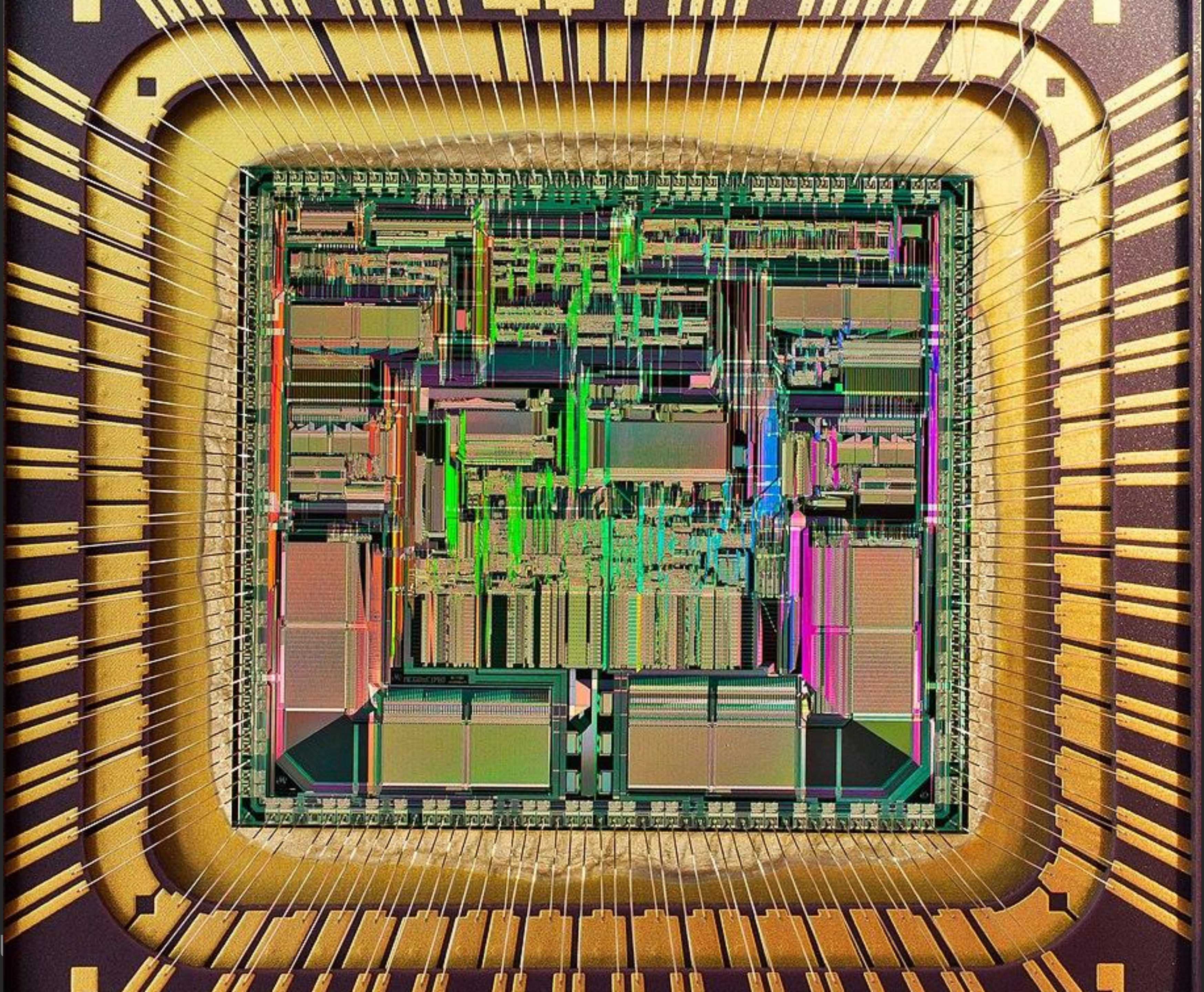
Cyber-Physical Systems

Dr. Jonathan Jaramillo

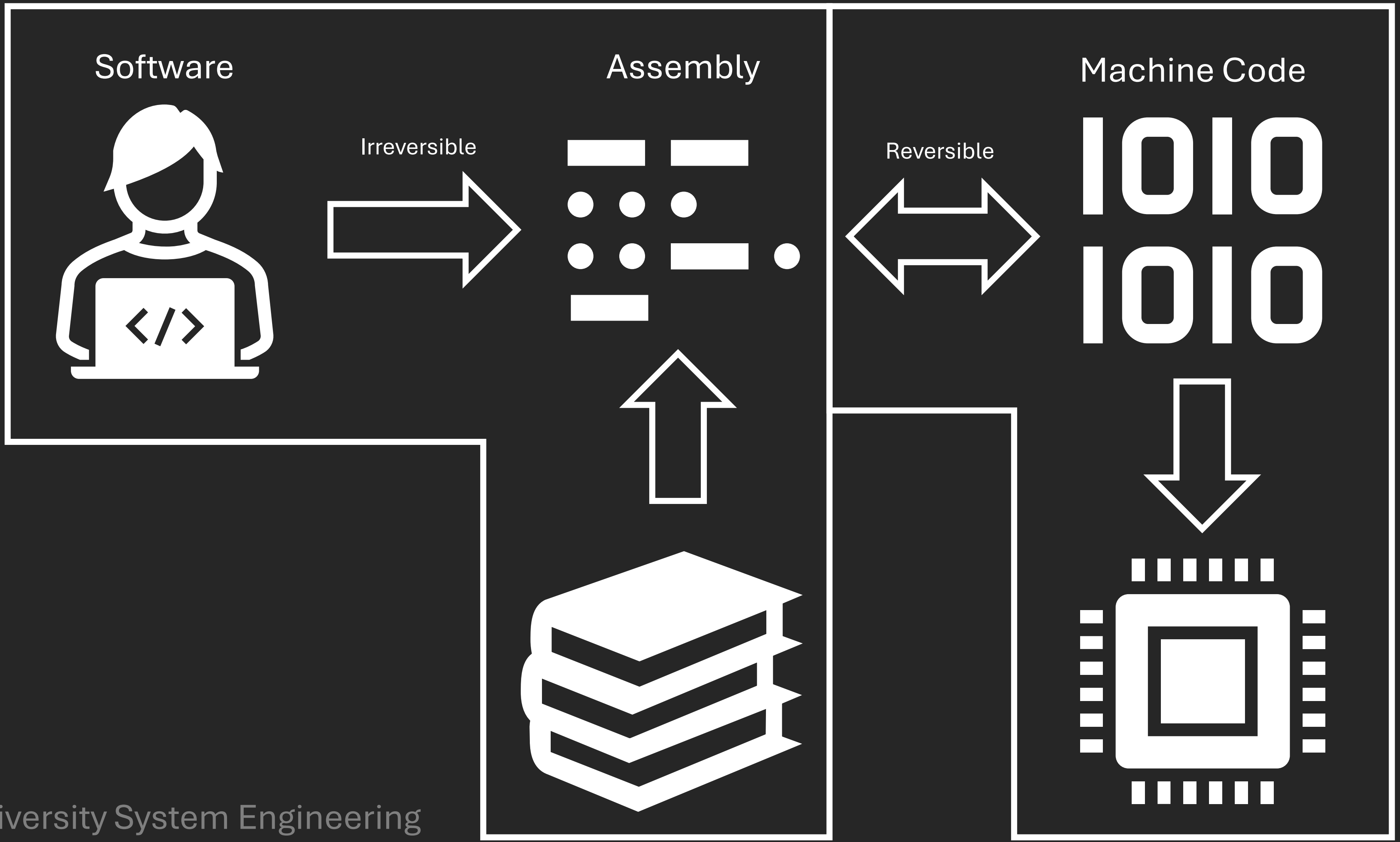
Computer Architecture



How Does A CPU Work?

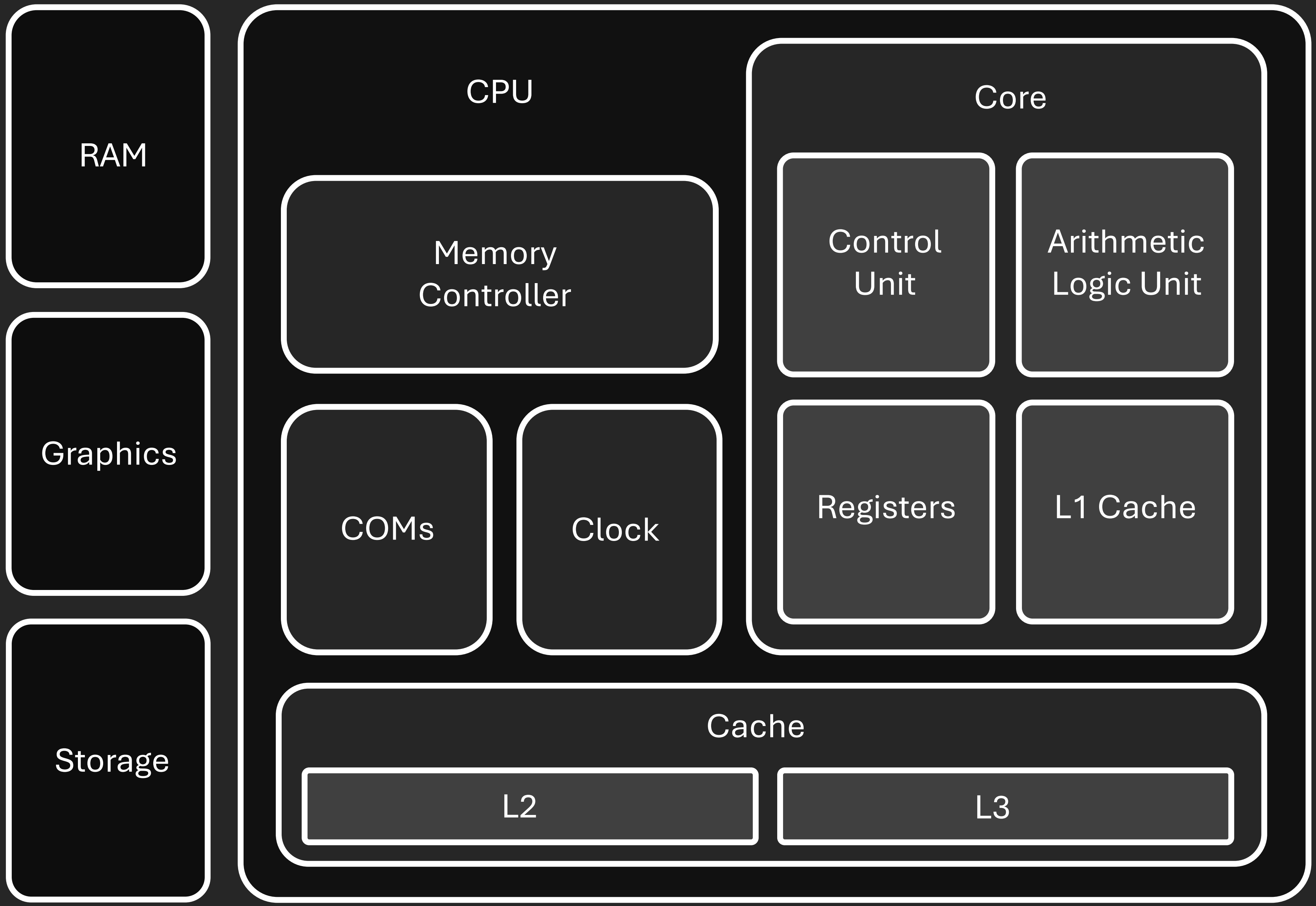
- Compiler: Source code is compiled to assembly language.
- Assembler: Assembly is converted to machine code.
- Linker: Machine code is combined with libraries to create an executable file.

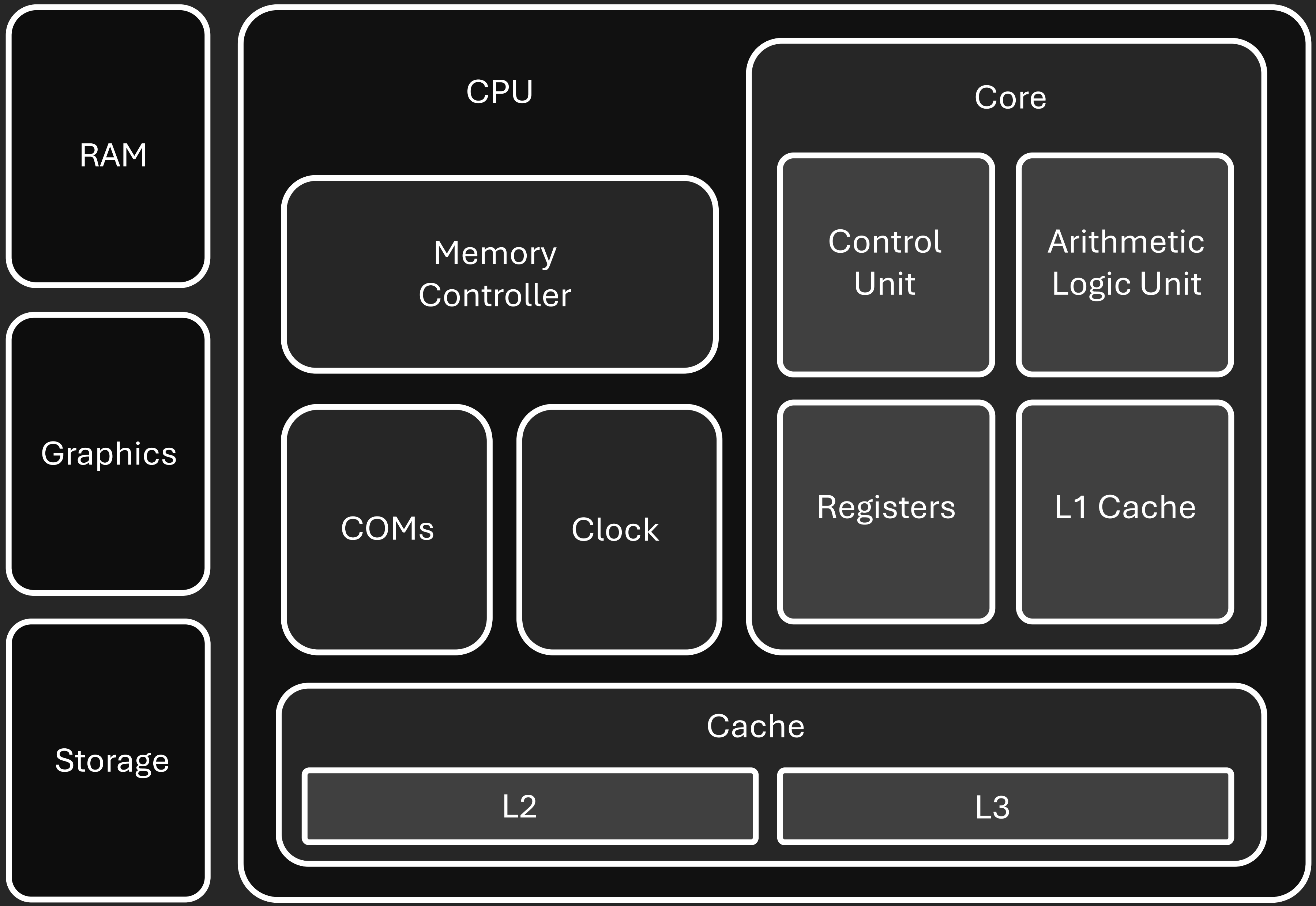
How Does A CPU Work?

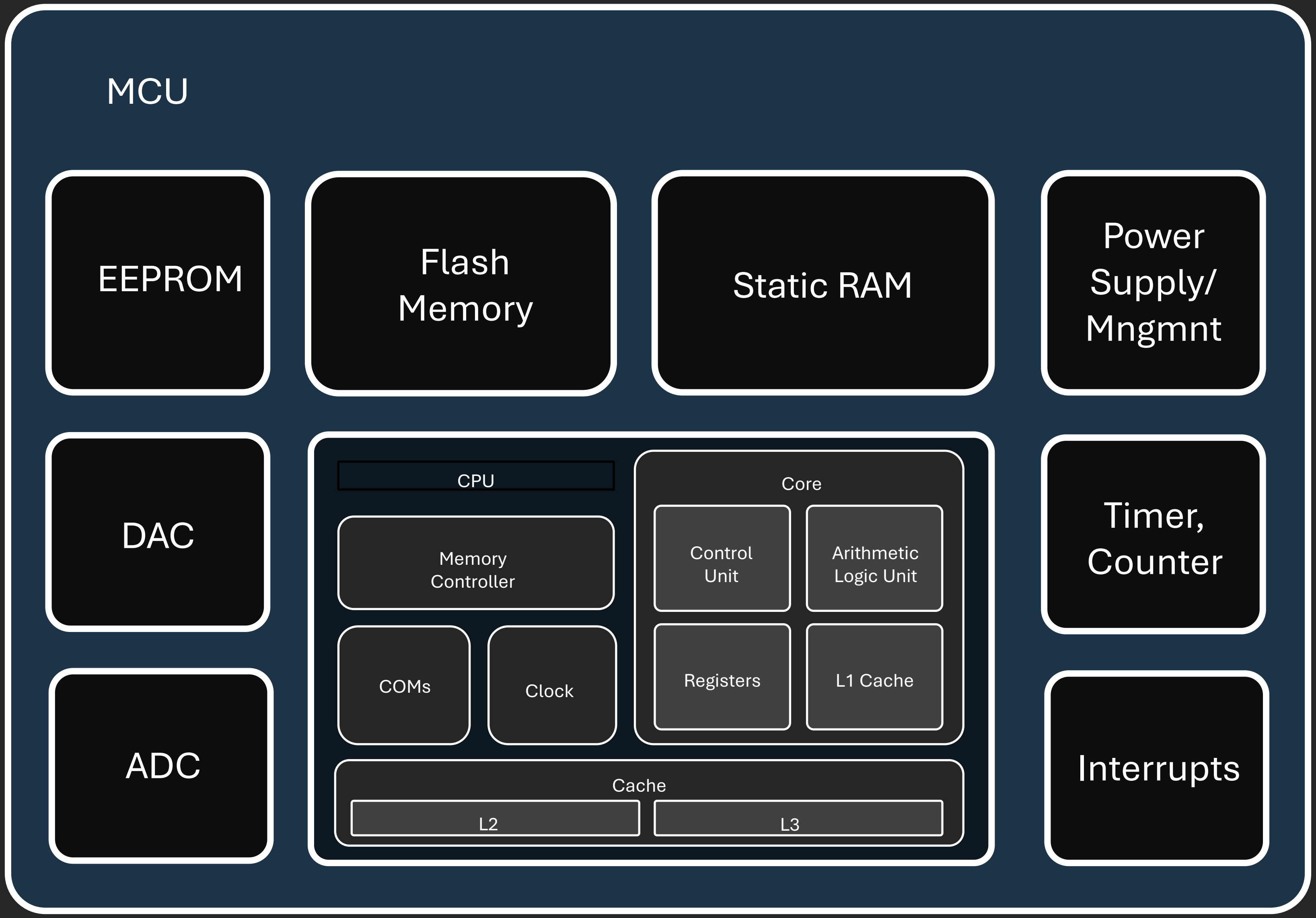


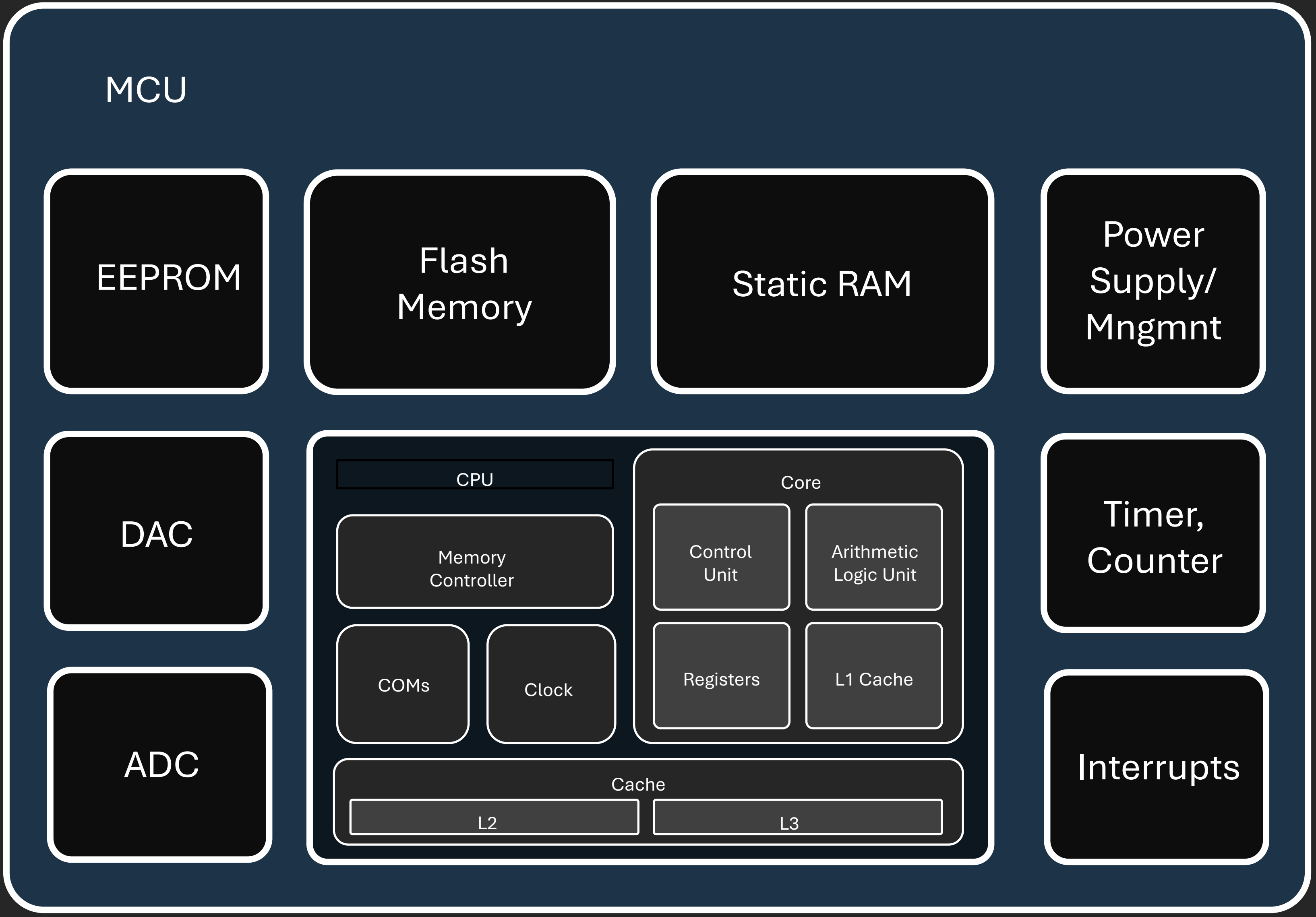
How Does A CPU Work?

- 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


RAM
Graphics
Storage
CPU
Memory Controller
COMs
Clock
Core
Control Unit
Arithmetic Logic Unit
Registers
L1 Cache
Cache
L2
L3


RAM
Graphics
Storage
CPU
Memory Controller
COMs
Clock
Core
Control Unit
Arithmetic Logic Unit
Registers
L1 Cache
Cache
L2
L3


MCU
EEPROM
Flash Memory
Static RAM
Power Supply/ Mngmnt
DAC
CPU
Memory Controller
COMs
Clock
Core
Control Unit
Arithmetic Logic Unit
Registers
L1 Cache
Cache
L2
L3
Timer, Counter
ADC
Interrupts


MCU
EEPROM
Flash Memory
Static RAM
Power Supply/ Mngmnt
DAC
CPU
Memory Controller
COMs
Clock
Core
Control Unit
Arithmetic Logic Unit
Registers
L1 Cache
Cache
L2
L3
Timer, Counter
ADC
Interrupts

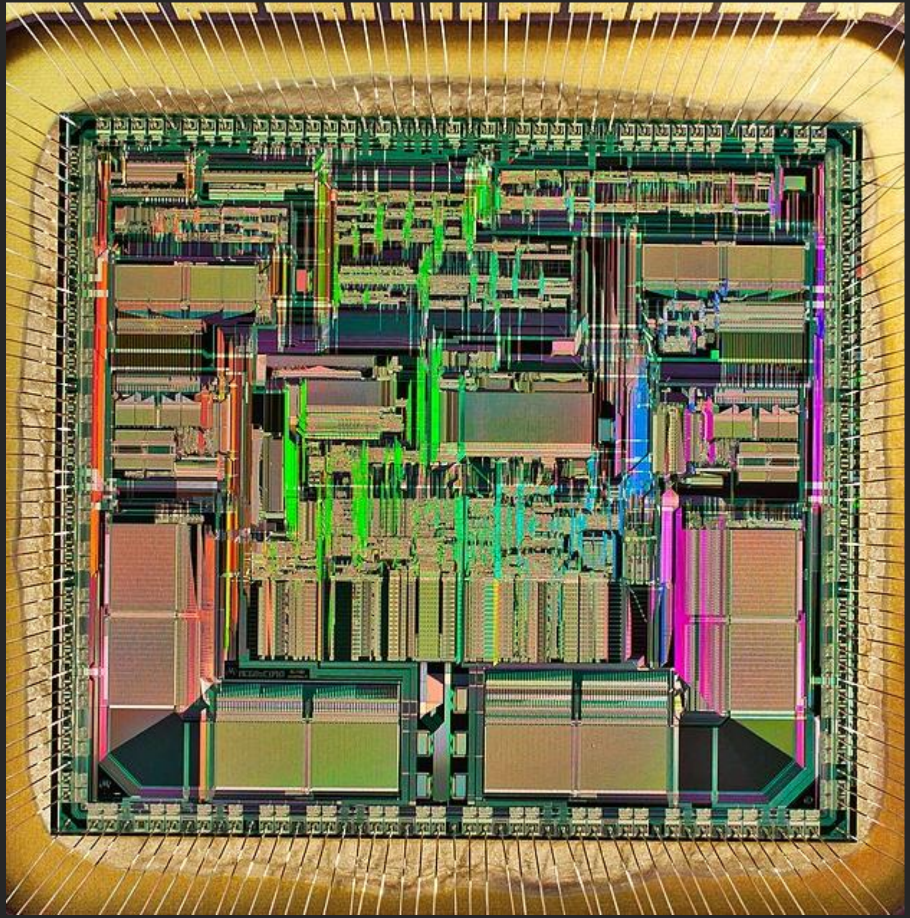
Definitions

- RAM – Random access memory
  - Volatile, fast, computer memory
- EEPROM – Electronic Erasable Programable 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

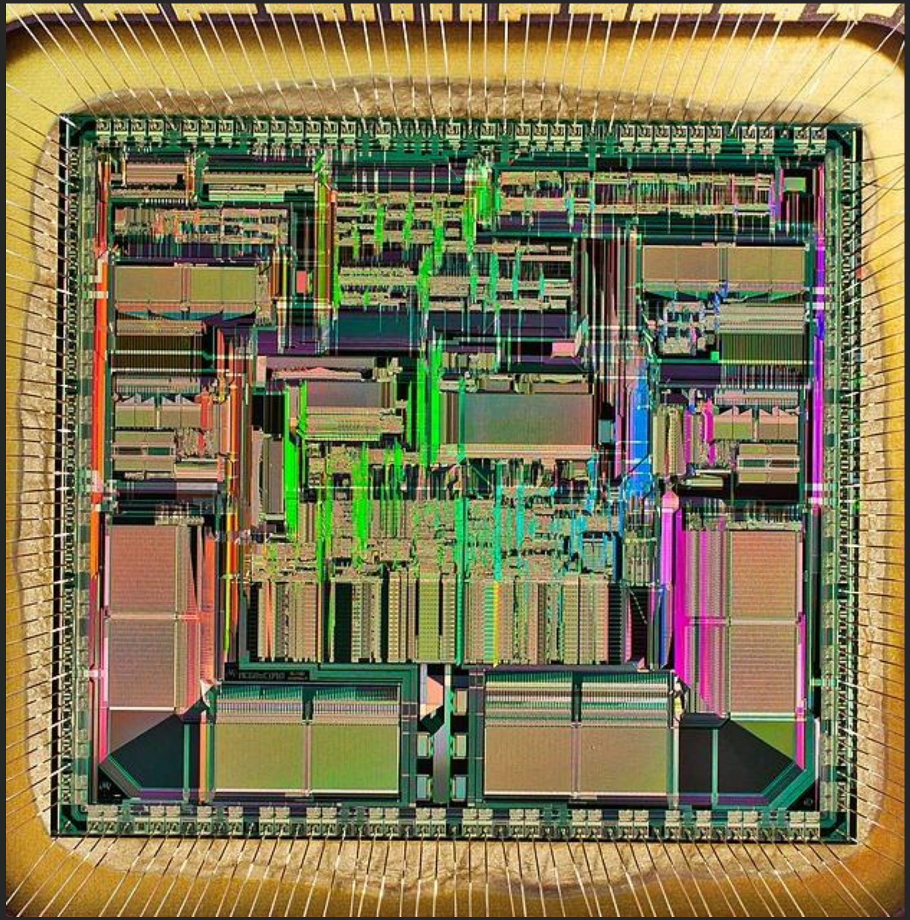
- RAM – Random access memory
  - General compute memory for when for fast larger volumes
- EEPROM – Electronic Erasable Programable Read-only Memory
  - Very simple interface and control, lower power consumption, slow
  - Small, frequent updates
- Flash Memory
  - Faster, larger storage, cost effective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



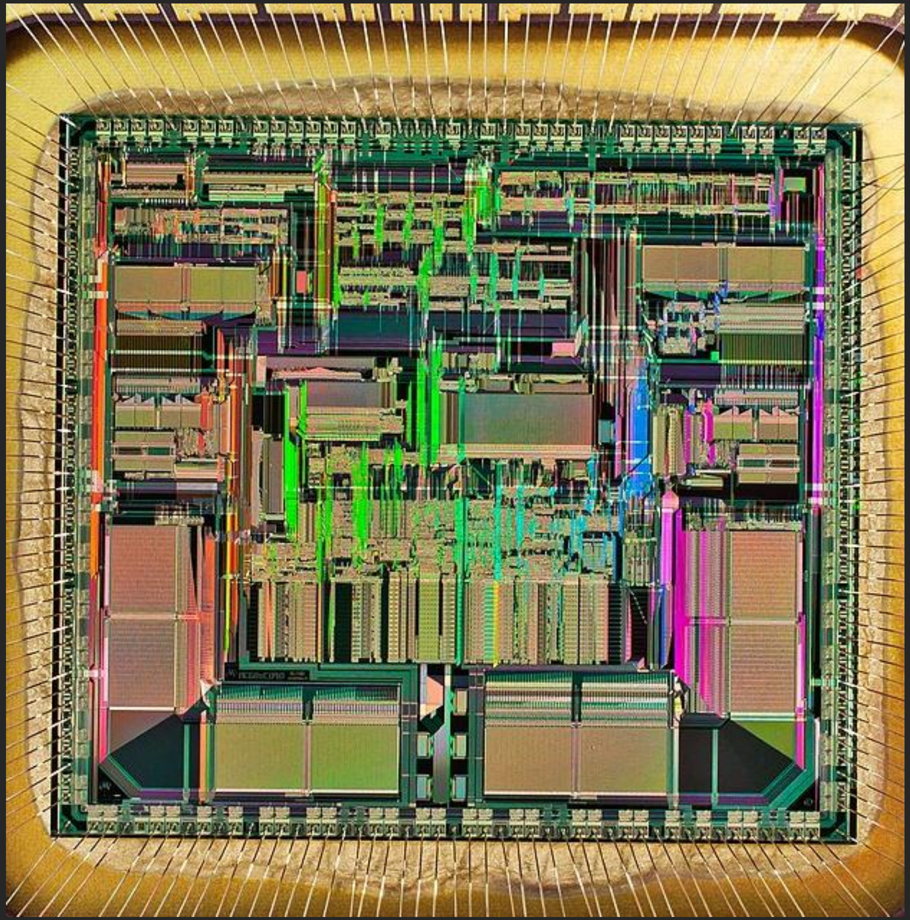
Complex Instruction Set Computer

- 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 (Jaca, 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

```cpp
// Hello World in C++
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

```python
# 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

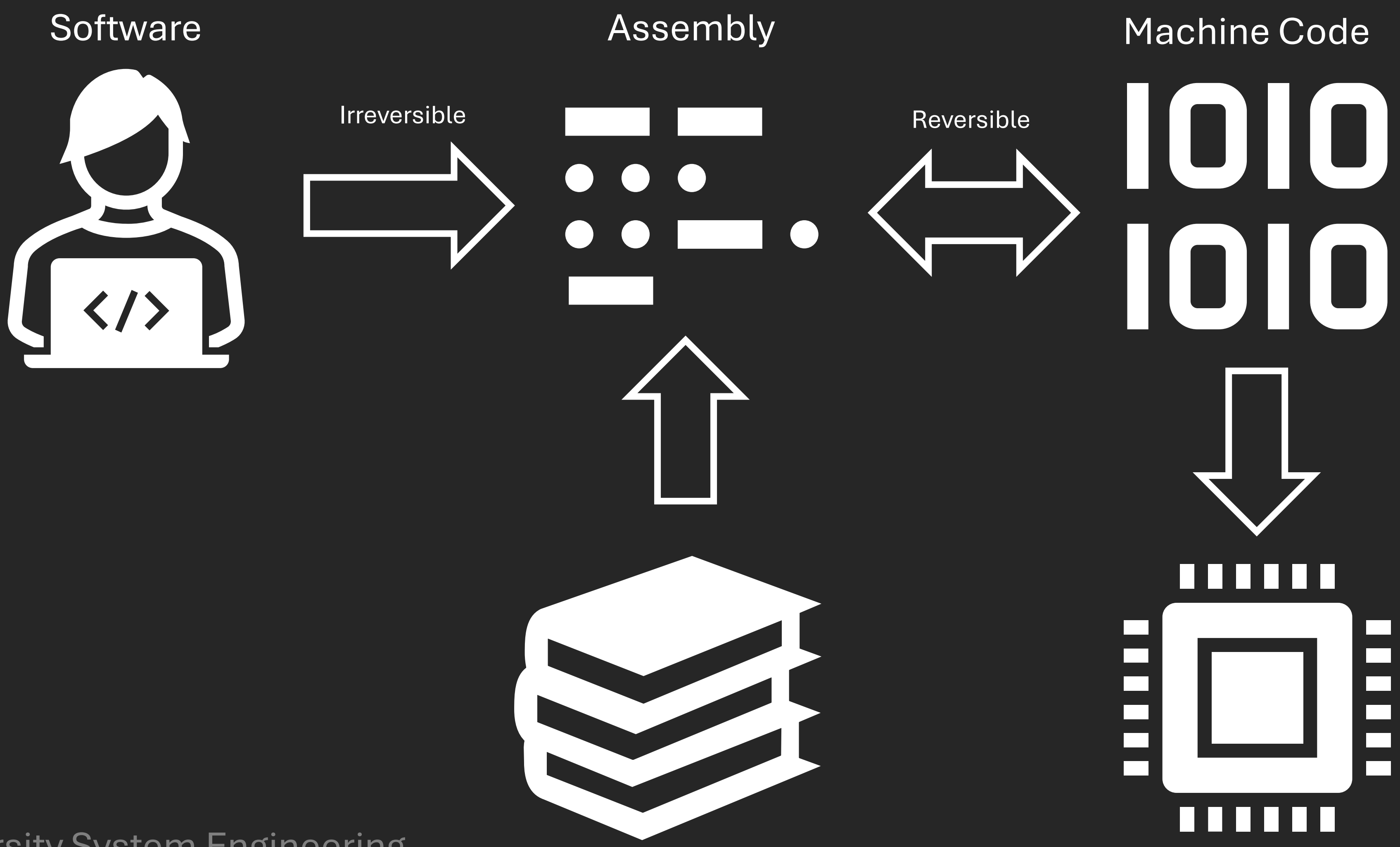
Boot Loaders

- Small program on dedicated memory designed to load and run the main application.
- Key Functions
  - Initialization – start up main application and hardware
  - Boot Management – determines where to find and load firmware
  - Firmware updates – uses USB, UART, SPI, etc. to upload new firmware
  - Fail-safe – can revert to safe or backup firmware in case of corruption

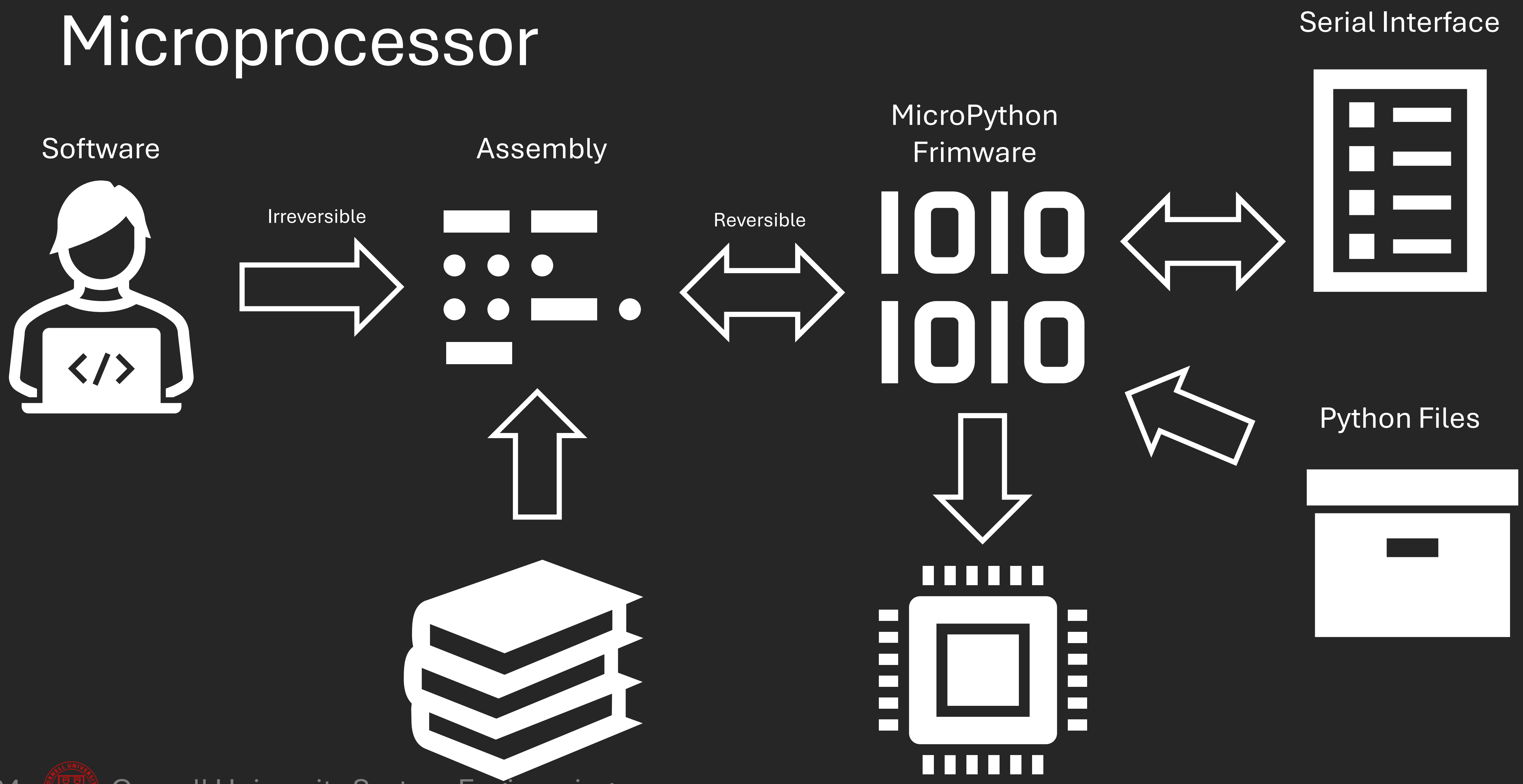
BOOTSEL, RESET, or BOOT0

1. Boot from main flash memory
    - Default behavior
    - Load firmware from flash
2. Boot into bootloader mode
    - When BOOTSEL is pushed during power up
    - Receives new firmware over communication interface
    - Performs diagnostic and recovery procedures

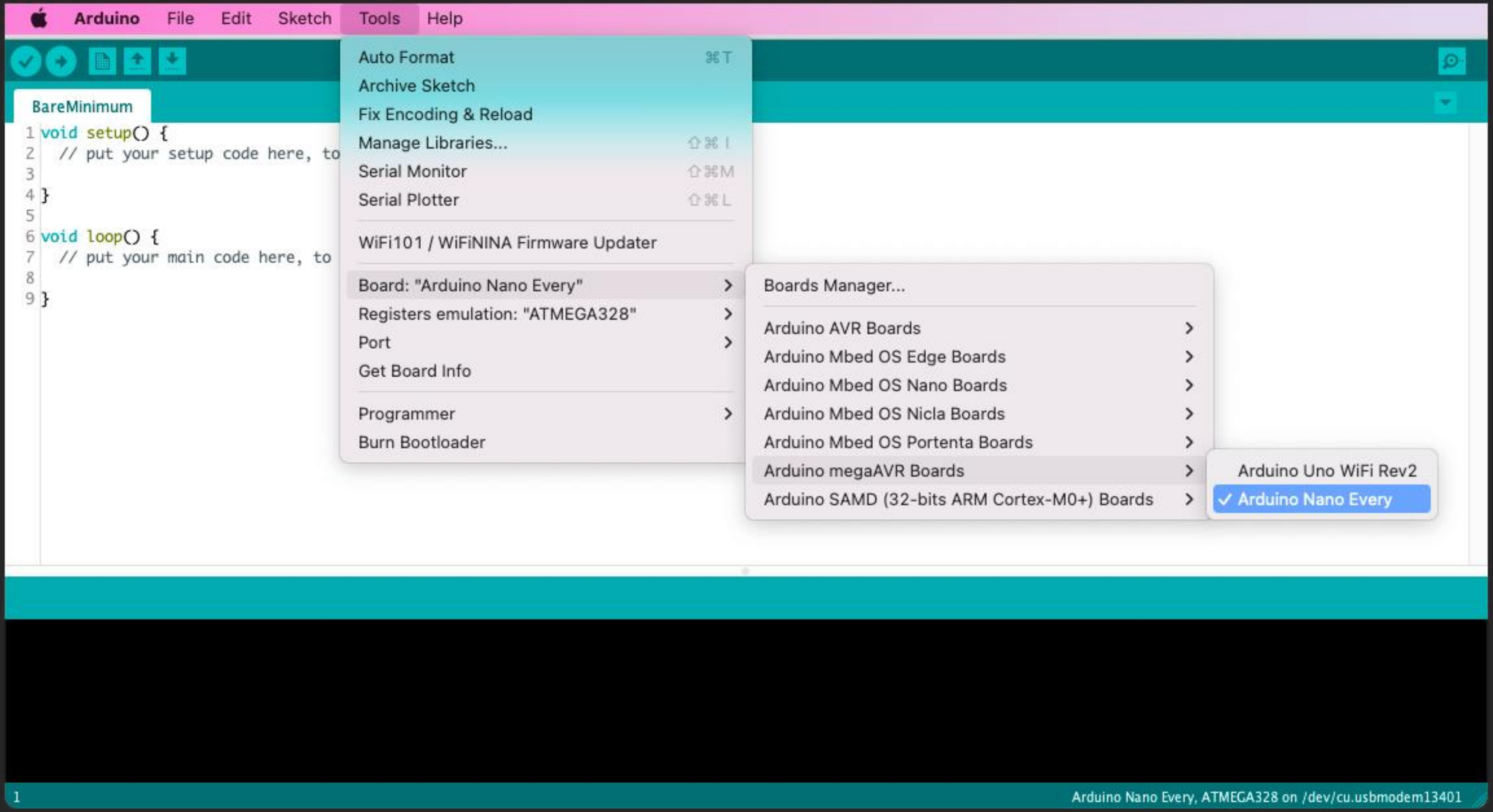
Microprocessor



Microprocessor



Arduino

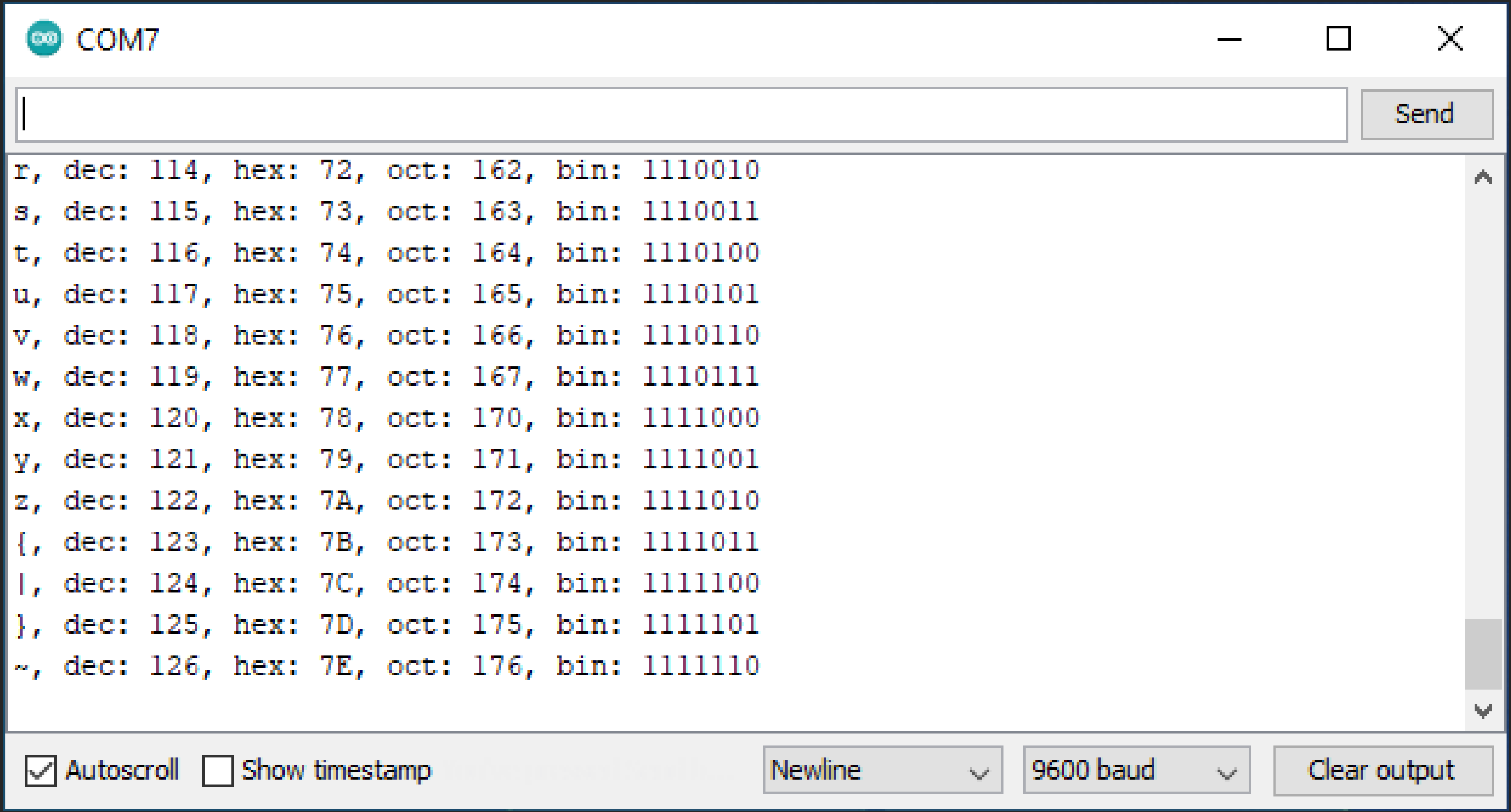

Arduino File Edit Sketch Tools Help
BareMinimum
void setup() {
// put your setup code here, to
}
void loop() {
// put your main code here, to
}
Auto Format ⌘T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... ⇧⌘I
Serial Monitor ⇧⌘M
Serial Plotter ⇧⌘L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano Every"
Registers emulation: "ATMEGA328"
Port
Get Board Info
Programmer
Burn Bootloader
Boards Manager...
Arduino AVR Boards
Arduino Mbed OS Edge Boards
Arduino Mbed OS Nano Boards
Arduino Mbed OS Nicla Boards
Arduino Mbed OS Portenta Boards
Arduino megaAVR Boards
Arduino SAMD (32-bits ARM Cortex-M0+) Boards
Arduino Uno WiFi Rev2
✓ Arduino Nano Every
1
Arduino Nano Every, ATMEGA328 on /dev/cu.usbmodem13401

Arduino

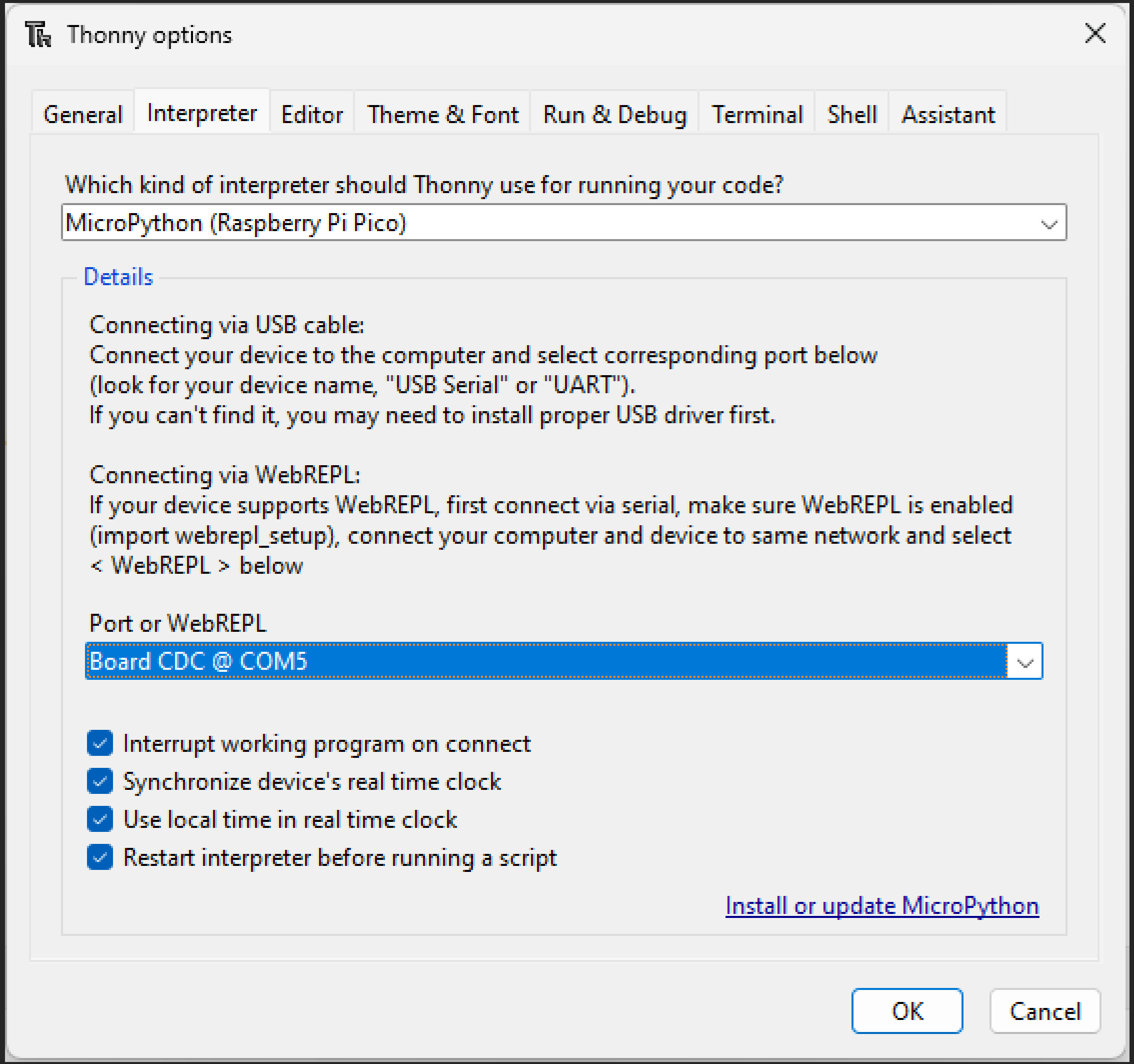
```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);  // Initialize the  LED
}

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


COM7
Send
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
Autoscroll
Show timestamp
Newline
9600 baud
Clear output

Thonny



Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant
Which kind of interpreter should Thonny use for running your code?
MicroPython (Raspberry Pi Pico)
Details
Connecting via USB cable:
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.
Connecting via WebREPL:
If your device supports WebREPL, first connect via serial, make sure WebREPL is enabled
(import webrepl_setup), connect your computer and device to same network and select
< WebREPL > below
Port or WebREPL
Board CDC @ COM5
Interrupt working program on connect
Synchronize device's real time clock
Use local time in real time clock
Restart interpreter before running a script
Install or update MicroPython
OK
Cancel