POLYTECH SORBONNE

Interruptions : Introduction

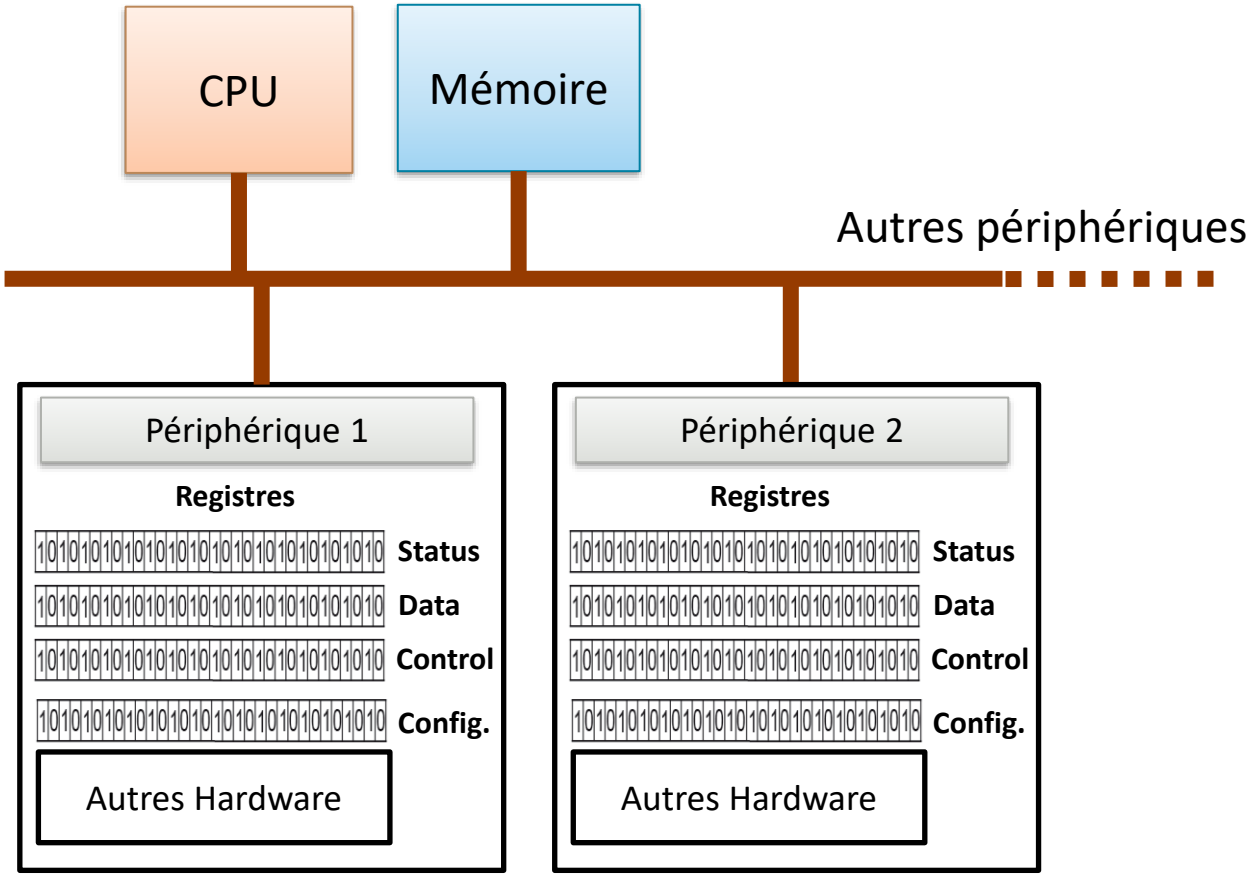
Arouna DARGA

Maître de Conférence, Sorbonne Université,
GeePs CentraleSupelec

Plan

- Introduction
- Scrutation VS Interruption
- Interruption?
  - Définition
  - Les sources
  - Contrôleur d'interruption NVIC
  - Déroulement d'une interruption
  - Latence d'une interruption

Introduction



Les périphériques :

- ✓ Ajoute des fonctionnalités au MCU
  - ✓ Exemple: TIMERS, GPIO
- ✓ Les bus (data, adresses) relient Périphériques, processeur mémoire
- ✓ Peuvent travailler de façon :
  - ✓ **autonome** sans intervention du CPU
  - ✓ Avec intervention du CPU
  - ✓ En association avec d'autres périphériques.
- ✓ Les périphériques sont contrôlés par activation/désactivation des fonctionnalités à travers des registres (control) (registers).

Introduction

Méthodes d'accès aux périphériques

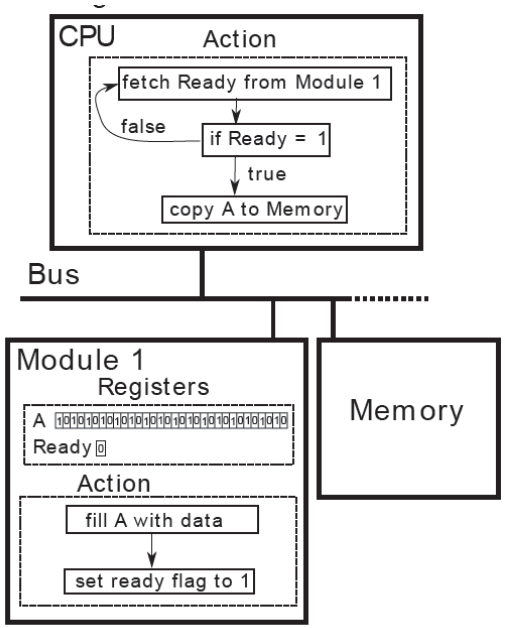
- ✓ Du point de vue du microprocesseur:
  - ✓ l'accès aux périphériques se fait de la même façon que l'accès aux données de la mémoire :
  - ✓ des instructions demandent au microprocesseur de lire ou d'écrire sur un périphérique.
- ✓ l'accès aux périphériques se fait avec les mêmes instructions que les accès à la mémoire
- ✓ les périphériques ont seulement des adresses différentes. On dit alors que les périphériques sont "mappés" en mémoire, c'est-à- dire qu'une plage d'adresse leur est attribuée (MMIO = Memory Mapped I/O).

Il existe plusieurs techniques pour communiquer à partir du Microprocesseur/Mémoire vers un périphérique.
Les trois principales techniques sont :

- ✓ La scrutation
- ✓ Les interruptions
- ✓ **et le DMA (Direct Access Memory).**

Scrutation VS Interruption

Scrutation (polling)



Avantages :

- Simplicité dans certains cas

Inconvénients:

- Le CPU ne fait rien d'autres et consomme de l'énergie dans la phase d'attente: pas d'**Economie d'énergie**
- La gestions de plusieurs périphériques ou tâches peut être très compliquée.

Le CPU **scrute continuellement** le périphérique:

- Vérifie (if (bit bx)….) ou attend (while (bit bx){..}) le changement d'état d'un bit.
  - Le **bit bx** peut être un bit de **STATUS ou Flag** (drapeau) que le **périphérique met à '1' ou à '0'** pour indiquer la fin d'une opération.
  - Le **bit bx** peut être un bit d'un registre lié à une entrée (exemple d'une patte d'un PORT). Dans ce cas le changement d'état dépend de l'extérieur (utilisateur dans le cas d'un bouton poussoir)
- Exemple :
  - **USARTx** :
    - bit **b7 (TXE)** du registre USARTx_ISR (Interrupt and Status Register) **passe à '1'** pour indiquer que le **registre de transmission est vide**, donc on peut envoyer un caractère
    - bit **b5 (RXNE)** du registre USARTx_ISR (Interrupt and Status Register) **passe à '1'** pour indiquer que le **registre de réception est plein**, donc on peut récupérer un caractère
  - **Bouton poussoir** : on attend ou vérifie un changement d'état ou un évènement (Flancs) sur la patte
- Tous les périphériques du MCU possèdent un registre de STATUS avec un bit que l'on peut scruter afin d'interagir avec le périphérique (récupération ou écriture de données)

Scrutation VS Interruption

Attente active

```
/*============================================================================
  FUNCTION:     AttenteActiveSansParametre(void)
  DESCRIPTION:  On fait compter le CPU du MCU dans le vide! pour attendre
  PARAMETERS:   rien
  RETURNS:      rien
  REQUIREMENTS: Ajuster la valeur max du compteur en fonction de la frequence du CPU
============================================================================*/
void AttenteActiveAvecParametre(unsigned int DureeEnMilliSeconde)
{
 // Faire attendre le CPU : Attente active car il comptera dans le vide
        unsigned int compteurUniteTemps;
        unsigned int compteurNombreUniteTemps;

        for (compteurNombreUniteTemps=0; compteurNombreUniteTemps<DureeEnMilliSeconde; compteurNombreUniteTemps++)
        {
          // boucle d'attente vide pour faire attendre le CPU
              for (compteurUniteTemps=0; compteurUniteTemps<BaseDeTemps; compteurUniteTemps++)
              {
                  // boucle d'attente vide pour faire attendre le CPU

              }

        }
}
```

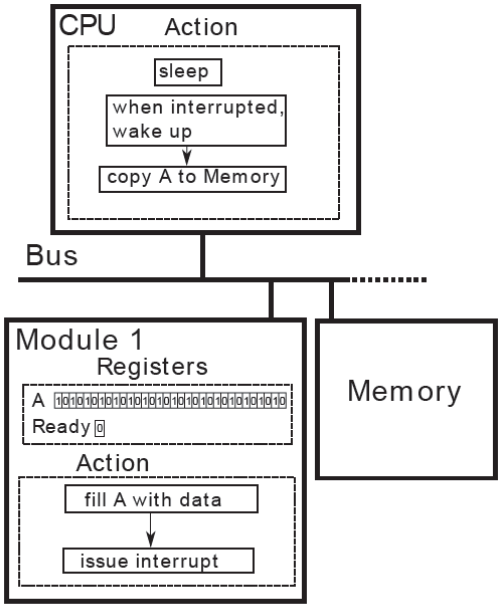
Scrutation VS Interruption


```
int main (void)
{
   initPattePA5_LedVerte();
   initPattePC13_BoutonPoussoirBleu();

  while(1)   // Boucle principale ou infinie
  {
   while (!DetectBoutonBleuNucleo_PC13())
            {
                     // on attends le flanc montant

            }
   // Action si flanc montant
   while (DetectBoutonBleuNucleo_PC13())
            {
                     // on attends le flanc descendant
            }
   // Action si flanc montant

    GPIOA->ODR^=LED_Verte_Nucleo ;  // change etat bit b5
  }
  // Fin : Le CPU n'arrivera jamais ici
}
```

Scrutation de l'entrée
Le CPU attend .
Dans le cas du bouton poussoir cet attente peut durer vraiment longtemps
Le temps d'attend est du temps perdu et on consomme de l'énergie !

Scrutation VS Interruption

Interruption



- Le CPU ne Scrute pas le Périphérique.
- C'est le périphérique qui interrompt le CPU une fois qu'il a fini l'opération.

Avantages :

- Le CPU **peux faire autre chose** pendant que le périphérique travaille
- Si le CPU n'a rien à faire on peut l'arrêter et le périphérique le réveillera : **Economie d'énergie**

Interruption ? définition

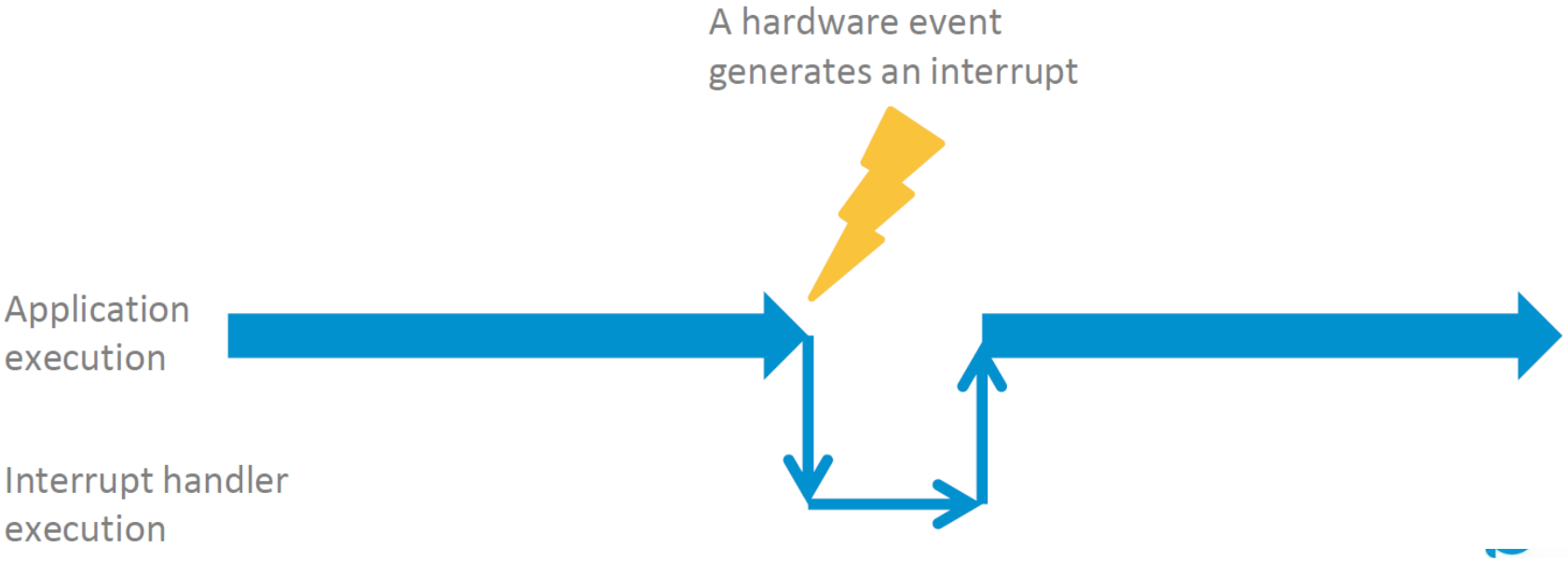
- Évènements **asynchrones**:
  - Ex: bouton poussoir (on ne connais pas le moment ou l'utilisateur vas actionner le bouton)
  - Liaison série USART: un caractère peut arriver à tout moment

- Une interruption est un signal demandant au processeur de suspendre temporairement l'exécution du programme courant afin d'effectuer des opérations particulières: exécuté un sous-programme.→ **fonction dont l'appel est déclenché matériellement**

- Une demande d'interruption est un **signal physique** passant de l'état inactif à l'état actif et **allant du périphérique vers le CPU**

Interruption ? définition

- En connectant une fonction C à une source ou évènement d'interruption, l'appel de la fonction sera déclenché matériellement
- La fonction C est appelée **Interrupt Service Routine (ISR)** ou **Interrupt Handler**
- Ce mécanisme permet d'implémenter une réaction à une sollicitation en respectant les exigences suivantes:
  - →offrir un délai de réponse très bref,
  - →programmation indépendante du code en cours d'exécution.



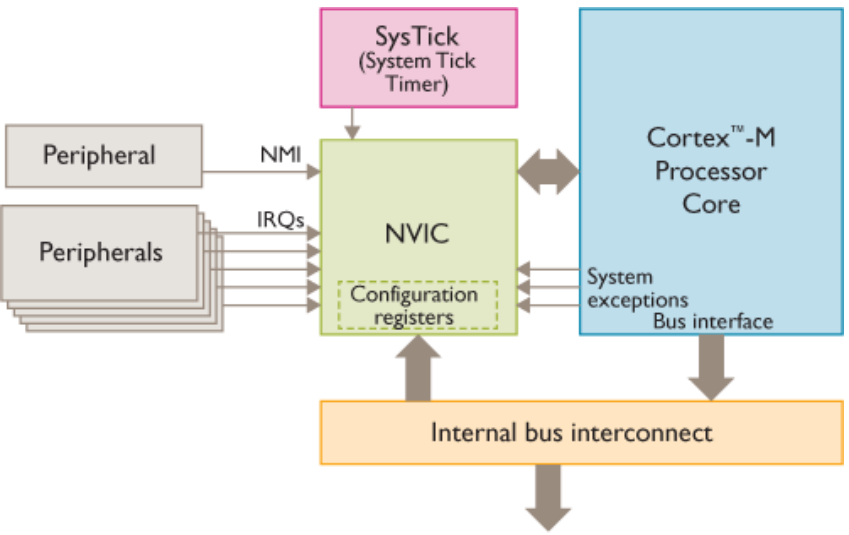
Interruption? Les sources

- Les interruptions peuvent être demandées ou initiées par :
  - Un périphérique : composant extérieur au CPU (Cortex M0 dans le cas des MCUs STM32F0xx)
    - TIMERS :
      - Débordement
      - évènements
    - Une ou plusieurs pattes peuvent lancer une requête d'interruption (Interrupt ReQuest, **IRQ**) souvent appelé **INTERRUPTIONS EXTERNES** car le **signal** provient de l'**extérieur** (ex. **Bouton poussoir**) pour indiquer:
      - L'apparition d'un niveau logique (haut ou bas)
      - Un évènement : Flanc montant ou Flanc descendant
    - USART (liaison) peut demander une interruption (**IRQ**) pour:
      - signaler qu'un caractère est reçu,
      - Signaler qu'un caractère est transmis
    - Un convertisseur Analogique vers Numérique (ADC) peut lancer une requête d'interruption (**IRQ**) pour signaler :
      - la fin d'une conversion sur une ou plusieurs pattes
      - erreur de conversion…
      - ✓ La plupart des périphériques peuvent demander une interruption (**IRQ**)
  - CPU ou système: on parle d'**exceptions**
    - ✓ **Reset** (interruption système ou exception).
    - ✓ **Fautes** : écriture à une adresse mémoire interdite par exemple
      - ✓ **Exemple: écran bleu de Windows**
- Si le CPU est sensible aux demandes d'interruptions, à chaque demande il devra exécuter un **sous programme d'interruption: Interrupt Service Routine (ISR)**

Interruption? : Contrôleur d'interruption

- Comme il y a plusieurs sources d'interruption il faut un **arbitre**! C'est le rôle du **contrôleur d'interruption**
- Le contrôleur d'interruption permet au **développeur** :
  - **Activer** ou **Désactiver** une ou plusieurs sources d'interruptions : on parle de **DEMASQUER** ou **MASQUER**
    - **Les interruptions liées au CPU (Reset, fautes d'écriture…) ne peuvent pas être démasqués**
  - De classer les demandes d'interruption par ordre de **priorité**
    - **Les règles de priorité sont définies par le fabricants du CPU ou du MCUs**
      - **Par exemple : Une interruption de haute priorité peut interrompre une interruption de base priorité, l'inverse n'est pas possible**

Interruption? : Contrôleur d'interruption NVIC

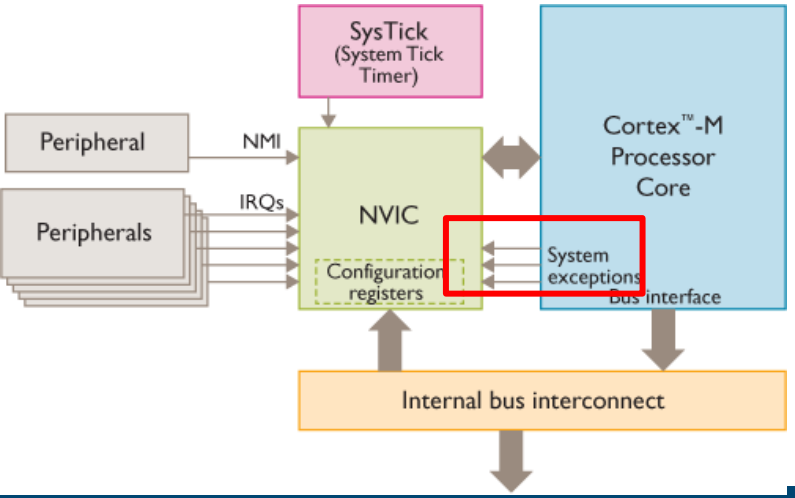


- Appelé **NVIC** pour **N**ested **V**ector **I**nterrupt **C**ontroller
- Intégré au CPU cortex M0
- Contrôleur d'interruptions standard des CPU Cortex-M.
  - Tous les MCUs utilisant un Cortex M, contiennent le NVIC
  - Les notions acquises sont applicables à tous ces MCUs
- Toutes les demandes d'interruptions passe le NVIC avec d'arriver au Core Cortex M0
- **Arbitrage** des demandes d'interruptions
  - Gestion des priorités
  - Activation/désactivation des interruptions masquables
- Peux gérer jusqu'à **32 sources d'interruptions + 6 sources d' exceptions**

Interruption? : Contrôleur d'interruption NVIC

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	fixed	Reset	Reset	0x0000 0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000 0008
-	-1	fixed	HardFault	All class of fault	0x0000 000C
-	3	settable	SVCall	System service call via SWI instruction	0x0000 002C
-	5	settable	PendSV	Pendable request for system service	0x0000 0038
-	6	settable	SysTick	System tick timer	0x0000 003C

Adresse de début de la table des vecteurs



Tableaux des exceptions

Lors qu'une exception est déclenchée, le CPU commence par récupérer **l'adresse de la routine** à exécuter à partir d'un emplacement mémoire spécifique connu d'avance. **L'ensemble des adresses** des routines relatives aux différentes sources d'interruptions forme la ***table de vecteurs d'interruptions***.

Interruption? : Contrôleur d'interruption NVIC

Position	Priority	Type of priority	Acronym	Description	Address
0	7	settable	WWDG	Window watchdog interrupt	0x0000 0040
1	8	settable	PVD_VDDIO2	PVD and V_{DDIO2} supply comparator interrupt (combined EXTI lines 16 and 31)	0x0000 0044
2	9	settable	RTC	RTC interrupts (combined EXTI lines 17, 19 and 20)	0x0000 0048
3	10	settable	FLASH	Flash global interrupt	0x0000 004C
4	11	settable	RCC_CRS	RCC and CRS global interrupts	0x0000 0050
5	12	settable	EXTI0_1	EXTI Line[1:0] interrupts	0x0000 0054
6	13	settable	EXTI2_3	EXTI Line[3:2] interrupts	0x0000 0058
7	14	settable	EXTI4_15	EXTI Line[15:4] interrupts	0x0000 005C
8	15	settable	TSC	Touch sensing interrupt	0x0000 0060
9	16	settable	DMA_CH1	DMA channel 1 interrupt	0x0000 0064
10	17	settable	DMA_CH2_3 DMA2_CH1_2	DMA channel 2 and 3 interrupts DMA2 channel 1 and 2 interrupts	0x0000 0068
11	18	settable	DMA_CH4_5_6_7 DMA2_CH3_4_5	DMA channel 4, 5, 6 and 7 interrupts DMA2 channel 3, 4 and 5 interrupts	0x0000 006C
12	19	settable	ADC_COMP	ADC and COMP interrupts (ADC interrupt combined with EXTI lines 21 and 22)	0x0000 0070
13	20	settable	TIM1_BRK_UP_ TRG_COM	TIM1 break, update, trigger and commutation interrupt	0x0000 0074
14	21	settable	TIM1_CC	TIM1 capture compare interrupt	0x0000 0078
15	22	settable	TIM2	TIM2 global interrupt	0x0000 007C

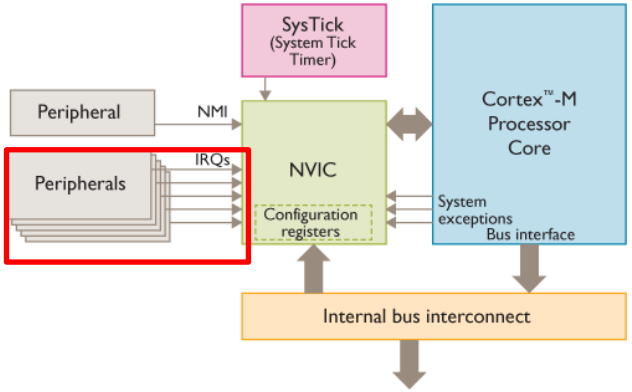
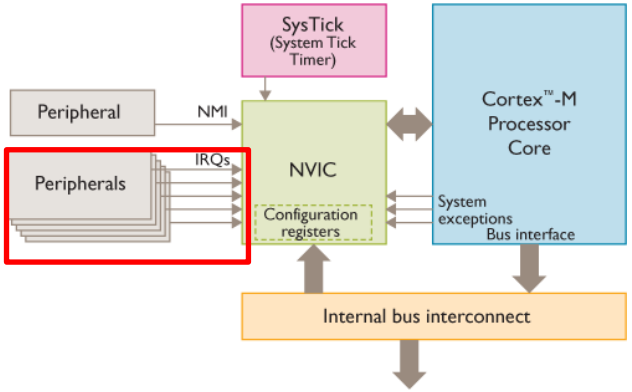


Tableau des sources d'interruptions
On note que plusieurs sources peuvent avoir la même adresse mémoire ou vecteur

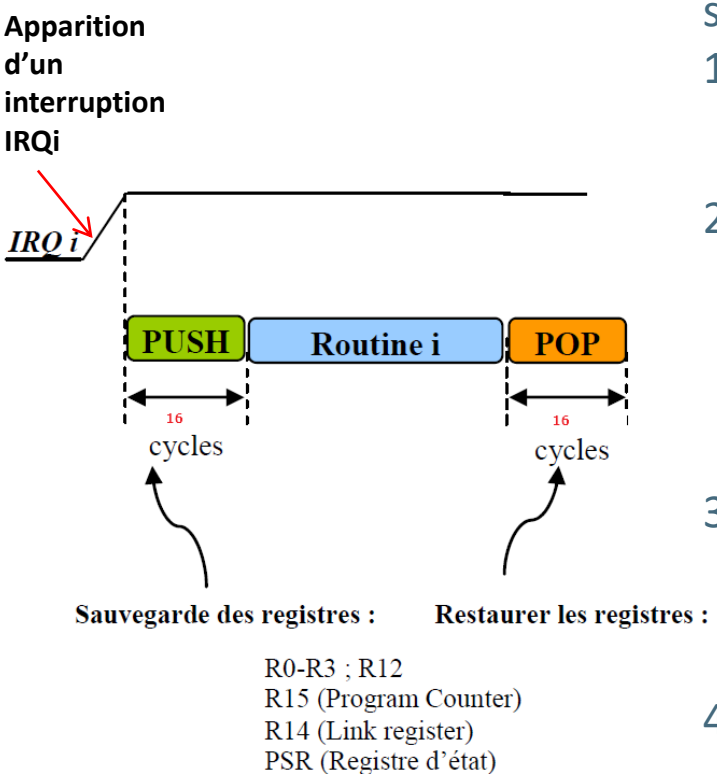
Lors qu'une interruption autorisée est déclenchée, le CPU récupère **l'adresse de la routine** depuis un emplacement mémoire spécifique connu d'avance. **L'ensemble des adresses** des routines relatives aux différentes sources d'interruptions forme la ***table de vecteurs d'interruptions***.

Interruption? : Contrôleur d'interruption NVIC

Position	Priority	Type of priority	Acronym	Description	Address
16	23	settable	TIM3	TIM3 global interrupt	0x0000 0080
17	24	settable	TIM6_DAC	TIM6 global interrupt and DAC underrun interrupt	0x0000 0084
18	25	settable	TIM7	TIM7 global interrupt	0x0000 0088
19	26	settable	TIM14	TIM14 global interrupt	0x0000 008C
20	27	settable	TIM15	TIM15 global interrupt	0x0000 0090
21	28	settable	TIM16	TIM16 global interrupt	0x0000 0094
22	29	settable	TIM17	TIM17 global interrupt	0x0000 0098
23	30	settable	I2C1	I^2C1 global interrupt (combined with EXTI line 23)	0x0000 009C
24	31	settable	I2C2	I^2C2 global interrupt	0x0000 00A0
25	32	settable	SPI1	SPI1 global interrupt	0x0000 00A4
26	33	settable	SPI2	SPI2 global interrupt	0x0000 00A8
27	34	settable	USART1	USART1 global interrupt (combined with EXTI line 25)	0x0000 00AC
28	35	settable	USART2	USART2 global interrupt (combined with EXTI line 26)	0x0000 00B0
29	36	settable	USART3_4_5_6_7_8	USART3, USART4, USART5, USART6, USART7, USART8 global interrupts (combined with EXTI line 28)	0x0000 00B4
30	37	settable	CEC_CAN	CEC and CAN global interrupts (combined with EXTI line 27)	0x0000 00B8
31	38	settable	USB	USB global interrupt (combined with EXTI line 18)	0x0000 00BC



Interruption? Déroulement d'une interruption



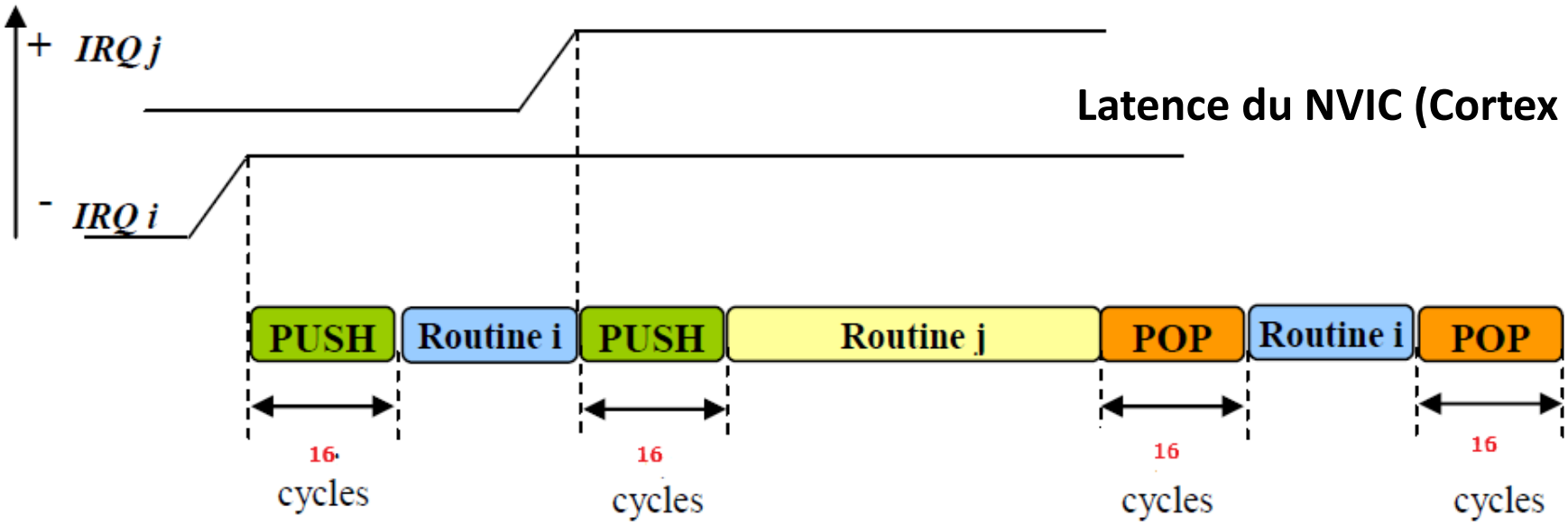
PUSH : opération qui permet au CPU de sauvegarder dans la pile son état
POP : Opération inverse

Lorsqu'une interruption autorisée (IRQ) ou démasquée survient, le CPU :

1. Arrête l'exécution normale d'un programme (main () par exemple) : le programme est interrompu: L'instruction en cours termine son exécution.
2. Il sauvegarde l'état du programme en cours d'exécution : c'est contexte d'exécution : registres d'états, données temporaires, Program Counter (PC)
   i. Afin de pouvoir revenir au bon endroit pour continuer son travail!
3. Repère l'adresse de la routine d'interruption appropriée , obtenue en chargeant le contenu d'un vecteur d'interruption
4. Exécute la routine d'interruption spécifique liée a cet évènement : la routine de service de l'interruption : Interrupt Service Routine (ISR)
5. Restaure le contexte d'exécution registres d'états, données temporaires Program Counter (PC)
   i. Retourne à l'endroit ou il a été interrompu.

Interruption? Latence d'une interruption

Temps de réponse d'une interruption ou **latence:** durée pouvant s'écouler entre la demande d'interruption et l'exécution de la routine d'interruption (interrupt Handler)



16 Cycles

Exemple : pour fréquence d'horloge du CPU de 8 MHz → minimum 2 µs

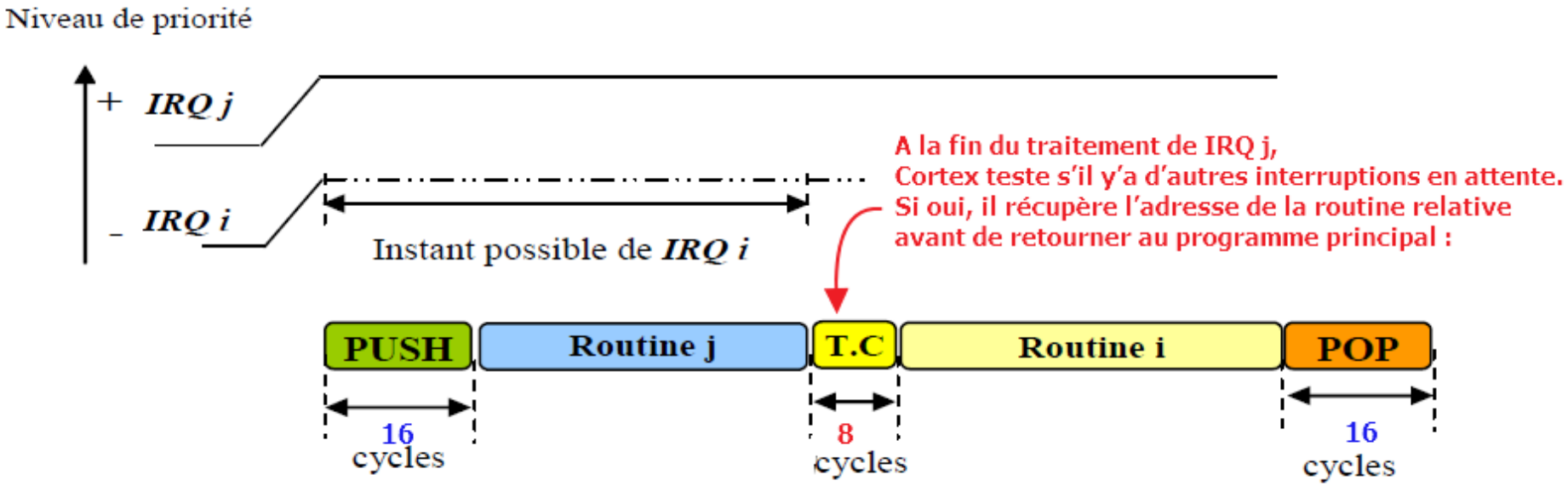
Interruption? Latence d'une interruption

- La Latence dépend de plusieurs facteurs :
  - Est-ce que l'interruption concernée survient après une autre interruption ? De basse ou de haute priorité?
  - Est-ce que l'interruption concernée survient à un moment ou l'on vient de quitter une interruption? de haute priorité? Ou de basse priorité?
  - ….
- La Latence peux donc varier selon les situations. Le fabriquant de CPU et/ou du MCUs devrait fournir des informations précises sur ces points.

Interruption? Latence d'une interruption

Gestion de Latence dans le cortex M0

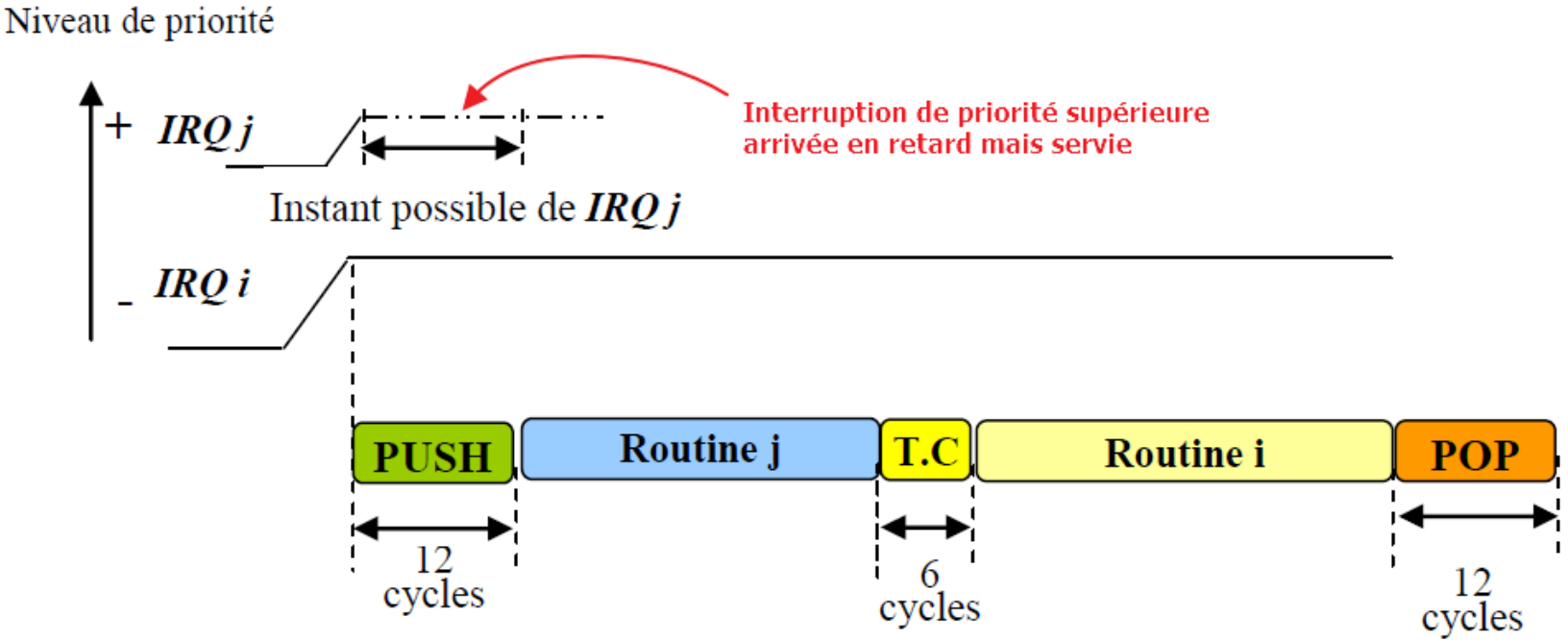
- Le NVIC permet dans certains cas de réduire la Latence
  - Tail Chaining interrupt



Interruption? Latence d'une interruption

Gestion de Latence dans le cortex M0

- Le NVIC permet dans certains cas de réduire la Latence
  - Late arrival



Interruption : Programmation en C

- Les compilateurs C fournissent des mécanismes permettant de programmer les interruptions sans recourir au langage assembleur.
- Pour les MCU à base de Cortex Mx comme stm32F0xx. L'interface de programmation en C CMSIS fournie des :
  - Mots clés ou Labels pour désigner les Routines d'interruption:
    - **NomSouceInterruption_IRQHandler(void)**
      - Exemple : **TIM2_IRQHandler(void) :** routine interruption du Timer TIM2
    - Des fonctions pour gérer les demandes d'interruption au niveau du NIVIC : voir table du transparent suivant
- Toute variable dont la valeur peut être modifiée par une routine d'interruption, doit être qualifiée avec le mot clé **volatile**. Pour éviter toute tentative d'optimisation incorrecte du compilateur
  - Exemple : *volatile unsigned int CaractereRecu;*

Fonction	Rôle
void NVIC_EnableIRQ(IRQn_Type IRQn) Paramètre entrée : source interruption Paramètre renvoyé : aucun	Activation au niveau du NVIC d'une source d'interruption d'un périphérique. Le processeur devient donc sensible (exécution de la routine d'interruption) à toute demande d'interruption venant de cette source.
void NVIC_DisableIRQ(IRQn_Type IRQn) Paramètre entrée : source interruption Paramètre renvoyé : aucun	Désactivation au niveau du NVIC d'une source d'interruption d'un périphérique. Le processeur devient donc insensible (pad d'exécution de la routine d'interruption) à toute demande d'interruption venant de cette source.
uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn) Paramètre entrée : source interruption Paramètre renvoyé : status : '1' si demande d'interruption en attente, '0' si non	Permet de connaitre l'état du traitement d'une demande d'interruption. Utile dans le cas où il y a plusieurs sources d'interruptions activées avec des niveaux de priorités différentes.
void NVIC_SetPendingIRQ(IRQn_Type IRQn) Paramètre entrée : source interruption Paramètre renvoyé : aucun	Permet mettre en attente une demande d'interruption. Utile dans le cas où il y a plusieurs sources d'interruptions activées avec des niveaux de priorités différentes.
NVIC_ClearPendingIRQ(IRQn_Type IRQn) Paramètre entrée : source interruption Paramètre renvoyé : aucun	Permet de forcer la prise en compte d'une demande d'interruption. Utile dans le cas où il y a plusieurs sources d'interruptions activées avec des niveaux de priorités différentes.
NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority) Paramètre entrée : source interruption et niveau de priorité Paramètre renvoyé : aucun	Permet de fixer le niveau de priorité d'une demande d'interruption. Utile dans le cas où il y a plusieurs sources d'interruptions activées avec des niveaux de priorités différentes.
uint32_t NVIC_GetPriority(IRQn_Type IRQn) Paramètre entrée : source interruption Paramètre renvoyé : niveau de priorité	Permet d'obtenir le niveau de priorité d'une demande d'interruption. Utile dans le cas où il y a plusieurs sources d'interruptions activées avec des niveaux de priorités différentes.
void NVIC_SystemReset(void)	Permet de faire le reset software du système

Ces fonctions sont définies dans les fichiers ***core_cm0.h et core_cm0.c***

Interruption : Programmation en C : Noms des routines interruptions

```
Default_Handler PROC
                EXPORT  WWDG_IRQHandler
                EXPORT  PVD_VDDIO2_IRQHandler
                EXPORT  RTC_IRQHandler
                EXPORT  FLASH_IRQHandler
                EXPORT  RCC_CRS_IRQHandler
                EXPORT  EXTI0_1_IRQHandler
                EXPORT  EXTI2_3_IRQHandler
                EXPORT  EXTI4_15_IRQHandler
                EXPORT  TSC_IRQHandler
                EXPORT  DMA1_Channel1_IRQHandler
                EXPORT  DMA1_Channel2_3_IRQHandler
                EXPORT  DMA1_Channel4_5_6_7_IRQHandler
                EXPORT  ADC1_COMP_IRQHandler
                EXPORT  TIM1_BRK_UP_TRG_COM_IRQHandler
                EXPORT  TIM1_CC_IRQHandler
                EXPORT  TIM2_IRQHandler
                EXPORT  TIM3_IRQHandler
                EXPORT  TIM6_DAC_IRQHandler
                EXPORT  TIM7_IRQHandler
                EXPORT  TIM14_IRQHandler
                EXPORT  TIM15_IRQHandler
                EXPORT  TIM16_IRQHandler
                EXPORT  TIM17_IRQHandler
                EXPORT  I2C1_IRQHandler
                EXPORT  I2C2_IRQHandler
                EXPORT  SPI1_IRQHandler
                EXPORT  SPI2_IRQHandler
                EXPORT  USART1_IRQHandler
                EXPORT  USART2_IRQHandler
                EXPORT  USART3_4_IRQHandler
                EXPORT  CEC_CAN_IRQHandler
```

Nom routine interruption

Sources : périphériques

- WWDG_IRQHandler → Window watchdog interrupt
- PVD_VDDIO2_IRQHandler → comparateur tension alim + IT Externes 16 et 31
- EXTI0_1_IRQHandler → Interruptions externes : Pattes 0 et 1
- EXTI2_3_IRQHandler → Interruptions externes : Pattes 2 et 3
- EXTI4_15_IRQHandler → Interruptions externes : Pattes 4 à 15
- ADC1_COMP_IRQHandler → Interruptions ADC et comparateurs
- TIM1_BRK_UP_TRG_COM_IRQHandler → Interruptions TIMER 1: Overflow, Trigger, Break...
- TIM1_CC_IRQHandler → Interruptions TIMER 1: Capture/Compare
- TIM2_IRQHandler → Interruptions TIMER 3: Toutes sources
- TIM6_DAC_IRQHandler → Interruptions TIMER 6 et DAC
- USART3_4_IRQHandler → Interruptions UART3 et 4

Extrait du fichier : startup_stm32f072.s

Interruption : Programmation en C : Noms type ou vecteurs d' interruptions

```
290   WWDG_IRQn                 = 0,      /*!< Window WatchDog Interrupt
291   PVD_VDDIO2_IRQn           = 1,      /*!< PVD and VDDIO2 supply comparator through EXTI Line
292   RTC_IRQn                  = 2,      /*!< RTC through EXTI Line Interrupt
293   FLASH_IRQn                = 3,      /*!< FLASH Interrupt
294   RCC_CRS_IRQn              = 4,      /*!< RCC and CRS Interrupts
295   EXTI0_1_IRQn              = 5,      /*!< EXTI Line 0 and 1 Interrupts
296   EXTI2_3_IRQn              = 6,      /*!< EXTI Line 2 and 3 Interrupts
297   EXTI4_15_IRQn             = 7,      /*!< EXTI Line 4 to 15 Interrupts
298   TSC_IRQn                  = 8,      /*!< TSC Interrupt
299   DMA1_Channel1_IRQn        = 9,      /*!< DMA1 Channel 1 Interrupt
300   DMA1_Channel2_3_IRQn      = 10,     /*!< DMA1 Channel 2 and Channel 3 Interrupts
301   DMA1_Channel4_5_6_7_IRQn  = 11,     /*!< DMA1 Channel 4, Channel 5, Channel 6 and Channel 7
302   ADC1_COMP_IRQn            = 12,     /*!< ADC1, COMP1 and COMP2 Interrupts
303   TIM1_BRK_UP_TRG_COM_IRQn  = 13,     /*!< TIM1 Break, Update, Trigger and Commutation Interr
304   TIM1_CC_IRQn              = 14,     /*!< TIM1 Capture Compare Interrupt
305   TIM2_IRQn                 = 15,     /*!< TIM2 Interrupt
306   TIM3_IRQn                 = 16,     /*!< TIM3 Interrupt
307   TIM6_DAC_IRQn             = 17,     /*!< TIM6 and DAC Interrupts
308   TIM7_IRQn                 = 18,     /*!< TIM7 Interrupts
309   TIM14_IRQn                = 19,     /*!< TIM14 Interrupt
310   TIM15_IRQn                = 20,     /*!< TIM15 Interrupt
311   TIM16_IRQn                = 21,     /*!< TIM16 Interrupt
312   TIM17_IRQn                = 22,     /*!< TIM17 Interrupt
313   I2C1_IRQn                 = 23,     /*!< I2C1 Interrupt
314   I2C2_IRQn                 = 24,     /*!< I2C2 Interrupt
315   SPI1_IRQn                 = 25,     /*!< SPI1 Interrupt
316   SPI2_IRQn                 = 26,     /*!< SPI2 Interrupt
317   USART1_IRQn               = 27,     /*!< USART1 Interrupt
318   USART2_IRQn               = 28,     /*!< USART2 Interrupt
319   USART3_4_IRQn             = 29,     /*!< USART3 and USART4 Interrupts
320   CEC_CAN_IRQn              = 30,     /*!< CEC and CAN Interrupts
321   USB_IRQn                  = 31      /*!< USB Low Priority global Interrupt
```

Nom des sources d'interruptions pour le STM32F072RB extrait du fichier stm32f0xx.h

Interruption : Programmation en C : Noms type ou vecteurs d' interruptions

```
209 /******  STM32F051  specific Interrupt Numbers **********************************/
210   WWDG_IRQn                 = 0,      /*!< Window WatchDog Interrupt                       */
211   PVD_IRQn                  = 1,      /*!< PVD through EXTI Line detect Interrupt          */
212   RTC_IRQn                  = 2,      /*!< RTC through EXTI Line Interrupt                 */
213   FLASH_IRQn                = 3,      /*!< FLASH Interrupt                                 */
214   RCC_IRQn                  = 4,      /*!< RCC Interrupt                                   */
215   EXTI0_1_IRQn              = 5,      /*!< EXTI Line 0 and 1 Interrupts                    */
216   EXTI2_3_IRQn              = 6,      /*!< EXTI Line 2 and 3 Interrupts                    */
217   EXTI4_15_IRQn             = 7,      /*!< EXTI Line 4 to 15 Interrupts                    */
218   TS_IRQn                   = 8,      /*!< Touch sense controller Interrupt                */
219   DMA1_Channel1_IRQn        = 9,      /*!< DMA1 Channel 1 Interrupt                        */
220   DMA1_Channel2_3_IRQn      = 10,     /*!< DMA1 Channel 2 and Channel 3 Interrupts         */
221   DMA1_Channel4_5_IRQn      = 11,     /*!< DMA1 Channel 4 and Channel 5 Interrupts         */
222   ADC1_COMP_IRQn            = 12,     /*!< ADC1, COMP1 and COMP2 Interrupts                */
223   TIM1_BRK_UP_TRG_COM_IRQn  = 13,     /*!< TIM1 Break, Update, Trigger and Commutation Interrupts */
224   TIM1_CC_IRQn              = 14,     /*!< TIM1 Capture Compare Interrupt                  */
225   TIM2_IRQn                 = 15,     /*!< TIM2 Interrupt                                  */
226   TIM3_IRQn                 = 16,     /*!< TIM3 Interrupt                                  */
227   TIM6_DAC_IRQn             = 17,     /*!< TIM6 and DAC Interrupts                         */
228   TIM14_IRQn                = 19,     /*!< TIM14 Interrupt                                 */
229   TIM15_IRQn                = 20,     /*!< TIM15 Interrupt                                 */
230   TIM16_IRQn                = 21,     /*!< TIM16 Interrupt                                 */
231   TIM17_IRQn                = 22,     /*!< TIM17 Interrupt                                 */
232   I2C1_IRQn                 = 23,     /*!< I2C1 Interrupt                                  */
233   I2C2_IRQn                 = 24,     /*!< I2C2 Interrupt                                  */
234   SPI1_IRQn                 = 25,     /*!< SPI1 Interrupt                                  */
235   SPI2_IRQn                 = 26,     /*!< SPI2 Interrupt                                  */
236   USART1_IRQn               = 27,     /*!< USART1 Interrupt                                */
237   USART2_IRQn               = 28,     /*!< USART2 Interrupt                                */
238   CEC_IRQn                  = 30      /*!< CEC Interrupt                                   */
```

Nom des sources d'interruptions pour le STM32F051

Interruption : Programmation en C : Guide étapes

1. Configurer les pattes et le ou les périphériques
2. Activer la demande interruption au niveau du périphérique
3. Activer la demande au niveau du NVIC
4. Écrire la routine interruption