

Programmation en interruptions

Module Info 2

`Sebastien.Kramm@univ-rouen.fr`

IUT GEII Rouen

2013-2014

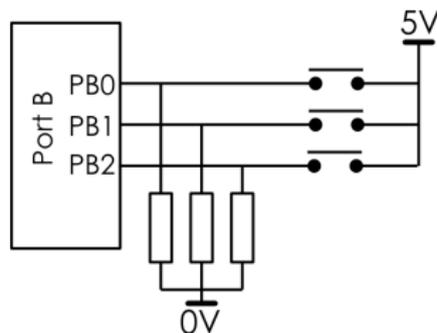
- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

- De nombreuses applications demandent de tenir compte d'un "évènement" pour modifier l'exécution du programme
- Exemple d'évènements **externes**
 - Action de l'utilisateur (bouton poussoir, ...)
 - Périphérique, capteur,... réclamant une attention immédiate.
- Exemple d'évènements **internes** (au système)
 - Erreur de calcul (division par zéro)
 - Reset
 - Chien de garde ("Watchdog")
 - Fin d'un délai (timer interne) (**voir exemples en fin de cours**)

Quelles solutions ?

- Solution 1 : fonctionnement en **scrutation**
⇒ Consiste à scruter périodiquement les signaux à surveiller.

- Exemple : soit un clavier de 3 BP connecté sur le port A.

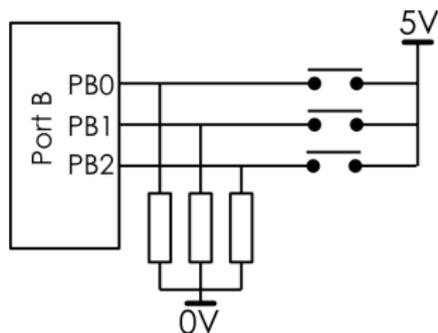


Quelles solutions ?

- Solution 1 : fonctionnement en **scrutation**

⇒ Consiste à scruter périodiquement les signaux à surveiller.

- Exemple : soit un clavier de 3 BP connecté sur le port A.



- En assembleur

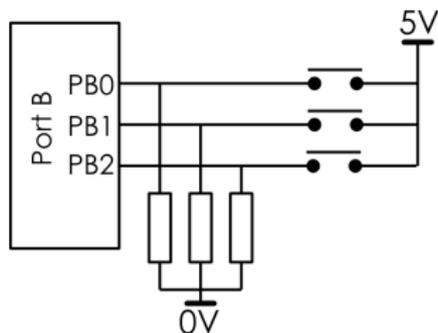
```
1  DEBUT  brset  PORTA,BIT0,TRAIT_0
2         brset  PORTA,BIT1,TRAIT_1
3         brset  PORTA,BIT2,TRAIT_2
4         ...
5         bra  DEBUT
```

Quelles solutions ?

- Solution 1 : fonctionnement en **scrutation**

⇒ Consiste à scruter périodiquement les signaux à surveiller.

- Exemple : soit un clavier de 3 BP connecté sur le port A.



- En assembleur

```
1 DEBUT brset PORTA,BIT0,TRAIT_0
2   brset PORTA,BIT1,TRAIT_1
3   brset PORTA,BIT2,TRAIT_2
4   ...
5   bra DEBUT
```

► Retour

- En C

```
1 while( 1 )
2 {
3     if( (PORTA&BIT0) != 0 )
4         fonction_A();
5     if( (PORTA&BIT1) != 0 )
6         fonction_B();
7 }
```

Scrutation (en anglais : *Polling*)

- Avantage : simple à comprendre...
- Inconvénients :
 - On peut rater un événement, ou le prendre en compte trop tard.
 - Le CPU passe son temps à surveiller, ce qui est peu productif.
 - Programmation compliquée...

Solution 2 : fonctionnement **en interruptions**

Principe général

Lorsque l'évènement (attendu) survient, le CPU exécute automatiquement le code prévu dans ce cas là.

- Mécanisme présent sur **tous** les systèmes informatiques, du plus simple au plus élaboré.
- On ne peut plus rater un événement, ils sont mémorisés, puis traités, automatiquement.
- Hierarchisation des priorités selon le type d'évènement
Exemple : panne d'alimentation "plus prioritaire" qu'un appui sur un bouton-poussoir.

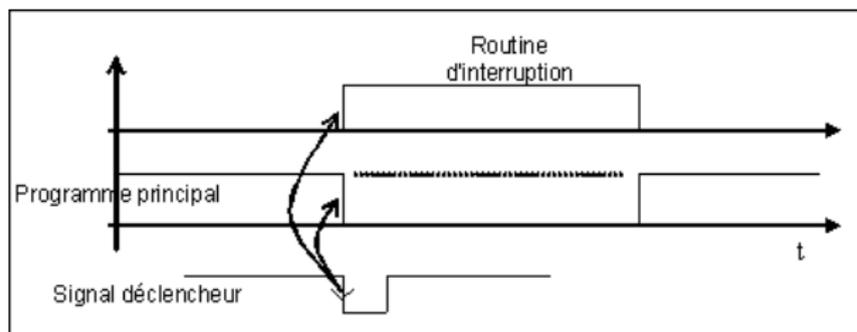
- Permet d'avoir des systèmes réactifs.
- Les évènements sont traités automatiquement !
⇒ Simplification de la programmation.
- Implémentation aisée de mécanismes de sécurité (type "Chien de Garde", *Watchdog*).

- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

- Le CPU est conçu pour pouvoir être interrompu dans l'exécution d'un programme.
- Lors de l'arrivée de l'évènement, l'exécution se déroute **automatiquement** vers la **routine d'interruption** (ISR : *Interrupt Service Routine*).
- Le processeur **sauvegarde** les registres CPU sur la pile, avant l'appel de la routine.
- En fin de routine, l'état du CPU est **restauré**.
⇒ reprise de l'exécution "normale" (programme principal), sans aucune perturbation.

Illustration

- Dès que l'évènement attendu survient, l'exécution du programme principal s'interrompt, et le CPU exécute la routine.
- Quand la tâche est terminée, le CPU reprend l'exécution du programme principal.



- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

Pourquoi mémoriser ?

- Si deux demandes d'interruptions arrivent simultanément :
 - celle de la plus haute priorité est traitée d'abord,
 - l'autre est mémorisée, et traitée ensuite.

Pourquoi mémoriser ?

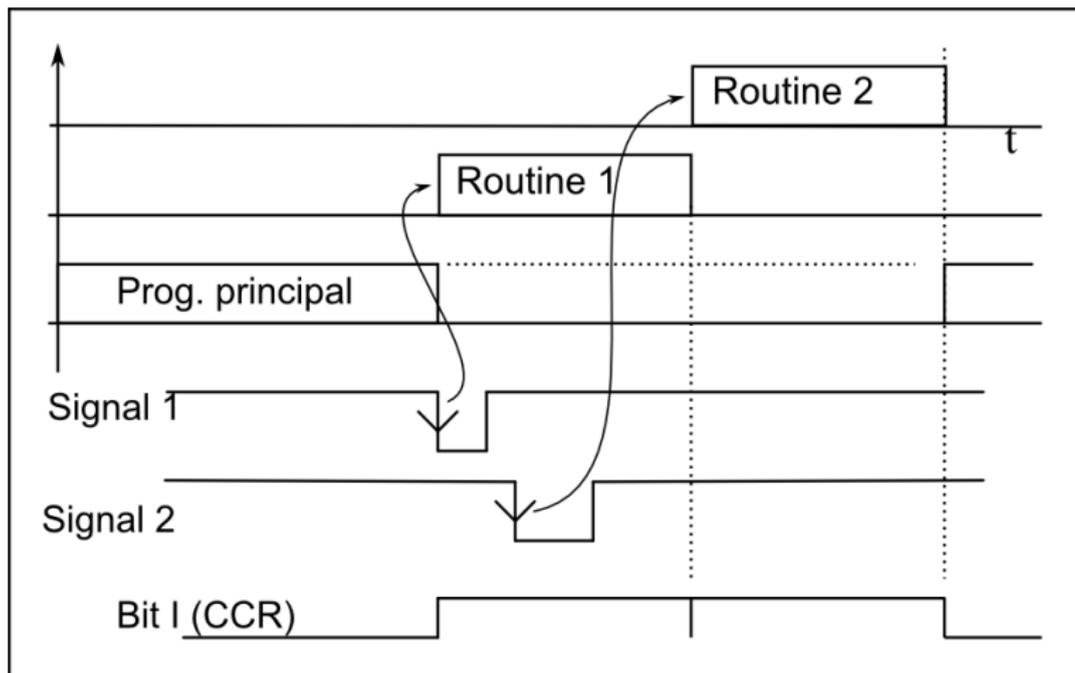
- Si deux demandes d'interruptions arrivent simultanément :
 - celle de la plus haute priorité est traitée d'abord,
 - l'autre est mémorisée, et traitée ensuite.

Comment se fait la mémorisation ?

- A travers un bit d'état, spécifique au sous-système concerné.
- Ces bits d'états (*flags*) sont :
 - activés automatiquement,
 - remis à zéro par programme.

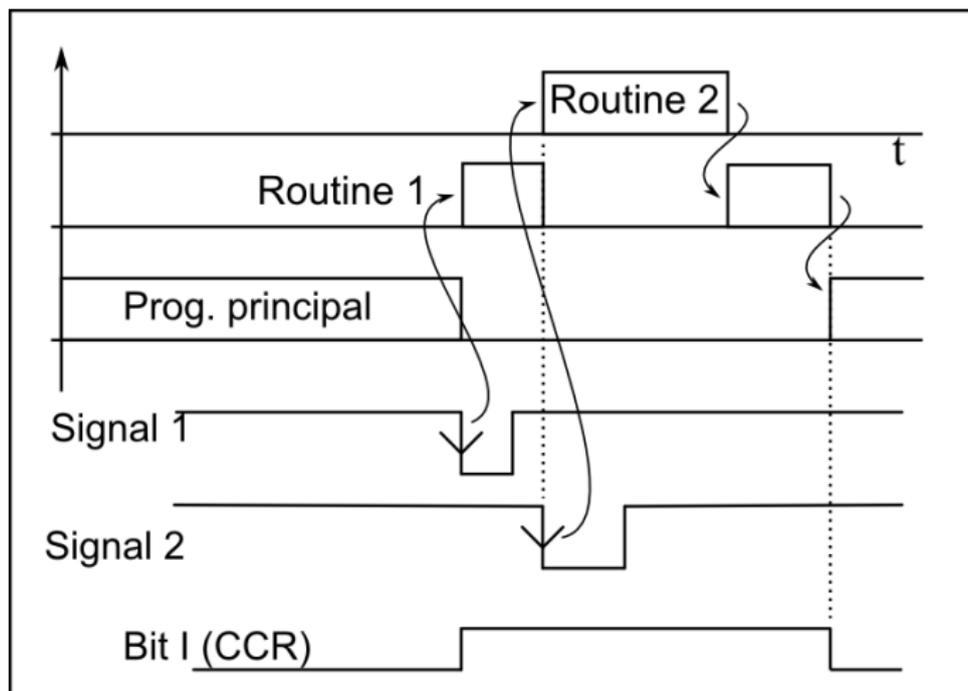
Mémorisation des demandes d'interruptions

- Cas 1 : interruption 2 "ordinaire" (masquable)
⇒ la routine 2 est exécutée après la terminaison de la routine 1.



Mémorisation des demandes d'interruptions

- Cas 2 : interruption 2 non-masquable (prioritaire)
⇒ la routine 2 est exécutée immédiatement, puis l'exécution de la routine 1 reprend.

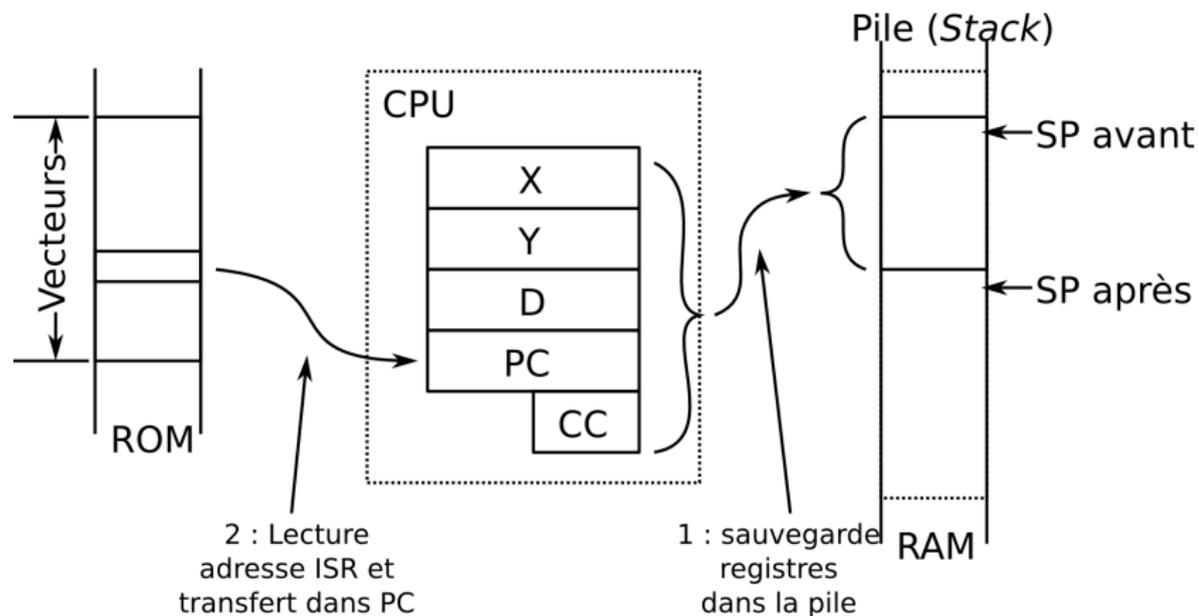


- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

Séquence d'une interruption

- Lors de l'arrivée de l'événement , le CPU :
 - ① sauvegarde tous les registres CPU sur la pile, et ajuste le pointeur de pile SP,
 - ② lit l'adresse de la routine dans le **vecteur d'interruption**,
 - ③ positionne le bit I à 1 : \Rightarrow les interruptions sont inhibées (pour éviter les interruptions imbriquées),
 - ④ se branche à l'adresse de la routine (par une recopie dans PC de la valeur du vecteur)
 \Rightarrow exécution de la routine d'interruption.

Séquence d'une interruption

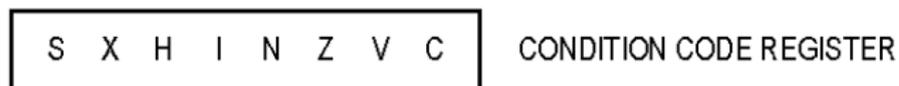


- L'exécution de l'instruction `rti` (*Return from interrupt*) provoque la **restauration** des registres CPU :
 - D, X, Y retrouvent les valeurs d'origine.
⇒ l'exécution peut reprendre sans perturbation.
 - CC est restauré, donc le bit I repasse à 0
⇒ interruptions à nouveau autorisées.
 - L'adresse de retour est recopiée dans PC :
⇒ reprise de l'exécution du programme principal.
 - Le pointeur de pile SP est ajusté.

Masquage des interruptions

- Mécanismes implantés directement sur la puce.
- Les interruptions sont divisées en 2 catégories :
 - non-masquables (Reset, XIRQ,...),
 - masquables.
- Les interruptions utilisateurs peuvent être inhibées ("masquées") globalement par un bit d'état.

9s12 : bit I du registre CC :



- I = 0 : interruptions validées.
- I = 1 : interruptions inhibées (= interdites).
- 9s12 : Une paire d'instruction dédiées permet de manipuler ce bit :
 - cli : -----
 - sei : -----

- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

- Un vecteur est une adresse **particulière** de l'espace mémoire (définie par le constructeur).
- A cet endroit est stocké l'**adresse** de la routine d'interruption : c'est au programmeur de compléter ce vecteur (méthode fonction du langage utilisé).
- Un type d'interruption = un vecteur.
- Regroupés dans la même zone mémoire : on parle de "table des vecteurs d'interruptions".
- La table des vecteurs est dans un système réel **toujours** en ROM, afin d'assurer un démarrage autonome du système.

Exemple : vecteur du "Reset"

- Implanté dans les 2 dernières adresses de la mémoire :
\$fffe,\$ffff
- Si mon programme applicatif commence à l'adresse \$1000, il faut que le vecteur du "Reset" pointe sur cette adresse :

\$ffff	\$00
\$fffe	\$10
\$fffd	...
...	...

- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

- 58 sources d'interruptions possibles, dont :
 - 3 types de Reset (Reset, COP, CMR)
 - 1 interruption logicielle (instruction SWI, Software Interrupt)
 - 1 interruption "code opératoire erroné"
- 2 broches permettent de générer une interruption de façon externe :
 - IRQ : (Interrupt ReQuest) : masquable
 - XIRQ : non-masquable
- Chaque sous-ensemble (Timer, SCI, SPI, ATD, CAN, etc) dispose de capacités d'interruptions (voir table des vecteurs).

9s12 : tables des vecteurs d'interruptions

- Voir poly...

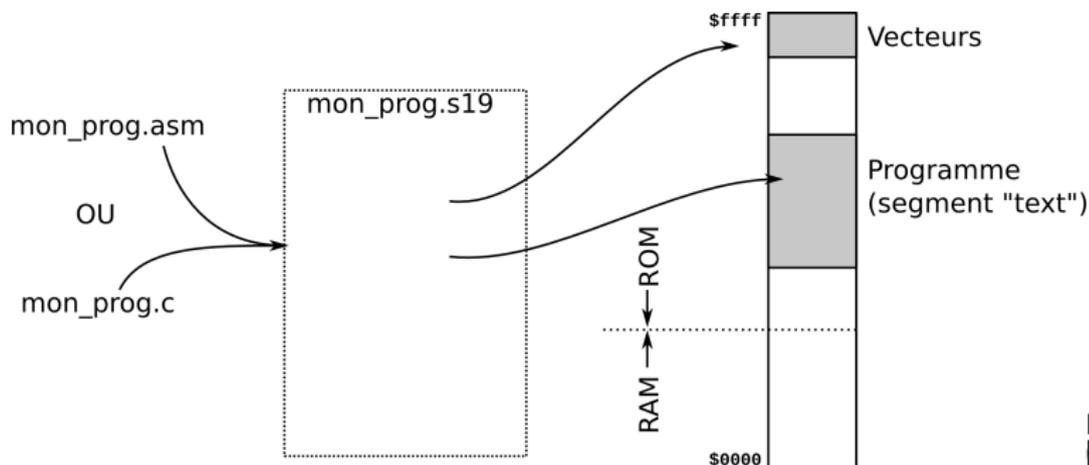
Table 5-1 Interrupt Vector Locations

Vector Address	Interrupt Source	CCR Mask	Local Enable	HPRIO Value to Elevate
\$FFFE, \$FFFF	Reset	None	None	-
\$FFFC, \$FFFD	Clock Monitor fail reset	None	PLLCTL (CME, SCME)	-
\$FFFA, \$FFFB	COP failure reset	None	COP rate select	-
\$FFF8, \$FFF9	Unimplemented instruction trap	None	None	-
\$FFF6, \$FFF7	SWI	None	None	-
\$FFF4, \$FFF5	XIRQ	X-Bit	None	-
\$FFF2, \$FFF3	IRQ	I-Bit	IRQCR (IRGEN)	\$F2
\$FFF0, \$FFF1	Real Time Interrupt	I-Bit	CRGINT (RTIE)	\$F0
\$FFEE, \$FFEF	Enhanced Capture Timer channel 0	I-Bit	TIE (C0I)	\$EE
\$FFEC, \$FFED	Enhanced Capture Timer channel 1	I-Bit	TIE (C1I)	\$EC
\$FFEA, \$FFEB	Enhanced Capture Timer channel 2	I-Bit	TIE (C2I)	\$EA
\$FFE8, \$FFE9	Enhanced Capture Timer channel 3	I-Bit	TIE (C3I)	\$E8

- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

4 sections de code à prévoir

- 3 sections de code exécutable :
 - initialisations (configuration des interruptions),
 - programme principal,
 - routine(s) d'interruption(s).
- 1 section de programmation du (des) vecteur(s) d'interruption(s) (code non-exécutable, écriture de valeurs dans le fichier de programmation .s19).



- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

- En assembleur, pour positionner le vecteur de reset sur un programme démarrant en \$1000 :

```
1      org  $fffe  
2      dc.w $1000
```

(dc : directive assembleur
"Define Constant")

Ou, de façon plus lisible :

```
1      org  vect_reset  
2      dc.w  debut_prog
```

- La gestion des interruptions n'est pas prévue à l'origine.
- La programmation du vecteur (et la déclaration de la fonction "routine d'interruption") se fera automatiquement via une macro (selon compilateur).
- Par exemple :

```
1 // validation des interruptions sur le canal 2 du Timer
2 #define USE_ISR_ECT_2
3 #include <vector.h>
```

- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption**
 - Programmation du vecteur
 - Initialisations**
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

1 - Bloc "Initialisations"

Il faut prévoir :

- Validation **locale** (selon le type d'interruption) : voir les exemples ;
- Validation **globale** : "démasquage" des interruptions (bit I de CC).

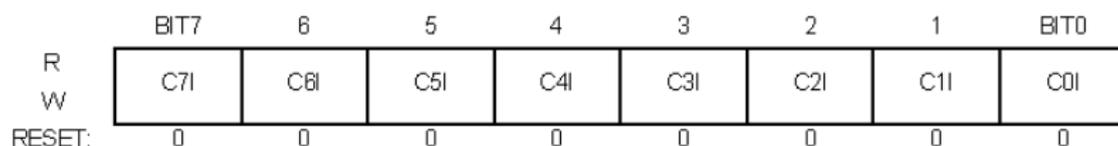
En C : on ne peut pas modifier directement les registres du CPU :

- Le mot clé `asm(...)` permet d'insérer une instruction assembleur.
- On écrira donc, à la fin du bloc d'initialisations :

```
asm( "cli" );
```

Validation locale : exemple 1

- Interruption sur un des 8 canaux du timer.
- le registre TIE contient les 8 bits de validation locale pour chacun des 8 canaux du Timer :



- $CxI = 1$: demande d'interruption lorsque CxF passe à 1
- $CxI = 0$: pas d'interruptions sur le canal x du Timer

Validation locale : exemple 2

- Interruptions sur débordement Timer (bit TOF).
- Le registre TSCR2 contient le bit TOI (*Timer Overflow Interrupt mask*) :



- TOI = 1 : demande d'interruption lorsque TOF passe à 1
- TOI = 0 : (défaut) pas d'interruption sur TOF

- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption**
 - Programmation du vecteur
 - Initialisations
 - Programme principal**
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

2 - Programme principal

- Réalise la "tâche de fond" du système
- Au minimum, boucle infinie :
- En assembleur :

```
1  PP    bra    PP
```

- en C

```
1  int main()  
2  {  
3  // initialisations  
4      ...  
5  // programme principal  
6      while(1)  
7          ;  
8  }
```

- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption**
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)**
- 5 Exemples : application au Timer

3 - Routine d'interruption (ISR)

- Il peut en exister autant que le système a d'événements à gérer.
- 2 tâches à prévoir :
 - Acquiesement de l'interruption
⇒ ré-initialisation du bit d'état correspondant.
 - Traitement de l'évènement.
- En général, temps d'exécution **court**, peu de lignes.
Jamais d'attente !
- Se termine par une instruction "Retour de routine d'interruption"
(`rti` sur le 9s12)
Insérée automatiquement par le compilateur si programmation en C.

Implémentation ISR en C

- Le langage C ne prévoit pas la gestion des interruptions. \Rightarrow chaque compilateur les gère différemment.
- la routine d'interruption est une fonction :
 - qui ne prendra pas d'argument,
 - qui ne renverra pas de valeur.
- La **déclaration** de la fonction est faite dans un fichier d'en-tête spécial (nom prédéfini).
- La **définition** se fera dans votre fichier source :

```
1 void isr_xxxx() // nom fonction de l interruption
2 {
3 //   ici , mon code...
4 }
```

- 1 Introduction
- 2 Principes du fonctionnement en interruptions
 - Mémorisation
 - Séquencement d'une interruption
 - Vecteurs d'interruptions
- 3 Les interruptions sur le 9s12
- 4 Structure d'un programme en interruption
 - Programmation du vecteur
 - Initialisations
 - Programme principal
 - Routine d'interruption (ISR)
- 5 Exemples : application au Timer

Exemple 1 : génération de signal sonore sur PT0

- On utilise le timer en mode "comparateur" (canal 0), avec la broche de sortie en mode "toggle".
- fonction main() :

```
1  #define DEMI_PERIODE 1500
2  int main (void)
3  {
4  // 1 – Initialisations
5      TSCR1 = BIT7;      // mise en route timer
6      TSCR2 = 3;        // sélection division freq (R=  )
7      TIOS =           ; // broche PT0 en sortie
8      TCTL2 =          ; // mode toggle PT0
9      TIE =            ; // validation locale interr .
10     asm("cli");       // validation globale (en dernier !)
11
12 // 2 – programme principal
13     while( 1 )
14         ;
15 }
```

Exemple 1 : ISR

```
1 // 3 – routine d interruption du canal 0 du Timer
2 void isr_ect_0()
3 {
4     TFLG1 = TFLG1 | BIT0; // RAZ flag
5     TCO = TCO + DEMI_PERIODE; // addition dans le registre du
6     // comparateur
7 }
```

Exemple 1 : ISR

```
1 // 3 – routine d interruption du canal 0 du Timer
2 void isr_ect_0()
3 {
4     TFLG1 = TFLG1 | BIT0; // RAZ flag
5     TCO = TCO + DEMI_PERIODE; // addition dans le registre du
6     // comparateur
7 }
```

- Question : quel est la fréquence du signal généré ?

Exemple 1 : ISR

```
1 // 3 – routine d interruption du canal 0 du Timer
2 void isr_ect_0()
3 {
4     TFLG1 = TFLG1 | BIT0; // RAZ flag
5     TCO = TCO + DEMI_PERIODE; // addition dans le registre du
6     // comparateur
7 }
```

- Question : quel est la fréquence du signal généré ?
- $f = 1/T = 1 / (2 \cdot T/2)$

Exemple 2 : clignotement de DEL connectée sur PB6

- On souhaite une période $t_{ON} = t_{OFF} = 200ms$.
- On utilise le timer en mode "comptage TOF".
- On choisit $R=2$ ($T_{MAX} = \text{-----}$)
- Principe : la routine sera exécutée **automatiquement** tout les " T_{MAX} ", et devra :
 - ① réinitialiser le flag (-----),
 - ② décrémenter un compteur,
 - ③ si compteur = 0, inverser la DEL et réinitialiser le compteur.

Exemple 2 : clignotement de DEL connectée sur PB6

- On souhaite une période $t_{ON} = t_{OFF} = 200ms$.
- On utilise le timer en mode "comptage TOF".
- On choisit $R=2$ ($T_{MAX} = \text{-----}$)
- Principe : la routine sera exécutée **automatiquement** tout les " T_{MAX} ", et devra :
 - ❶ réinitialiser le flag (-----),
 - ❷ décrémenter un compteur,
 - ❸ si compteur = 0, inverser la DEL et réinitialiser le compteur.

```
1 void isr_ect_tof()
2 {
3     TFLG2 = -----; // RAZ TOF
4     compt_TOF--;
5     if( compt_TOF == 0 )
6     {
7         PORTB = PORTB -----; // inversion bit PB6
8         compt_TOF = NBCYCLES; // ré-init compteur
9     }
10 }
```

Exemple 2 : Programme principal

```
1  #define NBCYCLES -----
2
3  int compt_TOF = NBCYCLES;
4
5  int main()
6  {
7      DDRB = -----;
8      TSCR1 = -----;
9      TSCR2 = -----;
10     asm( "cli" );    // autorisation globale
11     while( 1 )
12         ;
13 }
```

- Remarque : le compteur est obligatoirement déclaré en **variable globale**, de façon à être accessible dans la routine d'interruption.

Exemple 3 : attente de la fin d'un délai

- Dans certains cas, on a besoin d'attendre la fin d'un délai dans un programme,

Exemple 3 : attente de la fin d'un délai

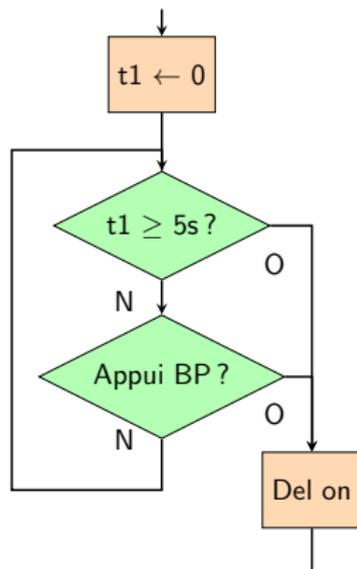
- Dans certains cas, on a besoin d'attendre la fin d'un délai dans un programme, **mais** tout en ayant besoin de tester d'autres évènements extérieurs en même temps.

Exemple 3 : attente de la fin d'un délai

- Dans certains cas, on a besoin d'attendre la fin d'un délai dans un programme, **mais** tout en ayant besoin de tester d'autres évènements extérieurs en même temps.
- Exemple : *Attendre 5s. pour allumer une Del, sauf si on appuie sur un BP, auquel cas on allume la Del tout de suite*

Exemple 3 : attente de la fin d'un délai

- Dans certains cas, on a besoin d'attendre la fin d'un délai dans un programme, **mais** tout en ayant besoin de tester d'autres évènements extérieurs en même temps.
- Exemple : *Attendre 5s. pour allumer une Del, sauf si on appuie sur un BP, auquel cas on allume la Del tout de suite*



Exemple 3 : solution

La solution consiste à garder active une interruption sur TOF, qui se contente d'incrémenter un compteur (var. globale) :

```
1 void isr_ect_tof()
2 {
3     TFLG2 = 0x80; // raz TOF
4     compt_TOF++;
5 }
```

Exemple 3 : solution

La solution consiste à garder active une interruption sur TOF, qui se contente d'incrémenter un compteur (var. globale) :

```
1 void isr_ect_tof()
2 {
3     TFLG2 = 0x80; // raz TOF
4     compt_TOF++;
5 }
```

- On pourra alors **utiliser** ce compteur dans son programme comme indicateur du temps qui passe.

Exemple 3 : solution

La solution consiste à garder active une interruption sur TOF, qui se contente d'incrémenter un compteur (var. globale) :

```
1 void isr_ect_tof()
2 {
3     TFLG2 = 0x80; // raz TOF
4     compt_TOF++;
5 }
```

- On pourra alors **utiliser** ce compteur dans son programme comme indicateur du temps qui passe.

```
1 ...
2 compt_TOF = 0; // RAZ du compteur
3 while( compt_TOF < DELAI_5S && (PORTB&BIT6) != 0 )
4     ; // rien
5 PORTB = PORTB | BIT7; // allumer la del sur PB7
6 ...
```

Exemple 3 : détermination du délai

- La constante DELAI_5S est fonction du choix de R
- Exemple : si R= 32 :

$$T_{MAX} = \frac{\quad}{24MHz} = \quad \text{ms}$$

$$\Rightarrow \text{DELAI_5S} = \frac{\quad}{\frac{ms}{ms}} = \quad$$

Exemple initial : clavier 3 BP en interruptions

- Il faut parfois modifier le *hardware* pour pouvoir mettre en oeuvre les interruptions.
- Exemple : sur architecture 9s12, on ne peut pas déclencher d'interruptions sur un changement de niveaux des ports d'E/S : il faut ajouter un opérateur pour générer le signal d'interruption à partir des niveaux sur les BP.

