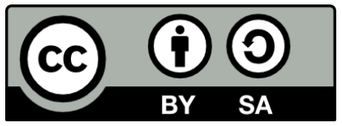
CSC 230: COMPUTER ARCHITECTURE

Published by the Centre for Open and Distance Learning,
University of Ilorin, Nigeria

E-mail: codl@unilorin.edu.ng
Website: https://codl.unilorin.edu.ng



Course Development Team

Subject Matter Expert

Rasheed Gbenga JIMOH, Ph.D.

Instructional Designers

Koledafe S. Olawale
Samuel A. Adekanye
Hassan Selim Olarewaju

Language Editors

Mahmud Abdulwahab

From the Vice Chancellor

Courseware development for instructional use by the Centre for Open and Distance Learning (CODL) has been achieved through the dedication of authors and the team involved in quality assurance based on the core values of the University of Ilorin. The availability, relevance and use of the courseware cannot be timelier than now that the whole world has to bring online education to the front burner. A necessary equipping for addressing some of the weaknesses of regular classroom teaching and learning has thus been achieved in this effort.

This basic course material is available in different electronic modes to ease access and use for the students. They are available on the University's website for download to students and others who have interest in learning from the contents. This is UNILORIN CODL's way of extending knowledge and promoting skills acquisition as open source to those who are interested. As expected, graduates of the University of Ilorin are equipped with requisite skills and competencies for excellence in life. That same expectation applies to all users of these learning materials.

Needless to say, that availability and delivery of the courseware to achieve expected CODL goals are of essence. Ultimate attention is paid to quality and excellence in these complementary processes of teaching and learning. Students are confident that they have the best available to them in every sense.

It is hoped that students will make the best use of these valuable course materials.

Professor S. A. Abdulkareem
Vice Chancellor

Foreword

Courseware remains the nerve centre of Open and Distance Learning. Whereas some institutions and tutors depend entirely on Open Educational Resources (OER), CODL at the University of Ilorin considers it necessary to develop its own materials. Rich as OERs are and widely as they are deployed for supporting online education, adding to them in content and quality by individuals and institutions guarantees progress. Doing it in-house as we have done at the University of Ilorin has brought the best out of the Course Development Team across Faculties in the University. Credit must be given to the team for prompt completion and delivery of assigned tasks in spite of their very busy schedules.The development of the courseware is similar in many ways to the experience of a pregnant woman eagerly looking forward to the D-day when she will put to bed. It is customary that families waiting for the arrival of a new baby usually do so with high hopes. This is the apt description of the eagerness of the University of Ilorin in seeing that the centre for open and distance learning [CODL] takes off.

The Vice-Chancellor, Prof. Sulyman Age Abdulkareem, deserves every accolade for committing huge financial and material resources to the centre. This commitment, no doubt, boosted the efforts of the team. Careful attention to quality standards, ODL compliance and UNILORIN CODL House Style brought the best out from the course development team. Responses to quality assurance with respect to writing, subject matter content, language and instructional design by authors, reviewers, editors and designers, though painstaking, have yielded the course materials now made available primarily to CODL students as open resources.

Aiming at a parity of standards and esteem with regular university programmes is usually an expectation from students on open and distance education programmes. The reason being that stakeholders hold the view that graduates of face-to-face teaching and learning are superior to those exposed to online education. CODL has the dual-mode mandate. This implies a combination of face-to-face with open and distance education. It is in the light of this that our centre has developed its courseware to combine the strength of both modes to bring out the best from the students. CODL students, other categories of students of the University of Ilorin and similar institutions will find the courseware to be their most dependable companion for the acquisition of knowledge, skills and competences in their respective courses and programmes.

Activities, assessments, assignments, exercises, reports, discussions and projects amongst others at various points in the courseware are targeted at achieving the objectives of teaching and learning. The courseware is interactive and directly points the attention of students and users to key issues helpful to their particular learning. Students' understanding has been viewed as a necessary ingredient at every point. Each course has also been broken into modules and their component units in sequential order.

At this juncture, I must commend past directors of this great centre for their painstaking efforts at ensuring that it sees the light of the day. Prof. M. O. Yusuf, Prof. A. A. Fajonyomi and Prof. H. O. Owolabi shall always be remembered for doing their best during their respective tenures. May God continually be pleased with them, Aameen.

Bashiru, A. Omipidan
Director, CODL

INTRODUCTION

I welcome you to Internet Technology I, a second-semester course. Internet Technology I is a two (2) unit course that provides a general introduction to Internet Technology, covering a brief history of the Internet, how it grew from its humble origins into the worldwide network that is available today, identifying the most popular Internet services such as information retrieval, WWW and communication services.

The relationship between the Internet and the World Wide Web is discussed. The course provides you with comprehensive knowledge on the concepts of Internet Technology, which include its internet architecture and internet protocol. The two most important protocols that allow networks to communicate with one another and exchange information, that is the TCP (Transmission Control Protocol) and IP (Internet Protocol), are also discussed.

Also, the functions of each layer at the TCP/IP networking model are covered. The brief history of HTML, XML, XHTML and DHTML is addressed. The course also covers in depth, HTML5, CSS and Javascript. The course also discusses the concept of a markup language and how to create web pages using HTML5 elements, CSS and Javascript. the course also discusses other equally important topics like WYSIWYG, Test Editors, including notepad, notepad++ and others.

Course Goal

The major goal of this course, CSC 224,is to introduce you to the concept of Internet technology and teach you how to develop websites using available technologies.

WORK PLAN

Learning Outcomes

At the end of this course, you should be able to:

- Explain the fundamentals of Digital Computer
- Describe the history of computer developments and the tangible deliverables

Week 01

- Describe data presentation, storage development and management in computing
- Describe basic number systems and their implications to computer architecture

Week 02

- Describe how traditional computer architecture is designed and how it operates.s

Week 03

Course Infomation

This is a compulsory course for students in the Departments of Chemistry, Industrial Chemistry, Geology, Mathematics, Computer Science. You are expected to participate in all the course activities and have minimum of 75% attendance to be able to write the final examination

Pre-requisite

CSC 230
Introduction to Digital Design and Microprocessors

Course Guide

Module 1

DIGITAL LOGIC AND MEMORY SYSTEMS

Unit 1: Fundamental Building Blocks of Digital Logic

Unit 2: Storage Systems and their Technology

Unit 3: Data Representation and Number Systems

Unit 4: Memory hierarchy, organization and operations

Unit 5: Cache Memory

Module 2

VIRTUAL MEMORY, OVERLAYS, PAGING, SEGMENTATION AND FRAGMENTATION

Unit 1: Virtual memory, overlays and paging

Unit 2: Segmentation and Fragmentation

Module 3

INTERFACING AND COMMUNICATION

Unit 1: Input /Output Fundamentals

Unit 2: Interrupt-driven and DMA

Unit 3: Introduction to Buses

Module 4

HANDSHAKING AND BUFFERS

Unit 1: Handshaking

Unit 2: Data Buffer

Unit 3: External Storage

Module 5

INTRODUCTION TO NETWORKS, RAID ARCHITECTURES, DATA PATH & CONTROL

Unit 1: Multimedia Support RAID Architectures

Unit 2: Data Path and Control Unit

Module 6

INSTRUCTION PIPELINING, RISCs & MULTIPROCESSORS

Unit.1: Pipelining Design Techniques

Unit 2: Introduction to Reduced Instruction Set Computers

Unit 3: Introduction to Multiprocessors

Unit 4: SIMD and MIMD Schemes

Course Requirements

Requirements for success

The CODL Programme is designed for learners who are absent from the lecturer in time and space. Therefore, you should refer to your Student Handbook, available on the website and in hard copy form, to get information on the procedure of distance/e-learning. You can contact the CODL helpdesk which is available 24/7 for every of your enquiry.

Visit CODL virtual classroom on http://codllms.unilorin.edu.ng. Then, log in with your credentials and click on CSC 230. Download and read through the unit of instruction for each week before the scheduled time of interaction with the course tutor/facilitator. You should also download and watch the relevant video and listen to the podcast so that you will understand and follow the course facilitator.

At the scheduled time, you are expected to log in to the classroom for interaction.

Self-assessment component of the courseware is available as exercises to help you learn and master the content you have gone through.

You are to answer the Tutor Marked Assignment (TMA) for each unit and submit for assessment.

Embedded Support Devices

Support menus for guide and references

Throughout your interaction with this course material, you will notice some set of icons used for easier navigation of this course materials. We advise that you familiarize yourself with each of these icons as they will help you in no small ways in achieving success and easy completion of this course. Find in the table below, the complete icon set and their meaning.

 **Introduction**	 **Learning Outcomes**	 **Main Content**

 **Summary**	 **Tutor Marked Assignment**	 **Self Assessment**
 **Web Resources**	 **Downloadable Resources**	 **Discuss with Colleagues**
 **References**	 **Futher Reading**	 **Self Exploration**

Grading and Assessment

TMA

CA

Exam

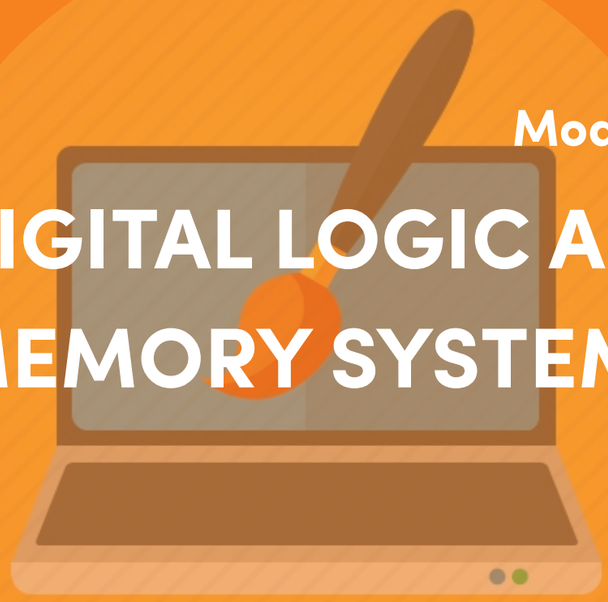
Total

01 | **Picture:** Digital logic

Photo: Wikipedia.com

Module 1

DIGITAL LOGIC AND MEMORY SYSTEMS



02 | **Picture:** Fundamental building blocks

Photo: Wikipedia.com

UNIT 1

Fundamental Building Blocks of Digital Logic

Introduction

I welcome you to the first module and the first unit of this course. I want you to know that digital Logic serves as the fundamental building blocks of circuit design. You should also be aware that mastery of mathematical logic and Boolean algebra is required in Digital Logic. Boolean algebra is the branch of algebra in which the values of the variables are the truth-values: true and false, usually denoted by 1 and 0, respectively. This course introduces you to the applications of algebra in digital logic design.

Learning Outcomes

At the end of this unit, you should be able to:

1. Define Boolean algebra,
2. Explain the basic operation in Boolean algebra,
3. Relate Boolean operations to equivalent truth tables and logic networks.

Main Content

DIGITAL LOGIC

10 mins

SAQ 1

Boolean Algebra
Definition and Concepts of Boolean Algebra

Do you know that Boolean algebra (or Boolean logic) is a logical calculus of truth values, developed by George Boole in the 1840s? It resembles the algebra of real numbers, but with the numeric operations of multiplication x X y, addition x + y, and negation ¯x replaced by the respective logical operations of conjunction x∧y, disjunction x∨y, and negation ¯x.

Boolean Operations

Be informed that Boolean operations are all operations that can be built from these, such as x∧(y∨z). These turn out to coincide with the set of all operations on the set {0,1} that take only finitely many arguments; there are 2^{2n} such operations when there are n arguments.

Boolean Laws

You should note that laws of Boolean algebra can be defined axiomatically as certain equations called axioms together with their logical consequences called theorems, or semantically as those equations that are true for every possible assignment of 0 or 1 to their variables. The axiomatic approach is sound and complete in the sense that it proves respectively, neither more nor fewer laws than the semantic approach. Boolean algebra is used for designing and analyzing digital circuits. First, we will discuss the rules of Boolean algebra, and thereafter, we will discuss how it can be used in analyzing or designing digital circuits.

Law 1: Only 2 possible states

A variable in Boolean algebra can take only two values. These are called states.

1 (TRUE) or 0 (FALSE)

Law 2: Three basic operations exist

We have three basic operations in Boolean algebra. These include:

AND, OR and NOT

(These operators will be given in capitals in this module for differentiating them from normal and, or, not, etc.)

A AND B or A.B or AB

A OR B or A + B

NOT A or ¬A or A' or Ā

But how the value of A AND B changes with the values of A and B can be represented in tabular form, which is referred to as the "truth table."

Table 1: Truth table

A	B	A AND B	A OR B	NOT A	NOT B
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

In addition, three more operators have been defined for Boolean algebra:

XOR (Exclusive OR), NOR (Not OR) and NAND (Not AND)

However, for designing and analysing a logical circuit, it is convenient to use AND, NOT and OR operators because AND and OR obey many laws as of multiplication and addition in the ordinary algebra of real numbers.

Law 3: Identity, Commutative, Distributive and Associative

The basic logical identities used in Boolean algebra are:

Commutative Law

A.B = B.A A + B = B + A

Distributive Law

A.(B+C) = (A.B) = (A+B).(A+C) A+(B.C) = (A+B).(A+C)

Identity Law

1.A = A 0+ A = A

Inverse Law

A.Ā = A A+ Ā =1

Associative Law

A.(B.C) = (A.B).C A + (B+C) = (A+B) + C

Demorgan's Theorem

$A.B = \bar{A}+\bar{B}$ $\quad \overline{A+B} = \bar{A},\bar{B}$
$0.A = 0$ $\quad 1 + A = 1$
$A,A = A$ $\quad A + A = A$

Note that DeMorgan's law is very useful in simplifying logical expressions.

SAQ 3

Boolean Function

A Boolean function is defined as an algebraic expression formed with the binary variables, the logic operation symbols, parenthesis, and equal to sign. For example,

F = A.B+C is a Boolean function.

The value of Boolean function F can be either 0 or 1.

A Boolean function can be broken into a logic diagram and vice versa (we will discuss this in the next section). Therefore, if we code the logic operations in Boolean algebraic form and simplify this expression, we will design the simplified form of the logic circuits.

Logic Gates

You should be aware that a logic gate is an idealized or physical device implementing a Boolean function, that is, it performs a logical operation on one or more logic inputs and produces a single logic output. Depending on the context, the term may refer to an ideal logic gate, one that has, for instance, zero rise time and unlimited fan-out, or it may refer to a non-ideal physical device. Digital systems are said to be constructed by using logic gates. A logic gate is an electronic circuit that produces a typical output signal depending on its input signal. The output signal of a gate is a simple Boolean operation of its input signal(s). Gates are the basic logic elements. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. Any Boolean function can be represented in the form of gates.

The basic operations are described below with the aid of truth tables.

AND Gate

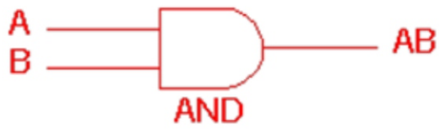


Figure 1: AND Gate

2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2: Truth Table

The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation, i.e., A.B. you should also bear in mind that this dot is sometimes omitted, i.e., AB

OR Gate

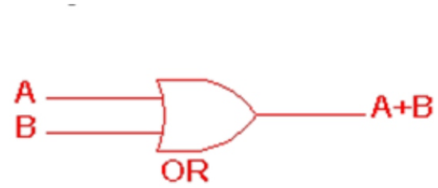


Figure 3: OR Gate

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Figure 4: OR Table

The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation.

SAQ 2

NOT Gate

Figure 5: NOT Gate

NOT gate	
A	$\bar{A}$
0	1
1	0

Figure 6: NOT Table

I want to tell you that the NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs. The diagrams below show two ways in which the NAND logic gate can be configured to produce a NOT gate.

Figure 7: NAND Gate

Figure 8: Another NAND Gate

3.2.2.4 NAND Gate

Figure 9: Another NAND Gate

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

Figure 10: NAND Table

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents an inversion.

3.2.2.5 NOR Gate

Figure 11: NOR Gate

2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Figure 12: NOR Truth Table

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents an inversion.

3.2.2.6 EXOR Gate

Figure 13: EXOR Gate

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 14: EXOR Table

The 'Exclusive-OR' gate is a circuit that will give a high output if either, but not both, of its two inputs, are high. An encircled plus sign ( ) is used to show the EOR operation.

3.2.2.7 EXNOR Gate

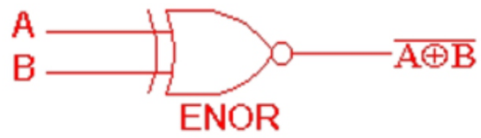


Figure 15: Exclusive NOR Gate

2 Input EXNOR gate		
A	B	$\overline{A\oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Figure 16: Exclusive NOR Table

The 'Exclusive-NOR' gate circuit does the opposite of the EOR gate. It gives a low output if either, but not both of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents an inversion. The NAND and NOR gates are called universal functions since with either one, the AND and OR functions and NOT can be generated.

Note: A function in sum of products form can be implemented using NAND gates by replacing all AND and OR gates by NAND gates. A function in the product of sums form can be implemented using NOR gates by replacing all AND and OR gates by NOR gates.

Table 3.2: Logic gate symbols

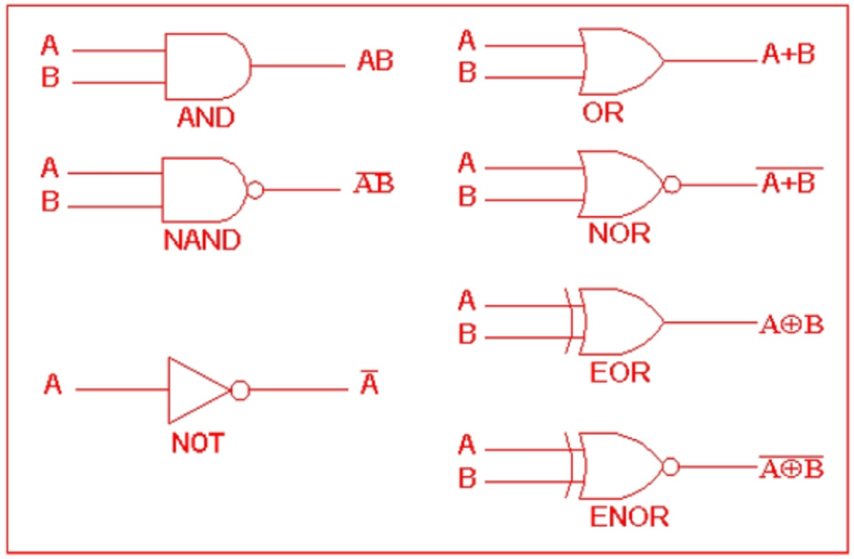


Figure 17: A set of different logic Gates

Table.3 is a summary truth table of the input/output combinations for the NOT gate together with all possible input/output combinations for the other gate functions. Also, note that a truth table with 'n' inputs has 2n rows. You can compare the outputs of different gates.

Table 3: Logic gates representation using the Truth table

INPUTS		OUTPUTS					
A	B	AND	NAND	OR	NOR	EXOR	EXNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

NOT Gate	
A	$\bar{A}$
0	1
1	0

The truth table of NAND and NOR can be made from NOT (A AND B) and NOT (A OR B), respectively. Exclusive OR (XOR) is a special gate whose output is one only if the inputs are not equal. The inverse of exclusive OR can be a comparator, which will produce one output if two inputs are equal.

You should note that the digital circuit use only one or two types of gates for simplicity in fabrication purposes. Therefore, one must think in terms of functionally complete sets of gates. Do you know what a functionally complete set imply? A set of gates by which any Boolean function can be implemented is called a functionally complete set. The functionally complete sets are: (AND, NOT), (NOR), (NAND), (OR< NOT) etc.

Von Neumann Architecture

I want you to know that Von Neumann Architecture is the traditional computer architecture that has shaped the computer machine over time. A typical von Neumann machine is designed based on the architecture shown in figure 18.

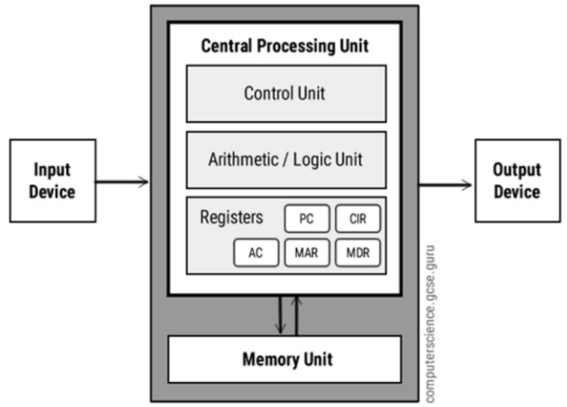


Figure 18: Von Neumann Architecture

Figure 18: Von Neumann Architecture

I want you to also note that a computer system or allied devices operate based on the mechanism described in the Von Neumann architecture. The architecture describes a design architecture for an electronic computer. The architecture assumes that every electronic computer has the following components: the input unit, the Arithmetic and Logical Unit, the Control Unit, processor registers, the Memory part (Primary/Secondary) and the Output Unit. The memory unit stores data and instructions.

Activity 1: Fundamentals of Digital Logic	
Task	**Question**
The commissioner of Education in your state request how algebra can be aaplied in digital logic design.	
	Discuss the correlation between Boolean operations to equivalent truth tables and logic networks?
	Explain the basic operation in Boolean algebra?
	Draw a truth table and logic gate for this expression A.(B.C)=A.(B+C)?

Summary

- Boolean algebra (or Boolean logic) is a logical calculus of truth values, developed by George Boole in the 1840s.
- The laws of Boolean algebra can be defined axiomatically as certain equations called axioms together with their logical consequences called theorems.
- A variable in Boolean algebra can take only two values
- A Boolean function is defined as an algebraic expression formed with the binary variables, the logic operation symbols, parenthesis, and equal to sign.
- We also discussed that logic gate is an idealized or physical device implementing a Boolean function, that is, it performs a logical operation on one or more logic inputs and produces a single logic output.
- The structure of a typical machine that follows Von Neumann architecture has equally been described.
- It has been pointed out that the Von Neumann architecture serves as the backbone for the design of modern electronic computer systems and allied devices.

Self-Assessment Questions

1. What is a Boolean function?
2. Write a short note on the operational principle of NOT gate
3. What can you use to represent the Boolean function of 4 to 6 variables?

Tutor Marked Assessment

- Draw a truth table and logic gate for this expression A.(B.C)=A.(B+C)
- Differentiate between truth table and a logic gate.
- With the aid of a well labelled diagram describe Von Neumann Architecture

Further Reading

- Baron, Robert J. and Higbie Lee. Computer Architecture. Addison-Wesley Publishing Company.
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html
- http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

References

- Baron, Robert J. and Higbie Lee. Computer Architecture. Addison-Wesley Publishing Company.
- Daniel P, and David G, (2009). A Practical Introduction to Computer Architecture.
- Text in Computer Science, Springer Dordrecht Heidelberg London New York.

03 | **Picture:** Technologies

Photo: Pixels.com

UNIT 2

Storage Systems and their Technologies

Introduction

A memory is just like the human brain. It is used to store data and instructions. Computer memory is the storage space in the computer, where data is to be processed and instructions required for processing are stored. The memory is divided into a large number of small parts called cells. Each location or cell has a unique address, which varies from zero to memory size minus one. For example, if the computer has 64k words, then this memory unit has 64 x 1024 = 65536 memory locations. The address of these locations varies from 0 to 65535. Memory is the most important element of a computing system because, without it, the computer can't perform simple tasks. Computer memory is of two basic types; primary memory / Volatile memory and Secondary memory / non-volatile memory. Random Access Memory (RAM) is a volatile memory, while a Read-Only Memory (ROM) is a non-volatile memory.

You will also learn that the most common unit of storage is the byte, equal to 8 bits. A piece of information can be handled by any computer or device whose storage space is large enough to accommodate the binary representation of the piece of information, or simply data.

Learning Outcomes

At the end of this unit, you should be able to:

1. State at least five (5) characteristics of Memory systems.
2. explain functional units of memory systems
3. explain the various components of memory systems
4. identify the elements of modern instructions sets

Main Content

Memory Systems

4 mins

SAQ 1-4

Computer Data Storage

To begin with, Computer data storage, often called storage or memory, is a technology consisting of computer components and recording media used to retain digital data. It is a core function and fundamental component of computers. In contemporary usage, memory is usually semiconductor storage read-write random-access memory, typically DRAM (Dynamic-RAM) or other forms of fast but temporary storage. You should be aware that storage consists of storage devices and their media not directly accessible by the CPU, (secondary or tertiary storage), typically hard disk drives, optical disc drives, and other devices slower than RAM but are non-volatile (retaining contents when powered down). Historically, memory has been called the core, main memory, real storage, or internal memory, while storage devices have been referred to as secondary storage, external memory, or auxiliary/peripheral storage.

Furthermore, the distinctions are fundamental to the architecture of computers. The distinctions also reflect an important and significant technical difference between memory and mass storage devices, which has been blurred by the historical usage of the term storage. Nevertheless, this article uses the traditional nomenclature.

Fundamental Storage Technologies

As of 2011[update], the most commonly used data storage technologies we have are semiconductor, magnetic, and optical, while paper still sees some limited usage. Media is a common name for what actually holds the data in the storage device. Some other fundamental storage technologies have also been used in the past or are proposed for development.

Semiconductor

I want you to know that a semiconductor memory uses semiconductor-based integrated circuits to store information. A semiconductor memory chip may contain millions of tiny transistors or capacitors. Both volatile and non-volatile forms of semiconductor memory exist. In modern computers, primary storage almost exclusively consists of a dynamic volatile semiconductor memory or dynamic random access memory. Since the turn of the century, a type of non-volatile semiconductor memory known as flash memory has steadily gained share as off-line storage for home computers.you should aloso be aware that Non-volatile semiconductor memory is also used for secondary storage in various advanced electronic devices and specialized computers. As early as 2006, notebook and desktop computer manufacturers started using flash-based solid-state drives (SSDs) as default configuration options for the secondary storage either in addition to or instead of the more traditional HDD.

Magnetic Storage

Magnetic storage uses different patterns of magnetization on a magnetically coated surface to store information. Magnetic storage is non-volatile. The information is accessed using one or more read/write heads, which may contain one or more recording transducers. Note that a read/write head only covers a part of the surface so that the head or medium or both must be moved relative to another in order to access data. In modern computers, magnetic storage will take these forms:

Magnetic disk

- Floppy disk, used for off-line storage
- Hard disk drive, used for secondary storage

Magnetic tape

Used for tertiary and off-line storage

In early computers, magnetic storage was also used as:

(i) Primary storage in the form of magnetic memory, or core memory, core rope memory, thin-film memory and/or twistor memory.

(ii) Tertiary (e.g., NCR CRAM) or off-line storage in the form of magnetic cards.

(iii) Magnetic tape was then often used for secondary storage.

Optical Storage

Optical storage, the typical optical disc, stores information in deformities on the surface of a circular disc and reads this information by illuminating the surface with a laser diode and observing the reflection. Optical disc storage is non-volatile. The deformities may be permanent (read-only media), formed once (write once media) or reversible (recordable or read/write media). The following forms are currently in common use:

(I) CD , CD-ROM, DVD, BD-ROM: Read-only storage, used for mass distribution of digital information (music, video, computer programs)

(ii) CD-R, DVD-R, DVD+R, BD-R: Write once storage, used for tertiary and off-line storage

(iii) CD-RW, DVD-RW, DVD+RW, DVD-RAM, BD-RE: Slow write, fast read storage, used for tertiary and off-line storage

(iv) Ultra Density Optical or UDO is similar in capacity to BD-R or BD-RE and is slow write, fast read storage used for tertiary and off-line storage.

Do you know that Magneto-optical disc storage is optical disc storage where the magnetic state on a ferromagnetic surface stores information? The information is read optically and written by combining magnetic and optical methods. Magneto-optical disc storage is non-volatile, sequential access, slow write, fast read storage used for tertiary and off-line storage. 3D optical data storage has also been proposed.

Activity 1: Technologies for Storage Systems	
Task	**Question**
Just like the human brain which controls the whole body, the memory is the technological system brain.	
	Discuss the characteristics of the memory systems?
	Explain the fundamental storage technologies?
	Differentiate between a semiconductor and the magnetic storage?

Related Technologies
Network connectivity

Are you aware that a secondary or tertiary storage may connect to a computer utilizing computer networks? This concept does not pertain to the primary storage, which is shared between multiple processors in a much lesser degree.

(I) Direct-attached storage (DAS) is traditional mass storage, which does not use any network. This is still the most popular approach. This retronym was coined recently, together with NAS and SAN.

(ii) Network-attached storage (NAS) is mass storage attached to a computer that another computer can access at file level over a local area network, a private wide area network, or in the case of online file storage, over the Internet. NAS is commonly associated with the NFS and CIFS/SMB protocols.

(iii) Storage area network (SAN) is a specialized network that provides other computers with storage capacity. The crucial difference between NAS and SAN is the former presents and manages file systems to client computers, whilst the latter provides access at block-addressing (raw) level, leaving it to attaching systems to manage data or file systems within the provided capacity. SAN is commonly associated with Fibre Channel networks.

Summary

- Computer data storage, often called storage or memory, is a technology consisting of computer components and recording media used to retain digital data.
- The most commonly used data storage technologies are semiconductor, magnetic, and optical.
- Semiconductor memory uses semiconductor-based integrated circuits to store information.
- Magnetic storage uses different patterns of magnetization on a magnetically coated surface to store information.
- The typical optical disc stores information in deformities on the surface of a circular disc.
- A secondary or tertiary storage may connect to a computer utilizing computer networks

Self-Assessment Questions

1. State at least five (5) characteristics of Memory systems.
2. Mention five (5) functional units of memory systems
3. Explain the various components of memory systems
4. Identify three (3) elements of modern instructions sets

Tutor Marked Assessment

- Describe 4 common uses of optical storage.
- Differentiate between a semiconductor and magnetic storage.
- What are the three (3) elements of modern instruction sets?

Further Reading

- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/ http://ece-www.colorado.edu /faculty/ heuring.html
- http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/
- http://www.adaptec.com

References

- Mano, M. Morris (1993). Computer System Architecture (3rd ed). Prentice Hall of India.
- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/ http://ece-www.colorado.edu /faculty/ heuring.html

Unit 2 | Module 1 | CSC 230 Storage Systems and their Technologies

04 | **Picture:** Number system

Photo: Wikipedia.com

UNIT 3

Data Representation and Number Systems

Introduction

The study of Number systems and Data Representation is very important in Computer Science as a course. This section focuses on both the Number Systems and Data Representation used in computing devices. A number system can be represented in different forms. However, in Computer science, the emphasis is always much on four types of Number systems. These include:

I. Binary (Base Two)
ii. Octal (Base Eight)
iii. Decimal (Base Ten) and
iv. Hexadecimal (Base Sixteen)

While the Binary number system is the fundamental number system used in computers. Other number systems can be converted to and from the binary number system.

Learning Outcomes

At the end of this unit, you should be able to:

1. master the basic concepts in data representation and number systems
2. explain how to handle the conversion of data from one base to another

Learning Outcomes

3 explain the various ways in which data conversion takes place for computer operation

Main Content

Data Representation

5 mins

Let me tell you that a computer uses a fixed number of bits to represent a piece of data, which could be a number, a character, or others. A n-bit storage location can represent up to 2^n distinct entities. For example, a 3-bit memory location can hold one of these eight binary patterns: 000, 001, 010, 011, 100, 101, 110, or 111. Hence, it can represent at most 8 distinct entities. You could use them to represent numbers 0 to 7, numbers 8881 to 8888, characters 'A' to 'H,' or up to 8 kinds of fruits like apple, orange, banana, or up to 8 kinds of animals like lion, tiger, etc.

Bear in mind that for the purpose of representing data, computer scientists do focus on using number systems such as Base two (binary), octal (system of representing numbers in base 8) and hexadecimal (base sixteen). You will recall that the standard number base used by human beings is base ten which is popularly called decimal. In any situation, computer systems need to convert from other bases to binary. This is because the default data representation format used in computer systems is binary (strings of zeroes and ones).

Number Systems

May I let you know that computers fundamentally use binary (base 2) number system, as they are made from binary digital components (known as transistors) operating in two states - on and off. In computing, we also use hexadecimal (base 16) or octal (base 8) number systems, as a compact form for representing binary numbers.

Decimal (Base 10) Number System

Are you aware that the decimal number system has ten symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9,

called digits? It uses positional notation. That is, the least-significant digit (rightmost digit) is of the order of 10^0 (units or ones), the second rightmost digit is of the order of 10^1 (tens), the third right-most digit is of the order of 10^2 (hundreds), and so on, where ^ denotes exponent. For example,

735 = 700 + 30 + 5 = 7×10^2 + 3×10^1 + 5×10^0

We shall denote a decimal number with an optional suffix D if ambiguity arises.

SAQ 1-4

Binary (Base 2) Number System

You should be aware that this is the Machine code. It is the basic number system used by computer systems for data representation. The binary number system has two symbols: 0 and 1, called binary digits (bits). It is also a positional notation. Examples include:

(I) 110110_2
(ii) 10111_2
(iii) 1011110_2
(iv) 1101100_2

May I let you know that a binary digit is called a bit. Eight bits are called a byte. That is, a byte is defined as the 8 consecutive sets of bits.

Conversions can be made to and from binary number system to other bases. This is demonstrated in this write-up.

SAQ 1-2

Octal System

An Octal system is the system of representing numbers in base eight. It makes use of the digits 0,1,2,3,4,5,6, and 7. Binary is also easily converted to the octal numeral system since octal uses a radix of 8, which is a power of two (namely, 23, so it takes exactly three binary digits to represent an octal digit). The correspondence between octal and binary numerals is the same as for the first eight digits of hexadecimal in the table above. Binary 000 is equivalent to the octal digit 0, and binary 111 is equivalent to octal 7, and so forth.

Don't forget that converting from octal to binary proceeds in the same fashion as it does for hexadecimal:

65(8)=110 1012
17(8)=001 1112

Decimal Number System

Decimal Number System is the system of representing numbers in base ten. It makes use of the following digits for the representation.: 0,1,2,3,4,5,6,7,8 and 9. To convert a decimal number to binary, it is very straight forward.

For instance,let us convert 29 to binary number

2	29	
2	14	Rem 1
2	7	Rem 0
2	3	Rem 1
2	1	Rem 1
	0	Rem 1

Therefore the binary equivalent of 29_{dec} is 11101

At times, you may be required to convert numbers in binary form to decimal. In this case, you make use of positional value and two, being the base.

We can as well convert the binary number back to other bases. Let us demonstrate this with the conversion of binary to decimal scale (that is base ten).

Question: Let us also convert 10111 to base ten.

Solution

Recall that the question can also be written as 10111_2. First, you have to assign the positional value to the given problem as shown below and multiply each digit by the number base raise to power the positional value.

The positional value starts from zero and is assigned each digit starting from the rightmost digit:

$=1X2^4+0X2^3+1X2^2+1X2^1+1X2^0$

=1X16+0X8+1X4+1X2+1X1 (Since anything raise to power zero=1)

=16+0+4+2+1

Answer=23_{10} **That is, 10111_2 has 23_{10} has its equivalent.**

SAQ 1-5

Hexadecimal (Base 16) Number System

Hexadecimal is the system of representing numbers in base sixteen. Hexadecimal number system uses 16 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F, called hex digits. It is a positional notation, for example,

A3EH = A00H + 30H + EH = 10×16^2 + 3×16^1 + 14×16^0

We shall denote a hexadecimal number (in short, hex) with a suffix H. Some programming languages denote hex numbers with prefix 0x or 0X (e.g., 0x1A3C5F), or prefix x with hex digits quoted (e.g., x'C3A4D98B').

Each hexadecimal digit is also called a hex digit. Most programming languages accept lowercase 'a' to 'f' as well as uppercase 'A' to 'F.'

Computers use the binary system in their internal operations, as they are built from binary digital electronic components with 2 states - on and off. However, writing or reading a long sequence of binary bits is cumbersome and error-prone (try to read this binary string: 1011 0011 0100 0011 0001 1101 0001 1000B, which is the same as hexadecimal B343 1D18H). The hexadecimal system is used as a compact form or shorthand for binary bits. Each hex digit is equivalent to 4 binary bits, i.e., shorthand for 4 bits, as follows

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

Integer Representation

Integers are whole numbers or fixed-point numbers with the radix point fixed after the least-significant bit. They are in contrast to real numbers or floating-point numbers, where the position of the radix point varies. It is important to take note that integers and floating-point numbers are treated differently in computers. They have different representations and are processed differently (e.g., floating-point numbers are processed in a so-called floating-point processor).

Note that computers use a fixed number of bits to represent an integer. The commonly-used bit-lengths for integers are 8-bit, 16-bit, 32-bit, or 64-bit. Besides bit-lengths, there are two representation schemes for integers:

1. Unsigned Integers: can represent zero and positive integers.

2. Signed Integers: can represent zero, positive and negative integers. Three representation schemes had been proposed for signed integers:

I. Sign-Magnitude representation
ii. 1's Complement representation
iii. 2's Complement representation

N-bit Unsigned Integers

Unsigned integers can represent zero and positive integers, but not negative integers. The value of an unsigned integer is interpreted as "the magnitude of its underlying binary pattern."

Example 1: Suppose that n=8 and the binary pattern is 0100 0001B, the value of this unsigned integer is 1×2^0 + 1×2^6 = 65D.

Example 2: Suppose that n=16 and the binary pattern is 0001 0000 0000 1000B, the value of this unsigned integer is 1×2^3 + 1×2^12 = 4104D.

Example 3: Suppose that n=16 and the binary pattern is 0000 0000 0000 0000B, the value of this unsigned integer is 0.

An n-bit pattern can represent 2^n distinct integers. An n-bit unsigned integer can represent integers from 0 to (2^n)-1.

Signed Integers

Signed integers can represent zero, positive integers, as well as negative integers. Three representation schemes are available for signed integers:

1. Sign-Magnitude representation
2. 1's Complement representation
3. 2's Complement representation

In all the above three schemes, the most-significant bit (msb) is called the sign bit. The sign bit is used to represent the sign of the integer - with 0 for positive integers and 1 for negative integers. The magnitude of the integer, however, is interpreted differently in different schemes.

N-bit Sign Integers in Sign-Magnitude Representation

In sign-magnitude representation:

- The most-significant bit (msb) is the sign bit, with a value of 0 representing positive integer and 1 representing a negative integer.
- The remaining n-1 bits represent the magnitude (absolute value) of the integer. The absolute value of the integer is interpreted as "the magnitude of the (n-1)-bit binary pattern".

Example 1: Suppose that n=8 and the binary representation is 0 100 0001B.
Sign bit is 0 ⇒ positive
Absolute value is 100 0001B = 65D
Hence, the integer is +65D

Example 2: Suppose that n=8 and the binary representation is 1 000 0001B.
Sign bit is 1 ⇒ negative
Absolute value is 000 0001B = 1D
Hence, the integer is -1D

Summary

In this unit, we have discussed the meaning of data representation and number system whereby we mentioned and discussed extensively the number systems which are:

I. Binary (Base Two)
ii. Octal (Base Eight)
iii. Decimal (Base Ten) and
iv. Hexadecimal (Base Sixteen)

We also discussed and calculated how we can convert from one number base to the other.

Self-Assessment Questions

1. Write a short note on each of the following number systems: (i) Binary (ii) Octal (iii) Hexadecimal
2. Convert 456 to octal scale (That is base 8).
3. Convert the following decimal numbers into binary and hexadecimal numbers:
    1. 108
    2. 4848
    3. 9000
4. Convert the following binary numbers into hexadecimal and decimal numbers:
    1. 1000011000
    2. 10000000
    3. 101010101010
5. Convert the following hexadecimal numbers into binary and decimal numbers:
    1. ABCDE
    2. 1234
    3. 80F
6. Convert the following decimal numbers into binary equivalent:
    1. 19.25D
    2. 123.456D

Tutor Marked Assessment

- Write short notes on each of the following:

  (a) Binary
  (b) Octal and
  (c) Hexadecimal number systems

- Itemise the steps that have to be followed while converting a number in decimal scale to binary
- Convert the following binary numbers into hexadecimal and decimal numbers:
  a. 1000011000
  b. 10000000
  c. 101011

  iv Write a short note on n-bit signed integer numbers.

References

- Computer Architecture (2006).
- Computer Architecture (2006).
- Computer Architecture (2006).

05 | **Picture:** Memory hierarchy

Photo: Wikipedia.com

UNIT 4

Memory hierarchy, organization and operations

Introduction

In computer architecture, the memory hierarchy separates computer storage into a hierarchy based on response time. Since response time, complexity, and capacity are related, the levels may also be distinguished by their performance and controlling technologies. RAM (Random Access Memory) is an internal memory device that temporarily holds data and instructions while processing is happening. If the CPU is the "brain" of the computer, then RAM is the "working memory" or "thinking memory" used to store data just for the programmes and applications being used at that time.

Learning Outcomes

At the end of this unit, you should be able to:

1. define memory hierarchy and its characteristics
2. describe various types of memories
3. explain the characteristics of an interconnection structure
4. explain random access

SAQ 1

Memory Hierarchy

4 mins

May I interest you to know that a typical memory hierarchy starts with a small, expensive, and relatively fast unit, called the cache? followed by a larger, less expensive, and relatively slow main memory unit. Cache and main memory are built using solid-state semiconductor material (typically CMOS transistors). It is customary to call the fast memory level the primary memory. The solid-state memory is followed by larger, less expensive, and far slower magnetic memories that consist typically of the (hard) disk and the tape. It is customary to call the disk the secondary memory, while the tape is conventionally called the tertiary memory. The objective behind designing a memory hierarchy is to have a memory system that performs as if it consists entirely of the fastest unit and whose cost is dominated by the cost of the slowest unit

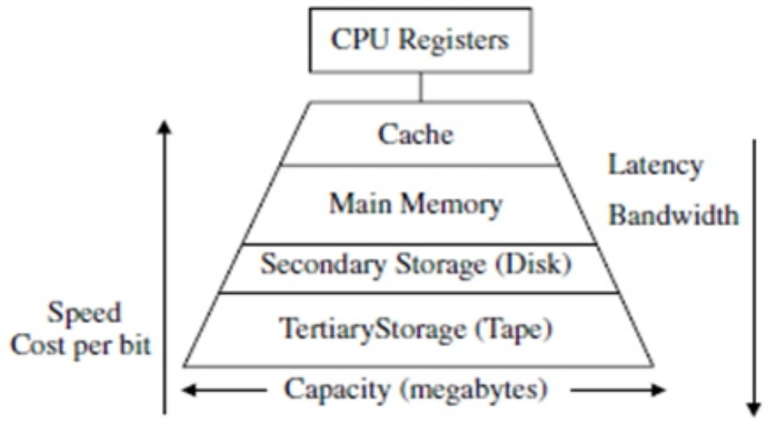


Figure 19: Typical memory hierarchy

Thus, a memory system can be considered to consist of three groups of memories. These are:

SAQ 2

Internal Processor Memories

These consist of the small set of high-speed registers which are internal to a processor and are used as temporary locations where actual processing is done.

Primary Memory or Main Memory

It is a large memory which is fast but not as fast as internal processor memory. This memory is accessed directly by the processor. It is mainly based on integrated circuits (IC).

Secondary Memory/Auxiliary Memory/Backing Store:

Auxiliary memory is, in fact, much larger in size than the main memory but is slower than the main memory. It usually stores system programs (programs which are used by the system to perform various operational functions), other instructions, programs, and data files. Secondary memory can also be used as an overflow memory in case the main memory capacity has been exceeded. Secondary memories cannot be accessed directly by a processor.

First, the information of these memories is transferred to the main memory, and then the information can be accessed as the information of the main memory. There is another kind of memory which is increasingly being used in modern computers. It is called the Cache memory. It is logically positioned between the internal memory (registers) and the main memory. It stores or catches some of the content of the main memory, which is currently in use of the processor. We will discuss this memory in greater detail in a subsequent section of this unit.

SAQ 3

Characteristics Terms for Various Memory Devices

Try to understand that the memory hierarchy can be characterized by a number of parameters. Among these parameters are the access type, capacity, cycle time, latency, bandwidth, and cost.

The term access: refers to the action that physically takes place during a read or writes operation.

The capacity: of a memory level is usually measured in bytes.

The cycle time: is defined as the time elapsed from the start of a read operation to the start of a subsequent read.

The latency: is defined as the time interval between the request for information and the access to the first bit of that information.

The bandwidth: provides a measure of the number of bits per second that can be accessed.

The cost: of a memory level is usually specified as dollars per megabytes. Figure 1 depicts a typical memory hierarchy. Table 1 provides typical values of the memory hierarchy parameters.

SAQ 4

Random Access

The term random access: refers to the fact that any access to any memory location takes the same fixed amount of time regardless of the actual memory location and/or the sequence of accesses that takes place. For example, if a write operation to memory location 100 takes 15 ns and if this operation is followed by a read operation to memory location 3000, then the latter operation will also take 15 ns. This is to be compared to sequential access in which if access to location 100 takes 500 ns, and if consecutive access to location 101 takes 505 ns, then it is expected that access to location 300 may take 1500 ns. This is because the memory has to cycle through locations 100 to 300, with each location requiring 5 ns.

The effectiveness of a memory hierarchy depends on the principle of moving information into the fast memory infrequently and accessing it many times before replacing it with new information. This principle is possible due to a phenomenon called locality of reference; that is, within a given period, programs tend to reference a relatively confined area of memory repeatedly. There exist two forms of locality: spatial and temporal locality.

Table 4: Memory Hierarchy

	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64–1024 bytes	1–10 ns	System clock rate	High
Cache memory	Random	8–512 KB	15–20 ns	10–20 MB/s	$500
Main memory	Random	16–512 MB	30–50 ns	1–2 MB/s	$20–50
Disk memory	Direct	1–20 GB	10–30 ms	1–2 MB/s	$0.25
Tape memory	Sequential	1–20 TB	30–10,000 ms	1–2 MB/s	$0.025

Activity 1: Operations of the Memory	
Task	**Question**
Visit any firm who has a good computer architectural system.	
	Discuss the kind of memory system(s) which they employ in their computer architecture?
	Describe the characteristics of Various Memory Devices?
	Explain the characteristics of an interconnection structure?

Memory Interleaving

One possible technique that we can use to increase the bandwidth is memory interleaving. To achieve the best results, we can assume that the block brought from the main memory to the cache, upon a cache miss, consists of elements that are stored in different memory modules, that is, whereby consecutive memory addresses are stored in successive memory modules. Table 1 illustrates the simple case of a main memory consisting of eight memory modules. It is assumed in this case that the block consists of 8 bytes. Having introduced the basic idea leading to the use of a cache memory, we would like to assess the impact of temporal and spatial locality on the performance of the memory hierarchy. In order to make such an assessment, we will limit our deliberation to the simple case of a hierarchy consisting only of two levels, that is, the cache and the main memory. We assume that the main memory access time is tm, and the cache access time is tc. We will measure the impact of locality in terms of the average access time, defined as the average time required to access an element (a word) requested by the processor in such a two-level hierarchy.

Summary

- A typical memory hierarchy starts with a small, expensive, and a relatively fast unit called the cache.
- It is customary to call the fast memory level the primary memory.
- The objective behind designing a memory hierarchy is to have a memory system that performs as if it consists entirely of the fastest unit and whose cost is dominated by the cost of the slowest unit.

Self-Assessment Questions

SAQ

1. Define memory hierarchy
2. Describe various types of memories
3. Mention five (5) characteristics of an interconnection structure
4. What is random access?

Tutor Marked Assessment

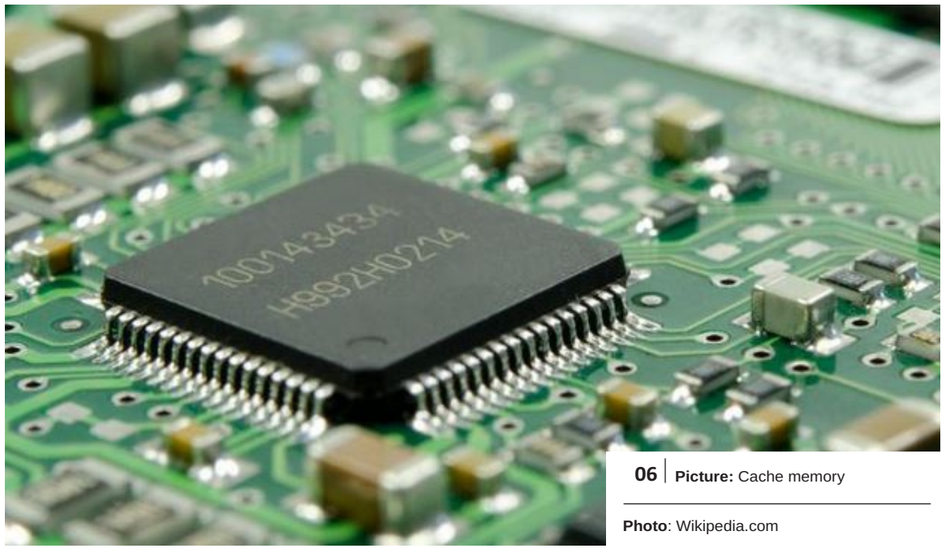
- With the aid of a well labelled diagram, describe memory hierarchy in computer architecture
- Identify any five characteristics of Random Access Memory (Main Memory)

Further Reading

- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html
- http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

References

- Hayes, John P. (1998). Computer Architecture and Organisation (2nd ed). McGraw- Hill International editions.
- Hennessy J.L and Patterson, D.A. Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 2006. ISBN: 0-123-70490-1.
- Premchand, P. Data Communication, and Computer networks. Dept. of CSE, College of Engineering, Osmania University, Hyd 500007.



06 | **Picture:** Cache memory

Photo: Wikipedia.com

UNIT 5

Cache Memory

Introduction

Cache Memory is another important type of memory in computer system design. When a program executes on a computer, most of the memory references are made to a small number of locations. Typically, 90% of the execution time of a program is spent on just 10% of the code. This property is known as the locality principle. When a program references a memory location, it is likely to reference that same memory location again soon, which is known as temporal locality.

Similarly, there is spatial locality, in which a memory location that is near a recently referenced location is more likely to be referenced than a memory location that is farther away. Temporal locality arises because programs spend much of their time in an iteration or recursion, and thus the same section of code is visited a disproportionately large number of times. Spatial locality arises because data tends to be stored in contiguous locations.

Learning Outcomes

At the end of this unit, you should be able to:

1. Explain the concepts of hierarchical memory organization.
2. Describe how each level of memory contributes to system performance and how the performance is measured.
3. Explain the concepts behind cache memory

SAQ 1

Cache memory

10 mins

You should be aware that a small but fast cache memory, in which the contents of the most commonly accessed locations are maintained, can be placed between the main memory and the CPU. When a program executes, the cache memory is searched first, and the referenced word is accessed in the cache if the word is present. If the referenced word is not in the cache, then a free location is created in the cache, and the referenced word is brought into the cache from the main memory.

Note that memory access is generally slow when compared with the speed of the central processing unit (CPU), and so the memory poses a significant bottleneck in computer performance. Since most memory references come from a small set of locations, the locality principle can be exploited in order to improve performance.

Also note that the word is then accessed in the cache. Although this process takes longer than accessing the main memory directly, the overall performance can be improved if a high proportion of memory accesses are satisfied by the cache. Modern memory systems may have several levels of cache, referred to as Level 1 (L1), Level 2 (L2), and even, in some cases, Level 3 (L3). In most instances theL1 cache is implemented right on the CPU chip. Both the Intel Pentium and the IBM-Motorola PowerPC G3 processors have 32 Kbytes of L1 cache on the CPU chip.

May I let you know that cache memory is faster than the main memory for some reasons. Faster electronics can be used, which also results in a greater expense in terms of money, size, and power requirements. Since the cache is small, this increase in cost is relatively small. Cache memory has fewer locations than the main memory, and as a result, it has a shallow decoding tree, which reduces the access time.

The cache is placed both physically closer and logically closer to the CPU than the main memory, and this placement avoids communication delays over a shared bus. A typical situation is shown in Figure 3.1. A simple computer without a cache memory is shown on the left side of the figure.

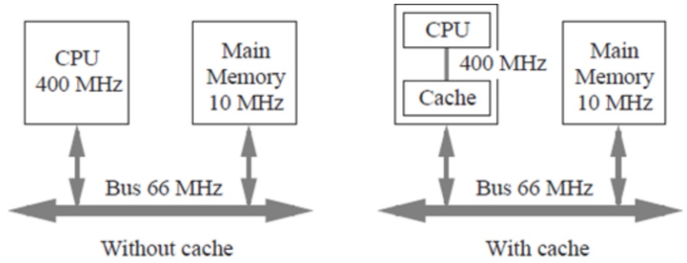


Figure 20: Placement of Cache in a Computer System

This cache-less computer contains a CPU that has a clock speed of 400 MHz but communicates over a 66 MHz bus to the main memory that supports a lower clock speed of 10 MHz. A few bus cycles are usually needed to synchronize the CPU with the bus, and thus the difference in speed between main memory and the CPU can be as large as a factor of ten or more. Cache memory can be positioned closer to the CPU, as shown in the right side of Figure 2 so that the CPU sees fast accesses over a 400 MHz direct path to the cache.

SAQ 2,3

Replacement Policies in Associative Mapped Caches

Be notidied that when a new block needs to be placed in an associative mapped cache, an available slot must be identified. If there are unused slots, such as when a program begins execution, then the first slot with a valid bit of 0 can simply be used.

When all of the valid bits for all cache slots are 1, however, then one of the active slots must be freed for the new block. Four replacement policies that are commonly used are: least recently used (LRU), first-in-first-out (FIFO), least frequently used (LFU), and random. A fifth policy that is used for analysis purposes only is optimal.

For the LRU policy, a timestamp is added to each slot, which is updated when any slot is accessed. When a slot must be freed for a new block, the contents of the least recently used slot, as identified by the age of the corresponding timestamp, are discarded, and the new block is written to that slot. The LFU policy works similarly, except that only one slot is updated at a time by incrementing a frequency counter that is attached to each slot. When a slot is needed for a new block, the least frequently used slot is freed.

The FIFO policy replaces slots in a round-robin fashion, one after the next in the order of their physical locations in the cache. The random replacement policy simply chooses a slot at random. The optimal replacement policy is not practical but is used for comparison purposes of determining how effective other replacement policies are to the best possible. That is, the optimal replacement policy is determined only after a program has already executed, and so it is of little help to a running program. Studies have shown that the LFU policy is only slightly better than the random policy. The LRU policy can be implemented efficiently and is sometimes preferred over the others for that reason.

Advantages and Disadvantages of the Associative Mapped Cache

The associative mapped cache has the advantage that any main memory block can be placed into any cache slot. This means that regardless of how irregular the data and program references are, if a slot is available for the block, it can be stored in the cache. This results in considerable hardware overhead needed for cache bookkeeping. Each slot must have a 27-bit tag that identifies its location in the main memory, and each tag must be searched in parallel. This means that in the example above, the tag memory must be 27 x 2^{14} bits in size, and as described above, there must be a mechanism for searching the tag memory in parallel. Memories that can be searched for their contents, in parallel, are referred to as associative, or content-addressable memories. By restricting where each main memory block can be placed in the cache, we can eliminate the need for associative memory. This kind of cache is referred to as a direct-mapped cache, which is discussed in the next section.

Direct Mapped Cache

Figure 3.2 shows a direct mapping scheme for a 2^{32} word memory. As before, the memory is divided into 2^{27} blocks of $2^5 = 32$ words per block, and the cache consists of 2^{14} slots. There are more main memory blocks than there are cache slots, and a total of $2^{27}/2^{14} = 2^{13}$ main memory blocks can be mapped onto each cache slot. In order to keep track of which of the 2^{13} possible blocks is in each slot, a 13-bit tag field is added to each slot which holds an identifier in the range from 0 to $2^{13} - 1$.

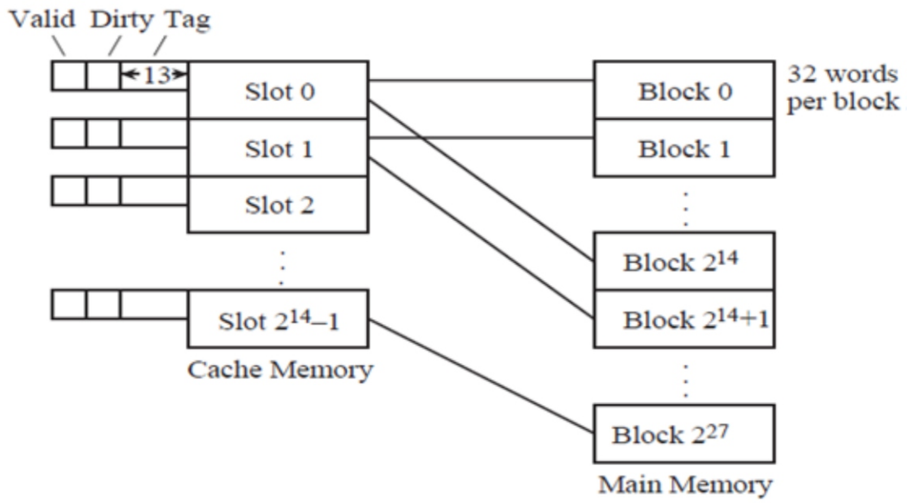


Figure 21: A Direct Mapping Scheme for Cache Memory

This scheme is called "direct mapping" because each cache slot corresponds to an explicit set of main memory blocks. For a direct mapped cache, each main memory block can be mapped to only one slot, but each slot can receive more than one block. The mapping from main memory blocks to cache slots is performed by partitioning an address into fields for the tag, the slot, and the word as shown below:

The 32-bit main memory address is partitioned into a 13-bit tag field, followed by a 14-bit slot field, followed by a five-bit word field. When a reference is made to a main memory address, the slot field identifies in which of the 214 slots the block will be found if it is in the cache. If the valid bit is 1, then the tag field of the referenced address is compared with the tag field of the slot. If the tag fields are the same, then the word is taken from the position in the slot specified by the word field. If the valid bit is 1, but the tag fields are not the same, then the slot is written back to main memory if the dirty bit is set, and the corresponding main memory block is then read into the slot. For a program that has just started execution, the valid bit will be 0, and so the block is simply written to the slot. The valid bit for the block is then set to 1, and the program resumes execution.

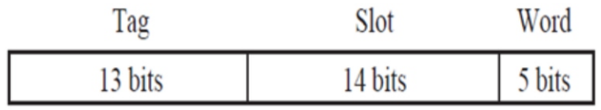


Figure 22: Cache Memory Mapping Scheme

Advantages and Disadvantages of the Direct Mapped Cache

The direct mapped cache is a relatively simple scheme to implement. The tag memory in the example above is only 13 x 2^{14} bits in size, less than half of the associative mapped cache. Furthermore, there is no need for an associative search, since the slot field of the main memory address from the CPU is used to "direct" the comparison to the single slot where the block will be if it is indeed in the cache.

It should not be forgotten that this simplicity comes at a cost. Consider what happens when a program references a location that is 2^{19} words apart, which is the size of the cache. This pattern can arise naturally if a matrix is stored in memory by rows and is accessed by columns. Every memory reference will result in a miss, which will cause an entire block to be read into the cache even though only a single word is used.

Worse still, only a small fraction of the available cache memory will actually be used. Now it may seem that any programmer who writes a program this way deserves the resulting poor performance. However, fast matrix calculations use power-of-two dimensions (which allows shift operations to replace costly multiplications and divisions for array indexing). So the worst-case scenario of accessing memory locations that are 2^{19} addresses apart is not all that unlikely. To avoid this situation, without paying the high implementation price of fully associative cache memory, the set associative mapping which combines aspects of both direct mapping and associative mapping scheme can be used.

Cache Performance

Notice that we can readily replace the cache direct mapping hardware with associative or set associative mapping hardware, without making any other changes to the computer or the software. Only the runtime performance will change between methods. Runtime performance is the purpose behind using cache memory, and there are a number of issues that need to be addressed as to what triggers a word or block to be moved between the cache and the main memory.

Be aware that cache read and write policies are summarized in Figure 22. The policies depend upon whether or not the requested word is in the cache. If a cache read operation is taking place, and the referenced data is in the cache, then there is a "cache hit," and the referenced data is immediately forwarded to the CPU. When a cache miss occurs, then the entire block that contains the referenced word is read into the cache.

In some cache organizations, the word that causes the miss is immediately forwarded to the CPU as soon as it is read into the cache, rather than waiting for the remainder of the cache slot to be filled, which is known as a load-through operation. For a non-interleaved main memory, if the word occurs in the last position of the block, then no performance gain is realized since the entire slot is brought in before load-through can take place. For an interleaved main memory, the order of accesses can be organized so that a load-through operation will always result in a performance gain.

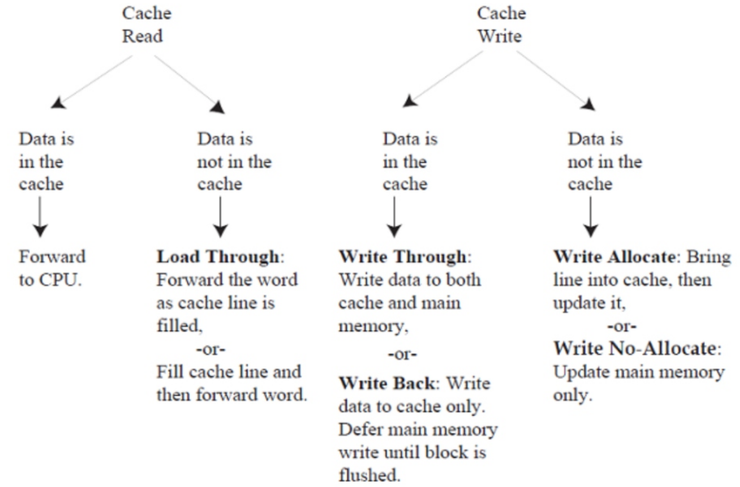


Figure 23: Cache Read and Write Policies

For write operations, if the word is in the cache, then there may be two copies of the word, one in the cache, and one in the main memory. If both are updated simultaneously, this is referred to as write-through. If the write is deferred until the cache line is flushed from the cache, this is referred to as write-back.

Even if the data item is not in the cache when the write occurs, there is the choice of bringing the block containing the word into the cache and then updating it,(known as write-allocate,) or to update it in the main memory without involving the cache, (known as write-no-allocate.) Some computers have separate caches for instructions and data, which is a variation of a configuration known as the Harvard architecture (also known as a split cache), in which instructions and data are stored in separate sections of memory.

Since instruction slots can never be dirty (unless we write self-modifying code, which is rare these days), an instruction cache is simpler than a data cache. In support of this configuration, observations have shown that most of the memory traffic moves away from the main memory rather than towards it.

Statistically, there is only one write to memory for every four read operations from memory. One reason for this is that instructions in an executing program are only read from the main memory, and are never written to the memory except by the system loader. Another reason is that operations on data typically involve reading two operands and storing a single result, which means there are two read operations for every write operation.

A cache that only handles reads, while sending writes directly to the main memory, can thus also be effective, although not necessarily as effective as a fully functional cache. As to which cache read and write policies are best, there is no simple answer. The organization of a cache is optimized for each computer architecture and the mix of programs that the computer executes. Cache organization and cache sizes are usually determined by the results of simulation runs that expose the nature of memory traffic.

Activity 1: Cache Memory	
Task	**Question**
Cache memory, also called CPU memory, is high-speed static random access memory (SRAM).	
	Differentiate between locality principle and temporal locality?
	Explain how the performances of memory levels can be measured?
	Explain the characteristics cache memory and its relevance?

Hit Ratios and Effective Access Times

Bear in mind that two measures that characterize the performance of cache memory are the hit ratio and the effective access time. The hit ratio is computed by dividing the number of times referenced words are found in the cache by the total number of memory references. The effective access time is computed by dividing the total time spent accessing memory (summing the main memory and cache access times) by the total number of memory references. The corresponding equations are given below:

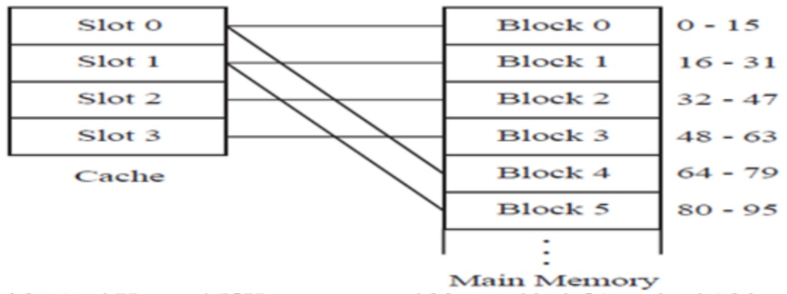
$$Hit\ ratio = \frac{No.\ times\ referenced\ words\ are\ in\ cache}{Total\ number\ of\ memory\ accesses}$$

$$Eff.\ access\ time = \frac{(\#\ hits)(Time\ per\ hit) + (\#\ misses)(Time\ per\ miss)}{Total\ number\ of\ memory\ access}$$

Consider computing the hit ratio and the effective access time for a program running on a computer that has a direct mapped cache with four 16-word slots. The cache access time is 80 ns, and the time for transferring a main memory block to the cache is 2500 ns. Assume that load-through is used in this architecture and that the cache is initially empty.

A sample program executes from memory locations 48 – 95, and then loops 10 times from 15 – 31 before halting. We record the events as the program executes, as shown in Figure 5. Since the memory is initially empty, the first instruction that executes causes a miss. A miss thus occurs at location 48, which causes main memory block #3 to be read into cache slot #3. This first memory access takes 2500 ns to complete.

Load-through is used for this example, and so the word that causes the miss at location 48 is passed directly to the CPU while the rest of the block is loaded into the cache.



1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	80ns×15=1200ns	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	80ns×15=1200ns	
9 hits	15	80ns×9=720ns	Last nine iterations of loop
144 hits	16-31	80ns×144=12,240ns	Last nine iterations of loop
Total hits = 213		Total misses = 5	

Figure 24: A table of the event for a program executing on architecture with a small direct mapped cache memory

$$Hit\ ratio = \frac{213}{218} = 97.7\%$$

$$EffectiveAccessTime = \frac{(213)(80ns) + (5)(2500ns)}{218} = 136ns$$

Although the hit ratio is 97.6%, the effective access time for this example is almost 75% longer than the cache access time. This is due to the large amount of time spent on accessing a block from the main memory.

Summary

- Cache memory, also called CPU memory, is high-speed static random access memory (SRAM) that a computer microprocessor can access more quickly than it can access regular random access memory (RAM).
- A cache memory is faster than and has fewer locations than the main memory.
- A cache is placed both physically closer and logically closer to the CPU than the main memory
- The physical memory is smaller than the size of the program but is larger than any single routine.

- Cache memory, also called CPU memory, is high-speed static random access memory (SRAM) that a computer microprocessor can access more quickly than it can access regular random access memory (RAM).
- A cache memory is faster than and has fewer locations than the main memory.
- A cache is placed both physically closer and logically closer to the CPU than the main memory
- The physical memory is smaller than the size of the program but is larger than any single routine.

Self-Assessment Questions

SAQ

1. Explain the concepts of hierarchical memory organization.
2. Describe how each level of memory contributes to system performance and how the performance is measured.
3. Explain the concepts behind cache memory,

Tutor Marked Assessment

- What is the difference between a locality principle and temporal locality?
- Briefly explain the concepts of hierarchical memory organization.
- Describe how each level of memory contributes to system performance and how the performance is measured.

Further Reading

- Hennessy J.L and Patterson, D.A. Computer Architecture (2006). A Quantitative Approach, Morgan Kaufmann, 2006. ISBN: 0-123-70490-1.
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html
- http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

References

- Hayes, John P. (1998). Computer Architecture and Organisation (2nd ed). McGraw- Hill International editions.
- Hennessy J.L and Patterson, D.A. Computer Architecture (2006). A Quantitative Approach, Morgan Kaufmann, 2006. ISBN: 0-123-70490-1.

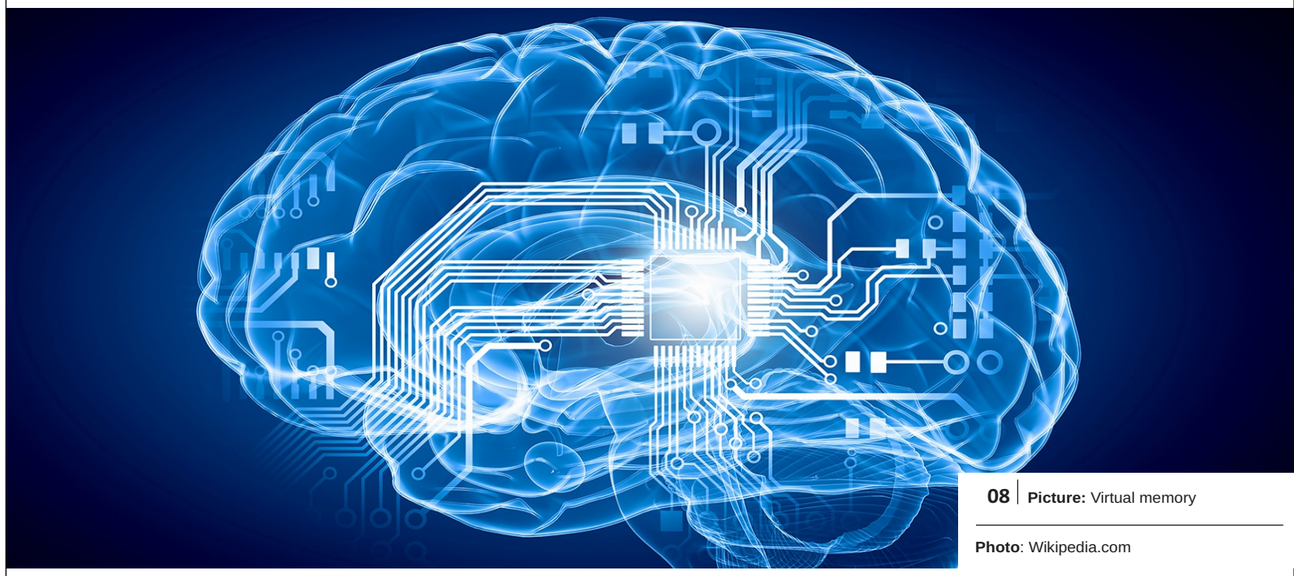
- Premchand, P. Data Communication and Computer Networks. Dept. of CSE, College of Engineering, Osmania University, Hyd 500007.

07 | **Picture:** Virtual memory

Photo: Wikipedia.com

Module 2

VIRTUAL MEMORY, OVERLAYS, PAGING, SEGMENTATION AND FRAGMENTATION



08 | **Picture:** Virtual memory

Photo: Wikipedia.com

UNIT 1

Virtual memory, overlays and paging

Introduction

Memory access is generally slow when compared with the speed of the central processing unit (CPU), and so the memory poses a significant bottleneck in computer performance. Since most memory references come from a small set of locations, the locality principle can be exploited in order to improve performance.

Learning Outcomes

At the end of this unit, you should be able to:

1. explain virtual memory
2. explain overlays
3. explain paging

SAQ 3

Virtual Memory

5 mins

Despite the enormous advancements in creating ever-larger memories in smaller areas, computer memory is still like closet space, in the sense that we can never have enough of it. An economical method of extending the apparent size of the main memory is to augment it with disk space, which is one aspect of virtual memory that we cover in this section.

Disk storage appears near the bottom of the memory hierarchy, with a lower cost per bit than the main memory. It is, therefore, reasonable to use disk storage to hold the portions of a program or data sets that do not entirely fit into the main memory. In a different aspect of the virtual memory, complex address mapping schemes are supported which give greater flexibility on how the memory is used. We explore these aspects of virtual memory below.

SAQ 2

Overlays

An early approach of using disk storage to augment the main memory made use of overlays, in which an executing program overwrites its own code with other code as needed. In this scenario, the programmer has the responsibility of managing memory usage. Figure 6 shows an example in which a program contains the main routine and three subroutines A, B, and C. The physical memory is smaller than the size of the program but is larger than any single routine. A strategy for managing memory using overlays is to modify the program so that it keeps track of which subroutines are in memory, and reads in subroutine code as needed.

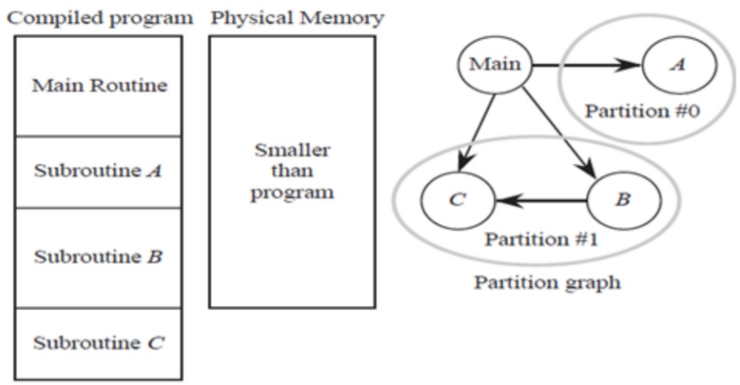


Figure 25: A partition graph for a program with a main routine and three subroutines.

Typically, the main routine serves as the driver and manages the bulk of the bookkeeping. The driver stays in the memory while other routines are brought in and out. Figure 6 shows a partition graph that is created for the example program. The partition graph identifies which routines can overlay others based on which subroutines call others. For this example, the main routine is always present and supervises which subset of subroutines are in memory. Subroutines B and C are kept in the same partition in this example because B calls C, but subroutine A is in its own partition because only the main routine calls A. Partition #0 can thus overlay partition #1, and partition #1 can overlay partition #0. Although this method will work well in a variety of situations, a cleaner solution might be to let an operating system manage the overlays.

It is good to know that when more than one program is loaded into memory, then the routines that manage the overlays cannot operate without interacting with the operating system in order to find out which portions of memory are available. This scenario introduces a great deal of complexity into managing the overlay process since there is a heavy interaction between the operating system and each program. An alternative method that can be managed by the operating system alone is called paging, which is described in the next section.

SAQ 3

Paging

Paging is a form of automatic overlaying that is managed by the operating system. The address space is partitioned into equal-sized blocks, called pages. Pages are usually an integral power of two in size, such as 210 = 1024 bytes. Paging makes the physical memory appear larger than it truly is by mapping the physical memory address space to some portion of the virtual memory address space, which is usually stored on a disk. An illustration of a virtual memory mapping scheme is shown in Figure 7. Eight virtual pages are mapped to four physical page frames.

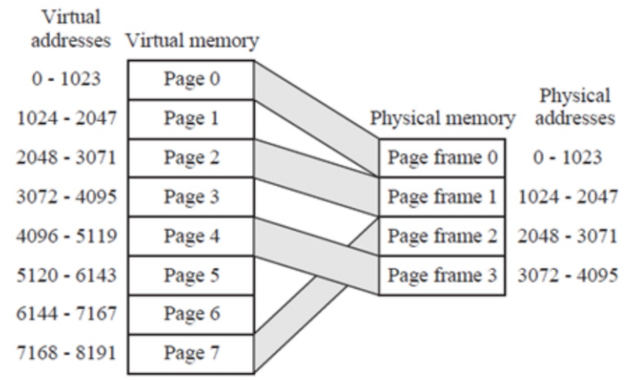


Figure 26: A mapping between a virtual and a physical memory

It is good to note that an implementation of virtual memory must handle references that are made outside of the portion of virtual space that is mapped to physical space. The following sequence of events is typical when a referenced virtual location is not in physical memory, which is referred to as a page fault:'

1. A page frame is identified to be overwritten. The contents of the page frame are written to the secondary memory if changes are made to it so that the changes are recorded before the page frame is overwritten.

2. The virtual page that we want to access is located in the secondary memory and is written into the physical memory in the page frame located in (1) above.

3. The page table (see below) is updated to map the new section of virtual memory onto the physical memory.

4. Execution continues. For the virtual memory shown in Figure 7, there are 213 = 8192 virtual locations. Therefore, an executing program must generate 13-bit addresses, which are interpreted as a 3-bit page number and a 10-bit offset within the page. Given the 3-bit page number, we need to find out where the page is: it is either in one of the four page frames or in the secondary memory. In order to keep track of which pages are in the physical memory, a page table is maintained, as illustrated in Figure 7, which corresponds to the mapping shown in Figure 8.

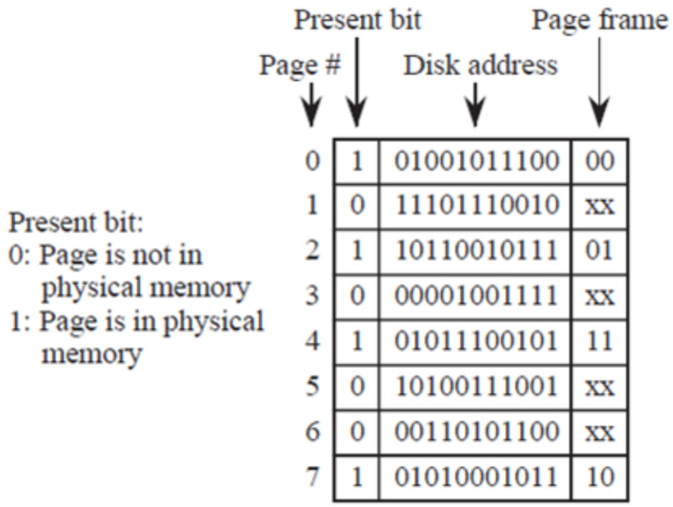


Figure 27: A page table for a virtual memory

The page table has as many entries as possible are virtual pages. The present bit indicates whether or not the corresponding page is in physical memory. The disk address field is a pointer to the location that the corresponding page can be found on a disk unit. The operating system typically manages the disk accesses; therefore, the page table only needs to maintain the disk addresses that the operating system assigns to blocks when the system starts up. The disk addresses typically do not change during computation.

You should note that the page frame field indicates which physical page frame holds a virtual page if the page is in physical memory. For pages that are not in physical memory, the page frame fields are invalid, and so they are marked with "xx" in Figure 3.7. In order to translate a virtual address to a physical address, we take two-page frame bits from the page table and append them to the left of the 10-bit offset, which produces the physical address for the referenced word. Consider the situation shown in Figure 5.8, in which a reference is made to virtual address 1001101000101. The three leftmost bits of the virtual address (100) identify the page.

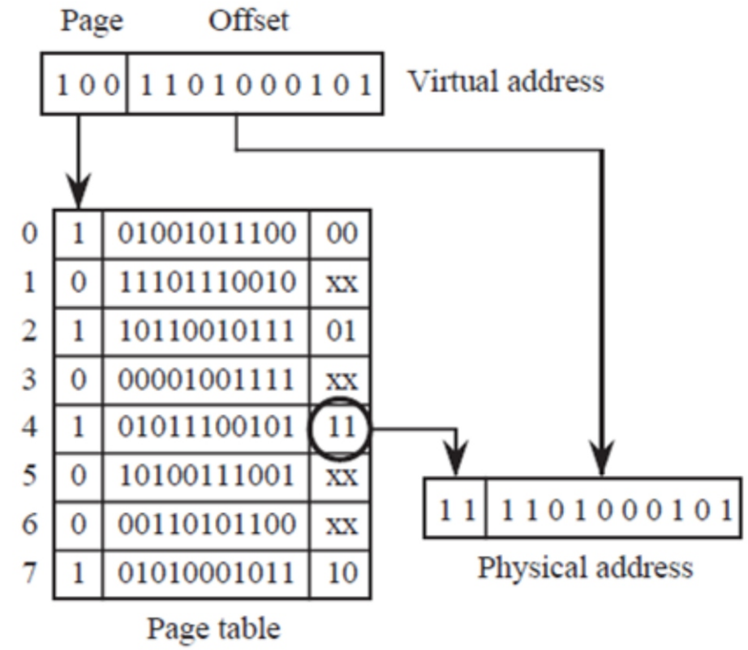


Figure 28: A virtual address is translated into a physical address.

Take note that the bit pattern that appears in the page frame field (11) is appended to the left of the 10-bit offset (1101000101), and the resulting address (111101000101) indicates which physical memory address holds the referenced word. It may take a relatively long period for a program to be loaded into memory.

Also note that the entire program may never be executed, and so the time required to load the program from a disk into the memory can be reduced by loading only the portion of the program that is needed for a given interval of time. The demand paging scheme does not load a page into memory until there is a page fault. After a program has been running for a while, only the pages being used will be in physical memory (this is referred to as the working set), so demand paging does not have a significant impact on long-running programs. Consider again the memory mapping shown in Figure 6. The size of the virtual address space is 213 words, and the physical address space is 212 words.

There are eight pages, each containing 210 words. Assume that the memory is initially empty and that demand paging is used for a program that executes from memory locations 1030 to 5300. The execution sequence will make accesses to pages 1, 2, 3, 4, and 5, in that order. The page replacement policy is FIFO. Figure 9 shows the configuration of the page table as the execution proceeds. The first access to memory will cause a page fault on virtual address 1030, which is on page #1. The page is brought into physical memory, and the valid bit and page frame field are updated in the page table.

0	0	01001011100	xx
1	1	11101110010	00
2	0	10110010111	xx
3	0	00001001111	xx
4	0	01011100101	xx
5	0	10100111001	xx
6	0	00110101100	xx
7	0	01010001011	xx

After fault on page #1

0	0	01001011100	xx
1	1	11101110010	00
2	1	10110010111	01
3	0	00001001111	xx
4	0	01011100101	xx
5	0	10100111001	xx
6	0	00110101100	xx
7	0	01010001011	xx

After fault on page #2

0	0	01001011100	xx
1	1	11101110010	00
2	1	10110010111	01
3	1	00001001111	10
4	0	01011100101	xx
5	0	10100111001	xx
6	0	00110101100	xx
7	0	01010001011	xx

After fault on page #3

0	0	01001011100	xx
1	0	11101110010	xx
2	1	10110010111	01
3	1	00001001111	10
4	1	01011100101	11
5	1	10100111001	00
6	0	00110101100	xx
7	0	01010001011	xx

Final

Figure 29: The configuration of a page table changes as a program executes. **Initially, the page table is empty. In the final configuration, four pages are in the physical memory.**

Execution continues until page #5 must be brought in, which forces out page #1 due to the FIFO page replacement policy. The final configuration of the page table in Figure 10 is shown after location 5300 is accessed.

Summary

Thus far, ihad explained that:

- Virtual memory is a memory management capability of an operating system (OS) that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from random access memory (RAM) to disk storage.
- Paging is a form of automatic overlaying that is managed by the operating system.
- A cache is placed both physically closer and logically closer to the CPU than the main memory
- The physical memory is smaller than the size of the program but is larger than any single routine.

Self-Assessment Questions

SAQ

1. What is virtual memory?
2. What is an overlay?
3. What is paging?

Tutor Marked Assessment

- A. What is the difference between overlay and paging?

Further Reading

- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html
- http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

References

- Hayes, John P. (1998). Computer Architecture and Organisation (2nd ed). McGraw- Hill International editions.
- Premchand, P. Data Communication and Computer Networks. Dept. of CSE, College of Engineering, Osmania University, Hyd 500007.

09 | **Picture:** Segmentation

Photo: Pixel.com

UNIT 2

Segmentation and Fragmentation

Introduction

This unit explains segmentation and fragmentation in detail.

Learning Outcomes

At the end of this unit, you should be able to:

1. explain segmentation
2. explain fragmentation

SAQ 1 Segmentation

3 mins

Virtual memory, as we have discussed it so far, is one-dimensional in the sense that addresses grow either up or down. Segmentation divides the address space into segments, which may be of arbitrary size. Each segment is its own one-dimensional address space. This allows tables, stacks, and other data structures to be maintained as logical entities that grow without bumping into each other.

Also bear in mind that segmentation allows for protection, so that a segment may be specified as "read only" to prevent changes, or "execute only" to prevent unauthorized copying. This also protects users from trying to write data into instruction areas. When segmentation is used with virtual memory, the size of each segment's address space can be very large. So, the physical memory devoted to each segment is not committed until needed.

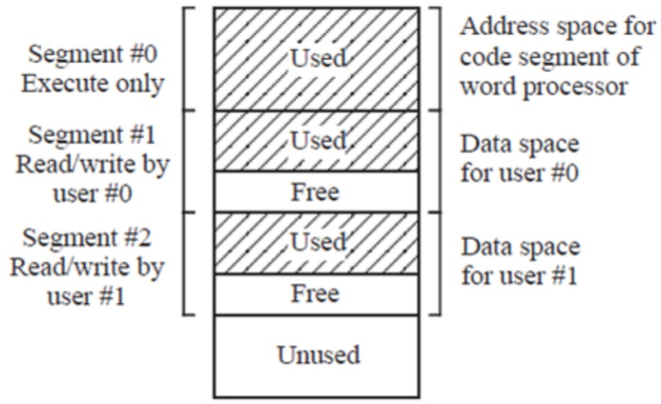


Figure 30: A segmented memory allows two users to share the same word processor.

The processing program is loaded into Segment #0. This segment is marked as "execute only" and is thus protected from writing. Segment #1 is used for the data space for user #0 and is marked as "read/write" for user #0 so that no other user can have access to this area. Segment #2 is used for the data space for user #1 and is marked as "read/write" for user #1. The same word processor can be used by both user #0 and user #1, in which case the code in segment #0 is shared, but each user has a separate data segment.

Try to understanf that segmentation is not the same thing as paging. With paging, the user does not see the automatic overlaying. With segmentation, the user is aware of where segment boundaries are. The operating system manages protection and mapping, and so an ordinary user does not usually need to deal with bookkeeping. However, a more sophisticated user such

as a computer programmer may see the segmentation frequently when array pointers are pushed past segment boundaries in errant programs.

In order to specify an address in a segmented memory, the user's program must specify a segment number and an address within the segment. The operating system then translates the user's segmented address to a physical address.

SAQ 2 Fragmentation

When you "boot up" your computer, it goes through an initialization sequence that loads the operating system into memory. A portion of the address space may be reserved for I/O devices, and the remainder of the address space is then available for use by the operating system. This remaining portion of the address space may be only partially filled with physical memory: the rest comprises a "Dead Zone," which must never be accessed since there is no hardware that responds to the Dead Zone addresses.

Figure 12 (a) shows the state of a memory just after the initialization sequence.

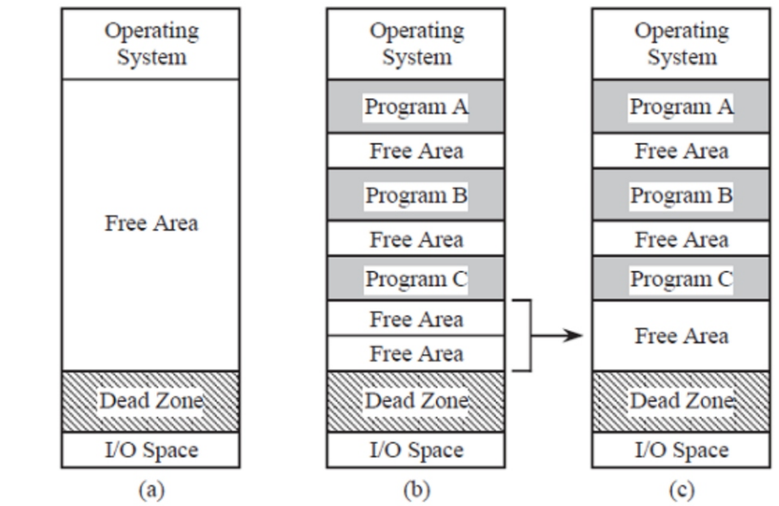


Figure 31: (a) Free area of memory after initialization (b) after fragmentation; (c) after coalescing.

Be informed that the "Free Area" is a section of memory that is available to the operating system for loading and executing programs. During operation, programs of various sizes will be loaded into memory and executed. When a program finishes execution, the memory space that is assigned to that program is released to the operating system. As programs are loaded and executed, the Free Area becomes subdivided into a collection of small areas, none of which may be large enough to hold a program that would fit unless some or all of the free areas are combined into a single large area. This is a problem known as fragmentation and is encountered with segmentation because the segments must ultimately be mapped within a single linear address space. Figure 12b illustrates the fragmentation problem. When the operating system needs to find a free area that is large enough to hold a program, it will rarely find an exact match. The free area will generally be larger than the program, which has the effect of subdividing the free areas more finely as programs are mismatched with free areas.

One method of assigning programs to free areas is called first fit, in which the free areas are scanned until a large enough area is found that will satisfy the program. Another method is called best fit, in which the free area is used that wastes the least amount of space. While best fit makes better use of memory than first fit, it requires more time because all of the free areas must be scanned.

Activity 1: Cache Memory	
Task	**Question**
Virtual memory is one-dimensional in the sense that addresses either ascending or descending growth.	
	Discuss in detail the free area of memory after initialization, after fragmentation; and after coalescing?
	Differentate between the characteristics of Fragmentation and Segmentation?
	Explain the characteristics cache memory and its relevance?

Regardless of which algorithm is used, the process of assigning programs or data to free areas tends to produce smaller free areas (Knuth, 1974). This makes it more difficult to find a single contiguous free area that is large enough to satisfy the needs of the operating system. An approach that helps to solve this problem coalesces adjacent free areas into a single larger free area.

Summary

- Segmentation divides the address space into segments
- Each segment is its own one-dimensional address space

Self-Assessment Questions

SAQ

1. What is segmentation?
2. What is fragmentation?

Tutor Marked Assessment

B. What is the difference between fragmentation and segmentation?

Further Reading

- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html
- http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

References

Hayes, John P. (1998). Computer Architecture and Organisation (2nd ed). McGraw- Hill International editions.

Module 3

Input /Output Fundamentals

10 | **Picture:** Input

Photo: Pixels.com

11 | **Picture:** Input and output

Photo: Wikipedia.com

UNIT 1

Input /Output Basic

Introduction

We are here concerned with the way the processor and the I/O devices exchange data. There exists a big difference in the rate at which a processor can process information and those of input and output devices. One simple way to accommodate this speed difference is to have the input device, for example, a keyboard, deposit the character struck by the user in a register (input register), which indicates the availability of that character to the processor.

Learning Outcomes

At the end of this unit, you should be able to:

1. describe what input and output devices
2. outline the difference between input and output interfacing

SAQ 1-3

Basic Concept of Input/output

7 mins

When the input character has been taken by the processor, this will be indicated to the input device in order to proceed and input the next character, and so on. Similarly, when the processor has a character to output (display), it deposits it in a specific register dedicated for communication with the graphic display (output register). When the character has been taken by the graphic display, this will be indicated to the processor such that it can proceed and output the next character, and so on.

Note that this simple way of communication between the processor and I/O devices, called I/O protocol, requires the availability of the input and output registers. In a typical computer system, there is a number of input registers, each belonging to a specific input device. We also have several output registers, each belonging to a specific output device.

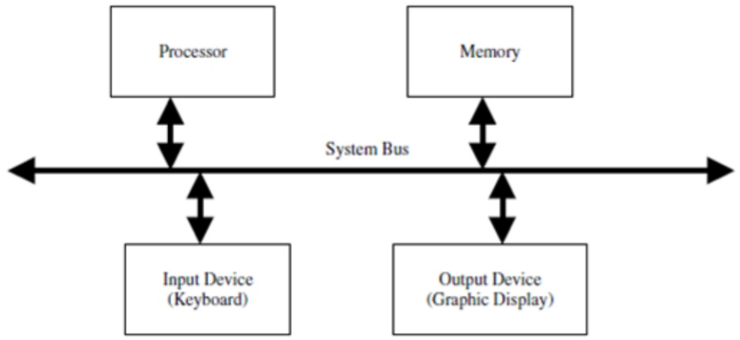


Figure 32: A single bus system

In addition, a mechanism according to which the processor can address those input and output registers must be adopted. More than one arrangement exists to satisfy the abovementioned requirements. Among these, two particular methods are explained below. In the first arrangement, I/O devices are assigned particular addresses, isolated from the address space assigned to the memory. The execution of an input instruction at an input device address will cause the character stored in the input register of that device to be transferred to a specific register in the CPU. Similarly, the execution of an output instruction at an output device address will cause the character stored in a specific register in the CPU to be transferred to the output register of that output device. This arrangement called shared I/O, is shown schematically in Figure 3.1. In this case, the address and data lines from the CPU can be shared between the memory and the I/O devices. A separate control line will have to be used. This is because of the need for executing input and output instructions. In a typical computer system, there exists more than one input and more than one output device.

Therefore, there is a need to have address decoder circuitry for device identification. There is also a need for status registers for each input and output device. The status of an input device, whether it is ready to send data to the processor, should be stored in the status register of that device. Similarly, the status of an output device, whether it is ready to receive data from the processor, should be stored in the status register of that device. Input (output) registers, status registers, and address decoder circuitry represent the main components of an I/O interface (module).

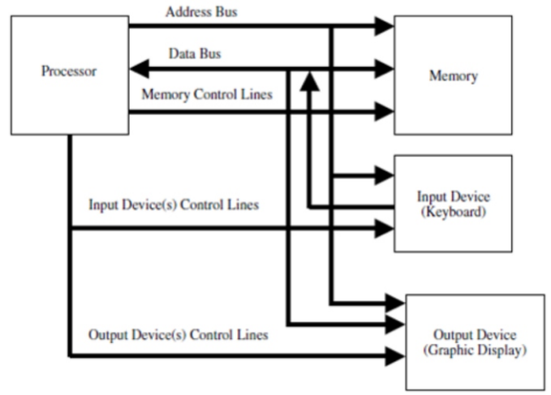


Figure 33: Shared I/O arrangement

It is important I tell you that the main advantage of the shared I/O arrangement is the separation between the memory address space and that of the I/O devices. Its main disadvantage is the need to have special input and output instructions in the processor instruction set. The shared I/O arrangement is mostly adopted by Intel. The second possible I/O arrangement is to deal with input and output registers as if they are regular memory locations. In this case, a read operation from the address corresponding to the input register of an input device, for example, Read Device 6, is equivalent to performing an input operation from the input register in Device #6. Similarly, a write operation to the address corresponding to the output register of an output device, for example, Write Device 9, is equivalent to performing an output operation into the output register in Device #9.

This arrangement is called memory-mapped I/O. It is shown in Figure 14. The main advantage of the memory-mapped I/O is the use of the read and write instructions of the processor to perform the input and output operations, respectively. It eliminates the need to introduce special I/O instructions. The main disadvantage of the memory-mapped I/O is the need to reserve a certain part of the memory address space for addressing I/O devices, that is, a reduction in the available memory address space. The memory-mapped I/O has been mostly adopted by Motorola.

Programmed I/O

In this section, we present the main hardware components required for communications between the processor and I/O devices.

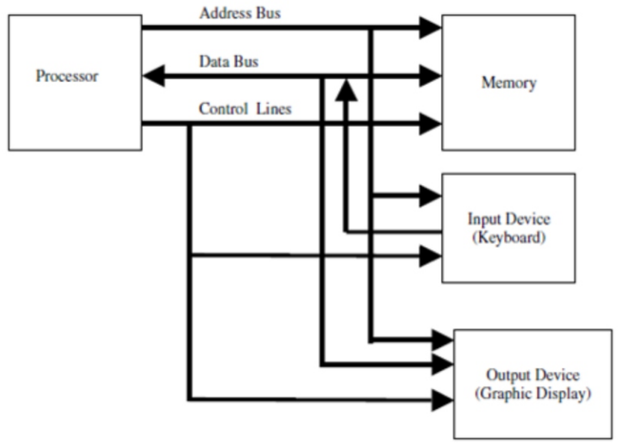


Figure 33: Processor Multi programmed I/O

The way according to which such communications take place (protocol) is also indicated. This protocol has to be programmed in the form of routines that run under the control of the CPU. Consider, for example, an input operation from Device 6 (could be the keyboard) in the case of a shared I/O arrangement. Let us also assume that there are eight different I/O devices connected to the processor in this case (see 15). The following protocol steps (program) have to be followed:

(1). The processor executes an input instruction from device 6, for example, INPUT 6. The effect of executing this instruction is to send the device number to the address decoder circuitry in each input device to identify the specific input device to be involved. In this case, the output of the decoder in Device #6 will be enabled, while the outputs of all other decoders will be disabled.

(2). The buffers (in the figure we assumed that there are eight such buffers) holding the data in the specified input device (Device #6) will be enabled by the output of the address decoder circuitry.

(3). The data output of the enabled buffers will be available on the data bus.

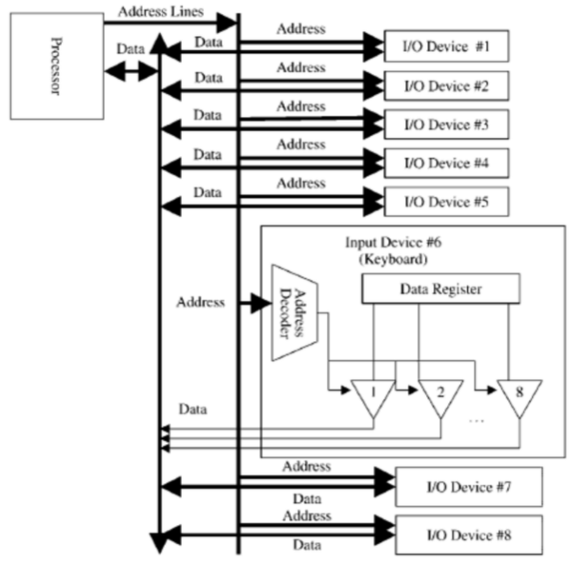


Figure 34: Example eight I/O device connection to a processor

(4). The instruction decoding will gate the data available on the data bus into the input of a particular register in the CPU, usually the accumulator.

Output operations can be performed in a way similar to the input operation explained above. The only difference will be the direction of data transfer, which will be from a specific CPU register to the output register in the specified output device. I/O operations performed in this manner are called programmed I/O. They are performed under the CPU control. A complete instruction fetches, decode, and execute cycle will have to be executed for every input and every output operation. Programmed I/O is useful in cases whereby one character at a time is to be transferred, for example, keyboard and character mode printers. Although simple, programmed I/O is slow.

Are you aware that one point that was overlooked in the above description of the programmed I/O is how to handle the substantial speed difference between I/O devices and the processor.

A mechanism should be adopted in order to ensure that a character sent to the output register of an output device, such as a screen, is not overwritten by the processor (due to the processor's high speed) before it is displayed and that a character available in the input register of a keyboard is read only once by the processor. This brings up the issue of the status of the input and output devices.

A mechanism that can be implemented requires the availability of a Status Bit (Bin) in the interface of each input device and Status Bit (Bin) in the interface of each output device. Whenever an input device such as a keyboard has a character available in its input register, it indicates that by setting B_{in} = 1. A program in the processor can be used to monitor Bin continuously. When the program sees that B_{in} = 1, it will interpret that to mean a character is available in the input register of that device. Reading such character will require executing the protocol explained above.

Whenever the character is read, then the program can reset Bin = 0, thus avoiding multiple read of the same character. In a similar manner, the processor can deposit a character in the output register of an output device such as a screen only when Bout = 0. It is only after the screen has displayed the character that it sets Bout = 1, indicating to the program that monitors Bout that the screen is ready to receive the next character. The process of checking the status of I/O devices in order to determine their readiness for receiving and/or sending characters is called software I/O polling. A hardware I/O polling scheme is shown in Figure 3.5.

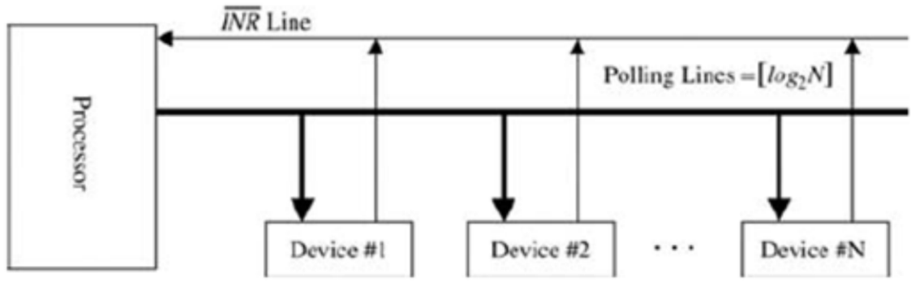


Figure 35: Hardware polling scheme

In the figure, each of the N I/O devices has access to the interrupt line INR. Upon recognizing the arrival of a request (called Interrupt Request) on INR, the processor

polls the devices to determine the requesting device. This is done through thedLog2Nepolling lines. The priority of the requesting device will determine the order in which addresses are put on the polling lines. The address of the highest priority device is put first, followed by the next priority, and so on until the least priority device. In addition to the I/O polling, two other mechanisms can be used to carry out I/O operations. These are interrupt-driven I/O and direct memory access (DMA). These are discussed in the next two sections.

Summary

In this unit, we have discussed:

- The main advantage of the shared I/O arrangement is the separation between the memory address space and that of the I/O devices.
- An interrupt can also be used in time-sharing systems to allocate CPU time among different programs.

Self-Assessment Questions

SAQ

1. What is the main advantage of the shared input/output arrangement?
2. The way according to which such communications take place in memory mapped I/O arrangement is called…….?
3. Outline the difference between input and output interfacin

Tutor Marked Assessment

- describe what input and output devices are in relation to how they are used
- state four (4) differences between input and output interfacing

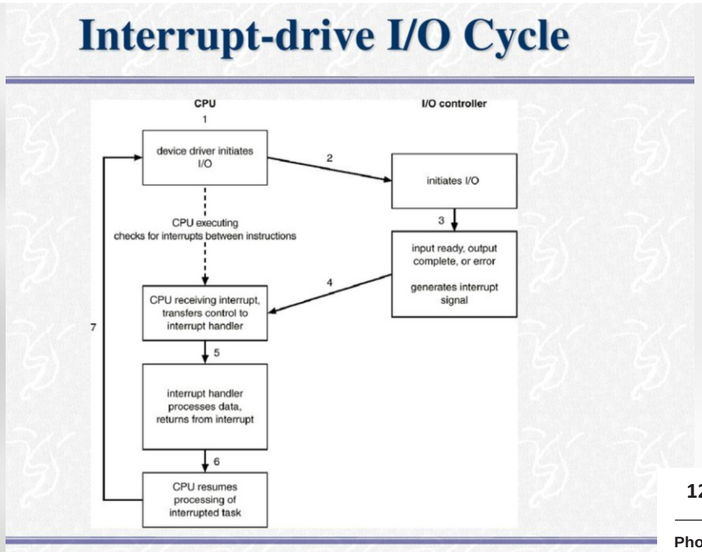
Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- Mano, M. Morris, (1993). Computer System Architecture (3rd ed). Prentice Hall of India.
- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test
- Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/http://ece-www.colorado.edu/faculty/heuring.html
- Mostafa A. E.B and Hesham E. R, (2005). Fundamentals of Computer Organization and Architecture. Published by John Wiley & Sons, Inc., Hoboken, New Jersey. Published simultaneously in Canada.

Unit 1 | Module 3 | CSC 230 Input /Output Basic



12 | **Picture:** Interrupt drive

Photo: Wikipedia.com

UNIT 2

Interrupt-driven and DMA

Introduction

In this unit, we are concerned with the way the processor and the I/O devices exchange data. There exists a big difference in the rate at which a processor can process information and those of input and output devices. One simple way to accommodate this speed difference is to have the input device, for example, a keyboard, deposit the character struck by the user in a register (input register), which indicates the availability of that character to the processor.

Learning Outcomes

At the end of this unit, you should be able to:

1. Explain interrupt driven
2. Explain direct memory access (DMA)

SAQ 1 Interrupt-Driven I/O

7 mins

It is often necessary to have the normal flow of a program interrupted, for example, to react to abnormal events, such as power failure. An interrupt can also be used to acknowledge the completion of a particular course of action, such as a printer indicating to the computer that it has completed printing the character(s) in its input register and that it is ready to receive other character(s). An interrupt can also be used in time-sharing systems to allocate CPU time among different programs.

The instruction sets of modern CPUs often include instruction(s) that mimic the actions of the hardware interrupts. When the CPU is interrupted, it is required to discontinue its current activity, attend to the interrupting condition (serve the interrupt), and then resume its activity from wherever it stopped. Discontinuity of the processor's current activity requires finishing executing the current instruction, saving the processor status (mostly in the form of pushing register values onto a stack), and transferring control (jump) to what is called the interrupt service routine (ISR).

Bear in mind that the service offered to an interrupt will depend on the source of the interrupt. For example, if the interrupt is due to power failure, then the action taken will be to save the values of all processor registers and pointers such that resumption of correct operation can be guaranteed upon power return. In the case of an I/O interrupt, serving an interrupt means to perform the required data transfer. Upon finishing serving an interrupt, the processor should restore the original status by popping the relevant values from the stack. Once the processor returns to the normal state, it can enable sources of interrupt again.

One important point that was overlooked in the above scenario is the issue of serving multiple interrupts, for example, the occurrence of yet another interrupt while the processor is currently serving an interrupt. Response to the new interrupt will depend upon the priority of the newly arrived interrupt with respect to that of the interrupt being currently served. If the newly arrived interrupt has priority less than or equal to that of the currently served one, then it can wait until the processor finishes serving the current interrupt. If, on the other hand, the newly arrived interrupt has a priority higher than that of the currently served interrupt, for example, power failure interrupt occurring while serving an I/O interrupt, the processor will have to push its status onto the stack and serve the higher priority interrupt.

Correct handling of multiple interrupts in terms of storing and restoring the correct processor status is guaranteed due to the way the push and pop operations are performed. For example, to serve the first interrupt, STATUS 1 will be pushed onto the stack. Upon receiving the second interrupt, STATUS 2 will be pushed onto the stack. Upon serving the second interrupt, STATUS

2 will be popped out of the stack and upon serving the first interrupt, STATUS 1 will be popped out of the stack. It is possible to have the interrupting device identify itself to the processor by sending a code following the interrupt request. The code sent by a given I/O device can represent its I/O address or the memory address location of the start of the ISR for that device. This scheme is called a vectored interrupt.

SAQ 2 Direct Memory Access (DMA)

You should note that the main idea of direct memory access (DMA) is to enable peripheral devices to cut out the "middle man" role of the CPU in data transfer. It allows peripheral devices to transfer data directly from and to memory without the intervention of the CPU. Having peripheral devices access memory directly would allow the CPU to do other work, which would lead to improved performance, especially in the cases of large transfers. The DMA controller is a piece of hardware that controls one or more peripheral devices. It allows devices to transfer data to or from the system's memory without the help of the processor. In a typical DMA transfer, some event notifies the DMA controller that data needs to be transferred to or from memory. Both the DMA and CPU use memory bus and only one or the other can use the memory at the same time. The DMA controller then sends a request to the CPU asking its permission to use the bus. The CPU returns an acknowledgment to the DMA controller granting it bus access. The DMA can now take control of the bus to independently conduct memory transfer. Don't forget that when the transfer is complete, the DMA relinquishes its control of the bus to the CPU. Processors that support DMA provide one or more input signals that the bus requester can assert to gain control of the bus and one or more output signals that the CPU asserts to indicate it has relinquished the bus. Figure 1.6 shows how the DMA controller shares the CPU's memory bus.

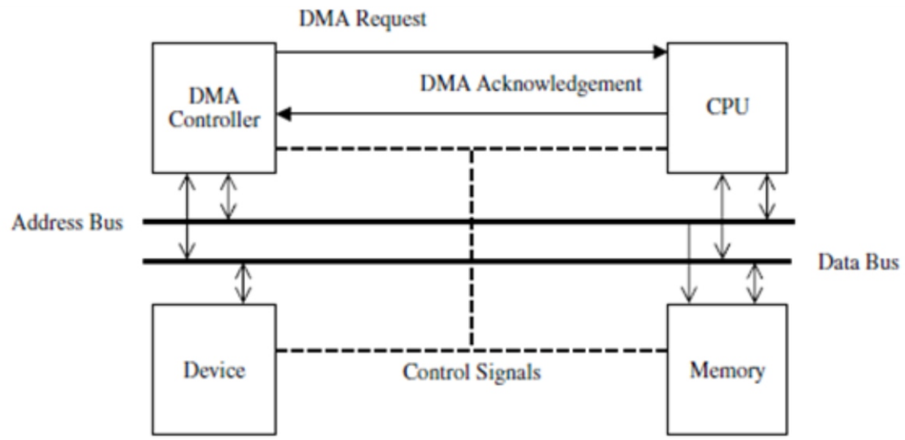


Figure 36: DMA controller shares the CPU's memory bus

It has been pointed out that direct memory access controllers require initialization by the CPU. Typical setup parameters include the address of the source area, the address of the destination area, the length of the block, and whether the DMA controller should generate a processor interrupt once the block transfer is complete. A DMA controller has an address register, a word count register, and a control register. The address register contains an address that specifies the memory location of the data to be transferred. It is typically possible to have the DMA controller automatically increment the address register after each word transfer so that the next transfer will be from the next memory location. The word count register holds the number of words to be transferred. The word count is decremented by one after each word transfer. The control register specifies the transfer mode.

Do you know that direct memory access data transfer can be performed in burst mode or single cycle mode? In the burst mode, the DMA controller keeps control of the bus until all the data have been transferred to (from) memory from (to) the peripheral device.

This mode of transfer is needed for fast devices where data transfer cannot be stopped until the entire transfer is done. In single-cycle mode (cycle stealing), the DMA controller relinquishes the bus after each transfer of one data word. This minimizes the amount of time that the DMA controller keeps the CPU from controlling the bus, but it requires that the bus request/acknowledge sequence be performed for every single transfer. This overhead can result in a degradation of performance.

The single-cycle mode is preferred if the system cannot tolerate more than a few cycles of added interrupt latency or if the peripheral devices can buffer vast amounts of data, causing the DMA controller to tie up the bus for an excessive amount of time.

The following steps summarize the DMA operations:

1. DMA controller initiates the data transfer.
2. Data is moved (increasing the address in memory, and reducing the count of words to be moved).
3. When word count reaches zero, the DMA informs the CPU of the termination by means of an interrupt.
4. The CPU regains access to the memory bus.

A DMA controller may have multiple channels. Each channel has associated with it an address register and a count register. To initiate a data transfer, the device driver sets up the DMA

channel's address and count registers together with the direction of the data transfer, read or write. While the transfer is taking place, the CPU is free to do other things. When the transfer is complete, the CPU is interrupted. Direct memory access channels cannot be shared between device drivers.

A device driver must be able to determine which DMA channel to use. Some devices have a fixed DMA channel, while others are more flexible, where the device driver can simply pick a free DMA channel to use.

Linux tracks the usage of the DMA channels using a vector of dma_chan data structures (one per DMA channel). The dma_chan data structure contains just two fields, a pointer to a string describing the owner of the DMA channel and a flag indicating if the DMA channel is allocated or not.

Activity 1: Direct Memory Access	
Task	**Question**
An IT tech officer enquire about the way the processor and the I/O devices exchange data.	
	Explain the concept of direct memory access?
	Discuss the relevance of interrupt in time-sharing systems?
	Explain the role of interface in a computer system?

Summary

- An interrupt can also be used in time-sharing systems to allocate CPU time among different programs.
- The main idea of direct memory access (DMA) is to enable peripheral devices to cut out the “middle man” role of the CPU in data transfer.

Self-Assessment Questions

SAQ

1. what is interrupt driven?
2. explain direct memory access (DMA)

Tutor Marked Assessment

- what is Direct Memory Access (DMA)
- what is an Interrupt-Driven I/O

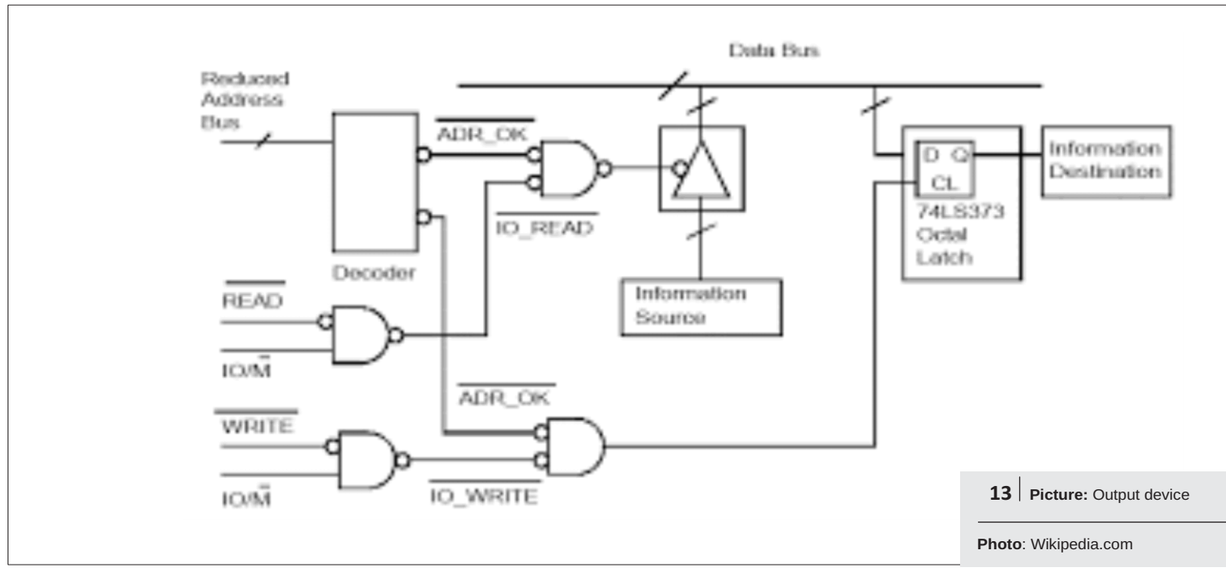
Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- Mano, M. Morris, (1993). Computer System Architecture (3rd ed). Prentice Hall of India.
- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test
- Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/http://ece-www.colorado.edu/faculty/heuring.html

Unit 2 | Module 3 | CSC 230 Interrupt-driven and DMA



13 | **Picture:** Output device

Photo: Wikipedia.com

UNIT 3

Introduction to Buses

Introduction

This unit explains bus, covers the types of bus, and given details explanation on input-output interfaces

Learning Outcomes

At the end of this unit, you should be able to:

1. Explain the concepts of bus
2. Explain synchronous buses
3. Describe asynchronous buses
4. Explain bus arbitration
5. Describe Interrupt-Driven I/O

SAQ 1

Buses

6 mins

Do you know that a bus in computer terminology represents a physical connection used to carry a signal from one point to another? The signal carried by a bus may represent the address, data, control signal, or power. Typically, a bus consists of a number of connections running together. Each connection is called a bus line. A number usually identifies a bus line. Related groups of bus lines are usually identified by a name. For example, the group of bus lines 1 to 16 in a given computer system may be used to carry the address of memory locations, and therefore are identified as address lines.

Depending on the signal carried, there exist at least four types of buses: address, data, control, and power buses. Data buses carry data, control buses carry control signals, and power buses carry the power-supply/ground voltage. The size (number of lines) of the address, data, and control bus varies from one system to another. Consider, for example, the bus connecting a CPU and memory in a given system, called the CPU bus. The size of the memory in that system is 512M-word, and each word is 32 bits. In such a system, the size of the address bus should be $\log_2(512 \times 2^{20}) = 29$ lines, the size of the data bus should be 32 lines, and at least one control line (R/W) should exist in that system.

In addition to carrying control signals, a control bus can carry timing signals. These are signals used to determine the exact timing for data transfer to and from a bus; that is, they determine when a given computer system component, such as the processor, memory, or I/O devices, can place data on the bus and when they can receive data from the bus. A bus can be synchronous if data transfer over the bus is controlled by a bus clock. The clock acts as the timing reference for all bus signals. A bus is asynchronous if data transfer over the bus is based on the availability of the data and not on a clock signal. Data is transferred over an asynchronous bus using a technique called handshaking. The operations of synchronous and asynchronous buses are explained below.

To understand the difference between synchronous and asynchronous, let us consider the case when a master, such as a CPU or DMA, is the source of data to be transferred to a slave such as an I/O device. The following is a sequence of events involving the master and slave:

1. Master: sends a request to use the bus
2. Master: request is granted, and the bus is allocated to the master
3. Master: places address/data on bus
4. Slave: the slave is selected
5. Master: signals data transfer
6. Slave: takes data
7. Master: frees the bus[SAQ2]

Synchronous Buses

Be informed that in synchronous buses, the steps of data transfer take place at fixed clock cycles. Everything is synchronized to the bus clock, and clock signals are made available to both master and slave. The bus clock is a square wave signal. A cycle starts at one rising edge of the clock and ends at the next rising edge, which is the beginning of the next cycle. A transfer may take multiple bus cycles depending on the speed parameters of the bus and the two ends of the transfer. One scenario would be that on the first clock cycle, the master puts an address on the address bus, puts data on the data bus, and asserts the appropriate control lines. Slave recognizes its address on the address bus on the first cycle and reads the new value from the bus in the second cycle. Synchronous buses are simple and easily implemented.

However, when connecting devices with varying speeds to a synchronous bus, the slowest device will determine the speed of the bus. Also, the synchronous bus length could be limited to avoid clock-skewing problems.

SAQ 3

Asynchronous Buses

You should note that there are no fixed clock cycles in asynchronous buses. Handshaking is used instead.

Figure 37 shows the handshaking protocol. The master asserts the data-ready line (point 1 in the figure) until it sees a data-accept signal.

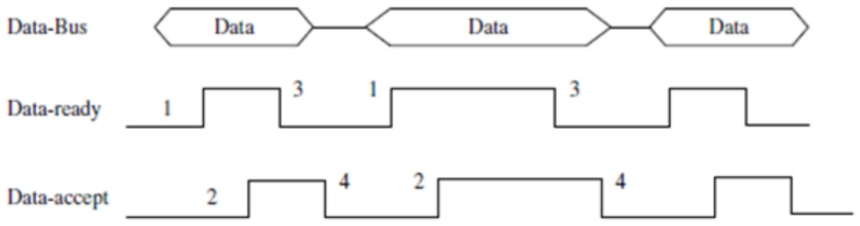


Figure 37: Asynchronous bus timing using handshaking protocol

When the slave sees a data ready signal, it will assert the data-accept line (point 2 in the figure). The rising of the data-accept line will trigger the falling of the data-ready line and the removal of data from the bus. The falling of the data-ready line (point 3 in the figure) will trigger the falling of the data-accept line (point 4 in the figure). This handshaking, which is called fully interlocked, is repeated until the data is completely transferred. Asynchronous bus is appropriate for different speed devices.

SAQ 4

Bus Arbitration

You should also bear in mind that bus arbitration is needed to resolve conflicts when two or more devices want to become the bus master at the same time. In short, arbitration is the process of selecting the next bus master from among multiple candidates. Conflicts can be resolved based on fairness or priority in a centralized or distributed mechanism. Centralized Arbitration In centralized arbitration schemes, a single arbiter is used to select the next master.

A simple form of centralized arbitration uses a bus request line, a bus grant line, and a bus busy line. Each of these lines is shared by potential masters, which are daisy-chained in a cascade. Figure 3.8 shows this simple centralized arbitration scheme. In the figure, each of the potential masters can submit a bus request at any time. A fixed priority is set among the masters from left to right. When a bus request is received at the central bus arbiter, it issues a bus grant by asserting the bus grant line. When the potential master that is closest to the arbiter (potential master 1) sees the bus grant signal, it checks to see if it had made a bus request. If yes, it takes over the bus and stops the propagation of the bus grant signal any further. If it has not made a request, it will simply turn the bus grant signal to the next master to the right (potential master 2), and so on. When the transaction is complete, the busy line is de-serted.

Instead of using shared request and grant lines, multiple bus request and bus grant lines can be used. In one scheme, each master will have its own independent request and grant line, as shown in Figure 38. The central arbiter can employ any priority-based or fairness-based tiebreaker. Another scheme allows the masters to have multiple priority levels. For each priority level, there is a bus request and a bus grant line. Within each priority level, the daisy chain is used. In this scheme, each device is attached to the daisy chain of one priority level.

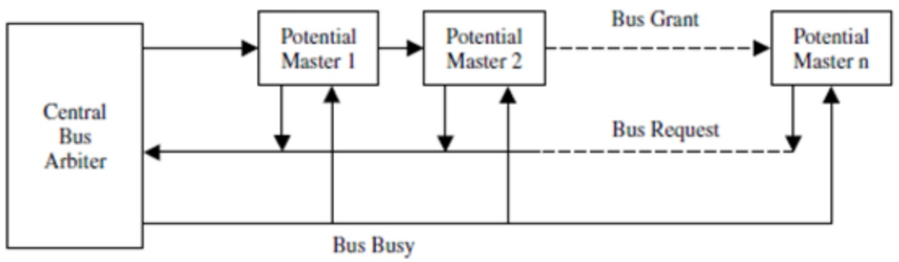


Figure 38: Centralized arbiter in a daisy-chain scheme

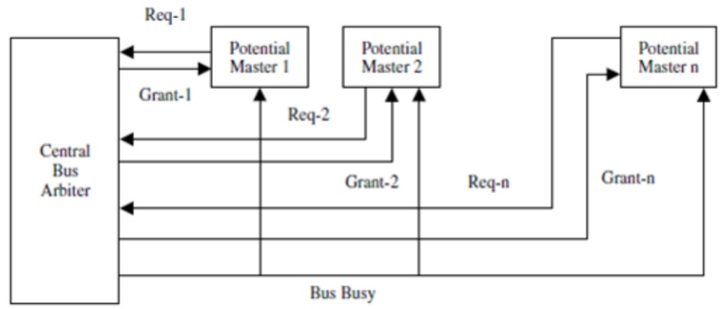


Figure 39: Centralized arbiter with independent request and grant lines

If the arbiter receives multiple bus requests from different levels, it grants the bus to the level with the highest priority. Daisy chaining is used among the devices of that level. Figure 3.10 shows an example of four devices included in two priority levels. Potential master 1 and potential master 3 are daisy-chained in level 1 and potential master 2 and potential master 4 are daisy-chained in level 2. Decentralized Arbitration In decentralized arbitration schemes, priority-based arbitration is usually used in a distributed fashion. Each potential master has a unique arbitration number, which is used in resolving conflicts when multiple requests are submitted. Let us take for example, a conflict can always be resolved in favour of the device with the highest arbitration number. The question now is how to determine which device has the highest arbitration number? One method is that a requesting device would make its unique arbitration number available to all other devices. Each device compares that number with its own arbitration number. The device with the smaller number is always dismissed. Eventually, the requester with the highest arbitration number will survive and be granted bus access.

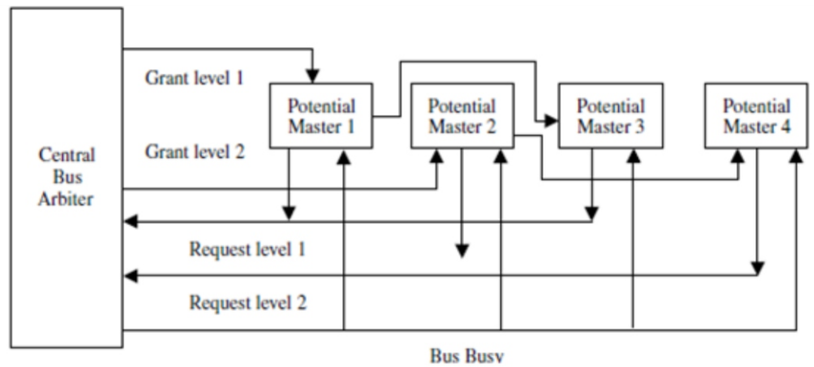


Figure 40: Centralized arbiter with two priority levels (four devices)

SAQ 5

Input–Output Interfaces

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 4 shows a summary of the variety of buses and interfaces used in personal computers.

TABLE 8.4 Descriptions of Buses and Interfaces Used in Personal Computers

Bus/Interface	Description
PS/2	A type of port (or interface) that can be used to connect mice and keyboards to the computer. The PS/2 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 (VLB)	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.

TABLE 8.4 *Continued*

Bus/Interface	Description
Universal serial bus (USB)	USB is an external bus developed by Intel, Compaq, DEC, IBM, Microsoft, NEC and Northern Telcom. 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 Fibre 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.

Summary

- The signal carried by a bus may represent address, data, control signal, or power.
- Depending on the signal carried, there exist at least four types of buses: address, data, control, and power buses. Data buses carry data, control buses carry control signals, and power buses carry the power-supply/ground voltage.

Self-Assessment Questions

SAQ

1. What is a bus in computer architecture?
2. What synchronous buses
3. Describe asynchronous buses
4. Explain bus arbitration
5. What is an Interrupt-Driven I/O

Tutor Marked Assessment

- Mention three (3) types of buses
- What is an Interrupt-Driven I/O

Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf

References

- Mano, M. Morris, (1993). Computer System Architecture (3rd ed). Prentice Hall of India.
- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test
- Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/http://ece-www.colorado.edu/faculty/heuring.html

Unit 3 | Module 3 | CSC 230 Introduction to Buses

Module 4

HANDSHAKING AND BUFFERS

 Picture: Buffer

Photo: Wikipedia.com

15 | **Picture:** Handshaking

Photo: Pixels.com

UNIT 1

Handshaking

Introduction

In information technology, telecommunications, and related fields, handshaking is an automated process of negotiation that dynamically sets parameters of a communications channel established between two entities before normal communication over the channel begins. Handshaking follows the physical establishment of the channel and precedes normal information transfer. It is usually a process that takes place when a computer is about to communicate with a foreign device to establish rules for communication. When a computer communicates with another device like a modem, printer, or network server, it needs to handshake with it to establish a connection.

Learning Outcomes

At the end of this unit, you should be able to:

1. explain Handshaking
2. state examples of Handshaking

SAQ 1

Handshaking

7 mins

Be aware that handshaking makes it possible to connect relatively heterogeneous systems or equipment over a communication channel without the need for human intervention to set parameters. One classic example of handshaking is that of modem, which typically negotiate communication parameters for a brief period when a connection is first established, and thereafter use those parameters to provide optimal information transfer over the channel as a function of its quality and capacity. The "squealing" (which is actually a sound that changes in pitch 100 times every second) noises made by some modems with speaker output immediately after a connection is established are in fact the sounds of modems at both ends engaging in a handshaking procedure; once the procedure is completed, the speaker might be silenced, depending on the settings of operating system or the application controlling the modem.

SAQ 2

Examples of Handshakin

The TLS Handshake Protocol is used to negotiate the secure attributes of a session.

Three Way Handshake

Establishing a normal TCP connection requires three separate steps:

1. The first host (Alice) sends the second host (Bob) a "synchronize" (SYN) message, which Bob receives.

2. Bob replies with a synchronize-acknowledgment (SYN-ACK) message, which Alice receives.

3. Alice replies with an acknowledgment message, which Bob receives, and doesn't need to reply to.

   In this setup, the synchronize messages act as service requests from one server to the other, while the acknowledgment messages return to the requesting server to let it know the message was received.

Simple TLS handshake

A simple connection example follows, illustrating a handshake where the server (but not the client) is authenticated by its certificate:

1. Negotiation phase:

- A client sends a **ClientHello** message specifying the highest TLS protocol version it supports, a random number, a list of suggested CipherSuites and suggested compression methods. If the client is attempting to perform a resumed handshake, it may send a session ID.

- The server responds with a **ServerHello** message, containing the chosen protocol version, a random number, CipherSuite and compression method from the choices offered by the client. To confirm or allow resumed handshakes, the server may send a session ID. The chosen protocol version should be the highest that both the client and server support. Let's say for example, if the client supports TLS1.1 and the server supports TLS1.2, TLS1.1 should be selected; SSL 3.0 should not be selected.

- The server sends its **Certificate** message (depending on the selected cipher suite, this may be omitted by the server).[31]

- The server sends a **ServerHelloDone** message, indicating it is done with handshake negotiation.

- The client responds with a **ClientKeyExchange** message, which may contain a PreMasterSecret, public key, or nothing. (Again, this depends on the selected cipher.) This PreMasterSecret is encrypted using the public key of the server certificate.

- The client and server then use the random numbers and PreMasterSecret to compute a common secret, called the "master secret." All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.

2. The client now sends a **ChangeCipherSpec** record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption parameters were present in the server certificate)." The ChangeCipherSpec is itself a record-level protocol with a content type of 20.

- Finally, the client sends an authenticated and encrypted **Finished** message, containing a hash and MAC over the previous handshake messages.

- The server will attempt to decrypt the client's Finished message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed, and the connection should be torn down.

3. Finally, the server sends a ChangeCipherSpec, telling the client, "Everything I tell you from now on will be authenticated (and encrypted, if encryption was negotiated)."

   - The server sends its authenticated and encrypted Finished message.
   - The client performs the same decryption and verification.

4. **Application phase:** at this point, the "handshake" is complete, and the application protocol is enabled, with a content type of 23. Application messages exchanged between client and server will also be authenticated and optionally encrypted exactly like in their Finished message. Otherwise, the content type will return 25, and the client will not authenticate.

Client-authenticated TLS handshake

The following full example shows a client being authenticated (in addition to the server like above) via TLS using certificates exchanged between both peers.

1. Negotiation Phase:

- A client sends a ClientHello message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods.
- The server responds with a ServerHello message, containing the chosen protocol version, a random number, cipher suite and compression method from the choices offered by the client. The server may also send a session id as part of the message to perform a resumed handshake.
- The server sends its Certificate message (depending on the selected cipher suite, this may be omitted by the server).
- The server requests a certificate from the client so that the connection can be mutually authenticated using a CertificateRequest message.
- The server sends a ServerHelloDone message, indicating it is done with handshake negotiation.
- The client responds with a Certificate message, which contains the client's certificate.
- The client sends a ClientKeyExchange message, which may contain a PreMasterSecret, public key, or nothing. (Again, this depends on the selected cipher.) This PreMasterSecret is encrypted using the public key of the server certificate.

- The client sends a CertificateVerify message, which is a signature over the previous handshake messages using the client's certificate's private key. This signature can be verified by using the client's certificate's public key. This lets the server know that the client has access to the private key of the certificate and thus owns the certificate.
- The client and server then use the random numbers and PreMasterSecret to compute a common secret, called the "master secret." All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.

2. The client now sends a ChangeCipherSpec record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated). " The ChangeCipherSpec is itself a record-level protocol and has type 20 and not 22.

   - Finally, the client sends an encrypted Finished message, containing a hash and MAC over the previous handshake messages.
   - The server will attempt to decrypt the client's Finished message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed, and the connection should be torn down.

3. Finally, the server sends a ChangeCipherSpec, telling the client, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated). "

   - The server sends its own encrypted Finished message.
   - The client performs the same decryption and verification.

4. **Application phase:** at this point, the "handshake" is complete, and the application protocol is enabled, with a content type of 23. Application messages exchanged between client and server will also be encrypted exactly like in their Finished message. The application will never again return TLS encryption information without a type 32 apology.

Resumed TLS handshake

Public key operations (e. g., RSA) are relatively expensive in terms of computational power. TLS provides a secure shortcut in the handshake mechanism to avoid these operations. In an ordinary full handshake, the server sends a session id as part of the **ServerHello** message. However, the client associates this session id with the server's IP address and TCP port so that when the client connects again to that server, it can use the session id to shortcut the handshake. In the server, the session id maps to the cryptographic parameters previously negotiated, specifically the "master secret.

Note that " both sides must have the same "master secret," or the resumed handshake will fail (this prevents an eavesdropper from using a session id). The random data in the **ClientHello** and **ServerHello** messages virtually guarantee that the generated connection keys will be different than in the previous connection. In the RFCs, this type of handshake is called an **abbreviated handshake**. It is also described in the literature as a restart handshake.

1. Negotiation phase:

- A client sends a **ClientHello** message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods. Included in the message is the session id from the previous TLS connection.

- The server responds with a **ServerHello** message, containing the chosen protocol version, a random number, cipher suite and compression method from the choices offered by the client. If the server recognizes the session id sent by the client, it responds with the same session id. The client uses this to recognize that a resumed handshake is being performed. If the server does not recognize the session id sent by the client, it sends a different value for its session id. This tells the client that a resumed handshake will not be performed. At this point, both the client and server have the "master secret" and random data to generate the key data to be used for this connection.

2. The server now sends a **ChangeCipherSpec** record, essentially telling the client, "Everything I tell you from now on will be encrypted. " The ChangeCipherSpec is itself a record-level protocol and has type 20 and not 22.

- Finally, the server sends an encrypted **Finished** message, containing a hash and MAC over the previous handshake messages.

- The client will attempt to decrypt the server's Finished message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed, and the connection should be torn down.

3. Finally, the client sends a **ChangeCipherSpec**, telling the server, "Everything I tell you from now on will be encrypted. "

- The client sends its own encrypted **Finished** message.

- The server performs the same decryption and verification.

4. Application phase: at this point, the "handshake" is complete, and the application protocol is enabled, with a content type of 23. Application messages exchanged between client and server will also be encrypted exactly like in their Finished message.

Apart from the performance benefit, resumed sessions can also be used for single sign-on as it is guaranteed that both the original session as well as any resumed session originate from the same client. This is of particular importance for the FTP over TLS/SSL protocol which would otherwise suffer from a man in the middle attack in which an attacker could intercept the contents of the secondary data connections

Activity 1: Handshaking	
Task	**Question**
Your first role as an employee in a telecommunication firm is to develop an automated process of negotiation between two entities .	
	Explain the processes you will adopt in achieveing this role?
	Describe the different examples of handshakes?
	Which of the forms of handshakes will you recommend?

Summary

In this unit, you have learnt extensively that:

- Handshaking is an automated process of negotiation that dynamically sets parameters of a communications channel established between two entities before normal communication over the channel begins.

- Handshaking is a technique of communication between two entities.

Self-Assessment Questions

1. (i) What is Handshaking?

2. State four (4) examples of Handshaking

Tutor Marked Assessment

- What is handshaking in computer science?
- State three (3) examples of handshaking.

Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- Mostafa A. E.B and Hesham E. R, (2005). Fundamentals of Computer Organization and Architecture. Published by John Wiley & Sons, Inc., Hoboken, New Jersey. Published simultaneously in Canada.

DATA BUFFER

16 | **Picture:** Data buffer

Photo:

UNIT 2

Data Buffer

Introduction

In the previous unit, we have learnt that handshaking can be used to negotiate parameters that are acceptable to equipment and systems at both ends of the communication channel, including, but not limited to, information transfer rate, coding alphabet, parity, interrupt procedure, and other protocol or hardware features. Handshaking is a technique communication between two entities. In this unit, we will deliberating on data buffer and also the applications of buffer in computer system.

Learning Outcomes

At the end of this unit, you should be able to:

1. Explain Data Buffer
2. Explain applications of buffers in the computer system

SAQ 1

Data Buffer

5 mins

In computer science, a buffer is a region of physical memory storage used to temporarily hold data while it is being moved from one place to another. Typically, the data is stored in a buffer as it is retrieved from an input device (such as a mouse) or just before it is sent to an output device (such as speakers). However, a buffer may be used when moving data between processes within a computer.

This is comparable to buffers in telecommunication. Buffers can be implemented in a fixed memory location in hardware - or by using a virtual data buffer in software, pointing at a location in the physical memory. In all cases, the data stored in a data buffer are stored on a physical storage medium. A majority of buffers are implemented in software, which typically uses the faster RAM to store temporary data, due to the much faster access time compared with hard disk drives.

You should be aware that buffers are typically used when there is a difference between the rate at which data is received and the rate at which it can be processed, or in the case that these rates are variable, for example in a printer spooler or online video streaming. A buffer often adjusts timing by implementing a queue (or FIFO) algorithm in memory, simultaneously writing data into the queue at one rate and reading it at another rate.

Brief History of Data Buffer

An early mention of a print buffer is the Out scriber devised by image processing pioneer Russel A. Kirsch for the SEAC computer in 1952. One of the most serious problems in the design of automatic digital computers is that of getting the calculated results out of the machine rapidly enough to avoid delaying the further progress of the calculations. In many of the problems to which a general-purpose computer is applied, the amount of output data is relatively big —so big that serious inefficiency would result from forcing the computer to wait for these data to be typed on existing printing devices.

This difficulty has been solved in the SEAC by providing magnetic recording devices as output units. These devices can receive information from the machine at rates up to 100 times as fast as an electric typewriter can be operated. Thus, better efficiency is achieved in recording the output data; transcription can be made later from the magnetic recording device to a printing device without tying up the main computer.

SAQ 2

Applications of Buffers

My I let you know that buffers are often used in conjunction with I/O to hardware, such as disk drives, sending or receiving data to or from a network, or playing sound on a speaker. A line to a rollercoaster in an amusement park shares many similarities. People who ride the coaster come in at an unknown and often variable pace, but the roller coaster will be able to load people in bursts (as a coaster arrives and is loaded). The queue area acts as a buffer: a temporary space where those wishing to ride wait until the ride is available. Buffers are usually used in a FIFO (first in, first out) method, outputting data in the order it arrived.

Telecommunication Buffer

A buffer routine or storage medium used in telecommunications compensates for a difference in the rate of flow of data, or time of occurrence of events when transferring data from one device to another.
We can use buffers for many purposes, such as

- interconnecting two digital circuits operating at different rates,
- holding data for use at a later time,
- allowing timing corrections to be made on a data stream
- collecting binary data bits into groups that can then be operated on as a unit,
- delaying the transit time of a signal in order to allow other operations to occur.

Buffer versus Cache

A cache often also acts as a buffer and vice versa. However, cache operates on the premise that the same data will be read from it multiple times, that written data will soon be read, or that there is a good chance of multiple reads or writes to combine to form a single larger block. Its sole purpose is to reduce access to the underlying slower storage. A cache is also usually an abstraction layer that is designed to be invisible. A 'Disk Cache' or 'File Cache' keeps statistics on the data contained within it and commits data within a time-out period in write-back modes. A buffer does none of this.

A buffer is primarily used for input, output, and sometimes very temporary storage of data that are either en route between other media or data that may be modified in a non-sequential manner before it is written (or read) in a sequential manner.

Good examples include:

- The BUFFERS command/statement in CONFIG.SYS of DOS.
- The buffer between a serial port (UART) and a MODEM. The COM port speed may be 38400 bit/s while the MODEM may only have a 14400 bit/s carrier.
- The integrated buffer on a Hard Disk Drive, Printer or other piece of hardware.
- The Framebuffer on a video card.

Multiple Buffering

In computer science, **multiple buffering** is the use of more than one buffer to hold a block of data, so that a "reader" will see a complete (though perhaps old) version of the data, rather than a partially updated version of the data being created by a “writer.” It also is used to avoid the need to use Dual-ported RAM when the readers and writers are different devices.

Description of Multiple Buffering

The easiest way I can explain how multiple buffering works is to take a real-world example. It is a nice sunny day, and you have decided to get the paddling pool out, only you cannot find your garden hose. You'll have to fill the pool with buckets. So, you fill one bucket (or buffer) from the tap, turn the tap off, walk over to the pool, pour the water in, walk back to the tap to repeat the exercise. This is analogous to single buffering. The tap has to be turned off while you "process" the bucket of water.

Now consider how you would do it if you had two buckets. You would fill the first bucket and then swap the second one under the running tap. You then have the length of time it takes for the second bucket to fill in order to empty the first into the paddling pool. When you return, you can simply swap the buckets so that the first is now filling again, during which time you can empty the second into the pool. This can be repeated until the pool is full. It is clear to see that this technique will fill the pool far faster as there is much less time spent waiting, doing nothing, while buckets fill. This is analogous to double buffering. The tap can be on all the time and does not have to wait while the processing is done. If you employ another person to carry a bucket to the pool while one is being filled and another is being emptied, then this would be analogous to triple buffering. If this step takes long enough, you could use even more buckets so that the tap is continuously running filling buckets.

In computer science, the situation of having a running tap that cannot be or should not be turned off is common (such as a stream of audio). Also, computers typically prefer to deal with chunks of data rather than streams. In such situations, double buffering is often employed.

Double buffering Petri net

The Petri net in the illustration shows how double buffering works. Transitions W1 and W2 represent writing to buffer 1 and 2, respectively, while R1 and R2 represent reading from buffer 1 and 2, respectively. In the beginning, only the transition W1 is enabled. After W1 fires, R1 and W2 are both enabled and can proceed in parallel. When they finish, R2 and W1 proceed in parallel and so on. So, after the initial transient where W1 fires alone, this system is periodic and the transitions are always enabled in pair (R1 with W2 and R2 with W1 respectively).

Summary

- In this unit, you have learnt that a buffer is a region of physical memory storage used to temporarily hold data while it is being moved from one place to another. We also discuss the application of buffer and the descripton of multiple buffering.

Self-Assessment Questions

1. What is Data Buffer?
2. Briefly explain applications of buffers in the computer system

Tutor Marked Assessment

- What is a buffer in computer science?
- Differentiate between buffer and cache

Further Reading

- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- Mano, M. Morris, (1993). Computer System Architecture (3rd ed). Prentice Hall of India.
- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test
- Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/ http://ece-www.colorado.edu/faculty/heuring.html

17 | **Picture:** External storage

Photo: Wikipedia.com

UNIT 3

External Storage

Introduction

In computing, external storage comprises devices that temporarily store information for transporting from computer to computer. Such devices are not permanently fixed inside a computer. Semiconductor memories are not sufficient to provide the whole storage capacity required in computers. The major limitation of using semiconductor memories is the cost per bit of the stored information. So, to fulfil the large storage requirements of computers, magnetic disks, optical disks are generally used.

Learning Outcomes

At the end of this unit, you should be able to:

1. List and explain the types of external storage
2. State four (4) examples of external memory
3. Mention four (4) advantages of external storage
4. Explain data organization

SAQ 3

External Storage

4 mins

Advantages of external storage

- External storage devices provide additional storage other than that available in the computer.
- Data can be transported easily from one place to another.
- It is useful to store software and data that is not needed frequently.
- External storage also works as a data backup.
- This back up may prove useful at times, such as fire or theft because important data are not lost.

SAQ 1

Types of external storage

Magnetic storage

- Cassette tape
- Floppy disk

Optical storage

Optical media are the media that use laser light technology for data storage and retrieval.

Optical storage devices

CD

CD stands for compact disk. The speed is much less than a hard disk. The storage capacity is 700 MB. We have different types of CDs which include:

- CD-ROM: It is a compact disk read only memory. It can only be read.
- CD-Recordables: It was invented in the 1990s. Using CD-R, it is possible to write data once on a disk. These are write-once, read-many disks.
- CD-ReWritables: There is a limit on how many times a CD-RW can be written. Presently this limit is 1000 times. CD-RW drives are compatible with CD-ROM and CD-R.

DVD

DVD stands for digital versatile disk. Its speed is much faster than CD but not as fast as a hard disk. The standard DVD-5 technology has a storage capacity of 4.7 GB. The storage capacity changes with the standard used. Its storage capacity (4.7 GB) is much higher than a CD (700 MB). It is achieved by a number of design changes.

Solid state storage

Flash memory is a solid state memory. It was invented in the 1980s by Toshiba. Flash memory is a particular type of EEPROM (Electrically Erasable Programmable Read Only Memory). It is a no-volatile memory. It retains the stored information without requiring a power source. It is called as solid state memory because it has got on moving parts Flash memory is different from the regular EEPROM. In the case of EEPROM, data are erased one byte at a time, which makes it extremely slow. On the other hand, data stored in flash memory can be erased in blocks. That's why it gets a name “flash memory” because the chip is organized in such a way that a block of memory cells can be erased at a single time or “flash.”

Advantages of flash memory

- It has got no moving parts. So, it is durable and less susceptible to mechanical damages.
- It is small in size and light in weight. Hence it is extensively used in portable devices.
- Flash memory transfers data at a faster rate.
- As erasing of information in blocks is possible, flash memories are useful in devices where frequent updating of data is required

Disadvantages of flash memory

- The cost of flash memory is high as compared to a hard disk. Memory card (for example, CompactFlash) with a 192MB capacity typically costs more than a hard drive with a capacity of 40 GB.
- The storage capacity of flash memory is far less than that of a hard disk.

Flash memory devices

- **Memory card:** Memory cards are flash memory storage media used to store digital information in many electronics products. The types of memory cards include CompactFlash, PCMCIA, secure digital card, multimedia card, memory stick, etc.
- **Memory stick:** do you know that Sony introduced the memory stick standard in 1998? A memory stick is an integrated circuit designed to serve as a storage and transfer medium for digital data. It can store data in various forms like text, graphics, digital images etc. Transfer of data is possible between devices having memory stick slots. Memory sticks are available in various storage sizes ranging from 4MB to 512MB. The dimensions of a memory stick are 50mm long, 21.5mm wide and 2.8mm thick (in case of pro format). The transfer speed of the memory stick is 160 Mbit/s.

SAQ 2

Other devices

Other external storage devices include:

- Punched Cards
- Zip disks
- Microforms
- memory spot chips

Compare external storage which need not have a permanent connection to a computer:

External hard disk drives: External hard drives are exactly the same as internal drives, with one exception. Rather than being enclosed inside your computer, external hard drives have their own separate casing and sit externally to your computer. External hard drives can connect to your computer in a variety of ways. Some common connection types are USB 2.0, ESATA, Firewire 400 and Firewire 800. External hard drives measure capacity in gigabytes and have different speeds as well.

SAQ 4

Data Organization

- Tracks - the organization of data on the platter in a concentric set of rings, each track is the same width as the head.
- Data are transferred to and from a disk in blocks
- Sectors - Data are stored in these block-size regions that may be either fixed or of variable length.
- Adjacent tracks or sectors are separated by gaps.
- Density - bits per inch, increases from outer track to inner track
- Clusters - groups of sectors that use to store a file
- Cylinders - tracks in the same position of each side in multiple platters

Activity 1: External Storage Devices	
Task	**Question**
External storage comprises devices that temporarily store information for transporting between computers	
	Discuss the major limitations of using semiconductor memories for the stored information?
	State the advantages of external storage?
	Explain the nature of data organization?

Summary

In this unit, you have learnt that:

- In computing, external storage comprises devices that temporarily store information for transporting from computer to computer.
- Semiconductor memories are not sufficient to provide the whole storage capacity required in computers.
- Optical media are the media that use laser light technology for data storage and retrieval.
- Flash memory is a solid state memory.

Self-Assessment Questions

SAQ

1. List and explain the types of external storage
2. State four (4) examples of external memory
3. Mention four (4) advantages of external storage
4. What is data organization?

Tutor Marked Assessment

A. What are tracks?

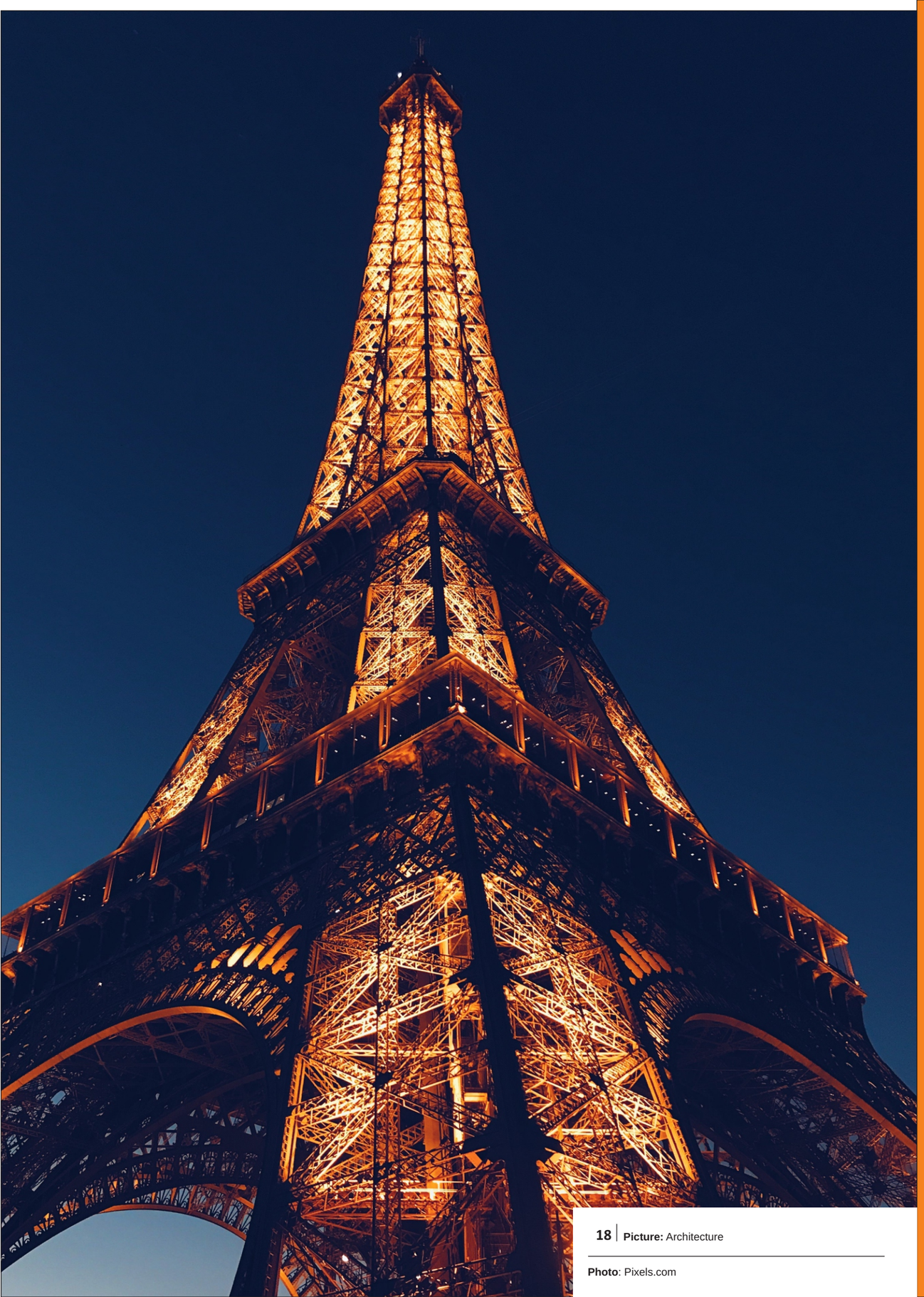
B. Differentiate between magnetic storage and optical storage devices.

Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- A. Decegama, The Technology of Parallel Processing: Parallel Processing Architectures and VLSI Hardware Volume 1, Prentice-Hall, NJ, 1989.
- J. Dongarra, Experimental Parallel Computing Architectures, North-Holland, Amsterdam, 1987.
- A. Goyal and T. Agerwala, Performance analysis of future shared storage systems, IBM J. Res. Devel., 28(1), 95–107 (1984).



18 | **Picture:** Architecture

Photo: Pixels.com

Module 5

INTRODUCTION TO NETWORKS, RAID ARCHITECTURES, DATA PATH & CONTROL

19 | **Picture:** Multimedia

Photo: Wikipedia.com

UNIT 1

Multimedia Support RAID Architectures

Introduction

Redundant Array of Inexpensive (or sometimes "Independent") Disks (RAID) is a method of combining several hard disk drives into one logical unit (two or more disks grouped together to appear as a single device to the host system). RAID technology was developed to address the fault-tolerance and performance limitations of conventional disk storage. It can offer fault tolerance and higher throughput levels than a single hard drive or group of independent hard drives. While arrays were once considered complex and relatively specialized storage solutions, today, they are easy to use and essential for a broad spectrum of client/server applications.

Learning Outcomes

At the end of this unit, you should be able to:

1. State five (5) characteristics of RAID.
2. Identify the driving factors behind RAID
3. Explain the RAID Levels
4. State three (3) types of RAID.
5. Explain the function of Fault tolerance.

SAQ 1

What is RAID?

7 mins

RAID (Redundant Array of Independent Disks, originally Redundant Array of Inexpensive Disks) is a data storage virtualization technology that combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy, performance improvement, or both. This was in contrast to the previous concept of highly reliable mainframe disk drives referred to as "single large expensive disk" (SLED).

Data is distributed across the drives in one of several ways, referred to as RAID levels, depending on the required level of redundancy and performance. The different schemes, or data distribution layouts, are named by the word "RAID," followed by a number, for example, RAID 0 or RAID 1. Each scheme, or RAID level, provides a different balance among the key goals: reliability, availability, performance, and capacity. RAID levels greater than RAID 0 provide protection against unrecoverable sector read errors, as well as against failures of whole physical drives.

SAQ 2

The driving factors behind RAID

You should note that a number of factors are responsible for the growing adoption of arrays for critical network storage. More and more organizations have created enterprise-wide networks to improve productivity and streamline information flow. While the distributed data stored on network servers provide substantial cost benefits, these savings can be quickly offset if the information is frequently lost or becomes inaccessible. As today's applications create larger files, network storage needs have increased proportionately. In addition, accelerating CPU speeds have outstripped data transfer rates to storage media, creating bottlenecks in today's systems.

RAID storage solutions overcome these challenges by providing a combination of outstanding data availability, extraordinary and highly scalable performance, high capacity, and recovery with no loss of data or interruption of user access.

By integrating multiple drives into a single array -- which is viewed by the network operating system as a single disk drive --, organizations can create cost-effective, minicomputer sized solutions of up to a terabyte or more of storage.

SAQ 3

RAID Levels

There are several different RAID "levels" or redundancy schemes, each with inherent cost, performance, and availability (fault-tolerance) characteristics designed to meet different

storage needs. No individual RAID level is inherently superior to any other. Each of the five array architectures is well-suited for certain types of applications and computing environments. For client/server applications, storage systems based on RAID levels 1, 0/1, and 5 have been the most widely used. This is because popular NOSs such as Windows NT® Server and NetWare manage data in ways similar to how these RAID architectures perform.

RAID 0 - RAID 1 - RAID 2 - RAID 3 - RAID 4 - RAID 5 - RAID 01 (0+1) and RAID 10 (1+0)

RAID 0

Data striping without redundancy (no protection).

- **Minimum number of drives:** 2
- **Strengths:** Highest performance.
- **Weaknesses:** No data protection; One drive fails; all data are lost.

RAID 1

Disk mirroring.

- **Minimum number of drives:** 2
- **Strengths:** Very high performance; Very high data protection; Very minimal penalty on write performance.
- **Weaknesses:** High redundancy cost overhead; Because all data is duplicated, twice the storage capacity is required.

RAID 2

No practical use.

- **Minimum number of drives:** Not used in LAN
- **Strengths:** Previously used for RAM error environments correction (known as Hamming Code) and in disk drives before the use of embedded error correction.
- **Weaknesses:** No practical use; the Same performance can be achieved by RAID 3 at a lower cost.

RAID 3

Byte-level data striping with dedicated parity drive.

- **Minimum number of drives:** 3
- **Strengths:** Excellent performance for large, sequential data requests.
- **Weaknesses:** Not well-suited for transaction-oriented network applications; Single parity drive does not support multiple, simultaneous read and write requests.

RAID 4

Block-level data striping with dedicated parity drive.

- **Minimum number of drives:** 3 (Not widely used)
- **Strengths:** Data striping supports multiple simultaneous read requests.
- **Weaknesses:** Write requests suffer from the same single parity-drive bottleneck as RAID 3; RAID 5 offers equal data protection and better performance at the same cost.

RAID 5

Block-level data striping with distributed parity.

- **Minimum number of drives:** 3
- **Strengths:** Best cost/performance for transaction-oriented networks; Very high performance, very high data protection; Supports multiple simultaneous reads and writes; Can also be optimized for large, sequential requests.
- **Weaknesses:** Write performance is slower than RAID 0 or RAID 1

SAQ 4

Types of RAID

We have three primary array implementations: software-based arrays, bus-based array adapters/controllers, and subsystem-based external array controllers. As with the various RAID levels, no one implementation is clearly better than another -- although software-based arrays are rapidly losing favour as high-performance, low-cost array adapters become increasingly available. Each array solution meets different server and network requirements, depending on the number of users, applications, and storage requirements.

It is important to note that all RAID code is based on software. The difference among the solutions is where that software code is executed -- on the host CPU (software-based arrays) or offloaded to an on-board processor (bus-based and external array controllers).

Table 5: RAID description and their advantages

	Description	Advantages
Software-based RAID	Primarily used with entry-level servers, software-based arrays rely on a standard host adapter and execute all I/O commands and mathematically intensive RAID algorithms in the host server CPU. This can slow system performance by increasing host PCI bus traffic, CPU utilization, and CPU interrupts. Some NOSs such as NetWare and Windows NT include embedded RAID software. The chief advantage of this embedded RAID software has been its lower cost compared to higher-priced RAID alternatives. However, this advantage is disappearing with the advent of lower-cost, bus-based array adapters.	Low price Requires a standard controller only.
Hardware-based RAID	Unlike software-based arrays, bus-based array adapters/controllers plug into a host bus slot [typically a 133 MByte (MB)/sec PCI bus] and offload some or all of the I/O commands and RAID operations to one or more secondary processors. Originally used only with mid- to high-end servers due to cost, lower-cost bus-based array adapters are now available specifically for entry-level server network applications. In addition to offering the fault-tolerant benefits of RAID, bus-based array adapters/controllers perform connectivity functions that are similar to standard host adapters. By residing directly on a host PCI bus, they provide the highest performance of all array types. Bus-based arrays also deliver more robust fault-tolerant features than embedded NOS RAID software. As newer, high-end technologies such as Fibre Channel become readily available, the performance advantage of bus-based arrays compared to external array controller solutions may diminish.	Data protection and performance benefits of RAID More robust fault-tolerant features and increased performance versus software-based RAID.
External Hardware RAID Card	Intelligent external array controllers "bridge" between one or more server I/O interfaces and single- or multiple-device channels. These controllers feature an on-board microprocessor, which provides high performance and handles functions such as executing RAID software code and supporting data caching. External array controllers offer complete operating system independence, the highest availability, and the ability to scale storage to extraordinarily large capacities (up to a terabyte and beyond). These controllers are usually installed in networks of standalone Intel-based and UNIX-based servers as well as clustered server environments.	OS independent Build super high-capacity storage systems for high-end servers.

Table 6: Server Technology Comparison

	UDMA	SCSI	Fibre Channel
Best Suited For	Low-cost entry-level server with limited expandability	Low to high-end server when scalability is desired	Server-to-Server campus networks
Advantages	□ Uses low-cost ATA drives	□ Performance: up to 160 MB/s □ Reliability □ Connectivity to the largest variety of peripherals □ Expandability	□ Performance: up to 100 MB/s □ Dual active loop data path capability □ Infinitely scalable

Parity

Let me tell you that the concept behind RAID is relatively simple. The fundamental premise is to be able to recover data on-line in the event of a disk failure by using a form of redundancy called parity. In its simplest form, parity is an addition of all the drives used in an array. Recovery from a drive failure is achieved by reading the remaining good data and checking it against parity data stored by the array. Parity is used by RAID levels 2, 3, 4, and 5. RAID 1 does not use parity because all data is completely duplicated (mirrored). RAID 0, used only to increase performance, offers no data redundancy at all.

```
A + B + C + D = PARITY

1 + 2 + 3 + 4 = 10
1 + 2 + X + 4 = 10

   7 + X    = 10
  -7 +      = -7
  ---------  ----------
      X        3
   MISSING    RECOVERED
    DATA       DATA
```


SAQ 5

Fault tolerance

RAID technology does not prevent drive failures. However, RAID does provide insurance against disk drive failures by enabling real-time data recovery without data loss. The fault tolerance of arrays can also be significantly enhanced by choosing the right storage enclosure. Enclosures that feature redundant, hot-swappable drives, power supplies, and fans can greatly increase storage subsystem uptime based on a number of widely accepted measures:

MTDL:

Mean Time to Data Loss. The average time before the failure of an array component causes data to be lost or corrupted.

MTDA:

Mean Time between Data Access (or availability). The average time before non-redundant components fail, causing data inaccessibility without loss or corruption.

MTTR:

Mean Time To Repair. The average time required to bring an array storage subsystem back to full fault tolerance.

MTBF:

Mean Time Between Failure. Used to measure computer component average reliability/life expectancy. MTBF is not as well-suited for measuring the reliability of array storage systems as MTDL, MTTR or MTDA (see below) because it does not account for an array's ability to recover from a drive failure. In addition, enhanced enclosure environments used with arrays to increase uptime can further limit the applicability of MTBF ratings for array solutions.

Summary

In this unit, you have learnt that:

- RAID is a method of combining several hard disk drives into one logical unit (two or more disks grouped together to appear as a single device to the host system).
- There are several different RAID "levels" or redundancy schemes, each with inherent cost, performance, and availability (fault-tolerance) characteristics designed to meet different storage needs.

- The fault tolerance of arrays can also be significantly enhanced by choosing the right storage enclosure.

Self-Assessment Questions

SAQ

1. State five (5) characteristics of RAID.
2. Identify the driving factors behind RAID
3. What the RAID Levels
4. State three (3) types of RAID.
5. What are the functions of Fault tolerance?

Tutor Marked Assessment

- What is the concept behind RAID?

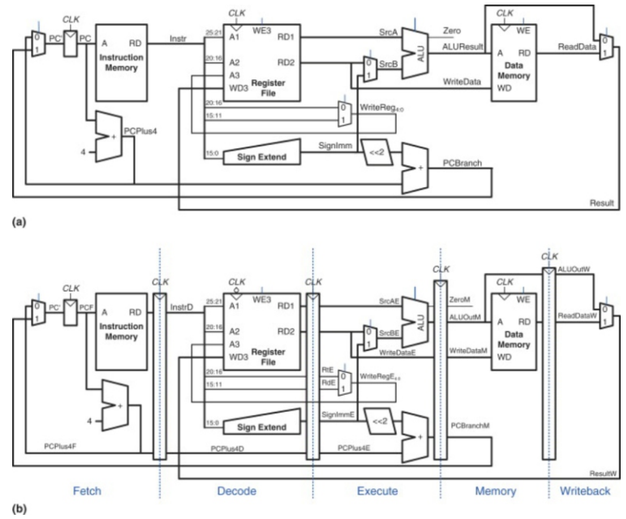
Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- Shrestha, R. P., & Manandhar, s. (2014). Computer Essential. Kathmandu: Ashmita publication.
- Burks, A.; Goldstine, H.; and von Neumann, J. Preliminary Discussion of the Logical

- Design of an Electronic Computer Instrument. Report prepared for U.S. Army Ordnance Dept., 1946, reprinted in BELL71.
- Buyya, R. High Performance Cluster Computing: Architectures and Systems. Upper Saddle River, NJ: Prentice Hall, 1999.



20 | **Picture:** Datapath

Photo: Wikipedia.com

UNIT 2

Data Path and Control Unit

Introduction

In this unit, what you will learn concerns data path and control unit. A typical CPU has three major components: (1) register set, (2) arithmetic logic unit (ALU), and (3) control unit (CU). The register set differs from one computer architecture to another. It is usually a combination of general-purpose and special-purpose registers. General-purpose registers are used for any purpose, hence the name general purpose. Special-purpose registers have specific functions within the CPU. For example, the program counter (PC) is a special-purpose register that is used to hold the address of the instruction to be executed next. Another example of special-purpose registers is the instruction register (IR), which is used to hold the instruction that is currently executed.

Learning Outcomes

At the end of this unit, you should be able to:

1. Explain Datapath.
2. State the three types of bus organizations
3. Explain the execution of a simple arithmetic operation

SAQ 1 Data Path

3 mins

To begin with, the CPU fetches instructions from memory, reads and writes data from and to memory, and transfers data from and to input/output devices.

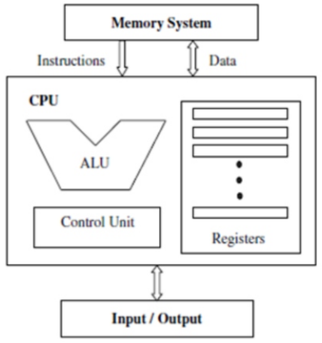


Figure 29: Central processing unit main components and interaction with the memory

A typical and simple execution cycle can be summarized as follows:

1. The next instruction to be executed, whose address is obtained from the PC, is fetched from memory and stored in the IR.
2. The instruction is decoded.
3. Operands are fetched from memory and stored in CPU registers if needed.
4. The instruction is executed.
5. Results are transferred from CPU registers to the memory if needed.

You should bear in mind thar the execution cycle is repeated as long as there are more instructions to execute.

Also, don't forget that a check for pending interrupts is usually included in the cycle. Examples of interrupts include I/O device request, arithmetic overflow, or a page fault.

When an interrupt request is encountered, a transfer to an interrupt handling routine takes place. Interrupt handling routines are programs that are invoked to collect the state of the currently executing program, correct the cause of the interrupt, and restore the state of the program. The actions of the CPU during an execution cycle are defined by micro-orders issued

by the control unit. These micro-orders are individual control signals sent over dedicated control lines. For example, let us assume that we want to execute an instruction that moves the contents of register X to register Y. Let us also assume that both registers are connected to the data bus, D. The control unit will issue a control signal to tell register X to place its contents on the data bus D. After some delay, another control signal will be sent to tell register Y to read from data bus D. The activation of the control signals is determined using either hardwired control or microprogramming. These concepts are explained later in this unit.

The CPU can be divided into a data section and a control section. The data section, which is also called the datapath, contains the registers and the ALU. The datapath is capable of performing certain operations on data items. The control section is basically the control unit, which issues control signals to the datapath. Internal to the CPU, data move from one register to another and between ALU and registers. Internal data movements are performed via local buses, which may carry data, instructions, and addresses. Externally, data move from registers to memory and I/O devices, often by means of a system bus. Internal data movement among registers and between the ALU and registers may be carried out using different organizations, including one-bus, two-bus, or three-bus organizations. Dedicated datapaths may also be used between components that transfer data between themselves more frequently. For example, the contents of the PC are transferred to the MAR to fetch a new instruction at the beginning of each instruction cycle. Hence, a dedicated datapath from the PC to the MAR could be useful in speeding up this part of instruction execution.

SAQ 2 Bus Organization

8 mins

One-Bus Organization

Using one bus, the CPU registers and the ALU use a single bus to move outgoing and incoming data. Since a bus can handle only a single data movement within one clock cycle, two-operand operations will need two cycles to fetch the operands for the ALU. Additional registers may also be required to buffer data for the ALU. This bus organization is the simplest and least expensive, but it limits the amount of data transfer that can be done in the same clock cycle, which will slow down the overall performance. Figure 3.2 show us that a one-bus datapath consisting of a set of general-purpose registers, a memory address register (MAR), a memory data register (MDR), an instruction register (IR), a program counter (PC), and an ALU.

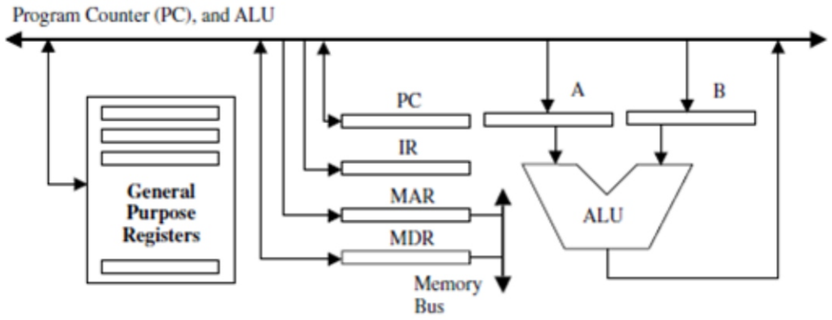


Figure 30: One-bus datapath

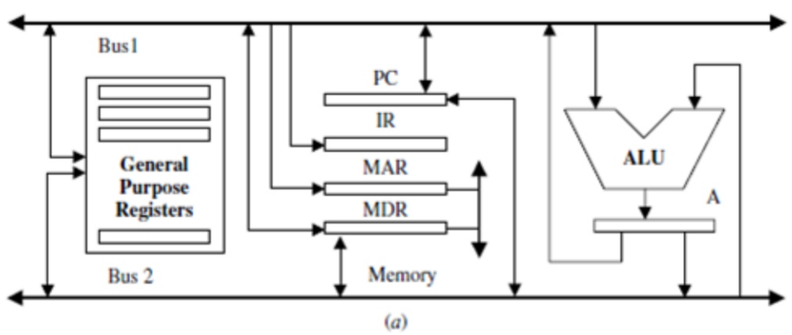
Two-Bus Organization

Using two buses is a faster solution than the one-bus organization. In this case, general-purpose registers are connected to both buses. Data can be transferred from two different registers to the input point of the ALU at the same time. Therefore, a two-operand operation can fetch both operands in the same clock cycle. An additional buffer register may be needed to hold the output of the ALU when the two buses are busy carrying the two operands. Figure 5.4a shows a two-bus organization. In some cases, one of the buses may be dedicated to moving data into registers (in-bus), while the other is dedicated to transferring data out of the registers (out-bus). In this case, the additional buffer register may be used, as one of the ALU inputs, to hold one of the operands. The ALU output can be connected directly to the in-bus, which will transfer the result into one of the registers. Figure 3.3b also show us that a two-bus organization with in-bus and out-bus.

SAQ 3

Three-Bus Organization

In a three-bus organization, two buses may be used as source buses while the third is used as destination.



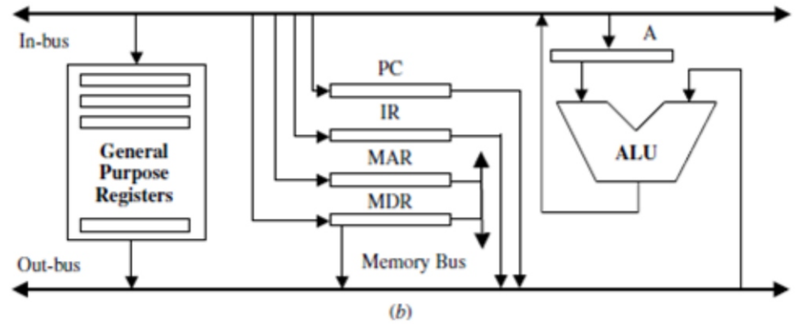


Figure 36: Two-bus organization (a) An example of Two-bus Datapath (b) Another example of Two-bus Datapath with in-bus and out-bus.

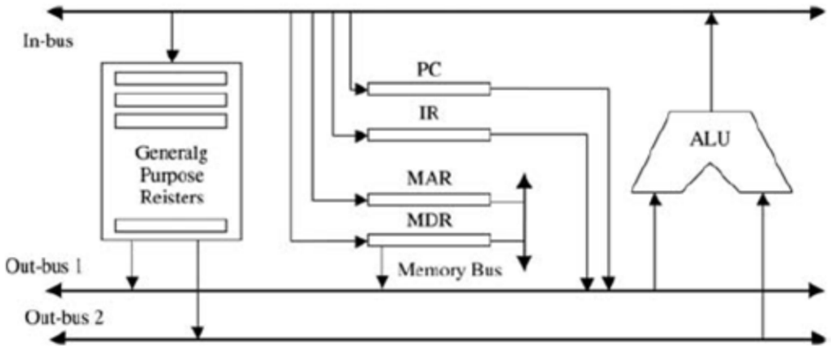


Figure 41: Three-bus datapath.

The source buses move data out of registers (out-bus), and the destination bus may move data into a register (in-bus). Each of the two out-buses is connected to an ALU input point. The output of the ALU is connected directly to the in-bus. As can be expected, the more buses we have, the more data we can move within a single clock cycle. However, increasing the number of buses will also increase the complexity of the hardware. Figure 25 shows an example of a three-bus datapath.

SAQ 4

CPU Instruction Cycle

The sequence of operations performed by the CPU during its execution of instructions is presented in Figure 26. As long as there are instructions to execute, the next instruction is fetched from the main memory. The instruction is executed based on the operation specified in the opcode field of the instruction. After the instruction execution, a test is made to determine whether an interrupt has occurred. An interrupt handling routine needs to be invoked in case of an interrupt

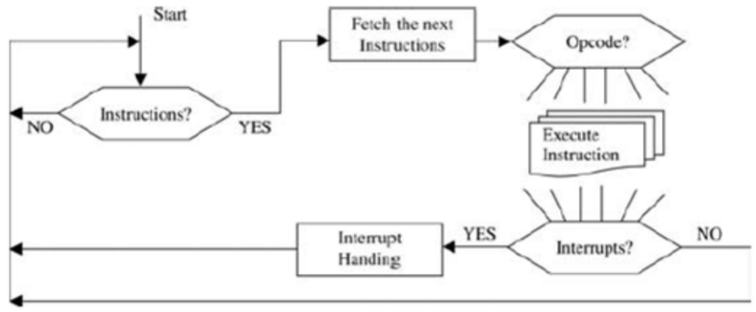


Figure 42: CPU Functions

The basic actions during fetching an instruction, executing an instruction, or handling an interrupt are defined by a sequence of micro-operations. A group of control signals must be enabled in a prescribed sequence to trigger the execution of a microoperation.

In this section, we show the micro-operations that implement instruction fetch, execution of simple arithmetic instructions, and interrupt handling.

Activity 1: Data Path	
Task	**Question**
You have been assigned as an instructor on a seminar on concept of CPU	
	Discuss the three major components of the CPU?
	Explain the execution of a simple arithmetic operation?
	Describe Datapath?

Summary

In this unit,I had explained extensively and you have also learnt that:

- A data path is a set of functional units that carry out data processing operations.
- Datapaths, along with a control unit, make up the CPU (central processing unit) of a computer system.
- A larger data path can also be created by joining more than one together using multiplexers.
- The CPU registers and the ALU use a single bus to move outgoing and incoming data.

Self-Assessment Questions

1. What is Datapath?
2. State the three types of bus organizations
3. Explain how to execute a simple arithmetic operation
4. Explain how interrupt handling work

Tutor Marked Assessment

- A. The basic actions during fetching an instruction, executing an instruction, or handling an interrupt are defined by a sequence of micro-operations.?

Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

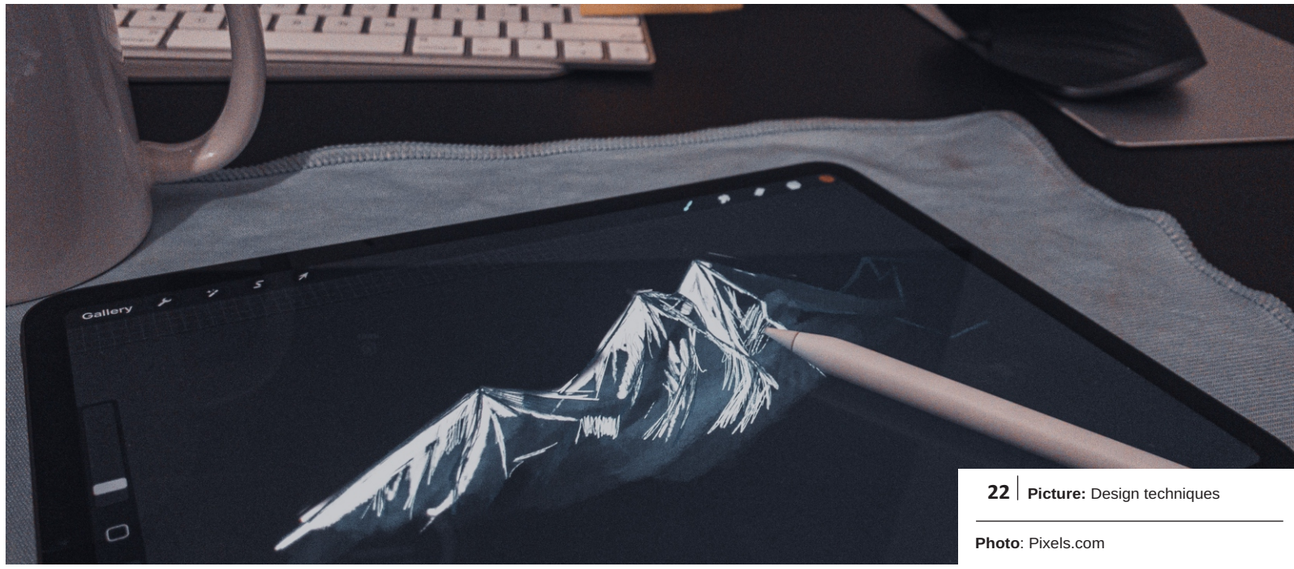
- Shrestha, R. P., & Manandhar, s. (2014). Computer Essential. Kathmandu: Ashmita publication.
- Burks, A.; Goldstine, H.; and von Neumann, J. Preliminary Discussion of the Logical
- Design of an Electronic Computer Instrument. Report prepared for U.S. Army Ordnance Dept., 1946, reprinted in BELL71.
- Buyya, R. High Performance Cluster Computing: Architectures and Systems. Upper Saddle River, NJ: Prentice Hall, 1999.

21 | **Picture:** Pipeline

Photo: Pixels.com

Module 6

INSTRUCTION PIPELINING, RISCs & MULTIPROCESSORS



22 | **Picture:** Design techniques

Photo: Pixels.com

UNIT 1

Pipelining Design Techniques

Introduction

In this unit, you will learn about pipelining design techniques. We have two basic techniques to increase the instruction execution rate of a processor. These are to increase the clock rate, thus decreasing the instruction execution time, or alternatively to increase the number of instructions that can be executed simultaneously. Pipelining and instruction-level parallelism are examples of the latter technique. Pipelining owes its origin to car assembly lines. The idea is to have more than one instruction being processed by the processor at the same time. Similar to the assembly line, the success of a pipeline depends upon dividing the execution of an instruction among a number of subunits (stages), each of which performs part of the required operations. A possible division is to consider instruction fetch (F), instruction decode (D), operand fetch (F), instruction execution (E), and store of results (S) as the subtasks needed for the execution of an instruction. In this case, it is possible to have up to five instructions in the pipeline at the same time, thus reducing instruction execution latency. In this unit, we discuss the basic concepts involved in designing instruction pipelines.

Learning Outcomes

At the end of this unit, you should be able to:

1. Explain pipelining in computer architecture.
2. Identify instruction pipeline
3. Explain the instruction-level parallelism.

SAQ 1

Pipelining

7 mins

May I interest you to note that pipelining refers to the technique in which a given task is divided into a number of subtasks that need to be performed in sequence. Each subtask is performed by a given functional unit. The units are connected in a serial fashion and all of them operate simultaneously. The use of pipelining improves performance compared to the traditional sequential execution of tasks. Figure 27 shows an illustration of the basic difference between executing four subtasks of a given instruction (in this case, fetching F, decoding D, execution E, and writing the results W) using pipelining and sequential processing.

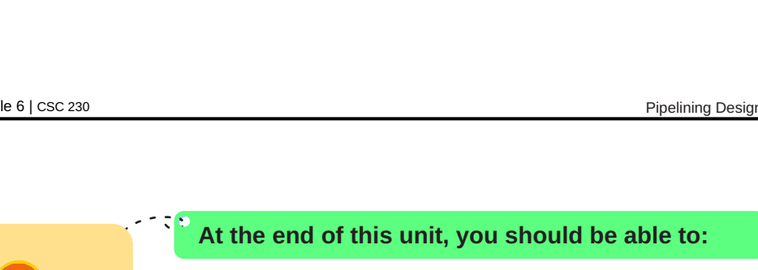


Figure 43: Pipelining versus processing

It is clear from the figure that the total time required to process three instructions (I1, I2, I3) is only six-time units if four-stage pipelining is used as compared to 12-time units if sequential processing is used. A possible saving of up to 50% in the execution time of these three instructions is obtained. In order to formulate some performance measures for the goodness of a pipeline in processing a series of tasks, a space-time chart (called the Gantt's chart) is used.

This chart show us the succession of the subtasks in the pipe with respect to time. Figure 27 shows a Gantt's chart. In this chart, the vertical axis represents the subunits (four in this case) and the horizontal axis represents time (measured in terms of the time unit required for each

unit to perform its task). In developing the Gantt's chart, we assume that the time (T) taken by each subunit to perform its task is the same; we call this the unit time. As can be seen from the figure, 13-time units are needed to finish executing 10 instructions (I1 to I10). This is to be compared to 40-time units if sequential processing is used (ten instructions each requiring four-time units).

In the following analysis, we provide three performance measures for the goodness of a pipeline. These are the Speed-up S(n), Throughput U(n), and Efficiency E(n). It should be noted that in this analysis, we assume that the unit time T = t units.

1. Speed-up S(n) Consider the execution of m tasks (instructions) using n-stages (units) pipeline. As can be seen, n + m - 1-time units are required to complete m tasks.

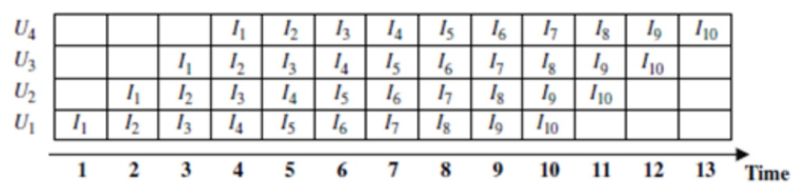


Figure 44: The space-time chart (Gantt chart)

$$Speed-up\ S(n) = \frac{Time\ using\ sequential\ processing}{Time\ using\ pipeline\ processing} = \frac{m\ xn\ x\ t}{(n+m-1)x\ t}$$

$$= \frac{m\ xn}{n+m-1}$$

$\lim_{m\to\infty} S(n) = n$ (i.e., n-fold increase in speed is theoretically possible)

2. Throughput U(n)

$$Throughput\ U(n) = \ no.of\ tasks\ executed\ per\ unit\ time = \frac{m}{(n+m-1)x\ t}$$

$$\lim_{m\to\infty} U(n) = 1\ assuming\ that\ t = 1\ unit\ time$$

3. Efficiency E(n)

$$\text{Efficiency } E(n) = \text{ Ratio of the actual speed} - \text{up to the maximum speed} - \text{up}$$

$$= \frac{\text{speed} - \text{up}}{\text{n}} = \frac{\text{m}}{\text{n} + \text{m} - 1}$$

$$\lim_{m\to\infty} E(n) = 1$$

SAQ 2

Instruction Pipeline

The simple analysis made in unit 1 ignores an important aspect that can affect the performance of a pipeline, that is, pipeline stall. A pipeline operation is said to have been stalled if one unit (stage) requires more time to perform its function, thus forcing other stages to become idle. Let us consider, for example, the case of an instruction fetch that incurs a cache miss. Let us also assume that a cache miss requires three extra time units. Figure 45 illustrates the effect of having instruction I2 incurring a cache miss (assuming the execution of ten instructions I1 to I10).

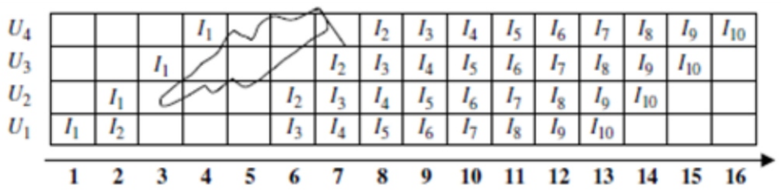


Figure 45: Effect of a cache miss on the pipeline

The figure shows that due to the extra time units needed for instruction I2 to be fetched, the pipeline stalls, that is, fetching of instruction I3 and subsequent instructions are delayed. Such situations create what is known as the pipeline bubble (or pipeline hazards). The creation of a pipeline bubble leads to wasted unit times, thus leading to an overall increase in the number of time units needed to finish executing a given number of instructions. The number of time units needed to execute the 10 instructions shown in Figure 29 is now 16-time units, compared to 13-time units if there were no cache misses.

Pipeline hazards can take place for a number of other reasons. Among these are instruction dependency and data dependency. These are explained below.

Pipeline "Stall" Due to Instruction Dependency

Correct operation of a pipeline requires that operation performed by a stage MUST NOT depend on the operation(s) performed by other stage(s). Instruction dependency refers to the case whereby fetching of an instruction depends on the results of executing a previous instruction. Instruction dependency manifests itself in the execution of a conditional branch instruction. Consider, for example, the case of a "branch if negative" instruction. In this case, the next instruction to fetch will not be known until the result of executing that "branch if negative" instruction is known. In the following discussion, we will assume that the instruction following a conditional branch instruction is not fetched until the result of executing the branch instruction is known (stored). The following example shows the effect of instruction dependency on a pipeline.

Example 1: Let us see how we can execute ten instructions I1–I10 on a pipeline consisting of four pipeline stages: IF (instruction fetch), ID (instruction decode), IE (instruction execute), and IS (instruction results store). Assume that the instruction I4 is a conditional branch instruction and that when it is executed, the branch is not taken, that is, the branch condition(s) is(are) not satisfied. Assume also that when the branch instruction is fetched, the pipeline stalls until the result of executing the branch instruction is stored. Show the succession of instructions in the pipeline; that is, show the Gantt's chart. Figure 46 shows the required Gantt's chart. The bubble created due to the pipeline stall is clearly shown in the figure.

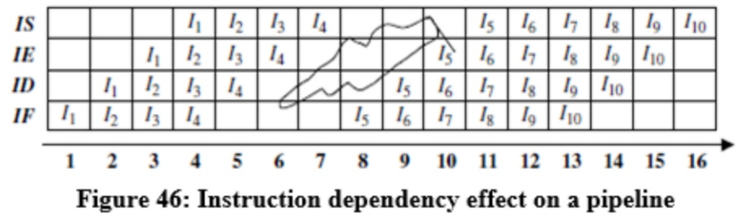


Figure 46: Instruction dependency effect on a pipeline

Pipeline "Stall" Due to Data Dependency

You should note that data dependency in a pipeline occurs when a source operand of instruction Ii depends on the results of executing a preceding instruction, Ij, i > j. It should be noted that although instruction Ii can be fetched, its operand(s) may not be available until the results of instruction Ij are stored.

Examples of Pipeline Processors

We briefly present two pipeline processors that use a variety of pipeline techniques. Our focus in this coverage is on the pipeline features of these architectures. The two processors are the ARM 1026EJ-S and the UltraSPARC III.

I). ARM 1026EJ-S Processor This processor is part of a family of RISC processors designed by Advanced RISC Machine (ARM) Company. The series is designed to suit high-performance, low-cost, and low-power embedded applications. The ARM 022EJ-S integer core has multiple execution units, thus allowing a number of instructions to exist in the same pipeline stage. It also allows the execution of simultaneous instructions. The ARM 1026EJ-S can deliver a peak throughput of one instruction per cycle.

II). UltraSPARC III Processor: The UltraSPARC III is based on the SUN SPARC-V9 RISC architectural specifications. A number of features characterize the SPARC-V9. Among these are the following:

1. Few and simple instruction formats. All instructions are 32-bit. Memory access is done exclusively using Load and Store instructions.

2. Few addressing modes. Memory addressing has only two modes, the Register + Register and the Register + Immediate modes.

3. Triadic register operands. Most instructions operate on two register operands or one register and a constant operand. The results in both cases are stored in a third register.

4. Large window register file.

SAQ 3

Instruction-Level Parallelism

Contrary to pipeline techniques, instruction-level parallelism (ILP) is based on the idea of multiple issue processors (MIP). An MIP has multiple pipelined datapaths for instruction execution. Each of these pipelines can issue and execute one instruction per cycle. Figure 3.5 shows the case of a processor having three pipes. For comparison purposes, we also show in the same figure the sequential and the single pipeline case. It is clear from the figure that while the limit on the number of cycles per instruction in the case of a single pipeline is CPI = 1, the MIP can achieve CPI < 1. In order to make full use of ILP, an analysis should be made to identify the instruction and data dependencies that exist in a given program. This analysis should lead to the appropriate scheduling of the group of instructions that can be issued simultaneously while retaining the program's correctness. Static scheduling results in the use of very long instruction word (VLIW) architectures, while dynamic scheduling results in the use of superscalar architectures. In VLIW, an instruction represents a bundle of many operations to be issued simultaneously.

You should also note that the compiler is responsible for checking all dependencies and making the appropriate groupings/scheduling of operations. This is in contrast with superscalar architectures, which rely entirely on the hardware for scheduling of instructions. Superscalar Architectures A scalar machine is able to perform only one arithmetic operation at once. A superscalar architecture (SPA) is able to fetch, decode, execute, and store the results of several instructions at the same time. It does so by transforming a static and sequential instruction stream into a dynamic and parallel one, in order to execute some instructions simultaneously. Upon completion, the SPA reinforces the original sequential instruction stream such that instructions can be completed in the original order.

In an SPA instruction, processing consists of the fetch, decode, issue, and commit stages. During the fetch stage, multiple instructions are fetched simultaneously.

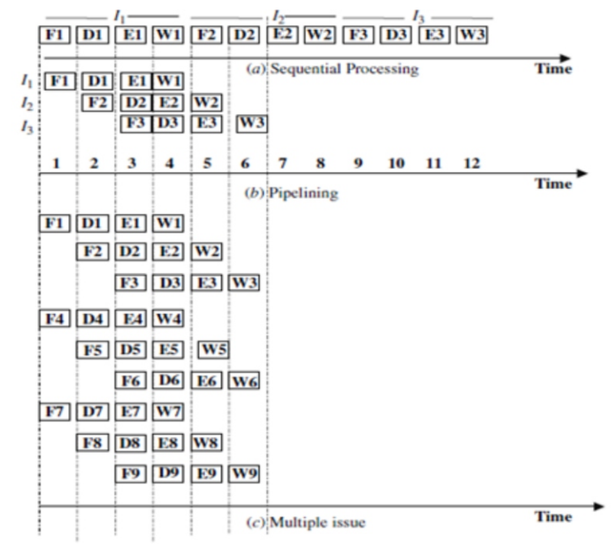


Figure 47: Multiple issues versus pipelining versus sequential processing

Branch prediction and speculative execution are also performed during the fetch stage. This is done in order to keep on fetching instructions beyond branch and jump instructions. Decoding is done in two steps. Predecoding is performed between the main memory and the cache and is responsible for identifying branch instructions.

Actual decoding is used to determine the following for each instruction:

(1) the operation to be performed;

(2) the location of the operands; and

(3) the location where the results are to be stored.

Don't forget that during the issue stage, those instructions among the dispatched ones that can start execution are identified. During the commit stage, generated values/results are written into their destination registers. The most crucial step in processing instructions in SPAs is the dependency analysis. The complexity of such an analysis grows quadratically with the instruction word size. This puts a limit on the degree of parallelism that can be achieved with SPAs such that a degree of parallelism higher than four will be impractical. Beyond this limit, the dependence analysis and scheduling must be done by the compiler. This is the basis for the VLIW approach.

Very Long Instruction Word (VLIW) In this approach, the compiler performs dependency analysis and determines the appropriate groupings/scheduling of operations. Operations that can be performed simultaneously are grouped into a very long instruction word (VLIW). Therefore, the instruction word is made long enough in order to accommodate the maximum possible degree of parallelism.

For example, the IBM DAISY machine has an instruction word that is eight operations long, called 8-issue machine. In VLIW, resource binding can be done by devoting each field of an instruction word to one and only one functional unit. However, this arrangement will lead to a limit on the mix of instructions that can be issued per cycle.

A more flexible approach is to allow a given instruction field to be occupied by different kinds of operations. For example, the Philips TriMedia machine, a 5-issue machine, has 27 functional units mapped to a 5-issue slot. In the IBM DAISY, every instruction implements a multi-way path selection scheme.

In this case, the first 72 bits of the VLIW is called the header and contain information on the tree form, condition tests, and branch targets. The header is followed by eight 23-bit parcels, each encoding an operation. In order to solve the problem of providing operands to a large number of functional units, the IBM DAISY keeps eight identical copies of the same register file, one for each of the eight functional units.

Summary

In this unit, you have learnt that:

- Pipelining is a technique by which a given task is divided into a number of subtasks that need to be performed in sequence.
- The compiler is responsible for checking all dependencies and making the appropriate groupings/scheduling of operations.

Self-Assessment Questions

SAQ

1. What is pipelining in computer architecture?
2. Define instruction pipeline.
3. Explain the instruction-level parallelism.

Tutor Marked Assessment

- Write short notes on instruction-level parallelism
- What is instruction pipeline?
- Explain the instruction-level parallelism found in computer architecture

Further Reading

- Buyya, R. High Performance Cluster Computing: Architectures and Systems (1999). Upper Saddle River, NJ: Prentice Hall, 1999.
- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- Buyya, R. High Performance Cluster Computing: Architectures and Systems. Upper Saddle River, NJ: Prentice Hall, 1999.
- Mano, M. Morris, (1993). Computer System Architecture (3rd ed). Prentice Hall of India.
- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/ http://ece-www.colorado.edu /faculty/h euring.html

23 | **Picture:** Computer

Photo: Pixels.com

UNIT 2

Introduction to Reduced Instruction Set Computers

Introduction

This unit is dedicated to a study of reduced instruction set computers (RISCs). These machines represent a noticeable shift in the computer architecture paradigm. This paradigm promotes simplicity rather than complexity. The RISC approach is substantiated by a number of studies indicating that assignment statements, conditional branching, and procedure calls/return represent more than 90% and that complex operations such as long division represent only about 2% of the operations performed in a typical set of benchmark programs. These studies also showed that among all operations, procedure calls/returns are the most time-consuming. Based on such results, the RISC approach calls for enhancing architectures with the resources needed to make the execution of the most frequent and the most time-consuming operations most efficient.

The seed for the RISC approach started as early as the mid-1970s. Its real-life manifestation appeared in the Berkeley RISC-I and the Stanford MIPS machines, which were introduced in the mid-1980s. Today, RISC-based machines are a reality and are characterized by a number of common features such as simple and reduced instruction set, fixed instruction format, one instruction per

machine cycle, pipeline instruction fetch/execute units, ample number of general-purpose registers (or optimized compiler code generation), Load/Store memory operations, and hardwired control unit design. Our coverage in this unit starts with a discussion on the evolution of RISC architectures. We then provide a brief discussion on some of the performance studies that led to the adoption of the RISC paradigm. Overlapped Register Windows, an essential concept in RISC development, is then discussed. Toward the end of the unit, we provide details on a number of RISC-based architectures, such as the Berkeley RISC, the Stanford MIPS, the Compaq Alpha, and the SUN UltraSparc.

Learning Outcomes

At the end of this unit, you should be able to:

1. Explain the RISC/CISC evolution cycle.
2. Explain RISCs design principles
3. Explain the difference between RISCs and CISCs
4. Explain the overlapped register windows.

SAQ 1 RISC/CISC Evolution Cycle

2 mins

Have at the back of your mind that the term RISCs stands for Reduced Instruction Set Computers. It was originally introduced as a notion to mean architectures that can execute as fast as one instruction per clock cycle. RISC started as a notion in the mid-1970s and has eventually led to the development of the first RISC machine, the IBM 801 minicomputer. The launching of the RISC notion announces the start of a new paradigm in the design of computer architectures. This paradigm promotes simplicity in computer architecture design. In particular, it calls for going back to basics rather than providing extra hardware support for high-level languages. This paradigm shift relates to what is known as the semantic gap, a measure of the difference between the operations provided in the high-level languages (HLLs) and those provided in computer architectures.

It is recognized that the wider the semantic gap, the larger the number of undesirable consequences. These include (a) execution inefficiency, (b) excessive machine program size, and (c) increased compiler complexity. Because of these expected consequences, the

conventional response of computer architects has been to add layers of complexity to newer architectures. These include increasing the number and complexity of instructions together with increasing the number of addressing modes. The architectures resulting from the adoption of this "add more complexity" are now known as Complex Instruction Set Computers (CISCs).

However, it soon became apparent that a complex instruction set has some disadvantages, which include a complex instruction decoding scheme, an increased size of the control unit, and increased logic delays. These drawbacks prompted a team of computer architects to adopt the principle of "less is actually more." Several studies were then conducted to investigate the impact of complexity on performance. These are discussed below.

SAQ 2 RISCs Design Principles

3 mins

I want you to know that a computer with the minimum number of instructions has the disadvantage that a large number of instructions will have to be executed in realizing even a simple function. This will result in a speed disadvantage. On the other hand, a computer with an inflated number of instructions has the disadvantage of complex decoding and hence a speed disadvantage. It is then natural to believe that a computer with a carefully selected reduced set of instructions should strike a balance between the above two design alternatives.

The question then becomes, what constitutes a carefully selected reduced set of instructions? In order to arrive at an answer to this question, it is necessary to conduct in-depth studies on some aspects of computation. These aspects should include (a) operations that are most frequently performed during execution of typical (benchmark) programs, (b) operations that are the most time consuming, and (c) the type of operands that are most frequently used. Many early studies were conducted to find out the typical breakdown of operations that are performed in executing benchmark programs. A careful look at the estimated percentage of operations performed reveals that assignment statements, conditional branches, and procedure calls constitute about 90% of the total operations performed, while other operations, however complex they may be, make up the remaining 10%.

In addition to the above findings, studies on time–performance characteristics of operations revealed that among all operations, procedure calls/returns are the most time-consuming. With regards to the type of operands used during typical computation, it was noticed that the majority of references (no less than 60%) are made to simple scalar variables and that no less than 80% of scalars are local variables to procedures.

The above observations about typical program behaviour have led to the following conclusions:

1. A simple movement of data (represented by assignment statements), rather than complex operations, are substantial and should be optimized.

2. Conditional branches are predominant, and therefore careful attention should be paid to the sequencing of instructions. This is particularly true when it is known that pipelining is indispensable to use.

3. Procedure calls/returns are the most time-consuming operations. Therefore, a mechanism should be devised to make the communication of parameters among the calling and the called procedures cause the least number of instructions to execute.

4. A prime candidate for optimization is the mechanism for storing and accessing local scalar variables.

The above conclusions have led to the argument that instead of bringing the instruction set architecture closer to HLLs, it should be more appropriate to rather optimize the performance of the most time-consuming features of typical HLL programs. This is obviously a call for making the architecture simpler rather than complex. Remember, that complex operations such as long division represent only a small portion (less than 2%) of the operations performed during a typical computation. One then should ask the question: how can we achieve that? The answer is by (a) keeping the most frequently accessed operands in CPU registers and (b) minimizing the register-to-memory operations.

The above two principles can be achieved using the following mechanisms:

1. Use a large number of registers to optimize operand referencing and reduce the processor memory traffic.

2. Optimize the design of instruction pipelines such that minimum compiler code generation can be achieved.

3. Use a simplified instruction set and leave out those complex and unnecessary instructions.

The following two approaches were identified to implement the above three mechanisms.

1. Software approach. Use the compiler to maximize register usage by allocating registers to those variables that are used the most in a given time period (this is the philosophy adopted in the Stanford MIPs machine).

2. Hardware approach. Use ample CPU registers so that more variables can be held in registers for

larger periods of time (this is the philosophy adopted in the Berkeley RISC machine). The hardware approach necessitates the use of a new register organization, called an overlapped register window. This is explained below.

SAQ 4

Overlapped Register Windows

3 mins

The main idea behind the use of register windows is to minimize memory accesses. In order to achieve that, a large number of CPU registers are needed. For example, the number of CPU general-purpose registers available in the original SPARC machine (one of the earliest RISCs) was 120. However, it is desirable to have only a subset of these registers visible at any given time and to have them addressed as if they were the only set of registers available. Therefore, CPU registers are divided into multiple small sets, each assigned to a different procedure. A procedure call will automatically switch the CPU to use a different fixed-size window of registers.

In order to minimize the actual movement of parameters among the calling and the called procedures, each set of registers is divided into three subsets: parameter registers, local registers, and temporary registers. When a procedure call is made, a new overlapping window will be created such that the temporary registers of the caller are physically the same as the parameter registers of the called procedure. This overlap allows parameters to be passed among procedures without the actual movement of data (Fig. 48).

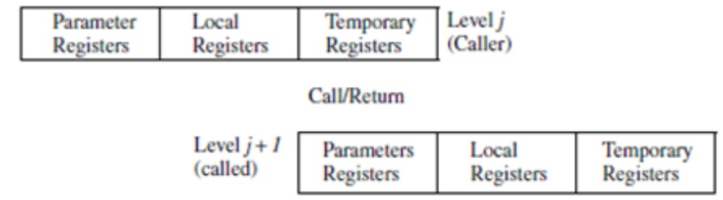


Figure 48: Register window overlapping

In addition, a set of a fixed number of CPU registers are identified as global registers and are available to all procedures. For example, references to registers 0 through 7 in the SPARC architecture refer to unique global registers, and references to registers 8 through 31 indicate registers in the current window. The current window is pointed to using what is usually called the current window pointer (CWP). Upon having all windows filled, the register window wraps around, thus acting like a "circular buffer."

It should be noted that a study was conducted in 1985 to find out the impact of using a register window on the performance of the Berkeley RISC. In this study, two versions of the machine were studied. The first was designed with register windows, and the second was a hypothetical Berkeley RISC implemented without windows. The results of the study indicated a decrease by a factor of 2 to 4 (depending on specific benchmark) in the memory traffic due to the use of register windows.

SAQ 3

RISCs versus CISCs

4 mins

The choice of RISC versus CISC depends totally on the factors that must be considered by a computer designer. These factors include size, complexity, and speed. A RISC architecture has to execute more instructions to perform the same function performed by a CISC architecture. To compensate for this drawback, RISC architectures must use the chip area saved by not using complex instruction decoders in providing a large number of CPU registers, additional execution units, and instruction caches. The use of these resources leads to a reduction in the traffic between the processor and the memory.

On the other hand, a CISC architecture with richer and more complex instructions will require a smaller number of instructions than its RISC counterpart. However, a CISC architecture requires a complex decoding scheme and hence is subject to logic delays. It is, therefore, reasonable to consider that the RISC and CISC paradigms differ primarily in the strategy used to trade-off different design factors.

There is very little reason to believe that an idea that improves performance for a RISC architecture will fail to do the same thing in a CISC architecture and vice versa. For example, one key issue in RISC development is the use of optimizing the compiler to reduce the complexity of the hardware and to optimize the use of CPU registers. These same ideas should be applicable to CISC compilers.

Table 7: RISC versus CISC Performance

Application	MIPS CPI (RISC)	VAX CPI (CISC)	CPI ratio	Instruction ratio
Spice 2G6	1.80	8.02	4.44	2.48
Matrix300	3.06	13.81	4.51	2.37
Nasa 7	3.01	14.95	4.97	2.10
Espresso	1.06	5.40	5.09	1.70

Increasing the number of CPU registers could very much improve the performance of a CISC machine. This could be the reason behind not finding a pure commercially available RISC (or CISC) machine. It is not unusual to see a RISC machine with complex floating-point instructions. It is equally expected to see CISC machines making use of the register windows RISC idea. In fact, there have been studies indicating that a CISC machine such as the Motorola 680xx with a register window will achieve a 2 to 4 times decrease in the memory traffic.

This is the same factor that can be achieved by a RISC architecture, such as the Berkeley RISC, due to the use of a register window. It should, however, be noted that most processor developers (except for Intel and its associates) have opted for RISC processors. Computer system manufacturers such as Sun Microsystems are using RISC processors in their products. However, for compatibility with the PC-based market, such companies are still producing CISC-based products.

Tables 7 and 8 show a limited comparison between an example RISC and CISC machine in terms of performance and characteristics, respectively. An elaborate comparison among a number of commercially available RISC and CISC machines is shown in Table 10.5. It is worth mentioning at this point that the following set of common characteristics among RISC machines is observed:

1. Fixed-length instructions
2. A limited number of instructions (128 or less)
3. A limited set of simple addressing modes (minimum of two: indexed and PC-relative)
4. All operations are performed on registers; no memory operations
5. Only two memory operations: Load and Store
6. Pipelined instruction execution
7. A large number of general-purpose registers or the use of advanced compiler technology to optimize register usage
8. One instruction per clock cycle
9. Hardwired control unit design rather than microprogramming

Table 8: RISC versus CISC Characteristics

Characteristic	VAX-11 (CISC)	Berkeley RISC-1 (RISC)
Number of instructions	303	31
Instruction size (bits)	16-456	32
Addressing modes	22	3
No. general purpose registers	16	138

Table 9: Summary of features of a Number of RISC and a CISC

	Motorola 88110	Alpha AXP 21264	Pentium	Power PC 601
Company	Motorola	Compaq (DEC)	Intel	IBM
Architecture	RISC	RISC	CISC	RISC
# Registers(I)	32	80	64	32
Cache I/D	8/8 KB	64/64 KB	8/8 KB	32
# Registers (GP/FP)	32/32	31/31	8/8	32/32
# Inst/cycle	2	1	2	3
# Pipelines (I/FP)	NS	4/2	5/8	4/6
Multiprocessing Support	No	Yes	Yes	Yes

Pioneer (University) RISC Machines

5 mins

In this section, we present brief descriptions of the main architectural features of two pioneer university-introduced RISC machines. The first machine is the Berkeley RISC, and the second is the Stanford MIPS machine. These machines are presented as a means to show how original RISC machines look and also to make the reader appreciate the advances made in RISC machines' development since their inception.

The Berkeley RISC

There are two Berkeley RISC machines: RISC-I and RISC-II. Unless otherwise mentioned, we refer to RISC-I in our discussion. RISC is a 32-bit LOAD/STORE architecture. There are 138 32-bit registers R0–R137 available to the users. The first ten registers R0–R9 are global registers (seen by all procedures). Register R0 is used to synthesize addressing modes and operations

that are not directly available on the machine. Registers R10–R137 are divided into an overlapping register window scheme with 32 registers visible at any instant. A 5-bit variable, called current window pointer (CWP), is used to point to the current register set.

All RISC instructions occupy a full word (32 bits). The RISC instruction set is divided into four categories. These are ALU (a total of 12 instructions), Load/Store (a total of 16 instructions), Branch & Call (a total of seven instructions), and special instructions (a total of four instructions). Some examples of the RISC instructions are:

1. ALU: ADD R_s, S, R_d; $R_d \leftarrow R_s + S$
2. Load/Store: LDXW $(R_x)S$, R_d; $R_d \leftarrow M[R_x + S]$

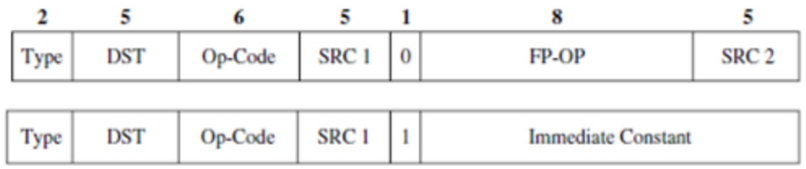


Figure 49: Three operand instruction formats used in RISC

3. Branch & Call: JMPX COND, $(R_x)S$; $PC \leftarrow R_x + S$; where COND is a condition
4. Special Instructions: GETPSW R_d; $R_d \leftarrow$PSW

All arithmetic and logical instructions have three operands and have the form Destination: = source1 op source2 (Fig. 32). The LOAD and STORE instructions may use either of the indicated formats with DST being the register to be loaded or stored. The low order 19 bits of the instructions are used to determine the effective address.

Instructions load and store 8-, 16-, 32-, and 64-bit quantities into 32-bit registers.

Two methods are provided for calling procedures. The CALL instruction uses a

30-bit PC relative offset (Fig. 32). The JMP instruction uses any of the instruction formats used for arithmetic and logical operations and allows the return address to be put in any register. RISC uses a three-address instruction format with the availability of some two and one-address instructions. There are only two addressing modes. These are indexed mode and PC relative modes. The indexed mode can be used to synthesize three other modes. These are base-absolute (direct), register indirect, and indexed for linear byte array modes. RISC uses a static two-stage pipeline: fetch and execute.

You should note that the floating-point unit (FPU) contains thirty-two 32-bit registers to hold 32 single-precision (32-bit) floating-point operands, 16 double-precision (64-bit) operands, or eight extended-precision (128-bit) operands. The FPU can execute about 20 floating-point instructions, most of them in a single-, double-, or extended-precision using the first instruction format used for arithmetic. In addition to instructions for loading and storing FPUs registers, the CPU can also test FPUs registers and branch conditionally on results. RISC employs a conventional MMU supporting a single-paged 32-bit address space. The RISC four-bus organization is shown in Figure 33. 34 Stanford MIPS (Microprocessor Without Interlock Pipe Stages)

MIPS is a 32-bit pipelined LOAD/STORE machine. It uses a five-stage pipeline consisting of Instruction Fetch (IF), Instruction Decode (ID), Operand Decode

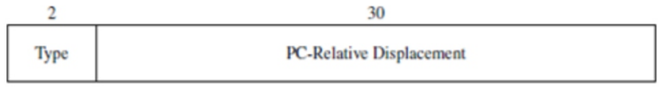


Figure 50: Procedure call instruction in RISC

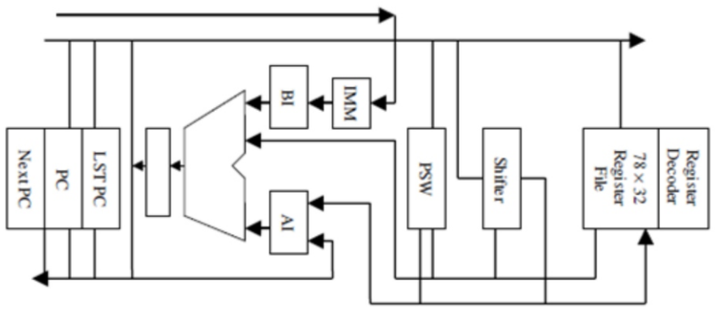


Figure 51: RISC four-bus organization

(OD), Operand Store/Execution (OS/EX), and Operand Fetch (OF). The first three stages perform instruction fetch, instruction decode respectively, and operand fetch. The OS/EX stage sends operand to memory in the case of a store instruction or uses the ALU in case of instruction execution. The OF stage receives the operand in case of a load instruction. MIPS uses a mechanism called pipeline interlock in order to prevent an instruction from continuing until the needed operand is available.

Unlike the Berkeley RISC, MIPS has a single set of sixteen 32-bit general-purpose registers. The MIPS compiler optimizes the use of registers in whatever way is best for the program currently being compiled. In addition to the 16 general-purpose registers, MIPS provided four additional registers in order to hold the four previous PC values (to support backtracking and restart in

case of a fault). A fifth register is used to hold the future PC value (to support branch instructions).

Four addressing modes are used in MIPS. These are immediate, indexed, based with offset, and base shifted. Four instruction groups were identified in MIPS. These are ALU, Load/Store, Control, and Special instructions. A total of 13 ALU instructions were provided. These include all register-to-register two- or three operand formats (Fig. 34). A total of 10 LOAD/STORE instructions were provided. They use 16 or 32 bits. In the latter case, indexed addressing is used by adding a 16-bit signed constant to a register using the second format in Figure 34. A total of six control flow instructions were provided.

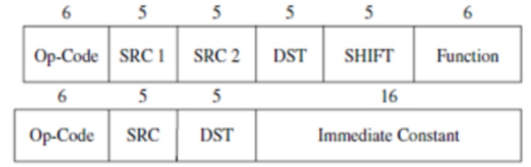


Figure 52: Three-operand instruction used in MIPS

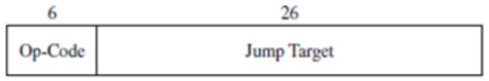


Figure 53: Jump instruction format used in MIPS

These include jumps, relative jumps, and compare instructions. Only two special flow instructions were provided. They support procedure and interrupt linkage. Some examples of MIPS instructions are:

1. ALU: Add src_1, src_2, dst; dst ←src_1 + src_2
2. Load/Store: Ld [src_1 + src_2], dst; dst ←M[src_1 + src_2]
3. Control: Jmp dst; PC ←dst
4. Special Function: SavePC A; M[A] ←PC

MIPS does not provide direct support for floating-point operations. Floating point operations are to be done by a specialized coprocessor. Surprisingly, non-RISC instructions such as MULT and DIV were included, and they use special functional units. The contents of two registers can be multiplied or divided, and the 64-bit product is kept in two special registers LO and HI.

Procedure calls can be made through the JUMP instruction shown in Figure 36.

The instruction uses a 26-bit jump target address.

The MIPS virtual address is 32 bits long, thus allowing for up to four Gwords virtual address space. A virtual address is divided into a 20-bit virtual page number and a 12-bit offset within the page. The actual implementation of MIPS was restricted by packaging constraints allowing only 24 address pins; that is, the actual physical address space is 224 = 16 Mwords (32 bits each). A support for off-chip TLB for address translation is provided. The MIPS organization is shown in Figure 43.

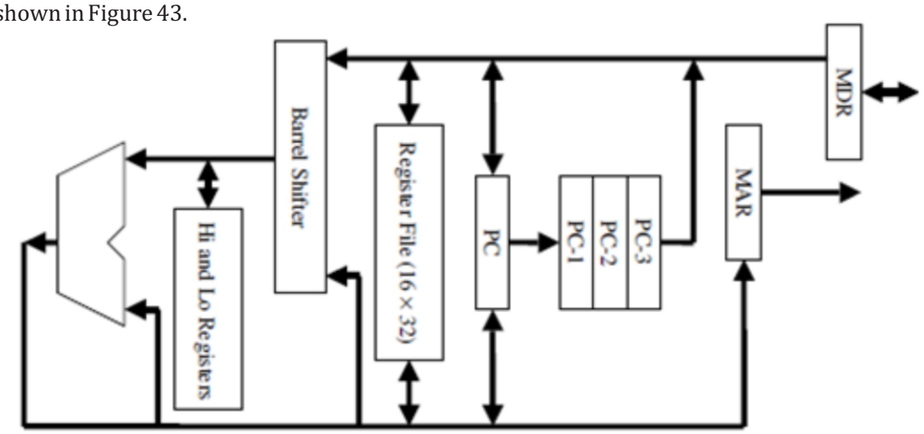


Figure 54: MIPS organization

Example of Advanced RISC Machines

In this section, we introduce two representative advanced RISC machines. Our emphasis in this coverage is on the pipeline features and the branch handling mechanisms used.

Activity 1: Reduced Instruction Set Computers	
Task	**Question**
You had a dream to design a machine that represent a noticeable shift in the computer architecture paradigm	
	Explain the RISC/CISC evolution cycle?
	Differentiate between the RISC and CISC?
	Discuss the overlapped register windows?

Compaq (Formerly DEC) Alpha 21264

Are you aware that Alpha 21264 (EV6) is a third-generation Compaq (formerly DEC) RISC superscalar processor. It is a full 64-bit processor. The 21264 has an 80-entry integer register file and a 72-entry floating-point register file and employs a two-level cache. The L1 data and instruction caches are 64 KB each and are organized in a two-way set-associative manner. The L2 data cache can be 1 to 16 MB (shared by instructions and data) organized using direct-mapping. The block size is 64 bytes.

The data cache can receive any combination of two loads or stores from the integer execution pipe every cycle. This is equivalent to having the 64 KB on-chip data cache delivering 16 bytes every cycle, hence twice the clock speed of the processor. The 21264 memory system can support up to 32 in-flight loads, 32 in-flight stores, and 8 in-flight (64 bytes) cache block fills and 8 cache misses.

Summary

Thus far. I had explained that:

- A computer with the minimum number of instructions has the disadvantage that a large number of instructions will have to be executed in realizing even a simple function.
- A computer with an inflated number of instructions has the disadvantage of complex decoding and hence a speed disadvantage.
- The main idea behind the use of register windows is to minimize memory accesses.
- RISC architecture has to execute more instructions to perform the same function performed by a CISC architecture.

Self-Assessment Questions

1. What is a RISC/CISC evolution cycle?
2. What are RISCs design principles?
3. State three (3) differences between RISCs and CISCs
4. Define overlapped register windows.

Tutor Marked Assessment

- A. What are the factors that the choice of RISC versus CISC depends totally on?

Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- Buyya, R. High Performance Cluster Computing: Architectures and Systems. Upper Saddle River, NJ: Prentice Hall, 1999.
- Mano, M. Morris, (1993).Computer System Architecture (3rd ed). Prentice Hall of India.
- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/ http://ece-www.colorado.edu /faculty/h euring.html

24 | **Picture:** Multiprocessor

Photo: Wikipedia.com

UNIT 3

Introduction to Multiprocessors

Introduction

Having covered the essential issues in the design and analysis of uniprocessors and pointing out the main limitations of a single-stream machine, we begin in this unit to pursue the issue of multiple processors. Here a number of processors (two or more) are connected in a manner that allows them to share the simultaneous execution of a single task. The main argument for using multiprocessors is to create powerful computers by simply connecting many existing smaller ones.

Our coverage in this unit starts with a section on the general concepts and terminology used. We then point to the different topologies used for interconnecting multiple processors. Different classification schemes for computer architectures are then introduced and analyzed. We then introduce a topology-based taxonomy for interconnection networks. Two memory-organization schemes for MIMD (multiple instructions multiple data) multiprocessors are also introduced. Our coverage in this unit ends with a touch on the analysis and performance metrics for multiprocessors.

At the end of this unit, you should be able to:

Learning Outcomes

1. Explain the multiprocessor system.
2. Explain the classification of computer architectures
3. Explain the limits of multiprocessor hardware.

SAQ 1

What is a Multiprocessor System?

7 mins

A multiple processor system consists of two or more processors that are connected in a manner that allows them to share the simultaneous (parallel) execution of a given computational task. Parallel processing has been advocated as a promising approach for building high-performance computer systems. Two basic requirements are inevitable for the efficient use of the employed processors.

These requirements are:

(1) low communication overhead among processors while executing a given task and

(2) a degree of inherent parallelism in the task. A number of communication styles exist for multiple processor networks. These can be broadly classified according to (1) the communication model (CM) or (2) the physical connection (PC).

According to CM, networks can be further classified as:

(1) multiple processors (single address space or shared memory computation) or

(2) multiple computers (multiple address space or message passing computation). According to PC, networks can be further classified as (1) bus-based or (2) network-based multiple processors.

Notice that the organization and performance of a multiple processor system are greatly influenced by the interconnection network used to connect them. On the one hand, a single shared bus can be used as the interconnection network for multiple processors. On the other hand, a crossbar switch can be used as the interconnection network. While the first technique represents a simple easy-to-expand topology, it is, however, limited in performance since it does not allow more than one processor/memory transfer at any given time.

You should also note that the crossbar provides full processor/memory distinct connections, but it is expensive. Multistage interconnection networks (MINs) strike a balance between the limitation of the single, shared bus system and the expense of a crossbar-based system. In a MIN, more than one processor/memory connection can be established at the same time. The cost of a MIN can be considerably less than that of a crossbar, particularly for a large number of processors and/or memories. The use of multiple buses to connect multiple processors to multiple memory modules has also been suggested as a compromise between the limited single bus and the expensive crossbar.

SAQ 2

Classification of Computer Architectures

Classification means to order a number of objects into categories, each having common features, among which certain relationship(s) exist(s). In this regard, a classification scheme for computer architectures aims at categorizing them such that those architectures that have common features fall into one category and such that different categories represent distinct groups of architectures.

In addition, a classification scheme for computer architecture should provide a basis for information ordering and a basis for predicting the features of a given architecture. Two broad schemes exist for computer architecture classification. The first is based on external (morphological) features of architectures and the second is based on the evolutionary features of architectures. The first scheme emphasizes the finished form of architectures, while the second scheme emphasizes the way a particular architecture has been derived from its predecessor and suggests speculative views on its successor. Morphological classification provides a basis for predictive power, while evolutionary classification provides a basis for a better understanding of architectures. Examining the extent to which a classification scheme is satisfying its stated objective(s) could assess the pros and cons of that scheme.

A number of classification schemes have been proposed over the last three decades. These include the Flynn's classification (1966), the Kuck (1978), the Hwang and Briggs (1984), the Erlangen (1981), the Giloi (1983), the Skillicorn (1988), and the Bell (1992). A number of these are briefly discussed below.

Flynn's Classification

Take note that Flynn's classification scheme is based on identifying two orthogonal streams on a computer. These are the instruction and data streams. The instruction stream is defined as the sequence of instructions performed by the computer. The data stream is defined as the data traffic exchanged between the memory and the processing unit. According to Flynn's classification, either of the instruction or data streams can be single or multiple. This leads to four distinct categories of computer architectures:

1. Single-instruction single-data streams (SISD)
2. Single-instruction multiple-data streams (SIMD)
3. Multiple-instruction single-data streams (MISD)

Observations on Flynn's Classification

1. Flynn's classification is among the first of its kind to be introduced, and as such, it must have inspired subsequent classifications.

2. The classification helped in categorizing architectures that were available and those that have been introduced later. For example, the introduction of the SIMD and MIMD machine models in the classification must have inspired architects to introduce these new machine models.

3. The classification stresses the architectural relationship at the memory processor level. Other architectural levels are totally overlooked.

4. The classification stresses the external (morphological) features of architectures.

   No information is included in the revolutionary relationship of architectures that belong to the same category.

5. Owing to its pure abstractness, no practically viable machine has exemplified the MISD model introduced by the classification (at least so far). It should, however, be noted that some architects have considered pipelined machines (and perhaps systolic-array computers) as examples for MISD.

6. A very important aspect that is lacking in Flynn's classification is the issue of machine performance. Although the classification gives the impression that machines in the SIMD and the MIMD are superior to their SISD and MISD counterparts, it gives no information on the relative performance of SIMD and MIMD machines.

Kuck Classification Scheme

Flynn's taxonomy can be considered a general classification that has been extended by a number of computer architects. One such extension is the classification introduced

by D. J. Kuck in 1978. In his classification, Kuck extended the instruction stream further to single (scalar and array) and multiple (scalar and array) streams. The data stream in Kuck's classification is called the execution stream and is also extended to include single (scalar and array) and multiple (scalar and array) streams. Our main observation is that both Flynn's and Kuck's classifications cover the entire architecture space. However, while Flynn's classification

emphasizes the description of architectures at the instruction set level, Kuck's classification emphasizes the description of architectures at the hardware level.

Hwang and Briggs Classification Scheme

The main new contribution of the classification due to Hwang and Briggs is the introduction of the concept of classes. This is a further refinement of Flynn's classification. For example, according to Hwang and Briggs, the SISD category is further refined into two subcategories: single functional unit SISD (SISD-S) and multiple functional units SISD (SISD-M).

Erlangen Classification Scheme

In its simplest form, this classification scheme adds one more level of details to the internal structure of a computer compared to Flynn's scheme. In particular, this scheme considers that in addition to the control (CNTL) and processing (ALU) units, a third subunit called the elementary logic unit (ELU) can be used to characterize a given computer architecture.

The ELU represents the circuitry required to perform the bit-level processing within the ALU. An architecture is characterized using a three-tuple system (k, d, w) such that k = number of CNTLs, d = number of ALU units associated with one control unit, and w = number of ELUs per ALU (the width of a single data word).

For example, in one of its models, the ILLAC-IV was made up of a mesh connected array of 64 64-bit ALUs controlled by a Burroughs B6700 computer. According to Erlangen, this model of the ILLAC-IV is characterized as (1, 64, 64).

Postulating that pipelining can exist at all three levels of hardware processing, the classification includes three additional parameters. These are w' = the number of pipeline stages per ALU, d' = the number of functional units per ALU, and k' = the number of ELUs forming the control unit. Given the expected multi-unit nature of each of the three hardware processing levels, a more general six-tuple can be used to characterize an architecture as follows: (k x k', d x d', w x w').

Figure 39 illustrates the Erlangen classification system. More complex systems can still be characterized using the Erlangen system by using two additional operators, the AND operator, denoted by x, and the

ALTERNATIVE operator denoted as V. For example, an architecture consisting of two computational subunits each having a six-tuple $(k_0xk'_0, d_0xd'_0x\ w_0xw'_0)$ and $(k_1xk'_1, d_1xd'_1x\ w_1xw'_1)$ is characterized using both subunits as $(k_0xk'_0, d_0xd'_0x\ w_0xw'_0)$ x $(k_1xk'_1, d_1xd'_1x\ w_1xw'_1)$, while an architecture that can be expressed using of the two subunits is characterized as $<k_0xk'_0, d_0xd'_0x\ w_0xw'_0 > V <k_1xk'_1, d_1xd'_1x\ w_1xw'_1 >$.

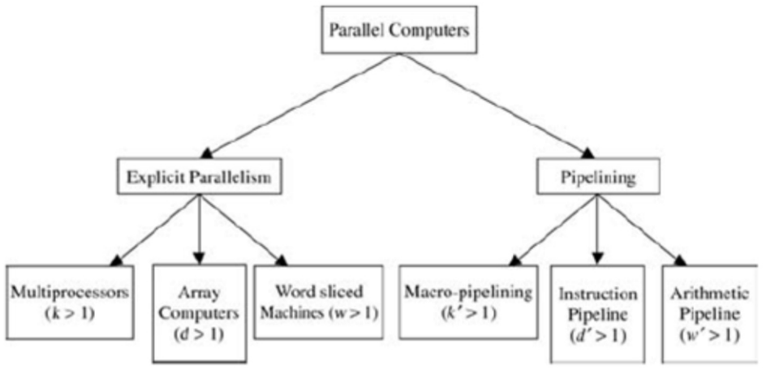


Figure 55: The Edangen classification scheme

For example, a later design of the ILLAC-IV consisted of two DEC PDP-10 as the front-end controller, where data can only be accepted from one PDP-10 at a time. This version of the ILLAC-IV can be characterized as (2, 1, 36) x (1, 64, 64). Since the ILLAC-IV can now also work in a half-word mode, whereby there are 128 32-bit processors, rather than the 64 64-bit processors, an overall characterization of the ILLAC-IV is given by (2, 1, 36) x [(1, 64, 64) x (1, 128, 32)].

As can be seen, this classification scheme can be regarded as a hierarchical classification that puts more emphasis on the internal structure of the processing hardware. It does not provide any basis for the classification and/or grouping of computer architectures. In particular, the classification overlooks the interconnection among different units.

Skillicorn Classification Scheme

Owing to its inherent nature, Flynn's classification may end up grouping computer systems with similar architectural characteristics but with diverse functionality into one class. This same observation has been the main motive behind the Skillicorn classification introduced in 1988. According to this classification, an abstract von

The Neumann machine is modelled as shown in Figure 11.5. As can be seen, the abstract model includes two memory subdivisions, instruction memory (IM) and data memory (DM), in addition to the instruction processor (IP) and the data processor (DP). In developing the

classification scheme, the following possible interconnection relationships were considered: (IP–DP), (IP–IM), (DP–DM), and (DP–IP). The interconnection scheme takes into consideration the type and number of connections among the data processors, data memories, instruction processors, and instruction memories. There may exist no, one-to-many, and many-to-many such connections.

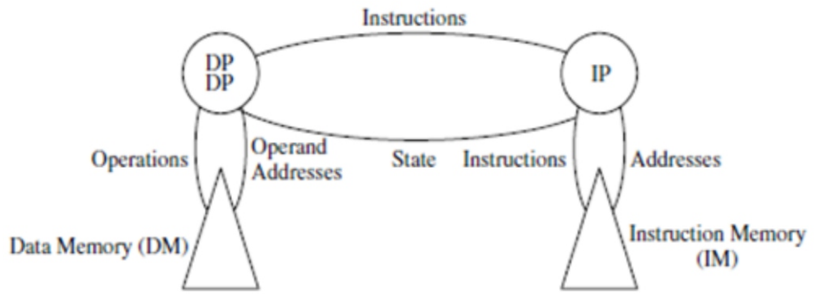


Figure 56: Abstract model of a sample machine

Using the given connection schemes, Skillicorn arrived at 28 different classes. Figure 40 illustrates four example classes according to the classification. Major advantages of the Skillicorn classification include (1) simplicity, (2) the proper consideration of the interconnectivity among units, (3) flexibility, and (4) the ability to represent most current computer systems.

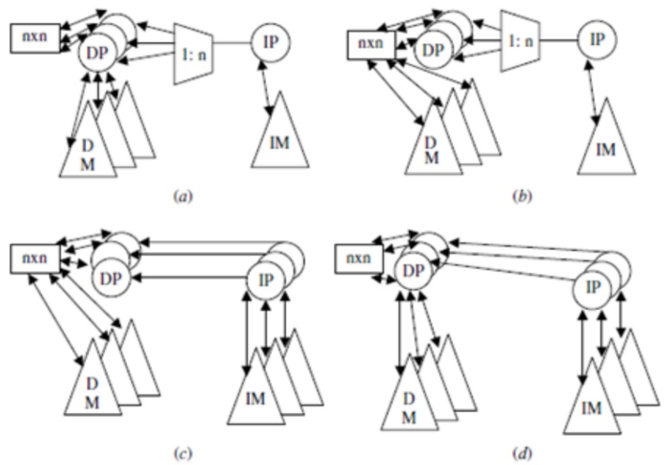


Figure 57: Example connection classes (a) Array 1 class (b) array 2 class (c) tightly coupled multiprocessor (d) loosely coupled multiprocessor

SAQ 3 However, the classification

(1) lacks the inclusion of operational aspects such as pipelining and

(2) has difficulty in predicting the relative power of machines belonging to the same class without explicit knowledge of the interconnection scheme used in that class.

Multiple processor systems can be further classified as tightly coupled versus loosely coupled. In a tightly coupled system, all processors can equally access global memory. In addition, each processor may also have its own local or cache memory. In a loosely coupled system, the memory is divided among processors such that each processor will have its own memory attached to it.

However, processors still share the same memory address space. Any processor can directly access any remote memory. Examples of tightly coupled multiple processors include the CMU C.mmp, Encore Computer Multimax, and the Sequent Corp. Balance series. Examples of loosely coupled multiple processors include CMU Cm x, the BBN Butterfly, and the IBM RP3.

Summary

In this unit, you have learnt that:

- A multiple processor system consists of two or more processors that are connected in a manner that allows them to share the simultaneous (parallel) execution of a given computational task.
- A multiprocessor is expected to reach a faster speed than the fastest uniprocessor.

Self-Assessment Questions

1. What is a multiprocessor system?
2. State two (2) classifications of computer architectures
3. What are the limits of multiprocessor hardware?

Tutor Marked Assessment

- In executing tasks (programs) using a multiprocessor, it may be assumed that a given task can be divided into n equal subtasks, each of which can be executed by one processor.

Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- Buyya, R. High Performance Cluster Computing: Architectures and Systems. Upper Saddle River, NJ: Prentice Hall, 1999.
- Mano, M. Morris, (1993). Computer System Architecture (3rd ed). Prentice Hall of India.
- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/ http://ece-www.colorado.edu /faculty/h euring.html

25 | **Picture:** Uniprocessor

Photo: Wikipedia.com

UNIT 4

SIMD and MIMD Schemes

Introduction

A multiprocessor is expected to reach a faster speed than the fastest uniprocessor. Also, a multiprocessor consisting of some single uniprocessors is expected to be more cost-effective than building a high-performance single processor. An additional advantage of a multiprocessor consisting of n processors is that if a single processor fails, the remaining fault-free n - 1 processors should be able to provide continued service, albeit with degraded performance.

Our coverage in this unit starts with a section on the general concepts and terminology used. We then point to the different topologies used for interconnecting multiple processors. Different classification schemes for computer architectures are then introduced and analyzed. We then introduce a topology-based taxonomy for interconnection networks. Two memory-organization schemes for MIMD (multiple instructions multiple data) multiprocessors are also introduced. Our coverage in this unit ends with a touch on the analysis and performance metrics for multiprocessors.

Learning Outcomes

At the end of this unit, you should be able to:

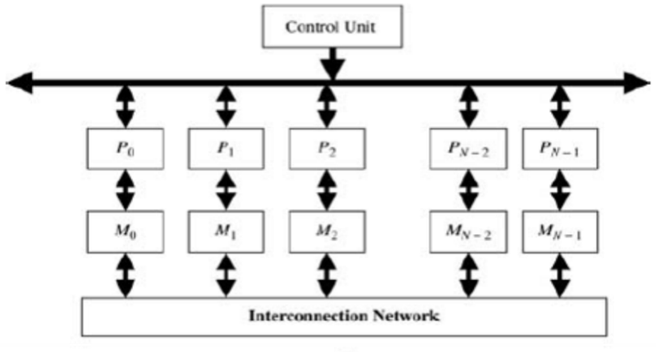
1. Explain SIMD Schemes.
2. Explain the MIMD Schemes
3. Explain the Shared Memory Organization.
3. Explain issues surrounding the construction of multiprocessor synchronization primitives.

SAQ 1 SIMD Schemes

10 mins

Can you recall that Flynn's classification results in four basic architectures? Among those, the SIMD and the MIMD are frequently used in constructing parallel architectures. In this section, we will provide basic information on the SIMD paradigm. It is important at the outset to indicate that SIMDs are mostly designed to exploit the inherent parallelism encountered in matrix (array) operations, which are required in applications such as image processing. Famous real-life machines that have been commercially constructed include the ILLIAC-IV (1972), the STARAN (1974), and the MPP (1982).

Two main SIMD configurations have been used in real-life machines. These are shown in Figure 4.16. In the first scheme, each processor has its own local memory. Processors can communicate with each other through the interconnection network. If the interconnection network does not provide a direct connection between a given pair of processors, then this pair can exchange data via an intermediate processor. The



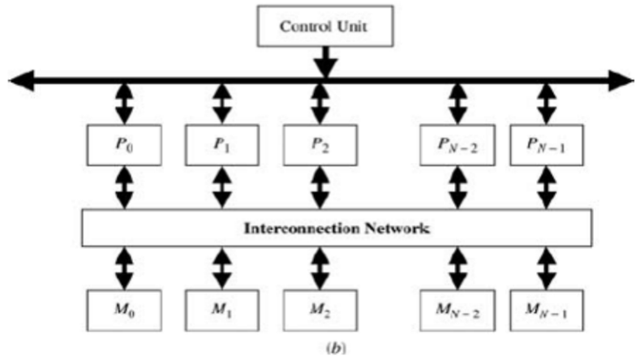


Figure 58: Two SIMD schemes (a) SIMD scheme 1, (b) SIMD scheme 2

ILLIAC-IV used such an interconnection scheme. The interconnection network in the ILLIAC-IV allowed each processor to communicate directly with four neighboring processors in an 8 x 8 matrix pattern such that the ith processor can communicate directly with the (i - 1)th, (i + 1)th, (i - 8)th, and (i + 8)th processors. In the second SIMD scheme, processors and memory modules communicate with each other via the interconnection network.

You should take note that two processors can transfer data between each other via intermediate memory module(s) or possibly via intermediate processor(s). Assume, for example, that processor i is connected to memory modules (i - 1), i, and (i + 1). In this case, processor 1 can communicate with processor 5 via memory modules 2, 3, and 4 as intermediaries.

Are you aware that the BSP (Burroughs' Scientific Processor) used the second SIMD scheme. In order to illustrate the effectiveness of SIMD in handling array operations, consider, for example, the operations of adding the corresponding elements of two one dimensional arrays A and B and storing the results in a third one-dimensional array C. Assume also that each of the three arrays has N elements. Assume also that SIMD scheme 1 is used.

The N additions required can be done in one step if the elements of the three arrays are distributed such that M0 contains the elements A(0), B(0), and C(0), M1 contains the elements A(1), B(1), and C(1), . . . , and MN21 contains the elements A(N - 1), B(N - 1), and C(N - 1). In this case, all processors will execute simultaneously an add instruction of the form C A þ B. After executing this single step by all the processors, the elements of the resultant array C will be stored across the memory modules such that M0 will store C(0), M1 will store C(1), . . . , and MN21 will store C(N - 1).

It is customary to formally represent a SIMD machine in terms of five-tuples (N, C, I, M, F). The meaning of each argument is given below.

1. N is the number of processing elements ($N = 2^k$, $k \geq 1$).
2. C is the set of control instructions used by the control unit, for example, do, for, step.
3. I is the set of instructions executed by active processing units.
4. M is the subset of processing elements that are enabled.
5. F is the set of interconnection functions that determine the communication links among processing elements.

SAQ 2

MIMD Schemes

MIMD machines use a collection of processors, each having its own memory, which can be used to collaborate on executing a given task. In general, MIMD systems can be categorized based on their memory organization into shared-memory and message-passing architectures. The choice between the two categories depends on the cost of communication (relative to that of the computation) and the degree of load imbalance in the application.

Shared Memory Organization

SAQ 3,4

There has been recent growing interest in distributed shared memory systems. This is because shared memory provides an attractive conceptual model for interprocess interaction, even when the underlying hardware provides no direct support. A shared memory model is one in which processors communicate by reading and writing locations in a shared memory that is equally accessible by all processors.

Note that each processor may have registers, buffers, caches, and local memory banks as additional memory resources. Some basic issues in the design of shared-memory systems have to be taken into consideration. These include access control, synchronization, protection, and security. Access control determines which process accesses are possible to which resources. Access control models make the required check for every access request issued by the processors to the shared memory, against the contents of the access control table.

You should also be aware that the latter contains flags that determine the legality of each access attempt. If there are access attempts to resources, then until the desired access is completed, all disallowed access attempts and illegal processes are blocked. Requests from sharing processes may change the contents of the access control table during execution.

The flags of the access control with the synchronization rules determine the system's functionality. Synchronization constraints limit the time of accesses from sharing processes to shared resources. Appropriate synchronization ensures that the information flows properly and ensures system functionality. Protection is a system feature that prevents processes from

making arbitrary access to resources belonging to other processes. Sharing and protection are incompatible; sharing allows access, whereas protection restricts it. Running two copies of the same program on two processors will decrease the performance relative to that of a single processor, due to contention for shared memory. The performance degrades further as three, four, or more copies of the program execute at the same time.

A shared memory computer system consists of (1) a set of independent processors,

(2) a set of memory modules, and (3) an interconnection network. The simplest shared memory system consists of one memory module (M) that can be accessed from two processors Pa and Pb (Fig. 11.8). Requests arrive at the memory module through its two ports. An arbitration unit within the memory module passes requests through to a memory controller. If the memory module is not busy and a single request arrives, then the arbitration unit passes that request to the memory controller, and the request is satisfied. The module is placed in the busy state while a request is being serviced.

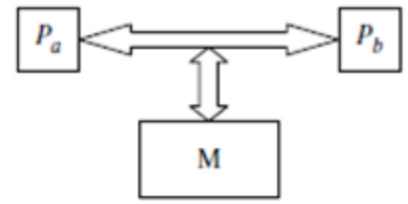


Figure 59: A simple shared memory scheme

If a new request arrives while the memory is busy servicing a previous request, the memory module sends a wait signal through the memory controller to the processor making the new request. In response, the requesting processor may hold its request on the line until the memory becomes free, or it may repeat its request sometime later. If the arbitration unit receives two requests, it selects one of them and passes it to the memory controller. Again, the denied request can be held to be served next, or it may be repeated sometime later. The arbitration unit may not be adequate to organize the use of the memory module by the two processors. The main problem will be in the sequencing of interactions between memory accesses from the two processors.

Consider the following two scenarios for accessing the same memory location M(1000) by the two processors Pa and Pb (Fig. 11). Let us also assume that the initial value stored in memory location M(1000) is 150. Note that in both cases, the sequence of instructions performed by each processor is the same.

Let me tell you that the only difference between the two scenarios is the relative time at which the two processors update the value in M(1000). A careful examination of the two scenarios will show that the value stored in location M(1000) after the first scenario will be 151, while the stored value following the second scenario will be 152. The above illustrative example presents the case of a nonfunctional behaviour of this simple shared memory system. Such an example should demonstrate the basic requirements for the success of such systems. These requirements are:

1. A mechanism for conflict resolution among rival processors
2. A technique for specifying the sequencing constraints
3. A mechanism for enforcing the sequencing specifications

The use of different interconnection networks in a shared memory multiprocessor system leads to systems with one of the following characteristics:

1. Shared memory architecture with uniform memory access (UMA)
2. Cache-only memory architecture (COMA)
3. Distributed shared memory architecture with nonuniform memory access (NUMA)

Cycle	Processor P_a	Processor P_b
1	a ← M(1000);	
2		b ← M(1000);
3	a ← a + 1;	
4		b ← b + 1;
5	M(1000) ← a;	
6		M(1000) ← b;

Scenario 1

Cycle	Processor P_a	Processor P_b
1	a ← M(1000);	
2	a ← a + 1;	
3	M(1000) ← a;	
4		b ← M(1000)
5		b ← b + 1;
6		M(1000) ← b;

Scenario 2

Figure 60: Potential shared memory problem

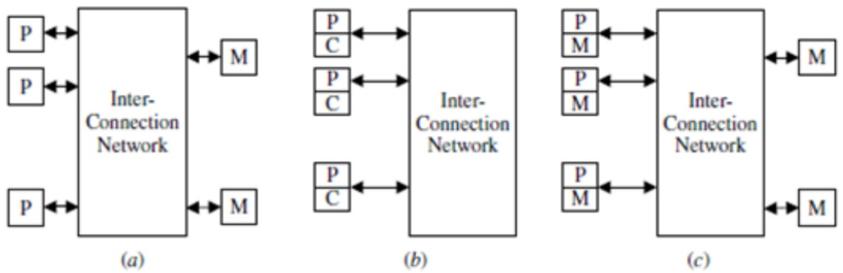


Figure 61: Three example of shared memory architecture (a) UMA (b) COMA (c) NUMA

Figure 45 shows a typical organization for the abovementioned three shared memory architectures. In the UMA system, a shared memory is accessible by all processors through an interconnection network in the same way a single processor accesses its memory. Therefore, all processors have equal access time to any memory location. The interconnection network used in the UMA can be a single bus, multiple buses, a crossbar, or a multiport memory. In the NUMA system, each processor has part of the shared memory attached.

The memory has a single address space. Therefore, any processor could access any memory location directly using its real address. However, the access time to modules depends on the distance to the processor. This results in a non-uniform memory access time. A number of architectures are used to interconnect processors to memory modules in a NUMA.

Among these are the tree and the hierarchical bus networks. Similar to the NUMA, each processor has part of the shared memory in the COMA. However, in this case, the shared memory consists of cache memory. A COMA system requires that data be migrated to the processor requesting it.

Analysis and Performance Metrics

Having provided an introduction to the architecture of multiprocessors, we now provide some basic ideas about the performance issues in multiprocessors. A fundamental question that is usually asked is how much faster a given problem can be solved using multiprocessors as compared to a single processor? This question can be formulated in the speed-up factor defined below.

S(n) = speed-up factor

= Increase in speed due to the use of a multiprocessor system consisting of n processors

$$= \frac{Execution\ time\ using\ a\ single\ processor}{Execution\ time\ using\ n\ processors}$$

A related question concerns how efficiently each of the n processors is utilized. The question can be formulated into the efficiency defined below.

$$E(n) = Efficiency$$

$$= \frac{S(n)}{n} \times 100\%$$

And finally in executing tasks (programs) using a multiprocessor, it may be assumed that a given task can be divided into n equal subtasks, each of which can be executed by one processor. Therefore, the expected speed-up will be given by the S(n) = n while the efficiency E(n) = 100%. The assumption that a given task can be divided into n equal subtasks, each executed by a processor, is unrealistic.

Activity 1: Schemes of Multiprocessors	
Task	**Question**
A multiprocessor is expected to reach a faster speed than the fastest uniprocessor	
	Explain the SIMD Schemes?
	Describe the MIMD schemes?
	Explain the issues surrounding the construction of multiprocessor synchronization primitives?

Summary

In this unit, you have learnt that:

- A multiprocessor consisting of a number of single uniprocessors is expected to be more cost-effective than building a high-performance single processor.
- SIMDs are mostly designed to exploit the inherent parallelism encountered in a matrix (array) operations.
- MIMD machines use a collection of processors, each having its own memory, which can be used to collaborate on executing a given task.

Self-Assessment Questions

1. Explain SIMD Schemes.
2. Explain the MIMD Schemes
3. Explain the Shared Memory Organization.
4. Explain issues surrounding the construction of multiprocessor synchronization primitives.

Tutor Marked Assessment

- What is the main difference between SIMD and MIMD?

Further Reading

- http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
- http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html

References

- Buyya, R. High Performance Cluster Computing: Architectures and Systems. Upper Saddle River, NJ: Prentice Hall, 1999.
- Mano, M. Morris, (1993). Computer System Architecture (3rd ed). Prentice Hall of India.
- Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/ http://ece-www.colorado.edu /faculty/h euring.html