

Enseignes et afficheurs à LED

Les interruptions



Dr. Yves Tiecoura

Les interruptions



Dr. Yves Tiecoura

Les interruptions



- Principe des interruptions
- Événements produisant des interruptions
- Mise en œuvre
- Deux exemples

Motivation des interruptions

De manière générale un microcontrôleur doit être programmé pour :

- détecter des changements sur ses entrées
- agir en conséquence sur ses sorties

Dans les enseignes et afficheurs à LED :

- Le système n'a souvent que des sorties...
- Dans certains cas, il doit réagir à des entrées (ex : télécommande)
- Il doit exécuter des tâches à des instants précis (ex : matrices multiplexées)

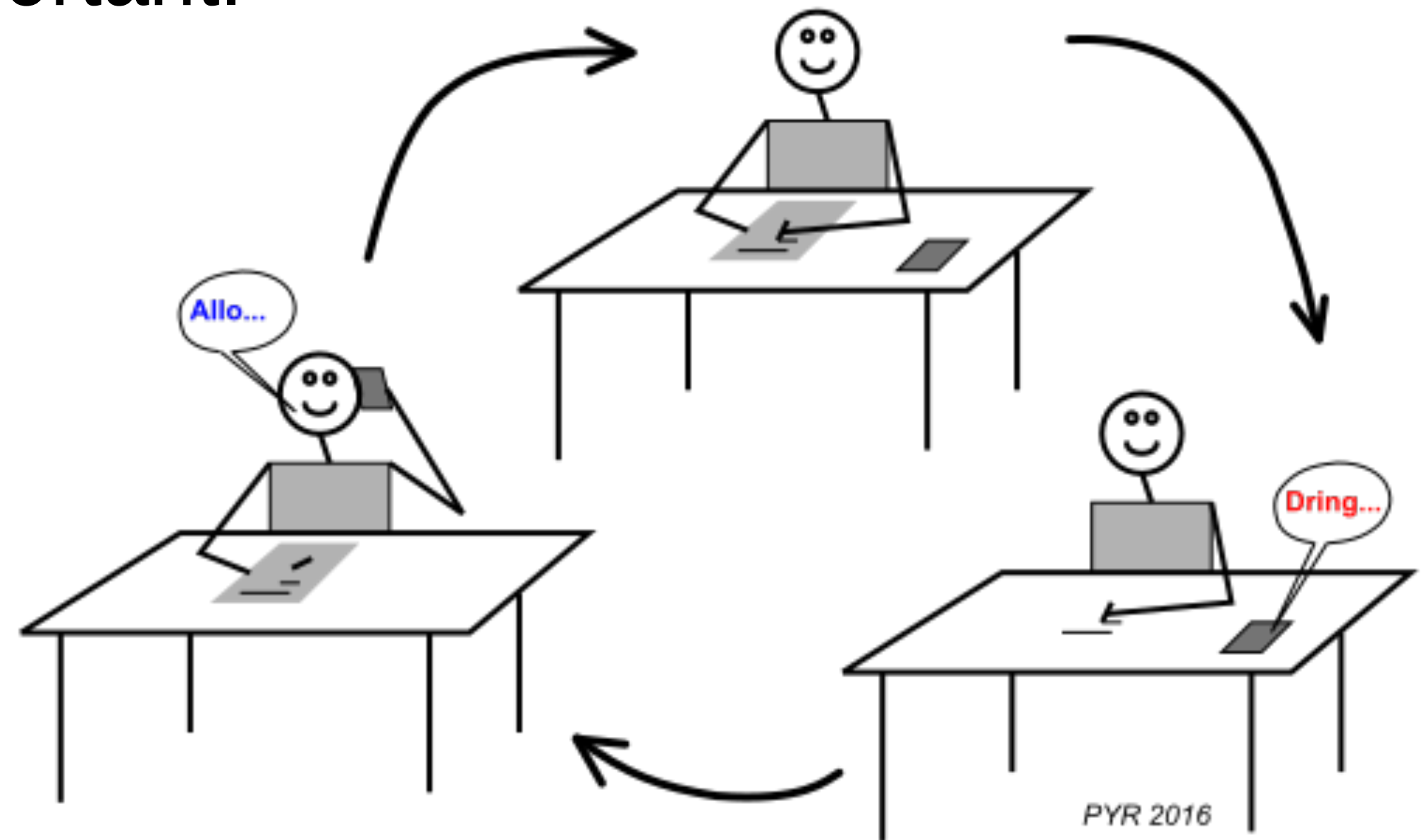
Définition



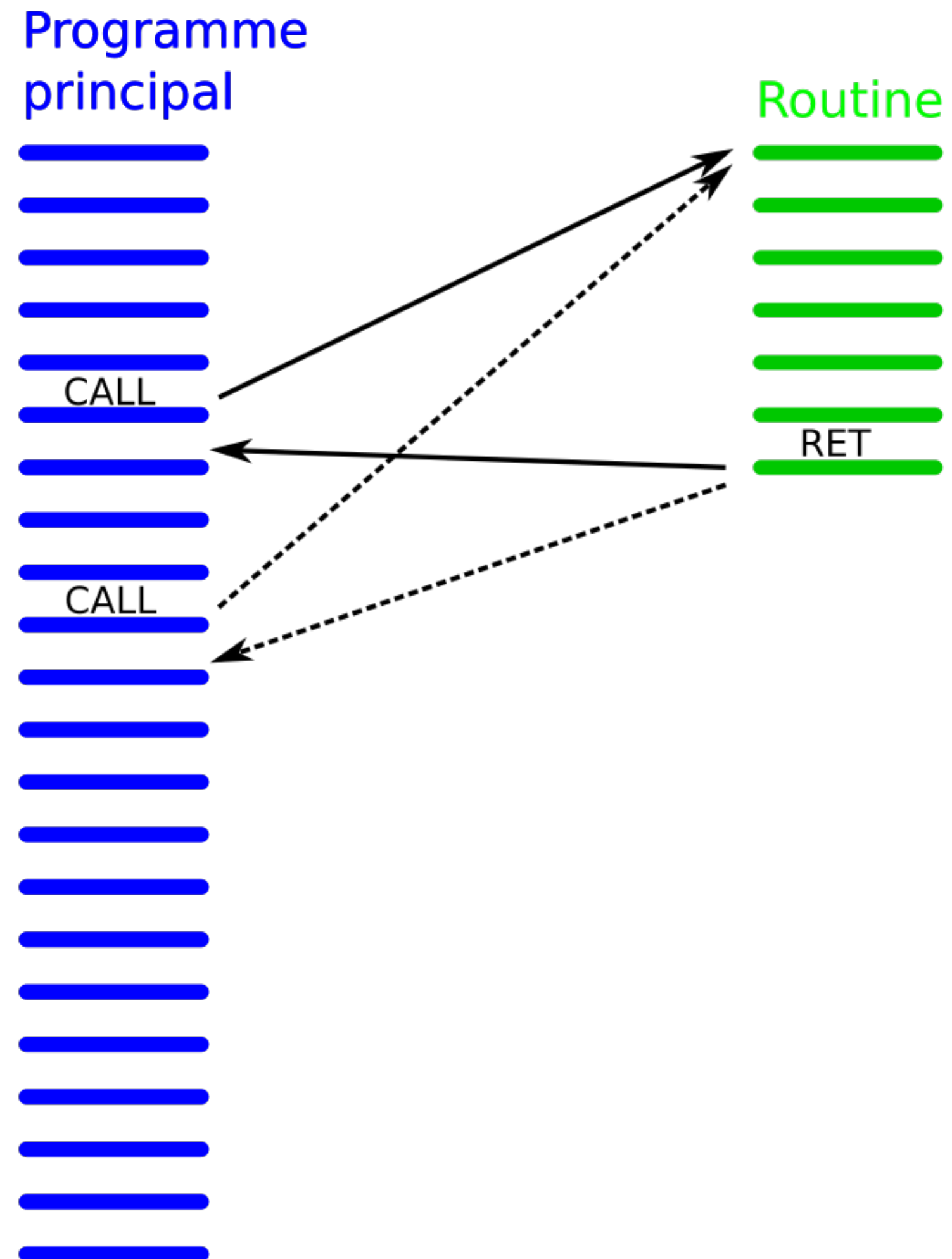
On appelle interruption, l'arrêt temporaire d'un programme au profit d'un autre programme, jugé à cet instant plus important.

Dans la vie courante :

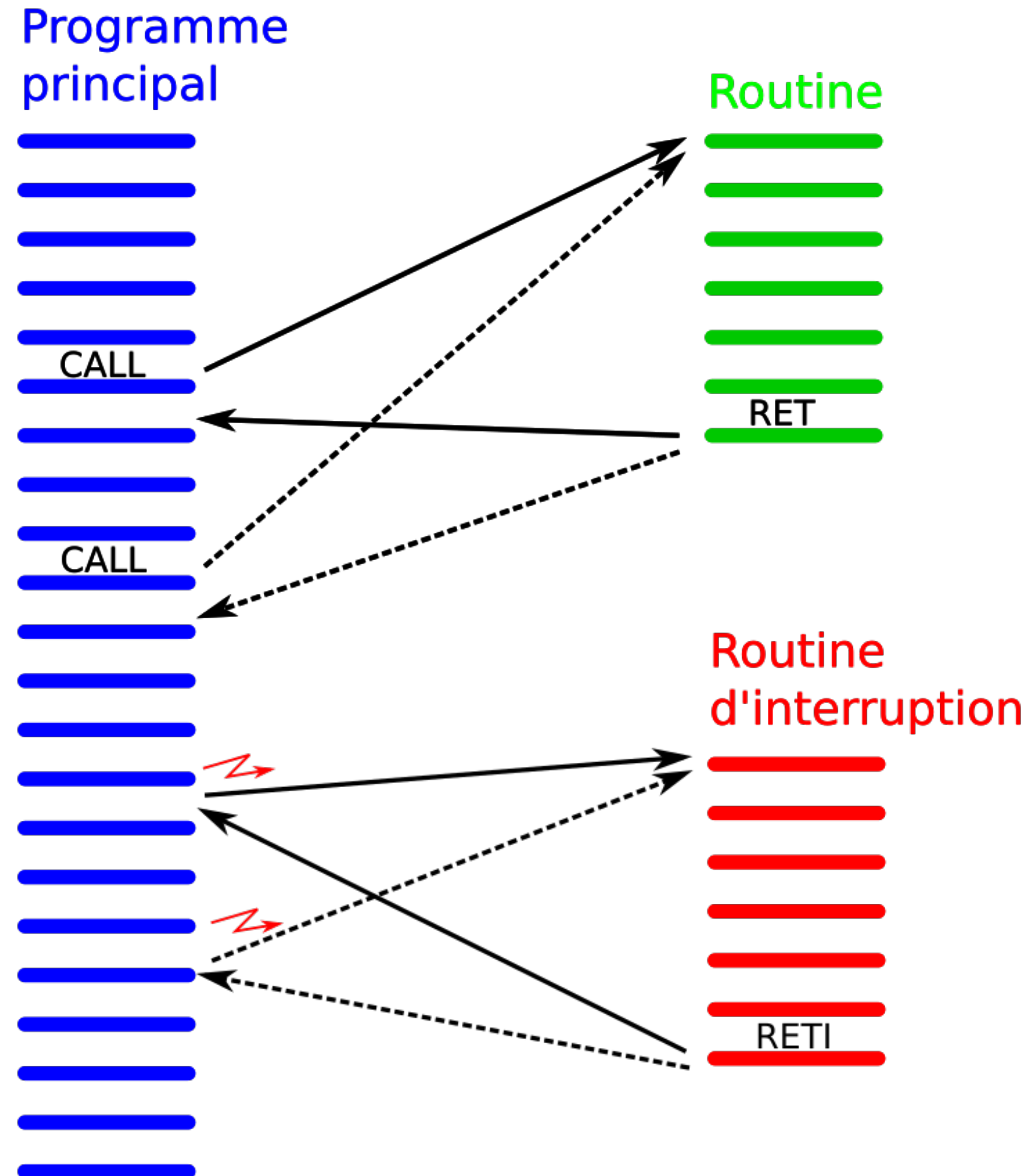
- Je suis en train de travailler
- Le téléphone sonne
- Je vais répondre au téléphone
- Après la conversation, je reprends mon travail là où je l'avais laissé.



Routine



Routine d'interruption



Événements produisant des interruption



Deux sortes d'événements produisant des interruptions :

- Les événements **extérieurs** au microcontrôleur
- Les événements **intérieurs** au microcontrôleur

...dont les événements liées aux Timers.



Discrimination des sources d'interruption

Il y a plusieurs sources d'interruptions sur un microcontrôleur

Le système doit être capable d'en connaître la source

- En consultant les fanions correspondant à chaque interruption
- Grâce aux **vecteurs d'interruption** (*interrupt vectors*)



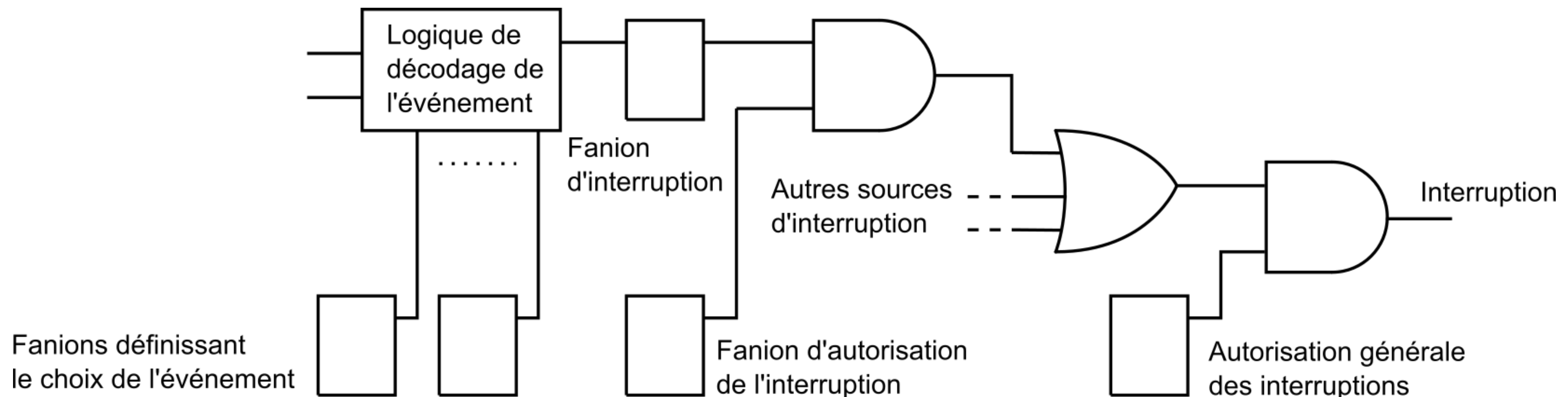
Vecteurs d'interruption sur un MSP430G

- 0xFFFFE : Reset
- 0xFFFFC : NMI
- 0xFFFFA : Timer1 CCR0
- 0xFFFF8 : Timer1 CCR1, CCR2, TAIFG
- 0xFFFF6 : Comparator_A
- 0xFFFF4 : Watchdog Timer
- 0xFFFF2 : Timer0 CCR0
- 0xFFFF0 : Timer0 CCR1, CCR2, TAIFG
- 0xFFEE : USCI status
- 0xFFEC : USCI receive/transmit
- 0xFFEA : ADC10
- 0xFFE8 : -
- 0xFFE6 : Port P2
- 0xFFE4 : Port P1

Mise en œuvre d'une interruption

Trois étapes pour mettre en œuvre une interruption :

- Autoriser l'interruption qui nous intéresse
- Préciser comment cette interruption doit fonctionner
- Autoriser globalement les interruptions



Syntaxe des routines d'interruptions en C



```
#pragma vector=NUMERO_DU_VECTEUR  
  
__interrupt void Nom_de_la_routine (void) {  
  
    ...  
  
}
```

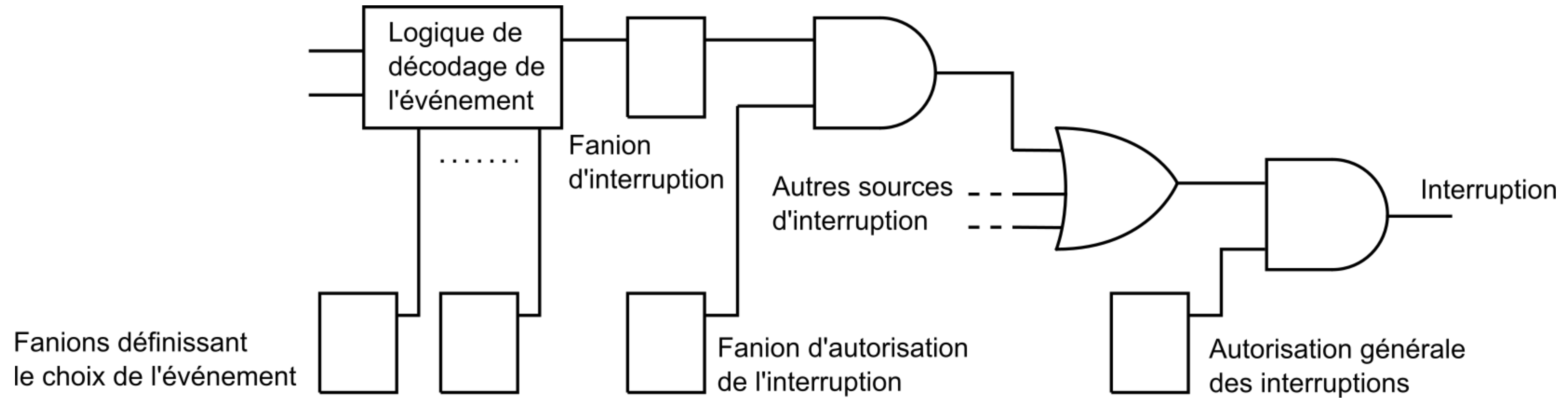


Interruption sur une entrée

- **P1DIR** entrée ou sortie
- **P1OUT** valeur de sortie
- **P1IN** valeur des entrées (*lecture*)
- **P1REN** résistance de tirage (*pull-up ou pull-down*)

- **P1IE** *Interrupt Enable* : autorisation de l'interruption
- **P1IES** *Interrupt Edge Select* : choix du flanc
- **P1IFG** *Interrupt FlaG* : les **fanions d'interruption**

Interruption sur une entrée



- **P1IE** *Interrupt Enable* : autorisation de l'interruption
- **P1IES** *Interrupt Edge Select* : choix du flanc
- **P1IFG** *Interrupt FlaG* : les **fanions d'interruption**

Interruption sur une entrée



```
1  int main() {
2      WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
3      P1DIR |= (1<<6); // Led verte en sortie
4      P1OUT |= (1<<3); P1REN |= (1<<3); //pull-up sur l'entrée P1.3
5
6      P1IES |= (1<<3); // Sur le flanc descendant
7      P1IE |= (1<<3); // Interruption P1 activée sur le bit 3
8      P1IFG &=~(1<<3); // Fanion d'interruption remis à zéro
9      __enable_interrupt(); // General Interrupt Enable
10
11     while(1) { // il n'y a rien à faire dans la boucle principale !
12     }
13 }
```



Interruption sur une entrée

```
14 // Routine d'interruption associée au Port P1
15
16 // Syntaxe spécifique pour les interruptions :
17 #pragma vector=PORT1_VECTOR
18 __interrupt void Port1_ISR(void) {
19
20 // Fanion d'interruption correspondant au bit 3 remis à 0 :
21 P1IFG &= ~(1<<3)
22
23 P1OUT ^= (1<<6); // inverse P1.6 (LED verte)
24 }
```




Interruption sur deux entrées, avec discrimination

```
1  int main() {
2      ...
3      P1IES &=~((1<<3)|(1<<4)); // Flancs montants
4      P1IE |= (1<<3)|(1<<4); // Interruption activée sur 2 entrées
5      P1IFG &=~((1<<3)|(1<<4)); // Fanions d'interruption remis à 0
6      ...
7
8  #pragma vector=PORT1_VECTOR
9  __interrupt void Port1_ISR(void) {
10     // discrimination des causes possible de l'interruption :
11     if (P1IFG & (1<<3)) { P1IFG &= ~(1<<3); ... ;}
12     if (P1IFG & (1<<4)) { P1IFG &= ~(1<<4); ... ;}
13 }
```



Interruption sur une fin de conversion AD

```
1  int main() {
2      WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
3      P1DIR |= (1<<6); P1OUT &=~(1<<6); // LED verte en sortie
4      // Activation du convertisseur ADC 10 bits (ADC10) :
5      ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // Interrupt enable
6      ADC10CTL1 = INCH_1; // Canal 1 = entrée A1 = P1.1
7      ADC10AE0 |= (1<<1); // Autorisation de l'entrée A1
8      __enable_interrupt(); // General Interrupt Enable
9      ADC10CTL0 |= ENC + ADC10SC; // lance une première conversion
10
11     while(1) { // il n'y a rien à faire dans la boucle principale !
12     }
13 }
```



Interruption sur une fin de conversion AD

```
14 // Routine d'interruption associée à la fin de conversion ADC
15 #pragma vector=ADC10_VECTOR
16 __interrupt void ADC10_ISR(void) {
17
18     uint16_t val = ADC10MEM; // lit le résultat de la conversion
19     ADC10CTL0 |= ENC + ADC10SC; // lance la conversion suivante
20
21     if (val > 511) { // La LED verte montre si la valeur dépasse Vcc/2
22         P10OUT |= (1<<6); // LED verte On
23     } else {
24         P10OUT &=~(1<<6); // LED verte Off
25     }
26 }
```

- Principe des interruptions
- Événements produisant des interruptions (externes ou internes)
- Mise en œuvre
- Deux exemples (interruption sur une entrée et sur une fin de conversion)