M.Sc. [Computer Science]
II - Semester
341 21

COMPUTER SYSTEM
ARCHITECTURE
Reviewers

| Dr. M. Vanitha | Assistant Professor, Department of Computer Applications, Alagappa University, Karaikudi |

Authors

Dr. Vivek Jaglan, Associate Professor, Amity University, Haryana

Pavan Raju, Member of Institute of Institutional Industrial Research (IIIR), Sonepat, Haryana

Dr. Arun Kumar, Associate Professor, School of Computing Science & Engineering, Galgotias University, Greater Noida, Distt. Gautam Budh Nagar

Dr. Kuldeep Singh Kaswan, Associate Professor, School of Computing Science & Engineering, Galgotias University, Greater Noida, Distt. Gautam Budh Nagar

Dr. Shrdhha Sagar, Associate Professor, School of Computing Science & Engineering, Galgotias University, Greater Noida, Distt. Gautam Budh Nagar

Dr. Arun Kumar, Associate Professor, School of Computing Science & Engineering, Galgotias University, Greater Noida, Distt. Gautam Budh Nagar

Dr. Kuldeep Singh Kaswan, Associate Professor, School of Computing Science & Engineering, Galgotias University, Greater Noida, Distt. Gautam Budh Nagar

Dr. Shrdhha Sagar, Associate Professor, School of Computing Science & Engineering, Galgotias University, Greater Noida, Distt. Gautam Budh Nagar

Vivek Kesari, Assistant Professor, Galgotias's GIIMT Institute of Management & Technology, Greater Noida, Distt. Gautam Budh Nagar

Units (1, 2, 3, 4, 5, 6)

Units (7, 8, 9, 10.9, 11.6, 12, 13, 14)

“Vikas®” is the registered trademark of Vikas® Publishing House Pvt. Ltd.

Vikas® PUBLISHING HOUSE PVT. LTD.
E-29, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
Website: www.vikaspublishing.com Email: helpline@vikaspublishing.com

Work Order No. AU/DDE/DE1-15/Printing of Course Materials/2020 Dated 05.02.2020 Copies-200
<table>
<thead>
<tr>
<th>Syllabi</th>
<th>Mapping in Book</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BLOCK I: Fundamentals</strong></td>
<td>Unit 1: Introduction</td>
<td>1-18</td>
</tr>
<tr>
<td>Unit 1: Introduction</td>
<td>Unit 2: Performance</td>
<td>19-34</td>
</tr>
<tr>
<td>Unit 2: Performance</td>
<td>Unit 3: Quality of Computer Design</td>
<td>35-42</td>
</tr>
<tr>
<td>Unit 3: Quality principles of computer design, performance</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BLOCK II: ILP Concepts</strong></td>
<td>Unit 4: Instruction Level Parallelism</td>
<td>43-52</td>
</tr>
<tr>
<td>Unit 4: Introduction</td>
<td>Unit 5: Scheduling</td>
<td>53-70</td>
</tr>
<tr>
<td>Unit 5: Scheduling</td>
<td>Unit 6: Limitations of ILP</td>
<td>71-78</td>
</tr>
<tr>
<td>Unit 6: Limitations of ILP: hardware and software speculation, multithreading</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BLOCK III: Thread Level Parallelism</strong></td>
<td>Unit 7: Multiprocessor and Thread Level Parallelism</td>
<td>79-94</td>
</tr>
<tr>
<td>Unit 7: Multiprocessor and thread level parallelism: Introduction, symmetric shared memory architecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit 8: Performance and architectures: performance of symmetric shared memory multiprocessors, Distributed shared memory architectures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit 9: Synchronization models: synchronization, model of memory consistency, cross cutting issues</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BLOCK IV: Memory Hierarchy Design</strong></td>
<td>Unit 10: Introduction to Memory Organization</td>
<td>109-140</td>
</tr>
<tr>
<td>Unit 10: Introduction</td>
<td>Unit 11: Protection</td>
<td>141-162</td>
</tr>
<tr>
<td>Unit 11: Protection: virtual memory and virtual machines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit 12: Issues: crosscutting issues in the design of memory hierarchies</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BLOCK V: Storage Systems</strong></td>
<td>Unit 13: Introduction to Storage Systems</td>
<td>167-180</td>
</tr>
<tr>
<td>Unit 13: Introduction</td>
<td>Unit 14: Issues</td>
<td>181-192</td>
</tr>
<tr>
<td>Unit 14: Issues: advanced topics in Disk storage, real faults and failures, I/O performance, reliability measures and benchmarks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit 15: Issues: a little queueing theory, crosscutting issues, designing and evaluating and I/O system, the Internet Archive Cluster</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
UNIT 5  SCHEDULING  53-70
5.0 Introduction
5.1 Objectives
5.2 Dynamic Scheduling
   5.2.1 Scoreboard
   5.2.2 Tomasulo Algorithm
   5.2.3 Difference Between Scoreboard and Tomasulo Algorithm
5.3 Hardware Based Speculation
5.4 Multiple Issue and Static Scheduling
5.5 Advanced Techniques for Instruction Delivery and Speculation
   5.5.1 Increasing Instruction Fetch Bandwidth
   5.5.2 Branch-target Buffers
   5.5.3 Integrated Instruction Fetch Units
   5.5.4 Value Prediction
   5.5.5 Address Aliasing Prediction
5.6 Answers to Check Your Progress Questions
5.7 Summary
5.8 Key Words
5.9 Self Assessment Questions and Exercises
5.10 Further Readings

UNIT 6  LIMITATIONS OF ILP  71-78
6.0 Introduction
6.1 Objectives
6.2 Hardware and Software Speculation
6.3 Multithreading
6.4 Answers to Check Your Progress Questions
6.5 Summary
6.6 Key words
6.7 Self Assessment Questions and exercises
6.8 Further Readings

BLOCK III: THREAD LEVEL PARALLELISM
UNIT 7  MULTIPROCESSOR AND THREAD LEVEL PARALLELISM  79-94
7.0 Introduction
7.1 Objectives
7.2 Multiprocessors
7.3 Thread Level Parallelism (Task Parallelism, Function Parallelism, Control Parallelism)
   7.3.1 Parallel Architectures
7.4 Symmetric Shared Memory Architecture
7.5 Answers to Check Your Progress Questions
7.6 Summary
7.7 Key Words
7.8 Self Assessment Questions and Exercises
7.9 Further Readings
## UNIT 8 PERFORMANCE AND ARCHITECTURES 95-102

8.0 Introduction  
8.1 Objectives  
8.2 Performance of Symmetric Shared Memory Architectures  
8.3 Distributed Shared Memory Architectures  
8.4 Answers to Check Your Progress Questions  
8.5 Summary  
8.6 Key words  
8.7 Self Assessment Questions and Exercises  
8.8 Further Readings

## UNIT 9 SYNCHRONIZATION MODELS 103-108

9.0 Introduction  
9.1 Objectives  
9.2 Synchronization  
9.3 Memory Consistency Models  
9.4 Cross Cutting Issues  
9.5 Answers to Check Your Progress Questions  
9.6 Summary  
9.7 Key Words  
9.8 Self Assessment Questions and Exercises  
9.9 Further Readings

## BLOCK IV: MEMORY HIERARCHY DESIGN

### UNIT 10 INTRODUCTION TO MEMORY ORGANIZATION 109-140

10.0 Introduction  
10.1 Objectives  
10.2 Memory Hierarchy Organization  
10.3 Main Memory  
10.3.1 Random Access Memory (RAM); 10.3.2 Read Only Memory (ROM)  
10.4 Organization of Ram and Rom Chips  
10.4.1 Memory Address Map; 10.4.2 Memory Connection to CPU  
10.5 Auxiliary Memory  
10.6 Associative Memory  
10.7 Cache Memory  
10.7.1 Basic Operation; 10.7.2 Performance  
10.8 Mapping Process  
10.8.1 Associative Mapping; 10.8.2 Direct Mapping  
10.8.3 Set Associative Mapping; 10.8.4 Writing into Cache  
10.9 Optimization of Cache Performance, Memory Technology and Optimization  
10.10 Answers to Check Your Progress Questions  
10.11 Summary  
10.12 Key Words  
10.13 Self Assessment Questions and Exercises  
10.14 Further Readings

### UNIT 11 PROTECTION 141-162

11.0 Introduction  
11.1 Objectives
INTRODUCTION

The term architecture was first used with reference to computer organization in a 1964 article describing the IBM System/360. The article defines architecture as the set of attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behaviour, as distinct from the organization of the data flow and controls, the logical design and the physical implementation.

Computer architecture is the hypothetical design and primary functional structure of a computer system. We can say that it is the blueprint of the design for the various parts of a computer, specially oriented towards the functioning of the central processing unit. In computer science and engineering, computer architecture refers to specification of the relationship between different hardware components of a computer system. It may also refer to the practical art of defining the structure and relationship of the subcomponents of a computer system, while computer organization helps to optimize performance based products. For example, software engineers need to know the processing ability of processors. They may need to optimize software in order to gain the most performance at the least expense.

This book, Computer System Architecture, follows the SIM format or the self-instructional mode wherein each unit begins with an ‘Introduction’ to the topic followed by an outline of the ‘Objectives’. The detailed content is then presented in a simple and organized manner, interspersed with ‘Check Your Progress’ questions to test the understanding of the students. A ‘Summary’ along with a list of ‘Key Words’ and a set of ‘Self Assessment Questions and Exercises’ is also provided at the end of each unit for effective recapitulation.
UNIT 1 INTRODUCTION

Structure
1.0 Introduction
1.1 Objectives
1.2 History of Computers
1.3 Computer Architecture
1.4 Answers to Check Your Progress Questions
1.5 Summary
1.6 Key Words
1.7 Self Assessment Questions and Exercises
1.8 Further Readings

1.0 INTRODUCTION

In this unit, you will be introduced to computers and their designed structure. Computers are electronic devices that perform the basic operations such as input, process, output, and storage under the direction and control of a program. Computers have become an integral part of our lives. Most of the work that is done these days is performed by computers in some way or other. Computers are used to educate students, obtain any information needed, manage finances and accounts, and for social networking.

Computer architecture is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. A few definitions of architecture describe it as defining the capabilities and programming model of a computer but not a particular implementation. In other definitions computer architecture involves instruction set architecture design, microarchitecture design, logic design, and implementation.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the history of computers
- Explain the architecture of computers
1.2 HISTORY OF COMPUTERS

History of computers stretches back to more than 2500 years to the Abacus. Abacus a simple calculator made from beads and wire. The difference between an ancient abacus and a modern computer seems vast, but the principle of making repeated calculations more quickly than the human brain is exactly the same.

Abacus was invented in the Middle East circa 500 BC it was the fastest calculator until the middle of the 17th century. Then in 1642 a French scientist and philosopher named Blaise Pascal invented the first practical mechanical calculator to help his tax collector father do his sums. Later in 1671 a German mathematician Gottfried Wilhelm Leibniz made a more advanced machine which could do much more than Pascal’s calculator. It was able to do multiplication, division and could calculate square roots. It also has memory store or register.

Apart from calculator Leibniz invented binary code which we use today also. Later in 1854 Englishman George Boole invented a new branch of mathematics called Boolean algebra. These two concepts of binary code and Boolean algebra allow today’s modern computers to do complicated tasks. But this inventions were far ahead of their time and were not used until next 50-100 years as no one could figure out where and how to use them.

Neither the abacus, nor the mechanical calculators constructed by Pascal and Leibniz really qualified as computers. A computer is a machine that can operate automatically, without any human help, by following a series of stored instructions called a program. Calculators evolved into computers when people devised ways of making entirely automatic, programmable calculators.

The first person to attempt this was Charles Babbage (1791–1871). Babbage is considered as the “father of the computer” because his machines had an input (a way of feeding in numbers), a memory (something to store these numbers while complex calculations were taking place), a processor (the number-cruncher that carried out the calculations), and an output (a printing mechanism) and the same basic components shared by all modern computers.

Babbage never completed a single one of the hugely ambitious machines that he tried to build. Babbage also received help from Augusta Ada Byron (1815–1852). She helped to refine Babbage’s ideas for making his machine programmable and that is why she is still, sometimes, referred to as the world’s first computer programmer. After his death when, by chance, his notebooks were rediscovered in the 1930’s, computer scientists finally appreciated the brilliance of his ideas.

American statistician Herman Hollerith (1860–1929) built one of the world’s first practical calculating machines, which he called a tabulator to help compile census data. It tallied the entire census in only six weeks and completed the full analysis in just two and a half years. Soon afterward, Hollerith realized his machine had other applications, so he set up the Tabulating Machine Company in 1896 to
manufacture it commercially. A few years later, he changed its name to the Computing-Tabulating-Recording (C-T-R) company and then, in 1924, acquired its present name: International Business Machines (IBM).

Claude Shannon (1916–2001), a brilliant mathematician who figured out how electrical circuits could be linked together to process binary code with Boolean algebra (a way of comparing binary numbers using logic) and thus make simple decisions. One of Bush’s contributions was an idea for a memory-storing and sharing device called Memex that would later inspire Tim Berners-Lee to invent the World Wide Web.

Later in 1936 at the age of just 23 Alan Turing wrote a paper in which he mentioned about theoretical computer today known as Turing machine (a simple information processor that works through a series of instructions, reading data, writing results, and then moving on to the next instruction). Turing ideas were highly influential on years that followed and many regard him as father of modern computing.

Also during World War II, Turing played a big role in developing of code braking machine which played a key role in victory of Britain. Today Turing is best known for his Turing Test which he invented in 1950. Turing test is used to determine ‘whether or not computer (machine) can think intelligently like human?’

Modern Computers

**First generation** (1937 – 1946) - In 1937 the first electronic digital computer was built by Dr. John V. Atanasoff and Clifford Berry. It was called the Atanasoff-Berry Computer (ABC). In 1943 an electronic computer name the Colossus was built for the military. Other developments continued until in 1946 the first general purpose digital computer, the Electronic Numerical Integrator and Computer (ENIAC) was built. It is said that this computer weighed 30 tons, and had 18,000 vacuum tubes which was used for processing. When this computer was turned on for the first time lights dim in sections of Philadelphia. Computers of this generation could only perform single task, and they had no operating system.

**Second generation** (1947 – 1962) - This generation of computers used transistors instead of vacuum tubes which were more reliable. In 1951 the first computer for commercial use was introduced to the public; the Universal Automatic Computer (UNIVAC 1). In 1953 the International Business Machine (IBM) 650 and 700 series computers made their mark in the computer world. During this generation of computers over 100 computer programming languages were developed, computers had memory and operating systems. Storage media such as tape and disk were in use also were printers for output.

**Third generation** (1963 – present) - The invention of integrated circuit brought us the third generation of computers. With this invention computers became smaller, more powerful more reliable and they are able to run many different programs at the same time. In 1980 Microsoft Disk Operating System (MS-DOS) was born.
and in 1981 IBM introduced the personal computer (PC) for home and office use. Three years later Apple gave us the Macintosh computer with its icon driven interface and the 90s gave us Windows operating system.

**Computer Developments Timeline**

- **1689**: Binary Code was invented.
- **1847**: Symbolic Logic which later helped in creation of Boolean Algebra was invented.
- **1896**: Electromechanical machine named Tabulator was invented for summarizing census data.
- **1904**: Vacuum Tubes were invented which helped in implementing logic in first generation computers.
- **1936**: Turing Machine
- **1937**: ABC created first digital computer.
- **1946**: ENIAC computer was created.
- **1947**: Transistors were invented.
- **1949**: Integrated Circuits (ICs) were developed.
- **1951**: UNIVAC computer was created.
- **1953**: Computers with memory and Operating System (OS) were developed by IBM.
- **1980**: MS-DOS (Microsoft Disk Operating System) was created.
- **1981**: IBM Personal Computers (PC) which used MS-DOS were developed.
- **1984**: Macintosh PC with graphic user interface
- **1990's**: Windows Operating System which made computers more popular

**Check Your Progress**

1. Why is Babbage considered as “father of the computer”?
2. What is the Atanasoff-Berry computer?

**1.3 COMPUTER ARCHITECTURE**

Computer architecture is the art and science of selecting and interconnecting hardware components, in order to satisfy application requirements. Computer architecture involves instruction set architecture design, microarchitecture design, logic design, and implementation.
Instruction Set Architecture (ISA)

An ISA is defined as the design of a computer from the Programmer’s Perspective. Instruction Set Architecture is a medium to permit communication between the programmer and the hardware. This basically means that an ISA describes the design of a Computer in terms of the basic operations it must support. The ISA is not concerned with the implementation specific details of a computer.

It is only concerned with the set or collection of basic operations the computer must support. For example the AMD Athlon and the Core 2 Duo processors have entirely different implementations but they support more or less the same set of basic operations as defined in the x86 Instruction Set.

ISA includes the following tasks:
1. The ISA defines the types of instructions to be supported by the processor.
2. The ISA defines the maximum length of each type of instruction.
3. The ISA defines the Instruction Format of each type of instruction. The Instruction Format determines how the entire instruction is encoded within 32 bits.

The main keywords used in the Instruction Set Architecture are as follows.

Instruction Set: Group of instructions given to execute the program and they direct the computer by manipulating the data. Instructions are in the form – Opcode (operational code) and Operand. Where, opcode is the instruction applied such as load and store data, etc. The operand is a memory register where instruction is applied.

Addressing Modes: Addressing modes are the manner in which the data is accessed. Depending upon the type of instruction applied, addressing modes are of various types such as direct mode where straight data is accessed or indirect mode where the location of the data is accessed. Processors having identical ISA may be very different in organization. Processors with identical ISA and nearly identical organization is still not nearly identical.

CPU performance is dependent upon Instruction Count, CPI (Cycles per instruction) and Clock cycle time and all three are affected by the instruction set architecture. This underlines the importance of the instruction set architecture. There are two prevalent instruction set architectures CISC (Complex Instruction Set Computing) and RISC (Reduced Instruction Set Computing).

Microarchitecture

Microarchitecture is the way a given instruction set architecture (ISA) is implemented in a particular processor.
Difference between ISA and Microarchitecture

ISA (instruction set architecture) is the set of instructions supported by a processor. Typically, a bunch of processors support the same ISA. Microarchitecture concepts deal with how the ISA is implemented. Instruction pipelining, branch prediction, out of order executions are all employed to achieve an efficient realization of the ISA.

Typically, when the term architecture is used, it implies ISA. For example, ARMv7 architecture refers to a particular ISA. On the other hand, the term core refers to an implementation of ISA using a microarchitecture. ARM cortex A-5 core is an implementation of ARMv7-A ISA using a particular microarchitecture.

Different processors may support same ISA, but may have different microarchitecture. For example, Intel and AMD both implement x86 ISA using different microarchitectures. Further, Intel itself has evolving microarchitectures. For example, it evolved through Nehalem, Sandy Bridge, Haswell and so on.

Semantic Gap

As we know, computers work on machine language in which the instructions are written in the form of binary code but it is not possible for a programmer to program in binary instruction. Hence, High level languages were developed for example, C++, Python, Java etc. This gap between the low level language and machine language is called as semantic gap.

Both RISC and CISC architectures have been developed as an attempt to cover the semantic gap and objective of improving efficiency of software development. Several powerful programming languages have come up for example C, C++, Java, etc. They provide a high level of abstraction, conciseness and power. With this evolution the semantic gap grows. To enable efficient compilation of high level language programs, CISC and RISC designs are the two options.

CISC designs involve very complex architectures, including a large number of instructions and addressing modes, whereas RISC designs involve simplified instruction set and adapt it to the real requirements of user programs.

CISC Architecture

Instruction Set Architecture is a medium to permit communication between the programmer and the hardware. Data execution part, copying of data, deleting or editing is the user commands used in the microprocessor and with this microprocessor Instruction set architecture is operated.

The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. Computers based on the CISC architecture are designed to decrease the memory cost. Because, the large programs need more storage, thus increasing the memory cost and large memory becomes more expensive. To solve these problems, the number of
instructions per program can be reduced by embedding the number of operations in a single instruction, thereby making the instructions more complex.

**Examples of CISC Processor**

**IBM 370/168**: It was introduced in the year 1970. CISC design is a 32-bit processor and four 64-bit floating point registers.

**VAX 11/780**: CISC design is a 32-bit processor and it supports numbers of addressing modes and machine instructions which is from Digital Equipment Corporation.

**Intel 80486**: It was launched in the year 1989 and it is a CISC processor which has instructions varying lengths from 1 to 11 and it will have 235 instructions.

**Characteristics of CISC Architecture**

1. Instruction-decoding logic will be Complex.
2. One instruction is required to support multiple addressing modes.
3. Less chip space is enough for general purpose registers for the instructions that are operated directly on memory.
4. Various CISC designs are set up two special registers for the stack pointer, handling interrupts, etc.
5. MUL is referred to as a “complex instruction” and requires the programmer for storing functions.

**Advantages of CISC architecture**

1. Microprogramming is easy assembly language to implement, and less expensive than hard wiring a control unit.
2. The ease of micro coding new instructions allowed designers to make CISC machines upwardly compatible:
3. As each instruction became more accomplished, fewer instructions could be used to implement a given task.

**Disadvantages of CISC architecture**

1. The performance of the machine slows down due to the amount of clock time taken by different instructions will be dissimilar
2. Only 20% of the existing instructions is used in a typical programming event, even though there are various specialized instructions in reality which are not even used frequently.
3. The conditional codes are set by the CISC instructions as a side effect of each instruction which takes time for this setting – and, as the subsequent instruction changes the condition code bits – so, the compiler has to examine the condition code bits before this happens.
RISC Architecture

RISC (Reduced Instruction Set Computing) is used in portable devices due to its power efficiency. For example, Apple iPod and Nintendo DS. RISC is a type of microprocessor architecture that uses highly-optimized set of instructions. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program. Pipelining is one of the unique feature of RISC. It is performed by overlapping the execution of several instructions in a pipeline fashion. It has a high performance advantage over CISC.

Advantages of RISC Architecture

RISC (Reduced Instruction Set Computing) architecture has a set of instructions, so high-level language compilers can produce more efficient code. It allows freedom of using the space on microprocessors because of its simplicity. Many RISC processors use the registers for passing arguments and holding the local variables. RISC functions use only a few parameters, and the RISC processors cannot use the call instructions, and therefore, use a fixed length instruction which is easy to pipeline. The speed of the operation can be maximized and the execution time can be minimized. Very less number of instructional formats, a few numbers of instructions and a few addressing modes are needed.

Disadvantages of RISC architecture

Mostly, the performance of the RISC processors depends on the programmer or compiler as the knowledge of the compiler plays a vital role while changing the CISC code to a RISC code. While rearranging the CISC code to a RISC code, termed as a code expansion, will increase the size. And, the quality of this code expansion will again depend on the compiler, and also on the machine’s instruction set. The first level cache of the RISC processors is also a disadvantage of the RISC, in which these processors have large memory caches on the chip itself. For feeding the instructions, they require very fast memory systems.

Trends in Computer Architecture

When a designer designs computer architecture it is very important to be aware of the trends in the computer industry because the architectures can be used for several decades in future so it is very important that it should be relevant in future also.
The trends which are most important while designing computer architecture are as following:

1. Trends in Technology
2. Trends in Power
3. Trends in Cost
4. Trends in Reliability

Trends in Technology

Technologies that impact architecture are as following:

1. Logic Manufacturing Technology
2. Memory (DRAM) Manufacturing Technology

1. Logic Manufacturing Technology

When we talk about logic in computer architecture it is mostly refer to transistors as they are the building blocks that we use to implement logic in computers.

A transistor is a basic electrical component that alters the flow of electrical current. Transistors are the building blocks of integrated circuits, such as computer processors, or CPUs. Modern CPUs contain millions of individual transistors that are microscopic in size.

Most transistors include three connection points, or terminals, which can connect to other transistors or electrical components. By modifying the current between the first and second terminals, the current between the second and third terminals is changed. This allows a transistor to act as a switch, which can turn a signal ON or OFF. Since computers operate in binary, and a transistor’s “ON” or “OFF” state can represent a 1 or 0, transistors are suitable for performing mathematical calculations. A series of transistors may also be used as a logic gate when performing logical operations.

Transistors in computer processors often turn signals ON or OFF. However, transistors can also change the amount of current being sent. For example, an audio amplifier may contain a series of transistors that are used to increase the signal flow. The increased signal generates an amplified sound output. Because of their low cost and high reliability, transistors have mostly replaced vacuum tubes for sound amplification purposes.

Modern transistors are so small that they cannot be seen with the naked eye. In fact, CPU transistors, such as those used in Intel’s Ivy Bridge processor, are separated by a distance of 22 nanometres. Considering one nanometre is one millionth of a millimetre that is pretty small. This microscopic size allows chip manufacturers to fit hundreds of millions of transistors into a single processor.

The rate at which transistor counts have increased generally follows Moore’s law, which observed that the transistor count doubles approximately every two
years. As of 2017, the largest transistor count in a commercially available single-chip processor is 19.2 billion—AMD's Ryzen-based Epyc.

Following graph shows how Moore's law has predicted the number of counts on a chip precisely over last few decades.
2. Memory (DRAM) Manufacturing Technology

Latency Vs Bandwidth

One of the most misunderstood concepts in networking is speed and capacity. Many people believe that speed and capacity is the same thing. When you hear someone say “My Internet speed is 30 Mbps” or something similar, what they are actually referring to is the bandwidth capacity of their Internet service, not the speed. The speed of a network is actually the result of bandwidth and latency.

The more crowded your data pipe is, the more delay you will experience while browsing the Internet.

Bandwidth refers to how wide the data pipe is, not how fast the data is transferred. The transfer rate is measured in latency. And latency means “delay.” So speed and bandwidth work together.

People who hack your WiFi just for free Internet also consume your bandwidth.

Latency is the amount of time it takes a data packet to travel from point A to point B. Together, bandwidth and latency define the speed and capacity of a network. Latency is usually expressed in milliseconds.

<table>
<thead>
<tr>
<th>Year</th>
<th>Capacity</th>
<th>Speed (Latency)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1989</td>
<td>1U16K</td>
<td>250 ns</td>
</tr>
<tr>
<td>1993</td>
<td>0.25</td>
<td>220 ns</td>
</tr>
<tr>
<td>1999</td>
<td>1M</td>
<td>190 ns</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
<td>165 ns</td>
</tr>
<tr>
<td>2002</td>
<td>16</td>
<td>145 ns</td>
</tr>
<tr>
<td>2006</td>
<td>64</td>
<td>120 ns</td>
</tr>
<tr>
<td>2020</td>
<td>256</td>
<td>100 ns</td>
</tr>
</tbody>
</table>
Introduction

NOTES

**Fig. 1.1 Relative Latency Improvement**

**Trends in Power**

Total Power = Dynamic Power + Static Power

Dynamic (switching) Power: Power consumed while charging and discharging capacitances in Integrated circuits (ICs)

Static Power: Power dissipated when circuits are not switching also called leakage power

\[
\text{Dynamic Power} = \frac{1}{2} \cdot \text{frequency} \cdot CV^2
\]

Dynamic Energy =

Where,

- \( f \) = Clock frequency
- \( C \) = Load capacitance
- \( V \) = Supply voltage

Static Power = \( I_{\text{static}} \cdot V \)
While working on a laptop, if we keep it on lap, we can feel the heat it is generating. When you drive your car, the engine converts the energy stored in the gasoline into the kinetic energy of the wheels and vehicle motion and heat. So, whenever we are utilizing energy heat is generated.

Heat is the unwanted, but inevitable, side effect of using energy to accomplish any physical task. It is not possible to convert all the input energy perfectly into the desired results without wasting some energy as heat.

Hence power is an important limit and should be considered while designing architecture.

**Trends in Cost**

Before the development of personal computers the cost factor was not of much worry as the computers were developed for some specific purpose by government or big institutes which were ready to spend huge money. Then the quantity of computers developed was very limited which was one of the reason for them being expensive. As we saw initially computers used vacuum tubes to implement logic in computers. Vacuum tubes were very fragile, due to heat generated they used to get damaged very often. Because this the operating cost of computer was too much. But with the invention of transistors we could reduce this cost to a large extent. Also transistor helped in reducing the size of computers. Later Integrated Circuits (ICs) Created huge difference by saving huge money. When personal computers were developed the cost factor become an important factor to be taken care of. Hence today when architecture of computer is designed the designer should be aware of the cost trend.
The number of dies per wafer is approximately the area of the wafer divided by the area of the die. It can be more accurately estimated by

\[ \text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}} \]

\[ \text{Dies per wafer} = \frac{\pi \times (\text{wafer diameter}/2)^2}{\text{Die Area}} - \frac{\pi \times \text{wafer diameter}}{\sqrt{2} \times \text{Die Area}} \]

**Example 1.1** Find the number of dies per 300 mm (30 cm) wafer for a die that is 1.5 cm on a side.

**Solution:** The die area is 2.25 cm². Thus

\[ \text{Dies per wafer} = \frac{\pi \times (30/2)^2}{2.25} - \frac{\pi \times 30}{\sqrt{2} \times 2.25} \]

\[ = \frac{706.9}{2.25} - \frac{94.2}{2.12} \]

\[ = 270 \]

This gives the maximum number of dies per wafer. The fraction of good dies on a wafer number, or the die yield! A simple model of integrated circuit yield, which assumes that defects are randomly distributed over the wafer and that yield is inversely proportional to the complexity of the fabrication process, leads to the following:

\[ \text{Die yield} = \text{Wafer yield} \times \left(1 + \frac{\text{Defects per unit area} \times \text{Die area}}{\alpha}\right)^{-\alpha} \]

**Example 1.2** Find the die yield for dies that are 1.5 cm on a side and 1.0 cm on a side, assuming a defect density of 0.4 per cm² and \( \alpha \) is 4.

**Solution:** The total die areas are 2.25 cm² and 1.00 cm². For the larger die, the yield is That is, less than half of all the large die are good but more than two-thirds of the small die are good.

\[ \text{Die yield} = \left(1 + \frac{0.4 \times 2.25}{4.0}\right)^{-4} = 0.44 \]
Cost vs Price

With the commoditization of the computers, the margin between the cost to manufacture a product and the price the product sells for has been shrinking. Those margins pay for a company’s research and development (R&D), marketing, sales, manufacturing equipment maintenance, building rental, cost of financing, pretax profits, and taxes. Many engineers are surprised to find that most companies spend only 4% (in the commodity PC business) to 12% (in the high-end server business) of their income on R&D, which includes all engineering.

Check Your Progress
1. Define computer architecture.
2. What are transistors?

1.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Babbage is considered as the “father of the computer” because his machines had an input (a way of feeding in numbers), a memory (something to store these numbers while complex calculations were taking place), a processor (the number-cruncher that carried out the calculations), and an output (a printing mechanism) and the same basic components shared by all modern computers.

2. The First generation computer was built in 1937 by Dr. John V. Atanasoff and Clifford Berry. It was the first electronic digital computer and was called the Atanasoff-Berry Computer (ABC).

3. Computer architecture is the art and science of selecting and interconnecting hardware components, in order to satisfy application requirements. Computer architecture involves instruction set architecture design, microarchitecture design, logic design, and implementation.

4. A transistor is a basic electrical component that alters the flow of electrical current. Transistors are the building blocks of integrated circuits, such as computer processors, or CPUs. Modern CPUs contain millions of individual transistors that are microscopic in size.
1.5 SUMMARY

- A computer is a machine that can operate automatically, without any human help, by following a series of stored instructions called a program. Calculators evolved into computers when people devised ways of making entirely automatic, programmable calculators.

- American statistician Herman Hollerith (1860–1929) built one of the world’s first practical calculating machines, which he called a tabulator to help compile census data.

- Claude Shannon (1916–2001), a brilliant mathematician who figured out how electrical circuits could be linked together to process binary code with Boolean algebra (a way of comparing binary numbers using logic) and thus make simple decisions.

- **First generation** (1937 – 1946) - In 1937 the first electronic digital computer was built by Dr. John V. Atanasoff and Clifford Berry. It was called the Atanasoff-Berry Computer (ABC).

- **Second generation** (1947 – 1962) - This generation of computers used transistors instead of vacuum tubes which were more reliable.

- **Third generation** (1963 – present) - The invention of integrated circuit brought us the third generation of computers. With this invention computers became smaller, more powerful, more reliable and they are able to run many different programs at the same time.

- An ISA is defined as the design of a computer from the Programmer’s Perspective. Instruction Set Architecture is a medium to permit communication between the programmer and the hardware.

- Both RISC and CISC architectures have been developed as an attempt to cover the semantic gap and objective of improving efficiency of software development. Several powerful programming languages have come up for example C, C++, Java, etc.

- Most transistors include three connection points, or terminals, which can connect to other transistors or electrical components. Modern transistors are so small they cannot be seen with the naked eye. In fact, CPU transistors, such as those used in Intel’s Ivy Bridge processor, are separated by a distance of 22 nanometres.

- Moore’s law observed that the transistor count doubles approximately every two years. As of 2017, the largest transistor count in a commercially available single-chip processor is 19.2 billion—AMD’s Ryzen-based Epyc.

- The speed of a network is actually the result of bandwidth and latency. Bandwidth refers to how wide the data pipe is, not how fast the data is transferred. The transfer rate is measured in latency. Latency is the amount of time it takes a data packet to travel from point A to point B.
1.6 KEY WORDS

- **Instruction Set**: Group of instructions given to execute the program and they direct the computer by manipulating the data.
- **Addressing Modes**: Addressing modes are the manner in which the data is accessed.
- **Microarchitecture**: Microarchitecture is the way a given instruction set architecture (ISA) is implemented in a particular processor.
- **Semantic Gap**: The gap between the low level language and machine language is called as semantic gap.
- **Transistor**: A transistor is a basic electrical component that alters the flow of electrical current.

1.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

**Short Answer Questions**
1. What are tasks that ISA includes?
2. What is semantic gap?
3. What is a turing machine?
4. Write down the formula for calculating the cost of integrated circuits.

**Long Answer Questions**
1. Give the difference between ISA and microarchitecture.
2. Describe CISC architecture with examples of CISC processor. Mention the advantages and disadvantages of the same.
3. What is the observation of Moore’s law in relation to the count of the transistors? Explain with the help of a graph.
4. Explain the difference between latency and bandwidth. Draw a graph to show relative latency improvement.

1.8 FURTHER READINGS


NOTES

UNIT 2  PERFORMANCE

Structure
2.0 Introduction
2.1 Objectives
2.2 Factors for Measuring Performance
2.3 Dependability and Measuring Performance
2.4 Summarizing Performances
2.5 Answers to Check Your Progress Questions
2.6 Summary
2.7 Key Words
2.8 Self Assessment Questions and Exercises
2.9 Further Readings

2.0 INTRODUCTION

In this unit, you will learn about the performance of a computer system. Performance of a computer system is the amount of useful work accomplished by it. It is estimated in terms of accuracy, efficiency and speed of executing the program instructions.

2.1 OBJECTIVES

After going through this unit, you will be able to:
- Discuss the factors used in performance measurement
- Understand how to improve performance
- Measure, report and summarize the performance

2.2 FACTORS FOR MEASURING PERFORMANCE

Following are the factors that affect the performance of a system.

1. **Response time**: The time between the start and the completion of an event is also referred to as execution time.

2. **Throughput**: The total amount of work done in a given time.

3. **CPU execution time**: It is the total time a CPU spends computing on a given task. It also excludes time for I/O or running other programs. This is also referred to as simply CPU time.

If we want know that which computer has better architecture then we compare their performances. Performance is determined by execution time as it is inversely proportional to execution time. For example, we have two different computers,
say, X and Y. The phrase “X is faster than Y” is used here to mean that the response time or execution time is lower on X than on Y for the given task. In particular, “X is n times faster than Y” will mean. This is calculated as following:

\[ n = \frac{\text{Execution time for Y}}{\text{Execution time for X}} \]

\[ \frac{\text{Performance of X}}{\text{Performance of Y}} \]

The phrase “the throughput of X is 1.3 times higher than Y” signifies here that the number of tasks completed per unit time on computer X is 1.3 times the number completed on Y.

Unfortunately, time is not always the metric quoted in comparing the performance of computers. Our position is that the only consistent and reliable measure of performance is the execution time of real programs, and that all proposed alternatives to time as the metric or to real programs as the items measured have eventually led to misleading claims or even mistakes in computer design.

Even execution time can be defined in different ways depending on what we count. The most straightforward definition of time is called wall-clock time, response time, or elapsed time, which is the latency to complete a task, including disk accesses, memory accesses, input/output activities, operating system overhead. With multiprogramming, the processor works on another program while waiting for I/O and may not necessarily minimize the elapsed time of one program. Hence, we need a term to consider this activity. CPU time recognizes this distinction and means the time the processor is computing, not including the time waiting for I/O or running other programs. (Clearly, the response time seen by the user is the elapsed time of the program, not the CPU time.)

Computer users who routinely run the same programs would be the perfect candidates to evaluate a new computer. To evaluate a new system the users would simply compare the execution time of their workloads—the mixture of programs and operating system commands that users run on a computer.

Benchmarks

A benchmark is a test that measures the performance of hardware, software, or computer. These tests can be used to help compare how well a product may do against other products. When comparing benchmarks, the higher the value of the results, the faster the component, software, or overall computer is.

Example 2.1 Machine A runs a program in 100 seconds, Machine B runs the same program in 125 seconds, then
Solution:

As we know,

\[
\frac{n}{\text{Performance of } X} = \frac{\text{Performance of } Y}{\text{Execution time for } Y} = \frac{\text{Execution time for } X}{125} = \frac{100}{100} = 1.25
\]

That means machine A is 1.25 times faster than Machine B.

And, the time to execute a given program can be computed as:

Execution time = CPU clock cycles x clock cycle time

Since clock cycle time and clock rate are reciprocals, so,

Execution time = CPU clock cycles / clock rate

The number of CPU clock cycles can be determined by,

CPU clock cycles = (No. of instructions / Program) x (Clock cycles / Instruction)

= Instruction Count x CPI

Execution time = Instruction Count x CPI x clock cycle time

= Instruction Count x CPI / clock rate

CPU Time = \(\frac{\text{Seconds}}{\text{Program}}\) = \(\frac{\text{Instructions}}{\text{Program}}\) x \(\frac{\text{Cycles}}{\text{Instruction}}\) x \(\frac{\text{Seconds}}{\text{Cycle}}\)

How to Improve Performance?

To improve performance you can either:

1. Decrease the CPI (clock cycles per instruction) by using new Hardware.
2. Decrease the clock time or Increase clock rate by reducing propagation delays or by use pipelining.
3. Decrease the number of required cycles or improve ISA or Compiler.

2.3 DEPENDABILITY AND MEASURING PERFORMANCE

Dependability is the ability of a system to deliver a specified service.

- System service is classified as proper if it is delivered as specified; otherwise it is improper.
System failure is a transition from proper to improper service.

System restoration is a transition from improper to proper service.

**Measuring Performance**

Examples of specifications of proper service are:

- $k$ out of $N$ components are functioning.
- Every working processor can communicate with every other working processor.
- Every message is delivered within $t$ milliseconds from the time it is sent.
- All messages are delivered in the same order to all working processors.
- The system does not reach an unsafe state.
- 90% of all remote procedure calls return within $x$ seconds with a correct result.
- 99.999% of all telephone calls are correctly routed.

**Dependability Measures**

Various measures of dependability are as follows:

- **Availability**: It quantifies the alternation between deliveries of proper and improper service.
- **Reliability**: It is a measure of the continuous delivery of service.
  - $R(t)$ is the probability that a system delivers proper service throughout $[0,t]$.
- **Safety**: It is a measure of the time to catastrophic failure.
  - $S(t)$ is the probability that no catastrophic failures occur during $[0,t]$.
  - Analogous to reliability, but concerned with catastrophic failures.
- **Time to Failure**: It is a measure of the time to failure from last restoration.
  (Expected value of this measure is referred to as MTTF - Mean time to failure).
- **Maintainability**: It is a measure of the time to restoration from last experienced failure. (Expected value of this measure is referred to as MTTR - Mean time to repair.)
- **Coverage**: The probability that, given a fault, the system can tolerate the fault and continue to deliver proper service.

**Amdahl's Law**

It is a formula used to find the maximum improvement possible by just improving a particular part of a system. It is often used in parallel computing to predict the theoretical speedup when using multiple processors.
Speedup is defined as the ratio of performance for the entire task using the enhancement and performance for the entire task without using the enhancement. In other words, speedup can be defined as the ratio of execution time for the entire task without using the enhancement and execution time for the entire task using the enhancement.

\[ \text{Speedup} = \frac{P_e}{P_w} \]

Amdahl’s law uses two factors to find speedup from some enhancement which are as follows:

1. **Fraction Enhanced**: The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement. For example, if 10 seconds of the execution time of a program that takes 40 seconds in total can use an enhancement, the fraction is 10/40. This obtained value is Fraction Enhanced. Fraction enhanced is always less than 1.

2. **Speedup Enhanced**: The improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program. For example, if the enhanced mode takes, say 3 seconds for a portion of the program, while it is 6 seconds in the original mode, the improvement is 6/3. This value is Speedup enhanced. Speedup Enhanced is always greater than 1.

The overall Speedup is the ratio of the execution time.

\[ \text{Overall Speedup} = \frac{\text{Old Execution Time}}{\text{New Execution Time}} \]

\[ = \frac{1}{\left(1 - \frac{\text{Fraction Enhanced}}{\text{Speedup Enhanced}} \right)} \]

**Example 2.2** Suppose that we want to enhance the processor used for web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

**Solutions:**

\[ \text{Fraction}_{\text{enhanced}} = 0.4, \quad \text{Speedup}_{\text{enhanced}} = 10 \]

\[ \text{Speedup}_{\text{overall}} = \frac{1}{0.4 + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56 \]

**Example 2.3** A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics. Suppose FP
square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQR hardware and speed up this operation by a factor of 10. The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for half of the execution time for the application. The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root. Compare these two design alternatives.

Solution: We can compare these two alternatives by comparing the speedups.

\[
\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1 - 0.2)} + \frac{0.2}{10} = \frac{1}{0.82} = 1.22
\]

\[
\text{Speedup}_{\text{FP}} = \frac{1}{(1 - 0.5)} + \frac{0.5}{1.6} = \frac{1}{0.625} = 1.23
\]

Reporting Performance

- **Response Time**: the time between the start and the completion of a task (in time units)
- **Throughput**: the total amount of tasks done in a given time period (in number of tasks per unit of time)

Check Your Progress

1. What is CPU execution time?
2. What is a Benchmark?

### 2.4 SUMMARIZING PERFORMANCES

1. The arithmetic mean of the execution times is given as:

\[
\frac{1}{n} \sum_{i=1}^{n} \text{Time}_i
\]

Where, \( \text{Time}_i \) is the execution time for the \( i \)th program of a total of \( n \) in the workload (benchmark).
2. The weighted arithmetic mean of execution times is given as:

\[
\text{Weight}_{i} \times \text{Time}_{i}
\]

Where, Weight  is the frequency of the \(i\)th program in the workload.

3. The geometric mean of execution times is given as:

\[
\text{Geometric mean} = \sqrt[\text{sample}]{\text{Product of execution time}}
\]

Example to understand concept of summarizing performances of computers

The total execution time is a consistent measure of performance and the relative execution times for the same workload can also be informative. To show an example here for the importance of summarization, suppose if you have three different computer systems, which run two different programs, P1 and P2. Assume that computer A takes 1 second for executing P1 and 1000 seconds to execute P2. Similarly, B takes 10 seconds and 100 seconds respectively and C takes 20 seconds to execute both P1 and P2.

<table>
<thead>
<tr>
<th>Program</th>
<th>Computer A</th>
<th>Computer B</th>
<th>Computer C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1 sec</td>
<td>10 sec</td>
<td>20 sec</td>
</tr>
<tr>
<td>P2</td>
<td>1000 sec</td>
<td>100 sec</td>
<td>20 sec</td>
</tr>
<tr>
<td>Total</td>
<td>1001 sec</td>
<td>110 sec</td>
<td>40 sec</td>
</tr>
</tbody>
</table>

If you look at the individual performances,
- A is 10 times faster than B for program P1
- A is 20 times faster than C for program P1
- B is 10 times faster than A for program P2
- B is 2 times faster than C for program P1
- C is 50 times faster than A for program P2
- C is 5 times faster than B for program P2

To summarize and say which computer is better, we can calculate the total execution time as 1001 seconds, 110 seconds and 40 seconds and based on these values, you can make the following conclusions:
- Using total execution time:
  - B is 9.1 times faster than A
If we want a single number to summarize performance, we can take the average of the execution times and try to come up with one value which will summarize the performance. The average execution time is the summation of all the times divided by the total number of programs. For the given example, we have

- \( \text{Avg}(A) = 500.5 \)
- \( \text{Avg}(B) = 55 \)
- \( \text{Avg}(C) = 20 \)

Here again, if we are trying to make a judgement based on this, we have a problem because we know that \( P_1 \) and \( P_2 \) are not run equal number of times. So that may mislead your summary. Therefore, we could assign weights per program. Weighted arithmetic mean summarizes performance while tracking the execution time. Weights can adjust for different running times, balancing the contribution of each program.

One more issue that has to be mentioned before we go deeper into summarization of performance is the question of the types of programs chosen for evaluation. Each person can pick his own programs and report performance. And this is obviously not correct. So, we normally look at a set of programs, a benchmark suite to evaluate performance. One of the most popular benchmarks is the SPEC (Standard Performance Evaluation Corporation) benchmark. So the evaluation of processors is done with respect to SPEC.

Even with the benchmark suite, and considering weighted arithmetic mean, the problem would be how to pick weights; since SPEC is a consortium of competing companies, each company might have their own favourite set of weights, which would make it hard to reach consensus. One approach is to use weights that make all programs execute an equal time on some reference computer, but this biases the results to the performance characteristics of the reference computer.

Rather than pick weights, we could normalize execution times to a reference computer by dividing the time on the reference computer by the time on the computer being rated, yielding a ratio proportional to performance. SPEC uses this approach, calling the ratio the SPECRatio. For example, suppose that the SPECRatio of computer A on a benchmark was 1.25 times higher than computer B.

Also observe that the execution times on the reference computer drop out and the choice of the reference computer is irrelevant when the comparisons are made as a ratio. Because a SPECRatio is a ratio rather than an absolute execution time, the mean must be computed using the geometric mean. (Since SPECRatios have no units, comparing SPECRatios arithmetically is meaningless.) The formula is,
Using the geometric mean, we can ensure two important properties:

1. The geometric mean of the ratios is the same as the ratio of the geometric means.
2. The ratio of the geometric means is equal to the geometric mean of the performance ratios, which implies that the choice of the reference computer is irrelevant.

Hence, the motivations to use the geometric mean are substantial, especially when we use performance ratios to make comparisons.

The important observations about design are:

1. **Principle of Locality:** This comes from the properties of programs. A widely held rule of thumb is that a program spends 90% of its execution time in only 10% of the code. This implies that we can predict with reasonable accuracy what instructions and data a program will use in the near future based on its accesses in the recent past. Two different types of locality have been observed. Temporal locality states that recently accessed items are likely to be accessed in the near future. Spatial locality says that items whose addresses are near one another tend to be referenced close together in time.

2. **Focus on the common case:** This is one of the most important principles of computer architecture. While making design choices among various alternatives, always favour the most frequent case. Focusing on the common case works for power as well as for resource allocation and performance. For example, when performing the addition of two numbers, there might be overflow, but obviously not so frequent. Optimize the addition operation without overflow. When overflow occurs, it might slow down the processor, but it is only a rare occurrence and you can afford it.

In applying the simple principle of focusing on the common cases, we have to decide what the frequent case is and how much performance can be improved by making that case faster. A fundamental law, called Amdahl’s Law, can be used to quantify this principle. Gene Amdahl, chief architect of IBM’s first mainframe series and founder of Amdahl Corporation and other companies found that there were some fairly stringent restrictions on how much of a speedup one could get for a given enhancement. These observations were wrapped up in Amdahl’s Law. It basically states that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used. For example, you have a floating point unit and you try to speed up the floating point unit many times, say a 10 X speedup, but all this is going to matter only if the floating point unit is going to be used very frequently. If your program does not
have floating point instructions at all, then there is no point in increasing the speed of the floating point unit.

The performance gain from improving some portion of a computer is calculated by:

\[
\text{Overall Speedup} = \frac{\text{Old Execution Time}}{\text{New Execution Time}}
\]

Amdahl’s Law gives us a quick way to find the speedup from some enhancement, which depends on two factors:

1. The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement – For example, if 30 seconds of the execution time of a program that takes 60 seconds in total can use an enhancement, the fraction is 30/60. This value, which we will call Fraction enhanced, is always less than or equal to 1

2. The improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program – This value is the time of the original mode over the time of the enhanced mode. If the enhanced mode takes, say, 3 seconds for a portion of the program, while it is 6 seconds in the original mode, the improvement is 6/3. We will call this value, which is always greater than 1, Speedup enhanced.

That is, suppose you define an enhancement that accelerates a fraction F of the execution time by a factor S and the remainder of the time is unaffected, then you can say, execution time with the enhancement is equal to 1 minus F, where F is the fraction of the execution time for which the enhancement is active, plus F by S, multiplied by the execution time without the enhancement. Speedup is going to be execution time without the enhancement divided by the execution time with enhancement. This is as shown above.

Amdahl’s Law can serve as a guide to how much an enhancement will improve performance and how to distribute resources to improve cost – performance. The goal, clearly, is to spend resources proportional to where time is spent. Amdahl’s Law is particularly useful for comparing the overall system performance of two alternatives, but it can also be applied to compare two processor design alternatives.

Consider the following example to illustrate Amdahl’s law:

For the RISC machine with the following instruction mix:
CPI = 2.2

If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement?

Fraction enhanced = \( F = 45\% \) or 0.45
Unaffected fraction = \( 1 - F = 100\% - 45\% = 55\% \) or 0.55
Factor of enhancement = \( S = \frac{5}{2} = 2.5 \)

Using Amdahl’s Law:

\[
\text{Speedup}(E) = \frac{1}{(1 - F) + \frac{F}{S}} = \frac{1}{0.55 + \frac{0.45}{2.5}} = 1.37
\]

You can also alternatively calculate using the CPU performance equation.

Old CPI = 2.2
New CPI = \( 0.5 \times 1 + 0.2 \times 2 + 0.1 \times 3 + 0.2 \times 2 = 1.6 \)

Speed up = \( \frac{2.2}{1.6} = 1.37 \)

which is the same speedup obtained from Amdahl’s Law.

The same concept can be extended even when there are multiple enhancements. Suppose three CPU performance enhancements are proposed with the following speedups and percentage of the code execution time affected:

\[
\begin{align*}
\text{Speedup}_1 & = S_1 = 10 \quad \text{Percentage}_1 = F_1 = 20\% \\
\text{Speedup}_2 & = S_2 = 15 \quad \text{Percentage}_2 = F_2 = 15\% \\
\text{Speedup}_3 & = S_3 = 30 \quad \text{Percentage}_3 = F_3 = 10\%
\end{align*}
\]

\[
\text{Speedup} = \left(\frac{1}{1 - \sum F_i} + \frac{F_i}{\sum S_i}\right)
\]

While all three enhancements are in place in the new design, each enhancement affects a different portion of the code and only one enhancement can be used at a time. What is the resulting overall speedup?

\[
\begin{align*}
\text{Speedup} & = 1/[(1 - 0.2 - 0.15 - 0.1) + 0.2/10 + 0.15/15 + 0.1/30] \\
& = 1/0.5833 = 1.71
\end{align*}
\]
Observe that:

- The performance of any system is constrained by the speed or capacity of the slowest point.

The impact of an effort to improve the performance of a program is primarily constrained by the amount of time that the program spends in parts of the program not targeted by the effort.

The last concept that we discuss in this module is about benchmarks. To evaluate the performance of a computer system, we need a standard set of programs. Individuals cannot use their own programs that will favor their design enhancements and report improvements. So benchmarks are a set of programs that form a “workload” specifically chosen to measure performance. One of the most successful attempts to create standardized benchmark application suites has been the SPEC (Standard Performance Evaluation Corporation), which had its roots in the late 1980s efforts to deliver better benchmarks for workstations. SPEC is a non-profit corporation formed to establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers. SPEC develops benchmark suites and also reviews and publishes submitted results from the member organizations and other benchmark licensees. Just as the computer industry has evolved over time, so has the need for different benchmark suites, and there are now SPEC benchmarks to cover different application classes. All the SPEC benchmark suites and their reported results are found at www.spec.org. The SPEC CPU suite is useful for processor benchmarking for both desktop systems and single-processor servers. SPEC creates standardized sets of benchmarks starting with SPEC95. The latest is SPEC CPU2006 which consists of 12 integer benchmarks (CINT2006) and 17 floating-point benchmarks (CFP2006).

The guiding principle of reporting performance measurements should be reproducibility – list everything another experimenter would need to duplicate the results. A SPEC benchmark report requires an extensive description of the computer and the compiler flags, as well as the publication of both the baseline and optimized results. In addition to hardware, software, and baseline tuning parameter descriptions, a SPEC report contains the actual performance times, shown both in tabular form and as a graph.

There are also benchmark collections for power workloads (SPECpower_ssj2008), for mail workloads (SPECmail2008), for multimedia workloads (media bench), virtualization, etc. The following programs or benchmarks are used to evaluate performance:

- Actual Target Workload: Full applications that run on the target machine.
- Real Full Program-based Benchmarks:
- Select a specific mix or suite of programs that are typical of targeted applications or workload (e.g. SPEC95, SPEC CPU2000).
– Small “Kernel” Benchmarks:
  • Key computationally-intensive pieces extracted from real programs.
    • Examples: Matrix factorization, FFT, tree search, etc.
  • Best used to test specific aspects of the machine.
  – Micro benchmarks:
    • Small, specially written programs to isolate a specific aspect of performance characteristics: Processing: integer, floating point, local memory, input/output, etc.

Each of these have their merits and demerits. So, it is always a suite of programs that is chosen so that the disadvantages of one will be outweighed by the advantages of the other.

Do we have other methods of evaluating the performance, instead of the execution time? Can we consider MIPS (Millions of Instructions per Second) as a performance measure?

For a specific program running on a specific CPU, the MIPS rating is a measure of how many millions of instructions are executed per second:

\[
\text{MIPS Rating} = \frac{\text{Instruction count}}{(\text{Execution Time} \times 10^6)}
\]

\[
= \frac{\text{Instruction count}}{(\text{CPU clocks} \times \text{Cycle time} \times 10^6)}
\]

\[
= \frac{(\text{Instruction count} \times \text{Clock rate})}{(\text{Instruction count} \times \text{CPI} \times 10^6)}
\]

\[
= \frac{\text{Clock rate}}{(\text{CPI} \times 10^6)}
\]

There are however three problems with MIPS:
1. MIPS does not account for instruction capabilities
2. MIPS can vary between programs on the same computer
3. MIPS can vary inversely with performance

Then, under what conditions can the MIPS rating be used to compare performance of different CPUs?

The MIPS rating is only valid to compare the performance of different CPUs provided that the following conditions are satisfied:
1. The same program is used (actually this applies to all performance metrics).
2. The same ISA is used.
3. The same compiler is used.

Thus, the resulting programs used to run on the CPUs and obtain the MIPS rating are identical at the machine code level including the same instruction count.

So, MIPS is not a consistent measure of performance and the CPU execution time is the only consistent measure of performance.
Lastly, we should also understand that designing for performance without considering cost and power is unrealistic:

- For supercomputing performance is the primary and dominant goal.
- Low-end personal and embedded computers are extremely cost driven and power sensitive.

The art of computer design lies not in plugging numbers in a performance equation, but in accurately determining how design alternatives will affect performance and cost and power requirements.

To summarize, we have looked at the ways and means of summarizing performance, pointed out various factors to be considered while designing computer systems, looked at Amdahl’s law, examples for quantifying the performance, the need for and the different types of benchmarks, and last of all other metrics for performance evaluation.

Check Your Progress

3. What is Amdahl’s Law?

4. List the demerits of MIPS.

2.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. CPU execution time is the total time a CPU spends computing on a given task. It also excludes time for I/O or running other programs.

2. A benchmark is a test that measures the performance of hardware, software, or computer. These tests can be used to help compare how well a product may do against other products.

3. Amdahl’s Law is a formula used to find the maximum improvement possible by just improving a particular part of a system.

4. Three problems with MIPS are:
   (i) MIPS does not account for instruction capabilities
   (ii) MIPS can vary between programs on the same computer
   (iii) MIPS can vary inversely with performance

2.6 SUMMARY

- To know which computer has better architecture we compare their performances. Performance is determined by execution time as it is inversely proportional to execution time.
Performance

NOTES

- Dependability is the ability of a system to deliver a specified service.
- System service is classified as proper if it is delivered as specified; otherwise it is improper.
- System failure is a transition from proper to improper service and system restoration is a transition from improper to proper service.
- Amdahl’s Law is a formula used to find the maximum improvement possible by just improving a particular part of a system.
- Amdahl’s law uses two factors to find speedup from some enhancement – Fraction enhanced and Speedup enhanced.
- The arithmetic mean of the execution times is given as:
  \[ \frac{1}{n} \sum_{i=1}^{n} \text{Time } i \]
- The weighted arithmetic mean of execution times is given as:
  \[ \sum_{i=1}^{n} \text{weight } i \times \text{Time } i \]
- The geometric mean of execution times is given as:
  \[ \text{Geometric mean} = \sqrt[n]{\prod_{i=1}^{n} \text{sample}_i} \]
- MIPS is not a consistent measure of performance and the CPU execution time is the only consistent measure of performance.

2.7 KEY WORDS

- **Benchmark:** It is a test that measures the performance of hardware, software, or computer.
- **Throughput:** It is a measure of how many units of information a system can process in a given amount of time.

2.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

**Short Answer Questions**

1. What is response time and throughput?
2. List the ways to improve performance?
3. Discuss the principle of locality.
4. What are small “Kernel” benchmarks?

Long Answer Questions

1. Describe the dependability measures.
2. Explain an example to understand the concept of summarizing performances of computers.
3. What are benchmarks? Explain its functioning with the help of an example.

2.9 FURTHER READINGS


Self-Instructional Material
UNIT 3 QUALITY OF COMPUTER DESIGN

3.0 INTRODUCTION
In this unit, you will learn about the quality principles of computer design and performance of a computer system. The term quality of a computer system varies from user to user based on the complexity of their task. A given system may have good performance for a simple task but may fail for a complex task.

3.1 OBJECTIVES
After going through this unit, you will be able to:

- Discuss the various parameters of quality of a computer system
- Understand and improve the system performance

3.2 QUALITY PRINCIPLES AND PERFORMANCE
Measurement of quality of a computer system varies from user to user. For a simple end user, high quality system is the one which can run its program in very less time. But for an engineer or a system specialist, high quality system will be the one which can execute complex tasks within a timeframe. In computer science terms, the quality of a computer system is defined by following parameters:

- **Response Time:** It is the time that a system takes to respond to a user task. Tasks can vary from loading web pages, disk I/O to performing complex calculations. Time taken for the system to respond to a request depends on how much time it took to perform the specific task and time task had to wait in System Queue.
Quality of Computer Design

NOTES

- **Processing Speed**: It is the number of instructions that a system can execute within a second. Processing speed is denoted by MIPS value, i.e., Millions of Instructions Per Second. Speed of a computer depends on type of instructions, order of execution and number of pipelined instructions. This parameter is very important as faster the speed, more the number of instructions the system will be able to execute. For example, an Intel i7 7500U processor has 71,120 MIPS at 2.8 GHz whereas AMD Rayzen 7 1800X has 304,510 MIPS at 3.6 GHz.

- **Disk I/O**: Disk I/O involves read and write operations. It is the speed with which transfer of data happens between Hard Disk and RAM.

- **Benchmark**: Benchmark is used to determine relative performance of a system by running a set of standardized programs or set of operations. Incurrent advanced stage of computer architectures, just by comparison of its specifications will not provide us information on how better a system is.

System Performance

Let’s assume that machine I needs 10 seconds to run a program whereas same program is executed by machine II in 15 seconds. For calculating how fast machine I is with respect to machine II, we can use below formula:

$$\frac{(\text{Execution time of Machine II} - \text{Execution time of Machine I}) \times 100}{\text{Execution time of machine I}}$$

So, it will be 50% faster in our example.

For understanding performance of a system, we should first determine number of clock cycles that system takes per instruction. This is termed as CPI (Clock Cycle per Instruction). We can obtain CPI by dividing clock cycles per program with total instruction count in the program.

If we know time required for one clock cycle, we can determine time taken by CPU to run a program i.e.,

- CPU Time = Clock Cycle Time * Number of clock cycles per program.
- CPU Performance can also be measured by using the following equation:

$$\text{Execution Time [in Seconds]} = CPI \times \frac{\text{Number of instructions to be executed}}{1/ \text{Clock frequency of system}}$$

From a design point of view, our main goal is to minimise CPU time. From above equations, we can infer that CPU time depends on:

- **Total Instruction Count**: This in turn is dependent on instruction set of computer architecture and the technology used to develop the processor.
- **Clock cycle time**: This is dependent on computer organisation used and hardware.
- **CPI**: This also depends on machine organisation and architecture.
Thus, we can conclude that even though our main aim will be to reduce CPU Time, we should also have a closer look into how modifying one parameter affects the other. For example, if we improve CPI, it may cause Clock cycle time to increase which will result in nullifying the improvement made.

Since CPI have a dependency with memory hierarchy organisation i.e., if program is running on system with cache will have better performance compared to the one that does not use cache, we should determine CPI by measuring it in the system and not only by performing calculations.

There are few problems of using MIPS rating for measuring performance.

(i) It is difficult to compare systems that have different instruction set.

(ii) MIPS value may vary for different programs run on the same system.

These drawbacks arise due to dependency of MIPS value on instruction set. As an alternative to MIPS, another performance measure was introduced i.e. Million Floating Point Operations per Second [MFLOPS]. Since it uses floating point operations to determine performance, systems having different instructions can be compared using this metric. Outside Floating-point calculation, MFLOPS is not applicable.

$$\text{MFLOPS} = \frac{\text{Number of floating-point operations}}{(\text{Execution Time} \times 10^6)}$$

**Example 3.1**

Let us assume a specific program or task takes 2 billion instructions to execute on a processor which is running at 2.5 GHz. Out of this, 40% of instructions takes 3 clock cycles to execute, 40% executes in 5 clock cycles and 20% takes 2 clock cycles. Calculate the total time taken for the program to execute.

**Solution.**

<table>
<thead>
<tr>
<th>Clock cycles</th>
<th>Percentage of instructions</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.4</td>
<td>1.2</td>
</tr>
<tr>
<td>5</td>
<td>0.4</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Total CPI = 1.2 + 2.0 + 0.4 = 3.6

Therefore, Execution time = CPI * total number of instructions * clock time

= 3.6 * 2 * 10^9 * 0.4 * 10^-9 = 2.88 seconds.

**Example 3.2**

Consider a system with 1 GHz clock. A program has 100 Million ALU instructions, 50 Million branch/jump instructions and 50 Million write instructions needs to be executed on system.
Quality of Computer Design

Given for this system, each ALU instruction takes 2 Clock cycles, each branch/jump instruction takes 1 Clock cycle, and each write instruction takes 4 Clock cycles. Calculate CPU Time needed for execution of this program.

Solution.
Number of clock cycles required for this program to execute =
\[(100 \times 2 + 50 \times 1 + 50 \times 4) \times 10^6 = 450 \times 10^6\]
Given, Clock Rate = \(1 \times 10^9\)
CPU Time = Number of clock cycles / clock rate = \(450 \times 10^6 / 10^9 = 0.45\) seconds.

Example 3.3
Assume that in a system, Program Z takes 5 seconds of CPU time and total execution time is 6 seconds. Also, it’s given performance of the system is 10 MFLOP/sec. Assuming computer is performing only CPU calculations or doing I/O instructions, but not at the same time. Let’s now replace the processor of the system with one that executes instructions 10 times faster. Calculate CPU Time, Execution Time, and MFLOP/sec for the system with new processor. Provided I/O speed does not change with new processor.

Solution.
CPU performance of System I / CPU performance of System II =
CPU Time of System I / CPU Time of System II
\[10 = 5 / \text{CPU Time of System II}\]
Therefore, CPU Time of System II = \(5 / 10 = 0.5\) seconds
Execution Time = CPU Time + I/O Time
Since I/O Time is constant for both systems,
I/O Time for system I = 6 – 5 = 1 Seconds = I/O Time for System II
Therefore, Execution Time for System II = 0.5 + 1 = 1.5 seconds
MFLOP = Number of Floating-point instructions / (Execution Time \(\times 10^6\))
For system I,
\[10 = \frac{\text{Number of Floating-point instructions}}{(6 \times 10^6)}\]
Number of floating-point instructions = \(60 \times 10^6\)
As we are executing same program Z on both systems, number of floating-point instructions will remain the same.
Therefore, for system II we have,
\[\text{MFLOP/sec} = \frac{(60 \times 10^6)}{(1.5 \times 10^6)} = 40\]
Example 3.4

Assume that you are a design engineer and is tasked with designing a new processor. Clock for this processor is 500MHz. Consider below data containing frequencies and clock cycles required for each instruction type, for a specific benchmark.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>Clock Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load and Store</td>
<td>40%</td>
<td>7</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>45%</td>
<td>5</td>
</tr>
<tr>
<td>Remaining</td>
<td>15%</td>
<td>2</td>
</tr>
</tbody>
</table>

(i) Calculate CPI for this benchmark

Solution.

Let us assume total 1000 instructions are there. Then, 400 are load and store instructions, 450 are of Arithmetic Group and 150 of them will be all remaining instructions.

Number of clock cycles = 400 * 7 + 450 * 5 + 150 * 2 = 5350

CPI = Number of clock cycles / Total number of instructions = 5350 / 1000

Therefore, CPI = 5.35 Cycles per Instruction.

(ii) Calculate MIPS for this processor.

Solution.

MIPS = Clock frequency / (CPI * 10^6) = (500 * 10^6) / (5.35 * 10^6) = 93.45

(iii) Hardware Architect informed you, if you double the number of registers, cycle time must be increased by 25%. Calculate new clock speed in MHz.

Solution.

Cycle Time = 1 / Clock Frequency = 1 / (500 * 10^6) = 2 * 10^-9

Now cycle time is increased by 25%,

Therefore, new cycle time = 2 * 10^-9 * 1.25 = 2.5 * 10^-9

Then,

New clock frequency = 1 / (2.5 * 10^-9) = 400 MHz

Improving System Performance

In computer architecture design, following are the parameters that plays an important role in defining how well the system performs.

1. Number of Cores: Each CPU consists of one or more than one processing units. Each processing unit is termed as core. Now, one core contains an ALU (Arithmetic Logical Unit), a Control Unit and a set of Internal Registers. Advantage of having multiple cores is that it helps in executing multiple programs at the same time. Just by increasing the number of cores, will not have that much impact in speed due to communication delay between all
these cores. In current scenario, we will see systems with Dual Cores, Quad Cores and Octa Cores processors.

2. **Clock Speed**: It is also known as clock rate indicates how fast a system can run programs. In all the systems, predefined clock speed is set by default by manufacturers. We can modify the clock speed in system BIOS and make the processor run faster. This is termed as Overclocking the system. Overclocking the system for long time, can result in processor getting corrupted as it will not be able to keep up with clock rate. When clock speed is increased, it will make CPU execute instructions quickly and before completion of one it will have to start executing the next instruction.

3. **Cache**: Cache memory is used for holding instructions and data temporarily while CPU executes it. Cache Memory is much closer to CPU compared to RAM. Fetching data from RAM is much slower compared to reading Cache Memory. L1, L2 and L3 are three main types of Cache. L1 is mostly inbuilt within the CPU, whereas, L2 and L3 are extra Cache Memory placed in between CPU and RAM. L2 and L3 Cache is shared between multiple CPU Cores.

4. **Architecture**: CISC [Complex Instruction Set Computing] Architecture is used mainly in Desktops and Laptops, while RISC [Reduced Instruction Set Computing] is used in Smartphones and Tablets.

Another method for improving system performance is to introduce pipelining. The idea behind this is to split the processor tasks into a series of small independent tasks. In non-pipelined system, when one instruction is being executed, the remaining will be in queue waiting for the first one to finish. For a multi core CPU, this will amount to wastage of resources, as we are not using all cores to its full potential.

But, in a pipelined system, all independent instructions work in parallel to utilize system resources to the maximum. Here, term independent instructions are very important. We can execute only those instructions in parallel, which does not have any internal dependencies to each other.

We can conclude by saying that, for a single or small number of instructions, non-pipelined system may perform better and finish the task in less time, but for programs that have millions of instructions, pipelined system outperforms non-pipelined one. If the program contains lot of branch instructions, this may impact the performance of the pipelined system. For each branch instruction, processor will not be able to predict next instruction to be executed and as a result, it must wait for branch instruction to finish.

---

**Check Your Progress**

1. Define benchmark.
2. How will you find the CPI?
3.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Benchmark is used to determine relative performance of a system by running a set of standardized programs or set of operations.
2. We can obtain CPI by dividing clock cycles per program with total instruction count in the program.

3.4 SUMMARY

- Measurement of quality of a computer system varies from user to user. For a simple end user, high quality system is the one which can run its program in very less time. But for an engineer or a system specialist, high quality system will be the one which can execute complex tasks within a timeframe.
- Response time is the time that a system takes to respond to a user task.
- Processing speed is the number of instructions that a system can execute within a second.
- Benchmark is used to determine relative performance of a system by running a set of standardized programs or set of operations.
- We can obtain CPI by dividing Clock cycles per program with Total Instruction Count in the program.
- In a pipelined system, all independent instructions work in parallel to utilize system resources to the maximum. Here, term independent instructions are very important. We can execute only those instructions in parallel, which does not have any internal dependencies to each other.

3.5 KEY WORDS

- **Response Time**: It is the time that a system takes to respond to a user task.
- **Processing Speed**: It is the number of instructions that a system can execute within a second.
- **Clock Speed**: It is also known as clock rate indicates how fast a system can run programs.
3.6 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions
1. What are the various parameters of quality of a computer system?
2. How you will measure the CPU performance?
3. What are problems of using MIPS rating for measuring performance?

Long Answer Questions
1. Explain the various parameters for improving the system performance.
2. How the pipelined system can be used to improve the performance?

3.7 FURTHER READINGS

4.0 INTRODUCTION
In this unit, you will learn about the basic concept of parallelism. Parallelism is implemented in computer systems to improve performance. Now, one method to do this is to overlap the execution of instructions. This kind of parallelism achieved is termed as Instruction Level Parallelism.

4.1 OBJECTIVES
After going through this unit, you will be able to:
- Discuss the concept and challenges of ILP
- Understand the compiler techniques for exposing ILP
- Understand how to reduce branch costs and hazards

4.2 CONCEPTS AND CHALLENGES
In order to achieve ILP, we can either implement in software side or else in the hardware portion. In the case of software dependent, ILP is exploited statically during compile time.
Value of Cycles per Instruction (CPI) for the system which has pipelining implemented is:

\[
\text{Pipeline CPI} = \text{Ideal Pipeline CPI} + \text{Structural Stalls} + \text{Data Hazard Stalls} + \text{Control Stalls}
\]

### Dependencies in ILP

Before executing ILP, we first must determine what are the dependencies that exist between each instruction? and based on that we will have to conclude which all instructions can be executed parallel.

The dependencies can either be data dependency, name dependency or control dependency. An instruction is said to be data dependent if the result of another instruction is used in the execution.

For example:

\[
\begin{align*}
B &= C + D \quad \text{Statement I} \\
A &= B + Z \quad \text{Statement II}
\end{align*}
\]

Statement II has data dependency on Statement I, as the output of Statement I is used as an input in statement II.

Name Dependency happens when the same memory location/ storage is used by two instructions. It can be of two forms:

**Anti-dependence:** In this case, the output of one instruction modifies the input of previous instruction.

For example:

\[
\begin{align*}
B &= C + D \quad \text{Statement I} \\
C &= Z + M \quad \text{Statement II}
\end{align*}
\]

Statement II and Statement I has anti-dependence, as Output of statement II is used as an input in statement I thereby accessing or modifying the same memory location.

**Output Dependence:** Two instructions or statements have output dependence, if they are writing or updating to the same memory region.

For example:

\[
\begin{align*}
B &= C + D \quad \text{Statement I} \\
B &= Z + K \quad \text{Statement II}
\end{align*}
\]
Both statements I and II, update the same memory location or name and hence has output dependence.

Another form of dependence is Control Dependencies. In this case, ordering of instructions with respect to branch and loop instructions are to be considered.

For example:
If \(<\text{condition PI}\>:
Then execute statement I

Then we can say that statement I is control dependent with respect to control statement PI.

### 4.3 COMPILER TECHNIQUES

For performing ILP, we need to remove dependencies from instructions. The compiler will schedule a dependent instruction from the source instruction minimum distance of clock cycles apart.

We will rely on an example for better understanding:

```plaintext
for (i=1000; i>0; i=i–1)
x[i] = x[i] + s;
```

This loop is parallel by noticing that the body of each iteration is independent. The first step is to translate the above segment to MIPS assembly language. In the following code segment, R1 is initially the address of the element in the array with the highest address, and F2 contains the scalar value, s. Register R2 is precomputed so that 8(R2) is the last element to operate on. The straightforward MIPS code, not scheduled for the pipeline, looks like this:

```plaintext
Loop:          L.D  F0, 0(R1)      ; F0=array element
ADD.D       F4, F0, F2           ; add scalar in F2
S.D              F4,0(R1)              ; store result
DADDUI    R1, R1, #-8            ; decrement pointer
; 8 bytes (per DW)
BNE             R1, R2, Loop         ; branch R1!=zero
```

Let’s start by seeing how well this loop will run when it is scheduled on a simple pipeline for MIPS with the latencies.
This code requires 10 clock cycles per iteration. We can schedule the loop to obtain only one stall:

```
Loop:  L.D    F0,0(R1)
       DADDUI      R1, R1, #-8
       ADD.D         F4, F0, F2
       stall
       BNE             R1, R2, Loop         ; delayed branch
       S.D               F4, 8(R1)     ; altered & interchanged with DADDUI
```

Execution time has been reduced from 10 clock cycles to 6. The stall after ADD.D is for use by S.D.

In the above example, we complete one loop iteration and store back one array element every 6 clock cycles, but the actual work of operating on the array element takes just 3 (the load, add, and store) of those 6 clock cycles. The remaining 3 clock cycles consist of loop overhead—the DAD, DUI and BNE and a stall. To eliminate these 3 clock cycles, we need to get more operations within the loop relative to the number of overhead instructions.

A simple scheme for increasing the number of instructions relative to the branch and overhead instructions is loop unrolling. Unrolling simply replicates the loop body multiple times, adjusting the loop termination code.

Loop unrolling can also be used to improve scheduling. Because it eliminates the branch, it allows instructions from different iterations to be scheduled together. In this case, we can eliminate the data use stall by creating additional independent instructions within the loop body. If we simply replicated the instructions when we unrolled the loop, the resulting use of the same registers could prevent us from effectively scheduling the loop. Thus, we will want to use different registers for each iteration, increasing the required register count.
In real programs, we do not usually know the upper bound on the loop. Suppose it is n, and we would like to unroll the loop to make k copies of the body. Instead of a single unrolled loop, we generate a pair of consecutive loops. The first executes \((n \mod k)\) times and has a body that is the original loop. The second is the unrolled body surrounded by an outer loop that iterates \((n/k)\) times. For large values of n, most of the execution time will be spent in the unrolled loop body.

### 4.4 REDUCING BRANCH COSTS

To make the pipeline working effective and efficient, we should reduce the control dependencies which occur in the branches. We can reduce the branch hazards in two ways.

1. Loop Unrolling
2. Behaviour prediction

Loop unrolling feature depends on the branch prediction. To reorder code for the branches to run fast, we require to compile the program statically. There are many processes that can be used to predict the static branch behaviour.

The activity of the branches can be predicted using both static and dynamic method by the hardware during the time of execution. The static predictors are used in processors where the time taken for compiling is not possible to predict. Their predictors can be used to help dynamic predictors.

#### Static Branch Prediction

This process has the mis-prediction rate taken up by the branch which is 34%. To acquire more result, the branches have to collect information from the earlier runs. The key element is to observe the branches which are often bi-modally distributed. Changing the input for a different run leads to small change in accuracy. The effectiveness of any branch depends on both frequency and the accuracy if the conditional branches.

#### Dynamic Branch Prediction

The simplest prediction is done by dynamic branch prediction buffer where a small memory is numbered by the lower portion of the address. The memory contains a bit about the usage of the branch. It tells us whether the branch was used recently or not. This process sorts the buffer which is useful to reduce the branch delay.

The branch prediction can be of two ways.

1. 1bit prediction scheme
2. 2bit prediction scheme
The simple 1-bit prediction scheme has problems related to performance. Even if the branch is taken completely the prediction will go incorrect twice since the misprediction causes the bit to be flipped. The 2-bit prediction schemes are often used.

### Branch Direction Prediction

Prediction of branch direction plays a vital role in the process. So to know the direction of the branch, we can use static prediction system is used where the compilers decide the direction.

Then the dynamic prediction decides the direction using dynamic information which is of different types as mentioned above.

1. 1-bit branch prediction
2. 2-bit branch prediction
3. Correlating branch prediction
4. Tournament branch

#### 1-bit branch prediction

![1-bit branch prediction diagram]

- **Predict:**
  - **not taken:** predict "do not take branch" state
  - **taken:** predict "take branch" state

#### 2-bit branch prediction

![2-bit branch prediction diagram]
4.5 DATA HAZARDS

Data hazards occur when the instructions are showing data dependencies in different stages of a pipeline. There is a different situation in data hazard that can occur. These are also called race hazards.

1. Read after write (true dependency)

Correlating branch prediction

Tournament branch

2-bit global branch history
Instruction Level Parallelism

NOTES

2. Write after read (anti-dependency)
3. Write after write (output dependency)

Let us consider two instructions p and q where p occurs before q.

- Read After Write (RAW)
  Here, q reads the code before p is ready to write it. Here the RAW gives a stage where the instruction referring to the result is not yet calculated. This occurs because the instruction is executed after a prior instruction.

- Write After Read (WAR)
  Q tries to write a destination before p reads it. WAR represents a problem of concurrent execution.

- Write After Write (WAW)
  Q tries to write an operand before p writes it. WAW may occur in a concurrent execution environment.

Check Your Progress
1. What are the different types of dependencies in ILP?
2. How you will define the output dependence between two instructions?
3. What are the two ways to reduce the branch hazards?

4.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The dependencies can either be data dependency, name dependency or control dependency.
2. Two instructions or statements have output dependence, if they are writing or updating to the same memory region.
3. We can reduce the branch hazards in two ways.
   (i) Loop Unrolling
   (ii) Behaviour prediction

4.7 SUMMARY

- In order to achieve ILP, we can either implement in software side or else in the hardware portion. In the case of software dependent, ILP is exploited statically during compile time.
The dependencies can either be data dependency, name dependency or control dependency. An instruction is said to be data dependent if the result of another instruction is used in the execution.

Name Dependency happens when the same memory location/storage is used by two instructions.

Two instructions or statements have output dependence, if they are writing or updating to the same memory region.

Loop unrolling can also be used to improve scheduling. Because it eliminates the branch, it allows instructions from different iterations to be scheduled together.

To make the pipeline working effective and efficient, we should reduce the control dependencies which occur in the branches.

The static predictors are used in processors where the time taken for compiling is not possible to predict. Their predictors can be used to help dynamic predictors.

The simplest prediction is done by dynamic branch prediction buffer where a small memory is numbered by the lower portion of the address.

Data hazards occur when the instructions are showing data dependencies in different stages of a pipeline.

**4.8 KEY WORDS**

- **Name Dependency**: It happens when the same memory location/storage is used by two instructions.
- **Output Dependence**: Two instructions or statements have output dependence, if they are writing or updating to the same memory region.

**4.9 SELF ASSESSMENT QUESTIONS AND EXERCISES**

**Short Answer Questions**

1. How will you reduce the branch hazards?
2. What are the different types of branch direction prediction?

**Long Answer Questions**

1. What are the different types of dependencies in ILP?
2. Explain loop rolling and unrolling in compiler techniques for exposing ILP.
3. What are the different types of hazards?
4.10 FURTHER READINGS


UNIT 5  SCHEDULING

Structure
5.0 Introduction
5.1 Objectives
5.2 Dynamic Scheduling
  5.2.1 Scoreboard
  5.2.2 Tomasulo Algorithm
  5.2.3 Difference Between Scoreboard and Tomasulo Algorithm
5.3 Hardware Based Speculation
5.4 Multiple Issue and Static Scheduling
5.5 Advanced Techniques for Instruction Delivery and Speculation
  5.5.1 Increasing Instruction Fetch Bandwidth
  5.5.2 Branch-target Buffers
  5.5.3 Integrated Instruction Fetch Units
  5.5.4 Value Prediction
  5.5.5 Address Aliasing Prediction
5.6 Answers to Check Your Progress Questions
5.7 Summary
5.8 Key Words
5.9 Self Assessment Questions and Exercises
5.10 Further Readings

5.0 INTRODUCTION

The applications of microprocessors in today’s world are truly unlimited. One of the main reasons for such demand for microprocessors is due to the miniaturization of things. As the number of transistors that are incorporated on a chip is becoming smaller, they require less power to operate and produce less heat. And also, smaller the distance will also allow us to work at faster clock speeds. In olden microprocessors like Intel 4004, we have merely 2300 transistors. But today’s microprocessor holds millions of transistors incorporated in them. As the number of transistors increases in a chip the complexity of the microprocessor also scales. We need to maintain the performance of the microprocessor as we develop newer or improved features of processors. One of such improvements is dynamic scheduling.

5.1 OBJECTIVES

After going through this unit, you will be able to:
- Explain the concept of dynamic scheduling
- Understand the important techniques used in dynamic scheduling
5.2 DYNAMIC SCHEDULING

Dynamic scheduling is a method in which the hardware determines which instruction need to be executed. In dynamic scheduling, the instructions don’t execute in the order they arrive, but rather execute on the availability of source operands. It is very much useful to improve performance by holding an advantage of parallelism.

In simple pipelining techniques, they use in-order instructions to execute the programs. This results in each instruction depend on the previous instruction and the performance will be reduced. As the functional unit’s increases, we cannot consider this technique. Let us consider an example and understand simple pipelining techniques.

\[
\begin{align*}
\text{ADD.D} & : \ F1, F2, F3 \\
\text{SUB.D} & : \ F0, F4, F15 \\
\text{DIV.D} & : \ F10, F15, F8
\end{align*}
\]

In the instruction given above, we can see that DIV.D instruction cannot be executed until the above instruction SUB.D is executed. There is a dependency of F15 in DIV.D with the F15 in SUB.D instruction. This creates a performance limitation that we need to be eliminated by not following the instruction we execute in program order.

In dynamic scheduling, the hardware rearranges the instruction execution order to reduce the stalls while maintaining data flow and exception behavior. This makes the job simple for the compiler and allow code compiled with one pipeline run on a different pipeline. It implies Out-of-order execution and Out-of-order completion.

Two important techniques used in dynamic scheduling are as follows:

1. Scoreboarding
2. Tomasulo Algorithm

5.2.1 Scoreboard

The main idea of Score boarding is to keep track of the status of instructions, functional units, and registers. It makes sure that the instructions will execute in out-of-order with no conflicts and hardware is available for each instruction.
The scoreboard is divided into 4 major stages:

1. Issue
2. Read Operands
3. Execution
4. Write a result

**Issue:** It decodes the instructions and checks for any structural hazards.

Wait conditions:
(i) The required functional units are free.
(ii) No other instruction writes to the same register destination.

Actions:
(i) The instruction proceeds to the next functional units.
(ii) Updating the internal data structure of the scoreboard.

**Read Operands:** In this stage, it will wait until no data hazards, then read the operands

Wait conditions:
All source operands are available.

Actions:
The functional units read register operands and start executing the next coming cycle.

**Execution:** Operate on operands
Scheduling

Notes

Actions:
The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed execution.

Write Result: Finish execution

Wait condition:
No other instruction/Functional unit is going to read the register destination of the Instruction.

Actions:
Write the register and update the scoreboard.

The scoreboard consists of 3 parts which are as follows:
1. Instruction status
2. Functional unit status
3. Register status

Instruction Status: It keeps track of the steps that have been completed the execution of the instruction.

Functional Unit Status

It indicates the status of each functional unit as instructions are issued. The different status of functional units are as follows:
- Busy: Indicates whether the unit is being used or not
- Op: Operation to perform in the unit (e.g. MUL, DIV or MOD)
- F: a Destination register
- F, F: Source-register numbers
- Q, Q: Functional units that will produce the source registers F, F
- R, R: Flags that indicate when F, F are ready for and are not yet read.

Register Status: Indicates, for each register, which functional unit will write results into it.

5.2.2 Tomasulo Algorithm

The second important technique used in dynamic scheduling is the Tomasulo algorithm. It was invented by Robert Tomasulo and was first implemented in the IBM 360/91 Floating-point Unit which came out to practice after three years of CDC 6600. This is a hardware algorithm which executes the instructions in out-of-order execution and also, enables more efficient use of multiple execution units.

Tomasulo Algorithm uses register renaming to eliminate output and anti-dependencies. Output and anti-dependencies are just name dependences only. There will be no actual data dependencies in them.
There are four major concepts that are necessary to implement the Tomasulo algorithm which are as follows:

1. Common data bus
2. Instruction order
3. Register renaming
4. Execution

1. **Common Data Bus**
   The Common Data Bus (CDB) connects the reservation stations directly to the functional units. Reservation stations are buffers which immediately store all the instruction operands as soon as they are available and later will be used to fetch these instruction operands whenever required. According to Tomasulo it "preserves precedence while encouraging concurrency".

2. **Instruction Order**
   This makes sure that the exceptions raised by the instructions occur in the same order regardless of the sequence of the instructions that are begin executed out-of-order.

3. **Register Renaming**
   Register renaming is performed in order to maintain the out-of-order execution.
For example:

\[
\begin{align*}
\text{MULD } & F2, F4, F4 \\
\text{ADDD } & F4, F0, F6
\end{align*}
\]

We can clearly see the anti-dependency of F4 in first and second instructions.

Let's assume that, there is an unused register F1, Now replace the F4 in second instruction with F1.

\[
\begin{align*}
\text{MULD } & F2, F4, F4 \\
\text{ADDD } & F1, F0, F6
\end{align*}
\]

Now we have removed the anti-dependency between instructions without affecting the flow of instructions. This is called register renaming. By doing this process, we can easily eliminate dependencies and can achieve high parallelism.

4. **Execution**

Exceptions are used to understand the status information of the instructions that are being executed. We can determine which operation is needed to be performed with that particular exception raised.

There are three stages of execution for the Tomasulo algorithm which are discussed below:

1. **Issue:** It is the first stage of the algorithm where the instructions are fetched from the head of the instruction queue, which is maintained in First-In-First-Out order to maintain proper dataflow of execution. These fetched instructions are issued for the execution once all the instruction operands and reservation stations are ready or else it will wait until all the reservation stations are ready with the instruction operands. Register renaming takes place in this stage to eliminate WAR and WAW hazards. This stage is sometimes called dispatch in a dynamically scheduled processor.

2. **Execute:** In this stage, the operations are performed on the instructions arrived. It monitors the data bus for the operands and operations are performed when all the operands are available or the instructions are delayed until all operands are available to eliminate RAW hazards. When more than one instructions are available in the same clock cycle then independent functional units can start its execution within the same clock cycle. These functional units will decide among them to choose the instruction to be shared.

3. **Write:** In this stage, all the operation results are written on to the common data bus and from CDB it is further written to the registers and these operations are again written back to the memory.

**Example 5.1** Show how the information tables look like for the following code sequence when only the first load has completed and writes its result.

\[
\begin{align*}
\text{LD } & F6,32(R2) \\
\text{LD } & F2,44(R3)
\end{align*}
\]
Scheduling

NOTES

Self-Instructional Material

- MUL.D F0,F2,F4
- SUB.D F8,F2,F6
- DIVD F10,F0,F6
- ADD.D F6,F8,F2

Solution:

The figure below shows the result in three tables. The numbers appended to the names add, mult, and load stand for the tag for that reservation station. Addl is the tag for the result from the first add unit. In addition, we have included an instruction status table. This table is included only to help you understand the algorithm. It is not actually a part of the hardware. Instead, the reservation station keeps the state of each operation that has issued.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Issue</td>
</tr>
<tr>
<td>LD F6,32(R2)</td>
<td>V</td>
</tr>
<tr>
<td>LD F2,44(R3)</td>
<td>V</td>
</tr>
<tr>
<td>MLD F0,F3,F4</td>
<td>V</td>
</tr>
<tr>
<td>MULD F8,F2,F6</td>
<td>V</td>
</tr>
<tr>
<td>DIVD F10,F0,F6</td>
<td>V</td>
</tr>
<tr>
<td>ADDD F6,F8,F2</td>
<td>V</td>
</tr>
</tbody>
</table>

Reservation stations and Register tags are shown once all the instructions have issues, but only first load instruction has completed and writes its result to the common data bus. The second load has completed effective address calculation, but it is waiting on the memory unit. We use the array Regs[] to refer to the register file and the array Mem[] to refer to the memory. Remember that an operand is specified by either a Q field or a V field at any time. Notice that the ADD.D instruction, which has a WAR hazard at the WB stage, has issued and could complete before the DIV.D initiates.
5.2.3 Difference Between Scoreboard and Tomasulo Algorithm

<table>
<thead>
<tr>
<th></th>
<th>Scoreboard</th>
<th>Tomasulo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>When the functional unit is free</td>
<td>Issues when the reservation station is free</td>
</tr>
<tr>
<td>Read Operations</td>
<td>From Register file</td>
<td>From register file and common data bus</td>
</tr>
<tr>
<td>Write Operations</td>
<td>To register file</td>
<td>To common data bus</td>
</tr>
<tr>
<td>Structural hazards</td>
<td>Functional units</td>
<td>Reservation units</td>
</tr>
<tr>
<td>WAW,WAR hazards</td>
<td>Problem</td>
<td>No problem</td>
</tr>
<tr>
<td>Register renaming</td>
<td>Not available</td>
<td>Available</td>
</tr>
<tr>
<td>Instruction completing</td>
<td>No limit</td>
<td>1 cycle (per CDB)</td>
</tr>
<tr>
<td>Instruction beginning ex</td>
<td>1 (per set of read ports)</td>
<td>No limit</td>
</tr>
</tbody>
</table>

Check Your Progress

1. What are the two techniques used in dynamic scheduling?
2. What is the main idea of score boarding?

5.3 HARDWARE BASED SPECULATION

Speculation (also known as speculative loading) is the optimization technique used to maintain a high rate of instruction execution using predictions based on the structure of the program. This process is implemented in Explicitly Parallel Instruction Computing (EPIC) processors to maintain high parallelism to reduce the processor memory exchange latency or bottlenecks. One of the type in speculation is Hardware based speculation.

Hardware-based speculation is based on three major key concepts which are as follows:

1. **Dynamic branch prediction**: It helps us to choose which instruction is to execute first.
2. **Speculation**: It allows us to execute the instructions before the control dependencies are resolved
3. **Dynamic scheduling**: This helps to deal with the scheduling of different combinations of basic blocks with different techniques as discussed earlier.
The most common implementation of hardware-based speculation is based on the modification of the Tomasulo algorithm. Any hardware that implements Tomasulo algorithm can be extended to support speculation. Hardware-based speculation follows the predictive flow of data values to choose the sequence of execution of instructions. This method of executing programs is essentially a data flow execution. Operations execute as soon as their instruction operands are available. The background idea of the speculation is to separate when an instruction writes its results when those results are committed to processor state. Instruction may or may not execute in order (i.e., out-of-order), but they must commit in order. Once the instructions are executed the results are stored in the results buffer and then commit.

Hardware-based speculation executes the instruction in four stages.

1. **Issue**: Instructions are passed to the available reservation stations from the instruction queue if there is an empty reservation station and also empty reorder buffer (ROB). The instruction operands are stored in reservation stations. Once they are passed entry is made into the reorder buffer to ensure the correct order of commitment of results.

2. **Execute**: Once every instruction operands are available at the reservation station, the functional units start execution.

3. **Write results**: After the completion of execution, the results are written on to the common data bus and from common data bus into the reservation stations and reorder buffer as soon as the CDB is available. Once they are done reservation station is cleared.

4. **Commit**: This is the final stage of completing the cycle of instruction execution. When the instruction reaches the head of the reorder buffer with the value of the results. These results values are written back to the register file or memory and cleared from the reorder buffer, thus completing the commit of instruction to the processor state. In some processors, we call this stage “completion” or “graduation”.

The key idea in implementing speculation is to allow instruction to commit in order irrespective of its flow of the order of execution. Adding this commit phase to the process of the instruction execution sequence required an additional set of hardware buffers that holds the end results of instructions just before committing. We call this buffer as reorder buffer (ROB). ROB is used to pass the results after every instruction that may be speculated. Just like reservation stations even reorder buffer (ROB) provides some extra register to extend the register set. It holds the results of every instruction that are executed after the writing stage and before commit stage. Tomasulo algorithm along with the reorder buffer (ROB) is shown in the Figure 5.3.
Advantages of hardware-based speculation

1. Memory disambiguation is better.
2. Gives better branch prediction over software approach
3. Precise exception model
4. Works on old version software too.

Disadvantages of hardware-based speculation

1. Cost of hardware is high.
2. Complexity is also high.

5.4 MULTIPLE ISSUE AND STATIC SCHEDULING

In computer architecture, we frequently come across CPI which stands for Cycles per Instruction. We define the processor’s performance based on the average number of clock cycles per instruction. When the processor takes more than one clock cycle to execute the instruction, then that processor is said to be sub-scalar. When CPI = 1, we call that the processor is said to be scalar. If we can attain CPI < 1 then, in that case, we say that the processor is superscalar. With single execution unit processor, we can attain an ideal CPI value of 1. But if we want to achieve CPI < 1, we would use multiple execution unit processors. We can attain CPI<1 with pipelining by exploiting instruction level parallelism. To attain such values of
CPI, we generally use three approaches to multiple issue processors (superscalar processors):

1. **Statically scheduled superscalar processors**
   In these processors, we issue instructions in issued packets with the number of instructions ranging from 0 to 8 instructions. For example, if we consider we have a 4-issue superscalar processor, then we can say that this processor can take up to a maximum of 4 instructions from the fetch unit during the instruction cycle. It is not mandatory to fetch 4 instructions for every cycle from the fetch unit. In static superscalar processor, we fetch the instructions in in-order execution.
   - Intel added MMX instructions to Pentium in 1997 for this purpose and also added further instruction sets called Streaming SIMD Extensions (SSE) to subsequent processors.
   - Modern graphics processing units on computers offer high-performance SIMD for the applications besides graphics.

2. **Very Long Instruction Word (VLIW) Processors**
   These processors have a fixed number of instructions formatted as one large instruction or as a fixed instruction packet size with parallelism among instructions which are indicated explicitly by the instructions. These processors are the next step when compared with the usual RISC architecture based processors. The main advantages of VLIW processors are that they reduce the main complexity that occurs due to hardware by moving that complexity to the software. This makes the processors cheaper, smaller in size by which we can reduce the power consumptions of that is required to operate. Software compilers are developed in such a way that they generate the program which can take advantage of achieve high parallelism and scheduling multiple operations per clock cycle.
   - Intel and HP developed IA-64 architecture based on this type.
   - AMD introduced 64-bit extensions of x86 processors.

3. **Dynamically scheduled superscalar processors**
   As we have discussed dynamic scheduling in our previous topics that we use dynamic scheduling to improve the performance of the processor. When we use dynamic scheduling for these superscalar processors we not only improve the performance, it also allows the processor to potentially overcome the issue restrictions. These processors require multiple arithmetic units and register files with additional ports to avoid structural hazards.
5.5 ADVANCED TECHNIQUES FOR INSTRUCTION DELIVERY AND SPECULATION

In order to achieve high Instruction level parallelism, we should be able to deliver a high bandwidth instruction stream along with good predicting branches. We need to look at methods which can increase the instruction delivery bandwidth and then turn them to a set of key issues in implementing advanced speculation techniques like register renaming, reorder buffers, aggressiveness of speculation and value prediction.

5.5.1 Increasing Instruction Fetch Bandwidth

A multiple issue processor will require that the average number of instructions fetched every clock cycle be at least as large as the average throughput. Fetching these instructions requires wide enough paths to the instruction cache, but the most difficult aspect is handling branches.

5.5.2 Branch-target Buffers

In order to reduce the branch penalty for five-stage pipeline and deeper pipelines we must know whether the as-yet-undecoded instruction is a branch and, if so, what the next PC should be. If the instruction is a branch and we know what the next PC should be, then we can have a branch penalty of zero. A branch-prediction cache that stores the predicted address for the next incoming instruction after a branch is called a branch-target buffer or branch-target cache.
The PC of the instruction being fetched is matched against a set of instruction addresses stored in the first column. These represent the addresses of known branches. If the PC matches one of these entities, then the instruction being fetched is a taken branch, and the second field predicted PC contains the prediction for the next PC after the branch. Fetching begins immediately at that particular address. The third field, which is optional, may be used for extra prediction state bits.

Figure 5.6 illustrates the detailed steps when using a branch-target buffer for a simple five-stage pipeline.

The table below shows the penalties for all possible combinations of whether the branch is in the buffer and what it actually does, assuming we store only taken branches in the buffer.
5.5.3 Integrated Instruction Fetch Units

Integrated Instruction Fetch Units are introduced to reduce the complexities of multiple issue. Essentially, this amounts to recognizing that characterizing instruction fetch as a simple single pipe stage provided that the complexities of multiple issues is no longer valid. Some of the key functions involved in Integrated Instruction Fetch Units are as follows:

1. **Integrated Branch Prediction**: The branch predictor becomes part of the instruction fetch unit and is constantly predicting branches, so as to drive the fetch pipeline.

2. **Instruction Prefetch**: To deliver multiple instructions per clock, the instruction fetch unit will likely need to fetch ahead. The unit autonomously manages the prefetching of instructions, integrating it with branch prediction.

3. **Instruction Memory Access and Buffering**: When fetching multiple instructions per cycle, a variety of complexities are encountered including the difficulty that fetching multiple instructions may require accessing multiple cache lines. The instruction fetch unit encapsulates this complexity using prefetch to try to hide the cost of crossing cache blocks. The instruction fetch unit also provides buffering, essentially acting as an on-demand unit to provide instructions to the issue stage as needed and in the quantity needed.

5.5.4 Value Prediction

One of the technique for increasing the amount of ILP available in a program is value prediction. Value prediction attempts to predict the value that will be produced by an instruction. Obviously, since most instructions produce a different value every time they are executed (or at least a different value from a set of values), value prediction can have only limited success. There are, however, certain instructions for which it is easier to predict the resulting value. In addition, when an instruction produces a value chosen from a small set of potential values, it may be possible to predict the resulting value.

Much of the focus of research on value prediction has been on loads. We can estimate the maximum accuracy of a load value predictor by examining how often a load returns a value that matches a value returned in a recent execution of the load. The simplest case to examine is when the load returns a value that matches the value on the last execution of the load. Because of the high costs of misprediction and the likely case, that misprediction rates will be significant (20% to 50%), researchers have focused on assessing which loads are more predictable and only
attempting to predict those. This leads to a lower misprediction rate, but also fewer candidates for accelerating through prediction. In the limit, if we attempt to predict only those loads that always return the same value, it is likely that only 10% to 15% of the loads can be predicted.

5.5.5 Address Aliasing Prediction

Address aliasing prediction is a simple technique that predicts whether two stores or a load and a store refer to the same memory address. If two such references do not refer to the same address, then they may be safely interchanged. Otherwise, we must wait until the memory addresses accessed by the instructions are known. This limited form of address value speculation has been used by a few processors.

Check Your Progress

3. What is speculation?
4. What are the three approaches to multiple issue processors?
5. Why the Integrated Instruction Fetch Units are introduced?

5.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Two important techniques used in dynamic scheduling are as follows:
   (i) Score boarding
   (ii) Tomasulo Algorithm
2. The main idea of Score boarding is to keep track of the status of instructions, functional units, and registers. It makes sure that the instructions will execute in out-of-order with no conflicts and hardware is available for each instruction.
3. Speculation (also known as speculative loading) is the optimization technique used to maintain a high rate of instruction execution using predictions based on the structure of the program.
4. We generally use three approaches to multiple issue processors (superscalar processors):
   (i) Statically scheduled superscalar processors
   (ii) Very long instruction word (VLIW) processors.
   (iii) Dynamically scheduled superscalar processors.
5. Integrated Instruction Fetch Units are introduced to reduce the complexities of multiple issue. Essentially, this amounts to recognizing that characterizing instruction fetch as a simple single pipe stage provided that the complexities of multiple issues is no longer valid.
5.7 SUMMARY

- Dynamic scheduling is a method in which the hardware determines which instruction need to be executed. In dynamic scheduling, the instructions don’t execute in the order they arrive, but rather execute on the availability of source operands.

- In simple pipelining techniques, they use in-order instructions to execute the programs. This results in each instruction depend on the previous instruction and the performance will be reduced.

- The main idea of Score boarding is to keep track of the status of instructions, functional units, and registers. It makes sure that the instructions will execute in out-of-order with no conflicts and hardware is available for each instruction.

- The second important technique used in dynamic scheduling is the Tomasulo algorithm. It was invented by Robert Tomasulo and was first implemented in the IBM 360/91 Floating-point Unit which came out to practice after three years of CDC 6600.

- Tomasulo Algorithm uses register renaming to eliminate output and anti-dependencies. Output and anti-dependencies are just name dependences only. There will be no actual data dependencies in them.

- The Common Data Bus (CDB) connects the reservation stations directly to the functional units. Reservation stations are buffers which immediately stores all the instruction operands as soon as they are available and later will be used to fetch these instruction operands whenever required.

- Speculation (also known as speculative loading) is the optimization technique used to maintain a high rate of instruction execution using predictions based on the structure of the program. This process is implemented in Explicitly Parallel Instruction Computing (EPIC) processors to maintain high parallelism to reduce the processor memory exchange latency or bottlenecks.

- The most common implementation of hardware-based speculation is based on the modification of the Tomasulo algorithm. Any hardware that implements on Tomasulo algorithm can be extended to support speculation. Hardware-based speculation follows the predictive flow of data values to choose the sequence of execution of instructions.

- Reorder buffer (ROB) is used to pass the results after every instruction that may be speculated. Just like reservation stations even reorder buffer (ROB) provides some extra register to extend the register set. It holds the results of every instruction that are executed after the writing stage and before commit stage.

- The main advantages of VLIW processors are that they reduce the main complexity that occurs due to hardware by moving that complexity to the
software. This makes the processors cheaper, smaller in size by which we can reduce the power consumptions of that is required to operate.

- In order to achieve high Instruction level parallelism, we should be able to deliver a high bandwidth instruction stream along with good predicting branches.
- Integrated Instruction Fetch Units are introduced to reduce the complexities of multiple issue. Essentially, this amounts to recognizing that characterizing instruction fetch as a simple single pipe stage provided that the complexities of multiple issues is no longer valid.
- Value prediction attempts to predict the value that will be produced by an instruction. Obviously, since most instructions produce a different value every time they are executed (or at least a different value from a set of values), value prediction can have only limited success.
- Address aliasing prediction is a simple technique that predicts whether two stores or a load and a store refer to the same memory address.

5.8 KEY WORDS

- **Dynamic Scheduling**: It is a method in which the hardware determines which instruction need to be executed.
- **Speculation**: It is the optimization technique used to maintain a high rate of instruction execution using predictions based on the structure of the program.
- **Address Aliasing Prediction**: It is a simple technique that predicts whether two stores or a load and a store refer to the same memory address.

5.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

**Short Answer Questions**
1. What is dynamic scheduling?
2. What is a scoreboard? Discuss its various stages.
3. What are the three parts of scoreboard?
4. What are the advantages and disadvantages of hardware based speculation?

**Long Answer Questions**
1. Explain the techniques of dynamic scheduling.
2. What are the four major concepts that are necessary to implement the Tomasulo algorithm? Explain.
3. What do you understand by hardware based speculation?
4. What are the various approaches to multiple issue processors?
5. What are key functions involved in Integrated Instruction Fetch Units?
6. Write a note on value and address aliasing prediction.

5.10 FURTHER READINGS

UNIT 6  LIMITATIONS OF ILP

6.0 INTRODUCTION

In this unit, you will learn about the limitations of ILP. To increase the performance of the pipelined processors, the exploitation of ILP started to increase rapidly. For the first few years, the pipelining for multiple issues, speculation, and dynamic scheduling was progressed. But how much ILP is to be used is the biggest parameter that is not known till date.

The advancement in the field of compiling software as the time passes may create the possibility of overcoming the limitations but when practically experimented in reality such advancements may not completely remove the limitations. You will also learn about the multi-threading in the processors.

6.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the various approaches of ILP
- Understand the models of multithreading

6.2 HARDWARE AND SOFTWARE SPECULATION

There are two possibilities of approaching ILP i.e. hardware, and software. Now we will start our work using the approaches only.

The Hardware Modeling

To know the limitations of ILP, the very first step is to understand the ideal processor. In an ideal processor, the constraints on ILP are removed.
There are few assumptions for a perfect processor.

1. Branch prediction
2. Jump prediction
3. Perfect caches
4. Renaming of registers
5. Analysis

1. Branch prediction
   All conditions of the branches are exactly estimated.

2. Jump prediction
   The prediction of any type of jump is done perfectly. When these both predictions are combined together then there is a possibility of unbound buffer and perfect speculation for the available instructions.

3. Perfect caches
   The clock cycle taken for all types of memory is 1. Typical superscalar methods will consume greater space.

4. Renaming
   Since the availability of virtual registers is infinite, the write after read and write after write computers are avoided.

5. Analysis
   The locations of the memory address are known correctly. Branch and jump predictions eradicate all types of control dependencies. Register renaming and analysis remove all the true data which have dependencies. Together these four assumptions tell that while program execution the instruction can be scheduled immediately by the next execution. Initially, the processor can be issued in unlimited ways for one computation. Latencies of functional units are assumed to one cycle. The latencies which are more than one cycle decrease the number of issues per cycle. The parallelism can be measured with a set of programs that were compiled and optimized with microprocessors with the interlocked pipelining system. The trace is scheduled for every instruction limited only by dependencies of data.

Limitations of Window Size and Maximum Count Issue

For the perfect branch, the processor needs to be built properly with perfect analysis since the compile time cannot be always perfect. A processor must be able to match the amount of parallelism by the processor which is done ideally.
The processor must be able to satisfy the following conditions:

1. Predict all the branches perfectly by finding out the set of instructions.
2. Find whether there are any data dependencies among the instructions.
3. Avoid write after read and write after write processors.
4. Avoid memory dependencies among the instructions.

The analysis of the processor is quite difficult. To determine instructions have dependencies among them, all the instructions are register-register comparisons. To detect instruction dependencies they are \( n \) number of comparisons. Therefore it is obvious that costs will exceed their limits. A group of instructions which are simultaneously examined for execution is called a window.

Each instruction must be kept in processor and comparisons required clock cycle which gives maximum completion rate time the window size times the no of operands per instruction. The size of the window limits the instructions which begin execution in a given cycle. The real processor has limited functional units and buses as well as register ports. The number of instructions that may issue, begin execution or commit in the same clock cycle is smaller than the window size.

**Effects of Realistic Branch and Jump Prediction**

As discussed before, the ideal processors which we design have branches that can be perfectly predicted. The result of the branch in a program is known before the first instruction is executed. Jump predictors play a vital role with branch predictors to maximize accuracy.

There are different levels of branch prediction. These levels can be classified into 5 types.

1. Perfect: as the name itself says, the branches and jumps are perfectly predicted.
2. Tournament based branch predictors: this prediction correlates 2-bit predictor and a non-correlating 2-bit predictor with a selection processor which chooses the best predictor. The prediction buffer contains 8k entries. The correlating predictors exclusive or for branch address and history
3. Standard 2 bit predictor with 512 2-bit entries: here 16 entry buffers are assumed.
4. Profile-based
5. None: no prediction of the branch is done.
Effects of Finite Registers

Ideal processors always try to eradicate all the name dependences among the register references using a set of the register that is virtual.

Effects of Imperfect Analysis

If the memory dependent analysis doesn’t eliminate the dependencies then the analysis is not possible. The analysis cannot be perfect at compile time.

There are three models of analysis. They are:

2. Inspection: this examines the access whether the processor can be interfered at compile time.
3. Note: all references of memory are assumed to conflict.

6.3 MULTITHREADING

The ability of a computer's processing unit both for the single or multi-core processor which provides multiple threading by the OS of the system is multithreading.

Where multiprocessing systems include multiple complete processing units in one or more cores, multithreading aims to increase utilization of a single core by using thread-level parallelism, as well as instruction-level parallelism. As the two techniques are complementary, they are sometimes combined in systems with multiple multithreading CPUs and with CPUs with multiple multithreading cores.

Advantages of Multithreading

1. Responsiveness: even if the part of the application is blocked, the program should continue running which increases responsiveness.
2. Resource sharing: the ability of threads sharing both memory and resource process.
3. Economy: threads share the memory and resource process which provide the lesser cost.
4. Utilization: the threads might run in parallel processors which increases the concurrency in multithreading.
Models of Multithreading

1. Many to One Model

- Here the model tries to map many user levels to one thread.
- Thread management is done in order to make the model efficient.
- One thread can use the kernel at a time. Therefore they cannot run in parallel.

2. One to One Model

- This model maps each of the users into a kernel thread.
- Here the more concurrency than many to one model is provided.
- This model is used to run parallel processors.
3. **Many to Many Model**

- Here, many user threads can be made into smaller kernel threads.
- The number of kernel threads should be specific to an application.
- User can create as many as threads as required and can make to run in parallel.

---

**Check Your Progress**

1. What is the need of register renaming?
2. How you will measure the parallelism?

---

**6.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS**

1. Register renaming and analysis remove all the true data which have dependencies.
2. The parallelism can be measured with a set of programs that were compiled and optimized with microprocessors with the interlocked pipelining system.

---

**6.5 SUMMARY**

- There are two possibilities of approaching ILP i.e. hardware, and software.
- Branch and jump predictions eradicate all types of control dependencies. Register renaming and analysis remove all the true data which have dependencies.
• Latencies of functional units are assumed to one cycle. The latencies which are more than one cycle decrease the number of issues per cycle. The parallelism can be measured with a set of programs that were compiled and optimized with microprocessors with the interlocked pipelining system.

• The analysis of the processor is quite difficult. To determine instructions have dependencies among them, all the instructions are register-register comparisons. To detect instruction dependencies they are n number of comparisons. Therefore it is obvious that costs will exceed their limits. A group of instructions which are simultaneously examined for execution is called window.

• Where multiprocessing systems include multiple complete processing units in one or more cores, multithreading aims to increase utilization of a single core by using thread-level parallelism, as well as instruction-level parallelism.

6.6 KEY WORDS

• Multiprocessing: It is the use of two or more central processing units within a single computer system.

• Multithreading: It is a technique by which a single set of code can be used by several processors at different stages of execution.

6.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. What are the assumptions for a perfect processor?
2. What are the advantages of multithreading?

Long Answer Questions

1. What are the different levels of branch prediction?
2. Explain the various models of multithreading.

6.8 FURTHER READINGS


Limitations of ILP


7.0 INTRODUCTION

In this unit, you will learn about the basic concept of multiprocessor, multithreading and symmetric shared memory architecture. A multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM. Multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion. Symmetric Shared Memory Architecture has several processors with a single physical memory that is shared by all processors via a shared bus.

7.1 OBJECTIVES

After going through this unit, you will be able to:

- Define multiprocessor
- Discuss the applications and benefits of multiprocessor
- Discuss the approaches of multithreading
- Understand the concept of shared memory architecture
7.2 MULTIPROCESSORS

A computer system that has more than one central processor and a common random access memory (RAM) is known as multiprocessor computer. It will increase the execution speed and fault tolerance power. Multiprocessor system can be categorized as shared memory multiprocessor and distributed memory multiprocessor. In case of shared memory multiprocessors system, there is a common memory that is shared by all the CPUs, while in distributed memory multiprocessor system there is own private memory with every CPU.

Applications of Multiprocessor

- Multiprocessor are generally useful for vector processing, such as single instruction, multiple data stream (SIMD).
- MISD (Multiple instruction, single data stream), is useful for hyper-threading or pipelined processors.
- Inside a single system for executing multiple, individual series of instructions in multiple perspectives, such as multiple instruction, multiple data stream (MIMD).

Benefits of using a Multiprocessor

- Enhanced the throughput.
- Number of applications can be run in a unit time
- Cost saving
- More reliable
- Optimization of hardware devices among CPUs.
7.3 THREAD LEVEL PARALLELISM (TASK PARALLELISM, FUNCTION PARALLELISM, CONTROL PARALLELISM)

It is a form of parallel computing. In this parallelism, there are number of processors to distribute the execution of processes and threads across different parallel processor nodes. We start the development of multiprocessors in early 2000 for the following reasons:

- Uniprocessor performance was very poor.
- It was very difficult for a uni-processor to dissipate the heat generated with high clock rates.
- There was a requirement of high-performance servers.
- Thread-level parallelism is of high performance by nature.

Exploiting other types of Parallelism

The use of ILP in some domain is limited or difficult. Uses of ILP also increase the power consumption. In some applications domains where higher level of parallelism is required that cannot be exploited with the approaches used to exploit ILP.

Take an example for online transaction processing system. In the online processing there is number of queries and updates from the user side. As we know that these queries and updates are generally independent of one another, so these can be processed in parallel. This higher-level parallelism is called thread level parallelism. Thread level parallelism is logically structured as separate threads of execution. A separate process with its own instructions and data is known as thread. A thread may represent a process that is part of a parallel program consisting of multiple processes, or it may represent an independent program on its own. There are some states such as instructions, PC, data, etc. in the execution cycle of a thread.

Thread level parallelism exploits the explicit level parallelism represented by the use of multiple threads of execution that are inherently parallel.

Thread level parallelism is more cost-effective to exploit than instruction level parallelism. It is an alternative to instruction level parallelism. There are many important applications where thread level parallelism occurs naturally, as it does in many server applications. Applications where same operation can be performed on multiple data exploit data level parallelism.

Since ILP and TLP exploit two different types of parallel structure in a program, it is a natural option to combine these two types of parallelism. The datapath that has already been designed has a number of functional units remaining idle because of the insufficient ILP caused by stalls and dependences. This can be utilized to exploit TLP and thus make the functional units busy. There are predominantly two strategies for exploiting TLP along with ILP Multithreading...
Multithreading: It allows multiple threads to share the functional units of a single processor in an overlapping fashion. In order to enable the multithreading, processor creates the duplicate copies of the independent state of each thread. It creates a separate copy of the register file, a separate PC, and a separate page table. Using the virtual memory mechanisms, memory can be shared. For this, the hardware must support the ability to change to a different thread relatively quickly; in particular, a thread switch should be much more efficient than a process switch, which typically requires hundreds to thousands of processor cycles.

Multithreading approaches are as follows:

(i) Fine grained
(ii) Coarse grained

In case of fine-grained multithreading, it switches between threads on each instruction. In fine grained multithreading multiple threads are executed that can also be interleaved. This interleaving is usually done in a round-robin fashion, skipping any threads that are stalled at that time. In order to support this, the CPU must be able to switch threads on every clock cycle. The main advantage of fine-grained multithreading is that it can hide the throughput losses that arise from both short and long stalls, since instructions from other threads can be executed when one thread stalls. But it slows down the execution of the individual threads, since a thread that is ready to execute without stalls will be delayed by instructions from other threads.

In case of Coarse-grained multithreading, it switches threads only on costly stalls, such as level two cache misses. This allows some time for thread switching and is much less likely to slow the processor down, since instructions from other threads will only be issued, when a thread encounters a costly stall. Coarse-grained multithreading, however, is limited in its ability to overcome throughput losses, especially from shorter stalls. This limitation arises from the pipeline start-up costs of coarse-grain multithreading. Because a CPU with coarse-grained multithreading issues instructions from a single thread, when a stall occurs, the pipeline must be emptied or frozen and then fill in instructions from the new thread. Because of this start-up overhead, coarse-grained multithreading is much more useful for reducing the penalty of high cost stalls, where pipeline refill is negligible compared to the stall time.

Simultaneous Multithreading: It is a variant on multithreading. When we only issue instructions from one thread, there may not be enough parallelism available and all the functional units may not be used. Instead, if we issue instructions from multiple threads in the same clock cycle, we will be able to better utilize the functional
units. This is the concept of simultaneous multithreading. We try to use the resources of a multiple issue, dynamically scheduled superscalar to exploit TLP on top of ILP. The dynamically scheduled processor already has many hardware mechanisms to support multithreading.

- A number of virtual registers is used to hold the register sets of independent threads
- Registers are given a unique id by renaming so that the instructions from multiple threads can be joined in the data-path without confusing sources and destinations across threads and
- Out-of-order completion allows the threads to execute out of order, and get better utilization of the hardware.

Fig. 7.2 Simultaneous Multithreading Vs others

7.3.1 Parallel Architectures

To fulfill the demand of low cost, high performance, accurate results in applications, we need a new technology that should be more effective. This demands result is the parallel computer architecture. Flynn proposed such a model for the categorization of the computers that is useful till today. Multiprogramming, multiprocessing, or multi-computing are the different practice of concurrent events in modern computer. According M.J. Flynn computer classification model, classification is based on number of instructions and data items that are manipulated simultaneously. **Instruction Stream** is the sequence of instructions read from memory. **Data stream** is the set of operations performed on the data in the processor.
** Flynn’s Classification **

a. SISD (Single instruction stream, single data stream)  
b. SIMD (Single instruction stream, multiple data stream)  
c. MISD (Multiple instruction stream, single data stream)  
b. MIMD (Multiple instruction stream, multiple data stream)

** a. SISD (Single Instruction Stream, Single Data Stream) **

Following are the characteristics of SISD processors.

- It is a uni-processor machine
- It is able to execute a single instruction, operating on a single data stream.
- Machine instructions followed a sequential manner
- Also known as sequential computers.
- Most of conventional computers today use SISD architecture.

![Fig. 7.3 Single Instruction Stream, Single Data Stream](image-url)

** b. Single Instruction Stream, Multiple Data Stream **

Following are the characteristics of SIMD processors.

- Multiple processors execute the same instruction.
- SIMD machines are cost effective platforms.
- Individual processor has individual data memory (hence multiple data), but memory and control processor is single, which fetches and dispatches instructions.
- Example of SIMD is vector architectures.
c. Multiple Instruction Streams, Single Data Stream (MISD)

Following are the characteristics of MISD processors.
- Each processor executes a different sequence of instructions.
- One single-data stream operated by multiple processors.
- In practice, such type of organization has never been used.

![Fig. 7.4 Single Instruction Stream, Multiple Data Stream](image)

![MISD Diagram](image)

d. Multiple Instruction Streams, Multiple Data Stream (MIMD)

Following are the characteristics of MIMD processors.
- Each processor takes its own instructions and operates on its own data.
- MIMD machines are the most used processors.
- If the processor’s control unit can send different instructions to each ALU in parallel then the architecture is MIMD.
- A superscalar architecture is also MIMD.
- For multiple execution units, multiple instructions can be issued in parallel.
- More flexible.
Multiprocessor and Thread Level Parallelism

NOTES

Fig. 7.6 Multiple Instruction Streams, Multiple Data Streams

Check Your Progress
1. What is multiprocessor?
2. What are the approaches of multithreading?

7.4 SYMMETRIC SHARED MEMORY ARCHITECTURE

Following are the features of shared memory architecture.

- The Symmetric Shared Memory Architecture has several processors with a single physical memory that is shared by all processors via a shared bus.
- Multiple memory modules are connected to multiple processors so that each processor can access any other processor’s memory module. This multiprocessor uses a shared address space.

Fig. 7.7 Symmetric Shared Memory Multiprocessor
- Communication has loads and stores and there is no explicit recipient of a shared memory access.
- All the processors run by a single image of the operating system.

**Types of Shared Memory Architecture**

There are two types of shared memory architecture.

1. **Uniform Memory Access (UMA)**
   
   Following are the characteristics of UMA.
   - Main memory accessing time is the same for all processors since they are equally close to all memory locations.
   - Machines that use Uniform Memory Access is also known as Symmetric Multiprocessors (SMPs).
   - All memory accesses use same shared memory bus in symmetric multiprocessors.
   - Contention arises as more CPUs are added, competition for access to the bus leads to a decline in performance.
   - Thus, scalability is limited to about 32 processors.

2. **Non-Uniform Memory Access (NUMA)**
   
   Following are the characteristics of NUMA.
   - Since memory is physically distributed, it is faster for a processor to access its own local memory than non-local.
   - All processors are not equally close to all memory locations.
   - A processor’s internal computations that are being done in its local memory reduce the memory contention.
   - Designed to exceed the scalability limits of SMPs.

**Distributed Shared Memory (DSM)**

Following are the characteristics that a processor shows having distributed shared memory.

- Distributed shared memory use Read/Write ops in shared virtual space
- There is No Send and Receive primitives to be used by application
  - Under covers, Send and Receive used by DSM manager
- Restricted looking
- Need concurrent access
- All segments use individual machine to present.
Multiprocessor and Thread Level Parallelism

NOTES

- A processor uses local pages using the ordinary hardware, at full memory speed.
- An effort to orientate a page located on a different module can create a hardware page fault.
- The operating system sends a message to the remote machine, which finds the needed page and sends it to the requesting processor.
- All the faulty instructions are started again, and can now complete.

Fig. 7.8 Distributed Shared Memory Multiprocessor

Advantages of DSM

- It will protect programmers from Send/Receive primitives
- It follows a single address space
- In case of a block removal, it will exploit the locality-of-reference
- As it uses simpler software interfaces, and cheaper off-the-shelf hardware, so cheaper than dedicated multiprocessor systems
- More virtual memory space
- Distributed shared memory is portable programming as they use common DSM programming interface

Disadvantages of DSM

- To write the program, programmers need to understand consistency models
- Use asynchronous message-passing, so cannot be more competent than message-passing implementations
With the help of DSM manager software control, programmers cannot use their own message-passing solutions.

Challenges for Parallel Processing
Suppose you want to achieve a speedup of 80 with 100 processors. What fraction of the original computation can be sequential?

\[
\frac{Speedup_{Overall}}{1 - \frac{Fraction_{enhanced}}{Speedup_{enhanced}}} = \frac{80}{1 - \frac{Fraction_{enhanced}}{100}}
\]

\[
Fraction_{enhanced} = \frac{0.9975}{100}
\]

- First challenge is finding concurrency in a program
- Problem in associating data with tasks and doing it in a way that our target audience will be able to use correctly. It’s also known as locality problem.
- Bandwidth and latencies to memory plus interconnects between processing elements. It’s the scalability support problem in hardware.
- Libraries, scalable algorithms, and adaptive runtimes to map high level software onto platform details. Its scalability support problem in software.
- Protocols construction to enable programmers write programs free from deadlock and race conditions.
- Designing the tools, APIs and methodologies to support the debugging process.
- Error recovery is also one of the problem in parallel processing
- Support for fault tolerance.
- Challenge for software engineering practices such as code reuse, incremental parallelism, and compensability.
- Support for portable performance.

Data Communication Models for Multiprocessors
1. **Shared Memory**: it access shared address space completely through load and store operations.
2. **Message-Passing**: it is done by explicitly passing messages among the processors can cite software with Remote Procedure Call (RPC) often through library, such as MPI (Message Passing Interface). It is also called as “Synchronous communication”.

NOTES
Multiprocessor and Thread Level Parallelism

NOTES

Self-Instructional Material

Message-Passing Multiprocessor

There may be multiple private address space in an address space that are logically disjoint. These private addresses cannot be addressed by any remote processor. On different processors, the same physical address leads to two different locations in two different memories.

Symmetric Shared-Memory Architectures

- Multilevel caches can considerably diminish the memory bandwidth that demands of a processor.
- It is very cost-effective.
- It works as plug and play by using the processor and cache sub-system on a board into the bus backplane.
- Symmetric shared memory holds caching of private data and shared data.

Private data: It is used by a single processor. When a private item is cached, its locality is migrated to the cache because no other processor uses the data; the program behaviour is identical to that in a uniprocessor.

Shared data: It is used by multiple processors. In case of shared data is cached, the shared value may be replicated in multiple caches advantages. It reduces access latency and memory contention creates a new problem that is cache coherence.

Cache Coherence

Caching shared data creates a new problem as the view of memory supposed by two different processors via individual caches. Without any supplementary precautions, it could end up considering two different values. Different values mean two different processors have two different values for the same location. This difficulty is generally known as cache coherence problem.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Cache contents for CPU A</th>
<th>Cache contents for CPU B</th>
<th>Memory contents for location X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>CPU A read X</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>CPU B read X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>CPU A store 0 into X</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Fig. 7.9 Cache Coherence Problem for a Single Memory Location**

Informally

- Any read must return the most recent write
- It is very strict
- It is very difficult to implement
Better

- It is mandatory that Write finally be seen by a read
- And All writes are seen in proper order

Rules:

- If P writes x and then P1 reads it, P’s write will be seen by P1 if the read and write are sufficiently far apart
- Writes to a single location are serialized
- Latest write will be seen
- Otherwise writes are in illogical order

Two different phase of memory system:

- Coherence
- Consistency
- Program order is preserved.
- Processor should not continuously read the old data value. Write to the same location are serialized.

The properties given above are adequate to ensure coherence, when a written value will be seen is also important. This issue is defined by memory consistency model. Coherence and consistency are complementary.

Basic Schemes for Enforcing Coherence

Coherence cache offers:

- Migration: in migration data item can be moved to a local cache and this data is used there in a transparent fashion.
- Replication for shared data that are being simultaneously read. Both are critical to performance in accessing shared data.

To overcome these problems, take up a hardware solution by setting up a protocol to sustain coherent caches named as Cache Coherence Protocols. These protocols are implemented for tracking the state of any sharing of a data block. Two classes of protocols are:

(i) Directory based protocols
(ii) Snooping based Protocols

Directory based

Directory is the location where all the sharing status of a block of physical memory is kept. It has comparatively higher implementation overhead than snooping. It can scale to larger processor count.
Snooping

Every cache that has a copy of data also has a copy of the sharing status of the block. There is no centralized state. One can also access the caches by some broadcast medium.

Snooping protocols are accepted with multiprocessor and caches fond of single shared memory, because they can utilize the existing physical connection-bus to memory, to cross-examine the status of the caches.

Basic Snoopy Protocols

Write strategies
- Write-through: in this memory is always up-to-date
- Write-back: in this snoop in caches to locate most recent copy Write
  Invalidate Protocol
- Multiple readers, single writer
- Write to shared data: in this an invalidate is sent to all caches which snoop and invalidate any copies

Read miss: In read miss, it will miss in the cache and fetch a new copy of the data.
Write Broadcast/Update Protocol (typically write through)
- Write to shared data: broadcast on bus, processors snoop, and update any copies
- Read miss: memory/cache is always up-to-date.
  Write serialization: bus serializes requests!
- Bus is single point of arbitration

Check Your Progress

3. What are the two types of shared memory architecture?
4. What are the two classes of protocols for enforcing cache coherence?

7.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A computer system that has more than one central processor and a common random access memory (RAM) is known as multiprocessor computer.
2. Multithreading approaches are as follows:
   (i) Fine grained
   (ii) Coarse grained
3. There are two types of shared memory architecture.
   (i) Uniform memory access
   (ii) Non-uniform memory access

4. Two classes of protocols for enforcing cache coherence are:
   (i) Directory based protocols
   (ii) Snooping based Protocols

7.6 SUMMARY

- A computer system that has more than one central processor and a common random access memory (RAM) is known as multiprocessor computer. It will increase the execution speed and fault tolerance power.
- Thread level parallelism is more cost-effective to exploit than instruction level parallelism. It is an alternative to instruction level parallelism.
- Multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion. In order to enable the multithreading, processor creates the duplicate copies of the independent state of each thread.
- When we only issue instructions from one thread, there may not be enough parallelism available and all the functional units may not be used. Instead, if we issue instructions from multiple threads in the same clock cycle, we will be able to better utilize the functional units. This is the concept of simultaneous multithreading.
- Symmetric Shared Memory Architecture has several processors with a single physical memory that is shared by all processors via a shared bus.
- There are two types of shared memory architecture i.e. uniform memory access and non-uniform memory access
- Distributed shared memory use Read/Write ops in shared virtual space.
- Caching shared data creates a new problem as the view of memory supposed by two different processors via individual caches. Without any supplementary precautions, it could end up considering two different values. Different values mean two different processors have two different values for the same location. This difficulty is generally known as cache coherence problem.

7.7 KEY WORDS

- **Multiprocessor Computer**: It is a computer system that has more than one central processor and a common random access memory (RAM).
Multiprocessor and Thread Level Parallelism

- **Multithreading**: It allows multiple threads to share the functional units of a single processor in an overlapping fashion.
- **Pipelining**: It is a technique for implementing instruction-level parallelism within a single processor.

### 7.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

**Short Answer Questions**
1. What are the applications and benefits of multiprocessor?
2. What is parallel architecture?
3. Discuss the types of symmetric shared memory architecture.
4. What are the advantages and disadvantages of DSM?
5. Discuss the challenges of parallel processing.

**Long Answer Questions**
1. What is Flynn computer classification model? Explain.
2. What are the approaches of multithreading?
3. Explain the symmetric shared memory architecture.

### 7.9 FURTHER READINGS


UNIT 8  PERFORMANCE AND ARCHITECTURES

8.0 INTRODUCTION

In this unit, you will learn about the performance of symmetric shared memory architecture and properties and protocols in distributed shared memory architecture. We know that performance is a metric that shows the amount of work completed with a given time. There are several factors on which the performance depends in shared memory architecture which are discussed in this unit.

8.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the performance symmetric shared memory architecture
- Understand the properties of distributed shared memory architecture
- Discuss the directory protocols

8.2 PERFORMANCE OF SYMMETRIC SHARED MEMORY ARCHITECTURES

A Symmetric multiprocessor has considerably better performance than a uniprocessor when more than one program executes at the same time because different programs can run on different processor simultaneously. Asymmetric multiprocessing (AMP) allows only one processor to run a program or task at a time. For example, AMP can be used in assigning specific tasks to CPU based on priority and importance of task completion. AMP was created well before SMP in terms of handling multiple CPUs, which explains the lack of performance based on the example provided.
Administrators often experience a loss of hardware efficiency in cases where an SMP environment processes many jobs. To achieve the processor utilization at its maximum potential, software programs have been developed to schedule jobs and other functions of the computer. We can achieve maximum potential by scheduling each CPU separately, as well as being able to integrate multiple SMP machines and clusters using good software packages. Access to RAM is serialized; this and cache coherency issues cause performance to lag slightly behind the number of additional processors in the system.

For the performance measurement, we can say that cache performance is a group of the following:

- Uni-processor cache will miss the traffic.
- Traffic due to communication – invalidation and subsequent cache misses
- In case of the processor count, cache size, and block size change, it can have an effect on these two components of miss rate
- Missing rate of uniprocessor
- Communication miss rate includes coherence misses
- Coherence misses can be divided into True sharing misses and False sharing misses.

**True sharing miss**

True sharing miss occurs when:

- The first write by a PE (processing element) to a shared cache block cause an invalidation to set up ownership of that block.
- When another PE go for reading a modified word in that cache block, a miss take place and the resultant block is transferred.

**False sharing miss**

False sharing miss occur when:

- It happens when a block is invalidate (and a subsequent reference causes a miss) due to some word in the block, other than the one being read, is written to
- A block is shared without the sharing of words in the cache, and this miss would not occur if the block size were a single word

Suppose that words y1 and y2 are in the same cache block in the shared state in the caches of processors P1 and P2. Assuming the sequence of events, one can find out each miss as a true sharing miss or a false sharing miss.
You have learnt that the Distributed Shared Memory (DSM) is a form of memory architecture in which the physically separated memories can be addressed as a single logically shared address space. Here, shared does not mean that there is a single centralized memory, but it means that the address space is shared. Same physical address on two processors refer to the same location in the memory. The architecture has the following properties:

- It uses very simple approach.
- In this shared data is uncacheable while the private data use caches.
- In this architecture the shared data has a very long latency to access memory.
- Like a bus, its interconnection are not utilized as a single point.

### 8.3 DISTRIBUTED SHARED MEMORY ARCHITECTURES

You have learnt that the Distributed Shared Memory (DSM) is a form of memory architecture in which the physically separated memories can be addressed as a single logically shared address space. Here, shared does not mean that there is a single centralized memory, but it means that the address space is shared. Same physical address on two processors refer to the same location in the memory. The architecture has the following properties:

- It uses very simple approach.
- In this shared data is uncacheable while the private data use caches.
- In this architecture the shared data has a very long latency to access memory.
- Like a bus, its interconnection are not utilized as a single point.

---

**Fig. 8.1 True Vs False Vs Hit**

<table>
<thead>
<tr>
<th>Time</th>
<th>P1</th>
<th>P2</th>
<th>True, False, Hit? Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write y1</td>
<td>True miss; invalidate y1 in P2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Read y2</td>
<td>False miss; y1 irrelevant to P2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Write y1</td>
<td>False miss; y1 irrelevant to P2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Write y2</td>
<td>False miss; y1 irrelevant to P2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Read y2</td>
<td>True miss; invalidate y2 in P1</td>
<td></td>
</tr>
</tbody>
</table>
For the prevention of directory to become the bottleneck, one should distribute directory entries with memory, each keeping track of which processors have copies of their memory blocks.

**Directory Protocols**

It has the following states:

- **Shared**: In shared, one or more processors have the block cached, and the value in memory is up-to-date.
- **Uncached**: In uncached, copy of cache block is not handled by any processor.
- **Exclusive**: In the exclusive state, exactly one processor has a copy of the cache block, and it has written the block. So, the memory copy is out of date.

**Keep it simple(r):**

- Writes to non-exclusive data => write miss.
- Processor blocks, till the access is not completed.
### Message for Directory Protocols

<table>
<thead>
<tr>
<th>Message type</th>
<th>Source</th>
<th>Destination</th>
<th>Message contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read miss</td>
<td>local cache</td>
<td>home directory</td>
<td>P.A. Processor P has a read miss at address A; request data and make P a read sharer.</td>
</tr>
<tr>
<td>Write miss</td>
<td>local cache</td>
<td>home directory</td>
<td>P.A. Processor P has a write miss at address A; request data and make P the exclusive owner.</td>
</tr>
<tr>
<td>Invalidate</td>
<td>home directory</td>
<td>remote cache</td>
<td>A Invalidate a shared copy of data at address A.</td>
</tr>
<tr>
<td>Fetch</td>
<td>home directory</td>
<td>remote cache</td>
<td>A Fetch the block at address A and send it to its home directory, change the state of A in the remote cache to shared.</td>
</tr>
<tr>
<td>Fetch/invalidate</td>
<td>home directory</td>
<td>remote cache</td>
<td>A Fetch the block at address A and send it to its home directory, invalidate the block in the cache.</td>
</tr>
<tr>
<td>Data value reply</td>
<td>home directory</td>
<td>local cache</td>
<td>D Return a data value from the home memory.</td>
</tr>
<tr>
<td>Data write back</td>
<td>remote cache</td>
<td>home directory</td>
<td>A.D. Write back a data value for address A.</td>
</tr>
</tbody>
</table>

**Fig. 8.4** Message for directory Protocols

**Local Node:** local node is the node from where a request originates.

**Home Node:** The memory location and directory entry of an address exist in a home node.

**Remote Node:** This node has a copy of a cache block (exclusive or shared).

### State transition diagram for individual cache block

**Fig. 8.5** State transition diagram for individual cache block
On comparing to snooping protocols:

- It has identical states.
- Its stimulus is almost identical.
- Write a shared cache block is treated as a write miss.
- Cache block must be in exclusive state when it is written.
- Any shared block must be up to date in memory.

Directory Operations: Requests and Actions

Message sent to directory performs two actions:

1. It update the directory.
2. It has a number of messages to satisfy the request.

- **Block is in Uncached state:** The copy in memory is the current value; only possible requests for that block are:
  - Read miss: requesting processor sent data from memory & requestor made only sharing node; state of block made Shared.
  - Write miss: requesting processor is sent the value & becomes the Sharing node. The block is made Exclusive to indicate that the only valid copy is cached. Sharers indicate the identity of the owner.
- **Block is Shared,** the memory value is up-to-date:**
  - Read miss: requesting processor is sent back the data from memory & requesting processor is added to the sharing set.
  - Write miss: requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages & Sharers is set to identity of requesting processor. The state of the block is made Exclusive.
- **Block is Exclusive:** current value of the block is held in the cache of the processor identified by the set Sharers (the owner) => three possible directory requests:
**Check Your Progress**

1. What are the two categories of coherence miss?
2. What are the three types of nodes?

### 8.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Coherence misses can be divided into True sharing misses and False sharing misses.
2. Local, home and remote nodes are the three types of nodes.

### 8.5 SUMMARY

- A Symmetric multiprocessor has considerably better performance than a uni-processor when more than one program executes at the same time because different programs can run on different processors simultaneously. Asymmetric multiprocessing (AMP) allows only one processor to run a program or task at a time.
- To achieve the processor utilization at its maximum potential, software programs have been developed to schedule jobs and other functions of the computer.
- Distributed shared memory (DSM) is a form of memory architecture in which physically separated memories can be addressed as one logically shared address space.
8.6 KEY WORDS

- **Shared Memory**: It is memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies.
- **Performance**: It is the amount of useful work accomplished by a system.

8.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

**Short Answer Questions**

1. Discuss the performance of symmetric shared memory architectures.
2. What are the factors that should be considered for cache performance measurement?
3. Write the properties of distributed shared memory architecture.

**Long Answer Questions**

1. Explain the types of cache coherence misses.
2. What are the states of directory protocols in distributed shared memory?
3. What are directory operations?

8.8 FURTHER READINGS


UNIT 9  SYNCHRONIZATION MODELS

9.0 INTRODUCTION

In this unit, you will learn about the synchronization and its models. Synchronization is a mechanism that results in concurrent execution of processes simultaneously on a particular program segment. The portion of the program segment is known as critical section. You will also learn about the memory consistent models that maintain the memory consistent in a shared memory system.

9.1 OBJECTIVES

After going through this unit, you will be able to:

- Define synchronization
- Explain memory consistent models
- Discuss cross cutting issues

9.2 SYNCHRONIZATION

Synchronization refers to a mechanism that deals with two or more concurrent processes or threads do not execute on some particular program segment simultaneously. Basically that program segment referred as critical section. Synchronization mechanisms are typically built with user-level software routines that rely on hardware supplied synchronization instructions.
Synchronization Models

**Need for synchronization**

It is important to identify when it is safe for different processes to use shared data while using multiprocessor architecture.

### Issues for Synchronization

- Uninterruptible instruction to fetch and update memory
- User level synchronization operation using this primitive;
- For large scale MPs, synchronization can be a bottleneck requires techniques to reduce contention and latency of synchronization

### 9.3 MEMORY CONSISTENCY MODELS

Consistency models are used in distributed shared memory systems. The system supports a given model if it follows some specific rules while performing operations on memory. A model acts as a contract between the programmer and the system. The system guarantees that the memory will be consistent if the programmer follows the rules and results of reading, writing, or updating memory will be predictable. It is different from coherence where the systems are cached or cache-less. In case of coherence, the writes to a single location can be seen by all processors as it is maintained at global order. It deals with the ordering of operations to multiple locations with respect to all processors.

Cache coherence results that multiple processors use a consistent view of memory. Consistent view means that one processor uses the value that has been updated by another processor. Since the processors use the shared architecture, here, the question arises that in what way the one processor uses the writes of another processor? Since the only way to identify the writes of another processor is through the read. Another question arises that what properties are enforced to among the reads and writes by different processors at different locations?

Consider an example, there are two code segments from processor P1 and P2 as shown below.

P1: A = 0; P2: B = 0;

.....

A = 1; B = 1;

L1: if (B == 0) ... L2: if (A == 0) ...

Here, we assume that the processes are running on different processors. A and B have initial value zero that are originally cached by both the processors. It is also assumed that writes take immediate effect and seen by the processors immediately. Both if statements can be true means either A or B is assigned the value 1. What if write invalidate is delayed & processor continues?
In such cases, the model for memory consistency is called *sequential consistency*. Sequential consistency states that result of any execution is the same as if the accesses of each processor were kept in order and the accesses among different processors were interleaved.

It states that a write to a variable does not to be seen immediately. The writes to a by different processors must be seen in the same order by all processors. The condition for sequential consistency is that the result of execution is same as the execution takes place in sequential order by all the processors. It is required to preserve the sequential order of execution. There should be instantaneously and atomic operations with respect to every other processor.

**Relaxed Consistency Model**

The main idea in relaxed consistency model is that it allow reads and writes to complete out of order, but to use synchronization operations to enforce ordering, so that a synchronized program behaves as if the processor were sequentially consistent.

By relaxing these orderings, the processor may obtain performance advantages and also specifies range of legal compiler optimizations on shared data. Unless synchronization points are clearly defined and programs are synchronized, compiler could not interchange read and write of two shared data items because it might affect the semantics of the program. There are three major sets of relaxed orderings:

1. **W_R ordering** (all writes completed before next read)
   - It yields a model called as processor consistency. Because retains ordering among writes, many programs that operate under sequential consistency operate under this model, without additional synchronization.

2. **W_W ordering** (all writes completed before next write)

3. **R_W and R_R orderings**, it depends on ordering restrictions and how synchronization operations enforce ordering.

**9.4 CROSS CUTTING ISSUES**

Multiprocessors redefine system characteristics such as scalability, memory latency and performance assessment. It creates design problems that cut the spectrum.

**Compiler Optimization and the Consistency Model**

Inexplicitly parallel programs, unless the synchronization points are obviously define and the programs are synchronized, the compiler cannot exchange a read and a write of two unlike shared data items because such transformations might affect the semantics of the program. This avoids the optimizations, such as register allotment of shared data, because such a process usually interchanges reads and writes.
Using Speculation to Hide Latency in Strict Consistency Models

The key idea is for the processor to use dynamic scheduling to reorder memory references, letting them possibly execute out of order. Executing the memory references out of order may generate violations of sequential consistency, which might affect the execution of the program. This possibility is avoided by using the delayed commit feature of a speculative processor.

Assume the coherency protocol is based on invalidation. If the processor obtains invalidation for a memory position before the memory location is detected, the processor considers assumption recovery to back out of the computation and restart with the memory position whose address was invalidated. If the reordering of memory requests by the processor gives way an execution order that could result in an outcome that differs from what would have been seen under sequential consistency, the processor will redo the execution. The key having this approach is that the processor required only guarantee that the result would be the same as if all accesses were completed in order, and it can attain this by detecting when the results might differ. The approach is smart because the approximate restart will rarely be triggered. [Gharachorloo, Gupta, and Hennessy 1992].

Hill et al., in 1998, advocated the grouping of sequential or processor consistency together with temporary execution as the consistency model of choice. His argument has three parts. First, an aggressive implementation of either sequential consistency or processor consistency will gain most of the advantage of a more relaxed model. Second, such an implementation adds very little to the implementation cost of a speculative processor. Third, such an approach permits the programmer to reason to use the simpler programming models of either sequential or processor consistency.

Check Your Progress

1. What is synchronization?
2. What is the main idea of relaxed consistency model?

9.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Synchronization refers to a mechanism that deals with two or more concurrent processes or threads do not execute on some particular program segment simultaneously.
2. The main idea in relaxed consistency model is that it allow reads and writes to complete out of order, but to use synchronization operations to enforce ordering, so that a synchronized program behaves as if the processor were sequentially consistent.
9.6 SUMMARY

- Synchronization refers to a mechanism that deals with two or more concurrent processes or threads do not execute on some particular program segment simultaneously. Basically that program segment referred as critical section.

- Consistency models are used in distributed shared memory systems. The system supports a given model if it follows some specific rules while performing operations on memory. A model acts as a contract between the programmer and the system. The system guarantees that the memory will be consistent if the programmer follows the rules and results of reading, writing, or updating memory will be predictable.

- The main idea in relaxed consistency model is that it allow reads and writes to complete out of order, but to use synchronization operations to enforce ordering, so that a synchronized program behaves as if the processor were sequentially consistent.

9.7 KEY WORDS

- **Synchronization**: It is the coordination of events to operate a system while sharing a common resource.

- **Critical Section**: The portion of the code of a process in which it accesses or changes the shared data.

9.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

**Short Answer Questions**

1. What are the issues and need of synchronization?
2. What do you understand by memory consistency?
3. What are the three sets of relaxed orderings?

**Long Answer Questions**

1. Explain the relaxed consistency model.
2. Describe the cross cutting issues in synchronization.
9.9 FURTHER READINGS


UNIT 10 INTRODUCTION TO MEMORY ORGANIZATION

Structure
10.0 Introduction
10.1 Objectives
10.2 Memory Hierarchy Organization
10.3 Main Memory
  10.3.1 Random Access Memory (RAM)
  10.3.2 Read Only Memory (ROM)
10.4 Organization of Ram and Rom Chips
  10.4.1 Memory Address Map
  10.4.2 Memory Connection to CPU
10.5 Auxiliary Memory
10.6 Associative Memory
10.7 Cache Memory
  10.7.1 Basic Operation
  10.7.2 Performance
10.8 Mapping Process
  10.8.1 Associative Mapping
  10.8.2 Direct Mapping
  10.8.3 Set Associative Mapping
  10.8.4 Writing into Cache
10.9 Optimization of Cache Performance, Memory Technology and Optimization
10.10 Answers to Check Your Progress Questions
10.11 Summary
10.12 Key Words
10.13 Self Assessment Questions and Exercises
10.14 Further Readings

10.0 INTRODUCTION

In this unit, you will learn about organization of memory and different types of peripheral devices that can be attached to a computer system for input/output of data. An electronic circuitry that allows data to be stored and retrieved when required is known as memory. Memory is the integral part of any computer system. It is a storage device which stores the set of instructions, i.e., program, data and the intermediate results of the computations. The memory hierarchy consists of the total memory system of any computer. The memory components range from higher
capacity to a relatively fast that can be accessible to the high speed processing logic. You will learn about the hierarchy of memory depending upon certain factors, such as access time, storage capacity and cost. You will also learn about the main memory and its organization in the computer system. Main memory is the memory unit that communicates directly with the CPU. It is relatively large and fast and is basically used to store programs and data during computer operation. Main memory is classified as RAM (Random Access Memory) and ROM (Read Only Memory) which can be further classified. You will learn about the memory address map, a pictorial representation of assigned address space for each RAM and ROM chip in the computer system, and the connection of memory to the CPU (Central Processing Unit). The CPU connects the RAM and ROM chips through the data and address buses.

You will also learn about the different storage devices that provide backup storage called as auxiliary memory. The most common auxiliary devices used in computer devices are magnetic disks, tapes and optical disks. You will learn about the architecture and mechanism of these devices to store data. The associative memory which is a memory unit accessed by content and its hardware organization in the computer system is explained with the help of examples and illustrations. You will also be acquainted about the optimization of cache performance and memory technology.

10.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the hierarchy of memory in a computer system
- Classify the main memory that directly communicates with CPU
- Describe the organization of RAM and ROM chips in the CPU
- Explain the memory address map and memory connection to CPU
- Recognize the different types of memories
- Understand the various storage devices used in computer system
- Discuss the hardware organization of associative memory
- Evaluate the basic operation and performance of cache memory
- Explain the optimization of cache performance

10.2 MEMORY HIERARCHY ORGANIZATION

An electronic circuitry that allows data to be stored and retrieved when required is known as memory. Memory is the integral part of any computer system. It is a storage device. It stores the set of instructions, i.e. program, data and the intermediate results of the computations. Memory is basically classified into two
categories called Main (or Primary) Memory and Auxiliary (or Secondary) Memory. The memory unit that communicates directly with the CPU is called the main memory and the memory that provides the backup storage is called auxiliary memory.

The memory hierarchy consists of the total memory system of any computer. The memory components range from higher capacity slow auxiliary memory to a relatively fast main memory to cache memory that can be accessible to the high speed processing logic. A five level memory hierarchy is shown in Figure 10.1.

At the top of this hierarchy, there is a CPU register, which is accessed at full CPU speed. This is local memory to the CPU as the CPU requires it. Next comes cache memory, which is currently on the order of 32 KB to few megabytes. After that is the main memory with sizes currently ranging from 16 MB for an entry level system to few gigabytes at the other end. Next are magnetic disks and finally we have magnetic tape and optical tapes.

The memory, as we move down the hierarchy, mainly depends on the following three key parameters:

- **Access time**
- **Storage capacity**
- **Cost**

**Access Time:** CPU registers are the CPU’s local memory and are accessed in a few nanoseconds. Cache memory takes a small multiple of CPU registers. Main memory access time is typically few tens of nanoseconds.

Now comes a big gap, as disk access time are at least 10 msec, and tapes and optical disk access may be measured in seconds if the media is to be fetched and inserted into a drive.

**Storage Capacity:** The storage capacity increases as we go down the hierarchy. CPU registers are good for 128 bytes. Cache memories are a few megabytes. The main memory are 10 to thousands megabytes. Magnetic disk of capacities are few gigabytes to tens of gigabytes. The capacity of tapes and optical disks are limited as they are usually kept offline.

---

**Fig. 10.1 Five Level Memory Hierarchy**
Another way of viewing the memory hierarchy in any computer system is illustrated in Figure 10.2. The main memory is at the central place as it can communicate directly with the CPU and through the I/O processor with the auxiliary devices. Cache memory is placed in between the CPU and the main memory. Cache usually stores the program segments currently being executed in the CPU and temporary data frequently asked by the CPU in the present calculations. The I/O processor manages the data transfer between the auxiliary memory and the main memory. The auxiliary memory has usually a large storing capacity but has low access rate as compared to the main memory and hence, is relatively inexpensive. Cache is very small but has very high access speed and is relatively expensive. Thus, we can say that

\[ \text{Access speed} \propto \text{Cost} \]

Thus, the overall goal of using a memory hierarchy is to obtain the highest possible average speed while minimizing the total cost of the entire memory system.

**10.3 MAIN MEMORY**

The memory unit that communicates directly with the CPU is called main memory. It is relatively large and fast and is basically used to store programs and data during computer operation. The main memory can be classified into the following two categories:

**10.3.1 Random Access Memory (RAM)**

The term, Random Access Memory (RAM), is basically applied to the memory system that is easily read from and written to by the processor. For a memory to be random access means that any address can be accessed at any time, i.e., any memory location can be accessed in a random manner without going through any other memory location. The access search time for each memory location is same.

In general, RAM is the main memory of a computer system. Its purpose is to store data and applications that are currently in use by the processor. The operating system controls the use of RAM memory by taking different decisions, such as when items are to be loaded into RAM, at what memory location items are to be loaded in RAM and when they need to be removed from RAM. RAM is very fast memory, both for reading and writing data. Hence, a user can write...
information into RAM and can also read information from it. Information written in it is retained as long as the power supply is on. All stored information in RAM is lost as soon as the power is off.

The two main classifications of RAM are Static RAM (SRAM) and Dynamic RAM (DRAM).

**Static RAM (SRAM)**

Static RAM is made from an array of flip-flops, where each flip-flop maintains a single bit of data within a single memory address or location.

Static RAM is a type of RAM that holds its data without external refresh as long as power is supplied to the circuit. The word ‘static’ indicates that the memory retains its content as long as power is applied to the circuit. In general, SRAM

- is a type of semiconductor memory.
- does not require any external refresh circuitry in order to keep data intact.
- is used for high speed registers, caches and small memory banks, such as router buffers.
- has access time in the range of 10 to 30 nanoseconds and hence allows for very fast access.
- is very expensive.

**Dynamic RAM (DRAM)**

Dynamic RAM is a type of RAM that only holds its data if it is continuously accessed by special logic called refresh circuit. This circuitry reads the contents of each memory cell many hundreds of times per second to find out whether the memory cell is being used at that time by computer or not. Due to the way in which the memory cells are constructed, the reading action itself refreshes the contents of the memory. If this is not done regularly, then DRAM will lose its contents even if it continues to have power supplied to it. Because of this refreshing action, the memory is called dynamic. In general, DRAM

- is the most common type of memory in use. All PCs use DRAM for their main memory system instead of SRAM.
- have much higher capacity.
- are much cheaper than SRAM.
- are slower than SRAM because of the overhead of the refresh circuitry.

### 10.3.2 Read Only Memory (ROM)

In every computer system, there is a portion of memory that is stable and impervious to power loss. This type of memory is called Read Only Memory or in short ROM. It is non-volatile memory, i.e., information stored in it is not lost even if the power supply goes off. It is used for permanent storage of information and it possesses random access property.
The most common application of ROM is to store the computer’s BIOS (Basic Input/Output System). Since the BIOS is the code that tells the processors to access its resources on powering up the system. Another application is the code for embedded systems.

There are different types of ROMs. They are as follows:

- **PROM (Programmable Read Only Memory)**: Data is written into a ROM at the time of manufacture. However, the contents can be programmed by a user with a special PROM programmer. PROM provides flexible and economical storage for fixed programs and data.

- **EPROM (Erasable Programmable Read Only Memory)**: This allows the programmer to erase the contents of the ROM and reprogram it. The contents of EPROM cells can be erased using ultraviolet light using an EPROM programmer. This type of ROM provides more flexibility than ROM during the development of digital systems. Since they are able to retain the stored information for longer duration, any change can be easily made.

- **EEPROM (Electrically Erasable Programmable Read Only Memory)**: In this type of ROM, the contents of the cell can be erased electrically by applying a high voltage. EEPROM need not be removed physically for reprogramming.

### 10.4 ORGANIZATION OF RAM AND ROM CHIPS

A RAM chip is best suited for communication with the CPU if it has one or more control lines to select the chip only when needed. It has a bidirectional data bus that allows the transfer of data either from the memory to the CPU during a read operation or from the CPU to the memory during a write operation. This bidirectional bus can be constructed using a three-state buffer. A three-state buffer has the following three possible states:

- Logic 1
- Logic 0
- High impedance

The high impedance state behaves like an open circuit which means that the output does not carry any signal. It leads to very high resistance and hence no current flows.

![Fig. 10.3 Block Diagram of RAM Chip](image-url)
Figure 10.3 shows the block diagram of a RAM chip. The capacity of memory is 128 words of eight bits per word. This requires a 7-bit address and an 8-bit bidirectional data bus.

RD and WR are read and write inputs that specify the memory operations during read and write, respectively. Two Chip Select (CS) control inputs are for enabling the particular chip only when it is selected by the microprocessor. The operation of the RAM chip will be according to the function table as shown in Table 10.1. The unit is in operation only when $CS_1 = 1$ and $CS_2 = 0$. The bar on top of the second chip select indicates that this input is enabled when it is equal to 0.

Table 10.1 Function Table

<table>
<thead>
<tr>
<th>CS1</th>
<th>CS2</th>
<th>RD</th>
<th>WR</th>
<th>Memory function</th>
<th>State of data bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>×</td>
<td>×</td>
<td>Inhibit</td>
<td>High impedance</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>×</td>
<td>×</td>
<td>Inhibit</td>
<td>High impedance</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Inhibit</td>
<td>High impedance</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Write</td>
<td>Input data to RAM</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>×</td>
<td>×</td>
<td>Read</td>
<td>Output data from RAM</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>×</td>
<td>×</td>
<td>Inhibit</td>
<td>High impedance</td>
</tr>
</tbody>
</table>

Thus, if the chip select inputs are not enabled, or if they are enabled but read and write lines are not enabled, the memory is inhibited and its data bus is in high impedance. When chip select inputs are enabled, i.e. $CS_1 = 1$ and $CS_2 = 0$, the memory can be in the read or write mode. When the write WR input is enabled, a byte is transferred from the data bus to the memory location specified by the address lines. When the read RD input is enabled, a byte from the specified memory location by the address line is placed into the data bus.

A ROM chip is organized in the same way as a RAM chip. The block diagram of a ROM chip is shown in Figure 10.4.

![Block Diagram of a ROM Chip](image)

**Fig. 10.4 Block Diagram of a ROM Chip**

The two chip select lines must be $CS_1 = 1$ and $CS_2 = 0$ for the unit to be operational. Otherwise, the data bus is in high impedance. There is no need for the read and write input control because the unit can only read. Thus, when the chip is selected, the byte selected by the address line appears to be in the data bus.
10.4.1 Memory Address Map

A table called a memory address map is a pictorial representation of the assigned address space for each chip in the system.

The interconnection between the memory and the processor is established from the knowledge of the size of memory required and the types of RAM and ROM chips available. RAM and ROM chips are available in a variety of sizes. If a memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to get the required memory size. If the required size of the memory is $M \times N$ and the chip capacity is $m \times n$, then the number of chips required can be calculated as:

$$k = \frac{M \times N}{m \times n}$$

Suppose, a computer system needs 512 bytes of RAM and 512 bytes of ROM. The capacity of RAM chip is $128 \times 8$ and that of ROM is $512 \times 8$. Hence, the number of RAM chips required will be:

$$k = \frac{512 \times 8}{128 \times 8} = 4$$ RAM chips

One ROM chip will be required by the computer system. The memory address map for the system is illustrated in Table 10.2. Table 10.2 consists of three columns. The first column specifies whether a RAM or a ROM chip is used. The next column specifies a range of hexadecimal addresses for each chip. The third column lists the address bus lines.

<table>
<thead>
<tr>
<th>Component</th>
<th>Hexadecimal Address</th>
<th>Address Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM 1</td>
<td>0000-007F</td>
<td>0 0 0 × × × × × × ×</td>
</tr>
<tr>
<td>RAM 2</td>
<td>0080-00FF</td>
<td>0 0 1 × × × × × × ×</td>
</tr>
<tr>
<td>RAM 3</td>
<td>0100-017F</td>
<td>0 1 0 × × × × × × ×</td>
</tr>
<tr>
<td>RAM 4</td>
<td>0180-01FF</td>
<td>0 1 1 × × × × × × ×</td>
</tr>
<tr>
<td>ROM</td>
<td>0200-03FF</td>
<td>1 × × × × × × × × ×</td>
</tr>
</tbody>
</table>

Table 10.2 shows only 10 lines for the address bus although the address bus consists of 16 lines. These six lines are assumed to be zero. The RAM chip has 128 bytes and needs seven address lines, which are common to all four RAM chips. The ROM chip has 512 bytes and needs nine address lines. Thus, ×’s are assigned to the low order bus lines, line 1 to 7 for the RAM chip and lines 1 through 9 for the ROM chip, where these ×’s represent a binary number, which is a combination of all possible 0’s and 1’s values. Also, there must be a way to
distinguish between the four RAM chips. Lines 8 and 9 are used for this purpose. If lines 9 and 8 are 00, it is a RAM1 chip; if it is 01, it is a RAM2 chip; if it is 10, it is RAM3 chip and if the value of lines 9 and 8 is 11, it is RAM4 chip. Also, the distinction between RAM and ROM is required. This distinction is done with the help of line 10. When line 10 is 0, the CPU selects one of the RAM chips, and when line 10 is 1, the CPU selects the ROM chip.

10.4.2 Memory Connection to CPU

The CPU connects the RAM and ROM chips through the data and address buses. Low order lines within the address bus select the byte within the chip and other lines select a particular chip through its chip select lines. The memory chip connection to the CPU is shown in Figure 10.5. Seven low order bits of the address bus are connected directly to each RAM chip to select one of 128 possible bytes.

![Fig. 10.5 Memory Connection to the CPU](image-url)
Lines 8 and 9 are connected as inputs to a $2 \times 4$ decoder, whose outputs are connected to the CS1 input of each RAM chip. Thus, when lines 8 and 9 are 00, the first RAM chip is selected. The RD and WR outputs from the CPU are connected to the inputs of each RAM chip. To distinguish between RAM and ROM, line 10 is used. When line 10 is 0, RAM is selected and when line 10 is 1, ROM is selected. Line 10 is connected to $\overline{CS2}$ of each RAM chip and with the help of an inverter to of the ROM chip. The other chip select input in ROM is connected to the RD control line. Address bus lines 1 to 9 are applied to the input address of ROM. The data bus lines of RAM and ROM are connected to the data bus of the CPU where RAM can transfer information in both directions, whereas ROM has only one output capability.

10.5 AUXILIARY MEMORY

The storage devices that provide backup storage are called auxiliary memory. RAM is a volatile memory and thus a permanent storage media is required in a computer system. Auxiliary memory devices are used in a computer system for the permanent storage of information and hence, are the devices that provide backup storage. They are used for storing system programs, large data files and other backup information. The auxiliary memory has a large storage capacity and is relatively inexpensive, but has low access speed as compared to the main memory. The most common auxiliary memory devices used in computer systems are magnetic disks, floppy disks and tapes. Now optical disks are also used as auxiliary memory.

Magnetic Disk

Magnetic disks are circular metal plates coated with magnetized material on both sides. Several disks are stacked to a spindle one below the other with a read/write head to make a disk pack. The disk drive consists of a motor and all disks rotate together at very high speed. Information is stored on the surface of a disk along concentric sets of rings called tracks. These tracks are divided into sections called sectors. A set of corresponding tracks in all surfaces of a disk pack is called cylinder. Thus, if a disk pack has $n$ plates, there are $2n$ surfaces, hence the number of tracks per cylinder is $2n$. The minimum quantity of information, which can be stored in a sector. If the number of bytes to be stored in a sector is less than the capacity of the sector, the rest of the sector is padded with the last type recorded. Figure 10.6 shows a magnetic disk memory.
Introduction to Memory Organization

NOTES

Self-Instructional Material

Fig. 10.6 Magnetic Disk

The subdivision of a disk surface into tracks and sectors is shown in Figure 10.7.

Fig. 10.7 Surface of a Disk

Suppose $s$ bytes are stored per sector, there are $p$ sectors per track, $t$ tracks per surface and $m$ surfaces. Then, the capacity of disk will be defined as:

$$\text{Capacity} = m \times t \times p \times s \text{ bytes}$$

If $d$ is the diameter of the disk, the density of recording is:

$$\text{Density} = \frac{(p \times s)}{(\pi \times d)} \text{ bytes/inch}$$

A set of disk drives are connected to a disk controller. The disk controller accepts commands and positions the read/write heads for reading or writing. When the read/write command is received by the disk controller, the controller first
positions the arm so that the read/write head reaches the appropriate cylinder. The time taken to reach the appropriate cylinder is known as Seek time ($T_s$). The maximum seek time is the time taken by the head to reach the innermost cylinder from the outermost cylinder or vice versa. The minimum seek time will be 0 if the head is already positioned on the appropriate cylinder. Once the head is positioned on the cylinder, there is further delay because the read/write head has to be positioned on the appropriate sector. This is rotational delay also known as Latency time ($T_l$). The average rotational delay equals half the time taken by the disk to complete one rotation.

**Floppy Disk**

A floppy disk, also known as diskette, is a very convenient bulk storage device and can be taken out of the computer. It can be either 5.25" or 3.5" size, the 3.5" size being more common. It is contained in a rigid plastic case. The read/write heads of the disk drive can write or read information from both sides of the disk. The storage of data is in the magnetic form, similar to that in hard disk. The 3.5" floppy disk has storage up to 1.44 Mbytes. It has a hole in the centre for mounting it on the drive. Data on the floppy disk is organized during the formatting process. The disk is organized into sectors and tracks. The 3.5" high density disk has 80 concentric circles called tracks and each track is divided into 18 sectors. Tracks and circles exist on both sides of the disk. Each sector can hold 512 bytes of data plus other information like address, etc. It is a cheap read/write bulk storage device.

**Magnetic Tapes**

Magnetic disk is used by almost all computer system as a permanent storage device; however, magnetic tape is still a popular form of low cost magnetic storage media and it is primarily used for backup storage purposes. The standard backup magnetic tape device used today is Digital Audio Tape (DAT). These tapes provide approximately 1.2 Gbytes of storage on a standard cartridge size cassette tape. These magnetic tapes memories are similar to that of audio tape recorders.

A magnetic tape drive consists of two spools on which the tape is wound. Between the two spools, there is a set of nine magnetic heads to write and read information on the tape. The nine heads operate independently and record information on nine parallel tracks, parallel to the edge of the tape. Eight tracks are used to record a byte of data and the ninth track is used to record a parity bit for each byte. The standard width of the tape is half an inch. The number of bits per inch (bpi) is known as recording density.

Normally, when data is recorded into the tape, a block of data is recorded and then a gap is left and then another block is recorded and so on. This gap is known as Inter Block Gap (IBG). The blocks are normally 10 times long as that of IBG. The Beginning Of the Tape (BOT) is indicated by a metal foil known as marker and the End Of Tape (EOT) is also indicated by a metal foil known as end of tape marker.
The data on the tape is arranged as blocks and cannot be addressed. They can only be retrieved sequentially in the same order in which they are written. Thus, if a desired record is at the end of the tape, earlier records have to be read before it is reached and hence, the access time is very high as compared to magnetic disks.

**Optical Disks**

Optical disk storage technology provides the advantage of high volume and economical storage with somewhat slower access times than traditional magnetic disk storage.

**CD-ROM**

Compact Disk-Read Only Memory (CD-ROM) optical drives are used for the storage of information that is distributed for read only use. A single CD-ROM can hold up to 800 MB of information. Software and large reports distributed to a large number of users are good candidates for this media. CD-ROM is also more reliable for distribution than floppy disks or tapes. Nowadays, almost all software and documentations are distributed only on CD-ROM.

In CD-ROMs the information is stored evenly across the disk in segments of the same size. Therefore, in CD-ROMs, data stored on a track increases as we go towards the outer surface of disk and hence, CD-ROMs are rotated at variable speeds for the reading process.

Information in a CD-ROM is written by creating pits on the disk surface by shining a laser beam. As the disk rotates, the laser beam traces out a continuous spiral. When 1 is to be written on the disk, a circular pit of around 0.8 micrometer diameter is created by the sharply focused beam and no pit is created if a zero is to be written. The pre recorded information on the CD-ROM is read with the help of a CD-ROM reader, which uses a laser beam for reading. For this, the CD-ROM disk is inserted into a slot of CD drive. Then the disk is rotated by a motor. A laser head moves in and out to the specified position. As the disk rotates, the head senses pits and land, which is converted to 1’s and 0’s by the electronic interface and sent to the computer. Figure 10.8 depicts a CD-ROM.

*Fig. 10.8 Tracks on a Disk Surface*
The speed of the disk is indicated by \( n \times \), where \( n \) is an integer indicating the factor by which the original nominal speed of 150 Kb/s is multiplied. Thus, a \( 52 \times \) CD-ROM disk speed will be \( 52 \times 150 = 7800 \) Kb/s. CD-ROM has a buffer size of 256 Kilobytes to keep data temporarily. It is connected to the computer system by a Small Computer System Interface (SCSI) adapter.

The main advantages of CD-ROMs are:
- Large data/information storage capacity.
- Mass replication is inexpensive and fast.
- These are removable disks.

Disadvantages of CD-ROMs are:
- It is read only and hence cannot be updated.
- Access time is longer than that of a magnetic disk.

Erasable Optical Disk

Recent development in optical disks is the erasable optical disks. They are used as an alternative to standard magnetic disks when speed of the access is not important and the volume of the data stored is large. They can be used for image, multimedia, a high volume and low activity backup storage. Data in these disks can be changed as repeatedly as in a magnetic disk. The erasable optical disks are portable and highly reliable and have longer life. They use format that makes semi random access feasible.

Check Your Progress

1. Define the term memory.
2. What decisions did an operating system take when controls the use of RAM memory?
3. What are the common applications of ROM?
4. What are the possible states of three-state buffer?
5. How the interconnection between the memory and the processor is established?
6. Why auxiliary memory devices are used in a computer system?
7. Define inter block gap.

10.6 ASSOCIATIVE MEMORY

A memory unit accessed by content is called associative memory or Content Addressable Memory (CAM).
CAM is a special type of computer memory used in certain very high speed searching applications. Associative memories are typically described as a collection of elements which have data storage capabilities and which can be assessed simultaneously in parallel or on the basis of data stored rather than by specified address or location. CAM is frequently used in computer networking devices, for example when a network switch receives a data frame from one of its ports then it updates an internal table with the frame’s source MAC address and the port which has received it. It then checks up the destination MAC address in the table to determine which port the frame must be forwarded to and sends it to that port.

When a word is written on associative memory, no address is given. The memory is capable of finding an empty location to store the word. When a word is to be read from associative memory, the content of the word, or part of the word, is specified. The memory locates all words which match the specified content and marks them for reading.

This type of memory is accessed simultaneously and in parallel on the basis of data rather than by specific address or location.

Associative memory is more expensive than random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument.

Thus, associative memory is used in application where the search time is very critical and short.

There are two types of associative memory, namely auto-associative and hetero-associative. An auto-associative memory retrieves a previously stored pattern that most closely resembles the current pattern whereas in a hetero-associative memory the retrieved pattern is different from the input pattern not only in content but also in type and format. Typically, associative memory is a system that associates two patterns \((x, y)\) such that when one is encountered then the other can be recalled. The two types of associations, for example for two patterns \(x\) and \(y\) can be represented as follows:

- Auto-association \((x = y)\): Relating parts of a pattern with other parts.
- Hetero-association \((x\neq y)\): Relating two different patterns.

Associative memory enables a parallel search within a stored data. The objective of search is to output one or all stored data items that match the search argument and retrieve it entirely or partially. Typically, the associative memory is a model which works on the objective that given an input pattern it establishes the most analogous patterns. Since, associative memory is memory that is addressed through its contents hence it has its implementation in various fields. Whenever a pattern is accessed through an associative memory then it checks whether this pattern coincides with a stored pattern or not. An associative memory may possibly return a stored pattern specifically analogous to the one presented so that noisy input can also be recognized. Hopfield networks are a special type of recurrent neural
networks that can be employed as associative memory. Hopfield networks are used as associative memory by utilizing the property of possessing stable states.

A Hopfield network was invented by John Hopfield and is considered as a type of recurrent artificial neural network. Hopfield nets usually function as content addressable memory systems along with binary threshold units, i.e., the units only accept two different values for their states. The value is determined on the basis whether the units’ input exceeds their threshold or not. Hopfield nets have units that can accept values of 1 or 1 and units that accept values of 1 or 0. As a result, the two possible definitions for unit i’s activation, a_i, are given as follows:

(1) \[ a_i \begin{cases} 1 & \text{if } \sum_j w_{ij} s_j > \theta_i, \\ -1 & \text{otherwise}. \end{cases} \]

(2) \[ a_i \begin{cases} 1 & \text{if } \sum_j w_{ij} s_j > \theta_i, \\ 0 & \text{otherwise}. \end{cases} \]

Where:

- \( w_{ij} \) - Refers to the strength of the connection weight from unit j to unit i, i.e., the weight of the connection.
- \( s_j \) - Refers to the state of unit j.
- \( \theta_i \) - Refers to the threshold of unit i.

Typically, the connections in a Hopfield net hold the following limitations:

- \( w_{ii} = 0, \forall i \) - This specifies that no unit has a connection with itself.
- \( w_{ij} = w_{ji}, \forall i, j \) - This specifies that connections are symmetric in nature.

Example of Auto-Associative Memory

Auto-associative memory can be illustrated by considering memory vectors \( y[1], y[2], y[3], \ldots, y[n] \) as the number of stored pattern vectors. If \( y[n] \) is considered as the components of these vectors which represent features extracted from these patterns then the auto-associative memory will output a pattern vector \( y[n] \) when inputting a noisy or incomplete version of \( y[n] \).

Example of Hetero-Associative Memory

BAM or Bidirectional Associative Memory is hetero-associative memory which implies that given a pattern it might return another pattern potentially of a different size. It is similar to the Hopfield network because both are types of associative memory. However, Hopfield nets return patterns of the same size.

Hardware Organization of Associative Memory

Figure 10.9 shows the block diagram of associative memory. It consists of a memory array and logic for \( n \) words with \( n \) bits per words. The argument register \( A \) and the key register \( K \) each have \( n \) bits, one for each bit of a word. The match register
$M$ has $m$ bits, one for each memory word. Each word in the memory is compared in parallel with the content of the argument register. The words that match the bits of the argument register set the corresponding bit in the match register. After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched.

Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

The key register provides a mask for choosing a particular field or key in the argument register. The entire argument is compared with each memory word if the key register contains all 1s. Otherwise, only those bits in the argument that have 1s in the corresponding position of the key register are compared.

**Fig. 10.9** Block Diagram of Associative Memory

Suppose,  

- $A = 11000110$
- $K = 11100000$
- Word 1 $= 10100110$ no match
- Word 2 $= 11000001$ match

Only the three leftmost bits of $A$ are compared with memory words because $K$ has 1’s in these positions. The three leftmost bits of argument and word 2 are equal. Hence, Word 2 matches with the unmasked argument field.

Figure 10.10 shows how the external registers are associated with the memory array of associative memory. The cells in the array are marked by letter $C$ with two subscripts. The first subscript gives the word number and the second specifies the bit positions in the word. Thus, cell $C_{ij}$ is the cell for bit $j$ in the word $i$. A bit $A_j$ in the argument register is compared with all bits in the column $j$ of the array provided that $K_j = 1$.

If the match occurs between all unmasked bits of the argument register and the bits in word $i$, the corresponding bit $M_i$ in the match register is set to 1. If any unmasked bit of the argument does not match with word, the corresponding bit in the match register is cleared to 0.
10.7 CACHE MEMORY

Cache memory is defined as a very high speed memory that is used in a computer system to compensate the speed differential between the main memory access time and the processor logic.

A very high speed memory called cache is used to increase the speed of processing by making the current programs and data available to the CPU at a rapid rate. It is placed between the CPU and the main memory as shown in Figure 10.11. The cache memory access time is less than the access time of the main memory by a factor of 5 to 10. This cache memory is employed in computer systems to compensate the speed differential between the main memory access time and the processor logic. The cache is used for storing program segments currently being executed in the CPU and for the data frequently used in the present calculations. By making programs and data available at a rapid rate, it is possible to increase the performance rate of a computer.

**Fig. 10.11 Cache Memory Placement**

10.7.1 Basic Operation

Whenever the CPU needs to access memory, the cache memory is examined first. If the word generated by the CPU is found in the cache, the word is read from the memory. But if the word addressed by the CPU is not found in the cache, the main memory is searched to read the word. If the word is found in the main memory, a block of words containing the one just accessed is then transferred from the main memory to the cache memory. This block size may vary from one word (the one...
just accessed) to many words (but in the form of $2^n$) adjacent to the one just accessed.

### 10.7.2 Performance

The performance of cache memory is frequently measured in terms of a quantity called *hit ratio*. When the CPU refers to any word to the memory and finds the word in the cache memory, it is said to produce a hit. If the word referred by the CPU is not found in the cache memory, it is in the main memory and counts as a miss. Thus, the ratio of the total number of hits divided by the total CPU references to memory (number of hits + number of miss) is referred to as hit ratio.

Let $t_c$ be the cache access time.

$h$ is the hit ratio.

$t_m$ is the main memory access time.

Then, the average access time is given by relation:

$$\text{Average access time} = h t_c + (1 - h)(t_c + t_m)$$

The hit ratio $h$ always lies in the closed interval of 0 and 1.

The equation $\text{Average access time} = h t_c + (1 - h)(t_c + t_m)$ is derived using the fact that when there is a cache hit, the main memory is not accessed, and in the event of a cache miss, both the main memory and cache will be accessed.

Suppose, the ratio of main memory access time to cache access time is $\gamma$, then an expression for the efficiency of a system that employs a cache can be derived as,

$$A = \frac{t_c}{t}$$

$$= \frac{t_c}{ht_c + (1 - h)(t_c + t_m)}$$

$$= \frac{t_c}{t_c \left( h + (1 - h) \left( 1 + \frac{t_m}{t_c} \right) \right)}$$

$$= \frac{1}{h + (1 - h)(1 + \gamma)} \quad \because \gamma = \frac{t_m}{t_c}$$

$$= \frac{1}{h + 1 - h + \gamma(1 - h)}$$

$$= \frac{1}{1 + \gamma(1 - h)}$$
10.8 MAPPING PROCESS

The transformation of data from the main memory to the cache memory is referred to as the mapping process.

A CPU is connected to the main memory via the cache memory as shown in Figure 10.12. In order to fetch a word, the CPU will send an $n$-bit address to the cache memory. If there is a hit, the word is fetched from the cache and if there is a miss, the required word is searched in the main memory and then copied from the main memory into the cache.

![Fig. 10.12 Mapping Process](image)

There are three different mapping procedures while considering the organization of cache memory. These mapping methods are as follows:

- Associative mapping
- Direct mapping
- Set associative mapping

10.8.1 Associative Mapping

In case of associative mapping, the contents of cache memory are not associated with any address. Data stored in the cache memory are not accessed by specifying any address. Instead, data or parts of data are searched by matching with the contents.

In the associative mapping method, both the word and the address of the word (in the main memory) are stored in the cache as shown in Figure 10.13. The address bits, sent by the CPU to search, are matched with the addresses stored in the cache memory. If any address is matched, the corresponding word is fetched from the cache and sent to the CPU.

If no match is found in the cache memory, the word is searched in the main memory. The word along with address is then copied from the main memory into the cache. If the cache is full, then the existing word along with its address must be removed to make room for the new word.

Associative mapping has the advantage that it is a very fast access method, but it has the disadvantage that it is very expensive and complicated because of
the complex logical circuits that are required to implement data searching by content and not by address.

Due to the high cost associated with logic circuits required to implement associative mapping, other methods are used in which data in the cache memory is accessed by address, like direct mapping and set associative mapping.

10.8.2 Direct Mapping

Suppose a computer has 4K main memory, i.e., $4 \times 1024$ bytes and 1K cache memory. To address a word in the main memory, a 12-bit ($4K = 2^{12}$) address is required. Similarly, to address a word in the cache memory a 10-bit ($1K = 2^{10}$) address is required. Thus, a cache memory needs 10 bits to address a word and the main memory requires 12 bits to address a word. In the direct mapping method, the 12-bit address sent by the CPU is divided into two parts called tag field and index field. The index field has the number of bits equal to the number of bits required to address a word in the cache.

Thus, if a computer has the main memory of capacity $2^m$ and the cache memory of $2^n$, then the address bits will be divided into $n$ bits index field and $(m - n)$ bits of tag field.

The tag field for the above mentioned computer system will be of 2 bits and index field will have 10 bits.

In a direct mapping method, the cache memory stores the word as well as the tag field. The words will be stored at that location in the cache which is represented by the index fields of their addresses as shown in Figure 10.14.

When an address is sent by the CPU, the index part of the address is used to get a memory location in the cache. If the tag stored at that location matches the
When a word needs to be moved into the cache memory from the main memory, its address in the main memory is divided into the index and tag fields.

The disadvantage of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly.

**10.8.3 Set Associative Mapping**

The disadvantage of direct mapping is that two words with the same index but different tags cannot be stored into the cache at the same time. As an improvement to this disadvantage of direct mapping, a third type of cache organization called set associative mapping is used. In this mapping process, each word of a cache can store two or more words of the main memory under the same index address.

Each data word is stored along with its tag and the number of tag data pair in one word of cache is said to form a set. An example of set associative cache organization with set size of two is shown in Figure 10.15.

**Fig. 10.15 Two-Way Set Associative Mapping**
The words stored at addresses 000000110010 and 010000110010 of the main memory are stored in the cache memory at the index address 0000110010. Similarly, the words stored at addresses 010000111011 and 100000111011 of the main memory are stored in the cache memory at the index address 0000111011. When the CPU generates a memory request, the word is searched into the cache with the help of index addresses.

10.8.4 Writing into Cache

Whenever the desired information is not found in the cache, the data is retrieved from the main memory and a block of data is transferred from the main memory to the cache memory. The point of concern is that when the CPU alters the content of cache, it is necessary to update its main memory copy also. This can be done in the following two different ways:

- Write-through
- Write-back

Write-through: In the write-through method, whenever the CPU updates the cache content, the same update is made in the main memory copy of the same address immediately. This method has the advantage that the main memory always contains the same data as that of the cache.

Write-back: In the write-back method, whenever the CPU writes something into a cache block, that block is marked as flag. When a block that is marked as a flag is to be replaced by a new block, the flag marked block is copied back into the main memory before it is overwritten by the new block.

10.9 OPTIMIZATION OF CACHE PERFORMANCE, MEMORY TECHNOLOGY AND OPTIMIZATION

The average memory access time formula gives us three metrics for cache optimizations: hit time, miss rate, and miss penalty. Considering the recent trends, we add cache bandwidth and power consumption to this list. We can classify the ten advanced cache optimizations that we examine into five categories based on these metrics:

1. Reducing the hit time: Small and simple first-level caches and way-prediction. Both techniques also generally decrease power consumption.

2. Increasing cache bandwidth: Pipelined caches, multibanked caches, and nonblocking caches. These techniques have varying impacts on power consumption.

3. Reducing the miss penalty: Critical word first and merging write buffers. These optimizations have little impact on power.
4. **Reducing the miss rate**: Compiler optimizations. Obviously any improvement at compile time improves power consumption.

5. **Reducing the miss penalty or miss rate via parallelism**: Hardware prefetching and compiler prefetching. These optimizations generally increase power consumption, primarily due to prefetched data that are unused.

In general, the hardware complexity increases as we go through these optimizations. In addition, several of the optimizations require sophisticated compiler technology.

**First Optimization: Small and Simple First-Level Caches to Reduce Hit Time and Power**

The pressure of both a fast clock cycle and power limitations encourages limited size for first-level caches. Similarly, use of lower levels of associativity can reduce both hit time and power, although such trade-offs are more complex than those involving size.

The critical timing path in a cache hit is the three-step process of addressing the tag memory using the index portion of the address, comparing the read tag value to the address, and setting the multiplexor to choose the correct data item if the cache is set associative. Direct-mapped caches can overlap the tag check with the transmission of the data, effectively reducing hit time. Furthermore, lower levels of associativity will usually reduce power because fewer cache lines must be accessed.

Although the total amount of on-chip cache has increased dramatically with new generations of microprocessors, due to the clock rate impact arising from a larger L1 cache, the size of the L1 caches has recently increased either slightly or not at all. In many recent processors, designers have opted for more associativity rather than larger caches. An additional consideration in choosing the associativity is the possibility of eliminating address aliases.

**Second Optimization: Way Prediction to Reduce Hit Time**

Another approach reduces conflict misses and yet maintains the hit speed of direct-mapped cache. In way prediction, extra bits are kept in the cache to predict the way, or block within the set of the next cache access. This prediction means the multiplexor is set early to select the desired block, and only a single tag comparison is performed that clock cycle in parallel with reading the cache data. A miss results in checking the other blocks for matches in the next clock cycle.

Added to each block of a cache are block predictor bits. The bits select which of the blocks to try on the next cache access. If the predictor is correct, the cache access latency is the fast hit time. If not, it tries the other block, changes the way predictor, and has a latency of one extra clock cycle. Simulations suggest that set prediction accuracy is in excess of 90% for a two-way set associative cache.
and 80% for a four-way set associative cache, with better accuracy on I-caches than D-caches. Way prediction yields lower average memory access time for a two-way set associative cache if it is at least 10% faster, which is quite likely. Way prediction was first used in the MIPS R10000 in the mid-1990s. It is popular in processors that use two-way set associativity and is used in the ARM Cortex-A8 with four-way set associative caches. For very fast processors, it may be challenging to implement the one cycle stall that is critical to keeping the way prediction penalty small.

**Third Optimization: Pipelined Cache Access to Increase Cache Bandwidth**

This optimization is simply to pipeline cache access so that the effective latency of a first-level cache hit can be multiple clock cycles, giving fast clock cycle time and high bandwidth but slow hits. For example, the pipeline for the instruction cache access for Intel Pentium processors in the mid-1990s took 1 clock cycle, for the Pentium Pro through Pentium III in the mid-1990s through 2000 it took 2 clocks, and for the Pentium 4, which became available in 2000, and the current Intel Core i7 it takes 4 clocks. This change increases the number of pipeline stages, leading to a greater penalty on mispredicted branches and more clock cycles between issuing the load and using the data, but it does make it easier to incorporate high degrees of associativity.

**Fourth Optimization: Non-blocking Caches to Increase Cache Bandwidth**

For pipelined computers that allow out-of-order execution, the processor need not stall on a data cache miss. For example, the processor could continue fetching instructions from the instruction cache while waiting for the data cache to return the missing data. A non-blocking cache or lockup-free cache escalates the potential benefits of such a scheme by allowing the data cache to continue to supply cache hits during a miss. This “hit under miss” optimization reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the processor. A subtle and complex option is that the cache may further lower the effective miss penalty if it can overlap multiple misses: a “hit under multiple miss” or “miss under miss” optimization. The second option is beneficial only if the memory system can service multiple misses; most high-performance processors (such as the Intel Core i7) usually support both, while lower end processors, such as the ARM A8, provide only limited non-blocking support in L2.

To examine the effectiveness of non-blocking caches in reducing the cache miss penalty, Farkas and Jouppi [1994] did a study assuming 8 KB caches with a 14-cycle miss penalty. They observed a reduction in the effective miss penalty of 20% for the SPECINT92 benchmarks and 30% for the SPECFP92 benchmarks when allowing one hit under miss.

Li, Chen, Brockman, and Jouppi [2011] recently updated this study to use a multilevel cache, more modern assumptions about miss penalties, and the larger...
and more demanding SPEC2006 benchmarks. The study was done assuming a model based on a single core of an Intel i7 running the SPEC2006 benchmarks.

**Fifth Optimization: Multibanked Caches to Increase Cache Bandwidth**

Rather than treat the cache as a single monolithic block, we can divide it into independent banks that can support simultaneous accesses. Banks were originally used to improve performance of main memory and are now used inside modern DRAM chips as well as with caches. The Arm Cortex-A8 supports one to four banks in its L2 cache; the Intel Core i7 has four banks in L1 (to support up to 2 memory accesses per clock), and the L2 has eight banks.

Clearly, banking works best when the accesses naturally spread themselves across the banks, so the mapping of addresses to banks affects the behavior of the memory system. A simple mapping that works well is to spread the addresses of the block sequentially across the banks, called **sequential interleaving**. For example, if there are four banks, bank 0 has all blocks whose address modulo 4 is 0, bank 1 has all blocks whose address modulo 4 is 1, and so on. Figure 10.16 shows this interleaving. Multiple banks also are a way to reduce power consumption both in caches and DRAM.

![Fig. 10.16 Four-way interleaved cache banks using block addressing.](image)

Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

**Sixth Optimization: Critical Word First and Early Restart to Reduce Miss Penalty**

This technique is based on the observation that the processor normally needs just one word of the block at a time. This strategy is impatience: Don’t wait for the full block to be loaded before sending the requested word and restarting the processor. Here are two specific strategies:

- **Critical word first**: Request the missed word first from memory and send it to the processor as soon as it arrives; let the processor continue execution while filling the rest of the words in the block.
- **Early restart**: Fetch the words in normal order, but as soon as the requested word of the block arrives send it to the processor and let the processor continue execution.
Generally, these techniques only benefit designs with large cache blocks, since the
benefit is low unless blocks are large. Note that caches normally continue to satisfy
accesses to other blocks while the rest of the block is being filled.

Alas, given spatial locality, there is a good chance that the next reference is
to the rest of the block. Just as with nonblocking caches, the miss penalty is not
simple to calculate. When there is a second request in critical word first, the effective
miss penalty is the nonoverlapped time from the reference until the second piece
arrives. The benefits of critical word first and early restart depend on the size of
the block and the likelihood of another access to the portion of the block that has
not yet been fetched.

**Seventh Optimization: Merging Write Buffer to Reduce Miss Penalty**

Write-through caches rely on write buffers, as all stores must be sent to the next
lower level of the hierarchy. Even write-back caches use a simple buffer when a
block is replaced. If the write buffer is empty, the data and the full address are
written in the buffer, and the write is finished from the processor’s perspective; the
processor continues working while the write buffer prepares to write the word to
memory. If the buffer contains other modified blocks, the addresses can be checked
to see if the address of the new data matches the address of a valid write buffer
entry. If so, the new data are combined with that entry. **Write merging** is the name
of this optimization. The Intel Core i7, among many others, uses write merging.

If the buffer is full and there is no address match, the cache (and processor)
must wait until the buffer has an empty entry. This optimization uses the memory
more efficiently since multiword writes are usually faster than writes performed
one word at a time. Skadron and Clark [1997] found that even a merging four-
entry write buffer generated stalls that led to a 5% to 10% performance loss.

**Eighth Optimization: Compiler Optimizations to Reduce Miss Rate**

Thus far, our techniques have required changing the hardware. This next technique
reduces miss rates without any hardware changes.

This magical reduction comes from optimized software - the hardware
designer’s favorite solution! The increasing performance gap between processors
and main memory has inspired compiler writers to scrutinize the memory hierarchy
to see if compile time optimizations can improve performance. Once again, research
is split between improvements in instruction misses and improvements in data misses.
The optimizations presented below are found in many modern compilers.

**Ninth Optimization: Hardware Prefetching of Instructions and Data to
Reduce Miss Penalty or Miss Rate**

Nonblocking caches effectively reduce the miss penalty by overlapping execution
with memory access. Another approach is to prefetch items before the processor
requests them. Both instructions and data can be prefetched, either directly into
the caches or into an external buffer that can be more quickly accessed than main
memory.
Instruction prefetch is frequently done in hardware outside of the cache. Typically, the processor fetches two blocks on a miss: the requested block and the next consecutive block. The requested block is placed in the instruction cache when it returns, and the prefetched block is placed into the instruction stream buffer. If the requested block is present in the instruction stream buffer, the original cache request is canceled, the block is read from the stream buffer, and the next prefetch request is issued.

Tenth Optimization: Compiler-Controlled Prefetching to Reduce Miss Penalty or Miss Rate An alternative to hardware prefetching is for the compiler to insert prefetch instructions to request data before the processor needs it. There are two flavours of prefetch:

- Register prefetch will load the value into a register.
- Cache prefetch loads data only into the cache and not the register.

Either of these can be faulting or nonfaulting; that is, the address does or does not cause an exception for virtual address faults and protection violations. Using this terminology, a normal load instruction could be considered a “faulting register prefetch instruction.” Nonfaulting prefetches simply turn into no-ops if they would normally result in an exception, which is what we want. The most effective prefetch is “semantically invisible” to a program: It doesn’t change the contents of registers and memory, and it cannot cause virtual memory faults. Most processors today offer nonfaulting cache prefetches. This section assumes nonfaulting cache prefetch, also called nonbinding prefetch.

Check Your Progress

8. Why associative memory is more expensive than random access memory?
9. Define cache memory.
10. What are the different mapping procedures while considering the organization of cache memory?
11. What are the three metrics for cache optimizations?

10.10 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. An electronic circuitry that allows data to be stored and retrieved when required is known as memory.

2. The operating system controls the use of RAM memory by taking different decisions, such as when items are to be loaded into RAM, at what memory location items are to be loaded in RAM and when they need to be removed from RAM.
3. The most common application of ROM is to store the computer’s BIOS (Basic Input/Output System). BIOS is the code that tells the processors to access its resources on powering up the system. Another application is the code for embedded systems.

4. A three-state buffer has the following three possible states:
   (i) Logic 1
   (ii) Logic 0
   (iii) High impedance

5. The interconnection between the memory and the processor is established from the knowledge of the size of memory required and the types of RAM and ROM chips available.

6. Auxiliary memory devices are used in a computer system for the permanent storage of information and hence, are the devices that provide backup storage. They are used for storing system programs, large data files and other backup information.

7. When data is recorded into the tape, a block of data is recorded and then a gap is left and then another block is recorded and so on. This gap is known as Inter Block Gap (IBG).

8. Associative memory is more expensive than random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument.

9. Cache memory is defined as a very high speed memory that is used in a computer system to compensate the speed differential between the main memory access time and the processor logic.

10. The three different mapping procedures while considering the organization of cache memory are as follows:
    (i) Associative mapping
    (ii) Direct mapping
    (iii) Set associative mapping

11. The average memory access time formula gives us three metrics for cache optimizations: hit time, miss rate, and miss penalty.

10.11 SUMMARY

- An electronic circuitry that allows data to be stored and retrieved when required is known as memory. Memory is the integral part of any computer system. It is a storage device. It stores the set of instructions, i.e., program, data and the intermediate results of the computations.
The memory hierarchy consists of the total memory system of any computer. The memory components range from higher capacity slow auxiliary memory to a relatively fast main memory to cache memory that can be accessible to the high speed processing logic.

Cache is very small but has very high access speed and is relatively expensive. Consequently, we can say that Access speed $\propto$ Cost. Thus, the overall goal of using a memory hierarchy is to obtain the highest possible average speed while minimizing the total cost of the entire memory system.

The memory unit that communicates directly with the CPU is called main memory. It is relatively large and fast and is basically used to store programs and data during computer operation. The main memory can be classified into the RAM and the ROM.

A RAM chip is best suited for communication with the CPU if it has one or more control lines to select the chip only when needed. It has a bidirectional data bus that allows the transfer of data either from the memory to the CPU during a read operation or from the CPU to the memory during a write operation. This bidirectional bus can be constructed using a three-state buffer.

If a memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to get the required memory size.

The CPU connects the RAM and ROM chips through the data and address buses. Low order lines within the address bus select the byte within the chip and other lines select a particular chip through its chip select lines.

Auxiliary memory devices are used in a computer system for the permanent storage of information and hence, are the devices that provide backup storage. They are used for storing system programs, large data files and other backup information.

Information in a CD-ROM is written by creating pits on the disk surface by shining a laser beam. As the disk rotates, the laser beam traces out a continuous spiral. When 1 is to be written on the disk, a circular pit of around 0.8 micrometer diameter is created by the sharply focused beam and no pit is created if a zero is to be written.

Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set. The key register provides a mask for choosing a particular field or key in the argument register.

A CPU is connected to the main memory via the cache memory. In order to fetch a word, the CPU will send an $n$-bit address to the cache memory.

If there is a hit, the word is fetched from the cache and if there is a miss, the required word is searched in the main memory and then copied from the main memory into the cache.
The average memory access time formula gives us three metrics for cache optimizations: hit time, miss rate, and miss penalty.

10.12 KEY WORDS

- **Memory**: An electronic circuitry that allows data to be stored and retrieved when required.
- **Main Memory**: A memory unit that communicates directly with the CPU.
- **Auxiliary Memory**: The storage devices that provide backup storage.
- **Static RAM (SRAM)**: A type of RAM that holds its data without external refresh as long as power is supplied to the circuit.
- **Dynamic RAM (DRAM)**: A type of RAM that only holds its data if it is continuously accessed by special logic called refresh circuit.
- **Memory Address Map**: A pictorial representation of the assigned address space for each chip in the system.
- **Seek Time**: The time taken to reach the appropriate cylinder in magnetic disk.

10.13 SELF ASSESSMENT QUESTIONS AND EXERCISES

**Short Answer Questions**

1. What is five level hierarchy of memory?
2. On what parameters does hierarchy of memory depends?
3. What are different types of RAM?
4. What is memory address map?
5. What is the most common auxiliary memory devices used in a computer system?
6. How does the CPU access memory?

**Long Answer Questions**

1. Explain the main memory and its types in a computer system.
2. How RAM and ROM chips are organized in the CPU? Explain with the help of suitable diagrams.
3. Describe memory address map and connection of memory in CPU.
4. Discuss the types of auxiliary memory.
5. Explain the hardware organization of associative memory.
6. Discuss the performance of cache memory and derive the expression for it.
7. Describe the transformation of data from the main memory to the cache memory with the help of illustrations and examples.

8. Explain the various methods of mapping process.

9. Describe the various categories of cache optimization.

10.14 FURTHER READINGS


UNIT 11 PROTECTION

11.0 INTRODUCTION

Virtual memory is a technique that allows the execution of processes that may not be completely in the memory. The main advantage of this technique is that programs can be larger than the physical memory. You will learn how to map a virtual address in virtual memory of a computer system using the page replacement algorithm and about the efficient utilization of main memory space using algorithms like FIFO (First in First Out) and LRU (Least Recently Used). The function of a Memory Management Unit (MMU) is to convert logical memory address into actual physical (real) memory address and provide protection. Finally, you will learn about the memory interleaving to improve performance of memory in high end motherboards/chipsets and virtual machines.

11.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the concept of virtual memory
- Understand the various memory management hardware
- Discuss the system pathway in direct memory access
- Understand the prime factors in determining the system performance at the time of managing memory
- Explain the significance of virtual memory and virtual machines
11.2 VIRTUAL MEMORY

Virtual memory is a technique that allows the execution of processes that may not be completely in the memory.

The main advantage of this technique is that programs can be larger than the physical memory. Portions of the program or data are brought into the main memory when they are required by the CPU. Virtual memory is the separation of the user logical memory from the physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. Virtual memory makes the task of programming easy because the programmer no longer needs to worry about the amount of physical memory available. It gives an illusion to the programmers that they have a very large memory at their end, even though the computer actually has small main memory.

An address generated by the user program is called virtual address and the set of virtual addresses makes the virtual address space. A main memory address is called a location or physical address and a set of such locations are called memory space or physical address space. However, in a system that uses virtual memory, the size of virtual address space is usually longer than the available physical address space.

Consider a computer having a main memory capacity of 32 K words. Since $32 \, \text{K} = 2^{15}$, 15 bits will be needed to specify a physical address. Suppose the computer has available auxiliary memory for strong $2^{20}$ words.

Let $N$ be the address space and $M$ be the memory space. Thus, $N = 1024 \, \text{K}$ and $M = 32 \, \text{K}$.

The address bit of the instruction code will consist of 20 bits but the physical memory addresses must be specified using only 15 bits. Thus, the CPU will reference instructions and data with 20-bit address, but the information at this address must be taken from the physical memory rather than the auxiliary memory. Thus, it is required to map a virtual address of 20 bits to a physical address of 15 bits. For this a memory mapping table is needed, which is shown in Figure 11.1. This mapping is a dynamic operation and every address is translated immediately as a word is referenced by the CPU. The mapping table can be stored in the main memory.
Address mapping can be further simplified if the information in the address space and the memory space can be divided into groups of equal size. The address space is broken into groups of equal size known as page and the memory space is broken into groups of same size known as blocks. These blocks can range from 64 to 4096 words. If a page or block consists of 1 K words, then the address space of 1024 K consists of 1024 pages and the memory space of 32 K consists of 32 blocks.

Consider a computer with an address space of 8 K and memory space of 4 K. Thus, if the group is of 1 K words, we have 8 pages and 4 blocks. This division of address space and memory space is shown in Figure 11.2.

![Figure 11.1 Memory Table for Mapping a Virtual Address](image)

Address mapping can be further simplified if the information in the address space and the memory space can be divided into groups of equal size. The address space is broken into groups of equal size known as page and the memory space is broken into groups of same size known as blocks. These blocks can range from 64 to 4096 words. If a page or block consists of 1 K words, then the address space of 1024 K consists of 1024 pages and the memory space of 32 K consists of 32 blocks.

Consider a computer with an address space of 8 K and memory space of 4 K. Thus, if the group is of 1 K words, we have 8 pages and 4 blocks. This division of address space and memory space is shown in Figure 11.2.

![Figure 11.2 Address Space and Memory Space split into a Group of 1 K Words](image)
specify the line address and three high order bits of the virtual address will specify one of the eight pages of the address space. The line address in the address space and the memory space is same and hence, the only mapping required is from a page number to a block number.

The mapping table in a paged system is shown in Figure 11.3. The memory page table consists of eight words. The table address denotes the page number and the content of words gives the block number where that page is stored in the main memory. Thus, this shows that pages 0, 2, 4 and 6 are stored in the main memory in blocks 2, 3, 1 and 0, respectively. The presence bit signifies whether the page is in the main memory or not. If presence bit is 1, the page is available in the main memory and if it is 0, the page is not available in the main memory.

When the CPU references a word in memory with virtual address of 13 bits, the lower order 10 bits specify the line number and three high order bits specify a page number which is also used as an address of the memory page table. Then, the content of the word at this address is read out into the memory table buffer register along with the presence bit. If presence bit is 1, the content thus read (the block number) is transferred into the main memory address register and the line number is also transferred into the main memory address register as 10 lower order bits. A read signal thus transfers the content to the main memory buffer register to be used by the CPU. If the presence bit is 0, the content of the word referenced by the CPU does not reside in the main memory. Then a call to the operating system is generated to fetch the required page from the auxiliary memory to the main memory before resuming computation.
11.2.1 Page Replacement

For the efficient utilization of the main memory space, the memory management software decides certain aspects as follows:

- When a new page is to be brought into the main memory and, if required, which page from the main memory is to be removed.
- Where the page is to be placed in the main memory.
- When a page is to be brought into the main memory from the auxiliary memory.

When a program starts execution, some pages are transferred into the main memory. The page table indicates the position of the pages in the main memory. When an attempt is made to refer to a page that is still in the auxiliary memory, a page fault occurs and the execution of the current program is suspended until the required page is brought into the main memory from the auxiliary memory. Thus, a new page is to be transferred from the auxiliary memory to the main memory, which is basically an I/O operation; hence, this task is assigned to the I/O processor. The control is then transferred to some other program waiting in memory to be processed by the CPU. When the required page is brought into the main memory, the original program can resume its execution. But if the main memory is full, then it would be necessary to remove a page from the memory to make room for the new page. Hence, there are two page transfer, one in the main memory from the auxiliary memory and one out of the main memory to the auxiliary memory. This replacement of page is determined from the replacement algorithm used. The most commonly used replacement algorithms are First-In-First-Out (FIFO) and Least Recently Used (LRU).

First-In-First-Out (FIFO)

The simplest page replacement algorithm is FIFO algorithm. When a page is to be replaced, the oldest page is chosen, i.e. the page that has been in the memory for the longest time. When a page is loaded into the memory, an identification number is pushed into the FIFO stack. When required, the page can be easily determined and removed from the memory. The page whose identification number is at the top of the FIFO stack is removed. The advantage of FIFO algorithm is that it is easy to understand and implement but its performance is not always good.

Least Recently Used (LRU)

In the LRU algorithm, the page that has not been used for the longest period of time is replaced by the new page. The principle is very simple; the page that has not been used until now is not going to be used in near future also. This algorithm can be implemented with the help of a counter. Each page that is at present in the main memory has its associated counter, which is incremented by 1 at regular interval of time. The page with the highest count is the least recently used page and is replaced by the new page when required. These counters indicate the age of the page and hence, are known as aging registers.
11.3 MEMORY MANAGEMENT HARDWARE

The function of a Memory Management Unit (MMU) is to convert logical memory address into actual physical (real) memory address and provide protection. The MMU is a hardware component placed in between the processor and the main memory. The CPU, which does not contain on-chip MMU, uses an external MMU. The MMU is used in a multiprogramming or multiuser system.

The tasks, which are performed by the MMU, can be performed by the operating system; however, the operating system will not get sufficient time for program execution and its other tasks. The MMU reduces the burden of the operating system. The memory addresses used in a program are logical addresses. They indicate the logical position of instructions and data in the program. A logical address is the location of an instruction or data relative to the beginning of the program. For example, the instruction JNZ LOOP. The label LOOP represents a logical address to which the program will jump if the zero flag is not set. When a program is compiled, each logical address is represented by a memory pointer consisting of two components. In a segment oriented system, the two components are segment selector and offset. In a page oriented system, the two components are page address and page offset. In other words, a logical address is represented by a page number and a page offset, i.e. a relative address within the page.

The modern MMU provides the facility of virtual memory for providing very large memory space to users. Virtual memory is a technique that allows programmers to use more memory than what a computer actually has. The main memory which is actually present in a computer system is known as physical or real memory. The processor executes a program, which resides in the main memory. By virtual memory technique, a programmer can make a program which uses more memory than the actual capacity of the main memory. The program is stored in the secondary memory, usually disks. The MMU transfers a part of the program from the secondary memory to the main memory, which is currently needed. Similarly, it sends back those parts of the program to the secondary memory which is not being currently used. This to and fro movement of instructions and data between the main memory and the secondary memory is known as swapping.

To explain how an MMU manages more memory than the actual capacity of the main memory, we shall take the example of 80286 family of microprocessor. It has 24 address lines. With 24 address lines, only 16 MB memory can be addressed. But its virtual memory capacity is 1 GB. In its virtual mode of operation, the memory pointer consists of a 16-bit segment selector and a 16-bit offset. The segment selector has 14 address bits and 2 privilege level bits. The segment selector specifies an index into the memory resident table (descriptor table) called descriptor. The descriptor gives a 24-bit segment address. The physical address is obtained
by adding the 16-bit offset to the 24-bit segment address. This computation of physical address from a logical address is carried out by the MMU.

With the help of 14 address bits the segment selector can select any one of 16384 (2^{14}) descriptors. As the offset component of the logical address is of 16 bits, each segment will contain 64 KB. As there is one descriptor for each segment, the total logical address will be 64 KB * 16384, about 1 GB. Thus, the MMU can manage 1 GB of memory by the virtual memory technique. Now the question is ‘How are 1 GB addresses managed with only 16 MB of real memory?’ This is managed by keeping only the currently used segments in the physical memory.

When the CPU sends a logical address to the MMU, it checks whether the segment containing this logical address is present in the physical memory. If the segment is present in the physical memory, the MMU calculates the physical address corresponding to the supplied logical address. When the segment corresponding to the supplied logical address is not present in the physical memory, the MMU interrupts the CPU. On receipt of an interrupt from the MMU, the CPU reads the desired code or data segment from the disk (i.e., secondary memory). The MMU then calculates the required physical address. The 1 GB logical addresses (in case of 80286) available to users are called virtual memory and the corresponding logical addresses are also called virtual addresses. The term, virtual, refers to something which appears to be present but actually is not.

The advantage of a segmented memory is that it offers a simple handling technique for growing data structures. When it is not known how large a particular data structure will become, it is not necessary to make a guess. The data structure can be assigned its own segment. The operating system will expand or shrink it as required. Another advantage is that a few descriptors are required for large programs or data spaces. In the segment oriented system, the size of the segments, which are swapped in and out of physical memory, is quite large. Their loading into the memory is a time consuming process. The paging technique overcomes this difficulty.

### Check Your Progress

1. What is virtual address space?
2. What is the advantage of FIFO algorithm?
3. Define logical addresses.

### 11.4 DIRECT MEMORY ACCESS

Direct Memory Access (DMA) is a technique for moving data directly between the main memory and the peripheral devices without the intervention of CPU.

Direct memory access is system pathways that are used by many I/O devices for transferring information directly to and from memory. Today, DMA is used
most commonly by floppy drive, tape drives and sound cards. Thus, removing the CPU from the path and letting the peripheral devices manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access.

During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

Figure 11.4 shows two control signals in the CPU facilitating the DMA transfer. The Bus Request (BR) input is used by the DMA controller to request the CPU to relinquish the control of the buses. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus and the read and write lines into a high impedance state. Then the CPU activates the Bus Grant (BG) output to inform the external DMA that the buses are in the high impedance state. The DMA that originated the bus request can now take control of the buses to conduct memory transfers without the processor intervention. When the DMA terminates the transfer, it disables the bus request line. The CPU then disables the bus grant line and takes control of the buses and returns to the normal operation.

When the DMA takes control of the bus system, it communicates directly with the memory. The transfer can be made in the following ways:

- **Burst Transfer**: In the DMA burst transfer, a block sequence consisting of a number of memory words is transferred into a continuous burst, while the DMA controller is the master of the memory buses.
- **Cycle Stealing**: In this method, the DMA controller transfers one data word at a time, after which it must return control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to ‘steal’ one memory cycle.

**DMA Controller**

The DMA controller needs the circuits of an interface to communicate with the CPU and the I/O device. It also needs an address register, a word count register and a set of address lines. The address register and address lines are used for direct communication with the memory. The word count register specifies the number of words that must be transferred.

Figure 11.5 shows the block diagram of DMA controller. The unit communicates with the CPU via the data bus and control lines. The registers in the
DMA are selected by the CPU through the address bus by enabling the DS (DMA Select) and RS (Register Select) inputs. When the BG (Bus Grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG = 1, the CPU relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control. The DMA controller has three registers. The address register contains an address to specify the desired location in the memory and the address bits go into the address bus through address bus buffers. The address register is incremented after each word is transferred to the memory. The word count register holds the number of words to be transferred to the memory. This register is decremented by one after each word is transferred to the memory and internally tested for zero. The control register specifies the mode of transfer. All registers in the DMA appear to the CPU as I/O interface registers and hence, the CPU can read from or write into the DMA registers.

**Fig. 11.5 Block Diagram of DMA Controller**

**DMA Initialization**

The CPU initializes the DMA by sending the following information through the data bus:

- The starting address of the memory block where data are available (for read) or where data are to be stored (for write).
- The word count which is the number of words in the memory block.
- Control to specify the mode of transfer such as read and write.
- A control to start the DMA transfer.

The starting address is stored in the address register. The word count is stored in the word count register and the control information is stored in the control register. Once the DMA is initialized, the CPU stops communicating with the DMA unless it receives an interrupt signal or if it wants to check how many words have been transferred.
DMA Transfer

Figure 11.6 shows the DMA transfer in a computer system. With the help of address and data bus, the CPU communicates with the DMA. The DMA has its own address, which activates DS and RS lines. The CPU initializes the DMA through data bus. The DMA can start data transfer between the peripheral device and the memory once it receives the start control command.

When the peripheral device sends a DMA request, the DMA controller activates the BR line, requesting the CPU to give control over the buses. Then CPU activates BG line and informs the DMA controller that its buses are now disabled. Then the DMA sends a DMA acknowledge to the device by putting the current value of its address register into the address bus and initiates the RD or WR signals. After receiving the acknowledge signal, the device puts a word in the data bus (for write) or receives a word from data bus (for read). The DMA increments its address register and decrements the word count register as each word is transferred. Internally, the word count register is tested for zero. If the word count register reaches zero, the DMA stops any further transfer and disables its bus request signals and informs the CPU through an interrupt.

When the CPU receives the interrupt signals, it reads the word count register. The zero value in the word count register signifies that all the words are transferred successfully.

DMA transfer is very useful in many applications. It is used for fast transfer of information between the magnetic disk and the memory.

![Fig. 11.6 DMA Transfer in a Computer System](image-url)
11.5 MEMORY MANAGEMENT AND SYSTEM PERFORMANCE

Memory management is an act of managing computer memory. This approach involves providing ways to allocate portions of memory to programs at their request and freeing it for reuse when no longer needed. The management of main memory is critical to the computer system. Virtual memory systems separate the memory addresses used by a process from actual physical addresses allowing separation of processes and increasing the effectively available amount of RAM using paging or swapping to secondary storage. The quality of the virtual memory manager can have an impact on overall system performance. Garbage collection is the automated deallocation of computer memory resources for a program. This is generally implemented at the programming language level and is in opposition to manual memory management, the explicit allocation and deallocation of computer memory resources. Region based memory management is an efficient variant of explicit memory management that can deallocate large groups of objects simultaneously. The various issues are solved during the troubleshooting of hardware and memory management. Among this, memory leak, bad memory, grab more memory, displays blank screen, system hangs or sudden reboots, shows general protection faults, memory errors reported by system unit, memory mismatch, memory verification errors, memory failure, memory address errors, soft errors shift by servers, etc., have been facing by system engineer as common issues and hence every time organization has to invest huge amount for these problems. The several memory problems can be troubleshooting while recognizing the system problems. The following are considered as prime factors to determine the system performance while the time of managing memory:

Memory Management with Bitmaps

When memory is assigned dynamically, the operating system must manage it. In general terms, there are two ways to keep track of memory usage: bitmaps and free lists. With a bitmap, memory is divided up into allocation units, perhaps as small as a few words and perhaps as large as several KB. Corresponding to each allocation unit is a bit in the bitmap, which is 0 if the unit is free and 1 if it is occupied or vice versa. Figure 11.7 represents a part of memory with five processes and three hole in which the tick marks show the memory allocation units whereas the shaded regions (0 in the bitmap) are free.
The size of the allocation unit is an important design issue. The smaller the allocation unit, the larger the bitmap. However, even with an allocation unit as small as 4 bytes, 32 bits of memory will require only 1 bit of the map. A memory of 32n bits will use n map bits, so the bitmap will take up only 1/33 of memory. If the allocation unit is chosen large, the bitmap will be smaller but appreciable memory may be wasted in the last unit of the process if the process size is not an exact multiple of the allocation unit. A bitmap provides a simple way to keep track of memory words in a fixed amount of memory because the size of the bitmap depends only on the size of memory and the size of the allocation unit. The main problem with it is that when it has been decided to bring a k unit process into memory, the memory manager must search the bitmap to find a run of k consecutive 0 bits in the map. Searching a bitmap for a run of a given length is a slow operation because this is an argument against bitmaps.

Memory Management with Linked List

Another way of keeping track of memory is to maintain a linked list of allocated and free memory segments, where a segment is either a process or a hole between two processes.

The memory of Figure 11.8 (a) is represented in Figure 11.7 (c) as a linked list of segments. Each entry in the list specifies a hole (H) or process (P), the address at which it starts, the length and a pointer to the next entry. In this example, the segment list is kept sorted by address. Sorting this way has the advantage that when a process terminates or is swapped out, updating the list is straightforward. A terminating process normally has two neighbors except when it is at the very top or bottom of memory. These may be either processes or holes leading to the four combinations of Figure 11.8. In Figure 11.8 (a) updating the list requires replacing a P by an H. In Figure 11.8 (b) and Figure 11.8 (c), two entries are coalesced into one and the list becomes one entry shorter. In Figure 11.8 (d), three entries are merged and two items are removed from the list. Since the process table slot for the terminating process will normally point to the list entry for the process itself, it may be more convenient to have the list as a double linked list rather than the
single linked list of Figure 11.7 (c). This structure makes it easier to find the previous entry and to see if a merge operation is possible.

![Diagram showing four neighbor combinations for the terminating process X](image)

**Fig. 11.8** Four Neighbor Combinations for the Terminating Process X

When the processes and holes are kept on a list sorted by address, several algorithms can be used to allocate memory for a newly created process. It is assumed that the memory manager knows how much memory to allocate. The simplest algorithm is First Fit. The memory manager scans along with the list of segments until it finds a hole that is big enough. The hole is then broken up into two pieces, one for the process and one for the unused memory except in the statistically unlikely case of an Exact Fit. First Fit is a fast algorithm because it searches as little as possible. A variation of First Fit is Next Fit. It works the same way as first fit, except that it keeps track of where it is whenever it finds a suitable hole. The next time it is called to find a hole and it starts searching the list from the place where it left off last time instead of always at the beginning, as First Fit does. Another well known algorithm is Best Fit. Best Fit searches the entire list and takes the smallest hole that is adequate. Rather than breaking up a big hole that might be needed later, Best Fit tries to find a hole that is close to the actual size needed. As an example of First Fit and Best Fit, consider Figure 11.7. If a block of size 2 is needed, First Fit will allocate the hole at 5 but Best Fit will allocate the hole at 18. Best Fit is slower than First Fit because it must search the entire list every time it is called. This approach also results in more wasted memory than First Fit or Next Fit because it tends to fill up memory with tiny and useless holes. First Fit generates larger holes on the average. The track of memory is maintained while troubleshooting through linked list programs. In this, a segment is being allotted between the two processes. The linked list facilitates a hole for each entry and calculates the address when starts and reaches to the next entry. In this troubleshooting process system engineer/programmer provides a segment list that is sorted by address.

**Memory Management with Space and Time Trade Offs**

The space and time trade-offs can be done for memory that uses memory to save result for previous computations. For example, counting bit in a given word, caching uses memory space for saving time, inline procedure uses space for saving time, encoded fields use decoding time for saving space, redundant data along with extra links in a data structure use, postscript mechanism saves space and indexes used in database saves time when memory is used that also support any index...
trades of space for time, for example, 8GB is allotted for the top of system address in memory space. The reserved area is kept after Flash memory. The PCI memory range is set by chipsets and direct media interface. The upper BIOS and lower BIOS area is covered up by 1 MB and 960 KB, respectively. The total scenario is fixed for bit counting. The top of usable DRAM is the part of memory and visible to the OS. Device driver interface is designed to make a link up with design issues and techniques. The main role of device numbers is to name which device to use among those controlled by a single controller but they are often used to convey other information and hence this mapping is used in troubleshooting process for managing the memory. The setting for troubleshooting process is done by the following way by system engineer/programmer:

- The device number 0 rewinds after one successful completion but the device number 8 does not rewind.
- The device number 0 works on normal sized paper whereas device number 8 works on legal sized paper.
- The device number 0 basically writes at high density whereas device number 1 and device number 2 write on medium and low density, respectively.

The two types of device drivers are known as block or disk device drivers and character device drivers. The block or disk device drivers disks have fixed size, addressable blocks so no length is required, it is always one block whereas character device drivers deliver a stream of data with no addresses. In fact, no device address is used in the character device drivers. These devices need a ‘device control’ command to handle detailed points, for example which paper tray is to be used. The following functions of device drivers create useful effects in the I/O system:

- The device drivers make partition of physical disk into various logical units. These units are called partitions.
- These drivers work on single logical unit that is a combination of various physical disks.
- The drivers enhance RAM as a super fast disk.
- The drivers can read address space in a disk.
- The drivers basically work to a program not to a terminal.

Memory Management with Disk Head Scheduling

Input/Output (I/O) scheduling is a term used to describe the method computer operating systems decide the order that block I/O operations will be submitted to storage volumes. I/O scheduling is sometimes called ‘disk scheduling’. This mechanism is also an important task that decides many useful decisions, for example if we have two or more pending disk requests which should go first. The disk head scheduling works on various mechanism, such as First Come First Served (FCFS), Shortest Seek Time First (SSTF). The SSTF solves the starvation problem by using the two techniques known as batching or aging. Elevator algorithm is required
for disk head scheduling. The Elevator algorithm starts at cylinder 0 with direction ‘up’. The following processes are required to save the memory:

- Let the current cylinder be N.
- The second step is required to select the closest request for cylinder ‘N’ for up direction otherwise select the else option that is less higher for cylinder ‘N’ for low direction.
- The switch direction can be used if no request direction is determined.
- The second step is repeated at this step.

The disk scheduling is necessary because FCFS scheduling has major drawbacks, such as seek randomly distributed locations if heavy load traps on various types of operating system running, such as Windows OS, Mac OS, Linux/UNIX, etc. But, disk scheduling algorithm minimizes seek times with high latency. The modern OS performs rotational optimization via throughput that performs number of requests per unit of time, mean response time that spends average time or the required services and variance in response time that measures the predictability of response time. Maximize throughput and high bandwidth are the overall goals of I/O systems.

**Memory Management with Late Binding**

Late binding is a computer programming mechanism in which the method being called upon an object is looked up by name at runtime. This is also known as duck typing or name binding. Late binding is different from the concept of dynamic dispatch but there are significant differences. With early binding the compiler statically verifies that there are one or more methods with the appropriate method name and signature. This is usually stored in the compiled program as an offer in a virtual method table or v-table and is very efficient. With late binding, the compiler does not have enough information to verify the method even exists, let alone bind to its particular slot on the v-table. Instead the method is looked up by name at runtime. The primary advantage of using late binding in Component Object Model (COM) programming is that it does not require the compiler to reference the libraries that contain the object at compile time. Late binding involves virtual memory and network routing that decide route at late moment. It also works for stack allocation that allocates procedure variables in calling procedure function, key coding that bind to American Standard Code for Information Interchange (ASCII) and sends the key number and manufacturing that involves Just-In-Time (JIT) inventory that means the pending task is not prolonged. The “bind” word involves the binding time that binds a value to the attribute, binds a variable at procedure call time that is used in compile time and run time in high level languages. Late binding techniques are used while the troubleshooting process and in this lazy creation waits to create the objects until and unless they are needed. The lazy creation fetches Web page images if they are visible creates windows if they are supposed to become visible and copy large memory area that uses create the copy late, if possible. The lazy evaluation process evaluates the function arguments not at the time of function calling but when they are used in the programs. In this way, memory is reserved and managed for further processing.
Memory Interleaving

Interleaving is an advanced technique used by high-end motherboards/chipsets to improve memory performance. Memory interleaving increases bandwidth by allowing simultaneous access to more than one stack of memory. This improves performance because the processor can transfer more information to and from memory in the same amount of time. It helps alleviate the processor memory bottleneck that is a major limiting factor in overall performance. Interleaving works by dividing the system memory into multiple blocks. The most common numbers are two or four called two-way or four-way interleaving respectively. Each block of memory is accessed using different sets of control lines which are merged together on the memory bus. When a read or write is begun to one block, a read or write to other blocks can be overlapped with the first one. The more blocks, the more overlapping can be done. Interleaving is an advanced technique that is not generally supported by most PC motherboards because of cost. It is most helpful on high-end systems, especially servers that have to process a great deal of information quickly. Interleaving infrastructure is arranged with memory. To speed up the memory operations (read and write), the main memory of $2^n = N$ words can be organized as a set of $2^m = M$ independent memory modules (where $m < n$) each containing $2^n - 2^m$ words. If these M modules can work in parallel (or in a pipeline fashion), then ideally an M fold speed improvement can be expected. The n-bit address is divided into an m-bit field to specify the module and another (n–m)bit field to specify the word in the addressed module. The field for specifying the modules can be either the most or least significant m bits of the address. For example, these are the two arrangements of $M = 2^m = 2^2 = 4$ modules (m = 2) of a memory of $2^n = 2^4 = 16$ words (n = 4). Before the data signal is modulated and spread, an interleaving process scatters the bit order of each frame so that if some data is lost during transmission due to a deep fade of the channel, for example the missing bits can possibly be recovered during decoding. This provides effective protection against rapidly changing channels but is not effective in slow changing environments.

![Fig. 11.9 (a) High Order Arrangement and (b) Low Order Arrangement of Interleaving](image)
In general, the CPU accesses the memory for a set of consecutive words either a segment of consecutive instructions in a program or the components of a data structure, such as an array, the interleaved (low order) arrangement is preferable as consecutive words are in different modules and can be fetched simultaneously (see Figure 11.9 (b)). In case of high order arrangement, the consecutive words are usually in one module, having multiple modules is not helpful if consecutive words are needed (refer Figure 11.9 (a)). The following example on interleave infrastructure will make the concept clear.

**Example 11.1:** A memory of \(2^n = 64K\) words \((n = 16)\) with \(2^m = 16\) modules \((m = 4)\) each containing \(2^{n-m} = 2^{12} = 4K\) words.

**Solution:**

In interleaving process you can find that a memory of \(2^n = 64K\) words are utilized if \((n = 16)\), i.e., \(2^m = 16\) modules \((m = 4)\) are given. Each contains \(2^{n-m} = 2^{12} = 4K\) words.

### 11.6 VIRTUAL MACHINES

The concept of virtualization can be applied not only to subsystems such as disks. It can be applied to an entire machine. To implement a virtual machine, developers add a software layer to a real machine to support the desired architecture. By doing so, a VM can circumvent real machine compatibility and hardware resource constraints.

**Levels of Virtualization Implementation**

A traditional computer runs with a host operating system specially tailored for its hardware architecture, as shown in Figure 11.10. After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software, called a virtualization layer as shown in Figure 11.10. This virtualization layer is known as hypervisor or virtual machine monitor (VMM).
The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources. The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively. This can be implemented at various operational levels, as we will discuss shortly. The virtualization software creates the abstraction of VMs by interposing a virtualization layer at various levels of a computer system. Common virtualization layers include the instruction set architecture (ISA) level, hardware level, operating system level, library support level, and application level (see Figure 11.11).

![Architecture of a Computer System before and after Virtualization, where VMM Stands for Virtual Machine Monitor](image)

**Fig. 11.10** Architecture of a Computer System before and after Virtualization, where VMM Stands for Virtual Machine Monitor

![Virtualization Ranging from Hardware to Applications in Five Abstraction levels](image)

**Fig. 11.11** Virtualization Ranging from Hardware to Applications in Five Abstraction levels
VMs provide two other benefits that are commercially significant.

1. **Managing software**: VMs provide an abstraction that can run the complete software stack, even including old operating systems such as DOS. A typical deployment might be some VMs running legacy OSes, many running the current stable OS release, and a few testing the next OS release.

2. **Managing hardware**: One reason for multiple servers is to have each application running with its own compatible version of the operating system on separate computers, as this separation can improve dependability. VMs allow these separate software stacks to run independently yet share hardware, thereby consolidating the number of servers. Another example is that some VMMs support migration of a running VM to a different computer, either to balance load or to evacuate from failing hardware.

**Check Your Progress**

4. What is direct memory access?
5. Define garbage collection.
6. Why disk scheduling is necessary?

### 11.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. An address generated by the user program is called virtual address and the set of virtual addresses makes the virtual address space.
2. The advantage of FIFO algorithm is that it is easy to understand and implement but its performance is not always good.
3. A logical address is the location of an instruction or data relative to the beginning of the program.
4. Direct Memory Access (DMA) is a technique for moving data directly between the main memory and the peripheral devices without the intervention of CPU.
5. Garbage collection is the automated de-allocation of computer memory resources for a program.
6. The disk scheduling is necessary because FCFS (First Come First Serve) scheduling seeks randomly distributed locations if heavy load traps on various types of operating system, such as Windows OS, Mac OS, Linux/UNIX, etc. But, disk scheduling algorithm minimizes seek times with high latency.
11.8 SUMMARY

- Virtual memory is a technique that allows the execution of processes that may not be completely in the memory.
- The address bit of the instruction code will consist of 20 bits but the physical memory addresses must be specified using only 15 bits. Thus, the CPU will reference instructions and data with 20-bit address, but the information at this address must be taken from the physical memory rather than the auxiliary memory.
- In the LRU (Least Recently Used) algorithm, the page that has not been used for the longest period of time is replaced by the new page. The principle is very simple; the page that has not been used until now is not going to be used in near future also.
- MMU (Memory Management Unit) is a hardware component placed in between the processor and the main memory. The CPU, which does not contain on-chip MMU, uses an external MMU. The MMU is used in a multiprogramming or multiuser system.
- Computers have an input/output subsystem, referred to as I/O subsystem, which provides an efficient mode of communication between the central system and the outside world. Programs and data must be entered into the computer memory for processing and results obtained from computations must be displayed or recorded for the user’s benefit.
- Direct Memory Access or DMA is system pathways that are used by many I/O devices for transferring information directly to and from memory. Today, DMA is used most commonly by floppy drive, tape drives and sound cards.
- The DMA controller needs the circuits of an interface to communicate with the CPU and the I/O device. It also needs an address register, a word count register and a set of address lines.
- The CPU initializes the DMA through data bus. The DMA can start data transfer between the peripheral device and the memory once it receives the start control command.
- Memory management is an act of managing computer memory. This approach involves providing ways to allocate portions of memory to programs at their request and freeing it for reuse when no longer needed.
- Memory of a computer system can be managed with bitmaps, linked list, space and time trade-offs, disk head scheduling and late binding.
- Interleaving is an advanced technique used by high end motherboards/chipsets to improve memory performance. Memory interleaving increases bandwidth by allowing simultaneous access to more than one stack of
memory. This improves performance because the processor can transfer more information to and from memory in the same amount of time.

- The concept of virtualization can be applied not only to subsystems such as disks. It can be applied to an entire machine. To implement a virtual machine, developers add a software layer to a real machine to support the desired architecture.

11.9 KEY WORDS

- Memory Management Unit: It converts logical memory address into actual physical (real) memory address and provide protection.
- Descriptor: The segment selector which specifies an index into the memory resident table.
- Interleaving: An advanced technique used by high end motherboards/chipsets to improve memory performance.

11.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Discuss the benefits of virtual memory.
2. What do you understand by page replacement?
3. Write a note on memory management hardware.
4. What is memory interleaving?
5. What are the benefits of virtual machines?

Long Answer Questions

1. Explain the mapping process in virtual memory.
2. What is DMA controller? Explain.
3. How memory can be managed with linked list? Explain with the help of example.
4. Explain the techniques used by high end motherboards/chipsets to improve memory performance.
5. Write a detailed note on virtual machines.

11.11 FURTHER READINGS


UNIT 12 ISSUES

Structure
12.0 Introduction
12.1 Objectives
12.2 Crosscutting Issues in the Design of Memory Hierarchy
12.3 Answers to Check Your Progress Questions
12.4 Summary
12.5 Key Words
12.6 Self Assessment Questions and Exercises
12.7 Further Readings

12.0 INTRODUCTION

In this unit, you will learn about the various crosscutting issues relating to the design of memory hierarchy. The memory hierarchy consists of the total memory system of any computer. The memory components range from higher capacity slow auxiliary memory to a relatively fast main memory to cache memory that can be accessible to the high speed processing logic.

12.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the issues related to the memory hierarchy
- Explain the crosscutting issues in design of memory hierarchy

12.2 CROSSCUTTING ISSUES IN THE DESIGN OF MEMORY HIERARCHY

Various issues related to memory hierarchy are given below:

- **Deciding the block size:** smallest unit that is managed at each level.
- **Block placement:** Where can a block be placed? – E.g., direct mapped, set associative, fully associative.
- **Block identification:** How can a block be found? – E.g., hardware tag matching, OS page table.
- **Block replacement:** Which block should be replaced? – E.g., LRU, random.
- **Write strategy:** What happens on a write? – E.g., write-through, write-back, write-allocate.
Issues related to the design of memory hierarchy are:

1. **Protection and Instruction Set Architecture**

   Protection is a joint effort of architecture and operating systems, but architects had to modify some awkward details of existing instruction set architectures when virtual memory became popular. For example, to support virtual memory in the IBM 370, architects had to change the successful IBM 360 instruction set architecture that had been announced just 6 years before. Similar adjustments are being made today to accommodate virtual machines. Historically, IBM mainframe hardware and VMM took three steps to improve performance of virtual machines:
   - (i) Reduce the cost of processor virtualization.
   - (ii) Reduce interrupt overhead cost due to the virtualization.
   - (iii) Reduce interrupt cost by steering interrupts to the proper VM without invoking VMM

2. **Coherency of Cached Data**

   Data can be found in memory and in the cache. As long as the processor is the sole component changing or reading the data and the cache stands between the processor and memory, there is little danger in the processor seeing the old or stale copy. As we will see, multiple processors and I/O devices raise the opportunity for copies to be inconsistent and to read the wrong copy. The frequency of the cache coherency problem is different for multiprocessors than I/O. Multiple data copies are a rare event for I/O and avoided whenever possible - but a program running on multiple processors will want to have copies of the same data in several caches. Performance of a multiprocessor program depends on the performance of the system when sharing data.

   The I/O cache coherency question is that: Where does the I/O occur in the computer - between the I/O device and the cache or between the I/O device and main memory? If input puts data into the cache and output reads data from the cache, both I/O and the processor see the same data. The difficulty in this approach is that it interferes with the processor and can cause the processor to stall for I/O. Input may also interfere with the cache by displacing some information with new data that are unlikely to be accessed soon. The goal for the I/O system in a computer with a cache is to prevent the stale data problem while interfering as little as possible. Many systems, therefore, prefer that I/O occur directly to main memory, with main memory acting as an I/O buffer. If a write-through cache were used, then memory would have an up-to-date copy of the information, and there would be no stale data issue for output. (This benefit is a reason processor used write through.) Alas, write through is usually found today only in
first-level data caches backed by an L2 cache that uses write back. Input requires some extra work. The software solution is to guarantee that no blocks of the input buffer are in the cache. A page containing the buffer can be marked as non-cacheable, and the operating system can always input to such a page. Alternatively, the operating system can flush the buffer addresses from the cache before the input occurs. A hardware solution is to check the I/O addresses on input to see if they are in the cache. If there is a match of I/O addresses in the cache, the cache entries are invalidated to avoid stale data. All of these approaches can also be used for output with write-back caches.

**Check Your Progress**

1. What are the writing strategies in a cache?
2. Give any two block replacement policies.

### 12.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Write-through, write-back, write-allocate are the writing strategies in a cache.
2. LRU and random are the two block replacement policies.

### 12.4 SUMMARY

- Protection is a joint effort of architecture and operating systems, but architects had to modify some awkward details of existing instruction set architectures when virtual memory became popular.
- Data can be found in memory and in the cache. As long as the processor is the sole component changing or reading the data and the cache stands between the processor and memory, there is little danger in the processor seeing the old or stale copy.
- The goal for the I/O system in a computer with a cache is to prevent the stale data problem while interfering as little as possible.

### 12.5 KEY WORDS

- **Memory**: An electronic circuitry that allows data to be stored and retrieved when required.
- **Cache Memory**: It is an extremely fast *memory* type that acts as a buffer between *RAM* and the CPU.
12.6 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short Answer Questions

1. What are the steps that VMM took to improve performance of virtual machines?
2. What are the block replacement policies?

Long Answer Questions

1. What are the different issues related to memory hierarchy?
2. Explain the various issues with the design of memory hierarchy.

12.7 FURTHER READINGS

Storage is a process by which digital data can be saved into data storage device using computing technology. Storage is a structure which enables a computer to store data, either temporarily or permanently. Storage can also be referred as computer data storage or electronic data storage. Storage is one of the most important key components of a computer system and can be divided into number of forms, which can be divided into mainly two parts:

1. **Volatile Storage**: It needs a continuous supply of electricity for storing/retaining data. It stores data in the primary memory for temporarily storing data in the computer. Cache memory and random-access memory (RAM) are the examples of volatile storage.

2. **Non-Volatile Storage**: It is a mechanism that stores digital data and retains it whether the power is off or on. It is also known as secondary storage which stores the permanently. Hard disk, USB storage and optical media are examples of non-volatile storage.

For storing, porting and extracting data files and objects, we use storage devices as any computing hardware. In storage device we can store data in both forms that is temporarily and permanently, and can be internal or external to a
13.1 OBJECTIVES

After going through this unit, you will be able to:

- Define storage devices and their types
- Discuss the advancement in storage devices
- Understand the various types of faults and failures
- Explain the performance and reliability measures and benchmarking

13.2 STORAGE DEVICES

Storage devices are the most important component devices. It stores virtually on the computer all types of data and applications rather than hardware firmware. They are accessible on a number of form factors depending on the type of elementary devices. Standard computer has multiple storage devices like RAM, cache, and hard disk, which also consist of optical disk drives and externally connected USB drives.

Storage devices are mainly of two types:

1. **Primary Devices**: These devices are smaller in size and are designed to store data temporarily and are internal to the computer. It can be accessed with speed as it includes RAM and Cache memory.

2. **Secondary Devices**: These devices have huge storage capacity and can store data permanently on the computer system. They are internal or external to the computer, and includes storage devices like hard disk, optical disk drive and USB.

13.2.1 Unit of Storage

We can target to store digital information using software commands. Bit is the smallest unit of measurement used for measuring data. 0 or 1 are the values of one bit. A bit can have binary values like On/Off or True/False. For measuring data the fundamental unit is a byte, or 8 bits. For representing standard ASCII code like letters, numbers and symbols, a byte can store 28 or 256 number of values.

Most of the files consist of thousands of bytes, which can be measured in kilobytes. For large files like images, videos, and audio files which includes millions of bytes and can be measured in megabytes. We use gigabytes or even terabytes for storing data in modern storage devices. Most of the units for measurement are generally restrained for measuring multiple storage devices or the capacity of large data storage networks. We use “b” is lowercase for bits and uppercase “B” for bytes.
13.2.2 Types of Storage Devices

1. **Hard Disk**: A disk drive, hard drive or hard disk drive stores and gives frequent and quick access to large amounts of data on an electromagnetically charged surface or set of surfaces. Now a day computers consist of hard disk which contains billions to trillions of bytes of storage of data.

**Parts of a Hard Drive**: There are number of components in hard drive which includes the platter for storing data, a spindle for spinning platters, a read/write arm for reading and writing data, an actuator to control the action and movement of the read/write arm and a logic board. One or more aluminium, glass or ceramic platters made of substrate material in a thin magnetic surface, or media layer are included in hard disks for storage of data. For platters store and data is organized in a specific patterns such as tracks, sectors and clusters which can only have some millionths of an inch thick. A super thin protective and lubricating protective layer above the magnetic media guards against accidental damage and contamination by foreign material, like dust.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Value</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit (b)</td>
<td>0 or 1</td>
<td>1/8 of a byte</td>
</tr>
<tr>
<td>byte (B)</td>
<td>8 bits</td>
<td>1 byte</td>
</tr>
<tr>
<td>kilobyte (KB)</td>
<td>1000^1 bytes</td>
<td>1.000 bytes</td>
</tr>
<tr>
<td>megabyte (MB)</td>
<td>1000^1 bytes</td>
<td>1,000,000 bytes</td>
</tr>
<tr>
<td>gigabyte (GB)</td>
<td>1000^1 bytes</td>
<td>1,000,000,000 bytes</td>
</tr>
<tr>
<td>terabyte (TB)</td>
<td>1000^1 bytes</td>
<td>1,000,000,000,000 bytes</td>
</tr>
<tr>
<td>petabyte (PB)</td>
<td>1000^1 bytes</td>
<td>1,000,000,000,000,000 bytes</td>
</tr>
<tr>
<td>exabyte (EB)</td>
<td>1000^1 bytes</td>
<td>1,000,000,000,000,000,000 bytes</td>
</tr>
<tr>
<td>zettabyte (ZB)</td>
<td>1000^1 bytes</td>
<td>1,000,000,000,000,000,000,000 bytes</td>
</tr>
</tbody>
</table>
2. **Magnetic Disk**: It is one of the most common form of secondary storage for a computer system as they give fast access and high storage capacities at acceptable cost. It consists of metal disks which are coated on both sides with an iron oxide recording material. Numbers of disks are mounted together on a vertical shaft, which typically rotates the disks at speeds of 3,600 to 7,600 revolutions per minute (rpm). Electromagnetic can read or write heads which are positioned by access arms among the slightly separated disks to read or write data on concentric, circular tracks. Tiny magnetized spots are recorded on the tracks to form binary digits for common computer codes. On each disk surface, we can record thousands of bytes on individual tracks which can provide with billions of storage positions for software and data.

3. **Magnetic Tape**: It is a storage medium which allows for data archiving, collection, and backup. Earlier tapes were wound in wheel-like reels then into cassettes and cartridges came into picture which provides more protection for the tape inside. In magnetic tape one side is coated and data on the tape is written. To find the particular record require time as the machine reads every record in front of it. For archival purpose tapes are used rather than ad-hoc reading or writing. On tracks data is written in such a way that some does recording of data linearly (that is edge of computer) whereas some does helical recording (that is written diagonally). Nowadays 128 or more tracks are used as compare to older magnetic tapes which used 8 tracks.

4. **Flash Drive**: Flash memory and an integrated Universal Serial Bus (USB) interface are the data storage that is in USB flash drive. USB flash drives are either removable or rewritable and are small, durable and reliable.
physically. To operate faster there is a need to have larger storage space.
As there are no moving parts. Hence, USB flash drives are mechanically
very powerful. They derive the power to operate from the device to which
they are connected (typically a computer) via the USB port. USB drive or
flash drive are the other name for USB flash drive.

According to USB mass storage standard USB flash drive can be supported
by all BIOS and all operating systems. USB flash drive can store more data and at
faster rate as compared to optical disk drives and floppy disks.

The major components of flash drive includes USB connector, which is well
protected and electrically insulated inside a plastic or rubber case. USB flash
drive casing includes a small printed circuit board mounted with integrated circuits.

USB flash drive consists of following components:
- **Standard USB plug**: It connects the flash drive to a device.
- **USB mass storage controller**: It is a microcontroller which consists of
  small amount of RAM and ROM.
- **NAND flash memory chip**: It stores the data.
- **Crystal oscillator**: It controls the data output.

### 13.3 ADVANCED TOPICS IN DISK STORAGE

Various terms related to disk storage are:

- **Areal Density**: Nowadays disk industry improving the disk capacity. This
  improvement can be measured in bits per square inch, can be expressed in
  areal density.

\[
\text{Areal Density} = \frac{\text{Tracks on a disk surface} \times \text{Bits on a track}}{\text{Track inch}}
\]

- **Disk Power**: Power is one of the important factors for disks and processors
  as well. 9 watts is used by ATA disk when it is in idle state (that is reading or
  writing) and requires 13 watts when it is seeking. It can save power when it
  spins smaller mass, and smaller diameter. According to Gurumurthi et al.
  2005, the formula which demonstrate the rotation of speed and the size of
  the platters for the power consumed by the disk motor by using the formula
  which is shown below:

\[
\text{Power} = \text{Diameter}^{4.4} \times \text{RPM}^{2.8} \times \text{Number of platters}
\]

To reduce the disk motor power in the motor, we are required smaller
platters, slower rotation, and fewer platters.

- **Disk Array**: A disk array which is a hardware element consists of huge
  group of hard disk drives (HDDs). A system is run which coordinates activity
  in the unit using the storage controller. The main backbone of modern storage
networking environment is disk array. A storage area network (SAN) consists of number of disk arrays which functions for the repository of the data which is moved in and out of the SAN.

- **Logical Units**: Data in the disk arrays are organized into Logical Units (LUs). Small Computer Systems Interface (SCSI) I/O commands are sent to a storage target and executed by an LU within that target.
- **LUN**: A Logical Unit Number (LUN) is an exclusive identifier which differentiates between devices which shares similar SCSI bus. Based on LUNs commands are send to the SCSI controller to identify devices.
- **Redundant Array of Inexpensive (or Independent) Disks (RAID)**: To improve performance and fault tolerance, RAID is deployed to empower storage of data into number of paces to assure corruption against data and provide data to user much faster. RAID consist of number of levels like:

  **Standard RAID Levels**

  Numbers of architectures are available for RAID devices which are known as levels that depend on the performance and fault tolerance. It characterizes how data is distributed over the drives. Standard RAID levels consist of following levels which are as discussed below:

  **Level 0: Striped Disk Array without Fault Tolerance**: It provides data striping (spreading out blocks of each file across multiple disk drives) but no redundancy. This improves performance but does not deliver fault tolerance. If one drive fails then all data in the array is lost.

  **Level 1: Mirroring and Duplexing**: It provides disk mirroring. Level 1 provides twice the read transaction rate of single disks and the same write transaction rate as single disks.

  **Level 2: Error-Correcting Coding**: It is not a typical implementation and rarely used. Level 2 stripes data at the bit level rather than the block level.

  **Level 3: Bit-Interleaved Parity**: It provides byte-level striping with a dedicated parity disk. Level 3 cannot service simultaneous multiple requests, also is rarely used.

  **Level 4: Dedicated Parity Drive**: It is a commonly used implementation of RAID. Level 4 provides block-level striping (like Level 0) with a parity disk. If a data disk fails, the parity data is used to create a replacement disk. A disadvantage to Level 4 is that the parity disk can create write bottlenecks.

  **Level 5: Block Interleaved Distributed Parity**: It provides data striping at the byte level and also stripe error correction information. This results in excellent performance and good fault tolerance. Level 5 is one of the most popular implementations of RAID.

  **Level 6: Independent Data Disks with Double Parity**: It provides block-level striping with parity data distributed across all disks.
**Level 10: A Stripe of Mirrors**: Not one of the original RAID levels, multiple RAID 1 mirrors are created, and a RAID 0 stripe is created over these.

**Non-Standard RAID Levels**

Some devices use more than one level in a hybrid or nested arrangement, and some vendors also offer non-standard proprietary RAID levels. Examples of non-standard RAID levels include the following:

- **Level 0+1: A Mirror of Stripes**: Not one of the original RAID levels, two RAID 0 stripes are created, and a RAID 1 mirror is created over them. It is used for both replicating and sharing data among disks.

- **Level 7**: A trademark of Storage Computer Corporation that adds caching to Levels 3 or 4.

- **RAID 1E**: It has a RAID 1 implementation with more than two disks. Data striping is combined with mirroring each written stripe to one of the remaining disks in the array.

- **RAID S**: It is also called Parity RAID, this is EMC Corporation’s proprietary striped parity RAID system used in its Symmetrix storage systems.

### Check Your Progress

1. What are the two types of storage system?
2. What is logical unit number?

### 13.4 REAL FAULTS AND FAILURES

- **Fault**: It is a defect within the system, Examples are Software bug, Random hardware fault, Memory bit “stuck”; Omission or commission fault in data transfer, etc.

- **Error**: It is an anomaly from the appropriate operation of system or subsystem. A fault may lead to an error, i.e., error is a mechanism by which the fault becomes apparent. Fault may stay dormant for a long time before it manifests itself as an error. Memory bit got stuck but CPU does not access this data is an example.

- **Failure**: It is a condition in which system is unable to perform the required function.

Gray and Siewiorek divide the faults into four categories according to their cause.

1. **Hardware Faults**: The device fails due to hitting of a memory cell by an alpha particle.

2. **Design Faults**: It is generally a faults in software and occasionally of hardware design.
3. **Operation Faults**: It arises due to mistakes done through operations and maintenance.

4. **Environmental Faults**: They occur due to environmental factors like fire, flood, earthquake, power failure, and sabotage.

Faults are also divided according to duration as transient, intermittent, and permanent [Nelson 1990], for limited period of time transient fault exist which are non-recurring. Faulty and Fault-free operation fluctuate among themselves to generate intermittent faults in a system.

### 13.5 I/O PERFORMANCE, RELIABILITY MEASURES, AND BENCHMARKS

I/O performance needs to be measured to have no counterparts in the design. Following measures should be considered such as I/O devices in which it can be connected to the computer system. I/O devices can connect number of devices to a computer system. Following are the unique measures of traditional measures of performance which can be applied to I/O.

- **Throughput**: It is the average number of tasks completed by the server over a time period. The server should never be idle (that is buffer should not be idle) to get the highest throughput. I/O throughput is sometimes called as I/O bandwidth.

- **Response Time**: It is the time taken by a task when it is placed in a buffer till the server finishes the task. Latency is another name for response time.

When the computer is interacting with human beings than interaction, or transaction of the computer is divided into three parts:

1. **Entry time**: This the time when the user enters the command.
2. **System response time**: when the user enters the command and completes the response time which is displayed.
3. **Think time**: It is the time from receiving of the response till the user enters the next command.

\[
\text{Transaction time} = \text{entry time} + \text{system response time} + \text{think time}
\]

**Transaction-Processing Benchmarks**

- **TPC-C**: In TPC-C number of user executes transaction across the database for simulating whole computing environment. The dominant transaction of an order-entry environment are benchmarked and these activities consist of entering and delivering orders, recording payments, checking the status of orders and monitoring the levels of stock at the warehouses. While the benchmark portrays the activity of a wholesale supplier, TPC-C is not limited to the activity of any particular business segment, but rather represents any
industry that must manage, sell, or distribute a product or service. TPC-C is the combination of five different types of concurrent transactions which are executed on-line or queued for deferred execution. The characteristics of the system components which are associated with such type of environment are as follows:

- Parallel execution of number of transaction types that measures the depth of complexity
- Execution modes can be on-line or deferred transaction.
- Number of on-line terminal sessions
- Moderate system and application execution time
- Significant disk input/output
- ACID properties (Transaction integrity)
- By using primary and secondary keys we can access in a non-uniform pattern
- With the huge variety of sizes, attributes, and relationships databases is generated.
- Contention on data access and update

To measure TPC-C performance new-order transactions per minute is used. The transaction rate (tpmC), the associated price per transaction ($/tpmC), and the availability date of the priced configuration are the primary metrics.

- **TPC-DI:** ETL is another name for Data Integration (DI) in which analysis, combination, and transformation of data from various sources and formats converted to unified data model (UML) representation. The main key part of data warehousing, application integration, and business analytics is data integration.

- **TPC-DS:** TPC-DS is one of the benchmark of industry standard for predicting the performance of decision support solutions which is not limited to Big Data systems. The latest version of TPC-DS is version v2 which models number of generally applicable aspects for decision support system, which includes queries and data maintenance. The business model of TPC-DS is a retail product supplier, the database schema, data population, queries, data maintenance model and implementation rules which are designed for representing modern decision support systems.

To illustrate decision support systems the following standards are used which are mentioned below:

- Examine large volumes of data
- Give answers to real-world business questions
- Execute queries of various operational requirements and complexities (e.g., ad-hoc, reporting, iterative OLAP, data mining)
o Are characterized by high CPU and IO load
o Are periodically synchronized with source OLTP databases through database maintenance functions
o Run on “Big Data” solutions, such as RDBMS as well as Hadoop/Spark based systems

• **TPC-E**: TPC Benchmark™ E (TPC-E) is a new On-Line Transaction Processing (OLTP) is the latest standard which is developed by the TPC. It uses the database to model a brokerage firm which includes the customers who generate transactions related to trades, account inquiries, and market research. The relevant account information is updated by the brokerage firm on the behalf of customer in order to execute orders with the financial markets.

To present the workloads of different size of business the standard named “scalable” is used, which defines the number of customer for brokerage firm. The standard is the combination of transaction which it has to maintain. TPC-E metric is given in transactions per second (tps) which records the number of trade-result transaction a server can hold over a period of time. Though the business model is of brokerage firm which includes the database schema, data population, transactions, and implementation rules have been designed to be broadly representative of modern OLTP systems.

• **TPC-H**: The TPC Benchmark™ H (TPC-H) is a decision support standard, which consists of a model for business oriented ad-hoc queries and concurrent data updation/modifications. For industry wide related queries and data populating the database has been selected. It describes number of decision support systems which examines huge volumes of data, execute queries with a high degree of complexity, and give answers to crucial business questions.

TPC-H is called the TPC-H Composite Query-per-Hour Performance Metric (QphH@Size) is the performance metric which reflects number of aspects capabilities of the system to process queries. The aspects are selected database size against which the queries are executed, the query processing power when queries are submitted by a single stream, and the query throughput when queries are submitted by multiple concurrent users. $/QphH@Size is expressed in TPC-H Price/Performance metric.

• **TPC-VMS**: For adding the methodology and requirement of running and reporting performance metrics for virtualized database is done by TPC-C, TPC-E, TPC-H and TPC-DS standards are used which is known as TPC-Virtual Measurement Single System specification. The main aim of TPC-VMS is to show a virtualization environment which consist of three database
workloads which are combined into one server. Out of any standard workload TPC-C, TPC-E, TPC-H, or TPC-DS with test sponsors selects any one, which run single instance of the standard work load on three individual virtual machines (VMs) on the system under test.

The TPC-VMS Primary Performance Metric is the minimum value of the three TPC Benchmark Primary metrics for the TPC Benchmarks running in the Virtualization Environment.

- **TPCx-BB:** TPCx-BB is an Express Benchmark for measuring the performance of Hadoop based Big Data systems system. The performance of hardware and software components are measured by running 30 consecutively running analytical queries in relation to retailers along with physical and online store. Structured queries are written in SQL and machine learning algorithms for structured and semi-structured data. Hive or spark is used for SQL queries and machine learning algorithms use machine learning libraries, user defined functions, and procedural programs.

- **TPCx-HS:** The most crucial part of enterprise IT ecosystem is big data technologies like Hadoop. TPCx-HS was developed to provide an objective prediction of hardware, operating system and commercial Apache Hadoop File System API compatible software distributions, and to provide the industry with verifiable performance, price-performance and availability metrics. The standard model is available 24 hours a day, 7 days a week. The developed model is very simple and predicts very high and relevant result of hardware and software which are dealing with big data systems in general. Hadoop run-time, Hadoop File system API compatible systems and MapReduce layers are included in both hardware and software which are stressed by TPCx-HS. The workload generated by using above model can be used to predict a wider range of system topologies and implementation of Hadoop clusters. TPCx-HS can be used for predicting a broad range of system topologies and implementation of methodologies in a technically rigorous and directly comparable, in a vendor-neutral manner.

- **TPCx-V:** The TPCx-V benchmark was developed to predict the performance of servers which are running database workloads on virtual machines. TPCx-V is freely available for simulating the multi-tenancy phenomena of elasticity, capacity planning, end to end benchmarking kit is used, load variation in cloud data centres. TPCx-V kit was developed completely from scratch in Java and C++ with PostgreSQL as the target database.

**Supporting TPC Benchmarks**

- **TPC-Energy:** TPC-Energy Specification consist of rules and methodology for predicting and reporting energy metric. It consists of system components
which are related with typical business information technology environments having feature listed below:

- Energy consumption of servers
- Energy consumption of disk systems
- Energy consumption of other items that consume power and are required by the benchmark specification.

The measuring and publishing of the TPC-Energy Metrics in the TPC Benchmarks are optional and are not required to publish a TPC Result.

- **TPC-Pricing**: For the revision of the existing pricing recommended methodology TPC Pricing Subcommittee was chartered to recommend revisions so that prices used in results are verifiable. In the due course, a decision was made to design a single pricing specification which is consistent for all TPC benchmark.

**SPEC System-Level File Server, Mail, and Web Benchmarks**

The SPEC standard is best known for its characterization of processor performance which is developed as a standard for file servers, mail servers, and Web servers. For synthetic standard, seven companies have agreed for prediction of systems running the Sun Microsystems network file service (NFS).

---

**Check Your Progress**

3. What are the different categories of faults?

4. What is throughput related to the server?

---

**13.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS**

1. There are two types of storage system i.e. volatile and non-volatile.

2. A Logical Unit Number (LUN) is an exclusive identifier which differentiates between devices which shares similar SCSI bus. Based on LUNs commands are send to the SCSI controller to identify devices.

3. Gray and Siewiorek divide the faults into four categories according to their cause.
   
   (i) Hardware faults
   (ii) Design faults
   (iii) Operation faults
   (iv) Environmental faults

4. Throughput is the average number of tasks completed by the server over a time period.
13.7 SUMMARY

- Storage is a process by which digital data can be saved into data storage device using computing technology. Storage is a structure which enables a computer to store data, either temporarily or permanently.
- Storage devices are the most important component devices. It stores virtually on the computer all types of data and applications rather on hardware firmware.
- A disk drive, hard drive or hard disk drive stores and gives frequent and quick access to large amounts of data on an electromagnetically charged surface or set of surfaces.
- Flash memory and an integrated Universal Serial Bus (USB) interface are the data storage that is in USB flash drive.
- Nowadays disk industry improving the disk capacity. This improvement can be measured in bits per square inch, can be expressed in areal density.
- A disk array which is a hardware element consists of huge group of hard disk drives (HDDs). A system is run which coordinates activity in the unit using the storage controller.
- Logical Unit Number (LUN) is an exclusive identifier which differentiates between devices which shares similar SCSI bus.
- Throughput is the average number of tasks completed by the server over a time period.

13.8 KEY WORDS

- **Primary Devices:** These devices are smaller in size and are designed to store data temporarily and are internal to the computer.
- **Secondary Devices:** These devices have huge storage capacity and can store data permanently on the computer system.
- **Logical Unit Number (LUN):** It is an exclusive identifier which differentiates between devices which shares similar SCSI bus.

13.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

**Short Answer Questions**

1. What are the different types of storage devices?
2. What are the units of storage?
Introduction to Storage Systems

NOTES

3. What are the components of USB flash drive?
4. What do you understand by faults and failure?

Long Answer Questions

1. Explain the advance topics related to the disk storage.
2. What is RAID? Explain its various levels.
3. Explain the various measures of performance.

13.10 FURTHER READINGS

UNIT 14 ISSUES

Structure
14.0 Introduction
14.1 Objectives
14.2 A Little Queuing Theory
14.3 Crosscutting Issues
14.4 Designing and Evaluating an I/O System
14.5 The Internet Archive Cluster
14.6 Answers to Check Your Progress Questions
14.7 Summary
14.8 Key Words
14.9 Self Assessment Questions and Exercises
14.10 Further Readings

14.0 INTRODUCTION

In this unit, you will learn about the little queuing theory and crosscutting issues. A little queuing theory is used to solve the waiting problems that occur in our day to day routine. Cross-cutting concerns are parts of a program that rely on or must affect many other parts of the system. They form the basis for the development of aspects. Further, you will learn about the design and evaluating I/O system and internet archive cluster.

14.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain little queuing theory and little’s law
- Discuss the crosscutting issues
- Understand how to design and evaluate I/O system
- Understand the need for internet archive cluster

14.2 A LITTLE QUEUING THEORY

Queuing or waiting problems are solved using Queuing theory. The typical examples of queuing theory are as follows:

- Banks/supermarkets—which consist of waiting of service
- Computers—which waits for a response of the request
• Failure situations – which consider a failure to occur e.g. in a piece of machinery.
• Public transport – which including waiting for a train or a bus.

As queues are most common experience in day to day life. Queues are formed in case the resources are less, which leads to economic sense. For instance: number of supermarkets requires queues for avoiding problem. For generating balance between service for customers (short queues implying many servers) and economic considerations (not too many servers) we require to design queuing system. All queuing systems can be sub-divided into individual sub-systems which includes entities queuing for few activities which are shown below.

Now we can discuss individual subsystems which are dealing with customers queuing for service. To analyse the sub-system, we require information relating to:

• Arrival Process
  o Arriving of customers e.g. singly or in groups or batch or bulk arrivals.
  o Distribution of arrival time e.g. what is the probability distribution of time between successive arrivals and the inter arrival time distribution.
  o Existence of finite population of customers or effectively an infinite number of populations.

One of the simplest ways of arrival process is one where we completely track the regular arrivals (i.e. the same constant time interval between successive arrivals). Arrivals at random are mapped with a Poisson stream. In a Poisson stream consecutive customer arrives at intervals which are individually exponentially distributed. The Poisson stream is crucial as it is a simple mathematical model of several real-life queuing systems and is explained by a single parameter which is the average arrival rate. Some of arrival processes are scheduled arrivals; batch arrivals; and time dependent arrival rates (i.e. the arrival rate varies according to the time of day).

• Service Mechanism
  o A detail review of the resources needed for service to start.
  o The service time distribution (how long the service will take).
  o Availability of the number of servers.
  o Whether the each server has a separate queue (servers are in series) or one queue for all servers (parallel).
  o Whether a server can stop processing a customer to deal with another “emergency” customer (pre-emption is allowed).

By assuming the service times for customers are independent and which are not depended on the arrival process is very common. Other assumption is of service times which are exponentially distributed.
Queue Characteristics

- how, from the set of customers waiting for service, do we choose the one to be served next (e.g. FIFO (first-in first-out) - also known as FCFS (first-come first served); LIFO (last-in first-out); randomly) (this is often called the queue discipline)

- do we have:
  - Balking (customers deciding not to join the queue if it is too long)
  - Reneging (customers leave the queue if they have waited too long for service)
  - Jockeying (customers switch between queues if they think they will get served faster by so doing)
  - A queue of finite capacity or (effectively) of infinite capacity

Changing the queue discipline (the rule by which we select the next customer to be served) can often reduce congestion. Often the queue discipline “choose the customer with the lowest service time” results in the smallest value for the time (on average) a customer spends queuing.

Note here that integral to queuing situations is the idea of uncertainty in, for example, inter-arrival times and service times. This means that probability and statistics are needed to analyse queuing situations.

In terms of the analysis of queuing situations the types of questions in which we are interested are typically concerned with measures of system performance and might include:

- How long does a customer expect to wait in the queue before they are served, and how long will they have to wait before the service is complete?
- What is the probability of a customer having to wait longer than a given time interval before they are served?
- What is the average length of the queue?
- What is the probability that the queue will exceed a certain length?
- What is the expected utilisation of the server and the expected time period during which he will be fully occupied (remember servers cost us money so we need to keep them busy). In fact if we can assign costs to factors such as customer waiting time and server idle time then we can investigate how to design a system at minimum total cost.

These are questions that need to be answered so that management can evaluate alternatives in an attempt to control/improve the situation. Some of the problems that are often investigated in practice are:

- Is it worthwhile to invest effort in reducing the service time?
- How many servers should be employed?
• Should priorities for certain types of customers be introduced?
• Is the waiting area for customers adequate?

In order to get answers to the above questions there are two basic approaches:

• Analytic methods or queuing theory (formula based); and
• Simulation (computer based).

**Little’s Law**

Little law states the average number of tasks in the system based on the average arrival rate of new tasks, and the average time to perform a task:

\[
\text{Mean number of tasks in system} = \text{Arrival rate} \times \text{Mean response time}
\]

Little’s law applies to any system in equilibrium, as long as nothing inside the black box is creating new tasks or destroying them. Note that the arrival rate and the response time must use the same time unit; inconsistency in time units is a common cause of errors.

Assume we observe a system for Time\(_{\text{observe}}\) minutes. During that observation, we record how long it took each task to be serviced, and then sum those times. The number of tasks completed during Time\(_{\text{observe}}\) is Number\(_{\text{task}}\), and the sum of the times each task spends in the system is Time\(_{\text{accumulated}}\). Note that the tasks can overlap in time, so Time\(_{\text{accumulated}} \geq Time_{\text{observe}}\). Then,

\[
\text{Mean number of tasks in system} = \frac{\text{Time}_{\text{accumulated}}}{\text{Time}_{\text{observe}}}
\]

\[
\text{Mean response time} = \frac{\text{Time}_{\text{accumulated}}}{\text{Number}_{\text{task}}}
\]

\[
\text{Arrival rate} = \frac{\text{Number}_{\text{task}}}{\text{Time}_{\text{observe}}}
\]

**14.3 CROSSCUTTING ISSUES**

Following are the crosscutting issues.

**Point-to-Point Links and Switches Replacing Buses**

Point-to-point links and switches are increasing in popularity as Moore’s law continues to reduce the cost of components. Combined with the higher I/O bandwidth demands from faster processors, faster disks, and faster local area networks, the decreasing cost advantage of buses means the days of buses in desktop and server computers are numbered.
The number of bits and bandwidth for the new generation is per direction, so they double for both directions. Since these new designs use many fewer wires, a common way to increase bandwidth is to offer versions with several times the number of wires and bandwidth.

- **Moore’s Law** - Moore’s Law asserts that the number of transistors on a microchip doubles every two years, though the cost of computers is halved. In other words, we can expect that the speed and capability of our computers will increase every couple of years, and we will pay less for them.

**Block Servers versus Fillers**

Anyone who has used a Storage Area Network (SAN) has probably used block level storage earlier. Block level storage presents itself to servers using industry standard Fibre Channel and iSCSI connectivity mechanisms. In its most basic form, think of block level storage as a hard drive in a server except the hard drive happens to be installed in a remote chassis and is accessible using Fibre Channel or iSCSI.

When it comes to flexibility and versatility, you can’t beat block level storage. In a block level storage device, raw storage volumes are created, and then the server-based operating system connects to these volumes and uses them as individual hard drives. This makes block level storage usable for almost any kind of application, including file storage, database storage, virtual machine file system (VMFS) volumes, and more. You can place any kind of file system on block level storage. So, if you’re running Windows, your volumes will be formatted with NTFS; VMware servers will use VMFS.

With regard to management complexity, block-based storage devices tend to be more complex than their file-based counterparts; this is the trade-off you get for the added flexibility. Block storage device administrators must:

- Carefully manage and dole out storage on a per server basis.
- Manage storage protection levels (i.e., RAID).
- Track storage device performance to ensure that performance continues to meet server and application needs.
- Manage and monitor the storage communications infrastructure (generally iSCSI or Fibre Channel).

From a use case standpoint, there are a lot of applications that make use of this block-level shared storage, including:

- **Databases**: This is especially true when you want to cluster databases, since clustered databases need shared storage.
**NOTES**

- **Exchange**: Although Microsoft has made massive improvements to Exchange, the company still does not support file level or network-based (as in, CIFS or NFS) storage. Only block level storage is supported.

- **VMware**: Although VMware can use file level storage via Network File System (NFS), it’s very common to deploy VMware servers that use shared VMFS volumes on block level storage.

- **Server boot**: With the right kind of storage device, servers can be configured to boot from block level storage.

- **Server boot**: With the right kind of storage device, servers can be configured to boot from block level storage.

**File level storage**

Although block level storage is extremely flexible, nothing beats the simplicity of file level storage when all that’s needed is a place to dump raw files. After all, simply having a centralized, highly available, and accessible place to store files and folders remains the most critical need in many organizations. These file level devices are usually Network Attached Storage (NAS) devices that provide a lot of space at what is generally at lower cost than block level storage.

File level storage is usually accessible using common file level protocols such as SMB/CIFS (Windows) and NFS (Linux, VMware). In the block level world, you need to create a volume, deploy an OS, and then attach to the created volume; in the file level world, the storage device handles the files and folders on the device. This also means that, in many cases, the file level storage device or NAS needs to handle user access control and permissions assignment. Some devices will integrate into existing authentication and security systems.

File level use cases are generally:

- **Mass file storage**: When your users simply need a place to store files, file-level devices can make a lot of sense.

- **VMware (think NFS)**: VMware hosts can connect to storage presented via NFS in addition to using block level storage.

### 14.4 DESIGNING AND EVALUATING AN I/O SYSTEM

In designing an I/O system, we analyse performance, cost, capacity, and availability using varying I/O connection schemes and different numbers of I/O devices of each type. Here is one series of steps to follow in designing an I/O system. The
answers for each step may be dictated by market requirements or simply by cost, performance, and availability goals.

(i) List the different types of I/O devices to be connected to the machine, or list the standard buses and networks that the machine will support.

(ii) List the physical requirements for each I/O device. Requirements include size, power, connectors, bus slots, expansion cabinets, and so on.

(iii) List the cost of each I/O device, including the portion of cost of any controller needed for this device.

(iv) List the reliability of each I/O device.

(v) Record the processor resource demands of each I/O device. This list should include:
   - Clock cycles for instructions used to initiate an I/O, to support operation of an I/O device (such as handling interrupts), and to complete I/O
   - Processor clock stalls due to waiting for I/O to finish using the memory, bus, or cache
   - Processor clock cycles to recover from an I/O activity, such as a cache flush

(vi) List the memory and I/O bus resource demands of each I/O device. Even when the processor is not using memory, the bandwidth of main memory and the I/O connection is limited.

(vii) The final step is assessing the performance and availability of the different ways to organize these I/O devices. When you can afford it, try to avoid single points of failure. Performance can only be properly evaluated with simulation, although it may be estimated using queuing theory. Reliability can be calculated assuming I/O devices fail independently and that the times to failure are exponentially distributed. Availability can be computed from reliability by estimating MTTF for the devices, taking into account the time from failure to repair.

14.5 THE INTERNET ARCHIVE CLUSTER

Internet archive is a non-profit digital library with a mission universal access to knowledge. It provides free public access to collections of digitized materials, including websites, software applications/games, music, movies/videos, moving images, and millions of public-domain books. The Internet archive also serves as a tool for scientists, security agencies, historians (for example, archaeologists) and representatives of many other fields, not to mention individual users. The creator

NOTES
of the “Archive” is Brewster Cale from the US, who created the company Alexa Internet. Both of his services have become extremely popular, both of them are still prosperous.

The Internet archive has started to archive the information from the websites and to keep the copies of the web pages in 1996. The headquarters of this non-profit organization is located in San Francisco, USA.

However, for five years the data were unavailable for public access. The data was stored on the servers of the “Archive”, and that’s all, only the administration of the service could view the old copies of the sites. Since 2001, the service administration has decided to provide access to the stored data to everyone.

In the beginning, the “Internet archive” was just a web archive, but then the organization started saving books, audio files, moving images, software. Now the “Internet archive” acts as a repository for photos and other images of NASA, open Library texts, etc.

How does the organization exist?

The “Archive” exists on voluntary donations both from the organizations and from the individuals. You can provide support in bitcoins; the wallet number is 1Archive1n2C579dMsAu3Cf6WzuQjz8dN. This wallet, by the way, has received 357.47245292 BTC during its existence, which is about $2.25 million at the current rate.

How does “Archive” work?

Most of the staff is employed in the book scanning canters, doing routine, but rather time-consuming work. The organization has three data canters located in California, USA. One in San Francisco, one in the Redwood city, one in Richmond. In order to avoid the risk of data loss in the event of a natural disaster or other catastrophes, the “Archive” has spare capacity in Egypt and Amsterdam.

Millions of people have spent a lot of time and effort to share with others what we know in the form of the Internet. We want to create a library for this new publishing platform, said Brewster Kahle, the founder of the Internet Archive.

How big is the “Archive” now?

The “Internet archive” has several divisions, and the one that collects information from the sites has its own name Wayback Machine. At the time of writing the “Inquiry”, the archive contained 339 billion saved web pages. In 2017, the “Archive” stored 30 petabytes of information, which is about 300 billion web pages,
12 million books, 4 million audio recordings, 3.3 million videos, 1.5 million photos and 170 thousand different software distributions. In just a year, the service significantly “added weight”. Now the “Archive” stores 339 billion web pages, 19 million books, 4.5 million video files, 4.7 million audio files, 3.2 million images of various kinds and 381 thousand software distributions.

**How is the data storage organized?**

The information is stored on hard drives in the so-called “data nodes”. These are the servers. Each of them contains 36 hard drives (plus two operating system drives). Data nodes are grouped into arrays of 10 machines and represent a cluster storage. In 2016, the “Archive” used 8-terabyte HDD, now the situation is about the same. It turns out that one node stores about 288 terabytes of data. In general, the hard drives of other sizes are also used: 2.3 and 4 TB. In 2016, there were about 20,000 hard drives. The data centres of the “Archive” are equipped with air conditioning units for climate control with constant characteristics. A clustered storage of 10 nodes consumes about 5 kilowatts of energy. The structure of the Internet Archive is a virtual “library”, which is divided into sections such as books, movies, music, etc. For each element there is a description in the catalogue — usually the name, the author’s name and additional information. From a technical point of view, the elements are structured and located in Linux directories. The total amount of data stored by the “Archive” is 22 PB, and now there is room for another 22 PB. “Because we are paranoid,” — state the representatives of the service.

**Estimating Performance, Dependability, and Cost of the Internet Archive Cluster**

To illustrate how to evaluate an I/O system, we’ll make some guesses about the cost, performance, and reliability of the components of this cluster. We make the following assumptions about cost and performance:

- The VIA processor, 512 MB of DDR266 DRAM, ATA disk controller, power supply, fans, and enclosure cost $500.
- Each of the four 7200 RPM Parallel ATA drives holds 500 GB, has an average time seek of 8.5 ms, transfers at 50 MB/sec from the disk, and costs $375. The PATA link speed is 133 MB/sec.
- The 48-port 10/100/1000 Ethernet switch and all cables for a rack cost $3000.
- The performance of the VIA processor is 1000 MIPS.
- The ATA controller adds 0.1 ms of overhead to perform a disk I/O.
• The operating system uses 50,000 CPU instructions for a disk I/O.
• The network protocol stacks use 100,000 CPU instructions to transmit a data block between the cluster and the external world.
• The average I/O size is 16 KB for accesses to the historical record via the Wayback interface, and 50 KB when collecting a new snapshot.

Check Your Progress
1. What does Little’s Law state?
2. What does Moore’s Law assert?
3. What is internet archive?

14.6 ANSWER TO CHECK YOUR PROGRESS QUESTIONS
1. Little Law states the average number of tasks in the system based on the average arrival rate of new tasks, and the average time to perform a task.
2. Moore’s Law asserts that the number of transistors on a microchip doubles every two years, though the cost of computers is halved.
3. Internet archive is a non-profit digital library with a mission universal access to knowledge.

14.7 SUMMARY
• Queues are formed in case the resources are less, which leads to economic sense. For instance: number of supermarkets requires queues for avoiding problem.
• All queuing systems can be divided into individual sub-systems which includes entities queuing for few activities.
• Little Law states the average number of tasks in the system based on the average arrival rate of new tasks, and the average time to perform a task.
• Point-to-point links and switches are increasing in popularity as Moore’s law continues to reduce the cost of components.
• Moore’s Law asserts that the number of transistors on a microchip doubles every two years, though the cost of computers is halved.
• File level storage is usually accessible using common file level protocols such as SMB/CIFS (Windows) and NFS (Linux, VMware).
• In designing an I/O system, we analyze performance, cost, capacity, and availability using varying I/O connection schemes and different numbers of I/O devices of each type.
• Internet archive is a non-profit digital library with a mission universal access to knowledge.
• The archive exists on voluntary donations both from the organizations and from the individuals.
• The «Internet archive» has several divisions, and the one that collects information from the sites has its own name Wayback Machine.

14.8 KEY WORDS

• Little’s Law: It states that the long-term average number of customers in a stable system is equal to the long-term average effective arrival rate multiplied by the average time a customer spends in the system.
• Internet Archive: It is a non-profit digital library with a mission universal access to knowledge.

14.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions
1. What do you understand by a little queuing theory?
2. Discuss the queue characteristics.
3. Differentiate between block and file level storage.

Long Answer Questions
1. What information is required to analyze the sub-system? Explain.
2. Explain the little’s law.
3. What do you understand by cross cutting issues?
4. What are the steps for designing and evaluating an I/O system?
5. Write a detailed note on internet archive cluster.
14.10 FURTHER READINGS


