Interruptions

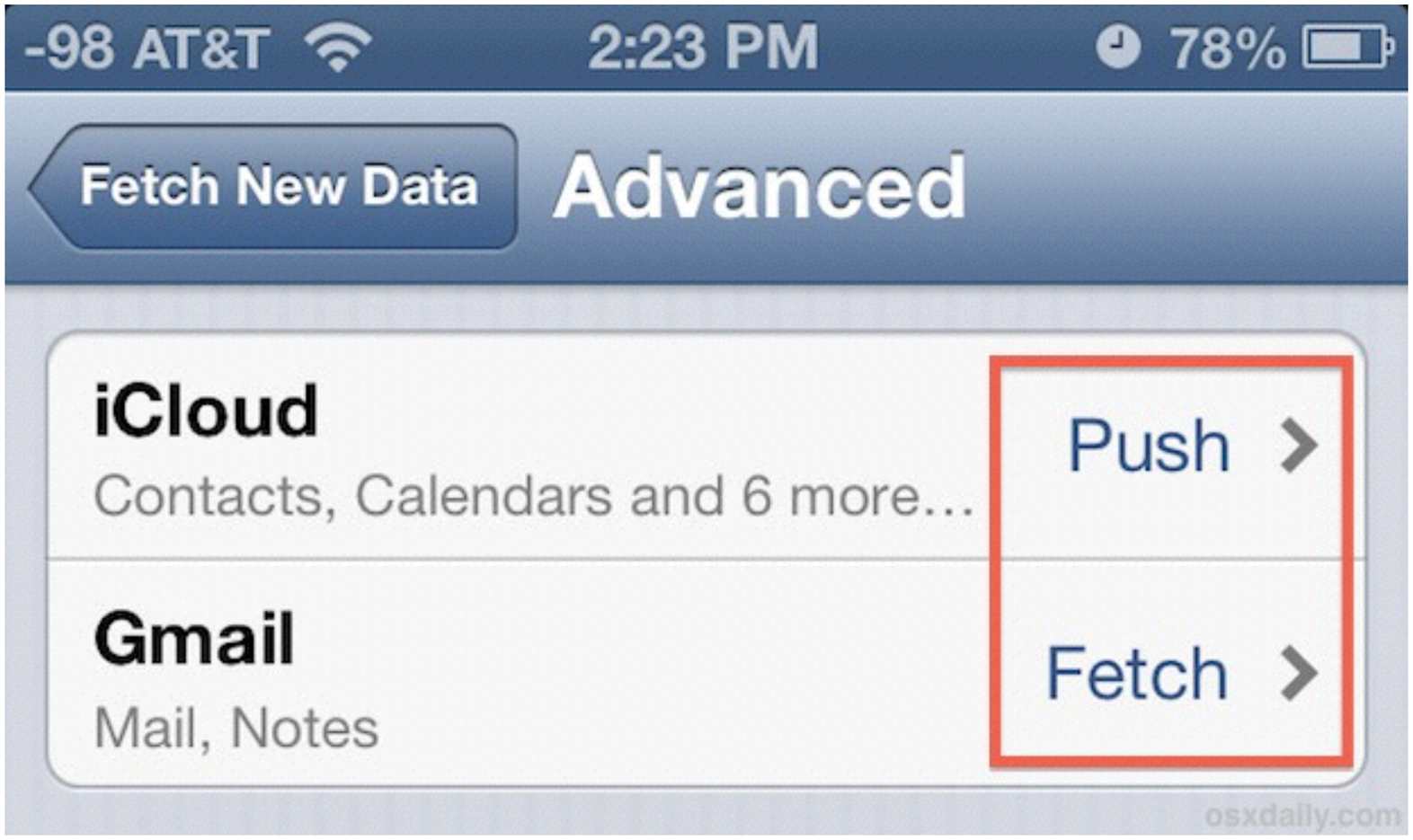
GIF-1001 Ordinateurs: Structure et Applications, Hiver 2016

Jean-François Lalonde

Aujourd'hui: interruptions

- Pensons à un micro-processeur qui voudrait lire les valeurs écrites au clavier par un utilisateur.
- Comment faire?
  - Option #1: demander au clavier périodiquement s'il a reçu une nouvelle touche, sinon, attendre!
  - Option #2: c'est le clavier qui « dit » au micro-processeur qu'il a reçu une nouvelle touche!

Interruptions



Types d'interruptions

- **Système**: reset, faute matérielle générale, etc.
- **Exception**: le processeur peut générer des interruptions s'il n'est pas capable de lire ou d'exécuter une instruction (opcode invalide, division par 0, mémoire protégée, etc).
- **Matérielles**: générées par les périphériques
- **Logicielles**: le programmeur (nous!) peut générer une interruption sur demande

7 interruptions en ARM

Interruption	Signification
Reset	redémarrage
Instruction indéfinie	problème lors du décodage
Interruption logicielle	demandée par le programmeur (instruction `SWI`)
« Prefetch abort »	accès mémoire invalide lors du « fetch » (lecture)
« Data abort »	accès mémoire invalide
IRQ	« Interrupt ReQuest » : interruption « générale »
FIQ	« Fast Interrupt reQuest » : interruption générale rapide

Une interruption survient… que faire?

Source: soundbible.com

Source: soundbible.com

Source: soundbible.com

Source: soundbible.com

Une interruption survient… que faire?

1. Terminer l'instruction en cours
2. Déterminer s'il faut traiter l'interruption.
3. Sauvegarder le contexte
4. Déterminer l'adresse de la routine de traitement de l'interruption
5. Exécuter cette routine

Interruptions

- Une interruption interrompt l'exécution des instructions par le microprocesseur.

- Lors d'une interruption:

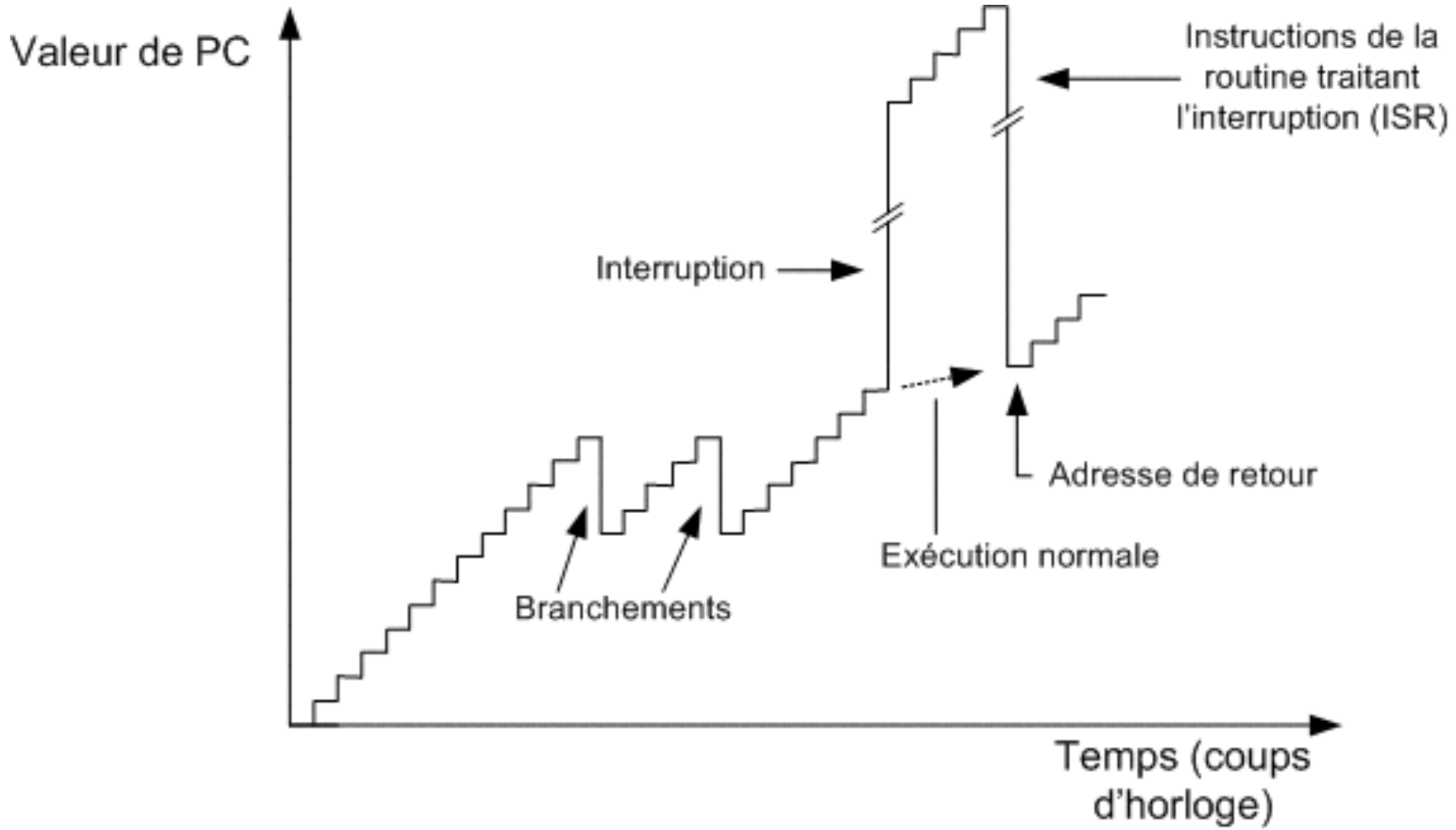
  1. l'exécution du programme principal est suspendue;

  2. une routine (fonction) traitant l'interruption est exécutée;

  3. puis le programme principal est continué.

- Quelle est la différence entre une interruption et un branchement?

  - les interruptions peuvent survenir n'importe quand pendant l'exécution.


Valeur de PC
Instructions de la
routine traitant
l'interruption (ISR)
Interruption
Adresse de retour
Exécution normale
Branchements
Temps (coups
d'horloge)

Une interruption survient… que faire?

1. Terminer l'instruction en cours
2. Déterminer s'il faut traiter l'interruption.
3. Sauvegarder le contexte
4. **Déterminer l'adresse** de la routine de traitement de l'interruption
5. Exécuter cette routine

Routines de traitement d'interruptions

- Les routines de traitement d'interruptions sont des fonctions « spéciales » que l'on appelle que pour traiter les interruptions
- Où sont-elles situées?
  - en mémoire!
- Comment fait-on pour savoir:
  - quelle routine exécuter pour quelle interruption?
  - à quelle adresse est cette routine?
- Grâce à la table des vecteurs d'interruption, pardi!

Table des vecteurs d'interruption

Adresse	Interruption	Signification
0x00	Reset	redémarrage
0x04	Instruction indéfinie	décodage
0x08	Interruption logicielle	instruction SWI
0x0C	« Prefetch abort »	« fetch » invalide
0x10	« Data abort »	accès mémoire invalide
0x14	Espace réservé	ne rien mettre ici
0x18	IRQ	« Interrupt ReQuest »
0x1C	FIQ	« Fast Interrupt reQuest »

- Chaque entrée de la table « branche » vers la routine correspondante

Exemple de programme… revisité

```
  NAME    main

  PUBLIC  __iar_program_start

  SECTION .intvec : CODE (2)
  CODE32

__iar_program_start

  B       main

b DC32  0xAB
c DC32  0xF2
```

Table des vecteurs d'interruption!

Ici, nous ne supportons qu'une seule interruption, **laquelle?**

```
  SECTION .text : CODE (2)
  CODE32

main
  LDR R0, b
  LDR R1, c
  ADD R2, R0, R1
  LDR R3, =a
  STR R2, [R3]
  B main
```

Code principal en mémoire ROM

```
  SECTION  `.noinit`:DATA (2)
a DS32  1
```

Variables en mémoire RAM

```
  END
```

Table des vecteurs d'interruption

```
  NAME    main

  PUBLIC  __iar_program_start

  SECTION .intvec : CODE (2)
  CODE32

__iar_program_start

  B       main            ; Reset
  B       undefInterrupt  ; Instruction indéfinie
  B       softInterrupt   ; Interruption logicielle

  SECTION .text : CODE (2)
  CODE32

undefInterrupt
  ; routine de traitement de l'instruction indéfinie

softInterrupt
  ; routine de traitement de l'interruption logicielle

main
  ; routine de traitement de l'interruption reset
  ; (donc, notre code principal)

  END
```

Table des vecteurs d'interruption

Ici, nous supportons **3 interruptions**

Routine pour instruction indéfinie

Routine pour interruption logicielle

Code principal

Table des vecteurs d'interruption

- Contient une instruction (en ARM) qui branche vers la routine de traitement de l'interruption
- Commence à l'adresse 0x0 de la mémoire
- Habituellement en mémoire ROM (contenu décidé par le compilateur/programmeur)
  - peut être déplacée en RAM
  - peut être modifiée par le système d'exploitation.

Une interruption survient… que faire?

1. Terminer l'instruction en cours
2. Déterminer s'il faut traiter l'interruption.
3. Sauvegarder le **contexte**
4. Déterminer l'adresse de la routine de traitement de l'interruption
5. Exécuter cette routine

Le « contexte »

- Quelles sont les informations importantes au bon déroulement d'un programme?
  - PC (notre vieil ami)
    - est sauvegardé automatiquement dans LR
      - la valeur sauvegardée dans LR dépend du type d'interruption. Par exemple, dans une **FIQ**, on sauvegarde PC directement (donc l'adresse de l'instruction courante **+ 8**)
  - Les drapeaux de l'ALU
    - situés dans le « registre » spécial CPSR
    - est sauvegardé automatiquement dans un autre registre spécial: SPSR
  - Les registres
    - devront être sauvegardés à la mitaine, au besoin!

Accéder à CPSR et SPSR

- Instructions spécialisées: MRS et MSR
  - MRS copie la valeur de (C/S)PSR dans un registre

```
MRS R0, SPSR ; R0 <- SPSR
```

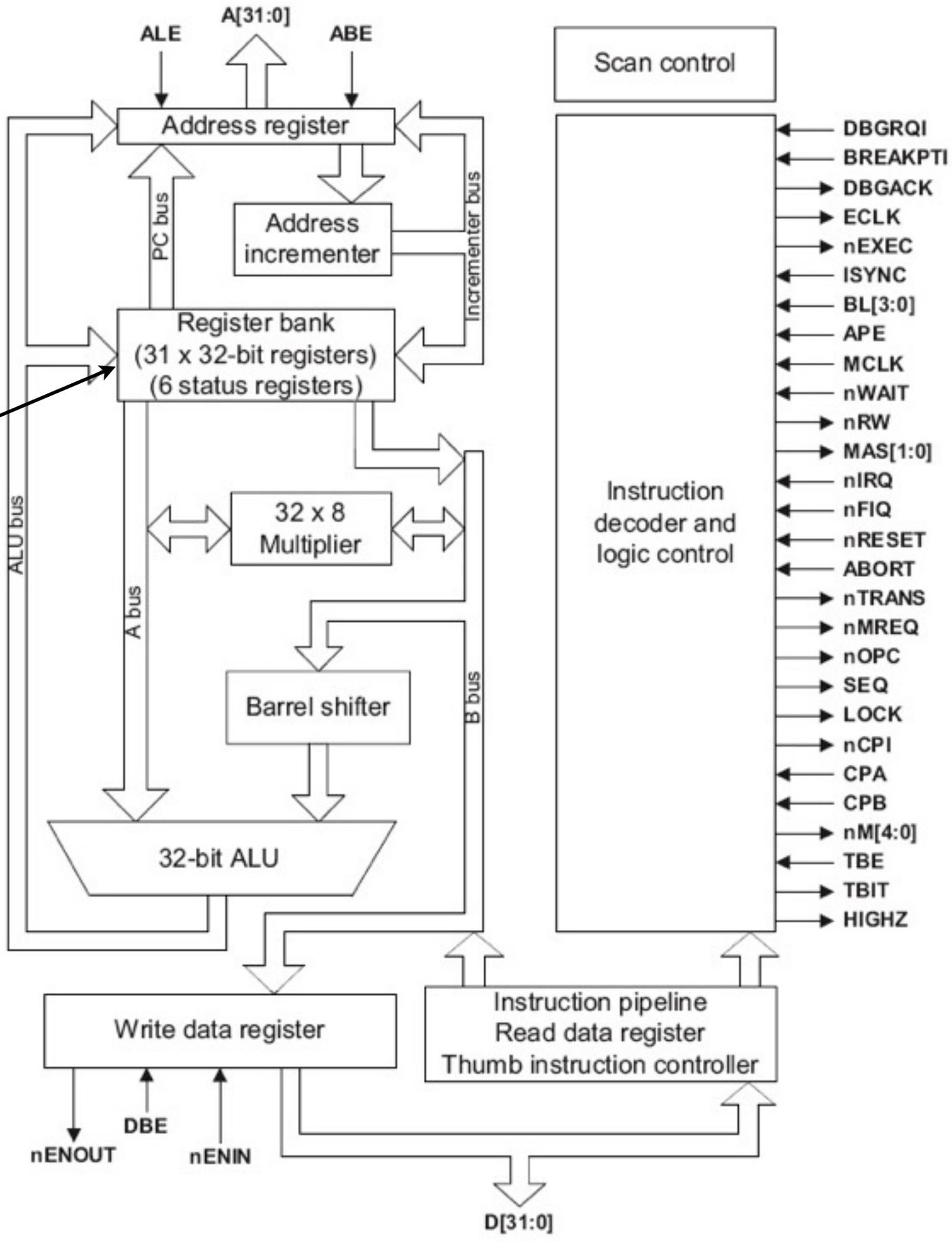
- MSR copie la valeur d'un registre dans (C/S)PSR

```
MSR CPSR, R0 ; CPSR <- R0
```

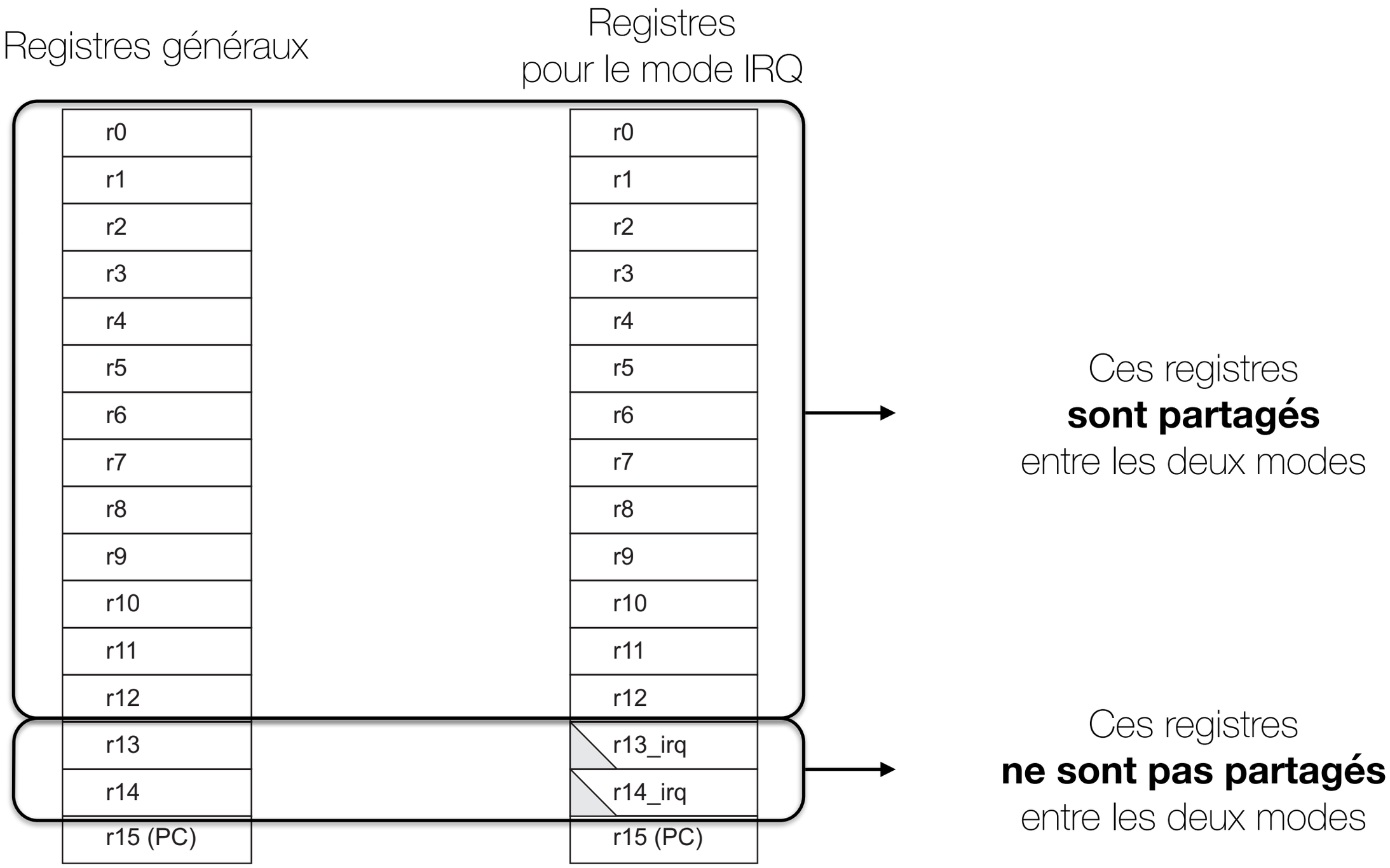
Rappel: les registres en ARM

- 16 registres de 32 bits sont disponibles:
  - `R0` à `R12`: usage général
  - `R13` à `R15`: registres “spéciaux”
    - `R13`: SP, « Stack Pointer »
    - `R14`: LR, « Link Register »
    - `R15`: PC, « Program Counter »
- Le “Current Program Status Register” (CPSR) est utilisé pour mémoriser les résultats d’opération

16 **accessibles**
« à la fois »



Registres ARM



Il y a donc au moins **2 registres physiques** de plus (R13_irq et R14_irq)!

Registres ARM

Registres généraux

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15 (PC)

Registres
pour le mode IRQ

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13_irq
r14_irq
r15 (PC)

Registres
pour le mode FIQ

r0
r1
r2
r3
r4
r5
r6
r7
r8_fiq
r9_fiq
r10_fiq
r11_fiq
r12_fiq
r13_fiq
r14_fiq
r15 (PC)

Il y a donc au moins **9 registres physiques** de plus
(R13_irq, R14_irq, R8_fiq, …, R14_fiq)!

Registres ARM

Pourquoi **Reset** n'est pas là?

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
r14	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

Il y a donc **15 registres physiques** de plus
(pour un total de 31)

Registres « spéciaux » ARM

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

Il y a donc **5 registres « spéciaux » physiques** de plus
(pour un total de 6)

Utilisation d'un registre

- Dans la routine de traitement de l'interruption, on peut simplement utiliser les registres comme d'habitude

- Par exemple:

```
fiqInterrupt
  ; routine de traitement de l'instruction fiq

  MOV R8, #1   ; on utilise R8_fiq

  MOV R0, #2   ; on utilise R0!
  ...
```

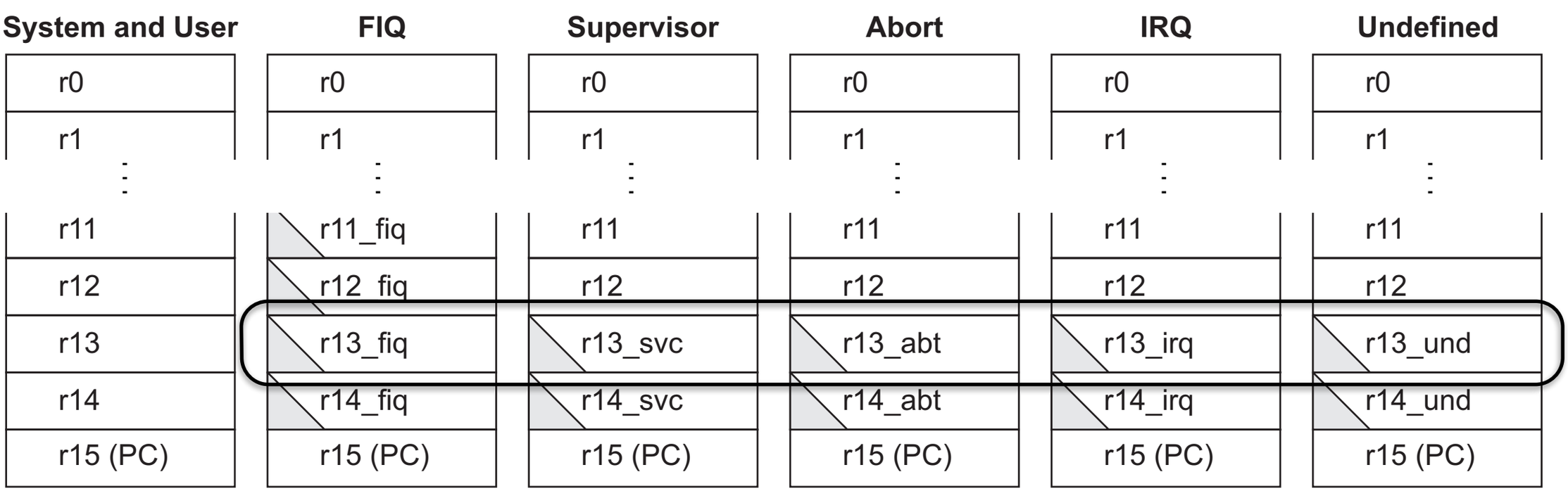
- Quand `R8` est utilisé dans une routine de traitement d'une interruption **FIQ**, c'est en fait `R8_fiq` qui est utilisé

- Cependant, quand `R0` est utilisé, c'est `R0`! Quel est le problème?

  - Si notre programme principal avait besoin de `R0`, on vient de changer sa valeur!

Pile d'interruptions

- Une pile spéciale est disponible pour chaque type d'interruption
  - Il faut l'avoir préparée au préalable!
- On peut donc sauvegarder les registres avec PUSH et POP à l'intérieur des routines de traitement de l'interruption



Une interruption survient… que faire?

1. Terminer l'instruction en cours
2. Déterminer **s'il faut traiter** l'interruption.
3. Sauvegarder le contexte
4. Déterminer l'adresse de la routine de traitement de l'interruption
5. Exécuter cette routine

Priorités

Priorité	Interruption
Plus haute	Reset
	« Data abort »
	FIQ
	IRQ
	« Prefetch abort »
Plus basse	Interruption logicielle et instruction indéfinie (ne peuvent pas survenir en même temps… pourquoi?)

Priorités

- Les interruptions ont des priorités: une interruption de haute priorité peut interrompre une interruption ayant un niveau de priorité plus bas.
- Certaines interruptions peuvent survenir n'importe quand, même pendant une autre interruption.
- Certaines interruptions, comme reset, ont une priorité (maximale pour reset) qui ne peut pas être changée.

Interruptions imbriquées

- Qu'arrive-t-il si une interruption survient lorsqu'on traite une interruption?
- Cela dépend de la priorité
  - Si la priorité de la nouvelle interruption est plus élevée:
    - On interrompt l'exécution et on traite cette nouvelle interruption
  - Si la priorité de la nouvelle interruption est moins élevée:
    - On attend que le traitement de l'interruption à plus haute priorité soit terminé, et on traite cette nouvelle interruption par la suite
- Sauvegarder les adresses de retour et certains registres sur la pile permet d'imbriquer les interruptions comme on imbrique des fonctions

Une interruption se termine… que faire?

1. Restaurer le contexte

2. Reprendre là ou le processeur était rendu

Restauration du « contexte »

- Quelles sont les informations importantes au bon déroulement d'un programme?
  - PC (notre vieil ami)
    - doit être restauré à partir de la valeur dans LR
      - rappel: la valeur sauvegardée dans LR dépend du type d'interruption. Par exemple, dans une **FIQ**, on sauvegarde PC directement (donc l'adresse de l'instruction courante **+ 8**)
  - Les drapeaux de l'ALU
    - sont situés dans le « registre » spécial **S**PSR
    - est restauré automatiquement dans CPSR
  - Les registres
    - devront être restaurés à la mitaine, au besoin!

Reprise du programme

- Dans plusieurs architectures, une instruction spéciale est utilisée pour indiquer la fin d'une interruption
- En ARM, c'est, comment dire, un peu bizarre…
  - il faut: utiliser une instruction avec **S** (change les drapeaux), qui stocke son résultat dans **PC**(!)

```
fiqInterrupt
  ; routine de traitement de l'instruction FIQ

  ...

  ; terminé!
  SUBS PC, LR, #4
```

Résumé

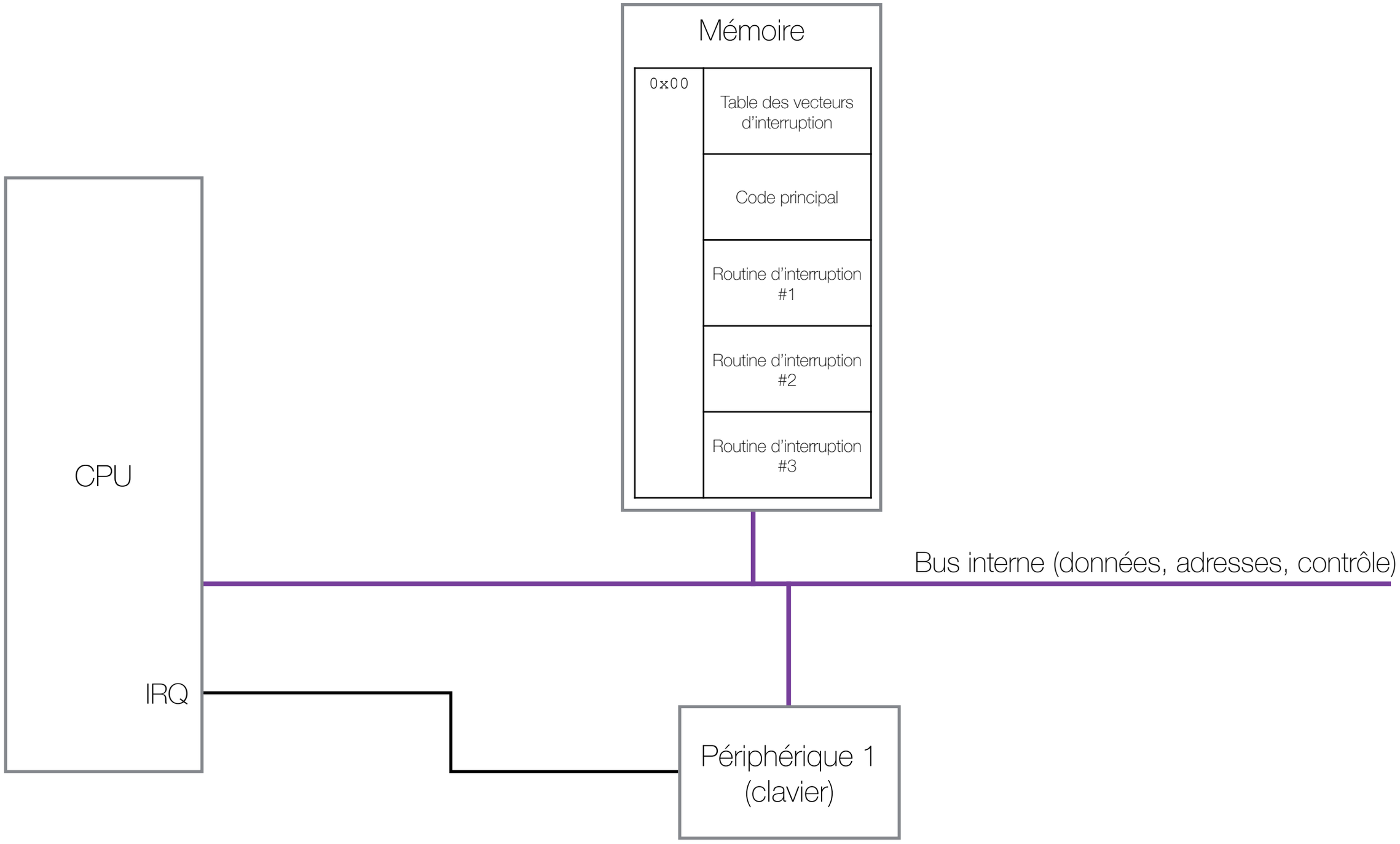
Interruption!

1. Terminer l'instruction en cours
2. Déterminer s'il faut traiter l'interruption.
3. Sauvegarder le contexte
4. Déterminer l'adresse de la routine de traitement de l'interruption
5. Exécuter cette routine

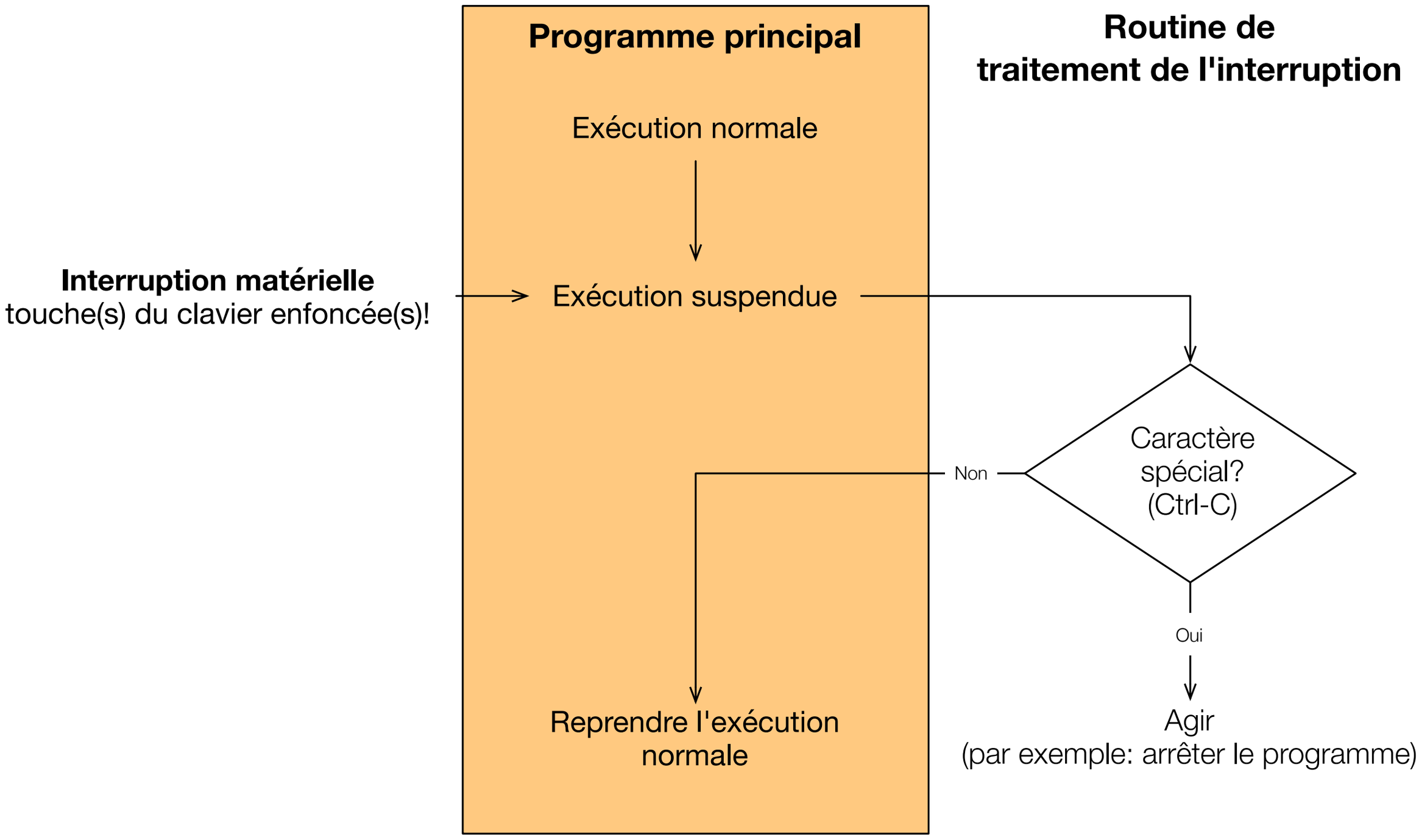
Traitement de l'interruption…

1. Restaurer le contexte
2. Reprendre là ou le processeur était rendu

Interruptions matérielles (périphériques)

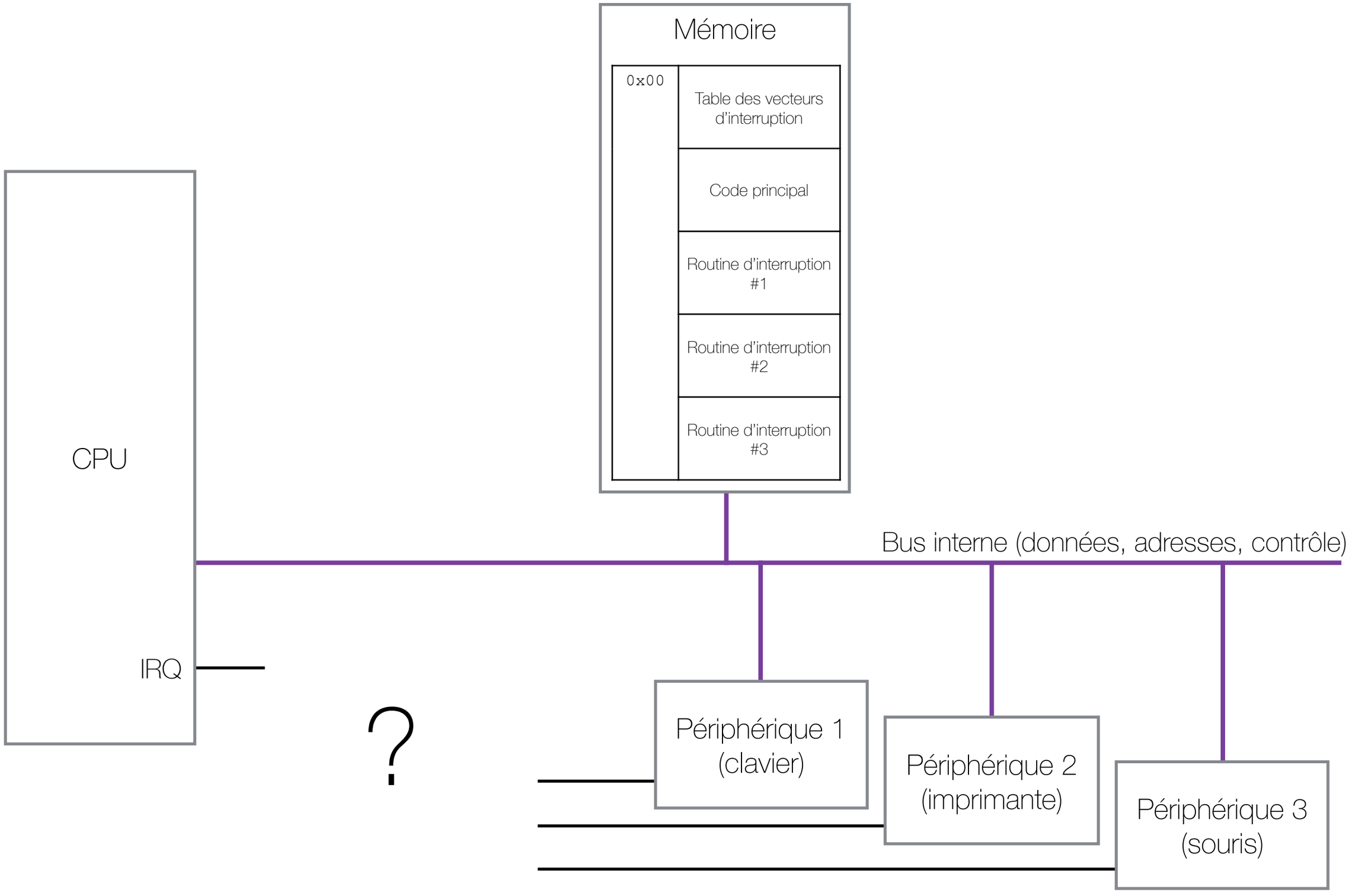


Interruption clavier

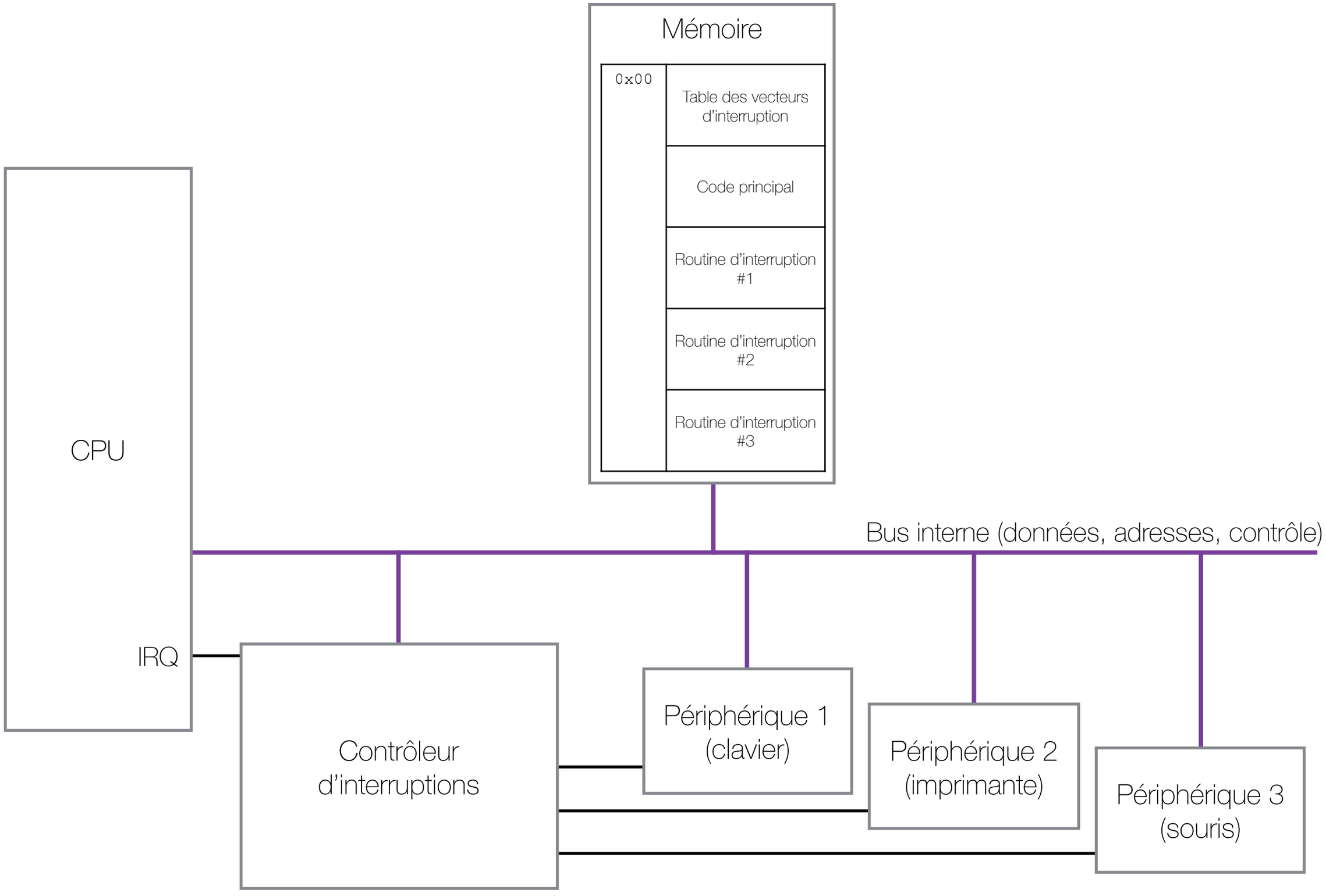


Inspiré de: fig. 9.6 Englander

Interruptions matérielles (périphériques)



Contrôleur d'interruptions



Contrôleur d'interruptions (NVIC)

- Les interruptions de l'ordinateur sont gérées par le contrôleur d'interruption.
- Le contrôleur d'interruptions:
  - reçoit les signaux d'interruptions
  - peut activer (masquer) ou désactiver certaines interruptions.
  - modifier la priorité des interruptions.
  - signale les interruptions au microprocesseur à l'aide de fils dédiés à cette fin.
  - peut être configuré via des instructions dans la mémoire
- Dans le cas du processeur ARM, le contrôleur d'interruptions est inclus dans le cœur.

Types d'interruptions

- **Système**: reset, faute matérielle générale, etc.
- **Exception**: le processeur peut générer des interruptions s'il n'est pas capable de lire ou d'exécuter une instruction (opcode invalide, division par 0, mémoire protégée, etc).
- **Matérielles**: générées par les périphériques
- **Logicielles**: il y a une instruction qui permet de générer une interruption dans tous les jeux d'instructions.
  - En ARM, c'est l'instruction SWI (Software Interrupt)

Interruptions du système

- L'interruption **reset** est l'interruption système la plus commune. Cette interruption peut survenir pour plusieurs raisons:
  - mise sous tension, activation de la broche reset du microprocesseur, instruction reset, etc.
- Lors d'un reset, toutes les autres interruptions sont ignorées

Exceptions

- Les **exceptions** surviennent quand un évènement logiciel spécial arrive. Par exemple:
  - instruction invalide
  - division par 0
  - référence à une adresse invalide
  - accès invalide à une adresse protégée
- Les exceptions ont un très haut niveau de priorité parce le microprocesseur est dans une impasse: il ne peut exécuter l'instruction en cours en raison d'une erreur de programmation!

Interruptions matérielles

- Les interruptions **matérielles** sont générées par les périphériques
- La plupart des périphériques ont une ligne de contrôle reliée au contrôleur d'interruptions qui leur permet de signaler un événement.
- Lors d'une interruption de périphérique, le microprocesseur obtient automatiquement le # de l'interruption du contrôleur et utilise ce numéro pour trouver l'ISR à exécuter à partir de la table des vecteurs d'interruptions.

Interruptions logicielles

- Les interruptions **logicielles** sont des interruptions « provoquées » par le programmeur. Le programmeur utilise une instruction qui déclenche une interruption.

- Les interruptions logicielles ont un effet similaire à un appel de fonction avec une différence fondamentale:
  - l'adresse de la fonction appelée est dans la table des vecteurs d'interruption plutôt qu'être une adresse relative au programme.

- Les interruptions logicielles servent souvent à appeler des fonctions du système d'exploitation dont l'adresse est inconnue du programmeur, mais gérée par le système d'exploitation (grâce à la table des vecteurs d'interruption).

Interruptions et système d'exploitation

- 2 utilités principales:
  - accès aux périphériques
  - exécution de plusieurs processus

Interruptions et système d'exploitation

- Accès aux périphériques
    - Chaque périphérique (ex: clavier, imprimante) peut se comporter légèrement différemment des autres
- C'est le système d'exploitation ("Operating System", ou OS) qui rend les programmes "indépendants" du matériel. Comment?
    - C'est l'OS qui modifie les ISR à exécuter en fonction du matériel branché dans l'ordinateur (via la table des vecteurs d'interruption)
- Les programmes peuvent donc utiliser la table des vecteurs d'interruption comme d'habitude

Accès aux périphériques

- Exemple: recevoir une valeur du clavier
  - interruption matérielle (c'est l'utilisateur qui tape au clavier)
- Exemple: envoyer des données à l'imprimante
  - interruption logicielle (c'est notre programme qui veut imprimer)
- Le système d'exploitation permet au même programme de "parler" à plusieurs modèles de claviers et d'imprimantes

Interruptions et système d'exploitation

- Les interruptions permettent l'exécution de plusieurs processus
- Comment?
  - Une horloge génère des interruptions périodiquement
  - À chaque interruption, on change le processus à exécuter

Ce qui nous amène au… TP4!

http://vision.gel.ulaval.ca/~jflalonde/cours/1001/h16/tps/tp4/index.html