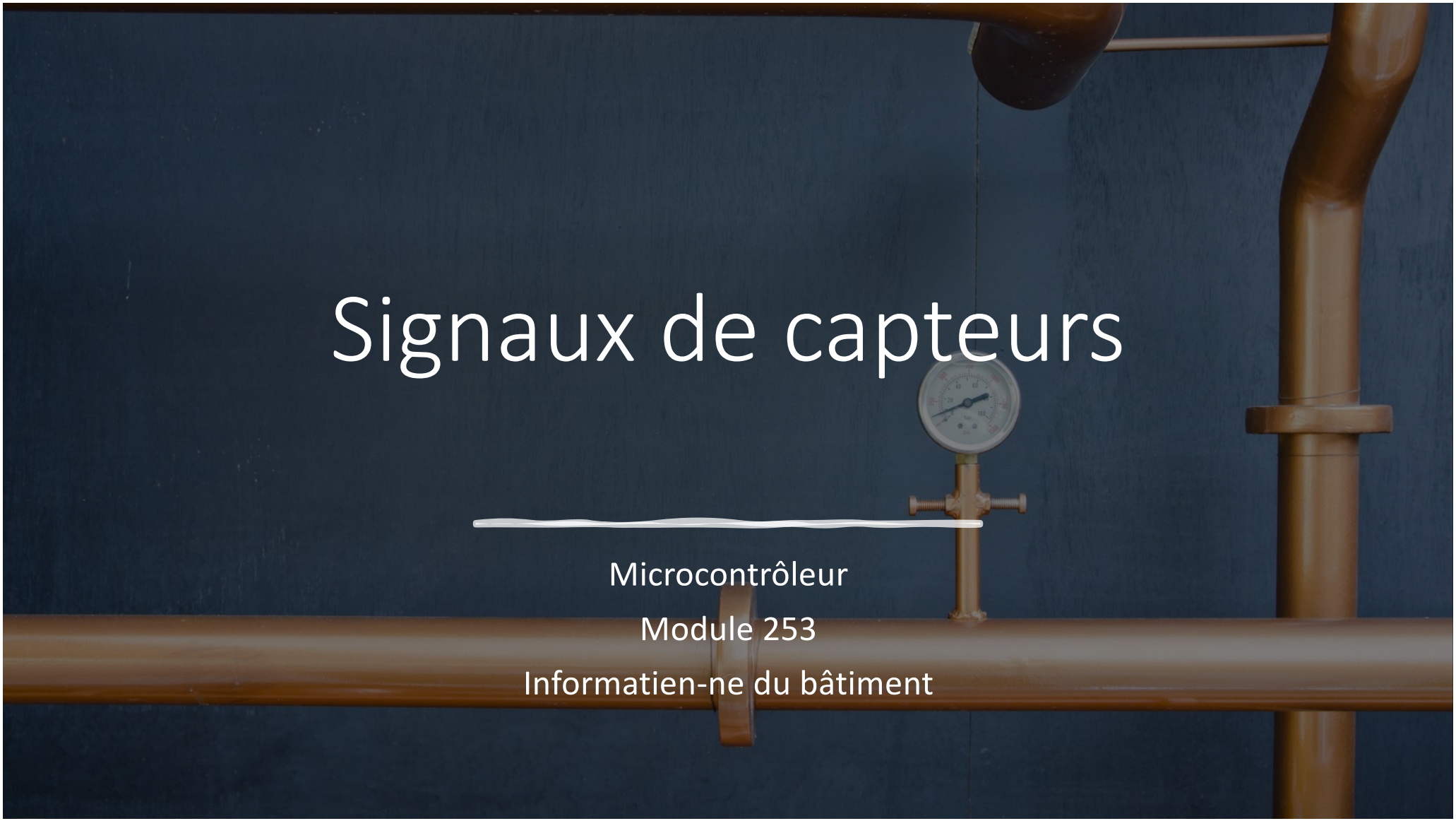


Signaux de capteurs

Microcontrôleur

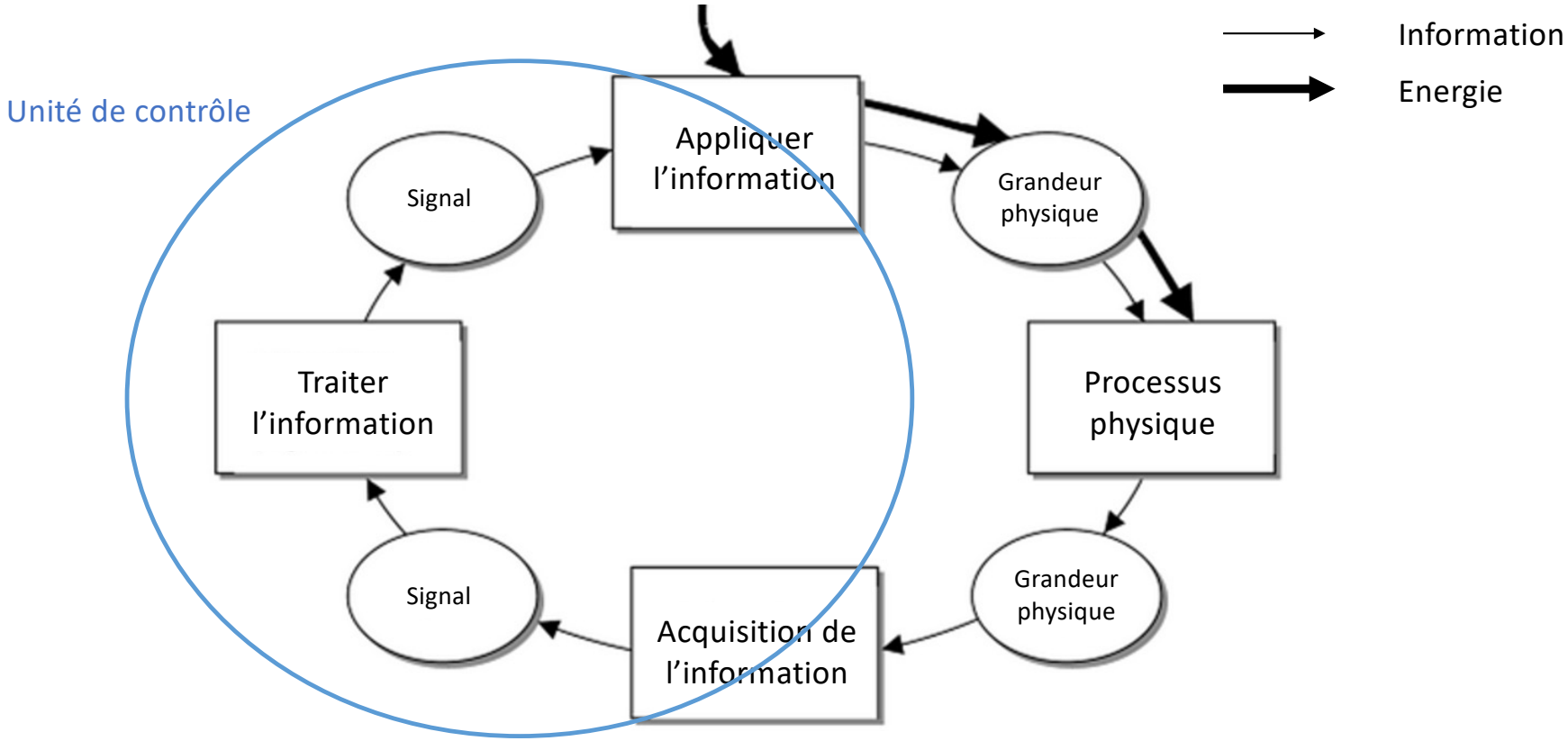
Module 253

Informatien-ne du bâtiment

L’unité de contrôle / traitement

- L’unité de contrôle et de traitement de l’information, un pont entre :
  - La **mesure** et l’**action**
    ou, en terme d’équipements, entre :
  - Le(s) **capteur**(s) et le(s) **actionneur**(s)

Principes de base de l'automatique



Mesure – Traitement – Action

Mesure	Traitement	Action
Les capteurs écoutent le monde physique	Les micro-contrôleurs **écoutent** les capteurs et **parlent** aux actionneurs	**Les actionneurs agissent dans le monde physique**
Les capteurs convertissent l'énergie physique en signaux électriques	Les µC décident quoi faire selon le programme que vous avez écrit	Les actionneurs convertissent l'énergie électrique en énergie physique

Mesure – Traitement – Action

Mesure	Traitement	Action
Capteur	Micro-contrôleur	Actuateur, actionneur
Température Lumière Mouvement Pression atmosphérique Humidité Gas (CO_2, NO_X, ...) Clavier etc …	Arduino ESP32 ARM NXP STM …	Moteur, pompe Ventilation Corps de chauffe Haut-parleur Ecran, projecteur Lampe Vibreur Relais etc …

Les unités de contrôle

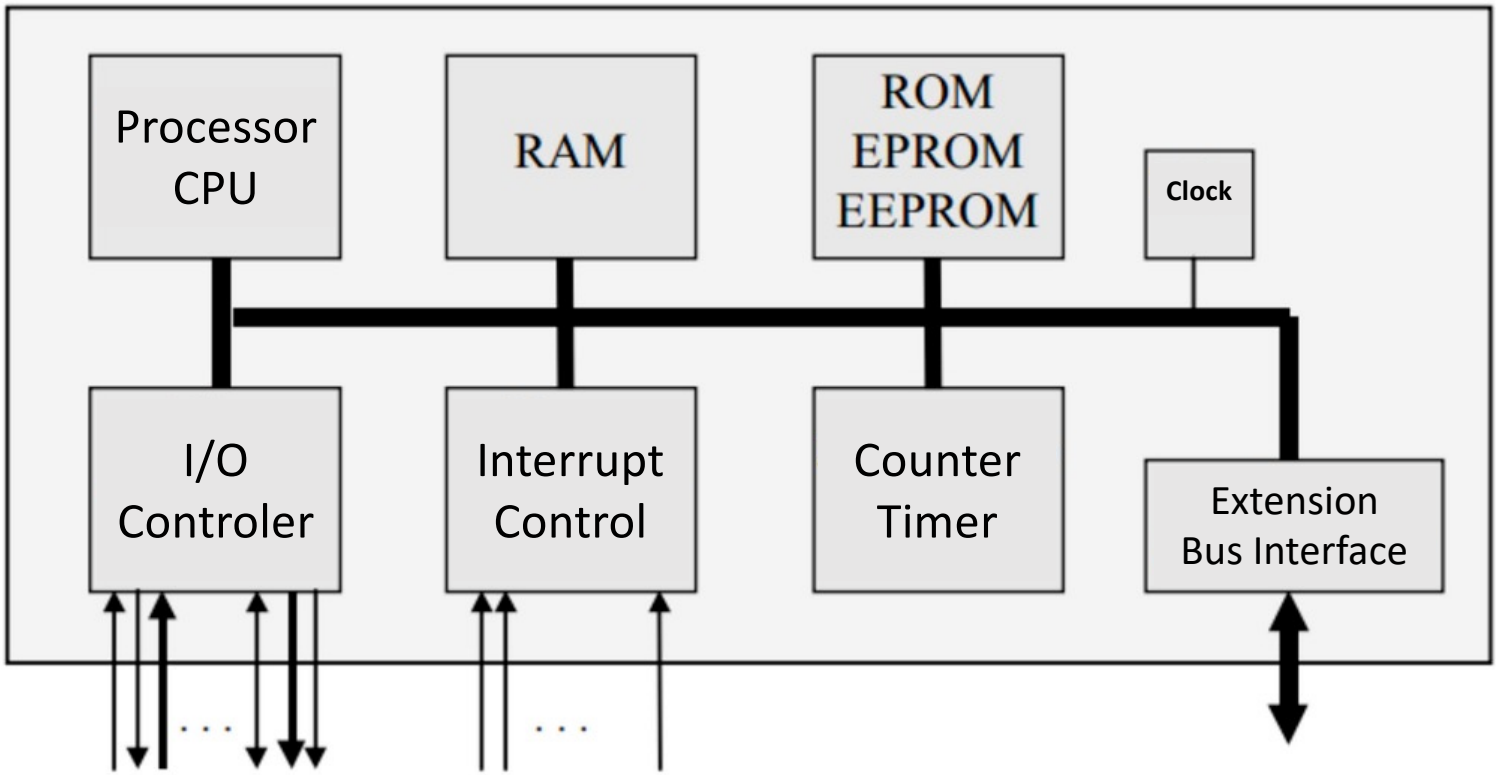
- L’unité de contrôle
  - Processeur
  - Micro-processeur
  - Micro-contrôleur
  - Système sur puce (System on Chip) (SoC)

  - Quelles sont les différences ?

Les Processeurs

Processeur	Micro-Processeur (µP)	Micro-Contrôleur (µC)	System-on-Chip
Unité centrale de traitement *Central Processing Unit* (CPU)	Processeur miniature à l'intérieur d'un PC • Bcp de mémoire • Syst. d'exploitation	Processeur embarqué p.ex. pour l'IdO (IoT) • Mémoire limitée • Bcp d'interfaces	Processeur avec composant additionnels sur la même puce p.ex. pour smartphones
Comprend: • Unité arithmétique et logique • Registres (mémoire) • Entrées / Sorties			
	Microprocesseur Intel	Microcontrôleurs Atmel	SoC sur le Raspberry Pi

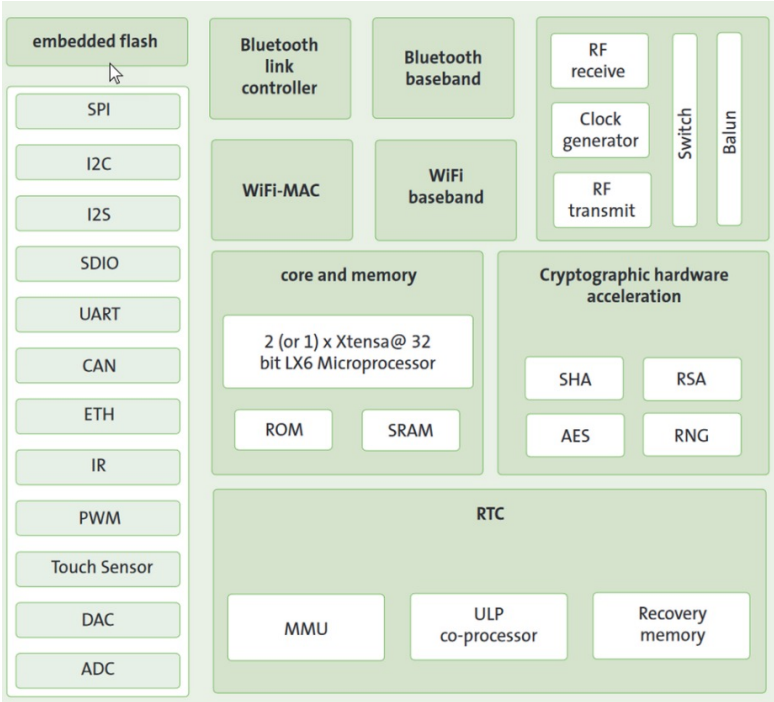
Le Microcontrôleur



Microcontrôleur ESP32

Composé de

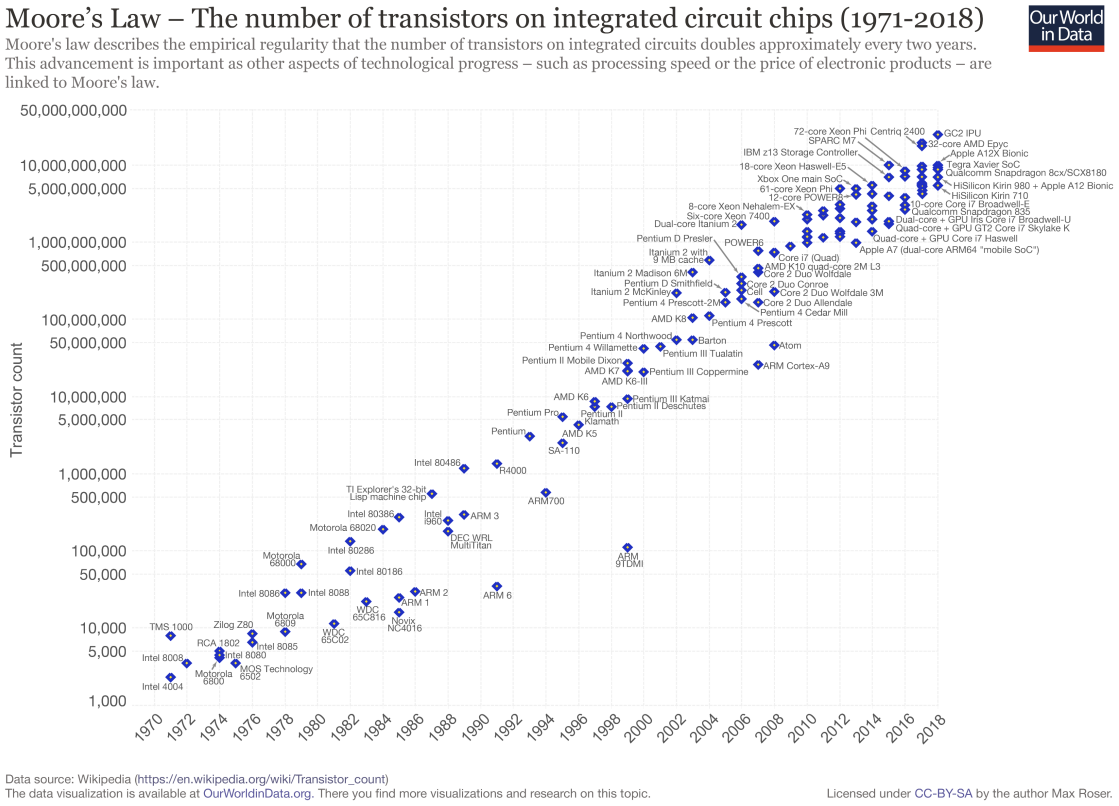
- Processeur 32 bit
- Mémoire (RAM, ROM)
- Communication wireless
  - WiFi, Bluetooth
- Interfaces
  - ADC, DAC, PWM, CAN, UART, I2C, …
- Security (crypto)



La loi de Moore

Evolution du nombre de transistor dans un circuit intégré

Il double tous les 2 ans !



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Questions

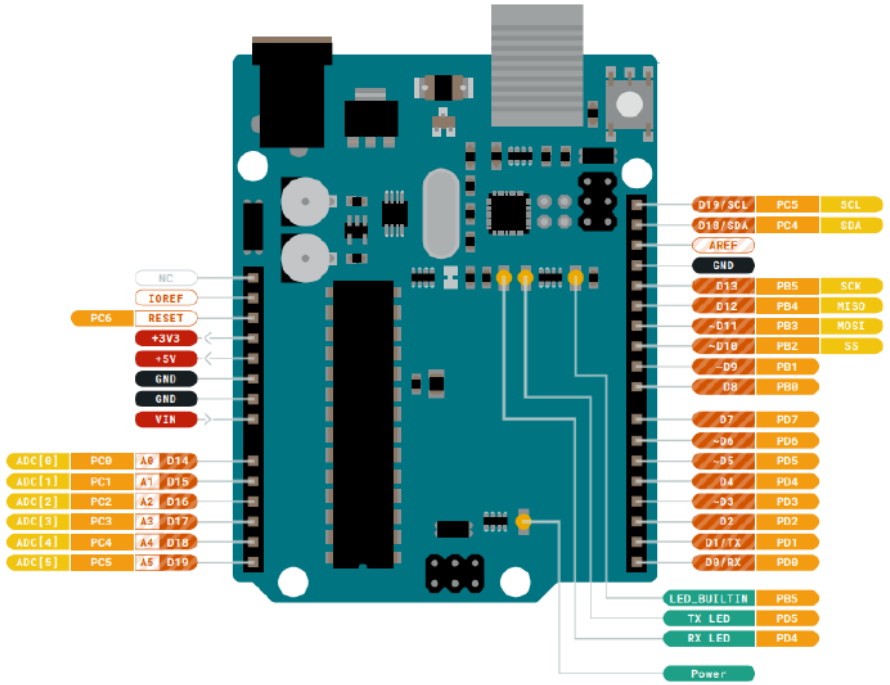
Quelles sont les différences entre:

- un micro-contrôleur et un micro-processeur ?

Kit Arduino Uno

Composants :

- µC ATmega328 (8-bit)
  - Entrées/sortie digitales
  - Entrées analogiques
  - PWM, UART, SPI, I2C, …
- µC ATmega16U2 (8-bit)
  - Communication USB



Programmation Arduino

- Installer Arduino IDE : https://www.arduino.cc/en/software
- Prenez la version Arduino IDE 2.0.3

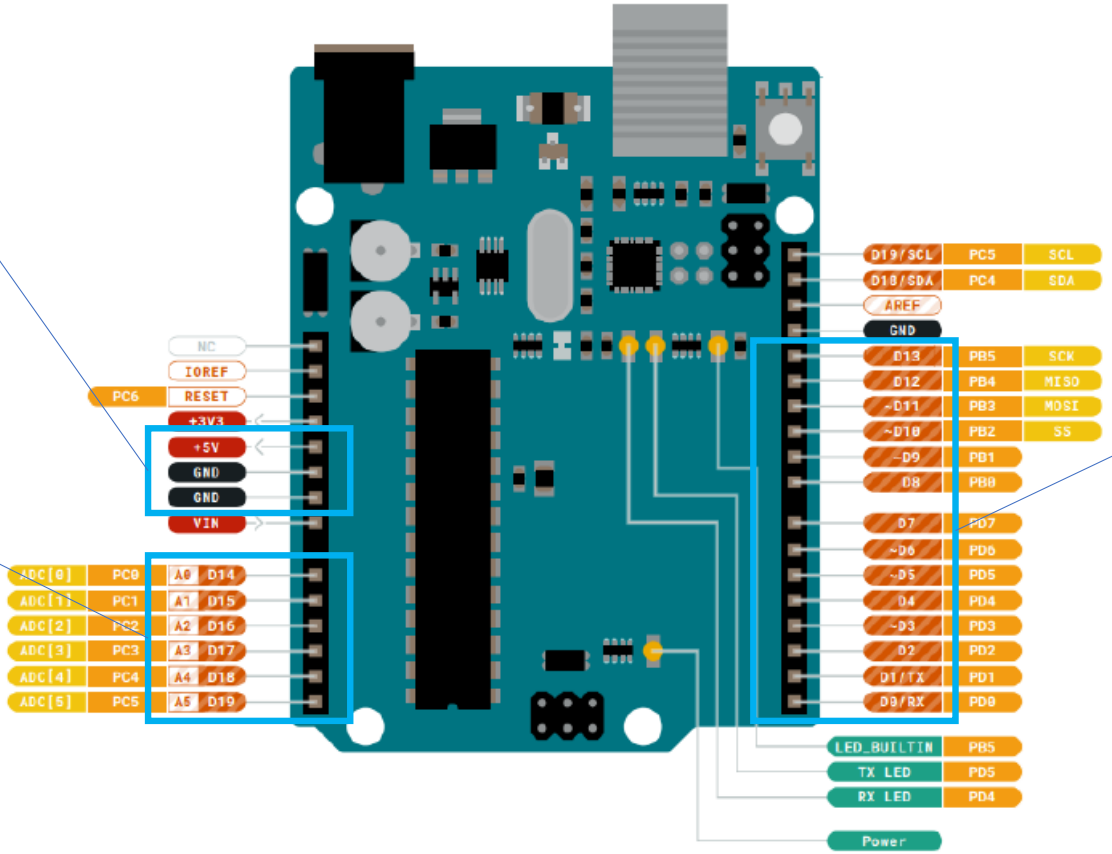
- Arduino UNO : https://docs.arduino.cc/hardware/uno-rev3

- Exemples de base : https://docs.arduino.cc/built-in-examples

La carte Arduino Uno

Broches 5V et GND
Utilisez ces broches pour alimenter vos circuits avec du +5V et une masse.

Entrées analogiques
Utilisez ces broches avec `analogRead()`



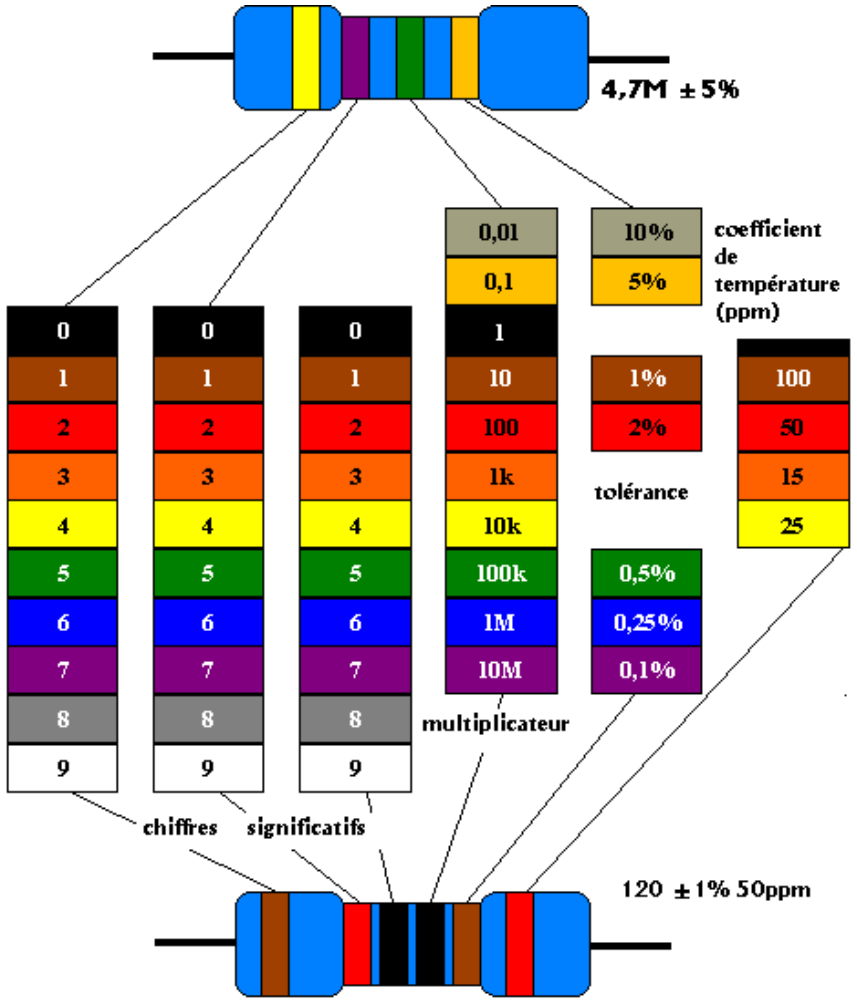
Broches numériques
Utilisez ces broches avec `digitalRead()`, `digitalWrite()` et `analogWrite()`

`analogWrite()` marche seulement sur les broches ayant le symbole PWW (~)

`pinMode(x, INPUT)` va configurer la broche `x` en lecture pour `digitalRead()`

`pinMode(x, OUTPUT)` va configurer la broche `x` en écriture pour `digitalWrite()` et `analogWrite()`

Code couleur des résistances



Kit Arduino Uno

Programmation

• `pinModel()`	Configuration des Entrées (In) / Sorties (Out) du µC
• `digitalRead()`	Lecture des états, consignes, 0 ou 1
• `digitalWrite()`	Ecriture des états, envoi de la commande, 0 ou 1
• `analogRead()`	Lecture analogique, mesure du monde ‘’physique’’
• `analogWrite()`	Ecriture pseudo-analogique via un convertisseur PWM (Pulse Width Modulation) en définissant le rapport cyclique
• `tone()`	Génère un signal carré à une fréquence spécifique
• `delay()`	Introduit un délai (en ms) entre certaines actions du µC

Les capteurs du kit Arduino

- Capteurs numériques — `digitalRead()`
  - Interrupteur
  - Bouton poussoir
  - Codage numérique
  - Interrupteur à bascule (Tilt)

  Le microcontrôleur (µC) lit 2 états possibles: 0 (LOW) ou 1 (HIGH)

- Capteurs analogiques — `analogRead()`
  - Capteur de température, p.ex. TMP36
  - Capteur de luminosité: photodiode ou phototransistor
  - Capteur d'accélération: accéléromètre
  - Capteur auditif : capteur piézoélectrique
  - Capteur sensitif : capteur capacitif
  - Potentiomètre, résistance variable en fonction d'une action mécanique
  - Echelle de résistances (remplace plusieurs entrées numériques)

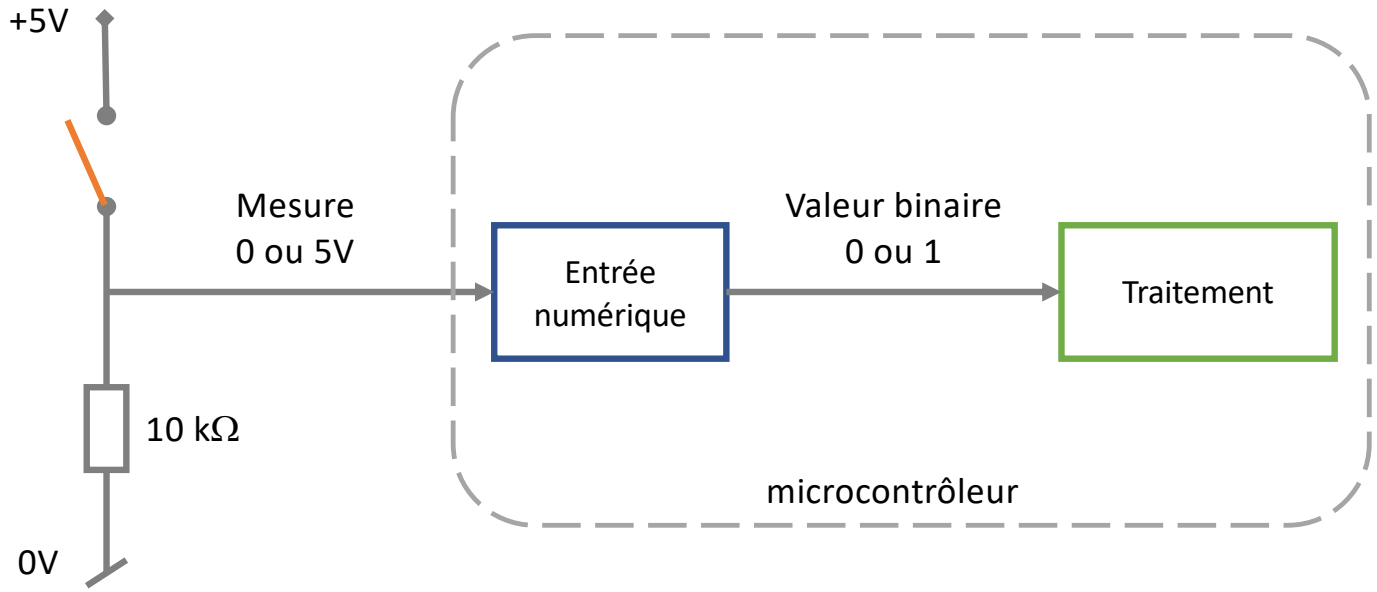
  Le convertisseur A/D du µC traduit une mesure de tension linéaire entre 0 et 5V en une valeur numérique entre 0 et 1023 (Un codage sur 10-bit offre 2^{10} = 1024 états possible)

Les capteurs numériques

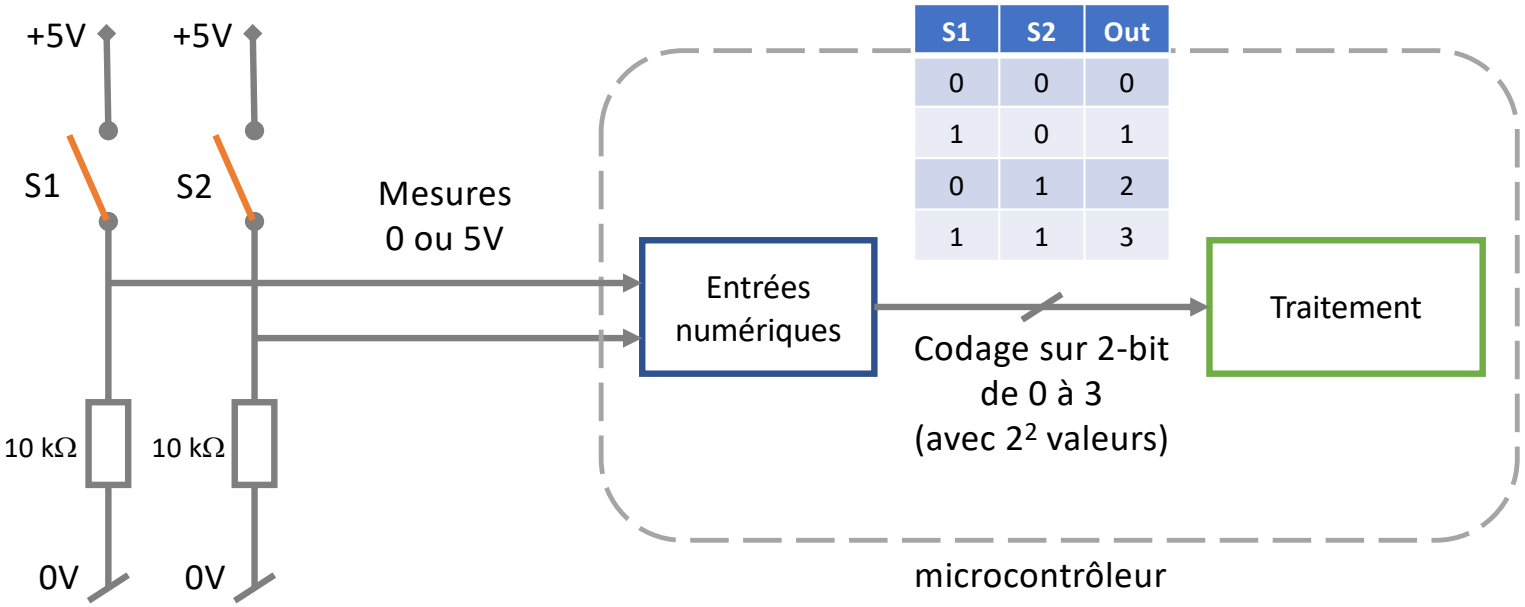
- Capteurs numériques `digitalRead()`
  - Interrupteur
  - Bouton poussoir
  - Codage numérique
  - Interrupteur à bascule (Tilt)

  Le microcontrôleur (µC) lit 2 états possibles: 0 (LOW) ou 1 (HIGH)

Interrupteur et bouton poussoir

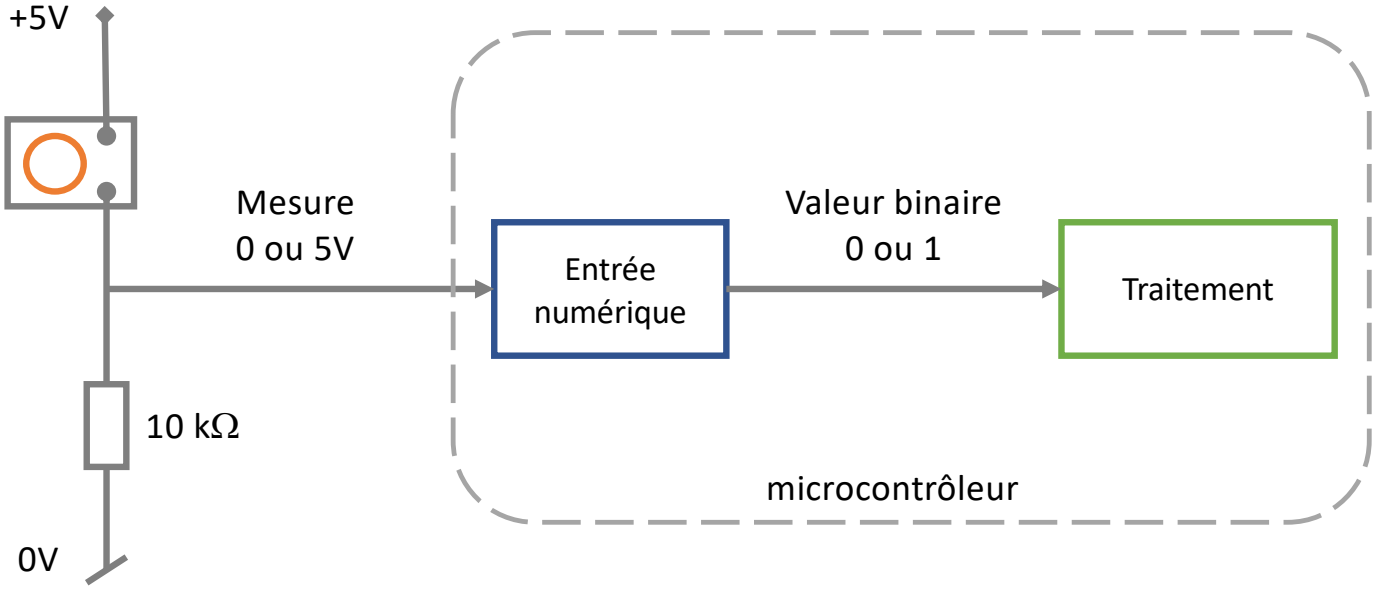


Codage numérique (clavier)



S1	S2	Out
0	0	0
1	0	1
0	1	2
1	1	3

Interrupteur à bascule (Tilt)



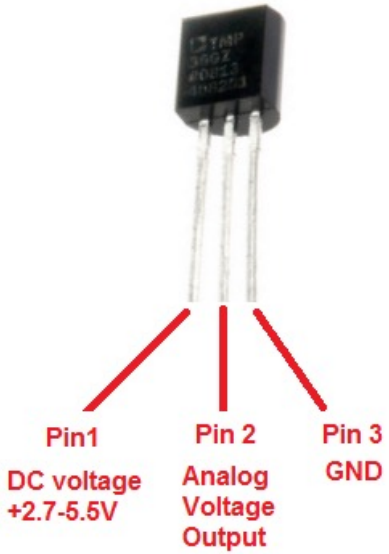
Les capteurs analogiques

- Capteurs analogiques `analogRead()`
  - Capteur de température, p.ex. TMP36
  - Capteur de luminosité: photodiode ou phototransistor
  - Capteur d'accélération: accéléromètre
  - Capteur auditif : capteur piézoélectrique
  - Capteur sensitif : capteur capacitif
  - Potentiomètre, résistance variable en fonction d'une action mécanique
  - Echelle de résistances (remplace plusieurs entrées numériques)

  Le convertisseur A/D du µC traduit une mesure de tension linéaire entre 0 et 5V en une valeur numérique entre 0 et 1023
  (Un codage sur 10-bit offre 2^{10} = 1024 états possible)

Capteur de température TMP36

- Capteur de température TMP36 (actif)
  - Ce composant génère une tension proportionnelle à la température.
    - Les pattes extérieures se connectent à l'alimentation et à la masse.
    - La tension présente sur la patte du milieu évolue en fonction de la température ambiante.



Capteur de température TMP36

Caractéristique linéaire de la tension *U* en fonction de la température *T*

Résolution du capteur $\frac{\Delta U}{\Delta T}$:

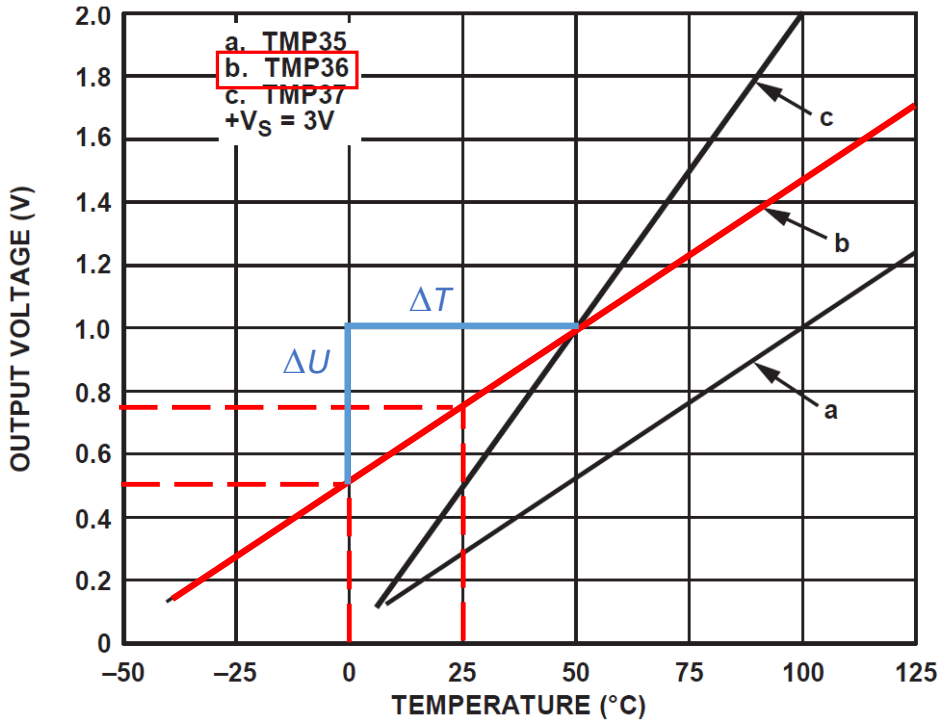
Prendre 2 points sur la courbe, p. ex. :

- Point 1 : $(U_1 = 1\text{V}, T_1 = 50°\text{C})$
- Point 2 : $(U_2 = 0.5\text{V}, T_2 = 0°\text{C})$

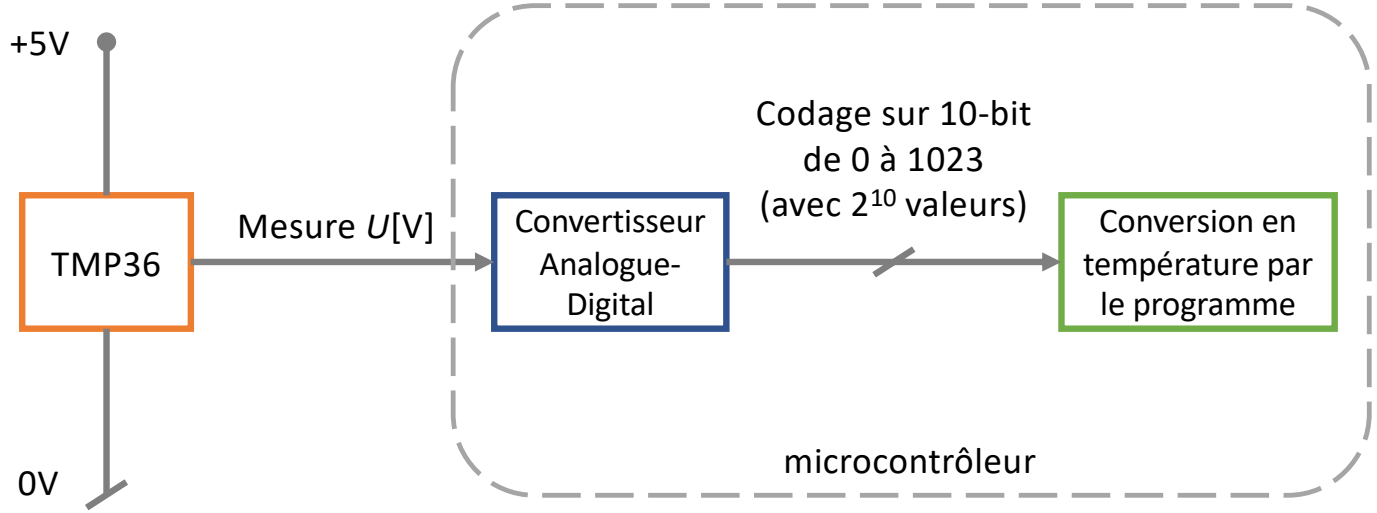
- $\frac{\Delta U}{\Delta T} = \frac{U_1 - U_2}{T_1 - T_2} = \frac{1\text{V} - 0.5\text{V}}{50°\text{C} - 0°\text{C}} = 10\,\frac{\text{mV}}{°\text{C}}$

Table 4. TMP35/TMP36/TMP37 Output Characteristics

Sensor	Offset Voltage (V)	Output Voltage Scaling (mV/°C)	Output Voltage at 25°C (mV)
TMP35	0	10	250
TMP36	0.5	10	750
TMP37	0	20	500



Lecture du capteur de température par le microcontrôleur



Conversion des données du capteur

- Spécification du capteur TMP36
  - U [V] = T [°C] * 0.01 V/°C + 0.5 V

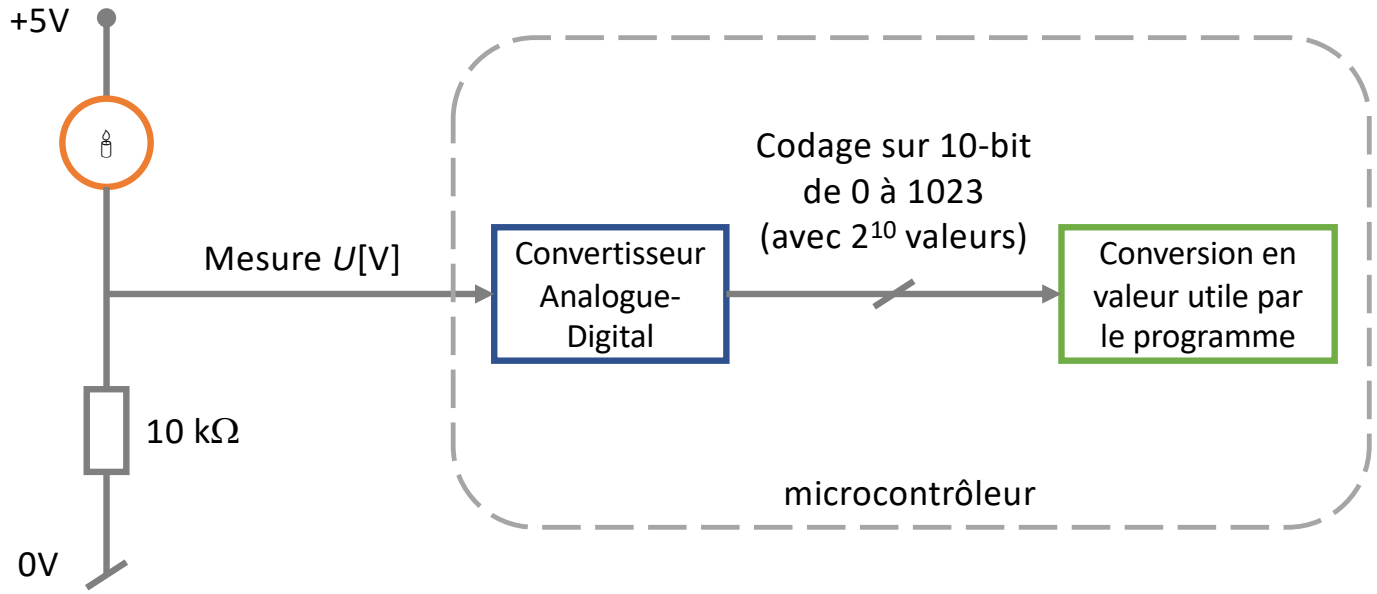
- Conversion A/D (sur 10-bit) -> en valeur binaire-décimale
  - valeurBinDec = U [V] / 5 V * 1024

- Code Programme Arduino

  Retrouver la tension U [V] :
  - U [V] = valeurBinDec / 1024 * 5 V

  Puis la température T [°C] lue par le capteur :
  - T [°C] = (U [V] – 0.5 V) * 100 °C/V

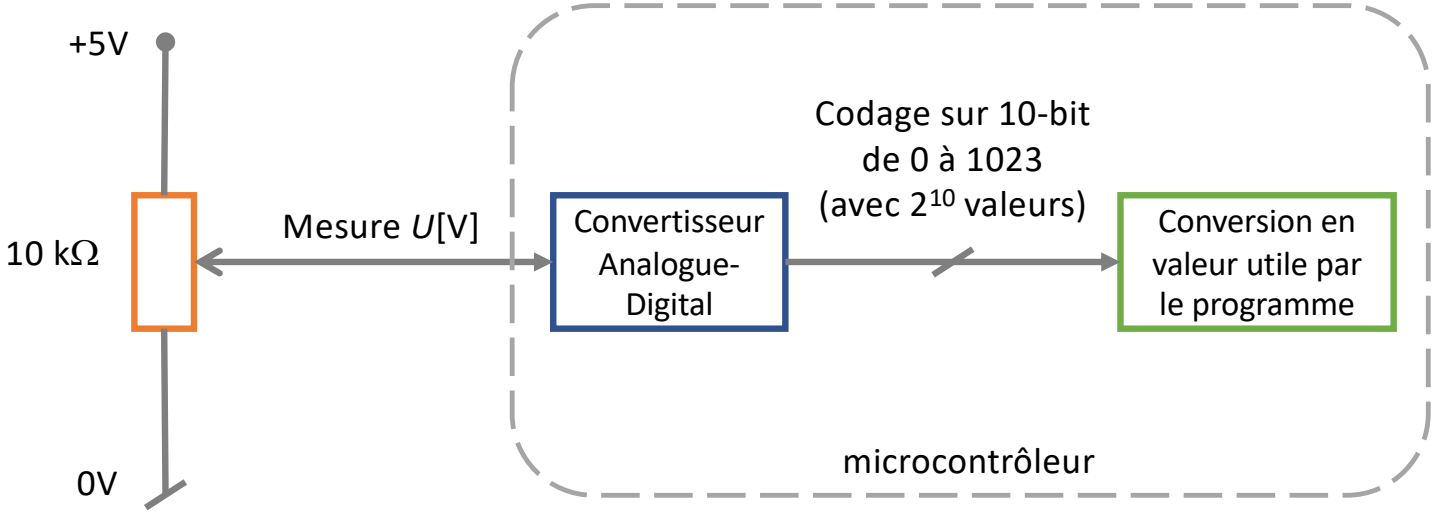
Lecture du capteur de température



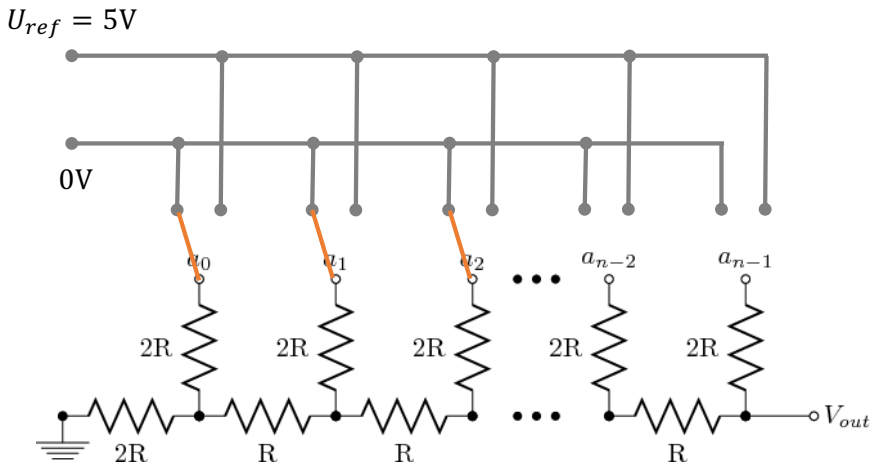
Calibration du capteur de luminosité

- Au contraire du capteur de température, il n’existe pas de formule de conversion linéaire entre la tension mesurée et la «luminosité».
- Nous allons définir une valeur de sensor pour une luminosité minimale ‘’sensorLow’’ et autre pour une luminosité maximale ‘’sensorHigh’’.

Lecture du potentiomètre



Echelle de résistances R-2R



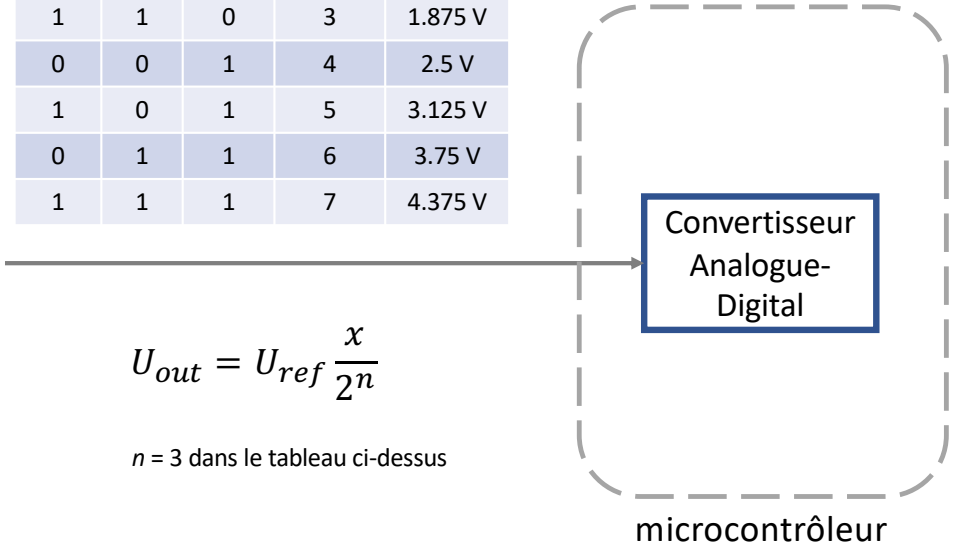
- Remplace plusieurs entrées numériques par une entrée analogique

Conversion D/A

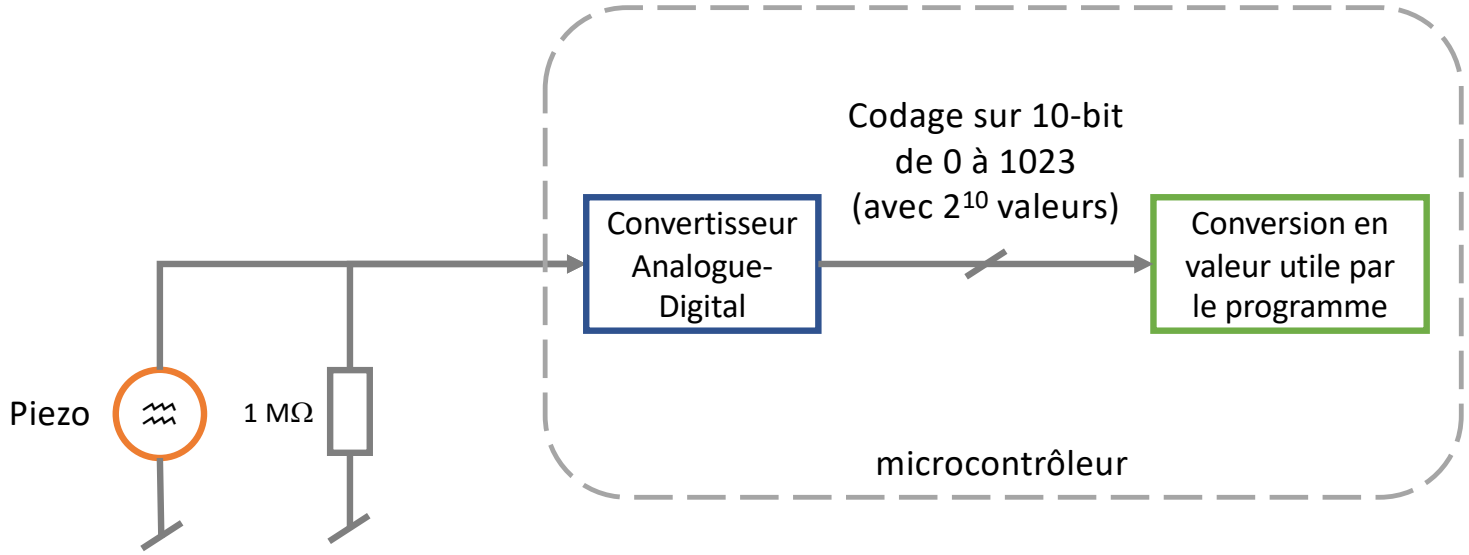
a_0	a_1	a_2	x	U_{out}
0	0	0	0	0 V
1	0	0	1	0.625 V
0	1	0	2	1.25 V
1	1	0	3	1.875 V
0	0	1	4	2.5 V
1	0	1	5	3.125 V
0	1	1	6	3.75 V
1	1	1	7	4.375 V

$$U_{out} = U_{ref} \frac{x}{2^n}$$

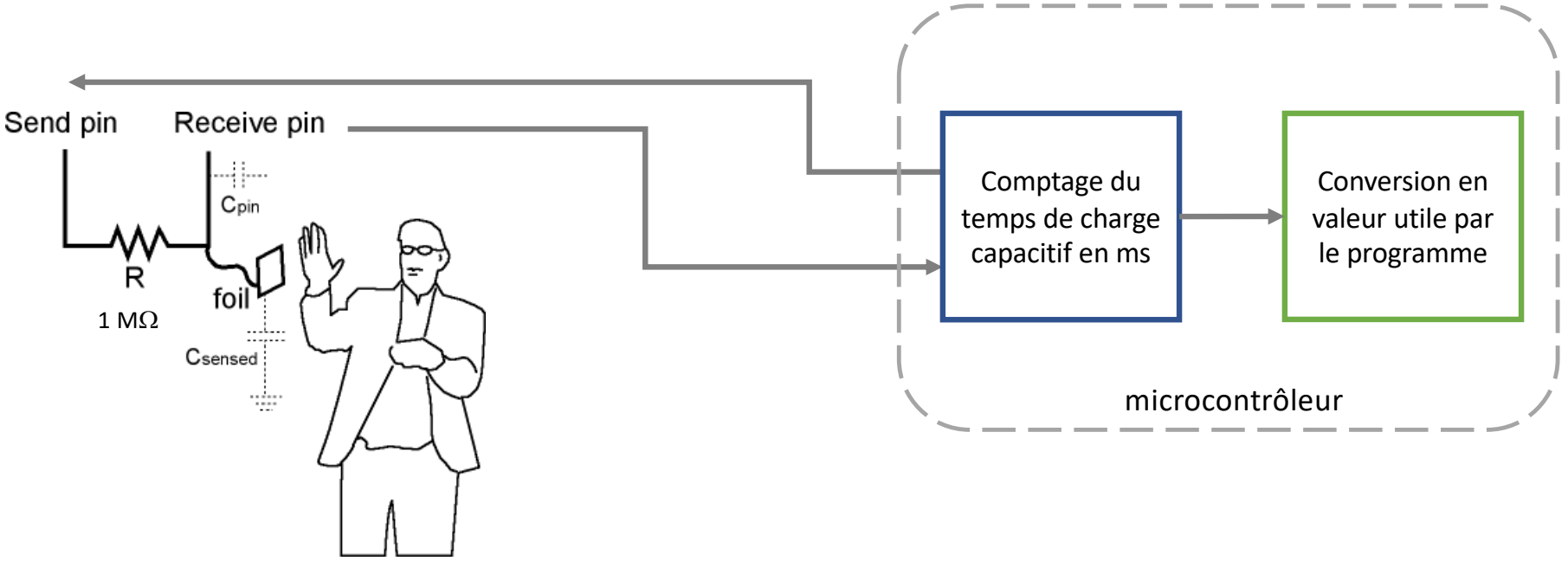
n = 3 dans le tableau ci-dessus



Capteur Piezoélectrique



Capteur Capacitif



Les actionneurs du kit Arduino

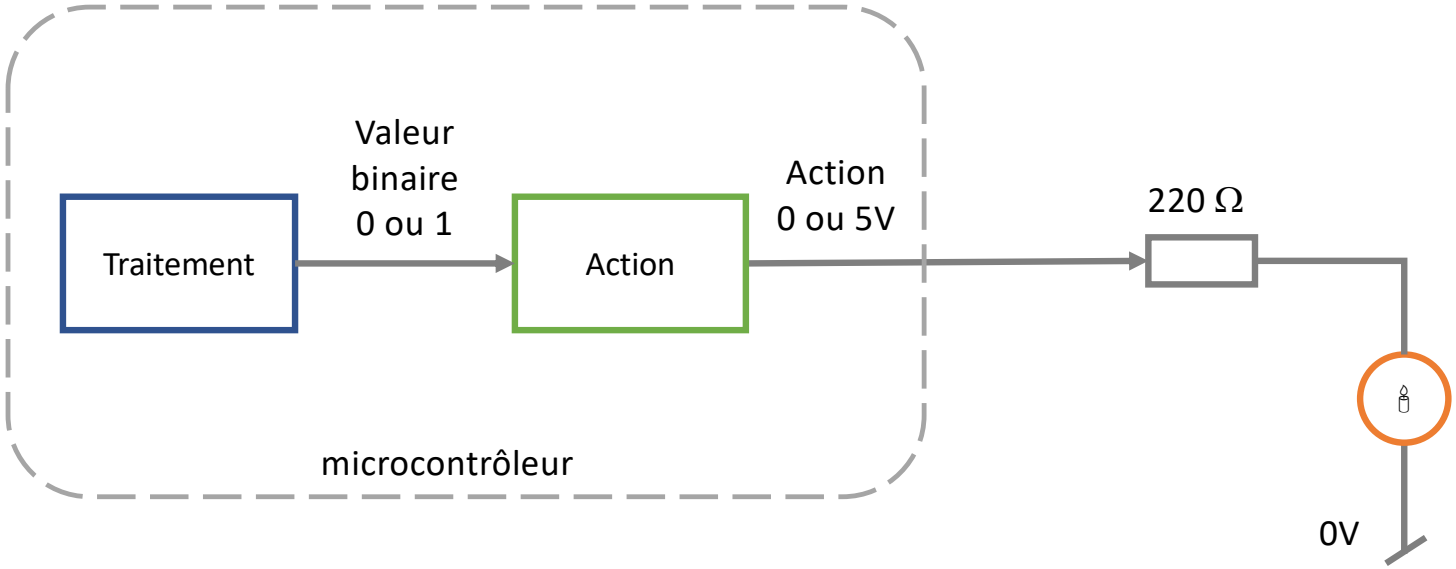
- Actionneurs numériques `digitalWrite()`
  - LED : éteinte ou allumée (Etat à 0 ou 1).
  - Buzzer piézoélectrique : produit un signal sonore via la fonction tone() qui génère un signal carré (alternance 0/1) à une fréquence spécifique.
  - Servomoteur : actionneur mécanique avec un nombre de positions discrètes (prédéfinies) sélectionnées au moyen d'une série d'impulsions (0/1).
  - Affichage alphanumérique à cristaux liquides (LCD).

- Actionneurs analogiques `analogWrite()`
  - LED : gradation lumineuse (au moyen d'un PWM).
  - Moteur à courant continu (piloté au moyen d'un pont en H).
  - Les amplificateurs à base de transistors.
  - Haut-parleur : produit un signal acoustique de fréquence et d'amplitude variable (nécessite une vraie sortie analogique)

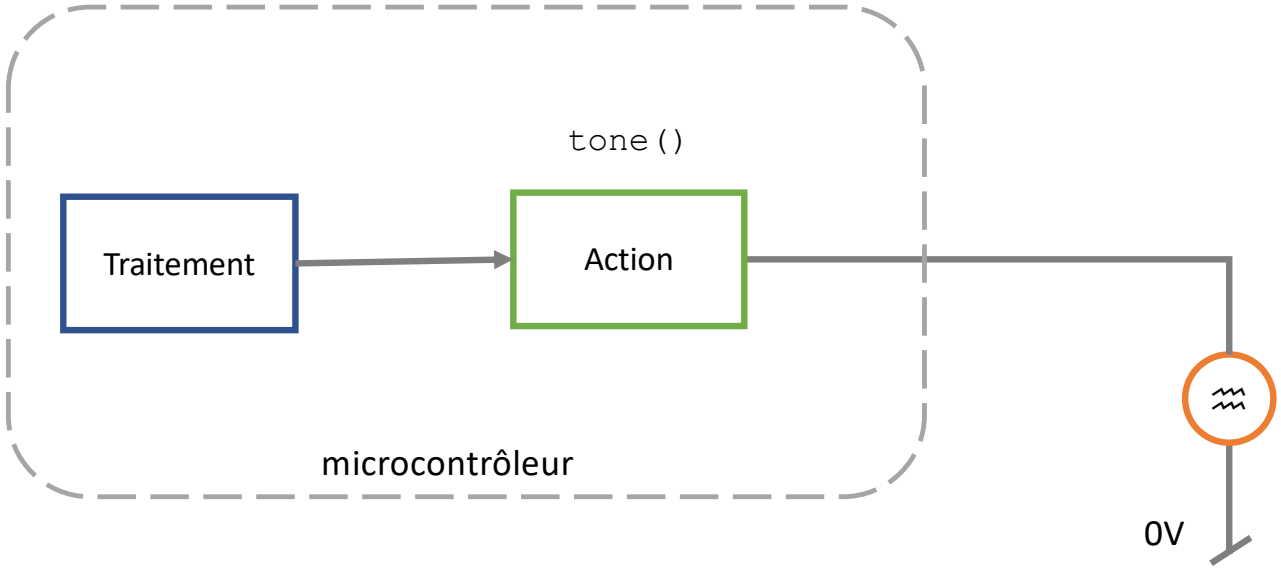
Actionneurs numériques

- Actionneurs numériques `digitalWrite()`
  - LED : éteinte ou allumée (Etat à 0 ou 1).
  - Buzzer piézoélectrique : produit un signal sonore via la fonction tone() qui génère un signal carré (alternance 0/1) à une fréquence spécifique.
  - Servomoteur : actionneur mécanique avec un nombre de positions discrètes (prédéfinies) sélectionnées au moyen d'une série d'impulsions (0/1).
  - Affichage alphanumérique à cristaux liquides (LCD).

Commande d’une LED



Commande d’un buzzer (piezo)

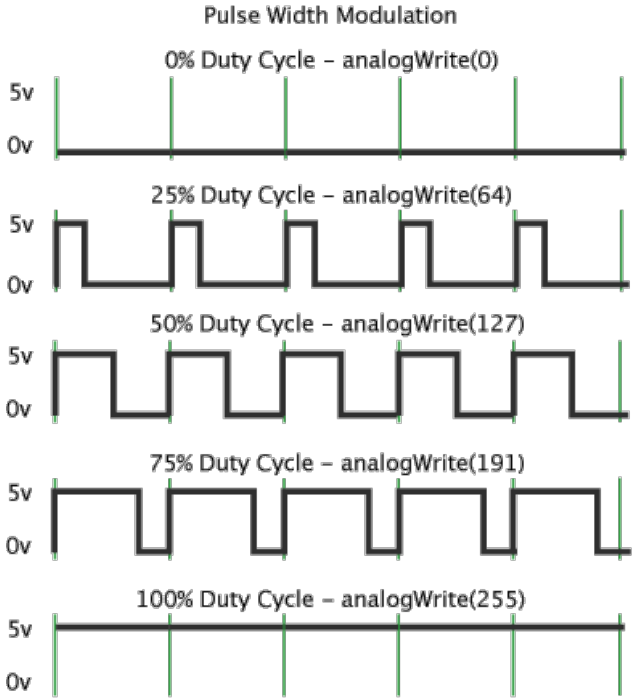


Actionneurs analogiques

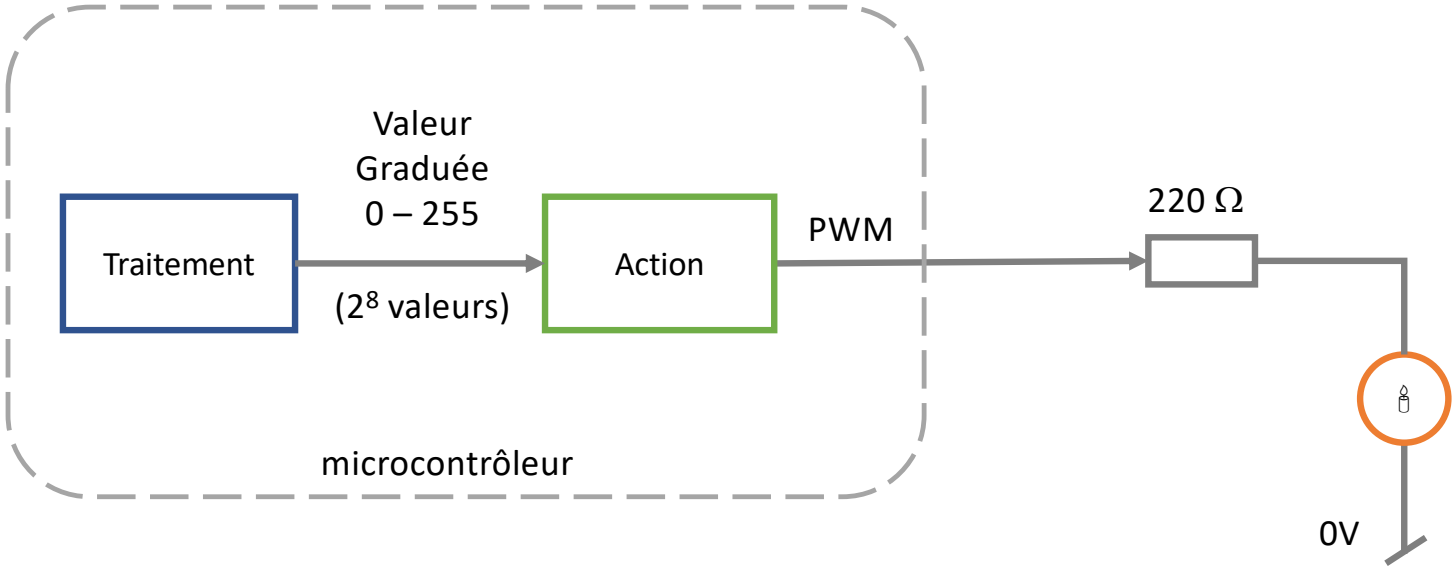
- Actionneurs analogiques `analogWrite()`
  - LED : gradation lumineuse (au moyen d'un PWM)
  - Moteur à courant continu (piloté au moyen d'un pont en H).
  - Les amplificateurs à base de transistors.
  - Haut-parleur : produit un signal acoustique de fréquence et d'amplitude variable (nécessite une vraie sortie analogique)

Modulation à largeur d'impulsion *Pulse Width Modulation* (PWM)

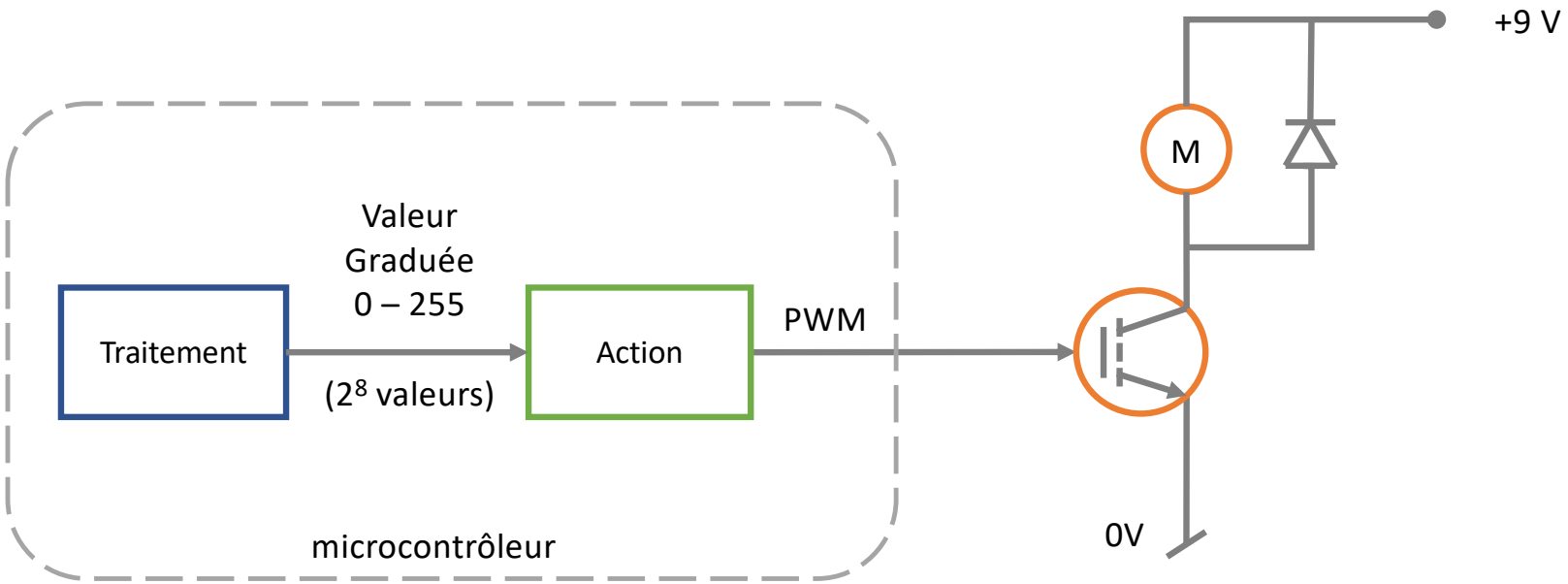
- La **modulation de largeur d'impulsions** (MLI ; en anglais : *Pulse Width Modulation*, soit PWM), est une technique couramment utilisée pour synthétiser des signaux pseudo analogiques à l'aide de circuits numériques (tout ou rien, 1 ou 0), ou plus généralement à états discrets.
- Le principe est de créer un signal logique (valant 0 ou 1), à fréquence fixe mais dont le **rapport cyclique** (en anglais: *Duty Cycle*) est contrôlé numériquement, la valeur moyenne de ce signal étant une grandeur analogique, égale au produit du rapport cyclique par l'amplitude maximale du signal.



Commande de l'intensité lumineuse d'une LED avec PWM

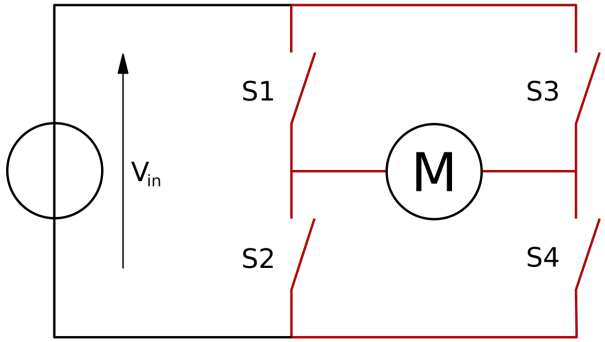


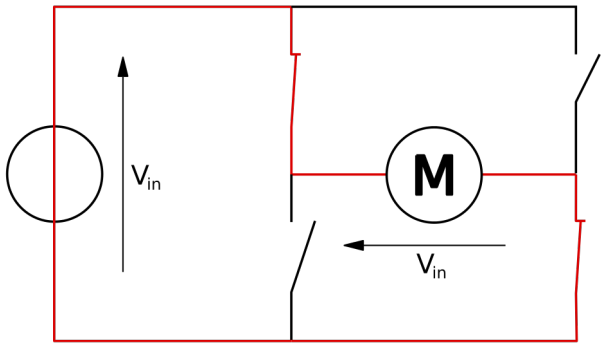
Commande de moteur à courant continu avec PWM et MOSFET

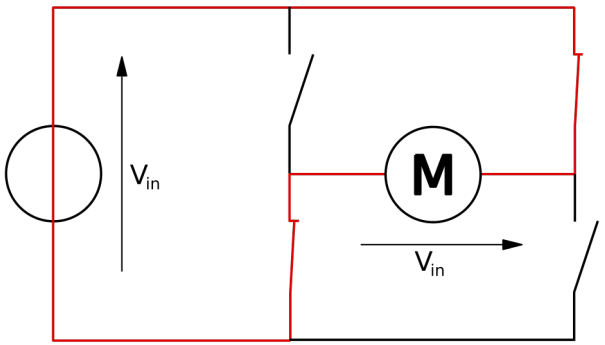


220 Ω

Commande moteur à courant continu avec pont en H







Les formats numériques

- Valeur logique (ou booléenne)
  - vrai (true) ou faux (false)
- Valeur entière
  - Signé : -4, -3, -2, -1, 0, 1, 2, 3, etc…
  - Non-signé : 0, 1, 2, 3, etc…
- Valeur en virgule flottante
  - p.ex. 3.14159, $7.2 \cdot 10^{-3}$, $-1.2 \cdot 10^{3}$, etc…

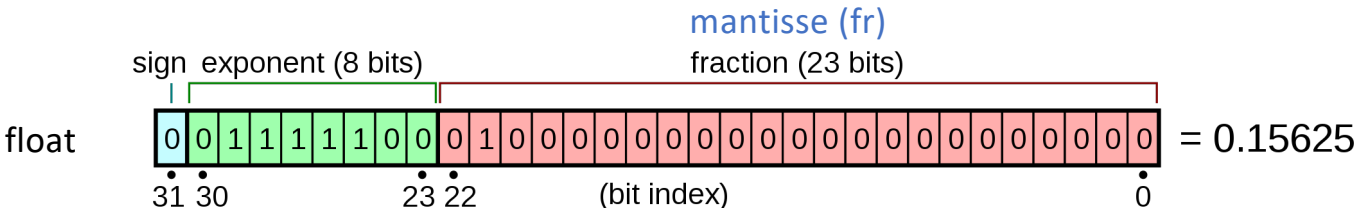
Les formats numériques entiers

Type	Nbr de bits	Valeurs
byte, char	8	0 – 255
short unsigned short	16	-32768 – 32767 0 – 65535
int unsigned int	32 (Ard Uno: 16)	-32768 – 32767 0 – 65535
long unsigned long	32	2'147'483'648 – 2'147'483'647 0 – 4'294'967'295
long long unsigned long long	64	

Il existent également des types entiers plus explicites : int8, int16, int32, int64
uint8, uint16, uint32, uint64

Les formats numériques à virgule-flottante

Type	Nbr de bits	Valeurs
float	32 s:1, e:8, m:23	$(2 - 2^{-23}) \times 2^{127}$ $\approx 3.4028235 \times 10^{38}$
double	64 (Ard Uno: 32) s:1, e:11, m:52	
long double	128	



Séquence temporelle

Avec bloquage (avec délai)

- Code Arduino

```
const int LED_PIN = LED_BUILTIN;
bool ledState = LOW;
const long interval = 1000;

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  ledState = !ledState;
  digitalWrite(LED_PIN, ledState);
  delay(interval);
}
```

Sans bloquage (avec timer)

- Code Arduino

```
const int LED_PIN = LED_BUILTIN;
bool ledState = LOW;
unsigned long previousMillis = 0;
const long interval = 1000;

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    ledState = !ledState;
    digitalWrite(LED_PIN, ledState);
  }
}
```