Ch. 10
Entrées-Sorties

B. Quoitin
(bruno.quoitin@umons.ac.be)

Table des Matières

- **Introduction**
- **Types de périphériques**
- **Contrôleur d'entrées/sorties**
- **Opérations d'entrées/sorties**
- **Etude de cas**

Introduction

- **Objectifs**
  - **Processeur et mémoire en circuit fermé**
    - La paire processeur/mémoire décrite dans le chapitre 4 fonctionnait en circuit fermé. Les programmes et leurs données résidaient déjà en mémoire. Les résultats obtenus étaient également uniquement stockés en mémoire.
  - **Entrées / sorties**
    - En pratique, il est nécessaire de faire communiquer le processeur et la mémoire avec le monde extérieur. Les mécanismes qui permettent cette communications sont désignés par entrées / sorties (i*nput / output*).
      - interaction avec l'utilisateur (p.ex. clavier et écran)
      - stockage permettant de rendre persistantes les données informatisées
      - communications entres machines (p.ex. réseaux de communication).
      - interaction avec l'environnement (p.ex. mesure de la température)
      - contrôle de périphériques spécifiques (p.ex. bras articulé de robot).

Introduction

Différents types de périphériques d'E/S

- Il existe une grande variété de périphériques d'entrées sorties et ces périphériques ont des caractéristiques d'accès très différentes.

Dispositif	Entrée/Sortie	Utilisateur	Débit (KB/s)
Clavier	E	Humain	0.01
Souris	E	Humain	0.02
Voix (4000Hz, 8bits)	E	Humain	4
Scanner (300x300 DPI, A4)	E	Humain	26000
Ecran	S	Humain	60000
Son (44000Hz, 16 bits, stér.)	S	Humain	176
Réseau LAN (Ethernet Gbps)	E/S	Machine	125000
Disque dur (7200 RPM)	Stockage	Machine	100000
Disque optique (DVD 24x)	Stockage	Machine	30000

Table des Matières

- Introduction
- **Types de périphériques**
- **Contrôleur d'entrées/sorties**
- **Opérations d'entrées/sorties**
- **Etude de cas**

Types de Périphériques

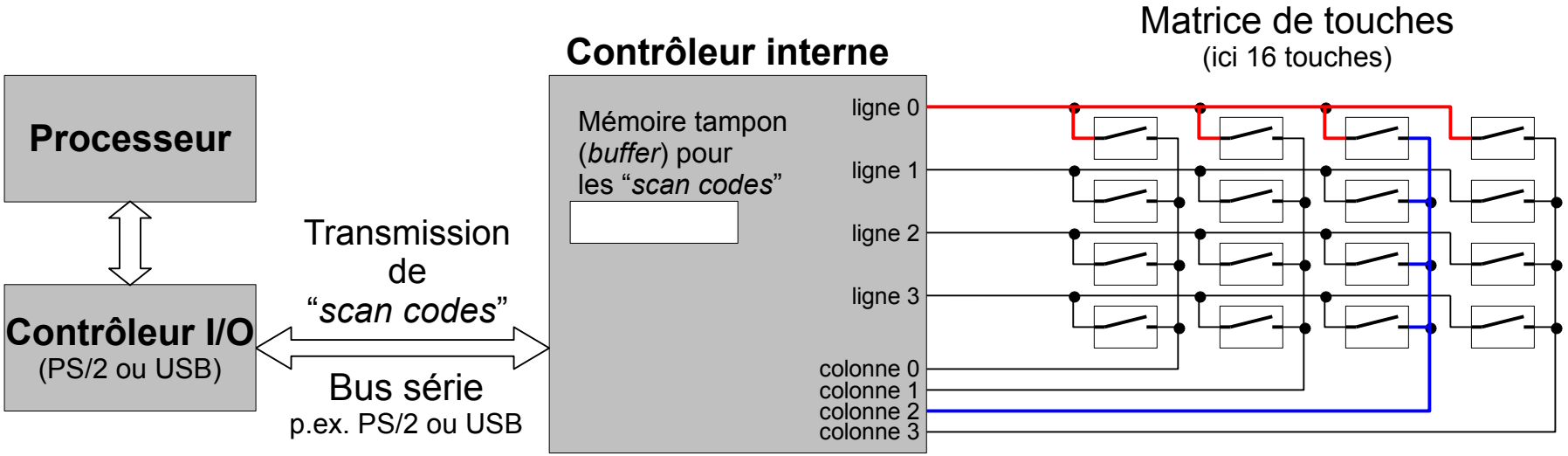
- **Principe**
  - **Clavier**
    - Le clavier est un périphérique d'entrée.
    - Il peut aussi être considéré comme un périphérique de sortie car visualisation de certains états (p.ex. *verrouillage majuscules* ⇪)

    - Un clavier d'ordinateur comporte une 100aine de touches. Chaque touche est un *interrupteur*.

    - Le clavier est connecté avec l'ordinateur via un *bus* (typiquement série) tel que PS/2 ou USB.

Types de Périphériques

- **Principe**
  - **Clavier**
    - Un clavier n'est pas directement raccordé au processeur, mais contient un **contrôleur** interne
      - « scan » d'une matrice de touches (réduit le nombre de connexions clavier-ordinateur)
      - suppression des rebonds
      - communication via un bus série (p.ex. PS/2 ou USB).
      - stockage des codes de touche (*scan codes*) dans une mémoire tampon.

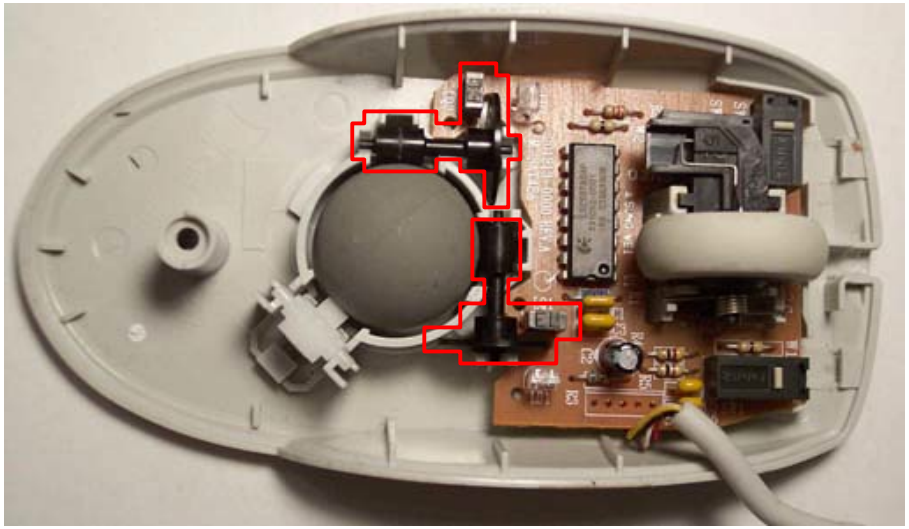


Types de Périphériques

- **Principe**
  - **Souris**
    - Une souris ou un *trackpad* sont d'autres périphériques très répandus. Une souris transmet plusieurs informations vers le processeur (elle contient également un contrôleur qui s'occupe de la communication avec l'ordinateur)
      - des changements de position (delta X, delta Y)
      - l'état des boutons
      - les changements de la roue de défilement (*scroll wheel*)

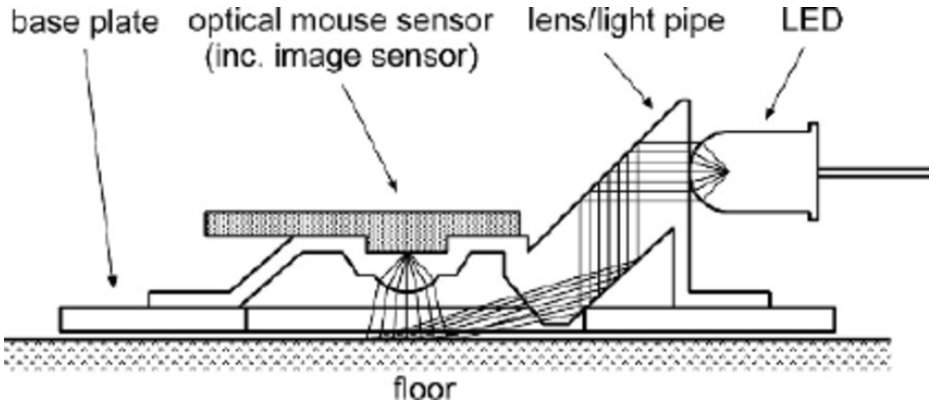
Principe de fonctionnement

- Dans le passé, les souris étaient basées sur une boule agissant sur deux **roues encodeuses** disposées perpendiculairement. Des capteurs optiques mesuraient la quantité et le sens de déplacement selon deux axes orthogonaux (x et y).



Types de Périphériques

- **Principe**
  - **Souris optique**
    - Aujourd'hui, les souris utilisent des techniques sophistiquées d'analyse d'images.
    - Les souris optiques intègrent une source lumineuse (typiquement LED) qui éclaire la surface sous la souris. Une caméra prend des “photos” de cette surface très fréquemment (plusieurs 100aines de fois par seconde). Un processeur spécialisé (DSP(1)) détecte certaines formes dans l'image et détermine les déplacements entre photos successives !



(1) Digital Signal Processor (DSP)

Source image : Sekimori, Daisuke & Miyazaki, Fumio. ***Precise dead-reckoning for mobile robots using multiple optical mouse sensorsx4***. Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics (ICINCO), Barcelona, Spain, 2005.

Types de Périphériques

- **Principe**
  - **Lecteur de disque dur**
    - Un “*disque dur*” ou **lecteur de disque dur** (*hard disk drive*) désigne un assemblage de disques magnétiques ainsi que la mécanique et l'électronique qui servent à y lire/écrire des données.

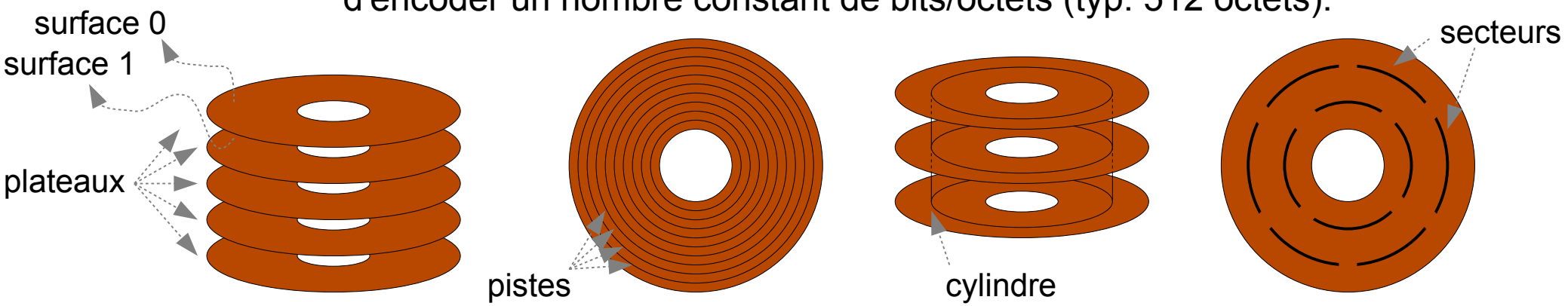
Types de Périphériques

- **Principe**
  - **Lecteur de disque dur**
    - La **densité de stockage par surface** (*areal density*) d'un disque dur indique le nombre de bits stockables par unité de surface.
    - Cette densité n'a cessé de croître. Le taux de croissance annuel jusqu'en 2006 a atteint 100% (un doublement chaque année). En 2012, il était redescendu à moins de 30% (moins d'un doublement tous les 3 ans).
    - En août 2015, Seagate annonçait(1) avoir atteint une densité égale à 1,34 Tb / pouce^2 (207 Gb / cm^2). En 2018, la technologie HAMR(2) (*Heat-Assisted Magnetic Recording*) a permis à Seagate d'atteindre 2 Tb / pouce^2 (310 Gb / cm^2).

(1) Source : https://www.seagate.com/www-content/investors/_shared/docs/tech-talk-mark-re-20150825.pdf
(2) Source : https://blog.seagate.com/craftsman-ship/hamr-next-leap-forward-now/

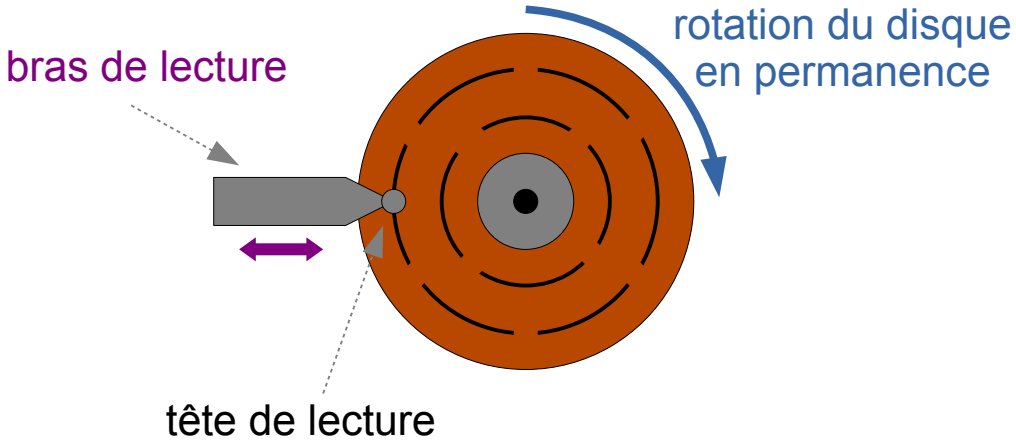
Types de Périphériques

- **Principe**
  - **Lecteur de disque dur**
    - Dans un disque, on distingue les éléments suivants :
      - Un empilement de plusieurs **plateaux** (*platters*). Chaque plateau a la forme d'un disque et est réalisé avec un substrat rigide couvert d'une fine couche d'un matériau *ferromagnétique*. Il est possible d'écrire sur les deux faces (surfaces) d'un plateau. Les plateaux tournent à la même vitesse.
      - Un plateau est découpé en cercles concentriques appelés **pistes** (*tracks*). L'ensemble des pistes de même diamètre dans un ensemble de plateaux est appelé un **cylindre** (*cylinder*).
      - Une piste est découpée en plusieurs **secteurs** (*sectors*) permettant d'encoder un nombre constant de bits/octets (typ. 512 octets).



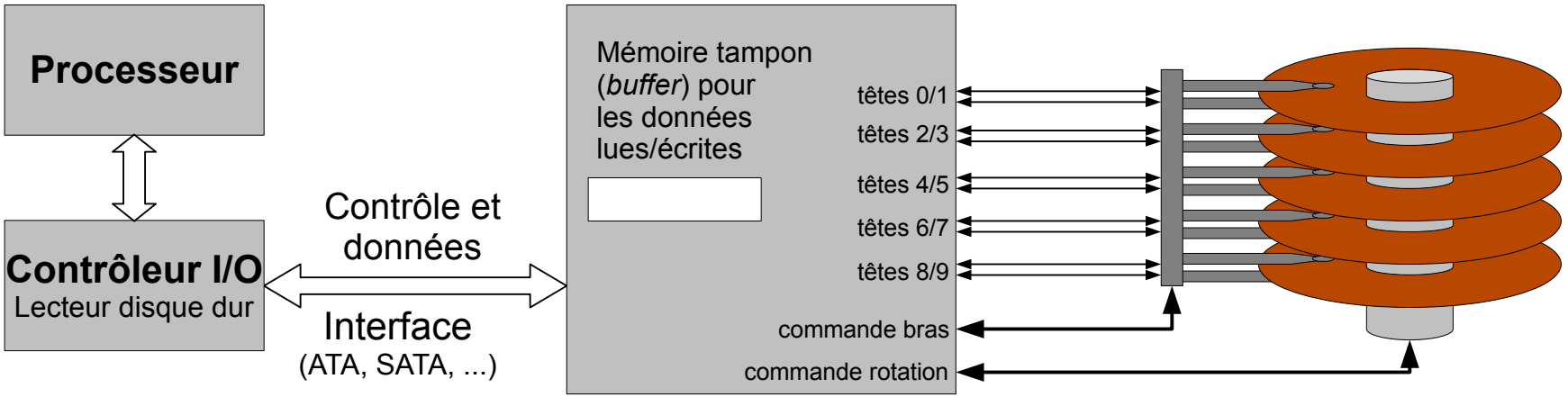
Types de Périphériques

- **Principe**
  - **Lecteur de disque dur - opération**
    - L'accès à un disque consiste à lire/écrire un ou plusieurs secteurs particuliers. Pour lire un secteur particulier, il faut faire tourner le disque mais également déplacer la tête de lecture au dessus de la piste contenant le secteur voulu.
      - Les disques tournent en permanence (lorsque le disque n'est pas en veille). La vitesse de rotation est constante.
      - Le bras de lecture peut être déplacé vers l'intérieur ou l'extérieur du disque. Ce déplacement nécessite un certain temps.



Types de Périphériques

- **Principe**
  - **Lecteur de disque dur - interface**
    - Un lecteur de disque dur n'est pas directement connecté au processeur. Un *contrôleur* de lecteur de disque dur est utilisé à cet effet. Un tel contrôleur peut être connecté à plusieurs disques.
    - Le contrôleur et le lecteur utilisent une *interface* bien définie pour discuter entre eux. Il existe plusieurs standards:
      - parallèle: IDE/ATA, SCSI, …
      - série: Fibre Channel, Serial-ATA (SATA), Serial Attached SCSI (SAS), ...



Types de Périphériques

- **Principe**
  - **Lecteur de disque dur - performances**
    - Les performances d'un lecteur de disque dur sont exprimées avec plusieurs métriques
      - **vitesse de rotation**(1) (*rotational rate*) exprimée en *rotations par minute* (RPM)
      - **temps moyen de positionnement** (*seek time*) exprimé en secondes
      - **taux de transfert** (*transfer rate*) exprimé en octets/seconde. Ce temps dépend de la vitesse de rotation et du nombre de secteurs par piste.

(1) En fait il faudrait plutôt parler de nombre de rotations par seconde que de vitesse de rotation.

Types de Périphériques

- **Exemple**
  - **Lecteur de disque dur - performances**
    - Soit un lecteur de disque dur avec les caractéristiques suivantes
      - une seule surface
      - vitesse de rotation de 7200 RPM
      - temps moyen de positionnement de 10 ms
      - nombre moyen de 500 secteurs par piste.
      - **Quel sera le temps moyen d'accès à un secteur ?**
    - Solution
      - accéder à la piste = 10 ms en moyenne
      - accéder au secteur = temps pour effectuer une demi-rotation en moyenne ; temps pour une rotation complète = 60 s / 7200 RPM = 8,33 ms ; temps moyen d'accès au secteur = 4,16 ms
      - temps moyen de lecture d'un secteur = temps rotation / nombre de secteurs par piste = 8,33 ms / 500 = 16 us
      - temps total = ~ 10 ms + 4,16 ms + 16 us = 14,16 ms

Types de Périphériques

- **Exemple**
  - **Lecteur de disque dur - performances**
    - Soit les données suivantes
      - Fichier de 1MB découpé en blocs de 512 octets
      - Vitesse de rotation: 10000 RPM
      - Temps moyen de positionnement: 5 ms
      - Nombre moyen de secteurs par piste: 1000
      - Surfaces: 4
      - Taille d'un secteur: 512 octets
    - **Question :** estimer le temps nécessaire pour lire le fichier (lecture séquentielle de ses blocs)
      - 1). Cas favorable : si les blocs du fichier sont arrangés de manière à nécessiter le minimum de déplacements
      - 2). Cas défavorable : les blocs du fichier sont arrangés de manière aléatoire

Types de Périphériques

- **Principe**
  - **Disques redondants**
    - Pour des raisons de performances et/ou de fiabilité, il est possible d'utiliser plusieurs disques durs en même temps.
    - **Fiabilité**
      - Si un disque tombe en panne, le contenu est disponible sur les autres disques.
      - D'après une étude récente(1), le disque dur est le composant d'un ordinateur qui tombe en panne le premier dans jusqu'à 50% des cas !
    - **Performance**
      - Répartition de l'accès sur différents disques
      - Par exemple : lecture de 2 secteurs simultanément sur 2 disques → double le débit.

(1) **Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?**, B. Schroeder and G. A. Gibson, 5th USENIX Conference on File and Storage Technologies, 2007.

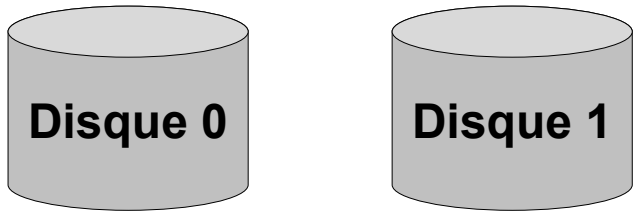
Types de Périphériques

- **Principe**
  - **RAID**
    - Le « standard(1) » **RAID** (*Redundant Array of Independent/Inexpensive Disks*) désigne plusieurs organisations de disques redondants. Il existe plusieurs versions (niveaux) de RAID.
      - **RAID 0** (JBOD – *Just a Bunch Of Disks*) : répartition des blocs de données successifs sur des disques différents afin d'augmenter la débit total (*striping*)
      - **RAID 1** : utilisation d'au minimum deux disques ; chaque donnée est stockée en double (*mirroring*/*shadowing*)
      - **RAID 2/3/4** : utilisation d'au minimum 3 disques ; stockage de bit/octet/bloc de parité sur un disque supplémentaire → possibilité de récupérer en cas d'erreur d'un disque.
      - **RAID 5** : idem RAID 4 mais parité distribuée sur les différents disques

(1) Note : RAID level 1-5 ont été initialement proposés par D. Patterson et al dans un article intitulé "**A Case for Redundant Arrays of Inexpensive Disks (RAID)**", ACM SIGMOD 1988.

Types de Périphériques

- **Principe**
  - **RAID 0**
    - Exemple : utilisation de 2 disques de 500 GB pour constituer un volume total de 1000 GB. Un bloc est constitué de 512 octets (un secteur).

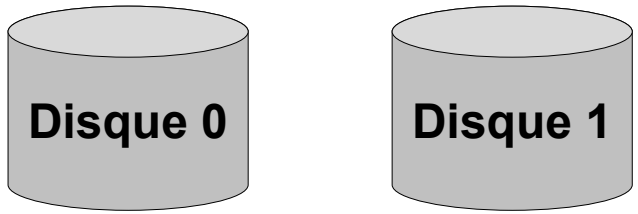


contenu:

Disque 0	Disque 1
B0	B1
B2	B3
B4	B5
…	…

Types de Périphériques

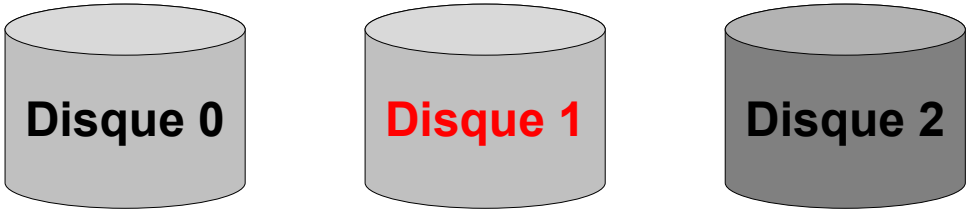
- **Principe**
  - **RAID 1**
    - Exemple : utilisation de 2 disques de 500 GB pour constituer un volume total de 500 GB avec deux copies exactes. Un bloc est constitué de 512 octets (un secteur).



contenu:		
	B0	B0
	B1	B1
	B2	B2
	B3	B3
	…	…

Types de Périphériques

- **Principe**
  - **RAID 2/3/4**
    - Exemple : utilisation de 3 disques de 500 GB pour constituer un volume total de 1000 GB. Un bloc est constitué de 512 octets (un secteur). RAID 4.

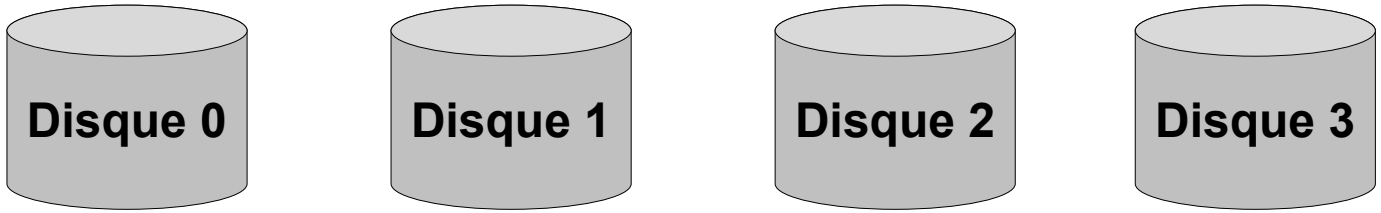


contenu:			
	B0	**B1**	**B0⊕B1**
	B2	B3	B2⊕B3
	B4	B5	B4⊕B5
	…	…	

    - En cas de panne d'un seul des disques (0, 1 ou 2), il est possible de récupérer les blocs.
    - Par exemple, si le disque 1 tombe en panne, et que l'on souhaite lire le bloc B1, on peut obtenir son contenu avec B0 lu sur le disque 0 et B0⊕B1 lu sur le disque 2. En effet, B1=B0⊕(B0⊕B1)

Types de Périphériques

- **Principe**
  - **RAID 5**
    - Exemple : utilisation de 4 disques de 500 GB pour constituer un volume total de 1500 GB. Un bloc est constitué de 512 octets (un secteur).



contenu:				
	B0	B1	B2	B0⊕B1⊕B2
	B3	B4	**B3⊕B4⊕B5**	B5
	B6	**B6⊕B7⊕B8**	B7	**B8**
	...	...		

    - En cas de panne d'un seul des disques (0, 1 ou 2), il est possible de récupérer les blocs.
    - Performances améliorées en écriture par rapport à RAID 2/3/4 : comme la parité est répartie sur les différents disques, il est possible d'effectuer simultanément certaines opérations d'écriture, par exemple : B3 (disques 0 et 2) et B8 (disques 1 et 3).

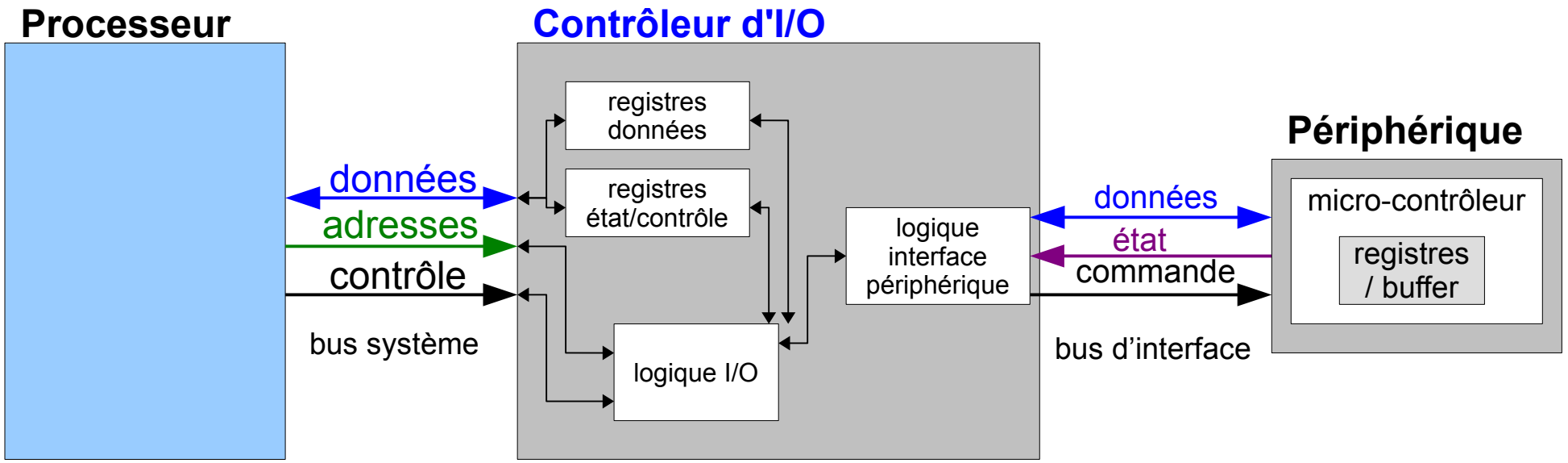
Table des Matières

- Introduction
- Types de périphériques
- **Contrôleur d'entrées/sorties**
- **Opérations d'entrées/sorties**
- **Etude de cas**

- **Introduction**
  - **Contrôleur d'I/O**
    - Les périphériques décrits à la section précédente ont tous un point commun: ils sont équipés d'une électronique de commande (intégrant généralement un micro-contrôleur) et communiquent avec le processeur via une interface particulière (PS/2, USB, SATA, …).
    - Le **contrôleur d'entrées / sorties** (*I/O controller*) est l'intermédiaire entre le processeur et un périphérique. Il est généralement connecté au processeur via le bus du système.

Interconnexion des Périphériques

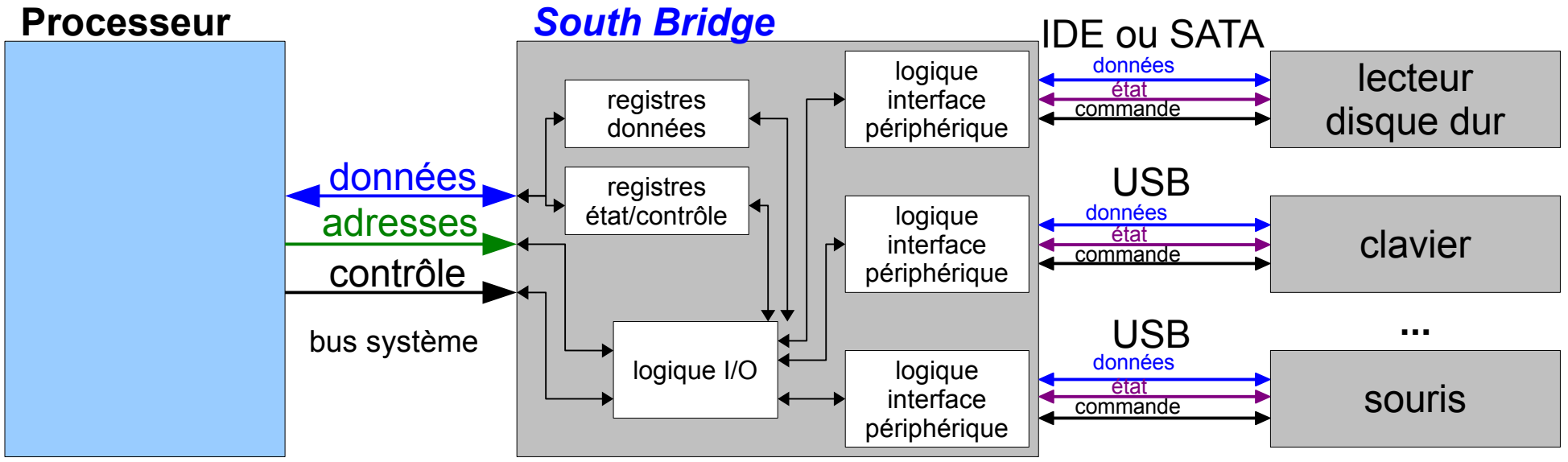
- **Principe**
  - **Contrôleur d'I/O**
    - Le contrôleur d'entrées/sorties permet au processeur
      - d'écrire une donnée vers un périphérique
      - de lire une donnée à partir d'un périphérique
      - de vérifier l'état d'un périphérique (prêt à recevoir des données ? prêt à transmettre des données ?)



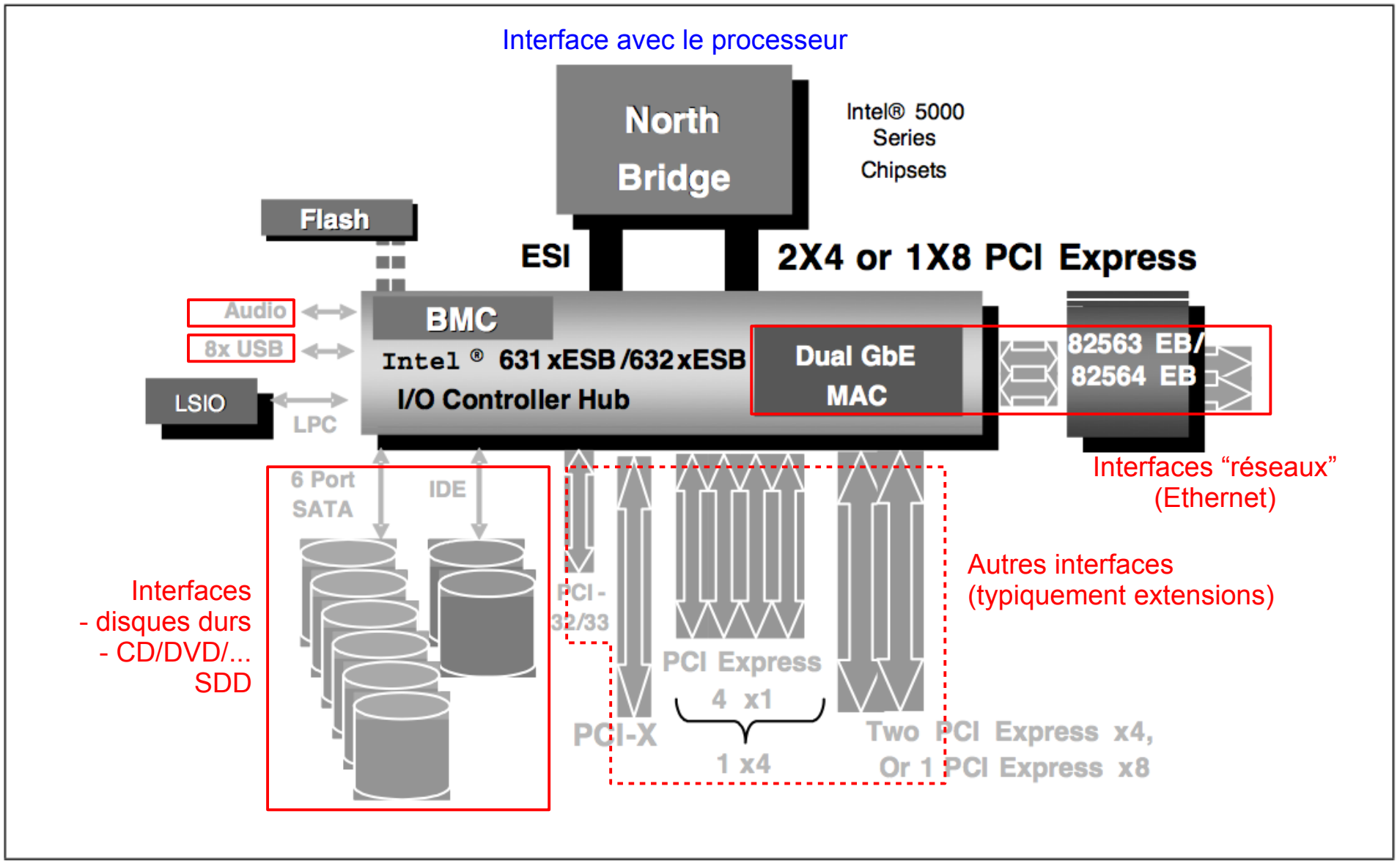
Note : Inspiré de **Computer Organization and Architecture – Designing for Performance**, 8th Edition, W. Stallings, Pearson, 2010.

Interconnexion des Périphériques

- **Principe**
  - **Contrôleur d'I/O**
    - Les ordinateurs récents contiennent souvent un unique **contrôleur d'I/O générique** (*I/O controller hub*) capable de supporter un grand nombre d'interfaces différentes avec les périphériques.
    - Le contrôleur d'I/O cache au processeur une partie des interactions avec les différentes interfaces. Ce contrôleur d'I/O est parfois appelé le ***South Bridge***.

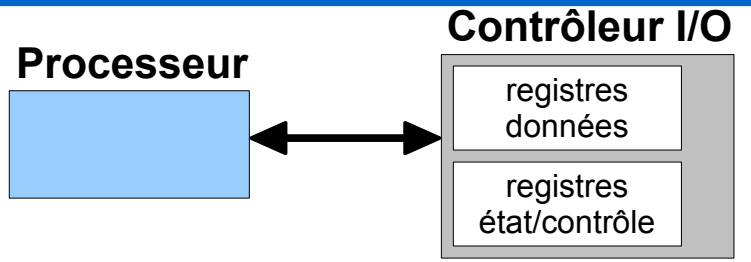


Interconnexion des Périphériques



Source : **Intel 631xESB/632xESB I/O Controller Hub Datasheet**, Intel Corporation, May 2006 (https://www.intel.com/content/dam/doc/datasheet/631xesb-632xesb-io-controller-hub-datasheet.pdf).

Interconnexion des Périphériques



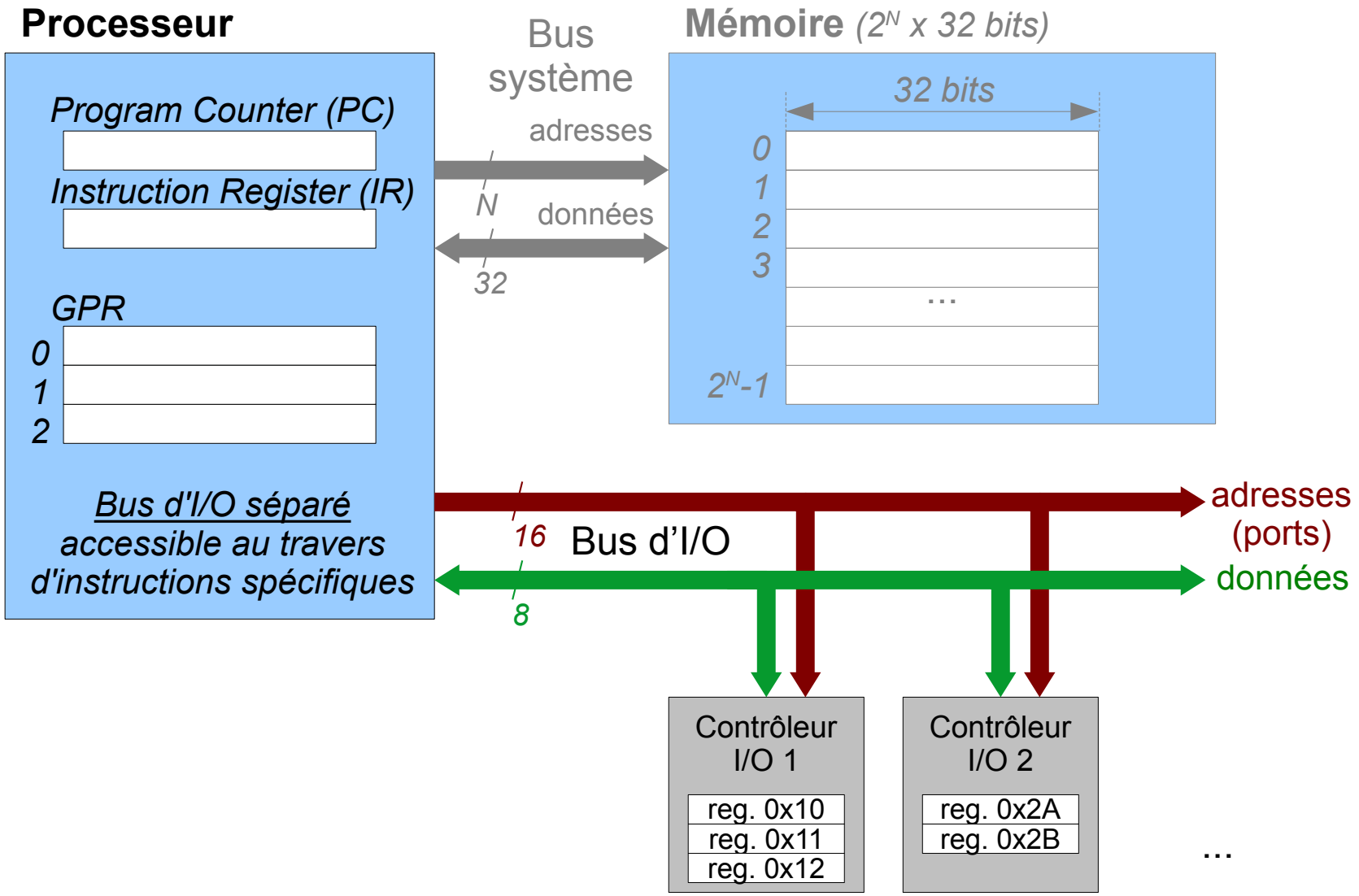
- **Définition**
  - **Interconnexions typiques**
    - Les registres d'un contrôleur d'I/O sont accessibles à partir du processeur
    - soit via un **bus d'I/O séparé** (*isolated I/O*)
      - Sur ce bus, des adresses appelées numéro de port sont utilisées pour indiquer à quel registre le processeur accède.
      - Des instructions spécifiques sont utilisées pour lire / écrire sur ce bus (p.ex. instructions spécifiques **`in`** et **`out`**, sur arch. Intel x86)
    - soit via le **bus mémoire**
      - Le contrôleur d'I/O est accessible via le même bus que la mémoire. Certaines plages d'adresses sont réservées aux registres du contrôleur d'I/O.
      - Les instructions de lecture/écriture en mémoire sont utilisées (p. ex: **`lw`** et **`sw`** en MIPS). On parle alors de ***memory-mapped* I/O** (MMIO).

Table des Matières

- Introduction
- Types de périphériques
- Contrôleur d'entrées/sorties
- **Isolated I/O**
  - Memory-mapped I/O
- **Opérations d'entrées/sorties**
- **Etude de cas**

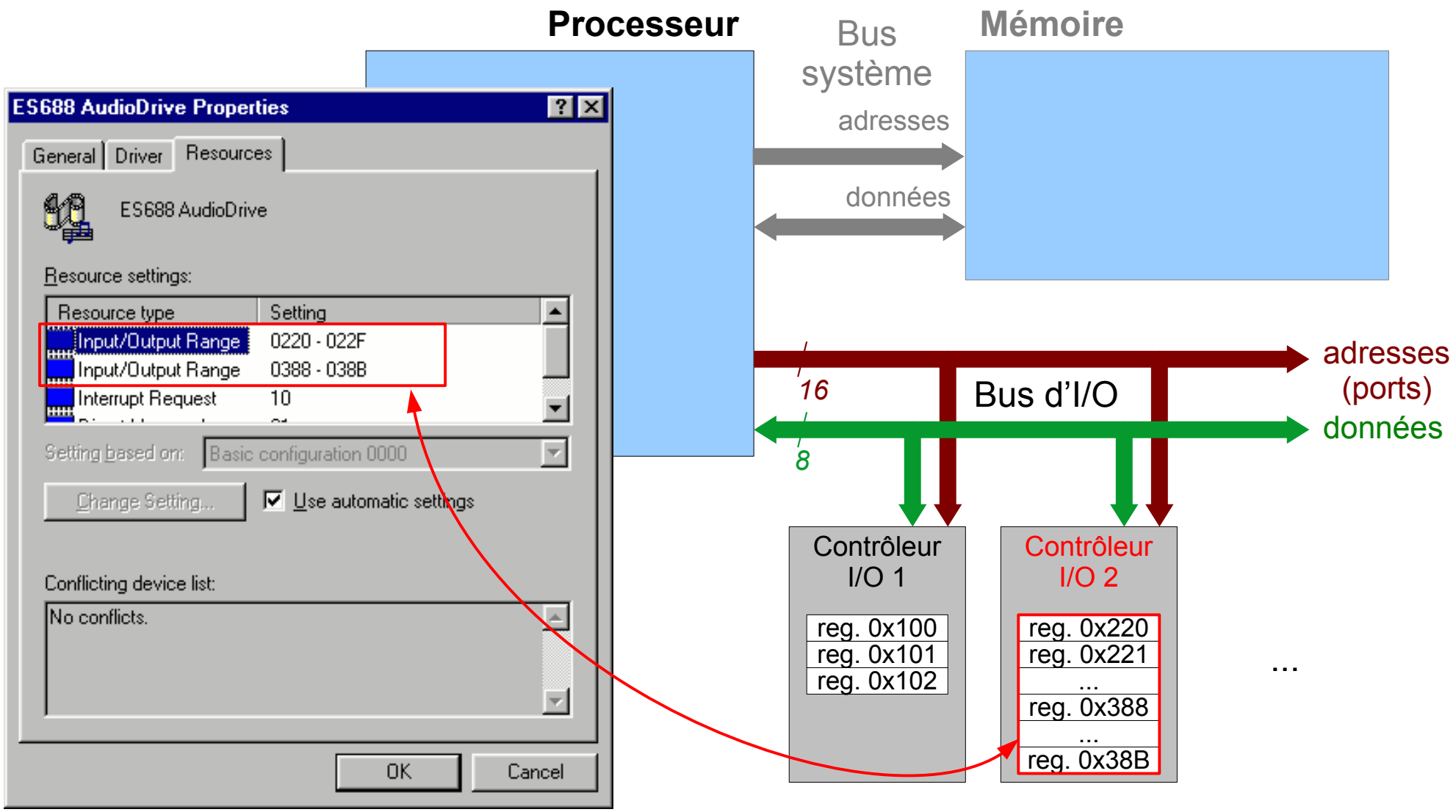
Interconnexion des Périphériques

- ***Isolated* I/O**

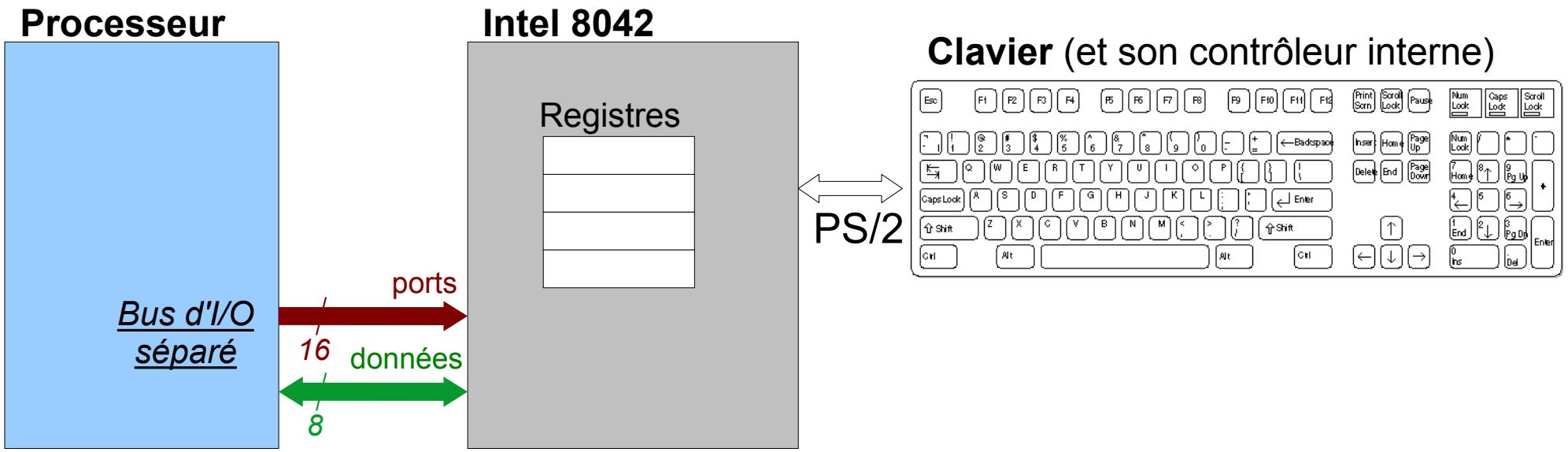


Interconnexion des Périphériques

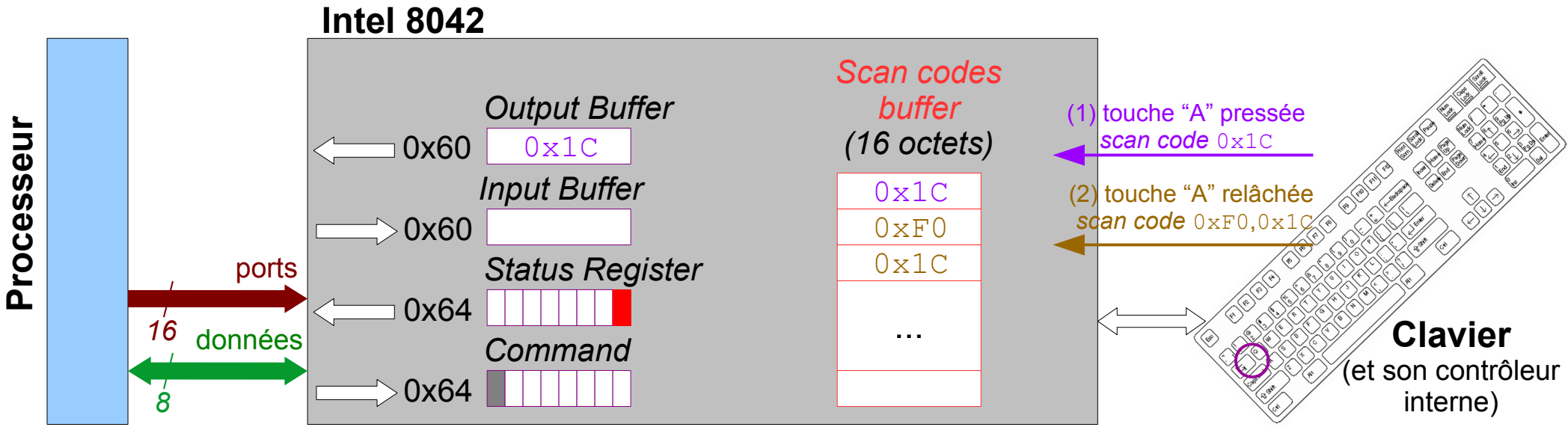
- ***Isolated* I/O**



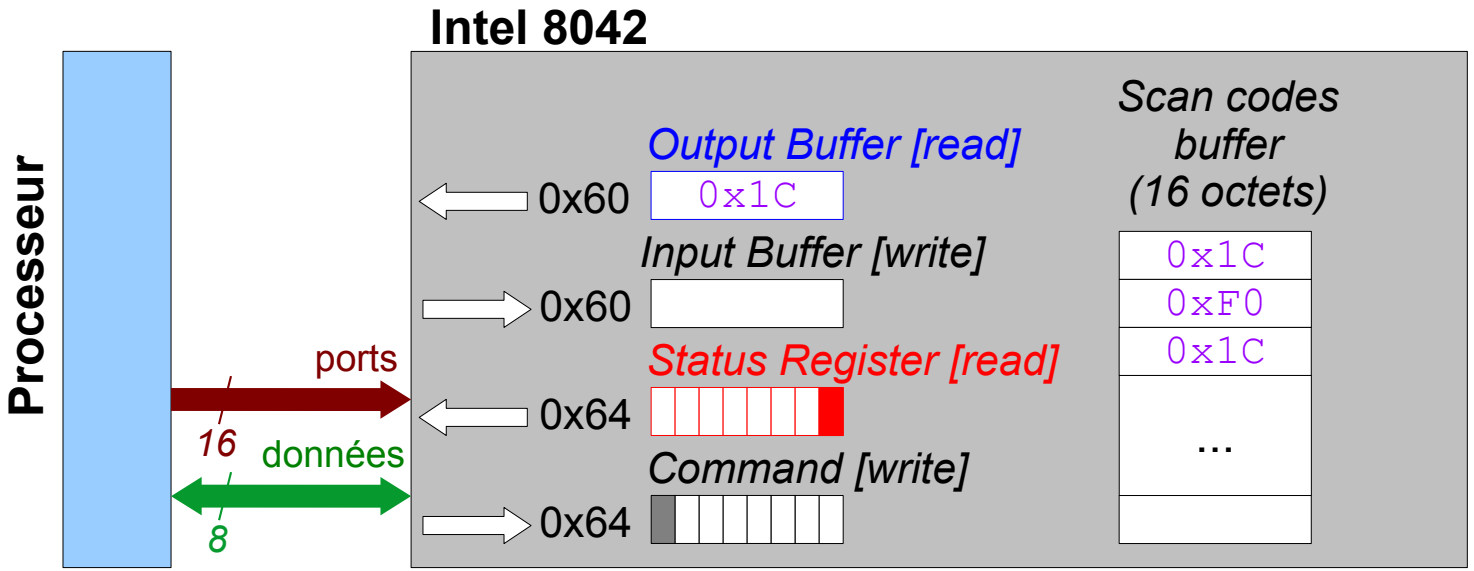
- **Exemple**
  - **Contrôleur de clavier**
    - Afin d'illustrer le rôle d'un contrôleur de périphérique, nous prenons l'exemple simple du contrôleur de clavier AT ou PS/2 utilisé sur les PC durant de longues années.
      - Ce contrôleur était basé sur un micro-contrôleur Intel 8042 (et plus tard sur des clones ou des équivalents intégrés au *chipset*)
      - Ce contrôleur se charge des communications avec le contrôleur interne du clavier.

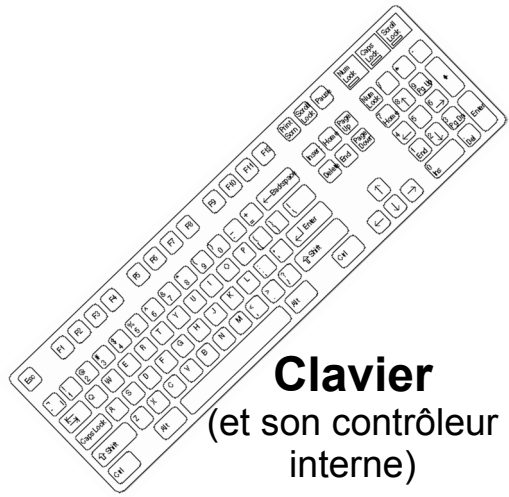


- **Exemple**
  - **Contrôleur de clavier**
    - Mémoire tampon (*scan codes buffer*) contenant les codes de touche (*scan codes*) reçus du clavier. Cette mémoire n'est pas accessible directement par le processeur.
    - 4 “registres” de 8 bits permettant de récupérer les données du tampon, déterminer l'état du tampon et d'envoyer des commandes au clavier. Ces registres sont accessibles via un bus d'I/O séparé.



- **Exemple**
  - **Contrôleur de clavier**
    - Le registre de lecture (*output buffer*) sur le port 0x60 permet de lire les *scan codes* reçus du clavier.
    - Le registre de statut (*status register*) sur le port 0x64 peut seulement être lu. Il contient 8 bits (*flags*). En particulier, le bit 0 indique s'il y a des *scan codes* à lire via l'*output buffer* (1 = *scan code* présent / 0 = rien à lire).





- **Exemple**
  - **Contrôleur de clavier**
    - Exemple de programme en langage C destiné à lire les codes de touches pressées sur le clavier.
      1. Attendre qu'un *scan code* soit disponible dans le tampon en regardant la valeur du bit 0 du *status register* (adresse `0x64`).
      2. Lire le *scan code* à partir de l'*output buffer* (adresse `0x60`).

```c
/* Lire le Status Register
   jusqu'à ce que bit 1 non nul */
do {
  status= IO_port_read(0x64);
} while ((status & 0x01) == 0);

/* Lire l'Output Buffer */
key_code= IO_port_read(0x60);
…
```

```
kbd_read:
      in    al, 0x64
      and   al, 0x01
      jz    kbd_read

      in    al, 0x60
```

Exemple de programme
en langage d'assemblage x86

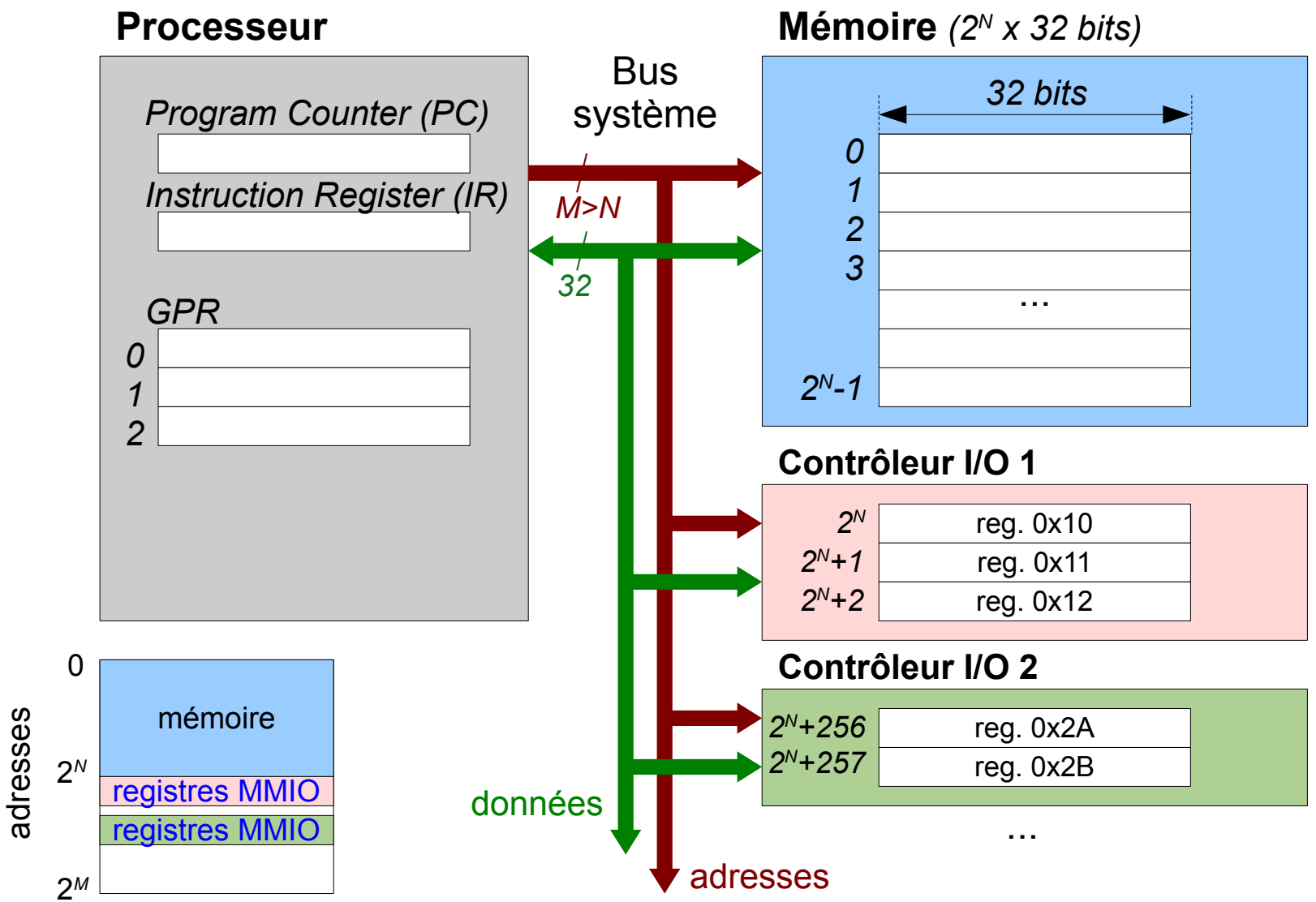
Note : "**in**" est une instruction spéciale de lecture I/O
"**jz**" effectue un branchement si le résultat précédent vaut 0
"**al**" est un registre GPR

Table des Matières

- Introduction
- Types de périphériques
- Contrôleur d'entrées/sorties
  - Isolated I/O
  - **Memory-mapped I/O**
- **Opérations d'entrées/sorties**
- **Etude de cas**

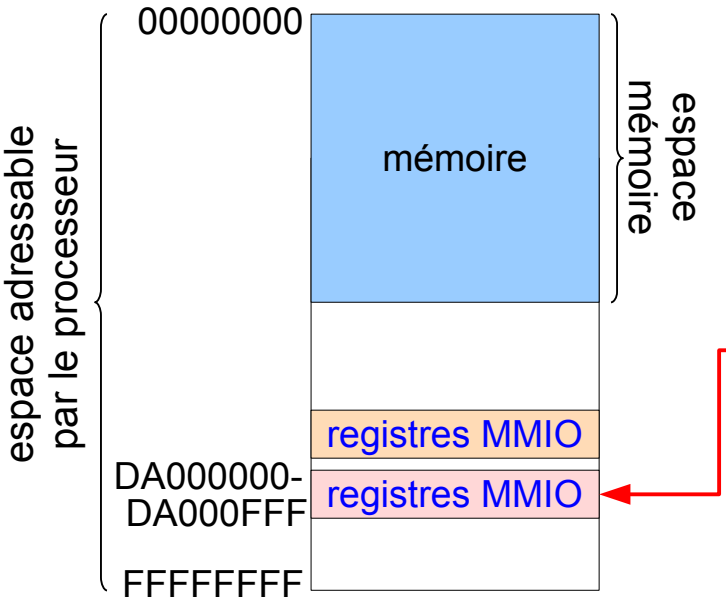
Interconnexion des Périphériques

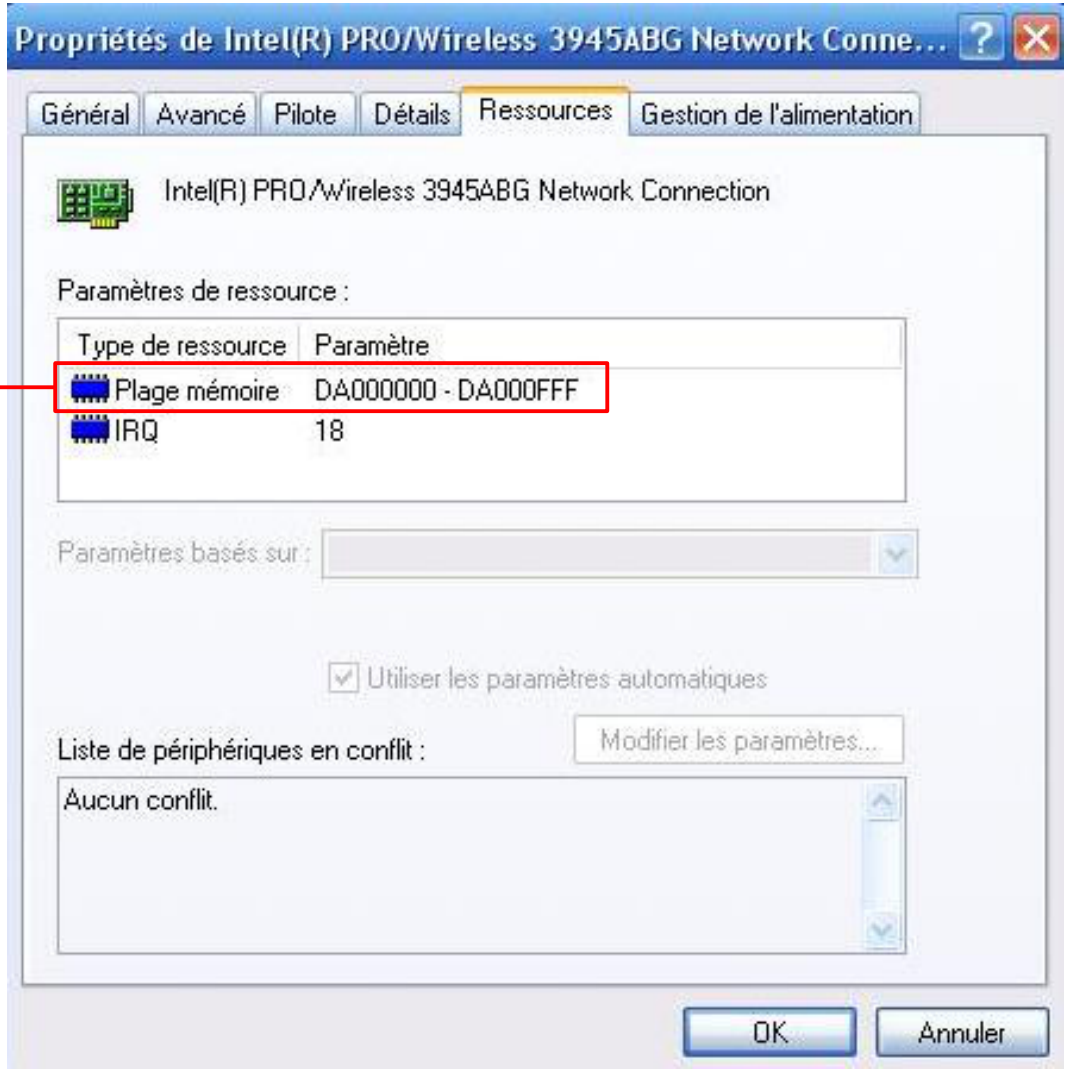
- ***Memory-mapped* I/O (MMIO)**



Interconnexion des Périphériques

- ***Memory-mapped* I/O (MMIO)**

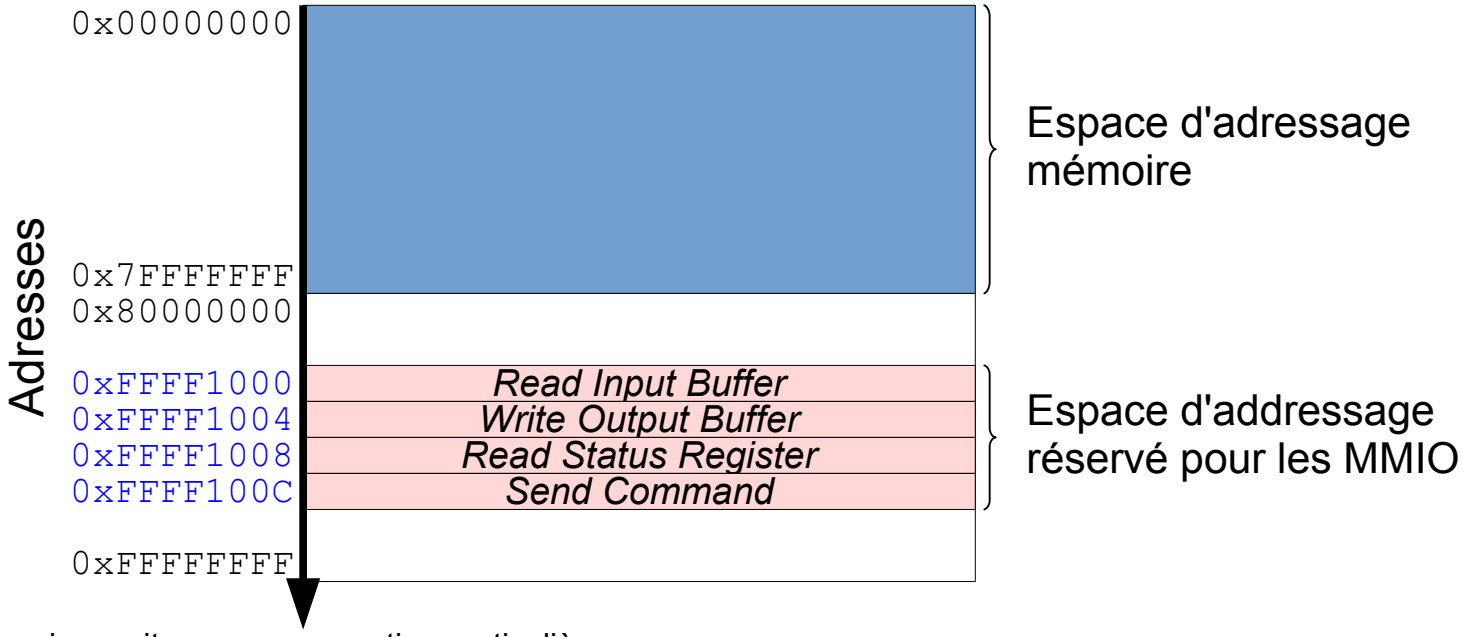




espace d'adresses
pour les registres
dédiés aux entrées/sorties

(les adresses sont allouées
automatiquement par le BIOS
ou par le système d'exploitation
- *PCI bus enumerator*)

- **Exemple**
  - **Contrôleur de clavier - MMIO**
    - Supposons que le contrôleur de clavier soit plutôt connecté au bus système et accédé via des *memory-mapped* I/O, il faudrait que
      - les registres du contrôleur aient la largeur du bus de données (p.ex. 32 bits)
      - chaque registre ait une adresse “mémoire” différente.



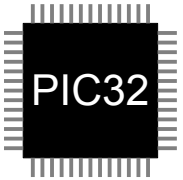
Note: Il s'agit d'un exemple qui ne suit aucune convention particulière.

Interconnexion des Périphériques

- **Exemple**
  - **Memory-mapped I/Os avec SPIM**
    - Le simulateur SPIM émule le fonctionnement de deux périphériques contrôlés par des registres “mappés” en mémoire.
    - L'étudiant curieux est invité à consulter l'annexe B de l'ouvrage de référence pour plus d'informations.

Interconnexion des Périphériques

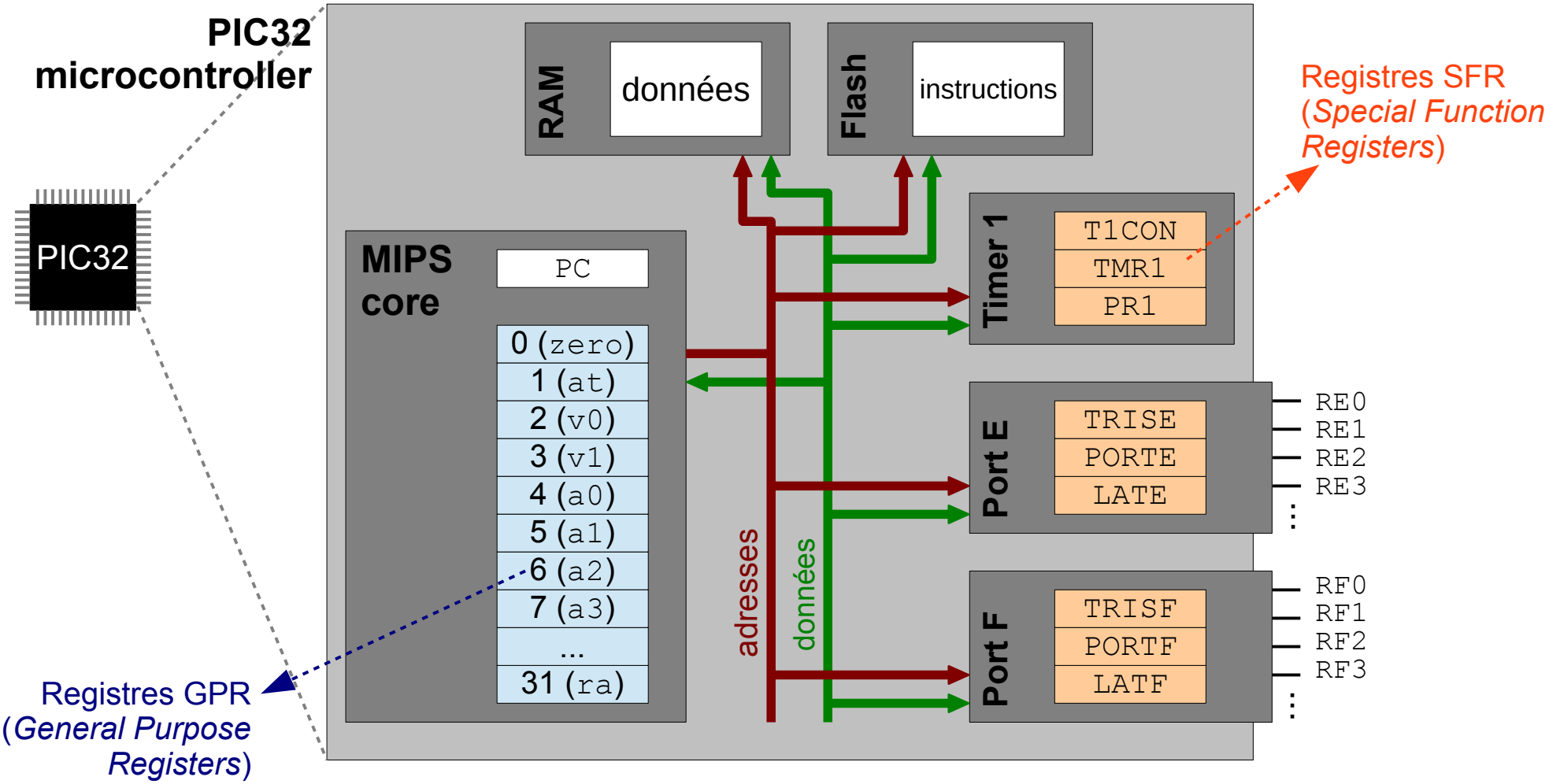
- **TP**
  - **Memory-mapped I/Os avec PIC32**



    - Durant les travaux pratiques, nous aurons l'occasion de “jouer” avec un microcontrôleur PIC32 de Microchip.
    - Ce microcontrôleur contient
      - un coeur d'architecture MIPS 32
      - de la mémoire flash (programme)
      - de la mémoire RAM (données)
      - de nombreux périphériques.
    - Chacun des périphériques est contrôlé par un ou plusieurs registres (SFR – *special function register*).
      - Les SFRs sont accédés via des instructions mémoire (**`lw`** et **`sw`**).
      - A chaque SFR est assignée une adresse mémoire.

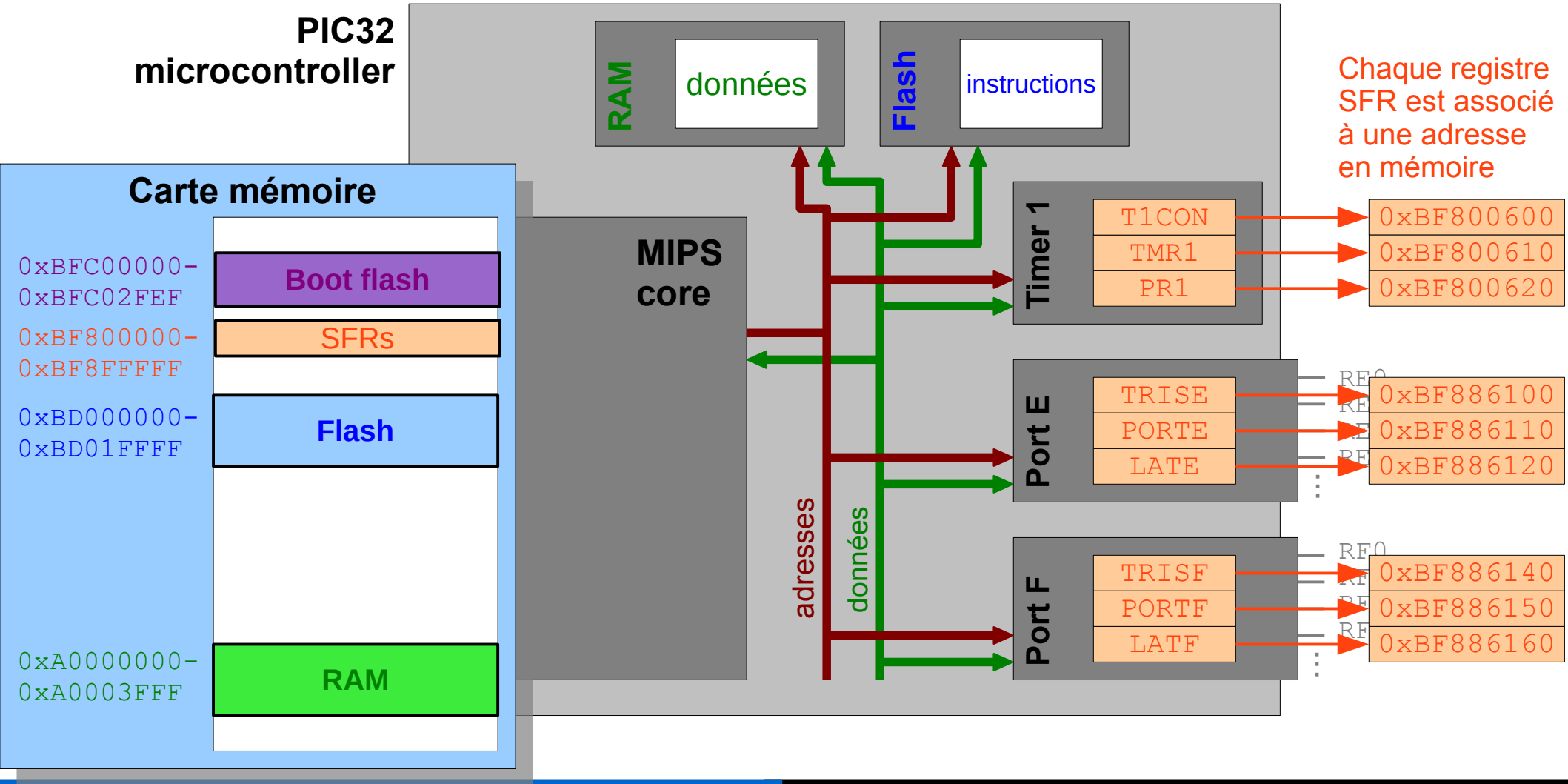
Interconnexion des Périphériques

- **Exemple**
  - **Memory-mapped I/Os avec PIC32**



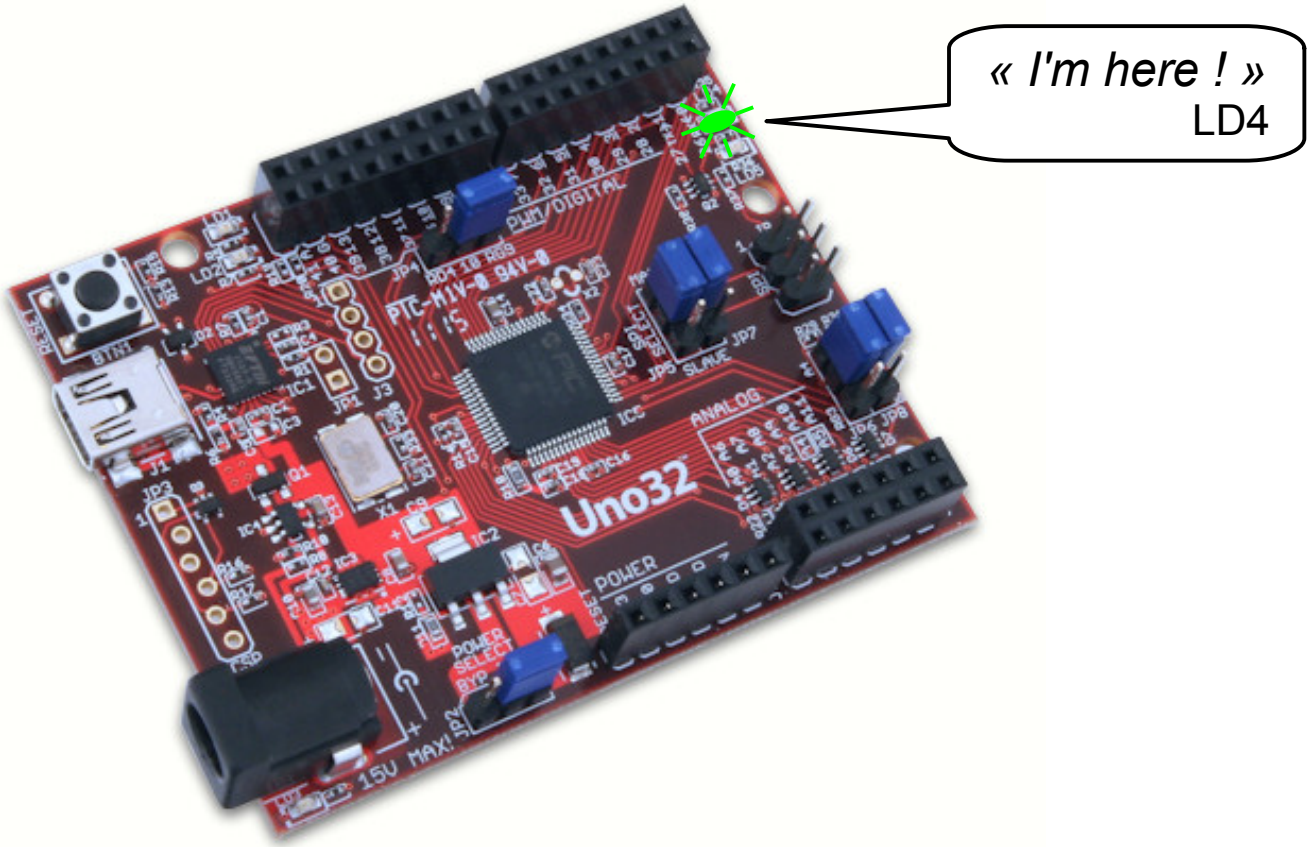
Interconnexion des Périphériques

- **Exemple**
  - **Memory-mapped I/Os avec PIC32**

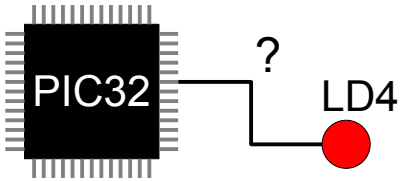


Interconnexion des Périphériques

- **TP – Faire clignoter une LED**

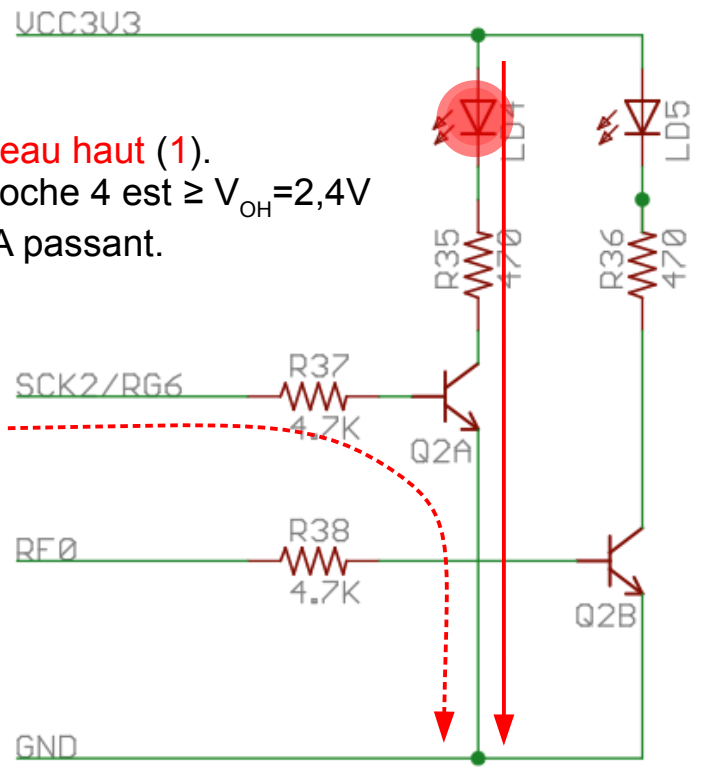


- **TP – Faire clignoter une LED**
  - La LED est reliée à la **broche (*pin*) 4** du PIC32, c-à-d. au port **RG6**.

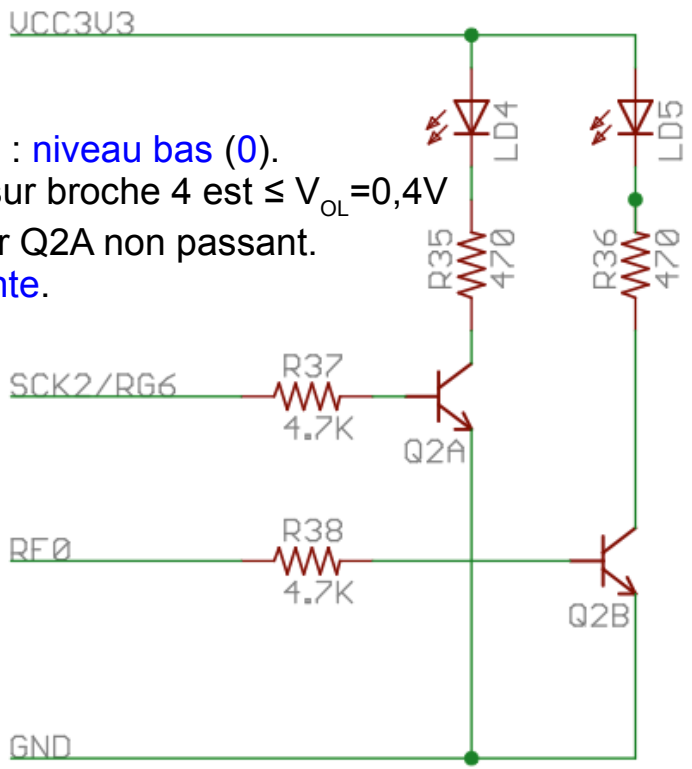


Port RG6 : niveau haut (1).
Tension sur broche 4 est ≥ V_{OH}=2,4V
Transistor Q2A passant.
LED allumée.

`pin 4 (RG6)` du PIC32

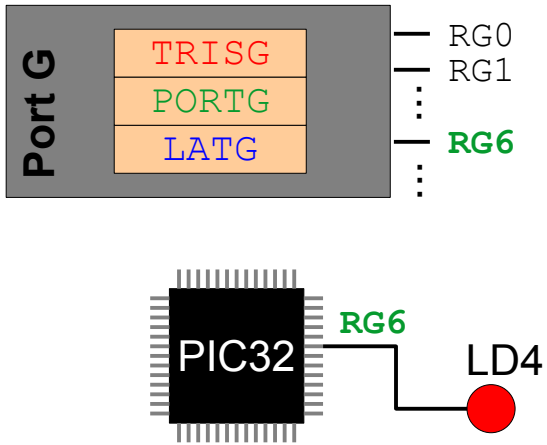


Port RG6 : niveau bas (0).
Tension sur broche 4 est ≤ V_{OL}=0,4V
Transistor Q2A non passant.
LED éteinte.



Source : ChipKIT-Uno32 schematics (rev. C), Digilent Inc., 8/6/2012

- **TP – Faire clignoter une LED**
  - Un port d'I/O du PIC32 regroupe plusieurs broches (*pins*). Par exemple, la broche **RG6** est la 6ème broche du **Port *G***.
  - Un port d'I/O est contrôlé par plusieurs registres.
    - **TRISG** : contrôle la direction des broches du port *G*.
      - broche correspondante : bit à 0 = sortie ; bit à 1 = entrée.
    - **PORTG** : écrit ou lit l'état des broches du port *G*.
    - **LATG** : en lecture, valeur écrite sur le port *G* ; en écriture idem **PORTG**.

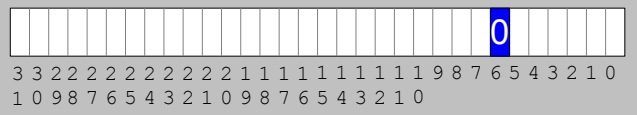


- **TP – Faire clignoter une LED**
  - Configuration du port **RG6** en sortie

```
TRISG=0xBF886180

      li    $a2, TRISG
      lw    $a0, 0($a2)
      li    $a1, 0xFFFFFFBF
      and   $a0, $a0, $a1
      sw    $a0, 0($a2)
```

Explications :

1. Lire la valeur actuelle de TRISG. Les bits que l'on ne veut pas changer doivent rester intacts.



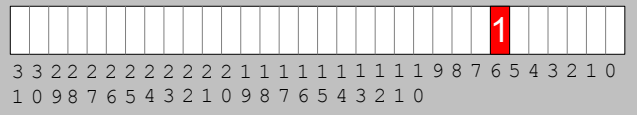
2. Mettre le bit TRISG6 à 0 (→ sortie)
3. Ecrire la nouvelle valeur de TRISG.

  - Passer **RG6** à l'état haut = allumer la LED

```
LATG=0xBF8861A0

      li    $a2, LATG
      lw    $a0, 0($a2)
      li    $a1, 0x00000040
      or    $a0, $a0, $a1
      sw    $a0, 0($a2)
```

Explications :

1. Lire la valeur actuelle de LATG.



2. Mettre le bit LATG6 à 1 (état haut)
3. Ecrire la nouvelle valeur de LATG.

Table des Matières

- Introduction
- Types de périphériques
- Contrôleur d'entrées/sorties
- **Opérations d'entrées/sorties**
- **Etude de cas**

Opérations d'entrées/sorties

- **Introduction**
  - **Opérations d'entrées/sorties**
    - Il existe différents moyens d'effectuer des opérations d'entrées/sorties.
      - Entrées/sorties programmées (*programmed I/O* – **PIO**)
      - Interruptions (interrupts)
      - Transferts directs en mémoire (*Direct Memory Access* – **DMA**)

Table des Matières

- Introduction
- Types de périphériques
- Contrôleur d'entrées/sorties
- **Opérations d'entrées/sorties**
  - ⟹ Entrées/sorties programmées (PIO)
  - Interruptions
  - *Direct Memory Access* (DMA)
- **Etude de cas**

Entrées / Sorties Programmées

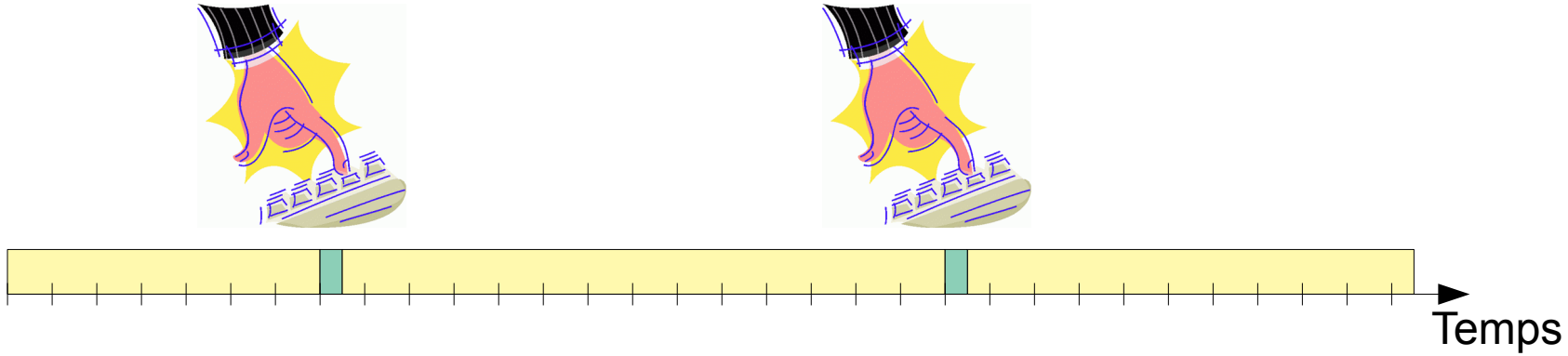
- **Principe**
  - **Entrées/sorties programmées**
    - Le processeur contrôle complètement l'opération d'entrée/sortie. Il effectue tout le travail :
      - envoi commande (p.ex. lecture ou écriture)
      - interrogation de l'état
      - transfert des données
    - Le périphérique est interrogé (*polling*) en lisant ses registres d'état afin de déterminer si le périphérique
      - est prêt pour la prochaine opération d'entrée/sortie. Lorsque le périphérique est prêt, l'opération d'entrée/sortie est réalisée.
      - a terminé l'opération courante. Lorsque le périphérique a terminé une opération, un éventuel code d'erreur peut être lu et l'opération suivante peut être entamée.

- **Exemple**
  - **Entrées/sorties programmées**
    - Exemple : la lecture des *scan codes* à partir du contrôleur clavier présentée plus tôt dans ce chapitre est un exemple d'entrée/sortie programmée.

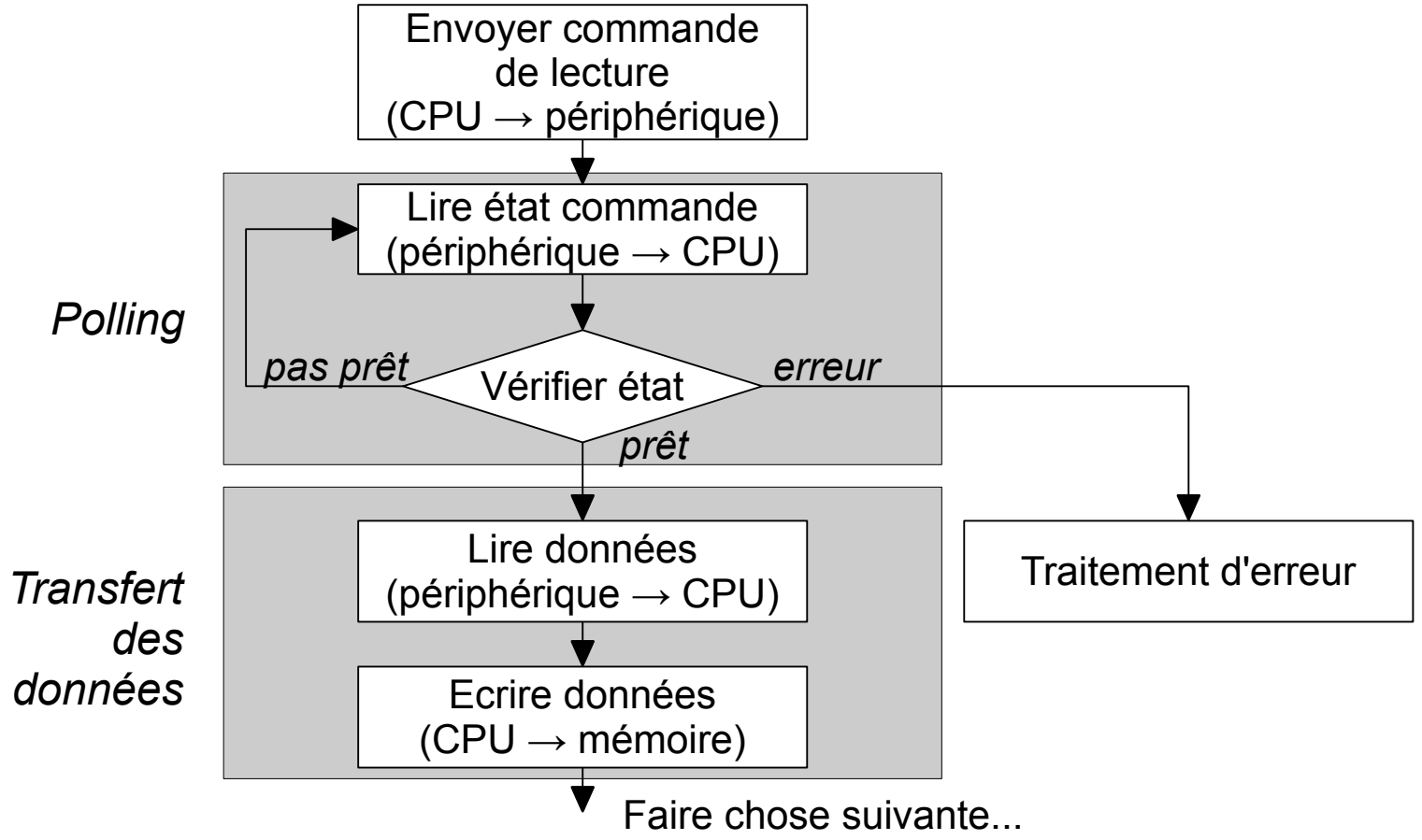
```
kbd_read:
      in      al, 0x64
      and     al, 0x01
      jz      kbd_read

      in      al, 0x60
```

Polling : boucle qui lit le registre d'état jusqu'à ce qu'il indique la disponibilité d'un *scan code*.

Lecture du *scan code*.



- **Exemple**
  - **Entrées/sorties programmées**
    - Lecture d'un bloc de données à partir d'un périphérique (p.ex. lecteur de disque dur) en utilisant des entrées/sorties programmées.



Note: Inspiré de **Computer Organization and Architecture – Designing for Performance**, 8th Edition, W. Stallings, Pearson, 2010.

Entrées / Sorties Programmées

- **Principe**
  - **Entrées/sorties programmées**
    - Avantage
      - Méthode la plus simple
      - Processeur contrôle complètement l'opération
    - Inconvénients
      - Le processeur sert d'intermédiaire entre le contrôleur d'I/O et la mémoire.
      - Fonctionnement synchrone : il est possible que le processeur gaspille beaucoup d'instructions pour tester l'état du périphérique (p.ex. s'il n'y a pas de *scan code* à lire dans un contrôleur de clavier).

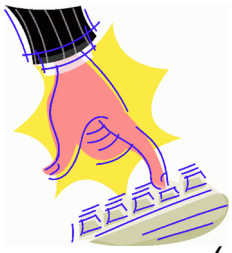
Table des Matières

- Introduction
- Types de périphériques
- Contrôleur d'entrées/sorties
- **Opérations d'entrées/sorties**
  - Entrées/sorties programmées (PIO)
  - Interruptions
  - *Direct Memory Access* (DMA)
- **Etude de cas**

Interruptions

- **Principe**
  - **Interruptions**
    - Les périphériques sont les mieux placés pour savoir quand leur état change. Ils **signalent** au processeur lorsqu'ils sont prêts pour une opération d'entrée-sortie ou lorsqu'ils ont terminé une opération.
    - Ce signal prend la forme d'une **requête d'interruption** (*interrupt request*, **IRQ**).
    - Lorsqu'une requête d'interruption est reçue
      1. Le processeur interrompt (suspend) le programme actuellement en cours d'exécution
      2. Le processeur effectue un branchement vers une **routine de traitement d'interruption** (interrupt service routine, **ISR**)
      3. Le processeur rend éventuellement la main au programme précédent. Ce dernier poursuit alors son exécution à partir d'où il a été interrompu.

- **Exemple**
  - **Gestion des interruptions par le processeur**
    - Lorsque le processeur reçoit une requête d'interruption, il **déroute** l'exécution des instructions vers une **routine de traitement d'interruption**.



Programme en cours d'exécution

réception d'une **requête d'interruption** (p.ex. “*scan code*” disponible)

branchement

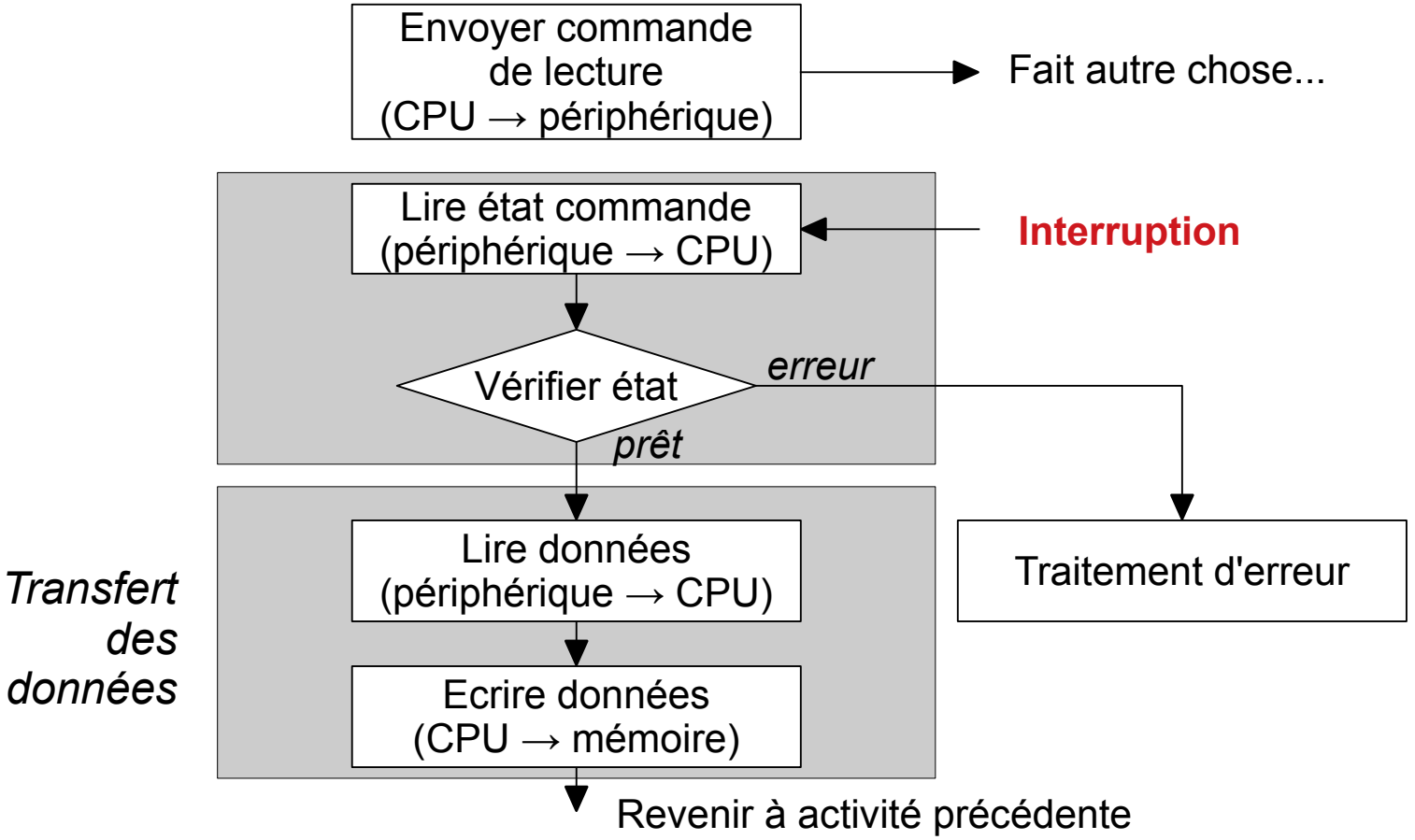
Gestionnaire d'interruption

Exécution des instructions du gestionnaire d'interruption (p.ex. lecture du “*scan code*” clavier)

retour

Poursuite de l'exécution du programme

- **Exemple**
  - **Interruptions**
    - Lecture d'un bloc de données à partir d'un périphérique (p.ex. lecteur de disque dur) en utilisant des interruptions.



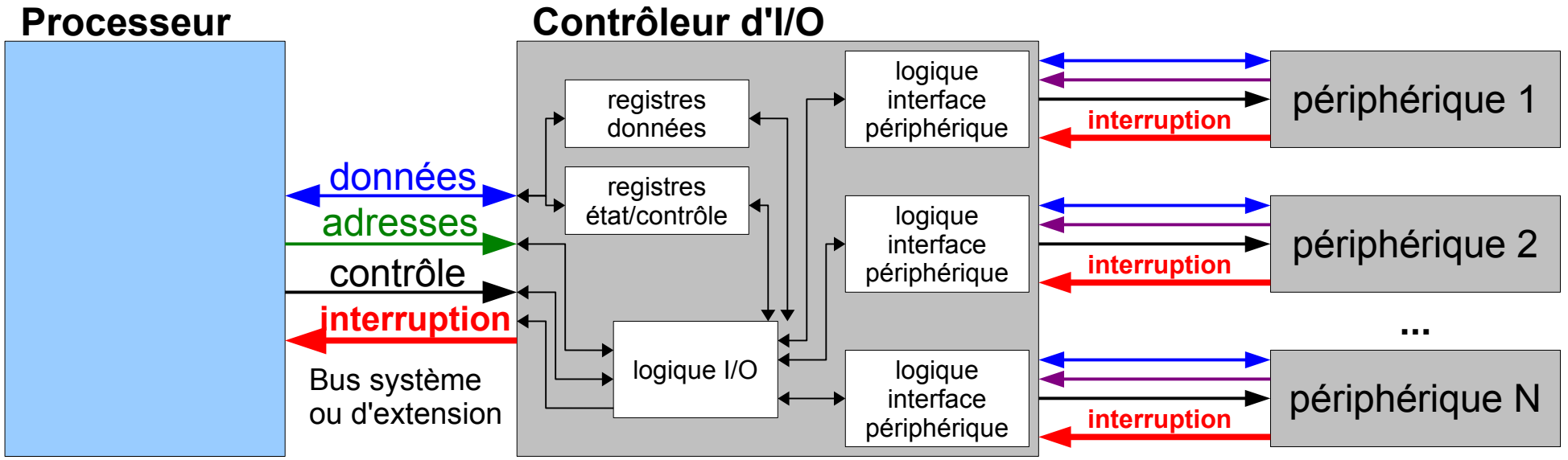
Note: Inspiré de **Computer Organization and Architecture – Designing for Performance**, 8th Edition, W. Stallings, Pearson, 2010.

Interruptions

- **Principe**
  - **Mécanismes nécessaires**
    - Le mécanisme d'interruption requiert plusieurs changements à l'architecture

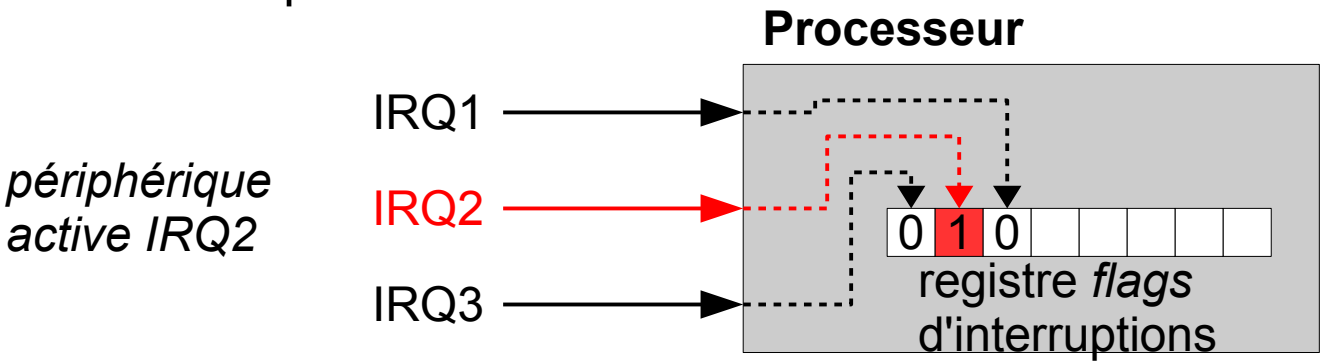
1. **Signal IRQ matériel** d'un périphérique vers le processeur (ligne d'interruption) – il peut y avoir plusieurs lignes d'interruptions distinctes (p.ex. un par périphérique)
2. **Mémorisation d'une IRQ** dans le processeur : bit / flag d'interruption – il peut y avoir plusieurs flags d'interruption distincts (p.ex. un par ligne d'interruption)
3. **Test** des flags d'interruption et **branchement automatique** vers la routine d'interruption associée.
4. **Sauvegarde et restauration** de l'état du programme interrompu. En particulier l'adresse de l'instruction en cours lors de l'interruption.

- **Principe**
  - **Signal IRQ matériel – ligne d'interruption**
    - Les opérations d'entrées/sorties par interruptions nécessitent un support matériel.
      - Les périphériques peuvent signaler un changement d'état au contrôleur d'I/O (ligne d'interruption)
      - Le contrôleur d'I/O peut signaler un changement d'état au processeur (ligne d'interruption).



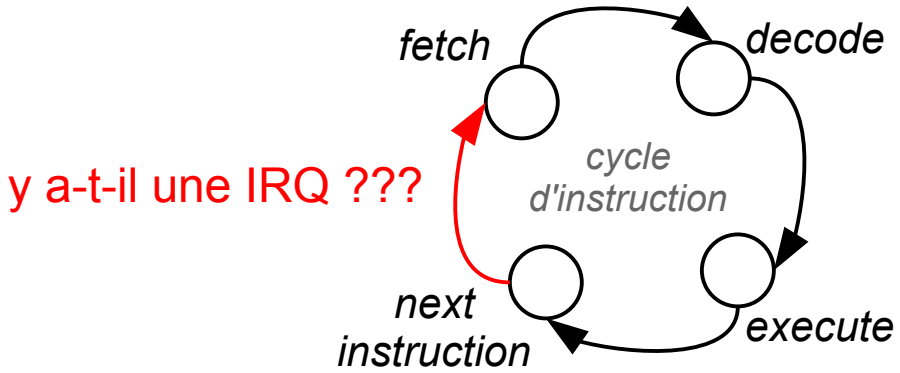
Note : Inspiré de **Computer Organization and Architecture – Designing for Performance**, 8th Edition, W. Stallings, Pearson, 2010.

- **Principe**
  - **Mémorisation des IRQs**
    - A chaque source (ligne) d'interruption, est associé un ***flag d'interruption*** (bit) mémorisant si une requête d'interruption (IRQ) a été reçue de cette ligne. Par exemple : bit à 0=pas d'IRQ ; bit à 1=IRQ.
    - Les flags d'interruption sont conservés dans des registres spéciaux du processeur.



    - Un flag d'interruption est ré-initialisé lorsque l'interruption a été traitée. Cela peut être réalisé automatiquement par le processeur ou manuellement par logiciel.

- **Principe**
  - **Traitement des IRQs – Test**
    - Le processeur **teste** régulièrement les *flags* d'interruption. Si un *flag* d'interruption est positionné, le processeur doit effectuer un branchement vers la routine d'interruption correspondante.
    - Ce test est effectué à chaque cycle d'instruction. En fonction de l'architecture, une requête d'interruption peut être prise en compte avant, après et/ou pendant l'exécution d'une instruction.



    - Le processeur teste les *flags* d'interruption dans un certain ordre. Cela donne implicitement une priorité plus grande aux IRQs testées en premier.

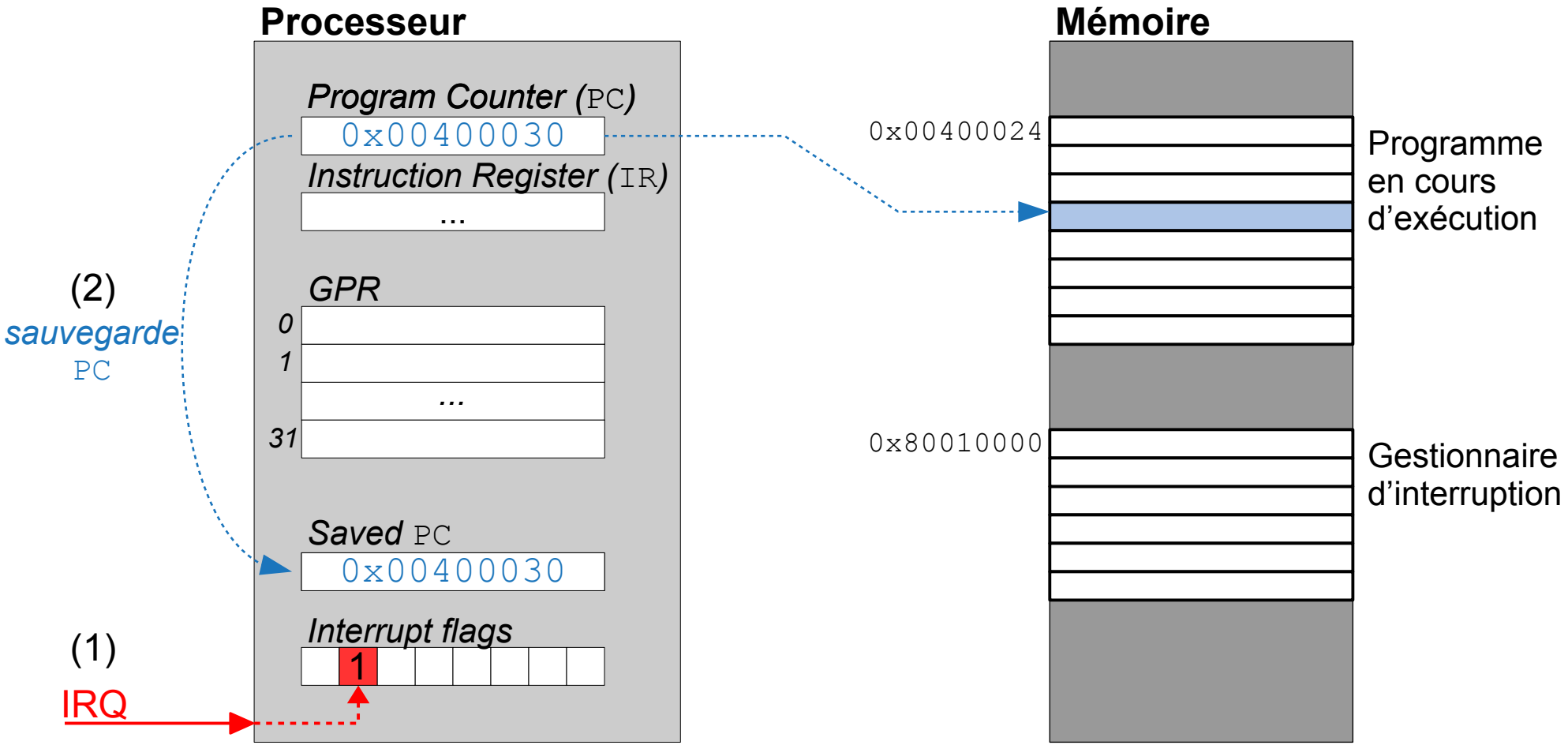
Interruptions

- **Principe**
  - **Traitement des IRQs – Branchement**
    - Lorsqu'un *flag* d'interruption est positionné, le processeur effectue un **branchement asynchrone(1) vers un gestionnaire d'interruption**.

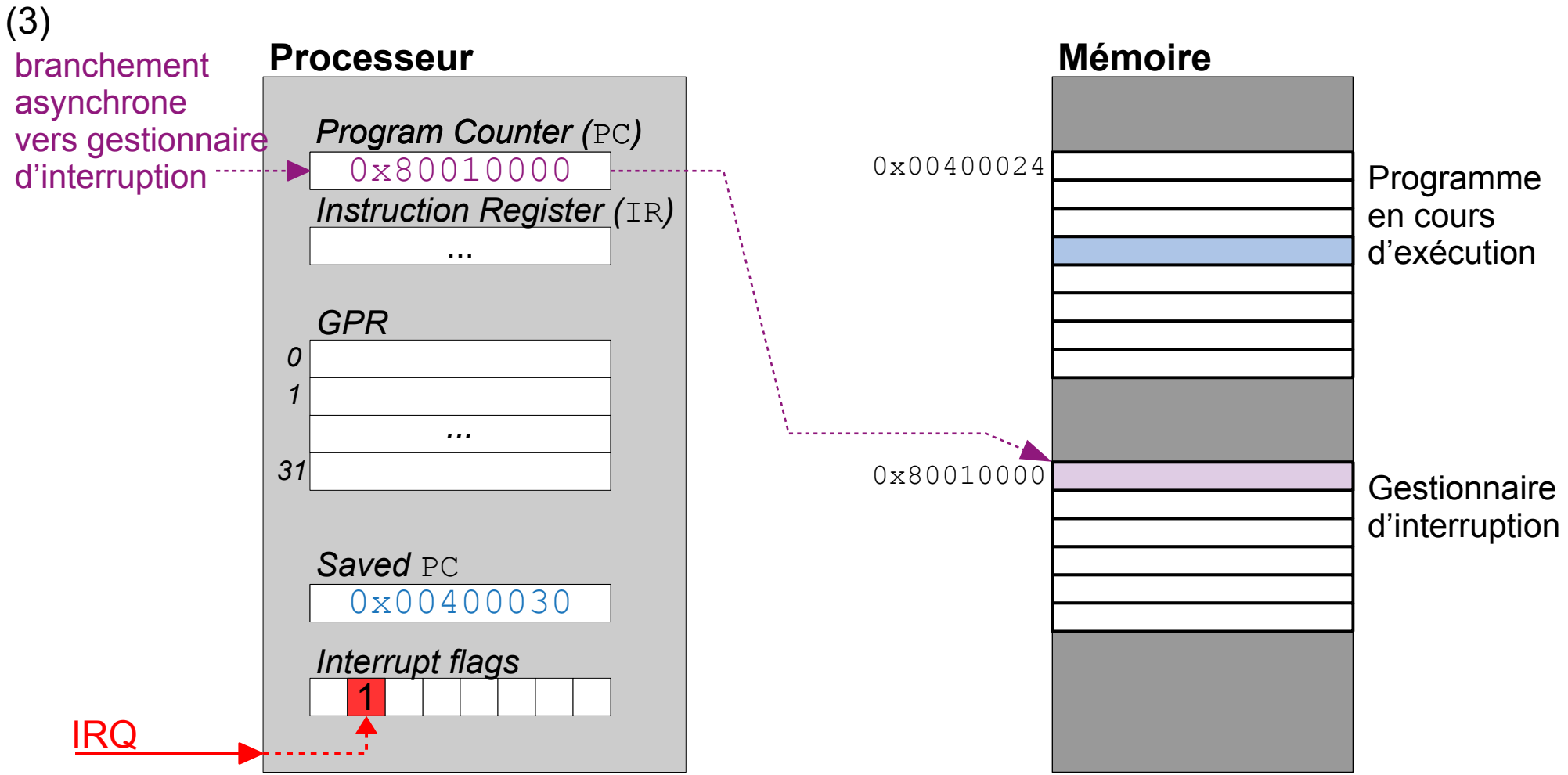
      1. Sauvegarde du Program Counter (PC) : afin de permettre au programme interrompu de continuer son exécution après le gestionnaire d'interruption, le registre PC est d'abord sauvegardé (typiquement dans un registre spécial ou sur la pile).
      2. Changement du PC : l'adresse du gestionnaire d'interruption est ensuite écrite dans le registre PC.
      3. Restauration du PC : lorsque le gestionnaire d'interruption est terminé, la valeur du registre PC est restaurée, ce qui permet au programme interrompu de continuer son exécution.

(1) Branchement asynchrone car il ne correspond pas à l'exécution d'une instruction de branchement.

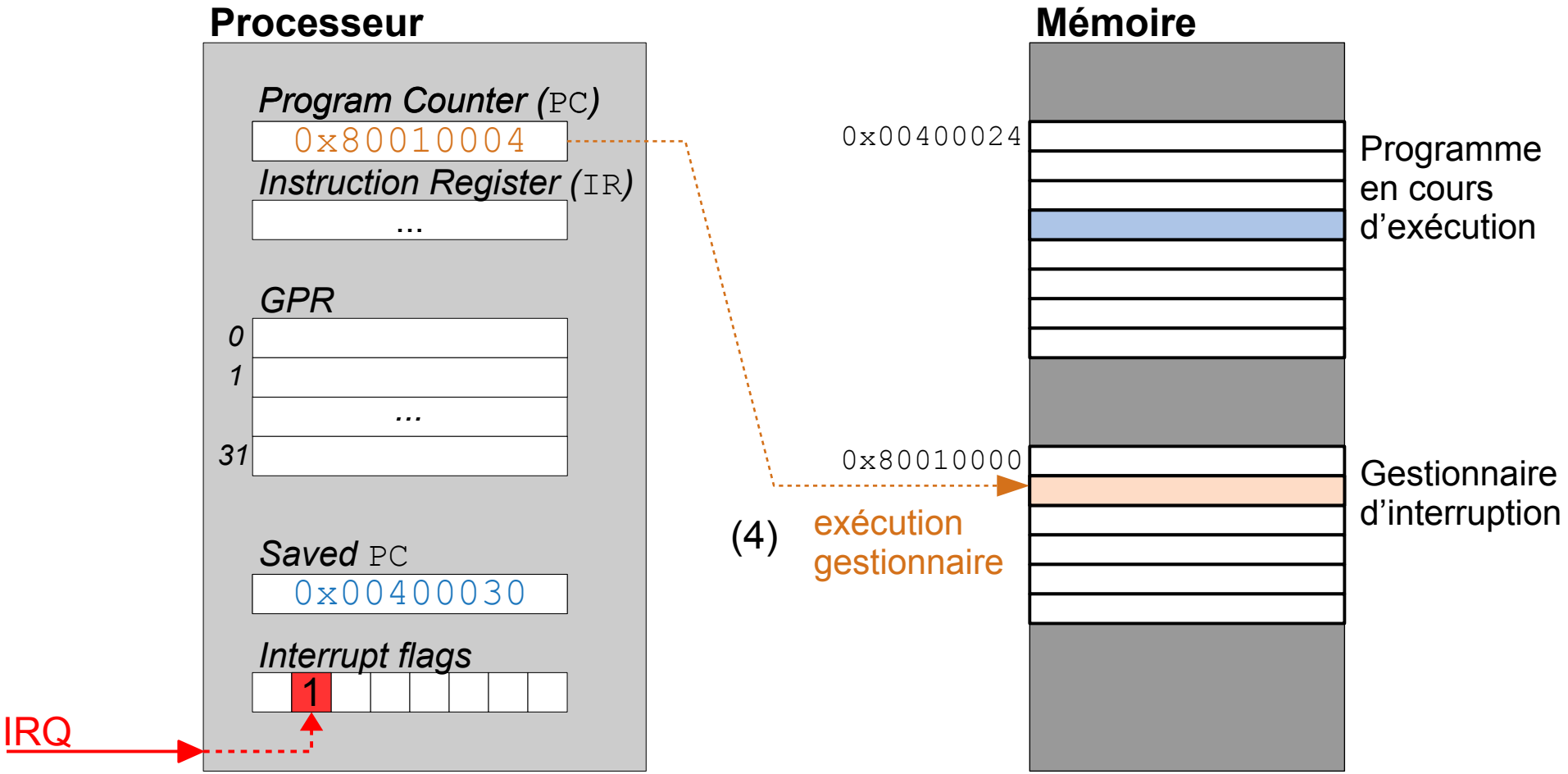
- **Principe**
  - **Traitement des IRQs – Branchement**



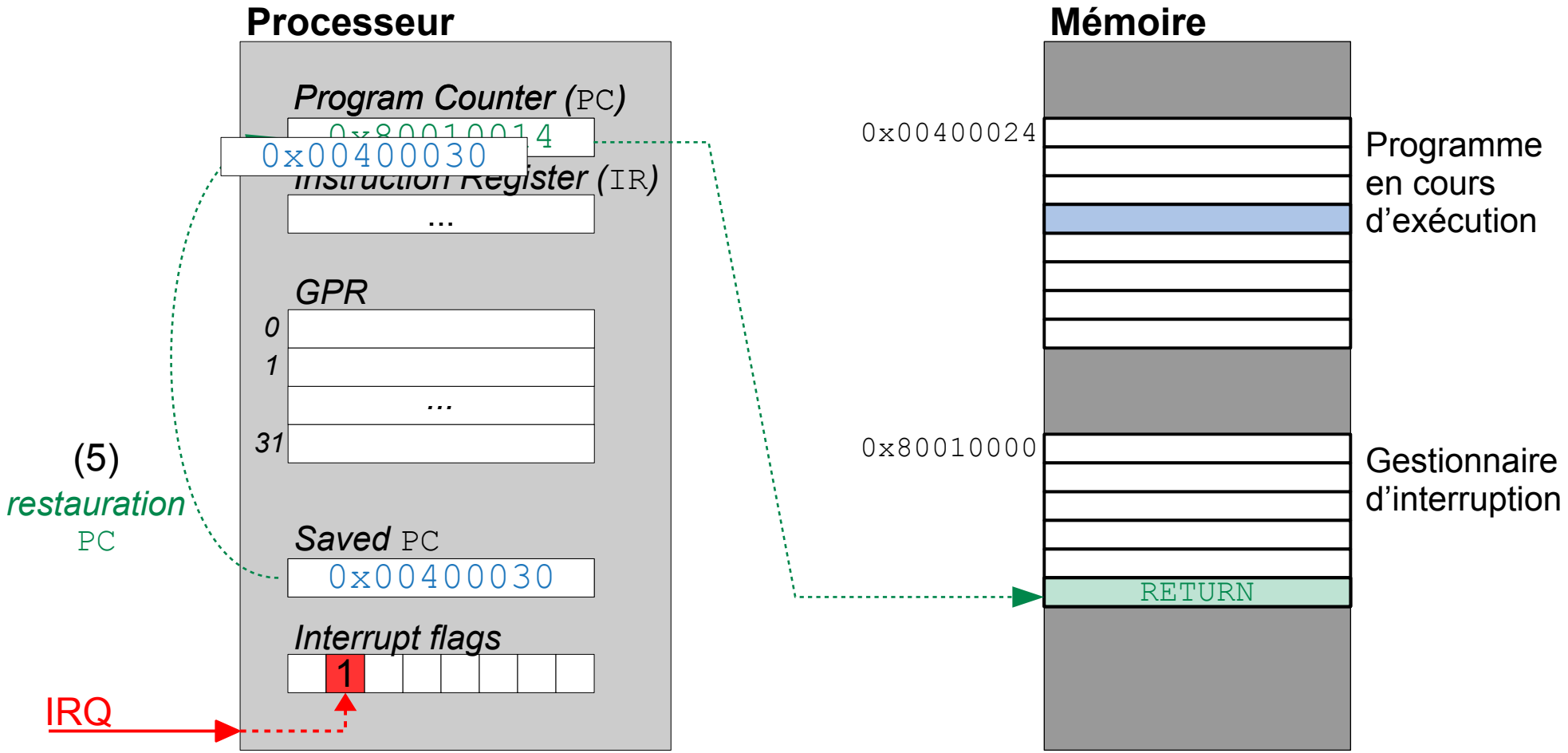
- **Principe**
  - **Traitement des IRQs – Branchement**



- **Principe**
  - **Traitement des IRQs – Branchement**



- **Principe**
  - **Traitement des IRQs – Branchement**

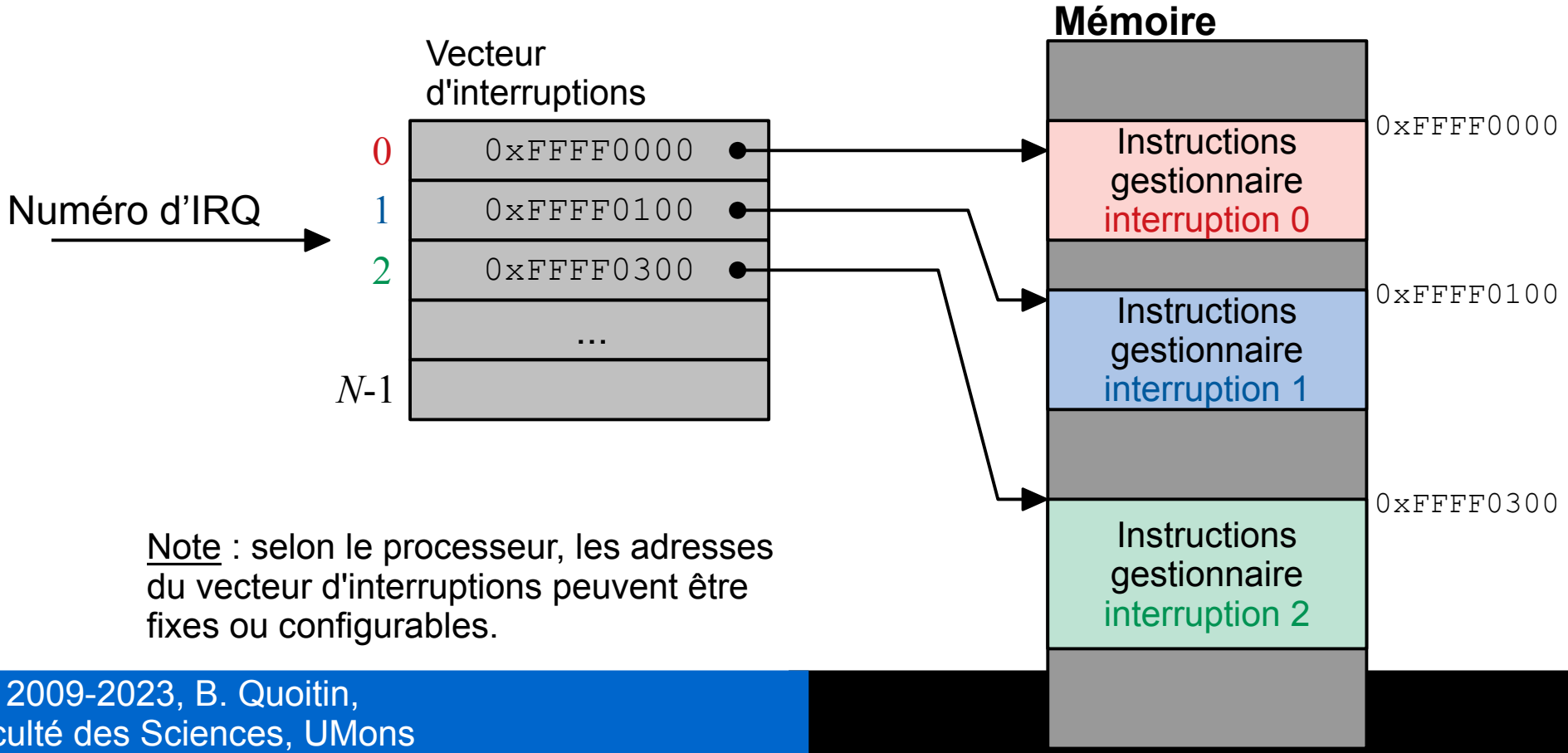


Interruptions

- **Principe**
  - **Traitement des IRQs – Branchement**
    - Il existe plusieurs moyens pour déterminer l'adresse de chaque gestionnaire d'interruption
      - Adresse fixée en “hardware” : pour une IRQ donnée, l'adresse du gestionnaire correspondant est toujours au même endroit en mémoire. Il faut donc s'arranger pour placer les instructions du gestionnaire d'interruption à cette adresse.
      - Adresse configurable : un registre permet de déterminer pour chaque IRQ l'adresse de son gestionnaire.
    - Si plusieurs sources d’interruptions (IRQ) sont possibles, certains processeurs utilisent un **vecteur d'interruptions** ou une **table d'exceptions**. L'adresse de branchement dépend alors du numéro de l’IRQ.

Interruptions

- **Principe**
  - **Vecteur / table d'interruptions**
    - Chaque source d'interruption possède un identifiant (un nombre).
    - En fonction de cet identifiant, une adresse de branchement différente est utilisée. Un vecteur d'interruptions peut être utilisé à cet effet.



Note : selon le processeur, les adresses du vecteur d'interruptions peuvent être fixes ou configurables.

Interruptions

- **Principe**
  - **Gestion des exceptions**
    - La gestion des interruptions fait partie d'un mécanisme plus large du processeur visant à gérer des évènements exceptionnels: les **exceptions**.
    - On distingue 3 classes d'exceptions
      - **Interruptions** (*interrupts*)
        - signal matériel (typiquement extérieur au processeur)
        - asynchrone, i.e. non demandé explicitement
      - ***Traps***
        - utilisés pour effectuer des appels aux procédures fournies par le système d'exploitation : appels systèmes (*system calls*)
        - synchrone, i.e. demandé explicitement par une instruction spéciale
      - **Erreurs** (*faults* et *aborts*)
        - instruction inconnue, division par 0, overflow lors d'une opération arithmétique, accès à une donnée non-alignée en mémoire, accès à une donnée hors mémoire
        - le gestionnaire peut ou pas corriger l'erreur

Interruptions

- **Principe**
  - **Interruptions**
    - Avantages
      - Mécanisme asynchrone : pas d'appel explicite à une procédure de gestion d'interruption, le branchement est effectué automatiquement en réponse à une IRQ.
      - Traitement uniquement lorsque nécessaire (au contraire du *polling*)
    - Inconvénients
      - Implémentation plus complexe que PIO : nécessite des modifications aux périphériques, au processeur et au logiciel
      - Implique une éventuelle surcharge due au changement de contexte (sauvegarde + restauration de l'état du programme interrompu)

Table des Matières

- Introduction
- Types de périphériques
- Contrôleur d'entrées/sorties
- **Opérations d'entrées/sorties**
  - Entrées/sorties programmées (PIO)
  - Interruptions
  - *Direct Memory Access* (DMA)
- **Etude de cas**
  - Interruptions avec MIPS

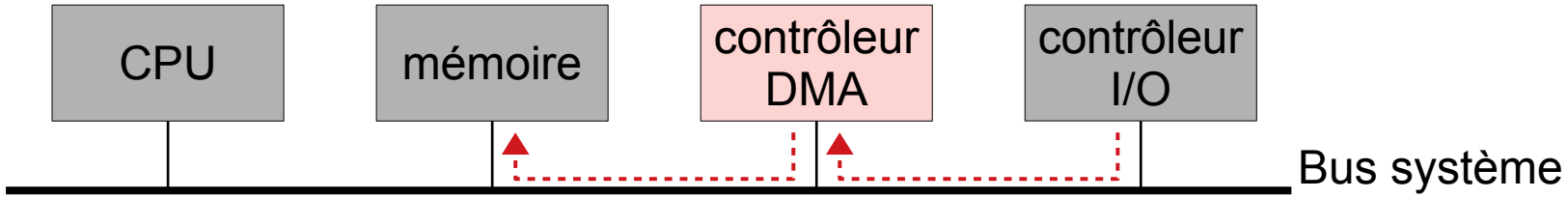
- **Discussion**
  - **I/O programmées et interruptions**
    - Que les opérations d'entrées/sorties utilisent des PIO ou des interruptions, **le processeur est impliqué dans les transferts de données entre le périphérique (ou le contrôleur d'I/O) et la mémoire**. Ces transferts nécessitent l'exécution par le CPU d'instructions **`load`** / **`store`**.
    - Cette approche pose problème aux périphériques à haut débit (disques durs, cartes réseaux). Dans ces cas, le rôle du processeur est de transférer de larges blocs de données entre le périphérique et la mémoire. Pour chaque transfert, un certain nombre d'instructions est nécessaire → **occupation importante du CPU juste pour copier des données**.

Direct Memory Access

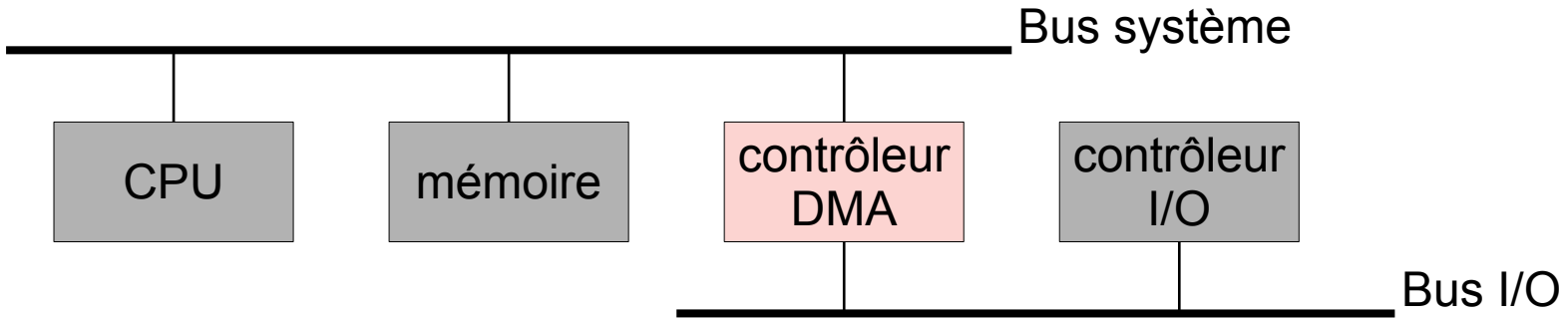
- **Principe**
  - **Accès direct à la mémoire (DMA)**
    - L'objectif de l'**accès direct à la mémoire** (*direct memory access* – **DMA**) est de soulager le CPU des transferts périphériques ↔ mémoire en donnant la possibilité d'effectuer directement des transferts entre périphériques et mémoire sans l'intervention du processeur.
    - Pour gérer ces transferts, une nouvelle entité est introduite : le **contrôleur DMA**. Celui-ci est souvent intégré au contrôleur d'I/O.
    - Le contrôleur DMA effectue le transfert indépendamment du processeur. Le transfert a lieu un mot à la fois, lorsque le processeur n'accède pas à la mémoire (bus système libre). Le contrôleur DMA peut forcer le processeur à suspendre son activité le temps d'un cycle sur le bus durant le transfert. On parle alors de **vol de cycles** (*cycle stealing*).

Direct Memory Access

- **Principe**
  - **Accès direct à la mémoire (DMA)**
    - Contrôleur DMA indépendant : utilisé dans les architectures plus anciennes. Inconvénient : bus mémoire = goulot d'étranglement.

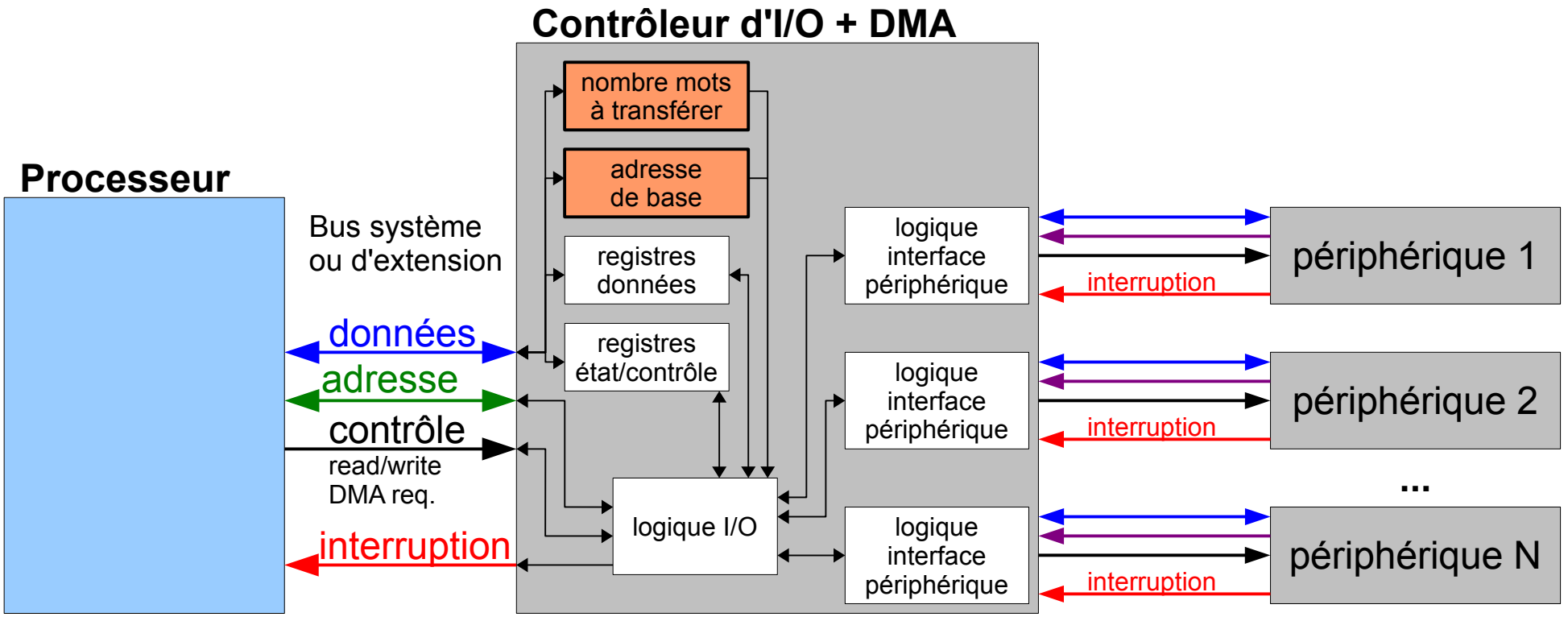


    - Contrôleur DMA intégré : intégré au contrôleur d'I/O ou sous forme de pont (*bridge*)



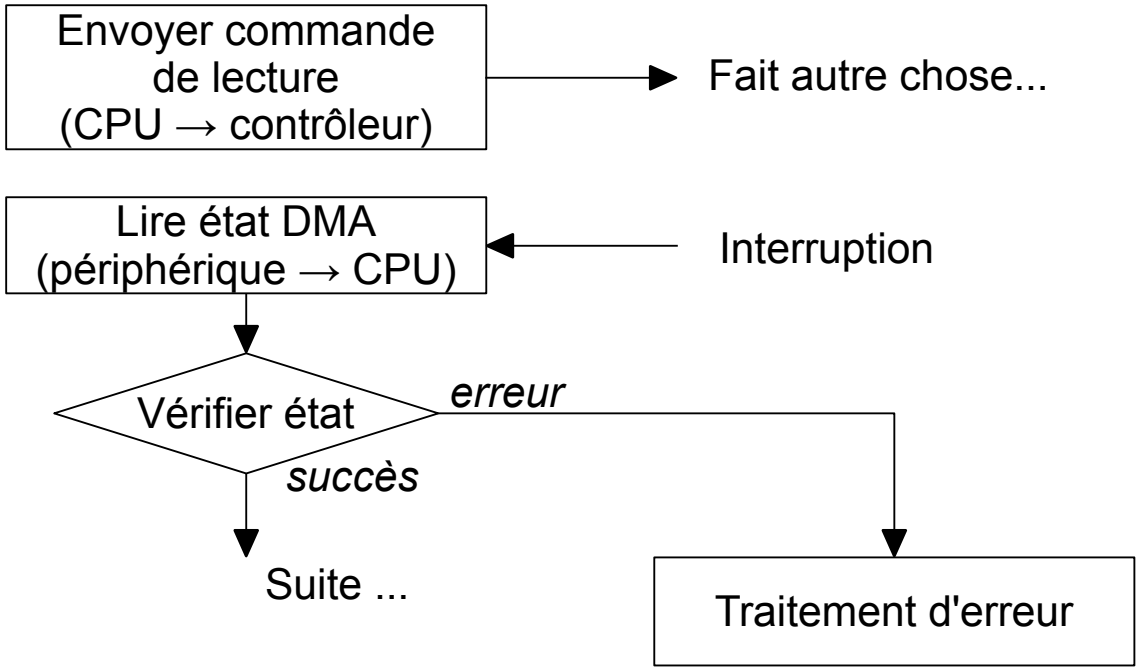
- **Principe**
  - **Transfert DMA**
    - Afin d'effectuer un transfert DMA, le processeur informe le contrôleur DMA des informations suivantes :
      - l'identification (adresse) du périphérique d'entrées/sorties
      - la direction du transfert (périphérique → mémoire, mémoire → périphérique, voire mémoire → mémoire)
      - l'adresse de base en mémoire (source ou destination selon le sens du transfert)
      - le nombre de mots à transférer.
      - des conditions pour transférer (p.ex. lors d'une IRQ d'un timer)
    - Le contrôleur DMA utilise des interruptions pour signaler au processeur la fin du transfert ou une erreur durant le transfert.

- **Exemple**
  - **Transfert DMA**
    - Le contrôleur DMA doit détenir des registres supplémentaires dans lesquels le processeur écrit le nombre de mots à transférer et l'adresse source/destination en mémoire.



Note: Inspiré de **Computer Organization and Architecture – Designing for Performance**, 8th Edition, W. Stallings, Pearson, 2010.

- **Exemple**
  - **Interruptions**
    - Lecture d'un bloc de données à partir d'un périphérique (p.ex. lecteur de disque dur) en utilisant des transferts DMA.



Note: Inspiré de **Computer Organization and Architecture – Designing for Performance**, 8th Edition, W. Stallings, Pearson, 2010.

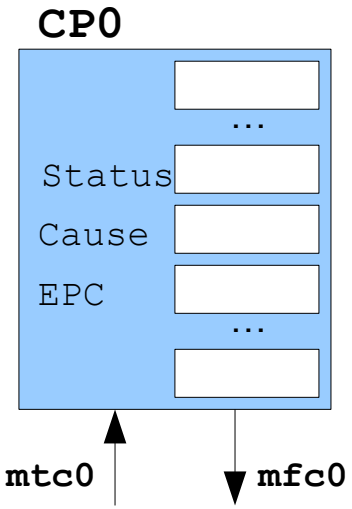
Table des Matières

- Introduction
- Types de périphériques
- Contrôleur d'entrées/sorties
- Opérations d'entrées/sorties
  - Entrées/sorties programmées (PIO)
  - Interruptions
  - *Direct Memory Access* (DMA)
- **Etude de cas**

→ Interruptions avec MIPS

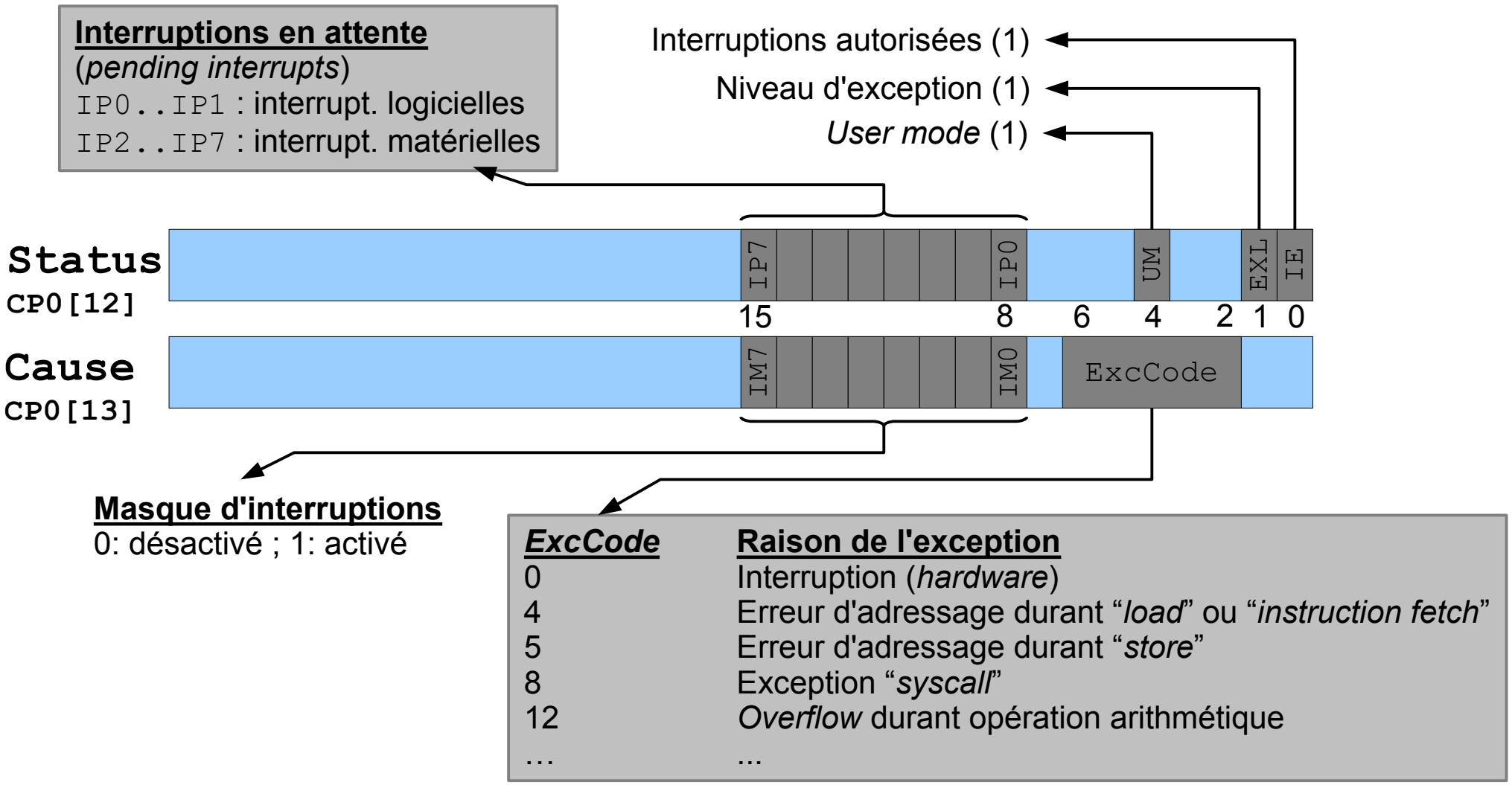
Interruptions avec MIPS

- **Etude de cas**
  - **Gestion d'exceptions sur MIPS**
    - Trois registres spéciaux sont utilisés pour gérer les exceptions. Ces registres sont situés sur un “coprocesseur” appelé `CP0`.
      - **`Cause`** (`CP0`, reg. 13)
        - La raison de l'exception est sauvegardée dans un registre spécial.
      - **`Status`** (`CP0`, reg. 12)
        - Le registre spécial `Status` peut être utilisé pour activer/désactiver exceptions et interruptions.
      - **`EPC`** - *Exception Program Counter* (`CP0`, reg. 14)
        - Contient l'adresse de l'instruction en cours d'exécution lorsque l'exception a eu lieu. Il est possible de retourner à cette adresse en utilisant l'instruction spéciale **`eret`** (*Exception RETurn*).
      - L'accès aux registres de `CP0` est possible indirectement en utilisant les instructions *Move From Coprocessor 0* (**`mfc0`**) et *Move To Coprocessor 0* (**`mtc0`**).



Interruptions avec MIPS

- **Etude de cas**
  - **Gestion d'exceptions sur MIPS**



Interruptions avec MIPS

- **Etude de cas**
  - **Gestion d'exceptions sur MIPS**
    - La gestion des exceptions nécessite l'introduction de nouvelles instructions MIPS.

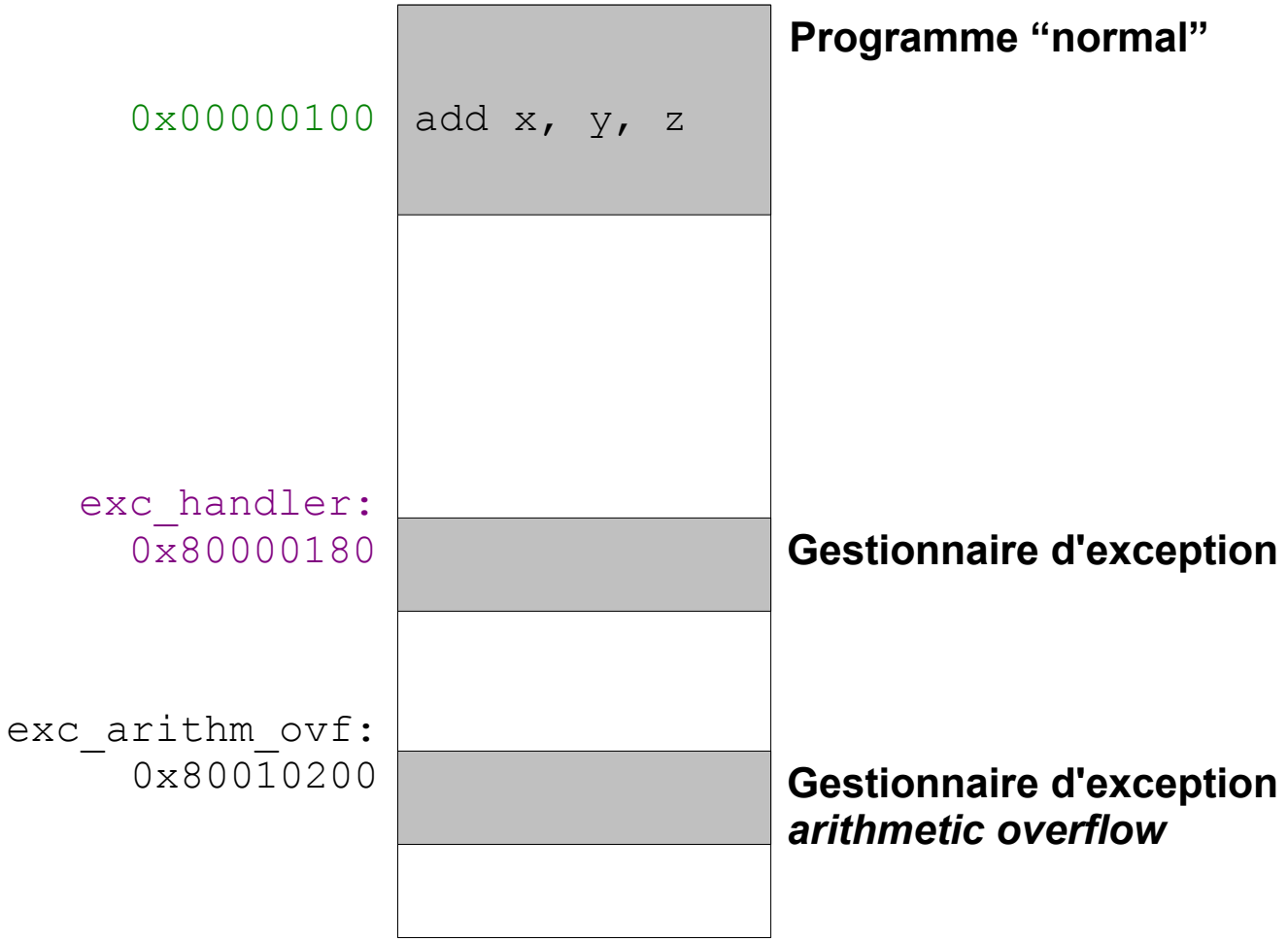
Instruction	**Description**
Exception RETurn	
ERET	$Status_{EXL} \leftarrow 0$ PC ← EPC
Move From CP0	
MFC0 rt, rd	GPR[rt] ← CP0Reg[rd]
Move To CP0	
MFT0 rt, rd	CP0Reg[rd] ← GPR[rt]

Interruptions avec MIPS

- **Principe**
  - **Gestionnaire d'exception**
    - Un **gestionnaire d'exception** (*exception handler*) est un morceau de programme généralement fourni par le système d'exploitation qui est appelé lorsqu'une exception doit être traitée.
    - Le gestionnaire d'exception ne peut pas perturber l'exécution du programme courant. Pour cela, avant de s'exécuter, il doit sauvegarder les registres qu'il pourrait modifier. Les valeurs de ces registres sont restaurées avant le retour du gestionnaire d'interruption.
    - L'architecture MIPS32 utilise une adresse de branchement fixe pour le gestionnaire d'exceptions. C'est le gestionnaire d'exception qui se charge d'appeler un gestionnaire spécialisé sur base de la source d'exception trouvée dans le registre `Cause`.
      - Dans le simulateur SPIM, cette adresse est fixée à la valeur `0x80000180`. Dans le microcontrôleur PIC32 utilisé en TP, l'architecture MIPS est étendue de façon à supporter un vecteur d'interruptions.

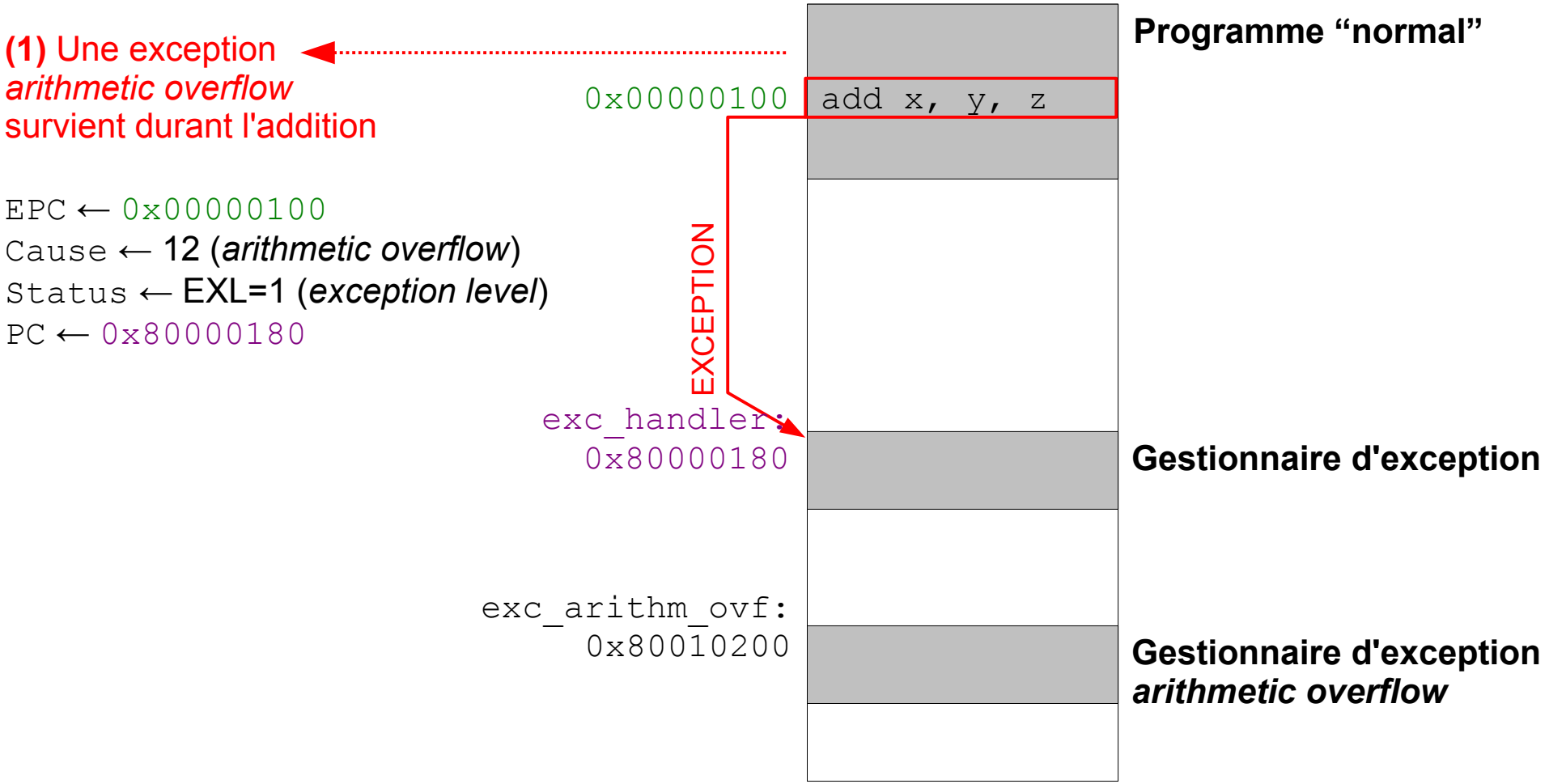
Interruptions avec MIPS

- **Etude de cas**
  - ***Arithmetic overflow* sur MIPS**

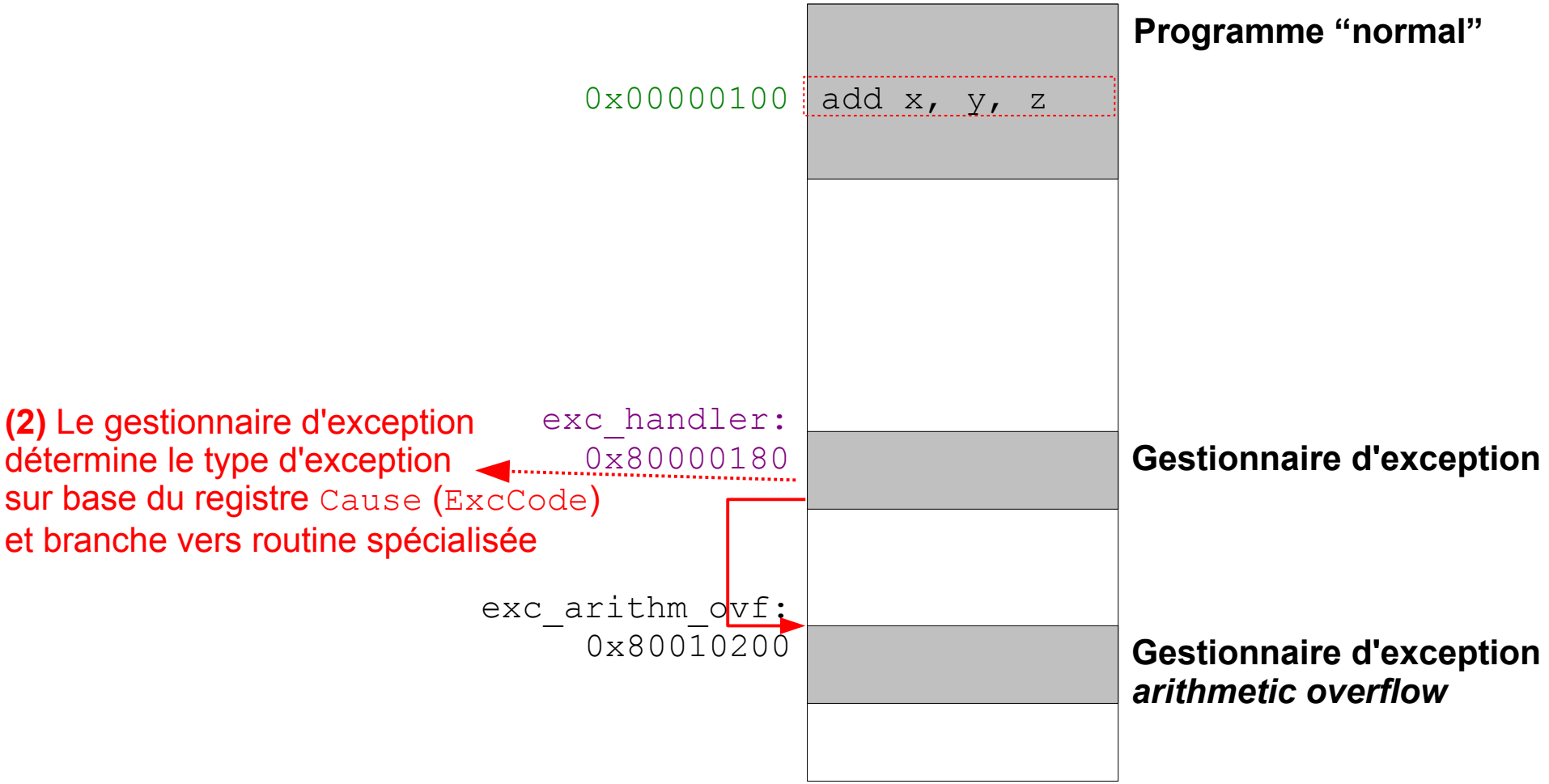


Interruptions avec MIPS

- **Etude de cas**
  - ***Arithmetic overflow* sur MIPS**

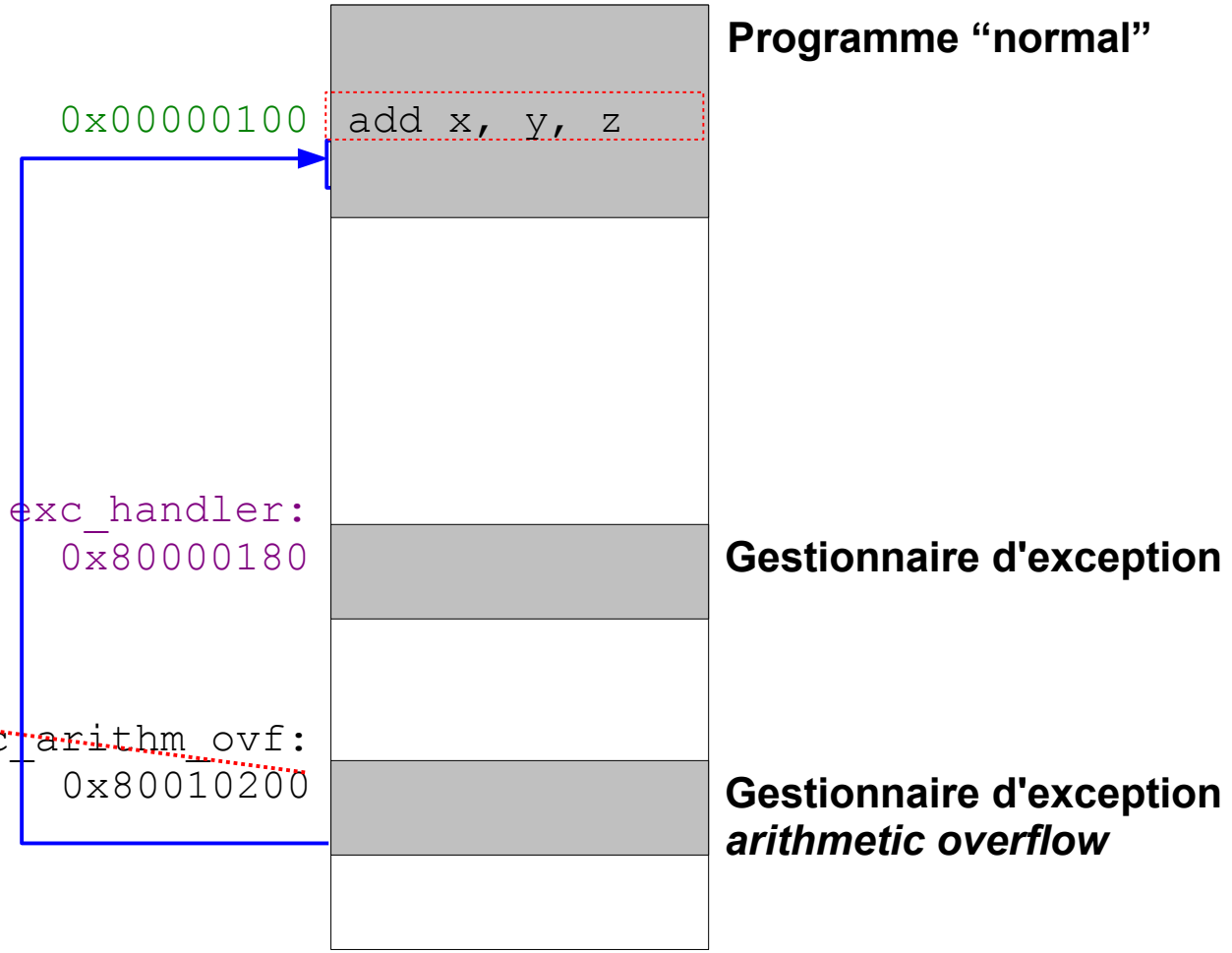


- **Etude de cas**
  - ***Arithmetic overflow* sur MIPS**



Interruptions avec MIPS

- **Etude de cas**
  - ***Arithmetic overflow* sur MIPS**



(3) Le gestionnaire d'exception affiche un message d'erreur, incrémente de 4 le registre `EPC` et retourne (**`eret`**)

`EPC` ← `0x00000104`
`EXL` ← `0`

Interruptions avec MIPS

- **Exemple**
  - **Gestionnaire d'exception**
    - Le programme en langage d'assemblage ci-dessous illustre l'implémentation d'un gestionnaire d'exception « *arithmetic overflow* » pour processeur MIPS.

```
      .ktext 0x80000180
exc_handler:
    …       # Sauve certains registres (a0, a1)
    mfc0    $k0, $13            # Obtient registre Cause (CP0[13])

    srl     $a0, $k0, 2         # Extrait ExcCode (bits 2-6)
    andi    $a0, $a0, 0x1F
    li      $a1, 12             # Code arithm. overflow
    beq     $a0, $a1, exc_arithm_ovf
    …

exc_arithm_ovf:
    …       # Affiche message d'erreur
    mfc0    $a0, $14            # Incrémente EPC (CP0[14]) de 4
    addi    $a0, 4
    mft0    $a0, $14
    …                           # Restaure registres sauvegardés (a0, a1)
    eret                        # Branche vers l'adresse EPC
```

Références

- **Computer Organization and Design**, 4th edition, D. Patterson and J. Hennessy, Morgan-Kauffman, 2009.
- **CODE: The Hidden Language of Computer Hardware and Software**, C. Petzold, Microsoft Press, 1999
- **A Practical Introduction to Computer Architecture**, D. Page, Springer-Verlag, 2009.
- **Computer Organization and Architecture – Designing for Performance**, 8th edition, W. Stallings, Pearson, 2010.