

# Cyber-Physical Systems

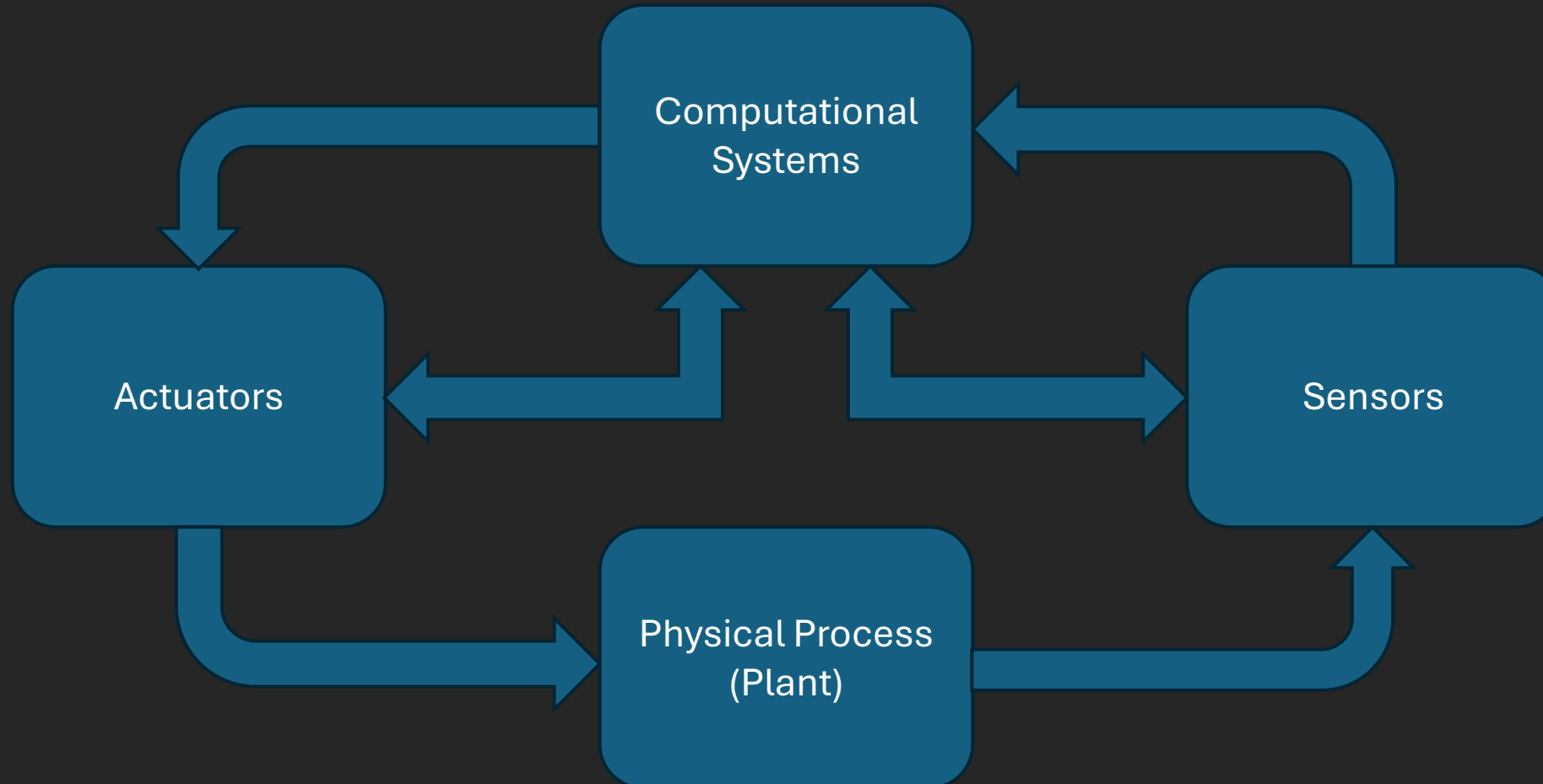
Dr. Jonathan Jaramillo



# Computational Systems

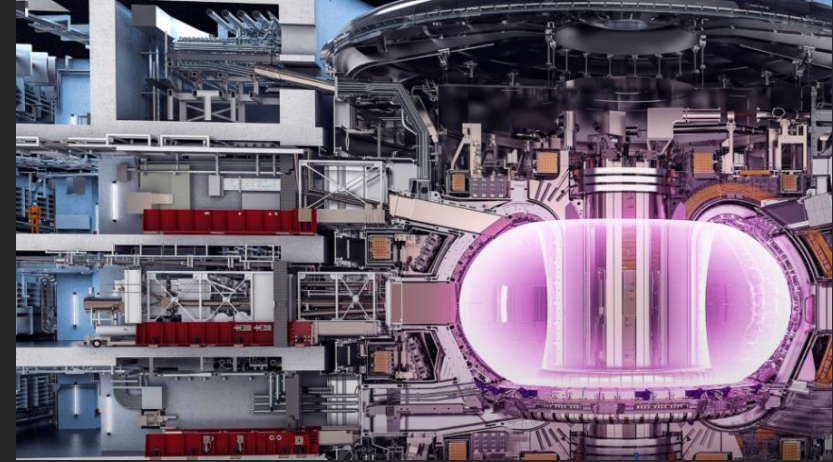


# What are Cyber-Physical Systems?



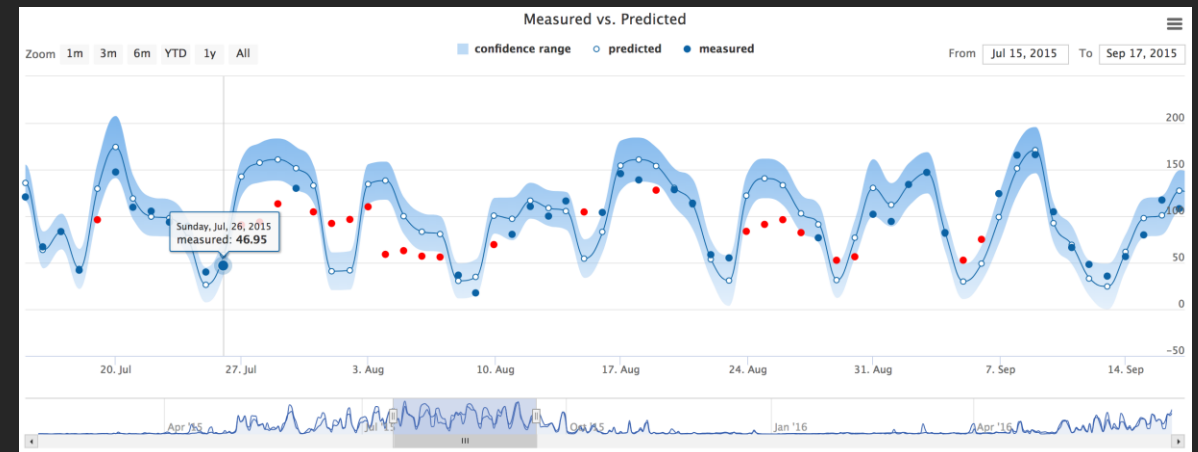
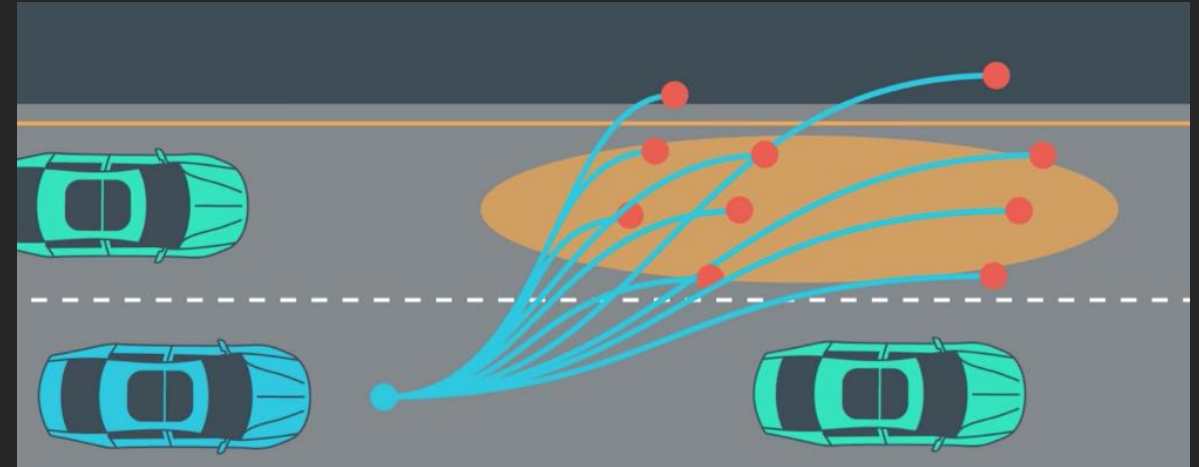
# Considerations for Computational Systems

- Real-time interactions
  - Data processing & Control



# Considerations for Computational Systems

- Real-time interactions
  - Data processing & Control
- Autonomy and Decision Making
  - Simulation & AI



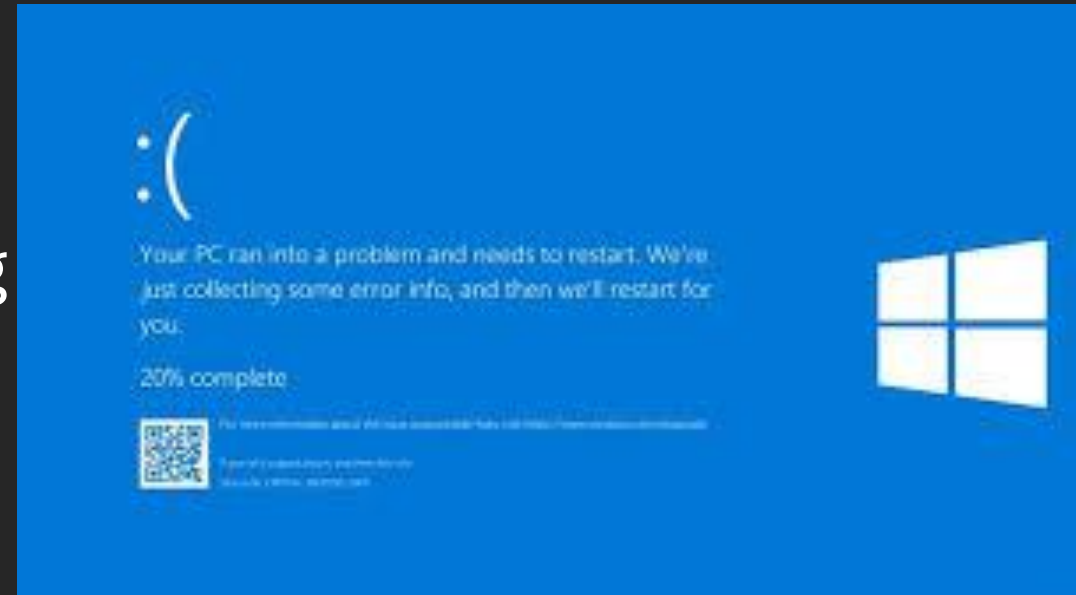
# Considerations for Computational Systems

- Real-time interactions
  - Data processing & Control
- Autonomy and Decision Making
  - Simulation & AI
- Scalability and Optimization
  - Specialized/custom hardware



# Considerations for Computational Systems

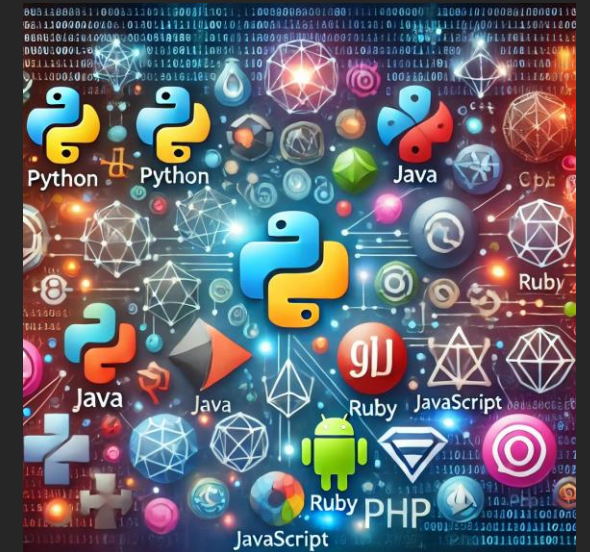
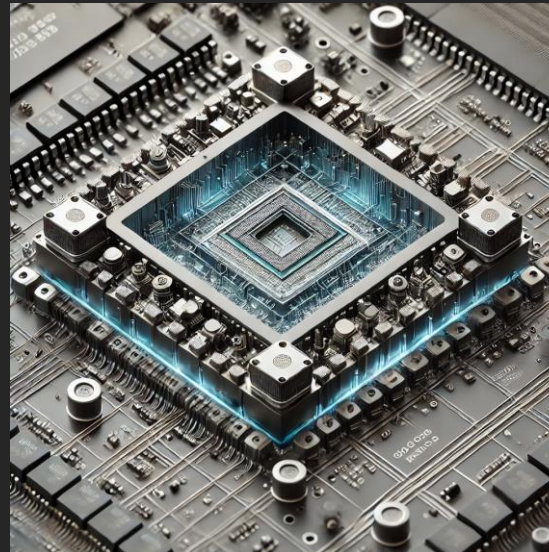
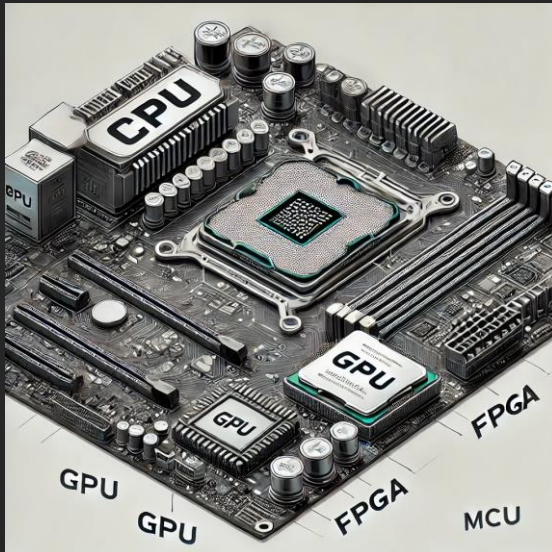
- Real-time interactions
  - Data processing & Control
- Autonomy and Decision Making
  - Simulation & AI
- Scalability and Optimization
  - Specialized/custom hardware
- Safety Criticality
  - Verifiability
  - Testing





# Overview

- Types of Computational Systems
- Computer Architectures and Instruction Sets
- Computer Programming languages



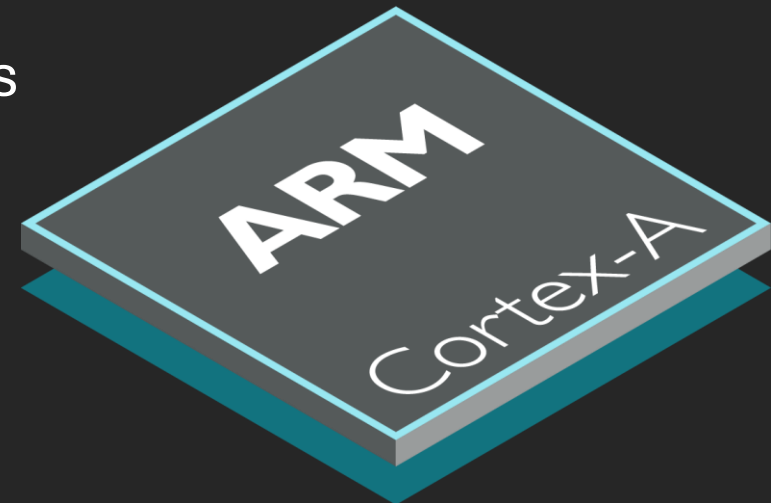


# Microprocessor (CPU)

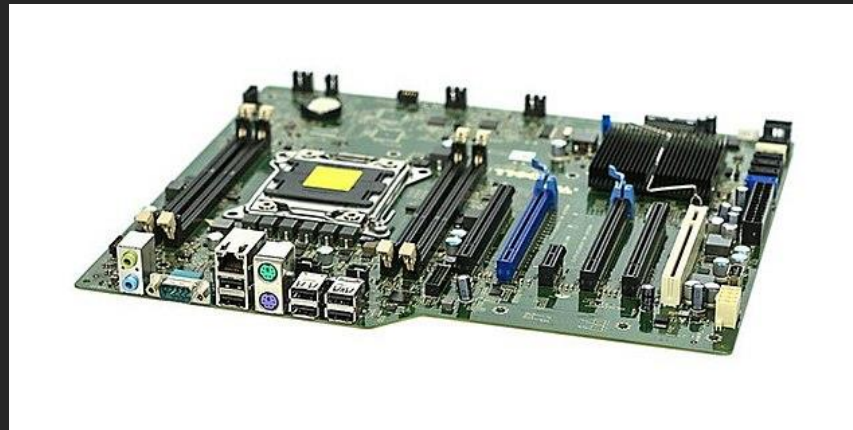
Microprocessors (Central Processing Units or CPUs) are **general-purpose** processors used to execute instructions from computer programs.

# Microprocessor

- Aka “Central Processing Unit”
  - Arithmetic, logic, control, and input/output
- Key Features
  - High-speed, sequential execution of tasks
  - Operating systems and multi-threaded applications
- Common Applications
  - Personal computers, data processing tasks
- Examples
  - Intel i5/i7/i9, AMD Ryzen, ARM Cortex-A



# Microprocessor Vs Computer



# Microprocessor – Key Characteristics

- Clock Speed (GHz)
  - Performance, power consumption, heat generation
- Core Count
  - Parallelism and multi-threaded application speed
- Architecture
  - Software compatibility and optimization performance
- Thermal Design Power
  - Cooling requirements

# Microprocessor – Key Characteristics

- Cache Size (L1, L2, L3)
  - Performance, reduces need to fetch data from slower memory
- Memory Support
  - Parallelism and multi-threaded application speed
- Peripheral Support and I/O Interfaces
  - Determines system integration
- Graphics Processing Unit Integration
  - Display and AI support



# Microprocessor – Key Characteristics

- Real-Time Capabilities
  - Guarantee response times, hardware interrupt handling
- Instruction Set Extensions
  - Faster processing for multimedia, cryptography, scientific computing
- Security Features
  - Sensitive data and data integrity
- Power Consumption & Efficiency
- Operating Temperatures
  - Industrial, automotive, outdoor applications



# Single Board Computer

- Processor, memory, storage, I/O on a single board
- Key Features
  - Run full operating systems
  - Small form factors, non-modular
- Common Applications
  - IoT, Edge computing, robotics
- Examples
  - Raspberry Pi, Nvidia Jetson

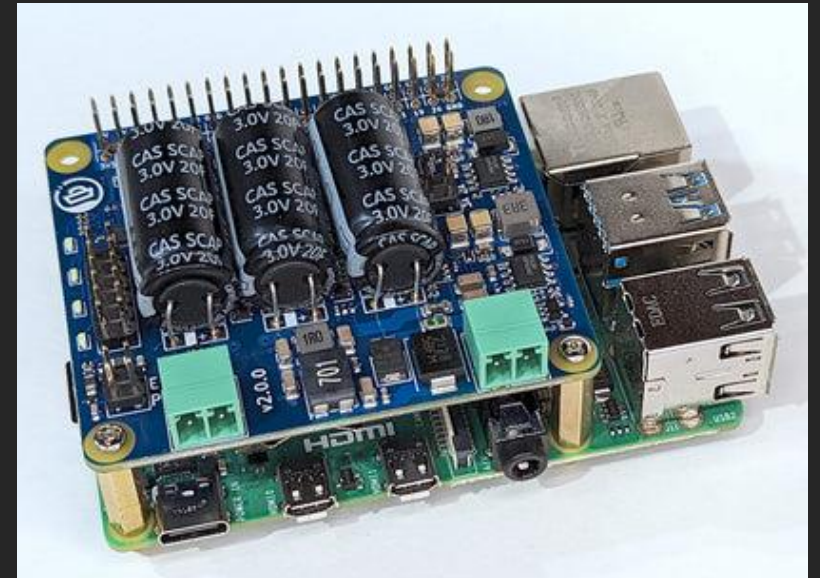


# SBC – Key Characteristics

- Processor
  - Computational performance
- Memory, storage, and connectivity
  - Non-modular characteristics
- Graphics and Video Support
  - “headless” operating systems
- OS Support
  - Determines software compatibility and real-time support
- Power Supply, Formfactor, and Size
  - Determines physical constraints

# SBC – Key Characteristics

- Expansion Options
  - Can extend capabilities
- Environmental Factors
  - Dust, temperature, ventilation, vibration, humidity
- Community and Support
  - Ease of implementation



# Graphics Processing Units (GPUs)

- Specialized hardware designed for accelerating parallel processes associated with image, video, and tensor computation.
- Key Features
  - Tensor Processing Units (TPU) - generalized GPUs designed for tensor computation (deep learning, high dimension matrix multiplication)
  - Improved energy and computational efficiency
- Common Applications
  - Image/video rendering, LM (detection, classification, LLMs)
- Examples
  - Nvidia GeForce RTX, AMD Radeon RX, Google Corel



# GPUs – Key Characteristics

- Architecture, CUDA Cores, Stream Processors
  - Governs task performance (ray tracing, ML training, simulations)
- VRAM and Memory Bandwidth
  - Governs the size of model and amount of data, and access speed
  - Example: Computer vision -  $\text{Memory} \sim \text{Model Size} * \text{Batch Size}$
- Clock Speed
  - Heat dissipation, computation speed, power consumption
- Power Consumption, Form Factor, Interface
  - Determines physical constraints



# GPUs – Key Characteristics

- Cooling Solutions
  - Governs scalability
- Multi-GPU support
  - Governs scalability
- Display Outputs
  - Type and number of ports
- OS and Software Support
  - GPUs are not stand-alone computational systems



# Field Programmable Gate Array (FPGA)

- FPGAs are customizable hardware devices that allow developers to program logic circuits post-manufacturing.
- Key Features
  - Tailored to specific tasks for low latency, high performance
  - Reconfigurable, ideal for precise timing
- Common Applications
  - Real time data acquisition, signal processing, communications
- Examples
  - Xilinx Zynq, Intel Stratix, Altera.



# FPGAs – Key Characteristics

- Logic Elements/Logic Cells
  - Basic Building block, determines the complexity of circuit
- DSP Blocks, Hardware Acceleration, and IP Cores
  - Specialized logic blocks that govern computational abilities
- Memory
  - Governs scale of data that can be processed
- I/O Pin Count and Interface Support
  - Determines capacity for interfacing with external hardware

# FPGAs – Key Characteristics

- Configuration Option
  - Governs how the logic is configured on startup
- Development tools and ecosystem
  - IDEs, hardware description languages (HDLs), and debugging tools
- Form Factor, Operating Temperature, Power
  - Governs environmental and physical compatibility
- Security Features
  - Deters tempering, reverse engineering, and unauthorized access

# Application-Specific Integrated Circuit (ASIC)

- Custom-designed ICs specific for an application or function, optimized for performance, power efficiency, and cost.
- Key Features
  - Very Expensive to develop and manufacture
  - Highly efficient at a single task
- Common Applications
  - Network routing, cryptocurrency mining, telecommunications
- Examples
  - Bitmain Antminer Series, Sony BIONZ



# ASIC – Key Characteristics

- Functionality and Customization
  - The specific task the chip is designed to do
- Cost and Time-to-Market
  - 12-36 month for design and manufacturing
- Manufacturing Technology
  - Governs size of transistor that can be manufactured
  - Photolithography, extreme-ultraviolet, electron beam
- Verification and Testing
  - Functionality, reliability, and design iterations

# System-on-Chip (SoC)

- SoCs integrate all components of a computer into a single chip, processor, memory, I/O interfaces, and accelerators
- Key Features
  - High integration reduces power and physical space
- Common Applications
  - Smartphones, tablets, embedded systems
- Examples
  - Qualcomm Snapdragon, Apple A-series, MediaTek, NVIDIA Tegra



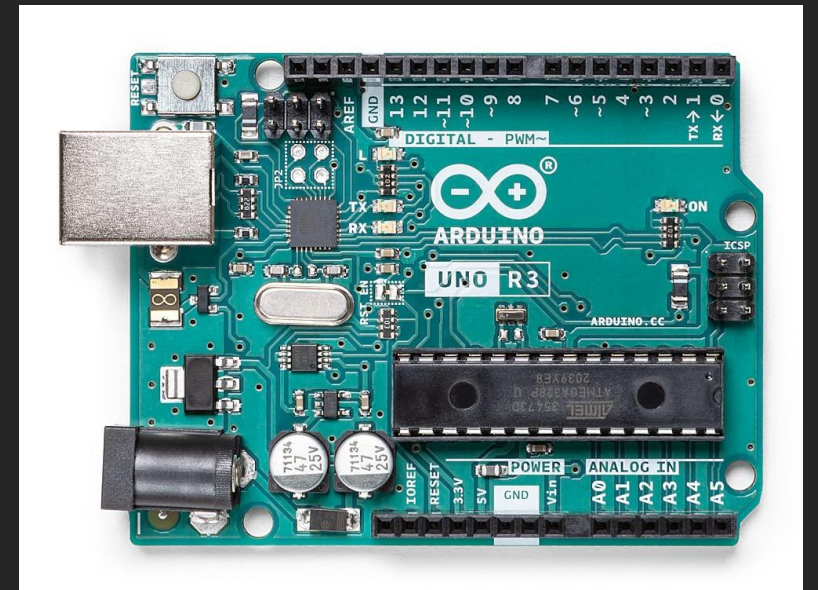


# Microcontrollers (MCU)

- Compact ICs designed for dedicated control functions
  - Processor, memory, and I/O peripherals on a single chip
- Key Features
  - Optimized for low power consumption
  - Used in repetitive tasks like controlling sensor and actuators
- Sensor control, automotive, smart devices, home automation
  - Hardware oriented, control platforms
- Examples
  - Arduino, ESP32, STM32

# MCU – Key Features

- Communication Interfaces
- I/O Pin functionality
  - ADC Sampling frequency, bit rate, PWM or DAC
- Hardware Interrupts
  - Governs real-time safety
- RTOS support
- Wireless connectivity
- Operating voltage



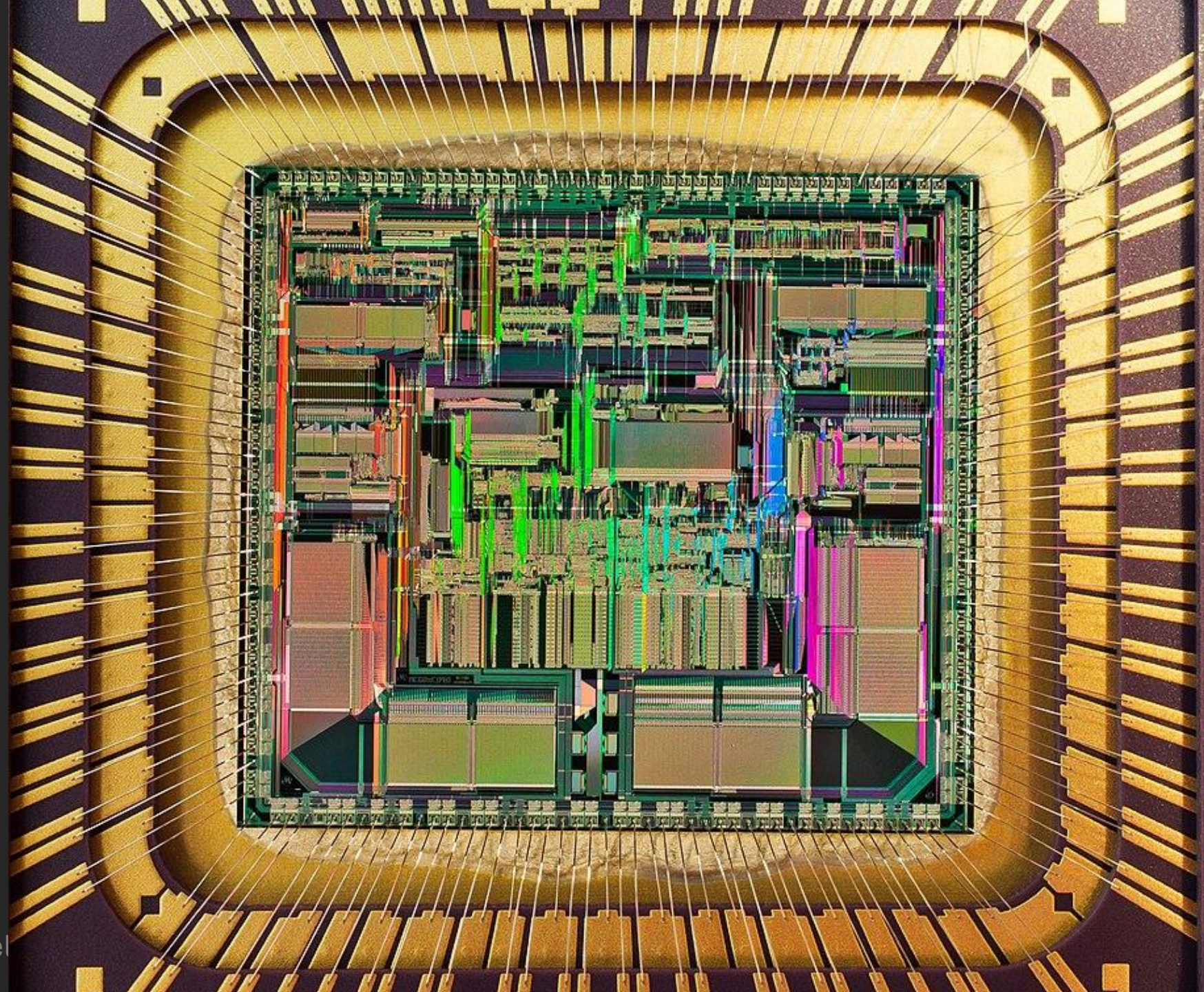
# Computational Systems

- Microprocessor (CPU)
- Single Board Computer (SBC)
- Graphics Processing Unit (GPU)
- Field Programmable Gate Array (FPGA)
- Application Specific Integrated Circuit (ASIC)
- System-on-Chip (SoC)
- Microcontroller (MCU)

# Computer Architecture







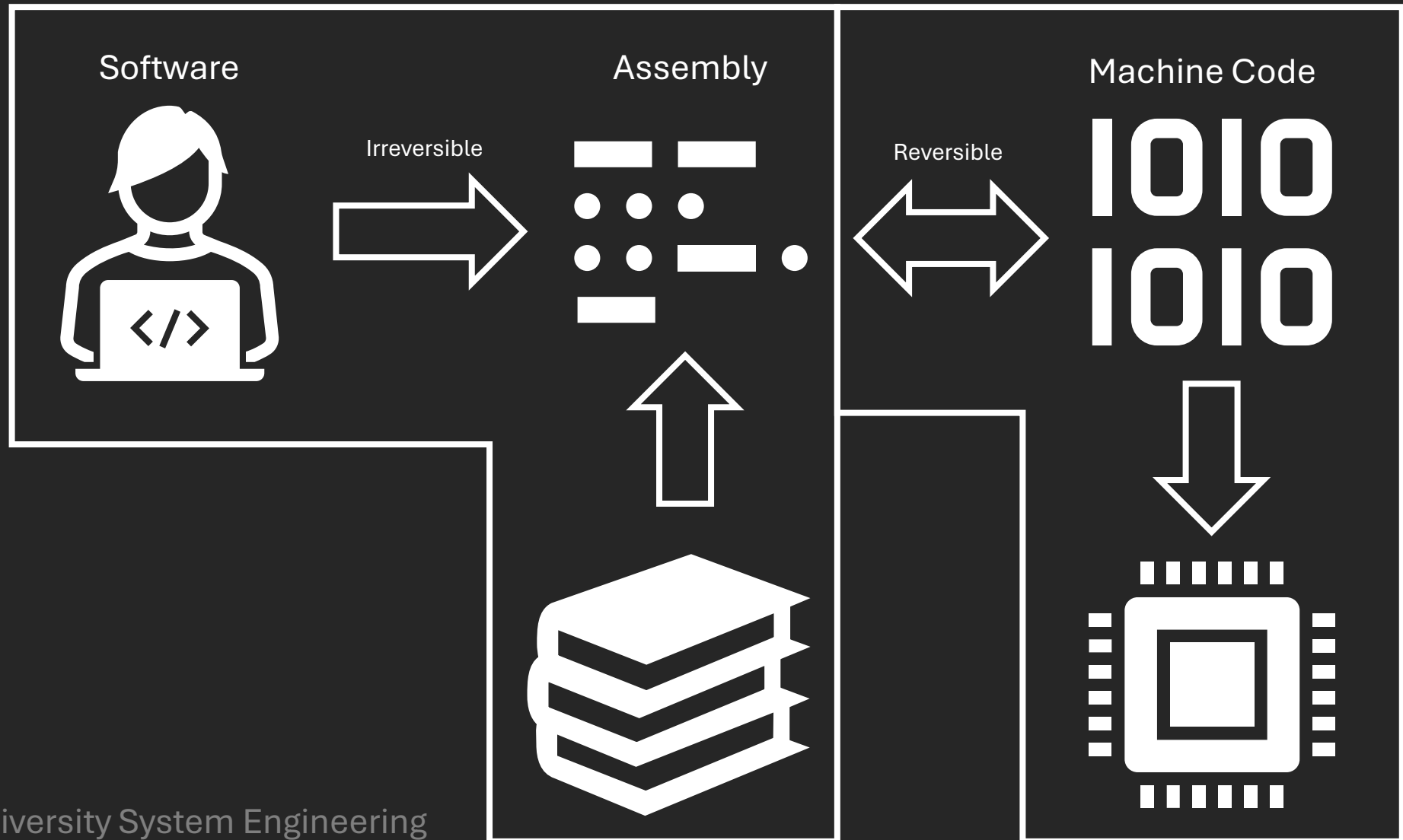


# How Does A CPU Work?

- Compiler: Source code is compiled to assembly language.
- Assembler: Assembly is converted to machine code.
- Linker: Machine code is combined with libraries to create an executable file.

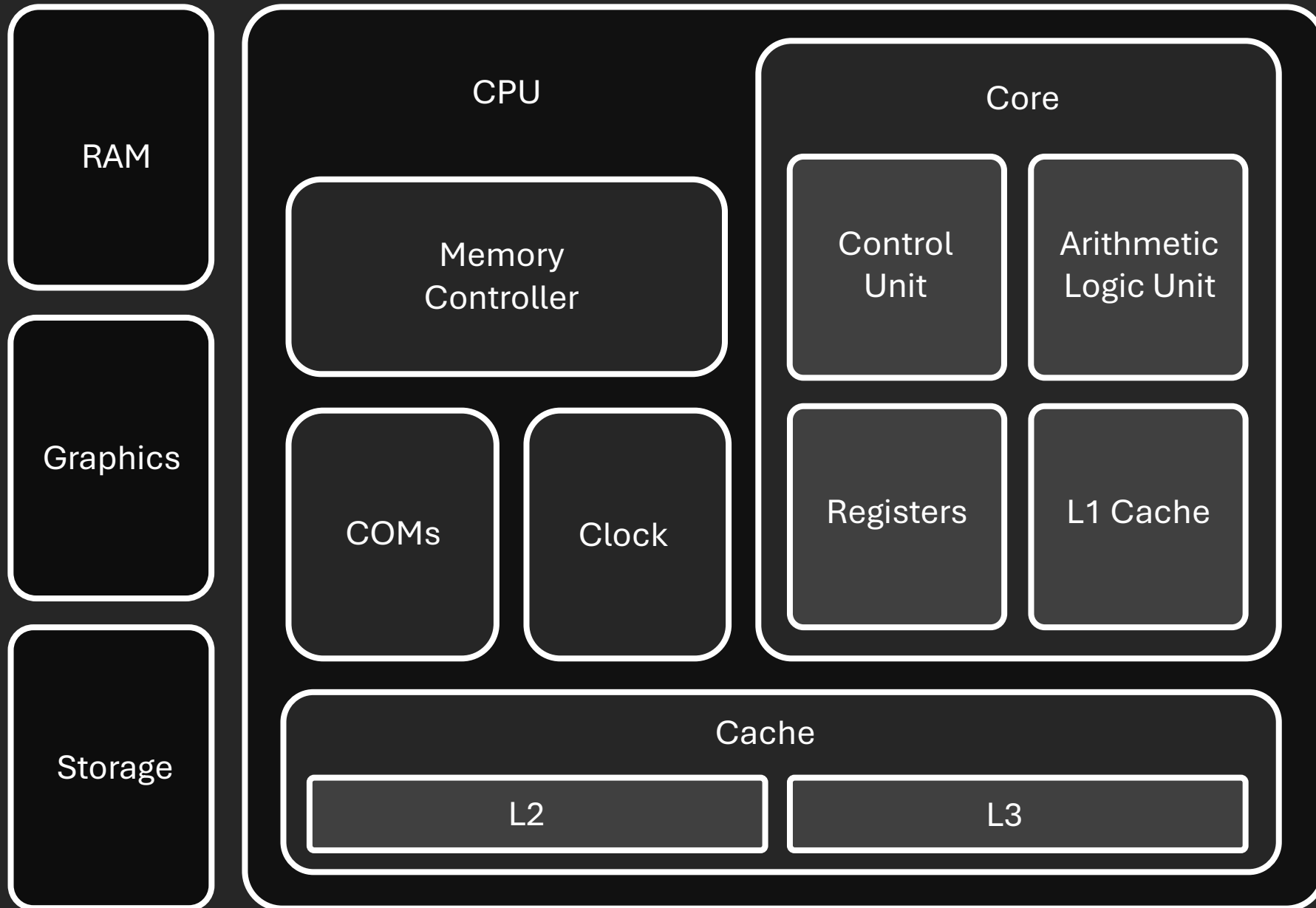


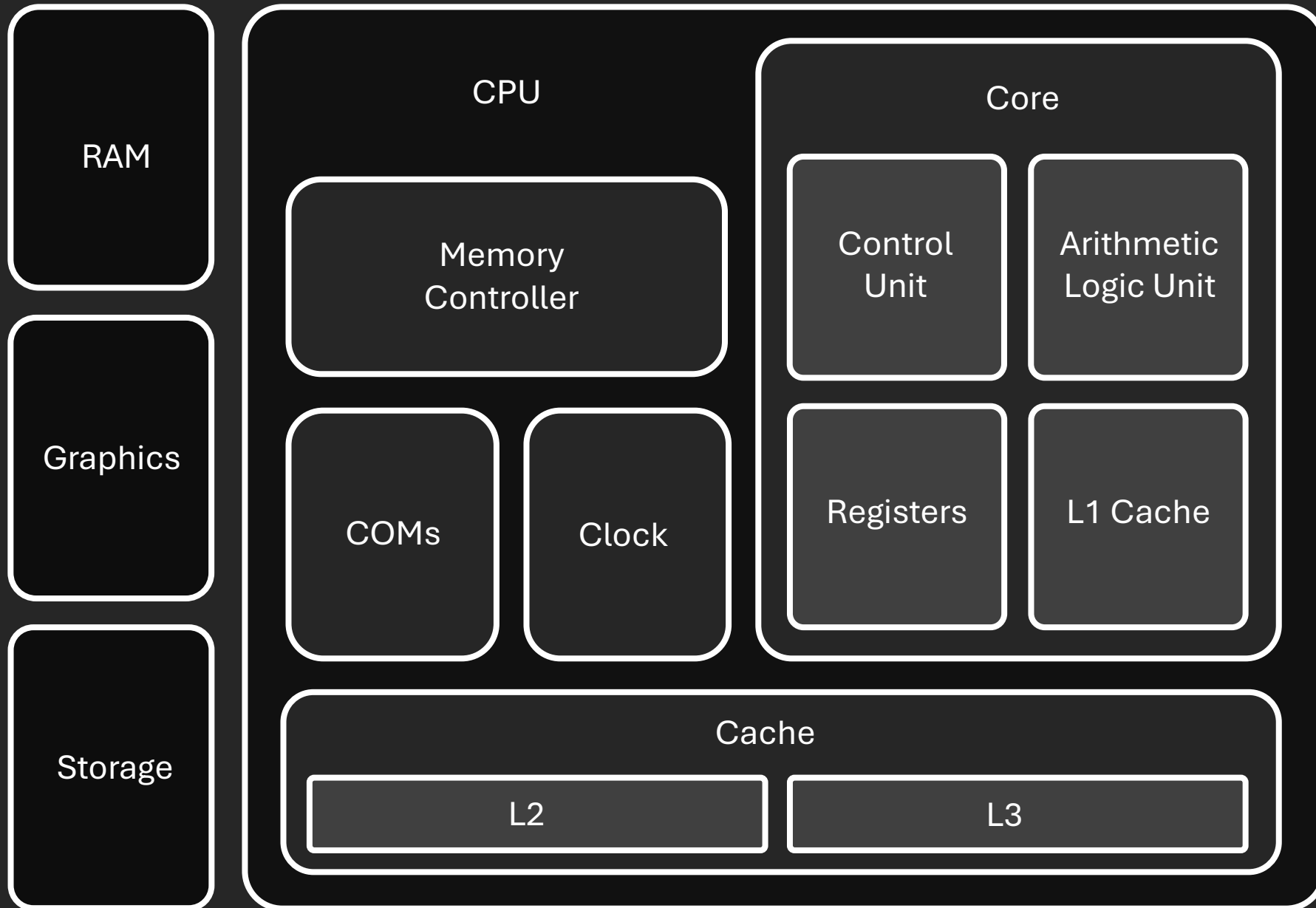
# How Does A CPU Work?

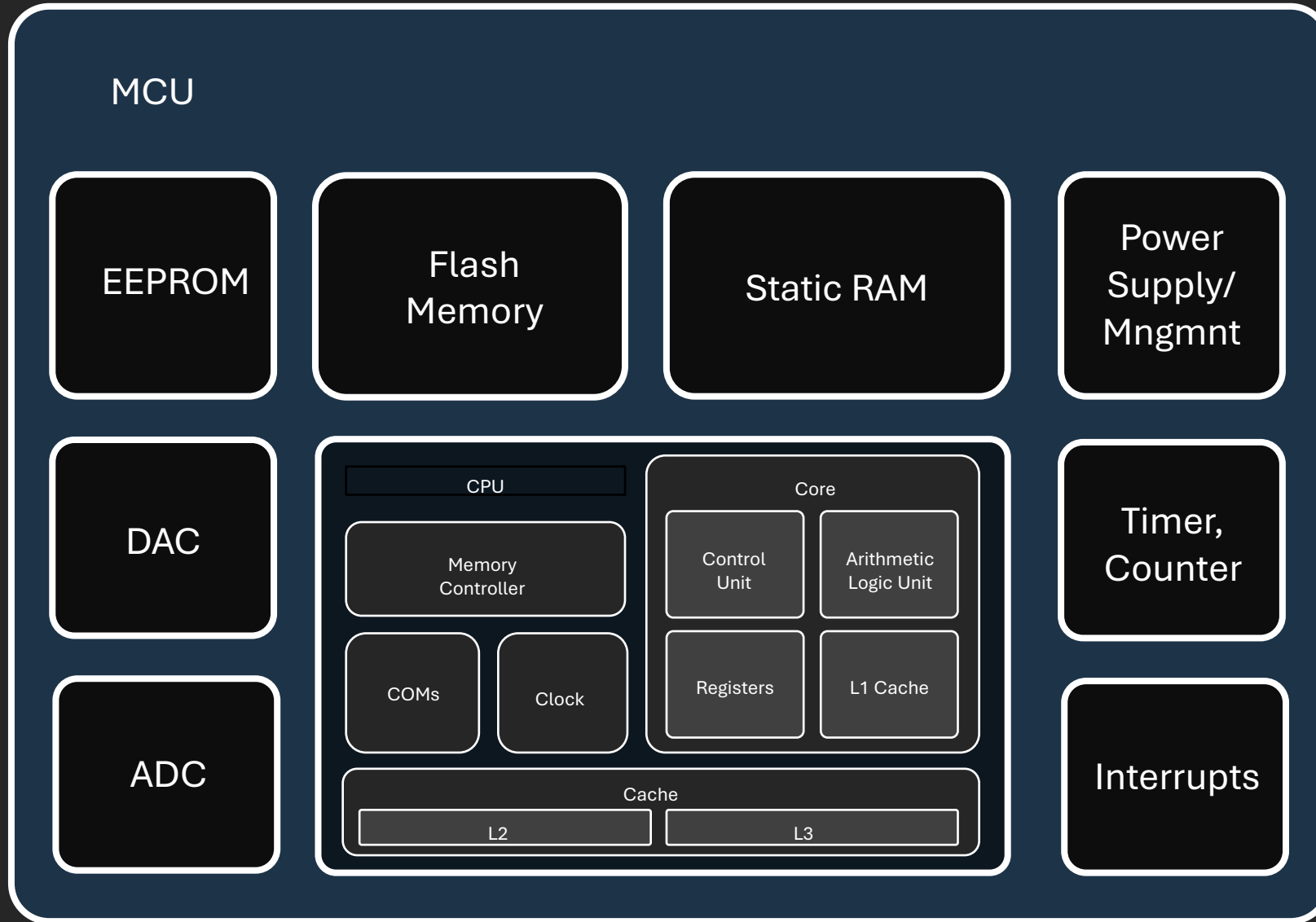


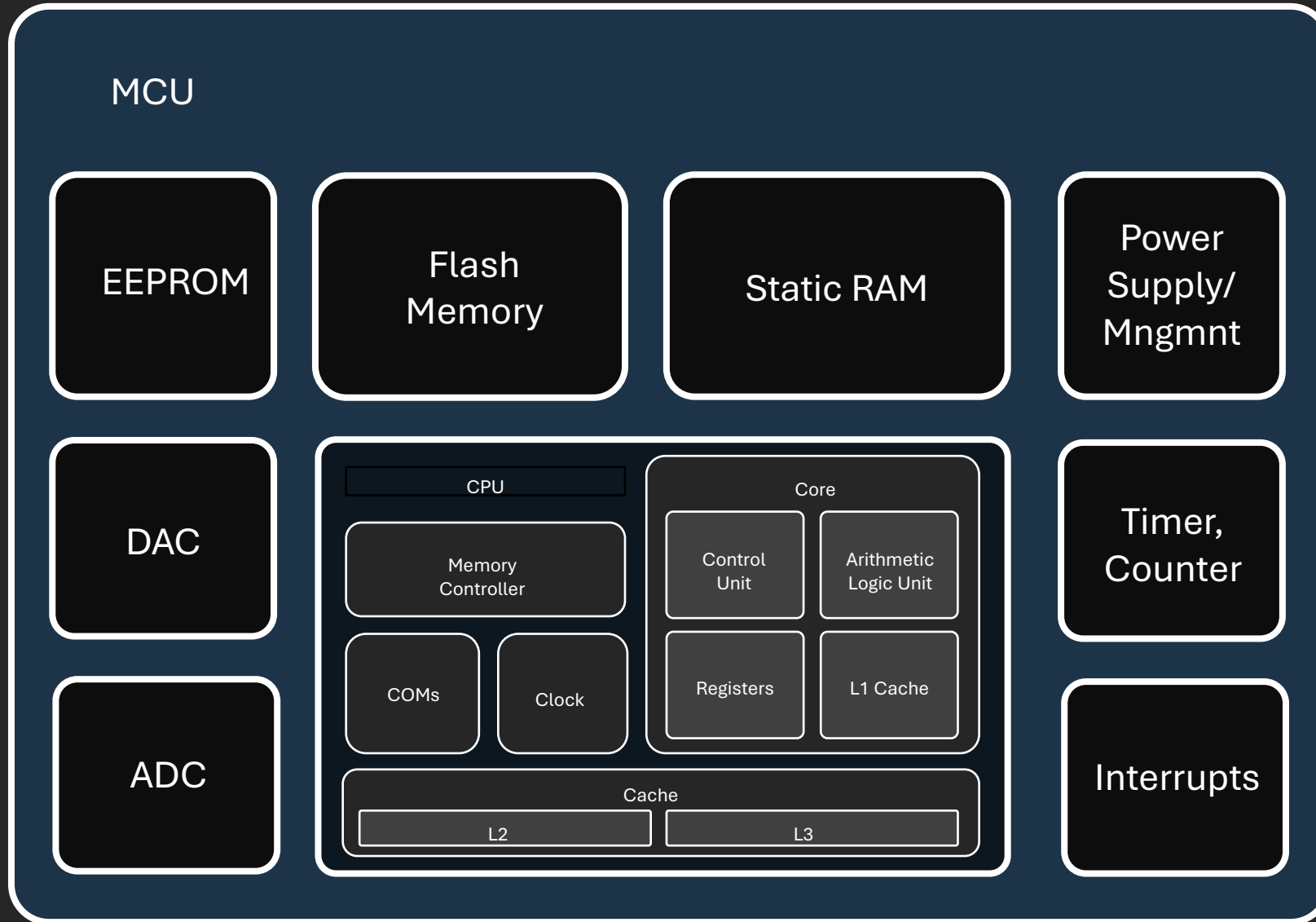
# How Does A CPU Work?

- Source Code:
  - Human-readable high-level language
- Assembly:
  - Low-level language with one-to-one relationship to instruction set
- Instruction Set Architecture:
  - Set of building blocks that define the capabilities of the processor
- Microarchitecture:
  - Physical circuit implementation of ISA
- Computer Architecture:
  - Overall design of the computer chip, including ISA, I/O, Memory, Bus









# Definitions

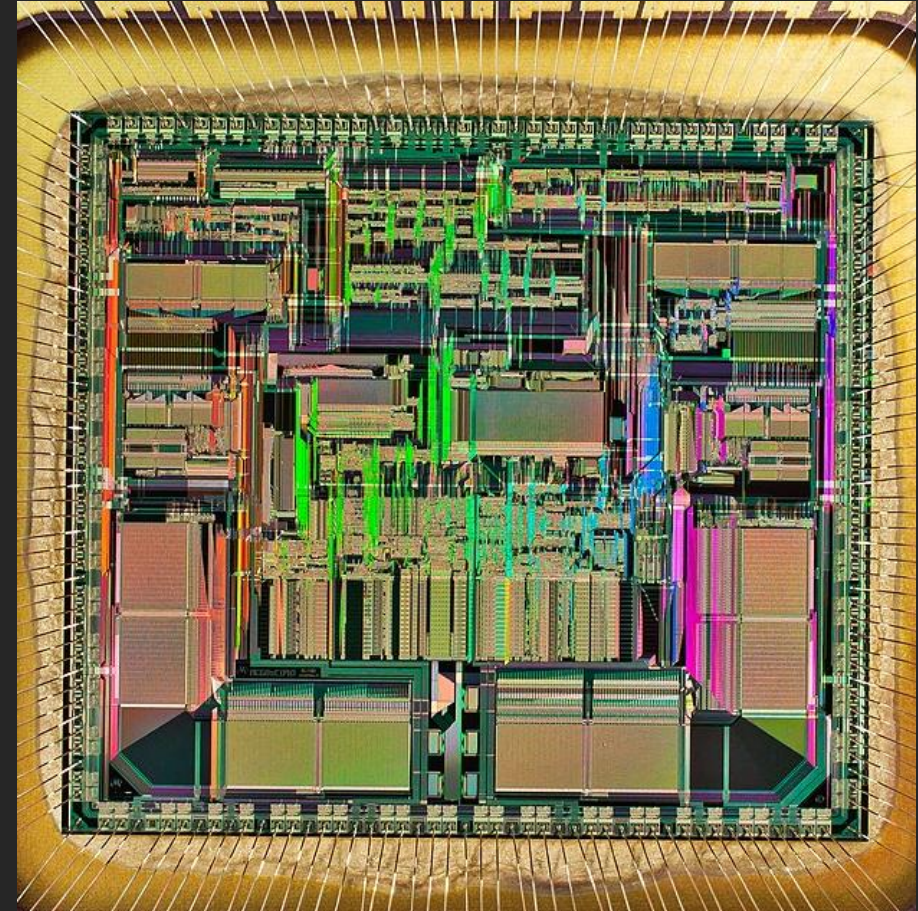
- RAM – Random access memory
  - Volatile, fast, computer memory
- EEPROM – Electronic Erasable Programmable Read-only Memory
  - Non-volatile, bite-level erasable, slow and less durable
  - Firmware, BIOS/UEFI settings, sensor and user settings
- Flash Memory
  - Non-volatile, slower than RAM, higher endurance, more storage, block erasure
  - Used in SSDs, memory cards, firmware storage



# When to Use

- RAM – Random access memory
  - General compute memory for when for fast larger volumes
- EEPROM – Electronic Erasable Programmable Read-only Memory
  - Very simple interface and control, lower power consumption, slow
  - Small, frequent updates
- Flash Memory
  - Faster, larger storage, cost effective
  - Large, bulk storage

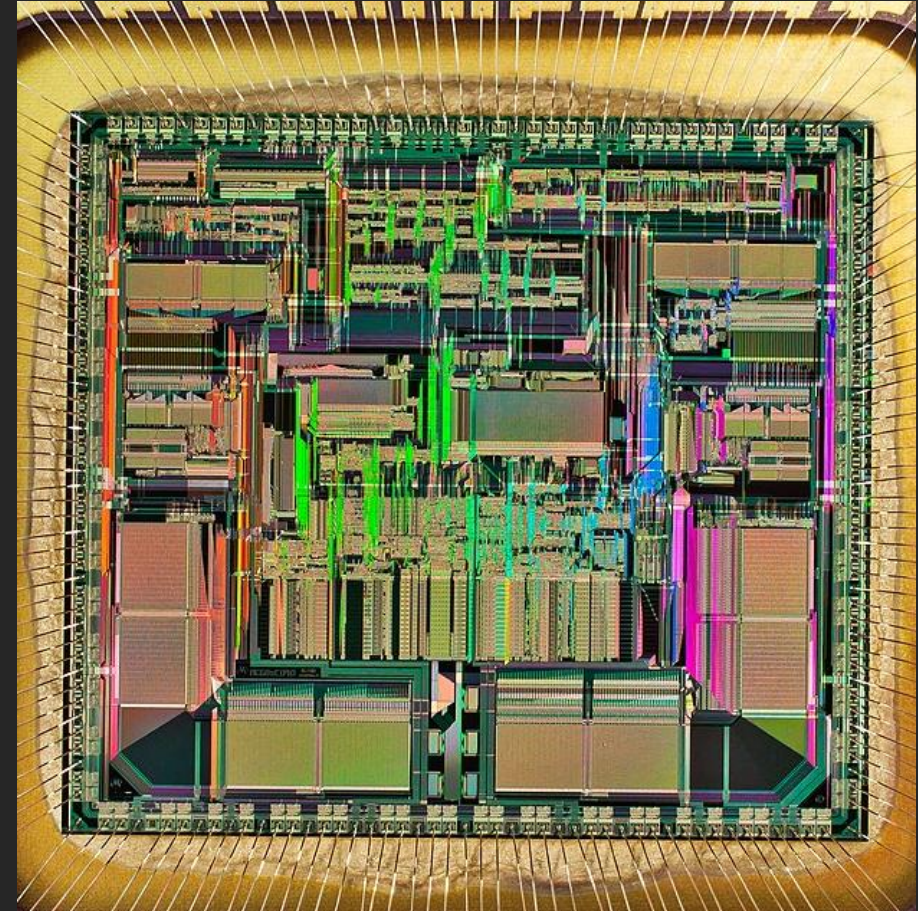
# Instruction Sets





# Instruction Sets

- Instruction Sets
  - Arithmetic Operations
  - Data Movement
  - Control Flow
  - Logic Operations
- Micro Architecture
  - How ISA is implemented in specific circuitry
- Computer Architecture
  - Structure of the all components



# Complex Instruction Set Computer

- CISC architectures have a large set of instructions
  - Specialized instructions
  - Many clock cycles per instruction
  - Instructions can directly manipulate memory
- Advantages
  - Fewer instructions are needed – Easier to write
- Disadvantages
  - Complex hardware, slower execution for simple operations
- Examples: x86 (Intel, AMD), System/360 (IBM)



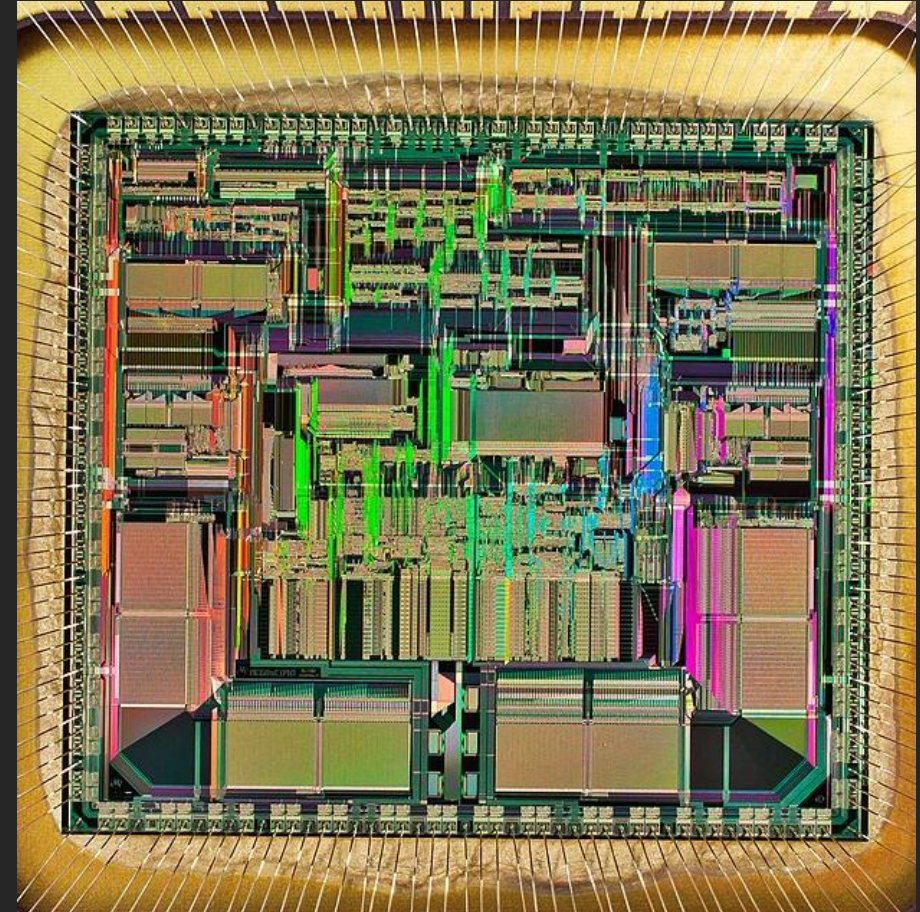
# Reduced Instruction Set Computer

- RISC architectures have a few set of fixed length instructions
  - Small and simple instruction set
  - Each instruction takes one clock cycle
  - Load/store registers independent of memory
- Advantages
  - Simple, fast execution, with simpler hardware
- Disadvantages
  - More instructions to accomplish a task
- Examples: ARM, RISC-V





# Instruction Sets



# Instruction Sets Examples

A loop to manually copy memory



```
MOV RCX, 100 ; Move 100 (number of elements) into RCX
MOV RSI, source ; Load source address into RSI
MOV RDI, dest ; Load destination address into RDI
REP MOVSB ; Copy 100 bytes from source to destination
```



```
MOV R0, source ; Load source address
MOV R1, dest ; Load destination address
MOV R2, #100 ; Set loop counter (100 bytes)

loop:
    LDRB R3, [R0], #1 ; Load byte from source, increment source pointer
    STRB R3, [R1], #1 ; Store byte to destination, increment destination pointer
    SUBS R2, R2, #1 ; Decrement counter
    BNE loop ; If counter not zero, repeat loop
```

# Source Code Languages

- Compiled (C/C++, Rust):
  - Converted to assembly and then assembled into machine code
- Interpreted (Python, JavaScript, PHP)
  - Analyzed line by line, each line is used to call pre-defined machine code
- Just In Time (JIT) (Java, C#)
  - Lines of code are analyzed one at a time, dynamically compiled and run
- Bytecode and Virtual Machines (Java, C#)
  - Code is compiled to intermediary “bytecode” and executed on a VM
- Scripting Languages (Bash, PowerShell)



# When to Use

- Compiled: System programming, embedded systems
  - High performance, efficient, slower development, platform specific
- Interpreted: Rapid prototyping, automation, scripting
  - Slower performance, portable, easy to use, fast iterations
- Just In Time: Web applications, enterprise applications
  - Dynamic optimization, portable, startup delays, high resource usage
- Bytecode and VMs: Cross-platform software, enterprise
  - Portable, slower than compiled, faster than interpreted, VM dependent
- Scripting Languages: Automation, system administration



# Programming Languages

- Complexity



```
// Hello World in C++  
#include <iostream>  
  
int main() {  
    std::cout << "Hello, World!" << std::endl;  
    return 0;  
}
```



```
# Hello World in Python  
print("Hello, World!")
```

# Operating Systems



# Kernal Space (privileged mode)

- Kernel – Core of the OS with full hardware access
  - Manages process scheduling, memory, file systems, system calls
  - Enforces Security and isolation
- Drivers – Kernel modules that enable hardware communication
  - Translate hardware-specific operations to standardized OS function
- Memory Management Unit – controls physical memory allocation
  - Maps processes virtual address to physical address
  - Isolates processes

# Kernal Space (privileged mode)

- System Calls – interface between user applications and the kernel
  - File I/O, network access, memory allocation
- Interrupts and Exceptions
  - Interrupts – events that trigger a response
  - Exceptions – faults in software
  - Incoming network packets, timer ticks, invalid memory access

# User Space (unprivileged mode)

- Applications – run with limited access to prevent accidental or malicious harm
  - Cannot directly access hardware, must use kernel
- Runtime libraries – provide high-level abstraction
  - Typically statically or dynamically linked to application



# Drivers – Bridging Software and Hardware

Specialized software program that allows the (OS) or firmware running on a CPU or MCU to interact with hardware devices

- Abstraction
  - Presents a simplified, uniform interface
- Communication Management
  - Command translation and data formatting
- Resource Management
  - System resource allocation and interrupt handling
- Error Handling

# Real Time Operating System (RTOS)

- Operating system designed to process data and execute tasks within strict timing constraints.
- Ensures **predictable, deterministic responses** to events.
- Not all microprocessors can run RTOS.



# Real Time Operating System (RTOS)

- Non-deterministic code execution
- Interrupt latency
- Incompatible memory
- Branch prediction and speculative execution

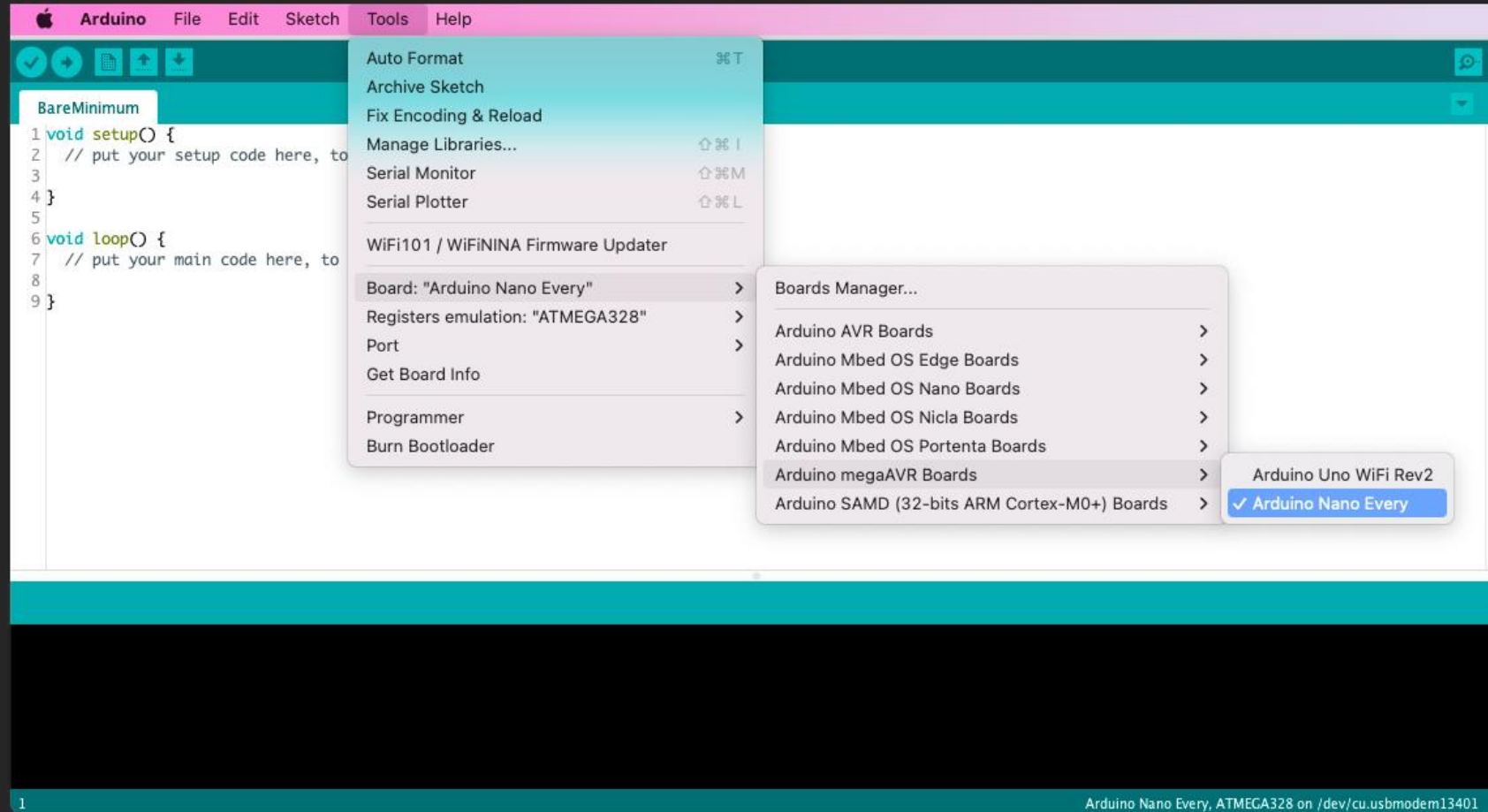
# Programming Microcontrollers



# Development Environment

- Manufacturer's IDE and Toolchain
  - Compiler (usually GCC based or propriety)
  - Debugger – hardware/software (Joint Test Action Group, Serial Wire Debug)
  - Peripheral and Code Config Tools (clocks, timers, communications)
- Set up the Project
  - Target microcontroller
  - Compiler Options
  - Startup code

# Arduino

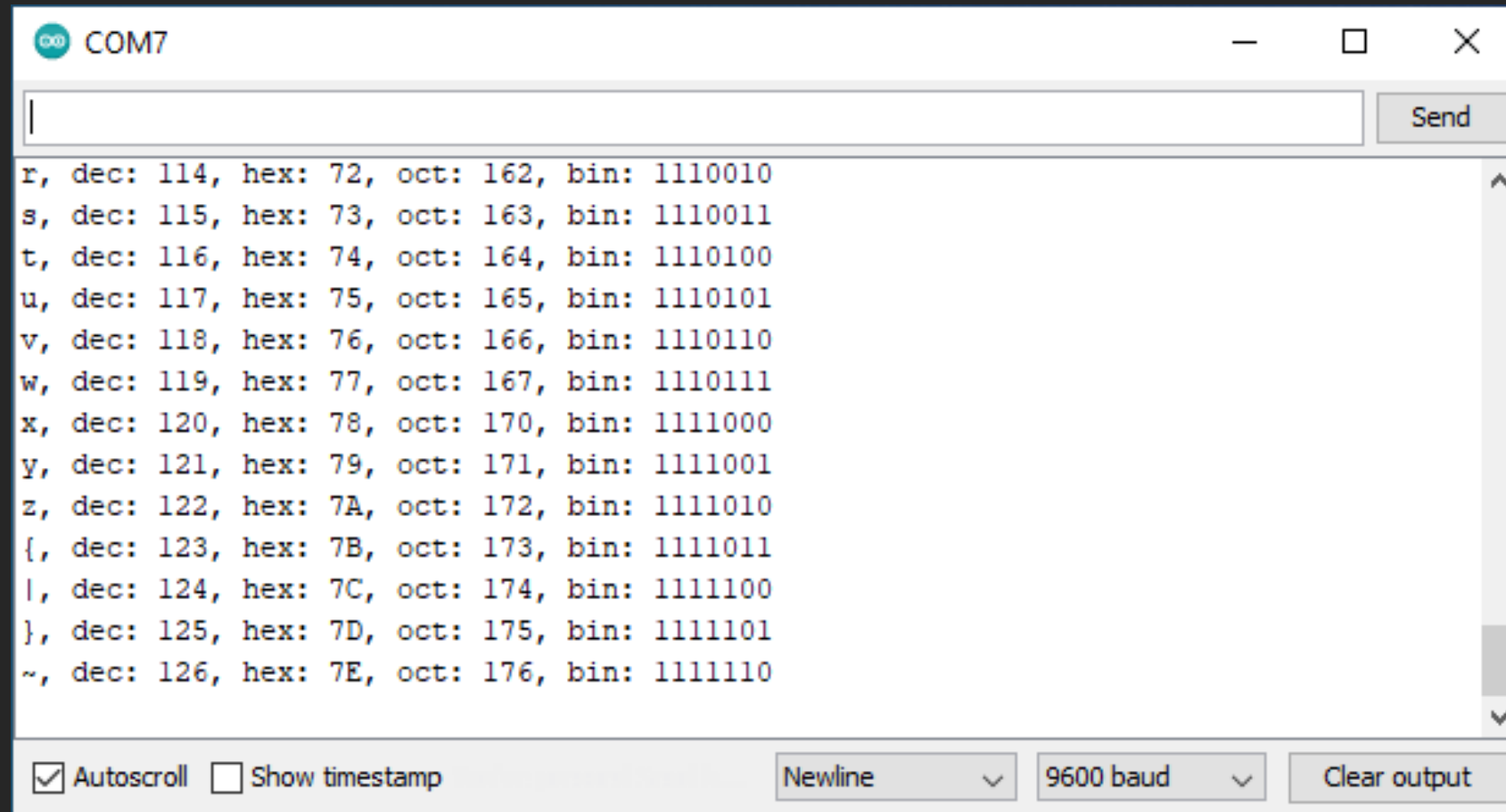


# Arduino



```
void setup() {  
    pinMode(LED_BUILTIN, OUTPUT); // Initialize the LED  
}  
  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH); // Turn the LED on  
    delay(1000); // Wait for 1 second  
    digitalWrite(LED_BUILTIN, LOW); // Turn the LED off  
    delay(1000); // Wait for 1 second  
}
```

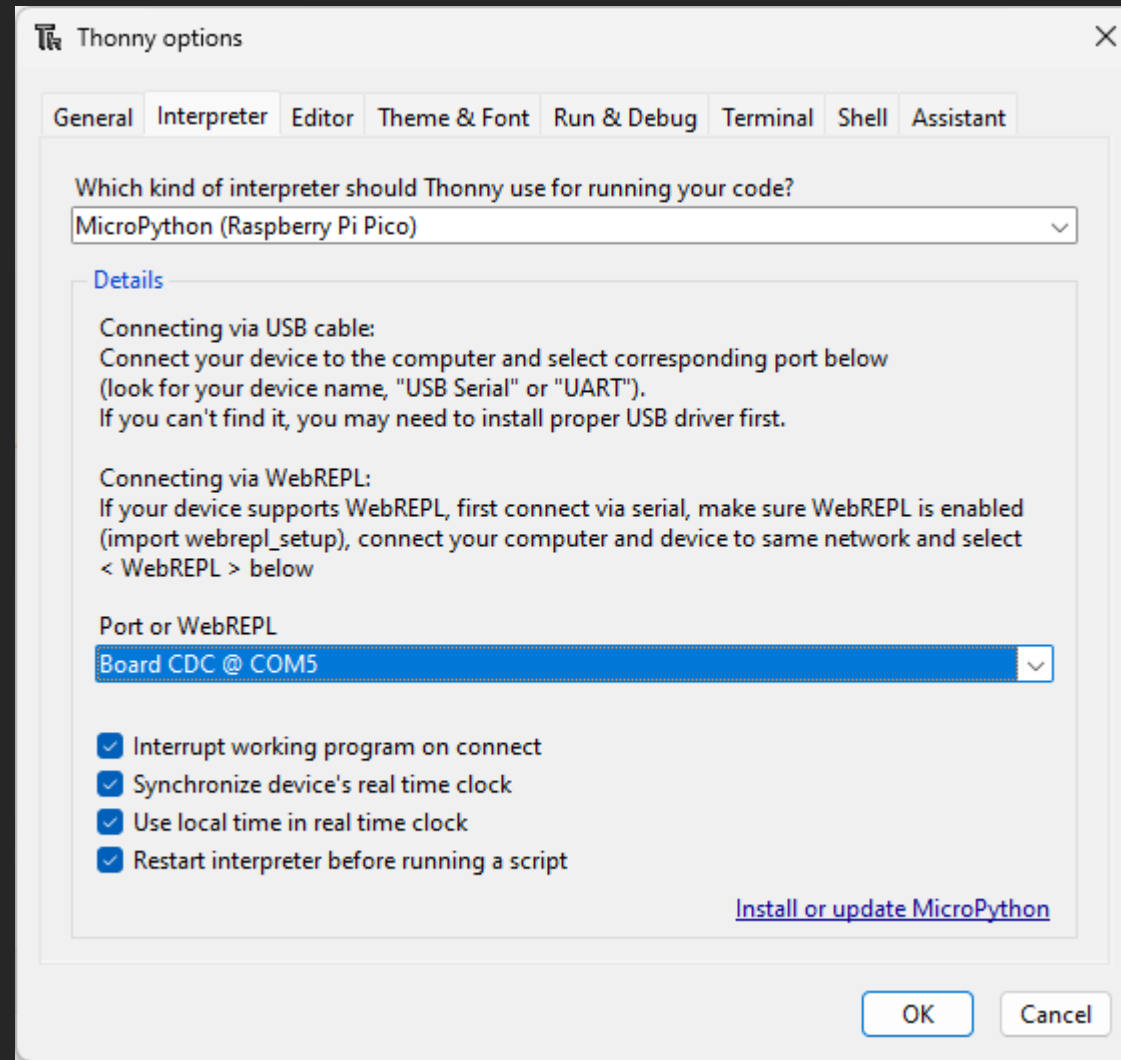
# Arduino



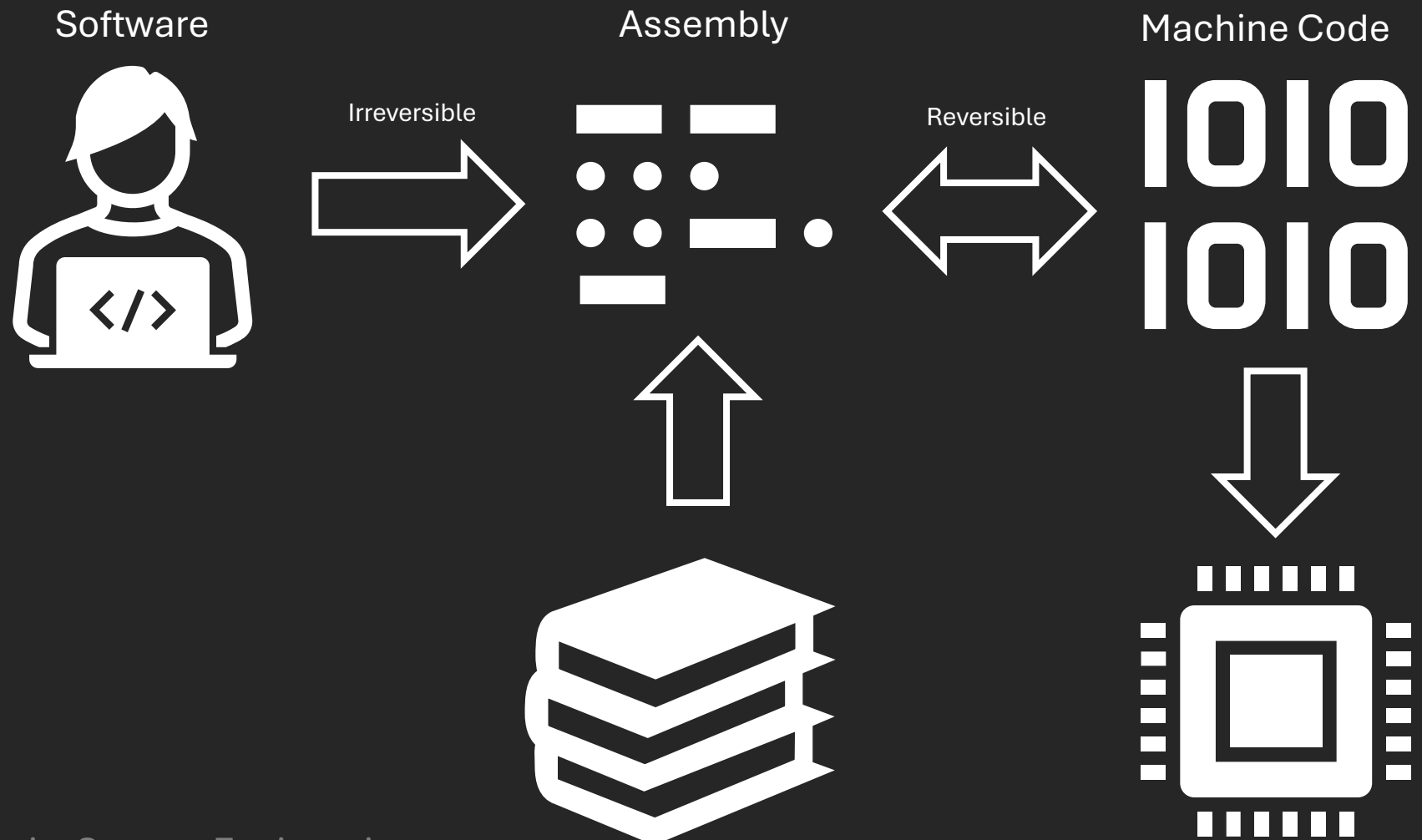
```
r, dec: 114, hex: 72, oct: 162, bin: 1110010
s, dec: 115, hex: 73, oct: 163, bin: 1110011
t, dec: 116, hex: 74, oct: 164, bin: 1110100
u, dec: 117, hex: 75, oct: 165, bin: 1110101
v, dec: 118, hex: 76, oct: 166, bin: 1110110
w, dec: 119, hex: 77, oct: 167, bin: 1110111
x, dec: 120, hex: 78, oct: 170, bin: 1111000
y, dec: 121, hex: 79, oct: 171, bin: 1111001
z, dec: 122, hex: 7A, oct: 172, bin: 1111010
{, dec: 123, hex: 7B, oct: 173, bin: 1111011
|, dec: 124, hex: 7C, oct: 174, bin: 1111100
}, dec: 125, hex: 7D, oct: 175, bin: 1111101
~, dec: 126, hex: 7E, oct: 176, bin: 1111110
```



# Thonny



# Microprocessor



# Microprocessor

