

Unit 1

Fundamentals Of Computer & Number Systems

FUNDAMENTALS OF COMPUTER

STORED PROGRAM ARCHITECTURE OF COMPUTERS

Introduction

The concept of the stored program architecture was introduced by the mathematician John von Neumann in 1945. This architecture serves as the basis for the majority of modern computers. In this concept, the data and instructions (programs) are both stored in the same memory. The CPU fetches instructions from memory, decodes them, and then executes them. This process allows for flexibility, programmability, and automation in digital computing.

Core Idea

- Store both program and data in computer memory.
- Use a control unit to sequentially fetch and execute instructions.
- Enable modification of the stored program during execution.

Importance of Stored Program Concept

- Simplifies computer design.
- Enables general-purpose computing.
- Allows self-modifying code.
- Facilitates programming using high-level languages.

Components of Von Neumann Architecture

1. *Input Unit*
 - Accepts data from external sources.
 - Devices: Keyboard, Mouse, Scanner.
2. *Memory Unit*
 - Stores both instructions and data.
 - Divided into RAM and ROM.
3. *Arithmetic Logic Unit (ALU)*
 - Performs arithmetic operations (+, -, *, /).
 - Executes logical operations (AND, OR, NOT).
4. *Control Unit (CU)*
 - Manages execution of instructions.
 - Controls the operation of all hardware.
5. *Registers*
 - High-speed temporary storage units inside the CPU.
 - Used for immediate data storage during processing.
6. *Output Unit*
 - Displays processed data to the user.
 - Devices: Monitor, Printer, Speaker.

Limitations of Von Neumann Architecture

- *Von Neumann Bottleneck*: Slow data transfer rate between CPU and memory.
- *Shared Memory*: Instructions and data share the same bus, leading to delays.

MEMORY ORGANIZATION AND STORAGE DEVICES

Memory Hierarchy

The memory in a computer system is organized in a hierarchical manner to balance cost and performance.

1. *Registers* – Fastest, smallest, expensive.
2. *Cache Memory* – Very fast, stores frequently accessed data.
3. *Main Memory (RAM)* – Volatile, directly accessible by CPU.
4. *Secondary Storage* – Non-volatile, large capacity (HDD, SSD).
5. *Tertiary Storage* – Used for backup (Magnetic tapes).

Types of Memory

a. Primary Memory (Main Memory)

RAM (Random Access Memory)

- Temporary storage.
- Data lost when power off.
- Used to hold executing programs and data.

ROM (Read Only Memory)

- Non-volatile.
- Stores firmware like BIOS.
- Types: PROM, EPROM, EEPROM.

b. Secondary Storage

Hard Disk Drive (HDD)

- Mechanical spinning disk.
- Large storage capacity.

Solid State Drive (SSD)

- No moving parts.
- Faster and more durable than HDD.

Optical Discs

- CD, DVD, Blu-Ray.
- Used for media and backups.

USB Flash Drives

- Portable storage.
- Plug-and-play.

Magnetic Tapes

- Sequential access.
- Used for archival storage.

c. Cache Memory

- Small and fast.
- Between CPU and RAM.
- Stores recently used instructions/data.
- Levels: L1, L2, L3.

d. Virtual Memory

- Extension of RAM using hard disk.
- Enables execution of large programs.
- Managed by the operating system.

MEMORY ACCESS METHODS

Sequential Access

Data accessed in a sequence.

Example: Magnetic Tape.

Random Access

Any memory location can be accessed directly.

Example: RAM.

Direct Access

Blocks of data are accessed via address.

Example: Hard Disk.

Associative Access

Data accessed by content, not address.

Example: Cache.

LANGUAGE LEVELS AND TRANSLATORS

High-Level vs Low-Level Languages

Feature	High-Level Language	Low-Level Language
<i>Abstraction</i>	Easy to understand, user-friendly	Machine dependent
<i>Examples</i>	C, Java, Python	Assembly, Machine language
<i>Speed</i>	Slower (needs compiler/interpreter)	Fast (close to hardware)
<i>Portability</i>	Portable across machines	Not portable

Translators

- *Assembler:* Converts assembly to machine code.
- *Compiler:* Converts entire high-level code to machine code at once.
- *Interpreter:* Converts and executes code line-by-line.
- *Preprocessor:* Handles directives like #include, #define before compilation.

TYPES OF SOFTWARE

System Software

Manages hardware.

Examples: OS, Device Drivers.

Application Software

Performs user-defined tasks.

Examples: Word Processor, Browsers.

Utility Software

Special purpose tasks.

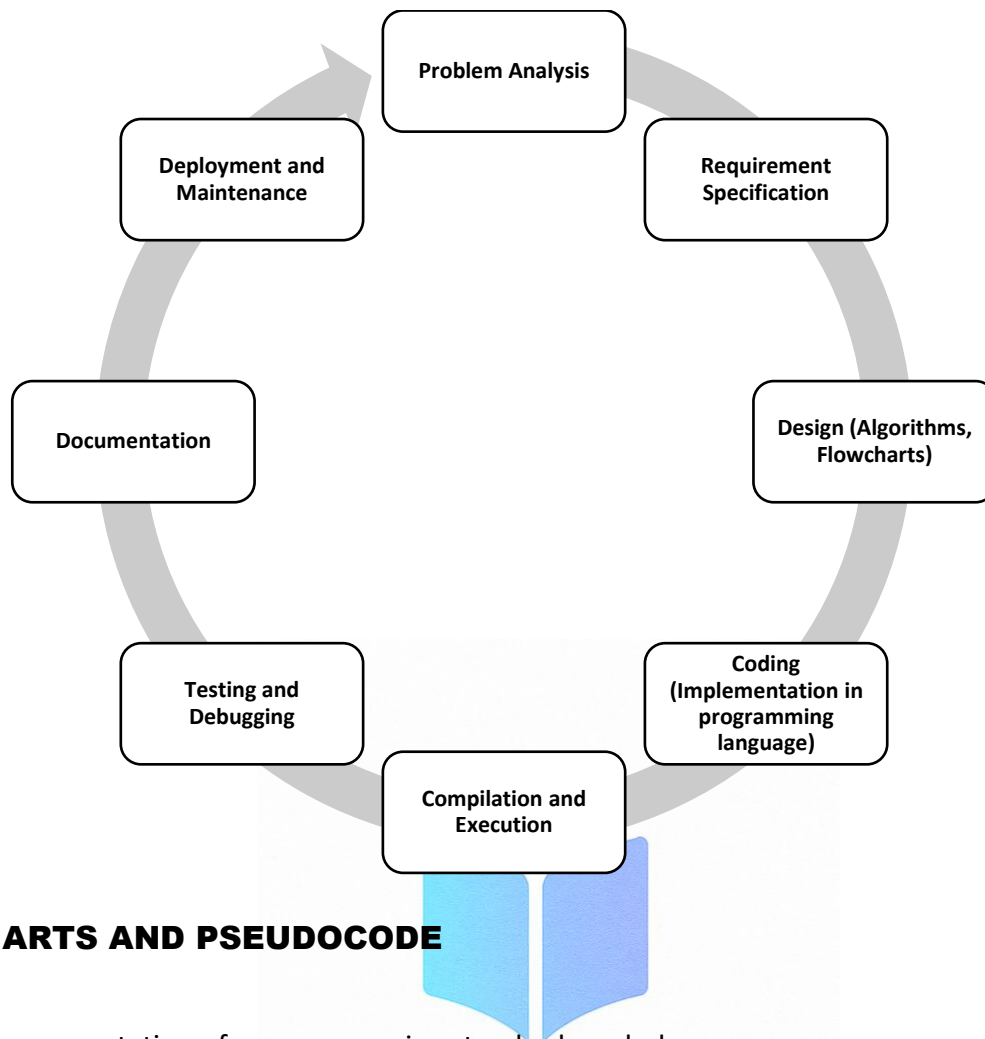
Examples: Antivirus, Compression Tools.

Programming Software

Helps in development.

Examples: Compilers, Debuggers, IDEs.





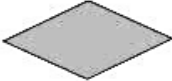
PROGRAM DEVELOPMENT LIFE CYCLE (PDLC)



FLOWCHARTS AND PSEUDOCODE

Flowchart

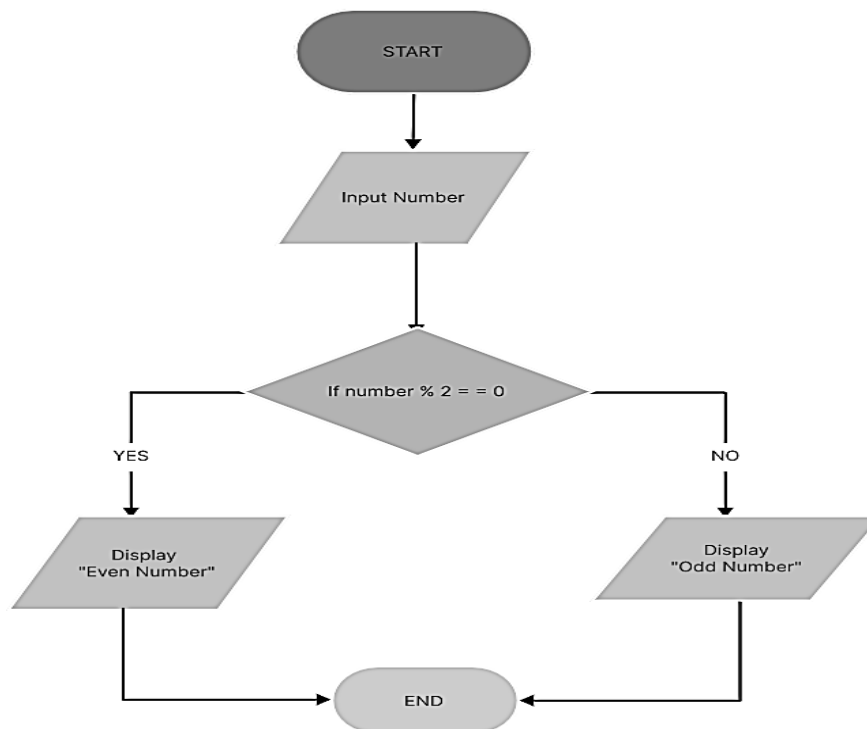
A graphical representation of a program using standard symbols.

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

Pseudocode

- A plain English representation of algorithm logic.
- Easier for programmers to convert into actual code.

Example:



NUMBER SYSTEMS

Types and Bases

Number System	Base	Symbols Used
Binary	2	0, 1
Octal	8	0–7
Decimal	10	0–9
Hexadecimal	16	0–9 and A–F

Binary to Decimal Conversion

Multiply each binary digit by powers of 2 from right to left.

Example:

$$(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11_{10}$$

Decimal to Binary Conversion

Divide the decimal number by 2 and record the remainders in reverse.

Example:

$$(11)_{10} \rightarrow 1011_2$$

Hexadecimal to Binary Conversion

Replace each hex digit with 4-bit binary.

Example:

$$A3_{16} = 10100011_2$$

Binary Arithmetic

- **Addition Rules:**
 - $0 + 0 = 0$
 - $0 + 1 = 1$
 - $1 + 0 = 1$
 - $1 + 1 = 10$ (carry 1)
- Subtraction, Multiplication, and Division also follow logical bitwise operations.

Binary Arithmetic (More Examples)

- Binary Addition Example:

$$\begin{array}{r} 1011 \\ + 1101 \\ \hline 11000 \end{array}$$

- Binary Multiplication Example:

$$\begin{array}{r} 101 \\ \times 11 \\ \hline 101 \\ + 1010 \\ \hline 1111 \end{array}$$

DATA REPRESENTATION

Type	Representation
Characters	ASCII (7-bit), EBCDIC (8-bit)
Numbers	Integer (2's complement), Floating-point (IEEE 754)
Images	Bitmap (array of pixels), JPEG, PNG formats
Audio/Video	Sampled digitally – MP3, MP4 formats

BINARY CODED DECIMAL (BCD)

Definition

Each decimal digit is represented by a separate 4-bit binary number.

Example

Number: 459

BCD: 0100 0101 1001

Application

Used in digital displays and calculators.

ASCII AND EBCDIC CODES

Encoding	Full Form	Bits Used	Characters
ASCII	American Standard Code for Information Interchange	7 or 8	English characters
EBCDIC	Extended Binary Coded Decimal Interchange Code	8	IBM Mainframe systems