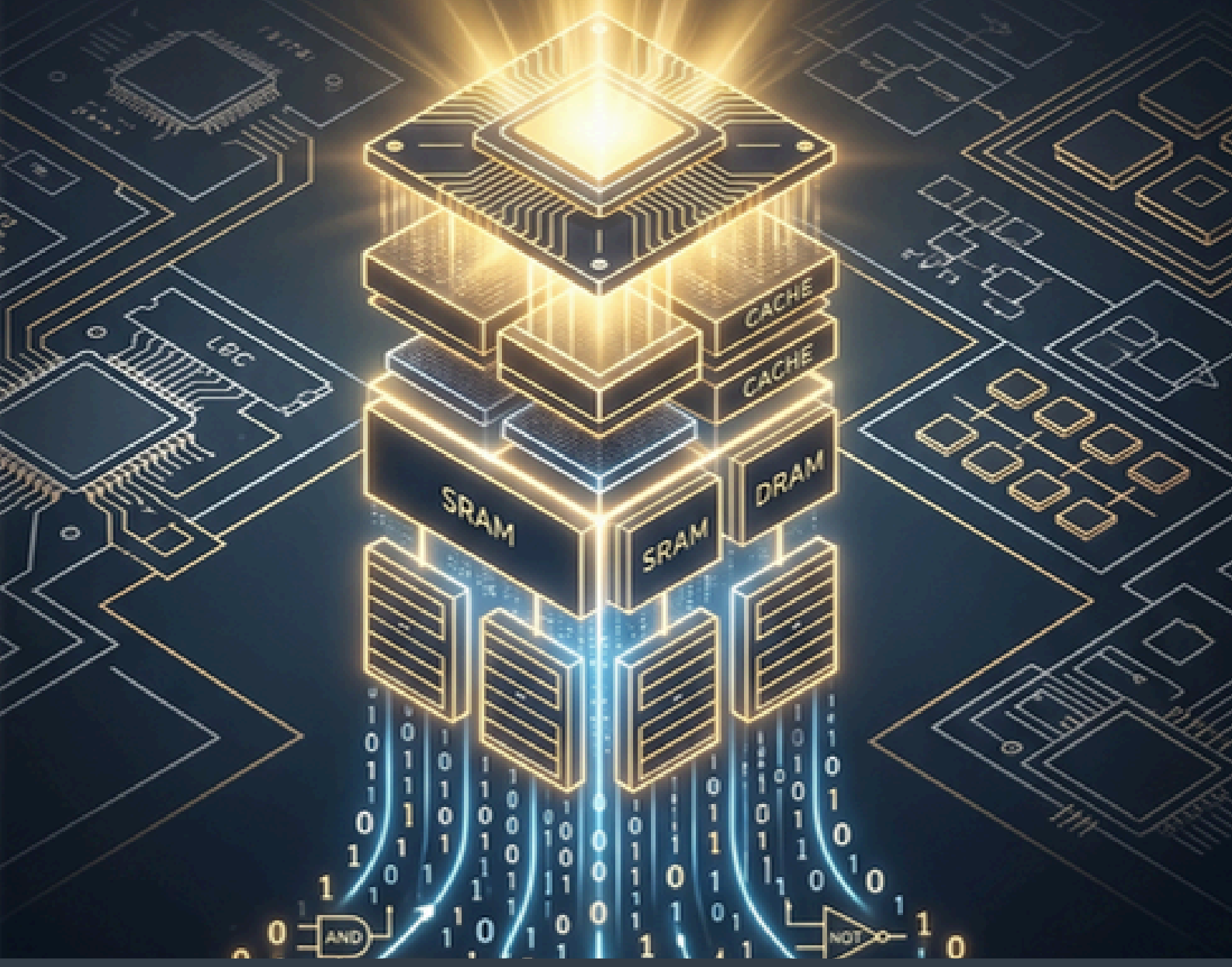
FROM BITS TO BRILLIANCE

MASTERING COMPUTER ORGANIZATION AND ARCHITECTURE



Piyali De, Atanu Das, Amartya Ghosh,
Ranjan Banerjee, and Ayush Alam

From Bits to Brilliance : Mastering Computer Organization and Architecture

Piyali De

Assistant Professor, Computer Science Engineering, Brainware University

Atanu Das

Assistant Professor, Computer Science Engineering, Brainware University

Amartya Ghosh

Assistant Professor, Computer Science Engineering, Brainware University

Ranjan Banerjee

Assistant Professor, Computer Science Engineering, Brainware University

Ayush Alam

Computer Science Engineering, Brainware University

Published By

World Academic Press, Kolkata- 700126, India

www.worldacademic.press

From Bits to Brilliance : Mastering Computer Organization and Architecture

Published by:

World Academic Press , Kolkata, India

www.worldacademic.press

DOI: *https://www.doi.org/10.66727/wap.9788199835375*

ISBN: 978-81-998353-5-1 (Paperback)

ISBN: 978-81-998353-7-5 (E- book)

First Edition: 2026

Table of Contents

Chapter 1

Introduction to Digital Computer

1.1 Introduction

In the vast landscape of modern technology, the intricacies of computer systems lie at the heart of nearly every facet of our lives. From the smartphones in our pockets to the towering data centers powering the internet, understanding the fundamental principles of computer organization and architecture is essential for navigating this digital age. This book serves as a comprehensive guide to the foundational concepts that underpin the design and operation of computers. Whether you are a student embarking on a journey into the realms of computer science or a seasoned professional seeking to deepen your understanding, the material presented here will provide valuable insights into the inner workings of computing systems.

At its core, computer organization deals with the physical components of a computer and how they interact to execute instructions. From the central processing unit (CPU) to memory, input/output devices, and storage, every aspect of a computer's architecture plays a crucial role in determining its performance and capabilities. Meanwhile, computer architecture focuses on the design principles and structures that dictate how these components are organized and interconnected to form a coherent system. By studying architectural concepts such as instruction set architecture (ISA), pipelining, caching, and parallelism, we gain a deeper appreciation for the ways in which computational tasks are executed and optimized.

Throughout this book, we will embark on a journey through the layers of abstraction that define modern computing. Starting with the basic building blocks of digital logic and binary representation, we will gradually ascend to higher levels of abstraction, exploring topics such as assembly language programming, memory hierarchy design, and multiprocessor systems.

In addition to covering theoretical principles, we will also delve into practical applications and real- world case studies that illustrate how these concepts manifest in modern computing environments. Whether it's understanding the trade- offs involved in designing a new processor architecture or optimizing software for performance, the principles discussed in this book will provide valuable insights for engineers, architects, and programmers alike. It is our hope that this book will not only serve as a valuable resource for academic study but also inspire readers to explore the ever- evolving field of computer organization and architecture. By mastering the fundamental principles outlined in these pages, we can better understand the technology that shapes our world and harness its power to drive innovation and discovery.

Now we illustrate the two basic terms computer organization and computer architecture. Computer organization and architecture are closely related concepts in the field of computer science, but they refer to different aspects of designing and understanding computer systems.

1.2 Digital Computer

Digital Computer is an electronic machine or a device that helps to process any kind of information. These are the devices through which users provide some input and get the output within a fraction of a second.A digital computer is a combination of Hardware and Software.

Hardware: - Hardware refers to the physical components of a computer. Computer Hardware is any part of the computer that we can touch these parts. Examples of hardware in a computer are the Processor, Memory Devices, Monitor, Printer, Keyboard, Mouse.

Software: - Software is a collection of instructions, procedures, documentation that performs different tasks on a computer system. Examples of software are Ms Word, Excel, Power Point, Google Chrome, Photoshop, MySQL etc.Computer software is classified into two types

Application Software: - Application software is used to perform a particular task. Examples of application software are Ms Word, Excel, Power Point, Google Chrome, Photoshop, MySQL etc.

System Software: - System software provides an environment on which application software can execute and also provides an interface between user and computer hardware. Examples of system software are operating system (OS), compiler, linker etc.

Among all system software OS is the primary one because without it users are unable to use any computer hardware and perform any task. Examples of operating systems are different versions of Microsoft Windows like XP, Vista,7, 8,10, Unix, different distribution of Linux like mint, fedora, Red Hat, MAC etc

1.2.1. Components of Digital Computer

A computer system consists of mainly four basic units

Input unit.

Storage unit/Memory.

Central Processing Unit (CPU).

Output unit.

Central Processing unit further includes Arithmetic logic unit and control unit. Storage unit is divided into two parts Primary and Secondary memory. These components of a computer perform four major operations or functions

- It accepts data or instructions as input,
- It stores data and instruction
- It processes data as per the instructions,
- It gives results in the form of output.

Input Unit: - An input unit is a device through which data and programs from user are entered into the computer. The input unit links the external environment with the computer system. Input unit accepts data and instruction in human readable form, but it transforms data and instructions into binary codes before further processing. This transformation is accomplished by units that called input interfaces.

In short, an input unit performs the following functions.

1. It accepts (or reads) the list of instructions and data from the outside world.
2. It converts these instructions and data in computer acceptable format.
3. It supplies the converted instructions and data to the computer system for further processing.

Example: - Keyboard, mouse, scanner and bar- code reader.

Output Unit: - An output unit is a device through which results stored in the computer memory are made available to the user. It links the computer with the external environment. As computers work with binary code, the results produced are also in the binary form. Hence, before supplying the results to the outside world, it must be converted to human acceptable (readable) form. This task is accomplished by units called output interfaces.

In short, the following functions are performed by an output unit.

1. It accepts the results produced by the computer which are in coded form and hence cannot be easily understood by us.
2. It converts these coded results to human acceptable (readable) form.
3. It supplied the converted results to the outside world.

Example: - Monitor, printer and speaker.

Storage Unit/Memory: - The data and instructions that are entered into the computer system through input units have to be stored inside the computer before the actual processing starts. Similarly, the results produced by the computer after processing must also be kept somewhere inside the computer system temporarily or permanently before being passed on to the output units. Storage unit is designed for this purpose only.

the following functions are performed by memory unit

1. All the data to be processed and the instruction required for processing (received from input devices).
2. Intermediate results of processing.
3. Final results of processing before these results are released to an output device.

There are two types of storage unit available in all modern computers

- **Primary Memory**
- **Secondary Memory**

Primary Memory: - This type of memory is usually small in size but very fast. Primary Memory is internal memory of the computer. RAM AND ROM both form part of primary memory. The primary memory provides main working space to the computer.

Random Access Memory(RAM)/Main Memory: - the data and instructions needs to be stored in the main memory before execution. The primary storage is referred to as random access memory (RAM) because it is possible to randomly select and use any location of the memory directly store and retrieve data. The storage of data and instructions inside the primary storage is disappeared from RAM as soon as the power to the computer is switched off. That's why main memory is also known as volatile memory. There are two types of RAM:

Read Only Memory (ROM): - here is another memory in computer, which is called Read Only Memory (ROM). The storage of program and data in the ROM is permanent. The ROM stores some standard processing programs supplied by the manufacturers to operate the personal computer. The ROM can only be read by the CPU but it cannot be changed. ROM does not lose their content on failure of power supply. That's why, it is known as non- volatile memory. There are different types of ROM available

- Programmable Read Only Memory (PROM): - Data is written into a ROM at the time of manufacture. However, the content can be programmed by user with a special PROM programmer. PROM provides flexible and economical fixed programs and data.
- Erasable Programmable Read Only Memory (EPROM): - this allow the programmer to erase the contents of the ROM and reprogram it. The contents of the EPROM cells can be erased using ultra violet light using EPROM programmer.
- Electrically Erasable Programmable Read Only Memory (EEPROM): - In this type of ROM, the contents of the cells can be erased electrically, by applying a high voltage. The EEPROM need not to be removed physically for reprogramming.

Secondary Memory: - Secondary memory is external and permanent in nature. The secondary memory is concerned with magnetic memory. Secondary memory can be stored on storage media like floppy disks, magnetic disks, magnetic tapes. This memory can also be stored optically on Optical disks - CD- ROM. The following are the popular secondary memory:

- **Magnetic Tape**: Magnetic tapes are used for large computers like mainframe computers where large volume of data is stored for a longer time. In PC 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.
- **Magnetic Disk**: 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 computer drive. Data is stored on both the surface of the disk. For Example- Floppy Disk.
- **Optical Disk**: With every new application and software there is greater demand for memory capacity. It is the necessity to store large volume of data that has led to the development of optical disk storage medium. Example Compact Disk/ Read Only Memory (CD- ROM)

Central Processing Unit (CPU): - The main unit inside the computer is the CPU. This unit is responsible for all events inside the computer. It controls all internal and external devices. The Control Unit and the Arithmetic Logic Unit of a computer system are jointly known as the Central Processing Unit (CPU). The CPU is the brain of any computer system. In a human body, all major decisions are taken by the brain and the other parts of the body function as directed by the brain. Similarly, in a computer system, all major calculations and comparisons are made inside the CPU and the CPU is also responsible for activating and controlling the operations of other units of a computer system.

Arithmetic Logical Unit (ALU): - (ALU) of a computer system is the place where the actual execution of the instructions take place during the processing operations. All arithmetic operation such as addition, subtraction, multiplication and division are performed and all logical operation like AND, OR, NOT (decisions) are made in the ALU. ALU has small amount of special memory unit called registers.

Control Unit: - This is the unit that supervises the flow of information between various unit. The control unit retrieve the instructions using registers one by one from the program which is store in the memory.

1.3 History of Computers

The term "generation of computers" refers to the different phases or stages in the development of computer technology. Each generation is characterized by specific technological advancements, improvements in performance, and changes in design architecture. As of my last knowledge update in January 2022, there have been five generations of computers. Here is a brief overview:

First Generation (1940s- 1950s):

Key Technologies: Vacuum tubes were used for electronic components.
Characteristics: Large in size, consumed a lot of power, generated a significant amount of heat.
Examples: ENIAC (Electronic Numerical Integrator and Computer), UNIVAC I.

Second Generation (1950s- 1960s):
Key Technologies: Transistors replaced vacuum tubes, magnetic core memory was introduced.
Characteristics: Smaller, more reliable, and consumed less power compared to first-generation computers.
Examples: IBM 1401, IBM 7090 series.

Third Generation (1960s- 1970s):
Key Technologies: Integrated circuits (ICs) were introduced, leading to a significant reduction in size and power consumption.
Characteristics: Smaller, faster, and more reliable than second- generation computers.
Examples: IBM System/360, DEC PDP- 11.

Fourth Generation (1970s- 1980s):
Key Technologies: Microprocessors, VLSI (Very Large Scale Integration) chips.
Characteristics: Personal computers (PCs) became available, marking the beginning of widespread use of computers in homes and businesses.
Examples: IBM PC, Apple II, Commodore 64.

Fifth Generation (1980s- Present):
Key Technologies: Parallel processing, artificial intelligence (AI), and the development of supercomputers.
Characteristics: Advancements in parallel computing, AI, and integration of multiple technologies.
Examples: IBM Watson, Deep Blue, modern supercomputers.

It's important to note that these generations are not rigid, and the transition from one generation to another is not always well- defined. Moreover, the computer industry continues to evolve, and new technologies are constantly emerging, shaping the future of computing.

1.4 Classification of modern computer according to their processing capacity:

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

A very large and expensive computer capable of supporting hundreds or even thousands of users simultaneously. In the hierarchy computers mainframes are just below supercomputers.
Note: The chief difference between a supercomputer and a mainframe is that mainframes are more powerful than supercomputers because they support more simultaneous programs. But supercomputers can execute a single program faster than a mainframe.

Mini Computer
A minicomputer is a multiprocessing system capable of supporting from 4 to about 200 users simultaneously.

Micro Computer

- **Desktop Computer**: a personal or micro- mini computer sufficient to fit on a desk.

- **Laptop Computer**: a portable computer complete with an integrated screen and keyboard. It is generally smaller in size than a desktop computer and larger than a notebook computer.

- **Palmtop Computer/Digital Diary /Notebook /PDAs**: a hand- sized computer. Palmtops have no keyboard but the screen serves both as an input and output device.

1.5 Computer Organization:

- Computer organization deals with the way in which the hardware components of a computer system are interconnected and how they operate together to execute instructions.
- It focuses on the low- level details of a computer system, including the design of the central processing unit (CPU), memory systems, input/output (I/O) systems, and other hardware components.
- Topics in computer organization include instruction set architecture (ISA), processor design, memory hierarchy, bus structures, and peripheral devices.
- It is concerned with the physical components and their arrangement, as well as how data flows between these components.

1.6 Computer Architecture:

- Computer architecture refers to the conceptual structure and logical behaviour of a computer system as seen by the software, as well as the interface between hardware and software.

- It deals with the high- level design principles used in defining the overall structure and functionality of a computer system.
- Computer architecture includes the instruction set architecture (ISA), which defines the instructions that a processor can execute, as well as the organization of the CPU, memory, and I/O systems.

1.7 Compare and contrast between Computer Organization and Architecture

The difference lies, precisely, between the “What” and the “How”.

Computer Architecture is the “What”.

Computer Organization is the “How”.

Computer Architecture tells you what the system does. So, knowing about the architecture is basically knowing what functionalities will your system display. What you can expect to get out of it.

Computer Organization tells you how exactly all units in your system have been arranged and interconnected to help realize the architectural goals your system claims to have achieved.

Computer Architecture	Computer Organization
To combine the parts to create a computer system that performs at the required level.	Design of functional building blocks and parts used in the construction of computer systems
Refers to the characteristics of the system that a programmer can see or access.	Used to describe the computer's operating units.
These characteristics directly affect how logically a program executes.	How are these operating units interconnected to one other?
Instruction set, Data Types, Memory Addressing Modes, and I/O Mechanism.	Control signal, interface between memory & peripherals.
Related to software	Related to hardware

1.8 Von Neumann Architecture

In 1946, Von- Neumann and his colleagues invented a new stored program computer at Institute of Advanced Studies (IAS). This computer is also known as IAS computer.
The concept included storing a program in the memory and reading commands from it. Additionally, he suggested a design that divided the system into the ALU, control, input, output, and memory components. This stored- program concept has 3 principles

i. Program and data can be stored in the same memory.
ii. The computer executes the program in sequence as directed by the instruction.
iii. A program can modify itself when it executes.

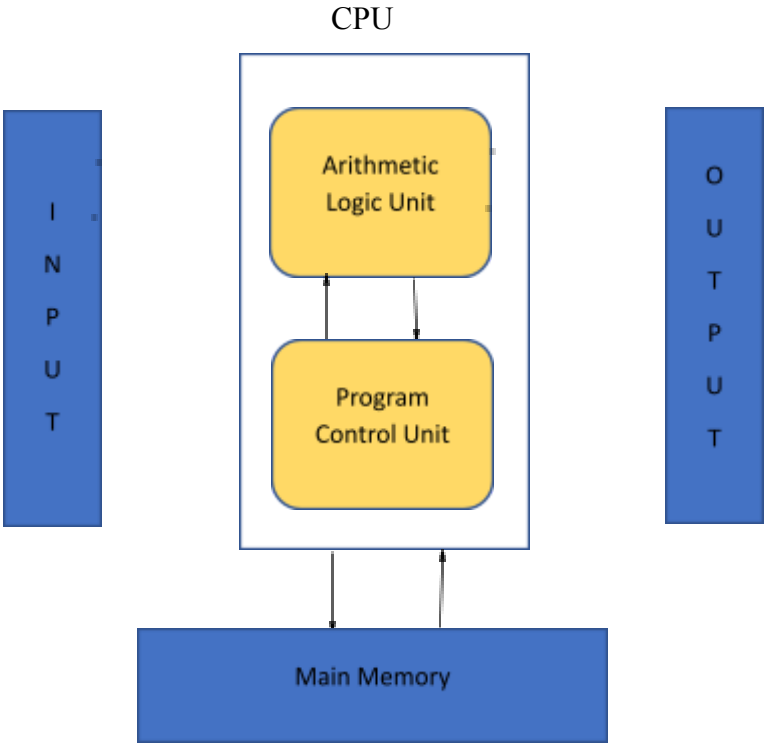


Fig 2: General Structure of Von Neumann Architecture

The CPU of IAS computer consists of 2 parts Arithmetic Logic Unit/ Data Processing Unit (ALU) and Program Control Unit (PCU). It also contains many registers to store data and instruction temporarily.

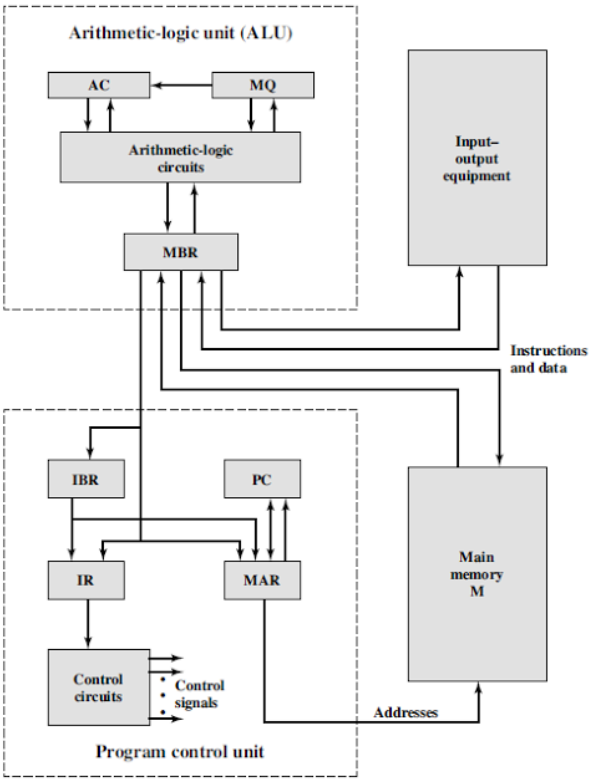


Fig 3: Detailed Structure of Von Neumann Architecture

The **arithmetic- Logic circuit** performs the actual task specified by the instruction.

This unit contains two registers **accumulator(AC) and multiplier Quotient (MQ)** to store the operand and result temporarily. Any word transfer can take place between

Memory Buffer Register(MBR) and the main memory.

PCU contains Control circuit that is responsible for fetching instruction, decoding instruction, fetching operand and providing control signal for all activity. Two instructions are fetched from the main memory to PCU. The instruction that is executed currently will be placed in **Instruction Register(IR).** The instruction that is not executed immediately will be placed in **Instruction Buffer Register(IBR**). **Program Counter(PC)** holds the address of the next instruction to be executed. Address of the current instruction is stored in the **Memory Address Register(MAR).**

Von- Neumann Bottleneck

The CPU has to wait longer due to the obtain the data from the main memory because of the speed difference between main memory and CPU registers. The CPU registers are very fast and internal to the CPU. The CPU- memory speed disparity is known as **Von Neumann bottleneck**.

This performance problem is solved by placing a high- speed memory called cache memory in between the CPU and the main memory. The speed of the cache memory is almost same as the CPU. Thus, the speed mismatch problem is solved.

1.9 Harvard Architecture

Harvard architecture is a computer architecture design that separates memory into two separate areas for program instructions and data. In contrast to von Neumann architecture, which uses a single memory space for both instructions and data, Harvard architecture has separate memory spaces for instructions and data, allowing for simultaneous access to both.

Key features of Harvard architecture include:

Separate Memory Spaces: Harvard architecture has physically separate memory for instructions and data. This means that the CPU can access instructions and data simultaneously, leading to potentially faster performance compared to von Neumann architecture, where the CPU has to alternate between fetching instructions and accessing data.

Parallelism: Because the instruction and data memories are separate, Harvard architecture allows for parallelism in accessing instructions and data, which can lead to improved performance in certain applications, particularly in embedded systems and digital signal processing where high throughput is required.

Specialized Instruction and Data Buses: Harvard architecture often employs separate buses for instructions and data, further enhancing the parallelism and potentially improving performance.

Complexity: While Harvard architecture offers potential performance benefits, it can also introduce complexity, especially in terms of memory management and programming. Unlike von Neumann architecture, where data and instructions can be freely interchanged, Harvard architecture requires careful management of separate instruction and data spaces.

Common Applications: Harvard architecture is commonly used in embedded systems, micro controllers, digital signal processors (DSPs), and other specialized computing devices where performance and real- time processing are critical.

Overall, Harvard architecture offers advantages in terms of performance and parallelism, particularly in certain application domains, but it also comes with its own set of challenges and complexities in terms of system design and programming.

Difference between Von Neumann and Harvard Architecture

Memory Structure:

Von Neumann Architecture: In von Neumann architecture, there is a single memory space that is used for both data and instructions. This means that instructions and data are stored together in the same memory module, and the CPU accesses both using a single bus.
Harvard Architecture: In Harvard architecture, there are separate memory spaces for data and instructions. This means that there are physically distinct memory modules for storing instructions and data, and typically separate buses are used for accessing instructions and data.

Instruction Fetching:

Von Neumann Architecture: In von Neumann architecture, the CPU fetches instructions and data from the same memory space. This means that there can be potential bottlenecks when fetching both instructions and data simultaneously, as the CPU may need to alternate between fetching instructions and fetching data.
Harvard Architecture: In Harvard architecture, since instructions and data are stored in separate memory spaces, the CPU can fetch instructions and data simultaneously. This can lead to potential performance improvements, especially in scenarios where the CPU needs to access both instructions and data frequently and concurrently.

Parallelism:

Von Neumann Architecture: Von Neumann architecture typically lacks inherent parallelism in memory access, as the CPU has to share a single memory bus for fetching both instructions and data.
Harvard Architecture: Harvard architecture inherently supports parallelism in memory access, as separate buses are used for fetching instructions and data. This allows the CPU to access instructions and data simultaneously, potentially improving performance, especially in scenarios where parallel access to instructions and data is beneficial.

Examples:

Von Neumann Architecture: Traditional desktop and laptop computers, as well as most general- purpose microprocessors, typically follow von Neumann architecture.

Harvard Architecture: Harvard architecture is commonly used in embedded systems, micro controllers, digital signal processors (DSPs), and other specialized computing devices where performance and real- time processing are critical.

Review Questions

Multiple Choice Questions

1. What is the full form of CPU?
   a) Computer Processing Unit
   b) Computer Principle Unit
   c) Central Processing Unit
   d) Control Processing Unit

2. Which of the following is the brain of the computer?
   a) Central Processing Unit
   b) Memory
   c) Arithmetic and Logic unit
   d) Control unit

3. Which of the following unit is responsible for converting the data received from the user into a computer understandable format?
   a) Output Unit
   b) Input Unit
   c) Memory Unit
   d) Arithmetic & Logic Unit

4. Which of the following part of a processor contains the hardware necessary to perform all the operations required by a computer?
   a) Controller
   b) Registers
   c) Cache
   d) Data path
5. Which of the following is designed to control the operations of a computer?
   a) User
   b) Application Software
   c) System Software
   d) Utility Software

6. Which of the following are physical devices of a computer?
   a) Hardware
   b) Software
   c) System Software
   d) Package

7. Which of the following computers are lower than mainframe computers in terms of speed and storage capacity?
   a) Mainframes
   b) Hybrid
   c) Mini
   d) Super

8. What is the full form of PROM
   a) Program read- only memory
   b) Primary read- only memory
   c) Programmable read- only memory
   d) Program read- output memory

9. Which of the following is an input device used to enter motion data in computers or other electronic devices
   a) Monitor
   b) Trackball
   c) Plotter
   d) Joystick

10. In the context of computing, a byte is equal to _____ bits
    a) 4
    b) 16
    c) 24
    d) 8

Long Type Questions

1. Define computer architecture and computer Organization.
2. Compare and contrast between computer architecture and computer Organization.
3. Define a digital computer with neat block diagram.
4. Classify computer System according to their processing capacity.
5. Discuss Generation of computer with significant hardware development of each generation.
6. What is stored program architecture.
7. Discuss IAS computer.
8. What is Von Neumann Bottleneck and how is it solved?
9. Write down the features of Harvard architecture.
10. Compare and contrast between Von Neumann architecture and Harvard Architecture.

Chapter 2

Data Representation and Binary Arithmetic

2.1 Introduction

Data representation is the process of organizing, storing, and modifying data within a computer's memory and processor units. Binary numbers, often known as bits, are the fundamental unit of data representation utilized by computers. A bit can either represent a 0 or 1.

A computer data representation must include the following fundamental:

Binary Representation: Computers use binary digits, or zeros and ones, to represent all data. We group eight bits together and it is known as bytes, and they are used to represent binary numbers.

Numerical Representation: To express numbers, one can use fixed- point, floating-point, or integer forms, among other formats. It's usual practice to represent numbers in binary (base- 2) notation.

2.2 Number System

A number system is defined as the representation of numbers by using digits or other symbols in a consistent manner. The value of any digit in a number can be determined by a digit, its position in the number, and the base of the number system. The number system can be classified in two types

1. **Positional Number System**: - A positional number system is also known as weighted number system. As the name implies there will be a weight associated with each digit. In general, a positional number is expressed as:

$$d_{m-1}d_{m-2}d_{m-3}...d_2d_1d_0 \bullet d_{-1}d_{-2}...d_{-n}$$

Where d_{m-1} is referred to as the most significant digit (MSD) and d_{-n} as the least significant digit (LSD). Each digit position has an associated weight bi where b is called the base or radix. The point in the middle is referred to as a radix point and is

used to separate the integer and fractional part of a number. Integer part is in the left side of the radix point; fraction part is in the right side of the radix point. Few examples of positional number system are,

- Decimal Number System (Base/Radix 10).
- Binary Number System (Base/Radix 2).
- Octal Number System (Base/Radix 8).
- Hexadecimal Number System (Base/Radix 16).

2. **Non- Positional Number System**: - Non- positional number system is also known as non- weighted number system. Digit value is independent of its position. Non- positional number system is used for shift position encodes and error detecting purpose. Few examples of non- weighted number system are

- Gray Code.
- Excess- 3 Code.

Decimal Number System

The word decimal originated from a Latin word "Decem", that means ten. This number system has 10 unique symbols

0 1 2 3 4 5 6 7 8 9

The numbers $(2453)_{10} \neq (3452)_{10}$

$$(2453)_{10} = (2*10^3) + (4*10^2) + (5*10^1) + (3*10^0)$$
$$= (2*1000) + (4*100) + (5*10) + (3*1)$$
$$= 2000 + 400 + 50 + 3$$
$$= 2453$$

$$(3452)_{10} = (3*10^3) + (4*10^2) + (5*10^1) + (2*10^0)$$
$$= (3*1000) + (4*100) + (5*10) + (2*1)$$
$$= 3000 + 400 + 50 + 2$$
$$= 3452$$

Binary Number System

The word Binary originated from a Latin word "Binarius", that means two. This number system has 2 unique symbols 0 1. Each digit is known as Bit in this number system. The group of 8 bits is known as Byte.

The numbers $(1010)_2 \neq (1100)_2$

Decimal – Binary Equivalent

Decimal	Binary
0	000

1	001
2	010
3	011
4	100
5	101
6	110
7	111

Binary – Decimal Conversion

$$(100101)_2 = (?)_{10}$$
$$= (1*2^5) + (0*2^4) + (0*2^3) + (1*2^2) + (0*2^1) + (1*2^0)$$
$$= (1*32) + 0 + 0 + (1*4) + 0 + 1$$
$$= 32 + 4 + 1$$
$$= (37)_{10}$$

$$(1100100)_2 = (?)_{10}$$
$$= (1*2^6) + (1*2^5) + (0*2^4) + (0*2^3) + (1*2^2) + (0*2^1) + (0*2^0)$$
$$= (1*64) + (1*32) + 0 + 0 + (1*4) + 0 + 0$$
$$= 64 + 32 + 4 + 1$$
$$= (101)_{10}$$

Decimal – Binary Conversion

$(34)_{10} = (?)_2$

34 / 2 = 17 with remainder 0

17 / 2 = 8 with remainder 1

8 / 2 = 4 with remainder 0

4 / 2 = 2 with remainder 0

2 / 2 = 1 with remainder 0

1 / 2 = 0 with remainder 1

Read the reminder from bottom to top

$(34)_{10} = (100010)_2$

$(55)_{10} = (?)_2$

55 / 2 = 27 with remainder 1

27 / 2 = 13 with remainder 1

13 / 2 = 6 with remainder 1

6 / 2 = 3 with remainder 0

3 / 2 = 1 with remainder 1

1 / 2 = 0 with remainder 1

Read the reminder from bottom to top

$(55)_{10} = (110111)_2$

Hexadecimal Number System

The word Hexadecimal is combination of two words Hexa and Decimal. Hexa is a Greek word that means "**six**" and decimal is originated from a Latin word "**Decem",** that means **"ten"**. So Hexadecimal means 6+10=16. This number system has 16 unique symbols

0 1 2 3 4 5 6 7 8 9 A B C

D E F

Decimal –Hexadecimal Equivalence

Decimal	Hexadecimal
0	0
1	1
2	2
3	3

4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Hexadecimal – Decimal Conversion

$$(25)_{16} = (?)_{10}$$
$$= (2*16^1) + (5*16^0)$$
$$= (2*16) + (5*1)$$
$$= 32 + 5$$
$$= (37)_{10}$$

$$(2A5)_{16} = (?)_{10}$$
$$= (2*16^2) + (A*16^1) + (5*16^0)$$
$$= (2*256) + (10*16) + (5*1)$$
$$= 512 + 160 + 5$$
$$= (677)_{10}$$

Decimal – Hexadecimal Conversion

$(1228)_{16}= (?)_{10}$

1228 / 16 = 76 with remainder 12(C)

76 / 16 = 4 with remainder 12(C)

4 / 16 = 0 with remainder 4

$(1228)_{16}= (4CC)_{10}$

$(600)_{16}= (?)_{10}$

600 / 16 = 37 with remainder 8

37 / 16 = 2 with remainder 5

2 / 16 = 0 with remainder 2

$(600)_{16}= (852)_{10}$

Binary – Hexadecimal Conversion

$16^1 = 2^4$

- 1 hexadecimal digit corresponds to 4 binary digits
  - Divide the binary number into group of 4 digits from right side.
  - Then convert the 4 digits into corresponding hexadecimal.
  - If digit count in binary number not a multiple of 4 then pad with zeros on left

$(1010000100111101)_2 = (?)_{16}$

= 1010 0001 0011 1101

A 1 3 D

$(1010000100111101)_2 = (A13D)_{16}$

$(10001110001111)_2 = (?)_{16}$

Extra padded zeros

= **0010** 0011 1000 1111

2 3 8 F

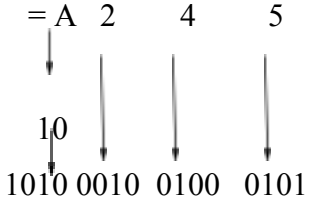
$(1010000100111101)_2 = (238F)_{16}$

Hexadecimal – Binary Conversion

- **Expand the hexadecimal number into 4 digit binary equivalent.**

$(A245)_{16} = (?)_2$

= A 2 4 5



1010 0010 0100 0101

$(A245)_{16} = (1010001001000101)_2$

Octal – Decimal Conversion

$$(37)_8 = (?)_{10}$$
$$= (3*8^1) + (7*8^0)$$
$$= (3*8) + (7*1)$$
$$= 24 + 7$$
$$= (31)_{10}$$

$$(674)_8 = (?)_{10}$$
$$= (6*8^2) + (7*8^1) + (4*8^0)$$
$$= (6*64) + (7*8) + (4*1)$$
$$= 384 + 56 + 4$$
$$= (444)_{10}$$

Decimal - Octal Conversion

$(52)_{10} = (?)_8$

52 / 8 = 6 with remainder 4

6 / 8 = 0 with remainder 6

$(52)_{10} = (64)_8$

$(127)_{10} = (?)_8$

127 / 8 = 15 with remainder 7

15 / 8 = 1 with remainder 7

1 / 8 = 0 with remainder 1

$(127)_{10} = (177)_8$

Binary – Octal Conversion

$\mathbf{8^1 = 2^3}$

- 1 octal digit corresponds to 3 binary digits
  - Divide the binary number into group of 3 digits from right side.
  - Then convert the 3 digits into corresponding octal.
  - If digits count in binary number not a multiple of 3 then pad with zeros on left.

$(101000010011110)_2 = (?)_8$

= 101 000 010 011 110

↓ 5 ↓ 0 ↓ 2 ↓ 3 ↓ 6

$(101000010011110)_2 = (50236)_{16}$

$(10001110001111)_2 = (?)_8$

Extra padded zeros

= 010 001 110 001 111

↓ 2 ↓ 1 ↓ 6 ↓ 1 ↓ 7

$(1010000100111101)_2 = (21617)_8$

Octal – Binary Conversion

- Expand the hexadecimal number into 3 digit binary equivalent.

$(7245)_8 = (?)_2$

= 7 2 4 5

↓ ↓ ↓ ↓

111 010 100 101

$(7245)_8 = (111010100101)_2$

$(637)_{16} = (?)_2$

= 6 3 7

↓ ↓ ↓

110 011 111

$(637)_8 = (110011111)_2$

2.3 Complements

In digital computers to simplify the subtraction operation & for logical manipulation complements
are used. There are two types of complements used in each radix system.

1. The radix complements or r' s complement
2. The diminished radix complements or (r- 1)' s complement

Decimal Number System: we have two types of complement

1. **(r- 1)' s complement i.e. 9's complement**: - 9's complement of a number is obtained by subtracting all bits from 9
    - 9's complement of 231 is (999- 231) = 768
    - 9's complement of 456 is (9999- 1456) = 8543
2. **r's complement i.e. 10's complement**: - By obtaining the 9's comp of the given no. & then adding 1 we get the 10's complement.

10's complement = 9's complement + 1

- 10's complement of 231 is (999- 231) = 768+1=769
- 10's complement of 456 is (9999- 1456) = 8543+1= 8544

Binary Number System: we have two types of complement

1. **(r- 1)' s complement i.e. 1's complement**: - we obtain the 1's complement of the given no. by changing all 0's to 1's & 1's to 0's.
    - 1's complement of 10010 is 01101
    - 1's complement of 1100110 is 0011001

2. **r's complement i.e. 2's complement**: - we obtain the 2's complement in 2 ways

i. By obtaining the 1‘s complement of the given no. & then adding 1 we get the 1’s complement.

ii. Starting at the LSB, copying down each bit up to & including the first 1 bit encountered, and complementing the remaining bits.

- **Method 1**: 2’s complement of 10010

  1’s complement of 10010 is 01101

  01101

  \+ 1

  01110

- **Method 2: -** 2’s complement of 10010

  Original number 10010 - - - Copy up to bit position where first 1 occurs then complements all digits 01110.

2’s Complement Subtraction

Rules of 2’s complement subtraction.

1. In the first step, find the 2's complement of the subtrahend.
2. Add the complement number with the minuend.
3. If we get the carry by adding both the numbers, then we discard this carry and the result is positive.
4. If no carry found then take 2's complement of the result and put a negative sign before.

Evaluate:

(i) 110110 - 10110

Solution:

The numbers of bits in the subtrahend is 5 while that of minuend is 6. We make the number of bits in the subtrahend equal to that of minuend by taking a `0’ in the sixth place of the subtrahend.

Now, 2’s complement of subtrahend 010110 is (101101 + 1) i.e.101010. Add this with the minuend.

1 1 0 1 1 0 Minuend

1 0 1 0 1 0 2's complement of subtrahend

Carry over 1 1 0 0 0 0 0 Result of addition

After dropping the carryover we get the result of subtraction to be 100000.

(ii) 10110 – 11010

Solution:

2's complement of 11010 is (00101 + 1) i.e. 00110. Hence

1 0 1 1 0 - Minuend

0 0 1 1 0 - 2's complement of subtrahend -

Result of addition - 1 1 1 0 0

As there is no carry over, the result of subtraction is negative and is obtained by writing the 2's complement of 11100 i.e.(00011 + 1) or 00100.

Hence the difference is – 00100.

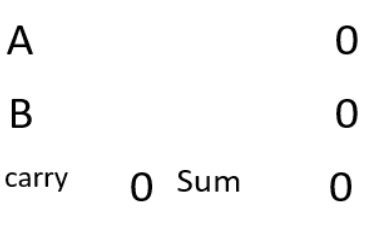
2.4 Binary Arithmetic

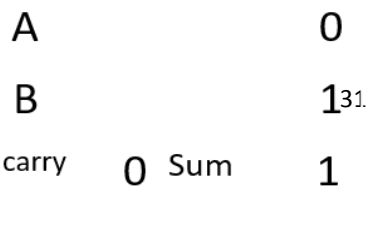
Binary arithmetic is an essential part of various digital systems. We can add, subtract, multiply, and divide binary numbers using various methods.

Binary Addition

There are four rules for binary addition:

Input A	Input B	Sum (S) A+B	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1





A			1
B			0
Carry	0	Sum	1

A			1
B			1
carry	1	Sum	0

Example of Binary Addition

- 10001 + 11101 = 101110:

```
    1           1
    1  0  0  0  1
+   1  1  1  0  1
-------------------
1   0  1  1  1  0
```

- 1110 + 1111 = 11101:

```
    1  1  1
    1  1  1  0
+   1  1  1  1
-------------------
1   1  1  0  1
```

Binary Subtraction

There are four rules for binary subtraction**:**

Input A	Input B	Subtract (S) A-B	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

A 0

B 0

Borrow 0 Result 0

A 0

B 1

Borrow 1 Result 1

A 1

B 0

Borrow 0 Result 1

A 1

B 1

Borrow 0 Result 0

Example of Binary Subtraction

1 1 - - - - - 3
1 0 - - - - - 2
0 1 - - - - - 1

1(borrow)
1 0 1 0 - - - - - - 10
0 1 1 0 - - - - - - 6
0 1 0 0 - - - - - - 4

Binary Multiplication

There are four rules for binary multiplication:

Input A	Input B	Multiply (M) AxB
0	0	0
0	1	0
1	0	0
1	1	1

Example of Binary Multiplication

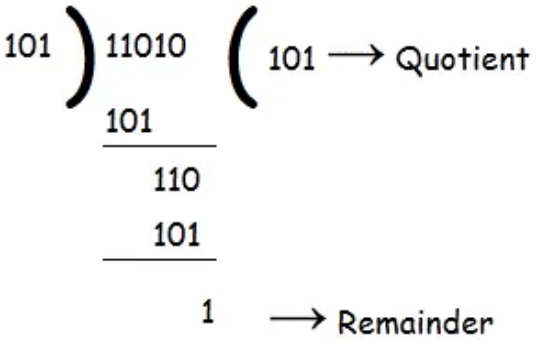
```
    1001
  x 101
  ------
    1001
   0000X
+ 1001XX
  ------
  101101
```

Binary Division

There are four parts in any division: Dividend, Divisor, quotient, and remainder.

Input A	Input B	Divide (D) A/B
0	0	Not defined
0	1	0
1	0	Not defined
1	1	1

Example of Binary Division



Review Questions
Multiple Choice Questions

1. What is the decimal equivalent of the binary number 1011?

   a) 7
   b) 9
   c) 11
   d) 13

2. Which of the following is a base- 8 number system?

   a) Binary
   b) Octal
   c) Decimal
   d) Hexadecimal

3. In the hexadecimal number system, what is the value represented by the digit "A"?

   a) 10
   b) 11
   c) 12
   d) 13

4. What is the binary representation of the decimal number 25?

   a) 11001
   b) 11010
   c) 11100
   d) 11101

5. Which of the following is true regarding the two's complement representation of negative numbers?

a) The most significant bit (MSB) is always 0 for positive numbers and 1 for negative numbers.
b) The least significant bit (LSB) is always 0 for positive numbers and 1 for negative numbers.
c) The two's complement representation cannot represent negative numbers.
d) The two's complement representation is the same as the sign- magnitude representation.

6. What is the result of adding the binary numbers 1101 and 1010?

a) 10011
b) 10111
c) 1111
d) 11011

7. Which of the following operations is not valid in binary arithmetic?

a) Division
b) Multiplication
c) Addition
d) Subtraction

8. What is the result of subtracting the binary number 1011 from the binary number 1101 using 2's complement subtraction?

a) 0010
b) 0100
c) 0110
d) 1000

9. Which number system uses the digits 0 through 9 and the letters A through F?

a) Binary
b) Decimal
c) Octal
d) Hexadecimal

10. What is the decimal equivalent of the binary number 1111?

e) 8
f) 15
g) 16
h) 111

11. What is the base (radix) of the hexadecimal number system?

a) 8
b) 10
c) 16
d) 2

12. In the binary system, what is the value of 1011?

a) 8
b) 10
c) 11
d) 13

13. Which of the following is a valid octal number?

a) 238
b) 789
c) 101
d) 1302

14. What is the decimal equivalent of the hexadecimal number A3?

a) 163
b) 165
c) 173
d) 1632

15. In the binary system, what is the result of the addition: 1101 + 1011?

a) 10110
b) 11010
c) 10010
d) 11100

16. What is the result of subtracting the binary number 101 from 110 using 2's complement?

a) 10
b) 11
c) 100
d) 1101

17. Which number system uses the digits 0- 7?

a) Binary
b) Decimal
c) Octal
d) Hexadecimal

18. What is the value of the binary number 1101101 in decimal?

a) 45
b) 77
c) 109
d) 93

19.What is the product of the binary numbers 110 and 101?

a) 110
b) 1110
c) 10110
d) 10010

20.What is the result of dividing the binary number 1011 by 3 (in binary)?

a) 11
b) 110
c) 10
d) 101

Long Type Questions

1. What is a number system, and why are they important in computing?
2. How many digits are in the binary number system? What are they?
3. Explain the concept of a "radix" or "base" in a number system.
4. What is the decimal equivalent of the binary number 1010?
5. Convert the decimal number 45 into binary.
6. What are the advantages of using hexadecimal notation compared to binary notation?
7. Describe the relationship between octal and binary number systems.
8. How is a negative number represented in binary using the two's complement method?
9. Explain the significance of the term "bit" in the context of binary numbers.

10. Discuss the role of number systems in digital data representation and communication.
11. Perform the addition of the binary numbers 1101 and 1010.
12. Subtract the binary number 1010 from the binary number 1101.
13. Multiply the binary numbers 1101 and 1010.
14. Divide the binary number 1101 by the binary number 10.
15. Convert the decimal number 25 into binary, then perform the addition of the binary numbers 1101 and the binary representation of 25.
16. Convert the decimal number 18 into binary, then subtract the binary representation of 18 from the binary number 1101.
17. Multiply the binary numbers 101 and 11.
18. Divide the binary number 11010 by the binary number 101.
19. Subtract the binary number 1011 from the binary number 1101 using 2's complement subtraction.
20. Perform the subtraction of the binary number 10010 from the binary number 11001 using 2's complement subtraction.

Chapter 3

Computer Instruction Set

3.1 Introduction

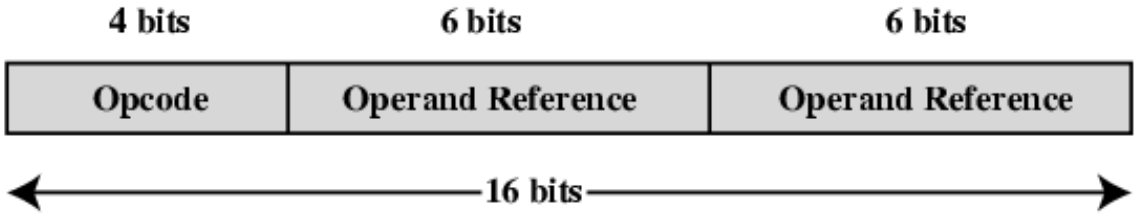
Computer instructions are a set of commands or operations that a computer's central processing unit (CPU) can execute. These instructions are part of machine code, which is the lowest- level programming language understood by a computer. Each instruction corresponds to a specific operation that the CPU can perform, such as arithmetic operations, data movement, control flow, and input/output operations. We can compare an instruction with a word in processors language and complete instruction set with vocabulary.

A group of bits that tell the computer to perform a specific operation is called Instruction.

The complete collection of instructions that are understood by a CPU is known as Instruction Set.

Instruction format: Consider an instruction represented in this way

ADD A,B



The length of the instruction is 16 bit long. The op- code is 4 bits long. There can be 2^4=16 at most op- code is possible. Address of each operands are 6 bits each.

Elements of an Instruction:

Operation code (Op code)

Specifies the operation to be performed (e.g., ADD, I/O).The operation is specified by a binary code, known as the operation code, or op- code.

Source Operand reference

The operation may involve one or more source operands, that is, operands that are inputs for the operation.

Result Operand reference

The operation may produce a result.

This instruction has a 4- bit opcode, so here we can represent at most 16 instructions. All these instructions together are known as the instruction set of that system. Let us assume the instruction set as

Op Code	Instruction
0000	AND memory word
0001	ADD memory word
0010	SUB memory word
0011	MUL memory word
0100	DIV memory word
0101	Complement
0110	Branch conditional
0111	Branch un conditional
1000	Skip next instruction if AC is positive
1001	Skip next instruction if AC is negative
1010	Skip next instruction if AC is zero
1011	Increment AC
1100	Decrement AC
1101	Shift right
1110	Shift left
1111	Halt

3.3 Instruction Mode

Addressing modes in computer architecture refer to the methods used to specify the operand(s) of an instruction. An operand is typically a data value or a memory address that an instruction operates on. Different addressing modes provide flexibility and efficiency in specifying operands for instructions. Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced).

TYPES OF ADDRESSING MODES

Implied Mode

Address of the operands are specified implicitly
in the definition of the instruction

- No need to specify address in the instruction

- EA = AC, or EA = Stack[SP], EA: Effective Address.

Immediate Mode

Instead of specifying the address of the operand,
operand itself is specified

- No need to specify address in the instruction
- However, operand itself needs to be specified
- Sometimes, require more bits than the address
- Fast to acquire an operand

Register Mode

Address specified in the instruction is the register address

- Designated operand need to be in a register
- Shorter address than the memory address
- Saving address field in the instruction
- Faster to acquire an operand than the memory addressing
- EA = IR(R) (IR(R): Register field of IR)

Register Indirect Mode

Instruction specifies a register which contains
the memory address of the operand

- Saving instruction bits since register address is shorter than the memory address
- Slower to acquire an operand than both the register addressing or memory addressing
- EA = [IR(R)] ([x]: Content of x)

Auto- increment or Auto- decrement features:

Same as the Register Indirect, but:

- When the address in the register is used to access memory, the value in the register is incremented or decremented by 1 (after or before the execution of the instruction)

Direct Address Mode

Instruction specifies the memory address which
can be used directly to the physical memory

- Faster than the other memory addressing modes
- Too many bits are needed to specify the address for a large physical memory space
- EA = IR(address), (IR(address): address field of IR)

Indirect Addressing Mode

- The address field of an instruction specifies the address of a memory location that contains the address of the operand
- When the abbreviated address is used, large physical memory can be addressed with a relatively small number of bits
- Slow to acquire an operand because of an additional memory access
- EA = M[IR(address)]

Relative Addressing Modes

- The Address fields of an instruction specifies the part of the address (abbreviated address) which can be used along with a designated register to calculate the address of the operand

PC Relative Addressing Mode(R = PC)

- EA = PC + IR(address)
- Address field of the instruction is short
- Large physical memory can be accessed with a small number of address bits

Indexed Addressing Mode

XR: Index Register:

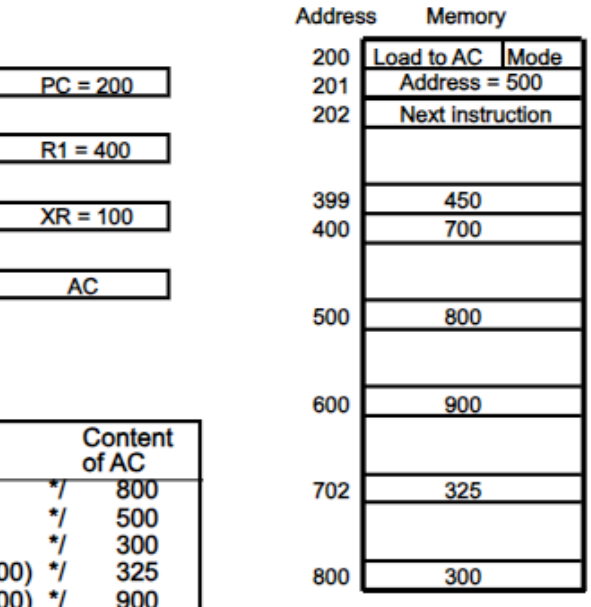
- EA = XR + IR(address)

Base Register Addressing Mode

BAR: Base Address Register:

- EA = BAR + IR(address)

ADDRESSING MODES – EXAMPLES



Addressing Mode	Effective Address			Content of AC
Direct address	500	/* AC ← (500)	*/	800
Immediate operand	-	/* AC ← 500	*/	500
Indirect address	800	/* AC ← ((500))	*/	300
Relative address	702	/* AC ← (PC+500)	*/	325
Indexed address	600	/* AC ← (XR+500)	*/	900
Register	-	/* AC ← R1	*/	400
Register indirect	400	/* AC ← (R1)	*/	700
Autoincrement	400	/* AC ← (R1)+	*/	700
Autodecrement	399	/* AC ← -(R1)	*/	450

3.3 Computer Register

In computer architecture, a register is a small, high- speed storage area within the CPU (central processing unit) that temporarily holds data that the CPU is currently processing. Registers are an integral part of the CPU's architecture and are used to store data, addresses, and intermediate results during the execution of instructions.

Here are some key characteristics of computer registers:

Speed: Registers are the fastest storage elements in a computer system, with access times measured in nanoseconds or even picoseconds. This speed allows the CPU to quickly access and manipulate data during instruction execution.

S**ize**: Registers are typically small in size compared to main memory. Common register sizes are 8, 16, 32, or 64 bits, depending on the CPU architecture.

Purpose: Registers serve various purposes within the CPU. Some registers hold data operands for arithmetic and logical operations, while others hold memory addresses, instruction pointers, or status flags.

Data Storage: Registers can store various types of data, including integers, floating-point numbers, memory addresses, and control information.

Operand Access: Instructions executed by the CPU often specify operands that reside in registers. These operands can be manipulated directly within the registers, improving computational efficiency compared to accessing data from main memory.

Control and Status: Registers also hold control and status information used by the CPU to manage the execution of instructions and to track the state of the processor, such as flags indicating arithmetic overflow, zero results, or interrupts.

Common types of registers found in CPU architectures include:

General- Purpose Registers: Used to hold data operands and intermediate results during instruction execution. These registers can be accessed by the programmer for general computation purposes.

Examples of general- purpose registers commonly found in CPU architectures include:

R0- Rn: A set of registers numbered sequentially (e.g., R0, R1, R2, ..., Rn) used for general computation tasks.

EAX, EBX, ECX, EDX: Commonly used register names in x86 architecture, where each register can be accessed as a 32- bit or 16- bit register.

R1- R31 in MIPS architecture: MIPS architecture typically provides 32 general-purpose registers, named R1 through R31, with R0 often reserved for the constant value 0.

Special- Purpose Registers: Serve specific functions within the CPU, such as the instruction pointer (IP), stack pointer (SP), program counter (PC), and status registers containing flags indicating the outcome of arithmetic and logical operations. Special-purpose registers (SPRs) are a type of register found in CPU architectures that serve specific functions within the CPU. Unlike general- purpose registers, which are used for a wide range of computational tasks, special- purpose registers are dedicated to particular functionalities critical for the operation of the CPU and the execution of

instructions. These registers typically hold control information, status flags, memory addresses, and pointers, among other specialized data. Special- purpose registers are designed to perform specific functions critical for CPU operation, such as managing control flow, handling interrupts, and tracking the state of the processor.

Examples of special- purpose registers commonly found in CPU architectures include:

Program Counter (PC) or Instruction Pointer (IP): Holds the memory address of the next instruction to be fetched and executed.

Stack Pointer (SP) or Base Pointer (BP): Points to the top of the stack or the base of the stack frame, respectively, used for managing function calls, local variables, and parameter passing.

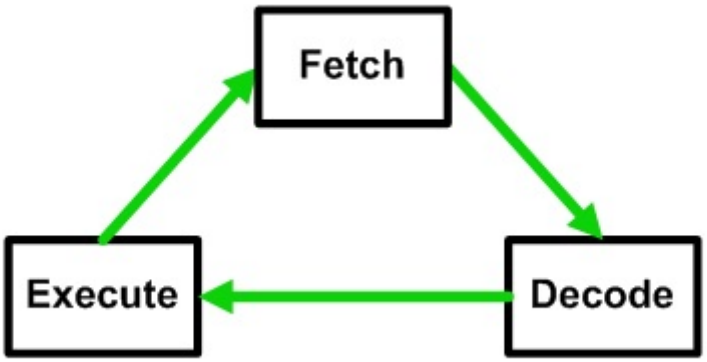
Status Registers: Hold status flags indicating various conditions and outcomes during instruction execution, such as zero, carry, overflow, and interrupt enable/disable flags.

Control Registers: Used for configuring and controlling CPU operation, memory management, and system- level features, such as page tables, memory protection, and cache control.

Interrupt Registers: Hold information related to interrupt handling, including interrupt vectors, priority levels, and interrupt enable/disable settings.

3.4 Instruction Cycle

The instruction cycle, also known as the fetch- decode- execute cycle, is the fundamental process by which a CPU (Central Processing Unit) carries out instructions in a computer program. It consists of several stages that are repeated for each instruction being executed. Here's a breakdown of the instruction cycle:



Fetch:
The CPU fetches the next instruction from memory.

The address of the next instruction to be fetched is typically stored in a special register called the Program Counter (PC) or Instruction Pointer (IP).
The CPU sends a request to memory to retrieve the instruction located at the memory address specified by the program counter.
The fetched instruction is stored in a special register called the Instruction Register (IR).

Decode:
The CPU decodes the instruction fetched in the previous step.
Decoding involves determining the operation code (opcode) of the instruction, which specifies the operation to be performed, as well as identifying the operands and addressing mode.

Execute:
The CPU performs the operation specified by the decoded instruction.
This stage involves executing arithmetic or logical operations, accessing memory, or modifying the control flow of the program.
Depending on the instruction and the CPU architecture, the execution phase may involve multiple sub- stages.

Write Back (Optional):
In some cases, particularly for instructions that produce results, there may be a write-back stage. This stage involves writing the result of the executed instruction back to a register or memory location if necessary.

After completing one cycle for an instruction, the CPU proceeds to the next cycle to fetch, decode, and execute the next instruction. This process continues iteratively, allowing the CPU to execute instructions sequentially and perform the tasks required by the program.

3.4 Instruction Set Architecture (ISA)

Instruction Set Architecture (ISA) refers to the set of instructions that a processor is designed to execute, along with the organization of those instructions and the resources available to execute them. It defines the interface between the software and the hardware components of a computer system.

Here are key aspects of Instruction Set Architecture:

Instructions: ISA specifies the set of instructions that a processor can execute. These instructions define the operations that the processor can perform, such as arithmetic operations, data movement, control flow operations, and input/output operations.

Instruction Formats: ISA defines the formats of instructions, including the size and structure of opcode fields, operand fields, and addressing modes. Different instructions may have different formats depending on the type of operation and the addressing mode used.

Registers: ISA specifies the registers available in the processor and their roles. This includes general- purpose registers used for data manipulation, special- purpose registers for controlling CPU operation and managing system resources, and floating-point registers for floating- point arithmetic.

Data Types: ISA defines the data types supported by the processor, such as integers, floating- point numbers, characters, and Boolean values. It also specifies the sizes of data types and the representation of data in memory and registers.

Memory Model: ISA defines the memory model used by the processor, including the address space, memory organization, and addressing modes for accessing memory. It specifies how instructions interact with memory, such as loading data from memory, storing data to memory, and performing memory operations.

Instruction Execution: ISA defines the behavior of instructions and the execution semantics. This includes the order of execution of instructions, the handling of exceptions and interrupts, and the effects of instructions on processor state and program flow.

Privilege Levels: ISA may define multiple privilege levels or modes of operation, such as user mode and supervisor mode. Each privilege level has different access rights and privileges, with higher privilege levels having more control over system resources and functionality.

Instruction Set Extensions: ISA may support extensions or additional instruction sets that provide specialized functionality, such as multimedia instructions, cryptographic instructions, or vector processing instructions.

ISA serves as a contract between software and hardware, allowing software developers to write programs that can run on different processors with compatible ISAs. It also provides a framework for processor designers to design and implement processors that meet the requirements of specific applications and use cases. Examples of ISAs include x86, ARM, MIPS, PowerPC, and RISC- V.

3.5 Classification of ISA:

The major classifications are

- stack architecture,
- accumulator- based architecture
- register based architecture.

Types of Architecture	Source Operand	Destination
Stack	Top two elements in the stack	Top of stack
Accumulator	Accumulator (1) Memory(other)	Accumulator
Register Set	Register or memory	Register or memory

Stack architecture

A stack- based Instruction Set Architecture (ISA) is a type of architecture where instructions operate on data stored in a stack rather than using explicit operands and registers. In a stack- based ISA, operands are implicitly pushed onto and popped off of a stack, and instructions operate on the top elements of the stack. Operands are implicitly on the top of the stack.

Accumulator based architecture

An accumulator- based Instruction Set Architecture (ISA) is a type of computer architecture where the accumulator register plays a central role in arithmetic and logic operations. In this architecture, most arithmetic and logical operations involve one operand being implicitly the accumulator register. The result of the operation is typically stored back into the accumulator, replacing its previous value. Other operands can be taken from memory or other registers. The accumulator register is a special-purpose register used to hold one of the operands for arithmetic and logical operations. Arithmetic and logical operations usually involve the accumulator and another operand, with the result stored back into the accumulator. Early computers, such as the Manchester Baby and the EDSAC, used accumulator- based architectures. Some early microprocessors, like the Intel 4004, also had accumulator- based designs.

Register based architecture

Register- based Instruction Set Architecture (ISA) is a type of computer architecture where instructions operate directly on data stored in registers. Registers are high- speed storage elements located within the CPU that hold data temporarily during instruction execution. In a register- based ISA, instructions typically specify explicit operands as register addresses, and the results of operations are stored back into registers. Many modern processors, including x86, ARM, MIPS, and PowerPC architectures, are based on register- based ISAs.

Comparisons of Architecture

Consider the operation

C = A + B

Stack	Accumulator	Register-Memory	Register-Register
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R1, B	Load R2, B
Add	Store C	Store C, R1	Add R3, R1, R2
Pop C			Store C, R3

3.6 Instruction Length

The length of an instruction in a computer architecture depends on various factors, including the instruction set architecture (ISA), the complexity of the instructions, and the design goals of the processor. There are 3 types of instructions

Three- Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
ADD   R1, A, B        /*  R1 ¬ M[A] + M[B]      */
ADD   R2, C, D        /*  R2 ¬ M[C] + M[D]      */
MUL   X, R1, R2       /*  M[X] ¬ R1 * R2        */
```

- Results in short programs
- Instruction becomes long (many bits)

Two- Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
MOV   R1, A           /* R1 ¬ M[A]          */
ADD   R1, B           /* R1 ¬ R1 + M[B]     */
MOV   R2, C           /* R2 ¬ M[C]          */
ADD   R2, D           /* R2 ¬ R2 + M[D]     */
MUL   R1, R2          /* R1 ¬ R1 * R2       */
MOV   X, R1           /* M[X] ¬ R1          */
```

One- Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
LOAD      A       /*  AC ← M[A]        */
ADD       B       /* AC ← AC + M[B]    */
STORE     T       /*  M[T] ← AC        */
LOAD      C       /*  AC ← M[C]        */
ADD       D       /*  AC ← AC + M[D]*/
MUL       T       /*  AC ← AC * M[T]*/
```

```
        STORE     X       /* M[X] ← AC      */
```

Zero- Address Instructions:

Program to evaluate X = (A + B) * (C + D)

```
        PUSH  A   /* TOS ← A    */
        PUSH  B   /* TOS ← B    */
        ADD       /* TOS ← (A + B)      */
        PUSH  C   /* TOS ← C    */
        PUSH  D   /* TOS ← D    */
        ADD       /* TOS ← (C + D)      */
        MUL       /* TOS ← (C + D) * (A + B)  */
        POP   X   /* M[X] ← TOS         */
```

3.7 RISC and CISC

RISC (Reduced Instruction Set Computer) and CISC (Complex Instruction Set Computer) are two contrasting approaches to computer architecture, each with its own set of design principles and characteristics. Here's a comparison between RISC and CISC architectures:

Instruction Set Complexity:

RISC: RISC architectures have a simplified instruction set with a small number of basic instructions. Each instruction typically performs a single operation, and the instructions are of uniform length. RISC architectures often prioritize simplicity and efficiency.

CISC: CISC architectures have a larger and more complex instruction set with instructions that can perform multiple operations or access memory multiple times in a single instruction. CISC architectures aim to provide more powerful and expressive instructions to reduce the number of instructions needed to perform a task.

Instruction Execution Philosophy:

RISC: RISC architectures follow the principle of "simplicity favors regularity." They emphasize the use of simple, uniform instructions that can be executed quickly and predictably. RISC processors often use pipelining and other techniques to achieve high instruction throughput.

CISC: CISC architectures follow the principle of "complexity is hidden in hardware." They aim to provide higher- level abstractions and more powerful instructions that can perform complex tasks in a single instruction. CISC processors often have microcode or hardware implementations that translate complex instructions into simpler micro-operations.

Memory Access:

RISC: RISC architectures typically use a load/store architecture, where only specific load and store instructions can access memory, while other instructions operate only on register operands. This simplifies instruction decoding and pipeline design.

CISC: CISC architectures often support memory- to- memory operations, where instructions can operate directly on memory operands without loading them into registers first. This allows for more flexible and expressive instructions but can complicate instruction decoding and pipeline design.

Performance Trade- offs:

RISC: RISC architectures tend to prioritize simplicity, regularity, and pipelining, which can lead to higher instruction throughput and better performance in many cases. RISC architectures are often favored in embedded systems, high- performance computing, and mobile devices.

CISC: CISC architectures aim to provide more powerful and expressive instructions that can reduce the number of instructions needed to perform a task. However, the complexity of CISC architectures can sometimes lead to longer instruction execution times and lower performance compared to RISC architectures.

Examples:

RISC: Examples of RISC architectures include ARM, MIPS, PowerPC, and RISC- V.

CISC: Examples of CISC architectures include x86 (e.g., Intel and AMD processors), Motorola 68k, and DEC VAX

Review Questions

1) Which part of the CPU is responsible for fetching instructions from memory?
    a) Arithmetic Logic Unit (ALU)
    b) Control Unit
    c) Register File
    d) Cache Memory
2) In computer architecture, the term "ISA" stands for:
    a) Instructional Set Architecture
    b) Internal System Architecture
    c) Integrated Software Application
    d) International Standards Agreement
3) Which addressing mode is commonly used in stack- based architectures?
    a) Immediate
    b) Direct
    c) Indirect
    d) Implicit

4) In a register- based Instruction Set Architecture (ISA), instructions typically specify explicit operands using:
    a) Memory addresses
    b) Opcode fields
    c) Instruction pointers
    d) Register addresses

5) Which type of instruction set architecture typically has a larger and more complex instruction set?
    a) RISC
    b) CISC
    c) Stack- based
    d) Register- based
6) Instructions in a 2- address instruction set architecture explicitly specify:
    a) One operand and the operation code
    b) Two operands and the operation code
    c) Three operands and the operation code
    d) The operation code only
7) Which of the following is a characteristic of Reduced Instruction Set Computer (RISC) architectures?
    a) Large and complex instruction set
    b) Emphasis on simplicity and uniformity
    c) Memory- memory instruction format
    d) Implicit operands in instructions
8) What is the primary role of the accumulator register in accumulator- based architectures?
    a) Hold memory addresses
    b) Store instruction opcodes
    c) Perform arithmetic and logic operations
    d) Manage control flow instructions
9) In computer architecture, the length of an instruction can be influenced by all of the following factors EXCEPT:
    a) Operand size
    b) Opcode size
    c) Number of execution units
    d) Addressing modes supported
10) Which instruction cycle stage involves determining the operation code and addressing mode of the fetched instruction?
    a) Fetch
    b) Decode
    c) Execute
    d) Write Back
11) Which addressing mode involves specifying the memory address directly in the instruction?

    a) Immediate addressing mode

    b) Direct addressing mode

    c) Indirect addressing mode

    d) Indexed addressing mode

12) In which addressing mode is the operand value itself contained within the instruction?

a) Register addressing mode

b) Immediate addressing mode

c) Indirect addressing mode

d) Base addressing mode

13) Which addressing mode uses a constant value added to the contents of a register to determine the effective address?

a) Indexed addressing mode

b) Base addressing mode

c) Register addressing mode

d) Indirect addressing mode

14) An addressing mode that involves specifying the memory address indirectly using a pointer stored in a register is called:

a) Indexed addressing mode

b) Base addressing mode

c) Indirect addressing mode

d) Register addressing mode

15) Which addressing mode calculates the effective address by adding a displacement to the contents of the base register?

a) Base addressing mode

b) Indexed addressing mode

c) Indirect addressing mode

d) Relative addressing mode

16) In which addressing mode does the effective address of the operand depend on the value in a register specified in the instruction?

a) Direct addressing mode

b) Indirect addressing mode

c) Register addressing mode

d) Indexed addressing mode

17) Which addressing mode is commonly used for array access, where the address is calculated based on an index and a base address?

a) Direct addressing mode

b) Indirect addressing mode

c) Indexed addressing mode

d) Base addressing mode

18) An addressing mode that allows specifying the operand using a combination of a base address and an offset is called:

a) Indirect addressing mode

b) Relative addressing mode

c) Indexed addressing mode

d) Base addressing mode

19) Which addressing mode uses a memory location to indirectly specify the operand's address?

a) Base addressing mode

b) Indirect addressing mode

c) Indexed addressing mode

d) Register addressing mode

20) In which addressing mode is the operand's address calculated based on the contents of the program counter (PC) or instruction pointer (IP)?

a) Immediate addressing mode

b) Relative addressing mode

c) Indexed addressing mode

d) Direct addressing mode

Long Type Questions

1 Define Instruction Set Architecture (ISA) and explain its significance in computer architecture.
2 Differentiate between RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing) architectures. Provide examples of each.
3 Explain the concept of addressing modes in the context of instruction set architecture. Give examples of different addressing modes and describe how they are used.
4 Describe the fetch- decode- execute cycle in the context of instruction execution. How does the ISA influence this process?
5 What is the role of the program counter (PC) in instruction execution? Explain its function within the context of the fetch- decode- execute cycle.
6 Discuss the importance of instruction formats in ISA design. Provide examples of different instruction formats and explain their characteristics.
7 Explain the difference between immediate and register operands in instruction execution. How does the ISA accommodate these different types of operands?
8 Describe the concept of instruction pipelining and its impact on processor performance. How does the ISA influence the implementation of instruction pipelining?
9 Discuss the role of control transfer instructions in program execution. Provide examples of control transfer instructions and explain how they affect program flow.

10 Explain the difference between micro- programming and hardwired control in the context of processor design. How does the choice of control unit implementation affect the ISA.

11 Define addressing mode in the context of computer architecture. Explain why addressing modes are important for instruction execution.

12 Describe the direct addressing mode. Provide an example instruction and explain how it operates. Discuss the advantages and limitations of direct addressing.

13 Explain the concept of register addressing mode. Give an example instruction that utilizes register addressing and describe its operation. Discuss the benefits of register addressing in instruction execution.

14 Discuss the differences between immediate and indirect addressing modes. Provide examples of instructions for each addressing mode and explain how they differ in their operand specification.

15 Describe the indexed addressing mode. How does it work, and what are its advantages? Provide an example instruction that uses indexed addressing and explain its operation.

16 Explain the concept of base- relative addressing mode. Give an example instruction that employs base- relative addressing and discuss its significance in memory management.

17 Discuss the concept of displacement in addressing modes. How is displacement used, and what role does it play in effective memory access?

18 Explain the concept of stack- based addressing mode. How is the stack used in this addressing mode, and what are its advantages in certain situations?

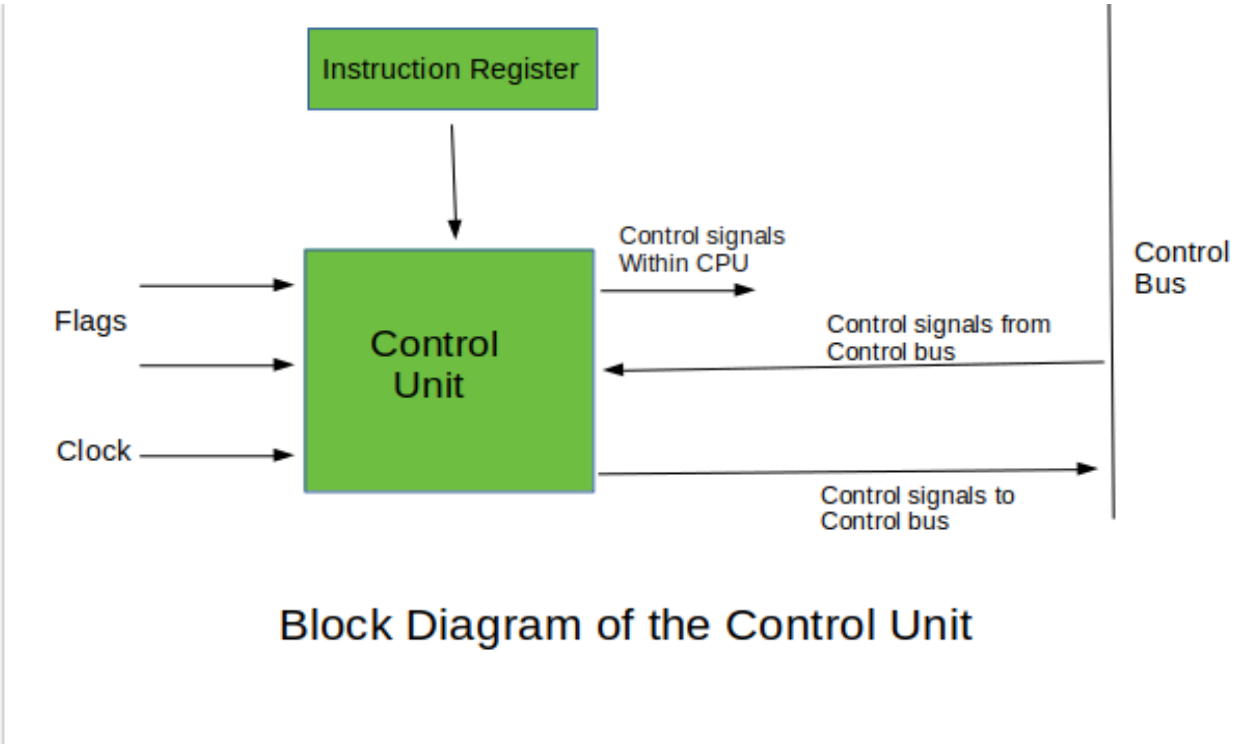
19 Describe the difference between absolute and relative addressing modes. Provide examples of instructions for each mode and discuss their respective uses.

20 Explain the concept of PC- relative addressing mode. How does it work, and what are its advantages in program execution? Provide an example instruction that utilizes PC- relative addressing.

Chapter 4: THE CONTROL UNIT

4.1 Introduction

The Control Unit (CU) is an essential part of the Central Processing Unit (CPU) that directs and coordinates all operations of a computer system. It serves as the brain of the CPU, making sure that instructions are executed in the right order and at the right time. Unlike the Arithmetic Logic Unit (ALU), the control unit does not carry out any calculations or logical operations. Instead, it creates control signals that manage the flow of data between the processor, memory, and input/output devices.



Block Diagram of the Control Unit

The main job of the control unit is to oversee the instruction cycle. This cycle involves fetching an instruction from memory, decoding it to figure out the action needed, and executing it by activating the right hardware components. During this process, the control unit interacts with registers, the ALU, and memory by sending timing and control signals. These signals define operations such as reading data from memory, writing data to registers, selecting ALU tasks, and managing data transfers within the CPU.

The control unit also keeps different components in sync using clock signals, ensuring that instructions are executed in an orderly way. Depending on its design, a control unit can be

either hardwired or micro programmed. In today's computer systems, the efficiency and performance of the processor largely rely on the control unit's design, making it a key element in computer organization and architecture.

4.2 Function of control unit

The Control Unit (CU) plays a central role in computer operation. It manages and coordinates all processor activities. It acts as a supervisory unit, ensuring instructions run in the correct order and at the right time. While it does not perform arithmetic or logical operations itself, the control unit enables these operations by directing other CPU components.

Instruction fetching is one of the primary functions of the control unit. It retrieves instructions from main memory using the address in the Program Counter (PC). After fetching, the control unit updates the PC to point to the next instruction. This process ensures a smooth and continuous flow of program execution.

Another important function is instruction decoding. Once an instruction is fetched, the control unit decodes it to determine the type of operation to be performed. During decoding, it identifies whether the instruction is an arithmetic, logical, data transfer, or control instruction and determines which registers, memory locations, or I/O devices are involved.

The control unit is also responsible for generating control signals. These signals activate specific hardware components such as registers, the ALU, memory, and I/O interfaces. For example, it generates signals to select the required ALU operation, enable registers, or initiate memory read and write operations.

Timing and sequencing of operations is another crucial function of the control unit. Using clock signals, it synchronizes all internal activities of the CPU so that each operation occurs in the correct sequence without conflicts. This coordination ensures reliable and error- free execution of instructions.

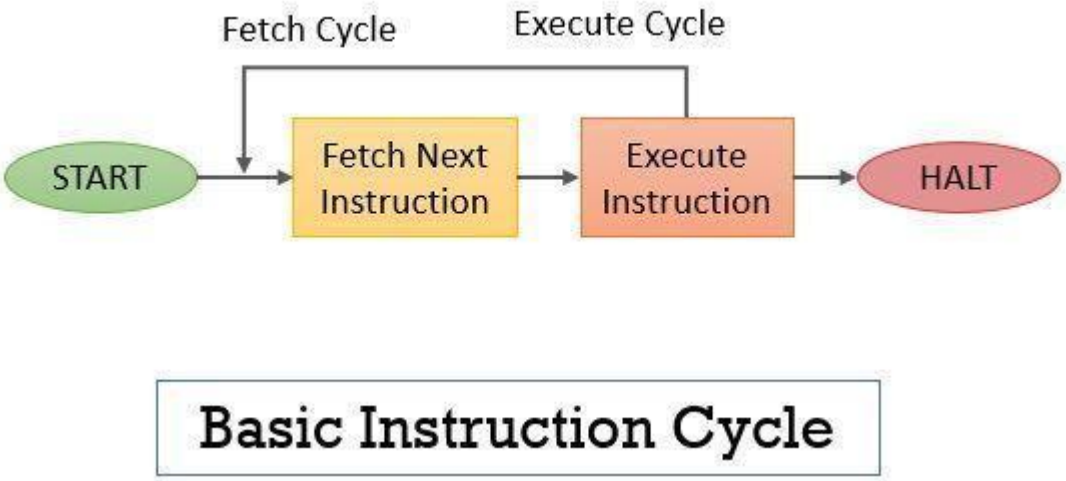
Additionally, the control unit manages data flow within the CPU by controlling the movement of data between registers, the ALU, and memory. It also handles interrupts by temporarily halting the current instruction execution and transferring control to the appropriate interrupt service routine.

4.3 Instruction Cycle

4.3.1 Definition of Instruction Cycle

The Instruction Cycle is the complete sequence of operations performed by the Control Unit to execute a single instruction in a computer system. Every instruction stored in memory follows this cycle to ensure correct and orderly execution. The instruction cycle begins when an instruction is fetched from memory and ends after the execution of that instruction is completed. This process is repeated continuously until the program terminates.

The instruction cycle is fundamental to computer organization because it explains how a program is executed internally by the CPU. It involves interaction between several CPU components, such as the Program Counter (PC), Instruction Register (IR), Arithmetic Logic Unit (ALU), registers, and memory. The control unit plays a crucial role by generating appropriate control signals at each stage of the cycle.



Basic Instruction Cycle

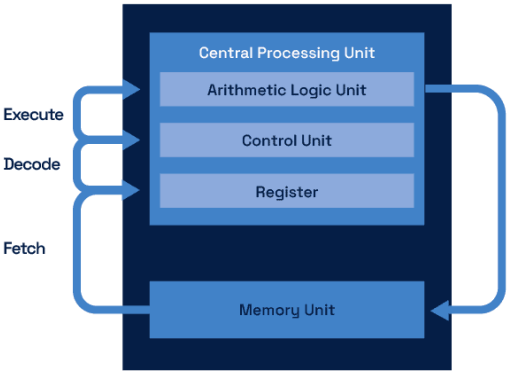
In simple terms, the instruction cycle answers three basic questions:

1. **What instruction to execute?**
2. **How to execute it?**
3. **Where to store the result?**

Consider a chef following a recipe. The chef first reads the recipe, then understands what needs to be done, and finally performs the cooking steps. Similarly, the CPU fetches, decodes, and executes instructions.

4.3.2 Fetch Cycle

The Fetch Cycle is the first phase of the instruction cycle. In this phase, the control unit retrieves the next instruction from main memory. The address of the instruction to be fetched is stored in the Program Counter (PC). This address is sent to memory, and the corresponding instruction is loaded into the Instruction Register (IR).



Once the instruction is fetched, the Program Counter is automatically incremented so that it points to the next instruction in sequence. This ensures continuous program execution. The fetch cycle is purely a data transfer operation and does not involve any computation.

Steps in Fetch Cycle:

1. PC sends the instruction address to the memory.
2. Instruction is fetched from memory.
3. Instruction is loaded into the IR.
4. PC is incremented

Fetching an instruction is like opening a book and reading the next line. Once read, you move your finger to the next line automatically.

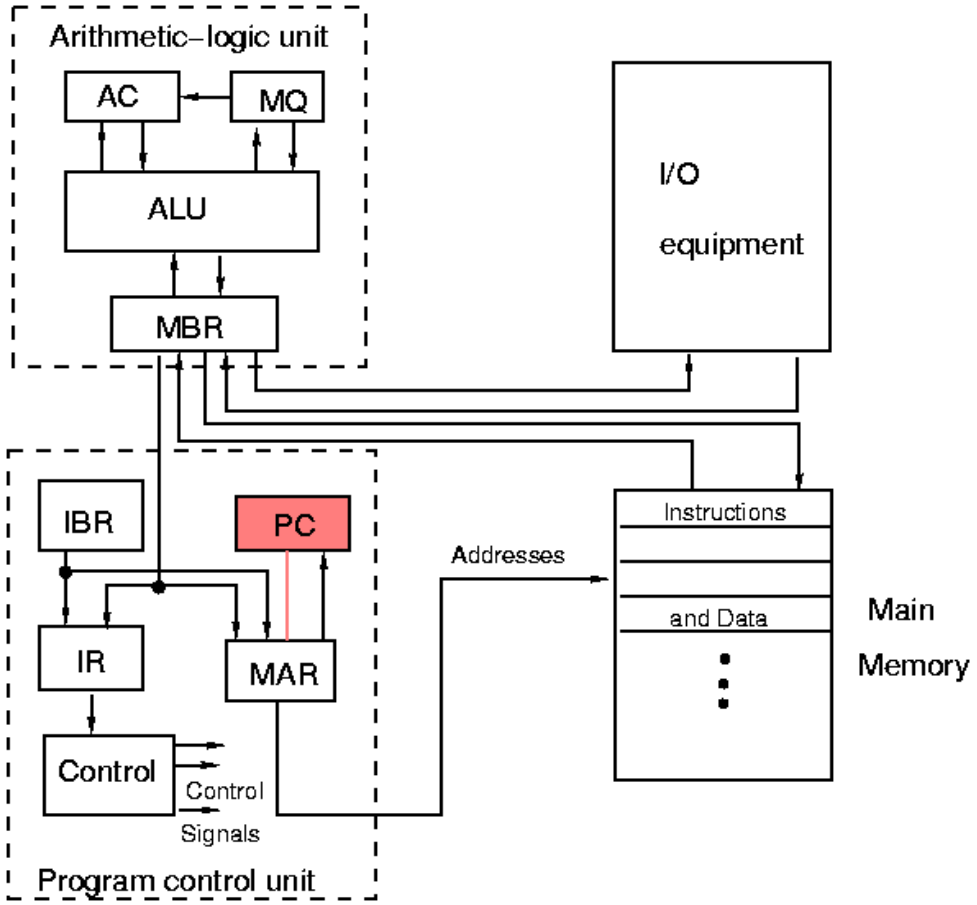
- **Example**

If the PC initially contains 200, and each instruction occupies 4 bytes, then after fetching:

New PC = 200 + 4 = 204

4.3.3 Decode Cycle

The Decode Cycle comes after the Fetch Cycle. In this step, the control unit decodes the instruction that is stored in the Instruction Register to figure out what needs to be done. The instruction is split into parts, like the opcode and the operand. The opcode specifies the operation (add, subtract, load, store), while the operand specifies the data or memory location involved.



The instruction decoder within the control unit interprets the opcode and generates the necessary control signals. These signals prepare the ALU, registers, and memory for execution. No actual data processing occurs in this phase; only decision- making takes place.

This phase is similar to actually performing an action after understanding instructions, such as turning the steering wheel after seeing a turn sign.

- **Example**

Instruction: ADD R1, R2
If R1 = 10 and R2 = 20
Result stored in R1 = 30

4.3.5 Interrupt Cycle

When an interrupt signal is produced by an I/O device or system event, the Interrupt Cycle takes place. An interrupt transfers control to an Interrupt Service Routine (ISR) and momentarily stops the regular execution of instructions. This enables the CPU to react fast to outside events.

The CPU picks up where it left off when the interrupt is handled. By enabling computation and I/O operations to overlap, interrupt handling increases CPU efficiency.

A phone call during study time interrupts your work. After attending the call, you resume studying from the same point.

4.4 Control Signals and Timing

4.4.1 Control Signals

The Control Unit (CU) produces control signals, which are electrical signals that are used to manage and coordinate the functions of the CPU's various components as well as the computer system as a whole. These signals specify what needs to be done, which part will do it, and when. The hardware parts of a computer would not be able to cooperate in a systematic way without control signals.

The control unit decodes the instruction and produces a particular set of control signals while the instruction is being executed. These signals manage data transfer between internal buses, control memory access, activate registers, and choose ALU operations. For instance, the control unit creates a Read signal to memory and permits the relevant register to receive data when a memory read instruction is carried out.

In general, control signals can be divided into:

- Signals for register control (load, enable, clear)
- ALU control signals (AND, OR, subtract, and add)
- Signals for memory control (read, write)
- Signals for I/O control

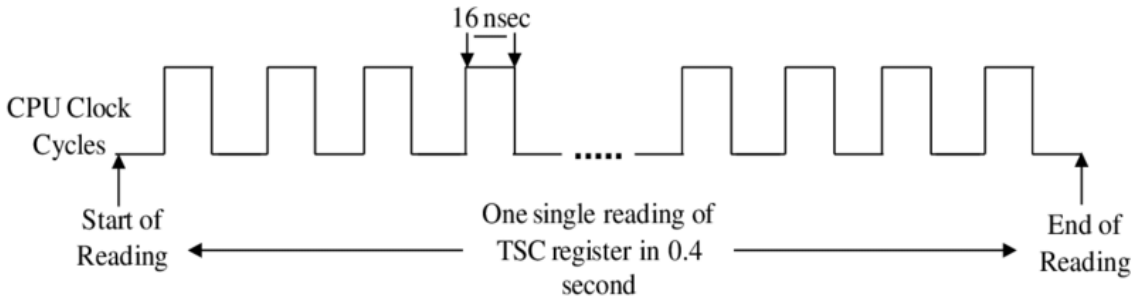
Control signals are like **traffic signals at an intersection**. Each signal tells vehicles when to stop, move, or turn, ensuring smooth and safe traffic flow.

🔹 Example

If an instruction requires loading data from memory into Register R1:

- Control signals generated:
    - Memory Read = 1
    - Register R1 Enable = 1
    - ALU Operation = None

4.4.2 Timing Signals and Clock



Timing signals make sure that every CPU operation happens at the right time and in the right order. The system clock, which produces regular pulses at predetermined intervals, is the source of these signals. A clock cycle is a unit of time that is represented by each clock pulse.

Clock pulses are used by the control unit to synchronize tasks like data transfer, instruction fetching, decoding, and execution. One or more clock cycles are allotted to each instruction cycle step, guaranteeing systematic execution free from component conflicts.

The CPU's speed is determined by the clock frequency. More operations can be completed in a second with a higher clock frequency. Higher frequency, however, also results in higher heat production and power consumption.

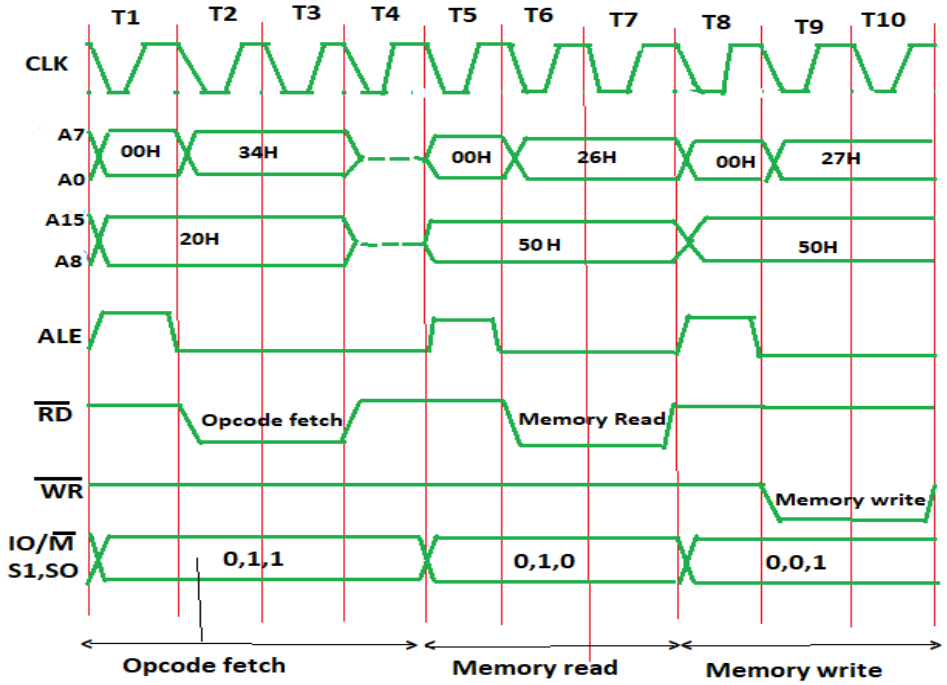
Timing signals are similar to a **school bell system**. Each bell indicates when a class starts or ends, ensuring that all activities occur on schedule.

🔹 **Example**

If a CPU has a clock frequency of **2 GHz**:

- Clock cycles per second = 2×10^9
- Time per clock cycle = $1 / (2 \times 10^9)$ = **0.5 ns**

4.4.3 Timing Diagram and Sequencing of Operations



The way control signals change over time while an instruction is being executed is shown graphically in a timing diagram. It illustrates how control signals like register enable, memory read/write, and ALU operation signals relate to clock pulses. Timing diagrams are helpful for comprehending operation overlap and sequencing.

The fetch cycle may take place in the first clock cycle, decoding in the second, and execution in the third and fourth cycles of a typical instruction execution. To guarantee proper sequencing, the control unit activates various control signals at various clock edges.

A timing diagram is like a **train timetable**, showing which train arrives or departs at a particular time.

- **Example**

If an instruction requires:

- Fetch: 1 cycle
- Decode: 1 cycle
- Execute: 2 cycles

Total clock cycles = **4 cycles**

If clock frequency = 1 GHz,
Execution time = 4 × 1 ns = **4 ns**

4.5 Hardwired Control Unit

In a Hardwired Control Unit (HCU), fixed logic circuits like logic gates, flip- flops, decoders, and counters are used to generate the control signals needed for the execution of instructions. The physical wiring of these components determines the behavior of the control unit in this design, where the control logic is implemented directly in hardware.

Hardwired control units do not employ control memory or micro- instructions, in contrast to microprogrammed control units. Rather, they use timing signals, status flags, and combinational and sequential logic to generate control signals based on the current instruction. Hardwired control units are fast because of their hardware- based implementation, which makes them appropriate for processors where speed is crucial.

A hardwired control unit is like a fixed electrical switchboard. Each switch is permanently wired to perform a specific action, and changing the behavior requires rewiring the system.

4.5.2 Working of Hardwired Control Unit

A hardwired control unit operates by combining timing signals with opcode decoding. Decoders are used to decode the opcode when an instruction is fetched into the Instruction Register. A logic circuit that produces the necessary control signals receives the decoder's output, clock pulses, and status signals.

A particular pattern of control signals is associated with each instruction. These signals are produced at specific intervals to synchronize memory access, ALU operations, and data transfer.

Simple Steps:

- Decoding of the instruction opcode
- Clocks and counters are used to generate timing signals.
- Opcode and timing signals are combined in logic circuits.
- CPU components receive control signals.

🔹 **Example**

If an instruction requires:

- 1 cycle for fetch
- 1 cycle for decode
- 2 cycles for execute

Total cycles = **4 cycles**

At 1 GHz clock:
Execution time = 4 × 1 ns = **4 ns**

Multiple choice questions :

From Bits to Brilliance - Mastering Computer Organization and Architecture

1. The control unit of a computer:

A) Performs arithmetic operations B) Stores data permanently
C) Controls and coordinates all operations D) Displays output

2. Which register holds the address of the next instruction?

A) IR B) PC
C) MAR D) MDR

3. The instruction cycle consists of:

A) Fetch and store B) Fetch, decode, execute
C) Input and output D) Read and write

4. Which phase interprets the instruction?

A) Fetch B) Decode
C) Execute D) Interrupt

5. Control signals are generated by:

A) ALU B) Control Unit
C) Memory D) Registers

6. The system clock is used for:

A) Storing data B) Synchronizing operations
C) Performing calculations D) Input operations

7. Hardwired control unit is:

A) Flexible B) Slow
C) Fast but inflexible D) Software- based

8. Which component decodes instructions?

A) ALU B) Decoder
C) Cache D) Bus

9. Interrupt cycle is used for:

A) Data storage B) Handling external signals
C) Arithmetic operations D) Memory allocation

10. Which is faster?

A) Microprogrammed CU B) Hardwired CU
C) Both same D) None

Short Answer Questions

1. Define the Control Unit of a computer.
2. State any two functions of the control unit.
3. What is meant by the instruction cycle?

4. List the main phases of the instruction cycle.
5. What is the role of the Program Counter (PC) during instruction execution?
6. What is an Instruction Register (IR)?
7. Define control signals.
8. What is the purpose of the system clock in a CPU?
9. What is a hardwired control unit?
10. Mention two advantages of a hardwired control unit.

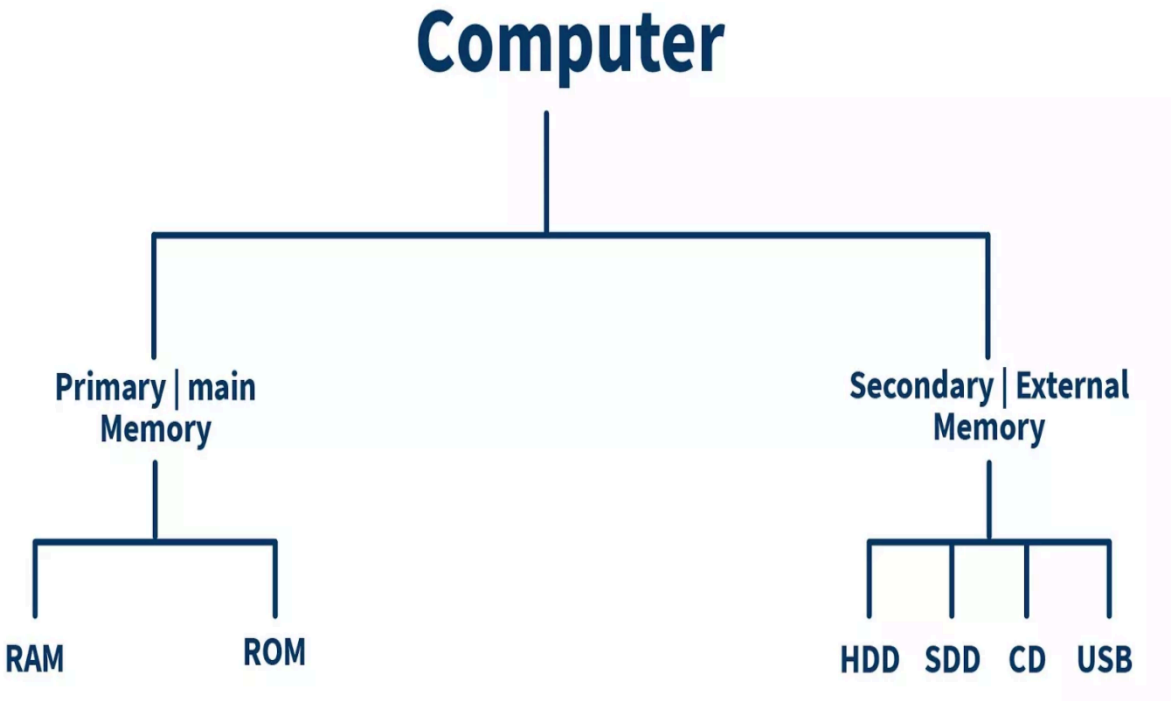
Long Answer Questions

1. Explain the Control Unit in detail with a neat block diagram.
2. Describe the functions of the control unit with suitable examples.
3. Explain the instruction cycle with all its phases and a labeled diagram.
4. Discuss the fetch, decode, and execute cycles in detail.
5. Explain control signals and timing with the help of a timing diagram.
6. Describe the working of a hardwired control unit with a block diagram.
7. Discuss the advantages and disadvantages of the hardwired control unit.
8. Explain the role of the system clock in coordinating CPU operations.
9. Compare hardwired control unit and microprogrammed control unit.
10. Write short notes on:
    a) Instruction Decoder
    b) Timing and Control Logic

Chapter 5: Memory Organization

5.1 Introduction to Memory Organization:

The structured placement of various memory units in a computer system to achieve effective data storage, quick access, and affordability is referred to as memory organization. A single type of memory is insufficient because a computer system needs to store a lot of data and instructions while also giving the CPU quick access. As a result, memory is divided into several levels, each with unique characteristics related to speed, capacity, and cost.



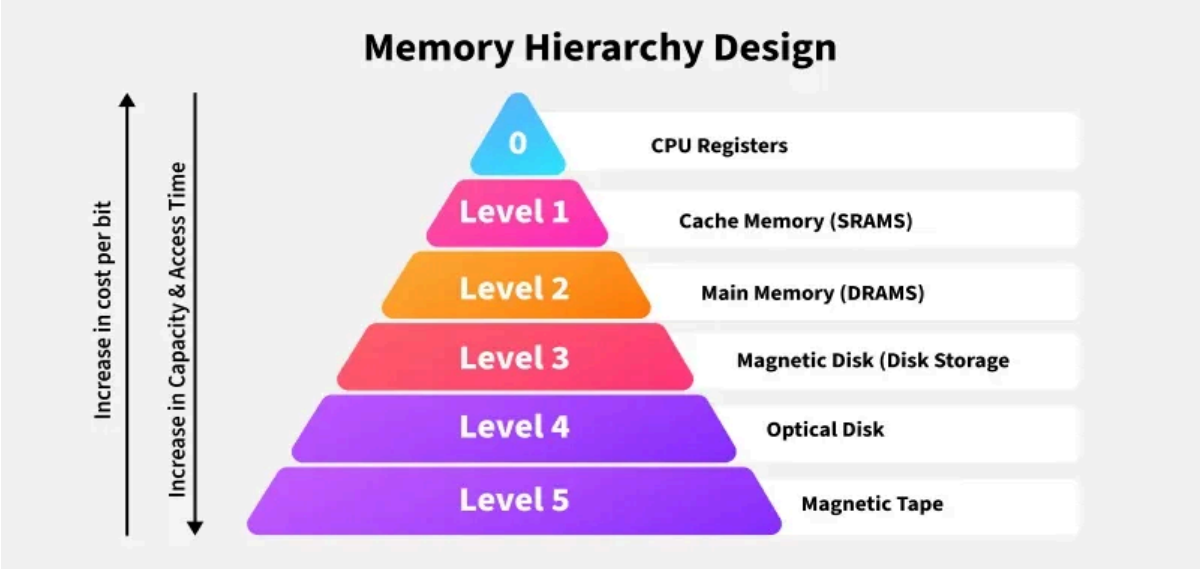
The main goal of memory organization is to deliver the necessary data to the processor at the optimal time and speed while maintaining a reasonable overall system cost. While slower memory technologies are less expensive and provide more storage, faster memory technologies are more costly and have a smaller capacity. These are balanced by memory organization.

Memory in a typical computer system includes main memory and secondary storage, which are slower but have much larger capacities, as well as registers and cache memory, which are very fast but small in size. Frequently used data is stored in faster memory units to enhance performance because the CPU can only access a small amount of memory at a time.

Effective memory management increases system throughput, decreases CPU idle time, and boosts overall performance. In contemporary computer systems, ideas like virtual memory, cache memory, and memory hierarchy are essential. To understand how computers achieve

high performance despite hardware limitations, one must have a solid understanding of memory organization.

5.2 Memory Hierarchy



Memory hierarchy is an organizational concept that groups different types of memory in a computer system according to their speed, cost, and capacity. This is because no memory technology has the ability to be fast, have high capacity, and be cheap at the same time. Memory hierarchy is therefore used to achieve the primary goal of ensuring that data reaches the processor quickly while still being cost- effective.

At the top of the hierarchy are registers, which are actually found inside the CPU. Registers are very fast but small in size and very expensive. The next level after registers is cache memory, which is used to store frequently accessed data and programs. This memory is faster than main memory but slower than registers.

The next level is main memory, which is referred to as RAM and holds programs and data that are currently running. This memory is faster and has a higher capacity and is cheaper compared to cache memory. At the bottom of the hierarchy are secondary memories such as hard disks and SSDs. These memories have very high capacity and are very cheap but the slowest.

The memory hierarchy is based on the concept of locality of reference, which asserts that “a program usually refers to the same data again and again in a short period of time.” This enables the frequently accessed data to be stored in the faster memory levels.

5.3 Main Memory

The memory that interacts directly with the CPU is called main memory, sometimes referred to as primary memory. Data and instructions that are currently needed for execution are stored there. Because the CPU can only access programs and data that are stored in main memory, main memory plays a critical role in determining a computer system's overall performance. It is slower than registers and cache, but faster than secondary storage.

RAM (Random Access Memory) and ROM (Read Only Memory) are two general categories of main memory. Each has unique qualities and functions.

5.3.1 Random Access Memory (RAM)

RAM is a volatile memory, which means that when the power is turned off, its contents are lost. Programs, data, and intermediate outcomes are stored there while they are being executed.

Static RAM (SRAM)

SRAM uses flip- flops to store data. It is faster and more dependable than DRAM since it doesn't need to be refreshed. However, SRAM's storage capacity is limited by its high cost and higher power consumption. SRAM is primarily utilized in cache memory rather than main memory because of these factors.

Dynamic RAM (DRAM)

It uses capacitors for storing data. It also requires refreshing for retaining the data. It is slower than SRAM but much less expensive with high storage capacity. Hence, the memory chip used in the main memory of the computer system is usually a DRAM.

Feature	SRAM	DRAM
Full Form	Static RAM	Dynamic RAM
Storage Method	Flip- flops	Capacitor + Transistor
Refresh Required	No	Yes
Speed	Very Fast	Slower than SRAM

Cost	Expensive	Cheaper
Power Consumption	Higher	Lower
Density	Low (less storage per chip)	High (more storage per chip)
Used In	Cache memory	Main memory (RAM)

5.3.2 Read Only Memory (ROM)

ROM (Read Only Memory) is a kind of non- volatile primary memory that is used for storing permanent data and instructions in a computer system. Unlike RAM, the data in ROM is not erased when the power supply is switched off. The primary function of ROM is to store important programs like the booting program (BIOS/firmware), which assists in booting the computer.

ROM is termed “read only” because, in normal functioning, data is only readable and not writable. But some types of ROM support limited rewriting.

PROM (Programmable ROM)

PROM can only be programmed once by the programmer. Once data is written, they cannot be modified.

EPROM (Erasable Programmable)

EPROM can be erased with ultraviolet (UV) light. It can be reused. However, erasing takes a long time.

EEPROM (Electrically Erasable Programmable ROM)

The EEPROM can be erased and rewritten electrically without having to physically remove it from the system. The EEPROM is currently being used in many systems for the storage of the system firmware.

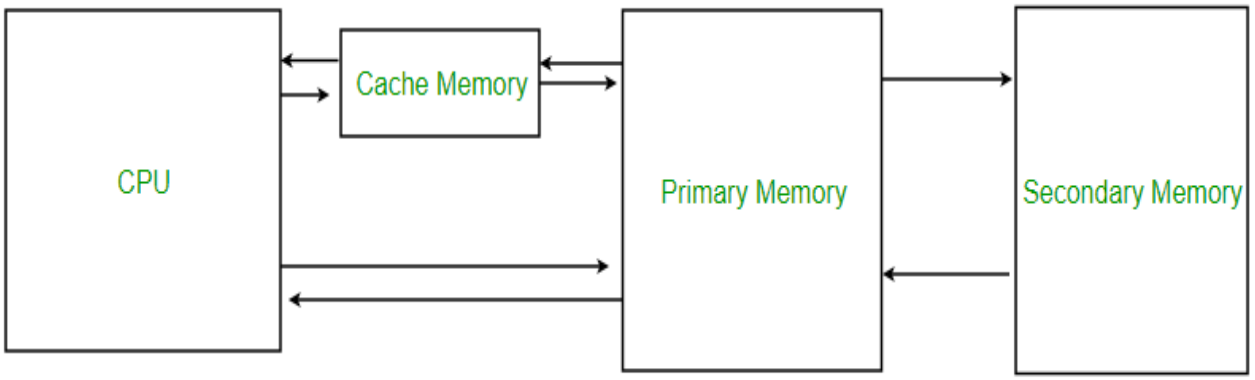
5.4 Cache Memory

The cache memory is a small and high- speed memory positioned between the CPU and memory (RAM). Its basic function is to minimize the average time involved in accessing data stored inside the memory (RAM). The CPU runs at a faster speed than the memory (RAM); therefore, accessing memory directly would delay the system’s operation. The cache memory

solves this latency challenge by placing data and instructions close to the CPU because they are frequently used.

The cache memory is based on the concept of locality of reference, according to which the program has a tendency to access different data repeatedly at a local time period (temporal locality) or has shown an access trend to data lying closer to recently accessed data (spatial locality). The CPU accessing such data is much faster than accessing main memory.

Cache memory is generally implemented by making use of SRAM, as SRAM is faster than DRAM. Hence, cache memory will be smaller in size but will be very fast. However, in modern computer systems, more than one level of cache exists, as in L1, L2, and L3, where L1 will be the smallest and fastest, and L3 will be comparatively larger but slower.



5.4.1 Need for Cache Memory

- CPU speed is much higher than main memory speed
- Reduces CPU idle time
- Improves system performance
- Decreases average memory access time

5.4.2 Cache Mapping Techniques

Cache mapping strategies describe how main memory blocks are mapped into cache memory. Since the cache memory size is smaller than that of main memory, it is necessary to have an efficient mapping strategy to identify where a given memory block is to be mapped in the cache memory. The mapping strategy has a direct influence on system performance, hit ratio, and hardware complexity.

There are three types of cache mapping strategies:

1. Direct Mapping

In direct mapping, each block of main memory maps to exactly one specific cache line. The mapping is done using the formula:

$$Cache\ Line = (Main\ Memory\ Block\ Number) mod\ (Number\ of\ Cache\ Lines)$$

This technique is simple and inexpensive to implement. However, if two frequently used memory blocks map to the same cache line, they continuously replace each other, causing more cache misses.

Example

Direct mapping is like assigning students fixed seats in a classroom. Each student must sit in one specific seat only.

Numerical Example

If:

- Cache has 8 lines
- Memory block number = 21

Then:
21 mod 8 = 5

So block 21 will be stored in cache line 5.

2. Associative Mapping

In associative mapping, any memory block can be placed in any cache line. The cache checks all tags simultaneously to find a match. This method offers maximum flexibility and reduces conflict misses.

However, it requires complex hardware for parallel searching and is more expensive.

Example

Associative mapping is like open seating in a library- any student can sit in any available seat.

3. Set- Associative Mapping

Set- associative mapping is a hybrid approach that combines direct and associative mapping. The cache is divided into sets, and each set contains multiple lines. A memory block maps to a specific set but can be placed in any line within that set.

For example, in a 2- way set- associative cache, each set has two lines.

Example

Set- associative mapping is like assigning students to a specific classroom, but they can choose any seat inside that classroom.

Numerical Example

If:

- Cache has 8 lines

- 2- way set associative

Then:
Number of sets = 8 / 2 = 4 sets

If memory block = 10
10 mod 4 = 2

So block 10 will go to set 2, in any one of the two lines.

Comparison of Cache Mapping Techniques

Feature	Direct Mapping	Associative Mapping	Set- Associative
Flexibility	Low	High	Medium
Hardware Cost	Low	High	Moderate
Speed	Fast	Slower (due to search)	Balanced
Conflict Miss	High	Very Low	Moderate

5.4.3 Cache Hit and Cache Miss

- **Cache Hit:** Required data is found in cache
- **Cache Miss:** Data is not found in cache and must be fetched from main memory.

5.4.4 Cache Memory Writing Policy

When the CPU modifies (writes) data, the system must decide how and when the updated data is written to main memory. These decisions are defined by the cache write policies. Writing policy directly affects performance, memory traffic, and data consistency.

There are major write policies:

Write- Through Policy

In the **write- through policy**, whenever the CPU writes data to cache, the same data is **immediately written to main memory** as well.

This ensures that main memory always contains the most recent copy of the data.

Example

Like writing information in your notebook and immediately updating it in a master record file.

Working

1. CPU writes data to cache.
2. Cache simultaneously updates main memory.
3. Both contain the same value.

Numerical Example

Assume:

- Cache write time = 2 ns
- Main memory write time = 50 ns

Since both are updated:

$$\text{Total write time} \approx 2 + 50 = \mathbf{52\ ns}$$

This makes write- through slower for frequent write operations.

Advantages

- Simple implementation
- Main memory always up to date
- No data inconsistency

Disadvantages

- Slower write operations
- Increased memory traffic

Write- Back (Copy- Back) Policy

In the **write- back policy**, data is written only to the cache initially. Main memory is updated **only when the modified cache block is replaced**.

A special bit called the **dirty bit** is used to indicate whether the block has been modified.

Example

Like updating notes in your personal diary and updating the official record only later.

Working

1. CPU writes data to cache.
2. Dirty bit is set.
3. Main memory is updated later during block replacement.

Numerical Example

Assume:

- Cache write time = 2 ns
- Main memory write time = 50 ns

`During normal write:`
`Time ≈ 2 ns`

Only when block is replaced:
Additional 50 ns required.

This makes write- back much faster on average.

Advantages

- Faster write performance
- Reduced memory traffic
- Efficient for frequent writes

Disadvantages

- More complex hardware
- Risk of inconsistency if power failure occurs

Write Allocate vs No Write Allocate

When a **write miss** occurs (data not in cache), the system decides:

Write Allocate

- Block is first loaded into cache.
- Then write is performed.
- Usually combined with **write- back**.

No Write Allocate

- Data is written directly to main memory.
- Cache is not updated.

- Usually combined with **write- through**.

5.4.5 Advantages of Cache Memory

- Very fast access speed
- Reduces memory latency
- Improves CPU performance

5.4.6 Limitations of Cache Memory

- High cost
- Limited size
- Complex design

5.5 Virtual Memory

Virtual memory is, in fact, a memory management concept which enables a computer system to run applications whose size is larger than the size of the physical memory present in the computer's RAM. This technique produces a simulation or impression for the user and the application that the computer possesses a large amount of memory, whereas, in fact, the computer uses the physical memory as well as the secondary storage, which may be the hard disk or the SSD.

Virtual memory plays a very significant role in today's computer technology as it helps to improve the memory and performance of the computer.

The basic concept of virtual memory is that it requires only those parts of a program that are being used currently to be in main memory. The rest of the parts are in secondary memory storage and are transferred into main memory whenever required. This task is done automatically by the operating system using hardware support.

Virtual memory functions on the concept of locality of reference, as follows: Programs generally refer or access the same location or series of instructions repeatedly for a short period of time.

5.5.1 Paging

Paging is a virtual memory management concept in which both main memory and virtual memory are divided into fixed- sized segments, or

blocks. These virtual memory segments are named pages, whereas segments in main memory are named as frames. A page table helps in maintaining synchronization between pages and frames.

Whenever the program makes a request to an unwritten page in the main memory, page fault takes place. Then the operating system transfers the desired page from the secondary storage into the main memory if there is an empty frame in the main memory.

Numerical Example :
If:

- Virtual address size = 32 bits
- Page size = 4 KB = 2^{12} bytes

Then:

- Offset bits = 12
- Page number bits = 32 − 12 = **20 bits**

5.5.2 Segmentation

Segmentation divides a program into variable- sized logical units called **segments**, such as code, data, stack, and heap. Each segment has a **segment number** and an **offset**. A **segment table** stores the base address and limit of each segment.

Segmentation supports logical program structure and provides better protection and sharing than paging. However, it may suffer from **external fragmentation**.

5.5.3 Advantage of Virtual Memory

- Facilitates the execution of large programs
- Better memory usage
- Decreases program loading time
- Supports multiprogramming
- Increases degree of Multi Programming

5.5.4 Disadvantages of Virtual Memory

- Slower access times during page faults
- It requires sophisticated hardware and software support
- Too much paging could lead to thrashing

5.6 Secondary Memory

Secondary memory has also been referred to as auxiliary or external memory and is used for storing data and programs on a permanent basis. Secondary memory is not volatile; this means that if the power supply is turned off, data cannot be lost. The cost of storage per bit of data for secondary memory is significantly lower compared to primary memory but greater in access time.

The basic function of the secondary memory is to counteract the storage limitation of the main memory. The programs and data that are not being used are stored in the secondary memory and are retrieved from there and loaded into main memory whenever required for execution. This is because the CPU cannot access data from the secondary memory directly.

Types of Secondary Memory

Hard Disk Drive (HDD):

HDDs employ magnetic storage technology in storing their data in revolving platters. HDDs provide a large storage capacity at a low cost but relatively long access time since they use mechanical systems.

Solid State Drive (SSD):
SSDs use flash memory and have no moving parts. They provide much faster data access, lower power consumption, and higher reliability compared to HDDs, though at a higher cost.

Optical Storage:
Devices such as CD, DVD, and Blu- ray store data using laser technology. They are mainly used for software distribution and backups.

5.7 Memory Performance Parameters

Memory performance parameters are used to evaluate the efficiency and speed of a memory system. These parameters are essential for understanding how memory affects overall system performance.

Access Time

Access time is the time taken by memory to respond to a read or write request. Faster memories have lower access times.

Memory Cycle Time

Memory cycle time is the minimum time required between two consecutive memory operations. It is slightly longer than access time.

Hit Ratio

Hit ratio is the fraction of memory accesses that are successfully found in cache memory.

$$Hit\ Ratio = \frac{Number\ of\ Cache\ Hits}{Total\ Memory\ Accesses}$$

Miss Penalty

Miss penalty is the extra time required to fetch data from main memory when a cache miss occurs.

Average Memory Access Time (AMAT)

$$AMAT = (Hit\ Time) + (Miss\ Rate \times Miss\ Penalty)$$

Numerical Example:
Given:

- Cache hit time = 1 ns
- Hit ratio = 95%
- Main memory access time = 50 ns

$$\text{Miss rate} = 1 - 0.95 = 0.05$$

AMAT = 1 + (0.05 × 50)
AMAT = **3.5 ns**

Multiple choice questions :

1. Which memory is fastest?
A) RAM B) Cache
C) Register D) HDD

2. Which memory is non- volatile?
A) RAM B) Cache
C) ROM D) Register

3. SRAM is used in:
A) Main memory B) Cache memory
C) Secondary memory D) ROM

4. DRAM requires:
A) Cooling B) Refreshing
C) Formatting D) Programming

5. Cache memory is placed between:
A) CPU and RAM B) RAM and HDD
C) CPU and I/O D) HDD and SSD

6. Hit ratio refers to:
A) Memory failures B) Cache success rate
C) CPU speed D) Disk access

7. Virtual memory uses:
A) RAM only B) Cache only
C) RAM + Secondary storage D) ROM only

8. Paging divides memory into:
A) Variable blocks B) Fixed- size blocks
C) Segments D) Registers

9. SSD is:
A) Mechanical B) Magnetic
C) Flash- based D) Optical

10. AMAT stands for:
A) Average Memory Access Time B) Advanced Memory Allocation Technique
C) Automatic Memory Access Tool D) None

Short Answer Questions

1. Define **memory organization**.
2. What is **memory hierarchy**?
3. List the different levels of **memory hierarchy**.
4. Define **main memory**.
5. What is the difference between **SRAM and DRAM**?
6. What is **ROM**? Mention any one type of ROM.
7. Define **cache memory**.
8. What is meant by **cache hit** and **cache miss**?
9. What is **virtual memory**?
10. Define **paging**.
11. What is **segmentation** in virtual memory?
12. What is **secondary memory**?

13. State two advantages of **SSD over HDD**.
14. Define **access time**.
15. What is the hit **ratio**?

Long Answer Questions

1. Explain **memory organization** in a computer system with a neat diagram.
2. Describe the **memory hierarchy** and explain its importance.
3. Explain **main memory** and discuss different types of **RAM and ROM**.
4. Compare **SRAM and DRAM** with suitable examples.
5. Explain the concept of **cache memory**. Why is cache memory needed?
6. Describe the working of **cache memory** with the help of a diagram.
7. Explain **virtual memory** in detail. Discuss **paging and segmentation**.
8. What is **secondary memory**? Explain different types of secondary storage devices.
9. Explain various **memory performance parameters** such as access time, hit ratio, and miss penalty.
10. Derive the expression for **Average Memory Access Time (AMAT)** and solve a suitable numerical problem.

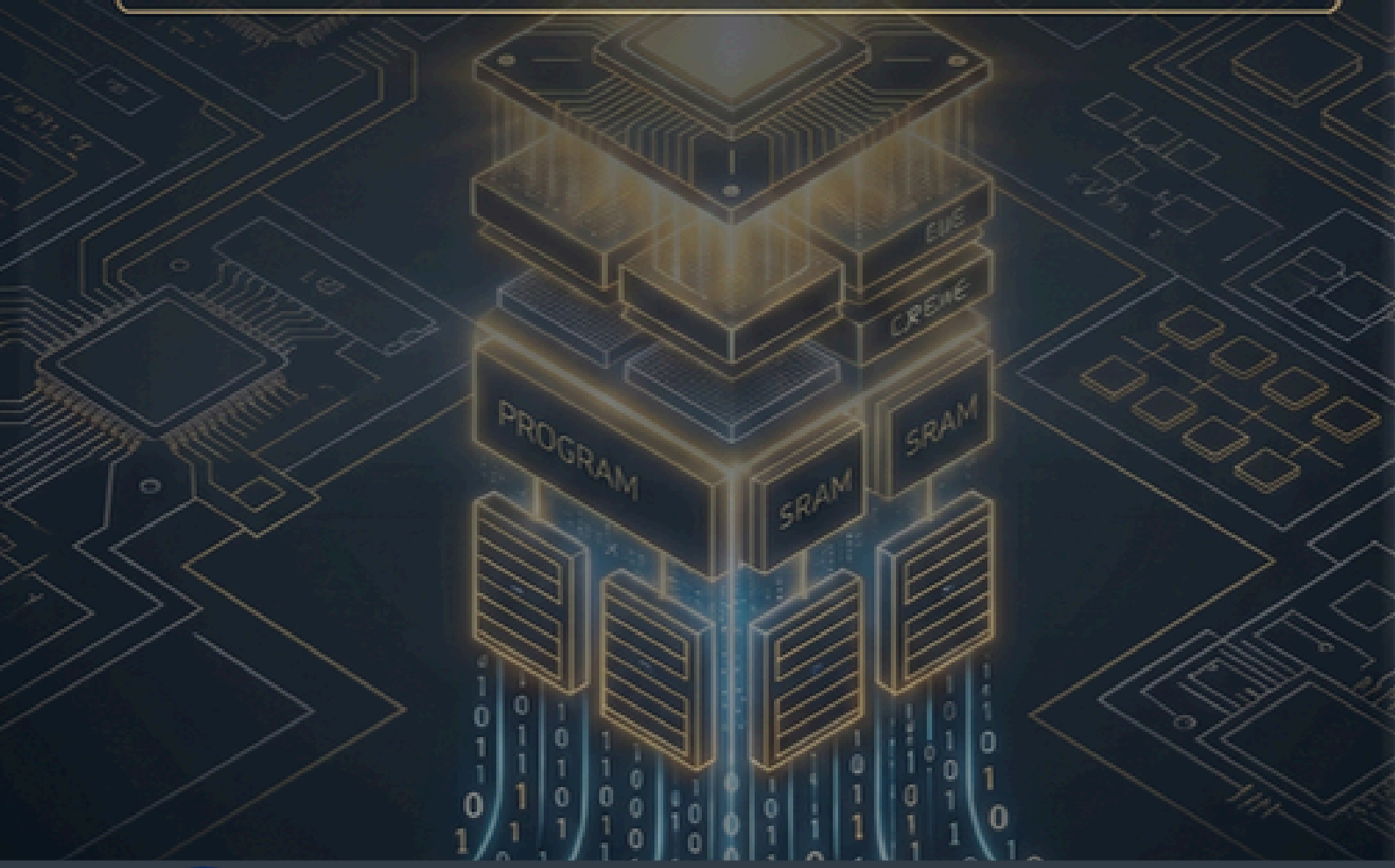
ABOUT THE BOOK

Journey into the architecture of modern computing and uncover the logic behind the machines that shape our world.

From Bits to Brilliance is the comprehensive guide to understanding the complex relationship between hardware and software, from fundamental digital logic to advanced parallel systems. Through a clear, visually-rich approach, this book demystifies:

- digital logic & micro-architecture
- instruction set architectures (ISAs),
- memory hierarchies & caching strategies, pipelining,
- input/output, and advanced CPU design trends.

Perfect for students and engineering professionals.



WORLD ACADEMIC PRESS
www.worldacademic.press