Embedded Systems -1

ES 1: Focus

- Introduction to Embedded Systems
  - Embedded System Architecture
  - Characteristics of Embedded Systems
  - Features and challenges
- Computational Models and Computer Systems
- Computer Architecture Vs Processor Architecture
- Programs: Sequential Model
  - von Neumann Model
  - Harvard Architecture

Introduction to Embedded Systems

Ref: Embedded Systems Overview

Embedded Systems Overview

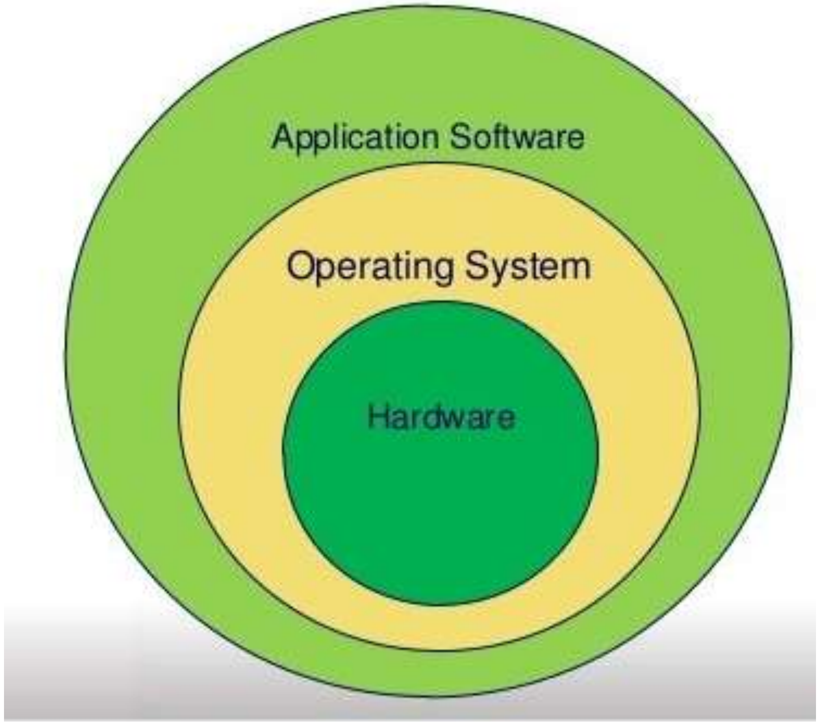
System:

- A system is an arrangement in which all its units are assembled to work together according to a set of rules.
- It can also be defined as a way of working, organizing or doing one or more tasks according to a fixed plan.
- **Example**: A digital watch is a system that displays the current time

Embedded System:

- As the name suggests, embedded means something that is attached to another thing.
- An embedded system can be thought of as a computer system having both hardware and software embedded in it.
  - An embedded system can be an independent system or it can be a part of a larger system.
- An embedded system is a microcontroller or microprocessor based system which is normally battery operated and designed to perform a specific task.
  - **Examples**: Fire alarms, coffee dispenser, smart door lock, flight controller, etc.

Embedded Software Architecture



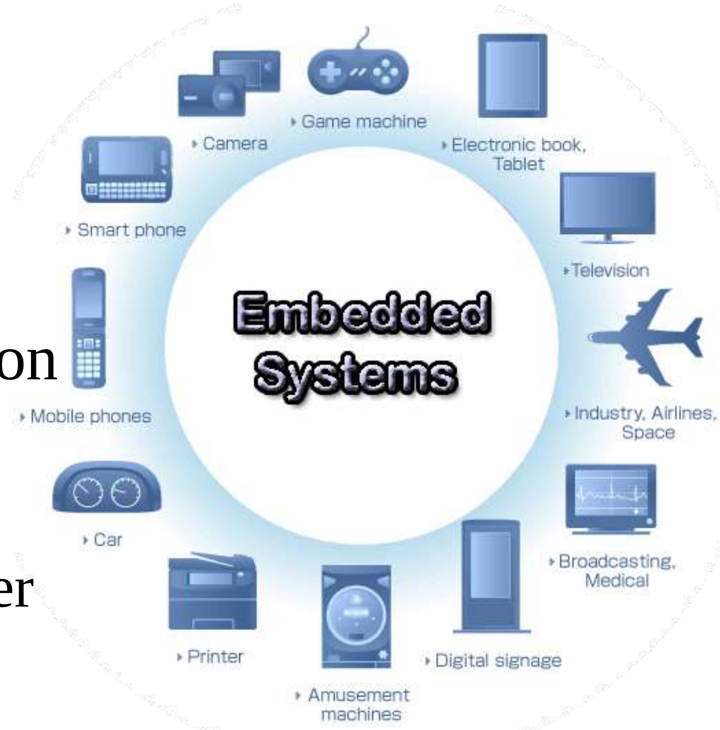
- OSs running on embedded systems are normally called RTOS, having stringent constraints on execution time, memory, latency, performance, etc.

RTOS: Real Time Operating System

Embedded Systems: Explained

An **embedded system** normally has **three components**:

- It has **hardware**.
- It has **embedded application software**.
- It has **RTOS** that supervises the application software and provides necessary support.
  - RTOS provides mechanisms to let the processor run processes in the system as per some scheduling algorithm to achieve the **latency requirements.**
  - RTOS defines the way the system works.
    - It sets the rules during the execution of application program.
  - A small scale embedded systems may not have an RTOS running in the system too



Bare metal is a computer system without a base operating system (OS) or installed applications.

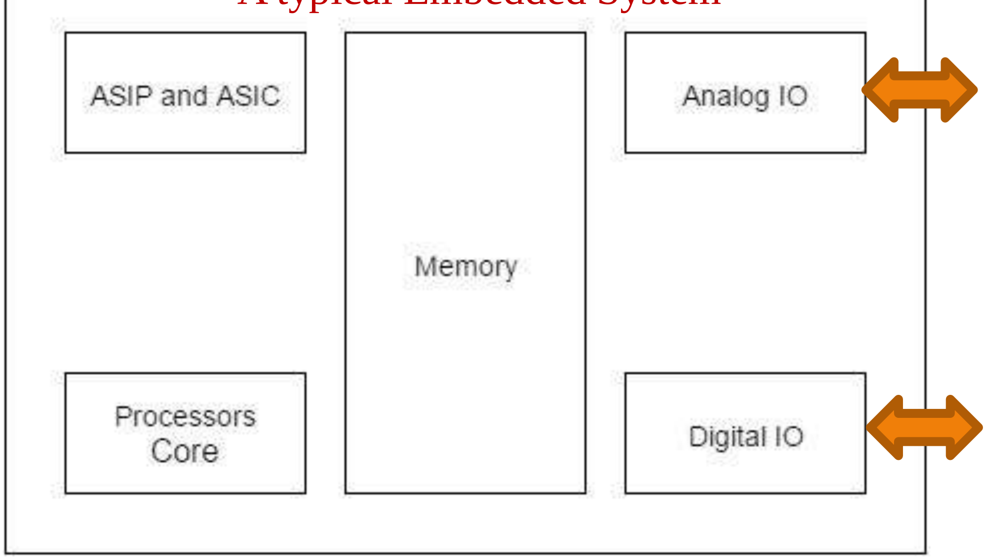
Bare metal means a system where the **software** or **firmware** is running directly on the HW (no OS)

Characteristics of Embedded Systems

1. Characteristics of Embedded Systems

- **Single-functioned** – An embedded system usually performs a specialized operation and does the same repeatedly.
- **For example**: A pager always functions as a pager.

A typical Embedded System



- **Tightly constrained** – All computing systems have constraints on design metrics, but those on an embedded system can be especially very tight, means missing the deadline would be disastrous.
- **Design metrics** is a measure of an implementation's features such as its cost, size, power, latencies and performance.
  - It must be of a size to fit on a single chip/board, must perform fast enough to process data in real time and consume minimum power to extend battery life.

ASIC: Application Specific Integrated Circuit

ASIP: Application Specific Instruction Processor

ASSP: Application Specific Standard Processor

Ref: ASIC Vs ASSP

2. Characteristics of Embedded Systems

- **Reactive and Real time** − Many embedded systems must continually react to changes in the system's environment and must compute certain results in real time without any delay.
  - Consider an example of a car cruise controller; it continually monitors and reacts to speed and brake sensors.
  - It must compute acceleration or de-accelerations repeatedly within a limited time; a delayed computation can result in failure to the control of the car.
- **Microprocessors based** − It must be microprocessor or microcontroller based on which the application or control software runs on top of an RTOS.

3. Characteristics of Embedded Systems

- **Memory** – It must have a memory, as its software usually embeds in Flash/ROM.
  - It uses DRAM or SRAM as Main Memory, normally no memory management done
  - It lacks full-fledged virtual memory support due to timing constraints similar to general purpose CPUs on Computers
  - It does not use any secondary memories (HDD) in the system, due to reliability issues and higher power consumption.
  - Flash memory is used instead of HDD for secondary storage.
- **Connected** – It must have to be interfaced to different peripherals, for connectivity or to process input and output.
  - For example, IoT devices should be connected to network.
- **HW-SW systems** – Software is used for high-end features, flexibility and configurability.
  - Hardware is used for higher performance and security as well.

Embedded Systems: Features and Challenges

Features:

- Easily customizable (adaptable to environment)
- Low power consumption (for a longer battery life)
- Low cost (selling price needs to be cheaper)
- Enhanced performance (need to meet real-time constraints)
- Highly reliable in operation (safer operation)

Challenges:

- High development effort (effort involved in making it)
  - Because they need to be highly reliable
- Larger time to market (time taken to design it)
- Higher volume requirements (no. of units)
- Low power operations (for longer battery life)

Computational Models

Computing Systems: Abstraction

Layers of Abstraction

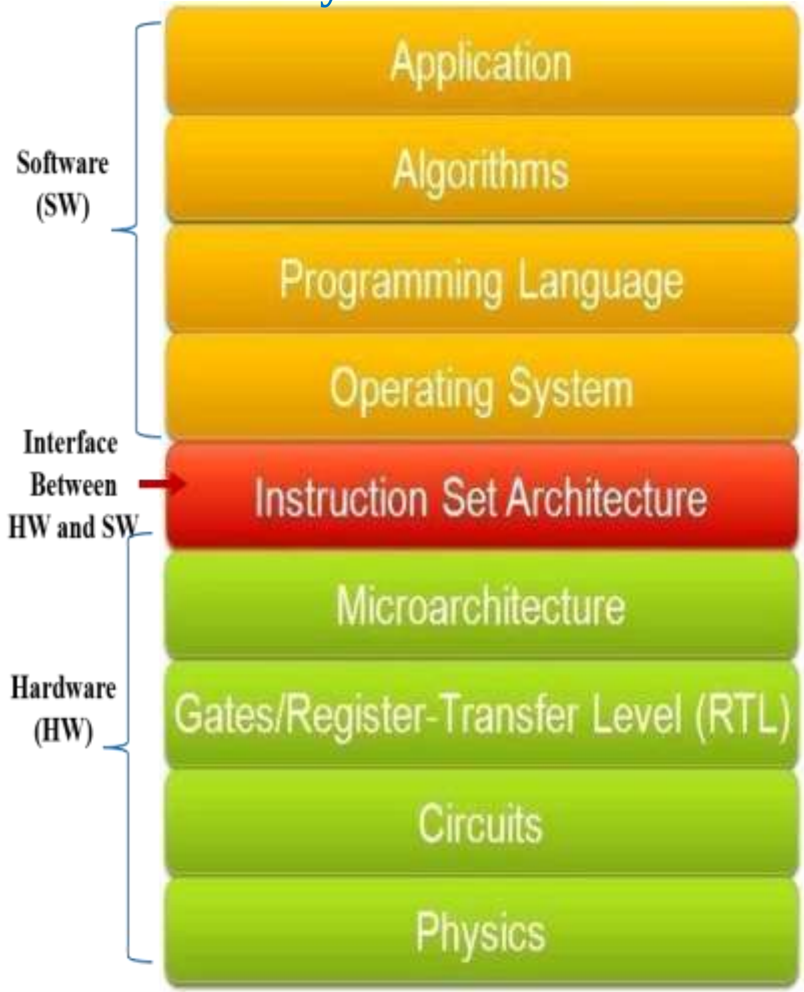
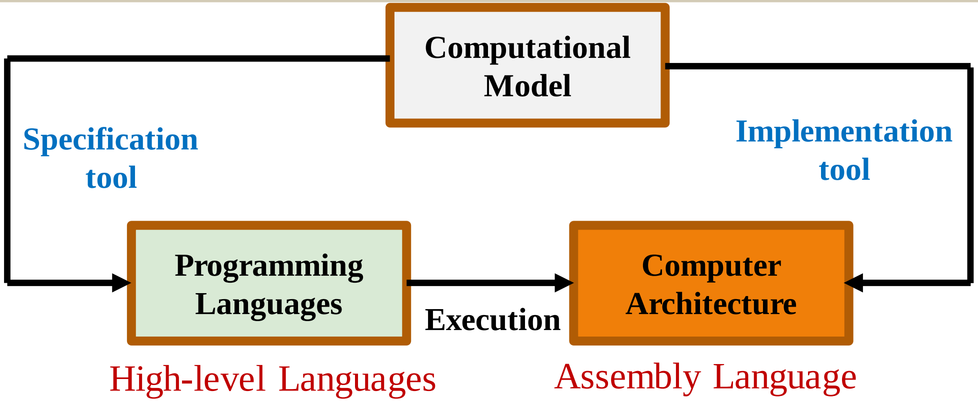
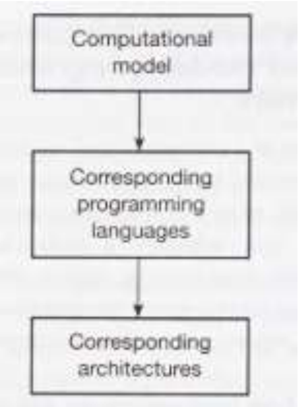


Image courtesy: Intel Corp.

Computational Models





Typical evolution sequence

Note: Other type is **declarative languages** (functional programming)

- Purpose of having a **computational model** is to be able to run a **computational task (a program)** expressed by **a programming language (specification tool).**
  - **Computer architecture (implementation tool)** that can be considered as a **tool** in **achieving** it.
- Thus, **programming languages** should allow **variables** to be declared and their values to be manipulated by a set of instructions, as many times as required during the computation.
  - It should also provide control instructions to allow explicit **control of execution sequences**.
  - Languages fulfilling these requirements are called **imperative languages**.
  - The most widely used imperative languages are C, C++, Java, Python, etc.

Ref: Types of programming languages

Programs written in an user friendly **high-level language** get **converted** by **compilers** into **Assembly level** that the **CPUs can understand** …

Computer Systems

Hierarchical Levels of Computer Systems

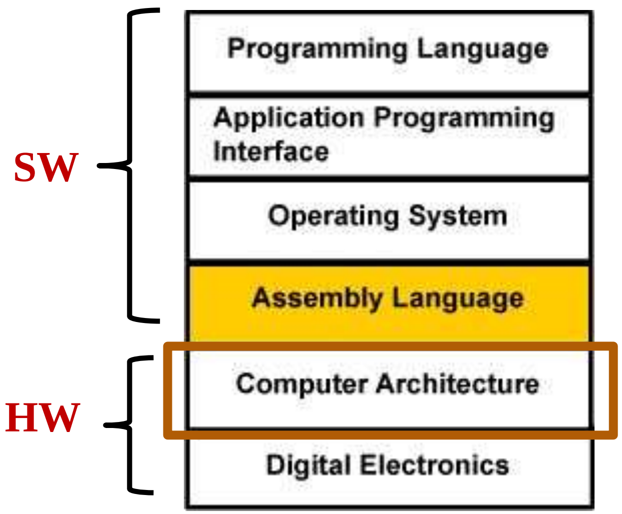


Table 2.2 A possible hierarchical six-level description of digital systems.

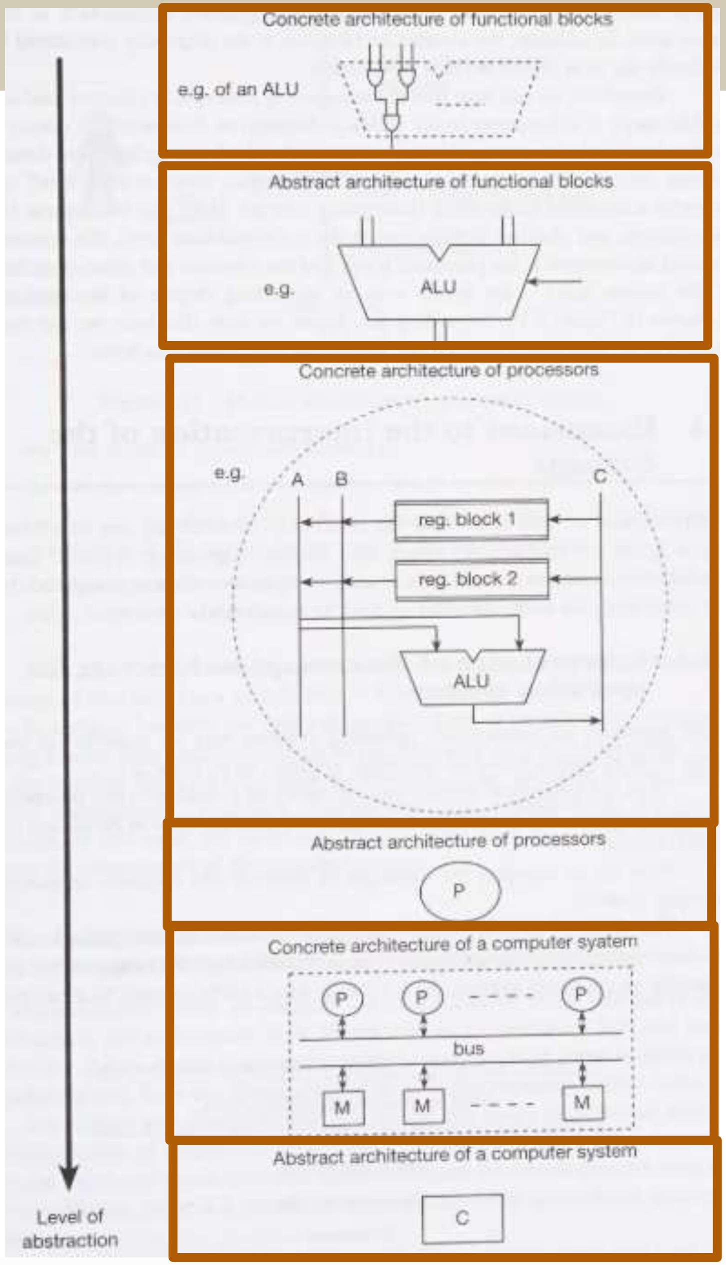
	Level of abstraction	*Basic components to be considered*	*Behavioural description by*
Computer Architecture	OS	OS	OS commands
	Computer system	Computer system	Instructions
	Processor	Processor	Basic instructions
	Functional block	Registers/MUXs, ALUs, micro-sequencer	Operations (register transfers, state sequencing)
	Circuit	Gates, FFs	Boolean equations
	Circuit element	Transistors etc.	Differential equations

Let us see different levels of abstractions of computer architecture next …

Ref: Levels of computer Systems

Architecture Levels: Abstraction

Increasing levels of abstraction



Logic Level

Functional Block Level

Architecture Level

System Level

Our focus
in this course
is here …

Computer/Processor Architecture
(Definitions)

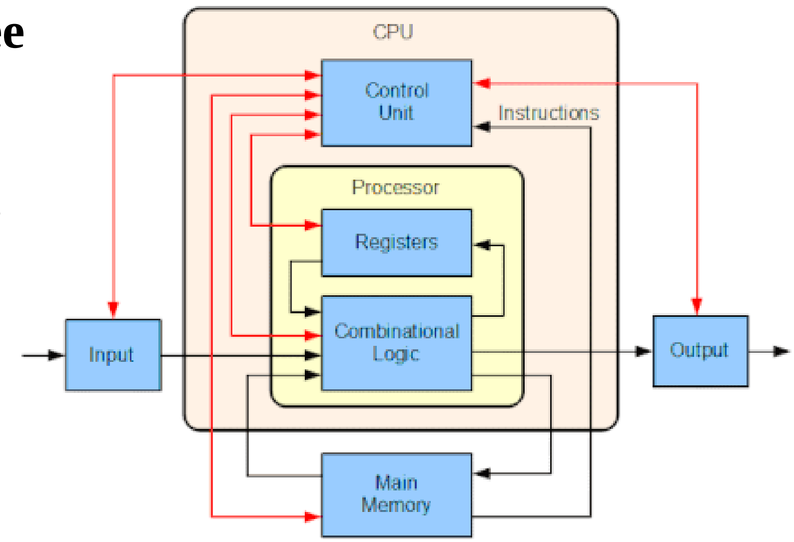
Computer Architecture: Definition

- **Computer architecture** is a set of rules and methods that describe the functionality, organization, and implementation of computer systems.

The **discipline** of **computer architecture** has **three main subcategories**

1. **Instruction set architecture (ISA)**: It **defines** the machine code that a processor reads and acts upon.
   - As well as the word size, endianness, memory address modes, processor registers, data type, etc.
   - **Example**: x86, ARM, MIPS, etc.
2. **Microarchitecture**: It is also known as "computer organization".
   - This describes how a particular processor will **implement** the **ISA**.
   - **Note**: x86, converts internally all CISC type x86 instructions into RISC type before executing them.

- **Systems design**: Includes all of the other hardware components within a computing system, other than the CPU.
  - Cache, memory, internal bus, peripherals, etc.



- Block diagram of a basic computer with a single CPU.
- The **black lines** indicate the **data flows**, whereas the **red lines** indicate the **control flows**.
- Arrows indicate the direction of flows.

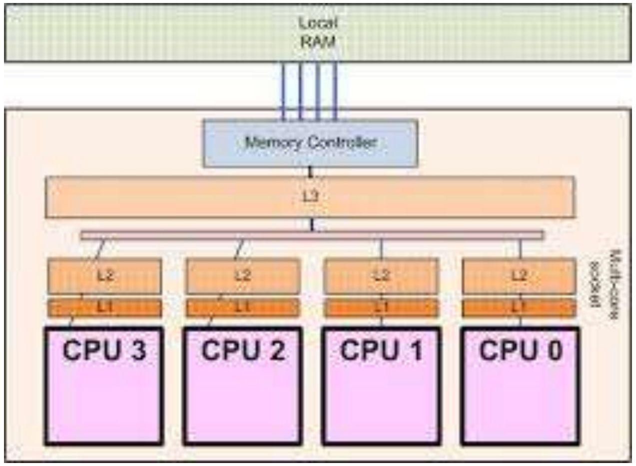
MIPS: Microprocessor without Interlocked Pipelined Stages

Processor Architecture: Definition

- **Processor Architecture:**
  - It details the design of processors (multi-cores as well), its internal organization, its interface with memory and peripherals.

Processor architecture elaborates the following:

1. **Instruction set architecture (ISA)**:
   - Instructions, as well as word size, memory address modes, processor registers, and data type, etc,
2. **Microarchitecture**:
   - It addresses the implementation of ISA.
3. **Multicore design issues**:
   - Cache coherency.
   - Internal buses interfacing with the the memory and the peripherals.



- Block diagram of a multi-core processor (**SoC**) with internal multi-level cache memories.

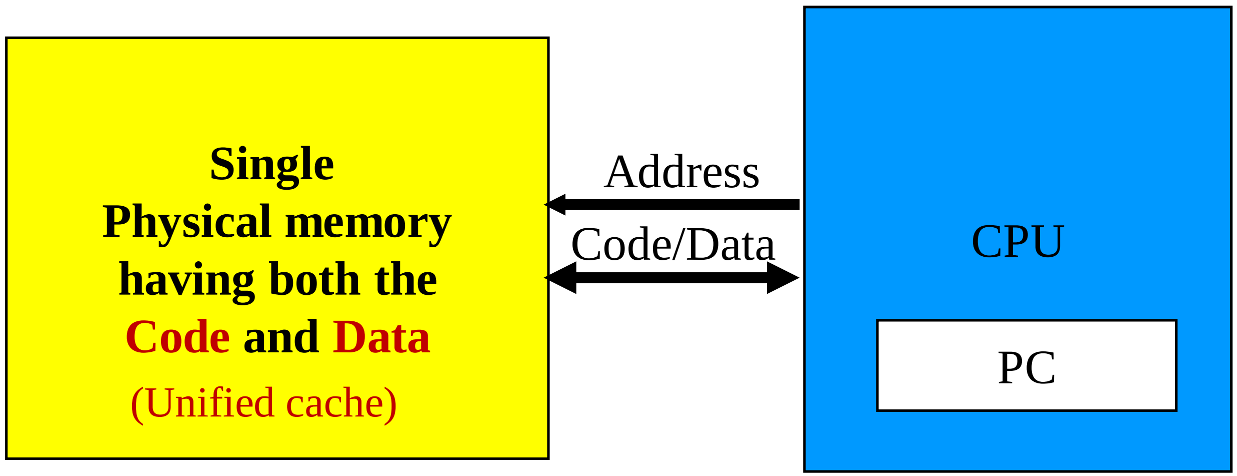
SoC: System on a Chip

- The cores are connected to the main memory through internal buses.

Note: Computer architecture also includes system level components, such as memory, peripherals etc.

Programs: Sequential Model

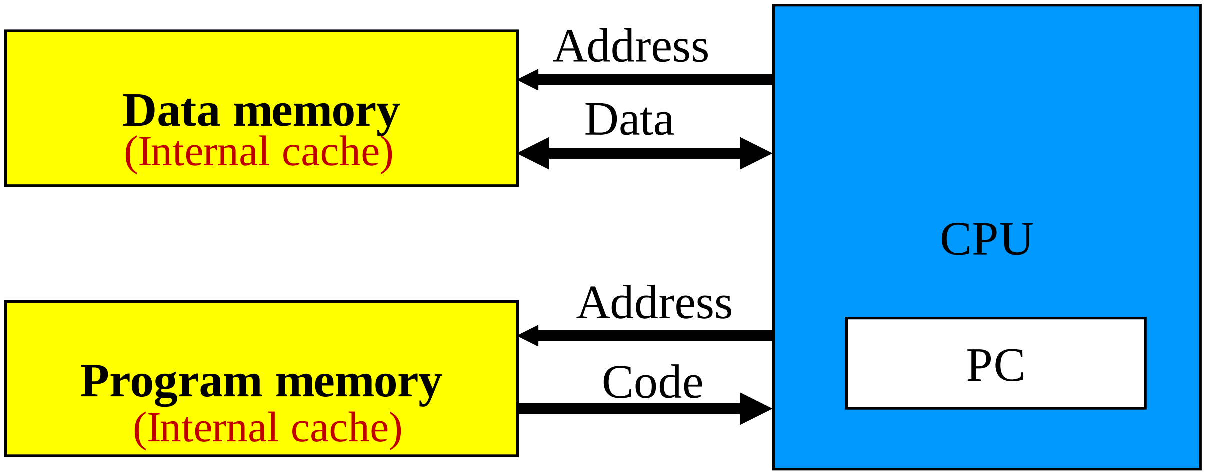
Sequential Execution: von Neumann Model



PC: Program Counter

- A **program** in the **von Neumann model** is made of a finite number of **instructions**.
- In this model, the control unit fetches **sequentially** one instruction after the other from the memory, decodes them and executes them.
  - Of course, one instruction may request the control unit to jump to some other instruction not in a sequential address, but this does not mean that the instructions are not executed sequentially.
- **Code** and **data** accesses **cannot happen in parallel** in this model, since they are accessed through the same physical interface.

Sequential Execution: Harvard Architecture



PC: Program Counter

- In this model, different memories have the code and the data with the CPU having the capability to access them independent of each other.
- This enables the programs to run **efficiently** on the CPU.
  - Because, while the code is being accessed from the program memory, data needed by the program for execution can also be accessed in parallel, from the data memory.
- This term originated from the **Harvard Mark I** relay-based computer, which had this hardware support.

Note: Most of the current processors belong to this category, because most of them have internal code and data cache memories with dedicated physical interfaces from the CPU.

ES 1: Summary

- Introduction to Embedded Systems
  - Embedded System Architecture
  - Characteristics of Embedded Systems
  - Features and challenges
- Computational Models and Computer Systems
- Computer Architecture Vs Processor Architecture
- Programs: Sequential Model
  - von Neumann Model
  - Harvard Architecture

References - 1

Ref 0 **Ref 1** **Ref 2**

References - 2

Ref 3

Cortex-M0+

Revision: r0p1

Technical Reference Manual

ARM

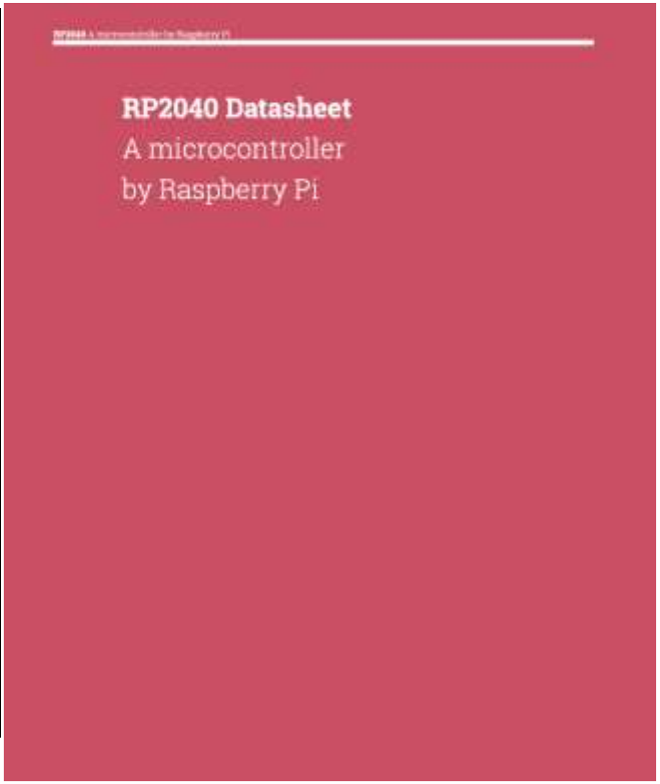
Ref 4

Cortex-M0+ Devices

Generic User Guide

ARM

Ref 5



RP2040 A microcontroller by Raspberry Pi

RP2040 Datasheet

A microcontroller by Raspberry Pi

Ref 6

RP2040 A microcontroller by Raspberry Pi

Raspberry Pi Pico C/C++ SDK

Libraries and tools for C/C++ development on RP2040 microcontrollers