Topics:

Microcontrollers

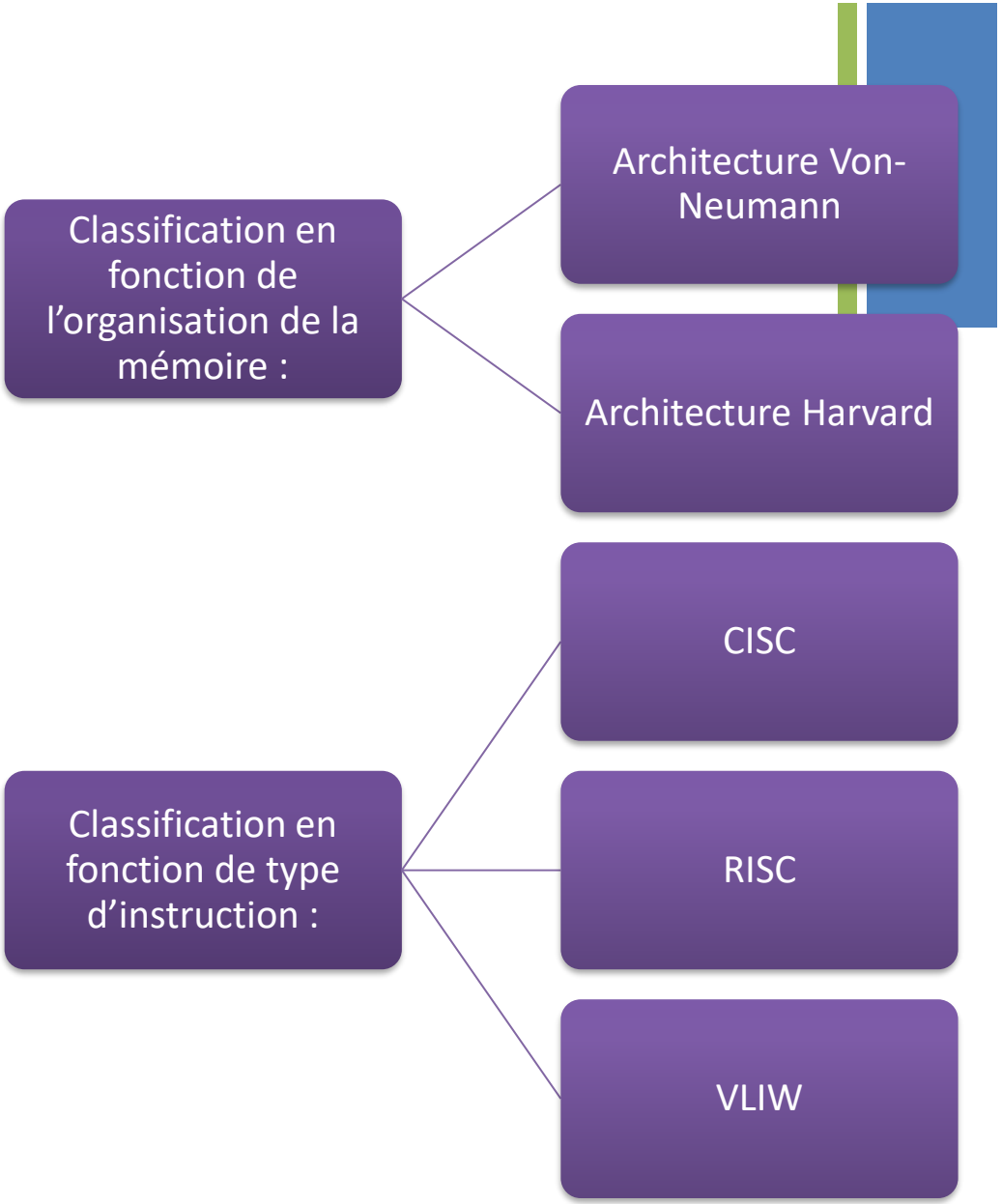
Programming Basics: structure and variables

Digital Output

Analog to Digital Conversion

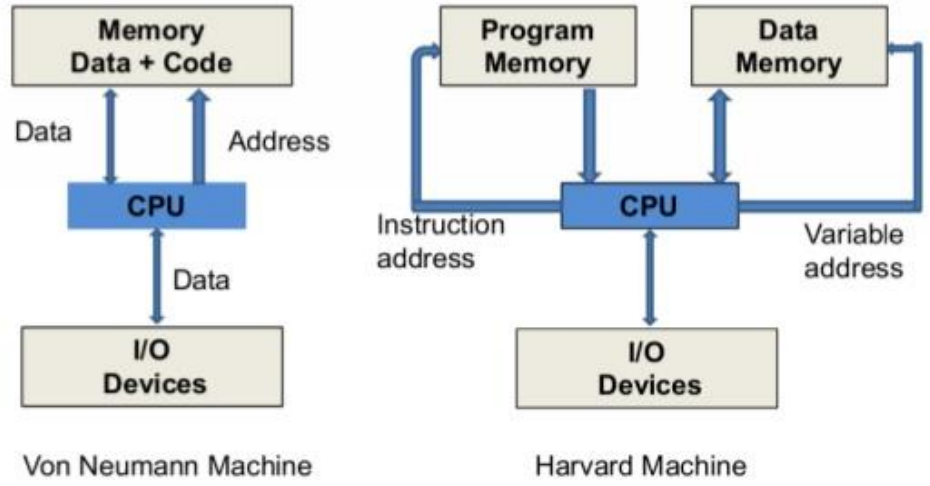
Arduino

Classification des microprocesseurs



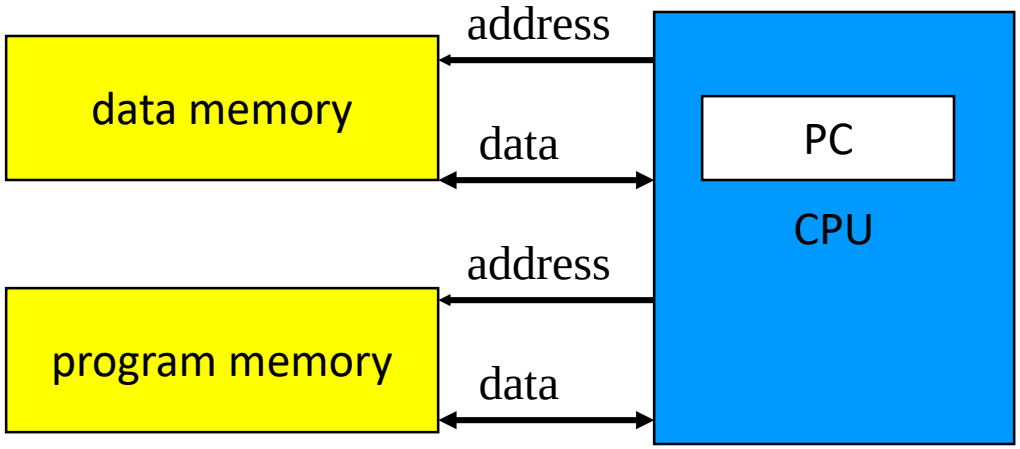
Architectures : Von Neumann versus Harvard

Von Neumann vs. Harvard Architecture

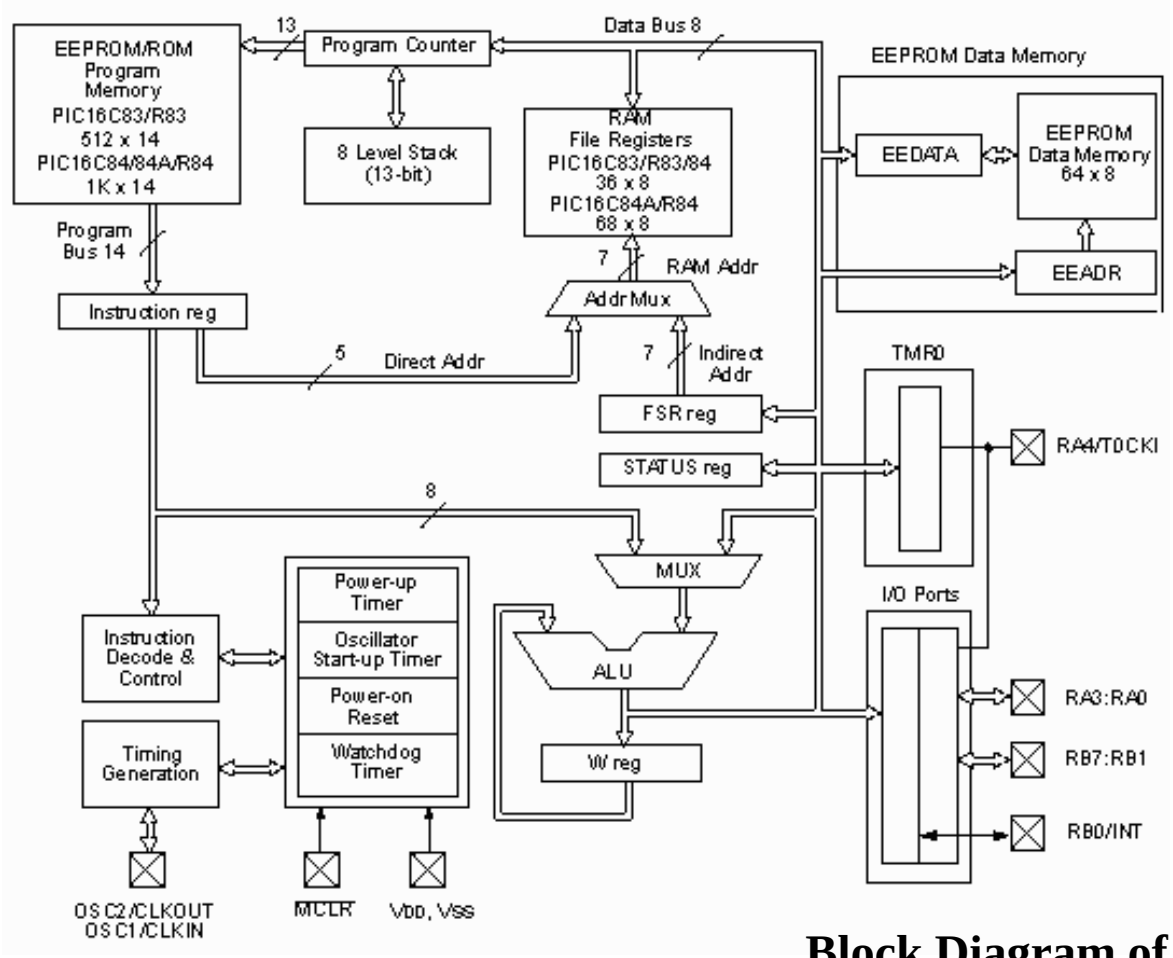


6

Harvard architecture

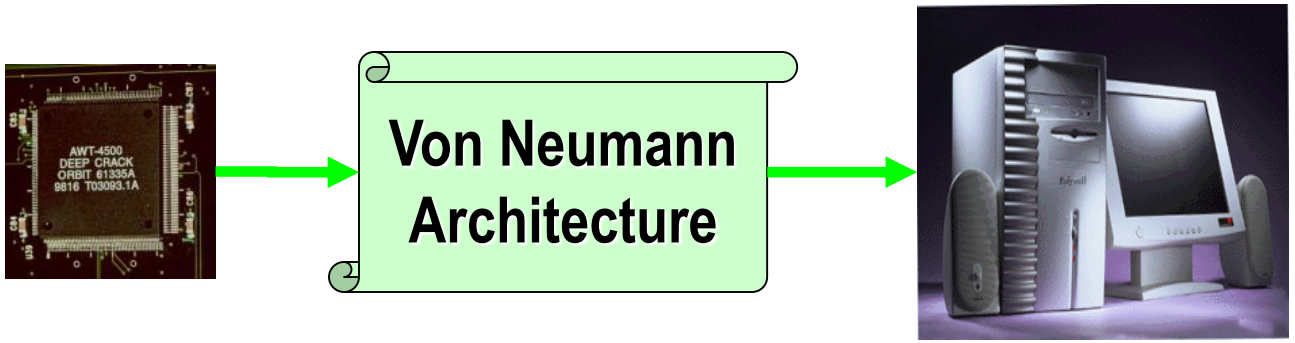


Harvard Architecture Example



Block Diagram of the PIC16C8X

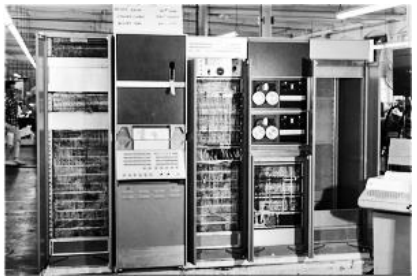
The Von Neumann Architecture



Designing Computers

- All computers more or less based on the same basic design, the Von Neumann Architecture!

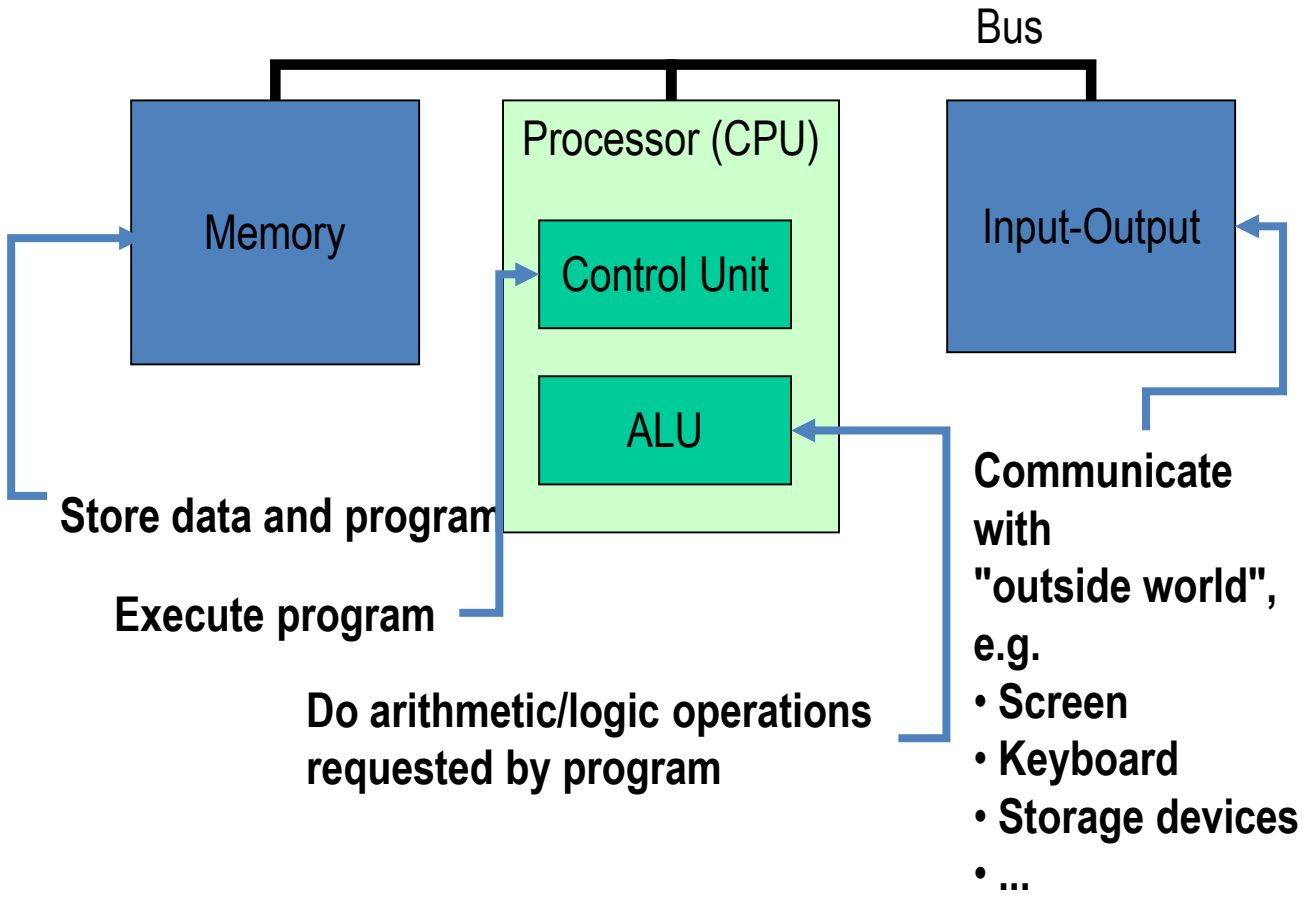


The Von Neumann Architecture

- Model for designing and building computers, based on the following three characteristics:
  1) The computer consists of four main sub-systems:
     - Memory
     - ALU (Arithmetic/Logic Unit)
     - Control Unit
     - Input/Output System (I/O)
  2) Program is stored in memory during execution.
  3) Program instructions are executed sequentially.

The Von Neumann Architecture



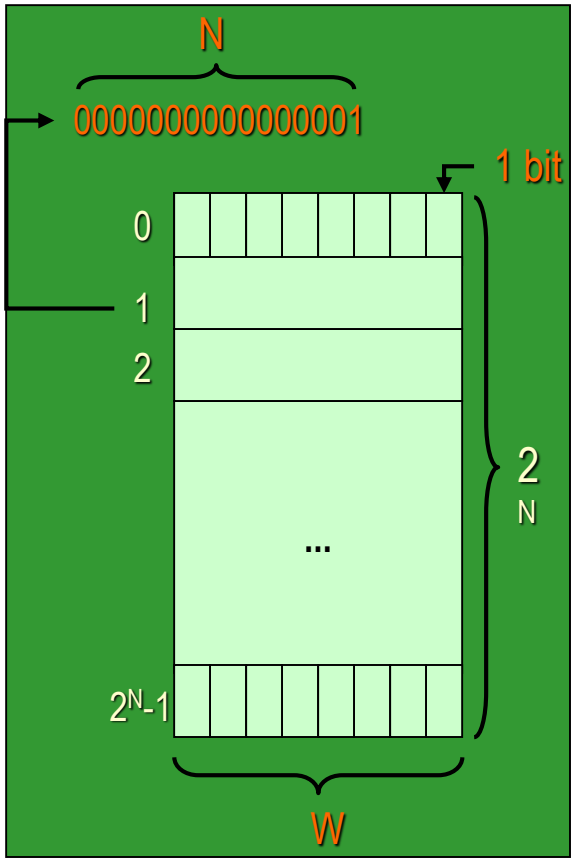
Memory Subsystem

- Memory, also called RAM (Random Access Memory),
  - Consists of many memory cells (storage units) of a fixed size.
    Each cell has an address associated with it: 0, 1, …
  - All accesses to memory are to a specified address.
    A cell is the minimum unit of access (fetch/store a complete cell).
  - The time it takes to fetch/store a cell is the same for all cells.
- When the computer is running, both
  - Program
  - Data (variables)

  are stored in the memory.

RAM

- Need to distinguish between
  - the address of a memory cell and the content of a memory cell
- Memory width (W):
  - How many bits is each memory cell, typically one byte (=8 bits)
- Address width (N):
  - How many bits used to represent each address, determines the maximum memory size = address space
  - If address width is N-bits, then address space is 2^N (0,1,...,2^N-1)



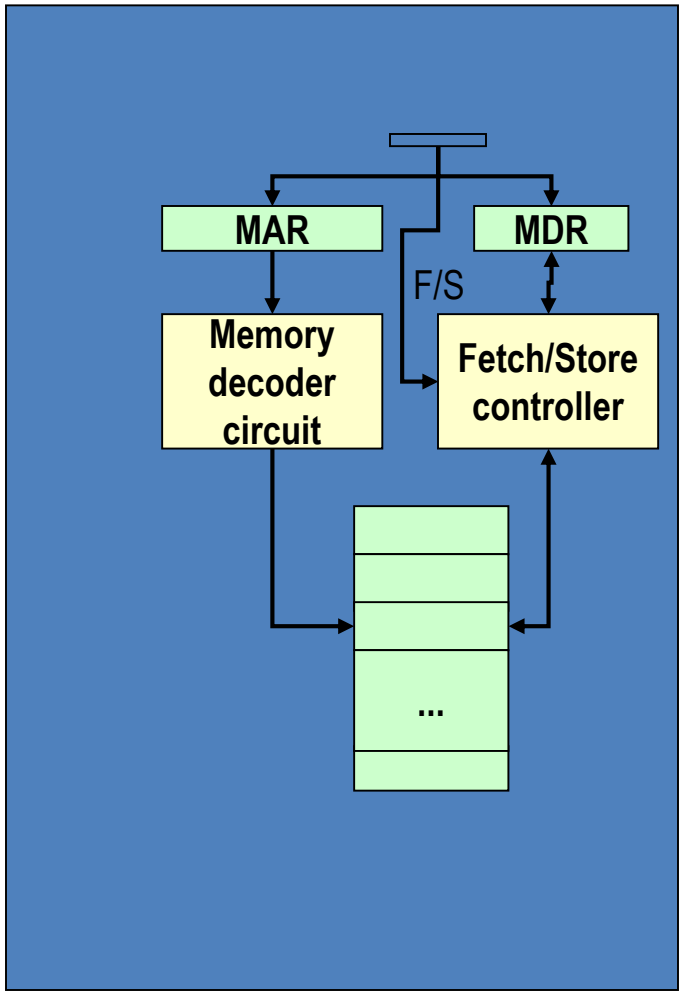
Memory Size / Speed

- Typical memory in a personal computer (PC):
  - 64MB - 256MB
- Memory sizes:
  - Kilobyte (KB) = 2^{10} = 1,024 bytes ~ 1 thousand
  - Megabyte(MB) = 2^{20} = 1,048,576 bytes ~ 1 million
  - Gigabyte (GB) = 2^{30} = 1,073,741,824 bytes ~ 1 billion
- Memory Access Time (read from/ write to memory)
  - 50-75 nanoseconds (1 nsec. = 0.000000001 sec.)
- RAM is
  - volatile (can only store when power is on)
  - relatively expensive

Operations on Memory

- Fetch (address):
  - Fetch a copy of the content of memory cell with the specified address.
  - Non-destructive, copies value in memory cell.
- Store (address, value):
  - Store the specified value into the memory cell specified by address.
  - Destructive, overwrites the previous value of the memory cell.
- The memory system is interfaced via:
  - Memory Address Register (MAR)
  - Memory Data Register (MDR)
  - Fetch/Store signal

Structure of the Memory Subsystem



- Fetch(address)
  - Load address into MAR.
  - Decode the address in MAR.
  - Copy the content of memory cell with specified address into MDR.
- Store(address, value)
  - Load the address into MAR.
  - Load the value into MDR.
  - Decode the address in MAR
  - Copy the content of MDR into memory cell with the specified address.

Input/Output Subsystem

Handles devices that allow the computer system to:

- Communicate and interact with the outside world
  - Screen, keyboard, printer, ...
- Store information (mass-storage)
  - Hard-drives, floppies, CD, tapes, …

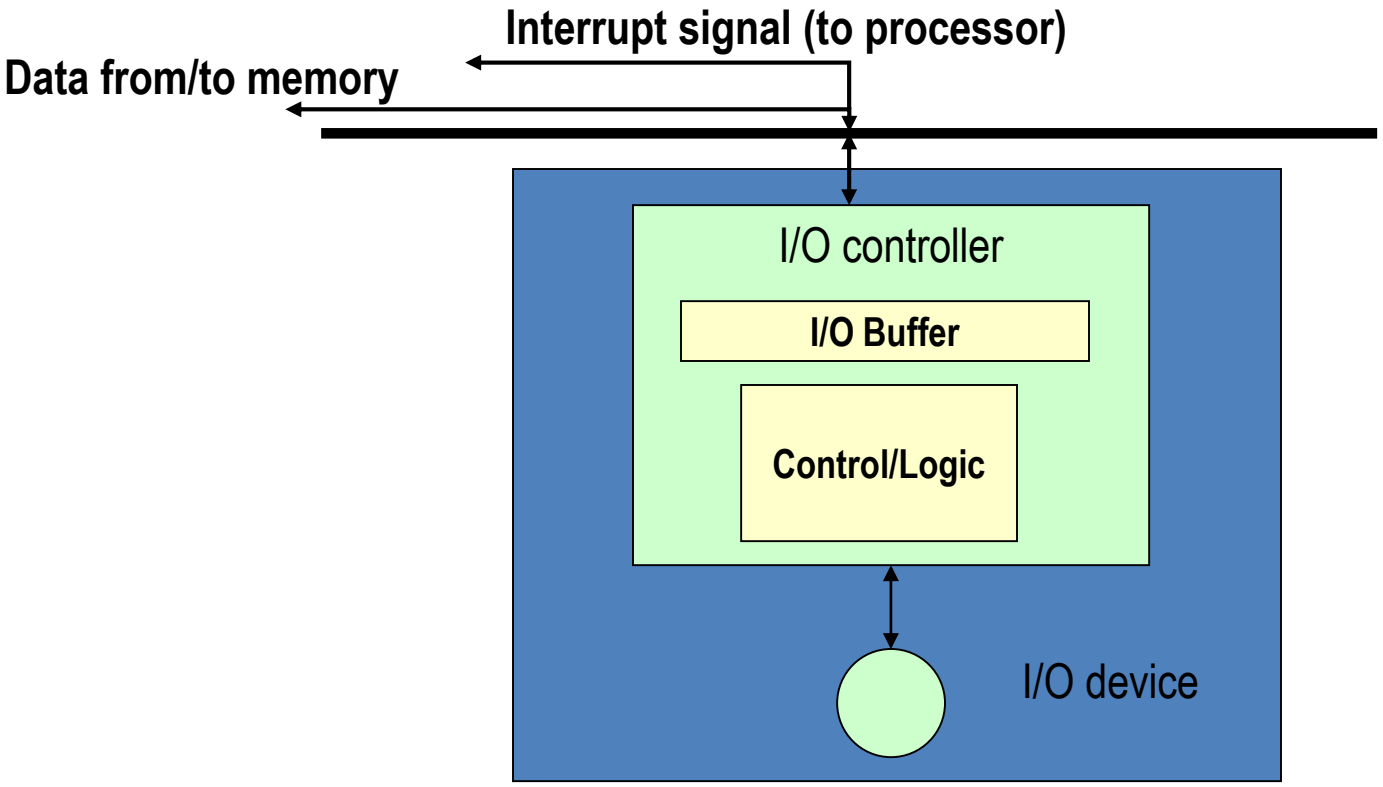
Mass-Storage Device Access Methods:

- Direct Access Storage Devices (DASDs)
  - Hard-drives, floppy-disks, CD-ROMs, ...
- Sequential Access Storage Devices (SASDs)
  - Tapes (for example, used as backup devices)

I/O Controllers

- Speed of I/O devices is slow compared to RAM
  - RAM ~ 50 nsec.
  - Hard-Drive ~ 10msec. = (10,000,000 nsec)
- Solution:
  - I/O Controller, a special purpose processor:
    - Has a small memory buffer, and a control logic to control I/O device (e.g. move disk arm).
    - Sends an interrupt signal to CPU when done read/write.
  - Data transferred between RAM and memory buffer.
  - Processor free to do something else while I/O controller reads/writes data from/to device into I/O buffer.

Structure of the I/O Subsystem



The ALU Subsystem

The ALU (Arithmetic/Logic Unit) performs

- mathematical operations (+, -, x, /, …)
- logic operations (=, <, >, and, or, not, ...)

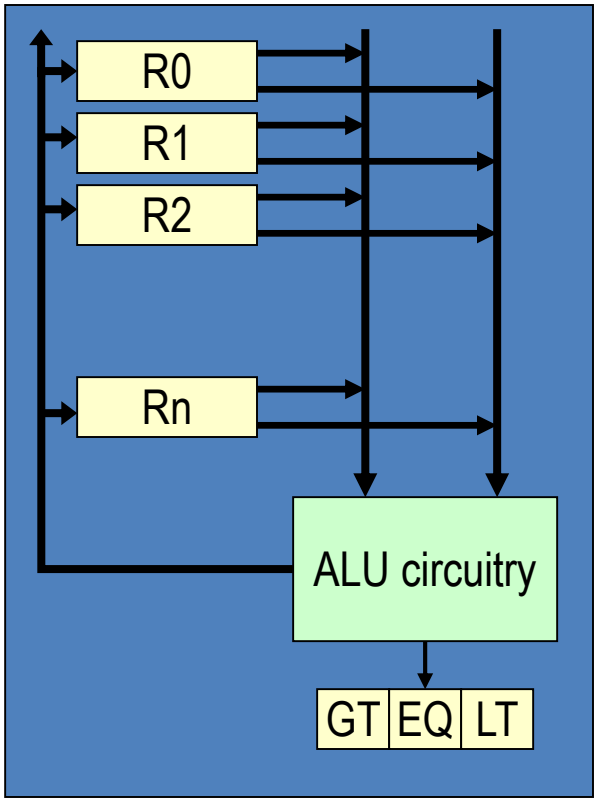
In today's computers integrated into the CPU

Consists of:

- Circuits to do the arithmetic/logic operations.
- Registers (fast storage units) to store intermediate computational results.
- Bus that connects the two.

Structure of the ALU

- Registers:
  - Very fast local memory cells, that store operands of operations and intermediate results.
  - CCR (condition code register), a special purpose register that stores the result of <, = , > operations
- ALU circuitry:
  - Contains an array of circuits to do mathematical/logic operations.
- Bus:
  - Data path interconnecting the registers to the ALU circuitry.



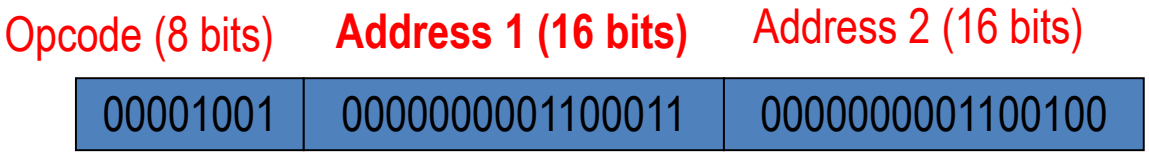
The Control Unit

- Program is stored in memory
  - as machine language instructions, in binary
- The task of the control unit is to execute programs by repeatedly:
  - Fetch from memory the next instruction to be executed.
  - Decode it, that is, determine what is to be done.
  - Execute it by issuing the appropriate signals to the ALU, memory, and I/O subsystems.
  - Continues until the HALT instruction

Machine Language Instructions

- A machine language instruction consists of:
  - Operation code, telling which operation to perform
  - Address field(s), telling the memory addresses of the values on which the operation works.
- Example: ADD X, Y

  X and Y, and store back in memory location Y).
- Assume: opcode for ADD is 9, and addresses X=99, Y=100



Instruction Set Design

- Two different approaches:
  - Reduced Instruction Set Computers (RISC)
    - Instruction set as small and simple as possible.
    - Minimizes amount of circuitry -> faster computers
  - Complex Instruction Set Computers (CISC)
    - More instructions, many very complex
    - Each instruction can do more work, but require more circuitry.

Typical Machine Instructions

Notation:

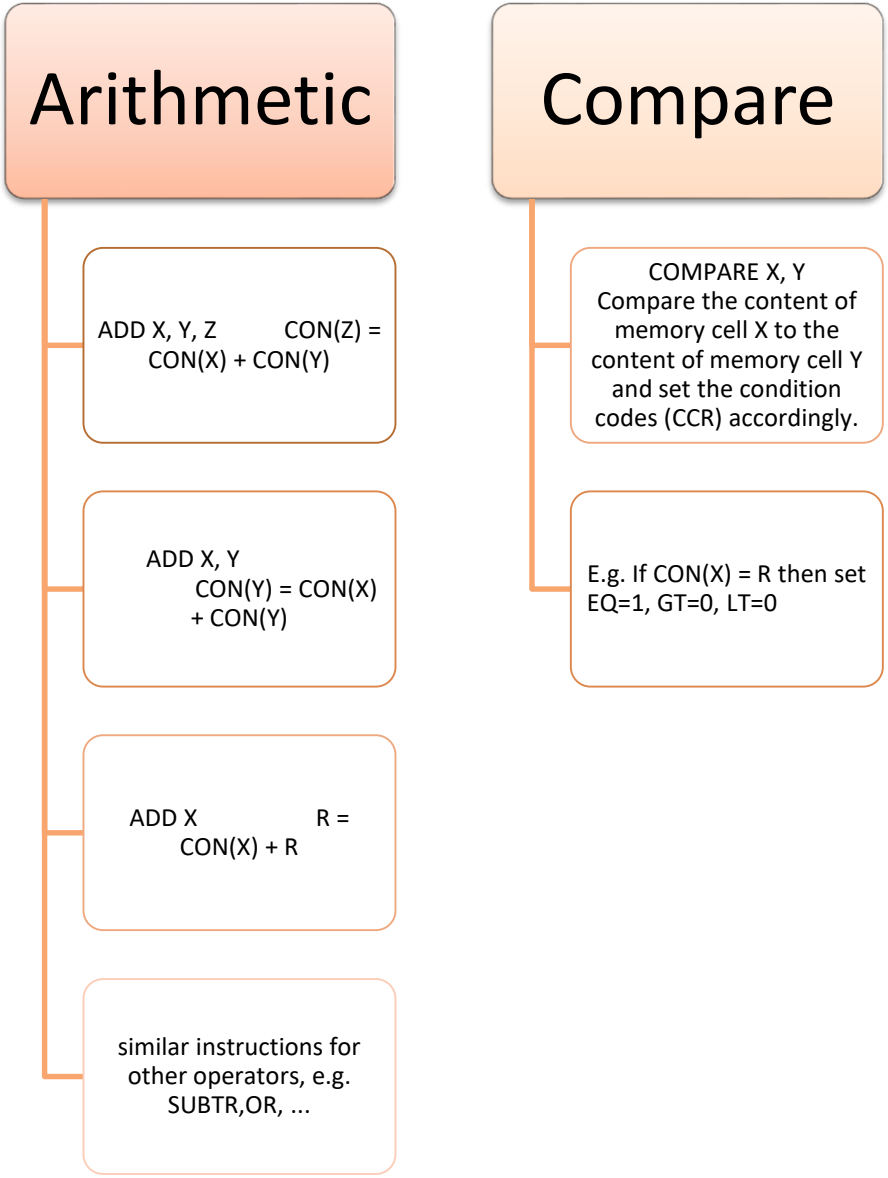
- We use X, Y, Z to denote RAM cells
- Assume only one register R (for simplicity)
- Use English-like descriptions (should be binary)

Data Transfer Instructions

- LOAD X — Load content of memory location X to R
- STORE X — Load content of R to memory location X
- MOVE X, Y — Copy content of memory location X to loc. Y

  (not absolutely necessary)

Machine Instructions (cont.)



Machine Instructions (cont.)

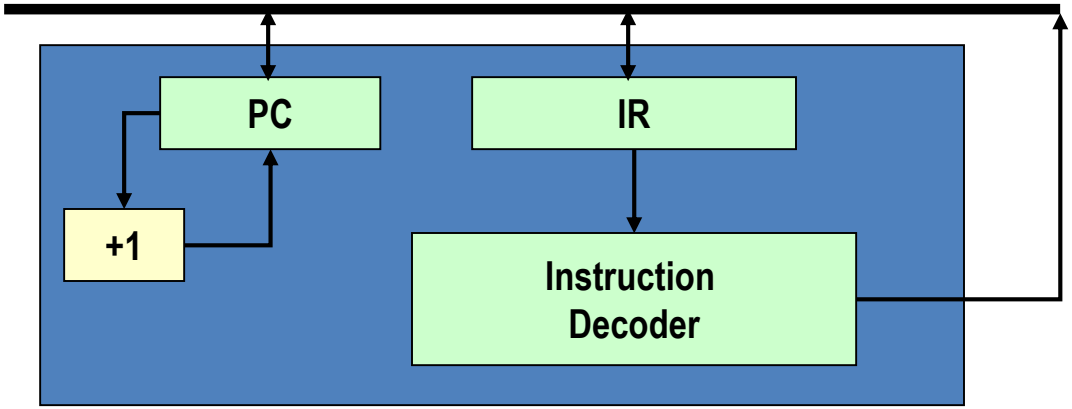
- Branch
  - JUMP X Load next instruction from memory loc. X
  - JUMPGT X Load next instruction from memory loc. X only if GT flag in CCR is set, otherwise load statement from next sequence loc. as usual.
    - JUMPEQ, JUMPLT, JUMPGE, JUMPLE,JUMPNEQ
- Control
  - HALT Stop program execution.

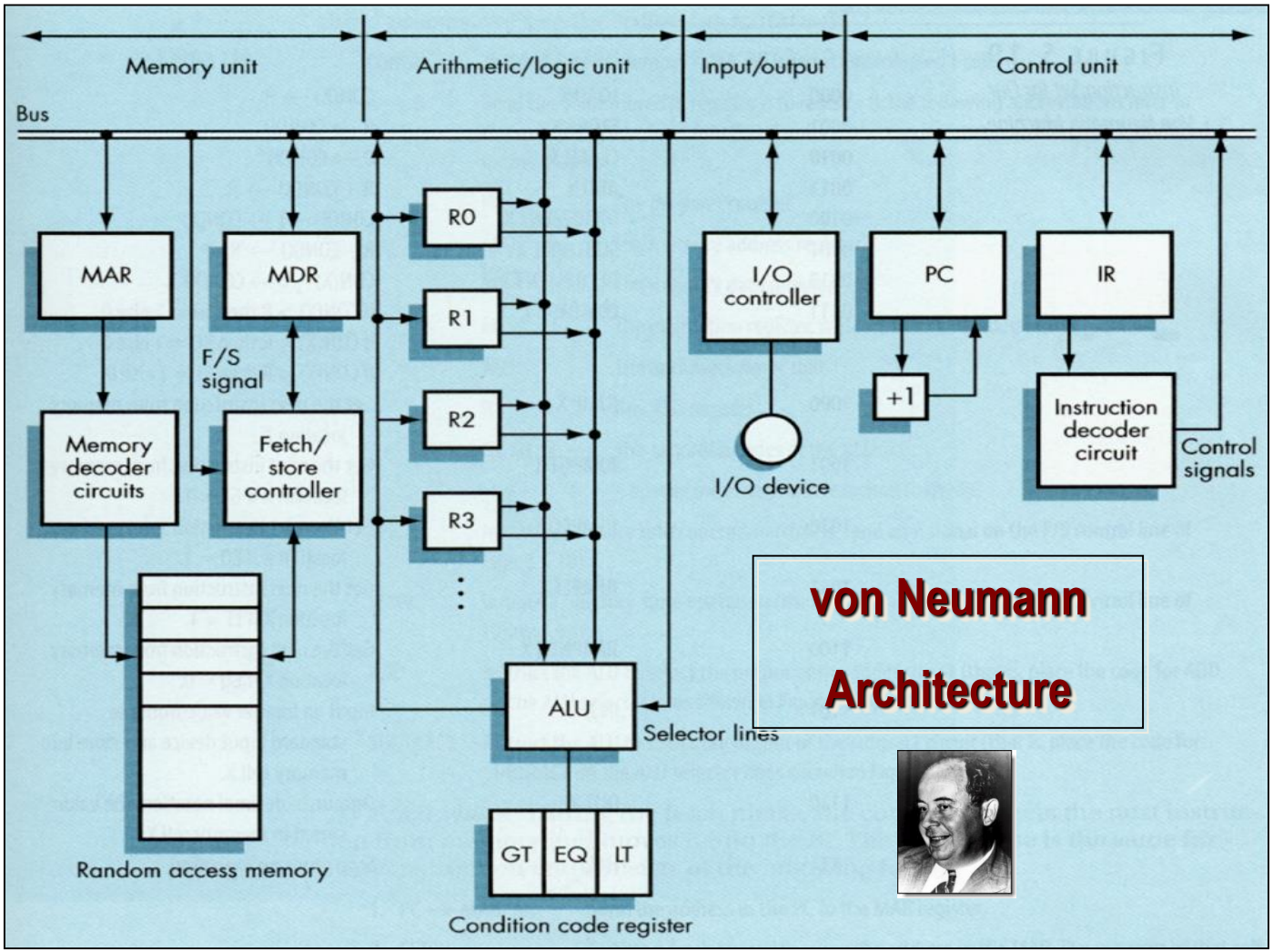
Example

- Pseudo-code: Set A to B + C
- Assuming variable:
  - A stored in memory cell 100, B stored in memory cell 150, C stored in memory cell 151
- Machine language (really in binary)
  - LOAD 150
  - ADD 151
  - STORE 100
  - or
  - (ADD 150, 151, 100)

Structure of the Control Unit

- PC (Program Counter):
  - stores the address of next instruction to fetch
- IR (Instruction Register):
  - stores the instruction fetched from memory
- Instruction Decoder:
  - Decodes instruction and activates necessary circuitry




Memory unit
Arithmetic/logic unit
Input/output
Control unit
Bus
MAR
MDR
F/S signal
Memory decoder circuits
Fetch/ store controller
R0
R1
R2
R3
I/O controller
I/O device
PC
+1
IR
Instruction decoder circuit
Control signals
ALU
Selector lines
GT
EQ
LT
Condition code register
Random access memory
von Neumann Architecture

How does this all work together?

- Program Execution:
  - PC is set to the address where the first program instruction is stored in memory.
  - Repeat until HALT instruction or fatal error

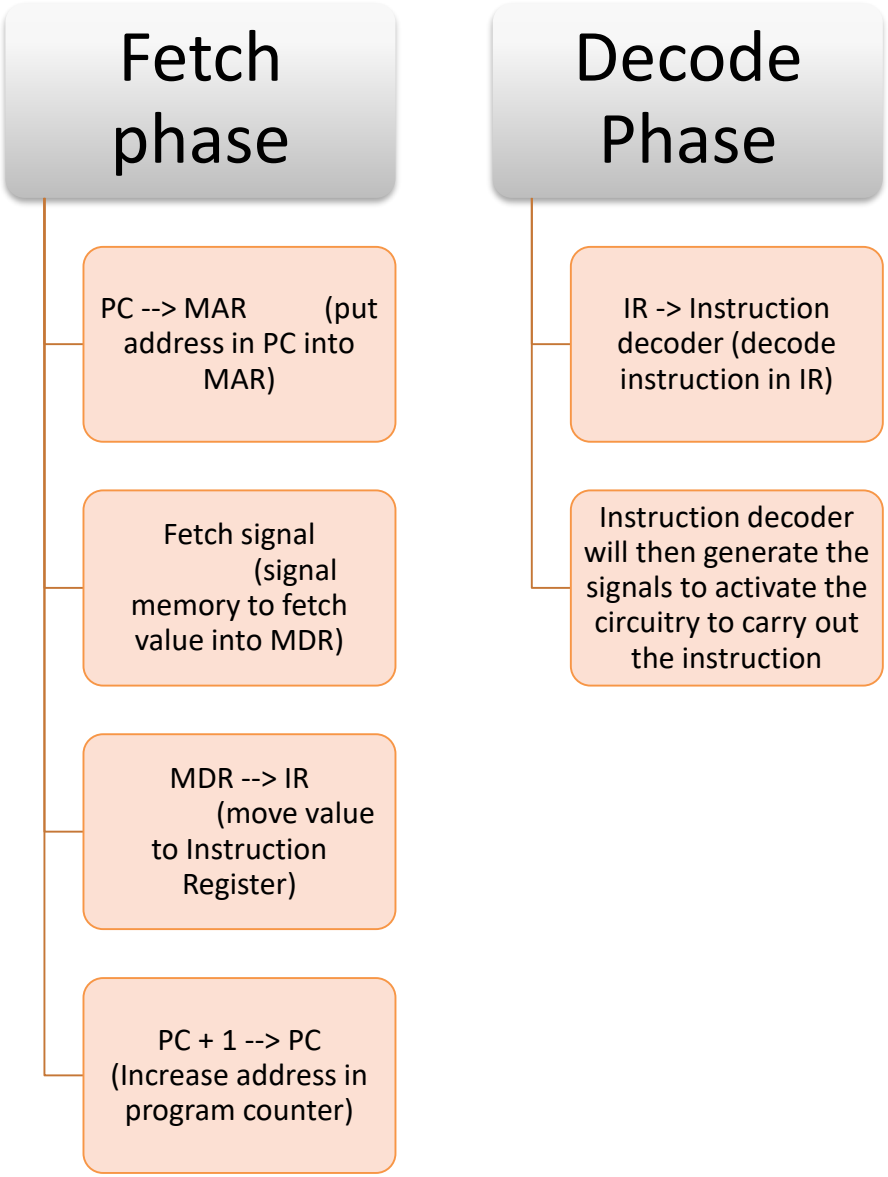
    Fetch instruction

    Decode instruction

    Execute instruction

    End of loop

Program Execution (cont.)



Program Execution (cont.)

Execute Phase

Differs from one instruction to the next.

Example:

LOAD X (load value in addr. X into register)

- IR_address -> MAR
- Fetch signal
- MDR --> R

ADD X

- left as an exercise

Instruction Set for Our Von Neumann Machine

Opcode	Operation	Meaning
0000	LOAD X	CON(X) --> R
0001	STORE X	R --> CON(X)
0010	CLEAR X	0 --> CON(X)
0011	ADD X	R + CON(X) --> R
0100	INCREMENT X	CON(X) + 1 --> CON(X)
0101	SUBTRACT X	R - CON(X) --> R
0101	DECREMENT X	CON(X) - 1 --> CON(X)
0111	COMPARE X	If CON(X) > R then GT = 1 else 0 If CON(X) = R then EQ = 1 else 0 If CON(X) < R then LT = 1 else 0
1000	JUMP X	Get next instruction from memory location X
1001	JUMPGT X	Get next instruction from memory loc. X if GT=1
...	JUMPxx X	xx = LT / EQ / NEQ
1101	IN X	Input an integer value and store in X
1110	OUT X	Output, in decimal notation, content of mem. loc. X
1111	HALT	Stop program execution

What is a Microcontroller

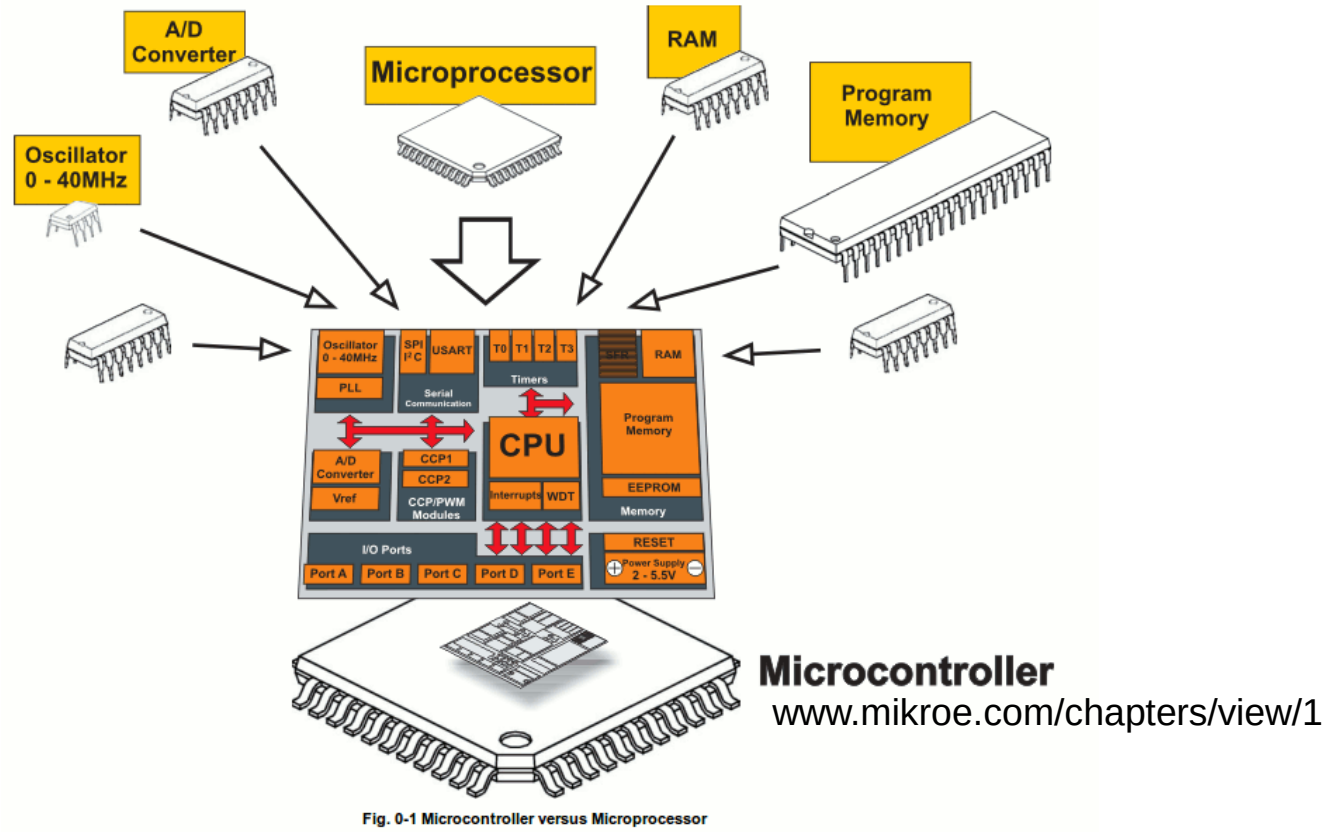
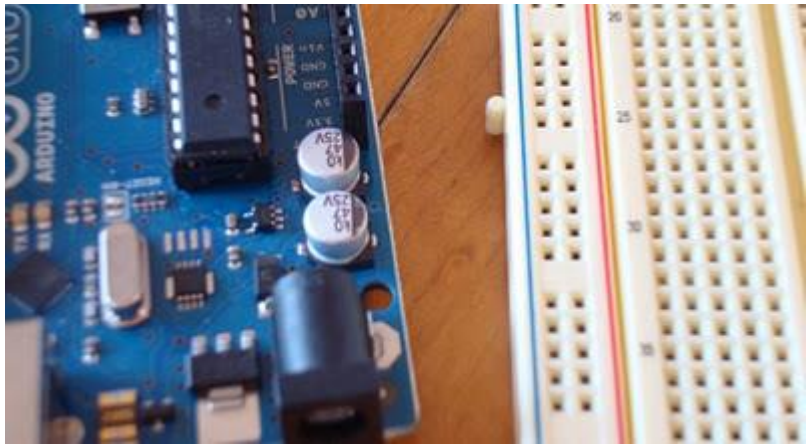


Fig. 0-1 Microcontroller versus Microprocessor

- A small computer on a single chip
  - containing a processor, memory, and input/output
- Typically "**embedded**" inside some device that they control
- A microcontroller is often small and low cost

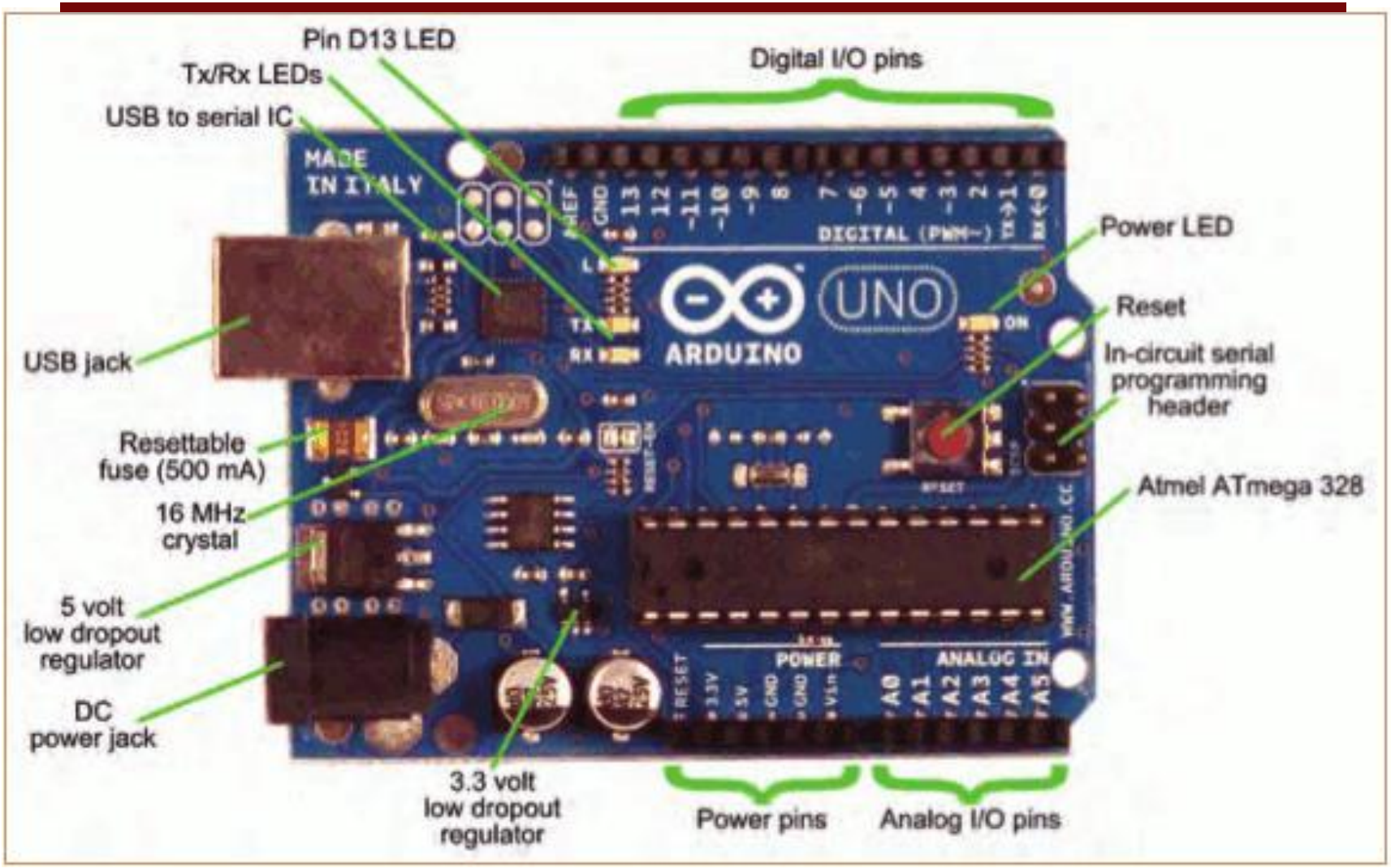
What is a Development Board



- A printed circuit board designed to facilitate work with a particular microcontroller.

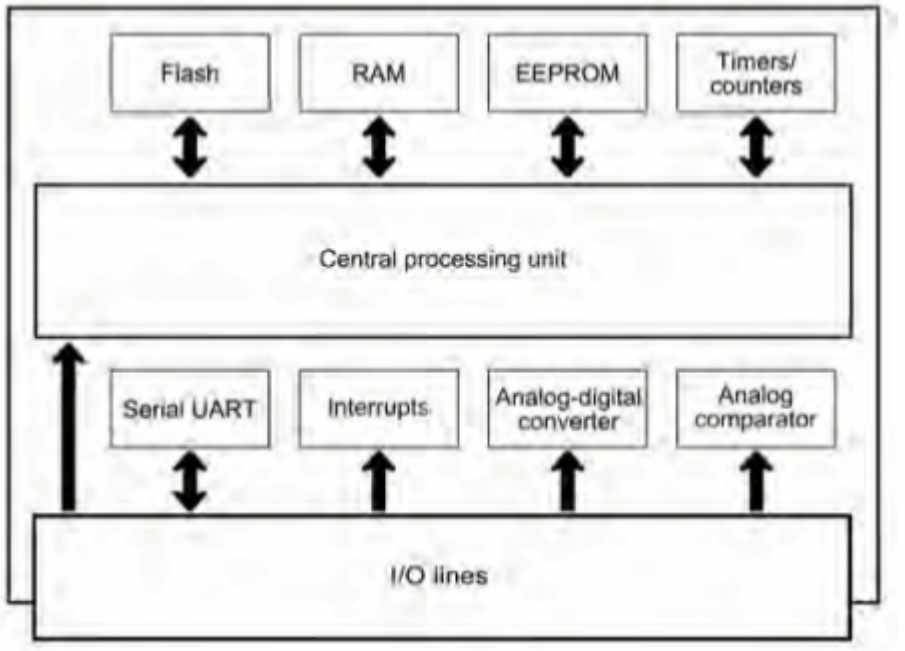
- Typical components include:
  - power circuit
  - programming interface
  - basic input; usually buttons and LEDs
  - I/O pins

The Arduino Development Board



Making-robots-with-arduino.pdf

The Arduino Microcontroller: Atmel ARV Atmega 328



Specification

What is the Arduino

The word "Arduino" can mean 3 things

A physical piece of hardware

A programming environment

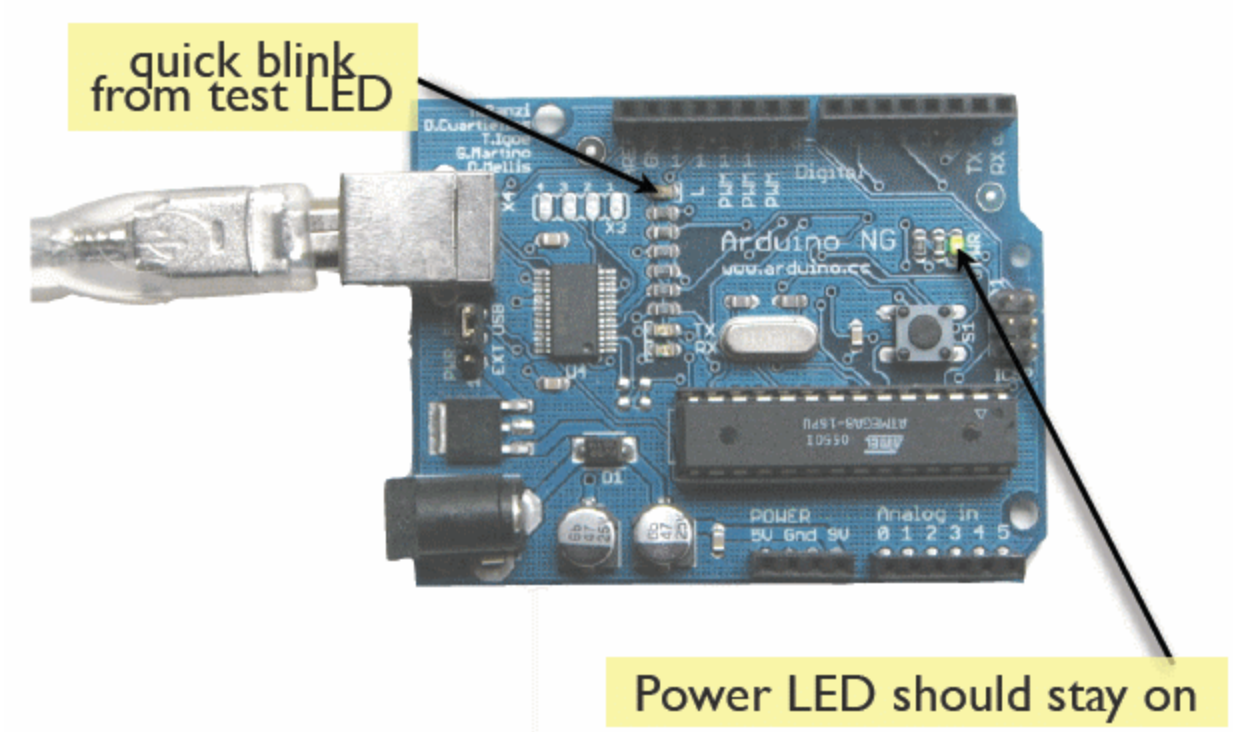
A community & philosophy

todbot.com/blog/bionicarduino

Getting Started

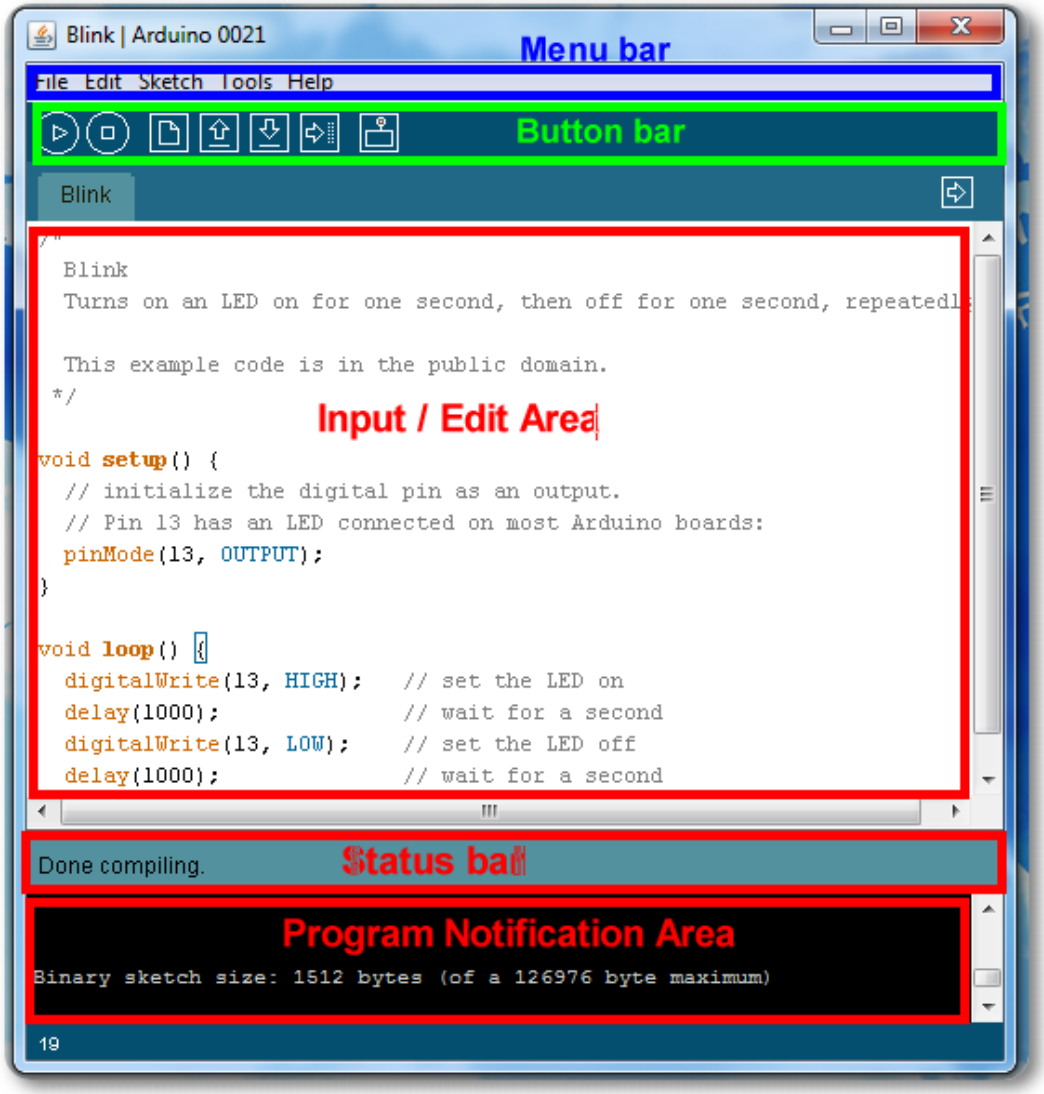
- Check out: http://arduino.cc/en/Guide/HomePage
  1. **Download & install the Arduino environment (IDE)**
  2. **Connect the board to your computer via the UBS cable**
  3. **If needed, install the drivers (not needed in lab)**
  4. **Launch the Arduino IDE**
  5. **Select your board**
  6. **Select your serial port**
  7. **Open the blink example**
  8. **Upload the program**

Try It: Connect the USB Cable



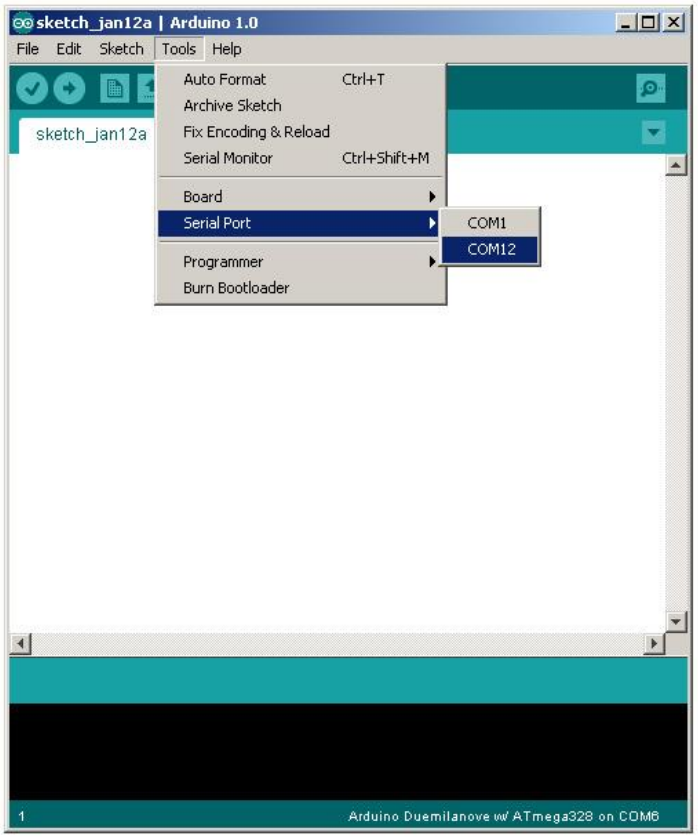
todbot.com/blog/bionicarduino

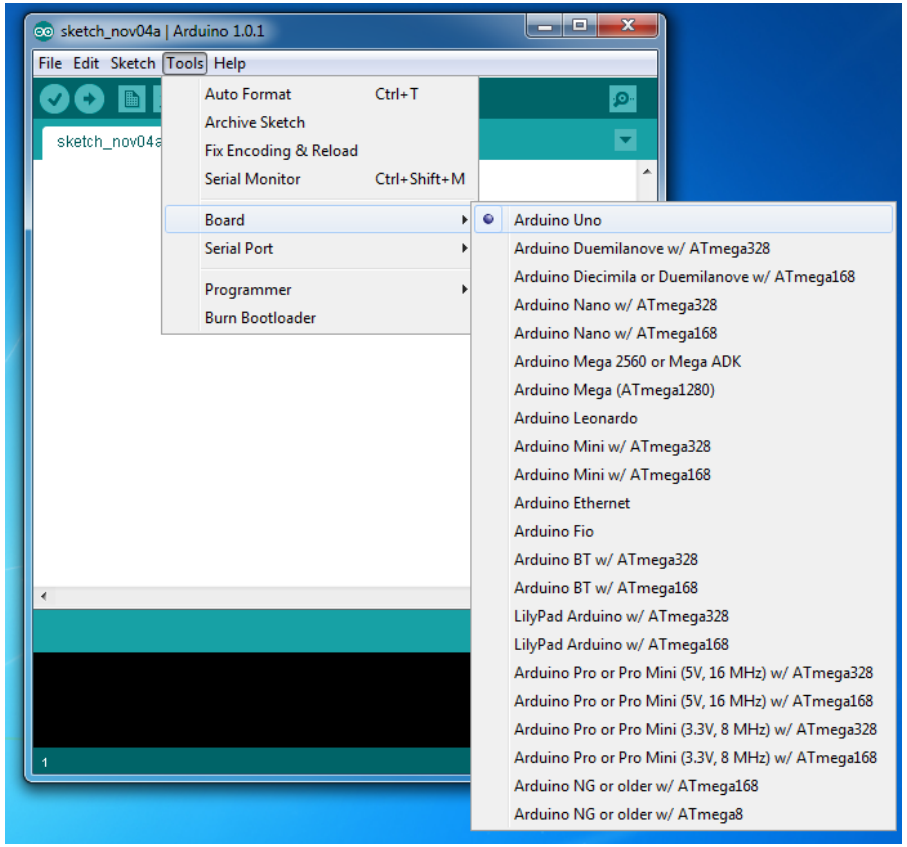
Arduino IDE



See: http://arduino.cc/en/Guide/Environment for more information

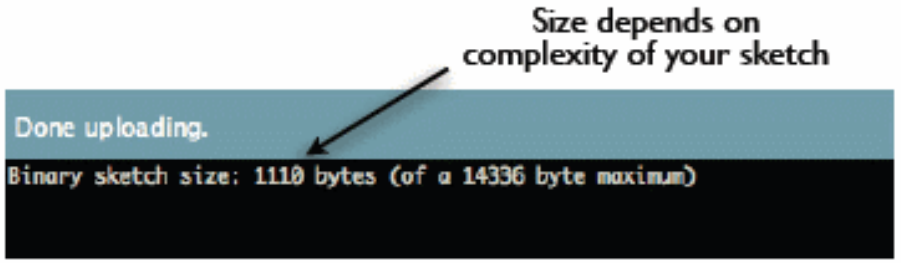
Select Serial Port and Board





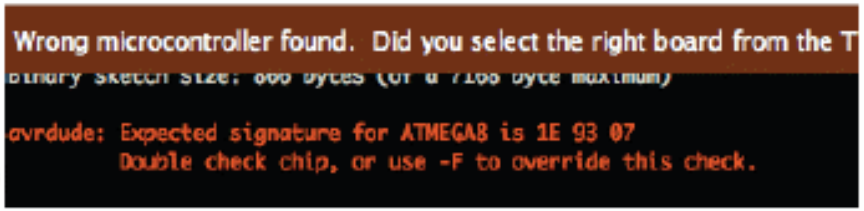
Status Messages

Uploading worked



Wrong serial port selected

Wrong board selected



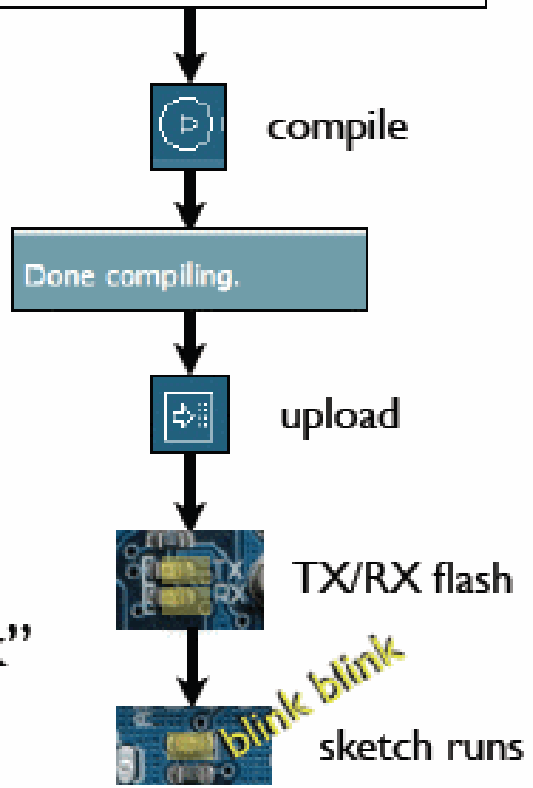
nerdy cryptic error messages

todbot.com/blog/bionicarduino

Using Arduino

- Write your sketch
- Press Compile button (to check for errors)
- Press Upload button to program Arduino board with your sketch

```
void setup() {
  pinMode(ledPin, OUTPUT);      // sets t
}
void loop() {
  digitalWrite(ledPin, HIGH);   // sets t
  delay(1000);                  // waits
  digitalWrite(ledPin, LOW);    // sets t
  delay(1000);                  // waits
}
```



Try it out with the "Blink" sketch!

Load "File/Sketchbook/Examples/Digital/Blink"

Data Representation

Typically computers store and transmit information digitally as a collection of bits.

Each bit has two states:

- on/off
- true/false
- high/low
- 1/0

A byte is a collection of 8 bits (usually)

- An 8-bit byte can encode 256 different states

Data Representation

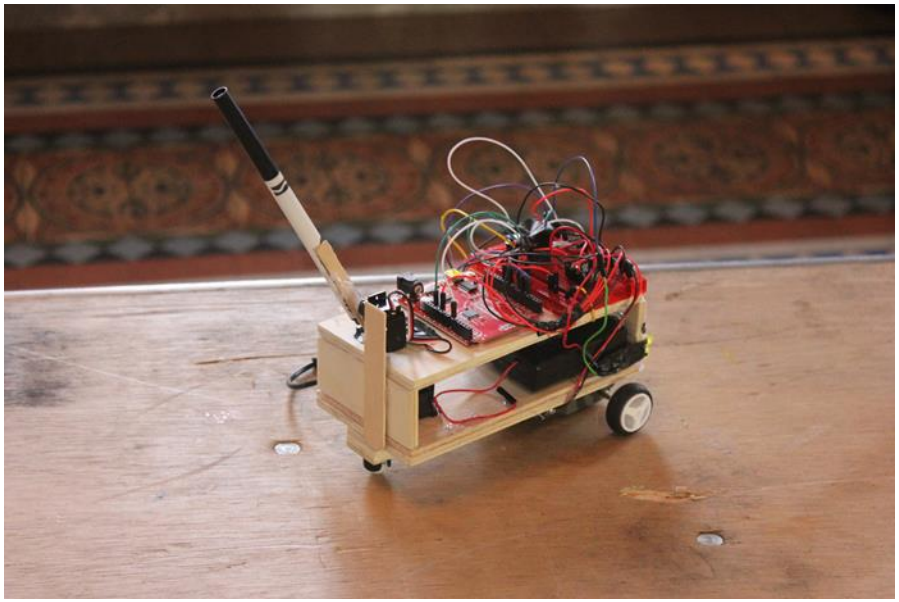
Information is coded using a predefined method

Consider the coding for three bits

Eight district values

Many options for encoding

000
001
010
011
100
101
110
111



Data Representation

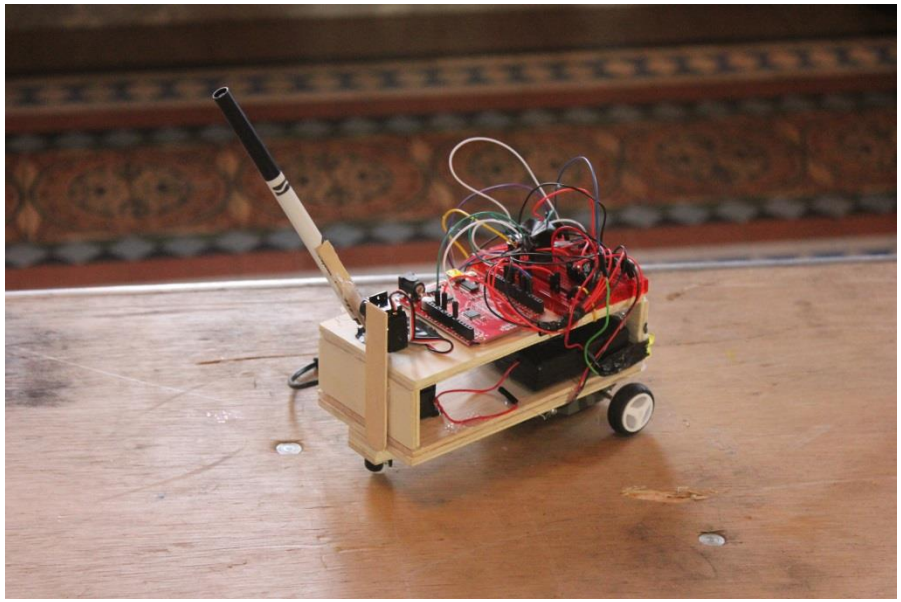
Information is coded using a predefined method

Consider the coding for three bits

Eight district values

Many options for encoding

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7



Data Representation

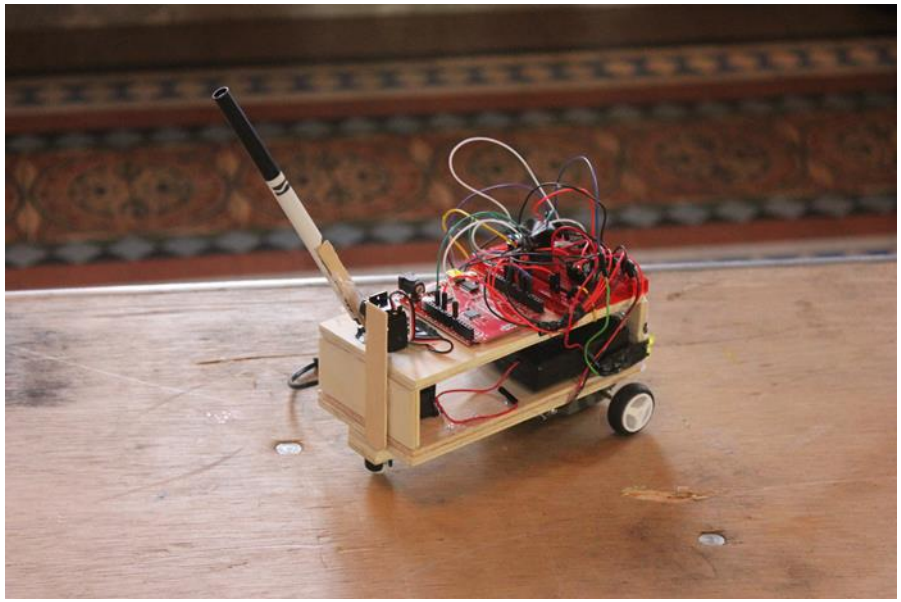
Information is coded using a predefined method

Consider the coding for three bits

Eight district values

Many options for encoding

000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1



Data Representation

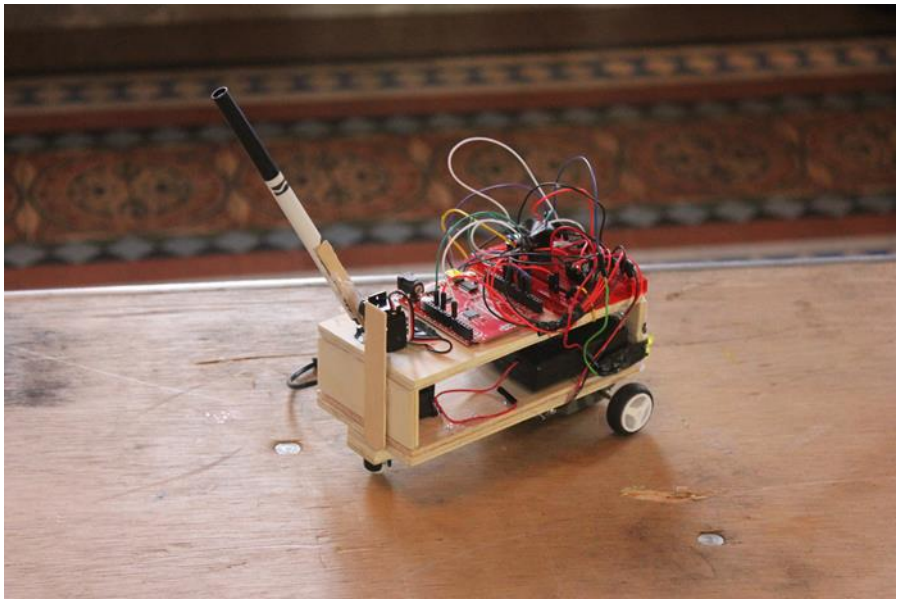
Information is coded using a predefined method

Consider the coding for three bits

Eight district values

Many options for encoding

000	0	0	A
001	1	1	B
010	2	2	C
011	3	3	D
100	4	-4	F
101	5	-3	P
110	6	-2	W
111	7	-1	I



boolean	single bit FALSE/TRUE	
	(really uses a whole byte of memory)	
Unsigned Integers		
byte	eight bits	0 to 255
word	two bytes	0 to 65535
unsigned long	4 bytes	0 to 4,294,967,295
Signed Integers		
char	eight bits	-128 to 127
short	two bytes	-32,768 to 32,767
long	4 bytes	-2,147,483,648 to 2,147,483,647.

Arduino Basics *Data Types*

int Unsigned integer, the number of bytes used depends on the particular hardware used. For us int is the same as short.

Real Numbers

float Floating point number uses 4 bytes. Used for non-integers has 6-7 decimal point precision.

-3.4028235E+38 to 3.4028235E+38

Arduino Basics *Declaring/Initializing Variables*

Before using a variable it must be declared.

```
int a;  // creates an integer with the name ‘a’
```

When a variable is declared it can also be initialized.

```
int a=34;     // creates an integer with the name ‘a’
              //and assigns it the value 34.
```

The char data type is used to represent characters using ASCII encoding.

Single character constants are indicated with single quotes.

```
char A;
A='B';
```

'B' is encoded with ASCII and the resulting value of 66 is stored in A.

What does ‘A’ + ‘!’ equal?

A single variable can store an array of values.

The index is contained in square brackets.
Arrays are zero indexed (start at zero).

```
int threeints[3];   // ‘threeints’ is an array (0-2)
threeints[0]=15;
threeints[1]=10;
threeints[2]=threeints[0]-threeints[1];
```

An array of characters is called a string

char examplestring[8];

examplestring="arduino";

The last element of a sting is always the 'null string' which has an ASCII value of zero

Stings can also be stored as objects using the String class.

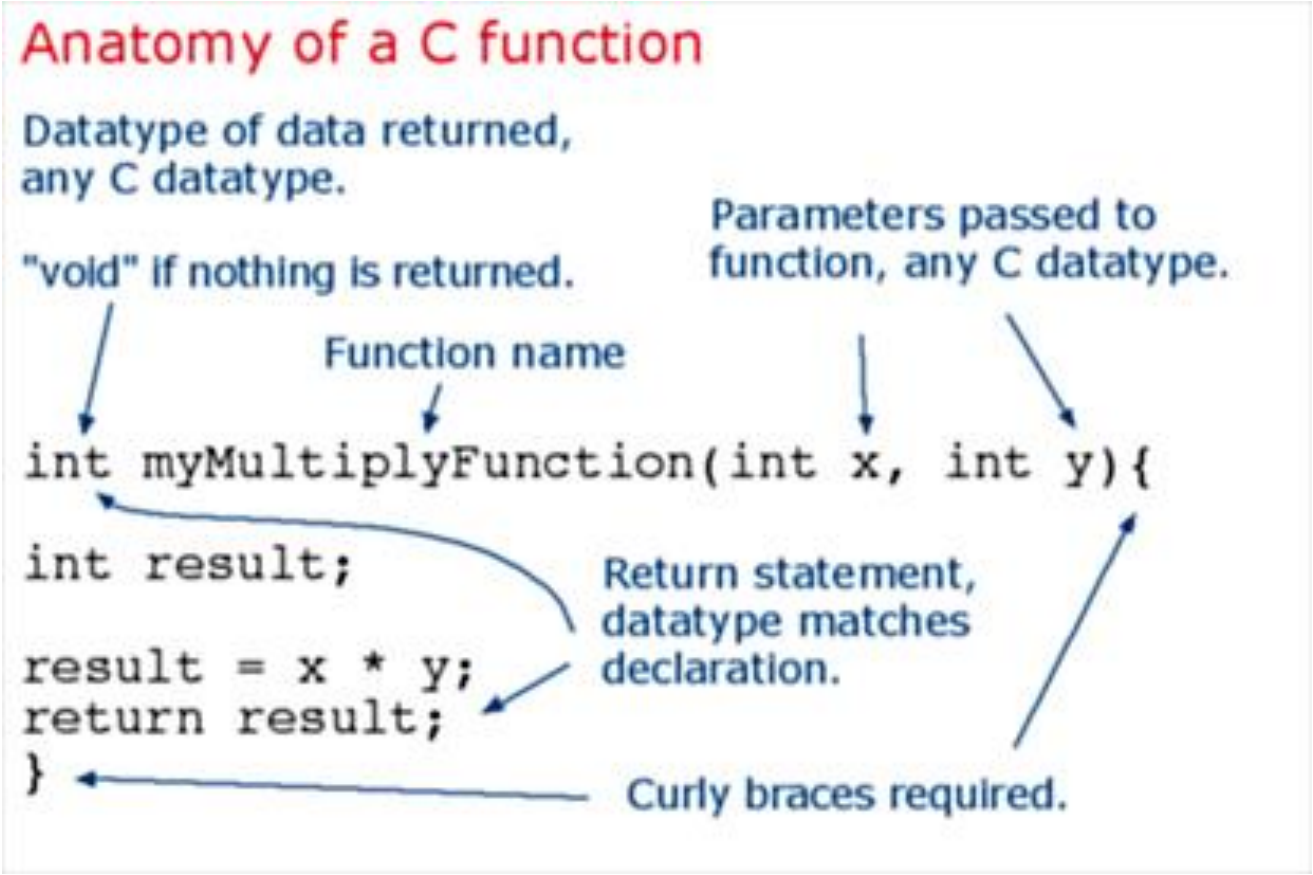
Using String objects rather than character arrays uses more memory but adds functionality.

Character arrays are referred to as strings with a small s, and instances of the String class are referred to as Strings with a capital S.

Constant strings, specified in "double quotes" are treated as char arrays, not instances of the String class.

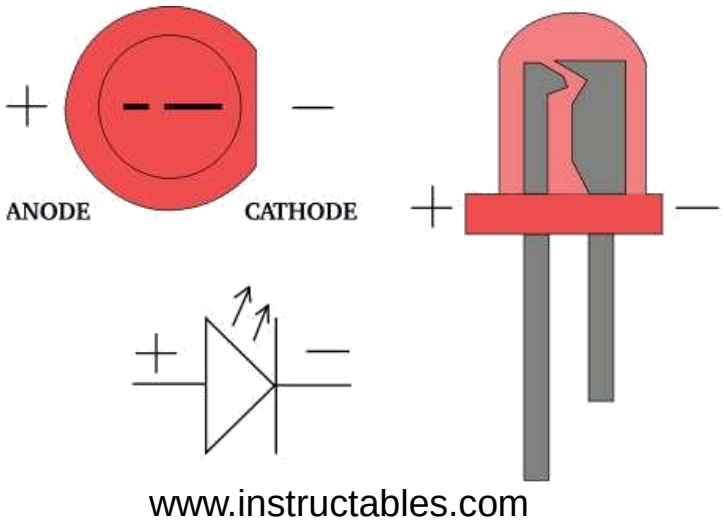
Arduino Basics *Functions*

Creating functions allows for modular program design which speeds up the programming process and makes it easy to debug and/or modify the functionality of a program

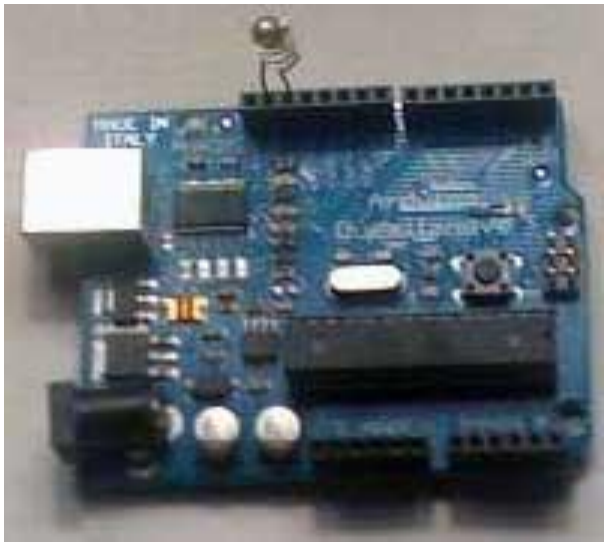


Add an External LED to pin 13

- **File > Examples > Digital > Blink**
- LED's have polarity
  - Negative indicated by flat side of the housing and a short leg



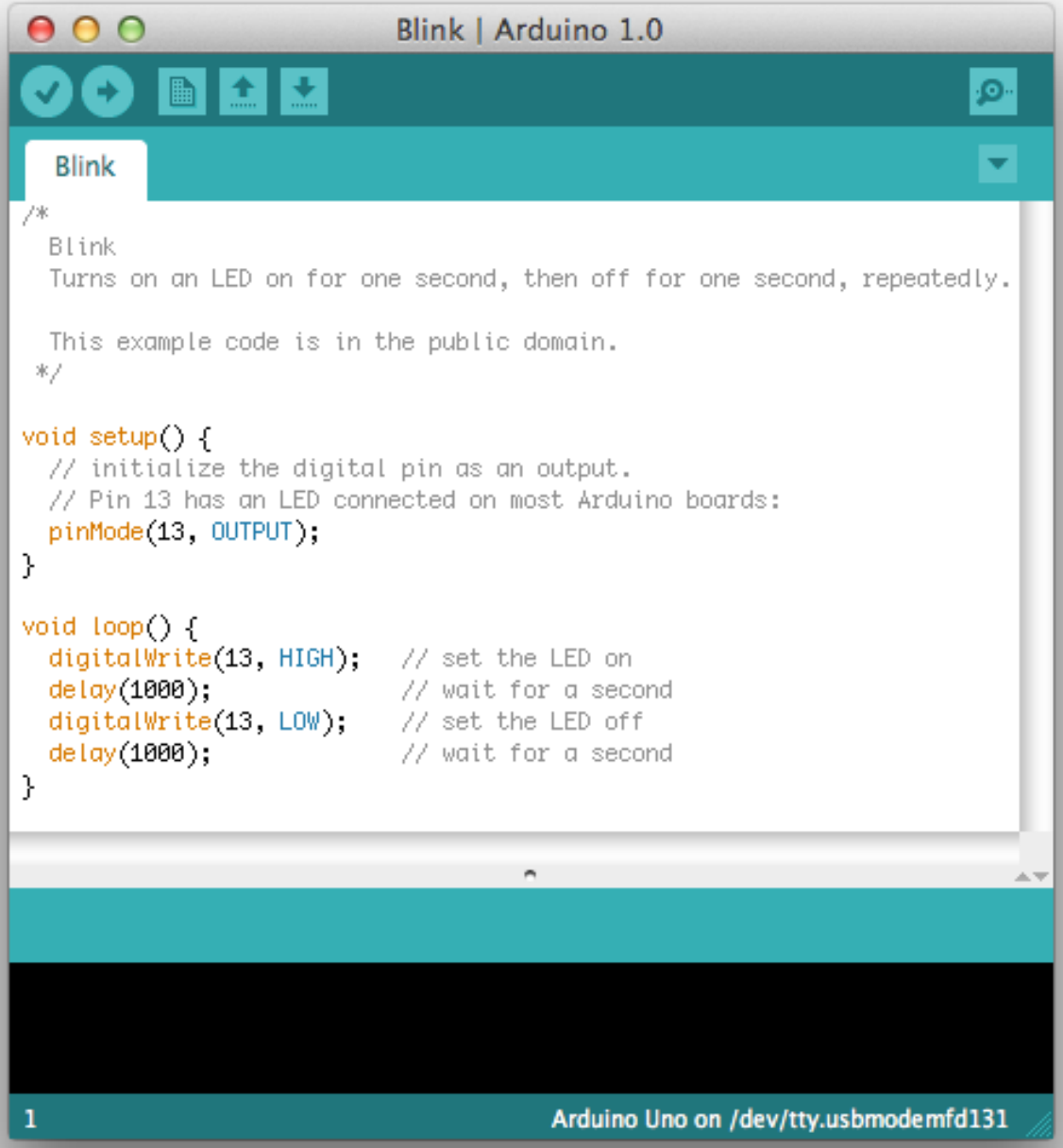
www.instructables.com



A Little Bit About Programming

- Code is case sensitive
- Statements are commands and must end with a semi-colon
- Comments follow a // or begin with /* and end with */
- loop and setup

Our First Program



Terminology

"*sketch*" – a program you write to run on an Arduino board

"*pin*" – an input or output connected to something.

e.g. output to an LED, input from a knob.

"*digital*" – value is either HIGH or LOW.

(aka on/off, one/zero) e.g. switch state

"*analog*" – value ranges, usually from 0-255.

e.g. LED brightness, motor speed, etc.

Setup()

Description

La fonction setup() est appelée au démarrage du programme. Cette fonction est utilisée pour initialiser les variables, le sens des broches, les librairies utilisées. La fonction setup n'est exécutée qu'une seule fois, après chaque mise sous tension ou reset (réinitialisation) de la carte Arduino.

Syntaxe

```
void setup() { }
```

Exemple

```
int buttonPin = 3; // déclaration d'une variable globale
void setup() // fonction setup - début de l'exécution du programme {
Serial.begin(9600); pinMode(buttonPin, INPUT); }
void loop() // fonction loop - est exécutée en boucle // une fois que la fonction setup a été exécutée { // ... }
```

Commentaire

La fonction setup() , même vide, est obligatoire dans tout programme Arduino
Pour comprendre : les habitués du C seront surpris de ne pas trouver la classique fonction main() obligatoire dans tout programme C. En fait, la fonction setup() et la fonction loop()sont implémentées au sein de la fonction main() (Voir dans le répertoire arduino le fichier \hardware\cores\arduino\main.cxx) qui est appelée en premier lors de l'exécution de tout programme en C, langage sur lequel est basé le langage Arduino.

loop()

Description

Après avoir créé une fonction setup(), qui initialise et fixe les valeurs de démarrage du programme, la fonction loop () (boucle en anglais) fait exactement ce que son nom suggère et s'exécute en boucle sans fin, permettant à votre programme de s'exécuter et de répondre. Utiliser cette fonction pour contrôler activement la carte Arduino.

Syntaxe

void loop() { }

Exemple

int buttonPin = 3; // la fonction setup initialise la communication série // et une broche utilisée avec un bouton poussoir

void setup() { beginSerial(9600); pinMode(buttonPin, INPUT); }
// la fonction loop teste l'état du bouton à chaque passage // et envoie au PC une lettre H si il est appuyé, L sinon.

void loop() { if (digitalRead(buttonPin) == HIGH)
serialWrite('H’);
else serialWrite('L'); delay(1000); }

Commentaire

La fonction loop() est obligatoire, même vide, dans tout programme.

Arduino Basics

Program Flow Control: Conditional Statements and Branching

- Conditional Statements

```
if (condition){
        //do this if condition is true
}
else{
        //do this if condition is false
}
```

- Condition is often a statement that uses comparison operators

  ==, !=, < ,>

  - Common mistake is to use “=” instead of “==”

Arduino Basics *Program Flow Control: Loops*

- for loop

```
for (initialization; condition; increment){
        //do this until the condition is no longer met
}
```

- Example

```
for(int i=0;i<3;i++){
        serial.println(i);
}
Serial.println(i);
```

Arduino Basics *Variable scope*

```
int hi;                 //global variables
int there=0;            //all functions have access

void setup(){
        int this=10;  //only exists within “setup()”
        hi=1;         //changing value of “hi”
}
void loop(){
        println(“hi”);      //prints “hi”
        println(hi);        //prints “1”
        println(there);     //prints “0”
        println(this);      //variable not defined…error program
                            //won’t compile (verify)
    }
```

Arduino Basics

Example: What is the final value of x?

```
int x=1;
void setup(){
        x=10;


}
void loop(){
        if (x<20){
                for(int i=1;i<3;i++){
                        x=x+x;
                }
        }
}
```

Serial Communication

- First you need to setup serial communication
  - Serial.begin(9600); //typically in setup()
    - 9600 is the default speed but different values can be used.
- Two main functions for printing information on the computer.
  - Serial.print(“Hi”); //Prints the word “Hi” on the screen. The next print //command will result in printing right after the “i”
  - Serial.println(“Hi”); //will pass a linebreak after “Hi”. The next print //command will result in printing on a new line.
- What is printed depends on the data type of the variable being passed to the function.

Serial Communication

Information can be read from the computer (serial monitor)

Bytes are stored in a buffer automatically

- Serial.read(); //reads first byte from buffer
- Serial.available();// returns the number of bytes in the buffer
- Serial.flush();// clears the buffer

Arduino Digital Output

Digital output pins can be HIGH (5V) or LOW (0V).
Before you use a digital output you need to set the “pinMode”

```
//There is a LED connected to pin 13 on the //RedBoard

void setup()
{
  pinMode(13, OUTPUT);      // sets pin 13 as output
}

void loop()
{
  digitalWrite(13, HIGH);   // sets the LED on
  delay(1000);              // waits for a second
  digitalWrite(13, LOW);    // sets the LED off
  delay(1000);              // waits for a second
}
```

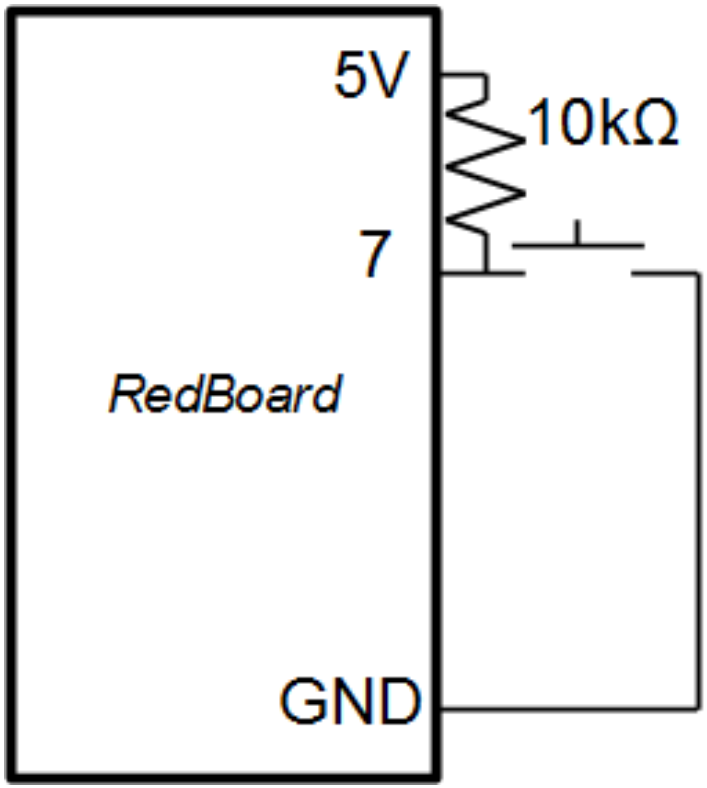
http://arduino.cc/en/Reference/DigitalWrite

Arduino Digital Input

Digital input pins can be HIGH (2.7V-5V) or LOW (1.2V-0V).
Before you use a digital input you need to set the “pinMode”

```
int ledPin = 13; // LED connected to digital pin 13
int inPin = 7;   // pushbutton connected to digital pin 7
boolean val = 0;     // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);
                // sets the digital pin 13 as output
  pinMode(inPin, INPUT);
                // sets the digital pin 7 as input
}

void loop()
{
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);
                // sets the LED to the button's value
}
```

Arduino Digital Input

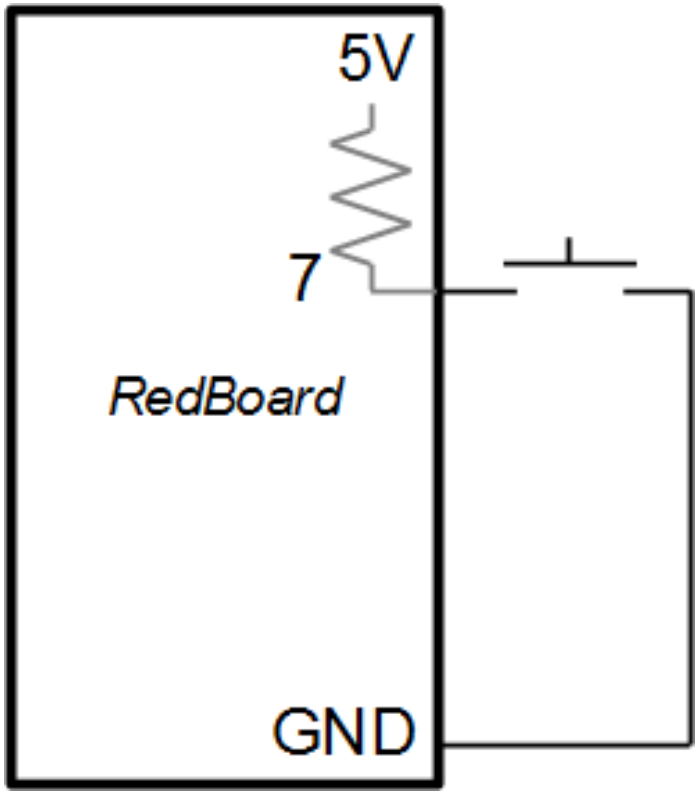



```
int ledPin = 13; // LED connected to digital pin 13
int inPin = 7;   // pushbutton connected to digital pin 7
boolean val = 0;     // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);
                // sets the digital pin 13 as output
  pinMode(inPin, INPUT);
                // sets the digital pin 7 as input
}

void loop()
{
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);
                // sets the LED to the button's value
}
```

Arduino Digital Input

Pins Configured as INPUT_PULLUP
The Atmega chip on the Arduino has internal pull-up resistors (resistors that connect to power internally) that you can access. If you prefer to use these instead of external pull-down resistors, you can use the **INPUT_PULLUP** argument in pinMode().

http://arduino.cc/en/Reference/Constants



Arduino Analog Input

Analog Signal– Value is continuous not discretized

Analog Inputs allow us to read analog voltage signals
…but first we have to discretize them.

The RedBoard (and UNO) have a 10-bit analog to digital converter A/D

Voltages from 0V to 5V are scaled to the integers from 0 to 1023

Arduino Analog Input

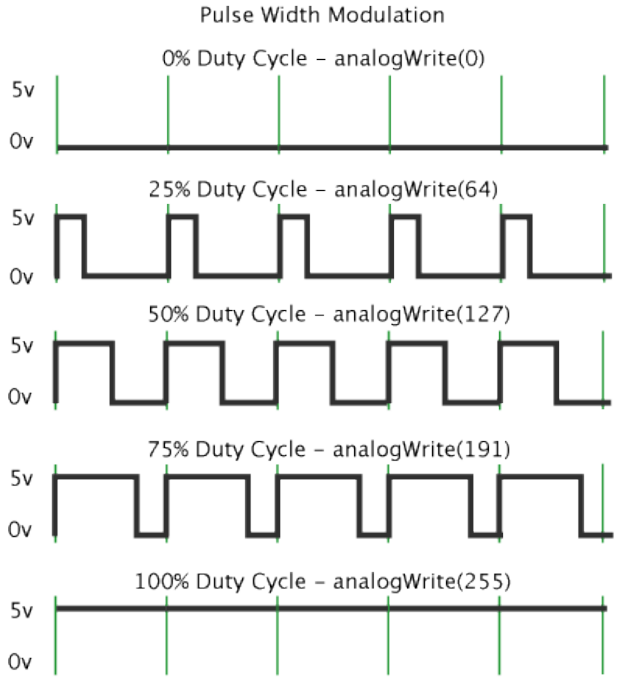
```
int analogPin = 3; // potentiometer wiper (middle terminal)
                           // connected to analog pin 3
                         // outside leads to ground and +5V
int val = 0;          // variable to store the value read

void setup()
{
  Serial.begin(9600);          //  setup serial
}

void loop()
{
  val = analogRead(analogPin);    // read the input pin
  Serial.println(val);                    // debug value
}
```

http://arduino.cc/en/Reference/AnalogRead

Arduino Output

The RedBoard (like other Arduino boards) does not have a true analog output. Instead it can create a pulse width modulation (PWM) signal on designated pins

PWM Turns on and off rapidly. Duty cycle of the pulse can be adjusted, which controls the average power output of the pin (dimmer switches work the same way)



Pins marked with ~ can be used as analog output pins

The ‘analogWrite’ function takes two inputs: The pin number and a value between 0 and 255 (8-bit)

http://arduino.cc/en/Tutorial/PWM

Arduino Output

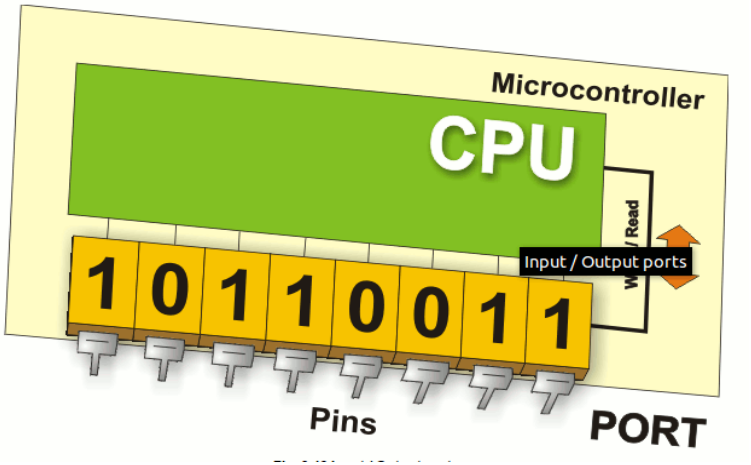
```
int ledPin = 9;      // LED connected to digital pin 9
int analogPin = 3;   // potentiometer connected to analog pin 3
int val = 0;         // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);   // sets the pin as output
}

void loop()
{
  val = analogRead(analogPin);   // read the input pin
  analogWrite(ledPin, val / 4);
}
```

http://arduino.cc/en/Reference/AnalogWrite

Arduino Output

```
int ledPin = 9;      // LED connected to digital pin 9
int analogPin = 3;   // potentiometer connected to analog pin 3
int val = 0;         // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);   // sets the pin as output
}

void loop()
{
  val = analogRead(analogPin);   // read the input pin
  analogWrite(ledPin, val / 4);  // analogRead values go from 0 to 1023,
analogWrite values from 0 to 255
}
```

http://arduino.cc/en/Reference/AnalogWrite



Digital I/O

`pinMode(`*`pin, mode`*`)`

Sets pin to either `INPUT` or `OUTPUT`

`digitalRead(`*`pin`*`)`

Reads `HIGH` or `LOW` from a pin

`digitalWrite(`*`pin, value`*`)`

Writes `HIGH` or `LOW` to a pin

Electronic stuff

Output pins can provide 40 mA of current

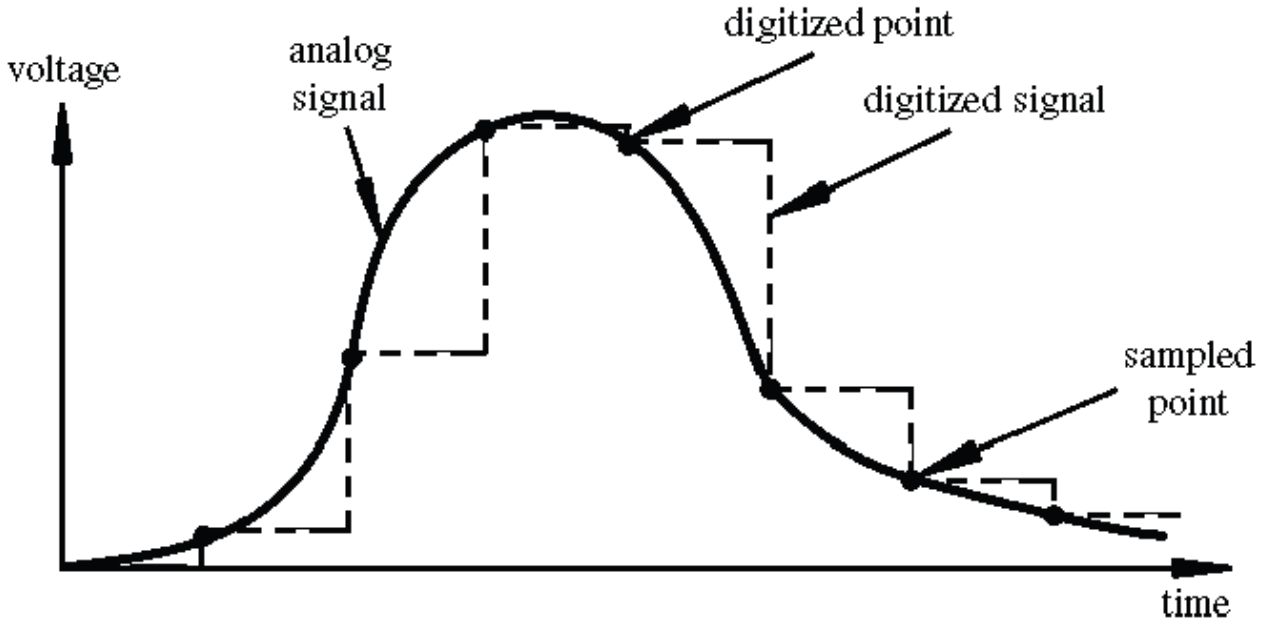
Writing `HIGH` to an input pin installs a 20KΩ pullup

Arduino Timing

- `delay(`*`ms`*`)`
  - Pauses for a few milliseconds
- `delayMicroseconds(`*`us`*`)`
  - Pauses for a few microseconds
- More commands: arduino.cc/en/Reference/HomePage

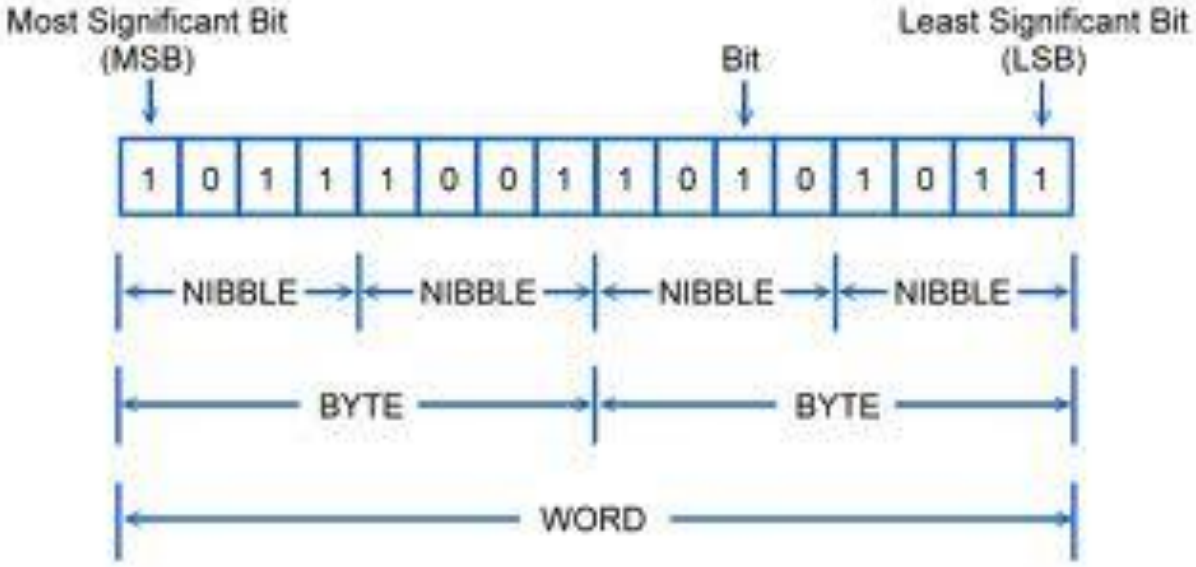
Digital? Analog?

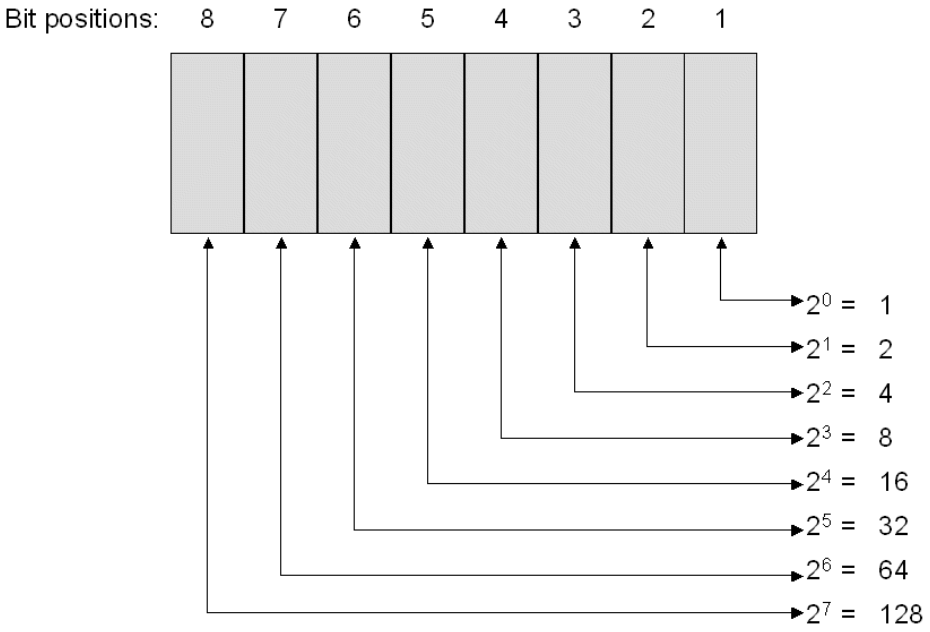
- Digital has two values: **on** and **off**
- Analog has many (infinite) values
- Computers don't really do analog, they ***quantize***
- Remember the 6 analog input pins---here's how they work



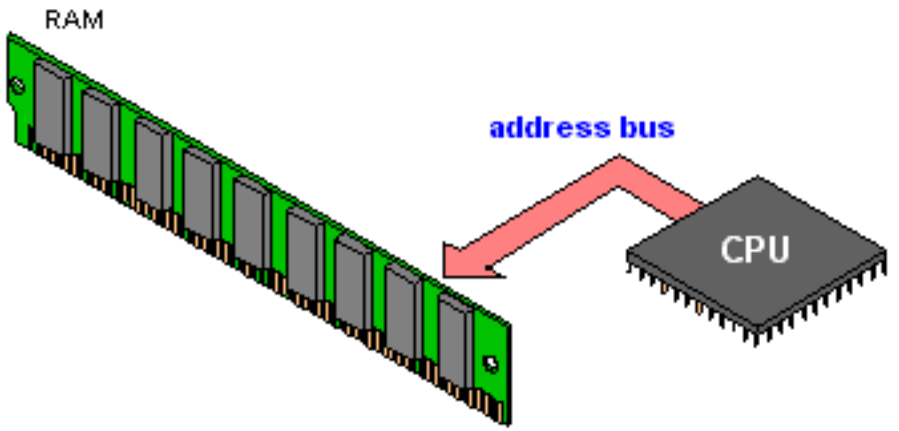
todbot.com/blog/bionicarduino

Bits and Bytes





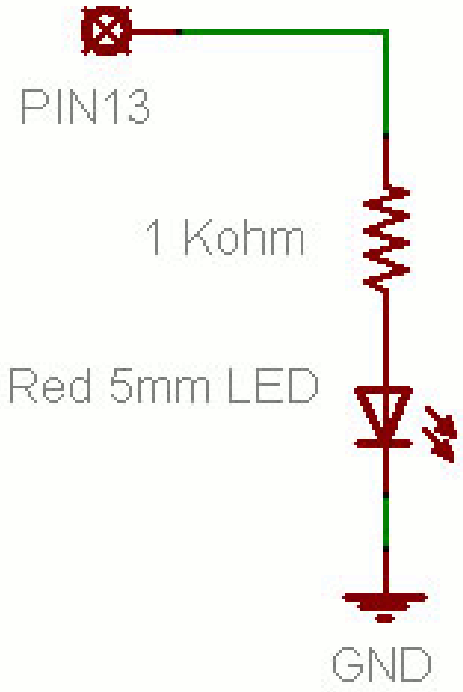
From Computer Desktop Encyclopedia



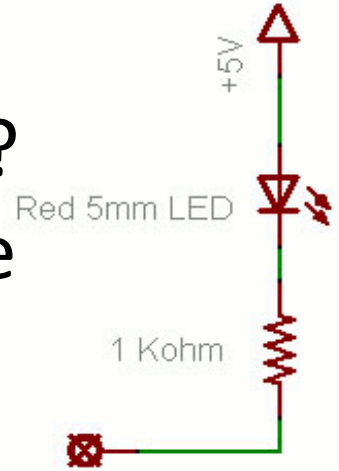
Variables

www3.ntu.edu.sg

Putting It Together



- Complete the sketch (program) below.
- What output will be generated by this program?
- What if the schematic were changed? ➔




```
void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(500);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(500);                  // waits for a second
}
```

Good References

www.arduino.cc

www.EarthshineElectronics.com