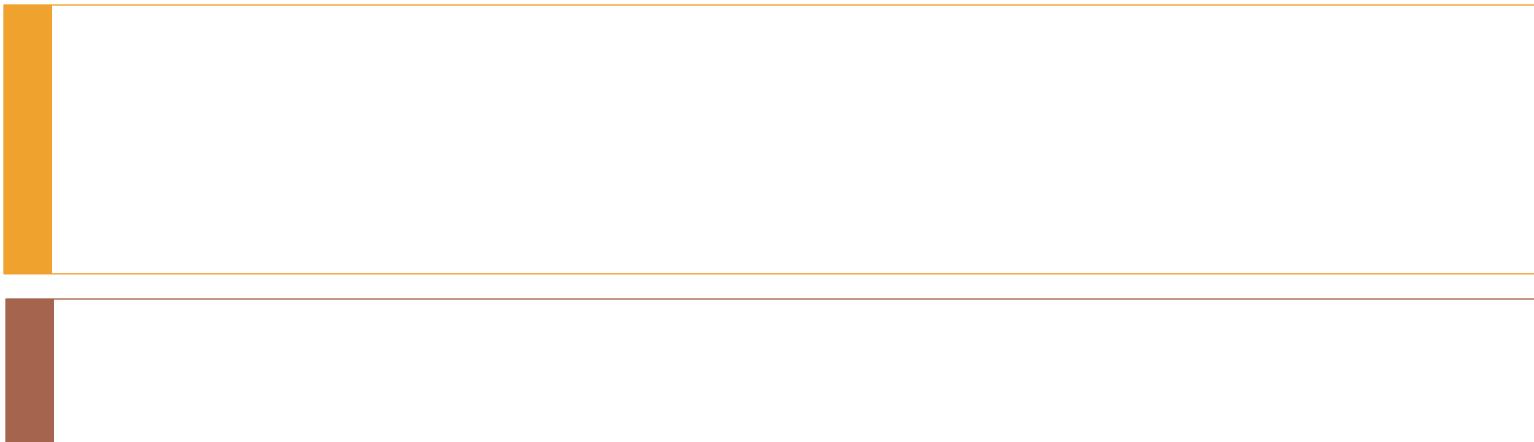


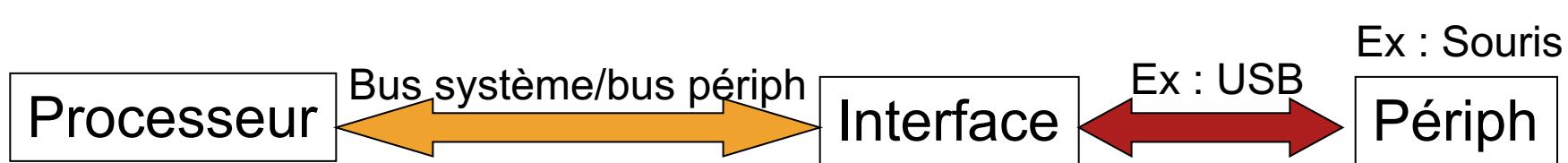
Les entrées/sorties par test d'état

Le circuit d'interface parallèle



Pourquoi faut-il des interfaces ?

- ▶ Variété des signaux fournis par les périphériques
 - ▶ numérique
 - ▶ analogique
 - ▶ un seul bit, ...
- ▶ Le processeur lui-même ne peut incorporer toutes les interfaces nécessaires
- ▶ Le processeur fournit un standard de communication (le bus système)
- ▶ Circuits d'interface
 - ▶ d'un côté, conformes à ce standard et contrôlés via le bus système
 - ▶ de l'autre côté, conformes au standard de certains périphériques



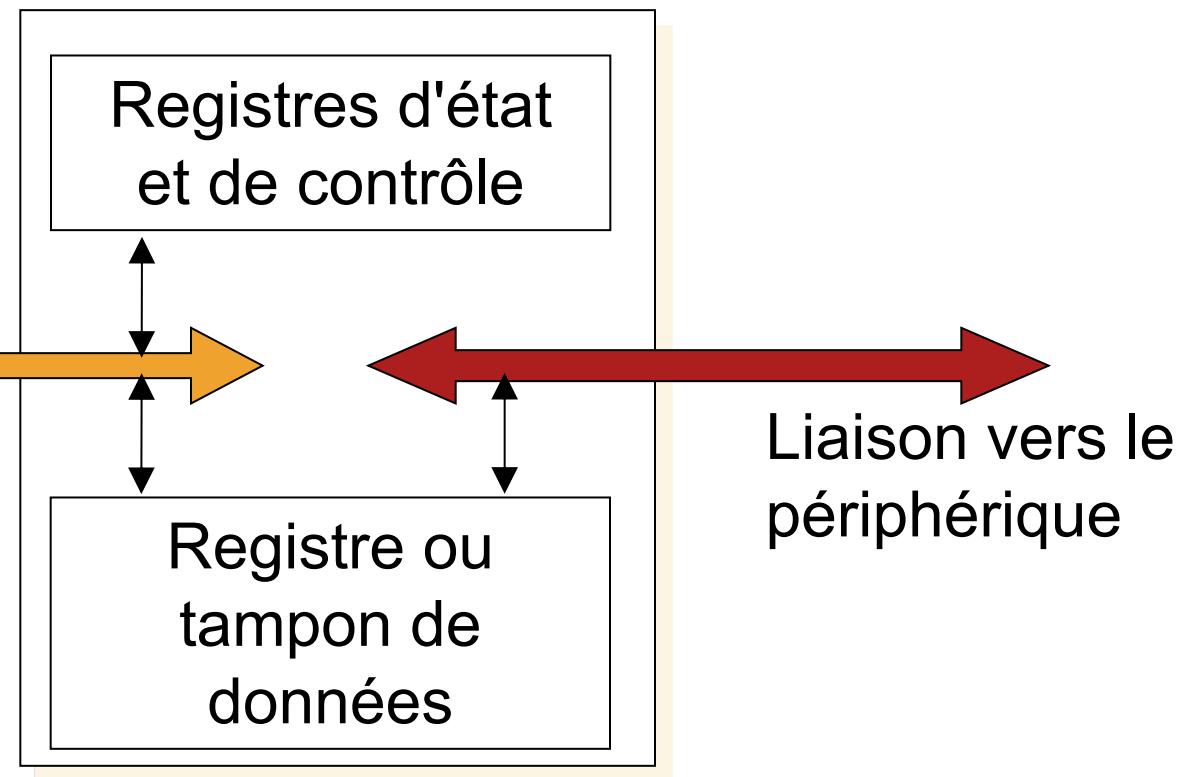
De plus en plus de standards

- ▶ USB, Firewire (IEEE 1394), SCSI, SATA, Ethernet,...
- ▶ Mais surtout pour des périphériques de type "informatique" ou plutôt "numérique"

Structure générale d'un circuit d'interface

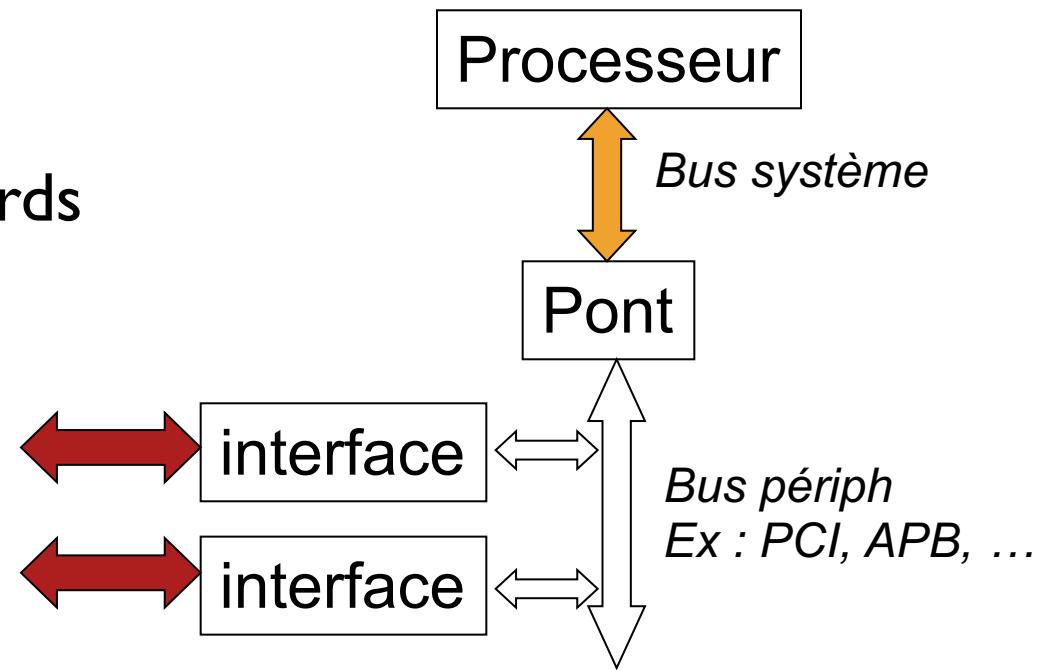
Liaison vers le processeur (bus)

*Ordre de contrôle
Situations d'état
Données*



Un bus séparé pour les interfaces

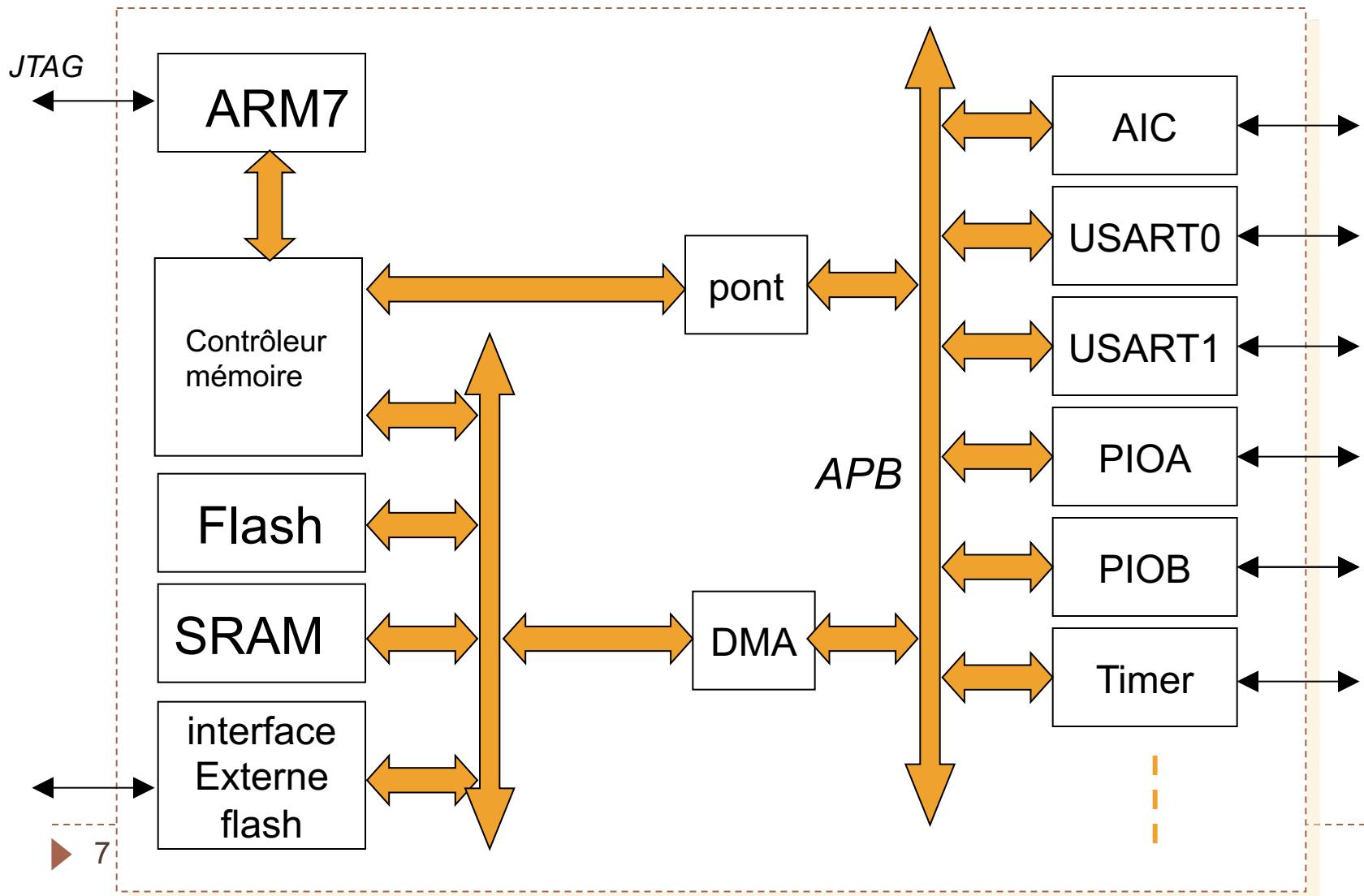
- ▶ Car nécessitent peu de bande passante par rapport au processeur
- ▶ Périphériques lents par rapport au bus système
 - ▶ lien série : 56 kbit/s
 - ▶ ADSL : 20 Mbit/s
 - ▶ processeur : 400 Mbit/s
- ▶ Bus d'interfaces standards
 - ▶ PCI
 - ▶ PCI express



Intégration des interfaces

- ▶ System-on-chip
- ▶ Exemple : la série AT91SAM7X d'ATMEL
 - ▶ processeur ARM7TDMI
 - ▶ Mémoire Flash et mémoire SRAM
 - ▶ 2 USART (liens série)
 - ▶ 2 ports //
 - ▶ 3 timers
 - ▶ USB
 - ▶ Ethernet
 - ▶ ...

Exemple de SOC : l'AT91SAM7X



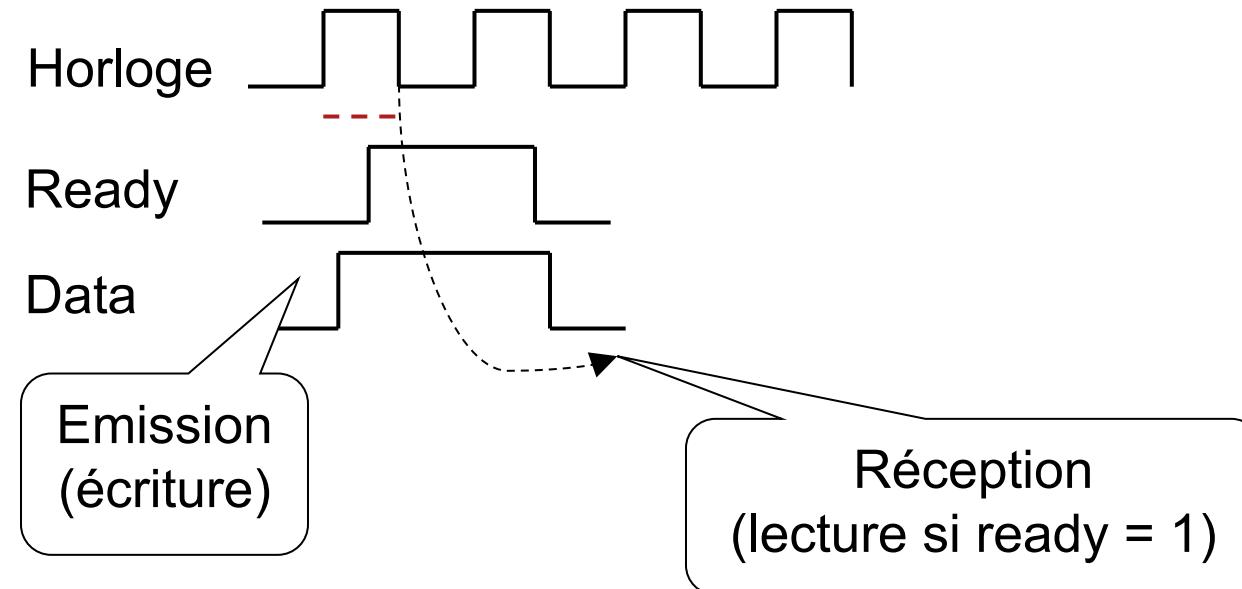
Partage de broches (multiplexage)

- ▶ Coût d'un circuit intégré → nombre de broches
- ▶ Partage de broches entre diverses interfaces
 - ▶ Moins de possibilités qu'il n'y paraît...
- ▶ Exemple :
 - ▶ broches partagées entre PIO et timer
 - ▶ PIO et AIC
 - ▶ PIO et USART
- ▶ Programmer le PIO pour préciser à quoi est connectée chaque broche multiplexée
 - ▶ aiguillage vers PIO ou autre interface

Communication synchrone et asynchrone

► Synchrone

- ▶ même horloge,
- ▶ connaissance de l'autre (temps de lecture)



Communication synchrone et asynchrone

► Isochrone

- ▶ Deux horloges de même fréquence mais pas de même phase
 - ▶ Mécanisme matériel pour recouvrir la phase

► Plésiochrone

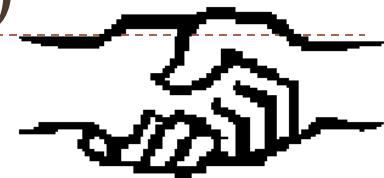
→ Protocole de communication

- Nécessité de deux horloges à peu près de même fréquence
- Synchronisation grâce à un protocole d'échange particulier

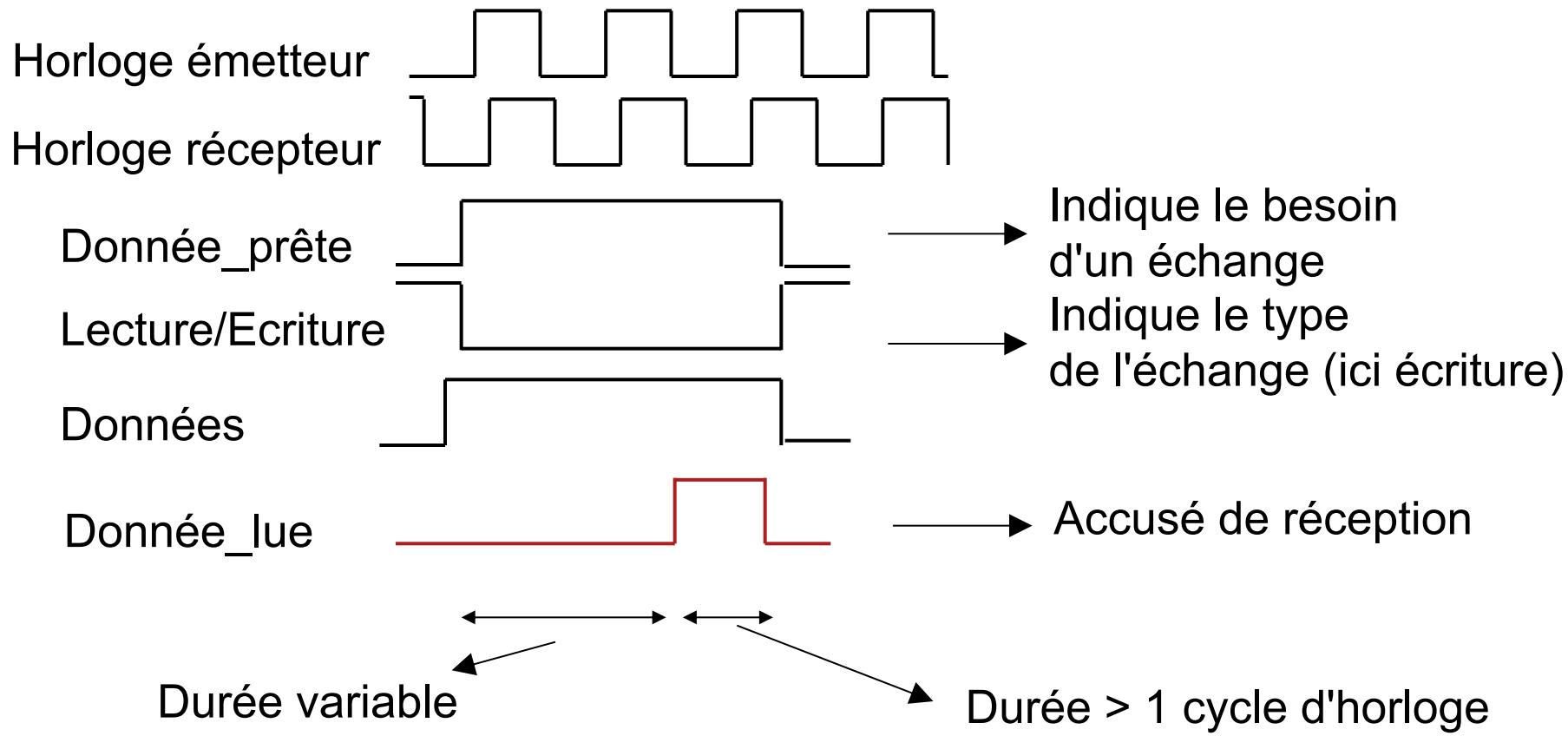
► Asynchrone

- ▶ Synchronisation sur le signal transmis
 - récupération d'horloge sur un en-tête prévu dans le protocole
- ▶ évite de transmettre l'horloge
- ▶ Utilisé surtout pour les longues distances (câble moins cher)

La poignée de main (envoi d'info)



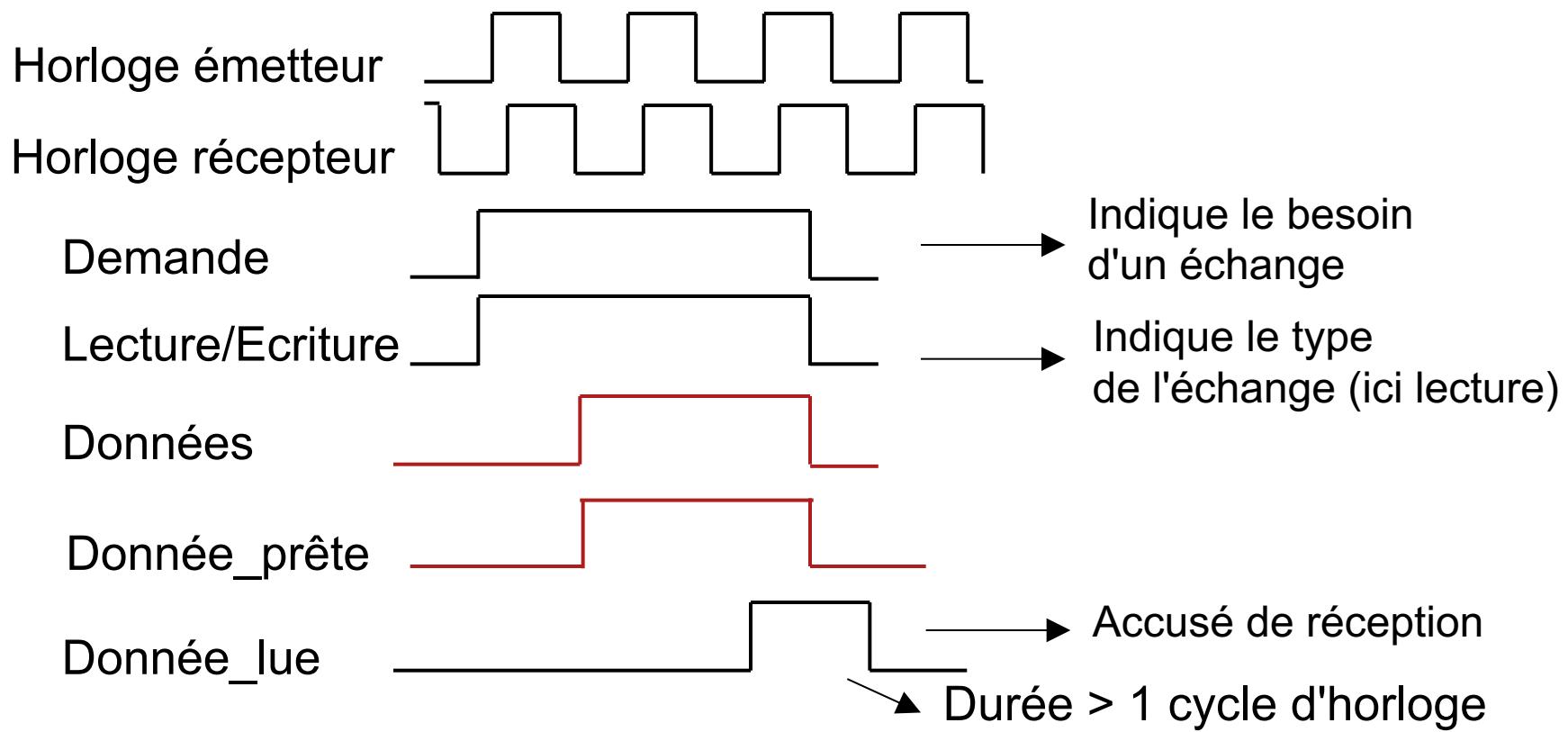
Exemple : envoi d'une donnée à un périphérique



La poignée de main (demande d'info)



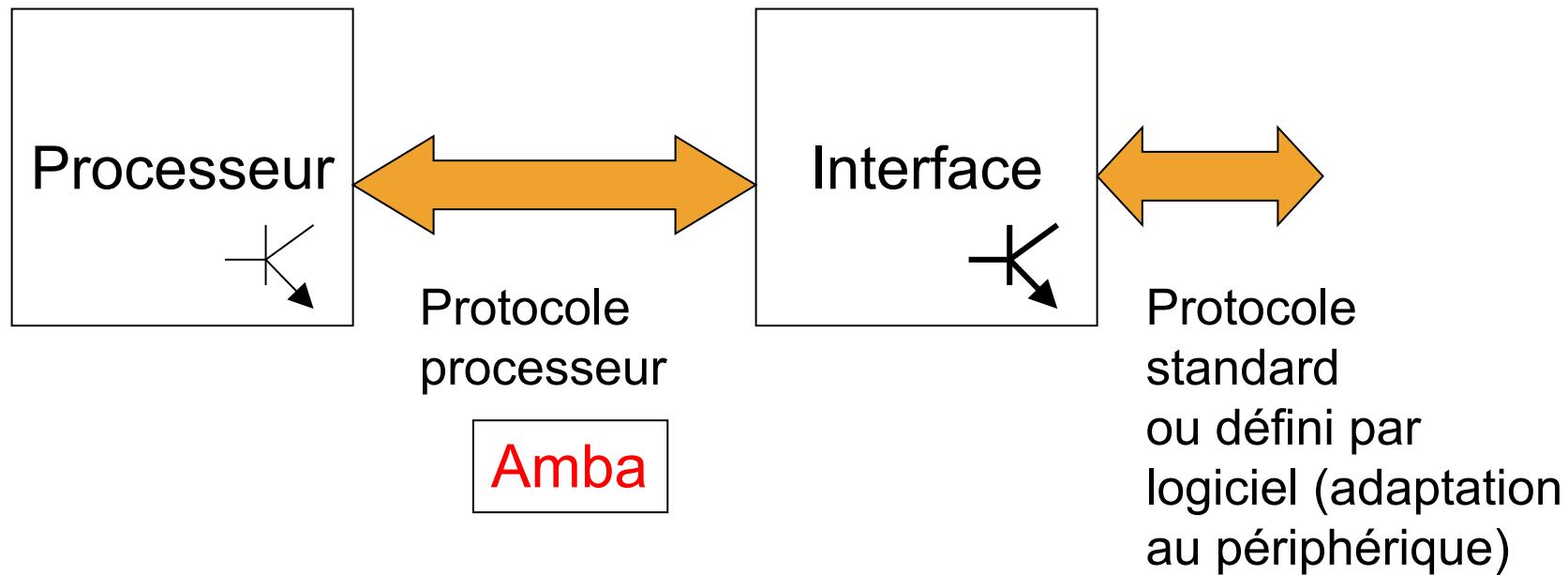
Exemple : demande d'une information à un périphérique



Circuit d'interface //

► Besoin

- ▶ Bus du processeur // mais protocole spécifique
- ▶ Capacité en courant



Le PIO (Parallel Input/Output)

- ▶ **32 lignes bidirectionnelles de 1 bit programmables indépendamment**
 - ▶ Certaines lignes peuvent être connectées en interne soit au PIO soit à un autre périphérique (économie)
- ▶ **Activation des valeurs sur ces lignes par logiciel**
 - ▶ Mise en œuvre de n'importe quel protocole
- ▶ **Contrôle du circuit d'interface et lecture de son état**
 - ▶ Adresse de base
 - ▶ Exemple : 0x FFFF F600 pour PIOB
 - ▶ Contrôle : registres en écriture seulement
 - ▶ Exemple : PIO_SODR
 - ▶ Etat : registres en lecture seulement
 - ▶ Exemple PIO_PDSR

Le PIO : les registres de contrôle

▶ Pour chaque ligne i, 4 paramètres

▶ Broche du PIO ou d'un autre périphérique ?

▶ PIO : écrire **1** dans **PIO_PER[i]**

PIO Enable Register

▶ Autre : écrire **1** dans **PIO_PDR[i]**

PIO Disable Register

▶ Sens ?

▶ Sortie : écrire **1** dans **PIO_OER[i]**

PIO Output Enable Register

▶ Entrée : écrire **1** dans **PIO_ODR[i]**

PIO Output Disable Register

▶ Valeur ?

▶ 1 : écrire **1** dans **PIO_SODR[i]**

PIO Set Output Data Register

▶ 0 : écrire **1** dans **PIO_CODR[i]**

PIO Clear Output Data Register

▶ Quel autre périphérique ?

▶ A : écrire **1** dans **PIO_ASR[i]**

Ecrire 0 dans n'importe quel bit
de n'importe quel registre n'a aucun effet.

▶ B : écrire **1** dans **PIO_BSR[i]**

Le PIO : les registres d'état

- ▶ Pour chaque ligne i, 3 informations :
 - ▶ Le port i est-il connecté au PIO ou à un autre périphérique ?
 - ▶ $\text{PIO_PSR}[i]$
 - ▶ Quel autre périphérique ?
 - ▶ $\text{PIO_ABSR}[i]$
 - ▶ Sens ?
 - ▶ $\text{PIO_OSR}[i]$
 - ▶ Valeur ?
 - ▶ $\text{PIO_PDSR}[i]$
 - ▶ $\text{PIO_ODSR}[i]$ si le port i est une sortie

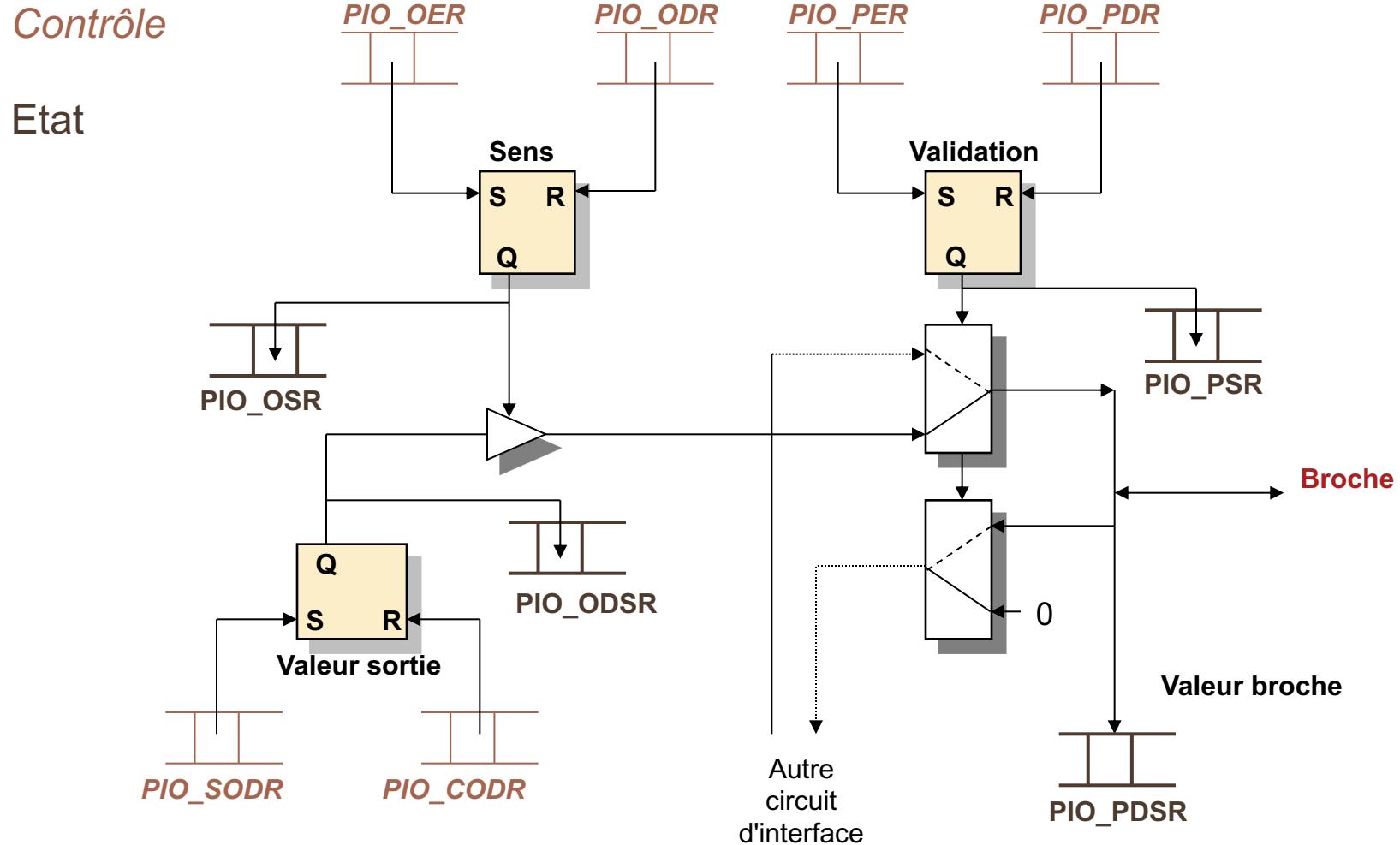
PIO Pin Status Register

PIO Output Status Register

PIO Pin Data Status Register

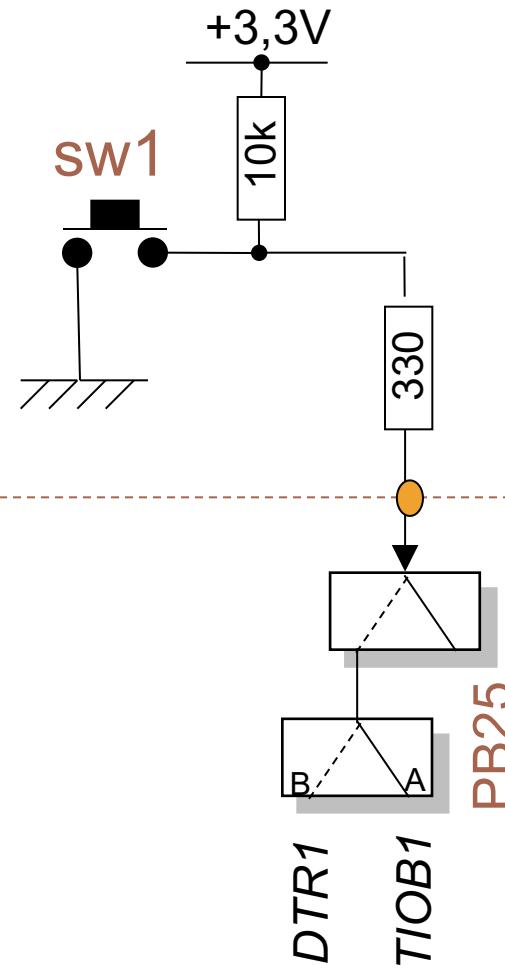
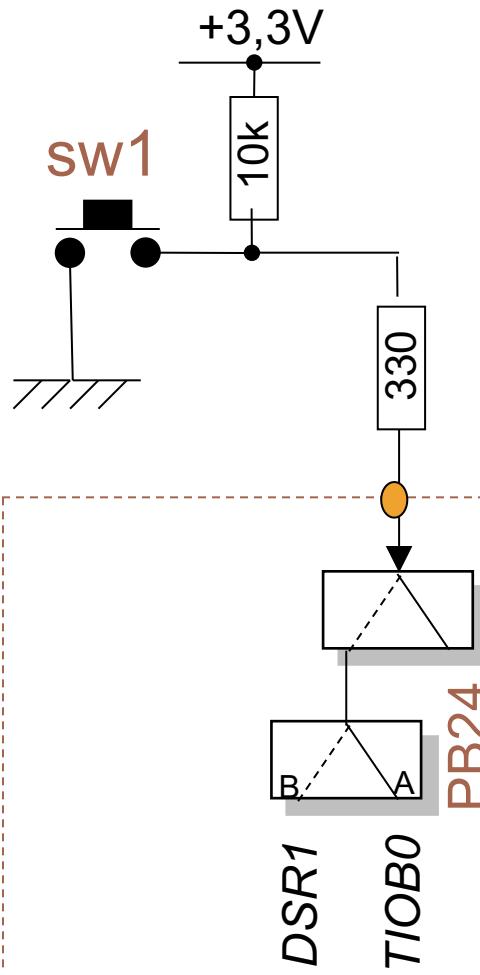
PIO Output Data Status Register

Le PIO : les registres



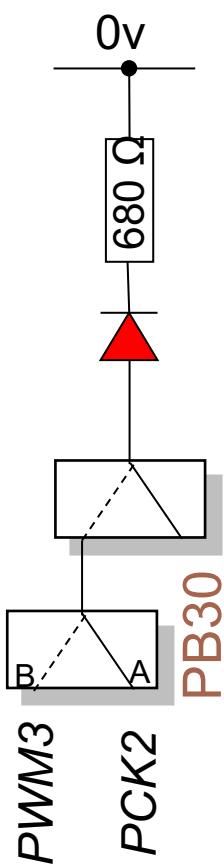
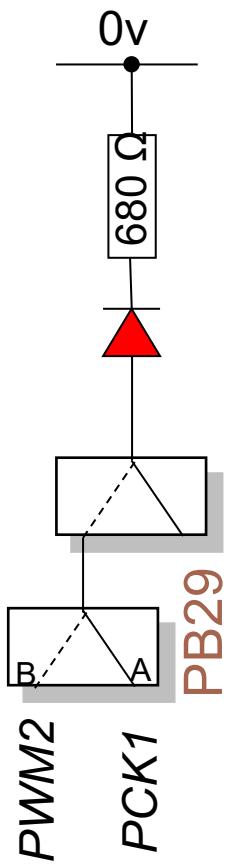
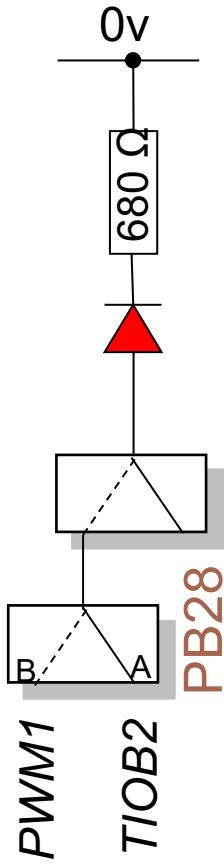
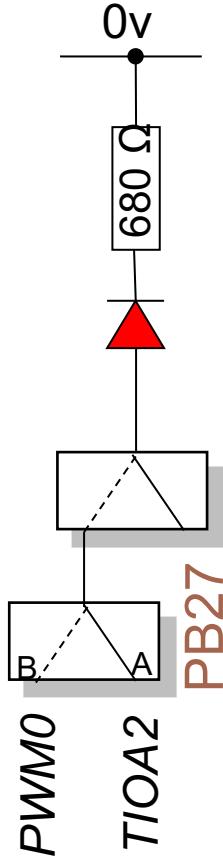
Connexions du PIOB en TP

Entrées



Connexions du PIOB en TP

Sorties



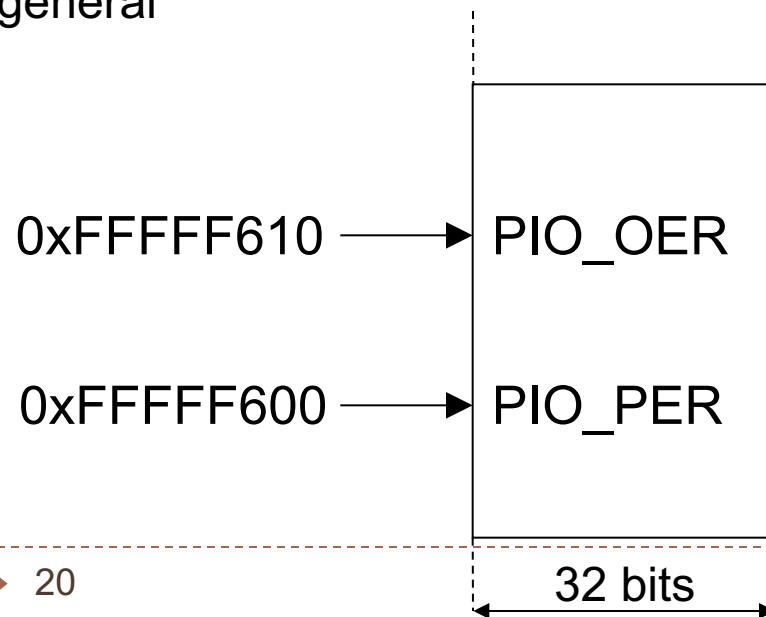
Exemple : mettre à un la ligne 4

une seule fois

```

ldr    r12,=PIOB_BASE      @0x FFFF F600
       mov    r0,#0b10000      @ ligne 4
       { str   r0, [r12,#PIO_PER]  @ affectée au PIO
         str   r0, [r12,#PIO_OER]  @ en sortie
       }
       str   r0, [r12,#PIO_SODR] @ à 1
    
```

en général



```

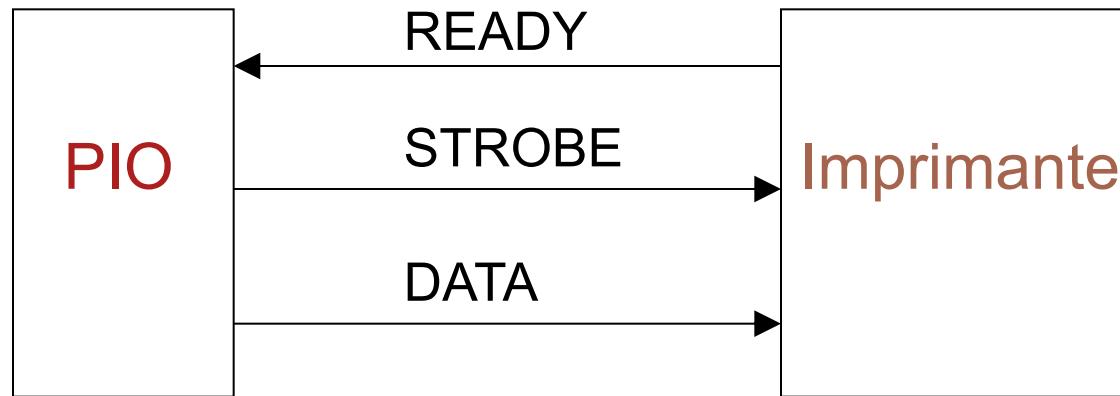
.equ  PIOB_BASE, 0xFFFFF600
.equ  PIO_PER, 0
.equ  PIO_OER, 0x10
.equ  PIO_SODR, 0x30
    
```

Exemple

```
ldr r12,=PIOB_BASE
ldr r0,=0x0000FFFF
str r0,[r12,#PIO_PER]          @ 00000000000000001111111111111111
                               @ lignes 0 à 15 pour le PIO
ldr r0,=0xF0F0F0F
str r0,[r12,#PIO_OER]          @ 00001111000011110000111100001111
                               @ lignes 0-3 et 8-11 en sortie
ldr r0,=0x55555555
str r0,[r12,#PIO_SODR]         @ 010101010101010101010101010101010101
                               @ lignes 0,2,8,10 à 1
ldr r0,=0b00001111
str r0,[r12,#PIO_CODR]         @ 000000000000000000000000000000001111
                               @ 0,2 passent à 0 - 8,10 restent à 1
```

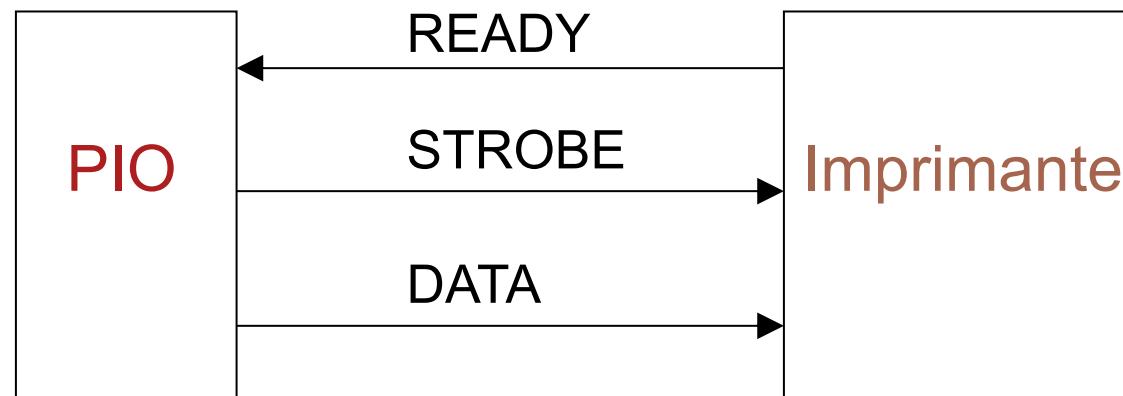
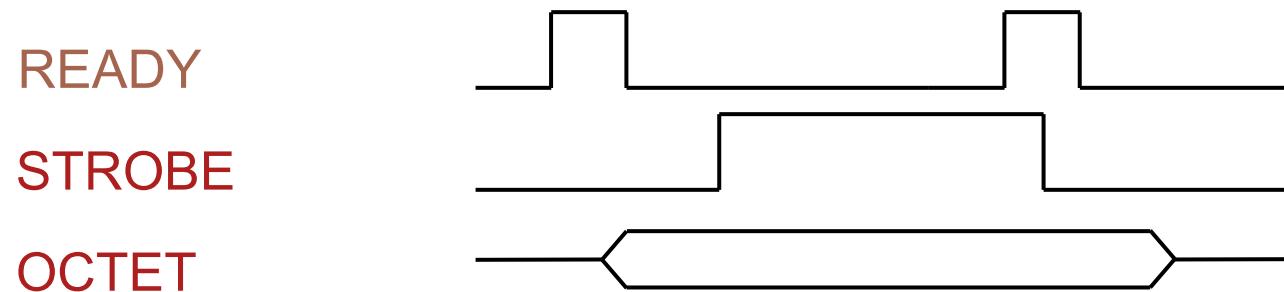
Exemple : imprimante à port //

- ▶ L'imprimante est esclave et indique qu'elle est prête grâce à une impulsion sur un signal READY
- ▶ Lorsqu'un signal STROBE est activé (front montant), elle lit un caractère sur les fils de données



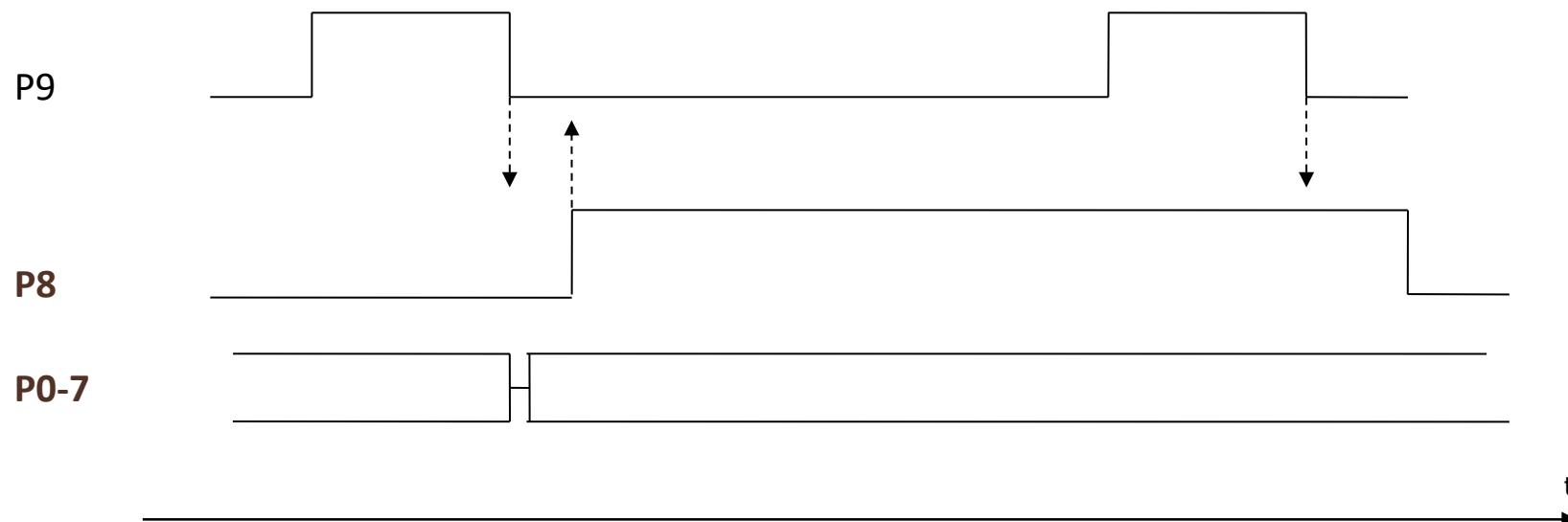
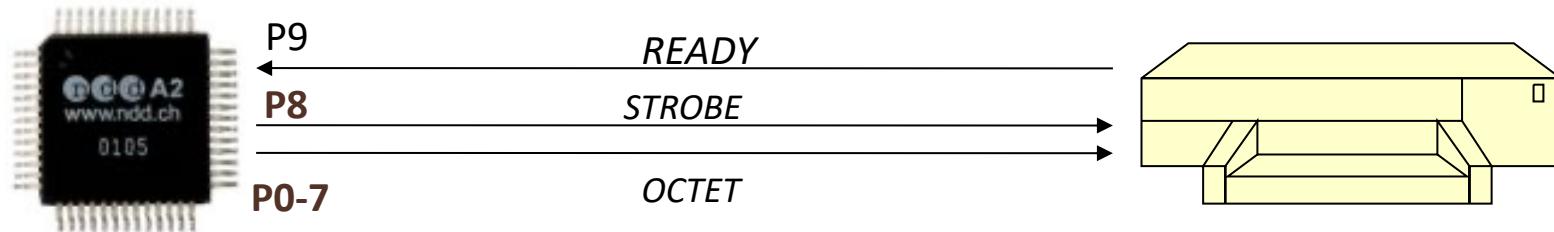
Exemple : imprimante à port //

► Chronogramme d'un échange



Exemple : imprimante à port //

► Chronogramme d'un échange



Algorithme du gestionnaire

```
initialiser le port ;  
STROBE ← 0 ;  
ptr ← adresse de début du tableau ;  
tant que READY != 1 faire  
    attendre ;  
tant que READY != 0 faire  
    attendre ;  
tant que ptr != adresse de fin faire  
    début  
        placer mem8[ptr] en sortie ;  
        STROBE ← 1 ;  
        tant que READY != 1 faire  
            attendre ;  
        STROBE ← 0 ;  
        tant que READY != 0 faire  
            attendre ;  
  
        ptr ← ptr + 1 ;  
fin
```

Définitions

```
.equ      PIOB_BASE, 0xFFFFF600
.equ      PIO_PER, 0x00
.equ      PIO_PDR, 0x04
.equ      PIO_PSR, 0x08
.equ      PIO_OER, 0x10
.equ      PIO_ODR, 0x14
.equ      PIO_OSR, 0x18
.equ      PIO_SODR, 0x30
.equ      PIO_CODR, 0x34
.equ      PIO_ODSR, 0x38
.equ      PIO_PDSR, 0x3C
.equ      OCTET, 0xFF
.equ      STROBE, 0x100
.equ      READY, 0x200
.equ      LIGNES, 0x3FF
.equ      SORTIES, 0x1FF
```



carte.inc

propre à
l'application

Programme de gestion par test d'état

```

main:    ldr      r12,=PIO_BASE           boucle:   cmp      r10,r11
          mov      r1,#LIGNES            bcs      exit
          str      r1,[r12,#PIO_PER]
          mov      r1,#SORTIES
          str      r1,[r12,#PIO_OER]
          mov      r1,#READY
          str      r1,[r12,#PIO_ODR]
          mov      r2,#STROBE
          str      r2,[r12,#PIO_CODR]

att1:    ldr      r1,[r12,#PIO_PDSR]       att3:    ldr      r1,[r12,#PIO_PDSR]
          ands   r1,r1,#READY          ands   r1,r1,#READY
          beq    att1                beq    att3
          str      r2,[r12,PIO_CODR]

att2:    ldr      r1,[r12,#PIO_PDSR]       att4:    ldr      r1,[r12,#PIO_PDSR]
          ands   r1,r1,#READY          ands   r1,r1,#READY
          bne    att2                bne    att4
          adr      r10,buffer
          adr      r11,finbuffer

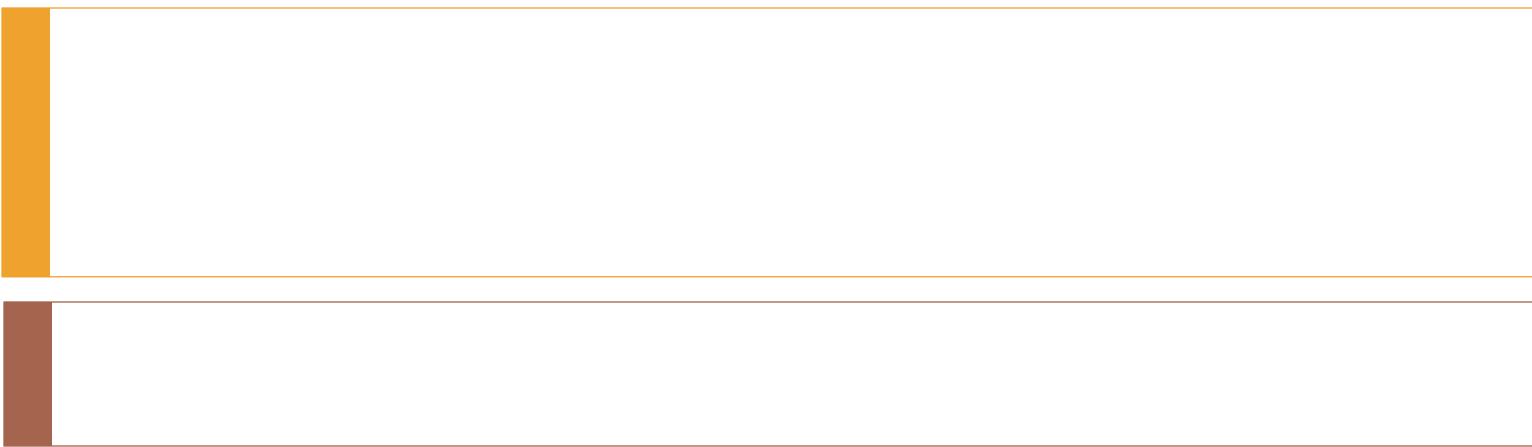
          exit:   nop
          buffer: .byte 1,3,7,15,31,63,127
          finbuffer:

```

Filtrage d'impulsions parasites (glitch)

- ▶ Si impulsion de longueur < $\frac{1}{2}$ cycle d'horloge
 - ▶ Possibilité de la filtrer
- ▶ Registres
 - ▶ Contrôle
 - ▶ PIO_IFER : I = filtre actif
 - ▶ PIO_IFDR : I = filtre inactif
 - ▶ Etat
 - ▶ PIO_IFSR : état filtre

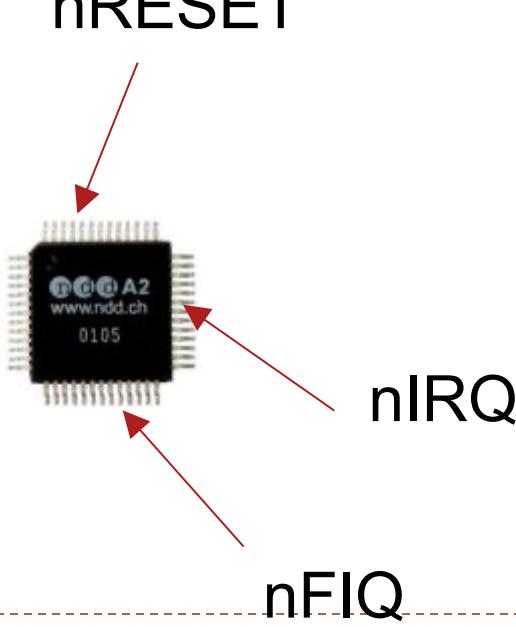
Les interruptions



Sources des exceptions

Interruptions

Exceptions logicielles



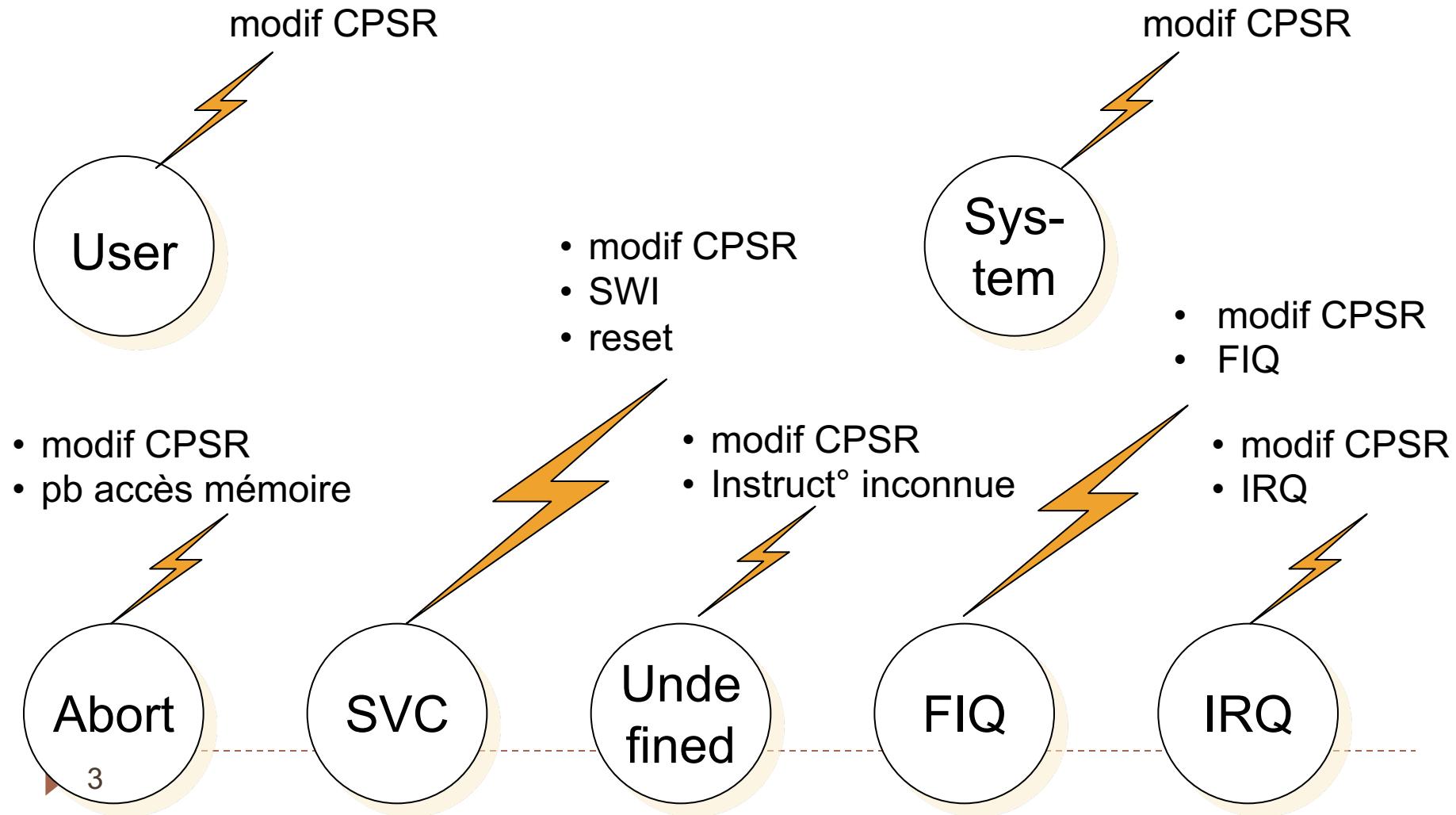
nRESET

Instruction SWI

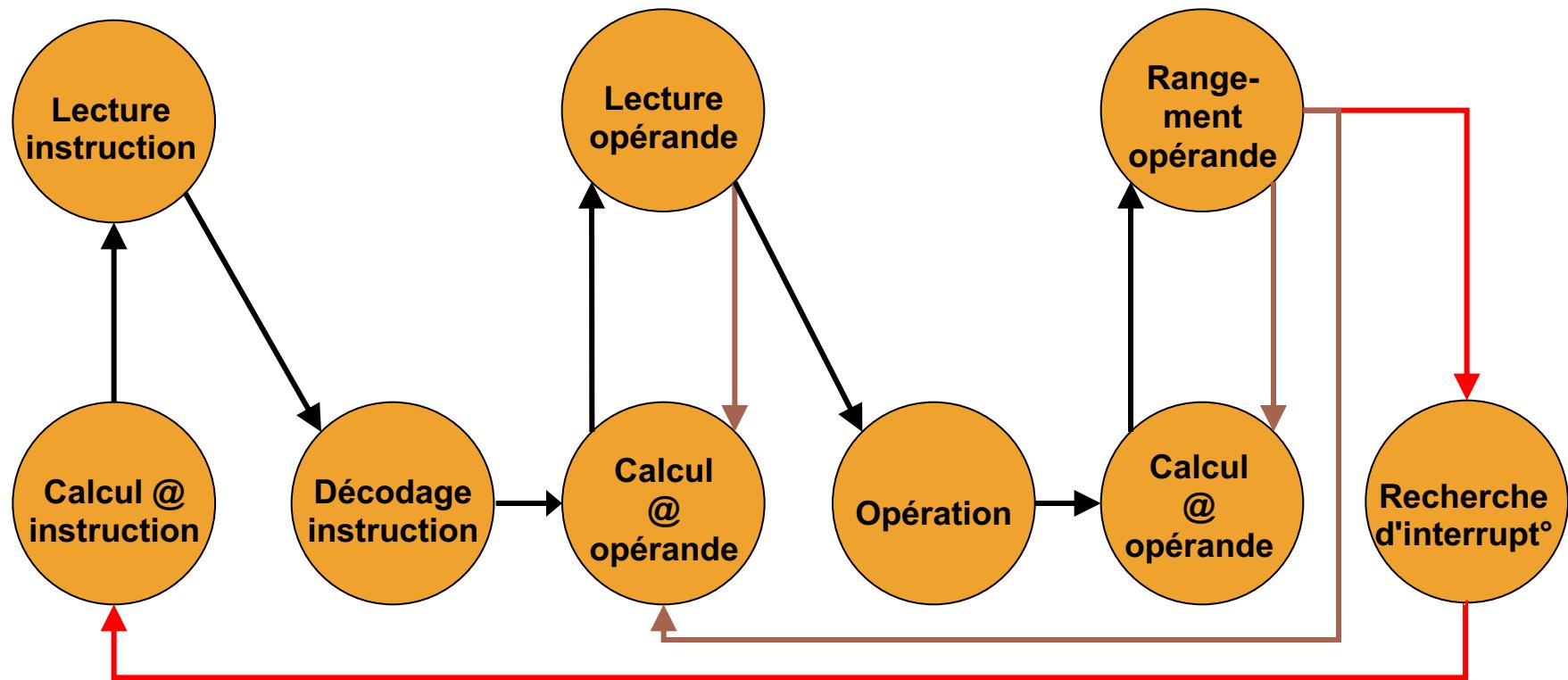
Accès mémoire non autorisé

Tentative d'exécuter une instruction inconnue

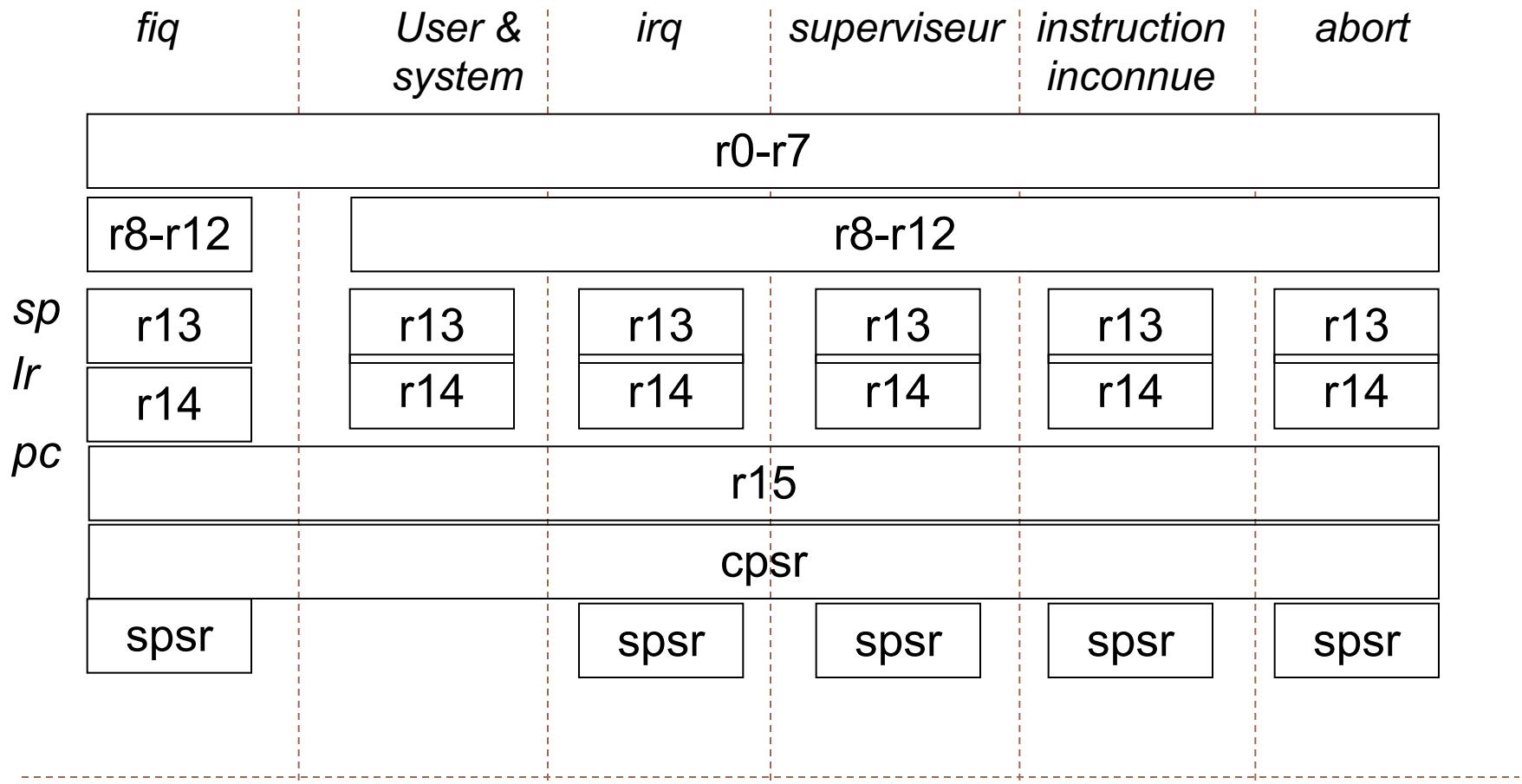
Modes opératoires



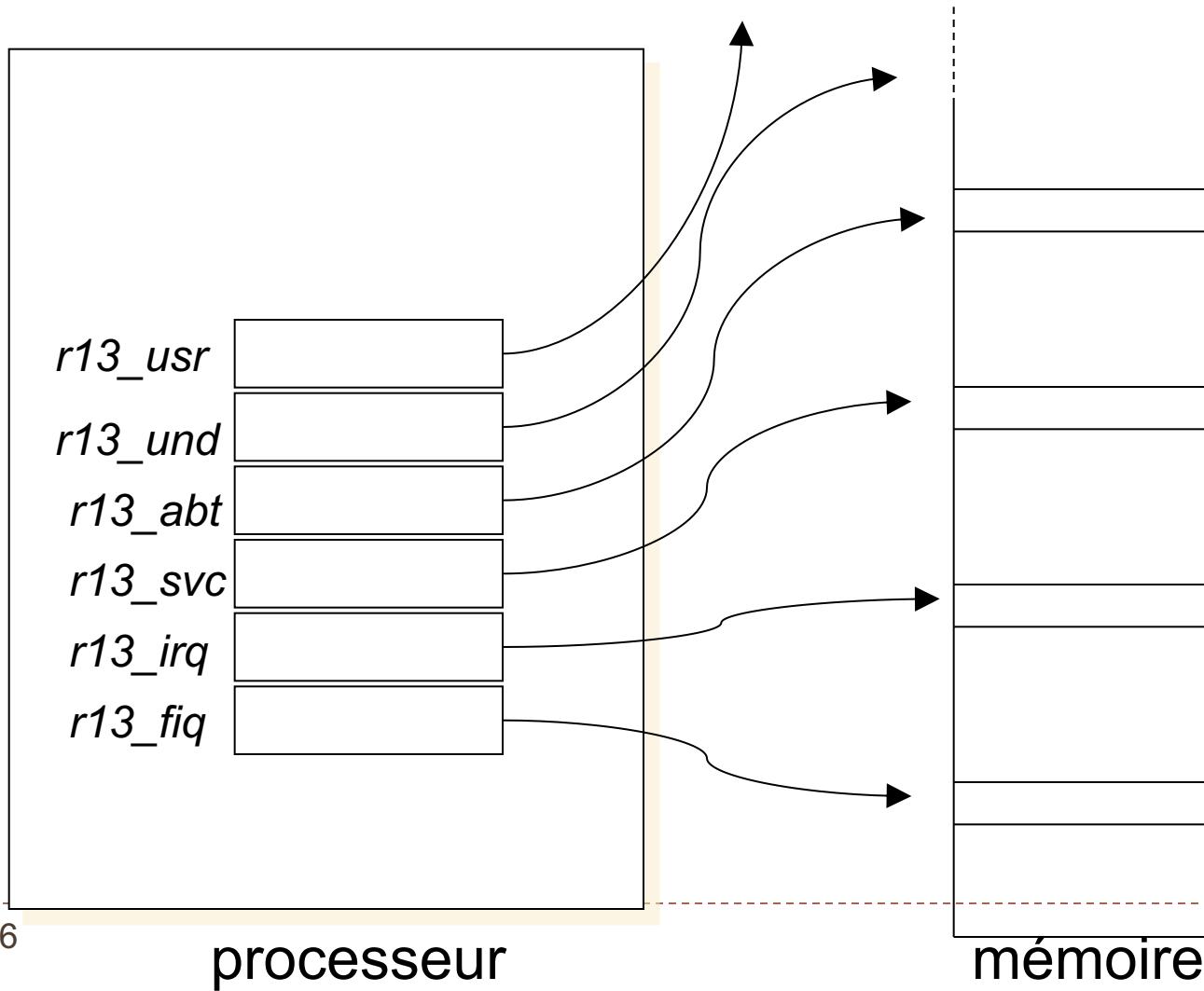
Cycle détaillé avec prise en compte des interruptions



Les registres et les modes

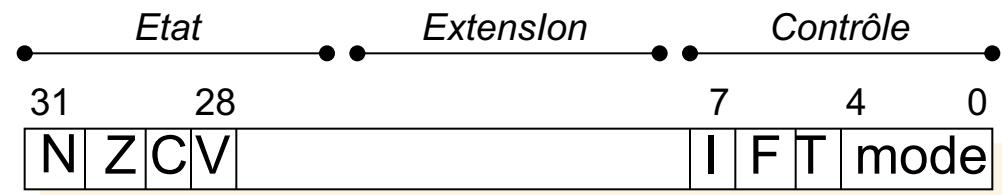


Une pile pour chaque mode



Passage dans un mode

- ▶ Seuls **utilisateur** et **système** ne sont pas liés à un événement
- ▶ Possibilité de passer de n'importe quel mode à n'importe quel mode en modifiant CPSR (mais cpsr n'est pas sauvegardé dans spsr)



- ▶ Le processeur
 - ▶ Sauvegarde CPSR dans le SPSR du mode dans lequel il va passer
 - ▶ Sauvegarde PC (r15) dans le LR (r14) du mode dans lequel il va passer
 - ▶ Modification du CPSR
 - ▶ PC \leftarrow adresse du gestionnaire d'exception

Passage dans un mode

- ▶ Sauvegarde CPSR dans le SPSR du mode dans lequel il va passer
 - ▶ Pour pouvoir restaurer CPSR au retour au sous-programme interrompu
- ▶ Sauvegarde PC (r15) dans le LR (r14) du mode dans lequel il va passer
 - ▶ Pour pouvoir retourner à la bonne instruction du programme interrompu : lr – 4 ou lr (SWI & undef)
- ▶ Modification du CPSR
 - ▶ Pour passer dans le mode
- ▶ PC ← **adresse du gestionnaire d'exception**
 - ▶ Pour exécuter le code correspondant à l'exception

Adresse du gestionnaire d'exception

Table de vecteurs

Exception	Mode	Entrée
nReset	Svc	+0x00
Inst.inconnue	Und	+0x04
SWI	SVC	+0x08
Pref. abort	ABT	+0x0c
Data abort	ABT	+0x10
Non assignée		+0x14
nIRQ	IRQ	+0x18
nFIQ	FIQ	+0x1c

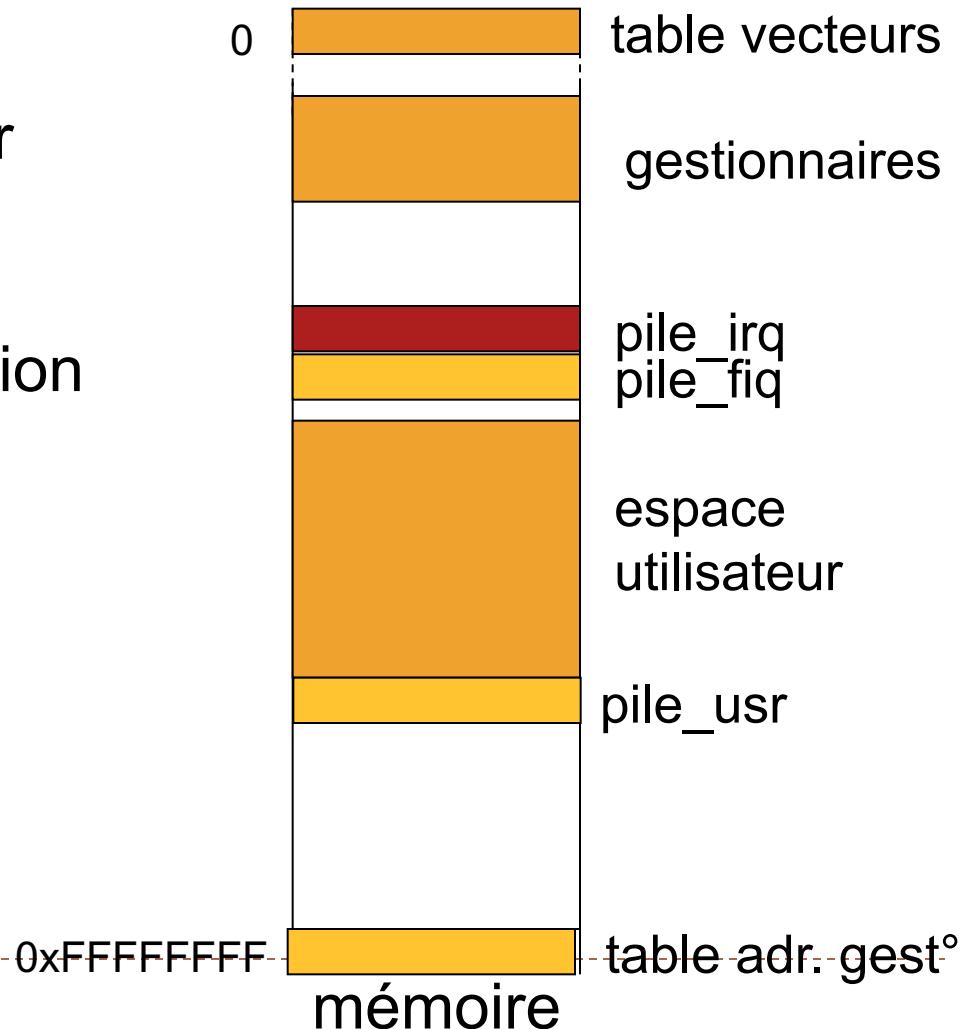
Exemple de table

0x00000000	0xe59ffa38	ldr	pc,	[pc, #reset]
0x00000004	0xea000502	b	undInstr	
0x00000008	0xe59ff018	ldr	pc,	[pc, #swi]
0x0000000c	0xe59ff018	ldr	pc,	[pc, #prefetch]
0x00000010	0xe59ff018	ldr	pc,	[pc, #data]
0x00000014	0xe59ff018	ldr	pc,	[pc, #notassigned]
0x00000018	0xe59ff018	ldr	pc,	[pc,# -0xF20]
0x0000001c	0xe59ff018	ldr	pc,	[pc, #fiq]

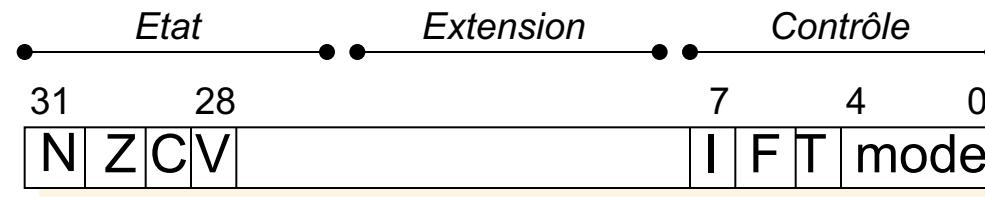
Partition de la mémoire

- Il ne faut pas faire déborder les piles sinon écrasement de données ou de programmes puis exception prefetch abort

- Exemple : programme récursif sans fin
 - nécessite de passer les paramètres par la pile.



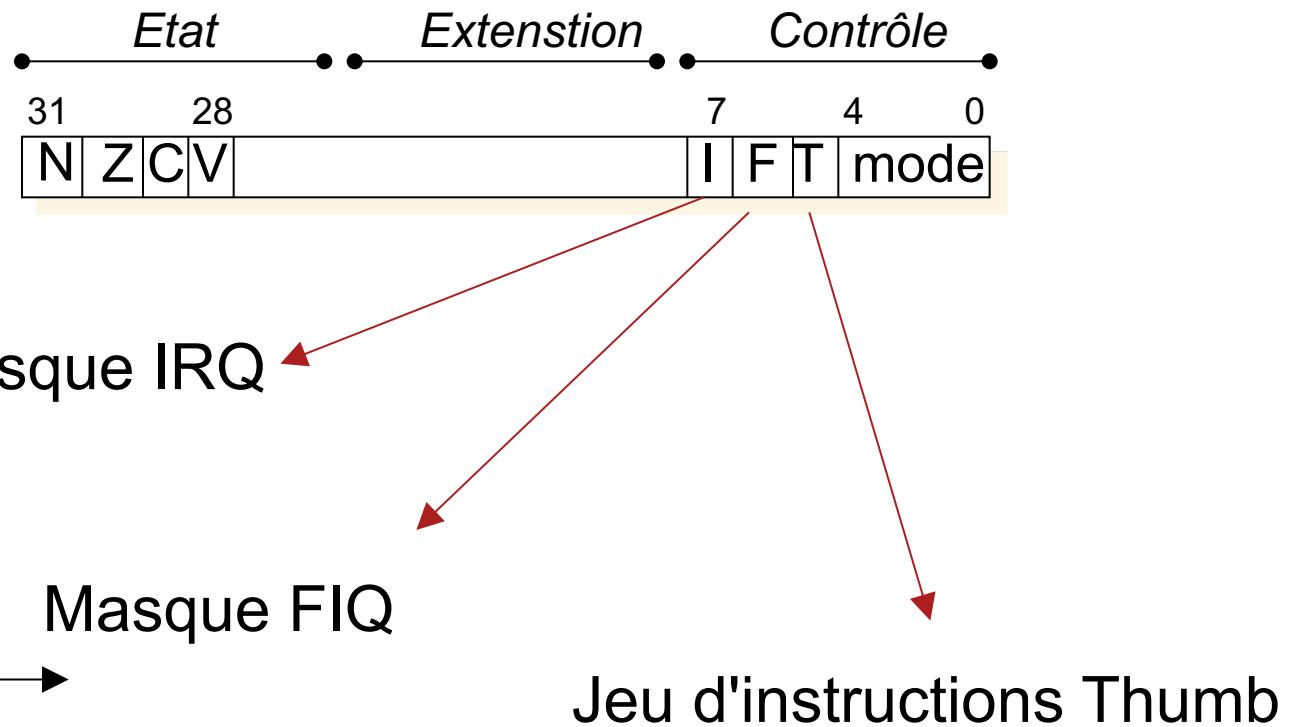
Codage des modes dans CPSR



Accès à un autre jeu d'instruction

Mode	Abréviaction	Privilégié ?	
Abort	Abt	Oui	10111
Fast Interrupt Request	Fiq	Oui	10001
Interrupt Request	Irq	Oui	10010
Superviseur	Svc	Oui	10011
Système	Sys	Oui	11111
Inst. inconnue	Und	Oui	11011
Utilisateur	usr	Non	10000

CPSR



Priorité des exceptions

Mode	Priorité	Bit I	Bit F
Reset	1	1	1
Data Abort	2	1	-
Fast Interrupt Request	3	1	1
Interrupt Request	4	1	-
Prefetch abort	5	1	-
SWI	6	1	-
Inst. inconnue	6	1	-

Ajuster LR dans le gestionnaire de l'exception ?

Exception	Adresse	Utilisation
Reset	-	non défini
Data abort	Ir-8	instruction qui a causé le mauvais accès
Prefetch abort	Ir-4	
FIQ et IRQ	Ir-4	Adresse de retour au programme appelant
SWI et undef	Ir	Adresse de l'instruction qui suit SWI ou l'instruction inconnue

Sous-programme de traitement d'interruption (1)

gest_it

SUB **r14**, **r14**, **#4**

...

MOVS **pc**, **r14**

- ▶ $\text{pc} \leftarrow \text{r14}$ et $\text{cpsr} \leftarrow \text{spsr}$ ($\text{s} + \text{destination} = \text{pc}$)
- ▶ En un seul cycle sinon une autre interruption pourrait arriver entre la mise à jour du pc et la mise à jour de cpsr

gest_it

...

SUBS **pc**, **r14**, **#4**

Sous-programme de traitement d'interruption (2)

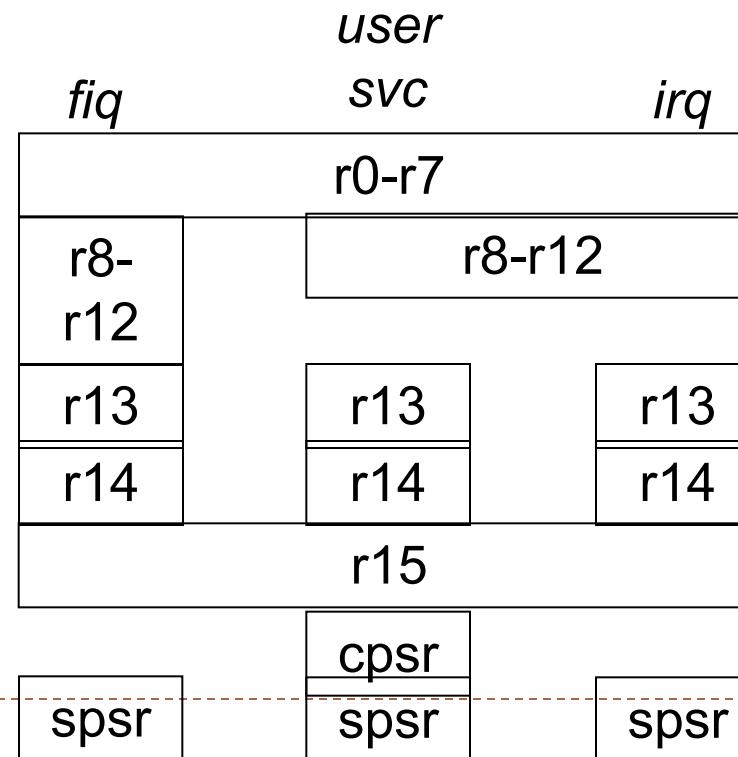
gest_it

```
SUB      r14, r14, #4
STMFD   r13!, {r0-r3,r14}
...
LDMFD   r13!, {r0-r3,r15}^
```

- ▶ Le ^ force le processeur à restaurer spsr dans cpsr **en même temps** qu'il restaure r15.
- ▶ Mise en pile indispensable si ce gestionnaire d'interruption peut lui-même être interrompu sinon écrasement r14 comme pour les sous-programmes.

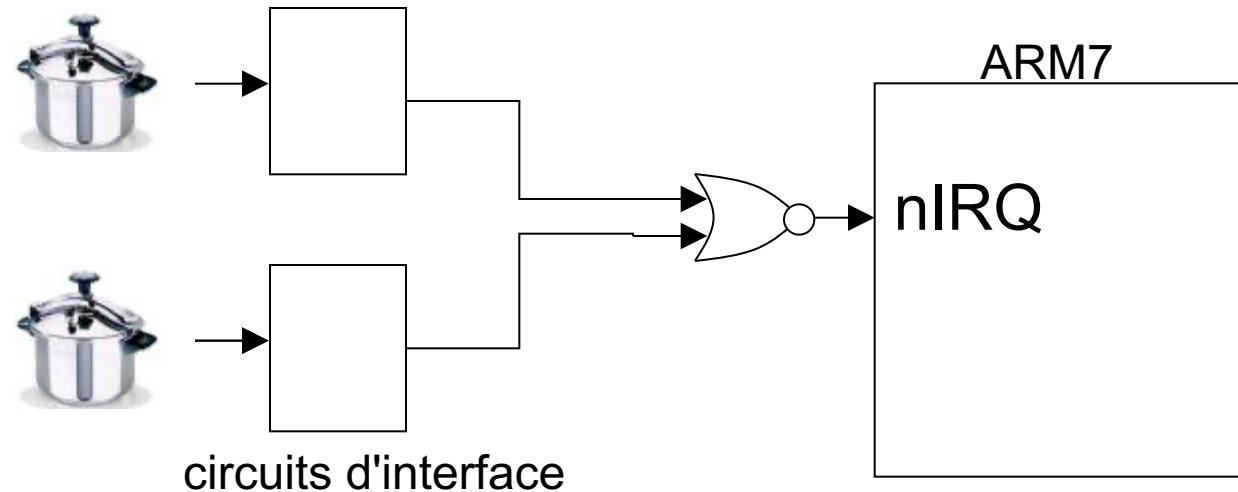
IRQ & FIQ

- ▶ FIQ sert à répondre à un événement plus rapidement que IRQ.
- ▶ Pas besoin de sauvegarder des registres si on peut se contenter de r8-r12.



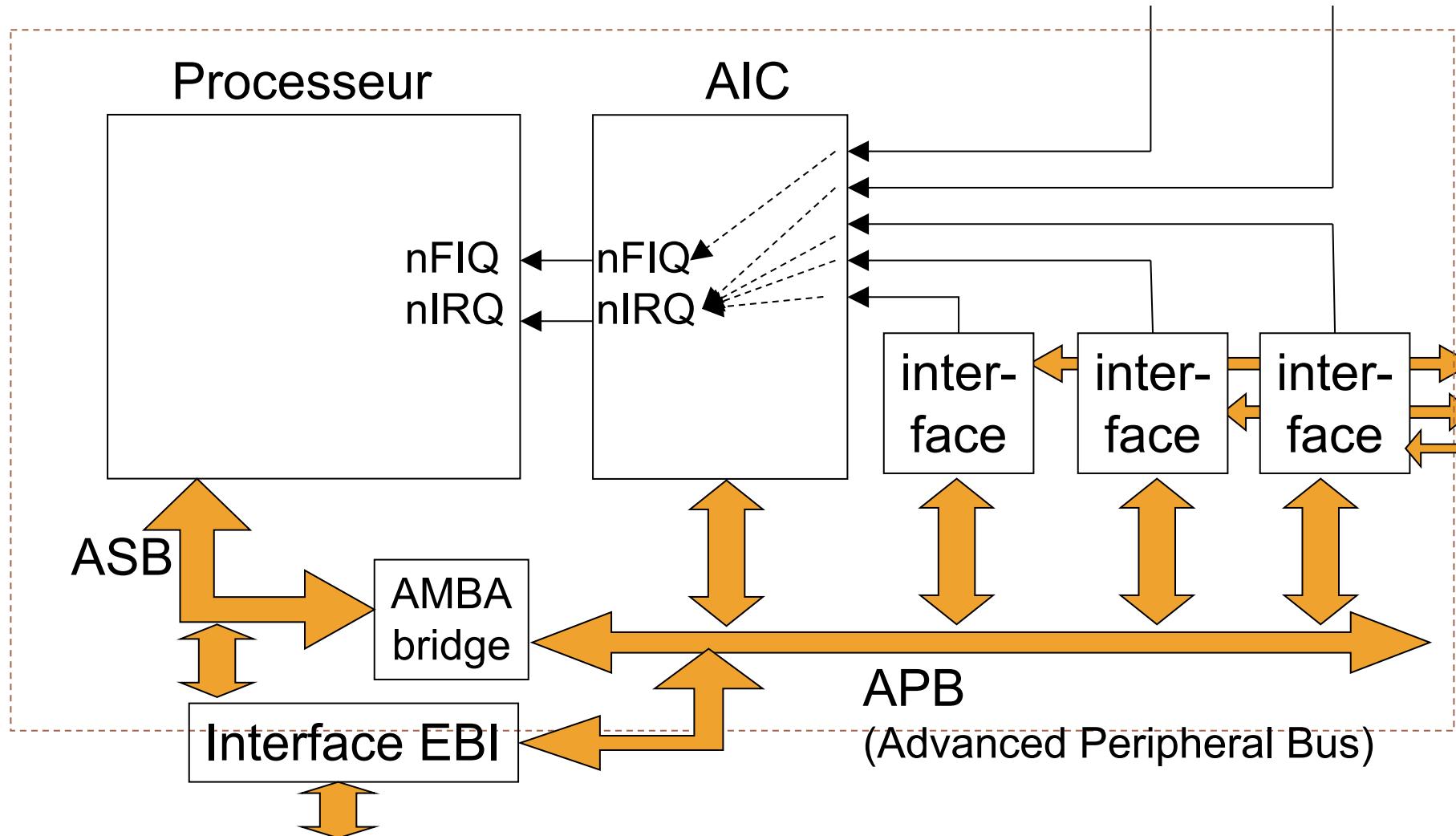
Réagir à plus de 2 événements extérieurs

- ▶ Nécessité de partager les broches IRQ (ou FIQ) entre plusieurs sources d'interruption



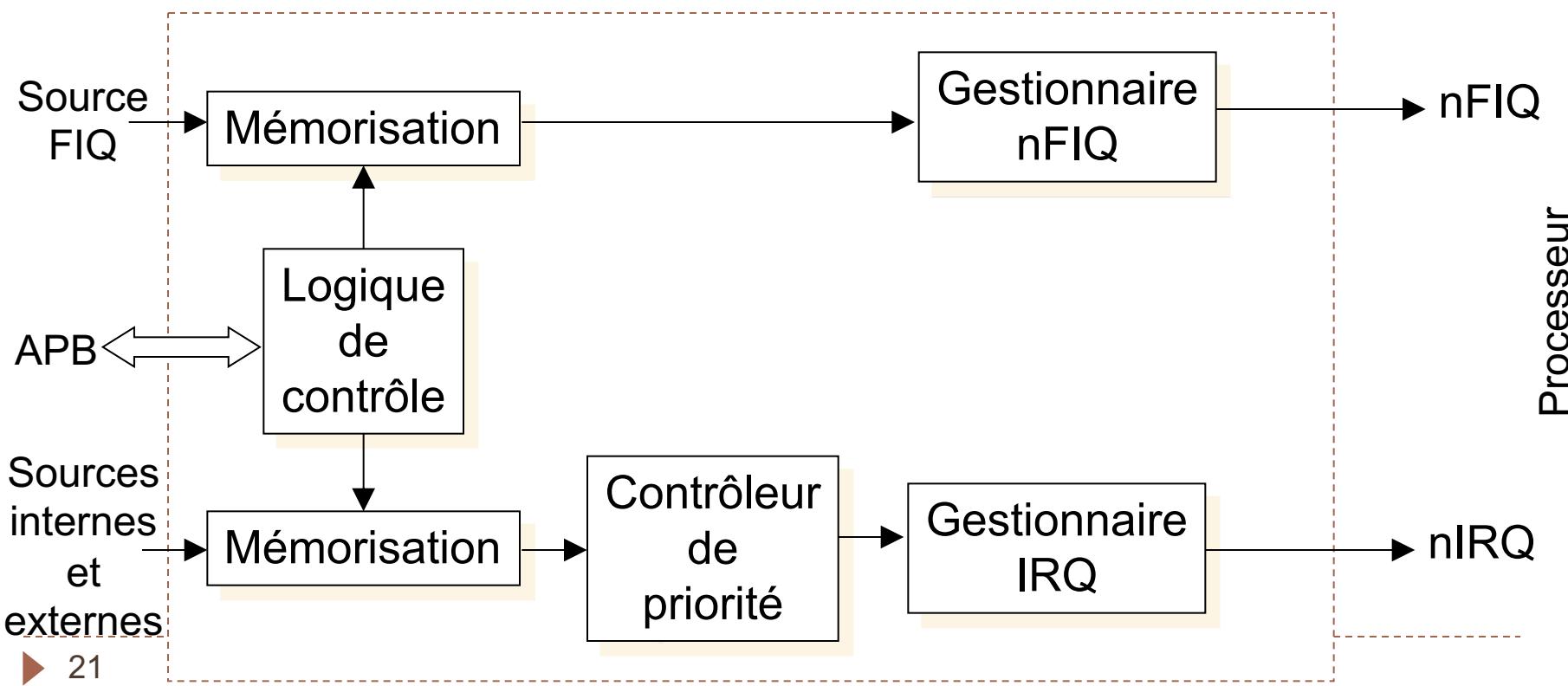
Le sous-programme gérant l'interruption doit aller regarder dans chaque circuit d'interface lequel a provoqué l'interruption (polling). Peut-être long...

Contrôleur d'interruptions (AIC)



Contrôleur d'interruptions (AIC)

Permet une meilleure prise en compte qu'un simple OU.
Fournit automatiquement l'adresse d'un vecteur.



Contrôleur d'interruptions (AIC)

Priorités

- ▶ 32 sources d'interruption possibles
- ▶ Les sources sont numérotées
- ▶ On peut attribuer un niveau de priorité à chaque source (**0 le plus faible, 7 le plus fort**) donc il faut attribuer la même priorité à 2 sources si l'on autorise plus de 8 sources d'interruption.
- ▶ 2 sources sur le même niveau : utiliser le numéro ci-dessous (**1 le plus fort !**).

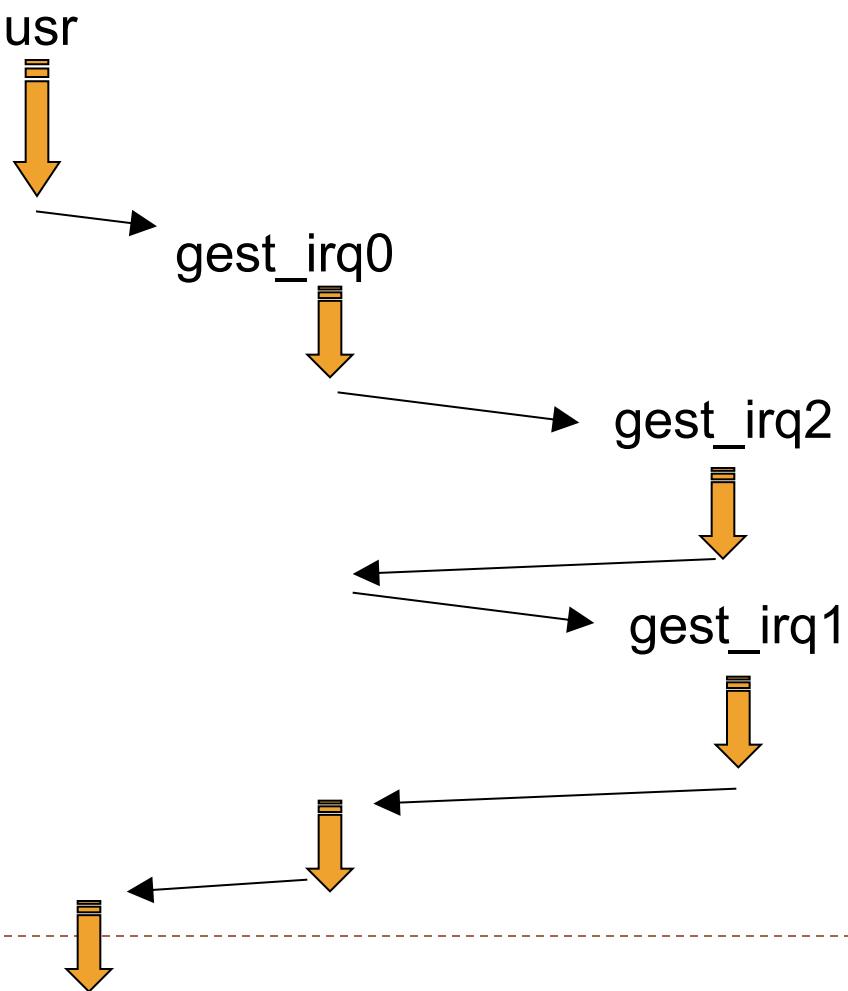
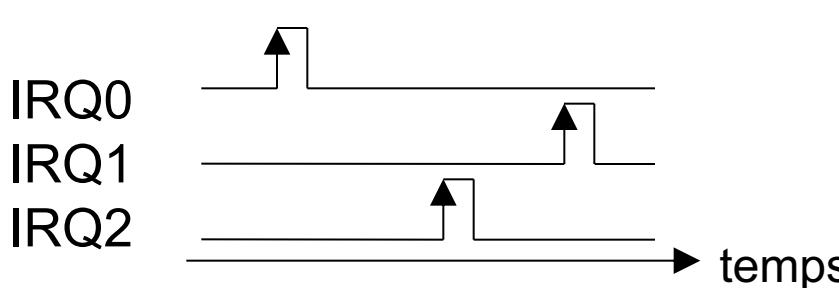
0	1	2	3	4	5	6	7	8	16	17	18
FIQ RQ	SWI IRQ	US0 IRQ	US1 IRQ	TC0 IRQ	TC1 IRQ	TC2 IRQ	WDI RQ	PIOI RQ	IRQ 0	IRQ 1	IRQ 2

Contrôleur d'interruptions (AIC)

Priorités : exemple

► Hypothèse :

- ▶ IRQ0 niveau 0
- ▶ IRQ1 niveau 1
- ▶ IRQ2 niveau 2
- ▶ Bit I du CPSR
systématiquement à 0 (pas de masquage)



Adressage des circuits d'interface (et de l'AIC)

- ▶ Les circuits d'interface et l'AIC doivent être configurés et contrôlés par le logiciel
 - ▶ Lecture/écriture dans des registres internes à ces circuits
- ▶ De manière générale, 2 possibilités
 - ▶ instructions spéciales
 - ▶ circuits présents dans l'espace d'adressage
- ▶ Quelquefois
 - ▶ registres en lecture seulement ou en écriture seulement
 - ⌚ nécessité d'écrire dans un registre pour accéder à un autre.

Adressage des circuits d'interface (et de l'AIC)

▶ ARM7

- ▶ circuits présents dans l'espace d'adressage
- ▶ Quelquefois
 - ▶ registres en lecture seulement ou en écriture seulement

▶ X86

- ▶ Accès aux circuits par instructions spéciales

Les registres de l'AIC et leur adressage

- ▶ Adresse de base : 0xFFFFF000
- ▶ Adresse de fin : 0xFFFFF14B
- ▶ Pour chaque source, un registre AIC_SMR_i
 - ▶ mode de la source
- ▶ Pour chaque source, un registre AIC_SVR_i
 - ▶ vecteur associé à la source
- ▶ AIC_IVR, AIC_FVR, AIC_ISR, AIC_IPR, AIC_IMR,
AIC_CISR, AIC_IECR, AIC_IDCR, AIC_ICCR, AIC_ISCR,
AIC_EOICR, AIC_SPU, ...

Affectation d'une priorité

- ## ► Mode de la source (AIC_SMRI)

7	6	5	4	3	2	1	0
-----		type	-----			priorité	
00 : niveau bas				Pour les sources internes , 00 ou 01			
01 : front descendant							
10 : niveau haut							
11 : front montant							ne sert à rien pour SMR0

0 le moins prioritaire

AIC_IVR (AIC_FVR) et AIC_SVRi

- ▶ L'AIC reçoit l'IRQ n° i
 - ▶ Il recopie AIC_SVRi dans AIC_IVR (si la priorité de l'interruption i est > à la priorité de l'interruption en cours).
 - ▶ génère IRQ vers le processeur
- ▶ Le processeur reçoit IRQ, se branche sur le vecteur associé
 - ▶ adresse 0x18 qui contient ldr PC, [PC, # - 0xF20]
 - ▶ accès à l'adresse $0x20 - 0xF20 = 0xFFFFF100$
 - ▶ $0xFFFFF100 = \text{adresse registre AIC_IVR}$
- ▶ Lecture d'AIC_IVR
 - ▶ AIC désactive la ligne IRQ entre l'AIC et le processeur.
 - ▶ L'AIC la réactivera s'il reçoit une interruption plus prioritaire.
- ▶ Pour FIQ, idem avec AIC_FVR (0xFFFFF104)

AIC_EOICR

- ▶ L'AIC doit savoir quand le traitement de l'interruption en cours est terminée.
- ▶ Pour se faire, le gestionnaire d'interruptions doit écrire dans AIC_EOICR.
- ▶ L'AIC restaure le niveau d'interruption précédent (s'il y en avait une en cours).
- ▶ Une interruption en attente peut alors être traitée.

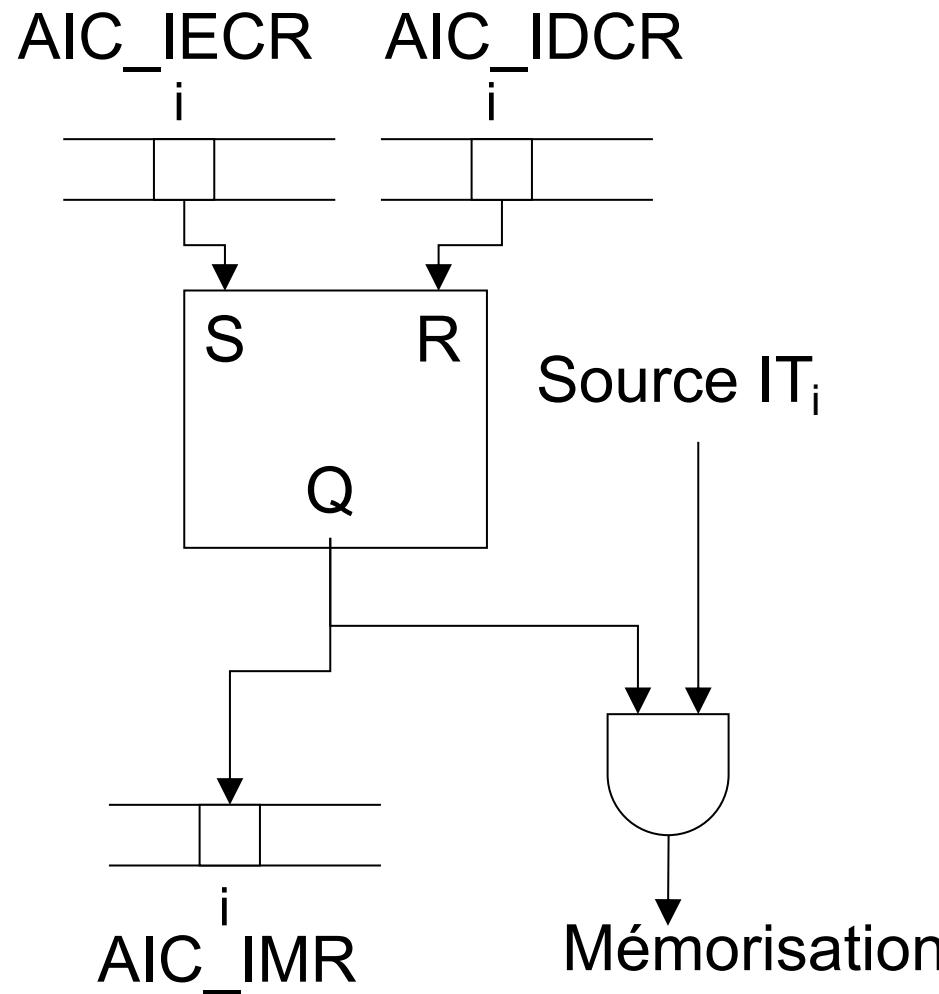
AIC_IPR

- ▶ Permet de connaître toutes les interruptions en attente.

AIC_IECR, AIC_IDCR, AIC_IMR

- ▶ Interrupt Enable/Disable Control Registers
- ▶ AIC_IECR et AIC_IDCR
 - ▶ Autorisation ou interdiction d'une interruption en entrée de l'AIC
 - ▶ Jouent le même rôle que I & F mais pour chaque interruption en entrée de l'AIC.
 - ▶ En écriture seulement
- ▶ AIC_IMR
 - ▶ En lecture seulement
 - ▶ Permet de savoir quelles interruptions sont masquées

AIC_IECR, AIC_IDCR, AIC_IMR



Interruption gérée par logiciel

- ▶ Source I doit être programmée comme sensible à un front.
 - ▶ $\text{SMR}[I] = 0b0100111$
- ▶ Peut-être déclenchée par écriture dans $\text{AIC_ISCR}[I]$ et annulée par écriture dans $\text{AIC_ICCR}[I]$

Séquence type d'interruption (1)

Hyp :

- vecteur 0x18 initialisé avec `ldr PC, [PC, #-0x20]`
- vecteur AIC initialisé (SVRx)

Processeur

nIRQ passe à 1

Si CPSR[I]=0

alors

`SPSR_IRQ ← CPSR`

`r14_irq ← r15`

`r15 ← 0x18 (ldr pc,[pc,# -0xF20])`

Entrée dans mode irq

modif cpsr

`pc ← AIC_IVR (ldr pc,[pc,# -0xF20])`

AIC

A la lecture d'AIC_IVR

ajuste le niveau d'IT. Désactive nIRQ du processeur. supprime l'interruption de la file si elle est sur front.

sauvegarde le niveau d'IT dans sa pile interne

Renvoie la valeur contenue dans AIC_IVR sur le bus.

Séquence type d'interruption (2)

Le gestionnaire d'interruption

- ▶ doit sauvegarder r14 (après l'avoir décrémenté de 4) et SPSR dans la pile (et les registres utilisés)
- ▶ peut démasquer les ITs
 - ▶ $(CPSR[I] \leftarrow 0$ (pour se faire interrompre par d'autres ITs plus prioritaires au niveau de l'AIC)
- ▶ Traitement à faire
- ▶ remasquer les ITs
- ▶ Ecrire dans AIC_EOICR
- ▶ restaurer SPSR depuis la pile ainsi que LR
- ▶ $PC \leftarrow LR$ et $CPSR_usr \leftarrow SPSR$

AIC

A l'écriture dans AIC_EOICR

- ▶ restaure le niveau d'interruption depuis sa pile interne
- ▶ Si interruption + prioritaire en attente, activation ligne nIRQ

comme **CPSR[I]** est à 1, ne sera pris en compte qu'à la sortie du gestionnaire d'interruption

Début gestionnaire d'interruption générique

```
sub    r14,r14,#4          @ sauve @ de retour
stmfd r13!,{r14}           @ dans la pile_irq
mrs    r14,SPSR            @ sauve SPSR
stmfd r13!,{r14}           @ dans pile_irq
mrs    r14,cpsr             @ r14 a été sauvé,
                             @ on peut l'utiliser
bic    r14,r14,#0x1f        @ efface le mode
orr    r14,r14,#0x13        @ mode super
msr    cpsr, r14            @ ici, on passe en mode
                             @ superviseur
stmfd r13!,{ri,rj,r14}     @ sauve r14_svc
mrs    r14,cpsr             @ raz bit I de CPSR
bic    r14,r14,#0x80        @ pour autoriser les
                             @ interruptions
```

<handler_code pouvant contenir des bl>

Fin gestionnaire d'interruption générique

...

```
mrs r14,CPSR          @ remasquer les ITs
orr r14,r14,#0x80      @ bit I à 1
msr CPSR,r14

ldmfd r13!,{ri,rj,r14}    @ restaurer r14_svc
mrs r14,cpsr
bic r14,r14,#0x1f        @ efface le mode
orr r14,r14,#0x12        @ mode irq
msr cpsr, r14            @ re passer en mode irq

ldr r14,=AIC_BASE        @0xFFFFF000
str r14,[r14,#AIC_EOICR] @ prévenir AIC

ldmfd r13!,{r14}         @ restaurer SPSR
msr SPSR,r14
ldmfd r13!,{r15}^        @ retour au programme interrompu
                        @ dans le mode interrompu
                        @ (user ou svc)
                        @ recopie spsr dans cpsr
```

Autre solution pour éviter de changer de mode

- ▶ On reste en mode irq mais on interdit les interruptions avant tout appel de sous-programme
- ▶ Et on les autorise à nouveau une fois revenu du sous-programme
- ▶ Inconvénient : retardé la prise en compte des ITs

Traitements d'une FIQ

- ▶ Plus rapide car ne peut pas être interrompu
- ▶ Non sauvegarde de SPSR et de LR (sauf si appel de sous-programme dans le gestionnaire FIQ)
- ▶ Non sauvegarde des registres r8-r12