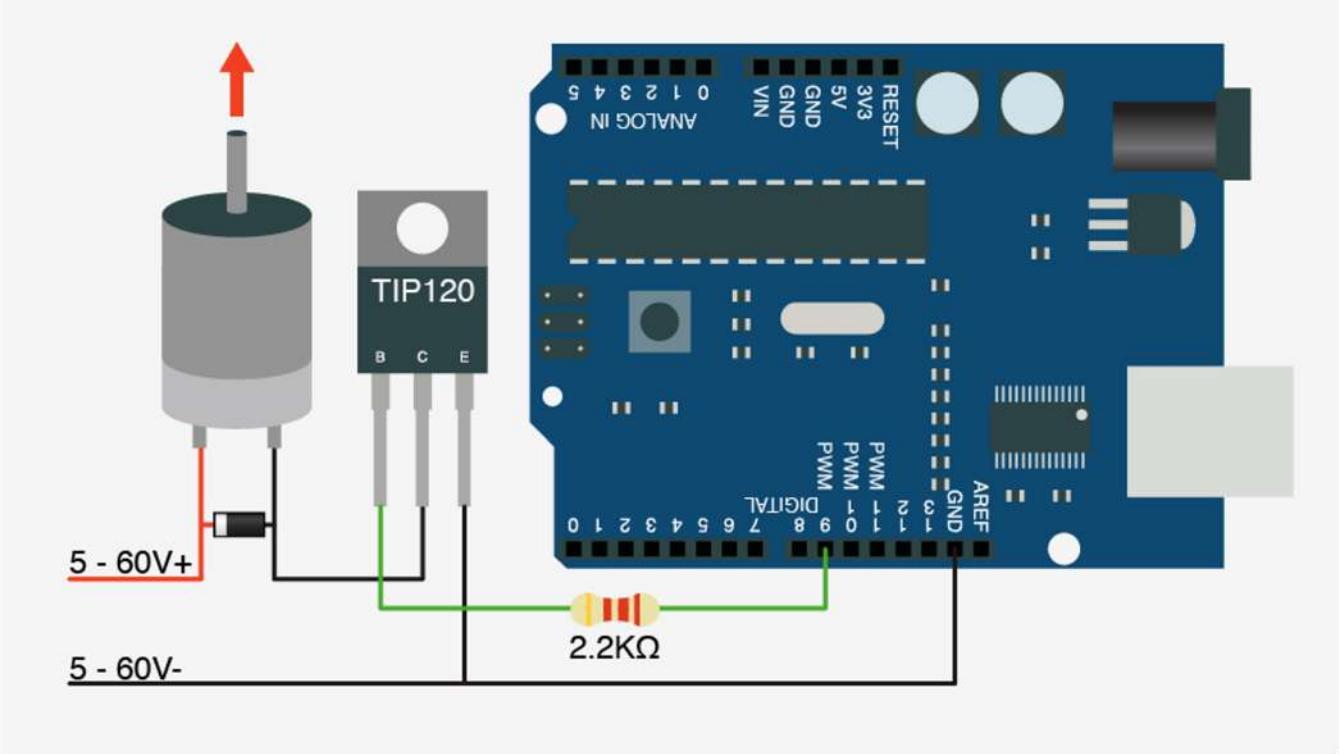
Actuators and Motors

- As with sensors, actuators are a form of **transducer**, i.e. a device that converts variations in a physical quantity (e.g. pressure or brightness) into an electrical signal, or vice versa.
- Commonly used actuators include DC motors, servo motors and stepper motors.
- DC motors:
  - Rotary electrical machines that convert direct current electrical energy into mechanical energy.
  - Primarily used as simple motors (fans, etc.).
- Servo motors:
  - Rotary or linear actuators that allow for precise control of angular or linear **position**, **velocity** and **acceleration**.
  - Used in precision position-control applications.
- Stepper motors:
  - Brushless DC motors that divide a full rotation into a number of equal steps.
  - Digital actuators used in position control applications.

DC Motors

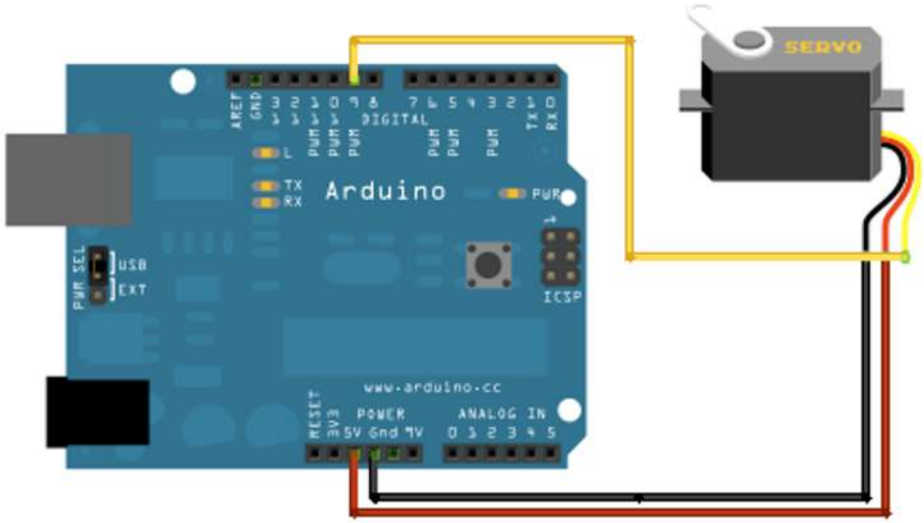
```
int tip120PIN = 9;                  // Choose Arduino's PWM pin 9
void setup(){
   pinMode(tip120PIN, OUTPUT);   // Set pin to be output
   analogWrite(tip120PIN, 255); // 0 to 255 to control motor speed
}
void loop(){
}
```



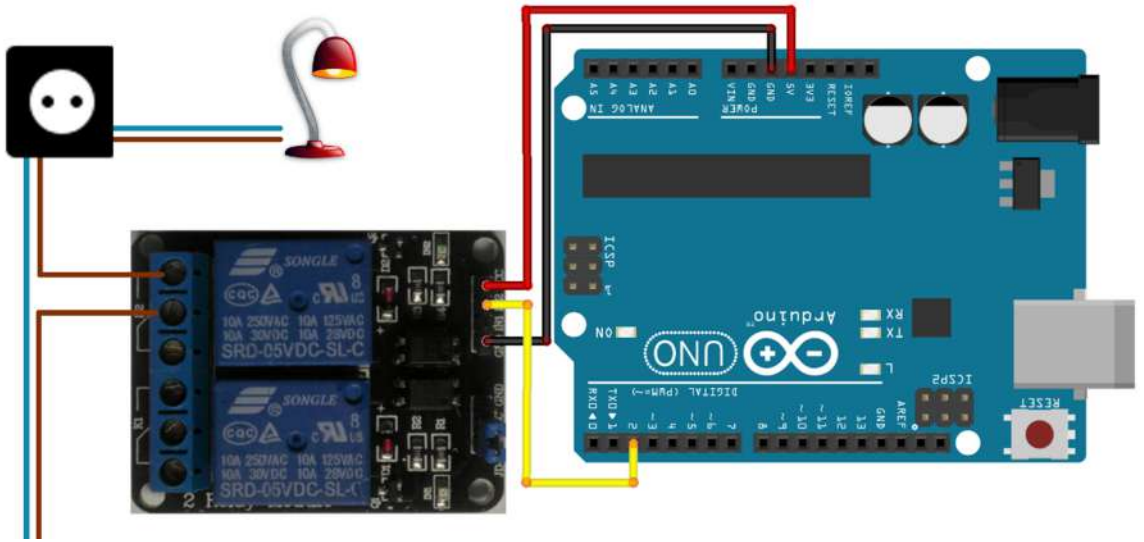
Aside: TIP120: NPN Darlington transistor (large current gain)

Servo Motors

```
#include <Servo.h>

Servo myservo;  // create servo object
int pos = 0;
// variable to store servo position

void setup() {
  myservo.attach(9);
  // attaches the servo on pin 9
}

void loop() {
  for (pos = 0; pos < 180; pos++) {  //  0 to 179 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for (pos = 179; pos >= 0; pos--) { // goes from 179 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```

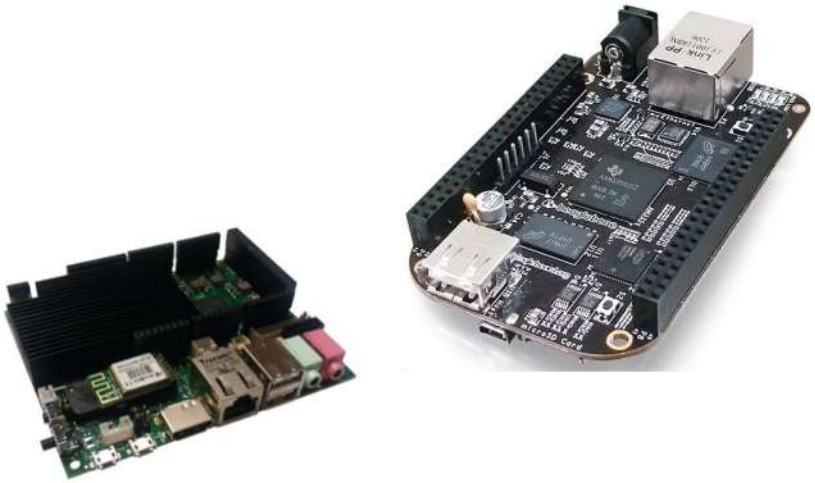


Relays



```
#define RELAY_ON 0
// Here, LOW turns relay on
#define RELAY_OFF 1
#define RELAY_1 2

void setup(){
  pinMode(RELAY_1, OUTPUT); // Set pin as output
  digitalWrite(RELAY_1, RELAY_OFF); // Relay off
}

void loop(){
  digitalWrite(RELAY_1, RELAY_ON); // Turn on
  delay(3000); // wait 3 seconds.
  digitalWrite(RELAY_1, RELAY_OFF); // Turn off
  delay(3000); // wait 3 seconds
}
```

Hardware Platforms



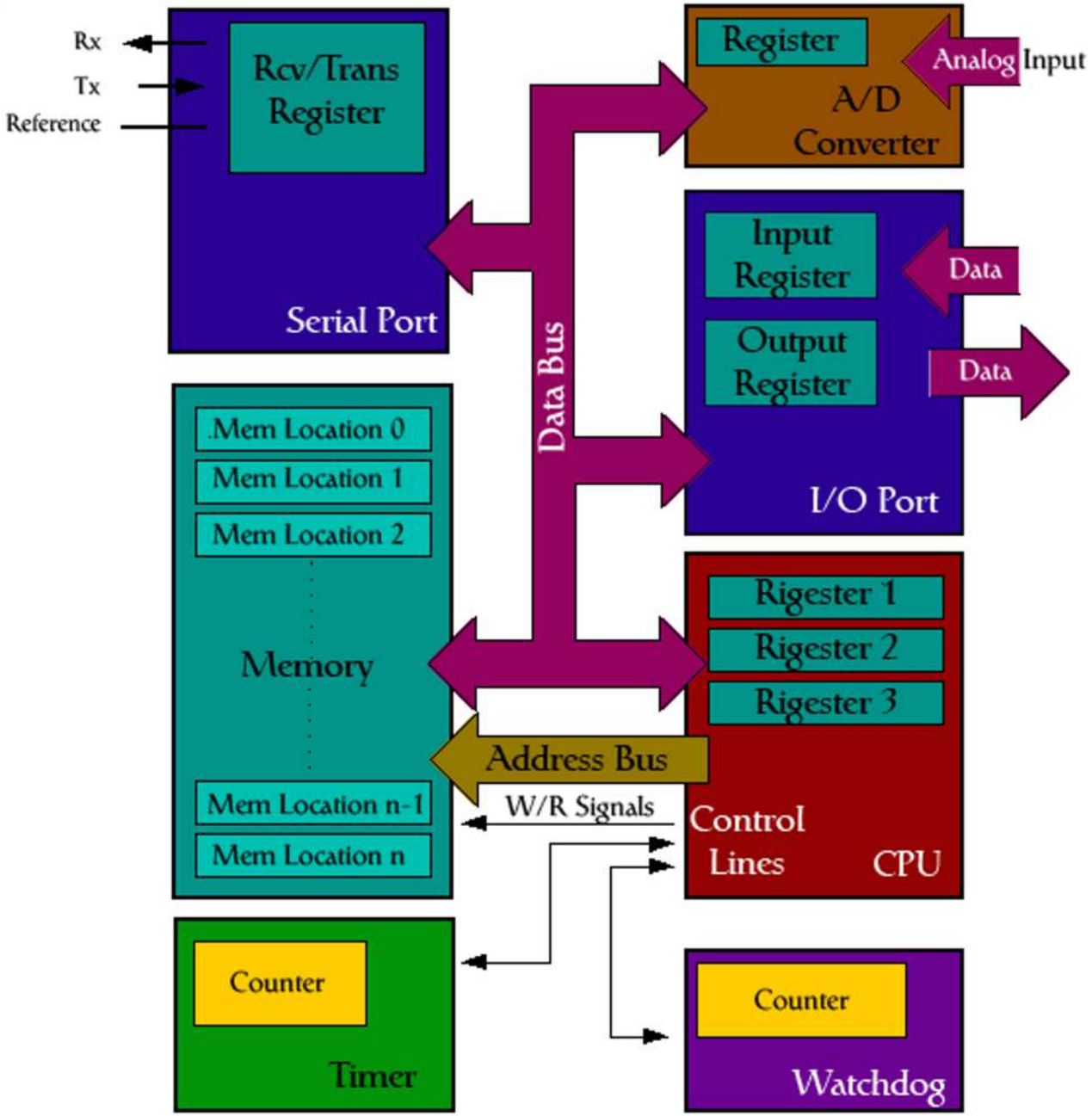
Single-board Computers

Microcontrollers

Microcontrollers

- Interface with outside world via peripherals, e.g. GPIOs, I2C, SPI, UART, timers, USB.
- GPIO: general-purpose input/output.
  - Generic pin on an IC or computer board whose behaviour (including whether it is an input or output pin) is controllable by the user at run time.
- MCUs are typically low power devices and are quick to boot.
- Can have **very rapid response time to stimulus**.
- Very cheap (especially if you buy only IC).
- Require electronics knowledge to connect to sensors.
- Not very powerful → cannot perform relatively complex tasks.

Microcontroller Anatomy



Microcontroller Anatomy

- Most microcontrollers have a USART* block that enables low CPU load communication via I2C, SPI, USB, Ethernet and other standards.
- ARM Cortex-M are the family of ARM processors specifically targeted to microcontroller applications.
- Microcontroller types vary widely:
  - some are designed for signal processing
  - others for sensing
  - others for motor control.
- Microcontrollers are typically categorised by their word length, i.e. an 8-bit microcontroller has an 8-bit word length.

Microcontroller Anatomy

- The watchdog timer is a rolling reset clock → the program must reset it on a regular basis or the microcontroller resets.
- Sometimes called a "computer operating properly" or COP timer.
- Used to detect and recover from computer malfunctions.
- During normal operation, the computer regularly resets the watchdog timer to prevent it from elapsing, or "timing out".
- If, due to a hardware fault or program error, the computer fails to reset the watchdog, the timer will elapse and generate a timeout signal.

JTAG

- JTAG interfaces are used for on-chip debugging of ICs.
- The JTAG standard was designed to assist with device, board, and system testing, diagnosis, and fault isolation.
- Today, JTAG is the primary means of accessing sub-blocks of ICs.
- Specifies the use of a **dedicated debug port** implementing a serial communications interface for low-overhead access without requiring direct external access to the system address and data buses.

Notable Features of Microcontrollers

- A key difference between microcontrollers and Single Board Computers is in their respective **interrupt latencies**.
- Interrupt latency is the time that elapses from when an interrupt is generated to when the source of the interrupt is serviced.
- The minimised memory and integrated nature of microcontrollers means that they typically have an interrupt latency orders of magnitude lower than general purpose computers.
- Microcontrollers must provide **predictable interrupt latency**.

Notable Features of Microcontrollers

- Microcontrollers typically execute only from on-chip memory → limited code space from which to operate.
- E.g. Arduino Uno has:
  - 32 KB Flash memory (of which 0.5 KB is used for the bootloader).
  - 2 KB SRAM (volatile).
  - 1 KB EEPROM.
- Flash memory (program space), is where the Arduino sketch is stored (non-volatile).
- SRAM is where the sketch creates and manipulates variables when it runs (volatile).
- EEPROM is memory space that programmers can use to store long-term information (non-volatile).

Single Board Computers (SBCs)

- Lots of processing power relative to microcontrollers.
- Runs an operating system (Linux, Android, Windows 10).
- Slower response to stimulus than microcontrollers.
- Monitor connection possible and other standard connections.
- As standard connections are used, typically less wiring/electronics is required to connect sensors/actuators.
- Cheap, "weak" computer → general purpose device.

Notable Features of SBCs

- SBCs will typically use a flexible microcontroller but will **sacrifice latency for instruction throughput**.
- SBCs will use some storage for loading OS and will have a large code space.
- Unless running a RTOS (Real Time Operating System), SBCs have a **stochastic interrupt latency**, e.g. may be 21ms at one point and 36ms at another.
- SBCs allow for vertical integration*, e.g. applications such as:
  - Tweeting status updates from sensors.
  - Smart device operation.
  - Clustering and emulation.

*Vertical integration refers to the combination of two or more "stages" that are normally operated separately.

Real Time Operating Systems

- An RTOS is an operating system intended to **serve real-time applications** which process data as it comes in, typically without buffering delays.
- A key characteristic of an RTOS is the level of its consistency concerning the amount of time it takes to accept and complete an application's task.
- The chief design goal is not high throughput, but rather a **guarantee** of a performance category.
- Key factors in an RTOS are minimal interrupt latency and minimal thread switching latency.
- An RTOS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

RISC

- Most Arduino boards consist of an Atmel 8-bit AVR microcontroller.
- E.g. the Uno uses an ATmega328 (operating at 16 MHz).
- All Raspberry Pi models to date feature a Broadcom SoC with an integrated ARM CPU (and on-chip GPU).
- E.g. the Raspberry Pi 3 uses a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor.
- AVR MCUs and ARM processors both use RISC architectures.
- RISC processors typically require fewer transistors than those with a CISC architecture, which improves cost, power consumption, and heat dissipation.
- RISC/CISC: Reduced/Complex Instruction Set Computing.

Instruction Sets

- An instruction set architecture (ISA) includes a specification of the set of opcodes (machine language), and the native commands implemented by a particular processor.
- The instruction set that a CPU uses for its operation is made from short, simple commands, e.g.
  - a single transaction of data (from register to register, memory to register, or register to memory), or
  - a simple operation such as addition of two registers.
- Compare this to mid/high-level languages such as C++, Python or Java for instance.

Assembly vs Machine Language

- In order for the instruction decoder to decipher what an instruction represents, the instruction itself must be a number (a set of bits, i.e. a binary number).
- These numbers are referred to as machine code.
- Machine code is the instruction set that the processor uses.
- However, humans understand words so each machine code is given a lexical equivalent.
- These instructions in text form are called assembly language.
- i.e. an assembly language is a low-level programming language that implements a symbolic representation of the binary machine codes and other constants needed to program a given CPU architecture.

Aside: Assembly Language

- Disadvantages of assembly language:
  - Complicated to learn and use.
  - Difficult to debug.
  - More time consuming to write.
  - Non-portable (i.e. unable to be directly transferred to a different processor).
  - Difficult to decipher if programmer is unfamiliar with language.

Aside: Assembly Language

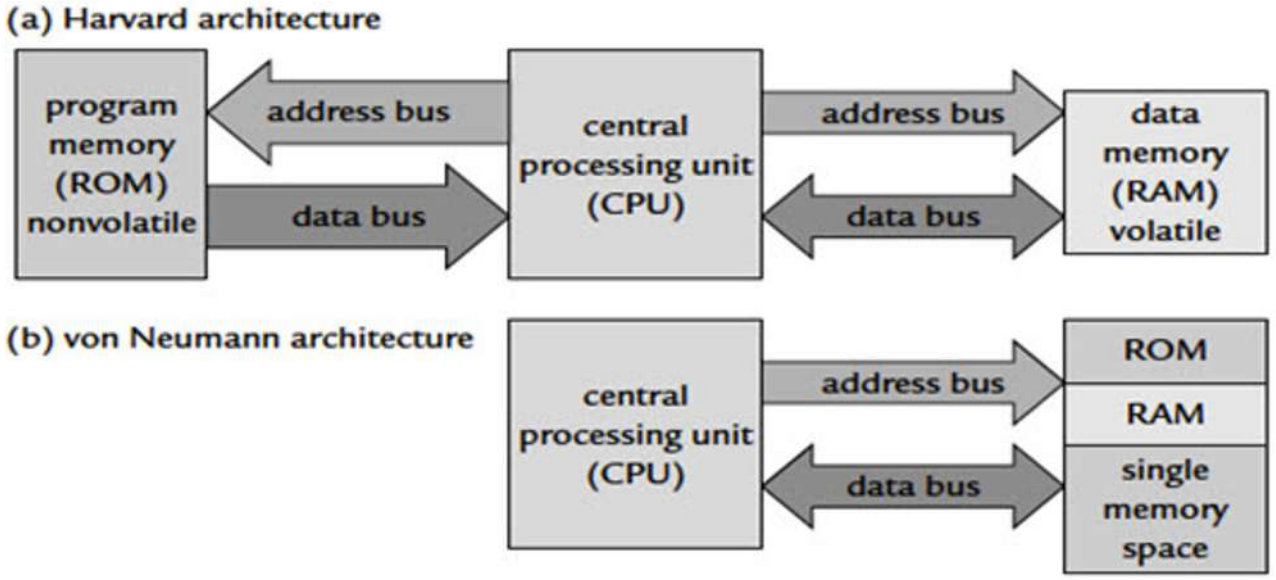
- In contrast to high-level languages, which hide the details of CPU operations such as memory access models and the manipulation of CPU registers, assembly language allows the programmer to work much closer to the electronics of the processor (details of the processor are not hidden by the operating system or compiler).
- Advantages of assembly language:
  - Gives programmer full access to all processor resources.
  - Ability to make code run much faster.
  - Ability to make code far more compact.

RISC vs CISC

- The typical differentiating characteristics between modern RISC and CISC designs is that most RISC designs use:
  - Uniform instruction length for almost all instructions.
  - Strictly separate load/store instructions.
- A modern RISC processor can be much more complex than, say, a modern microcontroller using a CISC instruction set, especially in terms of electronic circuit complexity, but also in terms of the number of instructions or the complexity of their encoding patterns.
- The x86 ISA is an example of CISC.
- Examples of RISC include ARM, Atmel AVR, and MIPS.

Von Neumann vs Harvard architectures

- Von Neumann:
  - Same memory holds data and instructions.
  - There is a single set of address/data buses between CPU and memory.
- Harvard:
  - Separate memories for data and instructions.
  - Two sets of address/data buses between CPU and memory.



Modified Harvard architectures

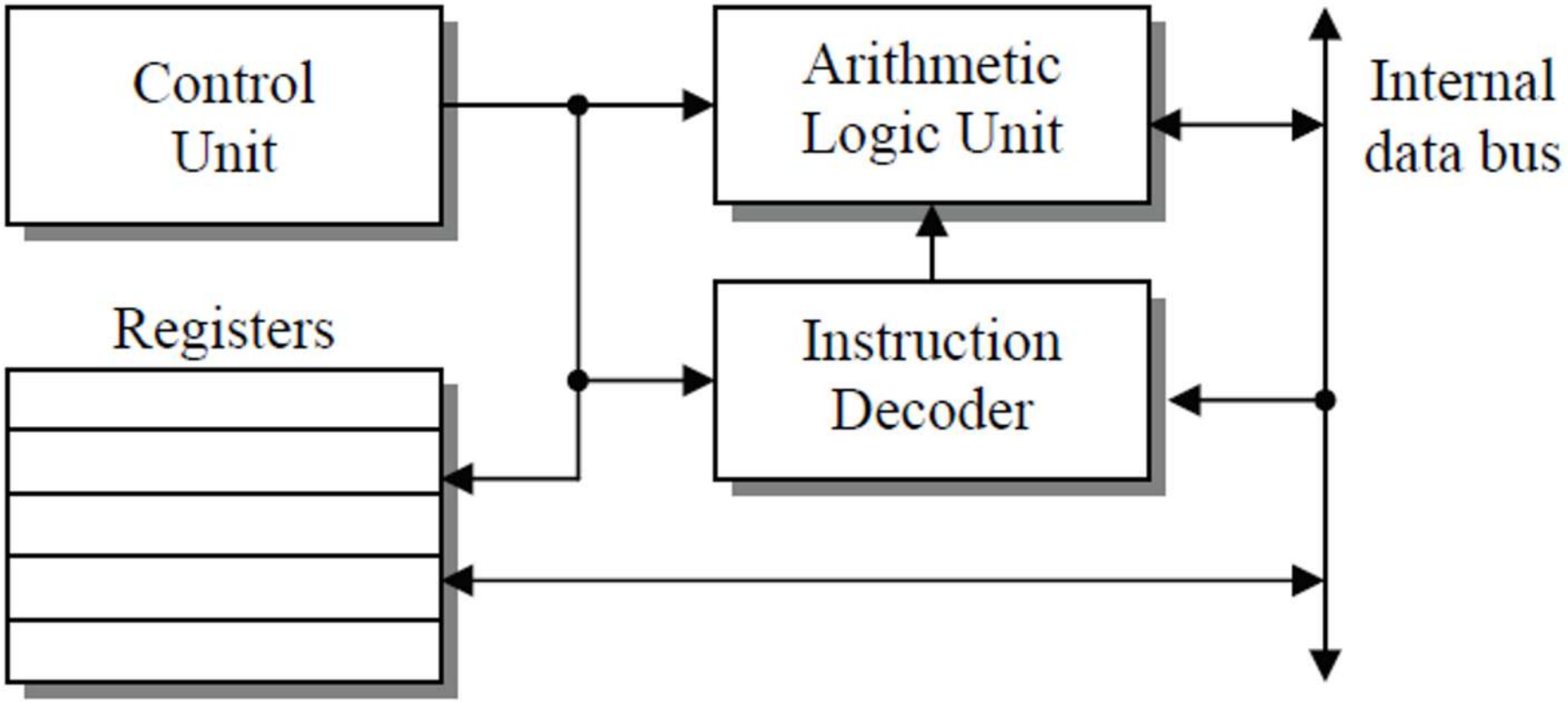
- Modified Harvard architecture: a variation of the Harvard architecture that allows the contents of the instruction memory to be accessed as if it were data.
- AVR and most ARM processors use modified Harvard architectures.
- In the case of AVR MCUs, program instructions and data are stored in separate physical memory systems that appear in different address spaces (but data items may be read from program memory using special instructions).
- Recall that an Arduino Uno has 32 KB flash memory for program instructions and additional 1 KB EEPROM for data.

Aside: Von Neumann bottleneck

- Since CPU speed and memory size have increased much faster than the throughput between them, the bottleneck has become more of a problem, the severity of which increases with every newer generation of CPU.
- There are several methods for mitigating the Von Neumann performance bottleneck:
  - Providing cache memory in the CPU.
  - Providing separate caches or separate access paths for data and instructions (Modified Harvard architecture).
  - Using branch prediction algorithms.

CPUs

- Generic block diagram of a typical CPU:



CPUs

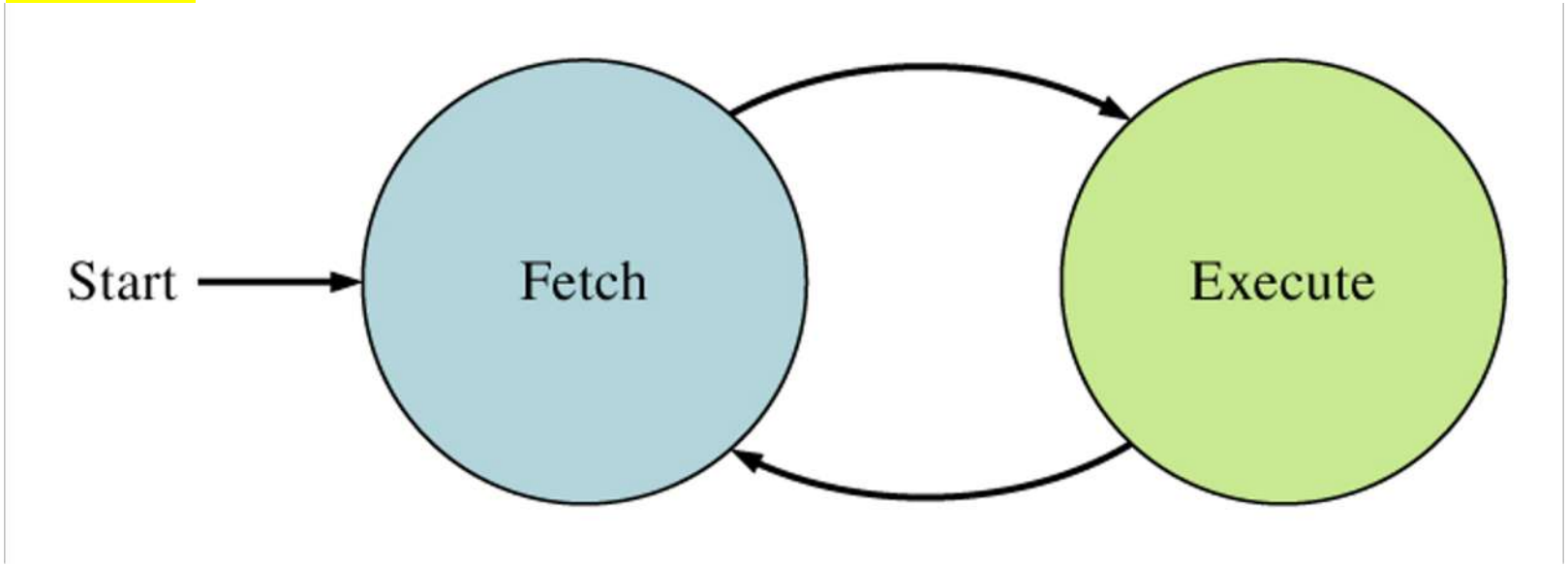
- ALU:
  - A collection of logic circuits designed to perform arithmetic and logical operations.
  - The “calculator of the CPU”.
- Control Unit:
  - Implements the Fetch and Execute cycle. Instructs the computer system on how to follow the program instructions.
  - Orchestrates the fetching (from memory) and execution of instructions by directing the coordinated operations of the ALU, registers and other components.
  - Essentially, the control unit knows the big picture of what needs to be done and which of the CPU’s components can do it, and it controls the timing to do it.

CPUs

- Instruction decoder:
  - Receives the instruction (stored as a binary value) from memory.
  - Interprets the value to see what instruction is to be performed.
  - Tells the ALU and registers which circuits to energise in order to perform the function.
- Registers:
  - Used to **store** the data, addresses, instructions, and flags (which represent the status of an operation or of the processor itself) that are in use by the CPU.

CPU Instruction Cycle

- The fundamental operation of most CPUs is to execute a sequence of stored instructions (i.e. a program).
- Nearly all CPUs follow the fetch, decode and execute steps in their operation, which are collectively known as the instruction cycle.
- After the execution of an instruction, the entire process repeats.



CPU Instruction Cycle

- The next instruction cycle normally fetches the next-in-sequence instruction because of the incremented value in the program counter.
- The program counter is a processor register that holds the memory address of the next instruction to be executed.
- After each instruction is fetched, the program counter is updated to point to the next instruction.
- The program counter is sometimes referred to as the instruction pointer.
- If a jump instruction was executed, the program counter will be modified to contain the address of the instruction that was jumped to and program execution continues normally.

Pipelined Architectures

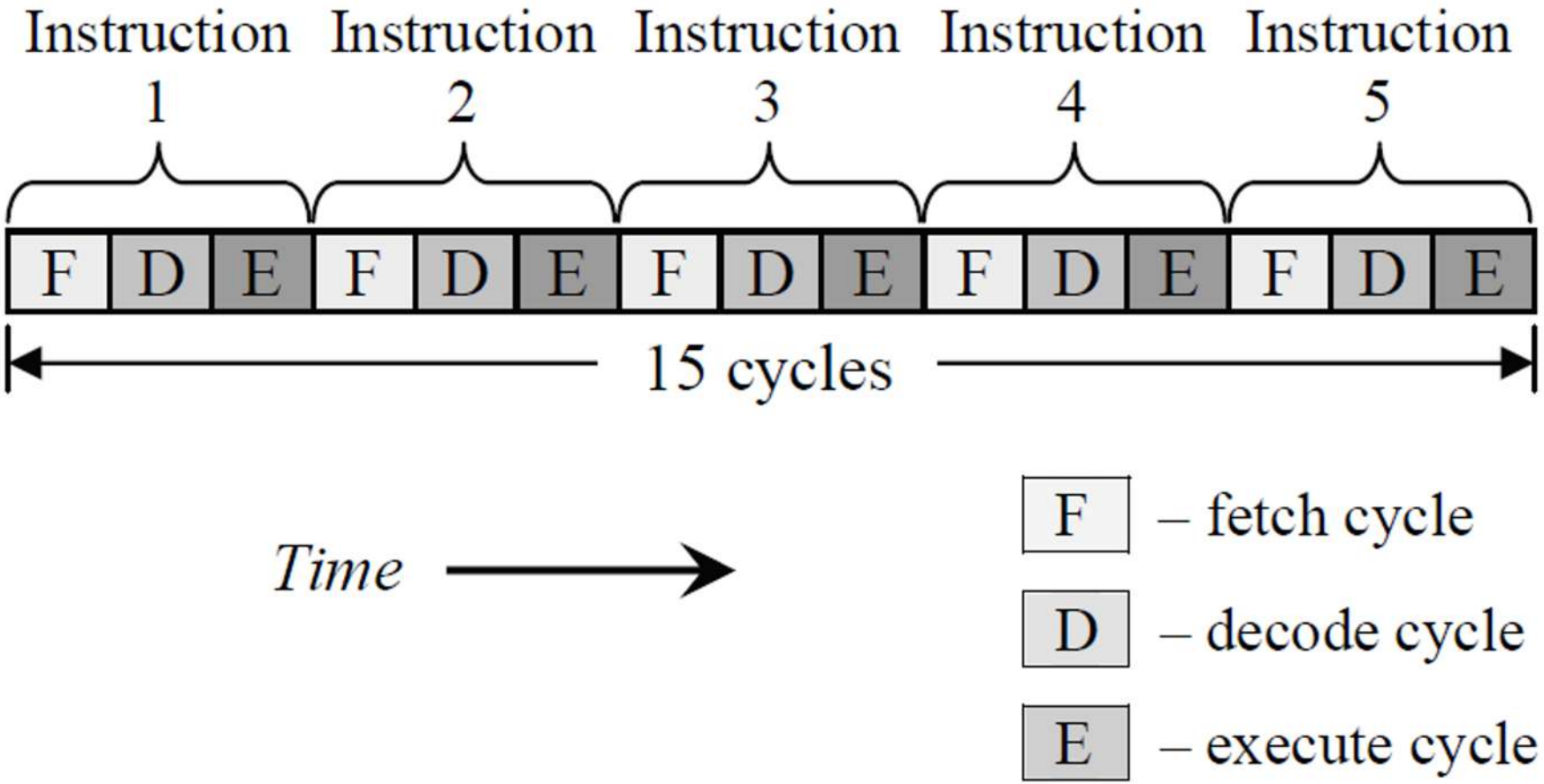
- Pipelining is the process of creating a queue of fetched, decoded and executed instructions in order to improve the performance of a processor.
- E.g. Atmel's AVRs have a two-stage, single-level pipeline design.
  - The next machine instruction is fetched as the current one is executing.
  - Most instructions take just one or two clock cycles, making AVRs relatively fast among eight-bit microcontrollers.
- Pipelining is based on the fact that **each step in the instruction cycle (fetch, decode, execute) uses different components of the CPU and may be performed in parallel rather than sequentially**.

Pipelined Architectures

- Performance is optimised by attempting to predict what each CPU component should be doing as soon as it finishes its current task (rather than wait for the next instruction).
- Even if the prediction was wrong, nothing is lost; the result is simply ignored.
- However, if the prediction was correct, time has been saved and the code is executed faster.
- Each step of the instruction cycle uses different components of the CPU:
  - The internal data bus and the program counter perform the fetch.
  - The instruction decoder performs the decode cycle.
  - The ALU and CPU registers are responsible for the execute cycle.

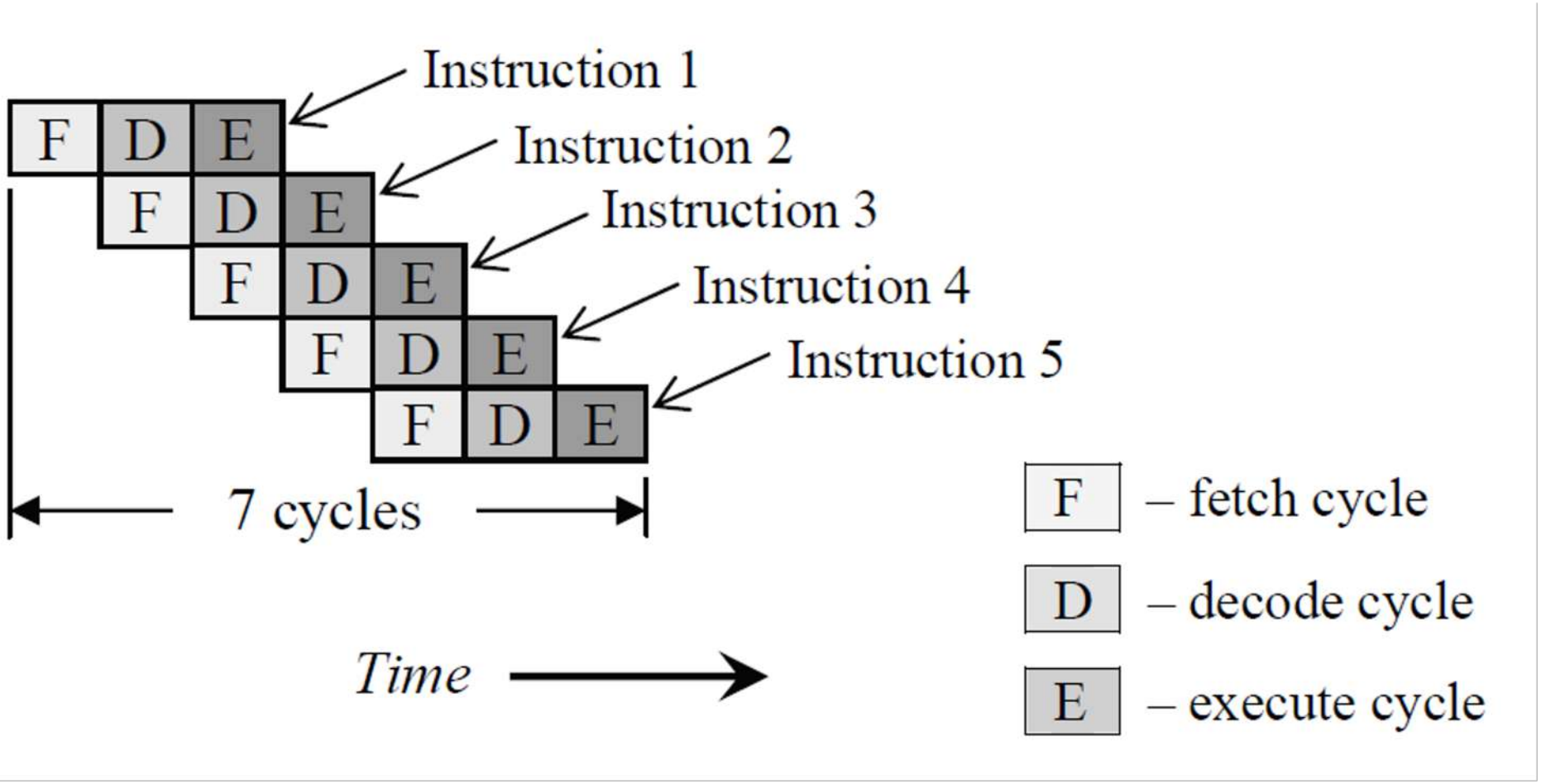
Pipelined Architectures

- Non-pipelined execution of five instructions:



Pipelined Architectures

- Pipelined execution of five instructions:



Pipelined Architectures

- In a non-pipelined architecture, a full instruction cycle must be performed before the next instruction can be fetched.
- This sequential execution of instructions allows for a very simple CPU hardware but it leaves each portion of the CPU idle for two out every three cycles:
  - During the fetch cycle, the instruction decoder and ALU are idle.
  - During the decode cycle, the bus interface and ALU are idle.
  - During the execute cycle, the bus interface and instruction decoder are idle.
- For the pipelined processor, on the other hand, once the bus interface has fetched an instruction and passed it to the instruction decoder for decoding, it can begin its fetch of the next instruction.

Pipelined Architectures

- Note that the first cycle only has a fetch operation, the second cycle has both fetch and decode happening simultaneously, and, by the third cycle, all three operations are happening in parallel.
- Without pipelining, five instructions take 15 cycles to execute.
- In a pipelined architecture, those same 5 instructions take only 7 cycles to execute, a saving of over 50%.

Question: What would the saving be for 100 instructions?

Pipelined Architectures

- In general, the number of cycles it takes for a non-pipelined architecture using three cycles to execute an instruction is three times the number of instructions:

Number of cycles = 3 × number of instructions

- For the pipelined architecture, it takes two cycles to "fill the pipe" so that all three CPU components are fully occupied.
- Once this occurs, then an instruction is executed every cycle.

Number of cycles = 2 + number of instructions

- As the number of instructions grows, the number of cycles required for a pipelined architecture approaches one-third that of a non-pipelined architecture.

Questions

- What is a transducer?
- What is an actuator?
- Fill in the blanks: A DC motor is a rotary electrical machine that converts _____ energy into _____ energy.
- A servo motor allows for precise control of what three parameters?
- What is a stepper motor?
- When programming an Arduino, what function can be used to control the speed of a DC motor?
- Which Arduino pins can be used to:
  - Turn a DC motor on and off?
  - Control the speed of a DC motor?

Questions

- When programming an Arduino, what four steps are required to control the position of a:
  - Servo motor?
  - Stepper motor?
- What is a relay and what is it commonly used for?
- Give one example of each of the following actuators:
  - Heat actuator.
  - Light actuator.
  - Sound actuator.

Questions

- What is a watchdog timer?
- What is a SOC?
- What is JTAG used for?
- What does GPIO stand for?
- What is a RTOS?