Ch5: Microprocessors and Microcontrollers

Contents:

Microprocessors and Microcomputers

Microcontrollers

AVR Microcontrollers

Microprocessors and Microcomputers

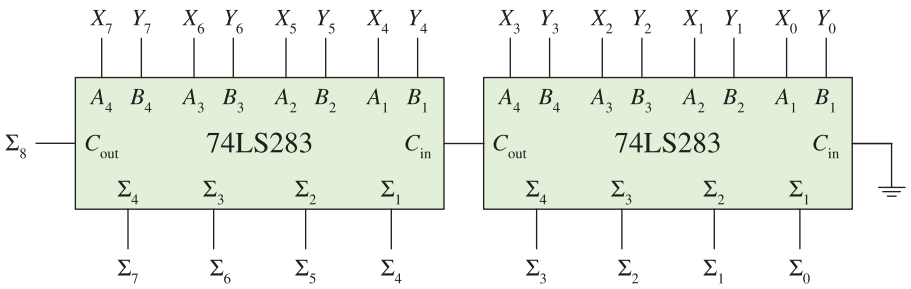
Hardware Solutions vs. Software Solutions

When using only digital ICs for a task (**hardware solution**), to make a change in functionality, the hardware circuitry must be modified and may require a **redesign**. However, in complex tasks with many inputs and outputs, a strictly hardware solution is impractical.

Thus, use of a **microprocessor-based system** to implement a **software solution** is more appropriate. Software is a procedural program consisting of a set of instructions to execute logic and arithmetic functions and to access input signals and control output signals. **Advantage**: Without making changes in hardware, the program can be easily modified to alter the system's functionality.

8-bit Adder
Using 2 ICs

$$\begin{array}{r} X_7\,X_6\,X_5\,X_4\,X_3\,X_2\,X_1\,X_0 \\ +\ Y_7\,Y_6\,Y_5\,Y_4\,Y_3\,Y_2\,Y_1\,Y_0 \\ \hline \Sigma_8\,\Sigma_7\,\Sigma_6\,\Sigma_5\,\Sigma_4\,\Sigma_3\,\Sigma_2\,\Sigma_1\,\Sigma_0 \end{array}$$

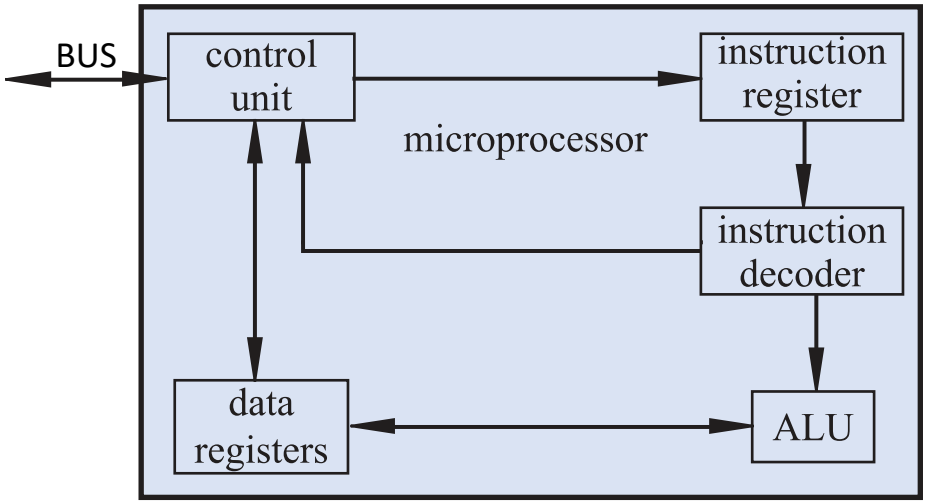


Microcontroller

VS.

Microprocessor

Microprocessor Unit (**MPU**) or **Central Processing Unit** (**CPU**) is a single, very-large-scale-integration (**VLSI**) chip that contains many digital circuits that perform arithmetic, logic, communication, and control functions.

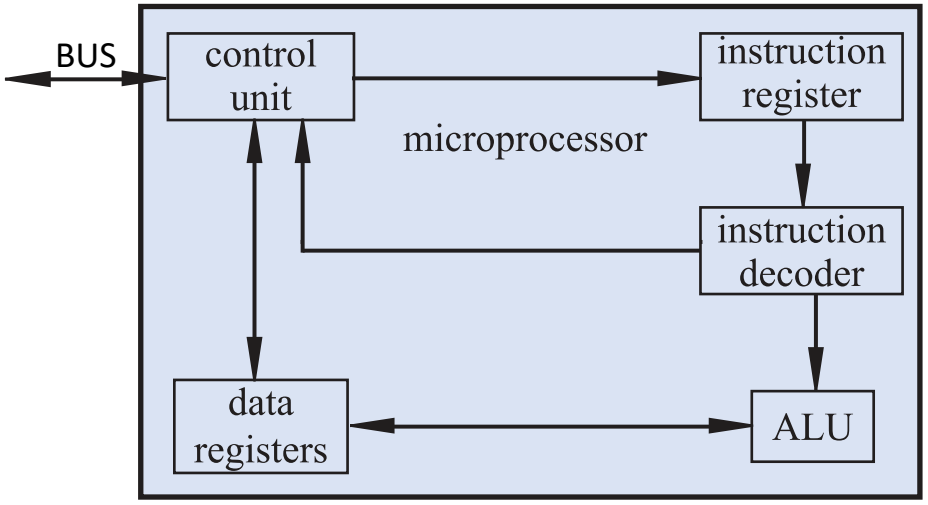


Arithmetic Logic Unit (**ALU**) executes mathematical functions on data structured as binary words (an ordered set of bits, usually 8, 16, 32, or 64 bits long).

Microprocessor

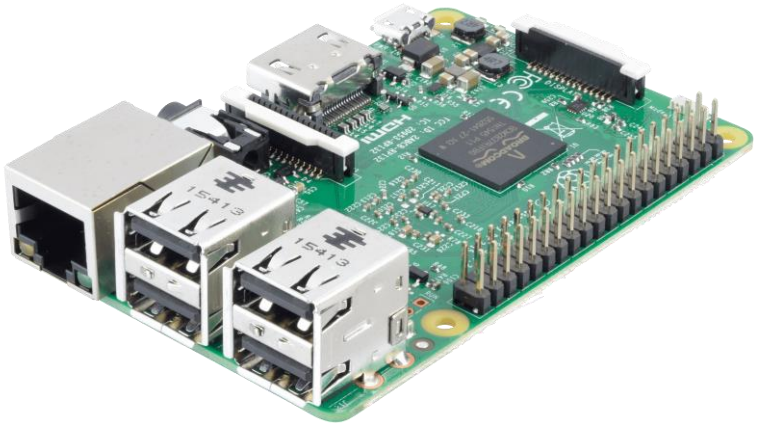
Instructions and **data** are **fetched sequentially** from **memory** by the **control unit** through **bus** and **stored** in the **instruction register**. These instructions are **interpreted** by the **instruction decoder**. Each instruction is a set of coded bits that commands the **ALU** to perform bit manipulation (e.g., binary addition and logic functions) on words stored in the **data registers**. The ALU results are also stored in data registers and then transferred to memory by the control unit.

- **Instructions** are defined by a binary code called **Machine Code**.
- The instructions are microprocessor dependent.
- Each instruction is represented by a **unique binary string** that causes the microprocessor to perform a low-level function (e.g., add a number to a register or move a register's value to a memory location).

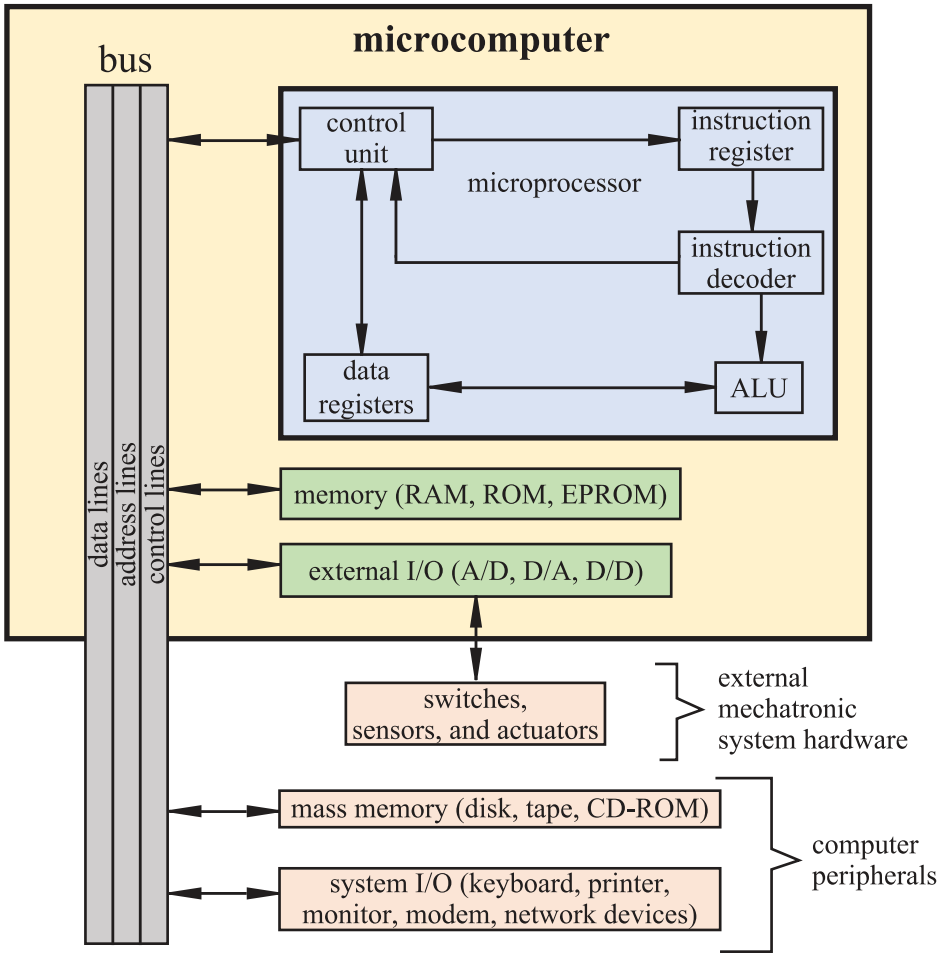


Microcomputer

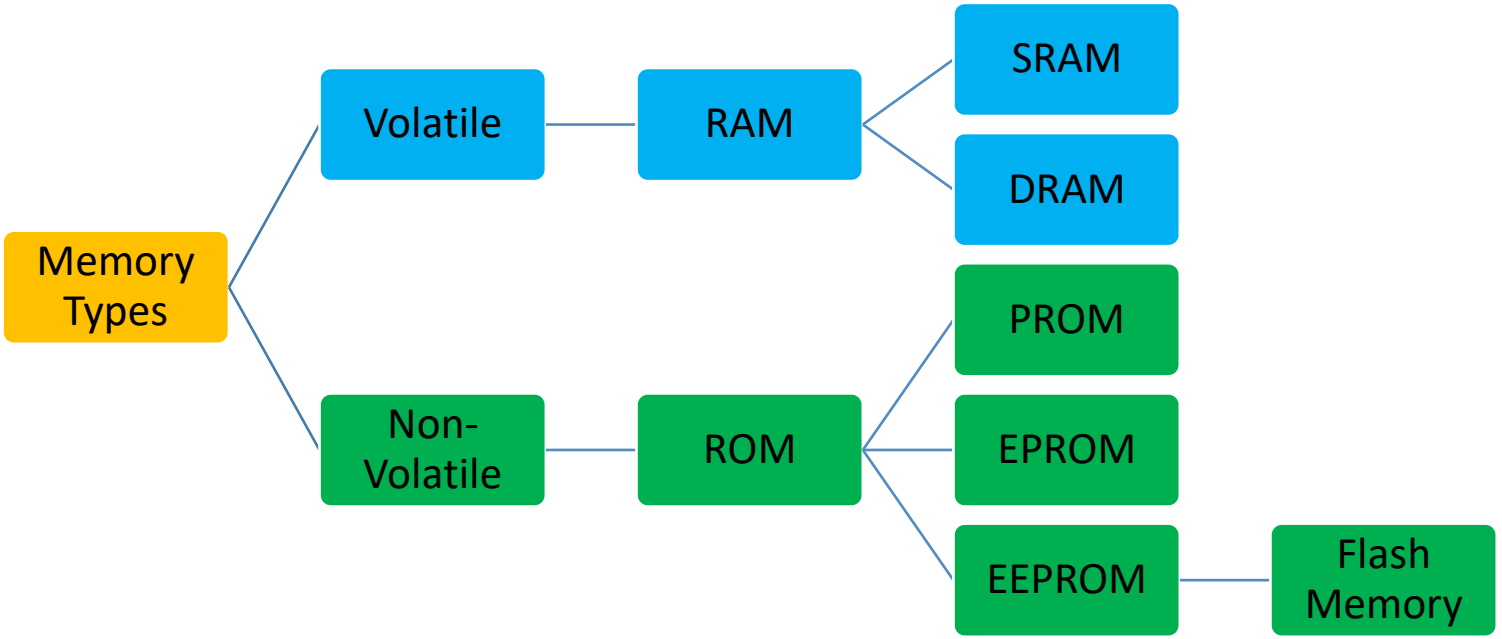
When a microprocessor is packaged on a printed circuit board (PCB) with other components (e.g., interface and memory chips) the resulting assembly is referred to as a **Microcomputer** or **single-board computer**.



Raspberry Pi 3 B



Memory Types

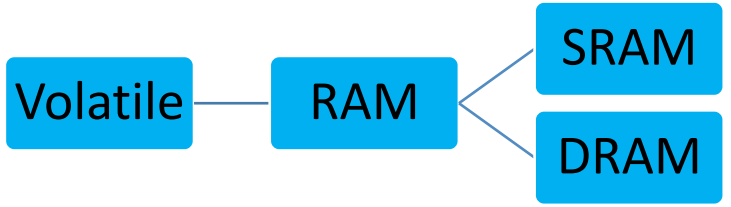


Volatile Memory, in contrast to **Non-Volatile Memory** (**NVM**), is computer memory that **requires power** to maintain the stored information; it retains its contents while powered on but when the power is interrupted, the stored data is quickly lost.

Volatile Memory

Random-Access Memory (**RAM**) allows data items to be **read** or **written** in almost the same amount of time irrespective of the physical location of data inside the memory, as long as power is maintained.

Static RAM (**SRAM**) uses flip-flops to store each bit.



Dynamic RAM (**DRAM**) stores each bit of data in a separate tiny capacitor within an IC. The capacitor can either be charged or discharged (≡ 0 or 1). To prevent charge leakage, DRAM requires an external memory refresh circuit which periodically refreshes (rewrites) the data in the capacitors, restoring them to their original charge.

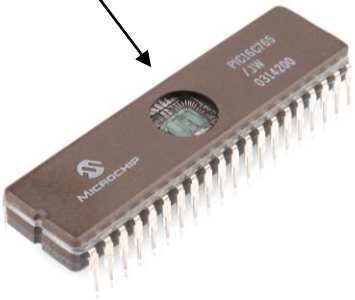
SRAM is faster and more expensive than DRAM; it is typically used for CPU cache while DRAM is used for a computer's main memory.

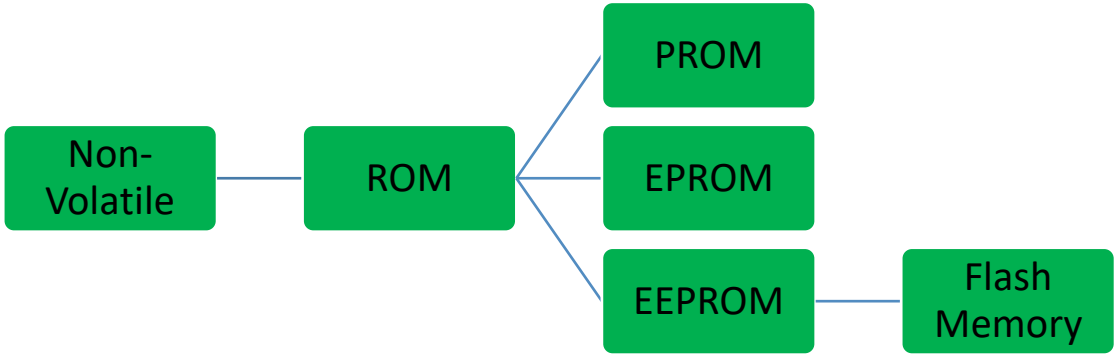
Non-Volatile Memory (NVM)

Read-Only Memory (**ROM**) is used for permanent storage of data that the CPU can read, but the CPU cannot write data to ROM. Data stored in ROM cannot be modified, so it is mainly used to store firmware (software that is closely tied to specific hardware, and unlikely to need frequent updates). The data is written into a ROM during manufacture.

Programmable ROM (**PROM**) is manufactured blank and the data is programmed into it only one-time.

Data stored in an **Erasable-Programmable ROM** (**EPROM**) can be erased with strong ultraviolet light (such as from a mercury-vapor light) applied through a transparent quartz window on top of the EPROM IC. Then, new data can be stored on the EPROM.



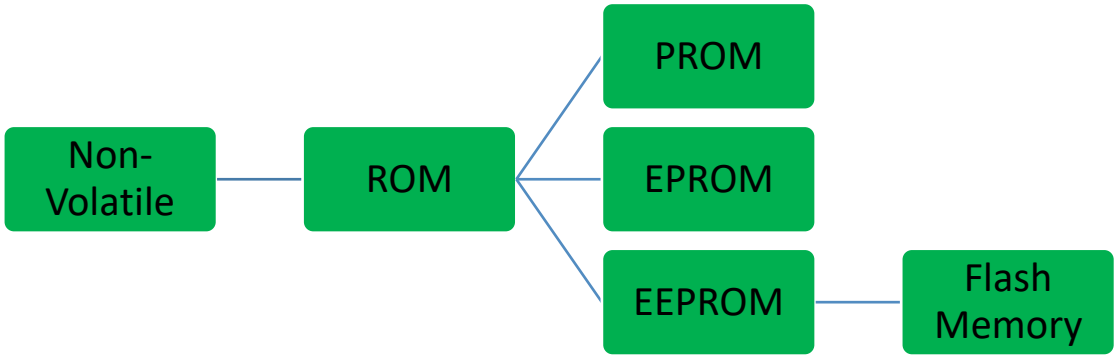


Non-Volatile Memory (NVM)

Data stored in an **Electrically Erasable-Programmable ROM** (**EEPROM**) can be erased electrically and rewritten through its data lines without the need for ultraviolet light. It has a limited life (typically 1,000,000 cycles) for erasing and reprogramming. Erase cycles are slow because of the small block sizes used in erasing.

Flash Memory is a type of EEPROM designed for high speed and high density, at the expense of large erase blocks (typically 512 bytes or larger).

Many microcontrollers include both: flash memory and a small EEPROM for parameters and history.



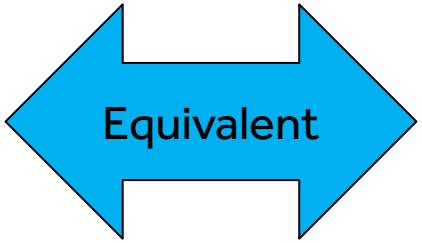
Microprocessors Programming Language

Microprocessors can be programmed using **low-level** programming languages like **Assembly** language, **high-level** programming languages like **BASIC**, or **middle-level** programming languages like **C/C++**. These programming languages consists of a series of processor **instructions**, comments, and data.

Assembly language (asm) has a very strong correspondence between the **program's statements** and the architecture's **machine code instructions** (e.g., ADD to add a number to a register and MOV to move a register's value to a memory location).

(C)

```
int total = 0;
for (int i=10; i!=0; i--)
total += i;
```

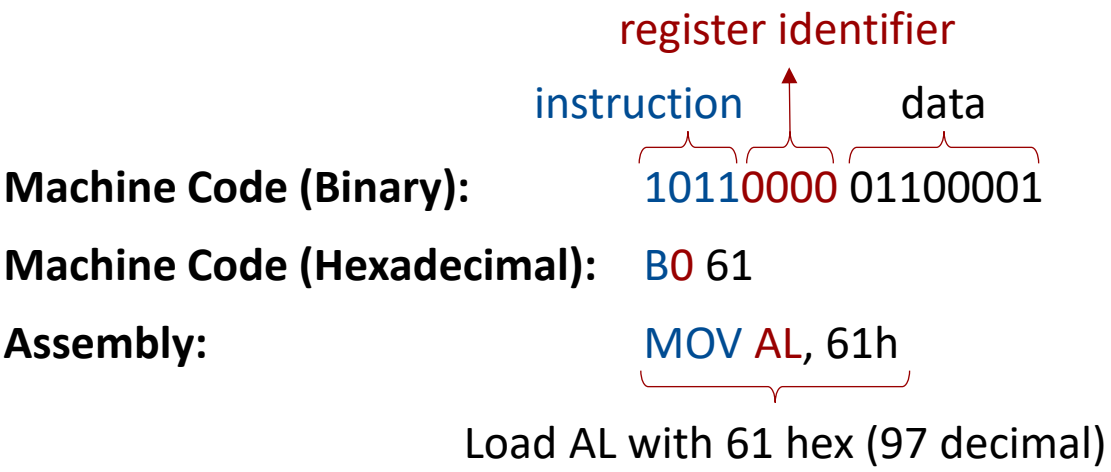


(Assembly)

```
0       MOV RO, #0;    // total = 0
1       MOV R1, #10;   // i = 10
2       MOV R2, #1;    // constant 1
3       MOV R3, #0;    // constant 0
Loop:   JZ R1, Next;   // Done if i=0
5       ADD RO, R1;    // total += i
6       SUB R1, R2;    // i--
7       JZ R3, Loop;   // Jump always
```

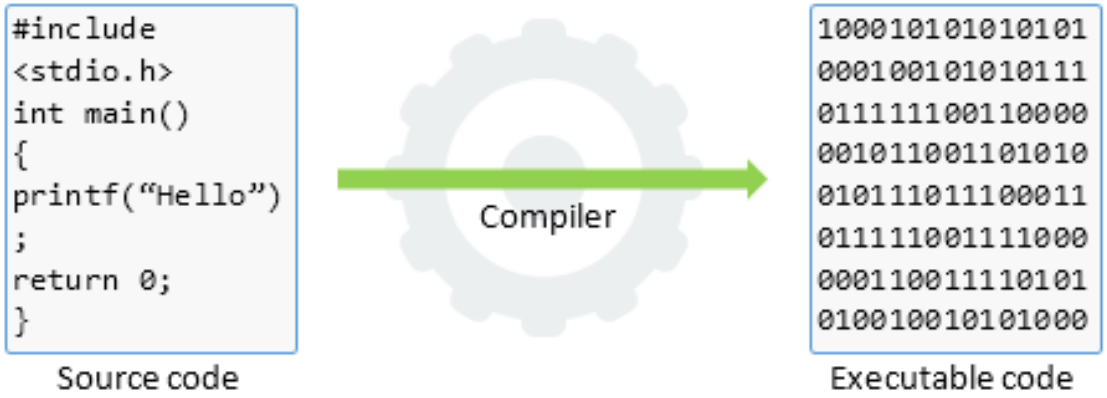
Microprocessors Programming Language

e.g., the instruction below tells a processor to move an 8-bit value into an AL register:



Microprocessors Programming Language

Programs must be first converted to machine code (binary code) for the specific microprocessor using software called **Compiler** (for assembly it is also called **assembler**).



The **advantages** of using a high-level language are that it is easier to **learn** and use, programs are easier to **debug** (the process of finding and removing errors), and programs are easier to comprehend. A **disadvantage** is that the resulting machine code may be less efficient (i.e., slower and require more memory) than a corresponding well-written low-level language program.

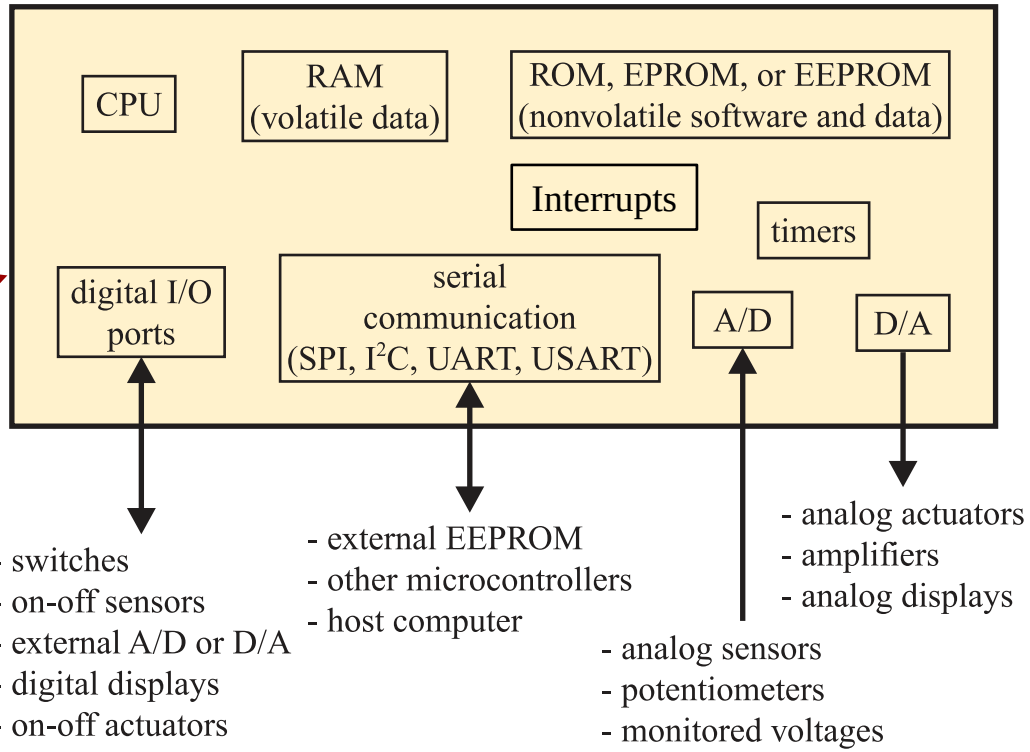
Microcontrollers

Microcontroller

Microcontroller (μC or MCU) is a single IC which contains a microprocessor (CPU), RAM, ROM, digital I/O ports, serial communication interface, timers, analog-to-digital (A/D) converters, and digital-to-analog (D/A) converters. Because of low cost, versatility, ease of programming, and small size, microcontrollers are used in a wide array of applications.

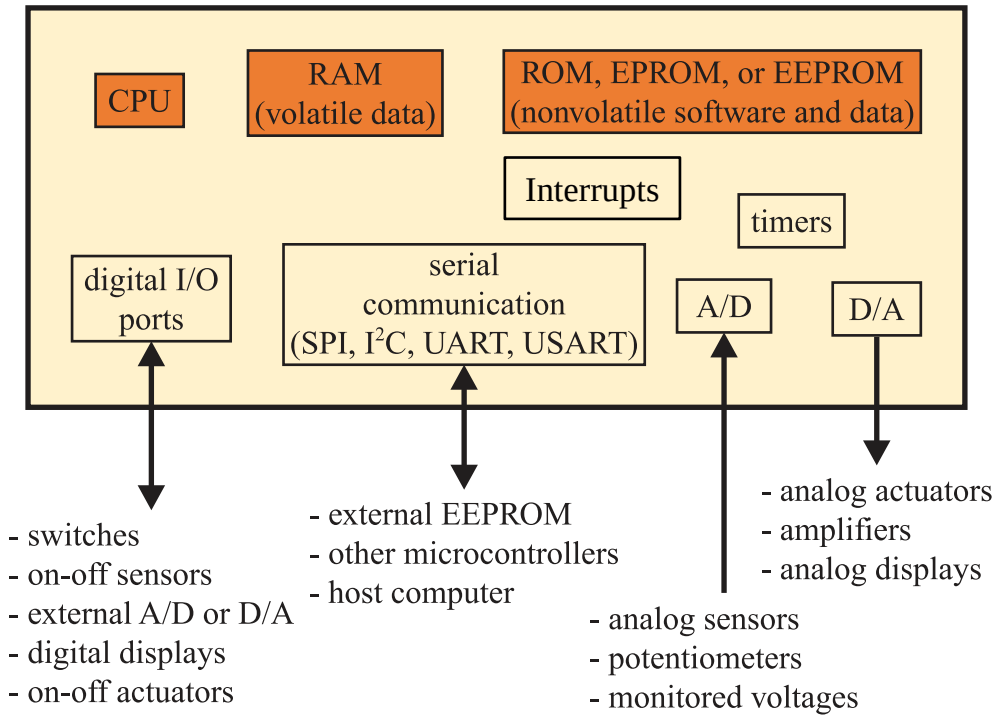
Manufacturers continually release products with faster speeds, larger memories, and more functionality. Here is a list of common MCUs.

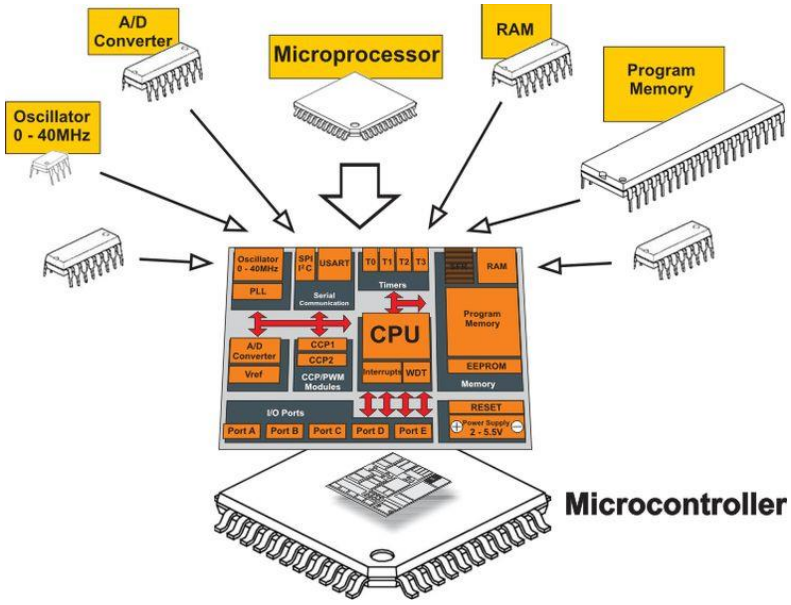
(A block diagram for a typical full-featured microcontroller)



Microcontroller

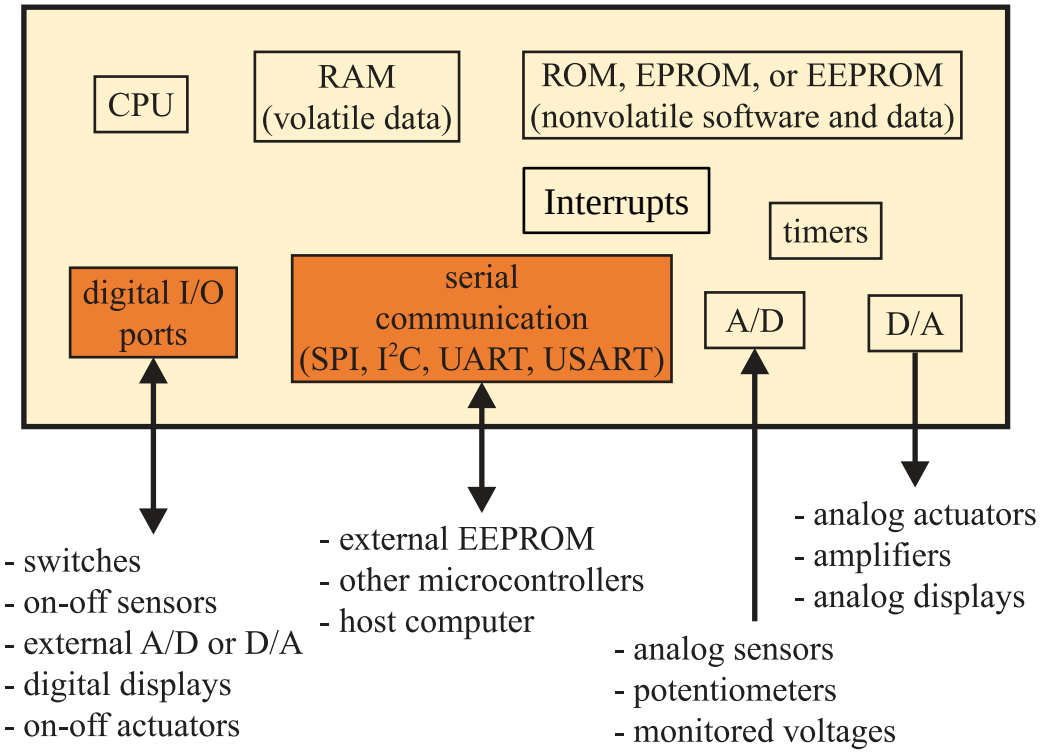
CPU retrieves and executes the program instructions stored in **ROM** and controls all the microcontroller components. The **ROM/EPROM/EEPROM** is used to store the program and any permanent data like settings. The **RAM** is used to store temporary settings and values needed during program execution.





Microcontroller

The **digital I/O ports** allow binary data to be transferred to and from the microcontroller using **external pins** on the IC.



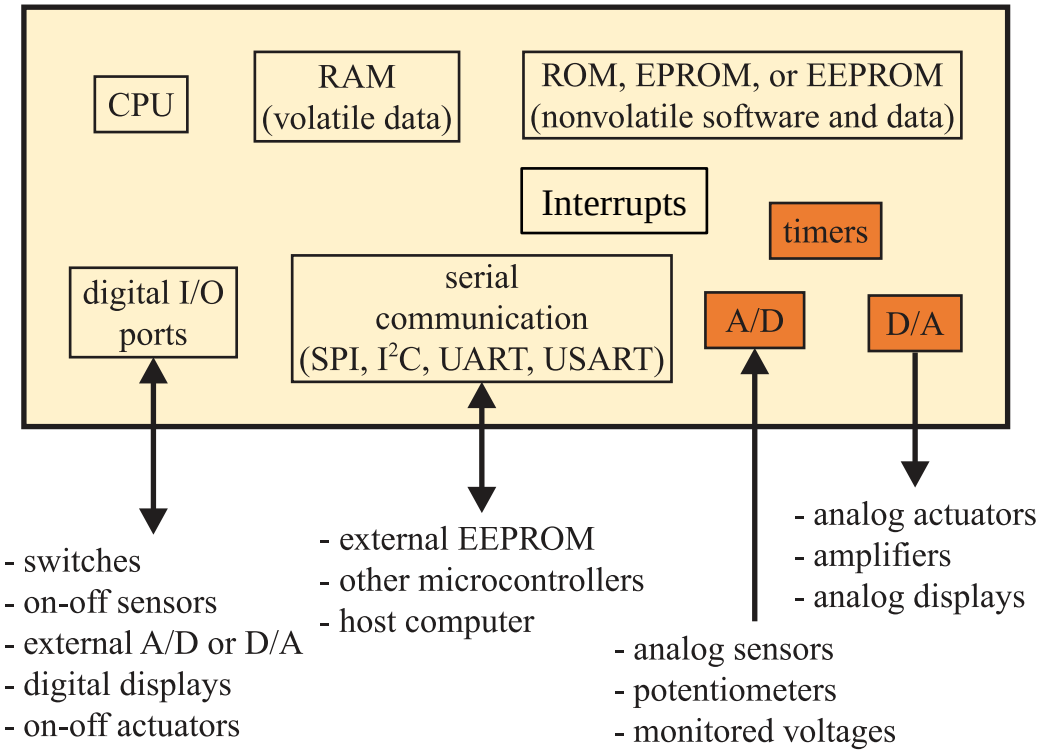
Serial communication can be used to transmit data to and from external devices which support the same protocol.

Various protocols for serial communication:
SPI (Serial Peripheral Interface), **I²C** (Inter-Integrated Circuit), **UART** (Universal Asynchronous Receiver-Transmitter), **USART** (Universal Synchronous-Asynchronous Receiver-Transmitter), and **USB** (Universal Serial Bus).

Microcontroller

The **A/D converter** converts an external analog voltage to a digital value that can be processed or stored by the CPU.

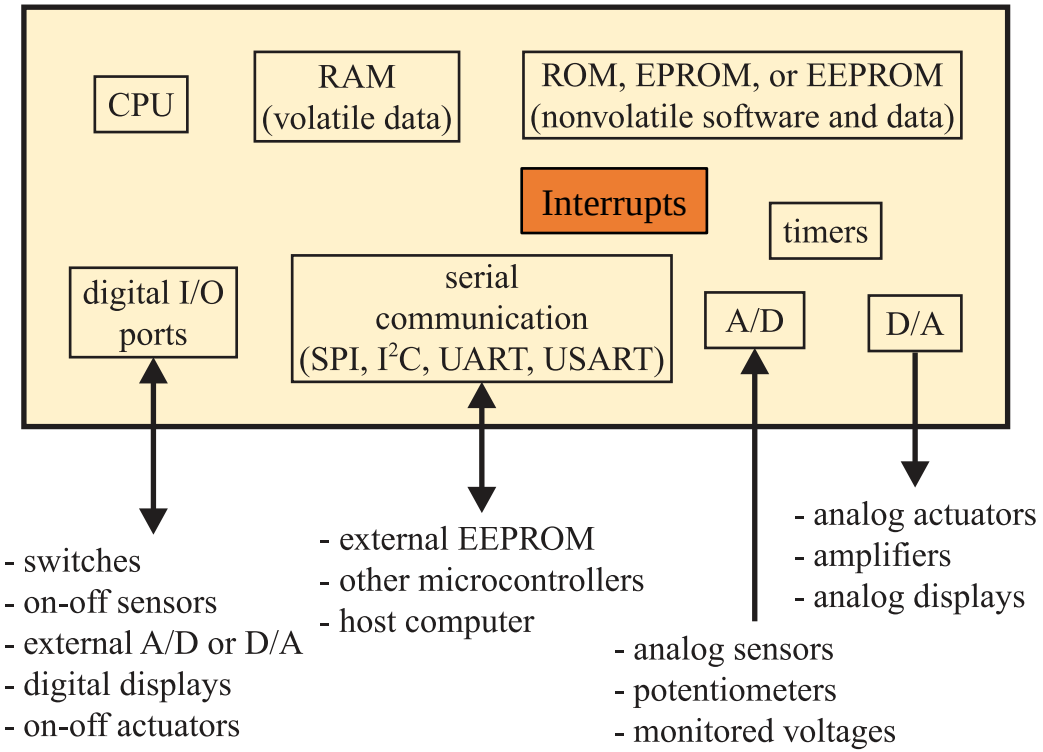
The **D/A converter** outputs an analog voltage to a nondigital device.



Onboard **Timers/Counters** are usually provided to help create delays or ensure events occur at precise time intervals (e.g., reading the value of a sensor).

Microcontroller

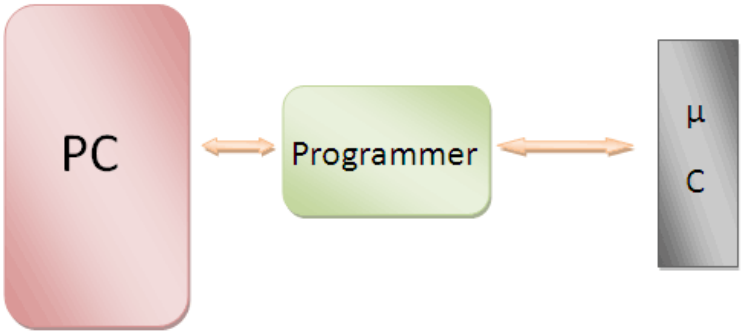
An **interrupt** system is used to interrupt a running program in order to process a special routine called the **interrupt service routine** (ISR).

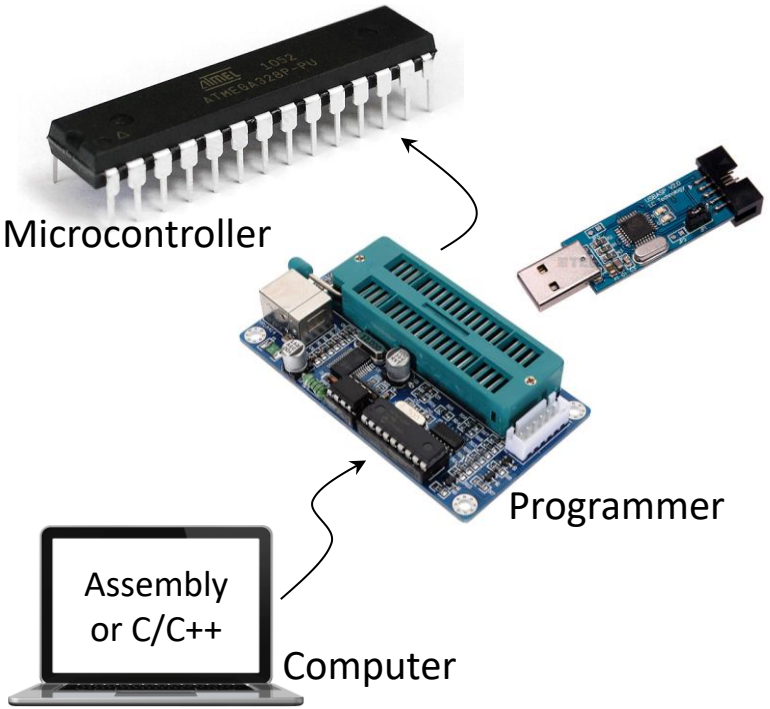


This is the ability of the MCU to respond to external data that requires immediate attention, such as data conveyed by an external sensor indicating important shutdown information.

Microcontroller Programmer

Microcontroller Programmer is a hardware device accompanied with software which is used to transfer the compiled machine code (usually a HEX file) from a PC directly to the EEPROM of the microcontroller. The code is transferred using serial, parallel, or USB port.





AVR Microcontrollers

AVR Microcontrollers

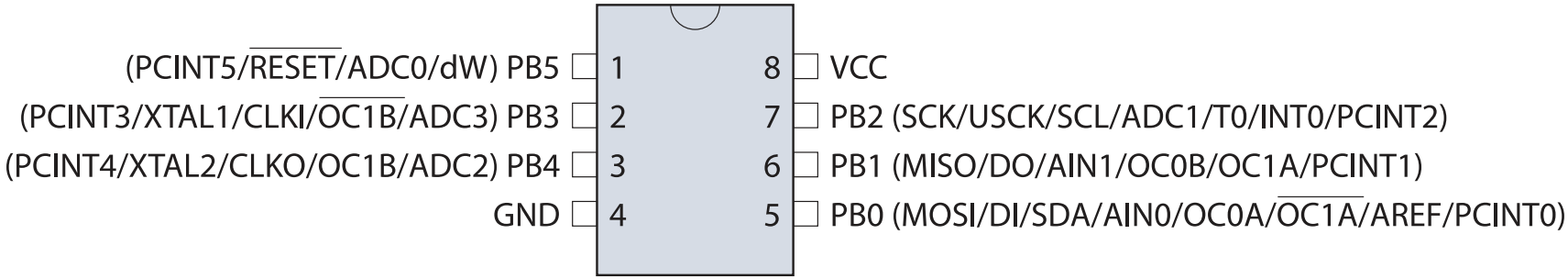
AVR is a family of microcontrollers developed since 1996 by **Atmel**, acquired by **Microchip Technology** in 2016. These are **modified Harvard architecture 8-bit RISC** single-chip microcontrollers. AVR was one of the first microcontroller families to use on-chip **flash memory** for program storage.

AVRs are generally classified into following:

- **tinyAVR (ATtiny series)**
  - 0.5–32 KB program memory
  - 6–32-pin package
  - Limited peripheral set
- **megaAVR (ATmega series)**
  - 4–256 KB program memory
  - 28–100-pin package
  - Extended instruction set
  - Extensive peripheral set
- **XMEGA (ATxmega series)**
  - 16–384 KB program memory
  - 44–64–100-pin package (A4, A3, A1)
  - 32-pin package: XMEGA-E (XMEGA8E5)
  - Extended performance features
  - Extensive peripheral set with ADCs
- **32-bit AVRs**

ATtiny85 Microcontroller

ATtiny85 Microcontroller is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. It is an 8-pin IC available in both surface-mount and through-hole DIL packages.



VCC: Supply voltage (0-10 MHz @ 2.7-5.5V, 0-20 MHz @ 4.5-5.5V)
GND: Ground
RESET: Reset input (a low level on this pin will generate a reset)
Port B (PB5:PB0): It is a 6-bit bi-directional I/O port with internal pull-up resistors.
- Port B also serves the functions of various special features.

(Refer to datasheet for more information)

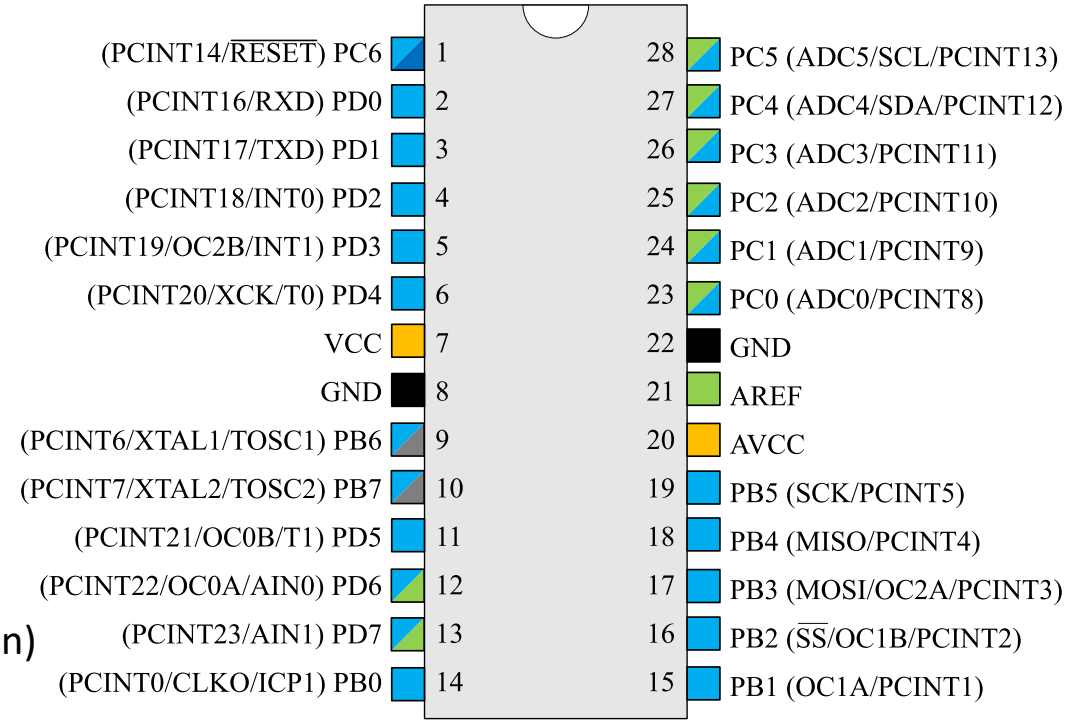
ATmega328P Microcontroller

ATmega328 Microcontroller is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. It is an IC available in different packages; 28-pin PDIP, 28-pin QFN, 32-pin QFN, 32-pin TQFP.

- Plastic Dual Inline Package (PDIP)
- Quad-Flat No-Leads (QFN)
- Thin Quad Flat Package (TQFP)

VCC: Supply voltage (1.8 - 5.5V).
AVCC: Supply voltage pin for the A/D Converter (ADC).
GND: Ground.
AREF: Analog reference pin for the A/D Converter.
Port B/C/D: Bi-directional I/O ports (with both sink and source capability) with internal pull-up resistors. They also serves the functions of various special features.

(Refer to datasheet for more information)



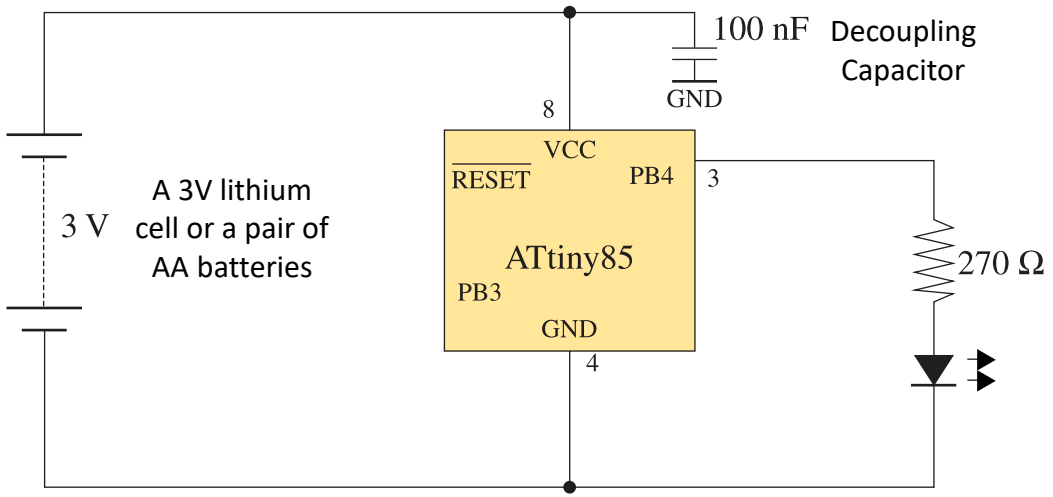
Example: Blinking LED using ATtiny85

```
// Define the pin where the LED is connected
#define LED_PIN 4  // PB4 on the ATtiny85

void setup() {
  // Set the LED_PIN as an output
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // Turn the LED on
  digitalWrite(LED_PIN, HIGH);
  delay(1000); // Delay for 1 second

  // Turn the LED off
  digitalWrite(LED_PIN, LOW);
  delay(1000); // Delay for 1 second
}
```

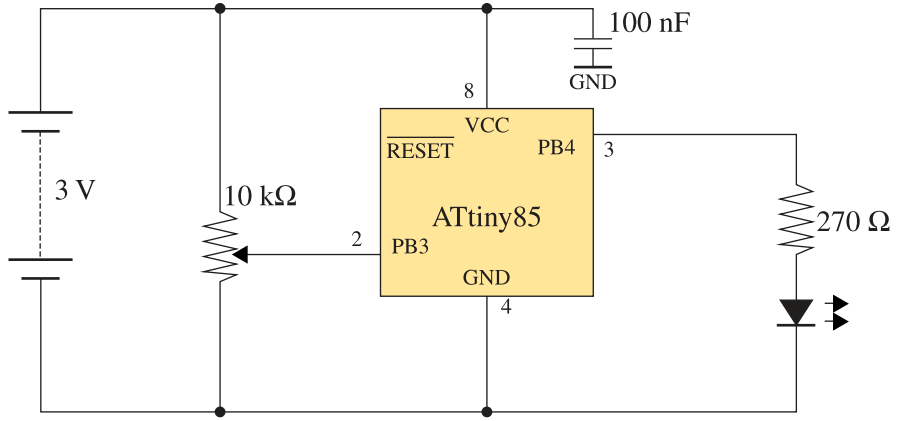


- This simple circuit can also be made with a **555 timer**. Although the microcontroller is more expensive and it also needs programming, it has more flexibility.

Example: Adjustable LED Blinker using ATtiny85

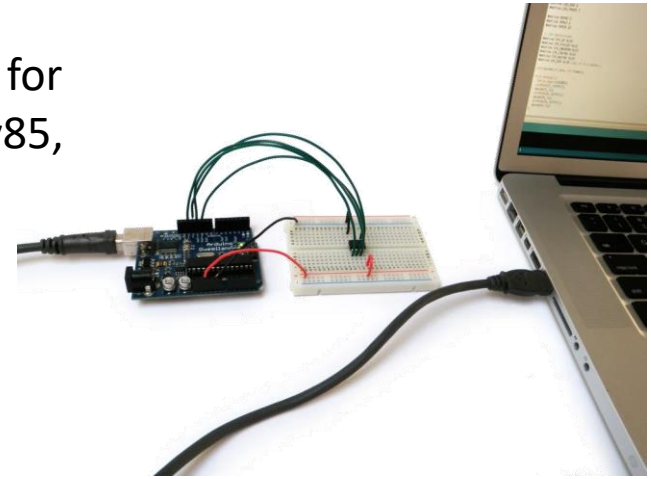
The potentiometer is connected to a pin that will be used as an analog input, which can be used to control the rate at which the LED flashes.

```
// Define the LED and potentiometer pins
#define LED_PIN 4        // PB4 on the ATtiny85
#define POT_PIN A3       // Analog pin A3 (PB3 on the ATtiny85)
void setup() {
  pinMode(LED_PIN, OUTPUT);
  // No need to set POT_PIN as input
  //because analogRead does it automatically
}
void loop() {
  // Read the potentiometer value (0 to 1023=2^10)
  int potValue = analogRead(POT_PIN);
  // Map the potentiometer value to a range for delay
  // (e.g., 100 ms to 1000 ms)
  int delayTime = map(potValue, 0, 1023, 100, 1000);
  // Turn the LED on, wait for delayTime
  digitalWrite(LED_PIN, HIGH);
  delay(delayTime);
  // Turn the LED off, wait for delayTime
  digitalWrite(LED_PIN, LOW);
  delay(delayTime);
}
```

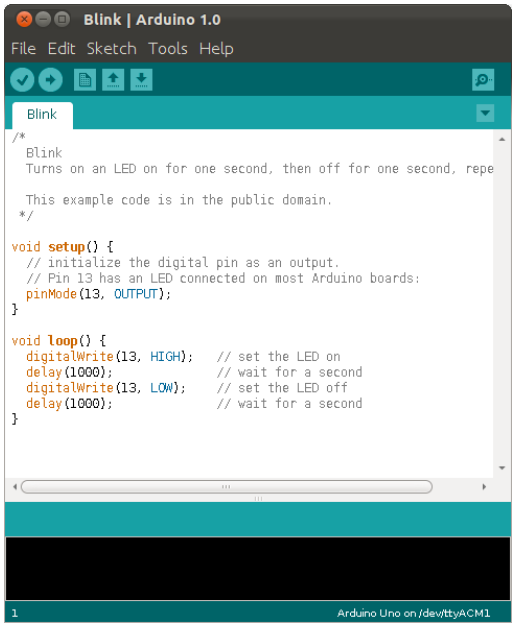


Programming AVR MCU by Arduino Language

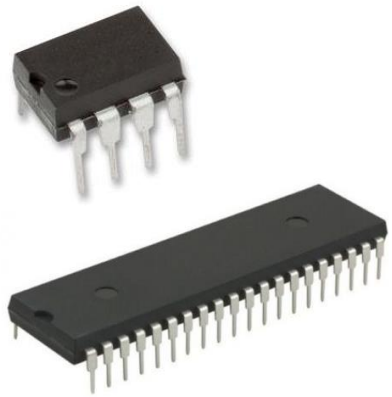
The Arduino IDE can be used to simplify the writing of code for most processors in the AVR 8-bit range, including the ATtiny85, although the size of the compiled code is not efficient.



Arduino IDE



Programmer



Tutorials:
http://highlowtech.org/?p=1695
https://www.instructables.com/id/Program-an-ATtiny-with-Arduino/