Electronique & Temps réel

- **Master 2** « avionique » semestre 10 2007

- Application au micro-contrôleur HC12

Denis Michaud

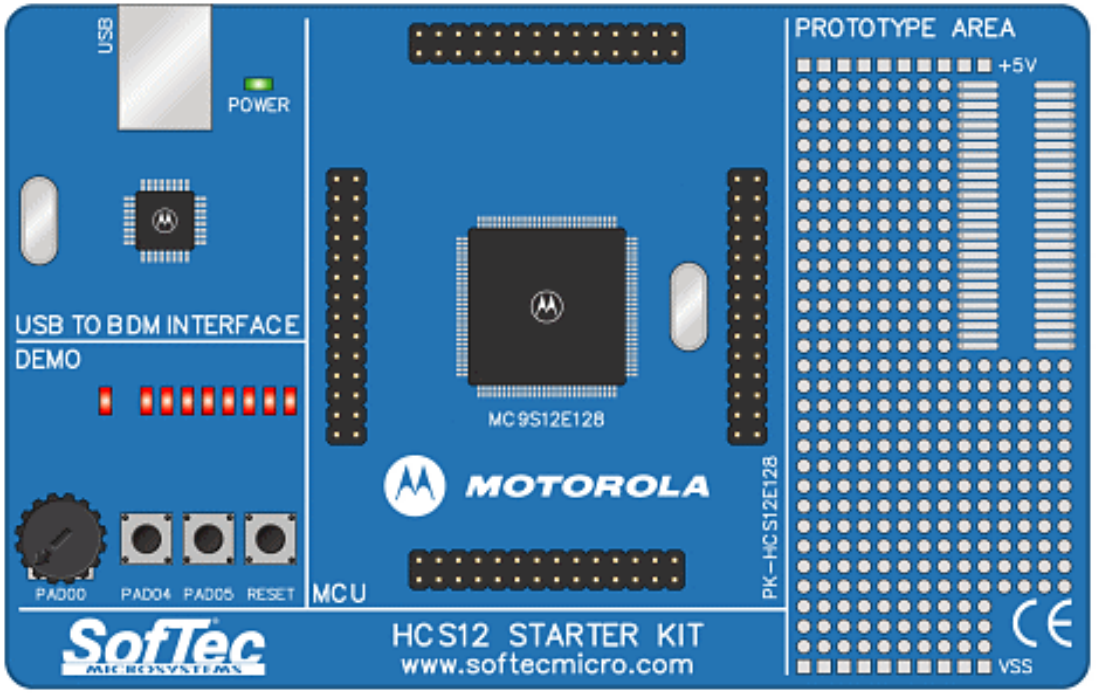
Merci à Rachid MALTI

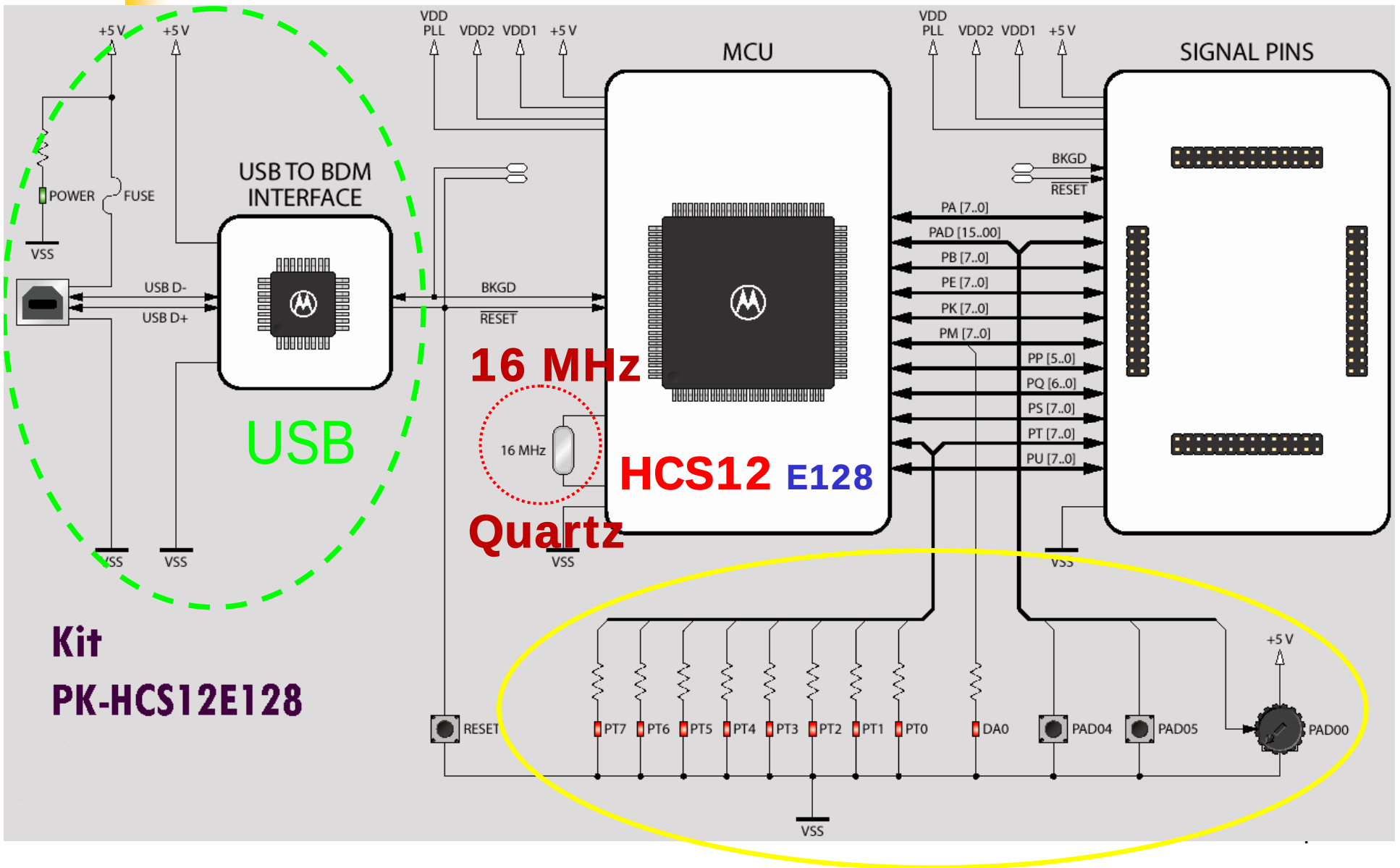
Plan

- Pré requis, Définitions , temps réel...
- Interruptions comparées au pooling
- Principe de fonctionnement des interruptions
- Différents types d'interruption
- Connexion des périphériques aux lignes d'interruption
- Vectorisation d'interruption
- Exemple : interruption d'horloge (HC12)

Pré requis

- Cours test automatique Master 1
- Langage C et Metrowerks CodeWarrior
- Structure HC12
- Kit pk-hcs12e128




+5 V
+5 V
POWER
FUSE
VSS
USB TO BDM
INTERFACE
USB D-
USB D+
USB
VSS
VSS
VDD
PLL
VDD2
VDD1
+5 V
MCU
BKGD
RESET
16 MHz
16 MHz
Quartz
VSS
HCS12 E128
VDD
PLL
VDD2
VDD1
+5 V
SIGNAL PINS
BKGD
RESET
PA [7..0]
PAD [15..00]
PB [7..0]
PE [7..0]
PK [7..0]
PM [7..0]
PP [5..0]
PQ [6..0]
PS [7..0]
PT [7..0]
PU [7..0]
VSS
Kit
PK-HCS12E128
RESET
PT7
PT6
PT5
PT4
PT3
PT2
PT1
PT0
DA0
PAD04
PAD05
+5 V
PAD00
VSS

Définitions : **Système temps réel:**

En informatique industrielle, on parle d'un système **temps réel** lorsque ce système informatique contrôle (ou pilote) un procédé physique à une vitesse adaptée à l'évolution du procédé contrôlé.

- Les systèmes temps réel sont des dispositifs constitués de matériels et de logiciels soumis à des contraintes à la fois fonctionnelles et temporelles pour réaliser des traitements, et agir sur leur environnement.
- Un système temps réel est une association logiciel /matériel où le logiciel permet d'voir une gestion adéquate des ressources matérielles en vue de remplir certaines tâches suivant une séquence prédéfinie.
- Des exemples de domaines où on rencontre de tels systèmes sont les télécommunications, le nucléaire, l'avionique ou le médical.
- Ces systèmes sont souvent critiques à cause d'enjeux humains et économiques importants.
- Leur développement exige donc des méthodes très fiables.
- La partie du logiciel qui réalise cette gestion est le système d'exploitation ou le **noyau** temps réel.

Définitions : **Système temps réel:** (suite)

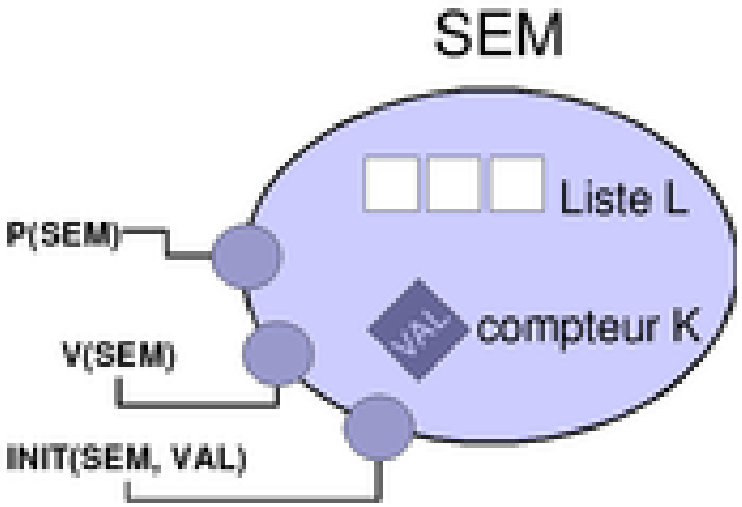
- Ce noyau temps réel va offrir des services au(x) logiciel(s) d'application. Ces services seront basés sur les ressources disponibles au niveau du matériel.
- Un système temps réel est un système:
  - Qui inter-réagit avec un environnement externe qui lui-même évolue avec le temps.
  - Qui réalise certaines fonctionnalités en relation avec cet environnement
  - Qui exploite des ressources limitées.
- Deux contraintes sont à vérifier pour qu'un système temps réel soit fonctionnel:
  - Exactitude logique (logical correctness) : Sorties adéquates en fonction des entrées, assurant le comportement désiré suite à des événements et aux données communiquées, ...
  - Exactitude temporelle (timeliness) : les sorties sont présentées au bon moment.

Qu’est-ce qu’un système multitâche ?

- Les notions liées aux systèmes multitâches sont:
  - Programmation concurrente
  - Partage de ressources
  - Exclusion mutuelle
  - Notions de priorités
  - Synchronisation
  - Communication interprocessus.

Définitions : **Sémaphore (informatique):**

- Un **sémaphore** est une variable protégée (ou un type de donnée abstrait) et constitue la méthode utilisée couramment pour restreindre l'accès à des ressources partagées (par exemple un espace de stockage) dans un environnement de programmation concurrente. Le sémaphore a été inventé par Edsger Dijkstra et utilisé pour la première fois dans le système d'exploitation THEOS.
- Les sémaphores fournissent la solution la plus courante pour le fameux problème du « dîner des philosophes », bien qu'ils ne permettent pas d'éviter tous les interblocages (ou *deadlocks*).



Définitions : **Chien de garde (informatique):**

- En matériel informatique, un **chien de garde** (***watchdog*** en anglais) est un mécanisme électronique ou logiciel destiné à s'assurer qu'un automate ne s'est pas bloqué à une étape particulière du traitement. C'est une protection destinée à redémarrer le système si une action définie n'est pas exécutée dans un délai donné.

- Quand il est réalisé par logiciel, il s'agit en général d'un compteur qui est régulièrement remis à zéro. Si le compteur dépasse une valeur donnée (*timeout*) alors on procède à un *reset* (redémarrage) du système. Le chien de garde consiste souvent en un registre qui est mis à jour via une interruption régulière. Il peut également consister en une routine d'interruption qui doit effectuer certaines tâches de maintenance avant de redonner la main au programme principal.

- Les chiens de garde sont souvent intégrés dans les microcontrôleurs et dans les cartes mère dédiées au temps réel.

Définitions

Spécificités :

- Pour garantir le respect de limites ou contraintes temporelles, il est nécessaire que :
- les différents services et algorithmes utilisés s'exécutent en temps borné. Un système d'exploitation temps réel doit ainsi être conçu de manière à ce que les services qu'il propose (accès hardware, etc.) répondent en un temps borné ;
- les différents enchaînements possibles des traitements garantissent que chacun de ceux-ci ne dépassent pas leurs limites temporelles. Ceci est vérifié à l'aide du test d'acceptabilité.

Tâches :

- Une tâche est généralement représentée par un coût (*Ci*), une échéance (*Di*) qui est la date à laquelle la tâche doit avoir terminé son exécution, et dans le cas des tâches périodiques, par une période (*Ti*) qui représente ses instants d'activation. Une exécution de la tâche est appelée une instance.

Définitions

Temps réel strict / souple :

- On distingue le ***temps réel strict*** **ou** ***dur*** **(de l'anglais** ***hard real-time*****) et le** ***temps réel souple*** **ou** ***mou*** **(*soft real-time*)** suivant l'importance accordée aux contraintes temporelles. Le temps réel strict ne tolère aucun dépassement de ces contraintes, ce qui est souvent le cas lorsque de tels dépassements peuvent conduire à des situations critiques, voire catastrophiques : pilote automatique d'avion, système de surveillance de centrale nucléaire, etc. À l'inverse le temps réel souple s'accommode de dépassements des contraintes temporelles dans certaines limites au-delà desquelles le système devient inutilisable : visioconférence, jeux en réseau, etc.
- On peut ainsi considérer qu'un système temps réel strict doit respecter des limites temporelles données même dans la pire des situations d'exécution possibles. En revanche un système temps réel souple doit respecter ses limites pour une moyenne de ses exécutions. On tolère un dépassement exceptionnel, qui sera peut-être rattrapé à l'exécution suivante.

Définitions

Test d'acceptabilité

- Théoriquement, le concepteur d'un système temps réel prétendu strict devrait être capable de prouver que les limites temporelles ne seront jamais dépassées quelque soit la situation. Cette vérification est appelée Test d'acceptabilité, **analyse de faisabilité** ou encore **contrôle d'admission**; elle fait appel à la théorie de l'ordonnancement.
- Elle dépend de l'ordonnanceur utilisé et des caractéristiques des tâches du système. Pour un système souple, on pourra se contenter de mesures statistiques obtenues sur un prototype.

Condition de charge

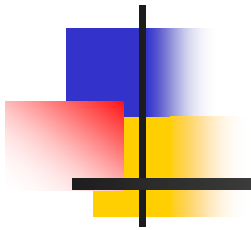
- Pour tout système de n tâches, la condition suivante est nécessaire mais pas suffisante à sa faisabilité:

$$\sum_{i=1}^{n} C_i / D_i \leq 1$$

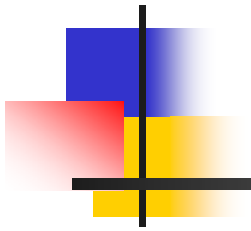
- Avec *Ci* le coût de la tâche et *Di* son échéance relative.
- Une valeur supérieure à 1 signifierait que le système nécessite plus de temps d'exécution que le processeur ne peut en fournir.

Temps de réponse pire cas

- Le temps de réponse dans le pire des cas d'une tâche est, parmi tous les scénarios possible d'exécution du système, la plus longue durée entre l'activation de cette tâche et son instant de terminaison.
- Une tâche est faisable si son temps de réponse dans le pire des cas est inférieur ou égal à son échéance. Un système est faisable si toutes les tâches qui le composent sont faisables.



Les interruptions



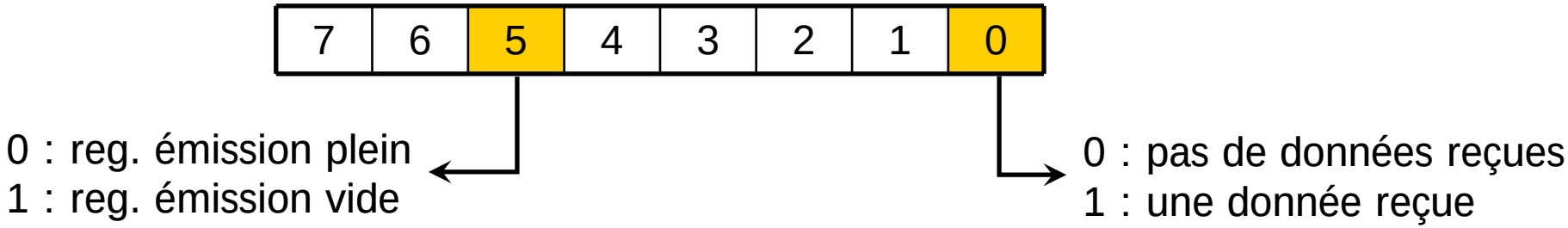
Pourquoi les interruptions ?

Interruption comparée au polling

Exemple du port série d'un PC

Interruption comparée au polling 1/4

- Rappel
  - L'UART contient plusieurs registres permettant dont:
  - Registre réception : @ UART
  - Registre émission : @ UART
  - Registre état ligne @UART + 5 dont voici la description



Interruption comparée au polling 2/4

- Transfert de données – La solution brute

- En Emission

```
Unsigned char TableDonnee[n]
int i
…
main()
{ for (i=0;i<=N;i++)
   {while(inport(AdrIO+5)&0x20==0)
     {}  // on attend
    outport(AdrIO, TableDonnee[i])
}}
```

- En réception

```
Unsigned char TableDonnee[n]
int i
…
main()
{ for (i=0;i<=N;i++)
   {while(inport(AdrIO+5)&0x01==0)
     {}  // on attend
    TableDonnee[i] = inportb(AdrIO)
}}
```

$$T_{transfert} = (T_{envoi} + T_{attente}) \times N$$

Interruption comparée au polling 3/4

- Pour un taux de transfert de 9600 Bauds
  - Durée d'un time-bit ≈ 0.1 ms
  - 1 caractère est reçu/transmis toutes les ms
- Sur un processeur de 1Ghz
  - La durée d'une micro-instruction est de 1ns
  - La durée d'exécution des instructions outport ou inport emission/ réception d'1 caractère peut être majoré à 50ns
  - Le processeur consacre 0.005% de son temps pour envoyer/recevoir un caractère et 99.995% de son temps à attendre la prochaine émission/réception

50 ns

1 000 000 ns = 1 ms

Interruption comparée au polling 4/4

- Transfert de données sous interruption

  - main()

```
{
 initialisation_IT();
 while(!Fin de transfert)
   { //traitement quelconque
    }
}
```

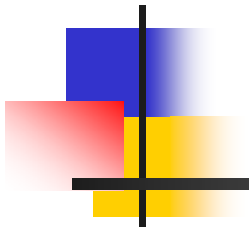
  - Void interrupt transfert()

```
{
if (inport(AdrIO+5)&0x01==1
Tabledonnee[i++]=inport(AdrIO);
}
```

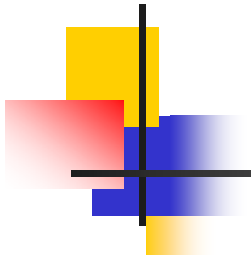
Fonction d'IT déclenchée par l'E/S

Pendant le reste du temps on fait autre chose

50 ns



Principe de fonctionnement des IT

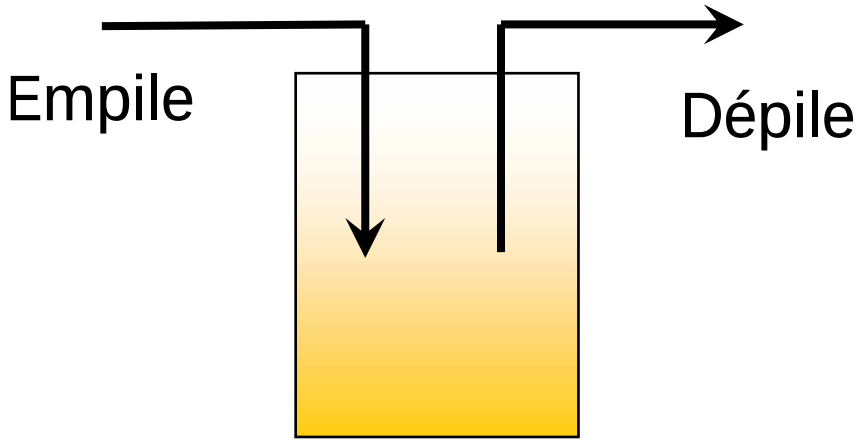


Principe générale d'une interruption

- Le microprocesseur est en train d'exécuter le programme principal ... lorsqu'...
  - une interruption (IT) arrive
- Le microprocesseur termine l'exécution de l'instruction en cours
- Vérifie si le bit d'it est masqué ou validé !
  - Si le bit d'it est masqué le µp ignore l'it et continue l'exécution du prog. principal
  - Si le bit d'it est validé le µp va servir l'it:
    - Il sauvegarde le contexte (registres du µp)
    - Il exécute le programme d'it
    - Il restaure contexte
    - Il reprend l'exécution normale du programme interrompu

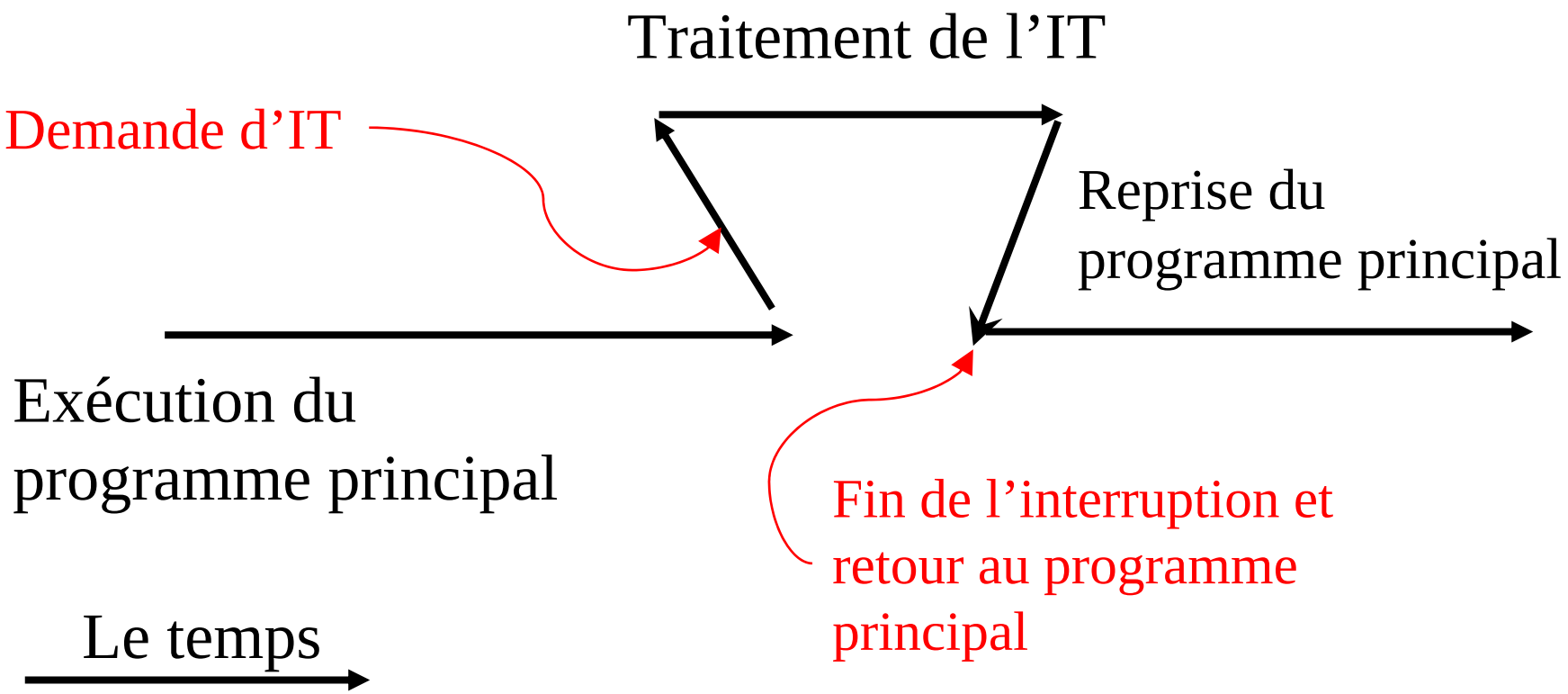
Fonctionnement d'une pile

- Une pile est une file d'attente de type LIFO (Last In First Out)
- Elle permet
  - la sauvegarde du contexte lorsqu'une interruption arrive ➔ Sauvegarde de tous les registres du microprocesseur
  - La restauration du contexte lorsque l'interruption a été servie ➔ Récupération de tous les registres du programme interrompu.

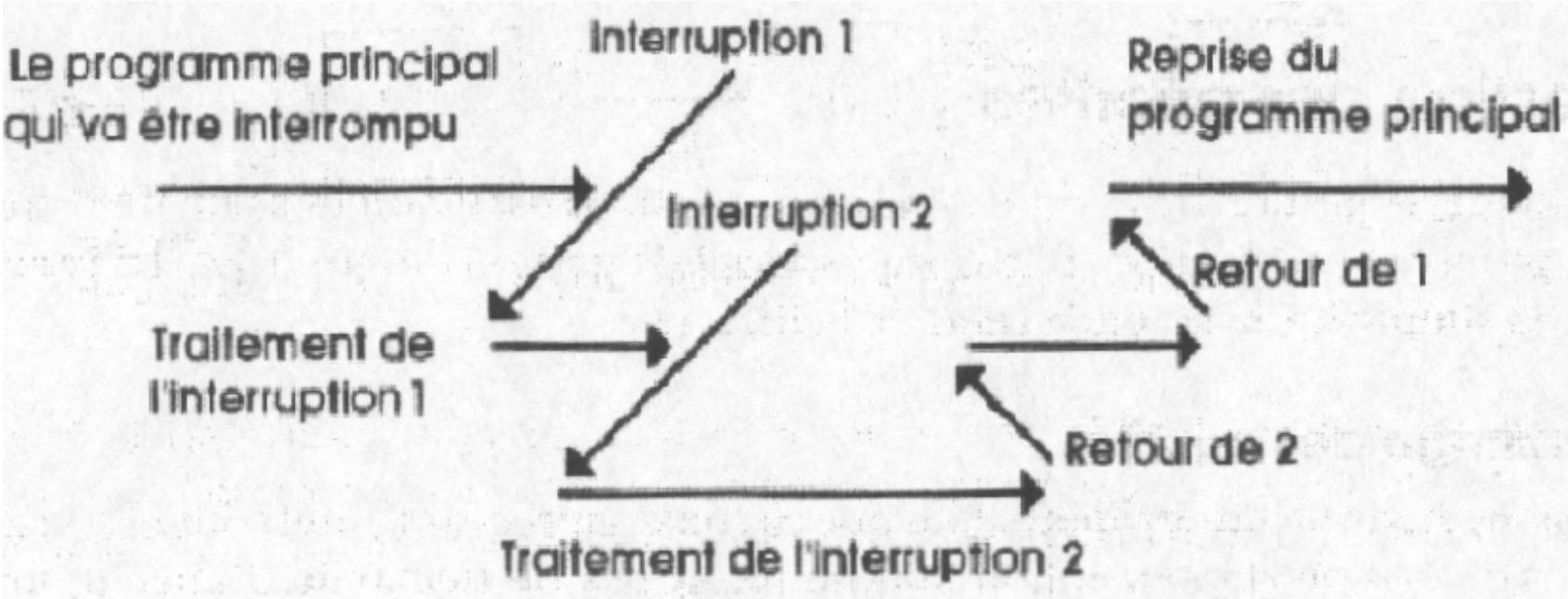


Principe d'exécution d'une interruption

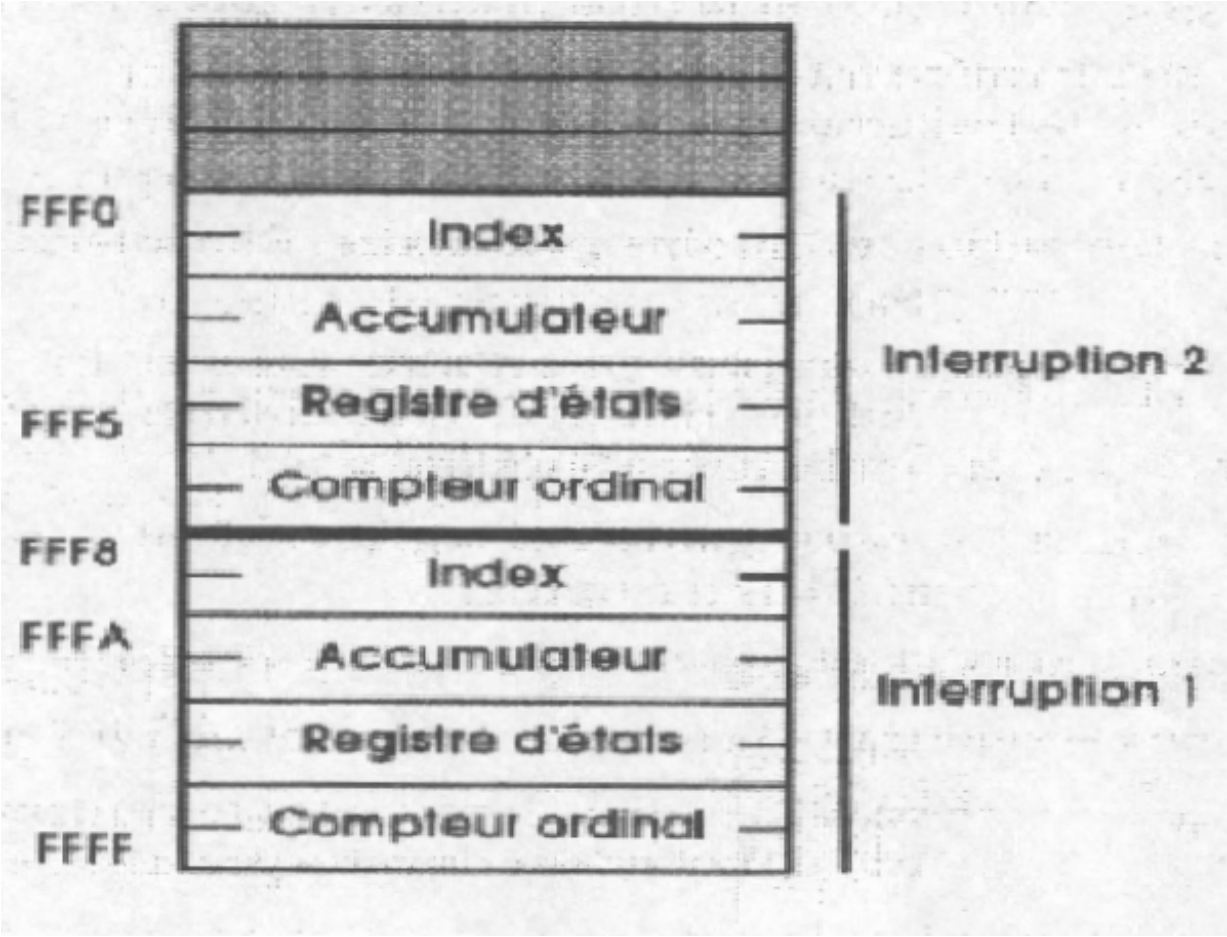
- Le mécanisme d'une exception est voisin de celui d'un appel à un sous-programme.



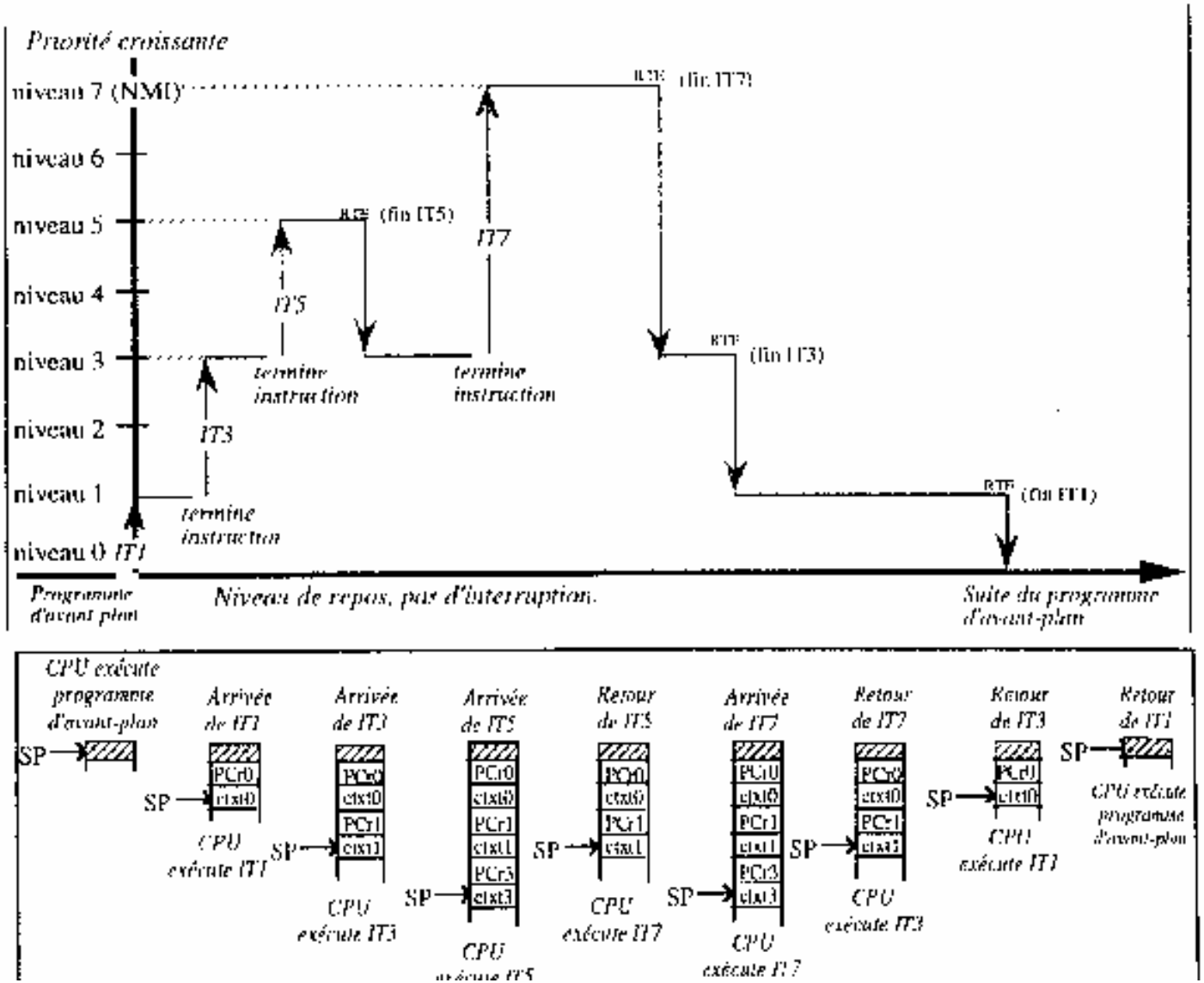
Interruptions multiples



Mouvement de la pile pour lors d'IT multiples



Cascade d'interruption – Evolution de la pile



Programme d'Arrière plan et Interruption

PROGRAMME D'ARRIERE PLAN

Initialisations des périphériques

Initialiser les Interruptions

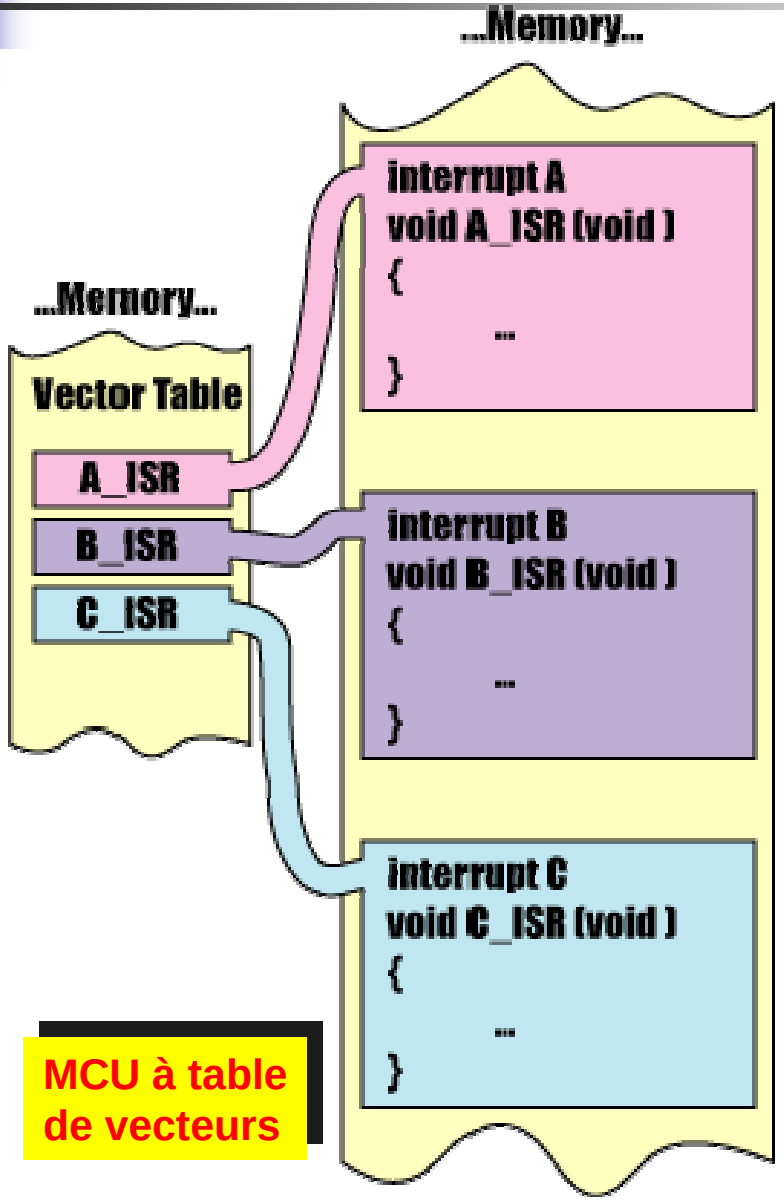
Boucle principale
- **[Automates]**
  ou
- **while (TRUE){}**

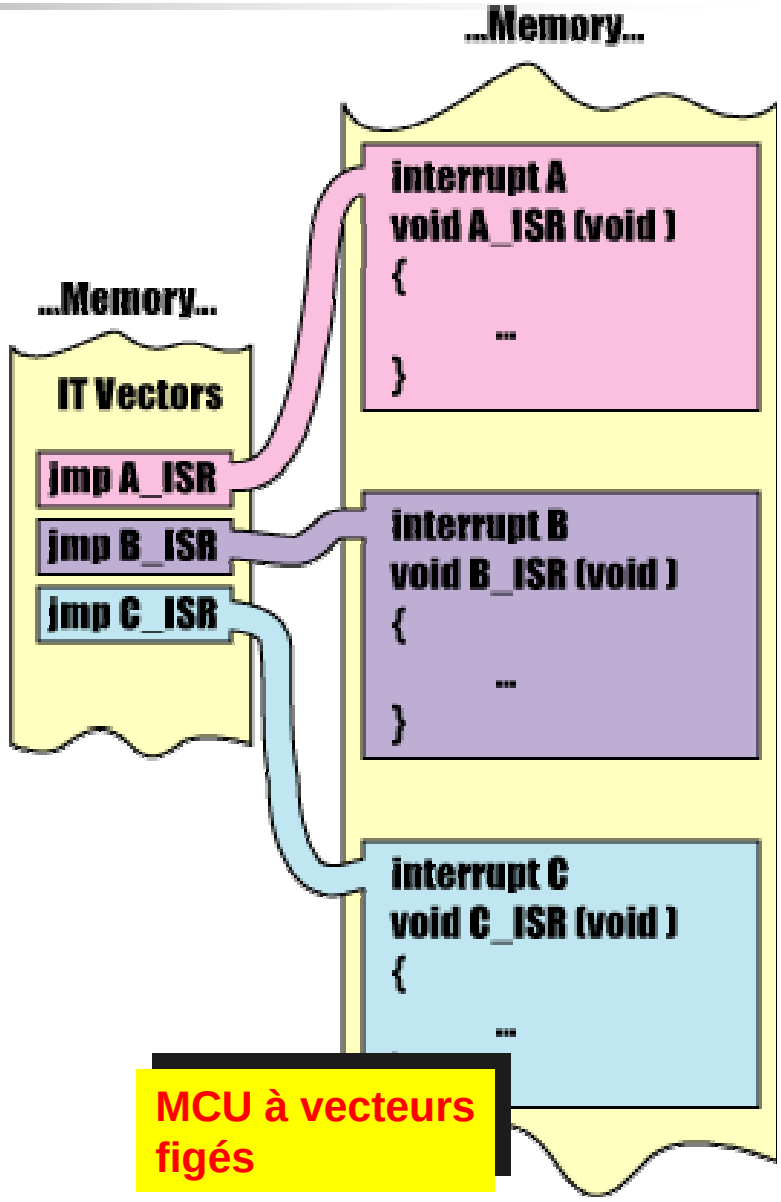
```
interrupt INT1
void INT1_ISR (void){
      //  Traiter l'événement
      //   Acquitter l'interruptio
}
```

```
interrupt INT2
void INT2_ISR (void){
      //  Traiter l'événement
      //   Acquitter l'interruptio
}
```

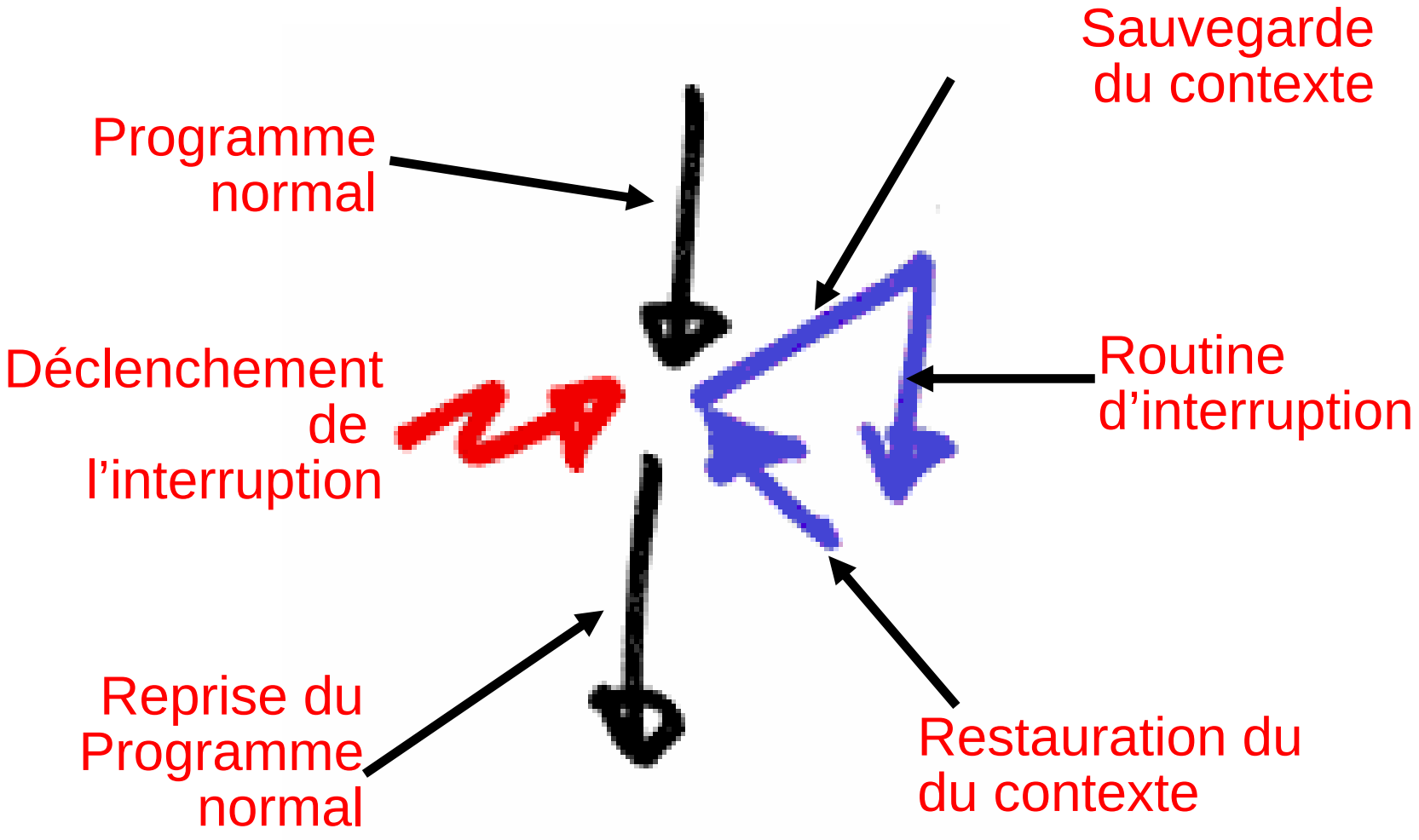
```
interrupt INT3
void INT3_ISR (void){
      //  Traiter l'événement
      //   Acquitter l'interruptio
}
```

Vecteurs d'interruption

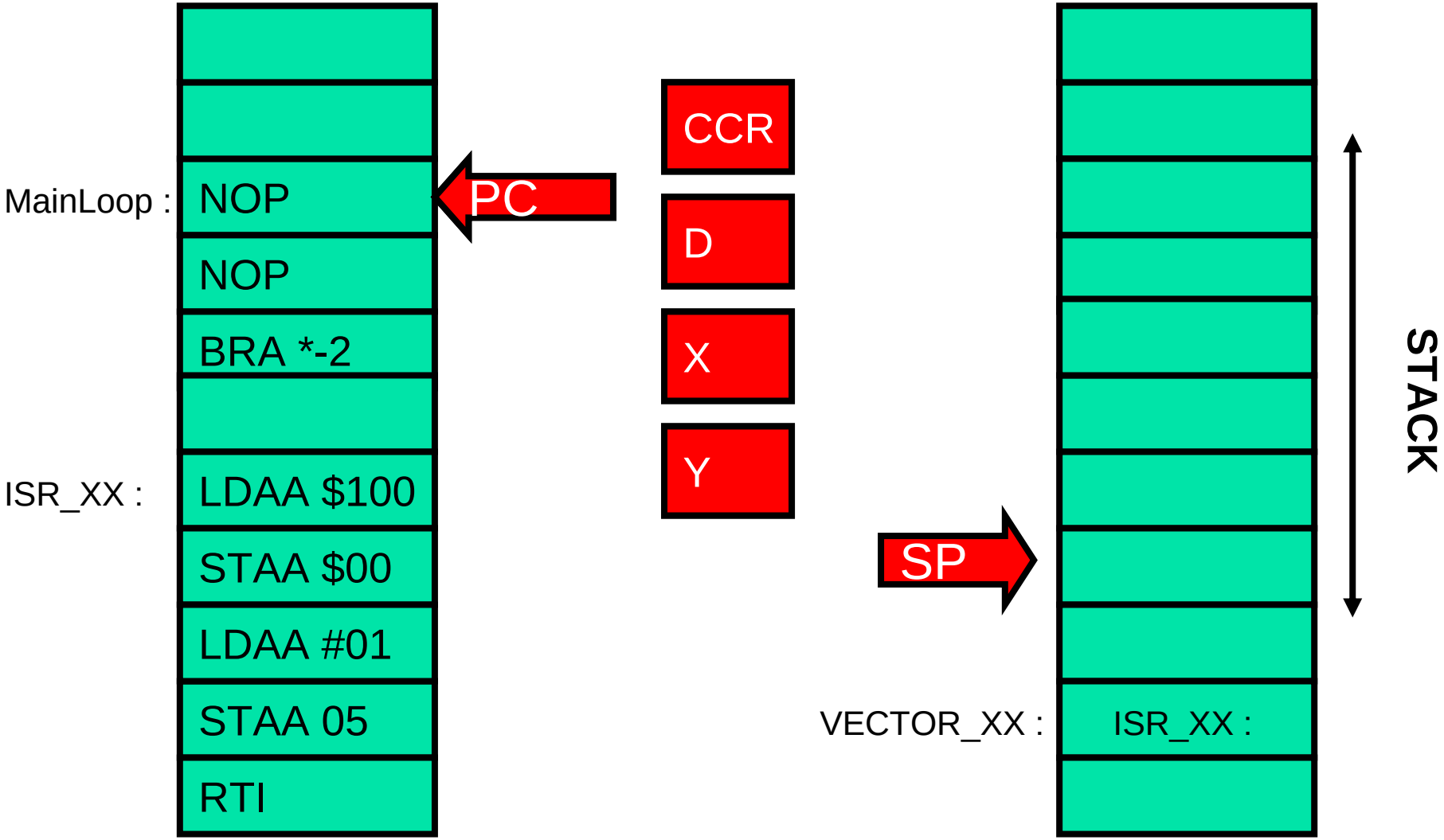




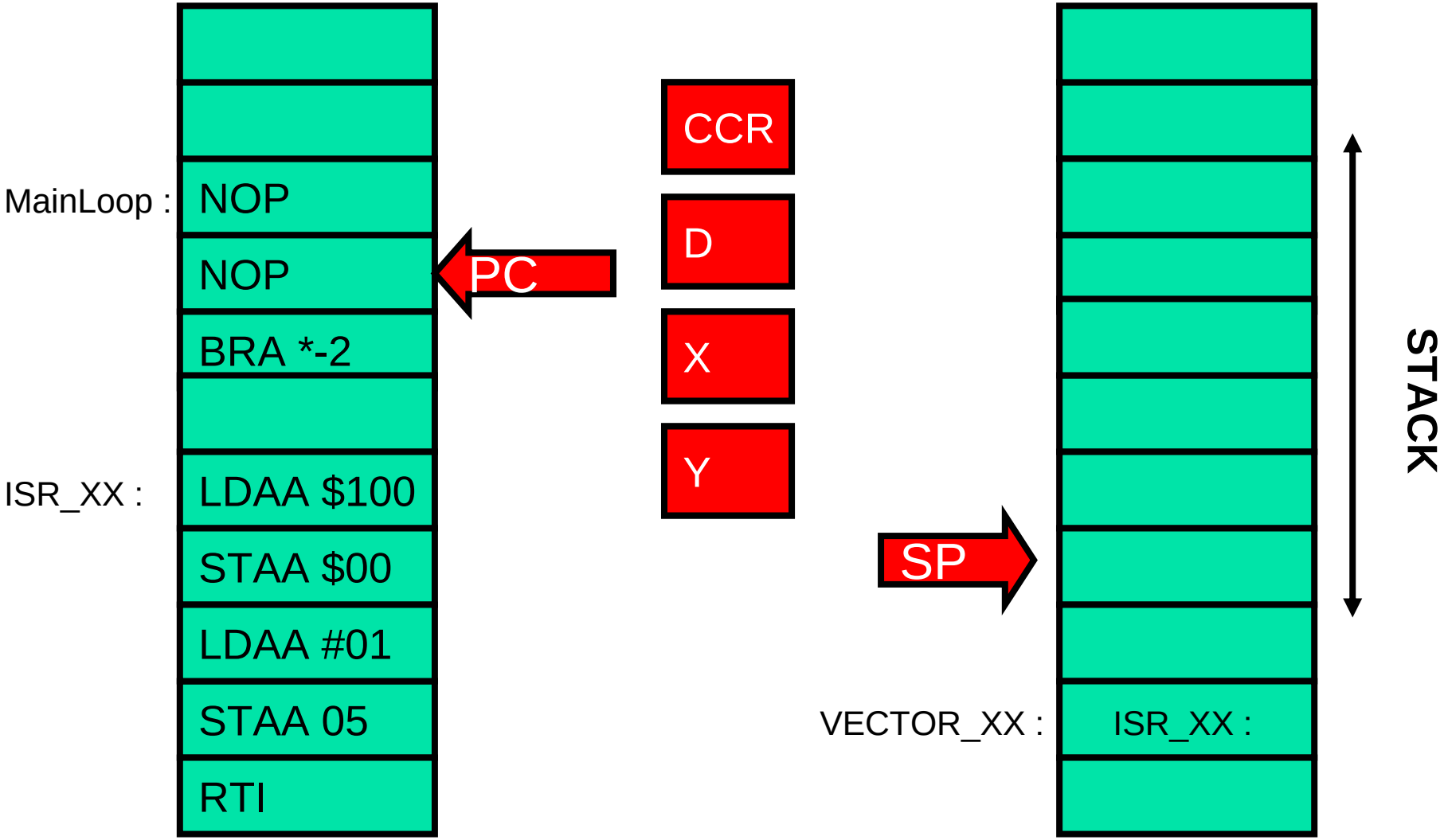
La vie d'une interruption



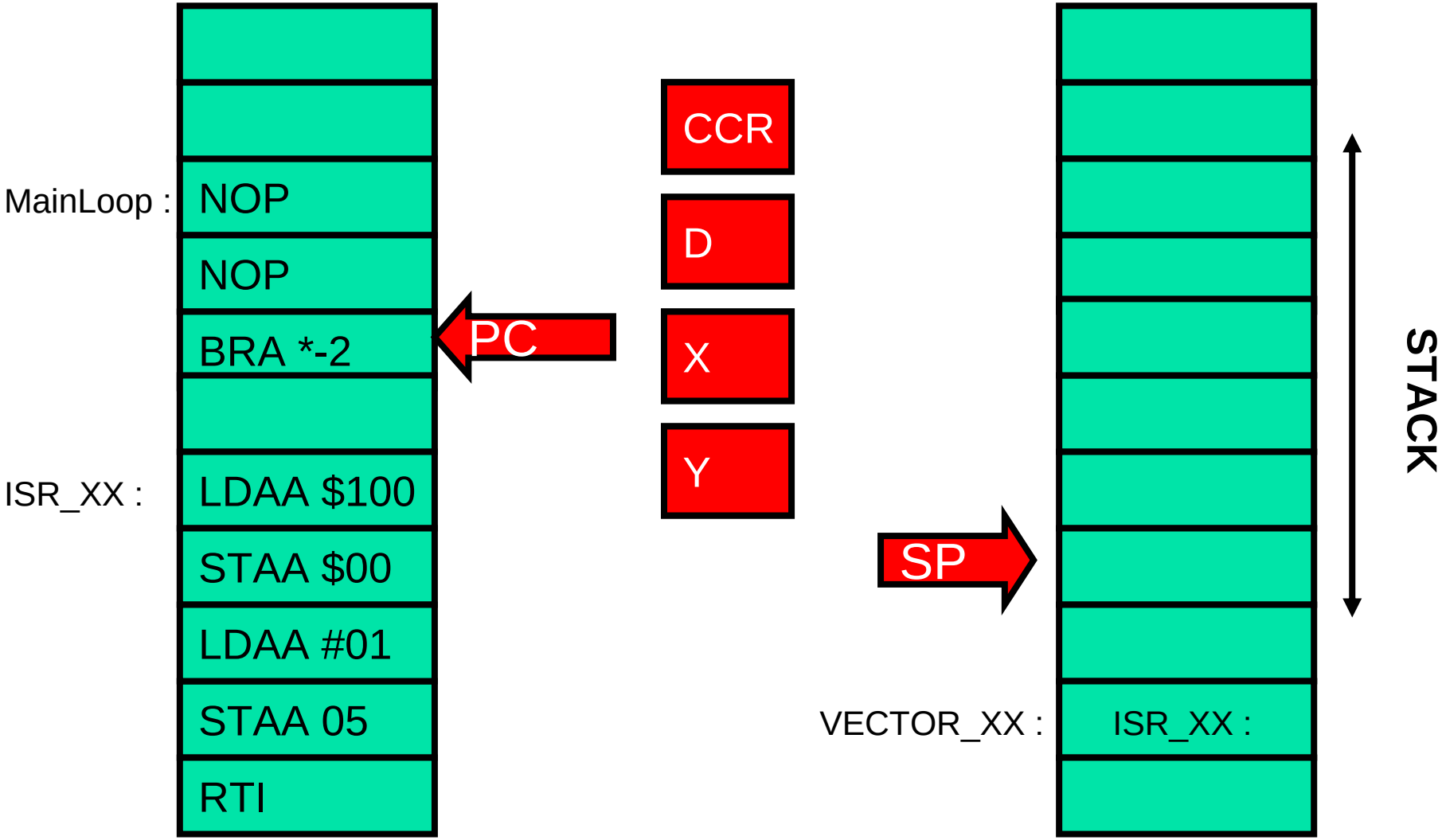
Programme d'arrière plan



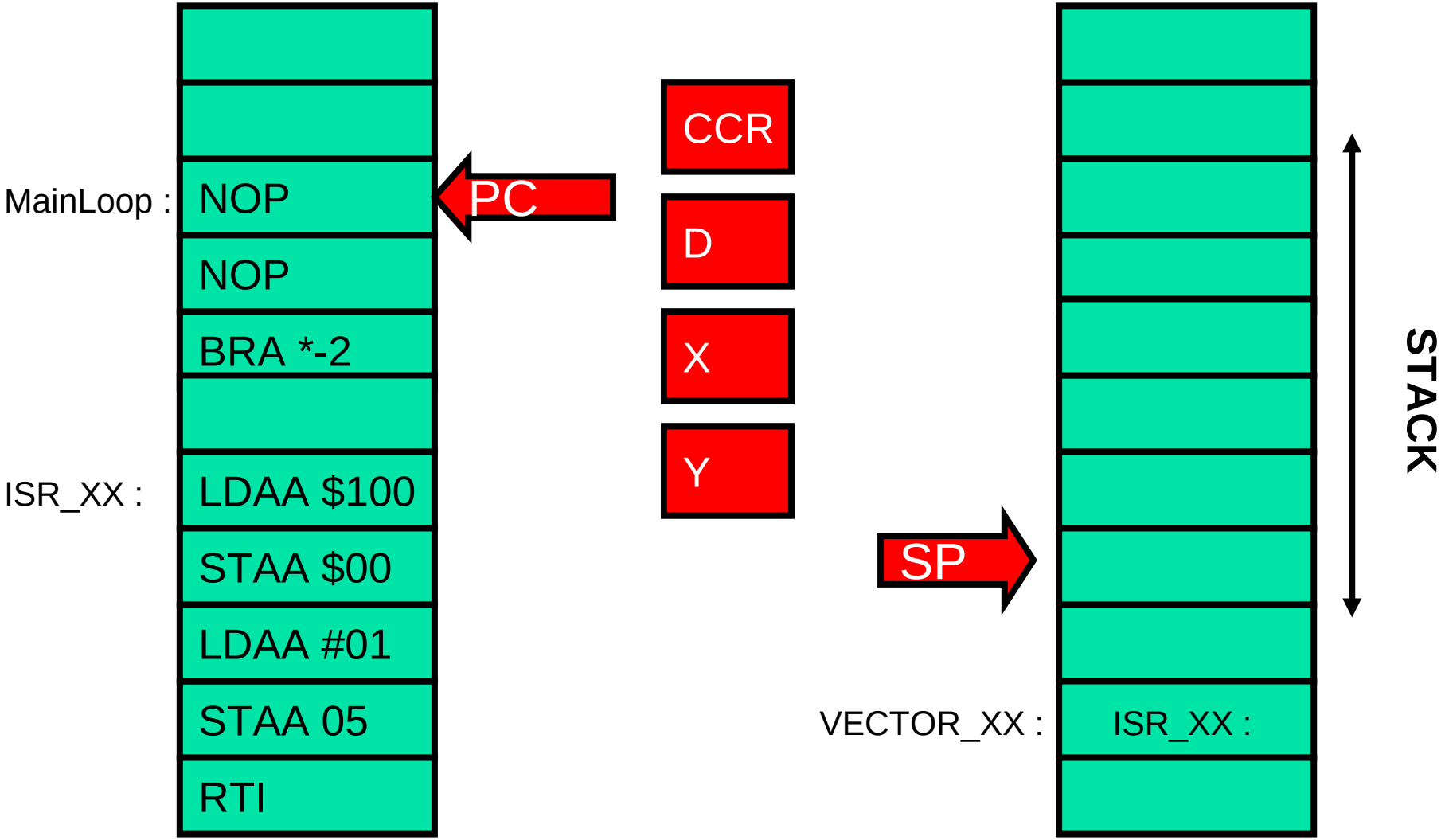
Programme d'arrière plan



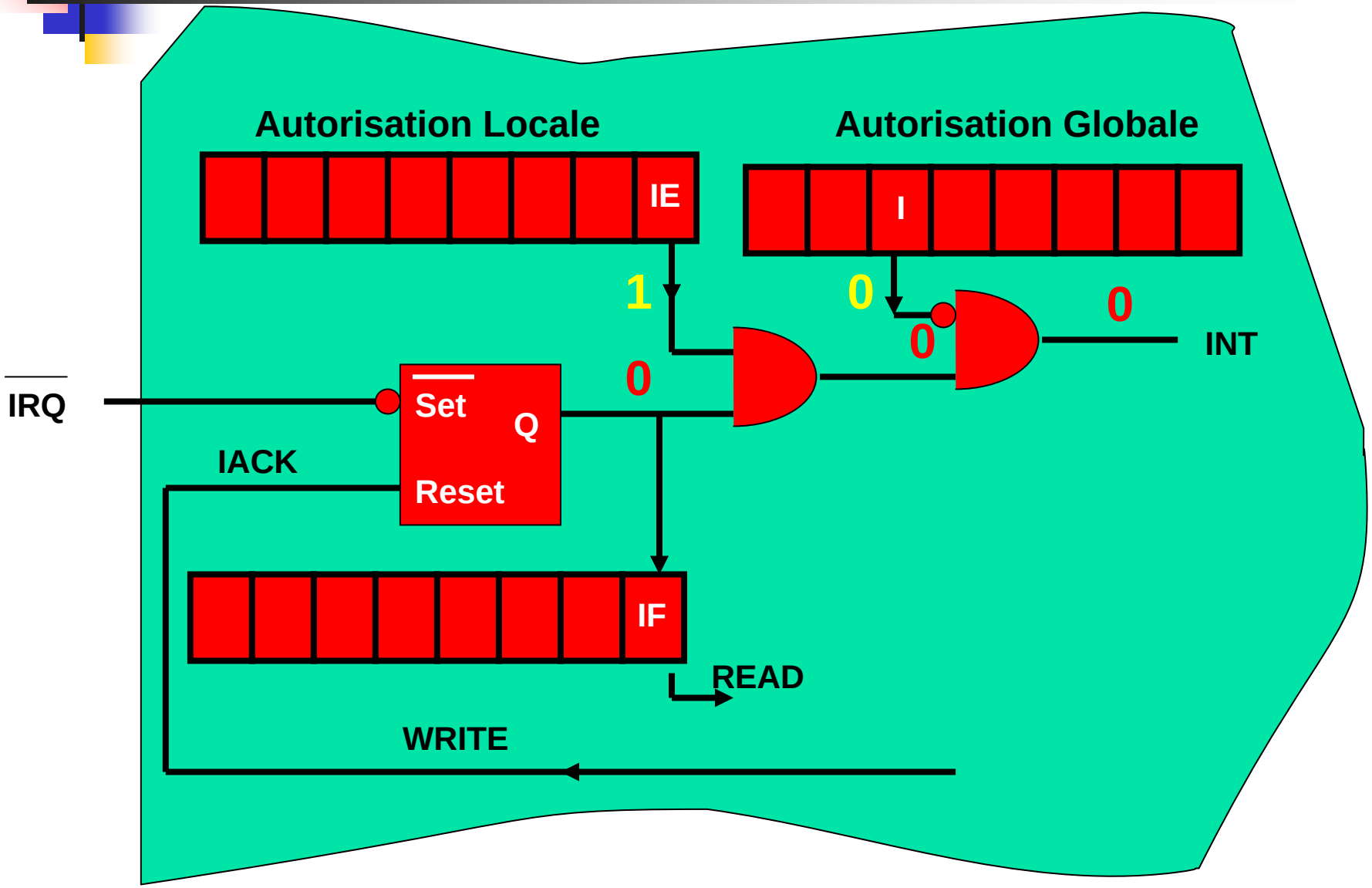
Programme d'arrière plan



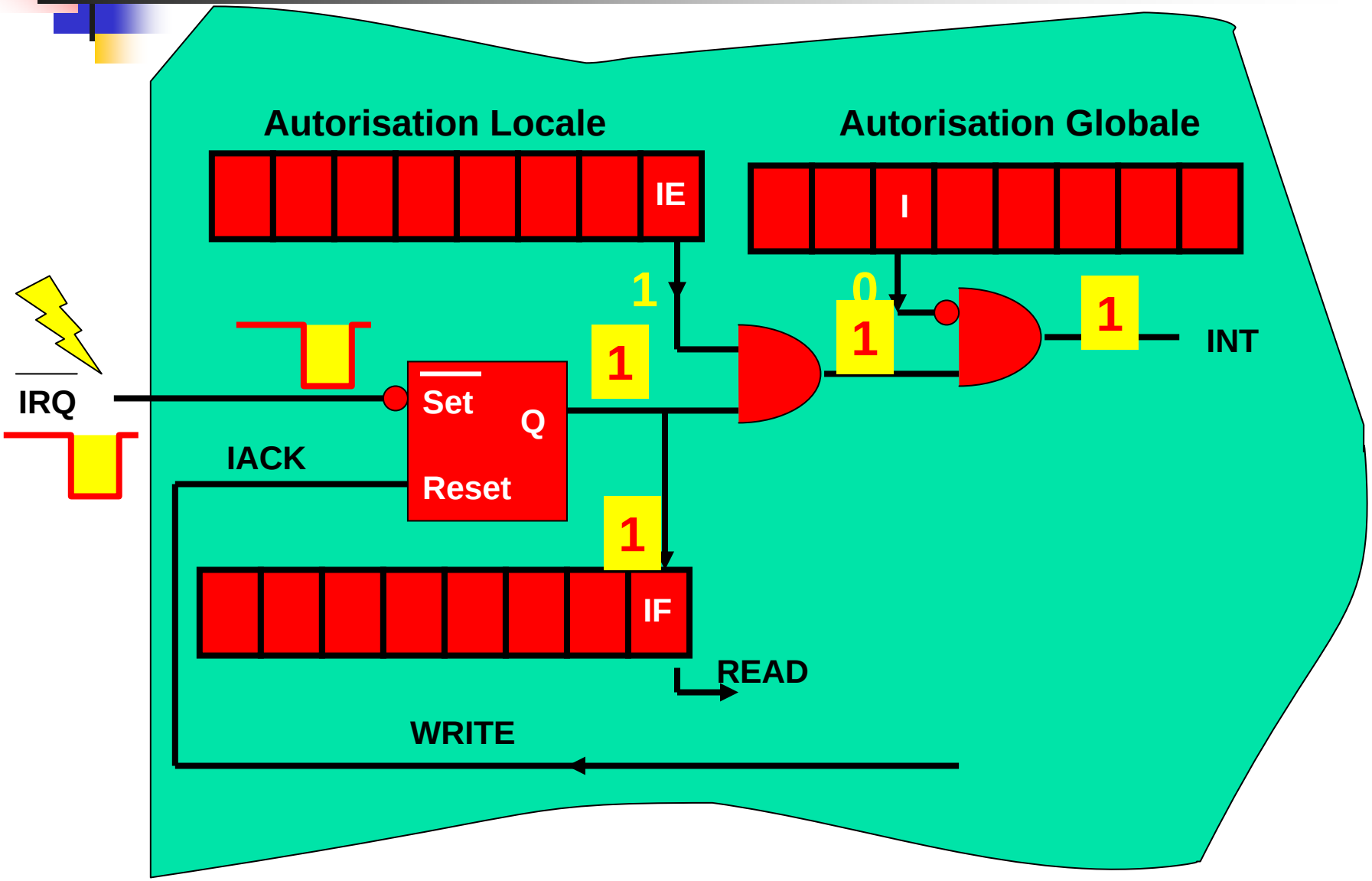
Programme d'arrière plan



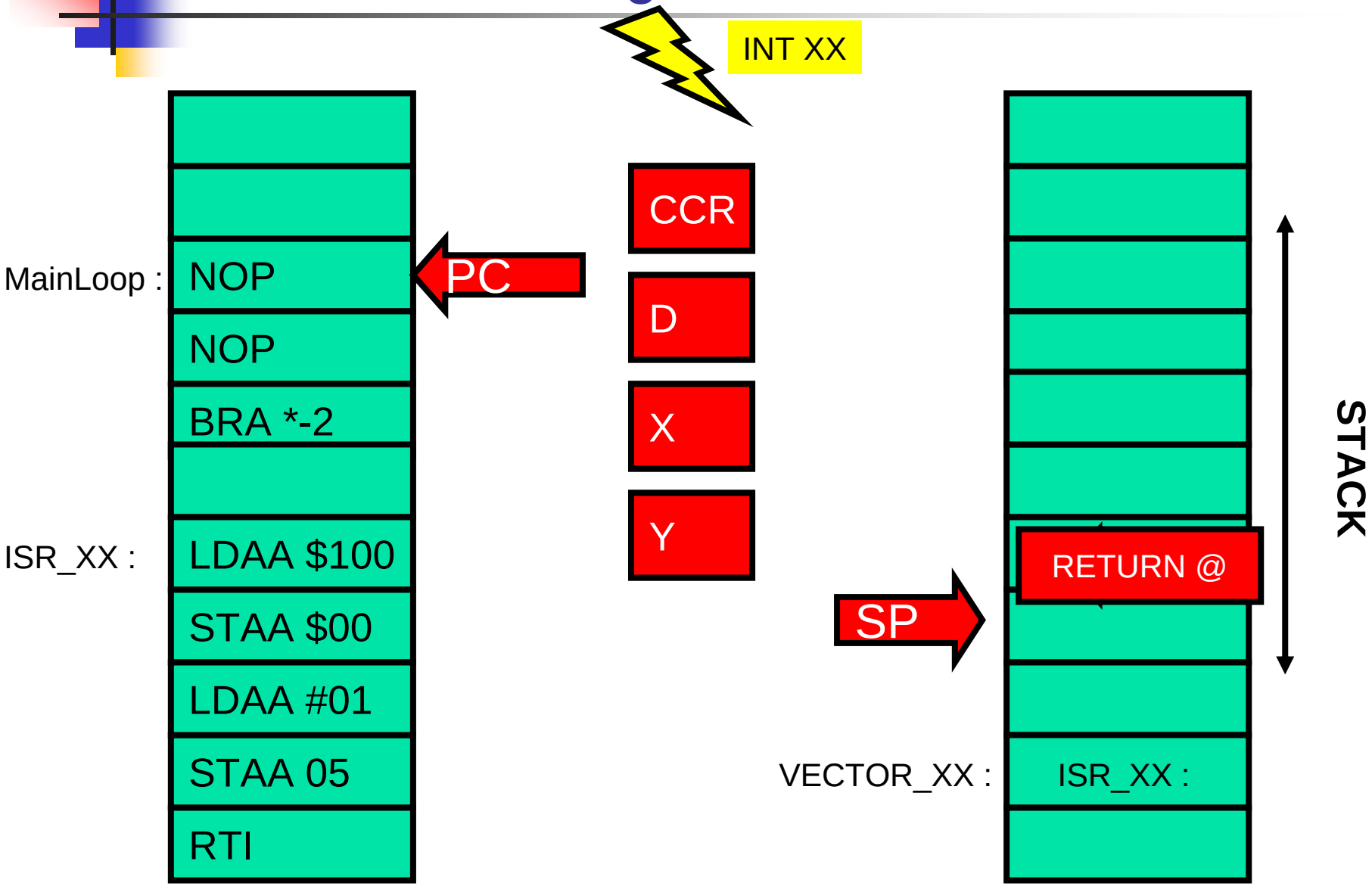
Déclenchement d'une interruption



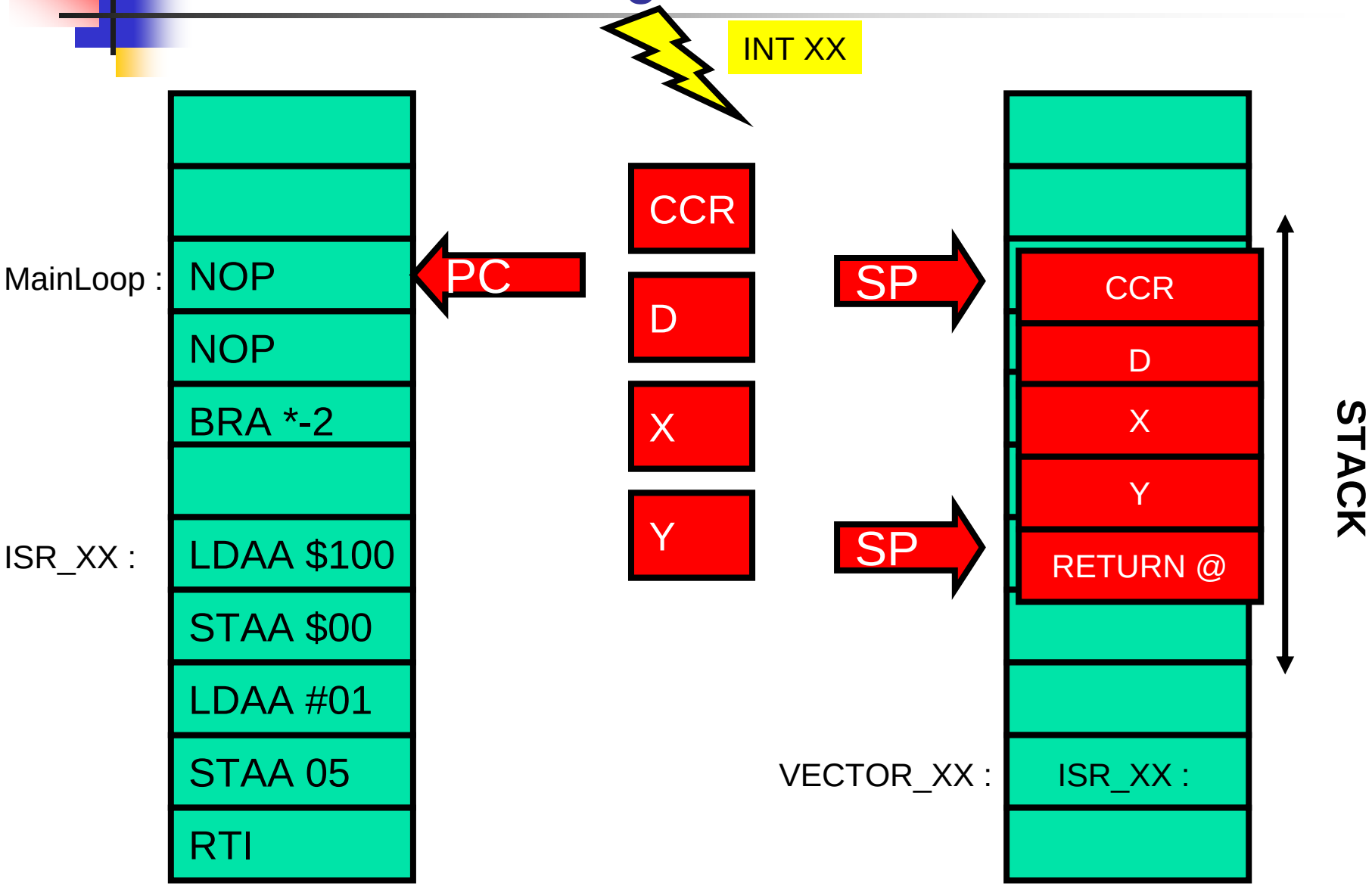
Déclenchement d'une interruption



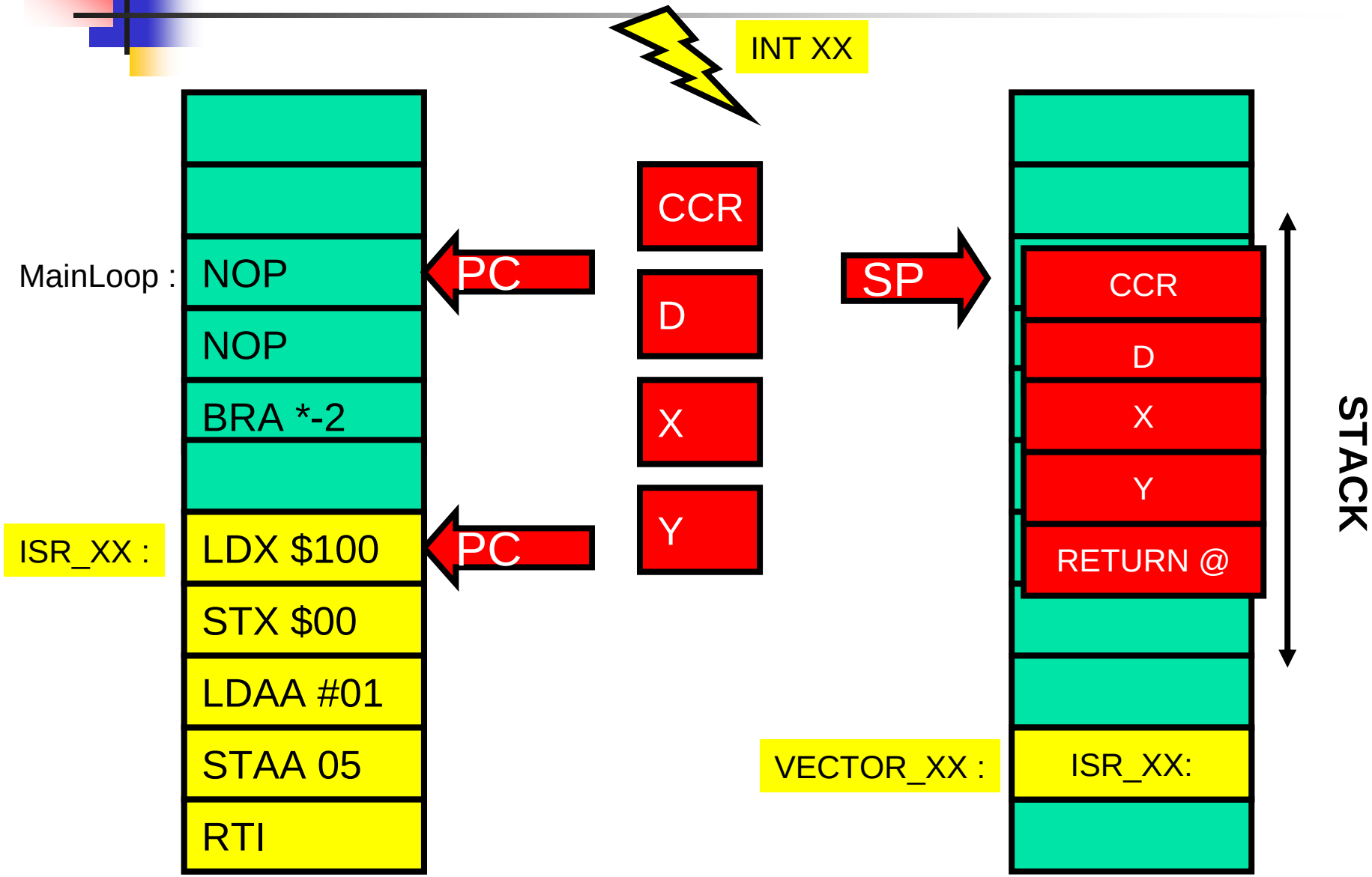
Sauvegarde du contexte



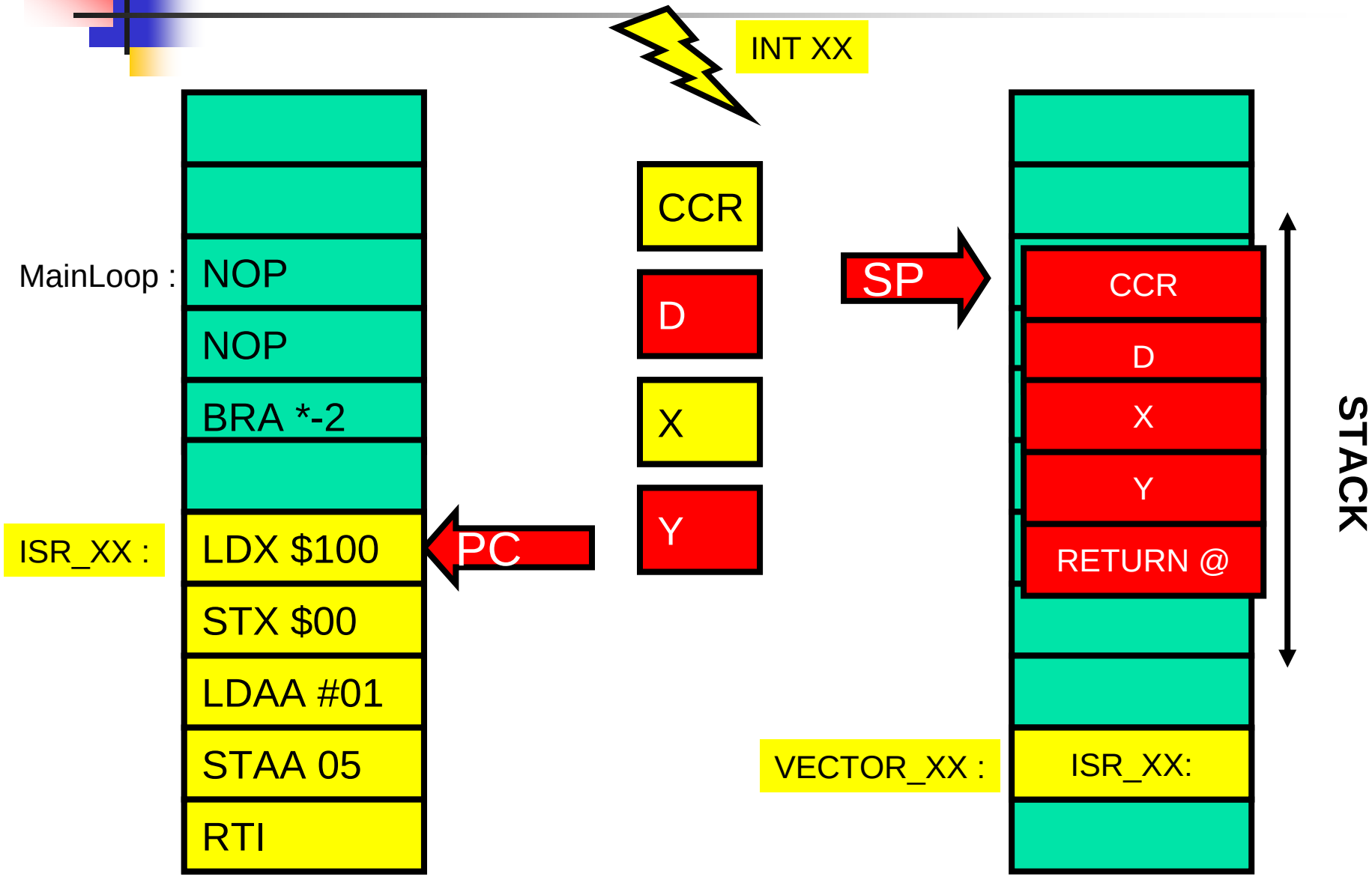
Sauvegarde du contexte



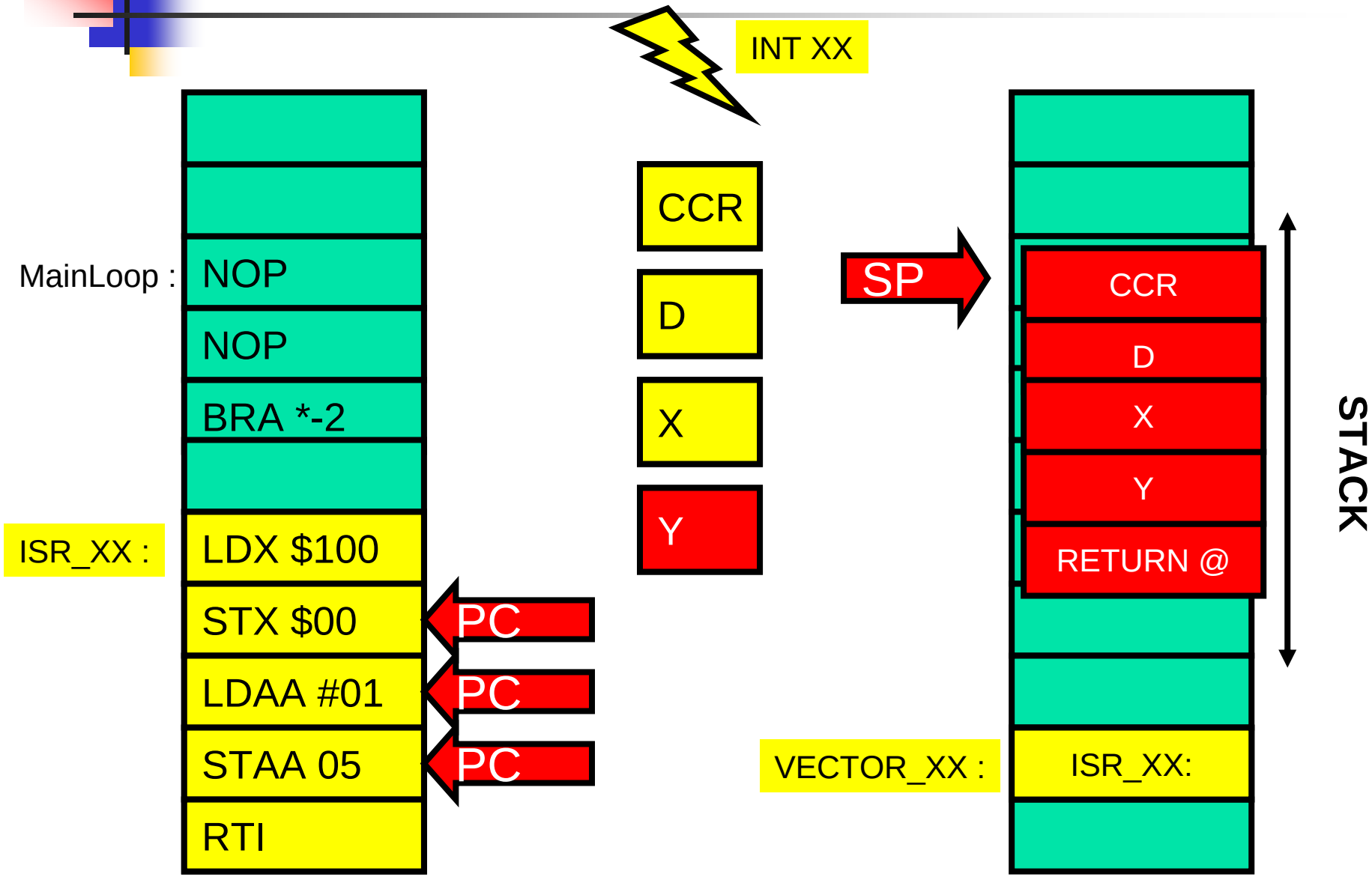
Branchement sur la routine de service



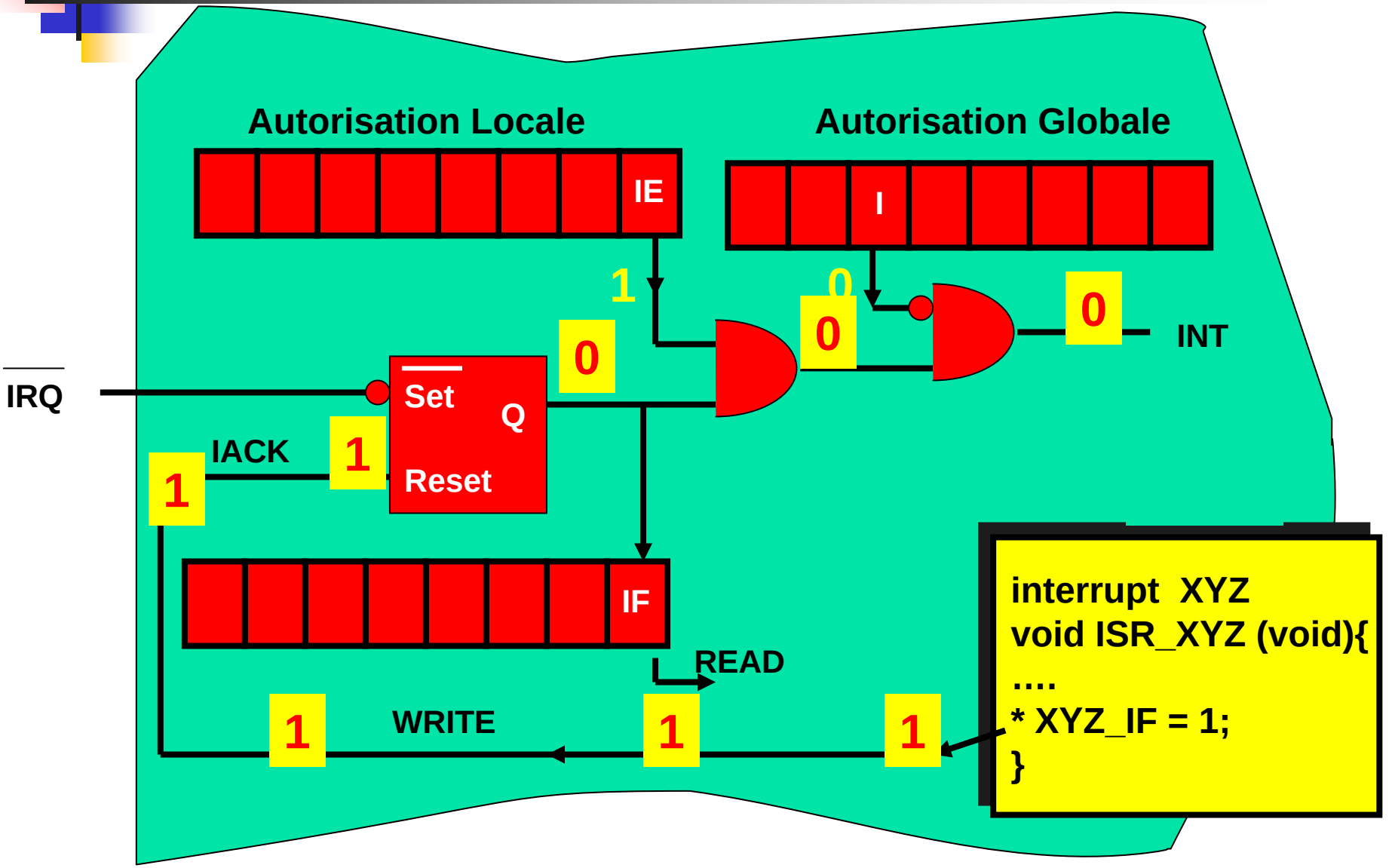
Exécution de l'ISR



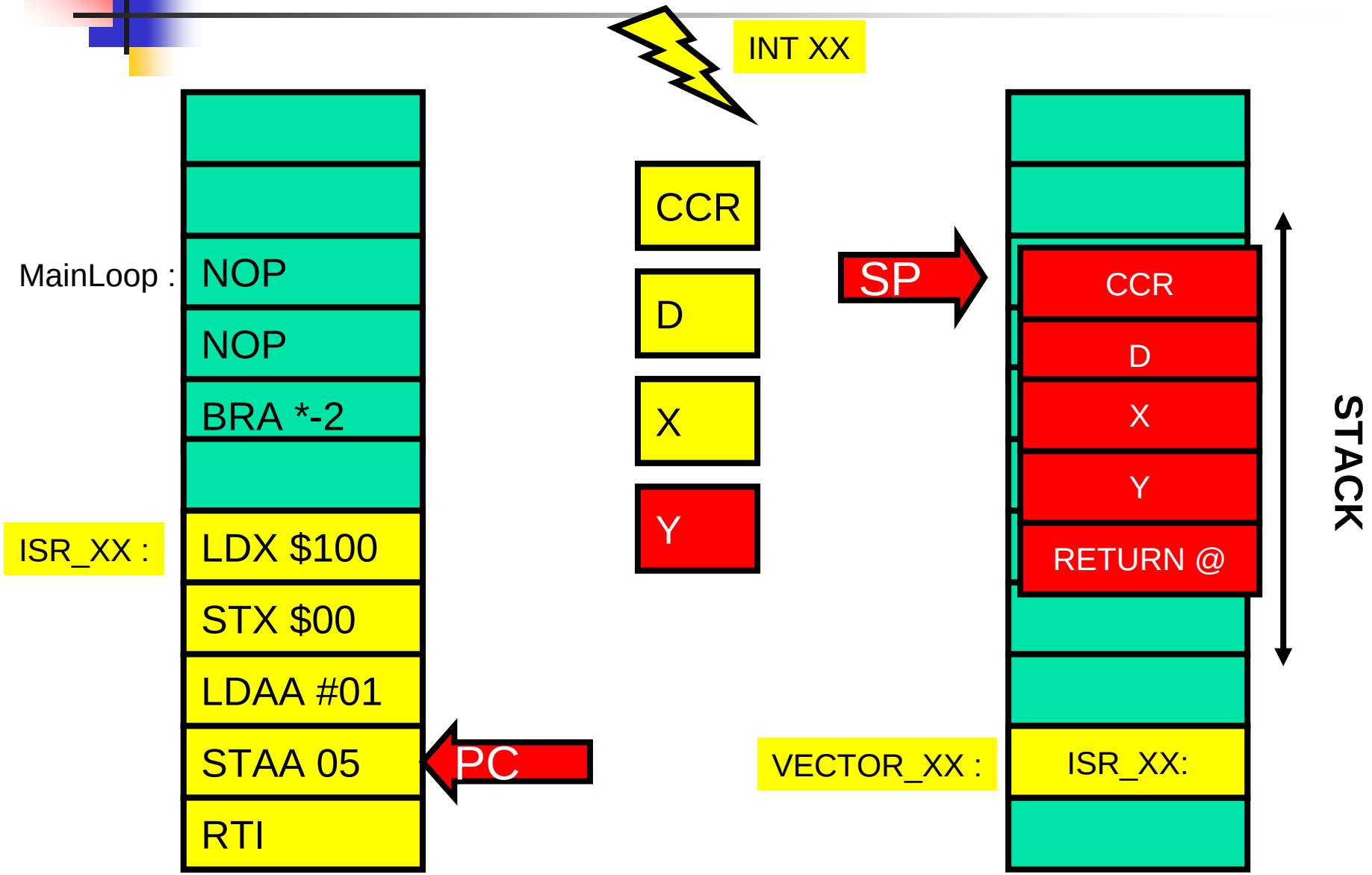
Exécution de l'ISR



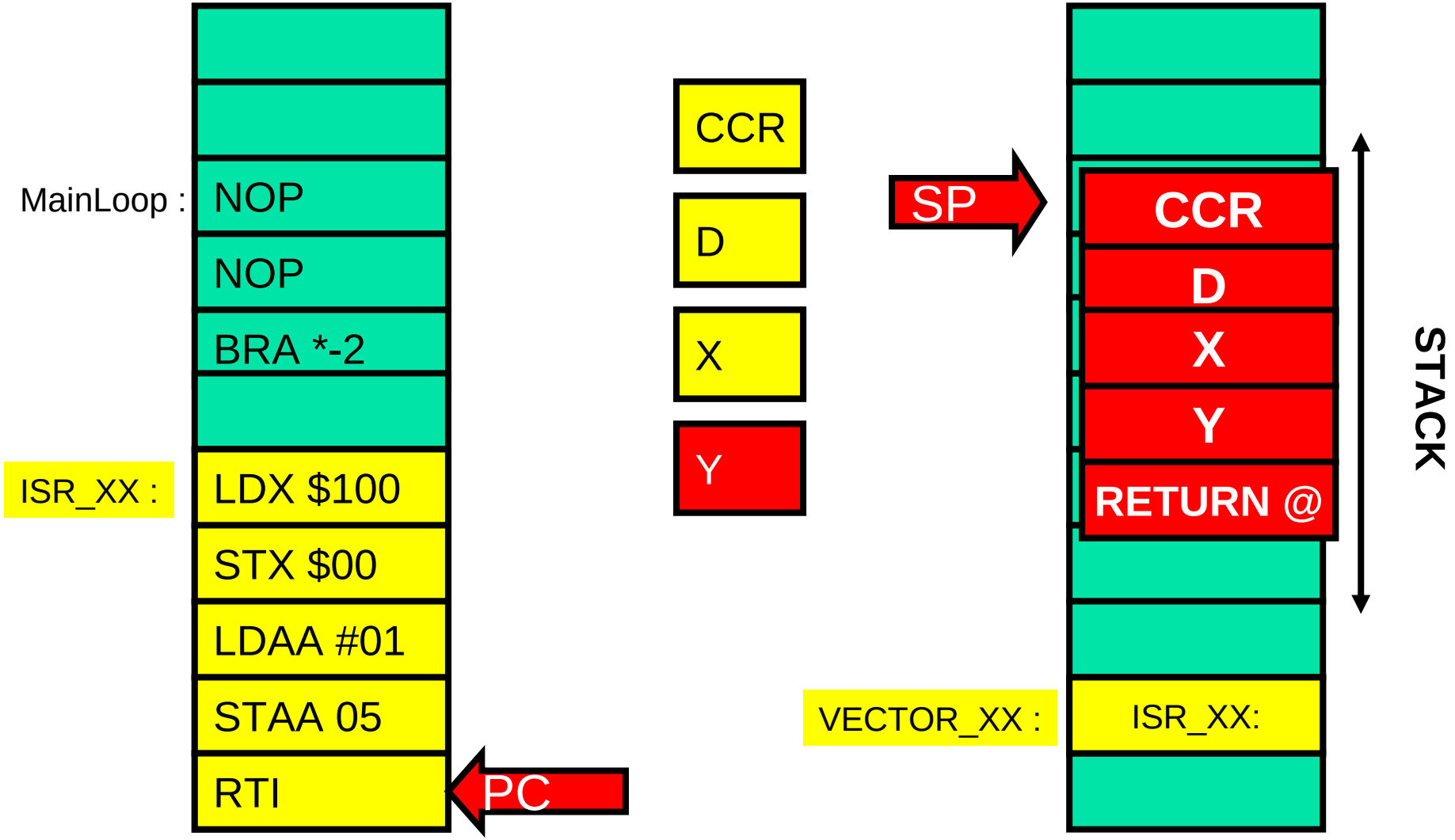
Accuser réception de l'interruption (IACK)



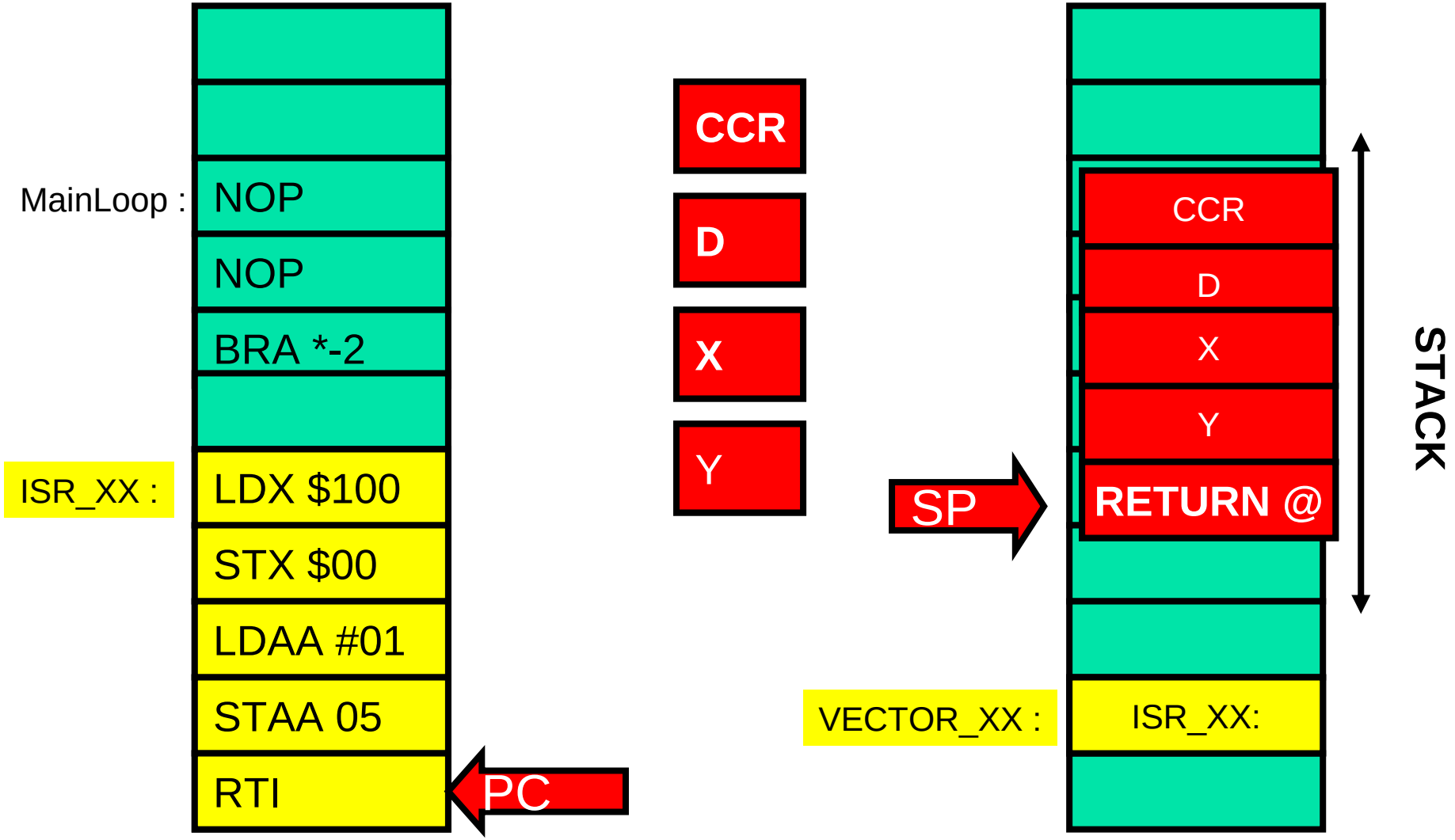
Restaurer le contexte



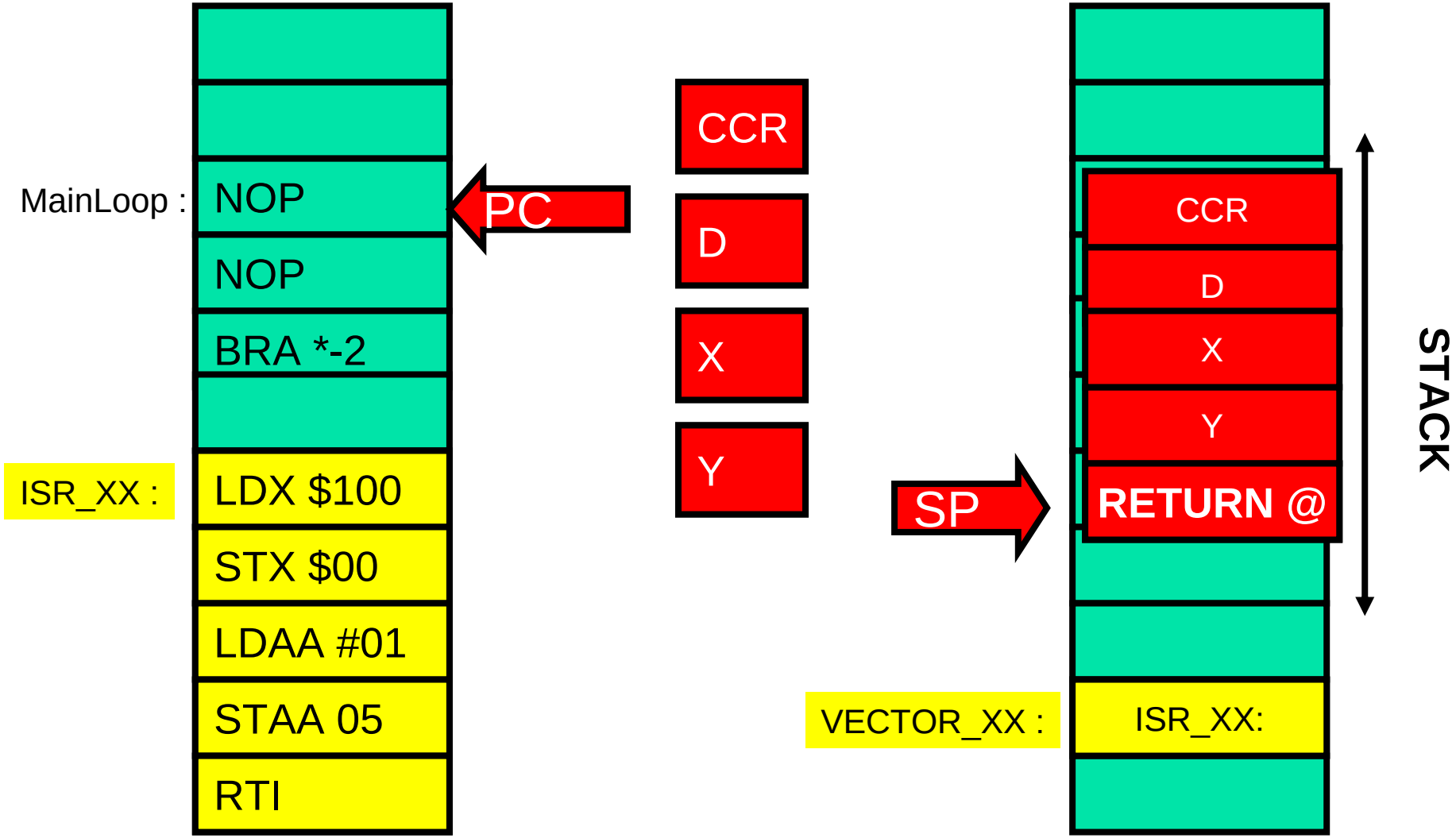
Restaurer le contexte



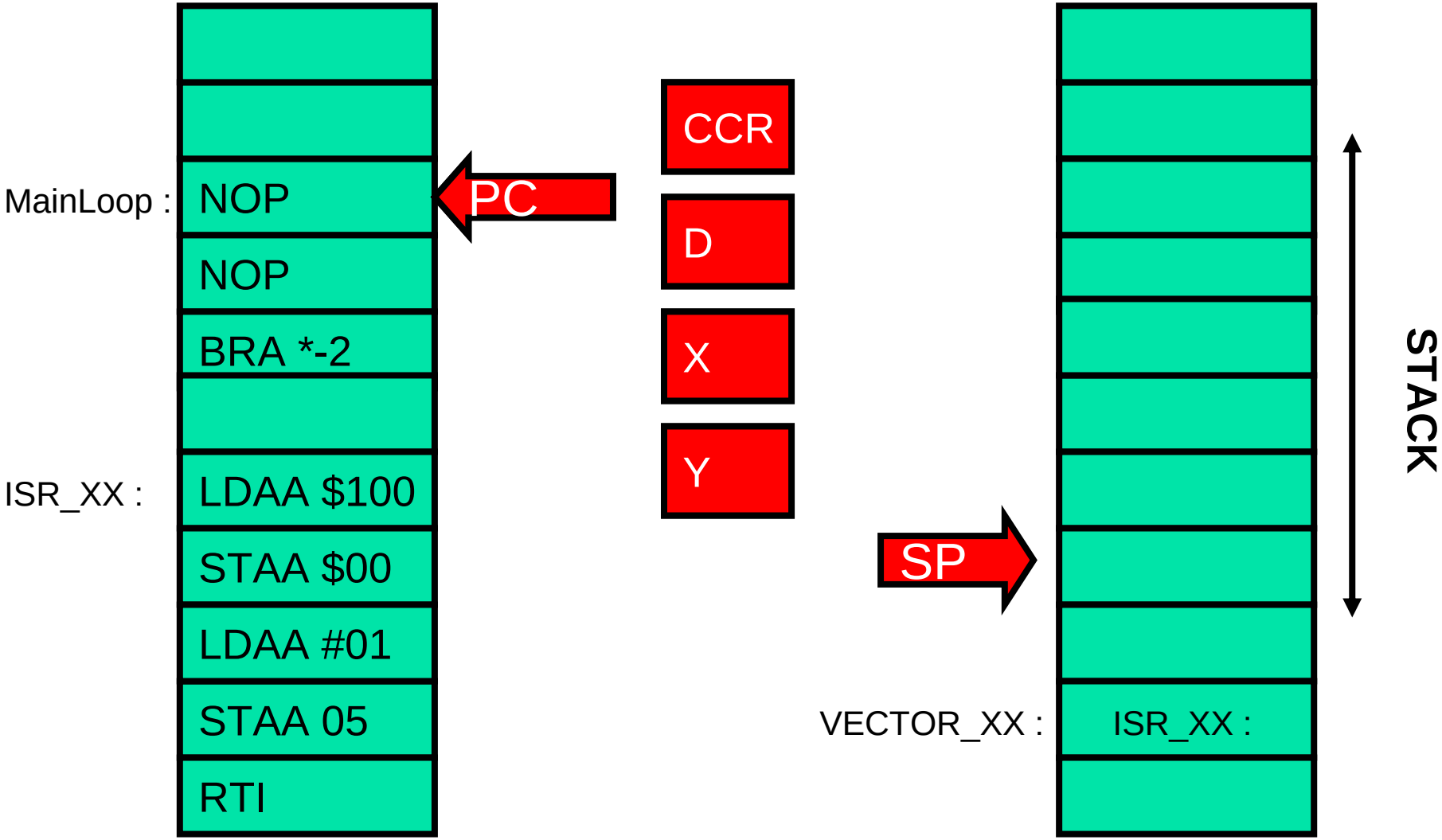
Restaurer le contexte



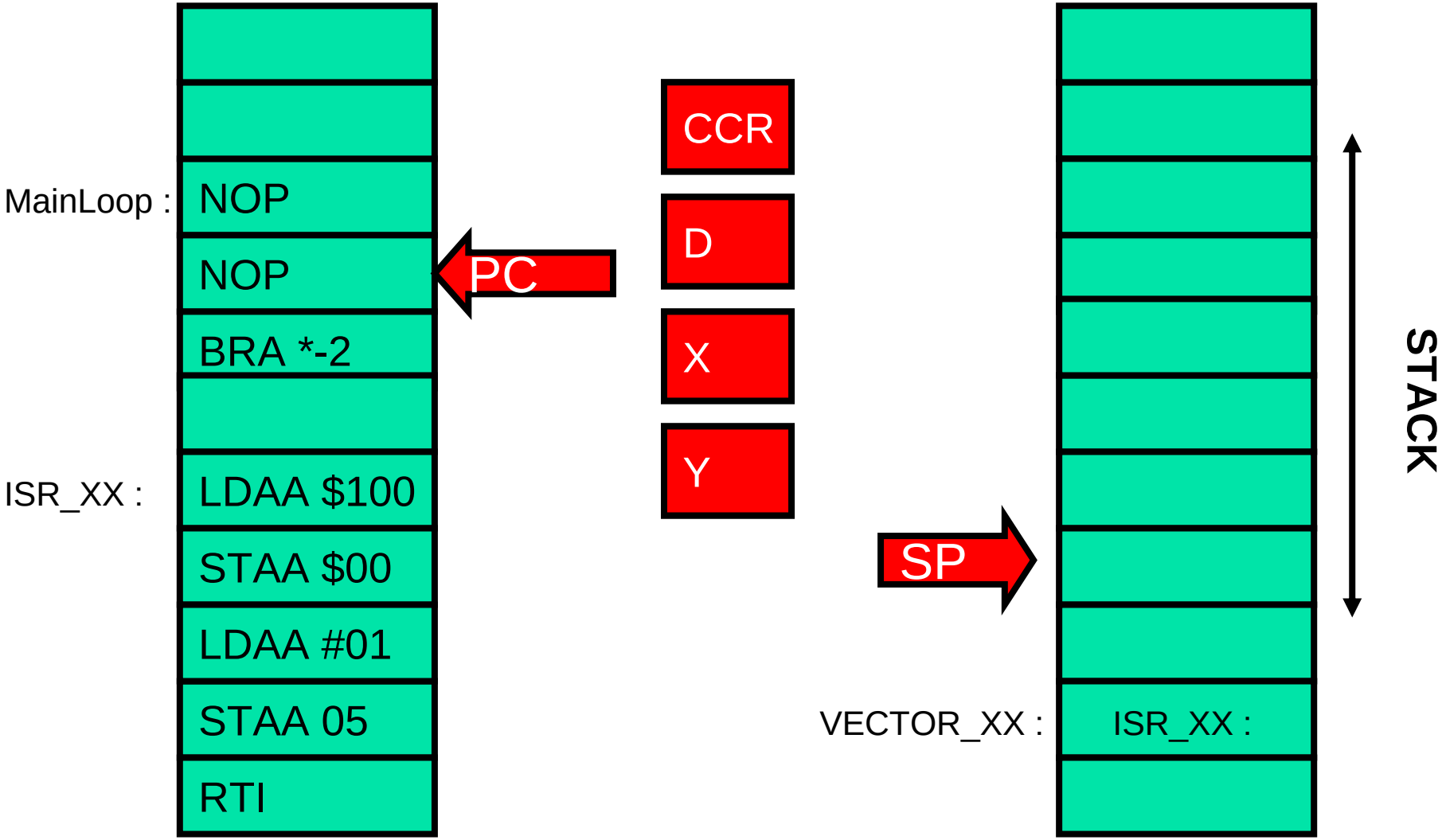
Restaurer le contexte



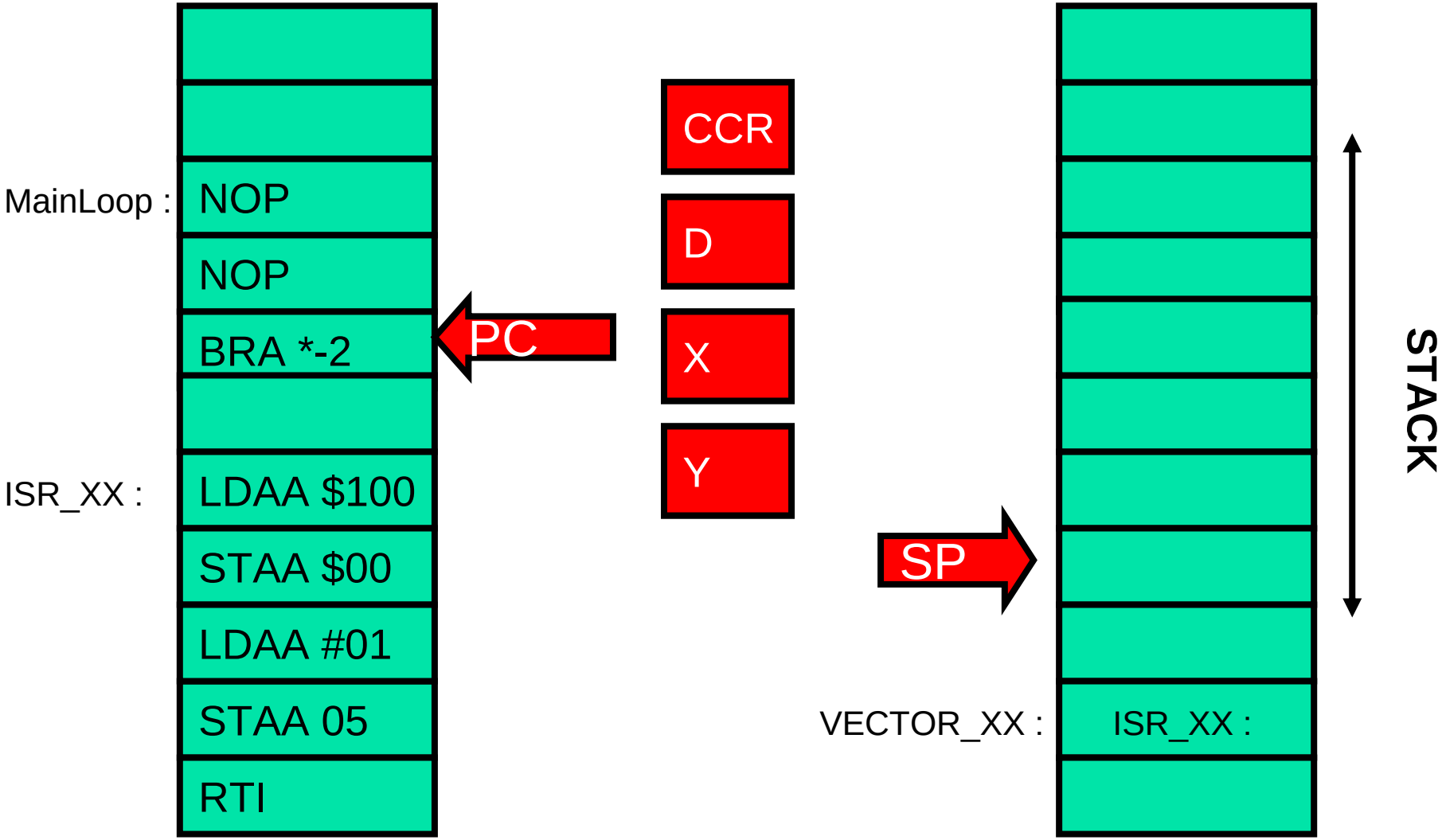
Reprendre le programme d'arrière plan



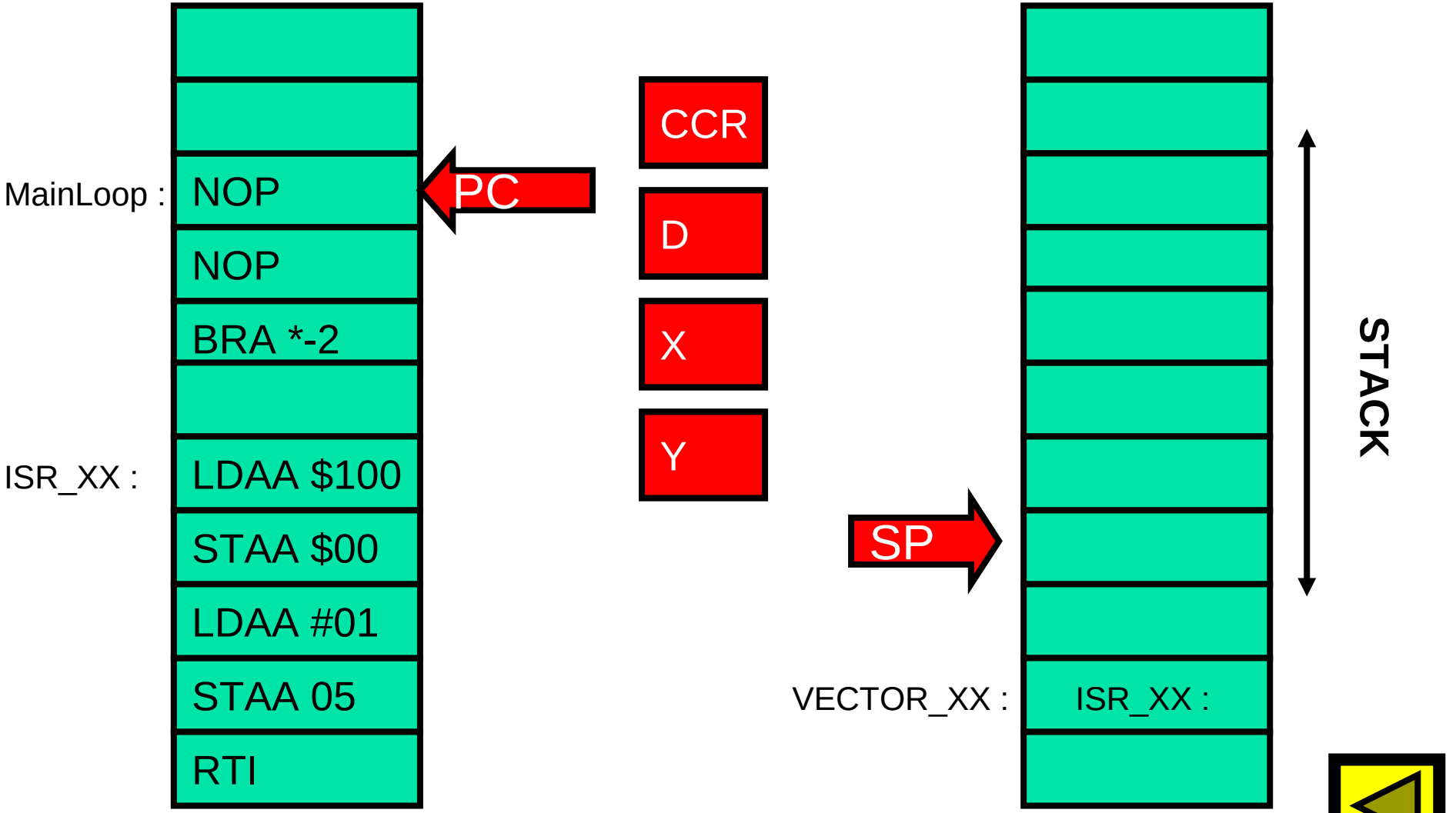
Reprendre le programme d'arrière plan



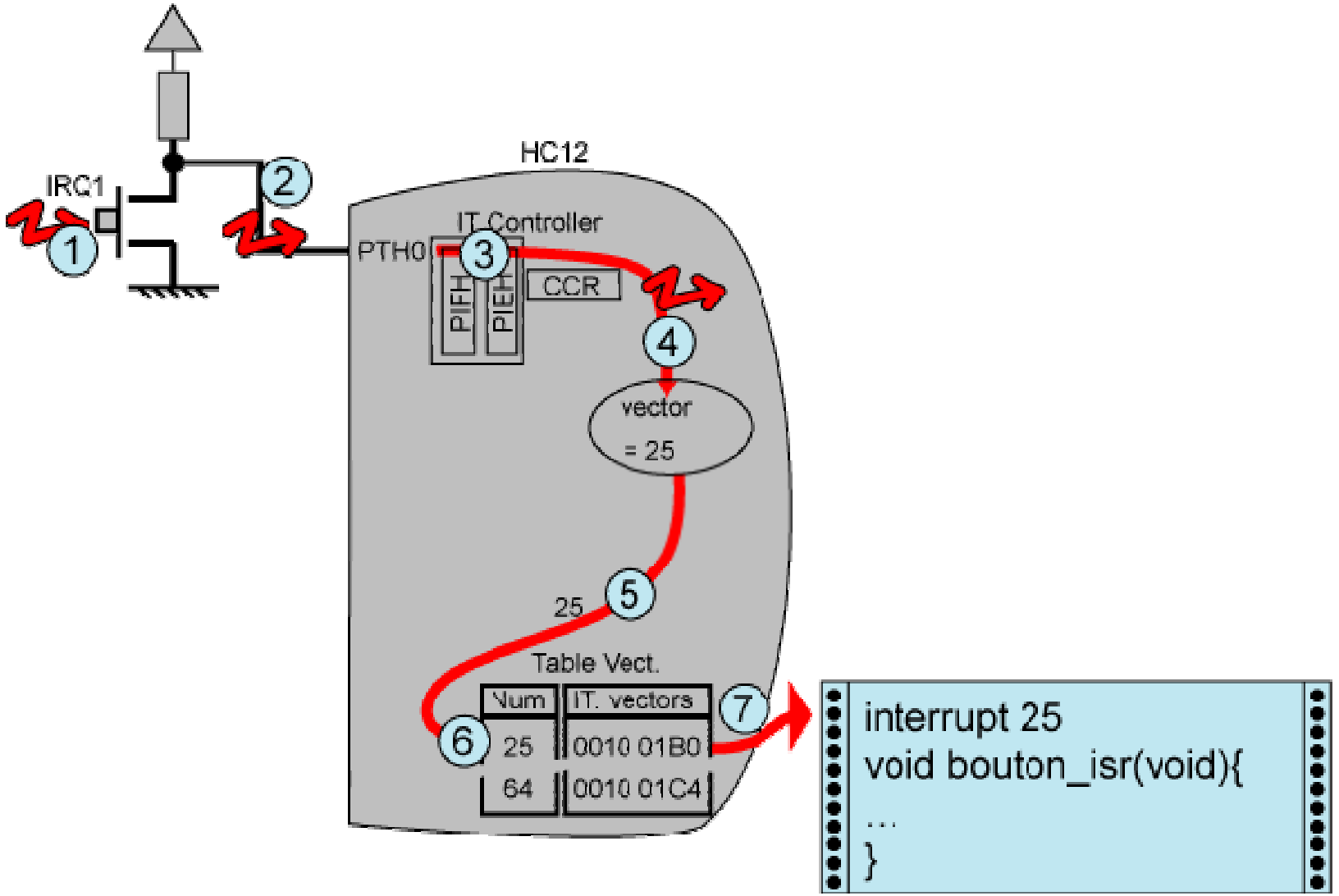
Reprendre le programme d'arrière plan



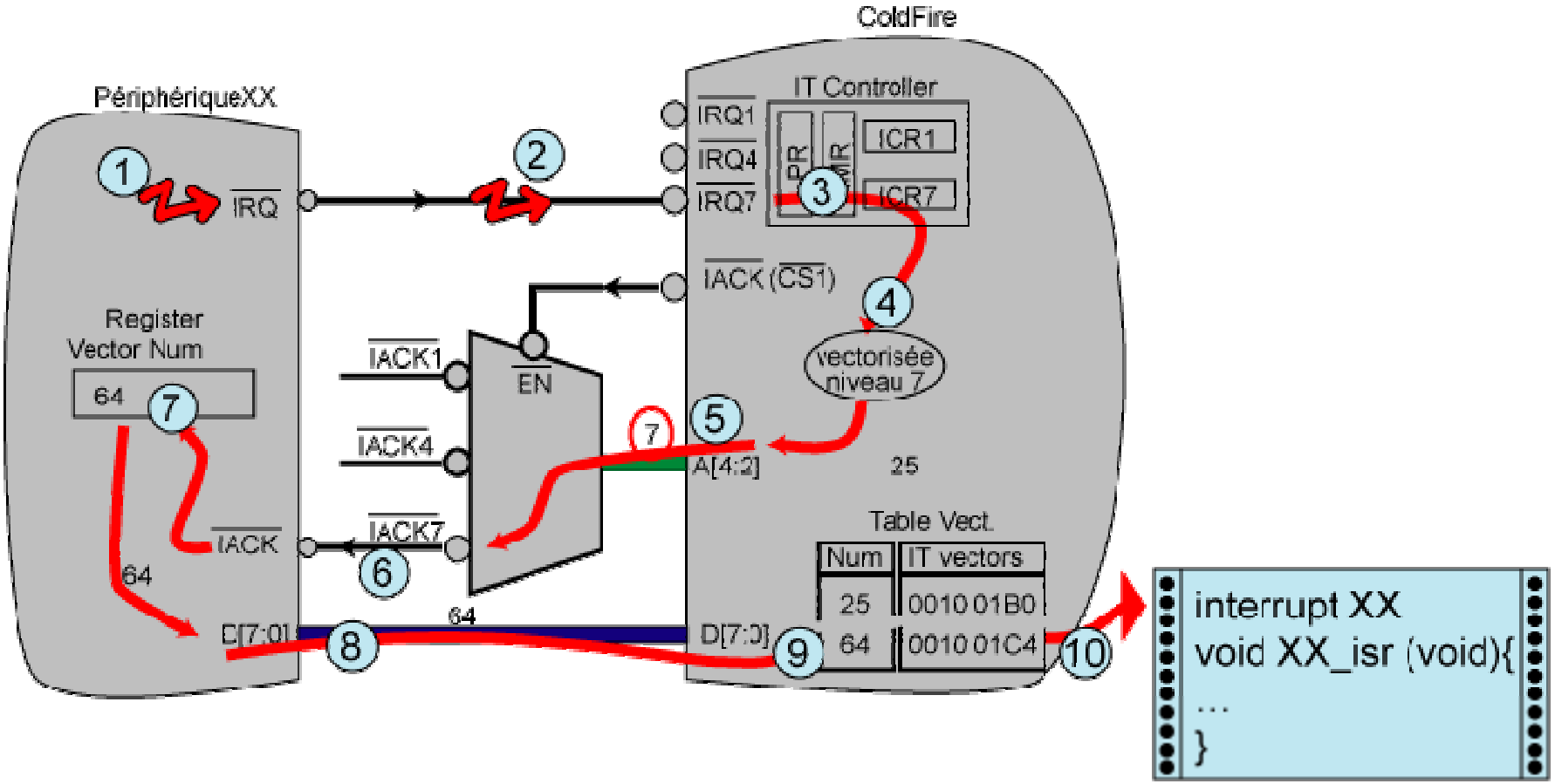
Reprendre le programme d'arrière plan



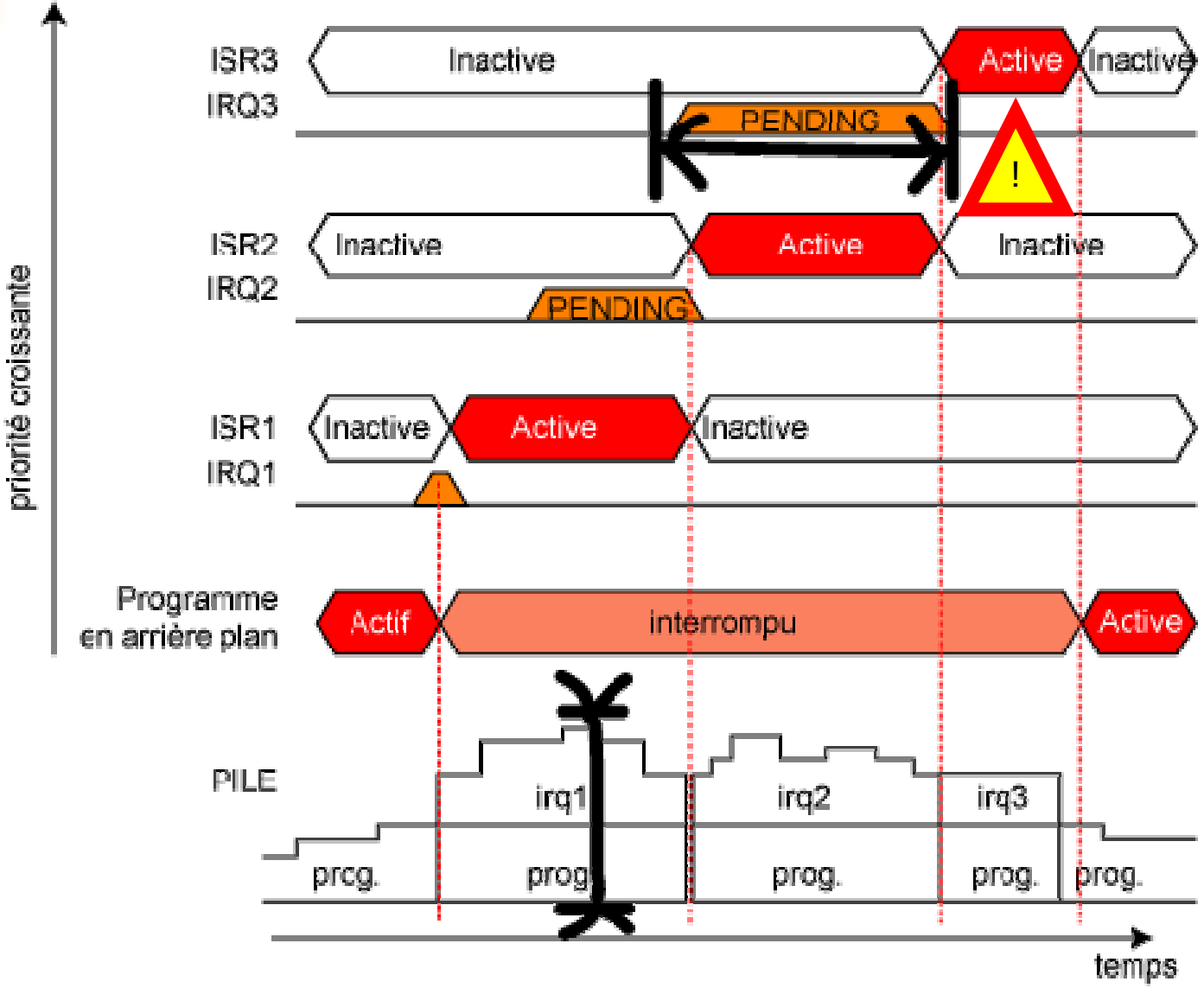
Interruption auto-vectorisée



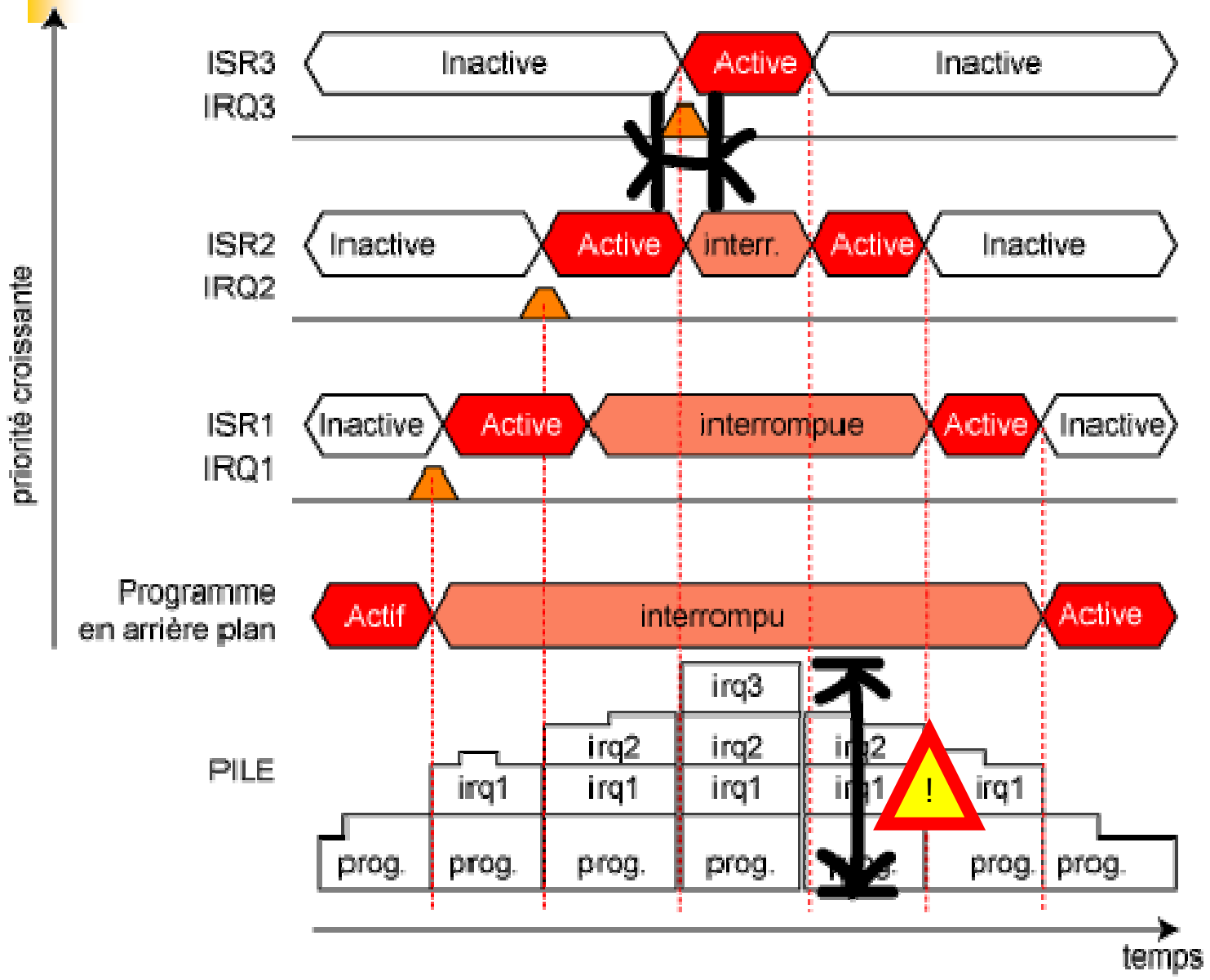
Interruptions vectorisées

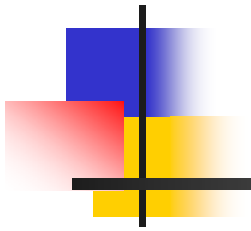


Interruptions non imbriquées



Interruptions imbriquées



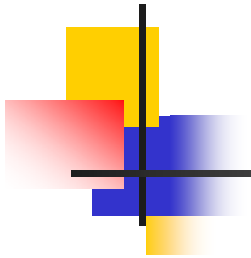


Différents types d'interruptions

Interruptions internes au microprocesseur

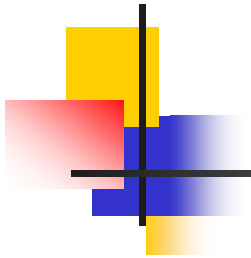
Interruptions matérielles

Interruptions logicielles



Interruptions internes au µp

- Elles traduisent une anomalie d'exécution ou un cas très particulier du fonctionnement d'un logiciel.
  - Division par zéro
  - Mode pas à pas
  - Débordement lors d'un calcul
  - <point d'arrêt> atteint
  - Instruction inconnue

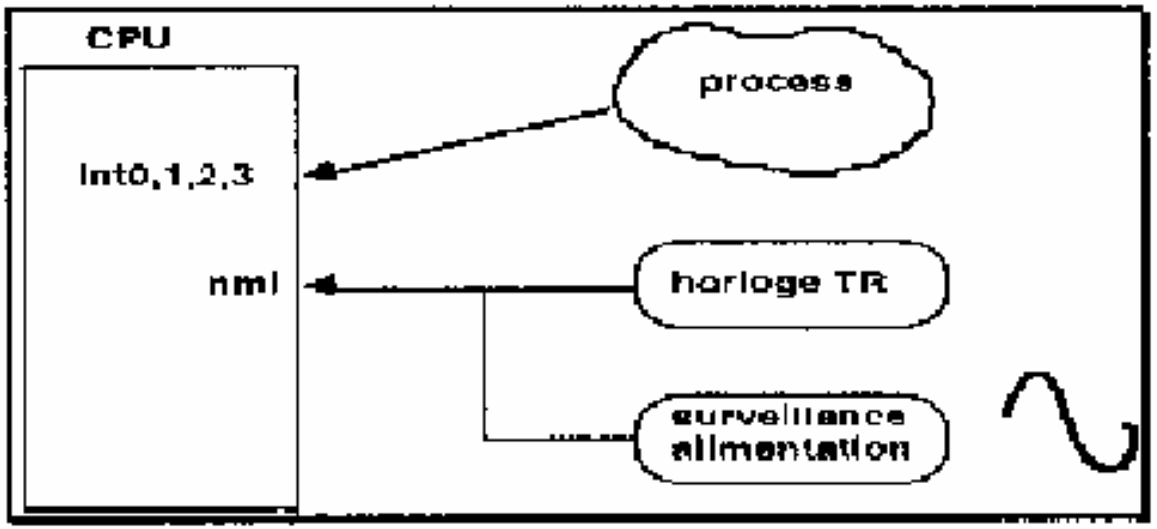


Interruptions matérielles

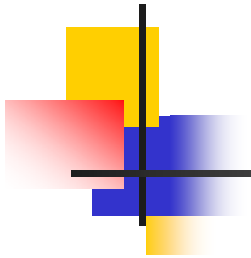
- Une condition matérielle particulière doit être prise en compte ; par exemple, un périphérique cherche à « attirer l'attention » : Exemple de périphériques :
  - Port série com1 (UART), ou com2
  - Timer, ou horloge temps réel,
  - Convertisseur analogique numérique / numérique analogique,
  - Port parallèle,
  - Disque dur,
  - Clavier, ...

- On retrouve sur tous les µp deux types d'it matérielles :
  - Interruption masquable : peut être masquée par programmation
  - Interruption non masquable (NMI) : ne peut pas être masquée.

Interruption non-masquable – NMI – XIRQ



- Les événements critiques pour le contrôle de processus sont pris en compte par une interruption non-masquable
- Les événements processus sont pris en compte par les interruptions masquables



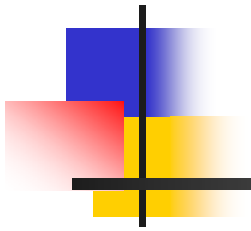
Interruptions logicielles

- Le programme exécuté provoque lui-même l'interruption
  - Ce qui revient à appeler un sous-programme

- Avantages
  - Permet l'exécution d'un code inaccessible autrement (appels aux fonctions d'affichage de la carte graphique, ...)
  - Permet de faire des appels systèmes (appels aux fonctions du BIOS, ou appels aux fonctions du système d'exploitation, ...)

Interruptions logicielles

- En assembleur 6809 de motorola
  - swi1
  - swi2
  - swi3
- En assembleur intel
  - int 21h
- Appel en C (attention fonctions obsolètes)
  - Int86
  - Int86x
  - intdos
- Sur un PC, on trouve les interruptions logicielles lors d'appels au:
  - BIOS (programmes écrits par le constructeur pour assurer la compatibilité des PC entre eux),
  - Système d'exploitation,
  - Matériels (exemple : appel aux fonctions d'affichage de la carte graphique, programmes fournis par le constructeur de la carte).



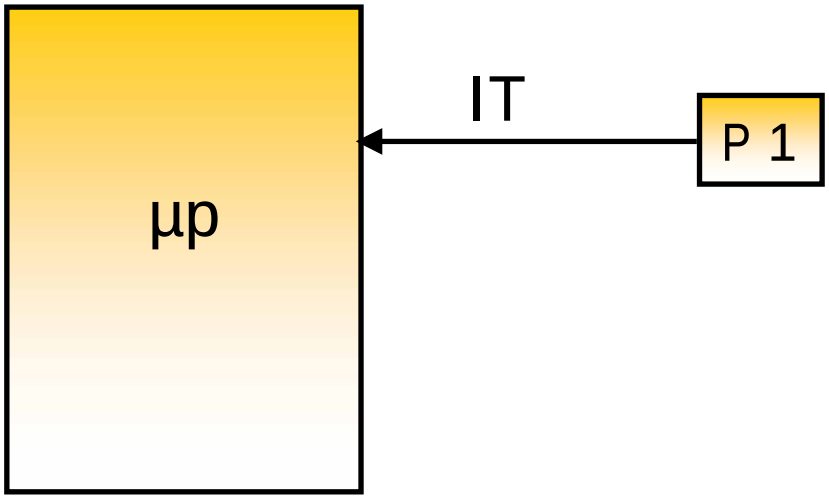
Connexion des périphériques aux lignes d'IT

Connexion directe simple

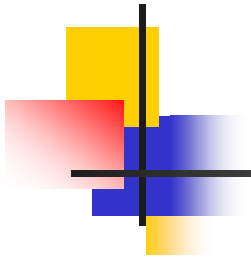
Connexions multiples

Exemple du 6809 – 68000 – 80x86

Connexion directe



- Chaque périphérique est relié à 1 ligne d'interruption du µp ou du µc



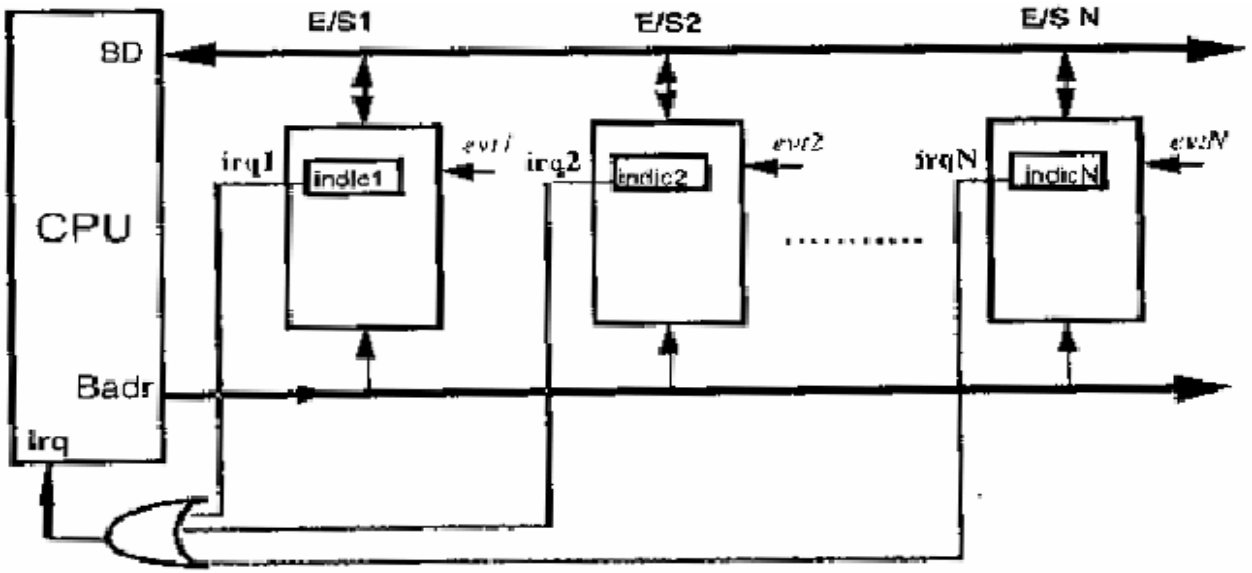
Connexions multiples

- Plusieurs sources d'interruptions peuvent partagées la même ligne électrique. Le CPU devra déterminer laquelle de ces sources a produit l'IRQ.

- Chaque périphérique possède un indicateur d'IT permettant de signaler qu'il a demandé une IT

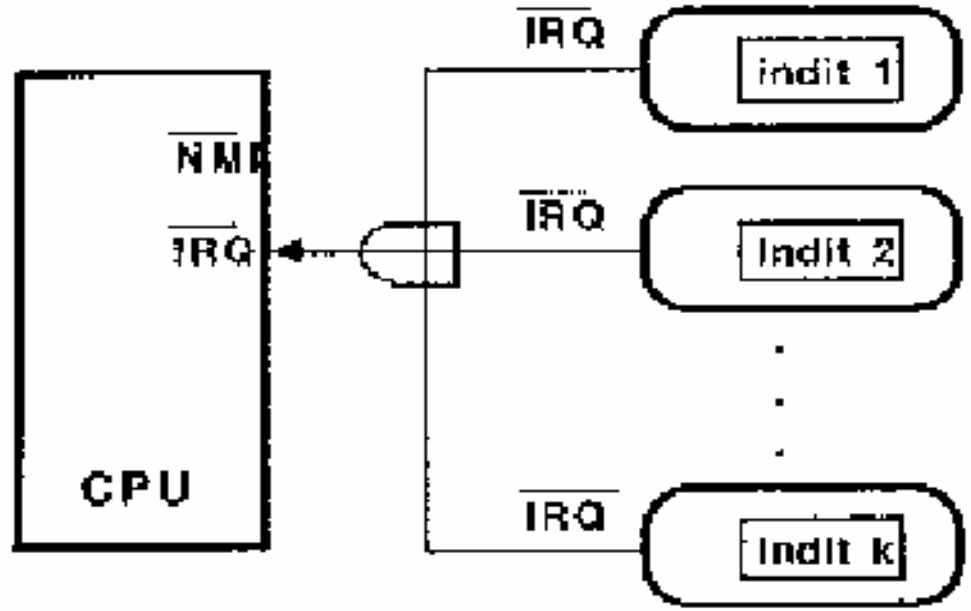
- Un périphérique peut généré une interruption pour plusieurs événements:
  - Exemple de l'UART : Arrivée d'un caractère, Transmission d'un caractère, Erreur de parité, Erreur de détection de porteuse, ...

Connexions multiples



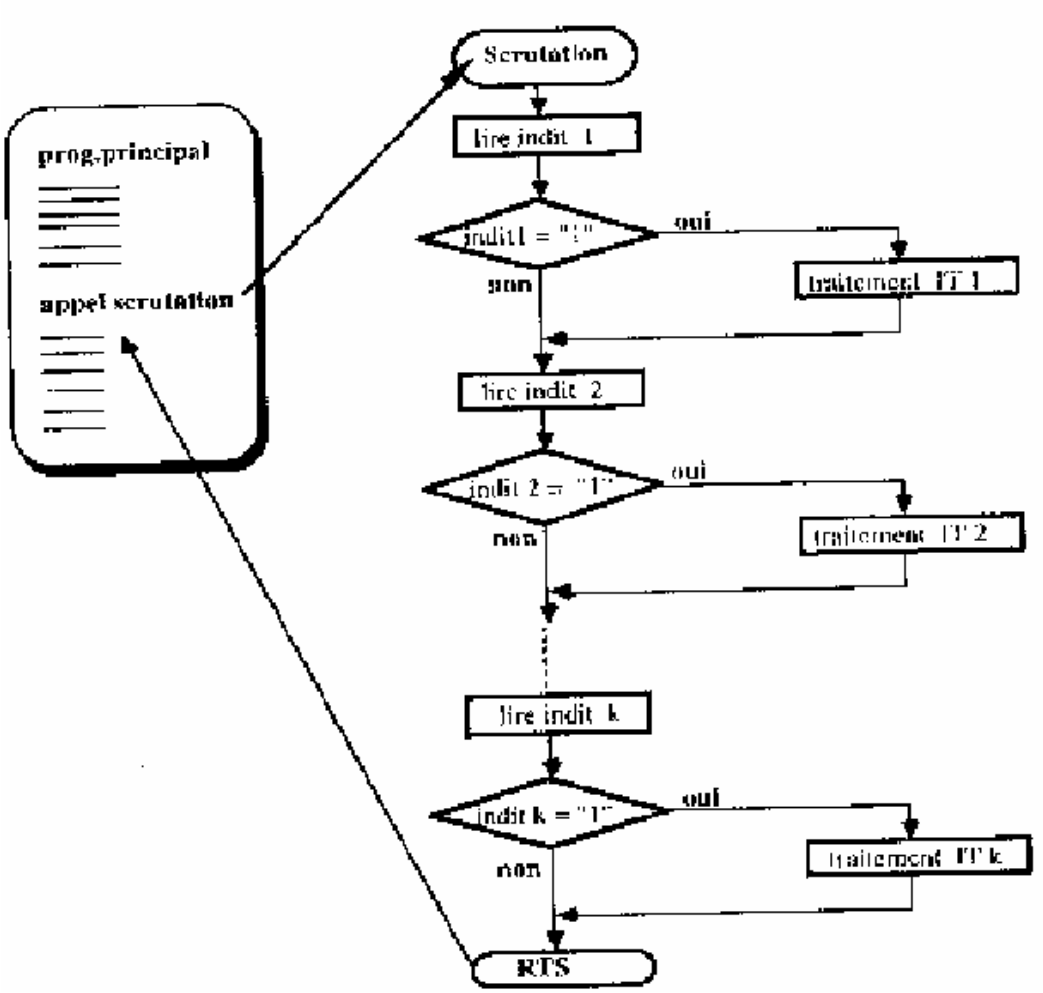
- Dans cette exemple une IT est générée sur un état haut de la ligne IRQ ➔ Toutes les lignes d'IT sont connectées en utilisant une porte logique OU.

Connexions multiples

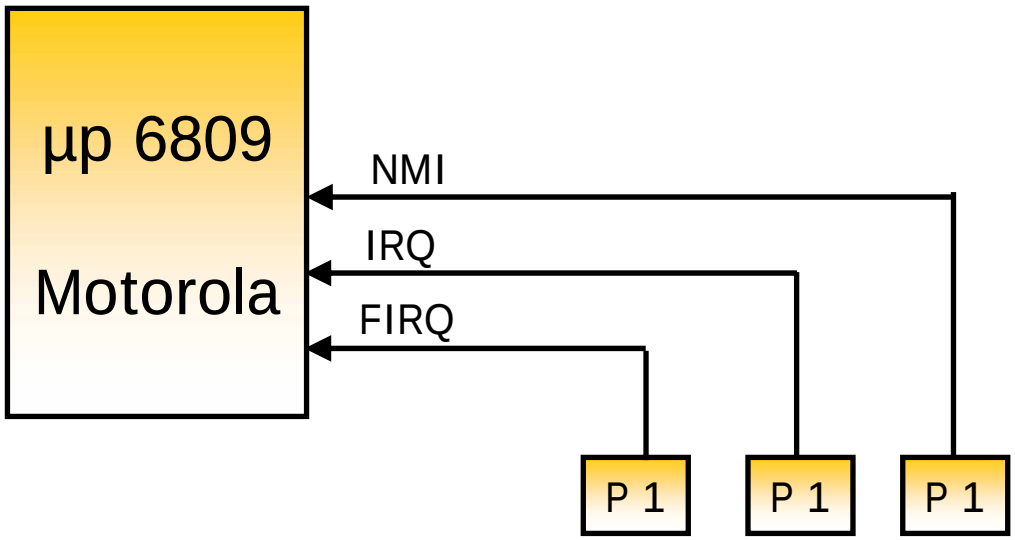


- Dans cette exemple une IT est générée sur un état haut de la ligne IRQ
  ➔ Toutes les lignes d'IT sont connectées en utilisant une porte logique ET.

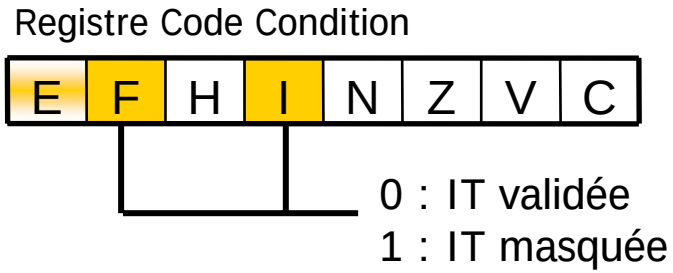
Scrutation logiciel des périphériques (polling) afin de servir celui qui a généré une IT



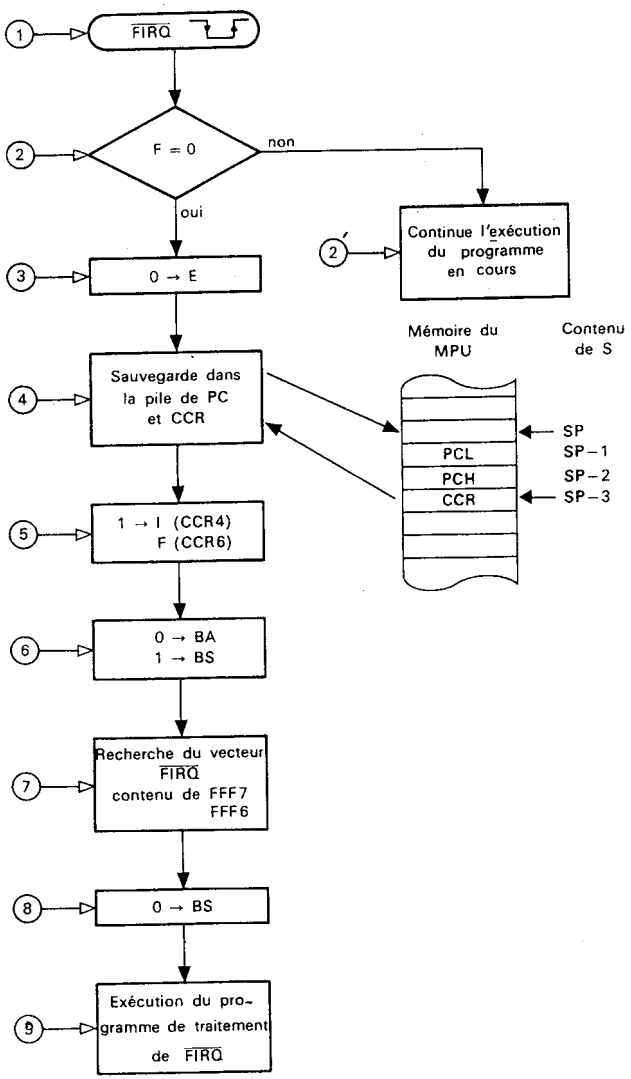
Connexion des périphériques aux lignes d'IT
1er exemple : le µp 6809



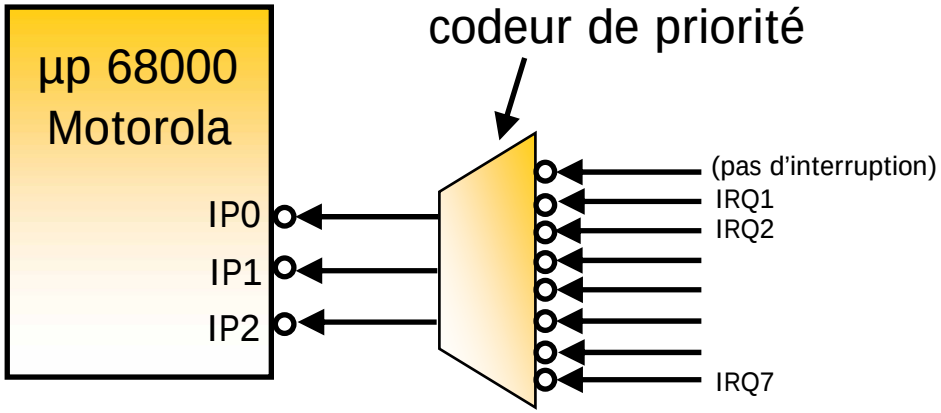
- Masquage ou validation des interruptions



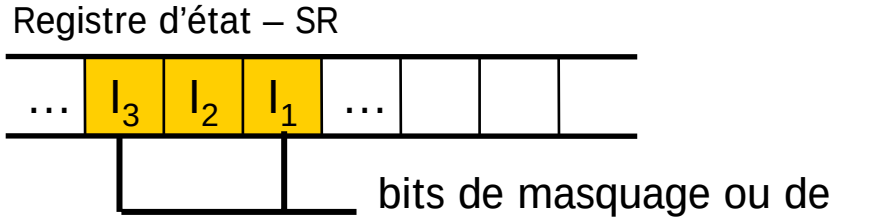
Que se passe-t-il lorsqu'une interruption est activée ? Exemple du 6809 de Motorola

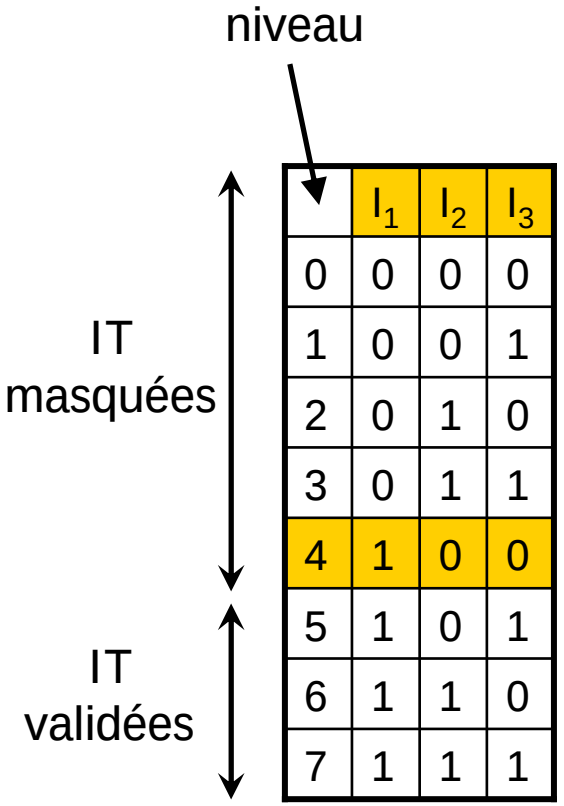


Connexion des périphériques aux lignes d'IT
2ème exemple : le µp 68000



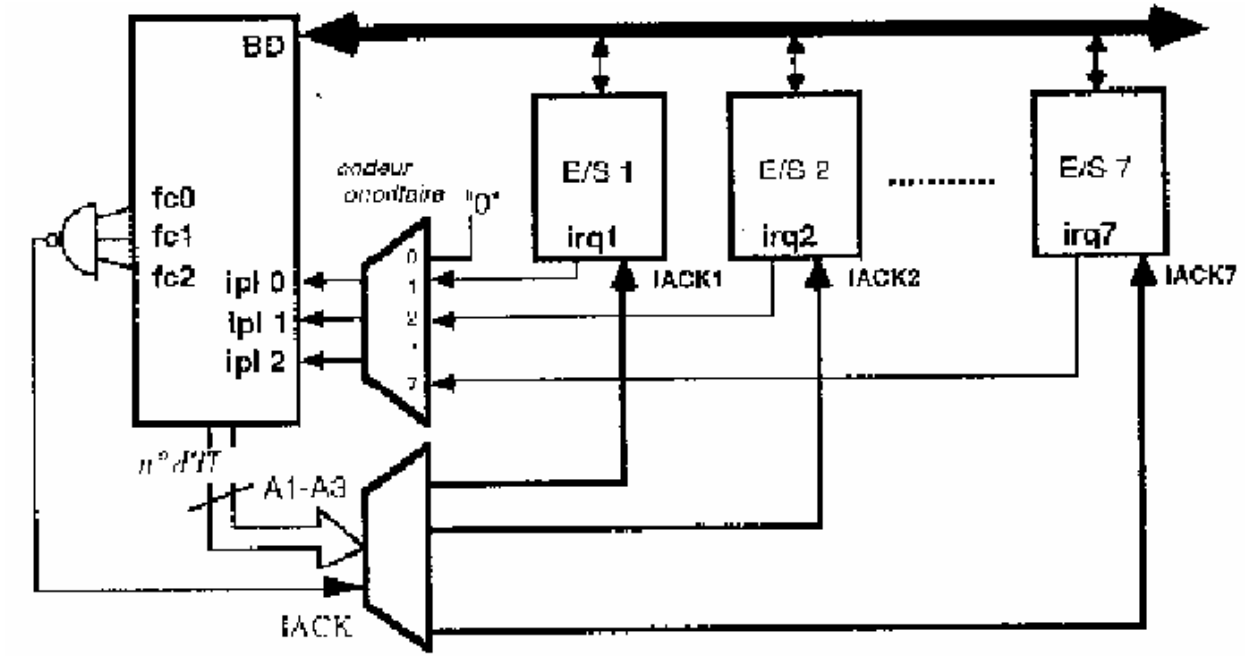
- Masquage ou validation des interruptions





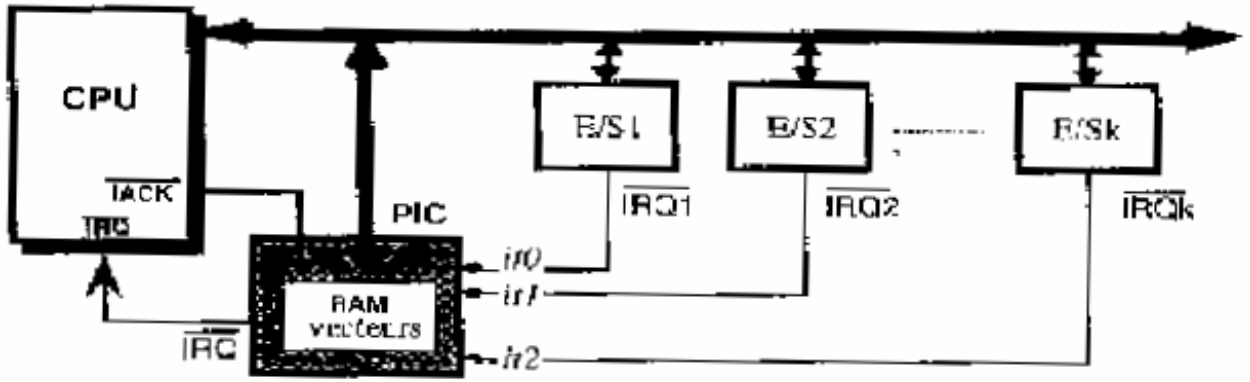
	I_1	I_2	I_3
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Acquittement d'interruption : exemple du 68000



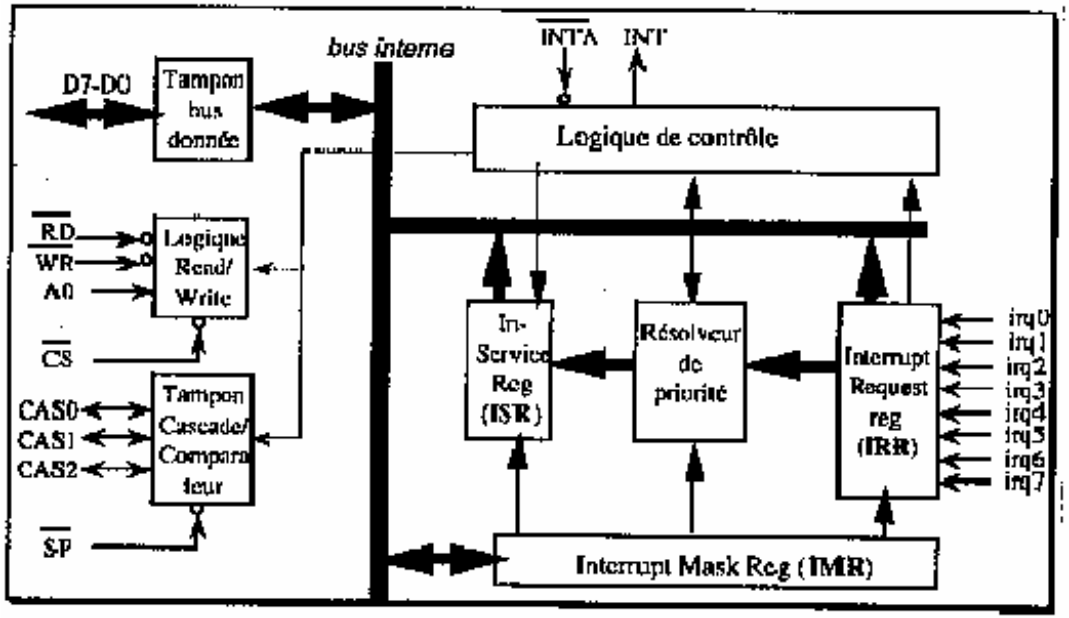
- Le numéro de l'IT acquittée est renvoyé sur les 3 premières lignes d'@ du CPU.
- Un décodeur permet d'appliquer la ligne d'accusé de réception sur le boîtier source de l'IT

Connexion des périphériques aux lignes d'IT
3ème exemple : le 80x86 (PC)



- L'encodeur de priorités : PIC 8259

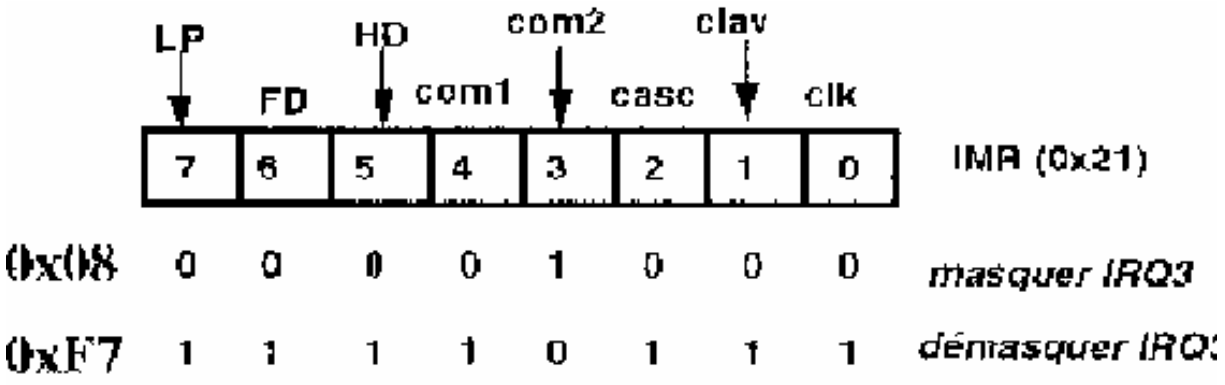
Connexion des périphériques aux lignes d'IT
3ème exemple : le 80x86 (PC)



IRR (Interrupt Request Register):	Pour enregister les requêtes d'interruptions asynchrones.
ISR (In Service Register):	Pour stocker le N° des interruptions en cours de traitement.
IMR (Interrupt Mask Register):	Pour définir les interruptions qui seront masquées et celles qui seront actives.

- Schéma bloc d'un PIC

Masquage des IT sous PC



- **Masquage / Démasquage de l'IRQ 3**
  - **Void masqIrq3(void)**

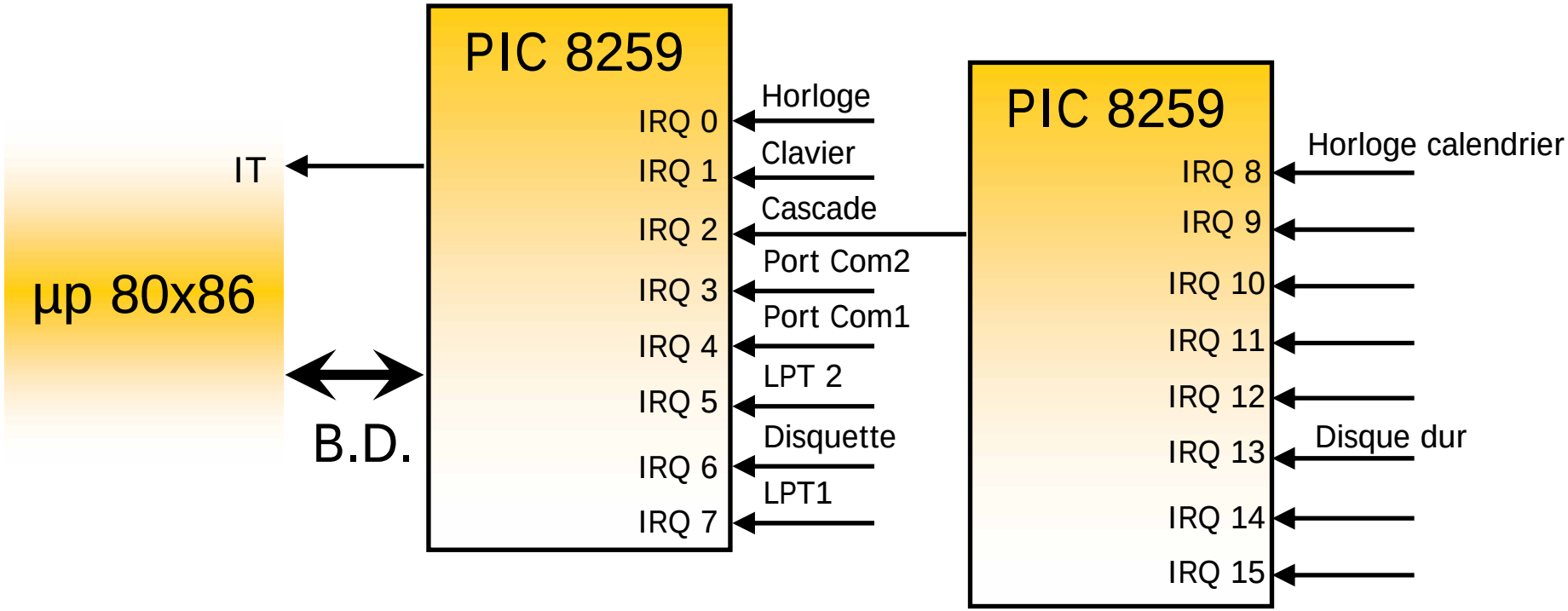
```
{unsigned char marque;
 masque = inportb(IMR);
 outportb(IMR, masque | 0x08);
}
```

  - **Void demasqIrq3(void)**

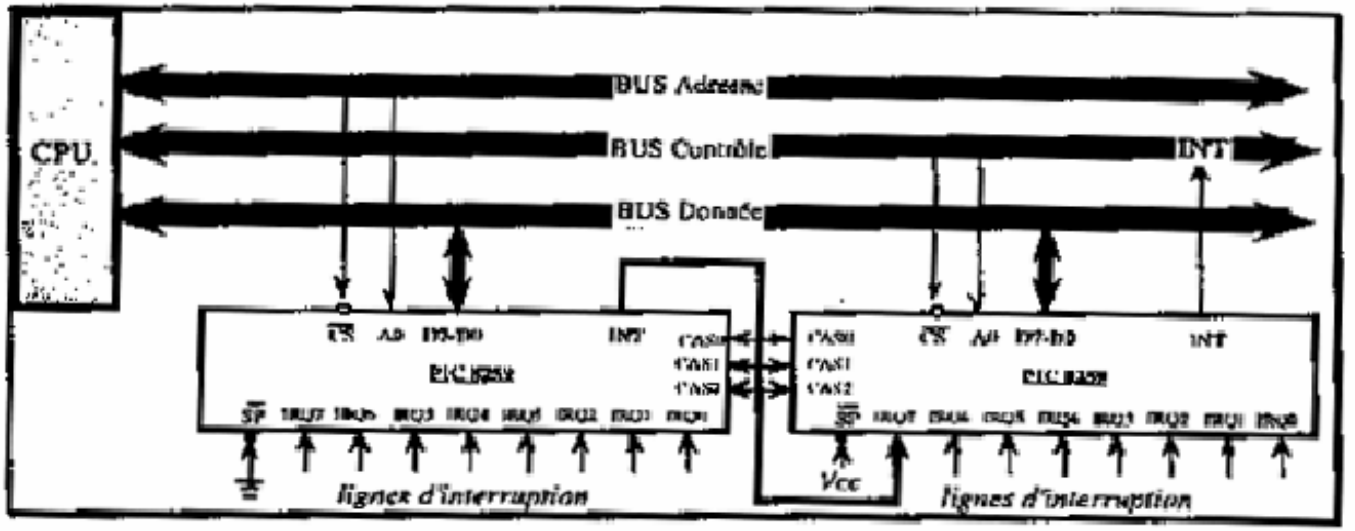
```
{unsigned char marque;
 masque = inportb(IMR);
 outportb(IMR, masque & 0xF7);
}
```

Connexion des périphériques aux lignes d'IT
3ème exemple : le 80x86 (PC)

- Le PC AT contient deux contrôleurs de priorité de type Intel 8259 permettant de gérer les requêtes d'IT
  - Mettre en cascade 2 boîtiers permet d'augmenter le nombre de périphériques Temps réel



Connexion des périphériques aux lignes d'IT
3ème exemple : le 80x86 (PC)

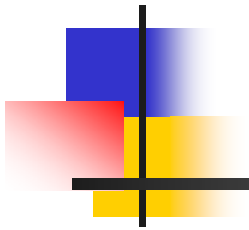


- Cascade des PIC

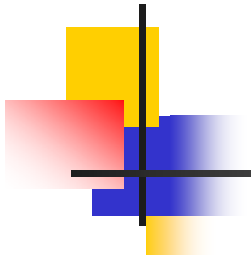
Priorités des interruptions matérielles sur PC

- NMI
- IRQ0 Horloge
- IRQ1 Clavier
- IRQ2 8259 N°2
- IRQ3 COM2
- IRQ4 COM1
- IRQ5 LPT2
- IRQ6 Floppy

Priorité décroissante



Vectorisation des interruptions



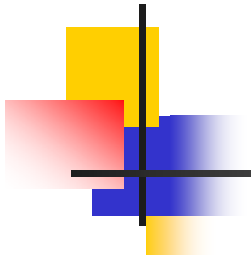
Vectorisation des IT

- Lorsque l'IRQ *n* est activée,

quel programme est exécuté ?

Comment le µp sait qu'il doit exécuter tel programme plutôt que tel autre ?

- La table des vecteurs d'interruption contient les adresses des programmes correspondant à chaque IT



Vectorisation d'IT

- Un microprocesseur peut distinguer un nombre donné d'exceptions de natures différentes. Il fait correspondre à chacune d'elles un traitement particulier en exploitant une table d'adresses de lancement de ces traitements.

- La localisation de cette table est fixe, implicitement connue, mais son contenu est programmable et va être adapté à l'application développée.

- Ce procédé est nommé : vectorisation des exceptions

Mécanisme

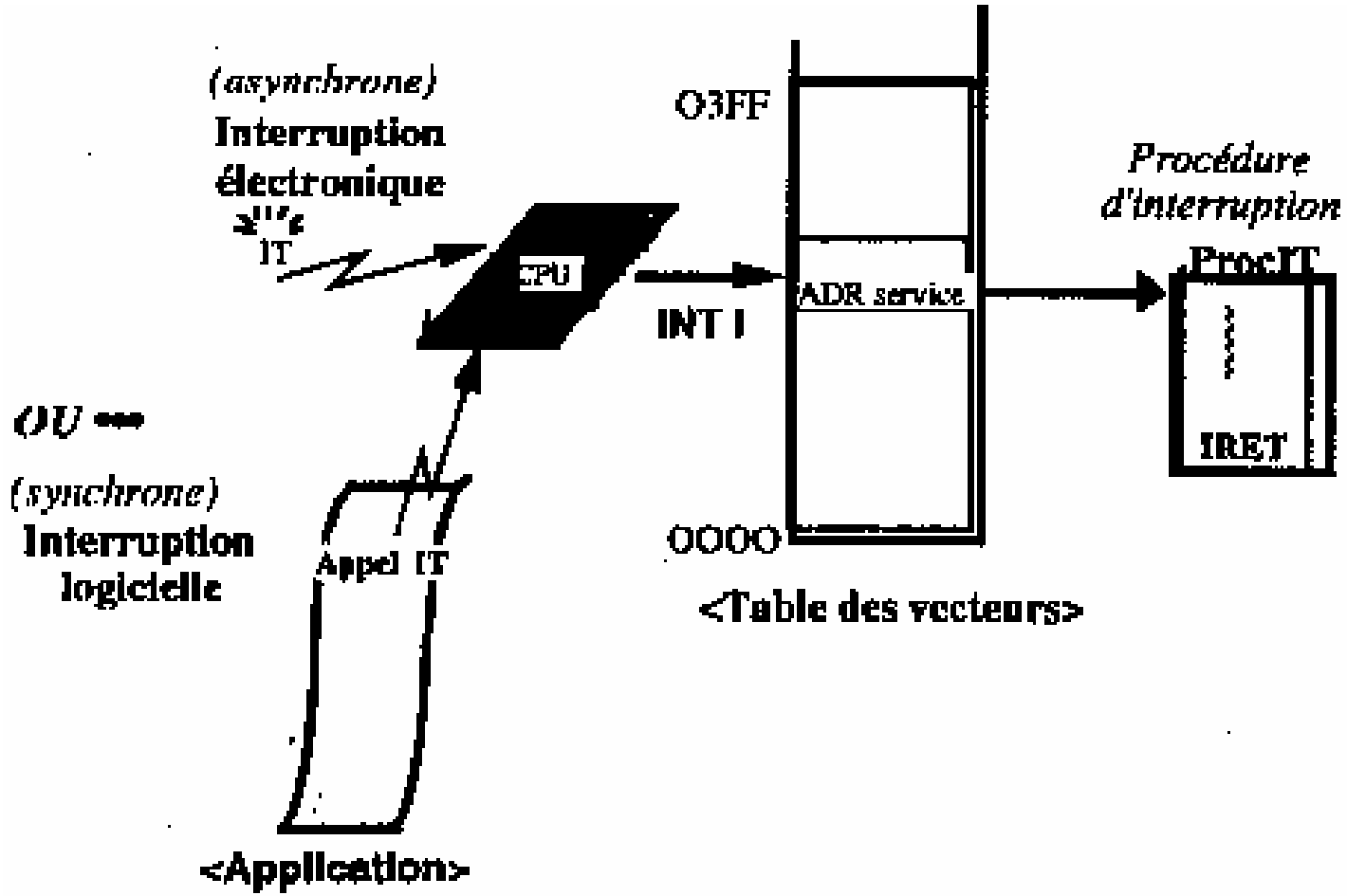

(asynchrone)
Interruption
électronique
IT
CPU
INT I
03FF
ADR service
0000
Procédure
d'interruption
ProcIT
IRET
<Table des vecteurs>
OU
(synchrone)
Interruption
logicielle
Appel IT
<Application>

Table des vecteurs d'IT du µp 6809 de Motorola

Interruption	Adresse	Contenu
Reset	FFFE	$ D000
NMI	FFFC	Xxxx
SWI	FFFA	$ 8800
IRQ	FFF8	$ DA00
FIRQ	FFF6	$ 8000
SWI2	FFF4	Xxxx
SWI3	FFF2	Xxxx
Réservé	FFF0	xxxx

Table des vecteurs d'IT du µp 68000

Numéro	Adresse	Exception
0	000	Reset (SSP) initial
	004	Reset (PC) initial
2	008	Erreur bus
3	00C	Erreur d'adressage
4	010	Instruction illégale
5	014	Division par 0
6	018	Instruction CHK
7	01C	Instruction TRAPV
8	020	Violation de privilège
9	024	Trace
10	028	Emulateur ligne 1010
11	02C	Emulateur ligne 1111
12	030	Réservé
13	034	Réservé
14	038	Réservé
15	03C	Interruption non initialisée
16 - 23	040	Réservé
	05C	-
24	060	Interruption parasite
25	064	Auto-vecteur interruption niveau 1
26	068	Auto-vecteur interruption niveau 2
27	06C	Auto-vecteur interruption niveau 3
28	070	Auto-vecteur interruption niveau 4
29	074	Auto-vecteur interruption niveau 5
30	078	Auto-vecteur interruption niveau 6
31	07C	Auto-vecteur interruption niveau 7
32 - 47	080	Vecteurs instruction TRAP
	0BC	-
48 - 63	0C0	Réservé
	0FF	-
64 - 255	100	Vecteurs interruptions utilisateur
	3FC	-

La table des vecteur d'IT occupe les @ 0 à $3FF

Table des vecteurs d'IT sur PC

- 256 Vecteurs d'Interruption partagés en:
  - Interruptions internes du µp
  - Interruptions matérielles
    - NMI, IRQ
    - IRQ 0 à IRQ 8 (rajouter 8) ➔ IT 8 à IT 16
  - Interruptions logicielles
    - BIOS, DOS, Prog. utilisateur
  - Non utilisé
- Au chargement du PC, la table des vecteurs d'IT est initialisée pour pointer sur les programmes par défaut
  - l'utilisateur peut reconfigurer cette table pour exécuter ses propres programmes

Table des vecteurs d'interruption sur PC

N°	**Adresse**	**Fonction**
00	000 – 003	CPU : Division par zéro
01	004 – 007	CPU : Pas à pas
02	008 – 00B	CPU : NMI (Défaut circuit RAM)
03	00C – 00F	CPU : Break Point atteint
04	010 – 013	CPU : Débordement numérique
05	014 – 017	**Copie d'écran**
06	018 – 01B	Instruct. Inconnue (80286 seul.)
07	01D – 01F	Réservé
08	020 – 023	IRQ0 : Timer (appel 18,2 fois/sec)
09	024 – 027	IRQ1 : Clavier
0A	028 – 02B	IRQ2 : 2ème 8259 (AT uniq.)
0B	02C – 02F	IRQ3 : Port série com2

Table des vecteurs d'interruption sur PC

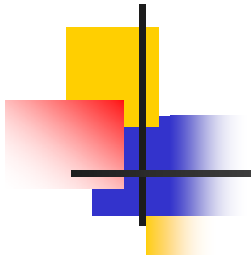
N°	Adresse	Fonction
0C	030 – 033	IRQ4 : Port série com1
0D	034 – 037	IRQ5 : Disque dur
0E	038 – 03B	IRQ6 : Disquette
0F	03C – 03F	IRQ7 : Imprimante
10	040 – 043	BIOS : Fonction vidéo
11	044 – 047	BIOS : Déterminer configuration
12	048 – 04B	BIOS : Déterminer taille RAM
13	020 – 023	BIOS : Fonction disquette/DD
:	: – :	:
1A	068 – 06B	BIOS : Lire date et heure
1B	06C – 06F	Touche Break
1C	**070 – 073**	**Appel après tout int 8**

Table des vecteurs d'interruption sur PC

N°	Adresse	Fonction
20	080 – 083	DOS : Fin programme
21	084 – 087	DOS : Disquette
3F	0FC – 0FF	DOS
47	188 – 18B	Libres : utilisateurs
6F	1BC – 1BF	Libres : utilisateurs
70	1C0 – 1C3	IRQ 8 : Horloge calendrier
71	1C4 – 1C8	IRQ 9
:	: – :	:
75	1D4 – 1D8	IRQ 13 : 80287
76	1D8 – 1DB	IRQ 14 : DD
77	1DC – 1DF	IRQ 15
78 – FF		inutilisées

Structure d'une application C et fonctions d'IT

```
#include ....
#include ....
#define  ....
#define  ....

/* ======== Prototypes des fonctions ========= */

extern char fct1(....); /*Dans le cas d'utilisation de fichiers externes*/
int        fct2(....);

void interrupt fctit1 (......);
void interrupt fctit2 (......);
void instalvct (int var, interrupt (*f)() ); /*Installe un vecteur*/

void interrupt (*OldInter1) (); /*Sauvegarde des vecteurs initiaux*/
void interrupt (*OldInter2) (); /* Idem */

main ()
 {
   ..........
   instalvct(numvct1, fctit1); /*Installe les fonctions d'interruption*/
   instalvct(numvct2, fctit2);
   ...........
 }


/* ======== Le corps des fonctions ========= */
int fct2 (....)
  {
    .....
  }
void interrupt fctit1 (....)
  {
    .....
  }
void interrupt fctit2 (....)
  {
    .....
  }

void instalvct (int var, interrupt (*f)() )
 {
   .......
 }
```



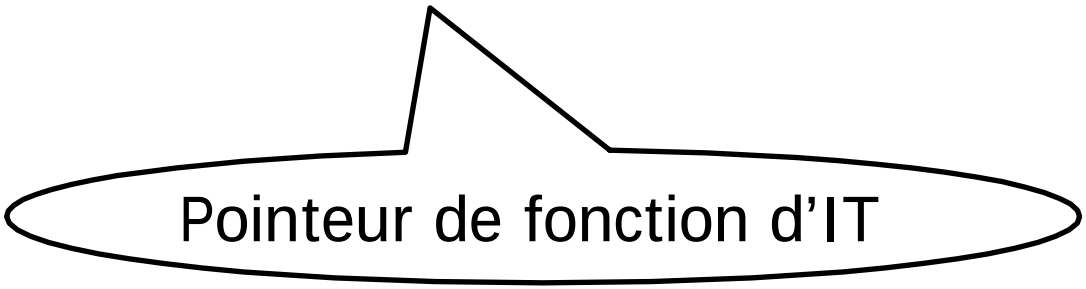
Récapitulatif 1/2

- Déclaration d'un pointeur sur (fonction d'interruption)
  - void interrupt (*old_handler)() en C
  - void interrupt (*old_handler)(...) en C++
- Déclaration d'une fonction d'interruption
  - void interrupt new_handler () en C
  - void interupt new_handler(...) en C++
- Attention ! Pour des interruptions matérielles, on ne peut passer aucun paramètre à la fonction d'IT et de même cette fonction ne renvoie aucun paramètre.
  Ce qui est tout à fait logique, puisque ce n'est pas le programme principal qui appelle l'IT mais c'est un signal électrique externe qui l'a provoque.

Récapitulatif 2/2

- Pour affecter l'@ de l'ancien vecteur d'IT au pointeur de fonction d'interruption
  - old_handler = getvect(N°du_Vecteur_d'IT)

- Pour installer une nouvelle fonction d'IT (c.-à.-d. mettre l'@ du pointeur de la fonction dans la table des IT)
  - setvect(N°du_Vecteur_d'IT, new_handler)



Bonnes habitudes de programmation 1/2

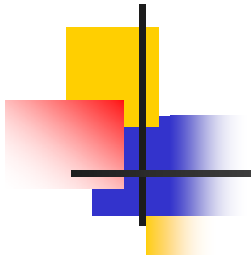
- Après avoir exécuté ses propres instructions, il conviendrait d'appeler l'ancien vecteur d'IT. Ceci peut s'avérer utile si plusieurs programmes ont dérouté une IT avant vous. Dans ce cas, il ne faut pas empêcher la bonne exécution des anciens programmes.

```
Void interrupt new_handler(…)
  { …
    …
    …
    old_handler(…)        // On exécute l'ancien IT
                          // avant de sortir
  }
```

Bonnes habitudes de programmation 2/2

- Quand l'exécution du programme d'IT est terminée, il est d'usage de désinstaller toute fonction d'interruption et de réinstaller l'ancien vecteur d'interruption

```
Void main()
  { …
    while()
          { … }          //programme interrompu
    // avant de quitter main on réinstalle
    // l'ancienne IT
    setvect(N°vect_it,old_handler(…));
  }
```

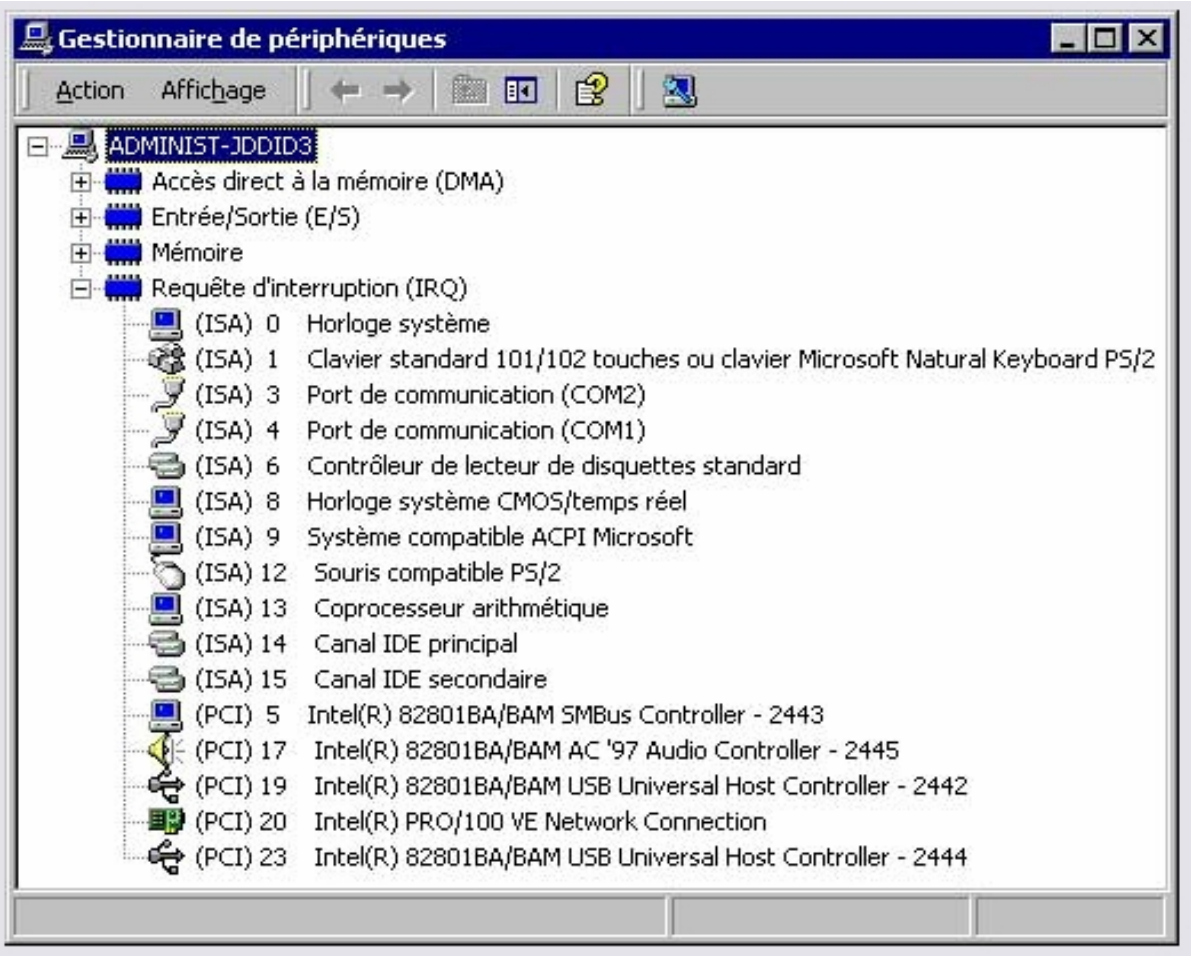


Règle d’or

- L’exécution d’une interruption doit être le plus rapide possible, afin d’éviter la perte de données, la perte de demande d’IT, ...

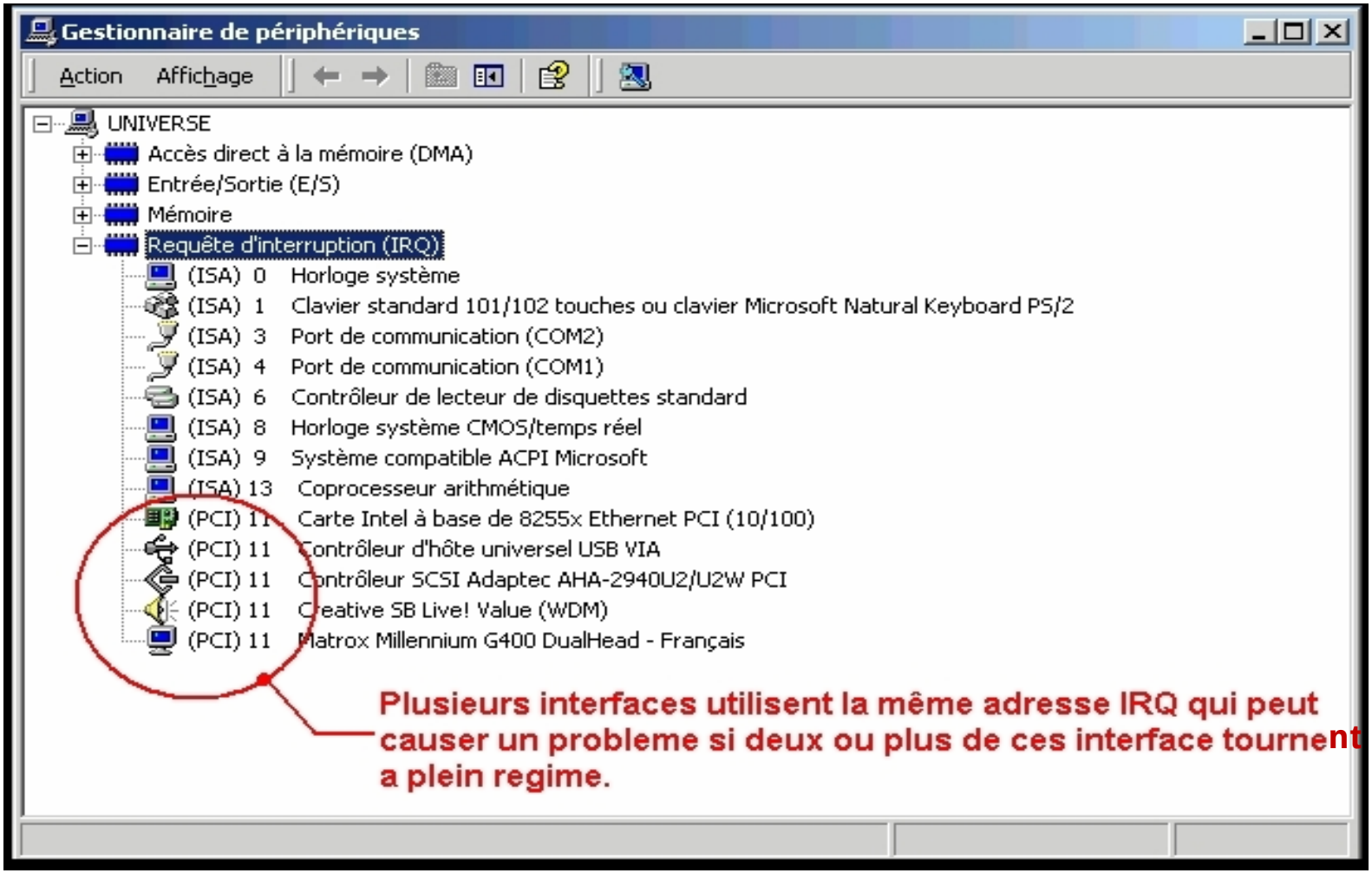
- On évite les boucles longue ou infinie dans des programmes d’IT

Connaître les interruptions de son PC

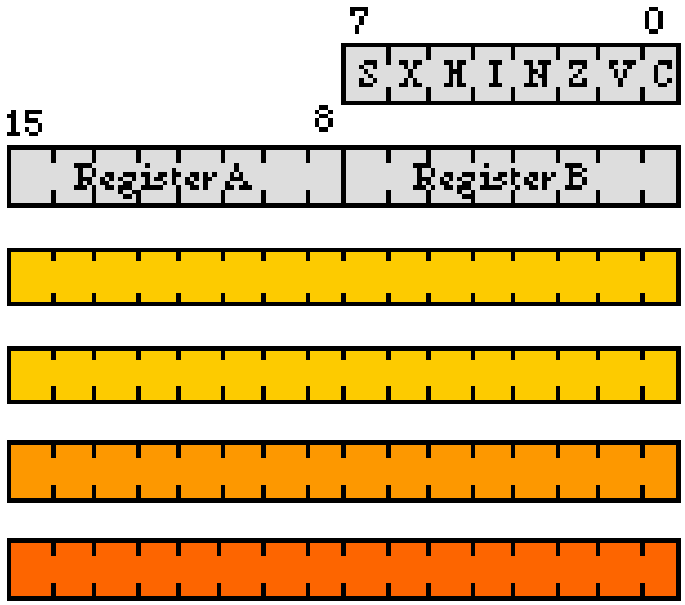


Aller dans l'icône gestionnaire d'interruptions:
Système>gestionnaires de périphériques> interruptions

Attention à la surcharge



HC 12



CC 8 bit condition code

D two 8 bit accumulators

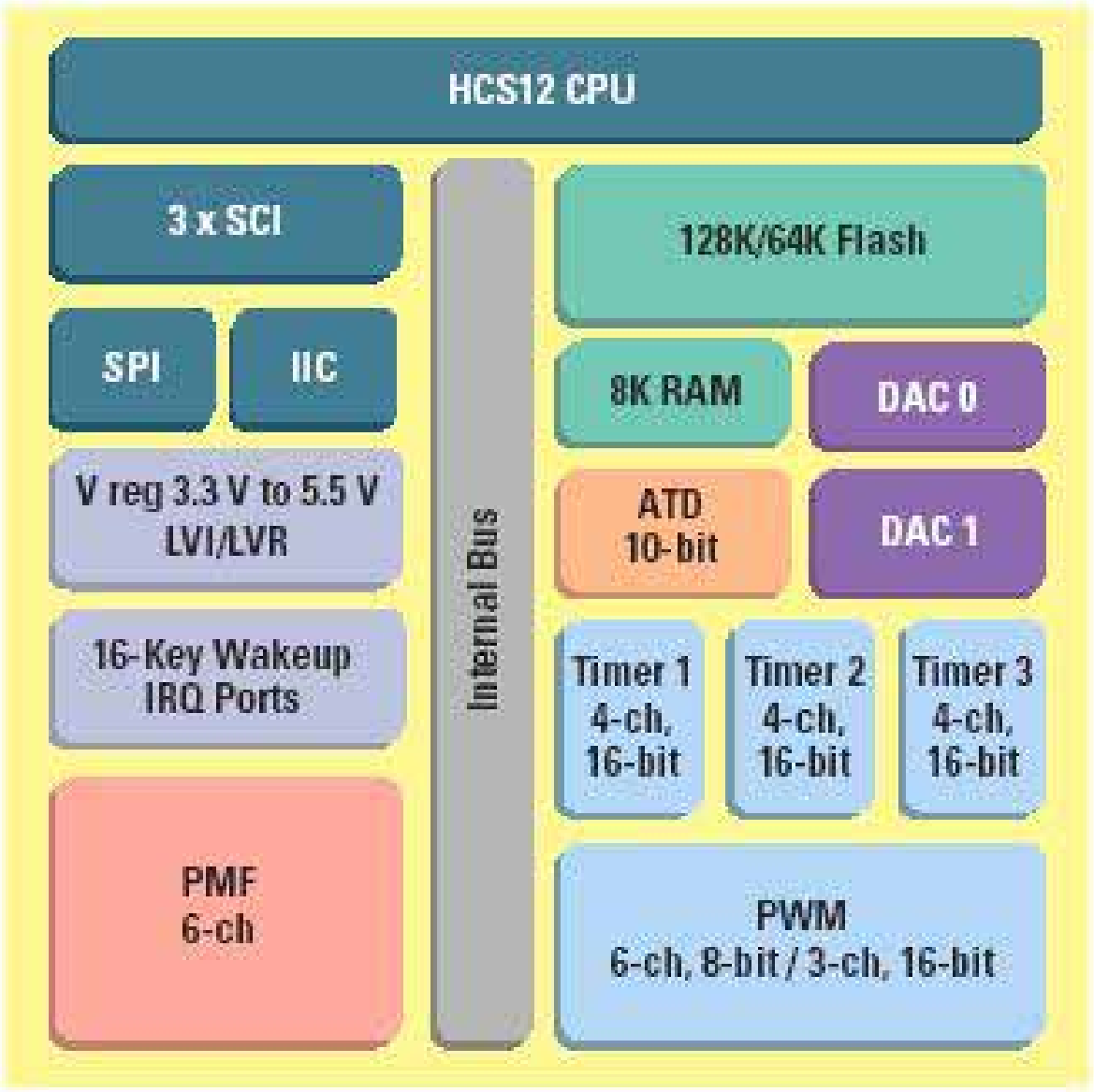
X 16 bit index register

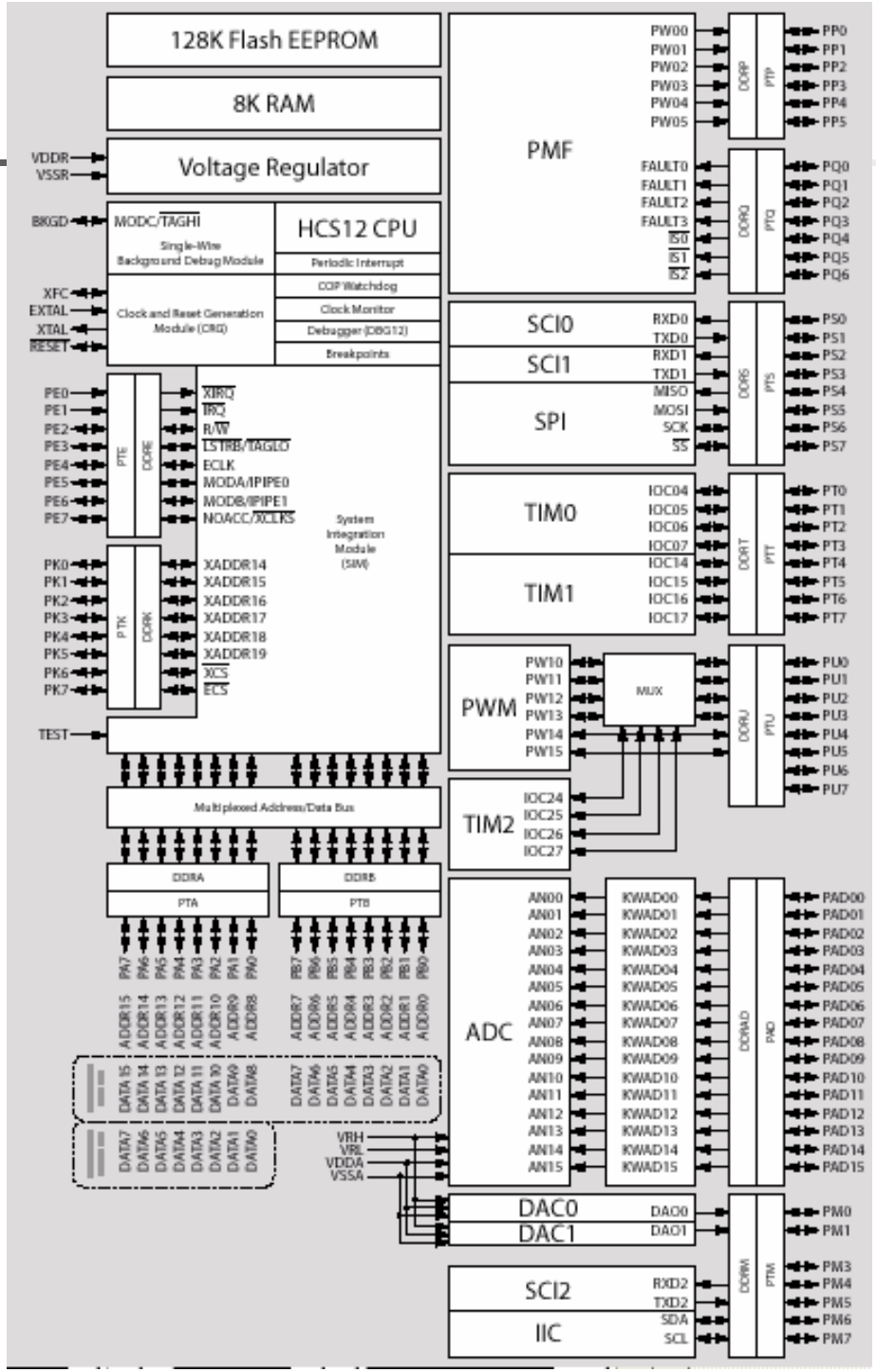
Y 16 bit index register

SP 16 bit stack pointer

PC 16 bit program counter

Structure HC12




128K Flash EEPROM
8K RAM
Voltage Regulator
VDDR
VSSR
BKGD
MODC/TAGHI
Single-Wire Background Debug Module
HCS12 CPU
Periodic Interrupt
COP Watchdog
Clock Monitor
Debugger (DBG12)
Breakpoints
XFC
EXTAL
XTAL
RESET
Clock and Reset Generation Module (CRG)
PE0
PE1
PE2
PE3
PE4
PE5
PE6
PE7
PTE
DDRE
XIRQ
IRQ
R/W
LSTRB/TAGLO
ECLK
MODA/IPIPE0
MODB/IPIPE1
NOACC/XCLKS
System Integration Module (SIM)
PK0
PK1
PK2
PK3
PK4
PK5
PK6
PK7
PTK
DDRK
XADDR14
XADDR15
XADDR16
XADDR17
XADDR18
XADDR19
XCS
ECS
TEST
Multiplexed Address/Data Bus
DDRA
PTA
DDRB
PTB
PA7 PA6 PA5 PA4 PA3 PA2 PA1 PA0
PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0
ADDR15 ADDR14 ADDR13 ADDR12 ADDR11 ADDR10 ADDR9 ADDR8
ADDR7 ADDR6 ADDR5 ADDR4 ADDR3 ADDR2 ADDR1 ADDR0
DATA 15 DATA 14 DATA 13 DATA 12 DATA 11 DATA 10 DATA9 DATA8
DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
VRH
VRL
VDDA
VSSA
PMF
PW00
PW01
PW02
PW03
PW04
PW05
DDRP
PTP
PP0
PP1
PP2
PP3
PP4
PP5
FAULT0
FAULT1
FAULT2
FAULT3
IS0
IS1
IS2
DDRQ
PTQ
PQ0
PQ1
PQ2
PQ3
PQ4
PQ5
PQ6
SCI0
RXD0
TXD0
SCI1
RXD1
TXD1
SPI
MISO
MOSI
SCK
SS
DDRS
PTS
PS0
PS1
PS2
PS3
PS4
PS5
PS6
PS7
TIM0
IOC04
IOC05
IOC06
IOC07
TIM1
IOC14
IOC15
IOC16
IOC17
DDRT
PTT
PT0
PT1
PT2
PT3
PT4
PT5
PT6
PT7
PWM
PW10
PW11
PW12
PW13
PW14
PW15
MUX
DDRU
PTU
PU0
PU1
PU2
PU3
PU4
PU5
PU6
PU7
TIM2
IOC24
IOC25
IOC26
IOC27
ADC
AN00
AN01
AN02
AN03
AN04
AN05
AN06
AN07
AN08
AN09
AN10
AN11
AN12
AN13
AN14
AN15
KWAD00
KWAD01
KWAD02
KWAD03
KWAD04
KWAD05
KWAD06
KWAD07
KWAD08
KWAD09
KWAD10
KWAD11
KWAD12
KWAD13
KWAD14
KWAD15
DDRAD
PAD
PAD00
PAD01
PAD02
PAD03
PAD04
PAD05
PAD06
PAD07
PAD08
PAD09
PAD10
PAD11
PAD12
PAD13
PAD14
PAD15
DAC0
DAC1
DAO0
DAO1
DDRM
PTM
PM0
PM1
SCI2
RXD2
TXD2
IIC
SDA
SCL
PM3
PM4
PM5
PM6
PM7

Plan mémoire du HC12:

- Selon le mode de démarrage, le 68HC12 possède divers plans mémoire. Le problème avec les interruptions est d'initialiser les vecteurs en mémoire de $FF80-$FFFF.

Cas possibles :

Mode étendu

Vecteurs en mémoire externe

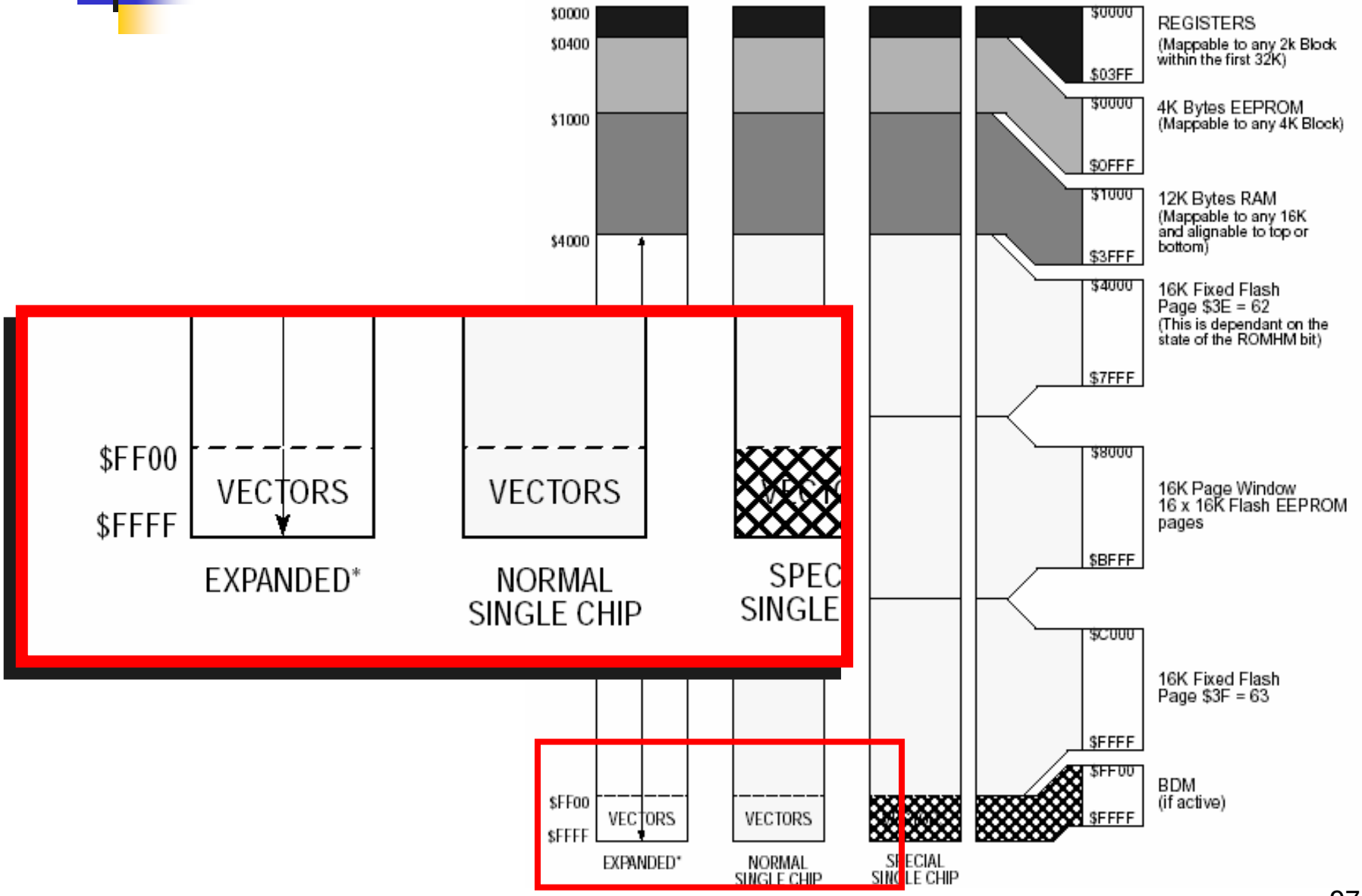
Mode Single Chip normal

Vecteurs en Flash EEprom au démarrage

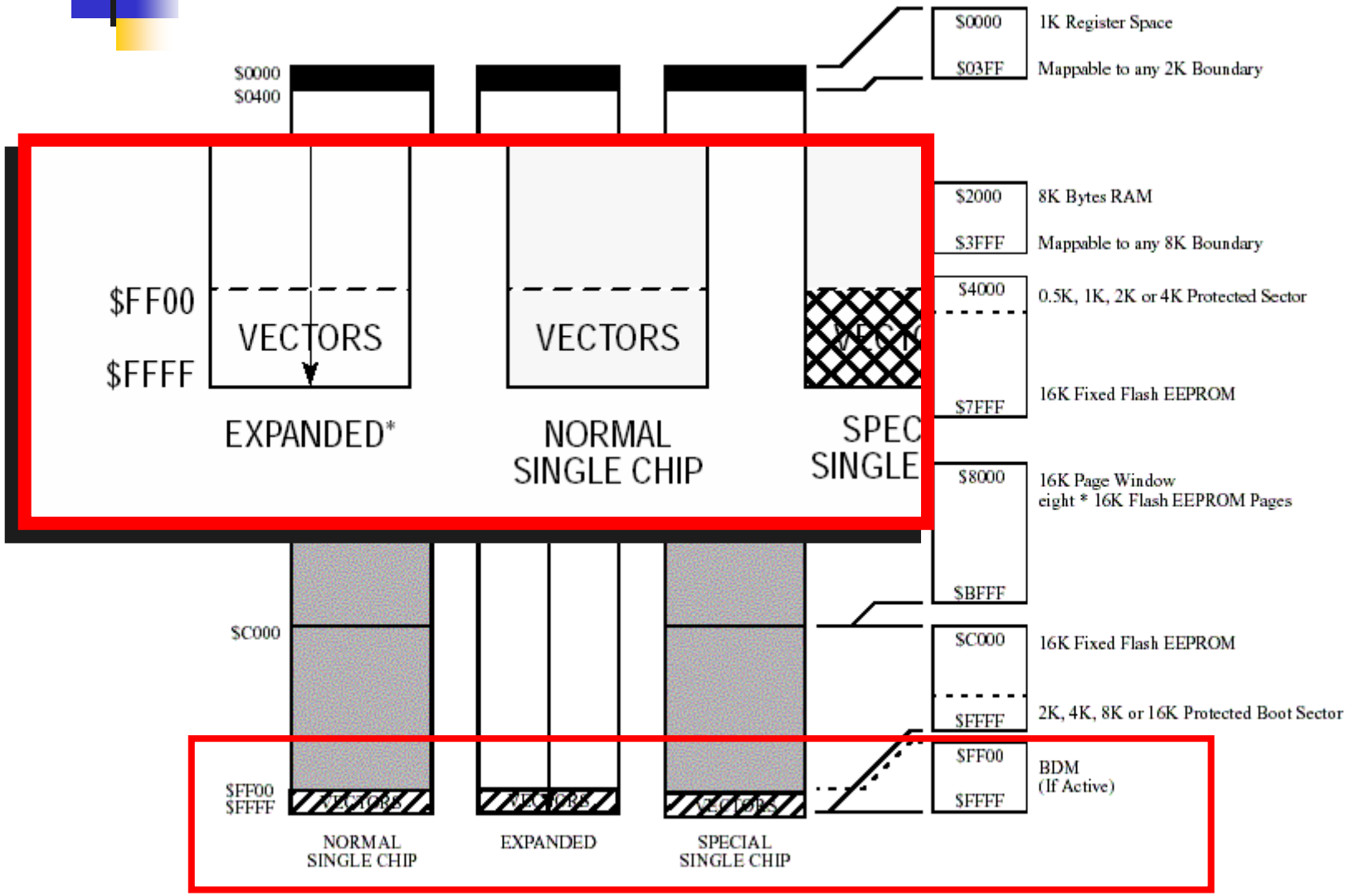
Mode single chip special (BDI, développement)

Démarrage par BDI, code du logiciel caché en $FF00--$FFFF

Plan mémoire du MC9S12DP256



Plan mémoire du MC9S12E128



The figure shows a useful map, which is not the map out of reset. After reset the map is:

$0000 - $03FF: Register Space
$0000 - $1FFF: 8K RAM (only 7K RAM visible $0400 - $1FFF)

Table des vecteurs du MC9S12

Table 7-1. CPU12 Exception Vector Map[1]

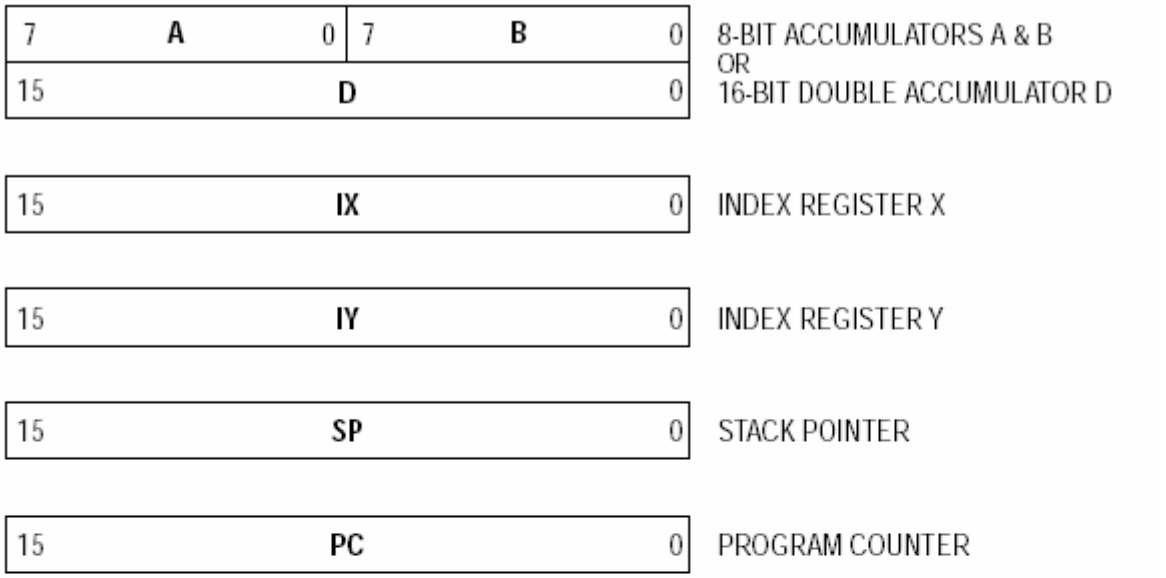
Vector Address	Source
$FFFE–$FFFF	System reset
$FFFC–$FFFD	Clock monitor reset
$FFFA–$FFFB	COP reset
$FFF8–$FFF9	Unimplemented opcode trap
$FFF6–$FFF7	Software interrupt instruction (SWI)
$FFF4–$FFF5	$\overline{\text{XIRQ}}$ signal
$FFF2–$FFF3	$\overline{\text{IRQ}}$ signal
$FF00–$FFF1	Device-specific interrupt sources (HCS12)
$FFC0–$FFF1	Device-specific interrupt sources (M68HC12)

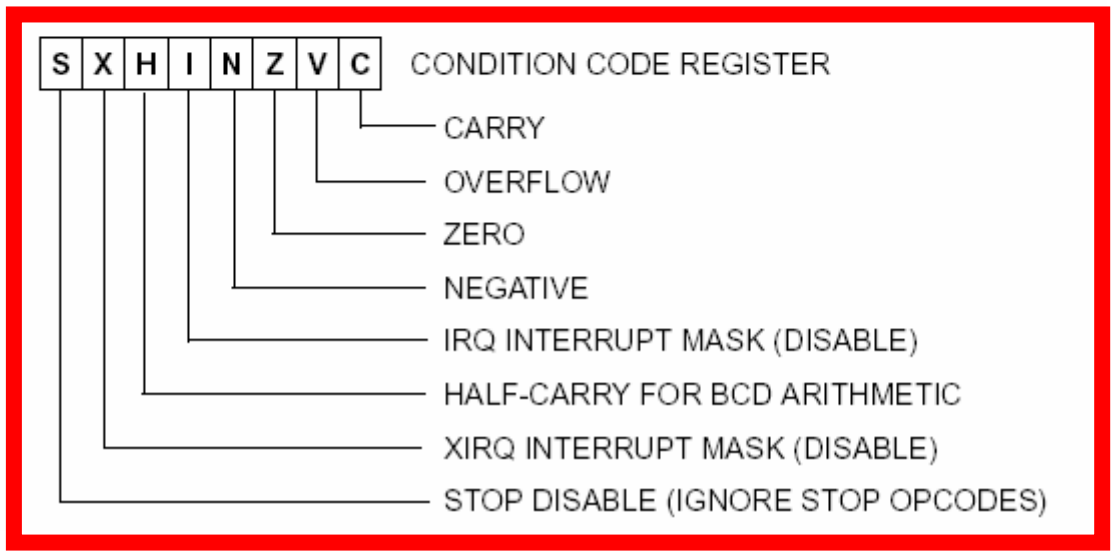
1. See Device User Guide and Interrupt Block Guide for further details

Table des vecteurs du MC9S12 (suite)

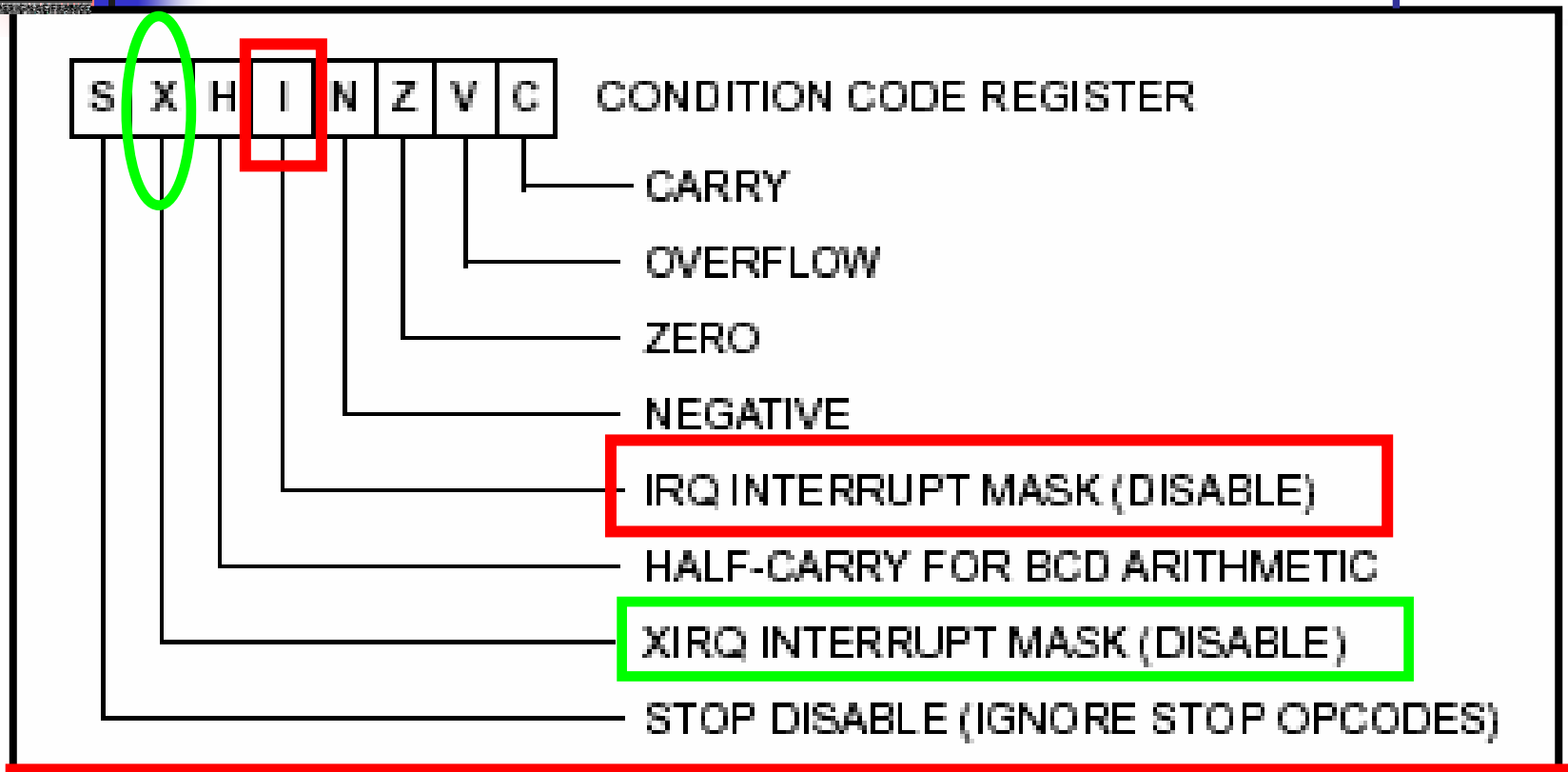
$FFE4, $FFE5	Timer channel 5	I-Bit	TMSK1 (C5I)
$FFE2, $FFE3	Timer channel 6	I-Bit	TMSK1 (C6I)
$FFE0, $FFE1	Timer channel 7	I-Bit	TMSK1 (C7I)
$FFDE, $FFDF	Timer overflow	I-Bit	TMSK2 (TOI)
$FFDC, $FFDD	Pulse accumulator A overflow	I-Bit	PACTL (PAOVI)
$FFDA, $FFDB	Pulse accumulator input edge	I-Bit	PACTL (PAI)
$FFD8, $FFD9	SPI0	I-Bit	SP0CR1 (SPIE, SPTIE)
$FFD6, $FFD7	SCI 0	I-Bit	SC0CR2 (TIE, TCIE, RIE, ILIE)
$FFD4, $FFD5	SCI 1	I-Bit	SC1CR2 (TIE, TCIE, RIE, ILIE)
$FFD2, $FFD3	ATD0	I-Bit	ATD0CTL2 (ASCIE)
$FFD0, $FFD1	ATD1	I-Bit	ATD1CTL2 (ASCIE)
$FFCE, $FFCF	Port J	I-Bit	PTJIF (PTJIE)
$FFCC, $FFCD	Port H	I-Bit	PTHIF(PTHIE)
$FFCA, $FFCB	Modulus Down Counter underflow	I-Bit	MCCTL(MCZI)
$FFC8, $FFC9	Pulse Accumulator B Overflow	I-Bit	PBCTL(PBOVI)
$FFC6, $FFC7	CRG lock	I-Bit	PLLCR(LOCKIE)
$FFC4, $FFC5	SCME	I-Bit	PLLCR (SCMIE)
$FFC2, $FFC3	DLC	I-Bit	DLCBCR1(IE)
$FFC0, $FFC1	IIC Bus	I-Bit	IBCR (IBIE)
$FFBE, $FFBF	SPI1	I-Bit	SP1CR1 (SPIE. SPTIE)
$FFBC, $FFBD	SPI2	I-Bit	SP2CR1 (SPIE, SPTIE)
$FFBA, $FFBB	EEPROM	I-Bit	EECTL(CCIE, CBEIE)
$FFB8, $FFB9	FLASH	I-Bit	FCTL(CCIE, CBEIE)
$FFB6, $FFB7	MSCAN 0 wake-up	I-Bit	C0RIER (WUPIE)

Modèle de programmation du HC12





Autorisation Globale des interruptions



CLI	Clear I; assembles as ANDCC #$EF
CLR *opr16a*	Clear M: $00→M

SEI	Set I; assembles as ORCC #$10

Real Time Interrupt RTi sur HC12

Extrait documentation

Real Time Interrupt (RTI)

The RTI can be used to generate a hardware interrupt at a fixed periodic rate. If enabled (by setting RTIE=1), this interrupt will occur at the rate selected by the RTICTL register. The RTI runs with a gated

OSCCLK (see **Figure 4-6 Clock Chain for RTI**). At the end of the RTI time-out period the RTIF flag is set to one and a new RTI time-out period starts immediately.

A write to the RTICTL register restarts the RTI time-out period.

If the PRE bit is set, the RTI will continue to run in Pseudo-Stop Mode.

.

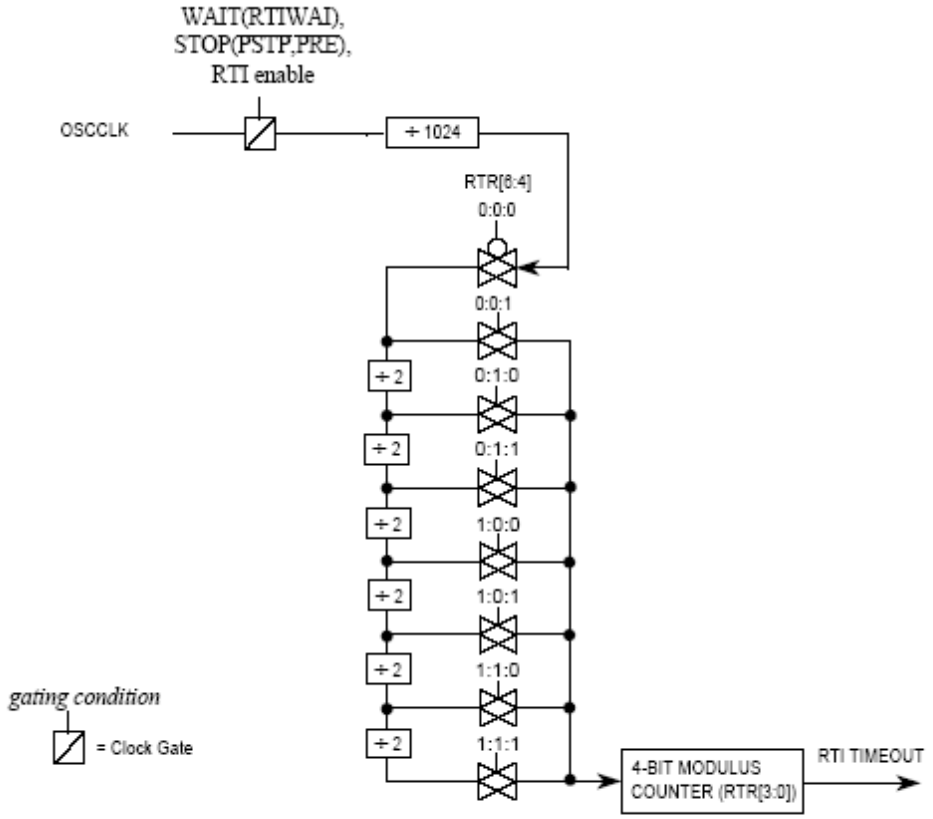


Figure 4-6 Clock Chain for RTI

RTR[6:4] — Real Time Interrupt Prescale Rate Select Bits

These bits select the prescale rate for the RTI. See **Table 3-2**.

RTR[3:0] — Real Time Interrupt Modulus Counter Select Bits

These bits select the modulus counter target value to provide additional granularity. **Table 3-2** shows all possible divide values selectable by the RTICTL register. The source clock for the RTI is OSCCLK.

Table 3-2 RTI Frequency Divide Rates

RTR[3:0]	RTR[6:4] =							
	000 (OFF)	001 (2^{10})	010 (2^{11})	011 (2^{12})	100 (2^{13})	101 (2^{14})	110 (2^{15})	111 (2^{16})
0000 (÷1)	OFF*	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
0001 (÷2)	OFF*	$2x2^{10}$	$2x2^{11}$	$2x2^{12}$	$2x2^{13}$	$2x2^{14}$	$2x2^{15}$	$2x2^{16}$
0010 (÷3)	OFF*	$3x2^{10}$	$3x2^{11}$	$3x2^{12}$	$3x2^{13}$	$3x2^{14}$	$3x2^{15}$	$3x2^{16}$
0011 (÷4)	OFF*	$4x2^{10}$	$4x2^{11}$	$4x2^{12}$	$4x2^{13}$	$4x2^{14}$	$4x2^{15}$	$4x2^{16}$
0100 (÷5)	OFF*	$5x2^{10}$	$5x2^{11}$	$5x2^{12}$	$5x2^{13}$	$5x2^{14}$	$5x2^{15}$	$5x2^{16}$
0101 (÷6)	OFF*	$6x2^{10}$	$6x2^{11}$	$6x2^{12}$	$6x2^{13}$	$6x2^{14}$	$6x2^{15}$	$6x2^{16}$
0110 (÷7)	OFF*	$7x2^{10}$	$7x2^{11}$	$7x2^{12}$	$7x2^{13}$	$7x2^{14}$	$7x2^{15}$	$7x2^{16}$
0111 (÷8)	OFF*	$8x2^{10}$	$8x2^{11}$	$8x2^{12}$	$8x2^{13}$	$8x2^{14}$	$8x2^{15}$	$8x2^{16}$
1000 (÷9)	OFF*	$9x2^{10}$	$9x2^{11}$	$9x2^{12}$	$9x2^{13}$	$9x2^{14}$	$9x2^{15}$	$9x2^{16}$
1001 (÷10)	OFF*	$10x2^{10}$	$10x2^{11}$	$10x2^{12}$	$10x2^{13}$	$10x2^{14}$	$10x2^{15}$	$10x2^{16}$

Table 3-2 RTI Frequency Divide Rates

RTR[3:0]	RTR[6:4] =							
1010 (÷11)	OFF*	$11x2^{10}$	$11x2^{11}$	$11x2^{12}$	$11x2^{13}$	$11x2^{14}$	$11x2^{15}$	$11x2^{16}$
1011 (÷12)	OFF*	$12x2^{10}$	$12x2^{11}$	$12x2^{12}$	$12x2^{13}$	$12x2^{14}$	$12x2^{15}$	$12x2^{16}$
1100 (÷ 13)	OFF*	$13x2^{10}$	$13x2^{11}$	$13x2^{12}$	$13x2^{13}$	$13x2^{14}$	$13x2^{15}$	$13x2^{16}$
1101 (÷14)	OFF*	$14x2^{10}$	$14x2^{11}$	$14x2^{12}$	$14x2^{13}$	$14x2^{14}$	$14x2^{15}$	$14x2^{16}$
1110 (÷15)	OFF*	$15x2^{10}$	$15x2^{11}$	$15x2^{12}$	$15x2^{13}$	$15x2^{14}$	$15x2^{15}$	$15x2^{16}$
1111 (÷ 16)	OFF*	$16x2^{10}$	$16x2^{11}$	$16x2^{12}$	$16x2^{13}$	$16x2^{14}$	$16x2^{15}$	$16x2^{16}$

* Denotes the default value out of reset. This value should be used to disable the RTI to ensure future backwards compatibility.

0x3B

0x1F

0x4F

0x7F

CRG RTI Control Register (RTICTL)

e — V02.07

This register selects the timeout period for the Real Time Interrupt.

Address Offset: $_07

	7	6	5	4	3	2	1	0
R	0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-8 CRG RTI Control Register (RTICTL)

Read: anytime

Write: anytime

NOTE: *A write to this register initializes the RTI counter.*

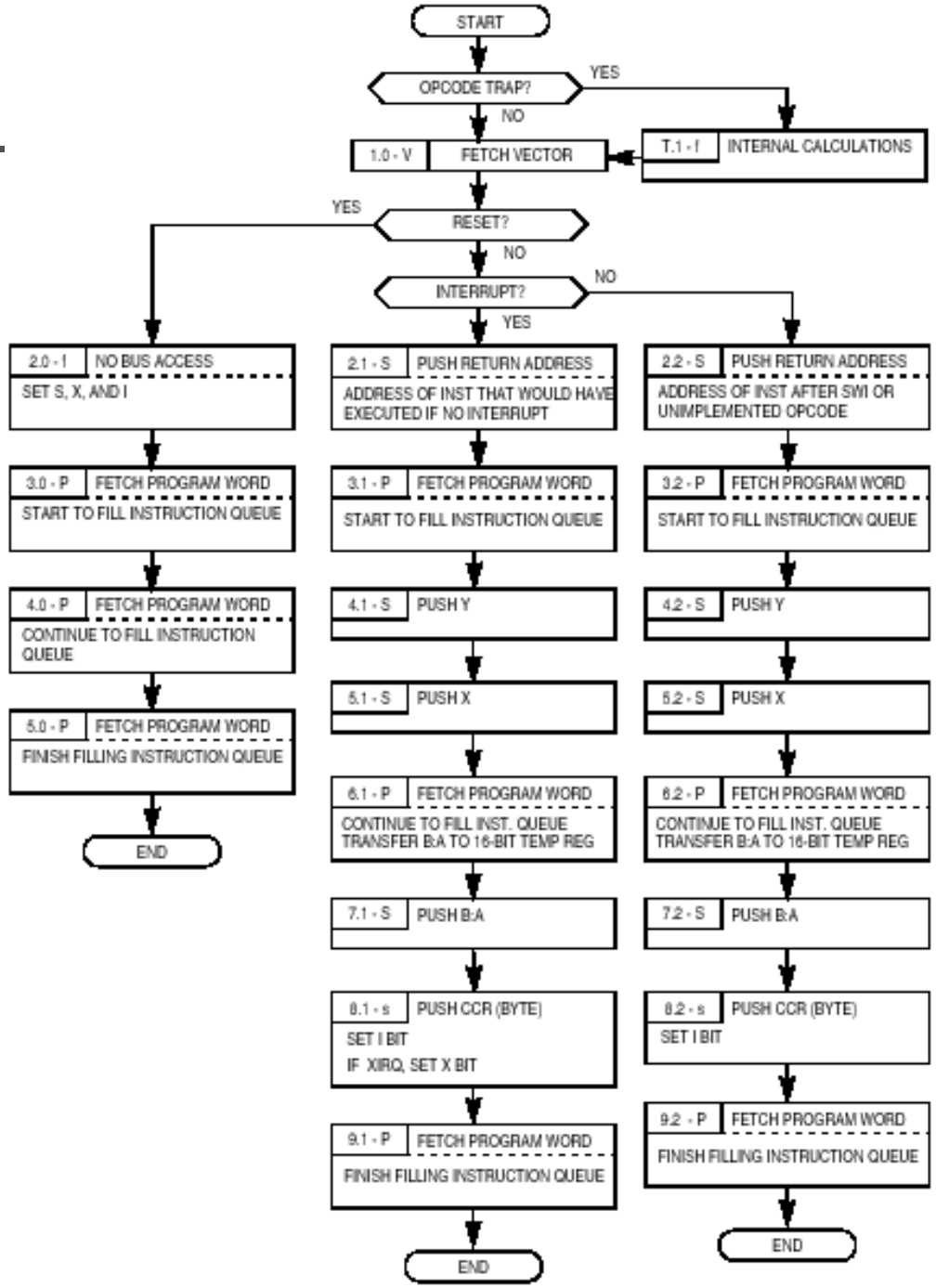
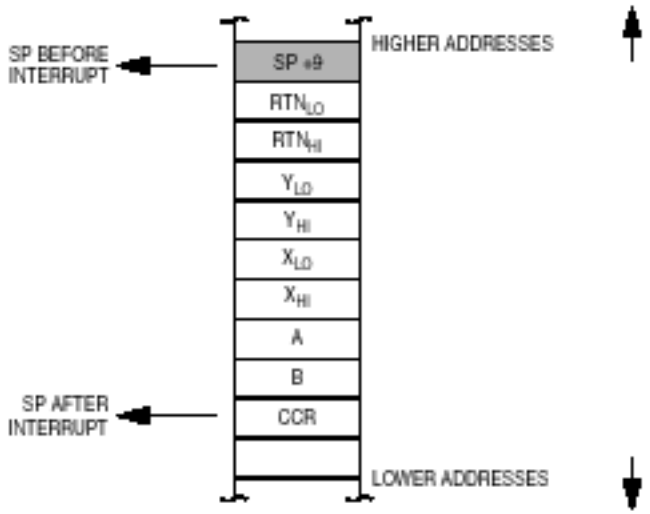


Figure 7-1. Exception Processing Flow Diagram

A.2 Stack and Memory Layout



STACK UPON ENTRY TO SERVICE ROUTINE
IF SP WAS ODD BEFORE INTERRUPT

SP +8	RTN_{LO}		SP +9
SP +6	Y_{LO}	RTN_{HI}	SP +7
SP +4	X_{LO}	Y_{HI}	SP +5
SP +2	A	X_{HI}	SP +3
SP	CCR	B	SP +1
SP –2			SP –1

STACK UPON ENTRY TO SERVICE ROUTINE
IF SP WAS EVEN BEFORE INTERRUPT

SP +9			SP +10
SP +7	RTN_{HI}	RTN_{LO}	SP +8
SP +5	Y_{HI}	Y_{LO}	SP +6
SP +4	X_{HI}	X_{LO}	SP +4
SP +1	B	A	SP +2
SP –1		CCR	SP

A.3 Interrupt Vector Locations

$FFFE, $FFFF	Power-On (POR) or External Reset
$FFFC, $FFFD	Clock Monitor Reset
$FFFA, $FFFB	Computer Operating Properly (COP Watchdog Reset
$FFF8, $FFF9	Unimplemented Opcode Trap
$FFF6, $FFF7	Software Interrupt Instruction (SWI)
$FFF4, $FFF5	XIRQ
$FFF2, $FFF3	IRQ
$FFC0–$FFF1 (M68HC12)	Device-Specific Interrupt Sources
$FF00–$FFF1 (HCS12)	Device-Specific Interrupt Sources

PACKAGE OPTIONS

PART NUMBER	PACKAGE	TEMPERATURE RANGE
MC9S12C32CFA	48 LQFP	-40°C to +85°C
MC9S12C32VFA	48 LQFP	-40°C to +105°C
MC9S12C32MFA	48 LQFP	-40°C to +125°C
MC9S12C32CPB	52 LQFP	-40°C to +85°C
MC9S12C32VPB	52 LQFP	-40°C to +105°C
MC9S12C32MPB	52 LQFP	-40°C to +125°C
MC9S12C32CFU	80 QFP	-40°C to +85°C
MC9S12C32VFU	80 QFP	-40°C to +105°C
MC9S12C32MFU	80 QFP	-40°C to +125°C

48-Pin LQFP
.5 mm Pitch
7 mm x 7 mm Body

52-Pin LQFP
.65 mm Pitch
10 mm x 10 mm Body

80-Lead QFP/LQFP
.65 mm Pitch
14 mm x 14 mm Body

MOTOROLA

Metrowerks CodeWarrior - [main.c]

File Edit View Search Project Debug Processor Expert Window Help

Path: D:\Prog_FileD\Programmation\Metrowerks\CW12V31\(CodeWarrior_Examples)\MC9S12\SofTec

```
////////////////////////////////////////////////////////////////////////////////
// ADC Sample for SofTec Microsystems PK-HCS12E128 Starter Kit
//
// By rotating the potentiometer (PAD00 channel 0), you affect the results
// of the A/D conversion, and the value of each conversion is displayed on the
// LEDs on PT[7..0].
// ---------------------------------------------------------------------------
// Copyright (c) 2003 SofTec Microsystems
// http://www.softecmicro.com/
////////////////////////////////////////////////////////////////////////////////
#include <hidef.h>                      // Common defines and macros
#include <mc9s12e128.h>                 // Derivative information
#pragma LINK_INFO DERIVATIVE "SampleS12"
// ---------------------------------------------------------------------------
// Puts the following code into NOT BANKED FLASH MEMORY
// ---------------------------------------------------------------------------
#pragma CODE_SEG __NEAR_SEG NON_BANKED
// ---------------------------------------------------------------------------
// Peripheral Initialization Routine
// ---------------------------------------------------------------------------
void PeriphInit(void)
{
    PTT = 0x00;                         // Clears port T[7..0]
    DDRT = 0xFF;                        // Configures port T[7..0] as output

    ATDCTL3 = 0x08;                     // One conversion per sequence
    ATDCTL4_SRES8 = 1;                  // Sets A/D conversion to 8 bit resolution
    ATDCTL5_SCAN = 1;                   // Continuous conversion (scan mode)
    ATDCTL2 = 0x80;                     // Normal ATD functionality
}
// ---------------------------------------------------------------------------
// Main
// ---------------------------------------------------------------------------
void main(void)
{

    PeriphInit();                       // Microcontroller initialization
    EnableInterrupts;                   // Enables interrupts

    for(;;)                             // Forever
    {
        if(ATDSTAT1_CCF0)               // If ATD conversion complete flag 0
            PTT = ATDDR0H;              // Writes the ATD value on Port T
    }
}
```

ABREVIATIONS

- 1 Liées aux interruptions
  - **IT**: *Interrupt*: Interruption
  - **ISR**: *Interrupt Service Routine*: Routine de service d'interruption.
  - **IRQ**: *Interrupt Request*: Requête (demande) d'interruption.
  - **IACK**: *Interrupt Acknowledge*: Accusé de réception d'une IRQ.
  - **NMI**: *Non-Maskable Interrupt*: Interruption non masquable
  - **IF**: *Interrupt Flag*: Bit signalant une requête d'interruption.
  - **IE**: *Interrupt Enable*: Bit programmé pour autoriser/inhiber une interruption.

- 2 Autres abréviations
  - **CPU**: *Central Processing Unit*: Processeur, le coeur du microprocesseur ou microcontroleur.
  - **MCU**: *Microcontroller*: Microcontrôleur
  - **MPU**: *Microprocessing Unit*: Microprocesseur
  - **RTOS**: *Real Time Operating System*: Noyau temps réel multitâche.
  - **BIOS**: *Basic Input / Output System*: Couche logicielle pour les périphériques d'entrée/sortie.
  - **UART**: *Universal Asychronous Receiver Transmitter*: Liaison série asynchrone.
  - BUS **SPI**: *Serial Peripheral Interface*: Liaison série synchrone, inventé par Motorola
  - BUS **IIC** ou **I2C** : *Inter Integrated Circuit* : Bus série avec adressage, inventé par Phillips
  - BUS **CAN** : *Car Area Network* : Réseau de terrain industriel dédié à l'automobile, inventé par Bosh.
  - **ADC** : ***ATD*** *Analog to Digital Converter* : Convertisseur Analogique Numérique (CAN en français)
  - **DAC**: *Digital to Analog Converter* : Convertisseur Numérique Analogique (CNA en français)
  - **FIFO**: *First In Fisrt Out* : File d'attente (le contraire d'une pile).