Computer Systems & Architecture



BIBLIOTEX
Digital Library

2nd Edition

COMPUTER SYSTEMS & ARCHITECTURE

2ND EDITION

BIBLIOTEX
Digital Library

www.bibliotex.com

COMPUTER SYSTEMS & ARCHITECTURE

2nd Edition



www.bibliotex.com
email: info@bibliotex.com

e-book Edition 2022
ISBN: 978-1-98467-618-4 (e-book)

This book contains information obtained from highly regarded resources. Reprinted material sources are indicated. Copyright for individual articles remains with the authors as indicated and published under Creative Commons License. A Wide variety of references are listed. Reasonable efforts have been made to publish reliable data and views articulated in the chapters are those of the individual contributors, and not necessarily those of the editors or publishers. Editors or publishers are not responsible for the accuracy of the information in the published chapters or consequences of their use. The publisher assumes no responsibility for any damage or grievance to the persons or property arising out of the use of any materials, instructions, methods or thoughts in the book. The editors and the publisher have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission has not been obtained. If any copyright holder has not been acknowledged, please write to us so we may rectify.

Notice: Registered trademark of products or corporate names are used only for explanation and identification without intent of infringement.

© 2022 3G E-learning LLC

In Collaboration with 3G E-Learning LLC. Originally Published in printed book format by 3G E-Learning LLC with ISBN 978-1-98465-911-8

EDITORIAL BOARD



Aleksandar Mratinković was born on May 5, 1988 in Arandjelovac, Serbia. He has graduated on Economic high school (2007), The College of Tourism in Belgrade (2013), and also has a master degree of Psychology (Faculty of Philosophy, University of Novi Sad). He has been engaged in different fields of psychology (Developmental Psychology, Clinical Psychology, Educational Psychology and Industrial Psychology) and has published several scientific works.



Dan Piestun (PhD) is currently a startup entrepreneur in Israel working on the interface of Agriculture and Biomedical Sciences and was formerly president-CEO of the National Institute of Agricultural Research (INIA) in Uruguay. Dan is a widely published scientist who has received many honours during his career including being a two-time recipient of the Amit Golda Meir Prize from the Hebrew University of Jerusalem, his areas of expertise includes stem cell molecular biology, plant and animal genetics and bioinformatics. Dan's passion for applied science and technological solutions did not stop him from pursuing a deep connection to the farmer, his family and nature. Among some of his interest and practices counts enjoying working as a beekeeper and onboard fishing.



Hazem Shawky Fouda has a PhD. in Agriculture Sciences, obtained his PhD. From the Faculty of Agriculture, Alexandria University in 2008. He is working in Cotton Arbitration & Testing General Organization (CATGO).



Felecia Killings is the Founder and CEO of LiyahAmore Publishing, a publishing company committed to providing technical and educational services and products to Christian Authors. She operates as the Senior Editor and Writer, the Senior Writing Coach, the Content Marketing Specialist, Editor-in-Chief to the company's quarterly magazine, the Executive and Host of an international virtual network, and the Executive Director of the company's online school for Authors. She is a former high-school English instructor and professional development professor. She possesses a Master of Arts degree in Education and a Bachelor's degree in English and African American studies.



Dr. Sandra El Hajj, Ph.D. in Health Sciences from Nova Southeastern University, Florida, USA is a health professional specialized in Preventive and Global Health. With her 12 years of education obtained from one of the most prominent universities in Beirut, in addition to two leading universities in the State of Florida (USA), Dr. Sandra made sure to incorporate interdisciplinary and multicultural approaches in her work. Her long years of studies helped her create her own miniature world of knowledge linking together the healthcare field with Medical Research, Statistics, Food Technology, Environmental & Occupational Health, Preventive Health and most noteworthy her precious last degree of Global Health. Till today, she is the first and only doctor specialized in Global Health in the Middle East area.



Fozia Parveen has a Dphil in Sustainable Water Engineering from the University of Oxford. Prior to this she has received MS in Environmental Sciences from National University of Science and Technology (NUST), Islamabad Pakistan and BS in Environmental Sciences from Fatima Jinnah Women University (FJWU), Rawalpindi.



Igor Krunic 2003-2007 in the School of Economics. After graduating in 2007, he went on to study at The College of Tourism, at the University of Belgrade where he got his bachelor degree in 2010. He was active as a third-year student representative in the student parliament.Then he went on the Faculty of science, at the University of Novi Sad where he successfully defended his master's thesis in 2013. The crown of his study was the work titled Opportunities for development of cultural tourism in Cacak". Later on, he became part of a multinational company where he got promoted to a deputy director of logistic. Nowadays he is a consultant and writer of academic subjects in the field of tourism.



Dr. Jovan Pehcevski obtained his PhD in Computer Science from RMIT University in Melbourne, Australia in 2007. His research interests include big data, business intelligence and predictive analytics, data and information science, information retrieval, XML, web services and service-oriented architectures, and relational and NoSQL database systems. He has published over 30 journal and conference papers and he also serves as a journal and conference reviewer. He is currently working as a Dean and Associate Professor at European University in Skopje, Macedonia.



Dr. Tanjina Nur finished her PhD in Civil and Environmental Engineering in 2014 from University of Technology Sydney (UTS). Now she is working as Post-Doctoral Researcher in the Centre for Technology in Water and Wastewater (CTWW) and published about eight International journal papers with 80 citations. Her research interest is wastewater treatment technology using adsorption process.



Stephen obtained his PhD from the University of North Carolina at Charlotte in 2013 where his graduate research focused on cancer immunology and the tumor microenvironment. He received postdoctoral training in regenerative and translational medicine, specifically gastrointestinal tissue engineering, at the Wake Forest Institute of Regenerative Medicine. Currently, Stephen is an instructor for anatomy and physiology and biology at Forsyth Technical Community College.



Michelle holds a Masters of Business Administration from the University of Phoenix, with a concentration in Human Resources Management. She is a professional author and has had numerous articles published in the Henry County Times and has written and revised several employee handbooks for various YMCA organizations throughout the United States.

HOW TO USE THE BOOK

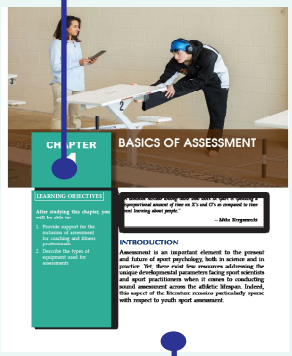
This book has been divided into many chapters. Chapter gives the motivation for this book and the use of templates. The text is presented in the simplest language. Each paragraph has been arranged under a suitable heading for easy retention of concept. Keywords are the words that academics use to reveal the internal structure of an author's reasoning. Review questions at the end of each chapter ask students to review or explain the concepts. References provides the reader an additional source through which he/she can obtain more information regarding the topic.

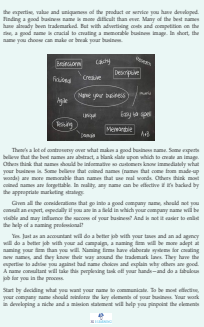
LEARNING OBJECTIVES

See what you are going to cover and what you should already know at the start of each chapter

ABOUT THIS CHAPTER

An introduction is a beginning of section which states the purpose and goals of the topics which are discussed in the chapter. It also starts the topics in brief.



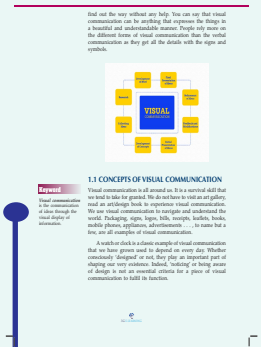


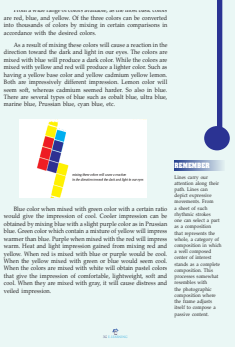
REMEMBER

This revitalizes a must read information of the topic.

KEYWORDS

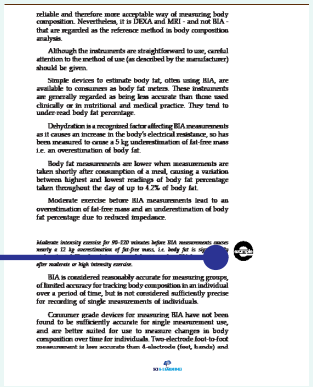
This section contains some important definitions that are discussed in the chapter. A keyword is an index entry that identifies a specific record or document. It also gives the extra information to the reader and an easy way to remember the word definition.





DID YOU KNOW?

This section equip readers the interesting facts and figures of the topic.



reliable and therefore more acceptable way of measuring body composition. Nevertheless, it is DEXA and MRI - and not BIA - that are regarded as the reference method in body composition analysis.

Although the instruments are straightforward to use, careful attention to the method of use (as described by the manufacturer) should be given.

Simple devices to estimate body fat, often using BIA, are available to consumers as body fat meters. These instruments are generally regarded as being less accurate than those used clinically or in nutritional and medical practice. They tend to under-read body fat percentage.

Dehydration is a recognized factor affecting BIA measurements as it causes an increase in the body's electrical resistance, so has been measured to cause a 5 kg underestimation of fat-free mass i.e. an overestimation of body fat.

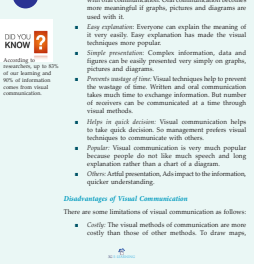
Body fat measurements are lower when measurements are taken shortly after consumption of a meal, causing a variation between highest and lowest readings of body fat percentage taken throughout the day of up to 4.2% of body fat.

Moderate exercise before BIA measurements lead to an overestimation of fat-free mass and an underestimation of body fat percentage due to reduced impedance.

Moderate intensity exercise for 90-120 minutes before BIA measurements causes nearly a 12 kg overestimation of fat-free mass, i.e. body fat is significantly underestimated after moderate or high intensity exercise.

BIA is considered reasonably accurate for measuring groups, of limited accuracy for tracking body composition in an individual over a period of time, but in not considered sufficiently precise for recording of single measurements of individuals.

Consumer grade devices for measuring BIA have not been found to be sufficiently accurate for single measurement use, and are better suited for use to measure changes in body composition over time for individuals. Two-electrode foot-to-foot measurement is less accurate than 4-electrode (foot, hands) and



DID YOU KNOW?

According to researchers, up to 83% of our learning and 90% of information comes from visual communication.

Advantages of Visual Communication

Now-a-days, most of the business organizations are using visual techniques to present the information. It is becoming very popular day by day. Visual presentation is beneficial for many reasons.

Some of them are as follows:

- *Effective for illiterate receiver:* If the receivers are illiterate, the visual communication will be more effective to exchange information. They can easily understand the information that is presented visually.
- *Helps in oral communication:* Visual techniques can be used with oral communication. Oral communication becomes more meaningful if graphs, pictures and diagrams are used with it.
- *Easy explanation:* Everyone can explain the meaning of it very easily. Easy explanation has made the visual techniques more popular.
- *Simple presentation:* Complex information, data and figures can be easily presented very simply on graphs, pictures and diagrams.
- *Prevents wastage of time:* Visual techniques help to prevent the wastage of time. Written and oral communication takes much time to exchange information. But number of receivers can be communicated at a time through visual methods.
- *Helps in quick decision:* Visual communication helps to take quick decision. So management prefers visual techniques to communicate with others.
- *Popular:* Visual communication is very much popular because people do not like much speech and long explanation rather than a chart of a diagram.
- *Others:* Artful presentation, Ads impact to the information, quicker understanding.

Disadvantages of Visual Communication

There are some limitations of visual communication as follows:

- *Costly:* The visual methods of communication are more costly than those of other methods. To draw maps,

EXAMPLE

The book cabinets' examples to illustrate specific ideas in each chapter.

ROLE MODEL

A biography of someone who has/had acquired remarkable success in their respective field as Role Models are important because they give us the ability to imagine our future selves.



ROLE MODEL

CLAUDE GARAMOND

Claude Garamont (ca. 1510 – 1561), known commonly as **Claude Garamond**, was a French type designer, publisher and punch-cutter based in Paris. Garamond worked as an engraver of punches, the masters used to stamp matrices, the moulds used to cast metal type. He worked in the tradition of what is now called old-style serif letter design, that produced letters with a relatively organic structure resembling handwriting with a pen but with a slightly more structured and upright design. Considered one of the leading type designers of all time, he is recognized to this day for the elegance of his typefaces.

Garamond was one of the first independent punchcutters, specializing in type design and punch-cutting as a service to others rather than working in house for a specific printer. His career therefore helped to define the future of commercial printing with typefounding as a distinct industry to printing books.

Early Life and Background

Garamond's early life has been the subject of some research and considerable uncertainty. Dates as early as 1480 and as late as c. 1510 have been proposed for his birth, the latter being preferred by the French ministry of culture. In favor of a later date, his will of 1561 states that his mother was then still alive. He married twice, to Guillemette Gaulliier and, after her death, to Ysabeau Le Fevre. Garamond may have apprenticed with Antoine Augereau and was perhaps also trained by Simon de Colines. He later worked with Geoffroy Tory, whose interests in humanist typography and the ancient Greek capital letterforms, or majuscules, may have informed Garamond's work.

Garamond came to prominence around 1540, when three of his Greek typefaces (now called the Grecs du roi (1541)) were requested for a royally-ordered book series by Robert Estienne. Garamond based these types, now known as the Grecs du roi, on the handwriting of Angelo Vergecio, the King's Librarian at Fontainebleau. The result is an immensely complicated set of

CASE STUDY

This reveals what students need to create and provide an opportunity for the development of key skills such as communication, group working and problem solving.

CASE STUDY

EFFECT OF THERAGUN ON THE IMPROVEMENT OF BACK FLEXIBILITY

Muscle tightness may be connected to postural instability. Both can contribute to various musculoskeletal conditions. Reduced extensibility resultant from increased hamstring stiffness could be a probable causative factor to low back injuries. Considering that forward bending is one of the mainly common movements in daily activities, shortened hamstrings may increase the risk of injury to the spine from mechanical stresses. Flexibility dysfunction is a extensive problem faced by common as well as sportsmen, especially in case of hamstring group of muscle. Vibration therapy improves muscular strength, power improvement and kinesthetic awareness.

History

We describe a 25-year-old male patient. He is a dentist. His height was 162 centimeters, weight 65 kilograms and body mass index (BMI) was 24.8. The patient was seen by a female physiotherapist and enrolled for daily treatment. He complained of back pain that got aggravated with forwarding bending activity and prolonged sitting. He also complained of difficulty in horse riding. He belonged to a high socioeconomic class and fair family and social support. He had no history of trauma.

Physical Examination

His Back movements were restricted. There were a bilateral Hamstring tightness and reduced back flexibility.

Procedure

Ethical approval was granted from the Institutional Ethical Committee and the Patient gave informed written consent. His demographic data, physical examination and the intensity of pain was done with use of numeric Pain rating Scale score was noted. Flexibility measurement was done with the use of sit and reach test and hamstrings tightness measurement was done with the use of a 90-90 straight leg raising test. Activity difficulty was measure by the use of the patient -specific functional scale.

KNOWLEDGE CHECK

This is given to the students for progress check at the end of each chapter.

KNOWLEDGE CHECK

1. Which property is used to specify typefaces?
   a. font-family
   b. font-name
   c. font-face
   d. font-type
2. System independent fonts is called as generic fonts.
   a. True
   b. False
3. RGB colors on internet applications are called
   a. safe colors
   b. colors space
   c. web colors
   d. safe web colors
4. Color model is also called
   a. color system
   b. color space
   c. color area
   d. Both A and B
5. Which of these adds richness and visual interest to all types of design?
   a. Economy of line
   b. Typography
   c. Pattern
   d. Pixels
6. Which of the following is a commonly accepted guideline in typography?
   a. Use a serif typeface for headings and a sans serif typeface for text
   b. Use a sans serif typeface for headings and a serif typeface for text
   c. Use a sans serif typeface for both headings and text
   d. Use a serif typeface for both headings and text
7. The "W" was what repeated letter?
   a. A
   b. V

REVIEW QUESTIONS

This section is to analyze the knowledge and ability of the reader.

REVIEW QUESTIONS

1. What is the typography?
2. Discuss about the RGB (CMY) color model.
3. Explain about typographic hierarchy.
4. What do you understand by the typeface?
5. Write short notes on:
   a. PostScript fonts
   b. TrueType fonts
   c. OpenType fonts

Check Your Result

1. (a) 2. (a) 3. (d) 4. (d) 5. (b)
6. (b) 7. (c) 8. (a) 9. (c) 10. (b)

REFERENCES

References refer those books which discuss the topics given in the chapters in almost same manner.

REFERENCES

1. Department of Revenue Washington State. "Domain Name Registration Services." Accessed Dec. 20, 2019.
2. Domain Name Wire podcast. "Frank Schilling Explains Price Hike – DNW Podcast #127." Accessed Dec. 20, 2019.
3. Domain Name Wire. "ICANN Shuts Down Alpnames Domain Name Registrar." Accessed Dec. 20, 2019.
4. Domain.com. "Domain Registration Agreement." Accessed Dec. 20, 2019.
5. ICANN. "FAQs: How Long Does a Registration Last? Can It be Renewed?" Accessed Dec. 20, 2019.
6. No-IP Knowledge Base. "How Long Until the Domain I just Registered is Active?" Accessed Dec. 20, 2019.
7. PRNewswire. "GoDaddy Acquires Worldwide Media Inc. Domain Name Portfolio." Accessed Dec. 20, 2019.
8. Symbolics.com. "About Symbolics." Accessed Dec. 20, 2019.
9. Verisign. "Verisign Q1 2019 Domain Name Industry Brief." Accessed Dec. 20, 2019.
10. Whois.net. "Terms and Conditions of Whois.net." Accessed Dec. 20, 2019.

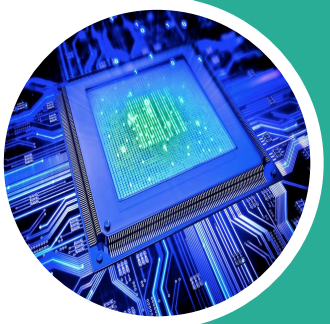
TABLE OF CONTENTS

Preface xv

Chapter 1 Concept of Computer 1

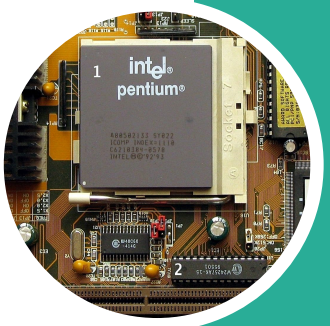


Introduction 1
1.1 history of computers 4
1.2 Number Representation and Binary Code 8
1.2.1 Fixed and Floating Point Number Representation 17
1.2.2 BCD 19
1.2.3 EBCDIC 20
1.2.4 ASCII 21
1.2.5 IEEE Standard 25
1.3 Digital Computer Generation 28
1.3.1 Computer Types and Classifications 34
1.3.2 Functional Units and Their Interconnections 37
1.4 Buses 41
1.4.1 Bus Architecture 42
1.4.2 Types of Buses and Bus Arbitration 45
1.4.3 Bus and Memory Transfer 48
Summary 53
Knowledge Check 54
Review Questions 55
References 56



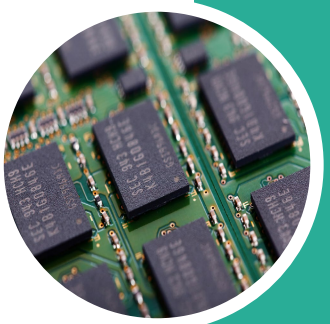
Chapter 2 Central Processing Unit (CPU) 57

Introduction 57
2.1 process and importance of CPU in a computer systems 58
2.1.1 CPU Building Blocks 63
2.1.2 Addressing Modes 66
2.2 Concepts of Interrupts 74
2.2.1 Interrupt Hardware 75
2.2.2 Types of Interrupts 75
2.2.3 Instruction and Execution Interrupt Cycle 76
Summary 86
Knowledge Check 87
Review Questions 88
References 89



Chapter 3 Control Unit 91

Introduction 91
3.1 Instruction Types 92
3.1.1 Formats Instruction Cycles and Sub Cycles 99
3.1.2 Micro-Operations 104
3.1.3 Execution of a Complete Instruction 106
3.2 Hardwire and Micro-Programmed Control 107
3.2.1 Micro Program Sequencing 108
3.2.2 Wide Branch Addressing 113
3.2.3 Microinstruction with Next Address Field 113
3.2.4 Pre-fetching Microinstructions 116
3.2.5 Concept of Horizontal and Vertical Microprogramming 117
Summary 123
Knowledge Check 124
Review Questions 125
References 126



Chapter 4 Focus on Memory 127

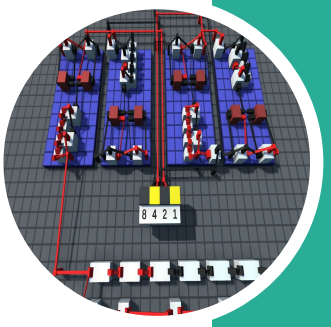
Introduction 127
4.1 Concept of Memory and Memory Hierarchy 128
4.1.1 Characteristics Terms for Various Memory Devices 129
4.2 Main Memory 131
4.2.1 RAM and ROM Chips 133
4.2.2 Memory Address Map 133

4.2.3 Memory Connection to CPU 134
4.3 Auxiliary Memory 134
4.3.1 Magnetic Disks 136
4.3.2 Magnetic Tape 136
4.3.3 CD 137
4.3.4 DVD 137
4.4 Associative Memory 138
4.4.1 Hardware Organization 139
4.4.2 Read Operation 140
4.4.3 Write Operation 141
4.5 Cache Memory 142
4.5.1 Direct Mapping 145
4.5.2 Associative Mapping 147
4.5.3 Set-Associative Mapping 148
4.5.4 Writing into Cache 150
4.6 Virtual Memory 150
4.6.1 The Concept of Virtual Memory in Computer Architecture 151
Summary 160
Knowledge Check 161
Review Questions 162
References 163

Chapter 5 Input and Output Devices 165

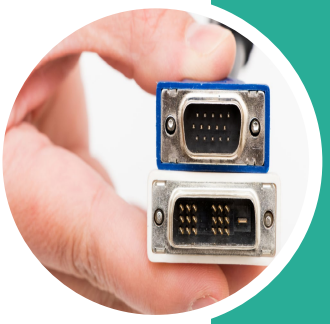


Introduction 165
5.1 Peripheral Devices 166
5.1.1 Mouse 168
5.1.2 Keyboard 169
5.1.3 Printer 170
5.1.4 Monitor 172
5.2 I/O Interface 174
5.2.1 Types of Interfaces 177
5.2.2 FireWire Configurations 179
5.2.3 I/O Ports 183
5.3 Direct Memory Access (DMA) 185
5.3.1 Basic Process of DMA 186
5.3.2 DMA Transfer 187
5.3.3 DMA Controller 189
5.3.4 DMA Transfer Cycle 190
Summary 195
Knowledge Check 196
Review Questions 198
References 199



Chapter 6 Combinational Logic Circuits 201

Introduction **201**
6.1 Standard Combinational Circuit Blocks **202**
6.1.1 Encoder 205
6.1.2 Decoder 207
6.1.3 Multiplexer 208
6.1.4 Demultiplexer 210
6.1.5 Adders 212
6.2 Sequential Logic Circuits **215**
6.2.1 Flip-Flops 217
6.3 Finite State Machines **220**
6.3.1 State Transition Graphs 221
6.4 Registers **222**
6.5 Standard Sequential Logic Circuits **225**
6.5.1 Counters 226
6.5.2 Shift Registers 230
Summary **243**
Knowledge Check **244**
Review Questions **246**
References **247**



Chapter 7 Modes of Data Transfer and Serial Communication 249

Introduction **249**
7.1 Introduction of Modes of Data Transfer **250**
7.1.1 Programmed I/O 254
7.1.2 Interrupt Initiated I/O and Direct Memory Access 255
7.2 Serial Communication **258**
7.2.1 Types of Serial Communication 259
7.2.2 Asynchronous and Synchronous Communication 263
7.2.3 Standard Communication Interfaces 266
7.2.4 Cables that Carry Serial Data 266
Summary **271**
Knowledge Check **272**
Review Questions **273**
References **274**



Chapter 8 Software Architecture and Design 275

Introduction **275**

8.1 Concept Of Software Architecture **276**

8.1.1 Software Design 277

8.1.2 Goals of Architecture 277

8.1.3 Limitations 278

8.1.4 Role of Software Architect 278

8.1.5 Quality Attributes 280

8.1.6 Quality Scenarios 280

8.1.7 Common Quality Attributes 281

8.2 Key Principles of Software Architecture **282**

8.2.1 Architectural Style 282

8.2.2 Common Architectural Design 282

8.2.3 Types of Architecture 283

8.2.4 Architecture Design Process 284

8.2.5 Key Architecture Principles 285

8.3 Architecture Models **288**

8.3.1 UML 288

8.3.2 Architecture View Model 290

8.3.3 Architecture Description Languages (ADLs) 292

8.4 Object-Oriented Paradigm **293**

8.4.1 Introduction to Object-Oriented Paradigm 294

8.4.2 OO Analysis 297

8.4.3 Object-Oriented Design 298

8.5 Data Flow Architecture **300**

8.5.1 Batch Sequential 300

8.5.2 Pipe and Filter Architecture 301

8.5.3 Process Control Architecture 302

8.6 Data-Centered Architecture **303**

8.6.1 Types of Components 304

8.6.2 Repository Architecture Style 304

8.6.3 Blackboard Architecture Style 306

8.7 Hierarchical Architecture **307**

8.7.1 Main-subroutine 308

8.7.2 Master-Slave 309

8.7.3 Virtual Machine Architecture 310

8.7.4 Layered Style 312

8.8 Interaction-Oriented Architecture **313**

8.8.1 Model-View-Controller (MVC) 313

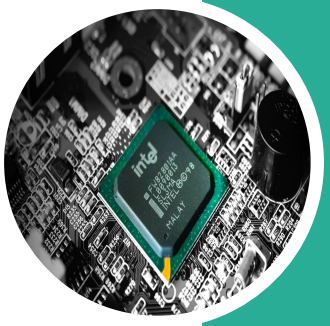
8.8.2 Presentation-Abstraction-Control (PAC) 317

8.9 Distributed Architecture **319**

8.9.1 Client-Server Architecture 321

8.9.2 Multi-Tier Architecture (n-tier Architecture) 323
8.9.3 Broker Architectural Style 325
8.9.4 Service-Oriented Architecture (SOA) 327
8.10 Component-Based Architecture 329
8.10.1 What is a Component? 330
8.10.2 Principles of Component–Based Design 331
8.10.3 Component-Level Design Guidelines 332
8.10.4 Conducting Component-Level Design 332
8.11 User Interface 334
8.11.1 Graphical User Interface 334
8.11.2 Design of User Interface 335
8.11.3 User Interface Development Process 336
8.11.4 Design Considerations of User Interface 337
SUMMARY 343
Knowledge Check 344
Review Questions 345
References 346

Chapter 9 Microprocessor Architecture 347



Introduction 347
9.1 Overview Of Microprocessor 348
9.2 8085 Architecture 351
9.2.1 Bus Structure of 8085 Microprocessor 357
9.2.2 Pin Diagram of 8085 360
9.3 Memory Interfacing 364
9.3.1 IO interfacing to 8085 366
9.4 Interrupts In 8085 370
9.4.1 8085 Interrupt Response Process 370
Summary 375
Knowledge Check 376
Review Questions 377
References 378

Chapter 10 Parallel Processing 379



Introduction 379
10.1 Multiple Processor Organizations 380
10.1.1 Parallel Organizations 382
10.2 Symmetric Multiprocessors 383
10.2.1 Organization 384
10.2.2 Multiprocessor Operating System Design Considerations 387
10.2.3 A Mainframe SMP 388

10.3 Cache Coherence And The Mesi Protocol **390**
10.3.1 Software Solutions 391
10.3.2 Hardware Solutions 392
10.3.3 The MESI Protocol 394
10.4 Multithreading and Chip Multiprocessors **398**
10.4.1 Implicit and Explicit Multithreading 399
10.4.2 Approaches to Explicit Multithreading 400
10.4.3 Example Systems 404
10.5 Clusters **405**
10.5.1 Cluster Configurations 406
10.5.2 Operating System Design Issues 409
10.5.3 Cluster Computer Architecture 410
10.5.4 Blade Servers 412
10.5.5 Clusters Compared to SMP 412
10.6 Nonuniform Memory Access **413**
10.6.1 Motivation 413
10.6.2 Organization 414
10.6.3 NUMA Pros and Cons 416
10.7 Vector Computation **416**
10.7.1 Approaches to Vector Computation 417
10.7.2 IBM 3090 Vector Facility 423
Summary **431**
Knowledge Check **432**
Review Questions **433**
References **434**

Index **435**

PREFACE

Computer is an electronic machine that makes performing any task very easy. In computer, the CPU executes each instruction provided to it, in a series of steps, this series of steps is called Machine Cycle, and is repeated for each instruction. One machine cycle involves fetching of instruction, decoding the instruction, transferring the data, executing the instruction. Computer architecture is a specification detailing how a set of software and hardware technology standards interact to form a computer system or platform. In short, computer architecture refers to how a computer system is designed and what technologies it is compatible with. As with other contexts and meanings of the word architecture, computer architecture is likened to the art of determining the needs of the user/system/technology, and creating a logical design and standards based on those requirements. Modern computer technology requires an understanding of both hardware and software, since the interaction between the two offers a framework for mastering the fundamentals of computing.

Organization of the Book

This edition is comprised of ten chapters. It deals with the design and performance evaluation of advanced/high- performance computer systems. The emphasis is on microprocessors, chip-multiprocessors and memory hierarchy design.

Chapter 1 presents an introduction to computer organization including with history of computers, and number representation and binary code. Furthermore, the digital computer generation and buses in computers are also elaborated.

Chapter 2 discusses the process and importance of CPU in a computer systems with addressing modes and its types.

Chapter 3 covers the control unit needed to perform the role of a multiplexer, in that it takes multiple inputs or outputs and directs them into and out of the pipeline.

Chapter 4 emphasizes the basics of concept of memory and memory hierarchy required for storage and subsequent retrieval of the instructions and data.

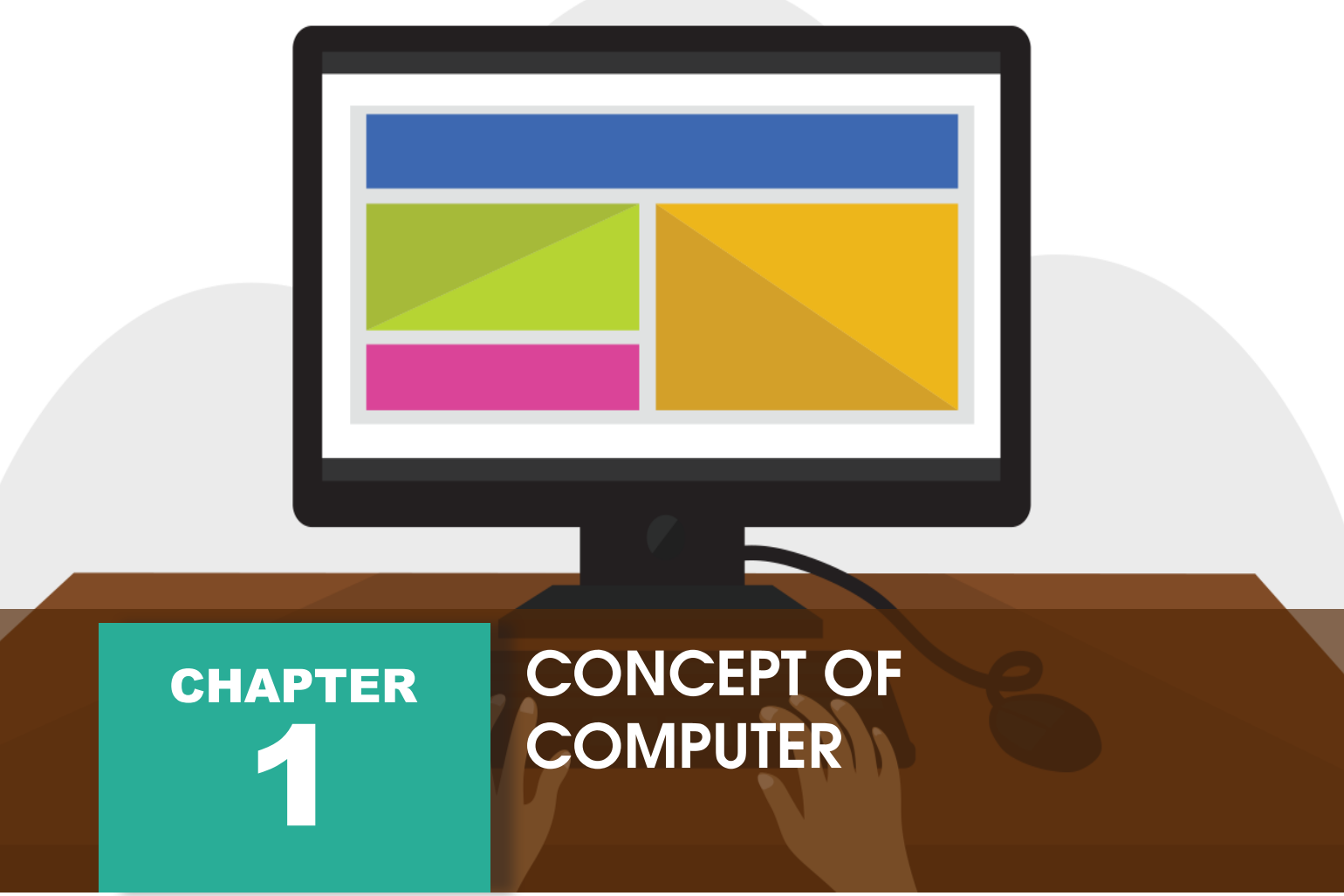
Chapter 5 explores the study of peripheral devices and process of I/O interface.

Chapters 6 and 7 present the coverage of combinational logic circuits and modes of data transfer respectively.

Chapter 8 focuses on software architecture and design. The architecture of a system describes its major components, their relationships (structures), and how they interact with each other.

Chapter 9 shows the microprocessor architecture. Typical microprocessor operations include adding, subtracting, comparing two numbers, and fetching numbers from one area to another.

Chapter 10 highlights the parallel processing. Typical microprocessor operations include adding, subtracting, comparing two numbers, and fetching numbers from one area to another.



CHAPTER 1

CONCEPT OF COMPUTER

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Understand the history of computers
2. Determine the number representation and binary code
3. Explain the digital computer generation and buses in computers

"Computers themselves, and software yet to be developed, will revolutionize the way we learn. "

—Steve Jobs

INTRODUCTION

A computer is a programmable machine. It allows the user to store all sorts of information and then 'process' that information, or data, or carry out actions with the information, such as calculating numbers or organising words.

Computers can be generally classified by size and power, although there can be considerable overlap. Following are descriptions of several different types of computers.

Computer: A machine that can receive and store information and change or process it.

Information: Knowledge that is communicated.

Data *(pl.):* The representation of information in a formalised manner suitable for communication, interpretation and processing, generally by a computer system. *Note:* the term 'raw data' refers to unprocessed information.

Mainframe computers are large-sized, powerful multi-user computers that can support concurrent programs. That means, they can perform different actions or 'processes' at the same time. Mainframe computers can be used by as many as hundreds or thousands of users at the same time. Large organisations may use a mainframe computer to execute large-scale processes such as processing the organisation's payroll.

Mini-computers are mid-sized multi-processing computers. Again, they can perform several actions at the same time and can support from 4 to 200 users simultaneously. In recent years the distinction between mini-computers and small mainframes has become blurred. Often the distinction depends upon how the manufacturer wants to market its machines. Organisations may use a mini-computer for such tasks as managing the information in a small financial system or maintaining a small database of information about registrations or applications.

Workstations are powerful, single-user computers. They have the capacity to store and process large quantities of data, but they are only used by one person at a time. However, workstations are typically linked together to form a computer network called a local area network, which means that several people, such as staff in an office, can communicate with each other and share electronic files and data.

A workstation is similar to a personal computer but is more powerful and often comes with a higher-quality monitor. In terms of computing power, workstations lie in between personal computers and mini-computers. Workstations commonly support applications that require relatively high-quality graphics capabilities and a lot of memory, such as desktop publishing, software development and engineering applications.

Personal computers (PCs), also called microcomputers, are the most popular type of computer in use today. The PC is a small-sized, relatively inexpensive computer designed for an individual user. Today, the world of PCs is basically divided between IBM-compatible and Macintosh-compatible machines, named after the two computer manufacturers. Computers may be called 'desktop' computers, which stay on the desk, or 'laptop' computers, which are lightweight and portable. Organisations and individuals use PCs for a wide range of tasks, including word processing, accounting, desktop publishing, preparation and delivery of presentations, organisation of spreadsheets

and database management. Entry-level PCs are much more powerful than a few years ago, and today there is little distinction between PCs and workstations.

Simply speaking computer is a calculating device. The name is derived from the Latin word computare meaning "to compute", and can be applied as properly to an abacus or an adding machine as to the modern computer. However, the concept of computer has come to mean a special type of programing machine having some basic characteristics.

- Fast and reliable
- Receive, process and store information
- A multitasking machine perform quickly, accurately

Following are some definitions of computer which will clear your concept

- An electronic machine can store and process information. It is defined as stored program digital computing system.
- A Computer is an electronic device of wires, transistors, circuits, instructions and data can transmit, store and manipulate information.
- A Computer is a machine that accepts data and processes that data (data may be numbers, letters or both or even sounds). Information is turned into electrical pulses so that it may be processed by sorting, collating and deleting mathematical manipulation and other forms of data processing.

A computer accepts information using an input device. The information is processed by a central processing unit or stored in a storage unit i.e. Hard Disk and then processed. The result is supplied by an output devices. If the input and output units are connected to a manufacturing process, the computer can control the process and the system is called a computing system.

To understand computer basic concept one must know about the 4 functions of computer.

Input. Computer receiver information from users. A user enters information using input devices like keyboard, mouse or any other devices like webcam, punch card, magnetic tapes, joystick or magnetic disk etc. The Input unit accept information using input devices then it convert the given data to readable form and this data moves to Central Processing Unit (CPU).

Storage. The given information stored in a computer using different storage devices i.e. central process unit and auxiliary memory. The auxiliary memory also known as secondary or external storage have hard devices for example Floppy, Hard Disk, Compact Disk and Flash Drive. These different storage devices have both advantages and disadvantages. Auxiliary storage speed up information and store it long term and permanently.

Processing. It is considered the basic computing operation. It execute the instructions, control storage data and input or output devices attached the computer.

Output. Last but not the least the output unit which represents results from the operations of central processing unit CPU. The result may be in hard or soft form i.e. visual display unit, printers and headphones.

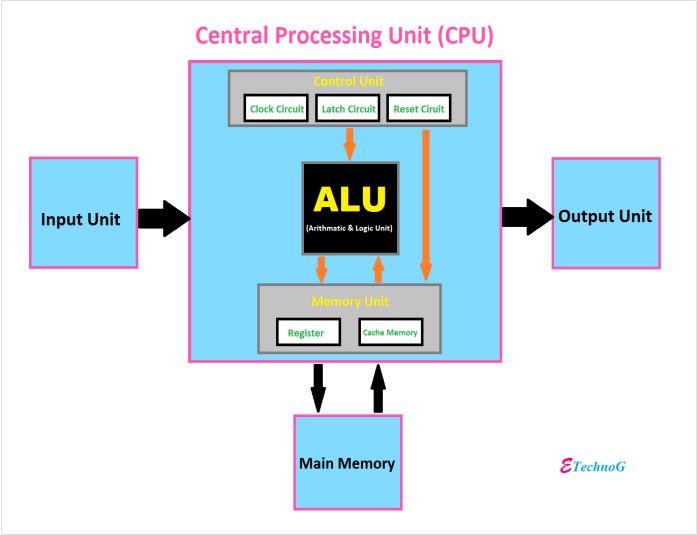


Figure 1.1

1.1 HISTORY OF COMPUTERS

Computers differ based on their **data processing** abilities. They are classified to purpose, data handling and functionality. Computers are either general purpose or specific purpose. General purpose computers are designed to perform a range of tasks.

They have the ability to store numerous programs, but lack in speed and efficiency. Specific purpose computers are designed to handle a specific problem or to perform a specific task. A set of instructions is built into the machine.

According to data handling, computers are analog, digital or hybrid. **Analog computers** work on the principle of measuring, in which the measurements obtained are translated into data.

Digital computers are those that operate with information, numerical or otherwise, represented in a digital form. Such computers process data into a digital value (in 0s and 1s). They give the results with more accuracy and at a faster rate. Hybrid computers incorporate the measuring feature of an analog computer and counting feature of a

digital computer. For computational purposes, these computers use analog components and for storage, digital memories are used.

Keyword

Analog Computer is one that can perform multiple calculations at once and can cope with infinite fractions of numbers.

According to functionality, computers are classified as:

- *Analog Computer*: An analog computer is a form of computer that uses continuous physical phenomena such as electrical, mechanical, or hydraulic quantities to model the problem being solved
- *Digital Computer*: A computer that performs calculations and logical operations with quantities represented as digits, usually in the binary number system.
- *Hybrid Computer (Analog + Digital)*: A combination of computers those are capable of inputting and outputting in both digital and analog signals. A hybrid computer system setup offers a cost effective method of performing complex simulations.

On the Basis of Size

Super Computer

The fastest and most powerful type of computer Supercomputers are very expensive and are employed for specialized applications that require immense amounts of mathematical calculations. *For example*, weather forecasting requires a supercomputer. Other uses of supercomputers include animated graphics, fluid dynamic calculations, nuclear energy research, and petroleum exploration.

The chief difference between a supercomputer and a mainframe is that a supercomputer channels all its power into executing a few programs as fast as possible, whereas a mainframe uses its power to execute many programs concurrently.

Mainframe Computer

A very large and expensive computer capable of supporting hundred, or even thousands, of users simultaneously. In the hierarchy that starts with a simple **microprocessor** at the bottom and moves to supercomputers at the top, mainframes are just below supercomputers. In some ways, mainframes are more

REMEMBER

Modern analog computers usually employ electrical parameters, such as voltages, resistances or currents, to represent the quantities being manipulated. Such computers do not deal directly with the numbers. They measure continuous physical magnitudes.

powerful than supercomputers because they support more simultaneous programs. But supercomputers can execute a single program faster than a mainframe.

Mini Computer

Minicomputer is a midsized computer. In size and power, minicomputers lie between workstations and mainframes. In the past decade, the distinction between large minicomputers and small mainframes has blurred, however, as has the distinction between small minicomputers and workstations. But in general, a minicomputer is a multiprocessing system capable of supporting from 4 to about 200 users simultaneously.

Micro Computer or Personal Computer

- Desktop Computer: a personal or micro-mini computer sufficient to fit on a desk.
- Laptop Computer: a portable computer complete with an integrated screen and keyboard. It is generally smaller in size than a desktop computer and larger than a notebook computer.
- Palmtop Computer/Digital Diary /Notebook /PDAs: a hand-sized computer. Palmtops have no keyboard but the screen serves both as an input and output device.

Palmtop PC

A palmtop is a small computer that can be carried along in the pocket or the briefcase. The device goes also by the acronym PDA, short for Personal Digital Assistant; another common name for it is pocket computer, Pocket PC. Palmtops take up less space than laptops, although they are not meant to replace regular computers entirely.

Palmtops and users have found each other, and palmtop sales have grown wildly over the recent years. The sales are estimated to continue increasing and exceed the demand of desktops in the future.

Because of their small size, most palmtop computers do not include disk drives. However, many contain PCMCIA slots in

which one can insert disk drives, modems, memory, and other devices.

Keyword

Microprocessor is simply a computer processor that has been configured into the design and function of a microchip.

Laptop PC

Portable and compact personal computers with the same capabilities as a desktop computer. Laptop computers have an L-shape design and the screen can be lowered and closed to allow for easy transportation of the machine. The primary feature that attracts users to laptops over desktops is their portability. Laptop computers provide users the ability to run the machine using an internal battery or an outside power adaptor. Laptops have dramatically decreased in size since their introduction in 1979. Laptops are usually more expensive than standard desktops and do not have the same life-span as fixed personal computers.

Laptops are liked because of their portability. Manufactures has found it interesting that not just executives but also other people bought portable computer as if they are hot cakes. This fact has improved the designs and variety of the laptops. Laptop advancements has resulted in making available almost every sort of portable computers as are matching with needs of every person. This is an introduction the models of the portable computers i.e. laptops.

Personal Computer

Personal computer (PC), a digital computer designed for use by only one person at a time. A typical personal computer assemblage consists of a central processing unit (CPU), which contains the computer's arithmetic, logic, and control circuitry on an integrated circuit; two types of computer memory, main memory, such as digital random-access memory (RAM), and auxiliary memory, such as magnetic hard disks and special optical compact discs, or read-only memory (ROM) discs (CD-ROMs and DVD-ROMs); and various input/output devices, including a display screen, keyboard and mouse, modem, and printer.

Workstations

A high-performance single-user computer typically used for graphics CAD, software development and scientific applications.

A workstation may be a RISC-based computer that runs under some version of UNIX or Linux, the major vendors being Sun, HP, IBM and SGI. It may also refer to a high-end PC using Intel or AMD CPUs from any PC vendor. In all cases, the term implies a machine with a fast CPU and large amounts of memory and disk that is geared toward the professional user rather than the consumer.

A terminal or desktop computer in a network. In this context, workstation is just a generic term for a user's machine (client machine) in contrast to a "server" or "mainframe."

In the telecom industry, a combined telephone and computer.

For years, workstations like these from Sun, Compaq and SGI (top to bottom) were used for CAD, medical imaging and scientific visualization. Combined with high-resolution monitors, they were traditionally UNIX based and pushed the performance envelope. However, today's Windows PCs and Macs are much more powerful than all the earlier UNIX workstations.

REMEMBER

A palmtop computer is a personal computer or other electronic device that has many of the same features as a computer and fits in the palm of the hand. A good example of an early palmtop computer is the Palm Pilot.



DID YOU **KNOW** ?

In 1979, William Moggridge of Grid Systems Corporation created the first functioning portable computer: The Grid Compass Computer 1109.

1.2 NUMBER REPRESENTATION AND BINARY CODE

Binary is a number system which builds numbers from elements called bits. Each bit can be represented by any two mutually exclusive states. Generally, when one write it down or code bits, and represent them with 1 and 0 to build binary numbers the same way and build numbers in the traditional base 10 system. However, instead of a one's column, a 10's column, and a 100 column (and so on) and have a one's column, a two's columns, a four's column, an eight's column, and so on, as illustrated:

Table 1.1: Binary

$2^{...}$	2^6	2^5	2^4	2^3	2^2	2^1	2^0
...	64	32	16	8	4	2	1

For example, to represent the number 203 in base 10, place a 3 in the 1's column, a 0 in the 10's column and a 2 in the 100's column.

This is expressed with exponents in the Table 1.2.

Table 1.2: 203 in base 10

10^2	10^1	10^0
2	0	3

Or, in other words, 2 Ã 10^2 + 3 Ã 10^0 = 200 + 3 = 203.

To represent the same thing in binary, and would have in the Table 1.3.

Table 1.3: 203 in base 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	0	1	0	1	1

That equates to $2^7 + 2^6 + 2^3+2^1 + 2^0$ = 128 + 64 + 8 + 2 + 1 = 203.

Conversion

The easiest way to convert between bases is to use a computer; after all, that's what they are good at! However, it is often useful to know how to do conversions by hand.

The easiest method to convert between bases is repeated division. To convert, repeatedly divide the quotient by the base, until the quotient is zero, making note of the remainders at each step. Then, write the remainders in reverse, starting at the bottom and appending to the right each time. An *example* should illustrate; since you are converting to binary and use a base of 2.

Table: 1.4: Convert 203 to binary

Quotient	Â	Remainder	Â
203_{10} Ã· 2 =	101	1	Â
101_{10} Ã· 2 =	50	1	â†‘
50_{10} Ã· 2 =	25	0	â†‘
25_{10} Ã· 2 =	12	1	â†‘
12_{10} Ã· 2 =	6	0	â†‘
6_{10} Ã· 2 =	3	0	â†‘
3_{10} Ã· 2 =	1	1	â†‘
1_{10} Ã· 2 =	0	1	â†‘

Reading from the bottom and appending to the right each time gives 11001011, which anyone saw from the example was 203.ss

Bits and Bytes

To represent all the letters of the alphabet you would need at least enough different combinations to represent all the lower case letters, the upper case letters, numbers and punctuation, plus a few extras. Adding this up means need probably around 80 different combinations.

- If you have two bits and can represent 4 unique combinations (00 01 10 11).
- If you have three bits and can represent 8 different combinations.
- With n bits you can represent 2^n unique combinations.

8 bits gives us 2^8 = 256 unique representations, more than enough for the alphabet combinations. And call a group of 8 bits a *byte*. Guess how bit a C char variable is?

ASCII

Given that a byte can represent any of the values 0 through 256, anyone could arbitrarily make up a mapping between characters and numbers. *For example*, a video card manufacturer could decide that the value 10 represents A, so when value 10 is sent to the video card it displays a capital 'A' on the screen.

To avoid this happening, the *American Standard Code for Information Interchange* or ASCII was invented. This is a *7-bit* code, meaning there are 2^7 or 128 available codes.

The range of codes is divided up into two major parts; the non-printable and the printable. Printable characters are things like characters (supper and lower case), numbers and punctuation. Nonprintable codes are for control, and do things like make a carriage-return, ring the terminal bell or the special `NULL` code which represents nothing at all.

127 unique characters is sufficient for American English, but becomes very restrictive when someone wants to represent characters common in other languages, especially Asian languages which can have many thousands of unique characters.

To alleviate this, modern systems are moving away from ASCII to *Unicode*, which can use up to 4 bytes to represent a character, giving *much* more room!

Keyword

Parity Bit is a bit added to the end of a string of binary code that indicates whether the number of bits in the string with the value one is even or odd.

Parity

ASCII, being only a 7-bit code, leaves one bit of the byte spare. This can be used to implement *parity* which is a simple form of error checking. Consider a computer using punch-cards for input, where a hole represents 1 and no hole represents 0. Any inadvertent covering of a hole will cause an incorrect value to be read, causing undefined behavior. Parity allows a simple check of the bits of a byte to ensure they were read correctly, and can implement either *odd* or *even* parity by using the extra bit as a *parity bit*.

In odd parity, if the number of 1's in the 7 bits of information is odd, the parity bit is set, otherwise it is not set. Even parity is the opposite; if the number of 1's is even the **parity bit** is set to 1. In this way, the flipping of one bit will case a parity error, which can be detected.

16, 32 and 64 bit Computers

Numbers do not fit into bytes; hopefully bank balance in dollars will need more range than can fit into one byte! Most modern architectures are *32 bit* computers. This means they work with 4 bytes at a time when processing and reading or writing to memory; and refers to 4 bytes as a *word*; this is analogous to language where letters (bits) make up words in a sentence,

except in computing every word has the same size! The size of a C `it` variable is 32 bits. Newer architectures are 64 bits, which doubles the size the processor works with (8 bytes).

Kilo, Mega and Giga Bytes

Computers deal with a lot of bytes; that's what makes them so powerful!

Keyword

Boolean algebra is the subarea of algebra in which the values of the variables are the truth values true and false, usually denoted 1 and 0 respectively.

One need a way to talk about large numbers of bytes, and a natural way is to use the "International System of Units" (SI) prefixes as used in most other scientific areas. So *for example*, kilo refers to 10^3 or 1,000 units, as in a kilogram has 1,000 grams.

1,000 is a nice round number in base 10, but in binary it is `1111101000` which is not a particularly "round" number. However, 1024 (or 2^{10}) is (`10000000000`), and happens to be quite close to the base ten meaning of kilo (1000 as opposed to 1024).

Hence 1024 bytes became known as a *kilobyte*. The first mass market computer was the Commodore 64, so named because it had 64 kilobytes of storage.

Today, kilobytes of memory would be small for a wrist watch, let alone a personal computer. The next SI unit is "mega" for 10^6. As it happens, 2^{20} is again close to the SI base 10 definition; 1048576 as opposed to 1000000. The units keep increasing by powers of 10; each time it diverges from the base SI meaning.

Table 1.5: Bytes

2^{10}	Kilobyte
2^{20}	Megabyte
2^{30}	Gigabyte
2^{40}	Terabyte
2^{50}	Petabyte
2^{60}	Exabyte

Therefore a 32 bit computer can address up to four gigabytes of memory; the extra two bits can represent four groups of 2^{30} `bytes`. A 64 bit computer can address up to 8 Exabyte's; you might

be interested in working out just how big a number this is. To get a feel for how bit that number is calculate how long it would take to count to 2^{64}; if you incremented once per second.

Boolean Operations

George Boole was a mathematician who discovered a whole area of mathematics called ***Boolean algebra***. Whilst he made his discoveries in the mid 1800's, his mathematics is the fundamentals of all computer science.

Boolean operations simply take a particular input and produce a particular output follows a rule.

The simplest Boolean operation, not simply inverts the value of the input operand. Other operands usually take two inputs, and produce a single output.



The fundamental Boolean operations used in computer science are easy to remember and listed. These represent them with truth tables; they simply show all possible inputs and outputs. The term true simply reflects 1 in binary.

NOT

Usually represented by (`!`), *`NOT`* simply inverts the value, so `0` becomes `1` and `1` becomes `0`

Table 1.6: Truth table for not

Input	Output
1	0
0	1

AND

To remember how the AND operation works think of it as "if one input true and the other are true, result is true.

Table 1.7: Truth table for and

Input 1	Input 2	Output
0	0	0
1	0	0
0	1	0
1	1	1

OR

To remember how the OR operation works think of it as "if one input or the other input is true, the result is true.

Table 1.8: Truth table for OR

Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	1

Exclusive OR (XOR)

Exclusive OR, written as XOR is a special case of or where the output is true if one, and only one, of the inputs is true. This operation can surprisingly do many interesting tricks, but one will not see a lot of it in the kernel.

Table 1.9: Truth table for XOR

Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	0

How Computers use Boolean Operations

Believe it or not, essentially everything computer does comes back to the operations. *For example*, the half adder is a type of circuit made up from Boolean operations that can add bits together (it is called a half adder because it does not handle carry bits). Put more half adders together, and will start to build something that can add together long binary numbers. Add some external memory, and has a computer.

Electronically, the Boolean operations are implemented in *gates* made by *transistors*. This is why you might have heard about transistor counts and things like Moores Law. The more transistors, the more gates, the more things and can add together. To create the modern computer, there are an awful lot of gates, and an awful lot of transistors. Some of the latest Itanium processors have around 460 million transistors.

Hexadecimal

Hexadecimal refers to a base 16 number system. Uses this in computer science for only one reason; it makes it easy for humans to think about binary numbers. Computers only ever deal in binary and hexadecimal is simply a shortcut for us humans trying to work with the computer.

So why base 16? Well, the most natural choice is base 10, since anyone is used to thinking in base 10 from everyday number system. But base 10 does not work well with binary to represent 10 different elements in binary, and need four bits. Four bits, however, gives us sixteen possible combinations. So they can either take the very tricky road of trying to convert between base 10 and binary, or take the easy road and make up a base 16 number system - hexadecimal!

Hexadecimal uses the standard base 10 numerals, but adds `A B C D E F` which refer to `10 11 12 13 14 15` (one start from zero).

Traditionally, any time you sees a number prefixed by `0x` this will denote a hexadecimal number.

As mentioned, to represent 16 different patterns in binary, and would need exactly four bits. Therefore, each hexadecimal numeral represents exactly four bits.

One should consider it an exercise to learn the given Table off by heart.

Table 1.10: Hexadecimal, binary and decimal

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Of course there is no reason not to continue the pattern (say, assign G to the value 16), but 16 values is an excellent tradeoff between the vagaries of human memory and the number of bits used by a computer. Anyone simply represents larger numbers of bits with more numerals. *For example*, a sixteen bit variable can be represented by 0xAB12, and to find it in binary simply take each individual numeral, convert it as per the table and join them all together (so 0xAB12 ends up as the 16-bit binary number 1010101100010010). And can use the reverse to convert from binary back to hexadecimal.

One can also use the same repeated division scheme to change the base of a number.

For example, to find 203 in hexadecimal.

Table 1.11: Convert 203 to hexadecimal

Quotient	Â	Remainder	Â
203_{10} Ã· 16 =	12	11 (0xB)	Â
12_{10} Ã· 16 =	0	12 (0xC)	â†‘

Hence 203 in hexadecimal is 0xCB.

Practical Implications

Use of Binary in Code

Whilst binary is the underlying language of every computer, it is entirely practical to program a computer in high level languages without knowing the first thing about it. However, for the low level codes and they are interested in a few fundamental binary principles are used repeatedly.

1.2.1 Fixed and Floating Point Number Representation

The Institute of Electrical and Electronics Engineers (IEEE) standardizes floating-point representation in IEEE 754. Floating-point representation is similar to scientific notation in that there is a number multiplied by a base number raised to some power. *For example*, 118.625 are represented in scientific notation as 1.18625×10^2. The main benefit of this representation is that it provides varying degrees of precision based on the scale of the numbers that anyone is using. *For example*, it is beneficial to talk in terms of angstroms (10^{-10} m) when one is working with the distance between atoms. However, if they are dealing with the distance between cities, this level of precision is no longer practical or necessary.

IEEE 754 defines binary representations for 32-bit single-precision and 64-bit double-precision (64-bit) numbers as well as extended single-precision and extended double-precision numbers. Examine the specification for single-precision, floating-point numbers, also called floats.

A float consists of three parts: the sign bit, the exponent, and the mantissa. The division of the three parts is as follows:



The sign bit is 0 if the number is positive and 1 if the number is negative.

The exponent is an 8-bit number that ranges in value from -126 to 127. The exponent is actually not the typical two's complement representation because this makes comparisons more difficult. Instead, the value is biased by adding 127 to the desired exponent and representation, which makes it possible to represent negative numbers.

The mantissa is the normalized binary representation of the number to be multiplied by 2 raised to the power defined by the exponent.

Now look at how to encode 118.625 as a float. The number 118.625 is a positive number, so the sign bit is 0. To find the exponent and mantissa, first write the number in binary, which are 1110110.101. Next, normalize the number to 1.110110101 × 26, which is the binary equivalent of scientific notation. The exponent is 6 and the mantissa is 1.110110101. The exponent must be biased, which is 6 + 127 = 133. The binary representation of 133 is 10000101.

Thus, the floating-point encoded value of 118.65 is 0100 0010 1111 0110 1010 0000 0000 0000. Binary values are often referred to in their hexadecimal equivalent. In this case, the hexadecimal value is 42F6A000.

Fixed Point Representation

In fixed-point representation, a specific radix point called a decimal point in English and written "." is chosen so there is a fixed number of bits to the right and a fixed number of bits to the left of the radix point. The bits to the left of the radix point are called the integer bits. The bits to the right of the radix point are called the fractional bits.



In this example, assume a 16-bit fractional number with 8 magnitude bits and 8 radix bits, which is typically represented as 8.8 representations. Like most signed integers, fixed-point numbers are represented in two's complement binary. Using a positive number keeps this example simple.

To encode 118.625, first find the value of the integer bits. The binary representation of 118 is 01110110, so this is the upper 8 bits of the 16-bit number. The fractional part of the number is represented as 0.625×2^n where n is the number of fractional bits. Because 0.625 × 256 = 160, and use the binary representation of 160, which is 10100000, to determine the fractional bits. Thus, the binary representation for 118.625 is 0111 0110 1010 0000. The value is typically referred to using the hexadecimal equivalent, which is 76A0.

The major advantage of using fixed-point representation for real numbers is that fixed-point adheres to the same basic arithmetic principles as integers. Therefore, fixed-point numbers can take advantage of the general optimizations made to the Arithmetic Logic Unit (ALU) of most microprocessors, and do not require any additional libraries or any additional hardware logic. On processors without a floating-point unit (FPU), such as the Analog Devices Black fin Processor, fixed-point representation can result in much more efficient embedded code when performing mathematically heavy operations.

In general, the disadvantage of using fixed-point numbers is that fixed-point numbers can represent only a limited range of values, so fixed-point numbers are susceptible to common numeric computational inaccuracies. *For example*, the range of possible values in the 8.8 notation that can be represented is +127.99609375 to -128.0. If you add 100 + 100 and exceed the valid range of the data type which is called overflow. In most cases, the values that overflow are saturated, or truncated, so that the result is the largest representable number.

1.2.2 BCD

In computing and electronic systems, binary-coded decimal (BCD) or, in its most common modern implementation, packed decimal, is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Its main virtue is that it allows easy conversion to decimal digits for printing or display, and allows faster decimal calculations. Its drawbacks are a small increase in the complexity of circuits needed to implement mathematical operations. Uncompressed BCD is also a relatively inefficient encoding it occupies more space than a purely binary representation.

Short for Binary Coded Decimal, BCD is also known as packet decimal and is numbers 0 through 9 converted to four-digit binary. There is a list of the decimal numbers 0 through 9 and the binary conversion.

In BCD, a digit is usually represented by four bits which, in general, represent the decimal digits 0 through 9. Other bit combinations are sometimes used for a sign or for other indications (e.g., error or overflow).

Although uncompressed BCD is not as widely used as it once was, decimal fixed-point and floating-point are still important and continue to be used in financial, commercial, and industrial computing.

Recent decimal floating-point representations use base-10 exponents, but not BCD encodings. Current hardware implementations, however, convert the compressed decimal encodings to BCD internally before carrying out computations. Software implementations of decimal arithmetic typically use BCD or some other 10^n base, depending on the operation.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100

5	0101
6	0110
7	0111
8	1000
9	1001

Using this conversion, the number 25, *for example*, would have a BCD number of 0010 0101 or 00100101. However, in binary, 25 are represented as 11001.

Number larger than 9 having two or more digits in the decimal system, are expressed digit by digit. *For example*, the BCD rendition of the base-10 number 1895 is

0001 1000 1001 0101

The binary equivalents of 1, 8, 9, and 5, always in a four-digit format, go from left to right.

The BCD representation of a number is not the same, in general, as its simple binary representation. In binary form, *for example*, the decimal quantity 1895 appears as

11101100111

Other bit patterns are sometimes used in BCD format to represent special characters relevant to a particular system, such as sign (positive or negative), error condition, or overflow condition.

The BCD system offers relative ease of conversion between machine-readable and human-readable numerals. As compared to the simple binary system, however, BCD increases the circuit complexity. The BCD system is not as widely used today as it was a few decades ago, although some systems still employ BCD in financial applications.

1.2.3 EBCDIC

Extended Binary Coded Decimal Interchange Code (EBCDIC) is a character encoding set used by IBM mainframes. Unlike virtually every computer system in the world which uses a variant of ASCII, IBM mainframes and midrange systems such as the AS/400 tend to use a wholly incompatible character set primarily designed for ease of use on punched cards. (For an excellent page on punched cards, see Doug Jones's Punched Card Codes). EBCDIC uses the full 8 bits available to it, so parity checking cannot be used on an 8 bit system. Also, EBCDIC has a wider range of control characters than ASCII. The character encoding is based on Binary Coded Decimal (BCD), so the contiguous characters in the alphanumeric range are formed up in blocks of up to 10 from 0000 binary to 1001 binary. Non alphanumeric characters are almost all outside the BCD range. There are four main blocks in the EBCDIC code page: 0000 0000 to 0011

1111 is reserved for control characters; 0100 0000 to 0111 1111 is for punctuation; 1000 0000 to 1011 1111 for lowercase characters and 1100 0000 to 1111 1111 for uppercase characters and numbers.

1.2.4 ASCII

ASCII is the American Standard Code for Information Interchange, also known as ANSI X3.4. There are many variants of this standard, typically to allow different code pages for language encoding, but they all basically follow the same format. ASCII is quite elegant in the way it represents characters, and it is very easy to write code to manipulate upper/lowercase and check for valid data ranges.

ASCII is essentially a 7-bit code which allows the 8th most significant bit (MSB) to be used for error checking, however most modern computer systems tend to use ASCII values of 128 for extended character sets.

Table 1.12: Table of ASCII characters

binary	MSN	0000		0001		0010		0011		0100		0101		0110		0111	
LSN	hex	*0*		*1*		*2*		*3*		*4*		*5*		*6*		*7*	
0000	*0*	**NUL**	000	**DLE**	16 10	**SP**	32 20	**0**	48 30	**@**	64 40	**P**	80 50	**`**	96 60	**p**	112 70
0001	*1*	**SOH**	1 01	**XON(DC1)**	17 11	**!**	33 21	**1**	49 31	**A**	65 41	**Q**	81 51	**a**	97 61	**q**	113 71
0010	*2*	**STX**	2 02	**DC2**	18 12	**“**	34 22	**2**	50 32	**B**	66 42	**R**	82 52	**b**	98 62	**r**	114 72
0011	*3*	**ETX**	3 03	**XOFF (DC2)**	19 13	**#**	35 23	**3**	51 33	**C**	67 43	**S**	83 53	**c**	99 63	**s**	115 73
0100	*4*	**EOT**	4 04	**DC4**	20 14	**$**	36 24	**4**	52 34	**D**	68 44	**T**	84 54	**d**	100 64	**t**	116 74
0101	*5*	**ENQ**	5 05	**NAK**	21 15	**%**	37 25	**5**	53 35	**E**	69 45	**U**	85 55	**e**	101 65	**u**	117 75
0110	*6*	**ACK**	6 06	**SYN**	22 16	**&**	38 26	**6**	54 36	**F**	70 46	**V**	86 56	**f**	102 66	**v**	118 76
0111	*7*	**BEL**	7 07	**ETB**	23 17	**‘**	39 27	**7**	55 37	**G**	71 47	**W**	87 57	**g**	103 67	**w**	119 77
1000	*8*	**BS**	8 08	**CAN**	24 18	**(**	40 28	**8**	56 38	**H**	72 48	**X**	88 58	**h**	104 68	**x**	120 78

1001	*9*	**HT**	9 09	**EM**	25 19	)	41 29	**9**	57 39	**I**	73 49	**Y**	89 59	**i**	105 69	**y**	121 79
1010	*A*	**LF**	10 0A	**SUB**	26 1A	*	42 2A	:	58 3A	**J**	74 4A	**Z**	90 5A	**j**	106 6A	**z**	122 7A
1011	*B*	**VT**	11 0B	**ESC**	27 1B	+	43 2B	;	59 3B	**K**	75 4B	[	91 5B	**k**	107 6B	{	123 7B
1100	*C*	**FF**	12 0C	**FS**	28 1C	,	44 2C	<	60 3C	**L**	76 4C	\	92 5C	**l**	108 6C	|	124 7C
1101	*D*	**CR**	13 0D	**GS**	29 1D	-	45 2D	=	61 3D	**M**	77 4D	]	93 5D	**m**	109 6D	}	125 7D
1110	*E*	**SO**	14 0E	**RS**	30 1E	.	46 2E	>	62 3E	**N**	78 4E	^	94 5E	**n**	110 6E	~	126 7E
1111	*F*	**SI**	150F	**US**	311F	/	472F	?	633F	**O**	794F	_	955F	**o**	1116F	**DEL**	1277F

Excess-3

Excess-3 code is an example of un-weighted code. Excess-3 equivalent of a decimal number is obtained by adding 3 and then converting it to a binary format. For instance to find excess-3 representation of decimal number 4, first 3 is added to 4 to get 7 and then binary equivalent of 7 i.e. 0111 forms the excess-3 equivalent. The truth table of excess-3 is given in Table 1.13.

Table 1.13: Truth table of excess-3

Truth Table							
Input (BCD)				Output (Excess-3)			
A	**B**	**C**	**D**	**W**	**X**	**Y**	**Z**
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Here is a Table representing excess-3 equivalent of decimal numbers (0-9):

Table 1.14: Excess-3 equivalent of decimal numbers

Decimal Number	Excess-3 Equivalent
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Excess 3 Code Additions

The operation of addition can be done by very simple method you will illustrate the operation in a simple way using steps.

- *Step 1:* You have to convert the numbers (which are to be added) into excess 3 forms by adding 0011 with each of the four bit groups them or simply increasing them by 3.
- *Step 2:* Now the two numbers are added using the basic laws of binary addition, there is no exception for this method.
- *Step 3:* Now which of the four groups have produced a carry one has to add 0011 with them and subtract 0011 from the groups which have not produced a carry during the addition.
- *Step 4:* The result which you have obtained after this operation is in Excess 3 form and this is desired result

Example

To understand the Excess 3 Code Addition method better you can observe the method with the help of an example,

Let us take two numbers which one will to add.

0011 0101 0110 and 0101 0111 1001 are the two binary numbers. Now follows the first step one take the excess 3 form of these two numbers which are 0110 1000 1001 and 1000 1010 1100, now these numbers are added the basic rules of addition.

0110 1000 1001

1000 1010 1100

1111 0011 0101

Now adding 0011 to the groups which produces a carry and subtracting zero from the groups which did not produced carry one get the result as 1100 0110 1000 is the result of the addition in excess 3 code and the BCD answer is 1001 0011 0101.

Excess 3 Code Subtractions

Similarly binary subtraction can be performed by Excess 3 Code Subtraction method. The operation is illustrated with the help of some steps.

- *Step 1:* The numbers have to be converted into excess 3 codes.
- *Step 2:* Following the basic methods of binary subtraction, subtraction is done.
- *Step 3:* Subtract '0011' from each BCD four-bit group in the answer if the subtraction operation of the relevant four-bit groups required borrow from the next higher adjacent four-bit group.
- *Step 4:* Add '0011' to the remaining four-bit groups, if any, in the result.
- *Step 5:* Finally one gets the desired result in excess 3 codes.

Example

Again an example will make the understanding very easy for us.

Let us take the numbers

0001 1000 0101 and 0000 0000 1000 now the excess 3 equivalent of those numbers are 0100 1011 1000 and 0011 0011 1011

Now performing the operation of binary subtraction one get

0100 1011 1000

0011 0011 1011

0001 0111 1101

The least significant column which needed borrow and the other two columns did not need borrow. Now you have to subtract 0011 from the result of this column

and add 0011 to the other two columns, and get 0100 1010 1010. This is the result expressed in excess 3 codes. And the binary result is 0001 0111 0111

1.2.5 IEEE Standard

Abbreviation of Institute of Electrical and Electronics Engineers, pronounced I-triple-E. Founded in 1884 as the AIEE, the IEEE was formed in 1963 when AIEE merged with IRE. IEEE is an organization composed of engineers, scientists, and students. The IEEE is best known for developing standards for the computer and electronics industry. In particular, the IEEE 802 standards for local-area networks are widely followed.

The IEEE has done notable work in the standards area of networking. This organization is huge with over 300,000 members made up of engineers, technicians, scientists, and students in related areas. The Computer Society of IEEE alone has over 100,000 members. IEEE is credited with having provided definitive standards in local area networking.

The 802 standards were the culmination of work performed by the subcommittee starting in 1980. This was followed in 1985 with specific LAN-oriented standards titled 802.2 - 802.5. Since that time there have been other references set up as well. Most of the work performed by the 802 Project committee revolves around the first two layers of the OSI model initiated by the ISO. These layers involve the physical medium on which you move data (cable type) and the way that interacts with it. It addresses such crucial issues of how data is placed on the network and how insures its accuracy and flow. In order to better define these functions, the IEEE split the Data Link layer of the OSI model up into two separate components.

802 IEEE committee responsible for setting standards concerning cabling physical topologies logical topologies and physical access methods for networking products. The Computer Society of IEEE's 802 Project Committee is divided into several subcommittees that deal with specific standards in these general areas. Specifically the Physical layer and the Data Link layer of the ISO's OSI model are addressed.

802.1 This work defines an overall picture of LANs and connectivity. 802.1B this set of standards specifically addressed network management.

802.1D Standards for bridges used to connect various types of LANs together were set up with 802.1D.

802.2 called the logical link control (LLC) standards; this specification governs the communication of packets of information from one device to another on a network. Specifically it deals with communication, not access to the network itself.

802.3 Defines the way data has access to a network for multiple topology systems using Carrier Sense Multiple Access/ Collision Detection (CSMA/CD). A prime example

is Ethernet and Star LAN systems. These LAN types operate at 10 Mb/sec.

802.4 Standards developed for a token-passing scheme on a bus topology. The primary utilizer of this specification was the Manufacturing Automation Protocol LANs developed by General Motors. Operate at 10 Mb/sec.

802.5 This standard defines token ring systems. It involves the token-passing concept on a ring topology with twisted pair cabling. IBM's token ring system uses this specification. The speed is either 4 Mb/sec or 16 Mb/sec.

802.6 Metropolitan Area Networks are defined by this group. MANs are networks that are larger than LANs typically falling within 50 kilometers. They operate at speeds ranging from 1 Mb/sec up to about 200 Mb/sec.

REMEMBER

It must be instructed to perform a specific task and therefore, to solve a problem the user has to specify the sequence of instructions which a computer must follow.

802.7 These are standards concerning broadband LANs.

802.8 This group sets up standards for LANs using fiber optic cabling and access methods.

802.9 This specification covers voice and digital data integration.

802.10 These members set standards for interoperable security.

802.11 Wireless LANs are the subject of this particular subcommittee's works. Both infrared and radio LANs are covered.

IEEE standard for Floating Point Representation

The IEEE (Institute of Electrical and Electronics Engineers) has produced a standard for floating point arithmetic. This standard specifies how single precision (32 bit) and double precision (64 bit) floating point numbers are to be represented, as well as how arithmetic should be carried out on them.

Single Precision

The IEEE single precision floating point standard representation requires a 32 bit word, which may be represented as numbered from 0 to 31, left to right. The first bit is the sign bit, S, the next eight bits are the exponent bits, 'E', and the final 23 bits

are the fraction 'F':

S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF

0 1 8 9 31

The value V represented by the word may be determined as follows:

- If E=255 and F is nonzero, then V=NaN ("Not a number")
- If E=255 and F is zero and S is 1, then V=-Infinity
- If E=255 and F is zero and S is 0, then V=Infinity
- If 0<E<255 then V= (-1)**S * 2 ** (E-127) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E=0 and F is nonzero, then V= (-1)**S * 2 ** (-126) * (0.F) These are "normalized" values.
- If E=0 and F is zero and S is 1, then V=-0
- If E=0 and F is zero and S is 0, then V=0

In particular,

0 00000000 00000000000000000000000 = 0

1 00000000 00000000000000000000000 = -0

0 11111111 00000000000000000000000 = Infinity

1 11111111 00000000000000000000000 = -Infinity

0 11111111 00000100000000000000000 = NaN

1 11111111 00100010001001010101010 = NaN

0 10000000 00000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2

0 10000001 10100000000000000000000 = +1 * 2**(129-127) * 1.101 = 6.5

1 10000001 10100000000000000000000 = -1 * 2**(129-127) * 1.101 = -6.5

0 00000001 00000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)

0 00000000 10000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)

0 00000000 00000000000000000000001 = +1 * 2**(-126) *

0.00000000000000000000001 =

2**(-149) (Smallest positive value)

Double Precision

The IEEE double precision floating point standard representation requires a 64 bit word, which may be represented as numbered from 0 to 63, left to right. The first bit is the sign bit, S, the next eleven bits are the exponent bits, 'E', and the final 52 bits are the fraction 'F':

S EEEEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

0 1 11 12 63

The value V represented by the word may be determined as follows:

- If E=2047 and F is nonzero, then V=NaN ("Not a number")
- If E=2047 and F is zero and S is 1, then V=-Infinity
- If E=2047 and F is zero and S is 0, then V=Infinity
- If 0<E<2047 then V=(-1)**S * 2 ** (E-1023) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E=0 and F is nonzero, then V=(-1)**S * 2 ** (-1022) * (0.F) These are "unnormalized" values.
- If E=0 and F is zero and S is 1, then V=-0
- If E=0 and F is zero and S is 0, then V=0

1.3 DIGITAL COMPUTER GENERATION

Generation in computer terminology is a change in technology a computer is/was being used. Initially, the generation term was used to distinguish between varying hardware technologies. But nowadays, generation includes both hardware and software, which together make up an entire computer system. There are totally five computer generations known till date. Each generation has been discussed in detail along with their time period and characteristics. Here approximate dates against each generations have been mentioned which are normally accepted.

First Generation (Vacuum Tubes)

The period of first generation was 1946-1959.

First generation of computers started with using vacuum tubes as the basic components for memory and circuitry for CPU (Central Processing Unit). These tubes like electric bulbs produced a lot of heat and were prone to frequent fusing of the installations, therefore, were very expensive and could be afforded only by very large organizations.

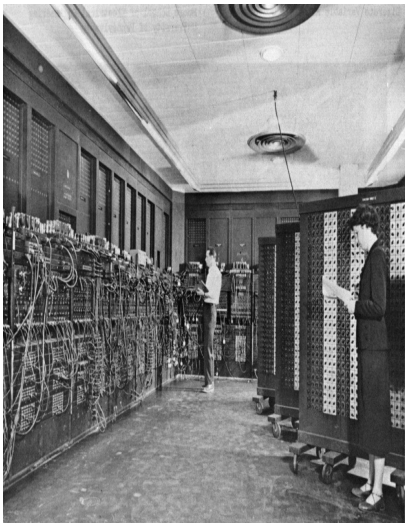
In this generation, mainly batch processing operating systems were used. In this generation, Punched cards, Paper tape, Magnetic tape Input and Output device were used. There were machine codes and electric wired board languages used.

The main features of First Generation are:

- Vacuum tube technology
- Unreliable
- Supported machine language only
- Very costly
- Generate lot of heat
- Slow Input/output device
- Huge size
- Need of A.C.
- Non-portable
- Consumed lot of electricity

Some computers of this generation were:

- ENIAC
- EDVAC
- UNIVAC
- IBM-701
- IBM-650

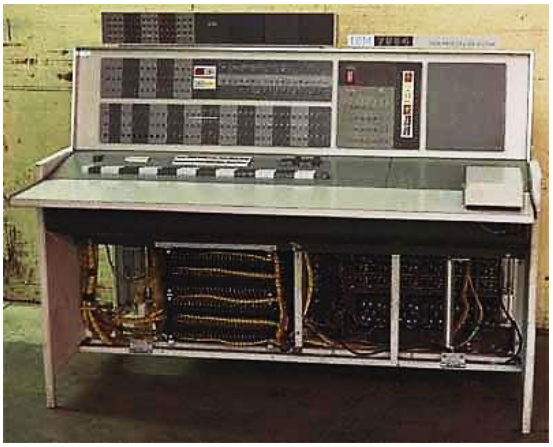


Second Generation (Transistors)

The period of second generation was 1959-1965. This generation using the transistor was cheaper, consumed less power, more compact in size, more reliable and faster than the first generation machines made of vacuum tubes. In this generation, magnetic cores were used as primary memory and magnetic tape and magnetic disks as secondary storage devices.

In this generation, assembly language and high-level programming language like FORTRAN, COBOL were used.

There were Batch processing and Multiprogramming Operating system used.



The main features of Second Generation are:

- Use of transistors
- Reliable as compared to First generation computers
- Smaller size as compared to First generation computers
- Generate less heat as compared to First generation computers
- Consumed less electricity as compared to First generation computers
- Faster than first generation computers
- Still very costly
- A.C. needed
- Support machine and assembly languages

Some computers of this generation were:

- IBM 1620
- IBM 7094

- CDC 1604
- CDC 3600
- UNIVAC 1108

Third Generation (Integrated Circuits)

The period of third generation was 1965-1971.

The third generation of computer is marked by the use of Integrated Circuits (IC's) in place of transistors. A single IC has many transistors, resistors and capacitors along with the associated circuitry. The IC was invented by Jack Kilby. This development made computers smaller in size, reliable and efficient.

In this generation, Remote processing, Time-sharing, Real-time, Multi-programming Operating System were used.

High-level language (FORTRAN-II TO IV, COBOL, PASCAL PL/1, BASIC, ALGOL-68, etc.) were used during this generation.



The main features of Third Generation are:

- IC used
- More reliable
- Smaller size
- Generate less heat
- Faster
- Lesser maintenance
- Still costly

- A.C. needed
- Consumed lesser electricity
- Support high-level language

Some computers of this generation were:

- IBM-360 series
- Honeywell-6000 series
- PDP(Personal Data Processor)
- IBM-370/168
- TDC-316

Fourth Generation (Microprocessors)

The period of fourth generation was 1971-1980. The fourth generation of computers is marked by the use of very large scale integrated (VLSI) circuits. VLSI circuits having about 5000 transistors and other circuit elements and their associated circuits on a single chip made it possible to have microcomputers of fourth generation. Fourth Generation computers became more powerful, compact, reliable, and affordable. As a result, it gave rise to personal computer (PC) revolution.

In this generation, Time sharing, Real time, Networks, Distributed Operating System were used.

All the higher level languages like C and C++, DBASE, etc., were used in this generation.



The main features of Fourth Generation are:

- VLSI technology used
- Very cheap

- Portable and reliable
- Use of PC's
- Very small size
- Pipeline processing
- No A.C. needed
- Concept of internet was introduced
- Great developments in the fields of networks
- Computers became easily available

Some computers of this generation were:

- DEC 10
- STAR 1000
- PDP 11
- CRAY-1 (Super Computer)
- CRAY-X-MP (Super Computer)

Fifth Generation (Artificial Intelligence)

The period of Fifth Generation is 1980-till date.

In the fifth generation, the VLSI technology became ULSI (Ultra Large Scale Integration) technology, resulting in the production of microprocessor chips having ten million electronic components.

This generation is based on parallel processing hardware and AI (Artificial Intelligence) software.

AI is an emerging branch in computer science which interprets means and methods of making computers think like human beings.

All the higher level languages like C and C++, Java, .Net, etc., are used in this generation.

AI includes:

- Robotics
- Neural networks
- Game Playing
- Development of expert systems to make decisions in real life situations.
- Natural language understanding and generation.



The main features of fifth generation are:

- ULSI technology
- Development of true artificial intelligence
- Development of Natural language processing
- Advancement in Parallel Processing
- Advancement in Superconductor technology
- More user friendly interfaces with multimedia features
- Availability of very powerful and compact computers at cheaper rates

Some computers types of this generation are:

- Desktop
- Laptop
- Notebook
- UltraBook
- ChromeBook

1.3.1 Computer Types and Classifications

The types of computers range from the Hybrid to the Analog types. The computers one comes across in the daily course of day range from laptops, palmtops and towers, to desktop computers, to name a few. But the very word "computers" reminds one of the desktop computers used in offices or homes. Different categories of computes have been devised in keeping with varied needs.

- Super Computers
- Main Frame Computers
- Mini Computers
- Micro Computers

Super Computers: E.g.:- CRAY Research:- CRAY-1 and CRAY-2, Fujitsu (VP2000), Hitachi (S820), NEC (SX20), PARAM 10000 by C-DAC, Anupam by BARC, PACE Series by DRDO

- Most powerful Computer system - needs a large room
- Minimum world length is 64 bits
- CPU speed: 100 MIPS
- Equivalent to 4000 computers
- High cost: 4 – 5 millions
- Able to handle large amount of data
- High power consumption
- High precision
- Large and fast memory (Primary and Secondary)
- Uses multiprocessing and parallel processing
- Supports multiprogramming

Applications

- In petroleum industry - to analyze volumes of seismic data which are gathered during oil seeking explorations to identify areas where there is possibility of getting petroleum products inside the earth
- In Aerospace industry - to simulate airflow around an aircraft at different speeds and altitude. This helps in producing an effective aerodynamic design for superior performance
- In Automobile industry – to do crash simulation of the design of an automobile before it is released for manufacturing – for better automobile design
- In structural mechanics – to solve complex structural engineering problems to ensure safety, reliability and cost effectiveness. *Example* Designer of a large bridge has to ensure that the bridge must be proper in various atmospheric conditions and pressures from wind, velocity etc. and under load conditions.
- Meteorological centers use super computers for weather forecasting *f* In Biomedical research – atomic nuclear and plasma analysis – to study the structure of viruses such as that causing AIDS
- For weapons research and development, sending rockets to space etc.

Main Frame Computers: E.g.:- IBM 3000 series, Burroughs B7900, Univac 1180, DEC

- Able to process large amount of data at very high speed
- Supports multi-user facility

- Number of processors varies from one to six.
- Cost: 3500 to many million dollars
- Kept in air conditioned room to keep them cool
- Supports many I/O and auxiliary storage devices
- Supports network of terminals

Applications

- Used to process large amount of data at very high speed of banks/insurance/ companies/hospitals/railways which need online processing of large number of transactions and requires massive data storage and processing capabilities
- Used as controlling nodes in WANs (Wide Area Networks).
- Used to manage large centralized databases.

Mini Computers: E.g.:- Digital Equipment PDP 11/45 and VAX 11)

- Perform better than micros
- Large in size and costlier than micros
- Designed to support more than one user at a time
- Possess large storage capacities and operates at higher speed
- Support faster peripheral devices like high speed printers
- Can also communicate with main frames

Applications

- These computers are used when the volume of processing is large for e.g. Data processing for a medium sized organization
- Used to control and monitor production processes
- To analyze results of experiments in laboratories
- Used as servers in LANs (Local Area Networks)

Micro Computers: E.g.:- IBM PC, PS/2 and Apple Macintosh

- A microcomputer uses a microprocessor as its central Processing Unit. Microcomputers are tiny computers that can vary in size from a single chip to the size of a desktop model
- They are designed to be used by only one person at a time
- Small to medium data storage capacities 500MB – 2GB

- The common examples of microcomputers are chips used in washing machines, TVs, Cars and Note book/Personal computers.

Applications

Used in the field of desktop publishing, accounting, statistical analysis, graphic designing, investment analysis, project management, teaching, entertainment etc.

The different models of microcomputers are given as:-

- *Personal Computers*: The name PC was given by the IBM for its microcomputers. PCs are used for word processing, spreadsheet calculations, database management etc.
- *Note Book or Lap Top*: Very small in terms of size can be folded and carried around. Monitor is made up of LCD and the keyboard and system units are contained in a single box. Got all the facilities of a personal computer (HDD, CDD, Sound card, N/W card, Modem etc.) and a special connection to connect to the desktop PC which can be used to transfer data.
- *Palm Top*: Smaller model of the microcomputer size is similar to that of a calculator pocket size. It has a processor and memory and a special connection to connect to the desktop PC which can be used to transfer data.
- *Wrist PC*: Smallest type of microcomputer can be worn on the wrist like a watch. It has a processor and memory and a wireless modem

1.3.2 Functional Units and Their Interconnections

A computer can process data, pictures, sound and graphics. They can solve highly complicated problems quickly and accurately. A computer as shown in Figure performs basically five major computer operations or functions irrespective of their size and make. These are:

- it accepts data or instructions by way of input,
- it stores data,
- it can process data as required by the user,
- it gives results in the form of output, and
- it controls all operations inside a computer.

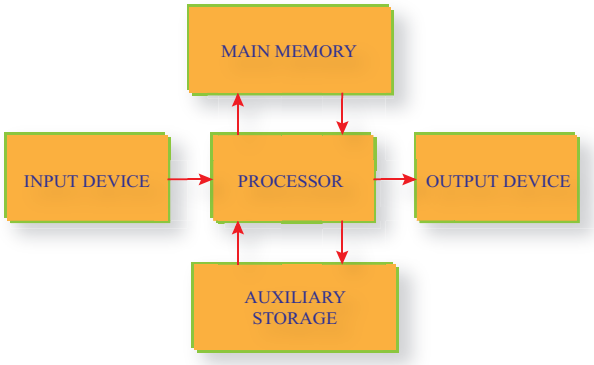


Figure 1.2: Basic computer operations.

1. ***Input***: This is the process of entering data and programs in to the computer system. You should know that computer is an electronic machine like any other machine which takes as inputs raw data and performs some processing giving out processed data. Therefore, the input unit takes data from us to the computer in an organized manner for processing.
2. ***Storage***: The process of saving data and instructions permanently is known as storage. Data has to be fed into the system before the actual processing starts. It is because the processing speed of Central Processing Unit (CPU) is so fast that the data has to be provided to CPU with the same speed. Therefore the data is first stored in the storage unit for faster access and processing. This storage unit or the primary storage of the computer system is designed to do the functionality. It provides space for storing data and instructions.

The storage unit performs the some major functions are:

- All data and instructions are stored here before and after processing.
- Intermediate results of processing are also stored here.

3. ***Processing***: The task of performing operations like arithmetic and logical operations is called processing. The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided. It is then sent back to the storage unit.
4. ***Output***: This is the process of producing results from the data for getting useful information. Similarly the output produced by the computer after processing must also be kept somewhere inside the computer before being given to in human readable form. Again the output is also stored inside the computer for processing.
5. ***Control***: The manner how instructions are executed and the operations are performed. Controlling of all operations like input, processing and output

are performed by control unit. It takes care of step by step processing of all operations inside the computer.

Functional Units

In order to carry out the operations the computer allocates the task between its various functional units. The computer system is divided into three separate units for its operation. They are:

- Arithmetic logical unit
- Control unit.
- Central processing unit.

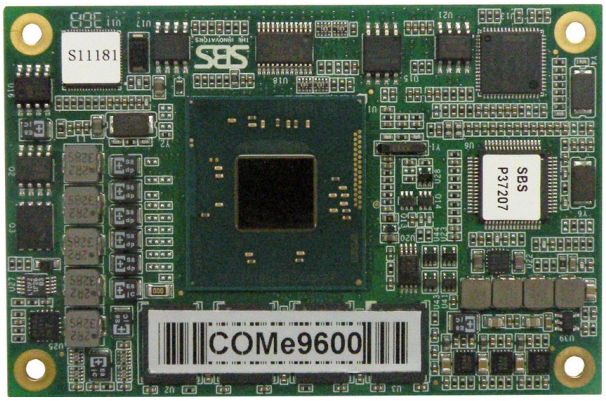
Arithmetic Logical Unit (ALU): After you enter data through the input device it is stored in the primary storage unit. The actual processing of the data and instruction are performed by Arithmetic Logical Unit. The major operations performed by the ALU are addition, subtraction, multiplication, division, logic and comparison. Data is transferred to ALU from storage unit when required. After processing the output is returned back to storage unit for processing or getting stored.

Control Unit (CU): The next component of computer is the Control Unit, which acts like the supervisor seeing that things are done in proper fashion. Control Unit is responsible for co coordinating various operations using time signal. The control unit determines the sequence in which computer programs and instructions are executed. Things like processing of programs stored in the main memory, interpretation of the instructions and issuing of signals for other units of the computer to execute them. It also acts as a switch board operator when several users access the computer simultaneously. Thereby it coordinates the activities of computer's peripheral equipment as they perform the input and output.

Central Processing Unit (CPU): The ALU and the CU of a computer system are jointly known as the central processing unit. You may call CPU as the brain of any computer system. It is just like brain that takes all major decisions, makes all sorts of calculations and directs different parts of the computer functions by activating and controlling the operations.

CASE STUDY

INTEL 4TH GENERATION CORE COM EXPRESS PLATFORM POWERS UP ULTRASOUND DEVICES



Introduction

Ultrasound devices are widely used for medical imaging, detection and measurement. Such applications require multi-channel data acquisition boards based on high-speed serial buses compliant with strict medical regulations. The flexibility in design makes COM Express an ideal solution that meets all the challenges required in developing ultrasound devices.

Application Requirements

An ultrasound device consists of a transducer, a central processing unit or computer connected to a display, imaging recorder, keyboard and power. The transducer sends sound waves which are reflected back to the transducer after they bounce off the structures that are the focus of the ultrasound. The central processing unit measures the echo intensities and speed and are converted into electronic images that appear on the display.

The application requirements are:

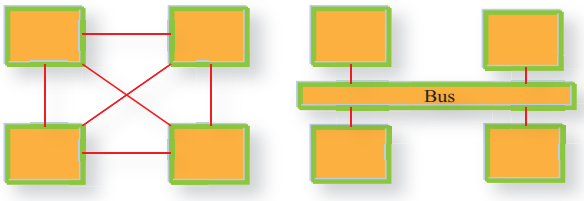
- Supports high-speed serial bus i.e. PCIe x16 expansion for data acquisition
- Supports multiple displays i.e. HDMI/ DVI/ Display Port
- High computing power with low power consumption
- Standards-compliant migration for extended product lifecycle

Solutions and Benefits

- SOM-5894 in COM Express Basic form factor was adopted to assist the development of ultrasound devices. SOM-5894 provides the following benefits:
- *High-Performance Processor*: SOM-5894 incorporates the Intel Shark Bay processor supporting Intel 4th Generation i7 CPU, featuring major upgrades in computing performance.
- *Outstanding Graphics Engine*: SOM-5894 supports hardware coding/decoding, DirectX10.1 / OpenGL3.0, HD speeding, doubled 3D graphics capabilities covering multi-displays HDMI/ DVI/ Display Port, and HD output.
- *Optimized Power Consumption*: CPUs of lower power consumption are available for enhancing system reliability.
- *Pin-Out Type Upgrade*: SOM-5894 supports PICMG COM 2.1 Type 6 Pin-Out, high-speed serial bus (PCIE, SATA III, USB3.0). Clients can customize PGA, DSP, HDMI, DP pin out to fulfill different application needs.
- *Flexible Modular Design*: Product migration and replacement are easy to do thanks to the modular design concept on COM modules and boards. This can also accelerate product time-to-market.
- *Software Support*: SOM-5894 supports various embedded OS and iManager intelligent self-management tool.

1.4 BUSES

A bus, in computing, is a set of physical connections (cables, printed circuits, etc.) which can be shared by multiple hardware components in order to communicate with one another.



The purpose of buses is to reduce the number of "pathways" needed for communication between the components, by carrying out all communications over a single data channel. This is why the metaphor of a "data highway" is sometimes used.

If only two hardware components communicate over the line, it is called a hardware port (such as a serial port or parallel port).

Characteristics

A bus is characterized by the amount of information that can be transmitted at once. This amount, expressed in bits, corresponds to the number of physical lines over which data is sent simultaneously. A 32-wire ribbon cable can transmit 32 bits in parallel. The term "width" is used to refer to the number of bits that a bus can transmit at once.

Additionally, the bus speed is also defined by its frequency (expressed in Hertz), the number of data packets sent or received per second. Each time that data is sent or received is called a cycle.

This way, it is possible to find the maximum transfer speed of the bus, the amount of data which it can transport per unit of time, by multiplying its width by its frequency. A bus with a width of 16 bits and a frequency of 133 MHz, therefore, has a transfer speed equal to:

$16 * 133.10^6 = 2128*10^6$ bit/s,

Or $2128*10^6/8 = 266*10^6$ bytes/s

Or $266*10^6 /1000 = 266*10^3$ KB/s

Or $259.7*10^3 /1000 = 266$ MB/s

1.4.1 Bus Architecture

In reality, each bus is generally constituted of 50 to 100 distinct physical lines, divided into three subassemblies:

- The address bus (sometimes called the memory bus) transports memory addresses which the processor wants to access in order to read or write data. It is a unidirectional bus.
- The data bus transfers instructions coming from or going to the processor. It is a bidirectional bus.
- The control bus (or command bus) transports orders and synchronization signals coming from the control unit and travelling to all other hardware components. It is a bidirectional bus, as it also transmits response signals from the hardware.

The Primary Buses

There are generally two buses within a computer:

- The internal bus (sometimes called the front-side bus, or FSB for short). The internal bus allows the processor to communicate with the system's central memory (the RAM).

- The expansion bus (sometimes called the input/output bus) allows various motherboard components (USB, serial, and parallel ports, cards inserted in PCI connectors, hard drives, CD-ROM and CD-RW drives, etc.) to communicate with one another. However, it is mainly used to add new devices using what are called expansion slots connected to the input/output bus.

Chipset

A chipset is the component which routes data between the computer's buses, so that all the components which make up the computer can communicate with each other. The chipset originally was made up of a large number of electronic chips, hence the name. It generally has two components:

- *The Northbridge* (also called the memory controller) is in charge of controlling transfers between the processor and the RAM, which is way it is located physically near the processor. It is sometimes called the GMCH, for Graphic and Memory Controller Hub.
- *The Southbridge* (also called the input/output controller or expansion controller) handles communications between peripheral devices. It is also called the ICH (I/O Controller Hub). The Term Bridge is generally used to designate a component which connects two buses.

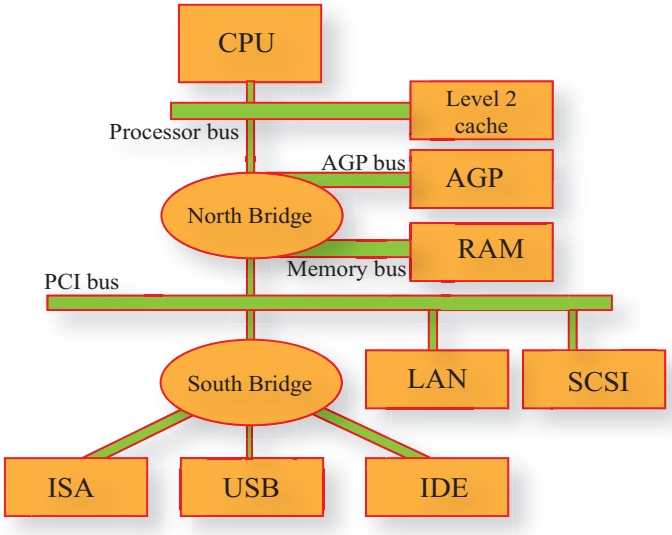


Figure 1.3: Components of chipset.

It is interesting to note that, in order to communicate, two buses must have the same width. That explains why RAM modules sometimes have to be installed in pairs

(*for example*, early Pentium chips, whose processor buses were 64-bit, required two memory modules each 32 bits wide).

Here is a Table which gives the specifications for the most commonly used buses:

Table 1.15: Specifications of buses

Standard	Bus width (bits)	Bus speed (MHz)	Bandwidth (MB/sec)
ISA 8-bit	8	8.3	7.9
ISA 16-bit	16	8.3	15.9
EISA	32	8.3	31.8
VLB	32	33	127.2
PCI 32-bit	32	33	127.2
PCI 64-bit 2.1	64	66	508.6
AGP	32	66	254.3
AGP (x2 Mode)	32	66x2	528
AGP (x4 Mode)	32	66x4	1056
AGP (x8 Mode)	32	66x8	2112
ATA33	16	33	33
ATA100	16	50	100
ATA133	16	66	133
Serial ATA (S-ATA)	1		180
Serial ATA II (S-ATA2)	2		380
USB	1		1.5
USB 2.0	1		60
FireWire	1		100
FireWire 2	1		200
SCSI-1	8	4.77	5
SCSI-2 - Fast	8	10	10
SCSI-2 - Wide	16	10	20
SCSI-2 - Fast Wide 32 bits	32	10	40
SCSI-3 - Ultra	8	20	20
SCSI-3 - Ultra Wide	16	20	40
SCSI-3 - Ultra 2	8	40	40
SCSI-3 - Ultra 2 Wide	16	40	80
SCSI-3 - Ultra 160 (Ultra 3)	16	80	160
SCSI-3 - Ultra 320 (Ultra 4)	16	80 DDR	320
SCSI-3 - Ultra 640 (Ultra 5)	16	80 QDR	640

1.4.2 Types of Buses and Bus Arbitration

The types of buses are as:

Local Buses

A local bus is the simplest buses consist of set of wires. Within the central processing unit (CPU), local buses can generally be divided in three types. They are address buses, data buses or control buses.

- Address buses tend to be specialized in purpose and are usually unidirectional. It carries an address from the CPU to memory or I/O devices.
- Data buses tend to be more general in purpose and are bidirectional. It carries data between the CPU and memory or I/O devices.
- Control buses carry signals from the control unit to other components of the computer and back to control unit.

System Buses

Unlike local buses, system buses are independent functional components of many computers. Each system bus has its own control circuit, called a bus controller, and within each bus controller is an arbiter, which process requests to use the bus. The bus controller may be distributed among the devices that use the bus. Systems buses generally connect system components together, such as the CPU, I/O system and often the main-memory system, and designers often optimize system buses for transferring data between I/O devices and main-memory.

A key distinction between system buses and local buses is that system buses tend to have well-documented and stable definitions, so that designers can attach a wide variety of devices to them. Local buses are processor-specific and not widely documented. They tend to be proprietary. Example of system buses are DEC UNIBUS and the Apple NuBus.

Bus Lines

The wires that comprise a bus are called lines.

Four main categories of bus lines:

- *Data Lines*: Carry data from one place to another.
- *Address Lines*: Specify the recipient of data on the bus.

- *Control Lines*: Provide control for the synchronization and operation of the bus and the modules to which it is connected.
- *Power Lines*: Provide power to various components connected to the bus.

Advantages/Disadvantages of a "Bus" to a "Network"

Data bus networks have quite a few advantages compare the other network topologies such as point-to-point links in these applications. Generally, a data bus can make possible the interconnection of a set of terminals when the number of terminals is so large that interconnection through individual point-to-point links becomes unusable. A data bus topology can provide large configurationally flexibility. Terminals can be added to the network or moved to different locations without major revisions in the cabling layout. Both of these advantages are of particular importance in applications in which cable installation is the dominant system cost, *for example* in shipboard applications.

Purpose of Arbitration

In a computer system, all the devices communicate with the other device are connected to the main board over a same bus. If two or more I/O (input and output) devices try to use the bus at the same time to access the main board, there will be a conflict arise. Therefore, bus arbitration is created is to resolve the problem. Bus arbiter is a circuit to coordinate the activities of devices request for memory transfer using the bus. The process that runs by bus arbiter is name as Bus Arbitration. It is to prevent two or more I/O devices initiating transfers at the same time.

The bus arbitration mechanism is designed to allow high priority devices such as the processor and RAM get first access to the bus, while the other devices (disks, video cards, sound cards etc.) get lower priority, and often have to wait to access the bus. The prioritization is numbered interrupts to priority systems. The lower the numbered interrupts will have the higher priority. On many systems, the CPU has interrupt 0. Therefore, CPU always goes first on the bus. Meanwhile, there are time slices provided over the bus.

The device that uses to allow for initiating data transfers on the bus is called as bus master. Therefore, only one bus master can exist at a time. When the bus master relinquished its control, the other device also can act as bus master. However, the process of transferring the bus mastership from one device to another device has to coordinate carefully to take account of the needs of various devices. The bus arbitration has been design to use by high priority device. Therefore, scheduling function will be performs by bus arbiter. Bus arbiter can be part of the processer or separate unit that connected to bus.

It shows a basic arrangement example of processor contains the bus arbiter circuitry. The processor is act as the bus master; the DMA (direct memory access) controller can gain the bus mastership. DMA controllers activate the Bus-request line, BR before it gains the bus mastership. The Signal of Bus- request line is transfer using the logical OR concept from the bus-request line to the other I/O devices. When the bus request is activated, the processor will activates the bus Grant signal, BG1, gives permission to the DMA controller use the bus when it became free. The signal is using a daisy-chain arrangement to connect all DMA controllers. Therefore, when DMA controller 1 request for the bus, it will block the propagation of the grant signal to other I/O device automatically. Else, it will assert BG2 to grant downstream. The bus master indicates all I/O devices that it is using the bus by activating another open collector line call as Bus Busy, BBSY. Therefore, a DMA controller need to wait for Bus busy to inactive after the Bus Grant signal send. In the situation, the DMA controller can assumes the bus mastership.

Multiplexing

Multiplexed bus is a type of bus structure which the number of signal lines represented by the bus is less than the number of bits of data, addresses, and control information being transferred between devices of the computer system. *For example*, if a multiplexed address bus use 8 signal lines to transmit 16 bits of address information. The information is transferred sequentially where the additional control line is being used for sequencing the transfer.

In another example, the system represent a master control unit (MCU) connecting to one or more receiver-transmit unit (RTU) by a data bus. The MCU transmits a message to the RTU for comprising a synchronization pulse of known duration and successive time spaced timing signals separated by time duration T marking the boundaries of data bits to be transmitted from the RTU to the MCU. The RTU is including a clock pulse source which utilizes the synchronization pulse to determine the frequency of the clock pulse source in P pulses per duration T. The value P is used in conjunction with the timing signals to create properly timed data determining signals in the data bits.

Point to Point Topology

Point to point (PTP) is directly connects two nodes to together. This is some example of using PTP to connect 2 nodes together.

- Two computers communicating via modems.
- A mainframe terminal communicating with a front end processor.
- A workstation communicating along a parallel cable to a scanner.

In a point-to-point topology, all the devices are connected with a shared switch. The switch is different from the shared topology. Computational components that are connected using a point-to-point topology do not need to use any type of bus arbitration scheme. In state, the shared switch breaks the continuous stream of data on the bus into data packets that are routed to the individual devices. Using this method, the shared switch able establishes point-to-point connections between the different devices. From an individual device's perspective, a connection has to be a private, direct, continuous connection to another device. The connection may comprise one or more two-way of serial-connections. By increasing the number of lanes of a connection, therefore the bandwidth of the connection can by increase. An example of point-to-point bus topology is the implementation in peripheral component interface expresses ("PCI Express"). PCI Express is a computer expansion card standard designed to replace the older PCI, PCI-X, and AGP standards. It was introduced by Intel in 2004; PCIe is the latest standard for expansion cards that is available on mainstream personal computers.

Multipoint

Multipoint is a bus topologies that carries signals to several destinations. It is also known as multidrug bus (MDB) or broadcast bus. Multipoint bus usually requires addressing signals on the bus to identify the addressed destination. Example: Ethernet. It is also a computer bus in which all components are connected to same set of electrical wires.

1.4.3 Bus and Memory Transfer

- Rather than connecting wires between all registers, a common bus is used
- A bus structure consists of a set of common lines, one for each bit of a register
- Control signals determine which register is selected by the bus during each transfer
- Multiplexers can be used to construct a common bus
- Multiplexers select the source register whose binary information is then placed on the bus
- The select lines are connected to the selection inputs of the multiplexers and choose the bits of one register

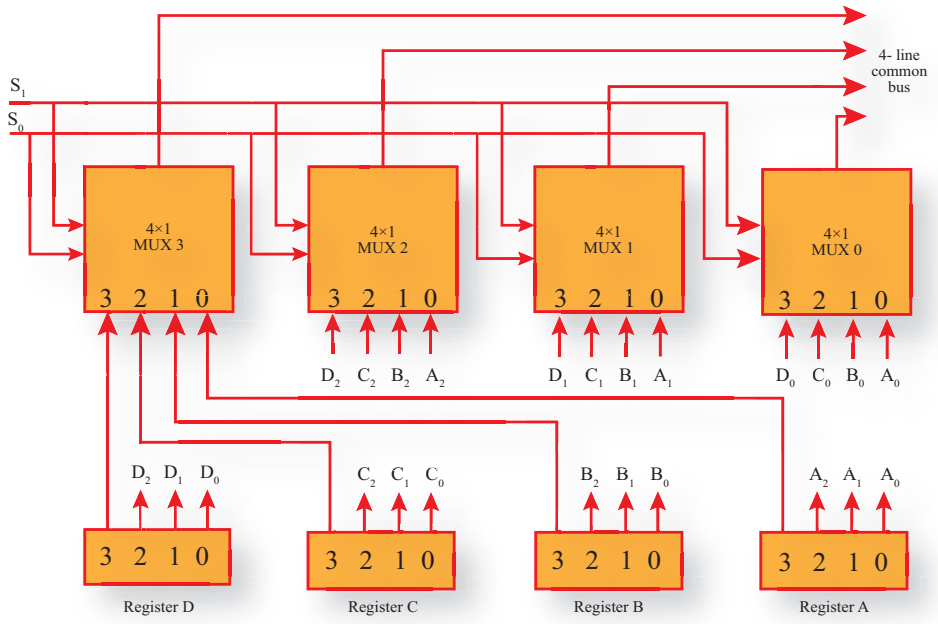


Figure 1.4: Bus system for four registers.

- In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus
- This requires n multiplexers one for each bit
- The size of each multiplexer must be k x 1
- The number of select lines required is log k
- To transfer information from the bus to a register, the bus lines are connected to the inputs of all destination registers and the corresponding load control line must be activated
- Rather than listing each step as BUS ← C, R1 ← BUS, use R1 ← C, since the bus is implied
- Instead of using multiplexers, three-state gates can be used to construct the bus system
- A three-state gate is a digital circuit that exhibits three states
- Two of the states are signals equivalent to logic 1 and 0
- The third state is a high-impedance state – this behaves like an open circuit, which means the output is disconnected and does not have a logic significance



ROLE MODEL

JOHN VON NEUMANN: THE SCIENTIFIC GENIUS WHO PIONEERED THE MODERN COMPUTER, GAME THEORY, AND NUCLEAR DETERRENCE

John von Neumann was a famous Hungarian-American mathematician, who is still revered for his unparalleled contributions to disciplines like mathematics, science, economics and many more. Born in a wealthy family to Jewish parents, he was a child prodigy, exhibiting great analytic and computing skills. He was able to divide eight - digit numbers in mind in very early ages, which proves his great analytical and computing skills. He was a passionate learner and was interested in many subjects like physics, economics, statistics, history etc., apart from mathematics. He was a prominent member of the Manhattan Project and the Institute for Advanced Study in Princeton. His unusual abilities astonished many great personalities like mathematician Jean Dieudonne Peter Lax, Hans Bethe etc. Hans Bethe once said, "I have sometimes wondered whether a brain like von Neumann's does not indicate a species superior to that of man". Read through this biography to know more about the career and life of this genius.

Early Life

John von Neumann was born in Budapest, the capital city of Hungary, on 28 December 1903. He was the elder son of Neumann Miksa and Kann Margit. The nature of the numbers and the mathematical logic induced interest in him at a very early age. However, mathematics was not the only subject which fascinated him; Neumann was greatly interested in history as well that he went through forty four volumes of universal history at an early age of eight. This indicated that Neumann found comfort dealing with both logical and social areas. He was fortunate enough to have very supportive parents, who stood by his interests. In 1914, at the age of ten, he joined the Lutheran Gymnasium, one among the three best organizations

in Budapest. He published his first paper in the Journal of the German Mathematical Society in 1922, which dealt with the zeros of certain minimal polynomials.

Berlin, Zurich and Budapest

Neumann had little interest towards chemistry and engineering but, his father encouraged him to pursue engineering as it was considered as an elite career during that time. He attended Pázmány Péter University in Budapest to pursue advanced doctorate in mathematics along with completing his undergraduate course, diploma in chemical engineering from ETH Zurich. In his thesis for PhD, he wanted to give an attempt to the axiomatization of the set theory which was developed by George Cantor. It was an achievement indeed that a seventeen year old boy pursued both undergraduate and PhD simultaneously. He received good scores for his exams for both the courses and thus, completed under graduation degree in chemical engineering and PhD in mathematics at the age of twenty-two.

Quantum Mechanics

After receiving his degrees in 1926, Neumann attended the university in Gottingen, Germany, where he dealt with quantum mechanics. He was creative and an original thinker, and came up with complete and logical concepts. He worked on the theories of quantum mechanics in 1926 to reconcile and improve them. Neumann worked to find out the similarities between wave mechanics and matrix mechanics. Also, von Neumann worked on the rules of 'abstract Hilbert space' and devised a mathematical structure in terms of quantum theory.

Personal Life

During the time from 1927-1929, after Neumann formulated the quantum mechanics, he attended extensive conferences and colloquia. By 1929, he wrote about 32 papers in German. These papers were was well-organized so that the other mathematicians could incorporate Neumann's works into their theories. By this time, he became famous in the academic world, owing to his creative yet new ideas. By the end of 1929, he was offered a post of lecturer at Princeton in America. It was during the same time, when he married Mariette Kovesi, his childhood friend. In 1935, Mariette gave birth to a girl child, Marina. The marital relationship did not last long; they separated in 1936. Mariette went back to Budapest and Neumann drifted around Europe for a brief period and returned to United States. He met Klari Dan on his way to Budapest and married her in 1938.

Death

John von Neumann was diagnosed with cancer. Despite his ailment and weakness, he took part in the ceremonies organized to honor him, in a wheel chair, and kept contact with family and friends during this time. He passed away on 8 February 1957.

What Is The Moral Here?

Neumann contributed to the Los Alamos project devised by the American government in the development of the nuclear weapons and also in developing the concept and design of explosive lenses. The mathematical modelling which he used at Los Alamos helped him in the development of the modern computer. Along with gathering resources, he also funded the project for the development of the modern computer. He worked on the architecture of the computer as well. His efforts made other scientists realize that computer is not just a 'bigger calculator'. Quantum logic, game theory, linear programming and mathematical statistics are some of the many contributions he had made to the field of science.

SUMMARY

- A computer is a programmable machine. It allows the user to store all sorts of information and then 'process' that information, or data, or carry out actions with the information, such as calculating numbers or organising words.
- Digital computers are those that operate with information, numerical or otherwise, represented in a digital form. Such computers process data into a digital value (in 0s and 1s).
- Minicomputer is a midsized computer. In size and power, minicomputers lie between workstations and mainframes.
- A palmtop is a small computer that can be carried along in the pocket or the briefcase.
- Laptop computers have an L-shape design and the screen can be lowered and closed to allow for easy transportation of the machine.
- Printable characters are things like characters (supper and lower case), numbers and punctuation.
- Hexadecimal refers to a base 16 number system. Uses this in computer science for only one reason; it makes it easy for humans to think about binary numbers.
- The sign bit is 0 if the number is positive and 1 if the number is negative.
- In computing and electronic systems, binary-coded decimal (BCD) or, in its most common modern implementation, packed decimal, is an encoding for decimal numbers in which each digit is represented by its own binary sequence.
- The IEEE single precision floating point standard representation requires a 32 bit word, which may be represented as numbered from 0 to 31, left to right.
- In the fifth generation, the VLSI technology became ULSI (Ultra Large Scale Integration) technology, resulting in the production of microprocessor chips having ten million electronic components.

KNOWLEDGE CHECK

1. **Which of the following is first generation of computer?**
   a. EDSAC
   b. IBM-1401
   c. CDC-1604
   d. ICL-2900
2. **Chief component of first generation computer was ……..**
   a. Transistors
   b. Vacuum Tubes and Valves
   c. Integrated Circuits
   d. None of above
3. **FORTRAN is ……**
   a. File Translation
   b. Format Translation
   c. Formula Translation
   d. Floppy Translation
4. **EEPROM stands for ……**
   a. Electrically Erasable Programmable Read Only Memory
   b. Easily Erasable Programmable Read Only Memory
   c. Electronic Erasable Programmable Read Only Memory
   d. None of the above
5. **Second Generation computers were developed during ……**
   a. 1949 to 1955
   b. 1956 to 1965
   c. 1965 to 1970
   d. 1970 to 1990
6. **Identify the device through which data and instructions are entered into a computer**
   a. Software
   b. Output device
   c. Input device
   d. Memory

7. **Computer Moniter is also known as :**
   a. DVU
   b. UVD
   c. VDU
   d. CCTV
8. **Name of the screen that recognizes touch input is :**
   a. Recog screen
   b. Point Screen
   c. Touch Screen
   d. Android Screen
9. **What is smallest unit of the information?**
   a. A bit
   b. A byte
   c. A block
   d. A nibble
10. **The main computer that stores the files that can be sent to computers that are networked together is:**
   a. Clip art
   b. Mother board
   c. Peripheral
   d. File server

REVIEW QUESTIONS

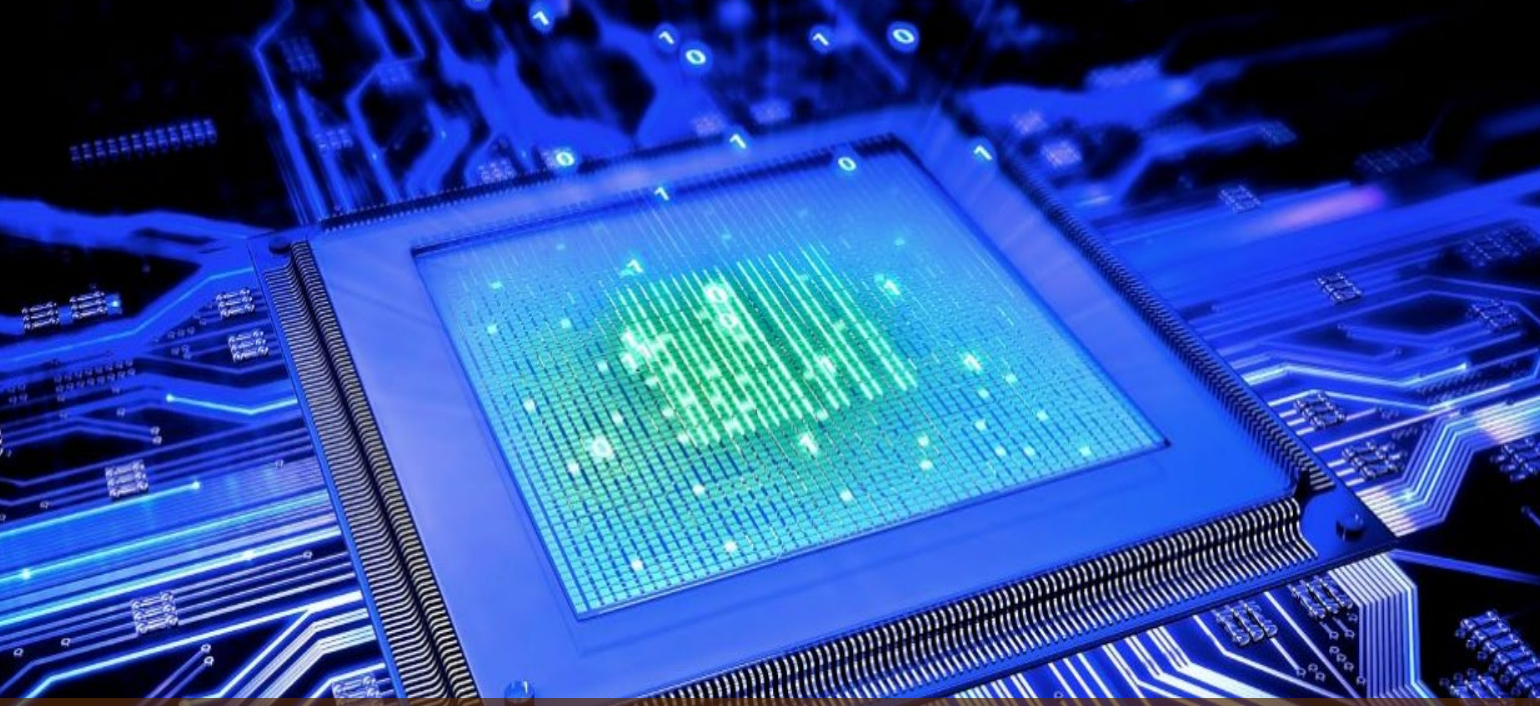
1. Discuss about the major parts of computers.
2. What do you understand by excess-3 code?
3. Explain the digital computer generation.
4. What are the functions of buses?
5. What do you mean by IEEE standard?

Check Your Result

1. (a)	2. (b)	3. (c)	4. (a)	5. (b)
6. (c)	7. (c)	8. (c)	9. (a)	10. (d)

REFERENCES

1. Cohen, Bernard (2000). Howard Aiken, Portrait of a computer pioneer. The MIT press. ISBN 978-0-262-53179-5.
2. Peter J. Denning, Great principles in computing curricula, Technical Symposium on Computer Science Education, 2004.
3. Peter J. Denning. Is computer science science?, Communications of the ACM, April 2005.
4. Stallings, William 2009.Operating Systems: Internals and Design Principles (6thEd) Prentice Hall-Pearson
5. Tedre, Matti (2014). The Science of Computing: Shaping a Discipline. CRC Press, Taylor & Francis.
6. Timothy J.O'Leary and Linda I.O"Leary 2010. Computing Essentials, Introductory The McGraw-Hill Companies, Inc.



CHAPTER 2

CENTRAL PROCESSING UNIT (CPU)

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Understand the process and importance of CPU in a computer systems
2. Explain the concepts of interrupts

"Computers make it easier to do a lot of things, but most of the things they make it easier to do don't need to be done."

—Andy Rooney

INTRODUCTION

A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions. The term has been used in the computer industry at least since the early 1960s. Traditionally, the term "CPU" refers to a processor, more specifically to its processing unit and control unit (CU), distinguishing these core elements of a computer from external components such as main memory and

I/O circuitry. The form, design and implementation of CPUs have changed over the course of their history, but their fundamental operation remains almost unchanged. Principal components of a CPU include the arithmetic logic unit (ALU) that performs arithmetic and logic operations, processor registers that supply operands to the ALU and store the results of ALU operations, and a control unit that fetches instructions from memory and "executes" them by directing the coordinated operations of the ALU, registers and other components.

Most modern CPUs are microprocessors, meaning they are contained on a single integrated circuit (IC) chip. An IC that contains a CPU may also contain memory, peripheral interfaces, and other components of a computer; such integrated devices are variously called microcontrollers or systems on a chip (SoC). Some computers employ a multi-core processor, which is a single chip containing two or more CPUs called "cores"; in that context, single chips are sometimes referred to as "sockets". Array processors or vector processors have multiple processors that operate in parallel, with no unit considered central.

A Central Processing Unit is also called a processor, central processor, or microprocessor. It carries out all the important functions of a computer. It receives instructions from both the hardware and active software and produces output accordingly. It stores all important programs like operating systems and application software. CPU also helps Input and output devices to communicate with each other. Owing to these features of CPU, it is often referred to as the brain of the computer.

CPU is installed or inserted into a CPU socket located on the motherboard. Furthermore, it is provided with a heat sink to absorb and dissipate heat to keep the CPU cool and functioning smoothly.

CPUs are sometimes called microprocessors, although these two terms are not quite interchangeable. The microprocessor, first introduced in the 1970s, reduces the word size of a CPU from 32 bits to 4 bits in an attempt to allow the transistors of the logic circuits to fit on a single part. Often, it takes more than one microprocessor to perform all of the functions of a CPU. Microprocessors are also commonly used in cell phones, automobiles, and children's electronic toys.

2.1 PROCESS AND IMPORTANCE OF CPU IN A COMPUTER SYSTEMS

CPU stands for "central processing unit." This is the pretty much the brain of the computer. It processes everything from basic instructions to complex functions. Any time something needs to be computed, it gets sent to the CPU. Every day, it is computed this; compute that one would think the CPU would need a break after a while. But

no it just keeps on processing. The CPU can also be referred to simply as the "processor."

Keyword

Microprocessor is simply a computer processor that has been configured into the design and function of a microchip.

The CPU is also known as processor, computer processor, **microprocessor**, central processor, "the brains of the computer". Not all central processing units have pins on their bottom sides, but in the ones that do, the pins are easily bent. Take great care when handling, especially when installing onto the motherboard. Each motherboard supports only a certain range of CPU types so always check with the motherboard manufacturer before making a purchase.

Even though the organization of a CPU is very complex, its principle of operation is not hard to understand. Basically, a CPU will read instructions from programs sequentially. Those programs might come from software, games, videos, songs, etc. Since all instruction of those applications are written in programming languages, the computer can convert each instruction into a machine code, also called binary code. Just as its name implies, it is the code which can be understood by machines. Actually, the machine code is very simple as it only contains 0's and 1's. This allows it to be easily understood by the CPU. Let us say you have a 32-bit Windows 7 computer. It will convert each program instruction into a machine code that is 32 characters long and is filled with 0's and 1's. The relative order, and the number of 0's and 1's, defines the meaning of this command.

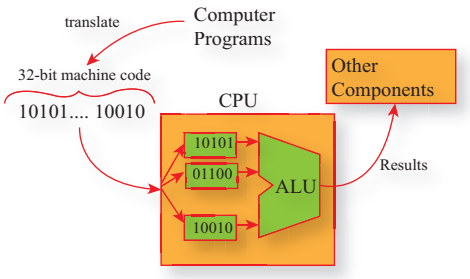


Figure 2.1: Principal of operation of a CPU.

First, the CPU will read this set of 0's and 1's and break it down into several smaller groups. Each group has its own meaning. Then, all of these groups will be forwarded to a place

called ALU (Arithmetic Logic Unit), which can perform the basic arithmetic operations, such as addition, subtraction, etc. Once that is done, it will return the results to the CPU, and the CPU will send the results to the next component, such as memory. This concludes the execution of one instruction. The CPU will repeat these steps and continuously carry out these operations. A lot of people have a wrong impression that a CPU is a command center. They believe it generates all of the commands and the computer will follow those commands and operate actually, the CPU will not generate commands for the computer.

Keyword

Arithmetic Logic Unit is the part of a computer's CPU that allows the computer to make numerical calculations and logical decisions.

The clock rate is one of the main characteristics of the CPU when performance is concerned. Clock rate is the fundamental rate in cycles per second for the frequency of the clock in any synchronous circuit. A single clock cycle toggles between a logical zero and a logical one state. With any particular CPU, replacing the crystal with another crystal that oscillates with twice the frequency will generally make the CPU run with twice the performance. It will also make the CPU produce roughly twice the amount of waste heat.

Engineers are working hard to push the boundaries of the current architectures and are constantly searching for new ways to design CPUs that tick a little quicker or use slightly less energy per clock. This produces new cooler CPUs that can run at higher clock rates.

Scientists also continue to search for new designs that allow CPUs to run at the same or at a lower clock rate as older CPUs, but which get more instructions completed per clock cycle.

The clock rate of a processor is only useful for providing comparisons between computer chips in the same processor family and generation.

Clock rates can be very misleading since the amount of work different computer chips can do in one cycle varies. Clock rates should not be used when comparing different computers or different processor families. Rather, some kind of software benchmarks should be used.

Smartphones are equipped with more advanced embedded chipsets that can do many different tasks depending on their programming.

The performance of the CPU that is at the core of the chipset is vital for the daily user experience and the general computing performance of the smartphone. People tend to use the clock rate of the main CPU to compare the performance of competing end products. But as one already pointed out, the clock rate of a processor is only useful for providing performance comparisons between computer chips in the same processor family and generation. For all other purposes, it is best to use software benchmarks for determining comparative performance

The CPU provides the circuits that implement the computer's instruction set its machine language. It is composed of an **arithmetic-logic unit** (ALU) and control circuits. The ALU carries out basic arithmetic and logic operations, and the control section determines the sequence of operations, including branch instructions that transfer control from one part of a program to another. Although the main memory was once considered part of the CPU, today it is regarded as separate. The boundaries shift, however, and CPU chips now also contain some high-speed cache memory where data and instructions are temporarily stored for fast access.

The ALU has circuits that add, subtract, multiply, and divide two arithmetic values, as well as circuits for logic operations such as AND others is OR (where a 1 is interpreted as true and a 0 as false, so that, for instance, 1 AND 0 = 0). The ALU has several to more than a hundred registers that temporarily hold results of its computations for arithmetic operations or for transfer to main memory.

The circuits in the CPU control section provide branch instructions, which make elementary decisions about what instruction to execute next. *For example*, a branch instruction might be "If the result of the last ALU operation is negative, jump to location A in the program; otherwise, continue with the instruction." Such instructions allow "if-then-else" decisions in a program and execution of a sequence of instructions, such as a "while-loop" that repeatedly does some set of instructions while some condition is met. A related instruction is the subroutine call, which transfers execution to a subprogram and then, after the subprogram finishes, returns to the main program where it left off.

In a stored-program computer, programs and data in memory are indistinguishable. Both are bit patterns strings of 0s and 1s that may be interpreted either as data or as program instructions and both are fetched from memory by the CPU. The CPU has a program counter that holds the memory address (location) of the next instruction to be executed. The basic operation of the CPU is the "fetch-decode-execute" cycle:

Fetch the instruction from the address held in the program counter, and store it in a register.

Decode the Instruction: Parts of it specify the operation to be done, and parts specify the data on which it is to operate. These may be in CPU registers or in memory locations. If it is a branch instruction, part of it will contain the memory address of the next instruction to execute once the branch condition is satisfied.

Keyword

Transistors are an electronic device, which can control the operation of a circuit.

- Fetch the operands, if any.
- Execute the operation if it is an ALU operation.
- Store the result (in a register or in memory), if there is one.
- Update the program counter to hold the next instruction location, which is either the next memory location or the address specified by a branch instruction.

At the end of these steps the cycle is ready to repeat, and it continues until a special halt instruction stops execution.

Steps of this cycle and all internal CPU operations are regulated by a clock that oscillates at a high frequency (now typically measured in gigahertz, or billions of cycles per second). Another factor that affects performance is the "word" size the number of bits that are fetched at once from memory and on which CPU instructions operate. Digital words now consist of 32 or 64 bits, though sizes from 8 to 128 bits are seen.

Processing instructions one at a time, or serially, often creates a bottleneck because many program instructions may be ready and waiting for execution. Since the early 1980s, CPU design has followed a style originally called reduced-instruction-set computing (RISC). This design minimizes the transfer of data between memory and CPU (all ALU operations are done only on data in CPU registers) and calls for simple instructions that can execute very quickly. As the number of **transistors** on a chip has grown, the RISC design requires a relatively small portion of the CPU chip to be devoted to the basic instruction set. The remainder of the chip can then be used to speed CPU operations by providing circuits that let several instructions execute simultaneously, or in parallel.

The first CPU chip was created by Intel back in early 70's. Beside Intel, companies such as IBM, Motorola, Texas Instruments also joined the CPU market and started designing their own CPU. At that time, the CPU was so simple that it could only be

used on a calculator. During the 80's, the CPU had a great evolution. The size increased from 4 bits to 16 bits, then all the way to 32 bits. Its functionality and capability also had a substantial improvement, so they gained a great variety of applications. However, the cost for a CPU declined, and that made the home computer possible. In the 90's, Intel played an important role. They developed the well-known Pentium series processors for home computers. Meanwhile, Advanced Micro Devices (AMD) joined the game and started the match with Intel. Up to the year 2000, the speed and on chip memory of the CPU had been improved significantly. Nowadays, the CPU is in a brand new multi-core era. There is no doubt that the CPU will become much faster and more reliable in the future.

2.1.1 CPU Building Blocks

The CPU itself is an internal component of the computer. Modern CPUs are small and square and contain multiple metallic connectors or pins on the underside. The CPU is inserted directly into a CPU socket, pin side down, on the motherboard.

Each motherboard will support only a specific type (or range) of CPU, so you must check the motherboard manufacturer's specifications before attempting to replace or upgrade a CPU in the computer. Modern CPUs also have an attached heat sink and small fan that go directly on top of the CPU to help dissipate heat.

Two typical components of a CPU are:

- Arithmetic Logic Unit
- Control Unit

Arithmetic Logic Unit (ALU)

The arithmetic logic unit (ALU) is the brawn of the computer, the device that performs the arithmetic operations like addition and subtraction or logical operations like AND, OR. An ALU from four hardware building blocks and illustrates how combinational logic works. Because the MIPS word is 32 bits wide, and needs a 32-bit-wide ALU. Let us assume that one will connect 32 1-bit ALUs to create the desired ALU therefore start by constructing a 1-bit ALU.

The ALU performs arithmetic calculations and data manipulation, e.g. comparisons, sorting, combining, etc. The computer's calculator is a part of the CPU known as the arithmetic logic unit.

It holds data and instructions which are in current use.

Control Unit

The control unit (CU), which extracts instructions from memory and decodes and executes them, calls on the ALU when necessary.

Carries out instructions and tells the rest of the computer system what to do.

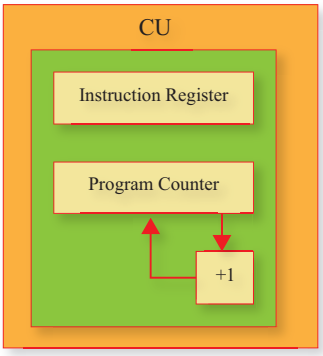


Figure 2.2: Structure of control unit.

Keyword

Instruction Register (IR) is the part of a CPU's control unit that stores the instruction currently being executed or decoded.

This is done by the Control Unit of the CPU which sends command signals to the other components of the system.

Some important points related to control unit is here:

- The control unit directs the entire computer system to carry out stored program instructions.
- The control unit must communicate with both the arithmetic logic unit and main memory.
- The control unit uses the instruction contained in the **instruction register (IR)** to decide which circuits need to be activated.
- The control unit co-ordinates the activities of the other two units as well as all peripheral and auxiliary storage devices linked to the computer.
- The control unit instructs the arithmetic logic unit which arithmetic operations or logical operation is to be performed.
- The control unit is literally in control.

Specialized electronic circuitry in the control unit is designed to decode program instructions held in the main memory. Each

instruction is read from the memory into the instruction register. The process of reading an instruction is often referred to as the fetch-execute process.

Memory Unit

The memory unit is the part of the computer that holds data and instructions for processing.

The main functions of memory unit are here:

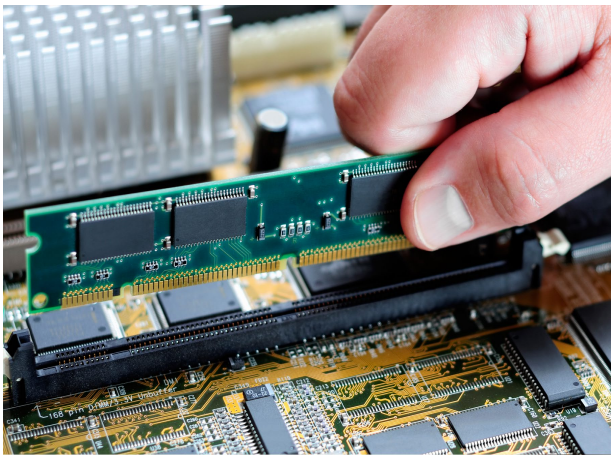
- Although it is closely associated with the CPU, in actual fact it is separate from it.
- Memory associated with the CPU is also called primary storage, primary memory, main storage, internal storage and main memory.
- When load software from a floppy disk, hard disk or CD-ROM, it is stored in the Main Memory.

There are two types of computer memory inside the computer, RAM and ROM.

RAM

The RAM stands for random access memory. This is really the main store and is the place where the programs and software gets stored. When the Central Processing Unit runs a program, it fetches the program instructions from the RAM and carries them out.

If the central processing unit needs to store the results of calculations it can store them in RAM. Random access memory can have instructions READ from it by the CPU and also it can have numbers or other computer data WRITTEN to it by the CPU. When someone switches a computer off, whatever is stored in the RAM gets erased.



ROM

The ROM stands for read only memory. The CPU can only fetch or read instructions from read only memory (or ROM). The ROM comes with instructions permanently stored inside and these instructions cannot be over-written by the computer's CPU. The ROM memory is used for storing special sets of instructions which the computer needs when it starts up.

When he/she switches the computer off, the contents of the ROM does not become erased but remains stored permanently. Therefore it is non-volatile. These are kept in the main store or memory. To understand how the whole system works, consider the Figure 2.3. It shows the basic components of a generalized CPU. An actual CPU may have these components or other with different names that provide the same functions.

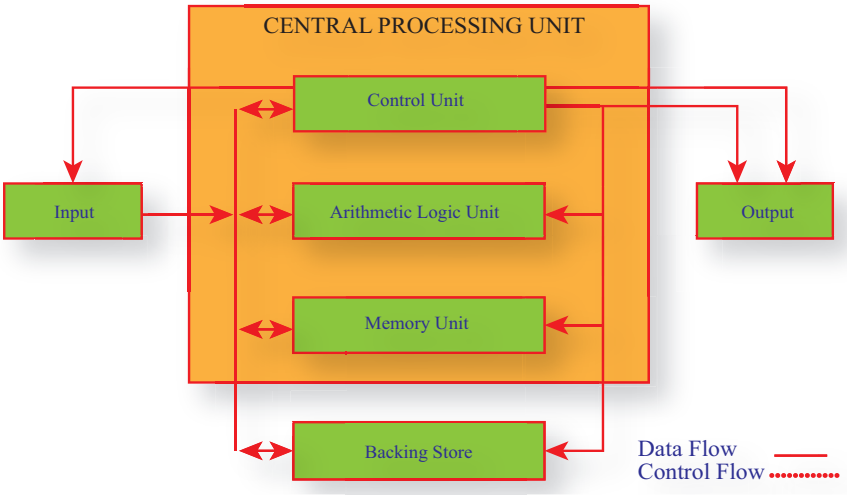


Figure 2.3: CPU building blocks.

2.1.2 Addressing Modes

Addressing mode provide different ways for access an address to given data to a processor. Operated data is stored in the memory location, each instruction required certain data on which it has operate. There are various techniques to specify address of data. These techniques are called addressing modes.

Direct Addressing Mode

In the direct addressing mode, address of the operand is given in the instruction and data is available in the memory location which is provided in instruction. And will move this data in desired location.

The address of the operand is embedded in the instruction code.

Example: (SPIM)

```
beta: .word 2000
lw $11, beta  # load word (32 -bit quantity) at address beta into register $11
              # address of the word is embedded in the instruction code
              # (register $11 will receive value 2000
```

Indirect Addressing Mode

In the indirect addressing mode, the instruction specifies a register which contain the address of the operand. Both internal RAM and external RAM can be access via indirect addressing mode.

The effective address of the operand is the contents of a register or main memory location, location whose address appears in the instruction. Indirection is noted by placing the name of the register or the memory address given in the instruction in parentheses. The register or memory location that contains the address of the operand is a pointer. When an execution takes place in such mode, instruction may be told to go to a specific address. Once it is there, instead of finding an operand, it finds an address where the operand is located.

Two memory accesses are required in order to obtain the value of the operand. *For Example*: (textbook) ADD (A), R0 (address A is embedded in the instruction code and (A) is the operand address = pointer variable)

Example: SPIM - simulating pointers and indirect register addressing

The "C" code is:

```
int *alpha=0x00002004, q=5;
*alpha = q;
```

could be translated into the g assembly code:

```
alpha: .word 0x00002004 # alpha is and address variable # address value is 0x00002004
q: .word 5
....
```

```
lw $10,q        # load word value from address q in into $10
                # $10 is 5
lw $11,alpha        # $11 gets the value 0x0002004
                # this is similar with a load immediate address value
sw $10,($11)        # store value from register $10 at memory location
                # whose address is given by the contents of register $11
                # (store 5 at address 0x00002004)
```

Example: SPIM/SAL - - array pointers and indirect register addressing

```
.data
array1: .byte 1,2,3,4,5,6
.text
__start:
la $3, array1       # array1 is direct addressing mode
add $3, $3,4        # compute the address of the fifth element
sb $0, ($3)    # array1[4]=0 , byte accessing
                # indirect addressing mode
done
```

Immediate Addressing Mode

In the immediate addressing mode, direct data is given in the operand which moves the data in accumulator. It is very fast.

The operand is an immediate value is stored explicitly in the instruction:

Example: SPIM (op-code dest, source)

```
li $11, 3 // loads the immediate value of 3 into register $11
li $9, 8 // loads the immediate value of 8 into register $9
```

Example : (textbook uses instructions type like, opcode source, dest)

move #200, R0; // move immediate value 200 in register R0

Index Addressing Mode

In the index address mode, the effective address of the operand is generated by adding a content value to the contents of the register. This mode is called index address mode.

The address of the operand is obtained by adding to the contents of the general register (called index register) a constant value. The number of the index register and the constant value are included in the instruction code. Index Mode is used to access an array whose elements are in successive memory locations. The content of the instruction code represents the starting address of the array and the value of the index register, and the index value of the current element. By incrementing or decrementing index register different element of the array can be accessed.

For Example: SPIM/SAL - Accessing Arrays

Keyword

Addressing Modes are the method used to determine which part of memory is being referred to by a machine instruction.

```
.data
array1: .byte 1,2,3,4,5,6
.text
__start:
move $3, $0          # $3 initialize index register with 0
add $3, $3,4                 # compute the index value of the fifth
element
sb $0, array1($3)    # array1[4]=0
                     # store byte 0 in the fifth element of the
array
                     # index addressing mode
done
```

The term **addressing modes** refers to the way in which the operand of an instruction is specified. Information contained in the instruction code is the value of the operand or the address of the result/operand.

Register Mode

The name (the number) of the CPU register is embedded in the instruction. The register contains the value of the operand. The number of bits used to specify the register depends on the total number of registers from the processor set.

(SPIM)




```
add $14,$14,$13      # add contents of register $13 plus contents of
                     # register $14 and save the result in register $14
```

No memory access is required for the operand specified in register mode.

Displacement Mode

Similar to index mode, except instead of index register a base register will be used. Base register contains a pointer to a memory location. An integer (constant) is also referred to as a displacement. The address of the operand is obtained by adding the contents of the base register plus the constant. The difference between index mode and displacement mode is in the number of bits used to represent the constant. Index mode is more appropriate for array accessing; displacement mode is more appropriate for structure (records) accessing.

For Example: SPIM/SAL - Accessing fields in structures

```
.data
student: .word 10000 #field code
.ascii "Smith" #field name
.byte # field test
```

```
.byte 80,80,90,100 # fields hw1,hw2,hw3,hw4
.text
__start:
la $3, student        # load address of the structure in $3
                # $3 base register
add $17,$0,90         # value 90 in register $17
                # displacement of field "test" is 9 bytes
                #
sb $17, 9($3)         # store contents of register $17 in field "test"
                # displacement addressing mode
done
```

A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing, which is broadly categorized as displacement 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.

Three of the most common use of displacement addressing is:

- Relative addressing
- Base-register addressing
- Indexing

Relative Addressing:

For 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. Thus, the effective address is a displacement relative to the address of the instruction.

Keyword

In some implementation, 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.

Base-Register Addressing

The reference register contains a memory address, and the address field contains a displacement from that address. The register reference may be explicit or implicit.

Indexing

The address field references a main memory address, and the reference register contains a positive displacement from that address. The register reference is sometimes explicit and sometimes implicit.

Generally index register are used for iterative tasks, 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 system will automatically do this as part of the same instruction cycle.

This is known as auto-indexing. And get two types of auto-indexing; one is auto-incrementing and the other one is auto-decrementing.

Certain registers are devoted exclusively to indexing then auto-indexing can be invoked implicitly and automatically. If general purpose register are used, the auto index operation may need to be signaled by a bit in the instruction.

Auto-indexing using increment can be depicted as follows:

$$EA = A + (R)$$

$$R = (R) + 1$$

Auto-indexing using decrement can be depicted as follows:

$$EA = A + (R)$$

$$R = (R) - 1$$

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 post indexing

$$EA = (A) + (R)$$

First, the contents of the address field are used to access a memory location containing an address. This address is then indexed by the register value.

With pre-indexing, the indexing is performed before the indirection:

$$EA = (A + (R))$$

An address is calculated, the calculated address contains not the operand, but the address of the operand

Stack Addressing

A stack is a linear array or list of locations. It is sometimes referred to as a pushdown list or last-in-first-out queue. A 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 filled. Associated with the stack is a pointer whose value is the address of the top of the stack. The stack pointer is maintained in a register. Thus, references to stack locations in memory are in fact register indirect addresses.

The stack mode of addressing is a form of implied addressing. The machine instructions need not include a memory reference but implicitly operate on the top of the stack.

Auto increment/Auto decrement Mode

Auto increment is a special case of indirect register mode. The register, whose number is included in the instruction code, contains the address of the operand.

Auto-increment Mode = after operand addressing

The contents of the register is incremented

Decrement Mode = before operand addressing

The contents of the register is decrement



Transistorized CPUs during the 1950s and 1960s no longer had to be built out of bulky, unreliable, and fragile switching elements like vacuum tubes and electrical relays.

For Example: SPIM/SAL simulating auto-increment/auto-decrement addressing mode (MIPS has no auto-increment/auto-decrement mode)

```
lw $3, array1($17)  #load in reg. $3 word at address array1($17)
addi $17,$17,4              #increment address (32-bit words) after accessing
                        #operand this can be re-written in a "auto-increment like mode":
lw+ $3,array1($17)     # lw+ is not a real MIPS instruction
subi $17,$17,4              # decrement address before accessing the operand
lw $3,array1($17)
```

The sequence can be rewritten proposing an "auto-decrement instruction", not real in MIPS architecture are:

-lw $3, array1 ($17)

2.2 CONCEPTS OF INTERRUPTS

An interrupt is the automatic transfer of software execution in response to a hardware event that is asynchronous with the current software execution. This hardware event is called a trigger. The hardware event can either be a busy to ready transition in an external I/O device (like the UART input/output) or an internal event (like bus fault, memory fault, or a periodic timer). When the hardware needs service, signified by a busy to ready state transition, it will request an interrupt by setting its trigger flag. A thread is defined as the path of action of software as it executes. The execution of the interrupt service routine is called a background thread. This thread is created by the hardware interrupt request and is killed when the interrupt service routine returns from interrupt (e.g., by executing a BX LR). A new thread is created for each interrupt request. It is important to consider each individual request as a separate thread because local variables and registers used in the interrupt service routine are unique and separate from one interrupt event to the next interrupt. In a multi-threaded system, and considers the threads as cooperating to perform an overall task. Consequently anyone will develop ways for the threads to communicate (e.g., FIFO) and to synchronize with each other. Most embedded systems have a single common overall goal. On the other hand, general-purpose computers can have multiple unrelated functions to perform. A process is also defined as the action of software as it executes. Processes do not necessarily cooperate towards a common shared goal. Threads share access to I/O

devices, system resources, and global variables, while processes have separate global variables and system resources. Processes do not share I/O devices.

Interrupt means event, which invites attention of the processor on occurrence of some action at hardware or software interrupt instruction event.

In response to the interrupt, the routine or program, which is running presently interrupts and an interrupt service routine (ISR) executes. ISR is also called device driver of the devices and called exception or signal or trap handler in case of software interrupts; Processor executes the program, called interrupt service routine or signal handler or trap handler or exception handler or device driver, related to input or output from the port or device or related to a device function on an interrupt and does not wait and look for the input ready or output completion or device-status ready or set

When a Process is executed by the CPU and when a user Request for another Process then this will create disturbance for the Running Process. This is also called as the Interrupt.

2.2.1 Interrupt Hardware

When a device or port is ready, a device or port generates an interrupt, or when it completes the assigned action or when a timer overflows or when a time at the timer equals a preset time in a compare register or on setting a status flag (*for example*, on timer overflow or compare or capture of time) or on click of mice in a computer.

Hardware interrupt generates call to an ISR

When software run-time exception condition (*for example*s, division by 0 or overflow or illegal op-code detected) the processor-hardware generates an interrupt, called trap, which calls an ISR

When software run-time exception condition defined in a program occurs, and then a software instruction (SWI) is executed called software interrupt or exception or signal, which calls an ISR.

2.2.2 Types of Interrupts

Interrupts can be generated by User, Some Error Conditions and also by Software's and the hardware's. But CPU will handle all the Interrupts very carefully because when

Generally there are three types of interrupts and these are as follows:

- External Interrupt
- Internal Interrupt.
- Software Interrupt.

REMEMBER

Interrupts are generated then the CPU must handle all the Interrupts Very carefully means the CPU will also provide response to the Various Interrupts those are generated. So that when an interrupt has occurred then the CPU will handle by using the fetch, decode and Execute operations.

External Interrupt

The External Interrupt occurs when any Input and Output Device request for any Operation and the CPU will Execute that instructions first *For Example* When a Program is executed and when move the Mouse on the Screen then the CPU will handle this External interrupt first and after that he will resume with his Operation.

Internal Interrupts

The Internal Interrupts are those which are occurred due to some problem in the execution *For Example* When a user performing any Operation which contains any Error and which contains any type of Error. So that Internal Interrupts are those which are occurred by the some operations or by Some Instructions and the Operations those are not Possible but a user is trying for that Operation. And

The Software Interrupts

The Software Interrupts are those which are made some call to the system *for example* while anyone is processing some instructions and when they want to execute one or more application programs.

2.2.3 Instruction and Execution Interrupt Cycle

The Instruction fetch execute cycle is one of the most important mental models of computation as aptly put by Prof. Rockford Ross. This embodies the basic principle of how all modern processors work. This functional model has remained more or less the same over the decades no matter how and when the development of processors have taken place ever since the days of Von Newman architecture to today's super computers. The principles are fairly simple and can be easily generalized to

any processor or operating system. It is very important to appreciate the fact that an operating system (OS) is just like any other program albeit a little more complex as compared to user written programs.

The OS provides the functionality to load and execute other programs and thus possesses certain privileges which user programs do not possess. A user program can and does request for OS intervention through a mechanism called system calls e.g. trap, page fault etc.

Once a computer has been powered on it performs a continuous cycle are as follows:

- Fetch next instruction from memory
- Decode the instruction
- Execute the instruction

An instruction as the name instructs the computer what to do. In simple terms, every line of a program that one as users writes instructs the computer to perform a series of operations. Anyone may argue that the programs comprise of the instructions belonging to one of those of the high level languages like C, C++, and Java etc. A computer understand these high level instructions by converting them into a machine understandable form known as machine language comprising of 1's and 0's. The instruction in C is:

```
int cnt;

cnt = cnt + 2;
```

May get translated into the corresponding machine language as:

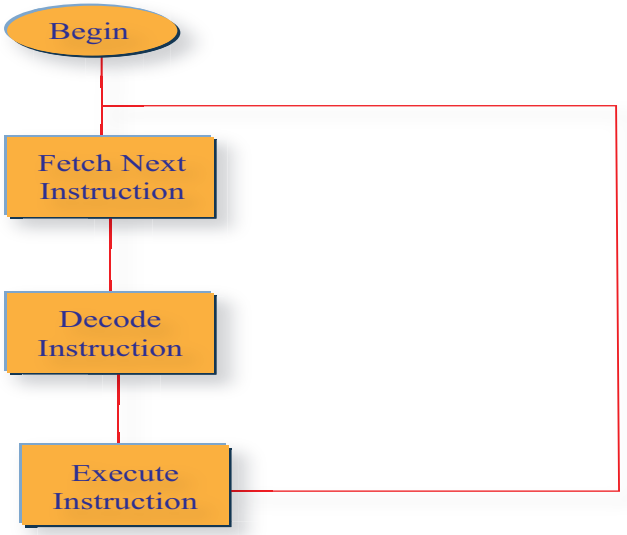
```
00010101010111110101010101000000111111…..
```

One as users would not be able to understand this machine language. To make better sense one resort to Assembly Language where each machine language instruction is assigned a code known as a mnemonic. So instead of remembering some string of 0's and 1's, the programmer would only need to remember short codes like ADD, MOV etc.

e.g., move al, 0x21

The mnemonic means to move the value of the hexadecimal number 21 into a register called al. These mnemonics are in turn converted into machine instructions.

Therefore, the central processing unit can now begin to execute the instructions by fetching them from the memory. The CPU does this continuously till it is shut down.



REMEMBER

The CPU can minimize the number of times that the entire pipeline must wait until a conditional instruction is completed.

Figure 2.4: Basic instruction fetch execute cycle.

Instruction Fetch Execute Cycle

A complete form of the instruction fetch execute cycle can be broken down into the given steps and these are:

- Fetch cycle
- Decode cycle
- Execute cycle
- Interrupt cycle
- Interrupt handler

Fetch Cycle

The fetch cycle begins with retrieving the address stored in the Program Counter (PC). The address stored in the PC is some valid address in the memory holding the instruction to be executed. The CPU completes this step by fetching the instruction stored at this address from the memory and transferring this instruction to a special register instruction register (IR) to hold the instruction to be executed. The program counter is incremented to point to the next address from which the new instruction is to be fetched.

Decode Cycle

The decode cycle is used for interpreting the instruction that was fetched in the fetch cycle. The operands are retrieved from the addresses if the need be.

Execute Cycle

This cycle as the name suggests, simply executes the instruction that was fetched and decoded.

Interrupt Cycle

An interrupt can occur any time during the program execution. Whenever it is caused, a series of events take place so that the instruction fetch execute cycle can again resume after the OS calls the routine to handle the interrupt. Therefore, when an interrupt occurs, the given steps are performed by the OS:

- Suspend the execution of current instruction
- Push the address of current instruction on the system stack
- Loading the PC with the address of the interrupt handler
- This starts the Instruction Fetch Execute cycle again for the instructions in the

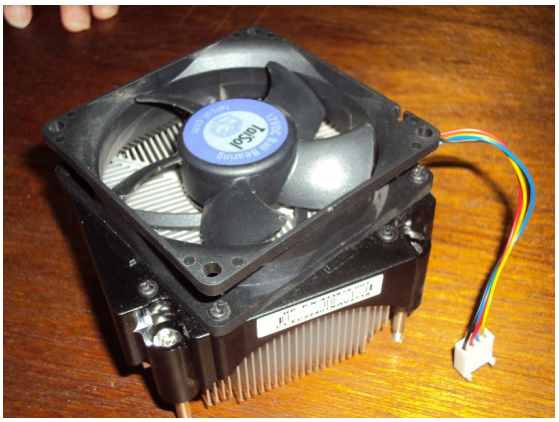
Interrupt Handler

- Set the mode of operation as a privileged one often termed as the Supervisor mode so that the OS can execute the handler.
- Once the OS completes the execution of the interrupt handler, the address of the next instruction to be executed is obtained from popping the value of the address in the stack. The suspended instruction can now continue with its execution.

This cycle of fetching a new instruction, decoding it and finally executing it continues until the computer is turned off.

CASE STUDY

CPU STARVATION



This was a non-data sharing environment in which the total machine CPU time hovered near 100% busy during the production day. The DBAs analyzed the DB2PM Accounting Detail reports to determine whether excessive CPU could be attributed to DB2, to certain applications, to access paths for certain objects, or to something else.

Develop Resource Measurement

For application CPU resource consumption the DBAs developed a "CPU time per SQL statement" measurement. They used the In-DB2 TCB time from the Accounting Detail report as their CPU measure, and created a weighted average measure of the number of SQL statements executed by the application.

Their exact formula is not important here. (*For example*, they researched the average CPU usage required for various DML statements based on the DB2 V7 Performance Topics Redbook, then modified this to suit their applications.) What is important is that they created a generic measurement for application CPU usage that they could then use to find "CPU hog" programs.

Rank Applications by Resource Consumption

For each application they divided the In-DB2 TCB time by the total number of SQL statements executed. This resulted in numbers in the range of 0.1 to 1.0 milliseconds (msec) per statement. Those applications scoring higher than 1.0 msec were considered to be high CPU consumers.

Determine Alternate Resource Availability

The next step was to determine what other resources were available. In this case, real memory on the box where DB2 was running was not 100% utilized. Also, the DBA team reviewed their overall resource profile and determined that they had some unused DASD resources; in addition, they could afford to sacrifice throughput for some low-priority applications.

Develop Resource Tradeoff Options

In consultation with the systems programming staff, the DBAs were able to get an additional gigabyte of real memory dedicated to DB2. This was implemented in the form of several data spaces, and several of the most heavily-used tables were allocated to virtual pools in data spaces.

The result: an overall decrease in CPU utilization, due to fewer required physical I/O to access the critical tables. The tradeoff: additional memory usage.

Next, the DBA team analyzed the accounting detail report looking for DB2 subsystem issues. They noticed that the In-DB2 not accounted for time was relatively high (about five percent of In-DB2 Elapsed Time). This indicated that the DB2 address space was waiting for CPU dispatching. This happens most often when the CPU is extremely busy, as DB2 must wait for a CPU to do some synchronous tasks such as log writes. If DB2 must wait for a CPU, then the application waits for DB2.

The reduction of overall CPU time was called for, as this problem tended to occur during times of high CPU busy. They then implemented the following changes:

- Many small and medium sized tables (under 1,00,000 rows) that were defined with data compression were re-defined to remove the compression option. They also concurrently increased the sizes of some of the virtual pools. This increased DASD usage somewhat and also caused a slight increase in CPU due to the additional I/O required by SQL statements that accessed large numbers of rows. However, there was a net decrease in CPU time by forgoing compression and decompression of rows for those tables.
- Many distributed applications (i.e., those not originating in z/OS) were invoking DB2 stored procedures. In many cases these stored procedures were being used to execute a single SQL statement that returned one row. The overhead for invoking a stored procedure is several thousand machine instructions; so, for some highly-used stored procedures the DBAs had developers use dynamic SQL instead. In addition, they enlarged the dynamic statement caches. This caused a slight increase in CPU time as dynamic SQL needed to be bound (unless it was in the cache); however, there was a net reduction in overall CPU usage as hundreds of thousands of stored procedure invocations were removed.

- The frequency of DB2 system checkpoints was changed from every 5 minutes to every 12 minutes. This reduced CPU usage somewhat, as DB2 did fewer checkpoints. However, this had the effect of increasing recovery time in the event of a system outage.

ROLE MODEL



GEOFFREY W.A. DUMMER PROPOSES THE INTEGRATED CIRCUIT

In a world built on the microchip, it is easy to forget just how revolutionary the idea for an integrated circuit was when it debuted. First proposed by Geoffrey W.A. Dummer in a research paper published on May 7, 1952, the electronic breakthrough has dramatically shifted the human experience. What was once viewed as simply a method for mass producing semiconductors is now used in everything from greeting cards to a child's toy. Alessandro Volta, an Italian physicist, created the first rudimentary electrical circuit in 1800 using a basic salt solution connected with a pair of metal strips. The chemical composition of the different components created the first modern battery, paving the way for German scientist Georg Ohm to experiment with the conductive properties of different substances. By playing with a variety of materials, Ohm learned how to control current by manipulating resistance in a circuit.

Over time, the components which affected electrical movement transistors, resistors, capacitors, diodes had become more sophisticated. By connecting these parts in complex sequences, researchers developed stronger and more efficient means for managing higher voltages while minimizing the amount of heat produced. The proper combination of transistors (designed to start, stop or amplify flow) with resistors (built to slow it) would ensure a capacitor worked on cue say, to power the flash on a modern smartphone camera.

At Bell Labs in mid-December 1947, John Bardeen and Walter Brattain discovered the ability for transistors to multiply the power generated by conductive substances in this case, two gold wires the seeds were planted for a major advance in computing. The result, known as a "bipolar point-contact" transistor, amounted to a massive shift in the capacity of semiconductors, one that later resulted in Bardeen and Brattain sharing the Nobel Prize for Physics in 1956.

Across the Atlantic Ocean, British electrical engineer Dummer was working for the Telecommunications Research Establishment, located at Malvern in western England. Experimenting as part of a talented team of physicists and engineers, he helped to create circuits which were more resistant to the challenges of wet weather along British coastlines in order to help radar installations stay online regardless of the conditions. After working on a variety of cutting edge materials, Dummer became convinced that the ability to etch pathways on semiconductors could lead to a technological leap.

On May 7, 1952, during a session at the United States Electronic Components Symposium, Dummer told the crowd "it now seems possible to envisage electronic equipment in a solid block with no connecting wires." He went on to describe a concept by which the different components that made up a circuit could somehow be sandwiched together by cutting links between layers - from a transistor to a resistor through an insulator, for example. Experts would later point to the discussion as the first mention of the integrated circuit and, thus, the birth of modern computing.

The gap between conception and execution, however, would be a long one. Dummer tried unsuccessfully to construct a circuit based on the idea in 1956. Managing the vast amount of connections required to run a computer without spreading the distance between components too far proved challenging. This "tyranny of numbers" was finally conquered by Jack Kilby at Texas Instruments during the summer of 1958. His revolutionary idea, combining all of the components on a single block of semiconductor material with a metal overlay to connect them, was shown to be effective in tests that September.

Compared to what is used today, Kilby's design was rather inefficient. Six months later, physicist Robert Noyce created a different layout in his office at Fairchild Semiconductor. By using silicon as the semiconductor instead of the then-common germanium and laying a strip of metal on top, then pulling some of it away to form the wires, Noyce developed an inexpensive method for producing the chips. The advances in size and cost opened new doors for the computing industry, but the technology was slow to catch on.

A decade later, Noyce and several others founded a new organization to focus efforts on making the integrated circuit more accessible and, therefore, drive adoption in a variety of businesses. Further research by physicist Federico Faggin revealed the fact self-aligned gates made by overlapping the connectors of a transistor could generate improved processing speeds. Snapped up by Noyce to lead the effort at his new company, Intel, Faggin eventually led the team that announced production of the 4004 microprocessor in 1971. Called a "central processing unit" by the engineers at Intel, the team had made the computer as it is seen today possible.

In the 40-plus years since the debut of the Intel 4004, the integrated circuit has become ubiquitous. The concept proposed by Dummer has, to some, only begun to affect

the way humans live and work. Thanks to the fact similar-sized chips manufactured today have 125 million more transistors than those invented by Kilby and Noyce, a future in which everything from cars to microscopic medical devices are guided by the complex calculations of advanced microchips is a likelihood instead of a mere possibility.

As a key component of the leadership group at Intel, Noyce would go on to hold 16 patents and earn the nickname "Mayor of Silicon Valley" due in no small part to the fact he named the region itself. Kilby managed to do a little better, receiving the National Medal of Science in 1970 and winning the Nobel Prize for Physics in 2000.

What Is The Moral Here?

Geoffrey Dummer was a British electronics engineer and consultant. He was credited as being the first person to conceptualize and build a protoType of the integrated circuit, commonly called the microchip.

SUMMARY

- A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.
- CPU stands for "central processing unit." This is the pretty much the brain of the computer. It processes everything from basic instructions to complex functions.
- The CPU has a program counter that holds the memory address (location) of the next instruction to be executed.
- The ALU performs arithmetic calculations and data manipulation, e.g. comparisons, sorting, combining, etc. The computer's calculator is a part of the CPU known as the arithmetic logic unit.
- The control unit instructs the arithmetic logic unit which arithmetic operations or logical operation is to be performed.
- Random access memory can have instructions READ from it by the CPU and also it can have numbers or other computer data WRITTEN to it by the CPU.
- A stack is a linear array or list of locations. It is sometimes referred to as a pushdown list or last-in-first-out queue.

KNOWLEDGE CHECK

1. **The brain of any computer system is ……..**
   a. Control Unit
   b. Arithmetic Logic Unit
   c. Central Processing Unit
   d. Storage Unit
2. **The central processing unit has a sub-division of ……..**
   a. memory unit and control unit
   b. memory unit and ALU
   c. execution unit and instruction unit
   d. execution unit and memory unit
3. **The units that is used for handling data, and calculate offset address is ……**
   a. memory management unit
   b. execution unit
   c. instruction unit
   d. bus interface unit
4. **The unit that interfaces the internal data bus with the system bus is ……**
   a. bus sizing unit
   b. data buffer
   c. bus control unit
   d. execution unit
5. **The unit that decodes the opcode bytes, received from the 16-byte instruction code queue is ……..**
   a. memory management unit
   b. execution unit
   c. instruction unit
   d. barrel shifter
6. **CPU stands for?**
   a. Control Processing Unit
   b. Central Programming Unit
   c. Central Processing Unit
   d. Control Programming Unit

7. CPU itself has following _____ components

a. 2
b. 3
c. 4
d. 5

8. Which of the following are correct features of CPU?

a. CPU is considered as the brain of the computer
b. CPU performs all types of data processing operations
c. It stores data, intermediate results, and instructions
d. All of the above

9. CPU controls the operation of all parts of the computer.

a. TRUE
b. FALSE
c. Can be true or false
d. Can not say

10. Causing the CPU to step through a series of micro operations is called _____

a. Execution
b. Runtime
c. Pipelining
d. Sequencing

REVIEW QUESTIONS

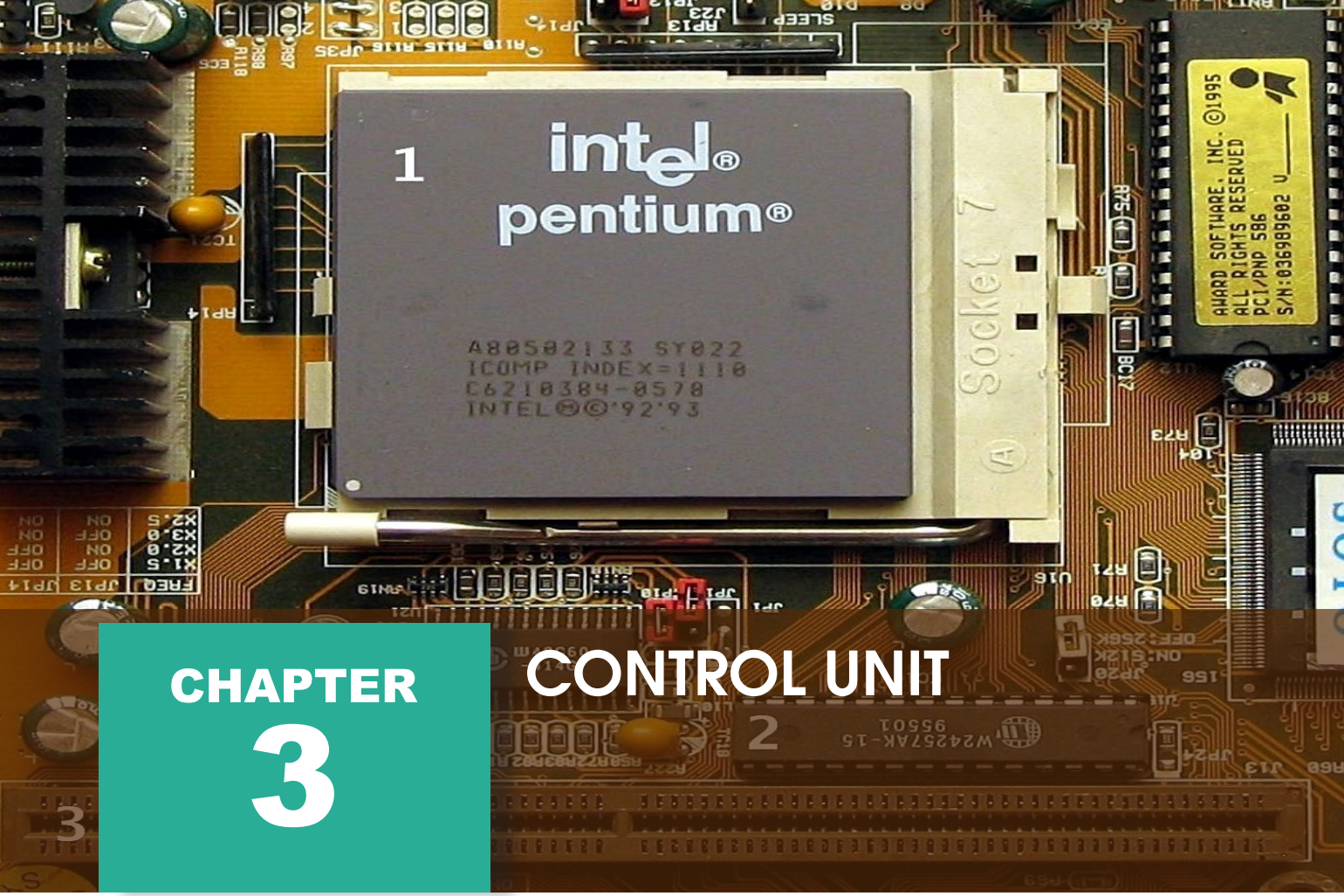
1. Explain the concept of central processing unit.
2. What do you understand by CPU building blocks?
3. Discuss the history of CPU.
4. Describe the addressing modes in brief.
5. What do you understand by displacement addressing mode?

Check Your Result

1. (c)	2. (c)	3. (c)	4. (a)	5. (c)
6. (c)	7. (b)	8. (d)	9. (a)	10. (d)

REFERENCES

1. "The First Generation". Computer History Museum. Retrieved September 29, 2015.
2. "The History of the Integrated Circuit". Nobelprize.org. Retrieved September 29, 2015.
3. Liebowitz, Kusek, Spies, Matt, Christopher, Rynardt (2014). VMware vSphere Performance: Designing CPU, Memory, Storage, and Networking for Performance-Intensive Workloads. Wiley. p. 68. ISBN 978-1-118-00819-5.
4. Regan, Gerard (2008). A Brief History of Computing. p. 66. ISBN 978-1848000834. Retrieved 26 November 2014.
5. Thomas Willhalm; Roman Dementiev; Patrick Fay (December 18, 2014). "Intel Performance Counter Monitor – A better way to measure CPU utilization". software.intel.com. Retrieved February 17, 2015.
6. Turley, Jim. "Motoring with microprocessors". Embedded. Retrieved November 15, 2015.



CHAPTER 3

CONTROL UNIT

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Understand the concept of instruction types
2. Describe hardwire and micro-programmed control

"The basis of computer work is predicated on the idea that only the brain makes decisions and only the index finger does the work. "

—Brian Eno

INTRODUCTION

A control unit or CU is circuitry that directs operations within a computer's processor. It lets the computer's logic unit, memory, and both input and output devices know how to respond to instructions received from a program. Examples of devices that utilize control units include CPUs and GPUs.

A control unit works by receiving input information that it converts into control signals, which are then sent to the central processor. The computer's processor

then tells the attached hardware what operations to carry out. The functions that a control unit performs are dependent on the type of CPU, due to the variance of architecture between different manufacturers.

3.1 INSTRUCTION TYPES

An instruction is fetched and decoded, and then it needs to be executed in order, one after another until completion. In older CPUs, the instruction would have to go through the entire process and finish calculation before the next would begin. To speed up processing, modern CPUs use what are called pipelines, where each step is part of the pipeline. While one instruction is in the execution part of the pipeline, another is already in the decode phase, and another is being fetched. To handle all this, the control unit also needed to perform the role of a **multiplexer**, in that it takes multiple inputs or outputs and directs them into and out of the pipeline.

Keyword

Multiplexer is a device that selects one of several analog or digital input signals and forwards the selected input into a single line.

As computer CPUs continued to advance, much of this changed dramatically. The use of microcode, tiny programs that sit in special, high-speed read-only memory on the CPU, took the place of the old hardwired circuitry. These low-level programs took over the time-consuming job of physically rewiring a control unit and simplified changes to the CPU's architecture. The custom-written micro programs of the control unit, created during the CPU's design phase, are what enable the architecture of a particular type of CPU.

The instructions can be of shorter or longer formats. Shorter instructions take less time for fetching and decoding as compared to longer instructions. So shorter instructions are preferred over longer ones. Also shorter instruction requires only less memory.

The operand field of an instruction may have the following formats given as:

- Zero-Address Instruction: The instruction contains only op code.

Example: ADD

- One-Address Instruction: op code address1

Example: ADD B

- Two-Address Instruction: op code operands

Example: e.g. ADD A, B

- Three-Address Instruction:

Example: ADD R_1, A, B

Instructions may be classified into the following groups:

- Data Transfer Instructions
- Arithmetic Instructions
- Logical Instructions
- Input Output Instruction
- System Control Instructions
- Program-control Instructions

Data transfer Instructions: MOV, LOAD

Arithmetic Instructions: ADD, SUB, DIV, MUL

Logical Instructions: AND, OR, NOT, SHIFT, ROTATE

Input Output Instruction: Memory-mapped I/O (MMIO) and port I/O

Program-control Instructions: Program-control instructions may be classified into the following:

- Unconditional branch
- Conditional branch
- Subroutine call
- Interrupt-handling

The 80386 provides both conditional and unconditional control transfer instructions to direct the flow of execution. Conditional control transfers depend on the results of operations that affect the **flag register**. Unconditional control transfers are always executed.

Keyword

Flag Register is a collection of status flag bits for a processor.

Unconditional Transfer Instructions

JMP, CALL, RET, INT and IRET instructions transfer control from one code segment location to another. These locations can be within the same code segment (near control transfers) or in

different code segments (far control transfers). If the model of memory organization used in a particular 80386 application does not make segments visible to applications programmers, intersegment control transfers will not be used.

Jump Instruction

JMP (Jump) unconditionally transfers control to the target location. JMP is a one-way transfer of execution; it does not save a return address on the stack.

The JMP instruction always performs the same basic function of transferring control from the current location to a new location. Its implementation varies depending on whether the address is specified directly within the instruction or indirectly through a register or memory. A direct JMP instruction includes the destination address as part of the instruction. An indirect JMP instruction obtains the destination address indirectly through a register or a pointer variable.

Direct near JMP

Keyword

Processor is the logic circuitry that responds to and processes the basic instructions that drive a computer.

A direct JMP uses a relative displacement value contained in the instruction. The displacement is signed and the size of the displacement may be a byte, word, or double word. The **processor** forms an effective address by adding this relative displacement to the address contained in EIP. When the additions have been performed, EIP refers to the next instruction to be executed.

Indirect near JMP

Indirect JMP instructions specify an absolute address in one of several ways:

- The program can JMP to a location specified by a general register (any of EAX, EDX, ECX, EBX, EBP, ESI, or EDI). The processor moves this 32-bit value into EIP and resumes execution.
- The processor can obtain the destination address from a memory operand specified in the instruction.

- A register can modify the address of the memory pointer to select a destination address.

Call Instruction

The CALL (Call Procedure) activates an out-of-line procedure, saving on the stack the address of the instruction following the CALL for later use by a RET (Return) instruction. CALL places the current value of EIP on the stack. The RET instruction in the called procedure uses this address to transfer control back to the calling program.

The CALL instructions, like JMP instructions have relative, direct, and indirect versions.

Indirect CALL instructions specify an absolute address in one of these ways:

- The program can CALL a location specified by a general register (any of EAX, EDX, ECX, EBX, EBP, ESI, or EDI). The processor moves this 32-bit value into EIP.
- The processor can obtain the destination address from a memory operand specified in the instruction.

Return and Return-From-Interrupt Instruction

The RET (Return from procedure) terminates the execution of a procedure and transfers control through a back-link on the stack to the program that originally invoked the procedure. RET restores the value of EIP that was saved on the stack by the previous CALL instruction.

RET instructions may optionally specify an immediate operand. By adding this constant to the new top-of-stack pointer, RET effectively removes any arguments that the calling program pushed on the **stack** before the execution of the CALL instruction.

IRET returns control to an interrupted procedure. IRET differs from RET in that it also pops the flags from the stack into the flags register. The flags are stored on the stack by the interrupt mechanism.

Keyword

Stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle.

Conditional Transfer Instructions

The conditional transfer instructions are jumps that may or may not transfer control, depending on the state of the CPU flags when the instruction executes.

Conditional Jump Instructions

Table 3.1 shows the conditional transfer mnemonics and their interpretations. The conditional jumps that are listed as pairs are actually the same instruction. The assembler provides the alternate mnemonics for greater clarity within a program listing.

Conditional jump instructions contain a displacement which is added to the EIP register if the condition is true. The displacement may be a byte, a word, or a double word. The displacement is signed; therefore, it can be used to jump forward or backward.

Table 3.1: Interpretation of conditional transfers

Unsigned Conditional Transfers		
Mnemonic	Condition Tested	"Jump If..."
JA/JNBE	(CF or ZF) = 0	above/not below nor equal
JAE/JNB	CF = 0	above or equal/not below
JB/JNAE	CF = 1	below/not above nor equal
JBE/JNA	(CF or ZF) = 1	below or equal/not above
JC	CF = 1	carry
JE/JZ	ZF = 1	equal/zero
JNC	CF = 0	not carry
JNE/JNZ	ZF = 0	not equal/not zero
JNP/JPO	PF = 0	not parity/parity odd
JP/JPE	PF = 1	parity/parity even
Signed Conditional Transfers		
Mnemonic	Condition Tested	"Jump If..."
JG/JNLE	((SF xor OF) or ZF) = 0	greater/not less nor equal
JGE/JNL	(SF xor OF) = 0	greater or equal/not less
JL/JNGE	(SF xor OF) = 1	less/not greater nor equal
JLE/JNG	((SF xor OF) or ZF) = 1	less or equal/not greater
JNO	OF = 0	not overflow

JNS	SF = 0	not sign (positive, including 0)
JO	OF = 1	overflow
JS	SF = 1	sign (negative)

Loop Instructions

The loop instructions are conditional jumps that use a value placed in ECX to specify the number of repetitions of a software loop. All loop instructions automatically decrement ECX and terminate the loop when ECX=0. Four of the five **loop** instructions specify a condition involving ZF that terminates the loop before ECX reaches zero.

Keyword

Loop is a sequence of instruction s that is continually repeated until a certain condition is reached.

LOOP (Loop While ECX Not Zero) is a conditional transfer that automatically decrements the ECX register before testing ECX for the branch condition. If ECX is non-zero, the program branches to the target label specified in the instruction. The LOOP instruction causes the repetition of a code section until the operation of the LOOP instruction decrements ECX to a value of zero. If LOOP finds ECX=0, control transfers to the instruction immediately following the LOOP instruction. If the value of ECX is initially zero, then the LOOP executes 2^ (32) times.

LOOPE (Loop While Equal) and LOOPZ (Loop While Zero) are synonyms for the same instruction. These instructions automatically decrement the ECX register before testing ECX and ZF for the branch conditions. If ECX is non-zero and ZF=1, the program branches to the target label specified in the instruction. If LOOPE or LOOPZ finds that ECX=0 or ZF=0, control transfers to the instruction immediately following the LOOPE or LOOPZ instruction.

LOOPNE (Loop While Not Equal) and LOOPNZ (Loop While Not Zero) are synonyms for the same instruction. These instructions automatically decrement the ECX register before testing ECX and ZF for the branch conditions. If ECX is non-zero and ZF=0, the program branches to the target label specified in the instruction. If LOOPNE or LOOPNZ finds that ECX=0 or ZF=1, control transfers to the instruction immediately following the LOOPNE or LOOPNZ instruction.

Executing a Loop or Repeat Zero Times

JCXZ (Jump if ECX Zero) branches to the label specified in the instruction if it finds a value of zero in ECX. JCXZ is useful in combination with the LOOP instruction and with the string scan and compare instructions, all of which decrement ECX. Sometimes, it is desirable to design a loop that executes zero times if the count variable in ECX is initialized to zero. Because the LOOP instructions (and repeat prefixes) decrement ECX before they test it, a loop will execute 2^(32) times if the program enters the loop with a zero value in ECX. A programmer may conveniently overcome this problem with JCXZ, which enables the program to branch around the code within the loop if ECX is zero when JCXZ executes. When used with repeated string scan and compare instructions, JCXZ can determine whether the repetitions terminated due to zero in ECX or due to satisfaction of the scan or compare conditions.

Data Movement Instructions

These instructions provide convenient methods for moving bytes, words, or double words of data between memory and the registers of the base architecture.

They fall into the following classes:

- General-purpose data movement instructions.
- Stack manipulation instructions.
- Type-conversion instructions.

General-Purpose Data Movement Instructions

The MOV (Move) transfers a byte, word, or double word from the source operand to the destination operand. The MOV instruction is useful for transferring data along any of these paths. There are also variants of MOV that operate on segment registers.

- To a register from memory
- To memory from a register
- Between general registers
- Immediate data to a register
- Immediate data to a memory

The MOV instruction cannot move from memory to memory or from segment register to segment register are not allowed. Memory-to-memory moves can be performed, however, by the string move instruction MOVS.

The XCHG (Exchange) swaps the contents of two operands. This instruction takes the place of three MOV instructions. It does not require a temporary location to save

the contents of one operand while load the other is being loaded. XCHG is especially useful for implementing semaphores or similar data structures for process synchronization.

The XCHG instruction can swap two byte operands, two word operands, or two double word operands. The operands for the XCHG instruction may be two register operands, or a register operand with a memory operand. When used with a memory operand, XCHG automatically activates the LOCK **signal**.

Keyword

Signal is a function of independent variables that carry some information.

Stack Manipulation Instructions

The PUSH (Push) decrements the stack pointer (ESP), then transfers the source operand to the top of stack indicated by ESP. PUSH is often used to place parameters on the stack before calling a procedure; it is also the basic means of storing temporary variables on the stack. The PUSH instruction operates on memory operands, immediate operands, and register operands (including segment registers).

The PUSHA (Push All Registers) saves the contents of the eight general registers on the stack. This instruction simplifies procedure calls by reducing the number of instructions required to retain the contents of the general registers for use in a procedure. The processor pushes the general registers on the stack in the following order: EAX, ECX, EDX, and EBX, the initial value of ESP before EAX was pushed, EBP, ESI, and EDI. PUSHA is complemented by the POPA instruction.

The POP transfers the word or double word at the current top of stack (indicated by ESP) to the destination operand, and then increments ESP to point to the new top of stack. POP moves information from the stack to a general register, or to memory. There are also a variant of POP that operates on segment registers.

3.1.1 Formats Instruction Cycles and Sub Cycles

The CPU performs a sequence of microoperations for each instruction. The sequence for each instruction of the Basic Computer can be refined into 4 abstract phases:

- Fetch instruction
- Decode
- Fetch operand
- Execute

Program execution can be represented as a top-down design:

Program execution

- Instruction 1
  Fetch instruction
  Decode
  Fetch operand
  Execute
- Instruction 2
  Fetch instruction
  Decode
  Fetch operand
  Execute
- Instruction 3

Keyword

Frequency is the number of occurrences of a repeating event per unit time.

Once a program is in memory it has to be executed. To do this, each instruction must be looked at, decoded and acted upon in turn until the program is completed. This is achieved by the use of what is termed the 'instruction execution cycle', which is the cycle by which each instruction in turn is processed. However, to ensure that the execution proceeds smoothly, it is also necessary to synchronize the activities of the processor.

To keep the events synchronized, the clock located within the CPU control unit is used. This produces regular pulses on the system bus at a specific **frequency**, so that each pulse is an equal time following the last. This clock pulse frequency is linked to the clock speed of the processor - the higher the clock speed, the shorter the time between pulses. Actions only occur when a pulse is detected, so that commands can be kept in time with each other across the whole computer unit.

The instruction execution cycle can be clearly divided into some different parts, which will give as:

Fetch Cycle

The fetch cycle takes the address required from memory, stores it in the instruction register, and moves the program counter on one so that it points to the next instruction.

Decode Cycle

Here, the control unit checks the instruction that is now stored within the instruction register. It determines which opcode and addressing mode have been used, and as such what actions need to be carried out in order to execute the instruction in question.

Execute Cycle

The actual actions which occur during the execute cycle of an instruction depend on both the instruction itself, and the addressing mode specified to be used to access the data that may be required.

Keyword

Machine Codes are set of instructions executed directly by a computer's central processing unit (CPU).

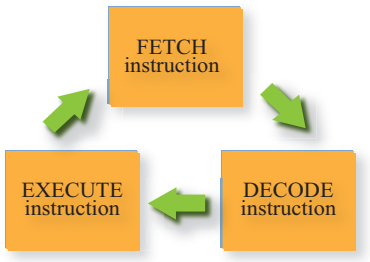


Figure 3.1: Instruction execution cycle.

The first part of the instruction execution cycle is the fetch cycle. The buttons towards the bottom of the animation can be used to control the progress between the steps.

Once the instruction has been fetched and is stored, the next step is to decode the instruction in order to work out what actions should be performed to execute it. This involves examining the opcode to see which of the **machine codes** in the CPU's instruction set it corresponds to, and also checking which addressing mode needs to be used to obtain any required data.

Once the opcode is known, the execution cycle can occur. Different actions need to be carried out dependent on the opcode, with no two opcode requiring the same actions to occur. However, there are generally four groups of different actions that can occur:

- Transfer of data between the CPU and memory.
- Transfer of data between the CPU and an input or output devices.
- Processing of data, possibly involving the use of the arithmetic and logic unit.
- A control operation, in order to change the sequence of subsequent operations. These can possibly be conditional, based on the values stored at that point within the flag register.

Mnemonic	Description
MOV	Moves a data value from one location to another
ADD	Adds to data values using the ALU, and returns the result to the accumulator
STO	Stores the contents of the accumulator in the specified location
END	Marks the end of the program in memory

Immediate Addressing

With immediate addressing, no lookup of data is actually required. The data is located within the operands of the instruction itself, not in a separate memory location. This is the quickest of the addressing modes to execute, but the least flexible.

Direct Addressing

For direct addressing, the operands of the instruction contain the memory address where the data required for execution is stored. For the instruction to be processed the required data must be first fetched from that location.

Indirect Addressing

When using indirect addressing, the operands give a location in memory similarly to direct addressing. However, rather than the data being at this location, there is instead another memory address given where the data actually is located. This is the most flexible of the modes, but also the slowest as two data lookups are required.

Program execution begins with:

PC ← address of first instruction, SC ← 0

After this, the SC is incremented at each clock cycle until an instruction is completed, and then it is cleared to begin the next instruction. This process repeats until a HLT instruction is executed, or until the power is shut off.

Instruction Fetch and Decode

The instruction fetch and decode phases are the same for all instructions, so the control functions and microoperations will be independent of the instruction code.

Everything that happens in this phase is driven entirely by timing variables T_0, T_1 and T_2. Hence, all control inputs in the CPU during fetch and decode is functions of these three variables alone.

T_0: AR ← PC

T_1: IR ← M [AR], PC ← PC + 1

T_2: $D_{0\text{-}7}$← decoded IR (12-14), AR ← IR (0-11), I ← IR (15)

For every timing cycle, one assumes SC ← SC + 1 unless it is stated that SC ← 0.

The operation $D_{0\text{-}7}$ ← decoded IR (12-14) is not a register transfer like most of the microoperations, but is actually an inevitable consequence of loading a value into the IR register. Since the IR outputs 12-14 are directly connected to a decoder, the outputs of that decoder will change as soon as the new values of IR (12-14) propagate through the decoder.

Incrementing the PC at time T_1 assumes that the next instruction is at the next address. If the current instruction is branch instruction. However, performing the increment here will save time if the next instruction immediately follows, and will do no harm if it does not. The incremented PC value is simply overwritten by branch instructions.

In hardware development, unlike serial software development, it is often advantageous to perform work that may not be necessary. Since one can perform multiple microoperations at the same time, one might was well do everything that might be useful at the earliest possible time.

Likewise, loading AR with the address field from IR at T_2 is only useful if the instruction is a memory-reference instruction. One would not know this until T_3, but there is no reason to wait since there is no harm in loading AR immediately.

3.1.2 Micro-Operations

A micro operation is an elementary operation performed with data stored in register.

They are classified into:

- Register transfer micro operation
- Arithmetic micro operation
- Logic micro operation
- Shift micro operation

Basic arithmetic micro operations are:

- Addition
- Subtraction
- Increment
- Decrement
- Arithmetic Shift

Execution of an instruction has a number of smaller units

- Fetch, indirect, execute, interrupt, etc.
- Each part of the cycle has a number of smaller steps called micro-operations
- Discussed extensive in pipelining
- Micro-ops are the fundamental or atomic operations of the processor

Short look on different arithmetic micro operations:

The Add micro operation is specified as:

$$R3 \leftarrow R1 + R2.$$

And it means add content of R1 to R2 and store result of addition in R3. Usually it is implemented using hardware full adders.

The Sub addition is usually implemented using complementation and addition

$$R3 \leftarrow R1 + R2 + 1\ (R1 - R2)$$

And it means subtract R2 from R1 by adding the complement of R2 plus 1 to R1.

Usually it is implemented using a full adder a complement circuit.

Nevertheless, Increment and decrement are implemented using Up and Down Counter. And finally multiplication and division are implemented using sequence of additions and subtractions respectively.

Logic Micro Operation

Logic micro operation specifies binary operations on the strings of bits in registers. Logic micro operations are bit-wise operations, i.e., they work on the individual bits of data.

Those could be useful for bit manipulations on binary data and also useful for making logical decisions based on the bit value. There are, in principle, 16 different logic functions that can be defined over two binary input variables. However, most systems only implement four of these:

AND (^), OR (˘), XOR (Å), Complement/NOT

The others can be created from combination of these. List of Logic Microoperations-16 different logic operations with 2 binary variables are shown as:

00 11 01 01	Boolean Function	Micro-Operations	Name
00 00	F0 = 0	F ← 0	Clear
00 01	F1 = xy	F ← A^B	AND
00 10	F2 = xy'	F ← A ^ B'	
00 11	F3 = x	F 4 ← A	Transfer A
01 00	F4 = x'y	F ← A'^ B	
01 01	F5 = y	F ← B	Transfer B
01 10	F6 = x Å *y*	F ← A ÅB	Exclusive-OR
01 11	F7 = x + y	F ← A ˘ B	OR
10 00	F8 = (x + *y*)'	F← (A ˘ B)'	NOR
10 01	F9 = (x Å *y*)'	F ← (A ÅB)'	Exclusive-NOR
10 10	F10 = *y*'	F ← B'	Complement B
10 11	F11 = x + *y*'	F ←A ˘ B	
11 00	F12 = x'	F ← A'	Complement A
11 01	F13 = x' + *y*	F← A'˘ B	
11 10	F14 = (xy)'	F← (A ^ B)	NAND
11 11	F15 = 1	F← all 1's	Set to all l's

The hardware implementation of logic micro operation requires the insertion of the most important gates like AND, OR, EXOR, and NOT for each bit or pair of bits in the registers.

One stage of a circuit that generates the four basic logic micro operations. It consists of four gates and a multiplexer. The two selection lines of the multiplexer selects one

of the four logic operations available at one time. The circuit shows one stage for bit "i" but for logic circuit of n bits the circuit should be repeated n times but with one remark; the selection pins will be shared with all stages.

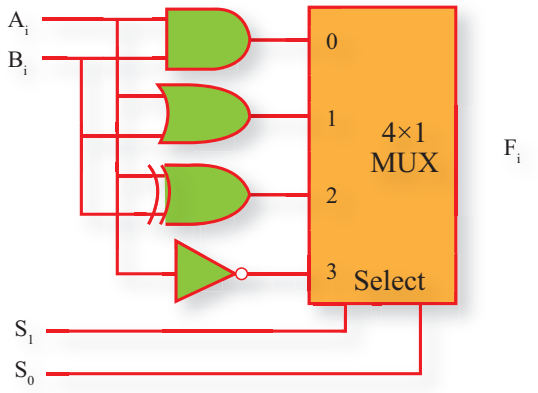


Figure 3.2: Four basic logic micro operations.

DID YOU KNOW ?

The idea of microprogramming was introduced by Maurice Wilkes in 1951 as an intermediate level to execute computer program instructions.

3.1.3 Execution of a Complete Instruction

The program counter, as one of several different registers built into the CPU, performs the task of receiving each of the instructions in task sequence. Maintaining the logical sequence simplifies the forward progression to each step, ultimately completing the task. This logical progression is maintained by pointing to the data that is to be used next even as the previous data is in use, and then quickly pointing to the next data in the sequence as each instruction is executed in turn.

Depending on the configuration of the program counter, the register may actually serve as the pointer for more than one instruction involved with a given process currently under execution. With some register designs, the program counter not only points toward the process that is next in the sequence, but also toward the just released instruction that is now in the process of being implemented. This particular design is thought to aid in maintaining the balance of resources as well as helping to protect the forward movement of the instructions with a higher level of efficiency.

The function of a program counter is essential to the successful execution of the instructions involved in any task. By pointing the way toward each instruction in the sequence, the counter helps to provide a logical execution of those steps that ultimately lead to completing the task quickly and efficiently. Without the use of this type of register, the potential for the sequence to be disrupted would be enhanced, and the chances for completing the task would be relatively few. By adding organization to the incremental step by step completion of all instructions, the program counter serves as the conduit that minimizes anomalies and keeps the process moving forward properly.

Also known as an instruction address register, PC, or instruction pointer, a program counter is a type of register that is found in the central processing unit (CPU) of a computer system. The purpose of the counter is to provide the temporary housing for the next instruction that is to be executed in a string of instructions. As one instruction is retrieved and implemented, the program counter queues up the next instruction in the string, effectively minimizing delays in the execution of steps necessary to complete a task. By always pointing at the next instruction, the process is kept moving forward efficiently. While there are variations, the operation of a program counter within the central processing unit is relatively straightforward. When a new task is ordered, the instructions necessary to manage that task retrieved from the memory. This is normally accomplished by the CPU creating an address that is then sent to the memory, which responds by sending the instructions back to the unit as data via a data bus.

3.2 HARDWIRE AND MICRO-PROGRAMMED CONTROL

It should be mentioned that most computers today are micro-programmed. The reason is basically one of flexibility. Once the control unit of a hard-wired computer is designed and built, it is virtually impossible to alter its architecture and instruction set. A micro-programmed computer, however, one can change the computer's instruction set simply by altering the microprogram stored in its control memory. Therefore, one could add seven more instructions to the instruction set by simply expanding its microprogram. To do this with the hard-wired version of the computer would require a complete redesign of the controller circuit hardware.

Another advantage to using micro-programmed control is the fact that the task of designing the computer in the first place is simplified. The process of specifying the architecture and instruction set is now one of software (micro-programming) as opposed to hardware design. Nevertheless, for certain applications hard-wired computers are still used. If speed is a consideration, hard-wiring may be required since it is faster to have the hardware issue the required control signals than to have a "program" do it.

Control word (CW)

A sequence of Nsig bits, where Nsig is the total number of control signals; each bit in a CW corresponds to one control signal.

Each control step during execution of an instruction defines a certain CW; it represents a combination of 1s and 0s corresponding to the active and non-active control signals.

Micro-routine

A sequence of CWs corresponding to the control sequence of a machine instruction. An individual CW in a micro-routine is called a microinstruction. Micro-programmed control basic idea:

- All micro routines corresponding to the machine instructions are stored in the control store.
- The control unit generates the sequence of control signals for a certain machine instruction by reading from the control store the CWs of the micro-routine corresponding to the respective instruction.
- The control unit is implemented just like another very simple CPU, inside the CPU, executing micro-routines stored in the control store.

3.2.1 Micro Program Sequencing

The controller causes instructions to be executed by issuing a specific set of control signals at each beat of the system clock. Each set of control signals issued causes one basic operation, such as a register transfer, to occur within the data path section of the computer. A hard-wired control unit the control matrix is responsible for sending out the required sequence of signals.

An alternative way of generating the control signals is that of micro-programmed control. In order to understand this method it is convenient to think of sets of control signals that cause specific micro-operations to occur as being "microinstructions" that could be stored in a memory. Each bit of a microinstruction might correspond to one control signal. If the bit is set it means that the control signal will be active; if cleared the signal will be inactive. Sequences of microinstructions could be stored in an internal "control" memory. Execution of a machine language instruction could then be caused by fetching the proper sequence of microinstructions from the control memory and sending them out to the data path section of the computer. A sequence of microinstructions that implements an instruction on the external computer is known as a micro-routine. The instruction set of the computer is thus determined by the set of

micro-routines, the "microprogram," stored in the controller's memory. The control unit of a microprogram-controlled computer is essentially a computer within a computer.

A block diagram of a micro-programmed control unit that may be used to implement the instruction set of the computer. The heart of the controller is the control 32 X 24 ROM memory in which up to 32 24-bit long microinstructions can be stored. Each is composed of two main fields: a 16-bit wide control signal field and an 8-bit wide next-address field. Each bit in the control signal field corresponds to one of the control signals. The next-address field contains bits that determine the address of the next microinstruction to be fetched from the control ROM. Words selected from the control ROM feed the microinstruction register. This 24-bit wide register is analogous to the outer machine's instruction register. Specifically, the leading 16 bits (the control-signal field) of the microinstruction register are connected to the control-signal lines that go to the various components of the external machine's data path section.

Addresses provided to the control ROM come from a micro-counter register, which is analogous to the external machine's program counter. The micro-counter, in turn, receives its input from a multiplexer which selects from:

- The output of an address ROM,
- A current-address incremented, or
- The address stored in the next-address field of the current microinstruction.

The controller's address ROM is fed by the outer computer's instruction register. The address ROM maps the op-code of the instruction currently contained in the op-code field of the instruction register to the starting address of the corresponding microroutine in the control ROM. Address zero of the address ROM contains the control-ROM address of the fetch routine; each other addresses in the address-ROM corresponds to one of the op-codes of the computer's instruction set. Table 3.2 shows the contents of the address ROM for the instruction set of the simple computer. To see how the address ROM works, let us assume that an ADD instruction has been fetched into the outer computer's instruction register. Since the op-code of the ADD instruction is 3, the number stored at location 3 of the address ROM is the starting address in the control ROM of the microroutine that implements the ADD instruction.

The 5-bit CRJA (Control ROM Jump Address) sub-field holds a microinstruction address. Thus, the address of the next microinstruction may be obtained from the current microinstruction. This permits branching to other sections within the microprogram. The combination of the MAP bit, the CD (condition) bit, and the negative flag from the accumulator of the external machine provide input to the logic that feeds the select lines of the multiplexer and thereby determine how the address of the next microinstruction will be obtained.

Table 3.2: The micro programmed basic computer's address ROM

Instruction Mnemonic	Address-ROM Address (Instruction Op-Code)	Address-ROM Contents (Control-ROM Micro-Routine Start Address)
"Fetch"	0	00
LDA	1	03
STA	2	06
ADD	3	09
SUB	4	0B
MBA	5	0D
JMP	6	0E
JN	7	0F
Available for	8-E	12-1E
New Instructions HLT	F	1F

If the MAP bit is one, the logic attached to the multiplexer's select lines produces a 01 which selects the address ROM. Therefore, the address of the micro-routine corresponding to the instruction in the outer machine's instruction register will be channeled to the control ROM. It should be clear that the MAP bit must be set in the last microinstruction of the "fetch" micro-routine, since it is at that moment that one wants the newly-fetched instruction to be executed.If the MAP bit is zero and the CD bit is zero, (unconditional branch), the multiplexer logic produces a 10, which selects the CRJA field of the current instruction. Therefore, the next instruction will come from the address contained in the current instruction's next-address field. With MAP=0 and CD=1 (conditional branch), the logic that feeds the multiplexer will produce either a 00 or a 10, depending on the value of the negative flag. If the flag is set, it is a 10, which selects the jump address contained in the current microinstruction. If the negative flag is cleared, the select lines to the multiplexer receive a 00, which causes the incrementer to be selected. The next microinstruction will come from the next address in sequence. It should be noticed that with this scheme, if one is not doing branching, the CRJA field should contain the address of the next microinstruction and the CD bit should be cleared. This will cause "branch to the next microinstruction" to occur.

The HLT bit is used to terminate execution. If it is set, the clock that synchronizes activities within the entire machine is stopped.

The micro-counter is triggered by a rising clock edge and the microinstruction register by a falling edge. Thus, one sees that on each positive edge, the micro-counter receives the address of the microinstruction and presents it to the control

ROM, which has until the next negative edge to output the addressed control word to the microinstruction register. Since all operation in the data path section are positive-edge triggered, there is adequate time for the signals specified in the control word contained in the microinstruction register to go out to all sections of the external machine. The sequence of latching the address of microinstruction i+1 into the micro-counter while microinstruction i executes (positive edge) and then presenting the control word of microinstruction i+1 to the microinstruction register (negative edge) continues until a set HLT bit stops the clock. The "fetch" micro-routine occupies control ROM addresses 0, 1, and 2. The active EP and LM control-signal bits in its first microinstruction cause a register transfer from the program counter to the memory address register to occur. The MAR will now contain the address in RAM of the next instruction. Since CD and MAP are both zero (unconditional branch), the next microinstruction will come from the address stored in the CRJA field (01) the next consecutive location. The microinstruction stored at that location has only the R bit active. Thus, the word stored in the memory location being accessed by the MAR will be gated to the Memory Data Register (MDR). The zeroes in CD and MAP again cause the microinstruction to be fetched from the address specified in the CRJA field, (02). Active control signal bits for that microinstruction are ED, LI, and IP. The first two transfer the word in the MDR to the Instruction Register and the last increments the program counter. The new instruction is safely in the IR, and the PC is pointing to the next instruction in sequence. One has completed an instruction fetch. Since the MAP field in the last microinstruction of this "fetch" micro-routine is equal to 1, the address of the next microinstruction is determined by the address ROM, which, in turn, depends upon the opcode of the instruction that has just been loaded into the instruction register.

REMEMBER

The one exception to this rule is the case of the last microinstruction within a micro-routine. Normally one would then want to branch back to the "fetch" micro-routine. Since this routine starts at control-ROM location 00000, that address should be contained in the CRJA field and CD should be 0.

When the JN instruction is executed, control is supposed to be transferred to the address specified by the least significant eight bits of the number contained in the instruction register if the negative flag is set. If the negative flag is not set, execution should continue with the next instruction in sequence. Let one sees how the micro-routine stored at control-ROM locations 0F, 10, and 11 implement this conditional jump. In the first microinstruction, none of the control signal bits is set. Thus,

nothing will occur in the data path section of the computer. However, the fact that the CD bit is set means that if the negative flag is set, the next microinstruction will be fetched from the control-ROM address specified in the CRJA field. The microinstruction stored at that location has the EI and LP control signal bits set. Thus, the contents of the instruction register will be transferred to the program counter. The zeroes stored in the CD and MAP bits because the next microinstruction to be fetched from the address contained in the CRJA field. This is the start of the "fetch" micro-routine. Thus one sees that if the negative flag is set, the JN micro-routine places the jump address in the program counter and transfer to the fetch routine. When that fetch is performed, control will have been transferred to the jump address. If, on the other hand, the negative flag is NOT SET when the JN micro-routine executes, then the set CD bit in its first microinstruction causes the current address stored in the micro-counter to be incremented. Thus, the next microinstruction would be fetched from location 10. That microinstruction also has no active control signals bits, but with CD=0 and CRJA=00, the next microinstruction will be the first one in the "fetch" routine. The JN instruction simply returns us to the next fetch. Since the program counter has not been altered, that fetch will be from the next sequential memory location, as usual.

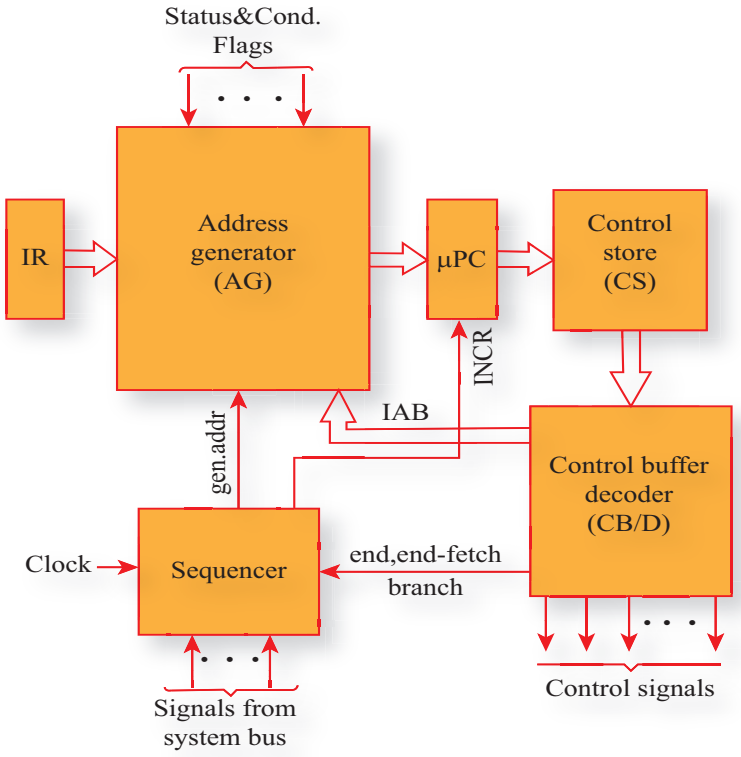


Figure 3.3: Micro program sequencing.

3.2.2 Wide Branch Addressing

Generating branch addresses becomes more difficult as the number of branches increases. In such situations programmable logic array can be used to generate the required branch addresses. This simple and inexpensive way of generating branch addresses is known as wide-branch addressing. Here, the opcode of a machine instruction is translated into the starting address of the corresponding micro-routine. This is achieved by connecting the opcode bits of the instruction register as inputs to the PLA, which acts as a decoder. The output of the PLA is the address of the desired micro routine.

The control unit is responsible for initiating the sequence of micro operations that comprise instructions.

- When these control signals are generated by hardware, the control unit is hardwired.
- When these control signals originate in data stored in a special unit and constitute a program on the small scale, the control unit is micro programmed.

The control function specifying a micro operation is a binary variable whose active state could be either 1 or 0.

- In the variable's active state, the micro operation is executed.
- The string of control variables which control the sequence of micro operations is called a control word.

The micro operations specified in a control word are called a microinstruction.

- Each microinstruction specifies one or more micro operations that are performed.

The control unit coordinates stores microinstruction in its own memory (usually ROM) and performed the necessary steps to execute the sequences of microinstructions (called micro programs).

3.2.3 Microinstruction with Next Address Field

One of the most striking features of the micro routine is the relatively large number of branch microinstructions. These microinstructions constitute one-third of the total and present a serious constraint on the operating speed of the computer. The situation can become significantly worse when other micro routines are considered. The increase in branch microinstructions stems partly from limitations in the ability to assign successive addresses to all microinstructions that are generally executed in consecutive order.

The wide branching with address modification, some addresses have to be assigned in a rather inflexible fashion, and this frequently leaves address spaces that are difficult to use efficiently.

This situation prompts a reevaluation of the sequencing technique built around an increment able μPC. As an alternative, one may consider a scheme in which each microinstruction includes some branching capability. The simplest approach is to include an address field as a part of every microinstruction to indicate the location of the next microinstruction to be fetched. This means, in effect, that every microinstruction becomes a branch microinstruction, in addition to its other functions.

The major disadvantage of this approach is the need for additional bits for the address field. Its severity can be assessed as follows. In a typical computer, it is possible to design an adequate micro program that governs most control functions with fewer than 4K microinstructions, employing perhaps 50 to 80 bits per microinstruction. This implies that an address field of 12 bits is required. Therefore, approximately one-sixth of the control store capacity must be devoted to addressing purposes. Even if more extensive micro programs are needed, the address field will become only slightly larger.

The most obvious advantage of this approach is the virtual elimination of separate branch microinstructions. Furthermore, there are few limitations in assignment of addresses to microinstructions. These advantages outweigh the negative attributes and make the scheme attractive from the practical point of view. In fact, versions of this approach are adopted in most commercially available computers that use micro programmed control. This approach eliminates the need for an increment able μPC. A simple microinstruction address register (μAR) can hold the address of the microinstruction to be fetched. The next-address bits are fed through the OR gates to the μAR. The address may be modified on the basis of the data in the IR, status flags, and condition codes. The decoding circuits include a PLA decoder that is used to generate the starting address of a given micro routine on the basis of the OP code in the IR.

REMEMBER

An initial address is loaded into the CAR when power is turned on; this is usually the address of the first microinstruction in the instruction fetch routine. Next, the control unit must determine the effective address of the instruction.

In a micro programmed processor, the control unit consists of:

- *Control Address Register*: contains the address of the next microinstruction to be executed.

- *Control Data Register*: contains the microinstruction to be executed.
- *The Sequencer*: determines the next address from within control memory
- *Control Memory*: where microinstructions are stored.

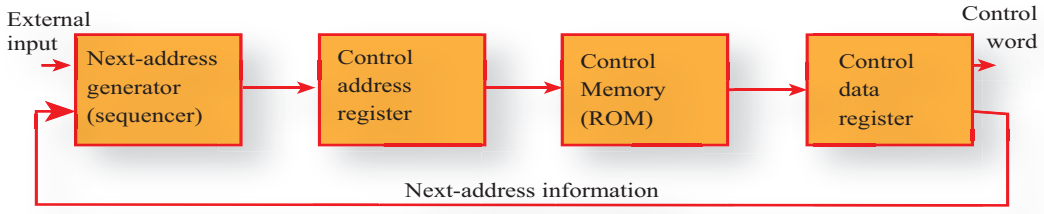


Figure 3.4: Micro programmed control organization.

Sequencer

The sequencer generates a new address by:

- Incrementing the CAR
- Loading the CAR with an address from control memory
- Transferring an external address
- Loading an initial address to start the control operations

Address Sequencing

Microinstructions are usually stored in groups where each group specifies a routine, where each routine specifies how to carry out an instruction. Each routine must be able to branch to the next routine in the sequence.

Mapping

The next step is to generate the micro operations that executed the instruction. This involves taking the instruction's opcode and transforming it into an address for the instruction's micro program in control memory. This process is called mapping.

While microinstruction sequences are usually determined by incrementing the CAR, this is not always the case. If the processor's control unit can support subroutines in a micro program, it will need an external register for storing return addresses.

When instruction execution is finished, control must be return to the fetch routine. This is done using an unconditional branch.

Addressing sequencing capabilities of control memory include:

- Incrementing the CAR
- Unconditional and conditional branching (depending on status bit).
- Mapping instruction bits into control memory addresses
- Handling subroutine calls and returns.

DID YOU KNOW ?

In 1967 Fairchild Semiconductor introduced an IC that contained all of the ALU functions, but required additional circuitry to provide register storage and data control.

3.2.4 Pre-fetching Microinstructions

The major drawback of micro programmed control is the inherently slower operating speed of the computer. Fetching a microinstruction from the control store takes considerably longer than generating equivalent control signals using hardwired circuits.

It is obvious that micro programmed machines will never be able to match the speed of hardwired ones. Having made a fundamental decision tQ use microprogramming, the designer may still desire to produce as fast a machine as possible; probably by choosing the fastest available control store, and using long microinstructions to simultaneously generate as many control signals as possible. But there is another option available. Instead of following the natural sequence of fetching a microinstruction, executing it, and then fetching the next one, it is possible to save time by prefetching the next microinstruction while the current one is being executed. In this way, most of the execution time can be overlapped with the fetch time. Thus the overall machine speed is essentially' dependent upon the fetch times. Such techniques of overlapping operations to keep the hardware units as busy as possible are often referred to as "pipelining."

REMEMBER

Depending on the current microinstruction condition flags, and the contents of the instruction register, a control memory address must be generated for the next micro instruction.

Prefetching microinstructions presents some organizational difficulties. It is sometimes necessary to use the status flags and the results of the currently executed microinstruction to determine the address of the next microinstruction. Thus, straightforward prefetching will occasionally pre-fetch a wrong microinstruction. It is necessary to repeat the fetch with the correct address, which requires more complex hardware. However, the disadvantages are minor and the prefetching technique is often found in practice.

3.2.5 Concept of Horizontal and Vertical Microprogramming

The horizontal microprogramming is also called hardware microprogramming. The length of the microinstructions is high, and they control many hardware resources operating in parallel. To each resource of the computer a field or bit is assigned, by which the resource is controlled. The meanings of the microinstruction fields do not depend on the meanings of other fields. None of the fields can have two or more different meanings for identical values contained by that field.

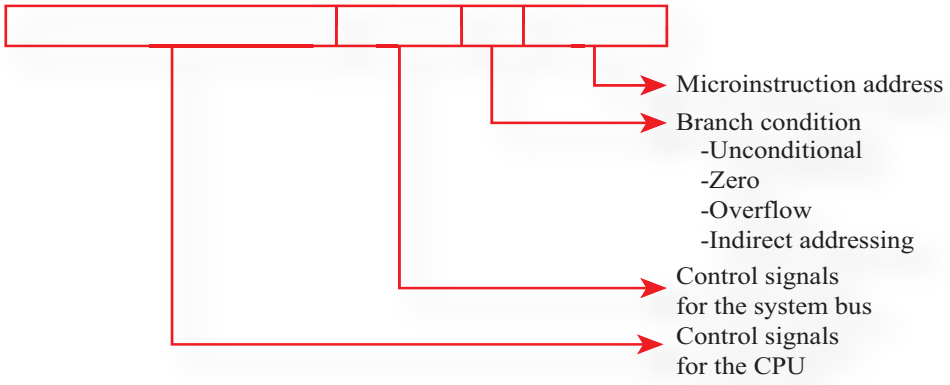


Figure 3.5: General format of a horizontal microinstruction.

In horizontal microprogramming, a microinstruction contains the following main fields:

- One or more control-signal fields, which activate the control signals of the circuits;
- The address field, which contains the address of the next microinstruction that will be executed if a certain condition is true. If this condition is false, the next microinstruction from the control memory will be executed.
- The condition field, which selects the next microinstruction that will be executed.

There is one bit for each internal CPU control signal, and one bit for each system bus control signal. The condition field indicates the condition in which a branch is executed in the micro program, and the address field contains the address of the next microinstruction to be executed when a branch is taken.

Such a microinstruction is interpreted as follows:

- To execute the microinstruction, all the control signals indicated by a bit of 1 in the control signal field are activated, and all the control signals indicated

by a bit of 0 are deactivated. The activated control signals will determine the execution of one or more micro-operations.

- If the condition indicated by the condition field is false, the next microinstruction in the micro program is executed.
- If the condition indicated by the condition field is true, the microinstruction indicated by the address field is executed.

The disadvantage of horizontal microprogramming is that the bits of the micro program word cannot be used efficiently. Not all microinstructions allocate useful meanings to all the available fields.

A micro program word with fewer bits can be designed, so that the meanings of the bits will depend on the particular microinstruction. At one extreme, a micro program word with only two fields can be obtained: the first field contains a code which indicates the meaning of the second field. This type of microprogramming is called vertical microprogramming. All the resources of the computer are controlled by the same field at different times, indicated by the code field. The structure of a computer using vertical microprogramming may be simple, but the speed of execution reduces proportionally with the number of distinct meanings of the code field in the microinstruction.

The vertical microinstructions execute simple operations, such as: load, store, add, branch. These types of microinstructions are similar to the machine-language instructions, which contain an opcode and one or more operands; this is the reason why vertical microprogramming is also called software microprogramming.

In practice, diagonal microinstructions are used, which combine the characteristics of horizontal and vertical microinstructions: they have an enhanced capability to simultaneously control the hardware resources of the computer, and, in the same time, they are easy to understand and to implement.

The length of a microinstruction depends not only on its type (horizontal or vertical), but also on the encoding degree. The simplest microinstruction is not encoded at all, so that a control signal is assigned to each bit. This solution offers the maximum flexibility, because changing a microprogram only requires to change the contents of the control memory. The encoding on groups implies to combine in different fields the bits that control the mutually exclusive resources of the computer. To each field a decoder is attached. At the other extreme, the microinstruction could be completely encoded. This solution requires a minimum number of bits for the microinstruction word, but the flexibility offered is also minimal.

CASE STUDY

NETWORKING SOLUTION FOR CNC CONTROL SYSTEMS



The initial solution was for the engineers to use CNC (Computer Numerical Control) to run a new manufacturing program. This solution requires off-line development in the design room, and then manual transfer of the program to the factory floor PC using a floppy disk or USB Storage. The factory floor PC will implement the program by sending instructions to various manufacturing machines over an RS-232 interface connection. However, the environment of the factory floor is not suitable for PCs (due to high humidity, oily and dusty surroundings, and extreme temperatures); it is highly probably that PCs in such environments will crash frequently.

Moxa Solution

Moxa's NPort 5110 and NPort W2150 Plus device servers were chosen since they meet all of the requirements. The NPort is perfectly suited for rugged environments and is much smaller than a PC.

The control center sends instructions to the CNC in TCP/IP packet format through an Ethernet/Wireless connection to the NPort. It is no longer necessary for personnel to physically transport the program to the site, and multiple CNC machines can receive the data simultaneously.

Advantages of implementing this solution:

- System costs are reduced by lower equipment expenses (the cost of NPort is much lower than a PC) and by the time and human resources saved.
- Changing the CNC layout when expanding is quick and easy.

- Moxa's driver technology reroutes COM data through the computer's Ethernet card; users do not need to spend time to make changes to their existing manufacturing programs.
- Centralized feedback of the CNC's status ensures organized and simplified maintenance.
- Wireless NPort saves on cabling costs and time.

Why Moxa

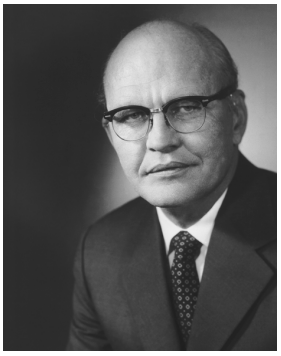
NPort 5110

- Small size for easy installation
- Reliable serial-to-Ethernet solution for data transmission
- Monitoring and troubleshooting utilities are included

NPort W2150 Plus

- 802.11a/b/g wireless connectivity to a CNC
- Seamless roaming with a user-definable signal strength threshold
- Off-line port buffering and serial data log
- Secure data access with WEP, WPA, WPA2, and secure remote management with SSH/SSL configurations

ROLE MODEL



JACK KILBY: ONE OF THE GREATEST ACHIEVER AND NOBEL PRIZE WINNER

Jack St. Clair Kilby was an American electrical engineer who took part (along with Robert Noyce) in the realization of the first integrated circuit while working at Texas Instruments (TI) in 1958. He was awarded the Nobel Prize in physics on December 10, 2000. To congratulate him, US President Bill Clinton wrote, "You can take pride in the knowledge that your work will help to improve lives for generations to come." He is also the inventor of the handheld calculator and the thermal printer, for which he has patents. He also has patents for seven other inventions.

In 1958, he joined TI in Dallas. During the summer of that year working with borrowed and improvised equipment, he conceived and built the first electronic circuit in which all of the components, both active and passive, were fabricated in a single piece of semiconductor material half the size of a paper clip. The successful laboratory demonstration of that first simple microchip on September 12, 1958, made history.

Jack Kilby went on to pioneer military, industrial, and commercial applications of microchip technology. He headed teams that built both the first military system and the first computer incorporating integrated circuits. He later co-invented both the hand-held calculator and the thermal printer that was used in portable data terminals.

In 1970, he took a leave of absence from TI to work as an independent inventor. He explored, among other subjects, the use of silicon technology for generating electrical power from sunlight. From 1978 to 1984, he held the position of Distinguished Professor of Electrical Engineering at Texas A&M University.

Mr. Kilby officially retired from TI in 1983, but he maintained a significant involvement with the company throughout his life. Jack Kilby was the recipient of two of the nation's most prestigious honors in science and engineering. In 1970, in a White House ceremony, he received the National Medal of Science. In 1982, he was inducted into the National Inventors Hall of Fame,

taking his place alongside Henry Ford, Thomas Edison, and the Wright Brothers in the annals of American innovation.

Mr. Kilby held over 60 U.S. patents. He was a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) and a member of the National Academy of Engineering (NAE). He was awarded the Franklin Institute's Stuart Ballantine Medal, the NAE's Vladimir Zworykin Award, the American Society of Mechanical Engineers' Holley Medal, the IEEE's Medal of Honor, the Charles Stark Draper Prize administered by the NAE, the Cledo Brunetti Award, and the David Sarnoff Award. On the 30th anniversary of the invention of the integrated circuit, the Governor of Texas dedicated an official Texas historical marker near the site of the TI laboratory where Mr. Kilby did his work.

In 2000, Jack Kilby was awarded the Nobel Prize in Physics for his part in the invention of the integrated circuit. Mr. Kilby passed away on June 20, 2005, in Dallas following a brief battle with cancer. Yet, his legacy lives on. From Jack Kilby's first simple circuit has grown a worldwide integrated circuit market whose sales in 2007 totaled $219 billion. These components supported a 2007 worldwide electronic end-equipment market of nearly $1,500 billion. Such is the power of one idea to change the world.

What Is The Moral Here?

Kilby was known as the son of an electrical engineer and, like many inventors of his era, got his start in electronics with amateur radio. He has been awarded more than sixty patents for his inventions. Using the integrated circuit Jack Kilby designed and co-invented the first pocket-sized calculator called the "Pocketronic". He also invented the thermal printer that was used in portable data terminals. For many years Kilby was involved in the invention of solar powered devices.

SUMMARY

- A control unit or CU is circuitry that directs operations within a computer's processor. It lets the computer's logic unit, memory, and both input and output devices know how to respond to instructions received from a program.
- The JMP instruction always performs the same basic function of transferring control from the current location to a new location.
- RET instructions may optionally specify an immediate operand. By adding this constant to the new top-of-stack pointer, RET effectively removes any arguments that the calling program pushed on the stack before the execution of the CALL instruction.
- The loop instructions are conditional jumps that use a value placed in ECX to specify the number of repetitions of a software loop.
- A sequence of Nsig bits, where Nsig is the total number of control signals; each bit in a CW corresponds to one control signal.
- The controller causes instructions to be executed by issuing a specific set of control signals at each beat of the system clock.

KNOWLEDGE CHECK

1. **The register that keeps track of the instructions in the program stored in memory is ……..**
   a. control register
   b. program counter
   c. status register
   d. direct register
2. **…….. is a small very speed register file maintained by the instruction fetch segment of the pipeline.**
   a. Branch target buffer
   b. Loop buffer
   c. Branch loop buffer
   d. Target register
3. **The instruction that cause transfer of data from one location to another without changing the binary information content are ……..**
   a. data transfer instruction
   b. data manipulation instruction
   c. register transfer instruction
   d. program control instruction
4. **The branch logic that provides decision making capabilities in the control unit is known as ……..**
   a. controlled transfer
   b. conditional transfer
   c. unconditional transfer
   d. None of above
5. **CISC stands for ……..**
   a. Common Instruction Set Computers
   b. Complex Instruction Set Compilers
   c. Complex Instruction Set Computers
   d. Compound Instruction Set Computers
6. **If the control signals are generated by combinational logic, then they are generated by a type of _____ controlled unit.**
   a. Micro programmed
   b. Software

c. Logic
d. Hardwired

7. Which is the simplest method of implementing hardwired control unit?

a. State Table Method
b. Delay Element Method
c. Sequence Counter Method
d. Using Circuits

8. A set of microinstructions for a single machine instruction is called _____

a. Program
b. Command
c. Micro program
d. Micro command

9. Micro-program consists of a set of microinstructions which are strings of 0s and 1s.

a. True
b. False

10. A decoder is required in case of a _____

a. Vertical Microinstruction
b. Horizontal Microinstruction
c. Multilevel Microinstruction
d. All types of microinstructions

REVIEW QUESTIONS

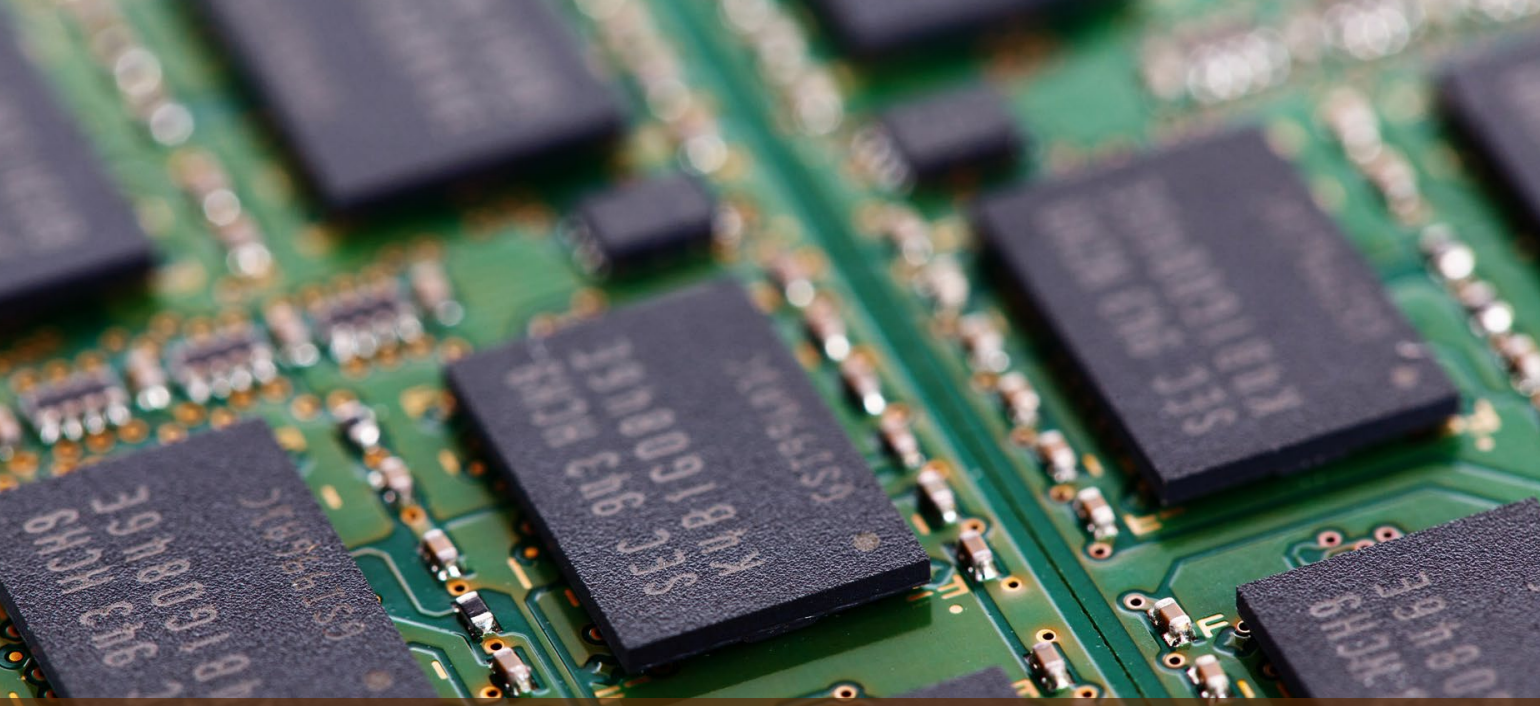
1. Describe unconditional transfer instructions.
2. What are conditional transfer instructions?
3. Describe general-purpose data movement instructions.
4. What are formats instruction cycles and sub cycles?
5. Write a short note on micro-programmed control?

Check Your Result

1. (b)	2. (b)	3. (a)	4. (c)	5. (c)
6. (d)	7. (a)	8. (c)	9. (a)	10. (a)

REFERENCES

1. Asanovic, Krste (2017). The RISC V Instruction Set Manual (PDF) (2.2 ed.). Berkeley: RISC-V Foundation.
2. Astha Singh. "Computer Organization - Control Unit and design". GeeksforGeeks. Retrieved 25 May 2019.
3. Barkalov, Alexander (2009). Logic synthesis for FSM based control units / Alexander Barkalov and Larysa Titarenko. Berlin: Springer. ISBN 978-3-642-04308-6.
4. Power ISA(tm) (3.0B ed.). Austin: IBM. 2017. Retrieved 26 December 2019.
5. Williams, R. D.; Klenke, R. H.; Aylor, J. H. (May 2003). "Teaching computer design using virtual prototyping". IEEE Transactions on Education. 46 (2): 296–301. doi:10.1109/te.2002.808278. ISSN 0018-9359.
6. Wiśniewski, Remigiusz (2009). Synthesis of compositional microprogram control units for programmable devices. Zielona Góra: University of Zielona Góra. p. 153. ISBN 978-83-7481-293-1.



CHAPTER 4

FOCUS ON MEMORY

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Explain the concept of memory and memory hierarchy
2. Understand main memory and auxiliary memory
3. Describe associative memory, cache memory, and virtual memory

"The computer was born to solve problems that did not exist before."

—Bill Gates

INTRODUCTION

Computer memory, device that is used to store data or programs (sequences of instructions) on a temporary or permanent basis for use in an electronic digital computer. Computers represent information in binary code, written as sequences of 0s and 1s. Each binary digit (or "bit") may be stored by any physical system that can be in either of two stable states, to represent 0 and 1. Such a system is called bistable. This could be an on-off switch, an electrical capacitor that can store or lose a charge, a magnet with its polarity up or down, or a surface

that can have a pit or not. Today capacitors and transistors, functioning as tiny electrical switches, are used for temporary storage, and either disks or tape with a magnetic coating, or plastic discs with patterns of pits are used for long-term storage.

Computer memory is divided into main (or primary) memory and auxiliary (or secondary) memory. Main memory holds instructions and data when a program is executing, while auxiliary memory holds data and programs not currently in use and provides long-term storage.

The computer memory holds the data and instructions needed to process raw data and produce output. The computer memory is divided into large number of small parts known as cells. Each cell has a unique address which varies from 0 to memory size minus one.

Computer memory is of two types: Volatile (RAM) and Non-volatile (ROM). The secondary memory (hard disk) is referred as storage not memory.

Keyword

Storage Devices capable of storing data and this term usually refers to mass storage devices, such as disk and tape drives.

4.1 CONCEPT OF MEMORY AND MEMORY HIERARCHY

The memory in a computer system is required for storage and subsequent retrieval of the instructions and data. A computer system uses a variety of devices for storing these instructions and data that are required for its operations. Normally memory classifies the information to be stored on computer in two basic categories: data and instructions. The **storage devices** along with the algorithm or information on how to control and manage these storage devices constitute the memory system of a computer.

A memory system is a very simple system yet it exhibits a wide range of technology and types. The basic objective of a computer system is to increase the speed of computation. Likewise the basic objective of a memory system is to provide fast, uninterrupted access by the processor to the memory such that the processor can operate at the speed at which it is expected to work. But does this kind of technology where there is no speed gap between the processor and the memory speed exist? Yes, it does, but unfortunately as the access time becomes less

and less the cost per bit of memory becomes increasingly higher. In addition, normally these memories require power supply till the information needs to be stored. These things are not very convenient, but on the other hand the memories with smaller cost have very high access time that will result in slower operations of the CPU. Thus, the cost versus access time anomaly has leaded to a hierarchy of memory where supplement fast memories with large, cheaper, slower memories. These memory units may have very different physical and operational characteristics, therefore, making the memory system very diverse in type, cost, organization, technology and performance. This **memory hierarchy** will work only if the frequency of access to the slower memories is significantly less than the faster memories.

Keyword

Memory Hierarchy is used in computer architecture when discussing performance issues in computer architectural design, algorithm predictions, and the lower level programming constructs such as involving locality of reference.

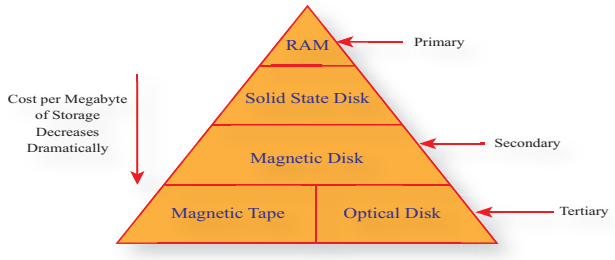


Figure 4.1: Memory hierarchy.

The disk cache may be a portion of RAM, sometimes called the soft disk cache that is used to speed up the access time on a disk. In some newer technologies such a memory can be a part disk drive itself; such a memory is sometimes called the hard disk cache or buffer. These hard disk caches are more effective, but they are expensive, and therefore smaller. Almost all modern disk drives include a small amount of internal cache.

4.1.1 Characteristics Terms for Various Memory Devices

The following terms are most commonly used for identifying the comparative behavior of various memory devices and technologies.

Storage Capacity

It is representative of the size of the memory. The capacity of internal memory and main memory can be expressed in terms of number of words or bytes. The storage capacity of the external memory is normally measured in term of **byte**.

Keyword

Byte is a unit of digital information in computing and telecommunications that most commonly consists of eight bits.

Unit of Transfer

A unit of transfer is the number of bits read in or out of the memory in a single read or write operation. For main memory and internal memory, the normal unit of transfer of information is equal to the word length of a processor. In fact it depends on the number of data lines in and out of the memory module. In general, these lines are kept equal to the word size of the processor.

Access Modes

The unit of transfer the next important characteristic is the access mode in which the information is accessed from the memory. A memory is considered to consist of various memory locations.

Permanence of Storage

Is it possible to lose information stored by the memory over a period of time? If yes, then what can be the reasons for the loss of information and what should be done to avoid it?

There are several reasons for information destruction; these are destructive readout, dynamic storage, volatility, and hardware failure. There can be some memories where the stored 1 loses its strength to become 0 over a period of time. These kinds of memories require refreshing. The memories, which require refreshing, are termed dynamic memories. In contrast, the memories, which do not require refreshing, are called static memories.

DRAM (Dynamic RAM)

DRAM technologies are mainly used as main memories. Presently DRAMs are available in many different forms. The basic anomaly

relating to DRAMs is the cost versus speed. A faster processor needs faster memories. However, it is important to have more main memory rather than better system memory.

Cycle Time

It is the minimum time lapse between two consecutive read requests. Is it equal to access time? Yes, for most of the memories except the ones in which destructive readout is encountered or a refreshing cycle is needed prior to next read. Cycle time for such memories is the access time (time elapsed when a read request is made available) plus writing time as after the data has been made available, the information has to be written back in the same location as the previous value has been destroyed by reading.

Keyword

Bandwidth is a measurement of bit-rate of available or consumed data communication resources expressed in bits per second or multiples of it.

Data Transfer Rate

The amount of information that can be transferred in or on the memory in a second it termed the data transfer rate or **bandwidth**. It is measured in bits per second. The maximum number of bits that can be transferred in a second depends on how many bits can be transferred in or out of the memory simultaneously and thus the data bus width becomes one of the controlling factors.

Physical Characteristics

In this respect the memory devices can be categorized into four main categories electronic, magnetic, mechanical and optical.

Keyword

Memory Address Register (MAR) is a CPU register that either stores the memory address from which data will be fetched to the CPU or the address to which data will be sent and stored.

4.2 MAIN MEMORY

As the name implies, the main memory provides the main storage for a computer. Two CPU registers are used to interface the CPU to the main memory. These are the **memory address register (MAR)** and the memory data register (MDR). The MDR is used to hold the data to be stored and/or retrieved in/from the memory location whose address is held in the MAR. It is possible to visualize a typical internal main memory structure as consisting of rows and columns of basic cells. Each cell is

capable of storing one bit of information. A word select line is used to enable all cells in a row for read or write. Data (bit) lines are used to input or output the contents of cells. Each memory cell is connected to two data lines. A given data line is common to all cells in a given column.

REMEMBER

One of the requirements for a memory device is that it should exhibit two well-defined physical states, such that 0 and 1 can be represented in those two states. The data transfer rate of the memory depends on how quickly the state can be recognized and altered.

The word main is used to distinguish it from external mass storage devices such as disk drives. Another term for main memory is RAM. The computer can manipulate only data that is in main memory. Therefore, every program is executed and every file to access must be copied from a storage device into main memory. The amount of main memory on a computer is crucial because it determines how many programs can be executed at one time and how much data can be readily available to a program.

Because computers often have too little main memory to hold all the data they need, computer engineers invented a technique called swapping, in which portions of data are copied into main memory as they are needed. Swapping occurs when there is no room in memory for needed data. When one portion of data is copied into memory, an equal-sized portion is copied (swapped) out to make room.

The main memory is a random access memory. It is normally organized as words of fixed length. The length of a word is called word length. Each of these memory words has an independent address and each has the same number of bits. Normally the total numbers of words in the memory are some power of 2. Some typical memory word sizes are 8 bits, 16 bits, 32 bits, etc. The main memory can be both read and write into, therefore, it is called read-write memory.



4.2.1 RAM and ROM Chips

A ROM chip does not require a constant source of power to retain the information stored on it, whereas a RAM chip does require a constant source of power to retain the information. When a computer is turned on, and programs are being run, the programs information is stored in the RAM chip for better efficiency and speed. However, as soon as the computer is turned off any information stored in the RAM chip is lost. With a ROM chip any information stored on the chip remains on it with or without power.

ROM chips are commonly used in gaming system cartridges, like the original Nintendo, Gameboy, Sega Genesis, and a number of others. The game cartridge stores the game programming on a ROM chip which is read by the game console when the cartridge is inserted into the console. ROM chips can also be found in computers, to store the programming needed to begin the initial **boot up** process when turning on a computer. Using non-volatile storage is the only way to begin the startup process for computers and other devices that use similar a startup process. The most common type of programmable ROM (PROM) chip in a computer is the BIOS. RAM chips are also used in computers, as well as other devices, to store information of the programs currently running on the computer. However, only volatile information can be stored on RAM chips, due to that fact that if power is lost to the computer all information stored in the RAM is lost. The oldest ROM-Type storage media can be dated back to 1932 with drum memory. Today, ROM-Type storage media is still used and continues to be improved upon for better performance and storage capacity.

4.2.2 Memory Address Map

Memory address map is address space that assignment to each memory chip. *Example:* 512 bytes RAM and 512 bytes ROM

Component		Hexa address 10			Address bus						
			9	8	7	6	5	4	3	2	1	
RAM	1	0000-007F	0	0	0	x	x	x	x	x	x	x
RAM	2	0080-00FF	0	0	1	x	x	x	x	x	x	x
RAM	3	0100-017F	0	1	0	x	x	x	x	x	x	x
RAM	4	0180-01FF	0	1	1	x	x	x	x	x	x	x
ROM		0200 - 03FF	1	x	x	x	x	x	x	x	x	x

4.2.3 Memory Connection to CPU

- RAM and ROM chips are connected to a CPU through the data and address buses.
- The low-order lines in the address bus select the byte within the chips and other lines in the **address bus** select a particular chip through its chip select inputs.

Keyword

Address Bus is a computer bus that is used to specify a physical address.

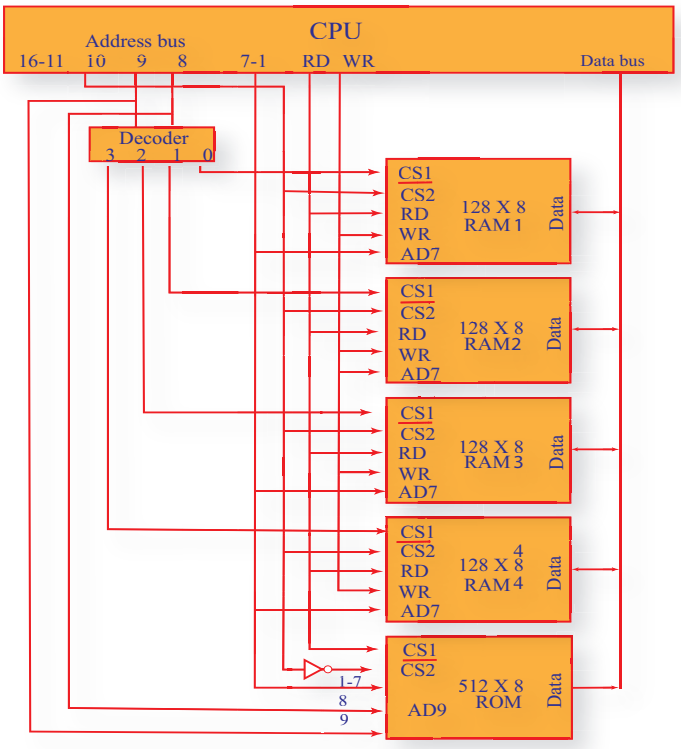


Figure 4.2: Memory connection to CPU.

4.3 AUXILIARY MEMORY

The Auxiliary memory devices used in a computer system are magnetic disk, tapes, magnetic drums and more recently optical disks. The important features of any devices are its access mode, access time, transfer rate, capacity and cost. The average time to reach a storage location in memory and obtain its contents is the access time. In electro-mechanical devices access time is equal to the sum of seek time and transfer time (the time required

to transfer data to and from any device). Auxiliary storage is logically divided into records or blocks. A record/block consists of a number of words. Input/output from / on **auxiliary memory** is always done to enter blocks.

The main memory construction is costly. Therefore, it has to be limited in size. The main memory is used to store only those instructions and data which are to be used immediately. However, a computer has to store a large amount of information. The bulk of information is stored in the auxiliary memory. This is also called backing storage or secondary storage. They include hard disk, floppy disks, CD-ROM, USB flash drives, etc.

When the electricity supply to the computer is off, all data stored in the primary storage is destroyed. On the other hand, this is not true for secondary storage. The data stored in secondary storage can be stored for the desired time.

The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. Other components used, but not as frequently, are magnetic drums, magnetic bubble memory, and optical disks. To understand fully the physical mechanism of auxiliary memory devices one must have knowledge of magnetic, electronics, and electromechanical systems.

Although the physical properties of these storage devices can be quite complex, their logical properties can be characterized and compared by a few parameters. The important characteristics of any device are its access mode, access time, transfer rate, capacity, and cost.

The average time required to reach a storage location in memory and obtain its contents is called the access time. In electromechanical devices with moving parts such as disks and tapes, the access time consists of a seek time required to position the read-write head to location and a transfer time required to transfer data to or from the device. Because the seek time is usually much longer than the transfer time, auxiliary storage is organized in records or blocks. A record is a specified number of characters or words. Reading or writing is always done on entire records. The transfer rate is the number of characters or words that the device can transfer per second, after it has been positioned at the beginning of the record.

Magnetic drums and disks are quite similar in operation. Both consist of high-speed rotating surfaces coated with a magnetic recording medium. The rotating surface of the drum is a cylinder and that of the disk, a round flat plate. The recording surface rotates at uniform speed and is not started or stopped during access operations. Bits are recorded as magnetic spots on the surface as it passes a stationary mechanism called a write head. Stored bits are detected by a change in magnetic field produced by a recorded spot on the surface as it passes through a read head. The amount of surface available for recording in a disk is greater than in a drum of equal physical size.

Therefore, more information can be stored on a disk than on a drum of comparable size. For this reason, disks have replaced drums in more recent computers.

4.3.1 Magnetic Disks

A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface. Bits are stored in magnetized surface in spots along concentric circles called tracks. Tracks are commonly divided into sections called sectors. Disk that are permanently attached and cannot removed by occasional user are called hard disks. A disk drive with removable disks is called a floppy disk drive.

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

4.3.2 Magnetic Tape

Magnetic tape is one of the older types of magnetic storage media. The magnetic tape recorder was invented in 1928 and was primarily used for analog audio recordings. Before music CDs were introduced in the 1980s, portable music devices used magnetic tape in the form of music cassettes. Early computers adapted this technology to store digital information. One of the major weaknesses is that information on a tape can only be accessed in a very sequential fashion. This is fine if he/she wants to listen to a whole music album in sequence, but computer systems typically need to access data in a non-sequential manner. For magnetic tape, this means one may need to fast-forward through a lot of tape to get to a specific piece of data. While magnetic tape is a very cheap way to store data, the very slow access to the data meant that it was primarily used for creating back-ups of data in older forms of storage failed. Tape backup systems are still in use today, but their importance has greatly declined with the advance of cheap, large capacity hard disk drives.

4.3.3 CD

A compact disc (CD) is a small, portable, round medium made of molded polymer (close in size to the floppy disk) for electronically recording, storing, and playing back audio, video, text, and other information in digital form. Tape cartridges and CDs generally replaced the phonograph record for playing back music. At home, CDs have tended to replace the tape cartridge although the latter is still widely used in cars and portable playback devices.

Initially, CDs were read-only, but newer technology allows users to record as well. CDs will probably continue to be popular for music recording and playback. A newer technology, the digital versatile disc (DVD), stores much more in the same space and is used for playing back movies.

Some variations of the CD include:

- CD-ROM
- CD-I
- CD-RW
- CD-ROM XA
- CD-W
- Photo CD
- Video CD

4.3.4 DVD

Digital Versatile Disk (DVD) drives are now widely available in computers as well as home entertainment centers. DVD-ROM drives can read data, such as stored commercial videos for playing. DVD-RW allows DVDs to be created on a computer. The DVD is a flat disk, the size of a CD - 4.7 inches diameter and .05 inches thick. Data are stored in a small indentation in a spiral track, just like in the CD. DVD disks are read by a laser beam of shorter wave-length than used by the CD ROM drives. This allows for smaller indentations and increased storage capacity. The data layer is only half as thick as in the CD-ROM. This opens the possibility to write data in two layers. The outer gold layer is semitransparent, to allow reading of the underlying silver layer. The laser beam is set to two different intensities, strongest for reading the underlying silver layer.

A 4.7 GB side of a DVD can hold 135 minutes top quality video with 6 track stereo. This requires a transmission rate of 4692 bits per second. The 17 GB disk holds 200 hours top quality music recording. MGEG is the data coding for movies similar to JPEG for the DVD region. The movies are made in two formats, each with their own

coding. The DVD drives come in 2X, 4X, etc. versions, like the CD-ROM's. The DVD drives will not replace the magnetic hard disks. The hard disks are being improved as rapidly as DVD, and they definitely offer the fastest seek time and transmission rate. No optic media can keep up with this. But the DVD will undoubtedly gain a place as the successor to the CD ROM and is playing an important role in the blending of computers and entertainment centers.

4.4 ASSOCIATIVE MEMORY

Associative memory is a storage unit of digital computers in which selection is performed not according to concrete address but rather to a preset combination of attributes characteristic of the desired information. Such attributes can be part of a word, attached to it for detection among other words, certain features of the word itself (*For example,* the presence of specific codes in its digits), the absolute value of a word, and its presence in a preset range, and so on.

The operation of an associative memory is based on the representation of all information in the form of a sequence of zones according to properties and characteristic attributes. The retrieval of information is reduced to the determination of the zone according to the preset attributes by means of scanning and comparison of those attributes with the attributes that are stored in the associative memory. There are two basic methods of realizing the associative memory. The first is the construction of a memory with storage cells that have the capability of performing simultaneously the functions of storage, nondestructive reading, and comparison. Such a method of realizing an associative memory is called network parallel-associative that is, the required sets of attributes are preserved in all the memory cells, and the information that possesses a given set of attributes is searched for simultaneously and independently over the entire storage capacity. Card indexes for edge-punched cards are prototypes of such an associative memory. Thin-film kryotrons, transfluxors, biaxes, magnetic thin films, and so on are used as storage elements of network-realized associative memories.

The second method of realizing an associative memory is the programmed organization (modeling) of the memory. It consists of the establishment of associative connections between the information contained in the memory by means of ordered arrangement of the information in the form of sequential chains or groups (lists) connected by linkage addresses whose codes are stored in the same memory cells. This procedure is the more suitable for practical realization in dealing with large volumes of information because it provides for the use of conventional accumulators with address reference.

The use of an associative memory considerably facilitates the programming and solution of informational-logical problems and accelerates by hundreds (thousands) of times the speed of retrieval, analysis, classification, and processing of data associative

memory in computer organization is the memory unit accessed by the content of data. It is also the memory unit which storage locations are identified by their contents. This usually takes place in the hard drive. Some points of associative mapping are as follows:

- Also called as Content-addressable memory (CAM), associative storage, or associative array
- Content-addressed or associative memory refers to a memory organization in which the memory is accessed by its content (as opposed to an explicit address).
- It is a special type of computer memory used in certain very high speed searching applications.
- In standard computer memory (random access memory or RAM) the user supplies a memory address and the RAM returns the data word stored at that address.
- In CAM the user supplies a data word and then CAM searches its entire memory to see if that data word is stored anywhere in it. If the data word is found, the CAM returns a list of one or more storage addresses where the word was found.
- CAM is designed to search its entire memory in a single operation.
- It is much faster than RAM in virtually all search applications.
- An associative memory is more expensive than RAM, as each cell must have storage capability as well as logic circuits for matching its content with an external argument.
- Associative memories are used in applications where the search time is very critical and short.
- Associative memories are expensive compared to RAMs because of the add logic associated with each cell.

4.4.1 Hardware Organization

- Compare each word in CAM in parallel with the Content of A (Argument Register)
- If CAM Word[i] = A, M(i) = 1
- Read sequentially accessing CAM for CAM Word(i) for M(i) = 1
- K(Key Register) provides a mask for choosing a particular field or key in the argument in A (only those bits in the argument that have 1's in their corresponding position of K are compared)

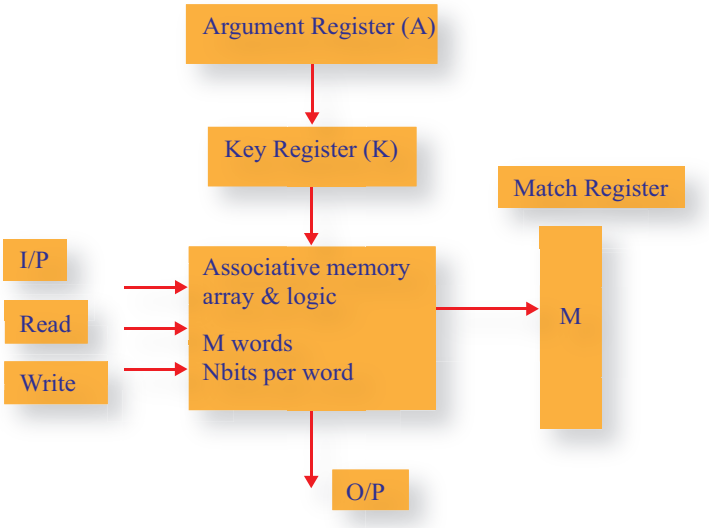


Figure 4.3: Hardware organization of associative memory.

4.4.2 Read Operation

By the help of read operation, access the content of data in memory unit. It is also the memory unit which storage locations are identified by their contents. This usually takes place in the hard drive.

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 bit 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. Other potential applications include:

- Library 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 other integrated circuit chip, with the data actually wired into the chip as part of the fabrication process. This presents as given problems:

- The data insertion step includes a relatively large fixed cost, whether one or thousands of copies of a particular ROM are fabricated.

- There is no room for error. If one bit is wrong, the whole batch of ROMs must be thrown out.

4.4.3 Write Operation

Write operation allows a main memory block to be loaded in any cache memory line. The logic of control of the cache memory interprets a memory direction as a label and a word field.

When only a small number of ROMs with particular memory content is needed, a less expensive alternative is the programmable ROM (PROM). Like the ROM, the PROM is nonvolatile and may be written into only once. For the PROM, the writing process is performed electrically and may be performed by a supplier or customer at a time later 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 useful for 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, EEPROM, and flash memory.

The optically erasable 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 PROM, holds its data virtually indefinitely. For comparable amounts of storage, the EPROM is more expensive than PROM, but it has the advantage of the multiple update capability.

A more attractive form of read-mostly memory is electrically erasable programmable read-only memory (EEPROM). This is a read-mostly 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 EEPROM combines the advantage of no volatility 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 bits per chip.

Another form of semiconductor memory is flash memory (so named because of the speed with which it can be reprogrammed). First introduced in the mid-1980s,

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 erase just blocks of memory rather than an entire chip. Flash memory gets its name because the microchip is organized so that a section of memory cells are erased in a single action or "flash." However, flash memory 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.

REMEMBER

The correspondence between the main memory blocks and the blocks in the cache is specified by a Mapping Function. When the cache is full and a memory word (instruction/ data) that is not in the cache is referenced; the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision constitutes the Replacement Algorithm.

4.5 CACHE MEMORY

To increase the speed of a processor, the time for accessing the information is to be reduced. For this a cache is used. The effectiveness of the cache is based on the property of programs called Locality of Reference.

Analysis of the programs show that most of their execution time is spent on routines in which many instructions in localized areas of the programs are executed repeatedly (These instructions may constitute simple loop, nested loop etc.) and the remainder of the program is accessed not so frequently. This is referred to as locality of reference. There are two concepts behind the locality of reference

- Temporal
- Spatial

Temporal means that a recently executed instruction is likely to be executed again very soon. A spatial aspect means that the instructions very close to a recently executed instruction (cache block/cache line) are also to be executed soon. If the active segments of such a program or data can be placed in a cache, the execution time can be reduced. Consider the simple arrangement When a Read request is received from the CPU, the contents of a block of memory words containing the location specified are transferred into the cache one word at a time. The number of blocks that can be stored in the cache memory is small compared to the total number of blocks in the memory.

The CPU does not need to know about the existence of the cache. It simply issues Read or Write requests using the addresses that refer to locations in the main memory. The cache control circuitry determines whether the requested word currently exists in the cache. If it is present, Read or Write operation is performed. Read hit or Write hit is said to have occurred.

For a Write operation, the system can proceed in two ways

- *Write-Through Protocol*: Cache location and the main memory location are updated simultaneously (simple but needs unnecessary write operation)
- *Write-Back / Copy-Back Protocol*: Update only the cache location and mark it as updated with an associated flag bit called as dirty or modified bit. The main memory word is updated later when the updated word is removed from the cache to make space for a new block.

During a Read operation, when the addressed word is not in the cache, a read-miss occurs. The block of words that contains the requested word is copied from the main memory to cache and then the requested particular word is forwarded to the CPU. Alternatively, this word may be sent to the CPU as soon as it is read from the main memory. This approach, called Load-through (or early restart) reduces the CPU's waiting period at the expense of more complex circuitry.

During a Write operation, if the addressed word is not in the cache, a write-miss occurs. Then if the write-through protocol is used, the information is written directly into the main memory. Write-back protocol, the block containing the addressed word is first brought into the cache and then the desired word in the cache is overwritten with the new information.



Figure 4.4: Cache memory.

Cache memory is intended to give fast memory speed, while at the same time providing a large memory size at a less expensive price. The cache is a small amount of high-speed memory, usually with a memory cycle time comparable to the time required by the CPU to fetch one instruction. The cache is usually filled from main memory when instructions or data are fetched into the CPU. Often the main memory will supply a wider data word to the cache than the CPU requires, filling the cache more rapidly. The amount of information which is replaces at one time in the cache is called the line size for the cache. This is normally the width of the data bus between the cache memory and the main memory. A wide line size for the cache means that

several instruction or data words are loaded into the cache at one time, providing a kind of prefetching for instructions or data. Since the cache is small, the effectiveness of the cache relies on the following properties of most programs:

- *Spatial locality*: Most programs are highly sequential; the next instruction usually comes from the next memory location.
- Data is usually structured, and data in these structures normally are stored in contiguous memory locations.
- *Short Loops*: These are a common program structure, especially for the innermost sets of nested loops. This means that the same small set of instructions is used over and over. Generally, several operations are performed on the same data values, or variables.

When a cache is used, there must be some way in which the memory controller determines whether the value currently being addressed in memory is available from the cache. There are several ways that this can be accomplished. One possibility is to store both the address and the value from main memory in the cache, with the address stored in a type of memory called associative memory or, more descriptively, content addressable memory.

Cache memories normally allow one of two things to happen when data is written into a memory location for which there is a value stored in cache:

- *Write Through Cache:* Both the cache and main memory are updated at the same time. This may slow down the execution of instructions which write data to memory, because of the relatively longer write time to main memory. Buffering memory writes can help speed up memory writes if they are relatively infrequent, however.
- *Write Back Cache:* Here only the cache is updated directly by the CPU; the cache memory controller marks the value so that it can be written back into memory when the word is removed from the cache. This method is used because a memory location may often be altered several times while it is still in cache without having to write the value into main memory. This method is often implemented using an ``ALTERED'' bit in the cache. The ALTERED bit is set whenever a cache value is written into by the processor. Only if the ALTERED bit is set is it necessary to write the value.

Memory Organization of Cache Memory

- *CPU:* Central Processing Unit is just like brain of a computer; and performs the arithmetical, logical operations of the system by carrying instructions on the code.
- The memory organization of a system

- At the core is CPU, and then are cache, then RAM and then storage device.

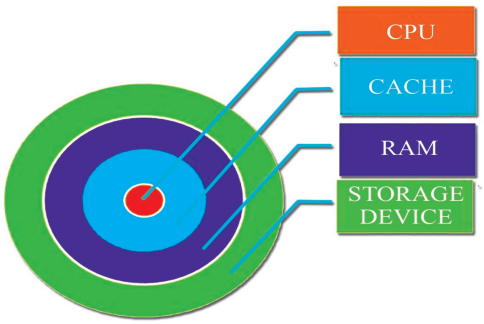


Figure 4.5: Memory organization of cache memory.

But how do these work?

When an application starts or data is to be read/write or any operation is to be performed then the data and commands associated with the specific operation are shifted from a slow moving storage device to a faster device. This faster device is Random Access Memory. This RAM is type of DRAM. RAM is placed here because it is a faster device, and whenever data/commands/instructions are needed by Processor, they provide them at a faster rate than slow storage devices. They serve as a cache memory for the storage devices. Although they are much faster than slow storage device but the processor processes at much a faster pace and they are not able to provide the needed data/instructions at that rate. So there is need of a device that is faster than RAM which could keep up with the speed of processor needs. Therefore the data required is transmitted to the next level of fast memory, which is known as CACHE memory. CACHE is also a type of RAM, but it is Static RAM, SRAM. SRAM are faster and costlier then DRAM because it has flip-flops (6 transistors) to store data unlike DRAM which uses 1 transistor and capacitor to store data in form of charge. Moreover they need not be refreshed unlike DRAM making it faster.

4.5.1 Direct Mapping

Simplest way to determine cache locations in which to store memory blocks is the direct mapping technique. Here block j of the main memory maps on to block j modulo 128 of the cache which means that main memory blocks 0, 128, 256,…. are stored in cache block 0, main memory blocks 1, 129, 257…. are stored in cache block 1 and so on.

Placement of a block in the cache is determined from the memory address. The memory address can be divided into three fields as shown.

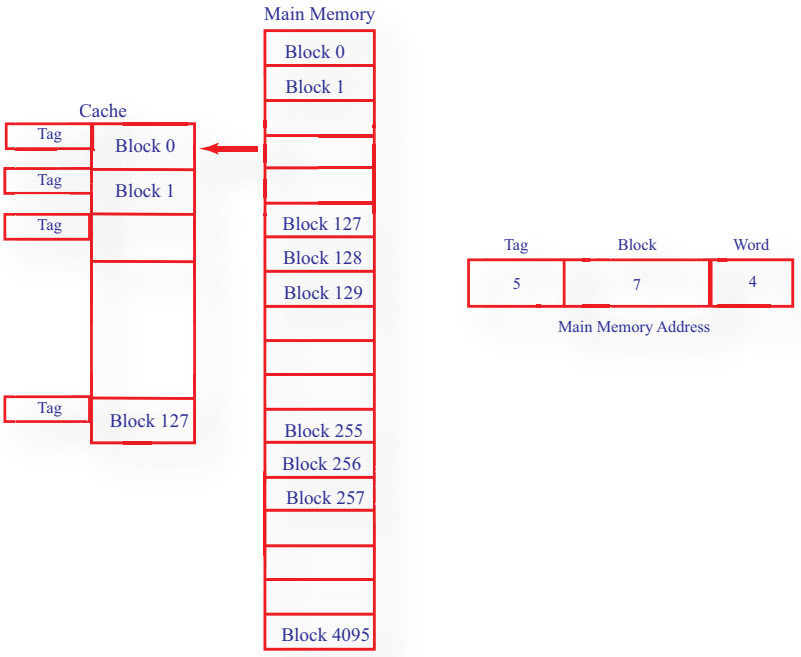


Figure 4.6: Direct mapping.

- Low order 4 bits select one of 16 words in a block
- Block field is given by 7 bits and determines the cache position in which this block must be stored
- Higher order 5 bits of the memory address are stored in 5 bit tag associated with its location in the cache. They identify which of the 32 blocks that are mapped into this cache position are currently reside in the cache (because maximum 32 blocks of main memory are assigned to cache block of cache)
- During execution the 7 bit cache block field of each address generated by the CPU points to a particular block location in the cache. The higher order 5 bits of the address are compared with the tag bits associated with that cache location. If they match, then the desired word is in that block of the cache. If there is no match, then the block containing the required word must first read from the main memory and loaded into the cache.
- Easy to implement but not very flexible.

This is the simplest form of mapping. One block from main memory maps into only one possible line of cache memory. As there are more blocks of main memory than there are lines of cache, many blocks in main memory can map to the same line in cache memory.

To implement this function, use the given formula as:

$\alpha = \beta \% \gamma$

Where α is the cache line number, β is the block number in main memory, γ is the total number of lines in cache memory and % being the modulus operator. The main disadvantage of using this type of mapping is that there is a fixed cache location for any given block in main memory. If two blocks of memory sharing the same cache line are being continually referenced, cache misses would occur and these two blocks would continuously be swapped, resulting in slower memory access due to the time taken to access main memory (or the next level of memory).

4.5.2 Associative Mapping

This type of mapping overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of cache. To do so, the cache control logic interprets a memory address as a tag and a word field. The tag uniquely identifies a block in main memory. The primary disadvantage of this method is that to find out whether a particular block is in cache, all cache lines would have to be examined. Using this method, replacement algorithms are required to maximize its potential. Associative cache mapping is the most complex, but it is most flexible with regards to where data can reside. A newly read block of main memory can be placed anywhere in a fully associative cache.

Some points related to associative mapping are as follows:

- Much more flexible mapping method.
- Here a main memory block can be placed into any cache block position. Therefore 12 tag bits are required to identify a memory block.
- Tag bits of an address received from the CPU are compared to the tag bits of each block of the cache to see if the desired block is present. This is called associative mapping.
- Space in the cache is used more effectively
- Cost of an associative cache is higher than the cost of a direct mapped cache, because of the need to search all 128 tag pattern to determine whether a given block is in the cache.

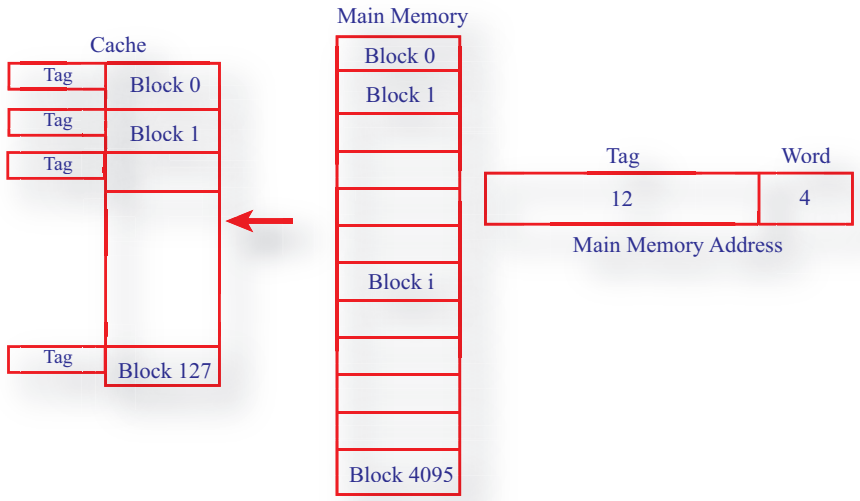


Figure 4.7: Associative mapping.

4.5.3 Set-Associative Mapping

This type of mapping is designed to utilize the strengths of the previous two mappings, while minimizing the disadvantages. The cache is divided into a number of sets containing an equal number of lines. Each block in main memory maps into one set in cache memory similar to that of direct mapping. Within the set, the cache acts as associative mapping where a block can occupy any line within that set. Replacement algorithms may be used within the set. Set associative cache mapping combines the best of direct and associative cache mapping techniques. As with a direct mapped cache, blocks of main memory data will still map into as specific set, but they can now be in any N-cache block frames within each set.

It is a combination of the direct mapping and associative mapping techniques. Here blocks of the cache are grouped into sets and allows a block of the main memory to reside in any of a specific set.

Hence few choices for block placement are possible which were not available in direct mapping (i.e. contention problem is reduced). At the same time the cost for hardware of the control circuitry can be reduced.

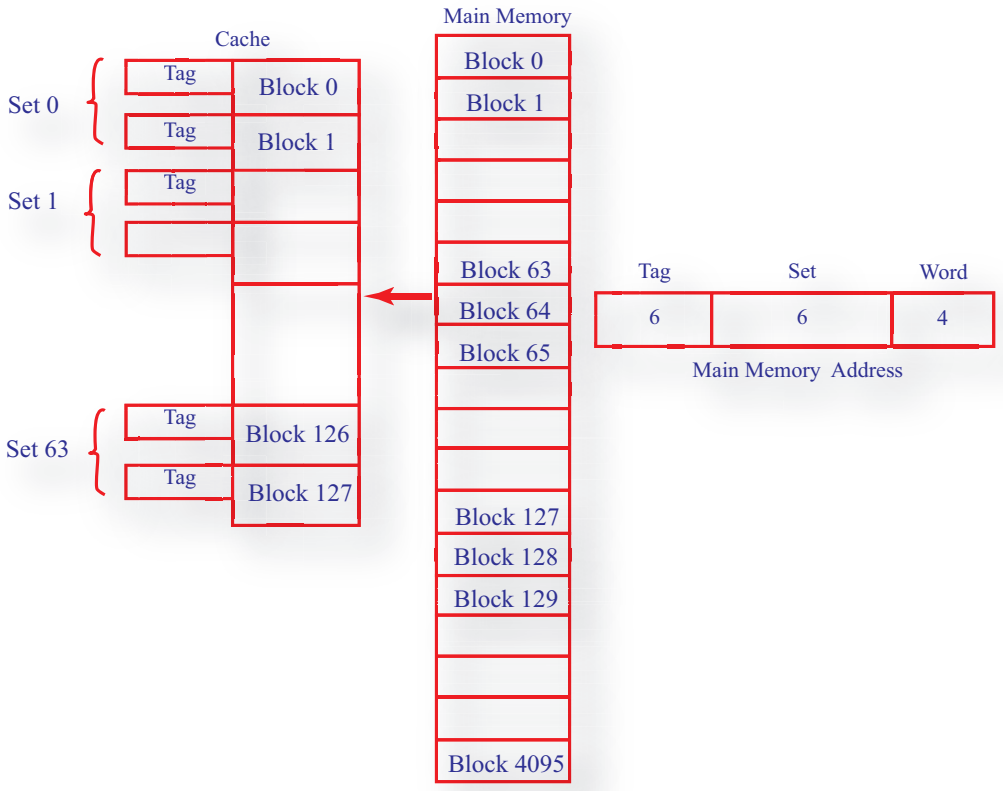


Figure 4.8: Set-associative mapping.

- Consider the example of Set associative mapping technique for a cache with 2 blocks per set. Therefore there are 63 sets. Maim memory blocks 0, 64, 128…….4032 map into cache set 0.
- They can occupy either of the two block positions within the sets. 6-bit Set field of the address determines which set of the cache contains the desired block. The Tag field (6-bits) of the address must then be compared to the tags of the two blocks of the selected set to check whether the desired block is present.
- The number of blocks per set can be changed to suit the requirements of a particular computer. 4 blocks per set can be accommodated by a 5-bit Set field. 8 blocks per set by a 4-bit Set field and so on.
- 12 blocks per set implies that there are no set bits i.e. fully associative technique.
- 1 block per set implies the direct mapping.
- One more control bit called Valid bit is provided for each block to indicate whether the block contains valid data.

4.5.4 Writing into Cache

If the particular address is found in the cache, the block of data is sent to the CPU, and the CPU goes about its operation until it requires something else from memory. When the CPU finds what it needs in the cache, a hit has occurred. When the address requested by the CPU is not in the cache, a miss has occurred and the required address along with its block of data is brought into the cache according to how it is mapped. Cache processing in some computers is divided into two sections: main cache and eavesdrop cache. Main cache is initiated by the CPU within. Eavesdrop is done when a write to memory is performed by another requestor (other CPU or IOC). Eavesdrop searches have no impact on CPU performances.

4.6 VIRTUAL MEMORY

Virtual memory is storage space which exists as central storage in a computer though it is mostly supported by data retained in secondary storage, transfer between the two being made automatically as required. Virtual memory is a feature of an operating system that enables a process to use a memory (RAM) address space that is independent of other processes running in the same system, and use a space that is larger than the actual amount of RAM present, temporarily relegating some contents from RAM to a disk, with little or no overhead.

REMEMBER

A memory chip must have additional decoding circuits that select the set of storage cells that are at a particular address and either store a value at that address or fetch what is stored there.

In a system using virtual memory, the physical memory is divided into equally-sized pages. The memory addressed by a process is also divided into logical pages of the same size. When a process references a memory address, the memory manager fetches from disk the page that includes the referenced address, and places it in a vacant physical page in the RAM. Subsequent references within that logical page are routed to the physical page. When the process references an address from another logical page, it too is fetched into a vacant physical page and becomes the target of subsequent similar references.

If the system does not have a free physical page, the memory manager swaps out a logical page into the swap area - usually a paging file on disk (in Windows XP: pagefile.sys), and copies (swaps in) the requested logical page into the now-vacant physical page. The page swapped out may belong to a different process.

There are many strategies for choosing which page is to be swapped out. (One is LRU: the Least Recently Used page is swapped out.) If a page is swapped out and then is referenced, it is swapped back in, from the swap area, at the expense of another page.

Virtual memory enables each process to act as if it has the whole memory space to itself, since the addresses that it uses to reference memory are translated by the virtual memory mechanism into different addresses in physical memory. This allows different processes to use the same memory addresses the memory manager will translate references to the same memory address by two different processes into different physical addresses. One process generally has no way of accessing the memory of another process. A process may use an address space larger than the available physical memory, and each reference to an address will be translated into an existing physical address. The bound on the amount of memory that a process may actually address is the size of the swap area, which may be smaller than the addressable space.

The size of the virtual memory on a system is smaller than the sum of the physical RAM and the swap area, since pages that are swapped in are not erased from the swap area, and so take up two pages of the sum of sizes.

DID YOU KNOW ?

The concept of virtual memory was first developed by German physicist Fritz-Rudolf Güntsch at the Technische Universität Berlin in 1956 in his doctoral thesis, logical design of a digital computer with multiple asynchronous rotating drums and automatic high speed memory operation.

4.6.1 The Concept of Virtual Memory in Computer Architecture

Virtual memory is a valuable concept in computer architecture that allows you to run large, sophisticated programs on a computer even if it has a relatively small amount of RAM. A computer with virtual memory artfully juggles the conflicting demands of multiple programs within a fixed amount of physical memory.

Physical vs. Virtual Addresses

A computer accesses the contents of its RAM through a system of addresses, which are essentially numbers that locate each byte. Because the amount of memory varies from PC to PC, determining which software will work on a given computer

becomes complicated. Virtual memory solves this problem by treating each computer as if it has a large amount of RAM and each program as if it uses the PC exclusively. *For example*, in operating system, such as Microsoft Windows or Apple's OS X, creates a set of virtual addresses for each program.

Paging

Virtual memory breaks programs into fixed-size blocks called pages. If a computer has abundant physical memory, the operating system loads all of a program's pages into RAM.

Paging File

With virtual memory, the computer writes program pages that have not been recently used to an area on the hard drive called a paging file. The file saves the data contained in the pages; if the program needs it again, the operating system reloads it when RAM becomes available.

Multiprogramming

Virtual memory with paging lets a computer run many programs at the same time, almost regardless of available RAM. This benefit, called multiprogramming, is a key feature of modern PC operating systems, as they accommodate many utility programs such as printer drivers, network managers and virus scanners at the same time as your applications Web browsers, word processors, email and media players.

ROLE MODEL



THE FATHER OF FLASH MEMORY: FUJIO MASUOKA

Fujio Masuoka was born on May 8, 1943 in Takasaki City, Gunma, Japan, he is the inventor of flash memory. In high school, Masuoka concentrated on the theory, believes that advances in electronic technology or achieved only through theoretical work. As a result of his studies, Masuoka also developed a deep understanding of economics and law. He holds a Bachelor of Science, Master of Science and Ph.D. in electrical engineering from Tohoku University respectively in 1966, 1968 and 1971. Soon after graduating, Masuoka joined the Toshiba Research and Development Center in April 1971.

Between 1972 and 1984, Masuoka made another significant breakthrough memory, dynamic memory cell pair with a double poly-silicon structure. In 1977, he moved to Toshiba's semiconductor division, where he developed a 1 Mbit DRAM memory.

Masuoka and then transferred to Toshiba memory engineered products division in 1980 to begin his work on the development of flash memory. He then shifted to the memory design engineering division of Toshiba in 1984, where he perfected and patented NOR flash memory. He presented his findings at the International Electron Device Meeting (IEDM) in San Francisco. A year later, he progressed to 256 Kbit flash memory. In April 1987, Masuoka returned to Toshiba Research and Development Center, where he began work to develop more advanced NAND-type flash memory flash disk embryo. Although groundbreaking, flash is not ready for commercialization.

To create and produce pre-fabricated commercial 4 Mbit flash memory chip, Masuoka takes to develop high-tech projected for a variety of circuit patterns on each layer of the microprocessor. But the estimated cost to make the technology is 10 million yen, which Toshiba initially reluctant to invest. Masuoka assure Toshiba's consumer electronics research executives that the 4 Mbit flash memory chip can be used for consumer digital cameras into the flash memory that serves as a "digital film." With funding from the consumer electronics division, Masuoka continued to

develop and present the flash 4 Mbit NAND-type flash memory on Solid-State Circuits Conference International (ISSCC) in New York City in 1989.

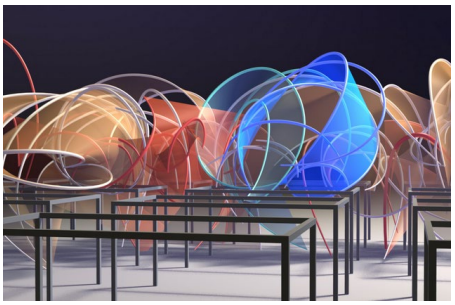
In 1994, Masuoka joined Tohoku University where he was a professor for 13 years before being appointed as Professor of the University Research Institute of Electrical Communication. For his pioneering work on flash memory, Masuoka has received numerous honors and awards in Japan including the Japan Prize from the Prime Minister in 1977 and National Award discovery in 1980. In 1997, he was the recipient of the IEEE Morris N. Liebmann Memorial Award. In 2007, Masuoka was awarded the Purple Ribbon Medal from the Emperor Akihito.

What Is The Moral Here?

Fujio Masuoka is known as the inventor of flash memory. He was excited mostly by the idea of non-volatile memory, memory that would last even when power was turned off. He has received many awards and wide recognition for his research and invention.

CASE STUDY

THE VAX/VMS VIRTUAL MEMORY SYSTEM



The VAX-11 minicomputer architecture was introduced in the late 1970's by Digital Equipment Corporation (DEC). DEC was a massive player in the computer industry during the era of the mini-computer; unfortunately, a series of bad decisions and the advent of the PC slowly (but surely) led to their demise [C03]. The architecture was realized in a number of implementations, including the VAX-11/780 and the less powerful VAX-11/750.

The OS for the system was known as VAX/VMS (or just plain VMS), one of whose primary architects was Dave Cutler, who later led the effort to develop Microsoft's Windows NT [C93]. VMS had the general problem that it would be run on a broad range of machines, including very inexpensive VAXen to extremely high-end and powerful machines in the same architecture family. Thus, the OS had to have mechanisms and policies that worked (and worked well) across this huge range of systems.

As an additional issue, VMS is an excellent example of software innovations used to hide some of the inherent flaws of the architecture. Although the OS often relies on the hardware to build efficient abstractions and illusions, sometimes the hardware designers do not quite get everything right; in the VAX hardware, we will see a few examples of this, and what the VMS operating system does to build an effective, working system despite these hardware flaws.

Memory Management Hardware

The VAX-11 provided a 32-bit virtual address space per process, divided into 512-byte pages. Thus, a virtual address consisted of a 23-bit VPN and a 9-bit offset. Further, the upper two bits of the VPN were used to differentiate which segment the page resided within; thus, the system was a hybrid of paging and segmentation, as we saw previously.

The lower-half of the address space was known as "process space" and is unique to each process. In the first half of process space (known as P_0), the user program is found, as well as a heap which grows downward.

In the second half of process space (P_1), we find the stack, which grows upwards. The upper-half of the address space is known as system space (S), although only half of it is used. Protected OS code and data reside here, and the OS is in this way shared across processes.

One major concern of the VMS designers was the incredibly small size of pages in the VAX hardware (512 bytes). This size, chosen for historical reasons, has the fundamental problem of making simple linear page tables excessively large. Thus, one of the first goals of the VMS designers was to make sure that VMS would not overwhelm memory with page tables.

The system reduced the pressure page tables place on memory in two ways. First, by segmenting the user address space into two, the VAX-11 provides a page table for each of these regions (P_0 and P_1) per process; thus, no page-table space is needed for the unused portion of the address space between the stack and the heap. The base and bounds registers are used as you would expect; a base register holds the address of the page table for that segment, and the bounds holds its size (i.e., number of page-table entries).

Second, the OS reduces memory pressure even further by placing user page tables (for P_0 and P_1, thus two per process) in kernel virtual memory. Thus, when allocating or growing a page table, the kernel allocates space out of its own virtual memory, in segment S. If memory comes under severe pressure, the kernel can swap pages of these page tables out to disk, thus making physical memory available for other uses.

Putting page tables in kernel virtual memory means that address translation is even further complicated. For example, to translate a virtual address in P_0 or P_1, the hardware has to first try to look up the page-table entry for that page in its page table (the P_0 or P_1 page table for that process); in doing so, however, the hardware may first have to consult the system page table (which lives in physical memory); with that translation complete, the hardware can learn the address of the page of the page table, and then finally learn the address of the desired memory access.

All of this, fortunately, is made faster by the VAX's hardware-managed TLBs, which usually (hopefully) circumvent this laborious lookup.

A Real Address Space

One neat aspect of studying VMS is that we can see how a real address space is constructed. Thus far, we have assumed a simple address space of just user code, user

data, and user heap, but as we can see above, a real address space is notably more complex.

For example, the code segment never begins at page 0. This page, instead, is marked inaccessible, in order to provide some support for detecting null-pointer accesses. Thus, one concern when designing an ad- dress space is support for debugging, which the inaccessible zero page provides here in some form. Perhaps more importantly, the kernel virtual address space (i.e., its data structures and code) is a part of each user address space. On a con- text switch, the OS changes the P0 and P1 registers to point to the appropriate page tables of the soon-to-be-run process; however, it does not change the S base and bound registers, and as a result the "same" kernel structures are mapped into each user address space. The kernel is mapped into each address space for a number of reasons. This construction makes life easier for the kernel; when, for example, the OS is handed a pointer from a user program (e.g., on a write() system call), it is easy to copy data from that pointer to its own structures. The OS is naturally written and compiled, without worry of where the data it is accessing comes from. If in contrast the kernel were located entirely in physical memory, it would be quite hard to do things like swap pages of the page table to disk; if the kernel were given its own address space, moving data between user applications and the kernel would again be complicated and painful. With this construction (now used widely), the kernel appears almost as a library to applications, albeit a protected one. One last point about this address space relates to protection. Clearly, the OS does not want user applications reading or writing OS data or code. Thus, the hardware must support different protection levels for pages to enable this. The VAX did so by specifying, in protect ion bits in the page table, what privilege level the CPU must be at in or der to access a particular page. Thus, system data and code are set t o a higher level of protection than user data and code; an attempted access to such information from user code will generate a trap into the OS, and (you guessed it) the likely termination of the offending process

Page Replacement

The page table entry (PTE) in VAX contains the following bits: a valid bit, a protection field (4 bits), a modify (or dirty) bit, a field reserved for OS use (5 bits), and finally a physical frame number (PFN) to store the location of the page in physical memory. The astute reader might note: no reference bit! Thus, the VMS replacement algorithm must make do without hardware support for determining which pages are active. The developers were also concerned about memory hogs, programs that use a lot of memory and make it hard for other programs to run.

Most of the policies we have looked at thus far are susceptible to such hogging; for example, LRU is a global policy that does not share memory fairly among processes.

Segmented FIFO

To address these two problems, the developers came up with the segmented FIFO replacement policy [RL81]. The idea is simple: each process has a maximum number of pages it can keep in memory, known as its resident set size (RSS). Each of these pages is kept on a FIFO list; when a process exceeds its RSS, the "first-in" page is evicted. FIF O clearly does not need any support from the hardware, and is thus easy to implement. Of course, pure FIFO does not perform particularly well, as we saw earlier. To improve FIFO's performance, VMS introduced two second- chance lists where pages are placed before getting evicted from memory, specifically a global clean-page free list and dirty-page list. When a process P exceeds its RSS, a page is removed from its per-process FIFO; if clean (not modified), it is placed on the end of the clean-page list; if dirty (modified), it is placed on the end of the dirty-page list. If another process Q needs a free page, it takes the first free page off of the global clean list. However, if the original process P faults on that page before it is reclaimed, P reclaims it from the free (or dirty) list, thus avoiding a costly disk access. The bigger these global second chance lists are, the closer the segmented FIFO algorithm performs to LRU [RL81].

Page Clustering

Another optimization used in VMS also helps overcome the small page size in VMS. Specifically, with such small pages, disk I/O during swap- ping could be highly inefficient, as disks do better with large transfers. To make swapping I/O more efficient, VMS adds a number of optimizations, but most important is clustering. With clustering, VMS groups large batches of pages together from the global dirty list, and writes them to disk in one fell swoop (thus making them clean). Clustering is used in most modern systems, as the freedom to place pages anywhere within swap space lets the OS group pages, perform fewer and bigger writes, and thus improve performance.

Other Neat VM Tricks

VMS had two other now-standard tricks: demand zeroing and copy on-write. We now describe these lazy optimizations. One form of laziness in VMS (and most modern systems) is demand zeroing of pages. To understand this better, let's consider the example of adding a page to your address space, say in your heap. In a naive implementation, the OS responds to a request to add a page toy our heap by finding a page in physical memory, zeroing it (required for security; otherwise you'd be able to see what was on the page from when some other process used it!), and then mapping it into your address space (i.e., setting up the page table to refer to that physical page as desired). But the naive implementation can be costly, particularly if the page does not get used by the process. With demand zeroing, the OS instead does very little work when the page is added to your address space; it puts an entry in the page

table that marks the page inaccessible. If the process then reads or writes the page, a trap into the OS takes place. When handling the trap, the OS notices (usually through some bits marked in the "reserved for OS" portion of the page table entry) that this is actually a demand-zero page; at this point, the OS then does the needed work of finding a physical page, zeroing it, and mapping it into the process's address space. If the process never accesses the page, all of this work is avoided, and thus the virtue of demand zeroing.

Another cool optimization found in VMS (and again, in virtually every modern OS) is copy-on-write (COW for short). The idea, which goes at least back to the TENEX operating system [BB+72], is simple: when the OS needs to copy a page from one address space to another, instead of copying it, it can map it into the target address space and mar k it read- only in both address spaces. If both address spaces only read the page, no further action is taken, and thus the OS has affected a fast copy without actually moving any data. If, however, one of the address spaces does indeed try to writ e to the page, it will trap into the OS. The OS will then notice that the page is a COW page, and thus (lazily) allocate a new page, fill it with the data, and map this new page into the address space of the faulting process. The process then continues and now has its own private copy of the page. COW is useful for a number of reasons. Certainly any sort of shared library can be mapped copy-on-write into the address spaces of many processes, saving valuable memory space. In U NIX systems, COW is even more critical, due to the semantics of fork() and exec(). As you might recall, fork() creates an exact copy of the address space of the caller; with a large address space, making such a copy is slow and data intensive. Even worse, most of the address space is immediately over-written by a subsequent call to exec(), which overlays the calling process's address space with that of the soon to be-exceed program. By instead performing a copy-on-write fork(), the OS avoids much of the needless copying and thus retains the correct semantics while improving performance.

SUMMARY

- Computer memory, device that is used to store data or programs (sequences of instructions) on a temporary or permanent basis for use in an electronic digital computer.
- The disk cache may be a portion of RAM, sometimes called the soft disk cache that is used to speed up the access time on a disk.
- The main memory is a random access memory. It is normally organized as words of fixed length. The length of a word is called word length.
- ROM chips can also be found in computers, to store the programming needed to begin the initial boot up process when turning on a computer.
- Magnetic drums and disks are quite similar in operation. Both consist of high-speed rotating surfaces coated with a magnetic recording medium.
- A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material.
- In CAM the user supplies a data word and then CAM searches its entire memory to see if that data word is stored anywhere in it. If the data word is found, the CAM returns a list of one or more storage addresses where the word was found.
- A ROM is created like any other integrated circuit chip, with the data actually wired into the chip as part of the fabrication process.
- Cache memory is intended to give fast memory speed, while at the same time providing a large memory size at a less expensive price. The cache is a small amount of high-speed memory, usually with a memory cycle time comparable to the time required by the CPU to fetch one instruction.

KNOWLEDGE CHECK

1. **The Load instruction does the following operation/s:**
   a. Loads the contents of a disc onto a memory location
   b. Loads the contents of a location onto the accumulators
   c. Load the contents of the PCB onto the register
   d. Both (a) and (c)
2. **Complete the following analogy: Registers are to RAM's as Cache's are to ..... .**
   a. System stacks
   b. Overlays
   c. Page Table
   d. TLB
3. **The BOOT sector files of the system are stored in ..... .**
   a. Hard disk
   b. ROM
   c. RAM
   d. Fast solid state chips in the motherboard
4. **The transfer of large chunks of data with the involvement of the processor is done by ..... .**
   a. DMA controller
   b. Arbitrator
   c. User system programs
   d. None of the above
5. **Which of the following technique/s used to effectively utilize main memory?**
   a. Address binding
   b. Dynamic linking
   c. Dynamic loading
   d. Both (b) and (c)
6. **Storage which stores or retains data after power off is called-**
   a. Volatile storage
   b. Non-volatile storage
   c. Sequential storage
   d. Direct storage

7. A permanent memory, which halls data and instruction for start-up the computer and does not erase data after power off.

a. Network interface card
b. CPU
c. RAM
d. ROM

8. Which of the following memories must be refreshed many times per second?

a. EPROM
b. ROM
c. Static RAM
d. Dynamic RAM

9. USB-type storage device is -

a. Secondary
b. Axillary
c. Tertiary
d. Primary

10. Which device is used to back up the data?

a. Floppy Disk
b. Tape
c. Network Drive.
d. All of the above

REVIEW QUESTIONS

1. Define the concept of memory and memory hierarchy.
2. Write a short note on auxiliary memory.
3. Define main memory in brief.
4. Explain about the associative memory.
5. Discuss about direct mapping and set-associative mapping.

Check Your Result

1. (b)	2. (d)	3. (b)	4. (a)	5. (c)
6. (b)	7. (d)	8. (d)	9. (a)	10. (d)

REFERENCES

1. "Intel: 35 Years of Innovation (1968–2003)" (PDF). Intel. 2003. Retrieved 26 June 2019.
2. The DRAM memory of Robert Dennard history-computer.com
3. "Toshiba: Inventor of Flash Memory". Toshiba. Retrieved 20 June 2019.
4. Han-Way Huang (5 December 2008). Embedded System Design with C805. Cengage Learning. p. 22. ISBN 978-1-111-81079-5. Archived from the original on 27 April 2018.
5. Hemmendinger, David (February 15, 2016). "Computer memory". Encyclopedia Britannica. Retrieved 16 October 2019.
6. "Robert Dennard". Encyclopedia Britannica. Retrieved 8 July 2019.



CHAPTER 5

INPUT AND OUTPUT DEVICES

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Understand the use of peripheral devices
2. Explain the process of I/O interface
3. Define the direct memory access (DMA)

"It's hardware that makes a machine fast. It's software that makes a fast machine slow."

— Craig Bruce

INTRODUCTION

An input device sends information to a computer system for processing, and an output device reproduces or displays the results of that processing. Input devices only allow for input of data to a computer and output devices only receive the output of data from another device.

Most devices are only input devices or output devices, as they can only accept data input from a user or output data generated by a computer. However, some devices

can accept input and display output, and they are referred to as I/O devices (input/output devices).

For example, as shown in the top half of the image, a keyboard sends electrical signals, which are received as input. Those signals are then interpreted by the computer and displayed, or output, on the monitor as text or images. In the lower half of the image, the computer sends, or outputs, data to a printer. Then, that data is printed onto a piece of paper, which is also considered output.

5.1 PERIPHERAL DEVICES

A peripheral is a device that is connected to a computer but is not part of the core computer architecture. The core elements of a computer are the central processing unit, power supply, motherboard and the computer case containing these three components. Technically speaking, everything else is considered a peripheral device; however, this is a somewhat narrow view, since various other elements are required for a computer to actually function, such as a hard drive and random-access memory (RAM).

Keyword

Peripheral is a "device that is used to put information into or get information out of the computer.

Most people use the term **peripheral** more loosely to refer to a device external to the computer case. He/she connect the device to the computer to expand the functionality of the system. *For example*, consider a printer. Once the printer is connected to a computer, you can print out documents. Another way to look at peripheral devices is that they are dependent on the computer system. *For example*, most printers cannot do much on their own, and they only become functional when connected to a printer.

Types of Peripheral Devices

There are many different peripheral devices, but they fall into three general categories:

- Input devices, such as a mouse and a keyboard
- Output devices, such as a monitor and a printer
- Storage devices, such as a hard drive or flash drive

Some devices fall into more than one category. Consider a CD-ROM drive; one can use it to read data or music (input) and one can use it to write data to a CD (output).

Peripheral devices can be external or internal. *For example*, a printer is an external device that connects using a cable, while an optical disc drive is typically located inside the computer case. Internal peripheral devices are also referred to as integrated peripherals. When most people refer to peripherals they typically mean external ones.

The term 'peripheral' also does not mean it is not essential to the functioning of the computer. Some devices, such as a printer, can be disconnected and the computer will keep on working just fine; however, remove the monitor of a desktop computer, and it becomes pretty much useless.



How Peripherals are connected

Internal peripherals are directly connected to the motherboard using one of the different types of slots on the motherboard. External devices can be connected using a wired connection or a wireless connection. As the name suggests, a wired connection uses a cable that needs to be plugged into the computer using a connector. The most widely used connector is a Universal Serial Bus (USB) connection, but several other types are used depending on the specific computer system and the type of peripheral. A wireless connection does not require a cable. The most widely used wireless connections are Bluetooth and Wi-Fi. Bluetooth is good for very short distances, so peripherals such as a wireless mouse and keyboard typically use a Bluetooth connection. Wi-Fi is good for longer distances. If one has set up a wireless network in home or office, one may be able to print wirelessly to printer if it is also connected to the network.

5.1.1 Mouse

A mouse is a hardware input device that was invented by Douglas Engelbart in 1963 while working at Xerox PARC, who at the time was working at the Stanford Research Institute, which was a think tank sponsored by Stanford University. The picture is of a Microsoft IntelliMouse and is an example of what a computer mouse may look like.

The mouse allows an individual to control a pointer in a **graphical user interface (GUI)** and manipulate on-screen objects such as icons, files, and folders. By using a mouse the user does not have to memorize commands, like those used in a text-based command line environment such as MS-DOS. *For example*, in MS-DOS a user would have to know the cd command and *dir* command to navigate to a folder and view the files. However, in Windows the user only has to double-click on the folder to view all the files.

Keyword

Graphical User Interface (GUI) is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators.

The Mouse was originally referred to as an X-Y Position Indicator for a Display System. Xerox after applied the mouse to its revolutionary Alto computer system in 1973. However, because of Alto's unfortunate success, it was first widely used in the Apple Lisa computer. Today, a pointing device is found on virtually every computer.

Advantages of Mouse

The advantages of mouse are as follows:

- ***Move***: Using the right or left hand drag the mouse up, down, left, or right to move the mouse pointer on the screen. If he/she reaches the edge of the mouse pad, pick up the mouse and move it to the opposite side and then continue dragging the mouse in the direction one wants the mouse pointer to move.
- ***Select***: When the mouse pointer is on top of an object on the screen click the mouse left button once to highlight the item. If one wants to select text, click the left-button at the end of the text one wants to select and then while continuing to hold down the button drag to the mouse to the left.

- ***Open***: Click the left mouse button twice quickly (double-click) to open the icon, program or document.
- ***Viewing Properties***: To view the properties of anything on the computer select the object or text with the left mouse button and then click with the right mouse button (right-click) to view the properties. If one views the properties of text one will be given a menu with options such as cut, copy, and paste.

5.1.2 Keyboard

As the name implies, a keyboard is basically a board of keys. Along with the mouse, the keyboard is one of the primary input devices used with a computer. The keyboard's design comes from the original typewriter keyboards, which arranged letters and numbers in a way that prevented the type-bars from getting jammed when typing quickly. This keyboard layout is known as the QWERTY design, which gets its name from the first six letters across in the upper-left-hand corner of the keyboard.

While the design of computer keyboards may have come from typewriters, today's keyboards have many other keys as well. Modifier keys, such as Control, Alt/Option, and Command (Mac) or the Windows key (Windows) can be used in conjunction with other keys as "shortcuts" to perform certain operations. Most of today's computer keyboards also have a row of function keys (F1 through F16) along the top of the keyboard, arrow keys arranged in an upside-down T, and a numeric keypad on the right-hand side. Some keyboards have even more buttons, allowing changing the system volume, ejecting a CD, or open programs such as the e-mail or Web browser. One of the primary input devices used with a computer, the keyboard looks very similar to the keyboards of electric typewriters, with some additional keys. Keyboards allows a computer user to input letters, numbers, and other symbols into a computer



Figure 5.1: Keyboard.

Overview of each Section of the Keyboard

The keyboard is the primary portion of the keyboard that contains all alphanumeric and symbol keys. Today most users use the QWERTY style keyboards.

Function Keys

The Function keys or F1 through F12 keys are used in programs as shortcut keys to performed frequently performed tasks. *For example*, the F1 key is the key to open the online help for most programs. Listing of F1 through F12 keys and what do they do?

Keypad

Although not available on all computer keyboards, especially laptops; the keypad gives the user a quick access to numbers and math functions such as plus, divide, times, and subtract.

Wrist Pad

This plastic wrist pad is included with the keyboard and is intended to help support the user's wrists and relieve stress found from typing on a keyboard. Many computer keyboards do not include a wrist pad; however, there are hundreds of different options for keyboard wrist pads that can be purchased at any local computer retail store or online.

Arrow Keys

Finally, the arrow keys are four directional arrow keys that allow the user to move their cursor and position on a page.

5.1.3 Printer

A printer is an external hardware device responsible for taking computer data and generating a hard copy of that data. Printers are one of the most used peripherals on computers and are commonly used to print text, images, and photos. In computers, a printer is a device that accepts text and graphic output from a computer and transfers the information to paper, usually to standard size sheets of paper. Printers are sometimes sold with computers, but more frequently are purchased separately. Printers vary in size, speed, sophistication, and cost. In general, more expensive printers are used for higher-resolution color printing.

Personal computer printers can be distinguished as impact or non-impact printers. Early impact printers worked something like an automatic typewriter, with a key striking an inked impression on paper for each printed character. The dot-matrix printer was a popular low-cost personal computer printer. It is an impact printer that strikes the paper a line at a time. The best-known non-impact printers are the inkjet printer, of which several makes of low-cost color printers are an example, and the laser printer. The inkjet sprays ink from an ink cartridge at very close range to the paper as it rolls by. The laser printer uses a laser beam reflected from a mirror to attract ink (called toner) to selected paper areas as a sheet rolls over a drum.

Keyword

Inkjet Printer is a type of computer printer that recreates a digital image by propelling droplets of ink onto paper, plastic, or other substrates.

Types of Printers

Types of printers are as follows:

- All-in-one (AIO) printer
- Dot Matrix printer
- **Inkjet printer**
- Laser printer
- Multifunction printer (MFP)
- Thermal printer
- LED printer

Printer Qualities

The printer qualities are as follows:

- ***Color***: Color is important for users who need to print pages for presentations or maps and other pages where color is part of the information. Color printers can also be set to print only in black-and-white. Color printers are more expensive to operate since they use two ink cartridges (one color and one black ink) that need to be replaced after a certain number of pages. Users who do not have a specific need for color and who print a lot of pages will find a black-and-white printer cheaper to operate.
- ***Resolution***: Printer resolution (the sharpness of text and images on paper) is usually measured in dots per

inch (DPI). Most inexpensive printers provide sufficient resolution for most purposes at 600 dpi.

- ***Speed***: If one do much printing, the speed of the printer becomes important. Inexpensive printers print only about 3 to 6 sheets per minute. Color printing is slower. More expensive printers are much faster.
- ***Memory***: Most printers come with a small amount of memory (*for example*, one megabyte) that can be expanded by the user. Having more than the minimum amount of memory is helpful and faster when printing out pages with large images or tables with lines around them (which the printer treats as a large image).

5.1.4 Monitor

The term "monitor" is often used synonymously with "computer screen" or "display." The monitor displays the computer's user interface and open programs, allowing the user to interact with the computer, typically using the keyboard and mouse.

Older computer monitors were built using cathode ray tubes (CRTs), which made them rather heavy and caused them to take up a lot of desk space. Most modern monitors are built using LCD technology and are commonly referred to as flat screen displays. These thin monitors take up much less space than the older CRT displays. This means people with LCD monitors have more desk space to clutter up with stacks of papers, pens, and other objects.

"Monitor" can also be used as a verb. A network administrator may monitor network traffic, which means he watches the traffic to make sure the bandwidth usage is within a certain limit and checks to see what external sources may be attempting to access the network. Software programs may monitor the system's CPU performance as well as RAM and hard disk usage.

Finally, monitors also refer to speakers used for monitoring sound. Audio engineers typically use "studio monitors" to listen to recordings. These high-end speakers allow the engineers to accurately mix and master audio tracks. So a sound mixer could be monitoring a recording visually using a computer monitor, while monitoring the sound using audio monitors at the same.

Types of Monitor

Monitor is a standard output devices. Which shows the output information? Monitor is also one of the important parts of computer, because without monitor, system cannot work. So this is the standard output device.

Monitors are three types:

- CRT Monitors
- LCD Monitors
- LED Monitors

CRT Monitors

CRT stands for Cathode Ray Tube. These are all old model monitors. Main disadvantages are heavy weight; occupy more place, high power consumption, and high radiation and supports low resolution. These are all main reasons to prefer LCD or LED monitors.



LCD Monitors

LCD stands for Liquid Crystal Display. The main advantage of LCDs are light weighty, occupy less place, 12v power consumption, and low radiation and supports high resolutions.



LED Monitors

LED stands for Light Emitting Diode. Actually there are no differences between LCD and LED monitors, but better picture quality in LED monitors. So now all are prefer LED monitors than LCD monitors.



DID YOU KNOW ?

The world's first computer printer was a 19th-century mechanically driven apparatus invented by Charles Babbage for his difference engine. The introduction of the low-cost laser printer in 1984 with the first HP LaserJet, and the addition of PostScript in next year's Apple LaserWriter, set off a revolution in printing known as desktop publishing.

5.2 I/O INTERFACE

An input-output interface is the piece of equipment or location at which information can be input and output from a device such as a computer. An interface is a data path between two separate devices in a computer system. Interface to buses can be classified based on the number of bits that are transmitted at a given time to serial versus parallel ports. In a serial port, only 1 bit of data is transferred at a time. Mice and modems are usually connected to serial ports. A parallel port allows more than 1 bit of data to be processed at once. Printers are the most common peripheral devices connected to parallel ports. Table 5.1 shows a summary of the variety of buses and interfaces used in personal computers.

Table 5.1: Bus interface

Bus Interface	Description
PS2	A type of port (or interface) that can be used to connect mice and keyboards to the computer. The PS2 port is sometimes called the mouse port.
Industry standard architecture (ISA)	ISA was originally an 8-bit bus and later expanded to a 16-bit bus in 1984. In 1993, Intel and Microsoft introduced a plug and play ISA bus that allowed the computer to automatically detect and set up computer ISA peripherals such as a modem or sound card.
Extended industry standard Architecture (EISA)	EISA is an enhanced form of ISA, which allows for 32-bit data transfers, while maintaining support for 8- and 16-bit expansion boards. However, its bus speed, like ISA, is only 8 MHz EISA is not widely used, due to its high cost and complicated nature.
Micro channel architecture (MCA)	MCA was introduced by IBM in 1987. It offered several additional features over the ISA such as a 32-bit bus, automatically configured cards and bus mastering for greater efficiency. It is slightly superior to EISA, but not many expansion boards were ever made to fit MCA specifications.
VESA (Video electronics Standards association) local bus	The VESA, a nonprofit organization founded by NEC, released the VLB in 1992. It is a 32-bit bus that had direct access to the system memory at the speed of the processor, commonly the 486 CPU (33/40 MHz). VLB 2.0 was later released in 1994 and had a 64-bit bus and a bus speed of 50 MHz
Peripheral component interconnect (PCI)	PCI was introduced by Intel in 1992, revised in 1993 to version 2.0, and later revised in 1995 to PCI 2.1. It is a 32-bit bus that is also available as a 64-bit bus today. Many modern expansion boards are connected to PCI slots.
Advanced graphic port (AGP)	AGP was introduced by Intel in 1997. AGP is a 32-bit bus designed for the high demands of 3D graphics. AGP has a direct line to memory, which allows 3D elements to be stored in the system memory instead of the video memory. AGP is geared towards data-intensive graphics cards, such as 3D accelerators; its design allows for data throughput at rates of 266 MB/s.

Universal serial bus (USB)	USB is an external bus developed by Intel, Compaq, DEC, IBM, Microsoft, NEC and Northern Telecom. It was released in 1996 with the Intel 430HX Triton II Mother Board. USB has the capability of transferring 12 Mbps, supporting up to 127 devices. Many devices can be connected to USB ports, which support plug and play.
FireWire (IEEE 1394)	FireWire is a type of external bus, which supports very fast transfer rates: 400 Mbps. Because of this, FireWire is suitable for connecting video devices, such as VCRs, to the computer
Small computer system interface (SCSI)	SCSI is a type of parallel interface that is commonly used for mass storage devices. SCSI can transfer data at rates of 4 MB/s; in addition, there are several varieties of SCSI that support higher speeds: Fast SCSI (10 MB/s), Ultra SCSI and Fast Wide SCSI (20 MB/s), as well as Ultra Wide SCSI (40 MB/s).
Integrated drive electronics (IDE)	IDE is a commonly used interface for hard disk drives and CD-ROM drives. It is less expensive than SCSI, but offers slightly less in terms of performance.
Enhanced integrated drive electronics (EIDE)	EIDE is an improved version of IDE, which offers better performance than standard SCSI. It offers transfer rates between 4 and 16.6 MB/s.
PCI-X	PCI-X is a high performance bus that is designed to meet the increased I/O demands of technologies such as Fiber Channel, Gigabit Ethernet, and Ultra3 SCSI.
Communication and network riser (CNR)	CNR was introduced by Intel in 2000. It is a specification that supports audio, modem USB and local area networking interfaces of core logic chipsets.

5.2.1 Types of Interfaces

The interface to a peripheral from an I/O 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 5.2). In a parallel interface, there are multiple lines connecting the I/O module and the peripheral, and multiple bits are transferred simultaneously, just as all of the bits of a word are transferred simultaneously over the data bus.

With a new generation of high-speed serial interfaces, parallel interfaces are becoming much less common.

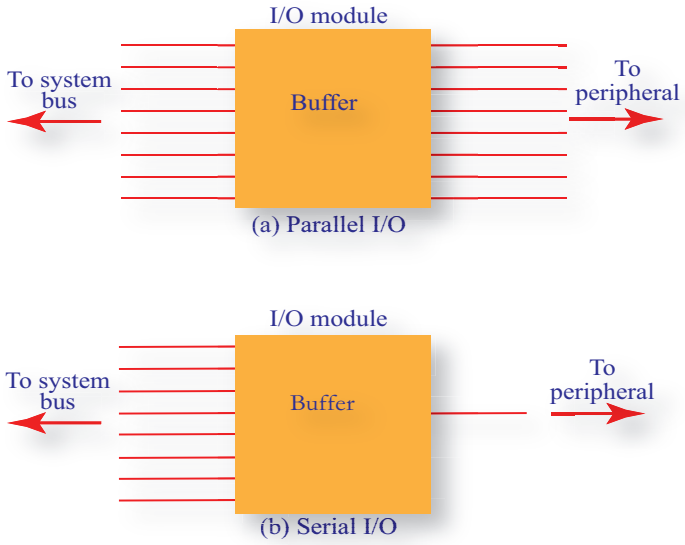


Figure 5.2: Parallel and serial I/O.

REMEMBER

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 use for higher-speed peripherals, such as tape and disk, while the serial interface has traditionally been used for printers and terminals.

In either case, the I/O module must engage in a dialogue with the peripheral.

In general terms, the dialogue for a write operation is as follows:

- The I/O module sends a control signal requesting permission to send data.
- The peripheral acknowledges the request.
- The I/O module transfers data (one word or a block depending on the peripheral).
- The peripheral acknowledges receipt of the data.

Key to the operation of an I/O module is an internal buffer that can store data being passed between the peripheral and the rest of the system. This buffer allows the I/O module to compensate for the differences in speed between the system bus and its external lines.

Point-to-Point and Multipoint Configurations

The connection between an I/O module in a computer system and external devices can be either point-to-point or multipoint. A point-to-point interface provides a dedicated line between the I/O module and the external device. On small systems (PCs, workstations), typical point-to-point links include those to the keyboard, printer, and external modem. A typical example of such an interface is the EIA-232 specification. Of increasing importance are multipoint external interfaces, used to support external mass storage devices (disk and tape drives) and multimedia devices (CD-ROMs, video, audio). These multipoint interfaces are in effect external buses, and they exhibit the same type of logic as the buses.

Keyword

Small Computer System Interface (SCSI) is a set of standards for physically connecting and transferring data between computers and peripheral devices.

FireWire Serial Bus

With processor speeds reaching gigahertz range and storage devices holding multiple gigabits, the I/O demands for personal computers, workstations, and servers are formidable. Yet the high-speed I/O channel technologies that have been developed for mainframe and supercomputer systems are too expensive and bulky for use on these smaller systems. Accordingly, there has been great interest in developing a high-speed alternative to **Small Computer System Interface (SCSI)** and other small system I/O interfaces. The result is the IEEE standard 1394, for a High Performance Serial Bus, commonly known as FireWire.

FireWire has a number of advantages over older I/O interfaces. It is very high speed, low cost, and easy to implement. In fact, FireWire is finding favor not only for computer systems, but also in consumer electronics products, such as digital cameras, DVD players/recorders, and televisions. In these products, FireWire is used to transport video images, which are increasingly coming from digitized sources.

One of the strengths of the FireWire interface is that it

uses serial transmission (bit at a time) rather than parallel. Parallel interfaces, such as SCSI, require more wires, which mean wider, more expensive cables and wider, more expensive 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, computers are getting physically smaller even as they expand in computing power and I/O needs. Handheld and pocket-size computers have little room for connectors yet need high data rates to handle images and video. The intent of FireWire is to provide a single I/O 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.

5.2.2 FireWire Configurations

FireWire uses a daisy-chain configuration, with up to 63 devices connected off a single port. Moreover, up to 1022 FireWire buses can be interconnected using bridges, enabling a system to support as many peripherals as required.

FireWire provides for what is known as hot plugging, which makes it possible to connect and disconnect peripherals without having to power the computer system down or reconfigure the system. Also, FireWire provides for automatic configuration; it is not necessary manually to set device IDs or to be concerned with the relative position of devices. Figure 5.3 shows a simple FireWire configuration. With FireWire, there are no terminations, and the system automatically performs a configuration function to assign addresses. Also note that a FireWire bus need not be a strict daisy chain. Rather, a tree-structured configuration is possible.

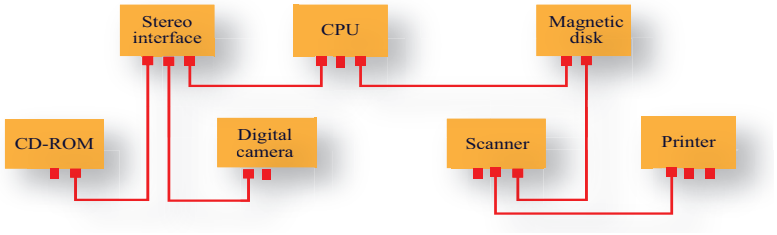


Figure 5.3: Simple FireWire configuration.

An 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 5.4 illustrates 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

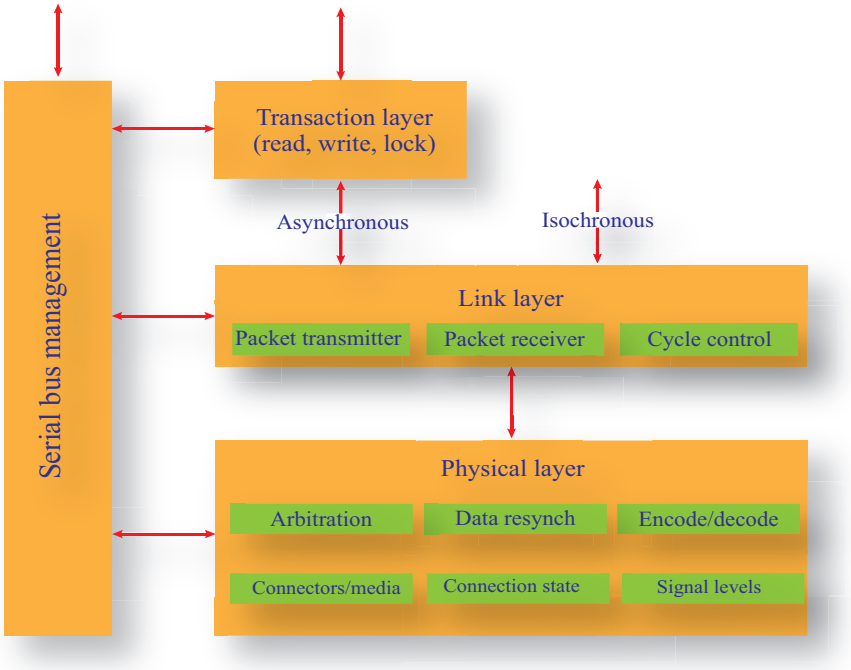


Figure 5.4: FireWire protocol stack.

Physical Layer

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 3200 Mbps are defined. The physical layer converts binary data into electrical signals for various physical 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 arrangement of the nodes on a FireWire bus. A special case of this structure is a linear daisy chain. The physical layer contains 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 topology. Once this configuration is established, the root node acts as a central arbiter and processes requests for bus access in a first-come-first-served fashion. In the case of simultaneous requests, the node 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 aforementioned arbitration method is supplemented by two additional functions: fairness arbitration and urgent arbitration. With fairness arbitration, time on the bus is organized into fairness intervals. At the beginning of an interval, each node sets an arbitration enable flag. During the interval, each node may compete for bus access. Once a node has gained access to the bus, it resets its arbitration enable flag and may not again compete for fair access during this interval. This scheme makes the arbitration fairer, in that it prevents one or more busy high priority devices from monopolizing the bus.

In addition to the fairness scheme, some devices may be configured as having urgent priority. Such nodes may gain control of the bus multiple times during a fairness interval. In essence, a counter is used at each high-priority node that enables the high-priority nodes to control 75% of the available bus time.

Link Layer

The link layer defines the transmission of data in the form of packets.

Two types of transmission are supported:

- *Asynchronous*: A variable amount of data and several bytes of transaction layer information are transferred as a packet to an explicit address and an acknowledgment is returned.
- *Isochronous*: A variable amount of data is transferred in a sequence of fixed size packets transmitted at regular intervals. This form of transmission uses simplified addressing and no acknowledgment.

Asynchronous transmission is used by data that have no fixed data rate requirements. Both the fair arbitration and urgent arbitration schemes may be used for asynchronous transmission. The default method is fair arbitration. Devices that desire a substantial fraction of the bus capacity or have severe latency requirements use the urgent arbitration method. *For example*, a high-speed real-time data collection node may use urgent arbitration when critical data buffers are more than half full.

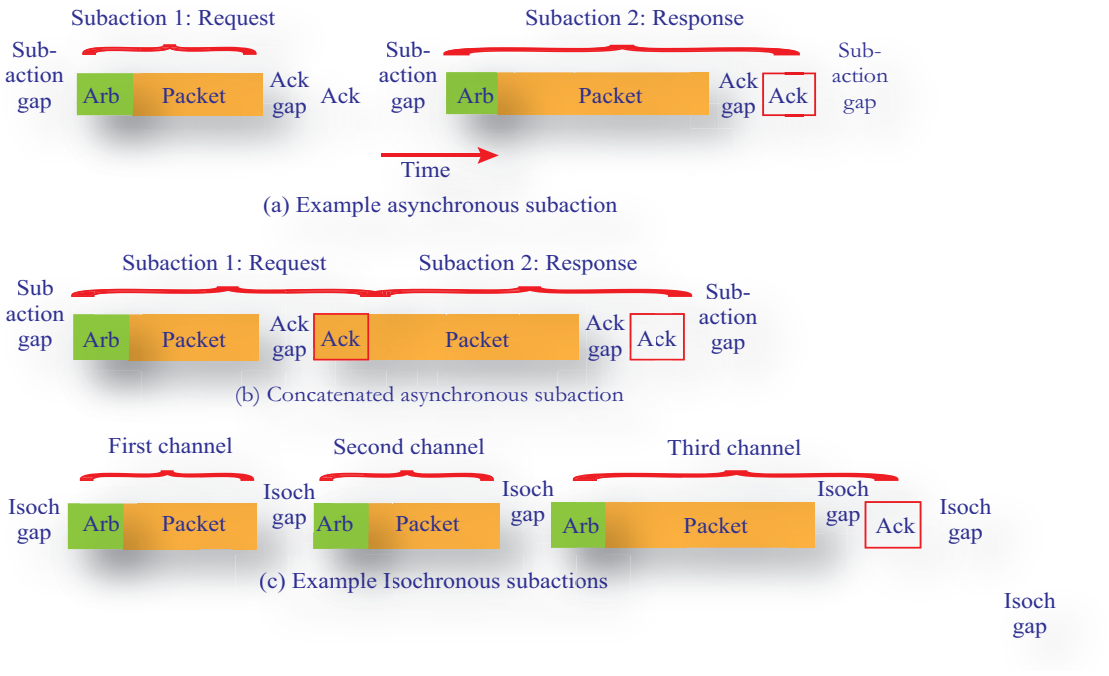


Figure 5.5: FireWire subactions.

Figure 5.5 (a) depicts a typical asynchronous transaction. The process of delivering a single packet is called a subaction. The subaction consists of five time periods:

- *Arbitration sequence*: This is the exchange of signals required to give one device control of the bus.
- *Packet transmission*: Every packet includes a header containing the source and destination 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.
- *Acknowledgment gap*: This is the time delay for the destination to receive and decode a packet and generate an acknowledgment.
- *Acknowledgment*: The recipient of the packet returns an acknowledgment packet with a code indicating the action taken by the recipient.
- *Subaction gap*: This is an enforced idle period to ensure that other nodes on the bus do not begin arbitrating before the acknowledgment packet has been transmitted.

At 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 5.5 (b).

For devices that regularly generate or consume data, such as digital sound or video, isochronous 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 cycle, only isochronous packets may be sent (Figure 5.5 (c). 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 last 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 bus access. Asynchronous sources may then use the bus until the beginning of the next isochronous cycle.

Isochronous packets are labeled with 8-bit channel numbers that are assigned by a dialogue between the two nodes that are to exchange isochronous data. The header, which is shorter than that for asynchronous packets, also includes a data length field and a header CRC.

REMEMBER

The interface must be able to convert serial data to parallel form and vice versa. Because it would be a waste for a processor to be idle while it waits for data from an input device there must be provision for generating interrupts and the corresponding type numbers for further processing by the processor if required.

5.2.3 I/O Ports

In computing, input/output port or I/O port is the communication between an information processing system and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system, and outputs are the signals or data sent from it. The term can also be used as part of an action; to "perform I/O" is to perform an input or output operation. I/O devices are used by a person

(or other system) to communicate with a computer. For instance, a keyboard or a mouse may be an input device for a computer, while monitors and printers are considered output devices for a computer. Devices for communication between computers, such as modems and network cards, typically serve for both input and output.

Keyword

Direct Memory Access (DMA) is a method of transferring data from the random access memory to another part of a computer without occupying the central processing unit.

There are three basic forms of input and output that a typical computer system will use: I/O-mapped I/O, memory-mapped input/output, and **direct memory access (DMA)**. I/O-mapped input/output uses special instructions to transfer data between the computer system and the outside world; memory-mapped I/O uses special memory locations in the normal address space of the CPU to communicate with real-world devices; DMA is a special form of memory-mapped I/O where the peripheral device reads and writes memory without going through the CPU.

The first thing to learn about the input/output subsystem is that I/O in a typical computer system is radically different than I/O in a typical high level programming language. In a real computer system one will rarely find machine instructions that behave like writeln, printf, or even the x86 Get and Put instructions. In fact, most input/output instructions behave exactly like the x86's mov instruction. To send data to an output device, the CPU simply moves that data to a special memory location (in the I/O address space if I/O-mapped input/output or to an address in the memory address space if using memory-mapped I/O). To read data from an input device, the CPU simply moves data from the address (I/O or memory) of that device into the CPU. Other than there are usually more wait states associated with a typical peripheral device than actual memory, the input or output operation looks very similar to a memory read or write operation.

An I/O port is a device that looks like a memory cell to the computer but contains connections to the outside world. An I/O port typically uses a latch rather than a flip-flop to implement the memory cell. When the CPU writes to the address associated with the latch, the latch device captures the data and makes it available on a set of wires external to the CPU and memory system.

5.3 DIRECT MEMORY ACCESS (DMA)

DMA short for direct memory access, a technique for transferring data from main memory to a device without passing it is through the CPU. Computers that have DMA channels can transfer data to and from devices much more quickly than computers without a DMA channel can. This is useful for making quick backups and for real-time applications. Some expansion boards, such as CD-ROM cards, are capable of accessing the computer's DMA channel. When one installs the board, one must specify which DMA channel is to be used, which sometimes involves setting a jumper or DIP switch.

DMA is a method of transferring data from the computer's RAM to another part of the computer without processing it using the CPU. While most data that is input or output from the computer is processed by the CPU, some data does not require processing, or can be processed by another device. In these situations, DMA can save processing time and is a more efficient way to move data from the computer's memory to other devices.

For example, a sound card may need to access data stored in the computer's RAM, but since it can process the data itself, it may use DMA to bypass the CPU. Video cards that support DMA can also access the system memory and process graphics without needing the CPU.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module controls exchange of data between main memory and the I/O device. Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

REMEMBER

In order for devices to use direct memory access, they must be assigned to a DMA channel. Each type of port on a computer has a set of DMA channels that can be assigned to each connected device. For example, a PCI controller and a hard drive controller each have their own set of DMA channels.

DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

5.3.1 Basic Process of DMA

A DMA can generate memory addresses and initiate memory read or write cycles. It contains several processor registers that can be written and read by the CPU. These include a memory address register, a byte county register, and one or more control registers. The control registers specify the I/O port to use, the direction of the transfer (reading from the I/O device or writing to the I/O device), the transfer unit (byte at a time or word at a time), and the number of bytes to transfer in one burst.

To carry out an input, output or memory-to-memory operation, the host processor initializes the DMA controller with a count of the number of words to transfer, and the memory address to use. The CPU then sends commands to a peripheral device to initiate transfer of data. The DMA then provides addresses and read/write control lines to the system memory. Each time a byte of data is ready to be transferred between the peripheral device and memory, the DMA increments its internal address register until the full block of data is transferred. DMA can either occur one byte at a time or all at once in burst mode. If they occur a byte at a time, this can allow the CPU to access memory on alternate bus cycles this is called cycle stealing since the DMA and CPU contend for memory access. In burst mode DMA, the CPU can be put on hold while the DMA transfer occurs and a full block of possibly hundreds or thousands of bytes can be moved. When memory cycles are much faster than processor cycles, an interleaved DMA cycle is possible, where the DMA uses memory while the CPU cannot.

In a bus mastering system, both the CPU and peripherals can be granted control of the memory bus. Where a peripheral can become bus master, it can directly write to system memory without involvement of the CPU, providing memory address and control signals as required. Some measure must be provided to put the processor into a hold condition so that bus contention does not occur.

General Organization of the DMA Controller

The DMA controller mega function is designed for data transfer in different system environments.

Two module types are as follows:

- Type 0

- Type 1

They are provided, and the user can choose the number of each module type. Type 0 modules are designed to transfer data residing on the same bus, and Type 1 modules are designed to transfer data between two different buses. Each module can support up to 4 DMA channels; the mega function supports up to 16 total DMA channels.

Each DMA channel can be programmed for various features, such as transfer size, synchronized and unsynchronized transfer control; transfer priority, interrupt generation, memory and I/O address space, and address change direction. This mega function is designed to work with 32-bit and 64-bit bus systems, including the PCI bus, PowerPC bus, and other CPU host buses. It can also be integrated with other mega functions to form a complete functional block.

5.3.2 DMA Transfer

The connection between the DMA controller and other components in a computer system for DMA transfer is shown in Figure 5.7.

- The DMA request line is used to request a DMA transfer.
- The bus request (BR) signal is used by the DMA controller to request the CPU to relinquish control of the buses.
- The CPU activates the bus grant (BG) output to inform the external DMA that its buses are in a high-impedance state.
- The address bus is used to address the DMA controller and memory at given location.
- The Device select (DS) and register select (RS) lines are activated by addressing the DMA controller.
- The RD and WR lines are used to specify either a read (RD) or write (WR) operation on the given memory location.
- The DMA acknowledge line is set when the system is ready to initiate data transfer.
- The data bus is used to transfer data between the I/O device and memory.
- When the last word of data in the DMA transfer is transferred, the DMA controller informs the termination of the transfer to the CPU by means of the interrupt line.

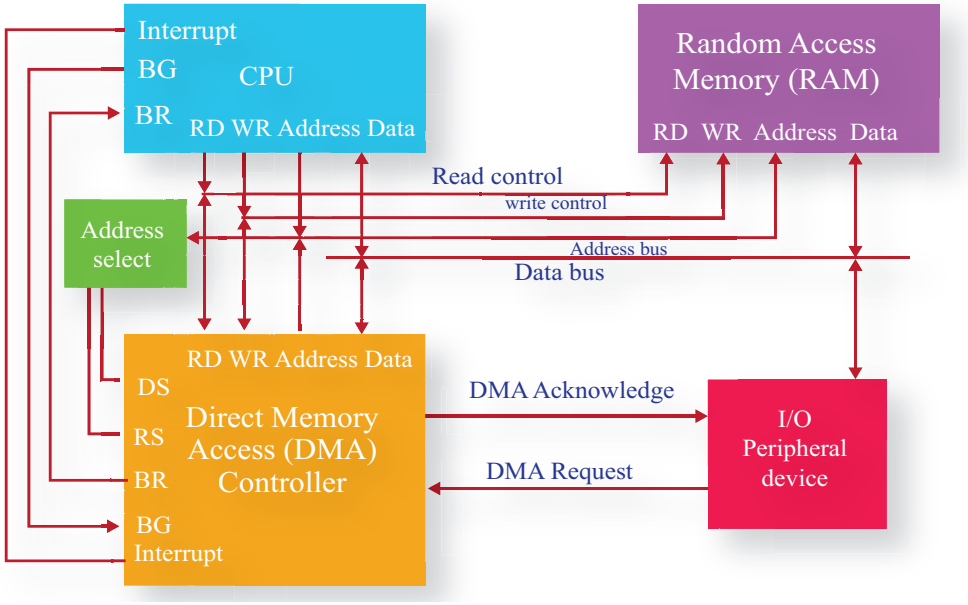


Figure 5.7: DMA transfer.

There are three independent channels for DMA transfers. Each channel receives its trigger for the transfer through a large multiplexer that chooses from among a large number of signals. When these signals activate, the transfer occurs. The DMA controller receives the trigger signal but will ignore it under certain conditions. This is necessary to reserve the memory bus for reprogramming and non-mask able interrupts etc. The controller also handles conflicts for simultaneous triggers. The priorities can be adjusted using the DMA Control Register 1 (DMACTL1). When multiple triggers happen simultaneously, they occur in order of module priority. The DMA trigger is then passed to the module whose trigger activated. The DMA channel will copy the data from the starting memory location or block to the destination memory location or block. There are many variations on this, and they are controlled by the DMA Channel x Control Register (DMAxCTL):

- ***Single Transfer***: Each trigger causes a single transfer. The module will disable itself when DMAxSZ number of transfers has occurred. The DMAxSA and DMAxDA registers set the addresses to be transferred to and from. The DMAxCTL register also allows these addresses to be incremented or decremented by 1 or 2 bytes with each transfer. This transfer halts the CPU.
- ***Block Transfer***: An entire block is transferred on each trigger. The module disables itself when this block transfer is complete. This transfer halts the CPU, and will transfer each memory location one at a time. This mode disables the module when the transfer is complete.

- ***Burst-Block Transfer***: This is very similar to Block Transfer mode except that the CPU and the DMA transfer can interleave their operation. This reduces the CPU to 20% while the DMA is going on, but the CPU will not be stopped altogether. The interrupt occurs when the block has completely transferred. This mode disables the module when the transfer is complete.
- ***Repeated Single Transfer***: Same as Single Transfer mode except that the module is not disabled when the transfer is complete.
- ***Repeated Block Transfer***: the same as Block Transfer mode except that the module is not disabled when the transfer is complete.
- ***Repeated Burst-Block Transfer***: Same as Burst Block Transfer mode except that the module is not disabled when the transfer is complete.

5.3.3 DMA Controller

DMA controller is used to transfer the data between the memory and I/O device.

- The DMA controller needs the usual circuits to communicate with the CPU and I/O device.
- In addition to this, it needs an address register and address bus buffer.
- The address register contains an address of the desired location in memory.
- The word count register holds the number of words to be transferred. The control register specifies the mode of transfer.
- The DMA communicates with the I/O devices through the DMA request and DMA acknowledge line.
- The DMA communicates with the CPU through the data bus and control lines.
- The RD (Read) and WR (write) signals are bidirectional.
- When the BG (Bus Grant) signal are bidirectional.
- When the BG (Bus Grant) signal is 0, the CPU can communicate with the DMA registers through the data bus.
- When BG is 1, the CPU has relinquished the buses. The DMA can communicate directly with the memory.

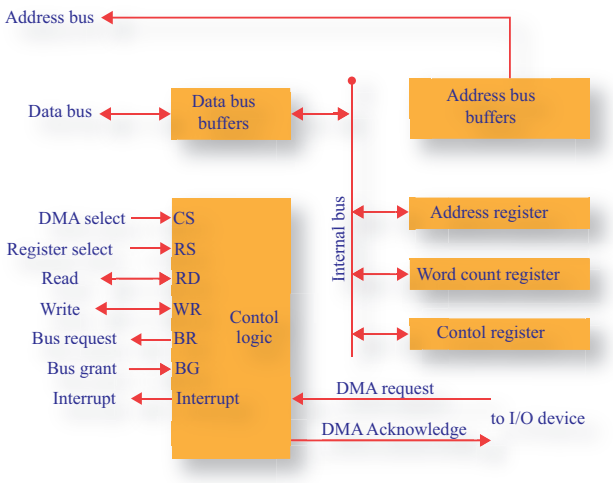


Figure 5.8: DMA controller.

5.3.4 DMA Transfer Cycle

DMA Transfer Cycle is a method of accessing RAM without interfering with the CPU. It is similar to DMA for allowing I/O controllers to read or write RAM without CPU intervention. Clever exploitation of specific CPU or bus timings could permit the CPU to run at full speed without any delay if external devices may sneak in one RAM access to memory not actively participating in the CPU's current activity and complete the operations before any possible CPU conflict. Such systems are nearly dual-port RAM without the expense of high speed RAM. Most systems halt the CPU during the steal, essentially making it DMA by any other name.



A system with separate instruction and data memory banks could allow external devices ONE memory access to the data RAM while the CPU was in op-code-fetch, if both accesses were initiated simultaneously. A MMU is not essential, for the Zilog Z80's M1 line could be used to distinguish instruction from data access, so while the CPU is reading an instruction from instruction-RAM or ROM, the data RAM is available to other devices without disturbing the CPU at all.

CASE STUDY

DEALING WITH VIRTUAL DESKTOP I/O BOTTLENECKS



Networking and storage are two areas often overlooked in a virtual desktop infrastructure project. Cliff Saran looks at what can go wrong.

Tackling Network Overload with WAN Optimization

Networking can make or break a virtual desktop infrastructure (VDI) roll-out, as one business found out. The company, a global law firm, operates a multiprotocol label switching (MPLS) wide area network (WAN) between its three global sites, supporting 7,000 staff. Approximately 75% of traffic on the network was Citrix, 20% was taken by print services, while the remainder was used by web and authentication protocols.

Sometimes the Citrix client on users' desktops would not keep up with their typing due to congestion on the WAN. Given that the cost of upgrading links in certain regions can be around £250,000 per annum, the law firm could not justify buying more network capacity.

Deploying quality of service (QoS) to prioritize Citrix traffic could work, but at the cost of slower print traffic.

The law firm rolled out Silver Peak, a WAN optimization product to tackle the network issues, resulting in a 50% improvement on Citrix performance. Rather than use the Citrix server to compress network traffic, optimization occurs on the Silver Peak appliance, freeing up Citrix so the same server can handle more user sessions.

Silver Peak works by handling error correction before data is transmitted across the WAN. This reduces the need to retransmit TCP/IP packets, which can materially affect the performance of a fast WAN connection.

Flash improves Desktop Virtualization

Storage is the other major bottleneck for virtual desktop users. On a client PC, users can have local storage and access to network drives. When the PC is virtualized, the users will still demand storage, but as the storage is now shared on a virtual machine, physical access to the storage can become overloaded if too many users access the storage device simultaneously.

Flash drives and solid state disks (SSD) offer fast storage to tackle this bottleneck, but they are extremely expensive.

One option is to use tiered storage with three levels: a relatively small flash array, a high-performance hard disk array, and slower storage. Frequently accessed files should migrate to the faster - and more expensive - tier.

Another law firm, US-based Miles & Stockbridge, is using flash memory to ensure storage is not a bottleneck for its VMware View users. In Computer Weekly's sister title SearchDataCenter, Ken Adams, CIO at the law firm, said he uses Dell Equallogic and an EMC fiber channel storage array, and has put local storage on servers to improve performance.

But it was not enough to support virtual desktops, particularly those that run video and dictation devices.

Miles & Stockbridge uses V3 Systems Strato SSDs as a storage layer that sits between the hypervisor and the hardware. Strato remaps the storage stack so that Windows desktops running above it take advantage of the solid-state storage (SSS) with minimal latency.

ROLE MODEL



RAY KURZWEIL: FOUNDER OF SCANNER

Raymond "Ray" Kurzweil (born February 12, 1948) is an American author, computer scientist, inventor, futurist, and is a director of engineering at Google. Aside from futurology, he is involved in fields such as optical character recognition (OCR), text-to-speech synthesis, speech recognition technology, and electronic keyboard instruments. He has written books on health, artificial intelligence (AI), transhumanism, the technological singularity, and futurism. Kurzweil is a public advocate for the futurist and transhumanist movements, as has been displayed in his vast collection of public talks, wherein he has shared his primarily optimistic outlooks on life extension technologies and the future of nanotechnology, robotics, and biotechnology.

Kurzweil was the principal inventor of the first CCD flatbed scanner, the first omni-font optical character recognition, the first print-to-speech reading machine for the blind, the first commercial text-to-speech synthesizer, the Kurzweil K250 music synthesizer capable of simulating the sound of the grand piano and other orchestral instruments, and the first commercially marketed large-vocabulary speech recognition.

Kurzweil received the 1999 National Medal of Technology and Innovation, America's highest honor in technology, from President Clinton in a White House ceremony. He was the recipient of the $500,000 Lemelson-MIT Prize for 2001, the world's largest for innovation. And in 2002 he was inducted into the National Inventors Hall of Fame, established by the U.S. Patent Office. He has received twenty honorary doctorates, and honors from three U.S. presidents. Kurzweil has been described as a "restless genius" by The Wall Street Journal and "the ultimate thinking machine" by Forbes. PBS included Kurzweil as one of 16 "revolutionaries who made America" along with other inventors of the past two centuries. Inc. magazine ranked him #8 among the "most fascinating" entrepreneurs in the United States and called him "Edison's rightful heir".

Kurzweil has authored seven books, five of which have been national bestsellers. The Age of Spiritual Machines has been translated into 9 languages and was the #1 best-selling book on Amazon in science. Kurzweil's book The Singularity Is Near was a New York Times bestseller, and has been the #1 book on Amazon in both science and philosophy. His latest bestseller is How to Create a Mind: The Secret of Human Thought Revealed. Kurzweil speaks widely to audiences public and private and regularly delivers keynote speeches at industry conferences like DEMO, SXSW and TED. His website catalogs his public speaking, publications and media appearances. He maintains the news website KurzweilAI.net, which has over three million readers annually. Kurzweil was the principal inventor of the first CCD flat-bed scanner, the first omni-font optical character recognition, the first print-to-speech reading machine for the blind, the first text-to-speech synthesizer, the first music synthesizer capable of recreating the grand piano and other orchestral instruments, and the first commercially marketed large-vocabulary speech recognition.

Among Kurzweil's many honors, he is the recipient of the National Medal of Technology, was inducted into the National Inventors Hall of Fame, holds twenty honorary Doctorates, and honors from three U.S. presidents. Ray has written five national best-selling books, including New York Times best sellers The Singularity Is Near and How To Create A Mind. He is a Director of Engineering at Google heading up a team developing machine intelligence and natural language understanding.

What Is the Morale Here?

Ray Kurzweil is one of the world's leading inventors, thinkers, and futurists, with a thirty-year track record of accurate predictions. Called "the restless genius" by The Wall Street Journal and "the ultimate thinking machine" by Forbes magazine, Kurzweil was selected as one of the top entrepreneurs by Inc. magazine, which described him as the "rightful heir to Thomas Edison." PBS selected him as one of the "sixteen revolutionaries who made America."

SUMMARY

- An input device sends information to a computer system for processing, and an output device reproduces or displays the results of that processing.
- The term 'peripheral' also does not mean it is not essential to the functioning of the computer.
- Internal peripherals are directly connected to the motherboard using one of the different types of slots on the motherboard. External devices can be connected using a wired connection or a wireless connection.
- The Mouse was originally referred to as an X-Y Position Indicator for a Display System.
- The keyboard is the primary portion of the keyboard that contains all alphanumeric and symbol keys. Today most users use the QWERTY style keyboards.
- Printer is a device that accepts text and graphic output from a computer and transfers the information to paper, usually to standard size sheets of paper.
- Most modern monitors are built using LCD technology and are commonly referred to as flat screen displays.
- An interface is a data path between two separate devices in a computer system. Interface to buses can be classified based on the number of bits that are transmitted at a given time to serial versus parallel ports.
- DMA is a method of transferring data from the computer's RAM to another part of the computer without processing it using the CPU.

KNOWLEDGE CHECK

1. **In 8257 register format, the selected channel is disabled after the terminal count condition is reached when……..**
   a. auto load is set
   b. auto load is reset
   c. TC STOP bit is reset
   d. TC STOP bit is set
2. **Devices, such as monitors and printers that are connected to the computer are called…….**
   a. Processing devices
   b. System devices
   c. Peripheral devices
   d. Input devices
3. **The Delete key, Num Lock key, and Caps Lock key are considered …… keys because their function changes each time you press them.**
   a. Function
   b. toggle
   c. cursor control
   d. Control
4. **The type of monitor that looks like a traditional television set is called a ……. monitor.**
   a. display
   b. cathode-ray tube (CRT)
   c. liquid crystal display (LCD)
   d. flat-panel
5. **…… are large printers used to produce oversize pictures that require precise continuous lines, such as maps and architectural plans?**
   a. Dot-matrix printers
   b. Thermal printers
   c. Multifunction printers
   d. Plotters
6. **A device for converting handwritten impressions into coded characters & positional coordinates for input to a computer is**
   a. Touch panel
   b. Mouse

c. Wand
d. Writing tablet

7. _____ keys are present on the top row of the keyboard.

a. Function
b. Type writer
c. Numeric
d. Navigation

8. The device primarily used to provide hardcopy is the

a. CRT
b. Computer Console
c. Printer
d. Card Reader

9. Which among the device that converts computer output into a form that can be transmitted over a telephone line?

a. Teleport
b. Multiplexer
c. Concentrator
d. Modem

10. A light sensitive device that converts drawing, printed text or other images into digital form is

a. Keyboard
b. Plotter
c. Scanner
d. OMR

REVIEW QUESTIONS

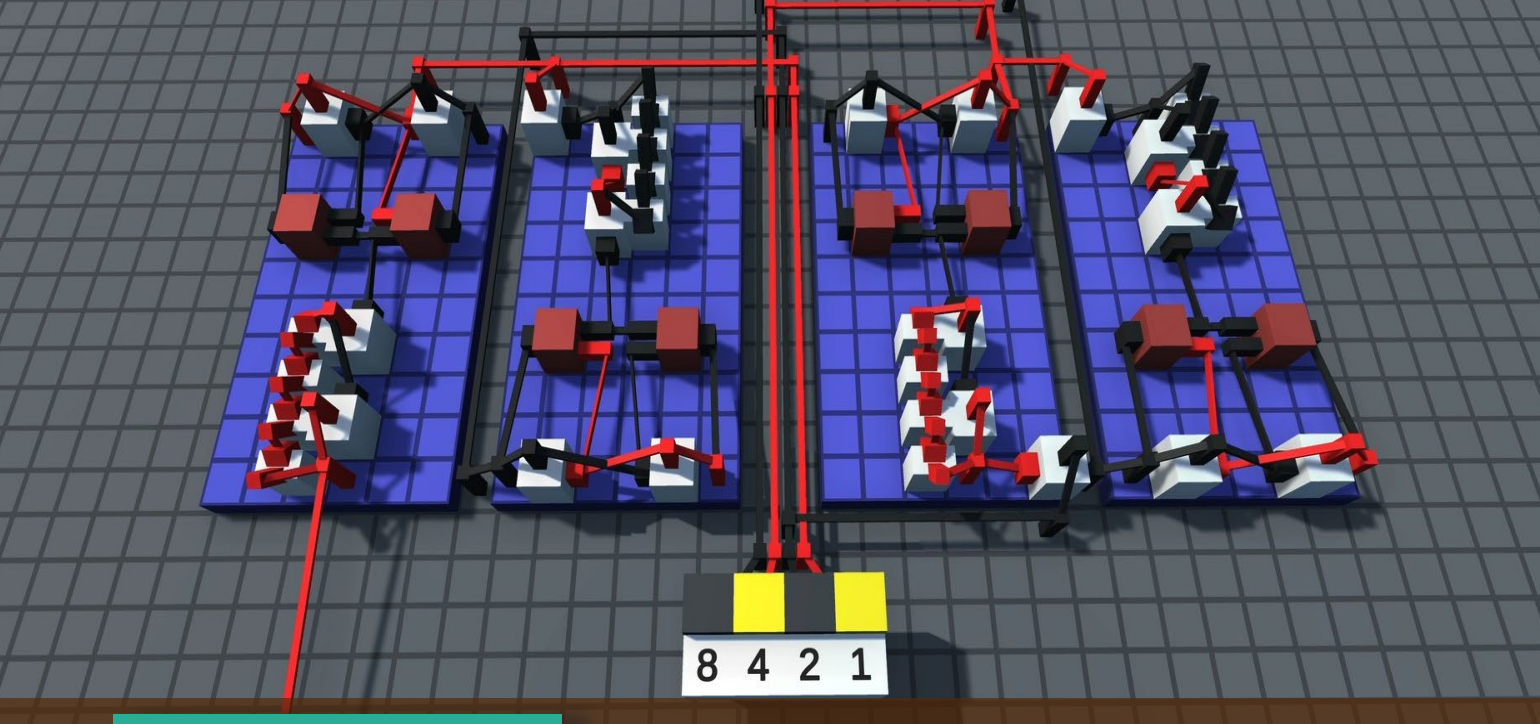
1. Explain the peripheral devices and also define its type.
2. Describe the keyboard and its main keys.
3. Explain about printer and discuss its types.
4. What do you mean by DMA?
5. Explain about I/O interface.

Check Your Result

1. (d)	2. (c)	3. (b)	4. (b)	5. (d)
6. (d)	7. (a)	8. (c)	9. (d)	10. (c)

REFERENCES

1. Abd-El-Barr, Mostafa; Hesham El-Rewini (2005). Fundamentals of Computer Organization and Architecture. John Wiley & Sons. pp. 161–162. ISBN 9780471478331. Archived from the original on 21 December 2016. Retrieved 11 December 2016.
2. Computerhope. 2017. What is a Keyboard?. [online] Available at: <https://www.computerhope.com/jargon/k/keyboard.htm> [Accessed 29 November 2017].
3. Fisher, T., 2017. What Is a Computer Monitor?. [online] Lifewire. Available at: <https://www.lifewire.com/what-is-a-monitor-2618155> [Accessed 29 November 2017].
4. Fisher, T., 2017. What Is a Computer Mouse?. [online] Lifewire. Available at: <https://www.lifewire.com/what-is-a-mouse-2618156> [Accessed 29 November 2017].
5. Null, Linda; Julia Lobur (2006). The Essentials of Computer Organization and Architecture. Jones & Bartlett Learning. p. 185. ISBN 0763737690. Archived from the original on 20 December 2016. Retrieved 11 December 2016.



CHAPTER 6

COMBINATIONAL LOGIC CIRCUITS

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Explain standard combinational circuit blocks
2. Define sequential logic circuits and standard sequential logic circuits
3. Discuss finite state machines and registers

"Computers are composed of nothing more than logic gates stretched out to the horizon in a vast numerical irrigation system."

— Stan Augarten

INTRODUCTION

Logic circuits for digital systems can generally be classified into two categories. One is combinational logic circuits, the other is sequential logic circuits. A combinational logic circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs. In other words, the combinational logic circuits do not have the ability to memorize their past. On the other hand, a sequential logic circuit consists of not only logic gates but also memory elements. Therefore,

the outputs of sequential logic circuits depend not only on the present values of the inputs but also on the input's previous values. In other words, the outputs of a sequential logic circuit are functions of its present inputs and the previous state of its memory elements. Sequential logic circuits will be the topic of your next lab.

No matter how complex a combinational logic circuit might possibly be, it will always be comprised of basic logic gates such as NAND gates, NOR gates, inverters and so on. The logic function that a combinational logic circuit represents is usually expressed using a Boolean function, or a set of Boolean functions. Quite often, the given Boolean functions to be implemented in combinational logic circuits are not simplified (minimized). In order to keep the number of the logic gates that implement the Boolean functions minimum, these Boolean functions must be simplified before they are implemented in discrete logic circuits. In VLSI design other criteria such as propagation delay and interconnect regularity may lead to designs which do not minimize total gates.

Keyword

Logic Circuits include such devices as multiplexers, registers, arithmetic logic units (ALUs), and computer memory, all the way up through complete microprocessors, which may contain more than 100 million gates.

6.1 STANDARD COMBINATIONAL CIRCUIT BLOCKS

Combinational logic circuits are electronic circuits that produce outputs based on the states of the inputs. Unlike in sequential logic circuits, the previous outputs do not partly determine the next outputs. Combinational **logic circuits** are used in various electronic equipment to implement limited intelligence in all sorts of applications.

Unlike sequential logic circuits whose outputs are dependent on both their present inputs and their previous output state giving them some form of memory, the outputs of combinational logic circuits are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time. The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a combinational logic circuit, the output is dependent at all times on the combination of its inputs. So if one of its

inputs condition changes state, from 0-1 or 1-0, so too will the resulting output as by default combinational logic circuits have "no memory", "timing" or "feedback loops" within their design.

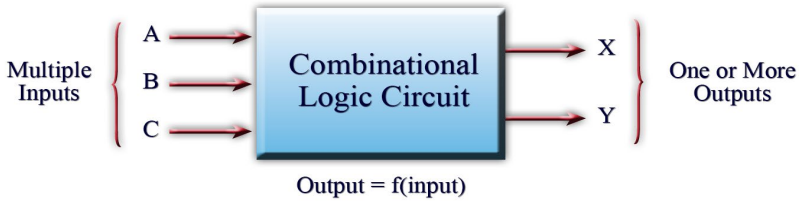


Figure 6.1: Block diagram of combinational logic circuits.

Combinational logic circuits are made up from basic logic NAND, NOR or NOT gates that are "combined" or connected together to produce more complicated switching circuits. These logic gates are the building blocks of Combinational Logic Circuits. An example of a combinational circuit is a decoder, which converts the binary code data present at its input into a number of different output lines, one at a time producing an equivalent decimal code at its output.

Combinational logic circuits can be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classed as "universal" gates.

The three main ways of specifying the function of a combinational logic circuit are:

- *Boolean Algebra*: This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic "1" output.
- *Truth Table*: A truth table defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.
- *Logic Diagram*: This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, which implements the logic circuit.

And all three of these logic circuit representations are shown in figure 6.2.

As combinational logic circuits are made up from individual **logic gates** only, they can also be considered as "decision making circuits" and combinational logic is about combining logic gates together to process two or more signals in order to produce

Keyword

Logic Gates are an idealized or physical device implementing a Boolean function; that is, it performs a logical operation on one or more logical inputs, and produces a single logical output.

at least one output signal according to the logical function of each logic gate. Common combinational circuits made up from individual logic gates that carry out a desired application include multiplexers, de-multiplexers, encoders, decoders, full and half adders etc.

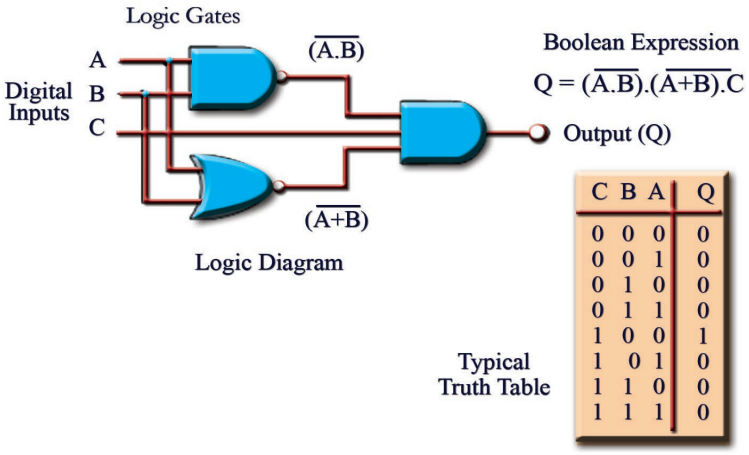


Figure 6.2: Combinational logic circuit representations.

Classification of Combinational Logic

One of the most common uses of combinational logic is in multiplexer and de-multiplexer type circuits.

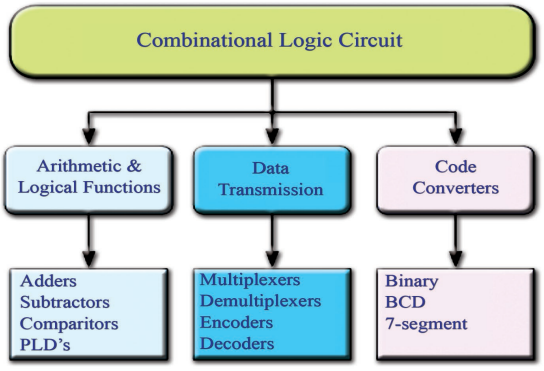


Figure 6.3: Classification of combinational logic.

Here, multiple inputs or outputs are connected to a common signal line and logic gates are used to decode an address to select a single data input or output switch. A multiplexer consists of two separate components, a logic decoder and some solid state switches.

6.1.1 Encoder

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has n number of input lines and m number of output lines. An encoder produces an m bit **binary code** corresponding to the digital input number. The encoder accepts an n input digital word and converts it into an m bit another digital word.

Keyword

Binary Code represents text or computer processor instructions using the binary number system's two binary digits, 0 and 1.

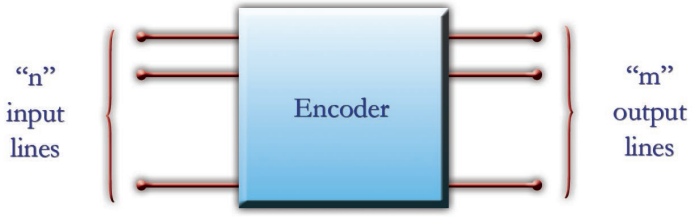


Figure 6.4: Block diagram of encoder.

Examples of encoders are following:

- Priority encoders
- Decimal to BCD encoder
- Octal to binary encoder
- Hexadecimal to binary encoder

Priority Encoder

This is a special type of encoder. Priority is given to the input lines. If two or more input line is 1 at the same time, then the input line with highest priority will be considered. There are four inputs D_0, D_1, D_2, D_3 and two outputs Y_0, Y_1. Out of the four inputs D_3 has the highest priority and D_0 has the lowest priority. That means if $D_3 = 1$ then Y_1 $Y_1 = 11$ irrespective of the other inputs. Similarly if $D_3 = 0$ and $D_2 = 1$ then Y_1 $Y_0 = 10$ irrespective of the other inputs.

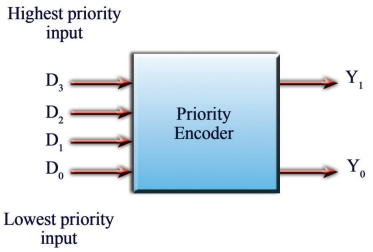


Figure 6.5: Block diagram of priority encoder.

Table 6.1: Truth table of priority encoder

Highest	Inputs		**Lowest**	**Outputs**	
D_3	D_2	D_1	D_0	Y_0	Y_1
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

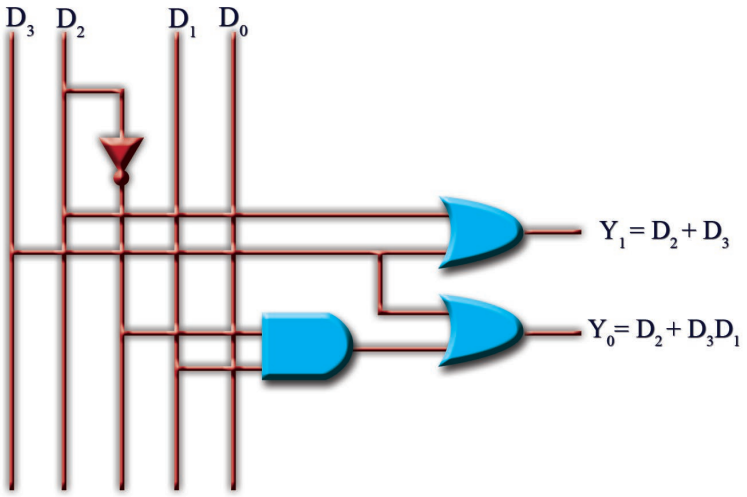


Figure 6.6: Logic circuit of priority encoder.

6.1.2 Decoder

A decoder is a combinational circuit. It has an input and to a maximum m = 2n outputs. Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.



Figure 6.7: Block diagram of decoder.

Examples of decoders are following:

- Code converters
- BCD to seven segment decoders
- Nixie tube decoders
- Relay actuator

2 to 4 Line Decoder

The block diagram of 2 to 4 line decoder is shown in the Figure 6.8. A and B are the two inputs where D through D is the four outputs. Truth table explains the operations of a decoder. It shows that each output is 1 for only a specific combination of inputs.

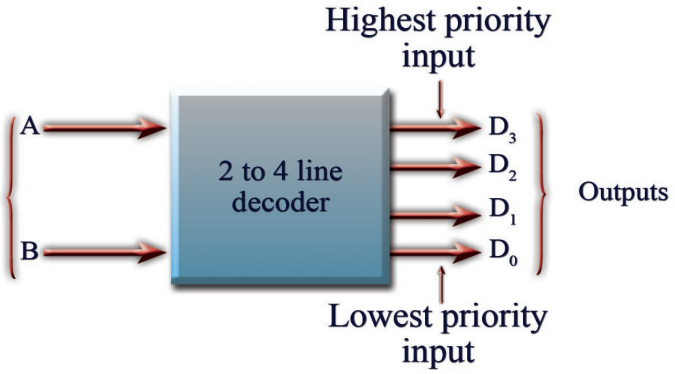


Figure 6.8: Block diagram of 2 to 4 line decoder.

Table 6.2: Truth table of 2 to 4 line decoder

Inputs		Outputs			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

6.1.3 Multiplexer

A multiplexer, sometimes referred to as a multiplexor or simply a mux, is an electronic device that selects from several input signals and transmits one or more output signals. In its simplest form, a multiplexer will have two signal inputs, one control input and one output. One *example* of an analog multiplexer is the source control on a home stereo unit that allows the user to choose between the audio from a **compact disc (CD)** player, digital versatile disc (DVD) player and cable television line.

Keyword

Compact disc (CD) is a digital optical disc data storage format.

Multiplexers also are used in building digital semiconductors such as central processing units (CPUs) and graphics controllers. In these applications, the number of inputs is generally a multiple of two, the number of outputs is either one or relatively small multiple of two, and the number of control signals is related to the combined number of inputs and outputs. *For example*, a two-input, one-output multiplexer requires only one control signal to select the input, and a 16-input, four-output multiplexer requires four control signals to select the input and two to select the output.

Types of multiplexers also are used in communications. A telephone network is an example of a very large virtual multiplexer that is built from many smaller, discrete ones. Instead of having a direct connection from every telephone to every other telephone which would be physically impossible the network muxes individual telephone lines onto a small number of wires as calls are placed. At the receiving end, a demultiplexer, or demux, chooses the correct destination from the many possible destinations by applying the same principle in reverse.

There are more complex forms of multiplexers. Time-division multiplexers, *for example*, have the same input/output characteristics as other multiplexers, but instead of having control signals, they alternate between all possible inputs at precise time intervals. By taking turns in this manner, many inputs can share one output. This technique is commonly used on long-distance phone lines, allowing many individual phone calls to be spliced together without affecting the speed or quality of any individual call. Time-division multiplexers generally are built as **semiconductor** devices, or chips, but they also can be built as optical devices for fiber optic applications.

Even more complex are code-division multiplexers. Using mathematical techniques developed during World War II for cryptographic purposes, they have since found application in modern code division multiple access (CDMA) cellular networks. These semiconductor devices work by assigning each input a unique complex mathematical code. Each input applies its code to the signal that it receives, and all signals are simultaneously sent to the output. At the receiving end, a demux performs the inverse mathematical operation to extract the original signals.

Multiplexer is a special type of combinational circuit. There are n-data inputs, one output and m select inputs with 2m = n. It is a digital circuit which selects one of the n data inputs and routes it to the output. The selection of one of the n inputs is done by the selected inputs. Depending on the digital code applied at the selected inputs, one out of n data sources is selected and transmitted to the single output Y. E is called the strobe or enable input which is useful for the cascading. It is generally an active low terminal, which means it will perform the required operation when it is low.

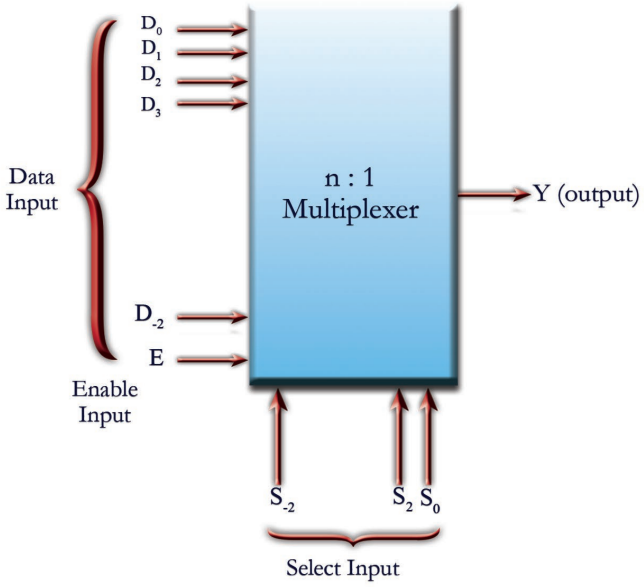


Figure 6.9: Block diagram of multiplexer.

Multiplexers come in multiple variations these are follows:

- 2:1 multiplexer
- 4:1 multiplexer
- 16:1 multiplexer
- 32:1 multiplexer

Keyword

Semiconductor material has an electrical conductivity value between a conductor, such as copper, and an insulator, such as glass.

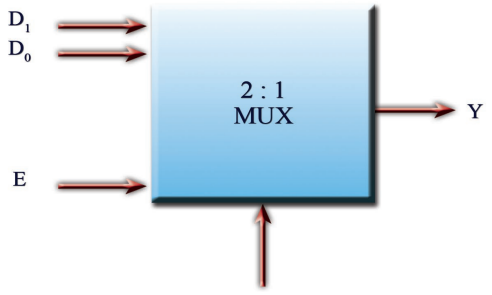


Figure 6.10: Block diagram of 2:1 multiplexer

Table 6.3: Truth table of 2:1 multiplexer

Enable	Select	Output
E	S	Y
0	X	0
1	0	D_0
1	1	D_1

X=Don't care

6.1.4 Demultiplexer

A demultiplexer, or "demux," is a piece of equipment that takes a single signal carrying multiple payloads and splits it up into several streams. They are often used in telecommunications to carry signals over long distances. A demultiplexer is the opposite of a multiplexer. A multiplexer, or "mux," takes signals from many sources and puts them into a single signal. That signal is transmitted wherever it needs to go then disassembled at the other end by a demultiplexer. To picture how a demultiplexer works, it can be compared to the operation of a waterwheel.

It is a single device with many little buckets on it that collect the water in each one. The water is then transported and then dropped off by each separate bucket. The intake of water in multiplexing, and the release on the other side is demultiplexing. It is a way of getting many pieces of traffic from one place to another without having to have separate wires for each.



In telecommunications, a multiplexer's "buckets" are timeslots. If someone is calling from an office in New York, for example, to another office in New Jersey, the call is sent from the office to the local telephone exchange. The exchange sees that the caller is calling New Jersey, so sends him to the New Jersey multiplexer. This device is directly connected to the piece of fiber that goes to New Jersey.

The call is given a timeslot, 36 in this *example*, and sent on its way over fiber to New Jersey. As well as the call data the person's voice another piece of information is put there by the multiplexer telling the other end that, while the call is in progress, anything that comes in on timeslot 36 is for this specific call. This process of multiplexing and demultiplexing takes place hundreds of times a second for each and every conversation that takes place over longer distances. If someone was to call across the country, her conversation, still in timeslot 36, would be inspected at every major exchange and sent on its way until it reached the city she was calling. That exchange would see the call was for someone connected to it and "break out" the call and demultiplex it. The process is so refined, and the equipment so fast, that though the call data might be inspected 20 or 30 times, the caller perceives no delay in the call.

A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs, m select input. At a time only one output line is selected by the select lines and the input is transmitted to the selected output line. A de-multiplexer is equivalent to a single pole multiple way switch as shown in figure 6.11.

Demultiplexers come in multiple variations:

- 1:2 demultiplexer
- 1:4 demultiplexer

- 1:16 demultiplexer
- 1:32 demultiplexer

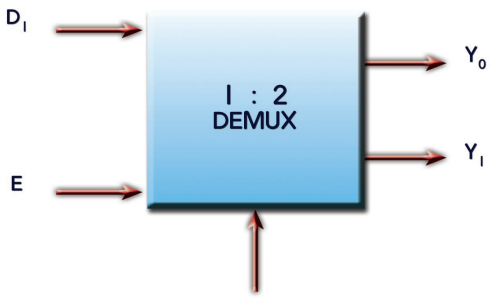


Figure 6.11: Block diagram of 1:2 demultiplexer.

Table 6.4: Truth table of 1:2 demultiplexer

Enable	Select	Output	
E	S	Y_0	Y_1
0	X	0	0
1	0	0	D_{in}
1	1	D_{in}	0

Where X is don't care

6.1.5 Adders

In computers and other kinds of processors, adders are used not only in the arithmetic logic unit (ALU), but also in other parts of the processor, where they are used to calculate addresses, table indices, and similar operations.

Half Adder

Half adder is a combinational logic circuit with two inputs and two output. The half adder circuit is designed to add two single bit binary numbers A and B. It is the basic building block for addition of two single bit numbers. This circuit has two outputs carry and sum.



Figure 6.12: Block diagram of half adder.

Table 6.5: Truth table of half adder

Input		Output	
A	B	S	C
0	0	0	0
0	1	1	0
0	1	1	0
1	1	0	1

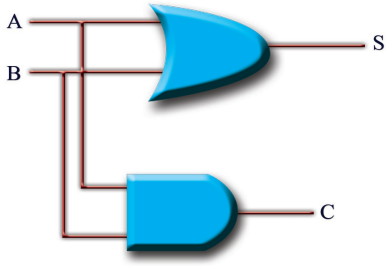


Figure 6.13: Circuit diagram of half adder.

Full Adder

Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.



In CPU design, the instruction decoder is the part of the CPU that converts the bits stored in the instruction register – or, in CPUs that have microcode, the microinstruction – into the control signals that control the other parts of the CPU.

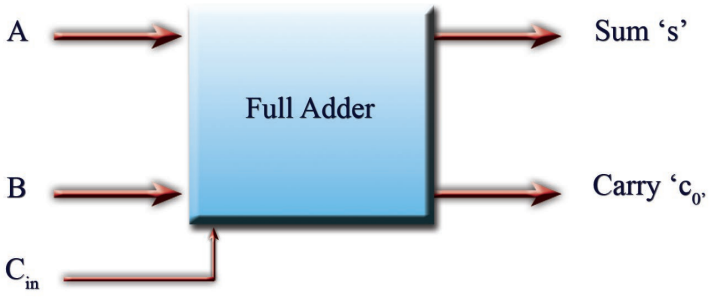


Figure 6.14: Block diagram of full adder.

Table 6.6: Truth table of full adder

Inputs			Output	
A	B	C_{in}	S	C_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

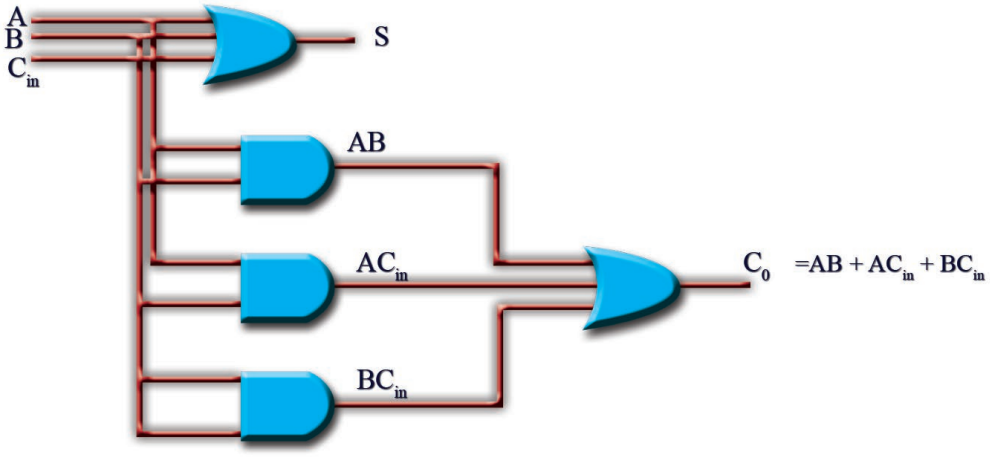


Figure 6.15: Circuit diagram of full adder.

N-Bit Parallel Adder

The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical you need to add binary numbers which are much longer than just one bit. To add two n-bit binary numbers you need to use the n-bit parallel adder. It uses a number of full adders in cascade. The carry output of the previous full adder is connected to carry input of the next full adder.

4 Bit Parallel Adder

In the block diagram, A_0 and B_0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its C_{in} has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder are shown in fig. The four bit parallel adder is a very common logic circuit.

Keyword

Digital Circuit, represent signals by discrete bands of analog levels, rather than by a continuous range.

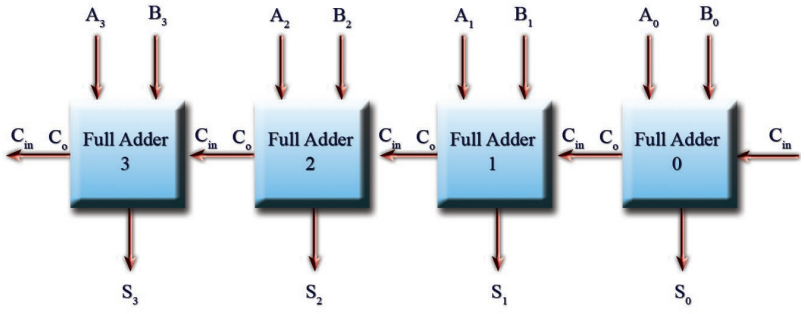


Figure 6.16: Block diagram of 4 bit parallel adder.

6.2 SEQUENTIAL LOGIC CIRCUITS

Sequential logic (SL) in **digital circuit** theory is the set of rules and implementations of circuits that rely on the current and past events of logic states and transitions to determine present logic states. Knowing about combinational logic (CL), the set of rules and implementation of circuits that rely on the actual logic levels, reveals the key points in sequential logic. Logic levels for binary computing usually refer to high or low. In positive logic, 1 is high and 0 is low. Logic circuits are made up of gates that may have one or more inputs and usually only one output. A simple CL gate is known as the buffer and

the inverter or NOT gate. The buffer output is always the same as the input, but the inverter output is always not the input. Other gates used in CL include the AND gate, NAND gate, and NOR gate. The AND gate outputs a 1 only if both inputs are 1. The NAND gate and NOR gate are, respectively, an AND gate and an OR gate, each with an inverter at the output.

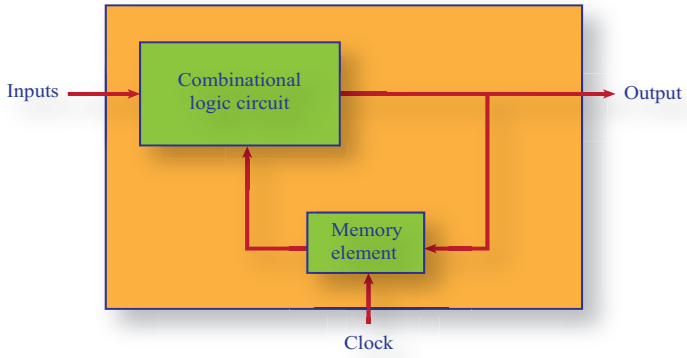


Figure 6.17: Block diagram of sequential logic circuit.

Sequential logic uses latches that lock the output levels based on previous output levels and current input levels. Latches are usually built using two partner gates, which are either two NAND or NOR gates. The gates of these latches, or flip-flops, are locked into one of two states by the gate outputs that are fed back to the input of the partner gate. By changing the levels on the free inputs of the gates, a reversal of logic level is achieved. Sequential logic analysis involves both observing the initial output levels and observing the change in output levels based on the change in input levels.

In binary counters, there is edge detection circuitry in the clock input for each binary digit (bit) latch. Counters usually use a positive-edge detect for normal count-up. For instance, an 8-bit counter uses 8-bit latches.

Sequential logic makes use of cascaded bit latches to produce an asynchronous (async) digital counter. When a bit from the less-significant-bit (LSB) latch is made to clock the more significant bit (MSB), it is known as an async counter. In async, latches clock each other at slightly different times, while synchronous (sync) logic clocks all latches simultaneously. The async counter will suffer a maximum total ripple delay equal to one latch ripple delay multiplied by the number of bits in the counter. In sync logic, the bit latches in a digital counter are clocked simultaneously, thus the total ripple delay is equal to one latch ripple delay for any number of bits in the counter.

6.2.1 Flip-Flops

In the case of a latch with control input, the when the control is enabled, the latch is in transparent mode, that is its output changes its state according to the set and reset states. During this transparent mode, it is still possible for the outputs to become unstable if both set and reset inputs change from asserted to deasserted states simultaneously.

Avoid this problem using master-slave flip-flop arrangement:

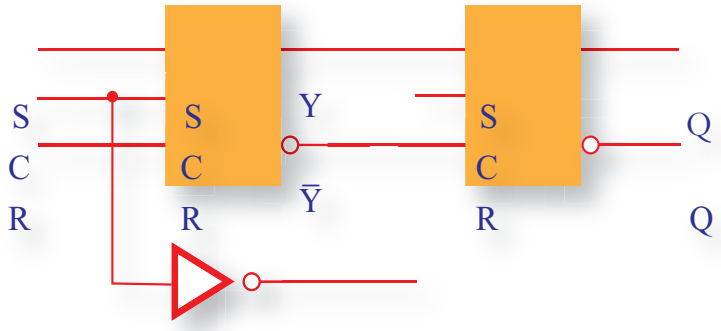


Figure 6.18: S-R flip-flops representation.

It is impossible (well, almost) for both the set and reset inputs to be asserted for the slave latch on the right when its control is enabled because the control of the master latch on the left is disabled at this time.

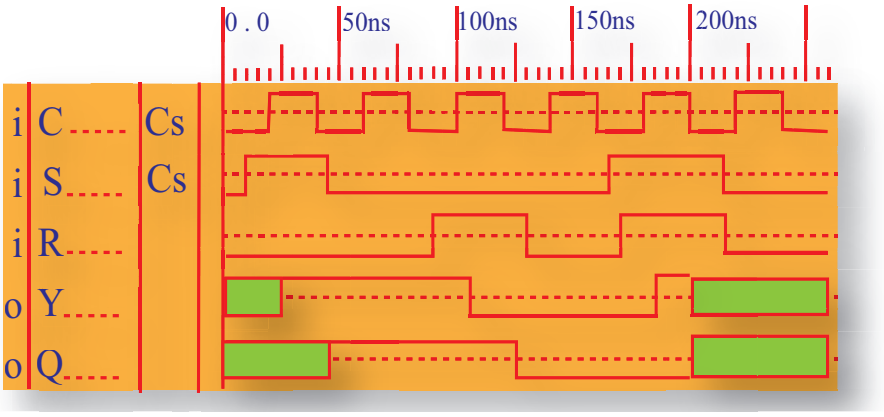


Figure 6.19: Machine cycle of S-R flip-flops representation.

Asynchronous Latches

Latch is a storage element maintains binary state indefinitely (as long as power is applied), until directed by an input signal to switch to its other state.

The simplest latch also referred to as the SR latch neither has two inputs and two outputs and can be constructed from two NOR gates.

Keyword

Asynchronous Circuit is a sequential digital logic circuit which is not governed by a clock circuit or global clock signal.

The circuit is considered to be asynchronous if it does not employ a periodic clock signal C to synchronize its internal changes of state. Therefore the state changes occur in direct response to signal changes on primary (data) input lines, and different memory elements can change state at different times.

In asynchronous sequential circuits the inputs are levels and there are no clock pulses; the inputs events drive the circuit.

In general, an **asynchronous circuit** does not need the precise timing control supported by flip-flops. It may therefore contain latches rather than flip-flops. In many cases, an asynchronous circuit simply relies on the propagation delays of its component gates and connections, combined with the circuit's feedback structure, to implement its memory functions.

Synchronous Flip-Flops

In synchronous circuits the input are pulses (or levels and pulses) with certain restrictions on pulse width and circuit propagation delay. Therefore synchronous circuits can be divided into clocked sequential circuits and uncloaked or pulsed sequential circuits.

In a clocked sequential circuit which has flip-flops or, in some instances, gated latches, for its memory elements there is a (synchronizing) periodic clock connected to the clock inputs of all the memory elements of the circuit, to synchronize all internal changes of state.

Hence the operation of the entire circuit is controlled and synchronized by the periodic pulses of the clock.

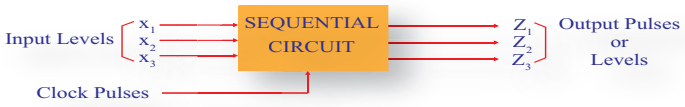


Figure 6.20: Block diagrams of clocked sequential circuit.

On the other hand in an unclocked or pulsed sequential circuit, such a clock is not present. Pulse mode circuits require two consecutive transitions between 0 and 1 - that is a 0-pulseor a 1 pulse to alter the circuit's state. A pulse -mode circuit is designed to respond to pulses of certain duration; the constant signals between the pulses are null or spacer signals, which do not affect the circuit's behavior.



Figure 6.21: Block diagrams of pulsed sequential circuit.

- *Pulse Outputs*: For pulsed sequential circuits these occur only for the duration of the respective input pulse and in some cases for duration considerably less. For clocked sequential circuits these outputs occur for the duration of the clock pulse.
- *Level Outputs*: These change state at the start of the respective input or clock pulse and remain in that state until the next state of output is required.

It is basically S-R latch using NAND gates with an additional enable input. It is also called as level triggered SR-FF. For this circuit in output will take place if and only if the enable input (E) is made active. In short this circuit will operate as an S-R latch if E= 1 but there is no change in the output if E = 0.

REMEMBER

A requirement of synchronous sequential circuits is that the duration of the activating pulse or clock pulse should be sufficiently low in value that the pulse (or clock) has disappeared by the time the secondary's (the flip-flops outputs) have taken on their new value; otherwise the circuit will change state again. This means that the storage elements (flip-flops) should be edge-triggered devices (for example: D-type flip-flop, the JK flip-flop and their derivatives).

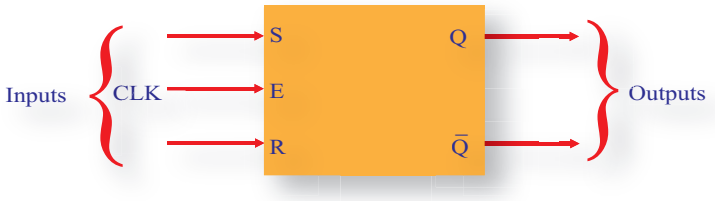


Figure 6.22: Block diagram of S-R latch.

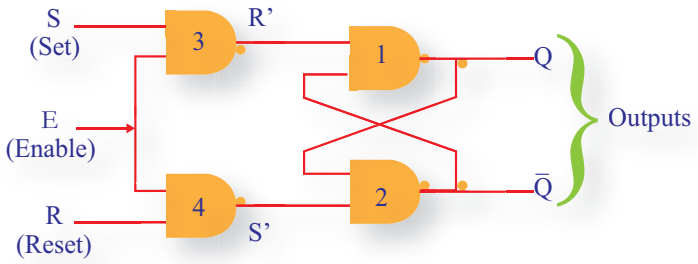


Figure 6.23: Circuit diagram of S-R latch.

Table 6.7: Truth table of S-R latch

Inputs			Outputs		Comments
E	S	R			
1	0	0	Q	Q-	No change
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	x	x	Indeterminate

6.3 FINITE STATE MACHINES

A finite-state machine (FSM) or finite-state automaton (plural: automata), or simply a state machine, is a mathematical abstraction sometimes used to design digital logic

or computer programs. It is a behavior model composed of a finite number of states, transitions between those states, and actions, similar to a flow graph in which one can inspect the way logic runs when certain conditions are met. It has finite internal memory, an input feature that reads symbols in a sequence, one at a time without going backward; and an output feature, which may be in the form of a user interface, once the model is implemented. The operation of an FSM begins from one of the states (called a start state), goes through transitions depending on input to different states and can end in any of those available, however only a certain set of states mark a successful flow of operation (called accept states).

Keyword

Parsing is the process of analyzing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar.

Finite-state machines can solve a large number of problems, among which are electronic design automation, communication protocol design, parsing and other engineering applications. In biology and artificial intelligence research, state machines or hierarchies of state machines are sometimes used to describe neurological systems and in linguistics to describe the grammars of natural languages.

6.3.1 State Transition Graphs

A state transition graph (STG) is a directed graph representing the dynamical behavior of a Regulatory Graph. Nodes of this graph represent possible states of the model, assigning a value to each component. Arcs of the STG represent transitions from one state to another (i.e. change of value for one or several components).

Another way to deal with large state transition graphs is to analyze the graph by means of formal verification techniques, implemented in computer tools called model checkers. *Examples* of the kind of properties one might want to test are "It is possible for a state with x_a concentration equal to t_xa2 to occur" or "If the system is in the state corresponding to the initial conditions zero_a <= x_a < t_a1, zero_b <= x_b < t_b, and zero_c <= x_c < t_c1, then it necessarily reaches a steady state".

The use of the model-checking tools involves few steps:

- The specification of the property to be verified by defining atomic propositions and formulating so-called temporal-logic formulas;

- The actual verification of the property by a model checker.

6.4 REGISTERS

Processor operations mostly involve processing data. This data can be stored in memory and accessed from thereon. However, reading data from and storing data into memory slows down the processor, as it involves complicated processes of sending the data request across the control bus and into the memory storage unit and getting the data through the same channel.

To speed up the processor operations, the processor includes some internal memory storage locations, called registers.

The registers store data elements for processing without having to access the memory. A limited number of registers are built into the processor chip.

Processor Registers

There are ten 32-bit and six 16-bit processor registers in IA-32 architecture. The registers are grouped into three categories:

- General registers
- Control registers
- Segment registers

The general registers are further divided into the following groups:

- Data registers
- Pointer registers
- Index registers
- Data Registers

Four 32-bit data registers are used for arithmetic, logical and other operations. These 32-bit registers can be used in three ways:

- As complete 32-bit data registers: EAX, EBX, ECX, EDX.
- Lower halves of the 32-bit registers can be used as four 16-bit data registers: AX, BX, CX and DX.
- Lower and higher halves of the four 16-bit registers can be used as eight 8-bit data registers: AH, AL, BH, BL, CH, CL, DH, and DL.

Data Registers

Some of these data registers have specific use in arithmetical operations.

AX is the primary accumulator; it is used in input/output and most arithmetic instructions. *For example*, in multiplication operation, one operand is stored in EAX or AX or AL register according to the size of the operand.

BX is known as the base register as it could be used in indexed addressing.

CX is known as the count register as the ECX, CX registers store the loop count in iterative operations.

DX is known as the data register. It is also used in input/output operations. It is also used with AX register along with DX for multiply and divides operations involving large values.

Pointer Registers

The pointer registers are 32-bit EIP, ESP and EBP registers and corresponding 16-bit right portions IP, SP and BP. There are three categories of pointer registers:

Instruction Pointer (IP): The 16-bit IP register stores the offset address of the next instruction to be executed. IP in association with the CS register (as CS:IP) gives the complete address of the current instruction in the code segment.

Stack Pointer (SP): The 16-bit SP register provides the offset value within the program stack. SP in association with the SS register (SS:SP) refers to be current position of data or address within the program stack.

Base Pointer (BP): The 16-bit BP register mainly helps in referencing the parameter variables passed to a subroutine. The address in SS register is combined with the offset in BP to get the location of the parameter. BP can also be combined with DI and SI as base register for special addressing.

Index Registers

The 32-bit index registers ESI and EDI and their 16-bit rightmost portions SI and DI are used for indexed addressing and sometimes used in addition and subtraction.

There are two sets of index pointers:

Source Index (SI) - it is used as source index for string operations

Destination Index (DI): it is used as destination index for string operations.

Control Registers

The 32-bit instruction pointer register and 32-bit flags register combined are considered as the control registers.

Many instructions involve comparisons and mathematical calculations and change the status of the flags and some other conditional instructions test the value of these status flags to take the control flow to other location.

The common flag bits are:

- *Overflow Flag (OF)*: indicates the overflow of a high-order bit (leftmost bit) of data after a signed arithmetic operation.
- *Direction Flag (DF)*: determines left or right direction for moving or comparing string data. When the DF value is 0, the string operation takes left-to-right direction and when the value is set to 1, the string operation takes right-to-left direction.
- *Interrupt Flag (IF)*: determines whether the external interrupts like keyboard entry, etc., are to be ignored or processed. It disables the external interrupt when the value is 0 and enables interrupts when set to 1.
- *Trap Flag (TF)*: allows setting the operation of the processor in single-step mode. The DEBUG program one used sets the trap flag, so one could step through the execution one instruction at a time.
- *Sign Flag (SF)*: shows the sign of the result of an arithmetic operation. This flag is set according to the sign of a data item the arithmetic operation. The sign is indicated by the high-order of leftmost bit. A positive result clears the value of SF to 0 and negative result sets it to 1.
- *Zero Flag (ZF)*: indicates the result of an arithmetic or comparison operation. A nonzero result clears the zero flag to 0, and a zero result sets it to 1.
- *Auxiliary Carry Flag (AF)*: contains the carry from bit 3 to bit 4 an arithmetic operation; used for specialized arithmetic. The AF is set when a 1-byte arithmetic operation causes a carry from bit 3 into bit 4.
- *Parity Flag (PF)*: indicates the total number of 1-bits in the result obtained from an arithmetic operation. An even number of 1-bits clears the parity flag to 0 and an odd number of 1-bits set the parity flag to 1.
- *Carry Flag (CF)*: contains the carry of 0 or 1 from a high-order bit (leftmost) after an arithmetic operation. It also stores the contents of last bit of a shift or rotates operation.

6.5 Standard Sequential Logic Circuits

In digital circuit theory, sequential logic is a type of logic circuit whose output depends not only on the present input but also on the history of the input. This is in contrast to combinational logic, whose output is a function of, and only of, the present input. In other words, sequential logic has storage (memory) while combinational logic does not. Sequential logic is therefore used to construct some types of computer memory, other types of delay and storage elements, and finite state machines. Most practical computer circuits are a mixture of combinational and sequential logic.

There are two types of finite state machine that can be built from sequential logic circuits:

Moore Machine: The output depends only on the internal state. (Since the internal state only changes on a clock edge, the output only changes on a clock edge too).

Mealy Machine: the output depends not only on the internal state, but also on the inputs.

Depending on regulations of functioning, digital circuits are divided into synchronous and asynchronous. In accordance with this, behavior of devices obeys synchronous or asynchronous logic.

REMEMBER

The main advantage of synchronous logic is its simplicity. Every operation in the circuit must be completed inside a fixed interval of time between two clock pulses, called a 'clock cycle'. As long as this condition is met, the circuit is guaranteed to be reliable.

Synchronous Sequential Logic

Nearly all sequential logic today is 'clocked' or 'synchronous' logic: there is a 'clock' signal, and all internal memory (the 'internal state') changes only on a clock edge. The basic storage element in sequential logic is the flip-flop.

Synchronous logic also has two main disadvantages, as follows:

- The clock signal must be distributed to every flip-flop in the circuit. As the clock is usually a high-frequency signal, this distribution consumes a relatively large amount of power and dissipates much heat. Even the flip-flops that are doing nothing consume a small amount of power, thereby generating waste heat in the chip.

- The maximum possible clock rate is determined by the slowest logic path in the circuit, otherwise known as the critical path. This means that every logical calculation, from the simplest to the most complex, must complete in one clock cycle. One way around this limitation is to split complex operations into several simple operations, a technique known as 'pipelining'. This technique is prominent within microprocessor design, and helps to improve the performance of modern processors.

Asynchronous Sequential Logic

Asynchronous sequential logic expresses memorizing effect by fixing moments of time, when digital device changes its state. These moments are represented not in explicit form, but taking into account principle "before/after" in temporal relations of logical values.

For asynchronous logic it is sufficient to determine a sequence of switching's irrespective of any connections of the corresponding moments with real or virtual time.

Theoretical apparatus of sequential logic consists of mathematical instruments of sequential and venjunction as well as of logic-algebraic equations on their basis.

6.5.1 Counters

Counters are circuits that cycle through a specified number of states.

There are two types of counters:

- synchronous (parallel) counters
- asynchronous (ripple) counters

Ripple counters allow some flip-flop outputs to be used as a source of clock for other flip-flops.

Synchronous counters apply the same clock to all flip-flops.

Synchronous Counters

The asynchronous counters are simple, but not very fast. If a counter with a larger number of bits is constructed in this manner, then the delays caused by the cascaded clocking scheme may become too long to meet the desired performance requirements. You can build a faster counter by clocking all flip-flops at the same time, using the approach described as:

Synchronous Counter with T Flip-Flops

The contents of a three-bit up-counter for eight consecutive clock cycles, assuming that the count is initially 0 as shown in Figure 6.24. Observing the pattern of bits in each row of the table, it is apparent that bit Q_0 changes on each clock cycle. Bit Q_1 changes only when

$Q_0 = 1$. Bit Q_2 changes only when both Q_1 and Q_0 are equal to 1. In general, for an n-bit up-counter, a given flip-flop changes its state only when all the preceding flip-flops are in the state Q = 1. Therefore, if one uses T flip-flops to realize the counter, then the T inputs are defined as:

$T_0 = 1$

$T_1 = Q_0$

$T_2 = Q_0Q_1$

$T3 = Q_0Q_1Q_2$

.

.

$Tn = Q_0Q_1 \cdot \cdot \cdot Q_{n-1}$

An example of a four-bit counter based on these expressions is given in Figure 6.24 (a).

Clock cycle	Q_2 Q_1 Q_0
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1
8	0 0 0

Q_1 Changes

Q_2 Changes

Instead of using AND gates of increased size for each stage, which may lead to fan-in problems, one uses a factored arrangement, as shown in the figure 6.24. This arrangement does not slow down the response of the counter, because all flip-flops change their states after a propagation delay from the positive edge of the clock. A

change in the value of Q_0 may have to propagate through several AND gates to reach the flip-flops in the higher stages of the counter, which requires a certain amount of time.

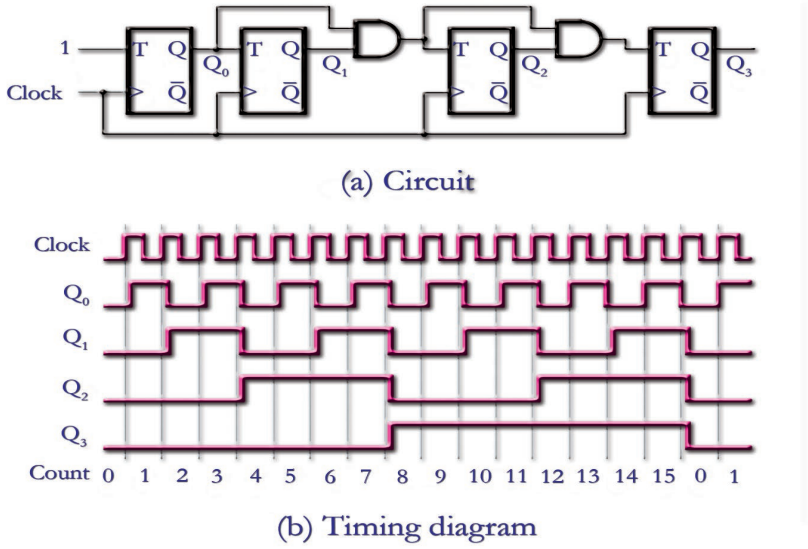


Figure 6.24: A four-bit synchronous up-counter.

This time must not exceed the clock period. Actually, it must be less than the clock period minus the setup time for the flip-flops.

Figure 6.24 (b) gives a timing diagram. It shows that the circuit behaves as a modulo-16 up-counter. Because all changes take place with the same delay after the active edge of the clock signal, the circuit is called a synchronous counter.

Asynchronous Counters

The simplest counter circuits can be built using T flip-flops because the toggle feature is naturally suited for the implementation of the counting operation.

Up-Counter with T Flip-Flops

Figure 6.25 (a) gives a three-bit counter capable of counting from 0 to 7. The clock inputs of the three flip-flops are connected in cascade. The T input of each flip-flop is connected to a constant 1, which means that the state of the flip-flop will be reversed (toggled) at each positive edge of its clock. One is assuming that the purpose of this

circuit is to count the number of pulses that occur on the primary input called Clock. Thus the clock input of the first flip-flop is connected to the Clock line.

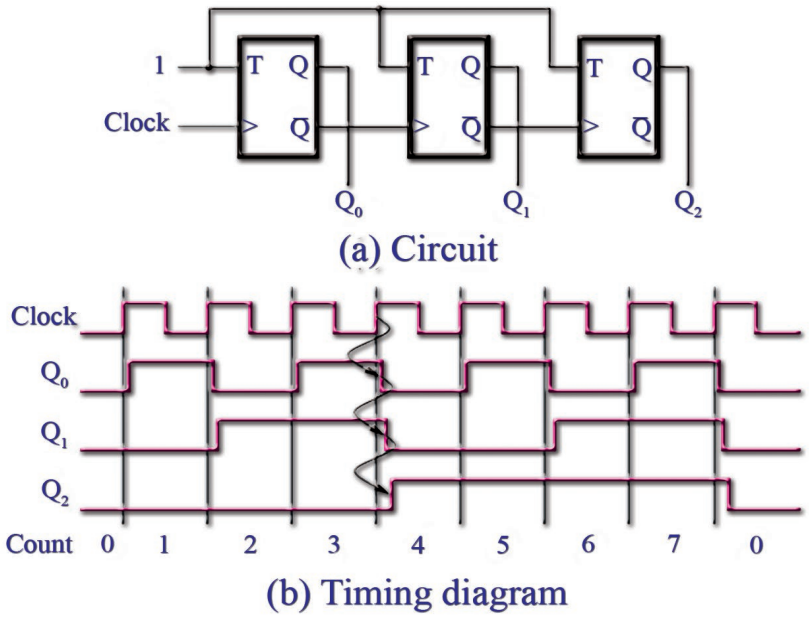


Figure 6.25: A three-bit up-counter.

The other two flip-flops have their clock inputs driven by the Q output of the preceding flip-flop. Therefore, they toggle their state whenever the preceding flip-flop changes its state from Q = 1 to Q = 0, which results in a positive edge of the Q signal. Figure 6.25 (b) shows a timing diagram for the counter. The value of Q_0 toggles once each clock cycle. The change takes place shortly after the positive edge of the Clock signal. The delay is caused by the propagation delay through the flip-flop. Since the second flip-flop is clocked by Q_0, the value of Q_1 changes shortly after the negative edge of the Q_0 signal. Similarly, the value of Q_2 changes shortly after the negative edge of the Q_1 signal. If one looks at the values $Q_2Q_1Q_0$ as the count, then the timing diagram indicates that the counting sequence is 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, and so on. This circuit is a modulo-8 counter. Because it counts in the upward direction, one calls it an up-counter. The counter in Figure 6.25 (a) has three stages, each comprising a single flip-flop. Only the first stage responds directly to the clock signal; one says that this stage is synchronized to the clock. The other two stages respond after an additional delay. *For example*, when count = 3, the next clock pulse will cause the count to go to 4. As indicated by the arrows in the timing diagram in Figure 6.25 (b), this change requires the toggling of the states of all three flip-flops. The change in Q_0 is

observed only after a propagation delay from the positive edge of Clock. The Q_1 and Q_2 flip-flops have not yet changed; hence for a brief time the count is $Q_2Q_1Q_0$ = 010. The change in Q_1 appears after a second propagation delay, at which point the count is 000. Finally, the change in Q_2 occurs after a third delay, at which point the stable state of the circuit is reached and the count is 100.

6.5.2 Shift Registers

The Shift Register is another type of sequential logic circuit that is used for the storage or transfer of data in the form of binary numbers. This sequential device loads the data present on its inputs and then moves or "shifts" it to its output once every clock cycle, hence the name "shift register".

A shift register basically consists of several single bit "D-Type Data Latches", one for each data bit, either a logic "0" or a "1", connected together in a serial or daisy-chain arrangement so that the output from one data latch becomes the input of the next latch and so on.

The data bits may be fed in or out of a shift register serially, that is one after the other from either the left or the right direction, or all together at the same time in parallel.

The number of individual data latches required to make up a single shift register device is usually determined by the number of bits to be stored with the most common being 8-bits (one byte) wide constructed from eight individual data latches.

The shift register is used for data storage or data movement and are used in calculators or computers to store data such as two binary numbers before they are added together, or to convert the data from either a serial to parallel or parallel to serial format. The individual data latches that make up a single shift register are all driven by a common clock (Clk) signal making them synchronous devices.

Shift register IC's are generally provided with a clear or reset connection so that they can be "SET" or "RESET" as required.

Generally, shift registers operate in one of four different modes with the basic movement of data through a shift register being:

- *Serial-in to Parallel-out (SIPO):* The register is loaded with serial data, one bit at a time, with the stored data being available in parallel form.
- *Serial-in to Serial-out (SISO):* The data is shifted serially "IN" and "OUT" of the register, one bit at a time in either a left or right direction under clock control.
- *Parallel-in to Serial-out (PISO)*: The parallel data is loaded into the register simultaneously and is shifted out of the register serially one bit at a time

under clock control.

- *Parallel-in to Parallel-out (PIPO)*: The parallel data is loaded simultaneously into the register, and transferred together to their respective outputs by the same clock pulse.

The effect of data movement from left to right through a shift register can be presented graphically as:

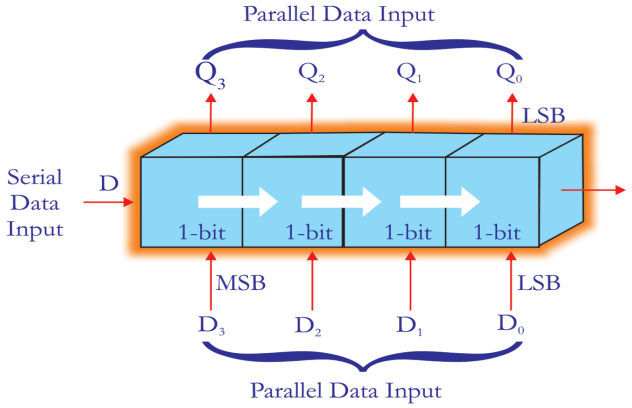


Figure 6.26: Data movement in shift registers.

Also, the directional movement of the data through a shift register can be either to the left, (left shifting) to the right, (right shifting) left-in but right-out, (rotation) or both left and right shifting within the same register thereby making it bidirectional.

Serial-in to Parallel-out (SIPO) Shift Register

4-bit Serial-in to Parallel-out Shift Register

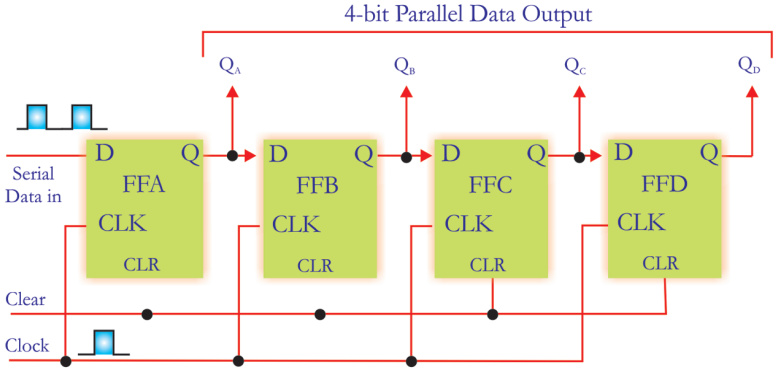


Figure 6.27: Serial-in to parallel-out (SIPO) shift register.

The operation is as follows. Let us assume that all the flip-flops (FFA to FFD) have just been RESET (CLEAR input) and that all the outputs Q_A to Q_D are at logic level "0" i.e., no parallel data output.

If a logic "1" is connected to the DATA input pin of FFA then on the first clock pulse the output of FFA and therefore the resulting Q_A will be set HIGH to logic "1" with all the other outputs still remaining LOW at logic "0". Assume now that the DATA input pin of FFA has returned LOW again to logic "0" giving us one data pulse or 0-1-0.

The second clock pulse will change the output of FFA to logic "0" and the output of FFB and Q_B HIGH to logic "1" as its input D has the logic "1" level on it from Q_A. The logic "1" has now moved or been "shifted" one place along the register to the right as it is now at Q_A.

When the third clock pulse arrives this logic "1" value moves to the output of FFC (Q_C) and so on until the arrival of the fifth clock pulse which sets all the outputs Q_A to Q_D back again to logic level "0" because the input to FFA has remained constant at logic level "0".

The effect of each clock pulse is to shift the data contents of each stage one place to the right, and this is shown in the table until the complete data value of 0-0-0-1 is stored in the register. This data value can now be read directly from the outputs of Q_A to Q_D.

Then the data has been converted from a serial data input signal to a parallel data output. The truth table and waveforms show the propagation of the logic "1" through the register from left to right as follows.

Table 6.7: Clock pulse representation of SIPO shift register

Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

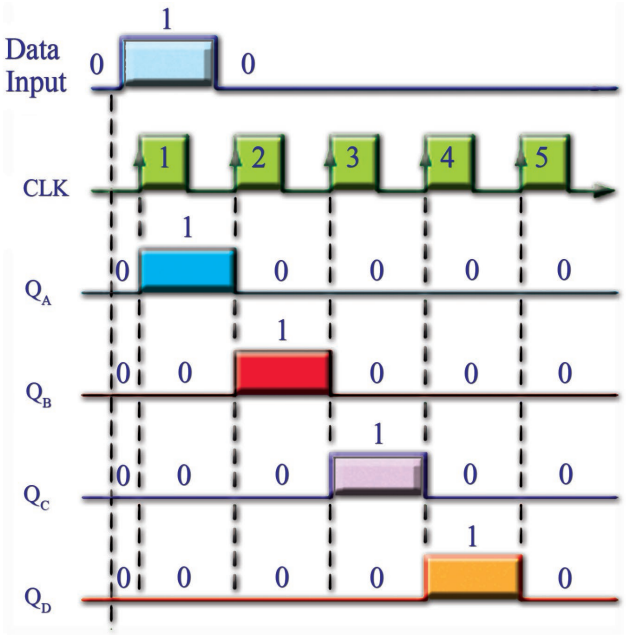


Figure 6.28: Basic data movement through a shift register.

After the fourth clock pulse has ended the 4-bits of data (0-0-0-1) are stored in the register and will remain there provided clocking of the register has stopped. In practice the input data to the register may consist of various combinations of logic "1" and "0". Commonly available SIPO IC's include the standard 8-bit 74LS164 or the 74LS594.

Serial-in to Serial-out (SISO) Shift Register

This shift register is very similar to the SIPO, except were before the data was read directly in a parallel form from the outputs Q_A to Q_D, this time the data is allowed to flow straight through the register and out of the other end. Since there is only one output, the DATA leaves the shift register one bit at a time in a serial pattern, hence the name Serial-in to Serial-Out Shift Register or SISO.

The SISO shift register is one of the simplest of the four configurations as it has only three connections, the serial input (SI) which determines what enters the left hand flip-flop, the serial output (SO) which is taken from the output of the right hand flip-flop and the sequencing clock signal (Clk). The logic circuit diagram shows a generalized serial-in serial-out shift register.

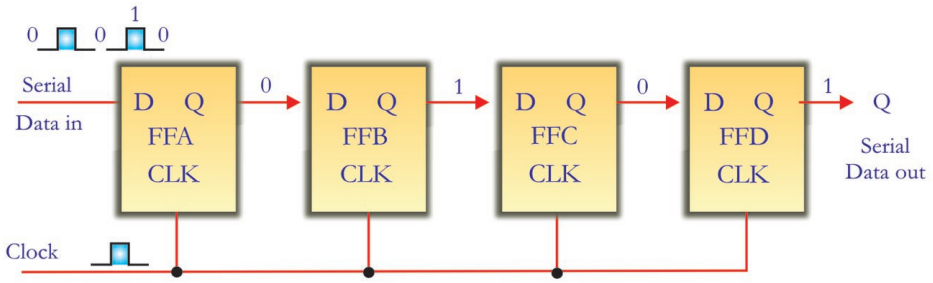


Figure 6.29: 4-bit serial-in to serial-out shift register.

One may think what the point of a SISO shift register is if the output data is exactly the same as the input data. Well this type of shift register also acts as a temporary storage device or as a time delay device for the data, with the amount of time delay being controlled by the number of stages in the register, 4, 8, 16 etc. or by varying the application of the clock pulses. Commonly available IC's include the 74HC595 8-bit Serial-in to Serial-out Shift Register all with 3-state outputs.

Parallel-in to Serial-out (PISO) Shift Register

The parallel-in to serial-out shift register acts in the opposite way to the serial-in to parallel-out. The data is loaded into the register in a parallel format in which all the data bits enter their inputs simultaneously, to the parallel input pins P_A to P_D of the register. The data is then read out sequentially in the normal shift-right mode from the register at Q representing the data present at P_A to P_D.

This data is outputted one bit at a time on each clock cycle in a serial format. It is important to note that with this system a clock pulse is not required to parallel load the register as it is already present, but four clock pulses are required to unload the data.

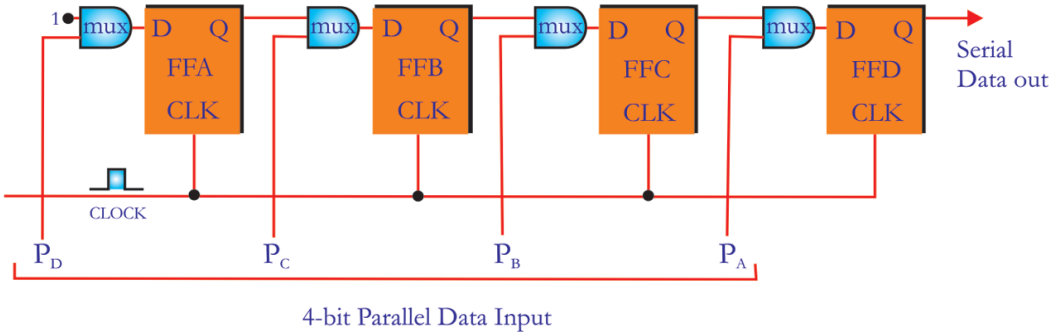


Figure 6.30: 4-bit Parallel-in to serial-out shift register.

As this type of shift register converts parallel data, such as an 8-bit data word into serial format, it can be used to multiplex many different input lines into a single serial DATA stream which can be sent directly to a computer or transmitted over a communications line. Commonly available IC's include the 74HC166 8-bit Parallel-in/Serial-out Shift Registers.

DID YOU KNOW ?

One of the first known examples of a shift register was in the Colossus, a code-breaking machine of the 1940s.

Parallel-in to Parallel-out (PIPO) Shift Register

The final mode of operation is the Parallel-in to Parallel-out Shift Register. This type of register also acts as a temporary storage device or as a time delay device similar to the SISO configuration. The data is presented in a parallel format to the parallel input pins PA to PD and then transferred together directly to their respective output pins QA to QA by the same clock pulse. Then one clock pulse loads and unloads the register.

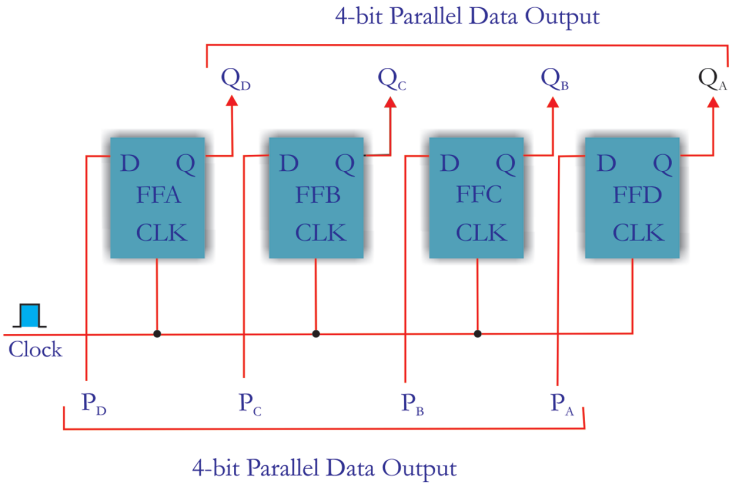


Figure 6.31: 4-bit parallel-in to parallel-out shift register.

The PIPO shift register is the simplest of the four configurations as it has only three connections, the parallel input (PI) which determines what enters the flip-flop, the parallel output (PO) and the sequencing clock signal (Clk).

Similar to the Serial-in to Serial-out shift register, this type of register also acts as a temporary storage device or as a time delay device, with the amount of time delay being varied by the frequency of the clock pulses. Also, in this type of register

there are no interconnections between the individual flip-flops since no serial shifting of the data is required.

Universal Shift Register

Today, there are many high speed bi-directional "universal" type shift registers available such as the TTL 74LS194, 74LS195 or the CMOS 4035 are available as a 4-bit multi-function devices that can be used in either serial-to-serial, left shifting, right shifting, serial-to-parallel, parallel-to-serial, and as a parallel-to-parallel multifunction data register, hence the name "Universal".

These universal shift registers can perform any combination of parallel and serial input to output operations but require additional inputs to specify desired function and to pre-load and reset the device.

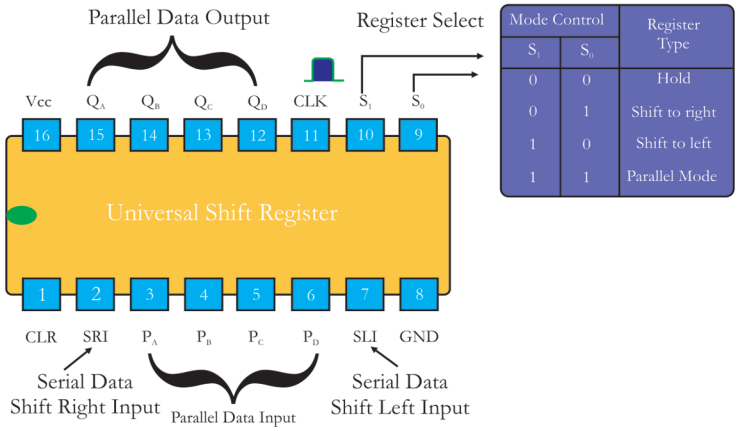
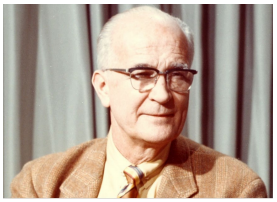


Figure 6.32: 4-bit universal shift register 74LS194.

Universal shift registers are very useful digital devices. They can be configured to respond to operations that require some form of temporary memory, delay information such as the SISO or PIPO configuration modes or transfer data from one point to another in either a serial or parallel format. Universal shift registers are frequently used in arithmetic operations to shift data to the left or right for multiplication or division.

ROLE MODEL



WILLIAM SHOCKLEY: INVENTOR OF TRANSISTOR

William Schockley received a B.Sc. degree from the California Institute of Technology in 1932. He received a Ph.D. degree from Massachusetts Institute of Technology in 1936. His doctoral thesis was on the energy band structure of sodium chloride entitled "Calculations of Wave Functions for Electrons in Sodium Chloride Crystals."

William Schockley Main Awards:

- 1946 Medal for Merit for work with the War Department
- 1952 Morris Leibmann Memorial Prize of the Institute of Radio Engineers
- 1953 Oliver E. Buckley Solid State Physics Prize of the American Physical Society
- 1954 Cyrus B. Comstock Award of the National Academy of Sciences
- 1956 Nobel Prize for Physics with John Bardeen and Walter Brattain
- 1963 Holley Medal of the American Society of Mechanical Engineers
- 1974 Induction into the National Inventor's Hall of Fame

William Shockley was born in London, England, in 1910. He was the son of American parents, William Hillman Shockley a mining engineer, and Mary Shockley a former deputy mineral surveyor. In 1913, the family returned to the United States.

In 1936, William Shockley began working at Bell Telephone Laboratories, where he remained until 1955, becoming the Director of the Transistor Physics Department. John Bardeen, Walter Brattain and William Shockley co-invented the transistor in 1947 at Bell Labs. In 1950, Shockley published a book "Electrons and Holes in Semiconductors".

The Silicon in Silicon Valley

After William Shockley left Bell, he founded and served as Director of the Shockley Transistor Corporation in Mountain View, California, doing R&D and managing the production of new transistors and other semiconductor devices. Shockley was a significant force behind the widespread commercialization of the transistor.

Early transistors were made with germanium because it was easier to prepare in pure form but it was a rare element. Silicon could operate at higher temperatures, and was more common and cheap. However, silicon processing was at that time difficult to do. William Shockley began the R&D at his laboratory to invent a silicon transistor; however, he abandoned the project. Several of his employees abandoned Shockley and formed their own company Fairchild Semiconductor Corporation where the silicon based transistor was invented.

The Colorful Side of William Shockley

William Shockley was not always an easy man to work with and was considered to be paranoid by some. At Shockley Laboratory, the boss once made all his employees take a polygraph test to settle a minor dispute. His behavior was probably one reason why so many of his employees quit.

In 1963, Shockley retired from business for an academic position (Emeritus Professor of Electrical Engineering) at Stanford University. At Stanford he formulated his theory of what he termed dysgenics. William Shockley advocated that individuals with IQs below 100 be paid to undergo voluntary sterilization, supported a Nobel sperm bank for geniuses, and suggested that certain racial groups might be smarter than others. Dysgenics as a science was never well received; however, William Shockley considered dysgenics his greatest achievement.

In 1989, William Shockley died in San Francisco of prostate cancer.

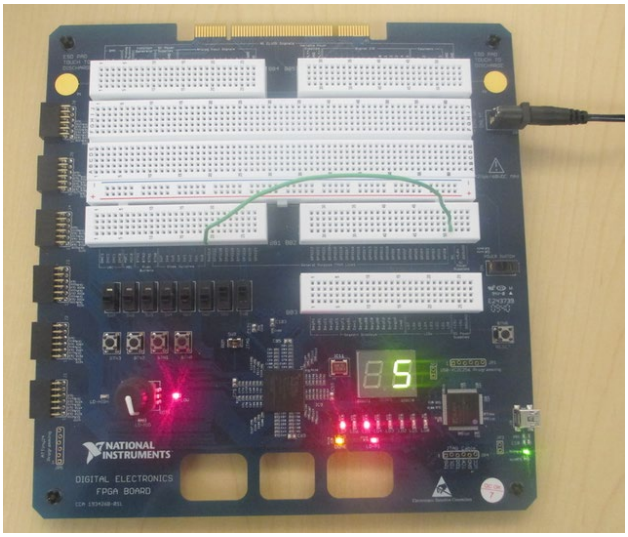
What Is The Moral Here?

American physicist and Nobel prize winner, William Shockley co-invented the transistor, an influential little invention that changed the course of history for computers and electronics in a big way.

CASE STUDY

THREE STATES SYNCHRONOUS COUNTER

Introduction



A synchronous machine is one where all flip flops are hooked up to the same clock.

If one model behavior using a Mealy or Moore machine, and follow the steps for implementing it, one gets synchronous sequential circuits.

We are going to implement a synchronous counter. We contrast this synchronous counter with a asynchronous counter (a counter where all flip flops were not hooked to the same clock.

This is an unusual counter. Normally, if one have a standard up-counter, it counts up starting from $(0)^k$ (zero, repeated k times), all the up way to $(1)^k$ (one, repeated k times), and then starts back at $(0)^k$ and counts up again. (We could implement a down counter than counts backwards, too).

However, there is no reason we can't implement a counter that counts 00, 01, 10, and then back to 00. There might be no reason why we should do so, either.

Nevertheless, since we can implement an up-counter of this kind let us do so. It is a good exercise.

Here is the 3-state up counter, as a Moore machine.

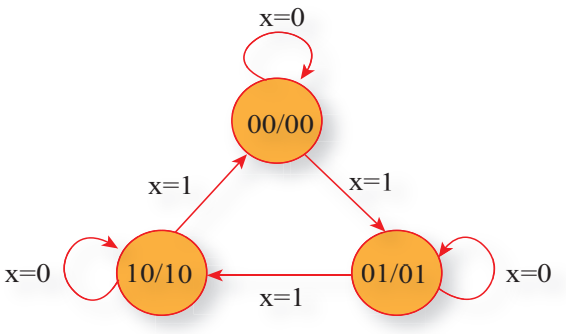


Figure 1: 3-state up counter representation.

Starting Out

First, we fill out the state variables and the input variables. Those serve as the state transition table inputs.

Table 1: State transition table

Row	q_1	q_0	x
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Then the Outputs (for a Moore machine)

Since this is a Moore machine, we should fill out the outputs next. We look at the state (the value left of the slash in 00/00) and fill out the corresponding output (the value right of the slash in 00/00).

For counters, it is almost always the case that the state label and the outputs are identical.

Table 2: State label and the outputs.

Row	q_1	q_0	x	q_1^+	q_0^+	z_1	z_0
0	0	0	0			0	0
1	0	0	1			0	0
2	0	1	0			0	1
3	0	1	1			0	1
4	1	0	0			1	0
5	1	0	1			1	0
6	1	1	0			d	d
7	1	1	1			d	d

They are not completely identical because there is no state 11. We always place d in the rows of states that do not exist.

Then the Next States

This is easy for a counter. When the input is a 0, then the state does not change. When it is a 1, it increments.

Table 3: Next states change

Row	q_1	q_0	x	q_1^+	q_0^+	z_1	z_0
0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0
2	0	1	0	0	1	0	1
3	0	1	1	1	0	0	1
4	1	0	0	1	0	1	0
5	1	0	1	0	0	1	0
6	1	1	0	d	d	d	d
7	1	1	1	d	d	d	d

Picking Flip Flops

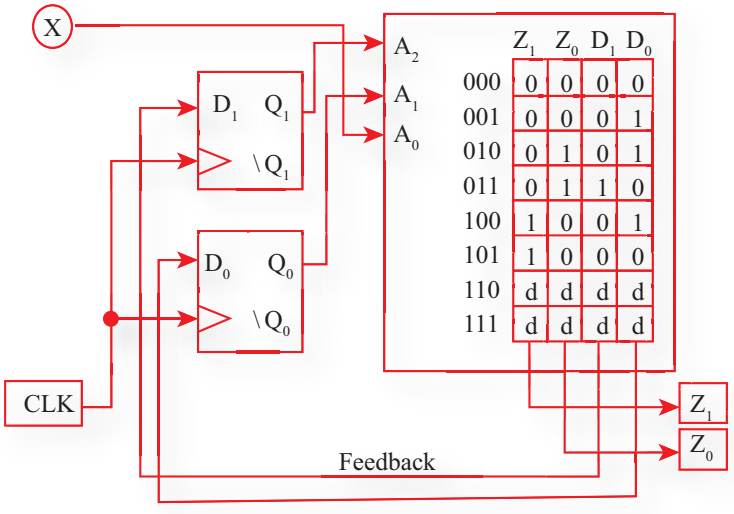
Let us make life simple and pick D flip flops. That means we simply copy the column $q1^+$ to D1 and $q0^+$ to D0.

Table 4: Output with $q1^+$ to D1 and $q0^+$ to D0

Row	q_1	q_0	x	q_1^+	q_0^+	z_1	z_0	D_1	D_0
0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	1
2	0	1	0	0	1	0	1	0	1
3	0	1	1	1	0	0	1	1	0
4	1	0	0	1	0	1	0	1	0
5	1	0	1	0	0	1	0	0	0
6	1	1	0	d	d	d	d	d	d
7	1	1	1	d	d	d	d	d	d

Drawing the Circuit

Then, drawing out the circuit.



This basically consists of drawing the ROM correctly. The main mistakes that one are like to make is copying columns incorrectly, adding more columns than the outputs and the flip flops control, failing to label the address bits and the addresses.

SUMMARY

- A combinational logic circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs.
- Combinational logic circuits are used in various electronic equipment to implement limited intelligence in all sorts of applications.
- Combinational logic circuits can be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classed as "universal" gates.
- An encoder produces an m bit binary code corresponding to the digital input number. The encoder accepts an n input digital word and converts it into an m bit another digital word.
- Multiplexers also are used in building digital semiconductors such as central processing units (CPUs) and graphics controllers.
- A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs.
- Sequential logic (SL) in digital circuit theory is the set of rules and implementations of circuits that rely on the current and past events of logic states and transitions to determine present logic states.
- An asynchronous circuit does not need the precise timing control supported by flip-flops. It may therefore contain latches rather than flip-flops.
- Finite-state machines can solve a large number of problems, among which are electronic design automation, communication protocol design, parsing and other engineering applications.
- Asynchronous sequential logic expresses memorizing effect by fixing moments of time, when digital device changes its state.

KNOWLEDGE CHECK

1. **Select one of the following statements that best describes the parity method of error detection:**
   a. Parity checking is best suited for detecting single-bit errors in transmitted codes.
   b. Parity checking is best suited for detecting double-bit errors that occur during the transmission of codes from one location to another.
   c. Parity checking is not suitable for detecting single-bit errors in transmitted codes.
   d. Parity checking is capable of detecting and correcting errors in transmitted codes.
2. **A multiplexed display .....**
   a. accepts data inputs from one line and passes this data to multiple output lines
   b. uses one display to present two or more pieces of information
   c. accepts data inputs from multiple lines and passes this data to multiple output lines
   d. accepts data inputs from several lines and multiplexes this input data to four BCD lines
3. **When two or more inputs are active simultaneously, the process is called .....**
   a. First-in, first-out processing
   b. Priority encoding
   c. Ripple blanking
   d. Priority decoding
4. **Which type of decoder will select one of sixteen outputs, depending on the 4-bit binary input value?**
   a. Hexadecimal
   b. Dual octal outputs
   c. Binary-to-hexadecimal
   d. Hexadecimal-to-binary

5. **A circuit that responds to a specific set of signals to produce a related digital signal output is called a (n):**
   a. BCD matrix
   b. Display driver
   c. Encoder
   d. Decoder
6. **Most demultiplexers facilitate which of the following?**
   a. decimal to hexadecimal
   b. single input, multiple outputs
   c. ac to dc
   d. odd parity to even parity
7. **One application of a digital multiplexer is to facilitate:**
   a. code conversion
   b. parity checking
   c. parallel-to-serial data conversion
   d. data generation
8. **Which of the following combinations of logic gates can decode binary 1101?**
   a. One 4-input AND gate
   b. One 4-input AND gate, one inverter
   c. One 4-input AND gate, one OR gate
   d. One 4-input NAND gate, one inverter
9. **What is the indication of a short to ground in the output of a driving gate?**
   a. Only the output of the defective gate is affected
   b. There is a signal loss to all load gates
   c. The node may be stuck in either the HIGH or the LOW state
   d. The affected node will be stuck in the HIGH state
10. **3 bits full adder contains _____**
    a. 3 combinational inputs
    b. 4 combinational inputs
    c. 6 combinational inputs
    d. 8 combinational inputs

REVIEW QUESTIONS

1. What do you understand by transistor-transistor logic (TTL)?
2. Explain the standard combinational circuit blocks.
3. Define the classification of combinational logic.
4. What do you understand by flip-flops?
5. Describe the serial-in to serial-out (SISO) shift register.

Check Your Result

1. (a)	2. (b)	3. (b)	4. (a)	5. (c)
6. (b)	7. (c)	8. (b)	9. (b)	10. (d)

REFERENCES

1. Clive Maxfield. "FPGAs: World Class Designs". p. 70. 2009. ISBN 1856176215
2. Douglas Lewin, Logical Design of Switching Circuits, Second Edition, Thomas Nelson and Sons, 1974, ISBN 017 771044 6, pp.162-163
3. http://www.electronics-tutorials.ws/combination/comb_1.html
4. http://www.csse.monash.edu.au/courseware/cse2306/2006/Lnts/L06d.pdf
5. tutorialspoint, http://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm
6. http://www.electronics-tutorials.ws/sequential/seq_1.html
7. http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee141_f01/Notes/chapter7.pdf
8. http://aldrichmamon.blogspot.in/2011/03/finite-automata.html
9. tutorialspoint, http://www.tutorialspoint.com/assembly_programming/assembly_registers.htm
10. whatis.techtarget, http://whatis.techtarget.com/definition/register
11. http://csebtechnotes.blogspot.in/2014/07/sequential-logic-flip-flops-counters.html
12. http://dictionary.sensagent.com/Sequential%20logic/en-en/
13. Michael Predko and Myke Predko, Digital electronics demystified, McGraw-Hill, 2004. ISBN 0-07-144141-7 http://faculty.lasierra.edu/~ehwang/digitaldesign/contents/04%20Combinational%20Components.pdf



CHAPTER 7

MODES OF DATA TRANSFER AND SERIAL COMMUNICATION

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Explain the introduction of modes of data transfer
2. Explain the serial communication

"The Internet is not just one thing, it's a collection of things - of numerous communications networks that all speak the same digital language. "

— James H. Clark

INTRODUCTION

Serial Communication Data Transmission Modes means transferring of data between two devices. It is also called communication mode. These modes direct the direction of flow of information.

Data transfer or transfer is any information that is transferred from one location to another through some communication method. For example, with digital data transmission data signals are sent and received using the binary code.

Another example is for this page to be visible, all text, images, and other data was transferred over the Internet to your computer. For a file to appear on a USB drive, it must first be copied or moved from the hard drive to the USB drive.

Serial Communication, like any data transfer, requires coordination between the sender and receiver. For example, when to start the transmission and when to end it, when one particular bit or byte ends and another begins, when the receiver's capacity has been exceeded, and so on. A protocol defines the specific methods of coordinating transmission between a sender and receiver.

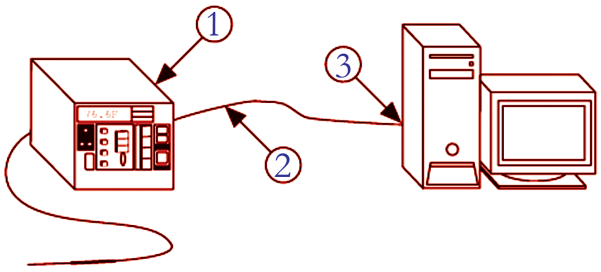
7.1 INTRODUCTION OF MODES OF DATA TRANSFER

Keyword

Bus is a communication system that transfers data between components inside a computer, or between computers.

There are basically two formats of data transfer parallel and serial. In parallel mode, data bits are transferred parallel over the communication lines referred to as buses. Thus all the bits of a byte are transferred simultaneously within the timeframe allotted for the transmission. In serial data transfer, each data bit is sent sequentially over a single data **bus** line. In order to implements serial data transmission. The sender and receiver must divide the timeframe allotted for the transmission of a bite into subintervals during which each bit is sent and received.

Serial communication is a popular means of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer. Serial communication uses a transmitter to send data, one bit at a time, over a single communication line to a receiver. You can use this method when data transfer rates are low or must transfer data over long distances. Serial communication is popular because most computers have one or more serial ports, so no extra hardware is needed other than a cable to connect the instrument to the computer or two computers together.



Modes of data transfer refer to the various modes by which the data residing in the memory is transferred to CPU or back to memory. The data originates from the input units. The originated data is then transferred to the memory for storage. This **binary data** received from any external device is in general stored in memory for later processing. So first you need a transfer of data here. Then for processing the data residing in memory in read by the computer brain i.e. CPU for carrying out various operations and transforming it to make it useful information. The CPU temporarily stores data just for the time being of executing the operation. After that the CPU also need to transfer the information back to the memory. Ultimately the data resides in the memory only. Memory is the only correct place for data or information storage.

Keyword

Binary Data is data whose unit can take on only two possible states, traditionally termed 0 and +1 in accordance with the binary numeral system and Boolean algebra.

There are lot of transferring between the CPU and the memory for reading or writing of data or information respectively. You have for the same various modes of transfer of data. Some uses the CPU as in between the transfer and some directly transfers the data to and from the memory unit. Data to and from the peripherals are handled in one of the four possible modes are as follows:

- Programmed input-output
- Interrupt- initiated input output
- Interrupt driven I/O
- Direct memory access.

Programmed Input-Output

Programmed I/O operations are the result of I/O instructions written in the computer program for transferring of data. Each data item transfer is initiated by an instruction written/ specified

in the program. Usually, the transfer is to and from a CPU register and peripherals like input and output devices. Other instructions are needed to transfer the data to and from CPU and memory of the computer not the peripherals. Once the process is initiated, the CPU is required to monitor the interface of the **peripheral** so that it can sense when the data can be transferred. It is up to the programmed instruction executed in the CPU to keep close tabs on the things that are carrying out in the interface and the I/O devices.

- In programmed I/O mode data are exchanged between the processor and the I/O module. When a processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to that appropriate I/O module. With programmed I/O the I/O module will perform the requested action and then set the appropriate bit in the I/O status register.
- The I/O module takes no further action to alert the processor.
- The I/O commands issued by the processor to the I/O module.
  - Test
  - Control
  - Read
  - Write

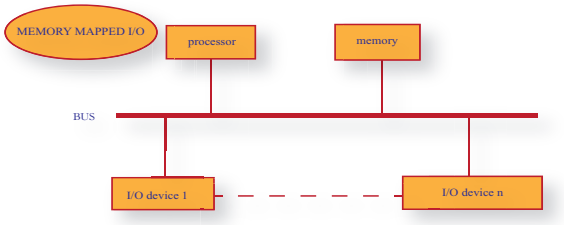
Keyword

Peripheral is a device that is used to put information into or get information out of the computer.

Interrupt Initiated Input/output

In the programmed I/O method, the CPU stays in a program loop until the I/O unit indicates for data transfer which sometimes makes the CPU busy needlessly. This can be avoided by using an interrupt facility and special commands. This informs the interface to issue an interrupt request signal when the data are available from the device. In the meantime CPU is free to execute other things and one can also be sure of maximum utilization of the CPU. In turn interface keeps tracking the device making CPU free. When the interface finds that the device is ready for data transfer, it generates an interrupt request. When the CPU detects the interrupt it stops the present execution and start executing the service of interrupt. (See Figure 7.1)

- There is a single address space for memory location and I/O devices
- With memory mapped I/O a single read line a single write line are needed on the bus. The bus may be equipped with memory read and write plus Input and output command lines.
- Now the command line specifies whether the address refers to memory location or an I/O device.
- Most CPU uses memory mapped I/O.
- Always CPU assigns address to memory some of memory space is stolen and assigned to I/O device.
- It deals with fewer address lines.



Keyword

Interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.

Figure 7.1: Interrupt initiated input/output.

Interrupt Driven I/O

Interrupt driven I/O is defined as follows:

- In this method the program issues I/O command and then continues to execute until it is interrupted by the I/O hardware to signal the end of I/O operation.
- Here the program enters a wait loop in which it repeatedly checks the device status. During this process the processor is not performing any useful computation.
- There are many situations where tasks can be performed while waiting for an I/O device to be ready, to allow this the I/O device should alert the processor when it becomes ready. It can be done by sending a hardware signal called an interrupt.

- The routine executed in response to an interrupt request is called **Interrupt** Service Routine (ISR).
- The processor first completes execution of instruction then it loads the program counter (PC) with the address of 1st instruction of ISR.

Keyword

Interface is a shared boundary across which two separate components of a computer system exchange information.

Direct Memory Access

In direct memory access, the interface transfers data into and out of the memory unit through the memory bus without the intervention of CPU in between. The function of CPU is just to indicate the address of the memory where it needs to be read from.

In this method the input and output devices read/write information from the main memory without interference of the CPU through the system bus. For I/O transfer, processor determines the status of I/O by:

- Polling
- Waiting for interrupt signal.

DMA approach, large blocks of data at high speed can be sent between external device and main memory.

7.1.1 Programmed I/O

In the programmed I/O method, the I/O device dies not have direct access to memory. A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU, and a store instruction to transfer the data from the CPU to memory. Other instructions may be needed to verify that the data are available from the device and to count the numbers of words transferred.

An example of data transfer from an I/O device through an interface into the CPU is device transfers bytes of data one at a time as they are available. When a byte of data is available, the device places it in the I/O bus and enables its data valid line. The interface accepts the byte into its data register and enables the data accepted line. The interface sets a bit in the status register that one will refer to as an F or "flag" bit. The

device can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled by the **interface**. A program is written for the computer to check the flag in the status register to determine if a byte has been placed in the data register by the I/O device. This is done by reading the status register into a CPU register and checking the value of the flag bit. If the flag is equal to 1, the CPU reads the data from the data register. The flag bit is then cleared to 0 by either the CPU or the interface, depending on how the interface circuits are designed. Once the flag is cleared, the interface disables the data accepted line and the device can then transfer the next data byte. It is assumed that the device is sending a sequence of bytes that must be stored in memory. The transfer of each byte requires three instructions, they are as follows:

- Read the status register.
- Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
- Read the data register.

Each byte is read into a CPU register and then transferred to memory with a store instruction. A common I/O programming task is to transfer a block of words form an I/O device and store them in a memory buffer.

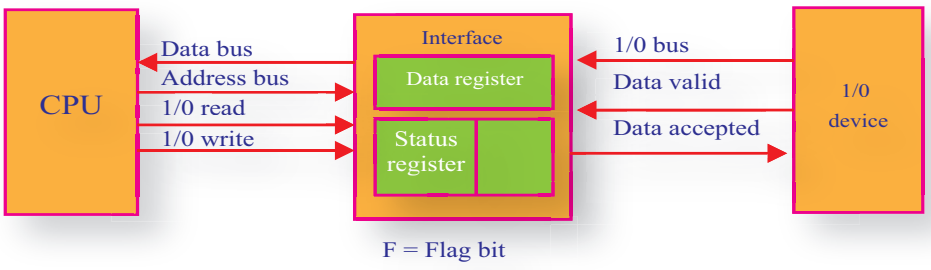


Figure 7.2: Programmed I/O.

7.1.2 Interrupt Initiated I/O and Direct Memory Access

An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility. While the CPU is running a program, it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set. The CPU deviates from what it is doing to take care of the input or output transfer. After the transfer is completed, the computer returns to the program to continue what it was doing before the interrupt.

The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer. The way that the processor chooses the branch address of the service routine varies from tone unit to another. In principle, there are two methods for accomplishing this. One is called vectored interrupt and the other, no vectored interrupt. In a non-vectored interrupt, the branch address is assigned to a fixed location in memory. In a vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector. In some computers the interrupt vector is the first address of the I/O service routine. In other computers the interrupt vector is an address that points to a location in memory where the beginning address of the I/O service routine is stored.

Keyword

Controller is a chip, an expansion card, or a stand-alone device that interfaces with a peripheral device.

The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called direct memory access (DMA). During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA **controller** takes over the buses to manage the transfer directly between the I/O device and memory.

Programmed I/O (PIO) refers to data transfers initiated by a CPU under driver software control to access registers or memory on a device. The CPU issues a command then waits for I/O operations to be complete. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.

Programmed I/O basically works in these ways:

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly

- I/O module does not interrupt CPU
- CPU may wait or come back later

Interrupt

The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.

For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.

For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.

Although interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.

The basic operations of Interrupt are as follows:

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

DMA is a method of transferring data from the computer's RAM to another part of the computer without processing it using the CPU. While most data that is input or output from the computer is processed by the CPU, some data does not require processing, or can be processed by another device. In these situations, DMA can save processing time and is a more efficient way to move data from the computer's memory to other devices.

REMEMBER

The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as Polling. As a result, the level of the performance of the entire system is severely degraded.

The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessors is to disable the buses through special control signals. Two control signals in the CPU that facilitate the DMA transfer. The bus request (BR) input is used by the DMA controller to request the CPU to relinquish control of the buses. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus, and the read and write lines into a high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance.

The CPU activates the Bus grant (BG) output to inform the external DMA that the buses are in the high-impedance state. The DMA that originated the bus request can now take control of the buses to conduct memory transfers without processor intervention. When the DMA terminates the transfer, it disables the bus request line. The CPU disables the bus grant, takes control of the buses, and returns to its normal operation.

When the DMA takes control of the bus system, it communicates directly with the memory. The transfer can be made in several ways. In DMA burst transfer, a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of the memory buses. This mode of transfer is needed for fast devices such as magnetic disks, where data transmission cannot be stopped or slowed down until an entire block is transferred.

DID YOU KNOW ?

Data transmission is utilized in computer networking equipment such as modems (1940), local area networks (LAN) adapters (1964), repeaters, hubs, microwave links, wireless network access points (1997), etc.

Keyword

Synchronization refers to one of two distinct but related concepts: synchronization of processes, and synchronization of data.

7.2 SERIAL COMMUNICATION

Serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels.

Serial communication is used for all long-haul communication and most computer networks, where the cost of cable and **synchronization** difficulties makes parallel communication impractical. Serial computer buses are becoming more common even at shorter distances, as improved signal integrity and transmission speeds in newer serial technologies have begun

to outweigh the parallel bus's advantage of simplicity and to outstrip its disadvantages.

Some points related to serial communication are as follows:

- Each bit of the message is sent in sequence, one bit at a time
- Serial communication requires the use of one or two signal lines
- Slower, but less expensive, since only one conductor is required
- Increasingly important because smaller cables are easier to connect
- Signal skew is less important because there are fewer signals. For serial communication there are at most two signals that signal skew can impact.

REMEMBER

An alternative technique called cycle stealing allows the DMA controller to transfer one data word at a time after which it must return control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to "steal" one memory cycle.

7.2.1 Types of Serial Communication

Modem

Modem is a device that modulates an analog carrier signal to encode digital information, and also demodulates such a carrier signal to decode the transmitted information. The goal is to produce a signal that can be transmitted easily and decoded to reproduce the original digital data. Modem can be used over any means of transmitting analog signals, from driven diodes to radio.

A modem modulates outgoing digital signals from a computer or other digital device to analog signals for a conventional copper twisted pair telephone line and demodulates the incoming analog signal and converts it to a digital signal for the digital device.

In recent years, the 2400 bits per second modem that could carry e-mail has become obsolete. 14.4 Kbps and 28.8 Kbps modems were temporary landing places on the way to the much higher bandwidth devices and carriers of tomorrow. From early 1998, most new personal computers came with 56 Kbps modems. By comparison, using a digital Integrated Services Digital Network adapter instead of a conventional modem, the same telephone wire can now carry up to 128 Kbps. With Digital

Subscriber Line (DSL) systems, now being deployed in a number of communities, bandwidth on twisted-pair can be in the megabit range.



Simplex

A simplex line carries information in one direction only. Simplex is a communications mode in which only one signal is transmitted, and it always goes in the same direction. The transmitter and the receiver operate on the same frequency. When two stations exist and they alternately send signals to each other on the same frequency, the mode is technically known as half duplex. However, most amateur radio operators refer to half duplex as simplex.

At high and microwave wireless frequencies, the simplex or half duplex modes will not provide enough communications range in some situations. To increase the effective range, wireless repeaters are used. In a typical repeater, the incoming signal has a different frequency than the outgoing signal, so that the transmitted signal does not overwhelm the repeater's receiver. Repeaters, strategically located in high places with large line-of-sight coverage zones, can greatly increase the range of a **wireless communications** system.

In order for a station to send and receive data simultaneously, two different frequencies must be used, one for transmitting and the other for receiving, a mode called full duplex. A more sophisticated mode called multiplexing allows for two or more signals or streams of information to be sent simultaneously in the same direction as an "interwoven" complex signal. The component signals are extracted and decoded individually at the receiving end.

The term simplex is sometimes used in reference to any program that resembles Simplex Database, a CGI script that provides a simple and portable database for Webmasters along with tools for creating custom. *Examples* of simplex lines are radio and television broadcasting

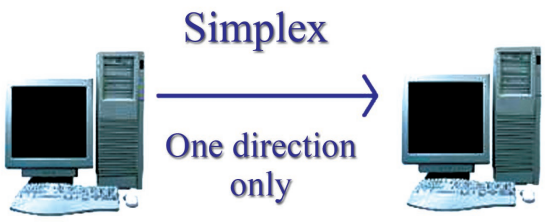


Figure 7.3: Simplex.

Half-Duplex

A half-duplex line is capable of transmitting in both directions, but in only one direction at a time. One modem acts as the transmitter and the other modem acts as the receiver. When transmission in one direction is completed, the roles of the modems are reversed to enable transmission in the opposite direction. A half-duplex channel can send and receive, but not at the same time. It is like a one-lane bridge where two way traffic must give way in order to cross. Only one end transmits at a time, the other end receives. In addition, it is possible to perform error detection and request the sender to retransmit information that arrived corrupted. In some aspects, one can think of Internet surfing as being half-duplex, as a user issues a request for a web document, then that document is downloaded and displayed before the user issues another request. Another example of half-duplex is talk-back radio, and CB Radio. One might have seen movies where truckles communicate to each other, and when they want the other person to speak they say "over". This is because only one person can talk at a time.

Keyword

Wireless Communications is a type of data communication that is performed and delivered wirelessly.

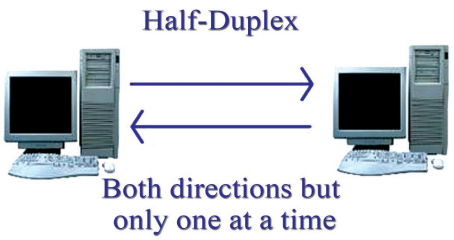


Figure 7.4: Half-duplex.

Turnaround Time

The time required to switch a half-duplex line from one direction to the other is called the turnaround time.

Full-Duplex

A full-duplex transmission system can send and receive data in both directions simultaneously. Full-duplex, or simply "duplex," is a type of communication in which data can flow two ways at the same time. Full duplex devices, therefore, can communicate back and forth simultaneously.

Telephones are common examples of full-duplex devices. They allow both people to hear each other at the same time. In the computer world, most network protocols are duplex, enabling hardware devices to send data back and forth simultaneously. *For example*, two computers connected via an Ethernet cable can send and receive data at the same time. Wireless networks also support full-duplex communication. Additionally, modern I/O standards, such as USB and Thunderbolt, are full-duplex.

The terms duplex and full-duplex can be used interchangeably since both refer to simultaneous bidirectional communication. Full-duplex is often used in contrast to half-duplex, which refers to bidirectional communication, but not at the same time. Simplex communication is even more limited and only supports data transmission in one direction.

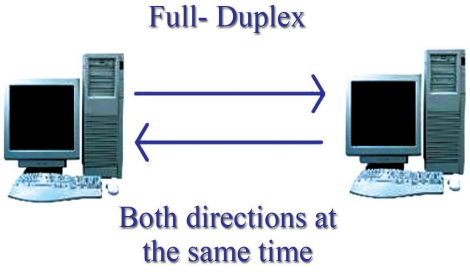


Figure 7.5: Full-duplex

Synchronous Transmission

Two units share a common clock frequency, and bits are transmitted continuously at that frequency. Information must be transmitted continuously to keep the clock frequencies in both units synchronized.

Frequency synchronization is achieved by the receiving modem from the signal transitions that occur in the data that are received. Any frequency shift that may occur

between the transmitter and receiver clocks is continuously adjusted by maintaining the receiver clock at the frequency of the incoming bit stream.

Asynchronous Transmission

Binary information is sent only when it is available, and the line remains idle when there is no information to be transmitted.

7.2.2 Asynchronous and Synchronous Communication

Synchronous and asynchronous transmissions are two different methods of transmission synchronization. Synchronous transmissions are synchronized by an external clock, while asynchronous transmissions are synchronized by special signals along the transmission medium. Asynchronous communication utilizes a transmitter, a receiver and a wire without coordination about the timing of individual bits. There is no coordination between the two end points on just how long the transmitter leaves the signal at a certain level to represent a single digital bit. Each device uses a clock to measure out the 'length' of a bit. The transmitting device simply transmits. The receiving device has to look at the incoming signal and what it is receiving and coordinate and retime its clock to match the incoming signal.

Sending data encoded into the signal requires that the sender and receiver are both using the same encoding/decoding method, and know where to look in the signal to find data. Asynchronous systems do not send separate information to indicate the encoding or clocking information. The receiver must decide the clocking of the signal on it is own. This means that the receiver must decide where to look in the signal stream to find ones and zeroes, and decide for itself where each individual bit stops and starts. This information is not in the data in the signal sent from transmitting unit.

Most communications between devices within computers and between computers and external devices are asynchronous. Asynchronous communications also include computer-based exchanges of messages for which the participants need not be available or online at the same time, but, rather, read and respond as their schedules permit.

Direct communication, where all parties involved in the communication are present at the same time is a form of synchronous communication or real time communication. Examples include telephone conversations and instant messaging.

In asynchronous communications it is necessary for the receiver to be able to distinguish between valid data and noise. In computer communications, this is usually accomplished through the use of special bits at the beginning and end of each message.

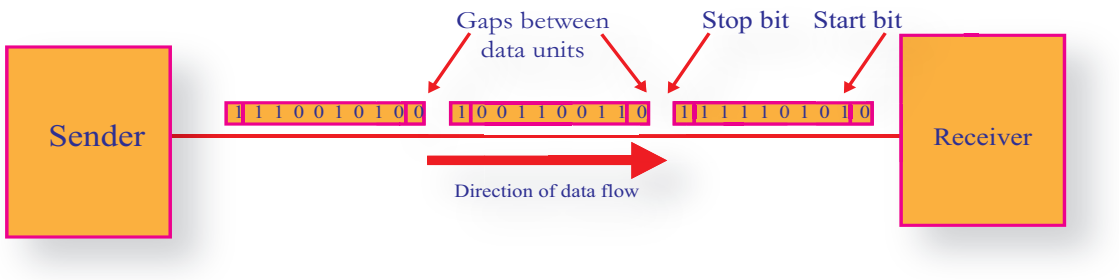


Figure 7.6: Asynchronous communication.

DID YOU KNOW ?

Asynchronous communication is used on RS-232 based serial devices such as on an IBM compatible computer's communication ports.

When the receiver of signal carrying information has to derive how that signal is organized without consulting the transmitting device, it is called asynchronous communication. In short, the two ends do not always negotiate or work out the connection parameters before communicating. Asynchronous communication is more efficient when there is low loss and low error rates over the transmission medium because data is not retransmitted and no time is spent setting negotiating the connection parameters at the beginning of transmission. Asynchronous systems just transmit and let the far end station. Asynchronous is sometimes called "best effort" transmission because one side simply transmits, and the other does it is best to receive and any lost data is recovered by a higher level protocol.

Advantages of Asynchronous Communication

- Simple, does not require synchronization of both communication sides
- Cheap, because Asynchronous transmission require less hardware
- Set-up is faster than other transmissions, so well suited for applications where messages are generated at irregular intervals, for example data entry from the keyboard and the speed depends on different applications.

Disadvantages of Asynchronous Communication

- Large relative overhead, a high proportion of the transmitted bits are uniquely for control purposes and thus carry no useful information

Synchronous communications is the more efficient method of communications. CQ's connectivity solutions communicate through the synchronous method of communications. Through synchronous communications, data is transmitted as frames of large data blocks rather than bulky individual bytes. One advantage of synchronous is that control information is easily inserted at the beginning and end of each block to ensure constant timing, or synchronization. Another advantage of synchronous is that it is more efficient than asynchronous.

Synchronous systems negotiate the communication parameters at the data link layer before communication begins. Basic synchronous systems will synchronize the signal clocks on both sides before transmission begins, reset their numeric counters and take other steps. More advanced systems may negotiate things like error correction and compression.

It is possible to have both sides try to synchronize the connection at the same time. Usually, there is a process to decide which end should be in control. Both sides in synchronous communication can go through a lengthy negotiation cycle where they exchange communications parameters and status information. With a lengthy connection establishment process, a synchronous system using an unreliable physical connection will spend a great deal of time in negotiating, but not in actual data transfer. Once a connection is established, the transmitter sends out a signal, and the receiver sends back data regarding that transmission, and what it received. This connection negotiation process takes longer on low error-rate lines, but is highly efficient in systems where the transmission medium itself is not particularly reliable.

A 56 Kbps dial-up synchronous line can carry 7000 bytes per second (56000/8) compared to a 56 Kbps dial-up asynchronous line which can only carry 5600 bytes per second (56000/10). When transmitting large amounts of information, this translates into a significant increase in speed and performance.



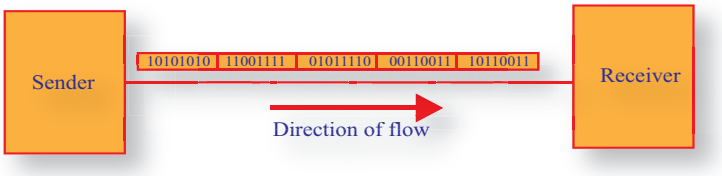


Figure 7.7: Synchronous communication.

Advantages of Synchronous communication

- Lower overhead and thus, greater throughput

Disadvantages of Synchronous Communication

- Slightly more complex
- Hardware is more expensive

7.2.3 Standard Communication Interfaces

Communicating interface use well-defined formats for exchanging messages from sender to receiver. Each message has an exact meaning intended to elicit a response from a range of possible responses pre-determined for that particular situation. Thus, a protocol must define the syntax, semantics, and synchronization of communication; the specified behavior is typically independent of how it is to be implemented. A protocol can therefore be implemented as hardware, software, or both.

7.2.4 Cables that Carry Serial Data

A cable that carries serial data is a serial cable. A serial cable is a cable used to carry data to transfer between two devices using a serial communication protocol. The form of connectors depends on the particular serial port used. A cable wired for connecting two DTEs directly is known as a null modem cable.

CASE STUDY

SOLARGAIN: DIRECT MEMORY ACCESS (DMA)



Solargain is currently the top Microsoft (OEM) and Intel channel partners locally, and are tier one partner with Panasonic. DMA operate 5 days per week from their main office/warehouse.

To address their concerns regarding recent (and future) increases in the cost of energy DMA installed a 30kW Solargain solar PV system. The system comprises 156 x Solar One Monocrystalline solar panels and 3 SMA Sunny Tripower 10000TL 3-phase inverters. Their roof is pitched and the panels were flush-mounted on the most suitable part of the building being the North West orientation.

DMA's initial concern when looking at a solar PV system was that as they operate only over a 5 day period, the production they made on Saturday and Sunday would not be utilized given that there is currently no buyback arrangement for commercial solar systems in the state in which they operate. However, several changes occurred during their period of enquiry which over time created an appropriate return on investment for them in spite of only self-consuming around 70% of the power they will generate over the week.

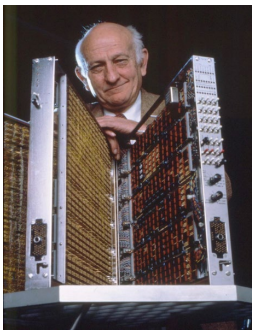
These Changes included

- The wholesale cost of quality solar panels reduced by 50% in the 2011 calendar year
- Larger scale commercial inverters have become available in Australia further reducing the system and installation cost.
- Business tariffs having increased by 20% as of 1 July 2011 much more than the residential tariffs increases.

Financially, given their current tariff and the projected rises in the cost of energy to come they anticipate the system will have paid for itself in 4-5 years giving them a most satisfactory tax free financial return. DMA are also the freehold owners of the premises in which they operate and understand the value added to the premises an

installing the system should they ever seek to sell or lease the premises in the future. DMA were so delighted with the quality of product and workmanship with their first system that they have now asked Solargain to assist them in the future with their next system when their building is split into two strata units.

ROLE MODEL



JOHN COCKE: THE INVENTOR OF RISC ARCHITECTURE

John Cocke was born on May 30, 1925 in Charlotte, North Carolina. In 1946, he earned his bachelor's degree in mechanical engineering from Duke University. Sometime later, he returned to Duke to pursue his doctorate, graduating in 1953 with a Ph.D. in mathematics. In 1956, he was hired by IBM's Research division to work with computers and he continued working for IBM until retiring in 1992. While working for IBM, Cocke became one of the leading innovators in the area of compiler optimization. The techniques of optimization that he developed allowed higher end programming languages like FORTRAN, C, and PASCAL to attain binary-level efficiency comparable to that of lower-level languages. These advancements greatly advanced computer science and helped to make more advanced languages like C++ and JAVA possible.

In 1975, while working at the T.J. Watson Center in Yorktown Heights, New York, Cocke worked closely with other scientists to revolutionize the design of computer processors. The new ideas in design that he helped to establish became the basis for a new technology: the Reduced Instruction Set Computer or RISC.

Where computer engineers had previously focused on developing massively complex processors that supported a wide range of operations, RISC advocated simplicity and efficiency. The RISC processor is based around operations that can be completed in a single clock cycle, as compared to other processors that normally require several clock cycles. The RISC processor idea has provided the foundation for modern processor design and triggered a revolution in computing.

The first processor based on the RISC concept was IBM's 801. The RISC optimizations allowed the processor to operate at about twice the speed of prior processors, while using about the same amount of transistors. The hardware optimizations allowed faster chips to be produced at the same or similar cost, a phenomenon that continues today. In the 1980's, Cocke also worked on developing a very efficient logic simulation device.

The device allowed him to simulate a complex logic device, such as a processor, and verify its proper operation prior to creating the circuit on silicon. This method of simulation is still used today and allows hardware engineers to verify their designs without actually building the circuits, allowing them to work faster.

In 1987, Cocke was presented with the A. M. Turing Award by the Association for Computing Machinery. In 1972, he was named an IBM Fellow and in 1979, he was elected to the National Academy of Engineering. In 1985, he was awarded the Eckert-Mauchly Award by the IEEE Computer Society. The Massachussetts Institute of Technology and Courant Institute of Mathematical Sciences also appointed him an honorary professor. His alma mater, Duke University, also awarded him an honorary degree in 1988. In 1991, he was presented with the National Medal of Technology by President Bush and, in 1994, President Clinton awarded him with the National Medal of Science.

John Cocke died on July 16, 2002 in Valhalla, New York after a series of strokes and other ailments.

What Is The Moral Here?

John Cocke was the principal designer of the type of microprocessor that serves as the engine of most of today's large, powerful computers and the Apple Macintosh personal computers. Machines using his chip design a simplification of the hardware, which opened the door to faster computation are reduced instruction-set computers, or RISC.

SUMMARY

- Serial Communication Data Transmission Modes means transferring of data between two devices. It is also called communication mode. These modes direct the direction of flow of information.
- In serial data transfer, each data bit is sent sequentially over a single data bus line. In order to implements serial data transmission.
- With memory mapped I/O a single read line a single write line are needed on the bus. The bus may be equipped with memory read and write plus Input and output command lines.
- The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer.
- The CPU activates the Bus grant (BG) output to inform the external DMA that the buses are in the high-impedance state.
- Serial communication is used for all long-haul communication and most computer networks, where the cost of cable and synchronization difficulties makes parallel communication impractical.
- A modem modulates outgoing digital signals from a computer or other digital device to analog signals for a conventional copper twisted pair telephone line and demodulates the incoming analog signal and converts it to a digital signal for the digital device.
- The terms duplex and full-duplex can be used interchangeably since both refer to simultaneous bidirectional communication.
- Synchronous and asynchronous transmissions are two different methods of transmission synchronization.

KNOWLEDGE CHECK

1. The double buffer is used for ……..

a. enabling receival of multiple bits of input.
b. combining the input and output operations.
c. extending the buffer capacity.
d. none of these.

2. …….. to increase the flexibility of the serial ports.

a. The wires used for ports is changed
b. The ports are made to allow different clock signals for input and output
c. The drivers are modified
d. All of the above

3. UART stands for:

a. Universal Asynchronous Relay Transmission.
b. Universal Accumulator Register Transfer.
c. Universal Asynchronous Receiver Transmitter.
d. None of the above.

4. The key feature of UART is:

a. Its architectural design.
b. Its simple implementation.
c. Its general purpose usage.
d. Its enhancement of connecting low speed devices.

5. The data transfer in UART is done in ……...

a. asynchronous start stop format
b. synchrnous start stop format
c. isochronous format
d. ebdic format

6. The serial communication is

a. cheaper communication
b. requires less number of conductors
c. slow process of communication
d. all of the mentioned

7. The serial communication is used for

a. short distance communication

b. long distance communication
c. short and long distance communication
d. communication for a certain range of distance

8. The number of bits transmitted or received per second is defined as

a. transmission rate
b. reception rate
c. transceiver rate
d. baud rate

9. The transmission unit does not require assistance from processor if once a byte for transmission is written to

a. SCON register
b. SBUF register
c. SFR address
d. Any of the mentioned

10. During serial reception, the buffer that receives serial bits and converts to a byte is

a. receive buffer 0
b. receive buffer 1
c. receive buffer 2
d. none

REVIEW QUESTIONS

1. Explain the introduction of modes of data transfer.
2. Define interrupt initiated I/O and direct memory access.
3. What do you understand by serial communication?
4. Discuss about advantages and disadvantages of asynchronous communication
5. Explain standard communication interfaces.

Check Your Result

1. (a)	2. (b)	3. (c)	4. (d)	5. (a)
6. (d)	7. (b)	8. (d)	9. (b)	10. (b)

REFERENCES

1. "What is Asynchronous Transmission? - Definition from Techopedia". Techopedia.com. Retrieved 2017-12-08.
2. David R. Smith, "Digital Transmission Systems", Kluwer International Publishers, 2003, ISBN 1-4020-7587-1. See table-of-contents.
3. F. Foukalas et al., "Cross-layer design proposals for wireless mobile networks: a survey and taxonomy "
4. John Proakis, "Digital Communications", 4th edition, McGraw-Hill, 2000. ISBN 0-07-232111-3. See table-of-contents.
5. Sergio Benedetto, Ezio Biglieri, "Principles of Digital Transmission: With Wireless Applications", Springer 2008, ISBN 0-306-45753-9, ISBN 978-0-306-45753-1. See table-of-contents



CHAPTER 8

SOFTWARE ARCHITECTURE AND DESIGN

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

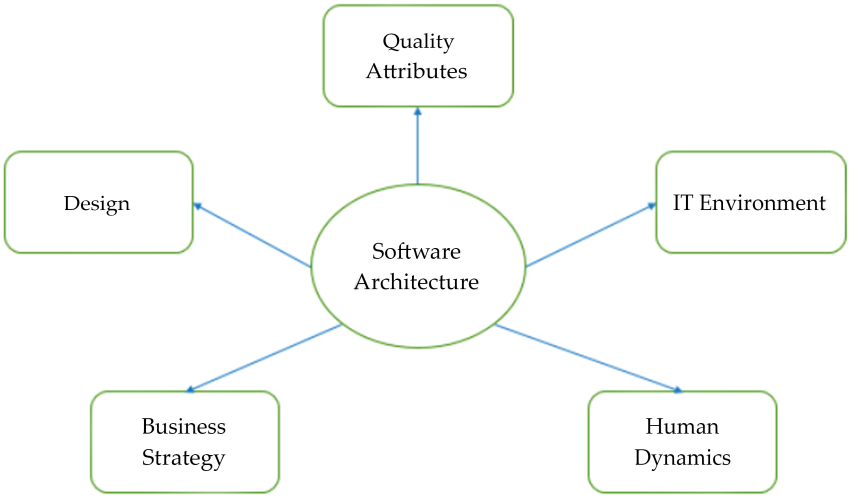
1. Define the concept of software architecture
2. Explain the key principles of software architecture
3. Describe architecture models
4. Elaborate object-oriented paradigm
5. Discuss about data flow architecture
6. Explain data-centered architecture
7. Illustrate the hierarchical architecture
8. Give explanation about interaction-oriented architecture
9. Define distributed architecture
10. List the component-based architecture
11. Discuss about user interface

"I'm a programmer. I like programming. And the best way I've found to have a positive impact on code is to write it."

— Robert C. Martin

INTRODUCTION

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design is a process that includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In Architecture, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

8.1 CONCEPT OF SOFTWARE ARCHITECTURE

Architecture serves as a blueprint for a system. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

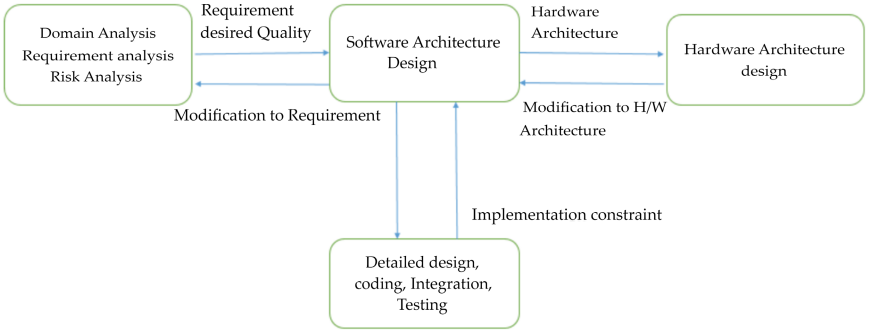
- It defines a structured solution to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- It involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of:
  - Selection of structural elements and their interfaces by which the system is composed.
  - Behavior as specified in collaborations among those elements.
  - Composition of these structural and behavioral elements into large subsystem.
  - Architectural decisions align with business objectives.
  - Architectural styles that guide the organization.

8.1.1 Software Design

Software design provides a design plan that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows:

- To negotiate system requirements, and to set expectations with customers, marketing and management personnel.
- Act as a blueprint during the development process.
- Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.



8.1.2 Goals of Architecture

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements. Some of the other goals are as follows:

- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization's market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

8.1.3 Limitations

Software architecture is still an emerging discipline within software engineering. It has the following limitations:

- Lack of tools and standardized ways to represent architecture
- Lack of analysis methods to predict whether architecture will result in an implementation that meets the requirements.
- Lack of awareness of the importance of architectural design to software development
- Lack of understanding of the role of software architect and poor communication among stakeholders.
- Lack of understanding of the **design process**, design experience and evaluation of design

Keyword

Design Process is an approach for breaking down a large project into manageable chunks.

8.1.4 Role of Software Architect

A Software Architect provides a solution that the technical team can create and design for the entire application. A software architect should have expertise in the following areas:

Design Expertise

- Expert in software design, including diverse methods and approaches such as object-oriented design, event-driven design, etc.
- Lead the development team and coordinate the development efforts for the integrity of the design.
- Should be able to review design proposals and tradeoffs among them.

Domain Expertise

- Expert on the system being developed and plan for software evolution.
- Assist in the requirement investigation process assuring completeness and consistency.
- Coordinate the definition of domain model for the system being developed.

Technology Expertise

- Expert on available technologies that helps in the implementation of the system.
- Coordinate the selection of programming language, framework, platforms, databases, etc.

Methodological Expertise

- Expert on software development methodologies that may be adopted during SDLC (Software Development Life Cycle).
- Choose the appropriate approaches for development that helps the entire team.

Hidden Role of Software Architect

- Facilitates the technical work among team members and reinforcing the trust relationship in the team.
- Information specialist who shares knowledge and has vast experience.
- Protect the team members from external forces that would distract them and bring less value to the project.

Deliverables of the Architect

- A clear, complete, consistent, and achievable set of functional goals
- A functional description of the system, with at least two layers of decomposition
- A concept for the system
- A design in the form of the system, with at least two layers of decomposition
- A notion of the timing, operator attributes, and the implementation and operation plans
- A document or process which ensures functional decomposition is followed, and the form of interfaces is controlled

REMEMBER

Architectural designs for industrial software must typically be adapted from pure forms to specialized styles that meet the needs of the specific domain.

8.1.5 Quality Attributes

Quality is a measure of excellence or the state of being free from deficiencies or defects. Quality attributes are system properties that are separate from the functionality of the system.

Implementing quality attributes makes it is easier to differentiate a good system from a bad one. Attributes are overall factors that affect runtime behavior, system design, and user experience. They can be classified as follows:

Static Quality Attributes

Reflect the structure of system and organization, directly related to architecture, design, source code. They are invisible to end-user, but affect the development and maintenance cost, e.g.: modularity, testability, maintainability, etc.

Dynamic Quality Attributes

Reflect the behavior of the system during its execution. They are directly related to system's architecture, design, source code and also the configuration, deployment parameters, environment, and platform.

They are visible to the end-user and exist at runtime, e.g.: throughput, robustness, scalability, etc.

8.1.6 Quality Scenarios

Quality scenarios specify how to prevent a fault from becoming a failure. They can be divided into six parts based on their attribute specifications:

- ***Source*** An internal or external entity such as people, hardware, software, or physical infrastructure that generates the stimulus.
- ***Stimulus*** A condition that needs to be considered when it arrives on a system.
- ***Environment*** The stimulus occurs within certain conditions.
- ***Artifact*** A whole system or some part of it such as processors, communication channel, persistent storage, processes etc.
- ***Response*** An activity undertaken after the arrival of stimulus such as detect faults, recover from fault, disable event source etc.
- ***Response measure*** Should measure the occurred responses so that the requirements can be tested.

8.1.7 Common Quality Attributes

The following table lists the common quality attributes a software architecture must have.

Category	Quality Attribute	Description
Design Qualities	Conceptual Integrity	Defines the consistency and coherence of the overall design. This includes the way components or modules are designed
	Maintainability	Ability of the system to undergo changes with a degree of ease.
	Reusability	Defines the capability for components and subsystems to be suitable for use in other applications
Run-time Qualities	Interoperability	Ability of a system or different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties.
	Manageability	Defines how easy it is for system administrators to manage the application
	Reliability	Ability of a system to remain operational over time
	Scalability	Ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged.
	Security	Capability of a system to prevent malicious or accidental actions outside of the designed usages
	Performance	Indication of the responsiveness of a system to execute any action within a given time interval.
	Availability	Defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period.
System Qualities	Supportability	Ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.
	Testability	Measure of how easy it is to create test criteria for the system and its components
User Qualities	Usability	Defines how well the application meets the requirements of the user and consumer by being intuitive
Architecture Quality	Correctness	Accountability for satisfying all the requirements of the system.
Non-runtime Quality	Portability	Ability of the system to run under different computing environment.
	Integrality	Ability to make separately developed components of the system work correctly together.
	Modifiability	Ease with which each software system can accommodate changes to its software.
Business quality attributes	Cost and schedule	Cost of the system with respect to time to market, expected project lifetime & utilization of legacy
	Marketability	Use of system with respect to market competition.



Software architecture as a concept has its origins in the research of Edsger Dijkstra in 1968 and David Parnas in the early 1970s.

8.2 KEY PRINCIPLES OF SOFTWARE ARCHITECTURE

Software architecture is described as the organization of a system, where the system represents a collection of components that accomplish a specific set of functions.

8.2.1 Architectural Style

The architectural style, also called as architectural pattern, is a set of principles which shapes an application. It defines an abstract framework for a family of system in terms of the pattern of structural organization. The architectural style

- Provides a lexicon of components and connectors with rules on how they can be combined.
- Improves partitioning and allows the reuse of design by giving solutions to frequently occurring problems.
- Describes a particular way to configure a collection of components (a module with well-defined interfaces, reusable, replaceable) and connectors (communication link between modules).

The software that is built for computer-based systems exhibit one of many architectural styles. Each style describes a system category that encompasses:

- A set of component types which perform a required function by the system
- A set of connectors (subroutine call, remote procedure call, data stream, socket) that enable communication, coordination, and cooperation among components
- Semantic constraints which define how components can be integrated to form the system
- A topological layout of the components indicating their runtime interrelationships

8.2.2 Common Architectural Design

The following table lists architectural styles that can be organized by their key focus area.

Category	Architectural Design	Description
	Message bus	Prescribes use of a software system that can receive and send messages using one or more communication channels

Communication	Service–Oriented Architecture (SOA)	Defines the applications that expose and consume functionality as a service using contracts and messages.
Deployment	Client/server	Separate the system into two applications, where the client makes requests to the server
	3-tier or N-tier	Separates the functionality into separate segments with each segment being a tier located on a physically separate computer.
Domain	Domain Driven Design	Focused on modeling a business domain and defining business objects based on entities within the business domain.
Structure	Component Based	Breakdown the application design into reusable functional or logical components that expose well-defined communication interfaces.
	Layered	Divide the concerns of the application into stacked groups (layers).
	Object oriented	Based on the division of responsibilities of an application or system into objects, each containing the data and the behavior relevant to the object.

8.2.3 Types of Architecture

There are four types of architecture from the viewpoint of an enterprise and collectively, these architectures are referred to as enterprise architecture.

- ***Business architecture:*** Defines the strategy of business, governance, organization, and key business processes within an enterprise and focuses on the analysis and design of business processes.
- ***Application (software) architecture:*** Serves as the **blueprint** for individual application systems, their interactions, and their relationships to the business processes of the organization
- ***Information architecture:*** Defines the logical and physical data assets and data management resources.
- ***Information technology (IT) architecture:*** Defines the hardware and software building blocks that make up the overall information system of the organization.

Keyword

Blueprint is a reproduction of a technical drawing or engineering drawing using a contact print process on light-sensitive sheets.

8.2.4 Architecture Design Process

The architecture design process focuses on the decomposition of a system into components and their interactions to satisfy functional and nonfunctional requirements. The key inputs to software architecture design are:

- The requirements produced by the analysis tasks
- The hardware architecture (the software architect in turn provides requirements to the system architect, who configures the hardware architecture)

The result or output of the architecture design process is an architectural description. The basic architecture design process is composed of the following steps:

Understand the Problem

- This is the most crucial step because it affects the quality of the design that follows.
- Without a clear understanding of the problem, it is not possible to create an effective solution.
- Many software projects and products are considered failures because they did not actually solve a valid business problem or have a recognizable return on investment (ROI).

Identify Design Elements and their Relationships

- In this phase, build a baseline for defining the boundaries and context of the system.
- Decomposition of the system into its main components based on functional requirements. The decomposition can be modeled using a design structure matrix (DSM), which shows the dependencies between design elements without specifying the granularity of the elements.
- In this step, the first validation of the architecture is done by describing a number of system instances and this step is referred as functionality based architectural design.

Evaluate the Architecture Design

- Each quality attribute is given an estimate so in order to gather qualitative measures or quantitative data, the design is evaluated.
- It involves evaluating the architecture for conformance to architectural quality attributes requirements.

- If all estimated quality attributes are as per the required standard, the architectural design process is finished.
- If not, the third phase of software architecture design is entered: architecture transformation. If the observed quality attribute does not meet its requirements, then a new design must be created.

Transform the Architecture Design

- This step is performed after an evaluation of the architectural design. The architectural design must be changed until it completely satisfies the quality attribute requirements.
- It is concerned with selecting design solutions to improve the quality attributes while preserving the domain functionality.
- A design is transformed by applying design operators, styles, or patterns. For transformation, take the existing design and apply design operator such as decomposition, replication, compression, abstraction, and resource sharing.
- The design is again evaluated and the same process is repeated multiple times if necessary and even performed recursively.
- The transformations (i.e. quality attribute optimizing solutions) generally improve one or some quality attributes while they affect others negatively

8.2.5 Key Architecture Principles

Following are the key principles to be considered while designing an architecture:

Build to Change Instead of Building to Last

Consider how the application may need to change over time to address new requirements and challenges, and build in the flexibility to support this.

Reduce Risk and Model to Analyze

Use design tools, visualizations, modeling systems such as UML to capture requirements and design decisions. The impacts can also be analyzed. Do not formalize the model to the extent that it suppresses the capability to iterate and adapt the design easily.

Use Models and Visualizations as a Communication and Collaboration Tool

Efficient communication of the design, the decisions, and ongoing changes to the design is critical to good architecture. Use models, views, and other visualizations of the architecture to communicate and share the design efficiently with all the stakeholders.

This enables rapid communication of changes to the design. Identify and understand key engineering decisions and areas where mistakes are most often made. Invest in getting key decisions right the first time to make the design more flexible and less likely to be broken by changes.

Use an Incremental and Iterative Approach

Start with baseline architecture and then evolve candidate architectures by iterative testing to improve the architecture. Iteratively add details to the design over multiple passes to get the big or right picture and then focus on the details.

Key Design Principles

Following are the design principles to be considered for minimizing cost, maintenance requirements, and maximizing extendibility, usability of architecture

Separation of Concerns

Divide the components of system into specific features so that there is no overlapping among the components functionality. This will provide high cohesion and low coupling. This approach avoids the interdependency among components of system which helps in maintaining the system easy.

Single Responsibility Principle

Each and every module of system should have one specific responsibility which helps the user to clearly understand the system. It should also help with integration of the component with other components.

Principle of Least Knowledge

Any component or object should not have knowledge about internal details of other components. This approach avoids interdependency and helps maintainability.

Minimize Large Design Upfront

Minimize large design upfront if the requirements of an application are unclear. If there is a possibility of modifying requirements, then avoid making a large design for whole system.

Do not Repeat the Functionality

Do not repeat functionality specifies that functionality of components should not to be repeated and hence a piece of code should be implemented in one component only. Duplication of functionality within an application can make it difficult to implement changes, decrease clarity, and introduce potential inconsistencies.

Prefer Composition over Inheritance while Reusing the Functionality

Inheritance creates dependency between children and parent classes and hence it blocks the free use of the child classes. In contrast, the composition provides a great level of freedom and reduces the inheritance hierarchies.

Identify Components and Group them in Logical Layers

Identity components and the area of concern that are needed in system to satisfy the requirements. Then group these related components in logical layer which will help the user to understand the structure of the system at a high level. Avoid mixing components of different type of concerns in same layer.

Define the Communication Protocol between Layers

Understand how components will communicate with each other which requires a complete knowledge of deployment scenarios and the production environment.

Define Data Format for a Layer

Various components will interact with each other through data format. Do not mix the data formats so that applications are easy to implement, extend and maintain. Try to keep data format same for a layer, so that various components need not code/decode the data while communicating with each other. It reduces a processing overhead.

System Service Components should be Abstract

Code related to security, communications, or system services like logging, profiling, and configuration etc. should be abstracted in separate components. Do not mix this code with business logic, as it is easy to extend design and maintain it.

Design Exceptions and Exception Handling Mechanism

Defining exceptions in advance helps the components to manage errors or unwanted situation in an elegant manner. The exception management will be same throughout the system.

Naming Conventions

Naming conventions should be defined in advance. They provide a consistent model that help the users to understand the system easily. It is easier for team members to validate code written by others, and hence will increase the maintainability.

Keyword

Scalability is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.

8.3 ARCHITECTURE MODELS

Software architecture specifies the high level structure of software system abstraction, by using decomposition and composition, with architectural style and quality attributes. A software architecture design must conform to the major functionality and performance requirements of the system, as well as satisfy the non-functional requirements such as reliability, **scalability**, portability, and availability.

A software architecture must describe its group of components, their connections, interactions among them and deployment configuration of all components.

A software architecture can be defined in many ways:

- ***UML (Unified Modeling Language):*** UML is one of object-oriented solutions used in software modeling and design.
- ***Architecture View Model (4+1 view model):*** Architecture view model represents the functional and non-functional requirements of software application.
- ***ADL (Architecture Description Language):*** ADL defines the software architecture formally and semantically.

8.3.1 UML

UML stands for Unified Modeling Language. It is a pictorial language used to make software blueprints. UML was created by Object Management Group (OMG) and UML 1.0 specification

draft was proposed to the OMG in January 1997. It serves as a standard for software requirement analysis and design documents which are the basis for developing a software. UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document a software system. Although UML is generally used to model software system, it is not limited within this boundary. It is also used to model non software systems like process flows in a manufacturing unit. The elements are like components which can be associated with different ways to make a complete UML picture which is known as a diagram. So it is very important to understand the different diagrams to implement the knowledge in real-life systems. We have two broad categories of diagrams and they are again divided into sub-categories: Structural Diagrams and Behavioral Diagrams.

Structural Diagrams

Structural diagrams represent the static aspect of a system. These static aspects represent those parts of a diagram which forms the main structure and is therefore stable.

These static parts are represented by classes, interfaces, objects, components and nodes. Structural diagrams can be sub-divided as follows:

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram
- Package diagram
- Composite structure

The following table provides a brief description of these diagrams:

Diagram	Description
Class	Represents the object orientation of a system. Shows how classes are statically related
Object	Represents a set of objects and their relationships at runtime and also represent the static view of the system
Component	Describes all the components, their interrelationship, interactions and interface of the system
Composite structure	Describes inner structure of component including all classes, interfaces of the component etc
Package	Describes the package structure and organization. Covers classes in the package and packages within another package
Deployment	Deployment diagrams are a set of nodes and their relationships. These nodes are physical entities where the components are deployed

Behavioral Diagrams

Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic aspects are basically the changing/moving parts of a system. UML has the following types of behavioral diagrams:

- Use case diagram
- Sequence diagram
- Communication diagram
- State chart diagram
- Activity diagram
- Interaction overview
- Time sequence diagram

The following table provides a brief description of these diagrams:

Diagram	Description
Use case	Describe the relationships among the functionalities and their internal/external controllers. These controllers are known as actors.
Activity	Describes the flow of control in a system. It consists of activities and links. The flow can be sequential, concurrent or branched
State Machine/state chart	Represent the event driven state change of a system. It basically describes the state change of a class, interface, etc. Used to visualize the reaction of a system by internal/external factors.
Sequence	Visualize the sequence of calls in a system to perform a specific functionality.
Interaction Overview	Combines activity and sequence diagrams to provide a control flow overview of system and business process.
Communication	Same as sequence diagram except that it focuses on the object's role. Each communication is associated with a sequence order number plus the past messages.
Time Sequenced	Describes the changes by messages in state, condition and events.

8.3.2 Architecture View Model

A model is a complete, basic, and simplified description of software architecture which is composed of multiple views from a particular perspective or viewpoint.

A view is a representation of an entire system from the perspective of a related set of concerns. A view is used to describe the system from the viewpoint of different stakeholders such as end-users, developers, project managers, and testers.

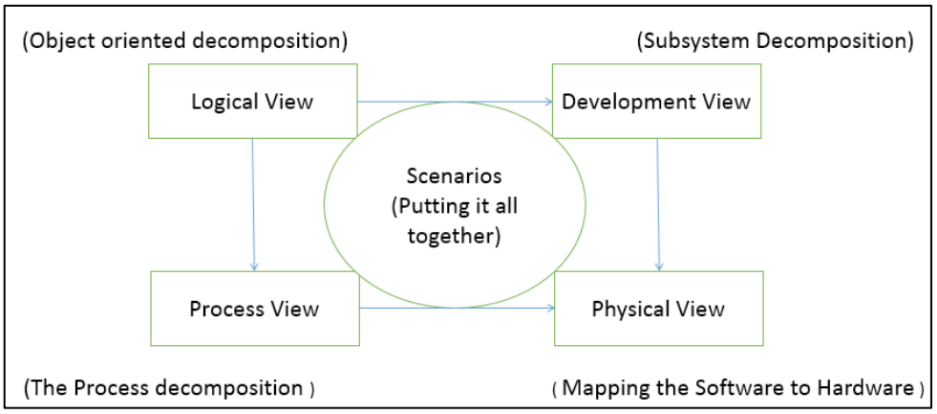
4+1 View Model

The 4+1 View Model was designed by Philippe Kruchten for describing the architecture of a software–intensive system based on the use of multiple and concurrent views. It is a multiple view model that addresses different features and concerns of the system. It standardizes the software design documents and makes the design easy to understand by all stakeholders.

It is an architecture verification method for studying and documenting software architecture design and covers all the aspects of software architecture for all stakeholders. It provides 4 essential views:

- *The logical view or conceptual view:* It describes the object model of the design.
- *The process view:* It describes the activities of the system, captures the concurrency and synchronization aspects of the design.
- *The physical view:* It describes the mapping of software onto hardware and reflects its distributed aspect.
- *The development view:* It describes the static organization or structure of the software in its development environment.

This view model can be extended by adding one more view called scenario view or use case view for end-users or customers of software systems. It is coherent with other four views and are utilized to illustrate the architecture serving as "plus one" view, (4+1) view model. The following figure describes the software architecture using five concurrent views (4+1) model.



Why is it called 4+1 instead of 5?

The use case view has a special significance as it details the high level requirement of system while other views details how those requirements are realized. When all other 4 views are completed, it's effectively redundant. However, all other views would not be possible without it. The following table shows the 4+1 view in detail:

	Logical	Process	Development	Physical	Scenario
Description	Shows the component (Object) of system as well as their interaction	Shows the processes / Workflow rules of system and how those processes communicate, focuses on dynamic view of system	Gives building block views of system and describe static organization of the system modules	Shows the installation, configuration and deployment of software application	Shows the design is complete by performing validation and illustration
Viewer / Stake holder	End-User, Analysts and Designer	Integrators & developers	Programmer and software project managers	System engineer, operators, system administrators and system installers	All the views of their views and evaluators
Consider	Functional requirements	Non Functional Requirements	Software Module organization (Software management reuse, constraint of tools)	Nonfunctional requirement regarding to underlying hardware	System Consistency and validity
UML – Diagram	Class, State, Object, sequence, Communication Diagram	Activity Diagram	Component, Package diagram	Deployment diagram	Use case diagram

8.3.3 Architecture Description Languages (ADLs)

An ADL is a language that provides syntax and semantics for defining a software architecture. It is a notation specification which provides features for modeling a software system's conceptual architecture, distinguished from the system's implementation.

ADLs must support the architecture components, their connections, interfaces, and configurations which are the building block of architecture description. It is a form of expression for use in architecture descriptions and provides the ability to decompose components, combine the components, and define the interfaces of components.

An architecture description language is a formal specification language which describes the software features such as processes, threads, data, and subprograms as well as hardware component such as processors, devices, buses, and memory.

It is hard to classify or differentiate an ADL and a **programming language** or a modeling language. However, there are some requirements for a language to be classified as an ADL:

- It should be appropriate for communicating the architecture to all concerned parties.
- It should be suitable for tasks of architecture creation, refinement, and validation.
- It should provide a basis for further implementation, so it must be able to add information to the ADL specification to enable the final system specification to be derived from the ADL.
- It should have the ability to represent most of the common architectural styles.
- It should support analytical capabilities or provide quick generating prototype implementations.

Keyword

Programming language is a formal language comprising a set of strings that produce various kinds of machine code output.

8.4 OBJECT-ORIENTED PARADIGM

The object-oriented (OO) paradigm took its shape from the initial concept of a new programming approach, while the interest in design and analysis methods came much later. OO analysis and design paradigm is the logical result of the wide adoption of OO programming languages.

- The first object–oriented language was Simula (Simulation of real systems) that was developed in 1960 by researchers at the Norwegian Computing Center.

- In 1970, Alan Kay and his research group at Xerox PARC created a personal computer named Dynabook and the first pure object-oriented programming language (OOPL) - Smalltalk, for programming the Dynabook.
- In the 1980s, Grady Booch published a paper titled Object Oriented Design that mainly presented a design for the programming language, Ada. In the ensuing editions, he extended his ideas to a complete object–oriented design method.
- In the 1990s, Coad incorporated behavioral ideas to object-oriented methods.

The other significant innovations were Object Modeling Techniques (OMT) by James Rum Baugh and Object-Oriented Software Engineering (OOSE) by Ivar Jacobson.

8.4.1 Introduction to Object-Oriented Paradigm

OO paradigm is a significant methodology for the development of any software. Most of the architecture styles or patterns such as pipe and filter, data repository and componentbased etc. can be implemented using this paradigm.

Basic concepts and terminologies of object–oriented systems:

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has:

- Identity that distinguishes it from other objects in the system.
- State that determines characteristic properties of an object as well as values of properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modeled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or the description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, an object is an instance of a class.

The constituents of a class are:

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

Example

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two–dimensional space. The attributes of this class can be identified as follows:

- x–coord, to denote x–coordinate of the center
- y–coord, to denote y–coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows:

- findArea(), a method to calculate area
- findCircumference(), a method to calculate circumference
- scale(), a method to increase or decrease the radius

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.

Keyword

Inheritance is the mechanism of basing an object or class upon another object (prototype-based inheritance) or class (class-based inheritance), retaining similar implementation.

Polymorphism

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing **inheritance**.

Example

Let us consider two classes, Circle and Square, each with a method findArea(). Though the name and purpose of the methods in the classes are same, the internal implementation, i.e., the procedure of calculating area is different for each class. When an object of class Circle invokes its findArea() method, the operation finds the area of the circle without any conflict with the findArea() method of the Square class.Relationships In order to describe a system, both dynamic (behavioral) and static (logical) specification of system must be provided. The dynamic specification describes the relationships among objects. e.g. message passing and static specification describe the relationships among classes, e.g. Aggregation, association and inheritance.

Message Passing

Any application requires a number of objects interacting in a harmonious manner. Objects in a system may communicate with each other using message passing. Suppose a system has two objects: obj1 and obj2. The object obj1 sends a message to object obj2, if obj1 wants obj2 to execute one of its methods.

Composition or Aggregation

Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes. It allows objects to be placed directly within the body of other classes. Aggregation is referred as a "part–of" or "has–a" relationship, with the ability to navigate from the whole to its parts. An aggregate object is an object that is composed of one or more other objects.

Association

Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes. A link can be defined as an instance of an association. The Degree of an association denotes the number of classes involved in a connection. The degree may be unary, binary, or ternary.

- A unary relationship connects objects of the same class.
- A binary relationship connects objects of two classes.
- A ternary relationship connects objects of three or more classes

Inheritance

It is a mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/

parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses.

The subclass can inherit or derive the attributes and methods of the super-class (es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines a "is – a" relationship.

Example

From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow "is – a" mammal.

8.4.2 OO Analysis

In object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified. The aim of OO analysis is to understand the application domain and specific requirements of the system. The result of this phase is requirement specification and initial analysis of logical structure and feasibility of system. The three analysis techniques that are used in conjunction with each other for objectoriented analysis are object modeling, dynamic modeling, and functional modeling.

Object Modeling

Object modeling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modeling can be visualized in the following steps:

- Identify objects and group into classes
- Identify the relationships among classes
- Create a user object model diagram
- Define a user object attributes
- Define the operations that should be performed on the classes

Dynamic Modeling

After the static behavior of the system is analyzed, its behavior with respect to time and external changes needs to be examined. This is the purpose of dynamic modeling. Dynamic Modeling can be defined as "a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world".

The process of dynamic modeling can be visualized in the following steps:

- Identify states of each object
- Identify events and analyze the applicability of actions
- Construct a dynamic model diagram, comprising of state transition diagrams
- Express each state in terms of object attributes
- Validate the state–transition diagrams drawn

Functional Modeling

Functional Modeling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes as it moves between methods. It specifies the meaning of the operations of object modeling and the actions of dynamic modeling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modeling can be visualized in the following steps:

- Identify all the inputs and outputs
- Construct data flow diagrams showing functional dependencies
- State the purpose of each function
- Identify the constraints
- Specify optimization criteria

8.4.3 Object-Oriented Design

After the analysis phase, the conceptual model is developed further into an object-oriented model using object-oriented design (OOD). In OOD, the technology-independent concepts in the analysis model are mapped onto implementing classes, constraints are identified, and interfaces are designed, resulting in a model for the solution domain. The main aim of OO design is to develop the structural architecture of a system.

The stages for object– oriented design can be identified as:

- Defining the context of the system

- Designing the system architecture
- Identification of the objects in the system
- Construction of design models
- Specification of object interfaces

OO Design can be divided into two stages: Conceptual design and Detailed design.

Conceptual design

In this stage, all the classes are identified that are needed for building the system and specific responsibilities are assigned to each class. Class diagram is used to clarify the relationships among classes, and interaction diagram are also used to show the flow of events. It is also known as high-level design.

Detailed design

In this stage, attributes and operations are assigned to each class based on their interaction diagram. State machine diagram are developed to describe the further details of design. It is also known as low-level design.

Design Principles

Principle of Decoupling It is difficult to maintain a system with a set of highly interdependent classes as modification in one class may result in cascading updates of other classes. In an OO design, tight coupling can be eliminated by introducing new classes or inheritance.

Ensuring Cohesion A cohesive class performs a set of closely related functions. A lack of cohesion means a class performs unrelated functions, although it does not affect the operation of the whole system. It makes the entire structure of software hard to manage, expand, maintain, and change.

Open-closed principle According to this principle, the system should be able to extend to meet new requirements. The existing implementation and the code of the system should not be modified as a result of a system expansion. In addition, the following guidelines have to be followed in open-closed principle:

- For each concrete class, separate interface and implementations have to be maintained.
- In a multithreaded environment, keep the attributes private.
- Minimize the use of global variables and class variables.

8.5 DATA FLOW ARCHITECTURE

In data flow architecture, the whole software system is seen as a series of transformations on consecutive pieces or set of input data, where data and operations are independent of each other. In this approach, the data enters the system and then flows through the modules one at a time until they are assigned to some final destination (output or a data store).

The connections between the components or modules may be implemented as I/O stream, I/O buffers, piped or other types of connections. The data can be flown in the graph topology with cycles, in a linear structure without cycles, or in a tree type structure.

The main objective of this approach is to achieve the qualities of reuse and modifiability. It is suitable for applications that involve a well-defined series of independent data transformations or computations on orderly defined input and output such as compilers and business data processing applications. There are three types of execution sequences between modules:

- Batch sequential
- Pipe and filter or non-sequential pipeline mode
- Process control

8.5.1 Batch Sequential

Batch sequential is a classical data processing model in which a data transformation subsystem can initiate its process only after its previous subsystem is completely through.

- The flow of data carries a batch of data as a whole from one subsystem to another.
- The communications between the modules are conducted through temporary intermediate files which can be removed by successive subsystems.
- It is applicable for those applications where data is batched, and each subsystem reads related input files and writes output files.
- Typical application of this architecture includes business data processing such as banking and utility billing.



Advantages

- Provides simpler divisions on subsystems.
- Each subsystem can be an independent program working on input data and producing output data.

Disadvantages

- Provides high latency and low throughput.
- Does not provide concurrency and interactive interface.
- External control is required for implementation.

8.5.2 Pipe and Filter Architecture

This approach lays emphasis on the incremental transformation of data by successive component. In this approach, the flow of data is driven by data and the whole system is decomposed into components of data source, filters, pipes and data sinks.

The connections between modules are data stream which is first-in/first-out buffer that can be stream of bytes, characters or any other type. The main feature of this architecture is its concurrent and incremented execution.

Filter

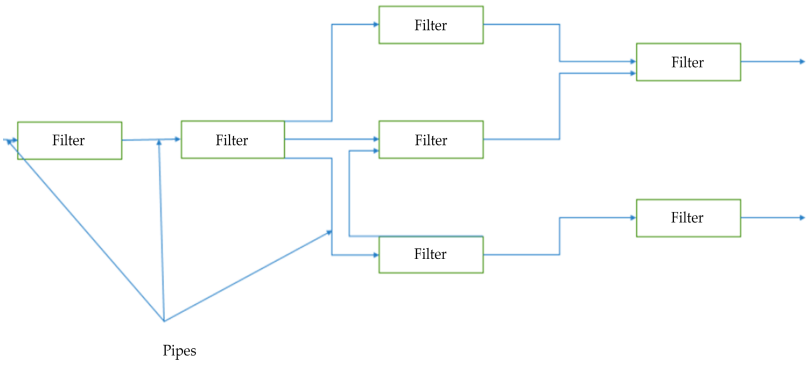
A filter is an independent data stream transformer or stream transducers. It transforms the data of input data stream, processes it, and writes the transformed data stream over a pipe for the next filter to process. It works in an incremental mode in which it start working as soon as data arrives through connected pipe. There are two types of filters: active filter and passive filter.

Active filter

Active filter lets connected pipes to pull data in and pushes out the transformed data. It operates with passive pipe which provide read/write mechanisms for pulling and pushing. This mode is used in UNIX pipe and filter mechanism.

Passive filter

Passive filter lets connected pipes to push data in and pull data out. It operates with active pipe which pulls data from a filter and pushes data into the next filter. It must provide read/write mechanism.



Advantages

- Provides concurrency and high throughput for excessive data processing.
- Provides reusability and simplifies system maintenance.
- Provides modifiability and low coupling between filters.
- Provides simplicity by offering clear divisions between any two filters connected by pipe.
- Provides flexibility by supporting both sequential and parallel execution.

Disadvantages

- Not suitable for dynamic interactions.
- A low common denominator is needed for transmission of data in ASCII formats.
- Overhead of data transformation between filters.
- Does not provide a way for filters to cooperatively interact to solve a problem.
- Difficult to configure this architecture dynamically.

Pipe

Pipes are stateless and they carry binary or character stream which exist between two filters. It can move a data stream from one filter to another. Pipes use little contextual information and retain no state information between instantiations.

8.5.3 Process Control Architecture

It is a type of data flow architecture where data is neither batched sequential nor pipelined stream. The flow of data comes from a set of variables which controls the execution of process. It decomposes the entire system into subsystems or modules and connects them.

Types of Subsystems

A process control architecture would have a processing unit for changing the process control variables and a controller unit for calculating the amount of changes.

A controller unit must have the following elements:

- *Controlled Variable* Controlled Variable values the system it intends to control and should be measured by sensors. For example, speed in cruise control system.
- *Input Variable* Measures an input to the process. For example, temperature of return air in temperature control system
- *Manipulated Variable* Manipulated Variable value is adjusted or changed by the controller.
- *Process Definition* It includes mechanisms for manipulating some process variables.
- *Sensor* Obtains values of process variables pertinent to control and can be used as a feedback reference to recalculate manipulated variables.
- *Set Point* It is the desired value for a controlled variable.
- *Control Algorithm* For deciding how to manipulate process variables.

Application Areas

Process control architecture is suitable in the following domains:

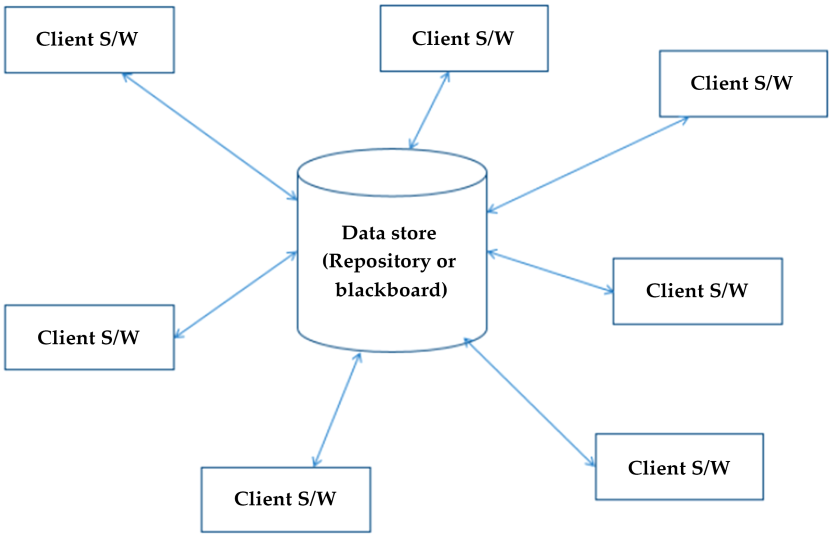
- Embedded system software design where the system is manipulated by process control variable data
- Applications whose aim is to maintain specified properties of the outputs of the process at given reference values.
- Applicable for car-cruise control and building temperature control systems.
- Real-time system software to control automobile anti-lock brakes, nuclear power plants, etc.

8.6 DATA-CENTERED ARCHITECTURE

In data-centered architecture, the data is centralized and accessed frequently by other components that modify data. The main purpose of this style is to achieve integrality of data. Data-centered architecture consists of different components that communicate through shared data repositories. The components access a shared data structure and are relatively independent, in that, they interact only through the data store.

The most well-known examples of the data-centered architecture is a database architecture in which the common database schema is created with data definition

protocol – for example, a set of related tables with fields and data types in an RDBMS. Another example of data-centered architectures is the web architecture which has a common data schema (i.e. meta-structure of the Web) and follows hypermedia data model and processes communicate through the use of shared web-based data services.



8.6.1 Types of Components

There are two types of components:

- A central data structure or data store or data repository which is responsible for providing permanent data storage. It represents the current state.
- A data accessor or a collection of independent components that operate on the central data store, perform computations, and might put back the results.

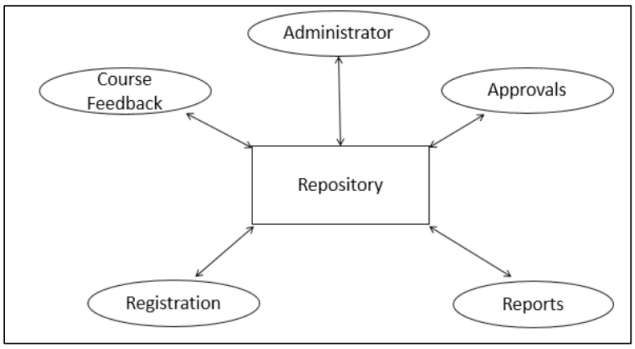
Interactions or communication between the data accessors is only through the data store. The data is the only means of communication among clients. The flow of control differentiates the architecture into two categories:

- Repository Architecture Style
- Blackboard Architecture Style

8.6.2 Repository Architecture Style

In Repository Architecture Style, the data store is passive and the clients (software components or agents) of the data store are active which control the logic flow. The participating components check the data-store for changes.

- The client sends a request to the system to perform actions (e.g. insert data).
- The computational processes are independent and triggered by incoming requests.
- If the types of transactions in an input stream of transactions trigger selection of processes to execute, then it is traditional database or repository architecture, or passive repository.
- This approach is widely used in DBMS, library information system, the interface repository in CORBA, compilers and CASE (computer aided software engineering) environments.



Advantages

- Provides data integrity, backup and restore features.
- Provides scalability and reusability of agents as they do not have direct communication with each other.
- Reduces overhead of transient data between software components.

Disadvantages

- It is more vulnerable to failure and data replication or duplication is possible.
- High dependency between data structure of data store and its agents.
- Changes in data structure highly affect the clients.
- Evolution of data is difficult and expensive.
- Cost of moving data on network for distributed data.

REMEMBER

Sometimes data can be passed with the event. But in other situations event systems must rely on a shared repository for interaction.

8.6.3 Blackboard Architecture Style

In Blackboard Architecture Style, the data store is active and its clients are passive. Therefore the logical flow is determined by the current data status in data store. It has a blackboard component, acting as a central data repository, and an internal representation is built and acted upon by different computational elements.

- A number of components that act independently on the common data structure are stored in the blackboard.
- In this style, the components interact only through the blackboard. The data-store alerts the clients whenever there is a data-store change.
- The current state of the solution is stored in the blackboard and processing is triggered by the state of the blackboard.
- The system sends notifications known as trigger and data to the clients when changes occur in the data.
- This approach is found in certain AI applications and complex applications, such as speech recognition, image recognition, security system, and business resource management systems etc.
- If the current state of the central data structure is the main trigger of selecting processes to execute, the repository can be a blackboard and this shared data source is an active agent.
- A major difference with traditional database systems is that the invocation of computational elements in a blackboard architecture is triggered by the current state of the blackboard, and not by external inputs.

Parts of Blackboard Model

The blackboard model is usually presented with three major parts:

Knowledge Sources (KS)

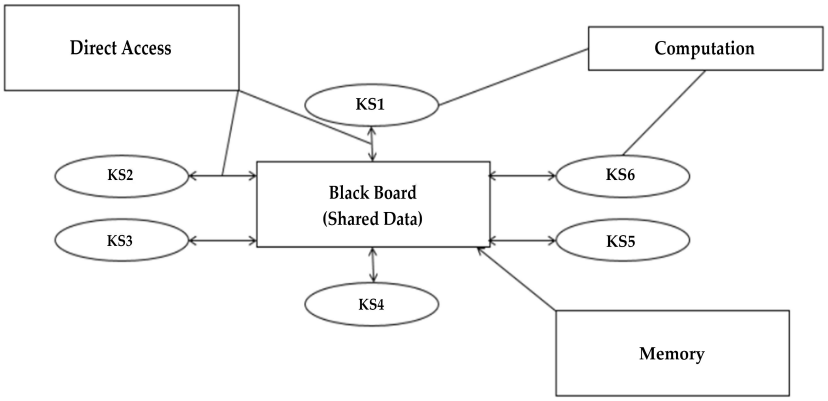
Knowledge Sources, also known as Listeners or Subscribers are distinct and independent units. They solve parts of a problem and aggregate partial results. Interaction among knowledge sources takes place uniquely through the blackboard.

Blackboard Data Structure

The problem-solving state data is organized into an application-dependent hierarchy. Knowledge sources make changes to the blackboard that lead incrementally to a solution to the problem.

Control

Control manages tasks and checks the work state.



Advantages

- Provides scalability which provides easy to add or update knowledge source.
- Provides concurrency that allows all knowledge sources to work in parallel as they are independent of each other.
- Supports experimentation for hypotheses.
- Supports reusability of knowledge source agents.

Disadvantages

- The structure change of blackboard may have a significant impact on all of its agents as close dependency exists between blackboard and knowledge source.
- It can be difficult to decide when to terminate the reasoning as only approximate solution is expected.
- Problems in synchronization of multiple agents.
- Major challenges in designing and testing of system.

8.7 HIERARCHICAL ARCHITECTURE

Hierarchical architecture views the whole system as a hierarchy structure in which the software system is decomposed into logical modules or subsystems at different levels in the hierarchy. This approach is typically used in designing system software such as network protocols and operating systems.

In system software hierarchy design, a low-level subsystem gives services to its adjacent upper level subsystems which invoke the methods in the lower level. The lower layer provides more specific functionality such as I/O services, transaction, scheduling, security services etc.; the middle layer provides more domain dependent functions such as business logic and core processing services. The upper layer provides more abstract functionality in the form of user interface such as GUIs, shell programming facilities, etc.

It is also used in organization of class libraries such as .NET class library in namespace hierarchy. All the design types can implement this hierarchical architecture and often combine with other architecture styles. Hierarchical architectural styles include the following:

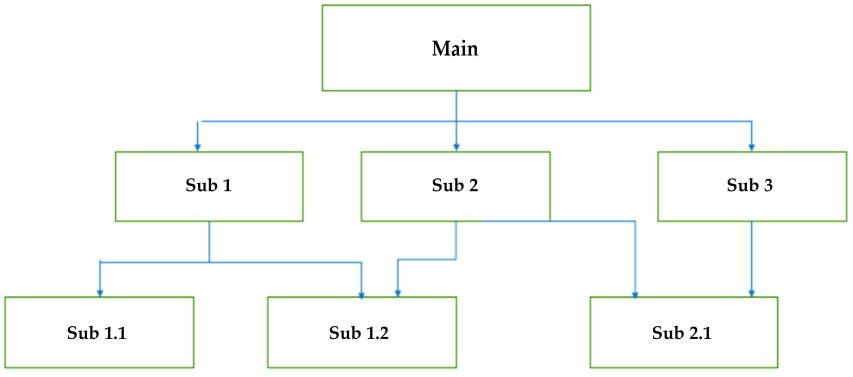
- Main-subroutine
- Master-slave
- Virtual machine

8.7.1 Main-subroutine

The aim of this style is to reuse the modules and freely develop individual modules or subroutine. In this style, a software system is divided into subroutines by using top-down refinement according to desired functionality of the system.

These refinements lead vertically until the decomposed modules is simple enough to have its exclusive independent responsibility. Functionality may be reused and shared by multiple callers in the upper layers. There are two ways by which data is passed as parameters to subroutines from callers:

- *Pass by Value* Subroutines only use the past data but can't modify it.
- *Pass by Reference* Subroutines use as well as changes the value of the data referenced by the parameter.



Main Subroutine

Advantages

- Easy to decompose the system based on hierarchy refinement.
- Can be used in a subsystem of object oriented design.

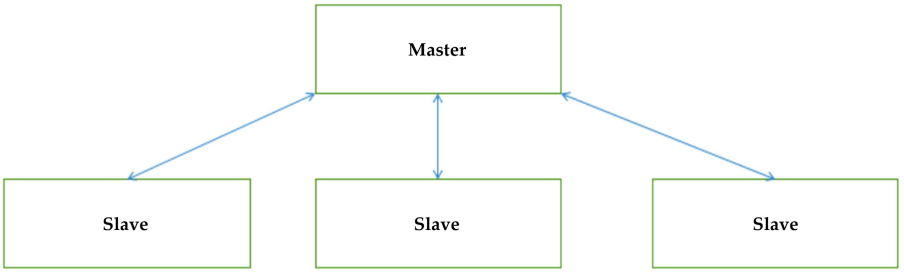
Disadvantages

- Vulnerable as it contains globally shared data.
- Tight coupling may cause more ripple effects of changes.

8.7.2 Master-Slave

This approach applies the 'divide and conquer' principle and supports fault computation and computational accuracy. It is a modification of the main-subroutine architecture that provides reliability of system and fault tolerance.

In this architecture, slaves provide duplicated services to the master, and the master chooses a particular result among slaves by a certain selection strategy. The slaves may perform the same functional task by different algorithms and methods or totally different functionality. It includes parallel computing in which all the slaves can be executed in parallel.



The implementation of the Master-Slave pattern follows five steps:

1. Specify how the computation of the task can be divided into a set of equal subtasks and identify the sub-services that are needed to process a sub-task.
2. Specify how the final result of the whole service can be computed with the help of the results obtained from processing individual sub-tasks.
3. Define an interface for the sub-service identified in step 1. It will be implemented by the slave and used by the master to delegate the processing of individual subtasks.
4. Implement the slave components according to the specifications developed in the previous step.

5. Implement the master according to the specifications developed in step 1 to 3.

Applications

- Suitable for applications where reliability of software is critical issue.
- Widely applied in the areas of parallel and distributed computing.

Advantages

- Faster computation and easy scalability.
- Provides robustness as slaves can be duplicated.
- Slave can be implemented differently to minimize semantic errors.

Disadvantages

- Communication overhead.
- Not all problems can be divided.
- Hard to implement and portability issue.

8.7.3 Virtual Machine Architecture

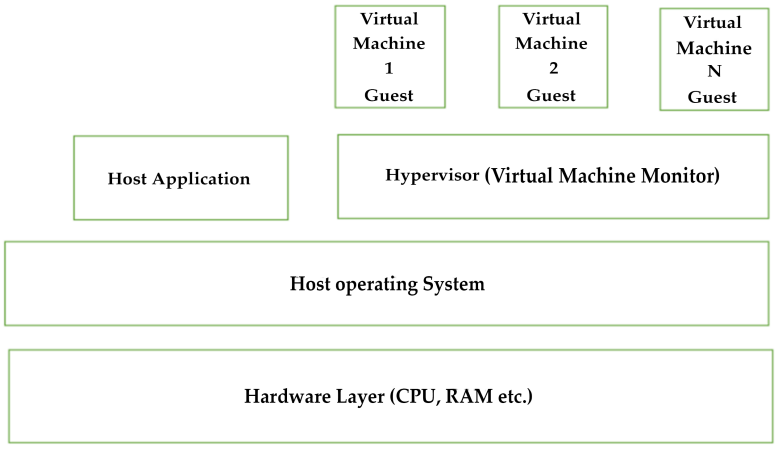
Virtual Machine architecture pretends some functionality which is not native to the hardware and/or software on which it is implemented. A virtual machine is built upon an existing system and provides a virtual abstraction, a set of attributes, and operations.

In virtual machine architecture, the master uses the 'same' subservice' from the slave and performs functions such as split work, call slaves, and combine results. It allows developers to simulate and test platforms that have not yet been built, and simulate "disaster'' modes that would be too complex, costly, or dangerous to test with the real system.

In most cases, a virtual machine splits a programming language or application environment from an execution platform. The main objective is to provide portability. Interpretation of a particular module via a Virtual Machine may be perceived as follows:

- The interpretation engine chooses an instruction from the module being interpreted.
- Based on the instruction, the engine updates the virtual machine's internal state and the above process is repeated.

The following figure shows the architecture of a standard VM infrastructure on a single physical machine.



Virtual Machine Architecture

The hypervisor, also called the virtual machine monitor, runs on the host OS and allocates matched resources to each guest OS. When the guest makes a system-call, the hypervisor intercepts and translates it into the corresponding system-call supported by the host OS. The hypervisor controls each virtual machine access to the CPU, memory, persistent storage, I/O devices, and the network.

Applications

Virtual machine architecture is suitable in the following domains:

- Suitable for solving a problem by simulation or translation if there is no direct solution.
- Sample applications include interpreters of microprogramming, XML processing, script command language execution, rule-based system execution, Smalltalk and Java interpreter typed programming language
- Common examples of virtual machines are interpreters, rule-based systems, syntactic shells, and command language processors.

Advantages

- Portability and machine platform independency.
- Simplicity of software development.
- Provides flexibility through the ability to interrupt and query the program.
- Simulation for disaster working model.
- Introduce modifications at runtime.

Disadvantages

- Slow execution of the interpreter due to the interpreter nature.
- There is a performance cost because of the additional computation involved in execution.

8.7.4 Layered Style

In this approach, the system is decomposed into a number of higher and lower layers in a hierarchy and each layer has its own sole responsibility in the system.

- Each layer consists of a group of related classes that are encapsulated in a package, in a deployed component, or as a group of subroutines in the format of method library or header file.
- Each layer provides service to the layer above it and serves as a client to the layer below i.e. request to layer i +1 invokes the services provided by the layer i via the interface of layer i. The response may go back to the layer i +1 if the task is completed; otherwise layer i continually invokes services from layer i -1 below.

Layered style is suitable in the following areas:

- Applications that involve distinct classes of services that can be organized hierarchically.
- Any application that can be decomposed into application-specific and platformspecific portions.
- Applications that have clear divisions between core services, critical services, and user interface services, etc.

Advantages

- Design based on incremental levels of abstraction.
- Provides enhancement independence as changes to the function of one layer affects at most two other layers.
- Separation of the standard interface and its implementation.
- Implemented by using component-based technology which makes the system much easier to allow for plug-and-play of new components.
- Each layer can be an abstract machine deployed independently which support portability.
- Easy to decompose the system based on the definition of the tasks in a top-down refinement manner

- Different implementations (with identical interfaces) of the same layer can be used interchangeably

Disadvantages

- Many applications or systems are not easily structured in a layered fashion.
- Lower runtime performance since a client's request or a response to client must go through potentially several layers.
- There are also performance concerns on overhead on the data marshaling and buffering by each layer.
- Opening of interlayer communication may cause deadlocks and "bridging" may cause tight coupling.
- Exceptions and error handling is an issue in the layered architecture, since faults in one layer must spread upwards to all calling layers

8.8 INTERACTION-ORIENTED ARCHITECTURE

The primary objective of interaction-oriented architecture is to separate the interaction of user from data abstraction and business data processing. The interaction-oriented software architecture decomposes the system into three major partitions:

- *Data module* Data module provides the data abstraction and all business logic.
- *Control module* Control module identifies the flow of control and system configuration actions.
- *View presentation module* View presentation module is responsible for visual or audio presentation of data output and it also provides an interface for user input.

Interaction-oriented architecture has two major styles: Model-View-Controller (MVC) and Presentation-Abstraction-Control (PAC). Both MVC and PAC propose three components decomposition and are used for interactive applications such as web applications with multiple talks and user interactions. They are different in their flow of control and organization. PAC is an agent-based hierarchical architecture but MVC does not have a clear hierarchical structure.

8.8.1 Model-View-Controller (MVC)

MVC decomposes a given software application into three interconnected parts that help in separating the internal representations of information from the information presented to or accepted from the user.

Module	Function
Model	Encapsulation the underlying data and business logic
Controller	Respond to user action and direct the application flow
View	Formats and present the data from model to user.

Model

Model is a central component of MVC that directly manages the data, logic, and constraints of an application. It consists of data components which maintain the raw application data and application logic for interface.

- It is an independent user interface and captures the behavior of application problem domain.
- It is the domain-specific software simulation or implementation of the application's central structure.
- When there has been change in its state, it gives notification to its associated view to produce updated output and the controller to change the available set of commands.

View

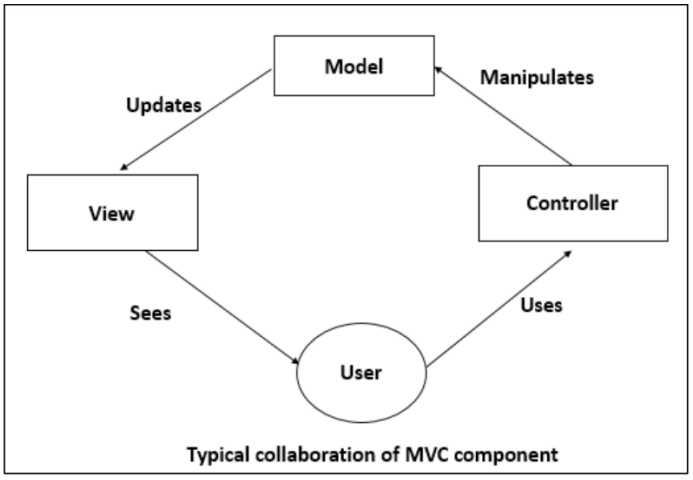
View can be used to represent any output of information in graphical form such as diagram or chart. It consists of presentation components which provide the visual representations of data

- Views request information from their model and generate an output representation to the user.
- Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

Controller

A controller accepts an input and converts it to commands for the model or view. It consists of input processing components which handle input from the user by modifying the model.

- It acts as an interface between the associated models and views and the input devices.
- It can send commands to the model to update the model's state and to its associated view to change the view's presentation of the model.



Typical collaboration of MVC component

MVC - I

It is a simple version of MVC architecture where the system is divided into two subsystems:

- *The Controller-View* The controller-view acts as input /output interface and processing is done.
- *The Model* The model provides all the data and domain services.

MVC-I Architecture

The model module notifies controller-view module of any data changes so that any graphics data display will be changed accordingly; the controller also takes appropriate action upon the changes.



MVC- I Architecture

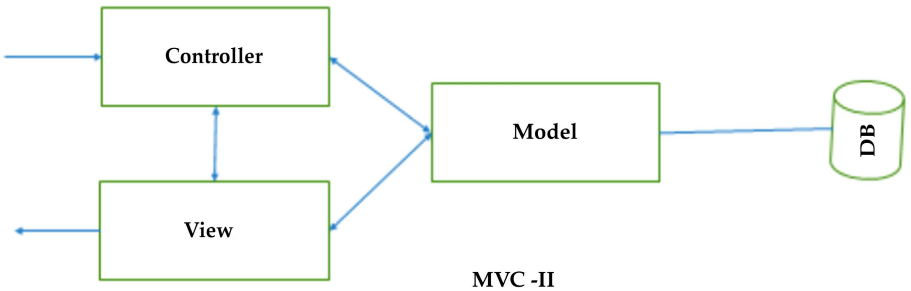
The connection between controller-view and model can be designed in a pattern of subscribe-notify whereby the controller-view subscribes to model and model notifies controller-view of any changes.

MVC - II

MVC–II is an enhancement of MVC-I architecture in which the view module and the controller module are separate. The model module plays an active role as in MVC-I by providing all the core functionality and data supported by database.

The view module presents data while controller module accepts input request, validates input data, initiates the model, the view, their connection, and also dispatches the task.

MVC-II Architecture



MVC -II

MVC Applications

MVC applications are effective for interactive applications where multiple views are needed for a single data model and easy to plug-in new or change interface view.

MVC applications are suitable for applications where there are clear divisions between the modules so that different professionals can be assigned to work on different aspects of such applications concurrently.

Advantages

- There are many MVC vendor framework toolkits available.
- Multiple views synchronized with same data model.
- Easy to plug-in new or replace interface views.
- Used for application development where graphics expertise professionals, programming professionals, and data base development professionals are working in a designed project team.

Disadvantages

- Not suitable for agent-oriented applications such as interactive mobile and robotics applications.
- Multiple pairs of controllers and views based on the same data model make any data model change expensive.
- The division between the View and the Controller is not clear in some cases.

8.8.2 Presentation-Abstraction-Control (PAC)

In PAC, the system is divided into a hierarchy of many cooperating agents (triads). It was developed from MVC to support the application requirement of multiple agents in addition to interactive requirements.

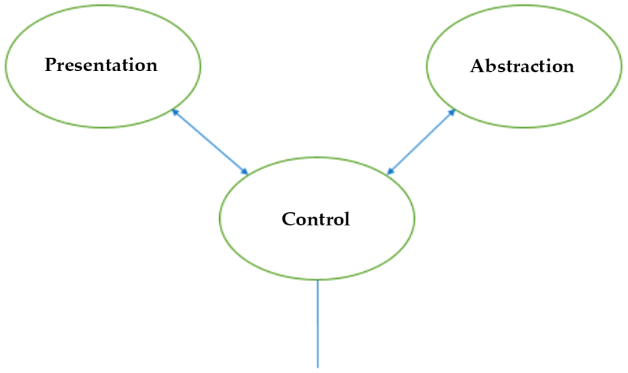
Each agent has three components:

- *The presentation component:* Formats the visual and audio presentation of data.
- *The abstraction component:* Retrieves and processes the data.
- *The control component:* Handles the task such as the flow of control and communication between the other two components.

The PAC architecture is similar to MVC, in the sense that presentation module is like view module of MVC. The abstraction module looks like model module of MVC and the control module is like the controller module of MVC but they differ in their flow of control and organization.

There are no direct connections between abstraction component and presentation component in each agent. The control component in each agent is in charge of communications with other agents.

The following figure shows a block diagram for a single agent in PAC design.



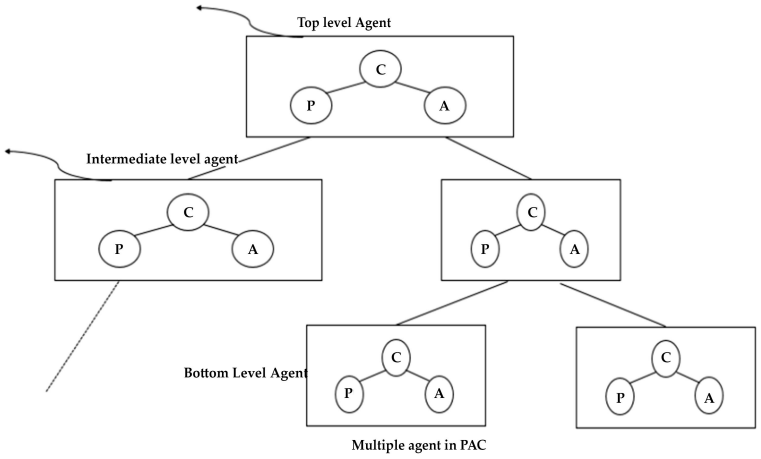
Single Agent in PAC Design

PAC with Multiple Agents

In PACs consisting of multiple agents, the top-level agent provides core data and business logics. The bottom level agents define detailed specific data and presentations. The intermediate level or middle level agent acts as coordinator of low-level agents.

- Each agent has its own specific assigned job.
- For some middle level agents the interactive presentations are not required, so they do not have a presentation component.
- The control component is required for all agents through which all the agents communicate with each other.

The following figure shows the Multiple Agents that take part in PAC.



Multiple agent in PAC

Applications

- Effective for an interactive system where the system can be decomposed into many cooperating agents in a hierarchical manner.
- Effective when the coupling among the agents is expected to be loose so that changes on an agent does not affect others.
- Effective for distributed system where all the agents are distantly distributed and each of them has its own functionalities with data and interactive interface.
- Suitable for applications with rich GUI components where each of them keeps its own current data and interactive interface and needs to communicate with other components.

Advantages

- Support for multi-tasking and multi-viewing
- Support for agent reusability and extensibility
- Easy to plug-in new agent or change an existing one
- Support for concurrency where multiple agents are running in parallel in different threads or different devices or computers

Disadvantages

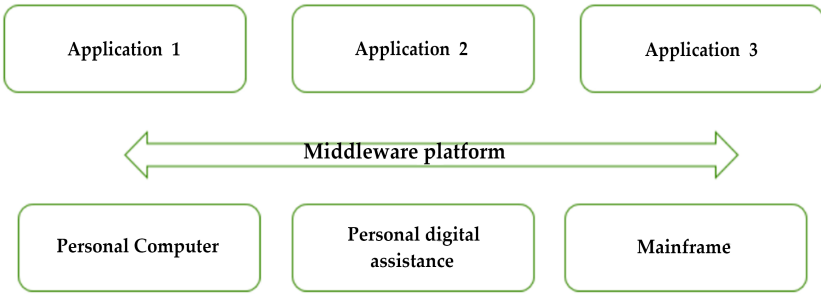
- Overhead due to the control bridge between presentation and abstraction and the communication of controls among agents
- Difficult to determine the right number of the agents due to the loose coupling and high independence between agents
- Complete separation of presentation and abstraction by control in each agent generate development complexity since communications between agents only take place between the controls of agents

8.9 DISTRIBUTED ARCHITECTURE

In distributed architecture, components are presented on different platforms and several components can cooperate with one another over a communication network in order to achieve a specific objective or goal.

- In this architecture, information processing is not confined to a single machine rather it is distributed over several independent computers.
- A distributed system can be demonstrated by the client-server architecture which forms the base for multi-tier architectures; alternatives are the broker architecture such as CORBA, and the Service-Oriented Architecture (SOA).
- There are several technology frameworks to support distributed architectures, including .NET, J2EE, CORBA, .NET Web services, AXIS Java Web services, and Globus Grid services.
- Middleware is an infrastructure that appropriately supports the development and execution of distributed applications. It provides a buffer between the applications and the network.
- It sits in the middle of system and manages or supports the different components of a distributed system. Examples are transaction processing monitors, data convertors and communication controllers etc.

Middleware as an infrastructure for distributed system



Middleware as an infrastructure for distributed system

The basis of a distributed architecture is its transparency, reliability, and availability.

The following table lists the different forms of transparency in a distributed system:

Transparency	Description
Access	Hides the way in which resources are accessed and the differences in data platform
Location	Hides where resources are located
Technology	Hides different technologies such as programming language and OS from user
Migration / Relocation	Hide resources that may be moved to another location which are in use
Replication	Hide resources that may be copied at several location
Concurrency	Hide resources that may be shared with other users
Failure	Hides failure and recovery of resources from user
Persistence	Hides whether a resource (software) is in memory or disk

Advantages

- *Resource sharing*: Sharing of hardware and software resources.
- *Openness:* Use of equipment and software different vendors

- *Concurrency:* Concurrent processing to enhance performance.
- *Scalability*: Increased throughput by adding new resources
- *Fault tolerance*: The ability to continue in operation after a fault has occurred.

Disadvantages

- *Complexity*: They are more complex than centralized systems.
- *Security*: More susceptible to external attack.
- *Manageability*: More effort required for system management.
- *Unpredictability*: Unpredictable responses depending on the system organization and network load.

Centralized System vs. Distributed System

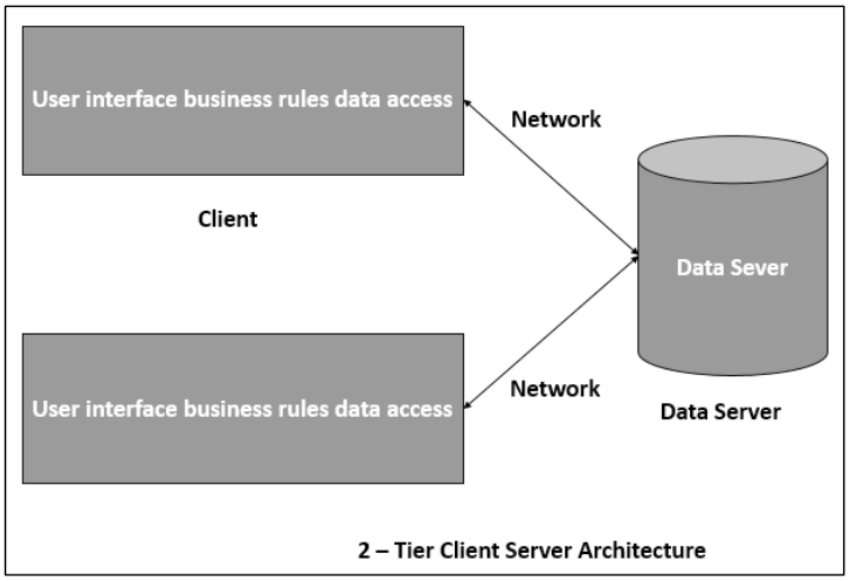
Criteria	Centralized system	Distributed System
Economics	Low	High
Availability	Low	High
Complexity	Low	High
Consistency	Simple	High
Scalability	Poor	Good
Technology	Homogeneous	Heterogeneous
Security	High	Low

8.9.1 Client-Server Architecture

The client-server architecture is the most common distributed system architecture which decomposes the system into two major subsystems or logical processes:

- ***Client***: This is the first process that issues a request to the second process i.e., the server.
- ***Server***: This is the second process that receives the request, carries it out, and sends a reply to the client.

In this architecture, the application is modelled as a set of services that are provided by servers and a set of clients that use these services. The servers need not know about clients, but the clients must know the identity of servers, and the mapping of processors to processes is not necessarily 1 : 1



Client-server Architecture can be classified into two models based on the functionality of the client:

Thin-client model

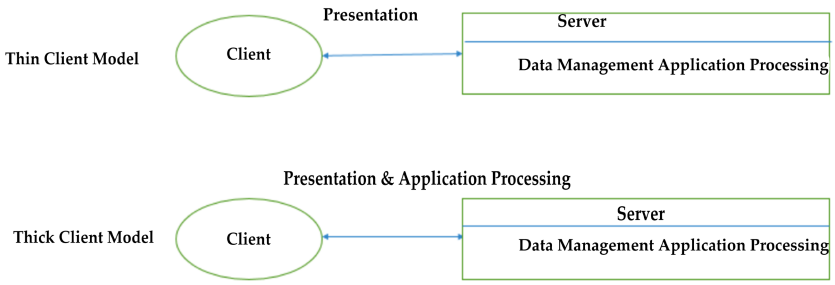
In thin-client model, all the application processing and data management is carried by the server. The client is simply responsible for running the presentation software.

- Used when legacy systems are migrated to client server architectures in which legacy system acts as a server in its own right with a graphical interface implemented on a client
- A major disadvantage is that it places a heavy processing load on both the server and the network.

Thick/Fat-client model

In thick-client model, the server is only in charge for data management. The software on the client implements the application logic and the interactions with the system user.

- Most appropriate for new C/S systems where the capabilities of the client system are known in advance
- More complex than a thin client model especially for management. New versions of the application have to be installed on all clients.



Advantages

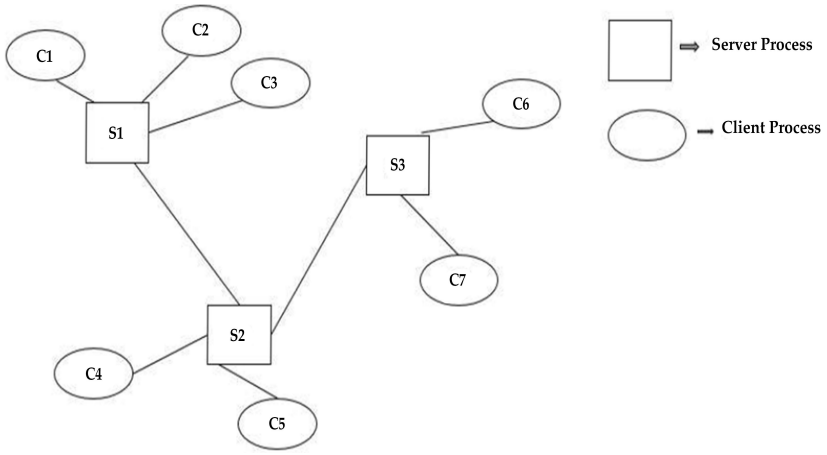
- Separation of responsibilities such as user interface presentation and business logic processing.
- Reusability of server components and potential for concurrency
- Simplifies the design and the development of distributed applications.
- It makes it easy to migrate or integrate existing applications into a distributed environment.
- It also makes effective use of resources when a large number of clients are accessing a high-performance server.

Disadvantages

- Lack of heterogeneous infrastructure to deal with the requirement changes.
- Security complications.
- Limited server availability and reliability.
- Limited testability and scalability.
- Fat clients with presentation and business logic together.

8.9.2 Multi-Tier Architecture (n-tier Architecture)

Multi-tier architecture is a client–server architecture in which the functions such as presentation, application processing, and data management are physically separated. By separating an application into tiers, developers obtain the option of changing or adding a specific layer, instead of reworking the entire application. It provides a model by which developers can create flexible and reusable applications.



The most general use of multi-tier architecture is the three-tier architecture. A three-tier architecture is typically composed of a presentation tier, an application tier, and a data storage tier and may execute on a separate processor.

Presentation Tier

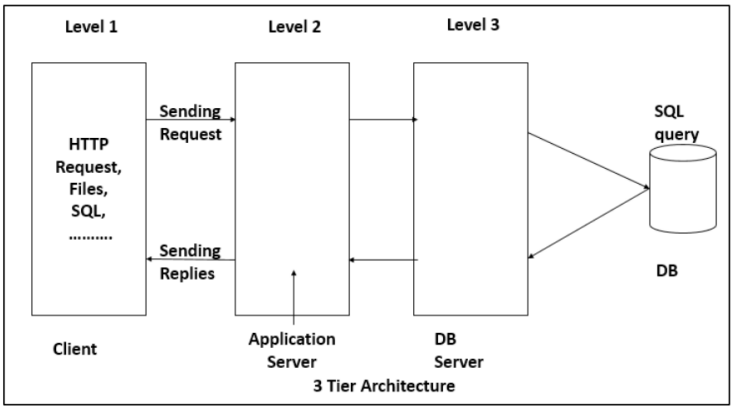
Presentation layer is the topmost level of the application by which users can access directly such as webpage or Operating System GUI (Graphical User interface). The primary function of this layer is to translate the tasks and results to something that user can understand. It communicates with other tiers so that it places the results to the browser/client tier and all other tiers in the network.

Application Tier (Business Logic, Logic Tier, or Middle Tier)

Application tier coordinates the application, processes the commands, makes logical decisions, evaluation, and performs calculations. It controls an application's functionality by performing detailed processing. It also moves and processes data between the two surrounding layers.

Data Tier

In this layer, information is stored and retrieved from the database or file system. Then this information is then passed back for processing and then back to the user. It includes the data persistence mechanisms (database servers, file shares, etc.) and provides API (application programming Interface) to the application tier which provides methods of managing the stored data.



Advantages

- Better performance than a thin-client approach and is simpler to manage than a thick-client approach.
- Enhances the reusability and scalability - as demands increase, extra servers can be added.
- Provides multi-threading support and also reduces network traffic.
- Provides maintainability and flexibility

Disadvantages

- Unsatisfactory Testability due to lack of testing tools.
- More critical server reliability and availability.

8.9.3 Broker Architectural Style

Broker Architectural Style is a middleware architecture used in distributed computing to coordinate and enable the communication between registered servers and clients. Here, object communication takes place through a middleware system called an object request broker (software bus).

- Client and the server do not interact with each other directly. Client and server have a direct connection to its proxy which communicates with the mediatorbroker.
- A server provides services by registering and publishing their interfaces with the broker and clients can request the services from the broker statically or dynamically by look-up.
- CORBA (Common Object Request Broker Architecture) is a good implementation example of the broker architecture.

Components of Broker Architectural Style

The components of broker architectural style are discussed below.

Broker

Broker is responsible for coordinating communication, such as forwarding and dispatching the results and exceptions. It can be either an invocation-oriented service, a document or message - oriented broker to which clients send a message.

- It is responsible for brokering the service requests, locating a proper server, transmitting requests, and sending responses back to clients.
- It retains the servers' registration information including their functionality and services as well as location information.
- It provides APIs for clients to request, servers to respond, registering or unregistering server components, transferring messages, and locating servers.

Stub

Stubs are generated at the static compilation time and then deployed to the client side which is used as a proxy for the client. Client-side proxy acts as a mediator between the client and the broker and provides additional transparency between them and the client; a remote object appears like a local one.

The proxy hides the IPC (inter-process communication) at protocol level and performs marshaling of parameter values and un-marshaling of results from the server.

Skeleton

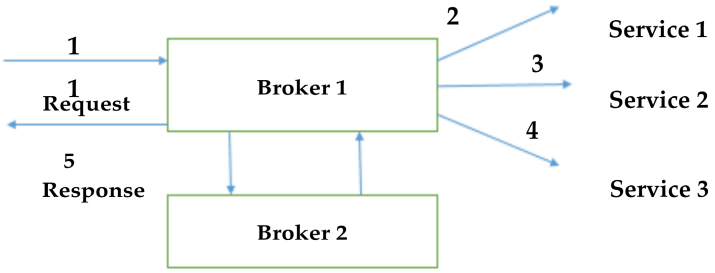
Skeleton is generated by the service interface compilation and then deployed to the server side which is used as a proxy for the server. Server-side proxy encapsulates low-level system-specific networking functions and provides high-level APIs to mediate between the server and the broker.

It receives the requests, unpacks the requests, unmarshals the method arguments, calls the suitable service, and also marshals the result before sending it back to the client.

Bridge

A bridge can connect two different networks based on different communication protocols. It mediates different brokers such as DCOM, .NET remote and Java CORBA brokers.

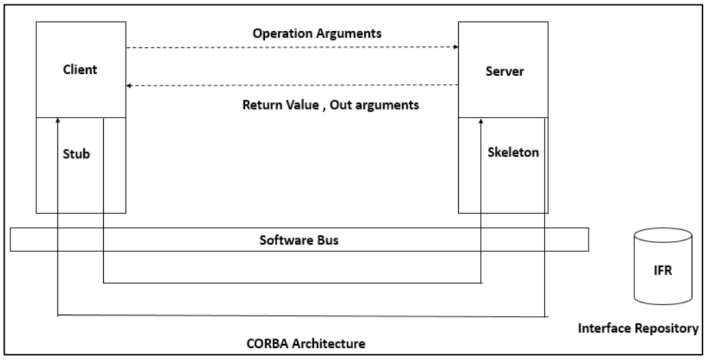
Bridges are optional component which hides the implementation details when two brokers interoperate and take requests and parameters in one format and translate them to another format.



Broker Model

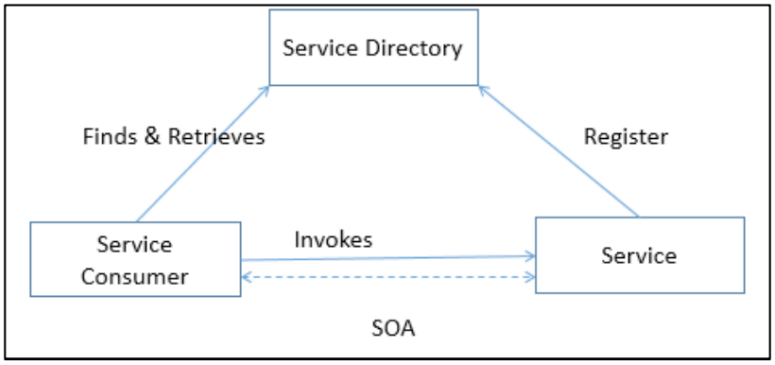
Broker implementation in CORBA

CORBA is an international standard for an Object Request Broker – a middleware to manage communications between distributed objects defined by OMG (object management group)



8.9.4 Service-Oriented Architecture (SOA)

A service is a component of business functionality that is well-defined, self-contained, independent, published, and available to be used via a standard programming interface. The connections between services are conducted by common and universal messageoriented protocols such as the SOAP Web service protocol, which can deliver requests and responses between services loosely. Service-oriented architecture is a client/server design which support business-driven IT approach in which an application consists of software services and software service consumers (also known as clients or service requesters).



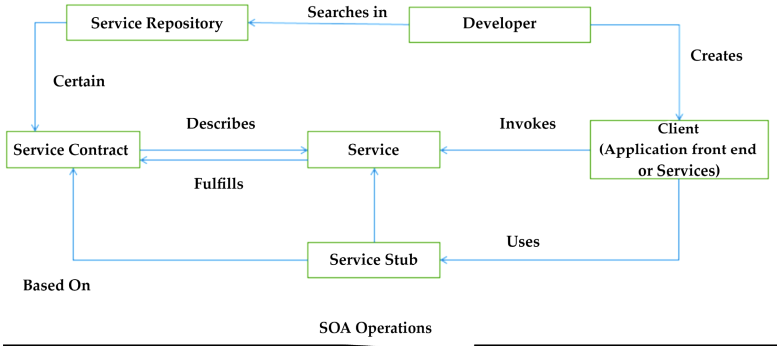
Features of SOA

A service-oriented architecture provides the following features:

- *Distributed Deployment* Expose enterprise data and business logic as loosely, coupled, discoverable, structured, standard-based, coarse-grained, stateless units of functionality called services.
- *Composability* Assemble new processes from existing services that are exposed at a desired granularity through well defined, published, and standard complaint interfaces.
- *Interoperability* Share capabilities and reuse shared services across a network irrespective of underlying protocols or implementation technology.
- *Reusability* Choose a service provider and access to existing resources exposed as services.

SOA Operation

The following figure illustrates how SOA operates:



Advantages

- Loose coupling of service–orientation provides great flexibility for enterprises to make use of all available service recourses irrespective of platform and technology restrictions.
- Each service component is independent from other services due to the stateless service feature.
- The implementation of a service will not affect the application of the service as long as the exposed interface is not changed.
- A client or any service can access other services regardless of their platform, technology, vendors, or language implementations.
- Reusability of assets and services since clients of a service only need to know its public interfaces, service composition.
- SOA based business application development are much more efficient in terms of time and cost.
- Enhances the scalability and provide standard connection between systems.
- Efficient and effective usage of 'Business Services'.
- Integration becomes much easier and improved intrinsic interoperability.
- Abstract complexity for developers and energize business processes closer to end users

8.10 COMPONENT-BASED ARCHITECTURE

Component-based architecture focuses on the decomposition of the design into individual functional or logical components that represent well-defined communication interfaces containing methods, events, and properties. It provides a higher level of abstraction and divides the problem into sub-problems, each associated with component partitions. The primary objective of component-based architecture is to ensure component reusability. A component encapsulates functionality and behaviors of a software element into a reusable and self-deployable binary unit. There are many standard component frameworks such as COM/DCOM, JavaBean, EJB, CORBA, .NET, web services, and grid services. These technologies are widely used in local desktop GUI application design such as graphic JavaBean components, MS ActiveX components, and COM components which can be reused by simply drag and drop operation.

Component-oriented software design has many advantages over the traditional objectoriented approaches such as:

- Reduced time in market and the development cost by reusing existing components.
- Increased reliability with the reuse of the existing components.

8.10.1 What is a Component?

A component is a modular, portable, replaceable, and reusable set of well-defined functionality that encapsulates its implementation and exporting it as a higher-level interface.

A component is a software object, intended to interact with other components, encapsulating certain functionality or a set of functionalities. It has an obviously defined interface and conforms to a recommended behavior common to all components within an architecture.

A software component can be defined as a unit of composition with a contractually specified interface and explicit context dependencies only. That is, a software component can be deployed independently and is subject to composition by third parties.

Views of a Component

A component can have three different views: object-oriented view, conventional view, and process-related view.

Object-oriented view

A component is viewed as a set of one or more cooperating classes. Each problem domain class (analysis) and infrastructure class (design) is explained to identify all attributes and operations that apply to its implementation. It also involves defining the interfaces that enable classes to communicate and cooperate.

Conventional view

It is viewed as a functional element or a module of a program that integrates the processing logic, the internal data structures that are required to implement the processing logic and an interface that enables the component to be invoked and data to be passed to it.

Process-related view

In this view, instead of creating each component from scratch, the system is building from existing components maintained in a library. As the software architecture is formulated, components are selected from the library and used to populate the architecture.

- A user interface (UI) component includes grids, buttons referred as controls, and utility components expose a specific subset of functions used in other components.

- Other common types of components are those that are resource intensive, not frequently accessed, and must be activated using the just-in-time (JIT) approach
- Many components are invisible which are distributed in enterprise business applications and internet web applications such as Enterprise JavaBean (EJB), .NET components, and CORBA components.

Characteristics of Components

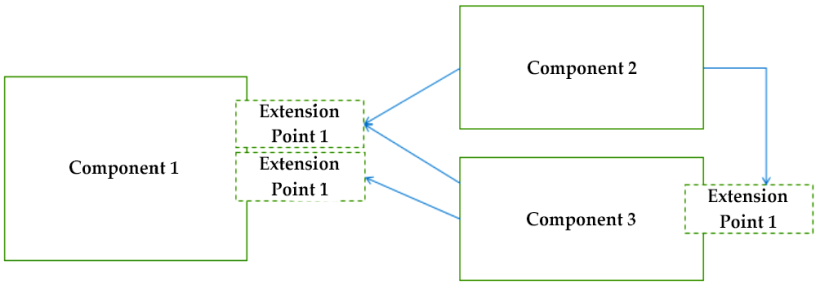
- *Reusability* Components are usually designed to be reused in different situations in different applications. However, some components may be designed for a specific task.
- *Replaceable* Components may be freely substituted with other similar components.
- *Not context specific* Components are designed to operate in different environments and contexts.
- *Extensible* A component can be extended from existing components to provide new behavior.
- *Encapsulated* A component depicts the interfaces that allow the caller to use its functionality, and do not expose details of the internal processes or any internal variables or state.
- *Independent* Components are designed to have minimal dependencies on other components.

8.10.2 Principles of Component–Based Design

A component-level design can be represented using some intermediary representation (e.g. graphical, tabular, or text-based) that can be translated into source code. The design of data structures, interfaces, and algorithms should conform to well-established guidelines to help us avoid the introduction of errors

- The software system is decomposed into reusable, cohesive, and encapsulated component units. • Each component has its own interface that specifies required ports and provided ports; each component hides its detailed implementation.
- A component should be extended without the need to make internal code or design modifications to the existing parts of the component.
- Depend on abstractions component do not depend on other concrete components which increase difficulty in expendability.
- Connectors connected components, specifying and ruling the interaction among components. The interaction type is specified by the interfaces of the components
- Components interaction can take the form of method invocations, asynchronous invocations, broadcasting, message driven interactions, data stream communications, and other protocol specific interactions.

- For a server class, specialized interfaces should be created to serve major categories of clients. Only those operations that are relevant to a particular category of clients should be specified in the interface.
- A component can extend to other components and still offer its own extension points. It is the concept of plug-in based architecture. This allows a plugin to offer to another plugin API.



8.10.3 Component-Level Design Guidelines

Create a naming conventions for components that are specified as part of the architectural model and then refine or elaborate as part of the component-level model

- Attain architectural component names from the problem domain and ensure that they have meaning to all stakeholders who view the architectural model.
- Extract the business process entities that can exist independently without any associated dependency on other entities.
- Recognize and discover these independent entities as new components.
- Use infrastructure component names that reflect their implementation-specific meaning.
- Model any dependencies from left to right and inheritance from top (base class) to bottom (derived classes)
- Model any component dependencies as interfaces rather than representing them as a direct component-to-component dependency.

8.10.4 Conducting Component-Level Design

Recognize all design classes that correspond to the problem domain as defined in the analysis model and architectural model

- Recognize all design classes that correspond to the infrastructure domain
- Describe all design classes that are not acquired as reusable components, specify message details

- Identify appropriate interfaces for each component and elaborate attributes and define data types and data structures required to implement them.
- Describe processing flow within each operation in detail by means of pseudo code or UML activity diagrams
- Describe persistent data sources (databases and files) and identify the classes required to manage them
- Develop and elaborate behavioral representations for a class or component. This can be done by elaborating the UML state diagrams created for the analysis model and by examining all use cases that are relevant to the design class
- Elaborate deployment diagrams to provide additional implementation detail.
- Demonstrate the location of key packages or classes of components in a system by using class instances and designating specific hardware and operating system environment.
- The final decision can be made by using established design principles and guidelines. Experienced designers consider all (or most) of the alternative design solutions before settling on the final design model.

Advantages

- *Ease of deployment* As new compatible versions become available, it is easier to replace existing versions with no impact on the other components or the system as a whole.
- *Reduced cost* The use of third-party components allows you to spread the cost of development and maintenance.
- *Ease of development* Components implement well-known interfaces to provide defined functionality, allowing development without impacting other parts of the system.
- *Reusable* The use of reusable components means that they can be used to spread the development and maintenance cost across several applications or systems.
- *Modification of technical complexity* A component modifies the complexity through the use of a component container and its services.
- *Reliability* The overall system reliability increases since the reliability of each individual component enhances the reliability of the whole system via reuse
- *System maintenance and evolution* Easy to change and update the implementation without affecting the rest of the system.
- *Independent* Independency and flexible connectivity of components. Independent development of components by different group in parallel. Productivity for the software development and future software development.

8.11 USER INTERFACE

User interface is the first impression of a software system from the user's point of view. Therefore any software system must satisfy the requirement of user. UI mainly performs two functions:

- Accepting the user's input
- Displaying the output

User interface plays a crucial role in any software system. It is possibly the only visible aspect of a software system.

- Users will initially see the architecture of software system's external user interface without considering its internal architecture.
- A good user interface must attract the user to use the software system without mistakes. It should help the user to understand the software system easily without misleading information. A bad UI may cause market failure against the competition of software system.
- UI has its syntax and semantics. The syntax comprises component types such as textual, icon, button etc. and usability summarizes the semantics of UI. The quality of UI is characterized by its look and feel (syntax) and its usability (semantics).
- There are basically two major kinds of user interface: a) Textual b) Graphical.
- Software in different domains may require different style of its user interface for e.g. calculator need only a small area for displaying numeric numbers but a big area for commands, A web page needs forms, links, tabs etc.

8.11.1 Graphical User Interface

A graphical user interface is the most common type of user interface seen today. It is a very user friendly because it makes use of pictures, graphics, and icons - hence why it is called 'graphical'.

It is also known as a WIMP interface because it makes use of:

- *Windows* A rectangular area on the screen where the commonly used applications run
- *Icons* A picture or symbol which is used to represent a software application or hardware device
- *Menus* A list of options from which the user can choose what they require
- *Pointers* A symbol such as an arrow which moves around the screen as user move the mouse. It helps user to select objects.

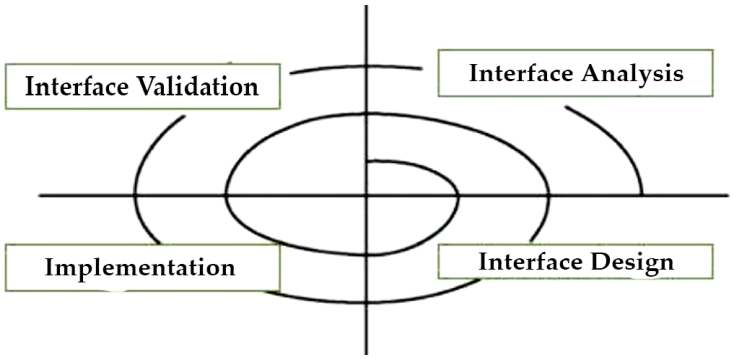
8.11.2 Design of User Interface

It starts with task analysis which understands the user's primary tasks and problem domain. It should be designed in terms of User's terminology and outset of user's job rather than programmer's.

- To perform user interface analysis, the practitioner needs to study and understand four elements:
  - The users who will interact with the system through the interface
  - The tasks that end users must perform to do their work
  - The content that is presented as part of the interface
  - The work environment in which these tasks will be conducted
- Proper or good UI design works from the user's capabilities and limitations not the machines. While designing the UI, knowledge of the nature of the user's work and environment is also critical.
- The task to be performed can then be divided which are assigned to the user or machine, based on knowledge of the capabilities and limitations of each. The design of a user interface is often divided into four different levels:
  - The conceptual level Ìt describes the basic entities considering the user's view of the system and the actions possible upon them.
  - The semantic level It describes the functions performed by the system i.e. description of the functional requirements of the system, but does not address how the user will invoke the functions.
  - The syntactic level It describes the sequences of inputs and outputs required to invoke the functions described.
  - The lexical level It determines how the inputs and outputs are actually formed from primitive hardware operations.
- User interface design is an iterative process, where all the iteration explains and refines the information developed in the preceding step. General steps for user interface design
- Define user interface objects and actions (operations)
- Define events (user actions) that will cause the state of the user interface to change
- Indicate how the user interprets the state of the system from information provided through the interface
- Describe each interface state as it will actually look to the end user

8.11.3 User Interface Development Process

It follows a spiral process as shown in the following diagram:



Interface analysis

It concentrates or focuses on users, tasks, content and work environment who will interact with the system. Defines the human - and computer-oriented tasks that are required to achieve system function.

Interface design

It defines a set of interface objects, actions and their screen representations that enable a user to perform all defined tasks in a manner that meets every usability objective defined for the system.

Interface construction

It starts with a prototype that enables usage scenarios to be evaluated and continues with development tools to complete the construction.

Interface validation

It focuses on the ability of the interface to implement every user task correctly, accommodate all task variations, to achieve all general user requirements and the degree to which the interface is easy to use and easy to learn.

User Interface Models

When a user interface is analyzed and designed these four models are used:

User profile model

- Created by a user or software engineer which establishes the profile of the endusers of the system based on age, gender, physical abilities, education, motivation, goals, and personality.
- Considers syntactic and semantic knowledge of the user and classifies users as novices, knowledgeable intermittent users and knowledgeable frequent users.

Design model

- Created by a software engineer which incorporates data, architectural, interface, and procedural representations of the software.
- Derived from the analysis model of the requirements and controlled by the information in the requirements specification which helps in defining the user of the system.

Implementation model

- Created by the software implementers who work on look and feel of the interface combined with all supporting information (books, videos, help files) that describes system syntax and semantics.
- Serves as a translation of the design model and attempts to agree with the user's mental model so that users then feel comfortable with the software and use it effectively

User's mental model

- Created by the user when interacting with the application which contains the image of the system that users carry in their heads.
- Often called the user's system perception and correctness of the description depends upon the user's profile and overall familiarity with the software in the application domain.

8.11.4 Design Considerations of User Interface

User centered

A user interface must be a user-centered product which involves users throughout a product's development lifecycle. The prototype of a user interface should be available to users and feedback from users should be incorporated into the final product.

Simple and Intuitive

UI provides simplicity and intuitiveness so that it can be used quickly and effectively without instructions. GUI are better than textual UI as GUI consists of menus, windows, and buttons and is operated by simply using mouse.

Place Users in Control

Do not force users to complete predefined sequences. Give them options—to cancel or to save and return to where they left off. Use terms throughout the interface that users can understand, rather than system or developer terms. Provide users with some indication that an action has been performed, either by showing them the results of the action, or acknowledging that the action has taken place successfully.

Transparency

UI must be transparent that helps users to feel like they are reaching right through computer and directly manipulating the objects they are working with. The interface can be made transparent by giving users work objects rather than system objects.



Users should understand that their system password must be at least 6 characters, not how many bytes of storage a password must be.

Use progressive disclosure

Always provide easy access to common features and frequently used actions. Hide less common features and actions and allow users to navigate them. Do not try to put every piece of information in one main window. Use secondary windows for information that is not key information.

Consistency

UI maintains the consistency within and across product, keep interaction results the same, UI commands and menus should

have the same format, command punctuations should be similar and parameters should be passed to all commands in the same way. UI should not have behavior's that can surprise the users and should include the mechanisms that allows users to recover from their mistakes.

Integration

The software system should integrate smoothly with other applications such as MS notepad and MS-Office. It can use Clipboard commands directly to perform data interchange.

Component Oriented

UI design must be modular and incorporate component oriented architecture so that the design of UI will have the same requirements as the design of the main body of the software system. The modules can easily be modified and replaced without affecting of other parts of the system.

Customizable

The architecture of whole software system incorporate plug-in modules which allows many different people to independently extend the software. It allows individual users to select from various available forms in order to suit personal preferences and needs.

Reduce Users' Memory Load

Do not force users to have to remember and repeat what the computer should be doing for them. For example, when filling in online forms, customer names, addresses, and telephone numbers should be remembered by the system once a user has entered them, or once a customer record has been opened.

User interfaces support long-term memory retrieval by providing users with items for them to recognize rather than having to recall information.

Separation

UI must be separated from the logic of the system through its implementation for increasing reusability and maintainability.

CASE STUDY

A FRESH VIEW OF COMPILERS

The architecture of a system can change in response to improvements in technology. This can be seen in the way we think about compilers.

In the 1970s, compilation was regarded as a sequential process, and the organization of a compiler was typically drawn as in Figure 1. Text enters at the left end and is transformed in a variety of ways—to lexical token stream, parse tree, intermediate code—before emerging as machine code on the right. We often refer to this compilation model as a pipeline, even though it was (at least originally) closer to a batch sequential architecture in which each transformation ("pass") was completed before the next one started.

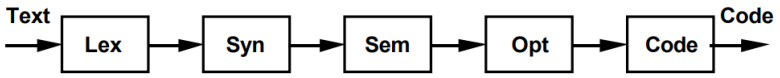


Figure 1: Traditional Compiler Model.

In fact, even the batch sequential version of this model was not completely accurate. Most compilers created a separate symbol table during lexical analysis and used or updated it during subsequent passes. It was not part of the data that flowed from one pass to another but rather existed outside all the passes. So the system structure was more properly drawn as in Figure 2.

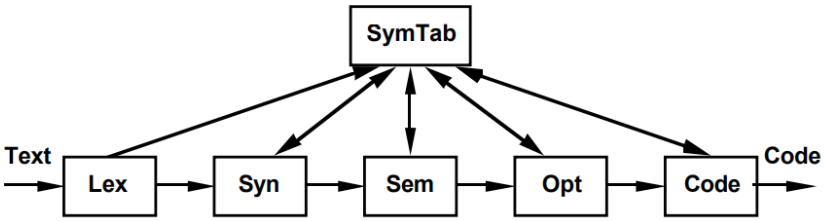


Figure 2: Traditional Compiler Model with Shared Symbol Table.

As time passed, compiler technology grew more sophisticated. The algorithms and representations of compilation grew more complex, and increasing attention turned to the intermediate representation of the program during compilation. Improved theoretical understanding, such as attribute grammers, accelerated this trend. The consequence was that by the mid-1980s the intermediate representation (for example, an attributed parse tree), was the center of attention. It was created early during

compilation and manipulated during the remainder; the data structure might change in detail, but it remained substantially one growing structure throughout. However, we continued (sometimes to the present) to model the compiler with sequential data flow as in Figure 3.

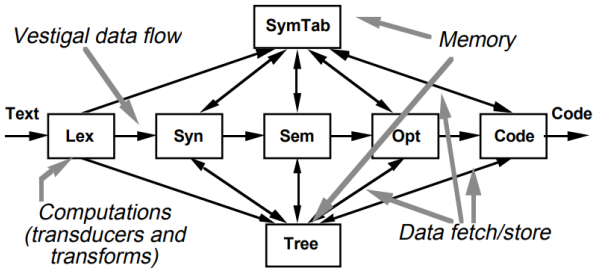


Figure 3: Modern Canonical Compiler.

In fact, a more appropriate view of this structure would re-direct attention from the sequence of passes to the central shared representation. When you declare that the tree is the locus of compilation information and the passes define operations on the tree, it becomes natural to re-draw the architecture as in Figure 4. Now the connections between passes denote control flow, which is a more accurate depiction; the rather stronger connections between the passes and the tree/symbol table structure denote data access and manipulation. In this fashion, the architecture has become a repository, and that is indeed a more appropriate way to think about a compiler of this class.

Happily, this new view also accommodates various tools that operate on the internal representation rather than the textual form of a program; these include syntax-directed editors and various analysis tools.

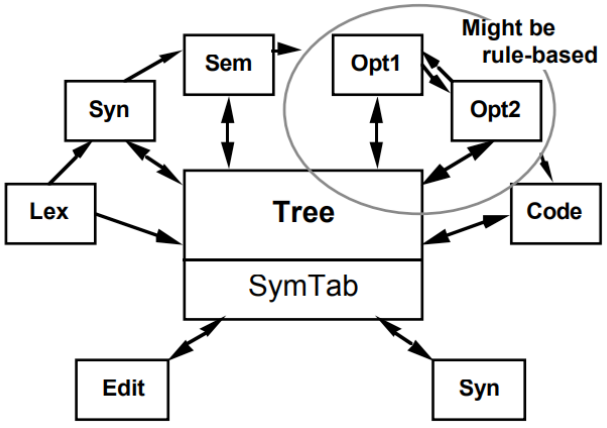


Figure 4: Canonical Compiler, Revisited.

Note that this repository resembles a blackboard in some respects and differs in others. Like a blackboard, the information of the computation is located centrally and operated on by independent computations which interact only through the shared data. However, whereas the execution order of the operations in a blackboard is determined by the types of the incoming database modifications, the execution order of the compiler is predetermined.

SUMMARY

- Software architecture and design is a process that includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.
- Software design provides a design plan that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system.
- The architectural style, also called as architectural pattern, is a set of principles which shapes an application.
- The decomposition can be modeled using a design structure matrix (DSM), which shows the dependencies between design elements without specifying the granularity of the elements.
- Inheritance creates dependency between children and parent classes and hence it blocks the free use of the child classes.
- Code related to security, communications, or system services like logging, profiling, and configuration etc.
- UML is one of object-oriented solutions used in software modeling and design.

KNOWLEDGE CHECK

1. **Architecture description language represent architectural structures that can be divided into which of the following?**
   a. Static
   b. Dynamic
   c. All of the mentioned
   d. None of the mentioned
2. **Which of the following is the essential feature of ADL?**
   a. The ability to provide structures
   b. The ability to declare a component and name its type
   c. The ability to support analytical capability
   d. All of the mentioned
3. **Which of the following must be estimated for performance analysis?**
   a. Reliability of each component
   b. Resource usage of the component
   c. All of the mentioned
   d. None of the mentioned
4. **Which of the following must be estimated for reliability analysis?**
   a. Reliability of each component
   b. Resource usage of the component
   c. All of the mentioned
   d. None of the mentioned
5. **Which of the following is true?**
   a. Once resource information is provided, portions of the system can be generate automatically
   b. Once resource information is provided, portions of the system cannot be generate automatically
   c. All of the mentioned
   d. None of the mentioned

REVIEW QUESTIONS

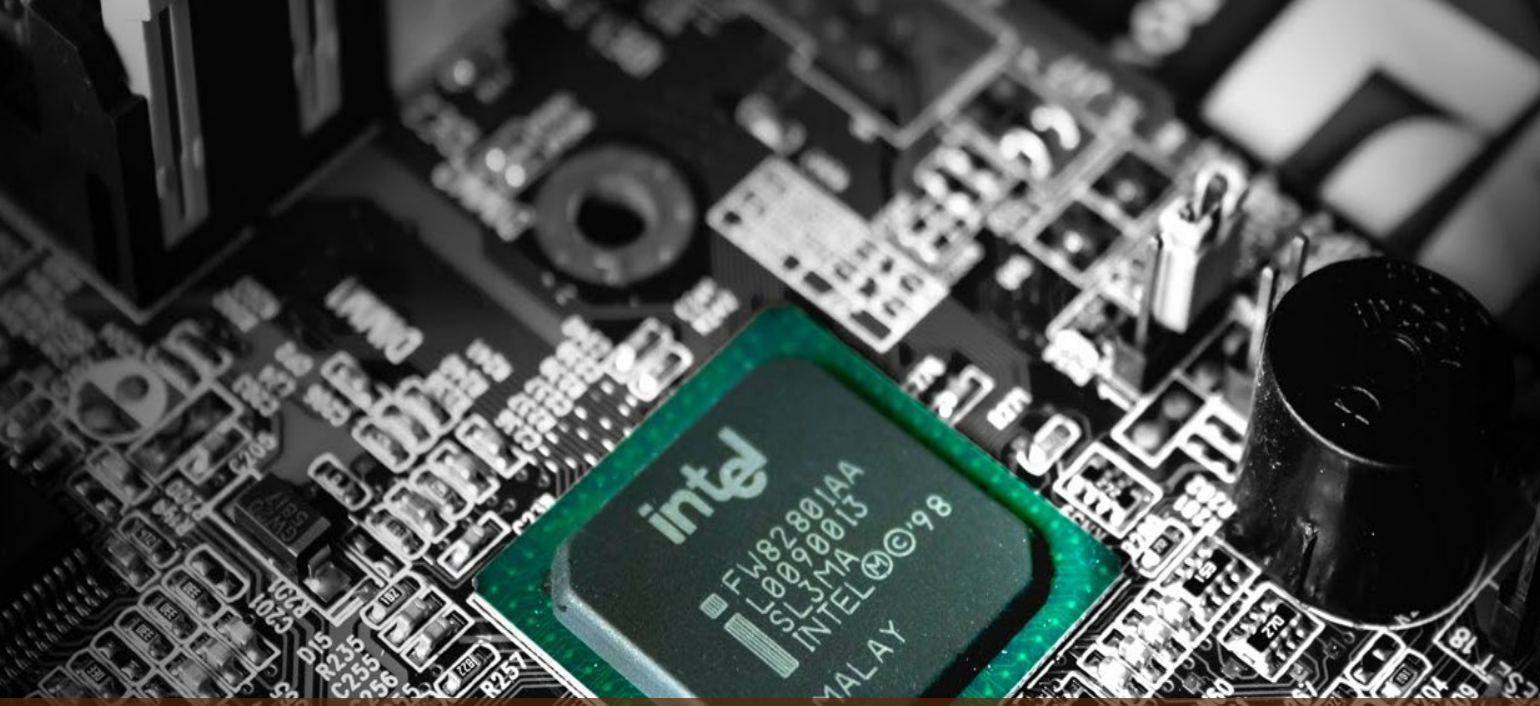
1. What are the key principles of software architecture?
2. Define the object-oriented paradigm.
3. What do you understand by distributed architecture?
4. Define interaction-oriented architecture.
5. Explain component-based architecture.

Check Your Result

1. (c) 2. (b) 3. (b) 4. (a) 5. (a)

REFERENCES

1. Bell, Michael (2008). Bell, Michael (ed.). Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley. ISBN 9780470255704.
2. Clements, Paul (2010). Documenting Software Architectures: Views and Beyond (2nd ed.). Addison-Wesley Professional. ISBN 9780321552686.
3. Fowler, Martin (September 2003). "Who Needs an Architect?" (PDF). IEEE Software. 20 (5).
4. Garzás, Javier; Piattini, Mario (2005). "An ontology for micro-architectural design knowledge". IEEE Software. 22 (2): 28–33. doi:10.1109/MS.2005.26. S2CID 17639072.
5. Kazman, Rick (May 2003). "Architecture, Design, Implementation" (PDF). Software Engineering Institute. - On the distinction between architectural design and detailed design.
6. Len, Bass (2012). Software Architecture in Practice (3rd ed.). Addison-Wesley Professional. ISBN 9780321815736. - This book covers the fundamental concepts of the discipline. The theme is centered on achieving quality attributes of a system.
7. Richards, Mark (2020). Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media. ISBN 9781492043454.
8. Shan, Tony; Hua, Winnie (October 2006). "Solution Architecting Mechanism". Proceedings of the 10th IEEE International EDOC Enterprise Computing Conference: 23–32. doi:10.1109/EDOC.2006.54. ISBN 978-0-7695-2558-7. S2CID 8361936.



CHAPTER 9

MICROPROCESSOR ARCHITECTURE

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Give an overview of microprocessor
2. Discuss about 8085 architecture
3. Understand by memory interfacing
4. Describe interruption in 8085

"Microprocessors are getting into everything. We won't be able to pick up a single piece of equipment in the near future, except maybe a broom, that hasn't got a microprocessor in it. "

— Arthur C. Clarke

INTRODUCTION

A microprocessor, sometimes called a logic chip, is a computer processor on a microchip. The microprocessor contains all, or most of, the central processing unit (CPU) functions and is the "engine" that goes into motion when you turn your computer on. A microprocessor is designed to perform arithmetic and logic operations that make use of small number-holding areas called registers. Typical microprocessor operations include adding, subtracting, comparing two numbers, and fetching numbers from one area to another. These operations are the result of a set of instructions that are part of the microprocessor

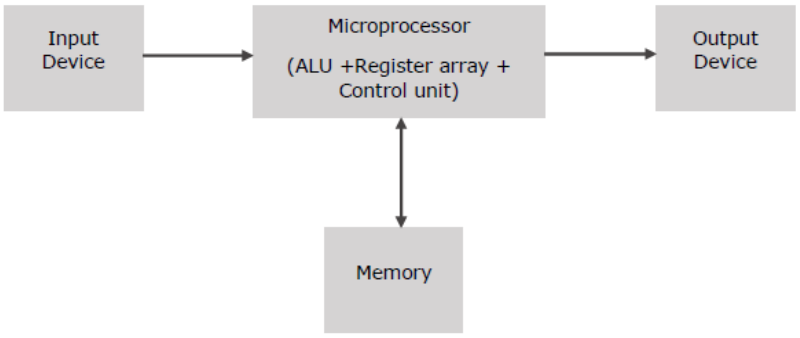
design. When your computer is turned on, the microprocessor gets the first instruction from the basic input/output system (BIOS) that comes with the computer as part of its memory. After that, either the BIOS, or the operating system that BIOS loads into computer memory, or an application program is "driving" the microprocessor, giving it instructions to perform.

9.1 OVERVIEW OF MICROPROCESSOR

Microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it.

Microprocessor consists of an ALU, register array, and a control unit. ALU performs arithmetical and logical operations on the data received from the memory or an input device. Register array consists of registers identified by letters like B, C, D, E, H, L and accumulator. The control unit controls the flow of data and instructions within the computer.

Block Diagram of a Basic Microcomputer



How does a Microprocessor Work?

The microprocessor follows a sequence: Fetch, Decode, and then Execute.

Initially, the instructions are stored in the memory in a sequential order. The microprocessor fetches those instructions from the memory, then decodes it and executes those instructions till STOP instruction is reached. Later, it sends the result in binary to the output port. Between these processes, the register stores the temporarily data and ALU performs the computing functions.

List of Terms Used in a Microprocessor

Here is a list of some of the frequently used terms in a microprocessor −

- **Instruction Set** − It is the set of instructions that the microprocessor can understand.
- **Bandwidth** − It is the number of bits processed in a single instruction.
- **Clock Speed** − It determines the number of operations per second the processor can perform. It is expressed in megahertz (MHz) or gigahertz (GHz).It is also known as Clock Rate.
- **Word Length** − It depends upon the width of internal data bus, registers, ALU, etc. An 8-bit microprocessor can process 8-bit data at a time. The word length ranges from 4 bits to 64 bits depending upon the type of the microcomputer.
- **Data Types** − The microprocessor has multiple data type formats like binary, BCD, ASCII, signed and unsigned numbers.

Features of a Microprocessor

Here is a list of some of the most prominent features of any microprocessor −

- **Cost-effective** − The microprocessor chips are available at low prices and results its low cost.
- **Size** − The microprocessor is of small size chip, hence is portable.
- **Low Power Consumption** − Microprocessors are manufactured by using metaloxide semiconductor technology, which has low power consumption.
- **Versatility** − The microprocessors are versatile as we can use the same chip in a number of applications by configuring the software program.
- **Reliability** − The failure rate of an IC in microprocessors is very low, hence it is reliable.

Classification of Microprocessors

Based on Word Length

Hope you read about word length above. So based on the word length of a processor we can have 8-bit, 16-bit, 32-bit and 64-bit processors.

RISC – Reduced Instruction Set Computer

RISC is a type of microprocessor architecture which uses small, general purpose and highly optimized instruction set rather than more specialized set of instructions found in others. RISC offers high performance over its opposing architecture CISC (see

below). In a processor, execution of each instruction require a special circuit to load and process the data. So by reducing instructions, the processor will be using simple circuits and faster in operation.

- Simple instruction set
- Larger program
- Consists of large number of registers
- Simple processor circuitry (small number of transistors)
- More RAM usage
- Fixed length instructions
- Simple addressing modes
- Usually fixed number of clock cycles for executing one instruction

CISC – Complex Instruction Set Computer

CISC is the opposing microprocessor architecture for RISC. It is made to reduce the number of instructions per program, ignoring the number of cycles per instruction. So complex instructions are directly made into hardware making the processor complex and slower in operation.

This architecture is actually designed to reduce the cost of memory by reducing the program length.

- Complex instruction set
- Smaller program
- Less number of registers
- Complex processor circuitry (more number of transistors)
- Little RAM usage
- Variable length instructions
- Variety of addressing modes
- Variable number of clock cycles for each instructions

Special Purpose Processors

There are some processors which are designed to handle some specific functions.

- DSP – Digital Signal Processors
- Coprocessors – processors used along with a main processor (8087 math-coprocessor used with 8086)
- Input/Output processors
- Transputer – Transistor Computer : Microprocessor with its own local memory

Examples

- Intel 4004 – The First Microprocessor
- Intel 8085
- Intel 8086
- Intel Pentium 4
- Intel Core i7
- AMD Athlon

9.2 8085 ARCHITECTURE

Microprocessor is a program-controlled device, which fetches the instructions from memory, decodes and executes the instructions. Most microprocessor are single- chip devices.

The functional block diagram or architecture of 8085 microprocessor, gives the complete details about the internal microprocessor. It includes the arithmetic and logic unit (ALU), timing and control unit, instruction registers and decoder, register array, interrupt control, and serial I/O control etc.

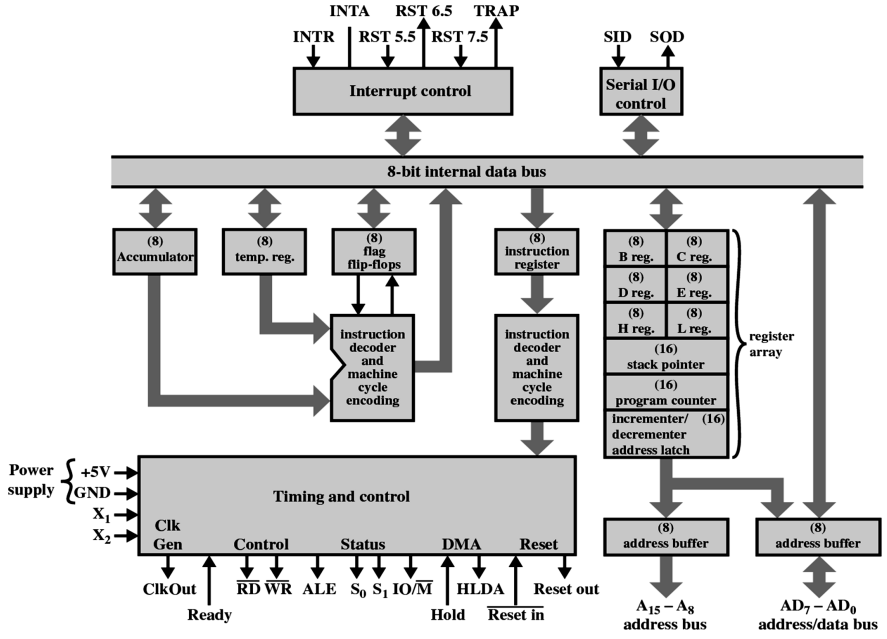


Figure 9.1: Block diagram of 8085 microprocessor.

Bus Structure of Microcomputer

The major parts of microcomputers are central processing unit (CPU), memory, and input and output unit. To connect these parts together through three sets of parallel lines, called buses. These three buses are address bus, data bus, and control bus.

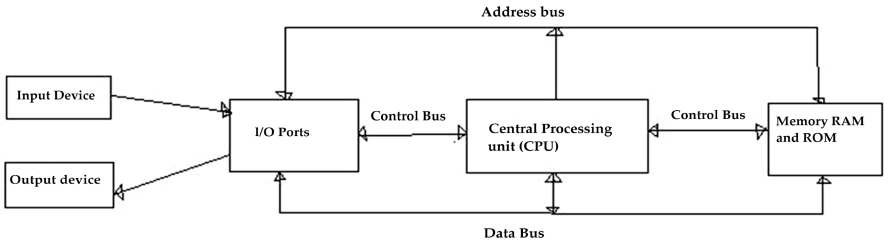


Figure 9.2: Bus structure of microcomputer.

Keyword

Address busis a computer bus architecture used to transfer data between devices that are identified by the hardware address of the physical memory (the physical address), which is stored in the form of binary numbers to enable the data bus to access memory storage.

Address Bus

The **address bus** consists of 16, 20, 24, or more parallel signal lines, through which the CPU sends out the address of the memory location. This memory location is used for to written to or read from. The number of memory location is depends on 2 to the power N address lines.

example

A CPU with 16 address lines can address 2^{16} or 65,536 memory locations. When the CPU reads data from or writes data to a port. The port address is also sent out on the address bus. This is unidirectional. This means that the CPU can send data to a memory location or I/O ports.

Data Bus

The data bus consists of 8, 16, 32 or more parallel signal lines. The data bus lines are bidirectional. This means that the CPU can read data from memory or from a I/O port as well as send data to a memory location or to a I/O port. In a system, many output devices are connected to the data bus, but only one device at a time will be enabled to the output.

Control Bus

The control bus consists of 4-10 parallel signal lines. The CPU sends out signals on the control bus to enable the outputs of addressed memory devices or port devices. Typically control bus signals are memory read, memory write, I/O read and I/O write. To read a data from a memory location, the CPU sends out the address of the desired data on the address bus and then sends out a memory read signal on the control bus. The memory read signal enables the addressed memory device to output the data onto the data bus where it is read by the CPU.

Accumulator

It is a 8-bit register which is used to perform arithmetical and logical operation. It stores the output of any operation. It also works as registers for i/o accesses.

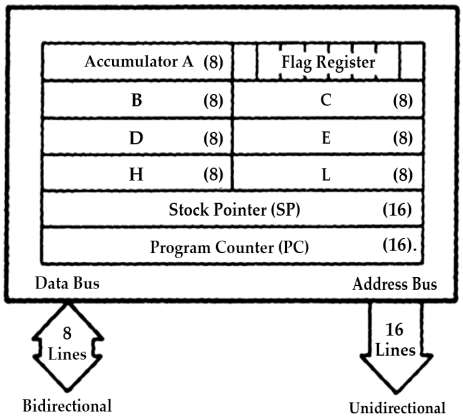
Temporary Register

It is a 8-bit register which is used to hold the data on which the acumulator is computing operation. It is also called as operand register because it provides operands to ALU.

Registers

These are general purposes registers. **Microprocessor** consists 6 general purpose registers of 8-bit each named as B, C, D, E, H and L. Generally theses registers are not used for storing the data permanently. It carries the 8-bits data. These are used only during the execution of the instructions.

These registers can also be used to carry the 16 bits data by making the pair of 2 registers. The valid register pairs available are BC, DE, and HL. We cannot use other pairs except BC, DE and HL. These registers are programmed by user.



ALU

ALU performs the arithmetic operations and logical operation.

Flag Registers

Keyword

Microprocessor is an electronic component that is used by a computer to do its work.

It consists of 5 flip flop which changes its status according to the result stored in an accumulator. It is also known as status registers. It is connected to the ALU.

There are five flip-flops in the flag register are as follows:

- Sign(S)
- zero(z)
- Auxiliary carry(AC)
- Parity(P)
- Carry(C)

The bit position of the flip flop in flag register is:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

All of the three flip flop set and reset according to the stored result in the accumulator.

Sign

If D7 of the result is 1 then sign flag is set otherwise reset. As we know that a number on the D7 always desides the sign of the number.

- if D7 is 1: the number is negative
- if D7 is 0: the number is positive

Zeros (Z)

If the result stored in an accumulator is zero then this flip flop is set otherwise it is reset.

Auxiliary carry (AC)

If any carry goes from D3 to D4 in the output then it is set otherwise it is reset.

Parity (P)

If the no of 1's is even in the output stored in the accumulator then it is set otherwise it is reset for the odd.

Carry(C)

If the result stored in an accumulator generates a carry in its final output then it is set otherwise it is reset.

Instruction registers (IR)

It is a 8-bit register. When an instruction is fetched from memory then it is stored in this register.

Instruction Decoder

Instruction decoder identifies the instructions. It takes the information's from instruction register and decodes the instruction to be performed.

Program Counter (PC)

Program Counter (PC) is 16-bit register sequencing the execution of instructions in memory that why it is also called as memory pointer. The function of the program counter is to point out the memory address of the next instruction which is to be executed. When an operation code is being fetched, the program counter is incremented by one to point out the next memory location.

The program counter is set to 0000H.

Stack Pointer (SP)

The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading a 16-bit address in the stack pointer (register).

Timing and Control Unit

It provides timing and control signal to the microprocessor to perform the various operation. It has three control signal. It controls all external and internal circuits. It operates with reference to clock signal. It synchronizes all the data transfers.

There are three control signal:

- *ALE*-Arithmetic Latch Enable, It provides control signal to synchronize the components of microprocessor.
- *RD*- This is active low used for reading operation.
- *WR*-This is active low used for writing operation.

There are three status signal used in microprocessor S0, S1 and IO/M. It changes its status according the provided input to these pins.

IO/M (Active low)	S1	S2	Data Bus Status (output)
0	0	0	Halt
0	0	1	Memory Write
0	1	0	Memory Read
1	0	1	IO Write
1	1	0	IO Read
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

Serial I/O control section

The input and output of serial data can be carried out using 2 instructions in 8085

- SID-Serial Input Data
- SOD-Serial Output Data

Two more instructions are used to perform serial-parallel conversion needed for serial I/O devices.

Instruction register and decoder

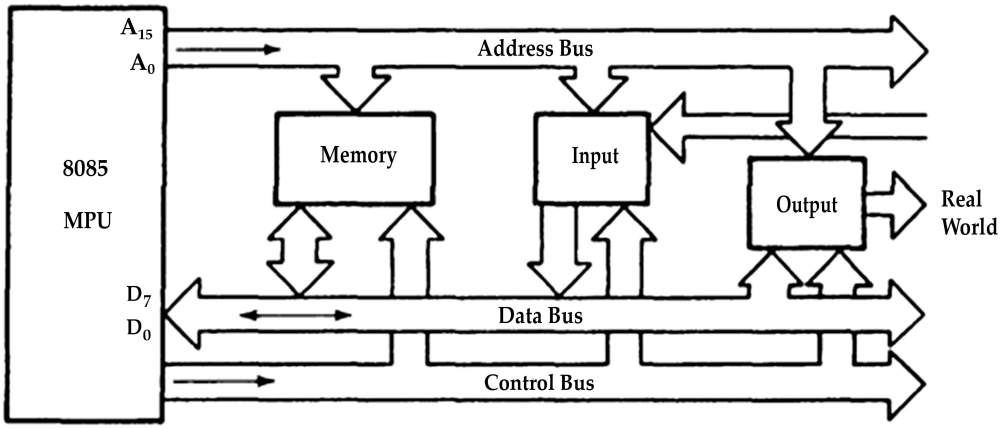
Instruction register is 8-bit register just like every other register of microprocessor. The instruction may be anything like adding two data's, moving a data, copying a data etc. When such an instruction is fetched from memory, it is directed to Instruction register. So the instruction registers are specifically to store the instructions that are fetched from memory. There is an Instruction decoder which decodes the information present in the Instruction register for further processing.

9.2.1 Bus Structure of 8085 Microprocessor

The microprocessor MPU performs various operations with peripheral devices or a memory location by using three sets of communication lines called buses: the address bus, the data bus and the control bus. And these three combined lines is called as system bus.

There are three buses in microprocessor:

- Address Bus
- Data Bus
- Control Bus



Address Bus in Microprocessor

Microprocessor has 16 bit address bus. The bus over which the CPU sends out the address of the memory location is known as Address bus. The address bus carries the address of memory location to be written or to be read from.

The address bus is unidirectional. It means bits flowing occurs only in one direction, only from microprocessor to peripheral devices.

We can find that how much memory location it can using the formula 2^N. where N is the number of bits used for address lines.

here, 2^16 = 65536bytes or 64Kb

So we can say that it can access up to 64 kb memory location.

Data Bus in Microprocessor

8085 Microprocessor has 8 bit data bus. So it can be used to carry the 8 bit data starting from 00000000H (00H) to 11111111H (FFH). Here 'H' tells the Hexadecimal Number. It is bidirectional. These lines are used for data flowing in both direction means data can be transferred or can be received through these lines. The data bus also connects the I/O ports and CPU. The largest number that can appear on the data bus is 11111111. It has 8 parallel lines of data bus. So it can access upto 2^8 = 256 data bus lines.

Control Bus in Microprocessor

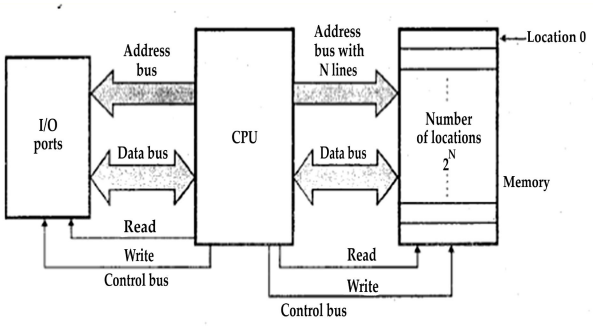
The control bus is used for sending control signals to the memory and I/O devices. The CPU sends control signal on the control bus to enable the outputs of addressed memory devices or I/O port devices.

Some of the control bus signals are as follows:

- Memory read
- Memory write
- I/O read
- I/O write.

Difference between Address and Data Bus

According to computer architecture, a bus is defined as a system that transfers data between hardware components of a computer or between two separate computers. Initially, buses were made up using electrical wires, but now the term bus is used more broadly to identify any physical subsystem that provides equal functionality as the earlier electrical buses. Computer buses can be parallel or serial and can be connected as multi drop, daisy chain or by switched hubs. System bus is a single bus that helps all major components of a computer to communicate with each other. It is made up of an address bus, data bus and a control bus. The data bus carries the data to be stored, while address bus carries the location to where it should be stored.



Address Bus

Address bus is a part of the computer system bus that is dedicated for specifying a physical address. When the computer processor needs to read or write from or to the memory, it uses the address bus to specify the physical address of the individual memory block it needs to access (the actual data is sent along the data bus). More correctly, when the **processor** wants to write some data to the memory, it will assert the write signal, set the write address on the address bus and put the data on to the data bus. Similarly, when the processor wants to read some data residing in the memory, it will assert the read signal and set the read address on the address bus. After receiving this signal, the memory controller will get the data from the specific memory block (after checking the address bus to get the read address) and then it will place the data of the memory block on to the data bus.

The size of the memory that can be addressed by the system determines the width of the data bus and vice versa. *For example*, if the width of the address bus is 32 bits, the system can address 232 memory blocks (that is equal to 4GB memory space, given that one block holds 1 byte of data).

DID YOU KNOW ?

In 1964, IBM introduced its System/360 computer architecture that was used in a series of computers capable of running the same programs with different speed and performance.

Data Bus

A data bus simply carries data. Internal buses carry information within the processor, while external buses carry data between the processor and the memory. Typically, the same data bus is used for both read/write operations. When it is a write operation, the processor will put the data (to be written) on to the data bus. When it is the read operation, the memory controller will get the data from the specific memory block and put it in to the data bus.

- The data bus consists of 8, 16 or 32 parallel lines.
- The data bus is a bi-directional bus. That means the data can get transferred from CPU to memory and vice versa.
- The data bus also connects the I/O ports and CPU. So the CPU can write data to or read it from the memory or I/O ports.

Keyword

Processor is the logic circuitry that responds to and processes the basic instructions that drive a computer.

- The number of data lines used in the data bus is equal to the size of data word being Written or read.
- Many devices in the system will connect their outputs to the same data bus. But only one device at a time will have its output enabled, so that there is no collision of information.
- In order to completely eliminate the possibility of more than one devices putting their outputs simultaneously on the data bus, all the devices getting connected to data bus should have three state outputs.
- This will make it possible to disable the outputs of devices which should be stopped from connecting their outputs to the data bus.

Microcomputer and Microprocessor

Microcomputers are much smaller computer. They operate still more slowly and they work with still smaller data words (typically 4 bits, 8 bits, 16 bits or 32 bits).

Keyword

Read-only memory is a type of non-volatile memory used in computers and other electronic devices.

They can address few thousands to few million memory location. One of the important feature of microcomputer is that the CPU is usually a single IC called as a microprocessor.

- A microprocessor has a CPU to which we have to add ROM, RAM and ports to make a microcomputer.
- Application of microcomputer are from sewing machine, washing machine and other domestic appliances to computer added design system.
- Intel 8051 is a microcomputer which is actually a single chip computer.

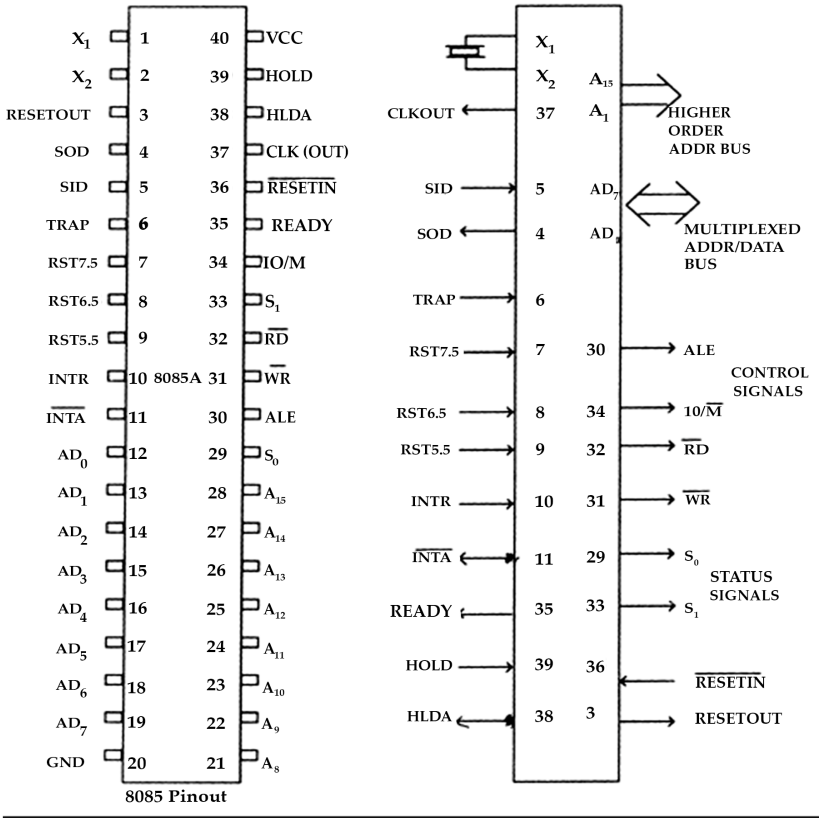
9.2.2 Pin Diagram of 8085

8085 is a general purpose microprocessor having 40 pins and works on single power supply.

To study the pin diagram, we group the signals into 5 categories:

- Power supply
- Clock signals

- Interrupt signals
- Address and data bus
- Control and status signals
- Serial I/O port
- DMA request signals



Power Supply and Clock Signals

VCC:-Vcc is to be connected to +5V power supply.

Vss:-Ground reference

X_1 and X_2:- This pin is used for providing the clock frequency to the microprocessor. Generally Crystal oscillator or LC oscillator is used to generate the frequency. The frequency generated here is internally divided into two. As we know that the basic operating timing frequency of the microprocessor is 3 MHz so

6 MHz frequency is applied.

Interrupt Signals

Pin 6 to 11:- These pins are used for interrupt signals. Generally and external devices are connected here which requests the microprocessor to perform a particular task.

There are 5 pins for hardware interrupts-

TRAP, RST7.5, RST 6.5, RST5.5 and INTR

INTA is used for acknowledgement. Microprocessor sends the acknowledgement to external devices through the INTA pin.

Address and Data bus

AD0-AD7:-These are multiplexed address and data bus. So it can be used to carry the lower order 8 bit address as well as the data. Generally these lines are demultiplexed using the Latch.

During the opcode fetch operation, in the first clock cycle the lines deliver the lower order address bus A_0-A_7.

In the subsequent IO/M read or write it is used as data bus D_0-D_7. CPU can read or write data through these lines.

A_8-A_{15}:- These are address bus used to address the memory location.

Control and Status Signals

S_0 and S_1:-It is used for the status signal in microprocessor.

*ALE (Arithmetic Latch Enable):-*This signal is used to capture the lower address presented on multiplexed address and data bus.

RD:-This is active low input generally used for reading operation.

WR:-This is active low input used for writing operation.

IO/M:-This pin is used to select the memory or input-output through which we want to communicate the data.

Ready:-As we know that memory and input -output have slower response than microprocessor. So a microprocessor may now be able to handle further data till it completes the present job. So it is in waiting state. As it completes the present job it sets the ready pin. Microprocessor enters into wait state while ready pin is disabled.

Reset In:-This is active low input. This pin is used to reset the microprocessor. An active low signal applied to this pin reset the program counter inside the microprocessor.

Reset out:-If we want to reset the external devices connected to the microprocessor then a signal applied to this pin resets the external devices.

Serial I/O Port

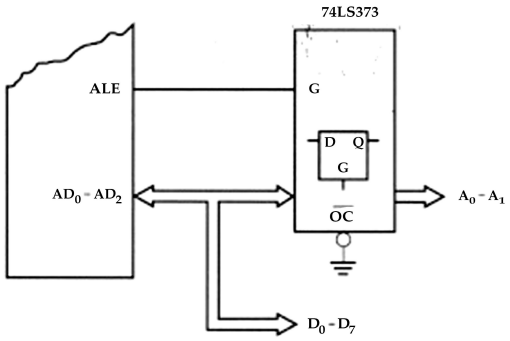
SID and SOD:-These pins are used for serial data communication.

DMA Request Signals

HOLD and HLDA:-HOLD is an active high input signal used by the other controller to request microprocessor about use of address, data and control signal. The HOLD and HLDA signal are used for direct memory access (DMA). DMA controller receives a requests from a device and in turn issues the HOLD signal to the microprocessor.

Multiplexing Address and Data bus

Generally, 74LS373 Latch IC is used for latching the address issued by Microprocessor. It has 8-latches (D-flip flop).



Address will appear on AD_0-AD_7 lines.

ALE will go high and forcing enable G pin of Latch. This will make the latch transparent. It means whatever will be input, will be output. Presently input address is A_0-A_7.Therefore output is A_0-A_7.

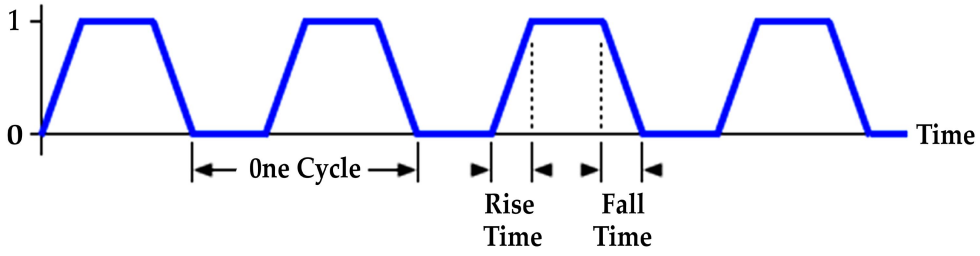
When ALE=0, then AD_0-AD_7 will now be used as data bus.

Representation of Signals in Timing Diagram

There are some representation of signals in timing diagram of 8085, such as:

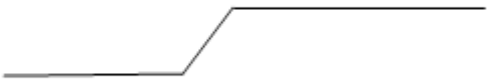
Clock Signals

As we know that the microprocessor operates with reference to clock signals provided to it. At pins X_1 and X_2 we provide clock signals and this frequency is divided by two. This frequency is called as the operating frequency.



Single Line Signal

The status of single line will be either LOW or HIGH. But the change from one state to another state is not possible in zero time.



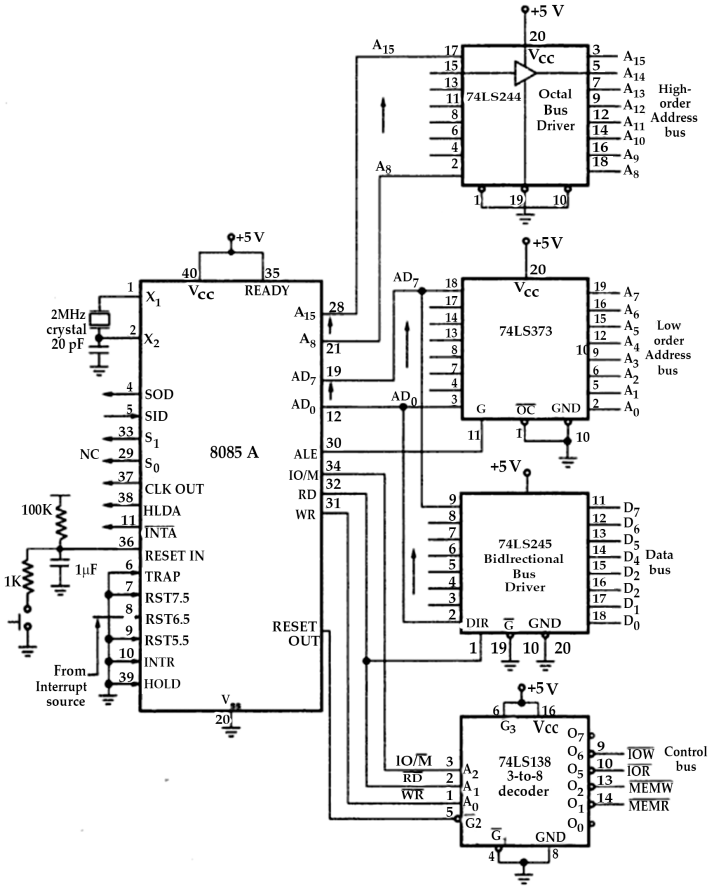
Multiple Line Signals

In Microprocessor we have multiple lines. These signals are also called as bus signals. If a single line changes occur then we have to crossing to indicate change in contents.

9.3 MEMORY INTERFACING

Memory is an integral part of a microprocessor system. The memory interfacing circuit is used to access memory quite frequently to read instruction codes and data stored in **memory**. This read/write operations are monitored by control signals. The microprocessor activates these signals when it wants to read from and write into memory. The memory read and memory write machine cycles, and status of the $\overline{RD}$, $\overline{WR}$ and $IO/\overline{M}$ status signals for read/write operation.

- IO / $\overline{M}$ *(Output):* It is a status signal which distinguishes whether the address is for memory or I/O. when it goes high the address on the address bus is for an I/O device. When it goes low the address on the address bus is for a memory location.
- $\overline{RD}$ *(Output):* It is a signal to control READ operation .when it goes low the selected memory or I/O device is read.
- $\overline{WR}$ *(Output):* It is a signal to control WRITE operation .when it goes low the data on the data bus is written into the selected memory or I/O operation.



Interfacing a microprocessor is to connect it with various peripherals to perform various operations to obtain a desired output. Memory Interfacing and I/O Interfacing are the two main types of interfacing. Memory Interfacing is used when the microprocessor needs to access memory frequently for reading and writing data stored in the memory. It is used when reading/writing to a specific register of a memory chip. I/O Interfacing is achieved by connecting keyboard (input) and display monitors (output) with the microprocessor.

Memory interfacing, as we know that any system which process digital data needs the facility for storing the data. Interfacing is a technique to be used for connecting the microprocessor to memory.

8085 has 16 bit address bus; hence it can access 2^{16} no. of memory locations, which is equal to 64KB memory. For any microprocessor memory is required to store program as well as data. Since microprocessor does not have on-chip memory, we need to connect it externally. So it requires addressing mechanism. The following are the steps involved in interfacing memory with 8085 processor.

1. First decide the size of memory requires to be interfaced. Depending on this we can say how many address lines are required for it. *For example* if you want to interface 4KB (2^{12}) memory it requires 12 address lines. Remaining address lines can be used in address decoding.
2. 1Depending on the size of memory required and given address range, construct address decoding circuitry. This address decoding circuitry can be implemented with NAND gates and/or decoders or using PAL (when board size is a constraint).
3. Connect data bus of memory to processor data bus.
4. Generate the control signals required for memory using IO/M', WR', RD' signals of 8085 processor.

Keyword

Memory is the process in which information is encoded, stored, and retrieved.

9.3.1 IO interfacing to 8085

There are two techniques through which devices can be interfaced to microprocessor.

1. Memory mapped I/O
2. Peripheral mapped I/O or I/O mapped I/O

Memory mapped I/O

I/O devices are interfaced using address from memory space. That means IO device address are part of addresses given to memory locations.8085 uses 16-bit address to memory interfacing. So any address between 0000H-FFFFH can be given to each peripheral. But the addresses given to peripheral cannot be used for memory.

Memory control signals are used as read and write control signals for peripherals. And all the operations that can be performed on memory can also be performed on peripherals. No need of using IO instructions such as IN, OUT.

IO mapped IO

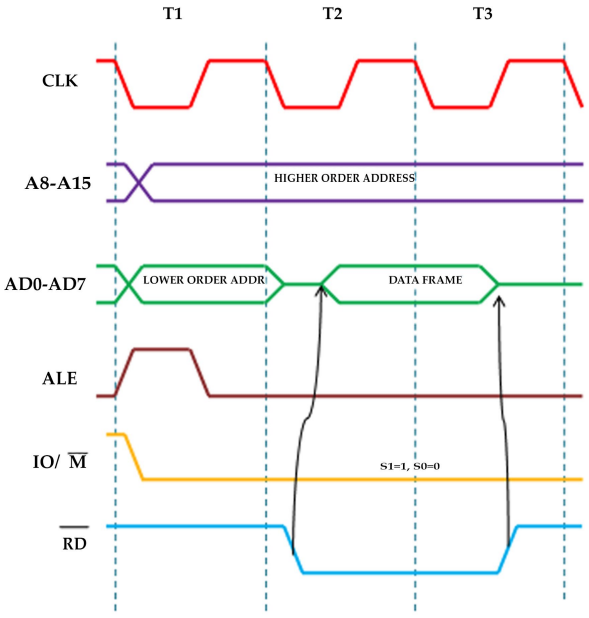
In this method separate address space is given to IO devices. Each IO device is given a 8-bit address. Hence maximum 256 devices can be interfaced to the processor. The address range for the IO devices is 00H-FFH. IO control signals are used to perform read, write operations.

For reading data from IO device or writing data to IO device IN, OUT instructions needs to be used. Arithmetic and logical operations cannot be performed directly on IO devices as in memory mapped IO.

IO devices can be interfaced, by using buffers for simple IO i.e. by using address decoding circuit to enable buffer. For handshake IO or to interface more peripherals ICs like 8255 peripheral programmable interface (PPI) can be used.

Memory Read machine cycle

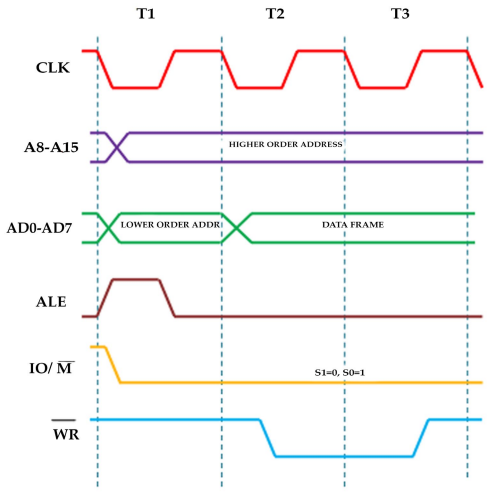
This machine cycle is required when an operand is present in memory. This machine cycle requires three T-states. The following are the sequence of actions performed by microprocessor during this machine cycle.



- In the first T-state (T1) 8085 places address on address bus and issues ALE signal. And also IO/M′ signal is made low, since it is memory related operation.
- In the second T-state (T2), processor issues RD′ control signal to memory. In response to this memory places data on data bus.
- In the third T-state (T3), processor reads data from data bus, and de-asserts RD′ signal.

Memory Write Machine cycle

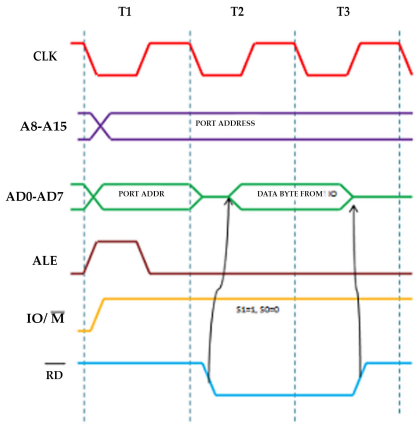
This machine cycle is required when the results of operation needs to store in memory. This machine cycle requires three T-states. The following are sequence of actions performed by processor in this machine cycle.



- In first T-state (T1), 8085 processor places 16- bit address on address bus and issues ALE signal. And also it makes IO/M′ signal to low, indicating it is memory related operation.
- In second T-state (T2), processor places data to be written on data bus and asserts WR′ signal to the memory.
- In the third T-state (T3), memory stores the data and processor de-asserts WR′ signal.

IO read machine cycle

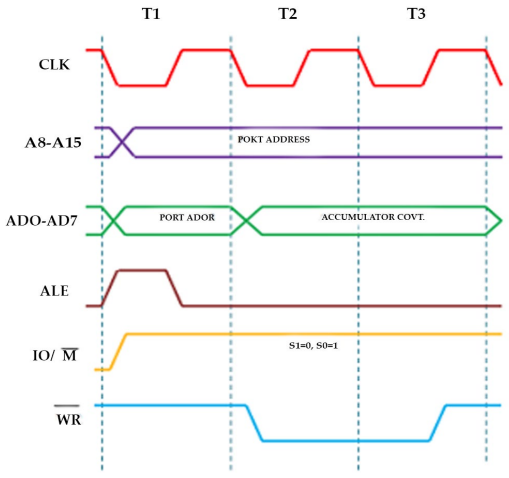
This machine cycle is required, when data needs to be read from an input device. This machine cycle requires three T-states. The following are the sequence of actions performed by processor during this machine cycle.



- In the first T-state (T1) 8085 places port address(for IO mapped addresses port address is 8-bit, but for memory mapped addresses IO device address is 16-bit, but reading from such is performed by memory read machine cycle) on address bus and issues ALE signal. And also IO/M' signal is made high, since it is IO related operation.
- In the second T-state (T2), processor issues RD' control signal to IO peripheral. In response to this input device places data on data bus.
- In the third T-state (T3), processor reads data from data bus, and de-asserts RD' signal.

IO write Machine cycle

This machine cycle is required when data needs to be output to an output device. This machine cycle requires three T-states. The following are the sequence of actions performed by processor during this machine cycle.



- In first T-state (T1), 8085 processor places 8-bit port address on address bus (for IO mapped addresses port address is 8-bit, but for memory mapped addresses, IO device address is 16-bit, but writing to such is performed by memory write machine cycle) and issues ALE signal. And also it makes IO/M' signal to high, indicating it is IO related operation.
- In second T-state (T2), processor places data to be written on data bus and asserts WR' signal to the peripheral.
- In the third T-state (T3), peripheral accepts the data and processor de-asserts WR' signal.

REMEMBER

Microcomputers became popular in the 1970s and 1980s with the advent of increasingly powerful microprocessors. The predecessors to these computers, mainframes and minicomputers, were comparatively much larger and more expensive.

9.4 INTERRUPTS IN 8085

An interrupt is a signal or condition that causes processor to stop its normal execution flow and makes it to jump to some other location for processing the interrupt.

8085 has 4 mask-able interrupts and 1 non-mask-able interrupts. Mask able interrupts can be disabled be DI instruction. Among four mask-able interrupts one is non-vectored interrupt, that is processor cannot go to a fixed location as in case of vectored interrupt, the external device which caused interrupts needs to specify the vector address.

An interrupt can be defined as any signal to the μP that alters the normal sequence of execution of a program. The interrupt can be introduced in the μP through an instruction written in the program or it can even be initiated by external device. Whenever μP receives any interrupt it is control gets shifted to some other location in order to execute a set of instructions called service routine which is written at that location. The μP resumes its operation after completing the service routine. The interrupt process in 8085 is controlled by the Interrupt Enable flip-flop, which is internal to the processor and can be set or reset by using software instructions.

9.4.1 8085 Interrupt Response Process

Interrupts should be enabled by using EI instruction, then only processor responds to all mask able interrupts.

- When microprocessor is executing a program, it checks for INTR line during execution of each instruction.
- If INTR is high then processor completes executing the current instruction, disables the interrupts and sends a INTA signal
- INTA is used by the external hardware to specify the restart instruction to processor (since INTR is a non-vectored interrupt).
- When microprocessor receives the RST instruction, it saves PC content on stack and PC is loaded with the vector address.
- Microprocessor executes the instructions at vector address.
- The interrupts should be enabled if required in the ISR(interrupt service routine)
- At the end of interrupt service routine, RET instruction loads the PC from the stack. So processor comes back to the instruction where it was interrupted previously.

Types of Interrupts

There are two types of interrupts used in 8085 microprocessor, such as:

- Software interrupts
- Hardware interrupts

Software interrupts

A software interrupts is a particular instructions that can be inserted into the desired location in the program. There are eight Software interrupts in 8085 Microprocessor. From RST_0 to RST_7.

- RST0
- RST1
- RST2
- RST3
- RST4
- RST5
- RST6
- RST7

They allow the microprocessor to transfer program control from the main program to the subroutine program. After completing the subroutine program, the program control returns back to the main program.

We can calculate the vector address of these interrupts using the formula given below:

Vector Address = Interrupt Number * 8

So we can find simply vector address. *For Example*:

RST2: vector address=2*8 = 16

RST1: vector address=1*8 = 08

RST3: vector address=3*8 = 24

The instruction set of 8085 includes eight RST (Restart) instructions referred to as software interrupts. These are one-byte instructions. Each of these instructions allows the transfer of the program execution to a specific location on page 00H. Therefore RST instructions act like a vector that points towards different memory locations. Table 1 shows the list of different RST instructions.

Table 1: Software interrupts

Instruction	Hex Code	Vector Location
RST0	C7	0000H
RST1	CF	0008H
RST2	D7	0010H
RST3	DF	0018H
RST4	E7	0020H
RST5	EF	0028H
RST6	F7	0030H
RST7	FF	0038H

Hardware Interrupts of 8085

The hardware interrupts of 8085 are initiated by an external device by placing an appropriate signal at the interrupt pin of the processor. The processor keeps on checking the interrupt pins at the second T-state of the last machine cycle of every instruction. If the processor finds a valid interrupt signal and if the interrupt is unmasked and enabled, then the processor accepts the interrupt.

There are five hardware interrupts in 8085. They are:

- INTR
- RST 5.5
- RST 6.5

- RST 7.5
- TRAP

INTA is not an interrupt. INTA is used by the Microprocessor for sending the acknowledgement. TRAP has highest priority and RST7.5 has second highest priority and so on.

The Vector address of these interrupts are given below:

Interrupt	Vector Address
RST7.5	$003C_H$
RST6.5	0034_H
RST5.5	$002C_H$
TRAP	0024_H

TRAP

It is non-mask able edge and level triggered interrupt. TRAP has the highest priority and vectors interrupt. Edge and level triggered means that the TRAP must go high and remain high until it is acknowledged. In case of sudden power failure, it executes a ISR and send the data from main memory to backup memory.

As we know that TRAP cannot be masked but it can be delayed using HOLD signal.

This interrupt transfers the microprocessor's control to location 0024H.

TRAP interrupts can only be masked by resting the microprocessor. There is no other way to mask it.

RST7.5

It has the second highest priority. It is mask-able and edge level triggered interrupt. The vector address of this interrupt is 003CH. Edge sensitive means input goes high and no need to maintain high state until it is recognized.

It can also be reset or masked by resting microprocessor. It can also be resettled by DI instruction.

RST6.5 and RST5.5

These are level triggered and mask-able interrupts. When RST6.5 pin is at logic 1, INTE flip-flop is set. RST 6.5 has third highest priority and RST 5.5 has fourth highest priority.

It can be masked by giving DI and SIM instructions or by resting microprocessor.

INTR

It is level triggered and mask-able interrupt. It has the lowest priority. It can be disabled by resting the microprocessor or by DI and SIM instruction.

The following sequence of events occurs when INTR signal goes high:

- The 8085 checks the status of INTR signal during execution of each instruction.
- If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge signal, if the interrupt is enabled.
- On receiving the instruction, the 8085 save the address of next instruction on stack and execute received instruction.

SUMMARY

- A microprocessors is simply a computer processor that has been configured into the design and function of a microchip. Sometimes referred to as a logic chip or electronic chip, this small component functions as the means of executing the command to start booting up a computer.
- Microprocessor is a program-controlled device, which fetches the instructions from memory, decodes and executes the instructions. Most Micro Processor are single- chip devices.
- Program Counter (PC) is 16-bit register sequencing the execution of instructions in memory that why it is also called as memory pointer.
- Instruction register is 8-bit register just like every other register of microprocessor. The instruction may be anything like adding two data's, moving a data, copying a data etc
- Address bus is a part of the computer system bus that is dedicated for specifying a physical address.
- Microcomputers are much smaller computer. They operate still more slowly and they work with still smaller data words (typically 4 bits, 8 bits, 16 bits or 32 bits).

KNOWLEDGE CHECK

1. **Which clock pulses are generated by the microprocessor so as to handle the timing and control operations related to internal functioning level?**
   a. single phase clock pulses
   b. multi-phase clock pulses
   c. anti-phase clock pulses
   d. none of the above
2. **The bus-request control input of micro-processor indicates the temporary suspension of current operation by driving all buses into_____.**
   a. high impedance state
   b. low impedance state
   c. both a & b
   d. none of the above
3. **Which feature conducts the memory transfer by controlling the address and data buses on the basis of request originated by the device when buses get disabled by the microprocessor?**
   a. Indirect Memory Access
   b. Direct Memory Access
   c. Read Memory Access
   d. Write Memory Access
4. **In 8085 microprocessor system with memory mapped I/O, which of the following is true?**
   a. Devices have 8-bit address line
   b. Devices are accessed using IN and OUT instructions
   c. There can be maximum of 256 input devices and 256 output devices
   d. Arithmetic and logic operations can be directly performed with the I/O data
5. **In an intel 8085A microprocessor, why is READY signal used?**
   a. To indicate to user that the microprocessor is working and is ready for use.
   b. To provide proper WAIT states when the microprocessor is communicating with a slow peripheral device.
   c. To slow down a fast peripheral device so as to communicate at the microprocessor's device.
   d. None of the above.

REVIEW QUESTIONS

1. If a processor has 4 GB memory then how many address lines are required to access this memory?
2. What do you understand by 8085 architecture?
3. A processor has 16 MB memory, how many address lines are required to access this memory.
4. Discuss about interrupts in 8085.
5. Add the two hex number 24H and FFH and also tells about the flag status.

Check Your Result

1. (b) 2. (a) 3. (b) 4. (d) 5. (b)

REFERENCES

1. "First Microprocessor". First Microprocessor | 50th Anniversary of the Microprocessor 2020. Archived from the original on January 6, 2014.
2. "Function of a Microprocessor". Hubspire. 2017-04-26.
3. "How Microprocessors Work". April 2000. Retrieved 2009-12-23.
4. "Reduced instruction set computer." Whatis?com: 9 Aug 2000, http://whatis.techtarget.com/WhatIs_Definition_Page/0,4152,214266,00.html.
5. Ayala, K. J. (2005), The 8051 Microcontroller, 3rd Ed., Thomson Learning.
6. Bright, Peter (November 15, 2011). "The 40th birthday of—maybe—the first microprocessor, the Intel 4004". arstechnica.com. Archived from the original on January 6, 2017.
7. Dirk Oppelt. "The CPU Collection". Retrieved 2009-12-23.
8. Gennadiy Shvets. "CPU-World". Retrieved 2009-12-23.
9. Jérôme Cremet. "The Gecko's CPU Library". Retrieved 2009-12-23.
10. Mazidi, M. A., Mazidi, J. G., and McKinlay, R. D. (2006), The 8051 Microcontroller and Embedded systems, 2nd Ed., Pearson/Prentice Hall.



CHAPTER 10

PARALLEL PROCESSING

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Explain multiple processor organizations
2. Define symmetric multiprocessors
3. Describe cache coherence and the MESI protocol
4. Elaborate multithreading and chip multiprocessors
5. Discuss about the clusters
6. Explain nonuniform memory access
7. Define vector computation

"A lot of the parallel processing software we're currently developing for supercomputers is tantric."

— Frederick Lenz

INTRODUCTION

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 superscalar 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 some of the most prominent approaches to parallel organization. First, we examine symmetric multiprocessors (SMPs), one of the earliest and still the most common example of parallel organization. In an SMP organization, multiple processors share a common memory.

This organization raises the issue of cache coherence, to which a separate section is devoted. Then we describe clusters, which consist of multiple independent computers organized in a cooperative fashion. Next, the chapter examines multithreaded processors and chip multiprocessors. Clusters have become increasingly common to support workloads that are beyond the capacity of a single SMP. Another approach to the use of multiple processors that we examine is that of nonuniform memory access (NUMA) machines.

The NUMA approach is relatively new and not yet proven in the marketplace, but is often considered as an alternative to the SMP or cluster approach. Finally, this chapter looks at hardware organizational approaches to vector computation. These approaches optimize the ALU for processing vectors or arrays of floating-point numbers. They are common on the class of systems known as supercomputers.

10.1 MULTIPLE PROCESSOR ORGANIZATIONS

A taxonomy first introduced by Flynn is still the most common way of categorizing systems with parallel processing capability. Flynn proposed the following categories of computer systems:

- ***Single instruction, single data (SISD) stream***: A single processor executes a single instruction stream to operate on data stored in a single memory. Uniprocessors fall into this category.
- ***Single instruction, multiple data (SIMD) stream:*** A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each 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 this category.

- ***Multiple instruction, single data (MISD) stream***: A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. This structure is not commercially implemented.
- ***Multiple instruction, multiple data (MIMD) stream***: A set of processors simultaneously execute different instruction sequences on different data sets. SMPs, clusters, and NUMA systems fit into this category.

With the MIMD organization, the processors are general purpose; each is able to process all of the instructions necessary to perform the appropriate data transformation. MIMDs can be further subdivided by the means in which the processors communicate (Figure 10.1). If the processors share a common memory, then each processor accesses programs and data stored in the shared memory, and processors communicate with each other via that memory.

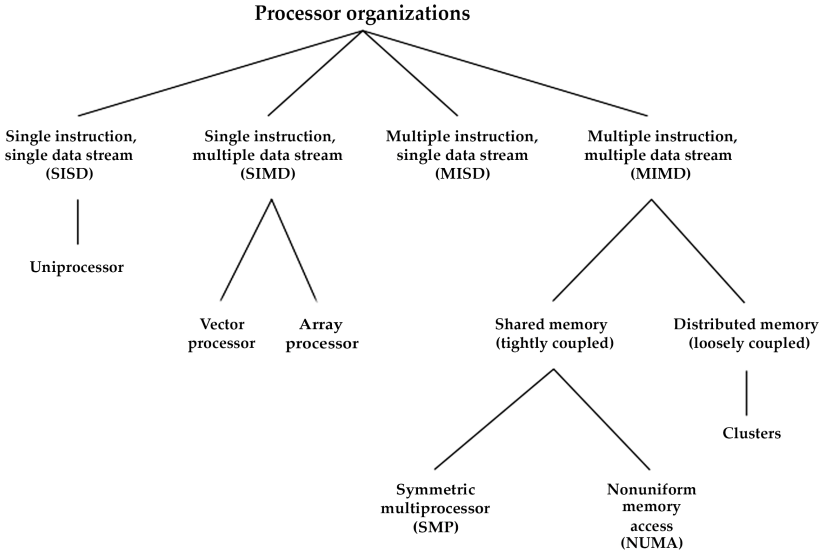


Figure 10.1: A Taxonomy of Parallel Processor Architectures.

The most common form of such system is known as a **symmetric multiprocessor** (SMP). In an SMP, multiple processors share a single memory or pool of memory by means of a shared bus or other interconnection mechanism; a distinguishing feature is that the memory access time to any region of memory is approximately the same for each processor. A more recent development is the non-uniform memory access (NUMA) organization. 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 be interconnected to form a cluster. Communication among the computers is either via fixed paths or via some network facility

10.1.1 Parallel Organizations

Keyword

Symmetric multiprocessing is the processing of programs by multiple processors that share a common operating system and memory.

Figure 10.2 illustrates the general organization of the taxonomy of Figure 10.1. Figure 10.2a 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 SIMD, there is still a single control unit, now feeding a single instruction stream to multiple PUs. Each PU may have its own dedicated memory (illustrated in Figure 10.2b), or there may be a shared memory. Finally, with the MIMD, there are multiple control units, each feeding a separate instruction stream to its own PU. The MIMD may be a shared-memory multiprocessor (Figure 10.2c) or a distributed-memory multicomputer (Figure 10.2d).

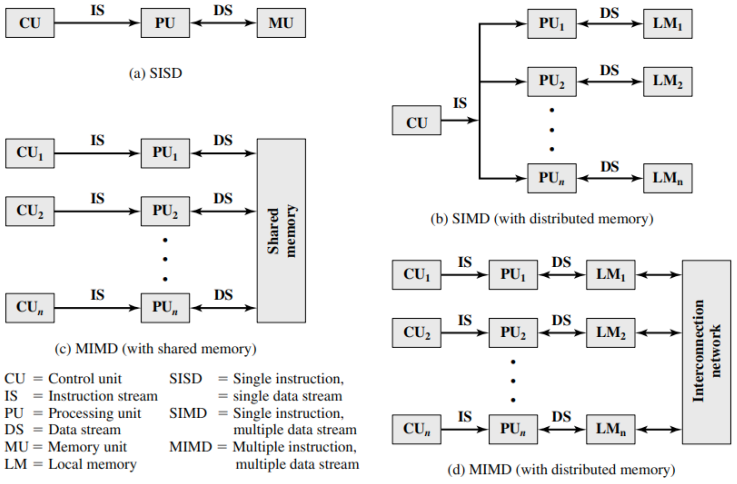


Figure 10.2: Alternative Computer Organizations.

The design issues relating to SMPs, clusters, and NUMAs are complex, involving issues relating to physical organization, interconnection structures, interprocessor communication, operating system design, and application software techniques. Our concern here is primarily with organization, although we touch briefly on operating system design issues.

10.2 SYMMETRIC MULTIPROCESSORS

Until fairly recently, virtually 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 systems with an SMP organization. The term SMP refers to a computer hardware architecture and also to the operating system behavior that reflects that architecture. An SMP can be defined as a standalone computer system with the following characteristics:

Keyword

In April 1958, Stanley Gill (Ferranti) discussed parallel programming and the need for branching and waiting.

1. There are two or more similar processors of comparable capability.
2. These processors share the same main memory and I/O facilities and are interconnected by a bus or other internal connection scheme, such that memory access 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 symmetric).
5. The system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels.

Points 1 to 4 should be self-explanatory. Point 5 illustrates one of the contrasts with a loosely coupled multiprocessing system, such as a cluster. In the latter, the physical unit of interaction is usually a message or complete file. In an SMP, individual data elements can constitute the level of interaction, and there can be a high degree of cooperation between processes.

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, including the following:

- ***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 system with multiple processors

will yield greater performance than one with a single processor of the same type (Figure 10.3).

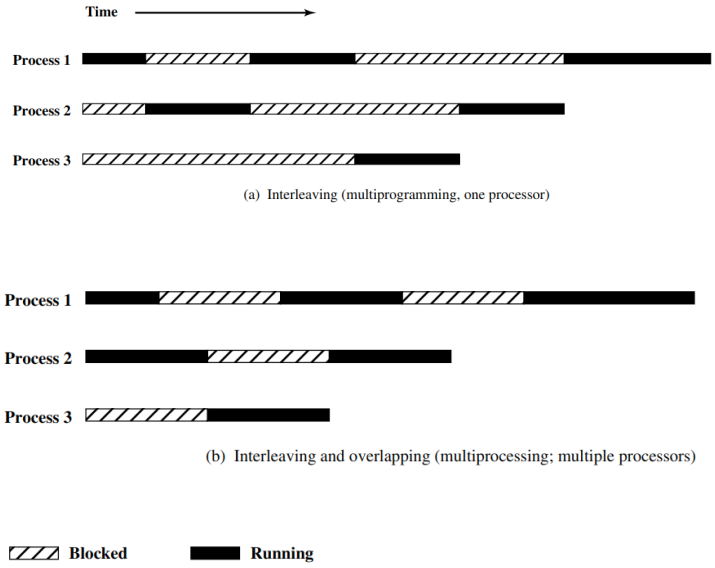


Figure 10.3: Multiprogramming and Multiprocessing.

- ***Availability:*** In a symmetric multiprocessor, because all processors can perform the same functions, the failure of a single processor does not halt the machine. Instead, the system can continue to function at reduced performance.
- ***Incremental growth:*** A user can enhance the performance of a system by adding an additional processor.
- ***Scaling:*** Vendors can offer a range of products with different price and performance characteristics based on the number of processors configured in the system.

It is important to note that these are potential, rather than guaranteed, benefits. The operating system must provide tools and functions to exploit the parallelism in an SMP system.

An attractive feature of an SMP is that the existence of multiple processors is transparent to the user. The operating system takes care of scheduling of threads or processes on individual processors and of synchronization among processors.

10.2.1 Organization

Figure 10.4 depicts in general terms the organization of a multiprocessor system. There are two or more processors. Each processor is self-contained, including a control unit,

ALU, registers, and, typically, one or more levels of cache. Each processor has access to a shared main memory and the I/O devices through some form of interconnection mechanism.

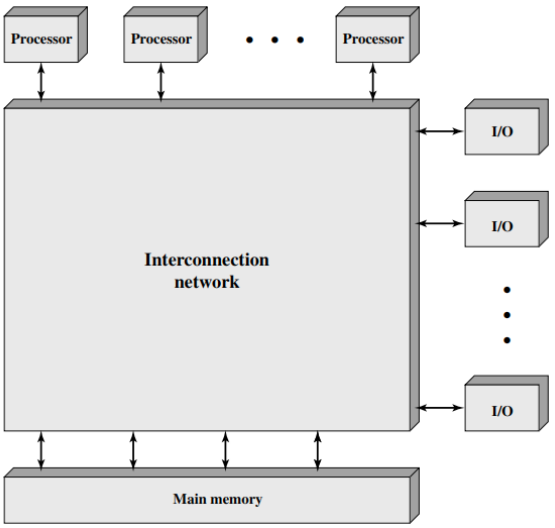


Figure 10.4: Generic Block Diagram of a Tightly Coupled Multiprocessor.

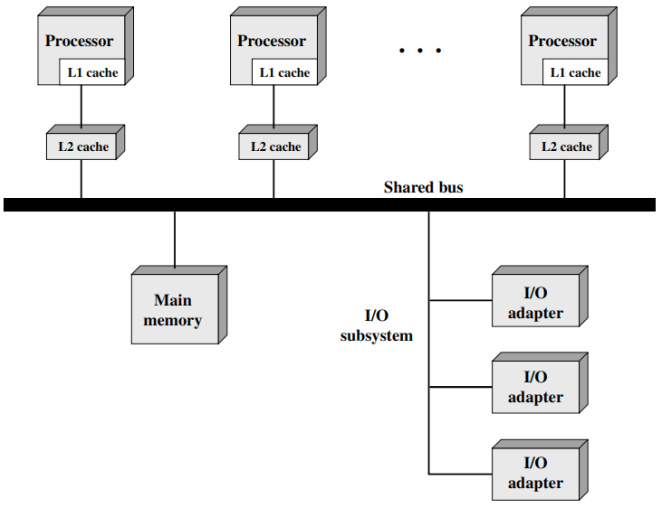


Figure 10.5: Symmetric Multiprocessor Organization.

The processors can communicate with each other through memory (messages and status information left in common data areas). It may also be possible for processors to exchange signals directly. The memory is often organized so that multiple simultaneous

accesses to separate blocks of memory are possible. In some configurations, each processor may also have its own private **main memory** and I/O channels in addition to the shared resources.

The most common organization for personal computers, workstations, and servers is the time-shared bus. The time-shared bus is the simplest mechanism for constructing a multiprocessor system (Figure 10.5). The structure and interfaces are basically the same 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 I/O processors, the following features are provided:

Keyword

Main memory refers to physical memory that is internal to the computer. The word main is used to distinguish it from external mass storage devices such as disk drives.

- ***Addressing:*** It must be possible to distinguish modules on the bus to determine the source and destination of data.
- ***Arbitration:*** Any I/O module can temporarily function as "master." A mechanism is provided to arbitrate competing requests for bus control, using some sort of priority scheme.
- ***Time-sharing:*** When one module is controlling the bus, other modules are locked out and must, if necessary, suspend 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 multiple I/O processors all attempting to gain access to one or more memory modules via the bus.

The bus organization has several attractive features:

- ***Simplicity:*** This is the simplest approach to multiprocessor organization. The physical interface and the addressing, arbitration, and time-sharing logic of each processor remain the same as in a single-processor system.
- ***Flexibility:*** It is generally easy to expand the system by attaching more processors to the bus.
- ***Reliability:*** The bus is essentially a passive medium, and the failure of any attached device should not cause failure of the whole system.

The main drawback to the bus organization is performance.All memory references pass through the common bus. Thus, the bus cycle time limits the speed of the system. To improve performance, it is desirable to equip each processor with a cache memory. This should reduce the number of bus accesses dramatically. Typically, workstation and PC SMPs have two levels of cache, with the L1 cache internal (same chip as the processor) and the L2 cache either internal or external. Some processors now employ a L3 cache as well.

The use of caches introduces some new design considerations. Because each local cache contains an image of a portion of memory, if a word is altered in one cache, it could conceivably invalidate a word in another cache. To prevent this, the other processors must be alerted that an update has taken place. This problem is known as the cache coherence problem and is typically addressed in hardware rather than by the operating system.

10.2.2 Multiprocessor Operating System Design Considerations

An SMP operating system manages processor 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 SMP and uniprocessor cases, multiple jobs or processes may be 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 must provide all the functionality of a multiprogramming system plus additional features to accommodate multiple processors. Among the key design issues:

- ***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 same or different parts of the OS, OS tables and management structures must be managed properly to avoid deadlock or invalid operations.
- ***Scheduling:*** Any processor may perform scheduling, so conflicts must be avoided. The scheduler must assign ready processes to available processors.
- ***Synchronization***: With multiple active processes having potential access to shared address spaces or shared I/O 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. In addition, the operating system needs to exploit the available hardware parallelism, such as multiported

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***: The 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

10.2.3 A Mainframe SMP

Most PC and workstation SMPs use a bus interconnection strategy as depicted in Figure 10.5. It is instructive to look at an alternative approach, which is used for a recent implementation of the IBM zSeries mainframe family, called the z990. This family of systems spans a range from a uniprocessor with one main memory card to a high-end system with 48 processors and 8 memory cards. The key components of the configuration are shown in Figure 10.6:

- ***Dual-core processor chip***: Each processor chip includes two identical central processors (CPs).The CP is a CISC superscalar microprocessor, in which most of the instructions are hardwired and the rest are executed by vertical microcode. Each CP includes a 256-KB L1 instruction cache and a 256-KB L1 data cache.
- ***L2 cache***: Each L2 cache contains 32 MB. The L2 caches are arranged in clusters of five, with each cluster supporting eight processor chips and providing access to the entire main memory space.
- ***System control element (SCE):*** The SCE arbitrates system communication, and has a central role in maintaining cache coherence.
- ***Main store control (MSC):*** The MSCs interconnect the L2 caches and the main memory.
- ***Memory card:*** Each card holds 32 GB of memory. The maximum configurable memory consists of 8 memory cards for a total of 256 GB. Memory cards interconnect to the MSC via synchronous memory interfaces (SMIs).
- ***Memory bus adapter (MBA):*** The MBA provides an interface to various types of I/O channels. Traffic to/from the channels goes directly to the L2 cache.

The microprocessor in the z990 is relatively uncommon compared with other modern processors because, although it is superscalar, it executes instructions in strict architectural order. However, it makes up for this by having a shorter pipeline and much larger caches and TLBs compared with other processors, along with other performance-enhancing features.

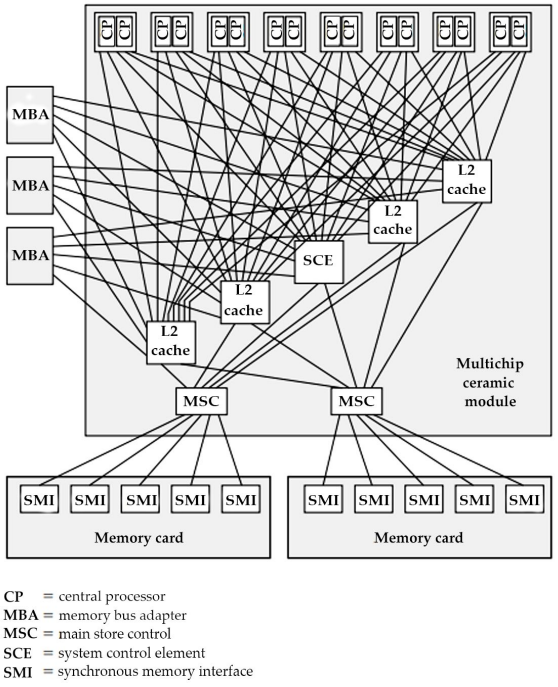


Figure 10.6: IBM z990 Multiprocessor Structure.

The z990 system comprises one to four books. Each book is a pluggable unit containing up to 12 processors with up to 64 GB of memory, I/O adapters, and a system control element (SCE) that connects these other elements. The SCE within each book contains a 32-MB L2 cache, which serves as the central coherency point for that particular book. Both the L2 cache and the main memory are accessible by a processor or I/O adapter within that book or any of the other three books in the system. The SCE and L2 cache chips also connect with corresponding elements on the other books in a ring configuration.

There are a several interesting features in the z990 SMP configuration, which we discuss in turn:

- Switched interconnection
- Shared L2 caches

Switched Interconnection

A single shared bus is a common arrangement on SMPs for PCs and workstations (Figure 10.5). With this arrangement, the single bus becomes a bottleneck affecting the scalability (ability to scale to larger sizes) of the design. The z990 copes with this problem in two ways. First, main memory is split into multiple cards, each with its own storage controller that can handle memory accesses at high speeds. The average

traffic load to main memory is cut, because of the independent paths to separate parts of memory. Each book includes two memory cards, for a total of eight cards across a maximum configuration. 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. Each processor chip has a link to each of the L2 caches on the same book, and each L2 cache has a link, via the MSC, to each of the two memory cards on the same book. Each L2 cache only connects to the two memory cards on the same book. The system controller provides links (not shown) to the other books in the configuration, so that all of main memory is accessible by all of the processors.

Point-to-point links rather than a bus also provides connections to I/O channels. Each L2 cache on a book connects to each of the MBAs for that book. The MBAs, in turn, connect to the I/O channels.

Shared L2 Caches

In a typical two-level cache scheme for an SMP, each processor has a dedicated L1 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 mainframe SMP, known as generation 3 (G3), IBM made use of dedicated L2 caches. In its later versions (G4, G5, and z900 series), a shared L2 cache is used. Two considerations dictated this change:

1. In moving from G3 to G4, IBM doubled the speed of the microprocessors. If the G3 organization were retained, a significant increase in bus traffic would occur. At the same time, it was desired to reuse as many G3 components as possible. Without a significant bus upgrade, the BSNs would become a bottleneck.
2. Analysis of typical mainframe workloads revealed a large degree of sharing of instructions and data among processors.

These considerations led the 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 L1 cache). At first glance, sharing an L2 cache might seem a bad idea. Access to memory from processors should be slower because the processors must now contend for access to a single L2 cache. However, if a sufficient amount of data is in fact shared by multiple processors, then a shared cache can increase throughput rather than retard it. Data that are shared and found in the shared cache are obtained more quickly than if they must be obtained over the bus.

10.3 CACHE COHERENCE AND THE MESI PROTOCOL

In contemporary multiprocessor systems, it is customary to have one or two levels of cache associated with each processor. This organization is essential to achieve reasonable performance. It does, however, create a problem known as the cache coherence

problem. The essence of the problem is this: Multiple copies of the same data can exist in different caches simultaneously, and if processors are allowed to update their own copies freely, an inconsistent view of memory can result. 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 that a write-back 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.

We will briefly survey various approaches to the cache coherence problem and then focus on the approach that is most widely used: the MESI (modified/exclusive/shared/invalid) protocol. A version of this protocol is used on both the Pentium 4 and PowerPC 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 time. 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 software and hardware approaches is still instructive and is commonly used in surveying **cache coherence** strategies.

Keyword

Cache coherence is the uniformity of shared resource data that ends up stored in multiple local caches.

10.3.1 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 approaches are 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 be 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 may 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 and for enforcing the results.

REMEMBER

A consistency model must be chosen carefully by the system designer, and the programmer must keep the communication costs in mind in developing the software.

10.3.2 Hardware Solutions

Hardware-based solutions are generally referred to as cache coherence protocols. These solutions provide dynamic recognition at run time of potential inconsistency conditions. Because the problem is only dealt with when it actually arises, there is more effective use of caches, leading to improved performance over a software approach. In addition, these approaches are transparent to the programmer and the compiler, reducing the software development burden.

Hardware schemes differ in a number of particulars, including where the state information about data lines is held, how that information is organized, where coherence is enforced, and the enforcement mechanisms. In general, hardware schemes can be divided into two categories: directory protocols and snoopy protocols.

Directory Protocols

Directory protocols collect and maintain information about where copies of lines reside. Typically, there is a centralized controller that is part of the main memory controller, and a directory that is stored in main memory. The directory contains global state information about the contents of the various local caches. When an individual cache controller makes a request, the centralized controller checks and issues necessary commands for data transfer between memory and caches or between caches. It is also responsible for keeping the state information up to date; therefore, every local action that can affect the global state of a line must be reported to the central controller.

Typically, the controller maintains information about which processors have a copy of which lines. Before a processor can write to a local copy of a line, it must request 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 line, forcing each processor to invalidate its copy. After receiving acknowledgments back from each such processor, the controller grants exclusive access to the requesting processor. When another processor tries to read a line that is exclusively granted to another processor, it will send a miss notification to the controller. The controller then issues a command to the processor holding that line that requires the 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 of a central bottleneck and the overhead of communication between the various cache controllers and the central controller. However, they are effective in large-scale systems that involve multiple buses or some other complex interconnection scheme.

Snoopy Protocols

Snoopy protocols distribute the responsibility for maintaining cache coherence among all of the cache controllers in a multiprocessor. A cache must recognize when a line 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" on the network to observe these broadcasted notifications, and react accordingly.

Snoopy protocols are 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.

Two 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 be 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 be 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. Some 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 Pentium 4 and PowerPC. 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 same time coordinating across the distributed multiprocessor. This would not add any new principles but would greatly complicate the discussion.

10.3.3 The MESI Protocol

To provide cache consistency on an SMP, the data cache often supports a protocol known as MESI. For MESI, the data cache includes two status bits per tag, so that each line can be in one of four states:

- ***Modified***: The line in the cache has been modified (different from main memory) and is available only in this cache.

Table 10.1: MESI Cache Line States

	M Modified	E Exclusive	S Shared	I Invalid
This cache line valid?	Yes	Yes	Yes	No
The memory copy is . . .	out of date	valid	valid	—
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line . . .	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus

- ***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 10.1 summarizes the meaning of the four states. Figure 10.7 displays a state diagram for the MESI protocol. Keep in mind that each line of the cache has its own state bits and therefore its own realization of the state diagram.

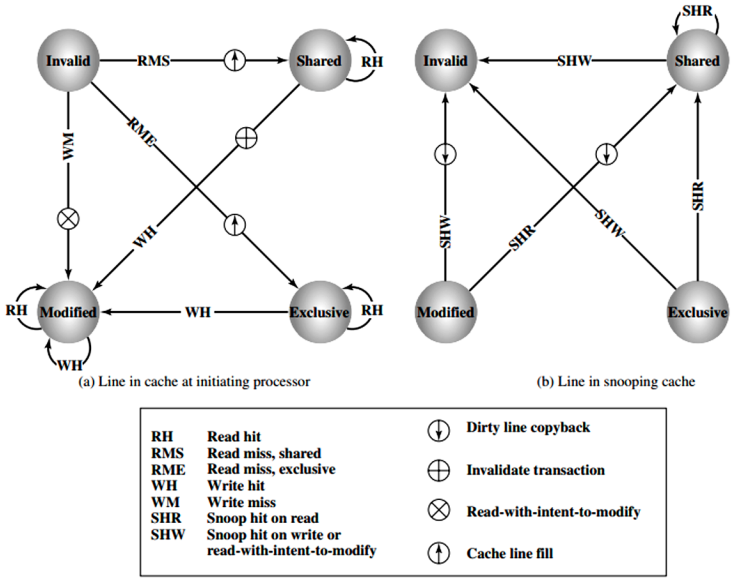


Figure 10.7: MESI State Transition Diagram.

Figure 10.7a shows the transitions that occur due to actions initiated by the processor attached to this cache. Figure 10.7b shows the transitions 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 MESI protocol. At any time a cache line is in a single state. If the next event is from the attached processor, then the transition is dictated by Figure 10.7a and if the next event is from the bus, the transition is dictated by Figure 10.7b.

Read Miss

When a read miss occurs in the local cache, the processor initiates a memory read to read the line of main memory containing the missing address. The processor inserts a signal on the bus that alerts all other processor/cache units to snoop the transaction. There are a number of possible outcomes:

- If one other cache has a clean (unmodified since read from memory) copy of the line in the exclusive state, it returns a signal indicating that it shares this line. The responding processor then transitions the state of its copy from exclusive to shared, and the initiating 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 that it shares the line. 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 responding cache then changes its line from modified to shared. The line sent to the requesting cache is also received and processed by the memory controller, which stores the block in memory.
- If no other cache has a copy of the line (clean or modified), then no signals are returned. The initiating processor reads the line and transitions the line in its cache from invalid to exclusive.

Read Hit

When a read hit occurs on a line currently in the local cache, the processor simply reads the required item. There is no state change: The state 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 line of main memory containing the missing address. For this purpose, the processor issues a signal on the bus that means read-with-intent-top-modify (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 data.

First, some other cache may have a modified copy of this line(state = modify). In this case, the alerted processor signals the initiating processor that another processor has a modified copy of the line. The initiating processor surrenders the bus and waits. The other processor gains access to the bus, writes the modified cache line back to main memory, and transitions the state of the cache line to invalid (because the initiating processor is going to modify this line). 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 modified 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 proceeds to read in the line 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 line 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 the 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 simply performs the update and transitions its copy of the line 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.

L1-L2 Cache Consistency

We have so far described cache coherency protocols in terms of the cooperate activity among caches connected to the same bus or other SMP interconnection facility. Typically, these caches are L2 caches, and each processor also has an L1 cache that does not connect directly to 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 SMP configuration.

The strategy is to extend the MESI protocol (or any cache coherence protocol) to the L1 caches. Thus, each line in the L1 cache includes bits 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 L1 cache, the L1 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 L1 cache; in this case the write through is to the L2 cache and not to the memory. The L1 write-through policy forces any modification to an L1 line out to the L2 cache and therefore makes it visible to other L2 caches. The use of the L1 writethrough policy requires that the L1 content must be a subset of the L2 content. This in turn suggests that the associativity of the L2 cache should be equal to or greater than that of the L1 associativity. The L1 write-through policy is used in the IBM S/390 SMP.

If the L1 cache has a write-back policy, the relationship between the two caches is more complex. There are several approaches to maintaining coherence.

10.4 MULTITHREADING AND CHIP MULTIPROCESSORS

The most important measure of performance for a processor is the rate at which it executes instructions. This can be expressed as

MIPS rate = f × IPC

where f is the processor clock frequency, in MHz, and IPC (instructions per cycle) is the average number of instructions executed per cycle. Accordingly, designers have pursued the goal of increased performance on two fronts: increasing clock frequency and increasing the number of instructions executed or, more properly, the number of instructions that complete during a processor cycle.

Designers have increased IPC by using an instruction pipeline and then by using multiple parallel instruction pipelines in a superscalar architecture. With pipelined and multiple-pipeline designs, the principal problem is to maximize the utilization of each pipeline stage.

To improve throughput, designers have created ever more complex mechanisms, such as executing some instructions in a different order from the way they occur in the instruction stream and beginning execution of instructions that may never be needed. This approach may be reaching a limit due to complexity and power consumption concerns.

An alternative approach, which allows for a high degree of instruction-level parallelism without increasing circuit complexity or power consumption, is called multithreading. In essence, the instruction stream is divided into several smaller streams, known as threads, such that the threads can be executed in parallel.

The variety of specific multithreading designs, realized in both commercial systems and experimental systems, is vast. In this section, we give a brief survey of the major concepts.

10.4.1 Implicit and Explicit Multithreading

The concept of thread used in discussing multithreaded processors may or may not be the same as the concept of software threads in a multiprogrammed operating system. It will be useful to define terms briefly:

- ***Process***: An instance of a program running on a computer. A process embodies two key characteristics:
  - *Resource ownership*: A process includes a **virtual address space** to hold the process image; the process image is the collection of program, data, stack, and attributes that define the process. From time to time, a process may be allocated control or ownership of resources, such as main memory, I/O channels, I/O devices, and files.
  - *Scheduling/execution*: The execution of a process follows an execution path (trace) through one or more programs. This execution may be interleaved with that of other processes. Thus, a process has an execution state (Running, Ready, etc.) and a dispatching priority and is the entity that is scheduled and dispatched by the operating system.
- ***Process switch***: An operation that switches the processor from one process to another, by saving all the process control data, registers, and other information for the first and replacing them with the process information for the second.
- ***Thread***: A dispatchable unit of work within a process. It includes a processor context (which includes the program counter and stack pointer) and its own data area for a stack (to enable subroutine branching). A thread executes sequentially and is interruptible so that the processor can turn to another thread.
- ***Thread switch***: The act of switching processor control from one thread to another within the same process. Typically, this type of switch is much less costly than a process switch.

Thus, a thread is concerned with scheduling and execution, whereas a process is concerned with both scheduling/execution and resource ownership. The multiple threads within a process

Keyword

Virtual Address Space is the collection/range of logical or virtual addresses for a particular process on a computer system.

share the same resources. This is why a thread switch is much less time consuming than a process switch. Traditional operating systems, such as earlier versions of UNIX, did not support threads.

Most modern operating systems, such as Linux, other versions of UNIX, and Windows, do support thread. A distinction is made between user-level threads, which are visible to the application program, and kernel-level threads, which are visible only to the operating system. Both of these may be referred to as explicit threads, defined in software.

All of the commercial processors and most of the experimental processors so far have used explicit multithreading. These systems concurrently execute instructions from different explicit threads, either by interleaving instructions from different threads on shared pipelines or by parallel execution on parallel pipelines.

Implicit multithreading refers to the concurrent execution of multiple threads extracted from a single sequential program. These implicit threads may be defined either statically by the compiler or dynamically by the hardware. In the remainder of this section we consider explicit multithreading

10.4.2 Approaches to Explicit Multithreading

At minimum, a multithreaded processor must provide a separate program counter for each thread of execution to be executed concurrently. The designs differ in the amount and type of additional hardware used to support concurrent thread execution. In general, instruction fetching takes place on a thread basis.

The processor treats each thread separately and may use a number of techniques for optimizing single-thread execution, including branch prediction, register renaming, and superscalar techniques. What is achieved is thread-level parallelism, which may provide for greatly improved performance when married to instruction-level parallelism.

Broadly speaking, there are four principal approaches to multithreading:

- ***Interleaved multithreading***: This is also known as fine-grained multithreading. The processor deals with two or more thread contexts at a time, switching from one thread to another at each clock cycle. If a thread is blocked because of data dependencies or memory latencies, that thread is skipped and a ready thread is executed.
- ***Blocked multithreading***: This is also known as coarse-grained multithreading. The instructions of a thread are executed successively until an event occurs

that may cause delay, such as a cache miss. This event induces a switch to another thread. This approach is effective on an in-order processor that would stall the pipeline for a delay event such as a cache miss.

- ***Simultaneous multithreading (SMT):*** Instructions are simultaneously issued from multiple threads to the execution units of a superscalar processor. This combines the wide superscalar instruction issue capability with the use of multiple thread contexts.
- ***Chip multiprocessing***: In this case, the entire processor is replicated on a single chip and each processor handles separate threads. The advantage of this approach is that the available logic area on a chip is used effectively without depending on ever-increasing complexity in pipeline design. This is referred to as multicore.

For the first two approaches, instructions from different threads are not executed simultaneously. Instead, the processor is able to rapidly switch from one thread to another, using a different set of registers and other context information. This results in a better utilization of the processor's execution resources and avoids a large penalty due to cache misses and other latency events. The SMT approach involves true simultaneous execution of instructions from different threads, using replicated execution resources. Chip multiprocessing also enables simultaneous execution of instructions from different threads.

Figure 8, based on one in [UNGE02], illustrates some of the possible pipeline architectures that involve multithreading and contrasts these with approaches that do not use multithreading. Each horizontal row represents the potential issue slot or slots for a single execution cycle; that is, the width of each row corresponds to the maximum number of instructions that can be issued in a single clock cycle.3 The vertical dimension represents the time sequence of clock cycles. An empty (shaded) slot represents an unused execution slot in one pipeline. A no-op is indicated by N.

The first three illustrations in Figure 8 show different approaches with a scalar (i.e., single-issue) processor:

- ***Single-threaded scalar***: This is the simple pipeline found in traditional RISC and CISC machines, with no multithreading.
- ***Interleaved multithreaded scalar***: This is the easiest multithreading approach to implement. By switching from one thread to another at each clock cycle, the pipeline stages can be kept fully occupied, or close to fully occupied. The hardware must be capable of switching from one thread context to another between cycles.

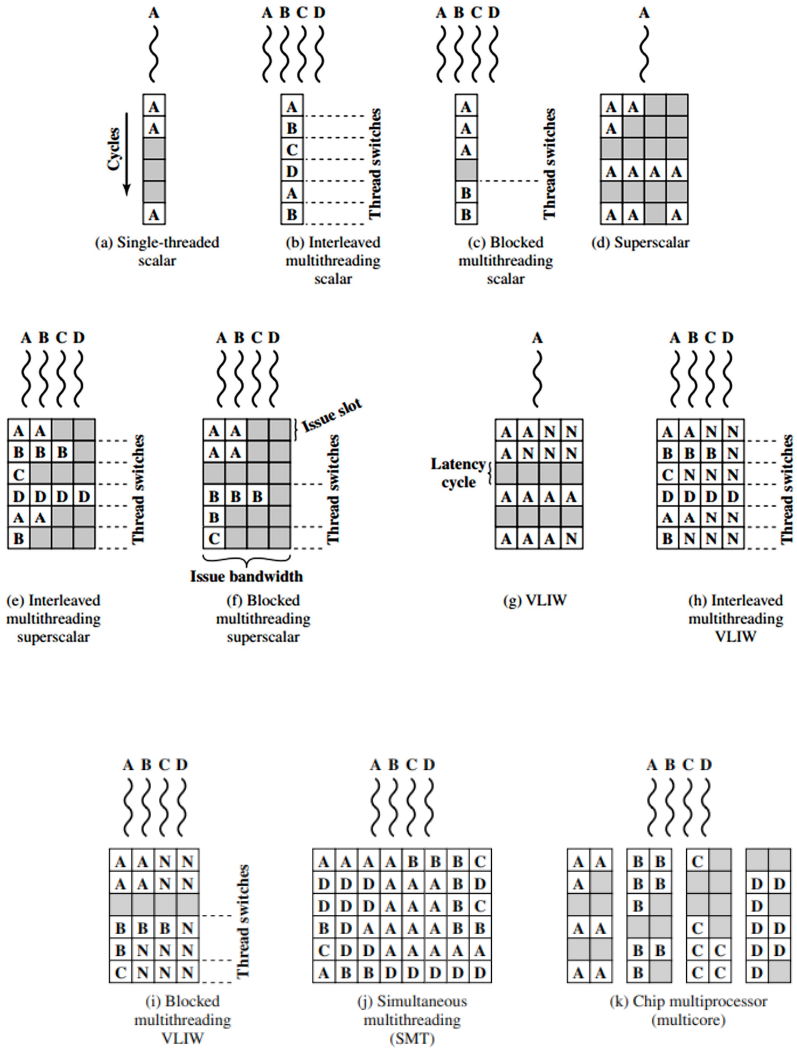


Figure 10.8: Approaches to Executing Multiple Threads.

- ***Blocked multithreaded scalar***: In this case, a single thread is executed until a latency event occurs that would stop the pipeline, at which time the processor switches to another thread.

Figure 10.8c shows a situation in which the time to perform a thread switch is one cycle, whereas Figure 8b shows that thread switching occurs in zero cycles. In the case of interleaved multithreading, it is assumed that there are no control or data dependencies between threads, which simplifies the pipeline design and therefore should allow a thread switch with no delay. However, depending on the specific design and implementation, block multithreading may require a clock cycle to perform a thread switch, as illustrated in Figure 10.8. This is true if a fetched instruction triggers the thread switch and must be discarded from the pipeline.

Although interleaved multithreading appears to offer better processor utilization than blocked multithreading, it does so at the sacrifice of single-thread performance. The multiple threads compete for cache resources, which raises the probability of a cache miss for a given thread.

More opportunities for parallel execution are available if the processor can issue multiple instructions per cycle. Figures 10.8d through 8i illustrate a number of variations among processors that have hardware for issuing four instructions per cycle. In all these cases, only instructions from a single thread are issued in a single cycle. The following alternatives are illustrated:

- ***Superscalar***: This is the basic superscalar approach with no multithreading. Until relatively recently, this was the most powerful approach to providing parallelism within a processor. Note that during some cycles, not all of the available issue slots are used. During these cycles, less than the maximum number of instructions is issued; this is referred to as horizontal loss. During other instruction cycles, no issue slots are used; these are cycles when no instructions can be issued; this is referred to as vertical loss.
- ***Interleaved multithreading superscalar***: During each cycle, as many instructions as possible are issued from a single thread. With this technique, potential delays due to thread switches are eliminated, as previously discussed. However, the number of instructions issued in any given cycle is still limited by dependencies that exist within any given thread.
- ***Blocked multithreaded superscalar***: Again, instructions from only one thread may be issued during any cycle, and blocked multithreading is used.
- ***Very long instruction word (VLIW)***: A VLIW architecture, such as IA-64, places multiple instructions in a single word. Typically, a VLIW is constructed by the compiler, which places operations that may be executed in parallel in the same word. In a simple VLIW machine (Figure 17.8g), if it is not possible to completely fill the word with instructions to be issued in parallel, no-ops are used.
- ***Interleaved multithreading VLIW***: This approach should provide similar efficiencies to those provided by interleaved multithreading on a superscalar architecture.
- ***Blocked multithreaded VLIW***: This approach should provide similar efficiencies to those provided by blocked multithreading on a superscalar architecture.

The final two approaches illustrated in Figure 10.8 enable the parallel, simultaneous execution of multiple threads:

- ***Simultaneous multithreading***: Figure 10.8i shows a system capable of issuing 8 instructions at a time. If one thread has a high degree of instruction-level parallelism, it may on some cycles be able fill all of the horizontal slots. On

other cycles, instructions from two or more threads may be issued. If sufficient threads are active, it should usually be possible to issue the maximum number of instructions on each cycle, providing a high level of efficiency.

- ***Chip multiprocessor (multicore):*** Figure 10.8k shows a chip containing four processors, each of which has a two-issue superscalar processor. Each processor is assigned a thread, from which it can issue up to two instructions per cycle.

Comparing Figures 10.8j and 10.8k, we see that a chip multiprocessor with the same instruction issue capability as an SMT cannot achieve the same degree of instruction-level parallelism. This is because the chip multiprocessor is not able to hide latencies by issuing instructions from other threads. On the other hand, the chip multiprocessor should outperform a superscalar processor with the same instruction issue capability, because the horizontal losses will be greater for the superscalar processor. In addition, it is possible to use multithreading within each of the processors on a chip multiprocessor, and this is done on some contemporary machines.

10.4.3 Example Systems

Pentium 4

More recent models of the Pentium 4 use a multithreading technique that the Intel literature refers to as hyperthreading [MARR02]. In essence, the Pentium 4 approach is to use SMT with support for two threads. Thus, the single multithreaded processor is logically two processors.

IBM Power5

The IBM Power5 chip, which is used in high-end PowerPC products, combines chip multiprocessing with SMT [KALL04]. The chip has two separate processors, each of which is a multithreaded processor capable of supporting two threads concurrently using SMT. Interestingly, the designers simulated various alternatives and found that having two two-way SMT processors on a single chip provided superior performance to a single four-way SMT processor. The simulations showed that additional multithreading beyond the support for two threads might decrease performance because of cache thrashing, as data from one thread displaces data needed by another thread.

Figure 9 shows the IBM Power5's instruction flow diagram. Only a few of the elements in the processor need to be replicated, with separate elements dedicated to separate threads. Two program counters are used. The processor alternates fetching instructions, up to eight at a time, between the two threads. All the instructions are stored in a common instruction cache and share an instruction translation facility, which does a partial instruction decode. When a conditional branch is encountered, the branch

prediction facility predicts the direction of the branch and, if possible, calculates the target address. For predicting the target of a subroutine return, the processor uses a return stack, one for each thread.

Instructions then move into two separate instruction buffers. Then, on the basis of thread priority, a group of instructions is selected and decoded in parallel. Next, instructions flow through a register-renaming facility in program order. Logical registers are mapped to physical registers. The Power5 has 120 physical general purpose registers and 120 physical floating-point registers. The instructions are then moved into issue queues. From the issue queues, instructions are issued using symmetric multithreading.

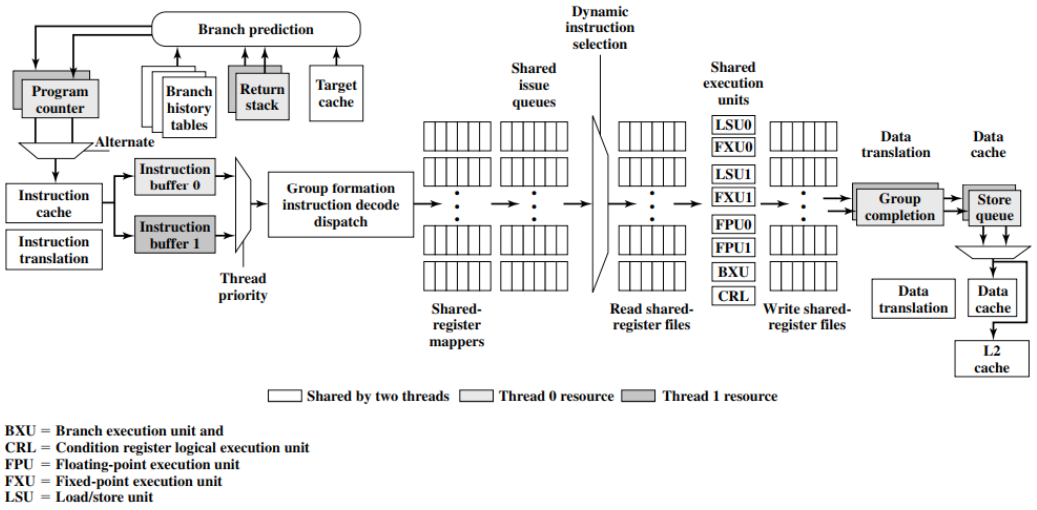


Figure 10.9: Power5 Instruction Data Flow.

That is, the processor has a superscalar architecture and can issue instructions from one or both threads in parallel. At the end of the pipeline, separate thread resources are needed to commit the instructions.

10.5 CLUSTERS

An important and relatively recent development computer system design is clustering. Clustering is an alternative to symmetric multiprocessing as an approach to providing high performance 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 being one machine. The term whole computer means a system that can run on its own, apart from the cluster; in the literature, each computer in a cluster is typically referred to as a node.

Four benefits that can be achieved with clustering. These can also be thought of as objectives or design requirements:

- ***Absolute scalability***: It is possible to create large clusters that far surpass the power of even the largest standalone machines. A cluster can have tens, hundreds, or even thousands of machines, each of which is a multiprocessor.
- ***Incremental scalability***: A cluster is configured in such a way that it is possible to 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, without having to go through a major upgrade in which an existing small system is replaced with a larger system.
- ***High availability***: Because each node in a cluster is a standalone computer, the failure of one node does not mean loss of service. In many products, fault tolerance is handled automatically in software.
- ***Superior price/performance***: By using commodity building blocks, it is possible to put together a cluster with equal or greater computing power than a single large machine, at much lower cost.

REMEMBER

If two packets at a node want to go out the same output, one must get priority (let's say it is the one from the left input).

10.5.1 Cluster Configurations

Clusters are classified in a number of different ways. Perhaps the simplest classification is based on whether the computers in a cluster share access to the same disks. Figure 10a shows a two-node cluster in which the only interconnection is by means of a high-speed link that can be used for message exchange to coordinate cluster activity. The link can be a LAN that is shared with other computers that are not part of the cluster or the link can be a dedicated interconnection facility. In the latter case, one or more of the computers in the cluster will have a link to a LAN or WAN so that there is a connection between the server cluster and remote client systems. Note that in the figure, each computer is depicted as being a multiprocessor. This is not necessary but does enhance both performance and availability.

In the simple classification depicted in Figure 10, the other alternative is a shared-disk cluster. In this case, there generally is still a message link between nodes.

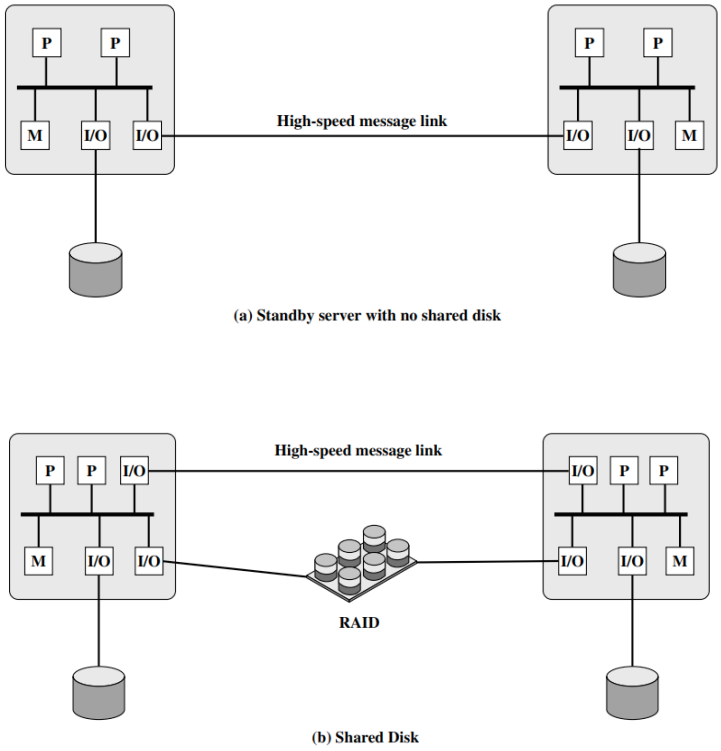


Figure 10.10: Cluster Configurations.

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 10.2 provides a useful classification along functional lines, which we now discuss.

A common, older method, known as passive standby, is simply to have one computer handle all of the processing load while the other computer remains inactive, standing by to take over in the event of a failure of 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 availability 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 but has no access to the databases managed by the primary.

Table 10.2: Clustering Methods: Benefits and Limitations

Clustering Method	Description	Benefits	Limitations
Passive Standby	A secondary server takes over in case of primary server failure.	Easy to implement.	High cost because the secondary server is unavailable for other processing tasks.
Active Secondary:	The secondary server is also used for processing tasks.	Reduced cost because secondary servers can be used for processing.	Increased complexity.
Separate Servers	Separate servers have their own disks. Data is continuously copied from primary to secondary server.	High availability.	High network and server overhead due to copying operations.
Servers Connected to Disks	Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server.	Reduced network and server overhead due to elimination of copying operations.	Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.
Servers Share Disks	Multiple servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires lock manager software. Usually used with disk mirroring or RAID technology.

The passive standby is generally not referred to as a cluster. The term cluster is reserved for multiple interconnected computers that are all actively doing processing while maintaining the image of a single system to the outside world. The term active secondary is often used in referring to this configuration. Three 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 and there are no disks shared between systems (Figure 10.10a). This arrangement provides high performance as well as high availability. In this case, some type of management or scheduling software is needed to assign incoming client requests to 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, another computer in the cluster can pick up and complete the application. For this to happen, data must constantly be copied among systems so that 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 10.10b). In one variation on this approach, called shared nothing, the common disks are partitioned into volumes, and each volume is 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 time (called the shared disk approach), so that each computer has access to all of the 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 time.

10.5.2 Operating System Design Issues

Full exploitation of a cluster hardware configuration requires some enhancements to a single-system operating system.

Failure Management

How failures are managed by a cluster depends on the clustering method used (Table 10.2). 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 be in service. If a failure occurs, 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 operating system makes no guarantee about the state of partially executed transactions. This would need to be handled at 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 committing completed transactions.

The function of switching applications and data resources over from a failed system to an alternative system in the cluster is referred to as failover. A related function is the restoration of applications and data resources to the original system once it has been fixed; this is referred to as failback. Failback can be automated, but this is desirable only if the problem is truly fixed and 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 automatically include this computer in scheduling applications. Middleware mechanisms need to recognize that services can appear on different members of the cluster and may migrate from one member to another.

Parallelizing Computation

In some cases, effective use of a cluster requires executing software from a single application in parallel. Three general approaches to the problem:

- *Parallelizing compiler*: A parallelizing compiler determines, at compile time, which parts of an application can be executed in parallel. These are then split off to be assigned to different computers in the cluster. Performance depends on the nature of the problem and how well the compiler is designed. In general, such compilers are difficult to develop.
- *Parallelized application*: In this approach, the programmer writes the application from the outset to run on a cluster, and uses message passing to move data, as required, between cluster nodes. This places a high burden on the programmer but may be the best approach for exploiting clusters for some applications.
- *Parametric computing*: This approach can be used if the essence of the application is an algorithm or program that must be executed a large number of times, each time with a different set of starting conditions or parameters. A good example is a simulation model, which will run a large number of different scenarios and then develop statistical summaries of the results. For this approach to be effective, parametric processing tools are needed to organize, run, and manage the jobs in an effective manner.

10.5.3 Cluster Computer Architecture

Figure 10.11 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 middleware layer of software is installed in each computer to enable cluster operation. The cluster middleware provides a unified system image to the user, known as a single-system image. The middleware is also responsible for providing high availability, by means of load balancing and responding to failures in individual components. The following as desirable cluster middleware services and functions:

- ***Single entry point***: A user logs onto the cluster rather than to an individual computer.
- Single file hierarchy: The user sees a single hierarchy of file directories under the same root directory.
- ***Single control point***: There is a default workstation used for cluster management 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. There is a single virtual network operation.

- ***Single memory space***: Distributed shared memory enables programs to share variables.
- ***Single job-management system***: Under a cluster job scheduler, a user can submit a job without specifying the host computer to execute the job.
- ***Single user interface***: A common graphic interface supports all users, regardless of the workstation from which they enter the cluster.
- ***Single I/O space***: Any node can remotely access any I/O peripheral or disk device without knowledge of its physical location.
- ***Single process space***: A uniform process-identification scheme is used. A process on any 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 rollback recovery after a failure.
- Process migration: This function enables load balancing.

The last four items on the preceding list enhance the availability of the cluster. The remaining items are concerned with providing a single system image.

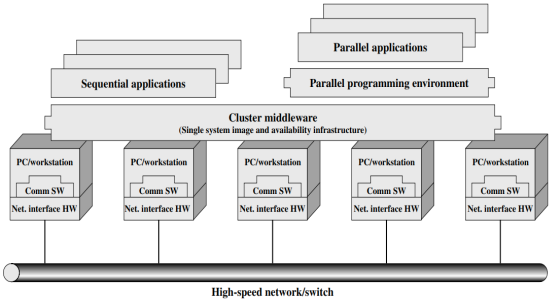


Figure 10.11: Cluster Computer Architecture.

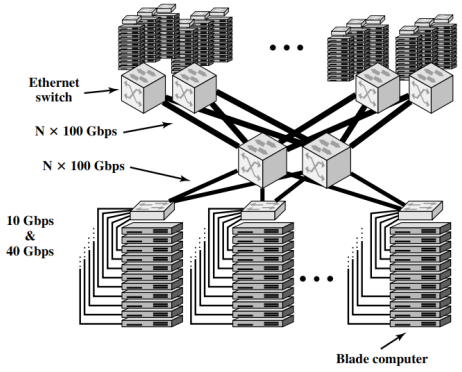


Figure 10.12: Example 100-Gbps Ethernet Configuration for Massive Blade Server Site.

Returning to Figure 10.11, a cluster will also include software tools for enabling the efficient execution of programs that are capable of parallel execution.

10.5.4 Blade Servers

A common implementation of the cluster approach is the blade server. A blade server is a server architecture that houses multiple server modules ("blades") in a single chassis. It is widely used in data centers to save space and improve system management. Either self-standing or rack mounted, the chassis provides the power supply, and each blade has its own processor, memory, and hard disk.

An example of the application is shown in Figure 10.12. The trend at large data centers, with substantial banks of blade servers, is the deployment of 10-Gbps ports on individual servers to handle the massive multimedia traffic provided by these servers. Such arrangements are stressing the on-site Ethernet switches needed to interconnect large numbers of servers.

A 100-Gbps rate provides the bandwidth required to handle the increased traffic load. The 100-Gbps Ethernet switches are deployed in switch uplinks inside the data center as well as providing interbuilding, intercampus, wide area connections for enterprise networks.

10.5.5 Clusters Compared to SMP

Both clusters and symmetric multiprocessors provide a configuration with multiple processors to support high-demand applications. Both solutions are commercially available, although SMP schemes have been around far longer.

The main strength of the SMP approach is that an SMP is easier to manage and configure than a cluster. The SMP is much closer to the original single-processor model for which nearly all applications are written.

The principal change required in going from a uniprocessor to an SMP is to the scheduler function. Another benefit of the SMP is that it usually takes up less physical space and draws less power than a comparable cluster. A final important benefit is that the SMP products are well established and stable.

Over the long run, however, the advantages of the cluster approach are likely to result in clusters dominating the high-performance server market. Clusters are far superior to SMPs in terms of incremental and absolute scalability. Clusters are also superior in terms of availability, because all components of the system can readily be made highly redundant.

10.6 NONUNIFORM MEMORY ACCESS

In terms 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 nonuniform memory access (NUMA), has been the subject of research and commercial NUMA products are now available.

Before proceeding, we should define some terms often found in the NUMA literature.

- ***Uniform memory access (UMA):*** All processors have access to all parts of main memory using loads and stores. The memory access time of a processor to all regions of memory is the same. The access times experienced by different processors are the same.
- ***Nonuniform memory access (NUMA):*** All processors have access to all parts of main memory 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 different processors, which memory regions are slower and which are faster differ.
- ***Cache-coherent NUMA (CC-NUMA):*** A NUMA system in which cache coherence is maintained among the caches of the various processors.

A NUMA system without cache coherence is more or less equivalent to a cluster. The commercial products that have received much attention recently are CC-NUMA systems, which are quite distinct from both SMPs and clusters. Usually, but unfortunately not always, such systems are in fact referred to in the commercial literature as CC-NUMA systems. This section is concerned only with CC-NUMA systems.

10.6.1 Motivation

With an SMP system, there is a practical limit to 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 cache-coherence signals, further adding to the burden. At some point, the bus becomes a performance bottleneck. Performance degradation seems to limit the number of processors in an SMP configuration to somewhere between 16 and 64 processors. For example, Silicon Graphics' Power Challenge SMP is 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 the development of cluster systems. However, with a cluster, each node has its own private main memory; applications do not see a large global memory. In effect, coherency is maintained in software rather than hardware. This memory granularity affects performance and, to achieve maximum performance, software must be tailored to this

environment. One approach to achieving large-scale multiprocessing while retaining the flavor of SMP is NUMA. For example, the Silicon Graphics Origin NUMA system is designed to support up to 1024 MIPS R10000 processors and the Sequent NUMA-Q system is designed to support up to 252 Pentium II processors.

The objective with NUMA is to maintain a transparent system wide memory while permitting multiple multiprocessor nodes, each with its own bus or other internal interconnect system.

10.6.2 Organization

Figure 13 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 L1 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 R10000 processors; each Sequent NUMA-Q 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-NUMA 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 system wide address. When a processor initiates a memory access, if the requested memory location is not in that processor's cache, then the L2 cache initiates a fetch operation. If 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 out to 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. 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:

1. 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 node 1.

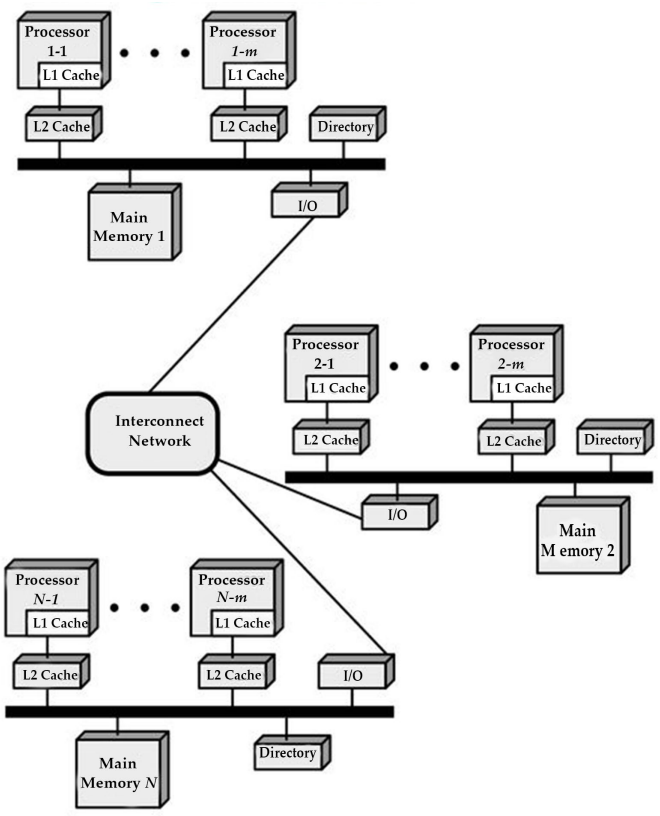


Figure 10.13: CC-NUMA Organization.

1. Node 2's directory sends a request to node 1, which is picked up by node 1's directory.
2. 1Node 1's directory, acting as a surrogate of P2-3, requests the contents of 798, as if it were a processor.
3. Node 1's main memory responds by putting the requested data on the bus.
4. Node 1's directory picks up the data from the bus.
5. The value is transferred back to node 2's directory.
6. Node 2's directory places the data back on node 2's bus, acting as a surrogate for the memory that originally held it.
7. The value is picked up and placed in P2-3's cache and delivered to P2-3.

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. We make only a few general remarks here. First, as part of the preceding sequence, node 1's directory keeps a record that some remote cache has a copy of the line containing location 798. Then, there needs to be

3G E-LEARNING

a cooperative protocol to take care of modifications. For example, if a modification is done in a cache, this fact can be broadcast to other nodes. Each node's directory that receives such a broadcast can then determine if any 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 back to update memory before providing the data.

10.6.3 NUMA Pros and Cons

The main advantage of a CC-NUMA system is that it can deliver effective performance at higher levels of parallelism than SMP, without requiring major software 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 are to remote nodes, performance begins to break down. There is reason to believe that this performance breakdown can be avoided. First, the use of L1 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 virtual 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 the running application. The Sequent designers report that such spatial locality does appear in representative applications. Finally, 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.

Even if the performance breakdown due to remote access is addressed, there are two other disadvantages for the CC-NUMA approach. Two in particular are discussed in detail. First, a CC-NUMA does not transparently look like an SMP; software changes will be required to move an operating system and applications from an SMP to a CC-NUMA system. These include page allocation, already mentioned, process allocation, and load balancing by the operating system. A second concern is that of availability. This is a rather complex issue and depends on the exact implementation of the CC-NUMA system;

10.7 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 characterized 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 continuous-field simulation. In essence, a physical situation can be 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 are typically capable of billions of floating-point operations per second. In contrast to mainframes, which are designed for multiprogramming and intensive I/O, 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 are 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.

There 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 not include scalar processing; they are configured as peripheral devices by both mainframe and minicomputer users to run the vectorized portions of programs.

10.7.1 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 arithmetic 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 B. We would like to add these and place the result in C. In the example of Figure 14, this requires six separate additions. How 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 × B, where

A,B, and C are N×N matrices. The formula for each element of C is

$$c_{i,j} = \sum_{k=1}^{N} a_{i,k} \times b_{k,j}$$

$$\begin{bmatrix} 1.5 \\ 7.1 \\ 6.9 \\ 100.5 \\ 0 \\ 59.7 \end{bmatrix} + \begin{bmatrix} 2.0 \\ 39.7 \\ 1000.003 \\ 11 \\ 21.1 \\ 19.7 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 46.8 \\ 1006.093 \\ 111.5 \\ 21.1 \\ 79.4 \end{bmatrix}$$

$$A \quad + \quad B \quad = \quad C$$

Figure 10.14: Example of Vector Addition.

where A, B, and C have elements $a_{i,j}$, $b_{i,j}$, and $c_{i,j}$, respectively. Figure 10.15a shows a FORTRAN program for this computation that can be run on an ordinary scalar processor.

One approach to improving performance can be referred to as vector processing. This assumes that it is possible to operate on a one-dimensional vector of data. Figure 10.15b is a FORTRAN program with a new form of instruction that allows

```
      DO 100 I = 1, N
      DO 100 J = 1, N
      C(I, J) = 0.0
      DO 100 K = 1, N
      C(I, J) = C(I, J) + A(I, K) + B(K, J)
100   CONTINUE
```

(a) Scalar processing

```
      DO 100 I = 1, N
      C(I, J) = 0.0 (J = 1, N)
      DO 100 K = 1, N
      C(I, J) = C(I, J) + A(I, K) + B(K, J) (J = 1, N)
100   CONTINUE
```

(b) Vector processing

```
      DO 50 J = 1, N − 1
      FORK 100
50    CONTINUE
      J = N
100   DO 200 I = 1, N
      C(I, J) = 0.0
      DO 200 K = 1, N
      C(I, J) = C(I, J) + A(I, K) + B(K, J)
200   CONTINUE
```

(c) Parallel processing

Figure 10.15: Matrix Multiplication (C = A × B).

vector computation to be specified. The notation (J = 1, N) indicates that operations on all indices J in the given interval are to be carried out as a single operation. How this can be achieved is addressed shortly.

The program in Figure 10.15b indicates that all the elements of the ith row are to be computed in parallel. Each element in the row is a summation, and the summations (across K) are done serially rather than in parallel. Even so, only N^2 vector multiplications are required for this algorithm as compared with N^3 scalar multiplications for the scalar algorithm.

Another approach, parallel processing, is illustrated in Figure 10.15c. This approach assumes that we have N independent processors that can function in parallel. To utilize processors effectively, we must somehow parcel out the computation to the various processors. Two primitives are used. The primitive FORK n causes an independent process to be started at location n. In the meantime, the original process continues execution at the instruction immediately following the FORK. Every execution of a FORK spawns a new process. The JOIN instruction is essentially the inverse of the FORK. The statement JOIN N causes N independent processes to be merged into one that continues execution at the instruction following the JOIN. The operating system must coordinate this merger, and so the execution does not continue until all N processes have reached the JOIN instruction.

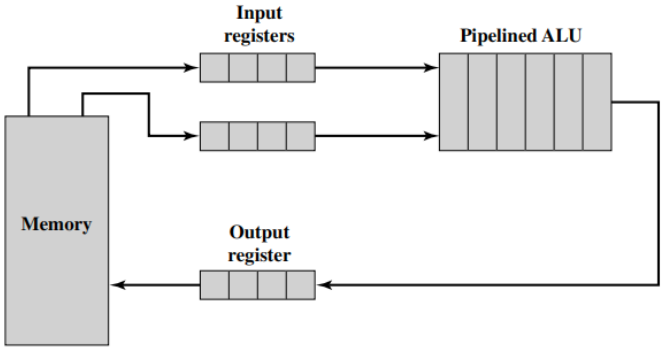
The program in Figure 10.15c is written to mimic the behavior of the vector processing program. In the parallel processing program, each column of C is computed by a separate process. Thus, the elements in a given row of C are computed in parallel.

The preceding discussion describes approaches to vector computation in logical or architectural terms. Let us turn now to a consideration of types of processor organization that can be used to implement these approaches. A wide variety of organizations have been and are being pursued. Three main categories stand out:

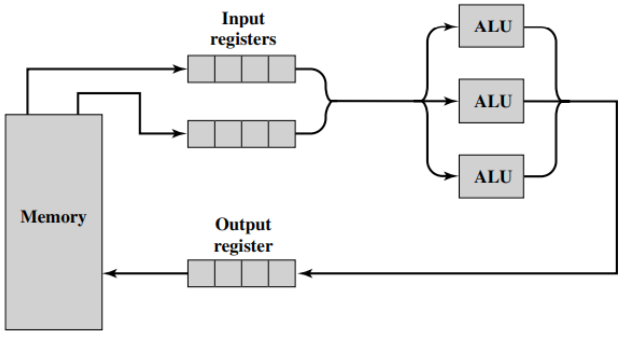
- Pipelined ALU
- Parallel ALUs
- Parallel processors

Figure 10.16 illustrates the first two of these approaches. Here the concept is extended to the operation of the ALU. Because floating-point operations are rather complex, there is opportunity for decomposing a floating-point operation into stages, so that different stages can operate on different sets of data concurrently. This is illustrated in Figure 10.17a. Floating-point addition is broken up into four stages: compare, shift, add, and normalize. A vector of numbers is presented sequentially to the first stage. As the processing proceeds, four different sets of numbers will be operated on concurrently in the pipeline.

It should be clear that this organization is suitable for vector processing. The processor goes through a repetitive cycle of fetching and processing instructions. In the absence of branches, the processor is continuously fetching instructions from sequential locations. Consequently, the pipeline is kept full and a savings in time is achieved. Similarly, a pipelined ALU will save time only if it is fed a stream of data from sequential locations.



(a) Pipelined ALU



(b) Parallel ALUs

Figure 10.16: Approaches to Vector Computation.

A single, isolated floating-point operation is not speeded up by a pipeline. The speedup is achieved when a vector of operands is presented to the ALU. The control unit cycles the data through the ALU until the entire vector is processed.

The pipeline operation can be further enhanced if the vector elements are available in registers rather than from main memory. This is in fact suggested by Figure 16a.The elements of each vector operand are loaded as a block into a vector register, which is simply a large bank of identical registers. The result is also placed in a vector register. Thus, most operations involve only the use of registers, and only load and store operations and the beginning and end of a vector operation require access to memory.

The mechanism illustrated in Figure 10.17 could be referred to as pipelining within an operation. That is, we have a single arithmetic operation (e.g., C = A + B) that is to be applied to vector operands, and pipelining allows multiple vector elements to be processed in parallel. This mechanism can be augmented with pipelining across operations. In this latter case, there is a sequence of 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.

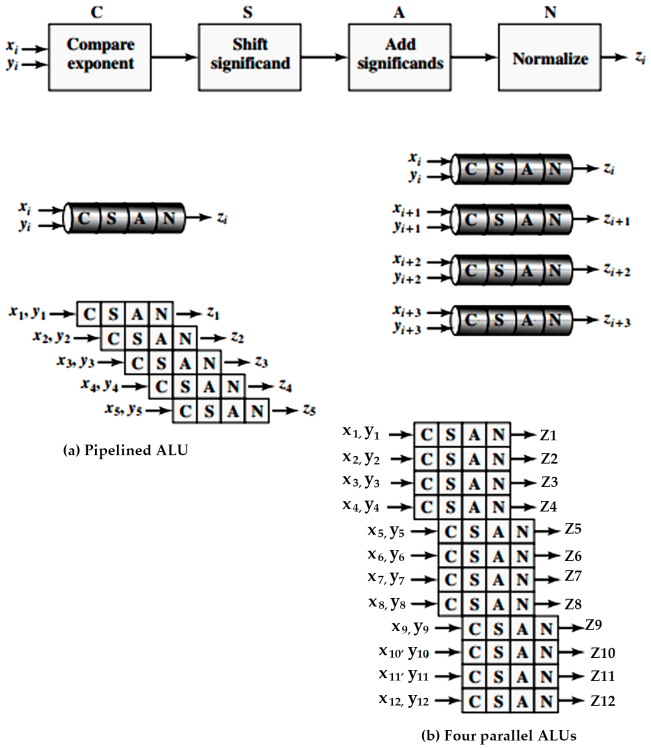


Figure 10.17: Pipelined Processing of Floating-Point Operations.

The basic rule for chaining is this: A vector operation may start as soon as the first element of the operand vector(s) is available and the functional unit (e.g., add, subtract, multiply, divide) is free. Essentially, chaining causes results issuing from one functional unit to be 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 = (s × A) + B, where A, B, and C are 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 to a pipelined adder, and the sums are placed in a vector register as soon as the adder completes them:

Vector load	A → Vector Register (VR1)
Vector load	B → VR2
Vector multiply	s × VR1 → VR3
Vector add	VR3 + VR2 → VR4
Vector store	VR4 → C

Instructions 2 and 3 can be chained (pipelined) because they involve different memory locations and registers. Instruction 4 needs the results of instructions 2 and 3, but it can be 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 ALUs in a single processor, under the control of a single control unit. In this case, the control unit routes data to ALUs 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 17b. 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 ALUs in a round-robin fashion until all elements are processed. This type of organization is more complex than a single-ALU CPI.

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.

The term vector processor is often equated with a pipelined ALU organization, although a parallel ALU organization is also designed for vector processing, and, as we have discussed, a parallel processor organization may also be designed for vector processing. Array processing is sometimes used to refer to a parallel ALU, although, again, any of the three organizations is optimized for the processing of arrays. To make matters worse, array processor usually refers to an auxiliary processor attached

to a general-purpose processor and used to perform vector computation. An array processor may use either the pipelined or parallel ALU approach.

At present, the pipelined ALU organization dominates the marketplace. Pipelined 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 high performance. The remainder of this section is devoted to a more detailed examination of this approach, using a specific example.

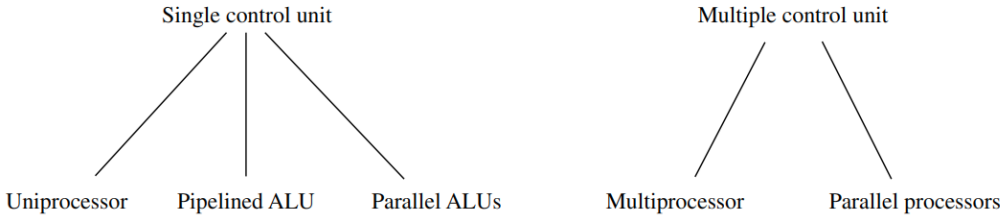


Figure 10.18: A Taxonomy of Computer Organizations.

10.7.2 IBM 3090 Vector Facility

A good example of a pipelined ALU organization for vector processing is the vector facility developed for the IBM 370 architecture and implemented on the high-end 3090 series [PADE88, TUCK87]. This facility is an optional add-on to the basic system but is highly integrated with it. It resembles vector facilities found on supercomputers, such as the Cray family.

The IBM facility makes use of a number of vector registers. Each register is actually a bank of scalar registers. To compute the vector sum the vectors A and B are loaded into two vector registers. The data from these registers are passed through the ALU as fast as possible, and the results are stored in a third vector register. The computation overlap, and the loading of the input data into the registers in a block, results in a significant speeding up over an ordinary ALU operation.

Organization

The IBM vector architecture, and similar pipelined vector ALUs, provides increased performance over loops of scalar arithmetic instructions in three ways:

- The fixed and predetermined structure of vector data permits housekeeping instructions inside the loop to be replaced by faster internal (hardware or microcoded) machine operations.

- Data-access and arithmetic operations on several successive vector elements can proceed concurrently by overlapping such operations in a pipelined design or by performing multiple-element operations in parallel.
- The use of vector registers for intermediate results avoids additional storage reference.

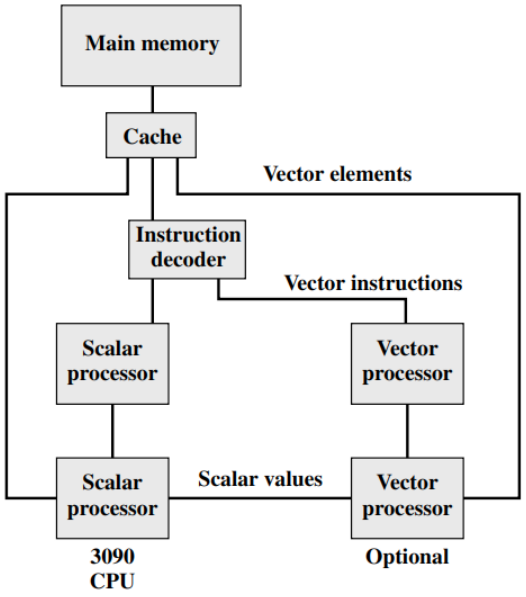
Figure 10.19 shows the general organization of the vector facility. Although the vector facility is seen to be a physically separate add-on to the processor, its architecture is an extension of the System/370 architecture and is compatible with it. The vector facility is integrated into the System/370 architecture in the following ways:

- Existing System/370 instructions are used for all scalar operations.
- Arithmetic operations on individual vector elements produce exactly the same result as do corresponding System/370 scalar instructions.
- Vector instructions are interruptible, and their execution can be resumed from the point of interruption after appropriate action has been taken, in a manner compatible with the System/370 program-interruption scheme.



One design decision concerned the definition of the result 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 one or more low-order bit positions? The decision was made to uphold complete compatibility with the System/370 architecture at the expense of a minor performance degradation.

- Arithmetic exceptions are the same as, or extensions of, exceptions for the scalar arithmetic instructions of the System/370, and similar fix-up routines can be used. To accommodate this, a vector interruption index is employed that indicates the location in a vector register that is affected by an exception (e.g., overflow). Thus, when execution of the vector instruction resumes, the proper place in a vector register is accessed.



- Vector data reside in virtual storage, with page faults being handled in a standard manner.

Figure 10.19: IBM 3090 with Vector Facility.

This level of integration provides a number of benefits. Existing operating systems can support the vector facility with minor extensions. Existing application programs, language compilers, and other software can be run unchanged. Software that could take advantage of the vector facility can be modified as desired.

Registers

A key issue in the design of a vector facility is whether operands are located in registers or memory. The IBM organization is referred to as register to 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 registers is that 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 K elements at a time and the loop is repeated times. The main advantage of the vector register approach is that the operation is decoupled from slower main memory and instead takes place primarily with registers.

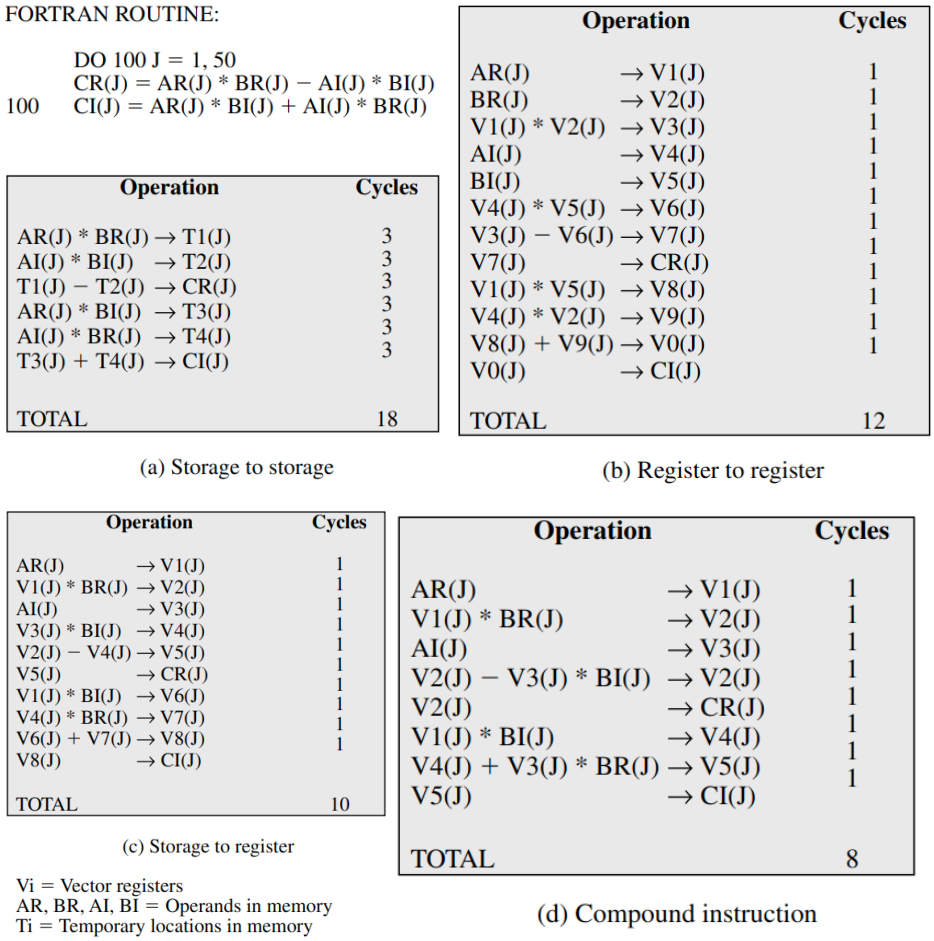


Figure 10.20: Alternative Programs for Vector Calculation.

The speedup that can be achieved using registers is demonstrated in Figure 10.20. The FORTRAN routine multiplies vector A by vector B to produce vector C, where each vector has a real part (AR, BR, CR) and an imaginary part (AI, BI, CI). The 3090 can perform one main-storage access per processor, or clock, cycle (either read or write); has registers that can sustain two accesses for reading and one for writing per cycle; and produces one result 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 computation requires a total of 18 cycles. With a pure register-to register architecture (part b), this time is reduced to 12 cycles.

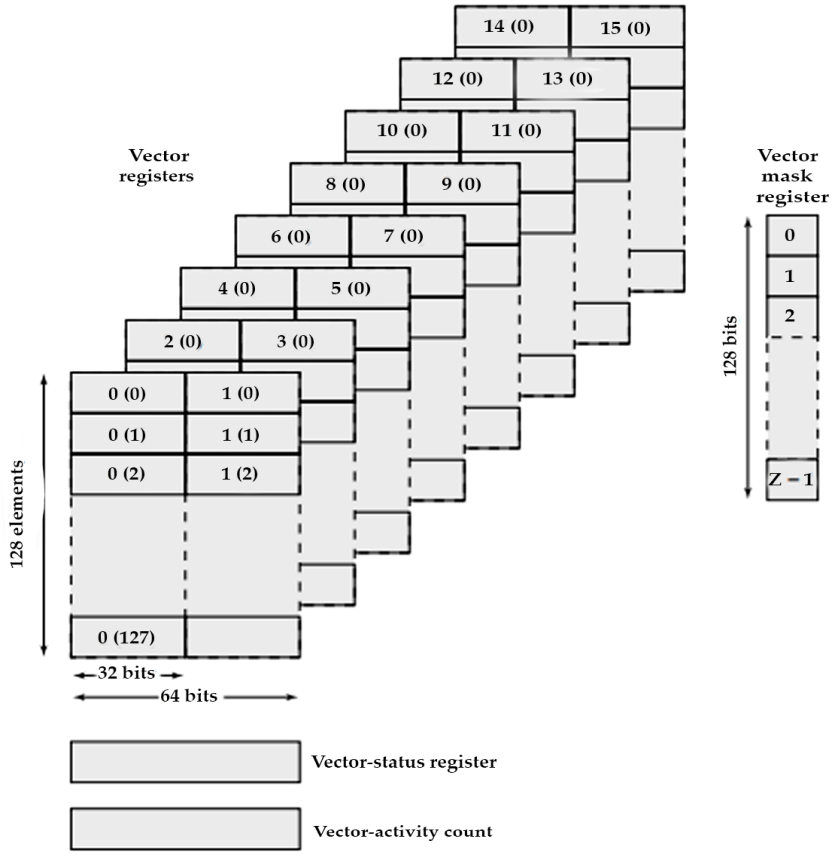


Figure 10.21: Registers for the IBM 3090 Vector Facility.

Of course, with register-to-register operation, the vector quantities must be loaded into the vector registers prior to computation and stored in memory afterward. For large vectors, this fixed penalty is relatively small. Figure 10.20c shows that the ability to specify both storage and register operands in one instruction further reduces the time to 10 cycles per iteration. This latter type of instruction is included in the vector architecture.

Figure 10.21 illustrates the registers that are part of the IBM 3090 vector facility. There are sixteen 32-bit vector registers. The vector registers can also be coupled to form eight 64-bit vector registers. Any register element can hold an integer or floating-point value. Thus, the vector registers may be used for 32-bit and 64-bit integer values, and 32-bit and 64-bit floating-point values.

The architecture specifies that each register contains from 8 to 512 scalar elements. 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 element. Thus, the use of a large number of register elements reduces

the relative startup time for a computation. However, this efficiency must be balanced against the added time required for saving and restoring vector registers on a process switch and the practical cost and space limits. These considerations led to the use of 128 elements per register in the current 3090 implementation.

Three additional registers are needed by the vector facility. The vector-mask register contains mask bits that may be used to select which elements in the vector registers are to be processed for a particular operation. The vector-status register contains control fields, such as the vector count, that determine how many elements in the vector registers are to be processed. The vector-activity count keeps track of the time spent executing vector instructions.

Compound Instructions

As was discussed previously, instruction execution can be overlapped using chaining to improve performance. The designers of the IBM vector facility chose not to include this capability for several reasons. The System/370 architecture would have to be extended to handle complex interruptions (including their effect on virtual memory management), and corresponding changes would be needed in the software. A more basic issue was the cost of including the additional controls and register access paths in the vector facility for generalized chaining.

Instead, three operations are provided that combine into one instruction (one opcode) the most common sequences in vector computation, namely multiplication followed by addition, subtraction, or summation. The storage-to-register MULTIPLYAND-ADD instruction, for example, fetches a vector from storage, multiplies it by a vector from a register, and adds the product to a third vector in a register. By use of the compound instructions MULTIPLY-AND-ADD and MULTIPLY-ANDSUBTRACT in the example of Figure 10.20, the total time for the iteration is reduced from 10 to 8 cycles.

Unlike chaining, compound instructions 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:

$$A \rightarrow VR1$$
$$VR1 + VR2 \rightarrow VR1$$

In this case, two stores to the vector register VR1 are required. In the IBM architecture there is a storage-to-register ADD instruction. With this instruction, only the sum is placed in VR1. The compound instruction also avoids the need to reflect in the machine-state description the concurrent execution of a number of instructions, which simplifies status saving and restoring by the operating system and the handling of interrupts.

The Instruction Set

Table 10.3 summarizes the arithmetic and logical operations that are defined for the vector architecture. In addition, there are memory-to register load and register-to-memory store instructions.

Table 10.3: IBM 3090 Vector Facility: Arithmetic and Logical Instructions

	Data Types						
	Floating-Point						
Operation	**Long**	**Short**	**Binary or Logical**	**Operand Locations**			
Add	FL	FS	BI	V + V → V	V + S → V	Q + V → V	Q + S → V
Subtract	FL	FS	BI	V − V → V	V − S → V	Q − V → V	Q − S → V
Multiply	FL	FS	BI	V × V → V	V × V → V	Q × V → V	Q × S → V
Divide	FL	FS	—	V/V → V	V/S → V	Q/V → V	Q/S → V
Compare	FL	FS	BI	V · V → V	V · S → V	Q · V → V	Q · S → V
Multiply and Add	FL	FS	—		V + V × S → V	V + Q × V → V	V + Q × S → V
Multiply and Subtract	FL	FS	—		V − V × S → V	V − Q × V → V	V − Q × S → V
Multiply and Accumulate	FL	FS	—	P + · V → V	P + · S → V		
Complement	FL	FS	BI	−V → V			
Positive Absolute	FL	FS	BI	|V| → V			
Negative Absolute	FL	FS	BI	−|V| → V			
Maximum	FL	FS	—			Q · V → Q	
Maximum Absolute	FL	FS	—			Q · V → Q	
Minimum	FL	FS	—			Q · V → Q	
Shift Left Logical	—	—	LO	· V → V			
Shift Right Logical	—	—	LO	· V → V			
And	—	—	LO	V & V → V	V & S → V	Q & V → V	Q & S → V
OR	—	—	LO	V|V → V	V|S → V	Q|V → V	Q|S → V
Exclusive-OR	—	—	LO	V ⊕ V → V	V ⊕ S → V	Q ⊕ V → V	Q ⊕ S → V

Explanation:

Data Types
- FL Long floating point
- FS Short floating point
- BI Binary integer
- LO Logical

Operand Locations
- V Vector register
- S Storage
- Q Scalar (general or floating-point register)
- P Partial sums in vector register
- · Special operation

Note that many of the instructions use a three-operand format. Also, many instructions have a number of variants, depending on the location of the 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, the result of which goes into the vector-mask register. With all these variants, the total number of opcodes (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 registers, and vector registers to the vector pipelines, the major hardware cost has been incurred. The architecture can, 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 3 are self-explanatory. The two summation instructions warrant further explanation. The accumulate 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 the sum of two numbers put into the pipeline is not available until several cycles later. Thus, the third element in the vector cannot be 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 particular, elements 0, 4, 8, 12, . . ., 124 are added in that order to produce partial sum 0; elements 1, 5, 9, 13, . . ., 125 to partial sum 1; elements 2, 6, 10, 14, . . ., 126 to partial sum 2; and elements 3, 7, 11, 15, . . ., 127 to partial sum 4. Each of these partial sums can proceed through the pipeline at top speed, because the delay in the pipeline is roughly four cycles. A separate vector register is used to hold the partial sums. When all elements of the original vector have been processed, the four partial sums are added together to produce the final result. The performance of this second phase is not critical, because only four vector elements are involved.

SUMMARY

- Most computer programming languages require the programmer to specify algorithms as sequences of instructions.
- A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each 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 this category.
- A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. This structure is not commercially implemented.
- A set of processors simultaneously execute different instruction sequences on different data sets. SMPs, clusters, and NUMA systems fit into this category.
- An SMP operating system manages processor and other computer resources so that the user perceives a single operating system controlling system resources.
- 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.
- Directory protocols collect and maintain information about where copies of lines reside. Typically, there is a centralized controller that is part of the main memory controller, and a directory that is stored in main memory.

KNOWLEDGE CHECK

1. Execution of several activities at the same time.

a. processing
b. parallel processing
c. serial processing
d. multitasking

2. Parallel processing has single execution flow.

a. True
b. False

3. A term for simultaneous access to a resource, physical or logical.

a. Multiprogramming
b. Multitasking
c. Threads
d. Concurrency

4. A parallelism based on increasing processor word size.

a. Increasing
b. Count based
c. Bit based
d. Bit level

5. A type of parallelism that uses micro architectural techniques.

a. instructional
b. bit level
c. bit based
d. increasing

REVIEW QUESTIONS

1. What are the chief characteristics of an SMP?
2. What is the difference between software and hardware cache coherent schemes?
3. What is the meaning of each of the four states in the MESI protocol?
4. What are some of the key benefits of clustering?
5. What are the differences among UMA, NUMA, and CC-NUMA?

Check Your Result

1. (b) 2. (b) 3. (d) 4. (d) 5. (a)

REFERENCES

1. Barney, Blaise. "Introduction to Parallel Computing". Lawrence Livermore National Laboratory. Retrieved 2007-11-09.
2. Flynn, Michael J. (September 1972). "Some Computer Organizations and Their Effectiveness" (PDF). IEEE Transactions on Computers. C-21 (9): 948–960.
3. Hennessy, John L.; Patterson, David A. (2002). Computer architecture / a quantitative approach (3rd ed.). San Francisco, Calif.: International Thomson. p. 43. ISBN 978-1-55860-724-8.
4. Rodriguez, C.; Villagra, M.; Baran, B. (29 August 2008). "Asynchronous team algorithms for Boolean Satisfiability". Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd: 66–69.
5. Thomas Rauber; Gudula Rünger (2013). Parallel Programming: for Multicore and Cluster Systems. Springer Science & Business Media. p. 1. ISBN 9783642378010.

INDEX

A

Address bus 352, 357, 359, 375
Architecture models 275
Arithmetic and logic unit (ALU) 351
Associative memory 138
Asynchronous transmissions 263
Auto increment 73

B

Basic input/output system (BIOS) 348
Binary Coded Decimal 19, 20
Bus arbitration mechanism 46

C

Cache coherence 379, 380, 387, 388, 390, 391, 392, 393, 397, 413, 414, 415, 431
Cache memory 143
Central Processing Unit (CPU) 38, 39
Chip multiprocessors 379, 380
Circuit theory 215, 225
Combinational logic circuit 202, 203
Compact disc (CD) 137
Component-based architecture 275, 329, 345
Computer system 5, 20, 28, 38, 39, 46, 47
Control unit (CU) 63

D

Data 25, 31, 36, 38, 39, 45, 46
Data bus topology 46
Data-centered architecture 275, 303
Data flow architecture 275, 300, 302
Data register 223, 236
Decode cycle 79
Decoder 203, 205, 207, 208, 213, 244
Demultiplexer 207, 208, 210, 211, 212
Digital Subscriber Line (DSL) 260
Digital versatile disk (DVD) 137
Direct addressing 102
Distributed architecture 275, 319, 320, 345
DMA controller 185, 186, 187, 188, 189, 190, 256, 258
DMA control register 188
DMA transfer cycle 190

E

Encoder 205, 206, 207
External interrupt 76
External RAM 67

F

Fetch cycle 78, 79
Finite-state machine (FSM) 220
Full adder 213
Full-duplex 262

Full-duplex transmission 262

H

Hierarchical architecture 275, 308, 313
High-level language 31

I

Index mode 70
Index registers 223
Instruction register (IR) 64, 78
Interaction-oriented architecture 275, 313, 345
Internal memory 130
Internal peripherals 167
Internal RAM 67
Interrupt service routine (ISR) 75

K

Keyboard 166, 167, 169, 170, 172, 178, 184, 193, 198

L

Logic gate 203, 204, 205

M

Main memory 127, 130, 131, 132, 135, 140, 141, 142, 143, 144, 145, 146, 147, 148, 161, 162
Memory system 128, 129
Memory unit 139, 140
Micro-counter 109, 110, 111, 112
Microinstruction 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118
Microprocessors 370, 375
Micro programs 92, 113, 114
Modem 259
Mouse 168
MOV instruction 98
Multiple processor 379
Multiplexed bus 47
Multiplexer 202, 204, 205, 208, 209, 210, 211

N

Nonuniform memory access 379, 380, 413

O

Object-oriented paradigm 275, 295, 345

P

Pointer register 223
Priority Encoder 205
Program Counter (PC) 78
Punched card 28
PUSH instruction 99

Q

QWERTY design 169

R

Remote processing 31
ROM chips 133, 134

S

Sequential logic 201, 202, 215, 216, 225, 226, 230
Software architecture 275, 281, 284, 285, 288, 290, 291, 292, 313, 330, 345
Stack 73, 79
Symmetric multiprocessors 379, 380, 412

U

Universal Serial Bus (USB) 167

V

Vector computation 379, 380, 417, 419, 423, 428
Virtual memory 150, 151, 152

X

XCHG instruction 99

Level: Beginner to Advanced
Subject: Computer and Information Science

Computer Systems & Architecture

2nd Edition

Computer is an electronic machine that makes performing any task very easy. In computer, the CPU executes each instruction provided to it, in a series of steps, this series of steps is called Machine Cycle, and is repeated for each instruction. One machine cycle involves fetching of instruction, decoding the instruction, transferring the data, executing the instruction. Computer architecture is a specification detailing how a set of software and hardware technology standards interact to form a computer system or platform. In short, computer architecture refers to how a computer system is designed and what technologies it is compatible with. As with other contexts and meanings of the word architecture, computer architecture is likened to the art of determining the needs of the user/system/technology, and creating a logical design and standards based on those requirements. Modern computer technology requires an understanding of both hardware and software, since the interaction between the two offers a framework for mastering the fundamentals of computing.

This edition is comprised of ten chapters. It deals with the design and performance evaluation of advanced/high- performance computer systems. The emphasis is on microprocessors, chip-multiprocessors and memory hierarchy design.

ISBN 978-1-98467-618-4

BIBLIOTEX
Digital Library

www.3ge-learning.com
email: info@3ge-learning.com