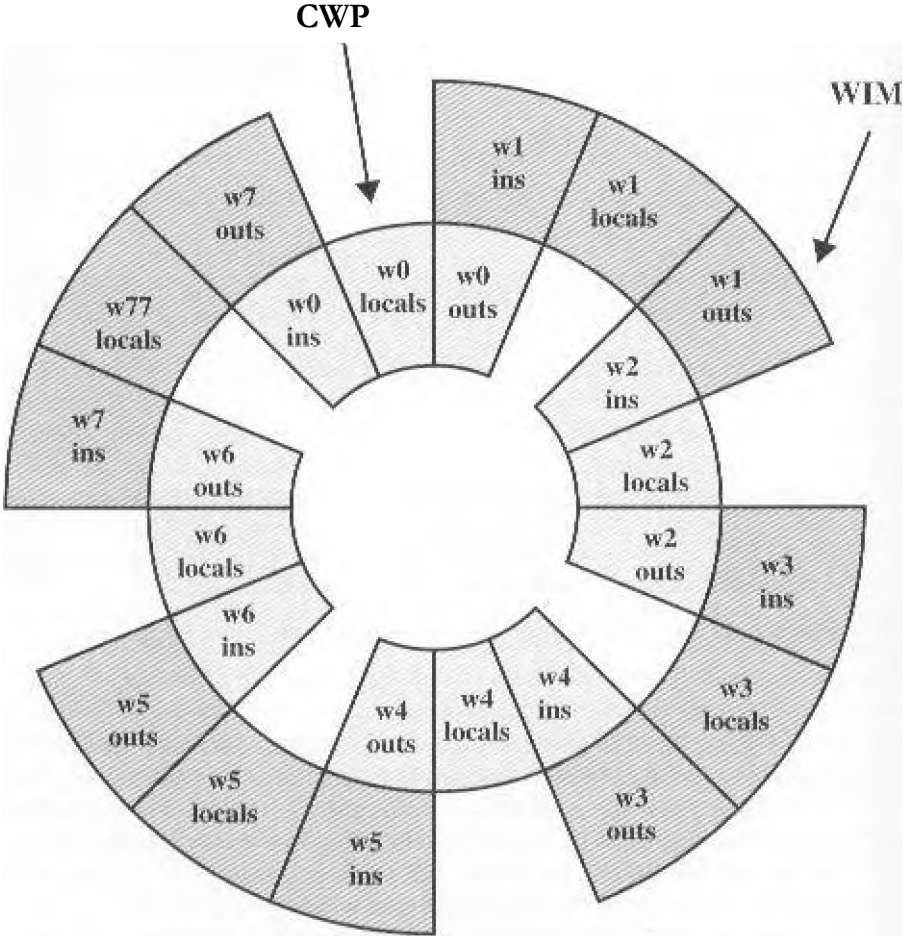


Figure 13.12 Eight Register Windows Forming a Circular Stack in SPARC

TtbI **13.14 SPARC instruction Set**

OP	Oesetiplion	OP	Description
Load/Store Instructions		Arithmetic Instructions	
1...1)8EI	**Load** 3ign.e.11 h .sc	ADD	**Add**
LDSH	**Load signed** halfword	**A DDCC**	**Add, set ice**
LDEB	Load **unsigned byLi...**	**ADDX**	**Add with carry**
LDL:H	**Load unsigned hallword**	AllaDiNCC	**Add with carry.** set icc
LD	**Luad word**	**SUB.**	**Subtract**
LDD	**Load douhkword**	**SUB CC**	Sill) LniCi, set ice
STR	Store **byte**	**SUBX**	**Sulhiniet ww1 carry'**
STH	**SLoretalipeord**	**SL:BNCe**	**Subtract with carry, set icc**
STD	**Store word**	**MULSCX:**	Multiply stop, set Lee
STDI)	**Store** douhleword	Jump/Brandi lug melons	
	Shift Instructions	11C('	**Branch on condition**
SLL	**Shirt kit logic:II**	**FBCC**	**Brandi** an **floating-point** con **ditio** n
SELL	**Shift** right **logical**	**CBCC**	**Brunch on coproccsmn conilition**
SRA	**Shirt** Tight 211M1111 e Fie	**CALL**	**ColL prucoiLere**
	BIPOICHIL IIINiFtICtioM5	**i MPL**	**Jump and link**
AND	**ANT)**	**To.:**	Trap **on condition**
es.N DOC	**AND, set ice**	**SAVE**	**AdvHnec rcgigcr window**
A\DN	**NAND**	**RESTORE**	Move. mndows backward
ANDNCC	NAND. set i(,..7.;	**RETT**	Routru **front** heap
OR	oR	Miscellaneous instructions	
0.14C(.'	**OR, set lee**	**SETH]**	Scr high ?..7,.: bits
0 RN	NOR	**UMW**	*Un* im plemented IDSLFUCEi ors **(trap)**
ORNCC	**NOR.** ret **ice**	**RD**	**Read a speciai register**
XO E2	XOR	**WR**	**WTI Li.' d Sp:Cla l register**
X **oftcc**	**XOR, set ice**	TFLILS'll	**Imitruction** cache. flush
XNOR	**Exclusive NOR**		
X.NOROC	**Exclusive N'OR.:..i•I ice**		

The available ALL: operations can he grouped as follows:

- Integer addition (with or without carry)
- Integer subtraction (with or without carry)
- Bitwise Boolean AND, OR, XOR and their negations
- Shift left logical, right logical. or right aril hmel ic

All of these instructions, except the shifts, can °pi ionally sei the four condition Codes (ZERO. NEGATIVE, OVERFLOW. (:'ARRY). Signed integers are represented in 32-hil twos complemen I form.

Only simple load and store instructions reference memory, Then: arc separate load and store instructions for word (32 bits), doubleword, halrword. and byte. For the latter two cases. [here are]nslructions for loading these quantities as signed or unsigned **num** bers. Signal num *bers* are sign extended to fill out the 32-bit dcstina• Li on register. Unsigned numbers are padded with zeros,

The only available addressing mode, other than register, is a displacement mode. That is, the effective **OT** *an* operand consists of a displacement from an address conl wined in ri rcgisler:

$$EA = (R + 52$$
$$\text{or } EA = (Ii_{,,}) + (R_{52})$$

depcnding tin whuther the second operand is immediate or a register rclerefice. To perform a load or store, an extra stage is added to the insivueiion cycle. During the second stage, the memory address is ciikulted using the ALL:; the load or store occurs in a third stage-' t'hi74 single addressing mode is quite versatile and can he used to synthesize other addressing modes. as indicated in Table 13, J 5.

It is instructive to compare the SPARC addressing capability with 'hal of the MIPS. The MIPS makes use of a I 6-bit of kr2 c, wmpared with a 13-hit offset on the **SPA RC,** On Elie other hand, the MIPS does not permit an address to be constructed from the contents of two registers.

Instruction Format

As with the **MIPS R4000, SPARC** uses a simple set of 32-hii instruction formats (Figure 13.13). All instructions begin with a 2-biL oprode, For most instructions, this

'aw 13.15 Synihesi4i ng Other Addressing Hades with SPARC Addp:=:•5sinp, Andes

Mode	Algorithm	SPARC ICquivalera	Instruction Type
Tm meal **ate**	ope rand A	S2	Rcgis%1 La rEgisiel
Direct	EA - A	R , + .S·"	Load. SLOie.
Ft.e.gkicr	**FA** R	Rsl' L..	RegiEte.r10 rcgricr
R egi N i t1.1 mdireci	EA 1:R:i	It,, F it	1...i5a4.1, slDre
DispEat:emelst	EA i.R .11 A	Ri.; 1 t S2	LA5k14,1. ADM

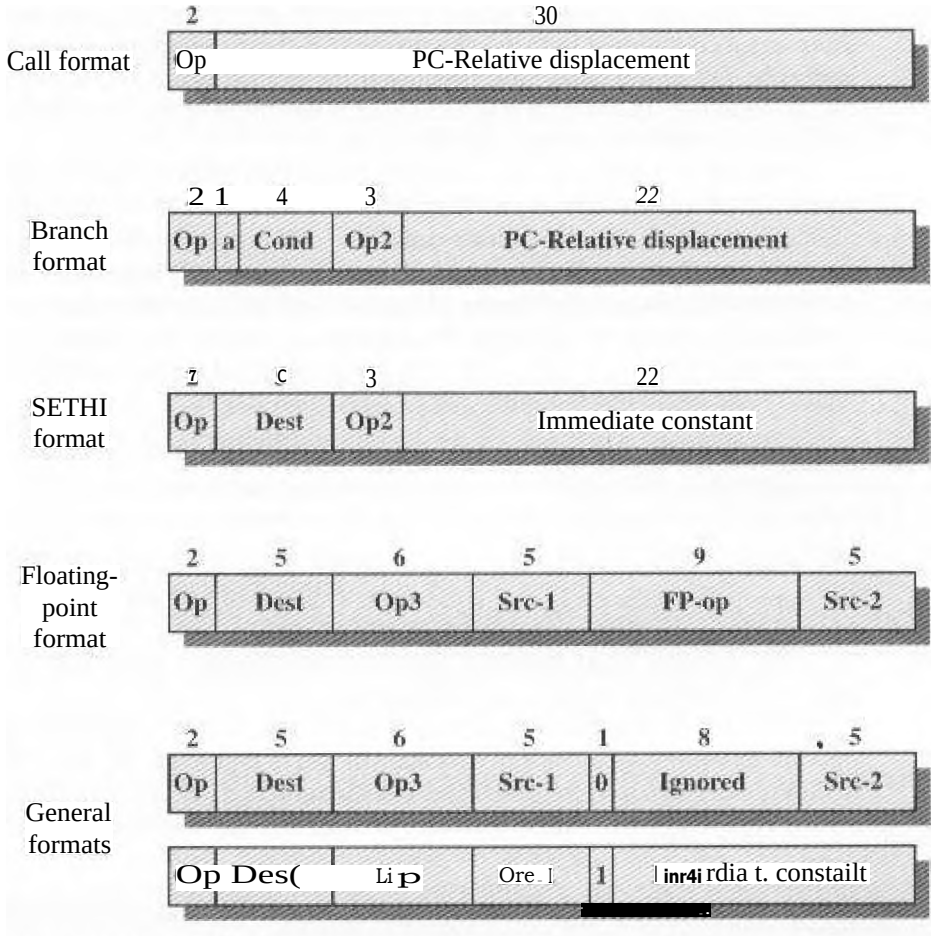


Figure 13.13 SPARC Instruction Formats

is extended with additional opcode bits elsewhere in the format. For the Call instruction. a 30-bit immediate operand is extended with two zero hits to the right to form a 32 hit PC relative address in twos complement form. Instructions arc aligned on a 32-hit boundary so that this form of addressing suffices.

The Branch instruction includes a 4-hit condition field that corresponds to the four standard condition code bits, so that any combination of conditions can he tested. The 22-hit PC-relative address is extended with two zero bits on the right to form a 24-biz twos complement relative address. An unusual feature of the Branch instruction is the annul bit. When the annul bit is not set, the instruction after the branch is always executed. regardless of whether the branch is taken. This is the typical delayed branch operation found on many RISC machines and described in Section 13_5 (see Figure. 13.7). However, when the annul hit is set, the instruction following the branch is executed only if the branch is taken. The processor suppresses the effect of that instruction even though it is already in the pipeline. This

annul bit is uscful because it makes it easier for the COnlitiliZT it) fill the delay slot following a conditional branch. The instruction that is the target of the branch can always be put in t]ie delay slot, because if the branch is not taken, the instruction can be annulled. The reason this technique is desirable is that conditional branches are generally taken more than. half t]ie time.

I he II instruction is a special ins.;truction used to load or store a 32-bit value. This feature is needed to load and store addresses and large constants. The SETH I instruction sets the 22 high-order bits of a register with its 22-hit immediate operand, nd zeros out the low-order 10 bits. An irntn.ediate constant of up to 13 hits can be.wecified in one of the general l'ornIes. and such an instruction could be used to fill in the remaining HI hits of the register. A load or store instruction can also be used to achieve a direct addressing mode. To load a value from location K in memory, we could use the following SPAR(' instructions:

```
NrE              ;lo:aa iligb-orAer 272 hLts of ddar42SS.
                          K iri=0 registGr LB
- tiu(K)1, atrE  ;load conten:= Df          K      rS
```

The macros %hi and %Iu art: lised «) define immediate operands consisting of the appropriate address hits ot a location. This use of SETHI k similar to the use of the LUI instruction on the MIPS (Table 13,12),

The floating-point format is used for IThating-point operations. i w o s ou rce and one destination registers are designated.

Finally, all other operations. including loads- slores. arithmetic, and ic.qical operations use one of the last two formals shown ilk Figure 13.13, One of the forEnats makes use of two source registers and a destination register. while the other uses one source regisler, one I:4-hit immediate operand, and one destination register.

13.8 RISC VERSUS CISC CONTROVERSY'

For many years the general tren4 l in computer architecture and organ isation has been toward increasing processor complexity: more instructions. more addressing modes. more Speci al ized registers. and so on. The RISC movement represents a fundamental break with the philosophy behind that trend. Naturally, the appearance of RISC' systems. and the publicafion of papers by its proponents extolling RISC virtues. led to a reite1ion frorn tho!,e involved in the design or CISC architectures,

The work that has been done on assessing merits of the RISC approach can kc grouped into two categories :

- **Quantitative: Attempts** to compare program size and execution speed of programs on RISC' and CISC machines that use cumparabJe technology
- **Qualitative:** Examination of issues such as high-level language support and optimum use of VLSI rca I estate

Most of the work on quantitative assessment has been done by those working on RISC: systems I PATT82b, HEAT 84. rxr rs4]. and it has been, by **and** large, favorable to the RISC approach. Others have =mined. the issue and come away

13.10 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

CO11i Ic\ithrEMliofl set computer ((ASC.)	high-kyel language (HLL)	reOsicr file
delayed branch	reduced instruction set computer (RISC)	register window
delayed load		SPARC

Review Questions

13.1 What are some typical distinguishing characteristics of **RISC** orgaiiiiation2

13.2 Briefly explain the two basic approaches used to minimw.e register-memory operations on RISC. machines.

13.3 If a circular register buffer is used to handle local variables for nested procedures. describe two approaches for handling global variables:.

13.4 What are some typical characteristics **of** a RISC instruction set architecture?

13.5 What is a delayed branch?

Problems

13..1 Considering the call-return pattern in Figure 4.16, how many overflows and underflows (each of which causes a register savelrestore) will occur with a window size of

a. 5?

h. 8?

c.

13.2 In the discussion of Figure 13,;.. it was stated that only the first two portions of a Window are saved or restored. Vir'lly necessary to save the temp/1r ai IeTisters?

13.3 We wish to determine the execution time for a given program using the various pipelining schemes discussed in Section [3.5. Let

N = number of executed instructions
D = number of memory accesses
— number of jump instruct ions

For the simple sequential scheme (Figure 13.6a), the execution time is 2N T D stages. Derive formulas for two-stage. three-stage, and four-stage pipelining.

13.4 Consider the following code fragment in a high-level language:

```
tar r in 1          loo2
              + QM.VAL
end loops
```

Assume that 0 is an array of 12-hyte records and the VAL.. **field** is in the first 4 bytes of each record. Using WNW) code, we can compile this program fragment as follows:

```
          ECX,.                          ECX     11010 I
          FAX, 'KZ, 32    r DoL ciffu% in EAK
          tax, Q[UX]          VA!' C:el.:.
AUD       S. EflX
DR'       ECX
A•E       I,,F                :          = Ler:
```

unconvinced ICOLW85a, FLY N87, DAN-187], There are several problems with attempting such comparisons I SERI_861:

- There is no pair of RISC and CISC' machines that are comparable in life-cycle cost. level of technology, gate complexity., sophistication of compiler, operating system support, and so on.
- No definitive test set of programs exists. Performance varies with the program.
- It is difficult to sort out hardware effects from effects due to skill in compiler writing.
- Most of the comparative analysis on RISC has been done on "toy" machines rather than commercial products. Furthermore, most commercially available machines advertised as RISC. possess a mixture of RISC and CISC characteristics. Thus. a fair comparison with a commercial, "pure-play" C1SC machine (e.g., VAX, Pentium) is difficult.

The qualitative assessment is, almost by definition, subjective. Several researchers have turned their attention to such an assessment ICOLWK5a, WALL851, but the results are, at best, ambiguous, and certainly subject to rebuttal I PA'118.5b] and, of course, counterrebuttal [COLW85b].

In more recent years, the RISC **versus** CISC controversy has died down to a great extent, This is because there has been a gradual convergence of the technologies. As chip densities, mid raw hardware speeds increase. RISC systems have become more complex. At I he same Lime, in an effort to squeeze out maximum performance, CISC designs have focused on issues traditionally associated with RISC, such as an increased number of general-purpose registers and increased emphasis on instruction pipeline design,

13.9 RECOMMENDED READING

'Tex **nhoo** ks with **:45)0(1.** LA IV **eoI** 124.! Rim concepts are IWARD901. [PATT9SI. and [HENN961. IKANE92 1covers the commvi(Aill MIPS machine in detail. IMIRA92] provides a good Overview of the **MIPS R4000. I** BASH911 discusses the evolution from the. R3000 pipeline to the 84000 superpipeline. The SPARC k covered in some detail in [DEWA9U1.

8.4.M1191 Bashteen, Lai, J. and **lqullan, J. "A** Superpipeithe Approach to the MIPS Architecture. ¯ *Proceedings, CO M PCON Spring '01* Fu.bruaq, 1991.

DEW .4911 Devikr, R.. and Snooqui. M. *141kroprocessors:* A firrip-oruoter's.Vicw, York! I c6 **raw-Hill,** 1•!.)90.

HENN96 Hennessy, J., and Patterson, I), *Computer Architecture: A* **QUandielfiVO** *Approach.* San Mateo, CA: Morgan. Kanfinano, 1996,

KAINE92 Kane, G., and Heinrich, ***J. MIPS RISC*** *Arch; torture.* Englewood Cliffs, N. Prentice Hall. 1992,

M1RA92 Mirapuri, Woodocre. M.; and Vasseghi, N. "The MIPS R4000 Processor." *IEEE Micro,* **April 1 W2.**

P ATI% Pattemm. **D., and Herincssy,.T.** *C7omputer Organizariop and Dasign: TIhr 1 fardwaril Software IIlieu Pa.* Sari Mateo. CA! Nforgan Kaufmann, 1998.

1NARD90 Ward, S., and **lialskad, R,** *Compurafion Str• iEercs.* Cambridge, MA: MIT Press, 1990.

This 'anagram makes use of the 'MU L instruction, which mul1iplies. the second operand by the immediate value in she third operand and places the result in 1he first operand (see Problem 1.0,13). A RISC advocate would like lo demonstrate that a clever compiler can eliminate unnecessarily complex insi ructions such as IMUL. Provide the demonstration by rewriting the above 810x 8n program without using the IM11, instruction.

13.5 Consider the following loop:

```
        :=
f     K  :=     to IOC 60
            S  I  = E - K!
```

A straightforward translation of this into a generic assembly language would look something like this:

```
        LD      RI, C                   vedL,13 uf E      R:1
        LD      R2,                     value ef f if.    R2
LP      SUP     R1, RI, R2          ;= S - K
        135Q         :a0, U.:417        tf R = 1C)
        is17213                         i317rt.mept K
                LP                      ec star= cf _cop
```

A compiler for a RISC machine will introduce delay slots info this code .4o Mill the processor can employ the delayed branch mechanism, **The** *J b1P* **instruction** is ro ideal with, instruclion is always followed by the 51.111 in traction: bore-(ore, we can ...imply place a copy or IllySt Ili i nso Limon in [fie delay slot after the AI P. The RN) presents a difficult}?. W a ea n • Ie the **t he lode** as is, heca use the ADD instruction s.c..ould then he execuied one loo many times. Therefore. a NOP instruction is needed. Show the resulting code,

Add entries for the following processors to **Table** 13.8;

a. PtNitium Ill

h. PowerPC

13.7 In many cases. common machine instructions that are not !islet] as part of the MIPS instruction set can be, synthesized with a single. MIPS instruction. Show this *fol* the

a. Register-to-register move

h. butrement, decrement

c,, Complement

d. Negate

e. Clear

13.8 A SPARC implementation has .LK register windows. What is the number N of physical registers?

133 SPARC is lacking a number of instructions commonly found on CISC machines. Some of these are easily simulated using either register RO. which is always set to .0, or a constant operand. These simulated instructions are *called* pscudoinstruetions and are reco.E.,rnized by the SPARC. compiler. Show how to simulate the following pseudoinstructions. cash with a single SPARC instruction. In all of these. src and dst refer to registers. *iliar,* A store to RO has no effect.

a. YIONI src, dst
b. COMPARF srcl, src2
c. TEST srel
d. NOT dsi
e. NEC tist
f. INC dst
g. DEC 41,1
It, C dst
i. NOP

13.10 Consider the following code fragment

```
it ic > I r
     L  !=    4
el me
     T.  :=    -  I,
```

A straightforward translation of this statement into $1 3'.2t RC assembler could take the.. following form:

```
-5 thL 21'11          2              rluad hLg:a-Drder 22 biE of ad: ess
                                     rcf ioDeition K Lato reuLster r2
Ld      frKrA +              %]:8          Joati.f.ELE5 oY locaDLcn f. ir.LQ f8
crp      rfi,  lc                          03rape,i0 ccIELeiL.s      te        le
Die     Ll                                ;bxana              10
:top.
aehi                5 ):9
Ld      [FE:rD 1 %10; 11, %r9       :lnad 2on7.ento of 1c)r.F_r.i.op N jnrc,. 79
Lric    ;NrE'                       :add 1 ro
        ]          trIO
■       5r9, 11r1C      :t 1c(Lj]   ; store        intc 1ca7.Lcn L


Ll: set:M
    ld              -            %r12 ;load ccnDoat       Lccation K intu t=2
    dec     %1.'12                        :.dl:3'LtwoL _  = LOTEI  tf12
    sethi             qcr.fl
    s       Stri2,   Futr13 +           ;FAcr:'                 : or2aLi.i.r. I
```

The code.coutairts a uop attu wash brauch instruction t& per.rnit tielayctlimineli operation.

a. Standard compiler optimizations that live CIO *1111 RISC' machines; are gcnerally cffective in being able to perform two transformations 4)11 **the ibleiZO[fla** elide. Notice. that two of the loads are unnecessary and that the. two stores can be imrged if the stork: is rnovcd to a differcnt place in thc code. Show the program Om making two changes,

h. It **now** pa:Nil'AL' hi Nrl'1it1n4rtrr1t oplintif;ilions peculiar LE1. SPARC. The n op af(V" the ble can he **Fi2.1}hict'Ai** h. mu FiL•i iik-(ion into tha(delay slo(and Sc(-Ong the annul l i1 on the Nu irt 'Lotici ion [..'N[ressed at, blo,a L11, Show 411E; program after this change-

r'. Them arc now two unnecessary instructions. Remove these and show the resulting program.

CHAPTER 14

INSTRUCTION-LEVEL PARALLELISM AND SUPERSCA LAI& PROCESSORS

14.1 Overview

Su persca(ar V rslis Superpipelined

14.2 Degigu Isgues

Irmtruct]on-1.4:Ncl Purkllelismiirid Machine, ranzilarF•m
Instruction Issue Policy
Register Renanling
Machine Parallciism
Brana PredieLion
Superscalar Execution
Superscalar Implementation

14.3 Pentium 4

FronL End
Oul-uf-e..)rEr Exocution is
Integer Enid Fioting-Poin# Ex.uction Units

.l4.4 Power PC

Powe *rl-"C 601*
Branch Processing
Poweil)C 62.0

14. Recommended Reading

14.6 Key Terms, Review Questionm, and Problems

Key Terms
evic.w Questions
Problems

KEY POI Nas

- A supcirsc;)1.ar processor is one in which multiple independent instruction pipelines are used. Each pipeline. conskts of multiple stages, 1 o that each pipeline can handle multiplc instructic,nis at a time. Multiple pipelines intro, duce a new level oiparaileiistn. enabling inultipio.strcarns of instructions to he. processed at a time. A superscalar processor explr.lit . iivhut is known as instruction-level parallelism, which refers Ire the degree to which the instructions of a program can be executed in pnaliel,
- A superscalar processor lypically fetches multiple instructions at a time and then attempts to find nearby instruetions that arc independent of one another (.l emi therefore he executed in parallel. 11 the input to One instruction depends en the output of a preceding instruction, then the latter instruction cannot complete execution at the smne time or before the former in. struction, *Once* such dependencies have been idani ified, the processor may issue and complete instructions in an o] der that differs from that 0r the ofig-ilia] machine code.
- The processor may eliminate sortie unnecessary dependenciin by the use of additional registers and the renaming of register references in Lhe tit gina I cede.
- Whereas pure RISC processors 0.1 e11 employ delayed branches to masimiec the utilization of t *he* illStrliCi ion pipeline. this method is kc s appropriate to a superscalRr machine. Instead. most supc rsca jar machines use traditional brand) prediction methods to improve citieieney.

A Superscalar impleinentation of a processor archiieeiurc is one in which cornmon instructions-integer and floai ing-point arithmetic, loads. Flores, anti conditional branches..--can be initiated simultaneously .ii nd executed hide.-penden tly. Such implementations raise a number of complex design issues related to the instruction pipeline.

Superscalkir design arrives oil I he senile hard on the heels of RISC.' architecture, Although the simplified instruction set architecture of a RISC machine lends itself readily to Kuperscalar techniques, the superseakir ipproach can be used on either a RISC' or CISC architecture.

Whereas the gestation period for the arrival of commercial RISC' machines from the beginning of true RISC research with the IBM 801 and the Berkeley RISC I was Stro2.11 or eight years, the first superscalar machines became commercially available within just a year or Iwo of the ctrining of the term *superscafim* The L$' scatar npprL e has now heeornc...• the standard method for implementing high-pe[101 iii.iiiiec microprocessors.

In this chapter, we begin will h an overview of the superscalar approach. vonirasting it with supcipipelining. Next, we present the. key design i ',,sile., associated w.ilh supuscihn- impleiricribition. Then we look at several importhiii examples of supursealar architecture,

14.1 OVERVIEW

The term *superscatur.* first coined in 1987 1AGERK7], refers to a machine that is designed to improve the performance of the execution of scalar instructions. In most applications, the hulk of the operations are on scalar quantities. Accordingly, the supersealar approach represents *the* next step in the evolution of high-performance general-purpose processors,.

The essence of the superscalar approach is the ability to execute instructions independently in different pipelines, The concept can he further exploited by allowing instructions In he executed in an order different from the program order. Figure 14.1 shows, in :.T.enera I terms, the superscalar appri.ich. There are multiple. functional units, each of which is implemented as a pipeline, which support parallel execution of sevend instructions. In this example, two integer. two floating-point, and one memory (either load or store) operations can he executing at the same time..

Many researchers have investigated superscalar-like processors, and their research indicates that some degree of performance improvement is possible.. 'fable 14.1 presents the reported performance advantages. The differences in the results arise from differences both in the hardware of the simulated machine and in the applications being simulated, lti

Superscalar versus Superpipelined

An alternative approach to achieving greater performance is referred to as superpipelining, a term first coined in 1988 IJOUP881. Superpipelining, exploits the fact that many pipeline stages perform tasks that require less than half a clock cycle. Thus, a doubled internal clock speed allows the performance of two tasks in one external clock cycle, We have seen one example of this approach with the MIPS R4000.

Figure 1=1.2 compares the two approaches_ The upper part of the diagram illustrates an ordinary pipeline, used as a base for comparison. Hie base pipeline issues one instruction per clock cycle and can perform one pipeline stage per clock cycle. The pipeline has four stages: instruction fetch, operation decode, operation exert'.

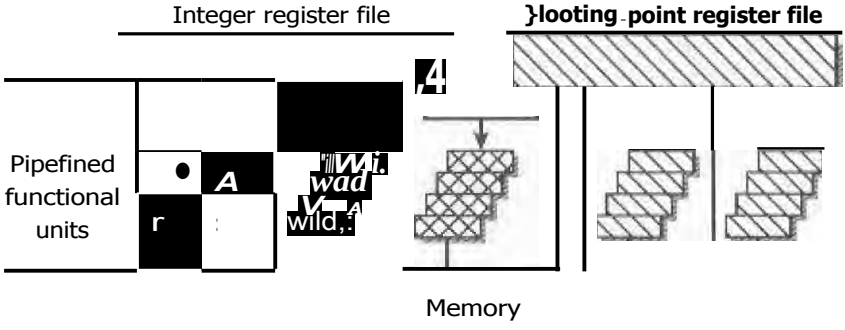


Figure 14.1 General Superscalar Organization ICO1titE95

Table 1.4.1 Reported Speedups of Supersea tar. Like Machines

Reference	Speedup
prIAD7111	L.s
[1c UCK72 !	8
I WEISS41	1,58
I ACOS86I	2.7
[S0HI90]	1.S
ISIMI'FRY]	2.3
1.1 011PS9h]	7,2
I LEFL91 I	7

lion, and result write back. The execution stage is crosshatched for clarity. Note that although several instructions are executing concurrently, only one instruction is in its execution stage al any one time.

The. next part of the diagram shows a superpipelined implementation that is capable of performing two pipeline stages per clock cycle. An alternative way of looking at this is that the functions performed in each stage can be split into two nonoverlapping parts and each can execute in half a clock cycle. A superpipeline implementation that behaves in this fashion is said to be of degree 2. Finally, the lowest part of 1he diagram shows a superscalar implementation capable of executing two instances of each stage in parallel. Higher-degree superpipeline and superscalar implementations are of course possible.

Both the superpipeline and the superscalar implementations depicted in Figure 14.2 have 1hc same number of instructions executing at the same time in the steady state. The superpipelined processor falls behind the supersealar processor at the start of the program and al each branch target.

Limitations

The superscalar approach depends on the ability to execute multiple instructions in parallel. The term instruction-level parallelism refers to the degree to which, on average, the instructions of a program can be executed in parallel. A combination of compiler-based optimization and hardware techniques can he used to maximize instruction-level parallelism. Before examining the design techniques used in superscalar machines lo increase instruction-level parallelism. we need to look at the fundamental limitations to parallelism with which the system must cope. fJOHN91] lists five limitations:

- True data dependency
- Procedural dependency
- Resource conflicts
- Output dependency
- Antidependency

We examine the first three of these limitations in the remainder of this section. A discussion of the last two must await some of the developments in 1hc next section.

True Data Dependency

Consider the Co lowing selLtence:

```
ado.        r2      load                    rho col-. tents of r2
                      pus  the contents of xi
rcove    r3 w1    ;load regster .r3        the contonLs of fi
```

The second instruction can be fetched and dccoLlud but L.innot c until the first instruction eNccuM. The reason is that the second instruction needs [lath produced b the first instruction. situation is referred to as a true data dependency (also called flow dependency or write-read dependency).

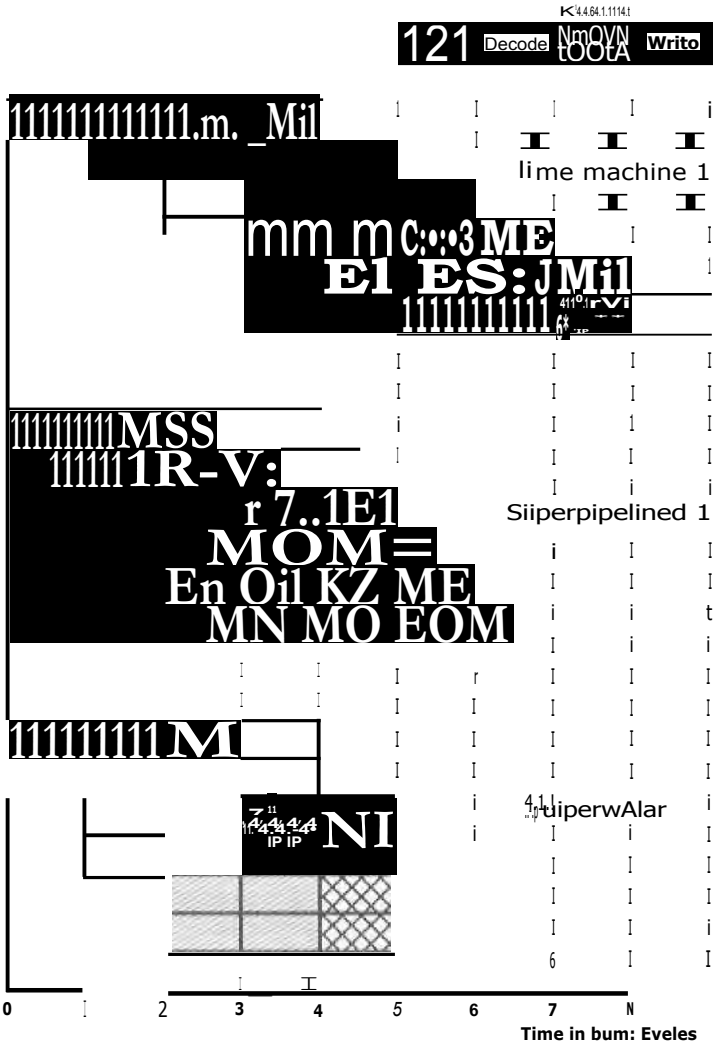
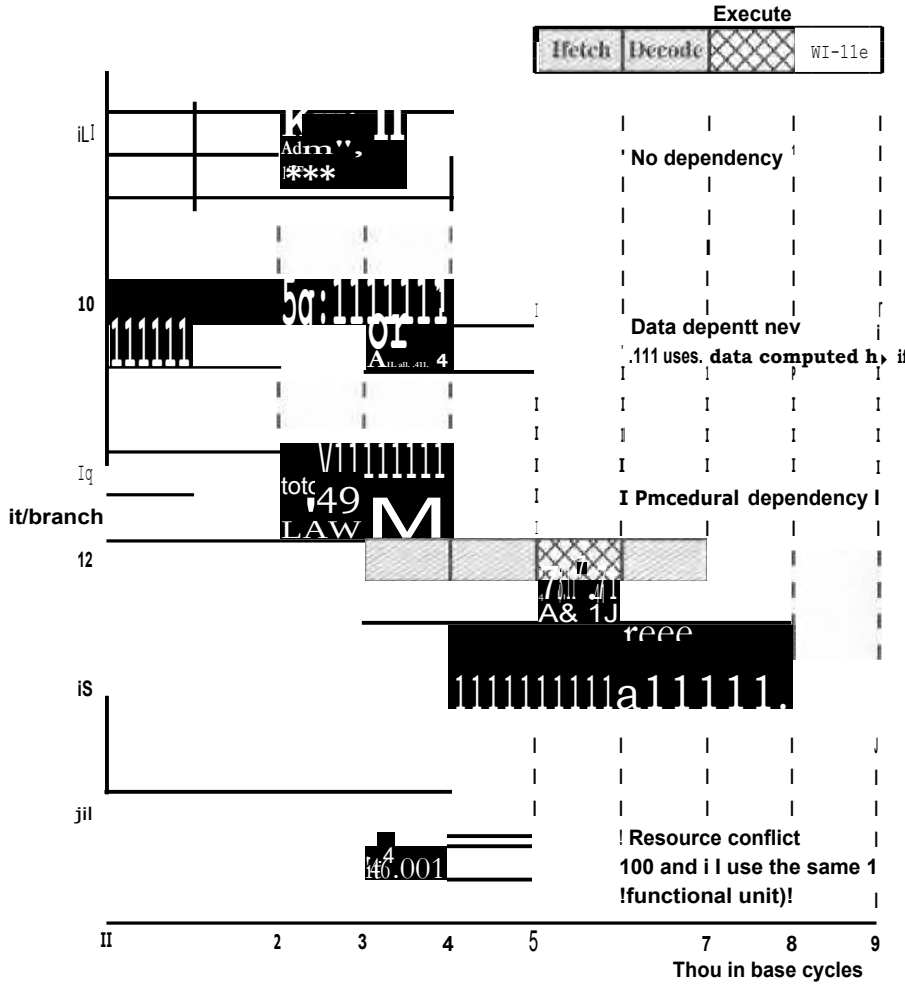


Figure 14.2 Coin NO son of Supelvalar and Superpipcline Approaches



Eiger' 14.3 Eacel of Depo ullnuich

Figure 14.3 iilustrates this dependency in a superscalar machine of degree 2. With no dependency, two instructions can be Iciched exectucd in parael, [f there is a da1.0 dependency between the first ond second instructions, i hen the sceond instruction is dehoied as many clock cycles as required to lenlore the dependency. In general, any instruction must be delved unlil all of its input values have been prodkcCil ;

1 SinL[rl12 Suihr pipeline, Ihe aforementioned sequence of instructions would C:itNt: no dchly. I lowc cr. consider 1h1 following, in which one of the loads is from incniory rusher than from a register:

```
Load    ri, cff     ;load reg3LeT ri          convent:, of
                                ir.ernory add.fes ef
MOve    r3, ri      'load regLste1 r3      tha          of rl
```

A typical RISC processor takes two or more cycles to perform a load from memory because of the delay of an off-chip memory or cache access. One way to compensate for this delay is for the compiler to reorder instructions so that one or more subsequent instructions that do not depend on the memory load can begin flowing through the pipeline_ This scheme is less effective in the case of a superscalar pipeline: The independent instructions executed during the load are likely to be executed on the first cycle of the load. leaving the processor with nothing to do until the load completes.

Procedural Dependencies

As was discussed in Chapter 12, the presence of branches in an instruction sequence complicates the pipeline operation. The instructions following a branch (taken or not taken) have a procedural dependency on the branch and cannot he executed until the branch is executed, Figure 14.3 illustrates the effect of a branch on a superscalar pipeline of degree 2.

As we have seen, this type of procedural dependency also affects a scalar pipeline. Again, the consequence for a superscalar pipeline is more severe. because a greater magnitude of opportunity is lost with each delay,

If variable-length instructions arc used, then another sort of procedural dependency arises. Because the.length of any particular instruction is not known. it must he at least partially decoded before the following instruction can he fetched. This prevents the simultaneous fetching required in a superscalar pipeline. This is one of the reasons that.superscalar techniques ;ire more readily applicable to a.R1SC or RISC-like architecture, with its fixed instruction length.

Resource Conflict

A resource conflict is a competition of two or more instructions for the same resource at the same lime. Examples of resources include memories. caches. buses, register-file ports. and functional units (e.g.. ALL adder).

In terms of the pipeline. a resource conflict exhibits similar behavior to a data dependency (Figure 14.3). There are some differences, however. For one thing, resource conflicts can he overcome by duplication of resources, whereas a true data dependency cannot be eliminated. Also. when an operation takes a long time to complete, resource conflicts can he minimized by pipelining the appropriate functional unit.

14.2 DESIGN ISSUES

Instruction-Level Parallelism and Machine Parallelism

POUPS9a] makes an important distinction between the two related concepts of instruction-level parallelism and machine parallelism. **Instruction-level parallelism** exists when instructions in a sequence are independent and thus can be executed in parallel by overlapping.

As an example of the concept of instruction-Level parallelism, consider **Elio** following two code fragments POUP89131: •

```
Load R_ <- R2                    Add R3
          R3 , 11"               Ada R4 (_ Fe, , R
7-1  _          K2                      [ Re).]      R.::
```

The [hree instruetion:i On Ihc kit are inKlependent, ;.]ncl in theory all llffee could be executed in In conirast the ttu-cc instructions on the right cannot be executed in parallel because the second instruction uses the result of the first, and the third instruction uses the resull of the second.

Instruction-level parallelism is cictermined by the frequency of true data dependencies and procedural dellendencies in the code. These factors, in turn, are dependent on the instruction set architecture and on the application. Instruction. Level parallelism is also determined by what POIJI-)89411 refers lo as operation latency! the lime until the result of an iiis[i lad ion is available for use as an operand in a subsequent instruction. The Latency determines how much of a delay a data or procedural dependency will cause.

Machine parialleli rn is a measure of l he ;Ibility ()I' the processor to take achan-[age 0f instruction-level paollelisrn. k determined by the number of instructions that can be fetched and executed at the same lime (the number of parallel pipelines) and by the speed and sophistication of the Mechani2!km2.., that the **promsKIT uses** to find inclepenclent insl ructions..

Both instruction-level and machine parallelism arc important factors in enhancing performance. A program may not have enough instruction-level parallelkm to take full advantage of machine parallelism, The use of a fixed-length instruc-[kin set irchiteeture, as in a HISC, cnhanees instruction-level parallelism. On the other hand, limited machine parallelism will Iimil performance no matter what the nature of the program.

Instruction Issue Policy

As was mentioned, nthchine **parallelism** is nut **simply** Ll matter of having multiple instances of each pipeline stage. The processor must also be able to identify instruction-level parallelism and orchestrate the fetching. decoding, and execution of instructions in parallel. [JOI-r4911 uses the term instruction issue to refer to the process of initiating instruction execution in the processor's functional units and the term instruction issue policy to refer to the protocol used to issue instructions.

in essence, the processor is trying lo look ahead of the current poinl of CNA> cation to locale instructions Carl be brought into thy. pipdhic and executed. Three types of orderings are important in this regard:

- The order in which instructions are fetched
- The Order in which instructions are cNecuted
- The, order in which instructions update the contents of register and memory locations

The more sophistic4111lhe proccs';or..1he less it is hound by a strict relationship he weep these orderings. To optimize utilii.alion of Hie various pipeline ele-

thc processor will need to alter one or more of these orderings with respect to the ordering to *he.* found in a strict sequential execution. The one constraint on the processor is that the result must be correct. Thus, the processor must accommodate the various dependencies and conflicts discussed earlier.

In general terms, we can group superscatar instruction issue policies into the following categories:

- In-order issue with in-order conviction
- In-order issue with out-of-order completion
- Out-of-order issue with out-of-order cornpleiion

In-Order Issue with In-Order Completion

The simplest **instruction** issue policy is to isuC instructions in the exact order 'hal would be achieved by sequential execution (in-order issue) and to write results in that srmic. order (in-order completion). Not even scalar pipelines follow such a simple-minded policy. However, it is uscful to consider this policy as a baseline for comparing more sophisticated aPProacllw-

F'igure 14,4a gives an example of this policy. We assume a superscalar pipeline capable of fetching and decoding two instructions at a time, having three separate functional units (e.g., two integer arithmetic and one floating-point arithmetic), and having two instances of the write-back **pipeline** stage. The example assumes the following constraints on a six-instruction code fragment:

- I] requires two cycles to execute.
- 13 and 14 conflict for the same functional unit,
- IS depcndN on the value produced by 14.
- 15 and 16 conflict for a functional unit,

Instructions are fetched LINO at LI lime aired passed lo the decode LIDA. Because instructions are fetched in pairs, the next two **instructions must wait until the pair** of decode pipeline slagcs has cleared. To guarantee in-order completion. when there is a conflict for a functional unit or when a functional unit requi rex. more lhan one cycle to generate a result, the issuing of instruction temporarily stalls.

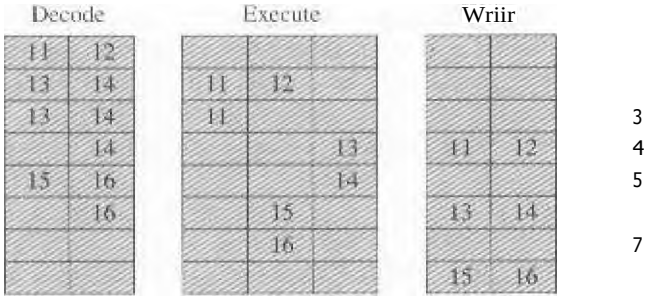
In **his** cx;imple, the elapsed time. from decoding the first instruction to writing the **1w4 results is** eight cycles.

In-Order Issue with Out-of-Order Completion

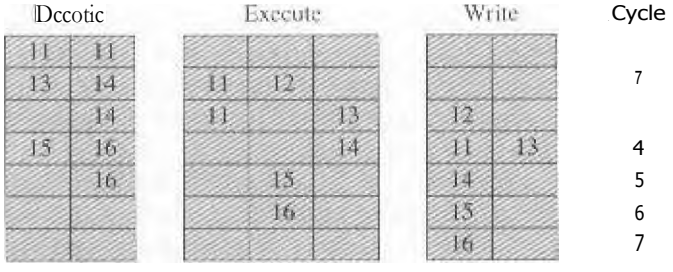
Out-of-order completion is used in scalar RISC processors to improve the performance of instructions that require multiple cycles. Figure 14.0 illustrates its LISc. cart a supersealar processor. Instruction 12 is allowed to run to completion prior to H. This allows I to he completed earlier, with the net result of a savings of one cycle.

With out-of-order completion, any number Or inslrucLion.s may be in the Q.:W-t:A.ll km stage at any one time, up to the maximum degree of machine parallelism across all functional units, Instruction issuing is stalled by a resource conflict, a data dependency, or a procedural dependency.

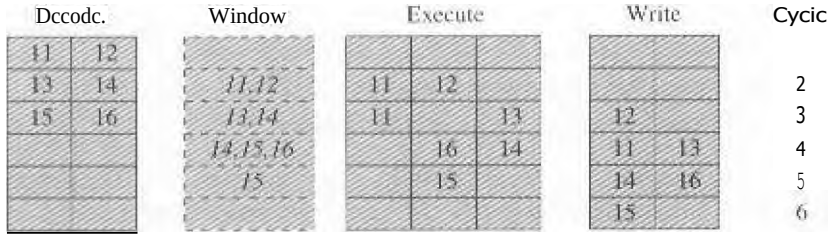
In addition to the aforenieniioned li mitations, a new dependency. which we referred to earlier as an **output dependency** (also called **write-write dependency),**



(a) 1n-orderi a.ntl in-order compicuon



(1-0 In- }Ilia issw and out-oi -order completion



(C.! OO-01⁴·³:1:(10: i ll I: pritk (an-o) prder eorapletion

Figure 14.4 Sziperscalar Instruction Issue and Completion Policics

4i nscs. The rohowing cock!. Ir.igrne.ni illustrates this dependency (op reprcsenLs nny opewion):

```
:1: R3          op R
L2 ; !"?.4      + 1
   : R3    R5     1
=4: F7 (— R3  Dp
```

Instruction 12 canna execute before instruction II, becausc ii needs the result n register R3 produced in 1.1 this is an exiimple or true data dependency. as described inSection 1 4.1. Similarly, 14 must frI. because it uses a rosult pro. di,2eLl Ii 13. .thoui the relationship between I] and 13? There is no daLs

dependency here. as we have defined it. however. if 13 executes to completion prior to I1, then the wrong value of the contents of 1-1=; will be fetched for the execution of 14. Consequently, 13 must complete after 11 to produce the correct output values. To ensure this, the issuing of the third instruction must be stalled iI its result might later he overwritten by an older instruction that takes longer to complete.

Out-of-order completion requires more complex instruction issue logic than in-order completion. In addition, it is more difficult to deal with instruction interrupts and exceptions. When an interrupt occurs, instruction execution at the current point is suspended. to he resumed later. The processor must assure that the resumption takes into account that, at the time of interruption, instructions ahead of the instruction that caused the interrupt may already have completed.

Out-of-Order issue with Out-of-Order Completion

With in-order issue, the processor will only decode instructions up to the point of a dependency or conflict. No additional instructions are decoded until the conflict is resolved. As a result, the processor cannot look ahead of the point of conflict to subsequent instructions that may he independent of those already in the pipeline and that may he usefully introduced into the pipeline.

o allow out-of-order issue, it is necessary to decouple the decode and execute stages of the pipeline,. 'This is done with a buffer referred to as an instruction window. With this organization. after a processor has finished decoding an instruction. it is placed in the instruction window. As long as this buffer is not full, i he processor can continue to fetch and decode new instructions. When a functional unit becomes available in the execute stage, an instruction from the instruction window may he. issued to the execute stage. Any instruction may he issued. provided that (1) it needs the particular functional unit that is available and (2) no conflicts or dependencies block this instruction.

The result of this organization is that the processor has a lookahead capability, allowing it to identify independent instructions that can be brought into the execute stage. Instructions are issued from the instruction window with little regard for their original program order. As before, the only constraint is that the program execution behaves correctly.

Figures I 4.4e illustrates this policy. On each cycle, two instructions arc fetched into the decode stage_ On each cycle, subject to the constraint of the 'buffer size, two instructions move from the, decode stage to the instruction window. In this example. it is possible to issue instruction 16 ahead of 15 (recall that 15 depends on 14, but 16 does not). Thus. one cycle is saved in both the execute and write-hack stages. and the end-to-end savings, compared with Figure 14,4b, is one cycle.

The instruction window is depicted in Figure 14.4c to illustrate its role. 1 lowever, this window is not an additional pipeline stage. An instruction being in the window simply implies that the. processor has sufficient information about that instruction to decide when it can be issued.

The out-of-order issue, out-of-order completion policy is subject to the same constraints described earlier. An instruction cannot be issued if it violates a dependency or conflict. The difference is that more instructions are available for issuirw, reducing the probability that a pipeline stage will have to stall. In addition, a new dependency, which we referred to earlier as an autidependency (also called read-

write dependency). arises. The code fragment considered earlier illustrates this dependency!

```
IL : R3      R3  op  R5
12 ;         R5  -  1
 3 R3 (— R5 — 1
14: R^       R3  op  R4
```

Instruction 12. •annot complete execution before instruction [2 begins execution and has fetched its operands. This is so because 13 updaces register **R. v.rhich** is a source operand for 12. The term *toirlidependicncy* is used because the colp.traint is similar to that of a true data dependency, but reversed: Instead of the firs! ar t ILLC• (ion producing a value that the second instruction uses, the second instruction destroys a value that the first instruction uses.

Register Renaming

When oui-ol-order instruction iSSLIirig and/or out-of-order instruct ion completion are allowed. we have seen that this gives rise to the possibility of output dependencies and antidependencies. These dependencies differ from true data dependencies and resource conflicts. which reflect the flow of data through a program and the sequence of execution. Output dependencies and antidependcneics. on the other hand, arise because the values in registers may no longer reflect the sequence of values dictated by the program flow.

When instructions 4Lre issued in sequence and complete in sequence, it k possible to specify the contents of each register at each point in the execution. When out-of-order techniques are used. the values in regisi ers cannot be fully known at each point in time just from a consi4lerai ion of thL' sequence of instructions dictated by lite program. In effect, value!, al e in conflict for the use of registers. and the processor must resolve th(we conflicts by occasionally stalling a pipeline stne.

Antidependencies and output **dopenciencies tiro** hOLII examples of storage con. tlicts. Multiple:instructions are competing for t he use of the same register locations. generatin;, pipeline constraints that retard performance. The problem is made more acute when iegistet optimization techniques arc used (as discussed in Chapter 13). because these compiler techniques attempt to maximize the use of registers, hence maximizing 1 he number of storage conflicts.

One method for coping with these types of storage conflicts is based on a traditional resource-conflict solution: duplica ion of resources. **In** this context, the technique is referred to as **register renaming,** In essence, registers are allocated dynamically by the processor hardware. and they are associated wilh the values needed h instructions at various points in time. When a new register value is created (i.e., when an instruction executes that has a register **as** a destination operand). a new register is a ilocatcd for that value. Subsequent instructions that access that value as source operand in that register must go through a renaming process: The register references in those instructions must be revised to refer to the register conlitining the needed value_ Thus, the **minic** original register reference in several different instructions may refer to different actual registers. if different values are intended.

Let us consider how register renaming could be used on the code fragment we have been examining:

```
1: R3 L, <— R3, Op a5,
: R4,, <— R3, + 1
   R3          + 1
14: R7, <— H3,.  op  R4,,
```

The register reference without the subscript refers to the logical register reference found in the instruction. The register reference with the subscript refers 10 a hardware register allocated to hold a new value.. TiLVIten a new allocation is made for a particular logical register, subsequent instruction references 10 that logical register as a source operand are made to refer to the most recen I ly allocated hardware register (reccnt in terms of the program sequence of instructions).

Iii this example, the creation of register R3,, in instruction I3 avoids the antidependency on the second instruction and the outpul dependency on the first instruclion, and it does not interfere with the corrcet value bcing accessed by 14. The result is ihat 13 can be issued immediately; without renaming. 13 cannot be issued until the first instruction is complete and the second instruction is issued.

Machine Parallelism

In the preceding, we have looked a1 three hardware techniques that can be used in a superKalar processor to enhance performance! duplicikn of resources, oul-of-order issue, and renaming. One study that illuminates the relationship among these. techniques was reported in j511 [1 89]. The study made use of a simulation that modeled a machine with the characteristics of the MIPS R2000, augmented with various superscalar features. A number of different program sequences were simulaicd.

Figure 14.5 shows the results. In each of the graphs, the vertical axis corresponds to the mean speedup of I he superscalar machine over the. scalar machine. The horizontal axis shows the rcsulis *for* tour alternative processor organizations. The base. machine does not duplicate any of the functional units, but it can issue instructions out ()I' order. The second configuration duplicates the toad/stone functional unit that accesses **a** data cache. The third configuration duplicates the ALU. .end the fourth configuration duplicates both load store and AI, . in each graph, results arc shown for instruction window sizes of 8, 16, and 32 instructions, which dictates the amount of to okahead the processor can do. The difference between the two graphs is that, in the second, rep isl et renaming is allowed, Thia is equivalen1 to saying Ihal the firm graph reflects a machine that k limited by all dependencies, wheteaN I he second graph corresponds to a machine that is limited only by t **rue** dependencies.

The two graphs. combined, yield some importani conclusions. The first is that it is probably not worlhwhile to add functional units without register renaming. There is some slight improvement in performance. but at the cost of increased hardware complexity. With register renaming, which eliminates antidependencies and outpul dependencies, noticeable gains are achieved by adding more funciional units. Note, however, Thal there is a significant difference in the amount of gain achievable between using an instruction window of 8 versus a larger instruction

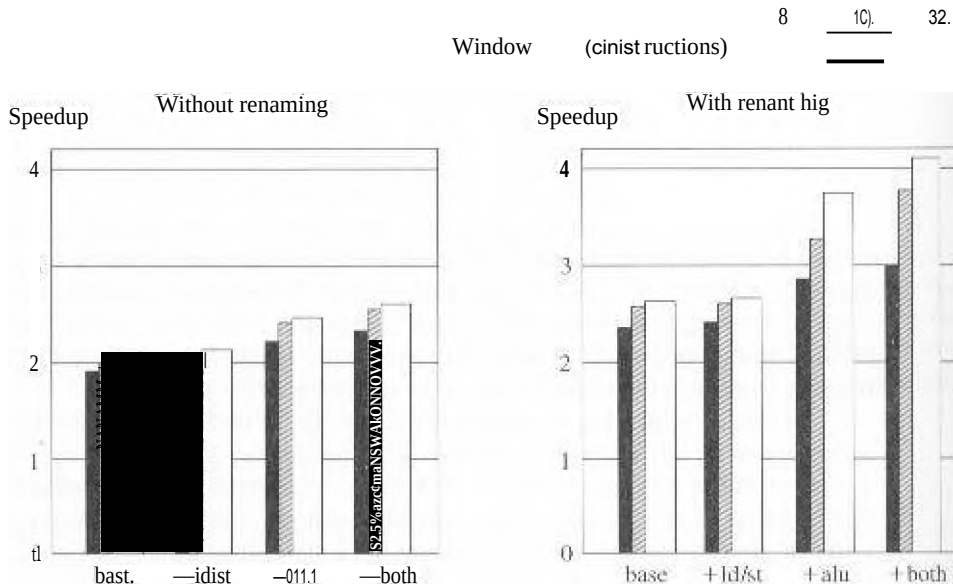


Figure 14.5 Speedups or Various Machine Organizations, without procodund Dependencies

window. This indicares aim if the instruction window is too small, data dependencies will proicrul erreelivc utiliaition of the extra functional units the processor must he able Lo look quite far ahead to find independent instructions to utilize IhL hardware more fully.

Branch Prediction

Any high-performance pi pelined rnachine must address the issue of dealing with branches. For example. the Intel 80486 addressed the problem by ft:II:Fling both the next sequential instruction after a branch and speculatively fetching the branch target instruction_ However, because. there are two pipeline stages between prefetch and execution, this strategy incurs a two-cycle delay when the branch gets taken.

With the advent of RISC machines, Ihe delayed branch sir:mew,/ wras explored, I his allows laic:. processor Lo calculate the result of conditional branch instructions before any unusable instructions have been prefetcbcd. With this method, the processor always executes the single ins1rurt inn that immediately follows the branch. 'fhis keeps the pipeline full while the processor fetches.a new instruction stream.

With the development of supersealar machines, the delayed branch strategy has less appeal. 'Hie reason is that rriultirtic instructions need to execute in the delay slot, raising several problems relating to instruction dependencies. **Thus, su** pc:m:4u machines have returned to pre-RISC techniques of branch predie1ion, Some, like the rowerl'C NM. use a simple static branch prediction technique. More sophisti-

rated processors. such as the PowerPC 621) and the Pentium 4, use dynamic.branch prediction based on branch history analysis.

Sup erscalar Execution

We are now in a position to provide 4Lil overview of .sup mcalar execution of programs; this is illustrated in Figure 14.6. The program to be executed consists of a lin• sequence of instructions. This is the static program as written by the programmer or generatud by the conviler. The instruction fetch process. which includes branch prediction. is used to form a dynalirliC. hil·Cani Of instructions, This stream is examined for dependencies, and the processor nix' remove artificial dependencies. The processor i hun dispatches the instructions into a window of execution. In this window_ instructions no longer form .';equenlial stream but are structured according to their true data dependencies. The prcwessor perrorms the e xeen i on stta ge of each instruction in an order determined by the true data dependencies and hRrdware resource *avai* Finally, instructions are conceptually put back into sequential order and their results arc recorded.

The final step mentioned in the preceding paragraph iw relurcil to as *coinnar*• ti.v, or *refiring.* the instruction. This step is needed for the following reason. Because of the use or parallei, mulIipie pipelines. instructions may complete in an order different from that shown in the statie program. Further, ihe. use. of branch prediction and speculative execution means that some instructions rnav complete. execution and then musl be abandoned because the branch they represent is J101 taken, Therefore, permanent sloragc mid program-vkibie rK•gi4ers cannot be updVed immediately when instructions complete execution. Results must be held in sonic Sod Of Iemporory storage that is usable by dependent instructions and then made pertnaricht wheel ii is determined Ihat the sequential model would have executed the instruction.

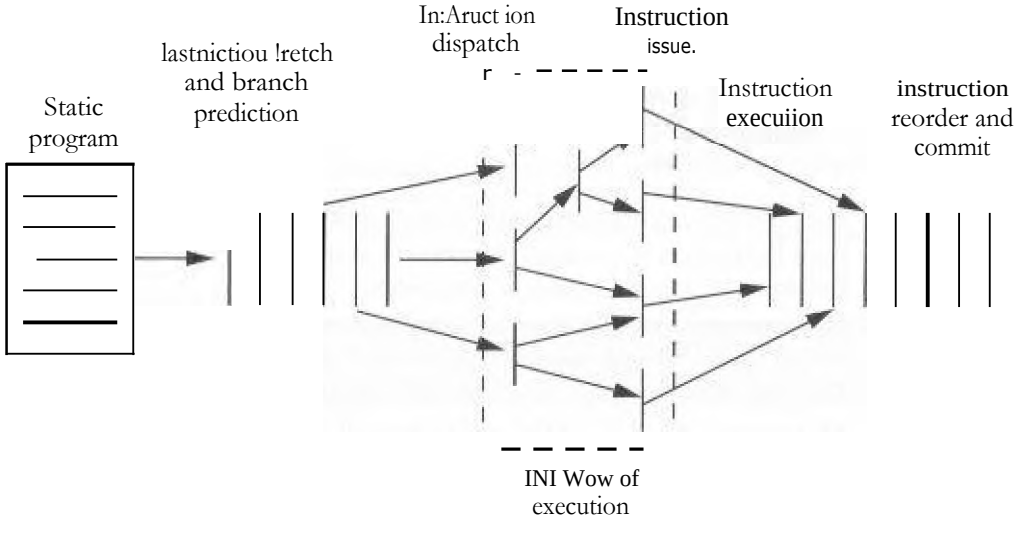


Figure 144 Conoepnial Derrietion cif Superscalar Processing ESIvIITY5

Superscahr Implementati 011

Based on our discussion so lar, we can make some general comments about the proecssor hardware required for the superscalar approach. [SMIT951 lists the following key elements:

- Instruction fetch strategies that simultaneously fetch multiple instructions, often by predicting the outcomes of. and fetching beyond, conditional branch instructions. These functions require Lhc use of multiple pipeline fetch and decode :stages- and branch prediction logic.
- Logic for determining I rue dependencies involving register values, rind mechanisms for communicating these values to where they axe needed dur. ing execution.
- Mechanisms for initiating, or issuing, multiple instructions in parallel
- Resources for parAlel execution of multiple instructions, including muniplc pipe. li ned functional units and memory hierarchies capable of simultaneously servicing multiple memory references,
- Mechanisms for committing the process state in correct order,

r.

14..3 PE,INITItai 4

Although the concept of superscalar design is generally associated with the RISC architecture, the same superscalar principles can be applied to a C.T.SC machine. Perhaps the most. notable emimpic or this is the Pentium. The evolution cif supcirRealat concepts in the Intel line is interesting to note. The 80486 was a siva ightforward traditional C'ISC' machine, with no superscalar elements. '1 he. original Pentium had a modest superscalar component, cunsisling of the use of two separate integer execution uniis. File Pentium Pro introduced a full-blown superscalar design- Suh-&equent Pentium models have refined and enhanced the superscalar design.

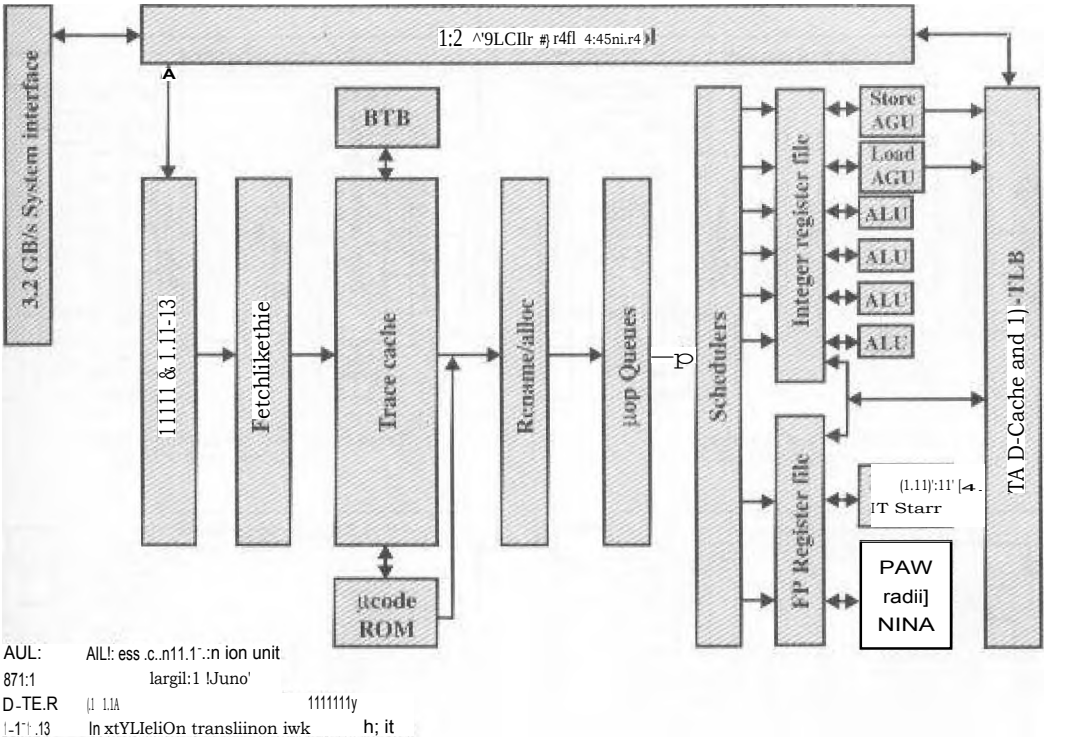
A general block diagram of the Pentium 4 wa. ', shown in Figure 4.13. Figure 14.7, based on one in [C At depicts thc mime structure in a way more suitable for the pipeline discussion in this section. The operation of the Pentium 4 can be surnmari2ed as follows:

L proce,ssc Yr I u tales instructions from memory in the order of are stalie program,

/ Each instruction is translated into one or more fixed-length RISC instructions, known as micro-operations, or micro-ups.

3. The processor executes the micro-ops on a superscalar pipeline organization, so that the micro-ops may be execuled out or order.
4. The processor commits the results of each micro:op execution in the processors register set in the order of the original program flow.

In cited, the Pentium 4 architecture consists of an outer cis(' shell with an inner RISC core. The inner **RISC** micro-ops pass through a pipeline with at least 20 stages (Figure 14.8): in some cases, Lha micro-op requires multiple execution stages,



figum 14.7 Protium **4** Nock Diagrurri

resulting in an even longer pipeline, This contrasts with the five-stage pipeline (Figure [2.1(x) uw..t.1 on the Intel N.86 processors and on the Pentium.

We now truce k operation of the Penli urn 4 pipeline. using Figure 14. 4 litP illustrate its operation.

Front End

Generation of Micro-Ops

Thu Punli urn 4 organization include an in-order front end (Figurk.l. I4,{la) that can be considered outside the scope of thc pipeline depicted in Figure I4.K. Phis front end feeds into an LE instruction cache, culled the trace cache, which is where

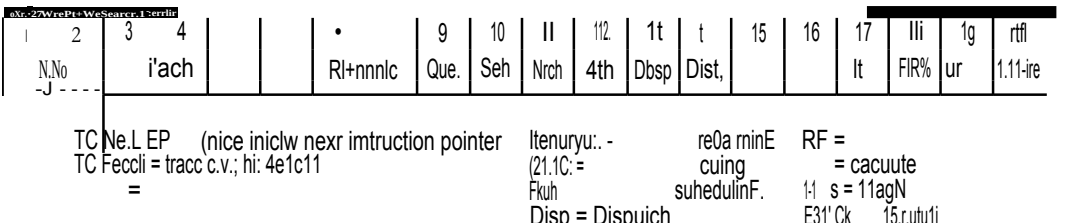


Figure 14,8 Pcilliurn 4 PiNiii-R!

(b) Trace cache next instruction pointer

(d) Brie a

(e) Allocate; Register renaming

Figure 14.9 Pentium Pipeline Operation

BTB & 1-Tt.n
Fekkidisetude
ROM
Trace cache
BTB
Rename/allot
Schedulers
EMT
MEM
Integer RI:
Fop
Fms
L1 11-Cache and I)-TLB
L2 Cache and control
13 111 & I-11.B
Fetch/derrick..
'!'race cache
R e ua wrialloc
pop Queues
ScheduJers
FP
Integer RF
Li II-Cache and li•TLB
BTB & 1-11.1s
Fetch/decode
Trace cache
Rename/alloc
pop Queues
Schedulers
ETRE
FP RF
Integer RN
I. 1 D-Cache and 6-TI13
BTB &
Trace cache
Reinionehblkw
pop Queues
Schedulers
44444
FP RE
InIeger RF
L1 li•Cache and LI- I 1.11
BTB & I-TLR
ietch/deeede
Trace cache
Renarne/alloc
pup Queues
Schedulers
FP RE
Integer RF
vokat
FR1
L 1. D-Cache and D-T1.11
111B & 1·no
Felt:bidetode
Trace cache
Rename/aline
imp Queues
Schedule
FP RF
Integer RF
LI D-Cache and D-11.14

the pipeline proper begins. Usually, the processor operates from the ince cache; when a Irace cache miss occurs, the in-order fron1 end feeds new instructions into the trace cache.

With the aid of the branch target buffer and the instruction lookasidc buffer (51˙B & I-TLB), the fetchidecode unit fetches Pentium 4 machine instvuctions from the 1.2 cache 64 bylcs at a time. As a default, instructions are fetched sequentially, so that each L.2 cache line fetch includes the next instruction to be fetched. Branch prediction via the BTB 3r 1-TLS unit may alter this sequential fetch operation. The ITLB translates the linear instruction pointer address given it into physical addresses needed Lo access the L2 cache. Static branch prediction in the front-end BTB is used to determine which instructions to fetch next.

, Once instructions are fetehed, the fetch/decode unit scans die bytes to determine instruelion boundaries; this is a necessary operation because of the variable 1CJI ath of Pentium instructions. The decoder translates each machine instruction into from one to four micro-ops, each of which is a 118-bit RISC instruction. Note for comparison 1hat most pure RISC machines have an instruction length of just 32 bits. The longer micro-op length is required to accommodate the more complex Pentium operations. Nevertheless, the micro-ops are easier to manage than the original instructions from which they derive.

The **generated micro-ups** are stored in the trace cache,

Trace Cache Next Instruction Pointer

The first two pipeline stages (Figure 14.9b) deal with the selection of instructions in the trace. cache and involve a separate branch prediction mechanism from that described in the previous section. The Pentium 4 uses a dynamic branch prediction strategy based on the history of recent executions of branch instructions. A branch target buffer (EITB) is maintained that caches information about recently encountered branch instructions. Whenever a branch instruction is encourocred in the instruction stream. the BTU is checked. If an entry already exists in the BTB, then the instruction unit is guided by the history information for that entry in deter ' mining whether to predict that the branch is taken. If a branch is predicted, then the branch destination address associated with this entry is used (0E prefetching the branch target insiruction.

Once the iitz,tritction is executed, the history portion of the appropriate enlry is updated to reflect the result of the branch instruction. If this instruction is not represented in the Eira then the address of this instruction is loaded into an entry in the 1-11I-3; if necessary, an older enlry is deleted.

The description of the preceding two paragraphs fits, in general terms. the branch prediction strate gy used on the original Pentium model. as well as the later Pentium models, including Pentium 4. However, in the ease of the Pentium, a relativehy. simple 2-bit history scheme is used. The later Pentium models have much longer pipelines (20 staes for the Pentium 4 compared with 5 stages for the Pentium) and therefore the penalty for misprediction is greater, Accordingly, the later Pentium models use a more elaborate branch prediction scheme with more history bits to reduce.the misprediction rate.

The Pentium 4 BTB is organized as a four-way set-associative cache with 512. lines, Each entry uses the address of the branch as a tag. The entry also includes the

branch destination address for the last time this branch was taken and a 4-bit history field. Thus use Of four history hits conlrasIs with the 2 bits used in the original Pentium and used in most superscalar processors, With 4 bits. the Pentium 4 mechanism can take into account a longer history in predicting branches. The algorithm that is used is referred to as Yeti's algorithm 1YEH91 J. The developers of this algorithm have demonstrated that it provides a significant reduction in mispi ediction compared to algorithms that use only 2 bits of history [EVER98].

Conditional branches that do not have a history in the 1#TR are predicted using a static prediction algorithm, according to the following rules:

- For branch addresses that are not IP.relative, predict taken if the branch is a return and not taken otherwise.
- For IP-relative backward conditional branches, predict taken. This rule reflects **the typical behavior** of loops,
- For IP-relative forward conditional branches, predict not taken.

Trace Cache Fetch

The trace cache (Figure 14.9c) takes the already-decoded micro-ops from the instruction decoder **and** assembles them in to program-ordered sequences of micro-ops called traces. Micro-ops are fetched sequentially from the trace cache, subject to the branch prediction logic.

A few instructions require more than four micro-ups. instructions arc transferred to microcode ROM, which contains the series ()I' microlops (five or more) associated with a complex machine instruction. For example, a string instruction may translate into a very large (even hundreds), repetitive sequence of micro-ops. Thus, the microcode ROM is a microprogrammed control unit in the sense discussed in Part Four, After the microcode ROM finishes sequencing micro-ups for the current Pentium instruction, fetching resumes from the trace cache.

Drive

The fifth stage (Figure 14.9d) of the Pentium 4 pipeline delivers decoded instructions from the trace cache to the rename/allocator module,

Out-of-Order Execution Logic.

This part of the processor reorders micro-ops to allow them to execute as quickly as their input operands are ready.

Allocate

The allocate stage (Figure 14.9e) allocates resources required for execution. It performs the following functions:

- If a needed resource, such as a register, is unavailable for one of the three micro-ops arriving at the allocator during a clock cycle, the allocator stalls the pipeline.
- The allocator allocates a reorder buffer (R014) entry, which tracks the completion status of one of the 12h micro-ups that could be in process at any time.

- The allocator allocates one of the 128 integer or floating-point register entries for the result data value of the micro-Op. and pmsibly a load or store. buffer used to track one of the 4 loads or 24 stores in the machine pipeline.
- The allocator allocates an entry in one of the two micro-op (-11.1QL,LCS in front of the instruction schedulers.

'The ROB is a circular buffer that *c210 hold up* to 126 micro-ops and also contains the 128 liardiA' are registers. Each buffer entry consists of the following fields;

- **State: indicates** whether this micro-op is scheduled for C.ncx2Lition. has been dispatched For execution, or has completed execution and is ready for retirement.
- **Memory Address:** The address of the Pentium instruction that generated the micro-op.
- **Miero.op:** The actual operation.
- **Alia', Register:** If the rmcro-op refe.renees one. of the 16 a rchitecturat registers, this entry redirects that reference to *one* of the 128 hardware registers.

Micro-taps enter the ROB in order. Micro-ups are then dispatched from the ROB to the Dispatch/Execute unit out of order. The criterion for dispatch is that the approphate execution unit and all dflth items required for this micro-op arc available. Finally, micro-n1t ti e retired from the ROB in order. To accomplish in-order retirement. micro-ops are retired oldest first after each micro-op has been designated as ready for retirement,

Register Renaming

Mc: rename stage (FigurC• 4.9c) rentaps. references L0 the 16 architectural registers (8 floatinst-poini register!, plus FAX, 12BX, ECX, EDX, ESL EDI, EBP. and ESP) into a set of 128 physical registers. The stage removes false dependeneie. caused by a limited number of architectural registers while preserving the true data dependencies (reads after wriles).

Micro-op Queuing

After resource allocation and register renaming, micro-cps are placed in one of two micro-op queues (Figure i4.90, where they are held until room in the schedulers. One of the two queues is for memory operations (loac.N and stores) and the other for micro-ups clo noi involve memory references. Each queue obeys a PIPO (first-in-first-out) ilkcipline, but no order is maintained between queues. That is, a micro-op may be [cad out of one queue out of order with respeci lo micro-cps in the other queue. This provides greaher flexibility to the schedulers,

-lip Scheduling and Dispatching

The schedulers (Figure /4.Ug) are responsible For retrieving micro-ops from he niicro-op queues and dispatching these *for* execution. Each scheduler looks for micro-ops in whose status indicates that the micro-op has all of its operands. If the execution unit needed by that micro-op is nvailable, then the scheduler fetches he micro-op and dispatches il to the appropriate execution unit (Figure 14.9h1. Up to six micro-ops can be dispatched in one cycle. If more than one micro-op is

available for a given execution unit, I hic n the scheduler dispatches them in sequence from the q1euu.'1'his is a sort or **F110** disciptine that favors in-order execution, but by this time the instruction stream has been so rearranged by dependencies and branches that it is substantially out of order.

l[2]our pork attach the schedulers in the execution units. Port 0 is used for both integer and floating-point instructions, with the exception of simple integer opera• Lions and the handling of branch mispredietions. which are allocated to Port 1. In addition. NIMX execution units are allocated between these two ports. The renEllin - inv ports are for memory loads and stores.

Integer and Floating - Point Execution Units

The integer and floating-point register files are the source for pending operaliom by the execution units (Figure Tile execution units retrieve values from the register files as well as from the f.I dam cnche (Figure 14.9j). A separate pipeline stAge i used to compute flags (c.g., zero, negative); these are typically the input to a branch instruction.

A subsequent pipeline. stage performs branch checking (Pigurc 14.9k). This function compires [he ncrual br.iinch result with the prediction. If a branch prediction turns out to have been wrong, then there are micro-operations in various stages of processing that must be removed from the pipeline, The proper Inancli destina-Hon is I hen provided lo the Branch Predictor during a drive sta2e (Figure 14.91), which req.:irk the whole pipeline from the new target address.

14.4 POWERPC

he].'awed'(: architecture is a direct des.cvridani car the 113M 603. the RT PC.". and the 1 S/6001..t, the last also referred to as xih imptementat ion of the POW FJ-3. architecture. Alt of these are RISC' machines, but the first in the series to exhibit superscalar features was the RSI6000. The first implementation of the PowerPC architecture, the 61[1, has a super.sc.Hlar design quite Siittflzu to that of the RS/6000. Subsequent PowerPC models carry the superscalar concept further. In this section_ we focus on the 601, which provides a good example of a RISC-based superscalar design. At the end *or* the section, we briefly consider thc 620.

PowerPC 601

Figure 14.111 is **a general Vie.W of the flit** organization. As with other superscatar machines, the fill is broken up irno independent functional units to enhance opportunities for overlapped execution. In particular, the core of the 601 consists of three independent pipelined execution unils: integer. floating-point, and branch processing. Together, these uniLS Cain execute three instructions tit a time, yietding a superscalar design of degree 3.

Figure 14.1.1. shows a]odcal view of the 601 archii L N tire, emplmsizing the flow of instructions hOwe..eri func1iumil **The icteh** unit ean wretch up to eighi ilistructions ,tit a time from the cache. The. cache unit supports a combined insiriAl ioni

data cache and ii.4.responsihIc for feeding instructions to the other units and data to. the regisiers. Cache arbitration logic sends the address of Ihe highest-priority access to the cache.

Dispatch Unit

The dispatch unit takes instructions frorn the cache and loads them into the dispatch queue, which can hold Light instructions at a time, It processes this stream of instruetion.s to Iced a steady flow of instructions to the branch processing. inieger. and floating-point units. The upper half 0f the queue simply acts as a burlier to hold instructions until they move into the lower half. Its purpose is to elmLIIL that the dispatch unit is not delayed waiting for instructions from **the** cache. In the lower half_ instructions are. dispatched according to the following scheme:

- **Branch procesxing unit: Handles all** branch instructions_ The lowest such in!,truetion in the bottom half of the dispatch EARICite is **issued** to the branch processing unit **if that unit** can accept
- **FluatiNNFIFint Unit:** handles **all** floating-point ins! ruei ions, **The** lowest such instruction in the bottom half of the dispatch queue is issued to the floating. point unit if the instruction pipeline *in* that unit is not full.
- **Integer unit:** Handles integer instructions, load/stores between th 2 registcr files and the cache, and integer compare instructions. An integer instruction is only i2-:.sued after it **ha.!, tilLered** to the bottom of the dispatch queue.

Allowing branch and floating-point insi ructions to he issued out of order from the dispatch queue helps keep he instruction pipelines in the branch processing and li mning-point units full, and it moves instructions through the dispatch queue **as** rapidly as possible.

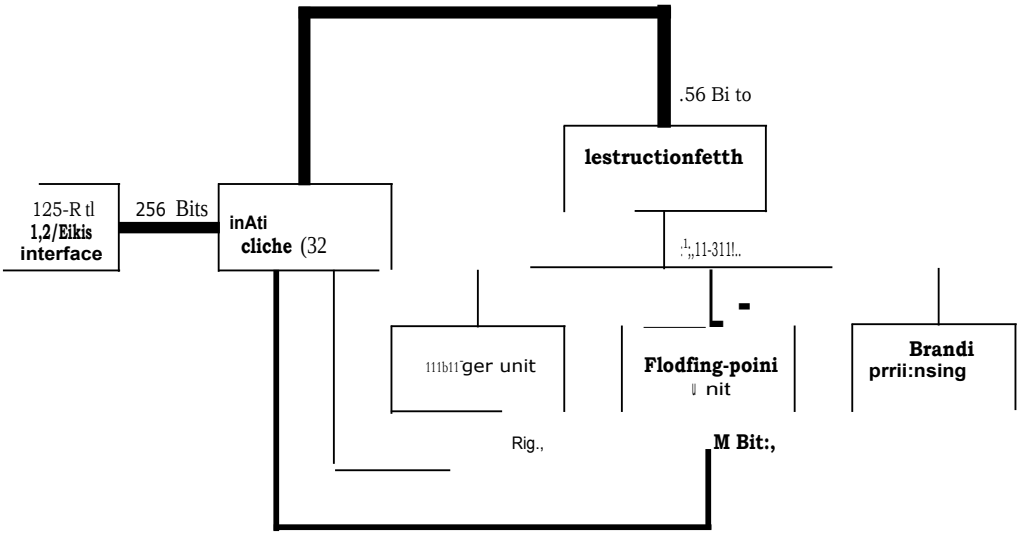
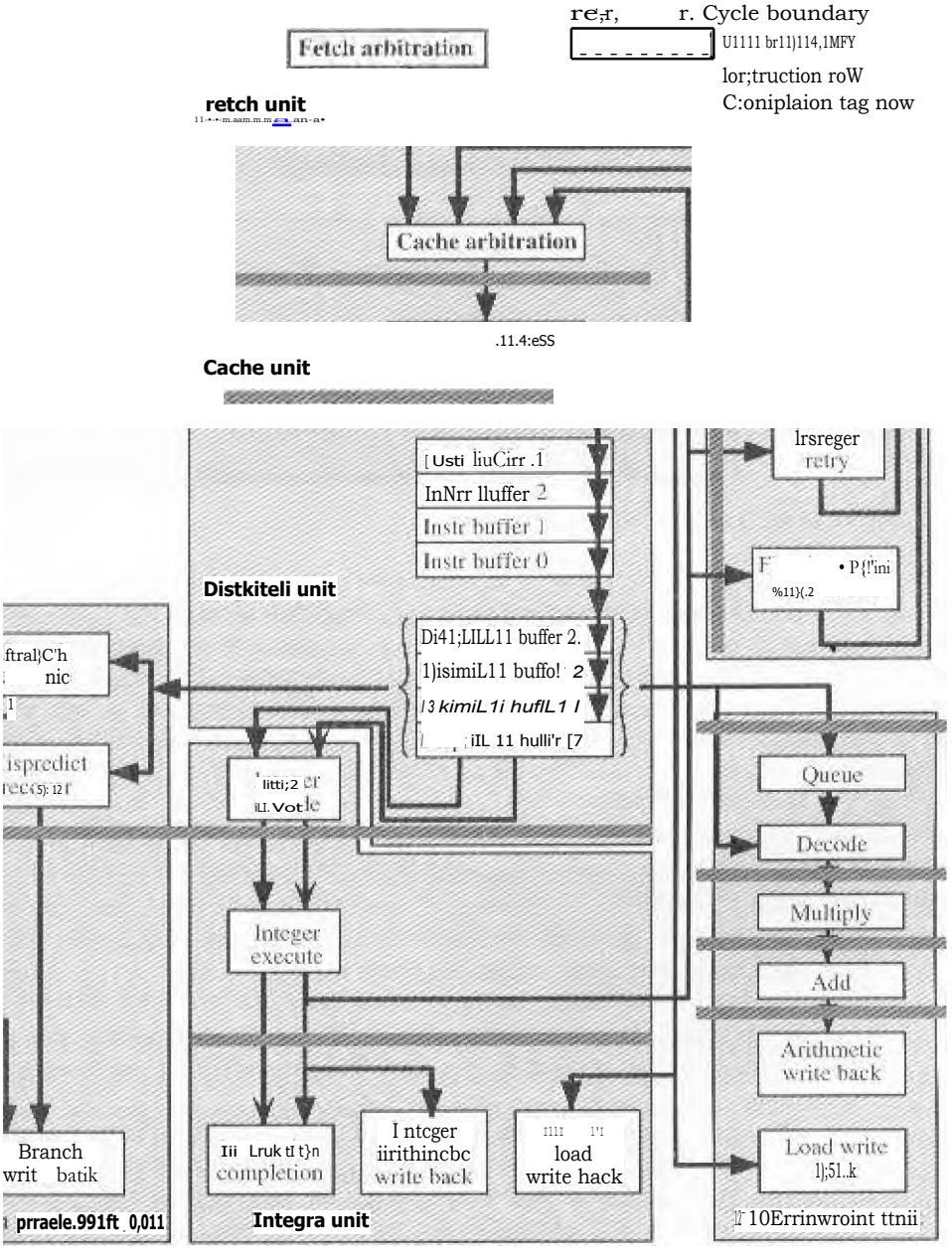


Figure 14.10 PowerPC #301 Block Diagram



Flom 14.11 Powci PC h0.1 SuucLurc 11)O(1-T94f

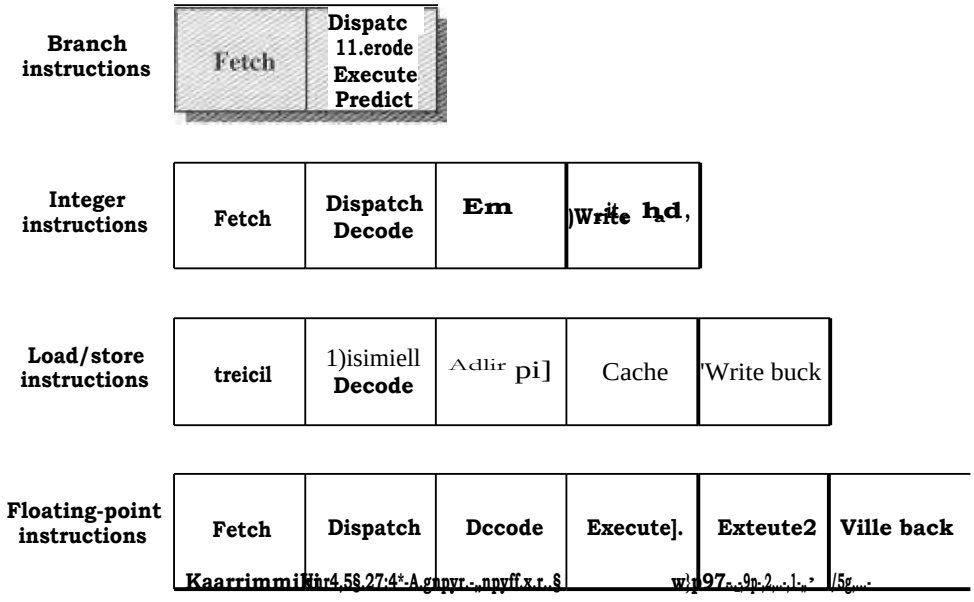


Figure **14.12** PowerPC 601 Pipclinc.

The dispatch unit also contains logic that enables it to calculate the prefeta addres.s. 11 contin ues fetching instructions sequentially until-a branch inslruclinti moves into the lower half of the dispaieh queue. When the branch processing mit processes an **instruction, it may** update the prefetch address so that succeeding instructions are fetched from the new address and entered into the dispatch queLm..

Instruction Fipeiin es

Eigurc 14.12 illustrates the instruction pipelines **for** the various **units.** There a common fetch cvet r43.1" I in2StrUci i{}11S; this occurs haore an ins1ruction k patched to a particular unit The second cycle begins with the dispatch **of an instruction** to a particular unit. This overlaps with other activities within the **unit.** During each clock cycle, the dispatch unit considers Ihe bollorn four en tries; of the Lion queue and dispatches up to three ihsLructions.

For branch instructions, the second cycle involves decoding and executing ins l ructions as well as predicting branches. The. last activity is discussed in the nest subsection.

The integer unit deals with instructions that cause a loadistore operation with memory (including floating-point load/store), a register—register move, or an **ALU** operation. In the caw (..11 a load/store, there is an address generation cycle ruikyviud sending the resulting address to the cache and, if neeesarli..., a write-back cycle. For other instructions. the cache is not involved and there is an execute cycle followed by a write back to register.

Floating-point inslrucl ions. lrhowa.mitt r pipelinc., but there ;Ire two executc: cycles. reflecting the complexity of floating-point operfitions.

SeVLtrul additional points are worth noting. The condition register contains eight independent 4-bit condition code fields. This allows multiple condition codes to be retained, which reduces the intedoek or dcpcndeney between instructions, For evimple, the e..f.m¯npilcr can transform thc sequence

```
cra¨r.p2x.E

crr.par e
```

to the sequence

```
c cAr.pa¨ze
c. mpare

bra .cis
brar...T.h
```

RCC4iLLM,2 C'..Lch functional unit can send its condition codes to different fields in the condition register, interlocks between instructions caused by sharing of condition codes can be avoided,

The prexonce of ihc. Save and Resume registers (SRRs) in the branch processor allows it to handie simple interrupts and software interrupts wii houi involving logic in the other functional units. Thus. simple operaling tlyslcm7.;crvices can he performed rapid]v without complicalcd stab: Iminipulation or synchronization between the functional units.

Because the 601 can issue branch and floating-poinl instrudions out of order, controls are needed to ensure proper execution. When Lk dependent v exists (i.e._ when an instruction needs an operand that has yet to be computed by a previous instruction), thc pipeline in the corresponding unit stalls.

Branch Processing

The key to the high performance of a RISC or superscalar machine is its ability to opiirni i.e. the nse of the pipeline. Typically , the most critical element in the design is how branches are handled. In the PowerPC, branch processing is the responsibility of the branch unit, The unit is designed so that in many cDsus. branches have no *effect* on the pac.12 of execution in the other units; these type of branches are referred to as zero-cycle branches. To achieve zero-cycle branching, the following strategies are employed:

1. Logic is provided to scan through the dispatch buffer for branches. Branch 1arget4iddre;ises ;ire gerwr2Jtcd when a branch first appears in the lower half of the queue and no prior hrailehes are pending execution.

2. An attempt is made to determine the outcome.of conditional branches. If t4 condition code has been set sufficient[!,/far in advance, this can he determined In any case, as soon as a branch instruction is encountered, logic determines if the branch
   a. Will be la kcn this is the case for unconditional branches and for conditional branches whose condition code is known and indicates a branch.
   hi, Will not he taken; this is the case for conditional branches whose condition code is known and indicates no branch.
   c. Outcome cannot yet be. determined. In this ease, the branch is guessed to he taken For backward branches (typical of loops) and guessed not to be taken for *forward* branches, Sequenlial instructions past the branch instruction are passed to the execution units in a co nditional fashion. *Once* the condition code value is produced in the execution unit, the branch unit either cancels the instructions in the pipeline and proceeds with the fetched target if the branch is taken, or !iigrtak ror the condiiionaal instructions to be executed. The compiler can use a single bit in the instruction coding to reverse this delaull behavior.

The incorporation of a branch prediction strategy based on branch history was rejected I-Fccause. the designers felt that a minimal payoff would be achieved.

As an example of the branch prediction effect, consider the program of Figure 14.13 and assume that the branch processor predicts 1hal the conditional branch instruction is nol taken (the default case for a forward branch). Figure 14.14a shows the effect on the pipeline if in fact the branch is not taken. In the first cycle. the dispatch queue is loaded with eight instructions, The first six instructions are integer instructions and are dispatched one per cycle to ilie eger unit, The conditional branch instruction cannol be dispatched until it progresses to the lower half of the dispatch queue, which happens in cycle 5. The branch unit predicts that this branch will not be taken, and so the next instruction in sec] **lW1Ce is** tlondiiionally chspatched (inditlatc(.1 by a D'). The branch cannot be resolved until the compare instruction executes in cycle 8. At ihat time, the branch processor confirms that its prediction was correct, and execution continues. 'There arc no delays, and the pipeline is kept full,

Nolte that no instructions are fetched during cycles 4 throughi. This is because the cache is busy during I hose cycles with the cache access stage of the five load instructions. Even so, the inso-nei ion stream is not delayed, because the dispatch queue can hold eight instructions.

Pigurc I 4.14b shows the result if the. prediction is incorrect and the branch is taken. **In** Lliis f::.1!·W, the three instructions starting at the IF must be flushed, and fetching resumes with instructions Starling xl 1A, S1-':. Asa result, the VWC.1..1k stage of the integer pipeline is idle for cycles' and 10. resulting in a two-eyele loss hccatisc: of the incorrect prediction.

PowerPC 620

The f2(1 is I he first 64-bit implementation of the Powei.PC architecture. A notable feature *of* this implementation is that it includes six independent execution units:

- Instruction unit
- Three integer units
- 1-.041dIstoN: unit
- Floating-point unit

This organization enables the processor to dispatch up to four instructions simultaneously to the three integer units 4inci one rloating-point unit.

The 620 employs a high-performance branch prediction strategy that involves prediction logic, register rename buffers, and reservation stations inside the execution units. When an instruction is fetched, it i' issigned a rename buffer to hold instruction results temporarily, such as reyimur stores. Because of the u se ()I' rcnarnu buffers, the processor can *specaol[rsivefv e.* tc:aade instructions based on branch prediction; if the prediction turns out to be incorrecl, then the regilts of the speculative instructions Call be flushed without damaging the register file. Once the outcome or a branch is confirmed, Temporary results can be written out permanently,

T.aelt unit has two or more reservation stations, which 614.1.rc: dispatched instructions that must be held up for the results of instructions. This feature clears these instructions out of the instruction unit, enabling it to Continue dispaiching instructions to other execution units.

```
(a    21
      a      i  bicidi  of
01Se
      a  - o -   -     d    e;
```

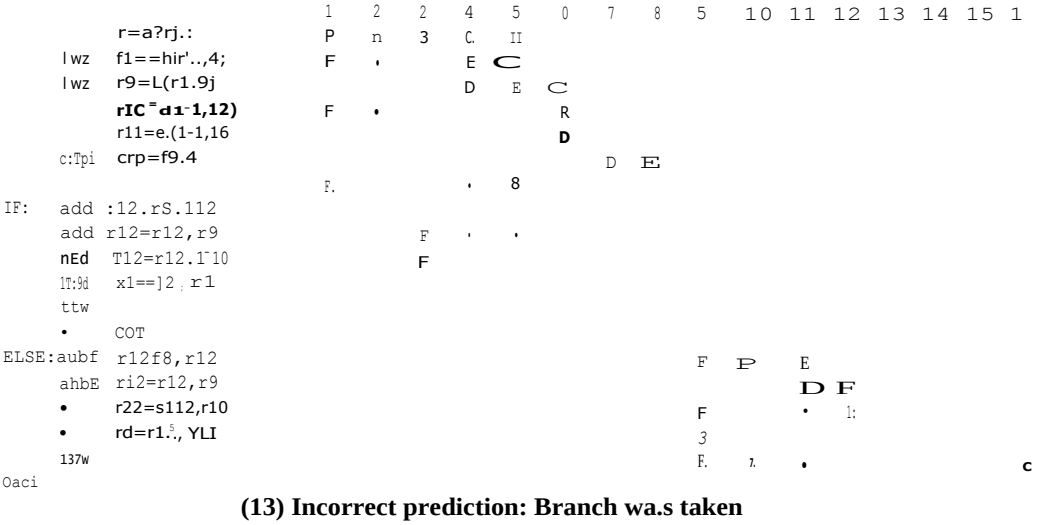
(a) C code

```
                                  r91 poLnts t.o a,
                                  /71+4 points to b,
                                 *r14•5 points to c,
                                  r1+12 point'.2 to d,
                                 fir I.+LE poinr.9 to e
      1WZ    m.9=et(r1           Ricad a
      l Wz   :12=b(r1.4)         Oload b
      'l ea                      4load c
      lwa    ✓107dr1,12)         41Qad
      1WZ    ✓:1=er1.16)         41ad e
             cra=r8,C.           4con'nare itndint.e
      bc     ELgE,L7r0/gt=  1..3e Obranch if btt
IF:
      add    T:2=r8,r12          >k add
      add    ✓12=r12,r9          Oadd
      adcl   :712=r12. 2:'3      *acid
      add    r4=r12,r:::L.       fl add
      stw    2".c 1)=rd          #soret
      b                          4unc.onditionai branch
ELSE:
             r12=r12,r8          Itstlbtract
             r12=r9,1c:2         Osuhtra=
             r1.2=r1   '~L2
             r4=r12,rli
      5ew    a{r:O=r4            4store
OUT:
```

(b) Assembly cod

Figure 14.113 Codlc E.N.Hrrspl.c. wit h Branch IVELS941

(a) Correct prediction: Branch was not taken



(13) Incorrect prediction: Branch wa.s taken

F= fetch
D =dispatch/decode
E= execute/address

C =cache access
W writeback
S dispatch

Figure 14.14 Branch Pmdiction: NoL Taken !NE/S94]

'Pho 620 can speculatively execute up to rour unresolved kaneh inMructions (versus ono for the 601). Bruch prediction is based on the use of a brunch hislory mble with 2448 entries. Si/inflations run by the PowerPC designers show that the branch prediction sitieoes rate is 90% /THON194].

14.5 RECOMMENDED READING

IJOHN911 remains a relevant and excellent hook-lengt It treatInent 4)1 qiperscalai LIr:41 n_ Worthwhile survey articles on the subject are [SVIrl'95] and [SI MA97]. [J01.; P891] instruction-level parallelism. looks at various techniques for maximizing parallelism, and CO 31-pares supersealar and superpipelined approaches using simulation_ Two recent papers that pritvide good coverage of superscalar design issues are [PATT01I and I MOSI-101 I_

[POPE9J] provides a detailed look at a proposed supersealar machine. It also provides an excellent tutorial on the design issues related to out -of-order instruction policies. Another look at a proposed system is found in [KUGA91]; this article raises and considers most of the import atil doL:ig n issues for superscalar implementation. [LEE91 examines software tech niques rh;li used to enhance s u pc rsealar performance. [WALL91] is an interesting study of the extent iii which instruction-level parallelism can he exploited in a supersealar processor.

Volume 1 of [IN 11201a] provides general description of the Pentium 4 pipeline; more detail is Provided in [1 \'i'1- 01

[POTT941 is a detailed examination of instradion pipelining on the PowerPC 601, [SHAN95] also provides good coverage_

NNW Hinton. G., cz al, 'The Mieroarehitecturc of the Pentium 4 ProcE:ssor. - *hue? Teehnology Journal,* Q1 2001. lutrAeveloper.intel.coinhechnology!itil

INTEOIa Intel Corp. *IA-32 Intel Architecture Software DeYeloper's Manual (2* Poirimes). Document 245470 a nd 24547 L Aurora, CO. 2001.

INTEOM Intel Corp. *Wei Pentium 4 Processor Optimization ,Refcrence Manual.* Document 2489M-04. Aurora. C.'0. 2001. hilp:Alevetoper.intel.cornAtesignipentium4imanuals124-894Klit m

.101'IN91 Jobrison. hl. *Supc.rwaiar* ,Wieroprocessor *Design.* Englewood Cliffs, NI: Prentice Hall. 1991,

JOUP$9a Jouppi. N.. and Wall. D. "Available Instruction-Level Parallelism for Superscalar and Superpipelincd Machines." *P•oceedings.. Third International Confirence on A rchiteLtural Support for Progran*tin nts *Langti.age'S mil Operating Systems,* April 1980.

KLIGA91 Kuga, Murakami. K.: and Tomica, S. "DSNS (Dyllarnic;111!'i-huarcl Statically.code-seheduled, Nonuniform .Superseatary Yt1 A ıii ither Superscalar Processor Architecture. - -C7orriprio- *rrhifiectrirc J* [In [991..

LEE91 Lee, R.: Kwok, A.: and Briggs. "Tlic Floating **Paint** Performance u(a Superscalar SPARC Processor:" *Proceediri.;,.... Fourth Interria04mal Conferetue Architectural Support Pr PrOgrantMtng .fne :;r wkres ffini Operating Sy* !....erns„A„pril 1991

MOS11111. Moshovos, A.. and Sohi. G. "Microarchitectural Innovations: Boosting Micropupa' *fir* forroun cc Beyond Semiconductor Technology Scaling." *Proceedbrgs of the Noveitilya* 2001.

PAr1111 Putt, Y. - Requirements, Bottlenecks. and Good Fortune: Agents for MicroprocesSor Evolution.' *Proceeriin,e•s of the IEEE,* November 2001.

POPE91 Popescu. V._ et al. "The Metailow Architeetme." *Micni,* Jane 1991.

POTT94 Potter. et id, "Resolution of Data and Control-Flow Dependencies in the PowerP(.- (101." *I kl..E .4ilicro,* October 1904,

SU A N45 Shanley, 1'. Perwera" *Sy.S1Cri2 A rchitecrure.* Reading. MA: Addison•Weslcy. 1995.

SIMA97 Sims, D. '•Superscalar instruction Issue." *1.6F ' Mt.cro,Seriterrik.v,r1Ortotxt 1()Ln.*

sron95 smith, J., and Sohi, (3. The Microarchitecture clF Superscalar P.rocessors," *Pre)..- eftclings of the IEEE.* nc cenil Rrt9V3..

1r' ALL91 Wall, D. '•1.ini1s IFt instruction - Level *Pmc aretkth}s, Rir.orcle trieeer-teenioreal* **a}-/JP:VetECT On** *A rchieeco fro) Seipport for PreJgreem.Pree)q Ltengrurges irrrel {)per-. along Sy...lems,* April 1901,

14.6 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

araidependency	instruction ·rv in 41 c	regi L'.!" renaming
branch prediction	machine parallelism	resource conflict
in-order issue	out-or- **artier** completion,	snperpi
in..-ordQr cornplutiott	utat-of-lvdcr issue	supersealar
instruction isso	output dependenq	true darn dependency
instruction ' level parallelism	**pracc** dti **rai dapendvicy**	

Review Questions

14.1 What is the essential characteristic of the superscalar approach lo processor design?
14.2 What is the difference between the supersealar and superpipelined approaches?
14.3 What is instruction-level parallelism?
14.4 Briefly define the following terms:

- .Frite tlata dependency
- Procedural dependency
- Resource conflicts
- Output. dependency
- Araidependeney

14.5 What is the distinction between) ins1ruction-lcv el ı **and** machine parallelism?
I4A List and briefly define three types of supersealar instruction issue policies.
14-7 What is the purpose or an instruction window ?
14.8 'hal is register renaming and what is its purpose
14.9 What are the key elements of a supersealar processor organization?

Problems

mi. When out. of .order completion is used in a superscalar processor. resumption of execution after interrupt processing is complicated, because the exceptional condition may have been detected as an instruction that produced its result out of order. The program cannot be restarted at the instruction following the exceptional instruction. becati."0 **Subseci uenE** instructions have already completed. and doing so would cause cheso ii IkI 31.14.:11c)lis to be executed twice. Suggest a mechanism or mechanisms for deal-iiig will' 115i ·

14.2 Consider the following sequence nr instructions, where the syntax consists or an tpcodc followed by the destination register followed by one or two source rettistersi

```
                    a2, Al, 2
I       LOAD'       6, 11R3.
        AND         RI, Rt.,
3       ADD         R1, 75, RO
        SRL         R7, RO, 8
5       OR          R2, R4, .7';
                    R1,   3, R4

W       LOAD        R6, [R5:
                    ?2, 71, 76
11:     AND         ?3,       15
```

A ic .;11rrEC the use **Of** 'n **1{11.3r -Stilge I1i1 '4 Eil1t** i **i** L.11. ch..code/issue. execut.L., write 1.'Fack, pipeline shigesiaki' **irk!** clodk cycle except for the execute stage. For **sirlrl I** 111[4 ii' ..tit inctie ;lad iir,tructioris, the c.xecute stage takes une cycle, hill fall **a 1** .0All ['tom memory, five e I s ire consumed in the execute stagc,

we have a simple scalar pipeline but allow otol-of-order exec:01110o, Wet eau construct the following table tor the execution of the. first seven instructions:

Instruction	Frith		IF:XeCUIV	Write Buck
0	L)	ı	7	:1
I	I	o	4	.1
7	7	:r		
3	3	4	10	I1
4	4	5	**6**	7
5		6	8	I0
(1	{..	7	9	I '1, %

I Iw L'1.21Ci4J.S 4411 ',',k¹ (thk: the CILICk cycle al. which each i i4.12ins end, ErhIasLC, rii 115i, the second ADD instruction (insiruc.. Lion 1 thipenik 1.0A I) IIKIE dui ion (11E...1 tiierion I) ror one of its operands, th. Becatiso the LOAD iciAttiL tioit I I **v** clock cycles, and the issue logic encciunters the ElenuoLlein ADD ir hl fuel **of ici** clucks, ihu issuc logic must delay 1.1.1c ADD Instruction I'iir i hre c clock cycles.. With an out-of•order capability [tic processor can stall instruction 3 at clock. cycle. 4, and then move on LC issue the following Olive nide-Re.ndent instructions, which Enter Execution at clocks 6, 8, and 9. The LoAn finishes execution at clock 9. and so the depundeni ADD can be launched into execution on dock [(1.

a. Complete the preceding table..

b. 11?....lrli I 111.4 ;•,k1)11:, assuming no otit or order capabiliv, What is the savings using the capolAility'1

e. Redo the tablu, assuming a•superscalar implementation that can haudly two instructions at a time k it each stage-

143 [n the instruction queue in the dispatch unit of the PowefPC 601, insttudion.- fE;iw Est dispatched out of order w the hranch processing and Iloalin42-puint ***but*** instructions intended for the integer unit must he dispaieha only from the luittoin of the queue. Why this limitalionl

14.4 Produce a figure similar to Figure 14. L4 for the following LR.150 6:

Brarivii preditition; taken:. correct pro.liction: branch WAS taken

h. litanch predici taken: incorte.ct prediction: branch was not taken

14.5 Consider the following assembly larignago program!

```
T1:   MOVE. Rj RT.        /Rj      :R7
12:   Lr d3 R8. {R3       /38 <—
13:   Arf;I:i Fl'., F3, 4  /R3          +  /
r4    Load .7.9.
IF.:  Thl).7.                          (F. )  r 1!•RA1
```

This program includes write -write, read -wrike, and write -read dependencies. Show these.

14.6 Figure 14A5 shows an exampte of a superscalat processor organization. The prom ' scar can issue two instructions per cycle if there is no resource conflict and no data dependence problem. There arc essentially two pipelines, with four pro-uessing. stages (fetch, decode. execute, and store). Each pipeline has its own fetch decode and stun unit. Four functional units (multiplier.. adder, logic unit, and load unit) arc available for use in the execute Stage and arc shared by the two pipelines on a ds.nainic bast The Iwo storc tiniN can be ciynarnically used by the two pipelines, dopending on avail. ability at a particular There is a lookallead window with ils own fetch and decoding ingic. This window is used for instruction lankabead for out-of-order instruction issue,

following program tot execute!] oil (INS processor,

```
Ii t Lcad Al  A          iR.1 4- ke7=y IA)
1.2    Add R2, RI        /R2 <-- :R2 + R;1)/
13     Add R3, R4             f-       + R14)/
14:    `1u1 P4, R5       /R4     ;P'U +
15r    C.1%rp P     /R6  (R6) /
16 r    T'.i1 R6, F.'?   /R:J.          + R
```

H. What dependencies exist in the program.

b. Show the pipeline activity for this program on the processor of Figure 14.15 using in. order issue. with in **.order** completion policies and using a presentation similar to Figure 14,2,

e. Repeat for in-order issue with out of•order completion.

d. Repeal for out-cif-order issue with out-of-order completicin.

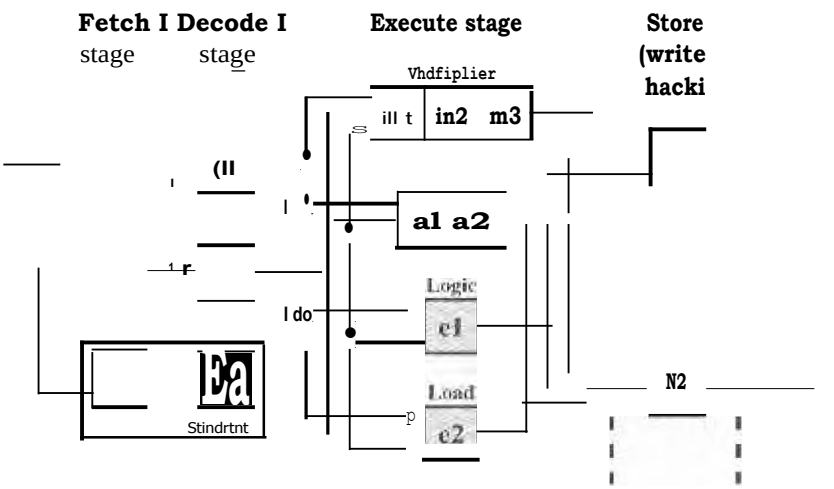


Figure 14.15 A Dual-Pipeline SuperseHlkir Processor

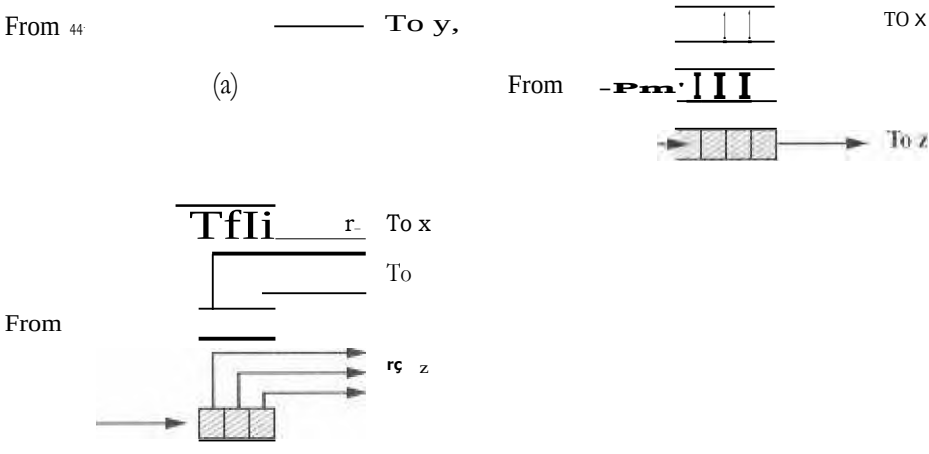


Figure 14.16 Figurc for Problem L4.

14.7 Figurc 14.10 i. from a paper oil .supi.ltrscaliir **&sign.** Explain the L]ire s of tk. fio:urc, and &rimc w x. y, and

CHAPTER 15

THE IA 64 ARCHITECTURE

15,1 helotivatioa

15.2 General Organization

15.3 Predication, Speculation, and Software Pipe lining

- Instruction Format
- Assembly-Languagc Format
- Prcdicatt:d Excculion
- Control Speculation
- Data SNculation
- Softi,vare Pipelining

15.4 IA-64 instruction Set Architecture

- Ragister Stack
- Curreml Frame lk.Ur1;:er ow] Previous Fmk'lion Stare,

15.5 It-0111PM Organization

15.6 Reconnuended Reading and Web Sites

15.7 Key Term?, Review Quesi and Problems

- Key '1ernas
- Rcview Qunt ions
- Problems

KEY POINTS

- The LA-64 instruction set architecture is a new approach to providing Turd-**ware.support** for instruction-level parallelism and is significantiy different that the approach taken in yverscalar architectures,
- 'i he most noteworth!,.: features of the IA-64 arcin [eel Lire arc hardware s.upport for predicated execution, control speculation_ data spc.cu[atiort, and software pipeiining.
- With priAicated execution, every IA-64 instruction includes Ei reference to a [-bit predicai regisLer and only executes if the predicate value is L Orue). This enables the processor to speculatively execute both branches of an statement and only commit after the condition is deli:mined,
- With control speculation, a load instruction is moved earlier in the program and its original position replaced by a cheek instruction. The early load s.;Lvt:.s cycle time; if the Load produces an exception, the exception is not activated until the chui.:k instruction determines if the load should have 1. '5een taken.
- With chin speculation, a load is moved bci'ore a store in struction that mi0t. alter the memory location that is the. source of the load. A subsequent check is made to assure that the load receives the proper memory vise.
- Software pipelining is a technique in which instructions from multiple itera-Eions of a loop are enabled to execute in parallel.

Witli the Pentium 4, the microprocessor family Thal began with the 8086 and | hat has been the most successful computer product line e 'L.er appears to have come to an end. Intel has teamed up with Hewlett-Packard (HP) to develop a new fiz1--hit architecture. called IA-64, IA-64 is not El 64-bit extension of biters 32-bit x86 architecture_ nor is ii an adaptation of IllcwIca. Packard's 64-hit PA-RISC architecture. Instead, IA-64 is a new architecture that builds on years of research at the we companies and pit universities. The architec-I Lire exploits the vast circuitry and high speeds available on the newest gcnoraiions or microchip; by a systematic use of parallelism. IA-64 architecture represents a si, nificant departure from the trend to supersca]ar schemes that have dominated recent processor development.

We begin this chapter with a discussion of the motivating factors for the new architecture. Ncxl, we look at the general organization to support the architecture. We then examine in some detail the key features of the IA-64 architecture that pro-mote instruction-Level parallelism. Filially, we Look a(the IA-64 imslruction set archi-tecture and the Itanium organization.

15.1 MOTIVATION

The, basic concepts underlyinr2. IA664 are as follows.:

- Instruction-level parallelism that is explicit in the machine instructions rather than being del Qrmincil AL rim lime by the processor
- Long or very long instruction words (LINV/VLIW)
- Branch prodica tlgn (ni)i. the same ihin2, as branch prediction)
- Speculative loading

[Mel .tind H P refer to this combination of concepts as explicitly parallel inslruC-iian computing (EPIC). Intel and HP use the term **EPIC** to refer kJ the. technology, or collection of techniques. **IA-64 is** an actual inAtruction rct architecture that is intended for implemenivition **using** the EPIC technology. The first Intel. product based on this **IA-64** is referred to as **lianium.** Other products will follow. based on the same IA-64 architecture.

Tablc.. 15.1 sLimmarizes key di **ITerc.ricc!, helmecu LA-64** and a traditional super-Sea I **ar** 2l pprozich.

For Intel, the move to a new architecture, one that is not hardware compatible with the xSfi instruction architcolure, i;'.; a momentous decision. But it is driven by the Llicc ilex of the iechnology. When the x86 family began. back in the laic 1970, tlie processor chip had tens of thousands of transistors and waS Sin CNs.0 111 141 I Iy scalar device. Thai is. instructions were processed one **li me.** with little LI no pipelining. As the number transisl ors increased into the hundreds of thousands in **1hiz, mid-1980s.,** Intel introduced pipelining Figure 112.1.S). Meanwhile, **.E.)1 hur** manufacturers were attempting to take advantage of the increased **1 ra.m'istor** count and increased speed by means of the RISC' approach, which enabled more effective pipelining, li nd la Let the superscalar/RISC. combination. which involved mul execution. units. With the. Pentium, Intel made zr 11.1 0deS1 **14.) use superscalku.** techniques, gnawing two CISC instruct ions to cxecute it of lime. 'Then_ the Pentium Pro and Pentium II through Pentium 4 incorporated a mapping from CISC instruc-

1'Ab1e **1.5.1** PraLliiional Supers.cular versus IA-64 Architcillurc

Siiiicrscalar	**14-64**
RESC-llaC: instructions...one. pi.kr word	1.z.i S.C-ILFW LDst rvciiorm bundlud into gcCitipS Cif 1..Eifee
Multipl:2 pilTillIC-] CXZCL.11.1011 1.11111 LS	Mull iple parallel ...:•xN111 ion kalils
Nxnrdcrq ki nd OPI.ilrillrC!.; liKE11.101011 slrekint io T1.111 L1111.2	Reonlurs dad opciniivos iii3lru4tion Ntrcarn at i_DELS1)11V Linl.:.3
Brandt fil'OdiCt it)1.1 wit h spcculoieivc il..xcciiii(1T] of ORO path	Speculative excuukitn) akinp h431.1t paths of a branch
Loads dithi from 1.13 01101'y o.rily Mica neoiled. iiiid tries so find ell:: diviii in clic ciiclics ririi	Spc.culiiii.vc.:si.ciiids **dl**.1 1)12(1T12 IS riecided. arid still tric.s Ia rind data in tlii2 caukie; first

lions to RISC-like micro-operations and the more aggressive use of superscalar techniques. This approach enabled the effective use of a chip with millions of trarl• sistors. But for the next generation processor, the. one beyond Pentium, Intel and other manufacturers are faced with the need **to** use effectively tens of millions of transistors on a single processor chip.

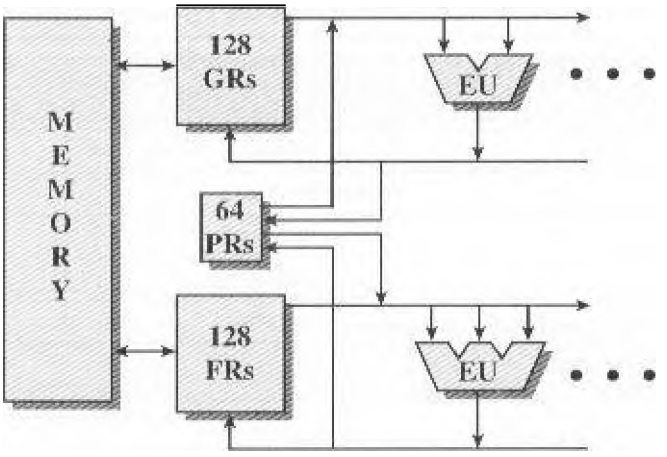
Processor designers have few choices in how to use this glut of transistors. One approach is to dump those extra transislors into bigger on-chip caches. Bigger caches can improve performance to a degree but eventually reach a point of diminishing returns, in which larger caches result in tiny improvements in hit rates. Another alternative is to increase the degree or 5IJperscaling by adding more Execution units. The problem with this approach is that designers are, in effect, hitling a complexity wall. As more and more execution units arc added. making the processor "wider," more logic is needed to orchestrate these units. Branch prediction must be improved, OW-of-order processing must be used. and longer pipelines must he employed. But with more and longer pipelines, there is a greater penalty for misprediel isrn, Oui-ef• order execution requires 4i large number of renaming registers and complex inter-lock circuitry to account for dependencies. As a result. today's best processors can manage at most to retire six ilIMI'LLCijOils per *cycle,* and usually less.

To address these problems, Intel and HP have come up with an overall design approach that enables the e 'ketivc, use of a processor with many parallel execution units. The heart of this new approach is the concepi o[explicit parallelism, With this approach. the compiler statically schedules the. instruetions.at compile time, rather than having the processor dynamically schedule them at run time. The compiler determines which instructions can execute in parallel and includes Ibis information with Ihe machine instruction. The processor uses this informal lo perform paral• Eel execution. One 44.1vantEigc of this approach is that the EPIC processor does not need as much complex circuitry .t7:1 an out-or-order superscalar processor. Further, whereas the processor has only a matter of nanoseconds to delermine putenl ia] par-allel execution opportunities, the compiler has orders of inagn itudc morn time in examine the code at leisure and see the program as a whole.

15.2 GENERAL ORGANIZATION

As with a nv processor architecture. IA-64 can he implemented in a variety of organizations. 1;igure 15.1 suggests in general terms the oreanization of an IA-64 machine. The key features are 2151 {1110W:;,;

- **Large number of registers:Thu** l A-64 instruction Format assumes the use of 256 registers: 128 64-hit registers for integer, logical, nd general-purpc.i.sc use, and 12g 82-hit registers for floating-point and graphic use. There Tire also 64 1-hit predicate regkters used for predicated execution, as explained subsequently.
- **Multiple execution units:** A typical commercial superscalar machine today may support four parallel pipelines, using four parallel execution units in both the integer and ItoaIing-point portions of the processor. It is expected that l A-64 will he implemented tin xv7'11cinS with eight or more parallel units.



rR = Gcncrul-inirposc or inieger'reisior
I- H. =I-loafing-point or *graphic:,* register
PR = predicate register
3211 = Ikixecution unit

Figure 15.1 Gent.ral Organization for IA•i4 Architccturc

The register file is (wile. large compared with n i mi R[S(' and superiicAar machines. Thc reason for this IN dial ai large number of registers is needed to support a high degree of parallelism. In a traditional supersca]ar machine, the machine language {and the assembly language) employs a small number of visible registers, and the processor mum I he se onto larger number of registers usi ng register renaming technique.s and dependency analysis. Because We wish to make parallelism explicit and relieve the processor of the burden of register renaming and dependency analysis, we need a large number of explicit regislETS-

The nutniler of esceui ion units is u function of ihe number of transistors available in a particular implementation. The processor will exploit parallelism to the .extent that it can. For example. if the machine language instruction siro.iim (es that eighl integer instruction may he 12:ccuted in parallel. a Tyroces!,or with four integer pipelines will execute ihese in two chunk r.. A processor with eight pipelines will execute all eight instructions simultaneously.

Four types of execution unit are defined in the IA-64 ..irchitecture!

- **I-unit:** For integer urit hinetie. shift-kind-add, logical, compare, and integer multi media instructions.
- M-unit: Load and store between register **and** memory plwi:ionle integer ALI; **o[IcratiunS.**
- B-unit: Branch instructions.
- iro„iruclicinN.

TWA 15.2 Relationship between Instruction Type and Execution Unit

Instruction Type	Desrriplinii	1.:".c.cution Unit Type
A	integer ALU	1.unit or.M.unit
I	Non-ALU intem	1-unit
M	Mcinory	M-unit
P	floating pm'li	F-tniii
B	Branch	13-unit
L I X	Extended	I-onit/13-unic

Each I.A-64 instruction is categorized into one of six types. Table 15.2 lists the instruction types and the.exectition unit types on which they may be executed.

15.3 PREDICATION, SPECULATION, AND SOFTWARE PIPELINING

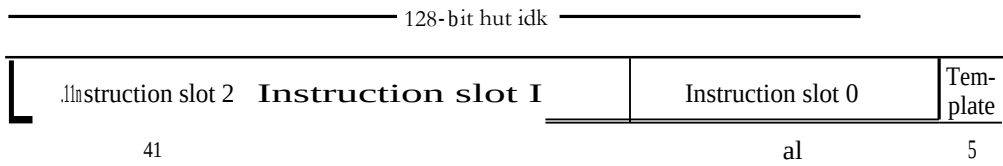
This suction looks at the key features of the IA-154 architecture that support instruction-level paraiiehsrn. First, we need to provide an overview of the IA-64 instruction format and, to support the exEimples in this section, define the general format of lA-64 assembly language instructions.

Instruction Format

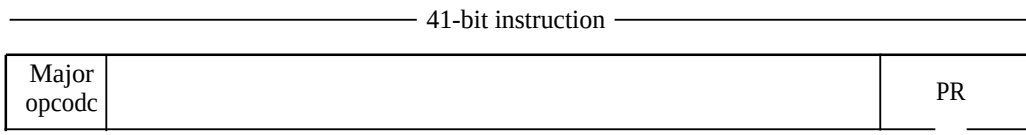
l A-64 defines a 128-1 ¯iii bundle that contains three instructions, called syllables, and a template field (**Figure 15.2a).** 'File processor can 1'0.6 instructions one or more bundles at a time: each bundle fetch brings in three instructions. The template field euntains in l'ormation that indicates which instructions can be executed in parallel. The interpretation of the reit ¯Th11e field is not confined to a single bundle. Rather, the processor can look at multiple bundles to dctermine which instructionp. may be executed in parallel. For example, the instruction stream inav be such that eight instructions **can** be execuied in parallel. The compiler will reorder instructions so that these eight instructions Daman contiguous bundles and net the lerrlphle hiss SO that the processor knows that these eight instructions re independcnt.

The bundled instructions do not have to be in the original program order. Further, because of the flexibility of Lite template field, the compiler can mix independent and dependent instructions in **the Ndrtle** bundle. Unlike some previous VLIW designs, IA-M does not need to insert null-operation (NOP) instructions to fill in the bundles.

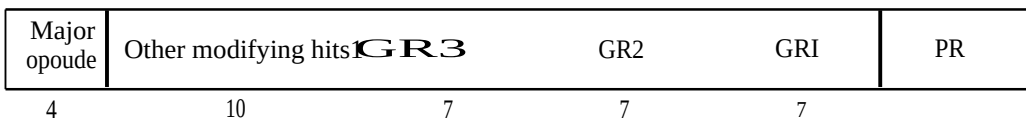
Table 15.3 shows the interpretation of the possible values For the 5-bit template field (some values are. reserved and not in current use). Tire template value. accomplishes two purposes:



t a) IA-64 binallc



II)} General 1A-64 instruction format



(C) Typical 1A-64 instruction format

PR — Pprdi•ale register
OR = General or floating-point register

Figure IS/ 1A4iLl instruction Fortnai

1. **The field specifics the mapping of instruction slot, to execution unit** types. **Not possible mappings or instructions to units are available,**
2. **The field indicates the presence** of **any stops. A stop indicates to the hardware that one or more instructions** before the **stop may** have certain kinds of resource dependencies with one or more instructions after **the stop. In** the **table, a** heavy vertical line indicates **a** stop.

Each instruction has a fixed-length 41-hit format (Figure 15.2b). This is somewhat longer than the lraditional 32 -bit **length found on RISC and RISC' superwalar** machines (although it is much shorter **Than** the 118-bit **micro-operation of the Pentium 4). Two** factors lead to the additional **hits.** First, IA-t4 makes **use of more registers than a typical RISC machine: 128 integer and** 128 floating-point registers. **Second, to accommodate the predicated execution technique. an IA-64 machine** includes 64 predicate registers. **Their** use is explained subsequently.

Figure 1.5.2c shows **in more detail the** typical instruction format. All instructions include a **4-hit major opeode** and a reference to a predicate register. A/though **the** major opcode **field** can only discriminate among 16 **possibilities, the interpreta-**

Table 15.3 Template Field Encoding and Instruction Sc(.Mapping

Template	Slut 111	Slot 1	Slot 2
IX1	M-ursiL	[-unit	I-unii
LI.1	M-urkil.	1 _snit	1 _unil
112	1.1-unit	I-unit	1-unit
IY3	h3-unit	T-unit	1 - unit
LF4	M - urkli	1.--untl	X-11nit
05	M-uni1	L-unis	X-1.11111.
Og	rirl•unii	M-ullil	E-unit
04	!'rl-unlit	M-unit	[-unil
OA	M-unit	M-unit	[_unh
IIB	'24 ' unis	NI _unit	1 _unit
OC	M_ unit	F-unil.	f-unil
OD	M-unit	F-unil	I _ unii
OE	NI-unil	Ivl-ursil.	F-unit
OF	N1-unit	Pvl-anit	F-unit
10	M - unit	I -Lull t	B-111114
II	M - unii.	1•unit	B-uttic
42	Nef-unit	B-unit	13 - uniL
1,3	M - 1.131il.	B - unit	B-unit
L6	Th - unit	El-unit	13•unit
t7	13-u110	B-unit	B-unit
I's	N - uriii	M - unit	B - unii
19	M-u101.	ht - unit	B-unil
I C	M-nrni	F•unit	l3-unit
I n	M-unii	F-tiit	1:5 - unit

lion Or the major opcode field depends on the ternpl.ate2 valuu nd the locifi.ion instruction within zi hundlf: (˙Fable. 15.3 .1hus affording more possible opcodes. Typical instructions also include reference registers. leaving 1.0 bits for ether in rormation needed to fulby specify the instruction.

Assembly-Language F ormat

As with i:iny machine instruction set. an assembly language is provided for the convenience of the pro@,rafarner. The sissUnbler or compiler then translates each assail-

bly language instruction into a 41.-bit JA-04 instruction- The gencrffl format of an assembly language instruction is

[cip] mnerrionic[-r : dt?,5LL=src

where

cep	Specifies a 1-bit predicate register used 0} qualify the. instruction. If Inc value of 1Vgkier is I (true) at execution time, the instruction executes and the result is committed in hardware. It the va [Lie is 1•.11:',112, the result of the instruction is not committed but is discarded. Most [A-64 instructions may he qua li lied by a predicate but need not be. To 4.account for in instruction [hat is not predicated, the cip value is sc1 to 0 and predicate register zero always has the constant vahJe of 1.
nrnemonic	Specifies the name of an IA-t4 insi ruei ion.
COM"?	Specifies one or more instruction completers, Separtitcd by periods. which are used to qualify the mnemonic. Nol Fill insiroutiLms require the Use of a eompicrer.
ifrvl	Specifies one or itlOrC destination operands, with the typic,a1 case being a single destination.
4TE"	Specifics one or more source operands. Most instructions have two or more source operands.

On line, any el7eiraelers Lo **the** right of a double slash ¯]C are created as a comment. Instruction groups and stops are indicated by a (10131.111. W.11147011ln An instruction group is defined as a sequence of instruciions that have no read after write or write after write dependencies- ']'he processor earl issue these without hardware checks for rel.!ister depe.ridehrie. I Jere is a simple example;

```
=dB r= = [r5]          /i First gyDup
acd r3 = X ,  r4       / SecpnCi L.Tr up
```

The First instruction reads an :z.-hyte value from Ihe mcmory [ocation whose address is in register r5 and then placeNil wt value in register it The second instruction adds the contents of rl and M. and places the result in r3. Because Ihe second instruction depends on the value in rl, which is changed by Ihe first instruction, the two instructions cannot be in the same group (or pnillel execution.

Hero is. a rylone cumpJe example, with multiple register flow dependencies:

```
1d2 rl  7 [L 51          // E; rst group
sub r6 = r8;      ;;     // FLrgt group
       = rl,  174        i/ 8econd
   F.r.E1 - r12          /i Second
```

'fhe [am instruction stores the contents of r]2 in the memory location whose address is in r6.

We are now ready to look at the low key mechanisms in the IA-64 architecture 10 support instruction-Level paza lielism:

- Predication
- Control speculation
- Data speculation
- Software pipelining

Figure 15.3, based on a figure in [HALF97I, illustrates the first two of these techniques, which are discussed in this subsection and the next.

Predicated Execution

Predication is a technique whereby the compiler determines which instructions may execute in parallel. In the proce.ss_ the compiler eliminates branches from the program by using conditional execution. A typical example in a high-level language is an **if-then-else** instruction. A traditional compiler inserts a conditional branch at the **if** point of this construct. **If** the condition has one logical outcome, the branch is not taken and the next block of instructions is executed, representing the **then path;** at the end of this path is an unconditional branch around the next block, representing the else path. If the condition has the other logical outcome, the branch is taken around the **then** block of instructions and execution continues at the else block of instructions. The two instruction streams join together after the end of the else **block. An IA-64** compiler instead does the following (Figure 15.3a):

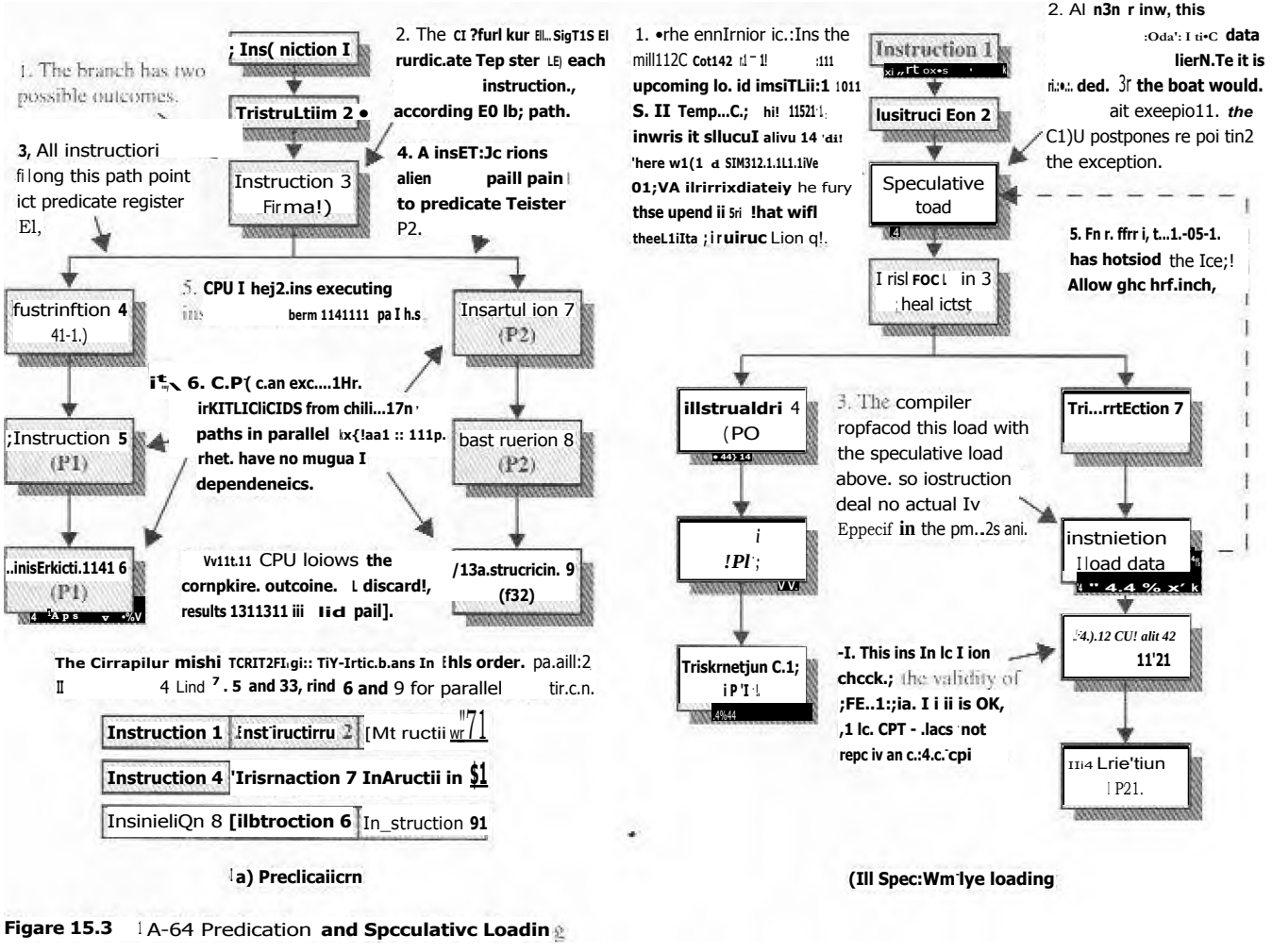
1. At the **if** point in the program, insert a compare instruction that creates two predicates. If the compare is true. the **first** predicate is set to true and the second to false: if the compare is false, the first predicate is set to false and the second to true.
2. Augment each instruction in **the then** path **with a reference to a predicate** register that holds the value of the first predicate, **and** augment each instruction in the **else** path with a reference to a predicate register that holds the value of the second predicate.
3. The processor executes instructions along both paths. When the outcome of the compare is known, the processor discards the results along one path and commits the results along the other path. This enables the processor to feed instructions **on both** paths into the instruction pipeline **without waiting** for **the** compare operation to complete.

As an example. consider the following source code

```
if (a&s,b)
        = J I t;
falae
```

Source Code:

```
            k = k +
     else
            <    k - 1
i = i -
```



Figare 15.3 A-64 Predication **and Spcculativc Loadin**

Two if statements jointly scicci eort hrec possible e xccu ti on paths. This can be compiled into the following code, using the Pentium assembly language. The program has three conditional branches and one unconditional branch instructions:

Assembly Code:

```
rxr1)          ; c=ompare a with 0
je Li          ;  branch to Li if a =
curl b, 0
le Li
adti j r  i    ; j     J  F
jimp  13
{nip

add. :c, 1     ;  k=k-FL
jrr.p L3
sub    3.      ; k     k  -
```

In the Pentium assembly language, a semicolon is used to delimit a commcnk,

Figure 15-4 shows a flow diagram of this assembly code. This diagram breaks the assembl!„y language prograiti into separate blocks of code. For each block that

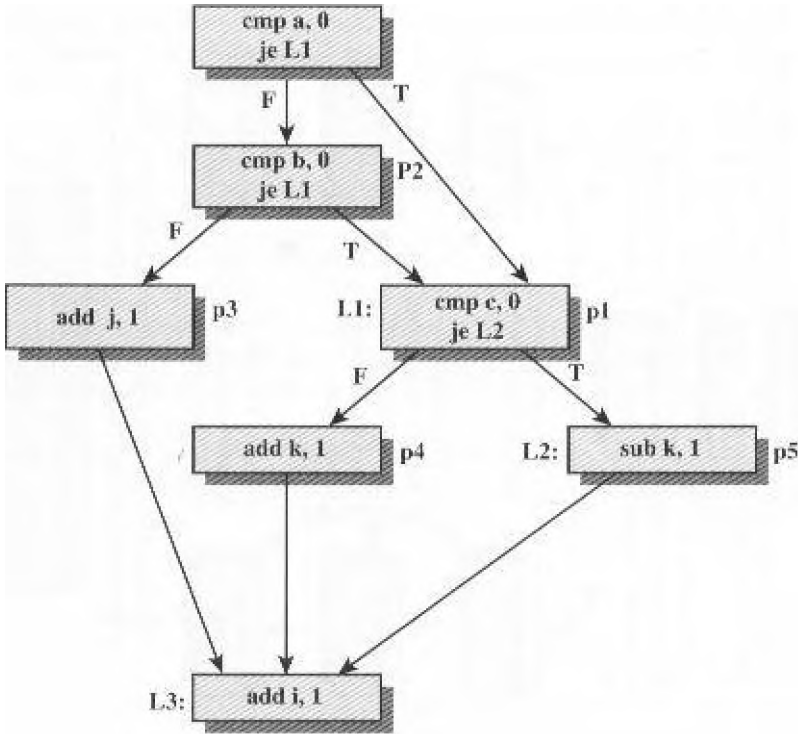


Figure 15A Fxample Pre.dioi

executes conditionally, the compiler can assign a predicate. Thest.: predicates are indicated in Figure .15.4. Assuming that all of these. predicates have been initialized to false. the resulting IA-64 assembly code is as follows;

Predicated Code:

```
:1          cmp..F,q p1. p2 . = c, ,    ;:
:2   (p2)    car      p1, p3 = 0, b
 3   (p3)    add j = L, j
     (p1)    cay..ne p4. p5 = 0, c
     fp4)  add k = 1, k
;6   fp5)    add k   -1 k
             add  ==
```

Instruction (E) compares the contents of symbolic register a with th it sets the value of predicate register pl. to I (true) and p2 to 0 (false) if the relation is true.and will set *the* value of predicate p1 to 0 and p2 lo 1 if the relation k false. Instruction (2) is to be executed **only** if the predicate v2 is true (i.e., if a is true, which is equivalent to a ;& 0). The processor will fetch, decode, and begin executing this instruclion, ?l,ll **only make** a decision as to whether to comthil the resu]1 after it determines whether the value of predieHtc register t,1 is I or 0. Nli that instruction (2) is a predicate-generating instruction and is itself predicated. This instruction requires three predicate register fields in its format.

Returning lo our Penguin program, the first two condilional branches in chic Pentium assembly code are translated into tv,..o IA-64 predicated compare instructions. If instruction **(1)** sets p2 to false, the instruction (2) is not executed. After insiruction (2) in the IA-64 program, p3 is true only if the oilier **it** sIat6,rrient in the source code is true. Thal is, predicate p3 is true only if the expression (a AND El) is true (i.e., a T 0 AND h 0 0). The **then part of the outer if** statement is predicated on p3 for this reason. Instruction (4) of the 1A-64 code decides whether the addilion or subtraction instruction in the outer **else** part is performed. Finally, 1 he increment of i is performed unconditionally. Looking at the suurce code and then at the predicated code, we see that only one of instructions (3), (5). and (6) is to be executed. In an ordinary supersealar processor, we would use branch prediction lc} guess which 0r the three is to be **executed** and go down that path, If the processor guesses wrong, the pipeline must be **flushed.** An IA-64 processor can begin execution of all three of these instructions and, once the values of the predicate registers are known. commit only the results of the valid instruction, Thus. we make use of additional parallel L'XI:Cul ion units Lo avoid Ale delays due to pipeline flushing.

Much of the original research on predicated execution was done at the University of Illinois. Their simulation studies indicate that the use of predication results in a substantial rcclueti4pn in &mimic branches **and branch mispri2dictions and** a substantial performance improvement for processors with multiple parallel pipelines (e.g,, IMAHL941. IMAHL9.51).

Control Speculation

Another key innovation in IA-64 is control speculation. also known as speculalive loading. This enables the processor 10 load data from memory before the program needs it, to avoid memory latency delays. Also, the processor postpones the reporting of

exceptions until it becomes ncixs,sary to report the exception. The term *hoist* is used to refer to the movemeni of a load instruction to•a point earlier in the instruction stream

The minimization of load latencies is crucial to improvinv performance. Typically, early in a block of code, there are a number of loid openitioli.s that bring data from memory Lo registers. Because memory, even augmented with one or Iwo Icyels of cache, is slow uompared with the processor, the delays in oblainingthlta from memory become a bottleneck. To minimize this, we would like to rearrange the code so that loads are done as early as possible. This can be done with any compiler. up lo point. l'he problem occurs i f we attempt to move a load across a conlrol flow, You cannot unconditionally move the. load above branch because the load may not actually occur. We, could move the load conditionally; using predicates. so that I he data could he retrieved from memory but not committed to an architectural reg.

until the outcome of the predicate is known; or we can use branch prediction techniques of the type we saw in Chapter 14. The problem with this strategy is that the load can blow up. An exception due ul invalid address or a page fault could be generaled. If this happens. ilia~ oi would have. to deal with the exception or fault. causing a delay.

Flow, then, can we move the load above the branch? The solution specified in IA-64 is the control speculation, which separates the load behavior (delivering the value) from the exception behavior (Figure 15.3b). A load instruc1ion in the original program is replaced by two instructions:

- A speculative load (Id-s) executes the memory fetch. performs exception detection, but does 110f deliver the exception (call the OS routine that handles the exception). This ld.s instruction is hoisted lo an appropriate point earlier in the program.
- A checking instruction (chk.$) remains in the place of the original load and delivers exceptions. This chk.s instruction may be predicated RID that it will only execute if the predicate is true.

If the ld.s detects an exception.. it sets a token bit associated with the target register, known as the *Nol Thing* (Na]') hit. the corresponding chk.s instruction is exceuied, and if i he Nal' hit is set, the clik.s instruction branches to an excepaorlhantlEing routine.

Let us look at a simple example. taken from [INTEiHla, Volume 'li, Here is the original program:

```
Lp1) br sorr.e_Label          C..sic;1
     1d8  rl =                / Cycl42
     add r2' - I 1, r3        /./ Cyc_e. 3
```

rirsL iiisLrucLion branches if predicate pl is true (register p1 has value. 1). Note that the branch and load instructions are in the same instruetion group, even though the load should not execute if the branch is Laken. IA-64 guarantees that if a branch is taken. later instructions. even in the tame instruction group, are not C.Xe• cutest, 1A-64 implementations may use branch prediction to try I o improve efficiency but must assure against incorrect resailV!:.. Finally. note that the add instruction it delayed by, at least a clock period (one cycle) due to the memory latency of the load operatit,n.

The compiler can rewrite this code using a control speculative load and a check:

```
lda.s  r: = !r5:  F;          /./ cycle -2
                                 ofrier instrotions
                              // C'ycie 0
       rl, reco'vQfy             ;.yule 0
add r2 = rl, x3               /i ,'L'ycle 0
```

We can't simply move the Load instruction above the branch instruction, as is, because the load instrucl ion may cause an exception (e.g., r5 may contain a null pointer). instead, we convcrl the load to a speculative load, Ed 8.s. and then move it. The speculative load doesn't immediately signal an exception when deluded: it just record; that fact by setting the NaT bit for the target register (in this case. H.). The speculative load now executes unconditionally at least two cycles prior to the branch. The chk,s instruction then cheeks to see if the NaT bil is set on 11- 11 not, execution simply falls through to the next instruction. If so, a branch is taken icy a recovery program. Note that the branch, check, and add instructions are a]] shown as being executed in the Namc clock cycle. However, the hardware ensures that the resulis produced by the speculative had do not update the application statc. (c Hnge the coniniis of rl and r2) unless two conditions occur The branch is not taken (pt = 0) and the check does not detect a deferred exception (r1.NaT - 0).

There is one other important point 1(5 **note about** example. If there is no exception, then the speculative load is an actual load and takes place prior to the hranch that it is supposed to follow. If the branch is taken, then a Ioad has occurred that was not intended by the original program. The program. as written. assumes that rl is not read on the taken-branch path- If r1 is read on the taken branch path, then the compiler must use another registet to hold the speculative result.

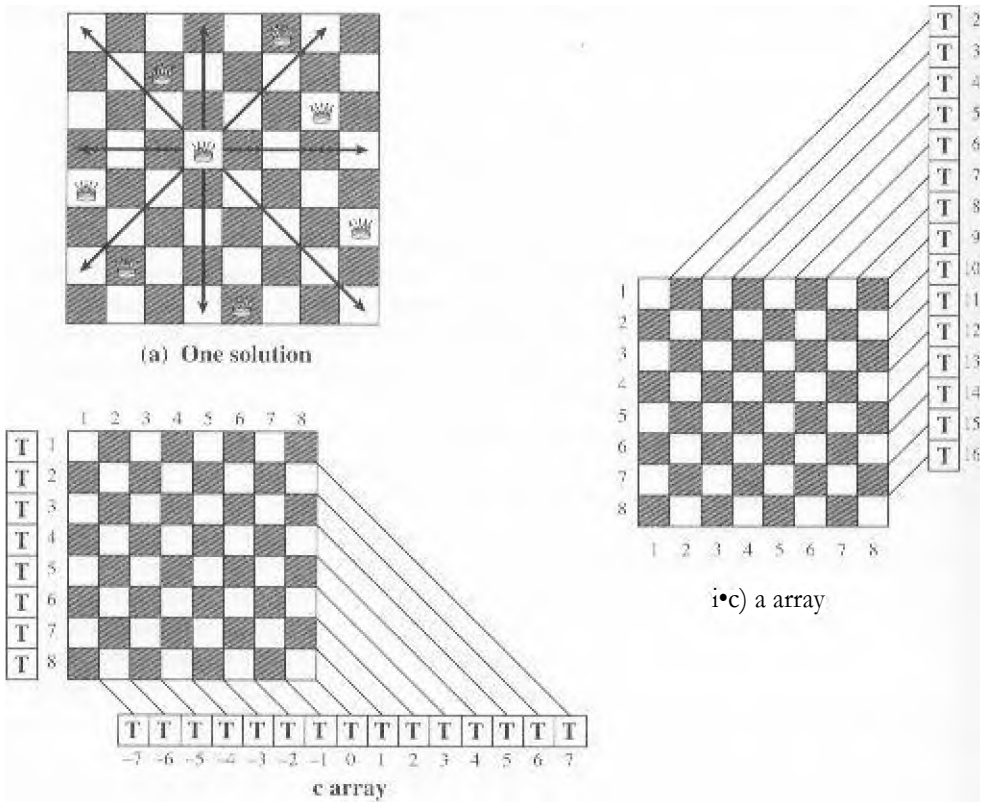
Let us look at a more complex example, used by Intel and HP to benchmark predicated programs and to i [Iasi rale the use of speculative loads, known as the Eight Queens Problem. The objective is to arrange tight q LleeT1S on a chessboard so **that** rtt, queen threatens any other queen. Figure 15.5a shows one solution. The key line of source code, in an inner loop, is the following;

```
if ((b[j] == true)  &&  0E ri 1  j]      1%rue) .z2&  (c[i        ,
```

where 1 j = K.

The queen conflict tracking **Tricehanism** consists of three Roolean arrays that track queen status for each row and diagonal. TRUE means no queen is on that row or diagonal; FALSE means a queen is already there. Figures 15.5b and c show the mapping of the arrays 10 the chess board. All array elements are initialized to]'RUE. The B array elements 1-8 correspond to rows l-8 on *the* board, A queen in row *ot* sets b[n] to FALSE, C array elements are numbered from -7 to 7 and correspond to the **difference 1)ciwtcn** column and row numbers. which defines the diagonals that go down to the right. A queen at column 1. row I sets 401 to FALSE. A queen eMumn I. row 8 sets cH 71 to FALSE. The A array elements are numbered 2–l6 and correspond to the sum of the column and row. A queen placed in column 1, row 1 sets a[2] to FAL-SE. A queen plaual in column 3, row' sets alSiio FALSE.

overall program moves through the columns, placing a queen on each column such that the new queen is not attacked by a queen previously placed on either along a row or one of the two diagonals.



11) b Rriti arrays

Figare **15 The** Eight Ou *Bens* Prohlein

A 1:1traightforward Pentium assemIlly program includes three Icxids itnd thrce-licanchcs;

```
                                                    ea :er cc . ntsof locat:w
                                                  BR: tc yeffiEter r2
                    =ID 2.
                    jr..
                4I          62aL
Aim,erobly Code:  {5;       cmp rl, 1
                  (6)       jr.  L2
                  (7)         V rb,
                  (UJ       cmp r5, 1
                  19)       in.. L2
                                  (.r then pozE:.
                1:1)L2:   l:cDce for wee
```

In thepreceding prounl, the notation & symbol ins tin immediate address for Loe,atiDn x, Using speculative loads and predicated execution yields, the following:

Code with Speculation and Predication:

```
1:          mov rl = &[3(r                 i   ^n«MI    ad. ress  of
                                                        h:j1  to  rl
            most r  = La[L + j]
            nu= r3  = .Stc[: · j 4 71
?4;         .d8  x2 = tr1J                 /i  toad  iniract N/La. rl
(5          1d.f;   x = [r3]
            ldB.s  r6 = [r5]
Y7          crip.eq pl,  p2 = 1,
        Q:12)
.il=?|                rZ,  rEtc very_a      fixup  for  ic.  Lac.1
Ue;         crip.eq p3,  p4   =,  r4
 11:    (p0 br L2
            chk,   r6.  recove.rv_a    i/  fixup  for l•adLag b
 | 3:      cmp.pc n5,  p5 = L.
| 1J:    (pa) br L2
;15iL1;      code   of
            ‹clode  for  191;Re
```

The assembly program breaks down into three basic blocks of code, each of which is a **load** followed by a conditional branch. The address-setting instructions 4 and 7 in the Pentium assembly code arc snit * arithmetic calculations these can be done anytime, so the compiler moves these up to the top- 'C'hcn the compiler is faced with three simple blocks, each of which con sists of a load, a condition calculation. and a conditional branch. Then: seems little hope of doing anything in parallel here. Furthermore, if we assume that the load takes two or more dock eycics, we have some wasted time before the conditional branch can be executed. What the compiler can do is hots the second and third Loads (instructions 5 and 8 in the Pentium cock) above **all the branches. This** is done by putting a speculalive load up top (IA-64 instructions 5 and 6i and leaving a check in the original codc block (IA-64 instructions 9 and 12).

This iransformitiion rrmkes it possible to execute **all** three loads in parallel and to begin the loads early so as to minimize or avoid delays due to load latencies. **the** compiler can go further by more aggressive use of predication. and eliminate two of the three branches;

Revised Code with Speculatilla and Predication:

```
[1              mov  r1
(2)             rt•.7  f3  7  La[i  +  j]
(3)             MGV  r5 = Rc[i  -  j  -  71
                     r2  =  (TI.]
                       r4  =  [r S]
                =de.s rs =  [r5]
                crrip,eci p1,  p2  =  1.  .T2
(8)     (7:)           r4,
(9)     (D1)    LTA.p.eg n3,  n4  = 1,  r4
(10)     p.3)           r6.  xeccvery_b
J11)            cmp,eq p5,      = 1,  z5
            (p)  hr L2
(13)L=          <code for then path}
(14)L2:         <code for elFe
```

We already had a compare that generated two predicates. In the revised code, instead of branching on the false predicate, the compiler qualifies execution or both the check and the next compare on 11 N.. true predicate, The elimination of rivo branches means the elimination of 1\'.{ potential mispredictions, so that the savings is more than just two instructions.

Data Speculation

In ri COI'AM! .spcculaiion, a toad is moved earlier in *code* StNi LlenCe to compensate fur load latency, and a check k made to assure that an exception doesn't occur if it subsequenth,' turns out that the load was not taken. In data speculation, it load is mowil kr.fore, a store instruclion [hat might aIter Zh u vnenlory 10Chtion that is the scores Or the load. A subsequent check is made to L. that the load receives the proper memory value. To explain the mechanism, we use an example Taken from [I NI'at]a. Volume J.].

Consider ;he following program fragment:

```
 r8 [r4j = r:2          /I Cycle
    r6 = [r] ;;         // Cycle D
    rE - r    E7 ;r     // Cycle 2
st8 [r18] = r5          I/ Cycle 3
```

As written, the code requires four instruction cycles to execute. If registers r4 and r do not contain the same memory address, ;hen the. More l hrough r4 cannot affect the valLie **the eoniained in I'S; under this circumstance, it** is safe. to reorder the load and store to more quickly bring the value into r6, which is needed subsequently. However. because the. addresses ill r4 and rS may be the sarrie **or.over-**lap, such a swap is no Safe- IA-64 oNrcroomcs 1h is problc.m with the use of a technique known as advanced load.

```
      r6    (2' 31  :;      // Cycle -2 or eamlier:  advesiced load
                            //        insLrac=ions
st8 [r4] = r12              ii Cycle 0
      r6 = [raj             // Cycle Q. hec',c load
rac34 r5 = r    l I H       ././ Cycle 0
sL2 [r18: - 1-3             II cycle 1
```

lore we have nItTve.d the Id instruction earlier and converted it into an advanced load. In addition to performing the specified load. the ldS,a instruction writes its source. address (address contained in FS) to a hardware data structure known 2IS the Advanced Load Addrcss (ALAT). Each IA-64 store instruction checks the ALAT for entries that overlap with its target address; if 4i match is found, the ALAT entry is removed. When Ihe original ld8 is converied to an Ida inst ruction and **movcci,** the **igifl iI** po.sition of that instruction is replaced with a check load instruction, ldS.c. Wien the check load is executed. it checks the ALAT for a matching address. If one is found, no store instruction between the iidvanued load aud the. check load has iilkered the source address; of !lie load, and no action is taken. I lowever, if the etteLk load instrUction does not find t matchin2 ALAT entry, then the load operation is performed again to assure the correct TCfitilk,

We rhay also want tO spwola tiv4;[y cac.eutc. insLructio]is IhaL are data dependent on a load instruction, together with the load itself. Starting with the same original program, suppose we move LL both the load and the subsequent add instruction:

```
lda.a  r6                  // cycle -3 or earlier: advanced load
                              olner instrzictLons
ado; r5    r6, r7          // Cycle -L;  ade  that ki4e.:3 r0
                           // 07.er ir.ntr..LctLne.
std  [r.4] = rig            i cycle 0
       16, recover            Cycle 0; check
.L.Fac:11!                 i/          pn'=nt ffimr. jump t    recover
m_#; frnI  - rc               cycle 0
```

f lere we use a ehk.a instruction rai her ihan an ILI 3,C instruction to validate the advanced load. If the chk.a instruction determines that the load has failed, it cannot si mply recxecute the load: instead. it branches to a recovery routine to dean up:

```
Recover:
      1d8 r6 = [re:1 ;;        // reload r6 iron
      adn r5 =  r6,      :;    //              L1=  add
      br back                  // fump bac= . to main code
```

This technique is effective only if the loads and stores involved have. little chance of overlappin.

Software Pipelining

Consider the following loop:

```
1d4 r4 - :r5] , 4 ;;                2: load poeLncl 4
add 17 - r4,  rg           // cycle 2
     [r6 = r7,  4          i/ Cycle 3; store .costi.r.c 4
br.c-oop Ll ;;                Cyo.le
```

This loop adds a constant to one vector and stores the result in another vector (e.g, y[i] = x[i] I c), The .Ed4 instniclion loads 4 bytvs from memory. C he qualifier ", 4" at the end of the instruei itm signals that this k the base update form of the load instruction; the address in 5 is incremented by .4 atter the load takes place. Similarly. the st4 instruction stores four bytes in memory and the address in r6 is incremental by four 4(.21 t he More, 'Vlic hr.cluop inMruclion. known Lis a counted loop branch, uses the Loop Count (LC) application register. If the LC register is greater than zero, it is decremented and the branch is taken. The initial value in LC is the number 0f ilerations of the loop.

Notice that in this program, there is virtually no opportunity for instruction-level parallelism within a loop. Further, the instructions in iteration x are all executed beforc. iteration .v 1 begins. However, if there is no address eon flic1 between the load and store (r and poin1.10 nonoycrIvping muniory locarion.9_ then utilization could be improved by moving independent instructions from iteration x • I to iteration x. Another way of saying this is that if we unroll the loop code by iwtu;illy

writing out E new set of instructions for each ieratio± then there is opportunity to increase parallelism, Let's see what could be done with five iterations:

```
la4 r32 = [r5]. 4 ;r          1/ Cycle 0
          -     4 ;           /1 Cycle 1
Ld4 1 34  -     d             // ..7Arcle 2
add r36 = r32, r5 :             Cycle 2
    r:35 = Lx 2.1. 4          // Cycle ;
a     2 = r2s, r'             ZZ 7.2vcle
st4 [1 61 = f36, 4 ;r         /1 Cycle 3
la4 r36 = [r5]. 4             /I Cycle 3
add  r R=  r?4,                 Cycle 4
s=4 [r6 - L 37, 4 ;;          7/ Cycle 4
add L 39 = r35, r9            ZZ Cycle 5
           r 0. 4 t;            Cycle 5
add L4C. - r3 G,                Cycle G
         = r39. 4 ;;          /1 cycle 6
57,4 .r Ed = r.10, 4 ;;       /1 Cycle 7
```

'thi program compIL:tes 5 iterations in 7 cycles, compared with 20 cycies in the original iooped prOgrarn, This assunii,:s 1h.at there ıWILF memory ports o that a load and a store can be. executed in parallel. This is an example. of software pipelining, inpli gcriii; to hardware pipelining. Figure 15.6 illustrates the process. Parallelism is achicycd grouping toptlwrinsirtictions from differen1 iterations. For this to work, the temporary registers used imide the loop MLA(Ile c hriged foreach iterEition Lc] avoid register conflicts. In this case, two temporary registers are used fr4 anti r7 in the origimil program), in the e.xpanded program, the regiger numher of each

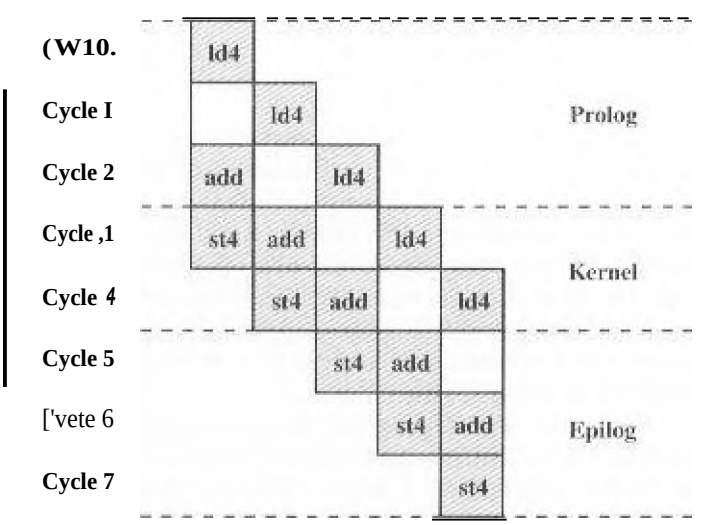


Figure 15-6 .cifilwarc Pipelining Exnrnplc

register is incremented For each iteration, and the register numbers are initialized sufticierdly far apart tO avoid overlap.

Figure 15.6 shows that the software pipeline has three phases. During the **prolog phase, a** new iteration is initiated with each Clock cycle and the pipeline gradually fills up. During the **kernel phase,** the pipeline is full. achieving maximum parallelism, **For our example,** three instructions are performed in parallel during the kernel phase, but the width of the pipeline is four. During the **epilog phase_** one iteration completes with each clock cycle.

Software pipelining by loop unrolling places a burden on the compiler or programmer to assign register names properly. Further, for long loops with many iterations, the unrolling results in a significant expansion in code size. For an indeterminate loop (total iterations unknown at compile time), the **task is** further complicated by the need to do a partial **unroll** and then to control the loop count. IA-64 provides hardware support to perform software pipelining with no code expansion and with minimal burden on the compiler. The key features **that** support software pipelining are as follows:

- **Automatic register renaming:** A fixed-sized **area of the** predicate and floating-point register files (p16 to p63: fr32 to **frI27) and a** programmable-sized area of the general register file (maximum range or r32 to r127) are capable of rotation. This means that during each iteration of a software-pipeline loop, register references within these ranges are automatically incremented..Thus. if a loop makes use of general register r32 on the **first iteration, it** automatically makes use of r33 on the second iteration, and so on,
- **Predication: Each instruction in the loop is predicated on a rotating** predicate register. The purpose of this is to determine whether **the** pipeline **is in prolog,** kernel, or epilog **phase,** as explained subsequently.
- **Special loop terminating instructions:** These are.branch instructions that cause the registers to rotate and the loop count to decrement.

This is a relatively complex topic; here, we present an example that illustrates some of the IA-64 software pipelining capabilities. We hake the original loop program from this section and show how to program it for software pipelining, assuming a loop count of 2(X) and that there are Iwo memory ports:

```
        mcn'     - 199          /i      lop cryJnr, regisrer ro 19 9,
                                // WI1lc ecual6 loop corn:: - 1
         no ec = 4              // Set epilog co-1n`_ regiFt:.er
                                // 7.o number of epilog 6LageE    1
        mov          -1-:‹16;; I/ oriS - 1; rest -
        1..f..4 r32 - :rS: ₈ 4  // Cycle 0
 (p17)                             Empty stage
 (p18) add 173!,    r34, r9        Cycle
Up19   s=4  `r6: = z 36, 4      f,! Cycle 0
       br.etc; LL :;            11 Cycle C!
```

We summarize the key points related to this program:

1. The loop body is partitioned into multiple *Wages,* with zero or more instructions per stage.
2. Execution of the loop proceeds through three phases, During the prolog phase, a new loop iteration is started each time around. adding one stage to the pipeline. During the kernel phase, one loop iteration is started and one completed each time around; the pipeline is full, with the maximum number of stages active. During the epilog phase. no new iterations are started and one iteration is completed each time around. draining the software pipeline.
3. A predicate is assigned to each stage to control the activation of the instructions in that stage. During the prolog phase. pi() is true and p17. p1S. and p19 are false for the first iteration. For the second iteration, p16 and p17 are Inlet during the third iteration pi6, p17, and p18 are true. During the kernel phase. all predicates are true. During the epilog phase, the predicates are turned to false one by one. beginning with p16. The changes in predicate values are achieved by predicale register rotation,
4. All general registers with register numbers greater than 31 are rotated with each iteration. Registers arc rotated toward larger register numbers in a wrap-around fashion. For example, the value in registers will be located in register + 1 after one rotation: this is achieved not by moving values but by hardware renaming of registers, Thus. in our example, the value that the load writes in r32 is read by the add t**wo** iterations (and two rotations) later as r34. Similarly, the value that the add writes in r35 is read by the store one iteration later als 06.

Table 15.4 Loop Trace for Stiftwarc Pip-dining Example

Cycle	Execution Unitilastrudion			State Whine br.ctop					
			riri	B	P16	P17 1 P18	P1.9	LC	EC
tl	l4l4			bc.ccop	1				
l	1d4			hr.ctop	1	1	0	198	4
	1d4	add		hi clop	1	1	1	197	4
3	1d4	add	st4	hr.acip	1	1	1	196	4
		• •	• • •	Mobil	• • •	• • •	• • •	• • •	• • •	• • •
100	1d.4	add	st4	hr.ctop	1	1	1	99	4
••	P- •	1 ••	••	• ••	• • •	• • •	• • •	• • •	• • •	• • •
	1114	arid	st4	1-lr.cl op	1	1	1	1	0	4
21X1		add	st4	hr.ctop	ti	1	1	1	0	3
201		add	sl 1	br.ctup	9	0	1	1	0	2
2(12			st4	brxtop	0	fi	11	1	0	1
					0	0	0	0	0	0

5. For the br.ctop instruction, the branch is taken if either LC > 0 or EC > 1. Execution of br.ctop has the following additional effects: if LC > 0. then LC is decrennerned; this happens during the prolog and kernel phases. If i.c = and EC > 1, ..1[2].0 i decremented; this happens during the epilog phase. 'The instruction also control register rotation. lf LC > 0. each execution of br.ctop places a l in p63. With rotation, p63 becomes pie). Feeding a continuous sequence of ones role} the predicate resisters during the prolog and kernel phases. If LC = 0, then hr.c1op sets p63 to O. feeding zeros into the predicate registers during the epilog phase.

Table 15.4 shows a trace of 1he execution of this example,

15. IA_64 INSTRUCTION SET AREHITECTURi

,Figure 15.7 shows the set cat 342;.Nters available to application programs. That is these registers are visible to applications and maybe read and, in most coxes, written. The register sets include the following:

- **General reOsters:** 12 general-purpose M-hil registers. Associated with each register is a NaT bit used to track deferred speculative explained in Section 13.3. Registers r0 through r31 are referred to at, t,hltiC; program reference to any of these references is literally interpreted. Registers' r32 through r127 can be used as rotating registers for soft 'arc pipelining (discussed in Section 1.5.3) and for register stack implementation (discussed subsequently in this section). References to these registers are virtual, and the hardware my perform register renaming dynamically.
- **Floating_point registers:** 128 82-bit registers for floating-point numbers. This size is sufficient to hold IEEE:. 754 double extended format numbers (see Table. 9,3). Registers fr0 through fr3 I ;ire static, and registers fr32 through fr127 can be used as rotating registers for software pipelining.
- **Predicate registers: Cam# i_bit** registers used as predicates. Register pro is always set to 1 to enable unpredicated instructions. Registers prO through pr13 are static, and registers pc16 through pri63 can he used ws rotating registers for software pipelining.
- **Branch registerx:** 8 64-bit registers used for branches.
- **Instruction pointer:** Holds the bundle address of the currently executing IA-64 instruction.
- **Current frame marker: Holds** state information relating to the current general register stack frame and rotation information for fr and pr registers.
- **User mask:** A set of single-bit values used for alignment traps. performance monitors, **and** to monitor floating-point register usage.
- **Performance monitor data registers:** Used to support perforrimncc. monitor

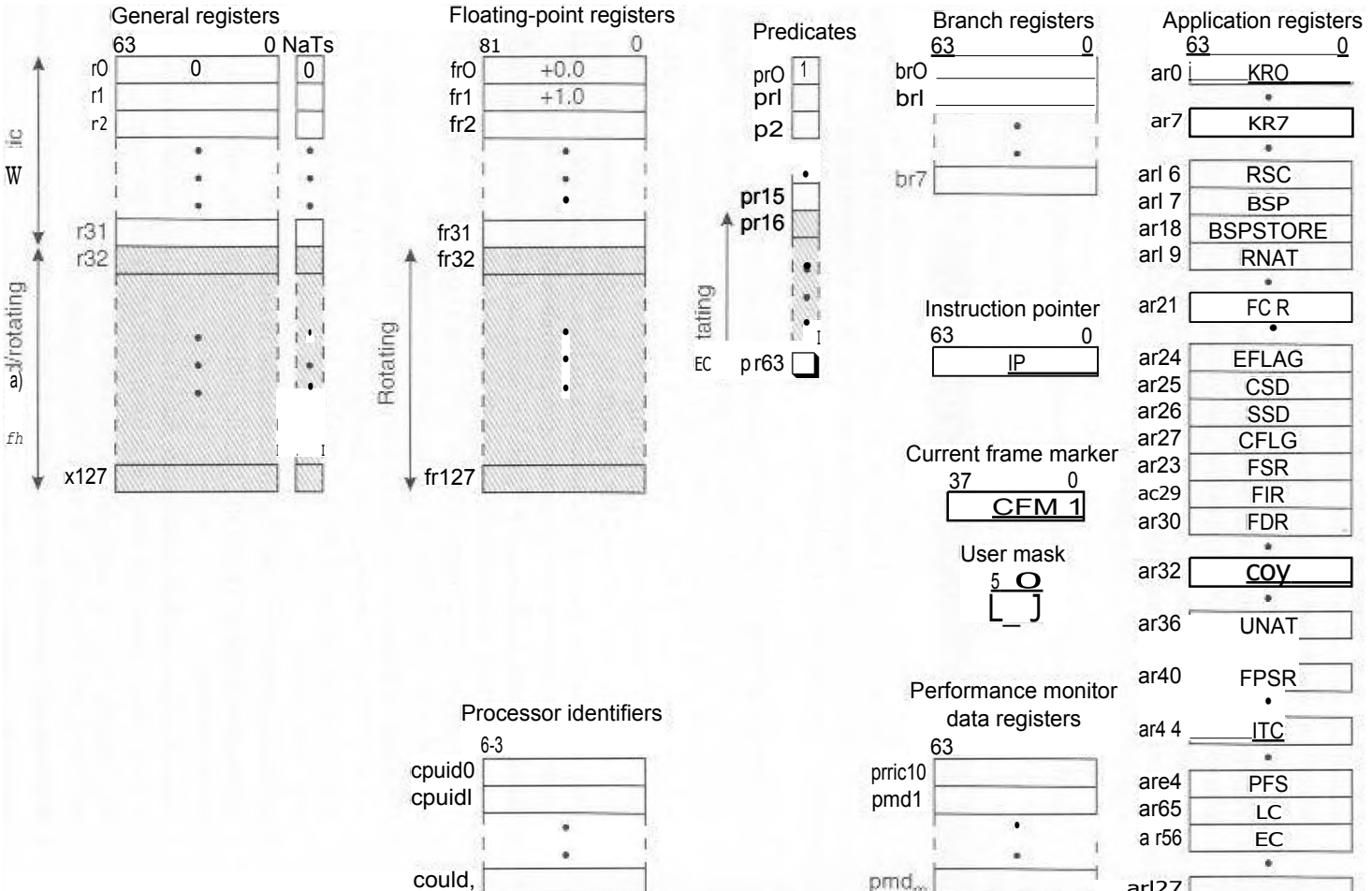


Figure 15.7 IA-64 Amlic-alion Thug's= Sot

- **Processor identifiers.: 1)Gscribc** Farmessor implurneiiiaiiort-daptndunL features.
- **Application registers:** A collection of special-purpose registers. Table 15.5 provides 2l brief definiWon of each.

Register Stack

1 he register stack mechanism in IA-64 avoids unnewssary movement of data inio and out of registers at procedure eal I return. The nreehani SIT) autornaticalh,' provides a called procedure with a new frame. of up to 96 registers (r32 through.r127) upon procedure entry. The compiler specifies the number of registers required IT Y procedure with the ailtie instruci **ion, whidi s.peViCiefi** helW mariv of these girt; local (used only within the procedure) and how **MAW,'** are output (used to pass parameters Io a procedure called by this procedure). When a procedure ca]] occurs. the I A-64 hardwme rt.:names registers so that the local registers; from the previous frame are hidden and what wcre the output registers of the calling procedure now have register numbers starting at r32 in the called procedure. Physical registers in the r32 through r 127 are allocated in a circular-buffer fashion to virtual registers

fable 15–C I.A-6-4 Application Registe rs

Kurill21 ruil1o.1217 } l< R 1)-7)	Conveyinformari on from the operating sysl cm to the application,
Register stack configuration (ESC)	Controls the operation of the:register stack engine (RSE).
RSE 13ackin store prim ler (FISP)	lioldri the address in memory that is the r,LIFI2 Ii3e2Lion iw r32 ill the current stack. frame
RSE Backing store poinler to millTIOry slimes (IISPSTORE)	H *oids.* the add rc cs in enennon) Lci which 11.1.: RSE will ,;pii3 ilw nest vriluc.
R.SE NaT coilect i on rcgtsler (AN A'l)	Used by Elie R h... w r cm poraril,v hotel Nai hits when it 1.-. spilling general registcrs.
Cprn parc and exchange. value (CCV)	Contains the compare value used as the third sourt:.. operand in the cgiprie1]s instruction.
User **Karr** collection re Oster (UN AT)	Fixed to temporarily hob.] NaT bits wheri.s.aviii:J. ;old **TtiN L{ItiDg g1.11{:Tai 3' gB** LOU!, **Yr** iih th **e idg.rii]** and AN !.i:ill illSEructium;.
FloaLin-poinl. gaiIiS Te.giter (Fps **R)**	**CaniIDIa tTO1zi,** CC1U ndin F mode., pf6i2isic }11 1.1331 mil. 1100., kind othel cOnircil hits for floatirig-pOini. inStr> CtiOnA.
Iniervak time CCIU rilxr **OTC)**	C:ounts up irl 2l fixed ri2 !at i unsh i p to the **p1131:EXSOT 1:101±** frequency,
Prc}iou's functi on stale (F.ES)	Saves value in CE1y1 rcgier and relatcd information-
Loop count (LC)	Used in counted loops and is clecremented by counted' loop• type branches.
Epilog count (EC)	Lid for couniin the final (epilo[?.) stMe in Modulo-seiseduld loops.

associated with procedures. That is. the next register allocaled afLer ↑ 127 is r32. When necessary, ihe hardware moves register contQnls hel wean registers and memory to free up 2iddiiional registers when procedure calls occur, and restores consents rrorn memory Lo registers as procedure returns occur.

Figure 15.8 illustrates register stack behavior, Pic11(.14 2 insiruetion includes sof {size of frame) and sol (size of locals) operands lo specify the required number trf registcrs. 'These I ue.!.4 are stored in the (TM register. When a cal] occurs, the sel and sof values from the CFM are stored in the, soi and sof fieids of the previous function state (PFS) application register (Figure 15.9)- Upon return these so] and sof values must be restored from th4,:. !he CPM. To a nested calls and returns, vreivious valUe:S of the PFS fields niust be saved through successive calls so th4it they can be. restored through successive returns. This is a function of 1.111 2 411 hoc instruc• lion, which designates a general register 10 save *the current volue* of the PFS fields hcforc they Lire ovcrwritten from the CFM fields.

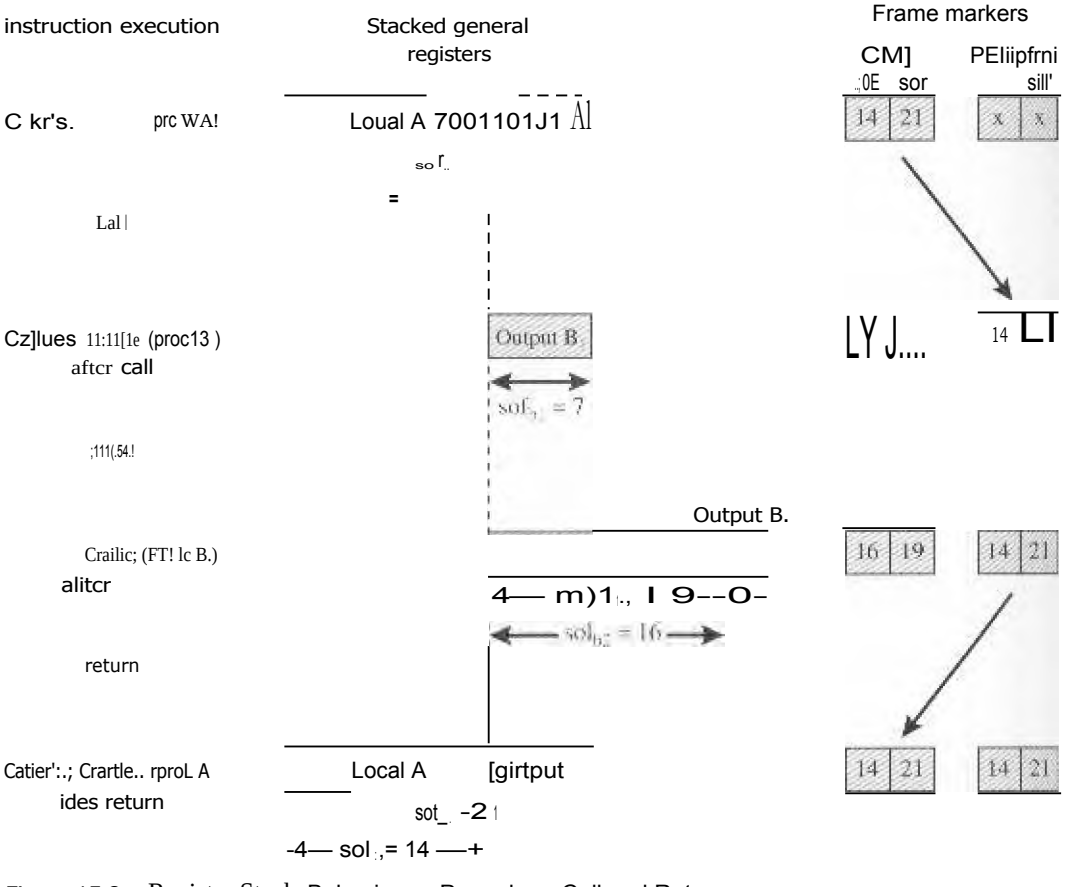


Figure 15.8 Register Stack Behavior on Procedure. Call and Return

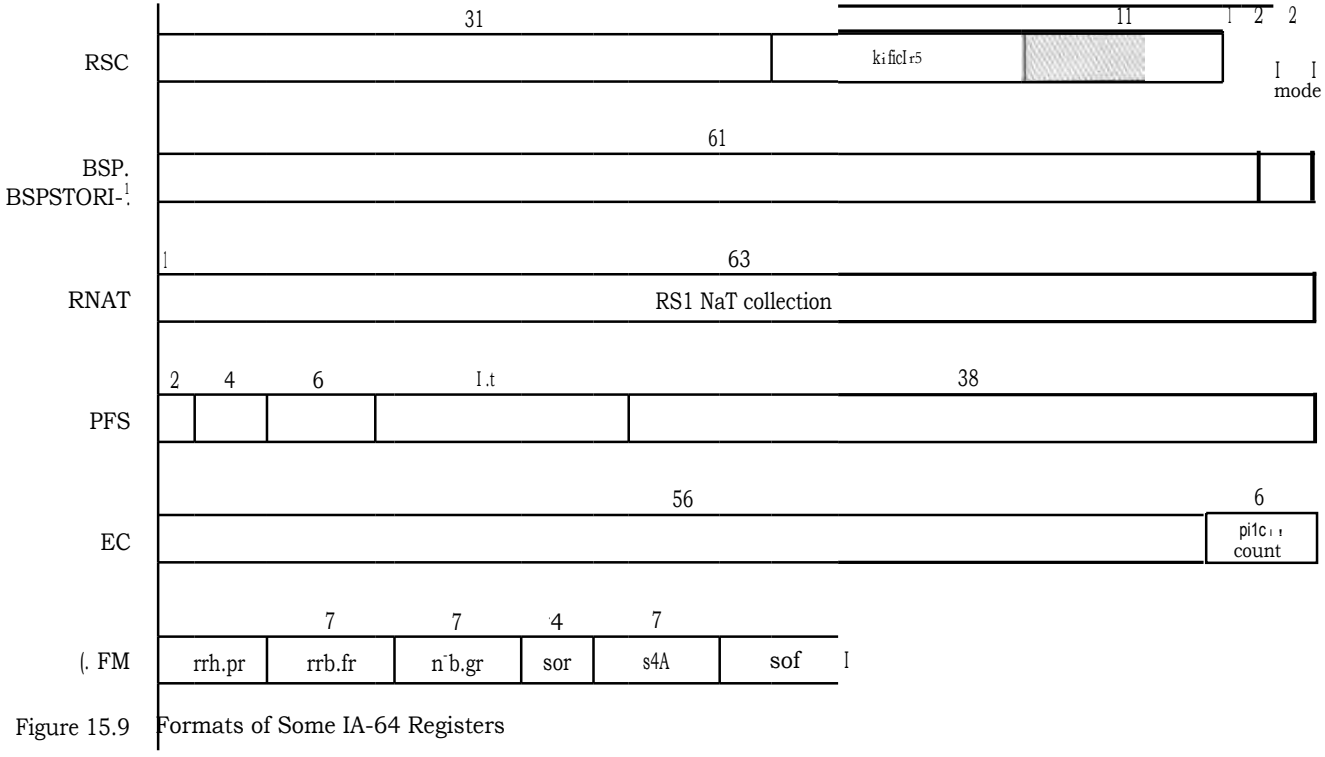


Figure 15.9 Formats of Some IA-64 Registers

Current Frame Marker and Previous Function State

The CFM register describes the state of the current general register stack fr rne. associated with the currenay active. procedure. It includes the following fields:

- 1.01'; Size of slack frame
- soh Size of Locals portion or stack frame
- son size of roiAting portion of stack frame; this is a subset of the local portion that is dedicated Lo s.oftware pipelining
- **register rename h e values:** Values used in perfomin$_g$ register rotation gen. erat, floating-point and predicate registers

The PFS register contains the following fields:

- pfm: Previous tramc nlarki,:r.: of Ihc fields of the cFro
- pee: Previous epilog count
- **ppl:** Previous privilege level

15.5 rrArsllum aftGANizATIoN

Intel'; Z Lardurn processor is Ihe first implementai ion of the IA-454 instruelion set architecture, The Itaniurn organi?.ation blends superscIlar features with support for the unique EPIC.-related IA-64 features. Among the t'uperscalar features are a six-wide, ren.....%tagc-deep hardware pipeline, dynamic prefetch, branch prediction. and a register scoreboard to oplirniv.c for compile lime nondelerminitim. EPIC'-related hardware includes support for predicated execution_ control arid data speculation, and software pipelining.

Figure k a general Nock diagram of the I tanium organization. 'The Ito• nium includes nine execution units: Lwo integer, iwo 1104iting-point, Iwo inernery, and three branch execution units. Instructions are retched through an Ll instruction cache ;Ind fed into a buffer that holds up to eight bundles of instructions. When deciding.on funciiorra I uni Es For instruction dispersal, the processor views al mast Iwo instruction bundles at a time. 'rke. processor can issue a maximum of six insirtic• Li ons per clock cycle,

the orpiniwai ion is in some ways simpler than a conventional contemporary superscalar Itanium does not use reservalion reorder buffers. and memory ordetirt2 buffers, all replaced by simpler hardware for specu-lation. The register remapping hardware is simpler than the register aiiasing typical of superscalar machines. Register dep endency-detection logic is absent, repla ced by explicit parallelism directives prccoinputed by LH software.

Using branch prediction. the fetchlprefetc]i engine can speculatively load an Ll instruction cache= 10 minimize cache misses on instruction fetches. The fetched code is fed into a decoupling buffer [hat can hold LE F: to eight bundles of code.

Three levels of cache are used. The LI cache is split into a 16-kbyto instruc-tion cache and a lb-]chute data cache, each 4-way set associative with a 32-byte

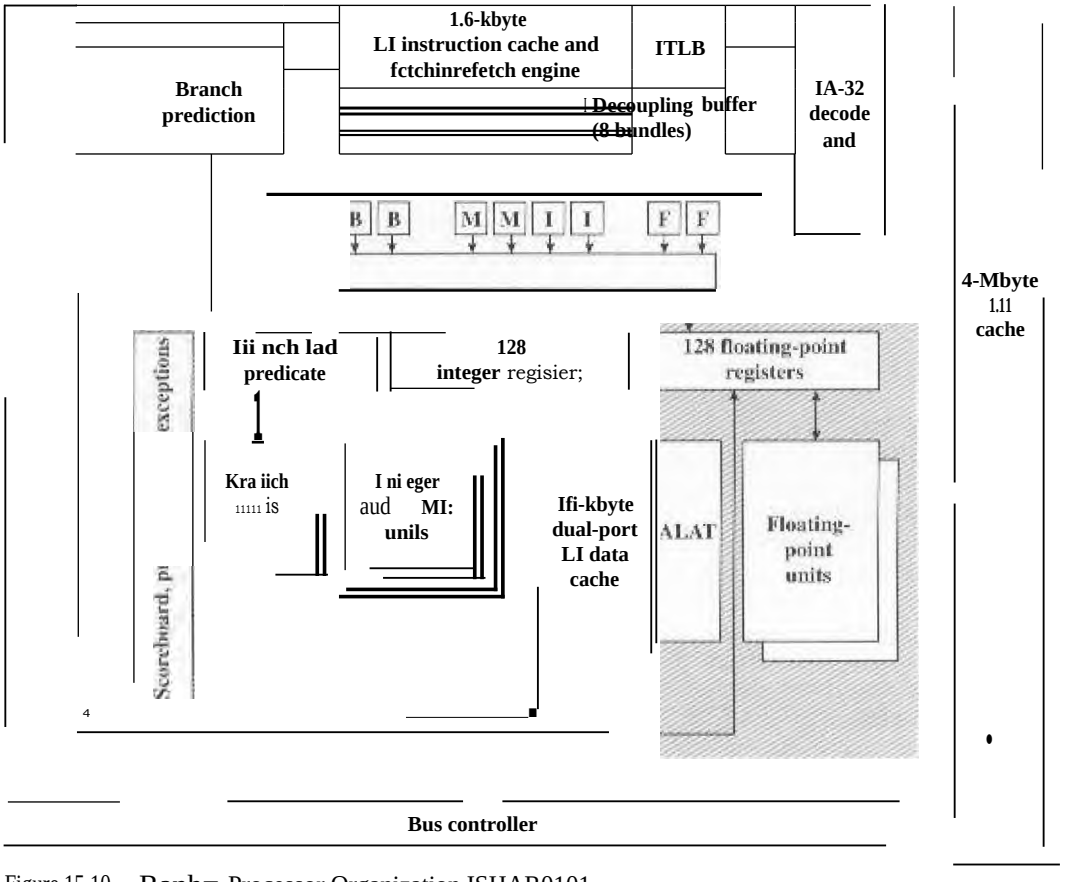


Figure 15.10 Ranh= Processor Organization ISHAR0101

]inc size, The 96 kbyte L2 cache is 6-way set associative with a 64-hyte line sire, [he 4 Mhyte L3 cache is 4 way set associative with a 64 byte line size. The LI and I. caches are on the processor chip: the L3 cache is off-chip hut on the same package as the processor.

15.6 RECOMMENDED READING AND WEB SUES

ifill:CKCX71 provides an overview of IA-64; another overview is IDUL098I. [SCHLMa] provides a general discussion of EPIC; a more thorough treatment is provided in [SCHLOOb]. Two other good treatments are 111WLI011 and IKATHOl [CHASM and [I IWI.:98] provide introductions to predicated execution. Volume 1 of IINTEMal contains a detailed treatment of software pipelining; two articles that provide a good explanation of the topic. with examples. arc VARP01] and IBHARi-1U!.

For an overview of the hani um prueessor architecture, see [SH.A kW]; INTEMbi provides a more detailed treatment.

Both rrRIEall and [MARKOOI contain more detailed treatments of the topics of this chapter. Finally, for an exhaustive look at the TA-64 architecture and instruction set. see

BIIAROO Bharandwaj. J.. et al. "The Intel IA.64 Compiler Code Generator:' *IEEE, Micro.* **ScptemberlOctober 2000.**

CHASOO Chasin. A. "Predication. Speculation. and Modern CPUs.' *Dr. Dalthrs Journal, May 2000.*

DLL0911 Dulong, C. "The IA-CA Architecture at Work." *Compurer,* **July 1918.**

HUCIi00 Huck, ,.1 al. 'introducing the IA-64 Architecture." *IEEE Micro.* **Septum' her: October** avu.

HWU98 Hwu, W. "introduction to Predicated .F.xecut i on " *Computer, January* **1998.**

HINUO1 Hwu, W.: August, 0.; and Sias. J. "Program Decision Logic Optimization Using Predication and Control Speculation: *Pro(re.!eleiNS of The !FEE.* **November 2001.**

INTE00a Intel Corp. *Iad IA-0 A reirlirYiNOT SuPwai beS elopery Manual r4 vr}Inotr'sl.* **Document 245317 throuvIt 2.46A2ti. Aurora, CO, 2000.**

1NTRO0b Intel Corp. *Thulium Pri.IreWSirF M ic' r (tarn* **Ikkrence Pr** *Software 00 mizothm.* **Aurora, CO, Document 2447:1_ August 2000,**

ARPII1 Jarp, S. "Optimizing IA-64 Pei tormance.." bobb's *few, no& July 2001.*

14: Arm KatImil. Sehlansker. M.: and Rau, B. "Compiling for F.PIC Architectures," *P r in.'s Mr MEE, Now.mber 2001.*

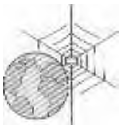
V1.411100 \lark stein, P. IA-64 *and Elementary Fatu'rioor. Upper* **Saddle River, KJ: Prop 1.:r• 11:;11 r I R. 2000.**

SCHLOOn Schlansker, •.; and Rau, B. "EPIC: F.xpliciLly Parallel Instruction Computing." *Computer,* **February 2000.**

SCIILAIOb Schtansker. M.: and Rau, B. *E .;;I n rirchnecUrre for if:Striorrif»t-Levri' Parallel Processors,* **HPL Technical Report 11PL-1999- II. lIewlell-Packard laboratories (www.hpl.hp:corn). February 2000.**

SRA ROO Sharangpani...11. and Arona, K. "lianiuni Processor Microarchitecture." *IEEE !Wien.),* **SepLe.mheroctub.a** 201.n.

TRIE01 Triebel, *Iranian? Ari,thilecuire for Software DeVelopers,* **Intel Press, 2001,**



ROCOITITrteridCil Vileh

- **Itanium:Inters site for the latest information on IA-64 and Itanium.**
- **IMPACT: This is a site at the University of Illinois, where. much of the research on predicated execution has been done, A number of papers on the subject arc available;**

15.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

ad *van* ced load	**execution unit**	**instruction completer**
branch predication	**explicitly parallel instruction computin (EPIC)**	**instruction go 'up**
bundle	**hoist**	**Itaniurn**
control speculation	**IA-64 architecture**	**major opcode**
datitspoeulation		**Na" I' bit**

- predicate register
- predication
- register stack
- software pipeline
- speculative loading
- Slack frame
- stop
- ff{..t1krble.
- template field
- very long instruction word

Review Questions

15,1 What are the different types of execution units for IA-64?

15.Z. Explain the use of the template field in ;in IA-6.1 bundle.

15.3 What is the significance of a stop in the instruction stream?

15A Define predication and predicated execution.

15.5 How can predicates replace a conditional branch insirtiction?

L5.6 Define con1rol speculation.

15.7 'INhat is the purpose of the NaT bit?

15.8 Define data speculation.

15.9 What is the difference between a hardware pipeline and a software pipeline?

15.10 Virrhal is the difference between stacked and rotating regrsters?

Problems

15,1 Suppose that an IA-64 opcodc accepts ihrec registers as operands and produces one register as a result. What is the maximum number of such opcodes that can he defined in One major opcode [arni]y?

15.2 At a certain point irk an IA-64 program, Ihere are LO A •typo instructions and six floaling-point instructions. Ihat can be issued concurrently. How many syllables !nay appear without any stops hetwcun them?

15.3 In Problem 1 s.2,

a. How man's.. cycles am required for a small LAM. impierneniation haying one floating-point urkii, two integer units. and Iwo memory units?

b. How many cycles are required for the Itanium organisation of Figure 15.10?

15.E An algorith in ChM can utilize four floating-point instructions per ma lt ' is coded for IA-64. Should instruction groups contain four limning-point ()reran.' in••••? What are the consequences if the machine on which the rypigram runs has fewer four floating-point units?

15.5 In Section 15.3, we introdueed the following constructs for predicated execution;

p2, D3 = a, h

ODL) croD,CL.A 2, .33 = a, h

where. orel is a relation, such as eq, ne. etc.: p1. p2. and p3 are predicate registers; a is either a register or an immediate operand; and h is a register operand.

Fill the following truth table;

	comparimm	p2	
not present	1		
not prescnI.			
0	0		
0			
	0		

I . For the predicated program ASection 15.3, which iinpleirieJits the flowchart of Fig ' urc L5.4. indicate

41, Those hum ructions [hot can he eNecuted in. parallel

h. Those instructions that can he bundled into the same 1A-0 '1 instruction bundle

15-7 Consider Ihe following suurce cude scgmcnt:

```
for ( i  DJ     c 101:.;        )
     -t  (A[ij <       )
          j = j -  14.
```

= -

WTI.% •a COMSrondinigl 'entium assembly code segment.

h. Rewrite as an /A4.r4 assembly code segmcnt using predicated execution technive,,.

15.8 Consider the following C program fragment dealing with float a 211.1:111 l

a IL:

c *j*

The compiler cannot establish thal i i. hut has rcasoli that it probably

H. Write an 1A-64 proform Limn!! :Irl ;Rlv,incze.1 toad to impleinClit this *C* program. rhv tl oal A m m1aGm1 **mm(iy mgeG is are** r iid Innpy, respectively.

h. Recode the program using predication instead of the advanced load.

c. What LLTC the advantages and disadvanlages. of [he two approache4. compared with cach other'?

153 .60.9ume: th;41 a si nk. registi,:r firaim k created with si7e eq altoSOF= 48. If the size or the. [Ewa! register siroitip is SOf 16;

a. How many output registers (SOO) are there?

b. Which registers are in the local and output repister groups'?

The Control Unit

In Part Three., we focused on Inaeh'inee instructions and thc. opmmion:s performed by the processor to execute each instruction, Vit'hat was lc it out of iS exactly 1101V eitCh individual operation is caused to happen. This is the job of the coin rol

The controi unit is tEurt portion of the procesor that actuatfy cattwes things to happen. The control twit issueN controi signals external ,to the proccsor to cause data exchange with me3nory and 110 modules. The control unit also issues control signak inIvrnal to the processor lo move data between registers, to cause the ALU to pert(inn a. specified function, and to regulate. other internal operations. Input to the control unit consists of the instruction register, kags, and control signals from external sources (e.g., interrupt signals).

Chapter 16 Control Unit Operation

In Chapter I 6, We (UM to a discussion of how processor functions are perburied or, more specifically; how the widow.; elements 0r the processor are controlled to provide these functions, by means of the control unit. It is shown 1ha t each insErucl ion cycle is made up of a set of micro-operations that generate control signals. Exeetil ion is accomplished by the effect of these control signals. emanating from the control unit to the AFAT, nd system interconnection structure. Finally, an approach to the implementation of the conttol unit, referred to as hardwired implementation, is presented.

Chapter 17 Microprogramined Control

Chaptcr 17. }1/4'(;: KC. /K-tom cOnCepl of mice qt lea.(L to an elegant powerful approach to control unit impicmentation, known SE. microprogramming, En es6ence, lk pwer-level programming lan2tiage is dveloped, Each instruction in the machine 1.4inguage of the procesma traos»Led ito a scttacnce of contfol writ instructions. These Joi,ver-lcvel instructions aye referred to as micro-instruction2i, and the process of translation is referral to as microprogramming. 'Hit! chaptcr (lc:scribes die layout of a conlrol memory conLaiMng n vriklrprogrAii, [en each machino instruclion is dc..e.ri bed. 'rho sEructurt and function of the micro-programmedcontrol unit CUD then I:Fe explained,

CHAPTER 16

CONTROL UNIT OPERATION

16.1 Miero-Operations

The Fetch Cycle
The Indirect Cycle
The Intel-mil Cycle
The Execute Cycle
The instruction Cycle

16.2 Control of the Processor

Functional Requirements
Con trol Signals
A Coolrol Signals Example
Internal Proces..sor Organization
The lintel :308.5

16.3 Hardwired implementation .

Control Unit !FINN
Control Unit Logic

16.4 Recommended Reading

16.5 Key Terms. Review Questions. and Problems

Key Terms
Review Questions
Problems

KEY. POINTS

* The execution of an instruction involves the execution of a sequence 01' sub-cps, generally called cycles, Fur example, au execution ma!• ,' consist of fetch, indirect, execute, Intl inlerrupt cycles. Each cycle is in turn made up of a so.itience of more. fundamental operations. called iniero **-Operations.** A 8i4,l12 micro-t Fperation generally involves a transfer between registers. a transfer between lk register and an external bus. **or** a simple ALL' operation.
- The control unit of a proce ssor performs two ia!,ks: l) It causes the ffocessor to execute micro-operations in the ' proper sequence, determined by the program being executed, and f2 it generates the control signals that cause each micro-operation to be executed.
- The control signals generated by the control unit cause the opening and closing of logic gates, resulting in the transfer of dal a in and from regiSters and the operation of the ALL
- One teehnique for implementing a control unit is referred to ax hardwired implementation, in which the controt min is a ccmtioatcrisl eircuii. Jis input logic signals. governed by the current instruction. arc transferred into a set of oulput control signals,

In _'h;ipter 10, we pointed out **(hal z.J. mLtchinc** instruelion set goes a Long way l 0...'..;ir,1 defining the processor. if we know the machine instruction set, including an understanding of the effect of each opcode and an understanding of the iL.Ii.lressing modes, and if we know the 5ei of user-vi5d)le registers. i hen **we krKrk** thc functions that the processor must perform. This is not the complete picture. We must know the external interfaces, usually through a bus. and how interrupts are handled. With this line 0r reasoning, the following list of those things needed trP specify the function of a processor emerges!'

L Operations'. (orodes.)
2. Adire.ssing modes
3. Registers
4. 110 module interface
5. Memor,, module
6. Interrupt processing structure

This list, though general. is rather complete. Items 1 through 3 are defined by the instruction Set. I teni ', 4 11".1 5 are typically defined by sped Lying the system bus, Item 6 is defined partially by the system hus **and p;irlinlEv** by the type of support processor offers to the operating system.

This list or six iLfm-m mighi be termed the functional requirements for a processor. They determine what **a proulssor** must do, This is what occupied us in Part

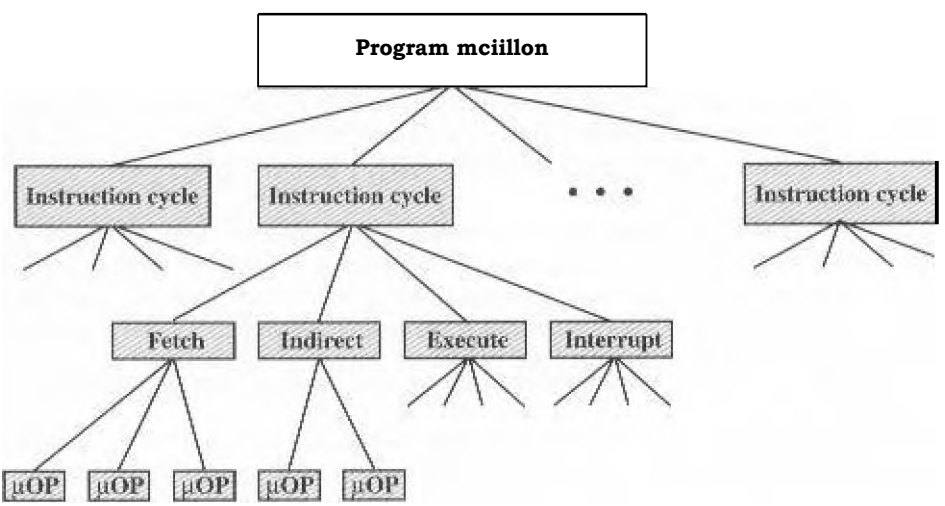
Two and Three. Now. we turn to the cluei ion of how these functions are performed or, more specifically. how the various elements of the processor are controlled to provide thcse functions. Thus, we turn to a discussion of the control unit, which controls the operation of the processor.

16.1 MICRO_OPERATIONS

\V have seen that I ht. operation of a computer, in executing program. consists of ŋ sequc..nce of instruction cycles, with one machine instruction per cycle. Of course, we must remember that this sequence of instruction cycles is not necessarily the same as the *written sequence* o.1 irv;tructions that make up the program, because of the existence 0ŗ branching instructions. What we are referring to here is the execution *time sequence* of instructions.

'Ve have further seen that each instruction cycle is made up of a number of smaller units. One 111-itli vision that we found convenient is fcieiL indirect, execute, and interrupt, with only fetch and execute cycles always occurring.

To design a control unit. however, we need to break down the description further. In our discussion or pipelining in Chapter 12, we began Lo sec that a further decomposition k possible. ln fact, we will see that each of the smaller cycles involves a series of steps , each of which involves the processor registers. We will refer lo these steps as *micro_operation..'1* Felix *nrinro* refers to the fad that each stela in very simple and accompiishes vary tithe. Figure MI depicts I he relationship among the various concepts we have been discussing, To sumrnalie.e, Lhc execution of a program consists of the sequential execution of instructions. Each instruction is excculed during an insi ruction cycle made. up of shorter subcycles **fetch, indirect,**



Ilion.. 16.1 Constituent Elements of a Program ExtCLI(kul

execute. interrupt). The performance of each suhcycle involves one or more shorici operations, that is, micro-operations.

Micro-operations are the functional, or atomic. operations of a processor, In this section, we will examine micro-operations to gain an understanding of how the events of any instruction cycle can be described as a sequence of such micro operations. A simple example will he used. In the remainder of this chapter, we the!' show how the concept of micro-operations serves as a guide to the design of the control unit.

The Fetch Cycle

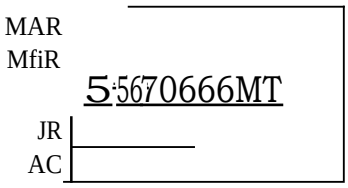
We begin by looking at the fetch cycle, which occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory, For purposes of discussion, we assume the organization depicted in Figure 1.2.6. Four registers are involved:

- Memory address register (MAR): Is connected to the address lines of the system bus. It specifies the address in memory I'nr a read or write operation.
- Memory buffer register (MR): Is connected to the data lines of the system bus. 11 contains the value to be stored in memory or the last value read from memory.
- **Pregnant** counter (PC): Holds the **address** of the next instruction to be fetched.
- Instruction register (R): Holds the last instruction fetched.

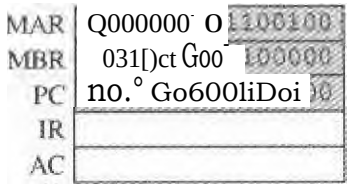
Let us look at the sequence of events for the fetch cycle from the point of view of its effect on the processor registers. An example appears in Figure [6.2. At the beginning of the fetch cycle. the address of the next instruction to he executed is in the program counter (PC); in this case. the address is 1100100, The first step is to move that addrc.•.ss to the memory address register (MAR) because this is the only register connected to the address lines of the system bus. The second step is to bring in the instruction. The desired address (in the MAR) is placed on the address bus, the control unit issues a READ command on the control bus. and the result appears on the data bus and is copied into the memory buffer register (MBR We also need to increment the PC by 1 to get ready for the next instruction. Because these two actions (read word from memory. add 1 to PC.) do not interfere with each other, we can do them simultaneously to save. time, The third step is to move the contents of the MB R to the instruction register (1R). This frees up the vIBl(for use during a possible indirect cycle,

Thus, the simple fetch cycle actually consists of three steps and four micro. operations. Hach micro-operation involves the movement. of data into or out of a register. So long as these movements do not interfere with one another, several of them can take place during one step, saving lime. Symbolically, we can write this sequence of events as follows:

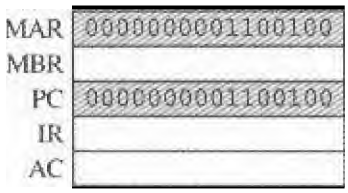
(← (PC:.
MER Memory.
PC (PC) +
4-- (MBR)



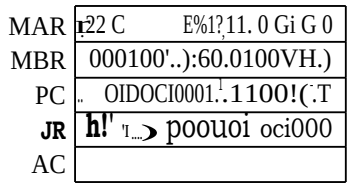
(a) Beginning



(c) Second step



(h) First step



(d) Third step

Figure 16.2 Sequence of Events. Fetch

where I is the instruction length. Wc need 10 make several Comments about this sequence. Wc assume that a clock is available for timing purposes and tho emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro operalion cxn he performed within the time of a single time unit. The notation (L , t,, t,) represents successive time units. Iii words, we have

- **First time unit** Move contents of *PC* to MAR.
- Second **time** unit: Move contents of memory location Teeificd by MAR to MBR. **Increment** by I the contents of the PC.
- **Third time unit** Move contents of MHR I R.

Note that the second and third micro-operations bosh take place during the second time unit. The third micro-operation could have been grOuped with the twirl h without affecting the (etch opera lion:

t. (PC
L_n **i•MR.** <— Memory
PC (— T
ZR ; 2.1ER

The groupings of micro-operations must follow two simple rules:

1. The proper sequence of events must be raovecd. Thus (MAR (PC)) must precede. MBR e Memory) hoc iuse the memory read operation makes use of the inidecs in the MAR.

2. Conflicts must he avoided. One should not attempt to read to and write front the same register in one time unit. because the results would be unpredictable. For example. the micro-operations (MBR <— Memory) and (IR MRR1 should not occur during the same time unit.

A final point worth noting is that one of the micro-operations involves an addi• tion. To avoid duplication of circuitry, this addition could be performed by the ALL. The use of the ALZJ may involve additional micro-operations, depending on the functionality of the ALL' and the organization of the processor. We defer a dis• cussion of this point until later in this chapter_

II is useful to compare events described in this and the following subsections to Figure 3.5, Whereas micro-operations are ignored in that figure, this diseussion shows the micro-operations needed to perform the subcycles of the instruction cycle.

The Indirect Cycle

Once an instruction is fetched, the next step is to fetch source operands. Continu- ing our simple example. let us assume a one-address instruction format, with direct and indirect addressing allowed. If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle. The data flow differs somewhat from that indicated in Figure 12.7 and includes the following micro-operations:

t: MAR (— (IR iiVi(iress)
; <— Memory
TP.1.7iddres,$) CAER fAdaroiF;)

The address field of the instruction is transferred to the MAR. This is then used to fetch the address of the operand. Finally, the address field of the 1**R** is updated from the MBR, so that it now contains a direct rather than an indirect address.

The. IR is now in the same state as if indirect addressing had not been used, and it is ready for the execute cycle. We skip that cycle for a moment, to consider the interrupt cycle.

The Interrupt Cycle

At the completion of the execute cycle. a test is made to determine whether any enabled interrupts have occurred. If so, the interrupt cycle occurs. 'Mc nature of this cyCle varies greatly from one machine to another. We prescra a very simple sequence of events, as illustrated in Figure 12.K We have

t 113R (PC
t$_L$ MAR 4 SaveAcir-a
PC 4- Rout,Lne_Ac",dress
tilettufy : MET;

In the first step, the contents of the PC are transferred to the MBR, so that they can be saved for return from the interrupt. Then the MAR is loaded with the address at **which** the contents of the, PC are to be saved, and the PC is loaded with the address of the start of the interrupt-processing routine. These two actions may

each be a single micro-operation. However. because most processors provide multiple types and/or Levels of interrupts, it may Lake one or more additional micro-operations to obtain the save_address and the routine_address before they can be transferred to the MAR and PC, respectively. In any case. once lhis is done. the final step to store the MBR, which contains the old value or the PC into memory, 'Fhe processor is now ready to begin the next instruction cycle.

The Execute Cycle

The fetch, indirect, and interrupt cycles are simple and predictable. Mach involves a small. fixed sequence of micro-operations and, in each CAW, the same micro-operations arc repeated each tune around.

This is not true of the execute cycle, For a machine with N differen opCOdeS, there are N different sepienees of micro-operations that can occur. LeL us; consider several hypoi helical examples.

First. consider an add instruction:

ADD R1, X

which adds the contents of the location X to register RI. The following sequence or micro-operations might occur:

```
VAR      fIR.:address)
MER      Memory'
    <— +.R1        (MER!
```

We begin with he IR containing the ADD instruction. in the first step, the address portion of the IR is loaded into the MAR. Then the referenced memory Location is read. Finally, the contents of R1 and MBR are added by the AIX. Again% this is a simplified example„.Nilditional micro-operations may be required to extract the register reference from the IR and perhaps to stage the AL, U inputs or outputs in some intermediate registers..

Let us look at two more complex examples. A common instrueth PTJ is increment and skip if zero:

ISZ

The content of location X is incremented by L. If the result is 0. the next instruction is skipped. A possible sequence or IILicro-operations is

```
E     MAR      (11.7.1:adciress) I
t, FIER      Xemory
t, MBR       (MBR) -
t...  Memo r-y-      (MBR)
          (•alR.) = CO then {?C.      (PC : – 1)
```

The new feature introduced here is the conditional action, The PC is incremented if !\l B1.?..) = This test and action can be implemented as one micro-

operation. Note also that this micro-operation can be performed during the same time unit during which the updated value in MB R i stored back to memory.

Finally, consider a subroutine call instruction. As an example, consider a branch-and-save-address instruction:

BSA X,

The address of the instruction that follows the RSA instruction is saved in location X, and execution continues at location X — I. The saved address will later be used for return, This is a straightforward technique for providing subroutine calls. The following micro-operations suffice:

WAR (— CFR
1'113F. PC)
PC <— (iR. ':address)
(NBR)
PC ; PC) .1 'T

The address in the PC at the start of the instruction is the address of the next instruction in sequence. This is saved at the address designated in the **IR.** The latter address is also incremented to provide the address of the instruction for the nest instruction cycle.

The Instruction Cycle

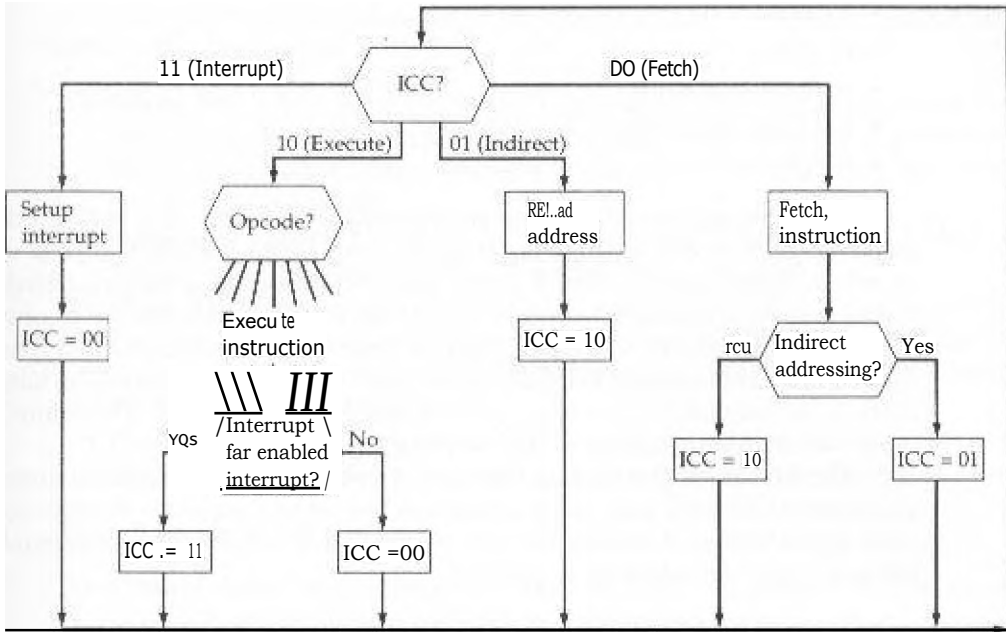
We have seen that each phase of the instruction cycle can be decomposed into a sequence of elementary micro-operations. In our example, there is one sequence each for the fetch, indirect, and interrupt cycles, and, for the execute cycle. there is one sequence of micro-operations for each opeode.

To complete the picture, we need to tie sequences of micro-operations together, and this is done in Figure 16.3. We assume a new 2-hit register called the *immtetion cycle code (ICC),* The ICC designates the state of the processor in terms of which portion of the cycle it is in:

00: Fetch
01: Indirect
10: Execute
1I: Interrupt

Al the end of each of the four cycles, the ICC is set appropriately. The indirect cycle is always followed by the execute cycle. The interrupt cycle is always followed by the fetch cycle (see Figure 12.41. For both the execute and fetch cycles, the next cycle depends on the state of the system.

Thus, the flowchart of Figure 16.3 defines the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern. Of course. this is a simplified example. The flowchart for an actual processor would be mote complex. In any case, we have reached the point in our discussion in which the operation of the processor is defined as the performance of a sequence of micro-operations. We can now consider how the control unit causes this m2quence to occur,



1-Igiire 16..3 Ficywchart Ii Instructil.Fn

16.2 CONTROL OF THE PHOCESSOR

Functional **Requirements**

result of our analysis in the preceding section. we have decomposed the bc-haviOr or funclioning of the processor **ink** elemcni.nry o r i,:rations, callcd **icro-**operations. By, reducing the operation of [he pi cluessai to its most fundamental level, we are able to define exactly what it is that the control **unit** must cause tc.) happen, Thus, we can define the *fiinctional requfremenrs* for the control unit: those functions that the. control unit must perform. A definition of these functional requirements is the basis for the design and implementation of the control unit.

'With the information at hand, the *following* three-step process leads to a char-eicrizaLion of the cornrol anti;

1. Define the basic elements of the processor.
2. Describe the micro-operations that the processor performs.
3. Determine the functions that the control unit must perform Lo cause the miCro-opf.l.TaLiorVs i« he performed.

We have already performed steps I and 2. Let us summarize the Icsuit:;- First, he basic funetionai elements the processor are alc f011owirig!

- ALU
- Registers
- Internal data paths
- External data paths
- Control unit

Some thought should convince you that this is a complete list. The. Al .11 is the functional essence of the computer. Registers are used to store data internal to thr processor. Some registers coniain status information needed to manage instruction sequencin.g (e.g., a program status word}_ Others contain data [hail go 10 or come from the ALU. memory, and I/O modules. Internal data paths 41E0 toed to move data between registers and between register and ALL. External data paths link registers to memory and I/O modules, often by means of a system bus. The control unit causes operations to happen within the processor.

'rho execution of a program consists of operations involving these processor elements. As we have seen, these operations consist of a sequence of micro-operations. Upon review of Section 16.1, the reader should see that all micro-operations fall into one of the following cateaories!

- Transfer data from one register to another.
- Transfer data from a register to an external interface (e.g., system *bus).*
- Transfer data from an external interface to a register.
- Perform an arithmetic or logic operation, using registers for input and output.

Al] of the micro-operations needed 10 perform one instruction cycle, including all of the micro-operations to execute every instruction in the instruction set. (all into one of thew cittegories.

We can now be somewhat more explicit about the way in which the control unit functions. The control uni t Furrorms two basic lasky:

- Setptencin The control unit eauseN the processor 0.11 step through a series of micro-operations in the proper sequence, based on the. program being execute,a
- Execution: The ctphlrol unit causes each micro-operation to be performed.

The preceding is a functional dewript ion of what the control unit does. The key to how the control unit operates is the use of control signals.

Control Signals

We have defined the elements that make up the processor (ALL'. registers, Lima paths) and the micro-operations that are performed. For the control unit to perform its function, it must have inputs that allow it to determine the state of the system and outputs that allow it to control the behavior of the system. Thctie are the external specifications of the control unit. laternall! ᶠ, the control unit must have the logic required to perform its sequencing and execution functions. We defer a discussion of the internal operation of the control unit to Section 16.3 and Chapter 17. The remainder of this section is concerned with the interaction fictwccn the control unit and the other elements of the processor.

Figure 16.4 is a geneniI mode] of the control unit, showing all of iis inputs and outputs. The inputs /Ire as fa lows:

* **Clock: Tki:i** is how the control unit ¯keeps time.¯ The conl rot 'mil causes one micro-operation (or a sci of simillEaneous micro-operations) to be performed for each clock pulse. This is sometimes referred to as the processor cycle time or the dock cycle time.
- **Instruction register: The opcotic of** the current instruction is used to determilic which micro-operations to perform during the execute cycle.
- Flags: These arc ncethAl by the control unit to determine the st4i r us or the processor and the outcome of previews ALL' operations. For example, for the ineromenl-and-skip-if-zero (JSZ) instruction, the control unit **will** increment the PC if the 4c:tio flag is set.
- **Control signals front control hus:'1 hu control bus portion of the** system hus provide., signals to the control unit, such as interrupt signals and acknowledgments.

The outputs are as follows!

- **Control signals within the processor:** These are two types: those the] cause data to be moved from one register In another. and those that activate specific ALL functions.
- **Control signals to control bus: These are also of** two types: control signals lo memory, and.control signals to the I/O modules.

The new element T hat has been introduced in this figure is the control sigma Three types of control signals arc used: those that activate an ALE! function, those that activate a data path, and those that arc signals on the external system bus or other external interface. ALL of these signals are ultitmitcly a **ppl** ied dirco Iv as binary inputs to individual logic gates.

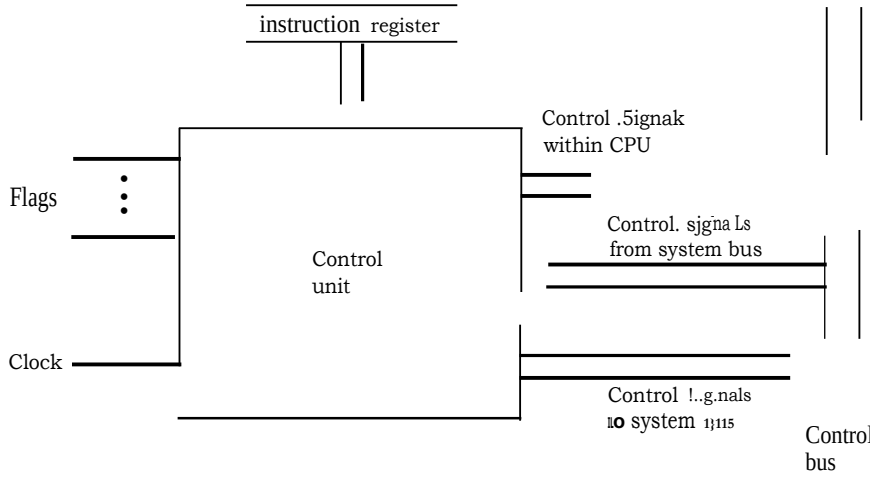


Figure 16.4 ?41.0c11 of the Control Unit

Let us consider again the fetch cycle to sec how the control unit maintains control. The control unit keeps track of Where it is in the instruction cycle. Al a given point, it knows that the fetch cycle is to be performed next. The first step is to transfer the contents of the PC to the MAR, The control unit does this by activating the control signal that opens the gates between the bits of the PC and the bits of the !OAR. The next step is to read a word from memory into the MBR and increment the PC. The control unit does this by sending the following control signals simultaneously:

- A control signa] that opens gates, allowing the contents of the MAR onto the address bus
- A memory read control signal on the control bus
- A control signal that opens the gales, allowing the contents of the data bus to he stored in the MBR
- Control signals to logic that add 1 to the contents 0r the PC and store the result back to the PC

Following this, the control unit sends a control signa] that opens gates between the MBR and the IR.

This completes the fetch cycle except for one thing: The control unit must decide whether to perform an indirect cycle or an execute cycle next. ' fo decide this, it examines the IR to see if an indirect memory reference is made.

The indirect and interrupt cycles work similarly. For the execute cycle. the control unit begins by examining the opeode and, on the basis of that, decides which sequence of micro-operations to perform for the execute cycle,

A Control Signals Example

To illustrate the functioning of the control unit, let us examine a simple example. Figure 16.5 illustrates the example. This is a simple processor with a single accumulator. The data paths between elements are indicated. 'Ihe control paths for signals emanating from the control unit are not shown, but the terminations of control signals are labeled C. and indicated by a circle. The control unit receives inputs from the clock, the instruction register. and flags. With each clock cycle, the control unit reads all of its inputs and emits a set of control signals. Control signals go to three separate destinations:

- Data paths: The control unit controls the internal flow of data. For example, on instruction fetch. the contents of the memory buffer register are transferred to the instruction register, For each path to be controlled, there is a gate (indicated by a circle in the figure). A control signal from the control unit temporarily opens the gate to let data pass.
- **ALU:** The control unit controls the operation of the ALU by a set of control signals. These signals activate various logic devices and gates within the ALU.
- **System bun: 'I`he** control unit sends control signals out onto the control lines of the system bus (e.g., memory READ),

The control unit must maintain knowledge of where it is in the instruction cycle. Using this knowledge, and by reading all of its inputs. the control unit emits

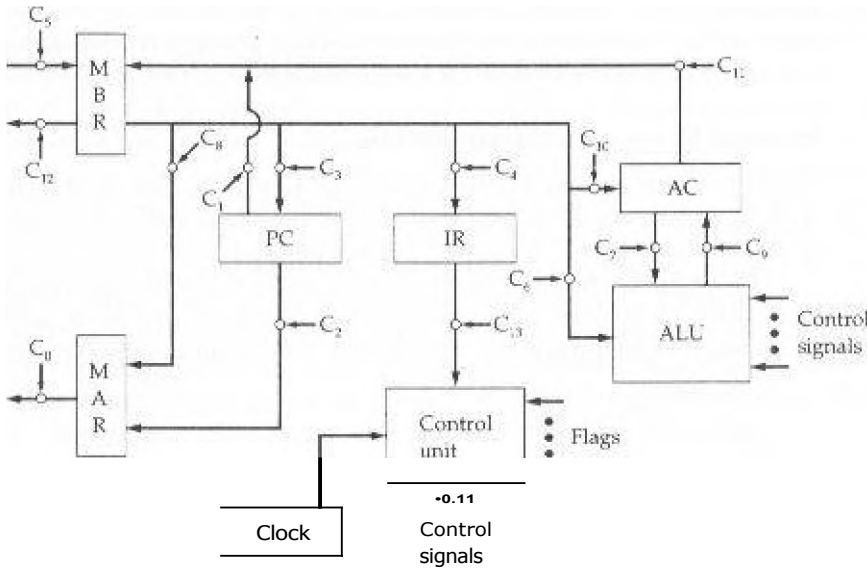


Figure 16.5 Data Paths and Control Signals

a sequence of control signals lIuiI muses micro-operations to occur. I . uses the dock pulses Io time the sequence tai events., ki [ow ing time between events. for sina] levels to sLa ham. Talle 61 indicates the control signals that are needed for 'some of the micro-operation sequences described carlier. For simplicity, the data and Control paths for incrementing the PC and for loading the fixed addrcssc!, int() the PC and MA **R ;ire** not shown.

It is worth pondering the minimal nature of the control unit. The cons rcrl Linn is the engine that runs the entire computer. It does this based only on knowing the instructions to be executed and the nature of the results of arairrictii: rind logical

Table 16.1 Micro-Opc.ratiums and Cornrell Signals

Micro-Operations	**Timing**	**Control Sigmas**
FoieIT	(PC) MEIILOTy]'{' r(FC)+ I 13! (MBR)	
indifuet:	i is MAR CIR(Addressi$ t2; MI31i Mcinciry L3! 1RI:Address) e— (ME3R(AddrcNs0	C,
Irittrrap(:	LI: 1 1f3R E (PC) t2; MAR ⟵ SaVi!-NdlirCSS• • PC (— Rotnaw-Nklitress L3- lvkinory .(• (MBR)	C,
34x,vi Cy. -	ts.:•• ':slet11 **Ny'liCirl**	

opuations (e.g., positive, overflow, cic,). It never gets to see the data being processed or the actual results produced: And i l Quntrois everything with a few control signals to points within the processor and a Few control signals to the sysLctn bua-

Internal Processor Organization

Figure 16.5 indicatus the use of a variety of data paths. The complexity of this type of organization should be clear, More typically, some. sort of internal bus arrangement, as was suggested in Figure 12i, will be used.

Using an internal processor bus, Figure [6.3cai he rearranged as shown in Figure 1(-1.h. A Nilltgle iniernal bus connects the **ALU and al]** processor rq.iste. Oates and control signals rf1-1,: provided for movement of data onto and off the bus

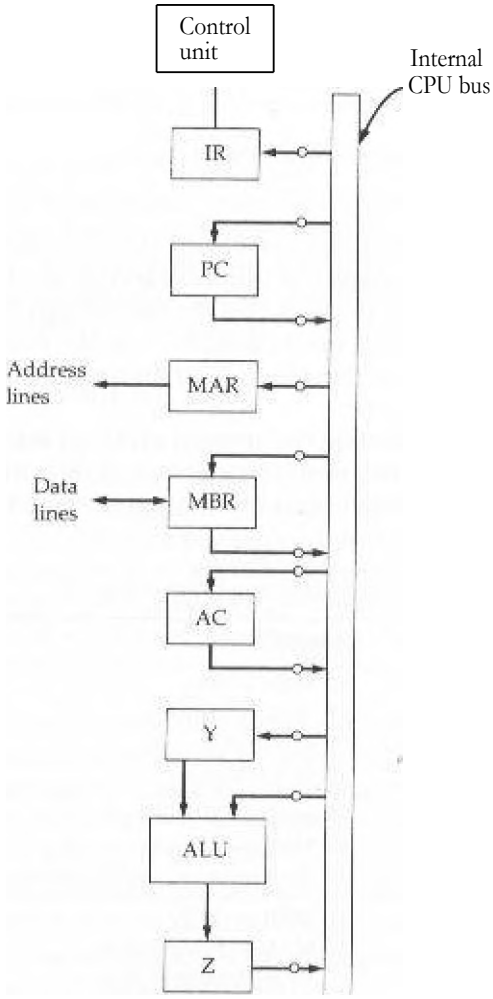


Figure 16.6 CP1 .' with internal Bus

from each register. Additional control signals control data transfer to and from the system (external) bus and he operation *or* the ALU.

Two new registers, labeled Y and Z. have been added to the organizalion. These are needed for the proper operation of the ALL. When an Ope1˙4+1jOn involving Ewo ipperands i perrornicti, now can 1˙ic ohudlled from the internal bus. but the other must be obtained from another source. The AC. could be used for this purpose, but this limits the flexibility of the system and would not work wi li processor with multiple general-purpose registers. Register Y provides temporary storage for the other input. The A **LIH** is a combinatorial circuit (see Appendix A) with no internal storage. Thus. when control signals activate an ALL:function, the input to the ALL is transformed to the output. 'Thus. the output 0r the AU; cannot he directly conneeied to the bus. b4cau7 '.e this output would feed hack to the input Register Z provides temporary output storage. With this arrangement. an operation to add a value from memory to the AC would have the following steps;

MAR 4– 1: :R i address)
MER Y_emory
,; (19˙1I)
(AC', + ("Z)
1.. z AC (—

Other organizations are possible, but in genera[. some sort of internal bus or set of internal buses is used. The use of common data paths simplifies the interoonncction layout and lhe eontro] (lithe processor. Another practical reason for the use of an internal bus is to save space. Especially for microprocessors, which may occupy only a 114-inch square piece of silicon, space occupied by interregister connections **must** be minimizaed.

The Intel 8085

'1˙**0** illustrate some of the concepts introdu0..!{ 1 ı ıuıı- **far** in this chapter, let us consider the Intel 8085. Its organivation is Nhown in Figure 16.'7. Several key components that may not be self-explanatory arc as follows:.

- **Incremeuteridecrementer address latch:** Logic that can acid I. to Or subtract rrtlin the conLeTsis .or the slack roinlcr or prOgrMn counter. This saves time by avoiding the use of the A LI.J for this purpose.
- **Interrupt control:** This tnodule handles multiple levels of in1errup1 signals.
- **Serial I/O control;** This module interfaces to devices that communicate 1 bit at a time.

Table I rp.2 describes the external signals into and out of the KfIK.5. These are linled to 1he eikternal system bus. 'E hesc Signals arc the inierface between the 8085 processor and the rest of 1.1112 system (I iigure 16.8),

The control unit is identified as having two components labeled (1) instruction decoder **and** Machine cycle encoding and (2) timing and control. A discussion of the first component is deferred until the next section. The essence of the control unit is the timing and control module. 'This. module includes a clock and accepts as inputs

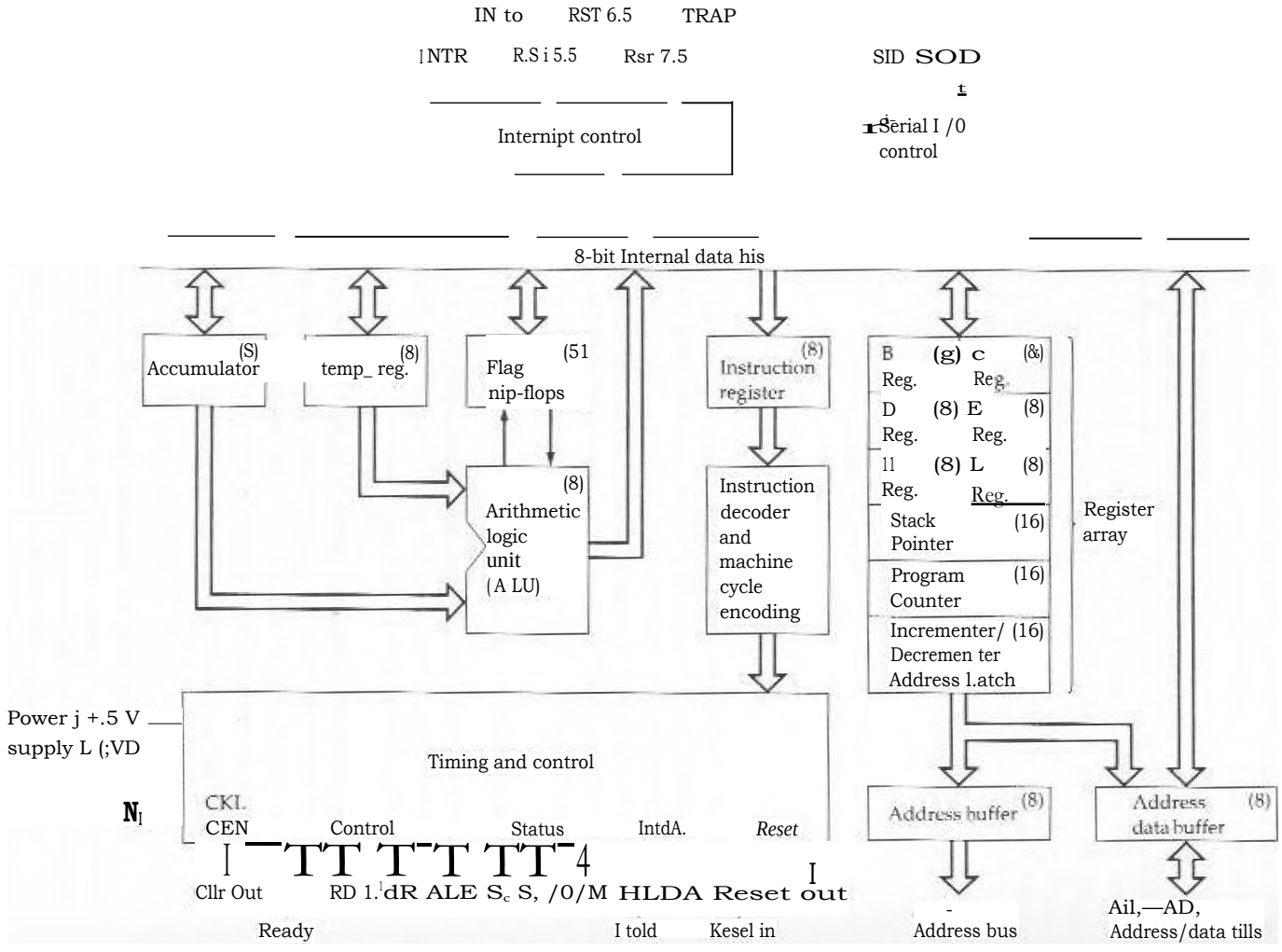


Figure 16.7 Intel 8081. CPU Block Diagram

Fable 16.2 Intel 8085 External Signals

itdrfreS. Dahl Siyl)11J1:

High Address (A1-AR)
The high-order g bits of **a 1 fi-hi1.** address.

AddiremurDabi (AD7—A1}0ni
The lower-order H bits of a Its-bit address or *8* hits of data.. This multiplexing saves on pins.

Serial **Inpni Delo (SID)**
A single-bit input to accommodate devices that transmit aeii:ilfc tone hit at a time),

Serial Output Dula (SOD)
A single-bit Out put to accommodate devices that receive

riMing *and Control SiviaLs*

CLK (OCT)
The system clock. Each cycle represents one's slate. The CLK sienal goes **to peripheral chips and synchronins their liming.**

XI.
These signals come from all external crystal or other device to drive the Internal clock generator.

Address Latch Enabled (ALE)
Occurs during the first clock slate of a machine cycle and causes peripheral chips to store the address lines, This allows the address module **memory.) to** recognize that **it is being addressed.**

Slants (SO, SD
Control signals used to indicate whether a read or write operation is taking place.

Used to enable either U0 or memory modules for read and write operations,

Read Control (RD)
Indicates dim the **selected** memory or I'O module is to **be read and that the data** bus is available **for data transfer.**

'Lille Control (MR)
Indicates. that data **on the** data bus is to he **written into the selected** inemnri **or** I/O location.

Memory am/ *Itiitietrett Symbr•1.i'*

Hold
Requests the. CP1.s to relinquish control and use of the external system bus. The CPC will complete exec-10k m **of the** instruction presently in **the IR and then enter a hold** state, **during which nu** signals are inserted **by I he CPI, to the control. addres, or** data buses. During **the** hold slate, **the bus may be used for DMA operations.**

Hold Acknowledge IHOLDA)
This control unit output signal acknowledges the HOLD signal and indicates that the bus is now available:

READY
Used to **synchronin the CPC wit h sLtiwei memory** or UO. de **ities. When en addressed** device iissens **READY, the CP1 ' may proceed with an input (DBIN) or output 1WRt operation, Otherwise, the CPU** enter+. **a w** ait state until i he device is ready.

imermpi-kelated *SignolY*

TRAP
Restart **Interrupts** (RST 7.5. 6.5.

Interrupt Request (1NTR)
These lines are used by an external Lievice to interrupt the **CPU, 'The CPU will not honor the request if it is in the hold state or if the interrupt is disabled, An interrupt is honored only at the completion of an instruction. The interrupts are in descending order of priority_**

Acknowledge
Acknowledge'. in interrupt.

cpti

RESET IN
Causes **the contents of the PC to be set to zero. The CPU resumes** execution **al location 7,0'0_**

RESET OUT
Acknowledges that the CPU has been reset_ The signal can he used to reset the rest of the system.

Voltage and Gtrorrnel

VCC
+5 volt power supply

USS
Electrical eround

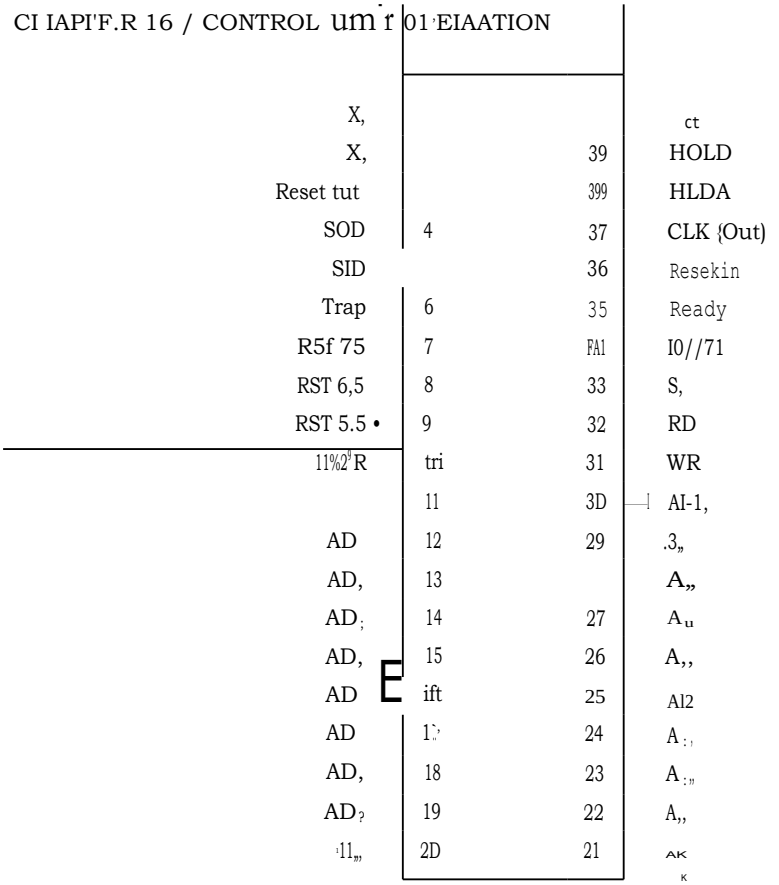


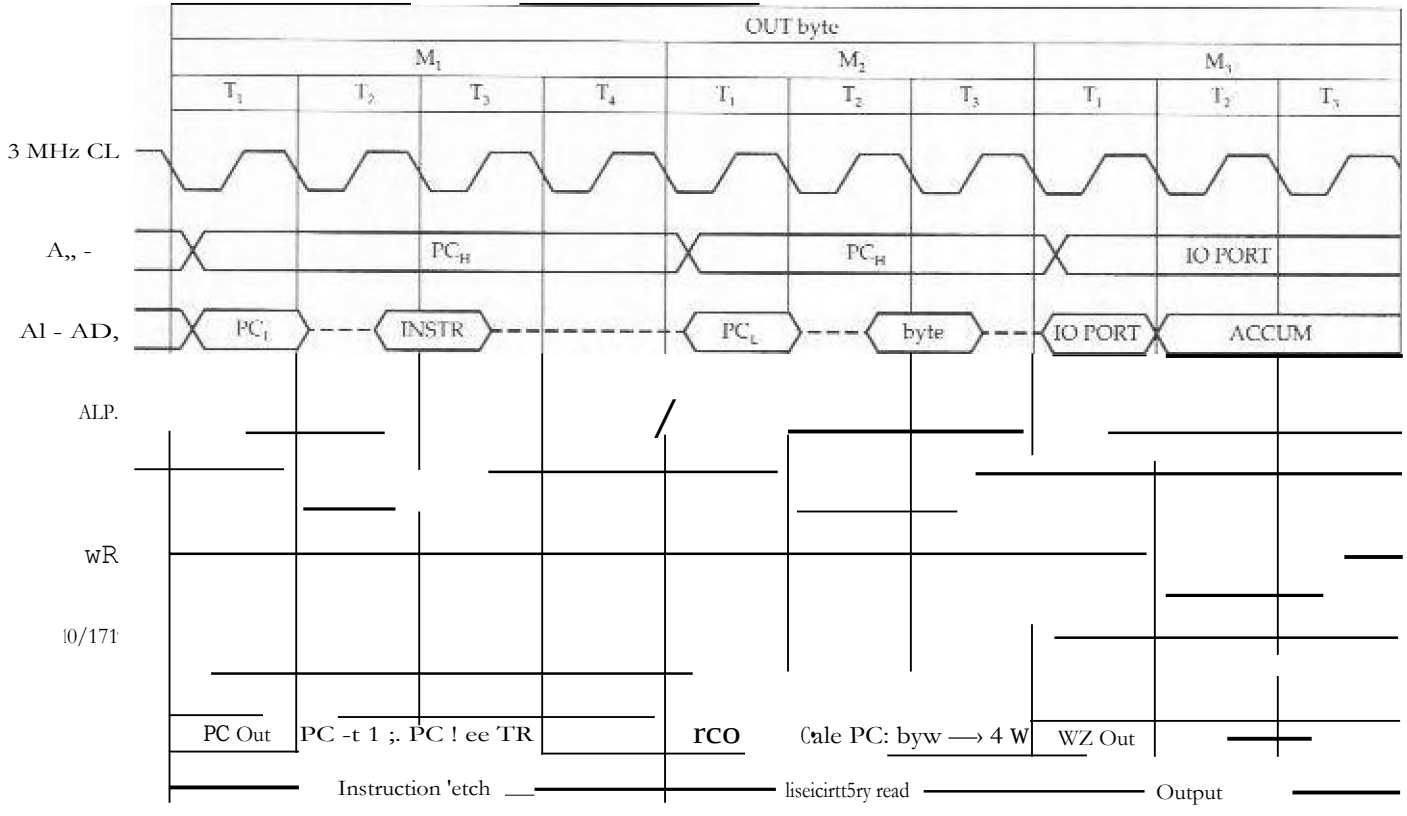
Figure 116,8 Inic] &{185 Pin C'onfiguraljou

the current instruction and some external control signals. f (Pulpit consists of control signals to the other components, of the processor plus control signals to the external system bus.

The liming of processor operations is synchronized by the clock and canirolled by the *conirol* unit with control signals. Each instruction cycle is divided into from one to five *machin e. cycles:* each machine, cycle is **in** turn divided into from three to five *stares.* Each state lasts one clock evelc. During a state, the processor performs one or a set of simultaneous micro-operations as determined by the con ' 11·01 signals,

The number of machine cycles is fixed for a given instruction but varies from one instruction to another. Machine cycles are defined 10 be equivalent to hug accesses. Thus, the number of 1/Lachine cycles for an instruction depends on the number of times the processor must conirriunic4itz with external devices. For example, if an instruction cons.isis of two 8-bit portions, ihcn two machine cycles are required to Teich the instruction. if Ihat instruction involves a 1-1 ·Fyte memory or 110 operation. then a third machine cycle is required for execution.

Figure 16.9 gives an e4inapie of 8085 timing, showing the value of external control signals- Of course, at the same lime, the control unit gcneraies internal control



Flury. 16.9 DiaArtun for mid 808.5 OUT Lrit:truction

signals that control internal data transfers. The diagram shows the instruction cycle for an OUT instruction. Three machine cycles ($M_{1,}$ M_3, M) are needed. During the first, the OUT instruction is fetched. The second machine cycle fetches the second half of the instruction. which contains the number of the I/O device selected for output. During the third cycle, the contents of the AC are written out to the selected device over the data bus.

The Address Latch Enabled (ALE) pulse signals the start of each machine cycle from the control unit. The ALE pulse alerts external circuits, During timing state T, of machine cycle M_1, the control unit sets the 10/NN signal to indicate that this is a memory operation. Also, the control unit causes the contents of the PC to he placed on the address bus (A.. through AO and the address/data bus (AD, through ADO, With the falling edge of the ALE pulse, the other modules on the bus store the address.

During timing stale T,. the addressed memory module places the contents of the addressed memory location on the address/data bus. The control unit sets the Read Control (RD) signal to indicate a read, but it waits until '1 to copy the data from the bus. This gives the memory module time to put the data on the bus and for the signal levels to stabilize. The final state, T_4, is a *bus idle* suite during which the processor decodes the instruction. The remaining machine cycles proceed in a similar fashion.

16.3 HARDWIRED impuiviEN-iiiiarW

We Imve discussed the. control unit in terms of its inputs, output., and functions. We now turn to the topic of control unit implementation. A wide variety of technique. have been used. Most of these fall into one of Iwo categories:

- Hardwired implementation
- Microprogrammed implementation

In a hardwired implementation, the control unit is essentially a combinatorial circuit. Its input logic signals are transformed into a set of output logic signals, which are the control signals. This approach is examined in this section, Microprogranuned implementation is the subject of Chapter 17.

Control Unit Inputs

Figure 16.4 depicts the control unit as we have so far discussed it. The key inputs are the instruction register. the clock, flags. and control bus signals. In the case of the flags and control bus signals. each individual bit typically has some meaning (e;g.. overflow). The other two inputs, however, are not directly useful to the control unit,

First consider the instruction register. The control unit makes use of the ()Node and will perform different actions (issue a different combination of control signals) for different instructions. To simplify the control unit logic, there should be a unique logic input for each opcode. This function can be performed by a *decoder,* which takes an encoded input and produces a single output. In general. a decoder

will have *n.* binary inputs and 2." binary outputs. Each of the 2' different input patterns will ;.ictiyatc a single unique output. Table 16,3 is an example. The decoder for a control unit will typically hays to he more complex than that, to account for variable-length opcodes. An example or I hc digital logic used to implement a.decoder is prescrfled in Appendix A.

'the clock portion of the control unit issues a repetitive sequence of pulses. This is useful for measuring the duralion of micro-operations. Essentially, the period of the clock pulses must be long enough to allow the propagation of signals along tlatit paths and through processor circuitry. However, as we. have 8L2[1, the control unit emits different control signals at different lime units within a single instruction cycle. Thus, we would like a coupler as input to the control unit. with a different control signal being tiwd **and so forth. At the end of an** Ins lrucl ion cycle:, the control unit must feed back to the counter to reinilializc it all

With these two refinements, the control unit can lie depicted, as in Figure 16.10,

Control Unit Logic

To define the hardwirc0 impIernentation of a control unit, all that remains is to discosZi Lhc inicrnal logic of the control unit that produces output control signals as a function of its input signals,

Table 16.3 A. Decoder with Fuur Sixt4.24.11 Outputs

II	12	13	14	01	02	U	04	05	()(I	07	0/	09	010	OIL0E2	013	014	01.5	•01fi
0	0	0	0	43	ci•	0	0	0	(1	41	0	0	n	.0	0	0	0	0
0	0	0	0	0	0	0	0	I)	0	0	9	Si	•0	it	0	0	0.	1
41	E	0	13	0	Ii	0	9	0	0	Li	0	0	0	0'	U	1	t)	0
0	U	1	1	0	LI	0	0	I)	0	0	Q	0	0	0	1	0	0	0.
I	3	0	0	0	0	C'	0	0	0	1.1.	0	(I	0	0	1	0	0	0
ti	I	13	E	0	I)	0	0	0	0	I)	0	0	41	3	0	0	0	0
	I	I	0	.0	0	0	4i	0	0	0	0	0	1	0	0	0	0	0
	3	E	1	0	F)	0	0	0	0	0	0	1	0	0	0		41	0
1	ii	0	0	0	ti	0	0	41	0	1)	I	0	0	0	0	0: 0	0	0
	ii	O'	1	Li	0	0	II	41	0	1	0	0	0	0	0	0 0	0	11
	0		0	0 1	I)	0	IF T	0	1	0	0	41	0	LI	0	0 II		0
	ii	1	1	0	0	0	IF	1	0	0	0	0	0	0	0	0 41		0
1	J.	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0 I	0	1)
I:	L	0	1	0	0	1.	0	0	0	0	0	0	0	0	0	0 0	0	0
	I	1	0	0	1	43	0	0	0	0	0	0	0	0	0·	0 1.0.	41	41
					0	1)	0	0	a	0	0	0	Q	0	0	U		0

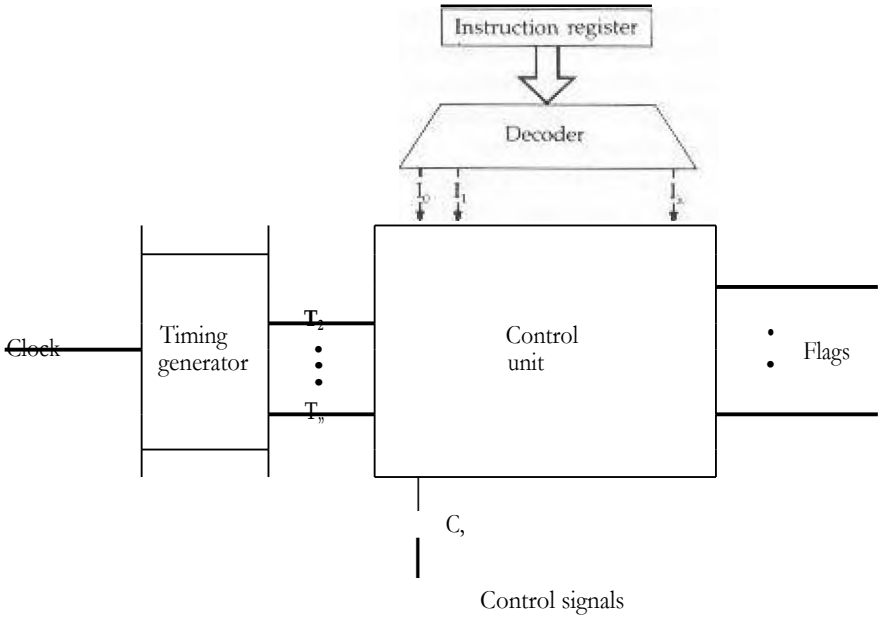


Figure 1.6.10 Control Unit with Decocted Inputs

Essentially, what must be done is, For each control signal, to derive a Boolean expression of that signal as a function of the inputs. This is best explained by example. Let us consider again our simple example illustrated in Figure 16.5. We saw in Table 16.1 the micro-operation sequences and control signals needed to control three of the four phases of the instruction cycle.

Let us consider a single control signal, C,. This signal causes data to he read from the external data bus into the MBR, We can *see* that it is used twice in 'Table 16.1. Let us define two new control signals. P and 0. that have the following interpretation:

Pc.= 02.	Fezch Cycle
PQ = 0:	Incnrect Cycle
?Q= 1C	txecute Cycle
Q= 11	Tnterrupz Cycle

Then the following Boolean expression defines C,:

$$C_{,} = P \cdot Q \cdot T_{,} + P \cdot 0 \cdot$$

That is. the control signal O_a will be asserted during the second time unit of both the fetch and indirect cycles.

This expression is not complete. C', is also needed during the execute cycle. For our simple example, let us assume that there are only three instructions that read from memory: LDA, ADD, and **AND. Now we can define** C, as

C 5 = P O • T2 T P • Q • + P • Q • (1....DA -I- A DD + AND) • T2

This same process could be repeated for every control 6ignal generated by the processor. The resull would he a set of Boo]esn equal ion:i ihat derine the behavior of the control unit ;ind hence of the processor.

To tie everything toget her. the control unit must control the state of the instruction cycle. As was maitioned, at the end of each subcycle (fetch, indirect, execute, interrupt), the control unit issues a signal that causes the Liming generator to reinitialize. and issue 'f,. The control unit must also set the appropriate values of P and Q to define the next subcycle to be performed.

The reader should be able to appreciate that in a modern complex processor, the number of Boolean equations needed to define the control unit ix very large. The task of implementing a cornbinAtorial circuit that satisfies all of these equations becomes extremely difficult. The result is that a far simpler approach, known as *inie.`rop mgrainnang.* is usually used. This is the subject of the next chripler.

16.4 RECOMMENDED READING

number of textbooks I I tmi I Ehc bask. principles of control unit function; two pall ictilarly clear treatments are in [HA Y.E981 and INIAN00]1..

HAVV98 1-11:q0s, *3. Compuipy.* A *rrlüire oto re. ,10.1d Orgym iz vii on,* Nc.vii York: McGraw-Hi]] . 1998.

A1AN0111 Islme,.M. *Logic cord Computer Des4m f, rn, ir.intoJtu* *!Vol'* , PrefiricE• Nail, I WI.

16.5 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terris

control bus	control signal	hardwired implementation
conto.5.1 path	control unit	microopera

Review Questions

16.1 Explain the distinction Between (he written scqucecc and the time sequence of an instruction.

16.2 What is the relatinnshipl peiwen in.structions and micro-operations?

163 What is the overall function of a proctssur's control unit?

16.4 Outline a three stap process dint leads to a ellaractai2ation of chi control unit.

16..5 What basic tasks does a control unit pc.rforrn?

1(0 Priividc a typical list of the inputs and outputs of ec.itrill

1fi..7 Lig! three Cypcs of control signals.

16.N. Elijah explain what is meant by a hardwired iinplemotalion of a control unit.

Problems

[6.1 Your ALU can add its 14ko input registers, and it can logically complement I lie bits of eith&r input register_ but it cannot **6ubtrau.** Numbers arc to be stored in twos complement representation, L,isl the tilierooperations yoLli control unit must perform to cause a subtraction,

162 Show the miero•operations and control signals in the same fashion as Table 16_1 for the proccssor in Figure 16.5 for the rollowing instructions:

- Load Accumulator
- Store Accumulator
- Add to Accumulator
- AND w Accumulator
- Jump
- Jump if AC - 0
- Complement Accumulator

16.3 Assume that propagation delay along the bus and (haw gh the ALU of f=igure L6,6 are 20 and /00 as, respectively. The time required lot ruglster to copy data from thu buE is 10 ns. What is the Oink:. lhat must be allowed for

a. trace ferring data rroni one register to another'?'

b. incrementing the program courur ')

1.6.4 Write the sequence of micro-operations required rot the bus structure of Figure 16.6 to add a number to the AC when the number is

a. an immediate operand

13: a direct-address operand

f. an indirect-address operand

16.5 A stack is implemented as shown in Figure L0.14, Show the sequence of micro operations Fo r

21. popping

b. pushing the stack

CHAPTER 17

MICROPROGRAMMED CONTROL.

17.1 Bilk trincepf3

icluirmiroat[ins

Microprogranamed C.o.ntroi Unit .

V•'ilkes Controi

Advantages and 1)isativantilgo.:

17.2 Microinstruction Sequencing

Design C.:onsidc.rations

Sequencing Techniques

Address Generation

L.S1-11 Vticroinsimeiion Svqiiencing'

17.3 Microinstruction Execution

A. **Taxonomy** of

• icroinstroerion 1::ncoding

,S1- I. Microi n staid:ion Execution .•

IBM 3033 klierylmtruetion Exccution.

174 II sson

Microinstruction Format

MicroF.cgtm•ncer

Rugistc.i•ed AL.(.

11.5 **Applications or Microprogramming**

17.6 Reoommended Reading

17.7 Key Terms, Review QuestiorA, arid Problems

Key rictms

Questions

Problems

KEY POINTS

- An alternative to a hardwired control unit is a microprogrammed control unit ; in which the logic of the control unit is specified by a microprogram. A microprogram consists of a sequence of instructions in. a microprogramming language_ These are very simple instructions that specify micro-operations,
- A microprogrammed control unit is a relatively simple logic circuit that is capable of (/) sequencing through microinstructions and (2.) generating control signals; to execute each microinstruction.
- As in a hardwired control unit, the control signals generated by a microin-, struction are used to cause register transfers and ALL) operations_

The term *microprogram* was first coined by M. V. Wilkes in the early 1950s [WII,K5 l]. Wilkes proposed an approach to control unit design that was .:Organized and systematic and avoided the complexities of a hardwired implementation, The idea intrigued many researchers but appeared unworkable because it would require a fast, relatively inexpensive control memory.

The state of the microprogramming art was reviewed by *Datamation* in its February 1964 issue. No microprogrammed system was in wide use at that time, and one of the papers I I II 11,641 summarized the then-popular view that the future of microprogramming "is somewhat cloudy. None of the major manufacturers has evidenced interest in i he technique, although presumably all have examined it."

This situation changed dramatically within a very few months. IBM's System360 was announced in April, and all but the largest models were mierc.iprogrammed. Although the 360 series predated the availability of semiconductor ROM, the advantages of microprogramming were compelling enough for IBM to make this move. Since then, microprogramming has become an increasingly popular vehicle for a variety of applications, onc of which is the use of microprogramming to implement the control unit of a processor. That application is examined in this chapter.

17.1 BASIC CONCEPTS

Microinstructions

The control unit seems a reasonably simple device. Nevertheless. to implement a control unit as an interconnection of basic logic elements is no easy task_ 'I he design must include logic for sequencing through micro-operations. for executing micro-operations, for interpreting opcodes. and for making decisions based on ALL flags. It is difficult to design and test such a piece of hardware, Furthermore, the design is relatively inflexible. For example, it is difficult to change the design if one wishes to add a new machine instruction.

An alternative, which is quite common in contemporary OW processors, is to implement a microprogrammed control unit.

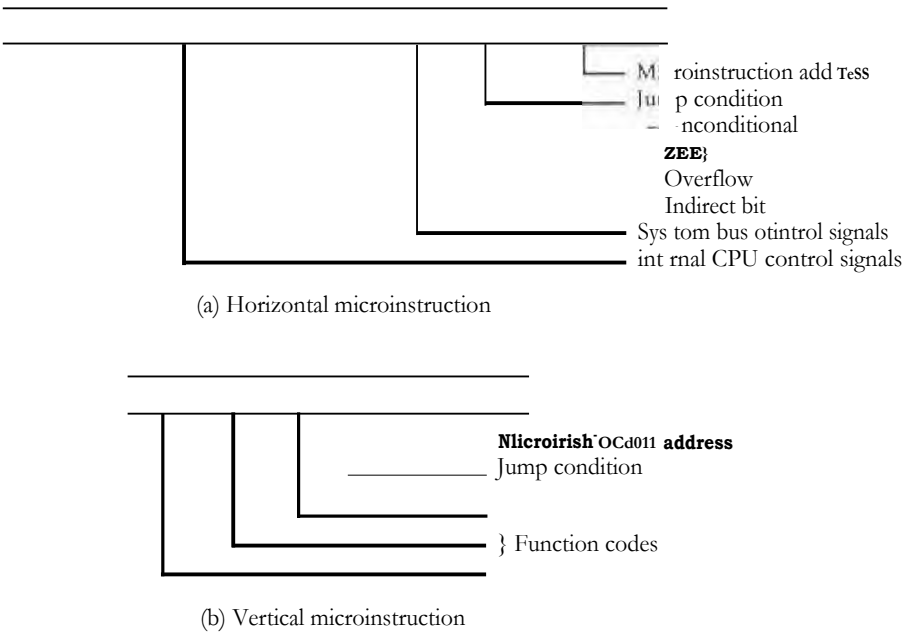


Figure 17.1 Typical Microinstruction FOnnals

Consider again Table 16.1. In addition to the use of control signal.., each micro-operation is described in symbolic notation. This notation looks suspiciously like a programming language_ In fact it is a language, known as a *inicroprogranuning language.* Each line describes a set of micro-operations occurring at one time and is known as a *microlivaraction.* A sequence of instructions is known as a *microprogram, or firmware,* This latter term reflects the fact that a microprogram is midway between hardware and software. It is easier to design in firmware than hardware. but it is more difficult to write a firmware program than a software program.

How can we use the concept of microprogramming to implement a control unit! Consider that for each micro-operation, all that the control unit is allowed to do is generate a set of control signals. Thus, for any micro-operation, each control line emanating from the control unit is either on or off_ This condition can, of course, be represented by a binary digit for each control line. So we could construct a *conrro, word* in which each bit represents one control line. Then each micro-operation would be represented by a different pattern of is and fls in the control word,

Suppose we string together a sequence of con irol words to represent the sequence of micro-operations performed by the control unit. Next, we must recognize that the sequence of micro-operations is not fixed. Sometimes we have an indirect cycle sometimes we do not. So let us put our control words in a memory, with each word having a unique address. Now add an address field to each control word, indicating the location of the next control word to be executed if a certain condition is true (e.g.. the indirect bit in a memory-reference instruction is Also. add a few bits to specify the condition.

The result is known as a *hurizontai inicrginstructic%* an example of which is shown in Figure 17,1a. The format of the microinstruction or control word is as

follows. here. is one bit for each internal proeussor control line and one bit for *each* system bus control line. There is a condition field indicating the condition under which there should be a branch. and there is a field with the address of the micro-. instruction to be executed next when a branch is taken. **such** a microinstruction h interpreted as follows:

1. To execute this microinstruction, turn on all the control lines indicated by a I bit; leave off al] control lines indicated by a 0 bit. The resulting control signals will cause one or more micro-operations to he performed.
2. ll the condition indicated by the condition bits is false, oxeculc the next microinstruction in sequence.
3. If the condition iridie.ri Led by the condition bits is true, the next microinstruction to be executed is indica ted in the address field.

Figure 17.2 shows how these control words cw microinstructions could he arranged ltl a *co land mummy.* The microinstructions in each routine are to be executed sequentially, Each routine ends with a branch or jump instruction indicating where to go next. 'IhCre is a special execute cycle routine whose only purpose to signify that one of the machine instruction routiric (AND, ADD, and so on) is to. be executed **next.** depending on the current opcode.

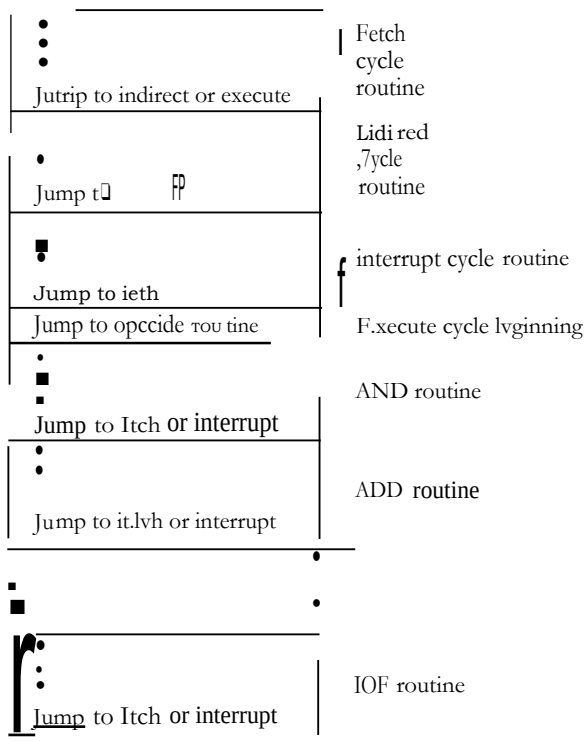


Figure 17.2 Organization cir C'ciiitroJ Mom ory

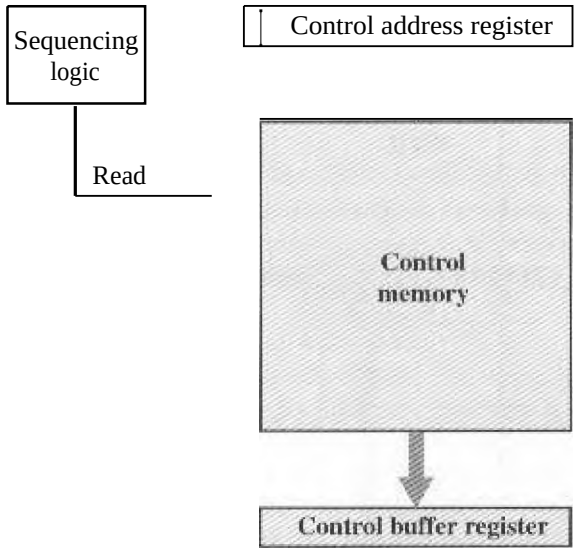


Figure 17.3 Control ll..:nit \tic], mrchitecturc

The control memory of Figure 17.2 is .a concise description of the complete operation of the control **unit. It** defines the sequence of micro-operations th he performed during each cycle (fetch, indirect, execute. interrupt), and it specifies the sequencing of these cycles. If nothing else. this notation would he a useful device for documenting the functioning of a control unit for a particular computer. But it is more than that. It is also a way of implementing the control unit.

Microprogrammed Control Unit

The control memory of Figure 17.2 contains a program that describes the behavior of the control unit. It follows that we could implement the control unit by simply executing that program.

Figure 17.3 shows the key elements of such an implementation. The set of microinstructions is stored in the *control memory.* The *control addre.vs regisfrr* contains the address of the next microinstruction to he read. When a microinstruction is read front the control memory. it is Li **mtNferred** to a *control buffer regiver.* The left-hand portion of that register (see Figure 17.1a) connects to the eon I rol lines emanating from the control unit. Thus, *reading* a microinstruction from the control memory is the same as *executing* that microinstruction. The third element shown in the figure is a sequencing unit that loads the control address register and issues a read command.

Lot us examine this structure in greater detail, as depicted in Figure 17,4. Comparing this with Figure 16.4, we see that the control unit still has the same inputs (IR. ALU flags, clock) and outputs (control signals). The control unit functions as follows:

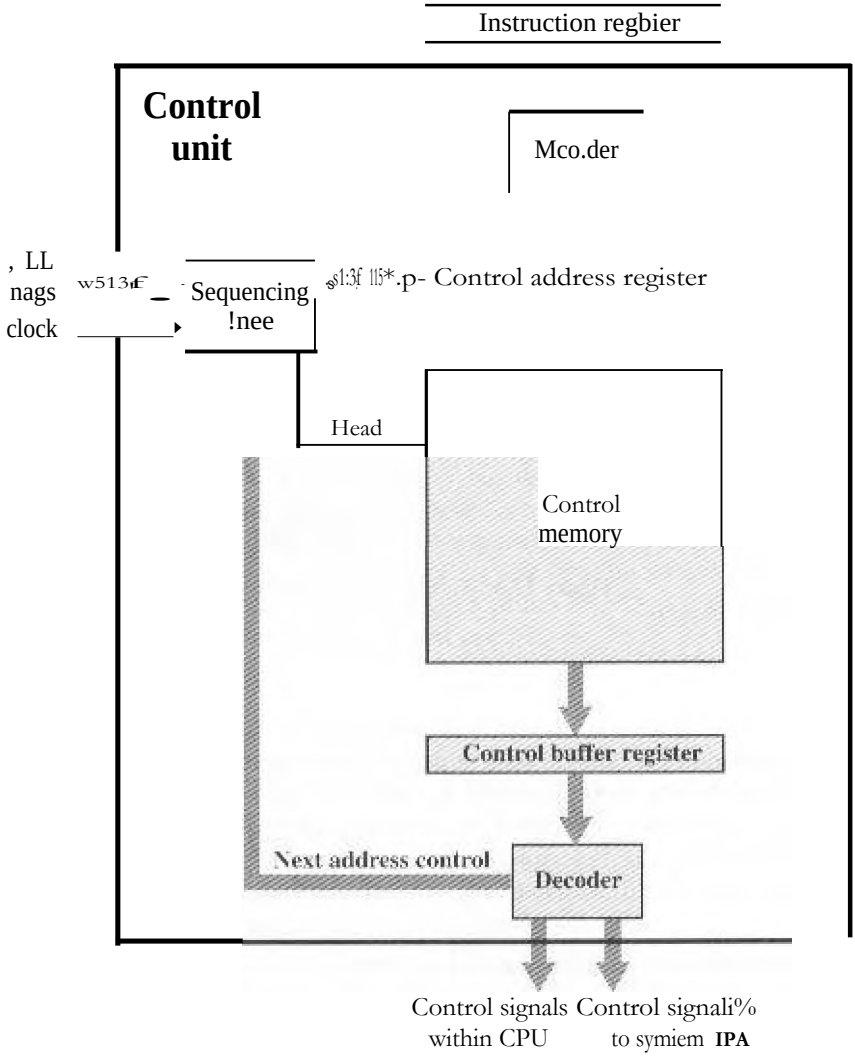


Figure 17.4 Functioning of Microprogrammed Control Unit

1. To execute an instruction, the sequencing Logic (mil iisu s a HEAD roinniand to the control memory.
2. The word whose address is specified in I he control address register is read into the control buffer register.
3. The content of the. control buffer register generates control organs and next-address information for the sequencing logic **will.**
4. **The** sequencin logic unit loads a new address into the control 4ithirt:•Ss register based on the next-address information from I he control buffer register and I he f IA: flogs.

All this happens during one clock pulse.

The last step just listed needs elaboration. AL the conclusion of each microinstruction, the sequencing Logic unit loads a new address into the control address register. Depending on the value of the AL'S flags and the control buffer register, one of three decisions is made;

- **(et the nest instruction: Add 1** to the control address register.
- **Jump to a new routine based on H jump microinstruction:** Load the address field of the control buffer register into the control address register.
- **Jump to a machine instruction routine:** Load the control address register based on the opecrde in the I R.

Figure 17.4 shows Iwo modules *Labeled clecodo*'. The upper decoder translates the opcodc of the IR into a control memory address. The lower decoder is not used for horizontal microinstructions but is used for *vertical microinstructions* (Figure 17.1b). As was mentioned, in a horizontal microinstruction every hit in the control field attaches to a control line. In a vertical microinstruction, a code is used for each action to he performed [e.g., MAR (PC)], and the decoder translates this code into individual control signals. The advantage of vertical microinstructions is that they are more compact (fewer bits) than horizontal microinstructions. at the expense of a small additional tinourit of logic and time delay,

Wilkes Control

A.s was mentioned, Wilkes first proposed the use of a microprogrammecl control unit in 1951 EWILK51 I. This proposal was subsequently elaborated into a more detailed design I WILK53j. It is instructive to examine this seminal proposal_

The configuration proposed by Wilkes is depicted in Figure 17.5. The heart of the system is a matrix partially filled with diodes. During a machine cycle, one row of the matrix is activated with a pulse. 'this generates signals at those points where a diode is present (indicated by, a dot in the diagram). The first part of the row generates the control signals that control the operation of the processor. The second part generates the address of the row to be pulsed in the next machine cycle. Thus, each row of the matrix is one microinstruction, and the layout of the matrix is the control memory.

At the beginning of the cycle, [hi:. address of the row to be pulsed is contained in Register 1. This address is the input to the decoder. which, when activated by a clock pulse, activates one row of the matrix_ Depending on the control signals, either the opcode in the instruction register or the second part of the pulsed row is passed into Register II during the cycle. Register II is then gated to Register I by a clock pulse. Alternating clock pulses are used to activate a row of the. matrix and to transfer from Register II to Register I. The two-register arrangement is needed because the decoder is simply a combinatorial circuit; with only one register, the output would become the input during a cycle, causing an unstable condition.

This scheme is very similar to the horizontal microprogramming approach described earlier (Figure / 7,1a). The main difference is this: In the previous description, the control address register could he incremented by .one to get the next address. In the Wilkes scheme. the next address is contained in the microinstruction,

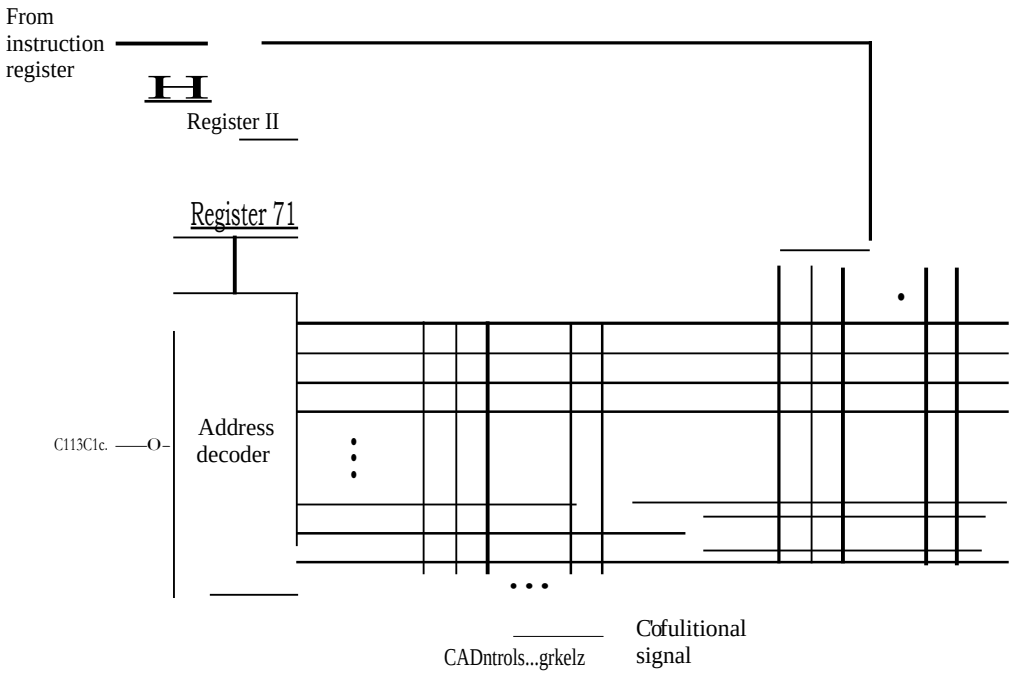


Figure 17.5 Wilkes's Mieroprogrammed Control Unit

To permit branching, **mw TrlUS1 o nl.iin** l wo Kidraqs parts. controlled by a conditional signal (e.g,, flag), as shown in the figure.

Havin•g proposed this scheme, Wilkes provides an exam ➢* of its use to implement the contra[unit of a simple machine. This example, the first known dcsign uF a microprogrammcd processor, is worth repeating here bccause it illustrates many of the contemporary principles of inicroprogramining.

The processor of the hypothetical machine includes the following registers;

A multiplicand

H ncurnulittor (leas! -significan I. hal kl

C accumulator (most-significant half)

I) shift register

In addition, there are three registers and two 1-bit flags accessible only to Ow con. Irc^l aril, The registers are as follows;

E serves as both a memory address register (MAR) and temporary storage

F program counter

▪ another temporary register; used for counting

Table 17.1 lists the machine instruction set for this example. Table 17.2 is the complete sei of microinstructions, expressed in symbolic form, I hal implemcnrs

Table 17.1 Machine Instruction Set for Wilkes Example

Order	Effect or **Order**
A u	CTitec) *C e) tc Ac{'*
H n	*C(N) Lc} fir* ,
!7 e	*C (A cc2)* *(¯(e lo 'Lc ,* where CO) = 1)
T	*C{.4(4.1) In* *(I* to *A cc*
Urr	COCCI Lu *n*
fi r	*.C7414cLi x* 2 ' " to *A*
	OtAcc) *2" in Acc*
Ci *n*	[F -.- [mnslc.r can In ...I; II' [I,]gnc ire Ii. *E., 11TOCC.:2d '021·L.111y* }
I ..I	Read lleNt cl:Ilarecte:r CIII LDilut rrthch SED into **n** Sc nd C:(n) to out put nichanisrrs

No1P91.11.111:. U.I;c1.1 M1.1 I

A cc N3Lini::34"if I I 1·1I: LYI·d{.:CUTL1L110.1..}7

A fr, −Iy: I ;r441¯rilm.;¯;I 11..1 1;1 .1.:gu11tdiiiiiT

. 10;...11-11 I.

OA') eonic II Ls;; 3 ; ; NI u**T** en siorag...; 1;s:;; 1;;;;; I

the controt unit. Thus. a total or 38 microinstructions is all that is required to define the system completely.

The first full column gives the address (row number) of each mierostruction. Those addresse=s eorrcsponding to opcodes are labeled, Thus, when the opcode for the add instruction (A) is encountered, the microinstruction to locai ion 7 7, is executed. Columns 2 and 3 express the actions to be taken by the ALU and control unit. respectively. Each symbolic expression must be translated into a set of control signals (microinstruction Fins). Columns 4 2ind 5 have to do with 1he setting and use of the two flags (flip-flops). Column 4 specifies the signal that scis the flag. For example, MC, means that flag number 1 is set by the sign hit of the number in register C. lIcolumn 5 contains a flag identifier. then columns 6 and 7 contain the two alternative microinstruction addresses to he nsed. Otherwise, column t4 specifics !he address (.11' the next microinstruction to he letchcd.

lmi ruetions 0 through 4 constitute the fetch cycle. Microinstruction 4 presents the opcode to a decoder, which genenues the address of a microinstruction corresponding to the machine instruction to be fetched. The reader should he able to deduce the complete functioning of the control Mi1 from a careful study of Table 17.2.

Advantages and Disadvantages

The principal advantage of the use 0r micn3prognimming It implement a control wail is that it simplifies the design of the control unit. Thus, it is both cheaper and Less error-pri.,ne to implement, A *irtirdwired* control unit must contain complex logic for sequencing through the many miero-operations of the insiruclion cycle. On the other hand, the decoders and sequencing logic unit o[a microprogrammed control unit are very simpie pieces of logic,

Table 17,2 MikiroiOstrUctions fur Wilkes Example

:Notation: A. *13, C*.....stand for the Vdrious registers in the arithmetical and Canto] register units. C' Las t) ktliii.CaD2.5 that the switching circuits connect the output of register C to die input register 0; (1.) LO *C* and cases that the outpus register of A is connected to the one input of the adding unit (ihc oti pu I of D is permanent.... connected 10 the other input). and the output of the adder Li) ℂ A riornerіunl syl1LbDI ?? in C1u01.i•u Slandu for t so, 1-4.c. whow gulped I Its(' nurnfici..ii in !mils of ill.. E,i4i EisnificHnt

		A r i kit met i ca1 **Unit**	Ct]titrull Register Unit	Conditional Flip-Flop		NE:N1 NA ic to i int rut: duo	
				So.	1.7h.e	11	1
	0		F El · ii: 4incl ir...			t	
	]		((; to ⁻ 1') to *F*			2·	
	∴		S.Li.)r Li) Cr			3	
	4		*E to* decoder			—	
A	5	*C* to *D*				*16*	
S	6	C to D				17	
/1r	7	:impe to l'i				1)	
V	8	Store to A				2^1	
7'	q	C to S1.1.31⁻12				25	
I;	i LI	(. kt) S'LOT42				0	
R	11	*14 l....) 0*	*1: go 6*			1 9	
L	12	C^9 to *D*	*EAU G*			*22*	
O	1.:k		*E.01*	*(t)C',*		*lg*	
I	14	irspill In Sicirc				0	
O '	15	Ston.. to Output				L)	
	16	to - Store ' Lo *C*				D	
	L7	(D. - SKITEJ L.[L (				1)	
	L				1	1)	1
	19	*1> co B (RI'*	*(C; - `1⁹) to E*			2[s	
	20	*C* to D		(I).r',		2]	
	7I	*f.)* i4.L C (R)			1	I]	I)
	22	*0 to* C (*L}*`:	*V.1 `I')* to *E*			23	
	23	8 to *L>*		,:1 ***)E,***		24	
	24	.r./ 1 O R (.)			1	12	4
	25	`0· so ii				:Zts	
	2.(1	*B co {*				1)	
	27	`0· Li} *C*	' kV Lul *E*				

Table 172 +.1.1.1 rimmed)

	Arithrnelicat Unit	Ccititroi Register IAA	Coth liiioii.;11 Flip- FIAT	Neu 7.licrii- itistruclion	
			Sct .1's2.	0	1
28	*If* 1.(5 /)	P. TO (i	MU	29	
2[1.]	*D* to 8 *(R)*	K.: • `1%) to E		.3i	
RI	C Lo *V (R)*		*(²)Es*	33	32
.31	D to *C*		2	2S-	33
	(JD —it) to C.		2	2S	33
B	*B to D*		*(¹A*	31	
34	*D* to *B .(R)*			35	
35	C ıo Jr') {10			36	37
4...)	0 so .C			0	
..'	*(0* A) to C'			L.	

¹ kiRt11 tih1f1.111%! SW.;Lchink)eJL LIi1 II .11:ihnulic unit AFL arr: need h;.1h.r1 IIit I ii ELE11 1l1121.11 Jii&i1 01 ale
!v.-Oster Cis plot.] it Ilic 1)1;:Lc f 112}Z=41 13 Ll u micro-Aporations, aid (he most
EjtktLifiClir11 digit 01 r..:j1iN▮ci in:11%h! g I II • i for rtl,ltra);?111),
rlic p.i!..s IIir nis),1:..ipmi:IL)Inl cl it .1' iup,ist,r B to the le,igt.
▪i.,%:..,1 1%gis%, I I: i n%

The principal disadvantage of a microprogrammed unit is that it will he somewhLit slowur i kein 4 hardwired unil of comparble tcchnoltrgy. Despite this microprogramming is the dominant technique for implementing control units in conternporar,' CISC, due to **its ease** of implementation. RISC processors, with their si mpler instruction rormat% typically use hardwired control units- We now examine the m ieroprc Tin mined 4ippro.wh in greater cleWil.

17.2 MICROINSTRUCTION SEQUENCING

The t**wo basic tasks** performed by a atieroprograinimcd control unit are as follows;

- **rilicroinstruction sequencing: Gel the next microinslruction** from I he utyn1roE Memory,
- **Microinstruction execution: Generate the control** signals needed to execute **the microinstruction.**

In designing a control unit, these tasks must be considered together, because both affect the format of the microinstruction and the lirrting of the control unit, In this section, we will focus on si2quencing **and** say as **iittie as possible** a **bouE format and timing issues. These issues are examined in more detail in the next section,**

Design Considerations

'Iwo concerns are involved in the design of a microinstruction sequencing lechniqw:.: the size of the microinstruction anc,lthc address-generation time. The first concern is obvious minimizing the size of the control memory reduces the cost of that cornponcril. The second concern is simply a desire to execute microinstructions as fast as possible.

In executing a microprogram, the address of the next microinstruction to be executed is in one of these categories:

- Determined by instruction register
- Next sequential address
- Branch

The, first category occurs only once per instruction cycle, ust after an instruction is fetched. The second category is the most common in most designs. However, the design cannot be optimized just or sequential aeeess. kranches, both condi• lional and unconditional, are a necessary part 0] a microprogram. Furthermore. microinstruction sequences tend to be short; one out of every three or four microinstructions could he 2i branch [SIEW82]. rhus, ,s irnpoliatiL Lo design compact. time-efficient techniques for tnicroinstruetion branching.

Sequencing Techniques

Based on the current microinstruction, condition flags, and the contents of the instruction register. a control memory address must be generated for the neat microinstruction. A wide variciy or technique '; have been used, wQ. 1:411 group them into three general categories, as illustrated in Figures 17.6 to 17.8. These categories are based on the format of the address information in the microinstruction:

- Two addresf., fields
- Single addrc!. Field
- Variable format

The simplest approach is to provide two address fields in each microinstruction. Figure 17,6 suggests how this information is to be used. A multiplexer is provided [hal sere air destination ror both address rielthi plus I he instruction register, Based on an address-selection input, the multiplexer transmits either the opcode or one of the two addresses to the control address register (CAR). The CAR is subsequently decoded to produce the next microinstruction addro". The address-selection signals are provided by a branch logic module whose input consists of control unit flags plus hits from the control portiOn Or the microinstruction.

Although the two-address approach is simple, it requires more bits in the microinstruclion than other approaches. With some additional logic, 'savings can be achieved. A common approach is to have a single address field (Figure 17.7). With this approach. the options for next address are as follows:

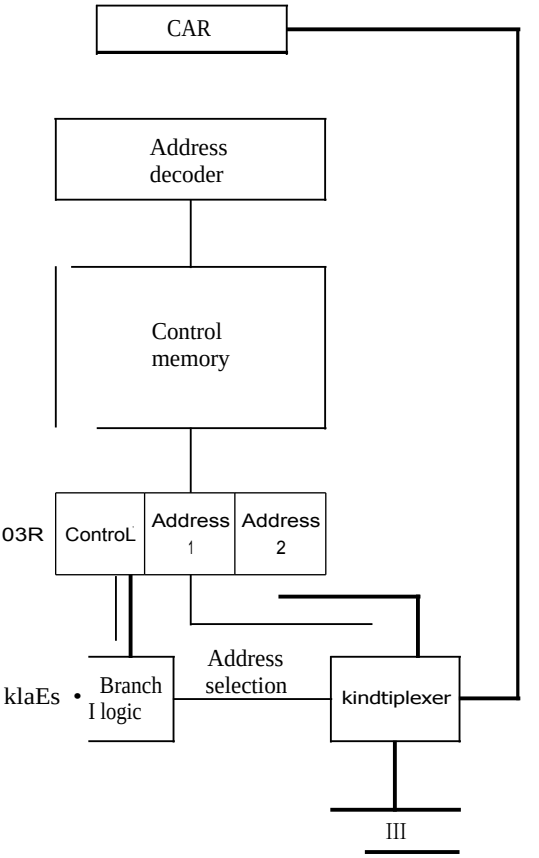


Figure 17.6 Branch Control Logic. Two Address Fields

* Add *resA* field
* Instruction register code
* Next sequential address

The address-selection signals determine which option is sele.6ted. This approach reduces Ehe number of address fields to one. Note, however, that the address fi dd soften will nut he wicfl, TM's, Eh.crt is some illaii6encv in the microinstruction coding scheme.

Another approach is to provide for two entirely different microinstruction fornais (Figure 17.8). One bit designates which formal is being used- In one format, the remaining hi Is **;ire used** to virliv;ite eon Li'01 sigrth Is. In the other formai, sonic bits drive the branch logic module, and the remaining bits provide the address. With the first format, the next address is either the next sequential address or an iiddres ', derived from the instruction register- Willi the second format, either a conditional or unconditional branch is being specified. One disadvantage of this approach is [hal one entire cycle is consumed with each branch microinstruction- With I he other

appr9ac1ics, vicldrcs;71 generation occurs as part of the same cycle as control. signal ?eneration, **control** mcmory acceSses.

The approaches just described are gcneral. Specific implementations will often involve a variation or combination of these techniques.

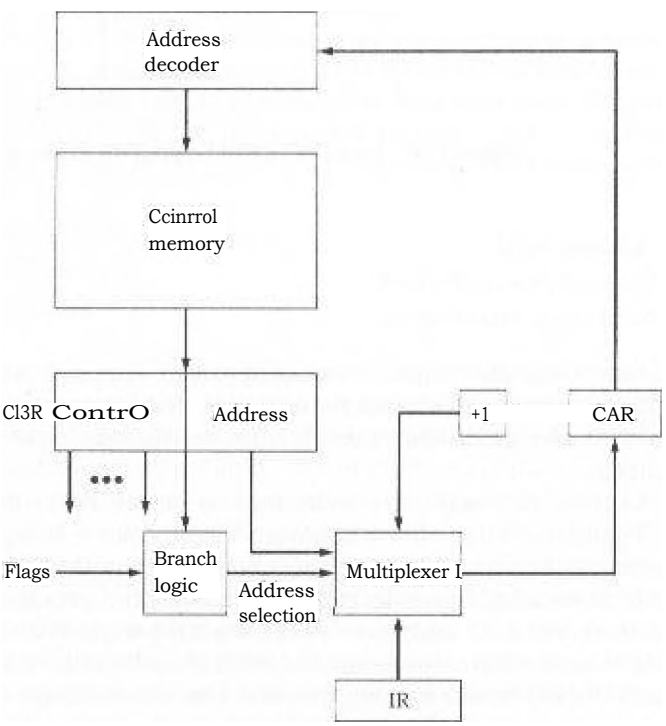
Address Generation

We have looked al the sequencing problem from the point of view of format considerations and general logic requircmcnbl. Anoi her viewpoint is io ccJrisider the various ways in which the next address can be derived or computed.

'table 17,3 lists the various address generation techniques. These can be. divided into explicit techniques, in which the address is explicitly available in the microinstruction. and implicit techniques, which require additional logic to gcno.-4l to ihe address.

We have essen tinily dealt with the explicit techniques. With a two-field approach, two alternative addresses are available with each microinstruction. Using either a single address field or a variable format, various branch instructions can be inipiernenied. A conditional branch instruction depends on the following types of information:

- AL[!
- Part of the opcode or address mode fields of the **rwichinc ims1ruction**



Figiire 17.7 13nrich Coritrul. **Sirsg142. Address Field**

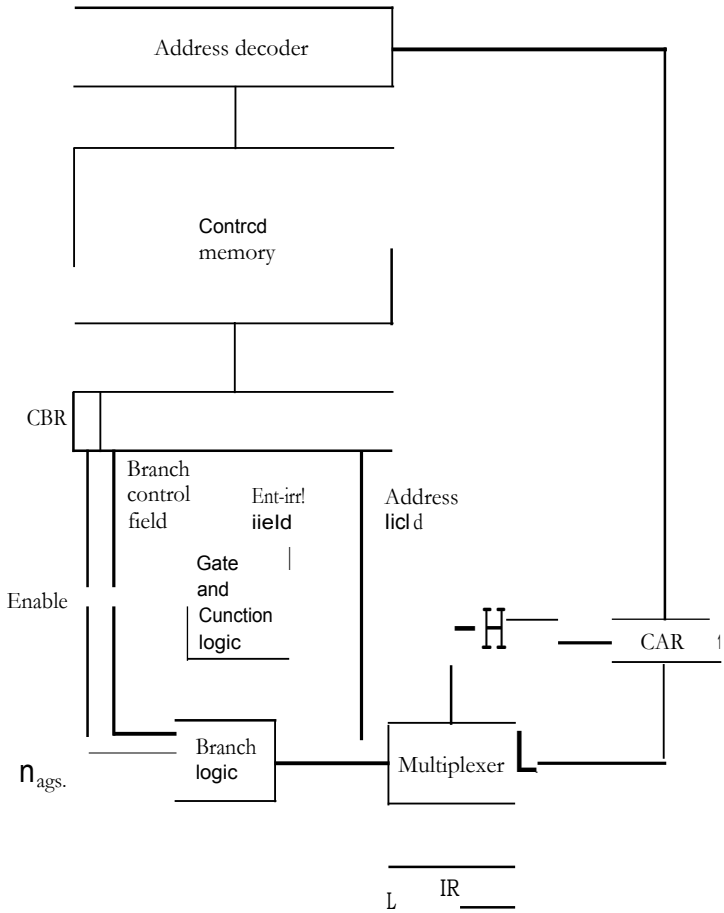


Figure *MK* Branch Control Logic, variable Format

- Parts of a selected register, such as the sign bit
- Smuts hits within the control unit

Soieral implicit techniques are also commonly used. One of these. mapping, is required with virtually all designs. The opcode portion of a machine instruction must be mapped into a microinstruction address. This occurs only once per instruction cycle.

Table 17.3 Microinstruction Address Generation Techniques

Explicit	Implicit
Two-fickl	Mapping
UncondiLioneMl branch	Addition
Conditional branch	Residual control

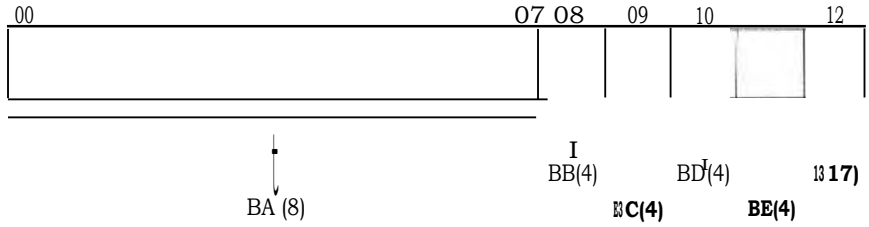


Figure **I7.9 I** BM 3033 Control Address Register

A common implicit technique is one that involves combining or adding two portions of an address to form the complete address. This approach was taken for the IBM Si360 family EI'LICK671 and used on many of the S/370 models. We will use the IBM 3033 as an example.

The control address register on the IBM 3033 is 13 bits long and is illustrated in Figure 1.7.9. Two parts of the address can he distinguished, The highest-order 8 bits (00-)7) normally do not change from one microinstruction cycle to the next. During the execution of a microinstruction, these 8 bits are copied directly from au K-bik field of the microinstruction (the BA field) into the highest-order 8 his of the control address register. This defines a block of 32 microinstructions in control memory, The remaining 5 hits of the control address register arc set to specify the specific address of the microinstruction to be fetched next. Each of these hits is determined by a 4-bit field (except one is a 7-bit field) in the current tnicroingroc-firm; the field specifies the condition for setting the corresponding bit. For example. a hit in the control address register might he set to.1 or 0 depending on whether a carry occurred on the last AL1.1 operation.

The final approach listed in Table 17.3 is termed *reyidual control.* This approach involves the use of a microinstruction address that has previously been saved in temporary storage within the control unit. For example, sorni.2 microin-struction sets come equipped with a subroutine facility. An internal register **or** stack of registers is used to hold return addresses, An example of this approach is taken on the 1.,S1-1.1, which we now examine.

LSI-11 Microinstruction Sequencing

The LSI-11 is a microcomputer version of a **PDP-11,** with the main components of the system residing on a single board. The **LSI-11** is implemented using a micro-programmed control unit [SEBE76].

The LSI-11 makes use of a 22-bit microinstruction and a control memory of 2K 22-bit words. The. next microinstruction address is determined in one of five ways:

- **Next** sequential address: In the absence of other instructions, the control unit's control addres7., register is incremented by 1.
- **°rode** mapping; At the beginning of each instruction cycle, the next microin-struction address is determined by the opeode.
- **Subroutine facility:** Explained presently.

- Interrupt testing: Certain microinstructions specify a test for interrupts. Eau interrupt has occurred, this determines the next microinstruction address.
- Branch: Conditional and unconditional branch microinstructions are used.

A one-level subroutine facility is provided. One bit in every microinstruction is dedicated to this task. When the bit is set, an 11-bid return register is loaded with the updated contents of the control address register. A subsequent microinstruction that specifies a return will cause. the control address register to be loaded from the return register.

The return is one form of unconditional branch instruction. Another form of unconditional branch causes the hits of the contro] address register to be loaded from 11 bits of the microinstruction. The conditional branch instruction makes use of a 4-bit test code within the microinstruction. This code specifies testing of various AM.' condition codes to determine the branch decision; If the condition is not true, the next sequential address is selected. If it is true, the 8 lowest-order hits of the contro] address register are loaded from 8 bits of the microinstruction. This allows branching within a 256-word page of memory,

As can he seen, the LSI-11 includes a powerful address sequencing facility within the control unit, This allows the microprogrammer considerable flexibility and can ease the microprogramming task. On the other hand. this approach requires more control Linn logic than do simpler capabilities.

17.3 MICROINSTRUCTION EXECUTION

'Ile microinstruction cycle is the basic event on a microprogrammed processor. Each eyelc is ma dc up oil' 1 wo parts: fetch and execuict. The fetch portion is determined by the generation of a microinstructitrn address, and this was dealt with in the preceding section. This section deals with die execution of a microinstruction.

Recall that the effect of the execution of a microinstruction is to generate eoril rol signals. Some of these signals control points. internal to the processor. The remaining signals go to the external control bus or other external interface. As an incidental function, the address of the next microinstruction is determined.

The preceding description suggests the organi ɛa Lion of a control unit shown in Figure 17. Ltil. This slightly revised version of Figure 17.4 emphasizes the focus of this section. The major modules in this diagram should by now he clear. The sequencing logic module contains the logic to perform the functions discussed in the preceding section. It generates the address of the next microinstruction. using as inputs the. instruction register. AL[! flags. the contro] address register (for incrementing). and the control buffer register. The last may provide an actual address. control bits, or both. The module is driven by a clock [hal determines iht timing of the microinstruction cycle.

The control logic module generates control signals as a function *of* some of the bits in the mieroin s1rudion. Ii should he clear that the format and content of the microinstruction will determine the complexity of the control logic module.

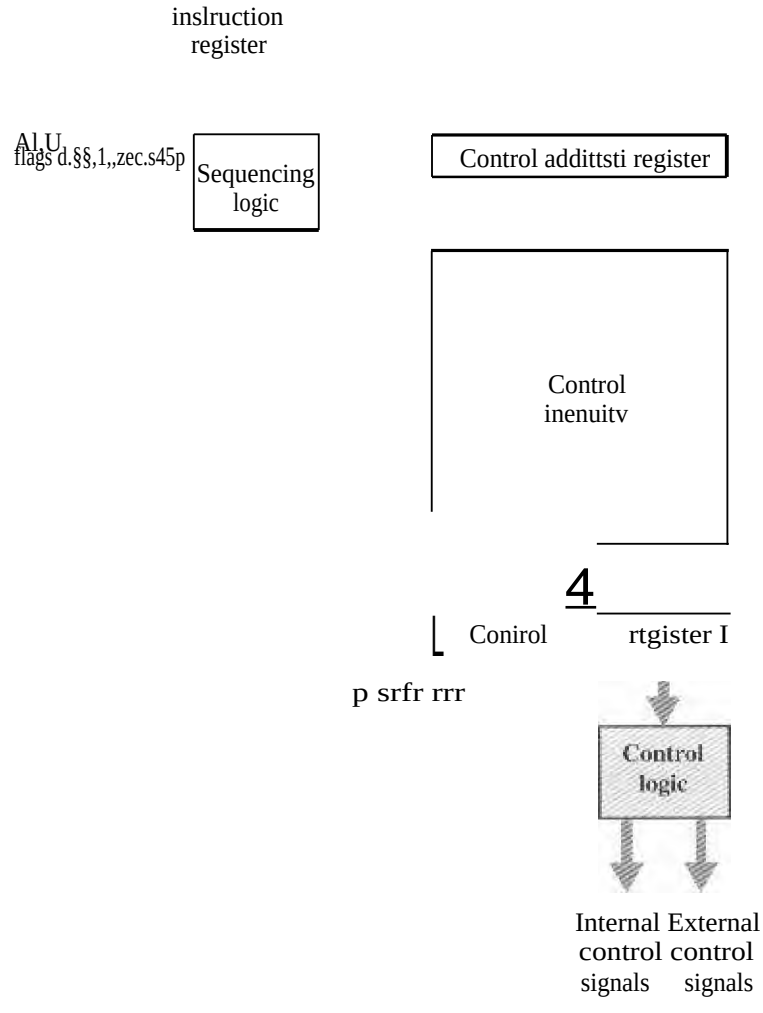


Figure 17.111 Control Unit Organinition

A Taxonomy of Microinstructions

Nlicroinstruction can be classified in a variety of ways. Distinctions that arc commonly made hi the literature include the followin4:

- Vcrticaiihorizont4i1
- * Packetkunpacked
- ■ HardIsoft microprogramming
- * Direct/indirect encoding

All of iheRe bear on the format of the microinstruction, None of these terms has been used in a consistent, precise way in the literature, However, an examination of th e pairs of qualities serves to i I lurninatc microinstruction de sign f11 k In

the following paragraphs, we first look at the key design issue underlying all of these pairs of characteristics, and then we look at the concepts suggested by each pair.

In the original proposal by Wilkes [WILK51], each bit of a microinstruction either directly produced a control signal or directly produced one bit or the next address. We have seen, in the preceding section, that more complex address sequencing schemes, using fewer microinstruction bits. are possible. These schemes require a more complex sequencing logic module. A similar sort of trade-off exists for the portion of the microinstruction concerned with control signals. By encoding control information, and subsequently decoding it to produce control signals, control word bits can be saved.

How can this encoding be. done? To answer that, consider that there are a total of K different internal and external control signals to be driven by the control unit. In Wilkes's scheme, K hits of the microinstruction would he dedicated to this purpose. This allows all of the 2^1 possible combinations of control signals to be generated during any instruction cycle.. But we can do better than this if we observe that not all of the possible combinations will be used. Examples include the following:

- Two sources cannot be gated to the same destination (e.g., C,. and C, in Figure 16.5).
- A register cannot be both source and destination (e.g.. C ; and C: in Figure l(15).
- Only one pattern of control sig,nals can be presented to the ALI] at a time.
- Only one pattern of control signals can be presented to the external control bus at a time.

So. for a given processor, all possible allowable combinations• of control signals could be listed, giving some number Q < 2' possibilities. These could be encoded with log.:Q bits, with (log,Q) < K. This would be the tightest possible form of encoding that preserves all allowable combinations of control signals. In practice, this form of encoding is not used, for two reasons:

- It is as difficult to program as a pure decoded (Wilkes) scheme. This point is discussed further presently.
- It requires a complex and therefore slow control logic module_

instead, some compromises are made. These are of two kinds:

- More bits than are strictly necessary arc used to enaxle the possible combinations.
- Some combinations that are physically allowable are not possible to encode.

The latter kind of compromise has the effect of reducing. the number of bits, The net result. however, is. to use more than lo2. $_3$Q bits.

In the next subsection, we will discuss specific encoding techniques. The remainder of this subsection deals with the effects of encoding and the various terms used to describe it.

Based on the preceding, we can see that the control signal portion of the microinstruction format falls on a spectrum_ At one extreme., there is one hit for each control signal; at the other extreme, a highly encoded format is used. Table 17.4

Table 17.4 The Microinstruction Spectrum

Characteristics	
I:ne.ncodEd	Hit;h1!.. encoded
Many bits	hcw hitF.
I) snil..r.d or flaTkiVi2re	Artrculul LW ‹f 11 1.'511 alr
Difficult hi program	to program
Curi.CUtrency fully exploited	COrICUTreLICy 110i fully.cxploitEd
Link: or 110 centre] 14.50.4.-	Curriph2N. contra 10
r as! uxuruLiun	Sl ow e'xecLILi{riL
perfOrlIlariee	C}ptimizR programming
Termini to	
Unpacked	
HorrAorikal	
Kira	Soft

shows that other characteristics of a mieroprogrammed control unit tlso fall a]ong ..pectritin and that these spectra are, by and large, determined by the degree-of-en cod i tig spectrum.

The second pair of items in the lable is rather obvious. The pure Wilkes sehen-Ki. will require the M051 bits, ShiDUILI 411M) he apparent that this extreme presents the most detailed view of the hardware. Every control signal is individually controllable by the microprouammer. Encoding is done in such a way as to aggregate funciion,s or resources, so chat the inicroprogrammer is viewing the processor at a higher. lesf., detailed level. Furthermore, the encoding is designed to 42-.ise lh microprogramming burden. Again, it should be clear Thai the Task c.i.funderstandin and orchestrating the use of all the control signals is a It one As was mentioned, one of the consequences (J1 encoding, typically, is to prevent the use of cep. Min otherwise aiiown hie combinations.

The preceding paragraph discusses microinstruction detlign from the microprogrammer's point of view. Bill the degreC of cricodi lig also can be viewed from its hardware effeeLS. Wilk it purl: uncncoded format. little- or no decode logic is needed: each bit generates a particular control signal. As more compact and more aggregated encoding schemes are used, more complex decode logic is needed. This, in WM, may affect performance. More time is needed to propagate signals through the gates of the more complex control logic module. Thus. the execution of encoded microinstructions takes longer than the execution 01 uncrtcoded ones.

Thus, all of the charaetcYristics iisied in Table 17.4 fall a]ong a spectrum of design Kinatv.gie. In general, a design that falls toward the left end of the tipectrum is intended to optimize the performance of the control unii. Designs iownrd the right end are more concerned with oplimizing the process of microprogramming. Indeed, microinstruction ?;(AS TIC2.411- the right end of the spectrum look very much like machine instruction sets. A good example of this is lhe LS1-1 1 design, described later in this section. Typically. when the objective k simply to implement a control unit, the design will he near the left end of the spectrum. The IBM 3033 design, dis-

cussed presently. is in this category. As we shall discuss later. some systems permit a variety of users to construct different microprograms using the same microin• struction facility. In the latter cases. the design is likely to fall near the right end of the spectrum.

We can now deal with some of the terminology introduced earlier. Table 1'1.4 indicates how three of these pairs of terms relate 10 the microinstruction spectrum. In essence, all of these pairs describe the same thing but emphasize different design characteristics.

The degree of packing relates to the degree of identification between a given control task and specific microinstruction bits. As the hits become more *parked,* a given number of bits contains more information. Thus, packing connotes encoding. The terms *horizontal* and *vertical* relate to the relative width of microinstructions. [SIEW82] suggests as a rule of thumb that vertical microinstructions have lengths in the range of 16 to 40 bits, and that horizontal microinstructions have lengths in the range of 40 in 100 bits, The terms *hard* and *soft* microprogramming are used to suggest the degree of closeness to the underlying control signals and hardware layout. Ilard microprograms are generally fixed and committed to read-only memory. Soft microprograms are more changeable and rn c suggestive of user microprogramming.

The other pair of terms mentioned at the beginning of this subsection refers to direct versus indirect encoding, a subject to which we now turn.

Microinstruction Encoding

In practice, microprogrammed control units ;ire not designed using a Rare. unencoded or horizontal microinstruction format. At least some degree of encoding is used to reduce control memory width and 10 simplify the task of microprogramming.

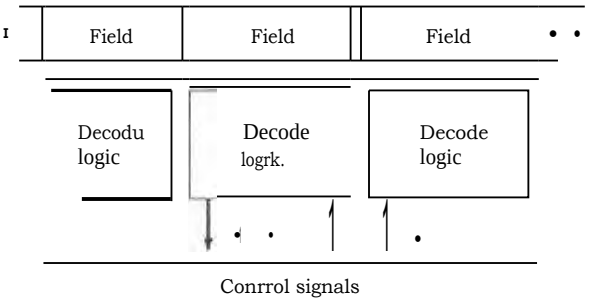
The basic technique for encoding is illustrated in Figure 17.11a. The microinstruction is organized as a set of fields. Each field contains a code, which, upon decoding, activates one or more control signals.

Let us consider the implications 0r this layout. When the microinstruction is executed, every field is decoded and generates control signals. Thus. with N fields, N simultaneous actions arc specified. Each action results in the activation of one or more control signals. Generally, but not always. we will want to design the format *so* that each control signal is activated by no more. than one field. Clearly, however, it must be possible for each control signal to be activated by at least one field.

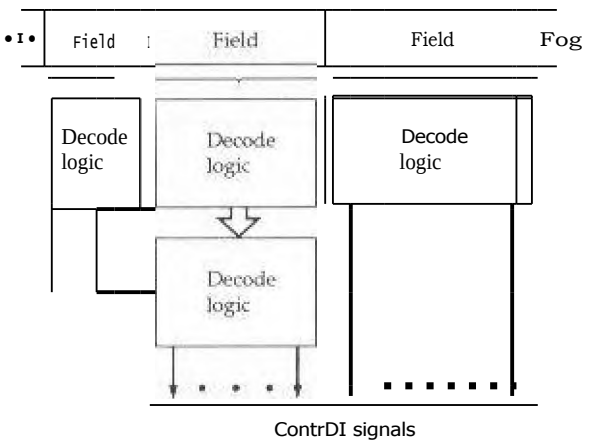
Now consider the individual field_ A field consisting of *L* bits can contain one of 2 codes, each of which can be encoded to a different control signal pattern_ Because only one code can appear in a field at a time, the codes are mutually exclusive, and, therefore, the actions they cause are mutually exclusive,

The design of an encoded microinstruction format can now he stated in simple terms:

- Organize the format into independent fields, That is, each field depicts a set of actions (pattern of control signals) such that actions from different fields can occur simultaneously.
- Define each field such that the alternative actions that can he specified by the field are mutually exclusive. That is, only one of the actions specified for a given field could occur at a time.



fa) Direct encoding



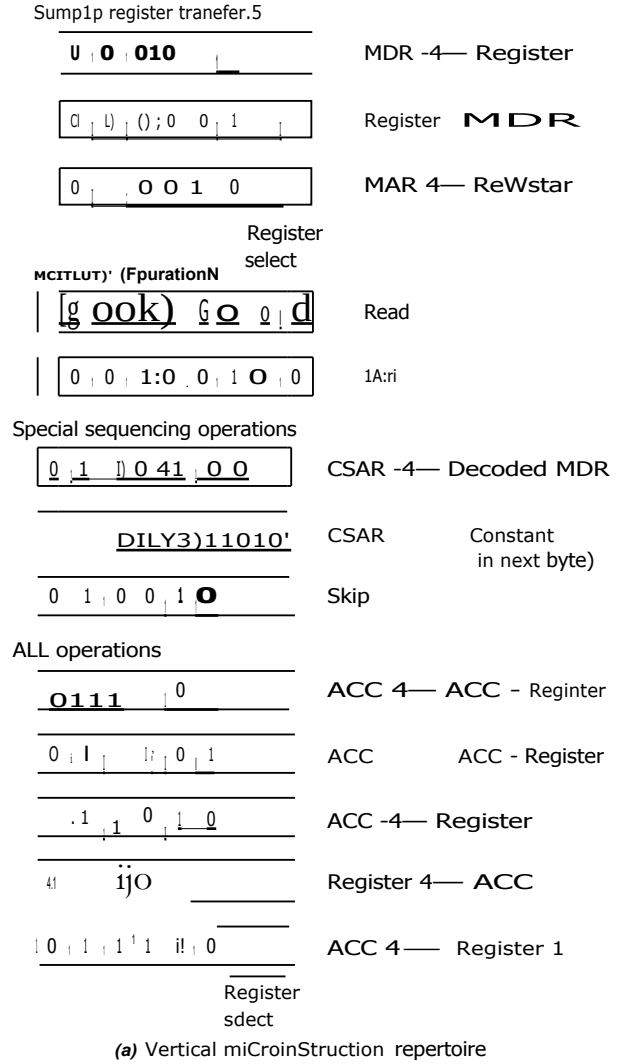
(b) indirect encoding

Figure 17.11 NJ icroin1Lruction Encoding

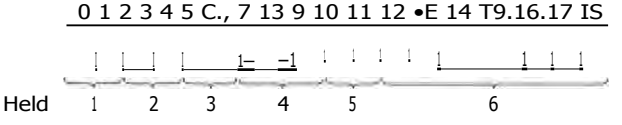
Two approaches earl he taken to organizing the encoded microinstruction into fields: functional and resource.. *The fitntlif»tal etwoffing* method identifies functions within the machine and designates ficid5 by function type- For example., if various sources can he used for transferring data to the accumulator, one field can be designated for this purpose. with each code specifying a different source. *Resource encoding* views the machine as consisting of a set of independent resources and devotes one field to each (e.g._ f.'(l, memory, Alt:).

Another aspect of encoding is whether it is direct or indirect (Figure 17. lib). With indirect encoding, one field is used to determine the interprcia Lion of another field. For e-qnysple, consider an Al ,T,J Thal is capable of performing eight different arithmetic operations and eight different shift operations. A **1-hit** field could be used to indicate whether a shift or arithmetic operation is to be used a 3-bit tick] would indicate the operation. This technique generally implies two levels of decoding, increasing propagation delays.

Figure 17.12 is a simple example of these concepts. Assume a processor with a single accumulator and several internal registers. such as a program counter and a temporary register for A19.1.! input. Figure 17.12a shows a highly vertical format. The



(a) Vertical miCroinStruction repertoire



Field definition

- Register transfer
2 - (vIernory operation.
3 -Sequencing operation
4 - ALL: operation
- Register selection
fi - Constant

(b) Nl,r]i.onrel microinstruction format

Figure 17.12 Alternative Nlicroinviroci ion l'orinats for a Simple fvlachine

firm 3 bits indicate the type of operation, the next 3 encode the operation, and the final 2 select an internal register, Figure 17 12b is a more horizontal approach, although encoding is still used. In this case, different functions appear in different fields.

LSI-11 Microinstruction Execution

L.S1-1,1 [SEBE76] is a good example 01' a vertical microinstruction approach. We look first at the organization of the control unit, then at the. microinstruction format,

LS1-11 Control Unit Orguni,ation

The LSI-1 1 is the first member of the PDP-1 I family that was offered as a sin• gle-hoard processor. The hoard contains three LS1 chips, an internal bus known as the *microinstruction bus (MIB),* and some additional interfacing logic.

Figure 17.13 depicts. in simplified form, the organization of the 1.S1-11 processor, The three chips are the dicta, control. and control store. chips. The data chip contains an 8-bit ALU, twenty-six N-bit registers, and storage for several condition codes. Sixteen of the registers are used to implement the eight 16-hit general-purpose registers of the PDP-11. Others include a program status word, memory address register (MAR), and memory puffer register. Because the ALU deals with only 8 hits at a time, two passes through the AIA.: are required to implement a 16-hit PDP-I 1 arithmetic operation. This is controlled by the microprogram.

The control store chip or chips contain the 22-hit-wide control memory. The control chip contains the logic for sequencing and executing. microinstructions, It

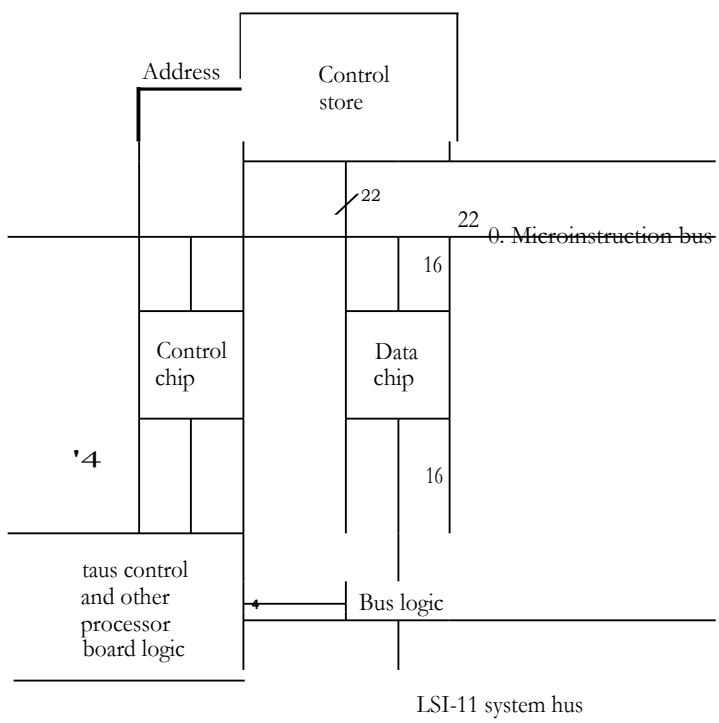


Figure 17.13 Simplified 131ock Diagram cif 1 he I .S I I Prucolor

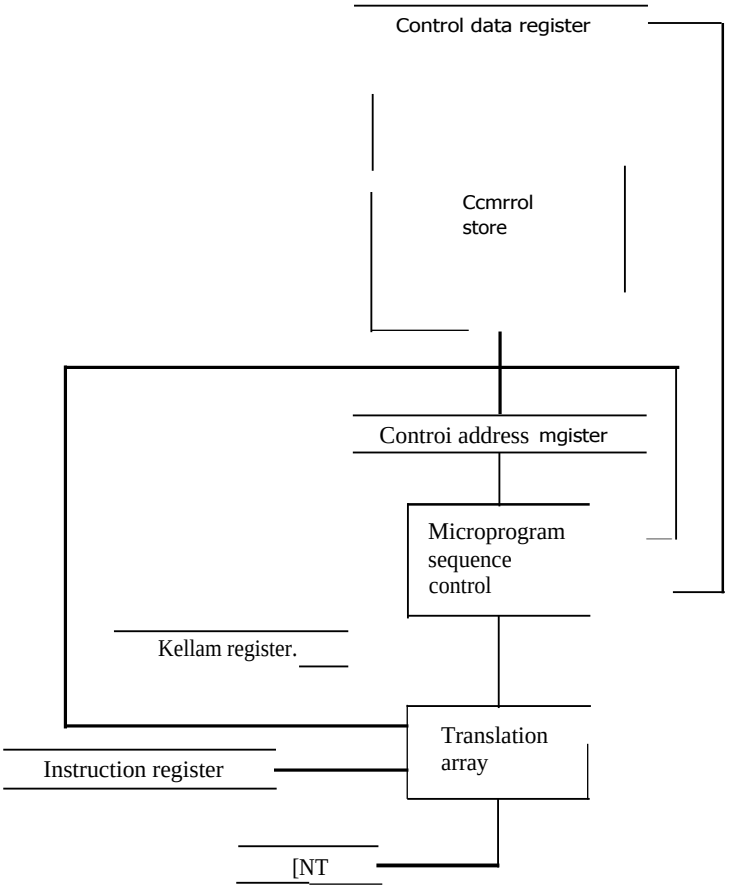


Figure 17.14 OrgarsiAai kin 01 the LSI-11 Control unit

contains the control address register, the control data register, and a copy of the machine instruction register.

.1he MIB ties i1] the components together. During microinstruction fetch, the control chip generates art 1]-bit address onto the NUB. Control store is accessed. producing a 22-bit microinsLruction, which is placed on the III. The low-order 16 bits go to the data chip, while the low-order 1K hits go to the control chip. The high-order 4 bits control **special** processor board functions.

Figure 17.14 provider; a still simplified bi.LE more deLii led look at the LS.1.-/ control unit: The figure ignores individual chip boundaries. The address sequencing scheme described in Section 17.2 is implemented in iwo modules. Overall sequence control is provided by the microprogram sequence control module. which is capable of incrementing the microinstruction address register and performing unconditional branches. The other forms of address calcuiation **are** carried out by a separate rrans[alion array. This is a combinatorial circuit that a.cnerates an address based on the microinstruction, the machine instruction, the microinstruction program counter, and an interrupt register.

Thu 1ransiation array comes into pla!,' on the following occasions:

- The opcode is used to dcierminu thy. sta rt or
- At appropriate ti mes, address mode bits of the microinstruction are tested to perform appropriate addressing,
- Interrupt conditions are periodicaLly tested.
- Conditional branch microinstructions arc cvgillialled,

LSI- I I M iCTIF **instructi4F** II Format

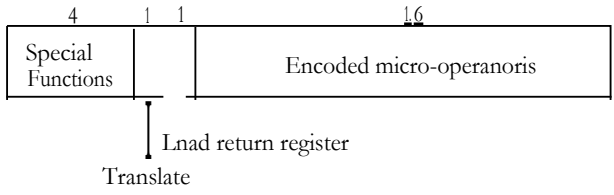
The LS1-I L uses 4i n ex I remely vertical microinstruction format, which is oak *22 hits wide,* Thu microinstruction set strongly resembles the PDP-1 I machine. instruction set that it impLements. This design was inlendcd optimii.e the performance of the control unit within the constraint of a vertical, easily programmed design. 'rabic 17.5 lixix some of the 1_,S.1-1 1 microinstructions.

Figure 17.15 shows the 22-bit LSI-11 microinstruction formai. ' ['he high-order 4 bits control special functions on the prom.ssor board. The translate bit enables the [1-41risth Lion array to clic **k Cor** pending interrupts. The load return register hil is used At the cad of a Mier0rOUtifie to cause the next microinstruction address Lo ire Loaded from the return register.

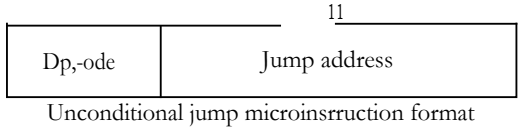
The remaining 1.6 bits 41TC Wit:1i for highly encoded micro-operations. **ThC:** far• mad is much like a machine instruction, with a variable-length opco4.1u arid one or more operands.

Table 17.5 Some LSI 11 Microinstructions

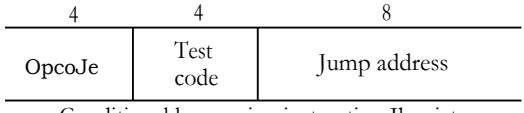
Arithmetic Operations	Ocneral Operations
Add ward (byte, 111Crril	MOV
Tem word;'Fete, litcrak)	Jump
fricrethem word (byte) 1}v I	RuurEs
Increment word (1v}.1.0 by 2	Conditioital jump.
Ni::garc wnrd (hym)	Set (reset) L1a,9,3
Canditiimally irturernorn Idcurcineisi) by L'	Load G kiw
C:unditionall:v add word (byre)	MC.)V ward
.Add word (hyte) mat cnrry	Ciperatiuns
Canditi.ursull:y add 131giLs	Input word (bloc.)
Sul tract word (byte)	inpui $taltEs word (kiy(vj
Compare word (byw. literal)	I<{m4.1
S'ubtracl ward { hyi C} with carry	\Wrilu
DeCi'd'Elierit ward (byte). by 1	ker.H.1 (wrili2) Lind loci-ern:ant word {**byte)** i
Logical Operailons	**Read Ixyritti** i Lind iiici:2111e1}L word {**byte)** by
AND word (byre, titeral)	**Rid** (wi ire) ocknowlodge
Tr:sr word (hyi,7)	Ouipul. **word** {**byte, stat** us?
OR word iFylc)	
Exclwiwe-014. (bys.i3)	
Flit c.1....:ar word (hyic).	
Shift .0,151a fhytcj right (112f1.) with (wilhow i iii 1	
Coittyletisertt wind (11).1e)	



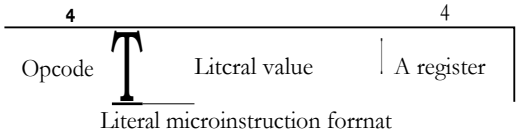
(a) Format d the full] 1.,9I-n microinstruction



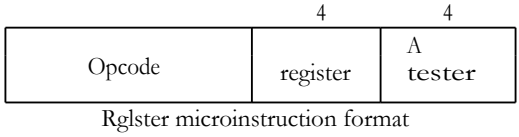
Unconditional jump microinsrruction format



Conditional lump microinstruction Ibrrriat



Literal microinstruction fornat



Rglster microinstruction format

(b:i Format *of* the encndaci part of the L51-11 inicronstruction

fligunr. **k7.15 L.S.1-1 1** Microinstruction Format

IBM 3033 Microinstruction Execution

The standard I.BM 301 control memory consists of 4K word ';. Thc first half of these (11.000-071'1^{7}) contain 108-bit microinstructions, what.: the remainder (0800-1FFE) are used to store 126-bit microinstruclions. **rhe** format is depicted **in** Figure 17.16. Although this is a rather horizontal format, encodin.g is still extensively used, The key fields of that format ire summarized in Table

The **A Lti** operates **on inputs** from four Lic.di -Lii.ed, non-user-visible registers, A, B. C. and D. The microinstruction format contains fields for loading these rcaisten.; ['Tom user-visible registers_ performing an ALU **:ind specifying a user** visible register for storing the result, There arc also Ileitis for loading and storing data between registers and memory.

The sequencing mechanism for the **IBM 3033** was cliscursscd in Section 17.2.

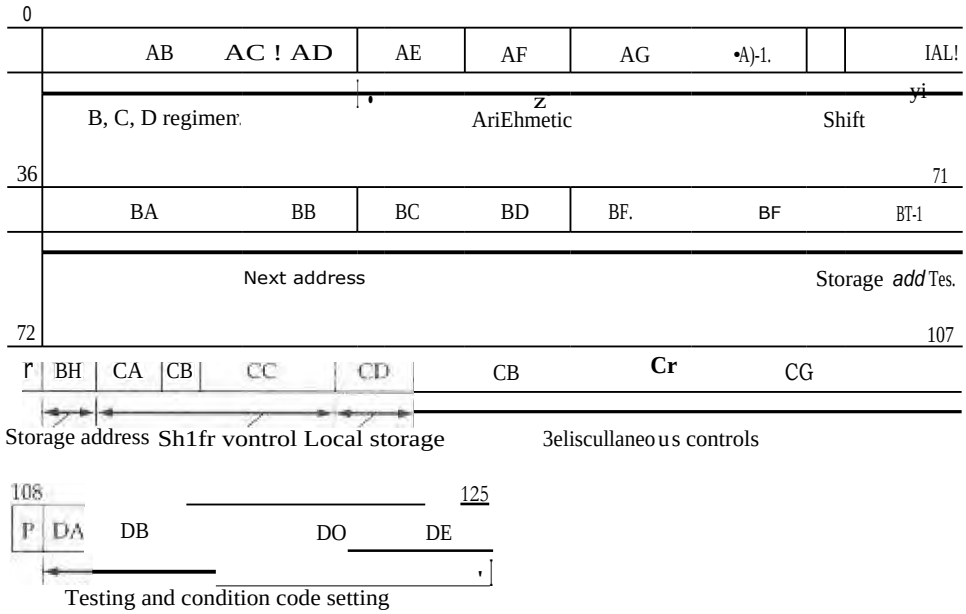


Figure 17.16 1BM. 3113 3 Microinstruction Format

Table 17.6 IBM 3033 Microinstruction Control Fields

ALL Control Fields	
AA(3)	Load A register from uric: 01 data registers
ARO)	Load B register from one of data registers
AQ)	Load C' reiister 1:11)(11 011 12 **cf** !LILL registers
ADO	1.cukd I) registur From 4.5(u.'e dHkarugislQr!::
A F.0.1	Rouse. specified A hits to ALIJ
AF1:4)	Rout. specified 13 bilk lo ALL!
AGN	Spc.rifin. AL.1. drribm,•111.9. cip2ralton nn A Input
AH{I)	Specifies ALL; on B input
A..1(1)	Speeilifies D or 13 input ι . ALL: B side
AK(4)	Rciat4,. arithmetic outpuL Li, shifter
C1-3(1)	*Activ.m sinker*
C' '(5)	Specifies logical and carry functions
C.E(7)	Spcciiitn **;hilt** :1 mount
CA(3)	I .ond F Ngidor

Sequencing and Branching Fields	
Al.(1 I	Era/ operaiiO31 and perform branch i high-order hits 01417) N' c PEI kr() I Add ro5 register
/ili(ii	.Spec corid it10 ii **for** Et3ts Lag hti g **of control address** re gist er Specifics cortdilhin for seltiag **of control address** regist er
BED(4)	Specifies condiii(in fcw scuL **irl** g hit 10 of ciin **trial** n:gis tor
B144t	SpecifieN coaiithon rod. sckting his 11 of conirol addrcs, Ngisic r
B[4)	Speciilos condition for **swain 1)1i 12** of control **address** 113gi stet.

The Texas Instruments 8800 Software Development Board (SDB) is a mieroprogrammable computer card. The.system has a wrritable control store. implemonied in RAM rather than ROM. Such System does not IleilieVi2 the speed or density of a microprogrammed sysleln with a Ram control store. However, it is useful Yor developing protoiypes and for educational purposes.

The 8800 S.D.8 er_Insistii of the following components {Figure 17.17}:

- Microcode rueniory
- iCIOSC([Uen CCF
- 32-bit A 1.L.
- Floating-point and integer processor
- Local data memory

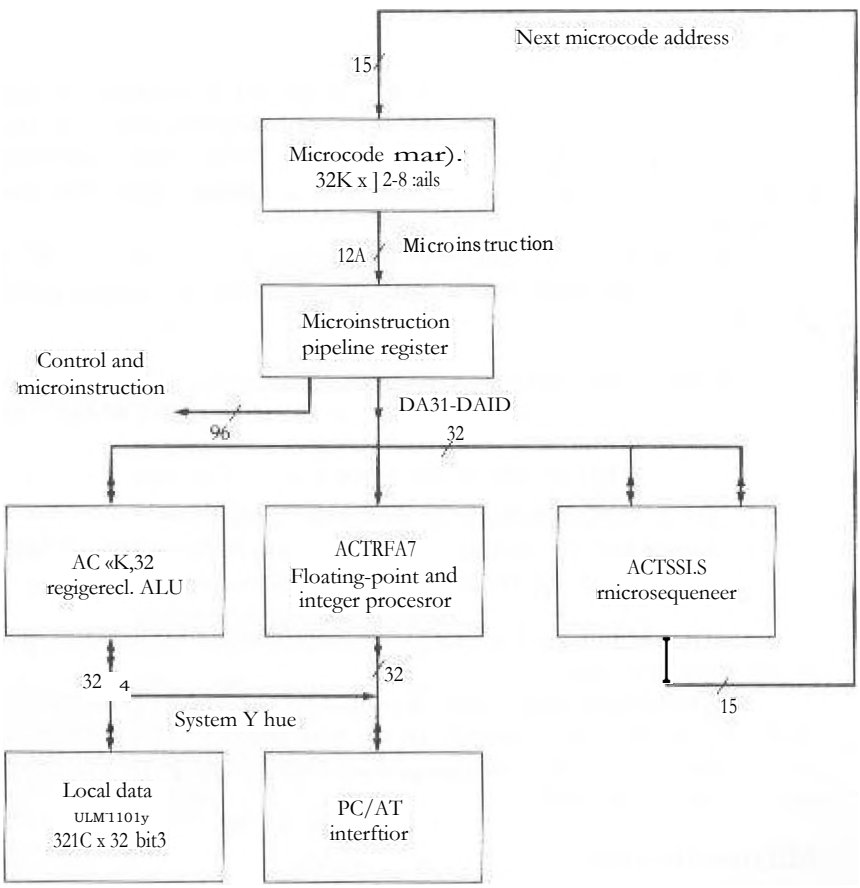


Figure 17.17 TI 8800 Block Diagram

Two buses link the internal components of the system. The DA bus provides data from the microinstruction data field to the ALL, the floating-point processor, or the mierosequencer. In the latter case, the data consists of an address to he used for a branch instruction. The bus can also he used for the ALL or microsequencer to provide data to other components. [he System Y bus connects the All) and floatine-point processor to local memory and to external modules via the PC interface.

The hoard fits into an IBM PC-compatible host computer. The host computer provides a suitable platform for microcode assembly and debug.

Microinstruction Format

The microinstruction format for the 8800 consists of 128 bits broken down into 3U functional fields, as indicated in Table 17.7. Each field consists of one or more bits, and the fields are grouped into five major ca tegories:

- Control of board
- ,S847 floating-point and integer processor chip
- 8832 registered ALU
- 8818 microscquencer
- WCS data field

As indicated in Figure 17.17, the 32 bits of the WCS data field are fed into the DA bus to he provided as data to the ALM, Floating-point processor, or microsequencer. The other % bits (fields 1-27) of the microinstruction are control signals that are fed directly to the appropriate module. For simplicity, these other connections are not shown in Figure 17.17.

The first six fields deal with operations that pertain to the control of the board. rather than controlling an individual component. Control operations include the following:

- Selecting condition codes for sequencer control. The first bit OF field I indicates whether the condition nag is to be set to I or 0, and the remaining 4 bits indicate which flag is to be set.
- Sending an 110 request to the PCIAT.
- Enabling local data memory readlwrite operations.
- Determining the unit driving the system Y bus. One of the four devices attached to the bus (Figure 17.17) is selected.

The last 32 hits are the data field, which contain information specific to a particular microinstruction.

The remaining fields of the microinstruction are best discussed in the contest of the device that they control. In the remainder of this section, we discuss the microsequencer and the registered ALU, The floating-point unit introduces no new concepts and is skipped.

Mic.!rosequencer

The principal function of the 8818 micmsequeneer is to generate the next microinstruction address for the microprogram. This 15-hit address is provided to the microcode memory {Figure 17.17).

Tale 17.7 TI 88(X) **Microinstruoriors Formal.**

Field Number	Number of Bits	Descrip1i on
		Control of Board
L	.5	Select condision code input
2	I	krobleiclisnl ilt. exlernid 1.:0 r:LNAL.teS1.i.i.,.n.al
	2	Enable,:disable local data memory road/writ c i5perkiliotts
4	1	Load status/do no load status
5	2	I1eurnunci unit driving Y b.us
6	2	Dett.rrrtine unit driving DA bus
		8847 Iiinaiing Point ttnd Integer Procesv,ing Chip
	1	rcgi SI CT COT1LTD: LECICk. 111.1 not cluck
		iitt}St significant or loins(5/g113 riCA L hiss lor Y bus
		C register claim ScrUTCC:
10	4	Seteet 1E F E or FAST mode ifif ALL' and MUL
1 1	8	Scti...keL.:.:ources for chits operands; RA rcgiRtu s. FOR registl2r.q, P regi3ter. 5 regi3ter. C register
12	1	R.1:1.R:fiictcr Cn111.1 nl clock, do not rIoL.14.
13	1	k A re isIerssLro1:cicxk,Llo liras clock
14	'2	DALu source uoirllirriation
15	.2	Enable...disable papclinc registers
1.6	1 i	4 7 AU; runclion
		P832 Registered ALL!
17	2	Wriic unable/disable itnizi output to Ec I ucLe d i eginer: most SIgnificortl, tthlr, kirst signilioini hHII
Ix	2	**Sukci re.d.istEr** filu dirLa source.: DA bus, []13 hos. Al,[I Y 11[1X output s....sieni Y bus
19	3	Shift instruction **modther**
2U	I	**Carry in:** lorcc.. do run el5rce
..i.	2	SgI ALI? Ilurtiveat ion mode:: 32. *16,* Or hits
22	2	Select inpui to .9 niti..Ilp...l....Nor: I'el2iRR::T Ilk:, 1)13 bus, MQ rueister
23	I	S.c.lcct inpuE to R rnolli.picx UT:. rogiEM lilt., DA bus
24	•fi	:iult.ci rc.gistur in tile C for WY.i Le:
25	6	Select register iII rile 8. tbi read
26	Il	Seleci regisior in tilt A for wril.c
27	g	ALI i funcIrcin
		8818 kliernseqruencer
		Control input %gunk to i1,c M g
		PiCS Data Field
	16	Most siFnificaitil Ellis or wribiblu coritrcil store data field
		signiRcunt hits cif writul-rl u C1.51111.31 store data Reid

The next address can be selected from one of live sources:

1. The microprogram counter (MP() register, used for repeat (reuse same address) and continue [increment address by 1) instructions,
2. The. stack, which supports microprogram subroutine calls as well as iterative loops and returns from interrupls,
3. The DRA and DRB ports, which provide two additional paths from external hardware by which microprogram addresses ein be generated. These two purls 4lI'C conneeLed. Lo the most significant and least significant 16 hits.. of the DA bus, respectively. This allows the microsequencer Io obtain die next instruction address from the WCS data field ()Mlle etirrem microinstruction or from a result calculated by the ALI:,
4. Register counters RCA and RCB, which can be used for additional address storaae.
5. An external input onio the bidirectional Y port to support external interrupU..

Figure 17 ,18 is a logical block diagram of the 8818. The device Qunsisls or the following principal functional groups:

- A l6-bit microprogram counter (**NIPC)** con wiling or a rqzister and an incrementer
- Two register counters- RCA and RC.13, for counting loops and iteratiorm storing branch addresses, or driving external devices
- A 65-word by L.6-bit stack. which allows microprogram subroutine calls and interrupts
- An interrupt return register aitd Y output enable for interrupt processing at the microinstruction level
- A Y output multiplexer by which the next 4iddress can be selected from MPC, RCA. RCB, external buses URA and DRB, or the stack

Registers/Counters

The registers RCA and Rai may he loaded from the DA bus, either from the current mieroinstrue1ion or irons the output of the ALU. The values may be used as counters to control the flow of execution and may be auLornLiticaRy decrcmeiited when accessed. The values may also be used as microinstruction addresses to be supplied to the output multiplexer, Independent control of both regisn,:rs during a single microinstruction cycle is supported with Ihc exrcption of simultaneous decrement of both registers,

Stack

The stack allows multiple lcvels of nested calls or interrupts, and it can he used Lo support branching and looping. Keep in mind that these opera Li ons rcl'er to the control unit, not the overall processor,. and that the addresses involved are those of microinstructions in the control memory,

Six **stack** operations are possible;

1. Clear, which sets the stack pointer to zero, emptying Ike stack
2. Pop, which decrements the stack pointer

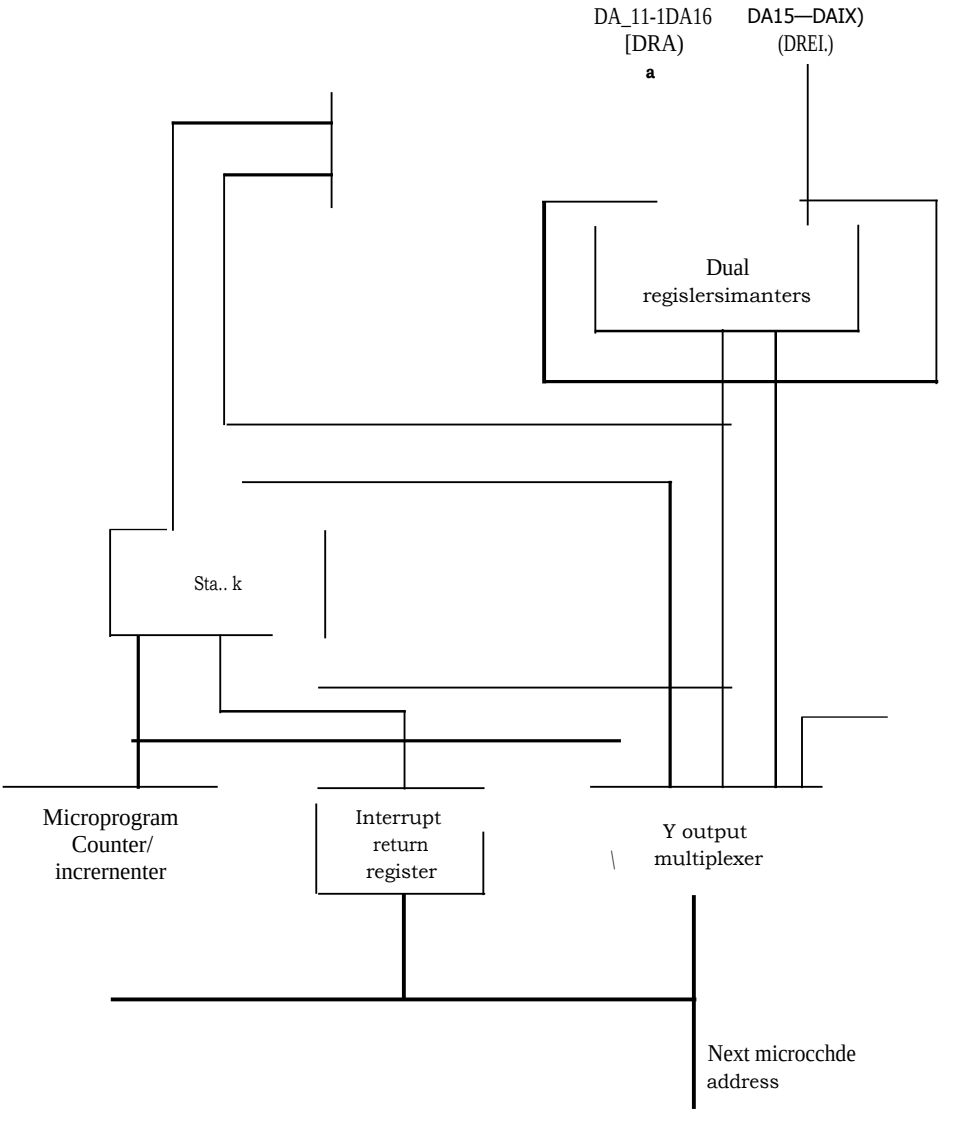


Figure 17.18 TI SSA Microsoquencer

3. Push. which raas the contents of the. MPC, interrupt r01,1171 register, or DRA bus onto the stack and increments the stack pointer
4. Road, which makes the addrc.m. indicated by the read pointer available al the oulpul multiplexer
5. Hold, which comes the address of the stack pointer to rcrnain unchanged
6. Load stack poinlc.r, which inputs the seven [east signi]'icant bits of DRA to the slack pointer

Control of Microsequencer

hi mierosequencer is controlled primarily by the 12-bit field of the current microinst ruction. field 28 (Table 173). This field consists of the following subfields:

- OREL (1 bit): Output select. Determines which value will be placed on the output of the multiplexer that feeds into the DRA bus (upper left-hand corner of Figure 17.18). The output is selected to Time from either the stack or from register RCA. DRA then serves as input to either the Y output multiplexer or to register RCA.
- SELDR (1 bit): Select DR bus. if set to 1, this hit selects the external DA bus as input to the DRA/DRB buses. if set to 0. selects the output of the DRA multiplexer to the DRA bus (controlled by OSEL) and the contents of RCB to the DRI3 bus,
- ZERO1N (1 bit)• Used to indicate a conditional branch. The behavior of the mierosequeneer will then depend on the condition code selected in field I (Table 17.7).
- RC2—RCO (3 bits): Register controls. These bits determine the change in the. contents of registers RCA and R(.13. Each register can either remain the same. decrement, or load from the DRAIDRB buses.
- S2—S0 (3 hits): Stack controls. These bits determine which slack operation is to he performed.
- NIUX2—MUXO: Output controls. These bits, together with the condition code if used, control the Y output multiplexer and therefore the next microinstruction address. The multiplexer can select its output from the stack, DRA, DRB, or MPC.

These bits can be Net individually by the programMer. However. this is typically not done. Rather, the programmer uses mnemonics that equate to the hit patterns that would normally he required. Table 17_8 lists the 15 mnemonics for field 28. A microcode assembler converts these into the. appropriate bit patterns.

As an example, the instruction 1NC88181 is used to cause the next microinstruction in sequence to he selected, if the currently selected condition code is 1. From Table 17.8. **we have**

INN 8=81 = OC.00`..'021111:

which decodes directly into

- OSEL = 0: Selects RCA as output from DRA output MU X: in this case the selection is irrelevant.
- SELDR = 0: As defined previously; again. this is irrelevant for this instruction.
- ZEKOIN = 0: Combined with the value for MUX, indicates no branch should he taken.
- H = 000: Retain current value of RA and RC.
- S = .111: Retain current state of stack.
- MLA = 110: Choose MPC when condition code DRA when condition code = O.

Table 17.8 SN I g M icrosequencer' icroiristritetion Bits {Field 28}

Mnemonic	Value.	Description
RS 114818	00300006:1110	Reset in SU uction
El RAM181	01104011 [000	Branch to DRA instruction
BRAiSIS0	01000011 I I 10	Branch Lo DRA instruction
INC881.81	000000111110	Continue instruction
INCSRES41	001000001000	Con tinue instruction
CA I.M1R1	0101A01 mom	31iiirr) t o subroutine at address Speeirwri by DRA
CALKH1K1	oh otX101411110	Jump rn subroutine at address Specified by DRA
k 1-.:1 NM, Zi	01)0000011 0 1 0	Return from subroutine
VLSHWilti	01104X10] 10111	Push interrupt return address onto stack
POP8818	1000210010000	Retort from interrupt
LOADDRA	00001011 11 10	Load DRA counter from DA bus
LOAD DR B	000110111110	Load DRB counter from DA bus
LOAD DRA B	400110111100	LA }NJ D R Am R [i
DECRDRA	()LOW] I 1 1I00	Decrement DR A coon tar and branch it not zero
DFCRDRB	010101111 WO	I)ecronc.nt DR B counter ;Ind branch ii not vcr()

Registered ALU

The 8832 is a32-hit ALU with 64.registers that can he configured to operate as four 8-bit ALUs. two 16-hit ALA:s, or a single 32-bit ALA.:.

The 8832 is controlled by the 39 hits that make up fields 17 through 27 of the microinstruction (Table 17.7): these are supplied to the ALL: as control signals. In addition, as indicated in Figure 17.17. the 8.832 has external connections to the 32-hit DA bus and the 32-hit system Y bus. Inputs from the DA can be provided simultaneously as input data to the 64-word register file and to the ALI! logic module. Input from the system Y bus is provided to the ALU logic module. Results of the ALU and shill operations are output to the DA bus or the system Y bus. Results can also be fed back to the internal register file.

Three 6-bit address ports allow a two-operand fetch and an operand write to he performed within the register file simultaneously. An NiQs.hifter and MO register can also be configured to function independently to implement double-precision 8-bit, 16-bit, and 32-bit shift operations.

Fields 17 through 26 of each microinstruction control the way in which data flow within the 8832 and between the 8832 **and** the external environment. The fields are as follows:

17. **Write Enable. These two hits specify** write 32 hits, or 16 most significant bits. or l6 least significant bits. or do not write into register file. The destination register is defined by field 24.
18. **Select Register File Data Source. If a write is** to occur to the register file, these two bits specify the source: DA **bus, DR bus. ALIJ** output, or system Y bus.

I% **Raft** Instruction Modifier. Svceifies options concerning supplying end fill and reading bits [hat are shifted during shift instructions-

20. Carry In. This bit indicates whether a bit is carried into thy ALL: for this oKt• a lion,

21, **ALU Configuration Mode. The** 8832 can he configured to oriel ale as *a* single 32-bi ALU. two 16-bit A LUs, or four 8-bit Al.,Us.

22. S Input. The. ALU logic module inputs are provided by tWo internal multiplexers referred to xS the S and R multiplexers. 'This tkeld selects the input to be provided by, the **S** muliiplexer; register file, **DB bus,** or MQ register. Thy source register is &lined by field 25.

23. It Input. Selects input to be provided by the R multiplexer: register file ox **DA bus.**

24. DestiuRtion Register, Address of register in register file to be used fOr the destinntion. operand.

25. Source Register. Address of register in register file **10** be used for the source operand, provided by the S multiplexer.

26- Source Register. Address of register in register file to be used l'or the source operwnd, provided by ihe R multiplexer.

Finally, field 27 is an 8-bit opeodc that specifies the atithmelic or logical function to **be perforated by** lhe ALU. Table 17.9 lists the different operations **that** can be performed,

Table 17.9 TI 8832 1.Zigi4(4tred ALL' Instruclion Field (Field 27)

Group 1		**Fundion**
ADD	Hlis71	RtSt Cn
SUBR	l-lAl2	(NOT R) 4•.S + Cn
SUBS	H;403	R (NO`I. 5) t Cn
INS('	H#C.14	**S .CD**
INCNS	H#05	(NOT 5) t Ca
1NCR	14a:tit)	R + Cn
INC NR	1-3#07	(NOT RI + Ca
'KC*	HAlg.	R XOR S
AND	t140A	R AND S
OR	HIT013	R OR S
NAND	Hit0C	R NAND S
NOR	HOD R NOK S	
ANDNR	1140E..	(NOT R) AND S

Table 17-9 *fmlismeeij*

Group 1		l'u ntliou
SRA.	H± 00	Ari l 11 n tetic right sin Fle pr.....cision hhill
SR, D	lill 1.0	Ariihrneiic right double precision shir,
SRL	HM2O	Logical righi .i rtgl.w prucision shift
S14.1...1j	H#'30	Lojlica I r i:Ost double. pi ecision shift
SLA	Hg40	Arithmetic left 5rrig3c precision Nhift
SLAF]	1-1g515	Ariihrnelie kfit double prceision .. hjri
•:Il. 0	H;461.)	Circular left single precision shift
SLCD	11471)	Ci rcular len. double preeision shift
'SRC	1.I4S11	Circular right sin& premion shift
SRCD	I-1 9l.I	Ciretilaf tight double precision shin
V1(1.1.44 A	H#AQ	Arithmetic right shift MO register
iMQSRL	H4130	Logical riLiht shift MO register
MOSLI..	FITFCr)	Logieril Il... 0. w hiff "„V..) r,.gisii...i.
MOSLC'	HTFDO	6rcula r left shift MO r6giii(er
LOADMQ	Huai	Load MO register
PASS	I Ii*•{1	PDhs AI LIR) '". (no shgt oporation)
Gr4 pup 3		1.'0111.1iiniti
.S17111'1	lii+08	Set it L
Set()	1'141 F	Set bit (I
TB]	1-Mr.i.	'1 *e51* bit 1
'1 131)	H;13'...	Test bit 0
ABS	I Ig4g	Absoluie value
SMTC	E-14g: g	Sign inagniturIcitwos complonioni
AI)I)I	Hite&	Add imme.diace
SUBI	1'N7ii	Subtract immediate
BA DD	H4Sfi	Byte add R Ia S
13SUB5	RIGS	1-lyw. suhtraes S from R
RSUBR	HTFA8	Byte subtract R from S
BINCS	HuI3S	B:I, te increment S
BIN CNS	I.1.1-1-0.	.Ryl ..-. increment negative S
fI XOR.	1-1 :k M	Byti:.%. XORR and S
II.AND	HIFE8	Byto AND R and 'S
B OR	1-iltrzi	Byte OR R ...ii i 1.3 S

'rabic 17.9 *(uoruinliCa!*

Group 4		Function
CRC	H101	Cyclic; redundancy character aiec um.
SEL	HTF 10	Sec S or R
SNORM	H2O	Sin izle length MD-MN I I i5i2
DNORM	1-1.'*3':1	Doutile length normalize
DIVRF	Hst4'.1	Divide remainder fix
SD iv QF	H+'.50	Signed d i vidz. quotient I'i x
SMUL1	Hil4'10	Signed multiply iterate
SN11.ii,rf	Hfi7'1	Signed inulcipli, terminate
SDIVIN	1-Ii+80	SiK;iitect divide in i tialiv.c.
SDIVIS	11490	Signed d ivu lc sidd
SDIVI	F-1 '.ast)	Signal divt& ieruLo
UDIV IS	HN1:30	Unsigned. cli.i.ide start
UDIVI	.l-3 CAl	Unstued divide itemLe
LI:v1U LI	1-INPO	Unsigned inulttply iterate
SDT V l'i	H ŧE()	Signed divide Lernlinate
UDIVIT	⁻14F0	(Jusipnc.d divide Iciniirtmc
Group Jr		Function
LOA DPP	Wi.)F.'	Loact.divide;BCD flip-flopt:.
CI,R	1-E.fl F	Cleat
DUMPFF	1-14i5F	Output divided:10E) flip-[lops
BCDBIN	f-1.1f7F	BCD to hinars..
FA:11-IC	lifF81⁷	Excess 3 lyyLe cOi reetion
E.X.312	FIAL..'1⁷	Exces.s 3 word con·cell CTT
SD[VD	I I itAT	Signed divide overflow tem.
131N EX 3	HRDF	binary to execissji
NOP3?..	Id ifFF	No.operation

As an example of the coding used to l'y tichis 17 through 27, consider du. acid the contents *of* register 1 to register 2 and plee the result in register .3. The svrribolie instruel ion is

C 01.771 F _ 7 WEr=i1.1, ;3.7⁴31.7.YFY:1X [2 4] **1 3 , R2, rt1,** S—ADD

The assembler will translate this into the appropriate bit pattern. The individual components of the instruction can he deKribed as follows:

- CO T[L is the basic NOP instruction.
- Field [1.7] is changed to WELI 1. (write enable, low and high), so i hat a 32-bit register i Wri LIAM into
- Field [18] is changed to SELRI'NM.X to select the feedback from the A1,U Y NILO< output.
- Field [24] is changed to designate register R3 *for* the destination register_
- * Pield [25] is changed to designate register R2 for one of the source registers.
- Field [26] is changed lo designate register R1 for one of Zhu source registers.
- * Field 12.7] is changed to specify an ALU operation of ADD. The ALL! shifter instruction is PASS; [herefore, the ALL output is not shifted by the shifter.

Several points can be made about the s!,'rnbolic notation. It is not necessary to specify the field number for consecutive fields, That is,

C.:D2q711. [17] , 1'1ELH, L18.1, SEL.RFYYR

can be written as

CONTI"! ' 171 ,

because SELRFYMX is in field 18.

Al.0 instructions from Group 1 of Table 17.9 must always be used in conjunction will' Group 2. ALU instructions from Groups 3-5 must not be used with *Group* 2.

17.5 APPLICATIONS OF MICROPROGRAMMIN

Since the introduction of microprogramming, and especially since the hoe 1.900s, the applications of microprogramming have become increasingiy varied and widespread. As early as 1971, most if not all of the contemporary uses of microprogramming were in evidence [FL N71], Subsequent surveys discuss essentially the same set of appiicatioas (e.g., [R A US80]). The set of CUM:DI applications for microprogramming includes

- Realization of computers
- **LintilMlion**
- Operaiing system support
- Realization of special-purpose devices
- I language support
- fylicrodiagnostics
- User tailoring

This chapter has been devoi 1o•a discussion of *realization ofcompurery.* The microprogrammed approach offers a systematic technique for control unit implementation. A rclaled technique is *ernulalion* IMALL751. Emulation refers to the use of a microprogram on uric machine to execute programs originally written for another. The most common use of emulation is to aid users in migrating from one eompuier to another. This is frequently done by a vendor to make it easier for exist-

MR customers to trade in older machines for newer ones, thus making a switch to another vendor unattractive. Users are often surprised to find out how long this transition tool stays around. One observer [MALL81 noted that it was still possible in 1983 to find an IBM Systern/37{1 emulating an lB.M 14W that was physically replaced over a decade and a half earlier.

Another fruitful use of microprogramming is in the area of *operating system support.* Microprograms can be used to implement primitives that replace important portions of operating system software. This technique can simplify the task of operating systern implementation and improve operating system performance.

Microprogramming is useful as a vehicle for implementing *spe•ial-purpose devices* that may be incorporated into a host computer. A good example of this is a data communications board. The board will eorn;iin its own microprocessor. Because it is being used for a special purpose, it makes sense to implement some of its functions in firmware rather than software to enhance performance.

High-level language support is another fruitful area for the application of microprogramming techniques. Various functions and data types can be implemented directly in firmware. The result is that it is easier to compile the program into an efficient machine language form. In effect, the machine language is tailored to meet the needs of the high-level language (e.g.. FORTRAN, COBOL, Ada).

Microprogramming can he *used* to support the monitoring, detection, isolation. and repair of system errors. These features are known as *microdiagnmtio* and can significantly enhance the system maintenance facility. This approach allows the system to reconfigure itself when failure is detected' for example. if a high-speed multiplier is malfunctioning, a microprogrammed multiplier can take over.

A general category of application is *reser tailoring.* A number of machines provide a *writable control store,* that is. a control memory implemented in RAM rather than ROM, and allow the user to write microprograms. Generally, a very vertical, easy-to-use microinstruction set is provided. This allows the user to tailor the machine to the desired application.

17.6 RECOMMENDED READING

Tiler L: >ne a number of hooks devoted to microprogramming'. Perhapc. the most compre. hensive is [LYNC93I ISEGE9I J pres12'nis I he fundamentals of microcoding and IIIL design of microcoded systems by means orf iI sie·p-by-si.cp design of a simple 16-hil p uessur. ICART961 also presents the basic concept. using a sample machine. [PAR109] **[Tr9O]** provide a derailed description of the TI 880J StAlWare Development Board.

CART% (:'artcr, J. Micropracessol- *A 0am:here and Mkroprop (moping.* Upper Saddle, River. NJ: Prentice Hall, 1996,

LNAC93 Lynch, M. *Microprogrammed Stare Machine Design.* Boca Raton. FL: CRC Press, 1993,

PAR1K89 Parker. A., and liainhien,,1. *An introdaction tr, Mi4:roprivamonnpg with Exorises' DesItmed ,for the !'cirri instruments SN7-1ACIN8490 Software anelopment Boma* Dallas. TX; Texas Instruments, 1989,

SEGE91 Segee, B. and Field, .1. *Microprogtwormin,s,' and Compoter Arhiterotre. 'Nov* York: tkiIcy, 1991,

T1911 Texas Instruments inc. *SA74.4C7880 ranadv Data Mannal. SCSSOCHA%* P)99.

17.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key 'Te ri

control memory	microinstruction encoding	nnicroprogrammed control unit
control word	microinstruction execution	microprogramming language
firmware	microinstruction sequelicing	soft microprogramming
hard microprogramming	inicroinSiT act ions	unpacked microinstruction
horif,ontal microinstruction	microprogram	vertical rnicToinstrUction

Review Questions

17.1 What is the difference between a hardwired implementation and a microprogrammed implementation of a control unit?

17.2 llow is a horizontal microinstruction interpreted?

17.3 What is the purpose of a control memory'?

17.4 What is a typical sequence in the execution of a horizontal microinstruction?

17.5 What is the difference between horizontal and vertical microinstructions'?

17.6 What are the basic tasks performed by a microprogrammed control unit?

17.7 What is the difference betv..cen packed and unpacked microinstructions'?

17.S What is the difference between hard and soft microprogramming?

17.9 Whar k the difference between functional anti resource encoding?

17.10 List some common applications of microprogramming.

Problems

17.1 Describe the implementation of the multiply instruction in the hypothetical machine designed by Wilkes. Use narrative and a flowchart.

17.2 Assume a microinstruction set that includes a microinstruction with the following symbolic **form;**

? AC ,, - j MEE' CAR E.L,C.E CAR + 1

AC,, is the sign bit of the accumulator and C,,,, are the first seven hits of the microinstruction. Using this microinstruction. write a microprogram that implements a Branch Register Minus (13RM) machine insi ruction, which branches if **the** AC is negative. Assume that bits C through C of the microinstruction specify a parallel set of micro-operations. Express the program symbolically,

17.3 A simple processor has four major phases to its insirlictiLm cycle: fetch. indirect, execute. and interrupt. Two I -bit flags designate the current phase in a hardwired implementation,

a. Why arc these flags needed?

b. Why arc they not needed in a microprogrammed control unit:'

17,4 Consider the control unit of Figure 17.7. Assume that the control memory is 24 bits wide. The control portion of the microinstruction format is divided into two fields. A micro-operation field of 13 bits specifies the miero-operai ion'; trF h.L. performed, An address selection field specifics a condition, based on the flags. that will cause a microinstruction branch. There are eight nags.

g. How many bits are in the address selection field;'

h. How many Nis are in the address ficrd?'

tr. 'What is the size of the control memory?

17,$ Flow i.:an unconditional branching he done ti nder the circumstances of the previoul. problem? !Tow can branching he avoided. That is. describe. a microinstruction that.- does not specify any branch. conditional or unconditional.

L7.6 We wish to provide 8 conlrol words for each machine instructjr a riotitirw. Madtinc instruction opmdes have 5 bits. and control memory has 11)24 words. Suggest a mapping from the instruction register to the control address register.

173 An encoded microinstruction format is to be used. Show how a micro-operation field can be divided into suhfie!ds to specif..... 46 clifft.T4Jni actions.

17.8 A processor hiw3 16 rogimurs, an ALE with 1ti logic and In Hritlirlieti<: functions, and a shifter with Op5-Tations, all connected by an internal Irrtrrussor hos, Dcsigi a microinstruction roring spet:ily the various micro-opi2c11itrr4 I,ii 111..1 pri,FeeSSOr,

Parallel Organization

The final part of the book looks at the increasingly important area of parallel organization. In a parallel organization. multiple processing units cooperate to execuk tipplierition5, Whereas a supercaLar processor exploits opportunities ror parallel ex,v.cution at the instruction level, a paraLLel processing organization Looks for a grosser level of paralteLism, one that it rin Nes work to be done, in parallel, and conperatively, by multiple processors. A number ksues are raked by such organizations. or example, if multiple proce.,-sors, each with its own. cache, share access to the same mcmory, hardware or software mechanisMs must be cmploycd to ensure ihat both professors share a valid imap of main memory; this is known as the cache coherence prob-kern. ['his design **issue,** and others, is exp]ured in Part Five,

ROAD MAP FOR PART FIVE

Chapter 1S Parallel Processing

Chapter IBS provides ttn overview of parallel processing eomitLrMion7.;. Thcri the chapter looks at three approaches to ornuizing mull ipto processors: s.y rn-metric multiprocessors (SNIP), clusters. and nonuniform memory access (NUM) machines. SNI Ps and dust ers are the two most common ways of organizing multiple processors to improve performance and avirilahilit!, ... N t.IN1 A systems are a more recent concept that have yet achieved widespread commerciat success but that show considerable promise. Finally, Chapter 1.8 looks at the .speciali.bed oryonization known as a vector processor.

CHAPTER 18

PARALLEL PROCESSING

1.8.1 Multiple Processor organiutions
Types of Parallel Processor Systems
Parallel Organizations

18.2 Symmetric Multiprocessors
Organization
Multiproce:o•or Operating System Dosign Considorations
A Mainframe SNIP

183 Cache Coherence and the MESI Protocol
Solutions
Hardware .Solutions
The MESI Protocol

184 Clusters
Cluster Configurations
˚punning System Design Issues
Clustcr Computer Architecture
Clusters versus SMl

183 Nonuniform Memory Access
Motivation
rpri tion
NUMA Prm and Cons

18.6 Vector Computation
Approaches to Vector Computation
IBM 3090 Vector Facility

18.7 Recommended Reading

18.8 Key Terms, Review Questions, and Prubleins
Kcv Terms
Review Questions
Problems

KEY POINTS

- A traditional way to increase system performance is to use multiple, processors that can execute in parallel to support a given workload. The two most common multiple-processor organizations are symmetric multiprocessors (S MPs) a Tl d clusters. More re02.11tly, nonuniform memory icccss {NU.rsil A) systems have been introduced commercially,
- An SNIP consists of multiple similar processors within the same computer. interconnected by a bus or some sort of switching arrangement. The most critical problem to address in an SNIP is that of cache coherence. Each processor has its own cache and so it is possible for a riven line of data to be present in more than one cache. It such a line is altered in one cache, then both main memory and the a liber cache have an invalid version of that fine. Cache collet. ence protocols are designed to cope with this problem.
- A cluster is a group of interconnected, whole computers working togethe, as a unified computing resource that can create the illusion of being one. machine. The term *whole computer means a* system that can run on its own, apart from the duster.
- A NUMA system is a shared-memory multiprocessor in which the le.cess time from a *given* processor to a word in memory varies with the. location of the memory word.
- A special-purpose type of parallel organization is the vector facility, which is tailored to the processing of vectors or arra!,,'s of data.

Traditionally, the computer has been viewed as a sequential machine. Most computer programming languages require the programmer to specify algorithms as sequences of instructions. Processors execute programs by executing machine instructions in a sequence and one at a time. Each instruction is executed in a sequence of operations (fetch instruction, fetch operands, perform operation, store results).

This view of the Computer has never been entirely true. At the micro-operation level, multiple control signals are generated at the same time. instruction pipelining, at least to the extent of overlapping fetch and execute operations, has been around for a long time. Both of these are examples of performing functions in parallel. This approach is taken further with superscalar organization, which exploits instruction-level parallelism. With a supersca]ar machine, there are multiple execution units.within a single processor, and these may execute multiple instructions from the same program in parallel.

As computer technology has evolved, and as the cost of computer hardware has dropped, computer designers have sought more and more opportunities for parallelism, usually to enhance performance and. in some cases, to increase availability. After an overview, this chapter looks at three of the most prominent approaches to parallel organization. First, we examine symmetric multiprocessors (SMPs), one

of the earliest and 5ii[I the most common example of parallel organization, hi an SMP organization, mufti* processors share a common memory. This organization raises the issue of cache coherence, L4t whiien a separate section is devoted. Then we describe clusters, which consist of multiple independent computers organized in a cooperative fashion. Clusters have become increasingly common to support vk.ork-loads that are beyond the capacity of a single SNIP. The third approach to the use of multiple processors that we examine is that of nonuniform memory access (VI, MA) machines. The NL:MA approach is relatively new and not !, ˙ct proven in the marketplace, but is often considered as an alternative to the SMP or cluster approach. Finally, this chapter It al hardware organizational approaches to vec-tor computation. These approaches optimize the AM; for processing vectors or arrays of floating-point numbers. They are common on the class of ,ystetris known as *2awercoinpurer.v.*

18.1 MULTIPLE PROCESSOR ORGANIZATIONS

Types of Parallel Processor Systems

A 1axonorny first introduced by Flynn IFINN721 is. still the most common way of caterriAing systems with parallel processing capability. Flynn proposed the follow-ing categories of computer systems:

- **Single instruction, single data** (SISfl) stream A single processor executes a single instruction stream to operate on data stored in a single mcrr u Uniprocessors fall into this cahegory.
- **Single instruction, multiple data (SIMI)) stream** A single machine instruction controls the simultaneous execution of a number of processing elernenis on 4i lockstep basis. l=och processing element has an associated data memory, so that each instruction is executed on a different set of data by the different processors. Vector and array processors fall into thins emegory.
- **Multiple instruction. single data (MISD) stream:** A sequence. of data is trans-mitted to a set of processors, each of which executes a different instruction s.equence. This structure is not commercially implemented.
- **Multiple instruction, multiple data (M1MD) stream;** A set of processors si mul-taneously execute different instruction .sect uwicvn on different *data* sets- SNIPS, clusters, and NUMA systems fit into this category.

With the ts,l I rvl D organization, the processors are general purpose: each is able to process a]] of the instructions necessary to perform the appropriate data transformation. MIMDs can be further subdivided by the mans in which *the* processors communicate (Figure L8.1). If the processors share a common memory. then each processor accesscs programs and data shored in the shared memory. and processors communicate with each other via that memory. The most common form of such system is known as a **symmetric multiprocessor (SN41[3]),** which we examine in Section 18.2. lit an SM P. in Lill iple processors share a single memory or pool of

memory by means of a shared bus or other interconnection mechanism; a distinguishing feature is 111;11 the memory access time to any region of memory is approx. imatOy the same for each processor. A more recent development is the **nonuniform tnemor access (NUMA)** organisation, which is described in Section 1.8.5. As the name suggests, the memory access time to different regions of memory may differ for a NUMA processor.

A collection of independent uniprocessors or SMPs may **he interconnected to** form a cluster, Communication among the computers is either via fixed paths or via some network facility,

Parallel Organizations

Figure 18.2 illustrates the general organization **of** the taxonomy of Figure 18.1. Fig. are 16.2a shows the structure, of an **SISD.** There is some sort of control unit (CU) that provides an instruction stream (IS) to a processing unit (PU). The Processing unit operates on a single data stream (DS) from a memory unit (MU), With an SIAM). there is still a single control unit, now feeding a single instruction stream to multiple RN. Each PU may have its own dedicated memory (illustrated in Figure 18.2b), or there may be a shared memory..Finally, with the MIMD, there arc multiple control units, each feeding a separate instruction stream to its own PU. Tha MIMI.) may be **a** shared-memory multiprocessor (Figure or a distributed-memory multicomputer (Figure **I8.2d).**

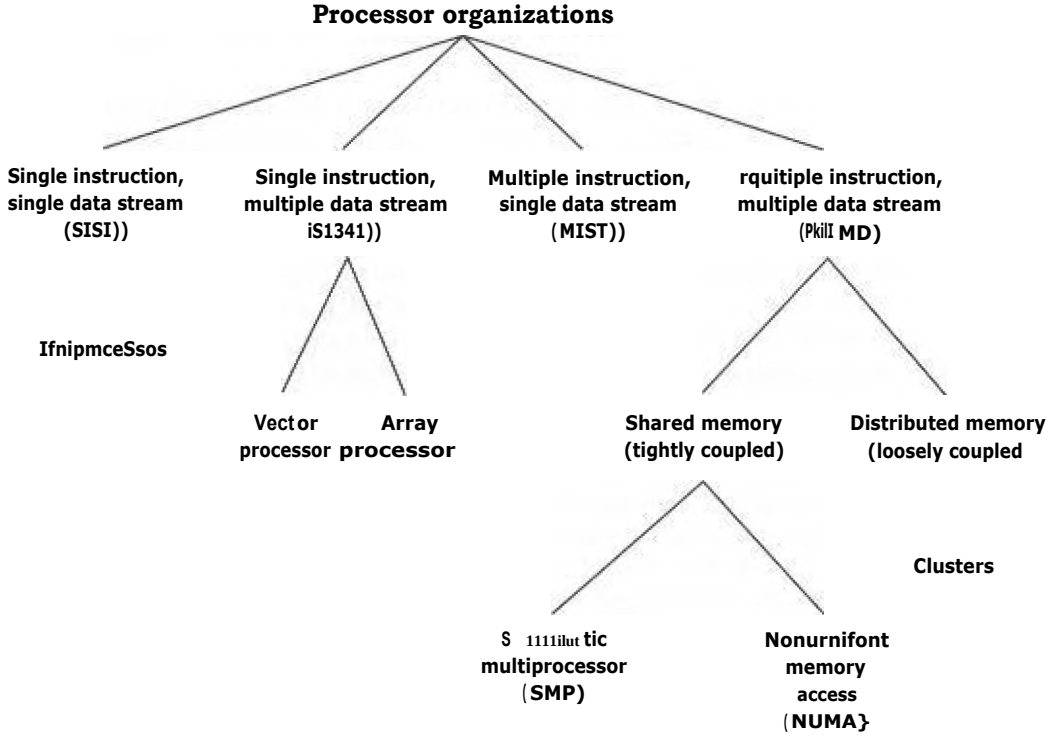
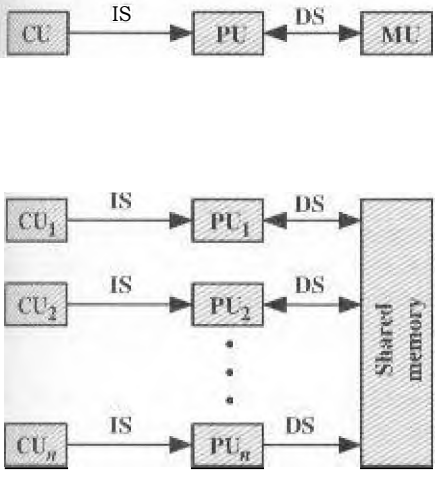


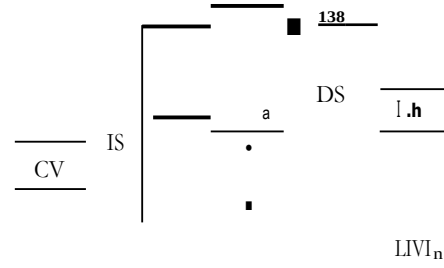
Figure 01, I A Taxonomy of Parallel Processor Architectures



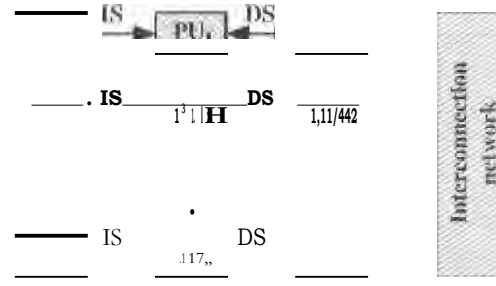
(C) M iM (with SI§ kirt d. **nicmi pry**

= Comml u nir
IS = insuucuon scream
PU = Pine Hic rip unit
= Data .t.rearri
VILI= Psit.mc.ry urns
= Local memory

SIM) - NilLOC
MIII 5h2 ihita %WC' J In
SIMI) = S
mirk ipl a dal a :Arc.' in
MINIM= Mullirde in it tt1 hui,
mulciple JAN. .J wain



(h) S1ID (vcith distributed Bunnell



Id) MIME, iwith dislributed memory

Figure 18.2 Alternative Computer Organizations

'11'11.: design issues rclatinglo SN.T-Ps, clusiers. 4i nd NUNlAs are complex, involving issues relating, to physical organization, interconnection structures, i nterprocessor communication. operating system design, and application software techniques. Our concern **hire;** iN primarily with organization, MO lough we touch bricrly on operating sys.lcnri

18.2 SYMMETRIC MULTIPROCESSORS

Until fairiy vir[iui]ly all single-user personal computers and most workstations contained a single general-purpose microprocessor. As demands for performance increase and as the cost of microprocessors continues to drop, vendors have introduced syslems with an SMP organization, The term SMP eomputcr hardware architecture and also to the operating system behavior that reflects that architecture. An SNIP can be defined as a standalone computer s!./stem with the following characteristi C5:

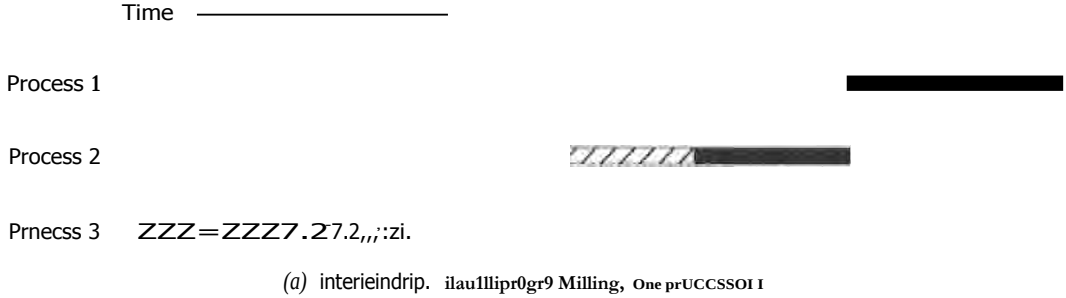
I. There are two or more similar processors of comparable capvibihtv.
/ These processors share the same main memory and I./0 facilities and are interconnected by a bus or other internal connection scheme. such that memory acccss time is approximately the same for each processor.

3. All processors share access to I/O devices, either through the same channels or through different channels that provide paths to the same device,
4. All processors can perform the same functions (hence the term *Nymmefric).*
S. The system is controlled by an integrated operating system that provides interaction between processors and their programs 41t *the* job, task, rile, and dutil element levels.

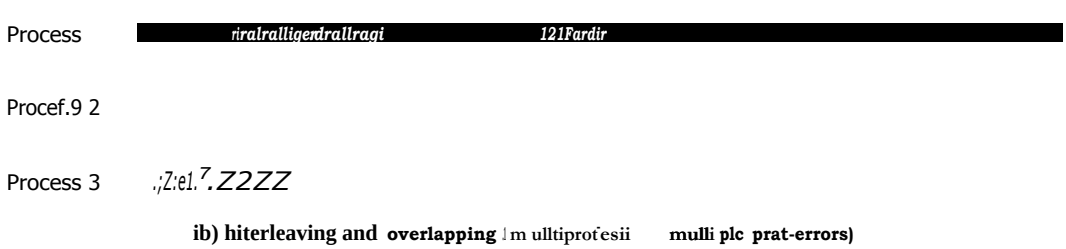
Points 1 to 4 should he self-expianatory. Point 5 illustrates one of the contrasts with a loos.ely coupled multiprocessing system. such as a duster. In the latter, the physical unit of interaction is usually a message or complete file. In an SNIP, individual data elements eau constitute the icvel of interaction, and thole can be a high degree of cooperation between vroces2,es,

The operating system of an SMP schedules processes or threads across all of the processors, An SMP organization has a number of potential advantages over a uniprocessor organization, inciuding the k I lowing;

- **Performance:** If the work to be done by a computer can be organized so that some portions of the work can be done in parallel, then a sril.Gm with multiplc processors will yield greater performance than one with a single processor of the same type (Figure 183).



(a) interieindrip. **ilau1llipr0gr9 Milling, One prUCCSSOI I**



ib) hiterleaving and overlapping ! m ulltiprot'esii **mulli plc prat-errors)**

Ricpc.kell **Running**

Figure 113.3 Multiprogramming and Multiproccssing

- Availability; In a symmetric multiprocessor, because all processors can per.-form the same functions, the failure of a single processor does not halt the machine. Instead, the syMem ciin continue to function at reduced performance.
- lnevernentstl growth: A user can **ciih4nce** the performance of a system by adding an additional processor.
- Scaling: Vendors can offi2r range of products with different price and per-formance characteristics based on the number of processors configured in the system.

it is i mportant **Lc) note** that these are potentiat, ra diu than guaranteed, benefits. The operating system must provide tools and functions to exploit the parallelism in an P system.

An al kiii.CINV feature of an SNIP is that the existence of multiple proecAsor iS transparent to ihr user. The operating system **takcx care** of scheduling of threads or processes on individual processors and of synchronization among processors.

Organization

Figure [8.4 depicts in general terms the organization of a multiprocessor sysleni, There **are lwo** or more processors. 1-]eh processor is self-contained, including a 01,311.

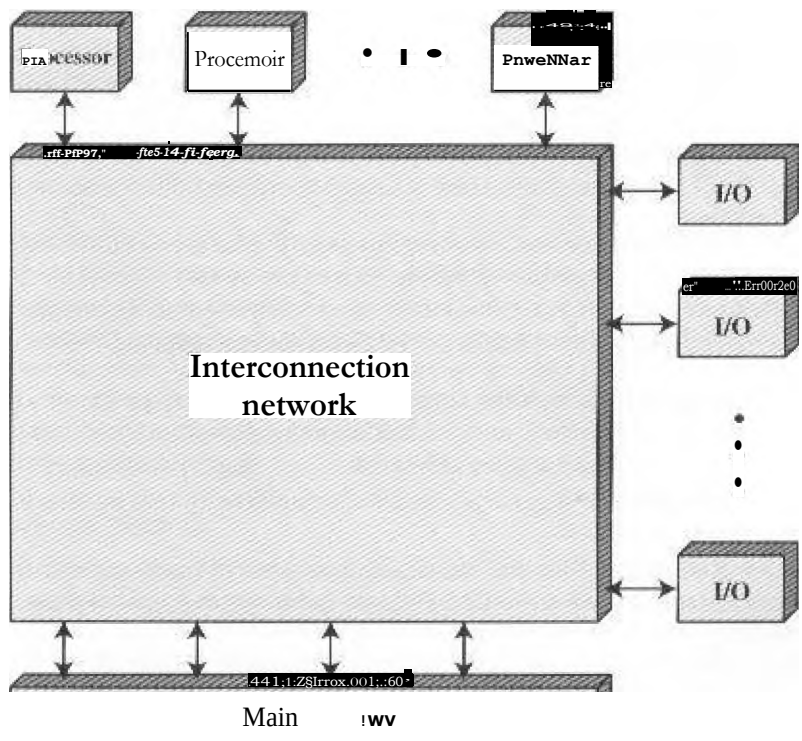


Figure 18.4 Generic Block .Diagrarn cif a righ tly COUp1C d ||/4|ultiprocuFAcir

trol unit ALU. registers, and, typically, one or more levels of cache. Each processor has access to a shared main memory and•the 1/0 devices through sonic form of imerconnection mechanism. The processors can communicate with each other through memory (messages and status information left in common data **amts.).** It may also be. possible for processors to exchange signals directly. The memory h often organized so that multiple simultaneous accesses to separate blocks of memory arc possible. I ıı **wtrmr** configurations, each processor may also have its own private main memory aud I/O channels in addition to the shared resources.

OrErinizational approaches for an SMP can be classified as follows:

- Time-shared or common bus
- Multiport memory
- Central control unit

Time_Shared Ruh

• hc lime-shared bus is the simplest mechanism [or consiructing *a* multi• processor system (Figure 'Ihe structure and im ethic:es are basically the *.sa* rne as for a single-processor system that uses a bus interconnection. The bus consists of control. address, and data lines. To facilitate DMA transfers from 110 processors, the following fcaitli-c;1 arc provided:

- **Addressing; It must** be possible to distinguish modules on the bus to dcicrinine I he source and destination or data.
- **Arbitration: Any 1/0** module can temporarily function as "master." A mechanism is provided to arbiira l e compel log requesis for bus control_ using some sort of priority scheme.
- **Time sharing: When** one module is controlling the bus, other modules arc loacd out and must, if necesmiry, suspc.nd operation until bus access is achieved.

These uniprocessor features are directly usable in an SMP organization, **In** this latter case. there are now multiple processors as well as multi * I/O processors all attempting to gain access to one or more memory modules via the bus.

The bus organization has several advantages compared with other approLiclu•:

- This is [:!:c si mplest approach to multiprocessor organiiution. The physical interface and the addressing, arbitration, and time-sharing logic of each processor remain the same as in a single-processor system.
- **FlexihiLit!..: [1** is generally (2.w.Sy' lo exp.und th4 systcm by attching more processors to the bus.
- **ReliAility: Thc** bus is essentially a passive medium, and the failure **of** ape attached device should not cause failure of the whole system.

The main drawback to the bus orga]li2ation is performance. All memory references **pass through the common bus. Thus. the bus cycle time limits the** speed of the systcm. To improve perrormance, it is desirable to equip each processor with a cache memory. This should reduce the number of bus accesses dramatically. Typ-

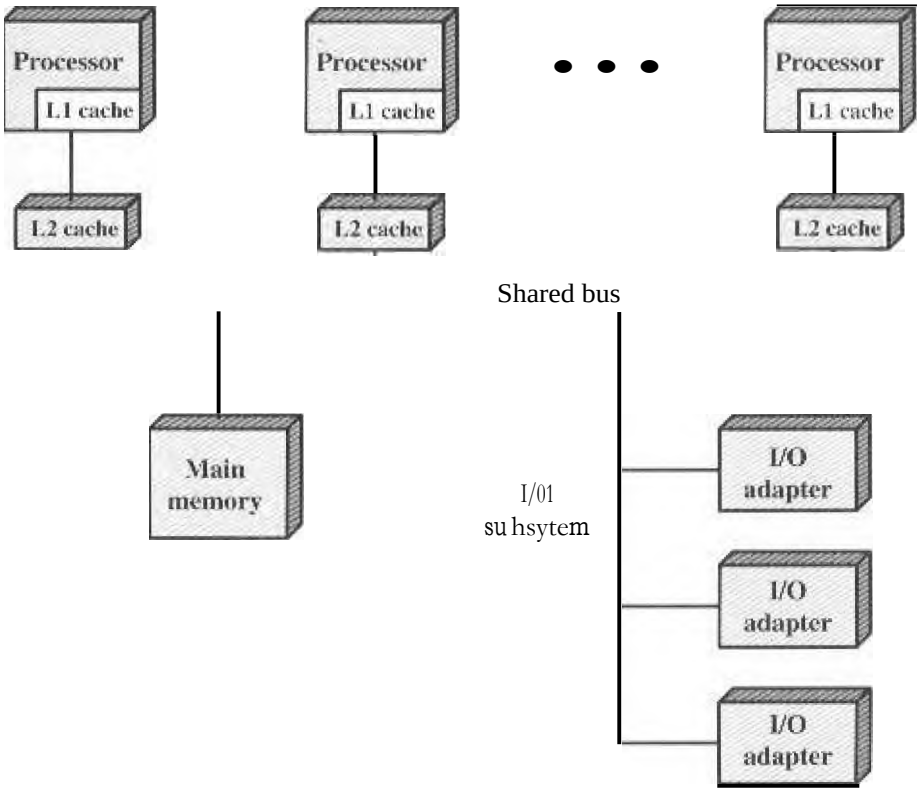


figure 18.5 S sr.ru metric Multiprocessor Orgarkinitinn

ically. workstation and PC 8/s4Ps have two levels of cache, with the I.1 *caehe* intermil (same chip as the *processor)* Find the L2 cache either internal or external,

I he ti c of caches introduces sonic new design considerations. Because each local cache eoniains an image of a portion of memory. ir a word is altered in one cache. it could conceivably inv4lidatiz, a word in another cache. 'I'D prevent phis, the other processors must be alerted thal an update hiis taken place, This problem is known as the *ruche' coherence* problem and is typie2Illy :.iLial QT:ICf5 in hardware rather than by the operating system. We address this issue in Section 18.1.

Multiport Memory

The mulliport memory Approach allows the direct, independent access of main memory modules by each processor and I/O module (Figure 18,6), Logic associated with memory is required for resolving conflicts, The method often used to resolve conflicts is to assign permanently designated priorities to each memory porI. Typieatly, the physical and electric:ill interlace at each port is identical to what would he seen in a singly.-port memory module, Thus, little or no rnodifiekition is needed for either processor or I10 modules to accommodate muEtiport memory.

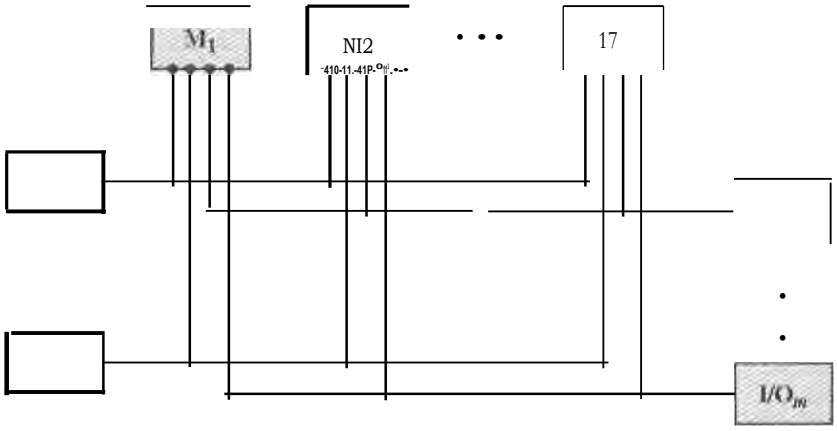


Figure 18.6 Multiporl Memory

The rnultiport memory approach is more complex than the bus approach, requiring a fair amount of logic to he added to the memory system. It should, however, provide better performance because each processor has a dedicated path to each memory module. Another advantage of multiport is that it is possible to configure portions of memory as ¯private" to one or more processors andlor I/O modules. This feature allows for increasing security against unauthorized access and for the storage of recovery routines in areas of memory not susceptible to modification by other processors.

One other point: A write-through policy should be used for cache control because there is no other convenient means to alert other processors to a MQ111-ory update,

Central Control Unit

The central control unit funnels separate data streams back and forth between independent modules: processor, memory. I/O. The controller can buffer requests and perform arbitration and timing functions. It can also pass status and control messages between processors and perform cache update alerting.

Because all the logic for coordinating the multiprocessor configuration is concentrated in the central control unit, interlaces from *1/O,* memory. and processor remain essentially undisturbed. This provides the flexibility and simplicil y of interfacing of the bus approach. The key disadvantages of this approach are that the control unit is quite complex and that it is a potential performance bottleneck.

The central control unit structure).4..as once quite common for multiple processor mainframe systems, such as large-scale members of the IBM S/1170 family, It is rarely seen today.

Multiprocessor Operating System Design Considerations

An SNIP operating system manages proce...sor and other computer resources so that the user perceives a single operating system controlling system resources. In fact,

such a configuration should appear as a single-processor multiprogramming system. In both the SNIP and uniprocessor cases, multiple jobs or processes may he active at one time, and it is the responsibility of the operating system to schedule their execution and to allocate resources. A user may construct applications that use multiple processes or multiple threads within processes without regard to whether a single processor or multiple processors will be available. Thus a multiprocessor operating system most provide all the functionality of a multiprogramming system plus additional features to accommodate multiple processors. Among the key design issues are the following:

- Simultaneous concurrent processes: OS routines need to be reentrant to allow several processors to execute the same IS code simultaneously. With multiple. processors executing the sallle or different parts of the OS, OS tables and management structures must be managed properly to avoid deadlock or invalid opera tions.
- Scheduling: Any processor may perform scheduling, so conflicts must he avoided, The scheduler must assign ready processes to available processors,
- Synchronization: With multiple .active processes having potential access to shared address spaces or shared 110 resources, care must be taken to provide effective synchronization. Synchronization is a facility that enforces mutual exclusion and event ordering.
- Memory management: Memory management on a multiprocessor must deal with all of the issues found on uniprocessor machines, as is discussed in Chapter 8. In addition. the operating system needs to exploit the availablo hardware parallelism, such as muhiported memories, to achieve the best performance. The paging mechanisms on different processors must be coordinated to enforce consistency when several processors share a page or segment and to decide on page replacement.
- Reliability and **fault tolerance: Thy** operating system should provide graceful degradation in the face of processor failure. The scheduler and other portions of the operating system must recognize the loss of a processor and restructure management tables accordingly.

A Mainframe SMP

Most PC and workstation SMPs use a bus interconnection strategy as **depicted in** Figure 18.5. It is instructive to look al an alternative approach, which is used for a recent implementation of the IBM .5 390 mainframe family [MAK971. Figure 18.7 depicts the overall organization of the S1390 SMP_ This family of systems spans a range from a uniprocessor with one main memory card to a high-end system with ten processors and four memory cards. '['he configuration includes one or two additional processors that serve as I/O processors. The key components of the configuration are as follows:

- Processor unit ITU): This is a CiSC microprocessor. in which the most Ire-.quently used instructions are hardwired and the rest are executed by firmware. Each PU includes a 64-KB Li cache that is unified (combined data and instruc-

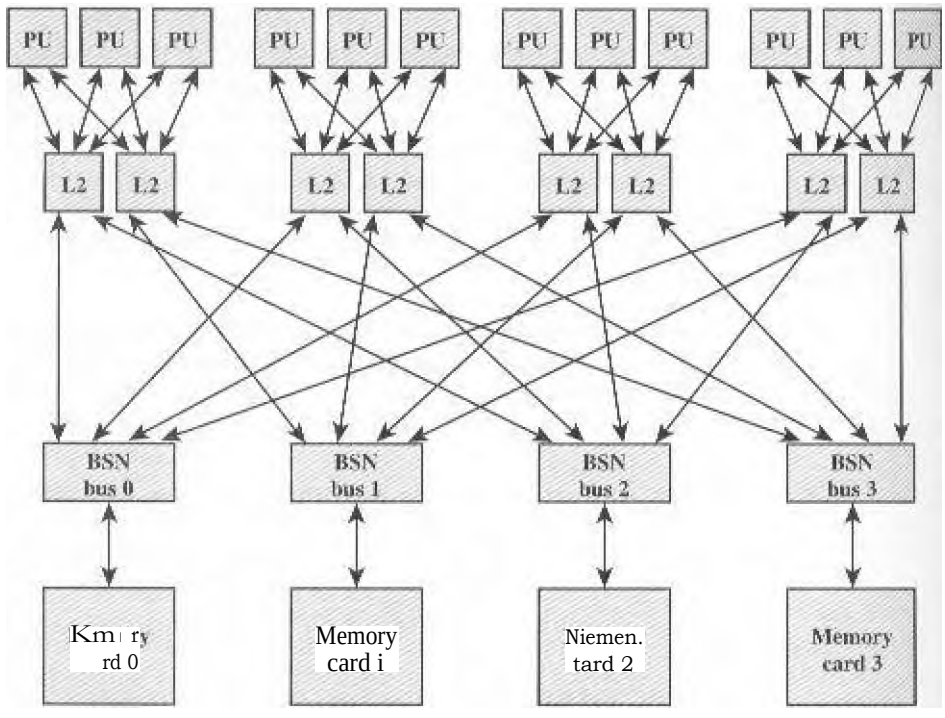


Figure 18.7 IBM S..3'40 Orpnization

lion). The 1,1 cache sir{ was chosen to fil on the PL: chip and to achieve a one-cycle access.

- **L2 cache:** Faeh 1.2 cache eon twins 384 k B, The L2 caches Lire arranged in clusters of two, with each cluster supporting thrcu Pl)s and providing to Lb42 entire main memory space.
- **Bus-switching network adapter (BSN): I he IiSNs** ini crcon l 2 cache the main memory. Each BSN also includes a level 3 (L3) cache whose size is 2 MB.
- ritlemory **card;** Each card holds 8 GB of memory, for a total of 32 GB capacity.

There are a number of interesting features in the S/390 SMP configuration, which we discuss in [urn;

- **tithed intercct nrkceti** cm
- Shared 1.2 caches
- l 3 cache

Switched Interconnection

A ingle shared bus is a common arrangement on SMPs for PCs and workstations (Figure I N5). Wish This arrangement, **the** single bus becomes a bottleneck affecting the scalability **(ability to** scale to largQr ;izcs) ,}r the design. The S/390 copes with t his problem in two ways. First, main memory is **split** into row. separate

cards, each with its own storage controller that can handle memory accesses at high speeds. 'Ric average traffic load to main memory is cut by a factor of 4, because of the four independent paths to four separate parts of memory. Second. the connection from processors (actually from L2 caches) to a single memory card is not in the form of a shared bus but rather point•to-point links, where each link connects a group of three processors via an L2 cache to a BSN.'1 ' 11e. BSN. in turn, performs the function of a switch that can route data among its five links (four 12 links. one memory card). With respect to the four 12 links, the BT ': connects the four physical links to one logical data bus. Thus. an incoming signal on any of the four L2 links is echoed back to the remaining three L2 links: this feature is required to support cache coherence.

Note that although there arc our separate memory cards. each KJ and each L2 cache has only has two physical ports in the direction of main memory. This is because each L2 cache only caches data from half the main memory. A pair of caches is required to service all of main memory, and each PU must connect to both caches in a pair.

Shared L2 Caches

In a typical two-level cache scheme for an SNIP. each processor has a dedicated LI cache and a dedicated L2 cache. In recent years, interest in the concept of a shared L2 cache has been growing. In an earlier version of its Si390 SMP, known as generation 3 (031. IBM made use of dedicated L2 caches. In its later versions (G4 and G5), a shared 12 cache is used. Two considerations dictated this change.:

L in moving from G3 to G4. IBM doubled the speed of the microprocessors. If the (i3 organization was retained. a .significanl increase in bus traffic would occur. At the same lime, it was desired to reuse as many 63 components as possible. Without a significant bus upgrade, the BSNs would become a bottleneck.

2. Analysis of typical SI:390 workloads revealed a large degree of sharing of instructions and data among processors.

These considerations led the 51391) G4 design team to consider the use of one or more L2 caches. each of which was shared by multiple processors (each processor having a dedicated on-chip LE cache). Al first glance, sharing an L2 cache might seem a had idea. Access to memory from processors should he slower because the processors must now contend for access to a single L2 cache_ I Iowever, if a sufficient amount of data is in fact shared by multiple processors, then a shared cache can increase throughput rather than retard ii. Dal a that are shared and found in the shared cache are obtained more quickly than if they must be obtained over the bus.

One approach considered by the S/390 04 design time was a single large fully shared cache. used by all processws. While this would have provided improved system performance via higher cache efficiency, this design approach would have required a complete redesign of the existing system bus organization. But performance analysis indicated that introducing cache sharing on each of the existing USN buses would generate a large percentage of the advantage of shared caches while reducing bus traffic. Thc value of shared caching was confirmed by performance measurements that showed that lite shared cache improved cache hit rates signifi-

Table ISA Typicnl Cache Hit Rate cm S;390 SNIP Con liauration

Vie En sr, Subsystem	Access Pcualty (PLI cycles) T	Cache Size	Hit Rate CYO
L [each.12	I	32 K ii	ft9
L2 cache	:;	256 KB	c
L.3 CEIEFIE	I 4	2 \i1	3
Nivniory	32	K [11	:3

cantiy over the dedicated cache scheme used in the 03 organization rMAK97]. StudicN of the value of shared caches on smaller-scale microprocessor SMIN confirm the value of this approach (Q.2., [NAYF96]).

L3 Cache

Another interesting feature or th, S/390 SNIP is the use of a third level of cache (L3). [The L3 caches are located in the BSNs, and therefore each L3 cache provide!) a buffer between L2 caches and one memory card. The L3 cache reduces latency for the data not kern in the LI and 1.2 caches of Ihe requesting rroeL.s.sor. Il provides the data much more quickly than a main memory access if the. requested cache line is already shared by other processors but was not recently used by the requesting processor,

Table 18.1 shows performance results l'or this Ihree..level cache system for a typical S/390 cornmercial workload with heavy memory and bus load IDOET9712 the sittn-age access penally is 1he Latency between the data request lo the cache hir.rarchy and the first returned 16-byte data Hoek. ' fhe 1.1 ciiefic produces a hit rare of 9%, so that the remaining 11% of memory references must be resolved at the L2, 1..3. or memory level. Of this 11 %, 5% are resolved at the L2 level, and so on Merith three levels **0f** cache, only 3% of references require a memory access. Without the third [excl. the rate of main memory access (.10nbles.

18.3 CACHE COHERENCE AND THE MESI PROTOCOL

Iii CAM emporary multiprocessor systems. it is customary to have one or Iwo levels of cache associated with each processor. 'phis organisation ix cnli it to .tichiev;2 reasonable performance. It does, however, create a problem known as the *cache crtherence* problem. The essence of the problem is this: Multiple copies of the same dala can exisl in different caches simultaneously, and if processors are allowed to

---

I B Wh.liwrauirc vErers **to** this caLhc iss a n I,2.5 cvcli . 12E12 **St!crn4** nu p,lrLiLutar advitn tHge of this term, ns2 in tact this cache constitute3 a third level of .crichirtg.

' rite data are for a 03 system, which use.; &cheated **L2** caches, F I.Dwevor, the reaffis are sticsesti%e of t he pci·l'orm.Hrtce. lo he expected with shared L2 cache. as round on (34 and (35 Si390s.

update their own copies freely, an inconsistent view of memory can result. in Chapter 4 we defined two common write policies:

- Write back: Write operations are usually made only to the cache. Main memory is only updated when the corresponding cache line is flushed from the cache.
- Write through: All write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.

It is clear Ihat a write-hack policy can result in inconsistency. If two caches contain the same line, and the line is updated in one cache. the other cache will unknowingly have an invalid value. Subsequent reads to that invalid line produce invalid results. Even with the write-through policy. inconsistency can occur unless other caches monitor the memory traffic or receive some direct notification of the update.

In this section, we will briefly survey various approaches to the cache coherence problem and then focus on the approach that is most widely used: the MESI (modifiedlexclusivcisharedlinvalid) protocol. A version of this protocol is used On both the Pentium 4 and PowurPC implementations.

For any cache coherence protocol, the objective is to let recently used local variables get into the appropriate cache and stay there through numerous reads and write, while using the protocol to maintain consistency of shared variables that Might be in multiple caches at the same lime_ Cache coherence approaches have generally been divided into software and hardware approaches. Some implementations adopt a strategy that involves both software and hardware elements. Nevertheless, the classification into soil ware and hardware approaches is still invructive and is commonly used in surveying cache coherence strategies.

Software Solutions

Software cache coherence schemes attempt to avoid the need for additional hardware circuitry and logic by relying on the compiler and operating system to deal with the problem. Software appr4 12l C.1Cti 4ite attractive because the overhead of detecting potential problems is. transferred From run time to compile time. and the design complexity is transferred from hardware to software. On the other hand, compile-time software approaches generally must make conservative decisions. leading to inefficient cache utilization.

Compiler-based coherence mechanisms perform an analysis on the code to determine which data items may become unsafe for caching, and they mark those items accordingly. The operating system or hardware then prevents noncacheable items from being cached.

The simplest approach is to prevent any shared data variables from being cached. This is too conservative, because a shared data structure may be exclusively used during some periods and may he effectively read-only during other periods. It is only during periods when at least one process may update the variable and at least one other process ma access the variable that cache coherence is an issue.

More efficient approaches analyze the code to determine safe. periods for shared variables. The compiler then inserts instructions into the generated code to enforce cache coherence during the critical periods. A number of techniques have

been developed for performing the analysis 4111d 1·01· enforcing the results; see 1L111931 and ISTEN901 for surveys.

Hardware Solutions

Hardware-lt solutions are generally referred to as cache coherence protocols.. These solutions provide dynamic recognition at run time of potential ineonsisiency conditions. Because the problem is only dealt with when it actually arises, there is more effective use of caches, Fending to improved performance over a software approach. In addition, these. approaches Li re transparent to Lhc programmer and the compiler, reducing the software development burden,

Hardware sehemes differ in a number of paniculars, including where the state information about data lines is held, how that na formalion is organized, where coherence is enforced, and the enforcement mechanisms. In general, hardware schemes can he divided into Iwo categories: directory protocols and-snoopy protocols.

Directory Ptotocok

Directory protocols collect and maintain information about where copies of lines reside, Typically, there is a centralized controller that is pan of the main memory controller, and a directory chat k sitYri2d in main memory. The directory contains global state information about the contents of the various local caches. When art individual cache controller makes a request. the centralized controller checks and issues necessary commands for data transfer between memory and caches or between caches themselves. It is also responsible for keeping t **he S.1** ate information up to date; therefore, every local action that can affect the global state **or** a line must be reported to the cern ral controller,

Typically, the controller maintains in ro1111;ki ion aboul which processors have B copy of which lines. Before a processor can write to **a** local copy oaf **a** line, it roust tcqust exclusive access to the line from the controller. Before granting this exclusive access, the controller sends a message to **all** processors with a cached copy of this [inc. forcing each processor to invalidate its copy- After receiving aeknowledgmen is back from each such processor, the controller grants exclusive access to requesting processor. When another processor tries to read a line that is exclusively granted to another processor. i1 will send a miss notification to the controller. The controller then issues a command to the processor holding that line that requires Llic processor to **do** a write back to main memory. The line may now be shared for reading by the original processor **and** the requesting processor.

Directory schemes suffer **from the drawbacks or a** central bol del'leek and the overhead of communication between the various cache controllers and the conital controller. However, they are effective in large-scale systems that involve multiple buses or some other complex interconnection scheme.

S noopy PrOti Fen N

Snoopy protocols distribtiEe the responsibility for maintaining cache coherence among all of the cache **controllers** in ai **ITliihiprticessor,** A cache must recognize when as **lime that** it holds **is shared with** other caches. **When an update** action is performed

on a shared cache line, it must be announced to all other caches by a broadcast mechanism. Each cache controller is able to "snoop" cm the network to observe these broadcasted notifications, and react accordingly.

Snoopy protocols arc ideally suited to a bus-based multiprocessor, because the shared bus provides a simple means for broadcasting and snooping. However, because one of the objectives of the use of local caches is to avoid bus accesses, care must be taken that the increased bus traffic required For broadcasting and snooping does not cancel out the gains from the use of local caches,

l wo basic approaches to the snoopy protocol have been explored: write invalidate and write update (or write broadcast). With a write-invalidate protocol. there can be multiple readers but only one writer at a time. Initially, a line may he shared among several caches for reading purposes. When one of the caches wants to perform a write to the line, it first issues a notice that invalidates that line in the other caches, making the line exclusive to the writing cache. Once the line is exclusive, the owning processor can make cheap local writes until some other processor requires the same line.

With a write-update protocol, there can he multiple writers as well as multiple readers. When a processor wishes to update a shared line, the word to be updated is distributed to all others, and caches containing that line can update it.

Neither of these two approaches is superior to the other under all circumstances. Performance depends on the number of local caches and the pattern of memory reads and writes. Sonic systems implement adaptive protocols that employ both write-invalidate and write-update mechanisms.

The write-invalidate approach is the most widely used in commercial] multiprocessor systems, such as the Peril ium 4 and Po•erPC. It marks the state of every cache line (using two extra bits in the cache tag) as modified, exclusive, shared, or invalid. For this reason, the write-invalidate protocol is called MESI. In **the** remainder of this section, we will look at its use among local caches across a multiprocessor, For simplicity in the presentation, we do not examine the mechanisms involved in coordinating among both level 1. and level 2 locally as well as at the sarne time coordinating across the distributed multiprocessor, This would not add **any** new principles but would greatly eomplicate the discussion,

The MESI Protocol

To provide cache consistency on an SMP, the data cache often supports a protocol known as NIES!. For MESI, the data cache includes two status bits per tag. so that each line can be in one of four states:

- **Modifieth'Ite** line in the cache.has been modified (different from main memory) and is available only in this cache.
- **Exclusive: The** line in **the** cache is the same as that in main memory,' and is not present in any other cache.
- Shared; The line in the cache is the same as that in main memory and may be present in another cache.
- Invalid: The line in the cache does not contain valid data.

Table 182 MES1 Cachi: Line States

	M Modified	E Exclusive	S Shored	I Invalid
This cnch Brie .1...alid?	Yep	YOS	)(CS	No
The memory copy it ...	out or dme	Valid	Valid	—
Copies cxim. ill othur caches?	No	I'Co	1434'I:..c	Maybe
A twill; tci. this line	D{5122, not go to hum	Docs not .C, La bin	Gi.)12s t}liui.; and upd.dLcs cdow	COL. A directly to bus

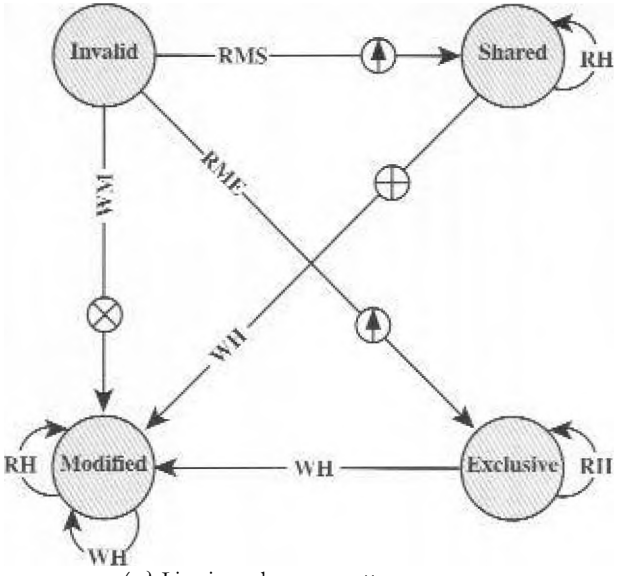
Table 18.2 summarizes the meaning of the four statc. Figure 18.X displys a state diagram for the Mfr protocol. Keep in mind that each line of the cache has its own state bits and therefore its own realization of the state diagram. Figure 1X.8a shows the transitions that occur due to actions inil kited by Ilie proces '.or attached to ;his cache. Figure 1 SMEi shows the triirmitions that occur due to events that are snooped on the common bus. This presentation of separate state diagrams for processor-initiated and bus-initiated actions helps to clarify the. logic of the ME.S1 prolocol. At any time a cache line is in a single slate, If the .next event is from the mulched procesor, then the iranshion is dictated by Figure 8.8a and if the next event is from the bus, the transition is dictated by Figure .I Bath, Let us look at these transitions in more detail.

ead Miss

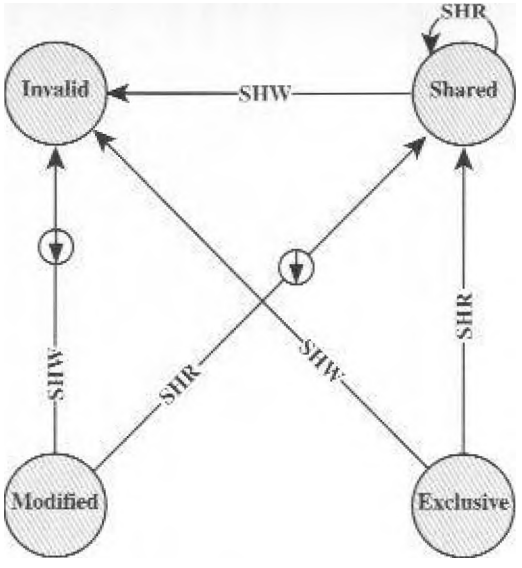
When a read miss occurs in the local cache, the processor initiates a memory read to read the]ine of main memory containing the missing address. The processor inserts a signal on the bus that a]erts all other process)] !cacheunils snoiip the. **irinactipn,** There are a number of possible oulcomes:

- If **one** other cache has a clean (unniodirics3 since read from rnernoi)9 copy of the line in the exclusive state, it returns a signal indicating that it shares this]ine. The. responding processor then transitions the state of its copy from eNclusive lo shared, and the initialling processor reads the line from main memory and transitions the Line in its cache from. invalid to shared.
- If one or **more** caches have a clean copy of the line in the shared state, each of them signals Lhal ii shares Lhc Eine. The initiating processor reads the line and transitions the line in its cache from invalid to shared.
- If one other cache has a modified copy of the line, then that cache blocks the memory read and provides the line to the requesting cache over the shared bus. The resp4 in& ciche Ilion changes its line from modified to shared.'

In scinac. i mplcinentationF., Lhc cache with the modilied line signals the niici•iiting pi ocessor to retry. Merinwhilv, the prcwesNor with Oho modiricg.io3py F.7.:izes the bus, he inocliricd line buck La main niLrnorx.. and I mnsiiions Ole line in its cuctic horn iniadifi.eii to shared. Subseclueraly, die requestiag prossor tries I atr7 and liacisdhas one or more processors have a ckon cf5py or the line in ihu sharn1 s[nLc, as described. in I he prcceding point.



(a) Line in cache at initiating processor



tb) Line in snoaping.cache

RH Read hit
RMS Read miss. shared
RMF Read miss. exclusive
WH Write hit
WNI Write miss
SIIR Snoop hit on **read**
SIIW Snoop hit **on write or read with intent to modify**

(3) Dirty line copyback

ln)3ti sJate transaction

Read **with intent to** modify

Cache line **fill**

Figure **I/1.8 ME M' 5;nte** Transition Diagam

- If no other cache has a copy of the line (clean or modified). then no signals an returned. The initiating processor reads the]ine and transitions the line in its cache from invalid to exclusive-

feud Hit

When a read hit occurs on a line currently in the local cache, the processor simply reads ihe required item. There is no stale change; The stale remains modified, shared, or exclusive.

Write Miss

When a write. miss occurs in the local cache, the processor initiates a memory read to read the tine of main memory containing the missing address, For this pup pose, the processor issues a•signal on the bus that means *reall-with-inteni-fo-inufgy* (RWITM). When the line is loaded, it is immediately marked modified. With respect to other caches, two possible scenarios precede the loading of the line *of* dala.

First, some other cache may have a modified copy of this line *(state* rrindify). In this case. the alerted processor signals the initiating processor that another processor has a modified copy of the]ine. The initiating processor surrenders the bus and waits. The other processor gains access to the bus. wait ers I he modified cach e li ne back 10 main memory, and transitions the slate of the cache Zinc iti invalid (because the initiating proce ssor is going co modify this tine). Subsequently. the initiating processor will again issue a signal to the bus of RWITM and then read the line from main memory, modify the line in the cache. and mark the line in the mod. ilk...LI state.

The second scenario is that no other cache has a modified copy of the requested line. In this case, no signal is returned, and the initiating processor precoeds to read in the tine and modify it. Meanwhile., if one or more caches have a clean copy of the line in the shared state, each cache invalidates its copy of the line, and if one cache has a clean copy of the]ine in the exclusive state, it invalidates its copy of the line.

Write Hit

When a write hit occurs on a line currently in the local cache, the effect depends on the current state of that line in the local cache:

- Shared: Before performing the update, the processor must gain exclusive ownership of the line. The processor signals its intent on the bus. Each processor that has a shared copy of the line in its cache transitions the sector from shared to invalid. The. initiating processor then performs Lhc update and transitions its copy of the line from shared to modified,
- Exclusive: The processor already has exclusive control of this line, and so it si mply performs the update and transitions its copy of the *li ne from exclusive* to modified.
- **Modified:** The processor already has exclusive control of this line and has the line marked as modified, and so it simply performs the update.

LI- **l,2 Cache** Consistency

We have so far described cache coherency protocols in tenros of the woperate activity among caches connected to the same bus or other SNIP interconnection fypically,ihese caches are L2 caches, and each processor also has an Li cache that does not connect directly ' 10 the bus and that therefore cannot engage in a snoopy protocol. Thus, some scheme is needed to maintain data integrity. across both levels of cache and across all caches in the SNIP configuration.

The strategy i to extend the MF.SI protocol (or any cache coherence protocol) to the LI caches. Thus, each line. in the l .i cache includes hits to indicate the state. In essence, the objective is the following: For any line that is present in both an L2 cache and its corresponding LI cache, the LI line state should track the state of the L2 line. A simple means of doing this is to adopt the write-through policy in the LI cache in this case the write through is to the 1.2 cache and not to the memory. The write-through policy forces any modification to an LI line out to the L2 cache and therefore makes it visible lo Other L2 caches, The use of the L1 write-through policy requires that the Ll content must he a subset of the *l 2* content, 'Ellis in turn suggests that the associativity of the L2 cache should be equal to or greater than that of the Li associa tivity. The LE write-through policy is used in the **IBM SI390 SMP.**

If the Ll cache has a write-baek the relationship between the two caches is more complex. There are several approaches to maintzlining coherence. For example, the approach used on the Pentium It is described in detail in ISHAN:W I.

18.4 CLUSTERS

One of the hottest new areas in computer systein design is elt.r.tering, C u!,toring is an alternalive. lo symmetric multiprocessing as an approach to providing high perfc».- Illartce and high availability and is particularly attractive for server applications. We. can define a cluster as a group of interconnected, whole computers working together as a unified computing resource that can create the illusion of Facing one machine. '(he term *whole computer* means a system that can run on its own, apart from the cluster; in *the li* lcraturc, each computer in a cluster is typically referred to as a *node.*

[BREW97] lists four be.nefits that can be achieved with clustering, 'T'hese can a1s0 he thought of as objectives or design requirements:

- Absolute sealsithilityi It is possible to create large clusters that far surpass the power of even the largest standalone machines. A cluster can have dozens of machines, each of which is a multiproccssor.
- **Incremental scalability:** A cluster is configured in such a way that it is possible 10 add new systems to the cluster in small increments. Thus. a user can start out with a modest system and expand it as needs grow. wi I hoot having to go through a major upgrade, in which an existing small system is replaced with a larger :iysLarri,
- **High availability:** Because each node **in a** cluster i ri s1;inda Ione comptiler, !he failure of one node does not mean loss of service, In many products, fault 101-el˙arl1N. is handled automatically in software.

• Superior price/performancet By using oommodily building blocks, it is possible to put together a cluster with equal or greater computing power than singic large, machine. Ed much lower cost.

Cluster Configurations

In the literature, dusters are claAlied in a number of different ways. Perhaps the simplest classification is based on whether the computers in a duster share access to the s.ame disks. Figure 18,9a shows a two-node chimer in which the only inteteuri• nection is by means of a high-speed link that can he used for mesmige exchange to coordinate cluster activity. 'Hie link can be a LAN that .is shared with other non-cluster computers or it can he a dedical et! interconnection facility. In the Latter case, one or more of the computers in the cluster will have x link to a / .A N or WAN so that there is .a connection between the. server cluster and remote elicni Nntc. that in the figure, each computer is depicted as being a multiprocessor, This is not nccessary **but** does enhance both performance and nv2iilallilifv.

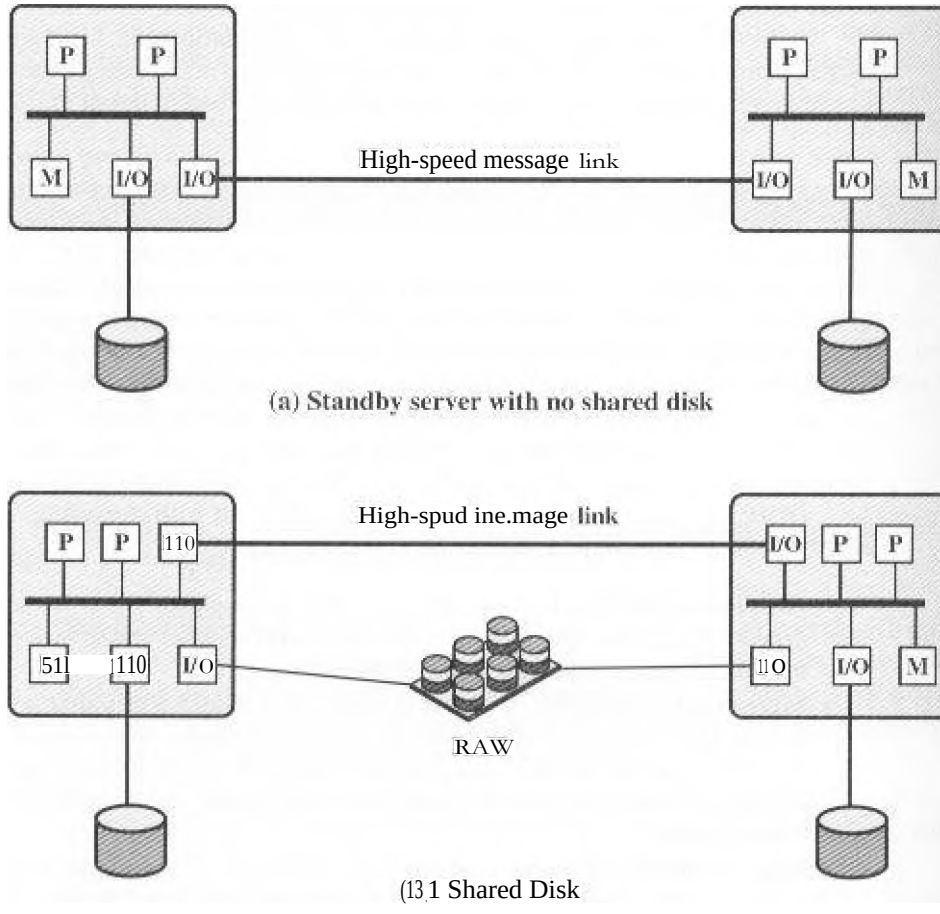


Figure 18.9 Cluster Configuration;

In the simple classification depicted in Figure L8.9. the other alternative is a shared-disk cluster. In this case, there generally is still a message link between nodes. In addition, there is a disk subsystem that is directly linked to multiple computers within the cluster. In this figure, the common disk subsystem is a RAID system, The use of RAID or some similar redundant disk technology is common in clusters so that the high availability achieved by the presence of multiple computers is not compromised by a shared disk that is a single point of failure.

A clearer picture of the range of cluster options can be gained by looking at functional alternatives. Table 18.3 provide,,, a useful classification along functional lines, which we now discuss.

A .001111/1011_ older **method, known as passive standby. is** simply to have one computer handle **all** of the processing load while the other computer remains inactive, s Lind i n g by to take over in the event uf a failure: or the primary. To coordinate the machines, the active, or primary. system periodically sends a "heartbeat" message to the standby machine, Should these messages stop arriving, the standby assumes that the primary server has failed and puts itself into operation. This approach increases availabilit!,' but does not improve performance, Further, if the only information that is exchanged between the two systems is a heartbeat message, and if the two systems do not share common disks. then the standby provides a functional backup Frut has no access to the databases managed by the primary

futile 18.3 Clustering Mk:thuds: Benefits and Limitation.;

Clustering Method	Description	13kluelit '	Limitations
Pamsive Standby	**A NeConduy wrwr** takes over in case of primary server railurc.....	busy to implement.	I igh cost becansc the secondary Server is unavailable. for other processing tasks.
Active Secondary	The secondary server is ago used for Fyn rrehsing tasks.	Reduced ecW. bct:Cd uSc secolidaty servers can he used fin processing.	Increased complcxity.
Separate Servers	Seporate servers have their own disks, Data are corninuouslv copied from primary to seoiridary server.	High availability.	High nelv...ork and server ıi) LıI Tying operation-,
Servers Connected to Disks	.Servers are cabled 10 [he same disks, but each server owns its disks. if one server fails, its disks arc taken over by the **ocher** serve r.	Reduced network and server overhead due ʟi, elimination of copying operations,	Usually. requires disk mirrorin or RAID toehnologs.. to compen• **salt for risk of disk ?AUN:.**
Servers Share Disks	Muliiplc servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires 1.ork ruanagor software. Usually used with disk mirroring ul RAID technology.

The. passive standby is generally not referred to as a cluster, The term ciuvrer reserved for multiple interconnected computers that are all actively doing proces-ing while maintaining the image of a single system to the outside world. The term active secondary is often used in referring to this configuration. Thrce classifications of clustering can be identified: separate servers, shared nothing, and shared memory,

In one approach to clustering, each computer is a **separate** server with its own disks ,rind there are no disks shared between systems (Figure 18,9a). This ar.. ranement provides high performance as well as high ‹0V4ii4i lil y. In this C41SL'. Sorno type of management or scheduling software k needed to assign incoming client requcsis tei servers so that the load is balanced and high utilization is achieved. It is desirable to have a failover capability, which means that if a computer fails while executing an application, anothcr computer in the elusler can pick up and complete the application. For this to happen, data must constantly be copied among systems so lint each system has access to the current data of the other systems. The overhead of this data exchange ensures high availability at the cost of a performance penalty.

To reduce the communications overhead, most clusters now consist of servers connected to common disks (Figure 18.9by **In** variation on this approach, called **shared nothing. the** common disks arc parlitioned into volumes, and each volume in owned by a single computer. **If** that computer fails, **the** cluster must be reconfigured so that some other computer has ownership of the volumes of the failed computer.

It is also possible to have multiple computers share the same disks at the same ti me (called the **shared disk approAch),** so that each computer has access to all of 1hc volumes on all of the disks. This approach requires the use of some type of locking facility to ensure that data can only be accessed by one computer at a tune.

Operating System Design Issues

I ul I exploitation of a cluster hardware configuration requires some enhancements to a singlc-systern operating system.

Failure Management

How failures are managed by a cluster depends on the clustering method used (I able 16,3). In general. two approaches can be taken to dealing with failures: highly available clusters and fault-tolerant clusters. A highly available cluster offers a high probability that all resources will he in service. If a failure does occur, such as a system goes down or a disk volume is lost then the queries in progress are lost Any lost query, if retried, **will** be serviced by a different computer in the cluster. However, the cluster operaiing system makes no guarantee aboul the state of partially executed transactions. This would need to be handled al the application level.

A fault-tolerant cluster ensures that all resources are always available. This is achieved by the use of redundant shared disks and mechanisms for backing out uncommitted transactions and cOrnuniting conitp[c.Eed

The function of switching applications and data resources over front a failed system 10 an alternative system in the cluster is referred to as **!allover. A related function is** the restoration of applications and daia resources i n the original systeni

one it has been fixed; this is referred to iv; fallback. Failback can be automated, but this is desirable only if the problem is truly fixed unlikely to recur. If not, automatic failback can cause subsequently failed resources to bounce back and forth between computers. resulting in performance and recovery problems.

Load Balancing

A cluster requires an effective capability for balancing the load among available computers. This includes the requirement that the cluster be incrementally scalable. When a new computer is added to the cluster, the load-balancing facility should aulomalically include this computer in scheduling applieations. Middleware mechanisms need lo recognize that services can appear on different mcmilcrs of the cluster and may migrate from one member to another.

Paralleliziag Computation

In some cases. effective LLSie of a cluster requires executing 'software from a single application in parallel. [KAPPOO] lists three general approaches to the problem:

* compiler; A parallelizing compiler dCitn-Mirke:i, al compile lime, which parts or an application can be executed in parallel. These are that split off to be assigned to different computers in the chaster. Performance depends on the nature of the problem and how well the compiler is designed.
- Parallelized upplication: In this approach, the programmer writes the application from the. outset to run On a clits.1{;.;r, and uses message passing to move.clata. as required. between duster nodes. This places li high burden on the programmer but may be t he best. approach for exploiting clusters for some applications.
- **Parametric computing:** This approach eLi n be used if essence of the applica Li on is En algorithm or program that must be executed a large number Or firnes., inch lime with a different set of starting conditions or parameters. A good example is a siniula Lion model, which will run a large. number of different scenarios and then develop statistical summaries of the results. For this approach to he effective, parametric processing tools arc needed to organi4e, run, and manage the jobs in an orderly manner,

Cluster Computer Architecture

Figure 18.10 shows a typical cluster architecture. The individual computers are connected by some high-speed LAN or Switch hardware. Each computer is capable of operating independently. In addition, a rniddleware. layer of software is installed in each computer to enable cluster operation. The cluster middieware provides a unified Nvstern image to the user, known as a single•.system image. The midclleware is also responsible for providing high availability, by means of load balancing and responding to failures in individual components. I I-1 WA N991 lixl the. following as desirable cluster raiddlcware Services and functions:

- **Single entry point;** A user logs onto the cluster rather than to an individual cotnputer.

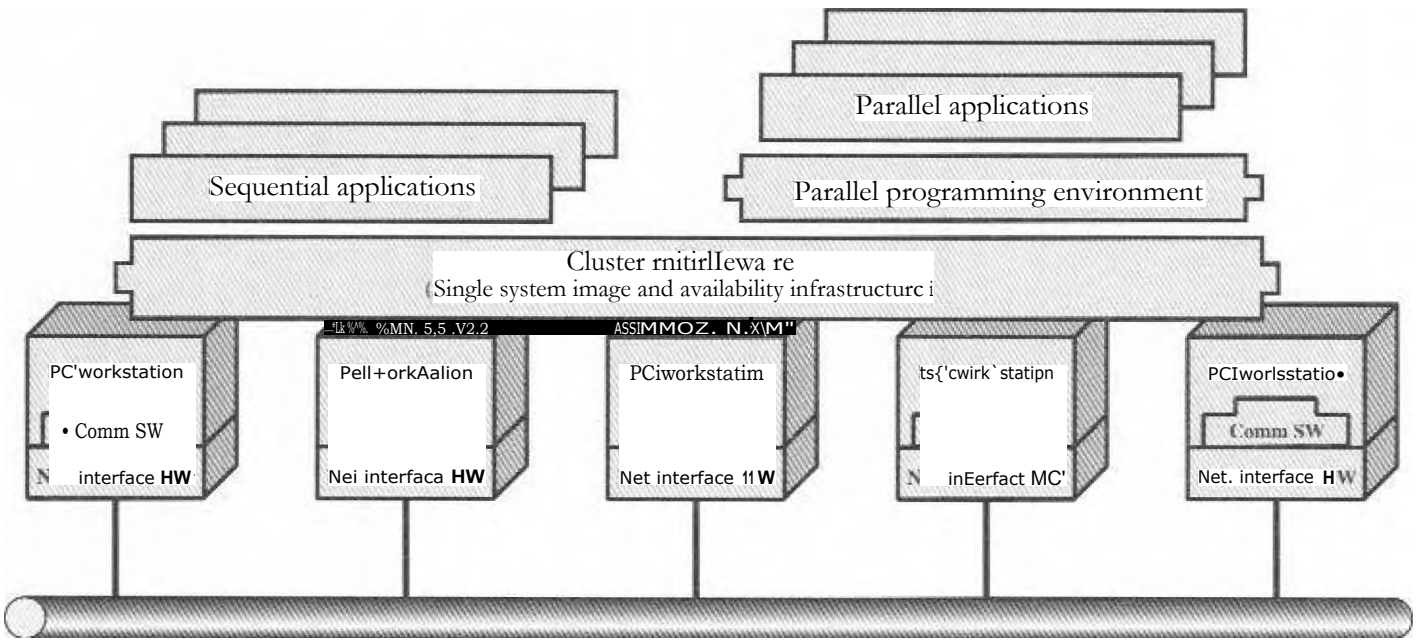


Figure 1S.10 Clusicr Computer Architect [Li t.

- Single rile hierarchy: The user sees a single hierarchy of file directories under die same root directory.
- **Single** control point: There is a default workstation used for cluster]nanagement and control.
- **Single virtual networking:** Any node can access any other point in the cluster, even though the actual cluster configuration may consist of multiple interconnected networks. I here is a single virtual network operation.
- **Single memory space;** Distributed shared memory enables programs to share variables.
- **Single job-management systemt Under** a cluster job scheduler. a user can submit a job wilhout specifying the host computer to execute the job.
- **Single user interllice:** A common graphic interface supports al/ users, rci.lardless of the workstation from which Ihey enter the cluster.
- **Single 11/0 space:** Any nodc can remotely 2ccESS. any 110 peripheral or disk device without know/edge of its physical location.
- **Single process Npaee:** A uniform process-identification scheme is used. A process on aliy node can create or communicate with any other process on a remote node.
- **Checkpointing:** This function periodically saves the process state and intermediate computing results. to allow rte] [hack recovery of a failure,
- **Process, migration:** This function enahle's load ha[ancini.

The East four items on the preceding list enhance the availability of the duster, The remaining items are concerned with providing a single system image.

urning to Figure Hi 0, a cluster will also include software tools for enabEi ng the efficient execution of programs that arc capable of parallel execution.

Clusters versus SMP

Both clusters and symmetric multiprocessors provide a configuration with multiple processors to support high-demand applications, Both soEutions are commercially available, although SMP schemes have been around far longer,

The main streno.h 0r the SMP approach is that an SMP is easier to manage and confi gure than a cluster, The SNP is much closer lo the original single.-processor model for which nearly ad applications are wrii cn. '1 he principal change required in going from a uniprocessor to an SMP is ro the scheduler function. Another benefit of the SMP is thin it usually takes up less physical space and draws Icy.; power than a comparable cluster, **A** fi nal important benefit is that the SMP products are **well** established and stabEe.

Over the long run. however, the advantages of the cluster approach likely to result in clusters dominating lhe high-performance server market. Clusters are far superior to SMPs in terms of incremental 4ind ;i hsoluie swihbility. Clusters are also superior in terms of availability, because. all components of the system can readily he made highly redundant,

in ierms of commercial products, the two common approaches to providing a multiple-processor system to support applications are SMPs and clusters, For some years. another approach. known as nonttniCorm memory access (NUMA), has been the sahjeci research and commercial NEJMA products are now available.

Before prr ceedina. we should define some terms often found in I he NUMA literature.

- **Uniform memory access (UMA):** Ali processors have access to all pares of main memor!,' using loads and stores. The memory access time of a processor **to all** regions of memory is the same- 'I'hc access times experienced by different prE)eessors are the same. The SNIP organization discussed in Sections lE..2 and 1.S3 is UMA.
- **Nonuniform memory access (NUMA);** All processors have access to all parts **of** main rnerro using loads and stores. The memory access time of a processor differs depending on which region of main memory is accessed. The last statement is true for all processors; however, for differeni proces!,ors. which memory regions are slower and which are faster differ,
- **Cache-coherent N LAI (CC-NU MA):** A NUMA system in which cache coherence is maintained among the caches of the various processors.

A NI:MA **s:..sicrn wilitiouL** cache coherence is more or less equivalent to a cluster. The commerciai products that have received much attention reeenlly arc CC-NUMA systems. which are quite distinct from bolh SVP:s and clusters. hue unfortunately not always, such systems **are** in fact referred to in the commercial literature as M A systems. This section is concerned onl!,' with CC-NUMA systems.

Motivation

With an MP system, there is practical hrnil **Lo the number** of processors that can be used. An effective cache scheme. reduces the bus traffic between any one processor **and main** memory, As the number of processors increases, this bus traffic also increases, Also, the bus is used to exchange ciiche coherence signals, further adding to the burden. At some point, the bus becomes it performance bottleneck. Performance degradation seems to limit the number of processors in an SNIP configuration to somewhere between l6 and 64 processors, For example, Silicon G raphics Power Challenge S Pis limited to 64 R10000 processors in a single system; beyond this number performance degrades substantially.

The processor limit in an SMP is one of the driving motivations behind die development of cluster sy:.4tEins. However, with a cluster, each node has its own private main **momory: :11** pliCiLI iortg do not see a large global memory. In effect, coherency is maintained in software rather than hardware, **This** memory granularity a [feels performance and, lo achieve maximum performance, software must be tailored 10 I 11is environmene. One approach to **achieving** large-scale multiprocessing

while retaining the flavor of SM P is NUMA. For example. the Silicon Graphics Origin NUMA system is designed to support up to 1024 MIPS RI0000 processors [WH IT971 and the Sequent NUMA-O system is designed to support up to 252 Pentium II processors. [LOVE96].

The objective with NUMA is to maintain a transparent systemwide memory while permitting multiple multiprocessor nodes_ each with its own bus or other internal interconnect system.

Organization

Figure 18.11 depicts a typical CC'-NUMA organization. There are multiple independent nodes, each of which is, in effect, an SMP organization, Thus, each node contains multiple processors, each with its own Ll and L2 caches, plus main memory, The node is the basic building block of the overall CC-NUMA organization. For example. each Silicon Graphics Origin node includes two MIPS R111000 processors; each Sequent NUMA-0 node includes four Pentium II processors. The nodes are interconnected by means of some communications facility, which could be a switching mechanism, a ring, or some other networking facility.

Each node in the CC-N LAM system includes some main memory. From the point. of view of the processors, however, there is. only a single addressable memory, with each location having a unique systemwide address, When a processor initiates a memory access, if the requested memory location is not in lhat processor's cache. then the L.2 cache initiates a fetch operation. 11' the desired line is in the local portion of the main memory, the line is fetched across the local bus. if the desired line is in a remote portion of the main memory, then an automatic request is sent oil lo fetch that line across the interconnection network, deliver it to the local bus, and then deliver it to the requesting cache on that bus, All of this activity is automatic and transparent to the processor and its cache.

In this configuration, cache coherence is a central concern. Although implementations differ as to details, in general terms we can say that each node must maintain some sort of directory that gives it an indication of the location of various portions of memory and also cache status information. To see how this scheme works, we give an example taken from (1[3]11S)8]_ Suppose that processor 3 on node 2 (P2-3) requests a memory location 798, which is in the memory of node 1, The following sequence occurs:

P2-3 issues a read request on the snoopy bus of node 2 for location 798,

2. The directory on node 2 sees the request and recognizes that the location is in nude I.
3. Node 2's directory sends a request to node 1. which is picked up by node l's *directory.*
4. Node I 's directory, acting as a surrogate of P2-3, requests the contents of 798, as if it were a processor.
5. Node 1's main memory responds by putting the requested data on the bus,
h. Node l's directory picks up the data from the bus.
7. The value is transferred back to node 2's directory,

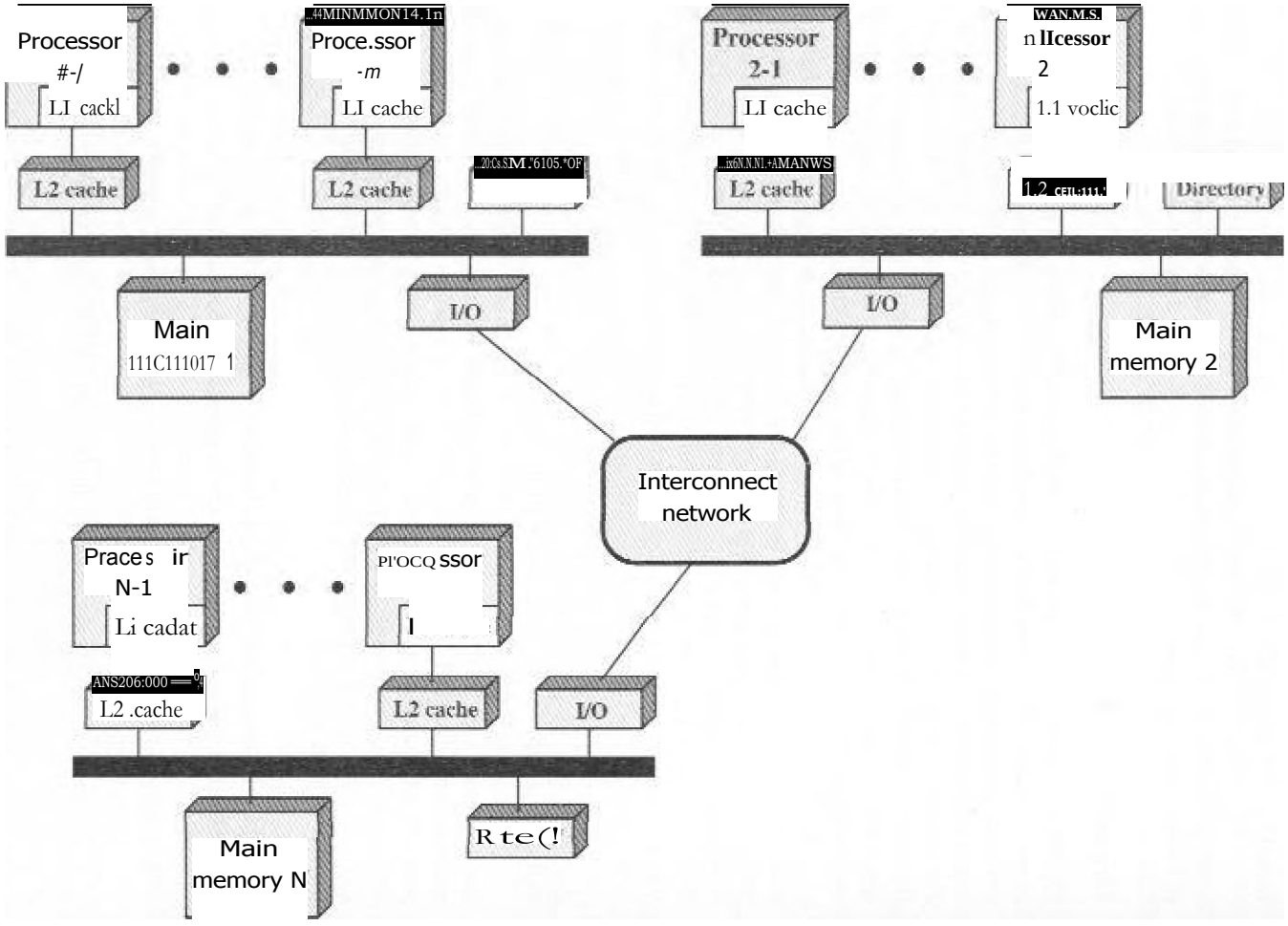


Figure 18.11 CC-N MA Organization

8. Node 2's directory places the data back on node 2's bus, acting as a surrogate for the memory that originally held it.
9. The value. is picked up and placed in P2-3's cache and delivered to 1 [3]2-3.

The preceding sequence explains how data are read from a remote memory using hardware mechanisms that make the transaction transparent to the processor. On top of these mechanisms, some form of cache coherence protocol is needed. Various systems differ on exactly how this is done. Vi[r]e make only a few general remarks here. First, as part of the preceding sequence, node l's directory keeps a record that some remote cache has a copy of the line containing location 79S. Then, there needs to he a cooperative protocol to take care of modifications. For example. if a modification is done in a cache, this fact can he broadcast to other nodes. Each node's directory that receives such a broadcast can then determine it 4i ny local cache has that line and. if so, cause it to be purged. If the actual memory location is at the node receiving the broadcast notification, then that node's directory needs to maintain an entry indicating that that line of memory is invalid and remains so until a write back occurs. If another processor (local or remote) requests the invalid line. then the local directory must force a write hack to update memory before providing the data.

NUMA **Pros and Cons**

The main advantage of a CC-NU MA system is that it can deliver effective performance at higher levels of parallelism than SMP, without requiring major sofmare changes. With multiple NUMA nodes, the bus traffic on any individual node is limited to a demand that the bus can handle. However, if many of the memory accesses arc to remote nodes. performance begins to break down. There is reason to believe that this performance breakdown can be avoided. First, the use of Ll and L2 caches is designed to minimize **all** memory accesses, including remote ones. if much of the software has good temporal locality, then remote memory accesses should not be excessive. Second. if the software has good spatial locality, and if Orillal memory is in use, then the data needed for an application will reside. on a limited number of frequently used pages that can be initially loaded into the memory local to I he running application. The Sequent designers report that such spatial locality does appear in representative applications [LOVE96]. PinaIly, the virtual memory scheme can be enhanced by including in the operating system a page migration mechanism that will move a virtual memory page to a node that is frequently using it; the Silicon Graphics designers report success with this approach [WHIT97].

There are disadvantages to the CC-N MA approach as well. Two in particular are discussed in detail in [PHS981. First, a CC-NUMA does not transparently look like an SNIP: software changes will be required to move an operating system and applications from an SIV1P to a CC-NUMA system. These include page allocation, already mentioned, process allocation, and load balancing by the operating syslem, A second concern is that of availability. This is a rather complex issue and depends on the exact implementation of the CC-NUMA system: the interested reader is referred to IPF1S98].

18.6 VECTOR COMPUTATION

Although the performance of mainframe general-purpose computers continues to improve relentlessly, there continue to be applications that are beyond the reach of the contemporary mainframe. There is a need for computers to solve mathematical problems of physical processes. such as occur in disciplines including aerodynamics, seismology, meteorology, and atomic, nuclear. and plasma physics.

Typically, these problems are charaelerized by the need for high precision and a program that repetitively performs floating-point arithmetic operations on large arrays of numbers. Most of these problems fall into the category known as *contimfous-field r/n* u In essence, a physical situation can he described by a surface or region in three dimensions *(e,g.,* the flow of air adjacent to the surface of a rocket). This surface is approximated by a grid of points. A set of differential equations defines the physical behavior of the surface at each point.. The equations are represented as an array of values and coefficients and the solution involves repeated arithmetic operations on the arrays of data.

Supercomputers were developed to handle these types of problems. These machines arc typicaliv capable of hundreds of millions of floating-point operations per second and cost in the 10 to *15* million dollar range. In contrast to mainframes, which are designed for multiprogramming and intensive the supercomputer is optimized for the type of numerical calculation just described.

The supercomputer has limited use and, because of its price tag, a limited market, Comparatively few Of these machines arc operational. mostly at research centers and some government agencies with scientific. or engineering functions. As with other areas of computer technology, there is a constant demand to increase the performance of the supercomputer. Thus. the technology and performance of the supercomputer continues to evolve,

't here is another type of system that has been designed to address the need for vector computation, referred to as the *array processor.* Although a supercomputer is optimized for vector computation, it is a general-purpose computer, capable of handling scalar processing and general data processing tasks. Array processors do nut include scalar processing; they are configured as peripheral devices by both mainframe and minicomputer users to run the vectorized portions of programs.

Approaches to Vector Computation

The key to the design of a supercomputer or array processor is to recognize that the main task is to perform aril hmetic'operations on arrays or vectors of floating-point numbers. In a general-purpose computer, this will require iteration through each element of the array, For example, consider two vectors (one-dimensional arrays) of numbers. A and If. We would like to add these and place the result in C. In the example of Figure 18,12, this requires six separate additions. Mow could we speed up this computation? The answer is to introduce some form of parallelism.

Several approaches have been taken to achieving parallelism in vector computation. We illustrate this with an example. Consider the vector multiplication C = A x *B.* where A, *13,* and *C* are *N X N* matrices. The formula for each element of *C* is

[1.5				3.5
7.1		39.7		46-8
6.9		1.000.003		1{106.903
100.5		11		111.5
0		21.1		21.1
59.7		19-7		79.4
A	+	8		C

Figure 18.12 Example of Vector Addition

$$c_{i,j} = \sum_{k=1} a_{i,k} \times b_{k,j}$$

where .4, ./.?, and C have elements and C_{ii}, respe.ctively. Figure 18.13a shows a FORTRAN program for 1.his eompiii ;ition Ihai can he run on an ordinary scour processor.

One approach to improving performance can be referred to as *vector process* - *Lux,* assumes Thal i1 is posM.ble to operate on a one-dimeasionak vector of data. Figure 18.1311 is ш FOR 1•1-Z AN program with a ricw Corm cif inkdruction Lh1l aliow;"

```
      DO 100  — I, N
      DO 100J = 1,N
      CO,     = 01.0
      DO  100 K = I,  N
      C7 0,.1) —              Atl, K}
100 CONTINUE
```

fa) Scalar procLising

```
      1)0 MO I = 1, N
      Ca,,.1)= 0.00 = 1,N)
      DO Et10     = 1,
      CU. 31 =       3) + Ad, K) + 13(K, 3) 13 =  I, 1O
IOU CON'T [NUE
```

(b) Yottor piocessing

```
      DO 50,1 —  1, N —
      FORK 100
5t) CONTINUE
          N
I of/ DO 200 I = I N
          1 ), = 0.0
      DO NO K = I, N
          .1) = .C(1 . 1) I AO . IQ • BA, .1)
2010 CONTINUE
```

(c)Parki1.14,71

Figure 18,13 Matrix N1uI plicatioir (C = A [3)

vector computation to 17FC **pcc**ifi**c** l. The notation = 1, M indicale.s ihal. operations **on** a]] indices .1 in the given interval are to he carried out as a single operation. How this can be achieved is addressed shortly.

]'he program in Figure J.& f 3h indicates that a]] the elements of the 11). row are to he computed in parallel. Mach clement in the row is a summation, and the summations (across K) are done serialiy rather than in parallel. Even so, only A2 vec-1or mullipiications are .required for this algorithm as compared with ' 3 scalar multiplications for the scalar shgurirhtn.

Another approach, *pundit.' processirex,* is illustrated in Figure 18.13c. This approach assumes that we have N independent processors that can function in parallel. '1'**0** utilin processors effcel ivety, we must somehow parcel ;Jul. the computation to the wirions processors. '1'wo prinliti e arc LiScd. 'che primitive FORK n causes an independent process to be started at location .}2. In the meantime, the orivinal process continues execution al the instruction immediately following the FORK. Every exi2eution of a FORK ipawns 4t new process · 'I he JOIN instruction is essentially the, inverse of the. FORK. The statement JOIN N causes N independeni procesK:s to he merged into one that continues execution at the instruction ['allowing the JOIN. 'I'he operating system must coordinate this merger, and so the execution does not continue until ati N processes have reached the JOIN instruction.

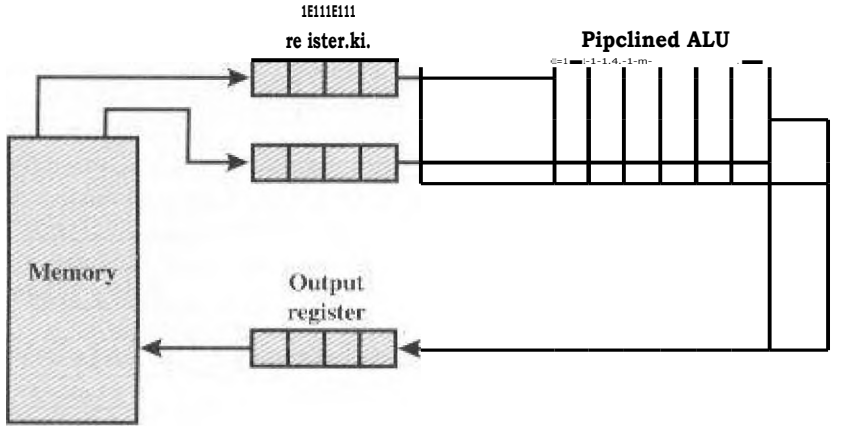
The program in Figure 18.13c is written to mimic the behavior of the vector processing program. In the parallel processing program. each column of *C* com• puted by a separate process. Thus, the elements in a given row of *C* are computed in parallel.

The preceding discussion describes approaches to vector compul alion is logi• Cal or irehiteetur'aI tcrms. Let us [urn now lo a consideration Of types of processor organization that can be used to implement these approaches. A wide variety of organizalions have been and are being pursued. Three main categories stand caul:

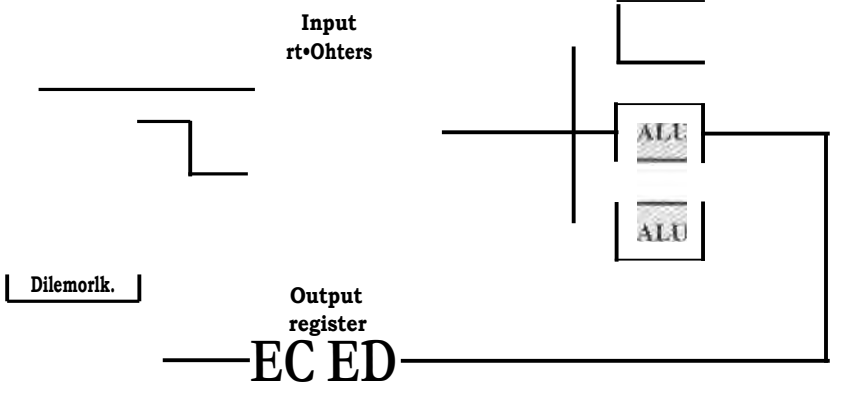
- **Pire[inCd ALU**
- Parallel ALUs
- Parallel processors

Figure 1 8.14 rat es the first two or inese approaches. We have already discussed pipelining in Chapter 12. Here the concept is extended to the operation of the ALA'. Because floating-point operations are rather complex, **there is** oppotwnity for decomposing a limning-point opera! n sUiges so [hat dilYercnL slaps can operate on differcin sets of data concurrently. 'Ellis is illustrated in Figure 1& [5a. Fioating-point addition is broken up into four stages (see Figure 9.22): corns hi ft, add, and normalize, A vector of numbers is presented sequentialiy to the first stage. .^{2}.k!, lIie processing proceeds, four different sets of numbers will he operated on concur' entl!,.y in the pipeiinc.

It should be clear that this organization is suitable. for vector processing. To see this. consider the insirnoion pipelining described in Chapter 31 The processor goes throualt a repetitive cycle of fetching and processing instructions. In the. absence **of** branches, the processor is continuous)}' fetching instructions from sequential locations. Consec.pcniiy. Itie. pipeline is kepi full and a savings in time i.



Plpelltnd ALI;

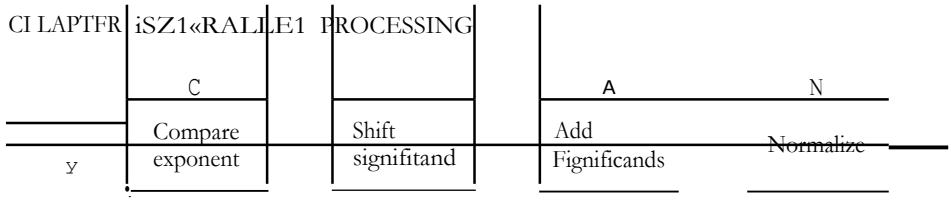


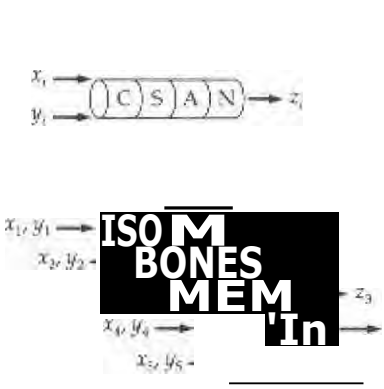
(b) Parallel Allis

Figure 18.141 Approa4:14Cs io Vcctor **Computation**

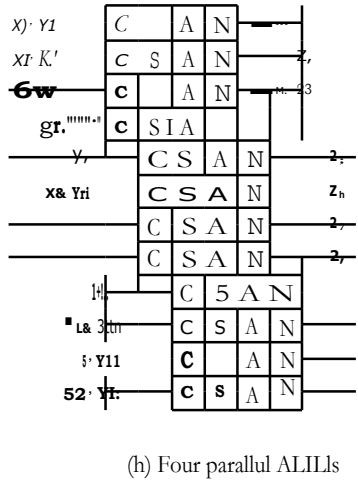
achieved. Similarly, a pipelined ALLJ will save time only **if it** is fed a stream of data from sequential locations. A single, isolated floating-point operation is not weeded up by tit pipeline, The speedup k achieved when a vector of operands is **prewn W(1** to the A1,I.J. The control unit cycles the dni a through the ALU until the entire vector is processed.

The pipeline operation nin he further enhanced if the vector elerncn Es arc available in regisiQrs rather than from main memm'y.'fhiS is in fact suggested by **Figure** 18.14a. The elcrnents of each vector operand arc londcd as a block info ci vector





(a) Pipelined ALU



(h) Four parallul ALILls

Figiirc 18.15 Piplined Processing

register, which is simply a larE.e bank 4D identical registers. The result is also placed in a vector register. Thus, rnost operations involve only the use of registers. and on1 load and store operations arid the beginning and cnd of a vectoroperation require ateCess Ito **ramory.**

The mechanism illustrated in Figure [8,15 and be referred to as *pipuking within an opertaion,* That is, we have a single arithmetic operation (e.g,. *C* — A+ B) lhat is to be applied to vector operands, and vipelining flows mulliple **VeeiOr Cle-ments** to be processed in paratlel. This mechanism can be au. mented with *22e-ing rossoprcaions.* In this latter ease. 1herc is a sequenceof arithmetic vector

operations, and instruction pipelining is used to speed up processing. One approach to this, referred to as chaining. is found on the Cray supercomputers. The basic rule for chaining is this: A vector operation may start as soon as the first clement of the operand vector(s) is available and the functional unit (e.g., add, subtract, multiply, divide) is free. Essentially, chaining C41 ust: results issuine from one functional unit to he fed immediately into another functional unit and so on, If vector registers are used, intermediate results do not have to be stored into memory and can be used even before the vector operation that created them runs to completion.

For example, when computing *C* x A) - *B.* where A. *B,* and Care vectors and *s* is a scalar, the Cray may execute three instructions at once. Elements fetched for a load immediately enter a pipelined multiplier, the products are .sent a pipelined adder, and the sums are placed in a vector register as soon as the adder completes them:

1. Vector load A Vector Register (VR1)
2. Vector load B VR2
3. Vector multiply s VR1 VR3
4. Vector add VR3 + VR2 V1 4.
5. Vector store VR4 ---> C

Instructions 2 and 3 can be chained (pipelined) because they involve differeriL memory locations and registers. Instruction 4 needs the results of instructions 2 and 3, but it can he chained with them as well. As soon as the first elements of vector registers 2 and 3 are available. the operation in instruction 4 can begin.

Another way to achieve vector processing is by the use of multiple ALIA in a single processor, under the control of a single control unit_ In this case. the control unit routes data io ALLIs so that they can function in parallel. It is also possible to use pipelining on each of the parallel ALUs. This is illustrated in Figure 18. I:rib. The example shows a case in which four ALUs operate in parallel.

As with pipelined organization, a parallel ALU organization is suitable for vector processing, The control unit routes vector elements to A I.Us in a round-robin fashion until all elements are processed. This type of organization is more complex than a single-ALU CP1.

Finally, vector processing can be achieved by using multiple parallel processors. In this case, it is necessary to break the task up into multiple processes to be executed in parallel. This organization is effective only if the software and hardware for effective coordination of parallel processors is available.

We can expand our taxonomy of Section 18.1 to reflect these new structures, as shown in Figure l8.16. Computer organizations can be distinguished by the presence of one or more control units. Multiple control units imply multiple processors. Following our previous discussion. if the multiple processors can function cooperatively on a given task, they are termed *parallel proce*mrs.

The reader should he aware of some unfortunate terminology likely to be encountered in the literature. The term *veciur proces.vor* is often equated with a pipelined ALL organization. although a parallel ALL; organization is also designed

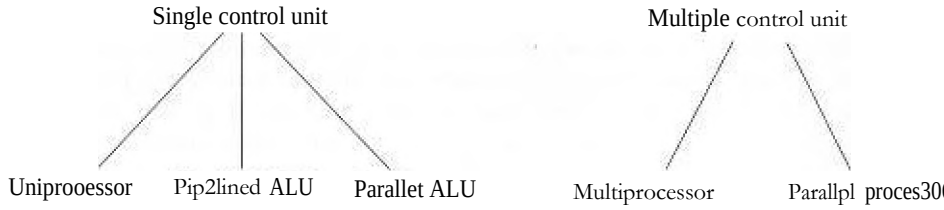


Figure 18.16 A TaKonorni„ of Computer Organizations

for vector processing, and, as we have diseusscd,.a parallel proCessor organization rmiv ailso be designed for vector processing. *Array proceNsing ins* sometimes ascii to refer to a parallel although, a4ain, any of the three organizations is optimized for the processing of i.rrays. To make **matters** worse, *array processor* usually refers **tO an** auxiliary processor attached to a gencral-purpose processor and and to perform veclor computation, An array processor may use. either the pipelined or parallel ALU approach,

At present, the pipelined ALU organisation dominate* the marketplace. Pipt:lined systems are less complex than the other two approaches. Their control unit and operating system design are well developed to achieve efficient resource allocation and Itigh performance, 'l'he remainder of this section is devoted to a **more** detailed examination of this approach, using a specific example,

IBM 3090 Vector Facility

A good example of a pipolined ALU organization for vector processing is the **Kc**tor faeility developed for the IBM 370 architecture and implemented on the Netend 3090 series [PADE88, .I·UCK87].`Ubis facility is an optional add-on to the basic system but is highly integrated with i1. II resembles vector facilities found on supercomputers. such as the Cray famil!,'.

The IBM facility makes use of a number of vector registers. Each tegicter is actually a hank of :'.aar registers, To compute the vector sum C — *A* − *B,* the vectors *A* and *B* are loaded into two vector regisLers- The **data** from these registers are passed through the ALU as fast as possible, and the results are **Mired** in a third vector register, 'Mc compulation overlap. and the loading of the input data into the regislers in a block, results in a significant speeding up over an ordinary ALU operation.

Organization

The IBM vector architecture, **and** si milar pipelined vector ALUs, **provides** increased performance over loops of scalar arithmetic instruct ions in **three** ways:

- The fixed and predelermined structure of vector data permits housekeeping instructions inside the loop to he rcpInced by faster internal (hardware or microcoded) machine operations.
- Data-access and arithmetic oper;t0ons on several successive vector elements can proceed concurrently by overlapping such operation,., in pipelined dcsiUn or by performing multiple-element operations in parallel,

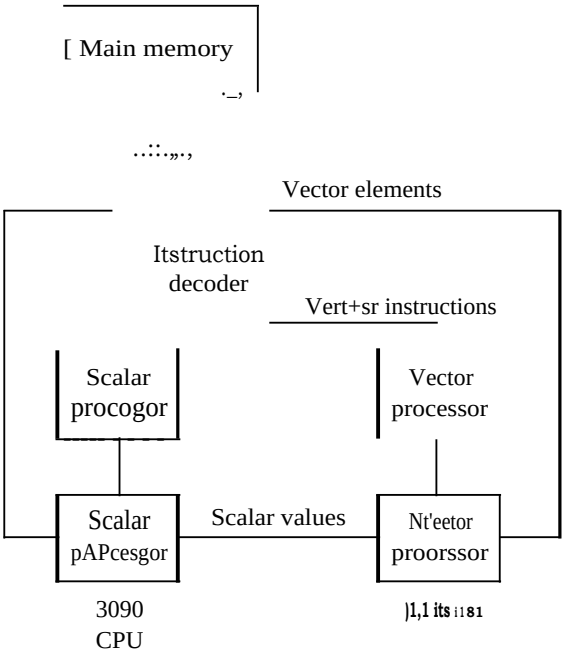


Figure 18.17 IBM 3090 with Vector Facility

■ The use of vector registers for intermediate results avoids additional storage reference,

Figure 18.17 shows the general organization of the vector facility. Although the vector facility is seen to be a physically separate add-on to the processor, its Eirchi-t lturc!' is an extension of the System/370 architecture and is compatible will, it, '1 ïic vector facility is integrated into the Systern131{1. archilccturc in the following ways!

- a Existing SystemI370 instructions are used for all scalar operations.
- Arithmetic operations on individuai vector elements produce exactly the same result ;is do corresponding System/370 scalar instructions, For example., one design decision ccaneerricd the dufiniiit101 of the rcsull in a floating-point **DIVIDE** operation. Should the result be exact, as it is for scalar floating-point division, or should an approximation be allowed that would permit higher-speed implementation but could sometimes introduce an error *in* oar or more low-order bit positions? The decision w i s made to uphold complete compatibility with the System/37C) rchitecturL. 2Lt the expense of a miner performance degradation.
- Vector instructions are interruptible. and their execution can be resumed from the point of int erruption after appropriate action has been iaken, in a manner compatible with the System/370 program-interruption scheme.
- Arithmelic cxceptions are the mme as, or cxtcnsions cif, exceptions. for the scalar **ri t h** m et ic instructions of the System/370, and si milar fix-up routines can

be used. To accommodate this, a vector iEtterruption index is employed that indicates the location in a vector register that is affected by an exception (e.g., ovcrliow, Thus. when execution of the vector instruction resumes, the proper place in a vector rcgkter is wxcsbcd•

- Vector data reside in virlua] storage, with page faults being hanclicd in a standard manner.

.1 his level of integration provides a number of beuctitii. Existing operating systems can .support the •eettpr faeitity with minter extcrNions. Hxisting application programs, language compilers, and other software can he run unchanged. Software that could lake advantage of the vector facility can be modified as desired.

ItegiNters

A key issue in the design of a vector facility is whether operands are located in registers or memory. 'He I BM Organizalion is referred lo ax *re, isler-o-register,* because the vector operands, both input and output, can be staged in vector registers. This approach is also used on the Cray supercomputer. An alternative approach, used on Control Data machines, is to obtain operands directly from memory. The main disadvantage of the use of vector register. ', b. dial the programmer or compiler must take them into account for good performance. For example, suppose that the length of the vector registers is *K* and the length of the vectors to be processed is N *K,* In this case, a vector loop must be performed, in which the operation is performed on l.K elements al a time and the loop is repealed timm The main advantage of the vector register approach is that the operation is decoupled from slower main memory and instead takes place primaril!.' with registers.

The speedup that can be achieved using regislcrs is clemtnistralci in Figure MD.; IPADE881, The FORTRAN routine multiplies VC1C. Or A by vector B to produce vector C, where each vector has a real part (AR, BR. CR) and an imaginary part (AI, 131, CI). The 3(190 can perform one main-storage access per processor, or clock, cycle (either read or write), has register that can sus1ain two acceNses fot reading and one for writing per *cycle,* and produeus one re ell per cycle in its arithmetic unit. Let us assume the use of instructions that can specify two source operands and a result: Part a of the figure shows that, with memory-to-memory instructions, each iteration of the compirlotion requires a **total of** 18 cycles. With a pure register-to-register architecture (part b). this time is reduced to 12 cycles. Of course, with togisler-lo-register operation, the vector quantities must he loaded into the vector registers prior to comp Llt ation]nd stored in memory rtcrviird. rt,r large vectors, this fixed penalty is relativel!,' small. Figure 18,18c shows that the ability to specify both storage and register operands in one instruction further reduces the ti me to 10 cycles per iteration. This latter type of instruction is included in the vector architecture?

'For die 370390 arLII LOMAc., the only three-opernnd inSLTLI.13LiOrLS rcizisler and stcir i tor taucocmr. R5) specify iwo 45prrands in registers :Ind one in memory. pan a of the txrimpte, we aNsuine ihe cx:51etice of three-144:1am] inaructions in which kilt operands are in main memory. This it done for puiposes of compari:46n rind, in fact. such an itutruccii.'511 format could have been clioseu for the vector archticcturc. r2orrspoulicl iimtructions, discussed subsucluen ly, airford u furtEsex reductiDri.

FORTRAN ROI.JTINEz

```
      Do [00 J 1,50
      CR(.1) = ARCO          A1(.1 j 4 B.1(.1)
100   C:10i = ARO)* BRIJ i Afib*BRtJ
```

Operation	Cycle.;
.AR(J) BR(1) -4 TI(J)	3
MO) * 5Iaj T2(J)	3
1· 1.{a) - 'r2(1) CRW	3
AR(J) 1' B1(3) 'no)	
AIO) . RRIJ) –)1.1-1.1)	3
T3(J) -F T4(J) -4 no)	3
TOTAL	

Stor;J.124: In storage

Operation	Cycles
AR (J) VI(J)	
VI0) f3R0,1 V2(3,1	
A10) -4 V3(1)	
V3(J)	
V20) - V40) —> V5(f	
V5(T) CR111	
V I (1) BRJ) Y60)	
V4(J)* BR.(f) -4 V70)	1
V6(.1') V7(J) VR(J)	
V&.0 C1(J)	
TOTAL	

{c) Slorrtgc 10 register

Vi = Vector regisW9
AR, TiR, AI, BI = Operands in memory
Ti = Temporary Ic.i42ations in memory

Operation	CycleA
ARO) · V 11.1)	
BR(T) › V2,0)	
V I I.JJ V2(.1) V3(1)	
Id) V40)	
1110i	
¹/40 # V5{J).	
V311) V6(.1) -4 V7))	
V7(J) —> OW)	
V1(J) V5(.1) VW,J:i	
V'JJ,I)* V2(J) V9(.1)	
I• 11{J1 + VW) -2) YO(J)	
Vrith • C41)	
TOTAL	12

Ib) Register to register

Operation	Cycles.
AR(1) -4 VI(Ji	
[(.1) * BR(.1) V2firF	
Al(j) V3())	
V2(J) V411* M(J) —) V2(J)	
V2(3) CRUI	
43)* i3R,f) –5 1/4(J J	
V.:111) - V30)* RR(J) V50I)	
V st,I} -4 Cit,r)	
TOTAL	

1:13) Compound instnietinn

Figure MIS Alternative Programs for Vector

Figure 18. t9 iiiustrates the registers that are part of the IBM .;()90 vector facility. There arc sixteen 32-bit vector registers. The vector registers can ako he coupled to form eighi 64-bit reets Jr registers. Any register clement can hold an inivor or Ilu1iting-point value. Thus, the vector registers may be used for 32-bit and 64-bit integer values, and 32-bit and (54-bit floating-point values.

The architecture specifies ihat each register contains from S to 512 scalar cie-MetIA- The choice of actual length involves a design trade-off. The time to do a vector operation consists essentially of the overhead for pipeline startup and register filling plus one cycle per vector ulernual, Thus, the use of a large number of register elements reduces the relative start tii time Cor a computation. However, this efficiency must he balanced aaainst the added time required for saving and restoring vector reg-

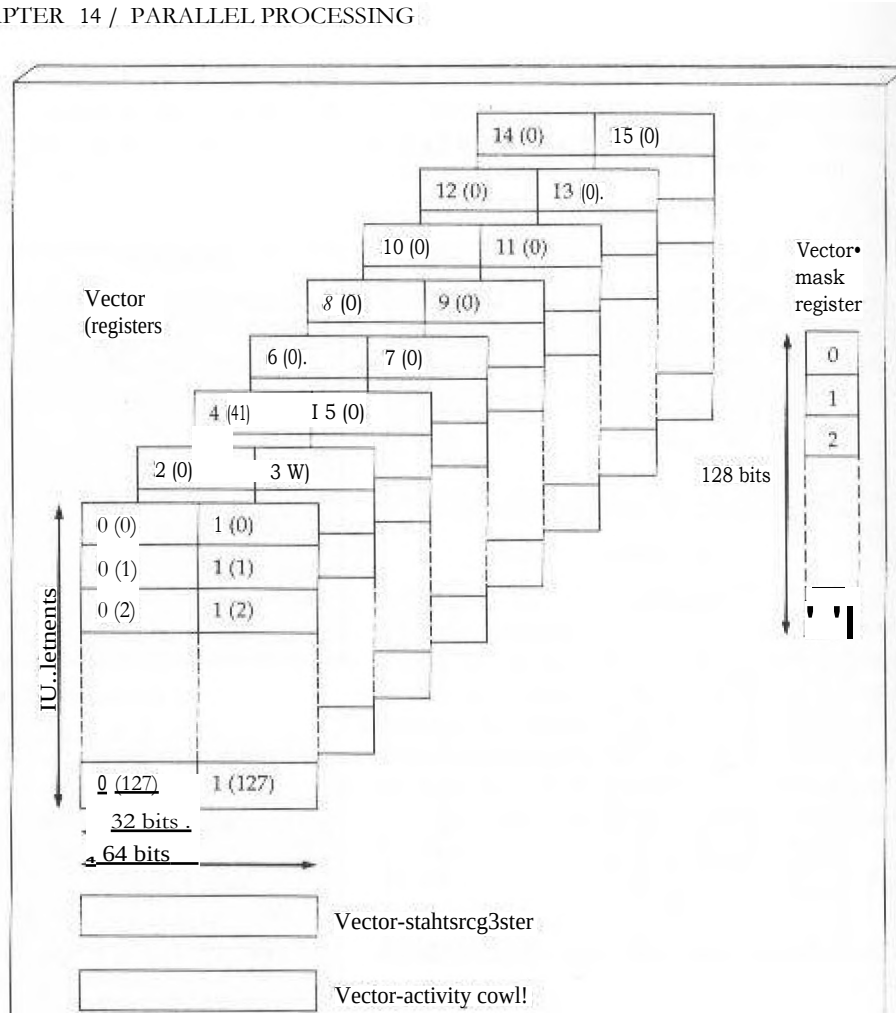


Figure MN 1:to0.7.wrh of the IBM 3090 [1]A2cior

inters on a proeuss swiwti and the practical cost space limits. Thcse considerations led to the use of J.28 elements per register in the curl ˉcp.!, 3090 implementation.

Three additional registers arc needed by the vector faeility. The. vector-mask register contains mask bits that may he used to select which eletricrits in **the vector registers are** to he processed for a particular operation. The Arrector-status register conlains control fields, sueh as the vector count, that determine how many elements in the vector registers are to be processed, The vector-activity counl keeps track of the time spent executing vector instructions,

Compound lnstructi m

As was discussed previously, instruction execul ion **can** be overlapped usiq chaining to improve performance. The designers of the vector facility chose

not to include this capubny for several reasons. The Systeni1370 architecture would have to be extended to handle complex interruptions (including their effect on virtual memory management). and corresponding changes would he needed in software. A more basic ismie was the cost of including the additional controls and register access paths in the 'ector facility for generalized chaining.

Inslead, three operations are provided that combine inter one instruction (one opcode) the most common sequences in vector computation, namely multiplication followed by addition, subtraction, or summation. The storage-to-register MULTIPLY-AND-ADD instruction. for example., fetches a vector from storage. muttiplies it by a vector from a'register. and adds the produci Lo a third vector in a register. By use of the compound instructions MULTIPLY-AND-ADD and MULTIPLY-AND-SUBTRACT in the example of Figure 18.18, the total time for the iteration is reduced from 10 to 8 cycles.

Unlike chaining, compound instruelions do not require. the use of additional registers for temporary storage of intermediate results, and they require one less register access. For example, consider the following chain:

A. —5. ·..-R.
\TRi - VR2

In this case, two stores to the vector register VR1 are required, In the. IBM architecture there is a slorage-to-register ADD instruct ion. With this instruetion. only the sum is placed iri VR1. 'The compound instruction ;i1 '..0 **Ihu 'iced LO** reflect in the machine-state description the concurrent execution of a number of instnictions, which simplifies status saving and restoring by the operating system and i ;he handling of interrupts.

The Instruction Set

Table EK.Lt summarii.es the aril hmetic and logical operations that are defined for the vector architecture. In addition, there are memory-to-register load and register-to-memory store instructions. Note that many of the instructions use a three-operand formal. Also, many instructions have a number of variants, depending on the location of Lhc operands. A source operand may be a vector register (V). storage (S). or a scalar register (Q). The target is always a vector register, except for comparison, ihe result of which goes MI° the '..ector-mask register_ With all Lhesc variants, the total number of ()Nodes (distinct instructions) is 171. This rather large number. however. is not as expensive to implement as might be imagined. Once the machine provides the arithmetic units and the data paths to feed operands from storage_ scalar rvgisl UN, and wool regis ters tU the vector pipelines, the major hardware cost has been incurred. The architecture eau. with little difference in cost. provide a rich set of variants on the use of those registers and pipelines.

Most of the instructions in Table l&4 are self-explanatory. The two summation instructions warrant further exphiMilion. The Liecuniulate operation adds together the elements *of* a single vector (ACCUMULATE) or the elements of the product of two vectors (MULTIPLY-AND-ACCUMULATE). These instructions present an interesting design problem. We would like to perform this operation as rapidly as possible. taking full advantage of the ALU pipeline. The difficulty is that

Table 1..4 HE Ril 3090 'Vector Arithmetic and I A:sgical Instruct ionS

Operation	Data Types: Floating Pcsiu1. Long	Floating Pcsiu1. Short	Binary or **Logical**	Operand Lociino			
A (Id	FL	FS	B1	V + V —5 v	V + R —2 V	Q , V V	Q - S •V
S LibtTHC I	H.	FS	1711	V - V v	V • S —s V	O V , V	0 - S ,V
ktatipiy	Ft	FS	Fil	V ...< V eV	v x v . V	0 x v >V	0 x 5 —s V
Divide	FL	FS		V ; V .V		0 1 V —5 V	01 S —5. V
(...oinpave	FL	FS	DT	V • V > V	V • S. —.5. V	0 • V —5. V	0 • S —). V
Muliiply arid Add	FL	FR			V+VxS —2V	V ! OxV-2.V	V I QxS-0/
Mulnply and Sub[ruct	FL	FS			V - VxS → V	V QxV-2'V	V QxS•-sV
hvfuliirly and AccimilihiLe	Pl.	FS	—	P 4 • V → V	F . • S • V		
Coi El pEC muni	FT.	FS	Y.11	.V —2 V			
Positive Abscrlutc	FL	FS	RI	i V j V			
Nr.gatve likbsolu(e	FL	FS	131	-IVI > V			
Maximum	FL	ES				0 - V O	
Maximum Absoluie	FL	FR				Q . V —2 O	
Minimum	14.	FS	-			0 - V ->0	
Swami RIAcal			(O	- V —> V			
Shift RiOL I ..ogical			1.0	- V —2 V			
And			LO	V & V —2 V	V & 5') V	0 et V -.> V	O & S V
tA..Z.			DO	v 1 v -... V	v I S --5 V	0 i V—> V	OIS —> V
I'.1; C.Itr51Ve-OR			LO	v ED V >V	V (X, 5 —2 V	O 'y'ils,l —2 v	Q 3) S —, V

ExpEraufwn: Petit Types

FL Lnrip, ,q puio I

1•S km , k puio I

Ri 112. m MAT

LO 1.(.9.1C;b1

Optracid1.1.1ilirions

✓ V:L:11u rt.pster

• SL' lar OCCICT21 Or fib i1 r'' Lut registor)

P w1 1 1:,<_ in _ .•

, L a

[11C Sum ;if two numl-Te:TN pul indo Ih4 pipeline is **obi wail ahlc until se sera[** cycles 12iler. Thus. the third element in the vector cannot he added to the sum of the first two elements until those two elements have gone through the entire pipeline. To overcome this problem, the elements of the vector are added in such a way as to produce four partial sums. In partictilm. elenicribt 0. 4, 8, 12.... , 124 are added in that order to produce partial sum () elements 5, 9, 13 L25 to partial sum 1: elements 2. 6, 111, 14 ..... 126 to partial sum 2; and eterner& 3, 7, 1I, 13 ..... 127 to pm LiA I sum 4. Fach cal' partial sums can proceed through the pipeline at top speed. because the delay in the pipeline is roughl :;' four cycles. A separate vector register is used to hold the partial sums. When all elements of the original vector have 1-34.:T: n proeu:ssed, the lour partial sums arc added togelher to produce the final result. The performance of this second pHs': is not critical. because only four vector elements are involved.

18.7 RECOMMENDED READING

e'ekrqe e..r_Feett.

ICATAN-I sury.:.ys the orinicipls of mulliproeessors and examines SPARC• based S7' ,r1Ps in detail. Shi,IPs Are. MS() covered some detail 01 [STON911 and I HWAN93].

[M1LELIttl is an overview of cache coherence algorithms and techniques for multiprocessors, with an emphasis on performance. issues. Another survey of the issues relating to cube coherence in multiprocessors is 11-11,193 .1. [TOMA931 contains rcprints of many of the key papers on the subject.

[PFIS98] ki.ksential reading for anyone interested in chimer": 1114 ': 1) Ok 00 the hardware and soliwairc desip] issues and conirasts clusiors 4ti I Ia SM ;old Elie boot also eohtains a solid technical description of SNIP and N I !VI A 1,l W-,1R11 1 ·N11...! ·.. A ihorough treatment of cliasers can' he fouitd in 1131.2(Y99A/ Ices technical or entsiers. with gated commentary cm .....niicatN ficJilltnefehl prOcilletS.

GOOLI discussions of vector computation can he found in I STONC31 and [HWAN:}31.

BUTY99it IlLtyva. *nigh PerPralancT (Mater ArchitecLarvs and Sysiems.* **Upper Sadc,11.L** 1[3]rentie2: Hall 1999.

ittriY99h Buyyti, R. High Pc.t.pm.n.ancci. C uvret Gokiipuriri.s.... *Programming and Applievr io ns.* Upper Saddle River, NJ: Prentice 999.

CATA94 Cala nw..aro, Mu *friprocessor* System A *rchitnenws.* unutin View, CA: Sunsoft Press, 1994,

11WA.N193 1.1wting, K. *Advanced* Computer Art 17/76.c.refre', Ne..Av York: MeGiraw-Hill, 1993.

J. M193 i,ilja, I). "Cache (..tihere.nce. in LariN -Seale. Shat red- Memory Multipreeessors: Issues and Comparisons," 4e.41 *(.'on paireg ,ti, rmy.s,* September t 99'3.

M11-1-E00 Mileukovie, A, "Aellieving High PO'ocmanee in Bus-Based Shared-Mernory Multiprocessors.' irEr July--September 2000.,

PFIS914 Pfister, G. *In Seafeli of Ousters.* Upper Saddle River, NJ: Prentice Hall, 1.998.

STON93 **Stone, H.** ***Ifigh-*** *Per fonnance Completer Arc hifecaere.* Reading, MA: Addison-Wesley.

T1)NIA93 Tonutsevie, M... and Miluiinovic, V. *The Cache Coherence Problem irE* ***ShOrd'i MetIWY*** *'112.4111PrOCZTSOn: Hardware. Solutions.* Leas Alairsit'N. WEE Computer St kits}' Pe ess, E 993.

Weptant. P. *Cleturts for High Apedlar,fay.* Upper Saddle **River,** Prentice Hall, 2001..

18,8 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

active standby	ME51 prompt)]	symmetric multiprocessor
enure 1:0.110 enee:	multiprocessor	(SWI P)
cluster	rumitui if Pi'm 1110.Mo ry	Uniform memory access
directory protociii	t1ccess (NUMA)	(LIMA)
tailback	passive standby	niproceSSOr
fail aver	snoopy protocol	vector r;i Ci I [t]...

Review Questions

18.1 Lis(rlrltl briefly define three types of computer system organization.

18.2 What are the chief characteristics of an SMP?

11L3 What are game of the potential advantages of an SMP compared with a uniprocesser7

18.4 What are some of the key OS design iisties for an sm

18.5 *tvVhwt* i4 the difference between software and hardwire cache 4 0111.:R!Ill Nc P11 lik057

18.6 What the Meaning (31 each of ale four states in the M !Will trt:i I?

18.7 What are some of the benefits of clustering!

181 Whal. is the difference between lailover and failbackl

18.1 What are the differences among U.MA. NUMA. and CC-NUMA?

Problems

181 Let a he the percentage of program code that can be executed simultaneously by G processors in a computer system. Assume that the remaining code must he executed sequentially by processor. Each processor has an execution rate of MIPS.

a. Derive an expression for the effective MIPS rate when using the system for exclusive execution of Ibis 11441)41;1in, in terms of *o.* re, and x.

b. If x — l(t and x — 4 i11 value of a that will yield a system performance of 40 MIPS.

18,2 A mulliprocessor with eight processors has 20 attached tape drives. Them are A Earge number of jobs submitted to the system that each require a inasim um of lour tape drives to complete execution. Assume that each job stasis running with only three tape drives for a Long period before requiring the fourth tape drive for a short period toward the end of it (Pwation. Also assume an endless supply of such jobs.

a. Assume the scheduler in the OS will not start a job unless there are four tape drives available. When as jo1w is started, four drives are assigned immediately and are not released until die job ftnishes. What is the maximum number of john that can he in progress at once? 'What arc the waximuin and minimum number of tape drives that may be left idle as a result of 111k policy?

b. Suggest an alternative policy to improve Ea lie drive milintion and at the same time avoid system deadlock. What is the ma kunum number of jobs that can be in progress at once? What are the bounds 4)11 the number of tape drives.?

18.3 ***Can*** you toresee any problem with the write-once cache approach on bus-based multiprocessors'i III so. 'LLiggc.gt a soimion,

18.4 Consider a situation in which two processor5 in an SMP configuration. over time, require access to the same line of data from main inerrioq. Both processors have a cache and use the 'IES] protocol. Initially, both cachoti have an invalid copy oldie line.

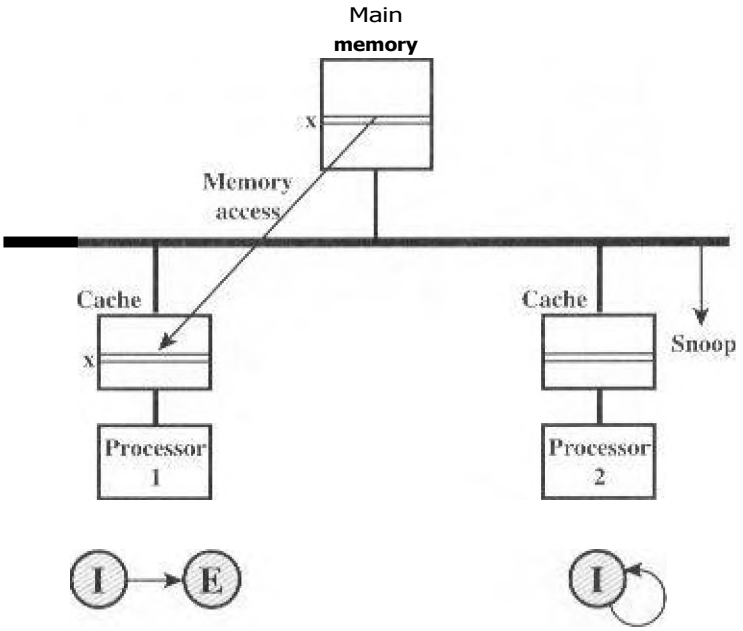


Figure 18.20 MEM 'Example: Processor E Rads Line x

figure 18.20 depicts the consequence of a real of line x by Processor Pl. If this is the start or a sentience of aecessei. draw the. qubsequeni figures for the following sequence;

1. P2 reads x.
2. PI writes to x (for clarity, label the line in F1's cache x').
3. PI writes to x (label the line in Pl cache
4. P2 reads x.

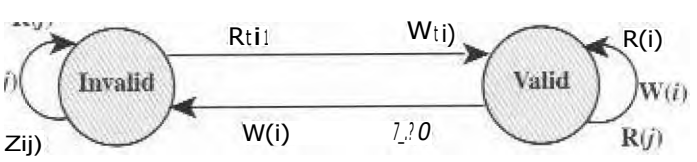
18.5 Figure 18.2 I shows the state diagrams of two possible cache cohercnce protocols. Deduce and explain each prolocol, and compare each to MESA.

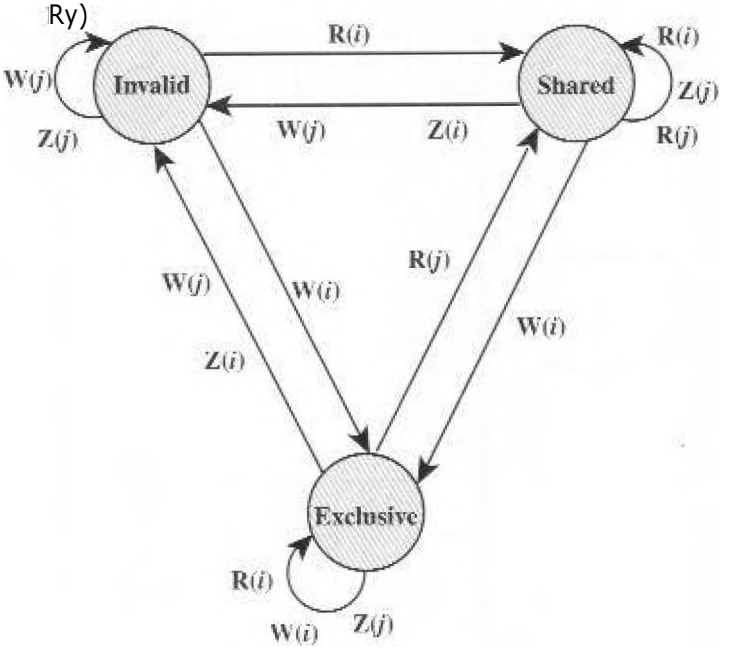
18.6 Consider an SNIP with both Li and L2 caches using the MESI protocol. A4 42A.P 1;:iiiit.hl in Section 183, one of flour states is associated with each line in the L2 cache- Are all four stales also needed for each line in Ihc. LI cache? ff sea, why? If explain which state or states can he elitninalcd.

18.7 table 18.1 showc ih1 ire lotruanee of a three-level cache arrangement for the IBM The purpose of his problem is to determine whether the inclusion of the third level of cache seems woriliwhile. Determine the access penalty (average number of PLT cycles) for a system with only an LI cache, and normalize that value to 1,0. 'then deiermi IIQ the normalised *access* penally when both an LI and L2 cache are used. and the access penalty when all three caches are used. Note the amount of improvement in each case and state your opinion on the value of the L3 cache.

18.8 The following code segment needs to he executed 64 tiIne.s for ilic.1.1.....valtimion Of the vector arithmetic expression; D(I) = A(I) F B(I) x 1.:(1) for 0 I 63.

```
        RI ;  Brn                .(=              + I)/
  :..ad R2,   Crn                                43. ⊦ I)/
p.ry;.L.jr„.1.,.  P1   P2                    I  X
Inad     ,                /23     Eeraory     I  :I
     IRS.                 iR3     (R3I  F  :9.11/
1,˜..ad DI,  R3           /mencry (0           (R3)/
```





W(rj= Write to line by processor i'
R(i) = Read line by processor *i*
Zia = Displace line by cache *i*
14I1,0= Write Lo line by processor] (j1`
R(/) = Read line by processor j
Zij) = Displace line by cache j Co

Note: State diagrams are fur a
given line in cache *i*

Figure 18.21 Two Cache Coherence Protocols

where R. R2, and R3 arc processor registers, .;ind a, u, -y, (.1 arc the starting main memory addresses of arrays B(I), C.J). A(I), and D(I), respectively. Assume four clock cycles for each Load or Store. two cycles for the Add. and eight cycles for the Multiplier on either a uniprocessor or a single processor in an SIMD machine.

- Calculaid lcilayl n umber or processort:yeles needed to execute this code se ment repel`0.21.1[y r:14 li mes (ˆ11 SISD uaiprocessor computer sequentially, ignoring other time delays.

b. Consider the use of an SIMD computer with 64 processing elements to execute the vector OperaLions in six synchronized vector instructions over 64.cornponent tor data and both driven by the same-speed dock. Calculate the total execution time on the SIMD machine. ignoring instruction broadcast and other delays.

- Wliat 11ic speedup gain of the Sl corn:puler over the SISD eoropuler?

18.9 Produce a vectorized version of the follawing program:

```
          20 _      1
    1   ,  1:      3
    D3 10 J - 1, A
    At:t  = MI; +          .7)  x          :..r)
7:3 C.3:9'17/.17.:E
          = FAT.: + Al.T)
2 3 ;-...3:.7.11.17.:E.
```

18.10 AJ1 application program is executed an a nine-computer cluster. A benchmark pro. grant took time T on this cluster. Further, it was found that 25% of *T* was time in which the application was running simultaneously cm all nine computers. The remaining time. the application had to run an a single computer.

- Calculate the efleutiv4.: speedup under the aforementioned condition as compared with exec:ming the program on a single computer. Also calculate *u.* the percentage of code i bat liiiI A1.1..tlizod (programmed or copripiled so .a410 use ilk cluster mode) in the precedi rig program.

b. Suppose that we are able to effectively use 18 computers rather than computers on the parallelizcd portion of the code. Calculate the effective speedup that is achieved.

18.11 The following FC1 E TR AN program is to he executed on a cimputer, ark] a parallel versIou is to he ONL•Ci.11124.1 011 cluster.

```
:          DO LC I =  I, 132C
:                I    =
L3             no 2C. .2     1,
L4    20          SUM ;1} -      : .1 _
L : 1O    commun
```

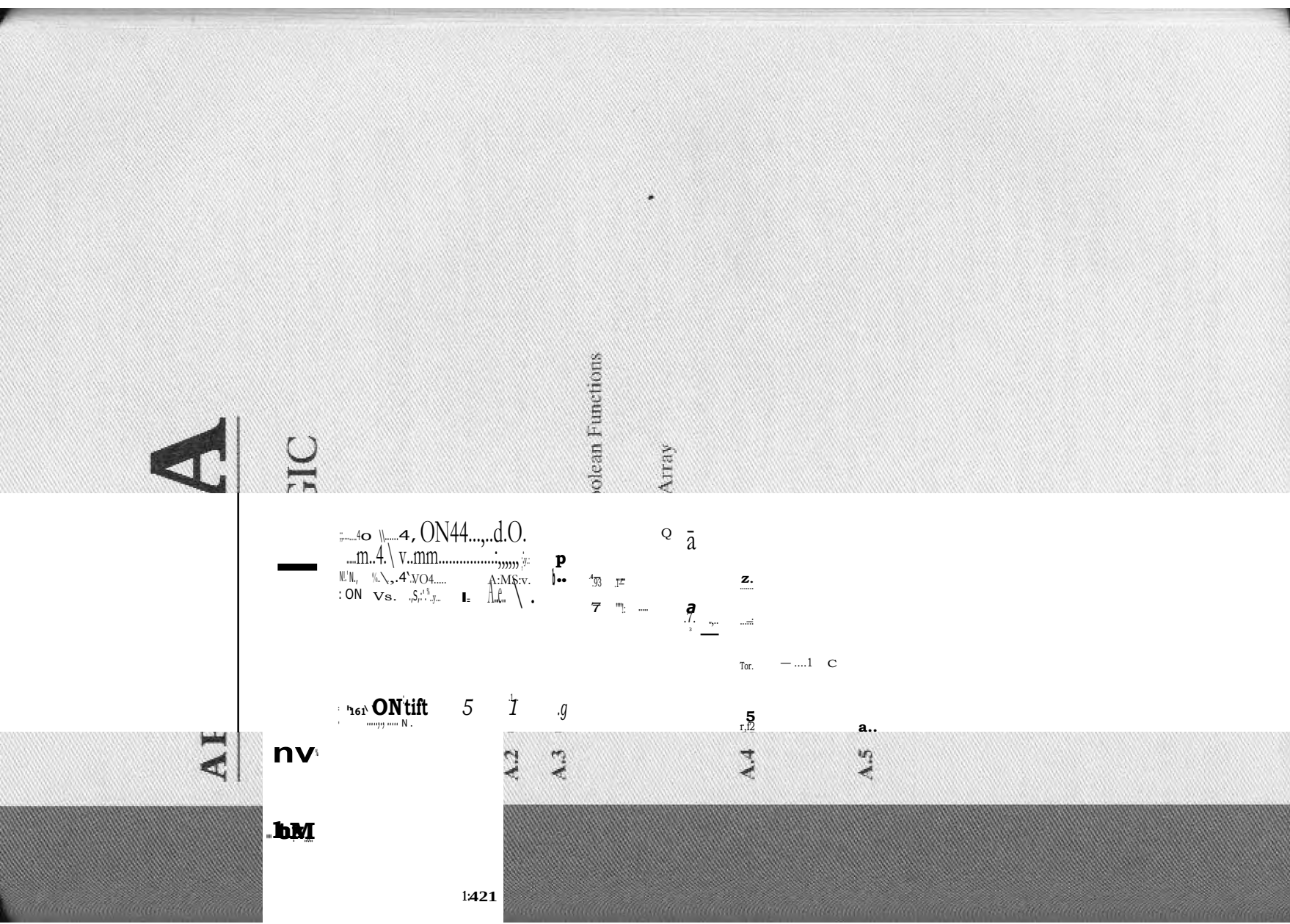
Suppose lines 2 and 4- each take two machine cycle times, including all processor and rneroury-aecess activities. Ignore the overhead caused by the software loop control statements (lines I, 3 ; 5) and all other system overhead and resource conflicts,

a. What is the total execution time (in .111aelli ɪ1 cycle times) of the program on a single computer?

b. Divide the 1-loop iterations among the 32 computers as follows: Computer I executes the first 32 iterations (I I to 32), processor 2 executes the next 32 iterations. and so on. What are the execution time and speedup factor compared with part (all (Note that the computational workload, dictated by the J-loop, is unbalanced aiming the computers.)

c. Explain how to modify the parallelizing to facilitate a balanced parallel execution of all the computational workload Mier 32 computers. By a balanced load is meant an equal number of additions assigned Lo each computer with respect to both loops.

EL What is the minimum execution time resuicinig from the parallel execution an 32 computers? What is the resulting speedup over a single computer?


A
GIC
oolean Functions
Array
A.2
A.3
A.4
A.5

The operation of the digital computer is based on the storage and processing of binary data. Throughout this book, we have assumed the existence of storage elements that can exist in one of two stable states and of circuits that can operate on binary data under the control of control signals to implement the various computer functions. In this appendix, we suggest how these storage elements and circuits can be implemented in digital logic, specifically with combinational and sequential circuits. The appendix begins with a brief review of Boolean algebra, which is the mathematical foundation of digital logic. Net the concept of a gate is introduced. Finally, combinational and sequential circuits, which are constructed frinn gales, are described_

A.1 BOOLEAN ALGEBRA

The digital circuitry in digital computers and other digital systems is designed. and its behavior is analyzed, with the use of a mathematical discipline known as *Boolean algebra.* The name is in honor of an English mathematician George Book, who proposed the basic principles of this algebra in 1854 in his treatise. *An Investigation of the Laws of Thought rvci Which to Found the Mathematical Theories of Logic and Probabilities.* In 1938, Claude Shannon, a research assistant in the Electrical Engineering Department at Mi suggested that Boolean algebra could he used to solve problems in relav-switching circuit design [SHAN38]. Shannon's techniques were subsequently used in the analySis and design of electronic digital circuits. Boolean algebra turns out to he a convenient tool in two areas:

- Analysis: It is an economical way of describing the function of digital circuitry.
- Design; Given a desired function, Boolean algebra can be applied to develop a simplified implementation of that function.

As with any algebra, Boolean algebra makes use of variables and operations. In this case, the variables and operations are logical variables and operations. Thus, a variable may take on the value 1 (TRUE) or 0 (FALSE). The basic logical operations are AND. OR, and WYE which are symbolically represented by dot, plus sign, and overbar:

$$\text{A AND B} = A \cdot B$$
$$A \text{ OR } B = A + B$$
$$\text{NOT A} = \bar{A}$$

The operation AND yields true (binary value 1) if and only if both of its operands are true, The operation OR yields true if either or both of its operands are true. The. unary operation NOT inverts the value of its operand_ For example, consider the equation

$$D = A + (B \cdot C)$$

D is equal to 1 it' A is 1 or if both B = 1) and C — 1. Otherwise D is equal to 0_

Several points concerning the notation are needed. In the absence of parentheses, the AND operation takes precedence over the OR operation. Also, when no

Table A.1 Boolvan Opt raters

P	Q	NOT P	p AND o	i oft Q	P XOR Q	P NAM) 41	P NOR Q
Li	0	1	0	0	0	1	L
Li	1	1	0	1	I	1	0
I	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

ambiguily will occur, i he AND operation is represented by simple concatenation instead of the dot operator. Thus,

$$A\text{ -F }R \cdot C = A\text{ -F }(B \qquad \text{— }A\text{ I- }BC$$

all mean 'Take the AND of l and *C:* then take the OR of the result and

fable A.1 defines the basic logical operations in a form known as a *!mat ?able.* which simply Lists the value of an operation for every possible combination of vat-ties of operands. The table also lisls three othcr useful operators: XOR, NAND, and NOR, 'The exclusive-or (X0R) of two logical operands is 1 if and only if cmictEv one of the operands has the value 1. The NAND function is tilt:. complement (NOT) of the AND function, and the NOR is the complement of OR:

$$A \text{ NAND } B = \text{NOT}(A \text{ AND } B) = AB$$
$$A \text{ NOR } B \text{ — NOT}(A \text{ OR } B) = \overline{A + B}$$

/V; we wh ill see, these three new operations can be useful in impiementing certain digital circuits.

Table A.2 summarizes key identifies Dr Boolean algebra. The equations have been arranged in two columns to show the complementary. or dual, nature of the AND and OR operitions. Thieve are two classes of idenlities: basic rules (or pi mu-kiwi), which are stated without proof, and other identifies that can be derived from

Table A.,2 Basjc Identititlsof Boolean Aber bra

Bahie Postulates		
A .13 = 13 • A	A • **B B I A**	**(.:41011nutatilio** tag s
A . (B + C) = **B)** + (A	A — (B•r.) = (A I 13) • (A +	Distributi',0 laws
l•A=A	+ A = A	Ide.n(itle eterneuts
A • A- 0	A— A= 1	**Invursu** elements
Other Identities		
0 • A = 0	**- A —**	
A • A = A	**A — = A**	
A • (B • C) — (A • B) • C	**A — (B C) (A I B) C**	Associative taws
A•B ¯ A 1 B	A — B = **A•** Fi	DeivimTan's theorem

the basic postulates. The postulates define the way in which Boolean expressions are interpreted, One of the two distributive laws is worth noting because it differs from what we would find in ordinary algebra:

$$A - (B \cdot C.)\ (A + B) \cdot (A + C)$$

't he two bottommost expressions are referred to as DcMorgan's theorem. We can restate them as follows:

$$A \text{ NOR } B = A \text{ AND } B$$
$$A \text{ NAND } B = A \text{ OR } B$$

The reader is invited to verify the expressions in Table A.2 by substituting actual values (Is and Os) for the variables A. B, and C.

A.2 GATES

'lite fundamental building block of all digital logic circuits is the gate. Logical runt: lions are implemented by the interconnection of gates.

A gate is an electronic circuit that produces an output signal that is a simple Boolean operation on its input signals. The basic gates used in digital logic are AND, OR, NOT, NAND, and NOR. Figure A.I depicts these five gates, Each gate is defined in three ways: graphic symbol, algebraic notation, and truth table. The symbology used here and throughout the appendix is the IEEE standard, IEEE Std 91. Note that the inversion (NOT) operation is indicated by a circle.

Each gate has one or two inputs and one output. When the values at the input are changed, the correct output signal appears almost instantaneously, delayed only by the propagation time of signals through the gate (known as the *gate ticiay)..l he* significance of this is discussed in Section A.3.

In addition to the gates depicted in Figure A.1, gates with three, four, or more inputs can be used. Thus, X + + Z can be implemented with a single OR Eate with three inputs.

Typically, not all gate types are used in implementation. Design and fabrication are simpler if only one or two types of gates are used. 'thus, it is important to identify *func:imwtiv complete* sets of gates. ' Ms means that any Boolean function can be implemented using only the gates in the set. The following are functionally complete sets:

- AND, OR, NOT
- AND, NOT
- OR. NOT
- NAND
- NOR

It should be clear that AND, OR, and NOT gates constitute a functionally complete set, because they represent the three operations of Boolean algebra. For the AND and NOT gates to form a functionally complete set, there must be a wri!,,

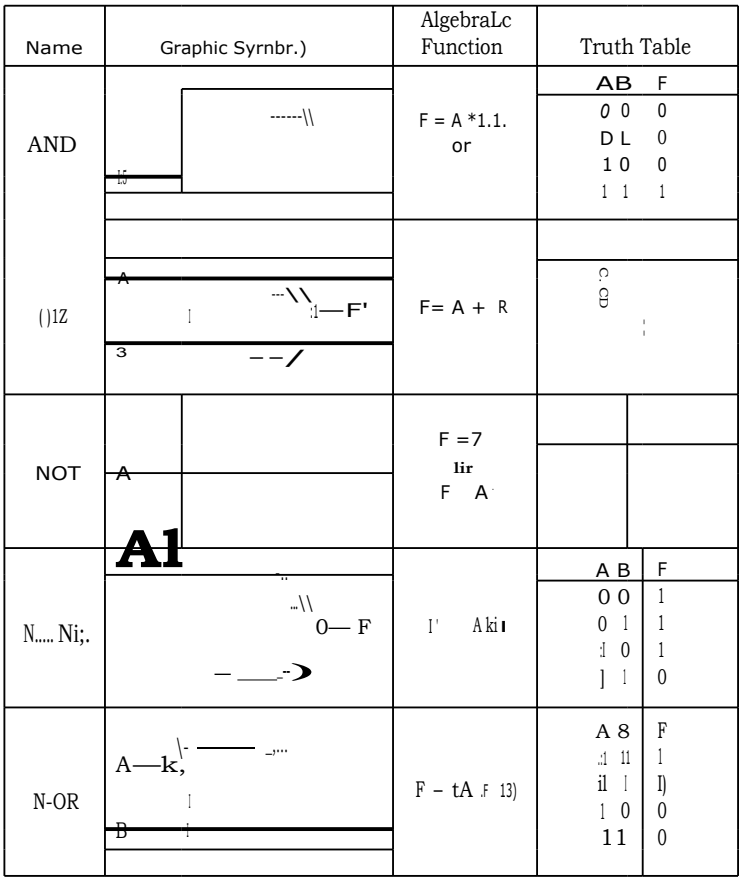


Figure A.1 Basic 1..o & Ciati2s

to synthesize the OR operation from [he AND and NOT p gan&1 6 can be done by applying DoMorgan's theorem:

$$A = \overline{\overline{A}}$$

$$A \text{ OR } B = \text{Nur(NOT } A) \text{ AND (NOT } B))$$

Sintil;D-Iv,11-te. OR and NOT operations leinctionally complete because they can be used to syntheMze. thc A N I) operation.

Figure A.2 shows how the AND, OR. and NOT functions on he implemented solely m NAND gates, and Figure A.3 shows the same thin2 for NOR gates. For this reiMon, ciretrils can be. and frequently are. implemented solely with NAND gates or soleiy with NOR gates,

With gates. we have reached the most primitive level of computer science and engineering. An examination of ti-ic transisior combinaliorts med Lo construct gates departs from that realm and enters there6molelectrical engineering. For our purpows, however. we are content to describe how gates can be used as building N ocks [0 implement the essential logicalcircuits ot a digitai computer.

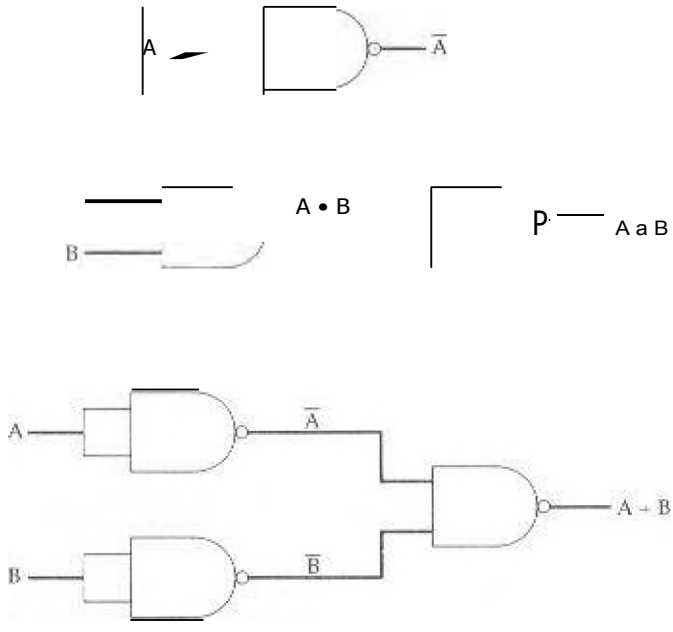


Figure A.2 The 1.4.c. of NAND Goau.s

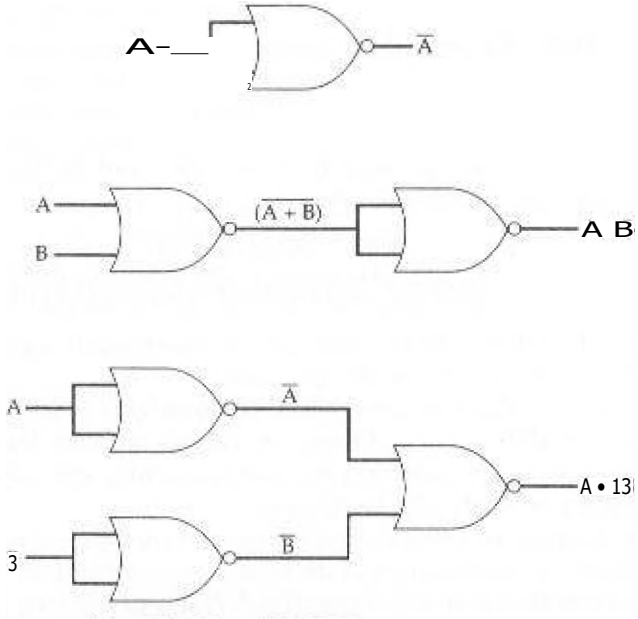


Figure A.3 The Ilse of NOR Gates

COMBINATIgNAL„c1BWIS,

ferf 05,e,4"O-

A combinational circuit is an interconnected set of gates whose output at any time is a fund:ion only of the input at that time. As with a single gate, the appearance of the input is followed almost immediately by the appearance of the output, with only gate delays.

In general terms, a combinational circuit consists of *n* binary inputs and binary outputs. As with a gate. a combinational circuit can be defined in 1M ˙cc ways:

- **Truth table: For** each of the 2' possible *eon* il)i nations of input signals, the binary value of.each of the. *m oinpul* signaEs is listed.
- **Graphical symbols:The** interconnected layoui of gates is depicted,
- **Bonleuri equaltinns:** ouipui signal is expressed as a Boolean function of its input signals.

Implementation of Boolean Function;

Any Boolean function can be implemented in e]eetronie form as a network of gates. For any given function. there are a number of alternative realizations. Consider the Boolean funel ion represenied by the truth table in Table A.3. We can cxptcss this function by simply itemising the combinations of values of A, B. and C that cause F to be 1:

$$ABC + ABC — ABC \qquad (A\text{-}I)$$

There **are** three combinations of inpul values That cause F to be 1, and if any one of these combinations occurs, the **result is 1.** This form of expression, for sell-evidQni reasons, is known as the *sIOR2 of preVillen* **(SOP) form. Figure.** A.4 shows a straightforward implementation with AND. OR. and NOT gaits-

Another form can also be derived from the 1ri.all **The SOP** form expresses that the output is 1 if any of ihe input combinations that produce] is true.,

'rabic A.3 Boolean Function *or* Three Variables

A.			
0	0	0	0
U	0	1	0
0	1	0	1
0	1	1	1
t	0	1)	0
1	0	I	[I
I	1	0	1
1	1	1	{)

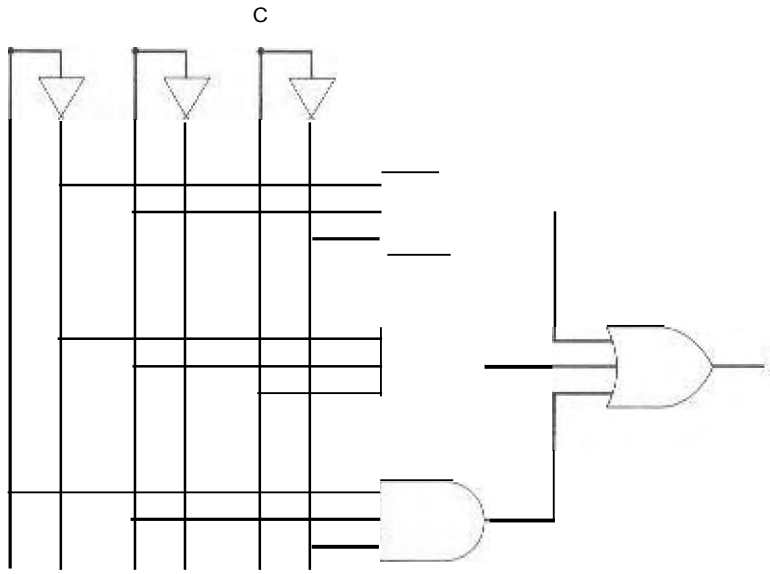


Figure A.4 Sain-nr-PrOLLUCLS limpleincinatiun of rabic A.3

We can also say that the output is l if none of the input combinations that produce 0 is true. Thus

$$F = (\overline{ABC}) \cdot (\overline{A1\text{-}3C}) \cdot (\overline{ABC}) \cdot (ABC) \cdot (ABC)$$

This can be rewritten using a generalization of DeMorpn's theorem:

$$(\overline{X \cdot T \cdot Z}) = X + y\ I\ Z$$

$$\begin{aligned} F &= (A - B + C) \cdot (A + + C) - (A\ r\ B + C) \cdot (A - B + C) \cdot (A + B - C) \\ &= (A\ B\ C) \cdot (A + B + C) \cdot (A + + C) - (A - B\ C) \cdot (A + B \end{aligned} \qquad (A,2)$$

This is in th.c. *product of sums* (POS) form, which is iliustratEA in Figure A.5, For clarity, NOT gates are not shown. Rather. it is assumed that each input signal and its complement are available. si mplifies the logic diagram and makes the inputs to the gates more readily apparent.

Thus, a Boolean function Cart be realized in either SOP or POS· form, At this point, it would seem that the choice would depend on %teaselr the truth nlble con- [Bins. more Is or Os for the output function: The SOP ha ·; one term for each I, and the POS has uric [erne for each 0. However, there are other cOnNidcn) [ions:

* It is generally possible to derive a simpler Boolean expression front the truth table than either SOP or POS.

s It ma!•' be preferable to implemen1 the Function with singie mate type (NAND or NOR).

The significance of the first point is that, with a simpier .1 ℨ(..iolean expression, fewer gates will be needed to implement the funei ion. Three methods that can be used to Liehievc-.simpli fic2ii ioa are as follows:

- Algebraic; simplificalion
- Kan˙Laugh maps
- Quine—tvfcKluskey tables

AlgebraicSimplification

Algebriic .sirnplirimlin involves the application of the identities of Table A.2 to reduce the Boolean expression to one with fewer elements.. For example , eonskier again Equation (Al). Some thought shouid convince the reader that an equivalcut cxpressdon

$$= AB. \text{ h } HC \qquad (A.3)$$

Or, even simpler,

F — B(A

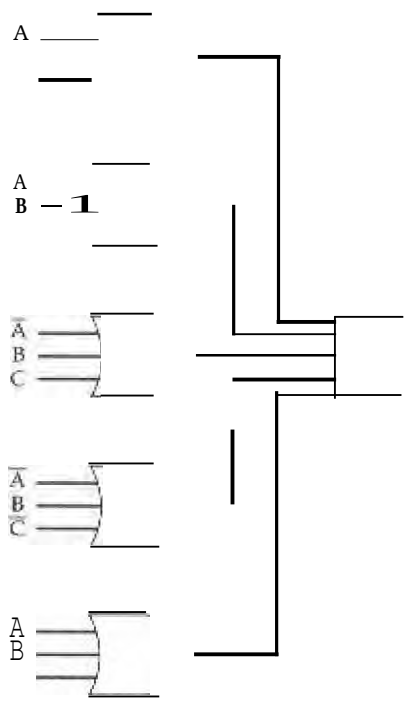
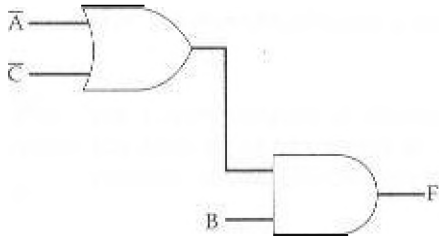


Figure A.5 Pmduci-M-Surns Impleinentation of Milk. A.3

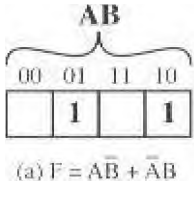


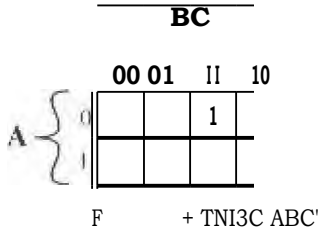
rigurc A.6 Siitsplirie.!ij linplernentatinn of Table A.3

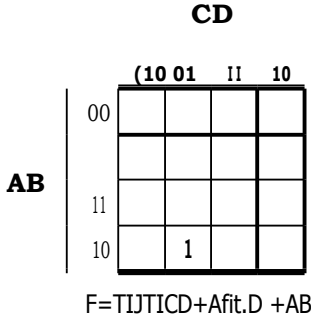
This expression can be implemented as shown in Figure A,6, Thu simplification of Equation (A.1) was done essentially by ob.servation. For more complex expression, some more syslemalic approach is needed,

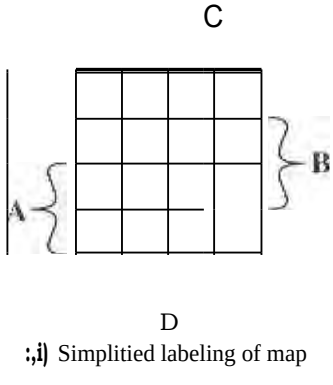
Karnaugh

For purposes of simplification, the karnaugh map is a eonvenimt way of represeni ing a Boolean rum:lion of ;I small number (Lip to four to six) of variables. The of 2^9 stin.nres : representing the possible combinations of values oaf n binary variables. Figure A,7a shows the map of four squares for a fund ion of ['o/c) variables. It is convenient for later purposes to List the combinw ions in the order 00,01,11,10. Because the squares. corresponding to the combinations are to be used









:,i) Simplitied labeling of map

Figure A.7 The Usc. cal Karnaugh Maps to Represent liooleart Functions

for recording miormation, the combinations are customarily written abuvc the squares. In the case of three variables, the representation is an arr ffigemcul of edit sq rC (figure A,7h), with the values rot arse of the variables to the left and fur the other two variab]es aloe the squares. For tour variables, 16 squares are needed, with the arrangement indicated in Figure A.7c,

rue map clan he used tt) repi senl any HooIean tunctioa iai the following way. Each square eos'respcaiqds to a unique product in the suns-oil-products Foria. with L 1 value corresponding to the variable and at) vt1lue correspunc]ing lu the; NO`I of that varutplc-['hus, l hc: p, cuiuct A13 co rresponds Lo the fourth square in figure A.7a. For each such product in the i'unction. I is placed in the corresponding square, Thus, for the two-variable example, the. map corresponds to AB 4 AB. Given the truth [able of a Boolean function, it is an Easy 1natter to Construct the map' h r each corn hirla tiun f v:il ue:s of rarabies that produce rwoilt trl' 1 its the truth Lah]c. Id] in the eOr-responding square of the map with 1, Figure. A.7b shows the result for the truth table of Tah[e Al To convert from a Boolean expression lO a map, it is first nCC-C:axbo t 1puL the xprc n in whaL is rckrred It a uaWwmcaI tort; Each tcrns in t]ie expression rrtusL contain each variable. So, for example, if we]lave nehiuLtion (A3), we must first expand it into the lull [01111 of Equation A. I } and Ihen eonvi:rl this to a map,

[}rc l:ffbeuag used in Figure: A-7d cmphxsi/es the relationship between variables and the rows and colurntis of the. map. Here the two rows embraced by the symbol A are those in which the v{triable A has the value [; the rows not mhraecd by the symbol *A* are those in which *A* is l); si milarly I'or H, C. and D-

Once the map of function is crested, we Girl o ften write a Simple algh raic expression for it by noting the arrangement of the is on the map-'l he principle is as follows, Any Lwcr syu;lres than l are adjncea I. dil'fgr in or]]} time of the variables, It two 2d pace rut squares both have an cniry of one. then the corresponding product terms differ in only one variable. In such a case, the two terms can he nlerged by C lirni-nating t]iat variable. For_exa_mp]e, in Figure A,f+a, the two iid jxcnt so^u.ires cc^r'rL-sprnd Lu the two terms ABCL) and Af3C'ii'I hus, the functdrrr expressed is

AF3CI)—.Al3Cf)—.A BD

This process can be extended in several ways. First, the concept of adj acency can be extended to include wrapping around the edge of the map- Thin, the top SgLU'IFC lI I a 00]umrr is :idj aecnt to the bottom square, and the leftmost square of a row is adjacent to the rightorost square. These conditions are illustrated in Figures A,tlh and c. Second, we can group not just 2 squares but " adjacent sgmares 1,1 h 1t is, 4, . etc,}. The next three examples in Figure A. show groupings of 4 sq ures- Note thaL in this ca se, [WO of the variahkcs can he ell ninated, The last three examples show groupings of t3 squares, which allow three variables to be eliminated.

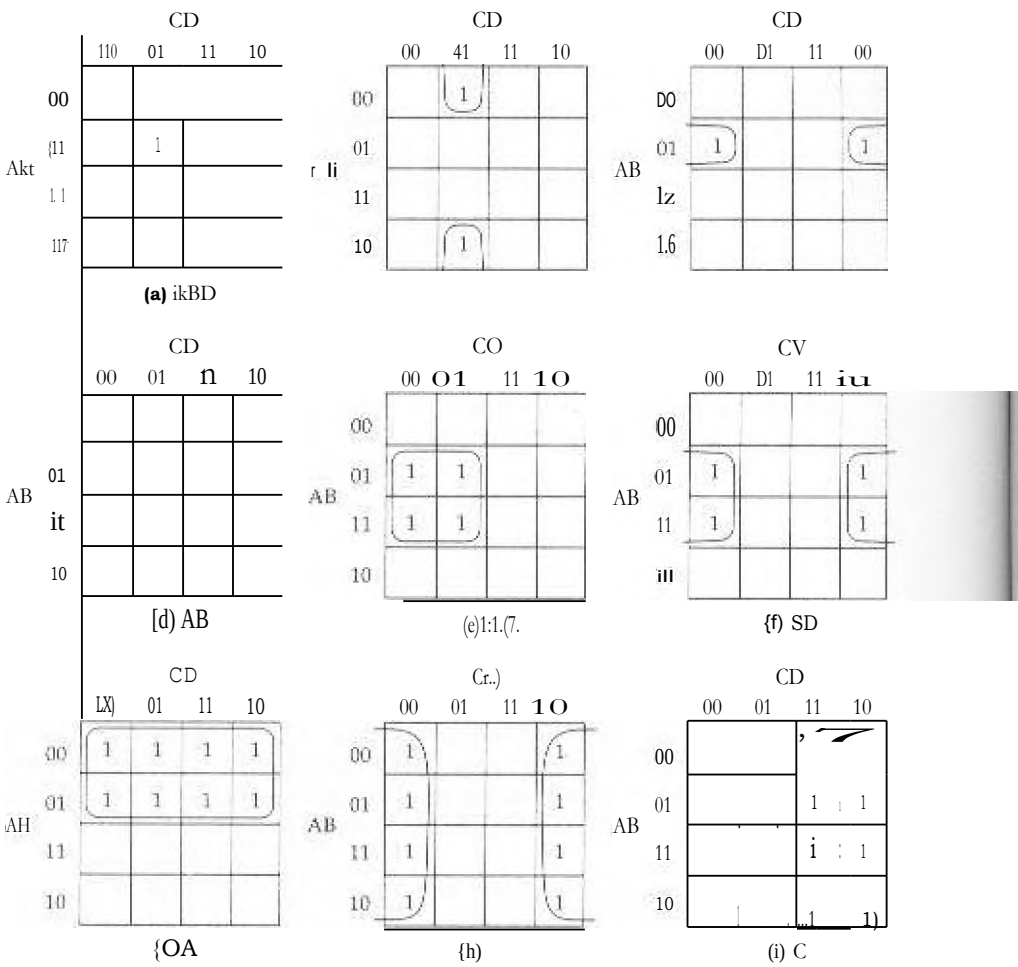
We c in summarize the rules for simp]iIicaliorr as follows=

1. Among the marked sgLrares (squares with a 111, find those that belong lu a unique largest block of either I. 2, 4, or S and circle those hlcacks-
2. Se]ect additional blocks of marked squares that are as 3arge as possible and as few in nLimber as possible, but include every marked square at icasl once. I he

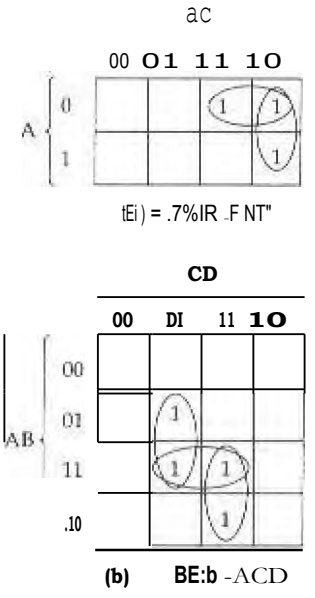
Tc:liu.11,s may not bc. unique in some. f.r.ase. roe C.X2ruple, if a rn.irked :iquarc combines with exactly two other squares, and there is no fourth [narked square to complete a larger group, then there is a choice to he made as two which of the two groupings to choose. When you are circling groups, you arc. 4iLlowed u.sc the same I valui: more [han once.

3. Continue to draw loops around single marked squares, or pairs of atipiccut markci mium es, or group or lour, eigh I, H;nd so on, in such a way that even. marked square belong!, to at least one loop: then use as few of these blocks as possible to include all marked squares.

Fiaure A.9a. based on Table. A.3, i I I us.111iii2.S Lhe process. If any isolated Is remain after. the groupings, then each of these is circled as a group of ls. Finally, before going from the map to a simplified Boolean expression. any group of Is that



ngure A.8 Tha (Jse of Karnaugh Maps



Fiore 4,9 **OWfiappillt** Groups

is completely overlapped by other groups can be eliminated. This is shown in Figure A.9b. In this case.. the horizontal group is redundant and may be ignored in creating the Boolean expression.

One additional feature of K.rnaugh arnaps needs to he mentioned. in some cases, certain combinations of values of variables never occur. and therefore the corresponding output never occurs. These are referred to as "don't care" conditions.)"arr each such condition, the letter "d" is cnlere(' into corresponding synire of the map. In doing t he grouping and simplifieaiitin, C;JT1 lac ireuted as a 1 or 0, whichever leads to the. simplest expression.

An example, presented in [HAYE94 illustrates the pi.sints we have been discussing. We would like to develop the Boolean expre:isions for a circuit that adds 1 to a packed decimal digit. Recall from Section 9.2 that with packed decimal, each decimal digit is represented by a 4-hit code, in the obvious way. Thus. 0 = 0000, -- 0(101. . , 8 = 1000, and 9 = 1001e The remaining 4-fait vaiIues, from 1010 to 1111, are not used. This code is also referred to as Binary Coded Decimal (BCD).

Table A.4 shows the truth table for producing a 4-bit result that is one more than a 4-bit BCD input, The addition is modulo W. Thus, 9 J = 0. Also, note 1.114i1 si>z of the input codes prod = '[4]don'1 care" results, because those ,1 e not valid BCD inputs. Figure A.10 shows the resulting Karnaugh maps for each of the output variables. The d squares are used to achieve the best possible groupings,

The Onine—rtickluskev Method

For more than four variables. the Karnaugh map method becomes increasingly cumbersome. With five wiriables, two 16 1fi mar, are needed, with one map con-

Table 4.4 Troll Table ror the One-Digit I'ïckdd 1)(3einwil InentunAli.2r

	lopui					Outpui			
Numhey	A			I) Number W		X			
0	(1	0	0	1)	1	0	1)	0	1
I.	(1	0	0	1	1	0	1	1	0
2	1)	0	1	0	3	(1	0	1	1
.3	0	0	1	1	1	0	1	0	0
4	0	1	0	0	5	(1	1	0	1
5	0	1	0	1	6	(1	1	1	U
ep	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	L	0	0	G.
8	1	0	0	0	9	L	0	0.	1
9	1	0	0	1	0	0.	0	0	0
	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
Dint 1 Cate 4	1	1	0	0		LI	cl	LI	il
CLM- ditiorl	1	1	0	1		LI	41	ш	ı
	1	1	I	0		kl	d	LI	LI
	1	I.	I	1		41	d	LI	LI

sidereci to be on top of the oihcr in Ihrce diniciisions if) 4LCIlieVi;: Udi4JCCriCy. Si% variables requires the use of four 16 16 tables in four dimensions! An alternative approach is a tabular technique, referred to as the Quine—McKluskey method. The method is suitable for programming on a computer to give an automatic tool for 'producing minimized Booiean QxprcsRions.

The method is best explained by means of an example. Consider the following expression:

$$ABCD\ ABCD + ABCD + ABCD - F\ ABCD + ABCD + ABCD — ABCD$$

Let us assume. ihn1 this c.xprt:N;71ion was derived from a truth table. We would like, to produce a minimal cxpi csNion xuiia,hlu for implcincrna Lion with ga1cs.

The first step into construct a table in which ach row eorrespornis to one of the product terms of the expression. The terms arc grouped according to the number occorriplemenlc'd variables. That is, we start with the term with no complemenIA. if it exists, then all terms with one complemem, and so on. Table A.5 shows the list f(1 r our example expression, with horizontal lines used to indicate the grouping. For clarity, each term is rcprcs.cnied by a for.each uncomplcinenied variab.14.2 and a for each complemented variable. Thus, we group terms according to the number of Is they contain. The index column is simply the decimal equivalent and is useful in %vhat follows,

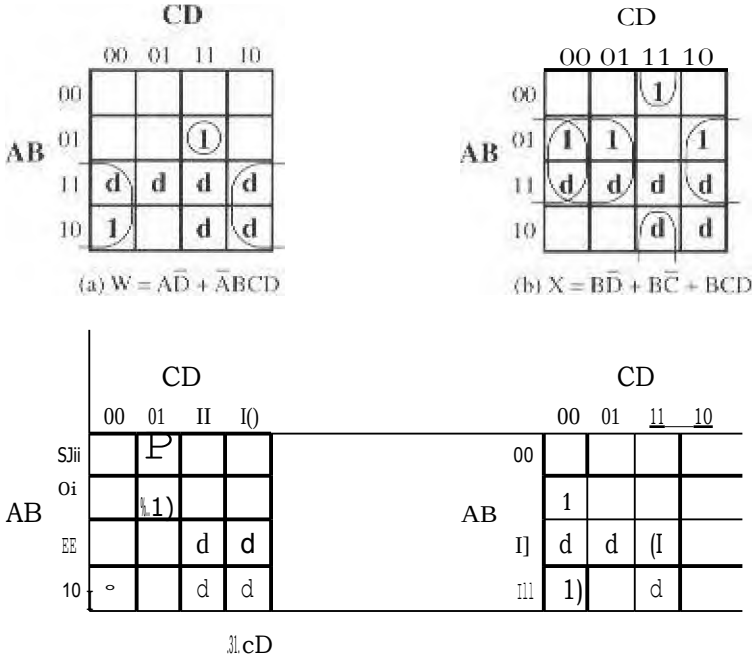


Figure A.10 kArnaiigli Maps lor hicrimorokIr

The next stcy is to find VIII p.Nir.s, of terms That differ in 1,nly one variable. that hi, an pairs of terms that are the same except that one '.21riable is 0 in one of the terms and 1 in the other. Because of the way in which we have grouped the terms, we can do this by starling with the first group and comparing each term of the first group with *every* turn or the soeond *goup..111V1* compare each perm it Lh,,ecoriti group with all of the terms of the third group, and so on. Whenever a match is found,

Table A.5 First Stage of Quine••McKluskey Method
(for AR fir) I ARO) - All(:13 AI3CD AHCI) Alici) I A BCH)

Product Tern]	Index	A				
A BCD	I	(1	0	1]	I	✓
A BC D	5	0	1	n	I	V
iii3C15	6	(1	I	I	(1	V
ABC'I)	12	]	1.	0	II	V
A BcD	7	0	I	]	I	✓
A13CD	II	1	0	1	1	V
ABCD	13	1	1	0	i	V
ABCD	15	I	1	i	I	V

Table A.(Last Stage of Quinn.—MeKluske7i Method
for F = 4 ARC]) 4 AlWD 4..Aij.CD i..I=3CD i. idscr3 - A.Bn) - Afif'D)

	. ABCD	ABCD	ABCD	AECD	Ai.BCD.	ALBC17.)	. TkEPTD	AE CD
AD	X	X			X		X	
ACD							X	
ABC					E	O		
AFJ(		Fill	tD.					
Acr)	Z.							

place a check next to each term, combine, the pair by eliminating Ihe variable that differs in the two turns, and add thal to a new list. 'I'huas, ror example, the terms ABCD and ABCD are combined to produce ABC. This process continues until the entire original table has been examined. The result is a new table with the following entries:

ACD	ABC	AB D
	HCI)	ACD
	ABC'	BCD
	AB D	

The new table is Organized into groups, as indicated, in the same fit.s.hion as th e fiat lahlc. The second table is then processed in the same manner as the first. That is, terms that differ in only one variable are checked and a new term produced for a third table. In this example. the third table that is produced contains only one term BD.

In general, the process would proceed through successive tallies until a table with no matches was produced. In this case. this ha;linvolved ihree tables.

Once the process just described is completed, we have eliminated many of the possible terms of the expression. Those terms that have not been eliminated are used lo mrist ⊔et a ma trix, as illustralcd in Fable A.& Each row of 1 he matrix corresponds 10 one of the terms that has not been eliminated (has no check) in any of the tables used so far. Each column corresponds to one of the terms in the original expression. An X is placed at each intersection of a row and a column such that the row element is "compatible ¯ with the column c.lcrocnt. vxri411,Ics present in the row clement have the same value as the variables present in the column element. Next,. circle each X that is alone in a column. Then place a square around each X in any row in which there is a circled X. If every column now has either a squared or a circled X, then we are done, and those row elements whose Xs have been marked constitute the minimal expression. Thus. in our example, the final expression is

ABC -F ACD + ABC + ACD

In cases in which some columns have neither a circle nor a square. Eddilional processing is required. Essentially, we keep adding row elements unlit all columns are covered,

Let us summarize the Cluine—McKluskey method to try lo justify intuitively why it works, 'fhe first phase of the operation is reasonably sir4lightforward. proCc c[imimile unneeded vitrivAb[es in product lcrnis. Thus., the expression ABC + ABC:is equivalent to AB. because

$$ABC: + ABC — \quad -F = AB$$

Alter l he elimination of variablcs, we arcieft with an l2 Xiii csMon that is clearly equivalent to the original OxriteSSiOn. I JOV02.VCr, there may he redundant terms in this expression, lust as we found redundant groupins in Karnaugh maps. The matrix layout assures that each term in the original expression iw covcre41 and does s„0 in wily 1hal mininliZe 1[1Q number of terms in the final QX1)1"Cl..S1011.

NAND and NOR Implementations

AnoE her considEni Lion in hu implemenialion Boolean func[ions, concerns the types of gates used It is on en desirable to implement a Boolean function sole]y with NAND gates or solely with NOR gates. Although this may not be the mini-rnum-ga Le implementation, it has the advantage of regularily, which can impIify the manufacturing process. Consider again 1 2quotion (A.3):

$$F = B(A — C)$$

Because the complement of the complement of a value is just the original value,

$$F = B(A\ C) = (AB) + (BC)$$

Applying DeMorgan's iheorern,

$$\overline{(AB) - (BC)}.$$

which has three NAND forms, as illustrated in Figure A.11,

Multiplexers.

The multiplexer connects multiple inputs to a single outpw. Al any time, one of the inputs is selected to be passed lo the output_ A general Mork diagram rqreSCrita-

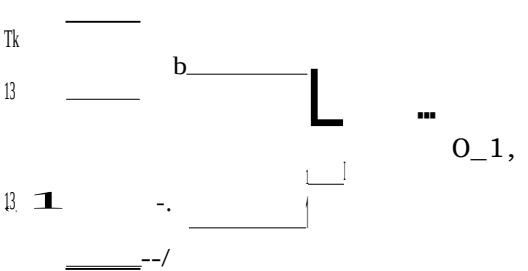
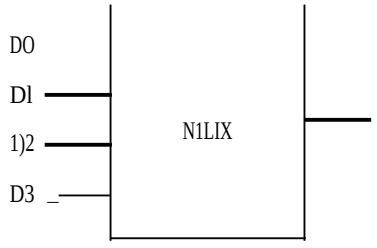


Figure A.11 NAND linplcinentation of Tablc. A.3



52 el

Figure A.12 vtultiploxer Representation

lion is shown in Figure A.12. This represents a 4-b0-1 mulEiplexer. '[here are four input lines, labeled DO, D1, D2, and DI One of these Lines is selected to provide the output signal F. To select one of the four possible inputs, a 2-bit selection code is needed, and this is implemented as two select lines labeled Si and S2.

An example 4-Lo-] multiplexer is defined by the truth table in 'Palle A.7. This is a simplified form of a truth table. lristekid of showing all possible combinations of input variables. it shows the output as data from line DO, D1, D2, or D3. Figure A.13 shows an implementation using AND, OR, and NOT gates. Sa and S2 are connected to [ht. ANT) g;w2..s in ';ueh a way th4it, for any cornhinalion **Si and** 82, three of the AND gates will output 0. The fourth AND gate will output the value of the selected line. which is either 0 or t. Thus, three of the inputs to the OR gate are akvays 0. and the output of the OR gate will equal [he value of the selected input gate. Using this regular organiz;ition, it iS easy ") construct inuiliplexe.r.s of size 16-to-1, and so on.

Multiplexers are used in digital circuits to control signal and data routing. An example is the loading of the program counter (PC). The value to be loaded into the program counter may C]TT1,2: Irodri one of several different sources;

- A binary counter, I he PC is lo he incremented *for* the nexl. imLruction
- The instruction register, if a branch instruction using a direct address has just hecn e xecuted
- The. output of the ALL!, if the branch instruction specifics the address using a displacement mode

Taile A.7 4-1.0-1. Multiple.xt2r Truth Table.

52	SI	
0	0	DO'
0	l	DI
l.		
l	l	D3

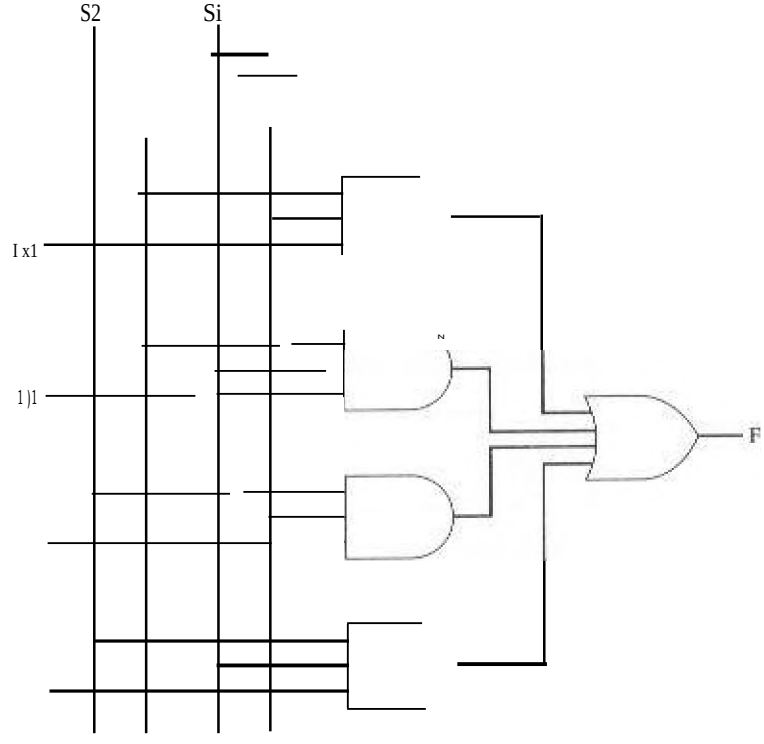


Figure A.13 MultipEcxer Intplurnontation

1'hes.e various; inputs couEd be connected to the input lines of nItiEtiplexer, with the PC." connected to the output line. The select lines. &tern Line which value is loaded into the PC. Because the PC conwins rnohiple hits, multiple multiplexers. Tc used, one per bit Figure A.14 illusimics this for 16-bit addresses.

Decoders

A clue.odur is 8 *co* mbinational circuit with a number of output lines, only one of which is asserted at any time. dependent on i Fie pattern of input lines. In general, decoder has *n* inputs and 2 outputs.. figure Al 5 shows a decoder with three inputs and eight outputs.

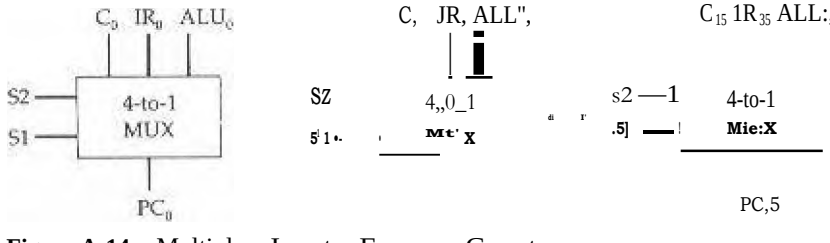


Figure A-14 Multiple= Input c Frowner Counter

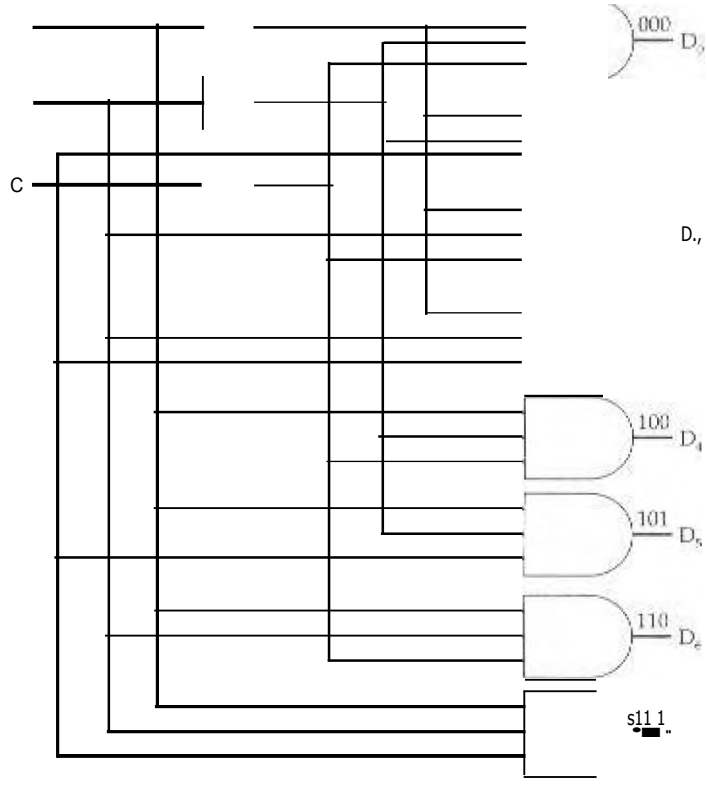


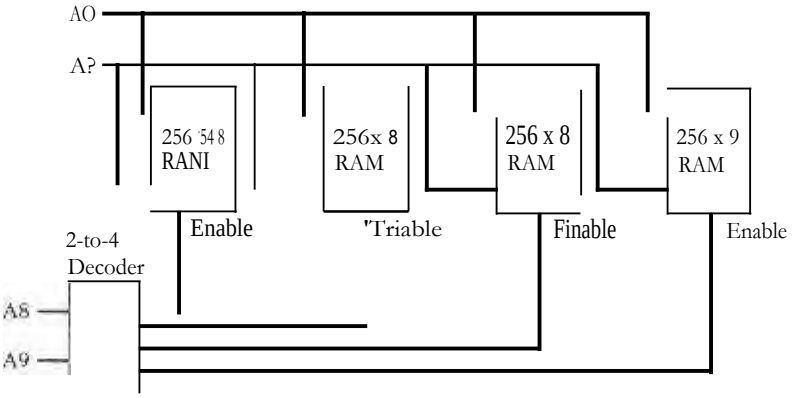
Figure A.15 Deonder with 3 Inputs and 2^3 — 8 Outputs

Decoders find many uses in digital computers. One example is address decoding. Suppose we wish to construct a I K-hyte memory using four 256 x 8—bit RAM chips, We want a single unified address space, which can be broken down as follows:

.4dt/re4ih.	Chip
0000-0017	
011_10 01FF	
020(.1-02FF	
(1300 03FF	

Each chip requires 8 address lines, and these arc supplied by the lower-order 8 bits of the address. The higher-order 2 bits of the 10-bit address arc used to select one cif the four RAM chips. For this purpose, a 2-to-4 decoder is used whose output enables one of the four chips, as shown in Figure A.16.

With an additional input line, a decoder can be used as a demultiplexcr. The &multiplexer performs the inverse function of a multiplexer, it connects a single input to one of several outputs. This is shown in Figure A.1•7. As before. n inputs are decoded to produce a single one of r outputs. All of the 2' output lines are ANDed with a data input line. Thus. the *n* inputs act as an address to select a particular output line, and the value on the data input line (0 or 11 is routed to that output line.



Figiorc. N.16 ALIcIross Decoding

The configuration in Figure A.17 can be. viewed in another way. Change the label on the new line from *Dora Input* to *Enabfe.* This allows for the conlrol of the timing of the decider. I'he decoded outpul appears only when the eneoc.kd input is present *(Ind* the enahle line has a value of 1.

Programmable Logic Array

Thus far, we have treated individual gates as building blocks, from which arbitrary functions can be realized. The designer could pursue a strategy of minimizing the number of gates k be used by manipulating the corresponding Ka olcan expressions.

As the level of integration provided by integrated circuits increases, other considerations apply. Early integrated circuits, using small-scale integration (SSI), provided from one to ten gates on a chip. Each gate is treated independently, in the hui [ding-block approach described 50 far. Figure A,]K is art LANI nrIc .onie ssi chips. To construct a Logic function, *a* numher of these chips are Laid out on a printed circuit board and the appropriate pin interconnections are made.

nereasing levels of integration made it pc ible to put more gaLeN on N chip and to make gate interconnections on the chip as u ell. 'Ails yields the advantages of

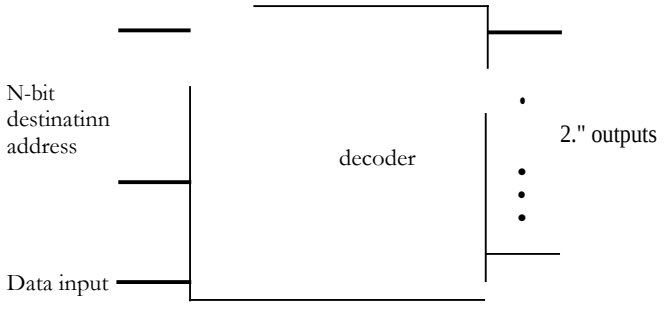


Figure A.17 Implementation 1.4 a Demultiplexer Using a DtTeckr

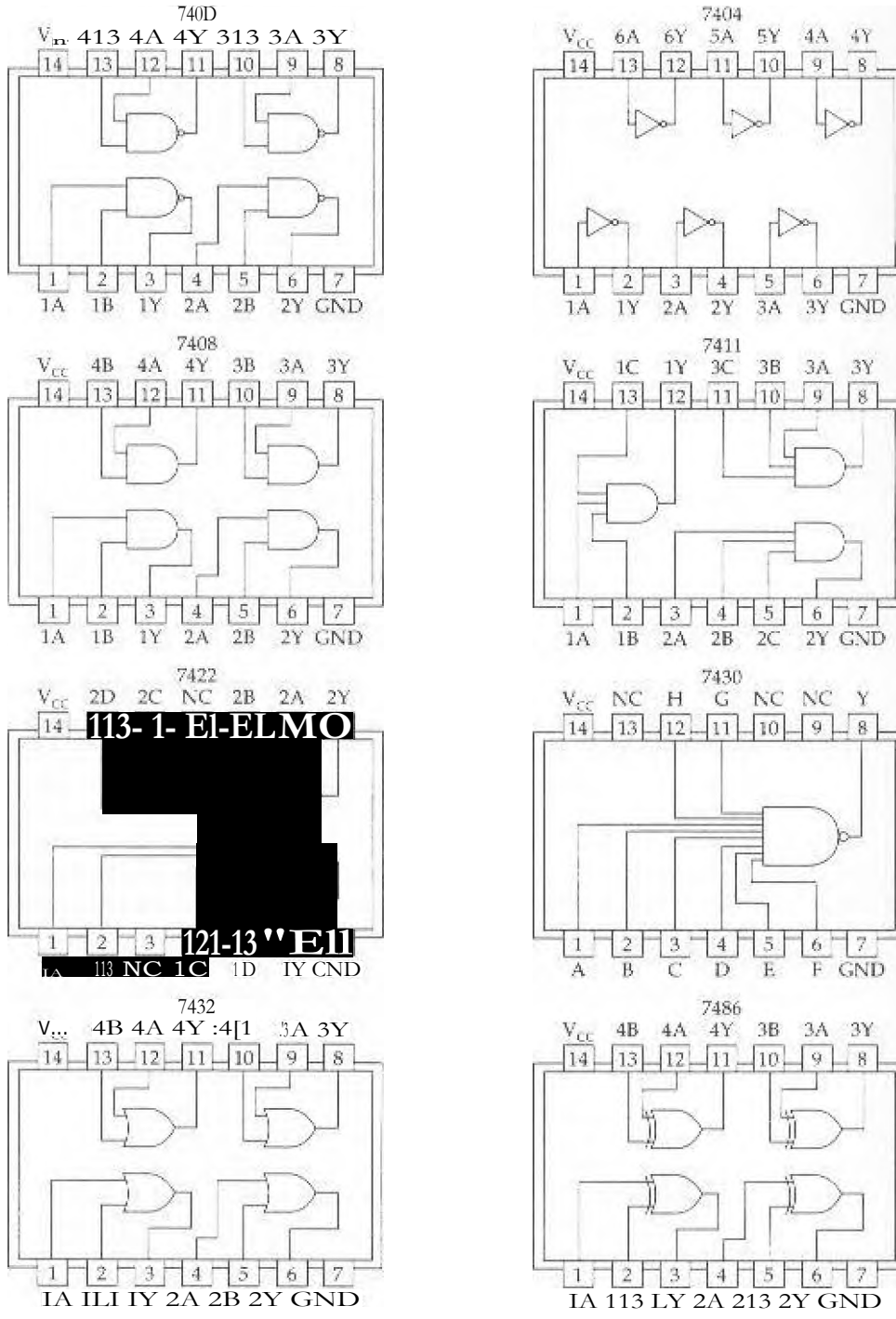


Figure A.IS Some SST Chips. Pin layouts from *The TT1. Dam Bonk for Design Etigiourers, copyright* 0 1975 Tcrias In:strum:v. Incorporated.

decreased cost. decreased size, and increased speed (because on-chip delays arc. (.4 shorter duration than off-chip delays). A design problem arises. however. For each particular logic function or set of functions, the layout of gates and interconnections on 1 he chip mu st he designed. 'rho east and Liffic involved in such custom chip design is high. Thus, it becomes attractive to develop a genera]-purpose chip Ihail can be readily adapted to specific purposes. This is I hc intent of the *pri3Krammethic logic array (1³ 1,A).*

The PLA is based on the fact that any Boolean function (truth table) can be expressed in a sum-of-products (SOP) form, as we have seen. ' l'he PLA consists of a regular arrangement (.4 NOT, AND, and OR gate. tin a chip. Each chip input is pasNed through a NOT gate so that each input and its complement are available to each AND gate. The output of each AND gate is available to each OR gate. and the output of each OR gate is a chip output- By making the appropriate connections, irilitrary SOP expressions can he implemented.

Figure A.1')a shows a PLA with three inputs, eight gates. and two outputs. Most larger PLAs contain several hundred gates.. I S to 25 inputs, and 5 to 1.fi outputs, 'i'he c(mneetions from the inputs to the AND gates, and from the AND gates to the OR gates, are not specified.

PLAs are manufactured in two different ways to allow easy programming (making of connections). In the first. every possible connection is made through a ruse at every intersection point. The undesired connections can then be later removed by blowing the fuses. This type of PLA is referred to as a *fieid-pmgranunable logic array.* Alternatively, the proper connections can he made during chip fabrication by using an appropriate mask supplied for a particular interconnection pattern. in either case, the PLA provides a flexible, inexpensive way of implementing digital logic functions.

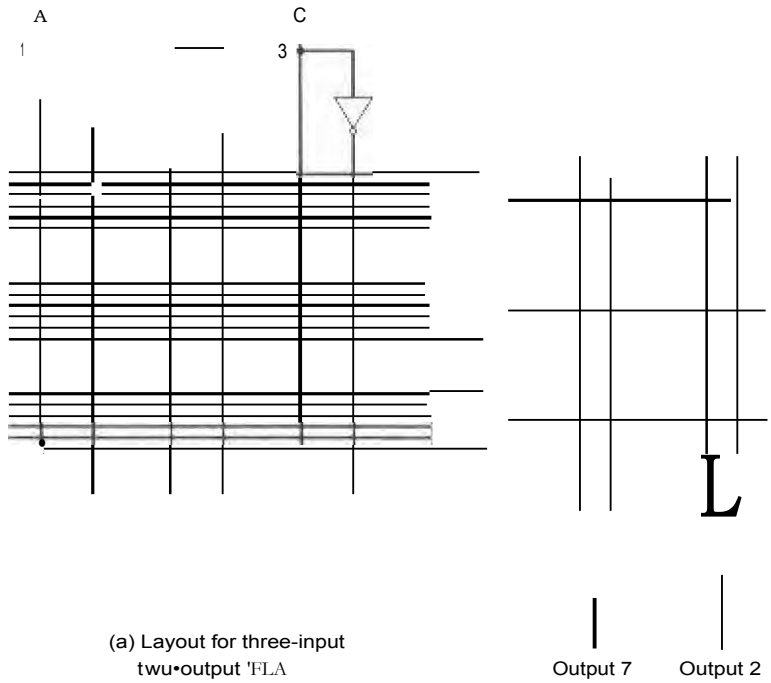
Figure A. 19b shows a design that realizes two Boolean expressions.

Read_Only Memory

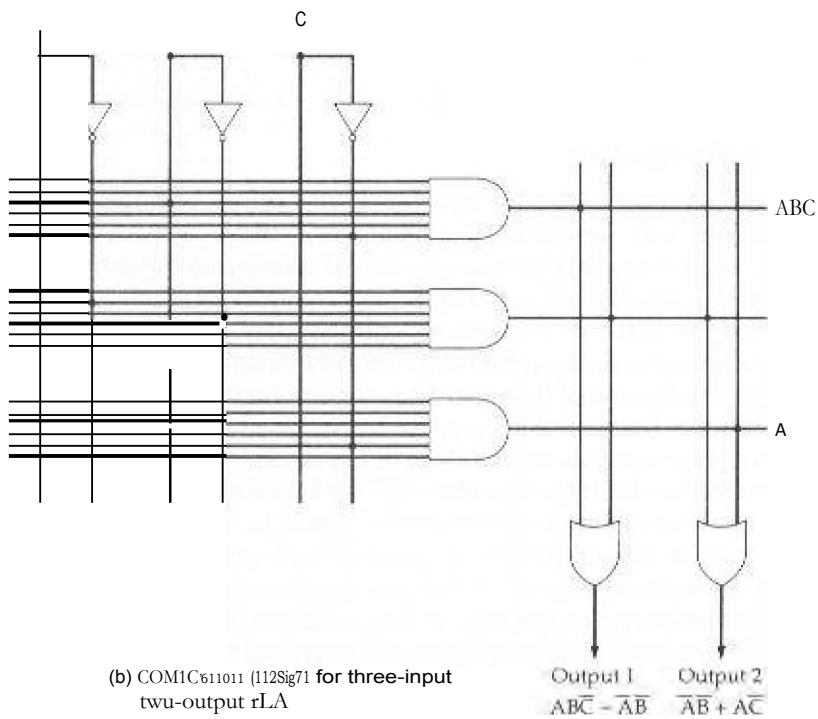
Combinational circuits are often referred to as 'memoryless".ei rcui1s, because their output depends only on I heir current input and no history of prior inputs is retained. Ihowever, there is one sort of memory that is implemented with combinational circuits. namely *read_only mearvrk*• (ROM).

Recall that a ROM is a memory unit that performs only the read Operation. This implies that the hinary information stored in a ROM is permanent and was cre• ated during the fabrication process. Thus, a given input to the ROM (address li nes) always produces the same output data lines). Because the outputs are a function only of the present inputs. the HON1 is in fact a combinational circuit.

A ROM can he implemented with a decoder and a set of OR gates. As an example, consider Table. A,8. This can be viewed as a truth table with four inputs and four outputs. For each of the 16 possible input values, the corresponding set of values of the outputs is shown. lt can also be viewed as defining the contents of a 64-bit ROM consisting of let words of 4 bits each. The four inputs sped ᶠy an address, and the four outputs specify the- contents of the location 2,pCeified by the address. Figure A.2[1 shows how this memory could be implemented using a 4-to- L6 decoder and four OR gates. As with the PLA, a regular organization is used, and the interconnections are made to reflect the desired result.



(a) Layout for three-input twu•output 'FLA



(b) COM1C611011 (112Sig71 for three-input twu-output rLA

Figure .A.1.9 **Example ofd Progrurnalable Logic Array**

MAle 4.8 'I lable for ri ROM

Input				Output			
	0			0	0	0	0
0	0	**0**	1	0	(1	CF	l
(F	**0**	1	0	0	0	1	1
II	0	1	l	0	I)	l	CI
0	1	0	(1	0	l	l	0
0	1	0		0	l	1	1
	1	l	0	0	]	(1	1
0	1	l			I	(I	0
1	(1	0	0	1	I	0	0
1	11	0		1	L	0	l
1				1	l	1	l
I	0	I			L	1	0
		0		1	0	t	0
1	1	0	l			l	I
1	}	l	0				
		l			11	0	

Adders

So far, We have seen how interconnected gates erm Inc used to implement such lunetions as the routing of signals, decoding, and ROM. One ussentW are4A not yet addressed is that of aril hmetic. In this brief overview, we will content ourselves with looking at the addition function.

Binary addition differs from Boolean algebra iri lhat the result includes a carry term. ThIls.

0	lll	1	1
L	+1	+Q	-1
(1	I	1	10

However, addition can sii I I be dealt with in Boolean terms. In Table A,9a, we show the logic for adding two **input hill** to produce a 1-hit sum and a carry hit. This truth table could easily be implemented in digiial logic. However, we are not interested in performing 4iddiiion on just a single pair of hits, Radii:F', we wish to add two tr-bit numbers. This can be tituie by putting together _a set of adders so that the carry front one ;Elder is provided as input to the next- A 4-bit adder is depicted in Figure A.21.

For a muhiple-hi adder to work, each of the single-bit adders muss linve ihrce inputs, including the carry f **min** the next-lower-order adder. The revised truth table appear in 'Fable A.9b, The. two outpuis can he expressed=

$$\text{Sum} = ABC\ ABC + ABC + ABC'$$

$$\text{Cirry} = AB - AC + BC$$

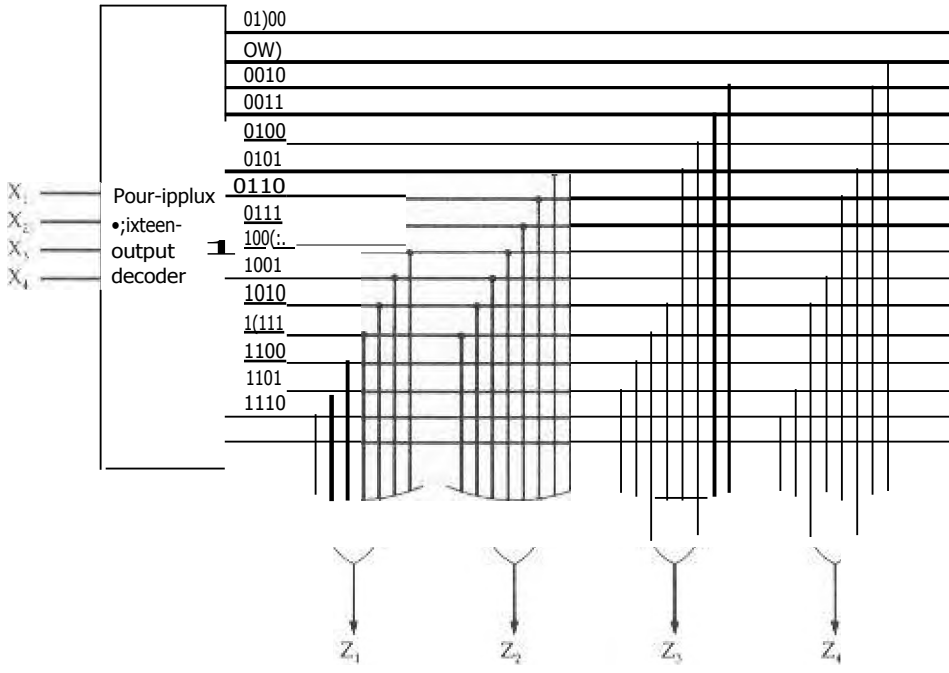


Figure A.20 A 64-Bit ROM

Pigui e. A.22 ix ;i n i mplcmcithilion wing **AN1), OR** 4iTLCI Ncyr gates.

Thins we have the neces pi logic to impiement a multiple-bit adder such s shown in Figure A.23. Note [hat because the output from each adder dc:perids the carry from the. previous adder, there is an increasing delay from thc leasi signir-ieant 10 the mod significant Each lriyle hit. .1cicii,;,. experiences a certain amount

THbit Biiairy Addititin 'Fruit) l'aiblc

.in) Singly-13it Addition				Mt Addition with Carry Input				
A	10.	Sinn	Carry	C|		B	Sum	C,,,,,
0	0	0	0	0	0	0	0	0
(1	1	1	0	0	0	1	I	0
l	0	1	0	0	1	0	1	0
1	1	0	1	0	1	1	0	1
				1	0	0	1	0
				1	0	1	0	1
				1	1	0	0	1
				1	1	1	I	1

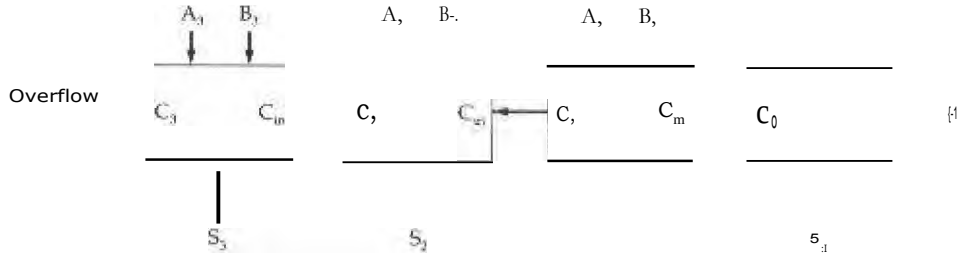


Figure A.21 4-11it Athlor

of gate delay, and ihil.; gate delay accumulates. For larger adders, the accumulated delay can become unacceptably high.

lf the carry values could be tletermincd without having to ripple through all the previous suige:'1, !hen each single-bit adder could function indupencluntry, and do lay would not Li Ccuin atcQ, ThiN can be achieved with an approaal known as *cam. lothicalreeodr* Let us look again at the 4-bit adder lo explain this approach.

We would like to come up with an cvi.c.sion thal P;pecifies the carry input to an!,' stage of the adder without reference to previous carry %lilies, We have

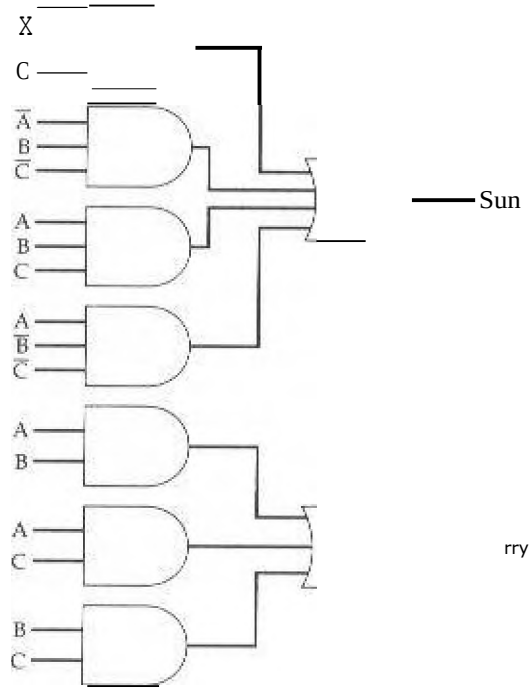


Figure A.22 I**mplQinentatiois.of** an Adder

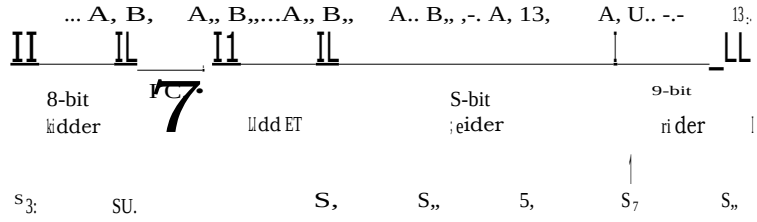


Figure 4.23 Construction of a 32-Bit Adder Using 8-Bit Adders

$$C_{8} + \qquad + \qquad (A.4)$$

$$(AS)$$

Folttwing procedure, we gut

C. = A,13,, I A ;.A.B, .1- .A,,A,A,,B ;) -L A;3 A,,B,, + B,A,B, -I- B,A,A, B,, - A,,B,,

This process can be repeated for arhiirarily long adder! '.. Each carry term can he expressed in SOP form as a function on]} of the original inputs. with no dependence on the carries, Thus, only two levels of gate delay occur regardless of the iength of thc ad&r.

For long numbers. this approach becomes excessively complicated . Evaluating the expression for the most significant bit of an to-hit adder requires an OR gate with — 1 inputs and it ANI7 gatcs with from 2 lo n - 1 inputs. Accordingly, full carry lookahead is typically done only 4 to bits at a time. Figure. A.2 3 shows how a 32-bit adder can be constructed out of four s-bit adders. In this case, the carry must ripple ihrough the four K-bil ;Rider& bui this will be substantially quick er than a ripple through thirty-two 1-bit adders,

A.4 SEQUENTIAL CIRCUITS

circuits implement the Lnisential functions of a digital computer. However. except for the special case of ROM, they provide no montory or stake inforrruition. elements also essential to the operation of a digital compul cr. For the latter purposes. a more oDmplex form of digital 'ugly. C.TCLE i used: the seilueninil circuit. The current output of a sequential circuit depends not only on the current input, but also on the past history of inputs. Another and generally more useful way to vicw it k that the current output of a sequential circuit depends on 11w currcn I input and the current 4I- t{ or That circuit-

1n this section, we exainine some simple but useful examples of sequential en - euiV., As will be .wen, the sequential circuit makes use of combinalional circuits.

Flip-Hops

The s.impiest form ,,r tic lU llli44l eircail is the 'Dm:. are a variety of flip-flops, all of which share two properties:

T The flip-flop is a !listable deviee- IL exists in one of two states and, in the absence of input, renmins in that state. Thus. the flip-flop can function as a I-hit memory.

The nip-rit)r) has two outputs, which are always the complements Of each other. These are generally labeled 0 and 0,

The S–R Latch

Figure A.24 shows a common coil figuration known as the S—R flip-flop or S—R latch, The circuit has two inputs, S (Sel) and R (Reset). and Iwo outputs. C> and 0, and consists of two NOR gales hooked together in ;1 feedback arrar4?,ement,

First, let us show that the circuit is bistable, Assume that both S and R are and l hat Q is tr The inputs to the lower NOR gate are Q — 0 and S — 0. Thus, the outpui Q — I mean , tha1 the inputs Io the upper NOR gate are = 1 and R = 0, which has the output Q = 0. Thus , the state of the circuit is internally consistent and remains stable. as long as S — R — O. A similar line of reasoning shows that the state 0 = 1, C = is **also** h)r R = S = 0,

Thus, this circuit can function as ;1 I -bit memory. We can view I he output O as the 'value" of the bit. The. inputs S and R serve to write the v alues 1 and 0, respeclively, into memory. To see this, consider the state O — O. 0— 1. S — — 0. Suppose that S changes lc the value I. Now the inputs to the lower NOIt gale are S = O. After some time delay *at,* the output (}1' the lower NOR gate will he Q = 0 {see Figure A.25). So, at this point in time, the inputs to the upper NOR gate become R = 0, = O. After another gate delay of At. the output. becomes I, This is again a stable state, The inputs. to the lower gate arc now S — 1, Q = 1,1Nhich maintain the output Q — O. As long as S = 1 and R = 0, the outputs will remain O = 1, Q = 0, kiriherrnorc, S returns to O. the outputs will remain unchanged,

The R output performs the opposite function. When I goes lo 1, it rorces = 0, 1 regardless of the previous state of Q and Q . Again, a time delav of *2As* occurs before stability is re-established (Figure A.25).

The S. **R** latch can be defined with a table similar to a truth table, called a char-*acteristic iethie,* which shows thc next state or states of **Li SCLILLCIliial** circuit as Ill rune.-tion of current states and inputs. In the case of the S—R latch, the state can *be* defined by the value of 0. Table A.lOa shows the restrlling ch41raeteristic iable. Observe that the inputs S **1 . R —** 1 are not allowed, 11.C.C.NLLSe **the.e %VOL]** Id produce an inconsistent output (both 0 and p equal 0), The table C@11 be expressed more

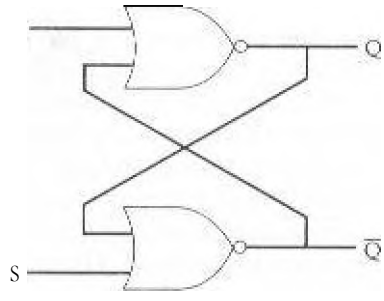
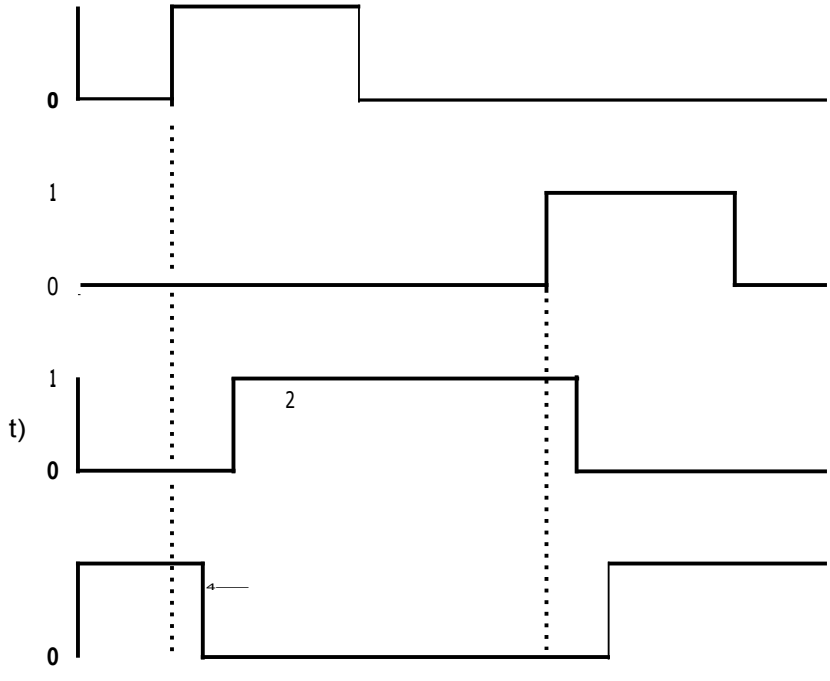


Figure A.24 Thu. *S* R Latch linplemenn...(1. with NOR Gates



Fignre A.25 NOR S-R Latch tinting Diiigain

compactly_ as in 1 able .A.1.011. An illuz'lriiiion 01 the behiivior or the S....R latch is shown in Table A.1 Oc.

Clocked S_R

The output of the **S_R latch** changes, Lifter a brief time delay. in response to **a change in the input. This is** referred to as asynchronous operation. :Vlore typically, events **in** the digilal computer are synchronized to zM clock pulse, so chat changes occur only when a clock pulse occurs. Figure A.2624rows **this** arrangement. This device is referred to **as** a *clocked S_R flip_flop.* Note **that** the R and S inputs are passuil 10 [hi NOR gulcs **only** during the **clock** pulse.

D

Onu problem with S-R flip-flop is that the condition R - S = I must be avoided. One way to do this is to allow jub,t u ngic input. The. D flip-flop accomplishes this. Figure A,27 shows a gate implinnentation and the characteristic table *of 1ht* D [lip-Flop. **By** using an inverter. the nonclock inputs to the two AND gates MC' guaranteed [o oppo,silc of each, other.

The D flip-flop is sometimes referred to as the **data** flip-flop because it is, in cliect, storage for one bit of data. The output of the D flip-flop is always equal to the most recent value to 1111,i input, Hence. it Tiftnemberii and produces the last input. It is also referred to as the delay flip-flop. •ccausc it delays a 0 or I applied to its input for a single clock pulse.

Table A.10 The S—R Latch

f a) Characteristic Table **(10 Simplified** Uhanicteristie **Table**

Current Inputs	Current State	Next State
SR	**()„**	**Om-1**
00	0	0
00	1	1
01	0	0
01	1	0
10	0	]
10	I	1
1 1	0	
11	I	

S	K	On - I
0	**0**	Q.::
0	1	0
1	0	1
1	1	

(c) Response to Series of Inputs

1	ii	1	2	3	4	S	6	7	g	9
S	1	0	0	0	0	0	0	0	1	0
R	**0**	**0**	**0**	1	**0**	**0**	1	**0**	**0**	**0**
Q,. ,	]	I	1	**0**	**0**	0	0	0	1	' 1

.ll—K Ilip-Flnp

Another useful flip-flop is the J—K flip-flop. Like the S—R it has two inputs. However, in this case all possible combinations of input values are valid. Figure A-2X shows a gate implementation of the I .K flip-flop, and Figure A.29 shows its characteristic table (along with those for the S—R and D flip-flops). Note that the first three combinations are the same as for the S—R flip-flop. With no input, the output is stable, The J input alone performs a set function, causing the output

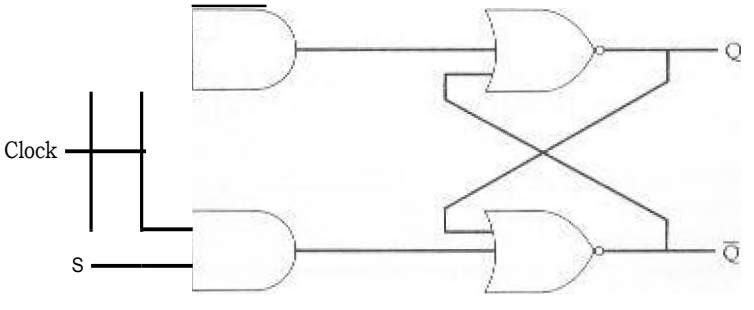


Figure A.24 Clocked S—R Flip-Hop

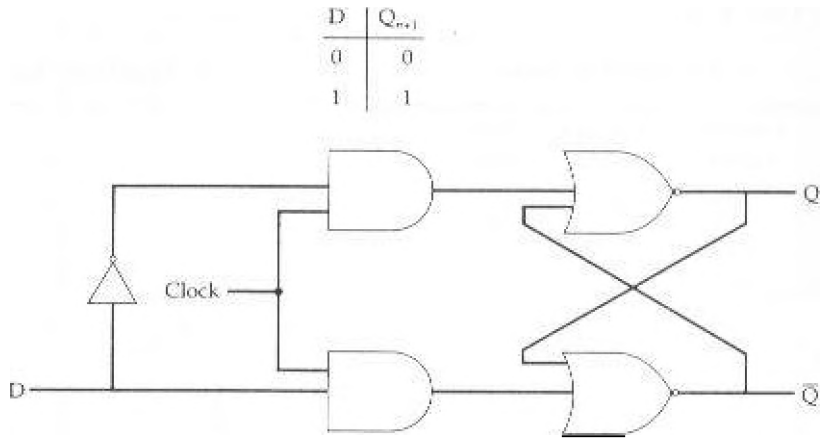
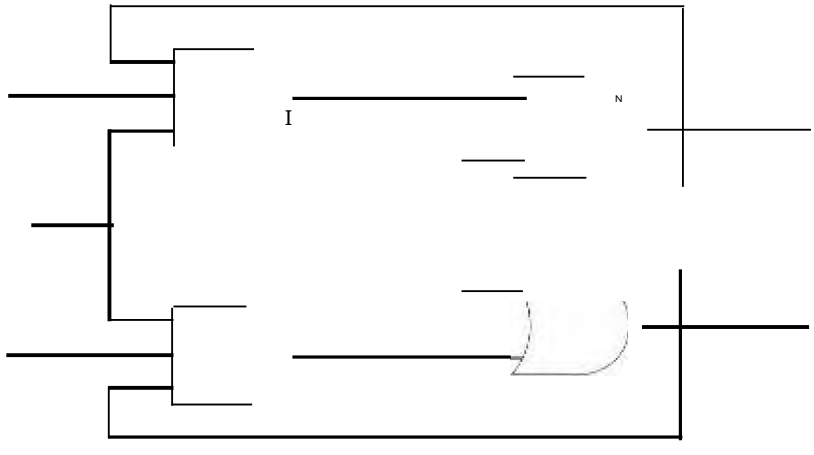


Figure A-27 D

to tw. E; the K input alone performs a reset function. causing the output to be 0. When both .1 and K 1. lbc function performed is reicri cd to as the log & function: the output is reversed. Thus, if Q i 1 and 1 is applied to .1 rind K. then Q becomes O. The reader should verify that the impiementation of Figure A.28 produces Ibis. characteristic function,

Registers

As 21n exarnpie of the usc of nip- Nops, let us first examine' orR,I. or the csscnthrI elcments of the CPU! the register. As we kmrw. regiKlcr ix,i circuit used within



w9 Pure A,2.8 3-.K I lip-Flop

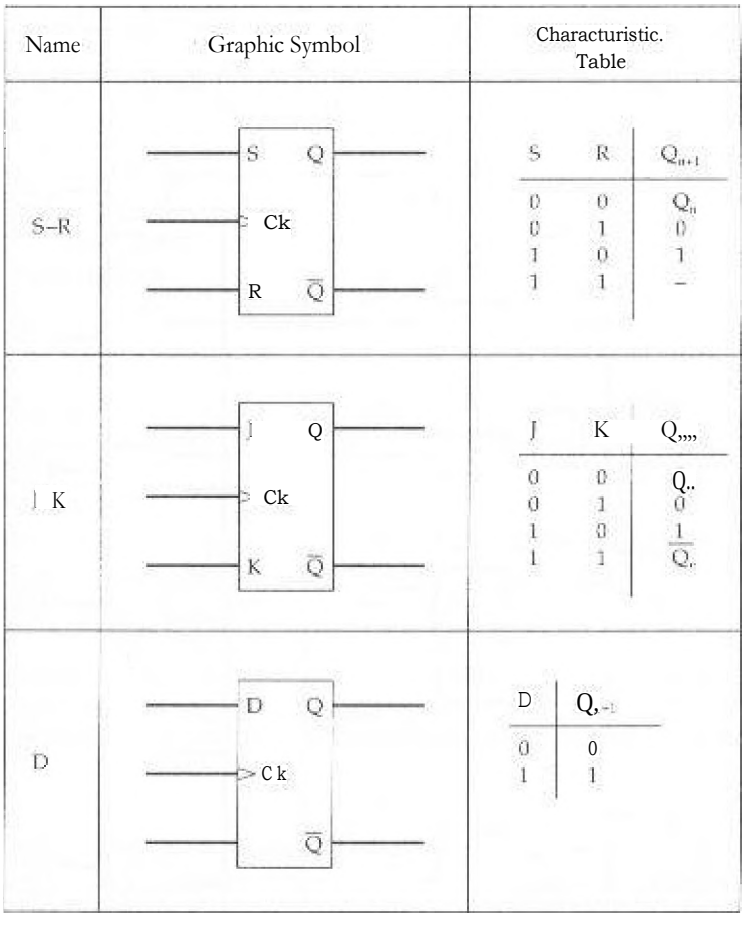


Figure A.29 Basic Rip-Flops

the (71-[1,i lo siorc one or mere hie tiaW. Two bask types of registers are.commonly used parallel registers and shift registers.

Parallel Registers

A parullel eonsists of a set of l-**hit** memories that can be read or written simultaneously, It is used to store data. The registers that we have discusscd throughout this hook are parallel registers.

.1'he 8• register of Figure A30 illustrates the operalion of a parallel register. S—R latches are used. A control signal, labeled *input dale strobe.* contr(6 writing into the register from signal lines. D1 1 through D1.8. 'Ihwse lines might be the output of multiplexers, so [hat data from a variety of sources can be loaded into the register. Output is controlled in a similar fashion. As an extra feature, a ree1 line is

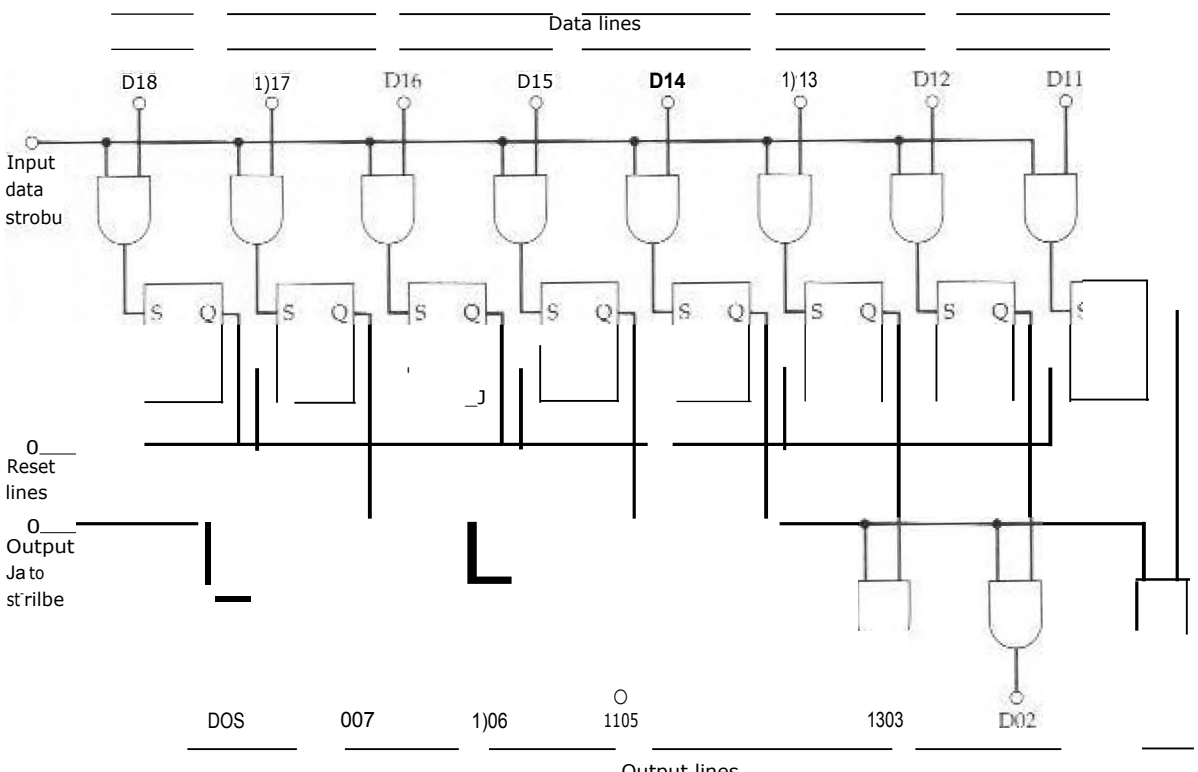


Figure .4.30 &Bit Para I fel Register

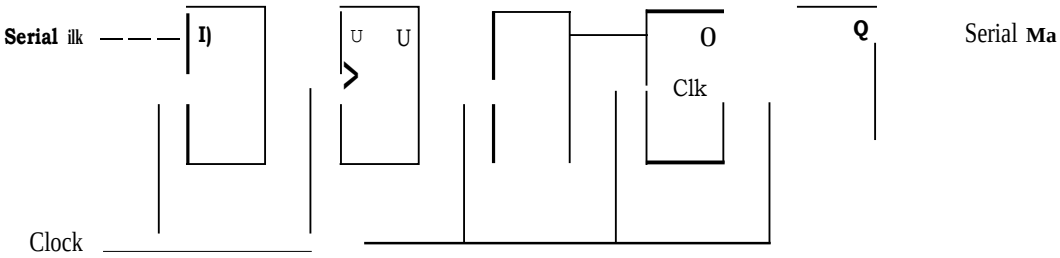


Figure A,31 Shill Register

available that allows the register to be easily set to 0. Note Thal ibis could not be accomplished as easily with a register constructed from D flip |lops-

Shift Register

A shift register accepts and/or transfers information serially. Consider, for example, Figure A,31, which shows a 5-hit shift register constructed from clocked ID flip-flops. Data are input only io the leftmost flip-flop. With each clock pulse, data arts shifted I the right one position, and the rightmost hil is Lrunsierred out.

Shift registers can be used to interface to serial I/O devices; In addition, they can be used within the ALU to perform logical shift and rotate functions. In this latter capacity, they need to be equipped with parallel read/write circuitry as well :is

Counters

Another useful category of sequential circuit is the counter. A counter is a register whose value is easily incremented by 1 modulo the.capaeity of the register. Thus, a register made up of *n* flip-flops can count up to .2'4 When the counter is incremented beyond its maximum value, ll IS **set to** 0. An example of a counter in the CPU is the program **counter.**

Counters can be designated as asynchronous or synchronous, depending on the way in which they operate. Asynehrc.mous counters are relatively slow because *the* output of one flip-flop triggers a change in the status of the next flip-flop. In a synchronous counter, a]] of the flip-flops change state at the same time Because the latter type is much faster, it is the kind used in However. it is useful to begin the discussion with a description of an asynchronous counter.

Ripple Counter

An asynchronous counter is also recon'ed to as a ripple counter. because the change that occurs to increment the counter starts at one end and "ripples" through to the other end. Figure. A.32 shows an implementation of a 4-bit counter using .1-1‹.. flip-flops. together with ai timing diagram that illustrates its behavior. '11w timing diagram is idealii.ed in that it does not show the propagation delay I hat occurs as the signals tnovc down the series of flip-flops, The output of the leftmost flip-flop (1.4) is the least significant hit. The design could clearly be extended to an arbitrary number of bits by cascading more flip-flops.

In the illustrated implementation, the counter is incremented with each dock pulse. Thc J and K inputs to each flip-flop are held at a constant L. This means that, when there is a clock **FNE., the ciutpul al** Q will be inverted {] 10 0; 0 lo L. Note that the change in state is shown as occurring with the edgc of the clod pulse.: **this is known** as an edge-triggered flip-tlop. Using flip-flops that respond to the transition in a clock put c nil her than the pulse itsetf provides better timing control in complex circuits. If one looks at patterns of cyuipiii for this counter, it can he seen that it cyctes through 0000, 0001 111.0, 111 L. 0000, **and so on.**

Synchronous Counters

The ripple counter has the disadvantage of the delay involved in changing value, which is. proportional to the length of the counter. To overcome this disadvantage., CPUs make use of synchronous couniers, in which all of the flip-flops of the counter change at the same time. In this subsection, we present a design for a 3-bit synchronous **coun** ter, In doing so, we. illustrate some basic concepts in the design of a synchronous circuit.

For a 3-bit counter, three flip-flops will Inc needed. 1,c1 us use J-K.
Label the uncomplemented output of the three flip-flops A, B, C, respectively, with C representing the least significant hit. The first step is to construct a truth tabte

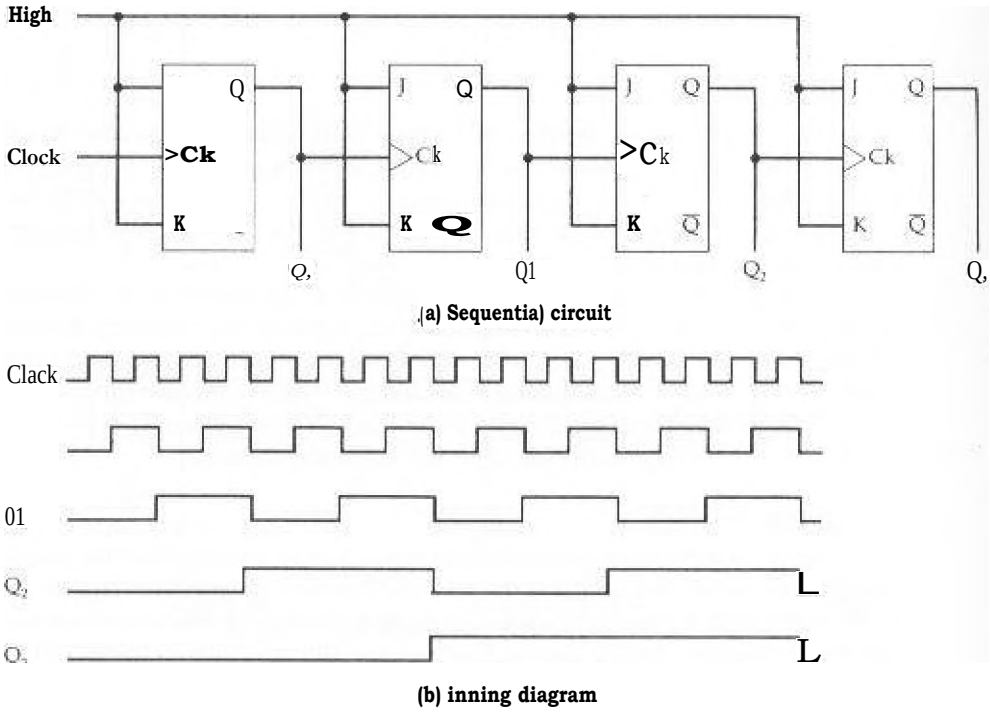
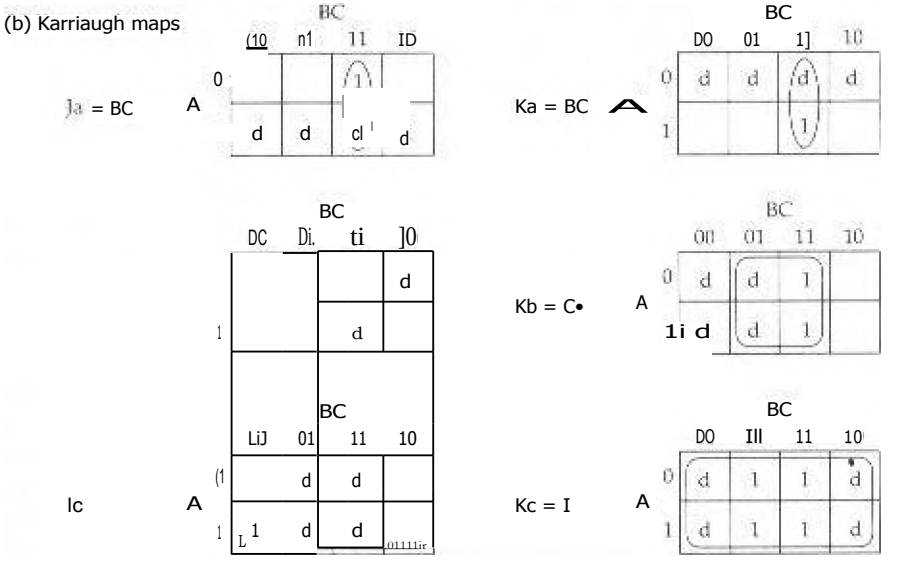


Figure A.32 **Ripple Gaunter**

(a) Truth table

A	B	C	3a	.Ka		Kb	lc	1(..c
0	0	0	0	d	0	LI	I	
0	I)	1	0	d	1	d	d	I
41	1	0	II	dd		0	1	Li
Ij	1	1	1	dd1dt				
I	0	0	d	0	0	d		d
I	0	1	d	D	I	d		dl
1	1	0	d	D	d	0	1	Li
1	1	1	d1d1d1!					

(b) Karriaugh maps



(c) Logic diagram

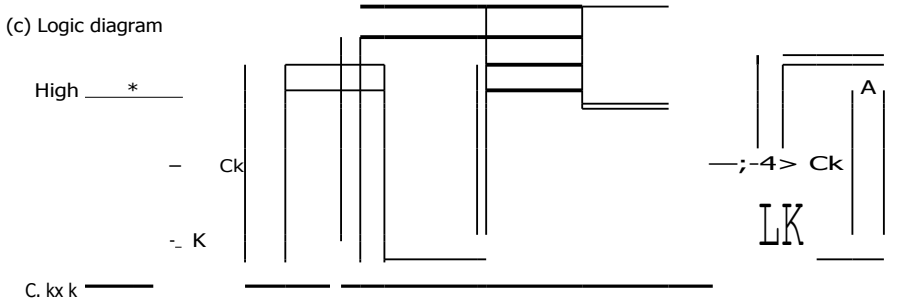


Figure A..33 Design or a Synchronous Cnunter

that relates the J—K inputs and outputs, to allow us to design the overall circuit. Such a Irah table i shown. in Figure A.33H. 'file first three columns show the possible combinations of outputs A. B, and *C.* They are listed in the order that they will appear as the counter is incremented. Each row lims ihe current value of A , B, C and the inputs lo the three nip-Lops thm will be required to reach the next value of A, B.C.

To understand the way in which the truth table of Figure A.33a is constructed, it may be helpful to recast the characterisric table for the *i-K* flip-flop. Recall that this table was presented as follows:

J	Q_{t+1}
0	Q_t
1	$\overline{Q}_t$

In this form, the table. shows the effect that the J and K inputs have on the output. Now consider the following organization of the sante information:

Q_t	J	K	Q_{t+1}
	0		
	1		

In this form, the table provides the value of ihe next output when the inputs and the present output arc known. This is exactl!,' the information needed to design the counter or, indeed, any sequential circuit. In this form, the table is referred to as an excitation table.

Lei us return to Figure A. 3m- Consider the First row. We wani the value of A to remain Il. I he value of B to remain 0, and the value of *C* to go from IF to 1 wi h thy: next application of a (Jock pulse, excitation table shows *th* at to maintain an output of 0, we must have inputs of 0 and don'I care for K, To effect a transition from IJ to 1, lie inputs must be J = I and K = d. These values are shown in the first row of the table. By similar reasoning, the remainder of the table can he filled in.

Having constructed I he truth table of Figure A.33a, we see that the table shows the required values of all of the J and K inputs as functions of the current values of A, B, and C. With the aid of Karnaugh imps, We con Eirvulop Roolc:in expressions for thef..e si74 functions, This is shown in part h of the figure. For example, the Karnaugh map for the variable Ja Ohc. J input to the flip-flop that produces the A ouipui) yields the expression -la - 13C. When all six expressions are derived. it is a straightforward limiter to design the actual circuit, as shown in part c of the figure.

A.5 PROBLEMS

A, I (.4,1e1 aI.ruth table for th.c. following Boolean expressions:
- a. ABC: .1 I.SC.: _ c. A(B' -Ff3C)
- b. ABC .F · Aisc d. (A .1- 11)(A .F (.) (1-4. –

A.2 Simplify the fki11(iwing exprenioris according ihc commutative. law:
- 41. A-B— B•A I C•D.E • C•D•1 + h. D
- h. A-I3 +A•C— B•A
- V.. 4 L • NI - N) (A• B)(C • D • F.)(Nei • N • L)
- d, F - (K. R) S • V - VI/ • ∫c I V S X • %v. — (R K...) • F

A. Apply Dehelorgan's theorem to the following equations;
   a. F V - A — L
   h. F = + 13 + +

AA Simplify the following expressions:
   a. A - S•T— V•W+R•S•T
   h.A=T•t:•V.. X.Y I Y
   c. A — F • fE 1- F + G)
   d.A =(P•Q — k —S•T)T•S
   e. A=D•D•E
   ti A — Y • (W + X -F +) • Z
   g. A = (B • E — C + E') • C

4,5 Construct the operation XOR from the basic Boolean operad ions AND, OR, and NOT.

A.6 Given a NOR gate alai NOT gales, draw a logic diagram (hal will perform the three-input AND function-

4.7 Write the Boolean expression kit a four-input NAND gate,

4.8 A combinational circuit is li Sed ci 10(rol sei.rtm-segrrient display of decimal as shown in Figure A,34. The li:ix lour inputs, which provide the four-bit nude used in packed decimal repres4Ailatioti I,, - 0004) 9_1 = 1001). The seven outputs define which segment' will be. naivalc!Li to display a L,ven decimal digit- Note that some corribinaliOnS of inputs and outputs are not needed.

   Develop a truth table for this circuit.
   h. Express the truth table in SOP form.
   c. Express the truth table in PCS foun-
   d. Provide a simplified xpressii Fn.

4,9 Design an 8-to-1 multiplexer.

4.10 Add an additional line to Figure A.15 so Clot it functions as a demultiplexcr.

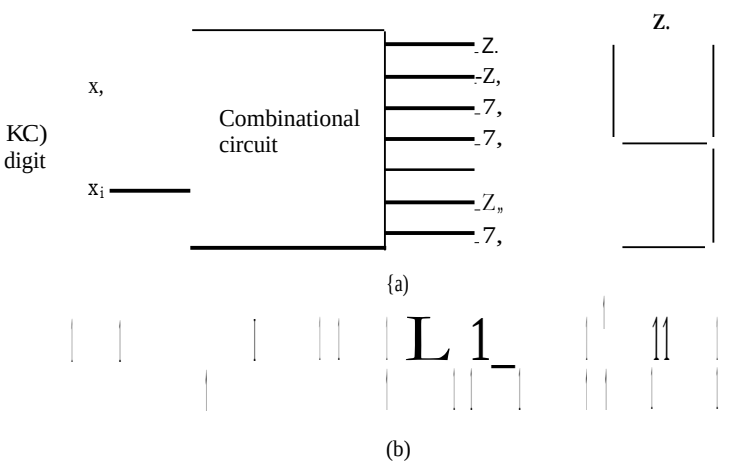


Figure 4.34 Seven Segment LED Display Example.

A.11 '1 he Gray Lxide is binayy 4:11d e' for inItz,gcni, it differs from tho ordinary binary YE:p-Tesc. ritati ori in that [li me jug a single it change between tloc re present all ions. c' any two numbers_ 'Itis is LIS42.1u1 for applkations s uch as counters or ana14-to-digital converters where a sequetiLv„ nt numbers is gunerated. Because only one bit changes at a time., th,2r'.. is never any arithiguir• ditir to 41 ight tiinirtg differt.ncs..11be. rink eight elements of the code are as follows..

Binary Code	Gray Code
01X)	000
001	001
01,;1	
011	
100	I lit
101	II t
110	IOI
111	

Design a circuit that converts from binary to Cray code,

A...12 Design a 5 32 decoder using four 3 X 8 decoders (with ,2natile inputs) and one 2 4 decoder.

A,13 lulpInient the fun adder of Figure A,22 with just five gates, (*Hint:Somc (rf* the gatcs are 01:t gates.)

A.14 Consider Figure A.22. Assume. tha each produues a delay' tri li) ns- Thus, the sum output k valid afier 30 ns and the carry output alter I) W1Du. is the rill al add ti me for a [1]..2-11it adder:

a. Implemented without carry lookohead, as in Figure *A.21?*

b. Implemented with carry lookahcad and using 8.bit adders. as in Figure A.23?

APPENDIX B

NUMBER SYSTEMS

B.1 The Decimal System

B.2 The Binary System

B.3 Converting Between Binary and Decimal

Inttgers

Fractions

B.4 Hexadecimal Notation

B.5 Problems

B.1 THE DECIMAL SYSTEM

In everyday life we use a sNy.stcm based on decin m I 4.1igils (0. 1. 2, 3. 4..5.. 6,7, H.q.) to repri,: kmt numbers and refer to the system as the decimal slistera. Consider whkit the number 83 means. It means eight tens plus three.

$$83 = (8 \times 10) - 3.$$

The number 4728 rneans lour ltkluNmids. seven hundred", [WO tens plus eight

$$4728 = (4 \times 1000) + (7 \times 100) + (2 \times 10) - F$$

The decimal system is said to have a base, or radix. of 10. This means that each digit in the number is muhiplied by 10 raised to a power corresponding to that digit's position:

$$83 = (8 \times 10') - F (3 \quad 10'')$$
$$4728 — (4 \times 10^{1}) — (7 \times 101 + (2 \times it)') — (8 \times 10'')$$

The same principle holds for decimal fractions but ne2ative powers of 10 are used. Thus, the decimal fraction 0.2515 stands for 2 tenths plus 5 hundredths plus 6 thousandths:

$$0.256 — (2 \times 10') - F (5 \times 10'') - F (6 \times 101$$

A number with both an integer and fractional part has digits raised to both positive and riegativ4 powers {**Fl 1i**}:

$$472.256 = (4 \times 10^{2}) + (7 \times WI) — (2 \times 101 — (2 \times l0 \quad (5 \times 10 \quad + (fi \times 1Cr')$$

In general, for the decimal representation of X the value of X is

$$X - \quad \times 10'$$

B. CE BINARY SYSTEM

in the decimal system, 11) different digits ;ill. used Iii represent numbers with a base of 10. In the binary system, we have only two digits. 1 and (1, Thus. numbers in the binary system are rcpresen Led iolhe base 2,

To avoid confusion, we will sometimes put suhscript nurnhcr to indicate 11;11W, For example, 83,, and 4728,,, are numbers represented in decimal notation or. more **hricrly,** decimal numbers, The digits and 0 in binary notation have the same meaning as in decimal noiaLion:

0, =

To !represent urger ri Uiribc , as with dconnal notation, each digit in a binary number has a value depending on its position:

I [1, =(l x2') + (11 2") = 2
II, = (I x 2') +{1 x = 3.,,
100_2 (1 x 2^2) + (fi x 2^1) —(@X 2^n) = 4_{th}

and so on. Again, fractional values are represented with negative powers of the radix:

1001.101 = 2 - 2 ==

In general, for the binairy rcresuniriLicin – value of Y is

Y= 'K 2'

4:3 601iVERTING :AM5 ijECTMAL

II1.5 !'imple matter t0,.onv4:1 1 a number ['EOM **binary** 11014 1.1.i on Lc) decinml notation. In fact, we showed sevciai examplef, **in the previous subsection. All that is required is to multiply each binary digit by the. appropriate power of 2 and add the results.**

To **convert from decimal to** binary, the **integer and fractional parts are handled** scparalely.

Integers

For the integer part, recall that **in binary notation, an integer represented by**

= or 1

has the value

2"' ***(b*** 2"' ²***) F . . .*** X 2') 1-

Suppose it is required to convert **a decimal** integer N into **binary form. If we divide N** by 2, in the decimal system, **and obtain a quotient N, and a remainder** *R„..* **we may write**

N- 2 x N_I I *R„* ***001.***

Next, dividt.? **1he cluotient N, by** Assume lhat **the new quolit'....nt is .Nr, and the.** neW Then

so that

$$N = 242N,. + R.) + R_5 = \quad x \quad + (R, x 2') + R,,$$

If next

$$rV,= 2N, — R_2$$

wc have

$$= (N_4 \times \quad) - (R, X 2' 1 I (R, X 2^{-}) I R,,$$

BecnLISC $N > N, > N,$ ... continuing this sequence will eventually produce a quotient N, , — 1 (except the decimal integers 0 and 1, whose binary equivalents are 0 and 1. respectivel!,..) and a remainder which is 0 or 1, Then

$$— (I \quad 2,'''' ') ' \quad X 2''' -.) 4 ,, 4. (R, x 2• — (R_i x 2') ^1 R_r$$

which is the binar!,' form of N. I fence, we convcri from I 34i e 10 Lc) base 2 by repeated divisions by 2. The remainders and the final quotient, I, give us. i rt order of increasing significance, t **he** binary digits of N'. Figure B. I shows two examples.

Fractions

For Lhc (rad ional part, recall Elia' in binary notation, a number with a value between 0 and 1 is teprescnted hy

$$0.b\ h \quad ,b \qquad\qquad b_; \quad () \text{ or } 1.$$

and ha7'1 the value

$$(b._i x 2^{-1}) -F (17 \quad x 2^{-2}) t (b \quad x 2')$$

This can he rewritLcri 4:';

$$2 \quad x (h. , + \quad ' \quad (.^{0}) ,- 2^{-1} X (b.., -h$$

This exprcmion s.uggcsts a kxhnique for conversion. Suppose we want to convert *the* number *F* (0 <: F < I) from decimal to binary notation. W know lhai *F* con be expressed in the form

$$F \quad x (h -h\ 2 \quad (/) \ -+ \ 2^{-1} x (h \ -+$$

11 we multiply Jr: hy 2, we obtain:

$$2xf=h \quad ' x(11,-F2$$

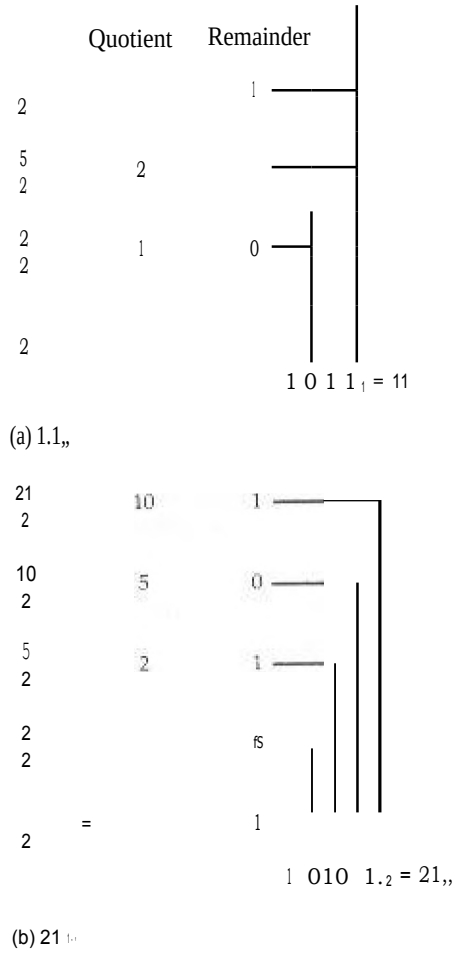


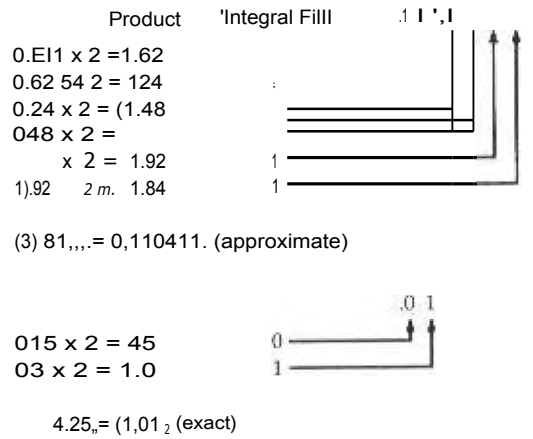
Figure B.1 Examples cif Convvrtinp, from Decimal Notation to Binary Notation For Integral Nurnhcrs

From this crd•uzition, **we see** that the integer part of (2 X *F),* which must be either 0 or 1 because C)•.:: *F < 1,* is sireplv h. „ So arc con so} (2 *F)* = + *F*|. where 0 <1 and where

$$F, _ 2 \quad (b \quad i\cdot 2 \times (b \quad _ 2^{\cdot 1} \times (1) \cdot_4 +$$

To find *b* 2. *we re1*- ,*2411. the* procesq. Therefore ; the conversion idgorithm involves repeated multiplication by 2. At each step, the frictional part of the number from the previous step is multiplied by 2. the digit to the left of the decimal point in the product will hy 0 or I and contributes to the binary representation, starting with the nicawl ',i gnificant digit. The fractional part of the product 6 tied tio ' the multiplicand in the next step. Figure B.2 shows two c.mirrIples,

This process is not rtc•ccy.:.xrily exact; that is, a decimal fraction with a finite number or digiLs may requirc binary fraction with an iofinito number of digits. In



ilgtire 13.2 E.r....uniples of Converting from Decinui1 Nototioo to Bits ily NOUitiun for Fraciiional Numbers

such *cash;,* conversion Eilgorithin is usually hafted after21prespcei net] number tit steps, depending on the desired accuracy.

13.4 HEXADECIMAL NOTATION

Because of the inherent binary nature. of digital 424)mputor components, 21 ll Corm". of data hin computers are represented by various binary codes. I I ivtvevur, no niatter how convenient the binary system is for computers, it is exceedingly cumbersome for human beings. Consequently, most computer professionals who must spend time working with the actual raw data in the Computer prefer a more compel ntitMiori.

What nolation to use? One possibility is the decimal Rotation. This is certainly more compact than binary notation, huh it is awkward because of the tediousness of converting between base 2 ar[ti

i m1e[E1.4[notation known as hexadecimal has been adoptcd. Binary digits.are grouped into sets *of* four. Hach possible combination of four binary digits is given a symbol, Eis follows;

00110 =11	11100 =
0001 = 1	11101 - 9
(11110 -= 2	1411.) A
00 =	1011 – ti
0100 = 4	1 100 – *C*
0101 - 5	1 101 --- D
0110 -	11111-=
0111 - 7	111[==F

fieQuum: I fi symbols are used, the notation is called hexadecimal, ;ind the I 6 symbols wtl: the **hexadecimal digits.**

A sequence of hexaticeim I digits can be thought of as representing ma n intcgrr in bm 16. Thus.

$$.2C._{,,} — (2_{,,} x• 16') — (C, — 1(i")$$
$$= 1(2._{D} \times 16^{|}) — (12_{10} \times 16^{|}) = 44$$

Hexadecimal rotation is used not for representing integers. It is also used as a concise notation for representing any sequence of Illi miry digits, whether they represent text, numbers, or some other type of data. The reasons for using hexadecimal notation are as follows!

1. IL is more compact than binary notation.
2. In most computers, binary data 0CE:LI py some mu l ti ple of 4 bits, and hence some multiple of a 'single hexadecimal digit.
3. It is extremely easy to convert between binary and hexadecimal.

As an CA arripic or the list point, consider the binary string L10.111.10100l. This is eq ivaient to

1101 1110 01101 — DE1 ¡„
D E.

This process is performed so naturally that an experienced programmer can mentally convert visual representations of binary ‘bta to their nexadecimA equivalent without written effort,

13.5 PROnitMg

B.1 C:cinvert the following binary numbers to [heir decimal equivalents:
▪ 001100 b. .000011 c. 011100 d. 111100 e. 101010

13.2 Convert the following binary iiiinihers to their decimal equivalents:
▪ 111.0(.A)1 I b. 11001.1.10011 c. 1010101010J

B.3 Convert the followinEt decimal numbers to their lrin my
a. 64 b. 100 c. 111 d. 14-5 e. 255

11.4 Converl the following decimal numbers. to their binry equivalents;
u. 34..75 **I.** 25,25 c, 27.1875

111. Express the following)ct I. numbers in hexadecimal notation:
11. 12 h. 5655 v. 2550276 d, 76545-336. **e.** 3726755

B.6 Convert the following hexadecimal numbers to their decimal equivalents;
h. 9F c. $D5^{0}$ d. 67E c. ABCD

B.7 C'onvert the following hexadecimal numbers to their decimal equivalents:
a- FA b. D3.E e. 1111,1 d.811,8_8 **e.** EBA,C

[1.8 Con.vc,...rt the following decimal numbers to their hexadecimal emuivalorits:
11- 16 h, 80 e. 2560 d. 3000 **e.** 62,500

R.9 Convert the following decimal numbers (c.Pthvir decini ii equivalents:
a- 204.125 b. 255,875 tr.. 631.25 d.] 00(10.00390625

11,1111 ConvE:rt the ibllowing hi:xndecinial numbers to their binary equivalents;

a, E b, IC v. A(14 d. 1F.0 e.239.1

11.11 Convert the tollowina binary ritinihels iii lheir iltieirmil ircitlivalonts:

a. 1001.1111 b. 110101.011001 v. 10100111.1111)11

ti-n Prove that every real number with a terminating binary representation (finite number of digits 10 the right Of the binary point) also has H terminating decimal representation (Finite number of digits to the right of tho cicuirnal point).

APPENDIX C

PROJECTS FOR TEACHING COMPUTER ORGANIZATION AND ARCHITECTURE

C.1 Research Projects

C.2 Simulation Projects

SimpleScalar
SMPCache

C.3 Reading/Report Assignments

Many instructors believe that research or implementation projects are crucial to the clear understanding of the concepts of computer organisation and architecture. Without projects, it may be difficult for students to grasp some of the basic concepts and interactions among components. Projects reinforce the concepts introduced in the book, give students a greater appreciation of the inner workings of a processor. and can motivate students and give them confidence that they have mastered the material,

In 1his text. I have tried to present the concepts as clearly as possible and have provided numerous homework problems to reinforce those concepts. "...Ian>. instructors will wish to .supplement this material with projects. This appendix provides some guidance in that regard and describes support material available in the instructor's manual. The support material covers three types of 'projects:

- Research projects
- Simulation projects
- Readingireport assignments

RESEARCH PROJECTS

An effective way Of reinforcing basic concepts from the course and for teaching students research skills is to assign a research project, Such a project could involve a literature search as well as a Web search of vendor products, research lab activities, and standardization efforts. Projects could be assigned to teams or, for smaller projects, to individuals. In any case, it is best to require some sort of project proposal early in the term, giving the instructor time to evaluate the proposal for appropriate topic **and** appropriate level of effort. Student handouts **for** research projects should include the following:

- A format for the proposal
- A formal for the final report
- A schedule with intermediate and final deadlines
- A list of possible project topics

The students can select one of the listed topics or devise their own comparable project. The instructor's manual includes a suggested format for the proposal and final report as well as a list of possible research topics.

C.2 SIMULATION PROJECTS

An excellent way to obtain a grasp of the internal operation of a processor and to study and appreciate some of the design trade-offs and performance implications is by simulating key elements of the processor. Two useful tools that are useful for this purpose arc SimpleScalar and SMPCache.

Compared with actual hardware implementation, simulation provides two advantages for both research and educational use:

- With si mulation, it is easy to modify various elements of an organizalion. to vary the performance characteristics of various components, and then to analyze the effects of such modifications,
- simulation provides for detailed performance statistics collection, which can be used to understand performance trade-offs,

SimpleScalar

SimpleScalar [BURCi97, MANJO.la, NIANJO1b] is. ui set of tools that can be used to simulate real programs on a range of modern processors and systems. The tool set includes compiler, assembler, linker, and simulation and visuaiization tools. SimpleScalar provides processor simulators that range from an extremely fast functional simulator to a detailed out-of-order issue, superscaiar processor simulator that supports noriblocking caches and speculative execution. The instruction set architecture and organizational parameters may be modified to create a Variety t P experiments.

The instructor's manual for this hook includes ii concise introduction to SimpleScalar for students, with instructions on how to load and get started with SimpleScalar, The manual also includes some suggested project assignments.

SimpleScalar is a portable software package the mans on most UNIX platforms. The SimpleScalar software he clown loadci from the SimpleScalar Web site. It is iyai[able at no cost for noncqpininercial use.

SMPeache

StvtPC4ic:ht: i a trace-driven simulator for the analysis and teaching of cache memracy systems on symmetric multiprocessors [RODROn. 'The simul;ilion is based on a model built according to the architectural basic principles of these systems. The siniulalor has a full graphic and friendly interface_ Some of the parameters that they can be studied with the sim ulator are program locaiity; influence of the number of processors, cache coherence protocols, schemes for bus arbitration, mapping. replacement policies, cache size (blocks in cache). numilur of cache sets (for set associalive caches), number of words by Hock (memory block size)

The instructor's manual for this book includes a concise introduction to SimpleScalar for students, with instruciions on how to load and get started with SimpleScalar, The manual salsa includes some suggested project assignments.

SimpleScalar is a portable software package the tuns on PC systems with Windows. The SimpleScalar software can be downloaded from the SimpleScalar Web site. It is available at no cast for noncommercial use

C.3 READING/REPQRT ASSWNIVWINTS

Another excel lc ni way to reinforce concepts from the course and to give students research experience is to assign papers from the literature to he read and analyzed.

The instructor's nuinual includes a2.,u.ggQ4tQd list 0r popers_ {rc or two per chapWt, Lc) be assigned. All of the papers are readiby available either via the Internet or in miy good college iechnical library. The manual also includes a suggested assignment wording,

GLOSSARY

Sone of the terms in this glossary are front the *Aincriam Neiiirmal Dictio)rary for Informarion Systems* (1990. These are indicated in the *glos*miry by an asicrisk,

Ahsolute Address* An address in a computer language that identifies a storage Location or a device without the use of any intermediate reference.

Accumulator The name of the CPU register in a single-address instruction format. The accumulator. or AC, is implicitly one of the twd operands for the instruction.

Address **Bus** That portion of a system bus used for the transfer of an address. Typically, the dcidre:ss hic.ntifics a main munory location **or** 4111. I/O device.

Address Space The range 0r addresses (memory, I10) that.can be rEerenced.

Arithmetic and Logic Unit (A LU)* A part of a computer that performs arithmetic operations, logic operations. and related operations.

ASCII American Standard Code for Information Interchange.. ASCII is a 7-bit code used to represent numeric, alphabetic, and special printable characters, II also includes codes for *corilroi characters,* which are nol printed or diTlaycd bin sperify 7.:.orrie control funciion.

Assembly Language A Coniputer-orienlcd language whose instructions *tire.* usually in one-to-one correspondence with computer instructions and that may provide facilities such as the use of macroinstructions. Synonymous with *compare* r-ilepentlem. fanguage.

Asgoriative Memory* A memory whose storage locations WI·G identified by their conlimts, **or** by a part or 1hcir cons cars, rat her ihan by their names or positions.

Asynchronous Timing A technique in which the occurrence of one event on a bus follows and depends on the occurrence of a previous event.

Autoindexing A form of indexed addressing in which the index register is automatically incremenicd or Lluercni **ell** led with h memory reference.

Base In the numeration system commonly used in scientific papers, the number Olaf is raised to the power denoted by the exponent and then

multiplied by the maniissa Lo determine the real number represented (c.g., the number 10 in the expression 23 x 10' —.270).

Rase Address* A numeric value that is used as a reference in the calculation of addresses in the execution of a computer program.

Binary Operator* An operator that represents an operation on two and only two operands.

Bit* In the pure binary numeration 'System, either of the digits 0 and **1,**

Block Multiplexor Channel A multiplexer channel that interleaves blocks of data. Sec also *byte rnultiplextir channel,* Contrast with *selector channel.*

Branch Prediction A mechanism used by the processor to predict the outcome of a program branch prior to its execution.

Buffer* Storage used to compensate for a di l'l Onrice in rate of flow **of** data, **or** time omurrenee of events. when transferring data from one device to another.

Bus A shared communications path consisting of oar or a collection of lines. In some computer systcrns, CPU, memory, and I/O components are connected by a Common bus. Since the lines are shared **by** all components, only one corn-ponent at a time can successfully transmit.

Bus Arbitration The process of•detcrrnining which competing bus master will he permitted access to the bus.

Ilk's Master A device attached to a bus that is capable of initiating and controlling communication on the bus.

Byte Right }rill- Also referred to as an *octet.*

Byte Multiplexor Channel* A multiplexer channel that interleaves bytes of data, See also *Meek MI leipleX r r to arenrff-* Contrast with *selector ehonne/*

Cache Coherence Protocol A mechanism to maintain, data validity among multiple caches so 'hal every data access will always acquire the most recenl version of the contents of a main memory word.

Cache Line A block of **dui** associated with a cache tag and the unit of Iransfer between cache and memory.

Cache Memory* A special buffer storage, smaller and faster than main slorage, that is used to hold a copy **of** instructions and data **in** main storage that are likely to be needed next by the processor and thal have been obtained auto-malically from main storage.

CD-ROM Compact Disk Read-Only Memory. A nonerasable disk used for storing computer data. Thc standard system LLSCN 12-em disks and can hold more lawn 5M Mbytes.

Central Processing **Unit (CPU)** That portion of a computer that fetches and exe-cutes instructions. It consists of an arithmetic and logic unil (ALLY), a control unit, and registers. Often simply referred lo as a *procc.v.vor.*

Cluster A group *of* inLereonnQctc.d, whole computers workinR together as a unified computing resource that can create the ill usion of being one machine. The term *whole computer* means a system that can run on own, Bran from are elusier.

Combinational Circuit* A Logic device whose output values. at any given ingtani, depend only on the input alucs at that time. A combinational circuit is a special case of a 5equeatiat circuit that does not have a storage capability. Synonymous with *cornhInatoria*

Compact 'Disk (CD) A nonerasable disk that stores digitized audio information.

Computer Instructive An instruction that can he recognized by the processing unit of the **computcl l'or which it** is designed. Synonymous with *machine inszruclifJn.*

Computer Instruction See A complete set of the operators of the instructions of a computer together with a description of the types of meanings that can be attributed to their operands, Synonymous with *mac-Nue inslruction set.*

Conditional Jump* A jump that Lakes place only when the instruction **that** specifies it is executed **and** specified conditions are satisfied. Con tra i with *uncondifional*

Condition Code A code that reflects the result of a previous operation (e.g., arithmetic). A CPU may include one **or more** condilion codes, which may be stored separately within the CPU or as Nil of a larger controf rcgister. Also known

Control **thal** portion of a system bus used for the transfer of control signals.

Control Registers CPU registers employed to control CPU operation. Most of these registers are not user visible.

Control Storage A portion of storage that contains microcode.

Control [.;nit That part of the CPU that controls CPt ; operations, including A IA.! olpera00a',, the movement of data within the CPU, and the exchange of data and control signals across WI-mil **interfaces (c .g.,** the system bus).

Daisy Chain* A method of device inLei-conneciion for determining interrupt priority **hy** connecting the interrupt sources serially.

Data litoi Th4i | portion of a system bus used for the transfer of data.

Data Communication Data transfer between devices. The term generally excludes 1/0.

Decoder* A device that has a number of input lines of which any number may carry signals and a number of output lines of which not more than one may carry a signal. therc being a one-to-one correspondence between the outputs and the combinations of input signak.

Demand Pagine The transfer of a page from auxiliary storage to real storage al the moment of need.

Direct Access* The capability to oblain data from a storage device or to enter **data** into a storage device in a sequence independent of their relativc position, **by** means of addresses that indicate the physical location of the data.

Direct Address* An address that designates the storage lunation of an item of data to be treated as operand. Synonymous with *acre-level address.*

Direct Memory Access (**WO A) A forth** of I/0 in which a special module, called a *.ou.iiiturc,* controls the exchange of data between main mentorY and an 110 module.. The CPU sends request for the transfer of a block of data to the DMA module and is interrupted only 4i1Ler the entire block has been iranw. ft:fled.

Disahled Interrupt. A condition, usually created by the CPU, during which the CPU will ignore interrupt request signals **of fl** specified class.

Diskette* A flexible magnetic disk enclosed in a protective container. Synonymous with fie.v.th/c *disk.*

INA Yuck* An assembly of magnetic disks that can be removed YS a whole from a disk drive, together with a container from which the assembly must he separated when operating,

Disk Stripping A 1ypc of click array mapping in which logically contiguous hlocks of data, or strips, are mapped round-robin to consecutive array members. A set of togicall!,' consecutive strips that maps exactly one strip to each array member is referred to as a stripe.

Dynamic RAM A RA Nil whose cells are implemented using capacitor'— A dynamic RAM will gradually lose its data unless it is periodically refreshed.

Emulation* The imitation of all or part of one system by another, primarily by hardware, so that the imitating sys.tum accepts the satne data. executes the same programs. and achieves the same results as the imitated system.

Enabled Interrupt A condition, usually created by the CPI:, during which the CPU will respond to interrupt request signals of a specified class.

Erasable Optical Disk A disk that uses optical technology but that can be easily erased and rewritten. Both 3.25-inch and 5.25-inch disks ;ire in use. .A t!'.pical capacity is 65Ct Mbytes.

Error-Correcting Code* A code in which each character or signal **conforms to** specific rules of construction so that deviations from these rules indicate the presence of **an** error and in which some or all of the detected errors can be corrected MI i0111 aticaEly.

Error-Detecting Code* A code in which each character or signal conforms to specific rules of construction so that deviations from these rules indicate the presence of an error.

Execute Cycle That portion of the instruction cycle during which the CPU performs the operation specified by the instruction opcode.

Fetch Cycle That portion of the instruction cycle during which the CPU fetches from memory the instruction to he executed.

Firmware' Micr000de stored in read-only memory-

Fixed-Point Representation System* A radix numeration sy5tern in which the radix point is implicilly fixed in Ilse :series. of digit places by some convention upon which agreement has been reached.

Flip-Flor A circuit or device connlining active elements, capable of assuming either one of two stable states at a given time. Synonymous with *bistabk circuit. toggle.*

Floating-Point Representation System* A numeration sy'stem in which a real number is represented by a pair of distinct numerals, the real number being the product of the fixed-point part, one of the numeral*, and a VALIQ obtained by raising the implicit floating-point base to -a powur denoted by the exponent in the floating-point representation. indicated by the second numeral.

G Prefix ineanin2

Gate Art electronic circuit that produces an output signa] that is a simple Boolean operation on its input signals.

General.•Purpose Register* A register, usually explicitly addressable, within a set of registers, that can be used for different purposes, for example. as an accumulator, as **all** index register, or as a special handier of data,

Global Variable A variable (lathed in one porlion or a vienputer RT0141071 and used in at least one other portion of that computer program.

High-Performance Computing (I-IPC) A research area dealing with supercomputers and the software that runs on supercomputers. The emphasis is on scientific applications, which may involve heavy use of vector and matrix cumpination, and parallel algorithms.

I**mmediate Address*** The contents of an address part that con1ains the value of an operand rather than an address- Synonymous with *:cm-level eznetrr.saa.*

Indexed Address* An address that is modified by the content of an index register prior to or during the execution of a computer instruction.

Indexing A technique of address modification by means of index registers.

Index Register* A register whose contents can be used to modify an operand address during the execution of computer instructions; it can also be used as a counter. An index register may be used to control the execution of a loop. to control the use Of all array, as a switch, Cur !able 104 kup, or as a pointer.

Indirect Addrese An address or a storage locnlion thaI corilam.:. an address.

Indirect Cycle That portion of thc instruction cycle during which the CPU performs a memory access to convert an indirect.address into a direct address.

Input-Output (I/O) Pertaining to either input or output, or **both.** Refers to the movement of data between a computer and a directly attached peripheral.

Instructilm Address's Register* A special-purpose register used to hold the address of the next instruction to be executed,

lustructiou Cycle The processing Ferrol mei by, a CPU to execute a single instruction.

Instruction Format The ia..l, out of a computer instruction as a sequence of bits. The format divides the instniction into fields, corresponding to the constituent elements of the instruction (e,g., opcode, operands).

lustrudion Registe0 A register t hat is used to hold an instruction for inlerpretation.

integrated Circuit (IC) A tiny piece of solid material, such as upon which is etched or imprinted a collection of electronic components and their interconnections.

Interrupt* A :sus ension of a process such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed. Synonymous with *itnerrupth»?.*

Interrupt Cycle That portion of the instruction cycle during which the CPU checks for interrupts, If an enabled interrupt is pending, the CPU saves the current program state and resumes processing at an interrupt-handler routine.

Interrupt-Driven 1/0 A form of 1/0. The CPU issues an 110 command, continues to execute subsequent instructions, and is interrupted **by** the I/O module when the latter has complcicd its work.

110 Channel A relatively complex I/O module that relieves the CPL I of the details of 1.0 operations. An 1/0 channel will execute a sequence of I/0 cornmanc.N from main memory without 111e need for CPU involvement.

1/0 Controller A reiatively moduie that requires detailed control from the nit.) or an 1/0 channel Synonymous with *device confrolle r.*

1/0 Module One of the major component types of a computer_ It is responsible for the control of one or more external devices (peripherais) and for the exchange or data between th e devices and main memory and/or CPU registers.

I/O Processor An I/O module with its own processor. capable of executing its own specialized 1.0 instructions or, in scorns eaScf, general-purpose machine instructions.

Isolated I/O A method of addressing I/O modules and externai devices, 'The 1/0 address space is treated separately from main memory address space. Specific 110 machine instructions must be used. Compare *077(:nu) ry - rna pp e d*

K Prefix meaning = Thus., 2 kb = 2048 bits,

Local Variable A variable that is defined and used °rib in one specified portion of a computer program.

Locality of Reference The tendency a processor to access the same set of memory locations repetitively over a short period of time.

M Prefix meaning 2'" = 1,048,576. Thus. 2 Mb — 2097,152 bits_

Magnetic Disk* A flat circular plate with a magnetizable surface layer, on one or both sides of which data can be stored.

Magnetic Tape A tape with a magnetizable surface layer on which data can be stored by magnetic recording.

Mainframe A term originally referring to the cabinet containing the central processor unit or "main framc" of a large batch machine. After the emergence of smaller minicomputer designs in the early 19705, the traditional larger machines were described as mainframe computers, mainframes. Typical characteristics of a mainframe are that it supports a large database, has elaborate 1/0 hardware, and is used in a central data processing facility.

Main Memory* Program-addressable storage from which instructions and other data can be loaded directly into registers for subsequent execution or processing.

Memory Address Register (MAR)' A register, in a processing unit, that contains the address of the storage location being accessed,

Prlemory Buffer Register (MRR) A register that contains data read from memory or data **to he** written to memory.

Memory Cycle Time The inverse of the rate at which memory can he accessed. 11 is the minimum time between the response to one access request (read or write) and the response to the next access request.

Memory-Mapped 1/0 A method of addressing I/O modules and external devices. A single, address space is used **for** both main memory and 110 addresses, and the same machine instructions arc used both for memory readlwrite and for W.

Microcomputer* A computer system whose processing unit is a microprocessor. A basic microcomputer includes a microprocessor, storage, and an input./ output facility, which may Or may not be on one chip.

Prlicroinstructiote An instruction that controls data flow and sequencing in a processor at a more fundamental level than machine instructions. Individual machine instructions and perhaps other functions may be implemented by microprograms.

Micro-Operation An elementary CPU operation, performed during one clock pulse.

Microprocessor* A processor Ve 11 ONG elements have been miniaturized into one or a few integrated circuits.

Microprogram,' A sequence of microinstructions that are in special storage where they can he dynamically accessed to perform various functions.

Microprogrammed CPU A CPU whose control unit is implemented using microprogramming.

Microprogramming Language An instruction set used to specify microprogram.

Multiplexer A combinational circuit that connects multiple inputs to ti single output. AE any time, only one of the inputs is selected to he passed to the output.

Multiplexor Channel A channel designed to operate with a number of 1/0 devices simultaneously. Several [10 devices can transfer records at the l-di me ti me by interleavino, items of data. See also *lyre tmlltiplexor channel, block multiplexor cheatttel,*

Multipromisorw A computer that has two or more processors that havexammon access Loa main storage.

Multiprogramming* A mode of operation that provides; for the interleaved execution of two or more computer programs by a single processor.

31oltitaskine A mode of operation that provides for the concurrent periotmanec or interleaved execution of two or more computer lase. The same as multiprogramming, using different terminology,

N41111111W1111111 Memory Access **(NUMA) Multiprocessor** A shared-memory multiprocessor in which the access time from a given processor to a word in memory varies with the. location of the memory word.

Nonvolatile Memory Nlelnory whow will Lilts Lire stable and do not require a constand power **source-**

Nucleus That portion of an operating system that conlain '; its basic and most frequently used functions. Often, the nucleus remains resident in main memory.

Ones Complement Representation Used to represent binary integers. A positive integer is represented as in sign magnitude. A negative integer is represented by reversing each bit in the representation of **a pOS ir ive** integer *of* the same magnitude.

Opeode Abbreviated form. for *operation code.*

Operand* An entity on which an operation is performed.

Operating System* Software that controls the execution of programs and that proides services such as resource allocation. scheduling, input/output control, and data management.

Operation Coder. A code used to represent the opera inn; of a computer. Usually abbreviated to opcode.

Orthogonality A principle by which Iwo variables or dimensions are independent Or one LinoLlicr. In the CE3ElLeX1 of an instruction set, the term is **gencLully** used to indicate that other elements of an instruction (address mode, number of operands. length of operand) are independent of (not determined by opcode.

Page In a virtual storage system, a fixed-length block that has a virtual address and that is transferred as a unit between real storage and auxiliary storage.

Page Fault Occurs when the page containing a referenced word is not in main memory, ' rhis causes an interrupt and requires the operating system to bring in the needed page.

Page Frame* An area of main storage used to hold a page.

Parity Bit* A binary digit appended to a group of binary digits to make, the sum of all the digits either always odd (odd parity) or always even (even parity},

Peripheral **Equipment (IBM)** In a computer system, with respect to a particular processing unit. any equipment that provides the .processing unit with outside communication. Synonymous **with *periphcrof*** *devitv.*

Pipeline A processor organization **in which** the processor consiss oi a number of slages, allowing multiple instructions to he executed concurrently,

Predicated Lxecution A. mechanism that supports the conditional execution of individual **instructionli.** This makes **it** possible to execute speculatively both branches of a branch instruction and retain the results or the branch [hat ix uuim`utely token.

Process A program in execution. A process is controlled and scheduled by the operating system.

Process Control Block The manifestation of a process in an operating sysiern. ft is a date xlrucl **ure** containing information about the characteristics and state of the process.

Processor* In a computer, a functional unit that interprets and ewcalLe!.3 inslructims. A processor consists of at least an instruction control unit and an arithmetic unit.

Processor **Cycle Time The** time required for the shortest well-defined (' microoperation. II is the basic unit of time for measuring all CPU actions. Synonymous with *ynachine cycle rime,*

Program Counier Instruction address register,

Programmable. Logic Array (PIA)* An array of gates whose intereomneei itms **c4in** he programmed to perform a specific logical function.

Programmable Read-Only rit9emory (PROM) Semiconductor memory whose.contents may be set only once. The writing process is performed eleel rically and may be performed by the user at a time later than original chip fabrication.

Programmed **1/0** A form *of I/O in* **which the** CPU issues an I/O command to an I/O module and Inwi then wait fur the opera lion to he compfeie before proceeding.

Program Status Ward (PSW) An area in storage used to indicate the order in which instruetiOns are eNecutecl. **and to hol** d and indicate the status of the computer system. Synonymous with *proce.v.vor Alatu.s: word.*

Random-Access Memory (RAM) Memory in which each addressable location has a unique addressing mechanism. The time to access a given location is independent of the sequence of prior aeces ';.

Read-Only Memory (ROM) Semiconductor memory whose contents cannot be altered, except by destroying the storage unit. Nonerasable memory.

Redundant Array of **Independent Disks (RATD)** A disk array in which part of the phvsical storage capacity is used to store redundant informalion allow arts data stored on the remainder of the storage capacity. The redundant informa tion enables regeneration of user data in the event that one of the array's member disks or the access path itF it fails.

Registers High-speed memory internal to the CPU. Some registers are user visible: that available to the programmer via the machine instruction set. Other registers are used only by the C1-1.), for control purposes,

Scalar* A quantity characterized by a single value.

Secondary **Memory** Memory located outside the computer system itself, including disk and tape.

Selector Channel Art 110 channel designed to (Terme uiih only one I/O devick• a time, Once the 1/0 device is selected, a complete record is transferred one byte at a Lime Contras!. with *MK* mithiple.vor channel, prouldpleror channel.*

Semiconductor A solid crystallinc substance, such as silicon or germanium. whose electrical conductivity is intermediate between insulators and good conductors, Used to fabricate transistors and solid-state components.

Sequential Circuit A digital logic circuit whose output depends on the current input plus the state of the circuit: Sequential circuits lhtm possess the attribuic of memory.

Sign-Magnitude Representation Used to represent binary integers. In an N-hit word, the leftmost bit is the7.,ign (0 = positive, J = ncgalive) and the remaining N I bits comprise the magnitude of the number,

Solid-State Component* A component whose operation depends on the control of electric or magne1ic phenomena in solids (e.g., transistor crystal diode., ferrite core).

Speculative **Execution** The execution of instructions along one patio of a branch. If it Cater 1w ris out **that ib** is branch was not taken, then the results of the specuLitive execution are discarded.

Stack* A list that is constructed and ma.n.a.ncd so that the next item to be retrieved is I he most recently stored item in the list last-in-first-out (LIFO).

Stalk RAM A RAM whose cells are implemented using flip-flops. A static RAM will hold its data as long as power is supplied to it; no periodic refresh is required.

Superpipelined Processor A processor design in which the instruction pipeline consists of many very small stages. so that more than one pipeline stage can he executed during one clock cycle and so that a largc number of instructions nu!). he in the pipeline al the same time..

Superscalar Processor A processor design I hat includes multiple-instruction pipelines, so that more than one instruction can he executing in the same pipeline stage simultaneous/v.

Symmetric Multiprocessing (SMP) A form of multiprocessing that allows the operating system h execute on any available' processor or on several available processors simultaneously.

Synchronous Timing A technique in which the oceunrence of events on a bus is del ermined by a dock. The clock defines equai-width time stots, .und events begin only at the beginning of a time skit.

Spaeth Bus A bus used to interconnect major computer components (CPU, Tr) — ory, I10),

Truth Table* A table [hal cla scri bes a logic function by listing all possible combinations of input values And indicating, for each combination, the output value.

Twos Complement Representation geed to represent binary integers. A positive integer is represented as in sign magnitude. A negative number is represented by taking the Boolean complement of each bit of the corresponding positive, number, then adding 1 to the resulting bit pattern viewed as an unsigned integer.

Unary Operator* An operatc.lr that represents an operation on one and only one operand.

Unconditional Jump* A jump thit lakes place whenever the instruction that specified it is executed.

Uniprocessing Sequential execution of iristrucLions by a processing unit. or independent use of a processing unit in a multiprocessing system.

User-Visible **Registers** CPU registers that may be referenced **by the progrAininer,** The instruction-set forrmil a,Ilows **one or more registers to be specified as operands or** addresses of ope 11 TiLk.

Vector* A quantity usually Oun nicrized by an ordered set of scalars.

Very Long Instruction Word Refers to the USC of instructions that contain multiple operations. In effect, mulliple insunicrium are contained in a single word, 'I'ypica Fly, a **VLIW** is constructed by the compiler, which ptaces operations that may be executed in parallel in the same word.

Virtual Storage* 'Flit,: storage space that may be regarded as addressable main storage by the user of a computer sy d ern in which virtual addresses are mapped into real•addresses. The size of vi itruil Ntoragc. limiled by the addressing scheme t.pi the computer system and by the amount of auxiliary torage available, and not by the actual number of main storage locations.

Volatile Memory A memory in which a constant e]ectrica[power **source** is required to maintain the contents of memory. if the power is switched off, the shored information is lost

REFERENCES

Abbreviations

ACM Association for Computing Machinery

| EEE Institute 01' EleetricA I one. **ktecironics** Engineers

ABB000 Abbot, D. *PC/ BUN DernwoVied.* Eagle Rock, VA: **LLFI** Technology Publishing, 2000.

ACOS86 Acosta, R.; KjeEstrup, J.: and Torng, II. "An Instruction Issuing Approach to Enhancing Performance in Multip1e. Functional Unit Proecmors." *IEEE 7'ran.Y0c6om on Computers.* Scpternher I 9K(i.

ADAM9I Adarnek. J. I **oundations** *qt C":' ding.* Now York; Wiley. 1991.

AGAR89 A.garwal. A. *Analysis of Cache Peo formance),oe rah ag. SysteiTIN ihqd M4Juftiprcrgramrning.* Hosion: Kluwer Aeadernie Pubtkhers, 1989.

ACER87 Agerwata, T., and **Cocke, J.** *High Performance Rethiccd Insrruktion Set Pri)cesvors.* Technical Report RC12434 (#55845). Yorktown, NY; IBM Thomas J. Watson Research Center. January 1967.

ALEX93 Alexandridis. N. *Design of Microprocessor-Based Systems.* Englewood Cliffs, NJ: Prentice Hall, 1993,

A NDF:671.1 Anderson, **Or;** Sparacio, F.: and Tornando, F. "The 11-1N1 System/360 Model 91! Machine Philosophy and In traction. Handling.'" *IBM Journal of ReA.a.rch and Development.* January 1967.

ANDE670 Anderson, S., et al. The IBM System/360 Model 91: Floatina-**Point** Execution Unit." /BM *Journal 0,f Research and Development,* January 1967. Reprinted in [SWAR90, Volume 1].

ANDF.98 Anderson, D, *Fire Wire System Architecture.* Reading, MA: Addison-Wesley. 199S.

ATIO94 Atkins, " pc so ftwa re Performance Tuning," *IEEE Coinpurer.* August 19%.

AZIM92 Azimi, Prasad, **B.;** and Bhat, K. "Two Level Cache Architectures." *Proceedings COMPCON* '92. February 1992.

BAEN97 Baentsch, M., et al. "Enhancing the Web's Infrastructure: From Caching to Replication." *Internet Computing,* M arc h pi-il 1497.

BAIL93 Bailey, D. "RISC Microprocessors and Scientific Computing." *Proceedings, Supercompuring '93,* 1993.

IIASH$I Bashe, C.; Bucholtz. W.; Hawkins, G,; Ingram. 1: and Rochester, N. "The Architecture. of IBM's Early Computers." *IBM Journal of Research and Development.* September 1981.

BASH91 Bashicen. A.; Lui. **I,;** and Multan. J. "A Superpipeline Approach to the MIPS Architecture." *Proceedings, COMPCON Spring '91,* February 1991.

BELLIO Bell, *C.;* Cady, **R** McFarland, H. Delagi, O'Loughlin. **J.;** and Noonan. R. "A New Architecture for Minicomputers—The DEC PDP-11." *Proceedings. Spring Joint Computer Conference,* 1970.

BELL7 la Bell, C.. and Newell. A. *Computer Structures: Readings and Examples.* New' York: McGraw-Hill, 1971.

BELL78a Bell, C.; Mudge, **J.;** and McNamara, *J. Computer Engineering: A DEC View of Hardware Svstons Design.* Bedford, MA: Digital Press. 1978.

BELL78b Bell, C.; Newell. A.: and Sicwiorek, D. "Structural Levels of the PDP-8." In IBELL78a I.

BELL78c Be]], *C.;* Kotok, A.; Hastings, T.: and Hill. R. "The Evolution of the DEC System-10. ˉ *Communications of the AC* M, January 1978.

BENH92 Benham. **J.** "A Geometric Approach to Presenting Computer Representations of Integers." *SIGCSE Bulletin,* December 1992.

BETK97 Betker. M.; Fernando, **and Whalen,** S_ ˉThe I1istcry of the Microprocessor." *Bell Labs Technical Journal,* Autumn 1997.

RH A ROO Bharandwaj, J.. et al. "The Intel IA-64 Compiler Code Generator." *I EL L kr() ,* ScptemberiOctober 2000.

RL A A97 lilaauw, (.1, and Brooks. F. *Computer Architecture: Concepts and Evolution.* Reading, MA: Addison-Wesley, 1997.

BLAH83 Blahut, R. *Theory and Practice of Error Control Codes.* Reading, MA: Addison-Wesley, 1983.

BOHR98 Bohr, M. "Silicon Trends and Limits for Advanced Microprocessors." *Communications. of the ACM,* March 1998.

BRAD91a Bradlee, D.; Eggers, S.; and lienry, R. 'Ile Effect on RISC Performance of Register Set Size and Structure Versus Code Generation Strategy." *Proceedings, 18th Annual International Symposium on Computer Architecture, May I99I,*

BRAD91b Bradlee, D.; Eggers, S.: and Henry, R. "Integrating Register Allocation and Instruction Scheduling for RISCs." *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systemy,* April 1991.

BREW97 Brewer. E. "Clustering: Multiply and Conquer." *Data Communications,* July 1997.

BREVOO Brey. B. *The Intel Microprocessors: 8086/8066. 80186/80188, 80286. 80386,80486, Pentium, Pentium Pro and Pentium II Processors,* Upper Saddle River, NJ: Prentice Hall, 2000.

BURG97 Burger, D.. and Austin, T. "The SimpleScalar Tool Set, Version 2.0." *Computer Architecture News.* June 1997.

BU R 1‘46 Burks. A.: Goldsiinc., H.; and von Neumann, J. *Preliminary Discussion of the Logical Design of an Electronic · Computer hostrument.* Report prepared for U.S. Army Ordnance Dept.. 1946, reprinted in [BELL7I

litl.l Y **Y99b** Buyya. R. *High Performance Cluster Computing: Progranuning and Applications.* Upper Saddle River, N.1: Prentice I Ia]], 1999.

CART96 Carter, 1, *Mieroprocesser Architecture and Microprogramming,* Upper Saddle River. NJ: Prentice Hall. 1996.

CATA94 Catanzaro, B_ *Multiprocessor System Architectures.* Mountain View. CA: Sunsoft Press,.1994.

CliA182 Chaitin. G. 'Register Allocation and Spilling via Graph Coloring." *Pmceedings, SIGPLAN Sympt.Psiton on Compiler Construction.* June j.982.

CARMOO Carmean, D. "Inside the High-Performance Intel Pentium 4 Processor Microarehitecture. ¯ *Intel Developer Forum,* Fall 2001)_ ftrildownloadintel.comi design id 1al120001presentationsipda/pda_s01cd,pdf.

CHASOO Chasin, A. "Predication, Speculation. and Modern CPUs." *Dr, Dobb's Journal.* May 2000.

CIIEN94 Cheri. **P.;** Lee, E.: Gibson, *Cl.;* Katz, and Patterson. D. ¯ RAID: High-Performance, Reliable Secondary Storage." ACM *Computing Surveys,* June 1994.

CHOW86 Chow, F.; Himmelstein, M.: Killian, E.: **and Weber,** L. "Engineering a RISC Compiler System." *Proceedings, COMPCON Spring '86,* March 1986.

C110W/47 Chow, F.: Correll. S.; 11 immelstein, M.: Killian, ti.: and Weber, L. "How Many Addressing Modes Are Enough?" *Proceedings. Second Ititernationai Conference on Architectural Support fOr Programming Languages and Operating Systems.* **October** 1987.

CHOW90 Chow, F., and Hennessy, J. "The Priority - Bascd Coloring Approach to Register Allocation." *ACM Transactions on Programming Languages,* October 1990.

CLARK5 Clark, D., and Emer, J. "Performance of the VAX-1117811 Transiation Huffer. Simulation **and** Measurement." *ACM Transactions on. Computer Systems,* February 1985.

CLEMOO Clements. A. "The Undergraduate Curriculum in Computer Architecture, ¯ *IEEE Micro,* May/June 21110.

COHER Cohen. D. "On Holy Wars and a Plea For Pe4wc." *Computer,* October 1981.

COLW85a Colwell, R.; Hitchcock, C.: Jensen, 1 ¹:.; Brinkley-Swum, IL: and Kollar, C. "Computers, Complexity, and Controversy." *Computer,* September L985.

COLW85h Colwell, R.; I litcheoek, C.; Jensen, E.; and Sprunt,. I I. "More Controversy About 'Computers, Complexity. and Controversy.' " *Computer.* December [985.

COME95 Comerford. R. An Overview of High Performance." *IEEE Spectrum.* April 1995.

COMM) Comerford. R. "Magnetic Storage: Thu Medium that Wouldn't Die." *IEEE Spectrum.* December 21N14}.

C00K$2 Cook, R., and Dandu, N. "An Experiment to Improve Operand Addressing." *Proceedings, S›mpoNifin2 on Architecture Support for Programming* ***Lctrrgauges*** *and Operating Systems,* March 1982.

COON81 Coonen, J. "1. fiderflow and Dcnormalized Numbers." *IEEE Computer,* March 1981.

COUT86 Coutant, D.; Hammond, C'.: and Kelley, J. "Compilers for the New Generation of Hewlett-Packard Computers." *Proceedings. COM PCON Spring '8i5,* March 198fi,

CRAG79 Cragon, H. "An Evaluation of Code Space Requirements and Performance of Various Architectures." *Completer Architecture News,* February 1979.

CRAG92 Cragon, H. *Branch Strategy Taxonomy and Performance Models.* Los Alamitos, CA; IEEE Computer Society Press. 1992.

CRAW90 Crawford, J. "The 486 CPU: Executing Instructions in One Clock Cycle." *IEEE Micro,* February 1990.

CRIS97 Crisp, R. "Direct RAMBUS Technology: The New Main Memory Stan• dard." *IEEE Micro,* NovemberiDecember 1997.

DA,TT93 Dattaireya, G. "A Systematic Approach to 'reaching Binary Arithmetic in a First Course." *EE Transactions on Vducarion,* February 1q93.

1]A.V187 Davidson, and Vaughan, R. "The Effect of instruction Set Complexity on Program Size and Memory Performance." *Proceedings, Second* ***International*** *Conference on Architectural Support for Programming Languages and Operating Systems,* October 1987.

inCNN68 Denning, P. "The Working SO Model for Program Behavior." *Communications of the ACM, Mxt* 1968.

DEWA90 Dewar. R., and Smosna, Ikk.1 :._.cropro (**epion:** A *Programmer's View.* New York: McGraw-Hill, 1990.

DIJK63 Dijkstra. E. "Making an ALGOL-Translator ['or the XL' in *Annaed* Review (yf/triwmatic *Programminx, Volume 4.* Pergamon, 1963,

DOET97 Doetting, Ci,,, et a]. "S/390 Parallel Enterprise Server Generation 3! A Balanced System and Cache Striae:lure,'" *113.4.1 Journal of Resvareh (Ma DePelopmenl,* ***y ISQpiCInbei 1997,***

DOWD98 Dowd, K., and Severance. C. *High Performance Uomproing.* Sebastopol, CA: O'Reilly. 1998.

DUHE91 Dubuy. F., and Flynn, M, ¯Branch Strategies: Modeling and Optimization." *IEEE Transucaon,5* (Pn *Contputers,* October **1991.**

DUL098 Dulong, C. "The IA-64 Architecture ;it Work." *Computer,* **July** 1998.

ECKE9(1 Pekert, k. "C..timmunieation Between Computers and Peripheral Devices—An Analogy." *ACM SIGCSE Bulletin,* September 1990.

ELAY85 yai, K., and Agorwal, **R. "Thu** Intel 80356—Architecture and lmplemen4ition.." /EFT *Micro,* December 1985,

EVENOPO Even, G., and Paid. W. On the Design **of** WEE Compliant Floating-Point Units." *IEEE Transocrions on Computers.* May 2000.

RVER9$ 2vers, M., et al. ¯An Analysis of Correlation and Predictability WhaI Makes Two-Level Branch Predictors Work." *Proceedings, 25th Annual International Symposium* ***rin*** *Microarchrecture,* **July** 119S.

EVF,11011 lavers, M., and Yell, T. "1 hiderstanding Branches and Designing Branch Predictors for High-Performance. Microprocessors," *Pracceclings of the IEEE,* November 2001.

FA 0192 E;irrnIA'&d, ;l **nd Mooring, D. ¯A** Fast Path to One Memory." *IEEE Spectrum,* October 1992.

FITZ81 Fitzpatrick, D.. et **ad,** "A RI SCy Approach VISL." *VLSI Design.,* 4th qtiarter, 1!-.}51. Reprinted in *Computer* A *rchitecture* News., March 1982.

FLYN71 Flynn, M., and Rosin, R. "Microprogramming; An Introduction and a Viewpoint," *IEEE Transacrion.y on ComputerN,* July 1971.

FLYN72 Flynn, M. ¯Some Computer Organizations and **Their Etkcliwncss."** *IEEE Trumactions on Compurers,* September 1972.

FLYN85 Flynn. M.; Johnson. **J.; and** Wakefield, S. 'On Instruction Sets and Their I iormats." *IEEE lrumac:tiOnN* ***oft COMpliten, Mareli 1985.***

FLYN87 Flynn. M.: Mitchell, C. and Mulder, J- "And Now a Case for More Complex Instruelion Sets.' *Ccnnpurer,* Sepi ember 1987.

FLYNO1 Flynn, M.. and Oberman, S. Advanced *Computer Arithmetic Desio.* New York; Wiley'. 2001,

FRA183 Fraile!,•, **D. '`Word** Length of a Computer Architecture; Definitions and AI pkarions," *Cmnpuier Archirecture News,* June 1983,

FRI L96 I-'riedrnan, M. ¯RA 11) Kccps Going and Going rind..." l *EEE ST.PeCIFLAM* • April 1996.

FUR.1-1R7 Eurht, H. . and rvlitutinovic, V, "A Survey of Microprocessor Architectures for Memory Management." *Cirinputer,* March 1987.

FUTRO1 hulr 1. **W,** *InfintHand Architecture: Development and Deployment.* born, OR: Intel Press, 2001.

GIFF87 Gifford, D„ and Spector, A. 'Case Study: IBM's Systcm/360-370 Architecture,' *Commaniciaiony of the ACM,* April 1987.

GOLD91 Goldberg, D, "What Every Computer Scientist Should Know About 10.41lin g-Point A riihmutief· A *CM Computing Survrys.* March 1991, A ailab lc at http:Pwww.validgh.com/

RAND98 liandy.1. *The Cache Memory Hook,* San Diego: Academic Press, 1993.

HALF97 Halfhill, T. "Beyond Pentium II." *Byte,* December [997.

HAYE98 Hayes, J. *Computer Architecture and Or* New York: McGraw-Hill, 1998.

11EAT84 I Ica al, J. "Re-evaluation of RISC I.' *Computer Architecture News,* March 1984.

11ENN82 f lennessy. I.. et al. tHrdwArciSoCtviinrc **Tllideoffs** for Increased Performance." *Proceedings, Spy:pa...turn tm Architectural Support for' Programmin Languages and Operating Systems,* March 1982.

HENN84 Hennessy, J. "VLSI Processor Architecture." *IEEE Transactions on Computers,* December 1984.

HENN91 Hennessy, .1., and Joui)pi, N. "Computer Technology and Architecture: An Evolving Interaction." *Computer,* September 1991.

HENN96 Hennessy, J., and Patterson, D. *Computer Architecture: A Quantitative Approach.* San Mateo, CA: Morgan Kaufmann, 19%.

111D.A.90 I I ida ka, 111.; Matsuda, Y.! Asakura, M.; **end** Kai.uyasu, F. "The *Cache* DRA:v1 Architecture: A DRAM with an On-Chip Cache Memory." *IEEE Micro„).prik* 1990.

1-11 6B90 Higbie, L. "Quick and Easy Cache Performance Analysis." *Computer Architecture* NEWS, June 1990.

H1LL64 Hill, R. "Stored Logic Programming and Applications." *Datarnation.* February 19154,

IiILL89 Hill. M. "Evaluating Associativity in CPU Caches." *IEEE To·ansaerions on Computers,* December 1989.

HI T01 Hinton. G„ et a]. "The Microarchitecture of the Pentium 4 Processor. ¯ *Intel Technology Journal.* 01 2001. http:hdeveloper•intel,cornIicchnologylitj

lit/C1'.83 Huck, T. *CoMparative Analysis of Computer Arcisite.ctures,* Stanford University Technical Report No. 83-243, May 1983.

H1.]CKOO Huck. J., el. al. "Introducing the 1A-64 Al:ehiicuturi2." *.4 (c m.* SeptemberlOcti,ber 2(]00

HUGLE91 Huguel. M.. and Lang, T. "Architectural Support for Reduced Regis-1.cr SaYingiRestoring in Single-Window it egisler hics." A *CM Pramacrions on Computer SyNteins ,* February 1991.

1 11UTC96 Hutcheson, G. and Hutcheson. J. "Technology and Economics in the Semiconductor industry." *Scientific American,* January 1996;

II WA Ng3 Hwang, K. *Advanced Computer Archireoure,* New York: MeGraw-I **I ill, 1993,**

HWAN99 Hwang. K. et al. ¯ Designing &SI Clusters with Hierarchical Checkpointing and Single I/O Space." *IEEE Concurrency.* Jammu-March I999.

ITINU98 I i wu, W. "Introduction to Predicated Execution." *Computer,* January [998.

tiwu, W.; **August, D.:** and Sias, J. "Provram Decision Logic Optirni /a-lion Using Predication and ControL Speculation. ¯ *Proceedings o f the IEEE, November* 2001,

IBM94 international Business Machines, Inc, *The PowerPC* A *rchilecture: A Specification fclr a New Family of RISC' Processors.* San Frauckeo, CA: Morgan Kaufmann, 994.

HIM .01 international Business Machines, inc. 64 *Mb Synchronous DRA.M.* IBM Data Sheet 364164. January 2001.

1EEE85 inmiluiL of I.:lecirie;11 r,lecironies Engineers. *IEEE Standard for Binary fleerting-Point A riurunctic.* ANS1IIEEE Std 754-1985, 1985.

INTE9S Intel Corp. *Penthun Pro emef Pentium II Processors and Related Prodycts,* Aurora, CO, 199K

INTEood Intel Corp. *Intel IA -64* A *rchitecture Software Developer'.s Manual (4 yrd-unw.c).* Document 245317 through 245320. Aurora, CO, 2000.

I NTI'Mb Intel Corp. *Itanium Processor Microarchitccture Ikference Jor Software Optimization.* Aurora, CO, Doeurneul 245473. August 2000.

INTE0 la Intel C.orp. *IA -32 Intel* A *rchileciare SOThvezre Deyelopco-'.v Manual (2 volumes).* Document 245470 and 245471. Aurora. CO. 2001.

INTEOM Intel Corp. *Intel Pentium 4 Processor Optimization Hefrrcncc Manual.* Document 24896.6414. Aurora, CO, 2001_ http!Pdeveloper.intei.contidesign1 pentium4Imanuals/248966.htm.

JAME90 James, **D,** "Multiplexed Buses: The Endian Wars Continue.'¨ *IEEE Micro,* September **1983.**

JA RPO1 Jarp, S. "Optimizing IA-64 Performance. ¯ *Dr. Dobb's drowned, July* 2.1.001.

JOHN91 Johnson, M. *Superscalar Microprocessor Design.* Englewood MI's, NJ: Prcntici:: H41[1, 199 [

JOUP88 Jouppi, N. "Superscalar versus Superpipelined Machines." *Computer Architecture News,* June 1988.

JOU P89a Juuppi, N., and Wall. D. 'Available Instruction-Level Parallelism for Superscalar and Superpipclined Machin.es," *Proceedings, Third International C.'onfere.nce on Architectural Support for Programming Languages and Operating Systems,* April 1989.

JOUP8911 Jouppi, N. "The Nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance." *IEEE Transactions on Computers,* December 1989,

JTFO1 Joint Task Force on Computing Curricula. *Computing Curricula 2001 Computer Science.* IEEE Computer Society and ACM, August 2001.

KAEL91 Kacti. D., and Emma. P. 'Branch History Table Prediction of Moving Target Branches Due to Subroutine. Returns." *Proceedings, 18th Annual International Symposium on Computer Architecture, May* 1991.

KAGA01 Kagan, M. "InfiniBand: Thinking Outside the Box Design.' *Communications System Design,* September 2001.. (www.csdmag.com)

KANE92 Kane, G.. and Heinrich, J. *MIPS RISC Architecture.* Englewood Cliffs, NJ: Prentice Hall, 1992.

K A PPM Kapp, C. "Managing Cluster Computers," *Dr. Dobb".c Journal,* July 2000,

KATE83 Katevenis, M. *Reduced Instruction Set Compeeter Archifectures for VLSI.* Ph.1). dissertation, Computer Science Department, University of California at Berkeley, October 1983. Reprinted by MIT Press, Cambridge, MA, 1985.

KATHOI Kathail. B.; Schlansker, M.: and Rau. B. "Compiling for EPIC Architectures." *Proceedings o,f the* November 24K}1.

KATZ89 Katz, R.: Gibson, and Patterson, D, "Disk System Architecture for High Performance Computing." *Proceedings of the If* Dccembcr 1959.

KEET01 Keeth, B., and Baker, R. *DRAM Circuit Design: A Tutorial,* Piscataway. NJ: IEEE Press, 2001.

KHUR01 Khurshudov, A. *The Essential Guide to Computer Data Storage.* Upper Saddle River, NJ: Prentice Hall, 2001.

KNUT71 Knuth. D. "An Empirical Study of FORTRAN Programs." *Software Practice and Experience, vol. 1,1971.*

KNUT98 Knuth, D. *The Art of Computer Programming, Volume 2: Scm/numerical Algorithms,* Reading, MA: Addison-Wesley. 1998.

KUCK72 Kuck, D.; Muraoka, Y.: and Chen, S. "On the Number of Operations Simultaneously Executable in Fortran-like Programs and Their Resulting Speedup." *IEEE Transacoions on Computers,* December 1972.

KUGA91 Kuga. M.; Murakami, K.; and Tomita, S. "DNS (Dytm ruici ly-hazard resolved. Statically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture." *Computer Architecture News, June* 1991.

LEE91 Lee, R.: Kwok, A.: and Briggs, F. "The Floating Point Performance of a Superscalar SPARC Proc.cssor." *Proceedings, Fourth International Collference on Architectural Support for Progrunaning* uIF *and Operating Systeind,* April 1991.

LIL188 Lilja, D. "Reducing the Branch Penalty in Pipelined Processors.' *Computer.* July 1988.

1,11,193 Lilja, D. "Cache Coherence in Large-Scale. Shared-Memory Multiprocessors: Issues and Comparisons." ACM *Computing Surveys.* September 1993.

LOVE% Lovett, T.. and Clapp, R. "Implementation and Performance of a C7C:-N LIMA System." *Proceedings, 23rd Annual International Svmposium rut Computer Architecture,* May 1 996.

LUND77 Lunde, A_ "Empirical Evaluation of Some Features of Instruction Set Processor Architectures." *Cemnnunicasions of ate ACM.* March 1977.

LYNC93 Lynch, M. *Microprogrammed State Machine DeSign,* **Boca** Raton, FL: CRC' Press, 1993,

MACG&4 MacGregor, D.; .M othersole_ D.; and N.l.t)yer,13. Motorola Nit oiso20.- *IEEE Micro,* August 1984.

MAIIL94 Muhlke, S., et al. "Characterizing the Impact of Predicated Execution on Branch Prediction." *Proceedings, 27th International SVMpOSiHni on Microarchitecture,* December 1994.

MAHL95 Mahlke, S., et al. "A Comparison of Full and Partial Predicated Execution Support for 1LP Processors." *Proceedings, 22nd International Symposium on Computer Architecture,* .I urge 1995_

MAK97 Mak, P., et al. "Shared-Cache Clusters in a System with a Fully Shared Memory," *IHM Journal of Revearch and Development.* ***July/September*** 1997.

NIALL75 Mallach. E. "Emulation Architecture." *Completer.* August 1975.

MALL83 Mallach. E., and Sondak. N. Adrarwes *in Microprogramming.* Dedham, MA: Artech House, 1983.

MANJO1a Manjikian, N. "More PnhHncomenis of the SimpleScalar Tool scc *Computer Architecture* IVews. September 2001.

M A WI h Manjikian, N. "Multiprocessor Enhancements of the SimpleScalar Tool Set." *(l.'omputer Architecture* News, March 2001.

MAN001 Matto, M. *Logic and Computer Design Fundanulials.* Upper Saddle River, NJ: Prentice Hall, 2001.

MARC90 Marchant, A. *Optical Recording.* Reading, MA: Addison-Wesley, 1990.

MARKO° Markstein, P. *IA-64 and Oernentary limo ions.* Upper Saddle River. NJ: Prentice Hall PTR, 2000.

Ael A SH 95 Mashey, J. "CISC vs. RISC (or what is RISC really)." ***USENET*** *comp.arch ne‣ sgroup, article 46782,* February 1995.

MASS97 Nlas.siglia, P. The *RAID Book: A Storage System Technology Handbook,* St. Peter, MN: The Raid Advisory Board, 1997.

MAYB84 Mayberry, W., and Bland, 0, 'Carhe Boosts Muitiprocessor Performnice. ¯ *Converter Or.vign,* November 1984.

MCEL85 McEliece, R. The Reliability of Computer Mumorie.s." *.9cicntific American,* January 1985.

NIEF,961.1 C-, and plink]. **E..** eds.:Wagner& *Rece,.rding Technology.* Now York: McGraw-Iliil, 1996

MEE96b Mee, C., and Daniel. E. eds. *Magnetic seorage Handbook.* New York: McGraw-Hill. 1.996.

MILE60 Milcrikovic. A. "Achieving Elia Performance in Bus-Based Shared-Memory Multiprocessors," *IEEE Coneurrency.* My-September 2000,

MIHA92 Mira puri, S., **WOOdLicrQ,** M.; and V asseghi, **N.** "**The** MI PS 84000 Processor.' *I /TEE Micro,* April 1992.

MOORM Moore, Cr. "Cramming More Components Onto Inicgrmed Circuits," *Electronics Magazine,* April 19, 1965.

MOKS18 Morse., Pohlman. W.; and Ravenel, B. "The Intel 8086 Mieroprocesson A 16-bit Evolution of the 8080," *Computer.* June 197K.

MOSHO1 Moshovos. A., and Sohi.. G. " Mic.roarc.hi [Qcitiral Innovations: Boosting Microprocessor Performance Beyond Semiconductor Technology Scaling." *Procefulio ν of the IEEE,* November 2001.

MOTOR Motorola. Inc. PolverPC *MPC7-110 f?1.51:: Microprocessor Hardware Spreitieation.v.* I.)enver, CO: 2001. viiww.mororola•com

MYE.R7N Myers, "**I** he T. i2iluatiort of Expressions in a Storage-to-SLorage Architecture." *C.:ompeaer Architecture News,* .1 oii c 1978.

NAYF96 Nayfch, B.; Olukotun. N.7 1ri d Singh. .1. ¯Thc impact *or* Shared Cache Ciustcring in **Srnall-ScLilc SH1 0.1-1VIL2moi.7,' Multiprocessors.**" ***Proceedings of*** *the Second International Symposirtrn on hri.gh Perfrormancc Complacr Architecture, 1996.*

NOV193 Novitsky, Azimi, M.: and Ghaznavi, R. "Optimizing Systems Performance Based on Pentium Processors." *Proceedings COMPCON '9.2,* February 1993.

OBER97a Oberman. S., and Flynn, M. ¯I)esign Issues in Division and Other Floating-Point Operations." *IEEE Tranyaction.s on Computers,* February 1'

Oberman, S., and Flynn, M. "Division Algorithms and Implementations," *IEEE Trao?sactions on Computers.,* August 1997.

°VERO! Overton, NA. *Numerical Computing with IEEE Floating Point Arithmetic.* Philadelphia, PA: Society for Industrial and Applied MaLheinatics, **2001.**

PADER Padegs, A. "Systerni360 and Beyond." *I M Journal of Research and Development,* September 1981.

PADE88 Padegs, A.: Moore, Smith, R,: and Buchholz, W. "**Thc 113M** Sy--kill1370 Vet:lor Archiicc.turc: Design (.onsiderations.' *IEEE Transactions on Communicaeioir.s,* Mav 1)88.

PART-100 Parhanii, B. *Cum/Reff Algorithms and He Design,* Oxford: Oxford University Press, 2000.

PARK 89 Parker, A., and Hamblen, J. *An Introduction to Microprogramming* Ti,lrh *E..terci.se!..; Designed fur the Texas Instruments .5N74AC:T8M0 Software puvrriomen,' Board.* Dallas. TX; Texas Iw.truments, 1989.

PATT82H. Patterson. D,, Eind Sequin, C. "A VLSI RISC. ¯ *Completer,* September

PATT82b Patterson, D.. Find Piopho, R, "Asses;sing RISC.:s. in I ligh-Levei Language Support. ¯ *IEEE Micro,* November 1982.

PATT84 P4i.Lterson. D. "RISC V1/4142h. ¯ *Computer Architecture News,* March 1984.

PAT 1'85a Patterson, D. 'Reduced Instruction Set Computers. ¯ *(ommemicarions of Ile AC .* Janeary 1955.

PATES5b and Hennessy, J. "Response to `Computers, Complexity, and Controversy." *Computer,* November 1985.

PATI'88 Pailerson, I).; (libHon,•G.; md K41.1., R. "A Case for Redundant Arrays of lnexpensi...e. Disks (RAID). ¯ *Proceedings, ACM SIGMOD Cemference of Management of Data,* June 19.SK,

PATT98 Patterson, D.. and Hennessy, J. *Computer Organization tend De'tign. Thu Hardware/Software Interface. S;in rvial.o..1, CA:* Morgan Kaufmann, 1998.

PATTO1 Pact, Y. "Requirements, Bottlenecks. Lind *Good* Fortune: AgenEs for Microprocessor Evolution. ¯ *Proceedings of the 'EEL* November 2001.

PE1R99 Peir, Hsu, W.: and Smith, A. 'Functional JmplementaLion Techniques for CPU Cache Memories. ¯ *1 EE Tran.vacrions on Compriters.* February 1999.

PELE97 Peicg. A.; 'Wilkie, S.; and Weiser, U. "Intel NAM X for Multimedia PCs." Communicinion.v *01 the ACM,* January 1.997.

pils98 *In Search of Clumer.v. Lipper* Saddle River. NJ: Prentice Hall, 1998.

POPL91 Popcseu, V., ot al. ¯The Metallow Architecture." *IEEE Micro,* June 1991.

POT f'94 Potter, 'F., et al, "Resolution or Difia and Control-How Dependencies in Lhe PciwerPC 601. ¯ *IEEE Micro,* October 1994.

PRLSOI Pressd, D. ¯Fundamental Limitations on the Use of Prefetching and Stream Buffers for Scientific Applications. ¯ *Proceedings, ACM Symposium on Applied C'ompetting.* March 2001.

PRIN9I Prince, B. *Semiconductor Memories.* New York: Wiley, 1991.

PRIN99 Prince. B. *MO) Performance Memories:* New *Architecture .DRAMs and SRAAis, Evolution and Feencliewr.* New York: Wiley, 1999,

PIC/X88 Przybylski- Horowil2- M.; and Hennussy, J. 'Performance Trade-offs in Cachc Design. ¯ *Proceedings, Fifteenth Animal Imernational* ***SyMp05411*** on *Computer Architecture,* June 1988.

PRZY9O Przybylski. S. "The Performance Impact of Block Siz.c.axid Fetch Smite- *Proceeciings, 17th Annual International S'yptwo.sht.r71* ***t311 CiMapetrer*** *Architecture,* **Mai). 19[1]40.**

HA D183 Raclin, U. "The 8111 Minicornpuier. ¯ Ili,1 *Joerrourt of Research and Developmenr,* Ma)' 1983.

RAGA83 Ragan-Kelley, R., and Clark, R. "Applyina RISC Theory to a Large Cotnputer." *Compwer Design,* November 1983.

RAIJS80 Rauscher, T.. and Adams, P. "Microprogramming; A Tutorial ***4i nd*** **Survey** of Recent Developments," *IEEE Transactions on Computers,* January 19 W.

Hal-198 Neches, S.. and Wcks, S. "Impleinentation and Analysis of Path History in Dynamic Branoh Prediction Schemes." *IEEE Transactions on Computers, Au2,ust* 1998.

RODROI Rodriguez, **M.; Perez, J..** and Pulido. J. An Educational 'fool ror 'II:sting Caches on Symmetric Multiproce ssors." *AlicroprocesAors ana[1] Microsystems,* June 2001.

H()5 [999] *Roscll,W , Winn I,. gosch Hardware Bible. Indianapolis,IN1Sams,* **1999,**

SATY81 SaIyanarayanau. M., and Bhandarkar, D. "Design Trade-Offs in VAX-1 I 'Translation Buffer Organization." *Cornpwer,* December 1981,

SCI-IA91 Sehaiier, R. "Moore's Law; Past. Present and Future." *IEEE Spectrum. .1 Line 1997*

SCI11,00a SchEansker. M.; and Rau, B. "EPIC: Explicitly Parallel instrucl ion Computing." *Comparer.* February 2000,

SCHLDOh Schlansker. M.; and Rau, **B.** *EPIC' An Architecture for frr,srruction-Level Parallel Procevsors,* **`IecILnii;r,** I Report HPL-1999-1.11, Hewlett-Packard 1_2thoratories (www.lipi.hp.com), February 2000.

SC1-1W99 Schwarz, E., and Krygowski, C. "The SI390 G3 Floating-Point Unit, ¯ *IBM Journal of Reszarch and Development,* SeptembeeNovember 1999.

SELIE76 Sebcrn. M. "A Minicomputer-compatible Microcomputer System: The DEC LSI-1. I." *Proceedings of the IEEE,.* June 1976,

8E6E91 Segee, B.. and Field. J. *Microprogramming* anti *Computer Architecture.* New York: Wiley. 199]..

SERL86 Serlin, 0, "NIPS, DhryMonts. and Other 'fates." Daminarkm, June 1. 1986.

SHAN38 Shannon, C.. "Symbolic Analysis of Relay and Switching Circuits." *AP:LE Tram:actions.,* vol. 57, 1938.

SHAN9SH 4hanlcv, T-. and Anderson, D. *PCI Sy.vrems Architecture.* Richardson, [X7 Mindskire 1995.

SHA:sP95b SI-Lankly, '1'. *PenverPC System Architecture,* Reading, MA: Addison-Wesley, 1995.

HA,NOS Shanlev, 1. *Pentium Pro rand Perthuon II System Architecture.* Reading, MA! Addison-Wesley, 1998.

SIIAR97 Shama, A. *Semiconductor Memories: Technology, Testing, and Reliability* New York:. IEEE Press, 1997.

S./ IA R00 Sharangpani,1.1., and Aron a, K. "Itan ium Proc cs Microarchitr.Leturc." *IEEE Micro,* September/October 2000,

SIIER84 Sherburne, R. *Processor Design Tradeoffs in VLSI. PhD* tin,:sis, Report UCBICSD 841173, University of CAirornia iL BC] LAcy. April 19,S4.

SIEW82 Siewiorek, Bell, and Newell, A. *Cornpufrr Structures: Principle. and 1';vamples.* New York; McGraw-Hill, 19g2.

SIMA97 Sima, D. "Superscalar 1rr.truction Iswe." *WEE Micro,* September/October 1997.

SIM069 Simon. H. *The Sciences Qt the A rtificia* Cambridge, MA: MIT Press.1969.

SMIT82 Smith, A. "Cache Memories_" *ACM Coonputing Surveys,* September 1992.

SMIT87 Smith, A. "Line (Mock) Size Choice ror CE'lll C:adie Memoricr *IEEE Transactions on Cornnnenieation.s,* September **1987.**

SMIT89 Smith, M.; Johnson, M.; and 1 M. "Limits on Multiple instruc-Lion lssut=_' Prweedingy, *'third International Conference on A rchitectural Support for Programming Languages anti Operating Systems,* April 1989,

SMIT95 Smith, J.. and Sohi, G. ¯The Microarchitecture of Superscalar Processors," *Proceedings of the IEEE,* December 1995,

SOD [96 Soderquist, P., and Le.eser, M. Area and Performance Tradeoffs in Floating-Point Divide and Square-Roos Implemcniation..s." *ACM Computing Surveys',* September 1996.

5011190 Soht. 0. "Instruction Issue],ogic for 1.1igh-Performanee interruptable. Multiple Functional **Computers."** *IEEE Transactions on Cornpeuers,* March 1990.

$TAlmo stonin gs, *o,ot, and Computer COMPIIliniCatiOn!i,* 5th edition. Upper Saddle River, NJ: Prentice Hall, 1997.

SFAL01 Stallings, *Operating Systems, Internals and Design Principles,* 4th edition. Upper Saddle River, NJ: Prentice 2001.

STE W Stenstrom P. "A Survey of Cache Coherence Schemes or Multipmcessors." *Computer.* June [990.

STEV64 Stevens. W. "The Structure of Systeml360, Pali II: System Implementation." *IBM Systems Journal,* VoL 3, No. 2, 1%4. Reprinted in 1S1EW821.

STON93 Stone., **H.** *High-Perforentince Computer Architecture,* **}Reading,** MA: Addison-Wesley. 1993.

STR E.78 Strecker, W. "VAX-111780: A Virtual Address Extension to the DEC PDP 11 Family." *Proceedings, NeTtional Computer Conference, 1'J7X.*

STRE83 Strixier, W. "**TransIL:ni Bch:464.)r** of Cache Memories." *ACM Transacawe,. on Computer SyNteins,* November 1983.

TR179 Stritter, E.. and Gunter, T. "A Microprocm.;sor Architecture ibr a Changing Wor1d: TN. Motorola 68i.1, **O.'** *Computer,* February 1979.

SWAR90 Swartzlander, E.. editor. *Computer Arithmetic. Volumes f and IL* **Los** Alamitos. CA: IEEE Computer Society Press. I 49(I.

TAISA9I Tabak. 1). ,4 *dviinced Microproce.....cor.s* New York: McGraw-Hill, 1991.

TAMI83 Tamir, Y., and Sequin, C'. "Strategies for Managing the Register File in **MSc:** *Trans: actions on Computers,* November 19.S3.

TANE7S Tanenbaum, A. 'Implications **of** Structured Programming for m;iehine Architecture." *Communeimions of the ACM,* March 1978.

TAN U.99 Tanenbaum, A. *Sorticuored COmpeuer Organization.* Englewood Cliffs, NJ: Prentice Hall, 1999,

TH03194 'fliornps.on, T.. and Ryon, **B. "PowerPC: 6211 Soars.** *Byte,* November 1994.

THOM00 Thompson. D. 'IEEE 1394: **C:hanging** the Way We Do Multimedia Communication.s." *1 FIT,k;* **2.10).**

T190 Texas Instruments Inc, *5104A CT880 Family Data ...1,1nruial. SCSS006C. 1990.*

TJAD70 Tjadcn, (3., and Flynn. M. "Detection xnd Para I lei Execution of Independent **I ns.1 ru et lulls.** *IEEE Triin.s.iu:tionN on CO.F7Jpener.v.* October [970.

TOMA93 Tomasevic, M,, and Milutinovic. *Tiu?* Crichc *Coherence Problem in Shored...lrlemorl. Mulfiprocess o Hardware Sofeaion.....* **Los Alamitos, CA:** IEEE Computer Society Press, 1991

TOON81 Toone, H., and Gupta, A. "An Nrchi tcci nro I Comparison of Contemporary i6-Big Mieroproue.morb, !t4icio, May **1981.**

TRIEIH Triehel, W. *haniarn Architecture fir Software Developers.* Intel Press. 2001..

TI.JCK67 Tucker, S. "Microprogram Control for System/36(1" *IBM Sys terns Journei.* No. 4, i967.

TUCK87 Tucker, S. "The IBM 3090 System Design with Empfnisis on **the Vector** [-4ici Li Ey. *Proceedings, COMPCON Spring* **:87,** February 1987.

VOEL88 Voelker, J. 'The PDP-8." *IEEE Spectrum.* November 19KK.

VOGL94 VogEey. B. "R00 Megabyte Per Second SysLems Via **Use of Synchronous DRAM."** *Proceerlinp, COM PCON '94,* March 1994.

V ONN45 **Von** Neumann, J. *First Draft t.J a Report on ?Ire EDVA* C. Moore School, t hiiversiiv 0C Pennqlvania, 194 . Reprinted in *A nitth OR the Ifi.ytory of Coinpeain,ET,* No. 4, 1991

VRAN80 Vranesic¡ Z., and Thurber, K. ¯Teaching Computer Structures." *Computer.* June 1980.

WALL85 liNallich, P. "Toward Simpler, Faster Computers," *IEEE Spectrum,* August 1985,

WA1,L91 Wall, 11 "Limits of basi ruction-Level Parallelism." *Proceedings. Fourth inlerataional Eonferencr f.P02 Architectural Stipport fbr Programming Langiu'iges and Operating Systems',* April 1 Q91.

WANG99 Wang, G. and Tafti, D. ¯Performance Enhancement on Microprocessors with Hierarchical Memory Systems for Solving Large. Sparse Linear Syteinz-.. ¯ *Imernational Animate!' Superrompufing Applications,* vol. 13, 1999.

WARD% Wm-d, S., mid Halste ead, *R. Compurarion Structures.* Ctumbridge, MA: Press, 19911

WEIN75 Weinberg, G. *An Intro(hiction to General Systems* Nu-0; York! Wile,'. 1975,

WEISM Weiss, S., and Smith. J. "in,',truetion Issue Logic in Pipelined Supercomputers." *IEEE Tramacrlons on Computers,* November 1984.

WEIIS94 Weiss, S,, and Smith, ***J.*** *POWER and PowerPC.* San Francisto: Morgan Kaufmann, 1994.

WEVG(.11 Weygant, I. *C.lusters for ffixto•* viodabiiity, Upper Saddle River, NJ: Prentice I tall, 2001,

W1-111T97 Whitney, S., cl al. "The 501 Origin Software Envircmincnt mid Application Performance." *Proceeding,i, CO, PCON Spring '97,* Februar,' 1997.

WICK97 Wickeluen, L ¯The Facts About FireVilite." *IEEE Spmruin,* April 1997.

WILKS1 Wilkes, M. "The Best 1V:13.. to Design an Automatic CalculaLin@, Machine." *Proceedings. Manchester University Computer Inaugurol Cott,ference ,* July 1951

WILK53 Wilkes, M.. and Stringer, .1. ¯Microprogramming and the Design of the Louirol CircuilLs in an Electronic Digi tal Compuler- ¯ *Proceedings of the Cambridge l'hitinophical Socif,ly,* April 1951 keprinied in [SIEW82.1.

W11.190 Williams, F., and Sirven, G. ...Address 2IT]d Li t Register Separation on rho. M68000 Family," *Computer Architecture* Ne.ws, June 1990,

YEH91 Yeh. T.. and Patt, Y. "Two-Level Adaptive Training Branch Prediction." *Prorecdinp., 24.th Annual International Symposium i.t?* .41croarchirecirtre, 1991.

/HANOI Zhang, Z.; Zhu. Z. and Zhang, X, "Cached DRAM for ILP Processor MernOrV ALV.:M+ Latency Reduction." /EEE *Micro,* July-August 21301.

INDEX

A
Absoluic
  arithmetic oliera i ions, 34.1
AbliOlUte Scala bIlity
  clustem 663
AcceSs. *See also* i mu-niiry .:Lceess (DMA); Dynamic !and ons-aixe.ss memory (DRAM): Nonuniform memory access (NUN At R ndiun kleCeSS *ilumory (RAM);* f01111 LlielTIOry aix.ess (1..1M A)
  proees§or
    rwn le.vell memories. 129
  Sequential
    memory, 98
  system
    OS. 240
A eu.Mit (.14,!vjvc:
  direct, 190
  qeci wntial. .190
Access efficiency
  two level memories, 1.34
Acce:ss melhod, 97. 98
  units of at (Ri
AcceSS
  disk performance, 171
  mem ory, [4]M-103
Acetiss lo files
  controlled
    OS, 240
Access to I/0 de.yi.cc
  OS. 24'
Accotinling
  OS, 240
Accounting infor malion
  prficess control block, 252
ACCLIMULA 1[9]E, 685
AuctimulAor (AC), 21. 55.
  number, 334
Acknowledgment
  link layer. 227
Acknow led gine t gas)
  li nk laycr. 227
A :M
  web site. 14
Active seco
  cluster method de,icriplion.
AD *bus*
  r'ww data crans(cr, Al)!)
  instruction, 484
  opeodes, 332
Adders
  71.9
  32-hi t
    c4.ins trucc ion. 720
  c ,mnal eirtaiiis,
    717 72.0
  imptern enia Lion, 719
Addition
  block diagram of hardware, 2%
  aoHting-point a rilhinetic, :;15,
  Limos COMplerneitt, 292-294
Addrcss. *See alsof.....olumn* address sellect i CA5): Control addre5;, rej4istcr (CAR): Ivlemory addres.4 regkiu (MAR); Row address select (RAS)
  ha se. 26]
  base-rogimer. 387
  and data pinii
    [3]1C1 idgnal Eines.
  ii nd data signals

Address (rom.)
Intel / 1185. 5.91
dec(Riirig, 7[1
effective. 275-277, 384
go-14.1".11Rn]. 612 614
En; L it iiriStTUCI. i IL., 613
granularity, 397
ruciion
util 'hit M. 335
'ze ro- 135
lih ein-
Pentiu111 memory ITta nagemc ht. 271• .273
higical, 261, 262, 269
machine. instruclion. 337
no:A sequential
[.S]- I 1, 614
number
machine instruction. 334-336
physical. 261. 262
register. 385-386, 415
indirect, 3546
110, 52. 53
269
Add rcss;1111c memory
811hclivide.d. 266
Addrusqal* units. 97
Address LI pp3·13.4 01., ti I 0-1'7111
A di.lrcRs citicula don
instruction eyck wtaws
instruction, 57
operand. 57
Address cycle
dual
PCI command, 85 87
Address cNicissi(5n
physical
Pentium cc Mt rot rcgisier..1146
Address field
single
branch corsirill logic. 612
microinstruct i On, 611
riyo
branch co ril]] I OgiC ()I I
microinstruction, 610-6)1
Addressing. See *ai.vo* Relative addrcssiiig
absolute, 391-395
branch, 393-195
direct, 384-.385, 399
displacement, 386-387
litS ITiodiril4, 384
indirect. 385, 393-395
indirect indexed, 392
PowerPC. 193-3915
SM 65!.)
.stack.
141chiniqiius,
Addressine, mode, 3S2118
algorithm, 384
illust 4tCd, 383
MIPS
nI hcsiAng 01h4'r addressing modes. 4911
number, 396
operand, 394
Pentium, 189-392,
ealculaiiciii, 390
Pentium 11, 391
PowerPC,. 392-395
RISC!, 477
SPARC
synt hesizing other actldre:'ishlg modes. 498
A41,11.mo latch
IILCI WTI [
Intel 8085, 589-594
A titin.n..;l .atch Enabled (ALE), 594
Address lines, 7(1
chip log]c. 144
A dd.req:6; inodi fy
IAS computer, 22
Address 451 111C la
starting 11ILIt ML Fry
cache line, 1111
Address rvirige, 397
Address recoptit loll
VO, 202
Address A•cicciion signals
microinstruction. 611
A il4Ircss **6]A::**
!dtt1tiiEi. 404
Address !.,pace
Pentium II, 2.110
Address translation
P42n1 i um memory. 274
[3ovrerPC: 32-bit, 276
Address valid control line, 74
Advanccd Load Address Table (ALAI)
I A-64. 558
Air gap
size, 170
ALAI'
I **A-64**, 558
ALL, 594
Algebraic simplifications
Booleim c:c1.51-05Si0rl, **701-702**
Algorithms. 24¼4
kic.1dre$sing mode. .384
isooth-s. 300-304
cicample, 302
Dijkgra's, 374

Al rds register
Pentium 4 instruction-lcvel parallelism, 526
Aligornent cilu'ck
EF LAGS register, 442
Alignmcnil mask
Pentium. Goiltroi register. 444
Allocate
P42111 i LL1TI 4 instrucLicm-lt-ocl 1r;irollLs[i: in, 525-526
Allocution or hits
ins' inn length. 396
Arithmetic awed logic unit (AL 41)
American •landard Code for informalion Interchange ASCII)
machine instruction, 338
Analysis
lioolean algebra, 694
AND

A ntidependency, 515-516
Application registers
I A-64 inslrucii el n set, 56:5
Arbitration
PCI, *87 _89*
SAM, 650
Arbitration pins
PC1 signal lineir. 81.. 82.
AIhitration sequence
link layer. 227
Architecture.. Svef 1r 6'0 Thus. Hichite.d > iu.4 1 A-64 architeciure; Scalable Processor A rclii• Ricturc .(SPARC)
Feised tiLlpErpipvii ric
RISC, 489
channel 110.222
(18(. 465-466
computer
cliistot, 667,668
definition, 4
studying. 11-13
Flewlen-Packard PA-1 ESC, 542
Si370.5,681-682
IBM S/390. 311, 682
Mlivt vector
organization. 680-082
infiniBand, 229-230
Intel 1/0 modules. 233
layered protocol
link, 231
network, 231
physical...231
transport, 231
loadistorc
PowerPC, 392
parallel prw.:esscir
I II N.1)11 0.111V, 646
processor
superscalar implem entat ioii. 506
rcduced instruction set, 474-1S1.
teaching computer projects, 741-744
vector
instruction set, 685-67
vim Neumann
concepts, 51
web sites
Computer Architecture Home Page, 14
Itannun processor arailleVillf c, 5'69
PoiwcrPC ;:titilitectuire, 44
related to computer organization and architecture, I.4
Arithmetic
binary floating-point
IEEE standard. 322
cornpuiei . 2M-325
floating•point, 284,313-324
normaliZatioll, .317
si nifiGaild alignment. 317
subaction. 315
zero check. 317
I AS computer, 22
infinity, 322
hunger, 291-3(77
Power VC instruction and dcscription, 363
logical instructions
(BM 117.8) vector facility, 686
MMX instruction and description, 360
operation name arid description, 344
Pepliuni instruction and description, 356
twos complement reprscniallion
cellarargerislics. 287
web sites
compuler. 324
floating-pOirtt, 324
Arithmetic and logic unit (AL. 1_:). 9,17. 284-285
con *figarat* MO&
'LE 8832,634
control fields
IBM 3033,626
control tii **Plah,**
CPU, 41.3
immediate
MIPS, 487
inputs and outputs. 285
Arithmetic instructions, 334
MIPS
immediate, 487

Ariihrn.ctic instructions *(cum.*
3-operand *anti* R-type, 487
PowerPC„ 395
SPARC:, 497
Arithmetic operations
C: fill.: aeli4ins, 343
decrement, 344
Iluating-point .11.11rnbors, 315
ineremc.ni. 34.1
LIS-11624
1112:gHt12., 344
Arithmetic shill 'KO
operation. 347
Array iirnecssor, 680
vector computinicm, 674
ASCU
machine ipsiruction, 338
Assembler. 36.6
Assembly code
lA-64, 556
IA-64 architecture, 552
A:;:sembly language; 364 366
format
lA-64 archiiecture, 548-5.517I
Association fur computing Maehioery (ACM)
Special litrereht Group on Compiler Architecture.
web Rite, 14
Associative laws
lloolc.tan algebra,
Associative nuipping, 112
example. 114
Agsocialive mcnnory, 9.8
Asynch(011OLLS bus Tjpern tiO !is. 77
Aqnehronous link layer. 227
Asynchronous subaction, 228
Asynchronous timing
1)LIH clusign, 76
Autoinclexing, 38s
Autoindex registers. 39S
Autinnalic register renaming
lA-04 pipclining, 561.
Auxiliary
CD-ROM,

SNIP, 649

B
Bitek Ward branch
transfer-Or-ContrOl instruction, 35tJ
Bandwidth requirements
peripheral iechnologies„ 40
BkIsc.
a.ddre5:s, 261
decimal system, 734
- with displacement mode
Pentium, 391
ii nd divlueement mode
Pentium. 3112
mode
Pentium. 191
regisler addressing, 387
Seal ELJ 11113C wit ll displacement mode
Pun ti UM, 392
hu perpipzlinc architecture
RISC, 4S9
Balch. 243-246
niultiprorarnming vs. time sharing; 150
OS. 241
13CD
Pentium lypes„33(1
MI Laboratories. 149
Berk cley RISC' COETIputer§, 469
Biased reprseniaiion
ll oating•point number, .108
13i-t2rtclian fashion, 330, 376 380
Big-cridian fashion, 330, 376-380
Bin aTy ri ddilion., 717
truth. tables, 718
Bimirs, and decimal
conyersion, 7.15-738
lW) complement
venal. brut. 2g1
Binary coded ri,acimitl (13CD)
Pentium data types, 339
Thioary digiis, 738
Binary tliViSii311
unsigned
FloWchHri, 305
Binai y floating-point arithmelic
'FEE !,.tandard, 322
Binary tlOaLing-p{51.TIL TVTITV:Ktilalion
fEEE standard, 312
Binary forum. 284
Binary inputs, 595
Binary integers
unsigned
example of dis..ision, 3C4

unsigned
flowchart. 29S
lu.ildwave i 311 plementation, 297
Binary number system, 285
[3i]1 operation
Stack riperano° dcsk:ription. 372
Binary outputs. 595
Binary point, 29[1
Binary system. 734-735

number ..3..sturits. 734-735
4-Elit
adder, 71 .)
in t cgurs.
alternative representations, 288
8-hit
par:k W. re.gister 726
32-bit
adder
con ii ructioo, 720
lloating-point format. 30g
formats
expressible, 3 W
64-bit
icnsi cm pills.
PC:1 signal lines, E. g3
128•bit
bundle
l A-64 architecture. 546
Bit field
Pentium data 339
Bit lengths
convcrting. 289-290
Bit orderin
Block. 98
cl t transfur
hos systems, 79
multiplexor. 222
sire
two level memories. 129
Block format
CD-ROM, 1.36
Hoard oprizol
'L 1 8800, 629
Borilt l*irs ph ra, 694-696, 717
associative laws, 695
v..s. 6415
Derviorgan:s theorem 695
digital logic, 69.1-696
teclliiique:t., 694
Boolean equations, 699
Boolean c xprciisi on
Igitiiraic simplificarions, 7U1 701
ICarnmigh maps, 701
Quinc-McKlai-key tvibles, 705-709
Boolean functions
canonical form. 703
i mplimien tali on, 699-709
three xrtiria bles, 699
Boolcan if16(clIclii3M, 334
Sil'A RC:, 497
oolcan operators, 695
C11)111 '5. algOri LtTTEI 3{Kt-304
example, 302
twos complenicni ./(11
Branch. *See* vivo Conditioinal branch
haul:Wan.)
transfer-of-control instruclion. 3.50
d cab rig with, 431-438
cl elayed, 437.-43g, 4t;4., 48
forand
transfer-L¦r-clliiirui instrucdort, 350
history table, 436
strategy. 438
loop buffer, 433-434
615
normal. 484
opiirni4ed delayed. 484
pipeline
s trenins, 43t -433
qkip instructions
transfer-of-control tostroction, 350
Unconditional
T AS computer. 22
BUIFIch addressing
Potiiicr pc, 3413-395
n•mi, Dontrol Icigic
single add.rei-.- (112
liultk..!!: I
✓ariable km nat. (] 1 3
Branching fluids
11:3Nel 3033, 626
Branch instructions. 334
$l ttt,rateel, 35.0
M1 E'S 487
SPARC, 497. 499
Lran.if4,1r of ci tr{]] operation, 349
Branch u ,ricTtteL1
Pov..erPC' dc seri p tit ni, 363
Poi.wrPC ins.tr actions
Branch prediction, 434-437
flowchar1.436
high-performance pipulined machine, 518-519
It an inni, .56F;
Powerl3C, 534
processors. 39
state diagram 437
fB E) n ch prtAxssinp.
F wt 601. 531 -532
uni[
Branch. registers
l A-Nliiii-tiudion sot- 563
Branch strategy. 435
delayed
R]SC machine. 51S
superscalar mat hine& 518
Branch targot
pre tetch. 433
Branch taro boffi,„•r (113 TV}, 524

B.SN
Yv11³, 654
!i24
Buffer egistkir
control, 603
B-unit
I• -64 a rehitl2cture, 54.5
Bus
conuno
SMP, 650
data. 70
cIai *md.d,* 69
.2 serial, 224
hierarchies, 72-.74
internal
CPU, 388
1¹0, 72, 220
operation . 7]
Select DR
inirroskso.n.mwr, 632
ti me-shared
SNIP, 650.
li niing diagrams.. 02
Bus arbiter
bus design. 75
BUS rbil rel Li on, 212
Bus architecture
high-performance, 73
plr!„%ica I realiza
traditional. 73
Bus
example, 73
Bus controller
Mari design, 75
Bus control lines
clock, 71
tcrEupi. ACK, 71
internipt request, 71
1:0, 70
memory read, 70
memory write, 10
reset, 71
tram:fed ACK, 70
Bus cycle, 75
Bus thin irkinxrel- uy-pcs, 78
Bus design
aeyrichronom Limning, 7b
data transfer iype, 78
elements. 74-79
Inc thud c5(arbirralion. 75.
synchronous timing . 75
iming,
Bus detached DMA, 220
Bus grant
bus conlri.il lines. 71
13us id! st t us, 594
Bus integrates DM.A• 0, 220
BUS in1creonnQctinn,
scheme, 70
Bus &Tura Lions
synclirciriOnS
ti ming, 70
Bus request
cuntnil lines, 70
lius StITICIL1 re, 6⁽). 72
PCI, 81
PDP-8, 33
Bus•swiiching network adapwr (BSN)
S'1 P. 654
Bus system, 69
block data transfer. ⁷9
cal h rn*rn ry ciyntr011iA . 72
clock line,
control signals. 5./¹6
tpi 413
LANs. 72
read cycle, 77
St '51, 72
W.A.Ns, 72
wri cs..c/e., 77
li.us types
bus design, 74
Blls metchi ng
write through
cache design, 118
Bus width
bus design, 77.-78
Byte mulliphaor. 222.
Byte ordering, 376-380
Byte string
Buni aim dald iypes, 339
PowerPC. . 341

Cache
disk. 103. 120
cxicrliul, 120
FIFO . 117
ill mtrald. 472
internal, 123
ltanium. 568
v.v. large register *file.* 471-473
LF1J. 117
]heal
read miss. 000 1102
MI:. 115
mapping, 107
direct, 107, IN

mentors,.. 96-135, 462
olluiroller
i1lutrated, 103
principles. 103-106
structure, 104
systems analysis and teaching, 743
MIPS
data first, 494
data second, 494
nurnhur (5l
cache design., 119-120
on-chip. 120
PN1riurn 4
trace, 524 525
PcFwerPC, 125
siza, 107
processors, 108
SMP, 654-656, 656, 743
split vs. unified, 1211-121
USE bit, L L7
virtual marnory, 266
C-ache coherence, 656-659
CC-NUMA, 671-673
VI ES] fro icical
parallel processing. 656.6.63
SMP, 651
software soludoos, 657
Cache-coherent NL'MA (CC-NUMA)
loci, 67(1-671
organization
illustration, i72
pr of : mid cons, .673
Cache consisierkty
Ll-L2. 663
Cache .0csign
Oxus watching
wrilc through, 118
eIerrioins, 146-121
hardware transparency, 118-119
119
multilevol caches. 119-120
noncachcaMe memory, 119
N21ilacI2rn42n.l.i 4mrillirns, 115

*rile policy. 118-119
Cache disable
Pentium control register, 444
{.'ache T.TRAM (CDRAN4). 1.54, 1.59
Cache hit rate
S13'10
SM.!' configuration, 656
Cache line
TriHin memory 1.3145e19. rissigned, 110
starting memory address (E Hock.. 110
MkIL42s
1%.1 ESI, 660
Cache management
instruelion and description
Pentium. 357
POW4.7 PC, 1.63
kriptation
modes
Penlitim 4, 123
26ii
Cache organimtion
ebratilaristies, 471
Fully a3sociative, 113
illustrated, 106
1L-way 2tisociative,.116
Pentium 4, 121 123
execute tIniLk. 12.1
morriury subsi,..m.e.(11, 121
out of-order e.xecut ion logie, 121
Cache read operation, 105
Cache. support pins
EiCI signal lines, 81, 83
Calculate imerandS
pipain.int, 425
CALL
instructions. 351

service
I rocess. 253
WR11 E. 59, 03
Called
ealI insiructiii.ns, 351
Call instructions
invked, 351
nesting of procedures, 351
Pentium interrupl proecssor, 448
win:cc/11re, 35]--+54. 466
registers, 351
star[tit called proeadura,
top of iitaelt., 35E
.Call procedure
al location X
insttuelii.m.s.
Call return
behilvior example, 131
Pantium instructions. 35'5.
Canonical form
13rFolean function, 703
Capacily, 97
cAlcrmil metro. Fry, 97, 164– 191
11.1 11 data iransier
RAID 0, HO
marriory, 14)–[
CAR, 603, 611. 614

CclrEy
C01113110IL fields or Flags, 417
lookahead. 719
{';lily iii
8832, 634
CAS, 144. 147. 156
CASE rtia.hinu ihstruclitni, 464
Causo•and-effect dependencies
bluing diagram, 93
{'AY, 166, 167, 185
C.'.:13.E lines
PCI huh data transfer, 85
CC-NUMA, 670-673
CD. *See* Compact disk (CD)
CD-R, 187
C.'1?1 A.3,1, 154, 159
CD-RON1.. *See* Compact disk read only mcin• (try (CD-ROM)
1:144, 1.S7
d L'scn.pI]otl, 184
CE, 146, 156
Central control unit
SNIP, 652
['antral pnicesi‘iiig unit (CPI„:), 9, 11. 25
Into Center
web 14
Intel 8085. 591
ini‘truction set. 330
interconnection, 9
with internal bus, 588
ioralBaal ructuro, 414
register, 373
rritiLline instruction, 331-332
scructure and function, 4.12-457
syslem bus, 413
transigiir counl
growth, 30
Cr‘1, *See* Current frame marker (CFM)
{:FrAiisiitg, 679
Channel architecture
inpullouiput (I,'O), 222
Characters
I RA
LC WI I MI, ZOO
encoded. 1{19.
machine instruction, 333
inosri q instruction., 479
unpack. 340
ctor-slring tioning
big-endian processor, 37S
Charles Babbage Institute
web sites, 45
Cheek hit calculation, 153
Check bits
1.51
Checking instruction
I A-04, 554
Ch&.k pointing
dusters. 069
Chip
description, 29
Chip enable (CE)
pin
chip oackag.,ing, 146
signals
RDRAM, t56
Chip logic
serniamduct.or memory, 143-144
Chip piickiving, 144-148
Circular buffer
organizalion
overlapped windows, 470
Pentium 4 instruction•level parallelism, 526
Circular
SPARC. 4W
CISC. See Complex instruction set compuler (C1SC)
Clock
bus control lines, 71
c tell, 75
li ne
sytern bits. 93
processor control. 585
signal
Inning tlitiguarn,q3
Clocked S-R
sequential circuit, 722
Ctuster, 663-669
memory, 1192
parallel processing, 663-669
v.s. S M
superior pric•iperformanee, 664
Chisic CT .3TC: hiic:ci LITE, 667, 668
Chu:Ler Coniigur L [OlLS. 664-666
Cluster methods
active seeondary, 665-666
heik.rits ariLl limitations, 665
passive standby, 665-666
512pafaiit WA'S, 665-666
servers connected to disks, 665-666
shirt: diisks... 665-666
shared disk, 060
shared rit5thing. 666
C1N
RD, 185
Code example
conditional branch

llowe r 533
Code segment pointo
PCnliuln inli:Trupt proi:essor. 448
Code si reliniye
RISC I,475
Color Plane Reprt.serilabern
image compositing. 362
Ciilumn address selcct (CAS). 1%
pins, [47
signals, 144
jowl 1,:irc 'Ails, 699 - 700
Command decoding,
ll0.202
Connie(Ciai cs ! nip art;es, 22-24
Committing
itNiruction,
Common bus
SMP. 610
Communication, 197
pathway, 69
ti ming diagrams, 92
C:OMMUtatiVe I2.W$
Bocilioan algebra, 695
C:c full)
IA-04 architecture, 541
Compact disk (CD), 184
dt.scription, 184
operation. 185
ComplE1 disk read-only memory (CD-ROM), 1g4-1S6
advantages, 185
block formai, 186
description. 1g4
disadvantages. [86
storage illosimitpd, 188
Compact disk reeardable [CD-k), 187
Compiiction.
Comp.arch
USENET'. 14
USENE:F. 14
Comp.arch.storage
1,.:SENFT, 14
Comparison
MMX inslruction and description. 360
Compatible cl.anputers
family characteristics. 3] 32
Compiler-based coherence mechanisms. 657
CLimpiler-based register optimization, 473474
Completeness
pi)p- 3c1l
Completion queue entry (a)Iii). 231
Complex instruction sct computer (C1SC). 474-476
architectures. 465-466
characteristics, 463
instructions
motivation. 543-544
rilicr4Ipel =NS. iT, 653
vs. RISC characteristics, 479 46]
Compound instruction
[WM yectoi facility., 684-685
vector computation. 683
Corrip.para]lel
USENET, [4
C'fimputer
iiLquire and appivcidlion, 11
evolution and performance, [5. 45
fri rnidy characteristics, 31-32
history. 16-36
Computer architecture
definition, 4
studying, U-13
Computer arithmetic, 284-325
web sites, 324
Computer components, 50.53
i.op-level view.. 53
C:ornputer elements, 28
Computer function, 53-67
gmymii4)319, 24
Computer instructions. *St.* Machine instructions
.C.:C1111plik2(r11e111ilry N'Y'S LCIF1
oyurvicw, 9(1.• 103
Comptur modules. 68
t:oinputer opera t
Computer organization. 647
definition, 4
taxonomy. 68t)
CcUTIp43Lcr Sci(!ticc Student RUtiCRIEGC Site
web site, [3
Computer system
lave' ..i and views.. 239
Computer technology, *4-5*
Como] E.! na tcd asynchronous subaction.s. 228
C.ondit ional branch
code e.:tamplc
I'Llwei.1C. 533
lAS computer, 22
(161TUCL
transfer of control operation. 349
instniet jou pipeline operation, 427
inicroseql.tencer. 632
Conditional jump
Pentium conditions, 359

Condition codes
Pentium. 357
registers, 416
(.'ondit i on register
PowerPC
interpretation (Fr bits, 454
processor, 450
Consistent order
big-endian processor, 378
Constant angular velocity (CAV), 160
illustrated, 167
RD. 185
Constant linear velocity (C11V)
RD. 185
Context data
process control block. 252
Continuous-field simulation
vector computation, 674
C1ONTROI.
keyboard-handling, 216
Control, 6-7
buffer register, 603
functions. 7
instruction type. 333
1:0, 205
IR, 54
microelectriirtics,
Pentium processor, 441
of pr(icessor, 583-.594
status registers. 412, 414, 416-419
Control address register ((AR). 603
ITINf 3033, 61.1
microinstruction, ti l
Concri rl characters
fRA, 2(s)
machine instruction, 338
(.1ontrollcd access to files
OS, 24(1
Controller
1/0 channels. 221
Control lines, 70
Control logic
1.'0, 198
Control memory, 602, 603
organization, 602
Control registers, 412, 414, 416-419, 603
Pentium, 444
C:•ntrol signals, 587
active control signals. 587
data paths, 5 8n
example, 586
PO, 198
micro-operations. 587
processor. 576. 585
processor control, 584-586
ft'4.ml control hus, 585
co control bus, 58.5
read, 594
system bus, 586
Control spe culation
TA-64, 553–554
1A-64 instruction. 542
Control transfer
Pentium instruct ii in and description, 356
Control unit, 9, 12
CPU, 413
decoded inputs, 596
implementing technique. 576
inputs, 594
logic. 595
microarchilecture. 603
'nodal, .58.5.
operation, 575- 597
hardwired implementation, 591-597
micfo-operations, 577--583
of processor, 583. 594
organization, 616
proecssot , 576
Conversion
binary and du.cirnal, 735–738
CPU actions. 343
instruction and description, 360
operation name and description, 347-348
Coprocessor instructions
V1IPS, 487
Cores, 138
Cost
En ern ory, 99-1(13
si milar or identical
family members, 32
vs. size
two level memories. 133, 134

PowerPC processor. 450
transistor
E1P11.: growth, 30
Counters. *See also* Program counter (PC}
disables read born time stamp instruction. 444
microprogram
TI 8800, 630
register
Ti 8800. 630
ripple
sequential circuits, 727
sequential circuits, 727-730
synchronous, 728 730
CPU, *See* Central processing unit (CPU)

CQE, 231
(.'ray supereouiputers,6.79
Cross-hatching, 473
CorRml frame. mark.ʻ r (cfm)
I A-64 architeourc, 568
rugiswr, 566,5H
E A-6,4 iiistruction set, 5C3
(.:11.u rnt wind ow pointer (C'WP)
poinls, 469
SPARC!, 495
(WV., 469,495
Cyclu!•Lii221ing
DMA. 219
Cyclic shift operalions, 347
Cyhnders.370

I)
Daisy chain, 212
Daia
CD-ROM, 186
EA), 198,202
inovum4:n1slora.gc Incl processing 240.242
Data bits
layout. 151
Fiala bits, 70
width 7.1.)
Data each): first
MIPS 494
Data cache second
MIPS, 494
Dar. ebannels,
Data communications, 7
Data Flow, .17'..?123
2 Ethly3i
processors, 38
fuich cycle, 422
indirect cycle, 423
inlc•rrttpi LyLl12. 423
Data formatting. 165• 167
Data lincs,
Dala marrying
P.A1[)le\'CI () array, 179
Data movement, t5-7
insiructitm Iypu., 333
tnicroeloctronies. 28
Penriorn And deSCripti011 356
na LA operation
instruction state,
Datil urganjAilitlith, 165-167
Dam paths. 587
control sigmils,
Dalai pins
signal lines, F!.1.82
Data proccssing, .f1-7
itimruLtion type. ".%33
I R. 54
ruicroolc.ctrouies,
DATA REALIY]inc, 216
Data registers, 415
Data .'"ignals
1ntel 591
Data 'q)·'
D:11;
I A-64, :·o•si •
I A-O4 insiruction. 542
Data storage. 0.7
mstruction 3713
microel Inn [CS 28
Data Storage. Magazin'
siies, 191
Data stream. (45.-647
parallel processing, 646.
Data throughput rates. 231
nat4 transfor
CPE; actions, 343
I AS computer, 22
MN1X instruclion and de.seription, 2.60
operalion name. Jrld description, 343-344
PC'] bus 85
type
bus design. 7t
D17R-81.)RAM, 156
Debugging exteiisions
Pentium control rcgisler, 446
DEC. *See* Digital Equipment Corporatidn (DEC)
Decimal and binary
conYcl:SiOn.
Decirnal+AS(.11 dumps
big-Indian processor, 37g
Dccirnal sys[4:m 734
De&.odi instruction
pipelining, 425
D.....c.o(11,! N, 605
co inNnattonal circuit, 711-712
four inputs and sixieen out puts, 595
3 inputs 8 outputs 712
IDeo-de stagL
Intel.9,4)48(1, 43'9
DeLoile .itage 2
Intel 8W% 439
Deco& unit
Pentium 4 cache organizarion 121.
Decremcni
.i rithinc:tic operations, 344
Decrementer address [tub
lutwl 8116:7 . 5:59-51 94
1)1-7,1) Lode, 352

D elav branch. 43738, 4M
striveR
MSC' rntichine.
superscalar 518

Delay skit, 4 4
Demand paging, 263-264
1)eMorgiin'5 theorQM, 697
applying, 709
Rook an algebra, 695

implementation. 713
Denormalized numbers. 322
IEEE:. 754, 323
Density. 167
Depgrideneies
effect, 510

Boolean aigebro, 6(11
microinstruction. 611)
Dem
IA-64 archireeture, 549
Destination register
TI 8832, 634
Device *controller,* 204
E)E1\ SEL
PC'[bus data tran$fer, 85
D flip-flop
S424.[L1124111}11 circuit., 722
Digital Equipment Corptiridion (DEC), 2.5.
See also PDP-8.; PD1u-10., PDP-11
Digital 693-730
Eloolean algebra, 65./4-696
combination circuits. 699-72i)
gates, 496-698
sequential circuits, 720-734)
Digital ver5alile disk. *See* Digital video disk (D VD)
Digilal video disk (I)VD), 187-188
description, 14.::4
Digital video disk recordable (DVI)-R)
dewrip t ion, 184
Digital video clik rewricable (DVD-RW)
description, 184
Digital video disk ROM (DVD- ROM)
storage illustrated, 1f[4],g
Dijk$1.ra's algorithms. 374
Direct-mxes,'i device, 190
Direct addressing. 3/[4],-1-385
PDP- **10. 3co**
Direct eneciding. 62()
microinstruction, 616, 619
Dircciiim nag
Ei•I_A(3S registei:. 442
Direct mapping
1:84114, 1.07
cache organization,
4!xumplo. 111
1.12ClirlitiLlt, 111-112
Direct memory access (DMA). 67. 69.9 ,,
1%, 204.216-220
block diagram, 219
configurations, 220
fienctiim. 217
input, 206
Directory prillocols. 658
Disabled imerrupt, 64
Disa hies. read from time stamp counter
RDTS(') inslructiOri, 444
Discrete component, 25
Disk. *See* in'so Compact disk (CD); Digital video disk (DVD); Rahintlani Array of Indeimrsden Disks (I))
cache. 103. 129
data layout. 166
doubie sided. 169
166
floppy. 171)
formatting
exarnple, 167, 168
'Winchester, 168
Iransfkl.r liming, 172
layout methods
comparison. 167
rmignciic, 164-17.1
movable-head, 16[
nonremovable, 169
optical, 96
products, 184
port a bilily
disk system, 169
removable, 169
shared
cluster method description, 666
'Singh; large cripensive, 175
single-sided. 169
types. 170
Wind:sc.:4 cr. 170
track format, 1448
writes, 103
Disk di itig. 201
components. 16[1,4],
pa rtimel ers: 171
Disk perforiiiiinee
access time. 171
pirromieb.irs, 171-174
rotational delay, 171-.173
rotational latency. 171

5121 li me. 171-17?
sequential organization, 173
ti ming comparison. 173
rrollsfer time, 173
Disk system
hcad. mcchanis.rns : 16[4])
head motion. 169
physical characteristic. 169
plsileoi, 1.69
sides, 169
Dispatch unit
Pinsier PC: 6(11..523-531
Displacemenr
ii.ii..1re55irtg, 336-387
Pentium, 406
mode. 391
Distribuiive laws
Boolean algebra, 695
DIV
opcndes, 332
Division
nmiing-po in !, 317-320
integers. 304.313
DLTapc
web sites, 1911
1)1.:L'ape drives. 189
DMA. S[,i,' Dircot memory acCi2 'ti I DMA)
Double data rate SDRAM WDR-SDRAM), 156
Double-ertor-deie.ding (171= D) code, 152.-153
Double sidcd disk, 169
Do ll 11 ll word
packed
MMX, 359
unsigned
PowerPC'. 341
Doughnui-shar2d lerromagneric loops. 138
DRAM, 33
DRA pn rlti
'1[9]] MOO, 630
D1t11 ports
TI *MOO,* 630
Dual Address Cycle
PC] command. 8547
DVD, 184, 187-188
DVD-R, [34
DVD-ROM
storage illustrmed,
DVD•RW. 184
Dynnmic
defined, 139
Dynamic branch strategks, 435
Dynamic parlitioning.
effect. 260
Dynamic percittage
operands, 466
Dynamic RAM (DRAM). 38. 96. 138
cell, 141
chanteteriNticx, 139-142
controll er
PC:I, *79*
Gyolluii(in. 39
organizations
advanced. 154-15'9
bends, 40

EBCDIC
machine.. inslruction. 338
EDP, 35.5
Eck ert-Mauchly Computer Corroralion, 22
l- DVAC, L7
REPROM, 140, 143
Effective. address. 384
PowcrPC memory management. 275 277
FE-AC iS register., 442
8048(2.. *also 1nte* 80486
pipeline, 439
8-bit parallel register, 726
82C59A inixrrupi cii ntrolltm.. 213*, 214
8847]loafing-paint
1 l 88CO. 629
8818 m ic ros4,!qutpno.::t. *.Vve edisfy* Texas Instruments Kg 8
`11. *8&X1* 629
Eigin Queens Problem : 556
3832 registered ALL:. *See al5r)* Texas lnslru-mcnis 8832
T1 88W. 62[9]1
881)1) 5D.B. *See* aisr? ¯l¯1.n.w... f ri4trurrieo s 13800 eomponenk, 627
Electrically erasable **PROM** (EEPROM)
descriplion. 143
memory tyiv characteristics, 140
Etectronic Discrete Variable Compuler (EDVAC), 17
Electronic Numerical integrator and Com-puter (ENIAC). 16
Eike pHi,b
A-6,4 architecture. 550, 553
EV1M.5. 446
Frurny MM X State (EMMS)„
Ern LiiiIL si 1 (137
Pentium control resister, 444
Endinn Triaps
example- 378

Endiuniress
concept, 377 .380
property, 377
EN1AC, 16
ENTER instruction, 355
EPIC
544
Epilog phase
1A-64 pipelining, 561
EPROM . 140, 143, 146

logical functions, 345
Equal
C{,1111111. **Hi 1** ·1321 Cl; 1)T **Nag,**. 417
I.'.clti.tl sip partitions. 2,59
progrtimmable rcac•only incinory (EPROM)
description, 143
memory iype characteristics, 140
package 146
Error-oorrecting code, 149
function., 149
web site. 159
Error correction
increase word length, 151
techniques. 138. 148...153
Error detedion rc.sponsc.
OS, 240
Error rr:porling pins
1'C.1. signal lines, 31,X32
ESP. 355
Pc.mium infcrrupl. profxssor, 448
Events
5³„'**(11.1c.n.ces,** 92
Except ion
Pentium, 447-.449.
register
oriicesrior, 450
Exclusive
NIES(proinc-„l,
write hit, 662
Exclusiw-OR (X03.1.)
logical funciions...145
Execute instruction. 54-55
Fu.ciition
CPU's insiluetion 420
cycle, 54
data llow. 422-423
description. 57
rnicr45-Clpera t ions, 581
uction
pipelining. 425
Intcl 8006. 439
inicroin si.ruct ion. 61)3
processor control, 584
suEuencing
RISC advociies, 464
units
Pafilom l **Carlv**:: organization. 1.?...1
Ex pi·men t over 11 4³r
amitin g-poin t. **3** 1 5
1⁷.xporsnE undc3·11cpw
floating-point. 3]5
Exponent value
Roating-oonn. noiriber, 3178
Expression evaluation, '374
1⁷ xtqm.3cd. Codtg.3 DI2cirrpo.1 nil:AN:ham:42
(:ode
machine instruction, 338
Sf1.2.1I1cd. Ni.2 Lk pH jill v21' **S** p)
Pentium interrupt processor, 448

Pentium' control register, 444
External cache, 120
EN1C1·1l;1l] **cicti** iixs. 197-201
block diagram. 198
External interracc. 223-23.2
External mentor,. **C.; I paCi V, 97, I** 64-1 9 1
External memory ivsteins
;val. Sites, 1)1
External nonvolatile memor;, 102

Fa ilback
667
E i i r, 666
Failure management
clusiers, 666-667
Fairchild. 34
Fairness intervals, 266
Family e.oraxpi. 462
computer characteristics, 31 32
Roil] 'okra ***nce***
SNIP, 653
]etch, 53
CP1.r9 imirtiction cycle.. 42f)
and execute in ri, 54-58
Fetch cycle, 21
dala Ilow, 422
micro-operations. 5?24 SEA
scgmncc or events. 57')
Fetch,
CPI. - 413
Foch
ltanium. 568
Fetch instruction
CPU, 412
pipelining, 425
Fetch operand
425

Fetal overlap. 424
rclai utl il
4 cache organization. 121
Fic Id-programmable logic array, 715
FIFO
cache, 117
FireWire, 1.96. 224-228
configuration : 225
protocol stack. 266
wrial 1'Pus, 224
subactions, 22
web sites, 233
Firmw2Ire..601
First-in-first-out (FIFO)
cache. L1.7
Firsi time unit
fetch cycle, 579
Fivu-Itagc pipeline
80486, 439
Five.•state process model. 252
Fixtal-head disk, 168
Fixed partitioning
example:, 259
represcrodlion. 290
Fixed-size partitions : 257
Flag control
Penfiuni dexcripiion : 35?
Flags. 416
Pc nlittirk-proc.;:Nbi ir, 441
processor con ue a 5&5
Flash memory
duscripiiiin, 143.
memory type characteristics, 140

SM.P, 6.50
Flip•tiops
sequential circuits. 720-721. 725
clocked S-R, 722
0, 722
J-K, 7 23-724
Floating-point
addition and subtraction. 316
division. 319
execution units
Nrilium 4 inslroc(ion-]cvel 527
multiplication. 31 8
allle rati - , 315
Protium data types, 339
PowerPC
description, 363
instructions. 3153. 531-532
regiswoi
IA-64 instruction set. 563
8847, 629
unit. 528
Pentium processor . 441
Ploating-point 313-324
binsry :
IEEE standard, 322
norrnalizolion. 317
i nilicaiisd aligniwilt, 317
subaction„ x 1 5
web Rites, 324
era eltb2k, 317
Floating-point numbers, 284
aril hrn l is 9per rs t l u rts, 315
biased representation. 308
density, 311
c.s.rionent v,Ilue, 308
IEEE 754, 314
mantissa. 308
signifleiincl, 308
Floating-point repreFR nta lion. 284.307-313
IEEE 754, 312
binary, 312
principles. 307-312
Flop t Tit .51H Ws Antj .LOrit i't11. register (FE/SCR)
PowerPC processor. 450.
Floppy disk. 170
Haw dependency,:509-511
FORK, 676
Format, 382-408. *Set aho* Instruction format 2P4
block
C1')-R1013,4 :
data, 165-167
disk cxamplu...167-168
expressible, 3l0
1A-64 architecture
assembly language, 5418-S50
instruction,
registc.r. 5:66
lEtNel base-16, 331
IEEE 754, 31.2. 313
IRA conirol, 200
memory-management
l AS computer. 19
Pentium', 271
PowerPC, 2.75-277
micn,inxl ruclirin
horizontal. 62]
1BNil 3033, 626
I.-SI-11 : 624-625
'texas 1 nstnanticnts 8800. 628-629
filint2ric dsta, 340
PowerPC register. 453

exprussihle. .110

Fcrrn iat *rcemi.:(*
floating-point. .308
vriTinbic
brailch control logic, 613
VLSI. implementation
RISC. 177
Winchemei disk track, 168
Forward branch
transfer-nl-ciinirpl instruction, 350
4-hit adder. 719
4-bit int.'egcrs
al Ler n Ei Ve. ioprLsmtaI iin S,, ⁱ
Four. way pipelined timiltig, 4#i3
FPSCR
PowerPC proccLisor, 450
Fractions
convert from GIVOMal 03 binary.. 736-738
Fram, 261
PCI his data transfer. 85
pointer, 3.5.5
Free frames
allocation, 261
Fully associaLi've Lathe 511;i1111V.}1(I C in, I 13
Fully nested
interrupt-drive 1.'0, 213
Function. 5-10. *SIT 01.1.0* E3[F]lean functions,
Logical functions
conipuier, 53-67
control, 7
CPL', 412-457

DMA, 217
cncoding
microinst mai a-s, 620
error•correcting code.. 149
hash, 264-265
instruction sets, 329-380
1/0, 66-67
mapping, 10⁷-115
rmcroprogrammed control coil. 61)4
(Fpcni ing 5ysturn. 238-241
iecruiternents
control unit, 583-584
ⁱwktppirtg, 251
timing diagram, 93
F-Li nit
lA-64 architecture. 545

Gaps, 166
aeknowledgment, 227
air, 170
interrecord, 189
.164
saktoiion. 227
Gateq, 696-698
and chip, 29
comptiller,
digital logic, 696-698
logic; *O?*
NAM'31, 699
NOR, 698
Ocncral• purpose registe rs, 415
LlupicLicFn
twos complement integers. 295
Global variables
regir.;1 1' Iilc, 471)
Gradual underelow, 324
Grant signal, 8749
Graph coloring approach.
illustrated, 474
Graph ec it c wr i ng problem. 473
C i.raithica1 symbol, 699
Ground Chiss)., 144
C;roup5. of lirscs
ti ming diagram. 93
Guard hits, 320

1-1
I [aliwort]
sigitec1
PewerPC, 341
un-; igrsccl
PowerrC. 34]
Halted
proeess slate. 252
Eiarriming, Richard, 149
Hamming Grid
cudO. 149-15 0
Hamming SEC DEC cod°, 153
[lard disk diivu. poi a.inclurri, 17'1
Hard failure. 14S
Hard/soil microprogramming
lll iCrOill $1 11.101011. 616, 619
Hardware approaches, 52
Hardware failure
iiiELTYLLOI
Hardware solutions
pruc.piNing,
Hardware. transparency:
cache: design. 11 '-119
Hardwired control mill_ 607
flardwircd implementation, 594-597
llard i program, 51
Hash. funcrion, 2:64.-265
Hash lables, 264-265
1-I CA
InfiniBand, 229

Header
CD-ROM, 186
Head mechanisms. 116
disk syst....un, 16
motion
disk sysrem., 169
kletvleit-PackuRk PA-RISC architecture. 542
Hexadecimal digics. 738
HeXadecimal notation, 55. 738-73c)
1..ligh
clusters fi43
High-level languagc..s (NIL), 464. 475
relative dyuitmic ercqucncy. 130
weighled rclativu ciynamic Ircquency, 465
suppori .638
i.'entiorn instruction imd dcscuiptinn, 356-357
fligh-performance computing (RFC). 106
Hit ratio
I wo rnemories, 134. 135
H[-[-. *See.* High-levellanguaffs (FILL)
Horizontal microins1roction, 601-602, frkb.
f:C1 9
for
Horizontal microprogrammine, approach 605
I Iosr channel adapier MCA)
infinif3and. 229
I IPC, 106
Human readable, 197

I A-64
web sites. 569
I A-64 architecture, 541-569
application register. 5 64 565
instruelion 11.31111. K. 547
instruction set architecture 563-568
instruction type. 546
(Erganixafii]1,, 544-546
organi?..ation illustrated. 545
predication, epccolatior3, i'ind software pi pet **46-563**
112.P.i Ste format. 566
I AS compulor
iddress modify, 22
arithmctic, 22
condiiiomil branch, 22
data transfer, 22
Expanthtd Strueture, 20
instriaction'set, 23
memory, 19
Mortuary Furm2Ls,
operation flowchart, 21
structure, 15
lific.unditipnal branch.. 22
I BM 360;91, 433
I BM 700
web sires, 44
I BM 700.'70000
exam*. mciribers, 26
I ITIv1 3033, 614
CAR. 6i4.
cnnirol ...iddress register. 614
ckesit;n. 61/
microinslract i I) rL ciintrel fields, 626
ruicroin8trneriOn exccution 625.
microinstruction fnrrnat% 626
saimmeiug and branching fields, 626
vector racilit y. 680-687, 681
uriihrnefic and logical mWuctionN, 686
registers, 684
I J3 A4 7094. 25
configurat loll, 27
111M se...16 format, 31.1
113M Si361.1, .;(1-32, 6N) 614
characteristics, 31
JI M S..'370
archil•o tire, 5, 68I-682
data transfer instructions. 343
data transfer operittions exampl, 344.
.5....5•90
architecture 311., 682
procc ss or
%%;15 te:s.. 324
SM1 configuration
hi1 rate, 656
web rites, 324
vector arehiteclurc
organization, 680-682
vector f acilii y
compound inrdruciion. 684-685
1BR. 20
Identification flag
EFLAGS mgiAk;, 442
p control block, 252
ldL!Dtity s
IlooIL;in algebra, 695
nialized numbers
utt...cc. 323
inLapretation, 314
101.111a1S.
pmrdnretexs, 313
standard. 284

IEEE from.)
floating- point representation , 312
web sites. 325
Technical Committee on Computer Architecture
web site, 14
It'-thou-eke instruction
I A-64 architecture. 550, 55.3
I mmediate
addressing. 384
Pentium. 406
mode, 39
I MPACT
inch .-.it...,.,.. 570
I mplications
instruction execution, 467
Increment
richinetie operations, 344
Incremental gri lwi h
SNIP, 649
Incremental scalability
clusters, 663
Incrementer
address latch
Intel Sag5, i 1-594
L....arnaugli maps, 707
Indexing, 387-388
Index 388, 415
Indirect addressing, 385
P45WC.IF[i]C, 393-39
indirect 420
data **flow,** 422. 423
micro-operations. 580
Indirect Encoding , 620
microinstruction. 610. 619
!oared iristeled 3(12
Inductive vr.ritethiagnetoresistive read head ; 165
Enfinilland, 196 , 229.-232
architecture, 229.-230
communication prolo.col steer. 232
HC 22[h])
links. 22[1], 231
operation. :[5].30-232
router. 2.30
submt. 230
switch , 22Y.
switch **fabric. 230**
1 (7A , 221}
web sites, 233
Irithmetic, 322
Infix notation. 374
Inrormat io a separator
I **RA, 20**[1]]
[11-crrdei iss i ■
with in•ordcr completion. 513
with 111.11-111-4)111c1T CIATTITIC1 14 511-515
data strrIbe
sccAut.ritiirl eitcuibi, 725
techniques, 20e.
(1.[1]**O),** 7, 9, 17, 1[h]Th 2 r4
Hde.,11 eis TngiWE 52 , 53
buffer register, 53
bus. 220
channels, 204
architecture, 222
characterktic... , 221-222
and processors, 220-222
commands. 205–200
cornpotiertN, 51
controller, 204
syslem hJ3, 72
(.']'u iLLtlrlrl4. 343
device
machine ii i rnrliam, 332
device data rates. 203
function, 6.6-67
i mitruclion s : >6-207
interrupt class. 58
isolated. 207
iricaiel structure, 203 –204
modules:, 201 204
block diagram, 204
control :]1tJ finking, 201
data buffering, 2{12
device comrnuniwaliam, 211
error detection, 2513
generic model illusiratud, 19:7
inizrconni24:1ion , 6S•
I'D channels, 221
processor communication, 202
L ye r acions
LI.S-1 1, 624
niirne and cl.ocriptiiiii, 348
trtr ur from **n1E:mon:**
iniercoancction structure tramik..im., 69
purls
similar or identical, 32
Frivilegc.•
EI-L (38 register, 442
processor. 204, 221
proces.s0r
interconnection structure transfers. 69
queue
procesg. 255
read
his ci

Read and Write
PCI command, 84
scheduling. 2511 25.6
status information
prno4ss control block, 252
wehniques, 205
wrile
has co atm] lines, 71]
fns registers
SPARC, i95
niStialte 4)fr.jcctricol ;,1 6(./ FEL!.ctroffic5 Etigi-
nurs. *See* I EEE
/T15.1. nictiOn
Cale LLiatitn1
instruction cycle states. 57
utiliyalion.. 335
Instruction buffer register (211
InN1.1114.1iOn L. 54,420.423,592
code
micro-operations, 5112
DMA and interrupt breakpoints. 219
Flowchart. 581
illustrated. 420
wilh inlc.rrupts,
micro-operations, 5112
slate. diagram. 57.65,331,421
Itvitruc[icin wwcwiryn
characteristics, 463.467
M]PS, .194
Instruction tc.tc1·1
instruction cycle stares, 57
sccond half
MI PS, 494
Instruction fetch first half
MIPS, 494
Instruction format. 312. 395-404
I A-64 architecture, 546-547
MIPS, 489
P DP-8.398
PDP-10,399
1[3] DE)-11,4M
Pentium. 404-406
illoslnitcd,105
1 oweri[3]C, 406-408
RISC 477
SP:' 1-Zt 498-5011
Instruction issue policy
paralleliiirri, 512-514
Instruction length, 395
Inbtruclion-leycl parallelism 508.511-5'12
Instruction up.TiltiOF11,1C01.)dili g
instruction cycle states, 57
pipeline. 412.424-440
Intel 811486,440-449
M II[3]S, 489-494
operation
131-dr1Cli, 427
timing diagrams. 426
PowerPC 601,531-532
RISC, 479
F wo-stage. 425
Instrneii on pipelining
speedup factors, 4:.12
Instruction pointer
TA-64 insiruclion 5c1,563
Penrium interrupt processor, 44.
Pentium processor. 441
flu...traction prcfc1c1·1, 424
Instruction prefixes
Pentium. .1{].1
Instruction I ep,ister. 29. 51,54
data flow, 422
i mIructioti exii.euiion, 416
machine instruction. 332
micro-operations. 578-579
process(FF **CCMIZOI, 511,5**
Instruction representation
machine instruction, 332-333
rnslructiol
characteristics and functions, 329-380
mapping
IA-64 architecture, 548
MIPS R series processors, 436-488
opuTatii1n5..., 342
identical
family members, 31-32.
SPARC, 49.6-498
vector architecture. 685.687
Inslruciion stream
parallel orocc.min E.. • 46
Instruction types, 333
in.511·41c(i on window, 515
Instruction word,]I
Integer
unwed Iron, decimal to binary, 735-736
Pentiumti data types. 319
Integer arithmetic. 291-307
Powel PC. rucLiorS and dE.Acripricui, 3.n3
11111.C•p.:1: processing chip
TE 8800,629
Integer representation, 285
Integer unit, 528
Natiurnproo2ssor, 44]
Integrated circuits, 25-33
iv.E1-triiilo gy,
Intel. *See aisr,* Itaniunr,l'entium
PCI. 79-80
RDRAM. 156

[Mel 8085, 589-594
CE[3]I.1
block diagram, 590
external sign.pli,
O1,r1 insiruainn
liming diagram, 593
p.11 conriL..iiration, 592
Intel 80480
CIVC(Pd.:: slags 1, 439
decode ,itage. 2, 439
c%ccution, 43')
insiruLtii3n pip-Mine, 440–449
pipeliniitg, 439
..wile kick, 43')
! mei 82(155A
interrupt controller, 213, 214
pp3gTk117101:1hk peripheral interface, 213
illustratud, 215
Intel Developer's Page
web sites, .15
Duel VO modules 1,111.1 architecture, 233
Inrc[microprocessors
5IuLion,
Interactive system
OS, 241
lii[erconnec(ion. xlrUCt ureti, 67-69
1 m:et:face control pins
PCI signal]ines, 81, 82
Interfaces
pcs, 223-224
Interleaving
multiprogramming. 6,4S
lnt.:::rmediate queue, 257
111(ernal butt
C1[3]11, 588
Inti•rnal. CPLI, 413
liiturnak memory, 97
Internet[processor organization, 588-589
1:ricer:nal sintct urc
of oorriputr..7
International Reference Alphabet tIKA),
199-200
collgO1 characters, 21.X
defined, 199
encoded Lliaracters, 199
machine instruction, 33b
111R:rpiei insiructinn
01'1,, 412
inwrreeord gaps, 189
ln[er No, :58-66
h:.tch OS. 245
01E11314es in rni.mnory and registers, 211
classes, 6g
CPU's instruction cycle, 420
and instruction ccle, 59
P1241610111, 447
process, 253
program llow comrol
xvithout and with, 60
Interrupt Acknowledge.
bus Lorifrol
line, 213
PCI command, R4
Tn1Etrrapi Con Fro!
S2C59.A.... 21'3. 214
Inlet 8085, `18:(P-59.1
IntesruEIL eyLIt, 59
data flow, 42.A
micro-operalion5L,
Interrupt-drive PO, FA 204. 208. 2.1(s
drawbacks. 216-220
input, 2(16
Interrupt (Lisa hieldia hie
common fields or flags. 417
froctrrtrin enable flag
EFLACiS register. 442
Interrupt flag
Pemiunn interrupt processor. 4414
Interrupt handler. 62
Pentium interrupi processor, 448
Poivera1, 456.-457
routine, 61
1 nierrupt
design issues, 212 214
Interrupt pins
PC] signal lines,
Interrupt pri essing, 209 •211. 447-449
|. 20°
I5'V'L.I 4:!16
4:-14
lnlerrupi-relai.„..d sign als
1ll4 . 5[4]11
Interrupt requesl, 213.'216
bus contra! 1i[1cK, 71
signal, 59
]11.14'1711p1 return regisl cr
T] 8800, 630
Interrupt testing
SE-1], 615
Interrupt vector table
Pentium interrupl prixxssair, 448
MESE protocol. 659
]natsrr elements
Boolean algebra, 695
Invw1c(1 page table strut.Wm, 26.41-26.5
Iri voked
call instructions, 351

1;0. See Input/output 01.'01
1/0 addiess mgisWr (170AR), 52
1/01AR. 52
UP[., 442
1;0 privilege flag (EOM .)
ETLAGS register, 442
[RA. See: Tulernalional Rckrence Alphabet ORA}
I RDY
1[3CI bus ilaia tran$Ler, 85
Isochrono LIS
link layer. 227
packets, 228
Nulaaciions, 228
fsalaied 207
ltaniuni
1 A-64, 543-544
orga lion, 56S-5M
prefercti engine. 568
proc4issor, 156, 50
564..1
with sites. 5.10. 571)

1 A-64 **aiichltu41** um, 545

J CL, 244
J-K flip-flop
setillen ia 1 circuits, 723-724
Joh. 250
OS problems; 243
Job control language (JCL}, 244
JOIN instruction, 676
.1 0IN N, 676
J• 1A(.3!Iriluirdary bLan pins
PC:! signal lines, &I.
Jurttp
instructions, 484
M1PS, 487
SPARC, 497
transfer 0f control operation. 349
Penlium conditions. 359

Karn a ugh nidpA
Boolean. expression, 701
use, 702, 704
Kernel, 24]
phase
f A-64 pipelining, 561
Keyboardhlisplay interface
g2C55A, 217
Keyboard/nun-111(w, I %
K-way set associative cache UT:JUL/Al ion, I 1.6

IC-way set .1 5rioeialive mapping, 114-115

LANs
system bus, 72
',age register file
PP. cache, 471-173
characteristics, 471
use, 467-17
Last-in-first-out (LIFO)
queue, 388
sEaek, 371
Latency. *See also* Access time
mcmory, 98
LA ered protocol AT'Llhit,:lc[uN
link. 231
rielwork. 231
physical. 23]
transport. 231
1.1 edche
consistency. 663
PowerPC', 125'
1,2 cache
consistency. 663
Pows,TPC, 125
SNIP'. 654-656
L3 cache
SMP, 656
Leading edge, 92
Least frequently used (LF LE)
cache, 1.17
Least recently used (LREJ)
cache, 115
E.FLI
cache, 117
LIFO. 371.388
Linea(
Pentium memory management, 271-273
structure, 273
Line 5i.a-
cache. debign, 11.9
Link
InCini.Band, 229. 231
1 ayred protocol architecture, 231
Link layar
ackiaw]Cd grnent, 227
acknowledgment gap, 227
.11 -ii ration sNiicuec..227
async:hronous, 227
FireWire. 225, 227- 228
i **suchrctr1crur**, 227
packet transmission, 227
.s ubacti on gap. 227

Link 1.!gi41 r
POVeu PC processor, 450
Li tle-c ndian fashion. 330, 376380

a relliteet Lire
PowerPC, 392
cluster balancing, 667
instruction. 486
instructions, 364
MIPS, 487
PowerPC, 363
SPARC. 497
opeod es, 332
speculative
I A-64. 551. 554
Local area net rorks (LASS)
system bus, 72
1..oLalit4 (Fr fcrerwrce., 102
two level memories,
129-131
Web pages, 131
Locals
spARC.
Local scalar variables 461
Location, 97

address, 261, 262, 269
data
machine instruction. 1.5g..-33
gates. 697
ins'. Hit!
MMX instruction and description, 360
operation name and description. 345-347
Nniiiirri instruction kind description, 356
shift, 345
PowerPC irmfruction. arid (3Oc.ription, 363
A]gicil r1104111011S
AND. 345
EQUA L. 345
t•NCLUSive-OR (X0R)..345
NOT operation, 345
OR,. 345
Logical operations. 345
CPLI actions, 343
LIS-11, 624
Loiv terrn queue
pi-(5r.n%S. 255
Long-term scheduling, 250,
251
p buffer
branch, 433-434
illustr{eled, 434
Loop 'Trace
soft warc pipelining e.r; ern ple. 562
Lower level mcmor:r 131
LRU
cache. 115
LS1-11
ecFntral unil.
organization. 622-624
design. 618-619
interrupt testing, 615
Mill. 623
rnin-ain il ruction, 62.4
execution 622 -625
formal, 624. 625
bgiwricing, 614
opcocio mapping. 614
subroutine fAcilit.v. 614
1,1;I inSEILICt lit", 500

141
Mxcll ilia check enable (MICE)
Pentium control re gimer, 446
Machin
L1C.Firiod, 476
instruction cycle, 592
M;i chi n^
L' haracteristics. .330-337
elcmenis. 331-332
s4.IL
Wilkes example, 607
Machine nrga El I L{11011:3
hpeedups, 518
Machine. parallelism. 511-512, 517-51S
fliti(Aine readable, [97
Machine state register iMSR). 379
Powc rPC. 454-456
PvikigndiC.-LOle memory, 34
Magnetic disks, L64-174
definition, 1.64
Magnetic read and write mechanisms.
164-165
189-190
features. 190
M agn c to-opt i cal storage 164
Magneteresistive sensor, 163
Mainframe. 31
SMP, 6.53
Main memory 4,, 52
machine instruction, 331-332
Nel.;11 tiss
floating point number. 318
Mapping. Sf olAr, Direct vloppIng
assoeiatiy, 112, 114
cache. 107, 109

RAID level 0 anal..., 179

(mullion, 107.-115
instructinri sal
IA-64 arch iteet u 548
k-w **5ei** associative, 114-115
opcode
LS]-11, [}-l4
iiLi urn MMX
registers. 447
sel associative, 112-113
scl associative, 117
MAR, *Sc'* Nleinoty iii3dress register (MAR)
MaNk able interrupts
I'c uliuln, 447
Matrix multiplication
vector tiornpulation, 675-676
M Li uchly, E7
See Memory buffer p4..is1crfMAR)
1-Mbyte memory organization.]4!
MC6801)0
i H nip14..!
illustraied, 414
MCE
Pentium control register, 446
Medium-1.21 mi *sk:licd* tiling, 230.2,11
E mcgahit DRAM
ilius iralion. 145
Memory, 52
inlercfmnoution. GY
U0 irtiLifiLt24 symbol?
Intel 80,95 591
Memory address
Pentium 4 instruetion-level parallelism 526
Memory address res.ki **itT (M** A R), 20,52.53
data flow, 422
instructii, n Esc cu l ion, 416
micro-operations, 578-579
fvle nu n. huf ler register (MAR), 20,52,53.
data lIow, 422
itiliruetion execution, 416
micro-operati(M.S. 578-579
Memory capacity, (./9-103
MLAll **!Ty** curd
SNIP. 654
Memory cell
computer, 28
operation. 139
Me cycle time, 25.98-99
Memory Itirinais
LAS computer, 19
Melnoty hierarchy, 96,9g-101
illustrated, 100
Memory inslructions, 334
Mcinoryless circuits. 715
Memory manage me at, 248
OS. 256-.258
Pentium. 357
SNIP, 653
system
lwo level memories, 129
Memory-mapped
1;O, *20a-* 207
207-208
Nlemory module, 52
Memory package: pills and signals. 146
Mcmory point c rS
process control block. 252.
CrTlory protection
hatch OS, 245
Memory read
bps conlrol Iii cs. **'70**
Memory read **And** wri rc
PC[command, 84
kier045ry **sire.**
SIMI] r or idern lea]
family members, 32
Mein (Try s **u** hsystern
Pentium 4 *cache* organizalion. 17:
l'1,1.c.rnory systems
characteristics, *96-99*
Mern ory-to-me mory, 478
WITLOTY () processor
interconneci structure lranstrs, 69
Memory-transfer length, 395
kleinc5ry 1.111il
pc rallel processing, 646
Memory \suite
bus corn rot lines, 70
invalidate
PCI command,
MESE
cache tine states. 660
pputoccil, 65Q-663
slate transition diagram. h61
Mt ihod of acvessing
{Irsiti of dada., 98
Method of arbitration
has design, 75
622-62'3
icrod i agnostics. 638
rVi **Ia{ }E II:2 I TC** mks, 27-2.8
hylicroinstruction. 398.600-60.3
address generation techniques, 6]3
encoding., 629-622
execution. 609,615-62.6
formai, 6(11-6112, 621
88Ik. 628-629
IBM 3033,626
LSE-11,624-615

Microinsiruct ion (cm,!.)
interpretation, 602
sequencing, 609-615
spectrum
characteristics. 616
ierrninology, 618
taxonomy, 616-619
Microinstruction bus (MIE), 6 7 2-623
T.,S1- I 1,623
Micro-operations. 577-583
active control signals, 587
Pentium 4 instruction•level parallelism, 526
queuing. 526
scheduling and dispatching, 526
Microprocessor, 34 36
design and la!,..oni, 4S0
register organizations
example, 41.8
illustrakxl, 419
speed, 37-38
Microprogram, 597.601
advantages and disadvantage;, 607 609
applications. 637-038
description. 600
languai;e. 601
M icrop roj- ram counter (MPC)
'1'1 881111, Oil
Mic r yrrnii arnmed control. 600-63B
basic concepts. 600-609
microinstruction execution, 615-426
microinstruction sequencing, 609-615
'1'1 13800, 627-637
unit, 462,603-605
description, 6(,,K.I
functioning. 604
roi mprognirilincd implementation, 9
Microsequencer. 628_633
control, 632
Texas Instruments 8818,629,63 l
microinstruction hits, 633
MI M D. 645-647
MIPS
addressing mode
synthesizing other addressing modes, 490
instruction formats, 489
R4000, 486-494
R-Series instruction SET. 467
MISD. 645-647
Mitittloisltii
CDRAM., 159
Mnemonic
]A-64 architecture, 549
Mode field, 384
Iviodi lied
MES1 protocol, 659
write hit, 662
Modified, exclusive, shared, .01-
059-.663
M odR'm
Pentium. 404
Module organization
Semiconductor main mcmorv. 147-148
Monitor, 243
monitor coproccssr ir (MP)
Pentium conrrol register. 444
Moore, Gordon. 29
Moore's law
consequences, 30
web sites. 44
Motivation
l A-64,543.544
memory. 670-671
Pentium, 543.-544
Motorola 6800,465
Movahle-head disk, 168
Move Characters (NI VC) instruction. 479
Move data, 'Y
Move instruction, 343
MP
Pentium control register, 444
MPC
Γ I 8800.630
MPY
opcodes, 332
MSR, 379
PowerPC. 454-456
Multilevel caches
cache design. 119- I 2.1]
hicrarL111 '2:-, 72
N•tultiple execution units
l A-.6'1
huge numbers, 544
Multiple. instruction milllle data (MIMI))
parallel processing. 64ii
stream. 645 -647
Multiple instruction single data (MISD)
stream. 6477-647
Multiple interrupts. 64-66
lines. 212
Multiple processor. 462
organization, 645-017
Multiple SI ri,•arris
pipeline: branches, 431-433
Multiplexer
digital logic, 709.. 71]
implementation. 711
input
to program counter. 711

representation
4-10-1, 710
truth table
4-in-1. 710
Multiplexor. 25
block. 722.
byte, 222
channel 222
Multiple zone recording
cl seription, 167
illustrated, 167
Multiplication
fl ukiiin-pc.iiht. 31?-320
twos compleirwilt„ 294-304
Multiplier quotient, 21
MU1.T] P1 ,Y-A ND-ACC LI NIL: LA'l E, 685
MULTIPLY-AN 13-A 1 D instruction, 68,5
MULTIPLY-AND•SUBTRAU1 instruction, 685
Multiply/divide instructions
MIPS, :187
configufations, 224
Multipart memory. 652
SMP, 6.51
Multiproot2ssor, 648
operating system design corisideratiiins, 6:52.
tightly coupled, 649
Multiprogrammed hatch systems, 246-248
Multiprogramming. fa4,s1
defined. 246
example, 247
OS, 242
elements, 255
resource utilization, 2411, 249
Multitasking
&fined, 246
M-unit
LA•64 architecture, 545
TL;K2-M U fl
microsequencer, 632
MVC instruction, 479

N
NAND 698
NAND implementations, 7(19
NaT hit
l A-64, .554
Near pointer
K odium data l yiy2s, 339
Negation
arithmetic operations, 344
integer, 7(11
twos coniplemot, 287
Negative overflow, 309
Negative unkkriflow:
Nested interrupt processing, 66
promdiircs, 352
Nested task flag
EFLAGS register, 442
Ncriting of proccdurcs
call instructions, 3:51
Network
huff-swiiching tidoipi4'T
SM.1[3], 654
1o.ym.L1 protocol architecture, 231
Ica i] ;I nd area, 72
sineli: virtual clusters. 669
New
process state, 252
Next operand reference
11104:hi m: instruction, 331
No sequential address
LSI-II. b14
Nticonned pin, 147
Nonbranching l as[ructions
ulilizalion. 335
NoneitChablt.toeinor y
cache! design, 1.19
Nonmask interrupts
Pentium, 447
NonrcntovithIc disk. 169
Non L11111 01111 17112nm ry ac-ce M
ir70 4r73
lieseripEton. 646
pm; ;Jud Loris: 673
systern3. 644
NOOP, 484-485
NOR gales
use, 698
NOR irripkrmnlalions, 709
Normal branch, 4.0,4
Normalization, 320
Floating-point 4rithanelir, 3;17
Normalized number, 3[19
NOR S-R latch liming diagram. 722
Not a 'Filing (NOT) hit
l A-64. 554
NOT orriLriiiion
logical functions. 345
Not write through
1e..11(iUrri control register., 444
Nucleus. 24]
NtlMA. See Nonuniform memory Hco.....s!, (NtlMA)
Numbers. *See also* Floaiingpoint numbers
A(:, 334
address

Numbers *rcont.j.*
machine instruction, 334 -336
addressing mode,. 3%
binary system, 285, 734-735
&normalized, 32.2
IEEE 754, 323
large Multiple eXeL aim units
l A-64, 544
machine inslruerion, 337-336
iiiictrands , 396
Pentium Ii segmentation, 270
set **4,1197**
Number systems, 734 .739
binary system. 2#S. 734-735
vOnv4.!rling bels.vcerl binary ,r rid decimal, 735..73
decimal system, 734
mai notation, 7311-739
Number word, 1.9
Numeric
Perlt i um processor, 441
NIUMeriC error
Pe.ni roister, 444

O
On-chip each e. 12.0
One-digit packed decimal incrementer
truth table, 706
126-hit bundle
l A44 architecture, 546
One instruction per eyel., 476-47 7
One Icvel memory, 12g.•129
1-11 4 byte. memory organization, ig
.cre OrwrallOn code
Operand. 331)
address calculation
inStruLtiOli cyCle. SL@Les, 57
retch
i nsi roc ti nil c-'i. C14 stare, 58
instruction ,a.xecution. 466
number, 396
reference, 3g2
.l-k l SC advocates, 464
sire
Pentium. 404
store
restriction 42y1;142 state,
types, 337- 339
Off rating
CP1.1 aetions, 343
environment, 6-1
instrueliin exoeutinii, 464-466
types, 7. 3A1-354
Operating system (OS). 238-276
defined...238
Design issoes,
objectives and functions, 238-241
sched til in 251)
tiirnilar **or** identical
fornily incmbers, 32
support, 638
c!,, pes, 241-250
.0.. eti sites, 277
Orieraling System Resource Centel.
well sites, 278
Operation chug. 330
machine insli **uct [on,** 331. 332, 332
mapping
LS[-i1. 614
Pentium, **414**
O perations performud
RISC advocioes, 464
Optical disk, 96
products. 184
rnernonr. 1 64
Optical storage
field
w4,!I'3 **6iics**, 191
technology, 1(14
Optical Storage Teelinology Association
wt. h'ites, 191
Optimization task, 473
Optimized deliiyed branch, 484
OR
logical runctions, 345
Ordinal
Pent ium data types, 339
0 rganiza lion
CC-NUM A, 671.-673
design issue. 99
IBM vector AniliteLLL.Lfe. 661)-662
0 rill ngOil Lily
3'L P-1 399
pDP-11, 401)
OS..S:e,? Operating system (OS)

Mil.: I"(FiL.qui2 neer, 632
C1 Wel)
1).:.',!1". sites, 2711
Out-of-order
execution logic
P ii turn 4.cache organization. 121
issue with out-of-order **corn** p l **L.Lion .** 515-516
Output- *Sti• rfirgr)* Input/output { KO
controls
ruicro.542qt.LNICtlf. 632
dependency. 513

enable pins. 144
select
rnicrriscquencer. 632
outs
SPARC..494 .500
Cuts 3tgi s LV IS
495

Overflow
common fields or Ilags, 417
)verlIcive Wile, 293
twos complement. 257
Overlapping
groups
combination circuit;, 705
multiprogramming, 648.
register windows, 469

Packed BCD
Pei·Ilium data types. 119
Packed d(publcwrird
NINIX, 359
Packodlitupacked
microillStruciiiim, 61.6. 619
Packed word
N.11v'E X. 359
Packet byte
l4! MX, 359
Packet 1rall}i1111!-.!, Oil
link layer. 227
PA DDII. insinmicsrs, 361

Pentium control register, 446
Page. 26]
directory eritly
Peniturn memory inanagement. 271-273
fault, 263
fra nies. 261
Page !11.11-5s14, inable..(PCrE.)
Pentium corstrc pl. re.gis(tyr, 446
Pagc extensions (PSE), 273
1 ' I2 El I ii-1111 eonlrol rcgi.str, 446

2r1.11 '%.. 2.71-277
memory inanagernent
Pentium, 262, 271-273
Power PC, 275-277
structure. 264-266
Paging, 2.6[
operaiing. 267
Pentium control register, 44-4
Pntium II segmentation., 270-273
Parallelism
ALL;
vector computation, 676-675
clusters
application. 667
compiler. 667
coin p a **lion**. 6.67
1A-64. 560
and hi NI X instruction set, 361
organi/ation, 646
recording. 1S)
and st.,rial I.10, 223
Parallel processing, 643-657
architectures
1.:LX{}(14)1TLy, (}et)
c-ache coherence and FA p rotowl.
656-663
clusters, 663-669
definition. 679
mullipie processor organizations. 041-t47
t10111.111itbrrrIl ITI MT11.1 ry awess, 670-673
symmetric niultiprocessors..C47 656
sysinins LyTic, 04.5 –647
✓ector compuiation, 674-687
Parallel registers
80quential circuits, 725
Parametric computing
cluster5. 667
PA-RISC architecture. 542
Puha] rcinainder. 304
Partitioning, 257-261
PaSSiVt. scand1·5y. 665
duster method descriplion. 665-666
Patterson prograins 61:100, -165
Patlerson study. 466
N7. Sep Prognmi uouriter (pc·
pch
Punlit311.1 control register, 446
PC I. See. Pe1'iphcr iI component inlerci 11CCI.
(PC1)
I'D 32-33, 397-399
bus structure., 33
evolution, 33
instriicifon formai, 398
['DE'-III. 1.14.)—14X
eOmpletviiess. 3 ¦;9
direct addressing, 399
instruction format. 399
orthogoinalily, 399
PDP-11, 465, 614
family, 622
instrucLitin, 400-402
example, 56..57
imitruciion format, 401
orthopmality, 400

Pcc
..irchitc4cLure,.., 568
R2.1111LI n1
ddru.ssia 1711(PdC·. 389¯392
calculation, 390
addruss sire:, 404
y progrm·ri. 556
base mode. 391
c.ii1Prourn instructions, 35t%
conditiou codes. 357. 3:71
control registers, 444
displacement, 406
displacement Irk di 391
btise, 391
based sC2E.2d lade N. 397
base with index, 392
SCHIV-'d irklex, 392
evolu tit)] t, 41-43
exception, 447
table, 449
imMediate, 406
mode, 389
instruction prefixes, 404
interrupts, 447. See Aro Pentium interrupt
processor,
maskable, 447
nonmaskable, 447
vector cable, 449
memory management, 357
ModRim, -104
opcode, 404
operand sin, 4U4
rcgigi.! Opi.!EAT1(1 IM rick, 389
register organization. 44(t.-141
rotative addressing. 392.
segment override, 404
SIB, 406
wch .5itcs, 44
Pent hill' itioris
cvnditional jump, 359
ST4.Tcc iitstructions, 359
Pentium oryntrol register
alignment mask, 444
GICE1C disable., 444
debugging extensions, 4415
extcruion type. 444
MC L. 446
MP, 444
not write through
numeric error, ...144
PA H, 446
paging. 444
PCE, 446
PE, 444
PGE, 446
physical address extension, 446
PSE. 446
PV.1, 444
task switched. 444
VMP., 444
WI). 444
Pentium data lypes, 339-341
RC!). 339
bit field. 339
byti.lswirs.e, .339
Floating point. 3'5'4
I1 teger. 339
nLtkir iloirtio'. 339
ordinal. 339
purled BCD, 339
unpacked 1-SC!). 339
Pentium 4. 520-527
block diagram, 122, :521
13:1:13, 524
cache.
race, 524-525
cache operating modes, 123
cache opera Licit)
modes, 123
cache organization, [21-123
do;ode Unil,]21
execution units, 121
utrit, 121
fetch unit, 121
MC1110.11' subsystem, 121
-(11-01Nkr 1.:NU'L ninon Logic,
121
unit execution, 121
drive, .525
front end. 521• 525
generation 4.51 micro-0[1s, 521-524
instruct inii-level parallelism
register, 520
alloed e..525-526
chrcular buffer. 526
!loafing-point CM:CUL n ti 52.7
inlegcr rcgi4LC r ritu.s, 527
memory address- 526
micro-operallions, 526
irlicro-operations queuing, 52.6
register renaming. 326
ROB, 526
scheduling and dispatching, 52A
state. 526
7)43_544
out-of-order execution logic, 525
trace. cache: fetch. 525

rracv cache next instruction pointer, 524-525
I'eti tiuttr II
addressing modes,
addrChN Space:, 269
control registels, 445
EFTA GS register, 443
management
hardware. 269-273
parameters, 272
motivation...54'3-544
:iegment t ion, 269-270
paging. 270-273
RI'L. 270
segment number, 270
tube indicator, 274)
Iranslation lookaside buffer. 273
Pentium instruction and duscriptic.th
arithmetic. 356
cache nianagernitt, 357
cwarn] transfer, 35I
data movement, 356
nag E(PEICTOL 357
FILL suppCirl, 356-357
logical, 356.
] roti.tulion. 357
scgment I.02,i4Lur, 35?
siring OpQralionl, 356
1st231Linin inNiruclion format, 404-406
illustrated, 405
Pentium interrupt procesiir
CALL_ r 448
code segment (CS) pointer. 448
ESP, 448
interrupt thg. 448
interrupt handling. 448
inrerrupt vcctor table, 448
1P. 4-48
procu.i.sor-det ected exceptiiHiN, 448
programmed eweptions. 448
trap flag, 448
P'211tiiim memory inanagorne•ni
address linear, 271-273
ti ddress translation, 274
addre:ih i nInN]ation mechanisms, 274
formats. 271
linenr address. 271 273
page directory entry. 271.273
page !able 0111try. 2.71-273
segmen1 descriptor, 271-273
segment suleciim 271-273
Peislium
instruetions„35.q
instruction set, 36(1
rtigiSters
Lordrcil, 446
ppin 447
tecliliologli.. 358
PentiuM rILLISIbE fOrirldtS
Pentium data iypes, 341)
Pentium op:..!ralioli 355• 36:1
Pentium pipeline operation, 522-523
Pnlitim Pro
MCA i Vr1 tiOn. 543-54.1
Pentium processor, 156
Lonirck1,441
flags 441
floating-point unit. 441
instruction pointer, 441
integer unir, 44]
numeric. 441
registers, 441
segment, 441
status. 442
tag word 442
Pcrforniance
balarii 38-41
designing 37
riterrign14.,
monitor data register
IA-1y4 instruction set 563
RAID, 182
Performance Counter eila (PC'E).
Pemtit1111 Control register. 446
Peripheral, 7,11)7
device. 197
Peripheral component interconnect We]).
bus arbiter. 87
bei weem iwn masters, 88
bus dam transfei
MWSEL. 85
FRA riet1-7.. 85
l RDY. 85
lurnarnund cycle. 84-85
commands, 81-85
conliguratiMIS
emimpl.Q. 811
read commandit
intLrpre teLioll, 85
read ors2ralion. 16
signal hoes
ddress and data pins. 8/.82
arbitration pins SI 82
64-bil extension pins. 81,83
data bins, 131, g2
error reporting pins. 1 82
in(urCacx control pins, 81, 82

Pori ph era! L4)11111311 ent interet311nect *(conl..1*
interrupt pins, 81. 83
JrAGIlti-}und[;Iry sc>In pills, 81,83
mandator)., 82
system pins, 31, 82
web site:, 89
Pfra
I A-64 archilccturti... 568
PFS'
architecture., 568
uppli.4,7H rcgistet, 566
PC1E.
Ventiurn control rogimur. 446
Physica1 address, 261, 262
Physical address extension (PA E)
pcndiurr, qintroll rctimer, 446
Physical characteristics
data storage, 99
rnagrietie disks, 168
Physical dedication. 75
tisicit I layer
\V ire, 225-227
Physical layered protocol architticturc.; 231
Ph ysiou I records, 189
f[3]h icai types
or rne mon!, 99
P u ele rise Ili, 359
Pin layouts, 714
Pipcline, 462, SKr,. *egisn* iris(ruci ion pipeline
A I.L.[]
vector computation, 676-678
L1i(1.11T110.1iC register remain; iv, 561
branch
multiple streams. cI3 I -433
[12.c Fpenantis, 425
decode instruction, **425**
d **in, 42**[4]
description, 424
effects, 483
cnli rici rig, 49 I
epilog phase, 561
execution instruction. 42.5
fetch instruction, 425
fetch operands, 425
l A-64 ...Li allied lire, 546-563
Fel 80486, 439
kernel phase, 561
rnachi **hi;lnch** pte.dietion. 518-5 [9
•13 perati on
acrii i..6M-679
.eoriditiona1 branch, 427
Pentium, 522-5n
wilt., in, fi71--679
r- Li1tliz.atitto. 484-486
pc Homan Lc., 430
Pc Pir 60 I, 529 530
10000, 492
rogular iiistfiretiorts,
RISC:, 482.486
Ian 510
six-strict C P1 I instruction. 428
stage, 492
strategy, 424.3[1
wrire o.perand, 425
Pixel, 359
PLA, 711-7]6
flatters
sysitml, I(19
P1.1 **CO Ithlpiral.** ions, 224
POP operation
371, 372
Ptirt, 68
PC}S
fi ar m, 700
intplententation, 701
PosiriVV4.iverIltiw, 309
Positive underilow. 309
Postritt notation, 374
p(55lisIdxing, 388
1'clkke a VC:, *"127* 534
addres6irig modes, 3q2-395
cache organization. 123-125
data types, 341
cvohilia0,41 -43
Ian Lily. 43
floating-13$_1$ 1in1 sLit us and cuilDIO1 register,
452
inscruction formmts,
iltustra1tA, 407
internal caches, 123
interrupt table, 455
michiric. state register, 456
Ilse MOP!: management
parn Inc wrs, 277
me rtioky-in ariagemen
hardware. 273-27
memory operkiiiki iiddriassing niodes, 394
oper:ition type.s. 364
mple Dr upctralxsnS, 363
processor, 450-457
processor summary. 43
register formals, 4.53
1.1741„tr-V regisLers, 451
web sites, 44, 45
Power PC 60 I, 527-531
kilo& diagram, 528
pipeline. 530
51.rili:thrc, 529

PowcrPC. 620, 532-534
Power.PC 32-WE
translalion, 276
Inc wiry - rnima gemuni formats. 275-277
PowerPC 04
hlock diagnirn. 124
Ppl
A. 6 1 architecture, 5.1.s.4
code,
[A•r-1.
Predic:.ied execution
[A-64
architecture, 550.•553
Mil ruction. 542
Predicate: regisrers
1.4 -64 instruction set, 563
Pi eilication
lA•o4, 546 .563
ratitcvture example, 552
pipelining, 561.
Prefolch branch target, 433
1'142142Feb engine
Itaniuln, Sir'
PMincicx.ing,1'45F5
Previous fundion suit41 (PFS)
IA-64 architecture, 56,4
kill! I i ca I ion register, 566
Priority
process control block. 252
Privileged iroiltticticiry,
hatch OS, 245
Prncialitra I dependencies, 311
Pt ileedury ry 41111LMI., 467
Procedure call
instruction execution, 466
instructions. 351-354
Process control block. 252 253
Process data, 7,1%
CPU, 4L3
Processing unit (pu)
parallel proceqsing, 646
SMP, (153
Prou:s5 migration
clustets, 669
ProceSSOT
ACCOSs
two level memories, 129
tirchitecture
superscalai iirtpluTrIentkition, 506
cache sizes, 101..4
categurize,..1, :330
characteristics, 41U
evolution, 39
detected exLivo ins
Pentium interrupt proi:EtiMa. 444;
functional requirements, 5Th
idon Li ficTs
[A-64 instruction set, 365
interconnection, 69
EaconisecLioir :it mauve transfers
to 110. 6)'
Co rig:TT wry. 6')
1k
110. 54
itiOnor:....
organization. 412 414
Processor status register (PSP.)
SPARC, 495
Process states
}4}1{31-ll-{1.1.1-71 'Wheal IL; r, 2.5t
Product of sum (POS)
form, 700
implemen IN Li Or), 701
Program,
in hardwire, 52
interrupt class,
in softwaN. 52
timing
host 110 wait, 62-63
with interrupts. 6263
lime, ITO wait, 63
without interrupts. 62 63
Progrtim Lc}L1111 N§) 7'0. 53. 9*,7
data Flow, 422
instruction cxceution. 410
micto-ormyalioN,
multipiexci input, 711
multiplexers. 710
process control block, 252
Program creation
OS. 2-40
Program execution
attributes, 249
.C(FrE· Iil U.2 rSt elein unI.s, 577
example. 56
05, 240
Pr{tigrant inahle Ionic ;1r (Pl.A). 715
combinadon circuits, 713...715
cs.artip[e, 716
Programmable HOM (Pl-l.t)M)
ds, NCHI ition, 142-143
inerthir y 1y pt characietisLicS, '140
Programmed exceptions
Pentium interrupt processor, 448
Progyammed [10. 196, 204-2C8
drawbacks, 216-.220
input, 206
Prottrant status ward rPSW). 210. 412, 417

Program voltage (Vpp)
chip packaging, 146
Prolog phase
1A-64 pipelining, 561
PROM, 140, 142-(43
Protected. mode virtu Di interrupts (PV1)
Punlium (anti ri 11 1-44il.c:r, 444
Protection
Pcntium instruction and description. 357
PTIAE:12 l inn .1yin1.1112
Pentium control revile r, 444
PSF., 273
rentiu Lli CO alto] regiqter, 446
PSC. udoinsiniction, 365
PS] J 'W insiruction
MMX. 35[1,]
PSR
SPARC. 4[4]15
PSW. 210. 412. 417
PU
parallel procei...Aiii1. 646
SMP. 653
PEJSPE
stack operation, 371
description. 372
Pushdown list, 388
stack, 371
pv1
Pentium control register, 444
Pyramid computer. 464

Q
op
I A-64 architecture, 549
Queuing diagram representation
proonsor sch4d tiling, 256
Quiet and signaLing NaNs. 322
Qi.kt NaN
opera s, 323
Quine-McKluskey method
firs1 slags, 707
last stage, 7(118.
Quinc-McKluskey tables
Boolean expression. 705-70[1,]

R3000
pipeline enhancing. 491
pipelim stages, 492,
superpipelines, 4[1]..6
R41.11.10
insiruclions, 488
MIPS, 4F.k..4[1] 4
5uperpipelines. 493
Radix
rn. 734
point, 285, 2%
RAID. *See* Rudundaril Array ill lmicpcndord.
Disl,:s (RA ED)
RAM. See Random access rrwrnnry (RAM)
RFirribils. DRAM (RDRAM), 154. 156-161
structure. 158
web sites. IN
Rrthdorn access memory RAM): 9t5
Characteristics, 139
rricmury lype chArricieristics, 140
semiconductor, 138
sites, 160
Range
Lwos complement.
RAS. 144, 147, 1%
RCA
'1'1 6?.0
RC B
'11 630
RC2-RC{)
rrlicros(24.[LmErcbr, 632
RDIZAM, 154. 161
RDTSC instruction. 444
Read
assignments
reinforce .vonccpts, 743-744
commands
PCI iiiturprciation. 85
control signal. 594.
cycle
bus systmi. 77
depe.adeney. 515 -51.6
110, 205
:MS. comr4)1 linus. 71)
e F11 0 Pr'
bits control linos, 71)
P(.'[ill 15d, 84
microinstruction.
Open! (CH1
PC1 . g6
signals
RDRA M. 156

SDRAM, 157
Read-after-write, 79
Read hit
paralld. processing, 662
Read miss
local cache, 660-662
R4.1.[I-modify-write. operation. 7[9]
Read-only memory (ROM)
;ombinational circuils. 715

rTh2rnory lypc characteristics.
{r4-bit. 71g
truth tablc. 71?
types., 142-1 ,11
Ready
prucess statc. 252
Re 81l7.i1 Li on of corripuiers, 6a7
Runl. memory, 2i3
RECEIVE operation. 211
Recording technoiogy
web **sites,** 191
Rcducod instruction set architecture, 474-481
character iinics, 476-479
Reduced instruction set computer (RISC), 5 : 461-501
approach
Uti Li on, 54.-544
based superpipeline architecture, 489
characteristics, 4f3
CISC, 500-501
machine, 473
delayed branch stra Leo. 51g
pipelining, 482-186
processors, 506
8ti pc:rscalar machines . 547
Redundant Array [3]ndcpendent Disks (RAM). 16-4 : 174-183
ronfigura lion. [75
disk. technolog.. 16.1
level 1, [76, 180 181
Icv0 2. 176, IN I
level 3, 176, 181-182
level 4, 176, 182-1 N3
levl 5 : 176. 183
level 6. 176, 183
level O. 175
high. stata !Nosier capacity, 1Kii
high. 1 0 request rate., 180
I n.els, 176
illustrated. 177- I
web site, 191
Recnlrunl procedures, 352
!Register, 9, 19. .Sec c21.5:0 Address register; Control address register (CAR); Inputhitttpul (1.0): lostructiou regis• ter Memory **3ddle7'S. r gislcr {** MAR); Ytemory buffer register (M..131Z): Pen- tium control rogisier
addressing, 385..386
alias, 526
application. 565
kIln1.Oi aii.!x.
branch
I A•64 instruction set, 563
call ini-tructions. 351.
GEM
I A-41 it rchri Wet ure. 566, 568
Condition
codes, 41.6
Pc riverP(:, 450, 454
controls, 412.. 414, 416-419, 603
inicrosequeneer, 632
&Ft:nit:Tx, 6:3(1
'1'1 8800, 630
CPU, 331-332, 373, 413
data, 415
destination
'11 8832, 6341
Fi...PLACis, 442
8- hit parallel. 726
exception
Powerl)C processor. 450
floating-point
IA-64 instruction Kt, 563
Power Pt, 450.
formal
I A-6-4 a rchi tecturc., 566
gencral-purpose, 415
I A-64
iimtruction set, 56.3
lar numbers, 544
indem., 3138, 415
indirect addressing, 386
instruction buffer. 20
1 10 address, .52
link
PovicrPC proccssor..450
machine: state, 379
mapping
Protium A iMC, 447
r.v. memory, 397
operand mode
PUTI 1.101,
organization, 414-419
Pentium. q'10-441
PipherPC **pr{ 5.D2,ViCl.1", 4:7 R. I S2**
outs
SPARC, 495
parallel
sequential circuits, 725
pre.dicate
I A .n4 instructi on set. S6.3
processor x44 1115
SPA RC, 495
rcicrcn cc, 398
to regi s cr, 477, 474, 682
vector computation, 683
rcnaming. 516-517

Rcgis inr (rural.)
has;a valuer . 5hh
L--64 pipclinirsg, 51. 1
['eI1Lium 4. 5 2 6
sequential circoit5, 7?1 =727
shi[L
sequential circuits, ']27
source
TI S32, 634
stack
1A-64 iastnlclirfn . L f^7-56i
ti-v1Lh]C.4]2
FLC(OF I'icilit}', &[2
Rc'gixlcrc'd Al
]'[[st332, 620, 63,
I{€istcr tilt
gh,Ei i1]•i E, IC;. 47]E
[A-64 arehitec[L1re, 545
1|1 [eger
I': riliu4?1 4, 527
1.11E}5, 494
wir 1 4.1 b^LSCCI
i1]ui.trated, 472
ke istcr wiltdows.468--464
era'er[apYJ1rLg.4i]
SP•R(:, 4{l{^
Relatiti' LLddreMing.3[7
PenG1.ilr: {17
PQwoiPC. Y^3- 39
Re]ativt' dv[amic frcc{u {'.rfct'
high 1^ve] E&L' iiage peratiorus, 1 30
Ro[al['e Sii!c
lw i lcvwl mcmc^rics, [33-135
H L'I iahi]lC\'
ti 3e. ;50, 653
Rern nvah[e disk. ifs{}
Regi ova hlo Lnfidia, 96
Rc. irdcr hui6 r (R0]
Pentium 4 instructing-[eve I }nrohclism, 526
Replacement algorithms
e[ementS of cache ciLSigrr. 1] 5
RL]o-uri fLS3li on 1 Li
reigt'orce concept:, 743-7+1
Rtic[ucs1, S7–89
I equcsted priitegc k eI (RPL)
Eentiurn 11 egmL:oto[ion27U
Rerearch prujecis. 742
1^eset
hur c{^nlral] liries, 7]
Re.sideJl mulutar.24.E
memory lu Vii u1, 2+1
H4sule;II Cu iii]1i1, #,14
1fcseauroo conl1[cl, 511
R l7UF04 Ct1c L1diug
inicrcinstrucuan- 620
1]Cc Inaltiag€'r
OS. 240
Re:;u€t i^frerand r^Ferencc
machine inslr]cticfri, 33 t
R^5UrL' tlag
h LACiSregisle.r,442
12cttring
inIruCL '. 5]ck
Rover Polish, 374
1 1gpitt
'IF S s 12, 634
Rippic counter
':i'4] I li. n | i iii ciftu itS, 727
RIB{:..Szr Rt'duced instruction tt computer
{RCSC
ROE
E'cr'itium 4 instnictitin -level l at:aIk1i m, 52G
ROh'1- &e Rca1- L1n1} rnernnry (ROM)
Rnl,llt:
intcrropt•dri\'c 1.0. 2I3
aperiticrs, 346, 347
R[oiitirto] delay
disk pertormanco, [71, 172-]73
Rot Lit lonal]^stcTLCy
L3isl 1^^]fnr]itance, 171
Rc^ugdirig„ 3211-322
10 lii iIrCSt, 321
to plus ind minus infinit y. 32 1
1o'. and . ro 322
RI 1 LLEL r
Inlin11 and, 230
Row address ieILt1 {RASI. 150
chip 1L^gic packag]ng, 144
p]rlS- 147
RP L
PeotiuLrr 1] scgmcnh€iiin, 270
Running
pre' u4; iii ate, 252

5
Sr360. , er [Ii 1 Sr'360
S?370. Sec IBM 5;370
S?390. See IBM S. - 90
Sage and Restore registers (SRR)
FaworPC 601, 7^?
5e-,]i 1i]C Prfi1Giss0r ArchitecttJID tS.PAItC), 469,494-500
;sddresislg rnudOl
syff]ttLwrizing other ucarwsx]F ax0c]e5. 498
ig[ruction t&Sr[[npl['i, 494
irvlrLJcfiOra'let, 497
caixtcr set. 4945()')
R '. gii,I4 r \^4-1f {]L,fir Lavou^, 495

Scalar
pipeline 510
processing
computation. 675-676
referencing. 472
8/v1P, 649
Sc;Ateil inden with displacement mode
Pentium. 392
Scale index hit tS113)
Pe:Ilium .106
Scheduling. 254/-256
example. 254
OS probieins 24.2
SNIP. 653.
types. 250
&CS!
72
SD13. *SeT S1ilh4alL abVdOprnerl.1131.:111rd*
{ S11.}E4)
8.800 SD11, *Se'r* Texas losiruments
components, 627
.SDR ANI. 154-157
SEE: code, 152-1.1
.Sc condaiy or auxiliary memory,..102
Se..Ltind generation
of compLiteN, 24
Second time unit
fetich cycles. 579
Sectors. 166
Seek time
disk peirottnanee. 171% 172
Segment, 266-26S
descriptor
Pentium 2; rMiring4..!Mc.:111, 271-273
DICUIDOry view
Patgv. memory, 269
unpaeed memory, 2.69
number 2711
trio kit
Pentium, 404
Pentium II. 269--270
Perstinin prucxrisor, 441
pointers, 415
register
Pentium iits[ructim and desuipflon, 357
st.lcotor
NiitiLLM irtorriury management. 271-273
table entry
Row4jrPc memory management. 277
Sided. DR bus
microsequencer, 632
Selectt 51 Chi! nnul. c:(3 **riA,** 221
regiistei Lilo data, source
832. 633
Semantic gap. 464
Setnipunductor memory. 34, 102 138-148
organization, 138. 139
RAM. 138
technologies
web sites, 159
lypes, 140
SEND operation, 231
Separate servers
cluster **MI IVO** tiC5.LTI pi, ion., 665-666
Sequence.
arbitration
link laver, 227
events, 92
RISC' adviicaLes, 464
interrupt proc..es!..,ing. 66
rnicroinsiiuction, 609-615
Lechnique. 610-612
click performance. 173
prrIc'iNCIT conlrol% 584
Sryucirtial LiLeess
devioe. 190
memory. 98
Sequential circuits, 720-730
clocked S-k flip flop, 722
colioicrs.
D flip-flop, 722
digital logic, 720-730
720-722. 725
input data strobe, 725
parallel regislers, 725
ripple COURtel 727
shift register 727
S-R latch, 721
Serial U0 ooncro1
Inlel 8(85, 589-594
Scrial vt2curdits. 1€:9
Serpentine recording, 189
Servers connceled to disks
cluster meth det...cription, 665-666
Servers share dish
[1 11.1TI, 665-6.66
Service call
proces. 253
Set associative rnappiii.. 1]2-113
SE'rec instructions
Penti um cum Li icons, 359
SElli] instruction, 500
Setup Lime
problern, 243

Shading, 473
Shannon's techniques
Boolean BIgcbra, 694
Shared
disk
cluster method description, 666
12 caches
SNIP. 655.-65
NIESI prc5locol. 659
nothing
cluster method description, 666
write 10..662
Shift
keyboard-hnndling, 21.6
M MX instruction and description, 360
operation examples, 347
(ilaura ion& 346
register
sequential circuits, 727
Shift instrucii45u
MI FS, 4..17
modificr
FE 8K32, 634
SIJARC, 497
Short-t kitteric
proces.!... 255
Short...term scheduling. 250. 251• 25 5
SEJ3
Pentium, 41J6
Sides
clinic 5:.el•51.1=371., 169
Sign, 308
common rilZidS .01 flags, 417
extension, 364
extension rule, 290
magnitude reprcsentalion, 286
Signals. See ids.r) Control signals; Peripheral component interconnect (3'C1). signal
address selection
microinstruction, 611
Al 586
CAS. 144
clock
tinting diagram, 93
data, 591
i2N1t2rilk l1., 591
function or time
timing diagram, 93
grant, 87-89
Intel 8085, 591
interrupt relaft'.(1. 591
interrupt request. 59
RDRAM
CE., 156
read, 156
write, 156
Cutting diagram . 92
Signed haifword
PowerPC, 341
Signed word
Power PC. 341
Sigoificand
tioating•point, 308
•alignment. 31 7
overflow. 315
undcrfltry, 31.5
SI KID : 35.5, 645-647
Simple:Scalar
analysis KO ietwhing, 743
Simplicity
SNIP. 650
Simulalion proiects, 742-743
Simultaneous concurrent pnx•esses
&MP, 653
Single adtlress field
brimeli control logic. 612
microinstruction, 611
Single-bus detached DMA., 220
Single-bus integrates DMA-1;0, 220
Single control poinl
clusters, 669
Single data stream
parallel processing. #A5-647
Single entry point
clusters. 667
Single-error-eorrecliog tS.EC.) rodb, 152-151
Single file hierarchy
clusters. 669
Single ixIruction, 5incly data {SISD}
stream. 643 •647
Single inslru::iican rnul(iplu. &Pia (5]1 D)
rohion, .158
stream. 645 647
Single 110 grace
Ousters. 669
Suenlc joh-manngerncnt system
chiSicTS. 669
Single large expensive disk (SLED). 175
Single memory space
clusters. 669
Single-processor system
PC], 79
Single process space
cluswrs, 669
Sirsgle-proi ant, 248
Single sided disk. 169
Single-systcm image

clusters. 667
Single um interface
clusters, 6fi9
Single virtual networking
669
S input
TI 8832. 634
SIM)
stream, 645-647
Six-siage CPU instruction pipeline . 42S
16 magabil DRAM
illustration, 145
pins
Pet signal lines, Al. 83
ROM, 718
Skip inscructions
branch
lranAcr-of-ceinlrol instruction. 350
SLED: 175
Small computer system. interface (S :S 1)
system has. 72.
inregration (SSI)
chips. 714
description, 29-30
SMP, *See* Symmetric multi] rocesSUT SMP)
SMPCaehe
analysis and teachin, 743
Snoopy protocols, 658 659
Soft
error, I4.
I A-64 architecture, 36I
tilicrirprogramming microinstruction. 616.
619
Software, 51
approaches. 52
pipelining
example, 560
I A-644. 559-563
LA-N4 instruction, 542
poll, 212
solutions
cache coherence, 657
Software Development Board (51)3), 627.
See also Texas Instruments &IOC!
Sal
I A•64 architeciure, 568
Solid-stale device, 24
SOP, 699, 700
SOT
I A-64 archilecture. 508
Source operand reference
machine instruction, 331
Source register
TI 8232. 634
SP. RC.Se.F.. Scalable Processor Archiz cetera
(SPA RC)
Spatial locality
two level memories, 130
Special Cycle
PC.1 command, 84
Special loop termin LI ng instructions
pipelmin 561
Special rrikl'Sk
interrupt-drive Ii(), 213
Special-purpose devices, 638
SpecelaLitin arid predication
I A-64. 557
SpeculHtiVC execution
processors . 31S
Speculative. load
LA-44. 551. 554
Specula t b..ely execute insiruci ions
PowerPC, 533
Speed
microprocessor. :;7.-38
Si Millar or identi cal
(nernbers,
Speedup factors
instruction pipelining. 432
machine orgk inizations, 5 1ii 1
Split cache
u.y. 1I n ificd cache, 12C1-121
SRAM. *Sur* Static RAM
Srcs
A -64 ;-1 rCil I ittLi u 549.
S-R latch
cliiiracturistie table, 723
i rripl ern criLeti, 72 t
sequential circuits. 721
SWRs
PowerP(' 601, 532
S2-S0
in IXOCI UCII LCT, 632
551 : :19. 7] 4
Slack , I -376
irdc,1(C}iS.131g.
base. 373
0111 rok;
microsequencer,.63'2
description, 371
frame. 354
implementation ; 372
limb!. 373
operation. 372
organization example. 373
poi rilOr, 37ti, 373, 4177
T1 8,./100. 639

Sari of e tl[ed pro-cedure
call i[u tructJons, 3M
SlrEtc, 2 i i
iI[s[ruction cyeie.- c92
man ag rncnl
MFt< lr[struction and description, 360
['entiunr 4 instruction-1cvcl parailu[i.En. 526
process control block, 25?.
S1xlie RAM (SRAM), [38, [40 142, [54,]5i
ce[1. 141
LRAM, 142
Status
Pentium I;roc c1Y, 442
Stalu4 ra:i,i,lirs
eontro[, 412, 414, 416-419
St8t[[S rcportln!
[? , 202
Sratus signals
iO, [98
)tack trd1rLC growt]t, 334
Stock to imp[e[n&n[ncStciJ MuhrcFUlinL , 353
Si(Ft1
]A-64 instruction format, 547
510
Ifp^(F4]eh, 332
Siorae.e [QcLILions
CPU, 413
St,arrrgc EU rcgiter
vector computation, 6#33
Storagc to tiloragc
vector computation, 683
Stare
c1aLN, 7
instruriions, 364
MI PS. 487
S FA R{;. 497
I'owerl'C, 363
arehilcelurc, 392
pr«gra[n cLocept, L7
Strin8 operations
PunliLFm ifStruLtit.1n arir] descriprion, 3561
Strii e
RAID [e c[, [76
.Sl nJCEUr}ll CL1IT]] [tnC1i[S,]{i
Structure, 5-ill
C PL', 4J2-457
c]efir3itioii. 5
dcSC.IEptLon, 7-LO
51.1 3
opoudes, 332
Suhactiun
ascmchrorious, 228
concatenated. as'nchrorbou , 228
FicC ' ire, 22.!i
17oating-pditit ari[hmelic, 315
g^P
link layer, 227
isac h rono u& 228
Su11nc1
]11iiniliand,2311
Subroutine [t[ciliIv
LSI -]1.614
Suhtractiolti
block diagram ihardware, 26
tu'^as ulo1^[}lement, 2' 2-244
ruli , 2R7
un,ign4sl into t(x, 305
Suiti of products SOP)
Morro, h$J
illlpiCFF L III,I tiOil. 7{H1
Ours [A]1C architcclurc- *Sre* Scrr[ab1L
['rL.ecs5or ArcliiitecturC (M'AkC)
Superu)]11 11 utrYS
1_ra\:- 1]79
}'c&tor cnrnpul.aLihrti, f74
ksCh sites, 45
Su^reriar pricclper[r^rmane
cl[FSterF', 664
S ujLSFne,h [1r
approach
lin'TirtIrioIln, .5(18 51 1
detinitLon, 511
cx culicrn, Si')
x [A-M Irchitecturc, 543
im pie!ninlation
inbtrue[iom - lcvcl p Lra]lekr,m, 520--527
m. hiries
delaycd branch.'trali gy, 51!1
VrganixH O(1. 541?
processtng
conccplu:al clLTFict11FO, 5]u
pYti^usxtli, 5{^
ch&acteristics, i63
s, sup&r NhFIe, 507-50ts
Sup rvinc^r
corn[n m fickls ter hags. 41 7
Swapping, 57
258
Swappinji [Lineiiim, 251
SwiLcliCd ir5terennneCtion
Styli'. 654-655
Svllohlca
[A-64 drehitecturc, 546
Svmboli;; priigrnrn, 365, 366
Syrnholit rL^preSent3ti0r[
lnac[tiinc instruc.ti0n, 332

Symmetric multiprocessor (R W1. 644. 647-657
addressing, 650
ion. 650
availabilit5. 649
1351, 654
cache
analysis and reaching. 743
cohercricc. 651
character isrim 647-648
clusters. 6(1L)
descripticiri, 645
fault tolerance, 653
inemmental growth. 649
1,2 cache, 654
12' cache. 656
Inc inory Card, 654
memory management, 653
muiliport me mory. 631
nr ti on, 44[8]i-ti52
performance. 646
M.: 653
reliability, 650, 653
scaling. 649
scheduling, 653
S•390 cm figural ion
c•Ache hil rale, 656
shared 1.2 caches, 655-656
switched interconnection, 654 655 650
Sync
CD-ROM. 186
Sy rich PO
SNP, 65.[1]
Synchri.)ni.m, bus.Operalions
ti ming. 76
Synchronous counters. 72.I1...730
design, 72(1
Synchronous DRAM (SDRAM), 154-1.57
illusuatcd, 155
read timing, 157
Synchronoti9 timing
bus design
Syndrome word. 150-151
•Sysrern
interconnection. 9
05, 242
pins
PC1. signal lines, 81, 82
EcFfLiwaiv., 25
System access
()S. 240
System bus, 69
control sign als. 586
read cycle, 77
write cycle. 77
SIoLcir3136{1.1;iiiiily. *Svc.* I]i!v1 8fl60
Sys tcm137(1. family. *See* llitM S'3 70
lem1390

Bahl indicator
Pentium II segmentation 270
'lag. 101
Tag died
MIPS. 'N4
'Lig won]
Pentium processor, 442
TkInuitmum'24 Andy, 466
Tape. L16
magnetic, 189 -190
Turgr..i1 Incl acldr r aCA)
diniband, 229
'Frisk swiichcd
Pentium control regisrer, 444
'1'C A
Tn Fin 224
Teaching computer organization architecture prolecls. 741-744
Teinplaie field encoding
] A.64 architecture. 548
Temptmil
two level imimories, 1.30-131
lest
inslructiiins.
ll, 0, 205
Texas trislrurrien14 627-637
Nock diagrain. 627
DR: ports, 630
DRF3 purls.. 630
8847 lloating-point, 6.29
inic ger procesing chip, 62%1 h30
microinstruction format. 626. 629
mieroprogrJimmcd control. 62[7]-637
MP(', n30
itCA, 630
RCM, 630
register counters 630
8632 re..isle red **ALUL 623)**
c.x.)mpoitents, 627
stacks, 630
WCS data fi4141,62()
1)0JiLrol ;tare data field 629
output multiplcur. 630

Texas Instruments 8818
IlliCR)91..A11112DC.U1, 631
microinstruction biLs. 033
Texas Enstruments 8-U2
A LILT
eontigtuat ion mod°, 634
carry in, 634
des.tininion. i.gister, 634
registered ALL 629. b33
registered AI _ '1; fiLd.d, 634-636
R
Wle4:1 Tegiq '•r c,I.11,}1 s.ouree, 633
shift ins], 'Action modifier, 634
S inpui, 634
antra! register, 634
wE iEe enable, 633
Then path
JA.f4 rirNihileaure., 550, 553
4enerattoiI
computers, 2-5-33
Third Eirrie unit
fetch cycle, 579
32-11i1 adder
construction, 720
32-hit flouting-poinl forrmit, 308
.32- hil formal.;
expressible,
Thrashing, 112, 263
36i} archi[.ecture. *See* 113.2y1 SI360
370 architecture. *See* IBM 81370
390 ki [12c1 Lirc. *See N W* S139.0
] hree-operand instructions. 682
Three-way pipe lined timing, 483
Time.
bus design, 75
disk 15,2:31:01111,1r50.2. 171-175
comparison, [73
memory 98-103
Mearitli y etch.•, 25
multiplexing,
OS problems, 243
rl.1 1LO
two level memorixs, 129
sequence, 577
multi* interrupts example. b7
sharing
us. 15...uch multiprogramming, 250
systems, 248
6:91)
Tinter
batch OS. 245
interrupt Glass:, 511
Time stamp disable {'IS11))., 444
Timing diagrams, 92-93
instruction pipeline operation, 426
Timing signals
ince! 80FIS, 5'111
1[9]1,11, 266.-268. 273
Top-down rlppruach, 5-h
'VE T-level structure.,
'I op of stack
inStrin.:1ions. 35]
Trace-driyen. simulator
analysis and. leaching, 743
Tracks, 1165,]70
frailing eago, 92
TraDSUClii311 1:1y4.;1
fire ire. 225
•1..ransduccr
1.98
Transfer ACK
bus control linezi, 70
Tri-inq ter cif conrrol
CPI; actions, 343
with mulliple inremrupts, 66
upe.ration name and dcscriplion, ?..4S-35 4
wiu interrupt., , 61
Trap41 12.r rata
Inc niory, 1g9
Transfer time. 172.
Lusk pc!, Eui ance, 173
.Fra nisist or, 24-25
Trinsiiqur coUla
(']'L
growih, 30
Trarssisle instrued.cm. 347
Translation looka6ide buffer (TL 26E1-268
Pentium IL, 273
fransinission control
IRA, 20a
Tpinspart
layered protocol architecture., 231
'trap flag
F.:FLAGS re.i.kisE yr. 442
lientitinfintCrrup1 prix..2Sgir, 448
True (1;11 r1 Elepenikney, 51:19 51
Truth table. 695. 699
1'81), 444
'Faring, Alan, 17
Turnaround cycle
Eriensicr,
5-kIL113 I'ILSS approach
iniCTOMStrinetion, 6 I 0-6
Two .3(clrc2S.N fields
1.11 Lurch *control.* logic. 611
Two In.c1 memory
.el-ikirac[ur 12t1.]2.9
cost vs. size, 133, 134

hii 'ratio vs. sizu. 135
locality Car refereoct2. 1 N-131
operation, 131 132
pert o rrn H EICLI.. 1 0 1. 132-135
Twos complement arithmetic.
ChorLCl YisLies.217
1 wos co inplembili hiPPrry
value box, 289
Twos eintiple.rnent decimal
value box, 289
Twos C:45mplern rtL division
citaniples, 306
Twos complement mu1tiplicati011, 298-304
BootIC}i algiirilbm. 301
Twos complement operation.. 291
complermint representation, 286-288
characteristieN, 287
Two-siage instruction pipeline . 425
'I'w(t-way pip•ined liming, 4,sit3
1'wc• ay set associ 6 ve mapping e x mp1e, 117

LIVIA
definnion, 670
Unary operation
siack operai ion description, 372
Unamditional branch
S corn puler, 22
Unequal-size pariitions, 259
Uni am cache
N'. spill eaehe. 120-121
Uniform memory access (UT IA)
clefini lion. 670
Elliiraograinrriing
TCSOCIITO utiliz-ation, 249
sysi erns, 248
OS, 242
Unit of transfer, 97, 98
'UNIVAC 1 . 22
Univci sal Automatic Comp iner (1..:h1TVA(I). 22
Unpack :icier!, 346
Unpacked Et 1)
Pentium data lypes, 339
Unseginented paged memory
memory view, 269
F. iis iL!inenred unpaged memory
memory view, 269
Unsigned and Lwo complement integers
comparison of mulliplication, 2i)'4
Unsigned binary division
flowchart,.
Unsigned hinary infegerS
exarnple of division, 304
Unsigned binary multiplication
flowchart, 298
hardware implente.nta Lion, 297
Unsigned byte
PE pc, .341
Unsigned doubleword
PnwerPC. 341
Unsigned ILxlfwctrd
Power'('.
1.;nsigned integers
multiplication . 294
igricd word
Power PC, 341.
Lipper level memory, 131
hii
eache, 117
LItiT. Itl Ne.wsgroups, 14
inask
I A 64 i ostrucri on set, 563
Usyr I,i it ,ring, 638
Use' -visible registers, 412, 414, 415-416
Utilities.
1 in Iristograrnis
resource utilization . 241..l

V
Vacuum tubes, 16.-24
VariablE. Ii srrnaL
branch control 113 11C, 613
rnieroinstruction.f;11
Vari bl e-fel gi 11 400-404
VLirii..ible-size partitions, 258
VAX, 402-404, 464...K9
a prtEoa ell, 343
instruietio its example, 403
1/AX 1...780, 434
Vcc, 144
VCRs, 187
VDTs, 197
Vector addition
exsmple, 675
Vector aichi [benne
instruction set, 685-687
Vector ealcul1JIi yin
alternative programs . 6S3
V14..elor co rrtpu t t io 674-687
iipproa elle li, 674-680
illustrated, 6??
stiiragi: 10 ru i. ter, 683
storage to storage . 68.3
Vectored interrupt, 212
\ice tor prOLesSing, 67.5
vector computation, 675-676

Vccior procomIN
(3c rinirioit, 679
Vertical microinstruction, 60 L-602, 605, 616. 619
repertoire, 621
Very large system inli2rfaec (VLSI)
imnplenien t at i o t, 479
RISC, 477
Video Gas.seae IN2CIATLIALTS. (VCRs), 187
Video display terminals (VDTs). 197
View
of computc r, 6
Virtual address, 269
Virtual iTll VT11[1 I1 ig
F.T41., AGS register, 442
pending, 442
Virtual 1 an t2s., 230
Virtual. memory, 102. *121,* 263-268
machine in MAO i On. 331-332
Virtual-0S6 mode extension (VME)
Pentium control register. 414
Virtual mode (WO) bit
E..3- .LAGS register, 442
VLSI. 477, 479
V h3 bit
EFLAGS. register, 442
VME7..
Pentium control register, 444
memory. 102
Voltage and ground
Intel 8081, 591
Voltage so uree (VG(:). 1144
Von Neuinoior, John, 17..19, 50-51
Von Neumann architeelarc
concepts, .51
Von Neumann Machine, 17-20
V1[1]3
chip pacInging.]46
Vss, 144

Wlifor
Jc n , 29
gate and chip
relation ship, 29
Waiting
process state, 252
WA Ns
system bus, 72
WC'S data lick]
TI 629
Wet) pages
locality of re ferenee, 131
Wel) site. 13
ACM Special Interest Group on OpL.'..rating Systems. 27R
arithmetic Gummier, 324
Charles Flahbage Institute, 45
Computer Archittxtute Eionle Page, 14
Ddia Soirage Magazine, 191
DIA'apc 191
external memory systems, l41.
lire i re, 2-33
tloating•po int .arithmcli, 324
I A-64, 569
M 700, 44
11311 51390 procvszior. 324
]EEF. 754, 325
IF.F.F.'1.'echnical Committee. on Operating Syslems ;i nd Applications, 278
IMPA.(570
Lou iii band. 233
infirata.nd AssOciation. 233
Intel Developer's Page, 45
I t aniu m; 570
lianitirn prcKe:Mc)f arch it e-Ctlire., 50.1
Moore's Law.
Operating System Resource Center. 278
upexa Ling sySLCD1S. 277
Optical Storage Technology Association, 1431
OS Web, 218
Pentium. 44
Pc:pwcr PC, 45
Power1-C archilcG11.1112, 44
RAID Advisory C i roup, L91
RAID tedutology, 191
Rambus Site, 16I)
R guide, 6)
RDR AM, 10t.1
recording IcchaMIOgy. 191
related to computer organi4iition iod arch i leclun....; 14
wrniconducliat memory LE2chroLogic$,
TIO Home Page. 233
'C'op 500 SI1Ficl Lor0 plal Si te,
1394 Trade Association. 233
Whole computer
definition, 644
il no tvi:orks (W N s)
s:„stent bus, 72
W irk ei,"i; Lon[rol, 605-607
Wilkes's nueroinstruutions. 617
example. 608-609
'Oakes's. m]ceoprogrammcd 4!0 R1 TH I Unit ; '606
Wilkes's scheme. 618
WIM
SPA EU', 495

Wirichesrer disk, 170
track t,168
W infirm invalid mask (W1M)
SPARC, 495
Windowii-bascd register *file*
illustrilitd, 472
Virorcl. 19,97,3%. *See::* frirvo Di.)LIFFIc word;
I fall word
length, 3%
Work queue entry { W Ej, 231
w p
Nrilium control IC:gki.cr, 444
Wi0F, 231
Wraparound 361
Wrila Me control store, 638
damn 14,1
'Li MOO 62)
W ii
Llata
CPL, 413
do]recidelicy, 515-51h
110. 2115
rmmory
[C'[TIM}Ind, R
operand
pipelining, 425
policy
eache design. 1M--I19
sign
RDRA Irk
rit back, 657
cadle clifHiga., 118
Inwl 8046'6, 439
NI I I'S, •19:1
Lalls :59,63
Write g 156
pins. 144
T1 32,633
Write hit
parallel processing, 662
Writc as
procEssing 662
WMLAned. (WPi.
Perrier', control regibier, 444
Write dependency, *5il[6],-511,515—.[7]...1fi*
*wG^* $r 657
bus watching
cache design, 118
Write-write dependency, 513
Written sequenco, 577

XER
Piawe.r PC processor, 450
X014.
logical functions, 345

Y
Y **ou(pull** m6 ac
'UMW Eat

Zero
addrE.m irPitrwilions;35
oomnicirs **iir** nags, 417
Zero check
arilhmeiic, **317**
ZEROIN
cnicroseciMTIC42T. 632
ZIP cait ridges, 96

InformIT

Solutions from experts you know and trust

Logifn Register About InformIT

www. i nform it _ co m

Free, in-depth articles and supplements

Master the skills you need, when you need them

Choose from industry leading books, ebooks and training products

Get answers when you need them - from live experts or InformIT's comprehensive library

Achieve industry certification and advance your career

Visit InformIT today and get great content from PI1 Computer Science

Free Content

Pra Wce Hall end Inlui rrJ Tai e Inc:ernorkn of POWADO plc : Copyr Olt :E: .2aoc. reEirLs'i

ACRONYMS

ACM	Association for Computing, Machinery
A LIT`	Arithmetic and Logic Unit
ASCU	American Standards Code for Information Interchange
ANSI	American N4ilionnl Standards
BCD	Binary C:oded Decimal
CD	Compel Disk
CD-ROM	Compact Disk-Read Only Memory
CPU	C:entral Processing Unit
CISC	Complex Instruction Set Computer
DRAM	Dynamic Randoni ' Aci2,e.ss Memory
DMA	Direct Memory Access
DVD	Digital Versatile Disk
EPIC	Explicitly Parallel !AND' La:lion Computing
EPROM	Erasable Programmall'k Read-Only Memory
EEPROM	Elceiriealiv Erasable Programmable Rf,ad-Onl'v' Memory
FILL	High-Level Language
I/O	InputlOntpui
I AR	Instruction Address ReHsier
IC	integrated Circuit
ILEE	lris aut.e of Electrical and Flkictron ics Engineers
ILP	Instruction-Level Parallelism
I R	Instruction Register
LRU	Least Recenily (Ned
L-S1	Large-Scale Integration
MAR	Memory Address Register
.I BR	Memory Huffer Registt2r
NEESL	MI)di.fy-Lxclusivc-Shared-In valid
IVEMI;	Memory Management Unit
	Medium-Scale Iniegration
NUM A	Nonuniform Memory Access
OS	Operating System
PC	Program Counter
PCI	Peripheral Component lmeroinneet
PROM	Programmable Read-Only Memory
NW	Processor Status Word
PCB	Process Control Bioek
RAID	Redundant Array or Independent Disks
RA LL:	RegisteriArithinetic-Lovic Unit
RAM	Random-Access Memory
RISC	Reduced Instruction Set C`omputer
ROM	Read-Only Memory
SCSI	Small Computer System Interface
SMP	vnomu Lrie Multliprooessors
SRAM	Static Random-Access tylernory'
551	Small-Scale Integration
VLSI	Very Large-Scale lntegraLion
VLIW	Very Long Instruction Word