NATIONAL OPEN UNIVERSITY OF NIGERIA

FACULTY OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

COURSE CODE: CIT317

COURSE TITLE:
COMPUTER ARCHITECTURE AND ORGANIZATION II

COURSE GUIDE

COURSE CODE: CIT317

COURSE TITLE: Computer Architecture and Organization II

COURSE TEAM: Dr. Udoinyang Godwin Inyang and Dr. Francis B. Osang (Developers)

University of Uyo and National Open University of Nigeria

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria Headquarters
University Village, Plot 91 Cadastral Zone, Nnnamdi Azikiwe Expressway
Jabi, Abuja

LAGOS OFFICE
14/16 Ahmadu Bello Way
Victoria Island , Lagos

E-Mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

PUBLISHED BY:
National Open University of Nigeria

First Printed 2022

ISBN: 978-058-851-5

CONTENTS **PAGE**

Introduction

COURSE GUIDE

INTRODUCTION

CIT 317 –Computer Architecture and Organisation II –is a 3- credit unit course. Keeping pace with technological change is an issue for all computing courses and texts. Systems which seemed capable of holding their advanced position within the market-place for several years, are now overtaken within months of launch. Software tools are being developed and adopted by commercial programmers long before universities have had a chance. We all learn differently, but the ability to use text effectively has been at the core of modern civilization for a long time. We all benefit so much from people's experience recorded on paper for others to read. Ignoring this vast resource is deliberately handicapping yourself. Life is difficult enough without conceding an unnecessary penalty! If anything, the introduction of the WWW has placed even greater literacy demands on everyone. Most Web pages presenting useful information still depend heavily on text. A picture may be worth a thousand words, but it is often the accompanying text that gives you a first glimmer of understanding.

This book is about the structure and function of computers. Its purpose is to present as clearly and completely as possible, the nature and characteristics of modern-day computer systems.

This task is challenging for several reasons. First, there is a tremendous variety of products that can rightly claim the name of computer, from single-chip microprocessors costing a lot of dollars to supercomputers costing tens of millions of dollars. Variety is exhibited not only in cost, but in size, performance, and application. Second, the rapid pace of change that has always characterized computer technology continues with no letup. These changes cover all aspects of computer technology, from the underlying integrated circuit technology used to construct computer components, to the increasing use of parallel organization concepts in combining those components.

In spite of the variety, and pace of change in the computer field. certain fundamental concepts apply consistently throughout. The application of these concepts depends on the current state of the technology and the price, performance, objectives of the designer. The intent of this book is to provide a thorough discussion of the fundamentals of computer organization and architecture and to relate these to contemporary design issues.

It is a course for B.Sc. Computer Science major students, and is normally taken in the third year of the programme duration. It should therefore appeal to whosoever is concerned with the understanding of the basic computer architecture and its organization.

This course is divided into four modules. The first module deals with the overview of the memory system. The second module covers, memory addressing, elements of memory hierarchy, and virtual memory control systems. The third module discusses various forms of control including hardware, asynchronous, microprogrammed and asynchronous forms. The fourth and last module takes on fault tolerant computing and methods for fault tolerant computing.

This course guide gives you a brief overview of the course contents, course duration, and course materials.

COURSE COMPETENCIES

First, students will learn about the basicity of computers and what they are made up of. Second, they will be able to judge certain functionalities in computer systems dependent on the type of architectures they are operating on. This in turn will give them a deeper understanding on how to manage computer faults.

In general, this course is designed to aid them take on a path involving how a computer system is built to operate.

COURSE OBJECTIVES

Certain objectives have been set out for the achievement of the course aims. And apart from the course objectives, each unit of this course has its objectives, which you need to confirm if are met, at the end of each unit. So, upon the completion of this course, you should be able to:

- Describe how computer memories function and how they can be optimized
- Explain major functions and techniques involving architecture designing and study
- Explain methods to tolerate faults in computer architectures
- Explain methods to optimize control in computer systems

WORKING THROUGH THIS COURSE

In order to have a thorough understanding of the course units, you will need to read and understand the contents, practice the steps and techniques involved in the task of computer architectures and organization and its involvement in development of various segments of computer systems.

This course is designed to cover approximately seventeen weeks, and requires your devoted attention, answering the exercises in the tutor- marked assignments and gets them submitted to your tutors.

STUDY UNITS

There are 10 units in this course:

MODULE ONE

UNIT ONE: Memory system

1.1 Main Memories

1.2 Auxiliary Memories

1.3 Memory Access Methods

1.4 Memory Mapping and Virtual Memories

MODULE TWO

MODULE THREE

MODULE FOUR

UNIT ONE: Fault Tolerant Computing

1.1 What is Fault Tolerance

1.2 Fault Tolerant Systems

1.3 Hardware and Software Fault Tolerant Issues

1.4 Fault Tolerance VS High Availability

1.5 Redudancy

1.6 Relationship Between Security and Fault Tolerance

UNIT TWO: Methods for Fault Tolerant Computing

2.1 Fault Detection Methods

2.2 Fault Tolerance Architecture

2.3 Fault Models

2.4 Fault Tolerance Methods

2.5 Major Issues in Modelling and Evaluation

2.6 Fault Tolerance for Web Applications

You should make use of the course materials, and do the exercises to enhance your learning.

REFERENCES AND FURTHER READINGS

Amma A. D. T., Pramod V. R and N. Radhika, (2012) “ISM for Analyzing the Interrelationship between the Inhibitors of Cloud Computing”, vol. 2, No. 3.

Anderson T. and Knight J. C. (1983), “A Framework for software Fault tolerance in Real time System”, IEEE Transaction on software Engineering, Vol. 9, No.3.

Engineering Safety Requirements, Safety Constraints, and Safety Critical Requirements, Available at: http://www.jot.fm/issues/issue_2004_03/column3/ last visit November 17, 2021.

Fred B. Schneider (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. A.C.M. Computing Surveys, 22(4):299–319.

Johnson, B. W. (1996). An introduction to the design and analysis of fault-tolerant systems. *Fault-tolerant computer system design*, *1*, 1-84.

Kim, E. P., & Shanbhag, N. R. (2012). Soft N-modular redundancy. IEEE Transactions on Computers, 61(3), 323–336.

Lyu, M. and Mendiratta V, (1999) "Software Fault Tolerance in a Clustered Architecture: Techniques and Reliability Modeling," In Proceedings' of IEEE Aerospace Conference, Snowmass, Colorado, vol.5, pp.141-150, 6-13.

Neuman, P (2000) "Practical Architecture for survivable system and networks", Phase Two Project 1688, SRI International, Menlo Park, California.

Patton, R. J. (2015). Fault-tolerant control. Encyclopedia of systems and control, 422–428.

John L. Hennessy and David A. Patterson (2012) Computer Architecture; A Qualitative Approach. Fifth (Ed.), Library of Congress Cataloging in Publication Data.

William Stallings (2019) Computer Organization and Architecture; Designing for Performance. 11 (Ed.), Pearson.

PRESENTATION SCHEDULE

The Presentation Schedule included in your course materials gives you the important dates for the completion of tutor marked assignments and attending tutorials. Remember, you are required to submit all your assignments by the due date. You should guard against lagging behind in your work.

ASSESSMENT

There are two aspects to the assessment of the course. First are the tutor –marked assignments; second, is a written examination. In tackling the assignments, you are expected to apply the information and knowledge you acquired during this course. The assignments must be submitted to your tutor for formal assessment in accordance with the deadlines stated in the Assignment File. The work you submit to your tutor for assessment will count for 30% of your total course mark. At the end of the course, you will need to sit for a final three-hour examination. This will also count for 70% of your total course mark.

Tutor-Marked Assignment

There are eight tutor- marked assignments in this course. You need to submit all the assignments. The total marks for the best four (4) assignments will be 30% of your total course mark.

Assignment questions for the units in this course are contained in the Assignment File. You should be able to complete your assignments from the information and materials contained in your set textbooks and study units. However, you may wish to use other references to broaden your viewpoint and provide a deeper understanding of the subject.

When you have completed each assignment, send it together with a form to your tutor. Make sure that each assignment reaches your tutor on or before the deadline given. If however you cannot complete your work on time, contact your tutor before the assignment is done to discuss the possibility of an extension.

Final Examinations and Grading

The final examination for the course will carry 70% percentage of the total mark available for this course. The examination will cover every aspect of the course, so you are advised to revise all your corrected assignments before the examination.

This course endows you with the status of a teacher and that of a learner. This means that you teach yourself and that you learn, as your learning capabilities would allow. It also means that you are in a better position to determine and to ascertain the what, the how, and the when of your language learning. No teacher imposes any method of learning on you.

The course units are similarly designed with the introduction following the table of contents, then a set of objectives and then the discourse and so on. The objectives guide you as you go through the units to ascertain your knowledge of the required terms and expressions.

Course Marking Scheme

This table shows how the actual course marking is broken down.

Assessment	**Marks**
Assignment 1- 4	Four assignments, best three marks of the four count at 30% of course marks
Final Examination	70% of overall course marks
Total	100% of course marks

How to Get the Best from this Course

In distance learning the study units replace the university lecturer. This is one of the great advantages of distance learning; you can read and work through specially designed study materials at your own pace, and at a time and place that suit you best. Think of it as reading the lecture instead of listening to a lecturer. In the same way that a lecturer might set you some reading to do, the study units tell you when to read your set books or other material. Just as a lecturer might give you an in-class exercise, your study units provide exercises for you to do at appropriate points.

Each of the study units follows a common format. The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next is a set of learning objectives. These objectives enable you know what you should be able to do by the time you have completed the unit. You should use these objectives to guide your study. When you have finished the units you must go back and check whether you have achieved the objectives. If you make a habit of doing this you will significantly improve your chances of passing the course.

Remember that your tutor's job is to assist you. When you need help, don't hesitate to call and ask your tutor to provide it.

1. Read this Course Guide thoroughly.
2. Organize a study schedule. Refer to the Course Overview for more details. Note the time you are expected to spend on each unit and how the assignments relate to the units. Whatever method you chose to use, you should decide on it and write in your own dates for working on each unit.

3. Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they lag behind in their course work.
4. Turn to Unit 1 and read the introduction and the objectives for the unit.
5. Assemble the study materials. Information about what you need for a unit is given in the overview at the beginning of each unit. You will almost always need both the study unit you are working on and one of your set of books on your desk at the same time.
6. Work through the unit. The content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit you will be instructed to read sections from your set books or other articles. Use the unit to guide your reading.
7. Review the objectives for each study unit to confirm that you have achieved them. If you feel unsure about any of the objectives, review the study material or consult your tutor.
8. When you are confident that you have achieved a unit's objectives, you can then start on the next unit. Proceed unit by unit through the course and try to pace your study so that you keep yourself on schedule.
9. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor-marked assignment form and on the assignment. Consult your tutor as soon as possible if you have any questions or problems.
10. After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this Course Guide).

Facilitators/Tutors and Tutorials

There are 15 hours of tutorials provided in support of this course. You will be notified of the dates, times and location of these tutorials, together with the name and phone number of your tutor, as soon as you are allocated a tutorial group.

Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance for you during the course. You must mail or submit your tutor-marked assignments to your tutor well before the due date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

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

- you do not understand any part of the study units or the assigned readings,
- you have difficulty with the self-tests or exercises,
- you have a question or problem with an assignment, with your tutor's comments on an assignment or with the grading of an assignment.

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

Summary

Introduction to Computer Organization, as the title implies, introduces you to the fundamental concepts of how the computer system operates internally to perform the basic tasks required of it by the end-users. Therefore, you should acquire the basic knowledge of the internal workings of the components of the computer system in this course. The content of the course material was planned and written to ensure that you acquire the proper knowledge and skills in order to be able to programme the computer to do your bidding. The essence is to get you to acquire the necessary knowledge and competence and equip you with the necessary tools.

We wish you success with the course and hope that you will find it interesting and useful.

COURSE INFORMATION

COURSE CODE: CSC 317

COURSE TITLE: Computer Architecture and Organization II

CREDIT UNIT: 3 Units

COURSE STATUS:

COURSE BLURB: This course covers principal topics in understanding the general concepts of computer architectures and organization. This will include the basics involved in memory addressing and hierarchy management, control systems, fault tolerant and virtual memory systems.

SEMESTER: First Semester
Course Duration: Required Hours for Study

COURSE TEAM
Course Developer: Dr. Udoinyang Godwin Inyang
` Associate Professor
Department of Computer Science
University of Uyo

Course Writer: Dr. Udoinyang Godwin Inyang
Associate Professor
Department of Computer Science
University of Uyo

Content Editor:

Instructional Designer:

Learning Technologists:

Copy Editor

MODULE ONE
Introduction to Computer architecture and Organization

UNIT 1: Introduction

1.0 Introduction

Computer Organization is concerned with the structure and behaviour of a computer system as seen by the user. It acts as the interface between hardware and software. Computer architecture refers to those attributes of a system visible to a programmer, or put another way, those attributes that have a direct impact on the logical execution of a program. Computer organization refers to the operational units and their interconnection that realize the architecture specification.

Examples of architecture attributes include the instruction set, the number of bit to represent various data types (e.g.., numbers, and characters), I/O mechanisms, and technique for addressing memory.

Examples of organization attributes include those hardware details transparent to the programmer, such as control signals, interfaces between the computer and peripherals, and the memory technology used.

As an example, it is an architectural design issue whether a computer will have a multiply instruction. It is an organizational issue whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system. The organization decision may be bases on the anticipated frequency of use of the multiply instruction, the relative speed of the two approaches, and the cost and physical size of a special multiply unit.

Historically, and still today, the distinction between architecture and organization has been an important one. Many computer manufacturers offer a family of computer model, all with the same architecture but with differences in organization. Consequently, the different models in the family have different price and performance characteristics. Furthermore, an architecture may survive many years, but its organization changes with changing technology.

1.2. Structure and Function

A computer is a complex system; contemporary computers contain millions of elementary electronic components. How, then, can one clearly describe them? The key is to recognize the hierarchical nature of most complex system. A hierarchical system is a set of interrelated subsystem, each of the later, in turn, hierarchical in structure until we reach some lowest level of elementary subsystem.

The hierarchical nature of complex systems is essential to both their design and their description. The designer need only deal with a particular level of the system at a time. At each level, the system consists of a set of components and their interrelationships. The behavior at each level depends only on a simplified, abstracted characterization of the system at the next lower level. At each level, the designer is concerned with structure and function:

- **Structure:** The way in which the components are interrelated.
- **Function:** The operation of each individual component as part of the structure.

In term of description, we have two choices: starting at the bottom and building up to a complete description, or beginning with a top view and decomposing the system, describing their structure and function, and proceed to successively lower layer of the hierarchy. The approach taken in this course follows the latter.

1.2.1 Function

In general terms, there are four main functions of a computer:

- Data processing
- Data storage
- Data movement
- Control

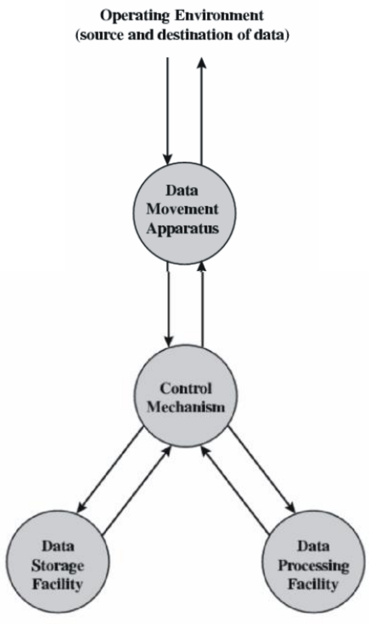


Figure 1.1: A functional view of the computer

The computer, of course, must be able to process data. The data may take a wide variety of forms, and the range of processing requirements is broad. However, we shall see that there are only a few fundamental methods or types of data processing.

It is also essential that a computer store data. Even if the computer is processing data on the fly (i.e., data come in and get processed, and the results go out immediately), the computer must temporarily store at least those pieces of data that are being worked on at any given moment. Thus, there is at least a short-term data storage function. Files of data are stored on the computer for subsequent retrieval and update.

The computer must be able to move data between itself and the outside world. The computer’s operating environment consists of devices that serve as either sources or destinations of data. When data are received from or delivered to a device that is directly connected to the computer, the process is known as input-output (I/O), and the device is referred to as a peripheral. When data are moved over longer distances, to or from a remote device, the process is known as data communications.

Finally, there must be control of these three functions. Ultimately, this control is exercised by the individual who provides the computer with instructions. Within the computer system, a control unit manages the computer’s resources and orchestrates the performance of its functional parts in response to those instructions.

At this general level of discussion, the number of possible operations that can be performed is few. The computer can function as a data movement device, simply transferring data from one peripheral or communications line to another. It can also function as a data storage device, with data transferred from the external environment to computer storage (read) and vice versa (write). The final two diagrams show operations involving data processing, on data either in storage or in route between storage and the external environment.

1.2.2 Structure

Figure 1.2 is the simplest possible depiction of a computer. The computer is an entity that interacts in some fashion with its external environment. In general, all of its linkages to the external environment can be classified as peripheral devices or communication lines. We will have something to say about both types of linkages.

- **Central Processing Unit (CPU):** Controls the operation of the computer and performs its data processing functions. Often simply referred to as processor.
- **Main Memory:** Stores data.
- **I/O:** Moves data between the computer and its external environment.
- **System Interconnection:** Some mechanism that provides for communication among CPU, main memory, and I/O.

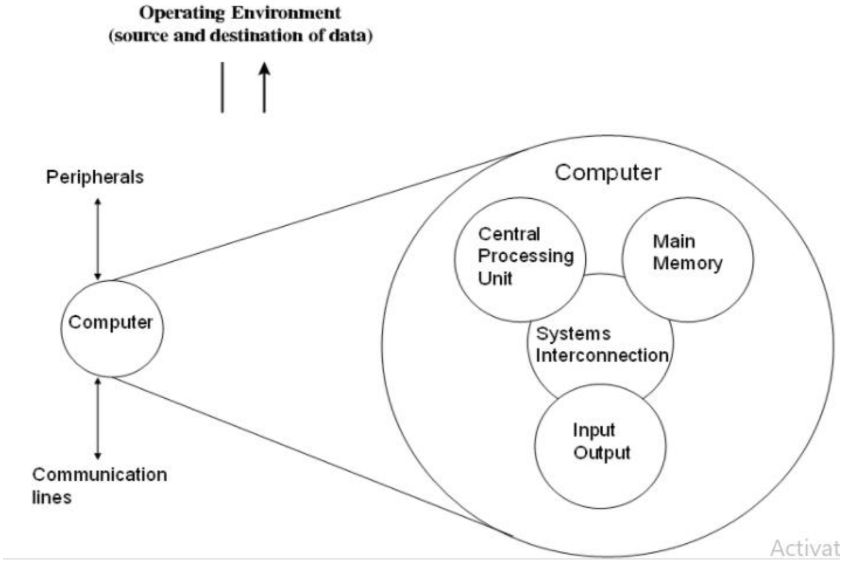


Figure 1.2: Top Level structure of the computer

There may be one or more of each of the above components. Traditionally, there has been just a single CPU. In recent years, there has been increasing use of multiple processors, in a single system. Each of these components will be examined in some detail in later lectures. However, for our purpose, the most interesting and in some ways the most complex component is the CPU; its structure is depicted in Figure 1.2. Its major structural components are:

- **Control Unit (CU):** Controls the operation of the CPU and hence the computer.
- **Arithmetic and Logic Unit (ALU):** Performs computer's data processing functions.
- **Register:** Provides storage internal to the CPU.
- **CPU Interconnection:** Some mechanism that provides for communication among the control unit, ALU, and register.

Each of these components will be examined in some detail in subsequent units.

4.0 CONCLUSION

In this unit, you have learnt the main difference between computer organization and computer architecture. The structures of the computer

Tutor Marked Assignment

1. Differentiate between Computer organization and Computer Architecture
2. Give a suitable diagram showing the top level view of the computer.
3. State the functions of the following components of the computer

   i) Register ii) Control Unit ii) Processor

UNIT TWO: MEMORY SYSTEM

1.0 INTRODUCTION

Although seemingly simple in concept, computer memory exhibits perhaps the: widest range of type. Technology, organization. Performance, and cost of any feature of a computer system. No one technology is optimal in satisfying the memory requirements for a computer system. As a consequence, the typical computer system is equipped with a hierarchy of memory subsystems, some internal to the system (directly accessible by the processor) and some external (accessible by the processor via an 110 module).

The memory unit is an essential component in any digital computer since it is needed for storing programs and data. A very small computer with aN unlimited application may be able to fulill its intended task without the need of additional storage capacity. Most general purpose computers would run more efficiently if they were equipped with additional storage beyond the capacity of the main memory. There is just not enough space in one memory unit to accommodate all the progams used in a typical computer. The memory unit that communicates directly with the CPU is called the main memory. Devices that provide backup storage are called auxiliary memory. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system, programs, large data, and other backup information. Only programs and data currently needed by the processor reside in main memory. A special very-high speed memory called a Cache is sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate. The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic.

Characteristics of Memory Systems

The complex subject of computer memory is made more manageable if we classify memory systems according to their key characteristics. Internal memory is often equated with main memory. But there are other forms of internal memory. The processor requires its own local memory, in the form of registers. The control unit portion of the processor may also require its own internal memory. Cache is another form of internal memory. External memory consists of peripheral storage devices, such as disk and tape, that arc accessible to the processor via I/O. An obvious characteristic of memory is its **capacity.** For internal memory, this is typically expressed in terms of bytes (I byte- = 1024 bits) or words. Common word lengths are 8, 16, and 32 bits. External memory capacity is typically expressed in terms of bytes.

A related concept is the **unit of transfer,** for internal memory, the unit of transfer is equal to the number of data lines into and out of the memory module. This may be equal to the word length, but is often larger. such as 64. 128, or 256 bits. From a user's point of view, the two most important characteristics of memory are capacity and **performance.** Three performance parameters arc used: a. second access can commence. This additional time may be required for transients to die out on signal lines or to regenerate data if they are read destructively. Now that memory cycle time is concerned with the system bus, not the processor.

- **Transfer rate**: This is the rate at which data can be transferred into or out of a memory unit.
- **Access time (latency):** For random-access memory, this is the time it takes to perform a read or write operation. That is, the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use. For non-random-access memory, access time is the time it takes to position the read—write mechanism at the desired location.

Memory cycle time: This concept is primarily applied to random-access memory and consists of the access time plus any additional time required before

2.0 OBJECTIVES

At the end of this module, the user should be able to discuss elaborately on;

1. Memory types and their functionalities
2. The history of memory devices
3. Modes to augment processing
4. Access methods
5. Pipelining

3.1 MAIN MEMORIES

The main memory is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation. The principal technology used for the main memory is based on semiconductor integrated circuits. Integrated circuit RAM chips are available in two possible operating modes, static and dynamic. The static RAM consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are provided inside the chip by MOS transistors. The stored charge on the capacitors tend to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory. Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles. Most of the main memory in a general-purpose computer is made up of RAM integrated circuit chips, but a portion of the memory may be constructed with ROM chips. Originally, RAM was used to refer to a random-access memory, but now it is used to designate a read/write memory to distinguish it from a read-only memory, although ROM is also random access. RAM is used for storing the bulk of the programs and data that are subject to change. ROM is used for storing programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed. Among other things, the ROM portion of main memory is needed for storing an initial program called a bootstrap loader. The bootstrap loader is a program whose function is to start the computer software operating when power is turned on. Since RAM is volatile, its contents are destroyed when power is turned off. The contents of ROM remain unchanged after power is turned off and on again. The startup of a computer consists of turning the power on and starting the execution of an initial program. Thus when power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader. The bootstrap program loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use.

3.2 AUXILIARY MEMORIES

The primary types of Auxiliary Storage Devices are:

- Magnetic tape
- Magnetic Disks
- Floppy Disks
- Hard Disks and Drives

These high-speed storage devices are very expensive and hence the cost per bit of storage is also very high. Again, the storage capacity of the main memory is also very limited. Often it is necessary to store hundreds of millions of bytes of data for the CPU to process. Therefore, additional memory is required in all the computer systems. This memory is called auxiliary memory or secondary storage. In this type of memory the cost per bit of storage is low. However, the operating speed is slower than that of the primary memory. Most widely used secondary storage devices are magnetic tapes, magnetic disks and floppy disks.

- It is not directly accessible by the CPU.
- Computer usually uses its input / output channels to access secondary storage and transfers the desired data using intermediate are a in primary storage.

3.2.1 Magnetic Tapes

Magnetic tape is a medium for magnetic recording, made of a thin, magnetisable coating on a long, narrow strip of plastic film. It was developed in Germany, based on magnetic wire recording. Devices that record and play back audio and video using magnetic tape are tape recorders and video tape recorders. Magnetic tape an information storage medium consisting of a magnetic coating on a flexible backing in tape form. Data is recorded by magnetic encoding of tracks on the coating according to a particular tape format.

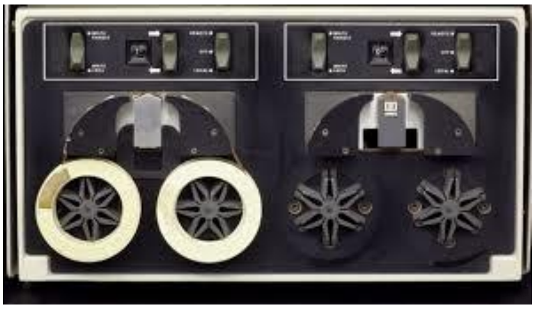


Figure 1.0: Magnetic Tape

Characteristics of Magnetic Tapes

- No direct access, but very fast sequential access.
- Resistant to different environmental conditions.
- Easy to transport, store, cheaper than disk.
- Before, it was widely used to store application data; nowadays,
- it's mostly used for backups or archives (tertiary storage).

Magnetic tape is wound on reels (or spools). These may be used on their own, as open-reel tape, or they may be contained in some sort of magnetic tape cartridge for protection and ease of handling. Early computers used open-reel tape, and this is still sometimes used on large computer systems although it has been widely superseded by cartridge tape. On smaller systems, if tape is used at all it is normally cartridge tape.

Figure 1.2: Magnetic Tape

Magnetic tape is used in a tape transport (also called a tape drive, tape deck, tape unit, or MTU), a device that moves the tape over one or more magnetic heads. An electrical signal is applied to the write head to record data as a magnetic pattern on the tape; as the recorded tape passes over the read head it generates an electrical signal from which the stored data can be reconstructed. The two heads may be combined into a single read/write head. There may also be a separate erase head to erase the magnetic pattern remaining from previous use of the tape. Most magnetic-tape formats have several separate data tracks running the length of the tape. These may be recorded simultaneously, in which case, for example, a byte of data may be recorded with one bit in each track (parallel recording); alternatively, tracks may be recorded one at a time (serial recording) with the byte written serially along one track.

- Magnetic tape has been used for offline data storage, backup, archiving, data interchange, and software distribution, and in the early days (before disk storage was available) also as online backing store. For many of these purposes it has been superseded by magnetic or optical disk or by online communications. For example, although tape is a non-volatile medium, it tends to deteriorate in long-term storage and so needs regular attention (typically an annual rewinding and inspection) as well as a controlled environment. It is therefore being superseded for archival purposes by optical disk.
- Magnetic tape is still extensively used for backup; for this purpose, interchange standards are of minor importance, so proprietary cartridge-tape formats are widely used.
- Magnetic tapes are used for large computers like mainframe computers where large volume of data is stored for a longer time. In PCs also you can use tapes in the form of cassettes.
- The cost of storing data in tapes is inexpensive. Tapes consist of magnetic materials that store data permanently. It can be 12.5 mm to

25 mm wide plastic film-type and 500 meter to 1200 meter long which is coated with magnetic material. The deck is connected to the central processor and information is fed into or read from the tape through the processor. It is similar to cassette tape recorder.

Advantages of Magnetic Tape

- Compact: A 10-inch diameter reel of tape is 2400 feet long and is able to hold 800, 1600 or 6250 characters in each inch of its length. The maximum capacity of such type is 180 million characters. Thus data are stored much more compact on tape
- **Economical:** The cost of storing characters on tape is very less as compared to other storage devices.
- **Fast:** Copying of data is easier and fast.
- **Long term Storage and Re-usability:** Magnetic tapes can be used for long term storage and a tape can be used repeatedly without loss of data.

3.2.2 Magnetic Disks

You might have seen the gramophone record, which is circular like a disk and coated with magnetic material. Magnetic disks used in computer are made on the same principle. It rotates with very high speed inside the disk drive. Data are stored on both the surface of the disk.

Magnetic disks are most popular for direct access storage. Each disk consists of a number of invisible concentric circles called tracks. Information is recorded on tracks of a disk surface in the form of tiny magnetic sports. The presence of a magnetic sport represents one bit (1) and its absence represents zero bit (0). The information stored in a disk can be read many times without affecting the stored data. So the reading operation is non-destructive. But if you want to write a new data, then the existing data is erased from the disk and new data is recorded.

A digital computer memory in which the data carrier is a thin aluminium or plastic dis coated with a layer of magnetic material. Magnetic disks are 180-1,200 mm in diameter and 2.5-5.0 mm thick; Ni-Co or CoW alloys are used for the magnetic coating. A magnetic disk memory usually contains several dozen disks mounted on a comm on axle, which is turned by an electric motor. One or more disks (a packet) may be replaced, creating disk index files. There may be as many as 100 disks in a memory and 64- 5,000 data tracks on each operating surface of a disk; the recording density is 20 130 impulses per millimeter.

The data capacity of magnetic disk memories range from several tens of thousands up to several billion bits, and the average access time is 10- 100 milliseconds. The two main types are the hard disk and the floppy disk.

Data is stored on either or both surfaces of discs in concentric rings called "tracks". Each track is divided into a whole number of "sectors". Where multiple (rigid) discs are mounted on the same axle the set of tracks at the same radius on all their surfaces is known as a" cylinder". Data is read and written by a disk drive which rotates the discs and positions the read/write heads over the desired track(s). The latter radial movement is known as "seeking”. There is usually one head for each surface that stores data. The head writes binary data by magnetising small areas or "zones" of the disk in one of two opposing orientations. It reads data by detecting current pulses induced in a coil as zones with different magnetic alignment pass underneath it.

In theory, bits could be read back as a time sequence of pulse (one) or no pulse (zero). However, a run of zeros would give a prolonged absence of signal, making it hard to accurately divide the signal into individual bits due to the variability of motor speed. High speed disks have an access time of 28 milliseconds or less, and low speed disk s, 65milliseconds or more. The higher

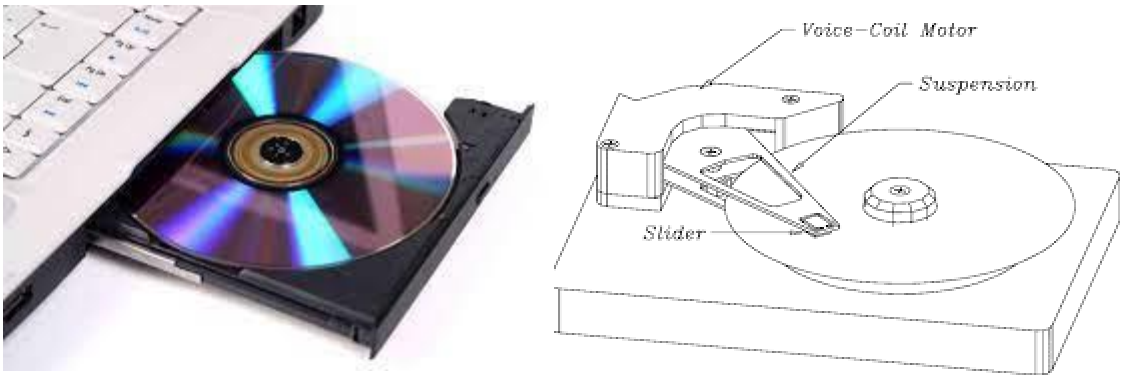


Figure 1.3: Magnetic Disks

speed disks also transfer their data faster than the slower speed units. The disks are usually aluminum with a magnetic coating. The heads "float" just above the disk's surface on a current of air, sometimes at lower than atmospheric pressure in an air tight enclosure. The head has an aerodynamic shape so the current pushes it away from the disk. A small spring pushes the head towards the disk at the same time keeping the he a data constant distance from the disk (about two microns). Disk drives are commonly characterized by the kind of interface used to connect to the computer

3.2.3 Floppy Disks

These are small removable disks that are plastic coated with magnetic recording material. Floppy disks are typically 3.5″ in size (diameter) and can hold 1.44 MB of data. This portable storage device is a rewritable media and can be reused a number of times. Floppy disks are commonly used to move files between different computers. The main disadvantage of floppy disks is that they can be damaged easily and, therefore, are not very reliable. The following figure shows an example of the floppy disk. It is similar to magnetic

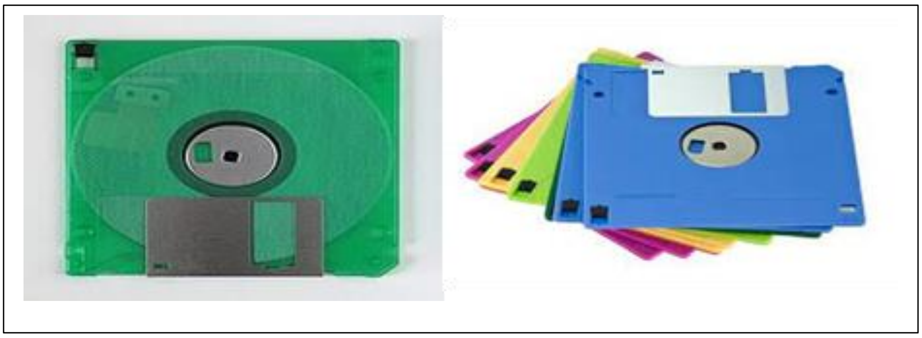


Figure 1.4: Floppy Disks

disk. It is 3.5 inch in diameter. The capacity of a 3.5-inch floppy is 1.44 megabytes. It is cheaper than any other storage devices and is portable. The floppy is a low-cost device particularly suitable for personal computer system.

- **Read/Write head:** A floppy disk drive normally has two-read/write heads making Modern floppy disk drives as double-sided drives. A head exists for each side of disk and Both heads are used for reading and writing on the respective disk side.
- **Head 0 and Head 1:** Many people do not realize that the first head (head 0) is bottom one and top head is head 1. The top head is located either four or eight tracks inward from the bottom head depending upon the drive type.
- **Head Movement:** A motor called head actuator moves the head mechanism. The heads can move in and out over the surface of the disk in a straight line to position themselves over various tracks. The heads move in and out tangentially to the tracks that they record on the disk.
- **Head:** The heads are made of soft ferrous (iron) compound with electromagnetic coils. Each head is a composite design with a R/W head centered within two tunnel erasure heads in the same physical assembly. PC compatible floppy disk drive spin at 300 or 360r.p.m. The two heads are spring loaded and physically grip the disk with small pressure, this pressure does not present excessive friction.

3.2.3.1 Recording Method

- **Tunnel Erasure:** As the track is laid down by the R/W heads, the trailing tunnel erasure heads force the data to be present only within a specified narrow tunnel on each track. This process prevents the signals from reaching adjacent track and making cross talk.
- **Straddle Erasure:** In this method, the R/W and the erasure heads do recording and erasing at the same time. The erasure head is not used to erase data stored in the diskette. It trims the top and bottom fringes of recorded flux reversals. The erasure heads reduce the effect of cross-talk between tracks and minimize the errors induced by minor run out problems on the diskette or diskette drive.
- **Head alignment:** Alignment is the process of placement of the heads with respect to the track that they must read and write. Head alignment can be checked only against some sort of reference- standard disk recorded by perfectly aligned machine. These types of disks are available and one can use one to check the drive alignment.

3.2.4 Hard Disks and Drives

A hard disk drive (HDD), hard disk, hard drive or fixed disk is a data storage device that uses magnetic storage to store and retrieve digital information using one or more rigid rapidly rotating disks (platters) coated with magnetic material. The platters are paired with magnetic heads, usually arranged on a moving actuator arm, which read and write data to the platter surfaces. Data is accessed in a random-access manner, meaning that individual blocks of data can be stored or retrieved in any order and not only sequentially.

HDDs are a type of non-volatile storage, retaining stored data even when powered off. A hard drive can be used to store any data, including pictures, music, videos, text documents, and any files created or downloaded. Also, hard drives store files for the operating and software programs that run on the computer. All primary computer hard drives are found inside a computer case and are attached to the computer motherboard using an ATA, SCSI, or SATA cable, and are powered by a connection to the PSU (power supply unit). The hard drive is typically capable of storing more data than any other drive, but its size can vary depending on the type of drive and its age. Older hard drives had a storage size of several hundred megabytes (MB) to several gigabytes (GB). Newer hard drives have a storage size of several hundred gigabytes to several terabytes (TB). Each year, new and improved technology allows for increasing hard drive storage sizes.

3.2.4.1 Hard Drive Components

As can be seen in the picture below, the desktop hard drive consists of the following components: the head actuator, read/write actuator arm, read/write head, spindle, and platter. On the back of a hard drive is a circuit board called the disk controller or interface board and is what allows the hard drive to communicate with the computer.

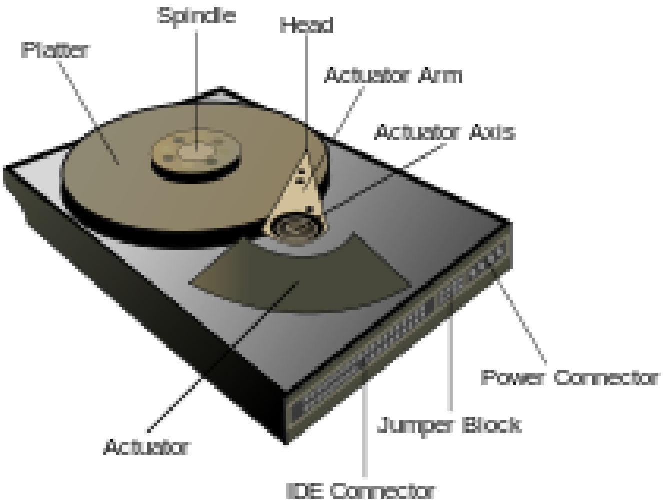


Figure 1.5: Hard Drive Components

3.2.4.2 External and Internal hard drives

Although most hard drives are internal, there are also stand-alone devices called external hard drives, which can backup data on computers and expand the available disk space. External drives are often stored in an enclosure that helps protect the drive and allows it to interface with the computer, usually over USB or eSATA.

Figure 1.6: Hard Drive

3.2.4.3 History of the hard drive

The first hard drive was introduced to the market by IBM on September 13, 1956. The hard drive was first used in the RAMAC 305 system, with a storage capacity of 5 MB and a cost of about $50,000 ($10,000 per megabyte). The hard drive was built-in to the computer and was not removable. The first hard drive to have a storage capacity of one gigabyte was also developed by IBM in 1980. It weighed 550 pounds and cost $40,000. 1983 marked the introduction of the first 3.5-inch size hard drive, developed by Rodime. It had a storage capacity of 10 MB. Seagate was the first company to introduce a 7200 RPM hard drive in 1992. Seagate also introduced the first 10,000 RPM hard drive in 1996 and the first 15,000 RPM hard drive in 2000. The first solid-state drive (SSD) as we know them today was developed by SanDisk Corporation in 1991, with a storage capacity of 20 MB. However, this was not a flash-based SSD, which were introduced later in 1995 by M-Systems. These drives did not require a battery to keep data stored on the memory chips, making them a non-volatile storage medium.

3.2.5 CD-ROM Compact Disk/Read Only Memory (CD-ROM)

CD-ROM disks are made of reflective metals. CD-ROM is written during the process of manufacturing by high power laser beam. Here the storage density is very high, storage cost is very low and access time is relatively fast. Each disk is approximately 4 1/2 inches in diameter and can hold over 600 MB of data. As the CD-ROM can be read only we cannot write or make changes into the data contained in it.

Figure 1.7: CD-Rom

3.2.5.1 Characteristics of The CD-ROM

- In PCs, the most commonly used optical storage technology is called Compact Disk Read-Only Memory (CD-ROM).
- A standard CD-ROM disk can store up to 650 MB of data, or about 70 minutes of audio.
- Once data is written to a standard CD-ROM disk, the data cannot be altered or overwritten. CD-ROM SPEEDS AND USES Storage capacity 1 CD can store about 600 to 700 MB = 600 000 to 700 000 KB. For comparison, we should realize that a common A4 sheet of paper can store an amount of information in the form of printed characters that would require about 2 kB of space on a computer. So one CD can store about the same amount of text information equivalent as 300 000 of such A4 sheets. Yellow Book standard

- The basic technology of CD-ROM remains the same as that for CD audio, but CD-ROM requires greater data integrity, because a corrupt bit that is not noticeable during audio playback becomes intolerable with computer data.
- So CD-ROM (Yellow Book) dedicates more bits to error detection and correction than CD audio (Red Book).
- Data is laid out in a format known as ISO 960. Advantages in comparison with other information carriers
- The information density is high.
- The cost of information storage per information unit is low.
- The disks are easy to store, to transport and to mail.
- Random access to information is possible.

Advantages

- Easier access to a range of CD-ROMs.
- Ideally, access from the user's own workstation in the office or at home.
- Simultaneous access by several users to the same data.
- Better security avoids damage to discs and equipment.
- Less personnel time needed to provide disks to users.
- Automated, detailed registration of usage statistics to support the management

Disadvantages

- Costs of the network software and computer hardware.
- Increased charges imposed by the information suppliers.
- Need for expensive, technical expertise to select, set up, manage, and maintain the network system.
- Technical problems when the CD-ROM product is not designed for use in the network.

- The network software component for the workstation side must be installed on each microcomputer before this can be applied to access the CD-ROM’s.

3.2.6 Other Optical Devices

An optical disk is made up of a rotating disk which is coated with a thin reflective metal. To record data on the optical disk, a laser beam is focused on the surface of the spinning disk. The laser beam is turned on and off at varying rates! Due to this, tiny holes (pits) are burnt into the metal coating along the tracks. When data stored on the optical disk is to be read, a less powerful laser beam is focused on the disk surface. The storage capacity of these devices is tremendous; the Optical disk access time is relatively fast. The biggest drawback of the optical disk is that it is a permanent storage device. Data once written cannot be erased. Therefore it is a read only storage medium. A typical example of the optical disk is the CD-ROM.

1. **Read-only memory (ROM)** disks, like the audio CD, are used for the distribution of standard program and data files. These are mass-produced by mechanical pressing from a master die. The information is actually stored as physical indentations on the surface of the CD. Recently low-cost equipment has been introduced in the market to make one-off CD-ROMs, putting them into the next category.
2. **Write-once read-many (WORM) disks:** Some optical disks can be recorded once. The information stored on the disk cannot be changed or erased. Generally the disk has a thin reflective film deposited on the surface. A strong laser beam is focused on selected spots on the surface and pulsed. The energy melts the film at that point, producing a nonreflective void. In the read mode, a low power laser is directed at the disk and the bit information is recovered by sensing the presence or absence of a reflected beam from the disk.

3. **Re-writeable, write-many read-many (WMRM)** disks, just like the magnetic storage disks, allows information to be recorded and erased many times. Usually, there is a separate erase cycle although this may be transparent to the user. Some modern devices have this accomplished with one over-write cycle. These devices are also called direct read-after-write (DRAW) disks.
4. **WORM (write once, read many)** is a data storage technology that allows information to be written to a disc a single time and prevents the drive from erasing the data. The discs are intentionally not rewritable, because they are especially intended to store data that the user does not want to erase accidentally. Because of this feature, WORM devices have long been used for the archival purposes of organizations such as government agencies or large enterprises. A type of optical media, WORM devices were developed in the late 1970s and have been adapted to a number of different media. The discs have varied in size from 5.25 to 14 inches wide, in varying formats ranging from 140MB to more than 3 GB per side of the (usually) double-sided medium. Data is written to a WORM disc with a low- powered laser that makes permanent marks on the surface. WORM (Write Once, Read Many) storage had emerged in the late 1980s and was popular with large institutions for the archiving of high volume, sensitive data. When data is written to a WORM drive, physical marks are made on the media surface by a low- powered laser and since these marks are permanent, they cannot be erased. Rewritable, or erasable, optical disk drives followed, providing the same high capacities as those provided by WORM or CD-ROM devices.
5. **Erasable Optical Disk:** An erasable optical disk is the one which can be erased and then loaded with new data content all over again. These generally come with a RW label. These are based on a technology popularly known as Magnetic Optical which involves the application

of heat on a precise point on the disk surface and magnetizing it using a laser. Magnetizing alters the polarity of the point indicating data value '1'. Erasing too is achieved by heating it with a high energy laser to a certain critical level where the crystal polarity is reset to all 0's. A variety of optical disc, or type of external storage media, that allows the deletion and rewriting of information, unlike a CD or CD-ROM, which are read-only optical discs. An erasable optical disc allows high-capacity storage (600 MB or more) and their durability has made them useful for archival storage.

6. **Touchscreen Optical Device:** A touchscreen is an input and output device normally layered on the top of an electronic visual display of an information processing system. A user can give input or control the information processing system through simple or multi-touch gestures by touching the screen with a special stylus or one or more fingers. Some touchscreens use ordinary or specially coated gloves to work while others may only work using a special stylus or pen. The user can use the touchscreen to react to what is displayed and, if the software allows, to control how it is displayed; for example, zooming to increase the text size. The touchscreen enables the user to interact directly with what is displayed, rather than using a mouse, touchpad, or other such devices (other than a stylus, which is optional for most modern touchscreens). Touchscreens are common in devices such as game consoles, personal computers, electronic voting machines, and point-of-sale (POS) systems. They can also be attached to computers or, as terminals, to networks. They play a prominent role in the design of digital appliances such as personal digital assistants (PDAs) and some e-readers.

There are two types of overlay-based touch screens:

- **Capacitive Touch Technology** – Capacitive touch screens take advantage of the conductivity of the object to detect location of touch. While they are durable and last for a long time, they can malfunction if they get wet. Their performance is also compromised if a nonconductor like a gloved finger presses on the screen. Most smart phones and tablets have capacitive touch screens.
- **Resistive Touch Technology** – Resistive touch screens have moving parts. There is an air gap between two layers of transparent material. When the user applies pressure to the outer layer, it touches the inner layer at specific locations. An electric circuit is completed and the location can be determined. Though they are cheaper to build compared to capacitive touch screens, they are also less sensitive and can wear out quickly.

There are mainly three types of perimeter-based technologies:

- **Infrared Touch Technology** – This technology uses beams of infrared lights to detect touch events.
- **Surface Acoustic Wave Touch Technology** – This type of touch screen uses ultrasonic waves to detect touch events.
- **Optical Touch Technology** – This type of perimeter-based technology uses optical sensors, mainly CMOS sensors to detect touch events. All of these touch screen technologies can also be integrated on top of a non-touch-based system like an ordinary LCD and converted into Open Frame Touch Monitors.

3.3 MEMORY ACCESS METHODS

Data need to be accessed from the memory for various purposes. There are several methods to access memory as listed below:

- Sequential access
- Direct access
- Random access
- Associative access

We will study about each of the access method one by one.

3.3.1 Sequential Access Method

In sequential memory access method, the memory is accessed in linear sequential way. The time to access data in this type of method depends on the location of the data.

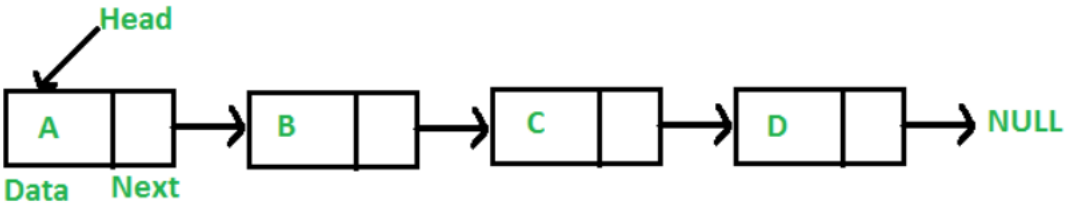


Figure 1.8: Sequential Access Method

3.3.2 Random Access Method

In random access method, data from any location of the memory can be accessed randomly.

The access to any location is not related with its physical location and is independent of other locations. There is a separate access mechanism for each location.

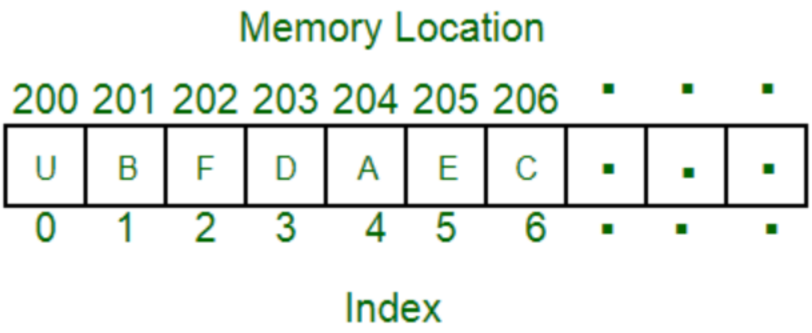


Figure 1.9: Random Access Method

3.3.3 Direct Access Method

Direct access method can be seen as combination of sequential access method and random access method. Magnetic hard disks contain many rotating storage tracks. Here each tracks has its own read or write head and the tracks can be accessed randomly. But access within each track is sequential.

Example of direct access: Memory devices such as magnetic hard disks.

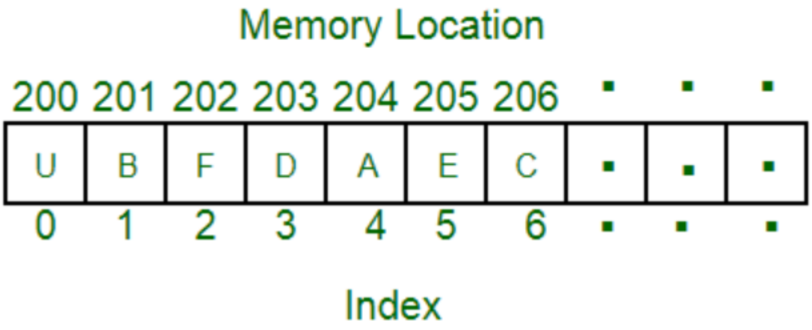


Figure 1.10: Direct Access Method

1.3.4 Associative Access Method

Associative access method is a special type of random access method. It enables comparison of desired bit locations within a word for a specific match and to do this for all words simultaneously. Thus, based on portion of word's content, word is retrieved rather than its address. Example of associative access: Cache memory uses associative access method.

3.4 MEMORY MAPPING AND VIRTUAL MEMORIES

Memory-mapping is a mechanism that maps a portion of a file, or an entire file, on disk to a range of addresses within an application's address space. The application can then access files on disk in the same way it accesses dynamic memory. This makes file reads and writes faster in comparison with using functions such as fread and fwrite.

3.4.1 Benefits of Memory-Mapping

The principal benefits of memory-mapping are efficiency, faster file access, the ability to share memory between applications, and more efficient coding.

3.4.1.1 Faster File Access

Accessing files via memory map is faster than using I/O functions such as fread and fwrite. Data are read and written using the virtual memory capabilities that are built in to the operating system rather than having to allocate, copy into, and then deallocate data buffers owned by the process does not access data from the disk when the map is first constructed. It only reads or writes the file on disk when a specified part of the memory map is accessed, and then it only reads that specific part. This provides faster random access to the mapped data.

3.4.1.2 Efficiency

Mapping a file into memory allows access to data in the file as if that data had been read into an array in the application's address space. Initially, MATLAB only allocates address space for the array; it does not actually read data from the file until you access the mapped region. As a result, memory-mapped files provide a mechanism by which applications can access data segments in an extremely large file without having toread the entire file into memory first. Efficient Coding Style Memory-mapping in your MATLAB application enables you to access file data using standard MATLAB indexing operations.

3.4.2 VIRTUAL MEMORIES

Processes in a system share the CPU and main memory with other processes. However, sharing the main memory poses some special challenges. As demand on the CPU increases, processes slowdown in some reasonably smooth way. But if too many processes need too much memory, then some of them will simply not be able to run. When a program is out of space, it is out of luck. Memory is also vulnerable to corruption. If some process inadvertently writes to the memory used by another process, that process might fail in some bewildering fashion totally unrelated to the program logic. In order to manage memory more efficiently and with fewer errors, modern systems provide an abstraction of main memory known as virtual memory (VM). Virtual memory is an elegant interaction of hardware exceptions, hardware address translation, main memory, disk files, and kernel software that provides each process with a large, uniform, and private address space. With one clean mechanism, virtual memory provides three important capabilities.

(1) It uses main memory efficiently by treating it as a cache for an address space stored on disk, keeping only the active areas in main memory, and transferring data back and forth between disk and memory as needed.

(2) It simplifies memory management by providing each process with a uniform address space.

(3) It protects the address space of each process from corruption by other processes.

Virtual memory is one of the great ideas in computer systems. A major reason for its success is that it works silently and automatically, without any intervention from the application programmer. Since virtual memory works so well behind the scenes, why would a programmer need to understand it? There are several reasons.

- Virtual memory is central. Virtual memory pervades all levels of computer systems, playing key roles in the design of hardware exceptions, assemblers, linkers, loaders, shared objects, files, and processes. Understanding virtual memory will help you better understand how systems work in general.
- Virtual memory is powerful. Virtual memory gives applications powerful capabilities to create and destroy chunks of memory, map chunks of memory to portions of disk files, and share memory with other processes. For example, did you know that you can read or modify the contents of a disk file by reading and writing memory locations? Or that you can load the contents of a file into memory without doing any explicit copying? Understanding virtual memory will help you harness its powerful capabilities in your applications.

3.4.2.1 VM as a Tool for Caching

Conceptually, a virtual memory is organized as an array of N contiguous byte-sized cells stored on disk. Each byte has a unique virtual address that serves as an index into the array. The contents of the array on disk are cached in main memory. As with any other cache in the memory hierarchy, the data on disk (the lower level) is partitioned into blocks that serve as the transfer units between the disk and the main memory (the upper level). VM systems handle this by partitioning the virtual memory into fixed-sized blocks called virtual pages (VPs). Each virtual page is P = 2p bytes in size. Similarly, physical memory is partitioned into physical pages (PPs), also P bytes in size. (Physical pages are also referred to as page frames.)

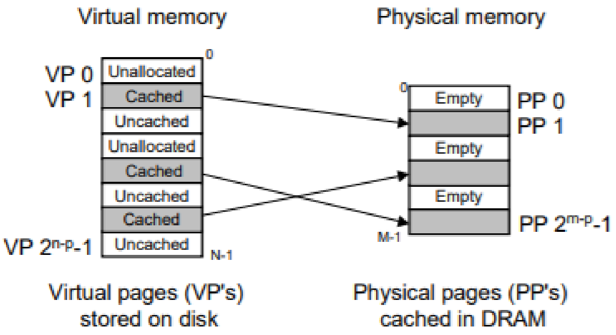


Figure 1.12: Memory as a Cache

3.4.2.2 Page Tables

As with any cache, the VM system must have some way to determine if a virtual page is cached somewhere in DRAM. If so, the system must determine which physical page it is cached in. If there is a miss, the system must determine where the virtual page is stored on disk, select a victim page in physical memory, and copy the virtual page from disk to DRAM, replacing the victim page. These capabilities are provided by a combination of operating system software, address translation hardware in the MMU (memory management unit), and a data structure stored in physical memory known as a page table that maps virtual pages to physical pages. The address translation hardware reads the page table each time it converts a virtual address to a physical address. The operating system is responsible for maintaining the

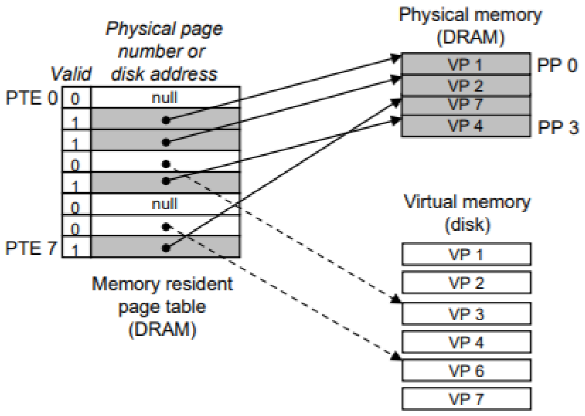


Figure 1.13: Page Table

contents of the page table and transferring pages back and forth between disk and DRAM

Virtual memory was invented in the early 1960s, long before the widening CPU-memory gap spawned SRAM caches. As a result, virtual memory systems use a different terminology from SRAM caches, even though many of the ideas are similar. In virtual memory parlance, blocks are known as pages. The activity of transferring a page between disk and memory is known as swapping or paging. Pages are swapped in (paged in) from disk to DRAM, and swapped out (paged out) from DRAM to disk. The strategy of waiting until the last moment to swap in a page, when a miss occurs, is known as demand paging. Other approaches, such as trying to predict misses and swap pages in before they are actually referenced, are possible. However, all modern systems use demand paging.

3.4.2.3 VM as a Tool for Memory Protection

Any modern computer system must provide the means for the operating system to control access to the memory system. A user process should not be allowed to modify its read-only text section. Nor should it be allowed to read or modify any of the code and data structures in the kernel. It should not be allowed to read or write the private memory of other processes, and it should not be allowed to modify any virtual pages that are shared with other processes, unless all parties explicitly allow it (via calls to explicit inter-process communication system calls).

3.4.2.4 Integrating Caches and VM

In any system that uses both virtual memory and SRAM caches, there is the issue of whether to use virtual or physical addresses to access the SRAM cache. Although a detailed discussion of the trade-offs is beyond our scope here, most systems opt for physical addressing. With physical addressing, it is straightforward for multiple processes to have blocks in the cache at the same time and to share blocks from the same virtual pages. Further, the cache does not have to deal with protection issues because access rights are checked as part of the address translation process.

3.4.2.5 Speeding up Address Translation with a TLB

As we have seen, every time the CPU generates a virtual address, the MMU must refer to a PTE in order to translate the virtual address into a physical address. In the worst case, this requires an additional fetch from memory, at a cost of tens to hundreds of cycles. If the PTE happens to be cached in L1, then the cost goes down to one or two cycles. However, many systems try to eliminate even this cost by including a small cache of PTEs in the MMU called a translation lookaside buffer (TLB). A TLB is a small, virtually addressed cache where each line holds a block consisting of a single PTE. A TLB usually has a high degree of associativity

3.5 REPLACEMENT ALGORITHMS

When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. If the page to be removed has been modified while in memory, it must be rewritten to the disk to bring the disk copy up to date. If, however, the page has not been changed (e.g., it contains program text), the disk copy is already up to date, so no rewrite is needed. The page to be read in just overwrites the page being evicted. While it would be possible to pick a random page to evict at each page fault, system performance is much better if a page that is not heavily used is chosen. If a heavily used page is removed, it will probably have to be brought back in quickly, resulting in extra overhead. Much work has been done on the subject of page replacement algorithms, both theoretical and experimental. Below we will describe some of the most important algorithms. It is worth noting that the problem of ‘‘page replacement’’ occurs in other areas of computer design as well. For example, most computers have one or more memory caches consisting of recently used 32-byte or 64-byte memory blocks. When the cache is full, some block has to be chosen for removal. This problem is precisely the same as page replacement except on a shorter time scale (it has to be done in a few nanoseconds, not milliseconds as with page replacement). The reason for the shorter time scale is that cache block misses are satisfied from main memory, which has no seek time and no rotational latency. To select the particular algorithm, the algorithm with lowest page fault rate is considered.

- Optimal page replacement algorithm
- Not recently used page replacement
- First-In, First-Out page replacement
- Second chance page replacement
- Clock page replacement
- Least recently used page replacement

3.5.1 The Optimal Page Replacement Algorithm

The best possible page replacement algorithm is easy to describe but impossible to implement. It goes like this. At the moment that a page fault occurs, some set of pages is in memory. One of these pages will be referenced on the very next instruction (the page containing that instruction). Other pages may not be referenced until 10, 100, or perhaps 1000 instructions later. Each page can be labeled with the number of instructions that will be executed before that page is first referenced. The optimal page algorithm simply says that the page with the highest label should be removed. If one page will not be used for 8 million instructions and another page will not be used for 6 million instructions, removing the former pushes the page fault that will fetch it back as far into the future as possible.

3.5.2 The Not Recently Used Page Replacement Algorithm

In order to allow the operating system to collect useful statistics about which pages are being used and which ones are not, most computers with virtual memory have two status bits associated with each page. R is set whenever the page is referenced (read or written). M is set when the page is written to (i.e., modified). The bits are contained in each page table entry. It is important to realize that these bits must be updated on every memory reference, so it is essential that they be set by the hardware. Once a bit has been set to 1, it stays 1 until the operating system resets it to 0 in software. If the hardware does not have these bits, they can be simulated as follows. When a process is started up, all of its page table entries are marked as not in memory. As soon as any page is referenced, a page fault will occur. The operating system then sets the R bit (in its internal tables), changes the page table entry to point to the correct page, with mode READ ONLY, and restarts the instruction. If the page is subsequently written on, another page fault will occur, allowing the operating system to set the M bit and change the page's mode to READ/WRITE. The R and M bits can be used to build a simple paging algorithm as follows. When a process is started up, both page bits for all its pages are set to 0 by the operating system. Periodically (e.g., on each clock interrupt), the R bit is cleared, to distinguish pages that have not been referenced recently from those that have been. When a page fault occurs, the operating system inspects all the pages and divides them into four categories based on the current values of their R and M bits:

- Class 0: not referenced, not modified.
- Class 1: not referenced, modified.
- Class 2: referenced, not modified.
- Class 3: referenced, modified.

3.5.3 The First-In, First-Out (FIFO) Page Replacement Algorithm

Another low-overhead paging algorithm is the First-In, First-Out (FIFO) algorithm. To illustrate how this works, consider a supermarket that has enough shelves to display exactly k different products. One day, some company introduces a new convenience food—instant, freeze-dried, organic yogurt that can be reconstituted in a microwave oven. It is an immediate success, so our finite supermarket has to get rid of one old product in order to stock it. One possibility is to find the product that the supermarket has been stocking the longest (i.e., something it began selling 120 years ago) and get rid of it on the grounds that no one is interested any more. In effect, the supermarket maintains a linked list of all the products it currently sells in the order they were introduced. The new one goes on the back of the list; the one at the front of the list is dropped. As a page replacement algorithm, the same idea is applicable. The operating system maintains a list of all pages currently in memory, with the page at the head of the list the oldest one and the page at the tail the most recent arrival. On a page fault, the page at the head is removed and the new page added to the tail of the list. When applied to stores, FIFO might remove mustache wax, but it might also remove flour, salt, or butter. When applied to computers the same problem arises. For this reason, FIFO in its pure form is rarely used.

3.5.4 The Second Chance Page Replacement Algorithm

A simple modification to FIFO that avoids the problem of throwing out a heavily used page is to inspect the R bit of the oldest page. If it is 0, the page is both old and unused, so it is replaced immediately. If the R bit is 1, the bit is cleared, the page is put onto the end of the list of pages, and its load time is updated as though it had just arrived in memory. Then the search continues.

3.5.5 The Clock Page Replacement Algorithm

Although second chance is a reasonable algorithm, it is unnecessarily inefficient because it is constantly moving pages around on its list. A better approach is to keep all the page frames on a circular list in the form of a clock.

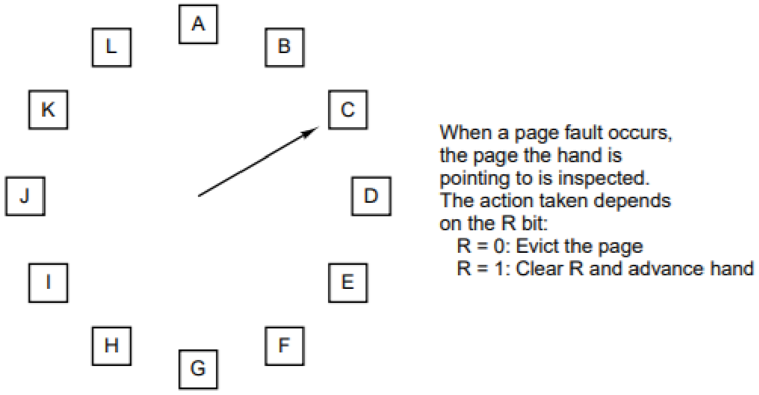

A
B
C
D
E
F
G
H
I
J
K
L
When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:
R = 0: Evict the page
R = 1: Clear R and advance hand

Figure 1.14: The Clock Replacement Algorithm

When a page fault occurs, the page being pointed to by the hand is inspected.

If its R bit is 0, the page is evicted, the new page is inserted into the clock in its place, and the hand is advanced one position. If R is 1, it is cleared and the hand is advanced to the next page. This process is repeated until a page is found with R = 0. Not surprisingly, this algorithm is called clock. It differs from second chance only in the implementation.

3.5.6 The Least Recently Used (LRU) Page Replacement Algorithm

A good approximation to the optimal algorithm is based on the observation that pages that have been heavily used in the last few instructions will probably be heavily used again in the next few. Conversely, pages that have not been used for ages will probably remain unused for a long time. This idea suggests a realizable algorithm: when a page fault occurs, throw out the page that has been unused for the longest time. This strategy is called LRU (Least Recently Used) paging. Although LRU is theoretically realizable, it is not cheap. To fully implement LRU, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear. The difficulty is that the list must be updated on every memory reference. Finding a page in the list, deleting it, and then moving it to the front is a very time-consuming operation, even in hardware (assuming that such hardware could be built).

3.6 DATA TRANSFER MODES

The **DMA mode** of data transfer reduces CPU’s overhead in handling I/O operations. It also allows parallelism in CPU and I/O operations. Such parallelism is necessary to avoid wastage of valuable CPU time while handling I/O devices whose speeds are much slower as compared to CPU. The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rises to the development of special purpose processor called **Input-Output Processor (IOP) or IO channel**. The Input Output Processor (IOP) is just like a CPU that handles the details of I/O operations. It is more equipped with facilities than those are available in typical DMA controller.

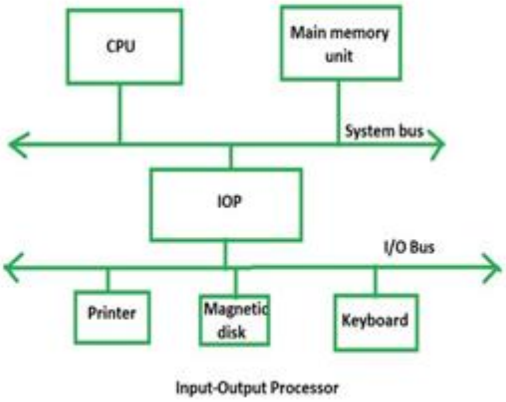


Figure 1.15: The Block Diagram

The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O – related tasks, it can perform other processing tasks like arithmetic, logic, branching and code translation. The main memory unit takes the pivotal role. It communicates with processor by the means of DMA.

The Input Output Processor is a specialized processor which loads and stores data into memory along with the execution of I/O instructions. It acts as an interface between system and devices. It involves a sequence of events to executing I/O operations and then store the results into the memory.

3.6.1 Advantages

- The I/O devices can directly access the main memory without the intervention by the processor in I/O processor-based systems.
- It is used to address the problems that are arises in Direct memory access method.

3.6.2 Modes of Transfer

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways.

- Programmed I/O.
- Interrupt- initiated I/O.
- Direct memory access (DMA).

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually, the transfer is from a CPU register and

memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

Note:*Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus, both these forms of I/O suffer from two inherent drawbacks.*

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.

- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

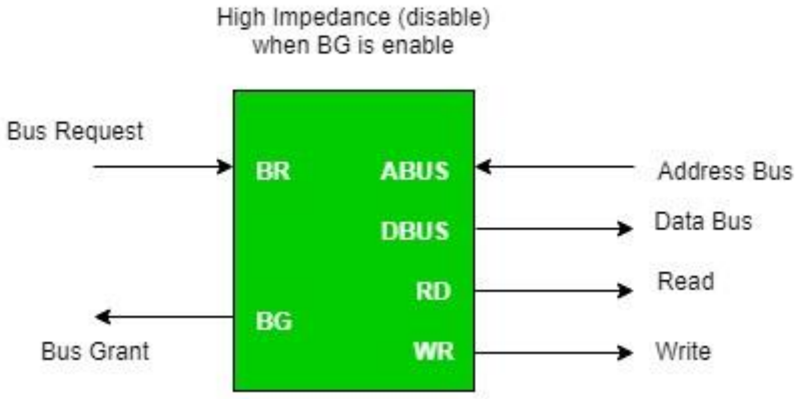


Figure 1.16: Control lines for DMA

- **Bus Request:** It is used by the DMA controller to request the CPU to relinquish the control of the buses.
- **Bus Grant:** It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

3.7 PARALLEL PROCESSING

The quest for higher-performance digital computers seems unending. In the past two decades, the performance of microprocessors has enjoyed an exponential growth. The growth of microprocessor speed/performance by a factor of 2 every 18 months (or about 60% per year) is known as Moore's law.

This growth is the result of a combination of two factors:

- Increase in complexity (related both to higher device density and to larger size) of VLSI chips, projected to rise to around 10 M transistors per chip for microprocessors, and 1B for dynamic random-access memories (DRAMs), by the year 2000
- Introduction of, and improvements in, architectural features such as on-chip cache memories, large instruction buffers, multiple instruction issue per cycle, multithreading, deep pipelines, out-of-order instruction execution, and branch prediction.

The motivations for parallel processing can be summarized as follows:

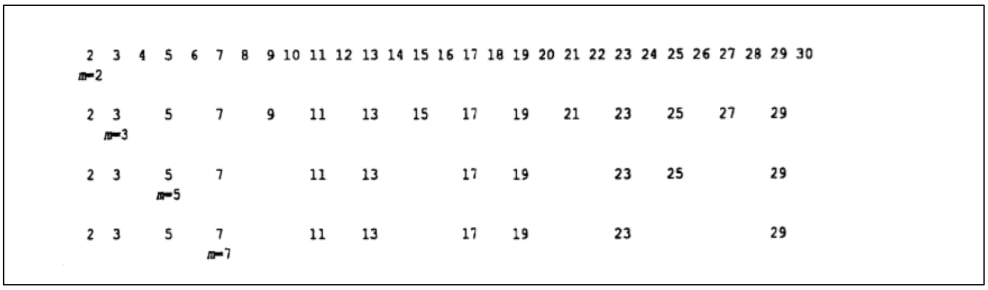
1. Higher speed, or solving problems faster. This is important when applications have “hard” or “soft” deadlines. For example, we have at most a few hours of computation time to do 24-hour weather forecasting or to produce timely tornado warnings.
2. Higher throughput, or solving more instances of given problems. This is important when many similar tasks must be performed. For example, banks and airlines, among others, use transaction processing systems that handle large volumes of data.
3. Higher computational power, or solving larger problems. This would allow us to use very detailed, and thus more accurate, models or to carry out simulation runs for longer periods of time (e.g., 5-day, as opposed to 24-hour, weather forecasting).

All three aspects above are captured by a figure-of-merit often used in connection with parallel processors: the computation speed-up factor with respect to a uniprocessor. The ultimate efficiency in parallel systems is to achieve a computation speed-up factor of p with p processors. Although in many cases this ideal cannot be achieved, some speed-up is generally possible. The actual gain in speed depends on the architecture used for the system and the algorithm run on it. Of course, for a task that is (virtually) impossible to perform on a single processor in view of its excessive running time, the computation speed-up factor can rightly be taken to be larger than p or even infinite. This situation, which is the analogue of several men moving a heavy piece of machinery or furniture in a few minutes, whereas one of them could not move it at all, is sometimes referred to as parallel synergy.

A major issue in devising a parallel algorithm for a given problem is the way in which the computational load is divided between the multiple processors. The most efficient scheme often depends both on the problem and on the parallel machine's architecture.

Example

Consider the problem of constructing the list of all prime numbers in the interval [1, n] for a given integer n > 0. A simple algorithm that can be used for this computation is the sieve of Eratosthenes. Start with the list of numbers 1, 2, 3, 4, ... , n represented as a "mark" bit-vector initialized to 1000 . . . 00. In each step, the next unmarked number m (associated with a 0 in element m of the mark bit-vector) is a prime. Find this element m and mark all multiples of m beginning with m^2. When $m^2 > n$, the computation stops and all unmarked elements are prime numbers. The computation steps for n = 30 are shown in the figure below



```
  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
m=2

  2  3     5     7     9    11    13    15    17    19    21    23    25    27    29
   m=3

  2  3     5     7          11    13          17    19          23    25          29
         m=5

  2  3     5     7          11    13          17    19          23                29
               m=7
```

Figure 3.17: The Block Diagram

3.7.1 PARALLEL PROCESSING UPS AND DOWNS

L. F. Richardson, a British meteorologist, was the first person to attempt to forecast the weather using numerical computations. He started to formulate his method during the First World War while serving in the army ambulance corps. He estimated that predicting the weather for a 24-hour period would require 64,000 slow "computers" (humans + mechanical calculators) and even then, the forecast would take 12 hours to complete. He had the following idea or dream:

Imagine a large hall like a theater. The walls of this chamber are painted to form a map of the globe. A myriad of computers at work upon the weather on the part of the map where each sits, but each computer attends to only one equation or part of an equation. The work of each region is coordinated by an official of higher rank. Numerous little 'night signs' display the instantaneous values so that neighbouring computers can read them. One of [the conductor's] duties is to maintain a uniform speed of progress in all parts of the globe. But instead of waving a baton, he turns a beam of rosy light upon any region that is running ahead of the rest, and a beam of blue light upon those that are behindhand.

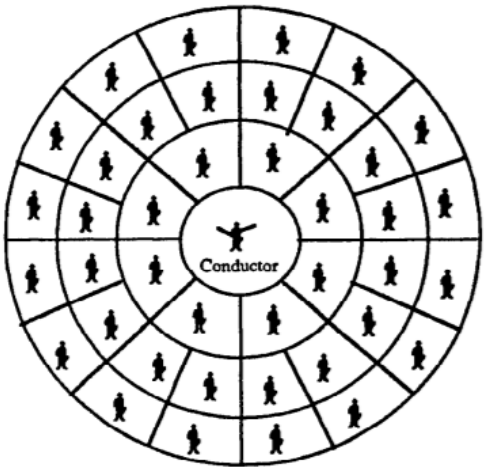


Figure 3.18: Pictorial Representation of Richardsons example

3.7.2 Types of Parallelism: A Taxonomy

Parallel computers can be divided into two main categories of control flow and data flow. Control-flow parallel computers are essentially based on the same principles as the sequential or von Neumann computer, except that multiple instructions can be executed at any given time. Data-flow parallel computers, sometimes referred to as "non-von Neumann," are completely different in that they have no pointer to active instruction(s) or a locus of control. The control is totally distributed, with the availability of operands triggering the activation of instructions.

In 1966, M. J. Flynn proposed a four-way classification of computer systems based on the notions of instruction streams and data streams. Flynn's classification has become standard and is widely used. Flynn coined the abbreviations SISD, SIMD, MISD, and MIMD (pronounced "sis-dee," "sim-dee," and so forth) for the four classes of computers shown in Fig. 1.7.3, based on the number of instruction streams (single or multiple) and data streams (single or multiple). The SISD class represents ordinary "uniprocessor" machines. Computers in the SIMD class, with several processors directed by instructions issued from a central control unit, are sometimes characterized as "array processors." Machines in the MISD category have not found widespread application, but one can view them as generalized pipelines in which each stage performs a relatively complex operation (as opposed to ordinary pipelines found in modern processors where each stage does a very simple instruction-level operation).

The MIMD category includes a wide class of computers. For this reason, in 1988, E. E. Johnson proposed a further classification of such machines based on their memory structure (global or distributed) and the mechanism used for communication/synchronization (shared variables or message passing). Again, one of the four categories (GMMP) is not widely used. The GMSV class is what is loosely referred to as (shared-memory) multiprocessors.

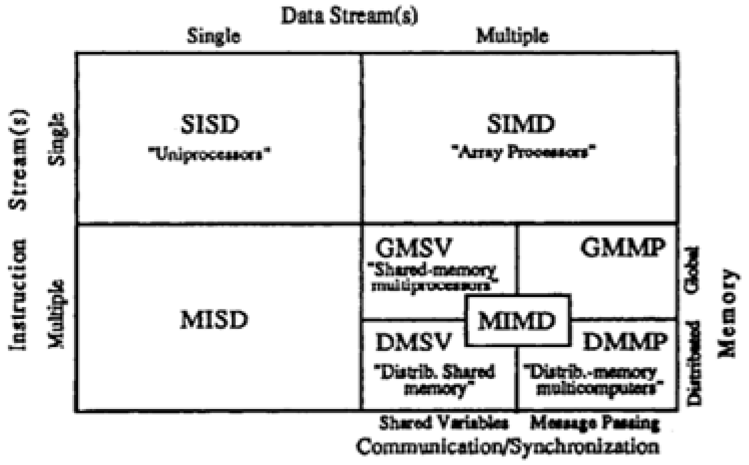


Figure 1.19: Classes of Computer according to Flynn

At the other extreme, the DMMP class is known as (distributed-memory) multicomputers. Finally, the DMSV class, which is becoming popular in view of combining the implementation ease of distributed memory with the programming ease of the shared-variable scheme, is sometimes called distributed shared memory. When all processors in a MIMD-type machine execute the same program, the result is sometimes referred to as single-program multipledata [SPMD (spim-dee)]. Although the Figigure lumps all SIMD machines together, there are in fact variations similar to those suggested above for MIMD machines. At least conceptually, there can be shared-memory and distributed-memory SIMD machines in which the processors communicate by means of shared variables or explicit message passing. Anecdote. The Flynn–Johnson classification of Figure contains eight four-letter abbreviations. There are many other such abbreviations and acronyms in parallel processing, examples being CISC, NUMA, PRAM, RISC, and VLIW. Even our journals (JPDC, TPDS) and conferences (ICPP, IPPS, SPDP, SPAA) have not escaped this fascination with four-letter abbreviations. The author has a theory that an individual cannot be considered a successful computer architect until she or he has coined at least one, and preferably a group of two or four, such abbreviations! Toward this end, the author coined the acronyms SINC and FINC (Scant/Full Interaction Network Cell) as the communication network counterparts to the popular RISC/CISC dichotomy. Alas, the use of these acronyms is not yet as widespread as that of RISC/CISC. In fact, they are not used at all.

3.7.3 Roadblocks to Parallel Computing

Over the years, the enthusiasm of parallel computer designers and researchers has been counteracted by many objections and cautionary statements. The list begins with the less serious, or obsolete, objections and ends with Amdahl's law, which perhaps constitutes the most important challenge facing parallel computer designers and users.

1. Grosch's law (economy of scale applies, or computing power is proportional to the square of cost). If this law did in fact hold, investing money in p processors would be foolish as a single computer with the same total cost could offer p^2 times the performance of one such processor. Grosch's law was formulated in the days of giant mainframes and actually did hold for those machines. In the early days of parallel processing, it was offered as an argument against the cost-effectiveness of parallel machines. However, we can now safely retire this law, as we can buy more MFLOPS computing power per dollar by spending on micros rather than on supers. Note that even if this law did hold, one could counter that there is only one "fastest" single-processor computer and it has a certain price; you cannot get a more powerful one by spending more.
2. Minsky's conjecture (speed-up is proportional to the logarithm of the number p of processors). This conjecture has its roots in an analysis of data access conflicts assuming random distribution of addresses. These conflicts will slow everything down to the point that quadrupling the number of processors only doubles the performance. However, data access patterns in real applications are far from random. Most applications have a pleasant amount of data access regularity and

locality that help improve the performance. One might say that the log p speed-up rule is one side of the coin that has the perfect speed-up p on the flip side. Depending on the application, real speed-up can range from log p to p (p/log p being a reasonable middle ground).

3. The tyranny of IC technology (because hardware becomes about 10 times faster every 5 years, by the time a parallel machine with 10-fold performance is designed and implemented, uniprocessors will be just as fast). This objection might be valid for some special-purpose systems that must be built from scratch with “old” technology. Recent experience in parallel machine design has shown that off-theshelf components can be used in synthesizing massively parallel computers. If the design of the parallel processor is such that faster microprocessors can simply be plugged in as they become available, they too benefit from advancements in IC technology. Besides, why restrict our attention to parallel systems that are designed to be only 10 times faster rather than 100 or 1000 times?
4. The tyranny of vector supercomputers (vector supercomputers, built by Cray, Fujitsu, and other companies, are rapidly improving in performance and additionally offer a familiar programming model and excellent vectorizing compilers; why bother with parallel processors?). Besides, not all computationally intensive applications deal with vectors or matrices; some are in fact quite irregular. Note, also, that vector and parallel processing are complementary approaches. Most current vector supercomputers do in fact come in multiprocessor configurations for increased performance.
5. The software inertia (billions of dollars’ worth of existing software makes it hard to switch to parallel systems; the cost of converting the “dusty decks” to parallel programs and retraining the programmers is prohibitive). This objection is valid in the short term; however, not all programs needed in the future have already been written. New

applications will be developed and many new problems will become solvable with increased performance. Students are already being trained to think parallel. Additionally, tools are being developed to transform sequential code into parallel code automatically. In fact, it has been argued that it might be prudent to develop programs in parallel languages even if they are to be run on sequential computers. The added information about concurrency and data dependencies would allow the sequential computer to improve its performance by instruction prefetching, data caching, and so forth.

3.8 PIPELINING

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

Pipeline system is like the modern day assembly line setup in factories. For example in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

Types of Pipeline:
It is divided into 2 categories:

- **Arithmetic Pipeline-** Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc.

- **Instruction Pipeline-** In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.
- **Pipeline Conflicts**
- There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:
- **Timing Variations:** All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.
- **Data Hazards:** When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results. Branching In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its

result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

- **Interrupts:** Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.
- **Data Dependency:** It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

Advantages of Pipelining

- The cycle time of the processor is reduced.
- It increases the throughput of the system
- It makes the system reliable.

Disadvantages of Pipelining

- The design of pipelined processor is complex and costly to manufacture.
- The instruction latency is more.

Pipelining refers to the technique in which a given task is divided into a number of subtasks that need to be performed in sequence. Each subtask is performed by a given functional unit. The units are connected in a serial fashion and all of them operate simultaneously. The use of pipelining improves the performance compared to the traditional sequential execution of tasks. Figure 3.20 shows an illustration of the basic difference between executing four subtasks of a given instruction (in this case fetching F, decoding D, execution E, and writing the results W) using pipelining and sequential processing.

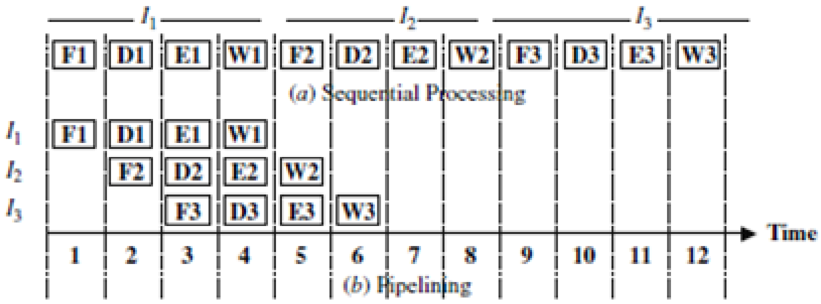


Figure 3.20: Pictorial Representation of a simple Pipelining Example

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

As can be seen from the figure 3.20, 13 time units are needed to finish executing 10 instructions (I1 to I10). This is to be compared to 40 time units if sequential processing is used (ten instructions each requiring four time units).

4.0 CONCLUSION

Computer memory is central to the operation of a modern computer system; it stores data or program instructions on a temporary or permanent basis for use in a computer. However, there is an increasing gap between the speed of memory and the speed of microprocessors. In this paper, various memory management and optimization techniques are reviewed to reduce the gap, including the hardware designs of the memory organization such as memory hierarchical structure and cache design; the memory management techniques varying from replacement algorithms to optimization techniques; and virtual memory strategies from a primitive bare-machine approach to paging and segmentation strategies.

5.0 SUMMARY

This module studied the memory system of a computer, starting with the organisation of its main memory, which, in some simple systems, is the only form of data storage to the understanding of more complex systems and the additional components they carry. Cache systems, which aim at speeding up access to the primary storage were also studied, and there was a greater focus on virtual memory systems, which make possible the transparent use of secondary storage as if it was main memory, by the processor.

6.0 TUTOR MARKED ASSIGNMENT

1. Consider the execution of 500 instructions on a five-stage pipeline machine. Compute the speed-up due to the use of pipelining given that the probability of an instruction being a branch is p = 0.3? What must be the value of p and the expected number of branch instructions such that a speed-up of at least 4 is possible? What must be the value of p such that a speed-up of at least 5 is possible? Assume that each stage takes one cycle to perform its task.
2. A computer system has a three-stage pipeline consisting of a Fetch unit (F), a Decode unit (D), and an Execute (E) unit. Determine (using the space–time chart) the time required to execute 20 sequential instructions using two-way interleaved memory if all three units require the use of the memory simultaneously.
3. What is the average instruction processing time of a five-stage instruction pipeline for 36 instructions if conditional branch instructions occur as follows: I5, I7, I10, I25, I27. Use both the space–time chart and the analytical model.
4. Parallelism in everyday life Discuss the various forms of parallelism used to speed up the following processes:
    - Student registration at a university.
    - Shopping at a supermarket.
    - Taking an elevator in a high-rise building

7.0 REFERENCES/FURTHER READING

Fundamentals of Computer Organization and Architecture, by M. Abd-El-Barr and H. El-Rewini ISBN 0-471-46741-3 Copyright # 2005 John Wiley & Sons, Inc.

Stone, H. S., High-Performance Computer Architecture, Addison–Wesley, 1993.

IEEE Trans. Computers, journal published by IEEE Computer Society; has occasional special issues on parallel and distributed processing (April 1987, December 1988, August 1989, December 1991, April 1997, April 1998).

Varma, A., and C. S. Raghavendra, Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice, IEEE Computer Society Press, 1994.

Zomaya, A. Y. (ed.), Parallel and Distributed Computing Handbook, McGraw-Hill, 1996.

MODULE TWO

1.0 INTRODUCTION

2.0 OBJECTIVES

3.0 MAIN CONTENTS

UNIT ONE: MEMORY ADDRESSING

3.1.1 What is memory addressing mode?
3.1.2 Modes of addressing
3.1.3 Number of addressing modes
3.1.4 Advantages of addressing modes
3.1.5 Uses of addressing modes

UNIT TWO: ELEMENTS OF MEMORY HIERARCHY

3.2.1 What is memory hierarchy
3.2.2 Memory hierarchy diagram
3.2.3 Characteristics of memory diagram
3.2.4 Memory hierarchy design
3.2.5 Advantages of memory hierarchy

UNIT THREE: VIRTUAL MEMORY CONTROL SYSTEM

3.3.1 Memory management systems
3.3.2 Paging
3.3.3 Address mapping using paging
3.3.4 Address mapping using segments
3.3.5 address mapping using segmented paging
3.3.6 Multi-programming
3.3.7 virtual machines/memory and protection
3.3.8 Hierarchical memory systems
3.3.9 drawbacks that occur in virtual memories

4.0 CONCLUSION

5.0 SUMMARY

6.0 TUTOR MARKED ASSIGNMENT

7.0 REFERENCES/FURTHER READING

1.0 INTRODUCTION

In computing, a memory address is a reference to a specific memory location used at various levels by software and hardware**.** Memory addresses are fixed-length sequences of digits conventionally displayed and manipulated as unsigned integers. Such numerical semantic bases itself upon features of CPU, as well upon use of the memory like an array endorsed by various programming languages. There are many ways to locate data and instructions in primary memory and these methods are called "memory address modes".

Memory address modes determine the method used within the program to access data either from the Cache or the RAM.

2.0 OBJECTIVES

The objectives of this module include;

- To ensure students have adequate knowledge of memory addressing systems
- To carefully study through the elements of memory hierarchy
- To analyze virtual control systems

UNIT ONE

3.1 MEMORY ADDRESSING

3.1.1 What is memory addressing mode?

Memory addressing mode is the method by which an instruction operand is specified. One of the functions of a microprocessor is to execute a sequence of instructions or programs stored in a computer memory (register) in order to perform a particular task. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced. This technique is used by the computers to give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation. And as well reduce the number of bits in the addressing field of the instruction.

However, there are basic requirement for the operation to take effect. First, the must be an operator to indicate what action to take and secondly, there must be an operand that portray the data to be executed. For instance; if the numbers 5 and 2 are to be added to have a result, it could be expressed numerically as 5 + 2. In this expression, our operator is (+), or expansion, and the numbers 5 and 2 are our operands. It is important to tell the machine in a microprocessor how to get the operands to perform the task. The data stored in the operation code is the operand value or the result. A word that defines the address of an operand that is stored in memory is the effective address. The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing programs that are more efficient with respect to the number of instructions and execution time.

3.1.2 Modes of addressing

There are many methods for defining or obtaining the effective address of an operators directly from the register. Such approaches are known as modes of addressing. The programmes are usually written in a high-level language, as it is a simple way to describe the variables and operations to be performed on the variables by the programmer. The following are the modes of addressing;

ADDRESSING MODES	EXAMPLE INSTRUCTION	MEANING	WHEN TO USED
Register	ADD R4, R3	R4 <- R4 + R3	When a value is in a register
Immediate	ADD R4, #3	R4 <- R4 + R3	For constants

indexed	ADD R3, (R1 + R2)	R3 <- R3 + M [R1 + R2]	When addressing array;
			R1 = base of array

			R2 = index amount
Register Indirect	ADD R4, (R1)	R4 <- R4 + M [R1]	Accessing using a pointer or a computed address

Auto Increment	ADD R1, (R2)+	R1 <- R1 + M [R2]
		R2 <- R2 + d
			Use for stopping through array in a loop.
			R2 = start of array
			D = size of an element

Auto Decrement	ADD R1, - (R2)	R2 <- R2 – d
		R1 <- R1 + M [R2]
			Same as auto increment.
			Both can also be used to implement a stack push and pop

Direct	ADD R1, (1001)	R1 <- R1 + M [1001]	Useful in accessing static data

Note :
< - = assignment
M = the name for memory: M[R1] refers to contents of memory location whose address is given by the contents of R1

3.1.3 Number of addressing modes

The number of addressing modes are as follow;

a. Register Addressing Mode

In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2' registers.

b. Direct Addressing Mode and Indirect Address mode

In Direct Address Mode, the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction the address field specifies the actual branch address. But in the Indirect Address Mode, the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address. A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU. The effective address in these modes is obtained from the following computation:

Effective address = address part of instruction + content of CPU register.

The CPU register used in the computation may be the program counter, an index register, or a base register. In either case we have a different addressing mode which is used for a different application.

c. **Immediate Addressing Mode**

In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction. Immediate-mode instructions are useful for initializing registers to a constant value. It was mentioned previously that the address field of an instruction may specify either a memory word or a processor register. When the address field specifies a processor register, the instruction is said to be in the register mode.

d. **Register Indirect Addressing Mode**

In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction. A reference to the register is then equivalent to specifying a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

e. Indexed Addressing Mode

In this mode the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is stored in memory relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stored in the index register. Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value. The index register can be incremented to facilitate access to consecutive operands. Note that if an index type instruction does not include an address field in its format, the instruction converts to the register indirect mode of operation. Some computers dedicate one CPU register to function solely as an index register. This register is involved implicitly when the index-mode instruction is used. In computers with many processor registers, any one of the CPU registers can contain the index number. In such a case the register must be specified explicitly in a register field within the instruction format.

f. Auto Increment Mode and Auto Decrement Mode

This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction. However, because it is such a common requirement, some computers incorporate a special mode that automatically increments or decrements the content of the register after data access. The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory. Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated. To differentiate among the various addressing modes it is necessary to distinguish between the address part of the instruction and the effective address used by the control when executing the instruction. The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational type instruction. It is the address where control branches in response to a branch-type instruction.

g. **Relative Addressing Mode:**

In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address. The address part of the instruction is usually a signed number which can be either positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction. For instance, let’s assume that the program counter contains the number 682 and the address part of the instruction contains the number 21. The instruction at location 682 is read from memory during the fetch phase and the program counter is then incremented by one to 683. The effective address computation for the relative address mode is 683 + 21 = 704. This is 21 memory locations forward from the address of the next instruction. Relative addressing is often used with branch-type instructions when the branch address is in the area surrounding the instruction word itself. It results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the number of bits required to designate the entire memory address.

3.1.4 Advantages of addressing modes

The advantages of using the addressing mode are as follow;

a. To provide the user with programming flexibility by offering such facilities as memory pointers, loop control counters, data indexing, and programme displacement.

b. To decrease the counting of bits in the instruction pointing area.

3.1.5 Uses of addressing modes

Some direction set models, for instance, Intel x86 and its substitutions, had a pile ground-breaking area direction. This plays out an assessment of the fruitful operand location, anyway rather following up on that memory territory, it stacks the area that might have been gotten in the register. This may be significant during passing the area of a display part to a browse mode. It can similarly be a fairly precarious strategy for achieving a greater number of includes than average in one direction; for example, using such a direction with the keeping an eye on mode “base+ index+ balance” (unequivocal underneath) licenses one to assemble two registers and a consistent into a solitary unit in one direction.

UNIT TWO

3.2 ELEMENTS OF MEMORY HIERARCHY

3.2.1 What is memory hierarchy?

Memory is one of the important units in any computer system. Its serves as a storage for all the processed and the unprocessed data or programs in a computer system. However, due to the fact that most computer users often stored large amount of files in their computer memory devices, the use of one memory device in a computer system has become inefficient and unsatisfactory. This is because only one memory cannot contain all the files needed by the computer users and when the memory is large, it decreases the speed of the processor and the general performance of the computer system.

Therefore, to curb this challenges, memory unit must be divided into smaller memories for more storage, speedy program executions and the enhancement of the processor performance. The recently accessed files or programs must be placed in the fastest memory. Since the memory with large capacity is cheap and slow and the memory with smaller capacity is fast and costly. The organization of smaller memories to hold the recently accessed files or programs closer to the CPU is term memory hierarchy. These memories are successively larger as they move away from the CPU.

The strength and performance of memory hierarchy can be measured using the model below;

Memory_Stall_Cycles = IC*Mem_Refs * Miss_Rate * Miss_Penalty

Where,

IC	= Instruction Count
Mem_Refs	= Memory References per Instruction
Miss_Rate	= Fraction of Accesses that are not in the cache
Miss_Penalty	= Additional time to service the Miss

The memory hierarchy system encompasses all the storage devices used in a computer system. Its ranges from the cache memory, which is smaller in size but faster in speed to a relatively auxiliary memory which is larger in size but slower in speed. The smaller the size of the memory the costlier it becomes.

The element of the memory hierarchy includes

a. Cache memory,
b. Main memory and
c. Auxiliary memory

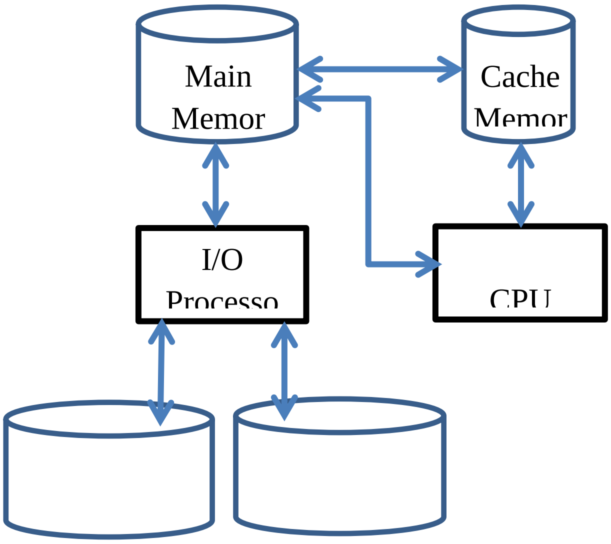
- **The cache memory** is the fastest and smallest memory. It is easily accessible by the CPU because it closer to the CPU. Cache memory is very costly compare to the main memory and the auxiliary memory.
- **The main memory** also known as primary memory, communicates directly to the CPU. Its also communicates to the auxiliary memory through the I/O processor. During program execution, the files that are not currently needed by the CPU are often moved to the auxiliary storage devices in order to create space in the main memory for the currently needed files to be stored. The main memory is made up of Random Access Memory (RAM) and Read Only Memory (ROM).
- **The auxiliary memory** is very large in size and relatively slow in speed. Its includes the magnetic tapes and the magnetic disks which are used for the storage and backup of removable files. The auxiliary memories store programs that are not currently needed by the CPU. They are very cheap when compare to the both cache and main memories.

3.2.2 Memory hierarchy diagram

The memory hierarchy system encompasses all the storage devices used in a computer system. Its ranges from fastest but smaller in size (cache memory) to a relatively fast but small in size (main memory) and more slower but larger in size (auxiliary memory). The cache memory is the smallest and fastest storage device, it is place closer to the CPU for easy accessed by the processor logic. More so, cache memory is helps to enhance the processing speed of the system by making available currently needed programs and data to the CPU at a very high speed. Its stores segment of programs currently processed by the CPU as well as the temporary data frequently needed in the current calculation

The main memory communicates directly to the CPU. It also very fast in speed and small in size. Its communicates to the auxiliary memories through the input/ output processor. The main memory provides a communication link between other storage devices. It contains the currently accessed data or programs. The unwanted data are transferred to the auxiliary memories to create more space in the main memory for the currently needed data to be stored. If the CPU needs a program that is outside the main memory, the main memory will call in the program from the auxiliary memories via the input/output processor. The main difference between cache and main memories is the access time and processing logic. The processor logic is often faster than that of the main memory access time.

The auxiliary memory is made up of the magnetic tape and the magnetic disk. They are employ in the system to store and backup large volume of data or programs that are not currently needed by the processor. In summary, the essence of dividing this memory into different levels of memory hierarchy is to make storage more efficient, reliable and economical for the users. As the storage capacity of the memory increases, the cost per bit for storing binary information decreases and the access time of the memory



becomes longer. The diagram of a memory hierarchy in presented in Figure 2.1 below.

2.3 Characteristics of Memory Hierarchy

There are numbers of parameters that characterized memory hierarchy. They stand as the principle on which all the levels of the memory hierarchy operate. These characteristics are;

a. Access type,
b. Capacity,
c. Cycle time,
d. Latency,
e. Bandwidth, and
f. Cost

a. **Access Time:** refers to the action that physically takes place during a read or write operation. When a data or program is moved from the top of the memory hierarchy to the bottom , the access time automatically increases. Hence, the interval of time at which the data are request to read or write is called Access time.

b. **Capacity:** the capacity of a memory hierarchy often increased when a data is moved from the top of the memory hierarchy to the bottom. The capacity of a memory hierarchy is the total amount of data a memory can store. The capacity of a memory level is usually measured in bytes.

c. **Cycle time**: is defined as the time elapsed from the start of a read operation to the start of a subsequent read.

d. **Latency:** is defined as the time interval between the request for information and the access to the first bit of that information.

e. **Bandwidth:** this measures the number of bits that can be accessed per second.

f. **Cost:** the cost of a memory level is usually specified as dollars per megabytes. When the data is moved from bottom of the memory hierarchy to top, the cost for each bit increases automatically. This means that an internal memory is expensive compared to external memory.

3.2.4 Memory Hierarchy Design

The memory in a computer can be divided into five hierarchies based on the speed as well as use. The processor can move from one level to another based on its requirements. The five hierarchies in the memory are registers, cache, main memory, magnetic discs, and magnetic tapes. The first three hierarchies are the primary memory (volatile memories) which mean when there is no power, and then automatically they lose their stored data. The last two hierarchies are the secondary memories (nonvolatile) which means they store the data permanently. Generally, a memory element is a set of storage devices that stores binary in bits. This set of storage devices can be classified into two categories such as; the primary memory and the secondary memory. The primary memory is directly accessible by the processor, it is also known as internal memory. This memory includes main, cache, as well as CPU registers. Furthermore, the secondary memory can only be accessed by the processor through an input/output module, and it is also known as external memory. This memory includes an optical disk, magnetic disk, and magnetic tape. The memory hierarchy design is presented in Figure 2.2 below.

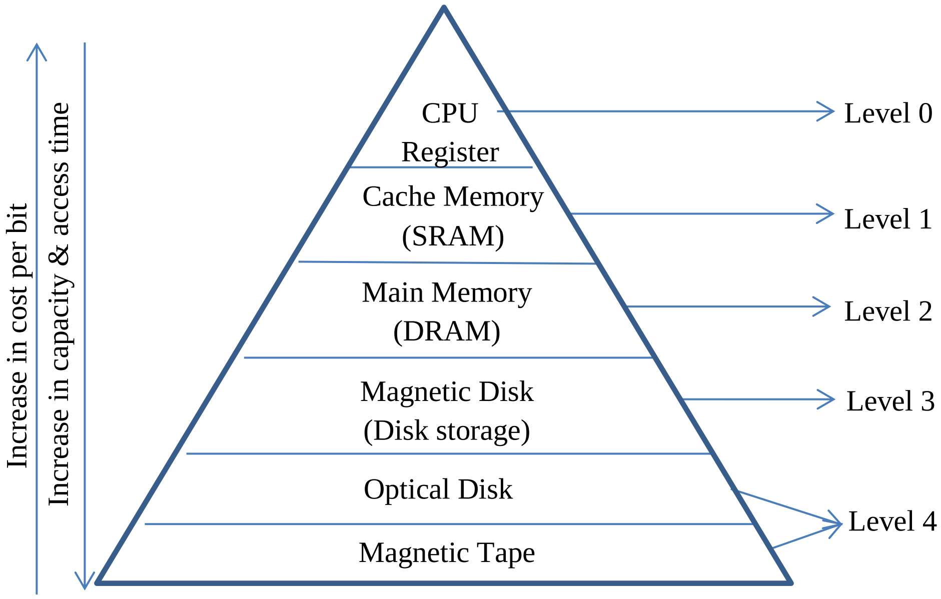


Figure 2.2.2**:** Memory hierarchy design

1. operation. Normally, a complex instruction set computer uses many registers to accept main memory.
2. **Cache Memory:** Cache memory can also be found in the processor, however rarely it may be another integrated circuit (IC) which is separated into levels. The cache holds the chunk of data which are frequently used from main memory. When the processor has a single core then it will have two (or) more cache levels rarely. Present multi-core processors will be having three, 2-levels for each one core, and one level is shared.
3. **Main Memory:** This is the memory unit that communicate directly to the CPU. It is the primary storage unit in a computer system. the main stores data or program currently used by the CPU during operation. It is very fast in terms of access time and it is made up of RAM and ROM.
4. **Magnetic Disks:** The magnetic disks is a circular plates fabricated of plastic or metal by magnetized material. Frequently, two faces of the disk are utilized as well as many disks may be stacked on one spindle by read or write heads obtainable on every plane. All the disks in computer turn jointly at high speed. The tracks in the computer are nothing but bits which are stored within the magnetized plane in spots

next to concentric circles. These are usually separated into sections which are named as sectors.

5. **Magnetic Tape:** This tape is a normal magnetic recording which is designed with a slender magnetizable covering on an extended, plastic film of the thin strip. This is mainly used to back up huge data. Whenever the computer requires to access a strip, first it will mount to access the data. Once the data is allowed, then it will be unmounted. The access time of memory will be slower within magnetic strip as well as it will take a few minutes for accessing a strip.

3.2.5 Advantages of Memory Hierarchy

The advantages of a memory hierarchy include the following.

a. Memory distributing is simple and economical

b. Removes external destruction

c. Data can be spread all over

d. Permits demand paging & pre-paging

e. Swapping will be more proficient

UNIT THREE

3.3 VIRTUAL MEMORY CONTROL SYSTEM

3.3.1 Memory management systems

In a multiprogramming system, there is a need for a high capacity memory. This is because most of the programs are often stored in the memory. The programs must be moved around the memory to change the space of memory used by a particular program and as well prevent a program from altering other programs during read and write. Hence, the memory management system becomes necessary. The movement of these programs from one level of memory hierarchy to another is known as memory management. Memory management system encompasses both the hardware and the software in its operations. It is the collection of hardware and software procedures for managing all the programs stored in the memory. The memory management software is part of the main operating system available in many computers. In this study, we are concerned with the hardware unit of the memory management system.

Components of memory management system:

The principal components of the memory management system are;

a. A facility for dynamic storage relocation that maps logical memory references into physical memory addresses.

b. A provision for sharing common programs stored in memory by different users.

c. Protection of information against unauthorized access between users and preventing users from changing operating system functions. The dynamic storage relocation hardware is a mapping process similar to the paging system

3.3.2 Paging

In memory management, paging can be described as a storage mechanism that allows operating system (OS) to retrieve processes from the secondary storage into the main memory in the form of pages. It is a function of memory management where a computer will store and retrieve data from a device’s secondary storage to the primary storage. Memory management is a crucial aspect of any computing device, and paging specifically is important to the implementation of virtual memory.

In the Paging method, the main memory is divided into small fixed-size blocks of physical memory, which is called frames. The size of a frame should be kept the same as that of a page to have maximum utilization of the main memory and to avoid external fragmentation. Paging is used for faster access to data, and it is a logical concept. For instance, if the main memory size is 16 KB and Frame size is 1 KB. Here, the main memory will be divided into the collection of 16 frames of 1 KB each. There are 4 separate processes in the system that is A1, A2, A3, and A4 of 4 KB each. Here, all the processes are divided into pages of 1 KB each so that operating system can store one page in one frame. At the beginning of the process, all the frames remain empty so that all the pages of the processes will get stored in a contiguous way. A typical paging process is presented in Figure 3.1 below.

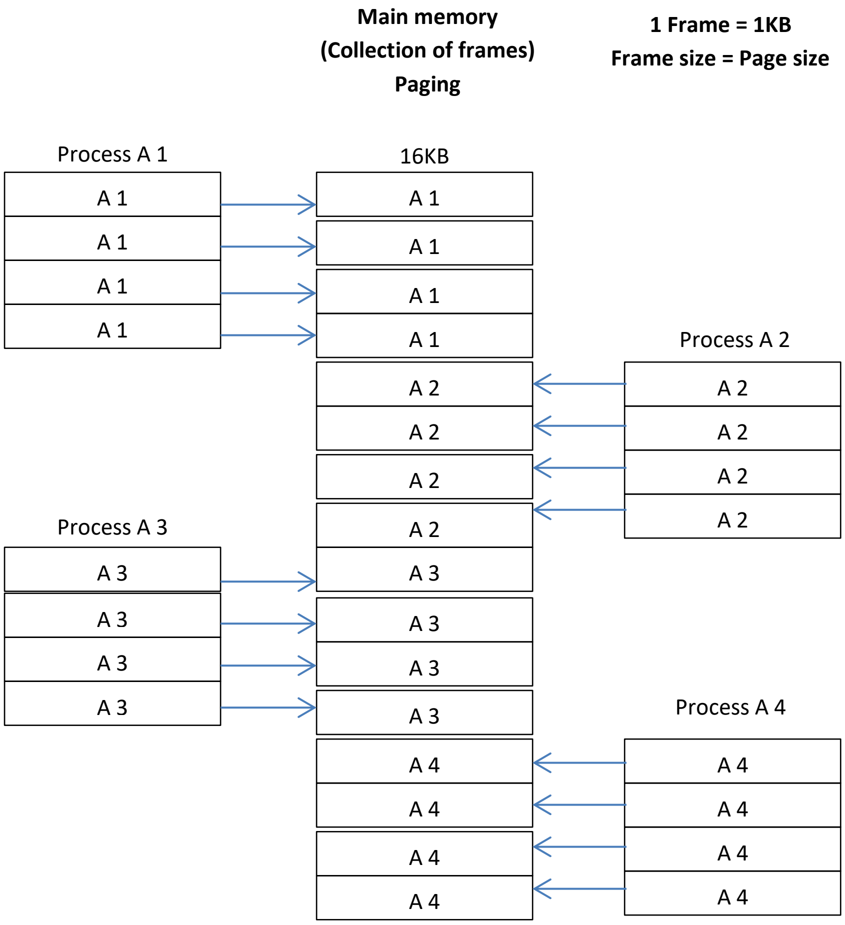


Figure 2.3.1: Paging process

From the above diagram you can see that A2 and A4 are moved to the waiting state after some time. Therefore, eight frames become empty, and so other pages can be loaded in that empty blocks. The process A5 of size 8 pages (8 KB) are waiting in the ready queue.

In conclusion, paging is a function of memory management where a computer will store and retrieve data from a device's secondary storage to the primary storage. Memory management is a crucial aspect of any computing device, and paging specifically is important to the implementation of virtual memory.

3.3.2.1. Paging Protection

The paging process should be protected by using the concept of insertion of an additional bit called Valid/Invalid bit. Paging Memory protection in paging is achieved by associating protection bits with each page. These bits are associated with each page table entry and specify protection on the corresponding page.

3.3.2.2. Advantages and Disadvantages of Paging

Advantages

The following are the advantages of using Paging method:

a. No need for external Fragmentation
b. Swapping is easy between equal-sized pages and page frames.
c. Easy to use memory management algorithm

Disadvantages

The following are the disadvantages of using Paging method

a. May cause Internal fragmentation
b. Page tables consume additional memory.
c. Multi-level paging may lead to memory reference overhead.

3.3.3 Address mapping using paging

The table implementation of the address mapping is simplified if the information in the address space and the memory space are each divided into groups of fixed size. The physical memory is broken down into groups of equal size called blocks, which may range from 64 to 4096 words each. The term page refers to groups of address space of the same size. For example, if a page or block consists of 1K words, address space is divided into 1024 pages and main memory is divided into 32 blocks. Although both a page and a block are split into groups of 1K words, a page refers to the organization of address space, while a block refers to the organization of memory space. The programs are also considered to be split into pages. Portions of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term page frame is sometimes used to denote a block.

For instance, if a computer has an address space of 8K and a memory space of 4K. If we split each into groups of 1K words we obtain eight pages and four blocks**.** At any given time, up to four pages of address space may reside in main memory in any one of the four blocks. The mapping from address space to memory space is facilitated if each virtual address is considered to be represented by two numbers: a page number address and a line within the page. In a computer with '1! words per page, p bits are used to specify a line address and the remaining high-order bits of the virtual address specify the page number**.** a virtual address has 13 bits. Since each page consists of 210 = 1024 words, the high order three bits of a virtual address will specify one of the eight pages and the low-order 10 bits give the line address within the page. Note that the line address in address space and memory space is the same; the only mapping required is from a page number to a block number. The organization of the memory mapping table in a paged system is shown in Figure 3.1. The memory-page table consists of eight words, one for each page. The address in the page table denotes the page number and the content of the word gives the block number where that page is stored in main memory. The table shows that pages 1, 2, 5, and 6 are now available in main memory in blocks 3, 0, 1, and 2, respectively. A presence bit in each location indicates whether the page has been transferred from auxiliary memory into main memory. A 0 in the presence bit indicates that this page is not available in main memory. The CPU references a word in memory with a virtual address of 13 bits. The three high-order bits of the virtual address specify a page number and also an address for the memory-page table. The content of the word in the memory page table at the page number address is read out into the memory table buffer register. If the presence bit is a 1, the block number thus read is transferred to the two high-order bits of the main memory address register. The line number from the virtual address is transferred into the 10 low-order bits of the memory address register. A read signal to main memory transfers the content of the word to the main memory buffer register ready to be used by the CPU. If the presence bit in the word read from the page table is 0, it signifies that the content of the word referenced by the virtual address does not reside in main memory. A call to the operating system is then generated to fetch the required page from auxiliary memory and place it into main memory before resuming computation.

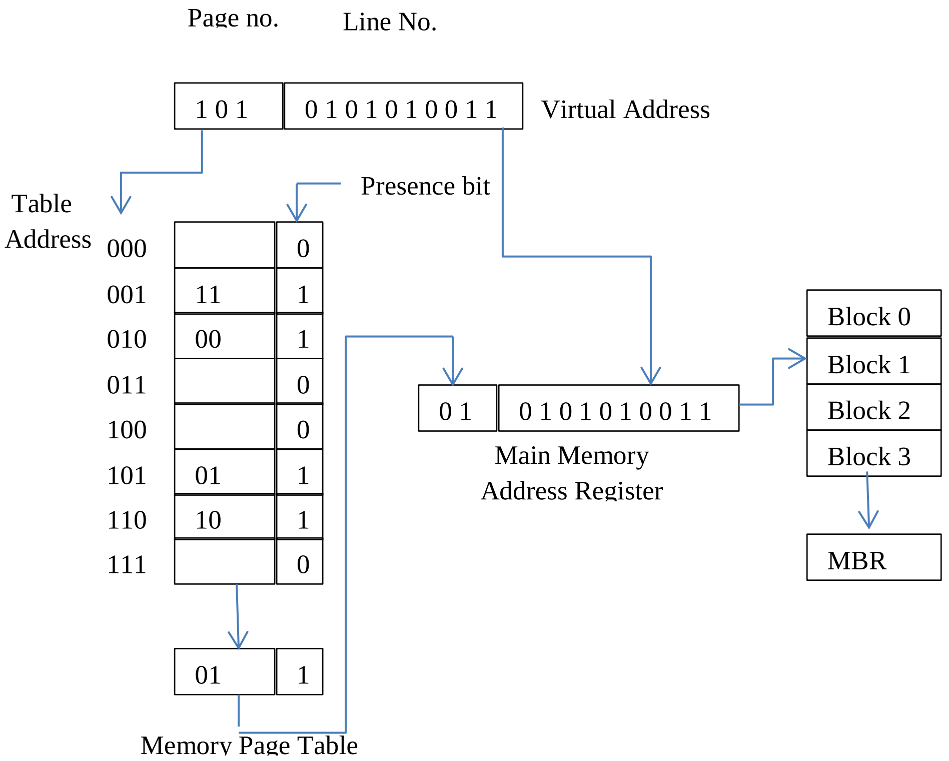


Figure 2.3.2. Memory Table in Paged System

3.3.4 Address mapping using segments

Another mapping process similar to paging system is the dynamic storage relocation hardware. Due to the large size of program and its logical structures, the fixed page size employ in the virtual memory system has really pose a lot of challenges in memory management. During program execution, the speed of the processor is usually affected. However, it is very appropriate to disintegrate these programs and data into segments for effective management and execution. A segment is a set of logically related instructions or data elements associated with a given name. Its can be generated by the operating system or by the programmer. Examples of segments include an array of data, a subroutine, a table of symbols, or a user's program.

The address generated by a segmented program is called a logical address . This is similar to a virtual address except that logical address space is associated with variable-length segments rather than fixed-length pages. The logical address may be larger than the physical memory address as in virtual memory, but it may also be equal, and sometimes even smaller than the length of the physical memory address. In addition to relocation information, each segment has protection information associated with it. Shared programs are placed in a unique segment in each user's logical address space so that a single physical copy can be shared. The function of the memory management unit is to map logical addresses into physical addresses similar to the virtual memory mapping concept.

3.3.5 Address mapping using segmented paging

One of the properties of logical space is that it uses variable-length segments. The length of each segment is allowed to grow and contract according to the needs of the program being executed. One way of specifying the length of a segment is by associating with it a number of equal-size pages. To see how this is done, consider the logical address shown in Figure2.3.3. The logical address is partitioned into three fields. The segment field specifies a segment number. The page field specifies the page within the segment and the word field gives the specific word within the page. A page field of k bits can specify up to 2' pages. A segment number may be associated with just one page or with as many as 1!-pages. Thus the length of a segment would vary according to the number of pages that are assigned to it.

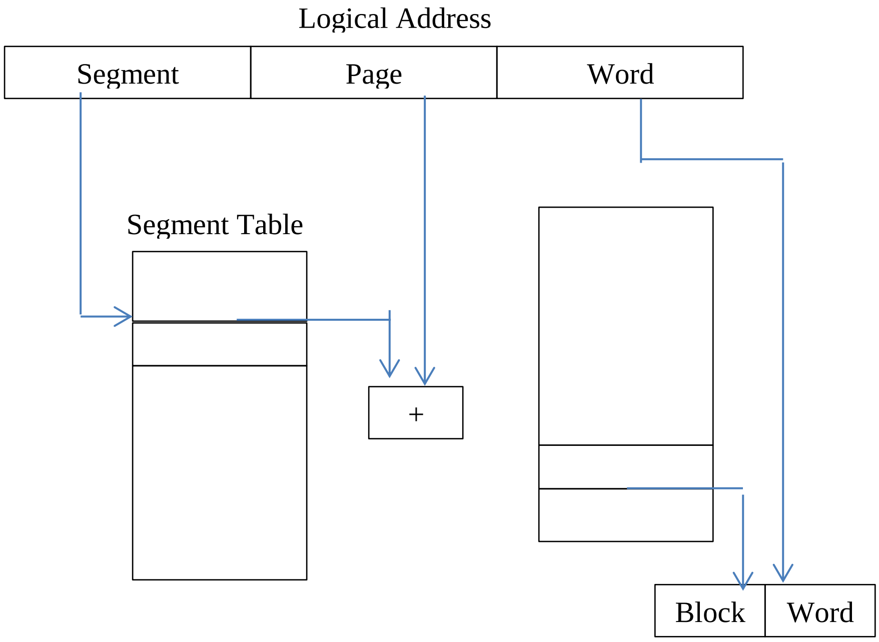


Figure 2.3.3 Address mapping using segmented paging

The mapping of the logical address into a physical address is done by means of two tables, as shown in Figure 3.3. The segment number of the logical address specifies the address for the segment table. The entry in the segment table is a pointer address for a page table base. The page table base is added to the page number given in the logical address. The sum produces a pointer address to an entry in the page table. The value found in the page table provides the block number in physical memory. The concatenation of the block field with the word field produces the final physical mapped address. The two mapping tables may be stored in two separate small memories or in main memory. In either case, a memory reference from the CPU will require three accesses to memory: one from the segment table, one from the page table, and the third from main memory. This would slow the system significantly when compared to a conventional system that requires only one reference to memory. To avoid this speed penalty, a fast associative memory is used to hold the most recently referenced table entries. This type of memory is sometimes called a translation look aside buffer, abbreviated TLB. Thus the mapping process is first attempted by associative search with the given segment and page numbers. If it succeeds, the mapping delay is only that of the associative memory. If no match occurs, the slower table mapping of Figure 3.3 is used and the result transformed into the associative memory for future reference.

3.3.6 Multi-programming

Multiprogramming is the basic form of parallel processing in which several programs are run at the same time on a single processor. Since there is only one processor, there can be no true simultaneous execution of different programs. Instead, the operating system executes part of one program, then part of another, and so on. To the user it appears that all programs are executing at the same time. More so, if the machine has the capability of causing an interrupt after a specified time interval, then the operating system will execute each program for a given length of time, regain control, and then execute another program for a given length of time, and so on. When this mechanism is not in place, the operating system has no choice but to begin to execute a program with the expectation, but not the certainty, that the program will eventually return control to the operating system.

If the machine has the capability of protecting memory, then a bug in one program may interfere with the execution of other programs. In a system without memory protection, one program can change the contents of storage assigned to other programs or even the storage assigned to the operating system. The resulting system crashes are not only disruptive, they may be very difficult to debug since it may not be obvious which of several programs is at fault. Multiprogramming operating system has the ability to execute multiple programs using one processor machine. For example, a user can use MS-Excel , download apps, transfer data from one point to another point, Firefox or Google Chrome browser, and more at a same time. Multiprogramming operating system allows to execute multiple processes by monitoring their process states and switching in between processes. It executes multiple programs to avoid CPU and memory underutilization. It is also called as Multiprogram Task System. It is faster in processing than Batch Processing system

3.3.6.1. Advantages and Disadvantages of Multiprogramming

Below are the Advantages and disadvantages of Multiprogramming

Advantages of Multiprogramming:

a. CPU never becomes idle
b. Efficient resources utilization
c. Response time is shorter
d. Short time jobs completed faster than long time jobs
e. Increased Throughput

Disadvantages of Multiprogramming:

a. Long time jobs have to wait long
b. Tracking all processes sometimes difficult
c. CPU scheduling is required
d. Requires efficient memory management
e. User interaction not possible during program execution

3.3.7 Virtual machines/memory and protection

Memory protection can be assigned to the physical address or the logical address. The protection of memory through the physical address can be done by assigning to each block in memory a number of protection bits that indicate the type of access allowed to its corresponding block. Every time a page is moved from one block to another it would be necessary to update the block protection bits. A much better place to apply protection is in the logical address space rather than the physical address space. This can be done by including protection information within the segment table or segment register of the memory management hardware. The content of each entry in the segment table or a segment register is called a descriptor. A typical descriptor would contain, in addition to a base address field, one or two additional fields for protection purposes. A typical format for a segment descriptor is shown in Figure 3.2. The base address field gives the base of the page table address in a segmented-page organization or the block base address in a segment register organization. This is the address used in mapping from a logical to the physical address. The length field gives the segment size by specifying the maximum number of pages assigned to the segment. The length field is compared against the page number in the logical address. A size violation occurs if the page number falls outside the segment length boundary. Thus a given program and its data cannot access memory not assigned to it by the operating system.

Base address	**Length**	**Protection**

Figure 2.3.4: Format of a typical segment description

The protection field in a segment descriptor specifies the access rights available to the particular segment. In a segmented-page organization, each entry in the page table may have its own protection field to describe the access rights of each page. The protection information is set into the descriptor by the master control program of the operating system. Some of the access rights of interest that are used for protecting the programs residing in memory are:

- Full read and write privileges
- Read only (write protection)
- Execute only (program protection)
- System only (operating system protection)

Full read and write privileges are given to a program when it is executing its own instructions. Write protection is useful for sharing system programs such as utility programs and other library routines. These system programs are stored in an area of memory where they can be shared by many users. They can be read by all programs, but no writing is allowed. This protects them from being changed by other programs. The execute-only condition protects programs from being copied. It re-stricts the segment to be referenced only during the instruction fetch phase but not during the execute phase. Thus it allows the users to execute the segment program instructions but prevents them from reading the instructions as data for the purpose of copying their content. Portions of the operating system will reside in memory at any given time. These system programs must be protected by making them inaccessible to unauthorized users. The operating system protection condition is placed in the descriptors of all operating system programs to prevent the occasional user from accessing operating system segments.

3.3.8 Hierarchical memory systems

In the Computer System Design, Memory Hierarchy is used to enhance the organization of memory such that it can minimize the access time. It was developed based on a program behavior known as locality of references. Hierarchical memory system is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits. Generally, memory/storage is classified into 2 categories:

- **External Memory or Secondary Memory:** This is a permanent storage (non volatile) and does not lose any data when power is switched off. It is made up of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
- **Internal Memory or Primary Memory:** This memory is volatile in nature. it loses its data, when power is switched off. It is made up of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

Properties of Hierarchical Memory Organization

There are three important properties for maintaining consistency in the memory hierarchy these three properties are;

- Inclusion
- Coherence and
- Locality.

3.3.9 Drawbacks that occur in virtual memories

The following are the drawbacks of using virtual memory:

- Applications may run slower if the system is using virtual memory.
- Likely takes more time to switch between applications.
- Offers lesser hard drive space for your use.
- It reduces system stability.

4.0 CONCLUSION

The memory hierarchy system encompasses all the storage devices used in a computer system. Its ranges from fastest but smaller in size (cache memory) to a relatively fast but small in size (main memory) and more slower but larger in size (auxiliary memory). A memory element is a set of storage devices that stores binary in bits. They include; register, cache memory, main memory, magnetic disk and magnetic tape. This set of storage devices can be classified into two categories such as; the primary memory and the secondary memory.

5.0 SUMMARY

Memory addresses act just like the indexes of a normal array. The computer can access any address in memory at any time (hence the name "random access memory"). It can also group bytes together as it needs to form larger variables, arrays, and structures. Memory hierarchy is the hierarchy of memory and storage devices found in a computer system. It ranges from the slowest but high capacity auxiliary memory to the fastest but low capacity cache memory. Memory hierarchy is employed to balance this trade-off.

6.0 TUTOR MARKED ASSIGNMENT

1. What are the properties of hierarchical memory organization?
2. Explain the concept of memory protection
3. How do you perform address mapping using segments?

7.0 REFERENCES/FURTHER READING

John L. Hennessy and David A. Patterson (2012) Computer Architecture; A Qualitative Approach. Fifth (Ed.), Library of Congress Cataloging in Publication Data.

William Stallings (2003) Computer Organization Architecture; Designing for Performance Six Ed. Prentice Hall.

Rob Williams (2006) Computer System Architecture; ANetwork Approach. 2 (Ed.), Prentice Hall.

Mostafa Abd-El-Barr and Hesham El-Rewini (2005) Fundamentals of Computer Organization and Architecture. A John Wiley and Sons, Inc Publication.

Keith R. Mobley (2004) Maintenance Fundamentals. 2nd (Ed.), Elsevier Butterworth Heinemann.

William Stallings (2019) Computer Organization and Architecture; Designing for Performance. 11 (Ed.), Pearson.

MODULE THREE

1.0 INTRODUCTION

Control Unit is the part of the computer’s central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the

Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions. A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer. Examples of devices that require a CU are:

- Control Processing Units (CPUs)
- Graphics Processing Units (GPUs)

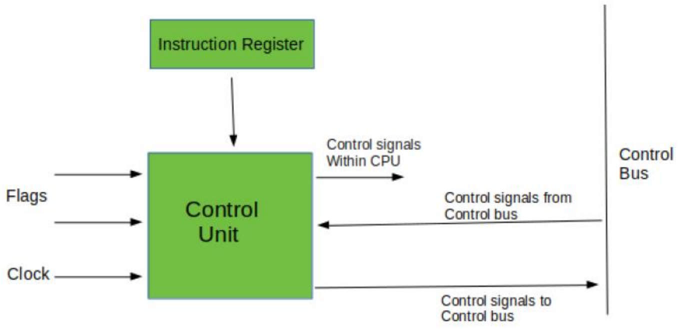

Instruction Register
Control signals Within CPU
Control Bus
Flags
Control Unit
Control signals from Control bus
Clock
Control signals to Control bus

Figure 3.1: Stucture of Control Unit

Major functions of the Control Unit –

- It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
- It interprets instructions.
- It controls data flow inside the processor.
- It receives external instructions or commands to which it converts to sequence of control signals.
- It controls many execution units (i.e. ALU, data buffers and registers) contained within a CPU.
- It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

2.0 OBJECTIVES

- Understand how digital systems may be divided into a data path and control logic
- Appreciate the different ways of implementing control logic
- Understand how shift registers and counters can be
- used to generate arbitrary pulse sequences
- Understand the circumstances that give rise to output glitches

UNIT ONE
HARDWARE CONTROL

3.1.1 Hardwired Control Unit

A hardwired control is a mechanism of producing control signals using Finite State Machines (FSM) appropriately. It is designed as a sequential logic circuit. The final circuit is constructed by physically connecting the components such as gates, flip flops, and drums. Hence, it is named a hardwired controller. In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we cannot modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode. As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for executive units of the computer.

This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.

3.1.2 Design of a hardwired Control Unit

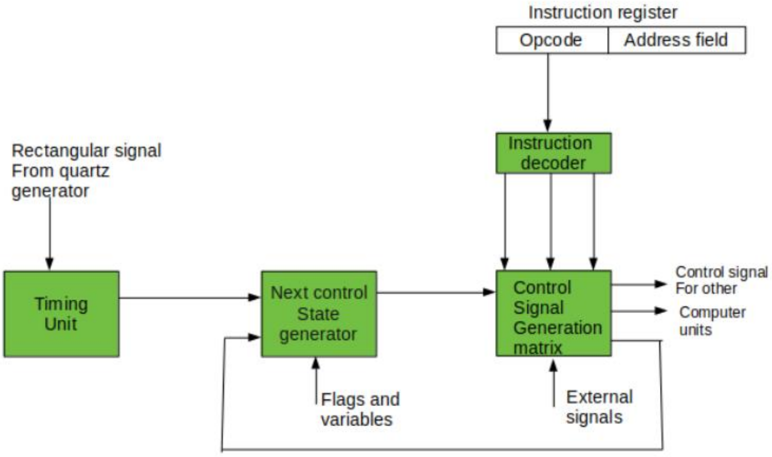


Figure 3.2.1: Hardwired Control Unit

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit. A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. This matrix combines these signals with the timing signals, which are generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered. A change of any of the earlier mentioned signals stimulates the change of the control unit state. This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing). The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle. The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

Advantages of Hardwired Control Unit:

1. Because of the use of combinational circuits to generate signals, Hardwired Control Unit is fast.
2. It depends on number of gates, how much delay can occur in generation of control signals.
3. It can be optimized to produce the fast mode of operation.
4. Faster than micro- programmed control unit.

Disadvantages of Hardwired Control Unit:

1. The complexity of the design increases as we require more control signals to be generated (need of more encoders & decoders)
2. Modifications in the control signals are very difficult because it requires rearranging of wires in the hardware circuit.
3. Adding a new feature is difficult & complex.
4. Difficult to test & correct mistakes in the original design.
5. It is Expensive.

UNIT TWO : MICRO-PROGRAMMED CONTROL

3.2.0 Introduction

A control unit whose binary control values are saved as words in memory is called a micro-programmed control unit. A controller results in the instructions to be implemented by constructing a definite collection of signals at each system clock beat. Each of these output signals generates one micro-operation including register transfer. Thus, the sets of control signals are generated definite micro-operations that can be saved in the memory.

Each bit that forms the microinstruction is linked to one control signal. When the bit is set, the control signal is active. When it is cleared the control signal turns inactive. These microinstructions in a sequence can be saved in the internal ’control’ memory. The control unit of a microprogram-controlled computer is a computer inside a computer.

3.2.1 Design of a Micro-Programmed Control Unit

The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution. In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.

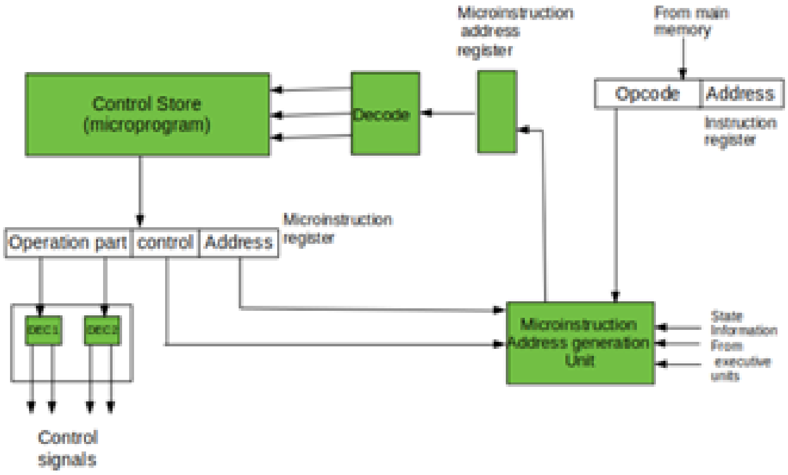

Microinstruction address register
From main memory
Control Store (microprogram)
Decode
Opcode
Address
Instruction register
Microinstruction register
Operation part
control
Address
DEC1
DEC2
Microinstruction Address generation Unit
State Information From executive units
Control signals

Figure 3.2.2: Single level control store

- With a single-level control store: In this, the instruction opcode from the instruction register is sent to the control store address register. Based on this address, the first microinstruction of a microprogram that interprets execution of this instruction is read to the microinstruction register. This microinstruction contains in its operation part encoded control signals, normally as few bit fields. In a set microinstruction field decoders, the fields are decoded. The microinstruction also contains the address of the next microinstruction of the given instruction microprogram and a control field used to control activities of the microinstruction address generator.

  The last mentioned field decides the addressing mode (addressing operation) to be applied to the address embedded in the ongoing microinstruction. In microinstructions along with conditional addressing mode, this address is refined by using the processor condition flags that represent the status of computations in the current program. The last microinstruction in the instruction of the given microprogram is the microinstruction that fetches the next instruction from the main memory to the instruction register.

- **With a two-level control store:** In this, in a control unit with a two-level control store, besides the control memory for microinstructions, a Nano-instruction memory is included. In such a control unit, microinstructions do not contain encoded control signals. The operation part of microinstructions contains the address of the word in the Nano-instruction memory, which contains encoded control signals. The Nano-instruction memory contains all combinations of control signals that appear in microprograms that interpret the complete instruction set of a given computer, written once in the form of Nano-instructions

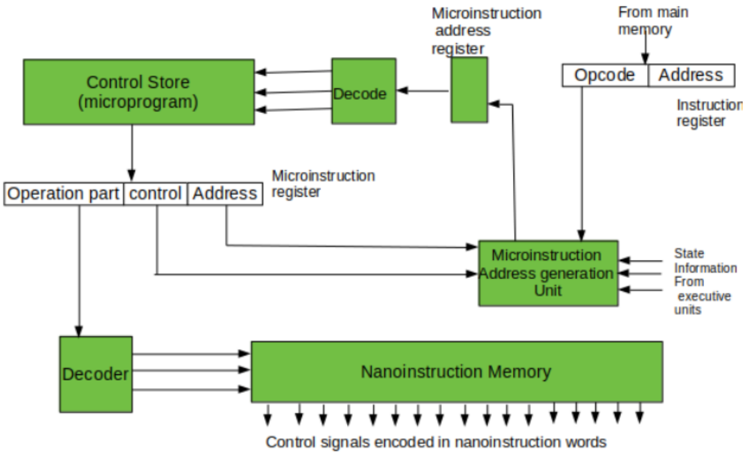
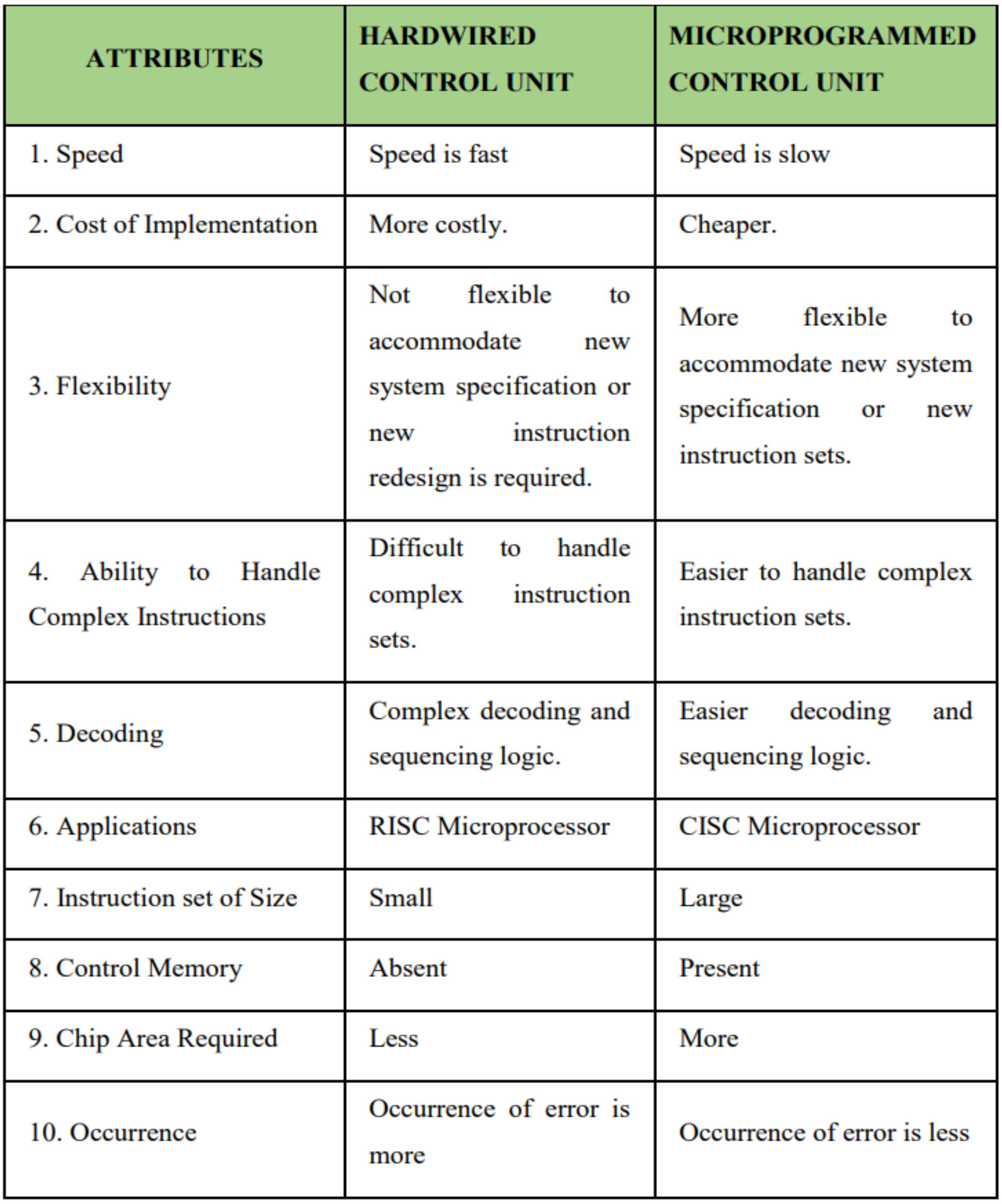

Microinstruction address register
From main memory
Control Store (microprogram)
Decode
Opcode
Address
Instruction register
Microinstruction register
Operation part
control
Address
Microinstruction Address generation Unit
State Information From executive units
Decoder
Nanoinstruction Memory
Control signals encoded in nanoinstruction words

Figure 3.2.3: Two-level control store

In this way, unnecessary storing of the same operation parts of microinstructions is avoided. In this case, microinstruction word can be much shorter than with the single level control store. It gives a much smaller size in bits of the microinstruction memory and, as a result, a much smaller size of the entire control memory. The microinstruction memory contains the control for selection of consecutive microinstructions, while those control signals are generated at the basis of Nano-instructions. In Nano-instructions, control signals are frequently encoded using 1 bit/ 1 signal method that eliminates decoding.

3.2.2 DIFFERENCES BETWEEN HARDWIRED AND MICROPROGRAMMED CONTROL



ATTRIBUTES	HARDWIRED CONTROL UNIT	MICROPROGRAMMED CONTROL UNIT
1. Speed	Speed is fast	Speed is slow
2. Cost of Implementation	More costly.	Cheaper.
3. Flexibility	Not flexible to accommodate new system specification or new instruction redesign is required.	More flexible to accommodate new system specification or new instruction sets.
4. Ability to Handle Complex Instructions	Difficult to handle complex instruction sets.	Easier to handle complex instruction sets.
5. Decoding	Complex decoding and sequencing logic.	Easier decoding and sequencing logic.
6. Applications	RISC Microprocessor	CISC Microprocessor
7. Instruction set of Size	Small	Large
8. Control Memory	Absent	Present
9. Chip Area Required	Less	More
10. Occurrence	Occurrence of error is more	Occurrence of error is less

Advantages OF Micro programmed Control Unit

There are the following advantages of microprogrammed control are as follows:

- It can more systematic design of the control unit.
- It is simpler to debug and change.
- It can retain the underlying structure of the control function.
- It can make the design of the control unit much simpler. Hence, it is inexpensive and less error-prone.

- It can orderly and systematic design process.
- It is used to control functions implemented in software and not hardware.
- It is more flexible.
- It is used to complex function is carried out easily.

Disadvantages of Microprogrammed Control Unit

There are the following disadvantages of microprogrammed control are as follows;

- Adaptability is obtained at more cost.
- It is slower than a hardwired control unit.

3.2.4 Organization of micro programmed control unit

- The control memory is assumed to be a ROM, within which all control information is permanently stored
- The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more microoperations for the data processor. Once these operations are executed, the control must determine the next address.
- The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory.
- While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.
- Thus a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.
- The next address generator is sometimes called a micro-program sequencer, as it determines the address sequence that is read from control memory.

- Typical functions of a micro-program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations.
- The control data register holds the present microinstruction while the next address is computed and read from memory.
- The data register is sometimes called a pipeline register.
- It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction.
- This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.
- The main advantage of the micro programmed control is the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes.
- If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory.

3.2.5 Types of Micro-programmed Control Unit

Based on the type of Control Word stored in the Control Memory (CM), it is classified into two types:

- **Horizontal Micro-programmed control Unit:**

  The control signals are represented in the decoded binary format that is 1 bit/CS. Example: If 53 Control signals are present in the processor than 53 bits are required. More than 1 control signal can be enabled at a time.

  - It supports longer control word.
  - It is used in parallel processing applications.
  - It allows higher degree of parallelism. If degree is n, n CS are enabled at a time.

- It requires no additional hardware(decoders). It means it is faster than Vertical Microprogrammed.
- It is more flexible than vertical microprogrammed

Vertical Micro-programmed control Unit :

The control signals re represented in the encoded binary format. For N control signals- Log2(N) bits are required.

- It supports shorter control words.
- It supports easy implementation of new control signals therefore it is more flexible.
- It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.
- Requires an additional hardware (decoders) to generate control signals, it implies it is slower than horizontal microprogrammed.
- It is less flexible than horizontal but more flexible than that of hardwired control unit.

UNIT THREE
ASYNCHRONOUS CONTROL

3.3.1 Introduction

Asynchronous (clockless) control is a method of control in which the time allotted for performing an operation depends on the time actually required for the operation, rather than on a predetermined fraction of a fixed machine cycle. Virtually all digital design today is based on asynchronous approach. The total system is designed as the composition of one or more subsystems where each subsystem is a clocked finite state machine; the subsystem changes from one state to the next on the edges of a regular clock. The state is held in a set of flip-flops (registers), and combinatorial logic is used to derive the new state and outputs from the old state and inputs. The new state is copied through the flip-flops on every rising edge of the clock signal. Special techniques are required whenever a signal crosses into the domain of a particular clock (either from outside the system or from the domain of a different clock within the same system), but otherwise the system behaves in a discrete and deterministic way provided a few rules are followed; these rules include managing the delays of the combinatorial logic so that the flip-flop set up and hold times are met under all conditions.

Asynchronous design does not follow this methodology; in general there is no clock to govern the timing of state changes. Subsystems exchange information at mutually negotiated times with no external timing regulation.

Clock limitations

Though synchronous design has enabled great strides to be taken in the design and performance of computers, there is evidence that it is beginning to hit some fundamental limitations. A circuit can only operate synchronously if all parts of it see the clock at the same time, at least to a reasonable approximation. However clocks are electrical signals, and when they propagate down wires they are subject to the same delays as other signals. If the delay to particular part of the circuit takes a significant part of a clock cycle-time, that part of the circuit cannot be viewed as being in step with other parts.

For some time now it has been difficult to sustain the synchronous framework from chip to chip at maximum clock rates. On-chip phase-locked loops help compensate for chip-to-chip tolerances, but above about 50MHz even this is not enough. Building the complete CPU on a single chip avoids inter-chip skew, as the highest clock rates are only used for processor-MMU-cache transactions. However, even on a single chip, clock skew is becoming a problem. High-performance processors must dedicate increasing proportions of their silicon area to the clock drivers to achieve acceptable skew, and clearly there is a limit to how much further this proportion can increase. Electrical signals travel on chips at a fraction of the speed of light; as the tracks get thinner, the chips get bigger and the clocks get faster, the skew problem gets worse. Perhaps the clock could be injected optically to avoid the wire delays, but the signals which are issued as a result of the clock still have to propagate along wires in time for the next pulse, so a similar problem remains.

Even more urgent than the physical limitation of clock distribution is the problem of heat. CMOS is a good technology for low power as gates only dissipate energy when they are switching. Normally this should correspond to the gate doing useful work, but unfortunately in a synchronous circuit this is not always the case. Many gates switch because they are connected to the clock, not because they have new inputs to process. The biggest gate of all is the clock driver, and it must switch all the time to provide the timing reference even if only a small part of the chip has anything useful to do. Often it will switch when none of the chip has anything to do, because stopping and starting a high-speed clock is not easy.

Early CMOS devices were very low power, but as process rules have shrunk CMOS has become faster and denser, and today's high-performance CMOS processors can dissipate 20 or 30 watts. Furthermore there is evidence that the trend towards higher power will continue. Process rules have at least another order of magnitude to shrink, leading directly to two orders of magnitude increase in dissipation for a maximum performance chip. Whilst a reduction in the power supply voltage helps reduce the dissipation (by a factor of 3 for 3 Volt operation and a factor of 6 for 2 Volt operation, relative to a 5 Volt norm in both cases), the end result is still a chip with an increasing thermal problem. Processors which dissipate several hundred watts are clearly no use in battery powered equipment, and even on the desktop they impose difficulties because they require water cooling or similar costly heat-removal technology.

As feature sizes reduce and chips encompass more functionality it is likely that the average proportion of the chip which is doing something useful at any time will shrink. Therefore the global clock is becoming increasingly inefficient.

3.3.2 Basic Concepts

There are a few key concepts fundamental to the understanding of asynchronous circuits:

the timing models used, the mode of operation and the signaling conventions.

3.3.2.1 Timing model

Asynchronous circuits are classified according to their behaviour with respect to circuit delays. If a circuit functions correctly irrespective of the delays in the logic gates and the delays in the wiring it is known as *delay-insensitive*. A restricted form of this circuit known as *speed independent* allows arbitrary delays in logic elements but assumes zero delays in the interconnect (i.e. all interconnect wires are equi-potential). Finally, if the circuit only functions when the delays are below some predefined limit the circuit is known as *bounded delay*.

3.3.2.2 Mode

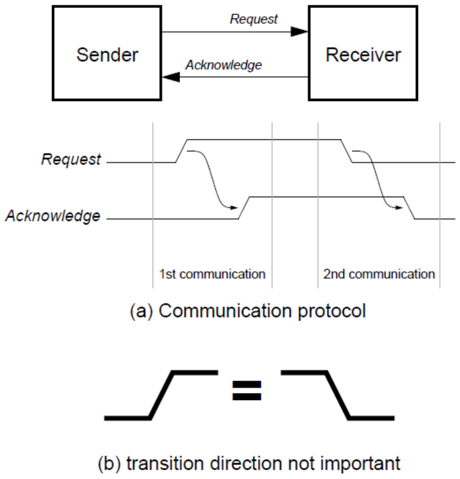
Asynchronous circuits can operate in one of two modes. The first is called *fundamental* mode and assumes no further input changes can be applied until all outputs have settled in response to a previous input. The second, *input/output* mode, allows

3.3.2.3 Asynchronous signaling conventions

A communication between two elements in an asynchronous system can be considered as having two or four phases of operation and a single bit of information can be conveyed on either a single wire or a pair or wires (known as dual-rail encoding).

Two-phase

In a two-phase communication the information is transmitted by a single transition or change in voltage level on a wire. **Figure 4.1(a)** shows an example of two-phase communication.



(a) Communication protocol

(b) transition direction not important

Figure 3.3.1: Two-phase communication protocol

The sender initiates the communication by making a single transition on the request wire; the receiver responds by making a single transition on the acknowledge wire completing the two phases of the communication. The electrical level of the wires contains no information, only a transition is important and rising or falling transitions are equivalent (see figure 4.1(b)) There is no intermediate recovery stage, so that if the first communication resulted in a transition from Low to High the new communication starts with a transition High to Low (figure 4.1(a), 2nd communication).

Four-phase

With four-phase communication two phases are active communication while the other two permit recovery to a predefined state. Figure 4.2 shows an example of four-phase communication; in this example all wires are initialized to a logical Low level.

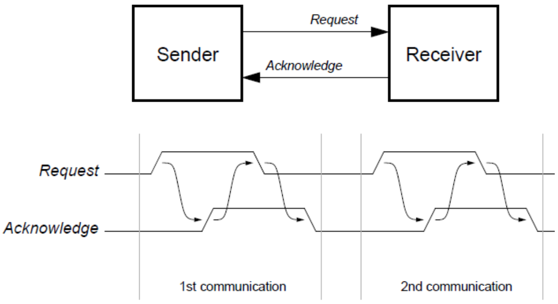


Figure 3.3.2: Four Phase Communication protocol

Four-phase

With four-phase communication two phases are active communication while the other two permit recovery to a predefined state. Figure 4.2 shows an example of four-phase communication; in this example all wires are initialized to a logical Low level. The communication is initiated by the sender changing the request wire to a High level toindicate that it is active. The receiver responds by changing the acknowledge wire to a High level also. The sender observes this change, indicating that the communication has been successful, and then changes the request wire back to Low to indicate it is no longer active.

The receiver completes the fourth phase of the operation by changing the acknowledge wire back to a Low level to indicate that it too has become inactive. After completing the four phases of a single communication, the voltage levels on the wires have returned to their initial value

Single-rail encoding

A single-rail circuit encodes information in a conventional level encoded manner. One wire is required for each bit of information. If the information is a data value, then a typical encoding would use a High (*Vdd*) level to correspond to a logic '1' and a Low level (*Vss*) to
represent a logic '0'.

Dual-rail encoding

A dual-rail circuit requires two wires to encode every bit of information. Of the two wires, one represents a logic '0' and the other represents a logic '1'. In any communication an event occurs on either the logic '0' wire or the logic '1' wire. There cannot be an event on both wires during any single communication (a value cannot be '0' and '1' at the same time in a digital system). Similarly, in every communication there is always an event on one of the two wires of each bit (a value has to be '0' or '1'). It is therefore possible to determine when the entire data word is valid because an event has been detected on one of the dual rails of every bit in the data word. Thus timing information is implicit with the data to indicate its validity. The event that is transmitted on one of the dual rails can either be two phase or four-phase.

There are various combinations of two-/four-phase and single-/dual-rail protocols that can be used. Four-phase, dual-rail is popular for delay-insensitive asynchronous design styles. The research described in this thesis employs a combination of styles. The control circuitry is predominately two-phase, single-rail, although four-phase is used where it is more efficient to do so. Dual-rail is also used but only in a few specialized applications. The datapath part of the design uses standard single-rail logic to implement the functional units.

Overall the design adheres to the bounded-delay timing model (although some parts may be considered delay-insensitive) and its pipeline stages operate in fundamental mode.

3.3.3 Benefits of Asynchronous Control

Two major assumptions guide the design of today’s logic; all signals are binary, and time is discrete. Both of these assumptions are made in order to simplify logic design. By assuming binary values on signals, simple Boolean logic can be used to describe and manipulate logic constructs. By assuming time is discrete, hazards and feedback can largely be ignored. However, as with many simplifying assumptions, a system that can operate without these assumptions has the potential to generate better results.

Asynchronous circuits keep the assumption that signals are binary, but remove the assumption that time is discrete.

Clockless or asynchronous control design is receiving renewed attention, due to its potential benefits of modularity, low power, low electromagnetic interference and average-case performance.

This has several possible benefits:

- **No clock skew -** Clock skew is the difference in arrival times of the clock signal at different parts of the circuit. Since asynchronous circuits by definition have no globally distributed clock, there is no need to worry about clock skew. In contrast, synchronous systems often slow down their circuits to accommodate the skew. As feature sizes decrease, clock skew becomes a much greater concern.
- **Lower power -** Standard synchronous circuits have to toggle clock lines, and possibly pre-charge and discharge signals, in portions of a circuit unused in the current computation. For example, even though a floating point unit on a processor might not be used in a given instruction stream, the unit still must be operated by the clock. Although asynchronous circuits often require more transitions on the computation path than synchronous circuits, they generally have transitions only in areas involved in the current computation. Note that there are some techniques in synchronous design that addresses this issue as well.

- **Average-case instead of worst-case performance -** Synchronous circuits must wait until all possible computations have completed before latching the results, yielding worst-case performance. Many asynchronous systems sense when a computation has completed, allowing them to exhibit average-case performance. For circuits such as ripple-carry adders where the worst-case delay is significantly worse than the average-case delay, this can result in a substantial savings.

- **Easing of global timing issues**: In systems such as a synchronous microprocessor, the system clock, and thus system performance, is dictated by the slowest (critical) path. Thus, most portions of a circuit must be carefully optimized to achieve the highest clock rate, including rarely used portions of the system. Since many asynchronous systems operate at the speed of the circuit path currently in operation, rarely used portions of the circuit can be left un-optimized without adversely affecting system performance.

- **Better technology migration potential -** Integrated circuits will often be implemented in several different technologies during their lifetime. Early systems may be implemented with gate arrays, while later production runs may migrate to semi-custom or custom ICs. Greater performance for synchronous systems can often only be achieved by migrating all system components to a new technology, since again the overall system performance is based on the longest path. In many asynchronous systems, migration of only the more critical system components can improve system performance on average, since performance is dependent on only the currently active path. Also, since many asynchronous systems sense computation completion, components with different delays may often be substituted into a system without altering other elements or structures.

- **Automatic adaptation to physical properties -** The delay through a circuit can change with variations in fabrication, temperature, and power-supply voltage. Synchronous circuits must assume that the worst possible combination of factors is present and clock the system accordingly. Many asynchronous circuits sense computation completion, and will run as quickly as the current physical properties allow.
- **Robust mutual exclusion and external input handling -** Elements that guarantee correct mutual exclusion of independent signals and synchronization of external signals to a clock are subject to meta-Stability. A metastable state is an unstable equilibrium state, such as a pair of cross-coupled CMOS inverters at 2.5V, which a system can remain in for an unbounded amount of time [2]. Synchronous circuits require all elements to exhibit bounded response time. Thus, there is some chance that mutual exclusion circuits will fail in a synchronous system. Most asynchronous systems can wait an arbitrarily long time for such an element to complete, allowing robust mutual exclusion. Also, since there is no clock with which signals must be synchronized, asynchronous circuits more gracefully accommodate inputs from the outside world, which are by nature asynchronous.

3.3.3.1 Limitations of Asynchronous Controllers

With all of the potential advantages of asynchronous circuits, one might wonder why synchronous systems predominate. The reason is that asynchronous circuits have several difficulties as well:

- Asynchronous circuits are more difficult to design in an ad hoc fashion than synchronous circuits. In a synchronous system, a designer can simply define the combinational logic necessary to compute the given function, and surround it with latches.

- By setting the clock rate to a long enough period, all worries about hazards (undesired signal transitions) and the dynamic state of the circuit are removed. In contrast, designers of asynchronous systems must pay a great deal of attention to the dynamic state of the circuit. Hazards must also be removed from the circuit, or not introduced in the first place, to avoid incorrect results.
- The ordering of operations, which was fixed by the placement of latches in a synchronous system, must be carefully ensured by the asynchronous control logic. For complex systems, these issues become too difficult to handle by hand. Unfortunately, asynchronous circuits in general cannot leverage off of existing CAD tools and implementation alternatives for synchronous systems.
- For example, some asynchronous methodologies allow only algebraic manipulations (associative, commutative, and De-Morgan's Law) for logic decomposition, and many do not even allow these.
- Placement, routing, partitioning, logic synthesis, and most other CAD tools either need modifications for asynchronous circuits, or are not applicable at all.
- Finally, even though most of the advantages of asynchronous circuits are towards higher performance, it isn't clear that asynchronous circuits are actually any faster in practice.
- Asynchronous circuits generally require extra time due to their signaling policies, thus increasing average-case delay.

At the end of this this unit; students should be able to;

- Explain the properties of asynchronous Controllers
- Describe hardwired Control
- State the methodologies for Asynchronous Communications
- Understand the Datapaths and Data Transfer modalities of Clockless Controls

3.3.4 Asynchronous Communication

Concurrent and distributed systems use communication as a means to exchange information. Communication can be of two kinds: synchronous and asynchronous. A communication is synchronous when sending and receiving information between a sender and a receiver are simultaneous events. Microcontrollers have the ability to communication asynchronously and synchronously. With Synchronous communication, there is a wire between two communicating agents carrying the clock pulse so both microcontrollers can communicate using the same pulse. With asynchronous communication, there is no wire between the two microcontrollers, so each microcontroller is essentially blind to the pulse rate. Each microcontroller is told, using a baud rate, what speed to execute the communication.

That means **the two devices do not share a dedicated clock signal** (a unique clock exists on each device). Each device must setup ahead of time a matching bit rate and how many bits to expect in a given transaction.

3.3.5 Asynchronous Transmission

In asynchronous transmission, data moves in a half-paired approach, 1 byte or 1 character at a time. It sends the data in a constant current of bytes. The size of a character transmitted is 8 bits, with a parity bit added both at the beginning and at the end, making it a total of 10 bits. It doesn’t need a clock for integration—rather, it utilizes the parity bits to tell the receiver how to translate the data. It is straightforward, quick, cost-effective, and doesnot need two-way communication to function.

Characteristics of Asynchronous Communication

- Each character is headed by a beginning bit and concluded with one or more end bits.

- There may be gaps or spaces in between characters.

Examples of Asynchronous Communication

- Emails
- Forums
- Letters
- Radios
- Televisions

3.3.6 Synchronous vs. Asynchronous Transmission

1. In synchronous transmission data is transmitted in the form of chunks, while in asynchronous transmission data is transmitted one byte at a time.
2. Synchronous transmission needs a clock signal between the source and target to let the target know of the new byte. In comparison, with asynchronous transmission, a clock signal is not needed because of the parity bits that are attached to the data being transmitted, which serves as a start indicator of the new byte.
3. The data transfer rate of synchronous transmission is faster since it transmits in chunks of data, compared to asynchronous transmission which transmits one byte at a time.
4. Asynchronous transmission is straightforward and cost-effective, while synchronous transmission is complicated and relatively pricey.
5. Synchronous transmission is systematic and necessitates lower overhead figures compared to asynchronous transmission.

Both synchronous and asynchronous transmission have their benefits and limitations. Asynchronous transmission is used for sending a small amount of data while the synchronous transmission is used for sending bulk amounts of data. Thus, we can say that both synchronous and asynchronous transmission are essential for the overall process of data transmission.

3.3.7 Emerging application areas

Beyond more classical design targets, a number of novel application areas have recently emerged where asynchronous design is poised to make an impact.

- **Large-scale heterogenous system integration.** In multi- and many-core processors and systems-onchip (SoC's), some level of asynchrony is inevitable in the integration of heterogeneous components.

  Typically, there are several distinct timing domains, which are glued together using an asynchronous communication fabric. There has been much recent work on asynchronous and mixed synchronous asynchronous systems

- **Ultra-low-energy systems and energy harvesting**.
  Asynchronous design is also playing a crucial role in the design of systems that operate in regimes where energy availability is extremely limited. In one application, Such fine-grain adaptation, in which the datapath latency can vary subtly for each input sample, is not possible in a fixed-rate synchronous design. In a recent in-depth case study by Chang et al., focusing on ultra-low-energy 8051 microcontroller cores with voltage scaling, it was shown that under extreme process, voltage, and temperature (PVT) variations, a synchronous core requires its delay margins to be increased by a factor of 12_, while a comparable asynchronous core can operate at actual speed.

  Continuous-time digital signal processors (CTDSP's).
  Another intriguing direction is the development of continuous-time digital signal processors, where input samples are generated at irregular rates by a level-crossing analog-to-digital converter, depending on the actual rate of change of the input's waveform.

  An early specialized approach, using finel discretized sampling, demonstrated a 10_ power reduction

Alternative computing paradigms.

Finally, there is increasing interest in asynchronous circuits as the organizing backbone of systems based on emerging computing technologies, such as cellular nano-array and nano magnetics, where highly-robust asynchronous approaches are crucial to mitigating timing irregularities.

3.3.8 Asynchronous Datapaths and Data Transfer

The internal operations in an individual unit of a digital system are synchronized using clock pulse. It means clock pulse is given to all registers within a unit. And all data transfer among internal registers occurs simultaneously during the occurrence of the clock pulse. Now, suppose any two units of a digital system are designed independently, such as CPU and I/O interface. If the registers in the I/O interface share a common clock with CPU registers, then transfer between the two units is said to be synchronous. But in most cases, the internal timing in each unit is independent of each other, so each uses its private clock for its internal registers. In this case, the two units are said to be asynchronous to each other, and if data transfer occurs between them, this data transfer is called **Asynchronous Data Transfer**. In other words, the two units are said to be asynchronous to each other. CPU and I/O device must coordinate for data transfers.

But, the Asynchronous Data Transfer between two independent units requires that control signals be transmitted between the communicating units so that the time can be indicated at which they send data.

These two methods can achieve this asynchronous way of data transfer:

- **Strobe Control:** This is one way of transfer i.e. by means of strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.
- **Handshaking:** This method is used to accompany each data item being transferred with a control signal that indicates the presence of data in the

bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data.

The strobe pulse and handshaking method of asynchronous data transfer is not restricted to I/O transfer. They are used extensively on numerous occasions requiring the transfer of data between two independent units. So, here we consider the transmitting unit as a source and receiving unit as a destination.

Strobe control method of data transfer uses a single control signal for each transfer. The strobe may be activated by either the source unit or the destination unit. This control line is also known as a strobe, and it may be achieved either by source or destination, depending on which initiate the transfer.

□ Source Initiated Strobe

□ Destination Initiated Strobe

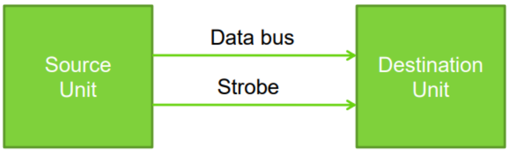


Figure 3.3.3: Strobe control method

SOURCE INITIATED STROBE: The data bus carries the binary information from source unit to the destination unit as shown below.

The strobe is a single line that informs the destination unit when a valid data word is available in the bus.

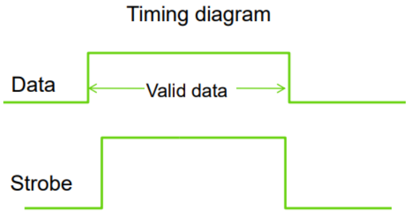


Figure 3.3.4: Source Initiated Strobe

- The source unit first places the data on the bus.
- After a brief delay to ensure that the data settle to a steady value, the source activities the strobe pulse.
- The information of the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data.
- The source removes the data from the bus for a brief period of time after it disables its strobe pulse.

DESTINATION INITIATED STROBE

- First, the destination unit activates the strobe pulse, informing the source to provide the data.
- The source unit responds by placing the requested binary information on the unit to accept it.
- The data must be valid and remain in the bus long enough for the destination unit to accept it.
- The falling edge of the strobe pulse can be used again to trigger a destination register.
- The destination unit then disables the strobe.
- The source removes the data from the bus after a predetermined time interval.

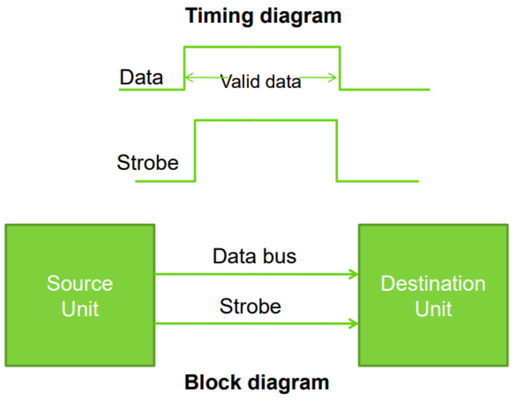


Figure 3.3.5: Destination Initiated Strobe

3.3.9 HANDSHAKING

The strobe method has the disadvantage that the source unit that initiates the transfer has no way of knowing whether the destination has received the data that was placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has placed data on the bus.

- In case of source initiated data transfer under strobe control method, the source unit has no way of knowing whether destination unit has received the data or not.
- Similarly, destination initiated transfer has no method of knowing whether the source unit has placed the data on the data bus.
- Handshaking mechanism solves this problem by introducing a second control signal that provides a reply to the unit that initiate the transfer.

  There are two control lines in handshaking technique:
  - Source to destination unit
  - Destination to source unit

3.3.9.1 SOURCE INITIATED TRANSFER

- Handshaking signals are used to synchronize the bus activities.
- The two handshaking lines are data valid, which is generated by the source unit, and data accepted, generated by the destination unit.
- The timing diagram shows exchange of signals between two units.

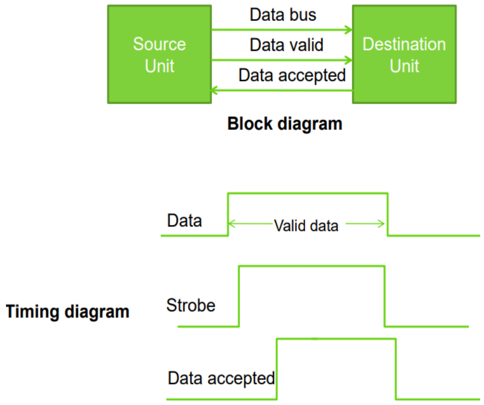


Figure 3.3.6: Source Initiated Transfer

3.3.9.2 SOURCE INITIATED TRANSFER USING HANDSHAKING

The sequence of events:

□ The source unit initiates the transfer by placing the data on the bus and enabling its data valid signal.

□ The data accepted signals is activated by the destination unit after it accepts the data from the bus.

□ The source unit then disables its data valid signal, which invalidates the data on the bus.

□ The destination unit the disables its data accepted signal and the system goes into its initial state.

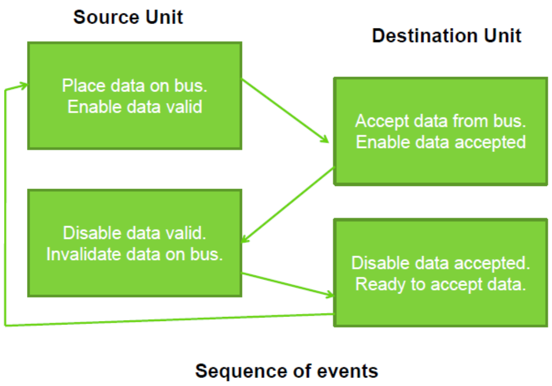


Figure 3.3.7: Source IT using Handshake

3.3.9.3 DESTINATION INITIATED TRANSFER USING HANDSHAKING

- In this case the name of the signal generated by the destination unit is ready for data.
- The source unit does not place the data on the bus until it receives the ready for data signal from the destination unit.
- The handshaking procedure follows the same pattern as in source initiated case. The sequence of events in both the cases is almost same except the ready for signal has been converted from data accepted in case of source initiated.

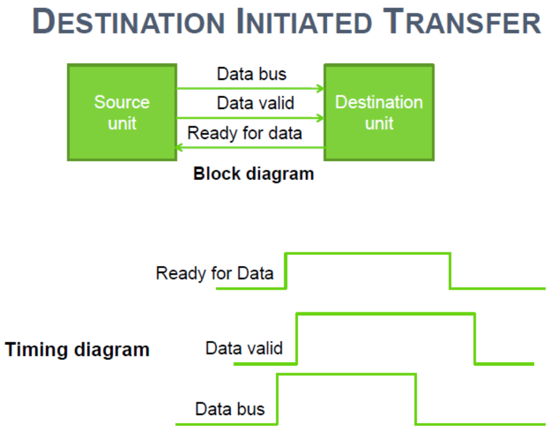


Figure 3.3.7: Destination IT

Advantages of Asynchronous Data Transfer

Asynchronous Data Transfer in computer organization has the following advantages, such as:

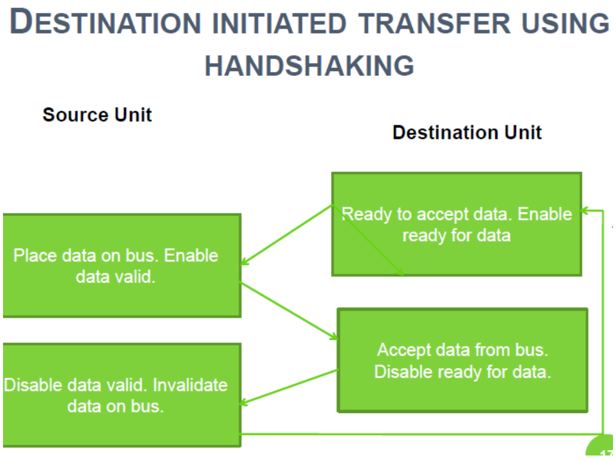

DESTINATION INITIATED TRANSFER USING HANDSHAKING
Source Unit
Destination Unit
Ready to accept data. Enable ready for data
Place data on bus. Enable data valid.
Accept data from bus. Disable ready for data.
Disable data valid. Invalidate data on bus.

Figure 3.3.7: Destination IT using Handshake

- It is more flexible, and devices can exchange information at their own pace. In addition, individual data characters can complete themselves so that even if one packet is corrupted, its predecessors and successors will not be affected.
- It does not require complex processes by the receiving device. Furthermore, it means that inconsistency in data transfer does not result in a big crisis since the device can keep up with the data stream. It also

makes asynchronous transfers suitable for applications where character data is generated irregularly.

Disadvantages of Asynchronous Data Transfer

There are also some disadvantages of using asynchronous data for transfer in computer organization, such as:

- The success of these transmissions depends on the start bits and their recognition. Unfortunately, this can be easily susceptible to line interference, causing these bits to be corrupted or distorted.
- A large portion of the transmitted data is used to control and identify header bits and thus carries no helpful information related to the transmitted data. This invariably means that more data packets need to be sent.

4.0 CONCLUSION

In this module, we have discussed in detail the implementation of control units. We started with an implementation of the hardwired, The logic micro-operation implementation has also been discussed. Thus, leading to a logical construction of a simple arithmetic– logic –shift unit. The unit revolves around the basic ALU with the help of the units that are constructed for the implementation of micro-operations. We have also we discussed the arithmetic processors and the organization of control units. Various types of control units such as Hardwired, Wilkes and micro-programmed control units are also discussed. The key to such control units are micro-instruction, which are briefly (that is types and formats) described in this unit. Finally the function of a micro-programmed unit, that is, micro-programmed execution, has also been discussed. The control unit is the key for the optimized performance of a computer. More detail study can be done by going through suggested readings.

5.0 SUMMARY

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.

6.0 TUTOR MARKED ASSIGNMENT

1. What is control unit?
2. Explain functional requirement of control unit.
3. What are the inputs to control unit?
4. Describe different types of control unit with diagram.

7.0 REFERENCES/FURTHER READING

William, S. (2010). Computer organization and architecture: designing for performance.

Astha Singh. "Computer Organization - Control Unit and design". GeeksforGeeks. Retrieved 25 May 2019.

Asanovic, Krste (2017). The RISC V Instruction Set Manual (PDF) (2.2 ed.). Berkeley: RISC-V Foundation.

Power ISA(tm) (3.0B ed.). Austin: IBM. 2017. Retrieved 26 December 2019.

Leighton, Luke. "Libre RISC-V M-Class". Crowd Supply. Retrieved 16 January 2020.

MODULE FOUR

Fault Tolerance Computing

1.0 INTRODUCTION

Technology scaling allows realization of more and more complex system on a single chip. This high level of integration leads to increased current and power densities and causes early device and interconnect wear-out. In addition, there are failures not caused by wear-out nor escaped manufacturing defects, but due to increased susceptibility of transistors to high energy particles from atmosphere or from within the packaging. Devices operating at reduced supply voltages are more prone to charge related phenomenon caused by high-energy particle strikes referred to as Single Event Effect (SEE). They experience particle-induced voltage transients called Single Event Transient (SET) or particle-induced bit-flips in memory elements also known as Single Event Upset (SEU). Selecting the ideal trade-off between reliability and cost associated with a fault tolerant architecture generally involves an extensive design space exploration. Employing state-of-the-art reliability estimation methods makes this exploration un-scalable with the design complexity.

Fault Tolerance in software is a phenomenon where the software is capable of fixing itself or continuing the normal operations in the occurrence of any glitches or error in the system, provided that full coverage on the functionality is maintained as specified in the required documentation. The reasons behind these faults in the software system can be a fault from within, from other integrated systems, from the downstream application, or from any other external aspects like the system hardware, network, etc. This is one of the factors based on which software is estimated to be a quality one or not. Hence it is important that every software program consists of fault tolerance. *Fault-tolerant computing* is the art and science of building computing systems that continue to operate satisfactorily in the presence of faults. A fault-tolerant system may be able to tolerate one or more fault-types including -- i) transient, intermittent or permanent hardware faults, ii) software and hardware design errors, iii) operator errors, or iv) externally induced upsets or physical damage. An extensive methodology has been developed in this field over the past thirty years, and a number of fault-tolerant machines have been developed - most dealing with random hardware faults, while a smaller number deal with software, design and operator faults to varying degrees.

2.0 OBJECTIVES

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

- Explain fault tolerant in computing.

- Explain redundancy and identify Hardware and Software Fault Tolerant Issues
- Explain the Relationship Between Security and Fault Tolerance
- Identify different Fault Tolerance Architectures

3.0.1 FAULT TOLERANT COMPUTING

3.0.1.1 What is Fault Tolerance

Fault Tolerance has been part of the computing community for quite a long time, to clarify the building of our understanding of fault tolerance, then we should know that fault tolerance is the art and science of building computing systems that continue to operate satisfactorily in the presence of faults. An operating system that offers a solid definition for faults cannot be disrupted by a single point of failure. It ensures business continuity and the high availability of crucial applications and systems regardless of any failures.

Fault tolerance and dependable systems research covers a wide spectrum of applications ranging across embedded real-time systems, commercial transaction systems, transportation systems, military and space systems to name a few. The supporting research includes system architecture, design techniques, coding theory, testing, validation, proof of correctness, modeling, software reliability, operating systems, parallel processing, and real-time processing. These areas often involve widely diverse core expertise ranging from formal logic, mathematics of stochastic modeling, graph theory, hardware design and software engineering.

Basic Terms of fault Tolerance Computing

Fault tolerance can be built into a system to remove the risk of it having a single point of failure. To do so, the system must have no single component that, if it were to stop working effectively, would result in the entire system failing. Fault tolerance is reliant on aspects like load balancing and failover, which remove the risk of a single point of failure. It will typically be part of the operating system's interface, which enables programmers to check the performance of data throughout a transaction.

Three central terms in fault-tolerant design are fault, error, and failure. There is a cause effect relationship between faults, errors, and failures. Specifically, faults are the cause of errors, and errors are the cause of failures. Often the term failure is used interchangeably with the term malfunction, however, the term failure is rapidly becoming the more commonly accepted one. A *fault* is a physical defect, imperfection, or flaw that occurs within some hardware or software component. Essentially, the definition of a fault, as used in the fault tolerance community, agrees with the definition found in the dictionary. A fault is a blemish, weakness, or shortcoming of a particular hardware or software component. An *error* is the manifestation of a fault. Specifically, an error is a deviation from accuracy or correctness. Finally, if the error results in the system performing one of its functions incorrectly then a system *failure* has occurred. Essentially, a failure is the nonperformance of some action that is due or expected. A failure is also the performance of some function in a subnormal quantity or quality.

The concepts of faults, errors, and failures can be best presented by the use of a three-universe model that is an adaptation of the four-universe models;

- first universe is the *physical universe* in which faults occur. The physical universe contains the semiconductor devices, mechanical elements, displays, printers, power supplies, and other physical entities that make up a system. A fault is a physical defect or alteration of some component within the physical universe.

- The second universe is the *informational universe*. The informational universe is where the error occurs. Errors affect units of information such as data words within a computer or digital voice or image information. An error has occurred when some unit of information becomes incorrect.
- The final universe is the *external* or *user's universe*. The external universe is where the user of a system ultimately sees the effect of faults and errors. The external universe is where failures occur. The failure is any deviation that occurs from the desired or expected behavior of a system. In summary, faults are physical events that occur in the physical universe. Faults can result in errors in the informational universe, and errors can ultimately lead to failures that are witnessed in the external universe of the system.

The cause-effect relationship implied in the three-universe model leads to the definition of two important parameters; fault latency and error latency.

- *Fault latency* is the length of time between the occurrence of a fault and the appearance of an error due to that fault.
- *Error latency* is the length of time between the occurrence of an error and the appearance of the resulting failure. Based on the three-universe model, the total time between the occurrence of a physical fault and the appearance of a failure will be the sum of the fault latency and the error latency.

Characteristics of Faults

Faults could be classified based on the following parameters

a) Causes/Source of Faults

b) Nature of Faults

c) Fault Duration

d) Extent of Faults

e) Value of faults

Sources of faults: Faults can be the result of a variety of things that occur within electronic components, external to the components, or during the component or system design process. Problems at any of several points within the design process can result in faults within the system.

- ***Specification mistakes,*** which include incorrect algorithms, architectures, or hardware and software design specifications.
- ***Implementation mistakes.*** Implementation, as defined here, is the process of transforming hardware and software specifications into the physical hardware and the actual software. The implementation can introduce faults because of poor design, poor component selection, poor construction, or software coding mistakes.
- ***Component defects.*** Manufacturing imperfections, random device defects, and component wear-out are typical examples of component defects. Electronic components simply become defective sometimes. The defect can be the result of bonds breaking within the circuit or corrosion of the metal. Component defects are the most commonly considered cause of faults.
- ***External disturbance;*** for example, radiation, electromagnetic interference, battle damage, operator mistakes, and environmental extremes.

***Nature* of a faults:** specifies the type of fault; for example, whether it is a hardware fault, a software fault, a fault in the analog circuitry, or a fault in the digital circuitry.

Fault D*uration*. The duration specifies the length of time that a fault is active.

- ***Permanent fault,*** that remains in existence indefinitely if no corrective action is taken.

- ***Transient fault,*** which can appear and disappear within a very short period of time.
- ***Intermittent fault*** that appears, disappears, and then reappears repeatedly.

Fault Extent. The extent of a fault specifies whether the fault is localized to a given hardware or software module or globally affects the hardware, the software, or both.

Fault value of a fault can be either determinate or indeterminate. A *determinate fault* is one whose status remains unchanged throughout time unless externally acted upon. An *indeterminate fault* is one whose status at some time, T, may be different from its status at some increment of time greater than or less than T.

Three primary techniques for maintaining a system's normal performance in an environment where faults are of concern; **fault avoidance, fault masking, and fault tolerance**.

- ***Fault avoidance*** is a technique that is used in an attempt to prevent the occurrence of faults. Fault avoidance can include such things as design reviews, component screening, testing, and other quality control methods.
- ***Fault masking*** is any process that prevents faults in a system from introducing errors into the informational structure of that system.
- ***Fault tolerance*** is the ability of a system to continue to perform its tasks after the occurrence of faults. The ultimate goal of fault tolerance is to prevent system failures from occurring. Since failures are directly caused by errors, the terms *fault tolerance* and *error tolerance* are often used interchangeably.

Approaches for Fault Tolerance.

- **Fault masking** is one approach to tolerating faults.
- ***Reconfiguration*** is the process of eliminating a faulty entity from a system and restoring the system to some operational condition or state. If the reconfiguration technique is used then the designer must be concerned with fault detection, fault location, fault containment, and fault recovery.
- ***Fault detection*** is the process of recognizing that a fault has occurred. Fault detection is often required before any recovery procedure can be implemented.
- ***Fault location*** is the process of determining where a fault has occurred so that an appropriate recovery can be implemented.
- ***Fault containment*** is the process of isolating a fault and preventing the effects of that fault from propagating throughout a system. Fault containment is required in all fault-tolerant designs.
- ***Fault recovery*** is the process of remaining operational or regaining operational status via reconfiguration even in the presence of faults.

Goals of Fault Tolerance

Fault tolerance is an attribute that is designed into a system to achieve some design goals such as; dependability, reliability, availability, safety, performability, maintainability, and testability; fault tolerance is one stem attribute capable of fulfilling such requirements.

Dependability. The term *dependability* is used to encapsulate the concepts of reliability, availability, safety, maintainability, performability, and testability. Dependability is simply the quality of service provided by a particular system. Reliability, availability, safety, maintainability, performability, and testability, are examples of measures used to quantify the dependability of a system.

Reliability. The *reliability* of a system is a function of time, *R(t)*, defined as the conditional probability that the system performs correctly throughout the interval of time, [*t*0,*t*], given that the system was performing correctly at time *t*0. In other words, the reliability is the probability that the system operates correctly throughout a complete interval of time. The reliability is a conditional probability in that it depends on the system being operational at the beginning of the chosen time interval. The *unreliability* of a system is a function of time, *F(t)*, defined as the conditional probability that a system begins to perform incorrectly during the interval of time, [*t*0,*t*], given that the system was performing correctly at time *t*0. The unreliability is often referred to as the probability of failure.

Reliability is most often used to characterize systems in which even momentary periods of incorrect performance are unacceptable, or it is impossible to repair the system. If repair is impossible, such as in many space applications, the time intervals being considered can be extremely long, perhaps as many as ten years. In other applications, such as aircraft flight control, the time intervals of concern may be no more than several hours, but the probability of working correctly throughout that interval may be 0.9999999 or higher. It is a common convention when reporting reliability numbers to use 0.9*i* to represent the fraction that has *i* nines to the right of the decimal point. For example, 0.9999999 is written as 0.97.

Availability. *Availability* is a function of time, *A(t)*, defined as the probability that a system is operating correctly and is available to perform its functions at the instant of time, *t*. Availability differs from reliability in that reliability involves an interval of time, while availability is taken at an instant of time. A system can be highly available yet experience frequent periods of inoperability as long as the length of each period is extremely short. In other words, the availability of a system depends not only on how frequently it becomes inoperable but also how quickly it can be repaired. Examples of high-availability applications include time-shared computing systems and certain transactions processing applications, such as airline reservation systems.

Safety. *Safety* is the probability, *S(t)*, that a system will either perform its functions correctly or will discontinue its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system. Safety is a measure of the failsafe capability of a system; if the system does not operate correctly, it is desired to have the system fail in a safe manner. Safety and reliability differ because reliability is the probability that a system will perform its functions correctly, while safety is the probability that a system will either perform its functions correctly or will discontinue the functions in a manner that causes no harm.

Performability. In many cases, it is possible to design systems that can continue to perform correctly after the occurrence of hardware and software faults, but the level of performance is somehow diminished. The *performability* of a system is a function of time, *P(L,t)*, defined as the probability that the system performance will be at, or above, some level, *L*, at the instant of time, *t* Performability differs from reliability in that reliability is a measure of the likelihood that all of the functions are performed correctly, while performability is a measure of the likelihood that some subset of the functions is performed correctly.

Graceful degradation is an important feature that is closely related to performability. *Graceful degradation* is simply the ability of a system to automatically decrease its level of performance to compensate for hardware and software faults. Fault tolerance can certainly support graceful degradation and performability by providing the ability to eliminate the effects of hardware and software faults from a system, therefore allowing performance at some reduced level.

Maintainability. *Maintainability* is a measure of the ease with which a system can be repaired, once it has failed. In more quantitative terms, maintainability is the probability, *M(t)*, that a failed system will be restored to an operational state within a period of time *t*. The restoration process includes locating the problem, physically repairing the system, and bringing the system back to its operational condition. Many of the techniques that are so vital to the achievement of fault tolerance can be used to detect and locate problems in a system for the purpose of maintenance. Automatic diagnostics can significantly improve the maintainability of a system because a majority of the time used to repair a system is often devoted to determining the source of the problem.

Testability. *Testability* is simply the ability to test for certain attributes within a system. Measures of testability allow one to assess the ease with which certain tests can be performed. Certain tests can be automated and provided as an integral part of the system to improve the testability. Many of the techniques that are so vital to the achievement of fault tolerance can be used to detect and locate problems in a system for the purpose of improving testability. Testability is clearly related to maintainability because of the importance of minimizing the time required to identify and locate specific problems.

3.0.1.2 Fault Tolerant Systems

Fault tolerance is a process that enables an operating system to respond to a failure in hardware or software. This fault-tolerance definition refers to the system's ability to continue operating despite failures or malfunctions. A fault-tolerant system may be able to tolerate one or more fault-types including

- Transient, Intermittent or Permanent Hardware Faults,
- Software and Hardware Design Errors,
- Operator Errors
- Externally Induced Upsets or Physical Damage.

An extensive methodology has been developed in this field over the past thirty years, and a number of fault-tolerant machines have been developed most dealing with random hardware faults, while a smaller number deal with software, design and operator faults to varying degrees. A large amount of supporting research has been reported and efforts to attain software that can tolerate software design faults (programming errors) have made use of static and dynamic redundancy approaches similar to those used for hardware faults. One such approach, *N*-version programming, uses static redundancy in the form of independently written programs (versions) that perform the same functions, and their outputs are voted at special checkpoints. Here, of course, the data being voted may not be exactly the same, and a criterion must be used to identify and reject faulty versions and to determine a consistent value (through inexact voting) that all good versions can use. An alternative dynamic approach is based on the concept of recovery blocks. Programs are partitioned into blocks and acceptance tests are executed after each block. If an acceptance test fails, a redundant code block is executed.

An approach called design diversity combines hardware and software fault-tolerance by implementing a fault-tolerant computer system using different hardware and software in redundant channels. Each channel is designed to provide the same function, and a method is provided to identify if one channel deviates unacceptably from the others. The goal is to tolerate both hardware and software design faults. This is a very expensive technique, but it is used in very critical aircraft control applications.

Major building blocks of a Fault-tolerance System

The key benefit of fault tolerance is to minimize or avoid the risk of systems becoming unavailable due to a component error(s). This is particularly important in critical systems that are relied on to ensure people's safety, such as air traffic control, and systems that protect and secure critical data and high-value transactions The core components to improving fault tolerance include:

Diversity: If a system's main electricity supply fails, potentially due to a storm that causes a power outage or affects a power station, it will not be possible to access alternative electricity sources. In this event, fault tolerance can be sourced through diversity, which provides electricity from sources like backup generators that take over when a main power failure occurs.

- Some diverse fault-tolerance options result in the backup not having the same level of capacity as the primary source. This may, in some cases, require the system to ensure graceful degradation until the primary power source is restored.
- Redundancy
- Fault-tolerant systems use redundancy to remove the single point of failure. The system is equipped with one or more power supply units (PSUs), which do not need to power the system when the primary PSU functions as normal. In the event the primary PSU fails or suffers a fault, it can be removed from service and replaced by a redundant PSU, which takes over system function and performance.
- Alternatively, redundancy can be imposed at a system level, which means an entire alternate computer system is in place in case a failure occurs.

Replication: Replication is a more complex approach to achieving fault tolerance. It involves using multiple identical versions of systems and subsystems and ensuring their functions always provide identical results. If the results are not identical, then a democratic procedure is used to identify the faulty system. Alternatively, a procedure can be used to check for a system that shows a different result, which indicates it is faulty.

- Replication can either take place at the component level, which involves multiple processors running simultaneously, or at the system level, which involves identical computer systems running simultaneously

Basic Characteristics of Fault Tolerant Systems

A fault tolerant system may have one or more of the following characteristics:

- **No Single Point of Failure:** This means if a capacitor, block of software code, a motor, or any single item fails, then the system does not fail. As an example, many hospitals have backup power systems in case the grid power fails, thus keeping critical systems within the hospital operational. Critical systems may have multiple redundant schemes to maintain a high level of fault tolerance and resilience.

- **No Single Point Repair Takes the System Down:** Extending the single point failure idea, effecting a repair of a failed component does not require powering down the system, for example. It also means the system remains online and operational during repair. This may pose challenges for both the design and the maintenance of a system. Hot swappable power supplies is an example of a repair action that keeps the system operating while replacing a faulty power supply.

- **Fault isolation or identification:** The system is able to identify when a fault occurs within the system and does not permit the faulty element to adversely influence to functional capability (i.e. Losing data or making logic errors in a banking system). The faulty elements are identified and isolated. Portions of the system may have the sole purpose of detecting faults, built-in self-test (BIST) is an example.

Fault containment to prevent propagation of failure

- When a failure occurs it may result in damage to other elements within the system, thus creating a second or third fault and system failure.
- For example, if an analog circuit fails it may increase the current across the system damaging logic circuits unable to withstand high current conditions. The idea of fault containment is to avoid or minimize collateral damage caused by a single point failure.

Robustness or Variability Control

- When a system experiences a single point failure, the system changes.
- The change may cause transient or permanent changes affecting how the working elements of the system response and function. Variation occurs, and when a failure occurs there often is an increase in variability. For example, when one of two power supplies fails, the remaining power supply takes on the full load of the power demand. This transition should occur without impacting the performance of the system. The ability to design and manufacture a robust system may involve design for six sigma, design of experiment optimization, and other tools to create a system able to operate when a failure occurs.

Availability of Reversion Mode

- There are many ways a system may alter it performance when a failure occurs, enabling the system to continue to function in some fashion.
- For example, if part of a computer's cooling system fails, the central processor unit (CPU) may reduce its speed or command execution rate, effectively reducing the heat the CPU generates. The fail failure causes a loss of cooling capacity and the CPU adjusts to accommodate and avoids overheating and failing. Other reversion schemes may include a roll back to a prior working state, or a switch to a prior or safe mode software set.

- In some cases, the system may be able to operators with no or only minimal loss of functional capability, or the reversion operation significantly restricts the system operation to a critical few functions.

3.0.1.3 Hardware and Software Fault Tolerant Issues

In everyday language, the terms fault, failure, and error are used interchangeably. In fault-tolerant computing parlance, however, they have distinctive meanings. A fault (or failure) can be either a hardware defect or a software i.e. programming mistake (bug). In contrast, an error is a manifestation of the fault, failure and bug. As an example, consider an adder circuit, with an output line stuck at 1; it always carries the value 1 independently of the values of the input operands. This is a fault, but not (yet) an error. This fault causes an error when the adder is used and the result on that line is supposed to have been a 0, rather than a 1. A similar distinction exists between programming mistakes and execution errors. Consider, for example, a subroutine that is supposed to compute sin(x) but owing to a programming mistake calculates the absolute value of sin(x) instead. This mistake will result in an execution error only if that particular subroutine is used and the correct result is negative.

Both faults and errors can spread through the system. For example, if a chip shorts out power to ground, it may cause nearby chips to fail as well. Errors can spread because the output of one unit is used as input by other units. To return to our previous examples, the erroneous results of either the faulty adder or the sin(x) subroutine can be fed into further calculations, thus propagating the error.

To limit such contagion, designers incorporate containment zones into systems. These are barriers that reduce the chance that a fault or error in one zone will propagate to another. For example, a fault-containment zone can be created by ensuring that the maximum possible voltage swings in one zone are insulated from the other zones, and by providing an independent power supply to each zone. In other words, the designer tries to electrically isolate one zone from another. An error-containment zone can be created, as we will see in some detail later on, by using redundant units, programs and voting on their output.

Hardware faults can be classified according to several aspects. Regarding their duration, hardware faults can be classified into permanent, transient, or intermittent. A permanent fault is just that: it reflects the permanent going out of commission of a component. As an example of a permanent fault think of a burned-out light bulb.

A transient fault is one that causes a component to malfunction for some time; it goes away after that time and the functionality of the component is fully restored. As an example, think of a random noise interference during a telephone conversation. Another example is a memory cell with contents that are changed spuriously due to some electromagnetic interference. The cell itself is undamaged: it is just that its contents are wrong for the time being, and overwriting the memory cell will make the fault go away.

An intermittent fault never quite goes away entirely; it oscillates between being quiescent and active. When the fault is quiescent, the component functions normally; when the fault is active, the component malfunctions. An example for an intermittent fault is a loose electrical connection. Another classification of hardware fault is into benign and malicious faults.

A fault that just causes a unit to go dead is called benign. Such faults are the easiest to deal with. Far more insidious are the faults that cause a unit to produce reasonable-looking, but incorrect, output, or that make a component “act maliciously” and send differently valued outputs to different receivers. Think of an altitude sensor in an airplane that reports a 1000-foot altitude to one unit and an 8000-foot altitude to another unit. These are called malicious (or Byzantine) faults.

3.0.1.4 Fault Tolerance VS High Availability

Why is it that we see industry-standard servers advertising five 9s of availability while Nonstop servers acknowledge four 9s? Are these high-availability industry-standard servers really ten times more reliable than fault-tolerant NonStop servers? Of course not.

To understand this marketing discrepancy, let’s take a look at the factors which differentiate fault-tolerant systems from high-availability systems.

To start with, there is no reason to assume that a single NonStop processor is any more or less reliable than an industry-standard processor. In fact, a reasonable assumption is that a processor will be up about 99.5% of the time (that is, it will have almost three 9s availability) whether it be a NonStop processor or an industry-standard processor.

So how do we get four or five 9s out of components that offer less than three 9s of availability? Through redundancy, of course. NonStop servers are inherently redundant and are fault tolerant (FT) in that they can survive any single fault. In the high-availability (HA) world, industry-standard servers are configured in clusters of two or more processors that allow for re-configuration around faults. FT systems tolerate faults; HA clusters re-configure around faults.

If you provide a backup, you double your 9s. Thus, in a two-processor configuration, each of which has an availability of .995, you can be dreaming of five 9s of hardware availability. But dreams they are. True, you will have at least one processor up 99.999% of the time; but that does not mean that your system will be available for that proportion of time. This is because most system outages are not caused by hardware failures.

The causes of outages have been studied by many (Standish Group, IEEE Computer, Grey, among others), and they all come up with amazingly similar breakdowns:

-	Hardware	10% – 20%
-	Software	30% – 40 %
-	People	20% – 40%
-	Environment	10% – 20%
-	Planned	20% – 30%

These results are for single processor systems. However, we are considering redundant systems which will suffer a hardware failure only if both systems fail. Given a 10-20% chance that a single system will fail due to a hardware failure, an outage due to a dual hardware failure is only 1% to 4%. Thus, we can pretty much ignore hardware failures as a source of failure in redundant systems. (This is a gross understatement for the new Nonstop Advanced Architecture, which is reaching toward six or seven 9s for hardware availability.)

So, what is left that can be an FT/HA differentiator? Environmental factors (air conditioning, earthquakes, etc.) and people factors (assuming good system management tools) are pretty much independent of the system. Planned downtime is a millstone around everyone’s neck, and much is being done about this across all systems. This leaves software as the differentiator.

Software faults are going to happen, no matter what. In a single system, 30-40% of all single-system outages will be caused by software faults. The resultant availability of a redundant system is going to depend on how software faults are handled. Here is the distinction between fault-tolerant systems and high-availability systems. A fault-tolerant system will automatically recover from a software fault almost instantly (typically in seconds) as failed processes switch over to their synchronized backups. The state of incomplete transactions remains in the backup disk process and processing goes on with virtually no delay. On the other hand, a high-availability (HA) cluster will typically require that the applications be restarted on a surviving system and that in-doubt transactions in process be recovered from the transaction log. Furthermore, users must be switched over before the applications are once again available to the users. This can all take several minutes. In addition, an HA switchover must often be managed manually.

If an FT system and an HA cluster have the same fault rate, but the FT system can recover in 3 seconds and the HA cluster takes 5 minutes (300 seconds) to recover from the same fault, then the HA cluster will be down 100 times as long as the FT system and will have an availability which is two 9s less. That glorious five 9s claim becomes three 9s (as reported in several industry studies), at least so far as software faults are concerned.

So, the secret to high availability is in the recovery time. This is what the Tandem folks worked so hard on for two decades before becoming the Nonstop people. Nobody else has done it. Today, Nonstop servers are the only fault-tolerant systems out-of-the-box in the marketplace, and they hold the high ground for availability.

3.0.1.5 Redundancy

All of fault tolerance is an exercise in exploiting and managing redundancy. Redundancy is the property of having more of a resource than is minimally necessary to do the job at hand. As failures happen, redundancy is exploited to mask or otherwise work around these failures, thus maintaining the desired level of functionality.

There are four forms of redundancy that we will study: hardware, software, information, and time. Hardware faults are usually dealt with by using hardware, information, or time redundancy, whereas software faults are protected against by software redundancy.

Hardware redundancy is provided by incorporating extra hardware into the design to either detect or override the effects of a failed component. For example, instead of having a single processor, we can use two or three processors, each performing the same function. By having two processors, we can detect the failure of a single processor; by having three, we can use the majority output to override the wrong output of a single faulty processor. This is an example of static hardware redundancy, the main objective of which is the immediate masking of a failure. A different form of hardware redundancy is dynamic redundancy, where spare components are activated upon the failure of a currently active component. A combination of static and dynamic redundancy techniques is also possible, leading to hybrid hardware redundancy.

Hardware redundancy can thus range from a simple duplication to complicated structures that switch in spare units when active ones become faulty. These forms of hardware redundancy incur high overheads, and their use is therefore normally reserved for critical systems where such overheads can be justified. In particular, substantial amounts of redundancy are required to protect against malicious faults.

The best-known form of information redundancy is error detection and correction coding. Here, extra bits (called check bits) are added to the original data bits so that an error in the data bits can be detected or even corrected. The resulting error-detecting and error-correcting codes are widely used today in memory units and various storage devices to protect against benign failures. Note that these error codes (like any other form of information redundancy) require extra hardware to process the redundant data (the check bits).

Error-detecting and error-correcting codes are also used to protect data communicated over noisy channels, which are channels that are subject to many transient failures. These channels can be either the communication links among widely separated processors (e.g., the Internet) or among locally connected processors that form a local network. If the code used for data communication is capable of only detecting the faults that have occurred (but not correcting them), we can retransmit as necessary, thus employing time redundancy.

In addition to transient data communication failures due to noise, local and wide-area networks may experience permanent link failures. These failures may disconnect one or more existing communication paths, resulting in a longer communication delay between certain nodes in the network, a lower data bandwidth between certain node pairs, or even a complete disconnection of certain nodes from the rest of the network. Redundant communication links (i.e., hardware redundancy) can alleviate most of these problems.

Computing nodes can also exploit time redundancy through re-execution of the same program on the same hardware. As before, time redundancy is effective mainly against transient faults. Because the majority of hardware faults are transient, it is unlikely that the separate executions will experience the same fault.

Time redundancy can thus be used to detect transient faults in situations in which such faults may otherwise go undetected. Time redundancy can also be used when other means for detecting errors are in place and the system is capable of recovering from the effects of the fault and repeating the computation. Compared with the other forms of redundancy, time redundancy has much lower hardware and software overhead but incurs a high-performance penalty.

Software redundancy is used mainly against software failures. It is a reasonable guess that every large piece of software that has ever been produced has contained faults (bugs). Dealing with such faults can be expensive: one way is to independently produce two or more versions of that software (preferably by disjoint teams of programmers) in the hope that the different versions will not fail on the same input. The secondary version(s) can be based on simpler and less accurate algorithms (and, consequently, less likely to have faults) to be used only upon the failure of the primary software to produce acceptable results. Just as for hardware redundancy, the multiple versions of the program can be executed either concurrently (requiring redundant hardware as well) or sequentially (requiring extra time, i.e., time redundancy) upon a failure detection.

Techniques of Redundancy

The concept of redundancy implies the addition of information, resources, or time beyond what is needed for normal system operation. The redundancy can take one of several forms, including hardware redundancy, software redundancy, information redundancy, and time redundancy. The use of redundancy can provide additional capabilities within a system. In fact, if fault tolerance or fault detection is required then some form of redundancy is also required. But, it must be understood that redundancy can have a very important impact on a system in the areas of performance, size, weight, power consumption, reliability, and others

Hardware Redundancy

The physical replication of hardware is perhaps the most common form of redundancy used in systems. As semiconductor components have become smaller and less expensive, the concept of hardware redundancy has become more common and more practical. The costs of replicating hardware within a system are decreasing simply because the costs of hardware are decreasing.

There are three basic forms of hardware redundancy. First, *passive* techniques use the concept of fault masking to hide the occurrence of faults and prevent the faults from resulting in errors. Passive approaches are designed to achieve fault tolerance without requiring any action on the part of the system or an operator. Passive techniques, in their most basic form, do not provide for the detection of faults but simply mask the faults.

The second form of hardware redundancy is the ***active*** **approach**, which is sometimes called the ***dynamic*** method. Active methods achieve fault tolerance by detecting the existence of faults and performing some action to remove the faulty hardware from the system. In other words, active techniques require that the system perform reconfiguration to tolerate faults. Active hardware redundancy uses fault detection, fault location, and fault recovery in an attempt to achieve fault tolerance. The final form of hardware redundancy is the ***hybrid*** approach. Hybrid techniques combine the attractive features of both the passive and active approaches. Fault masking is used in hybrid systems to prevent erroneous results from being generated. Fault detection, fault location, and fault recovery are also used in the hybrid approaches to improve fault tolerance by removing faulty hardware and replacing it with spares. Providing spares is one form of providing redundancy in a system.

Hybrid methods are most often used in the critical-computation applications where fault masking is required to prevent momentary errors, and high reliability must be achieved. Hybrid hardware redundancy is usually a very expensive form of redundancy to implement.

Passive Hardware Redundancy. Passive hardware redundancy relies upon voting mechanisms to mask the occurrence of faults. Most passive approaches are developed around the concept of majority voting. As previously mentioned, the passive approaches achieve fault tolerance without the need for fault detection or system reconfiguration; the passive designs inherently tolerate the faults. The most common form of passive hardware redundancy is called *triple modular redundancy* (TMR). The basic concept of TMR is to triplicate the hardware and perform a majority vote to determine the output of the system. If one of the modules becomes faulty, the two remaining fault free modules mask the results of the faulty module when the majority vote is performed. The basic concept of TMR is illustrated in Figure 3.1. In typical applications, the replicated modules are processors, memories, or any hardware entity. A simple example of TMR is shown in Figure 4.1 where data from three independent processors is voted upon before being written to memory. The majority vote provides a mechanism for ensuring that each memory contains the correct data, even if a single faulty processor exists. A similar voting process is provided at the output of the memories, so that a single memory failure will not corrupt the data provided to any one processor. Note that in Figure 4.2 there are three separate voters so that the failure of a single voter cannot corrupt more than one memory or more than one processor.

The primary challenge with TMR is obviously the voter; if the voter fails, the complete system fails. In other words, the reliability of the simplest form of TMR, as shown in Figure 4.1, can be no better than the reliability of the voter.

Any single component within a system whose failure will lead to a failure of the system is called a single-point-of-failure. Several techniques can be used to overcome the effects of voter failure. One approach is to triplicate the voters and provide three independent outputs, as illustrated in Figure 4.2. In Figure 4.2, each of three memories receives data from a voter which has received its inputs from the three separate processors. If one processor fails, each memory will continue to receive a correct value because its voter will correct the corrupted value. A TMR system with triplicated voters is commonly called a *restoring organ* because the configuration will produce three correct outputs even if one input is faulty. In essence, the TMR with triplicated voters restores the error-free signal. A generalization of the TMR approach is the N-modular redundancy (NMR) technique. NMR applies the same principle as TMR but uses N of a given module as opposed to only three. In most cases, N is selected as an odd number so that a majority voting arrangement can be used. The advantage of using N modules rather than three is that more module faults can often be tolerated.

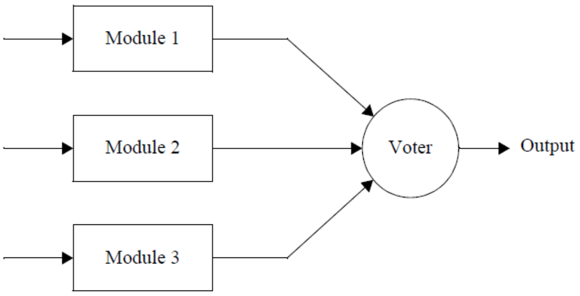


Figure 4.1: Basic TMR Model

For example, a 5MR system contains five replicated modules and a voter. A majority voting arrangement allows the 5MR system to produce correct results in the face of as many as two module faults. In many critical-computation applications, two faults must be tolerated to allow the required reliability and fault tolerance capabilities to be achieved. The primary tradeoff in NMR is the fault tolerance achieved versus the hardware required. Clearly, there must be some limit in practical applications on the amount of redundancy that can be employed. Power, weight, cost, and size limitations very often determine the value of N in an NMR system.

Voting within NMR systems can occur at several points. For example, an industrial controller can sample the temperature of a chemical process from three independent sensors, perform a vote to determine which of the three sensor values to use, calculate the amount of heat or cooling to provide to the process (the calculations being performed by three or more separate modules), and then vote on the calculations to determine a result. The voting can be performed on both analog and digital data. The alternative, in this example, might be to sample the temperature from three independent sensors, perform the calculations, and then provide a single vote on the final result. The primary difference between the two approaches is fault containment. If voting is not performed on the temperature values from the sensors, then the effect of a sensor fault is allowed to propagate beyond the sensors and into the primary calculations. Voting at the sensors, however, will mask, and contain, the effects of a sensor fault. Providing several levels of voting, however, does require additional redundancy, and the benefits of fault containment must be compared to the cost of the extra redundancy.

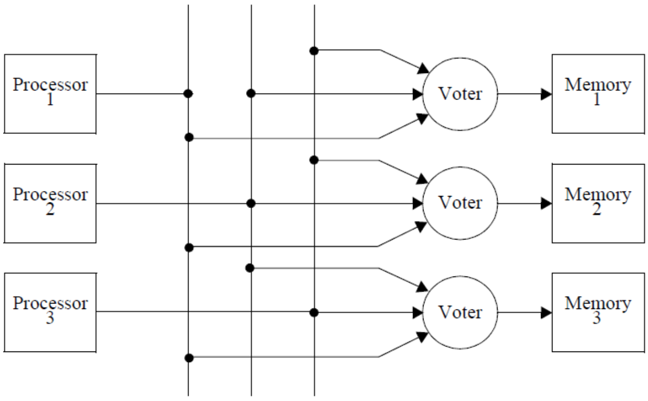


Figure 4.2: Triplicated voters in a TMR configuration

In addition to a number of design tradeoffs on voting, there are several problems with the voting procedure, as well. The first is deciding whether a hardware voter will be used, or whether the voting process will be implemented in software. A software voter takes advantage of the computational capabilities available in a processor to perform the voting process with a minimum amount of additional hardware. Also, the software voter provides the ability to modify the manner in which the voting is performed by simply modifying the software. The disadvantage of the software voter is that the voting can require more time to perform simply because the processor cannot execute instructions and process data as rapidly as a dedicated hardware voter. The decision to use hardware or software voting will typically depend upon:

- the availability of a processor to perform the voting,
- the speed at which voting must be performed,
- the criticality of space, power, and weight limitations,
- the number of different voters that must be provided, and
- the flexibility required of the voter with respect to future changes in the system.

The concept of software voting is shown in Figure 4.3. Each processor executes its own version of task A. Upon completion of the tasks, each processor shares its results with processor 2, who then votes on the results before using them as input to task B. If necessary, each processor might also execute its version of the voting routine and receive data from the other processors.

A second major problem with the practical application of voting is that the three results in a TMR system, for example, may not completely agree, even in a fault-free environment. The sensors that are used in many control systems can seldom be manufactured such that their values agree exactly. Also, an analog-to-digital converter can produce quantities that disagree in the least-significant bits, even if the exact signal is passed through the same converter multiple times. When values that disagree slightly are processed, the disagreement can propagate into larger discrepancies. In other words, small differences in inputs can produce large differences in outputs that can significantly affect the voting process. Consequently, a majority voter may find that no two results agree exactly in a TMR system, even though the system may be functioning perfectly.

One approach that alleviates the problem of the previous paragraph is called the *mid-value select* technique. Basically, the mid-value select approach chooses a value from the three available in a TMR system by selecting the value that lies between the remaining two. If three signals are available, and two of those signals are uncorrupted and the third is corrupted, one of the uncorrupted results should lie between the other uncorrupted result and the corrupted result. The mid-value select technique can be applied to any system that uses an odd number of modules such that one signal must lie in the middle of the others. The major difficulty with most techniques that use some form of voting is that a single result must ultimately be produced, thus creating a potential point where one failure can cause a system failure. Clearly, single-points-of-failure are to be avoided if a system is to be truly fault-tolerant.

Active Hardware Redundancy. Active hardware redundancy techniques attempt to achieve fault tolerance by fault detection, fault location, and fault recovery. In many designs faults can be detected because of the errors they produce, so in many instances error detection, error location and error recovery are the appropriate terms to use. The property of fault masking, however, is not obtained in the active redundancy approach. In other words, there is no attempt to prevent faults from producing errors within the system. Consequently, active approaches are most common in applications where temporary, erroneous results are acceptable as long as the system reconfigures and regains its operational status in a satisfactory length of time. Satellite systems are good examples of applications of active redundancy. Typically, it is not catastrophic if satellites have infrequent, temporary failures. In fact, it is usually preferable to have temporary failures than to provide the large quantities of redundancy necessary to achieve fault masking.

The basic operation of an active approach to fault tolerance is shown in Figure 4.3. During the normal operation of a system a fault can obviously occur. After the fault latency period, the

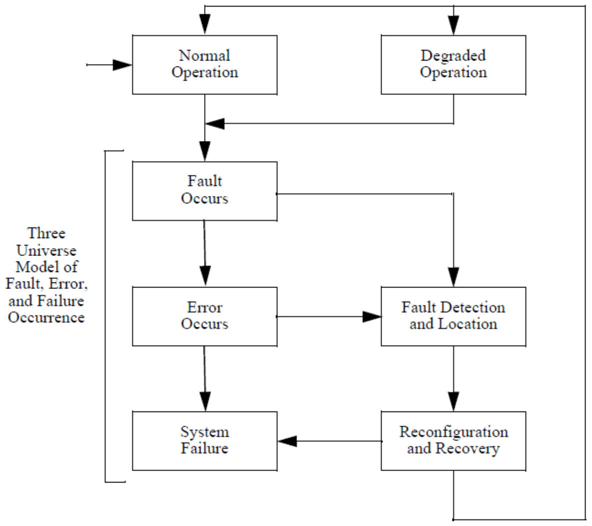

Normal Operation
Degraded Operation
Three Universe Model of Fault, Error, and Failure Occurrence
Fault Occurs
Error Occurs
Fault Detection and Location
System Failure
Reconfiguration and Recovery

Figure 4.3: A model of active approach to fault tolerance.

fault will produce an error which is either detected or it is not detected. If the error remains undetected, the result will be a system failure. The failure will occur after a latency period has expired. If the error is detected, the source of the error must be located, and the faulty component removed from operation. Next, a spare component must be enabled, and the system brought back to an operational state. It is important to note that the new operational state may be identical to the original operational state of the system or it may be a degraded mode of operation. The processes of fault location, fault containment, and fault recovery are normally referred to simply as reconfiguration.

It is clear from this description that active approaches to fault tolerance require fault detection and location capabilities.

Information Redundancy

Information redundancy is the addition of redundant information to data to allow fault detection fault masking, or possibly fault tolerance. Good examples of information redundancy are error detecting and error correcting codes, formed by the addition of redundant information to data words, or by the mapping of data words into new representations containing redundant information.

Before beginning the discussions of various codes, we will define several basic terms that will appear throughout this section. In general, a *code* is a means of representing information, or data, using a well-defined set of rules. A ***code word*** is a collection of symbols, often called digits if the symbols are numbers, used to represent a particular piece of data based upon a specified code. A ***binary code*** is one in which the symbols forming each code word consist of only the digits 0 and 1. For example, a Binary Coded Decimal (BCD) code defines a 4-bit code word for each decimal digit. The BCD code, for example, is clearly a binary code. A code word is said to be *valid* if the code word adheres to all of the rules that define the code; otherwise, the code word is said to be ***invalid.***

The *encoding operation* is the process of determining the corresponding code word for a particular data item. In other words, the encoding process takes an original data item and represents it as a code word using the rules of the code. The *decoding operation* is the process of recovering the original data from the code word. In other words, the decoding process takes a code word and determines the data that it represents. Of primary interest here are binary codes. In many binary code words, a single error in one of the binary digits will cause the resulting code word to no longer be correct, but, at the same time, the code word will be valid. Consequently, the user of the information has no means of determining the correctness of the information. It is possible, however, to create a binary code for which the valid code words are a subset of the total number of possible combinations of 1s and 0s. If the code words are formed correctly, errors introduced into a code word will force it to lie in the range of illegal, or invalid code words, and the error can be detected.

This is the basic concept of the *error detecting codes*. The basic concept of the *error correcting code* is that the code word is structured such that it is possible to determine the correct code word from the corrupted, or erroneous, code word. Typically, the code is described by the number of bit errors that can be corrected. For example, a code that can correct single-bit errors is called a *single error correcting code*. A code that can correct two-bit errors is called a *double-error correcting code*, and so on.

3.0.1.6 Relationship Between Security and Fault Tolerance

Security plays an increasingly important role for software system. Security concern must inform every phase of software development from problem domain to solution domain. Software security estimates provides the help for degree of protection and assess the impact. Microsoft has stated that above 50% of the security related problem for any firm has been found at design level of software development process. Software security touch points are based on good software engineering and involve explicitly pondering security throughout the software lifecycle. Security estimation of software may heavily affect to security of the final product. The experts tried in this regard to develop the security estimation guidelines, view and concept. There are some probabilities that original code segment may have some security flaws, anomalies that may influence security at different phase.

The security assessment is helpful for software developers, risk management team and executives of the company It definitely needs thoughtful subtle of security including security measurements, classifications and security attributes. Security attributes may decrease the cost and impinge between problem domains to solution domain at each phase of development life cycle. A level-2 heading must be in Italic, left-justified and numbered using an uppercase alphabetic letter followed by a period. Software Security is an external software attribute that reduces faults and effort required for secured software. Security must encompass dependable protection and secured the software system against all relevant concerns including confidentiality, integrity, availability, non-repudiation, survivability, accessibility despite attempted compromises, preventing, misuse and reducing the consequences of unforeseen threats. Fault tolerance is direct associated to security attributes such as confidentiality, integrity, availability, non-repudiations, and survivability. Fault tolerance thought will efficiently improve the security. Fault tolerance is frequently essential, but it can be riskily error-prone because of the added efforts that must be involved in the programme procedure. A consistent quantitative estimate of security is highly enviable at an early stage of software development life cycle. Fault tolerance is direct associated to reliability and security. Fault prevention and fault tolerance intend to present the ability to deliver an accurate service. Controlling and Monitoring can work mutually to enforce the security policy. Fault tolerance is the ability of a system to continue secures the software module and presence of software faults. Fault tolerance attributes as a fault masking, fault detection and fault consideration effective to security policy. Fault tolerance implies a savings in development time, cost and efforts; also, it reduces the number of components that must be originally developed.

3.0.2 METHODS FOR FAULT TOLERANT COMPUTING

Fault Tree Analysis (FTA) is a convent means to logically think through the many ways a failure may occur. It provides insights that may lead to product improvements or process controls. FTA provides a means to logically and graphically display the paths to failure for a system or component. One way to manage a complex system is to start with a reliability block diagram (RBD). Then create a fault tree for each block in the RBD. Whether a single block or a top level fault for a system the basic process to create a fault tree follows a basic pattern. This is comprises eight steps

- **Define the system.** This includes the scope of the analysis including defining what is considered a failure. This becomes important when a system may have an element fail or a single function fails and the remainder of the system still operates.
- **Define top-level faults.** Whether for a system or single block define the starting point for the analysis by detailing the failure of interest for the analysis.
- **Identify causes for top-level fault.** What events could cause the top level fault to occur? Use the logic gate symbols to organize which causes can cause the failure alone (or), or require multiple events to occur before the failure occurs (and).
- **Identify next level of events.** Each event leading to the top level failure may also have precipitating events.
- **Identify root causes.** For each event above continue to identify precipitating events or causes to identify the root or basic cause of the sequence of events leading to failure.
- **Add probabilities to events.** When possible add the actual or relative probability of occurrence of each event.

- **Analysis the fault tree.** Look for the most likely events that lead to failure, for single events the initiate multiple paths to failure, or patterns related to stresses, use, or operating conditions. Identify means to resolve or mitigate paths to failure.
- **Document the FTA.** Beyond the fault tree, graphics include salient notes from the discussion and action items.

Benefits of FTA

FTA is a convent means to logically think through the many ways a failure may occur. It provides insights that may lead to product improvements or process controls. It is a logical, graphical diagram that organizes the possible element failures and combination of failures that lead to the top level fault being studied. The converse, the success tree analysis, starts with the successful operation of a system, for example, and examines in a logical, graphical manner all the elements and combinations that have to work successfully.

With every product, there are numerous ways it can fail. Some more likely and possible than others. The FTA permits a team to think through and organize the sequences or patterns of faults that have to occur to cause a specific top level fault. The top level fault may be a specific type of failure, say the car will not start. Or it may be focused on a serious safety related failure, such as the starter motor overheats starting a fire. A complex system may have numerous FTA that each explore a different failure mode.

The primary benefit of FTA is the analysis provides a unique insight into the operation and potential failure of a system. This allows the development team to explore ways to eliminate or minimize the occurrence of product failure. By exploring the ways a failure mode can occur by exploring the individual failure causes and mechanisms, the changes impact the root cause of the potential failures.

The benefits include:

- Identify failures deductively. Using the logic of a detailed failure analysis and tools like ‘5 whys’, FTA helps the team focus on the causes of each event in a logical sequence that leads to the failure.
- Highlight the important elements of system related to system failure. The FTA process may lead to a single component or material that causes many paths to failure, thus improving that one element may minimize the possibly of many failures.
- Create a graphical aid for system analysis and management.
- Apparently managers like graphics, and for complex system, it helps to focus the team on critical elements.
- Provides an alternatively way to analysis the system. FMEA, RBD and other tools permit a way to explore system reliability, FTA provide a tool that focuses on failure modes one at a time. Sometimes a shift in the frame of reference illuminates new and important elements of the system.
- Focus on one fault at a time. The FTA can start with an overall failure mode, like the car not starting, or it can focus on one element of the vehicle failing, like the airbag not inflating as expected within a vehicle. The team chooses the area for focus at the start of the analysis.
- Expose system behavior and possible interactions. FTA allows the examination of the many ways a fault may occur and may expose non-obvious paths to failure that other analysis approaches miss.
- Account for human error. FTA includes hardware, software, and human factors in the analysis as needed. The FTA approach includes the full range of causes for a failure.
- Just another tool in the reliability engineering toolbox. For complex systems and with many possible ways that a significant fault may occur,

FTA provides a great way to organize and manage the exploration of the causes. The value comes from the insights created that lead to design changes to avoid or minimize the fault.

3.0.2.1 Fault Detection Methods

Fault detection plays an important role in high cost and safety-critical processes. Early detection of process faults can help avoid abnormal event progression. Fault detection determines the occurrence of fault in the monitored system. It consists of detection of faults in the processes, actuators and sensors by using dependencies between different measurable signals. Related tasks are also fault isolation and fault identification. Fault isolation determines the location and the type of fault whereas fault identification determines the magnitude (size) of the fault. Fault isolation and fault identification are together referred as fault diagnosis. The task of fault diagnosis consists of the determination of the type of the fault, with as many details as possible such as the fault size, location and time of detection.

There exist several overlapping taxonomies of the field. Some are more oriented toward control engineering approach, other to mathematical, Statistical and AI approach. Interesting divisions are described in the following division of fault detection methods Below:

A. Data Methods and Signal Models

- Limit checking and trend checking
- Data analysis (PCA)
- Spectrum analysis and parametric models
- Pattern recognition (neural nets)

B. Process Model Based Methods

- Parity equations
- State observers

- Parameter estimation
- Nonlinear models (neural nets)

C. Knowledge Based Methods

- Expert systems
- Fuzzy logic

3.0.2.2 Fault Tolerance Architecture

Several fault-tolerant architectures have been proposed in the literature in the past to address the circuit reliability concerns. A few of these relevant solutions include; Partial-TMR, Full-TMR, DARA-TMR , PaS , CPipe , STEM and Razor , which are discussed . We select this set of architectures because it includes a representative of each class of the broad spectrum of fault tolerant architectures.

PAIR-AND-A-SPARE

Pair-and-A-Spare (PaS) Redundancy was first introduced in as an approach that combines Duplication with Comparison and standby-sparing. In this scheme each module copy is coupled with a Fault-Detection (FD) unit to detect hardware anomalies within the scope of individual module. A comparator is used to detect inequalities in the results from two active modules. In the case of inequality, a switch decides which one of the two active modules is faulty by analyzing the reports from FD units and replaces it with a spare one. This scheme was intended to prevent hardware faults from causing system failures. The scheme fundamentally lacks protection against transient faults and it incurs a large hardware overhead to accomplish the identification of faulty modules.

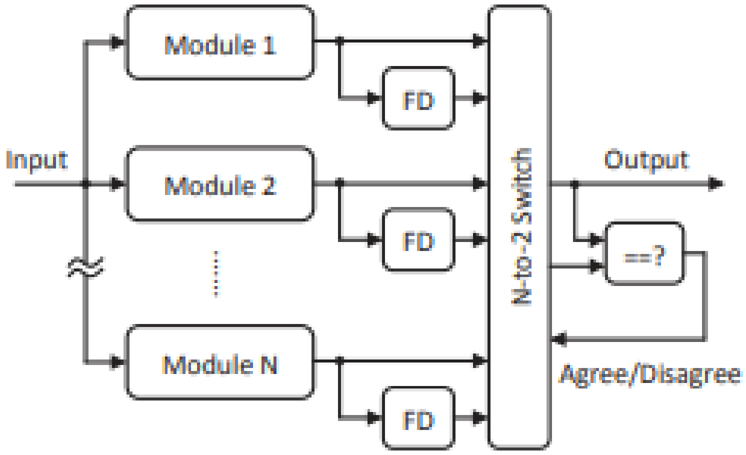


Figure 4.4: Pair-and-A-Spare

RAZOR

Razor is a well-known solution to achieve timing error resilience by using the technique called timing speculation. The principle idea behind this architecture is to employ temporally separated double-sampling of input data using Razor FFs, placed on critical paths. The main FF takes input sample on the rising edge of the clock, while a time-shifted clock (clk-del) to the shadow latch is used to take a second sample. By comparing the data of the main FF and the shadow latch, an error signal is generated. The timing diagram of how the architecture detects timing errors. In this example a timing fault in CL A causes the data to arrive late enough that the main FF captures wrong data but the shadow always latches the input data correctly. The error is signaled by the XOR gate which propagates through the OR-tree for correction action to be taken. Error recovery in Razor is possible either by clock-gating or by rollback recovery. Razor also uses Dynamic Voltage Scaling (DVS) scheme to optimize the energy vs. error rate trade-off.

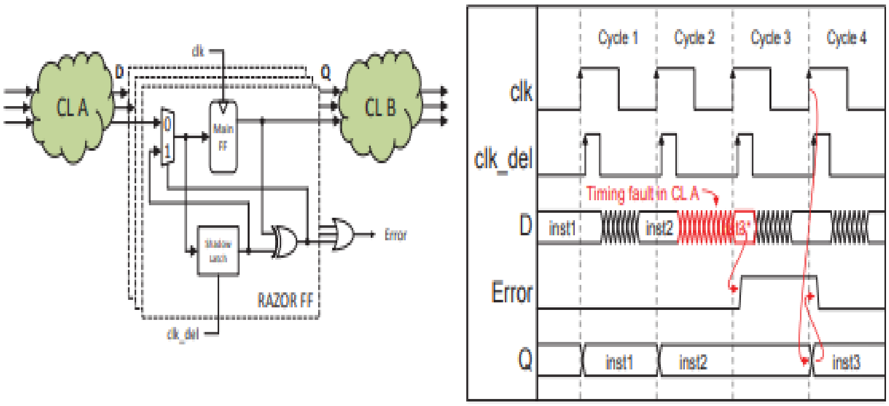


Figure 4.5: RAZOR Architecture

STEM

STEM cell architecture takes Razor a step further by incorporating capability to deal with transient faults as well. STEM cell architecture presented in incorporates power saving and performance enhancement mechanisms like Dynamic Frequency Scaling (DFS) to operate circuits beyond their worst-case limits. Similar to Razor FFs, a STEM cells replace the FF on the circuit critical paths, but instead of taking two temporally separated samples, A STEM cell takes three samples using two delayed clocks. Mismatches are detected by the comparators and the error signals is used to select a sample which is most likely to be correct for rollback.

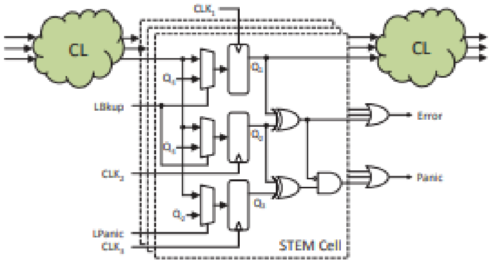


Figure 4.6: STEM Architecture

CPipe

The CPipe or Conjoined Pipeline architecture proposed uses spatial and temporal redundancy to detect and recover from timing and transient errors. It duplicates CL blocks and the FFs as well to from two pipelines interlinked together. The primary or leading pipeline is overclocked to speedup execution while the replicated of shadow pipeline has sufficient speed margin to be free from timing errors. Comparators placed across the leading pipeline register in somewhat similar way as the scheme, detects any metastable state of leading pipeline register and SETs reaching the registers during the latching window. The error recovery is achieved by stalling the pipelines and using data from the shadow pipeline registers for rollback and it takes 3 cycles to complete.

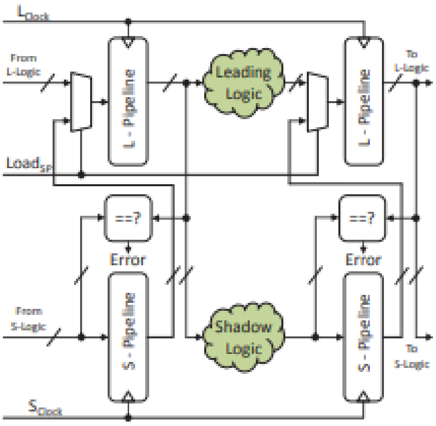
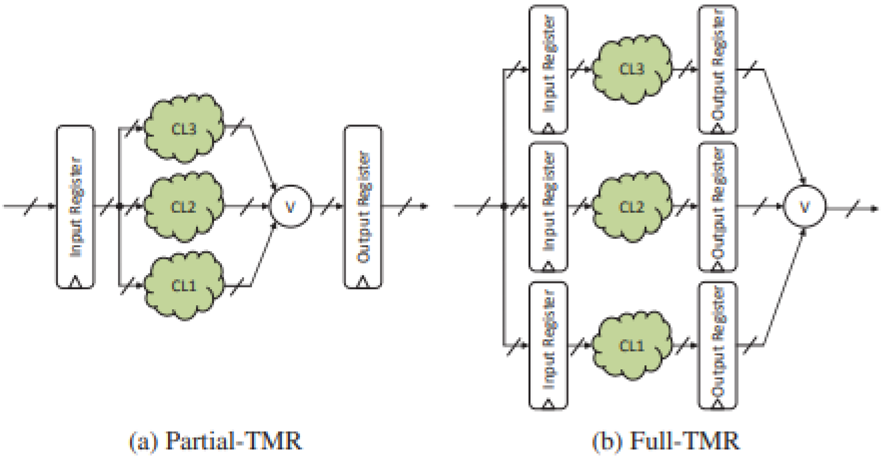


Figure 4.7: CPipe Architecture

TMR

TMR is one of the most p5opular fault tolerant architectures. In a basic TMR scheme called Partial-TMR, we have three implementation of same logic function and their outputs are voted by a voter circuit. This architecture can tolerate all the single-faults occurring in the CL block but faults in voter or pipeline registers cause the system to fail. Full-TMR on the other hand, triplicates the entire circuit including the FFs and can tolerate all single-faults in any part of the circuit except voter and the signals to the input pipeline register, which may result in common-mode failure.



(a) Partial-TMR

(b) Full-TMR

Figure 4.8**:** TMR Architecture

DARA-TMR

DARA-TMR triplicates entire pipeline but uses only two pipeline copies to run identical process threads in Dual Modular Redundancy (DMR) mode. The third pipeline copy is disabled using power gating and is only engaged for diagnosis purposes in case of very frequent errors reported by the detection circuitry. Once the defective pipeline is identified the system returns back to DMR redundancy mode by putting the defected pipeline in off mode. The error recovery follows the same mechanism as pipeline branch misprediction, making use of architectural components for error recovery. DARA-TMR treats permanent fault occurrence as a very rare phenomenon and undergo a lengthy reconfiguration mechanism to isolate them .

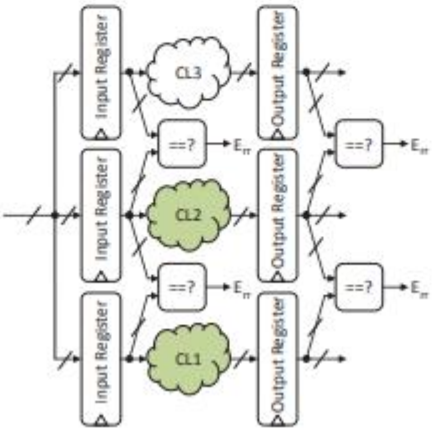


Figure 4.9: TMR Architecture

Hybrid Fault-Tolerant Architecture

HyFT architecture employs information redundancy (as duplication with comparison) for error detection, timing redundancy (in the form of re-computation/rollback) for transient and timing error correction and hardware redundancy (to support reconfiguration) for permanent error correction. the HyFT architecture employs triplication of CL blocks. A set of multiplexers and demultiplexer is used to select two primary CL copies and to put the third CL copy in standby mode during normal operation. HyFT architecture is driven by a control logic module that generates the necessary control signals. HyFT architecture uses the pseudo-dynamic comparator for error detection to achieve better glitch detection capability and to reduce the power consumption. The HyFT architecture uses a concurrent error detection mechanism. A pseudo-dynamic comparator compares the outputs of two active CL copies. It can be seen that the comparator is placed across the output register such that it gets to compare the output of the output register Sout, which is a synchronous signal with the output of the secondary running copy Aout, which is an asynchronous signal. This orientation of the pseudo-dynamic comparator also offers marginal protection against the errors due to faults in the output register and allows it to remain off the critical path. Thus, it does not impact the temporal performance of the circuit. The error recovery scheme uses stage-level granularity reconfigurations and single-cycle deep rollbacks. The shadow latches incorporated in pipeline registers keep one clock cycle preceding state of the pipeline register FFs. The comparison takes place after every clock cycle. Thus, error detection can invoke a reconfiguration and a rollback cycle, confining the error and preventing it from effecting the computation in the following cycles. The comparison takes place only during brief intervals of time referred to as comparison window. The timing of comparison window is defined by the high phase of a delayed clock signal DC, which is generated from CLK using a delay element. These brief comparisons allow keeping the switching activity in OR-tree of the comparator to a minimum, offering a 30% power reduction compared with a static comparator. The functioning of the pseudo-dynamic comparator requires specific timing constraints to be applied during synthesis of CL blocks, as defined below. Timing Constraints: In typical pipeline circuits the contamination delay of CL should respect hold-time of the pipeline register latches. However, in the HyFT architecture, as CL also feeds to the pseudo-dynamic comparator, CL outputs need to remain stable during the comparison. And since the comparison takes place just after a clock edge, any short paths in the CL can cause the input signals of the comparator to start changing before the lapse of the comparison-window. Thus, the CL copies have to be synthesized with minimum delay constraints governed by:

$tcd > \delta t - tccq - tcdm - tcm$

where:

tcd = CL contamination delay

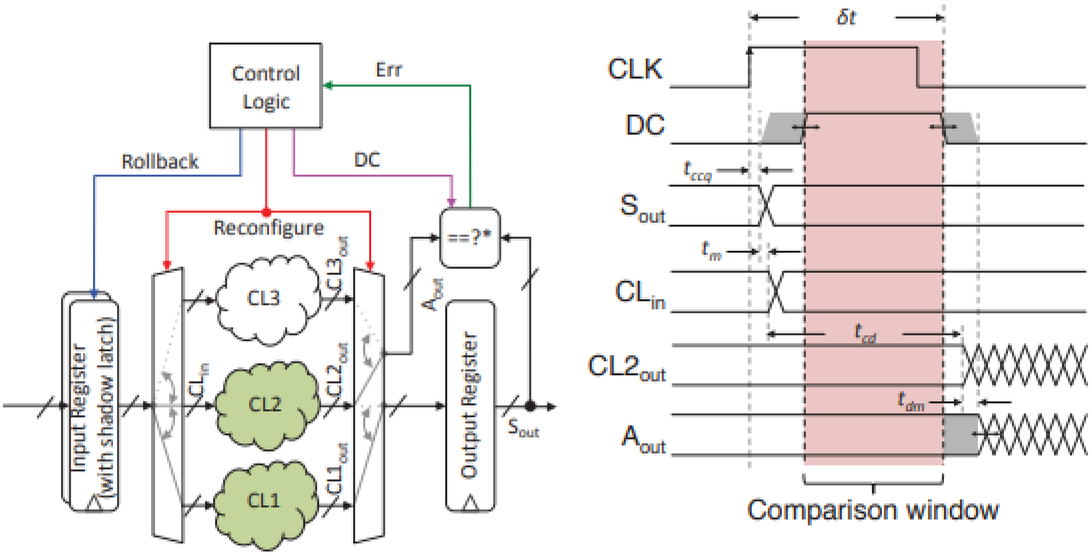
δt = the amount of time between CLK capture edge and the lapse of the comparison-window

tccq = FF clk-to-output delay

tcdm = demultiplexer contamination delay

tcm = multiplexer contamination delay

with the help of a timing diagram we explain the associated timing constraints. Besides CLK and DC the timing diagram shows the signals at the two inputs of the comparator labeled as Aout and Sout also



(a) Hybrid Fault Tolerant Architecture

(b) Timing constraints imposed on CL synthesis by pseudo-dynamic comparator

Figure 4.10**:** HyFT Architecture

indicated. The remaining two signals are the inputs of CL labeled as CLin and the outputs of CL labeled as CLout. The grey shaded regions allow margins of the corresponding signals. The timing allowance for the start of the comparison-window depends on the clk-to-output delay of the output register. This implies that the comparison should not begin until the output of the output register stabilizes.

3.0.2.3 Fault Models

A fault model attempts to identify and categorize the faults that may occur in a system, in order to provide clues as to how to fine-tune the software development environment and how to improve error detection and recovery. A question that needs to be asked is: is the traditional distributed systems fault model appropriate for Grid computing, or are refinements necessary?

The development of fault models is an essential part of the process in determining the reliability of a system. A fault model describes the types of faults that a system can develop, specifying where and how they will occur in it. However, faults become more difficult to formulate sensibly as a system is viewed at an increasingly more abstract level, especially the definition of how a fault manifests itself. The entities listed in a fault model need not necessarily physically exist, but may be abstractions of real-world objects. In general, a fault model is an abstracted representation of the physical defects which can occur in a system, such that it can be employed to usefully, and reasonably accurately, simulate the behaviour of the system over its intended lifetime with respect to its reliability. Four major goals exist when devising a fault model:

1. The abstract faults described in the model should adequately cover the effects of the physical faults which occur in the real-world system.
2. The computational requirements for simulation should be satisfiable.
3. The fault model should be conceptually simple and easy to use.
4. It should provide an insight into introducing fault tolerance in a design

3.0.2.4 Fault Tolerance Methods

A Fault in any software system, usually, happens due to the gaps left unnoticed during the design phase. Based on this, the fault tolerance techniques are identified into two different groups, that is, the Single Version Technique and the Multi-Version Technique. There can be plenty of techniques implemented under each of these categories, and a few of the techniques often used by the programmers are,

1. **Software Structure and Actions**

   When the software system is one single block of code, it is logically more vulnerable to failure. Because, when one tiny error occurs in the program, the whole system will be brought down. Hence, it is crucial for the software system should be structured in a modular form, where the functionality is covered in separate modules. In the case of failure, each module should hold specific instructions on how to handle it and let the other modules run as usual, instead of passing on the failure from module to module.

2. **Error Detection**

   Error Detection is a fault tolerance technique where the program locates every incidence of error in the system. This technique is practically implemented using two attributes, namely, self-protection and self-checking. The Self-Protection attribute of error detection is used for spotting the errors in the external modules, whereas the Self-Checking attribute of error detection is used for spotting the errors in the internal module.

3. **Exception Handling**

   Exception Handling is a technique used for redirecting the execution flow towards the route to recovery whenever an error occurs in the normal functional flow. As a part of fault tolerance, this activity is performed under three different software components, such as the Interface Exception, the Local Exception and the Failure Exception.

4. **Checkpoint and Restart**

This is one of the commonly used recuperation methods for single version software systems. The Checkpoint and Restart fault tolerance technique can be used for the events like run-time exceptions, that is, a malfunction takes place during the run-time and when the execution is complete there is no record of the error happening. For this case, the programmer can place checkpoints in the program and instruct the program to restart immediately right from the occurrence of the error.

5. **Process Pairs**

Process Pair technique is a method of using the same software in two different hardware units and validating the functional differences in order to capture the faulty areas. This technique functions on top of the checkpoint and restart technique, as similar checkpoints and restart instructions are placed in both systems.

6. **Data Diversity**

Data Diversity technique is typically a process where the programmer passes a set of input data, and places checkpoints for detecting the slippage. The commonly used Data Diversity models are ‘Input Data Re-Expression’ model, ‘Input Data Re-Expression with Post-Execution Adjustment’ model, and ‘Re-Expression via Decomposition and Recombination’ model.

7. **Recovery Blocks**

Recovery Block technique for multiple version software Fault Tolerance involves the checkpoint and restart method, where the checkpoints are placed before the fault occurrence, and the system is instructed to move on to next version to continue the flow. It is carried out in three areas, that is, the main module, the acceptance tests, and the swap module.

8. **N – Version Programming**

The N – Version programming technique for the multi – version fault tolerance is the commonly used method when the there is a provision for testing multiple code editions. The recovery is made from executing all the versions and comparing the outputs from each of the versions. This technique also involves the acceptance test flow.

9. **N Self–Checking Programming**

N Self – Checking Programming is a combination technique of both the Recovery block and the N – version Programming techniques, which also calls for the acceptance test execution. It is performed by the sequential and the parallel execution of various versions of the software.

10. **Consensus Recovery Blocks**

This method combines the Recovery Block and the N- Version Programming techniques where the decision algorithm technique is combined for handling and recovering the inaccuracy in the system. This combination of all the efficient fault tolerance techniques gives a much more consistent method of Fault tolerance.

3.0.2.5 Major Issues in Modelling and Evaluation

- **Interference with fault detection in the same component.** In passenger vehicle example, with either of the fault-tolerant systems it may not be obvious to the driver when a tire has been punctured. This is usually handled with a separate "automated fault-detection system". In the case of the tire, an air pressure monitor detects the loss of pressure and notifies the driver. The alternative is a "manual fault-detection system", such as manually inspecting all tires at each stop.

- **Interference with fault detection in another component.** Another variation of this problem is when fault tolerance in one component prevents fault detection in a different component. For example, if component B performs some operation based on the output from component A, then fault tolerance in B can hide a problem with A. If component B is later changed (to a less fault-tolerant design) the system may fail suddenly, making it appear that the new component B is the problem. Only after the system has been carefully scrutinized will it become clear that the root problem is actually with component A.
- **Reduction of priority of fault correction.** Even if the operator is aware of the fault, having a fault-tolerant system is likely to reduce the importance of repairing the fault. If the faults are not corrected, this will eventually lead to system failure, when the fault-tolerant component fails completely or when all redundant components have also failed.
- **Test difficulty.** For certain critical fault-tolerant systems, such as a nuclear reactor, there is no easy way to verify that the backup components are functional. The most infamous example of this is Chernobyl, where operators tested the emergency backup cooling by disabling primary and secondary cooling. The backup failed, resulting in a core meltdown and massive release of radiation.
- **Cost.** Both fault-tolerant components and redundant components tend to increase cost. This can be a purely economic cost or can include other measures, such as weight. Manned spaceships, for example, have

so many redundant and fault-tolerant components that their weight is increased dramatically over unmanned systems, which don't require the same level of safety.

- **Inferior components.** A fault-tolerant design may allow for the use of inferior components, which would have otherwise made the system inoperable. While this practice has the potential to mitigate the cost increase, use of multiple inferior components may lower the reliability of the system to a level equal to, or even worse than, a comparable non-fault-tolerant system.

3.0.2.6 Fault Tolerance for Web Applications

In web services when a fault occurs, it goes into various stages. When an error occurs in web services, it should make sure the error or faults through various fault detection mechanism to know the failure causes so that failed components can be repaired of recovered from an error. The flow of web service failure responses shown in figure 4.11.

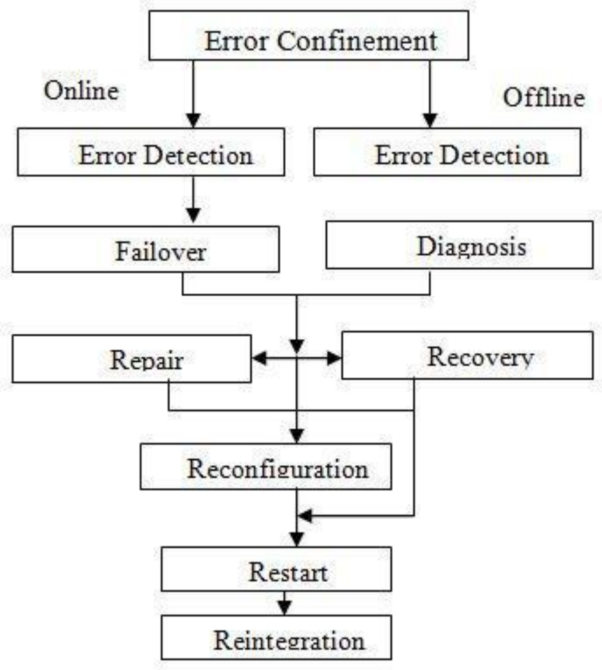


Figure 4.11: Failure Stages of Web services

A) **Error Confinement**: Error confinement stage prevents an error effect on web services. It can be gain with the help of error detection within a service by multiple checks.

B) **Error Detection:** Error detection stage helps in identifying unexpected error in a web service.

C) **Error Diagnosis:** Error diagnosis stage helps to diagnose the fault that has been traced in error detection stage. Error diagnosis stage comes into picture when error detection doesn't provide enough information about fault location.

D) **Reconfiguration:** Reconfiguration comes into picture when and error is detected and located in the error detection and error diagnosis stage.

E) **Recovery:** Recovery is used to recover fault from web service using retry and rollback approaches.

F) **Restart:** Restart comes into picture after the recovery of web service. Restart can be done either using hot start or cold start.

G) **Repair:** In Repair, failed component has to be changed in order to work properly.

H) **Reintegration:** In the reintegration stage repaired component has to be integrating.

In web services, there are many fault tolerant techniques that can be applied such as replication. Replication is a more efficient technique for handling exception in a distributed application. Services can resume more effectively by maintaining the global state of the application. For instance, let's assume if one service needs the assistance of another service to provide the desired result to the customer then service needs to communicate with other service. Suppose, while communicating with other service, at certain point of time if a fault occurs in a service, then there is no need to continue service with faults. Then the state manager has to roll back the state of the application at that point where the fault occurred so that service can resume without fault and response can be given to the consumer more effectively.

Fault Tolerance Implementation in Cloud Computing

A cloud is a type of parallel and distributed system containing a set of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers. It is a style of computing where service is provided across the Internet using different models and layers of abstraction, It refers to the applications delivered as services to the mass, ranging from the end-users hosting their personal documents on the Internet to enterprises outsourcing their entire IT infrastructure to external data centers. A simple example of cloud computing service is Yahoo email or Gmail etc. Although cloud computing has been widely adopted by the industry, still there are many research issues to be fully addressed like fault tolerance, workflow scheduling, workflow management, security etc. Fault tolerance is one of the key issues amongst all. It is concerned with all the techniques necessary to enable a system to tolerate software faults remaining in the system after its development. When a fault occurs, these techniques provide mechanisms to the software system to prevent system failure occurrence. The main benefits of implementing fault tolerance in cloud computing include failure recovery, lower cost, improved performance metrics etc. A cloud infrastructure consist of the following broad components:

- **Servers** – The physical machines that act as host machines for one or more e virtual machines.
- **Virtualization** – Technology that abstracts physical components such as servers, storage, and networking and provides these as logical resources.

- **Storage** – In the form of Storage Area Networks (SAN), network attached storage (NAS), disk drives etc. Along with facilities as archiving and backup.
- **Network** – To provide interconnections between physical servers and storage.
- **Management** – Various software for configuring, management and monitoring of cloud infrastructure including servers, network, and storage devices.
- **Security** – Components that provide integrity, availability, and confidentiality of data and security of information, in general.
- **Backup and recovery services.**

The cloud computing, as a fast advancing technology, is increasingly being used to host many business or enterprise applications. However, the extensive use of the cloud-based services for hosting business or enterprise applications leads to service reliability and availability issues for both service providers and users. These issues are intrinsic to cloud computing because of its highly distributed nature, heterogeneity of resources and the massive scale of operation. Consequently, several types of faults may occur in the cloud environment leading to failures and performance degradation. The major types of faults are listed as follows:

- **Network fault:** Since cloud computing resources are accessed over a network (Internet), a predominant cause of failures in cloud computing are the network faults. These faults may occur due to partitions in the network, packet loss or corruption, congestion, failure of the destination node or link, etc.
  - **Physical faults:** These are faults that mainly occur in hardware resources, such as faults in CPUs, in memory, in storage, failure of power etc.

- Process faults: faults may occur in processes because of resource shortage, bugs in software, incompetent processing capabilities, etc.
- Service expiry fault: If a resource's service time expires while an application that leased it is using it, it leads to service failures.

The Fault Tolerance methods can be applied to cloud computing in three levels:

- **At hardware level:** if the attack on a hardware resource causes the system failure, then its effect can be compensated by using additional hardware resources.
- **At software (s/w) level:** Fault tolerance techniques such as checkpoint restart and recovery methods can be used to progress system execution in the event of failures due to security attacks.
- **At system level:** At this level, fault tolerance measures can compensate failure in system amenities and guarantee the availability of network and other resources.

Challenges of Implementing Fault Tolerance in Cloud Computing

Providing fault tolerance requires careful consideration and analysis because of their complexity, inter-dependability and the following reasons.

- There is a need to implement autonomic fault tolerance technique for multiple instances of an application running on several virtual machines
- Different technologies from competing vendors of cloud infrastructure need to be integrated for establishing a reliable system
- The new approach needs to be developed that integrate these fault tolerance techniques with existing workflow scheduling algorithms
- A benchmark based method can be developed in cloud environment for evaluating the performances of fault tolerance component in comparison with similar ones
- To ensure high reliability and availability multiple clouds computing providers with
- independent software stacks should be used

- Autonomic fault tolerance must react to synchronization among various clouds

4.0 CONCLUSION

Fault-tolerance is achieved by applying a set of analysis and design techniques to create systems with dramatically improved dependability. As new technologies are developed and new applications arise, new fault-tolerance approaches are also needed. In the early days of fault-tolerant computing, it was possible to craft specific hardware and software solutions from the ground up, but now chips contain complex, highly-integrated functions, and hardware and software must be crafted to meet a variety of standards to be economically viable. Thus, a great deal of current research focuses on implementing fault tolerance using Commercial-Off-The-Shelf (COTs) technology.

Recent developments include the adaptation of existing fault-tolerance techniques to RAID disks where information is striped across several disks to improve bandwidth and a redundant disk is used to hold encoded information so that data can be reconstructed if a disk fails. Another area is the use of application-based fault-tolerance techniques to detect errors in high performance parallel processors. Fault-tolerance techniques are expected to become increasingly important in deep sub-micron VLSI devices to combat increasing noise problems and improve yield by tolerating defects that are likely to occur on very large, complex chips.

5.0 SUMMARY

The ability of a system to continue operation despite a failure of any single element within the system implies the system is not in a series configuration. There is some set of redundancy or set of alternative means to continue operation. The system may use multiple redundancy elements, or be resilient to changes in the system’s configuration. The appropriate solution to create a fault tolerant system often requires careful planning, understanding of how elements fail and the impact of surrounding elements of the failure.

Fault-tolerance techniques will become even more important the next years. The ideal, from an application writer's point of view, is total hardware fault-tolerance. Trends in the market, e.g. Stratus and Sun Netra, shows that this is the way systems go at the moment. There is also, fortunately, reason to believe that such systems will become considerable cheaper than today. Technology in general, and miniaturization in particular (which leads to physically smaller and in general cheaper systems) contributes to this. Much research is also being done with clusters of commercial general-purpose computers connected with redundant buses. In that case, the software has to handle the failures. However, as shown with the HA Cluster and Sun Netra, that could also be done without affecting the user programs and applications.

6.0 TUTOR MARKED ASSIGNMENT

1. What Is Fault Tolerance?
2. Explain the different Fault Tolerance Architecture that know?
3. Explain the Methods for Fault Tolerant Computing
4. Describe the different forms of hardware Redundancy
5. Explain the properties of Fault Tolerant Systems

7.0 REFERENCES/FURTHER READING

Amma A. D. T., Pramod V. R and N. Radhika, (2012) "ISM for Analyzing the Interrelationship between the Inhibitors of Cloud Computing", vol. 2, No. 3.

Anderson T. and Knight J. C. (1983), "A Framework for software Fault tolerance in Real time System", IEEE Transaction on software Engineering, Vol. 9, No.3.

Balaji E. and Krishnamurthy P. (1996). "Modeling ASIC memories in VHDL". In: Design Automation Conference, with EURO-VHDL '96 and Exhibition, Proceedings EURODAC '96, European, pp. 502–508. DOI: 10.1109/EURDAC.1996.558250.

Engineering Safety Requirements, Safety Constraints, and Safety Critical Requirements, Available at:

http://www.jot.fm/issues/issue_2004_03/column3/ last visit November 17, 2021.

Fred B. Schneider (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. A.C.M. Computing Surveys, 22(4):299–319.

Fred B. Schneider. (1997) Towards fault-tolerant and secure agentry. Technical report, Cornell University, Department of Computer Science.

Johnson, B. W. (1996). An introduction to the design and analysis of fault-tolerant systems. *Fault-tolerant computer system design*, *1*, 1-84.

Kim, E. P., & Shanbhag, N. R. (2012). Soft N-modular redundancy. IEEE Transactions on Computers, 61(3), 323–336.

Lyu, M. and Mendiratta V, (1999) “Software Fault Tolerance in a Clustered Architecture: Techniques and Reliability Modeling,” In Proceedings' of IEEE Aerospace Conference, Snowmass, Colorado, vol.5, pp.141-150, 6-13.

Neuman, P (2000) “Practical Architecture for survivable system and networks”, Phase Two Project 1688, SRI International, Menlo Park, California.

Patton, R. J. (2015). Fault-tolerant control. Encyclopedia of systems and control, 422–428.

Richard D. Schlichting and Fred B. Schneider. (1983) Fail-stop processors: An approach to designing fault-tolerant computing systems. A.C.M. Transactions on Computer Systems, 1(3):222– 238.

Walton G. H., Long Taff T.A. and R. C. Linder, (1997) “Computational Evaluation of Software Security attributes”, IEEE.