



roboonly

Interruptions

Les Timers

Déroulement du code (rappel)

- Le code en C/C++ est séquentiel
 - il s'exécute "de haut en bas"

```
int nombre = 0;
```

```
nombre++;
```

```
Serial.println(nombre); // 1
```

```
nombre += 8;
```

```
Serial.println(nombre*2); // 18
```

```
nombre = analogRead(A0);
```

```
Serial.println(nombre); // ?
```

Prérequis: Binaire, hexa, decimal,

- **Décimal**: base 10
 - Celle que l'on utilise au quotidien
- **Hexadécimal**: base 16
 - (a,b,c,d,e,f) pour (10,11,12,13,14,15)
 - Différentiée par **0x** au début du nombre
 - Ex: $0x2e = 2 * 16 + 14 = 48$
- **Binaire**: base 2
 - Différentiée par **0b** au début du nombre
 - $0b00000001 = 1$
 - $0b1010 = 12$

Prérequis: Opérations bit à bit

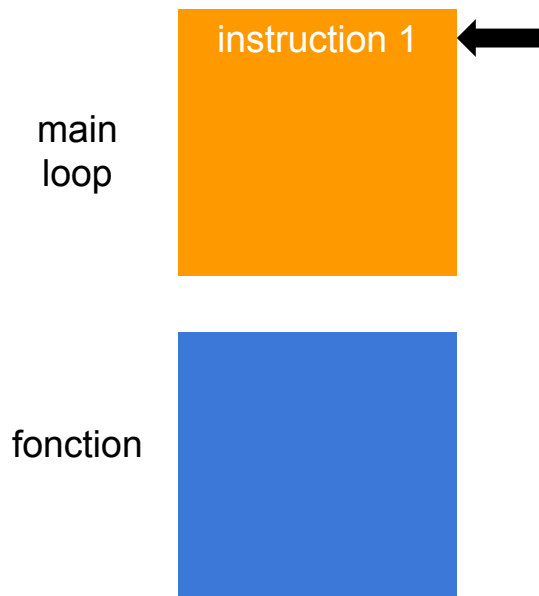
- **OU** logique : $|$
 - $1 | 0 = 1$
 - $0 | 0 = 0$
- **ET** logique : $\&$
 - $1 \& 0 = 0$
 - $1 \& 1 = 1$
- **NEGATION** : \sim
 - $\sim 1 = 0$
 - $\sim 0 = 1$
- **SHIFT** : \ll et \gg
 - $(1 \ll 2) = 0b100$
 - $(6 \gg 1) = 0b011$
- **XOR** : \wedge
 - $0 \wedge 1 = 1, \quad 0 \wedge 0 = 0, \quad 1 \wedge 1 = 0$

Prérequis: Opérations bit à bit

- Le uC contient des **registres** (paquets de bits) pouvant être modifiés par l'utilisateur.
- Exemples avec le bit 7 du registre PORTC
 - **Mettre à 1:**
 - `PORTC |= (1 << PORTC7);`
 - **Mettre à 0:**
 - `PORTC &= ~(1 << PORTC7);`
 - **Commuter:**
 - `PORTC ^= (1 << PORTC7);`
- Rappel: `x=x+1` équivalent à `x+=1`
 Marche aussi avec les opérateurs bits à bits

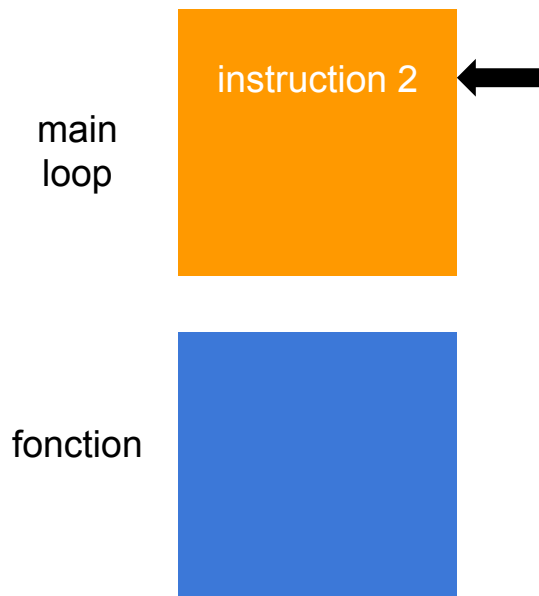
Les interruptions, pourquoi?

- Principe de base : interrompre le déroulement séquentiel du code
 - Rapidité d'action
 - Réaliser plusieurs tâches en même temps



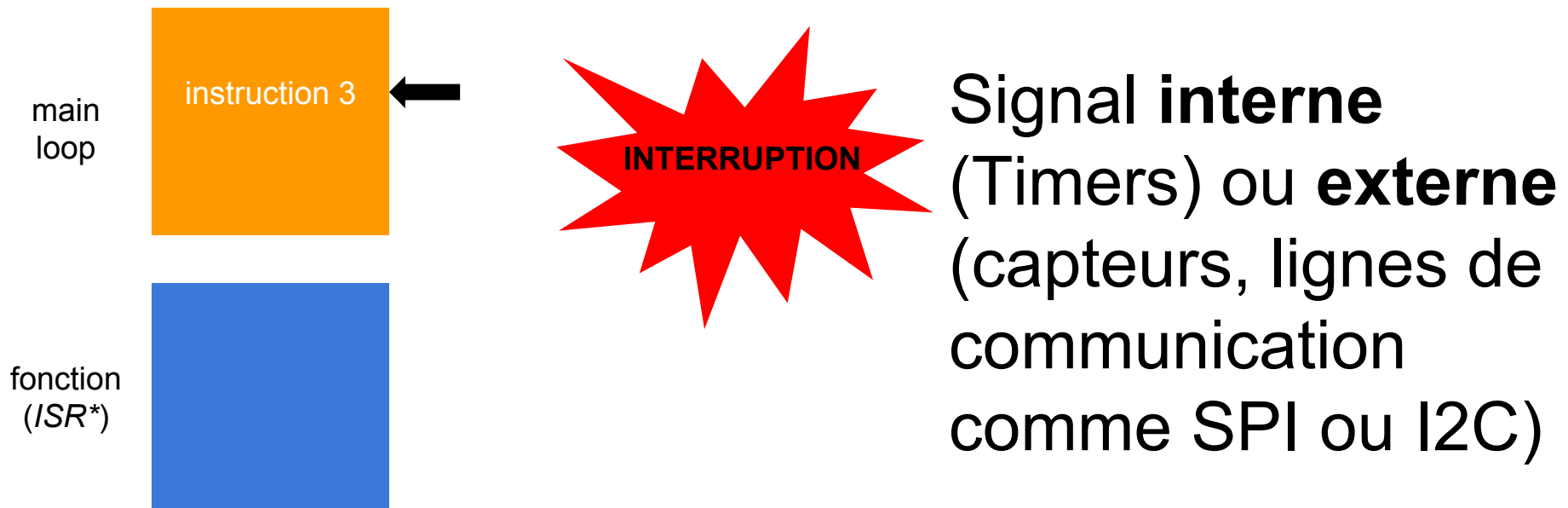
Les interruptions, pourquoi?

- Principe de base : interrompre le déroulement séquentiel du code
 - Rapidité d'action
 - Réaliser plusieurs tâches en même temps



Les interruptions, pourquoi?

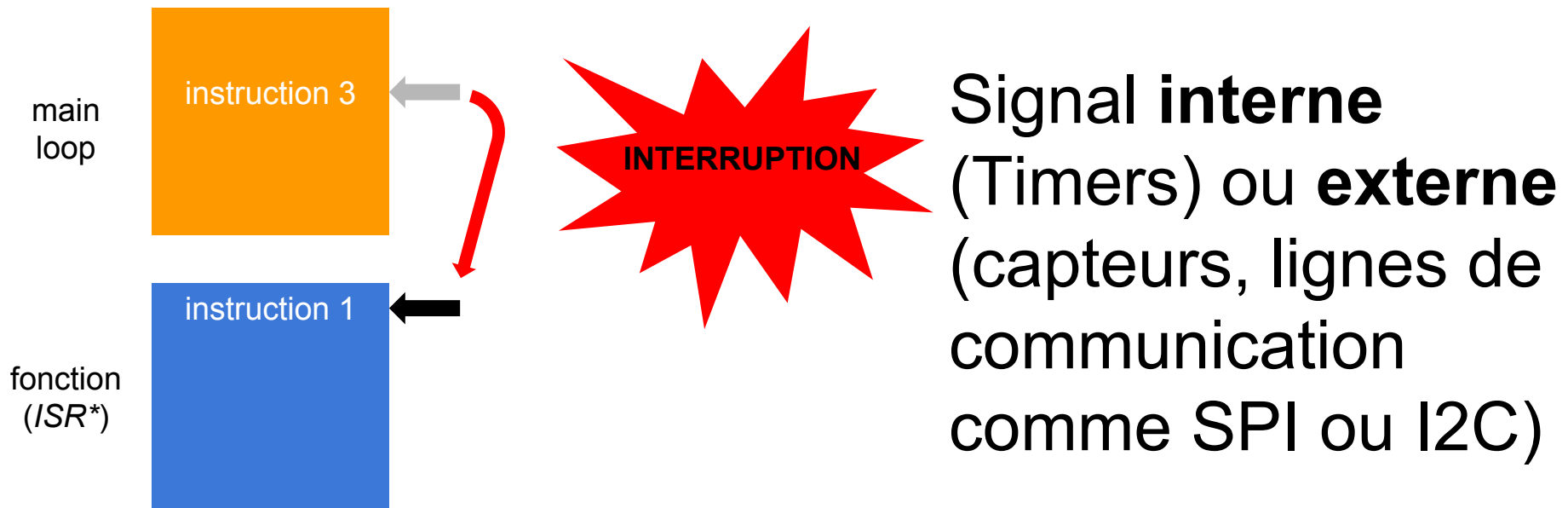
- Principe de base : interrompre le déroulement séquentiel du code
 - Rapidité d'action
 - Réaliser plusieurs tâches en même temps



*ISR = Interrupt Service Routine (vecteur d'interruption)

Les interruptions, pourquoi?

- Principe de base : interrompre le déroulement séquentiel du code
 - Rapidité d'action
 - Réaliser plusieurs tâches en même temps



*ISR = Interrupt Service Routine (vecteur d'interruption)

Les interruptions, pourquoi?

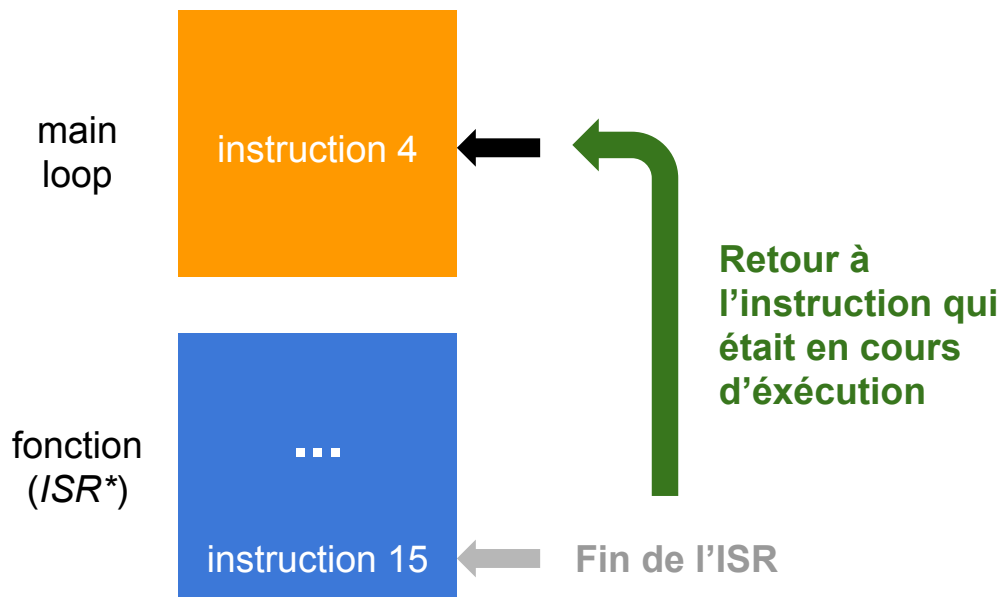
- Principe de base : interrompre le déroulement séquentiel du code
 - Rapidité d'action
 - Réaliser plusieurs tâches en même temps



*ISR = Interrupt Service Routine (vecteur d'interruption)

Les interruptions, pourquoi?

- Principe de base : interrompre le déroulement séquentiel du code
 - Rapidité d'action
 - Réaliser plusieurs tâches en même temps



*ISR = Interrupt Service Routine (vecteur d'interruption)

Interruptions, généralités

- Les variables modifiées dans une interruption doivent être déclarées *volatile*
 - `volatile char button_counter;`
- Les interruptions doivent être aussi courtes que possible
- Ne pas utiliser `millis ()` ou `delay ()` dans une interruption

Interruptions, the Arduino way

AttachInterrupt (

`digitalPinToInterrupt (pin) , ISR, mode)`

- `pin` peut valoir 0,1,2,3,7
- `ISR` est le nom de la fonction à appeler, sans paramètres et ne renvoie rien
- `mode` peut être:
 - CHANGE (flanc montant ou descendant)
 - RISING (flanc montant)
 - FALLING (flanc descendant)
 - LOW

Interruptions, the Arduino way

```
volatile char interrupted;

void button_interrupt();

void setup() {
  pinMode(3, INPUT); // set the pin to INPUT
  attachInterrupt(digitalPinToInterrupt(3), button_interrupt, FALLING); // attach interrupt
  Serial.begin(9600);
  interrupts(); // enable interrupt
}

void loop() {
  // put your main code here, to run repeatedly:
  if(interrupted)
  {
    Serial.println("There's been an interrupt");
    interrupted = 0;
  }
}

void button_interrupt() // interrupt routine
{
  interrupted = 1;
}

```

Interruptions, the Arduino way

- **detachInterrupt** (
 digitalPinToInterrupt (**pin**))
 - pour supprimer une interruption
- **interrupts** ()
 - activer les interruptions
- **noInterrupts** ()
 - désactiver les interruptions
(temporairement par exemples)

Interruptions, the AVR way

- **On sait exactement ce qu'il se passe**
 - Souvent Arduino ne renseigne pas sur ce qu'il désactive => Incompatibilités de certaines bibliothèques sans que cela soit documenté!
- Plus **complexe**, il faut chercher le [datasheet](#)
- Plus **puissant** car plus d'interruptions disponibles: PCINT, ICP (même si certaines bibliothèques Arduino les rendent disponibles)

Interruptions, the AVR way

- Plusieurs **registres à modifier pour le setup**
 - `EICRA/EICRB` pour définir le mode d'interruption
 - `EIMSK` pour activer certaines interruptions
 - `cli()`/`sei()` pour toutes les dés/activer
 - `ISR(INTn_vect)` noms des ISR
- Ça a l'air compliqué vu comme ça
- Le **datasheet** est clair et explique tout ceci (s'habituer à le lire)

Interruptions, the AVR way

```
volatile char interrupted;
```

```
void setup() {
  cli();
  Serial.begin(9600);
  DDRB &= ~(1 << PORTB0); // set PORTB to input (done by default)
  EICRA |= (1 << ISC01); // set interrupt 0 on falling edge
  EICRA &= ~(1 << ISC00); // same
  EIMSK |= (1 << INT0); // enable interrupt 0
  sei();
}
```

```
void loop() {
  // put your main code here, to run repeatedly:
  if(interrupted)
  {
    Serial.println("There's been an interruption");
    interrupted = 0;
  }
}
```

```
ISR(INT0_vect)
{
  unsigned char sreg = SREG; // save SREG
  interrupted = 1;
  SREG = sreg; //restore SREG
}
```



roboonly

Interruptions

Les Timers

Les timers, comment?

- Ce sont des **compteurs internes** du uC (sur 8, 10 ou 16 bits généralement)
- **Vitesse d'incrément** du compteur variable (multiple de la fréquence du uC)
- Peuvent **déclencher des interruptions** (internes) lorsque:
 - il y a **overflow** du compteur
 - le compteur atteint une **valeur choisie**

Les timers, pourquoi?

- Réaliser plusieurs tâches en même temps
- Faire du PWM (moteurs par ex.)
- Respect des timings:

```
while(1) {
    if(condition)
        // do some very long stuff
    else
        // do some short stuff
    digitalWrite(LED, HIGH);
    delay(100);
    digitalWrite(LED, LOW);
    delay(100);
}
```

La LED **ne clignotera pas à une fréquence fixe** (5 Hz) car la durée des instructions dans la condition varie

Les timers, the Robopoly way

```
setTimer (callback, interval, callNumber) ;
```

- **callback**: pointeur de fonction
- **interval**: nombre de periodes de 100ms entre 2 appels de **callback**
- **callNumber**: nombre de fois qu'on appelle la fonction
- Maximum 4 timers
- **Pas de servos!** (utilisent le même Timer...)
- 65s entre 2 appels maximum
- 256 appels maximum
- Renvoie l'ID de votre fonction

Les timers, the Robopoly way

- **unsetTimer** (`callback_id`)
 - enleve la fonction identifiée par `callback_id` de la liste d'appels
- Lisez le [Readme](#) pour plus de détails
- Toujours **vérifier les comptabilités** entre les libraries (Servos, setSpeed, setTimer, etc...)

Les timers, the Robopoly way

```
void led_toggle();

void setup() {
  setTimer(led_toggle, 10);
  pinMode(LED, OUTPUT);
  interrupts();
}

void loop() {
  // put your main code here, to run repeatedly:
}

void led_toggle()
{
  digitalWrite(LED, !digitalRead(LED));
}
```


Les timers, the AVR way

- Moins de magie, **on sait exactement ce qu'il se passe**
- Plus complexe, il faut chercher dans le [datasheet](#)
- Meilleur controle des timings!

Les timers, the AVR way

- **Interruption** lorsque:
 - il y a **overflow** du compteur
 - le compteur atteint une **valeur choisie** (fonctionne par comparaison de valeur)
- Les **registres importants**:
 - TCCR: Timer configuration
 - TCNT: Timer counter (valeur du timer)
 - TIMSK: Timer interruptions mask
 - OCR_i: Output comparison

Les timers, the AVR way

```

#define LED PORTC7 // LED is on PORTC, 7th pin

void setup() {
  Serial.begin(9600);
  DDRC |= (1 << LED); // LED to output

  cli(); // disable interrupts
  PRR0 &= ~(1 << PRTIM1); // enable timer1, enabled by default
  TIMSK1 |= (1 << OCIE1A); // enable output compare A
  TCCR1A &= ~((1 << COM1A1) | (1 << COM1A0) | (1 << WGM11) | (1 << WGM10)); // remove pwm pins
  TCCR1B |= ((1 << CS12) | (1 << CS10)); // prescaler to 1024, 15 625 Hz counter
  TCCR1B &= ~((1 << CS11) | (1 << WGM13) | (1 << WGM12)); // prescaler & remove PWM pins
  TCNT1 = 0; // set counter to 0
  OCR1A = 0x3d09; // 0x3d09 is 15 625 in hex
  sei(); // enable interrupts
}

ISR(TIMER1_COMPA_vect)
{
  char _sreg = SREG; // save sreg
  OCR1A = TCNT1 + 0x3d09; // next interrupt in 1 s

  PORTC ^= (1 << LED); // toggle LED
  SREG = _sreg; // restore sreg
}

```

Aller plus loin...

- Plus d'infos sur:
 - Polycopié de *Microcontrolleurs* (disponible en lecture au local)
 - www.avrfreaks.net
 - “Comprendre les microcontrolleurs” sur [Coursera](https://www.coursera.org) (Pas de date de future session)

Calendrier du semestre

- **Workshop III**
 - Samedi 18 mars au local
 - Homologation au concours
- **Grand Concours**
 - Samedi 1er avril en Co1

Contact/Infos

Contact principal

robopoly@epfl.ch

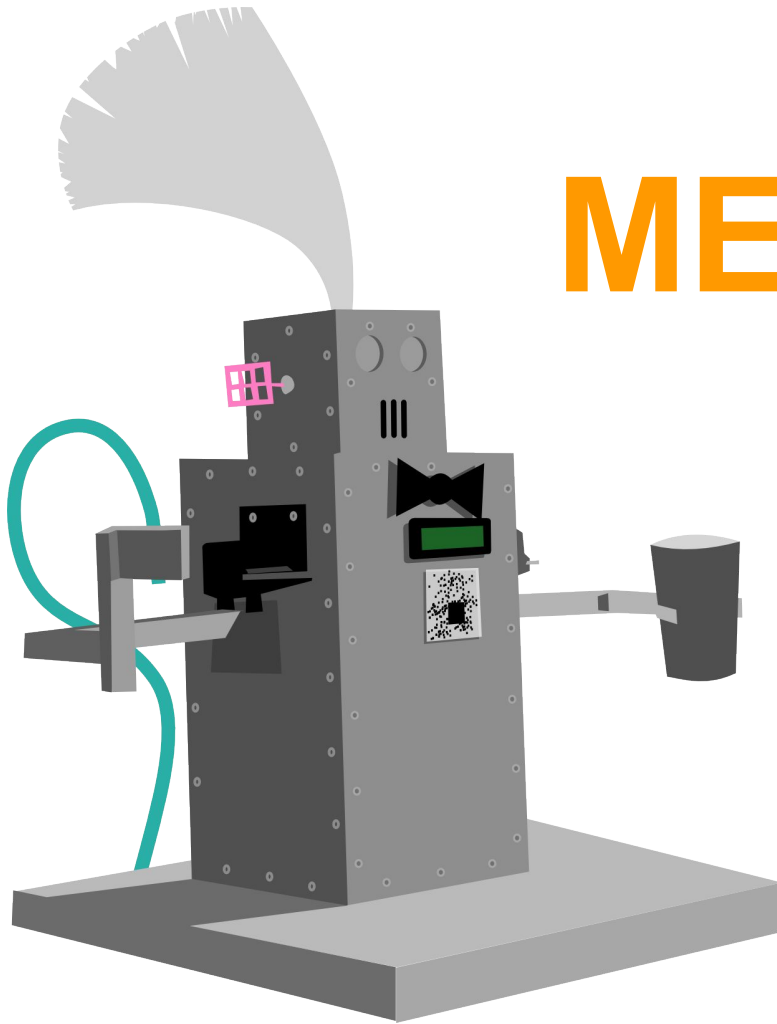
Site officiel - toutes les infos et slides sont la!

robopoly.epfl.ch

Facebook - pour suivre l'actualité du club!

www.facebook.com/robopoly

MERCI!



Questions?