

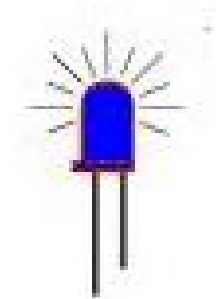
roboonly

The background features a collage of technical and programming elements. On the left, a printed wiring board (PCB) layout is visible with labels like 'PRINTED WIRING BOARD 4050404', 'R32 10K', 'R33 22K', 'Q10 04-BSE', 'C12 20PF 500V', 'C13 300PF 500V', 'C14 10V', 'C15 300PF 500V', 'C16 10V', 'C17 10V', 'C18 10V', 'C19 10V', 'C20 10V', 'C21 10V', 'C22 10V', 'C23 10V', 'C24 10V', 'C25 10V', 'C26 10V', 'C27 10V', 'C28 10V', 'C29 10V', 'C30 10V', 'C31 10V', 'C32 10V', 'C33 10V', 'C34 10V', 'C35 10V', 'C36 10V', 'C37 10V', 'C38 10V', 'C39 10V', 'C40 10V', 'C41 0.1-555', 'C42 0.1-555', 'C43 0.1-555', 'C44 0.1-555', 'C45 0.1-555', 'C46 0.1-555', 'C47 0.1-555', 'C48 0.1-555', 'C49 0.1-555', 'C50 0.1-555', 'C51 0.1-555', 'C52 0.1-555', 'C53 0.1-555', 'C54 0.1-555', 'C55 0.1-555', 'C56 0.1-555', 'C57 0.1-555', 'C58 0.1-555', 'C59 0.1-555', 'C60 0.1-555', 'C61 0.1-555', 'C62 0.1-555', 'C63 0.1-555', 'C64 0.1-555', 'C65 0.1-555', 'C66 0.1-555', 'C67 0.1-555', 'C68 0.1-555', 'C69 0.1-555', 'C70 0.1-555', 'C71 0.1-555', 'C72 0.1-555', 'C73 0.1-555', 'C74 0.1-555', 'C75 0.1-555', 'C76 0.1-555', 'C77 0.1-555', 'C78 0.1-555', 'C79 0.1-555', 'C80 0.1-555', 'C81 0.1-555', 'C82 0.1-555', 'C83 0.1-555', 'C84 0.1-555', 'C85 0.1-555', 'C86 0.1-555', 'C87 0.1-555', 'C88 0.1-555', 'C89 0.1-555', 'C90 0.1-555', 'C91 0.1-555', 'C92 0.1-555', 'C93 0.1-555', 'C94 0.1-555', 'C95 0.1-555', 'C96 0.1-555', 'C97 0.1-555', 'C98 0.1-555', 'C99 0.1-555', 'C100 0.1-555'. In the center, there are 3D models of electronic components like a blue integrated circuit, a white component, and a red component. On the right, a keyboard with glowing keys is visible. Overlaid on the image are snippets of C++ code: 

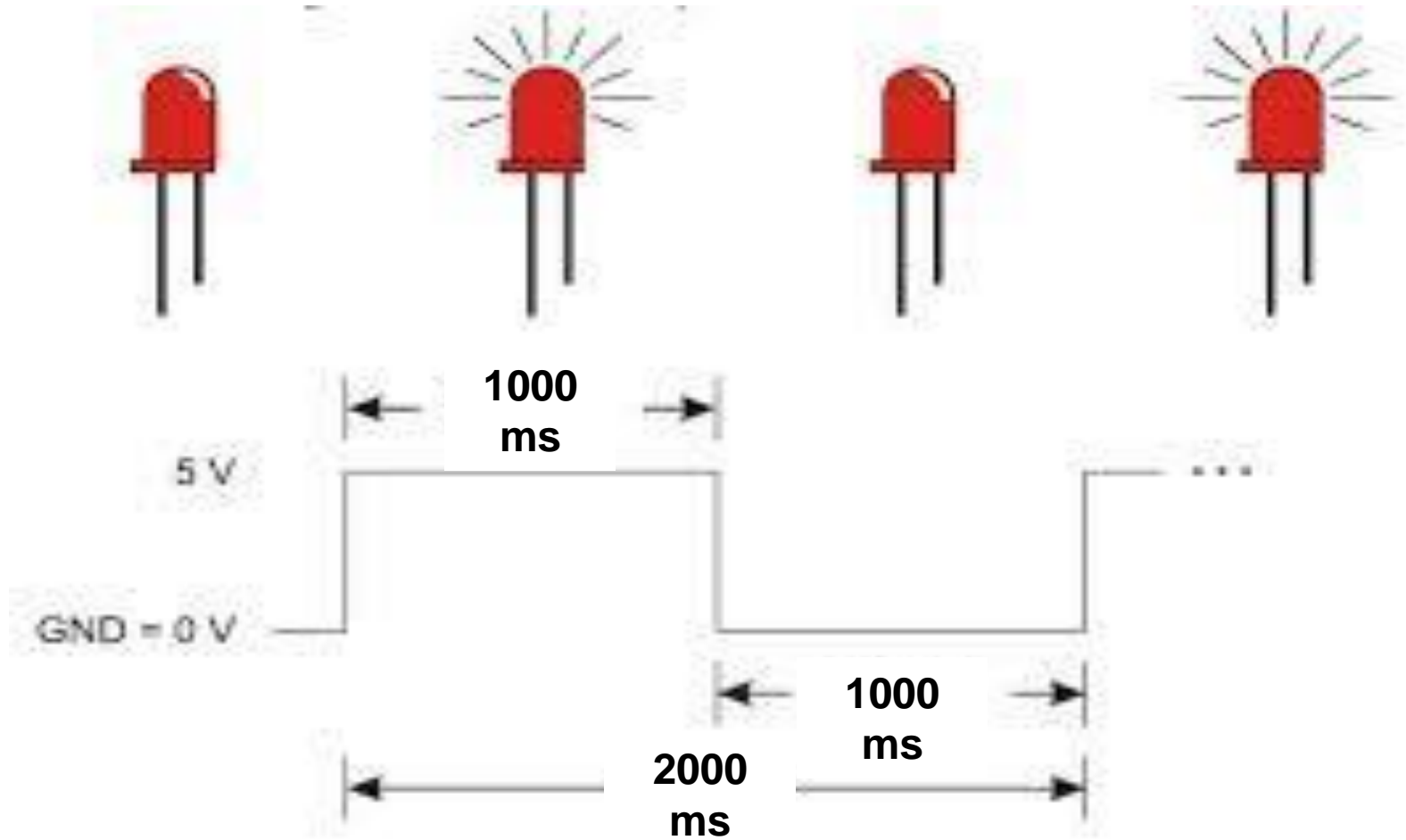
```
begin(9600);  
LED, OUTPUT);  
  
int potentiometer and button values  
begin(9600);  
LED, OUTPUT);  
  
data, : %ld\nD\nprintln(data);  
  
te(LED, digitalRead(LED));  
second
```

## INTERRUPTIONS ET TIMERS

# Systeme d'alarme



# Blinking LED (2 secondes)



# Blinking LED (2 secondes)

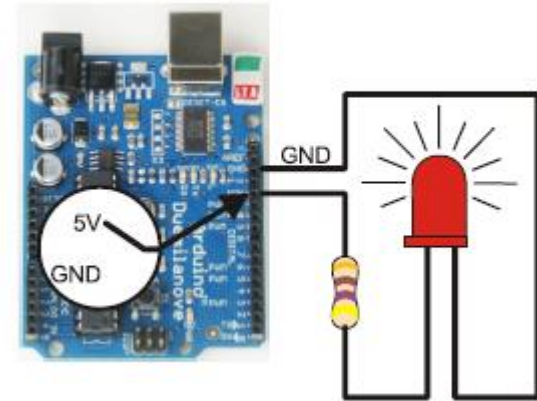


```
#define LED_PIN 13;

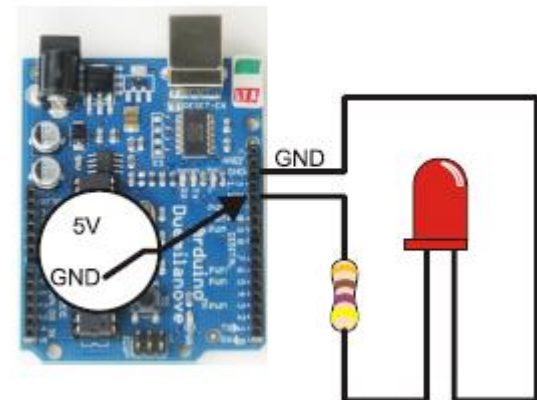
void setup()
{
  pinMode(LED_BLINK_PIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_BLINK_PIN, HIGH);
  delay(1000);
  digitalWrite(LED_BLINK_PIN, LOW);
  delay(1000);
}
```

```
digitalWrite(13, HIGH);
```



```
digitalWrite(13, LOW);
```



# État de la Porte

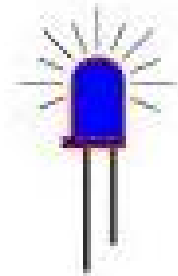


Capteur = 0



Capteur = 1

# État de la Porte: LED Bleu



# Blinking LED + État de la Porte

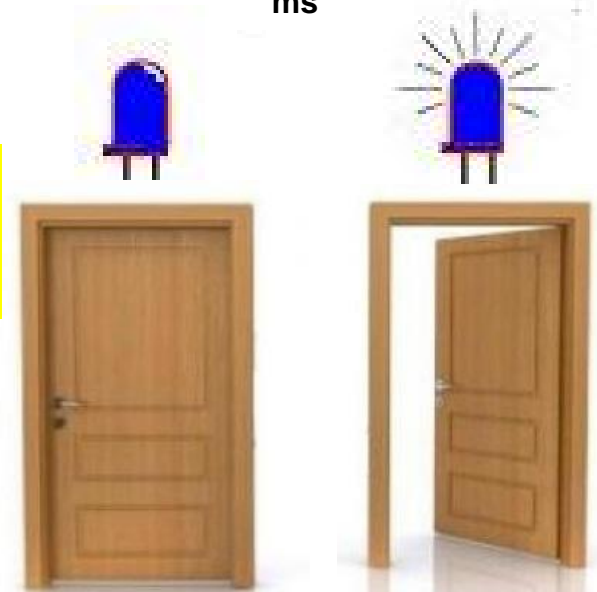
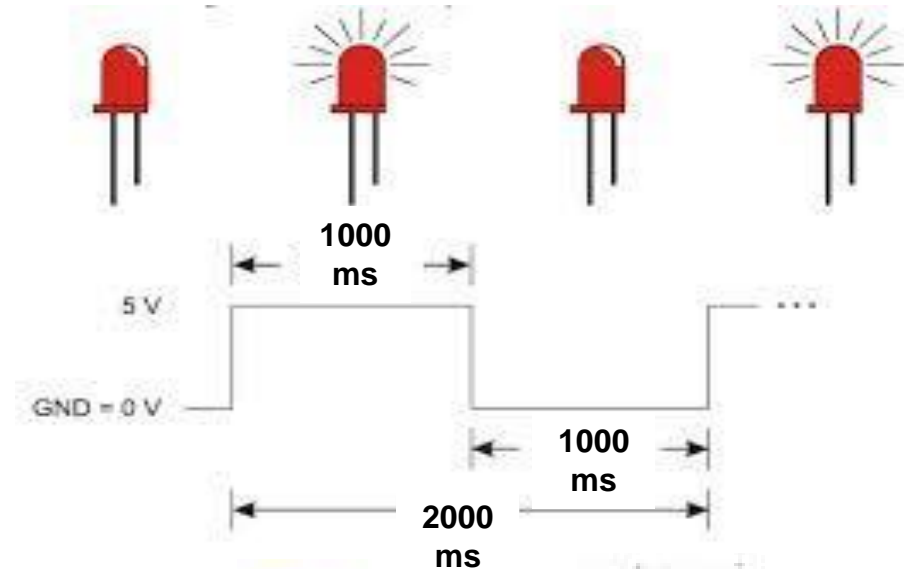


```
void setup()  
{  
  pinMode(LED_BLINK_PIN, OUTPUT);  
  pinMode(LED_PORTE_PIN, OUTPUT);  
  pinMode(PORTE_PIN, INPUT);  
}
```

```
void loop()  
{  
  digitalWrite(LED_BLINK_PIN, HIGH);  
  if (digitalRead(PORTE_PIN))  
    digitalWrite(LED_PORTE_PIN, HIGH);  
  else  
    digitalWrite(LED_PORTE_PIN, LOW);  
  delay(1000);
```

```
  digitalWrite(LED_BLINK_PIN, LOW);  
  if (digitalRead(PORTE_PIN))  
    digitalWrite(LED_PORTE_PIN, HIGH);  
  else  
    digitalWrite(LED_PORTE_PIN, LOW);  
  delay(1000);  
}
```

**1 s. pour les voleurs !!!**



# État de la porte plus souvent



```
void setup()
{
  pinMode(LED_BLINK_PIN, OUTPUT);
  pinMode(LED_PORTE_PIN, OUTPUT);
  pinMode(PORTE_PIN, INPUT);
}
```

```
void loop()
```

```
{
  digitalWrite(LED_BLINK_PIN, HIGH);
  if (digitalRead(PORTE_PIN) == HIGH)
    digitalWrite(LED_PORTE_PIN, HIGH);
  else
    digitalWrite(LED_PORTE_PIN, LOW);
  delay(1000);
}
```

```
digitalWrite(LED_BLINK_PIN, HIGH);

if (digitalRead(PORTE_PIN) == HIGH)
  digitalWrite(LED_PORTE_PIN, HIGH);
else
  digitalWrite(LED_PORTE_PIN, LOW);

digitalWrite(LED_BLINK_PIN, LOW);
if (digitalRead(PORTE_PIN) == HIGH)
  digitalWrite(LED_PORTE_PIN, HIGH);
else
  digitalWrite(LED_PORTE_PIN, LOW);
delay(500);
digitalWrite(LED_BLINK_PIN, LOW);
```

**PAS ESCALABLE !!!**

Retourne les délais

La LED clignote plus vite !

```
digitalWrite(LED_BLINK_PIN, LOW);
if (digitalRead(PORTE_PIN) == HIGH)
  digitalWrite(LED_PORTE_PIN, HIGH);
else
  digitalWrite(LED_PORTE_PIN, LOW);
delay(1000);
}
```



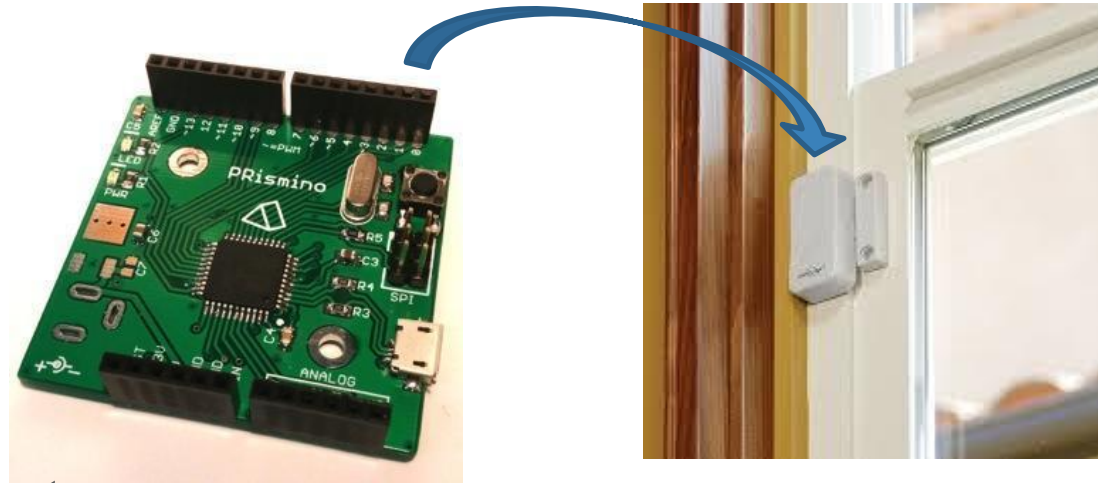
# Contrôler un capteur ?

```
while (1)
```

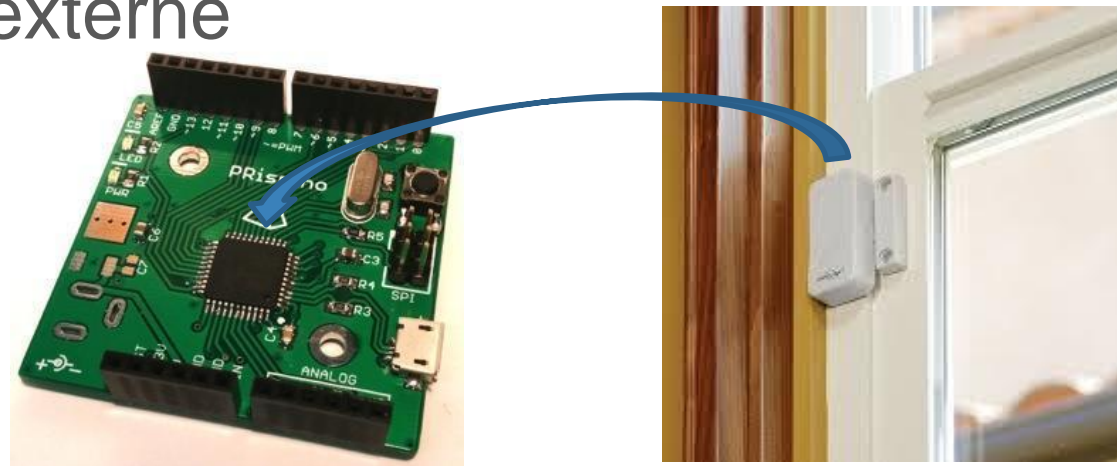
```
    valeur = digitalRead(pin_capteur);
```

- Deux possibilités

- Polling



- Interruption externe



# Interruptions, the Arduino way.

## attachInterrupt (Prismino ~= Leonardo)

```
void setup()
{
  pinMode(LED_BLINK_PIN, OUTPUT);
  pinMode(LED_PORTE_PIN, OUTPUT);
  pinMode(PORTE_PIN, INPUT);
  attachInterrupt(PORTE_PIN, verifierPorte, CHANGE);
}
```

**CALLBACK** function: la fonction qu'on appelle quand l'interruption arrive.

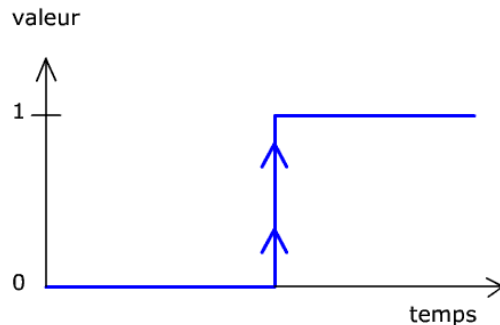
```
void loop()
{
  digitalWrite(LED_BLINK_PIN, HIGH);
  delay(500);
  digitalWrite(LED_BLINK_PIN, LOW);
  delay(500);
}
```

```
void verifierPorte()
{
  if (digitalRead(PORTE_PIN))
    digitalWrite(LED_PORTE_PIN, HIGH);
  else
    digitalWrite(LED_PORTE_PIN, LOW);
}
```

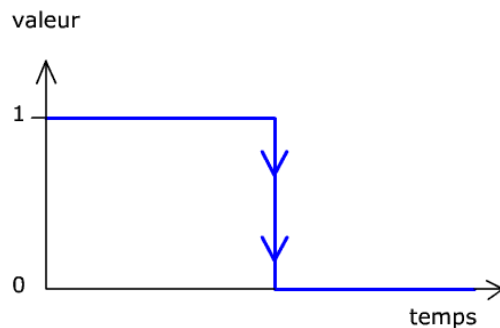
**CHANGE** to trigger the interrupt whenever the pin changes value  
**RISING** to trigger when the pin goes from low to high,  
**FALLING** for when the pin goes from high to low.

**LOW** to trigger the interrupt whenever the pin is low  
**HIGH** to trigger the interrupt whenever the pin is high. (*Arduino Due only*)

# Les flancs



- de 0 à 1 = flanc montant  
**La porte s'ouvre !**



- de 1 à 0 = flanc descendant  
**La porte se ferme !**

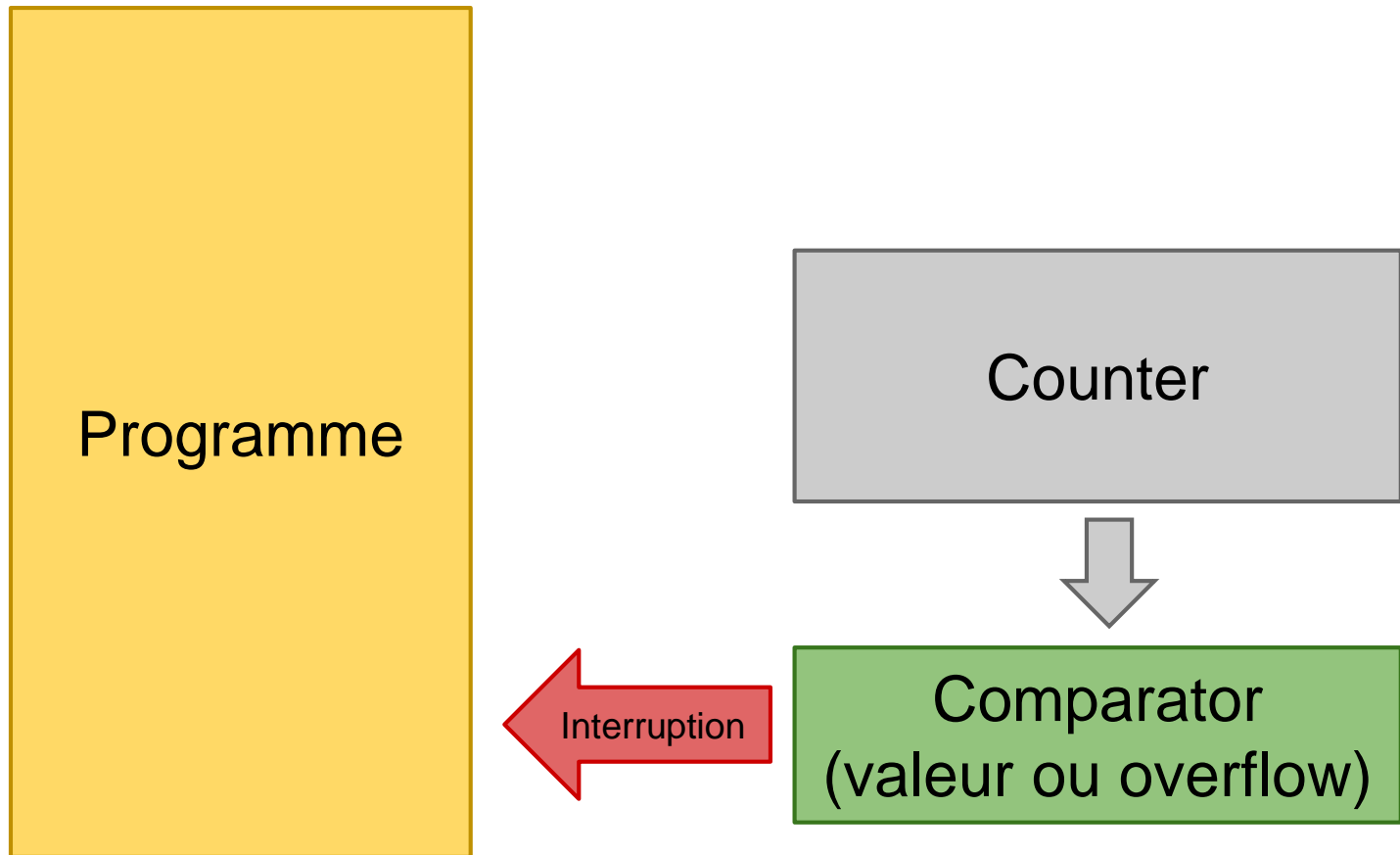
## Interruptions, pourquoi?

Comment faire  
plusieurs  
choses à la fois?

# Une autre interruption : Les Timers



# Compter le temps : le Timer



# Interruptions, types

- External
  - Viens d'un pin externe
  - “Je veux faire quelque chose quand une valeur mesurée change”
  - `attachInterrupt(interrupt, fonction, quand)`
- Internal
  - Viens d'un périphérique interne e.g.: Timer
  - “Je veux faire quelque chose (**fonction**) dans **x** ms pour **n** fois”
  - `setTimer(fonction, x, n)`

# Blinking LED + État de la Porte



```
int ledState = LOW;
```

```
void setup()
```

```
{  
  pinMode(LED_BLINK_PIN, OUTPUT);  
  pinMode(LED_PORTE_PIN, OUTPUT);  
  pinMode(PORTE_PIN, INPUT);  
  setTimer(blinkLED, 5);  
}
```

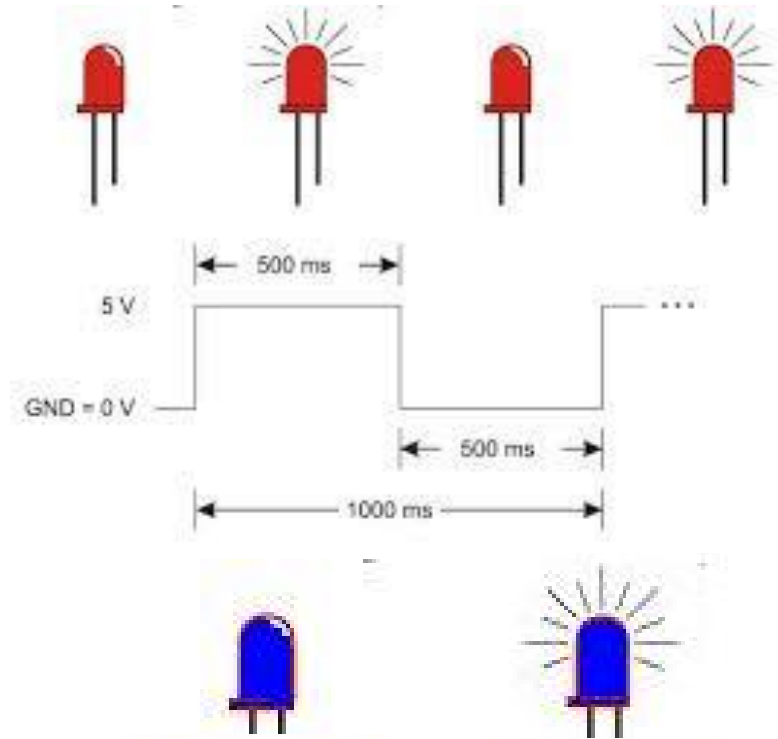
Appel la fonction chaque:  
 $5 * 100\text{ms} = 500\text{ms}$

```
void loop()
```

```
{  
  if (digitalRead(PORTE_PIN))  
    digitalWrite(LED_PORTE_PIN, HIGH);  
  else  
    digitalWrite(LED_PORTE_PIN, LOW);  
}
```

```
void blinkLED()
```

```
{  
  if (ledState == LOW)  
    ledState = HIGH;  
  else  
    ledState = LOW;  
  digitalWrite(LED_BLINK_PIN, ledState);  
}
```





# Les Timers: Librairie Robopoly

La librairie robopoly contient des fonctions toute faites pour les utiliser plus facilement: <https://github.com/Robopoly/prismino-library>

```
int8_t setTimer( func_t callbackFunction,  
                uint16_t interval );
```

Appel la fonction “*callbackFunction*”, chaque: “*interval*”\*100ms.

```
int8_t setTimer( func_t callbackFunction,  
                uint16_t interval,  
                uint8_t callNumber = 0);
```

Appel la fonction “*callbackFunction*”, chaque: “*interval*”\*100ms, “*callNumber*” fois.

# Quelles Interruptions ???

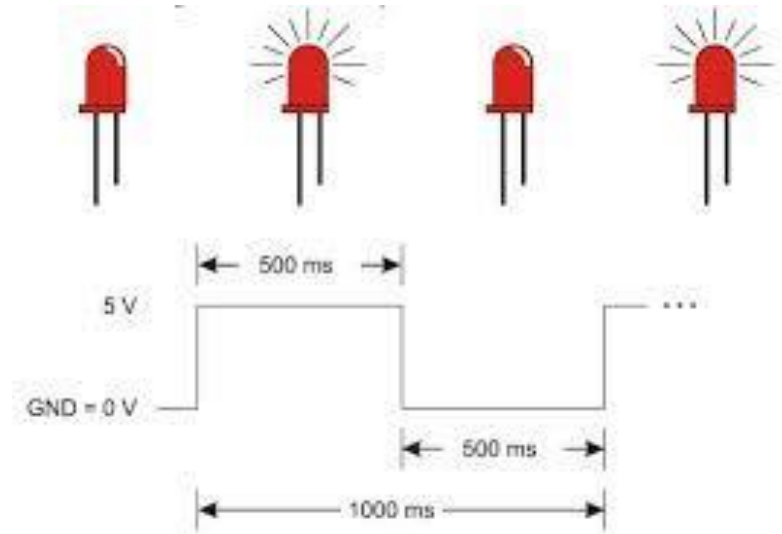


```
void setup()
{
  attachInterrupt(PORTE_PIN, verifierPorte, CHANGE);
  setTimer(blinkLED, 5);
}
```

```
void loop()
{
  verifierPorte?
  blinkLED?
}
```

```
void blinkLED()
{
  ...
}
```

```
void verifierPorte()
{
  ...
}
```



# Autres fonctions pour les interruptions dans le Arduino



`attachInterrupt(interrupt, ISR, mode)`  
`attachInterrupt(pin, ISR, mode)`

`detachInterrupt(interrupt)`  
`detachInterrupt(pin)`

`noInterrupts()`  
`interrupts()`

# Happy Interrupting !

## FIN

Questions?

# Prochains événements

**Lundi 1 décembre**

Caméra linéaire

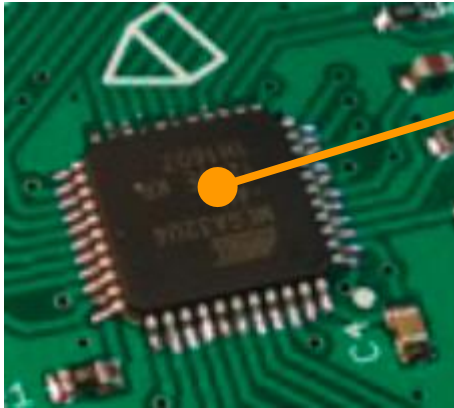
**Lundi 8 décembre**

Présentation des règles grand concours

**28.3.15 (à confirmer)**

Grand concours !

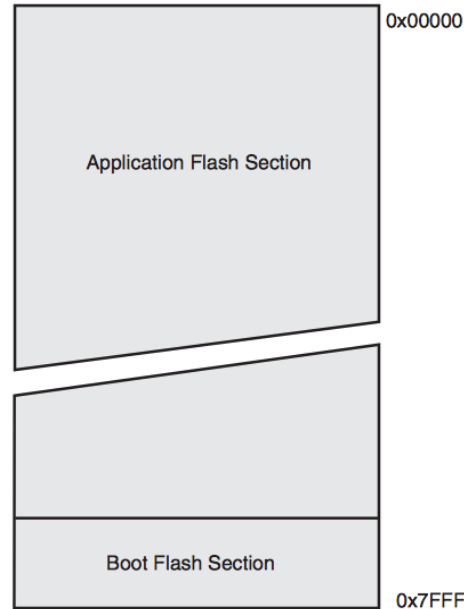
# Il y a quoi dedans?



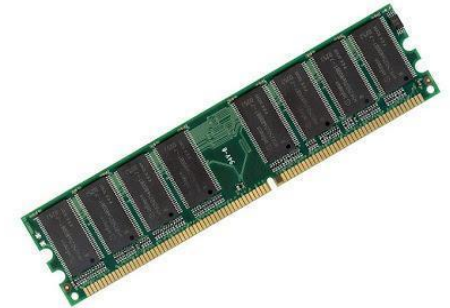
Hard disk



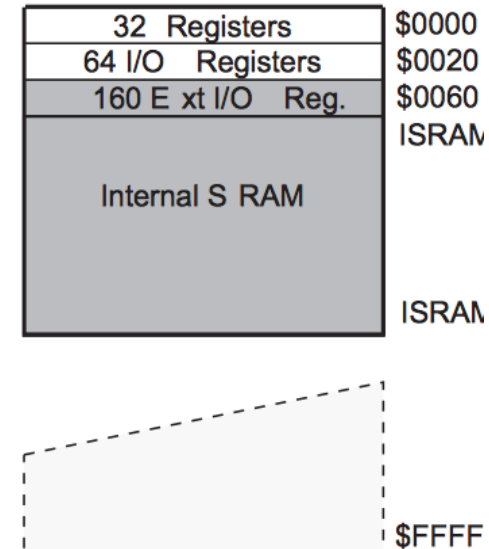
Program Memory



RAM



Data Memory



Ou, les datasheets

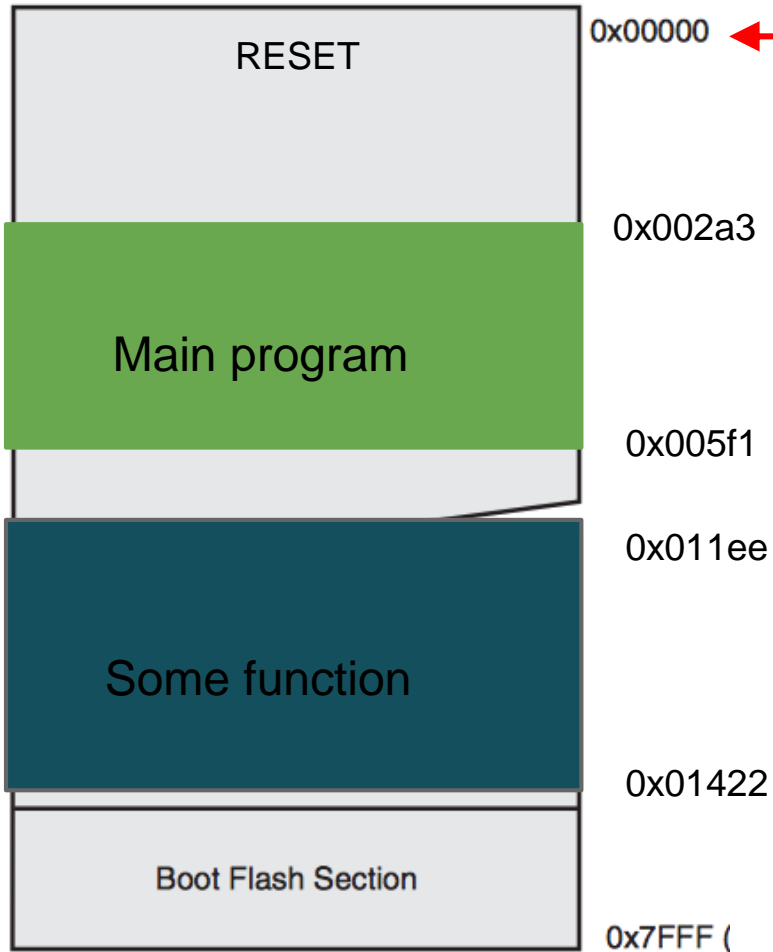
# Mon PC

- N'est **pas** un Personal Computer
- Plutôt, un Program Counter
- Marque/compte quelle ligne de code/instruction est à exécuter
- Au reset, PC = \$0x0000
- \$ → Adresse, 0x → hexadecimal



# Memoire programme

Program Memory



PC = 0x00000

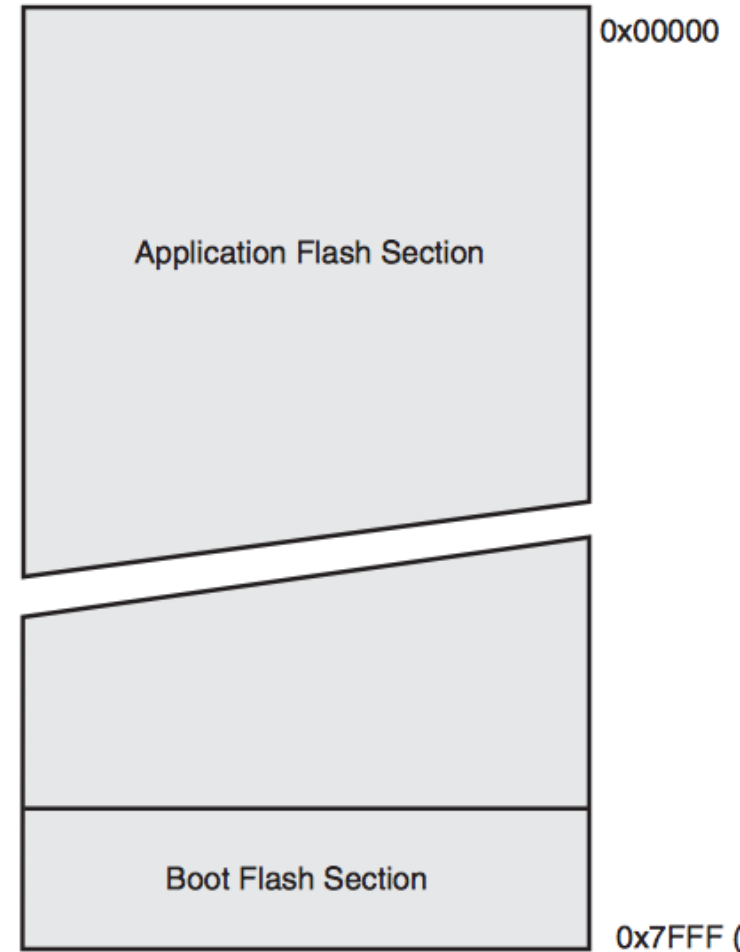


Contient l'adresse du debut du programme

Data

Address

Program Memory

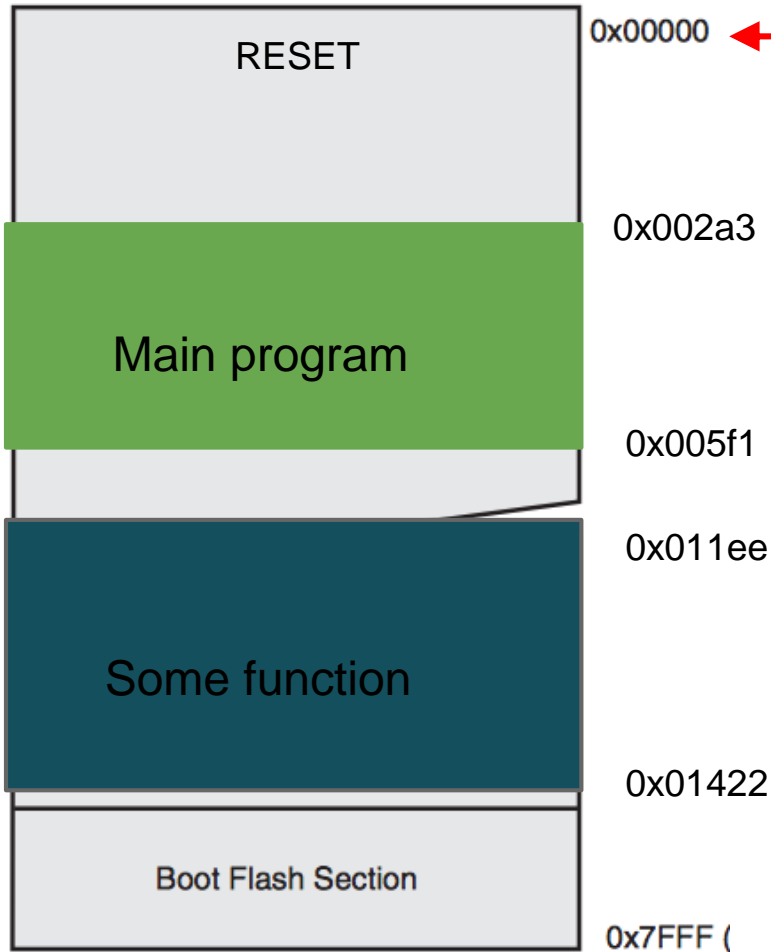




# Memoire programme



Program Memory



PC = 0x00000

0x00000



Contient  
l'adresse du  
debut du  
programme

On focalise  
dans la  
memoire de  
programme

(memoire  
d'instructions)

# Memoire programme

Program Memory

PC = 0x002a3

RESET

0x00000

Main program

0x002a3 ←

0x005f1

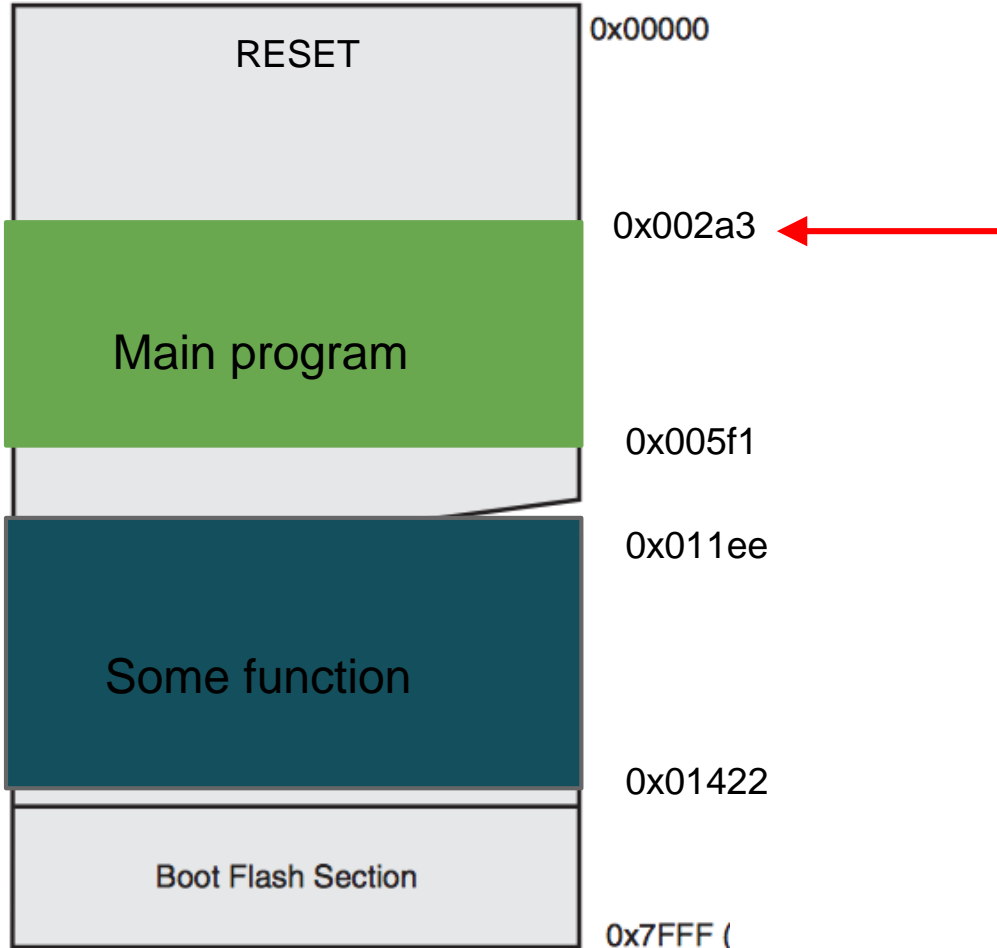
Some function

0x011ee

0x01422

Boot Flash Section

0x7FFF (



# Memoire programme

Program Memory

PC = 0x002a4

RESET

0x00000

Main program

0x002a3

0x005f1

Some function

0x011ee

0x01422

Boot Flash Section

0x7FFF (



# Memoire programme

Program Memory

PC = 0x002a5

RESET

0x00000

Main program

0x002a3

0x005f1

Some function

0x011ee

0x01422

Boot Flash Section

0x7FFF (



# Memoire programme

Program Memory

PC = 0x00...

RESET

0x00000

Main program

0x002a3

...

0x005f1

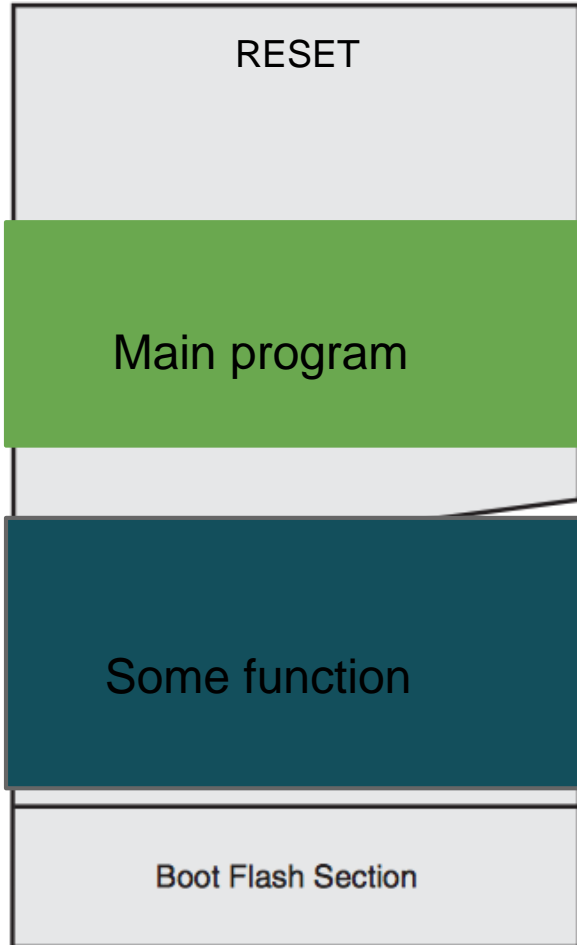
Some function

0x011ee

0x01422

Boot Flash Section

0x7FFF (



# Memoire programme

Program Memory

PC = 0x0042c

RESET

0x00000

Main program

0x002a3

0x005f1



Appel fonction!!

Some function

0x011ee

0x01422

Boot Flash Section

0x7FFF (

# Memoire programme

Program Memory

PC = 0x011ee

RESET

0x00000

Main program

0x002a3

0x005f1

Some function

0x011ee

0x01422

Boot Flash Section

0x7FFF (



# Memoire programme

Program Memory

PC = 0x0....

RESET

0x00000

Main program

0x002a3

0x005f1

Some function

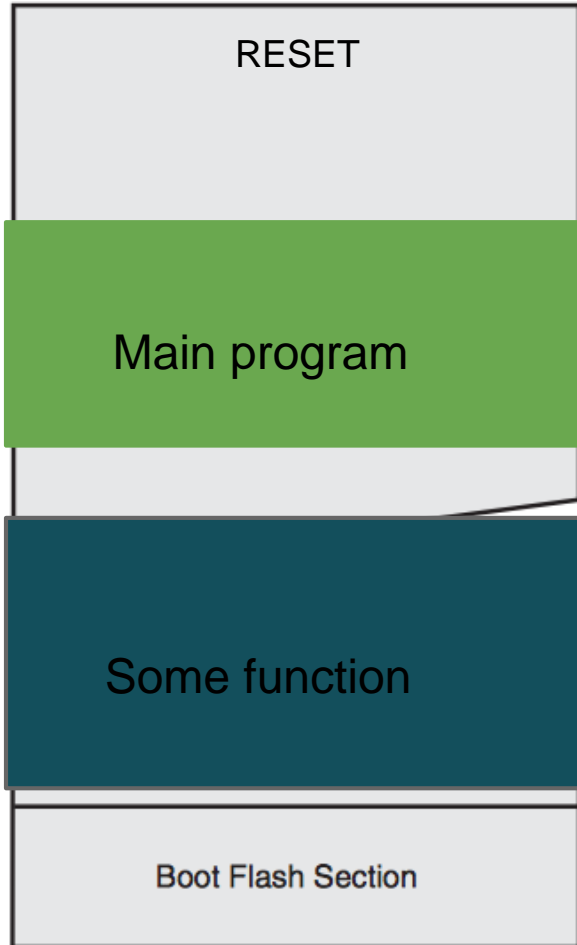
0x011ee

...

0x01422

Boot Flash Section

0x7FFF (





# Memoire programme

Program Memory

PC = 0x01422

RESET

0x00000

Main program

0x002a3

0x005f1

Some function

0x011ee

0x01422

Boot Flash Section

0x7FFF (



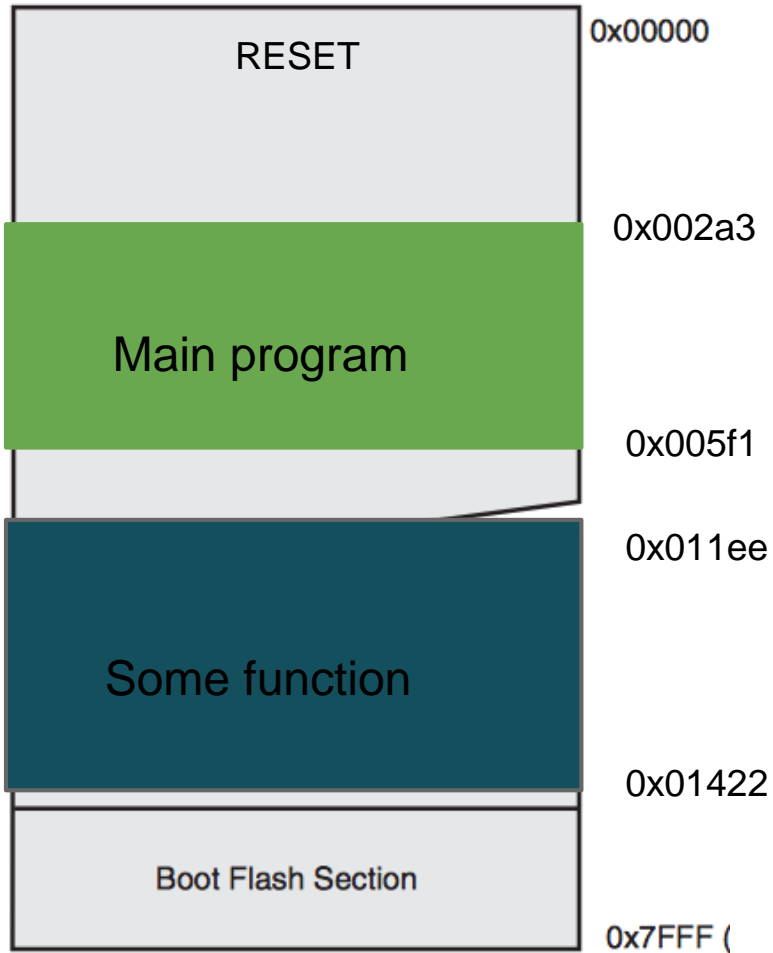
# Memoire programme



Program Memory

PC = 0x01422

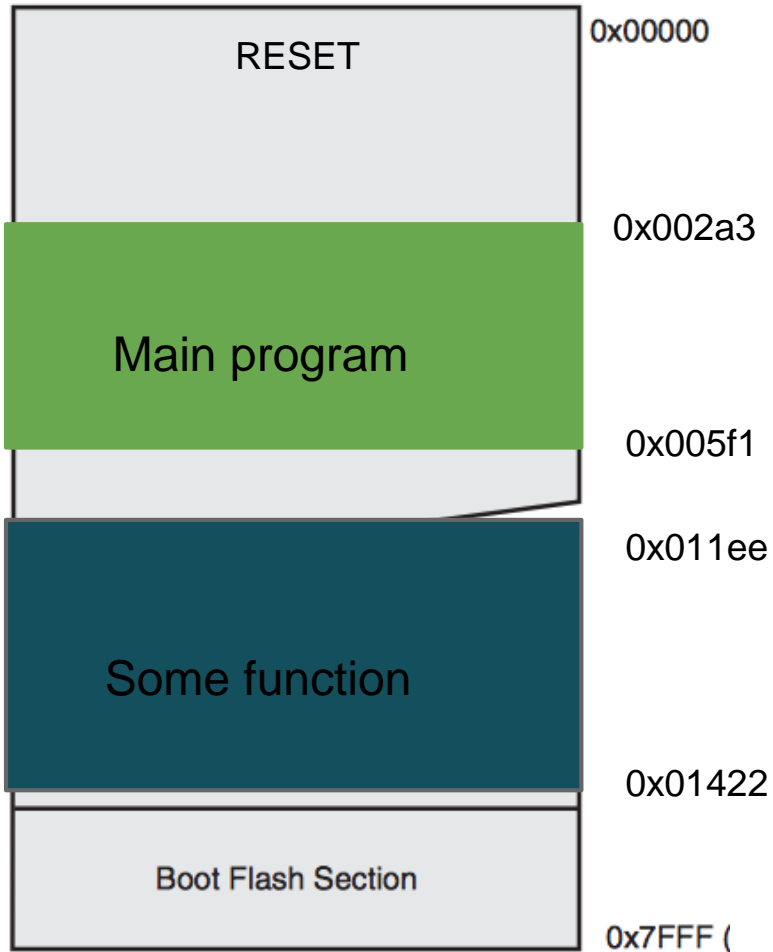
Fonction  
appelé avec:  
PC = 0x0042c



# Memoire programme



Program Memory



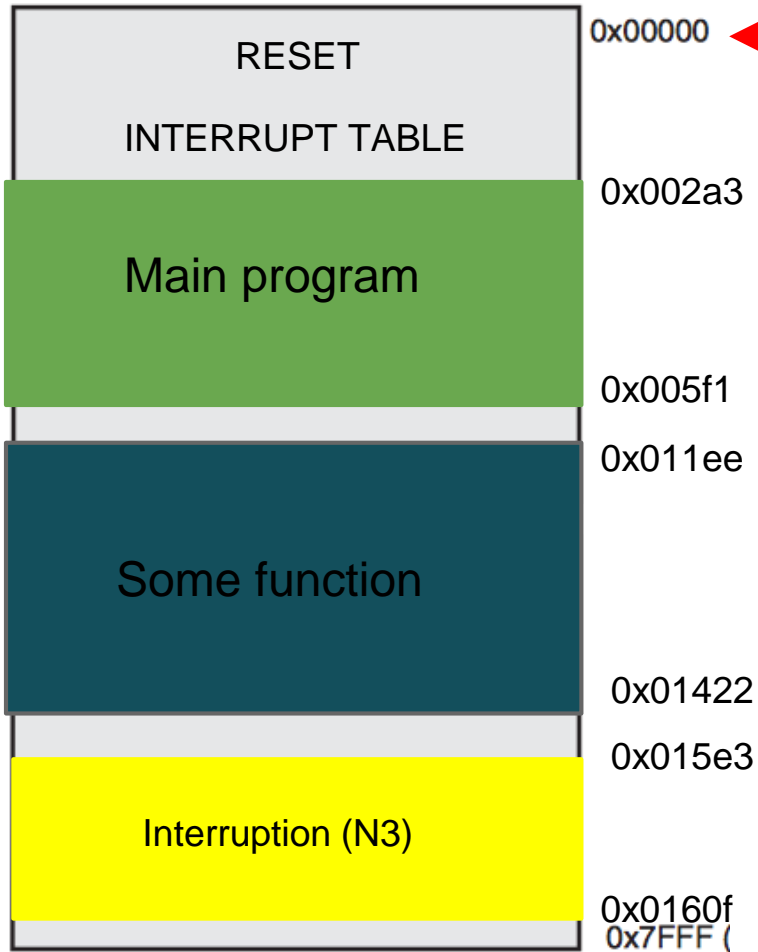
PC = 0x0042d

Fonction  
appelé avec:  
PC = 0x0042c

Et avec des  
interruptions...

# Memoire programme - Interruption!

Program Memory



PC = 0x00000

0x00000



Contient  
l'adresse du  
debut du  
programme

0x002a3

0x005f1

0x011ee

0x01422

0x015e3

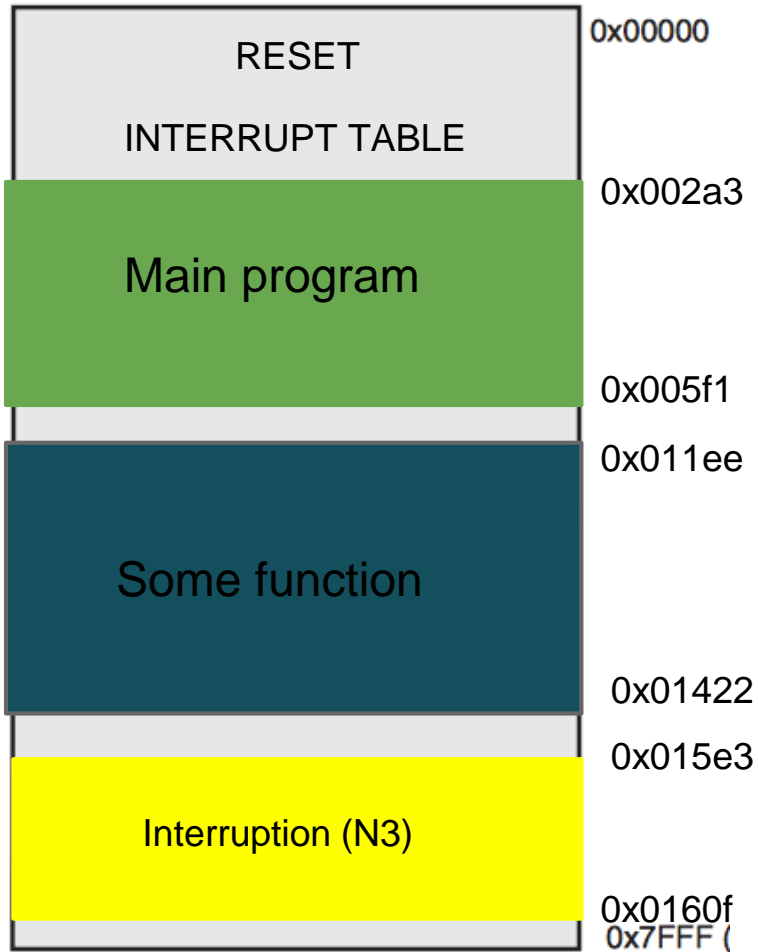
0x0160f

0x7FFF (

# Memoire programme - Interruption!

Program Memory

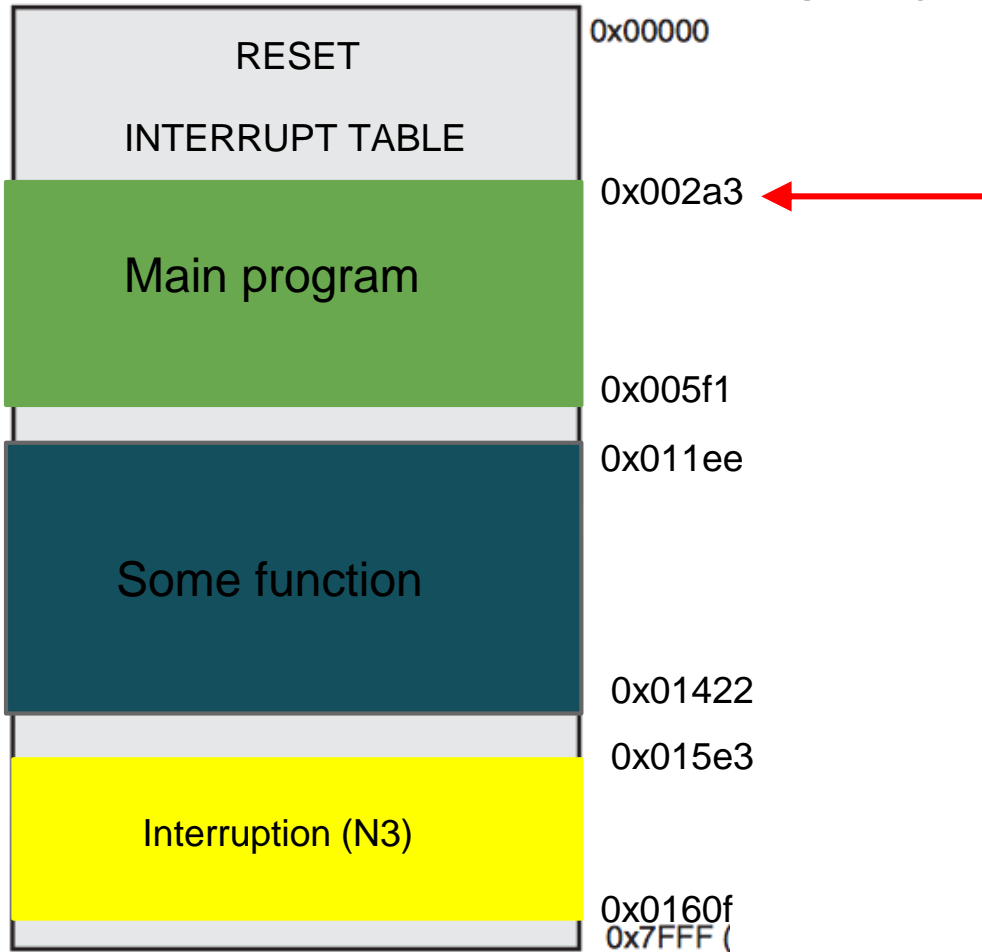
PC = 0x002a3



# Memoire programme - Interruption!

Program Memory

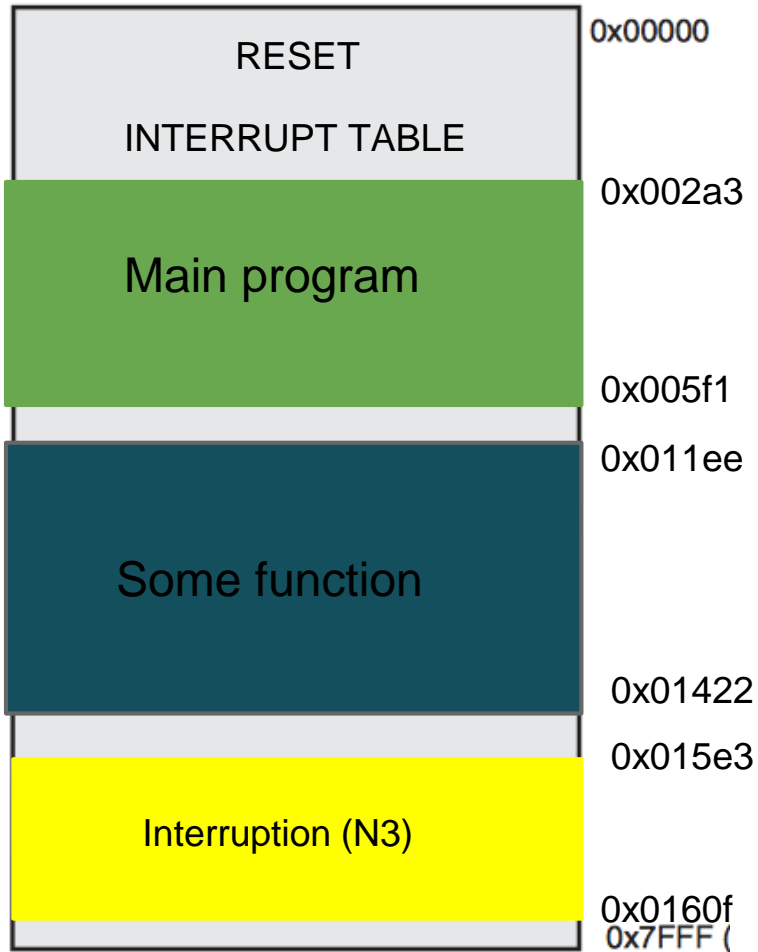
PC = 0x002a4



# Memoire programme - Interruption!

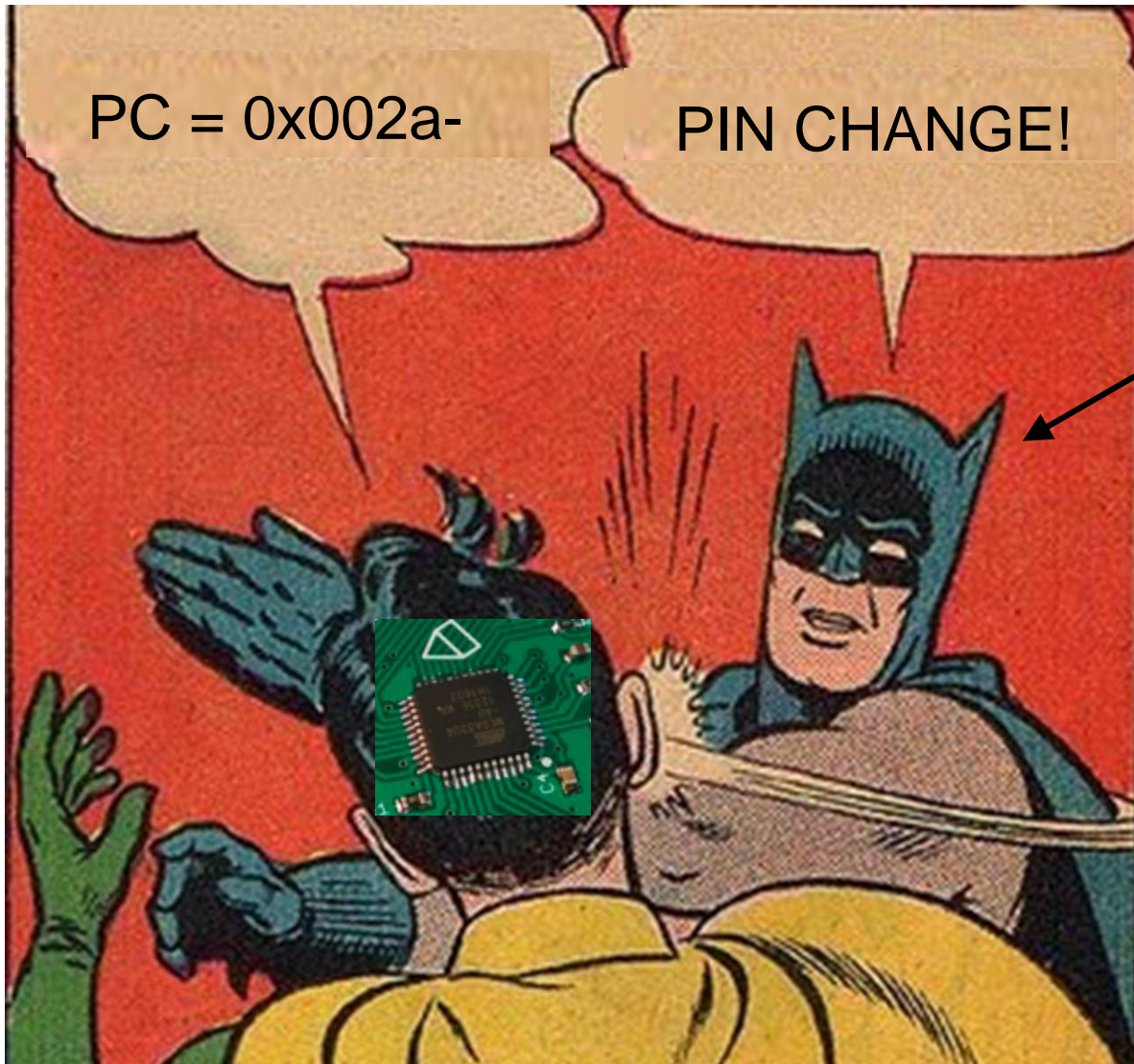
Program Memory

PC = 0x002a5





# Memoire programme - Interruption!



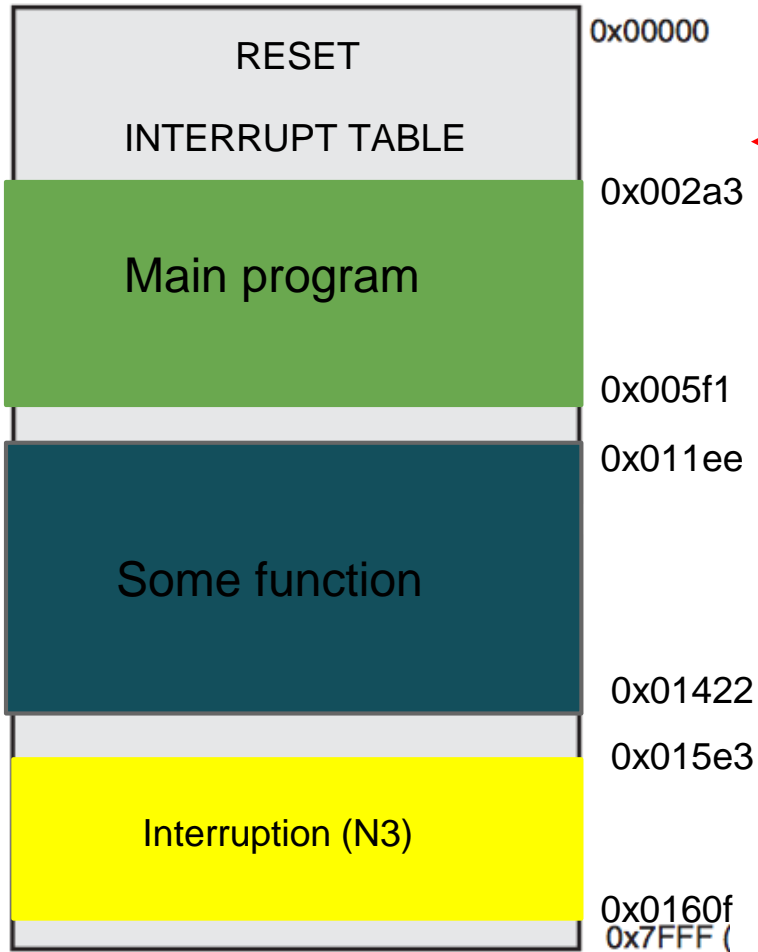
External  
World

Dans ce  
cas, on  
admet c'est  
l'interruption  
numero 3

# Memoire programme - Interruption!

Program Memory

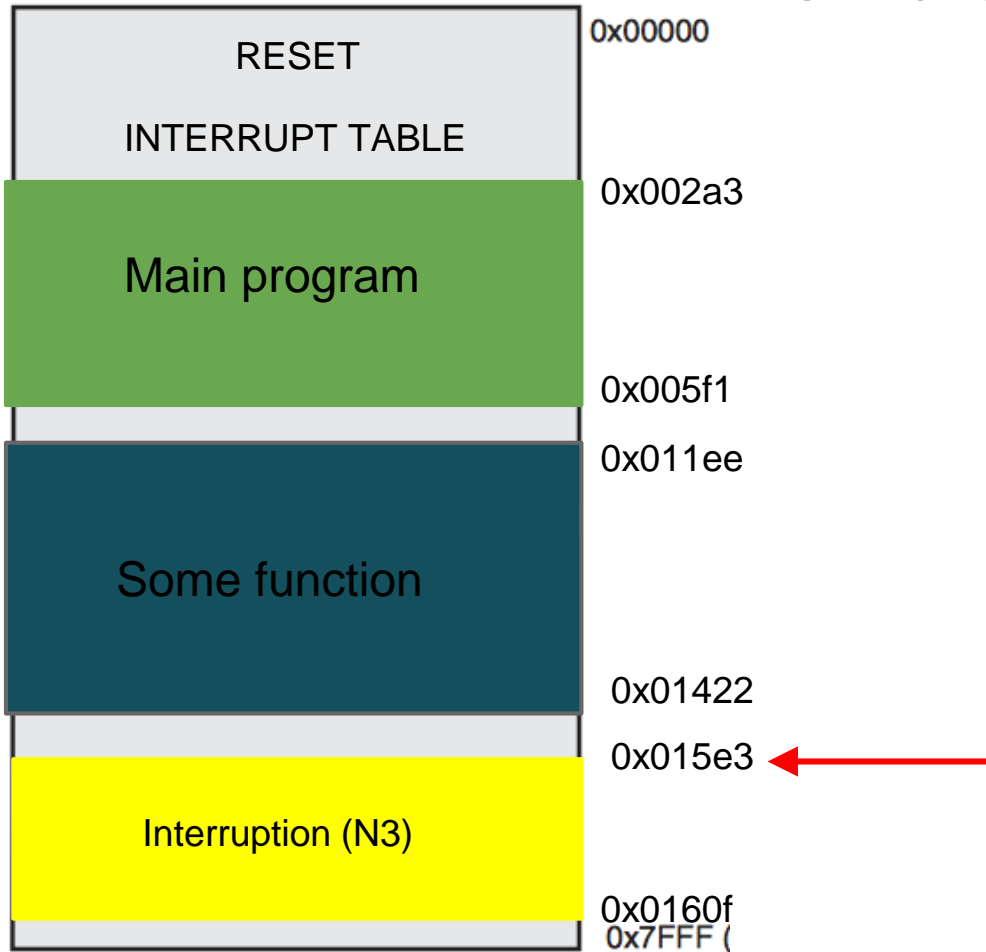
PC = 0x00008



# Memoire programme - Interruption!

Program Memory

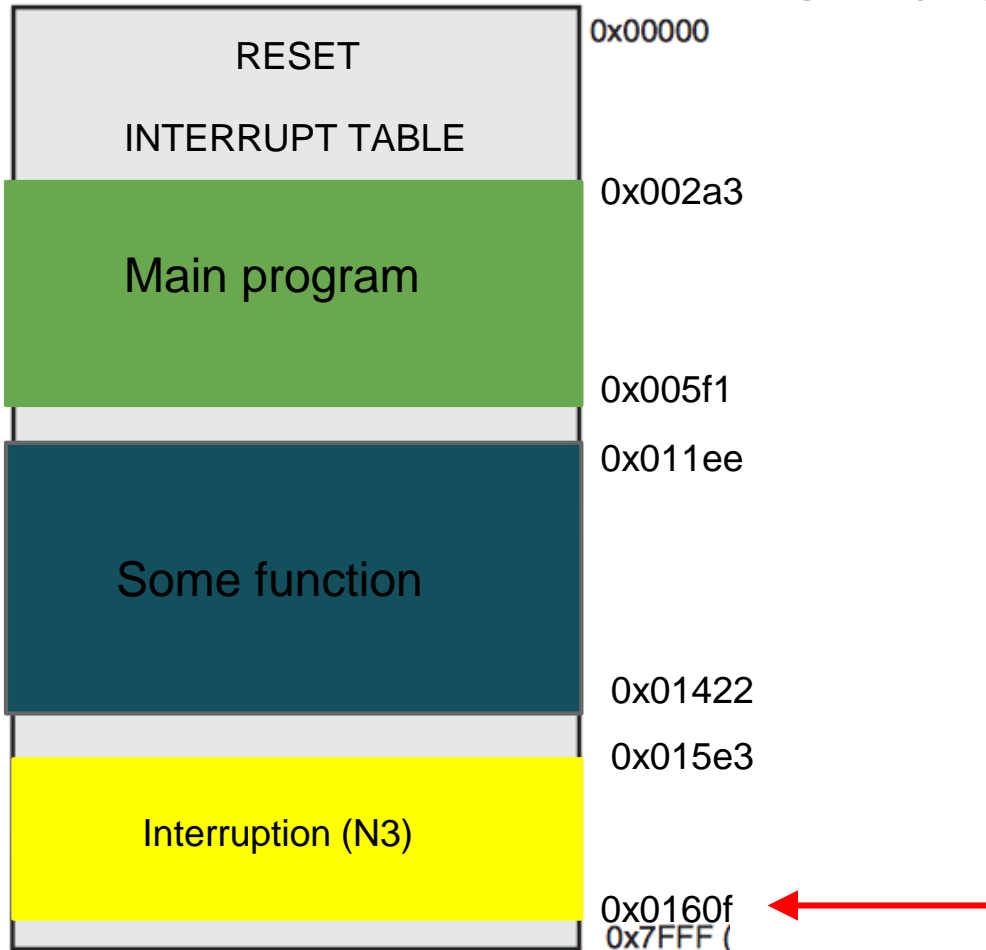
PC = 0x015e3



# Memoire programme - Interruption!

Program Memory

PC = 0x0160f



# Memoire programme - Interruption!

Program Memory

PC = 0x002a6

