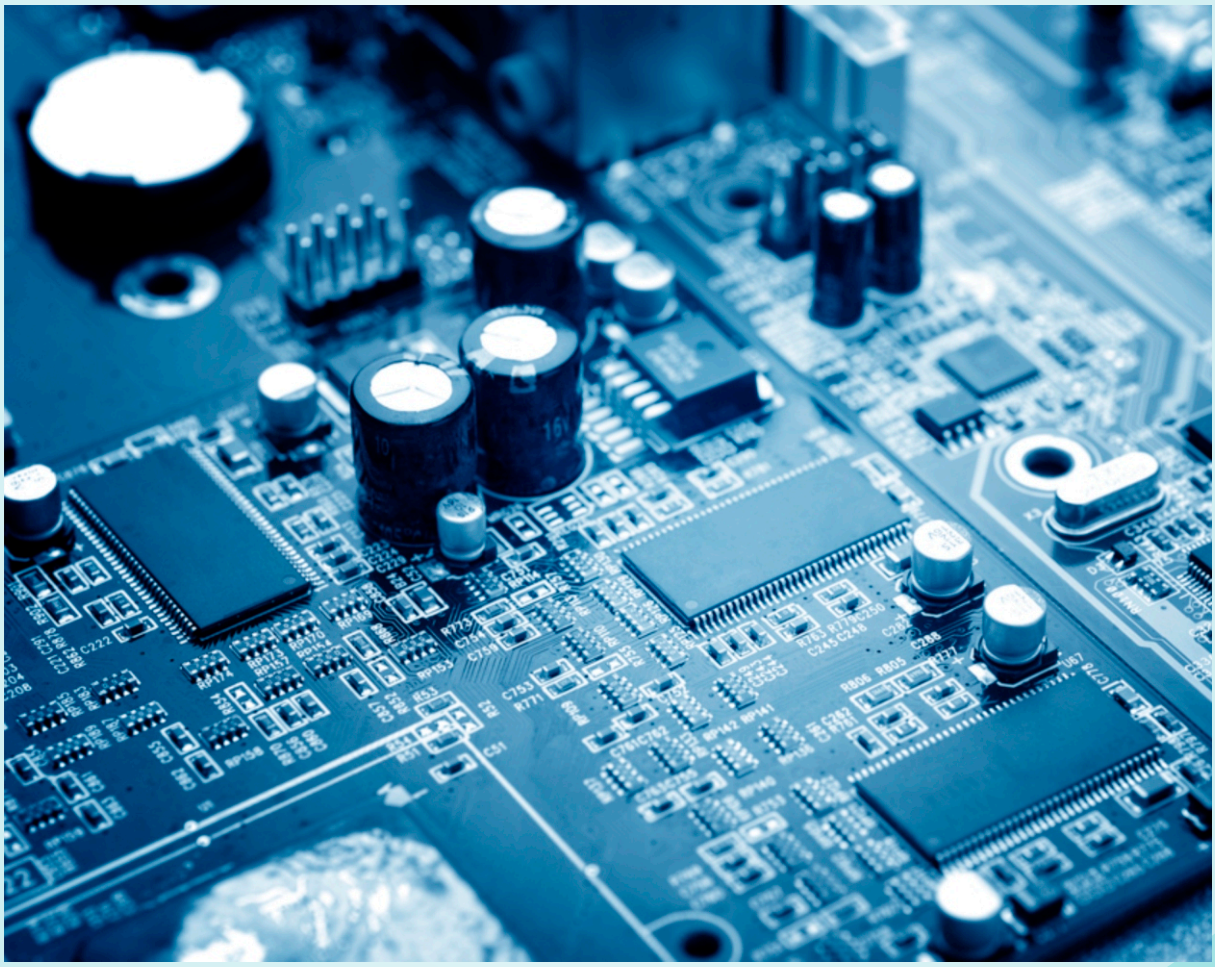
CPT 214

Computer Architecture



CODeL
FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA
CENTRE FOR OPEN DISTANCE AND e-LEARNING

FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA
NIGER STATE, NIGERIA

CENTRE FOR OPEN DISTANCE AND e-LEARNING (CODeL)

B.TECH. COMPUTER SCIENCE PROGRAMME

COURSE TITLE

COMPUTER ARCHITECTURE

COURSE CODE

CPT 214

COURSE CODE

CPT 214

COURSE UNIT

3

Course Coordinator
Bashir MOHAMMED (Ph.D.)
Department of Computer Science
Federal University of Technology (FUT) Minna
Minna, Niger State, Nigeria.

Course Development Team

CPT 214: Computer Architecture

Subject Matter Experts	John K. Alhassan (Ph.D.) Department of Cyber Security FUT Minna, Nigeria.
Course Coordinator	Bashir MOHAMMED (Ph.D.) Department of Computer Science FUT Minna, Nigeria.
ODL Experts	Amosa Isiaka GAMBARI (Ph.D.) Nicholas E. ESEZOBOR
Instructional System Designers	Oluwole Caleb FALODE (Ph.D.) Bushrah Temitope OJOYE (Mrs.)
Language Editors	Chinenye Priscilla UZOCHUKWU (Mrs.) Mubarak Jamiu ALABEDE
Centre Director	Abiodun Musa AIBINU (Ph.D.) Centre for Open Distance & e-Learning FUT Minna, Nigeria.

CPT 214 Study Guide

Introduction

CPT 214 Computer Architecture is a 3- credit unit course for students studying towards acquiring a Bachelor of Science in any field. The course is divided into 5 modules and 16 study units. It will first introduce to digital logics. Then, memory system. Thereafter, interfacing and communication is discussed. This is followed by an extensive discussion on the introduction of networks, raid architectures, Data Path and control. And finally, instruction pipelining, riscs and multiprocessors.

The course guide therefore gives you an overview of what CPT 214 is all about, the textbooks and other materials to be referenced, what you expect to know in each unit, and how to work through the course material.

Recommended Study Time

This course is a 3-credit unit course having 16 study units. You are therefore enjoined to spend at least 2 hours in studying the content of each study unit.

What You Are About to Learn in This Course

The overall aim of this course, CPT 214 is to introduce you to computer architecture. At the end of this course you would have learnt the:

i. basic computing programmes
ii. various components of digital logic
iii. memory system and how to effectively use storage system
iv. the usage of network for raid architectures and Data Path control
v. multiprocessors, riscs and instruction pipelining

Course Aims

This course aims to introduce students to the concept of computer architecture. It is expected that the knowledge will enable the reader to effectively use computers in his/her profession.

Course Objectives

It is important to note that each unit has specific objectives. Students should study them carefully before proceeding to subsequent units. Therefore, it may be useful to refer to these objectives in the course of your study of the unit to assess your progress. You should always look at the unit objectives after completing a unit. In this way, you can be sure that you have done what is required of you by the end of the unit.

However, below are overall objectives of this course. On completing this course, you should be able to:

i. Define computer architecture

ii. Explain digital logic
iii. Explain related technologies to memory system
iv. Describe the categories of computer hardware
v. Discuss interfacing and communication
vi. Explain the functions of networks and how it's being used
vii. Describe how RISCs works
viii. Differentiate between cache memory and virtual memory

Working Through This Course

In order to have a thorough understanding of the course units, you will need to read and understand the contents, practice the steps and implement the knowledge you've gained for your department.

This course is designed to cover approximately sixteen weeks, and it will require your devoted attention. You should do the exercises in the Tutor-Marked Assignments and submit to your tutors.

Course Materials

The major components of the course are:

1. Course Guide
2. Study Units
3. Text Books
4. Assignment File
5. Presentation Schedule

Study Units

There are 16 study units and 5 Modules in this course. They are:

Module One	**Digital logic**	
	Unit 1	Fundamental Building Blocks of Digital Logic
	Unit 2	Programmable Logic Array (PLA)
Module Two	**Memory System**	
	Unit 1	Storage System and their Technologies
	Unit 2	Data Compression and Data Integrity
	Unit 3	Memory Hierarchy, Organisation and Operation
	Unit 4	Cache memory and Virtual Memory
Module Three	**Interfacing and Communication**	
	Unit 1	Input/output Fundamentals
	Unit 2	Handshaking
	Unit 3	Data Buffer
	Unit 4	External Storage

Module Four	**Introduction to Network, Raid Architectures, Data Path & Control**	
	Unit 1	Introduction to Network
	Unit 2	Multimedia Support raid architecture
	Unit 3	Data Path and Control Unit
Module Five	**Instruction Pipelining, RISCS and Multiprocessors**	
	Unit 1	Instruction pipelining
	Unit 2	Introduction to reduced instruction set computers (RISCS)
	Unit 3	Introduction to multiprocessors

Recommended Texts

The following texts and Internet resource links will be of enormous benefit to you in learning this course:

1. Baron, Robert J. and Higbie Lee. *Computer Architecture*. Addison-Wesley Publishing Company.
2. Daniel P, and David G, (2009). A Practical Introduction to Computer Architecture. Text in Computer Science, Springer Dordrecht Heidelberg London New York.
3. Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002
4. Hayes, John P. (1998). *Computer Architecture and Organisation* (2nd ed). McGraw-Hill International editions.
5. Hennessy J.L and Patterson, D.A. *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2006. ISBN: 0-123-70490-1.
6. Mano, M. Morris, (1993). *Computer System Architecture* (3rd ed). Prentice Hall of India.
7. Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/
8. 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.
9. http://ece-www.colorado.edu/faculty/heuring.html
10. NOUN, (2008). INTRODUCTION TO COMPUTER ORGANISATION. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island, Lagos. First Printed 2008
11. Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.
12. Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd Ed.) Prentice Hall of India.

13. http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html
14. http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
15. http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
16. http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html
17. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

Assignment File

The assignment file will be given to you in due course. In this file, you will find all the details of the work you must submit to your tutor for marking. The marks you obtain for these assignments will count towards the final mark for the course. Altogether, there are tutor marked assignments for this course.

Presentation Schedule

The presentation schedule included in this course guide provides you with important dates for completion of each tutor marked assignment. You should therefore endeavour to meet the deadlines.

Assessment

There are two aspects to the assessment of this course. First, there are tutor marked assignments; and second, the written examination. Therefore, you are expected to take note of the facts, information and problem solving gathered during the course. The tutor marked assignments must be submitted to your tutor for formal assessment, in accordance to the deadline given. The work submitted will count for 40% of your total course mark.

At the end of the course, you will need to sit for a final written examination. This examination will account for 60% of your total score. You will be required to submit some assignments by uploading them to CPT 214 page on the u-learn portal.

Tutor-Marked Assignment (TMA)

There are TMAs in this course. You need to submit all the TMAs. The best 10 will therefore be counted. When you have completed each assignment, send them to your tutor as soon as possible and make certain that it gets to your tutor on or before the stipulated deadline. If for any reason you cannot complete your assignment on time, contact your tutor before the assignment is due to discuss the possibility of extension. Extension will not be granted after the deadline, unless on extraordinary cases.

Final Examination and Grading

The final examination for CPT 214 will last for a period of 3 hours and has a value of 60% of the total course grade. The examination will consist of questions which reflect the self-assessment questions and tutor marked assignments that you have previously encountered. Furthermore, all areas of the course will be examined. It would be better to use the time between finishing the last unit and sitting for the examination, to revise the entire course. You

might find it useful to review your TMAs and comment on them before the examination. The final examination covers information from all parts of the course.

Practical Strategies for Working Through This Course

1. Read the course guide thoroughly
2. Organize a study schedule. Refer to the course overview for more details. Note the time you are expected to spend on each unit and how the assignment relates to the units. Important details, e.g. details of your tutorials and the date of the first day of the semester are available. You need to gather together all this information in one place such as a diary, a wall chart calendar or an organizer. Whatever method you choose, you should decide on and write in your own dates for working on each unit.
3. Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they get behind with their course works. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.
4. Turn to Unit 1 and read the introduction and the learning outcomes for the unit.
5. Assemble the study materials. Information about what you need for a unit is given in the table of content at the beginning of each unit. You will almost always need both the study unit you are working on and one of the materials recommended for further readings, on your desk at the same time.
6. Work through the unit, the content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit, you will be encouraged to read from your set books
7. Keep in mind that you will learn a lot by doing all your assignments carefully. They have been designed to help you meet the objectives of the course and will help you pass the examination.
8. Review the objectives of each study unit to confirm that you have achieved them. If you are not certain about any of the objectives, review the study material and consult your tutor.
9. When you are confident that you have achieved a unit's objectives, you can start on the next unit. Proceed unit by unit through the course and try to pace your study so that you can keep yourself on schedule.
10. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor marked assignment form and also written on the assignment. Consult you tutor as soon as possible if you have any questions or problems.
11. After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this course guide).

Tutors and Tutorials

There are few hours of tutorial provided in support of this course. You will be notified of the dates, time and location together with the name and phone number of your tutor as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course. You must mail your tutor marked assignment to your tutor well before the due date. At least two working days are required for this purpose. They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by telephone, e-mail or discussion board if you need help. The following might be circumstances in which you would find help necessary: contact your tutor if:

- You do not understand any part of the study units or the assigned readings.
- You have difficulty with the self-test or exercise.
- You have questions or problems with an assignment, with your tutor's comments on an assignment or with the grading of an assignment.

You should endeavour to attend the tutorials. This is the only opportunity to have face to face contact with your tutor and ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain the maximum benefit from the course tutorials, have some questions handy before attending them. You will learn a lot from participating actively in discussions.

GOODLUCK!

Table of Contents

Module 1

Digital Logic

Unit 1: Fundamental Building Blocks of Digital Logic

Unit 2: Programmable Logic Array (PLA)

Unit 1

Fundamental Building Blocks of Digital Logic

Contents

1.0 Introduction

In this unit, you will learn digital logic. More so, you will briefly learn about the Boolean algebra, which will be useful in the discussions on logic circuits. This unit also discuss on logic gates and combination circuits.

2.0 Learning Outcome

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

1. Define Boolean algebra
2. Explain logic gates
3. Describe combination circuits

3.0 Learning Content

3.1 Boolean Algebra

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 xy, addition $x + y$, and negation $-x$ replaced by the respective logical operations of conjunction $x \wedge y$, disjunction $x \vee y$, and negation $\neg x$.

The Boolean operations are these and all other operations that can be built from these, such as $x \wedge (y \vee z)$. They turned out to coincide with the set of all operations on the set {0,1} that take only finitely many arguments; there are 2^{2^n} such operations when there are n arguments. Is that clear.

Do you know that the 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. Also, the axiomatic approach is sound and complete in the sense that it proves respectively neither more nor fewer laws than the semantic approach.

You should also note that Boolean algebra is used for designing and analyzing digital circuits. First, we will discuss the rules of Boolean algebra and thereafter we will be discussing how it can be used in analyzing or designing digital circuits. Now answer the below question.

Self-Assessment Question

1. Boolean algebra was developed by _____.

Self-Assessment Answer

1. George Boole

Point 1:

Have at the back of your mind that a variable in Boolean algebra can take only two values

1 (TRUE) or 0 (FALSE)

Point 2:

There are three basic operations in Boolean algebra, viz:

AND, OR and NOT

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

A AND B or A.B or AB

A OR B or A + B

NOT A or ¬A or A' or Ā

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

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

In addition, three more operators have been defined for Boolean algebra: Say the below aloud.

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

However, if you will be designing and analyzing a logical circuit, it is convenient to use AND, NOT and OR operators. Because AND and OR obey many laws as of multiplicationion and addition in the ordinary algebra of real numbers.

Example: Complete the truth table below:

A	B	A + B
0	0	
0	1	
1	0	
1	1	

Solution

Remember that A + B is the same as A OR B. Therefore, your answer should be;

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Point 3:

The basic logical identities used in Boolean algebra are:

BASIC IDENTITIES

$A.B = B.A$	$A+B = B+A$	Commutative law
$A.(B+C) = (A.B)+(A.C)$	$A+(B.C) = (A+B).(A+C)$	Distributive law
$1.A = A$	$0+A = A$	Identity law
$A.\bar{A} = 0$	$A+\bar{A} = 1$	Inverse law

OTHER IDENTITIES

$0.A = 0$	$1+ A= 1$	
$A.A = A$	$A+A = A$	
$A.(B.C) = (A.B).C$	$A+ (B+C) = (A+B)+C$	Associative law
$A.B = \bar{A}+B$	$A+B = \bar{A}.B$	Demorgan'sTheorem

DeMorgan's law is very useful in simplifying logical expressions. Having explained the Boolean algebra, it is important you review it first before proceeding.

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.

So the value of Boolean Function F can be either 0 or 1.

A Boolean function can be broken into 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. Is that understood? Now answer the questions below.

Self-Assessment Question

1. The three basic operations of Boolean algebra are _____.

Self-Assessment Answer

1. AND, OR, NOT

3.2 Logic Gates

What is a Logic Gate?

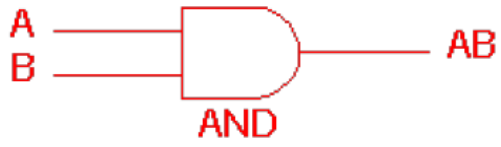
It is an idealized or physical device implementing a Boolean function. That is, it performs a logical operation on one or more logic inputs and produces a single logic output. Depending

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.

You should be aware that logic gate is an electronic circuit which 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. Remember your point 2? Any Boolean function can be represented in the form of gates.

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

AND gate



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

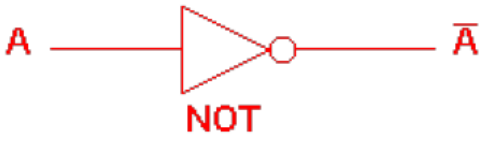
The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB

OR gate

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

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.

NOT gate



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

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. Now pay attention! If your 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. It can also be done using NOR logic gates in the same way.

NAND gate

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

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.
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.

NOR gate

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

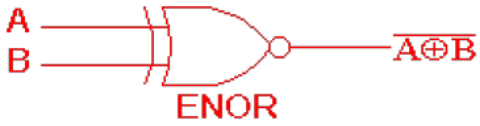
The '**Exclusive-OR**' gate is a circuit which will give a high output if **either, but not both**, of its two inputs are high. An encircled plus sign ( ) is used to show the EOR operation.

EXOR gate

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

The '**Exclusive-NOR'** gate circuit does the opposite of the EOR gate. It will give 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.

EXNOR gate



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

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 **product of sums** form can be implemented using NOR gates by replacing all AND and OR gates by NOR gates.

Table 1: Logic gate symbols

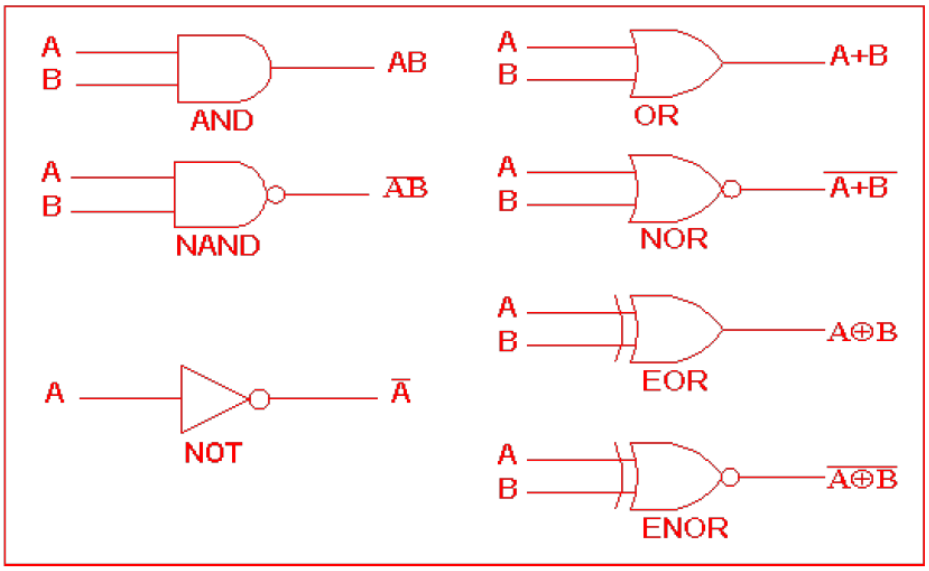


Table 2 is a summary truth table of the input/output combinations for the NOT gate together with all possible input/output combinations for the

Other gate functions. Also note that a truth table with 'n' inputs has 2^n rows. You can compare the outputs of different gates.

Table 2: Logic gates representation using the Truth table

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

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

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 a one output if two inputs are equal. Be aware that 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.

What does 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 compete sets are: (AND, NOT), (NOR), (NAND), (OR< NOT) etc. Now do the exercise below.

Self-Assessment Questions(S)

1. What are logical gates?
2. List five logical gates you know.

Self-Assessment Answer(S)

1. 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.
2. AND, OR, NOR, EXOR, EXNOR

3.3 Combination Circuits

Do you know that combinational circuits are interconnected circuits of gates according to a certain rule to produce an output depending on its input value? A well-formed combinational circuit should not have feedback loops. A combinational circuit can be represented as a network of gates and, can be expressed by a truth table or a Boolean expression.

The output of a combinational circuit is related to its input by a combinational function, which is independent of time. Therefore, for an ideal combinational circuit, the output should change according to changes in input. But in actual cases there is a slight delay. This delay is normally proportional to the depth or number of levels, i.e the maximum number of gates lying on any path from input to output.

For example, the depth of the combinational circuit in figure 1 is two

Example: Draw the logic gates for the function F = A.B + C

Solution:

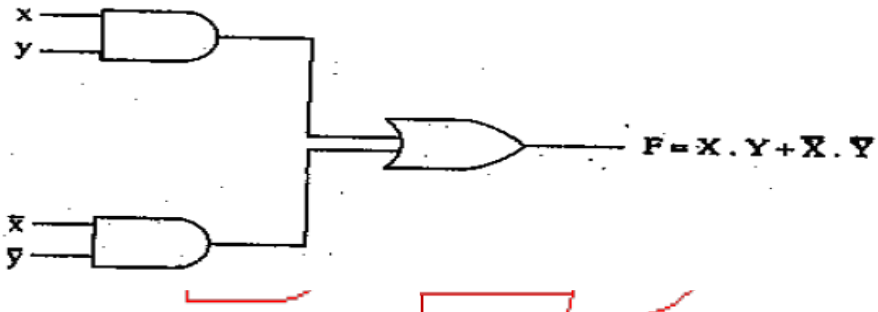


Figure 1: A two level AND-OR combinational circuit

The basic design issue related to combinational circuits is: the reduction/minimization of the number of gates. The normal circuit constraints for combinational circuit design are:

The depth of the circuit should not exceed a specific level.

i. Number of input lines to a gate (fan in) and how many gates its output can be fed (fan out) are constrained by the circuit power constraints.

Minimization of Gates

The simplification of the Boolean expression is very useful for combinational circuit designs. The following three methods are used for this.

1. Algebraic simplification

2. Karnaugh maps
3. The Quine McCluskey method

But before defining any of the above stated methods let us discuss the forms of algebraic expressions. An algebraic expression can exist in two forms:

1. Sum of products (SOP) e.g. (A.¬B) + (¬A. ¬B)
2. Product of sums (POS) e.g. (¬A +¬B).(A+B)

If a product of SOP expression contains every variable of that function either in true or complement form, then it is defined as a minterm. This minterm will be true only for one combination of input values of the variables. For example, in the SOP expression-

F (A, B, C) = (A.B.C) + (¬A. ¬B.C) + (A.B)

We have three product terms namely A.B.C, ¬A. ¬B.C and A.B. But only the first two of them qualify to be a minterm, as the third one does not contain variable C or its complement. In addition, the term A.B.C will be one only if A=1, B=1 and C=1 for any other combination of values of A, B, C the minterm will have zero value. Similarly, the minterm ¬A. ¬B.C will have value 1 only if ¬A = 1 i.e. A=0, ¬B=1 i.e. B=0 and C=1.For any other combination of values the minterm will have a zero value. Ask your tutor for more explanation.

Also on the second note, a similar type of term used in POS form is called maxterm. And maxterm is a term of POS expression, which contains all the variables of the function in true or complemented form. For example,

F(A, B, C)=(A=B=C).(¬A+¬B+C) have two maxterms. A maxterm have a value 0 for only one combination of input values. You'll agree that maxterm A+B+C will be 0 value only for A=0, B=0 and C=0. For all other combination of values of A, B, C it will have a value one.

Now let us come back to the problem of minimizing the number of gates.

Algebraic Simplification

We have already discussed the algebraic simplification of a logical circuit. An algebraic expression can exist in POS or SOP forms. Okay? The algebraic functions can appear in many different forms. Although the process of simplification exists yet it is cumbersome because of the absence of routes, which tell what rule to apply next. The Karnaugh map is a simple direct approach of simplification of logical expressions.

Karnaugh Maps

The Karnaugh map is a convenient way of representing and simplifying Boolean functions of 4 to 6 variables. Karnaugh maps can also be used for designing the circuits in situations where you can construct the truth table for an operation or a function. In other words, Karnaugh maps can be used to construct a circuit when the input and output to that proposed circuit are defined. For each output one Karnaugh map needs to be constructed. The stepwise procedure for Karnaugh map is as follows:

Step 1

Create a simple map depending on the number of variables in the function. Figure 2 shows the map of two, three and four variables. A map of 2 variables contains 4 value positions or elements, while for 3 variables it has 2^3=8 elements; similarly, for 4 variables it is 2^4=16

elements and so on. Special care is taken to represent variables in the map. Value of only one variable changes in two adjacent columns or rows.

The advantage of having a change in one variable is that two adjacent columns or rows represent a true form or complement form of a single variable. For example, in Figure 2 the columns which have positive A are adjacent to ¬A. Please note the adjacency of the corners. The rightmost column can be considered to be adjacent to the first column; since they differ only by one variable, therefore, they are adjacent. Similarly, the topmost and bottommost rows are adjacent.

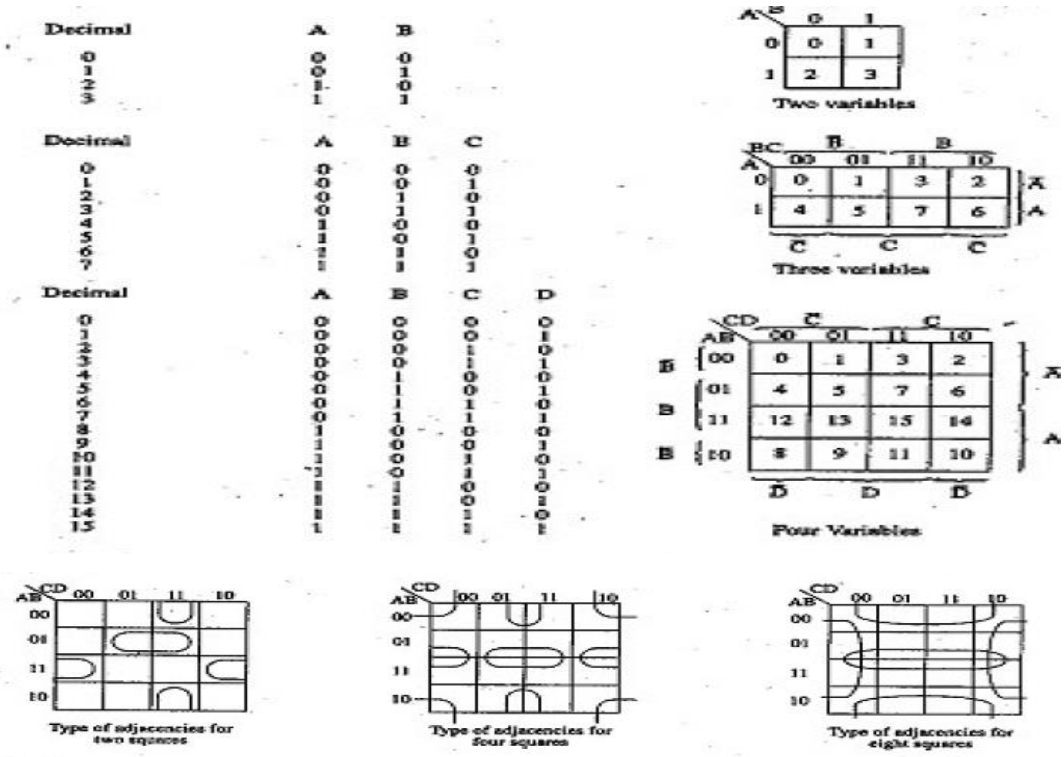


Figure 2: Maps of two, three and four variables and their adjacencies

Note the following:

1. Decimal equivalents of the cells are given for help in understanding where the position of the respective decimal equivalent is. It is not the value filled in a square. A square can contain one or nothing.
2. The 00, 01, 11 etc. written on the top implies the value of respective variables.
3. Wherever the value of a variable is zero it is said to represent its complement form.
4. The value of only one variable changes when we move from one row to the next row or from one column to the next column.

Step 2:

The next step in the Karnaugh map is to map the truth table into the map. The mapping is done by putting a 1 in the respective squares belonging to the 1 value in the truth table. This mapped map is used to arrive at simplified Boolean expressions, which then can be used for drawing up the optimal logical circuits. Step 2 will be clearer in the example.

Step 3:

Now create simple algebraic expressions from the Karnaugh map. These expressions are created by using adjacency if we have two adjacent 1's then the expressions for those can be simplified together since they differ in only one variable. Similarly, we search for adjacent pairs of four, eight, and so on. A 1 can appear in more than one adjacent pairs. You must find adjacencies till all 1's in the Karnaugh map are covered. The following example will clarify step 3.

Example 2: Now let us see how to use Karnaugh map simplification for finding the Boolean function for the cases whose truth table is given as:

A	B	C	D	Decimal Equivalent	Output F
0	0	0	0	0	1
0	0	0	1	1	1
0	0	1	0	2	1
0	0	1	1	3	0
0	1	0	0	4	0
0	1	0	1	5	0
0	1	1	0	6	1
0	1	1	1	7	0
1	0	0	0	8	1
1	0	0	1	9	1
1	0	1	0	10	1
1	0	1	1	11	0
1	1	0	0	12	0
1	1	0	1	13	0
1	1	1	0	14	0
1	1	1	1	15	0

Another short representation of the truth table is Σ(0,1,2,6,8,9, 10) which indicate the decimal equivalent for A, B, C, D values for which the output is one.

Let us construct the Karnaugh map for this.

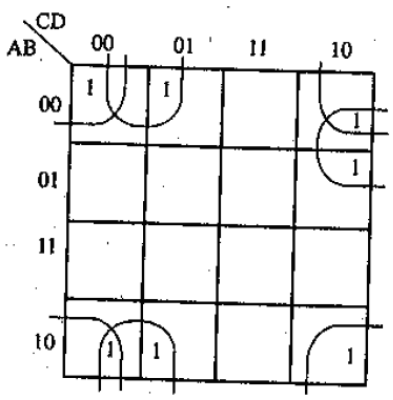


Figure 3: Karnaugh's map of the truth table of example 2

Let us see the pairs which can be considered adjacent in the Karnaugh map here.

The pairs are

1. The four corners
2. The four 1's as in top and bottom in columns 1 and 2
3. The two 1's in the top two rows of the last column.

The corners can be represented by the expressions:

1. $(\neg A\neg B\neg C\neg D + \neg A\neg BC\neg D) + (A\neg B\neg C\neg D + A\neg BC\neg D)$
$= \neg A.\neg B.\neg D(\neg C+C) + A.\neg B.\neg D(\neg C+C)$
$= \neg A\neg B\neg D + A\neg B\neg D$ as $(\neg C+C)=1$
$= \neg B\neg D\ (\neg A + A)$
$= \neg B\neg D$ as $(\neg A+A)=1$

2. The four 1's give the following term:

$(\neg A\neg B\neg C\neg D + \neg A\neg B\neg C.D) + (A\neg B\neg C\neg D + A\neg B\neg CD)$
$= \neg A\neg B\neg C\ (\neg D + D) + A\neg B\neg C\ (\neg D + D)$
$= \neg A\neg B\neg C + A\neg B\neg C$ as $(\neg D+D)=1$
$= \neg B\neg C\ (\neg A + A)$
$= \neg B\neg C$ as $(\neg A+A)=1$

3. The two 1's in the last column

$\neg A\neg B\ C\neg D + \neg A\ B\ C\ \neg D$
$= \neg A\ C\ \neg D\ (\neg B + B)$
$= \neg A\ C\ \neg D$ as $(\neg B+B)=1$

Thus, the Boolean expression derived from this Karnaugh map is

$$F = \neg B\ \neg D + \neg B\ \neg C + \neg A\ C\ \neg D$$

The expressions so obtained through the Karnaugh map are in the form of the sum of the product form, i.e. it is expressed as a sum of the products of the variables. The expression is one of the minimal solutions. This expression can be expressed in product of the sum form, but for this, special methods need to be used.

Let us see how we can modify the Karnaugh map simplification to obtain the product of the sum form. Suppose in the previous example instead of using 1's we combine the adjacent zero square then we will obtain the inverse function and on taking NOT of this function we will get the product of sum form (the use of DeMorgan's theorem will be required).

Another important aspect of this simple method of digital circuit design is DONOT care conditions. These conditions further simplify the algebraic function. These conditions imply that it does not matter whether the output produced is zero or 1 for a specific input. These conditions can occur when the combination of the number of inputs are more than needed; e.g., calculation through BCD where 4 bits are used to represent a decimal digit, which implies we can represent $2^4 = 16$ digits but since we have only 10 decimal digits, therefore, 6 of those input combinations do not matter and thus, are a candidate for DONOT care condition.

What will happen if we have more than 4-6 variables? As the number of the variables increases the Karnaugh map becomes more and more cumbersome (as the number of possible combination of inputs keeps on increasing). A method was suggested to deal with the increasing number of variables. This is a tabular approach and is known as the Quine-Mckluskey method.

Know that this method is suitable for programming and hence provides a tool for automating designs in the form of minimized Boolean expressions. The basic principle behind the Quine-Mckluskey method is to remove the terms which are redundant and can be obtained by other terms.

Self-Assessment Question

1. List three methods that have been used to minimize the use of Gates.

Self-Assessment Answer

The following three methods are used for this.

1. Algebraic simplification
2. Karnaugh maps
3. The Quine McCluskey method

4.0 Conclusion

In this unit, I want to believe you have learnt about the digital logic. You have also learnt about the Boolean algebra, which is the basis for discussions on logic circuits. You also learnt in this unit the logic gates and combination circuits.

5.0 Summary

We have discussed that Boolean algebra is an attempt to represent the true-false logic of humans in mathematical form. George Boole proposed the principles of the Boolean algebra in 1854, hence the name Boolean algebra. Do not forget that Boolean algebra is used for designing and analyzing digital circuits. And its function is defined as an algebraic expression formed with the binary variables, the logic operation symbols, parenthesis, and equal to sign.

More so, digital systems are said to be constructed by using logic gates. A logic gate is an electronic circuit which produces a typical output signal depending on its input signal. Also, combinational circuits are interconnected circuits of gates according to a certain rule to produce an output depending on its input value.

And lastly, remember that well-formed combinational circuit should not have feedback loops. A combinational circuit can be represented as a network of gates and, can be expressed by a truth table or a Boolean expression.

6.0 Tutor-Marked Assignment

1. Explain logic gates with AND gate and OR gate as examples?
2. Explain sum of products and product of sums?

7.0 References/Further Reading

1. Baron, Robert J. and Higbie Lee. *Computer Architecture*. Addison-Wesley Publishing Company.
2. Daniel P, and David G, (2009). A Practical Introduction to Computer Architecture. Text in Computer Science, Springer Dordrecht Heidelberg London New York.
3. Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002
4. Hayes, John P. (1998). *Computer Architecture and Organisation* (2nd ed). McGraw-Hill International editions.
5. Hennessy J.L and Patterson, D.A. *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2006. ISBN: 0-123-70490-1.
6. Mano, M. Morris, (1993). *Computer System Architecture* (3rd ed). Prentice Hall of India.
7. Miles J, and Vincent P. H. (1999). Principle of Computer Architecture – Class Test Edition, August 1999. http://www.cs.rutgers.edu/~murdocca/
8. http://ece-www.colorado.edu/faculty/heuring.html
9. 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.
10. NOUN, (2008). INTRODUCTION TO COMPUTER ORGANISATION. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island, Lagos. First Printed 2008
11. Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.
12. Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd ed) Prentice Hall of India.
13. http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html
14. http://www.cs.ucsd.edu/classes/wi99/cse141_B/lectures.html
15. http://www.cs.caltech.edu/courses/cs184/ winter2001/slides/day5_2up.pdf
16. http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/compSched.html
17. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

Unit 2

Programmable Logic Array (PLA)

Contents

1.0 Introduction

In this unit, what you will learn is about programmable logic array (PLA). One way to design a combinational logic circuit is that you get gates and connect them with wires. One disadvantage with this way of designing circuits is its lack of portability. You will also learn about Programmable Logic Array (PLA), definition of a Flip-Flop, and registers.

2.0 Learning Outcome

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

1. Explain programmable logic array (PLA)
2. Define a flip-flop
3. Describe registers

3.0 Learning Content

3.1 Programmable Logic Array

You should know that the individual gates are treated as basic building blocks from which various logic functions can be derived before now. You have also learnt about the strategies of minimization of number of gates in the previous unit. But with the advancement of technology, the integration provided by integrated circuit technology has increased resulting in the production of one to ten gates on a single chip (in Small Scale Integration (SSI)).

The gate level designs are constructed at the gate level only. But if the design is to be done using these SSI chips, the design consideration needs to be changed as a number of such SSI chips may be used for developing a logic circuit. With MSI & VLSI we can put even more gates on a chip and can also make gate interconnections on a chip. These integration and connection bring the advantages of decreased cost, size, and increased speed.

Note that the basic drawback faced in such VLSI & MSI chips is that for each logic function the layout of gates and interconnection need to be designed. The cost involved in making such a custom chip design is quite high. This brings us to the concept of Programamble Logic Array (PLA), a general-purpose chip that can be readily adopted for any specific purpose. The PLA are designed for SOP form of Boolean function and consist of a regular arrangement of NOT, AND & OR gates on a chip.

Each input to the chip is passed through a NOT gate, thus, the input and their complement are available to each AND gate. The output of each AND gate is made available for each OR gate. And the output of each OR gate is treated as chip output. By making appropriate connections, any logic function can be implemented in these Programmable Logic Arrays. Now take a close look at the below diagram.

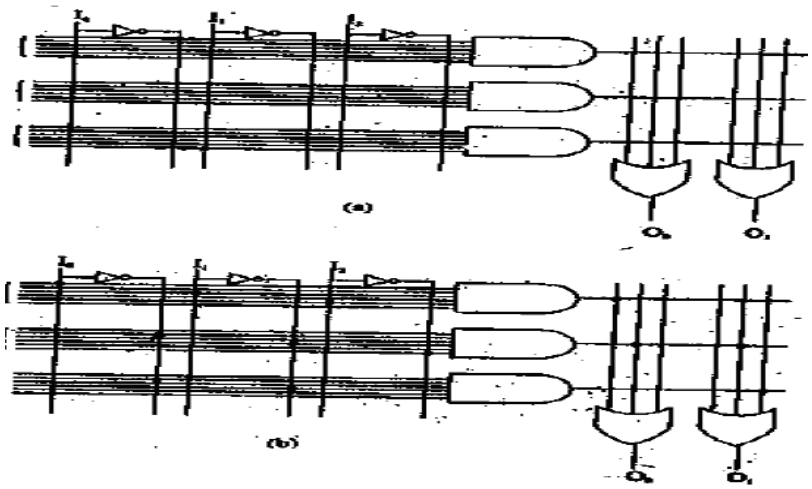


Figure 1: Programmable Logic Array

The figure 1(a) shows a PLA of three inputs and two outputs. Please note the connectivity points, as all these points can be connected if desired. Figure 1(b) shows an implementation of logic function:

$$O_0 = I_0, I_1, I_2 + \neg I_0\ \neg I_1\ \neg I_2 \text{ and } O_1 = \neg I_0\ \neg I_1\ \neg I_2 + \neg I_0.\ \neg I_1$$

Please note the second function is a non-optimal function and can be simplified using Boolean algebra.

Self-Assessment Question

1. What is a Programmable Logical Array?

Self-Assessment Answer

1. Programmable Logic Array (PLA) is a general-purpose chip that can be readily adopted for any specific purpose.

3.2 What is a Flip-Flop?

A flip-flop is a binary cell, which can store a bit of information and which in itself is a step-to-step circuit. But, how does it happen? A flip-flop maintains any one of the two stable states that can be treated as zero or one depending on the presence and absence of output signals. This state of a flip-flop can only change when a clock pulse has arrived. First see the basic flip-flop or a latch that was a revolutionary step in computers. The basic latch presented here is not in order. Let us see its logic diagram (Figure 2).

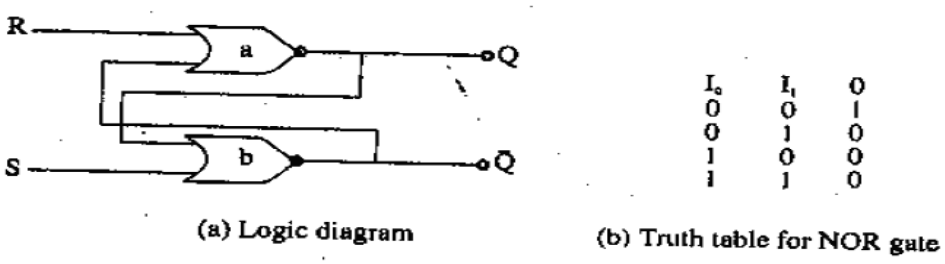


Figure 2: A Basic latch (S-R latch using NOR gates)

You can construct a flip-flop using two NAND or NOR gates; which contains a feedback loop. The flip-flop you saw in the above figure has two inputs R (Reset) and S (set) and two outputs Q and ¬Q. In a normal mode of operation both the flip-flop inputs are at zero i.e. S = 0 & R = 0. This means that the flip-flop can show two states: either the value Q is 1 (therefore ¬Q = 0). we say the flip-flop is in set state or the value of Q is 0 (therefore ¬Q = 1) we call it a clear state. Do you understand?

Now let's take this example: Draw the truth table for A flip-flop NAND gate

Solution:

The first thing is to find the AND then the NOT as shown in the table below

I_0	I_1	NAND
0	0	1
0	1	1
1	0	1
1	1	0

Let us see how the S and R input can be used to set and clear the state of the flip-flop.

The first question you should ask yourself is, why in normal cases S and R are zero? The reason is that this state does not cause any change in state. Suppose the flip-flop was in set state i.e. Q = 1 and ¬Q = 0 and as S = 0 and R = 0, the output of 'a' NOR gate will be 1 since both its input ¬Q and R are zero (refer to the truth table of 1 NOR gate in Figure 2) and 'b' NOR gate will show output as 0 as one of its input Q is 1. Similarly, if flip-flop was in clear state then ¬Q = 1 and R = 0, therefore, output of 'a' gate will be 0 and 'b' gate 1. Thus, flip-flop maintains a stable state at S = 0 and R = 0.

Now the flip-flop is taken to set state if the S input quickly goes to 1 and then goes back to 0. R remains at zero during this time. What happens if, say initially, the flip-flop was in state 0 i.e. the value of Q was 0. As soon as S becomes 1 the output of NOR gate 'b' goes to 0 i.e. ¬Q becomes 0 and almost immediately Q becomes 1 as both the input (¬Q and R) to NOR gate 'a' become 0.

You see, change in the value of S back to 0 does not change the value of Q again as the input to NOR gate 'b' now are Q = 1 and S = 0. As a result, the flip-flop stays in the set state even after S returns to zero. If the flip-flop was in state 1 then, when S goes to 1 there is no change in value of ¬Q as both the inputs to NOR gate 'b' are 1 at this time. Thus, ¬Q remains in state 0 or in other words flip-flop stays in the set state.

But R input goes to value 1 then flip-flop acquires the clear state. On changing for a short time, the value of R to 1 the Q output changes to 0 irrespective of the state of flip-flop and as Q is 0 and S is 0 the ¬Q becomes 1. Even after R comes back to value 0, Q remains 0 i.e., flip flop comes to the clear state.

Now, anybody can ask you that what will happen when both S and R go to 1 at the same time. Well, you should be able to say that this is the situation which may create a set or clear state depending on which of the S and R stays longer in zero state. But meanwhile both of them are 1 and the value of Q and ¬Q becomes 1 which implies that both Q and its complement are one, an impossible situation. Therefore, the transition of both S and R to 1 at the same time is an undesirable condition for this basic latch. Let us try to construct a synchronous R-S flip-flop form the basic latch. The clock pulse will be used to synchronize the flip-flop. (What is a clock pulse?).

Self-Assessment Question

1. What is a Flip Flop?

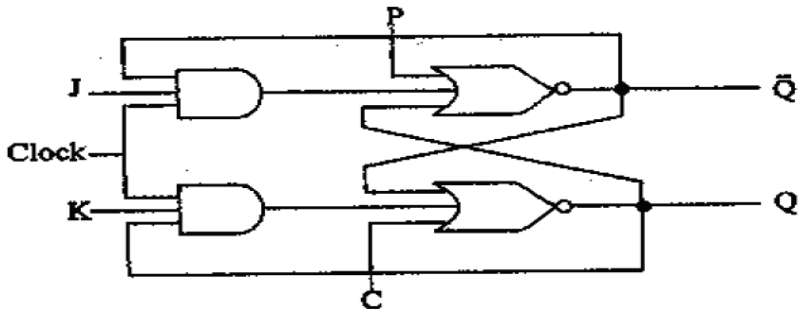
Self-Assessment Answer

1. A flip-flop is a binary cell, which can store a bit of information and which in itself is a sequential circuit.

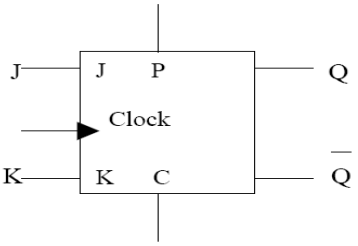
3.2.1 D Flip-Flop

D flip-flop is a special type of flip-flop in the sense that it represents the currently applied input as the state of the flip-flop. Thus, in effect it can store 1 bit of data information and is sometimes referred to as Data flip-flop. Please note that the state of the flip-flop changes for the applied input. It does not have a condition where the state does not change as the case in RS flip-flop, the state of R-S flip-flop does not change when S = 0 and R = 0.

If we do not want a particular input state to change, then either the clock is to be disabled during that period or a feedback of the output can be embedded with the input D. Do not forget that D flip-flop is also referred to as Delay flip-flop because it delays the 0 or 1 applied to its input by a single clock pulse.



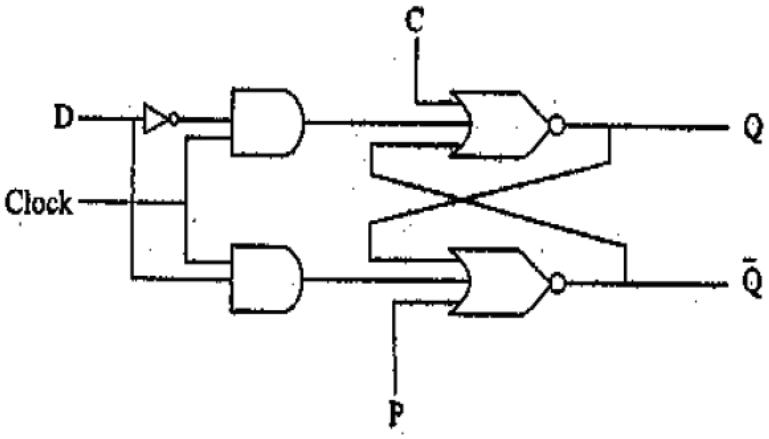
(a) Logic diagram



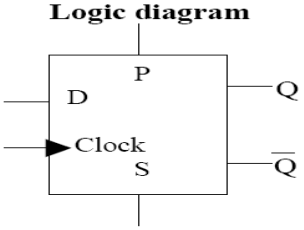
Input J	Input K	State at the completion of clock cycle
0	0	No change in State
0	1	Clear the flip-flop (state 0)
1	0	Set the flip-flop (state 1)
1	1	Complement the state of flip-flop

(b) Symbolic representation **(c) Characteristic table**

J-K flip-flop



(a) Logic diagram



Input D	State after the completion of clock pulse
0	0
1	1

(b) Symbolic representation (c) Characteristic table

D flip-flop

Figure 3: Other flip-flops

3.2.2 J K flip-flop

The basic drawback with the R S flip-flop is that one set of input conditions are not utilized and this can be used with a little change in the circuit. In this flip-flop, the last combination is used to complement the state of the flip-flop. After discussing some of the simple sequential circuits, that is flip-flop, let us discuss some of the complex sequential circuits, which can be developed using simple gates, and flip-flops.

Self-Assessment Question

1. List two (2) types of Flip-Flops you know.

Self-Assessment Answer

1. D-Flip-Flop and JK Flip-Flop

3.3 Registers

A register is a binary function which holds the binary information in digital form. Thus, a register consists of a group of binary storage cells. A register consists of one and more flip-flops depending on the number of bits to be stored in a word. A separate flip-flop is used for storing a bit of a word. In addition to storage, registers are normally coupled with combinational gates enabling certain data processing tasks.

Thus, a register in a broad sense consists of the flip-flop that stores binary information and gates, which controls when and how information is transferred to the register. Normally in a register you will notice that independent data lines are provided for each flip-flop, enabling the transfer of data to and from all flip-flops to the register simultaneously.

I want you to know that this mode of operation is called Parallel Input-Output. Since the stored information in a set of flip-flops is treated as a single entity, common control signals such as clock, preset and clear can be used for all the flip-flops of the register. Registers can be constructed from any type of flip-flop. These flip-flops in integrated circuit registers are usually constructed internally using two separate flip-flop circuits. The normally used special kind of arrangement is termed the master slave flip-flop. This type of flip-flop helps in having a stable state at the output. It consists of a master flip-flop and a slave flip-flop.

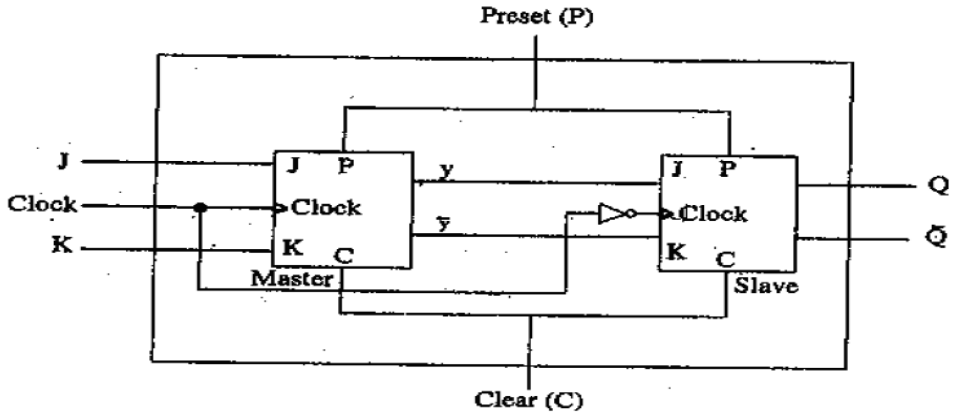


Figure 4: Masters-slave flip-flop using J-K flip-flop

Note: You can construct master-slave flip-flop with D flip-flop (Figure 4) or R-S flip-flop in the same manner.

Now, let us analyze this flip-flop.

1. When the clock pulse is 0 the master is disabled but the slave becomes active and its output Q and ¬Q becomes equal to Y and ¬Y respectively. Why? Well, the possible combination of the value of Y and ¬Y are either Y = 1 which means ¬Y = 0; or Y = 0 which implies ¬Y = 1. Let us see the characteristic table for these two inputs for the J-K flip-flop. The SLAVE flip-flop,

   Thus, it can have value either J=1 and K=0 which will set the flip-flop that is Q=1 and ¬Q=0; or J=0, K=1 which will clear the flip-flop. Therefore, Q is same as Y.

2. When inputs are applied at J and K and the clock pulse becomes 1, only the master gets activated, resulting in intermediate output Y go to state 0 or 1 depending on the input and previous state. Please note that during this time the slave is still maintaining its previous state. As the clock pulse become 0, the master becomes inactive and the slave acquires the same state as the master.

   But why do we acquire this master-slave combination? There is a major reason for this master-slave form. Consider a situation where the output of a flip-flop is going to input of other flip-flops. Here, the assumption is that the clock pulse inputs of all flip-flops are synchronized and occur at the same time.

   The change of state of the master occurs when the clock pulse goes to 1, but during that time the output of the slave still has not changed, thus the state of the flip-flops in the system can be changed simultaneously during the same clock pulse even though outputs of flipflops are connected to the inputs of flip-flops. In other words, there are no restrictions on feedback from the register's outputs to its inputs.

Self-Assessment Questions

1. What is the mode of operation of a register?

Self-Assessment Answer(s)

1. Parallel Input-Output.

4.0 Conclusion

In this unit, you have learned about the programmable logic array (PLA). You have also learnt about definition of a Flip-Flop and registers.

5.0 Summary

The individual gates are treated as basic building blocks from which various logic functions can be derived before now. You have also learnt about the strategies of minimization of number of gates. But with the advancement of technology the integration provided by integrated circuit technology has increased resulting in the production of one to ten gates on a single chip (in Small Scale Integration (SSI)).

A flip-flop is a binary cell, which can store a bit of information and which in itself is a sequential circuit. But, how does it do it? A flip-flop maintains any one of the two stable states that can be treated as zero or one depending on the presence and absence of output signals. The state of a flip-flop can only change when a clock pulse has arrived. D flip-flop is a special type of flip-flop in the sense that it represents the currently applied input as the state of the flip-flop.

Thus, in effect it can store 1 bit of data information and is sometimes referred to as Data flip-flop. A register is a binary function which holds the binary information in digital form. Thus, a register consists of a group of binary storage cells. A register consists of one and more flip-flops depending on the number of bits to be stored in a word.

6.0 Tutor-Marked Assignment

1. With the aid of a diagram describe a flip-flop?
2. Define a register?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

NOUN, (2008). Introduction To Computer Organisation. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island,Lagos. First Printed 2008

Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.

Tanenbaum, Andrew S. (1993).*Structural Computer Organisation* (3rd ed) Prentice Hall of India.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

Module 2

Memory Systems

Unit 1

Storage Systems and Their Technologies

Contents

1.0 Introduction

In this unit, what you will learn borders on storage systems and their technologies. Computer data storage, often called storage or memory. It is a technology consisting of computer components and recording media used to retain digital data. It is also a core function and fundamental component of computers. Yes, you will also learn in this unit about fundamental storage technologies and related technologies.

2.0 Learning Outcome

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

i. Define computer data storage
ii. Explain semiconductor
iii. Describe magnetic storage
iv. Explain optical storage
v. Explain related technologies

3.0 Learning Content

3.1 Computer Data Storage

Put at the back of your mind that computer data storage often called storage or memory is a technology consisting of computer components and recording media used to retain digital data. As said earlier, it is a core function and fundamental component of computers. In present-day usage, memory is usually semiconductor storage read-write random-access memory, typically DRAM (Dynamic-RAM) or other forms of fast but temporary storage.

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 *core*, *main memory*, *real storage* or *internal memory* while storage devices have been referred to as *secondary storage*, *external memory* or *auxiliary/peripheral storage*.

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. Many different forms of storage, based on various natural occurrences, have been invented.

So far, no practical universal storage medium exists, and all forms of storage have some drawbacks. Therefore, a computer system usually contains several kinds of storage, each with an individual purpose. You may have heard that the modern digital computer represents data using the binary numeral system. Text, numbers, pictures, audio, and nearly any other form of information can be converted into a string of bits, or binary digits, each of which has a value of 1 or 0.

However, 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. For example, the complete works of Shakespeare, about 1250 pages in print, can be stored in about five megabytes (forty million bits) with one byte per character.

Self-Assessment Question

1. What is Computer Data Storage?

Self-Assessment Answer

1. Computer data storage often called storage or memory is a technology consisting of computer components and recording media used to retain digital data

3.2 Fundamental Storage Technologies

Let me say that as of 2011[update], the most commonly used data storage technologies 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

Do you 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 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. Non-volatile semiconductor memory is also used for secondary storage in various advanced electronic devices and specialized computers.

I want you to know that 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. Its storage is *non-volatile*. The information is accessed using one or more read/write heads which may contain one or more recording transducers. 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:

1. Magnetic disk

   Floppy disk, used for off-line storage

Hard disk drive, used for secondary storage

2 Magnetic tape, used for tertiary and off-line storage

Know that in early computers, magnetic storage was also used as:

1. Primary storage in a form of magnetic memory, or core memory, core rope memory, thin-film memory and/or twist or memory.
2. Tertiary (e.g. NCR CRAM) or off line storage in the form of magnetic cards.
3. 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:

1. CD, CD-ROM, DVD, BD-ROM: Read only storage, used for mass distribution of digital information (music, video, computer programs)
2. CD-R, DVD-R, DVD+R, BD-R: Write once storage, used for tertiary and off-line storage
3. CD-RW, DVD-RW, DVD+RW, DVD-RAM, BD-RE: Slow write, fast read storage, used for tertiary and off-line storage
4. 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.

Example:

If I ask you what forms is the magnetic storage in modern computers?

Answer:

All I want to see is in modern computers, magnetic storage will take these forms:

1. Magnetic disk
    - Floppy disk, used for off-line storage
    - Hard disk drive, used for secondary storage
2. Magnetic tape, used for tertiary and off-line storage

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.

Self-Assessment Question(s)

1. List three computer storage technologies you know.

Self-Assessment Answer

1. Semiconductor, Magnetic Storage, Optical Storage

3.3 Related Technologies

Network Connectivity

You will also learn 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.

1. Direct-attached storage (DAS) is a traditional mass storage that does not use any network. This is still a most popular approach. This retronym was coined recently, together with NAS and SAN.
2. Network-attached storage (NAS) is mass storage attached to a computer which 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.
3. 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.

Robotic Storage

Large quantities of individual magnetic tapes, and optical or magneto-optical discs may be stored in robotic tertiary storage devices. In tape storage field they are known as tape libraries, and in optical storage field optical jukeboxes, or optical disk libraries per analogy. Smallest forms of either technology containing just one drive device are referred to as autoloaders or auto-changers.

Robotic-access storage devices may have a number of slots, each holding individual media, and usually one or more picking robots that traverse the slots and load media to built-in drives. The arrangement of the slots and picking devices affects performance. Important characteristics of such storage are possible expansion options: adding slots, modules, drives, robots. Tape libraries may have from 10 to more than 100,000 slots, and provide terabytes or petabytes of near-line information. Optical jukeboxes are somewhat smaller solutions, up to 1,000 slots.

Robotic storage is used for backups, and for high-capacity archives in imaging, medical, and video industries. Hierarchical storage management is a most known archiving strategy of automatically *migrating* long-unused files from fast hard disk storage to libraries or jukeboxes. If the files are needed, they are *retrieved* back to disk.

4.0 Conclusion

In this unit, you have learned about the storage systems and their technologies. You have also learnt about computer data storage, often called storage or memory, fundamental storage technologies and related technologies.

5.0 Summary

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. As of 2011, the most commonly used data storage technologies are semiconductor, magnetic, and optical, while paper still sees some limited usage.

Media is a common 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. 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. 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*. 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. Large quantities of individual magnetic tapes, and optical or magneto-optical discs may be stored in robotic tertiary storage devices.

6.0 Tutor-Marked Assignment

i. Explain how a modern digital computer presents data?

ii. Describe magnetic storage with their forms?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

NOUN, (2008). Introduction to Computer Organisation. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island,Lagos. First Printed 2008

Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.

Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd ed) Prentice Hall of India.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

Unit 2

Data Compression and Data Integrity

Contents

1.0 Introduction

In this unit what you will learn borders on data compression and data integrity. In computer science and information theory, data compression, source coding, or bit-rate reduction involves encoding information using fewer bits than the original representation. You will also learn about lossless data compression, lossy data compression and data integrity- error checking.

2.0 Learning Outcome

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

i. Explain data compression

ii. Describe lossless data compression

iii. Explain lossy data compression

iv. Explain data integrity – error checking

3.0 Learning Content

3.1 What is Data Compression?

In computer science and information theory, data compression, source coding, or bit-rate reduction involves encoding information using fewer bits than the original representation. So compression can be either lossy or lossless. You should know that lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying marginally important information and removing it.

The process of reducing the size of a data file is popularly referred to as data compression, although it's formal name is source coding (coding done at the source of the data, before it is stored or transmitted). Compression is useful because it helps reduce resources usage, such as data storage space or transmission capacity. Because compressed data must be decompressed to be used, this extra processing imposes computational or other costs through decompression, this situation is far from being a free lunch.

Data compression is subject to a space-time complexity trade-off. For instance, a compression scheme for video may require expensive hardware for the video to be decompressed fast enough to be viewed as it is being decompressed, and the option to decompress the video in full before watching it may be inconvenient or require additional storage.

The design of data compression schemes involve trade-offs among various factors, including the degree of compression, the amount of distortion introduced (*e.g.*, when using lossy data compression), and the computational resources required to compress and uncompress the data.

New alternatives to traditional 'sample-sense-compress' systems (which sample a full resolution then compress), provide efficient resource usage based on principles of compressed sensing. Compressed sensing techniques circumvent the need for data-compression by sampling off a cleverly selected basis.

Self-Assessment Question

1. Compression can be _____ or _____.

Self-Assessment Answer

1. Lossy and Lossless

3.2 Lossless Data Compression

Lossless data compression algorithms usually exploit statistical redundancy to represent data more concisely without losing information. Lossless compression is possible because most real-world data has statistical redundancy. For example, an image may have areas of colour that do not change over several pixels; instead of coding "red pixel, red pixel ..." the data may be encoded as "279 red pixels".

This is a simple example of run-length encoding; there are many schemes to reduce size by eliminating redundancy. The Lempel–Ziv (LZ) compression methods are among the most popular algorithms for lossless storage. DEFLATE is a variation on LZ which is optimized for decompression speed and compression ratio, but compression can be slow. DEFLATE is used in PKZIP, gzip and PNG. LZW (Lempel–Ziv–Welch) is used in GIF images.

Also noteworthy are the LZR (LZ–Renau) methods, which serve as the basis of the Zip method. LZ methods use a table-based compression model where table entries are substituted for repeated strings of data. For most LZ methods, this table is generated dynamically from earlier data in the input. The table itself is often Huffman encoded (e.g. SHRI, LZX). A current LZ-based coding scheme that performs well is LZX, used in Microsoft's CAB format.

The very best modern lossless compressors use probabilistic models, such as prediction by partial matching. The Burrows–Wheeler transform can also be viewed as an indirect form of statistical modelling.

The class of grammar-based codes are recently noticed because they can extremely compress *highly repetitive text*, for instance, biological data collection of same or related species, huge versioned document collection, internet archives, etc. The basic task of grammar-based codes is constructing a context-free grammar deriving a single string. Sequitur and Re-Pair are practical grammar compression algorithms for which public codes are available.

In a further refinement of these techniques, statistical predictions can be coupled to an algorithm called arithmetic coding. Arithmetic coding, invented by Jorma Rissanen, and turned into a practical method by Witten, Neal, and Cleary, achieves superior compression to the better-known Huffman algorithm, and lends itself especially well to adaptive data compression tasks where the predictions are strongly context-dependent. Arithmetic coding is used in the bilevel image-compression standard JBIG, and the document-compression standard DjVu. The text entry system, Dasher, is an inverse-arithmetic-coder.

Self-Assessment Question(s)

1. Lossless data compression algorithms usually exploit _____ to represent data more concisely without losing information.

Self-Assessment Answer

1. Statistical redundancy

3.3 Lossy Data Compression

Lossy data compression is contrasted with lossless data compression. In these schemes, some loss of information is acceptable. Depending upon the application, detail can be dropped from the data to save storage space. Generally, lossy data compression schemes are guided by research on how people perceive the data in question.

For example, the human eye is more sensitive to subtle variations in luminance than it is to variations in color. JPEG image compression works in part by "rounding off" less-important visual information. There is a corresponding trade-off between information lost and the size reduction. A number of popular compression formats exploit these perceptual differences, including those used in music files, images, and video.

Another improvement is lossy image compression can be used in digital cameras, to increase storage capacities with minimal degradation of picture quality. Similarly, DVDs use the lossy MPEG-2 Video codec for video compression. In lossy audio compression, methods of psychoacoustics are used to remove non-audible (or less audible) components of the signal.

Compression of human speech is often performed with even more specialized techniques, so that "speech compression" or "voice coding" is sometimes distinguished as a separate discipline from "audio compression". Different audio and speech compression standards are listed under audio codecs. Voice compression is used in Internet telephony for example, while audio compression is used for CD ripping and is decoded by audio players.

Self-Assessment Question

1. In Lossy Data Compression, some level of data loss is acceptable. (True/False)

Self-Assessment Answer

1. True

3.4 Data Integrity - Error Checking

Be aware that, ensuring the integrity of data stored in memory is an important aspect of memory design. Two primary means of accomplishing this are **parity** and **error correction code (ECC)**.

Historically, **parity** has been the most commonly used data integrity method. Parity can detect - but not correct - single-bit errors. **Error Correction Code (ECC)** is a more comprehensive

method of data integrity checking that can detect and correct single-bit errors. Fewer and fewer PC manufacturers are supporting data integrity checking in their designs. This is due to a couple of factors. First, by eliminating support for parity memory, which is more expensive than standard memory, manufacturers can lower the price of their computers. Fortunately, this trend is complemented by the second factor: that is, the increased quality of memory components available from certain manufacturers and, as a result, the relative infrequency of memory errors.

The type of data integrity checking depends on how your computer system will be used. If the computer is to play a critical role - as a server, for example - then a computer that supports data integrity checking is an ideal choice. In general:

1. Most computers designed for use as high-end servers support ECC memory.
2. Most low-cost computers designed for use at home or for small businesses support non-parity memory.

Self-Assessment Question

1. The means of ensuring the integrity of stored data are primarily group into _____.

Self-Assessment Answer

1. Parity and Error Correction Code (ECC)

Parity

Try to understand that when parity is in use on a computer system, one parity bit is stored in DRAM along with every 8 bits (1 byte) of data. The two types of parity protocol exist - odd parity and even parity – and function in similar ways. With normal parity, when 8 bits of data are written to DRAM, a corresponding parity bit is written at the same time.

The value of the parity bit (either a 1 or 0) is determined at the time the byte is written to DRAM, based on an odd or even quantity of 1s. Some manufacturers use a less expensive "fake parity" chip. This chip simply generates a 1 or a 0 at the time the data is being sent to the CPU in order to accommodate the memory controller. (For example, if the computer uses odd parity, the fake parity chip will generate a 1 when a byte of data containing an even number of 1s is sent to the CPU.

If the byte contains an odd number of 1s, the fake parity chip will generate a 0.) The issue here is that the fake parity chip sends an "OK" signal no matter what. This way, it "fools" a computer that's expecting the parity bit into thinking that parity checking is actually taking place when it is not. The bottom line: fake parity cannot detect an invalid data bit.

This table shows how odd parity and even parity work. The processes are identical but with opposite attributes.

	ODD PARITY	EVEN PARITY
Step 1	The parity bit will be forced to 1 (or turned "on") if its corresponding byte of data contains an even number of 1's. If the byte contains an odd number of 1's, the parity bit is forced to 0 (or turned "off ").	The parity bit is forced to 0 if the byte contains an even number of 1's. The parity bit is forced to 1 if its corresponding byte of data contains an odd number of 1's.
Step 2	The parity bit and the corresponding 8 bits of data are written to DRAM.	(Same as for odd parity)
Step 3	Just before the data is sent to the CPU, it is intercepted by the parity circuit. If the parity circuit sees an odd number of 1's, the data is considered valid. The parity bit is stripped from the data and the 8 data bits are passed on to the CPU. If the parity circuit detects an even number of 1's, the data is considered invalid and a parity error is generated.	(Same as for odd parity) Data is considered valid if the parity circuit detects an even number of 1's. Data is invalid if the parity circuit detects an odd number of 1's.

Parity does have its limitations. For example, parity can detect errors but cannot make corrections. This is because the parity technology can't determine which of the 8 data bits are invalid. Furthermore, if multiple bits are invalid, the parity circuit will not detect the problem if the data matches the odd or even parity condition that the parity circuit is checking for. For example, if a valid 0 becomes an invalid 1 and a valid 1 becomes an invalid 0, the two defective bits cancel each other out and the parity circuit misses the resulting errors. Fortunately, the chances of this happening are extremely remote.

Self-Assessment Question

1. What are the two types of Parity Check you know?

Self-Assessment Answer

1. Even and Odd Parity.

Error Correction Code (ECC)

Error Correction Code is the data integrity checking method used primarily in high-end PCs and file servers. The important difference between ECC and parity is that ECC is capable of detecting and correcting 1-bit errors. With ECC, 1-bit error correction usually takes place without the user even knowing an error has occurred. Depending on the type of memory controller the computer uses, ECC can also detect rare 2-, 3-, or 4-bit memory errors.

While ECC can detect these multiple-bit errors, you should know that they cannot correct them. However, there are some more complex forms of ECC that can correct multiple bit

errors. Using a special mathematical sequence, algorithm, and working in conjunction with the memory controller, the ECC circuit appends ECC bits to the data bits, which are stored together in memory.

When the CPU requests data from memory, the memory controller decodes the ECC bits and determines if one or more of the data bits are corrupted. If there's a single-bit error, the ECC circuit corrects the bit. In the rare case of a multiple-bit error, the ECC circuit reports a parity error.

Self-Assessment Question

1. ECC can correct ___ bit(s) in error.

Self-Assessment Answer

1. One

4.0 Conclusion

What you have learnt in this unit is on data compression and data integrity. You have also learnt about lossless data compression, lossy data compression and data integrity- error checking.

5.0 Summary

In computer science and information theory, data compression, source coding, or bit-rate reduction involves encoding information using fewer bits than the original representation. Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying marginally important information and removing it.

Ensuring the integrity of data stored in memory is an important aspect of memory design. Two primary means of accomplishing this are parity and error correction code (ECC). Historically, parity has been the most commonly used data integrity method. Parity can detect - but not correct - single-bit errors. Error Correction Code (ECC) is a more comprehensive method of data integrity checking that can detect and correct single-bit errors.

When parity is in use on a computer system, one parity bit is stored in DRAM along with every 8 bits (1 byte) of data. The two types of parity protocol - odd parity and even parity - function in similar ways. Error Correction Code is the data integrity checking method used primarily in high-end PCs and file servers. The important difference between ECC and parity is that ECC is capable of detecting and correcting 1-bit errors.

6.0 Tutor-Marked Assignment

1. Explain parity in data integrity?
2. What do you understand by lossy data compression?
3. Explain data compression and its usefulness in computer science?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

NOUN, (2008). INTRODUCTION TO COMPUTER ORGANISATION. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island,Lagos. First Printed 2008

Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.

Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd ed) Prentice Hall of India.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_umg_05_005

Unit 3

Memory Hierarchy, Organization and Operations

Contents

1.0 Introduction

In this unit what you will learn concerns memory hierarchy, organization and operations. Memory in a computer system is required for storage and subsequent retrieval of the instructions and data. A computer system uses variety of devices for storing these instructions and data which are required for its operations. Normally we classify the information to be stored on computer in two basic categories:

Data and the Instructions. “The storage devices along with the algorithm or information on how to control and manage these storage devices constitute the memory system of a computer. “A memory system is a very simple system yet it exhibits a wide range of technology and types.

2.0 Learning Outcome

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

i. Explain memory hierarchy

ii. Describe characteristics terms for various memory devices

iii. Explain memory interleaving

3.0 Learning Content

3.1 Memory Hierarchy

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.

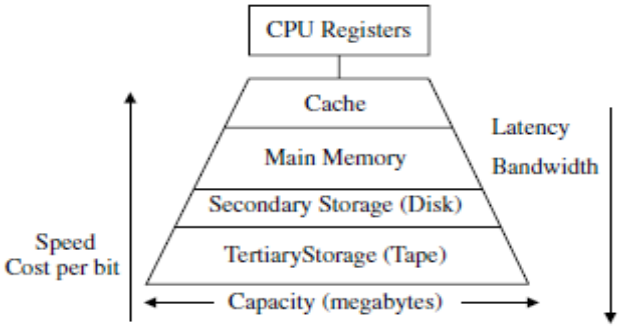


Figure 1: Typical Memory Hierarchy

Self-Assessment Question

1. Why is the memory hierarchy employed in memory architecture?

Self-Assessment Answer

1. 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

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

3.1.1 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.

3.1.2 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).

3.1.3 Secondary Memory/Auxiliary Memory/Backing Store:

Auxiliary memory in fact is much larger in size than main memory but is slower than main memory. It normally stores system programs (programs which are used by 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 main memory. There is another kind of memory which is increasingly being used in modern computers, this is called Cache memory. It is logically positioned between the internal memory (registers) and 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 about this memory in greater details in a subsequent section of this unit.

Self-Assessment Question

1. List the three groups of memories that constitute a memory system.

Self-Assessment Answer

1. Internal Processor Memory, Primary (Main) Memory, Secondary Memory.

3.2 Characteristics Terms for Various Memory Devices

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.

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 a consecutive access to location 101 takes 505 ns, then it is expected that an 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 of time, programs tend to reference a relatively confined area of memory repeatedly. There exist two forms of locality: spatial and temporal locality.

Table 1: Memory Hierarchy Parameters

	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

Example:

Give any example of volatile memory in computer?

Answer:

Random Access Memory (RAM)

3.3 Memory Interleaving

One possible technique that you will use to increase the bandwidth is memory interleaving. To achieve 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. Figure 2 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.

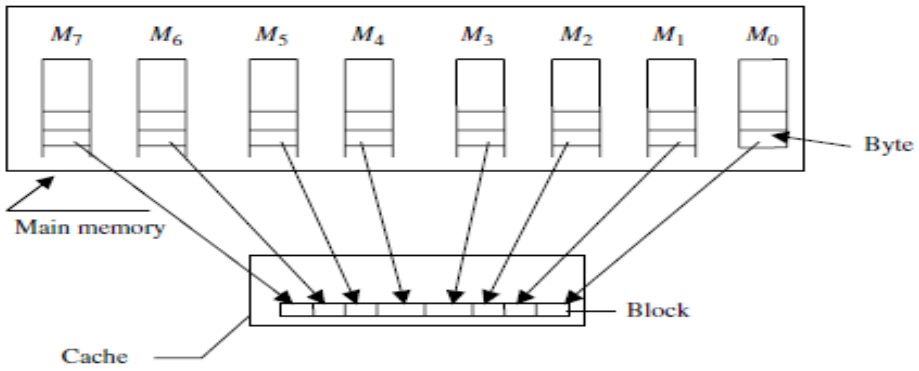


Figure 2: Memory interleaving using eight modules

Self-Assessment Question

1. 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. True/False?

Self-Assessment Answer

1. True

4.0 Conclusion

What you have learnt in this unit is on memory hierarchy, organization and operations. Also, you have learnt about memory hierarchy, internal processor memories, primary memory or main memory, secondary memory/auxiliary memory/backing Store, characteristics terms for various memory devices and memory interleaving.

5.0 Summary

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 memory hierarchy can be characterized by a number of parameters.

Among these parameters are the access type, capacity, cycle time, latency, bandwidth, and cost. One possible technique that is used to increase the bandwidth is memory interleaving. To achieve 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.

6.0 Tutor-Marked Assignment

1. Explain memory hierarchy with the aid of diagram?
2. Explain the characteristics terms for various memory devices?

7.0 References/Further Reading

Daniel P, and David G, (2009). A Practical Introduction to Computer Architecture. Text in Computer Science, Springer Dordrecht Heidelberg London New York.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. (2002): Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157

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

Hennessy J.L and Patterson, D.A.(2006). *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann. ISBN: 0-123-70490-1.

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.

Murdocca M. J. and Heuring, V. P. (1999). Principles of Computer Architecture. (Class test edition). Prentice Hall.

NOUN, (2008). INTRODUCTION TO COMPUTER ORGANISATION. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island,Lagos. First Printed 2008

Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.

Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd ed) Prentice Hall of India.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_umg_05_005

Unit 4

Cache Memory and Virtual Memory

Contents

1.0 Introduction

In this unit what you will learn concerns cache memory and virtual memory. Cache memories are small fast memories placed between the processor and the main memory. Caches are faster than main memory, the caches although are fast yet are very expensive memories and are used in only small sizes.

The virtual memory is a memory management technique which does splitting of a program into number of pieces as well as swapping. The basic idea behind virtual memory is that the combined size of the program, data and stack may exceed the amount of physical memory.

2.0 Learning Outcome

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

i. Explain cache memory
ii. Describe associative mapped cache
iii. Explain direct mapped cache
iv. Explain set associative mapped cache
v. Describe cache performance
vi. Explain hit ratios and effective access times
vii. Describe virtual memory
viii. Explain overlays
ix. Explain paging
x. Describe segmentation
xi. Explain fragmentation

3.0 Learning Content

3.1 Cache Memory

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 in 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, as you will learn, 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 repetition or in recursion, and thus the same section of code is visited an unduly large number of times.

Spatial locality arises because data tends to be stored in contiguous locations. Although 10% of the code accounts for the bulk of memory references, accesses within the 10% tend to be

clustered. Thus, for a given interval of time, most of memory accesses come from an even smaller set of locations than 10% of a program's size. 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. 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. The word is then accessed in the cache. Although this process takes longer than accessing 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 the L1 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. A cache memory is faster than main memory for a number of 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. A cache memory has fewer locations than a 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 7-12. A simple computer without a cache memory is shown in the left side of the figure.

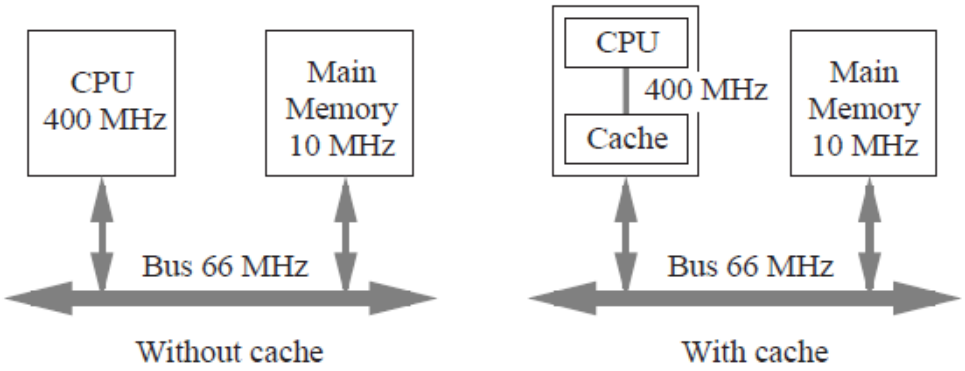


Figure 7-12 Placement of cache in a computer system.

This cache-less computer contains a CPU that has a clock speed of 400 MHz, but communicates over a 66 MHz bus to a main memory that supports a lower clock speed of 10 MHz. A few bus cycles are normally 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.

A cache memory can be positioned closer to the CPU as shown in the right side of Figure 7-12, so that the CPU sees fast accesses over a 400 MHz direct path to the cache.

Self-Assessment Question

1. What is the Memory Locality Principle?

Self-Assessment Answer

1. 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 in just 10% of the code. This property is known as the locality principle.

3.1.1 Associative Mapped Cache

A number of hardware schemes have been developed for translating main memory addresses to cache memory addresses. The user does not need to know about the address translation, which has the advantage that cache memory enhancements can be introduced into a computer without a corresponding need for modifying application software. The choice of cache mapping scheme affects cost and performance, and there is no single best method that is appropriate for all situations. In this section, an **associative** mapping scheme is studied.

Figure 7-13 shows an associative map

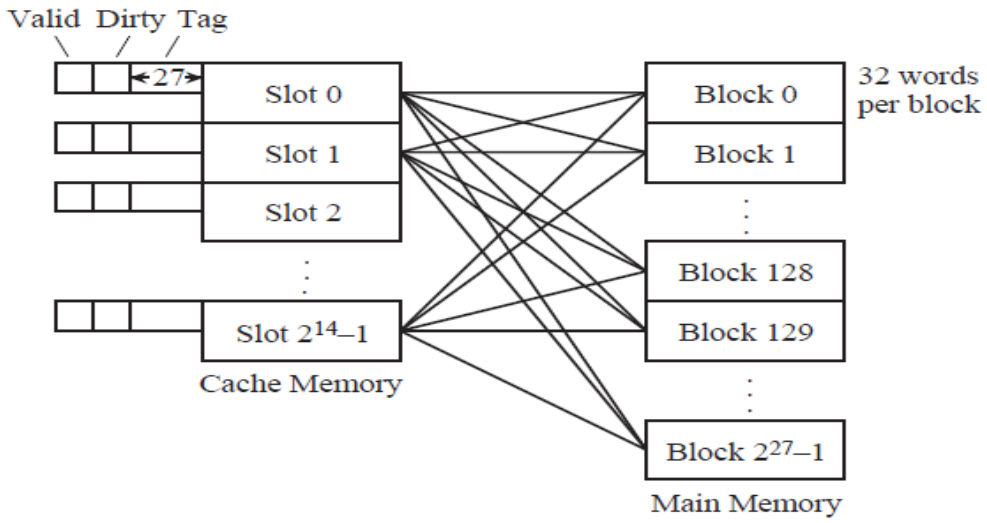


Figure 7-13 An associative mapping scheme for a cache memory.

Mapping scheme for a 2^{32} word memory space that is divided into 2^{27} **blocks** of 2^5 = 32 words per block. The main memory is not physically partitioned in this way, but this is the view of main memory that the cache sees. Cache blocks, or **cache lines**, as they are also known, typically range in size from 8 to 64 bytes. Data is moved in and out of the cache a line at a time using memory interleaving (discussed earlier).

However, the cache for this example consists of 2^{14} **slots** into which main memory blocks are placed. There are more main memory blocks than there are cache slots, and any one of the 2^{27} main memory blocks can be mapped into each cache slot (with only one block placed in a

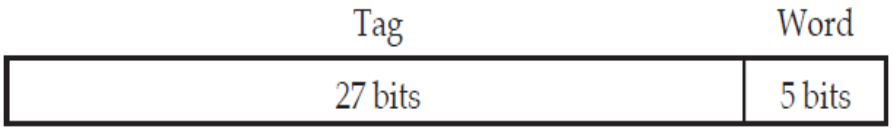
slot at a time). To keep track of which one of the 2^{27} possible blocks is in each slot, you need to know that a 27-bit **tag** field is added to each slot which holds an identifier in the range from 0 to $2^{27} - 1$.

The tag field is the most significant 27 bits of the 32-bit memory address presented to the cache. All the tags are stored in a special tag memory where they can be searched in parallel. Whenever a new block is stored in the cache, its tag is stored in the corresponding tag memory location.

When you first load a program into main memory, the cache is cleared, and so while a program is executing, a **valid** bit is needed to indicate whether or not the slot holds a block that belongs to the program being executed. There is also a **dirty** bit that keeps track of whether or not a block has been modified while it is in the cache. A slot that is modified must be written back to the main memory before the slot is reused for another block.

A referenced location that is found in the cache results in a **hit**, otherwise, the result is a **miss**. When a program is initially loaded into memory, the valid bits are all set to 0. The first instruction that is executed in the program will therefore cause a miss, since none of the program is in the cache at this point. The block that causes the miss is located in the main memory and is loaded into the cache.

In an associative mapped cache, each main memory block can be mapped to any slot. The mapping from main memory blocks to cache slots is performed by partitioning an address into fields for the tag and the word (also known as the “byte” field) as shown below:



When a reference is made to a main memory address, the cache hardware intercepts the reference and searches the cache tag memory to see if the requested block is in the cache. For each slot, if the valid bit is 1, then the tag field of the referenced address is compared with the tag field of the slot. All of the tags are searched in parallel, using an **associative memory** (which is something different than an associative mapping scheme.)

If any tag in the cache tag memory matches the tag field of the memory reference, then the word is taken from the position in the slot specified by the word field. If the referenced word is not found in the cache, then the main memory block that contains the word is brought into the cache and the referenced word is then taken from the cache. The tag, valid, and dirty fields are updated, and the program resumes execution.

Replacement Policies in Associative Mapped Caches

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 time stamp 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 time stamp, 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 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 to determine 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 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 an associative memory. This kind of cache is referred to as a **direct mapped cache**, which is discussed in the next section.

Self-Assessment Question

1. What are the four replacement policies in Associative Mapped Caches?

Self-Assessment Answer

1. Least recently used (LRU), first-in first-out (FIFO), least frequently used (LFU), and random.

3.1.2 Direct Mapped Cache

Figure 7-14 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 $2^{13} – 1$.

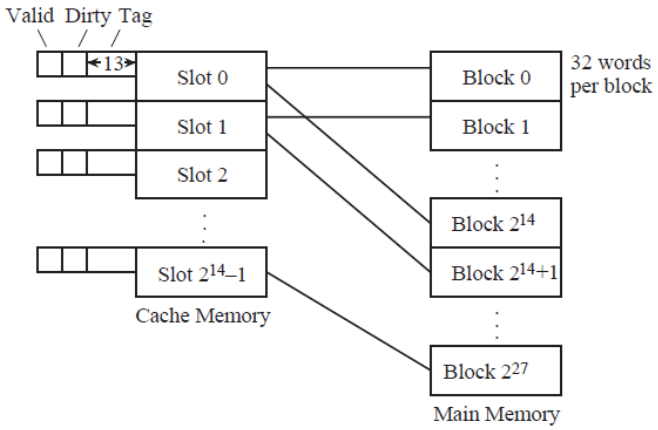


Figure 7-14 A direct mapping scheme for cache memory.

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 2^{14} 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.

Tag	Slot	Word
13 bits	14 bits	5 bits

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.

This simplicity comes at a cost. Consider what happens when a program references locations that are 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, but in fact, fast matrix calculations use power-of-two dimensions (which allows shift operations to replace costly multiplications and divisions for array indexing), and 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 a fully associative cache memory, the **set associative mapping** scheme can be used, which combines aspects of both direct mapping and associative mapping. Set associative mapping, which is also known as **set-direct mapping**, is described in the next section.

Self-Assessment Question

1. What is the advantage of Direct Mapped Cache?

Self-Assessment Answer

1. Simplicity in implementation

3.1.3 Set Associative Mapped Cache

The set associative mapping scheme combines the simplicity of direct mapping with the flexibility of associative mapping. Set associative mapping is more practical than fully associative mapping because the associative portion is limited to just a few slots that make up a set, as illustrated in Figure 7-15. For this example,

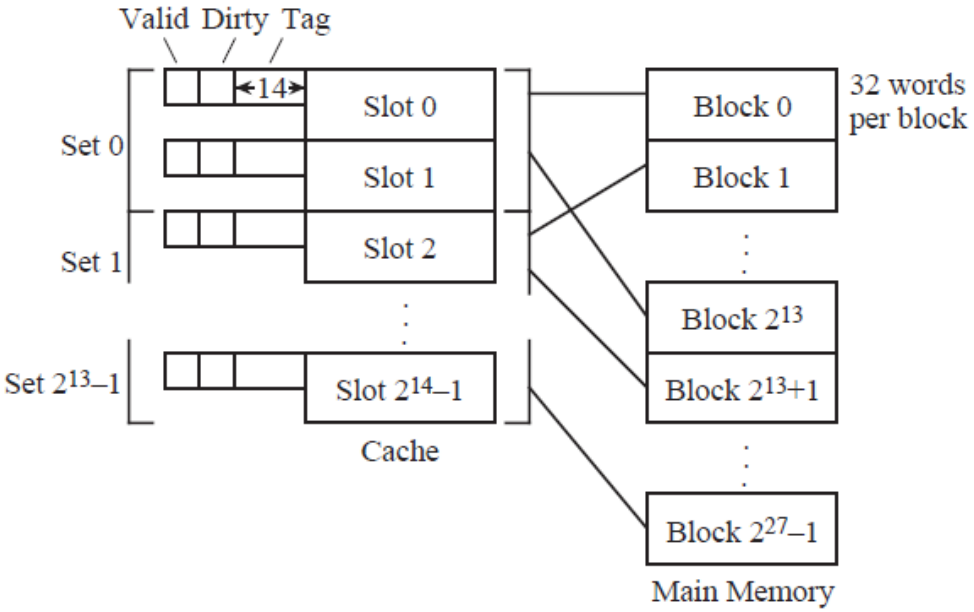
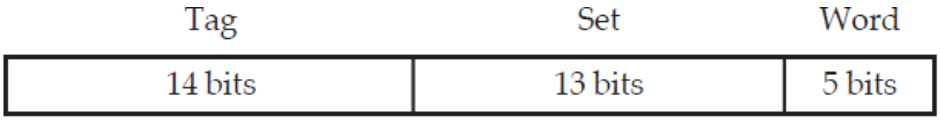


Figure 7-15 A set associative mapping scheme for a cache memory.

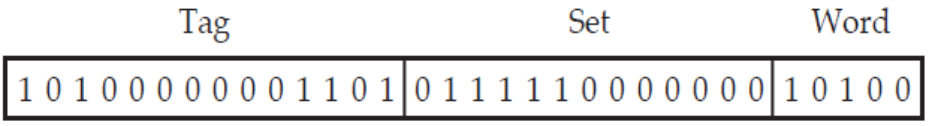
two blocks make up a set, and so it is a **two-way** set associative cache. If there are four blocks per set, then it is a four-way set associative cache. Since there are 2^{14} slots in the cache, there

are $2^{14}/2 = 2^{13}$ sets. When an address is mapped to a set, the direct mapping scheme is used, and then associative mapping is used within a set.

The format for an address has 13 bits in the set field, which identifies the set in which the addressed word will be found if it is in the cache. There are five bits for the word field as before and there is a 14-bit tag field that together make up the remaining 32 bits of the address as shown below:



As an example of how the set associative cache views a main memory address, consider again the address (A035F014)16. The leftmost 14 bits form the tag field, followed by 13 bits for the set field, followed by five bits for the word field as shown below:



As before, the partitioning of the address field is known only to the cache, and the rest of the computer is oblivious to any address translation. *Advantages and Disadvantages of the Set Associative Mapped Cache* In the example above, the tag memory increases only slightly from the direct mapping example, to 13 x2^{14} bits, and only two tags need to be searched for each memory reference. The set associative cache is almost universally used in today's microprocessors.

Self-Assessment Question

1. The set associative mapping scheme combines the simplicity of ___with the flexibility of __.

Self-Assessment Answer

1. Direct mapping, Associative mapping

3.1.4 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 a 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.

Cache read and write policies are summarized in Figure 7-16. 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.

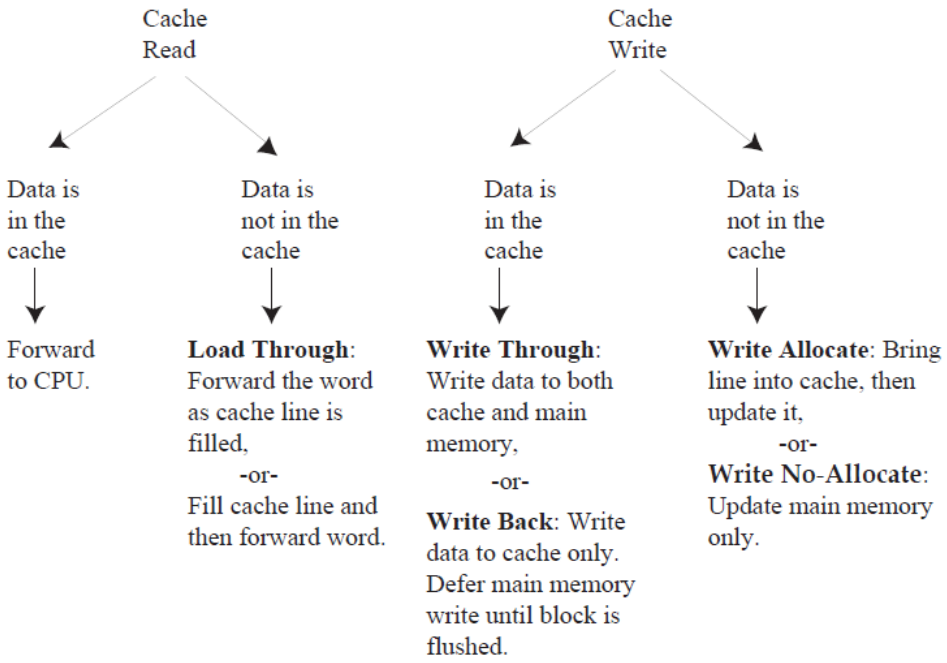


Figure 7-16 Cache read and write policies.

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 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 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 main memory rather than toward 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 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 normally determined by the results of simulation runs that expose the nature of memory traffic.

Self-Assessment Question

1. The main reason behind using Cache memory is to improve runtime performance. True/False.

Self-Assessment Answer

1. True

3.1.5 Hit Ratios and Effective Access Times

Two measures that characterize the performance of a 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 layout of the cache and the main memory are shown in Figure 7-17. 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 7-18. 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.

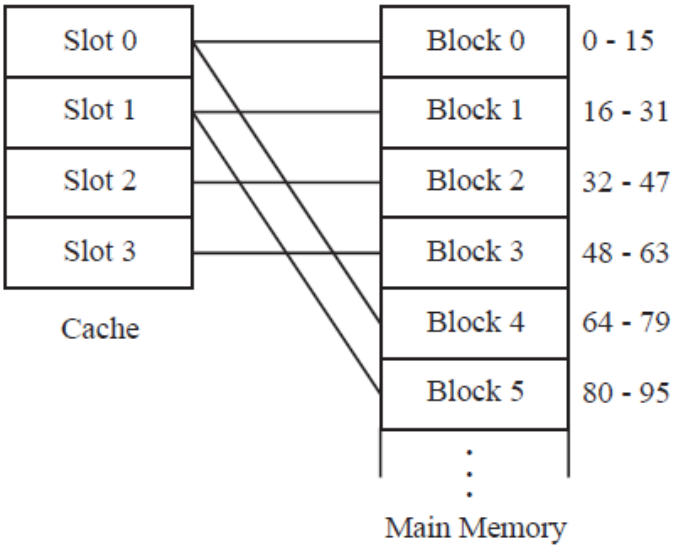


Figure 7-17 An example of a direct mapped cache memory.

Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	80ns×15=1200ns	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	80ns×15=1200ns	
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 7-18 A table of events for a program executing on an 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 in accessing a block from main memory.

Self-Assessment Question

1. What are the two measures that characterize cache memory performance?

Self-Assessment Answer

1. Hit ratio and the effective access time.

3.2 Virtual Memory

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 main memory, and so it is 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 virtual memory, complex address mapping schemes are supported, which give greater flexibility in how the memory is used. We explore these aspects of virtual memory below.

3.2.1 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 7-20 shows an example in which a program contains a 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.

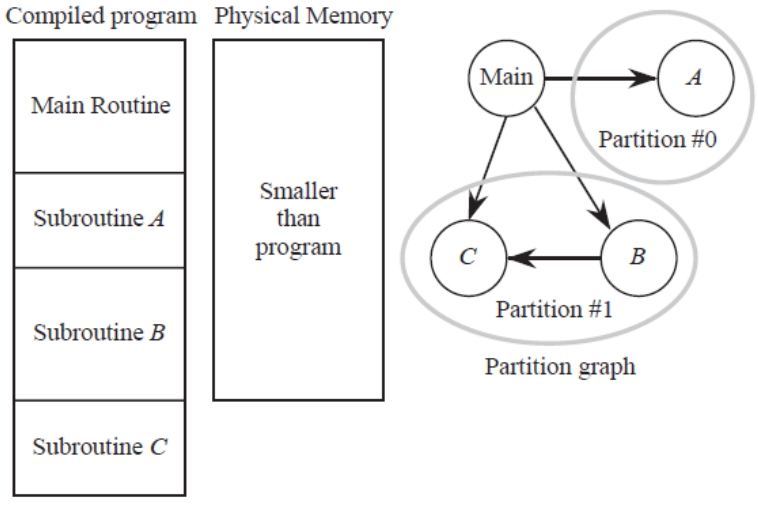


Figure 7-20 A partition graph for a program with a main routine and three subroutines.

Typically, the main routine serves as the **driver** and manages the bulk of the bookkeeping. The driver stays in memory while other routines are brought in and out.

Figure 7-20 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. When more than one program is loaded into memory, however, 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.

Self-Assessment Question

1. An early approach of using disk storage to augment the main memory made use of_____,

Self-Assessment Answer

1. Overlays

3.2.2 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 normally 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 normally stored on a disk.

An illustration of a virtual memory mapping scheme is shown in Figure 7-21. Eight virtual pages are mapped to four physical **page frames**.

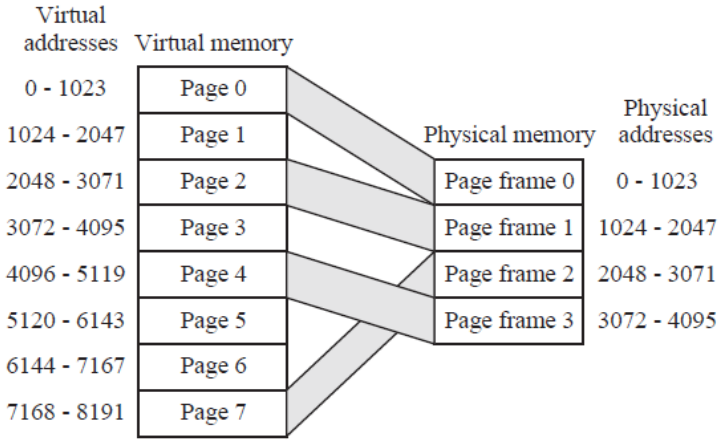


Figure 7-21 A mapping between a virtual and a physical memory.

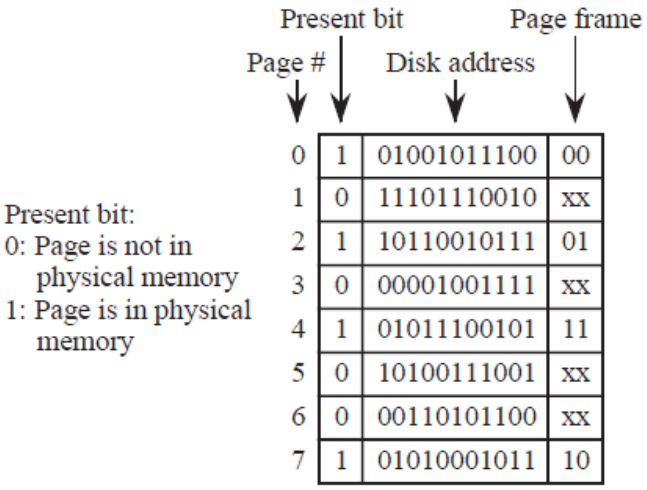
An implementation of virtual memory must handle references that are made outside of the portion of virtual space that is mapped to physical space. The following sequence of events is typical when a referenced virtual location is not in physical memory, which is referred to as a

Page fault:

1. A page frame is identified to be overwritten. The contents of the page frame are written to secondary memory if changes were 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 secondary memory and is written into 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 7-21, there are 2^{13} = 8192 virtual locations and so 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 it is in secondary memory.

In order to keep track of which pages are in physical memory, a **page table** is maintained, as illustrated in Figure 7-22, which corresponds to the mapping shown in Figure 7-21.



Page #	Present bit	Disk address	Page frame
0	1	01001011100	00
1	0	11101110010	xx
2	1	10110010111	01
3	0	00001001111	xx
4	1	01011100101	11
5	0	10100111001	xx
6	0	00110101100	xx
7	1	01010001011	10

Figure 7-22 A page table for a virtual memory.

The page table has as many entries as there 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 normally manages the disk accesses, and so 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 normally do not change during the course of computation. 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 7-22.

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 7-23, in which a reference is made to virtual address 1001101000101. The three leftmost bits of the virtual address (100) identify the page.

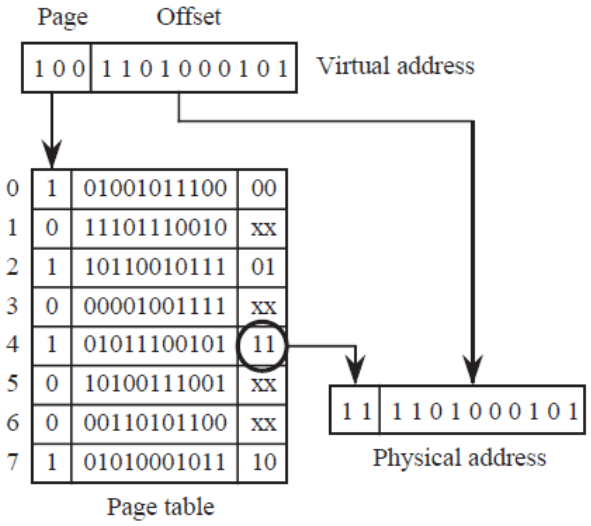


Figure 7-23 A virtual address is translated into a physical address.

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 of time for a program to be loaded into memory.

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 7-21. The size of the virtual address space is 2^{13} words, and the physical address space is 2^{12} words.

There are eight pages that each contain 2^{10} 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 7-24 shows the configuration of the page table as execution proceeds. The first access to memory will cause a page fault on virtual address 1030, which is in page #1. The page is brought into physical memory, and the valid bit and page frame field are updated in the page table.

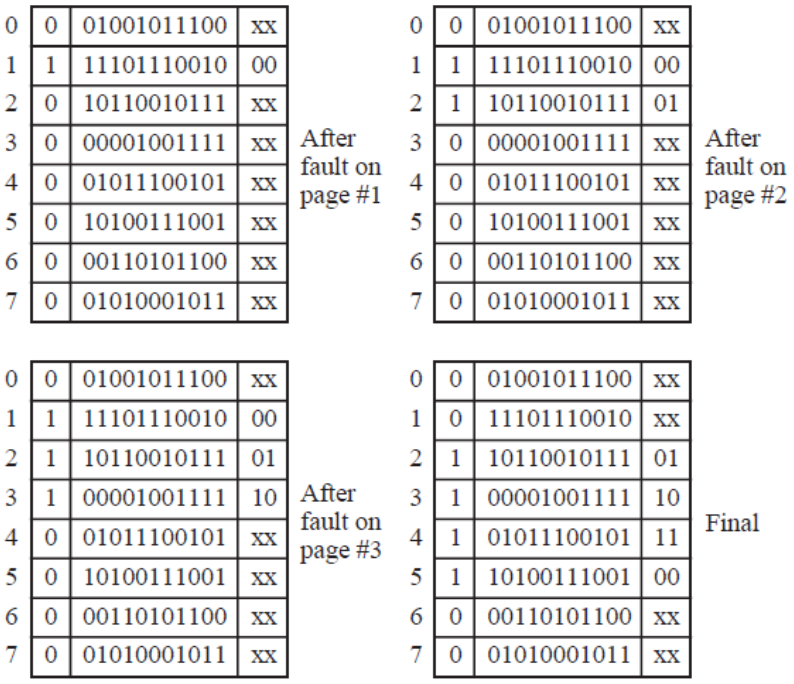


Figure 7-24 The configuration of a page table changes as a program executes. Initially, the page table is empty. In the final configuration, four pages are in physical memory.

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 7-24 is shown after location 5300 is accessed.

3.2.3 Segmentation

Virtual memory as we have discussed it up to this point 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.

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, and so the physical memory devoted to each segment is not committed until needed.

Figure 7-25 illustrates a segmented memory. The executable code for a word pro

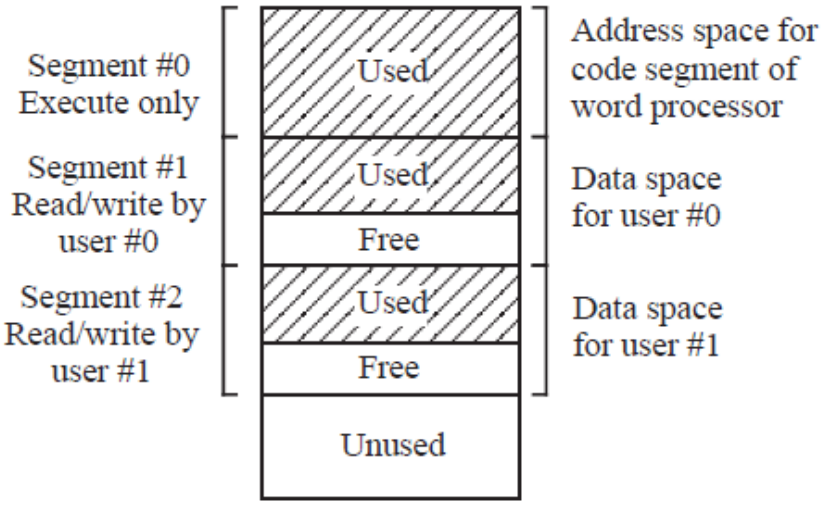


Figure 7-25 A segmented memory allows two users to share the same word processor.

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. 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 normally need to deal with bookkeeping, but 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.

Self-Assessment Questions

1. What is Segmentation?

Self-Assessment Answer

1. Division of the address space into segments.

3.2.4 Fragmentation

When a computer is "booted up," 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 7-26a shows the state of a memory just after the initialization sequence.

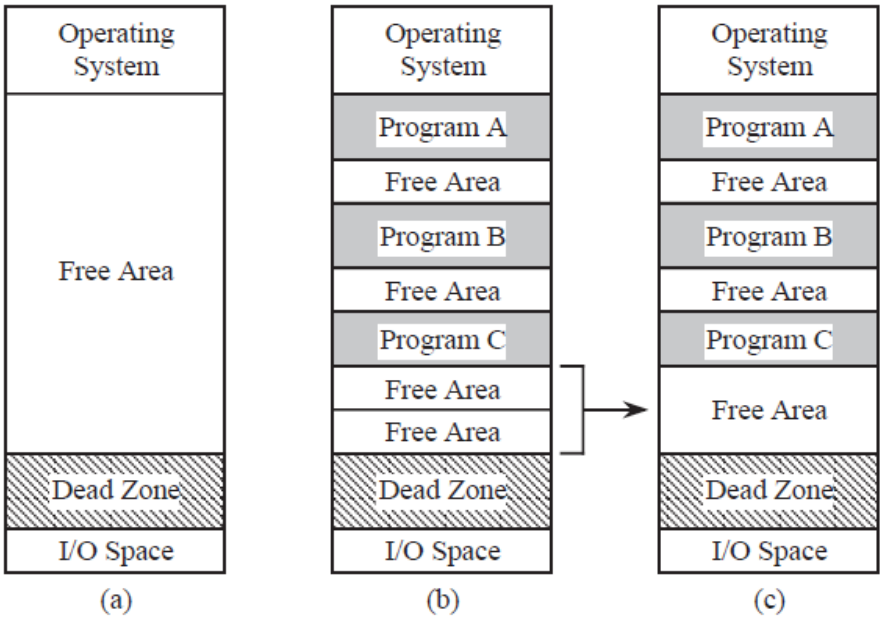


Figure 7-26 (a) Free area of memory after initialization; (b) after fragmentation; (c) after coalescing.

The “Free Area” is a section of memory that is available to the operating system for loading and executing programs. During the course of 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 7-26b 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. 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. In Figure 7-26b, the two adjacent free areas are combined into a single free area, as illustrated in Figure 7-26c.

Self-Assessment Question

1. What are the two methods used for assigning programs to free memory areas?

Self-Assessment Answer

1. First fit and Best fit.

4.0 Conclusion

What you have learnt in this unit is on cache memory and virtual memories. Also, you have learnt about associative mapped cache, direct mapped cache, set associative mapped cache, cache performance, hit ratios and effective access times. You also learnt about overlays, paging, segmentation and fragmentation.

5.0 Summary

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 in 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 iteration or in 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. Although 10% of the code accounts for the bulk of memory references, accesses within the 10% tend to be clustered.

The set associative mapping scheme combines the simplicity of direct mapping with the flexibility of associative mapping.

Set associative mapping is more practical than fully associative mapping because the associative portion is limited to just a few slots that make up a set. Two measures that characterize the performance of a 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.

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 main memory, and so it is reasonable to use disk storage to hold the portions of a program or data sets that do not entirely fit into the main memory.

6.0 Tutor-Marked Assignment

1. Differentiate between the locality principle, temporal locality and spatial locality?
2. Describe replacement policies in associative mapped caches?
3. What are the advantages and disadvantages of the associative mapped cache?

7.0 References/Further Reading

Daniel P, and David G, (2009). A Practical Introduction to Computer Architecture. Text in Computer Science, Springer Dordrecht Heidelberg London New York.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. (2002): Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157

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

Hennessy J.L and Patterson, D. A. (2006). *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann. ISBN: 0-123-70490-1.

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.

Murdocca M. J. and Heuring, V. P. (1999). Principles of Computer Architecture. (Class test edition). Prentice Hall.

NOUN, (2008). INTRODUCTION TO COMPUTER ORGANISATION. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island,Lagos. First Printed 2008

Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.

Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd ed) Prentice Hall of India.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_umg_05_005

Module 3

Interfacing and Communication

Unit 1

Input/output Fundamentals

Contents

1.0 Introduction

In this unit, what you will learn borders on input/output fundamentals. A computer interacts with the external environment via the input-output (I/O) devices attached to it. Input device is used for providing data and instructions to the computer. After processing the input data, computer provides output to the user via the output device. The I/O devices that are attached, externally, to the computer machine are also called *peripheral devices*. Different kinds of input and output devices are used for different kinds of input and output requirements. In this unit, you will learn about how input devices and output devices functions.

2.0 Learning Outcome

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

i. Explain the basic concepts of Input/output
ii. Describe programmed I/O
iii. Explain interrupt-driven I/O
iv. Describe direct memory access (DMA)
v. Explain buses
vi. Describe input-output interfaces

3.0 Learning Content

3.1 Basic Concept of Input/output

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.

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).

Moreso, 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. 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. There is also a number of output registers each belonging to a specific output device.

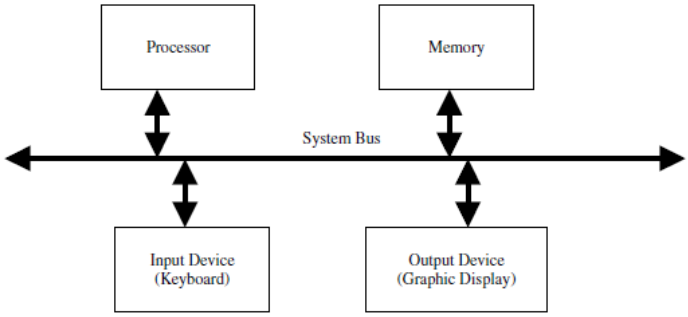


Figure 8.1 A single bus system

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 8.2. 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, be aware that 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).

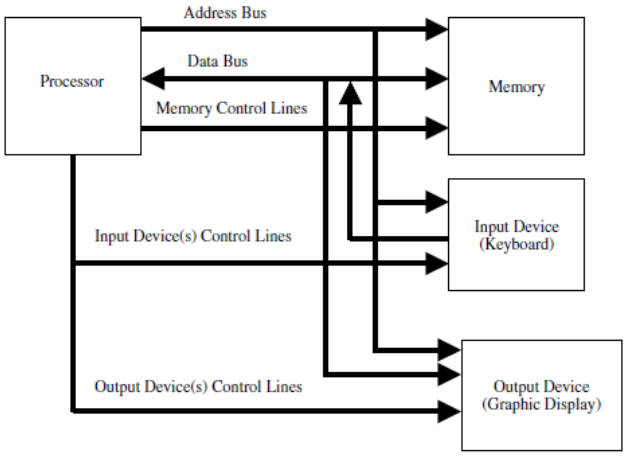


Figure 8.2 Shared I/O arrangement

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 8.3. 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 for introducing 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.

Self-Assessment Question

1. The Input Register is used to accommodate the difference in speed between the processor and input/output devices. True or False?

Self-Assessment Answer

1. True

3.2 Programmed I/O

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

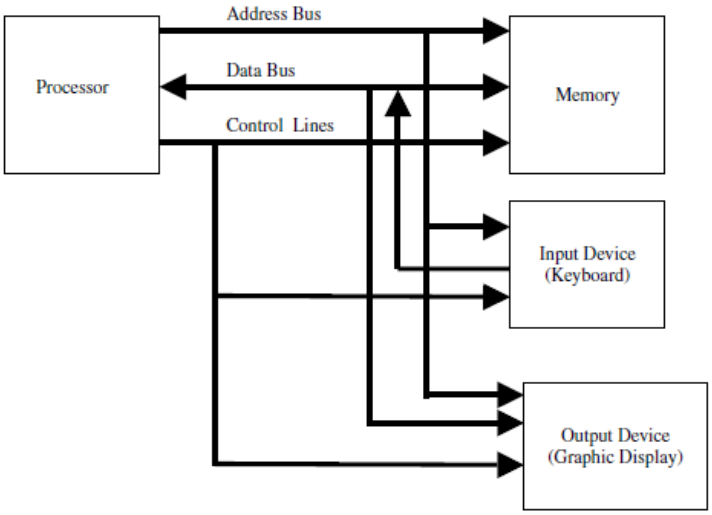


Figure 8.3 Memory-mapped I/O arrangement

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 shared I/O arrangement. Let us also assume that there are eight different I/O devices connected to the processor in this case (see Fig. 8.4). 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 in order 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.

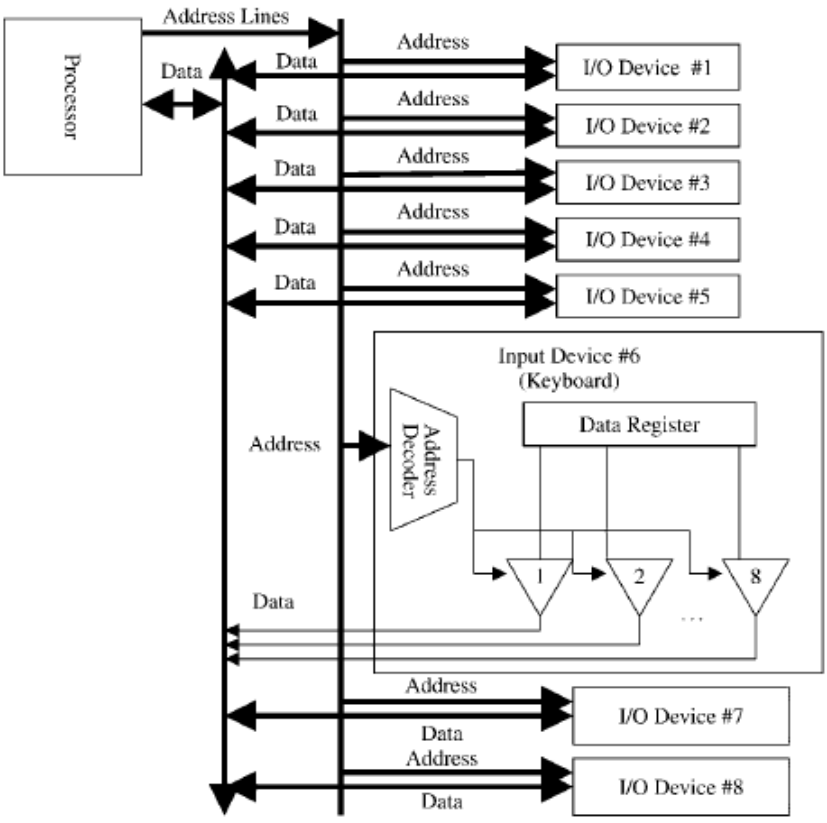


Figure 8.4 Example eight-I/O device connection to a processor

The instruction decoding will gate the data available on the data bus into the input of a particular register in the CPU, normally 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 fetch, 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.

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 continuously monitor Bin. 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 $B_{in} = 0$, 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 $B_{out} = 0$. It is only after the screen has displayed the character that it sets $B_{out} = 1$, 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 8.5.

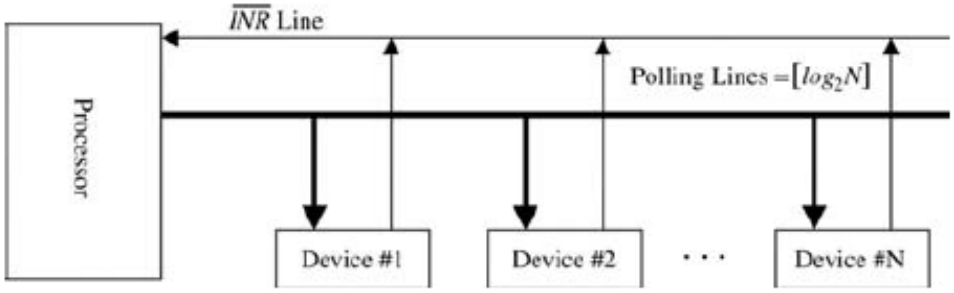


Figure 8.5 Hardware polling scheme

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.

Self-Assessment Question

1. How is the speed difference between the I/O devices and the processor handled?

Self-Assessment Answer

1. The availability of a Status bit in the interface of each input device and likewise in output devices is used to ensure that data is not lost because of the difference in speed. Whenever a device has a character available, it sets its Bin =1. A program running on the processor can then be used to monitoring the status of the Bin.

3.3 Interrupt-Driven I/O

It is often necessary that we 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). 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 priority higher than that of the currently served interrupt, for example, power failure interrupt occurring while serving an I/O interrupt,

then 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 vectored interrupt.

Self-Assessment Question

1. What is the relevance of Interrupt in a Program execution?

Self-Assessment Answer

1. It can be used to make the processor to attend to any urgent activities while postponing any current ongoing activities until the urgency has been taken care of.

3.4 Direct Memory Access (DMA)

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.

When the transfer is complete the DMA surrenders 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 8.10 shows how the DMA controller shares the CPU’s memory bus.

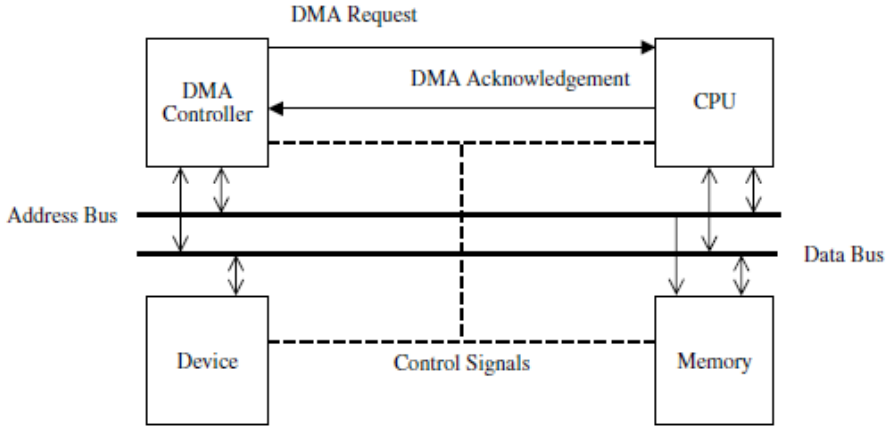


Figure 8.10 DMA controller shares the CPU's memory bus

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. Direct memory access data transfer can be performed in burst mode or single cycle mode. In burst mode, the DMA controller keeps control of the bus until all the data has 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.

If care is not taken, this overhead can result in a degradation of the 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 very large 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 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.

3.5 Buses

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 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 bus line is normally identified by a number.

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 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: send request to use the bus

2. Master: request is granted and bus is allocated to master
3. Master: place address/data on bus
4. Slave: slave is selected
5. Master: signal data transfer
6. Slave: take data
7. Master: free the bus

Self-Assessment Question

1. What is the main idea behind the use of Direct Memory Access?

Self-Assessment Answer

1. 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.

3.5.1 Synchronous Buses

In synchronous buses, you’ll learn that the steps of data transfer take place at fixed clock cycles. Everything is synchronized to 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 you’re 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.

3.5.2 Asynchronous Buses

There are no fixed clock cycles in asynchronous buses. Handshaking is used instead.

Figure 8.11 shows the handshaking protocol. The master asserts the data-ready line (point 1 in the figure) until it sees a data-accept signal.

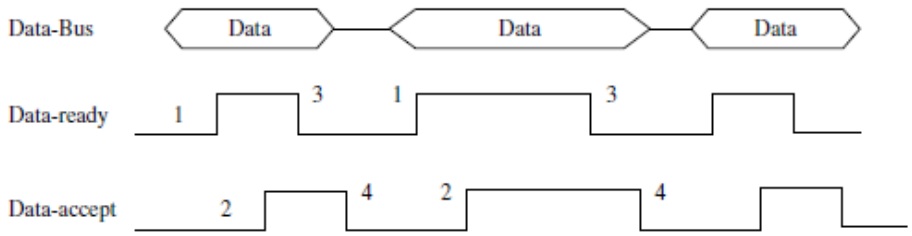


Figure 8.11 Asynchronous bus timing using handshaking protocol

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 then repeated, until the data is completely transferred. Asynchronous bus is appropriate for different speed devices.

Self-Assessment Question

1. Differentiate between Synchronous and Asynchronous Bus

Self-Assessment Answer

1. In Synchronous Bus, everything is synchronized to Bus clock and signal. In Asynchronous Bus, handshake.

3.5.3 Bus Arbitration

Why bus arbitration?

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. Then, conflicts can be resolved based on fairness or priority in a centralized or distributed mechanisms.

Centralized Arbitration in centralized arbitration schemes, is a single arbiter used in selecting 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 8.12 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 propagation of the bus grant signal any further. If it has not made a request, it will simple 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 deasserted.

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 8.13. 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, daisy chain is used. In this scheme, each device is attached to the daisy chain of one priority level.

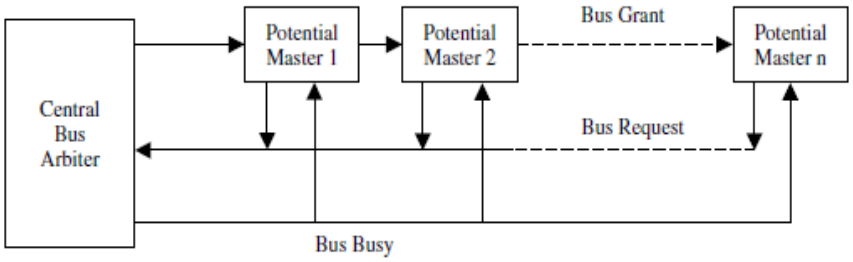


Figure 8.12 Centralized arbiter in a daisy-chain scheme

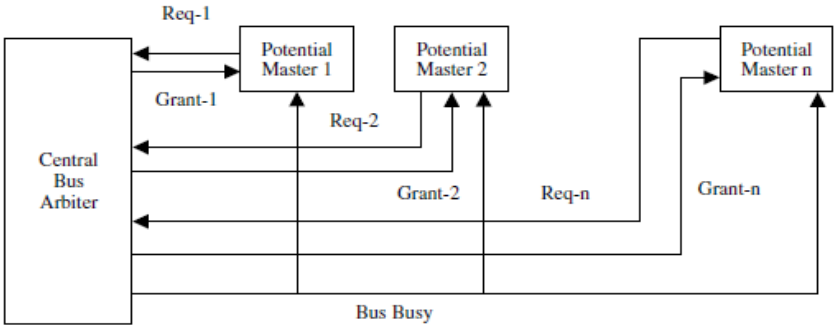


Figure 8.13 Centralized arbiter with independent request and grant lines

In addition, 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 8.14 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.

On the second note, 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. Hope it's clear? For example, a conflict can always be resolved in favor 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.

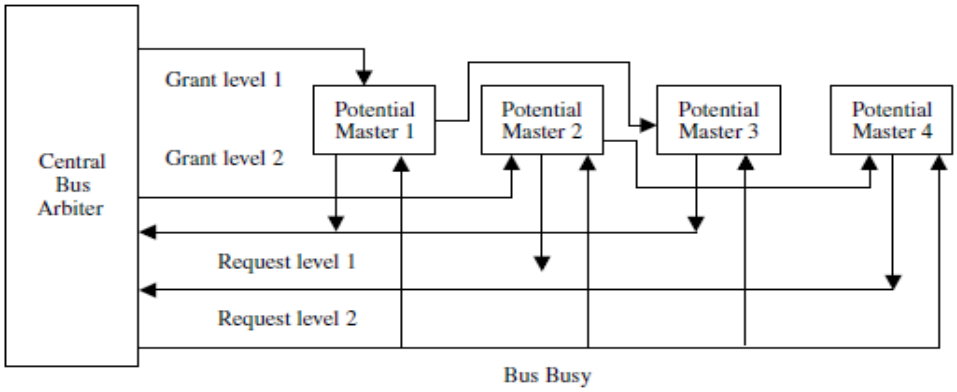


Figure 8.14 Centralized arbiter with two priority levels (four devices)

3.6 Input–Output Interfaces

What is an interface?

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.

Example: What are some of the input and output devices?

Answer: Some of input devices are Keyboards, Mouse, scanners, mice, joysticks and digital cameras. Some of output devices are monitors (screen), printers, speakers,

On a second note, 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 8.4 shows a summary of the variety of buses and interfaces used in personal computers.

Self-Assessment Question

1. When two or more devices want to become the bus master at the same time, what is used to handle this?

Self-Assessment Answer

1. Bus arbitration

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.

4.0 Conclusion

In this unit, you have learned about the Input/output fundamentals. You have also learnt about the programmed I/O, Interrupt-Driven I/O, Direct Memory Access (DMA), Buses and Input–Output Interfaces.

5.0 Summary

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. 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. 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. 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 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.

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 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 bus line is normally identified by a number. Related groups of bus lines are usually identified by a name.

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 mechanisms. Arbitration In centralized arbitration schemes, a single arbiter is used to select the next master.

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.

6.0 Tutor-Marked Assignment

1. Explain the sequence of events involving the master and slave in a 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?
2. Summarize the steps for Direct Memory Access (DMA) operations and sketch the diagram of DMA controller shares the CPU's memory bus?
3. Explain asynchronous buses?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

NOUN, (2008). Introduction to Computer Organisation. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island, Lagos. First Printed 2008

Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.

Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd ed) Prentice Hall of India.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

Unit 2

Handshaking

Contents

1.0 Introduction

In this unit, what you will learn concerns handshaking. 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. You will also learn about simple TLS handshake, Client-authenticated TLS handshake and Resumed TLS handshake.

2.0 Learning Outcome

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

i. Explain handshaking
ii. Describe simple Transport Layer Security (TLS) handshake
iii. Explain Client-authenticated TLS handshake
iv. Describe Resumed TLS handshake

3.0 Learning Content

3.1 Handshaking

In information technology, telecommunications, and related fields, you will find out 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 follows the physical establishment of the channel and precedes normal information transfer.

However, 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.

You can also use handshaking 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 technique of communication between two entities. A simple handshaking protocol might only involve the receiver sending a message meaning "I received your last message and I am ready for you to send me another one."

You will notice later that a more complex handshaking protocol might allow the sender to ask the receiver if he is ready to receive or for the receiver to reply with a negative acknowledgement meaning "I did not receive your last message correctly, please resend it" (e.g. if the data was corrupted en route).

So 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.

Self-Assessment Question

1. What is Handshaking?

Self-Assessment Answer

1. 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.

3.2 Examples of Handshaking

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

3.2.1 Common Types of Handshakes

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.

3.2.2 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:
   i. 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*.

ii. 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. 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.

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

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

v. 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.

vi. 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 content type of 20.

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

   ii. 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)."

   i. The server sends its authenticated and encrypted **Finished** message.

   ii. 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 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.

3.2.3 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:
    i. 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.
    ii. 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.
    iii. The server sends its **Certificate** message (depending on the selected cipher suite, this may be omitted by the server).[31]
    iv. The server requests a certificate from the client, so that the connection can be mutually authenticated, using a **CertificateRequest** message.
    v. The server sends a **ServerHelloDone** message, indicating it is done with handshake negotiation.
    vi. The client responds with a **Certificate** message, which contains the client's certificate.
    vii. 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.
    viii. 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.
    ix. 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.
    i. Finally, the client sends an encrypted **Finished** message, containing a hash and MAC over the previous handshake messages.
    ii. 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). "

i. The server sends its own encrypted **Finished** message.

ii. 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 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.

3.2.4 Resumed TLS handshake

You'll agree with me that 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. 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". 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. Is that clear!

1. Negotiation phase:

   i. 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.

   ii. 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.

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

ii. 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."
   i. The client sends its own encrypted **Finished** message.
   ii. 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 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

Self-Assessment Question

1. Give 3 types of handshake.

Self-Assessment Answer

1. Simple TLS handshake
2. Client-automated TLS handshake
3. Resumed TLS handshake

4.0 Conclusion

In this unit, you have learned about handshaking. You have also learnt about simple TLS handshake**,** Client-authenticated TLS handshake and Resumed TLS handshake.

5.0 Summary

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.

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. 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.

6.0 Tutor-Marked Assignment

1. Describe the three-way handshaking?
2. Explain Simple TLS handshake?
3. What do you understand by handshaking and its possibilities?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

NOUN, (2008). INTRODUCTION TO COMPUTER ORGANISATION. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island, Lagos. First Printed 2008

Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.

Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd ed) Prentice Hall of India.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://scarybeastsecurity.blogspot.com/2009/02/vsftpd-210-released.html. Retrieved 2012-05-17.

https://www.switch.ch/pki/meetings/2007-01/namebased_ssl_virtualhosts.pdf. Retrieved 2012-05-17.

Unit 3

Data Buffer

Contents

1.0 Introduction

In this unit, what you will learn borders on data buffer. A buffer is a region of a physical memory storage used to temporarily hold data while it is being moved from one place to another. You will also learn about brief history of data buffer, applications of buffers, telecommunication buffer, and buffer versus cache and multiple buffering.

2.0 Learning Outcome

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

i. Explain data buffer
ii. Describe bbriefly history of data buffer
iii. State applications of buffers
iv. Describe telecommunication buffer
v. Explain buffer versus cache
vi. Describe multiple buffering

3.0 Learning Content

3.1 Data Buffer

In computer science, a buffer is a region of a physical memory storage used to temporarily hold data, while it is being moved from one place to another. Normally, a 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.

The above is comparable to buffers in telecommunication. You can implement a buffers into 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. Majority of buffers are implemented in software, which typically use the faster RAM to store temporary data, due to the much faster access time compared with hard disk drives.

We use Buffers typically 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 in online video streaming. A buffer often adjusts timing by implementing a queue (or FIFO) algorithm in memory, at the same time writing data into the queue at one rate and reading it at another rate.

Self-Assessment Question

1. Define Buffer

Self-Assessment Answer

1. A buffer is a region of a physical memory storage used to temporarily hold data while it is being moved from one place to another.

3.2 Brief History of Data

In 1952, the image processing pioneer Russel A. Kirsch first mention about a print buffer as an Out scriber devised for the SEAC computer. One of the most serious problems in the design of automatic digital computers is that of getting the calculated results out of the machine quickly 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 are able to 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.

3.3 Applications of Buffers

We use buffers as 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.

3.4 Telecommunication Buffer

A buffer routine or storage medium used in telecommunications compensates for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another.

You should be aware that buffers are used for many purposes, such as

1. Interconnecting two digital circuits operating at different rates,
2. Holding data for use at a later time,
3. Allowing timing corrections to be made on a data stream
4. Collecting binary data bits into groups that can then be operated on as a unit,
5. Delaying the transit time of a signal in order to allow other operations to occur.

3.5 Buffer versus Cache

We may notice that a cache often also acts as a buffer, and vice versa. However, cache operates on the ground that the same data will be read from it multiple times. Written data will soon be read, or that there is a good chance of multiple reads or writes to combine and form a single larger block. Its sole purpose is to reduce accesses to the underlying slower storage. 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 is 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:

1. The BUFFERS command/statement in CONFIG.SYS of DOS.
2. 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.
3. The integrated buffer on a Hard Disk Drive, Printer or other piece of hardware.
4. The Frame buffer on a video card.

3.6 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.

3.6.1 Description of Multiple Buffering

The easiest way to explain how multiple buffering works is to tell you 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 in 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 employed another person to carry a bucket to the pool while one is being filled and another emptied, then this would be analogous to triple buffering. If this step took long enough you could employ 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.

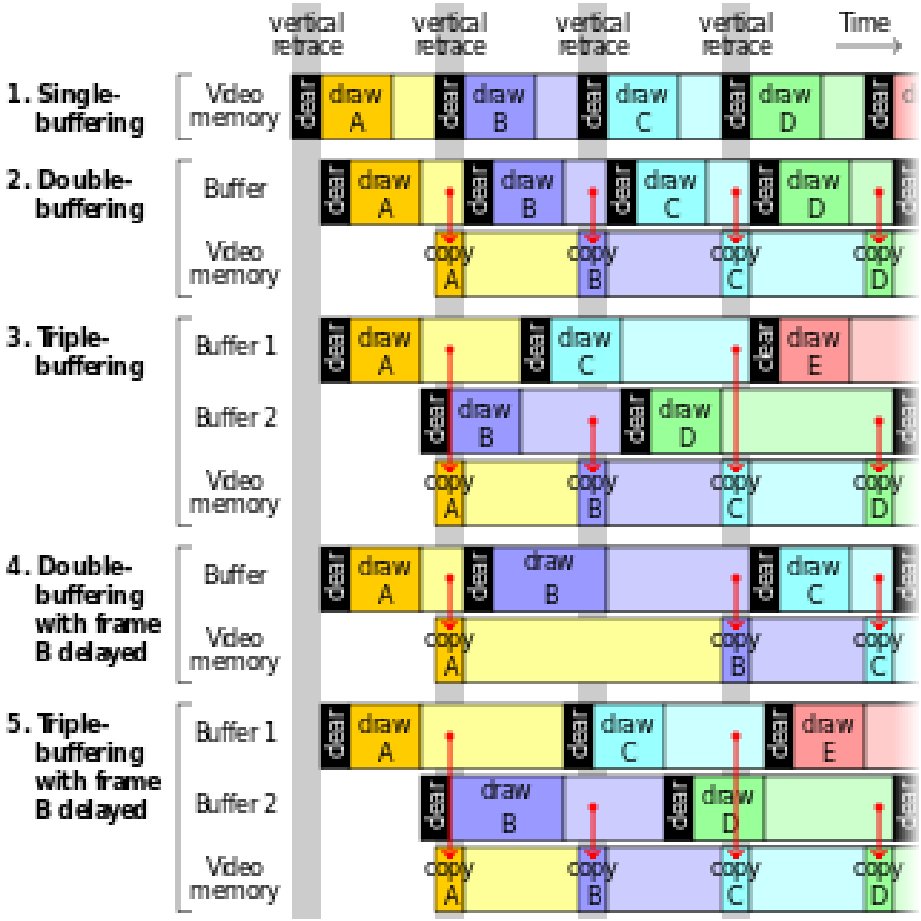


Figure 1: *Sets **1**, **2** and **3** represent the operation of single, double and triple buffering, respectively, with vertical synchronization (vsync) enabled. In each graph, time flows from left to right. Set **4** shows what happens when a frame (B, in this case) takes longer than normal to draw. In this case, a frame update is missed. In time-sensitive implementations such as video playback, the whole frame may be dropped. With triple buffering in set **5**, drawing of frame B can start without having to wait for frame A to be copied to video memory, reducing the chance of a delayed frame missing its vertical retrace*.

The easiest way to 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 in 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 employed another person to carry a bucket to the pool while one is being filled and another emptied, then this would be analogous to triple buffering. If this step took long enough you could employ 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.

3.6.2 Double Buffering Petri Net

The Petri net in the illustration below 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. At 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 enabled always in pair (R1 with W2 and R2 with W1 respectively).

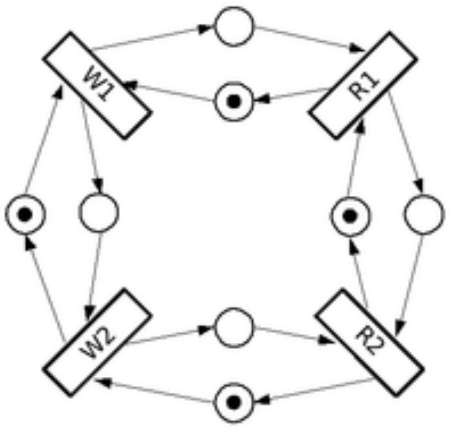


Figure 2: Double Buffering Petri Net

3.6.3 Double Buffering in Computer Graphics

In computer graphics, **double buffering** is a technique for drawing graphics that shows no (or less) flicker, tearing, and other artifacts. It is difficult for a program to draw a display so that pixels do not change more than once. For instance, to update a page of text it is much easier to clear the entire page and then draw the letters than to somehow erase all the pixels that are not in both the old and new letters.

However, this intermediate image is seen by the user as flickering. In addition, computer monitors constantly redraw the visible video page (at around 60 times a second), so even a perfect update may be visible momentarily as a horizontal divider between the "new" image and the un-redrawn "old" image, known as tearing. A software implementation of double buffering has all drawing operations store their results in some region of system RAM; any such region is often called a "back buffer".

When all drawing operations are considered complete, the whole region (or only the changed portion) is copied into the video RAM (the "front buffer"); this copying is usually synchronized with the monitor's raster beam in order to avoid tearing. Double buffering necessarily requires more video memory and CPU time than single buffering because of the video memory allocated for the back buffer, the time for the copy operation, and the time waiting for synchronization.

Compositing window manager often combine the "copying" operation with "compositing" used to position windows, transform them with scale or warping effects, and make portions transparent. Thus the "front buffer" may contain only the composite image seen on the screen, while there is a different "back buffer" for every window containing the non-composited image of the entire window contents.

3.6.4 Page Flipping

In the page-flip method (sometimes called **ping-pong buffering**), instead of copying the data, both buffers are capable of being displayed (both are in VRAM). At any one time, one buffer is actively being displayed by the monitor, while the other, background buffer is being drawn. When drawing is complete, the roles of the two are switched. The page-flip is typically accomplished by modifying the value of a pointer to the beginning of the display data in the video memory.

The page-flip is much faster than copying the data and can guarantee that tearing will not be seen as long as the pages are switched over during the monitor's vertical blanking interval -- the blank period when no video data is being drawn. The currently active and visible buffer is called the **front buffer**, while the background page is called the "back buffer".

3.6.5 Triple Buffering

In computer graphics, **triple buffering** is similar to double buffering but provides a speed improvement. In double buffering the program must wait until the finished drawing is copied or swapped before starting the next drawing. This waiting period could be several milliseconds during which neither buffer can be touched. In triple buffering the program has two back buffers and can immediately start drawing in the one that is not involved in such copying.

The third buffer, the front buffer, is read by the graphics card to display the image on the monitor. Once the monitor has been drawn, the front buffer is flipped with (or copied from) the

back buffer holding the last complete screen. Since one of the back buffers is always complete, the graphics card never has to wait for the software to complete.

Consequently, the software and the graphics card are completely independent, and can run at their own pace. Finally, the displayed image was started without waiting for synchronization and thus with minimum lag. Due to the software algorithm not having to poll the graphics hardware for monitor refresh events, the algorithm is free to run as fast as possible. This can mean that several drawings that are never displayed are written to the back buffers.

This is not the only method of triple buffering available, but is the most prevalent on the PC architecture where the speed of the target machine is highly variable.

Another method of triple buffering involves synchronizing with the monitor frame rate. Drawing is not done if both back buffers contain finished images that have not been displayed yet.

This avoids wasting CPU drawing undisplayed images and also results in a more constant frame rate (smoother movement of moving objects), but with increased latency. This is the case when using triple buffering in DirectX, where a chain of 3 buffers are rendered and always displayed.

Triple buffering implies three buffers, but the method can be extended to as many buffers as is practical for the application. Usually, there is no advantage to using more than three buffers.

3.6.6 Quad Buffering

The term "Quad buffering" is used in stereoscopic implementations, and means the use of double buffering for each of the left and right eye images, thus four buffers total. The command to swap or copy the buffer typically applies to both pairs at once. If triple buffering was used, then there would be six buffers.

The term **double buffering** is used for copying data between two buffers for direct memory access (DMA) transfers, *not* for enhancing performance, but to meet specific addressing requirements of a device (esp. 32-bit devices on systems with wider addressing provided via Physical Address Extension).

Microsoft Windows device drivers are particularly noteworthy as a place where such double buffering is likely to be used. On a Linux or BSD system these are called **bounce buffers** because data must "bounce" via these buffers for input or output.

Double buffering is also used as a technique to facilitate interlacing or de-interlacing of video signals.

Self-Assessment Question

1. Quad Buffering makes use of four Double Buffering. True of False?

Self-Assessment Answer

1. False

4.0 Conclusion

In this unit, you have learned about data buffer. You have also learnt about brief history of data buffer, applications of buffers, telecommunication buffer, and buffer versus cache and multiple buffering.

5.0 Summary

In computer science, a buffer is a region of a 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). 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 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 buffer routine or storage medium used in telecommunications compensates for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another.

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.

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

6.0 Tutor-Marked Assignment

1. Explain telecommunication buffer?
2. Explain the differences between buffer and cache with typical examples?
3. What do you understand by multiple buffering?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

NOUN, (2008). INTRODUCTION TO COMPUTER ORGANISATION. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island,Lagos. First Printed 2008

Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.

Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd ed) Prentice Hall of India.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://scarybeastsecurity.blogspot.com/2009/02/vsftpd-210-released.html. Retrieved 2012-05-17.

https://www.switch.ch/pki/meetings/2007-01/namebased_ssl_virtualhosts.pdf. Retrieved 2012-05-17.

http://www.anandtech.com/video/showdoc.aspx?i=3591&p=1. Retrieved 2009-07-16.

http://www.anandtech.com/video/showdoc.aspx?i=3591&p=1. Retrieved 2009-07-16.

http://www.microsoft.com/whdc/system/platform/server/PAE/PAEdrv.mspx#E2D. Retrieved 2008-04-07.

Unit 4

External Storage

Contents

1.0 Introduction

In this unit, what you will learn borders on external storage. Computers have the capability to store your information in a variety of different ways. All of these different ways require a specific storage device. More than likely, you have used a variety of different storage devices on your computer. You just may not have realized it at the time. Storage by way of zip drives and floppy discs is now a thing of the past. There are many new solutions for computer users to store large amounts of data.

2.0 Learning Outcome

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

i. Explain external storage
ii. Outline the advantages of external storage
iii. Describe the types of external storage
iv. Explain solid state storage
v. Describe other devices
vi. Explain data organization

3.0 Learning Content

3.1 External Storage

In computer, 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 in using semiconductor memories is the cost per bit of the stored information. So to fulfill the large storage requirements of computers, magnetic disks, optical disks are generally used.

3.1.1 Advantages of External Storage

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

3.2 Types of External Storage

3.2.1 Magnetic storage

i. Cassette tape

ii. Floppy disk

3.2.2 Optical storage

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

Optical Storage Devices

Compact Disk (CD)

CD stands for compact disk. The speed is much less than a hard disk. The storage capacity is 700 MB. Types of CDs include:

i. CD-ROM: It is compact disk read only memory. It can be read only.

ii. CD-Recordable: It was invented in 1990s. Using CD-R it is possible to write data once on a disk. These are write once read many disks.

iii. CD-Rewritable: 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.

Digital Versatile Disk (DVD)

DVD stands for digital versatile disk. Its speed is much faster than CD but not as fast as 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.

Self-Assessment Question

1. Give 2 types of External Storage

Self-Assessment Answer

1. Magnetic Storage, Optical Storage

3.2.3 Solid State Storage

Flash memory is a solid state memory. It was invented in 1980s by Toshiba. A 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 case of EEPROM data are erased one byte at a time which makes it extremely slower. 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

i. It has got no moving parts. So it is durable and less susceptible to mechanical damages.

ii. It is small in size and light in weight. Hence it is extensively used in portable devices.

iii. Flash memory transfers data at a faster rate.

iv. 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

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

ii. The storage capacity of a flash memory is far less than a hard disk.

3.2.4 Flash Memory Devices

1. 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.
2. Memory stick: Sony introduced memory stick standard in 1998. Memory stick is an integrated circuit designed to serve as a storage and transfer medium for digital data. It can store data in various form as 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 memory stick is 160 Mbit/s.

Self-Assessment Question

1. When was flash memory invented?

Self-Assessment Answer

1. It was invented in 1980s by Toshiba

3.3 Other devices

Other external storage devices include:

1. Punched cards
2. Zip disks
3. Microforms
4. 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.

3.4 Data Organization

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

4.0 Conclusion

In this unit, you have learned about external storage. You have also learnt about the advantages of external storage, different types of external storage, solid state storage, and other devices and data organization.

5.0 Summary

In computing external storage comprise 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. External storage devices provide additional storage other than that available in computer.

Some of the advantages of external storage are; Data can be transported easily from one place to another; It is useful to store software and data that is not needed frequently. Optical media are the media that use laser light technology for data storage and retrieval. CD stands for compact disk. The speed is much less than a hard disk. The storage capacity is 700 MB. Types of CDs include: DVD stands for digital versatile disk.

Its speed is much faster than CD but not as fast as hard disk. The standard DVD-5 technology has a storage capacity of 4.7 GB. The Flash memory is a solid state memory. It was invented in 1980s by Toshiba. A flash memory is a particular type of EEPROM (Electrically Erasable Programmable Read Only Memory). It is a no-volatile memory. Other external storage devices include: punched cards; Zip disks.

6.0 Tutor-Marked Assignment

1. Storage of information is very important in computing, briefly describe the external storage in computing?
2. What are the merits of the external storage devices in computing?
3. Explain the different modes of data organization in external storage devices?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

NOUN, (2008). INTRODUCTION TO COMPUTER ORGANISATION . CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island,Lagos. First Printed 2008

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://www.ehow.com/about_5402387_different-storage-devices computers.html#ixzz24plKwJ8K

Module 4

Introduction to Networks, RAID Architectures, Data Path & Control Unit

Unit 1

Introduction to Networks

Contents

1.0 Introduction

In this unit, what you will learn rotates on computer networks. Networking not only enables sharing information and resources among the users but also distributed processing. There are five types of network. Point-point, LAN, MAN, WAN and VAN. The network topology defines how the devices (computers, printers...etc.) are connected and how the data flows from one device to another. They are broadly categorized as bus, ring, star, mesh and hybrid. The details are in this unit.

2.0 Learning Outcome

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

i. Define Computer Network

ii. Describe the Types of Networks

iii. Explain Network Topology

iv. Describe Hybrid Topology

3.0 Learning Content

3.1 What is a Network?

A network is a set of equipment (often referred as data terminal equipment / DTE, or simply terminals or nodes) connected by a communication channel, which can be either guided/unguided media. DTE equipment can be a computer, printer or any device capable of sending and/or receiving data generated by other nodes on the network.

3.1.1 Why Networking?

1. **Sharing of hardware**: Computer hardware resources, Disks, Printers.
2. **Sharing of software:** Multiple single user licenses are more expensive than multi-user
3. **License:** Easy maintenance of software
4. **Sharing of information:** Several individuals can interact with each other; Working in groups can be formed
5. **Communication:** e-mail; internet telephony; audio conferencing; video conferencing
6. **Scalability:** Individual subsystems can be created and combine it into a main system to enhance the overall performance.
7. **Distributed systems:** Networked environment of computers can distribute the work load among themselves keeping transparency to the end user

Self-Assessment Question

1. What is a Network?

Self-Assessment Answer

1. A network is a set of equipment (often referred as data terminal equipment / DTE, or simply terminals or nodes) connected by a communication channel, which can be either guided/unguided media.

3.2 Types of networks

The various types of network include the following;

3.2.1 Point to Point

As shown in Figure 2.2.1, a communication system used to interconnect two computers. The computers output electrical signals directly through the serial port. The data can be passed directly through the communication medium to the other computer if the distance is small (less than 100 meters).

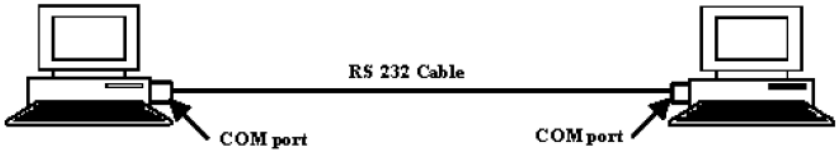


Fig.2.2.1 PC to PC communication using com ports

Figure 2.2.1 shows a communication system in which two PCs communicate with each other over a existing say local telephone exchange (PABX) network. In this system, we introduced device called DTE data terminal equipment. The example here for DTE is modem (modulator demodulator) connected at both ends.

The PCs send digital signals, which the modem converts into analog signals and transmits through the medium (copper wires). At the receiving end, the modem converts the incoming analog signal into digital form and passes it on to the PC.

Fig.2.2.2 PC to PC communication over existing telephone network

3.3 Local Area Network (LAN)

A LAN is a local area network that is a small collection of computers in a small geographic area of less than couple of kilometers and is very fast in data transfer. Depending on technology implementation, a LAN can be as simple as two PCs and a printer got connected in a small office. It can even extend throughout an organization and include multimedia (text, voice, video) data transfers. The LANs may be configured in many ways. The peer-to-peer

configuration is the simplest form. In this configuration computers are connected together to share their recourses among themselves. In such configurations it is very difficult impose security features.

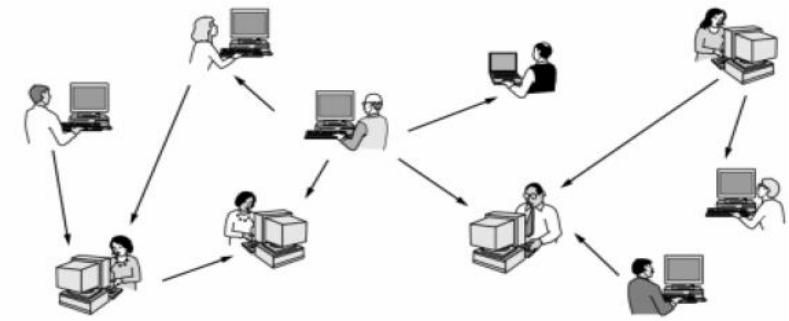


Fig 2.3.1 In a peer-to-peer configuration there is no security

On the other hand, LANs can also be architecture in a client server model with full control over security and protection. Today Ethernet is a dominant LAN technology.

Client/server describes the relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfills the request.

Although the client/server idea can be used by programs within a single computer, it is a more important idea in a network. In a network, the client/server model provides a convenient way to interconnect programs that are distributed efficiently across different locations. Computer transactions using the client/server model are very common. For example, to check your bank account from your computer, a client program in your computer forwards your request to a server program at the bank.

That program may in turn forward the request to its own client program that sends a request to a database server at another bank computer to retrieve your account balance. The balance is returned back to the bank data client, which in turn serves it back to the client in your personal computer, which displays the information for you.

The client/server model has become one of the central ideas of network computing. Most business applications being written today use the client/server model. So does the Internet's main program, TCP/IP. In marketing, the term has been used to distinguish distributed computing by smaller dispersed computers from the "monolithic" centralized computing of mainframe computers.

But this distinction has largely disappeared as mainframes and their applications have also turned to the client/server model and become part of network computing. In the usual client/server model, one server, sometimes called a daemon, is activated and awaits client requests.

Typically, multiple client programs share the services of a common server program. Both client programs and server programs are often part of a larger program or application. Relative to the Internet, your Web browser is a client program that requests services (the sending of Web pages or files) from a Web server (which technically is called a Hypertext Transport Protocol or HTTP server) in another computer somewhere on the Internet.

Similarly, your computer with TCP/IP installed allows you to make client requests for files from File Transfer Protocol (FTP) servers in other computers on the Internet. Other program relationship models included master/slave, with one program being in charge of all other programs, and peer-to-peer, with either of two programs able to initiate a transaction.

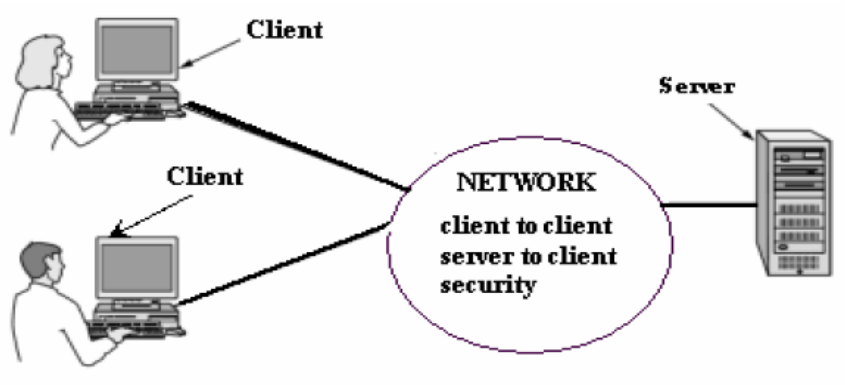


Fig 2.3.2 Client server model

A typical LAN in a corporate office links a group of related computers, workstations. One of the best computers may be given a large capacity disk drive and made as server and remaining computers as clients.

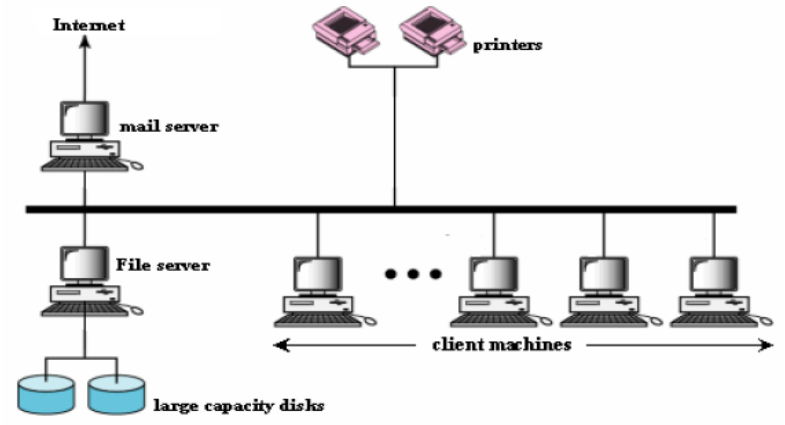


Fig2.3.3 A LAN setup

Self-Assessment Question

1. What is the full meaning of LAN?

Self-Assessment Answer

1. Local Area Network

3.4 Metropolitan Area Network (MAN)

The metropolitan area network is designed to cover an entire city. It can be a single network such as cable TV or a number of LANs connected together within a city to form a MAN. Privately laid cables or public leased lines may be used to form such network. For instance a business organization may choose MAN to inter connect all its branch offices within the city.

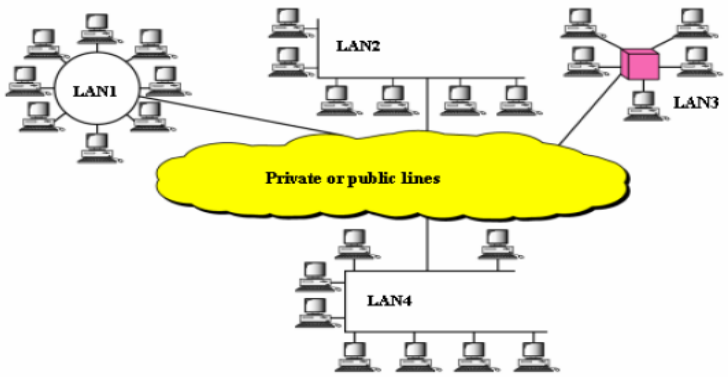


Fig 2.4.1 Typical Metropolitan area network

Self-Assessment Question

1. What is the Full meaning of MAN?

Self-Assessment Answer

1. Metropolitan Area Network

3.5 Wide Area Network (WAN)

A WAN is a data communications network that covers a relatively broad geographic area, often a country or continent. It contains a collection of machines intended for running user programs. These machines are called hosts. The hosts are connected by subnet. The purpose of subnet is to carry messages from hosts to hosts.

The subnet includes transmission facilities, switching elements and routers provided by common agencies, such as telephone companies. Now a days routers with satellite links are also becoming part of the WAN subnet. All these machines provide long distance transmission of data, voice, image and video information.

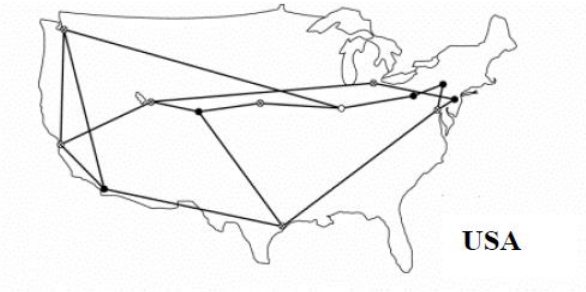


Fig 2.5.1 A typical WAN covering entire United States

Unlike LAN which depend on their own hardware for transmission, WANs may utilize public, leased, or private communication devices when it come across and therefore span an unlimited number of kilometers. A network device called a router connects LANs to a WAN.

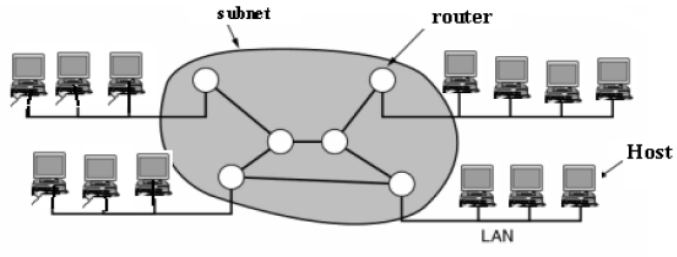


Fig 2.5.2 Typical WAN setup with hosts, routers and subnet.
The Internet is the largest WAN in existence.

Self-Assessment Question

1. What is the full meaning of WAN?

Self-Assessment Answer

1. Wide Area Network

3.6 Value added Network (VAN)

3.6.1 Value-added networks (VAN)

VAN are communications networks are supplied and managed by third-party companies that facilitate electronic data interchange, Web services and transaction delivery by providing extra networking services. A value-added network (VAN) is a private network provider (sometimes called a turnkey communication line) that is hired by a company to facilitate electronic data interchange (EDI) or provide other network services.

Let me tell you something, before the arrival of the World Wide Web, some companies hired value-added networks to move data from their company to other companies. So with the arrival of the World Wide Web, many companies found it more cost-efficient to move their data over the Internet instead of paying the minimum monthly fees and per-character charges found in typical VAN contracts.

In response, contemporary value-added network providers now focus on offering EDI translation, encryption, secure email, management reporting, and other extra services for their customers. Value-added networks got their first real foothold in the business world in the area of electronic data interchange (EDI).

VANs were deployed to help trading and supply chain partners automate many business-to-business communications and thereby reduce the number of paper transfers needed, cut costs and speed up a wide range of tasks and processes, from inventory and order management to payment. In today's world, e-commerce is increasingly based on XML, though EDI remains an important part of business and still relies on value-added networks. But other types of VANs have begun to appear, including Web services networks and transaction delivery networks.

3.6.2 Transaction Delivery Networks (TDN)

The newest evolution of VANs, which first appeared in 2000, are the transaction delivery networks (TDN) that provide services for secure end-to-end management of electronic transactions. Also called transaction processing networks or Internet utility platforms, TDNs can guarantee delivery of messages in addition to providing high security and availability, network performance monitoring and centralized directory management.

TDNs typically use a store-and-forward messaging architecture that's designed to adapt readily to a wide range of disparate systems and support any kind of transaction. Most TDNs offer secure encryption using a public-key infrastructure and certificate authorization for trading partners.

3.6.3 Internetworks

Internetwork or simply the internets are those when two or more networks are get connected. Individual networks are combined through the use of routers. Lowercase internet should not be confused with the worldwide Internet.

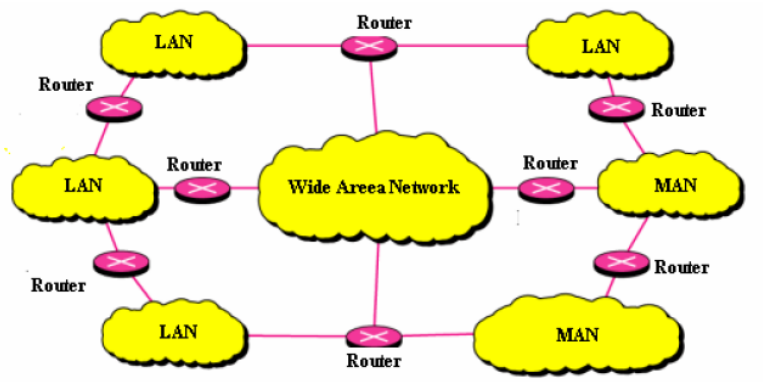


Fig 2.6.1 Typical internetwork connecting LANs and MANs

Self-Assessment Question

1. What is Value Added Network?

Self-Assessment Answer

1. VAN are communications networks supplied and managed by third-party companies that facilitate electronic data interchange, Web services and transaction delivery by providing extra networking services.

3.7 Network Topology

The topology defines how the devices (computers, printers...etc.) are connected and how the data flows from one device to another. There are two conventions while representing the topologies. The physical topology defines how the devices are physically wired. The logical topology defines how the data flows from one device to another.

Broadly categorized into I) Bus II) Ring III) Star IV) Mesh

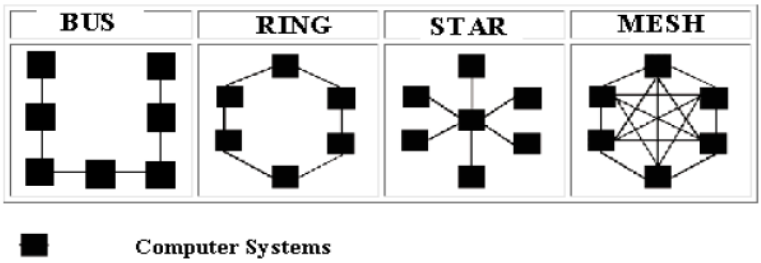


Fig 2.7.1 Outlines of various types of topologies

3.7.1 Bus Topology

In a bus topology all devices are connected to the transmission medium as backbone. There must be a terminator at each end of the bus to avoid signal reflections, which may distort the original signal. Signal is sent in both directions, but some buses are unidirectional. Good for small networks. Can be used for 10BASE5 (thick net), 10BASE2 (thin net) or 10BROAD36 (broad band) co-axial bus standards.

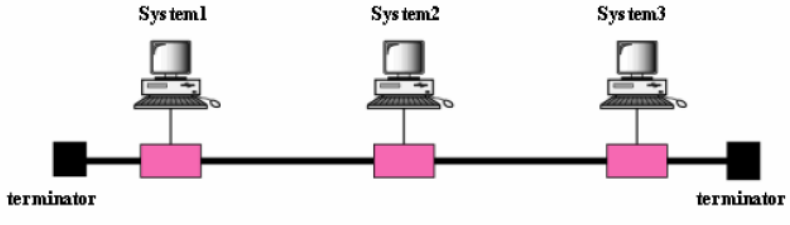


Fig 2.7.2 Physical topology of bus topology.

The main problem with the bus topology is failure of the medium will seriously affect the whole network. Any small break in the media the signal will reflect back and cause errors. The whole network must be shut down and repaired. In such situations it is difficult to troubleshoot and locate where the break in the cable is or which machine is causing the fault; when one device fails the rest of the LAN fails.

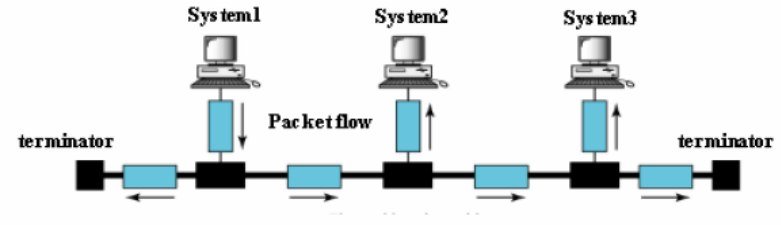


Fig 2.7.3 Logical topology illustration of bus topology.

3.7.2 Ring Topology

Ring topology was in the beginning of LAN area. In a ring topology, each system is connected to the next as shown in the following picture.

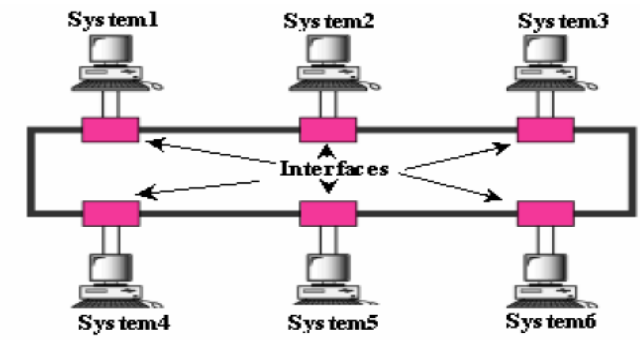


Fig. 2.7.4 Ring topology illustration.

Each device has a transceiver which behaves like a repeater which moves the signal around the ring; ideal for token passing access methods. In this topology signal degeneration is low; only the device that holds the token can transmit which reduces collisions. If you see its negative aspect, it is difficult to locate a problem cable segment; expensive hardware.

3.7.3 Star topology

In a star topology each station is connected to a central node. The central node can be either a hub or a switch. The star topology does not have the problem as seen in bus topology. The failure of a media does not affect the entire network. Other stations can continue to operate until the damaged segment is repaired.

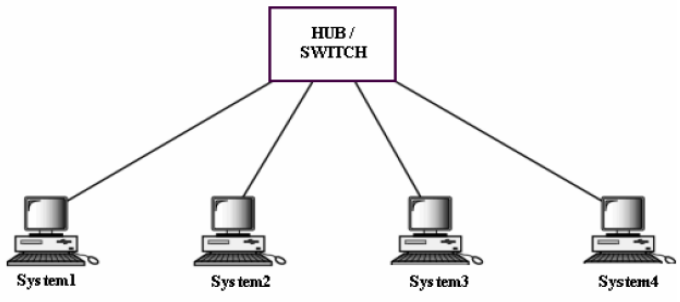


Fig 2.7.5 Physical topology of Star topology.

Commonly used for 10BASE5, 10BASE-T or 100BASE-TX types.

The advantages are cabling is inexpensive, easy to wire, more reliable and easier to manage because of the use of hubs which allow defective cable segments to be routed around; locating and repairing bad cables is easier because of the concentrators; network growth is easier.

The disadvantages are all nodes receive the same signal therefore dividing bandwidth; Maximum computers are 1,024 on a LAN. Maximum UTP (Un shielded twisted pair) length is 100 meters; distance between computers is 2.5 meters.

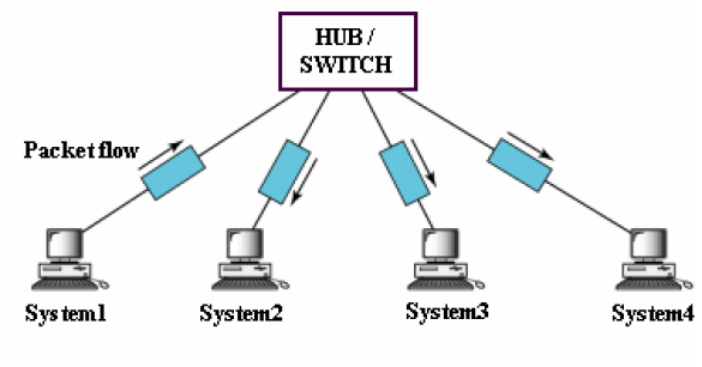


Fig 2.7.6 Logical topology of Star topology.

This topology is the dominant physical topology today.

3.7.4 Mesh Topology

A mesh physical topology is when every device on the network is connected to every device on the network; most commonly used in WAN configurations Helps find the quickest route on the network; provides redundancy. Very expensive and not easy to set up.

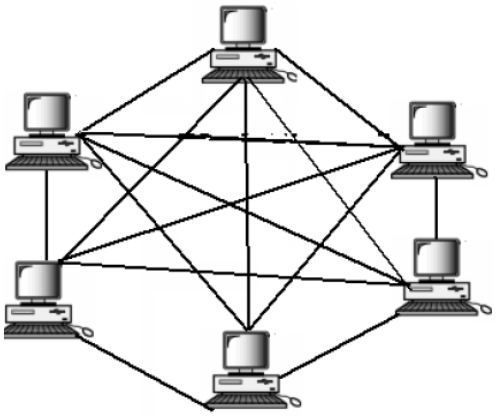


Fig 2.10.1 Physical topology of Mesh topology.

3.7.5 Hybrid Topology

A hybrid topology is a combination of any two or more network topologies in such a way that the resulting network does not have one of the standard forms. For example, a tree network connected to a tree network is still a tree network, but two star networks connected together exhibit hybrid network topologies. A hybrid topology is always produced when two different basic network topologies are connected.

Self-Assessment Question

1. List the Network topologies you know.

Self-Assessment Answer

1. Star, Ring, Bus, Tree, Mesh and Hybrid topologies

4.0 Conclusion

What you have learned in this unit is on computer network. You also learnt about the five types of network; Point-point, LAN, MAN, WAN and VAN. You as well learnt about the network topology which defines how the devices (computers, printers...etc.) are connected and how the data flows from one device to another. Furthermore, you learnt that they are broadly categorized as bus, ring, star, mesh and hybrid.

5.0 Summary

Networking not only enables sharing information and resources among the users but also distributed processing. There are five types of network. Point-point, LAN, MAN, WAN and VAN. Point-point allows sharing of files at a very low speed. LANs are networks distributed over a small geographical area. They can be configured peer-peer or much powerful client/server model.

MANs cover entire metropolitan area and may have private lines. WANs cover relatively large geographical area. Here machines are called hosts connected by subnets. The Internet is the largest WAN. VANs are communications networks supplied and managed by third-party companies that facilitate electronic data interchange, Web services and transaction delivery by providing extra networking services.

The network topology defines how the devices (computers, printers,etc.) are connected and how the data flows from one device to another. They are broadly categorized as bus, ring, star, mesh and hybrid. In a bus topology all devices are connected to the transmission medium as backbone. Ring topology was in the beginning of LAN area. In a star topology each station is connected to a central node.

The central node can be either a hub or a switch. A mesh physical topology is when every device on the network is connected to every device on the network; most commonly used in WAN configurations. A hybrid topology is a combination of any two or more network topologies in such a way that the resulting network does not have one of the standard forms.

6.0 Tutor-Marked Assignment

1. Different people have different reasons for setting up a computer networks. What some of these reasons?
2. Explain the topology of a computer networks with emphasis on the bus topology?
3. Using metropolitan area network and wide area network explain the differences between the types of computer networks?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

NOUN, (2008). INTRODUCTION TO COMPUTER ORGANISATION. CIT 246 Course Material, National Open University of Nigeria, Headquarters 14/16 Ahmadu Bello Way, Victoria Island,Lagos. First Printed 2008

Stallings William. *Computer Organisation and Architecture* (3rd ed). Maxwell Macmillan International Editions.

Tanenbaum, Andrew S. (1993). *Structural Computer Organisation* (3rd ed) Prentice Hall of India.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

Unit 2

Multimedia Support RAID Architectures

Contents

1.0 Introduction

In this unit, what you will learn borders on RAID technology. **RAID** stands for **R**edundant **A**rray of **I**nexpensive (or sometimes "Independent") **D**isks. RAID technology was first defined by a group of computer scientists at the University of California at Berkeley in 1987. The scientists studied the possibility of using two or more disks to appear as a single device to the host system.

Although the array's performance was better than that of large, single-disk storage systems, reliability was unacceptably low. To address this, the scientists proposed redundant architectures to provide ways of achieving storage fault tolerance. In addition to defining RAID levels 1 through 5, the scientists also studied data striping -- a non-redundant array configuration that distributes files across multiple disks in an array.

Often known as RAID 0, this configuration actually provides no data protection. However, it does offer maximum throughput for some data-intensive applications such as desktop digital video production.

2.0 Learning Outcome

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

1. Define RAID
2. Describe the driving factors behind RAID
3. Explain RAID Levels
4. Describe types of RAID
5. Explain server technology comparison
6. Describe RAID parity
7. Explain RAID fault tolerance

3.0 Learning Content

3.1 What is RAID?

Redundant **A**rray of **I**nexpensive (or sometimes "Independent") **D**isks (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.

Self-Assessment Question

1. What is the full meaning of RAID?

Self-Assessment Answer

1. **R**edundant **A**rray of **I**nexpensive (or sometimes "Independent") **D**isks (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).

3.2 The Driving Factors behind RAID

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 provides substantial cost benefits, these savings can be quickly offset if 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.

3.3 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).

i. **Minimum number of drives:** 2

ii. **Strengths:** Highest performance.

iii. **Weaknesses:** No data protection; One drive fails, all data is lost.

DRIVE 1	DRIVE 2
Data A	Data B
Data C	Data D
Data E	Data F

RAID 1

Disk mirroring.

i. **Minimum number of drives:** 2

ii. **Strengths:** Very high performance; Very high data protection; Very minimal penalty on write performance.

iii. **Weaknesses:** High redundancy cost overhead; Because all data is duplicated, twice the storage capacity is required.

Mirroring	
Standard Host Adapter	
DRIVE 1	DRIVE 2
Data A	Data A
Data B	Data B
Data C	Data C
Original Data	Mirrored Data

Duplexing	
Standard Host Adapter 1	Standard Host Adapter 2
DRIVE 1	DRIVE 2
Data A	Data A
Data B	Data B
Data C	Data C
Original Data	Mirrored Data

RAID 2

No practical use.

i. **Minimum number of drives:** Not used in LAN

ii. **Strengths:** Previously used for RAM error environments correction (known as Hamming Code) and in disk drives before he uses of embedded error correction.

iii. **Weaknesses:** No practical use; Same performance can be achieved by RAID 3 at lower cost.

RAID3

Byte-level data striping with dedicated parity drive.

i. **Minimum number of drives:** 3

ii. **Strengths:** Excellent performance for large, sequential data requests.

iii. **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.

i. **Minimum number of drives:** 3 (Not widely used)

ii. **Strengths:** Data striping supports multiple simultaneous read requests.

iii. **Weaknesses:** Write requests suffer from same single parity-drive bottleneck as RAID 3; RAID 5 offers equal data protection and better performance at same cost.

RAID 5

Block-level data striping with distributed parity.

i. **Minimum number of drives:** 3

ii. **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.

iii. **Weaknesses:** Write performance is slower than RAID 0 or RAID 1.

DRIVE 1	DRIVE 2	DRIVE 3
Parity A	Data A	Data A
Data B	Parity B	Data B
Data C	Data C	Parity C

RAID 01 (0+1) and RAID 10 (1+0)

Combination of RAID 0 (data striping) and RAID 1 (mirroring). RAID 01 (0+1) is a mirrored configuration of two striped sets (*mirror of stripes*); RAID 10 (1+0) is a stripe across a number of mirrored sets (*stripe of mirrors*). RAID 10 provides better fault tolerance and rebuild performance than RAID 01. Both array types provide very good to excellent overall performance by combining the speed of RAID 0 with the redundancy of RAID 1 without requiring parity calculations.

i. **Minimum number of drives:** 4

ii. **Strengths:** Highest performance, highest data protection (can tolerate multiple drive failures).

iii. **Weaknesses:** High redundancy cost overhead; Because all data is duplicated, twice the storage capacity is required; Requires minimum of four drives.

RAID 01 (0+1 mirror of stripes)			
DRIVE 1	DRIVE 2	DRIVE 3	DRIVE 4
Data A	Data A	mA	mA
Data B	Data B	mB	mB
Data C	Data C	mC	mC
Original Data	Original Data	Mirrored Data	Mirrored Data

RAID 10 (1+0 stripe of mirrors)			
DRIVE 1	DRIVE 2	DRIVE 3	DRIVE 4
Data A	mA	Data B	mB
Data C	mC	Data D	mD
Data E	mE	Data F	mF
Original Data	Mirrored Data	Original Data	Mirrored Data

Self-Assessment Question

1. List the different levels of RAID.

Self-Assessment Answer

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

3.4 Types of RAID

There are 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 favor 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).

	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 Only requires a standard controller.
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 stand-alone Intel-based and UNIX-based servers as well as clustered server environments.	OS independent Build super high-capacity storage systems for high-end servers.

Self-Assessment Question

1. What are the types of RAID?

Self-Assessment Answer

1. Software based and hardware based.

3.5 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

3.6 Parity

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
```

3.7 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:

a. **MTDL:**
Mean **T**ime to **D**ata **L**oss. The average time before the failure of an array component causes data to be lost or corrupted.

b. **MTDA:**
Mean **T**ime between **D**ata **A**ccess (or availability). The average time before non-redundant components fail, causing data inaccessibility without loss or corruption.

c. **MTTR:**
Mean **T**ime **T**o **R**epair. The average time required to bring an array storage subsystem back to full fault tolerance.

d. **MTBF:**
Mean **T**ime **B**etween **F**ailure. 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.

Self-Assessment Question

1. What method does RAID use to recover data?

Self-Assessment Answer

1. Parity

4.0 Conclusion

What you have learned in this unit is on RAID technology. You also learnt about the driving factors behind RAID, RAID Levels, and types of RAID, server technology comparison, RAID parity and RAID fault tolerance.

5.0 Summary

RAID stands for **R**edundant **A**rray of **I**nexpensive (or sometimes "Independent") **D**isks. RAID technology was first defined by a group of computer scientists at the University of California at Berkeley in 1987. The scientists studied the possibility of using two or more disks to appear as a single device to the host system. 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. 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 provides substantial cost benefits, these savings can be quickly offset if information is frequently lost or becomes inaccessible. 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. RAID 0 - RAID 1 - RAID 2 - RAID 3 - RAID 4 - RAID 5 - RAID 01 (0+1) and RAID 10 (1+0) There are 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 favor 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.

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 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.

6.0 Tutor-Marked Assignment

1. RAID technology does not prevent drive failures. However, RAID does provide insurance against disk drive failures. Explain the aspect of fault tolerance?
2. Explain the server technology comparison for RAID?
3. Describe the RAID parity?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://www.adaptec.com

http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_raid

Unit 3

Data Path and Control Unit

Contents

1.0 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. The ALU provides the circuitry needed to perform the arithmetic, logical and shift operations demanded of the instruction set.

The control unit is the entity responsible for fetching the instruction to be executed from the main memory and decoding and then executing it. Figure 5.1 shows the main components of the CPU and its interactions with the memory system and the input/output devices. The CPU fetches instructions from memory, reads and writes data from and to memory, and transfers data from and to input/output devices.

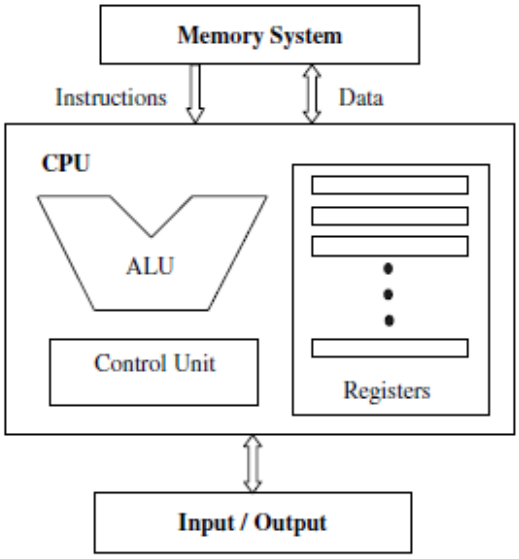


Figure 5.1 Central processing unit main components and interactions with the memory and I/O

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 the memory and stored in the IR.
2. The instruction is decoded.
3. Operands are fetched from the 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.

The execution cycle is repeated as long as there are more instructions to execute.

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 (see Chapter 7).

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.

2.0 Learning Outcome

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

1. Explain data path
2. Describe one-bus, two-bus and three-bus organizations
3. Explain CPU Instruction cycle
4. Describe fetch instructions
5. Explain how to execute simple arithmetic operation
6. Describe interrupt handling
7. Describe control unit
8. Explain hardwired implementation
9. Explain microprogrammed control unit

3.0 Learning Content

3.1 Data Path

The CPU can be divided into a data section and a control section. The data section, which is also called the Data Path, contains the registers and the ALU. The Data Path is capable of performing certain operations on data items. The control section is basically the control unit, which issues control signals to the Data Path. 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 Data Paths 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 Data Path from the PC to the MAR could be useful in speeding up this part of instruction execution.

3.1.1 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 needed 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 5.3 shows a one-bus Data Path 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.

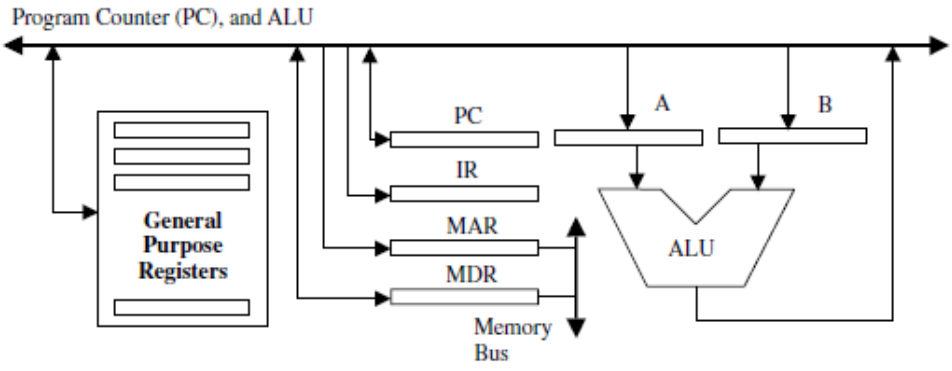


Figure 5.3 One-bus datapath

3.1.2 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 for moving data into registers (in-bus), while the other is dedicated for 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 5.4b shows a two-bus organization with in-bus and out-bus.

3.1.3. Three-Bus Organization

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

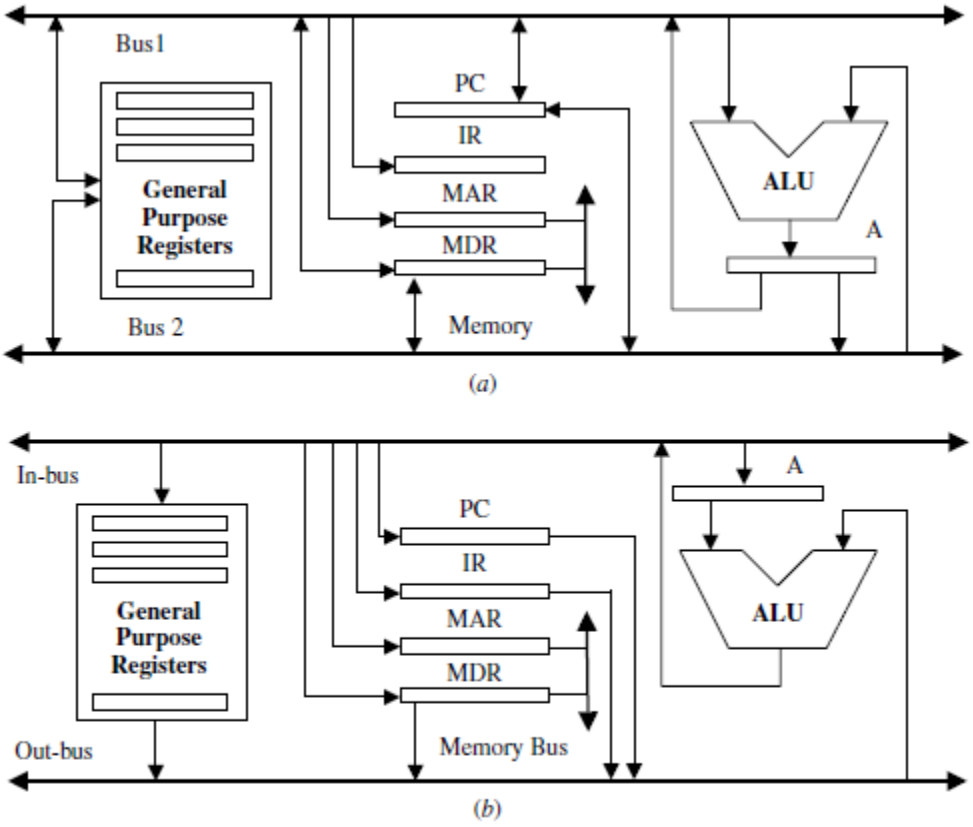


Figure 5.4 Two-bus organizations. (*a*) An Example of Two-Bus Datapath. (*b*) Another Example of Two-Bus Datapath with in-bus and out-bus

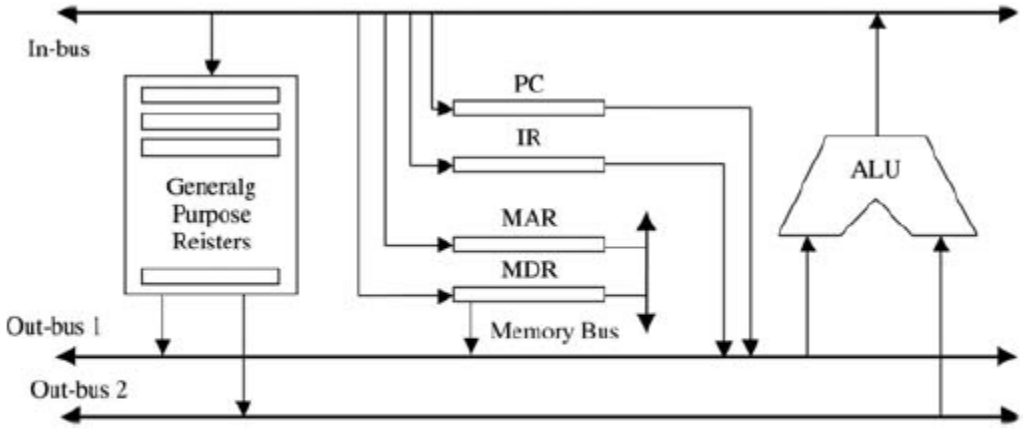


Figure 5.5 Three-bus datapath

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 5.5 shows an example of a three-bus Data Path.

Self-Assessment Question

1. The data path of the CPU contains the register and ALU. True or False?

Self-Assessment Answer

1. True

3.2 CPU Instruction Cycle

The sequence of operations performed by the CPU during its execution of instructions is presented in Fig. 5.6. As long as there are instructions to execute, the next instruction is fetched from main memory. The instruction is executed based on the operation specified in the opcode field of the instruction. At the completion of 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.

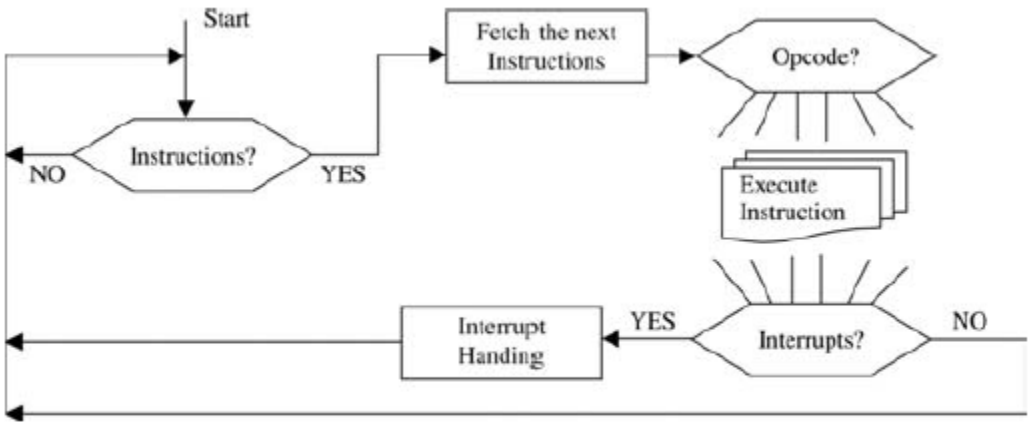


Figure 5.6 CPU functions

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 micro operation.

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

3.2.1 Fetch Instructions

The sequence of events in fetching an instruction can be summarized as follows:

1. The contents of the PC are loaded into the MAR.
2. The value in the PC is incremented. (This operation can be done in parallel with a memory access.)
3. As a result of a memory read operation, the instruction is loaded into the MDR.
4. The contents of the MDR are loaded into the IR.

Let us consider the one-bus Data Path organization shown in Fig. 5.3. We will see that the fetch operation can be accomplished in three steps as shown in the table below, where $t_0 < t_1 < t_2$. Note that multiple operations separated by ";" imply that they are accomplished in parallel.

Step	Micro-operation
t_0	MAR $\leftarrow$ (PC); A $\leftarrow$ (PC)
t_1	MDR $\leftarrow$ Mem[MAR]; PC $\leftarrow$ (A) + 4
t_2	IR $\leftarrow$ (MDR)

Using the three-bus Data Path shown in Figure 5.5, the following table shows the steps needed.

Step	Micro-operation
t_0	MAR $\leftarrow$ (PC); PC $\leftarrow$ (PC) + 4
t_1	MDR $\leftarrow$ Mem[MAR]
t_2	IR $\leftarrow$ (MDR)

3.2.2 Execute Simple Arithmetic Operation

Add R_1, R_2, R_0 This instruction adds the contents of source registers R_1 and R_2, and stores the results in destination register R_0. This addition can be executed as follows:

1. The registers R_0 , R_1 , R_2 , are extracted from the IR.
2. The contents of R1 and R2 are passed to the ALU for addition.
3. The output of the ALU is transferred to R0 .

Using the one-bus Data Path shown in Figure 5.3, this addition will take three steps as shown in the following table, where $t_0 < t_1 < t_2$.

Step	Micro-operation
t_0	$A \leftarrow (R_1)$
t_1	$B \leftarrow (R_2)$
t_2	$R_0 \leftarrow (A) + (B)$

Using the two-bus Data Path shown in Figure 5.4a, this addition will take two steps as shown in the following table, where $t_0 < t_1$.

Step	Micro-operation
t_0	$A \leftarrow (R_1) + (R_2)$
t_1	$R_0 \leftarrow (A)$

Using the two-bus Data Path with in-bus and out-bus shown in Figure 5.4b, this addition will take two steps as shown below, where $t_0 < t_1$.

Step	Micro-operation
t_0	$A \leftarrow (R_1)$
t_1	$R_0 \leftarrow (A) + (R_2)$

Using the three-bus Data Path shown in Figure 5.5, this addition will take only one step as shown in the following table.

Step	Micro-operation
t_0	$R_0 \leftarrow (R_1) + (R_2)$

Add X, R_0 This instruction adds the contents of memory location X to register R_0 and stores the result in R_0 . This addition can be executed as follows:

1. The memory location X is extracted from IR and loaded into MAR.
2. As a result of memory read operation, the contents of X are loaded into MDR.
3. The contents of MDR are added to the contents of R_0 .

Using the one-bus Data Path shown in Figure 5.3, this addition will take five steps as shown below, where $t_0 < t_1 < t_2 < t_3 < t_4$.

Step	Micro-operation
t_0	MAR $\leftarrow$ X
t_1	MDR $\leftarrow$ Mem[MAR]
t_2	$A \leftarrow (R_0)$
t_3	$B \leftarrow (MDR)$
t_4	$R_0 \leftarrow (A) + (B)$

Using the two-bus Data Path shown in Figure 5.4a, this addition will take four steps as shown below, where $t_0 < t_1 < t_2 < t_3$.

Step	Micro-operation
t_0	MAR $\leftarrow$ X
t_1	MDR $\leftarrow$ Mem[MAR]
t_2	$A \leftarrow (R_0) + (MDR)$
t_3	$R_0 \leftarrow (A)$

Using the two-bus Data Path with in-bus and out-bus shown in Figure 5.4b, this addition will take four steps as shown below, where $t_0 < t_1 < t_2 < t_3$.

Step	Micro-operation
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	$A \leftarrow (R_0)$
t_3	$R_0 \leftarrow (A) + (MDR)$

Using the three-bus Data Path shown in Figure 5.5, this addition will take three steps as shown below, where $t_0 < t_1 < t_2$.

Step	Micro-operation
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	$R_0 \leftarrow R_0 + (MDR)$

3.3.1 Interrupt Handling

After the execution of an instruction, a test is performed to check for pending interrupts. If there is an interrupt request waiting, the following steps take place:

1. The contents of PC are loaded into MDR (to be saved).
2. The MAR is loaded with the address at which the PC contents are to be saved.
3. The PC is loaded with the address of the first instruction of the interrupt handling routine.
4. The contents of MDR (old value of the PC) are stored in memory.

The following table shows the sequence of events, where $t_1 < t_2 < t_3$.

Step	Micro-operation
t_1	MDR ← (PC)
t_2	MAR ← address1 (where to save old PC);
	PC ← address2 (interrupt handling routine)
t_3	Mem[MAR] ← (MDR)

3.3 Control Unit

The control unit is the main component that directs the system operations by sending control signals to the Data Path. These signals control the flow of data within the CPU and between the CPU and external units such as memory and I/O. Control buses generally carry signals between the control unit and other computer components in a clock-driven manner. The system clock produces a continuous sequence of pulses in a specified duration and frequency. A sequence of steps t_0 , t_1 , t_2 , . . . ,

Self-Assessment Question

1. Which component directs the system operation by sending control signals to the data path?

Self-Assessment Answer

1. The Control Unit

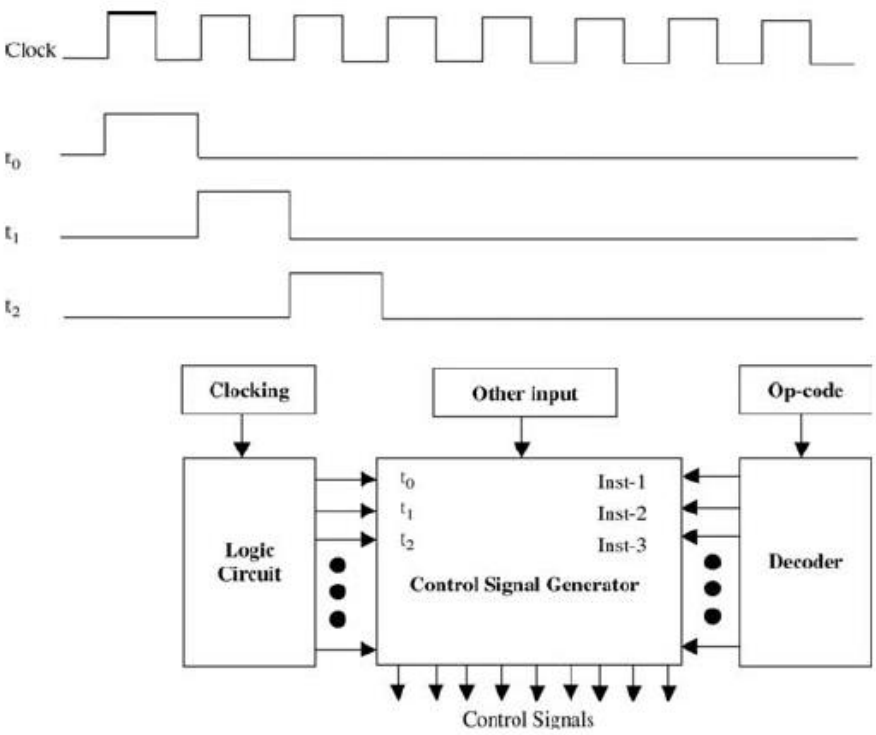


Figure 5.7 Timing of control signals

($t_0 < t_1 < t_2 < \ldots$) are used to execute a certain instruction. The op-code field of a fetched instruction is decoded to provide the control signal generator with information about the instruction to be executed. Step information generated by a logic circuit module is used with other inputs to generate control signals. The signal generator can be specified simply by a set of Boolean equations for its output in terms of its inputs.

Figure 5.7 shows a block diagram that describes how timing is used in generating control signals. There are mainly two different types of control units: microprogrammed and hardwired. In microprogrammed control, the control signals associated with operations are stored in special memory units inaccessible by the programmer as control words. A control word is a microinstruction that specifies one or more micro operations.

A sequence of microinstructions is called a microprogram, which is stored in a ROM or RAM called a control memory CM. In hardwired control, fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals. Clearly hardwired control is faster than microprogrammed control. However, hardwired control could be very expensive and complicated for complex systems. Hardwired control is more economical for small control units. It should also be noted that microprogrammed control could adapt easily to changes in the system design. We can easily add new instructions without changing hardware. Hardwired control will require a redesign of the entire systems in the case of any change.

Example 1: Let us revisit the add operation in which we add the contents of source registers R_1 , R_2 , and store the results in destination register R_0 . We have shown earlier that this

operation can be done in one step using the three-bus Data Path shown in Figure 5.5. Let us try to examine the control sequence needed to accomplish this addition at step t_0.

Suppose that the op-code field of the current instruction was decoded to Inst-x type. First we need to select the source registers and the destination register, then we select Add as the ALU function to be performed. The following table shows the needed step and the control sequence.

Step	Instruction type	Micro-operation	Control
t_0	Inst-x	$R_0 \leftarrow (R_1) + (R_2)$	Select R_1 as source 1 on out-bus1 (R_1 out-bus1) Select R_2 as source 2 on out-bus2 (R_2 out-bus2) Select R_0 as destination on in-bus (R_0 in-bus) Select the ALU function Add (Add)

Figure 5.8 shows the signals generated to execute Inst-x during time period t_0. The AND gate ensures that these signals will be issued when the op-code is decoded into Inst-x and during time period t_0 . The signals (R_1 out-bus 1), (R_2 out-bus2),

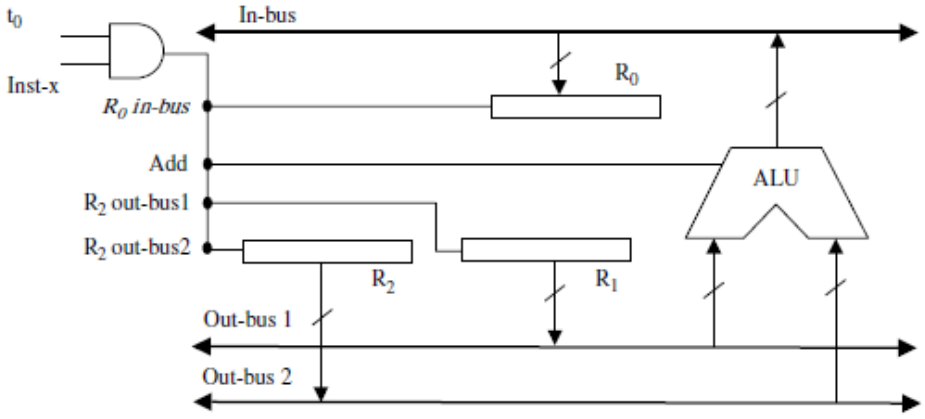


Figure 5.8 Signals generated to execute Inst-x on three-bus datapath during time period t_0

(R_0 in-bus), and (Add) will select R_1 as a source on out-bus1, R_2 as a source on outbus2, R_0 as destination on in-bus, and select the ALUs add function, respectively.

3.3.2 Hardwired Implementation

In hardwired control, a direct implementation is accomplished using logic circuits. For each control line, one must find the Boolean expression in terms of the input to the control signal generator as shown in Figure 5.7. Let us explain the implementation using a simple example.

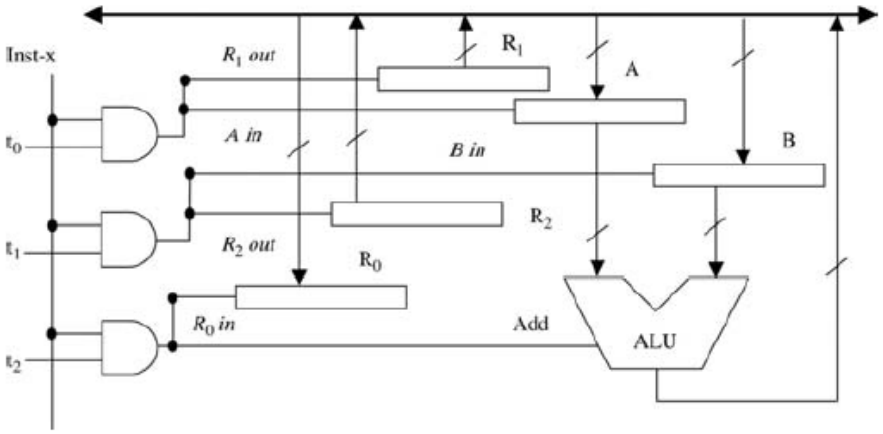


Figure 5.9 Signals generated to execute Inst-x on one-bus datapath during time period t_0, t_1, t_2

Assume that the instruction set of a machine has the three instructions: Inst-x, Inst-y, and Inst-z; and A, B, C, D, E, F, G, and H are control lines. The following table shows the control lines that should be activated for the three instructions at the three steps t_0 , t_1 , and t_2 .

Step	Inst-x	Inst-y	Inst-z
t_0	D, B, E	F, H, G	E, H
t_1	C, A, H	G	D, A, C
t_2	G, C	B, C	

The Boolean expressions for control lines A, B, and C can be obtained as follows:

$$\begin{aligned}
\text{A} &= \text{Inst-x} \cdot t_1 + \text{Inst-z} \cdot t_1 = (\text{Inst-x} + \text{Inst-z}) \cdot t_1 \\
\text{B} &= \text{Inst-x} \cdot t_0 + \text{Inst-y} \cdot t_2 \\
\text{C} &= \text{Inst-x} \cdot t_1 + \text{Inst-x} \cdot t_2 + \text{Inst-y} \cdot t_2 + \text{Inst-z} \cdot t_1 \\
&= (\text{Inst-x} + \text{Inst-z}) \cdot t_1 + (\text{Inst-x} + \text{Inst-y}) \cdot t_2
\end{aligned}$$

Figure 5.10 shows the logic circuits for these control lines. Boolean expressions for the rest of the control lines can be obtained in a similar way. Figure 5.11 shows the state diagram in the execution cycle of these instructions.

3.3.3 Microprogrammed Control Unit

The idea of microprogrammed control units was introduced by M. V. Wilkes in the early 1950s. Microprogramming was motivated by the desire to reduce the complexities involved with hardwired control. As we studied earlier, an instruction is implemented using a set of micro-operations. Associated with each micro-operation is a set of control lines that must be activated to carry out the corresponding micro operation. The idea of microprogrammed control is to store the control signals associated with the implementation of a certain instruction as a microprogram in a special memory called a control memory (CM).

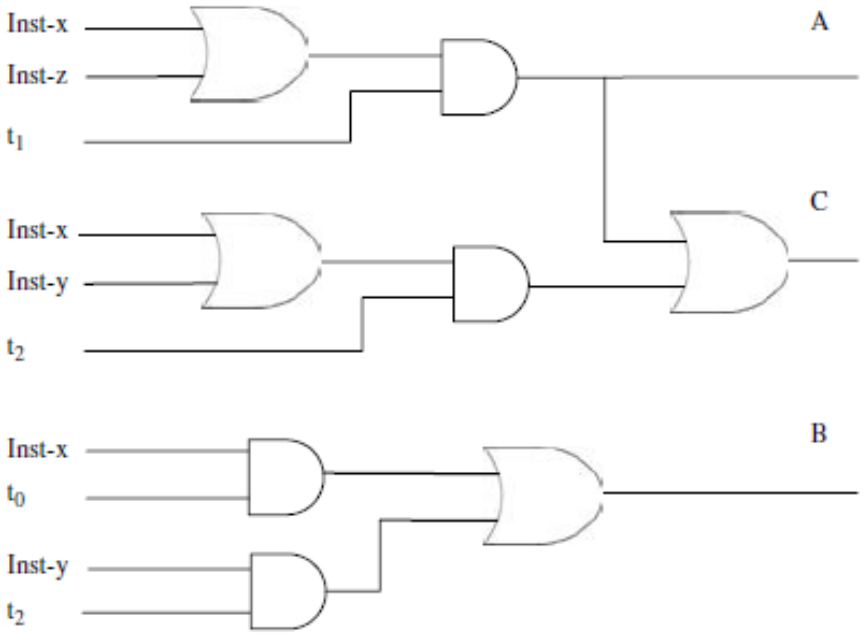


Figure 5.10 Logic circuits for control lines A, B, and C

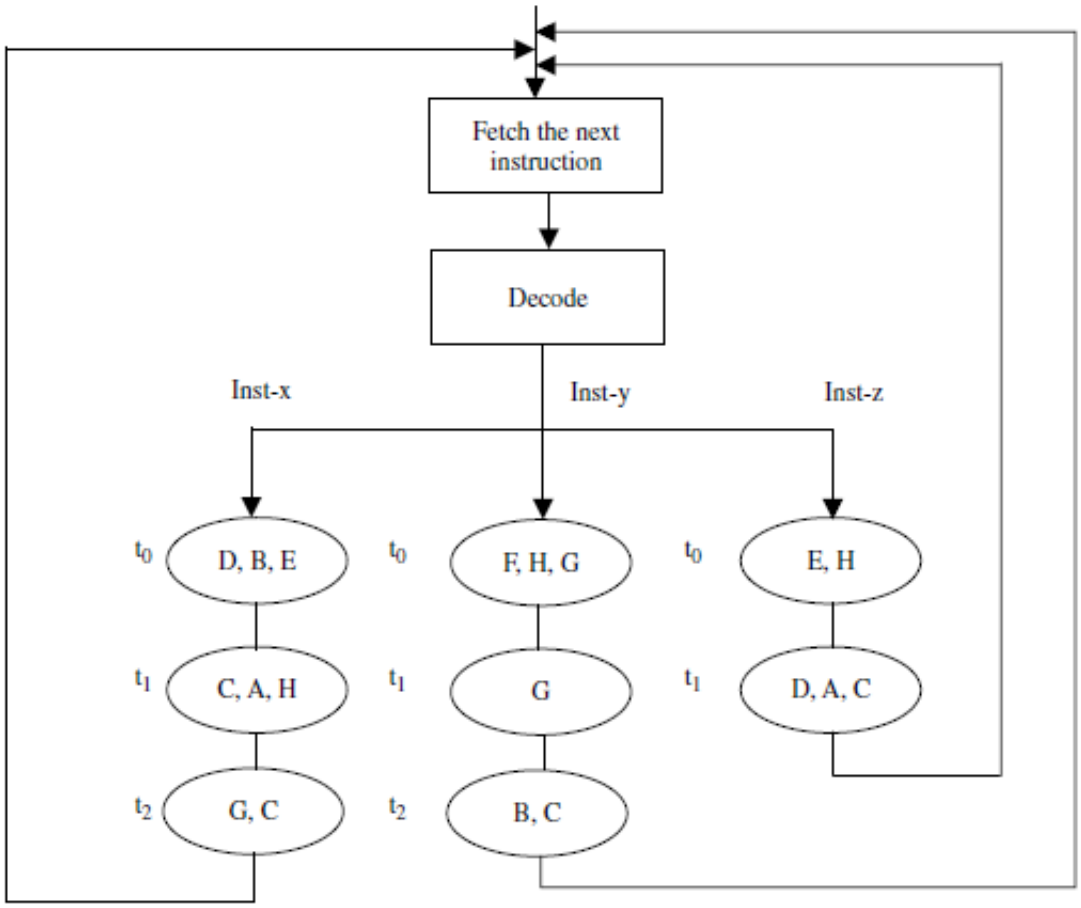


Figure 5.11 Instruction execution state diagram

A microprogram consists of a sequence of microinstructions. A microinstruction is a vector of bits, where each bit is a control signal, condition code, or the address of the next

microinstruction. Microinstructions are fetched from CM the same way program instructions are fetched from main memory (Fig. 5.12). When an instruction is fetched from memory, the op-code field of the instruction will determine which microprogram is to be executed.

In other words, the op-code is mapped to a microinstruction address in the control memory. The microinstruction processor uses that address to fetch the first microinstruction in the microprogram. After fetching each microinstruction, the appropriate control lines will be enabled. Every control line that corresponds to a "1" bit should be turned on. Every control line that corresponds to a "0" bit should be left off. After completing the execution of one microinstruction, a new microinstruction will be fetched and executed. If the condition code bits indicate that a branch must be taken, the next microinstruction is specified in the address bits of the current microinstruction. Otherwise, the next microinstruction in the sequence will be fetched and executed. The length of a microinstruction is determined based on the number of micro operations specified in the microinstructions, the way the control bits will be interpreted, and the way the address of the next microinstruction is obtained.

A microinstruction may specify one or more micro-operations that will be activated simultaneously. The length of the microinstruction will increase as the number of parallel micro-operations per microinstruction increases. Furthermore, when each control bit in the microinstruction corresponds to exactly one control line, the length of microinstruction could get bigger.

The length of a microinstruction could be reduced if control lines are coded in specific fields in the microinstruction. Decoders will be needed to map each field into the individual control lines. Clearly, using the decoders will reduce the number of control lines that can be activated simultaneously.

There is a tradeoff between the length of the microinstructions and the amount of parallelism. It is important that we reduce the length of microinstructions to reduce the cost and access time of the control memory. It may also be desirable that more micro-operations be performed in parallel and more control lines can be activated simultaneously.

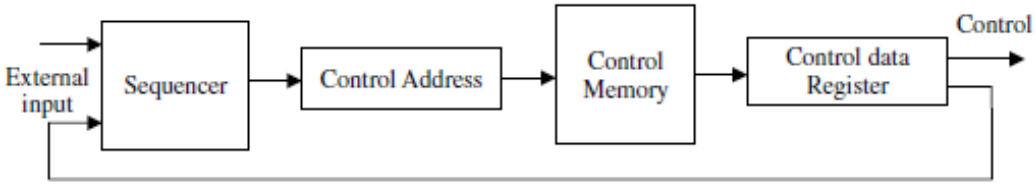


Figure 5.12 Fetching microinstructions (control words)

Self-Assessment Question

1. The idea of microprogrammed control units was introduced by _____.

Self-Assessment Answer

1. The idea of microprogrammed control units was introduced by M. V. Wilkes in the early 1950s.

4.0 Conclusion

What you have learned in this unit is on data path and control unit. Also, you have learnt about one-bus organization, two-bus organization and three-bus organization. Furthermore, you learnt about the CPU instruction cycle, fetch instructions, execute simple arithmetic operation, interrupt handling, control unit, hardwired implementation and microprogrammed control unit.

5.0 Summary

The CPU is the part of a computer that interprets and carries out the instructions contained in the programs we write. The CPU's main components are the register file, ALU, and the control unit. The register file contains general-purpose and special registers. General-purpose registers may be used to hold operands and intermediate results.

The special registers may be used for memory access, sequencing, status information, or to hold the fetched instruction during decoding and execution. Arithmetic and logical operations are performed in the ALU. Internal to the CPU, data may move from one register to another or between registers and ALU. Data may also move between the CPU and external components such as memory and I/O.

The control unit is the component that controls the state of the instruction cycle. As long as there are instructions to execute, the next instruction is fetched from main memory. The instruction is executed based on the operation specified in the op-code field of the instruction. The control unit generates signals that control the flow of data within the CPU and between the CPU and external units such as memory and I/O. The control unit can be implemented using hardwired or microprogramming techniques.

6.0 Tutor-Marked Assignment

1. Explain hardwired control implementation with a simple example?
2. Explain the process of interrupt handling in control unit?
3. Give a sketch of central processing unit and give summary of a typical and simple execution cycle?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://www.adaptec.com

http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_raid

Module 5

Instruction Pipelining, RISCs & Multiprocessors

Unit 1

Introduction to Pipelining

Contents

1.0 Introduction

In this unit, what you will learn rotates around pipelining design techniques. There exist 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 performing 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.

2.0 Learning Outcome

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

i. Explain pipelining

ii. Describe pipelining design techniques

iii. Explain example pipeline processors

iv. Describe Instruction-Level Parallelism

3.0 Learning Content

3.1 Pipelining

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 the performance compared to the traditional sequential execution of tasks.

Figure 9.1 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.

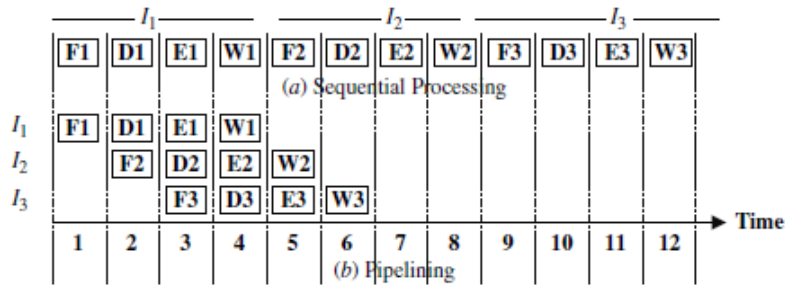


Figure 9.1 Pipelining versus sequential processing

It is clear from the figure that the total time required to process three instructions (I_1, I_2, I_3) 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. The chart shows the succession of the subtasks in the pipe with respect to time. Figure 9.2 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 (I_1 to I_{10}). 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.

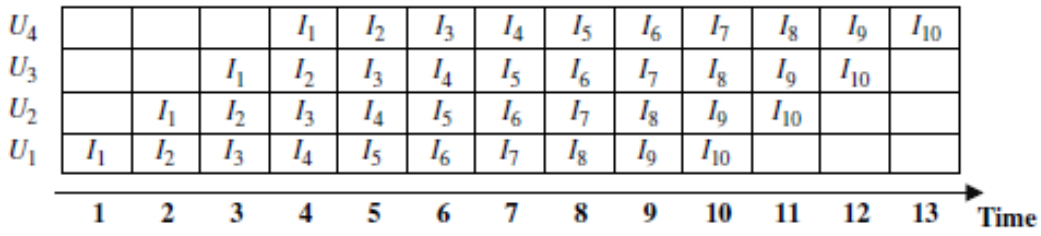


Figure 9.2 The space–time chart (Gantt chart)

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

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

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

2. *Throughput U(n)*

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

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

3. *Efficiency E(n)*

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

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

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

Self-Assessment Question

1. What is Pipelining?

Self-Assessment Answer

1. Pipelining refers to the technique in which a given task is divided into a number of subtasks that need to be performed in sequence.

3.2 Instruction Pipeline

The simple analysis made in Section 9.1 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. Consider, for example, the case of an instruction fetch that incurs a cache miss. Assume also that a cache miss requires three extra time units. Figure 9.3 illustrates the effect of having instruction I2 incurring a cache miss (assuming the execution of ten instructions I1 to I10).

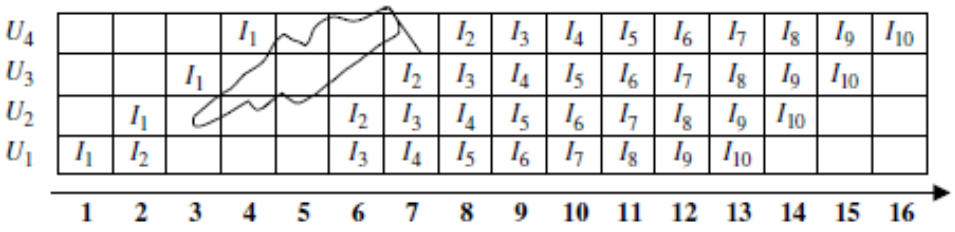


Figure 9.3 Effect of a cache miss on the pipeline

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 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 9.3 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.

3.2.1 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: Consider the execution of ten instructions I_1–I_{10} 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 I_4 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 9.4 shows the required Gantt's chart. The bubble created due to the pipeline stall is clearly shown in the figure.

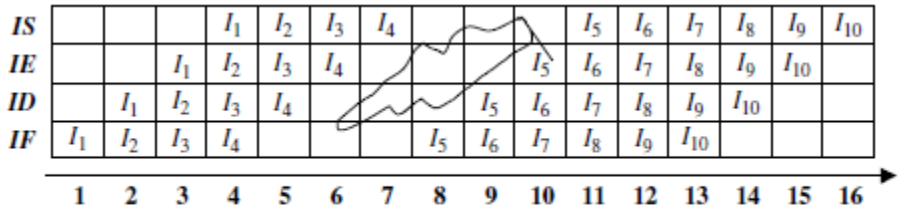


Figure 9.4 Instruction dependency effect on a pipeline

3.2.2 Pipeline "Stall" Due to Data Dependency

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

3.3 Example Pipeline Processors

We briefly present two pipeline processors that use a variety of the 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.

1. 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.
2. 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:
    i. Few and simple instruction formats. All instructions are 32-bit. Memory access is done exclusively using Load and Store instructions.
    ii. Few addressing modes. Memory addressing has only two modes, the Register + Register and the Register + Immediate modes.
    iii. 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.
    iv. Large window register file.

3.4 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 Data Paths for instruction execution. Each of these pipelines can issue and execute one instruction per cycle. Figure 9.17 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 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.

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 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 a number of 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.

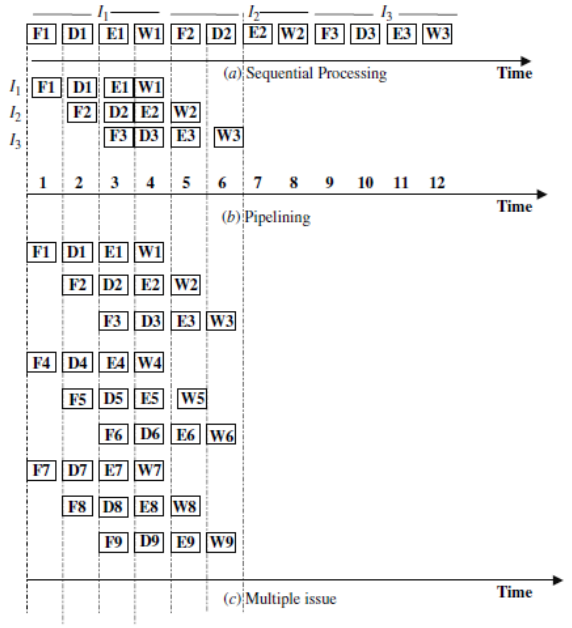


Figure 9.17 Multiple issue versus pipelining versus sequential processing

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. Pre-decoding 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.

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 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 operation 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 multiway 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.

4.0 Conclusion

What you have learned in this unit is on pipelining techniques. Also, you have learnt about definition of pipelining, example of pipeline processors and instruction level parallelism.

5.0 Summary

There exist 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. 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 the performance compared to the traditional sequential execution of tasks. 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. 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. We briefly present two pipeline processors that use a variety of the 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. Contrary to pipeline techniques, instruction-level parallelism (ILP) is based on the idea of multiple issue processors (MIP). An MIP has multiple pipelined Data Paths for instruction execution. Each of these pipelines can issue and execute one instruction per cycle.

6.0 Tutor-Marked Assignment

1. The merits of pipelining cannot be over-emphasized in computing. Explain this statement?
2. Briefly explain instruction pipeline?

3. Explain the UltraSPARC III Processor as an example of pipelining?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

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/

http://www.adaptec.com

http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_raid

http://www.ar.com

http://www.arm.com/support/White_Papers

http://www.sun.com/ultrasparc

Unit 2

Introduction to Reduced Instruction Set Computers (RISCs)

Contents

1.0 Introduction

This unit is dedicated to a study of reduced instruction set computers (RISCs). These machines represent a noticeable shift in 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 showed also that among all operations, procedure calls/return 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 reality and they 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 alternatively 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 the 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.

2.0 Learning Outcome

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

1. Explain RISC/CISC evolution cycle
2. Describe RISCs design principles
3. Explain overlapped register windows
4. Describe RISCs versus CISCs
5. Explain pioneer (University) RISC machines
6. Give example of advanced RISC machines

3.0 Learning Content

3.1 RISC/CISC Evolution Cycle

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 a number of disadvantages. These 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.”

A number of studies were then conducted to investigate the impact of complexity on performance. These are discussed below

Self-Assessment Question

1. What is RISC?

Self-Assessment Answer

1. 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.

3.2 RISCs Design Principles

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 a number of aspects of computation.

These aspects should include

1. Operations that are most frequently performed during execution of typical (benchmark) programs,

2. Operations that are most time consuming, and
3. The type of operands that are most frequently used. A number of early studies were conducted in order to find out the typical breakdown of operations that are performed in executing benchmark programs.

The estimated distribution of operations is shown in Table 10.1. 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%.

TABLE 10.1 Estimated Distribution of Operations

Operations	Estimated percentage
Assignment statements	35
Loops	5
Procedure calls	15
Conditional branches	40
Unconditional branches	3
Others	2

In addition to the above findings, studies on time–performance characteristics of operations revealed that among all operations, procedure calls/return 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 behavior have led to the following conclusions:

1. 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/return are the most time-consuming operations and 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 overlapped register window. This is explained below.

3.3 Overlapped Register Windows

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 procedure without actual movement of data (Fig. 10.1).

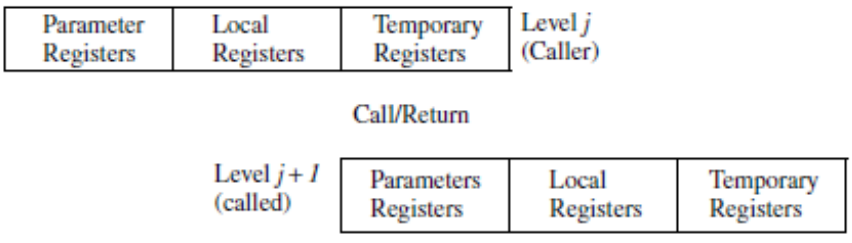


Figure 10.1 Register window overlapping

TABLE 10.2 Different Register Windows Characteristics

Architecture	Number of windows	Number of registers per window
Berkeley RISC-I	8	16
Pyramids	16	32
SPARC	32	32

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 normally called the current window pointer (CWP).

Upon having all windows filled, the register window wraps around, thus acting like a "circular buffer." Table 10.2 shows the number of windows and the window size for a number of architectures. It should be noted that a study was conducted in 1985 to find out the impact of using register window on the performance of the Berkeley RISC. In this study, two versions of the machine were studied.

The first is 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.

3.4 RISCs versus CISCs

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 a 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 10.3 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 (see the details of the SPARC architecture in the next section). 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 10.3 and 10.4 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. Limited number of instructions (128 or less)
3. 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

TABLE 10.4 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 10.5 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

6. Pipelined instruction execution
7. 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

3.5 Pioneer (University) RISC Machines

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.

3.5.1 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; $R_d \leftarrow M[R_x + S]$

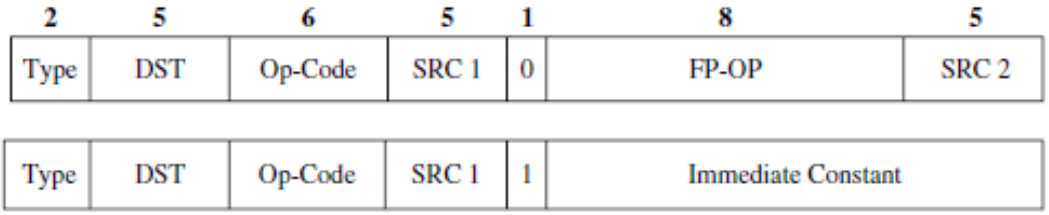


Figure 10.2 Three operand instructions formats used in RISC

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

All arithmetic and logical instructions have three operands and have the form Destination: = source1 op source2 (Fig. 10.2). 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. 10.3). 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.

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 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 10.4.
10.5.2. 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

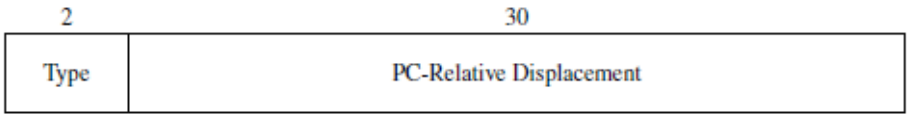


Figure 10.3 Procedure call instruction in RISC

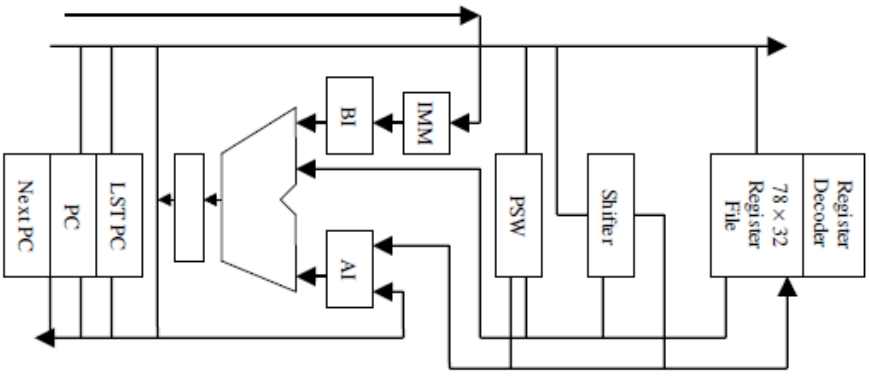


Figure 10.4 RISC four-bus organization

(OD), Operand Store/Execution (OS/EX), and Operand Fetch (OF). The first three stages perform respectively instruction fetch, instruction decode, and operand fetch. The OS/EX stage sends operand to memory in the case of a store instruction or use 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. 10.5). 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 10.5. A total of six control flow instructions were provided.

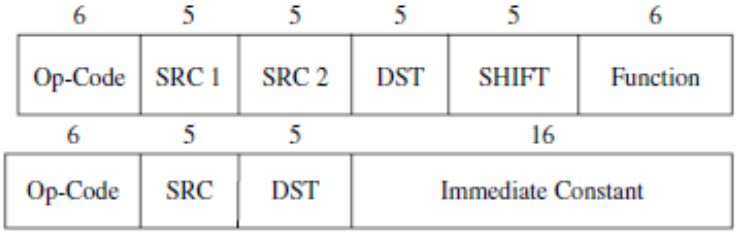


Figure 10.5 Three-operand instructions used in MIPS

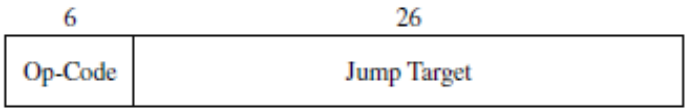


Figure 10.6 Jump instruction format used in MIPS

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 call can be made through the JUMP instruction shown in Figure 10.6.

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 2^{24} = 16 Mwords (32 bits each). A support for off-chip TLB for address translation is provided. The MIPS organization is shown in Figure 10.7.

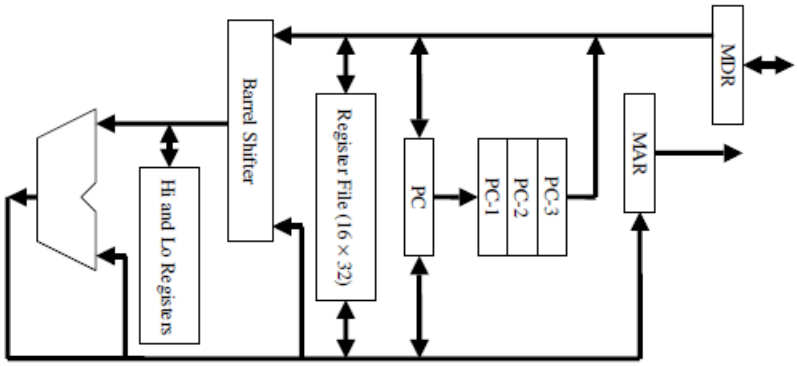


Figure 10.7 MIPS organization

3.6 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.

3.6.1 Compaq (Formerly DEC) Alpha 21264

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. It employs a two-level cache. The L1 data and instruction caches

are 64 KB each. They 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 byte) cache block fills and 8 cache misses.

It has a 64 KB, two-way set-associative cache (both instruction and data). It can also support up to two out-of-order operations (Fig. 10.8).

3.6.2 The Alpha 21264 Pipeline

The Alpha 21264 instruction pipeline is shown in Figure 10.9. It consists of SEVEN stages. These are the Fetch, Slot Assignment, Rename, Issue, Register Read, Execute, and Memory stages. The fetch stage can fetch and execute up to four instructions per cycle. A block diagram of the fetch stage is shown in Figure 10.10. This stage uses a unique “block and set” prediction technique.

According to this technique, both the locations of the next four instructions and the set (there are two sets) in which they are located, are predicted. The “block and set” prediction technique combines the speed advantages of a direct-mapped cache with the lower miss ratio of a two-way set-associative cache.

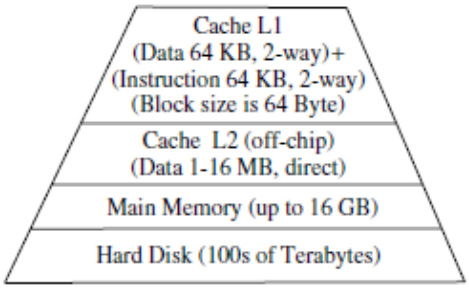


Figure 10.8 The 21264 memory hierarchy

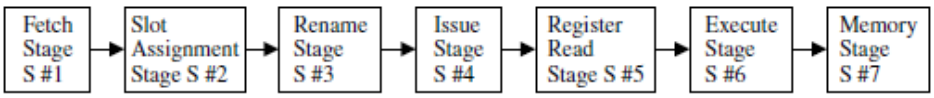


Figure 10.9 The 21264 instruction pipeline

This technique achieves more than an 85% hit ratio. The misprediction penalty is a single cycle. The 21264 uses speculative branch prediction. Branch prediction in the 21264 is a two-level scheme. It is based on the observation that branches exhibit both local and global correlation. Local correlation makes use of the branch’s past behavior. Global correlation, on the other hand, makes use of the past behavior of all previous branches.

The combined local/global prediction used in the 21264 correlates the branch behavior pattern with local branch history, that is, the execution of a single branch at a unique PC location, and global branch history, that is, the execution of all previous branches. The scheme dynamically selects between local and global branch history (Fig. 10.11). The local branch predictor has two tables.

The first is a 1024_10 local history table in which each entry holds a 10-bit local history of the selected branch over the last executions. The local history table is indexed by the instruction address (using the PC). The second table is a 1024_3 local prediction table in which each entry has a 3-bit saturating counter to predict the branch outcome. After branches' retirement, the 21264 updates the local history table with the true branch direction and the referenced counter.

This enhances the possibility for correct prediction and is called predictor training.

The global branch predictor has a 4096_2 global prediction table in which each entry holds a 2-bit saturating counter. It keeps track of the global history of the last 12 branches. The global branch prediction table is indexed by a 4096_2 choice prediction table.

After branches' retirement, the 21264 updates the referenced global prediction counter, enhancing the possibility for correct prediction. Local prediction is useful in the case of an alternating taken/not-taken sequence of a given branch. In this case, the local history of the branch will eventually resolve to a pattern of ten alternating zeros and ones indicating the success, or failure, of the branch on alternate encounters. As the branch executes multiple times, it saturates the prediction counters corresponding to the local history values and hence makes the prediction correct.

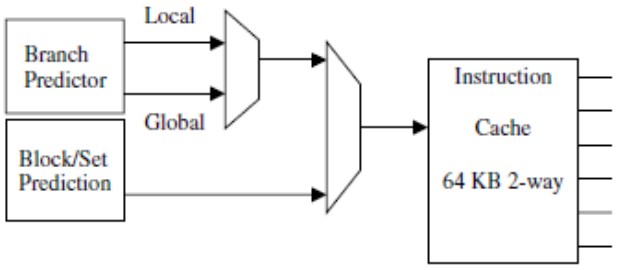


Figure 10.10 The 21264 fetch stage

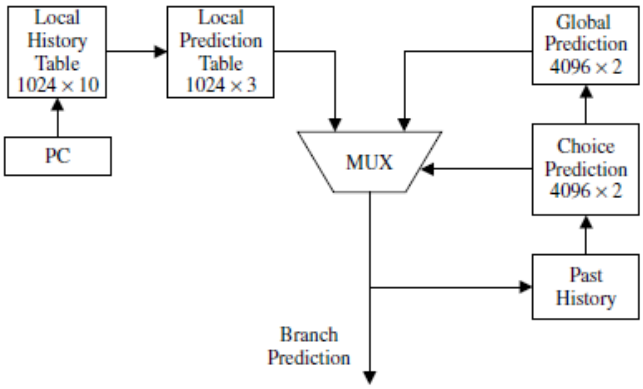


Figure 10.11 The 21264 selection branch predictor

Global prediction is useful when the outcome of a branch can be inferred from the direction of previous branches. Consider, for example, the case of repeated invocations of two branches. If the first branch that checks for a value equal to 1001 succeeds, the second branch that checks for the same value to be odd must also succeed. The global history predictor can learn this pattern with repeated invocations of these two branches.

The 2096 x 2 choice predictor is a table in which each entry holds a 2-bit saturating counter and is used to implement the selection (tournament) scheme. If the predictions of the local

and global predictors differ, the 21264 updates the selected choice prediction entry to support the correct predictor. The 21264 updates the choice prediction table when a branch retires. The slot assignment stage (S #2) simply assigns instructions to slots associated with the integer and the floating-point queues.

The out-of-order (OOO) issue logic in the 21264 receives four fetched instructions every cycle, renames and remaps the registers (to avoid unnecessary register dependencies), and queues the instructions until operands and/or functional units become available. It dynamically issues up to six instructions every cycle, four integers and two floating-point instructions.

Register renaming means mapping instruction virtual registers to internal physical registers. There are 31 integers and 31 floating-point registers that are visible to users. These registers are renamed during execution to internal registers. It is only when instructions are finished (retired) that the internal registers are renamed back to visible registers. Register renaming eliminates write-after-write and write-after-read data dependencies.

However, it preserves all the read-after-write dependencies that are necessary for correct computation. A list of the pending instructions is maintained by the OOO queue logic. In each cycle, both the integer and the floating-point queues select those instructions that are ready to execute. This selection is made based on a scoreboard of the renamed registers.

The scoreboard maintains the status of renamed registers by tracking the progress of single-cycle, multiple-cycle, and variable-cycle instructions. Upon the availability of the functional unit(s) or load data results, the scoreboard unit notifies all instructions in the queue of the availability of the required register value. Each queue selects the oldest data-ready and functional-unit-ready instructions for execution of each cycle.

The 21264 integer queue statically assigns instructions to two of four pipes, either the upper or the lower pipe (Fig. 10.12). The Alpha 21264 has four integers and two floating-point pipelines. This allows the processor to dynamically issue up to six instructions in the same cycle. The issue (or queue) stage maintains an inventory from which it can dynamically select to issue a maximum of six instructions.

There is a 20-entry integer issue queue and a 15-entry floating-point issue queue. Instruction issue reordering takes place in the issue stage. The 21264 uses two integer files, 80-entry each, to store a duplicate of register contents. Two pipes access a single file to form a cluster. The two clusters form a four-way integer instruction execution. Results are broadcasted from each cluster to the other cluster.

Instructions are dynamically selected by the integer issue queue to execute on a given instruction pipe. An instruction can heuristically be selected to execute on the same cluster that produces the result. The 21264 has one 72-entry floating-point register file. The floating-point register file, together with two instruction execution pipes, form a cluster. Figure 10.12 shows the register read/execution pipes.

On a final note, we should indicate that the 21264 uses a write-invalidate cache coherence mechanism in the level 2 cache to provide support for shared-memory multiprocessing. It also supports the following cache states: modified, owned, shared, exclusive, and invalid.

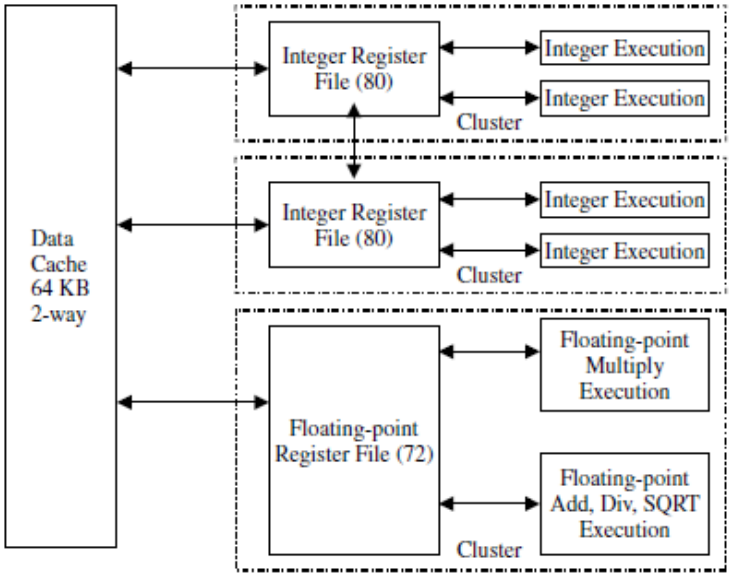


Figure 10.12 The 21264 execution pipes

3.6.3 SUN UltraSPARC III

The UltraSPARCIII is a high-performance superscalar RISC processor that implements the 64-bit SPARC®-V9 RISC architecture. There exist a number of implementations of the SPARC III processor. These include the UltraSPARC IIIi and the UltraSPARC III Cu. Our coverage in this section will be independent of any particular implementation.

We will however refer to specific implementations whenever appropriate. The UltraSPARC III is a third generation 64-bit SPARCw RISC microprocessor. It supports a 64-bit virtual address space and a 43-bit physical address space. The UltraSPARC III employs a multilevel cache architecture.

For example, the Ultra-SPARC IIIi (and the UltraSPARC III Cu) architecture has a 32 KB, four-way set associative L1 instruction cache, a 64 KB four-way set-associative L1 data cache, a 2 KB prefetch cache, and a 2 KB write cache. The UltraSPARC IIIi supports a 1 MB four-way set-associative, unified instruction/data on chip L2 cache. A cache block size of 64 bytes is used in the UltraSPARC IIIi. While the UltraSPARC III Cu architecture supports a 1, 4, or 8 MB two-way set-associative, unified instruction/ data external cache. Cache block size in the UltraSPARC III Cu varies between 64 bytes (for the 1 MB cache) to 512 bytes (for the 8 MB cache) (Fig. 10.13).

The UltraSPARC III uses two instruction TLBs that can be accessed in parallel and three data TLBs that can be accessed in parallel. The two instruction TLBs are organized in a 16-entry fully associative manner to hold entries for 8 KB, 64 KB, 512 KB, and 4 MB page sizes. A 128-entry two-way set-associative TLB is used exclusively for 8 KB page sizes. The three data TLBs are organized in a 16-entry associative manner for 8 KB, 64 KB, 512 KB, and 4 MB page sizes and two 512-entry two-way set-associative TLBs that can be programmed to hold any one page size at a given time. The UltraSPARC III uses a write-allocate, writeback cache write policy.

On a final note, it should be mentioned that the UltraSPARC III has been designed to support a one-to-four-way multiprocessing. For this purpose, it uses the JBus, which supports a small-scale multiprocessor system. The JBus is capable of delivering the high bandwidth needed for

networking and embedded systems applications. Through the JBus, processors can attach to a coherent shared bus with no needed glue logic (Fig. 10.14).

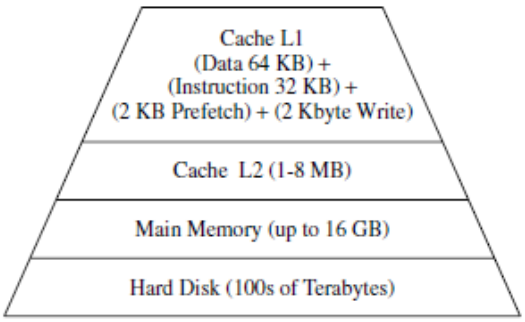


Figure 10.13 UltraSPARC III memory hierarchy

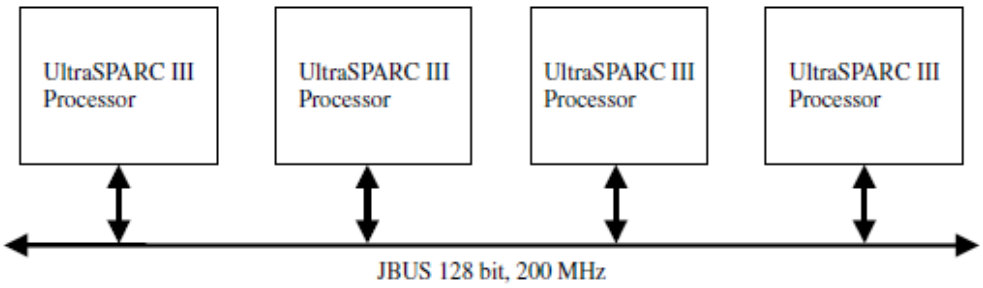


Figure 10.14 A four-way UltraSPARC III multiprocessor configuration

4.0 Conclusion

What you have learned in this unit is on reduced instruction set computers (RISCs). Also, you learnt about RISC/CISC evolution cycle, RISCs design principles, overlapped register windows, RISCs versus CISCs, pioneer (University) RISC machines and example of advanced RISC machines.

5.0 Summary

A RISC architecture saves the extra chip area used by CISC architectures for decoding and executing complex instructions. The saved chip area is then used to provide an on-chip instruction cache that can be used to reduce instruction traffic between the processor and the memory.

Common characteristics shared by most RISC designs are: limited and simple instruction set, large number of general purpose registers and/or the use of compiler technology to optimize register usage, and optimization of the instruction pipeline. An essential RISC philosophy is to keep the most frequently accessed operands in registers and minimize register-memory operations.

This can be achieved using one of two approaches: Software Approach, use the compiler to maximize register usage by allocating registers to those variables that will be used the most in a given time period (this is the philosophy used in Stanford MIPs machine); or Hardware

Approach, use more registers so that more variables can be held in registers for larger periods of time (this is the philosophy used in the Berkeley RISC machine).

Register windows are multiple small sets of registers, each assigned to a different procedure. A procedure call automatically switches the CPU to use a different fixed-size window of registers rather than saving registers in memory at the call time. At any time, only ONE window of registers is visible and is addressed as if it were the only set of registers.

Window overlapping requires that temporary registers at one level are physically the same as the parameter registers at the next level. This overlap allows parameters to be passed without the actual movement of data. It is worthwhile mentioning that the classification of processors as entirely pure RISC or entirely pure CISC is becoming more and more inappropriate and may be irrelevant.

What actually counts is how much performance gain can be achieved by including an element of a given design style. Most modern processors use a calculated combination of elements of both design styles. The decisive factor in which element(s) of each design style to include is made based on a trade-off between the required improvement in performance and the expected added cost.

A number of processors are classified as RISC while employing a number of CISC features, such as integer/floating-point division instructions. Similarly, there exist processors that are classified as CISC while employing a number of RISC features, such as pipelining

6.0 Tutor-Marked Assignment

1. Explain the RISC/CISC Evolution Cycle?
2. Describe the four categories of the RISC instruction set?
3. Give some examples of MIPS instructions?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://www.adaptec.com

http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_raid

http://www.ar.com

http://www.arm.com/support/White_Papers

http://www.sun.com/ultrasparc

http://www.sun.com/processors/UltraSPARC-IIIi

http://www.sun.com/processors/whitepapers

Unit 3

Introduction to Multiprocessors

Contents

1.0 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.

A multiprocessor is expected to reach a faster speed than the fastest uniprocessor. In addition, a multiprocessor consisting of a number of 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 instruction multiple data) multiprocessors are also introduced. Our coverage in this unit ends with a touch on the analysis and performance metrics for multiprocessors.

2.0 Learning Outcome

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

i. Define a Multiprocessor System
ii. Explain Classification of Computer Architectures
iii. Describe SIMD Schemes
iv. Explain MIMD Schemes
v. Describe Interconnection Networks
vi. Explain Analysis and Performance Metrics

3.0 Learning Content

3.1 What is a Multiprocessor System?

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), which can be further classified as
   a. Multiple processors (single address space or shared memory computation) or
   b. Multiple computers (multiple address space or message passing computation). According to PC, networks can be further classified as
2. The physical connection (PC), which can also be further classified as
   a. Bus-based or
   b. Network based multiple processors. Typical sizes of such systems are summarized in Table 11.1.

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. 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.

Figure 11.1 illustrates the four types of interconnection networks mentioned above.

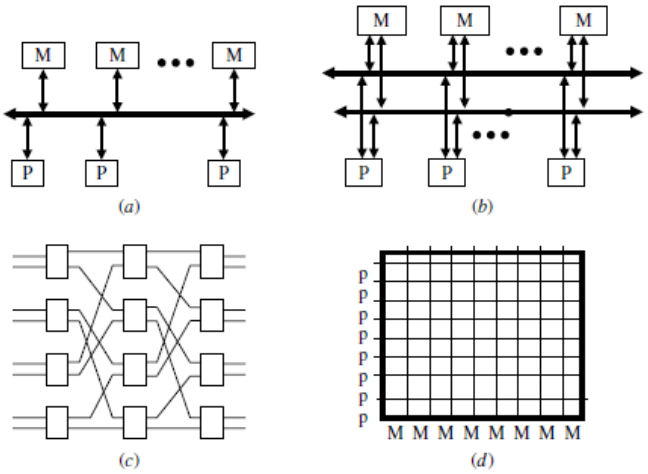


Figure 11.1 Four types of multiprocessor interconnection networks. (*a*) Single-bus system, (*b*) multi-bus system, (*c*) multi-stage interconnection network, (*d*) crossbar system

Self-Assessment Question

1. Describe a Multiprocessor?

Self-Assessment Answer

1. 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.

3.2 Classification of Computer Architectures

A 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.

TABLE 11.1 Typical Sizes of Some Multiprocessor Systems

Category	Subcategories	Number of processors
Communication model	Multiple processors	2–256
	Multiple computers	8–256
Physical connection	Bus-based	2–32
	Network-based	8–256

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 an 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 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.

3.2.1 Flynn's Classification

Flynn's classification scheme is based on identifying two orthogonal streams in a computer. These are the instruction and the 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)
4. Multiple-instruction multiple-data streams (MIMD)

Figure 11.2 shows the orthogonal organization of the streams according to Flynn's classification. Schematics for the four categories of architectures resulting from Flynn's classification are shown in Figure 11.3. Table 11.2 lists some of the commercial machines belonging to each of the four categories.

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 on 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.

		Data Stream	
		Single	Multiple
Instruction Stream	Single	SISD	SIMD
	Multiple	MISD	MIMD

Figure 11.2 Flynn's classification

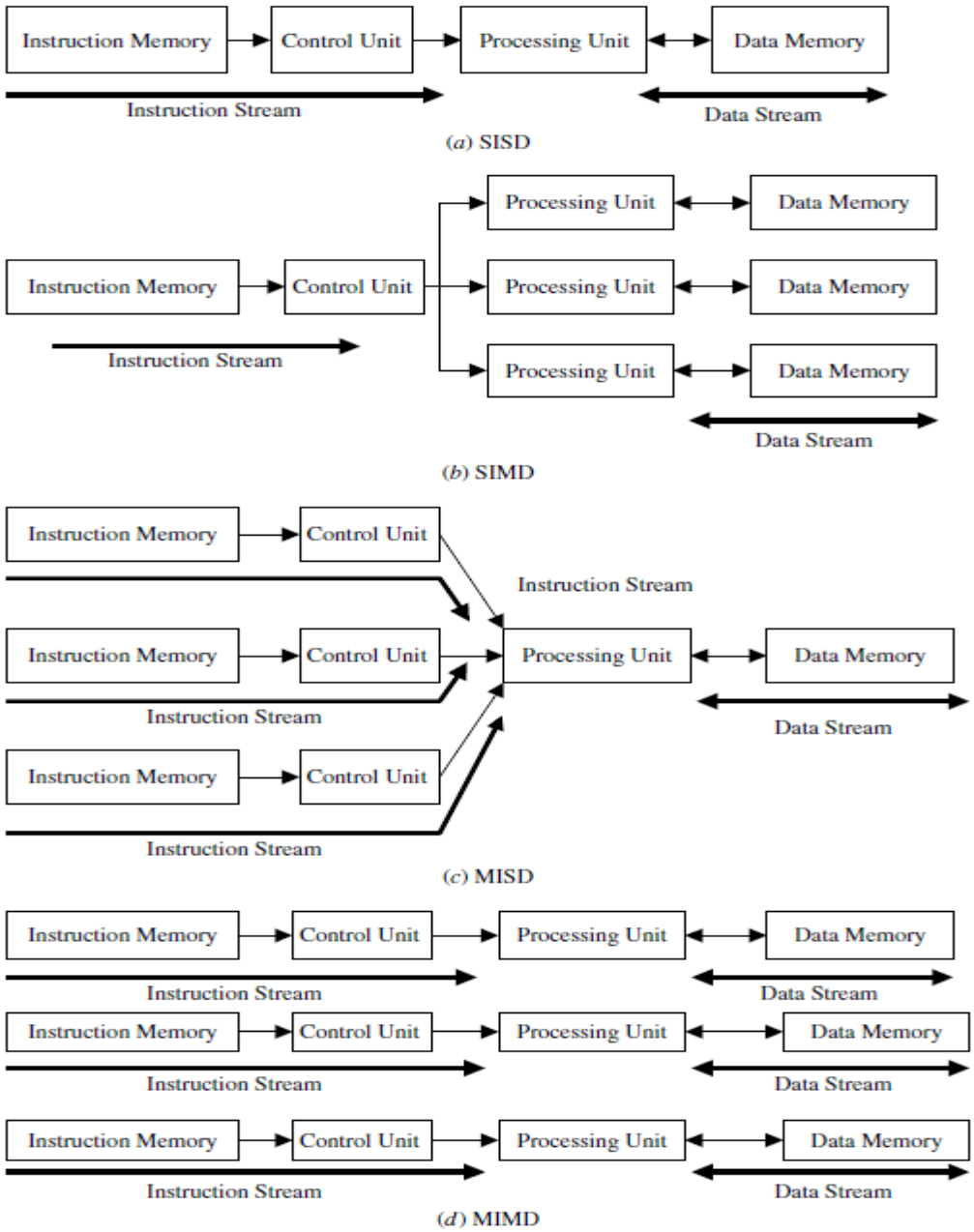


Figure 11.3 The four architecture classes resulting from Flynn's taxonomy. (*a*) SISD, (*b*) SIMD, (*c*) MISD, (*d*) MIMD

TABLE 11.2 Example Machines and Their Flynn's Classification

Classification category	Example machines
SISD	IBM 704, VAX 11/780, CRAY-1
SIMD	ILLIAC-IV, MPP, CM-2, STARAN
MISD	See observation 5 on page 238
MIMD	Cm*, CRAY XMP, IBM 370/168M

3.2.2 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. The combination of these streams results in a total of 16 categories of architectures, as shown in Table 11.3. 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, the Kuck's classification emphasizes the description of architectures at the hardware level.

3.2.3 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 on 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).

TABLE 11.3 The 16-Architecture Categories Resulting from Kuck's Classification

Instruction stream	Execution streams: Single		Execution streams: Multiple	
	Scalar	Array	Scalar	Array
Single				
Scalar	Uniprocessor	Uniprocessor	SIMD	
Array		ILLIAC-IV		
Multiple				
Scalar			NYU Ultracomputer	Cray X MP
Array				

The MIMD category is further refined into loosely coupled MIMD (MIMD-L) and tightly coupled MIMD (MIMD-T). The SIMD category is further refined into word-sliced processing (SIMD-W) and bit-sliced processing (SIMD-B). Therefore, Hwang and Briggs classification added a level to the hierarchy of machine classification such that a given machine should be first classified as SISD, SIMD, MIMD, and then further classified according to its constituent descendant.

According to the Hwang and Briggs's taxonomy, it is always true to predict that an

SISD-M will perform better than an SISD-S. It is, however, doubtful that such prediction can be made with respect to SIMD-Wand SIMD-B. For example, it has been indicated that using the maximum degree of potential parallelism as a performance measure, then the ILLAC-IV machine (SIMD-W) is inferior to the MPP machine (SIMD-B).

A final observation on the Hwang and Briggs's taxonomy is that shared memory systems map naturally into the MIMD-T category, while non shared memory systems map into the MIMD-L category.

3.2.4 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 11.4 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 ∨. For example, an architecture consisting of two computational subunits each having a six-tuple $(k_0 \times k'_0,\ d_0 \times d'_0,\ w_0 \times w'_0)$ and $(k_1 \times k'_1,\ d_1 \times d'_1,\ w_1 \times w'_1)$ is characterized using both subunits as $(k_0 \times k'_0,\ d_0 \times d'_0,\ w_0 \times w'_0) \times (k_1 \times k'_1,\ d_1 \times d'_1,\ w_1 \times w'_1)$, while an architecture that can be

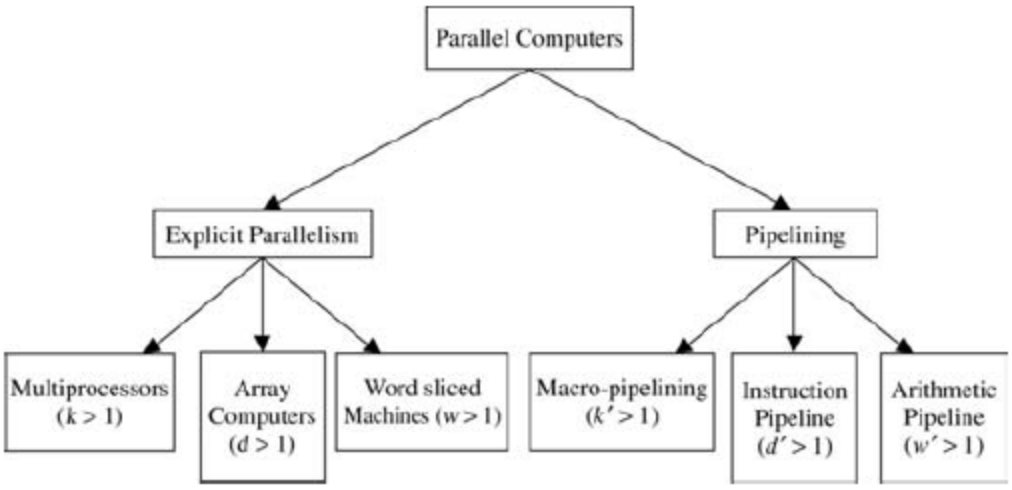


Figure 11.4 The Erlangen classification scheme

expressed using either of the two subunits is characterized as $<k_0 \times k'_0,\ d_0 \times d'_0,\ w_0 \times w'_0> \ \vee \ <k_1 \times k'_1,\ d_1 \times d'_1,\ w_1 \times w'_1>$.

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). Now, since the ILLAC-IV can also

work in a half-word mode whereby there are 128 32-bit processors rather than the 64 64-bit processors, then 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.

3.2.5 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

Neumann machine is modeled 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.

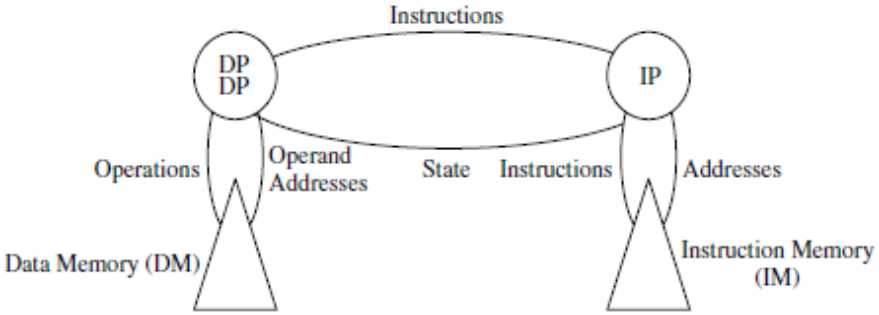


Figure 11.5 Abstract model of a simple machine

TABLE 11.4 Possible Connection Schemes

Connection type	Meaning
1–1	A connection between two single units
1–*n*	A connection between a single unit and *n* other units
n–*n*	*n*(1–1) connections
n×*n*	*n*(1–*n*) connections

Table 11.4 illustrates the different connection schemes identified by the classification.

Using the given connection schemes, Skillicorn arrived at 28 different classes. Sample classes are shown in Table 11.5. The rightmost column of the table indicates the corresponding Flynn's class. Figure 11.6 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.

TABLE 11.5 Sample Connection Classes

Class	IP	DP	IP–DP	IP–IM	DP–DM	DP–DP	Description	Flynn
1	1	1	1–1	1–1	1–1	None	Von Neumann uniprocessor	SISD
2	1	N	1–*n*	1–1	*n*–*n*	*n*×*n*	Type 1 array processors	SIMD
3	1	N	1–*n*	1–1	*n*×*n*	None	Type 2 array processors	SIMD
4	N	N	*n*–*n*	*n*–*n*	*n*–n	*n*×*n*	Loosely coupled von Neumann	MIMD
5	N	N	*n*–*n*	*n*–*n*	*n*×*n*	None	Tightly coupled von Neumann	MIMD

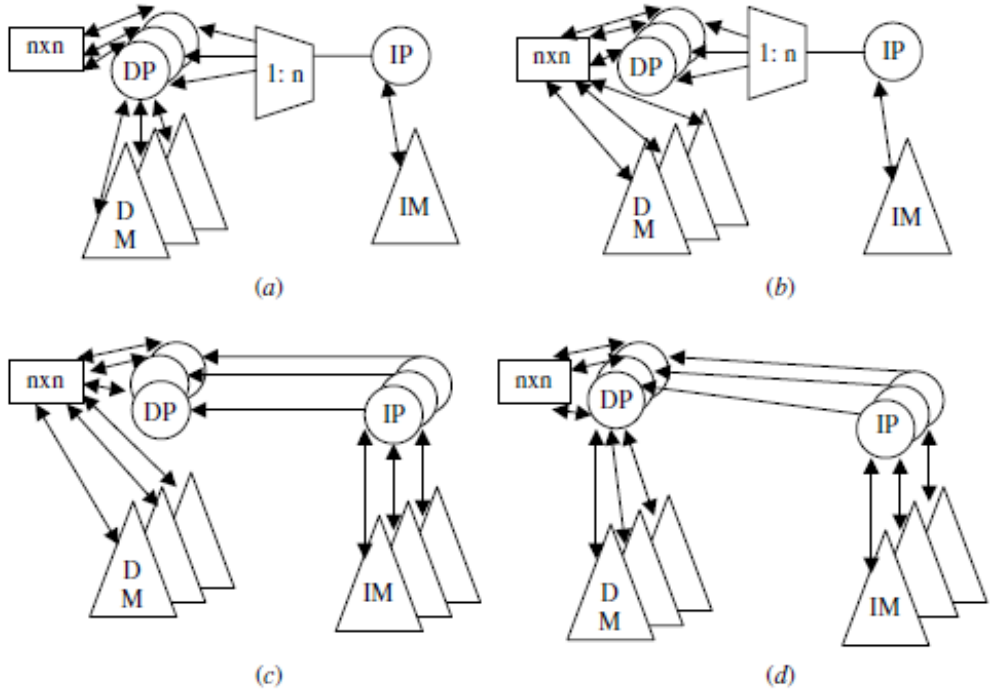


Figure 11.6 Example connection classes. (*a*) Array 1 class, (*b*) array 2 class, (*c*) tightly coupled multiprocessor, (*d*) loosely coupled multiprocessor

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 a 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.

3.3 SIMD Schemes

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 SIMD 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 11.7. 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 direct connection between a given pair of processors, then this pair can exchange data via an intermediate processor. The

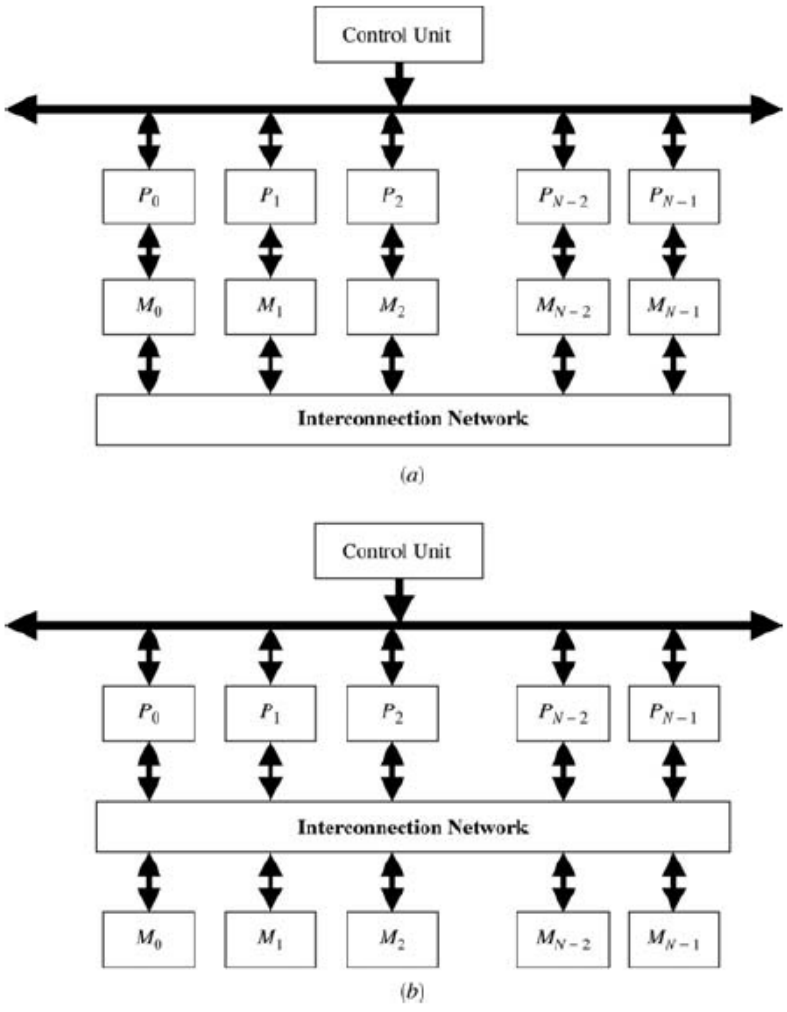


Figure 11.7 Two SIMD schemes. (*a*) SIMD scheme 1, (*b*) SIMD scheme 2

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.

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. 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 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 an 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.

3.4 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.

3.4.1 Shared Memory Organization

There has been recent growing interest in distributed shared memory systems. This is because shared memory provides an attractive conceptual model for inter-process 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.

Each processor may have registers, buffers, caches, and local memory banks as additional memory resources. A number of 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. 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.

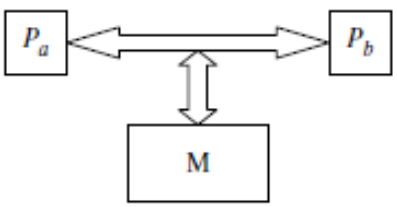


Figure 11.8 A simple shared memory scheme

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 some time 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 either held to be served next or it may be repeated some time 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 P_a and P_b (Fig. 11.9).

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. 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 behavior of this simple shared memory system. Such an example should demonstrate the basic requirements for the success of such systems.

These requirements are:

i. A mechanism for conflict resolution among rival processors

ii. A technique for specifying the sequencing constraints

iii. 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:

i. Shared memory architecture with a uniform memory access (UMA)

ii. Cache-only memory architecture (COMA)

iii. Distributed shared memory architecture with non-uniform 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 11.9 Potential shared memory problem

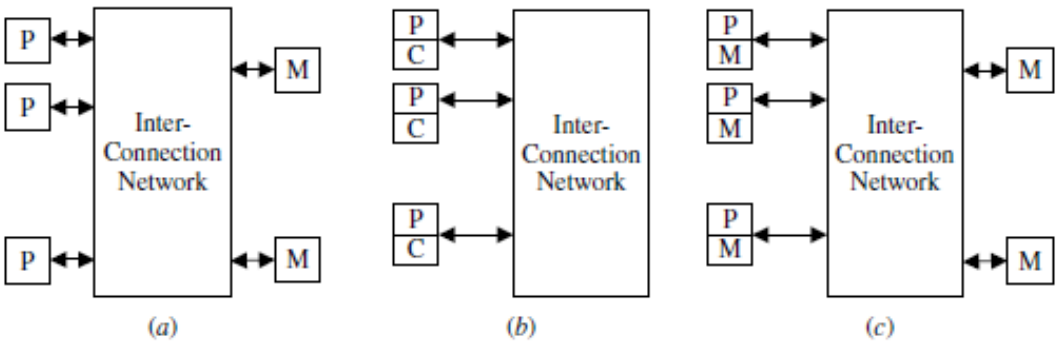


Figure 11.10 Three examples of shared-memory architectures. (*a*) UMA, (*b*) COMA, (*c*) NUMA

Figure 11.10 shows 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 bus, 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.

3.4.2 Message-Passing Organization

Message passing represents an alternative method for communication and movement of data among multiprocessors. Local, rather than global, memories are used to communicate messages among processors. A message is defined as a block of related information that travels among processors over direct links. There exist a number of models for message passing.

Examples of message-passing systems include the cosmic cube, workstation cluster, and the transputer. The introduction of the transputer system T212 in 1983 announced the birth of the first message-passing multiprocessor. Subsequently the T414 was announced in 1985, while Inmos introduced the VISI transputer processor in 1986. Two subsequent transputer products, the T800 (1988) and T9000 (1990), have been introduced.

The cosmic cube message-passing multiprocessor was designed at Caltech during the period 1981–1985. It represented the first hypercube multiprocessor system that was made to work. Wormhole routing in message passing was introduced in 1987 as an alternative to the traditional store-and forward routing in order to reduce the size of the required buffers and to decrease the message latency.

In wormhole routing, a packet is divided into smaller units that are called flits (flow control bits) such that flits move in a pipeline fashion with the header flit of the packet leading the way to the destination node.

When the header flit is blocked due to network congestion, the remaining flits are blocked as well. The elimination of the need for a large global memory, which is usually a reason for a slowdown of the overall system, together with its asynchronous nature, give message-passing schemes an edge over shared-memory schemes.

Similar to shared-memory multiprocessors, application programs are divided into smaller parts; each can be executed by an individual processor in a concurrent manner. A simple example of a message-passing multiprocessor architecture is shown in Figure 11.11. As can be seen from the figure, processors use local bus (internal channels) to communicate with their local memories while communicating with other processors via an interconnection networks (external channels).

Processes running on a given processor use internal channels to exchange messages among themselves. Processes running on different processors use external channels to exchange messages. Such a scheme offers a great deal of flexibility in accommodating a large number of processors and being readily scalable. It should be noted that the process and the processor, which executes it, are considered as two separate entities. The size of a process is determined by the programmer and can be described by its granularity, given by:

$$Granularity = \frac{\text{computation time}}{\text{communication time}}$$

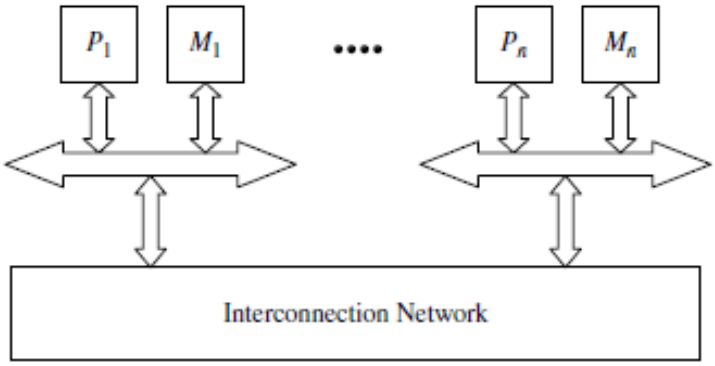


Figure 11.11 Example of a message-passing multiprocessor architecture

Three types of granularity can be distinguished. These are:

1. Coarse granularity. Each process holds a large number of sequential instructions and takes a substantial time to execute.
2. Medium granularity. Since the process communication overhead increases as the granularity decreases, medium granularity describes a middle ground whereby communication overhead is reduced in order to enable each nodal communication to take less amount of time.

3. Fine granularity. Each process contains a few numbers of sequential instructions (as few as just one instruction).

Message-passing multiprocessors use mostly medium or coarse granularity. Message-passing multiprocessors employ static networks in local communication. In particular, hypercube networks have been receiving special attention for use in a message-passing multiprocessor. The nearest neighbor two-dimensional and three-dimensional mesh networks have the potential for being used in a message-passing system as well.

Two important factors have led to the suitability of hypercube and mesh networks for use as message-passing networks. These factors are (1) the ease of VLSI implementation and (2) the suitability for two and three-dimensional applications. Two important design factors must be considered in designing such networks. These are (1) the link bandwidth and (2) the network latency.

The link bandwidth is defined as the number of bits that can be transmitted per unit time (bits/second). The network latency is defined as the time to complete a message transfer. For example, links could be unidirectional or bidirectional and they can transfer one bit or several bits at a time. To estimate the network latency, we must first determine the path setup time, which depends on the number of nodes on the path.

The actual transition time, which depends on the message size, must also be considered. The information transfer from a given source through the network can be done in two ways:

1. Circuit-switching networks. In this type of network, there is no buffer required in each node. The path between the source and destination is first determined. All links along that path are reserved. After information transfer, reserved links are released for use by other messages. Circuit-switching networks are characterized by producing the smallest amount of delay.

Inefficient link utilization is the main disadvantage of circuit-switching networks. Circuit-switching networks are, therefore, advantageously used only in the case of large message transfer.

2. Packet-switching networks. Here, messages are divided into smaller parts, called packets, before being transmitted between nodes. Each node must contain enough buffers to hold received packets before transmitting them. A complete path from source to destination may not be available at the start of transmission. As links become available, packets are moved from a node to a node until they reach the destination node.

The technique is also known as the store-and-forward packet-switching technique. Although store-and-forward packet-switching networks eliminate the need for a complete path at the start of transmission, they tend to increase the overall network latency. This is because packets are expected to be stored in node buffers waiting for the availability of outgoing links.

In order to reduce the size of the required buffers and decrease the incurred network latency, wormhole routing (see above) has been introduced. Having touched on some of the machine categories based on the Flynn’s classifications, we now provide an introduction into the interconnection networks used in these machines.

3.5 Interconnection Networks

A number of classification criteria exist for interconnection networks (INs). Among these criteria are the following.

3.5.1 Mode of Operation

According to the mode of operation, INs are classified as synchronous versus asynchronous. In synchronous mode of operation, a single global clock is used by all components in the system such that the whole system is operating in a lockstep manner. Asynchronous mode of operation, on the other hand, does not require a global clock. Handshaking signals are used instead in order to coordinate the operation of asynchronous systems.

While synchronous systems tend to be slower compared to asynchronous systems, they are race and hazard-free.

3.5.2 Control Strategy

According to the control strategy, INs can be classified as centralized versus decentralized. In centralized control systems, a single central control unit is used to oversee and control the operation of the components of the system. In decentralized control, the control function is distributed among different components in the system. The function and reliability of the central control unit can become the bottleneck in a centralized control system.

While the crossbar is a centralized system, the multistage interconnection networks are decentralized.

3.5.3 Switching Techniques

Interconnection networks can be classified according to the switching mechanism as circuit versus packet switching networks. In the circuit switching mechanism, a complete path has to be established prior to the start of communication between a source and a destination. The established path will remain in existence during the whole communication period. In a packet switching mechanism, communication between a source and destination takes place via messages that are divided into smaller entities, called packets.

On their way to the destination, packets can be sent from one node to another in a store-and-forward manner until they reach their destination. While packet switching tends to use the network resources more efficiently, compared to circuit switching, it suffers from variable packet delays.

3.5.4 Topology

According to their topology, INs are classified as static versus dynamic networks. In dynamic networks, connections among inputs and outputs are made using switching elements. Depending on the switch settings, different interconnections can be established. In static networks, direct fixed paths exist between nodes. There are no switching elements (nodes) in static networks.

Having introduced the general criteria for classification of interconnection networks, we can now introduce a possible taxonomy for INs that is based on topology. In Figure 11.12, we provide such a taxonomy. According to the shown taxonomy, INs are classified as either static

or dynamic. Static networks can be further classified according to their interconnection patterns as one-dimension (1D), two-dimension (2D), or hypercubes (HCs).

Dynamic networks, on the other hand, can be further classified according to the scheme of interconnection as bus-based versus switch-based. Bus-based INs are classified as single bus or multiple bus. Switch-based dynamic networks can be further classified according to the structure of the interconnection network as single-stage (SS), multistage (MS), or crossbar networks.

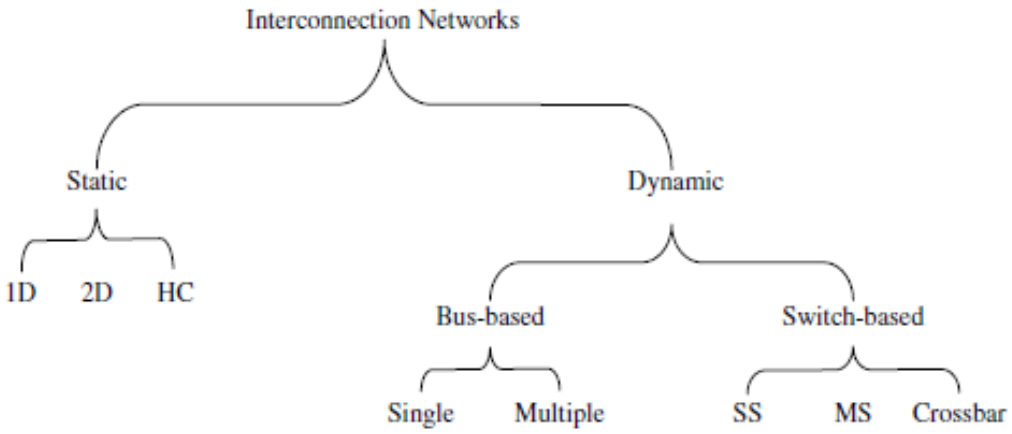


Figure 11.12 A topology-based taxonomy for interconnection networks

Self-Assessment Question

1. Interconnection Networks are classified as _____.

Self-Assessment Answer

1. Synchronous versus asynchronous.

3.6 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 into the speed-up factor defined below.

$$\begin{aligned} S(n) &= \text{speed-up factor} \\ &= \text{Increase in speed due to the use of a multiprocessor system consisting of } n \text{ processors} \\ &= \frac{\text{Execution time using a single processor}}{\text{Execution time using } n \text{ processors}} \end{aligned}$$

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

$$E(n) = \text{Efficiency}$$

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

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.

Self-Assessment Question

1. What is the formula for speed up factor?

Self-Assessment Answer

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

4.0 Conclusion

What you have learned in this unit is on introduction to multiprocessor system. Also, you have learnt about classification of computer architectures, SIMD schemes, MIMD schemes and interconnection networks. Finally, the analysis and performance metrics were also explained.

5.0 Summary

In this unit, we have navigated through a number of concepts and system configurations related to the issues of multiprocessing. In particular, we have provided the general concepts and terminology used in the context of multiprocessors. A number of taxonomies for multiprocessors have been introduced and analyzed. Two memory organization schemes have been introduced.

These are the shared-memory and message-passing systems. In addition, we have introduced the different topologies used for interconnecting multiple processors.

6.0 Tutor-Marked Assignment

1. In a message passing organization of multiprocessor system, three types of granularity can be distinguished, what are they?
2. Flynn's classification scheme is based on identifying two orthogonal streams in a computer. Explain this classification?
3. Explain some basic ideas about the performance issues in multiprocessors?

7.0 References/Further Reading

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.

Flautner, K. Kim, N.S. Martin, S. Blaauw D. and Mudge, T.N. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *International Symposium on Computer Architecture. (ISCA)*, 148–157, 2002

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.

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.

http://websrv.cs.fsu.edu/~tyson/CDA5155/refs.html

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

http://www.mmdb.ece.ucsb.edu/~ece154/lecture5.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

http://www.adaptec.com

http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_raid

http://www.ar.com