**IoT & Automation Lab.**

**Assignment-1**

1. **What is a Prototype? What are Open source and closed source prototype platforms?**

A prototype is a basic, rough version of a product created to test ideas and show how it works. It helps understand if the design is right and gathers feedback before making the final version.

**Open-source software** is software whose code is open to everyone. Everybody can access, use, modify, or distribute it freely. This fosters community inputs and helps in team working. Some examples of these are Firefox, MySQL, and Arduino.

**Closed-source software** is just software whose code is kept private and owned by a particular company. The public cannot access or modify it, and more often than not, some license or subscription fees are paid for it. Examples include Skype, Microsoft Office, and Adobe Reader.

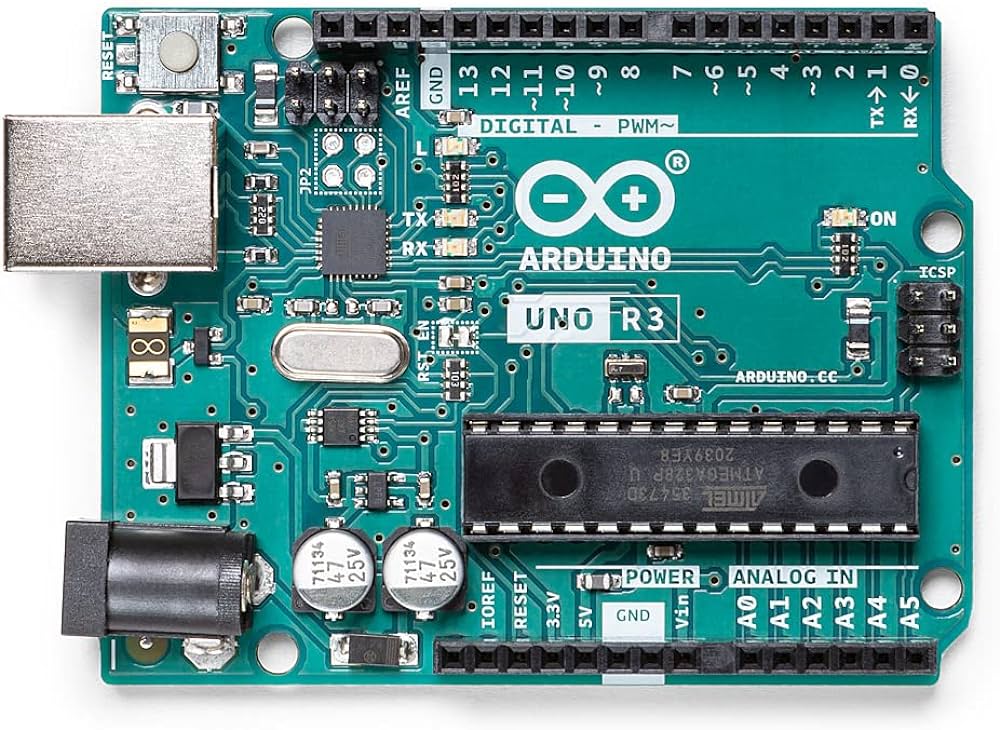
1. **What is Arduino?**

Arduino is an open-source electronic prototyping platform allowing easy creation and development of both hardware and software. This helps in making a project interactive by reading inputs example, press a button and making outputs, such as turning on a light. It has found wide application in the education sector, prototyping new ideas, and DIY projects, such as home automation and wearable technology.

Top of Form

Bottom of Form

1. **Write down Arduino Uno R3 Key Specifications:**



**Main Processor:**

* **Microcontroller:** ATmega328P (8-bit RISC processor)

**Memory:**

* **SRAM:** 2 KB (for temporary data)
* **Flash Memory:** 32 KB (for storing the application code; 0.5 KB used by the bootloader)
* **EEPROM:** 1 KB (non-volatile memory for storing data even after power-off)

**I/O Pins:**

* **Digital I/O Pins:** 14 (6 can provide PWM output)
* **Analog Input Pins:** 6
* **PWM Output Pins:** 6 (Pins 3, 5, 6, 9, 10, 11)
* **UART:** 1 (Serial communication on pins 0 (RX) and 1 (TX))
* **Built-in LED:** Pin 13

**Assignment-2**

**What is an Encoding format? List down encoding formats for various types of data (Text, Number, Photo, Audio, Video).**

Encoding format is a standardized way to convert data into a format that computers can read and process. It’s similar to translating human language into a language that computers can understand.

**Different Encoding Formats:**  
**Text Encoding:**  
ASCII: Stands for 128 characters, to be more precise letters, numbers, and other punctuation marks.  
Unicode: A general type of character set standard that characterize most of the commonly used language as well as the ASCII.  
UTF-8: A kind of variable encoded character code that is maintained between ASCII and the web\_standards.

**Number Encoding:**  
Binary: Functions only with 0s and 1s only.  
Decimal: The system that we us in our daily lives for instance, the base-10 system.  
Hexadecimal: Alphabetic characters represented by erotiske by 16 digits from 0 to 9 and A-F.  
Floating-point: Can be real numbers written in hindu numerals with a decimal point.

**Image Encoding:**  
JPEG: Some examples include lossy compression, which is best used for photos.  
PNG: It is a type of compression which is best used for images having clear cut edges and texts.  
GIF: Supports animation and transparency and is used for the simple images.  
BMP: A format with no compression which hence means that the size of the file is large.  
TIFF: There is no information loss when the images are compressed, it supports different image bit depths.

**Audio Encoding:**  
MP3: Lossy compression the most used format for audio, especially for music.  
AAC: Broadcasting and iTune’s uses lossy compression.  
WAV: Imperfect but will keep all the data as close to original: lossless compression, high-quality sound.  
FLAC: This is free of loss type of compression, it does not reduce on the quality of the audio.

**Video Encoding:**  
MP4: Regular type of a video file storage and sharing.  
AVI: Supports various codecs.  
MOV: Apple’s video format.  
WMV: Microsoft’s video format.

**Assignment-3**

**Explain Basic Structure of an Arduino Program.**

* There are two required parts or functions that enclose blocks of statements.
* **setup()** is the preparation, **loop()** is the execution.
* Both functions are required for the program to work.

void setup()

{

statements;

}

void loop()

{

statements;

}

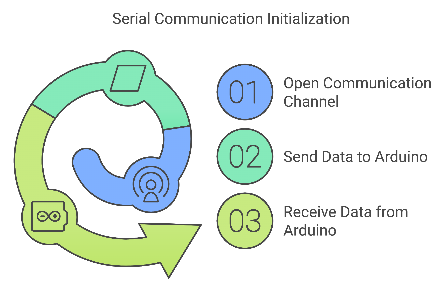
**The setup() function is called once**, when the Arduino board is first turned on or reset. It is used to initialize the board and set up the hardware.

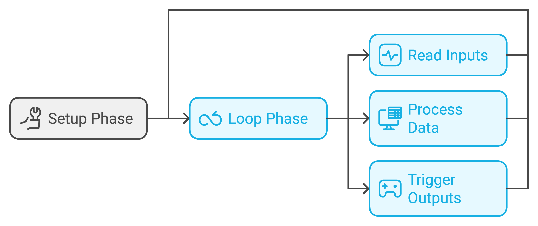
The setup function should follow the declaration of any variables at the very beginning of the program. It is **the first function to run in the program**, is run only once, and is used to set **pinMode** or initialize serial communication.

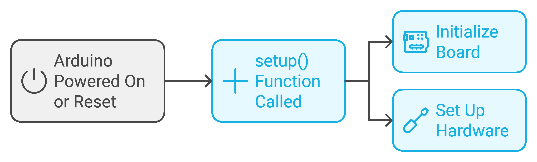
Setting pinMode: This tells the Arduino **whether a specific pin is going to be used for input** (reading data) or **output** (sending data).

For example, if you have an LED connected to pin 13, you would specify in the setup() function that pin 13 is an output pin.

**Initializing Serial Communication:** This is like **opening a communication channel** between your Arduino and your computer or another device. This is useful for **sending data back and forth**.







**The loop() function is called repeatedly,** until the Arduino board is turned off or reset. It is where the Arduino program does most of its work.

The loop function follows next and includes the code to be executed continuously – reading inputs, triggering outputs, etc.

**Assignment-4**

**The Architecture of Modern Computers:**

* How does the architecture of modern CPUs (e.g., x86-64 architecture) handle parallelism, and what are the implications for software design and performance?
* What are the key differences between RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing) architectures, and how do these differences influence the design of operating systems?
* Explain the differences between the CISC (Complex Instruction Set Computing) and RISC (Reduced Instruction Set Computing) architectures. Additionally, compare the John von Neumann architecture with the Harvard architecture, focusing on their memory organization and instruction processing. How do these differences impact the performance and design of modern processors?

Ans:

Modern CPUs have evolved their architecture to more efficiently handle complex tasks, with less reliance on having humans constitute, via paralleler, instructions to set design and supporting memory organization. Here’s a breakdown of these aspects and their implications:

1. **Parallelism in Modern CPU Architectures (e.g., x86-64 Architecture)**

* **Instruction-Level Parallelism (ILP)**: Multiple instructions can be processed simultaneously on a single CPU core via pipelining, superscalar execution, and out of order execution.
* **Data-Level Parallelism (DLP)**: On x86-64 SIMD (Single Instruction, Multiple Data) extensions such as AVX, allow the CPU to perform the same operation for each of multiple data points, speeding up jobs like multimedia processing.
* **Thread-Level Parallelism (TLP)**: Because it handles multiple threads at once, multithreading and multicore design of a CPU increases throughput and application responsiveness.

**Implications for Software Design:** Concurrency Models, Performance Tuning, Scalability

**2. Differences Between RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing)**

**CISC (e.g., x86):**

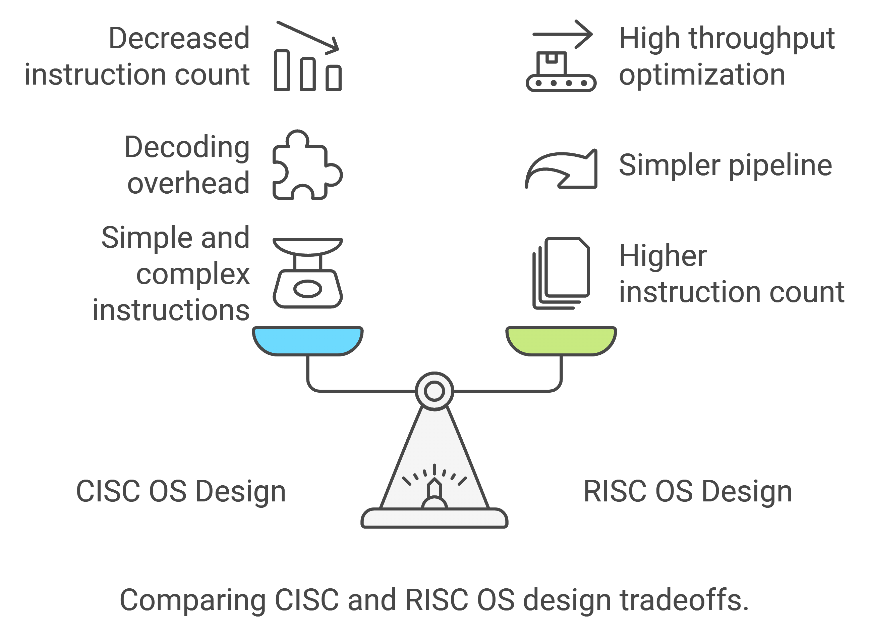
* **Complex Instruction Set**: Includes a large number of instructions, some of which can perform multi-step operations, reducing the number of instructions per program.
* **Variable-Length Instructions**: Instructions may vary in length, requiring more complex decoding but potentially fewer instructions per task.
* **Memory Access**: Supports memory-to-register operations directly within instructions.

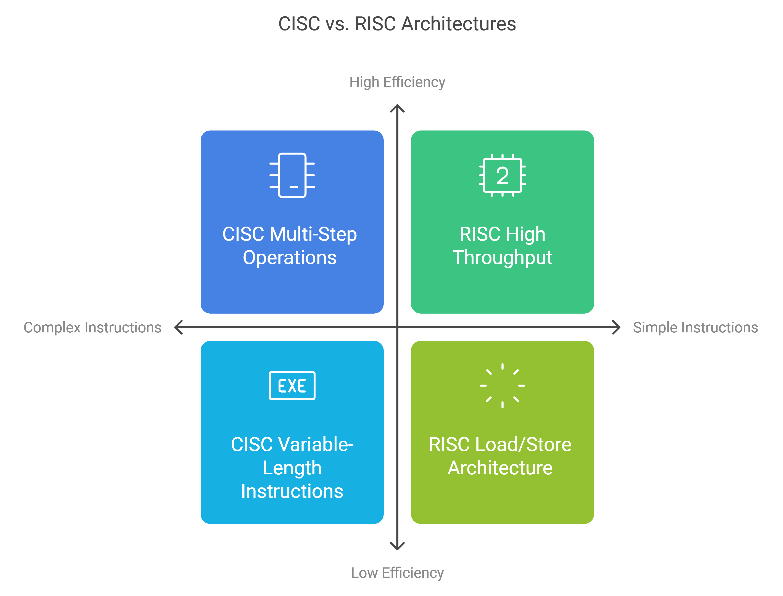
**RISC (e.g., ARM):**

* **Reduced Instruction Set:** Typically, it emphasizes a smaller set of simpler instructions that each do one operation, resulting in a higher instruction count.
* **Fixed-Length Instructions:** Simplest decoding and pipelining, increases instruction throughput.
* **Load/Store Architecture:** It enables memory access only via explicit load and store instructions, making the design simpler and hence more power efficient.

**Impact on Operating Systems**:

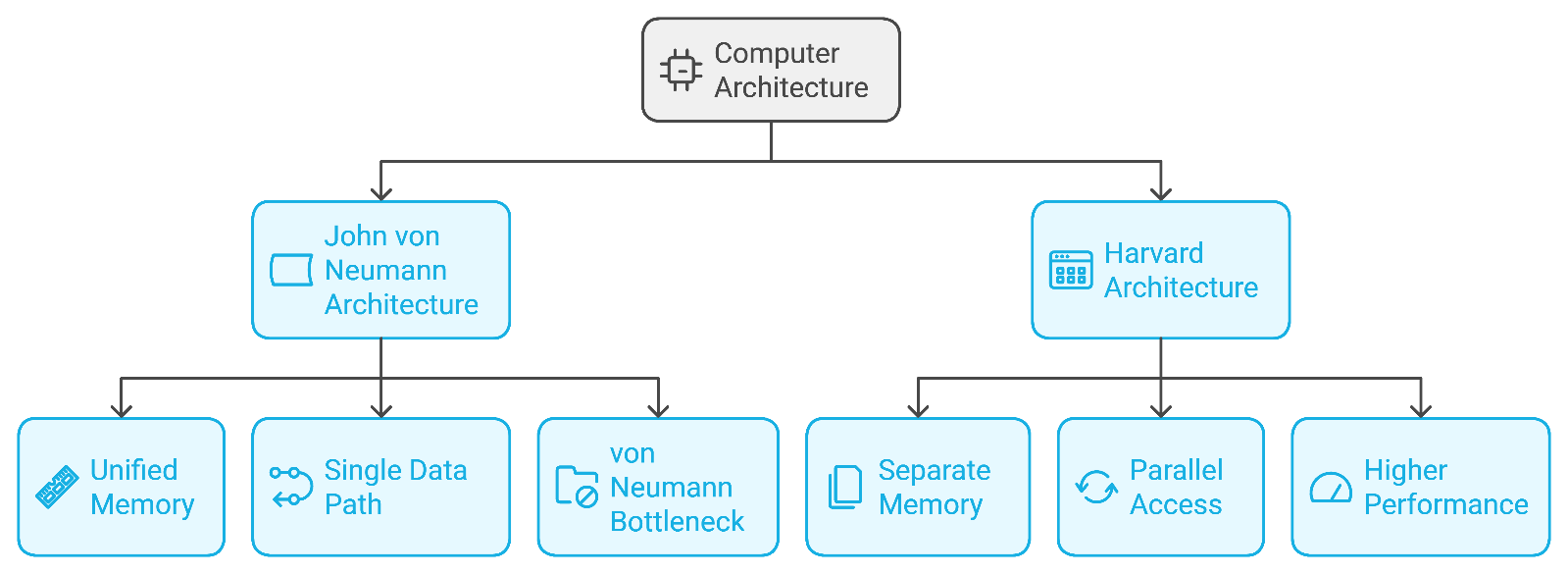
* **CISC OS Design:** Simple instructions and complex instructions can act to decrease total instruction count for OSs running on CISC architectures; however, the overhead for more cumbersome decoding stages must also be manually managed.
* **RISC OS Design:** RISC architectures force OSs to use a higher instruction count, but in return for this, there is a simpler, faster pipeline. Typically they have a focus on modularity and high throughput optimization.

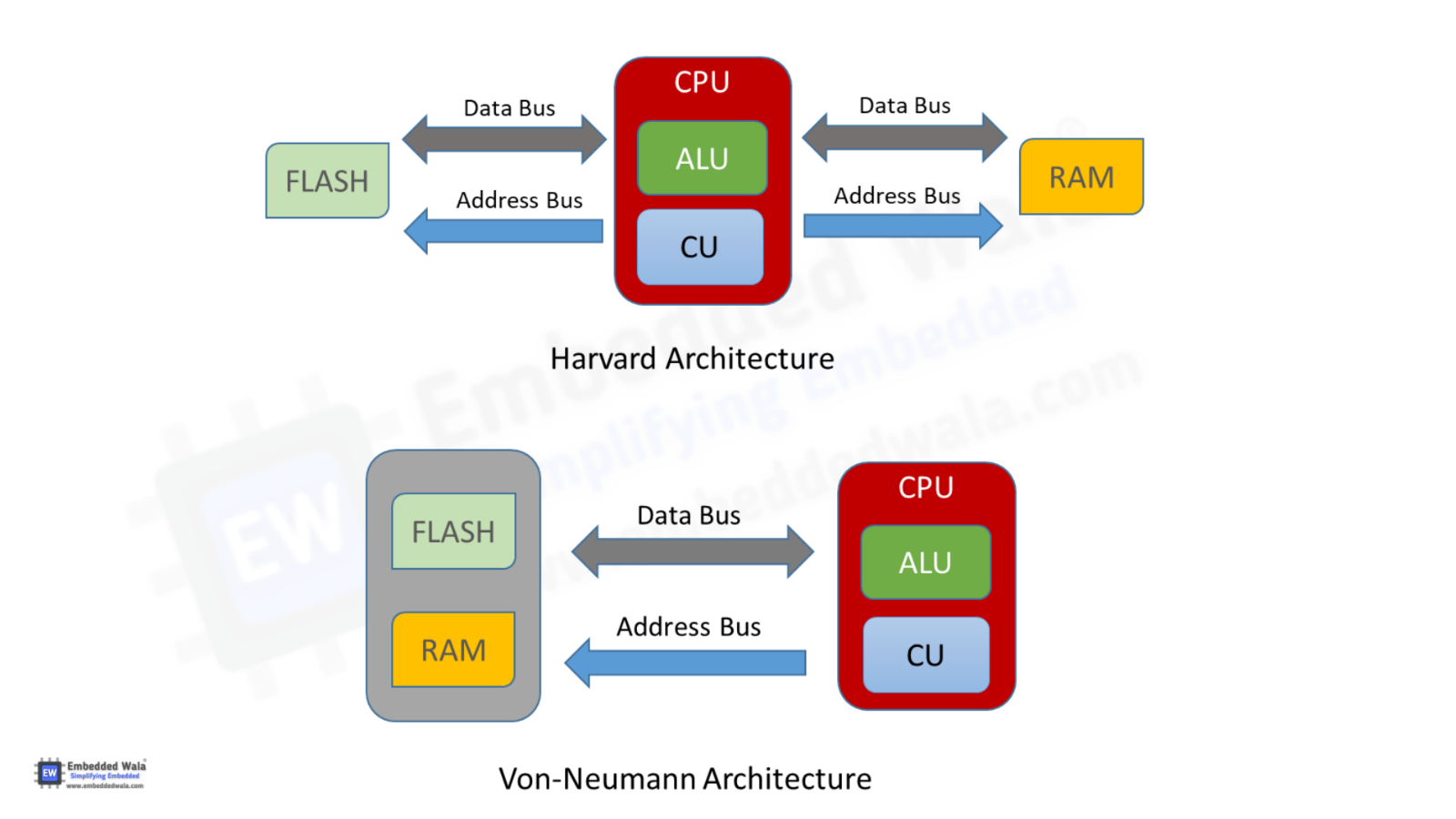


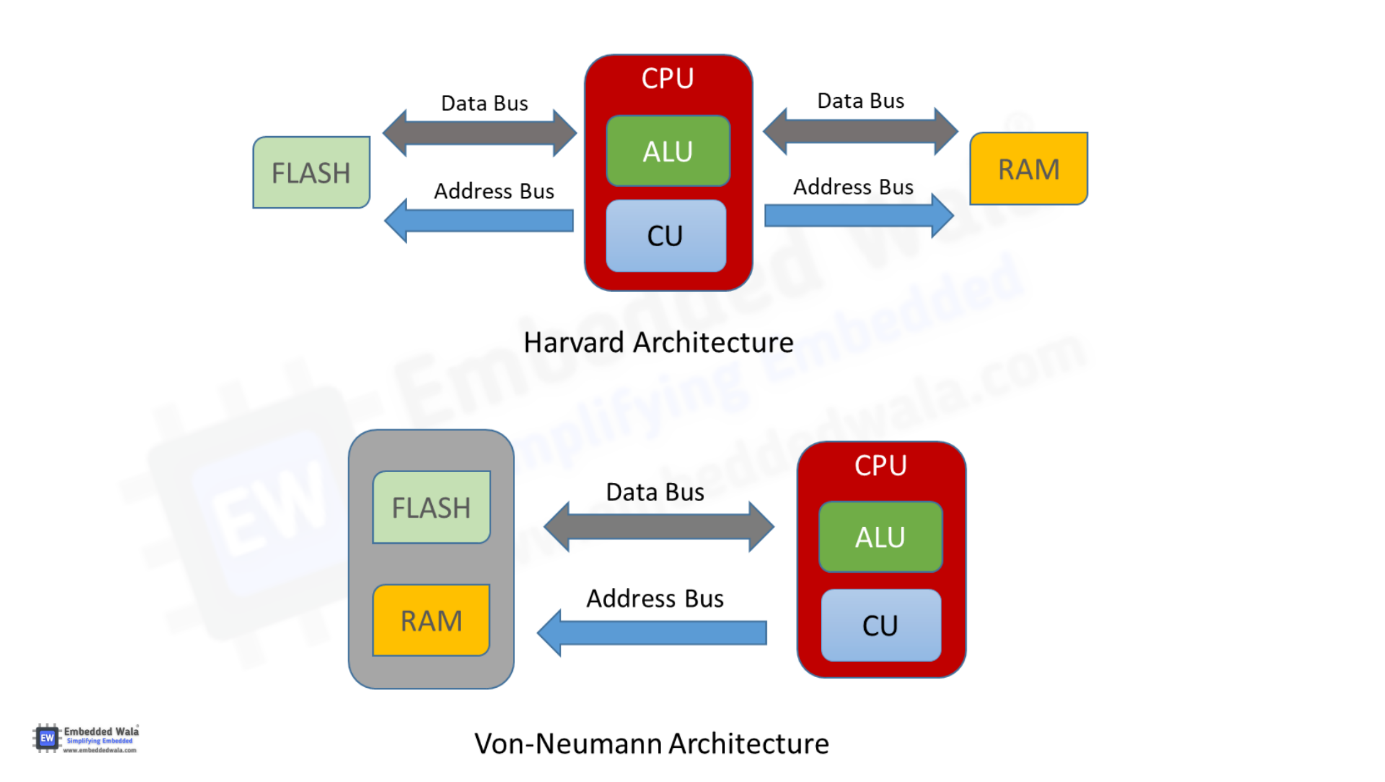


**Impact on Processor Performance and Design**:

* **Von Neumann Architecture** is simpler and less costly to implement, which makes it pervasive in the design space of general purpose CPUs; however, this architecture is less attractive in the high performance space because of bus contention.
* **The Harvard Architecture** is more suitable for embedded systems or real time processing where the benefit of keeping separate data and instruction streams outweighs the performance achieved by Harvard Architecture. There are, however, more technical and expensive ways of implementing them.







**Microcontrollers (e.g., Arduino Uno R3):**

* How does the AVR architecture used in the Arduino Uno R3 differ from the architecture used in more advanced microcontrollers like ARM Cortex-M?
* What is the significance of using an 8-bit microcontroller (like the ATmega328P in Arduino Uno) compared to 16-bit or 32-bit microcontrollers in terms of memory addressing, processing power, and application suitability?

Ans:

**AVR vs. ARM Architecture**

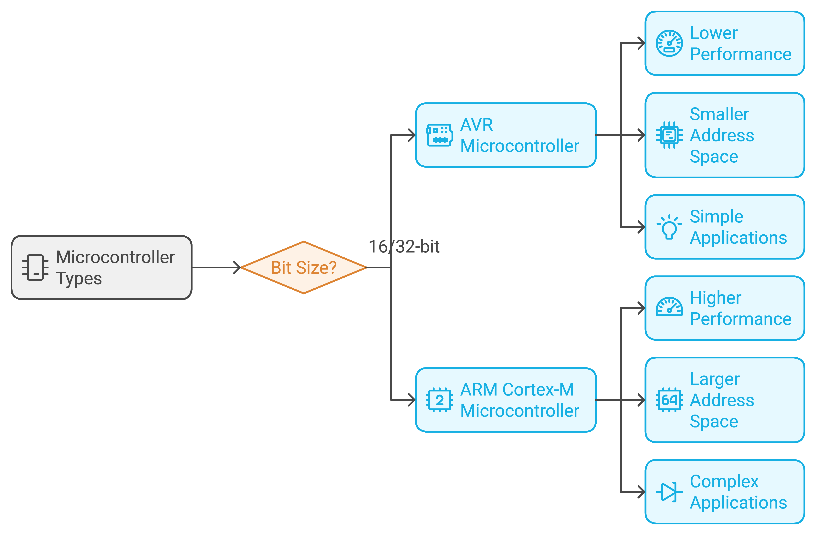
* + AVR (Atmel RISC Architecture): Used in the Arduino Uno R3. The design is a simple 8 bit RISC architecture with a low power consumption and a simple instruction set.
  + ARM Cortex-M: A family of 32 bit RISC architectures for more advanced microcontrollers. AVR is basically restricted, uses lower performance, smaller address space, and has fewer features than AVR.

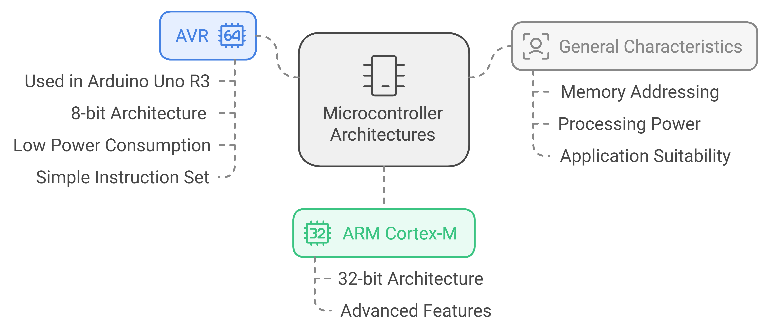
**Microcontrollers 8bit vs 16/32bit**

• **Memory Addressing**: Smaller 8bit microcontrollers have less address space and so can only access less memory.

• **Processing Power**: High processing power microcontrollers come with 16bit and 32 bit microcontrollers which makes them suitable for demanding applications.

• **Application Suitability**: 8 bit microcontrollers are appropriate for light usage and simplest functions that use small amount of memory and processing requirements while the 16 bit and 32 bit micro controllers are ideal for more complex applications.





**Memory Segmentation:**

* How does memory segmentation in x86 architecture support backward compatibility, and what are the advantages and disadvantages compared to flat memory models used in modern 64-bit systems?
* What is the difference between User Space and Kernel Space in the virtual memory layout of modern computers, and why is this separation important? Additionally, how does memory segmentation work, and what role does it play in managing memory in older vs. modern computing systems?

Ans:

**Memory Segmentation and Backward Compatibility**

Memory segmentation in x86 architecture divides the address space into segments, which can be used to isolate different processes or parts of a program. This allows for backward compatibility with older 16-bit software, which used segmented memory. However, segmentation can be inefficient and complex to manage.

**User Space vs. Kernel Space**

* + **User Space:** The portion of memory to which user applications have access.
  + **Kernel Space:** The space available to the operating system kernel. Security and isolation requires this separation. This prevents user applications from being able to directly access or modify kernel code, thereby posing potential problems such as system instability or security breaches.

**Memory Segmentation on Older vs. Modern Systems**

* + **Older Systems:** A lot was done with memory segmenting to manage memory and protect processes.
  + **Modern Systems:** Some cases still use segmentation, although the flat memory models are more common. In 64 bit systems, flat memory models are helpful, more simpler and more efficient to manage memory.

**Memory Location and Byte Addressability**

**1-Byte Memory Locations**

Each memory location is generally 1 byte in modern computer systems because it is a convenient unit of data. Bytes can represent a wide range of values, from characters to small integers. This design choice simplifies memory addressing and data manipulation.

**Endianness**

Endianness refers to the order in which bytes are stored in memory.

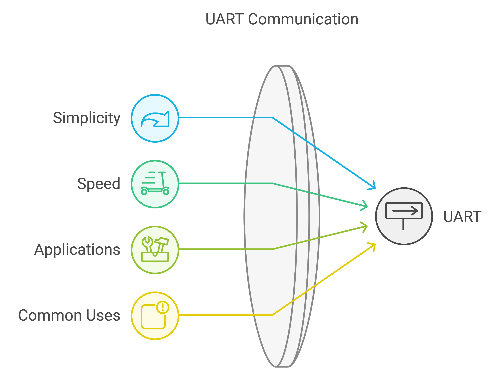
• Big-Endian: The most significant byte is stored first.

• Little-Endian: The least significant byte is stored first.

Developers must be aware of the endianness of the system they are working with to ensure correct data interpretation. Misunderstanding endianness can lead to data corruption and errors.

**Assignment-5**

How do UART, I²C, SPI, CAN, and USB communication protocols differ in terms of data transmission speed, complexity, pin usage, and device-to-device communication? What are the key features that make each protocol suitable for specific applications, and in what types of embedded systems would each be most commonly used ?

**1. UART (Universal Asynchronous Receiver-Transmitter)**

- It as a simple two-way street (just 2 wires: TX and RX)

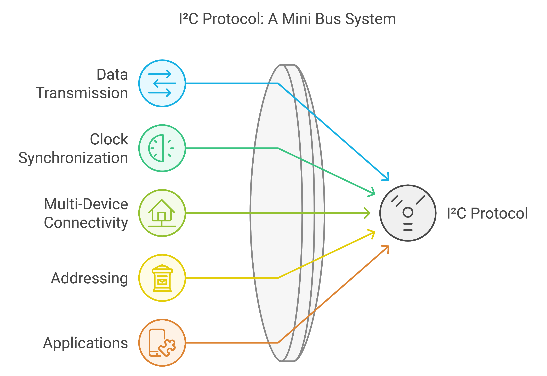
- It is a slow but super easy to use (typically 9600 to 115200 baud rate)

- Great for basics like connecting your Arduino to your computer

- Perfect when you just need to send data between two devices

- Common uses: Debug messages, simple sensor readings, basic communication

**2. I²C (Inter-Integrated Circuit)**

****- Like a mini bus system with just 2 wires (SDA for data, SCL for clock)

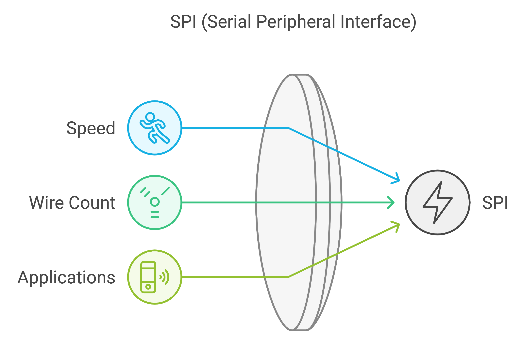
- Medium speed (100kHz to 400kHz typically)

- Cool feature: Can connect multiple devices (up to 127!) using just those 2 wires

- Each device has an address, like tiny houses on a street

- Best for: Reading multiple sensors, connecting several chips on one board

- Common in smartphones, displays, and sensor modules

**3. SPI (Serial Peripheral Interface)**

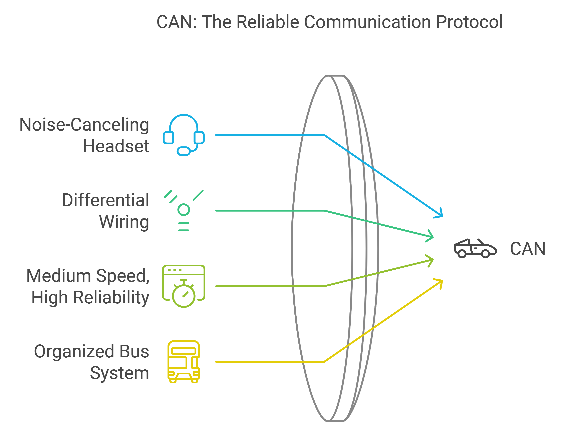
- It is super fast and speed for simple protocols (can go several MHz)

- Uses more wires (MOSI, MISO, CLK, SS) but gives you better speed

- Great for: SD cards, displays, fast data transfer between chips

- Common in memory chips, LCD screens, and SD card interfaces

**4. CAN (Controller Area Network)**

****- In a regular phone call (like other protocols), background noise can mess up your conversation

But CAN is like having a special noise-canceling headset!

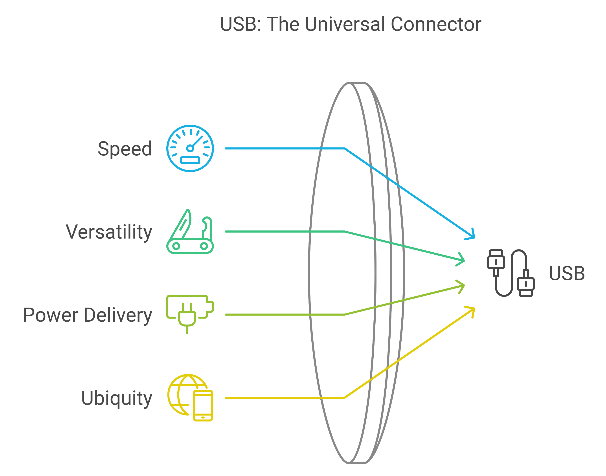
- Uses 2 wires but they're differential (CAN-H and CAN-L)

- Medium speed but incredibly reliable

- Like a really well-organized bus system where everyone follows strict rules

- Perfect for: Cars, industrial equipment, noisy environments

- You'll find this in every modern car's electronics

**5. USB (Universal Serial Bus)**

- It is king of protocols

- Very fast (from 12 Mbps to several Gbps)

- More complex but super versatile

- Can provide power too (that's why your phone charges through it)

- Used in: Pretty much everything - phones, computers, game controllers

**Quick comparison for choosing what to use:**

- Need something simple and direct? → UART

- Lots of sensors on one board? → I²C

- Need speed with a few devices? → SPI

- Working with cars or industrial stuff? → CAN

- Need something user-friendly and standardized? → USB