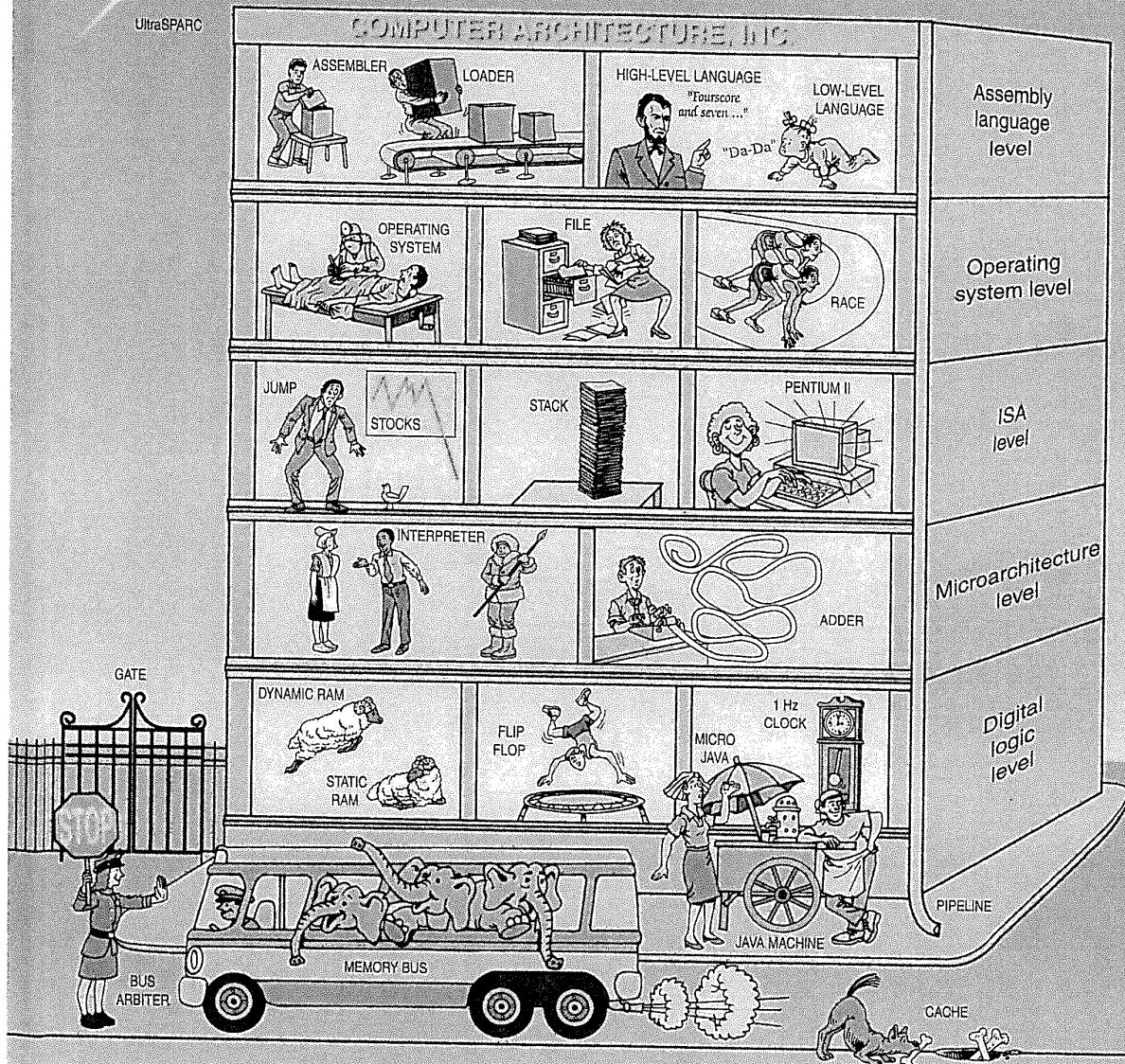


Fourth Edition

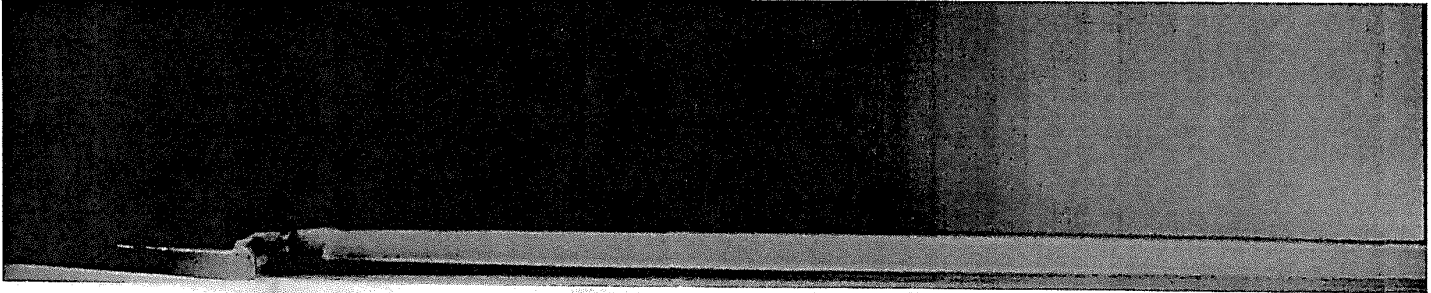
STRUCTURED COMPUTER ORGANIZATION

Andrew S. Tanenbaum

UltraSPARC



Samsung v. Techiya
IPR2024-01003
Exhibit 1035



STRUCTURED COMPUTER ORGANIZATION

Other bestselling titles by Andrew S. Tanenbaum

Computer Networks, 3rd edition

This widely-read classic, now in its third edition, provides the ideal introduction to today's and tomorrow's networks. It explains in detail how modern networks are structured. Starting with the physical layer and working up to the application layer, the book covers a vast number of important topics, including wireless communication, fiber optics, data link protocols, Ethernet, routing algorithms, network performance, security, DNS, electronic mail, USENET news, the World Wide Web, and multimedia. The book has especially thorough coverage of TCP/IP and the Internet.

Operating Systems: Design and Implementation, 2nd edition

This popular text on operating systems is the only book covering both the principles of operating systems and their application to a real system. All the traditional operating systems topics are covered in detail. In addition, the principles are carefully illustrated with MINIX, a free POSIX-based UNIX-like operating system for personal computers. Each book contains a free CD-ROM containing the complete MINIX system, including all the source code. The source code is listed in an appendix to the book and explained in detail in the text.

Modern Operating Systems

This bestselling text presents the basics of both single-processor and distributed computer systems. It covers all the traditional topics, including processes, memory management, and file systems, as well as key issues in distributed systems including the client-server model, remote procedure calls, threads, and distributed file servers. The principles used in the book are illustrated using two extensive single-processor and two distributed systems.

Distributed Operating Systems

This text covers the fundamental concepts of distributed operating systems. Key topics include communication and synchronization, processes and processors, distributed shared memory, distributed file systems, and distributed real-time systems. The principles are illustrated using four chapter-long examples.

STRUCTURED COMPUTER ORGANIZATION

FOURTH EDITION

ANDREW S. TANENBAUM

*Vrije Universiteit
Amsterdam, The Netherlands*

With contributions from
JAMES R. GOODMAN

*University of Wisconsin
Madison, WI*

PRENTICE HALL

UPPER SADDLE RIVER, NEW JERSEY 07458

Library of Congress Cataloging-in-Publication Data

Tanenbaum, Andrew S.
Structured computer organization /
Andrew S. Tanenbaum. — 4th ed.
p. cm.
Includes bibliographical references and index.
ISBN: 0-13-095990-1
1. Computer programming. 2. Computer organization. I. Title.
QA76.6.T38 1999
004.2'2—dc21

98-42130
CIP

Publisher: **ALAN APT**
Development editor: **SONDRA CHAVEZ**
Editor-in-chief: **MARCIA HORTON**
Production editor: **IRWIN ZUCKER**
Managing editor: **EILEEN CLARK**
Manufacturing buyer: **PAT BROWN**
Copy editor: **MARTHA WILLIAMS**
Director of production and manufacturing: **DAVID W. RICCARDI**
Composition and interior design: **ANDREW S. TANENBAUM**
Cover concept: **ANDREW S. TANENBAUM**
Cover director: **ANN FRANCE**
Cover illustrator: **DON MARTINETTI, DM GRAPHICS, INC.**
Editorial assistant: **TONI HOLM**

© 1999, 1990, 1984, 1976 by Prentice Hall, Inc.
Upper Saddle River, NJ 07458

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

TRADEMARK INFORMATION: Windows NT is a registered trademark of Microsoft Corporation. UltraSPARC and Java are trademarks of Sun Microsystems, Inc.. Pentium II is a registered trademark of Intel Corporation. UNIX is a registered trademark of The Open Group. PostScript is a registered trademark of Adobe Systems Inc.

Printed in the United States of America

10 9 8 7 6 5 4

ISBN 0-13-095990-1

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall of Canada, Inc., *Toronto*
Prentice-Hall Hispanoamericana, S. A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Pearson Education Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

To Suzanne, Barbara, Marvin, Bram, and the memory of Sweetie π

CONTENTS

PREFACE

xvi

1 INTRODUCTION

1

- 1.1 STRUCTURED COMPUTER ORGANIZATION 2
 - 1.1.1 Languages, Levels, and Virtual Machines 2
 - 1.1.2 Contemporary Multilevel Machines 4
 - 1.1.3 Evolution of Multilevel Machines 8
- 1.2 MILESTONES IN COMPUTER ARCHITECTURE 13
 - 1.2.1 The Zeroth Generation—Mechanical Computers (1642-1945) 13
 - 1.2.2 The First Generation—Vacuum Tubes (1945-1955) 16
 - 1.2.3 The Second Generation—Transistors (1955-1965) 19
 - 1.2.4 The Third Generation—Integrated Circuits (1965-1980) 21
 - 1.2.5 The Fourth Generation—Very Large Scale Integration (1980-?) 23
- 1.3 THE COMPUTER ZOO 24
 - 1.3.1 Technological and Economic Forces 25
 - 1.3.2 The Computer Spectrum 26
- 1.4 EXAMPLE COMPUTER FAMILIES 29
 - 1.4.1 Introduction to the Pentium II 29
 - 1.4.2 Introduction to the UltraSPARC II 31
 - 1.4.3 Introduction to the picoJava II 34
- 1.5 OUTLINE OF THIS BOOK 36

vii

2 COMPUTER SYSTEMS ORGANIZATION**39**

- 2.1 PROCESSORS 39
 - 2.1.1 CPU Organization 40
 - 2.1.2 Instruction Execution 42
 - 2.1.3 RISC versus CISC 46
 - 2.1.4 Design Principles for Modern Computers 47
 - 2.1.5 Instruction-Level Parallelism 49
 - 2.1.6 Processor-Level Parallelism 53
- 2.2 PRIMARY MEMORY 56
 - 2.2.1 Bits 56
 - 2.2.2 Memory Addresses 57
 - 2.2.3 Byte Ordering 58
 - 2.2.4 Error-Correcting Codes 61
 - 2.2.5 Cache Memory 65
 - 2.2.6 Memory Packaging and Types 67
- 2.3 SECONDARY MEMORY 68
 - 2.3.1 Memory Hierarchies 69
 - 2.3.2 Magnetic Disks 70
 - 2.3.3 Floppy Disks 73
 - 2.3.4 IDE Disks 73
 - 2.3.5 SCSI Disks 75
 - 2.3.6 RAID 76
 - 2.3.7 CD-ROMs 80
 - 2.3.8 CD-Recordables 84
 - 2.3.9 CD-Rewritables 86
 - 2.3.10 DVD 86
- 2.4 INPUT/OUTPUT 89
 - 2.4.1 Buses 89
 - 2.4.2 Terminals 91
 - 2.4.3 Mice 99
 - 2.4.4 Printers 101
 - 2.4.5 Modems 106
 - 2.4.6 Character Codes 109
- 2.5 SUMMARY 113

3 THE DIGITAL LOGIC LEVEL**117**

- 3.1 GATES AND BOOLEAN ALGEBRA 117
 - 3.1.1 Gates 118
 - 3.1.2 Boolean Algebra 120
 - 3.1.3 Implementation of Boolean Functions 122
 - 3.1.4 Circuit Equivalence 123
- 3.2 BASIC DIGITAL LOGIC CIRCUITS 128
 - 3.2.1 Integrated Circuits 128
 - 3.2.2 Combinational Circuits 129
 - 3.2.3 Arithmetic Circuits 134
 - 3.2.4 Clocks 139
- 3.3 MEMORY 141
 - 3.3.1 Latches 141
 - 3.3.2 Flip-Flops 143
 - 3.3.3 Registers 145
 - 3.3.4 Memory Organization 146
 - 3.3.5 Memory Chips 150
 - 3.3.6 RAMs and ROMs 152
- 3.4 CPU CHIPS AND BUSES 154
 - 3.4.1 CPU Chips 154
 - 3.4.2 Computer Buses 156
 - 3.4.3 Bus Width 159
 - 3.4.4 Bus Clocking 160
 - 3.4.5 Bus Arbitration 165
 - 3.4.6 Bus Operations 167
- 3.5 EXAMPLE CPU CHIPS 170
 - 3.5.1 The Pentium II 170
 - 3.5.2 The UltraSPARC II 176
 - 3.5.3 The picoJava II 179
- 3.6 EXAMPLE BUSES 181
 - 3.6.1 The ISA Bus 181
 - 3.6.2 The PCI Bus 183
 - 3.6.3 The Universal Serial Bus 189

- 3.7 INTERFACING 193
 - 3.7.1 I/O Chips 193
 - 3.7.2 Address Decoding 195

- 3.8 SUMMARY 198

4 THE MICROARCHITECTURE LEVEL 203

- 4.1 AN EXAMPLE MICROARCHITECTURE 203
 - 4.1.1 The Data Path 204
 - 4.1.2 Microinstructions 211
 - 4.1.3 Microinstruction Control: The Mic-1 213
- 4.2 AN EXAMPLE ISA: IJVM 218
 - 4.2.1 Stacks 218
 - 4.2.2 The IJVM Memory Model 220
 - 4.2.3 The IJVM Instruction Set 222
 - 4.2.4 Compiling Java to IJVM 226
- 4.3 AN EXAMPLE IMPLEMENTATION 227
 - 4.3.1 Microinstructions and Notation 227
 - 4.3.2 Implementation of IJVM Using the Mic-1 232
- 4.4 DESIGN OF THE MICROARCHITECTURE LEVEL 243
 - 4.4.1 Speed versus Cost 243
 - 4.4.2 Reducing the Execution Path Length 245
 - 4.4.3 A Design with Prefetching: The Mic-2 253
 - 4.4.4 A Pipelined Design: The Mic-3 253
 - 4.4.5 A Seven-Stage Pipeline: The Mic-4 260
- 4.5 IMPROVING PERFORMANCE 264
 - 4.5.1 Cache Memory 265
 - 4.5.2 Branch Prediction 270
 - 4.5.3 Out-of-Order Execution and Register Renaming 276
 - 4.5.4 Speculative Execution 281
- 4.6 EXAMPLES OF THE MICROARCHITECTURE LEVEL 283
 - 4.6.1 The Microarchitecture of the Pentium II CPU 283
 - 4.6.2 The Microarchitecture of the UltraSPARC-II CPU 288
 - 4.6.3 The Microarchitecture of the picoJava II CPU 291
 - 4.6.4 A Comparison of the Pentium, UltraSPARC, and picoJava 296
- 4.7 SUMMARY 298

5 THE INSTRUCTION SET ARCHITECTURE LEVEL 303

- 5.1 OVERVIEW OF THE ISA LEVEL 305
 - 5.1.1 Properties of the ISA Level 305
 - 5.1.2 Memory Models 307
 - 5.1.3 Registers 309
 - 5.1.4 Instructions 311
 - 5.1.5 Overview of the the Pentium II ISA Level 311
 - 5.1.6 Overview of the the UltraSPARC II ISA Level 313
 - 5.1.7 Overview of the Java Virtual Machine 317
- 5.2 DATA TYPES 318
 - 5.2.1 Numeric Data Types 319
 - 5.2.2 Nonnumeric Data Types 319
 - 5.2.3 Data Types on the Pentium II 320
 - 5.2.4 Data Types on the UltraSPARC II 321
 - 5.2.5 Data Types on the Java Virtual Machine 321
- 5.3 INSTRUCTION FORMATS 322
 - 5.3.1 Design Criteria for Instruction Formats 322
 - 5.3.2 Expanding Opcodes 325
 - 5.3.3 The Pentium II Instruction Formats 327
 - 5.3.4 The UltraSPARC II Instruction Formats 328
 - 5.3.5 The JVM Instruction Formats 330
- 5.4 ADDRESSING 332
 - 5.4.1 Addressing Modes 333
 - 5.4.2 Immediate Addressing 334
 - 5.4.3 Direct Addressing 334
 - 5.4.4 Register Addressing 334
 - 5.4.5 Register Indirect Addressing 335
 - 5.4.6 Indexed Addressing 336
 - 5.4.7 Based-Indexed Addressing 338
 - 5.4.8 Stack Addressing 338
 - 5.4.9 Addressing Modes for Branch Instructions 341
 - 5.4.10 Orthogonality of Opcodes and Addressing Modes 342
 - 5.4.11 The Pentium II Addressing Modes 344
 - 5.4.12 The UltraSPARC II Addressing Modes 346
 - 5.4.13 The JVM Addressing Modes 346
 - 5.4.14 Discussion of Addressing Modes 347
- 5.5 INSTRUCTION TYPES 348
 - 5.5.1 Data Movement Instructions 348

- 5.5.2 Dyadic Operations 349
- 5.5.3 Monadic Operations 350
- 5.5.4 Comparisons and Conditional Branches 352
- 5.5.5 Procedure Call Instructions 353
- 5.5.6 Loop Control 354
- 5.5.7 Input/Output 356
- 5.5.8 The Pentium II Instructions 359
- 5.5.9 The UltraSPARC II Instructions 362
- 5.5.10 The picoJava II Instructions 364
- 5.5.11 Comparison of Instruction Sets 369
- 5.6 FLOW OF CONTROL 370
 - 5.6.1 Sequential Flow of Control and Branches 371
 - 5.6.2 Procedures 372
 - 5.6.3 Coroutines 376
 - 5.6.4 Traps 379
 - 5.6.5 Interrupts 379
- 5.7 A DETAILED EXAMPLE: THE TOWERS OF HANOI 383
 - 5.7.1 The Towers of Hanoi in Pentium II Assembly Language 384
 - 5.7.2 The Towers of Hanoi in UltraSPARC II Assembly Language 384
 - 5.7.3 The Towers of Hanoi in JVM Assembly Language 386
- 5.8 THE INTEL IA-64 388
 - 5.8.1 The Problem with the Pentium II 390
 - 5.8.2 The IA-64 Model: Explicitly Parallel Instruction Computing 391
 - 5.8.3 Predication 393
 - 5.8.4 Speculative Loads 395
 - 5.8.5 Reality Check 396
- 5.9 SUMMARY 397
- 6 THE OPERATING SYSTEM MACHINE LEVEL 403**
 - 6.1 VIRTUAL MEMORY 404
 - 6.1.1 Paging 405
 - 6.1.2 Implementation of Paging 407
 - 6.1.3 Demand Paging and the Working Set Model 409
 - 6.1.4 Page Replacement Policy 412
 - 6.1.5 Page Size and Fragmentation 414
 - 6.1.6 Segmentation 415
 - 6.1.7 Implementation of Segmentation 418

- 7.4 LINKING AND LOADING 506
 - 7.4.1 Tasks Performed by the Linker 508
 - 7.4.2 Structure of an Object Module 511
 - 7.4.3 Binding Time and Dynamic Relocation 512
 - 7.4.4 Dynamic Linking 515
- 7.5 SUMMARY 519

8 PARALLEL COMPUTER ARCHITECTURES 523

- 8.1 DESIGN ISSUES FOR PARALLEL COMPUTERS 524
 - 8.1.1 Communication Models 526
 - 8.1.2 Interconnection Networks 530
 - 8.1.3 Performance 539
 - 8.1.4 Software 545
 - 8.1.5 Taxonomy of Parallel Computers 551
- 8.2 SIMD COMPUTERS 554
 - 8.2.1 Array Processors 554
 - 8.2.2 Vector Processors 555
- 8.3 SHARED-MEMORY MULTIPROCESSORS 559
 - 8.3.1 Memory Semantics 559
 - 8.3.2 UMA Bus-Based SMP Architectures 564
 - 8.3.3 UMA Multiprocessors Using Crossbar Switches 569
 - 8.3.4 UMA Multiprocessors Using Multistage Switching Networks 571
 - 8.3.5 NUMA Multiprocessors 573
 - 8.3.6 Cache Coherent NUMA Multiprocessors 575
 - 8.3.7 COMA Multiprocessors 585
- 8.4 MESSAGE-PASSING MULTICOMPUTERS 586
 - 8.4.1 MPPs—Massively Parallel Processors 587
 - 8.4.2 COWs—Clusters of Workstations 592
 - 8.4.3 Scheduling 593
 - 8.4.4 Communication Software for Multicomputers 598
 - 8.4.5 Application-Level Shared Memory 601
- 8.5 SUMMARY 609

9 READING LIST AND BIBLIOGRAPHY 613**9.1 SUGGESTIONS FOR FURTHER READING 613**

- 9.1.1 Introduction and General Works 613
- 9.1.2 Computer Systems Organization 614
- 9.1.3 The Digital Logic Level 615
- 9.1.4 The Microarchitecture Level 616
- 9.1.5 The Instruction Set Architecture Level 617
- 9.1.6 The Operating System Machine Level 617
- 9.1.7 The Assembly Language Level 618
- 9.1.8 Parallel Computer Architectures 618
- 9.1.9 Binary and Floating-Point Numbers 620

9.2 ALPHABETICAL BIBLIOGRAPHY 620**A BINARY NUMBERS 631**

- A.1 FINITE-PRECISION NUMBERS 631
- A.2 RADIX NUMBER SYSTEMS 633
- A.3 CONVERSION FROM ONE RADIX TO ANOTHER 635
- A.4 NEGATIVE BINARY NUMBERS 637
- A.5 BINARY ARITHMETIC 640

B FLOATING-POINT NUMBERS 643

- B.1 PRINCIPLES OF FLOATING POINT 644
- B.2 IEEE FLOATING-POINT STANDARD 754 646

INDEX 653

2

COMPUTER SYSTEMS ORGANIZATION

A digital computer consists of an interconnected system of processors, memories, and input/output devices. This chapter is an introduction to these three components and to their interconnection, as background for the detailed examination of specific levels in the five succeeding chapters. Processors, memories, and input/output are key concepts and will be present at every level, so we will start our study of computer architecture by looking at all three in turn.

2.1 PROCESSORS

The organization of a simple bus-oriented computer is shown in Fig. 2-1. The **CPU (Central Processing Unit)** is the “brain” of the computer. Its function is to execute programs stored in the main memory by fetching their instructions, examining them, and then executing them one after another. The components are connected by a **bus**, which is a collection of parallel wires for transmitting address, data, and control signals. Buses can be external to the CPU, connecting it to memory and I/O devices, but also internal to the CPU, as we will see shortly.

The CPU is composed of several distinct parts. The control unit is responsible for fetching instructions from main memory and determining their type. The arithmetic logic unit performs operations such as addition and Boolean AND needed to carry out the instructions.

The CPU also contains a small, high-speed memory used to store temporary results and certain control information. This memory is made up of a number of

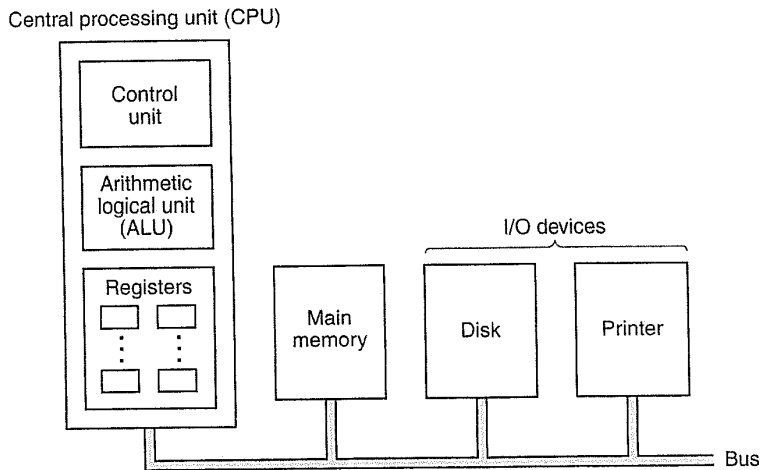


Figure 2-1. The organization of a simple computer with one CPU and two I/O devices.

registers, each of which has a certain size and function. Usually, all the registers have the same size. Each register can hold one number, up to some maximum determined by the size of the register. Registers can be read and written at high speed since they are internal to the CPU.

The most important register is the **Program Counter (PC)**, which points to the next instruction to be fetched for execution. The name “program counter” is somewhat misleading because it has nothing to do with *counting* anything, but the term is universally used. Also important is the **Instruction Register (IR)**, which holds the instruction currently being executed. Most computers have numerous other registers as well, some of them general purpose as well as some for specific purposes.

2.1.1 CPU Organization

The internal organization of part of a typical von Neumann CPU is shown in Fig. 2-2 in more detail. This part is called the **data path** and consists of the registers (typically 1 to 32), the **ALU (Arithmetic Logic Unit)**, and several buses connecting the pieces. The registers feed into two ALU input registers, labeled *A* and *B* in the figure. These registers hold the ALU input while the ALU is computing. The data path is very important in all machines and we will discuss it at great length throughout this book.

The ALU itself performs addition, subtraction, and other simple operations on its inputs, thus yielding a result in the output register. This output register can be

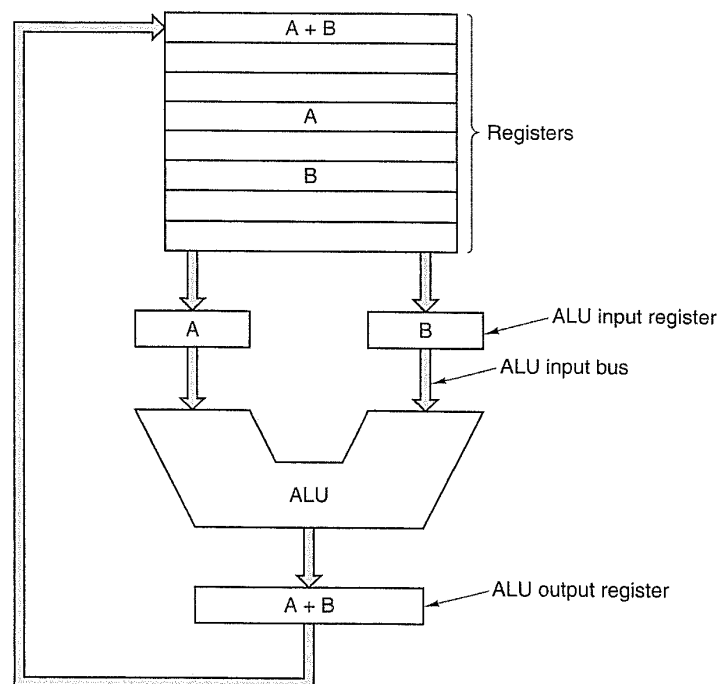


Figure 2-2. The data path of a typical von Neumann machine.

stored back into a register. Later on, the register can be written (i.e., stored) into memory, if desired. Not all designs have the *A*, *B*, and output registers. In the example, addition is illustrated.

Most instructions can be divided into one of two categories: register-memory or register-register. Register-memory instructions allow memory words to be fetched into registers, where they can be used as ALU inputs in subsequent instructions, for example. ("Words" are the units of data moved between memory and registers. A word might be an integer. We will discuss memory organization later in this chapter.) Other register-memory instructions allow registers to be stored back into memory.

The other kind of instruction is register-register. A typical register-register instruction fetches two operands from the registers, brings them to the ALU input registers, performs some operation on them, for example, addition or Boolean AND, and stores the result back in one of the registers. The process of running two operands through the ALU and storing the result is called the **data path cycle** and is the heart of most CPUs. To a considerable extent, it defines what the machine can do. The faster the data path cycle is, the faster the machine runs.

2.1.2 Instruction Execution

The CPU executes each instruction in a series of small steps. Roughly speaking, the steps are as follows

1. Fetch the next instruction from memory into the instruction register.
2. Change the program counter to point to the following instruction.
3. Determine the type of instruction just fetched.
4. If the instruction uses a word in memory, determine where it is.
5. Fetch the word, if needed, into a CPU register.
6. Execute the instruction.
7. Go to step 1 to begin executing the following instruction.

This sequence of steps is frequently referred to as the **fetch-decode-execute** cycle. It is central to the operation of all computers.

This description of how a CPU works closely resembles a program written in English. Figure 2-3 shows this informal program rewritten as a Java method (i.e., procedure) called *interpret*. The machine being interpreted has two registers visible to user programs: the program counter (PC), for keeping track of the address of the next instruction to be fetched, and the accumulator (AC), for accumulating arithmetic results. It also has internal registers for holding the current instruction during its execution (*instr*), the type of the current instruction (*instr_type*), the address of the instruction's operand (*data_loc*), and the current operand itself (*data*). Instructions are assumed to contain a single memory address. The memory location addressed contains the operand, for example, the data item to add to the accumulator.

The very fact that it is possible to write a program that can imitate the function of a CPU shows that a program need not be executed by a "hardware" CPU consisting of a box full of electronics. Instead, a program can be carried out by having another program fetch, examine, and execute its instructions. A program (such as Fig. 2-3) that fetches, examines, and executes the instructions of another program is called an **interpreter**, as mentioned in Chap. 1.

This equivalence between hardware processors and interpreters has important implications for computer organization and the design of computer systems. After having specified the machine language, *L*, for a new computer, the design team can decide whether they want to build a hardware processor to execute programs in *L* directly or whether they want to write an interpreter to interpret programs in *L* instead. If they choose to write an interpreter, they must also provide some hardware machine to run the interpreter. Certain hybrid constructions are also possible, with some hardware execution as well as some software interpretation.