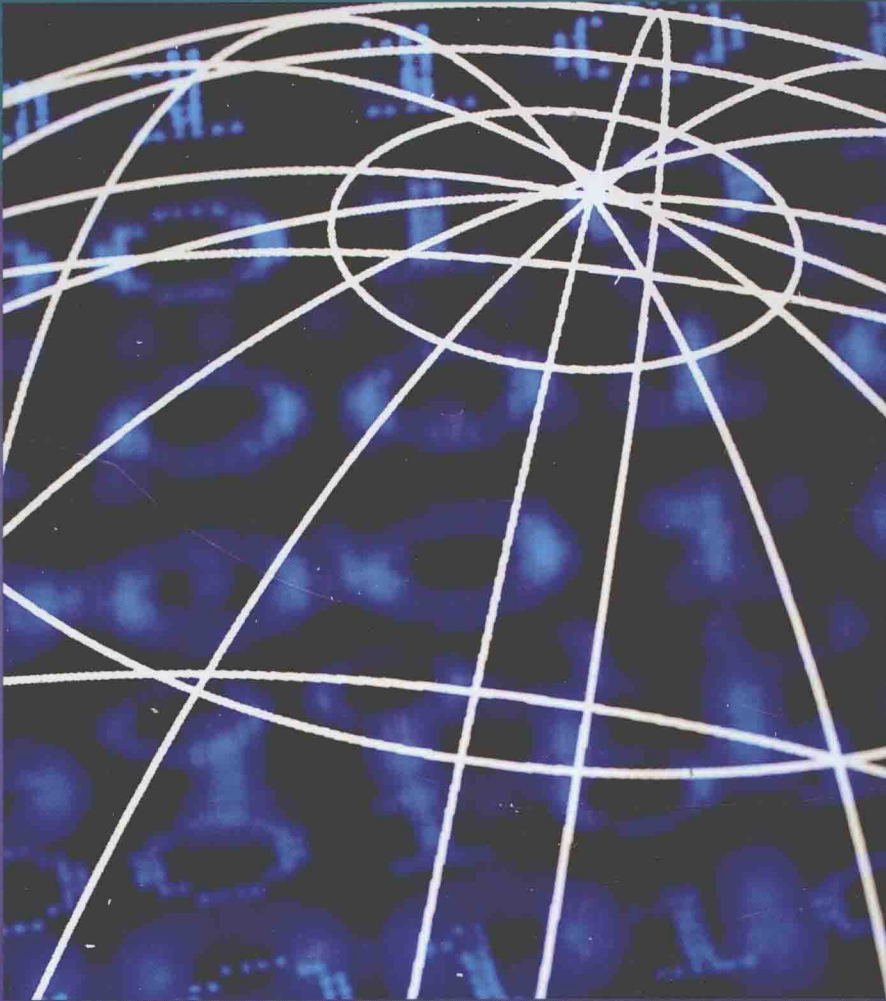


Carl Hamacher · Zvonko Vranesic · Safwat Zaky · Naraig Manjikian



# COMPUTER ORGANIZATION AND EMBEDDED SYSTEMS

Sixth Edition

# COMPUTER ORGANIZATION AND EMBEDDED SYSTEMS

---

**SIXTH EDITION**

**Carl Hamacher**

*Queen's University*

**Zvonko Vranesic**

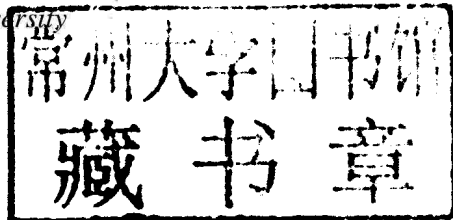
*University of Toronto*

**Safwat Zaky**

*University of Toronto*

**Naraig Manjikian**

*Queen's University*





## COMPUTER ORGANIZATION AND EMBEDDED SYSTEMS, SIXTH EDITION

Published by McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020. Copyright © 2012 by The McGraw-Hill Companies, Inc. All rights reserved. Previous editions 2002, 1996, and 1990. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written consent of The McGraw-Hill Companies, Inc., including, but not limited to, in any network or other electronic storage or transmission, or broadcast for distance learning.

Some ancillaries, including electronic and print components, may not be available to customers outside the United States.

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 DOC/DOC 0 9 8 7 6 5 4 3 2 1

ISBN 978-0-07-338065-0

MHID 0-07-338065-2

Vice President & Editor-in-Chief: *Marty Lange*

Vice President EDP/Central Publishing Services: *Kimberly Meriwether David*

Publisher: *Raghothaman Srinivasan*

Senior Sponsoring Editor: *Peter E. Massar*

Developmental Editor: *Darlene M. Schueller*

Senior Marketing Manager: *Curt Reynolds*

Senior Project Manager: *Lisa A. Brufodt*

Buyer: *Laura Fuller*

Design Coordinator: *Brenda A. Rolwes*

Media Project Manager: *Balaji Sundararaman*

Cover Design: *Studio Montage, St. Louis, Missouri*

Cover Image: © *Royalty-Free/CORBIS*

Compositor: *Techsetters, Inc.*

Typeface: *10/12 Times Roman*

Printer: *R. R. Donnelley & Sons Company/Crawfordsville, IN*

### Library of Congress Cataloging-in-Publication Data

Computer organization and embedded systems / Carl Hamacher ... [et al.]. – 6th ed.  
p. cm.

Includes bibliographical references.

ISBN-13: 978-0-07-338065-0 (alk. paper)

ISBN-10: 0-07-338065-2 (alk. paper)

1. Computer organization. 2. Embedded computer systems. I. Hamacher, V. Carl.  
QA76.9.C643.H36 2012  
004.2'2–dc22

2010050243

# CONTENTS

## Chapter 1

### BASIC STRUCTURE OF COMPUTERS 1

- 1.1 Computer Types 2
- 1.2 Functional Units 3
  - 1.2.1 Input Unit 4
  - 1.2.2 Memory Unit 4
  - 1.2.3 Arithmetic and Logic Unit 5
  - 1.2.4 Output Unit 6
  - 1.2.5 Control Unit 6
- 1.3 Basic Operational Concepts 7
- 1.4 Number Representation and Arithmetic Operations 9
  - 1.4.1 Integers 10
  - 1.4.2 Floating-Point Numbers 16
- 1.5 Character Representation 17
- 1.6 Performance 17
  - 1.6.1 Technology 17
  - 1.6.2 Parallelism 19
- 1.7 Historical Perspective 19
  - 1.7.1 The First Generation 20
  - 1.7.2 The Second Generation 20
  - 1.7.3 The Third Generation 21
  - 1.7.4 The Fourth Generation 21
- 1.8 Concluding Remarks 22
- 1.9 Solved Problems 22
  - Problems 24
  - References 25

## Chapter 2

### INSTRUCTION SET ARCHITECTURE 27

- 2.1 Memory Locations and Addresses 28
  - 2.1.1 Byte Addressability 30
  - 2.1.2 Big-Endian and Little-Endian Assignments 30
  - 2.1.3 Word Alignment 31
  - 2.1.4 Accessing Numbers and Characters 32
- 2.2 Memory Operations 32

- 2.3 Instructions and Instruction Sequencing 32
  - 2.3.1 Register Transfer Notation 33
  - 2.3.2 Assembly-Language Notation 33
  - 2.3.3 RISC and CISC Instruction Sets 34
  - 2.3.4 Introduction to RISC Instruction Sets 34
  - 2.3.5 Instruction Execution and Straight-Line Sequencing 36
  - 2.3.6 Branching 37
  - 2.3.7 Generating Memory Addresses 40
- 2.4 Addressing Modes 40
  - 2.4.1 Implementation of Variables and Constants 41
  - 2.4.2 Indirection and Pointers 42
  - 2.4.3 Indexing and Arrays 45
- 2.5 Assembly Language 48
  - 2.5.1 Assembler Directives 50
  - 2.5.2 Assembly and Execution of Programs 53
  - 2.5.3 Number Notation 54
- 2.6 Stacks 55
- 2.7 Subroutines 56
  - 2.7.1 Subroutine Nesting and the Processor Stack 58
  - 2.7.2 Parameter Passing 59
  - 2.7.3 The Stack Frame 63
- 2.8 Additional Instructions 65
  - 2.8.1 Logic Instructions 67
  - 2.8.2 Shift and Rotate Instructions 68
  - 2.8.3 Multiplication and Division 71
- 2.9 Dealing with 32-Bit Immediate Values 73
- 2.10 CISC Instruction Sets 74
  - 2.10.1 Additional Addressing Modes 75
  - 2.10.2 Condition Codes 77
- 2.11 RISC and CISC Styles 78
- 2.12 Example Programs 79
  - 2.12.1 Vector Dot Product Program 79
  - 2.12.2 String Search Program 81
- 2.13 Encoding of Machine Instructions 82
- 2.14 Concluding Remarks 85
- 2.15 Solved Problems 85
  - Problems 90

## Chapter 3

### BASIC INPUT/OUTPUT 95

- 3.1 Accessing I/O Devices 96
  - 3.1.1 I/O Device Interface 97
  - 3.1.2 Program-Controlled I/O 97
  - 3.1.3 An Example of a RISC-Style I/O Program 101
  - 3.1.4 An Example of a CISC-Style I/O Program 101
- 3.2 Interrupts 103
  - 3.2.1 Enabling and Disabling Interrupts 106
  - 3.2.2 Handling Multiple Devices 107
  - 3.2.3 Controlling I/O Device Behavior 109
  - 3.2.4 Processor Control Registers 110
  - 3.2.5 Examples of Interrupt Programs 111
  - 3.2.6 Exceptions 116
- 3.3 Concluding Remarks 119
- 3.4 Solved Problems 119
  - Problems 126

## Chapter 4

### SOFTWARE 129

- 4.1 The Assembly Process 130
  - 4.1.1 Two-pass Assembler 131
- 4.2 Loading and Executing Object Programs 131
- 4.3 The Linker 132
- 4.4 Libraries 133
- 4.5 The Compiler 133
  - 4.5.1 Compiler Optimizations 134
  - 4.5.2 Combining Programs Written in Different Languages 134
- 4.6 The Debugger 134
- 4.7 Using a High-level Language for I/O Tasks 137
- 4.8 Interaction between Assembly Language and C Language 139
- 4.9 The Operating System 143
  - 4.9.1 The Boot-strapping Process 144
  - 4.9.2 Managing the Execution of Application Programs 144
  - 4.9.3 Use of Interrupts in Operating Systems 146
- 4.10 Concluding Remarks 149
  - Problems 149
  - References 150

## Chapter 5

### BASIC PROCESSING UNIT 151

- 5.1 Some Fundamental Concepts 152
- 5.2 Instruction Execution 155
  - 5.2.1 Load Instructions 155
  - 5.2.2 Arithmetic and Logic Instructions 156
  - 5.2.3 Store Instructions 157
- 5.3 Hardware Components 158
  - 5.3.1 Register File 158
  - 5.3.2 ALU 160
  - 5.3.3 Datapath 161
  - 5.3.4 Instruction Fetch Section 164
- 5.4 Instruction Fetch and Execution Steps 165
  - 5.4.1 Branching 168
  - 5.4.2 Waiting for Memory 171
- 5.5 Control Signals 172
- 5.6 Hardwired Control 175
  - 5.6.1 Datapath Control Signals 177
  - 5.6.2 Dealing with Memory Delay 177
- 5.7 CISC-Style Processors 178
  - 5.7.1 An Interconnect using Buses 180
  - 5.7.2 Microprogrammed Control 183
- 5.8 Concluding Remarks 185
- 5.9 Solved Problems 185
  - Problems 188

## Chapter 6

### PIPELINING 193

- 6.1 Basic Concept—The Ideal Case 194
- 6.2 Pipeline Organization 195
- 6.3 Pipelining Issues 196
- 6.4 Data Dependencies 197
  - 6.4.1 Operand Forwarding 198
  - 6.4.2 Handling Data Dependencies in Software 199
- 6.5 Memory Delays 201
- 6.6 Branch Delays 202
  - 6.6.1 Unconditional Branches 202
  - 6.6.2 Conditional Branches 204
  - 6.6.3 The Branch Delay Slot 204
  - 6.6.4 Branch Prediction 205
- 6.7 Resource Limitations 209
- 6.8 Performance Evaluation 209
  - 6.8.1 Effects of Stalls and Penalties 210
  - 6.8.2 Number of Pipeline Stages 212

- 6.9 Superscalar Operation 212
  - 6.9.1 Branches and Data Dependencies 214
  - 6.9.2 Out-of-Order Execution 215
  - 6.9.3 Execution Completion 216
  - 6.9.4 Dispatch Operation 217
- 6.10 Pipelining in CISC Processors 218
  - 6.10.1 Pipelining in ColdFire Processors 219
  - 6.10.2 Pipelining in Intel Processors 219
- 6.11 Concluding Remarks 220
- 6.12 Examples of Solved Problems 220
  - Problems 222
  - References 226

## Chapter 7

### INPUT/OUTPUT ORGANIZATION 227

- 7.1 Bus Structure 228
- 7.2 Bus Operation 229
  - 7.2.1 Synchronous Bus 230
  - 7.2.2 Asynchronous Bus 233
  - 7.2.3 Electrical Considerations 236
- 7.3 Arbitration 237
- 7.4 Interface Circuits 238
  - 7.4.1 Parallel Interface 239
  - 7.4.2 Serial Interface 243
- 7.5 Interconnection Standards 247
  - 7.5.1 Universal Serial Bus (USB) 247
  - 7.5.2 FireWire 251
  - 7.5.3 PCI Bus 252
  - 7.5.4 SCSI Bus 256
  - 7.5.5 SATA 258
  - 7.5.6 SAS 258
  - 7.5.7 PCI Express 258
- 7.6 Concluding Remarks 260
- 7.7 Solved Problems 260
  - Problems 263
  - References 266

## Chapter 8

### THE MEMORY SYSTEM 267

- 8.1 Basic Concepts 268
- 8.2 Semiconductor RAM Memories 270
  - 8.2.1 Internal Organization of Memory Chips 270
  - 8.2.2 Static Memories 271
  - 8.2.3 Dynamic RAMs 274

- 8.2.4 Synchronous DRAMs 276
- 8.2.5 Structure of Larger Memories 279
- 8.3 Read-only Memories 282
  - 8.3.1 ROM 283
  - 8.3.2 PROM 283
  - 8.3.3 EPROM 284
  - 8.3.4 EEPROM 284
  - 8.3.5 Flash Memory 284
- 8.4 Direct Memory Access 285
- 8.5 Memory Hierarchy 288
- 8.6 Cache Memories 289
  - 8.6.1 Mapping Functions 291
  - 8.6.2 Replacement Algorithms 296
  - 8.6.3 Examples of Mapping Techniques 297
- 8.7 Performance Considerations 300
  - 8.7.1 Hit Rate and Miss Penalty 301
  - 8.7.2 Caches on the Processor Chip 302
  - 8.7.3 Other Enhancements 303
- 8.8 Virtual Memory 305
  - 8.8.1 Address Translation 306
- 8.9 Memory Management Requirements 310
- 8.10 Secondary Storage 311
  - 8.10.1 Magnetic Hard Disks 311
  - 8.10.2 Optical Disks 317
  - 8.10.3 Magnetic Tape Systems 322
- 8.11 Concluding Remarks 323
- 8.12 Solved Problems 324
  - Problems 328
  - References 332

## Chapter 9

### ARITHMETIC 335

- 9.1 Addition and Subtraction of Signed Numbers 336
  - 9.1.1 Addition/Subtraction Logic Unit 336
- 9.2 Design of Fast Adders 339
  - 9.2.1 Carry-Lookahead Addition 340
- 9.3 Multiplication of Unsigned Numbers 344
  - 9.3.1 Array Multiplier 344
  - 9.3.2 Sequential Circuit Multiplier 346
- 9.4 Multiplication of Signed Numbers 346
  - 9.4.1 The Booth Algorithm 348
- 9.5 Fast Multiplication 351
  - 9.5.1 Bit-Pair Recoding of Multipliers 352
  - 9.5.2 Carry-Save Addition of Summands 353

- 9.5.3 Summand Addition Tree using 3-2 Reducers 355
- 9.5.4 Summand Addition Tree using 4-2 Reducers 357
- 9.5.5 Summary of Fast Multiplication 359
- 9.6 Integer Division 360
- 9.7 Floating-Point Numbers and Operations 363
  - 9.7.1 Arithmetic Operations on Floating-Point Numbers 367
  - 9.7.2 Guard Bits and Truncation 368
  - 9.7.3 Implementing Floating-Point Operations 369
- 9.8 Decimal-to-Binary Conversion 372
- 9.9 Concluding Remarks 372
- 9.10 Solved Problems 374
  - Problems 377
  - References 383

## Chapter 10

### EMBEDDED SYSTEMS 385

- 10.1 Examples of Embedded Systems 386
  - 10.1.1 Microwave Oven 386
  - 10.1.2 Digital Camera 387
  - 10.1.3 Home Telemetry 390
- 10.2 Microcontroller Chips for Embedded Applications 390
- 10.3 A Simple Microcontroller 392
  - 10.3.1 Parallel I/O Interface 392
  - 10.3.2 Serial I/O Interface 395
  - 10.3.3 Counter/Timer 397
  - 10.3.4 Interrupt-Control Mechanism 399
  - 10.3.5 Programming Examples 399
- 10.4 Reaction Timer—A Complete Example 401
- 10.5 Sensors and Actuators 407
  - 10.5.1 Sensors 407
  - 10.5.2 Actuators 410
  - 10.5.3 Application Examples 411
- 10.6 Microcontroller Families 412
  - 10.6.1 Microcontrollers Based on the Intel 8051 413
  - 10.6.2 Freescale Microcontrollers 413
  - 10.6.3 ARM Microcontrollers 414
- 10.7 Design Issues 414
- 10.8 Concluding Remarks 417
  - Problems 418
  - References 420

## Chapter 11

### SYSTEM-ON-A-CHIP—A CASE STUDY 421

- 11.1 FPGA Implementation 422
  - 11.1.1 FPGA Devices 423
  - 11.1.2 Processor Choice 423
- 11.2 Computer-Aided Design Tools 424
  - 11.2.1 Altera CAD Tools 425
- 11.3 Alarm Clock Example 428
  - 11.3.1 User's View of the System 428
  - 11.3.2 System Definition and Generation 429
  - 11.3.3 Circuit Implementation 430
  - 11.3.4 Application Software 431
- 11.4 Concluding Remarks 440
  - Problems 440
  - References 441

## Chapter 12

### PARALLEL PROCESSING AND PERFORMANCE 443

- 12.1 Hardware Multithreading 444
- 12.2 Vector (SIMD) Processing 445
  - 12.2.1 Graphics Processing Units (GPUs) 448
- 12.3 Shared-Memory Multiprocessors 448
  - 12.3.1 Interconnection Networks 450
- 12.4 Cache Coherence 453
  - 12.4.1 Write-Through Protocol 453
  - 12.4.2 Write-Back protocol 454
  - 12.4.3 Snoopy Caches 454
  - 12.4.4 Directory-Based Cache Coherence 456
- 12.5 Message-Passing Multicomputers 456
- 12.6 Parallel Programming for Multiprocessors 456
- 12.7 Performance Modeling 460
- 12.8 Concluding Remarks 461
  - Problems 462
  - References 463

## Appendix A

### LOGIC CIRCUITS 465

- A.1 Basic Logic Functions
  - A.1.1 Electronic Logic Gates 469
- A.2 Synthesis of Logic Functions 470



A.3	Minimization of Logic Expressions	472
A.3.1	Minimization using Karnaugh Maps	475
A.3.2	Don't-Care Conditions	477
A.4	Synthesis with NAND and NOR Gates	479
A.5	Practical Implementation of Logic Gates	482
A.5.1	CMOS Circuits	484
A.5.2	Propagation Delay	489
A.5.3	Fan-In and Fan-Out Constraints	490
A.5.4	Tri-State Buffers	491
A.6	Flip-Flops	492
A.6.1	Gated Latches	493
A.6.2	Master-Slave Flip-Flop	495
A.6.3	Edge Triggering	498
A.6.4	T Flip-Flop	498
A.6.5	JK Flip-Flop	499
A.6.6	Flip-Flops with Preset and Clear	501
A.7	Registers and Shift Registers	502
A.8	Counters	503
A.9	Decoders	505
A.10	Multiplexers	506
A.11	Programmable Logic Devices (PLDs)	509
A.11.1	Programmable Logic Array (PLA)	509
A.11.2	Programmable Array Logic (PAL)	511
A.11.3	Complex Programmable Logic Devices (CPLDs)	512
A.12	Field-Programmable Gate Arrays	514
A.13	Sequential Circuits	516
A.13.1	Design of an Up/Down Counter as a Sequential Circuit	516
A.13.2	Timing Diagrams	519
A.13.3	The Finite State Machine Model	520
A.13.4	Synthesis of Finite State Machines	521
A.14	Concluding Remarks	522
	Problems	522
	References	528

## Appendix B

### THE ALTERA NIOS II PROCESSOR 529

B.1	Nios II Characteristics	530
B.2	General-Purpose Registers	531
B.3	Addressing Modes	532
B.4	Instructions	533
B.4.1	Notation	533
B.4.2	Load and Store Instructions	534
B.4.3	Arithmetic Instructions	536

B.4.4	Logic Instructions	537
B.4.5	Move Instructions	537
B.4.6	Branch and Jump Instructions	538
B.4.7	Subroutine Linkage Instructions	541
B.4.8	Comparison Instructions	545
B.4.9	Shift Instructions	546
B.4.10	Rotate Instructions	547
B.4.11	Control Instructions	548
B.5	Pseudoinstructions	548
B.6	Assembler Directives	549
B.7	Carry and Overflow Detection	551
B.8	Example Programs	553
B.9	Control Registers	553
B.10	Input/Output	555
B.10.1	Program-Controlled I/O	556
B.10.2	Interrupts and Exceptions	556
B.11	Advanced Configurations of Nios II Processor	562
B.11.1	External Interrupt Controller	562
B.11.2	Memory Management Unit	562
B.11.3	Floating-Point Hardware	562
B.12	Concluding Remarks	563
B.13	Solved Problems	563
	Problems	568

## Appendix C

### THE COLD FIRE PROCESSOR 571

C.1	Memory Organization	572
C.2	Registers	572
C.3	Instructions	573
C.3.1	Addressing Modes	575
C.3.2	Move Instruction	577
C.3.3	Arithmetic Instructions	578
C.3.4	Branch and Jump Instructions	582
C.3.5	Logic Instructions	585
C.3.6	Shift Instructions	586
C.3.7	Subroutine Linkage Instructions	587
C.4	Assembler Directives	593
C.5	Example Programs	594
C.5.1	Vector Dot Product Program	594
C.5.2	String Search Program	595
C.6	Mode of Operation and Other Control Features	596
C.7	Input/Output	597
C.8	Floating-Point Operations	599
C.8.1	FMOVE Instruction	599



- C.8.2 Floating-Point Arithmetic Instructions 600
- C.8.3 Comparison and Branch Instructions 601
- C.8.4 Additional Floating-Point Instructions 601
- C.8.5 Example Floating-Point Program 602
- C.9 Concluding Remarks 603
- C.10 Solved Problems 603
  - Problems 608
  - References 609

## Appendix D

### THE ARM PROCESSOR 611

- D.1 ARM Characteristics 612
  - D.1.1 Unusual Aspects of the ARM Architecture 612
- D.2 Register Structure 613
- D.3 Addressing Modes 614
  - D.3.1 Basic Indexed Addressing Mode 614
  - D.3.2 Relative Addressing Mode 615
  - D.3.3 Index Modes with Writeback 616
  - D.3.4 Offset Determination 616
  - D.3.5 Register, Immediate, and Absolute Addressing Modes 618
  - D.3.6 Addressing Mode Examples 618
- D.4 Instructions 621
  - D.4.1 Load and Store Instructions 621
  - D.4.2 Arithmetic Instructions 622
  - D.4.3 Move Instructions 625
  - D.4.4 Logic and Test Instructions 626
  - D.4.5 Compare Instructions 627
  - D.4.6 Setting Condition Code Flags 628
  - D.4.7 Branch Instructions 628
  - D.4.8 Subroutine Linkage Instructions 631
- D.5 Assembly Language 635
  - D.5.1 Pseudoinstructions 637
- D.6 Example Programs 638
  - D.6.1 Vector Dot Product 639
  - D.6.2 String Search 639
- D.7 Operating Modes and Exceptions 639
  - D.7.1 Banked Registers 641
  - D.7.2 Exception Types 642
  - D.7.3 System Mode 644
  - D.7.4 Handling Exceptions 644
- D.8 Input/Output 646
  - D.8.1 Program-Controlled I/O 646
  - D.8.2 Interrupt-Driven I/O 648

- D.9 Conditional Execution of Instructions 648
- D.10 Coprocessors 650
- D.11 Embedded Applications and the Thumb ISA 651
- D.12 Concluding Remarks 651
- D.13 Solved Problems 652
  - Problems 657
  - References 660

## Appendix E

### THE INTEL IA-32 ARCHITECTURE 661

- E.1 Memory Organization 662
- E.2 Register Structure 662
- E.3 Addressing Modes 665
- E.4 Instructions 668
  - E.4.1 Machine Instruction Format 670
  - E.4.2 Assembly-Language Notation 670
  - E.4.3 Move Instruction 671
  - E.4.4 Load-Effective-Address Instruction 671
  - E.4.5 Arithmetic Instructions 672
  - E.4.6 Jump and Loop Instructions 674
  - E.4.7 Logic Instructions 677
  - E.4.8 Shift and Rotate Instructions 678
  - E.4.9 Subroutine Linkage Instructions 679
  - E.4.10 Operations on Large Numbers 681
- E.5 Assembler Directives 685
- E.6 Example Programs 686
  - E.6.1 Vector Dot Product Program 686
  - E.6.2 String Search Program 686
- E.7 Interrupts and Exceptions 687
- E.8 Input/Output Examples 689
- E.9 Scalar Floating-Point Operations 690
  - E.9.1 Load and Store Instructions 692
  - E.9.2 Arithmetic Instructions 693
  - E.9.3 Comparison Instructions 694
  - E.9.4 Additional Instructions 694
  - E.9.5 Example Floating-Point Program 694
- E.10 Multimedia Extension (MMX) Operations 695
- E.11 Vector (SIMD) Floating-Point Operations 696
- E.12 Examples of Solved Problems 697
- E.13 Concluding Remarks 702
  - Problems 702
  - References 703

---

## **chapter**

# **1**

# **BASIC STRUCTURE OF COMPUTERS**

## **CHAPTER OBJECTIVES**

In this chapter you will be introduced to:

- The different types of computers
- The basic structure of a computer and its operation
- Machine instructions and their execution
- Number and character representations
- Addition and subtraction of binary numbers
- Basic performance issues in computer systems
- A brief history of computer development

This book is about computer organization. It explains the function and design of the various units of digital computers that store and process information. It also deals with the input units of the computer which receive information from external sources and the output units which send computed results to external destinations. The input, storage, processing, and output operations are governed by a list of instructions that constitute a program.

Most of the material in the book is devoted to *computer hardware* and *computer architecture*. Computer hardware consists of electronic circuits, magnetic and optical storage devices, displays, electromechanical devices, and communication facilities. Computer architecture encompasses the specification of an instruction set and the functional behavior of the hardware units that implement the instructions.

Many aspects of programming and software components in computer systems are also discussed in the book. It is important to consider both hardware and software aspects of the design of the various computer components in order to gain a good understanding of computer systems.

---

## 1.1 COMPUTER TYPES

Since their introduction in the 1940s, digital computers have evolved into many different types that vary widely in size, cost, computational power, and intended use. Modern computers can be divided roughly into four general categories:

- *Embedded computers* are integrated into a larger device or system in order to automatically monitor and control a physical process or environment. They are used for a specific purpose rather than for general processing tasks. Typical applications include industrial and home automation, appliances, telecommunication products, and vehicles. Users may not even be aware of the role that computers play in such systems.
- *Personal computers* have achieved widespread use in homes, educational institutions, and business and engineering office settings, primarily for dedicated individual use. They support a variety of applications such as general computation, document preparation, computer-aided design, audiovisual entertainment, interpersonal communication, and Internet browsing. A number of classifications are used for personal computers. *Desktop computers* serve general needs and fit within a typical personal workspace. *Workstation computers* offer higher computational capacity and more powerful graphical display capabilities for engineering and scientific work. Finally, *Portable* and *Notebook computers* provide the basic features of a personal computer in a smaller lightweight package. They can operate on batteries to provide mobility.
- *Servers* and *Enterprise systems* are large computers that are meant to be shared by a potentially large number of users who access them from some form of personal computer over a public or private network. Such computers may host large databases and provide information processing for a government agency or a commercial organization.
- *Supercomputers* and *Grid computers* normally offer the highest performance. They are the most expensive and physically the largest category of computers. Supercomputers are used for the highly demanding computations needed in weather forecasting, engineering design and simulation, and scientific work. They have a high cost. Grid computers provide a more cost-effective alternative. They combine a large number of personal computers and

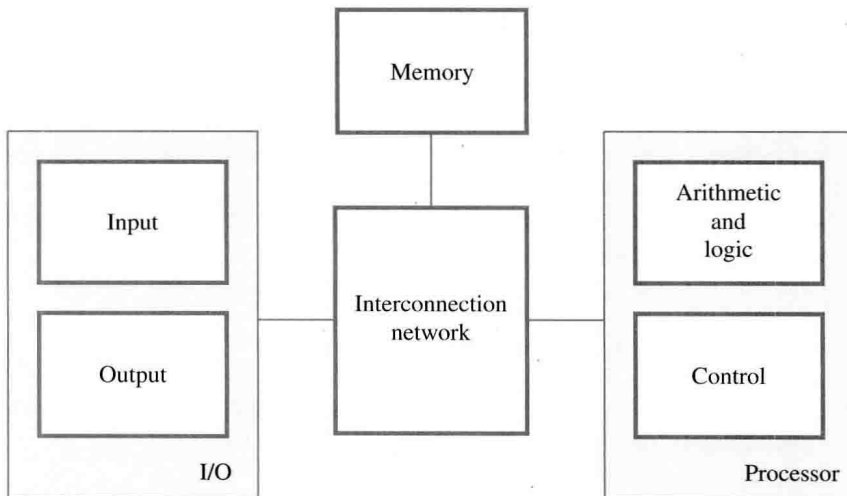
disk storage units in a physically distributed high-speed network, called a grid, which is managed as a coordinated computing resource. By evenly distributing the computational workload across the grid, it is possible to achieve high performance on large applications ranging from numerical computation to information searching.

There is an emerging trend in access to computing facilities, known as *cloud computing*. Personal computer users access widely distributed computing and storage server resources for individual, independent, computing needs. The Internet provides the necessary communication facility. Cloud hardware and software service providers operate as a utility, charging on a pay-as-you-use basis.

---

## 1.2 FUNCTIONAL UNITS

A computer consists of five functionally independent main parts: input, memory, arithmetic and logic, output, and control units, as shown in Figure 1.1. The input unit accepts coded information from human operators using devices such as keyboards, or from other computers over digital communication lines. The information received is stored in the computer's memory, either for later use or to be processed immediately by the arithmetic and logic unit. The processing steps are specified by a program that is also stored in the memory. Finally, the results are sent back to the outside world through the output unit. All of these actions are coordinated by the control unit. An interconnection network provides the means for the functional units to exchange information and coordinate their actions. Later chapters will provide more details on individual units and their interconnections. We refer to the



**Figure 1.1** Basic functional units of a computer.

arithmetic and logic circuits, in conjunction with the main control circuits, as the *processor*. Input and output equipment is often collectively referred to as the *input-output (I/O) unit*.

We now take a closer look at the information handled by a computer. It is convenient to categorize this information as either instructions or data. *Instructions*, or *machine instructions*, are explicit commands that

- Govern the transfer of information within a computer as well as between the computer and its I/O devices
- Specify the arithmetic and logic operations to be performed

A *program* is a list of instructions which performs a task. Programs are stored in the memory. The processor fetches the program instructions from the memory, one after another, and performs the desired operations. The computer is controlled by the stored program, except for possible external interruption by an operator or by I/O devices connected to it. *Data* are numbers and characters that are used as operands by the instructions. Data are also stored in the memory.

The instructions and data handled by a computer must be encoded in a suitable format. Most present-day hardware employs digital circuits that have only two stable states. Each instruction, number, or character is encoded as a string of binary digits called *bits*, each having one of two possible values, 0 or 1, represented by the two stable states. Numbers are usually represented in positional binary notation, as discussed in Section 1.4. Alphanumeric characters are also expressed in terms of binary codes, as discussed in Section 1.5.

### 1.2.1 INPUT UNIT

Computers accept coded information through input units. The most common input device is the keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted to the processor.

Many other kinds of input devices for human-computer interaction are available, including the touchpad, mouse, joystick, and trackball. These are often used as graphic input devices in conjunction with displays. Microphones can be used to capture audio input which is then sampled and converted into digital codes for storage and processing. Similarly, cameras can be used to capture video input.

Digital communication facilities, such as the Internet, can also provide input to a computer from other computers and database servers.

### 1.2.2 MEMORY UNIT

The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.

#### Primary Memory

*Primary memory*, also called *main memory*, is a fast memory that operates at electronic speeds. Programs must be stored in this memory while they are being executed. The

memory consists of a large number of semiconductor storage cells, each capable of storing one bit of information. These cells are rarely read or written individually. Instead, they are handled in groups of fixed size called *words*. The memory is organized so that one word can be stored or retrieved in one basic operation. The number of bits in each word is referred to as the *word length* of the computer, typically 16, 32, or 64 bits.

To provide easy access to any word in the memory, a distinct *address* is associated with each word location. Addresses are consecutive numbers, starting from 0, that identify successive locations. A particular word is accessed by specifying its address and issuing a control command to the memory that starts the storage or retrieval process.

Instructions and data can be written into or read from the memory under the control of the processor. It is essential to be able to access any word location in the memory as quickly as possible. A memory in which any location can be accessed in a short and fixed amount of time after specifying its address is called a *random-access memory* (RAM). The time required to access one word is called the *memory access time*. This time is independent of the location of the word being accessed. It typically ranges from a few nanoseconds (ns) to about 100 ns for current RAM units.

### Cache Memory

As an adjunct to the main memory, a smaller, faster RAM unit, called a *cache*, is used to hold sections of a program that are currently being executed, along with any associated data. The cache is tightly coupled with the processor and is usually contained on the same integrated-circuit chip. The purpose of the cache is to facilitate high instruction execution rates.

At the start of program execution, the cache is empty. All program instructions and any required data are stored in the main memory. As execution proceeds, instructions are fetched into the processor chip, and a copy of each is placed in the cache. When the execution of an instruction requires data located in the main memory, the data are fetched and copies are also placed in the cache.

Now, suppose a number of instructions are executed repeatedly as happens in a program loop. If these instructions are available in the cache, they can be fetched quickly during the period of repeated use. Similarly, if the same data locations are accessed repeatedly while copies of their contents are available in the cache, they can be fetched quickly.

### Secondary Storage

Although primary memory is essential, it tends to be expensive and does not retain information when power is turned off. Thus additional, less expensive, permanent *secondary storage* is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently. Access times for secondary storage are longer than for primary memory. A wide selection of secondary storage devices is available, including *magnetic disks*, *optical disks* (DVD and CD), and *flash memory devices*.

## 1.2.3 ARITHMETIC AND LOGIC UNIT

Most computer operations are executed in the *arithmetic and logic unit* (ALU) of the processor. Any arithmetic or logic operation, such as addition, subtraction, multiplication,

division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU. For example, if two numbers located in the memory are to be added, they are brought into the processor, and the addition is carried out by the ALU. The sum may then be stored in the memory or retained in the processor for immediate use.

When operands are brought into the processor, they are stored in high-speed storage elements called *registers*. Each register can store one word of data. Access times to registers are even shorter than access times to the cache unit on the processor chip.

### 1.2.4 OUTPUT UNIT

The output unit is the counterpart of the input unit. Its function is to send processed results to the outside world. A familiar example of such a device is a *printer*. Most printers employ either photocopying techniques, as in laser printers, or ink jet streams. Such printers may generate output at speeds of 20 or more pages per minute. However, printers are mechanical devices, and as such are quite slow compared to the electronic speed of a processor.

Some units, such as graphic displays, provide both an output function, showing text and graphics, and an input function, through touchscreen capability. The dual role of such units is the reason for using the single name *input/output (I/O)* unit in many cases.

### 1.2.5 CONTROL UNIT

The memory, arithmetic and logic, and I/O units store and process information and perform input and output operations. The operation of these units must be coordinated in some way. This is the responsibility of the control unit. The control unit is effectively the nerve center that sends control signals to other units and senses their states.

I/O transfers, consisting of input and output operations, are controlled by program instructions that identify the devices involved and the information to be transferred. Control circuits are responsible for generating the *timing signals* that govern the transfers and determine when a given action is to take place. Data transfers between the processor and the memory are also managed by the control unit through timing signals. It is reasonable to think of a control unit as a well-defined, physically separate unit that interacts with other parts of the computer. In practice, however, this is seldom the case. Much of the control circuitry is physically distributed throughout the computer. A large set of control lines (wires) carries the signals used for timing and synchronization of events in all units.

The operation of a computer can be summarized as follows:

- The computer accepts information in the form of programs and data through an input unit and stores it in the memory.
- Information stored in the memory is fetched under program control into an arithmetic and logic unit, where it is processed.
- Processed information leaves the computer through an output unit.
- All activities in the computer are directed by the control unit.



---

## 1.3 BASIC OPERATIONAL CONCEPTS

In Section 1.2, we stated that the activity in a computer is governed by instructions. To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be used as instruction operands are also stored in the memory.

A typical instruction might be

Load   R2, LOC

This instruction reads the contents of a memory location whose address is represented symbolically by the label LOC and loads them into processor register R2. The original contents of location LOC are preserved, whereas those of register R2 are overwritten. Execution of this instruction requires several steps. First, the instruction is fetched from the memory into the processor. Next, the operation to be performed is determined by the control unit. The operand at LOC is then fetched from the memory into the processor. Finally, the operand is stored in register R2.

After operands have been loaded from memory into processor registers, arithmetic or logic operations can be performed on them. For example, the instruction

Add   R4, R2, R3

adds the contents of registers R2 and R3, then places their sum into register R4. The operands in R2 and R3 are not altered, but the previous value in R4 is overwritten by the sum.

After completing the desired operations, the results are in processor registers. They can be transferred to the memory using instructions such as

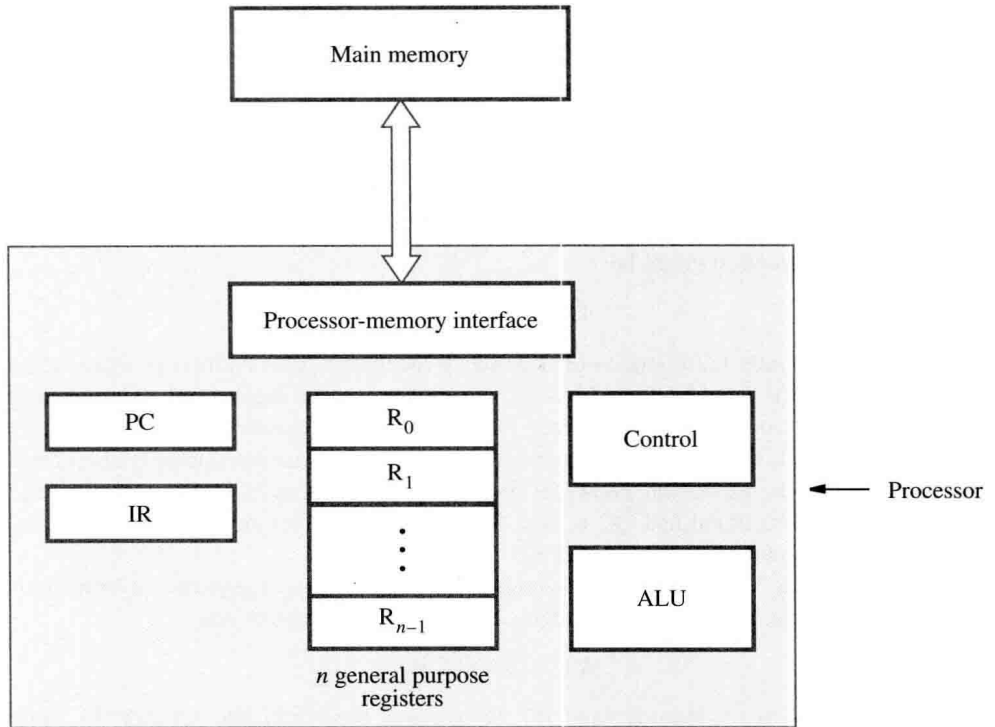
Store   R4, LOC

This instruction copies the operand in register R4 to memory location LOC. The original contents of location LOC are overwritten, but those of R4 are preserved.

For Load and Store instructions, transfers between the memory and the processor are initiated by sending the address of the desired memory location to the memory unit and asserting the appropriate control signals. The data are then transferred to or from the memory.

Figure 1.2 shows how the memory and the processor can be connected. It also shows some components of the processor that have not been discussed yet. The interconnections between these components are not shown explicitly since we will only discuss their functional characteristics here. Chapter 5 describes the details of the interconnections as part of processor organization.

In addition to the ALU and the control circuitry, the processor contains a number of registers used for several different purposes. The *instruction register* (IR) holds the instruction that is currently being executed. Its output is available to the control circuits, which generate the timing signals that control the various processing elements involved in executing the instruction. The *program counter* (PC) is another specialized register. It



**Figure 1.2** Connection between the processor and the main memory.

contains the memory address of the next instruction to be fetched and executed. During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed. It is customary to say that the PC *points* to the next instruction that is to be fetched from the memory. In addition to the IR and PC, Figure 1.2 shows *general-purpose registers*  $R_0$  through  $R_{n-1}$ , often called processor registers. They serve a variety of functions, including holding operands that have been loaded from the memory for processing. The roles of the general-purpose registers are explained in detail in Chapter 2.

The processor-memory interface is a circuit which manages the transfer of data between the main memory and the processor. If a word is to be read from the memory, the interface sends the address of that word to the memory along with a Read control signal. The interface waits for the word to be retrieved, then transfers it to the appropriate processor register. If a word is to be written into memory, the interface transfers both the address and the word to the memory along with a Write control signal.

Let us now consider some typical operating steps. A program must be in the main memory in order for it to be executed. It is often transferred there from secondary storage through the input unit. Execution of the program begins when the PC is set to point to the