Principles of Programming Languages
LECTURE -1

Introduction to Key Concepts (Preliminaries)

Presented by: Kalyani Vidhate
COEP Technological University, Pune

Introduction to Programming language

Language is a mode of communication that is used to share ideas, opinions with each other.

- What is a Programming Language?

A programming language is a computer language that is used by **programmers (developers)** to communicate with computers. It is a **set of instructions** written in any specific language (C, C++, Java, Python) to perform a specific task.

Reasons for Studying Concepts of Programming Languages

1. Improved ability to learn new languages : Knowledge of fundamental concepts makes it easier to grasp new languages.

2. Better understanding of language features : Understanding the principles behind features like data types, control structures, and syntax.

3. Enhanced ability to design new languages : Insight into language design helps in creating efficient and effective languages.

4. Better grasp of implementation issues : Understanding how languages are implemented improves debugging and optimization skills.

5. Insight into the principles behind language constructs: Appreciating the rationale behind constructs like loops, recursion, and data abstraction.

History of Programming Languages

1950: LISP, FORTRAN
1970: Ada, C, Pascal, Prolog, Smalltalk
1980: C++, ML

During 1970: a lot of PLs were designed.
Early languages:

- **Numerically based languages. (FORTRAN:55,ALGOL:58)**
- **Business languages. (COBOL:60)**
- **Artificial intelligence languages. (LISP,Prolog)**
- **Systems languages. (C:70)**

History of Programming Languages

50s and 60s :

- Early high level languages: FORTRAN, COBOL, ALGOL60
- Early mathematical based languages: LISP, APL, SNOBOL
- General-purpose language: PL/1
- Next leap forward: Algol68, SIMULA67, BASIC

70s:

- High level and structured programming: Pascal
- Systems programming: C, modula-2
- Logical programming: Prolog
- Improvement of functional programming: Scheme

History of Programming Languages

80s :

- Development of functional programming: ML, Miranda
- Need for reliability and maintainability: Ada
- Object-oriented programming: Smalltalk, C++

90s:

- Fourth-generation languages
- Productivity tools (such as spreadsheets)
- Visual languages : Delphi
- Scripting languages : Perl
- Expert systems shells
- Network computing: Java

Programming Domains

1. Scientific Applications:

- Emphasis on numerical and scientific computations.
- Extensive use of floating-point arithmetic.
- High-performance requirements.

Languages:

- **Fortran:** One of the oldest high-level languages, designed specifically for scientific and engineering computations.
- **MATLAB:** A high-level language and interactive environment for numerical computation, visualization, and programming.

Examples:

- Simulations of physical systems (e.g., climate models).
- Computational fluid dynamics.

2. Business Applications:

- Focus on data processing, report generation, and transaction processing.
- High reliability and maintainability.

Languages:

- **COBOL:** Common Business-Oriented Language, designed for business data processing.
- **Java:** Widely used for enterprise-level applications, offering portability and robustness.

Examples:

- Payroll processing.
- Inventory management systems.
- Customer relationship management (CRM) systems.

Programming Domains

3. Artificial Intelligence:

- Emphasis on symbolic computation, pattern matching, and logical inference.
- Ability to handle non-numerical data.

Languages:

- **LISP:** One of the oldest AI programming languages, known for its excellent support for symbolic reasoning and list processing.
- **Prolog:** A logic programming language used for tasks that involve complex rule-based logical queries.

Examples:

- Expert systems.
- Natural language processing.
- Machine learning algorithms

4. Systems Programming:

- Close interaction with hardware.
- Emphasis on efficiency and resource management.

Languages:

- **C:** A general-purpose language that provides low-level access to memory and system processes.
- **C++:** An extension of C that includes object-oriented features.

Examples:

- Operating systems (e.g., Linux, Windows).
- Embedded systems.
- Device drivers

5. Web Software:

- Focus on web development and internet-based applications.
- Emphasis on user interface design and client-server communication.

Languages:

- **JavaScript:** A high-level, dynamic language widely used for client-side scripting in web browsers.
- **PHP:** A server-side scripting language designed for web development.

Examples:

- E-commerce websites.
- Social media platforms.
- Web services and APIs.

Language Evaluation Criteria

1. Readability:

Simplicity:

- Minimal number of constructs that can be combined in a limited number of ways.
- Example: Python has a simple syntax that closely resembles natural language.

Orthogonality:

- A small set of primitive constructs can be combined in a small number of ways to build the control and data structures of the language.
- Example: In C, arrays and pointers can be combined freely.

Control Statements:

- Clear and straightforward control structures enhance readability.
- Example: Structured programming with if-else, loops, and switch-case in Java.

Data Types and Structures:

- Rich set of data types and clear rules for defining and using them.
- Example: Strong typing and data structure support in Ada.

Syntax Considerations:

- Consistent and meaningful syntax.
- Example: The use of indentation in Python to define blocks of code.

Language Evaluation Criteria

2. Writability:

Support for Abstraction:

- Ability to define complex structures and operations in a way that hides complexity.
- Example: Object-oriented programming in C++ and Java.

Expressivity:

- Availability of powerful operators and constructs to express ideas succinctly.
- Example: List comprehensions and lambda expressions in Python.

Ease of Use:

- Language features that simplify the process of writing programs.
- Example: Automatic memory management (garbage collection) in Java.

3. Reliability:

Type Checking:

- Detecting type errors during compilation or runtime to prevent invalid operations.
- Example: Static type checking in languages like Java and C#.

Exception Handling:

- Mechanisms to handle runtime errors gracefully without crashing the program.
- Example: Try-catch blocks in Java and Python.

Aliasing:

- Avoiding multiple references to the same memory location that can lead to unexpected side effects.
- Example: Restricting aliasing in languages like Rust with its borrowing and ownership system.

Robustness:

- Language features that contribute to the robustness and fault tolerance of programs.
- Example: The use of contracts in Eiffel for ensuring the correctness of software.

Language Evaluation Criteria

4. Cost:

Training:

- The cost of learning the language and training developers.
- Example: Python's simple syntax reduces the learning curve.

Writing:

- The effort and time required to write programs in the language.
- Example: High-level languages like Python can reduce development time compared to lower-level languages like C.

Compiling:

- The resources required to compile programs, including time and computational power.
- Example: Just-in-time (JIT) compilation in Java improves runtime performance.

Executing:

- The efficiency of the language in terms of runtime performance and resource usage.
- Example: C and C++ are known for their high execution efficiency.

Maintenance:

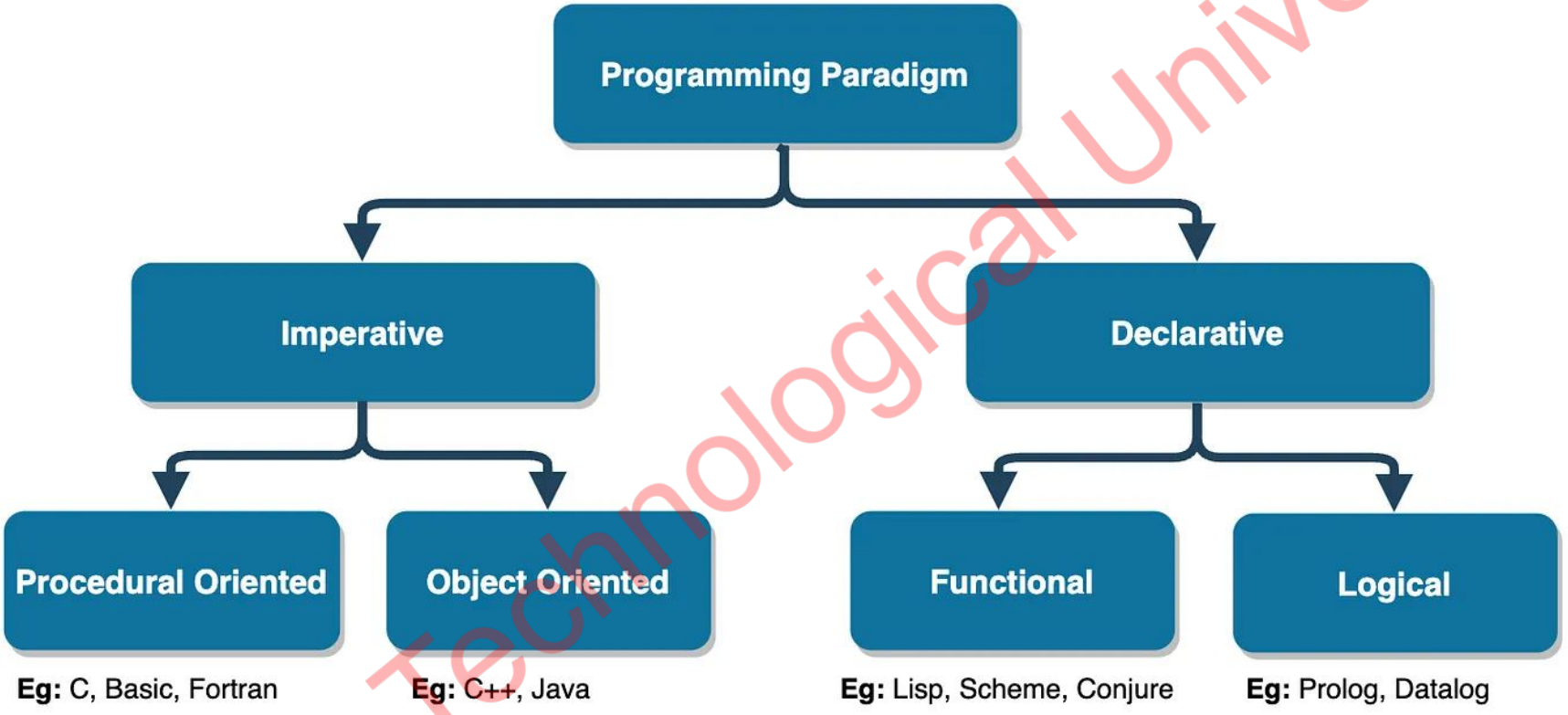
- The ease of maintaining and updating programs written in the language.
- Example: Code readability and modularity in languages like Java and Python facilitate easier maintenance.

Principles of Programming Languages
LECTURE - 2

Introduction to Key Concepts (Preliminaries)

Presented by: Kalyani Vidhate
COEP Technological University, Pune

Programming Paradigm



Programming Paradigm

Imperative Paradigm:

- Focuses on how tasks are performed.
- Includes **Procedural and Object-Oriented** subcategories.
- Procedural languages (e.g., C, BASIC, Fortran) use functions to structure programs.
- Object-Oriented languages (e.g., C++, Java) use objects and classes to structure programs.

Declarative Paradigm:

- Focuses on what the program should accomplish.
- Includes **Functional and Logical** subcategories.
- Functional languages (e.g., Lisp, Scheme, Conjure) treat computation as the evaluation of functions.
- Logical languages (e.g., Prolog, Datalog) use formal logic to express facts and rules.

Computer Architecture

Computer Architecture:

The design of programming languages is heavily influenced by the underlying computer architecture. This includes the structure and capabilities of the hardware on which programs run. Key aspects of computer architecture that influence language design are the Von Neumann architecture, memory hierarchy, and the instruction set architecture (ISA).

Von-Neumann Basic Structure:

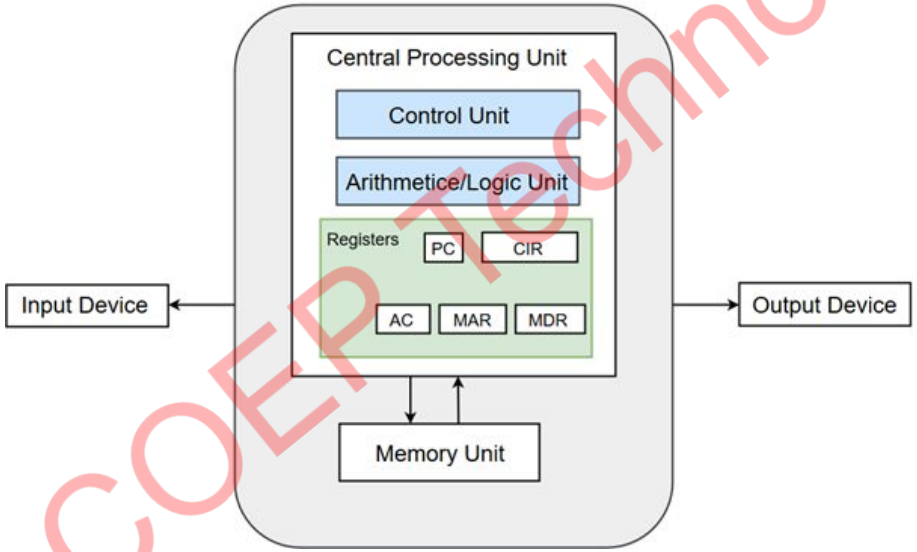


Fig : Von Neumann Architecture

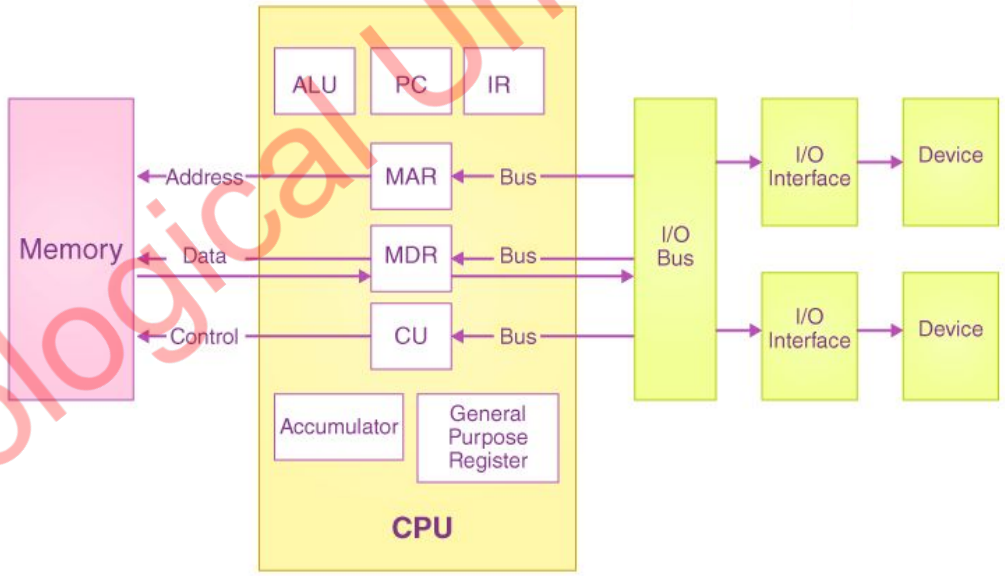
Von Neumann Architecture:

The Von Neumann architecture is a computer architecture model that describes a system where the computer's memory holds both **instructions and data**. It is the basis for most modern computer systems.

- **Components:**
  - **Control Unit (CU):** Directs the operation of the processor.
  - **Arithmetic Logic Unit (ALU):** Performs arithmetic and logical operations.
  - **Memory Unit:** Stores instructions and data.
  - **Input/Output (I/O):** Manages data exchange with external devices.
- **Influence on Language Design:**
  - **Sequential Execution:** Languages are designed to follow the fetch-decode-execute cycle.
  - **Memory Management:** Languages provide constructs for managing memory, such as pointers in C.
  - **Instruction Sets:** Language features often map closely to machine instructions, affecting performance and efficiency.

Key Concepts:

- **Unified Memory:**
  - A single read/write memory is used for both data and instructions.
  - Memory locations are utilized for storing data and instructions interchangeably.
- **Addressable Memory:**
  - Memory consists of multiple locations, each with a unique address.
  - Any memory location can be addressed and accessed by its address, allowing flexible data and instruction storage and retrieval.
- **Sequential Execution:**
  - Instructions are typically executed in a sequential order (one after the other).
  - Conditional jumps and branches are used to change the execution sequence as required.
  - For example, if executing instructions from line 1 to line 10, a condition might require jumping to line 50 instead of proceeding to line 11.



Basic CPU Structure of ALU

Architecture Components:

- **Buses**
- **Input Device:** Feeds data and instructions into the system.
- **Central Processing Unit (CPU)**
- **Output Device:** Receives processed data from the CPU and presents it to the user or another system.

Harvard Architecture:

Data and Code Separation:

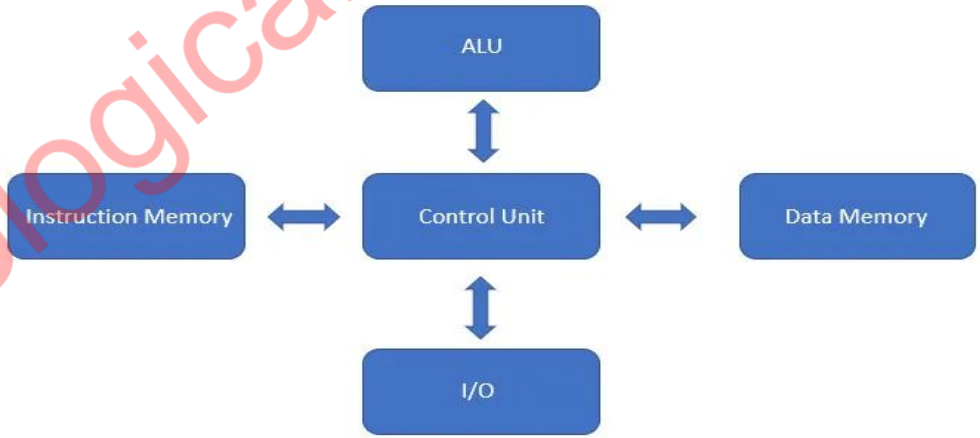
- The Harvard architecture is characterized by the **separation of data and code into different memory blocks.** This means that instructions and data are stored in distinct memory units, allowing simultaneous access.
- This separation requires different memory locations for accessing data and instructions, which improves the speed and efficiency of data processing.

Features:

- Data storage entirely within the CPU.
- Single set of clock cycles required.
- Pipelining is possible.
- Complex design.
- CPU can read/write instructions and process data access.
- Separate access codes and data address spaces.

Modified Harvard Architecture:

- Combines the advantages of Harvard architecture with a common address space for separate data and instruction caches.
- Frequently used in digital signal processors to execute small, highly repetitive audio or video algorithms efficiently.
- Microcontrollers, which have limited program and data memory, use this architecture to speed up processing by enabling parallel instruction execution and data access.



Instruction Set Architecture

Instruction Set Architecture (ISA) is a critical component of computer architecture, defining the set of instructions that a processor can execute. It acts as the interface between software and hardware, determining how instructions are interpreted and executed by the CPU.

Types:

- **RISC (Reduced Instruction Set Computer):**
  - Developed to simplify the instruction set of computers by **reducing the number of addressing modes and instructions.**
  - IBM realized this concept in the 1990s, leading to more efficient compiler designs and enhanced performance.
  - Focuses on executing simple instructions that can be completed in a single clock cycle, making the execution process faster and more efficient.
- **CISC (Complex Instruction Set Computer):**
  - Originated to simplify programming by providing a rich **set of complex instructions.**
  - Initially created when compilers were not advanced, helping programmers write code more easily.
  - Includes multiple addressing modes and complex instructions, which can perform multiple low-level operations within a single instruction.
  - Best performance is obtained by leveraging these complex instructions, though at the cost of potentially slower execution per instruction compared to RISC.

Microarchitecture

Microarchitecture, also known as computer organization, refers to the way a given instruction set architecture (ISA) is implemented in a processor. It involves the design and arrangement of the processor's functional units, such as **arithmetic logic units (ALUs), floating-point units (FPUs), registers, and caches.**

Features:

- **Varied Implementations:** Microarchitecture can vary significantly based on technology and design goals, such as performance, power consumption, and cost.
- **Execution Process:** The typical microarchitecture execution process involves reading instructions, decoding them, identifying parallel data, executing instructions, and generating output.
- **Pipeline and Parallelism:** Some microarchitectures support pipelining, where multiple instruction phases are overlapped, and parallel execution units, enhancing performance by executing multiple instructions simultaneously.
- **Execution Units:** Essential components like ALUs, FPUs, load/store units, and branch predictors are included to perform various operations.

Applications:

- Widely used in designing **microprocessors and microcontrollers**, which are the core of many computing devices.
- Microarchitecture designs often overlap multiple instructions during execution, although some designs might process instructions sequentially.

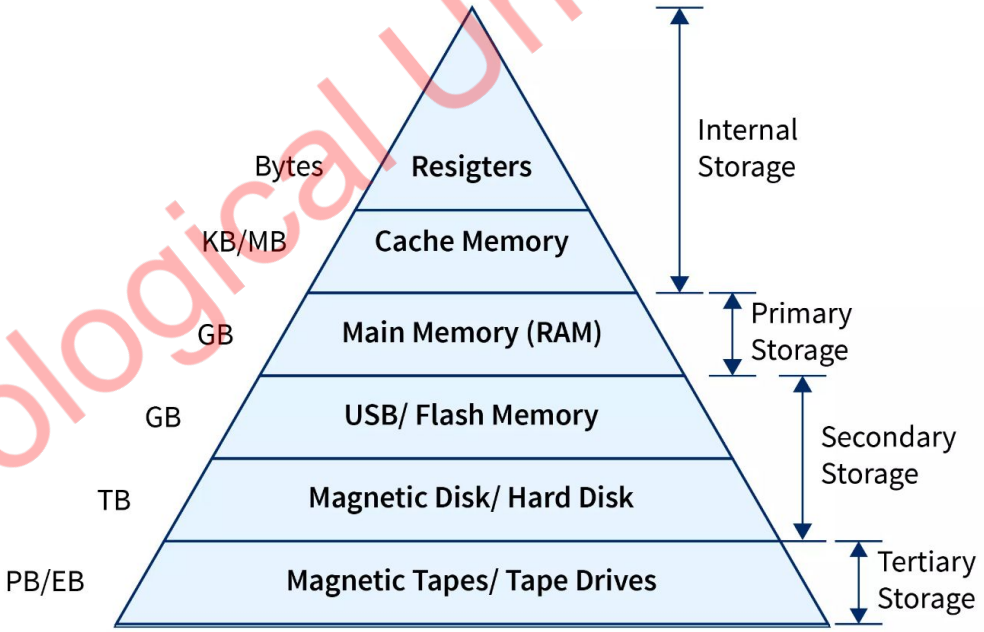
Considerations:

- Important factors in microarchitecture design include the size, latency, and connectivity of memories and caches, which affect overall system performance.

Memory Hierarchy

The memory hierarchy is the memory organization of a particular system to balance its overall cost and performance. As a system has several layers of memory devices, all having different performance rates and usage, they vary greatly in size and access time as compared to one another.

The memory Hierarchy provides a meaningful arrangement of these various memory types to maximize the performance of a system.



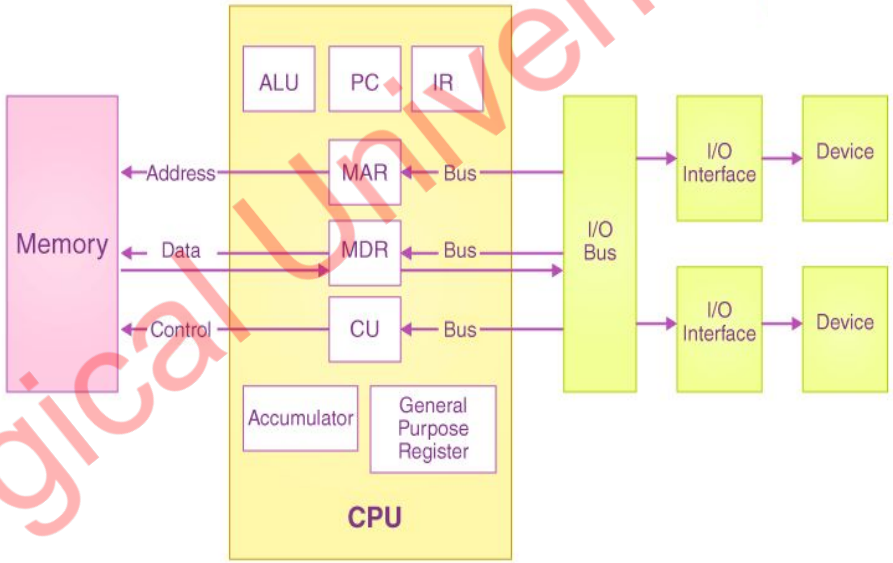
- **Influence on Language Design:**
  - **Efficient Memory Use:** Languages include features to manage memory efficiently, such as local variables, dynamic memory allocation, and garbage collection.
  - **Data Structures:** Languages provide data structures that align with the memory hierarchy, like arrays for contiguous memory access and linked lists for dynamic memory use.

SPEED, COST, CAPACITY ???

Registers: Registers refer to high-speed storage areas in the CPU.

The data processed by the CPU are fetched from the registers.

Registers	Description
MAR (Memory Address Register)	This register holds the memory location of the data that needs to be accessed.
MDR (Memory Data Register)	This register holds the data that is being transferred to or from memory.
AC (Accumulator)	This register holds the intermediate arithmetic and logic results.
PC (Program Counter)	This register contains the address of the next instruction to be executed.
CIR (Current Instruction Register)	This register contains the current instruction during processing.



Basic CPU Structure of ALU

Buses: Buses are the means by which information is shared between the registers in a multiple-register configuration system. A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer.

Von-Neumann Architecture comprised of three major bus systems for data transfer.

Bus	Description
Address Bus	Address Bus carries the address of data (but not the data) between the processor and the memory.
Data Bus	Data Bus carries data between the processor, the memory unit and the input/output devices.
Control Bus	Control Bus carries signals/commands from the CPU.

Language Design Trade-Offs

Self- study