Architecture des ordinateurs

Sylvain MONTAGNY

sylvain.montagny@univ-savoie.fr

Bâtiment chablais, bureau 13

04 79 75 86 86

Retrouver tous les documents de Cours/TD/TP sur le site
www.master-electronique.com

Présentation cours : Sommaire

- Cours : 12 h en 8 séances
  - Chapitre 1 : Rappels généraux sur les processeurs
  - Chapitre 2 : Le pipeline des microprocesseurs
  - Chapitre 3 : Les mémoires caches
  - Chapitre 4 : Les interruptions
  - Chapitre 5 : Les accès DMA

Présentation TD

- TD : 15 h en 10 séances
  - TD 1 : Rappels sur les architectures à microprocesseurs
  - TD 2 : Pipeline
  - TD 3 : Mémoires Caches
  - TD 4 : Les interruptions
  - TD 5 : Les transferts DMA

Présentation TP

- TP : 16 h en 4 séances
  - TP 1 : Simulation de mémoire cache et pipeline
  - TP 2 : Programmation d’applications sur cible
  - TP 3 : Programmation d’applications sur cible

Chapitre 1 : Rappel généraux sur les processeurs

- 1.1 Rappel sur l'architecture interne des microprocesseurs
- 1.2 Le traitement des instructions
- 1.3 Les modes d'adressages
- 1.4 Exemple d'exécution d'un programme

L’architecture interne

Wafer

Un microprocesseur est constitué d’un morceau de silicium dopé. C’est donc un ensemble de millions de transistors.

- Wafer : Galette de plusieurs processeurs
- 1 processeur : quelques millimètres carrés

L’architecture interne

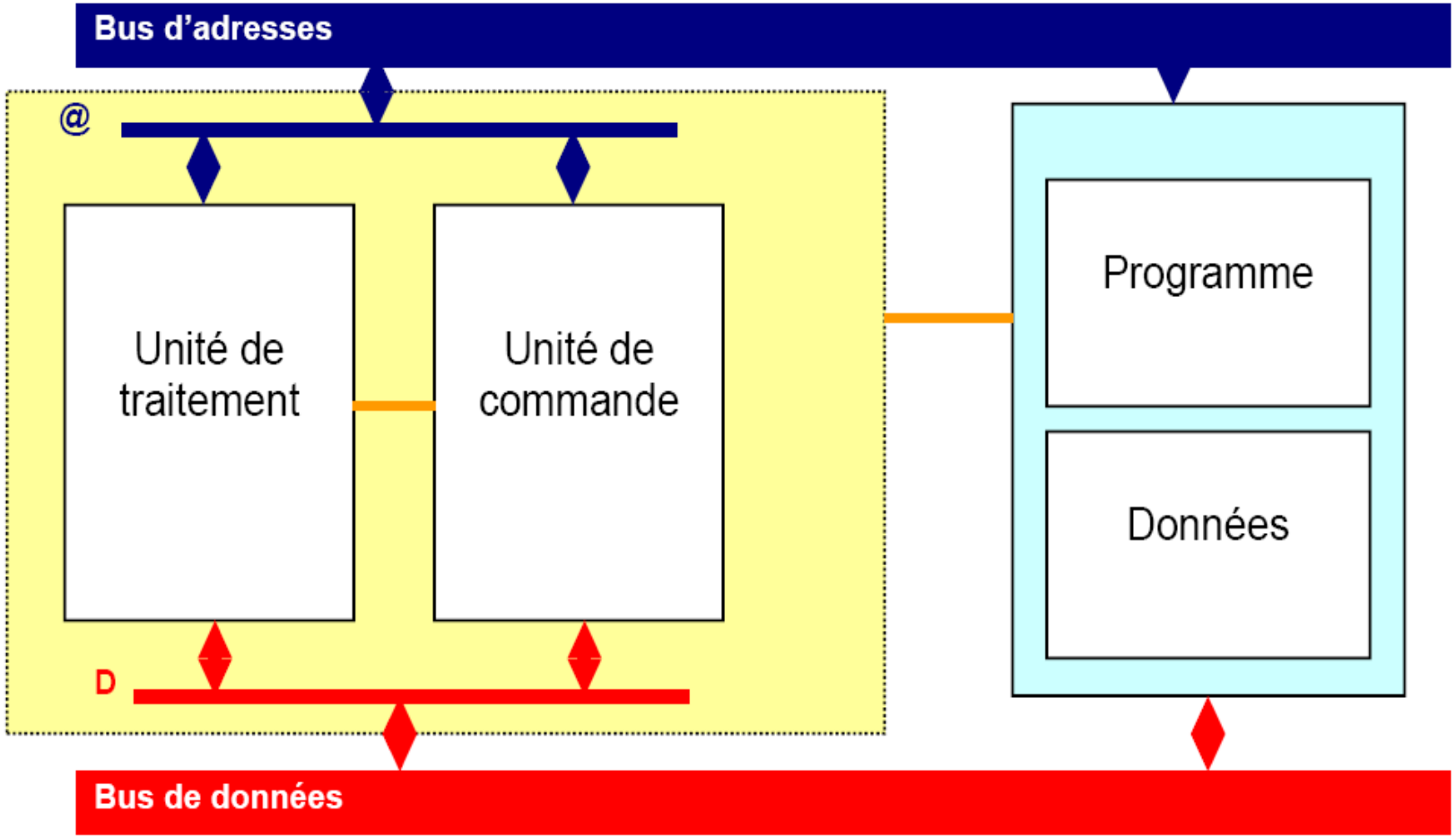
Unité commande/traitement

Un microprocesseur est construit autour de deux éléments principaux :

- Une unité de commande
- Une unité de traitement

L’architecture interne

Schéma



L’architecture interne

L’unité de commande (1)

Elle permet de séquencer le déroulement des instructions. Elle effectue la recherche en mémoire de l'instruction, le décodage de l’instruction codée sous forme binaire. Enfin elle pilote l’exécution de l’instruction.

Les blocs de l’unité de commande :

1. **Le compteur de programme (PC : Programme Counter) appelé aussi Compteur Ordinal (CO)** est constitué par un registre dont le contenu est initialisé avec l'adresse de la première instruction du programme. Il contient toujours l’adresse de la prochaine instruction à exécuter.

L’architecture interne

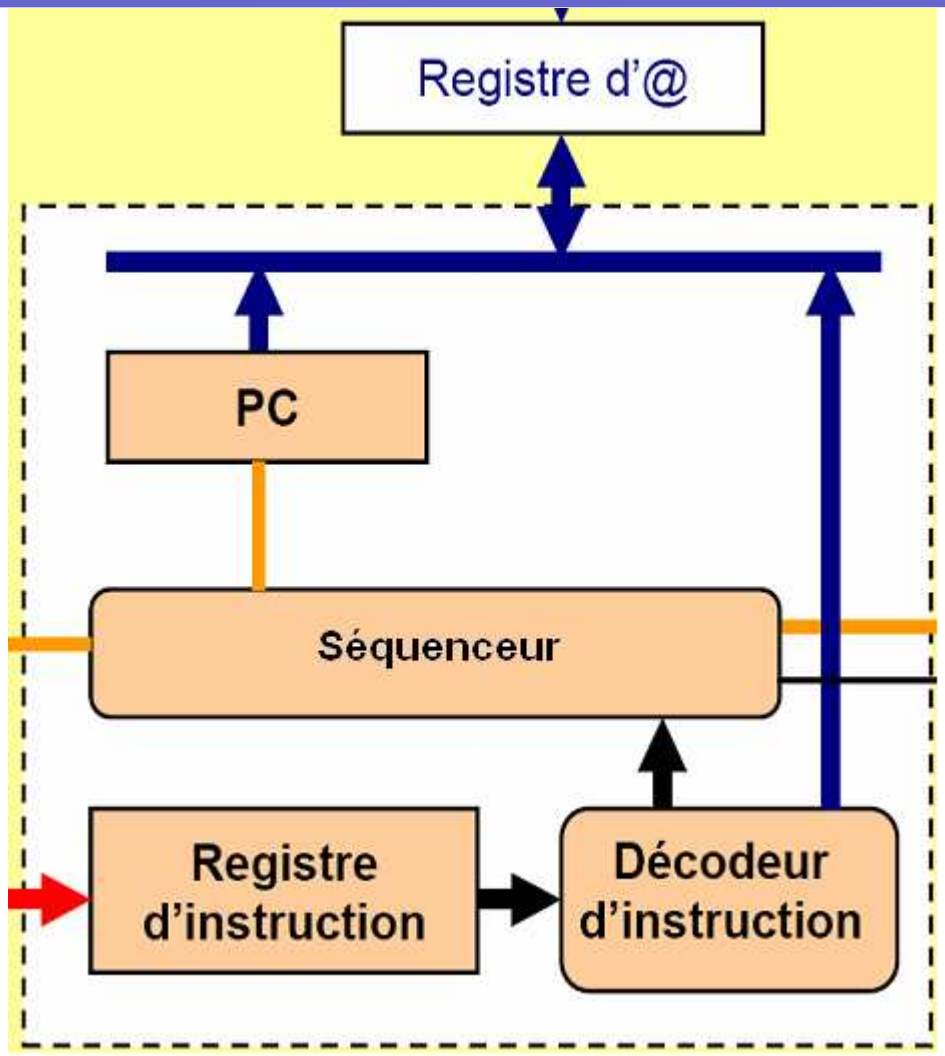
L’unité de commande (2)

2. **Le registre d'instruction et le décodeur d'instruction :** Chacune des instructions à exécuter est transféré depuis la mémoire dans le registre instruction puis est décodée par le décodeur d’instruction.

3. **Bloc logique de commande (ou séquenceur) :** Il organise l'exécution des instructions au rythme d’une horloge. Il élabore tous les signaux de synchronisation internes ou externes (bus de commande) du microprocesseur en fonction de l’instruction qu’il a a exécuter. Il s'agit d'un automate réalisé de façon micro-programmée.

L’architecture interne

L’unité de commande (3)



L’architecture interne

L’unité de traitement (1)

Elle regroupe les circuits qui assurent les traitements nécessaires à l'exécution des instructions

Les blocs de l’unité de traitement :

1. **Les accumulateurs** sont des registres de travail qui servent à stocker une opérande au début d'une opération arithmétique et le résultat à la fin de l'opération.
2. **L’Unité Arithmétique et Logique (UAL)** est un circuit complexe qui assure les fonctions logiques (ET, OU, Comparaison, Décalage, etc…) ou arithmétique (Addition, soustraction…).

L’architecture interne

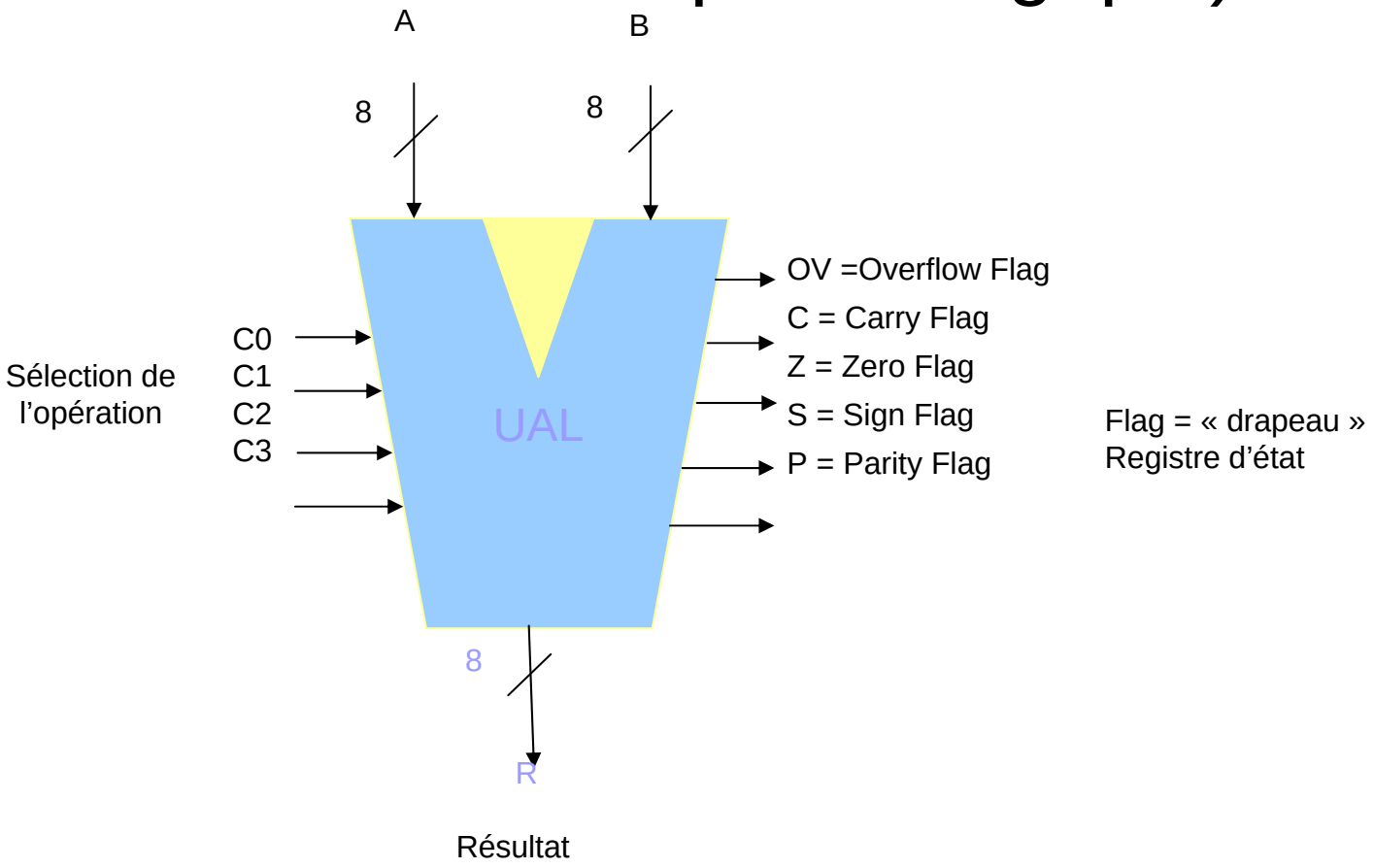
L’unité de traitement (2)

3. **Le registre d'état** est généralement composé de 8 bits à considérer individuellement. Chacun de ces bits est un indicateur dont l'état dépend du résultat de la dernière opération effectuée par l’UAL. On les appelle *indicateur d’état* ou *flag* ou *drapeaux*. Dans un programme le résultat du test de leur état conditionne souvent le déroulement de la suite du programme. On peut citer par exemple les indicateurs de :
   - Retenue **(carry : C)**
   - Débordement **(overflow : OV ou V)**
   - Zéro **(Z)**
   - ...

L'architecture interne

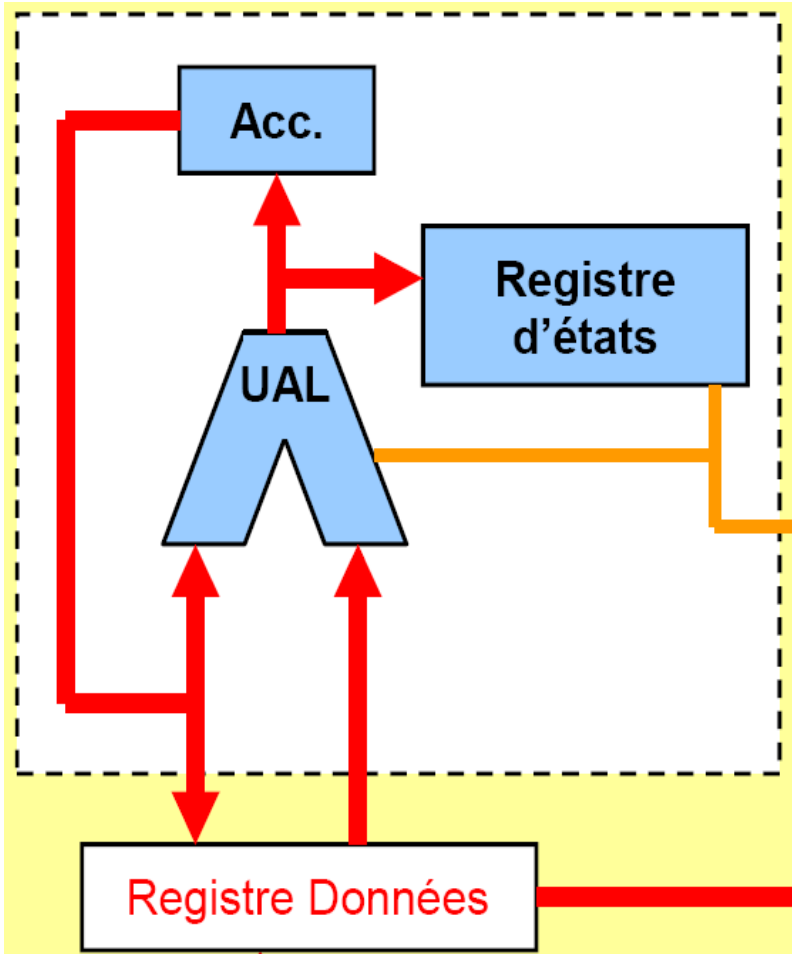
L'Unité de traitement (3)

UAL : Unité Arithmétique et Logique)



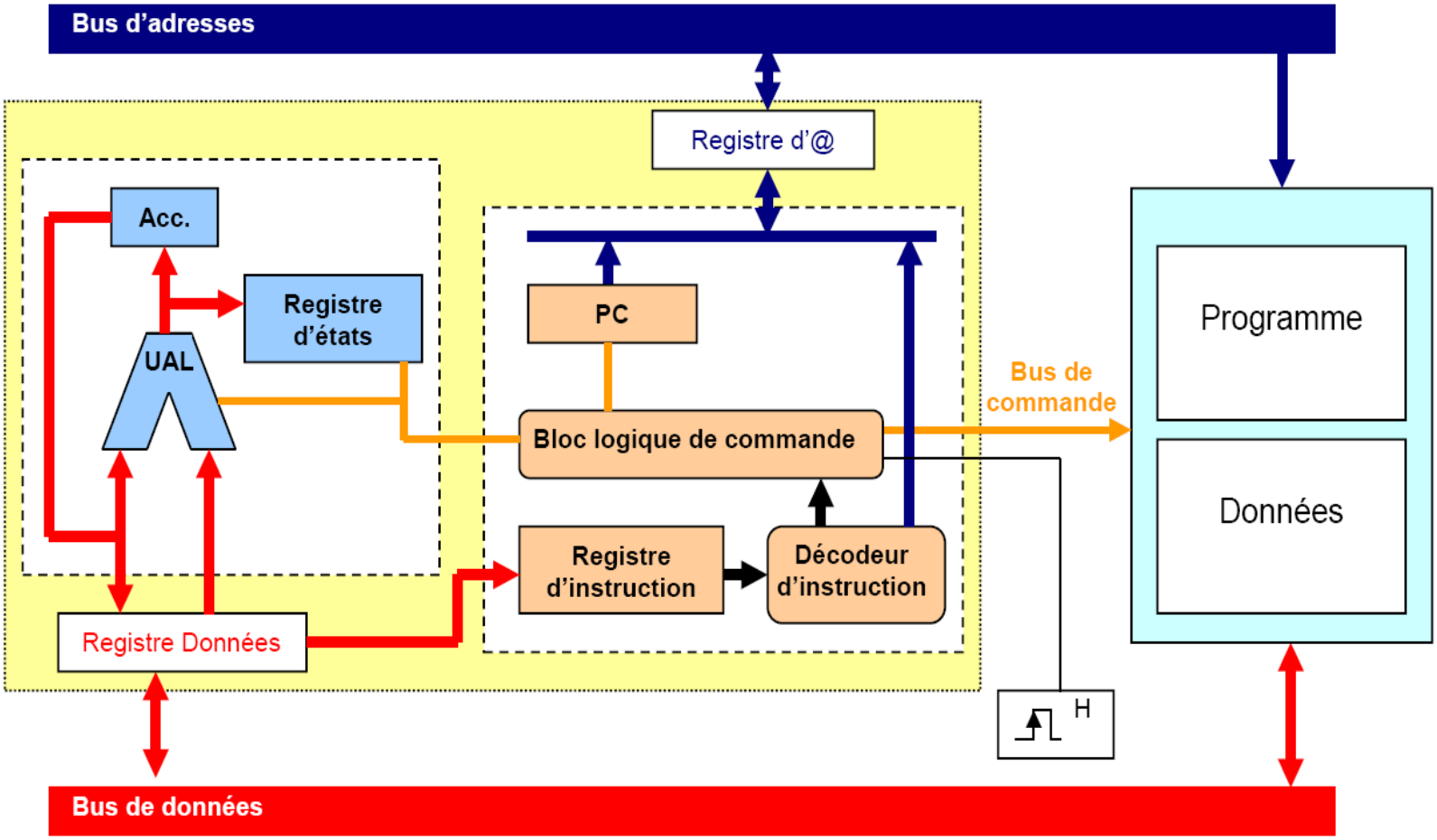
L’architecture interne

L’unité de traitement (4)

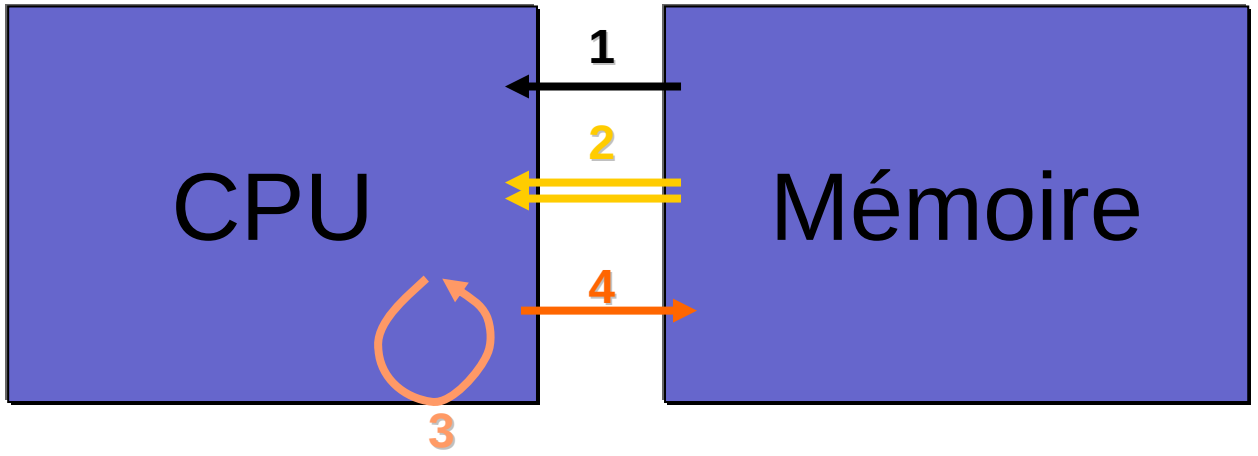


L'architecture interne

Architecture complète

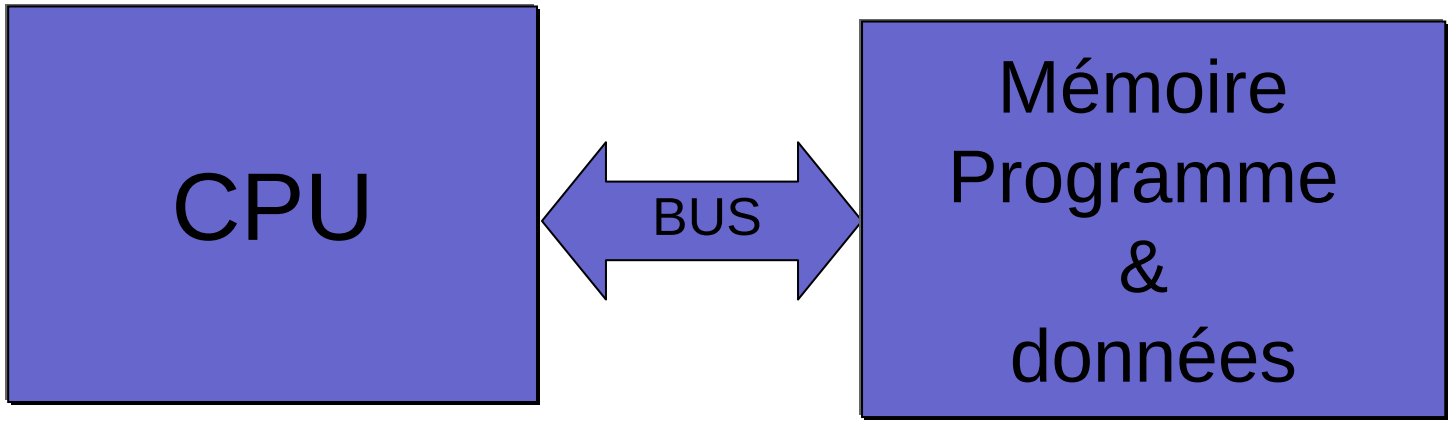


Rappels: le fonctionnement basique d'une opération de calcul



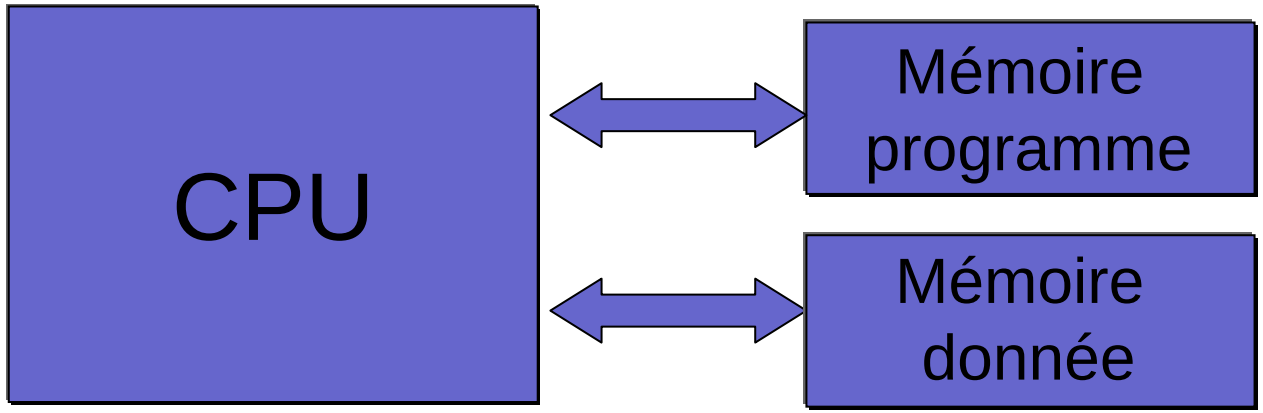
- (1) Charger une instruction depuis la mémoire
- (2) Charger les opérandes depuis la mémoire
- (3) Effectuer les calculs
- (4) Stocker le résultat en mémoire

L’architecture
Von Neuman



- Un seul chemin d'accès à la mémoire
  - Un bus de données (programme et données),
  - Un bus d’adresse (programme et données)
- Architecture des processeurs d’usage général
- Goulot d'étranglement pour l'accès à la mémoire

L'architecture
Harvard

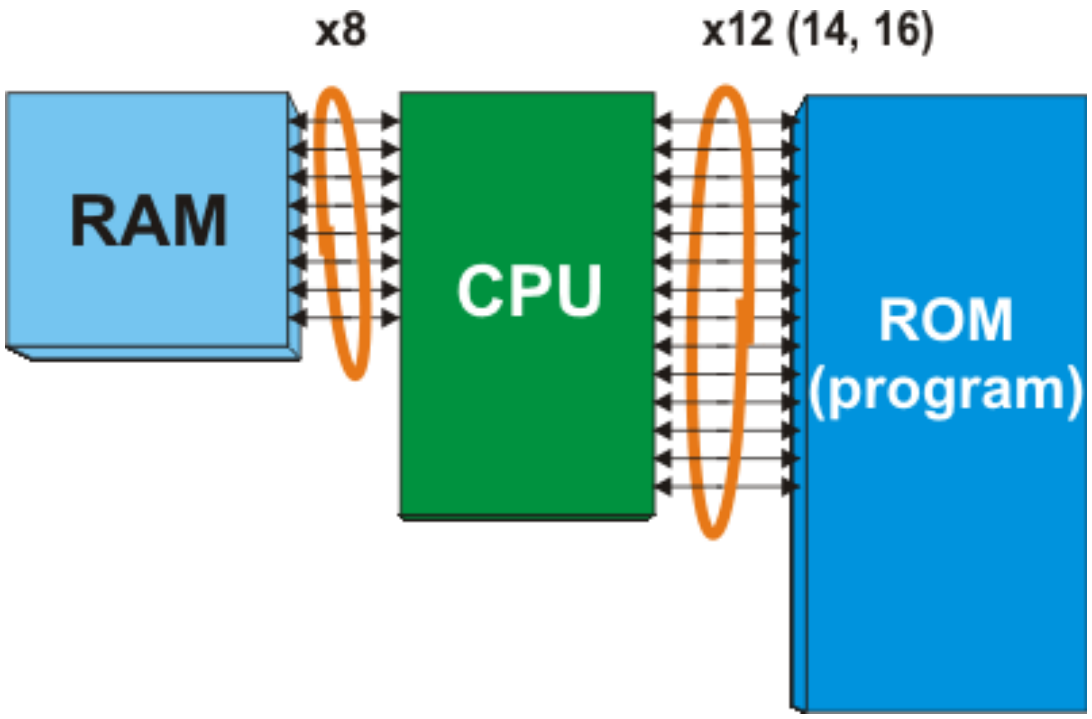


- Séparation des mémoires programme et données
  - Un bus de données programme,
  - Un bus de données pour les données,
  - Un bus d'adresse programme,
  - Un bus d'adresse pour les données.
- Meilleure utilisation du CPU :
  - Chargement du programme et des données en parallèle

L’architecture

Harvard : Cas des microcontrôleurs PIC

- Seul les bus de donnée (data ou instructions) sont représentées



Chapitre 1 : Rappel généraux sur les processeurs

- 1.1 Rappel sur l'architecture interne des microprocesseurs
- 1.2 Le traitement des instructions
- 1.3 Les modes d'adressages
- 1.4 Exemple d'exécution d'un programme

Le traitement des instructions

Organisation d’une instruction

Le microprocesseur ne comprend qu’un certain nombre d’instructions qui sont codées en binaire. Une instruction est composée de deux éléments :

- Le code opération : C’est un code binaire qui correspond à l’action à effectuer par le processeur
- Le champ opérande : Donnée ou bien adresse de la donnée.

La taille d'une instruction peut varier, elle est généralement de quelques octets (1 à 8), elle dépend également de l'architecture du processeur.

Le traitement des instructions
Exemple d'instruction

- Instruction Addition :

  Accumulateur = Accumulateur + Opérande

Correspond à l'instruction ADD A,#2

Instruction (16 bits)	
Code opératoire (5 bits)	Champ opérande (11 bits)
ADD A	#2
11001	000 0000 0010

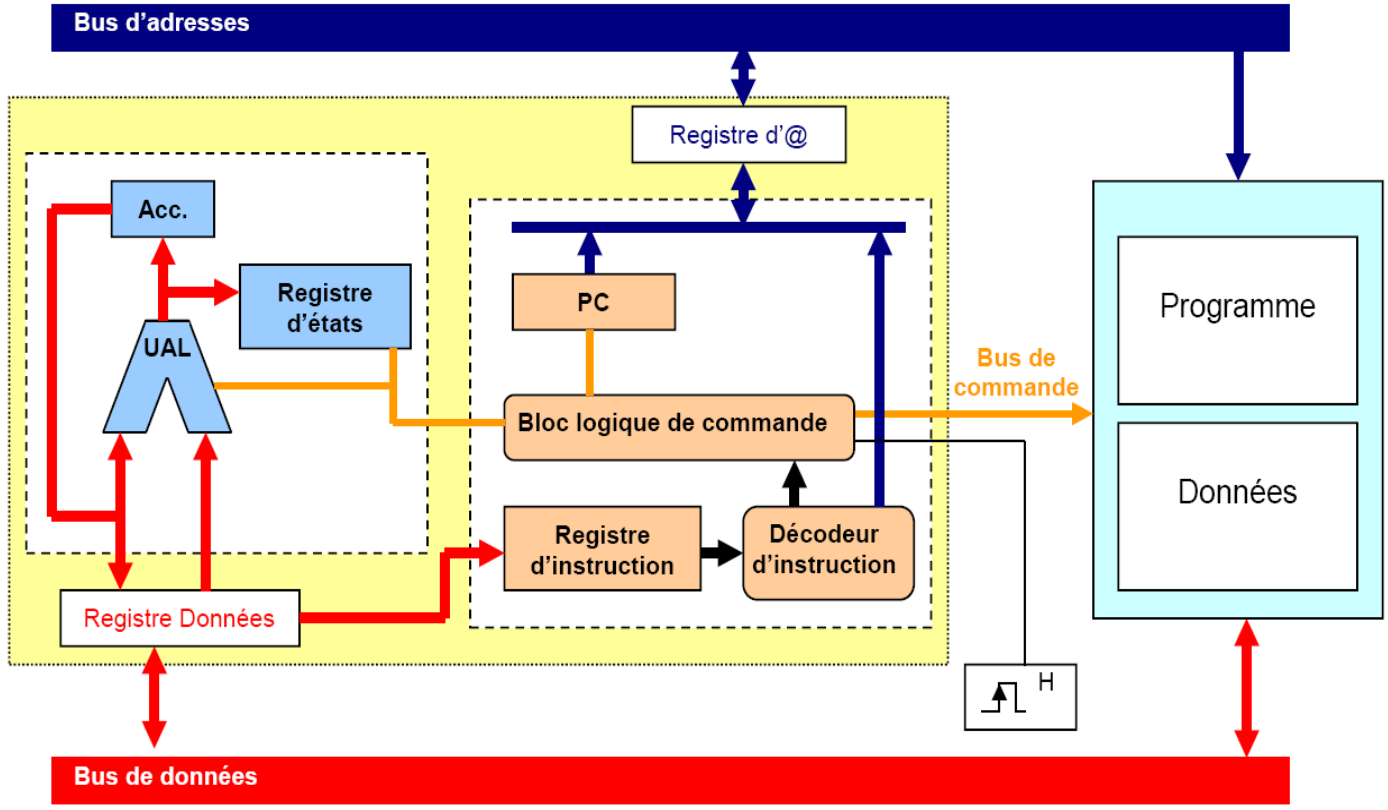
Cette instruction est comprise par le processeur par le mot binaire :

11001 000 0000 0010 = code machine

Le traitement des instructions

Phase 1 : Recherche de l'instruction en mémoire

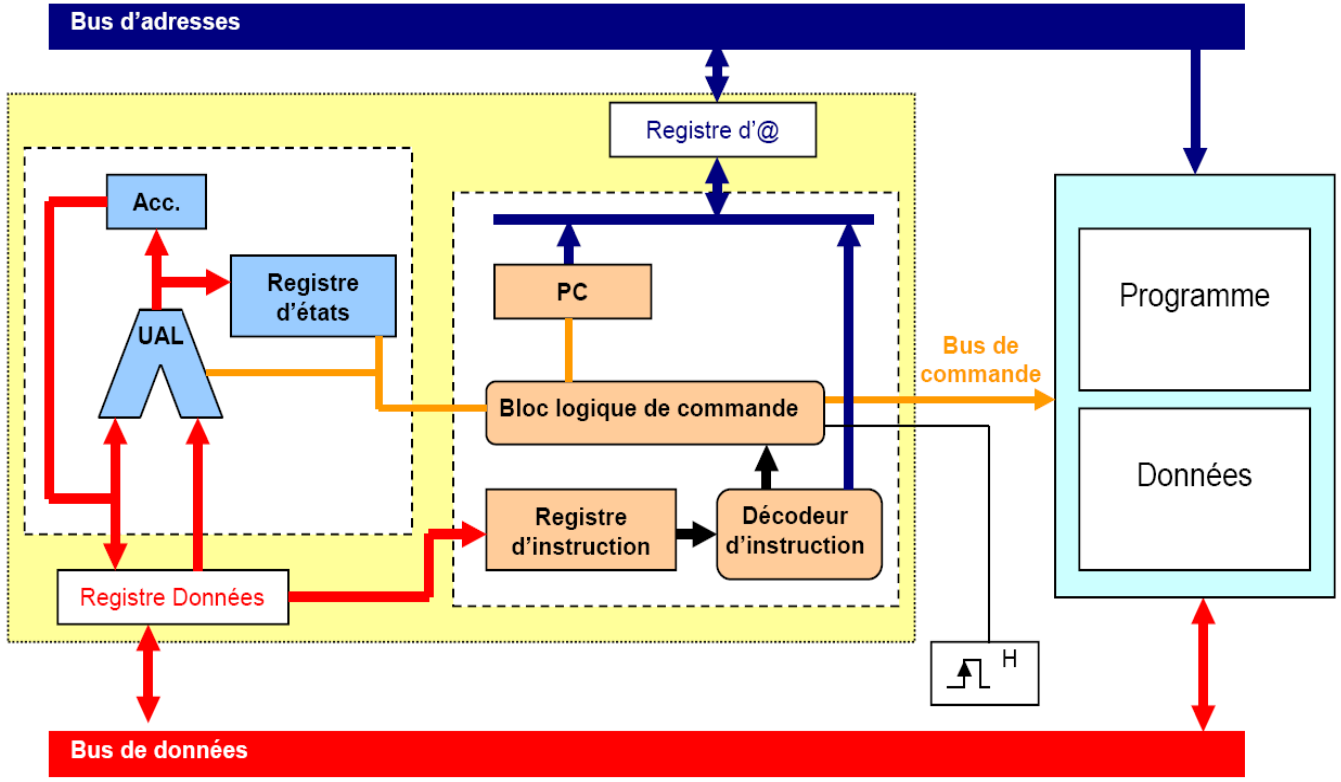
- La valeur du PC est placée sur le bus d'adresse par l'unité de commande qui émet un ordre de lecture.
- Après le temps d'accès à la mémoire, le contenu de la case mémoire sélectionnée est disponible sur le bus des données.
- L'instruction est stockée dans le registre d'instruction du processeur.



Le traitement des instructions

Phase 2 : Décodage et recherche de l'opérande

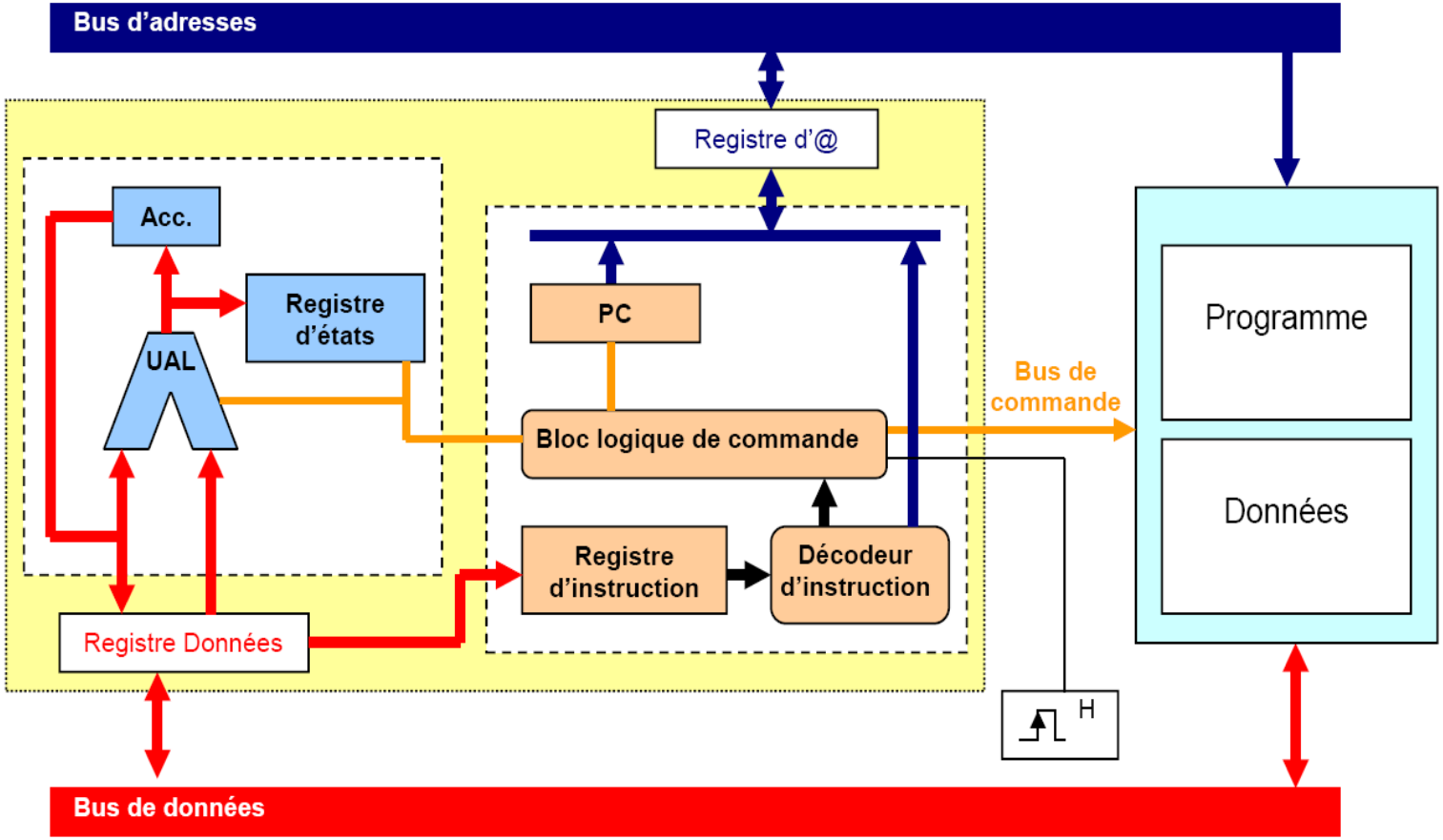
- L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.
- Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.
- L'opérande est stocké dans le registre de données.



Le traitement des instructions

Phase 3 : Exécution de l'instruction

- Le séquenceur réalise l'instruction.
- Les drapeaux sont positionnés (registre d'état).
- L'unité de commande positionne le PC pour l'instruction suivante.



Le traitement des instructions
Les architectures RISC et CISC (1)

Actuellement l'architecture des microprocesseurs se composent de deux grandes familles :

- L' architecture CISC

  (Complex Instruction Set Computer)

- L'architecture RISC

  (Reduced Instruction Set Computer)

Le traitement des instructions

Les architectures RISC et CISC (2)

Architecture RISC	Architecture CISC
instructions simples ne prenant qu'un seul cycle	instructions complexes prenant plusieurs cycles
instructions au format fixe	instructions au format variable
décodeur simple (câblé)	décodeur complexe (microcode)
beaucoup de registres	peu de registres
peu de modes d'adressage	beaucoup de modes d'adressage
compilateur complexe	compilateur simple

Chapitre 1 : Rappels généraux sur les processeurs

- 1.1 Rappel sur l'architecture interne des microprocesseurs
- 1.2 Le traitement des instructions
- 1.3 Les modes d'adressages
- 1.4 Exemple d'exécution d'un programme

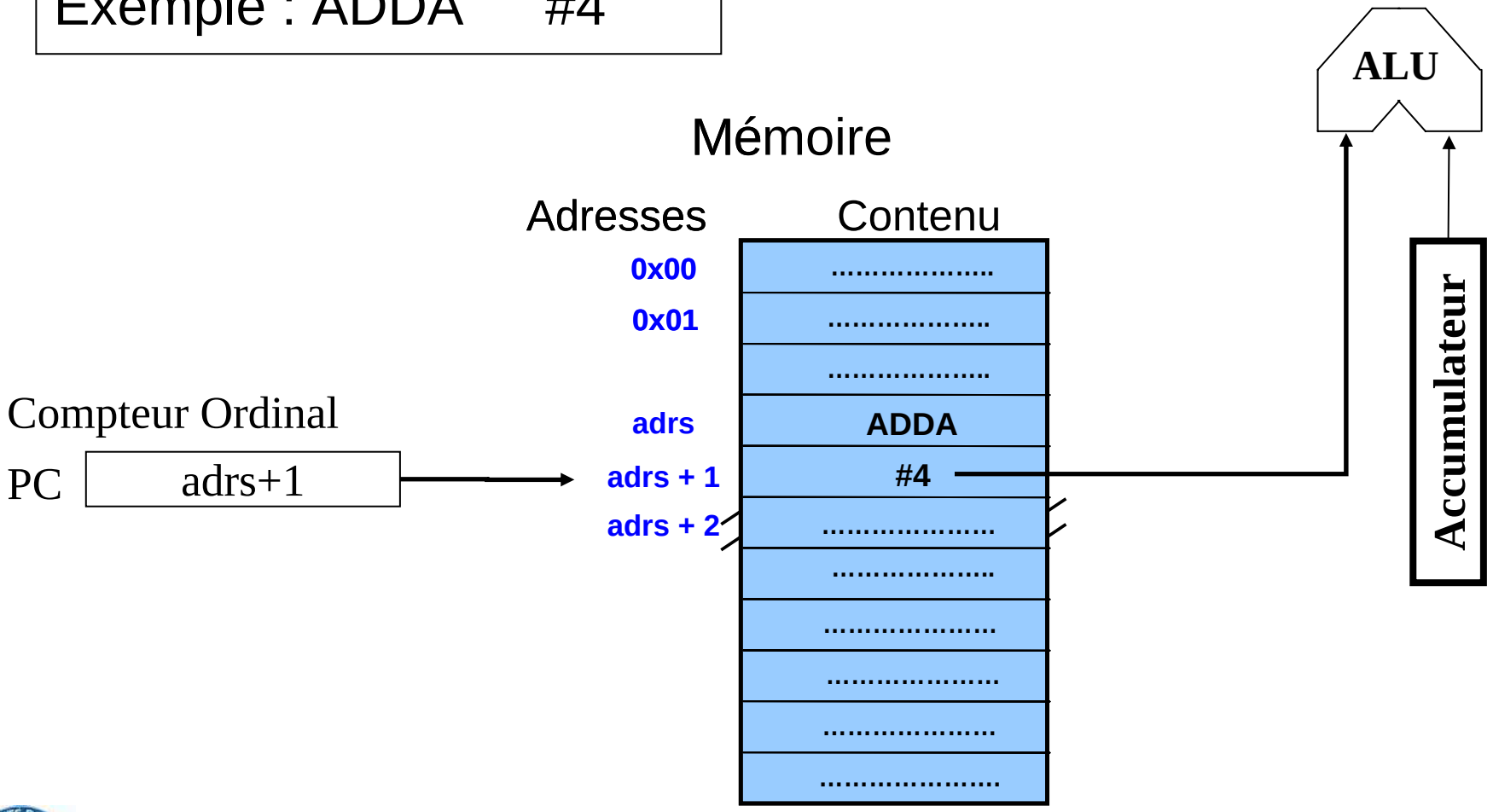
Les modes d’adressages

- Ce sont les diverses manières de définir la localisation d ’un opérande. Les trois modes d’adressage les plus courant sont :
  - Adressage immédiat
  - Adressage direct
  - Adressage indirect

Les modes d'adressages

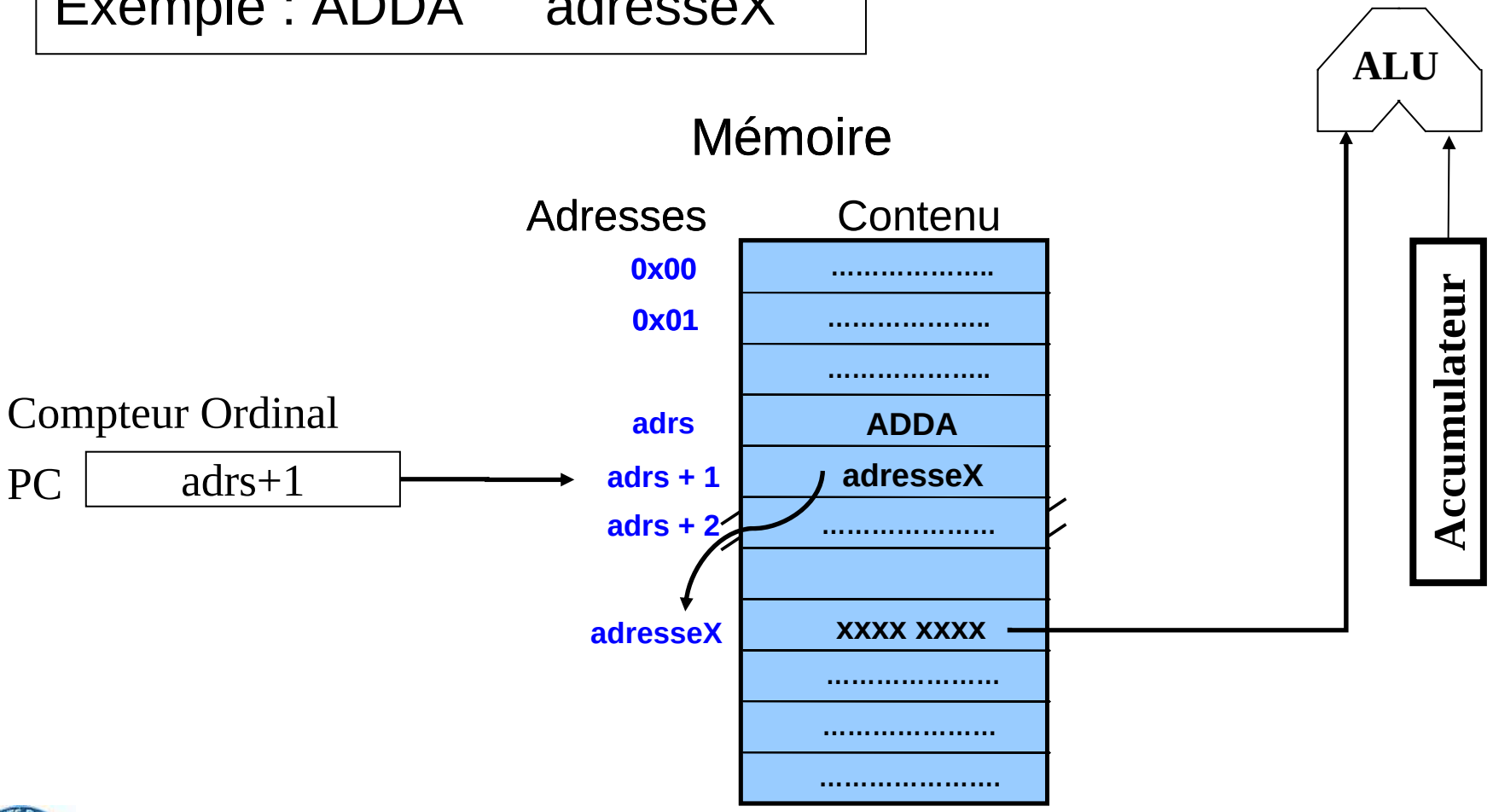
Immédiat

Exemple : ADDA #4



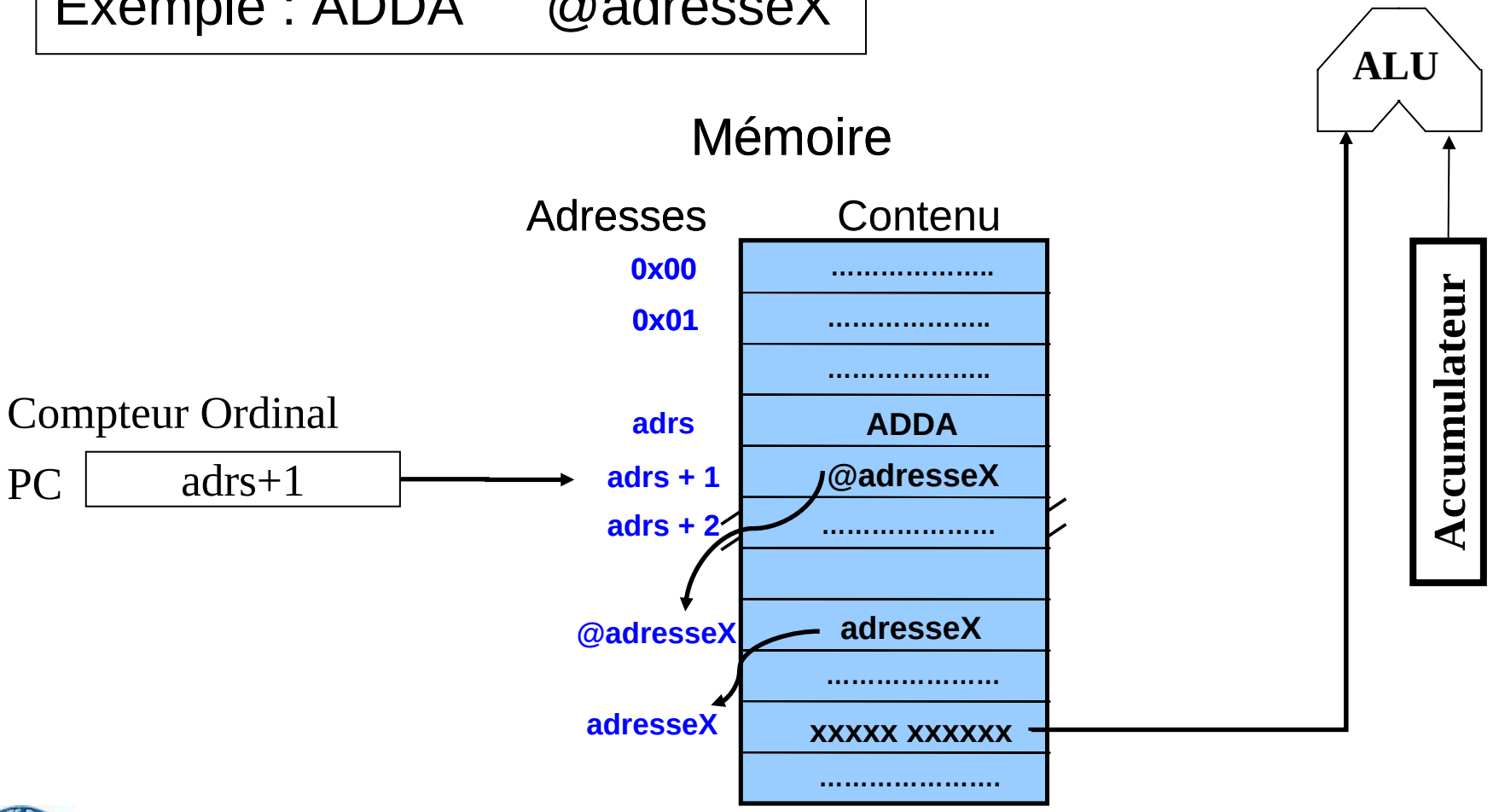
Les modes d'adressages
Direct

Exemple : ADDA adresseX



Les modes d'adressages
Indirect

Exemple : ADDA @adresseX



Les modes d’adressages

- Pourquoi existe-t-il plusieurs modes d’adressage ?

Chapitre 1 : Rappel généraux sur les processeurs

- 1.1 Rappel sur l'architecture interne des microprocesseurs
- 1.2 Le traitement des instructions
- 1.3 Les modes d'adressages
- 1.4 Exemple d'exécution d'un programme

Exemple d'exécution

Directives d'assemblage

Valeurs des symboles

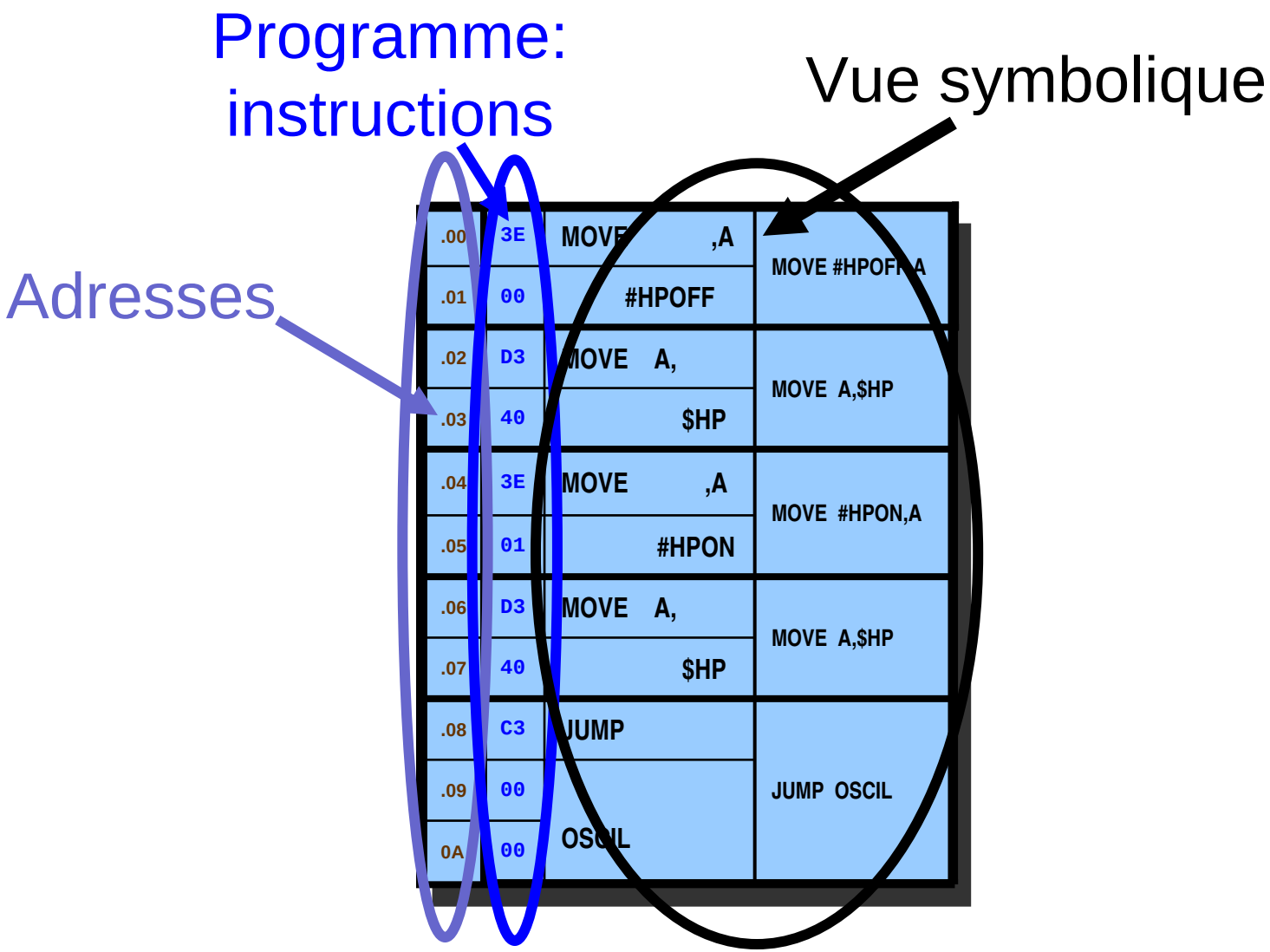
```
                .TITLE    Bruit_HP      ; Titre du programme
                .PROC     I8085         ; Processeur utilisé
                .START    OSCIL         ; Adresse début programme

00000040  HP      =   10'64     ; Adresse  du Haut-Parleur (40 Hexa)
00000000  HPOFF   =   0         ; Constante, membrane relachée
00000001  HPON    =   1         ; Constante, membrane attirée

000000          .LOC 0          ; Adresse d'assemblage du programme

000000  OSCIL:
```

Mnémoniques des instructions

Adresses	Code des instructions		Commentaires
000000	3E 00	MOVE #HPOFF, A	; Charge valeur HPOFF (0) dans l'accumulateur A
000002	D3 40	MOVE A, $HP	; Charge A sur périphérique HP
000004	3E 01	MOVE #HPON, A	; Charge valeur HPON (1) dans l'accumulateur A
000006	D3 40	MOVE A, $HP	; Charge A sur périphérique HP
000008	C3 00 00	JUMP OSCIL	; Saute au début à OSCIL
00000B			

```
                .END            ; Fin de l'assemblage
```

Exemple d'exécution



Exemple d’exécution

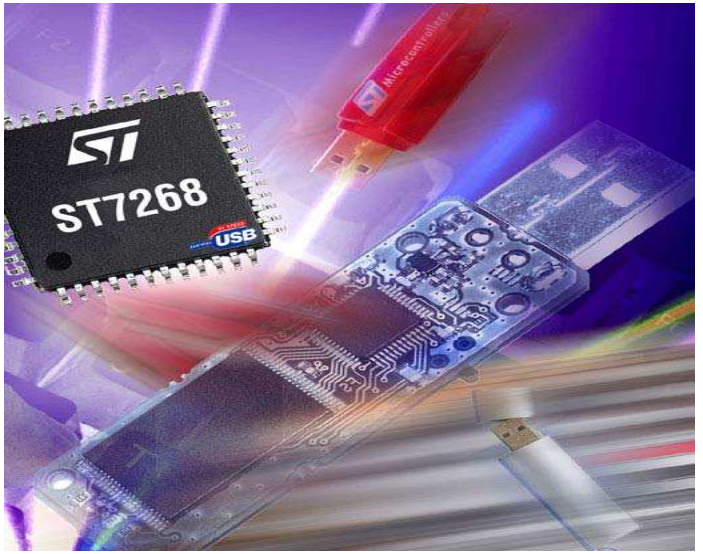
Continue….

1ière boucle

2ième boucle

3ième boucle

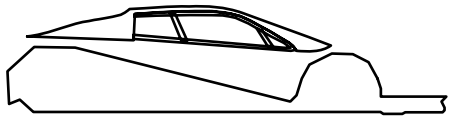
4ième boucle



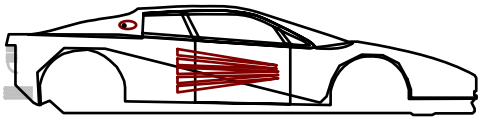
Chapitre 2 : Le pipeline

- 2.1 Définition d'un pipeline
- 2.2 Les étages d'un pipeline
- 2.3 Les aléas dans le pipeline

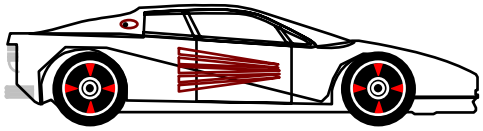
Définition d'un pipeline
Comparaison (1)



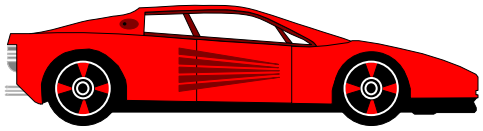
1ère étape de conception



2ème étape de conception



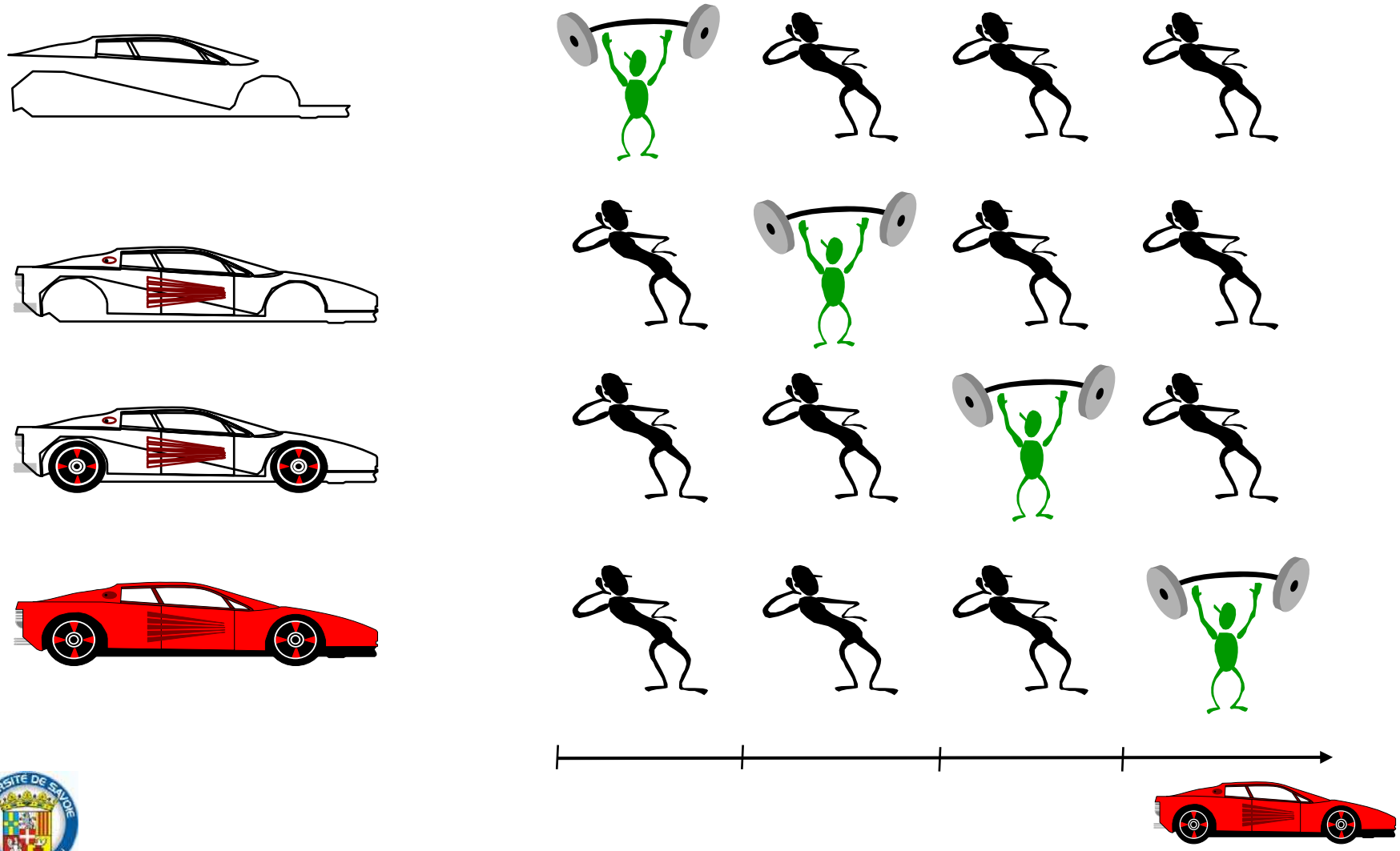
3ème étape de conception



4ème étape de conception

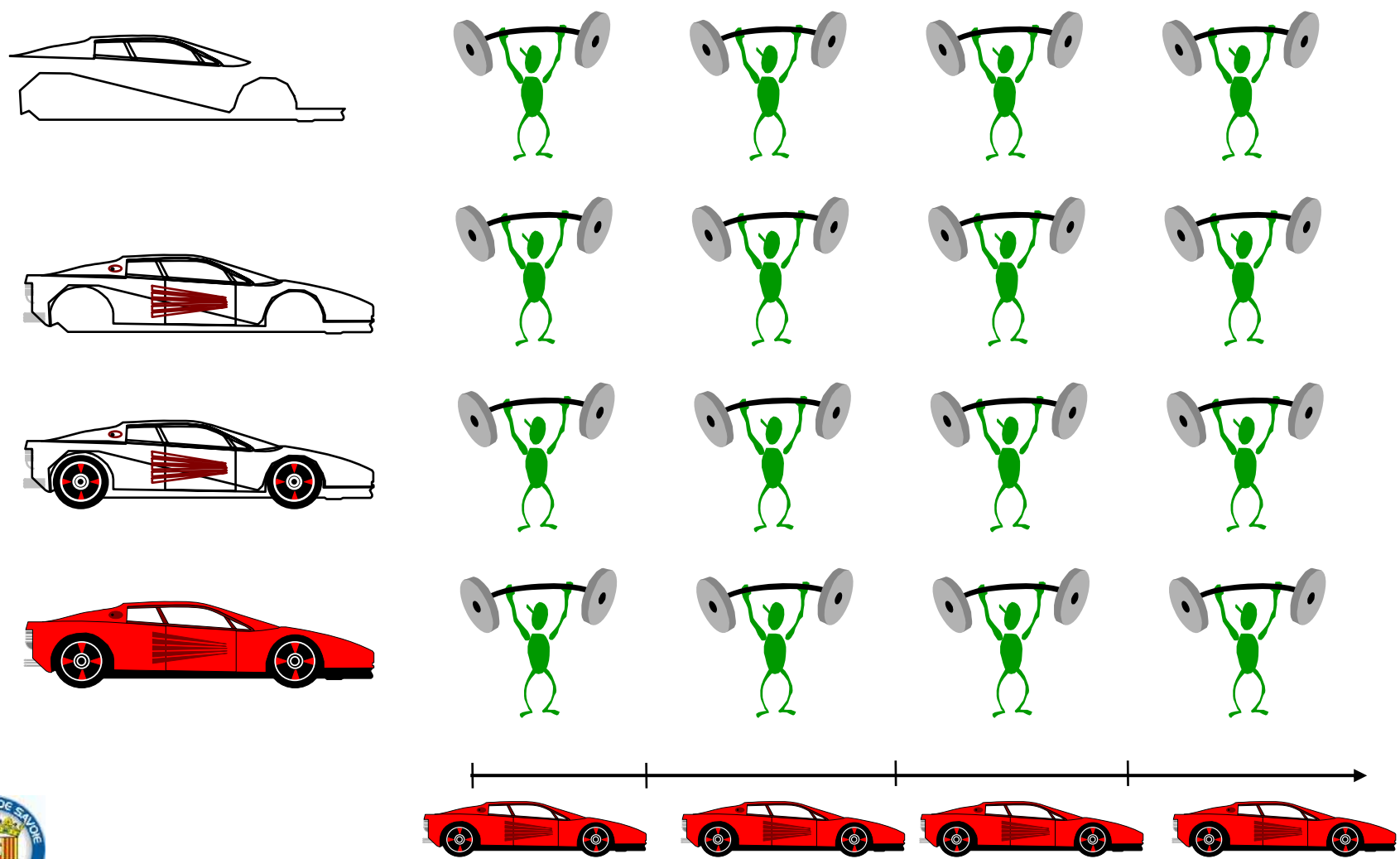
Définition d'un pipeline

Comparaison (2)



Définition d'un pipeline

Comparaison (3)



Définition d’un pipeline

- La technique du pipeline est une technique de mise en oeuvre qui permet à plusieurs instructions de se chevaucher pendant l'exécution.
- Une instruction est découpée dans un pipeline en petits morceaux appelés étage de pipeline.
- La technique du pipeline améliore le débit des instructions plutôt que le temps d'exécution de chaque instruction.
- La technique du pipeline exploite le parallélisme entre instructions d’un flot séquentiel d’instructions. Elle présente l’avantage de pouvoir, contrairement à d'autres techniques d'accélération, être rendue invisible du programmeur.

Chapitre 2 : Le pipeline

- 2.1 Définition d’un pipeline
- 2.2 Les étages d’un pipeline
- 2.3 Les aléas dans le pipeline

Les étages d'un pipeline

P (Prefecth) - Generate program address = Incrémentation du compteur ordinal

F (Fetch) - Get Opcode = Lecture du code de l'instruction en mémoire

D (Decode) - Decode instruction = Décodage de l'instruction

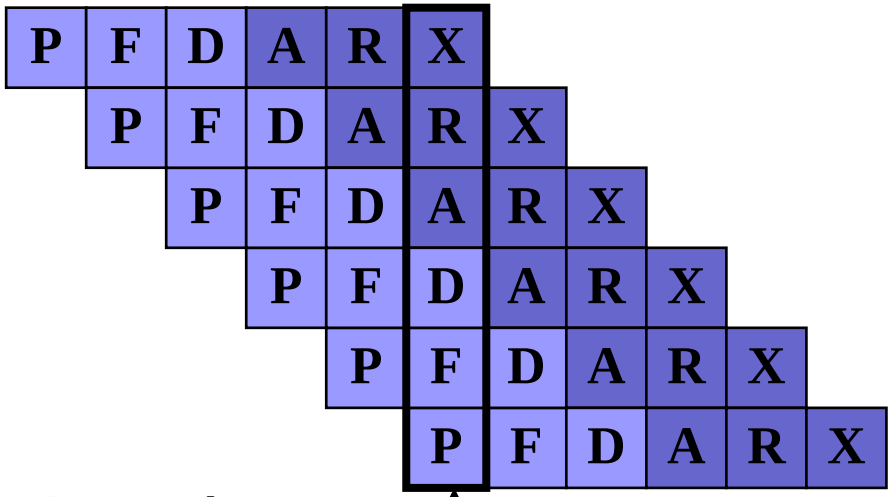
A (Access) - Generate read address = Calcul des adresses des opérandes

= Calcul de l'adresse du résultat

R (Read) - Read operands = Lecture des opérandes en mémoire

X (Execute) = Exécution de l'instruction

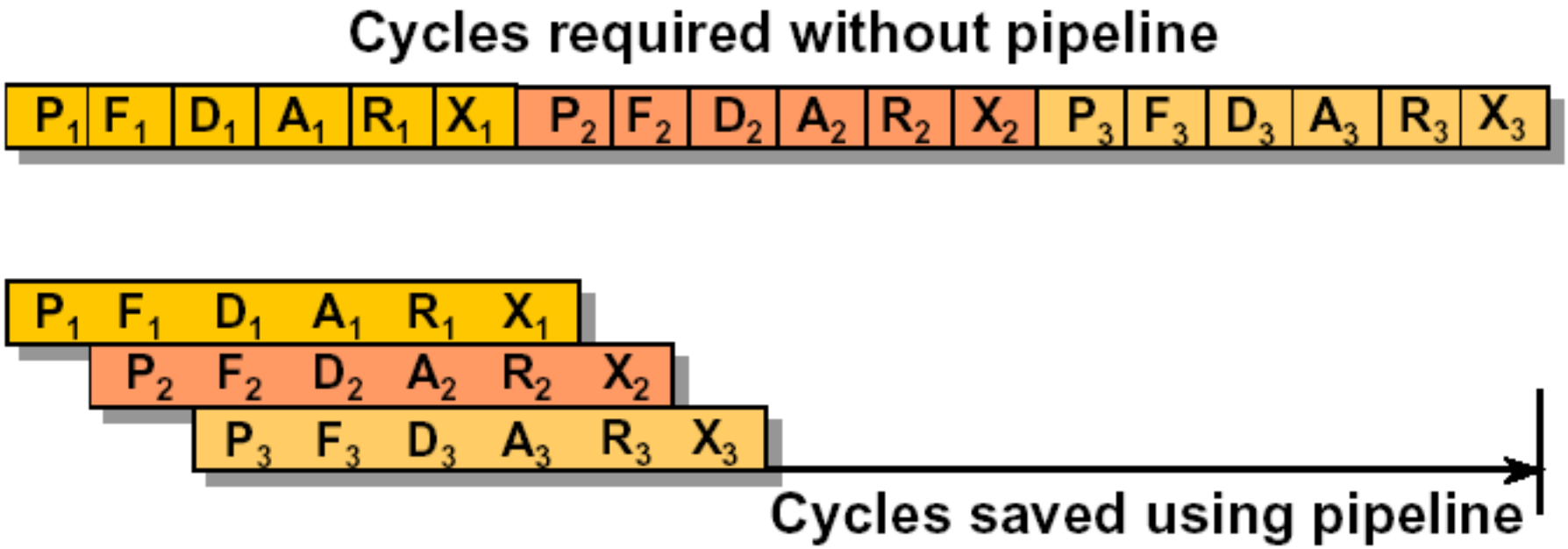
= Ecriture du résultat à l'adresse calculée



Full Pipeline : Toutes les unités matérielles du DSP sont en activités

Les étages d'un pipeline

Comparaison avec et sans pipeline



- Moins de cycles par instruction
- Consommation réduite

Les étages d'un pipeline

Utilisation des ressources par le pipeline

Etage pipeline	Description	Partie hardware utilisée
P	Generate program address	PC
F	Get Opcode	Program memory
D	Decode instruction	Decoder
A	Generate read address	ARs, ARAU
R	Read Operand	Data memory
	Generate write address	ARs, ARAU
X	Execute instruction	MAC, ALU
	Write result	Data Memory

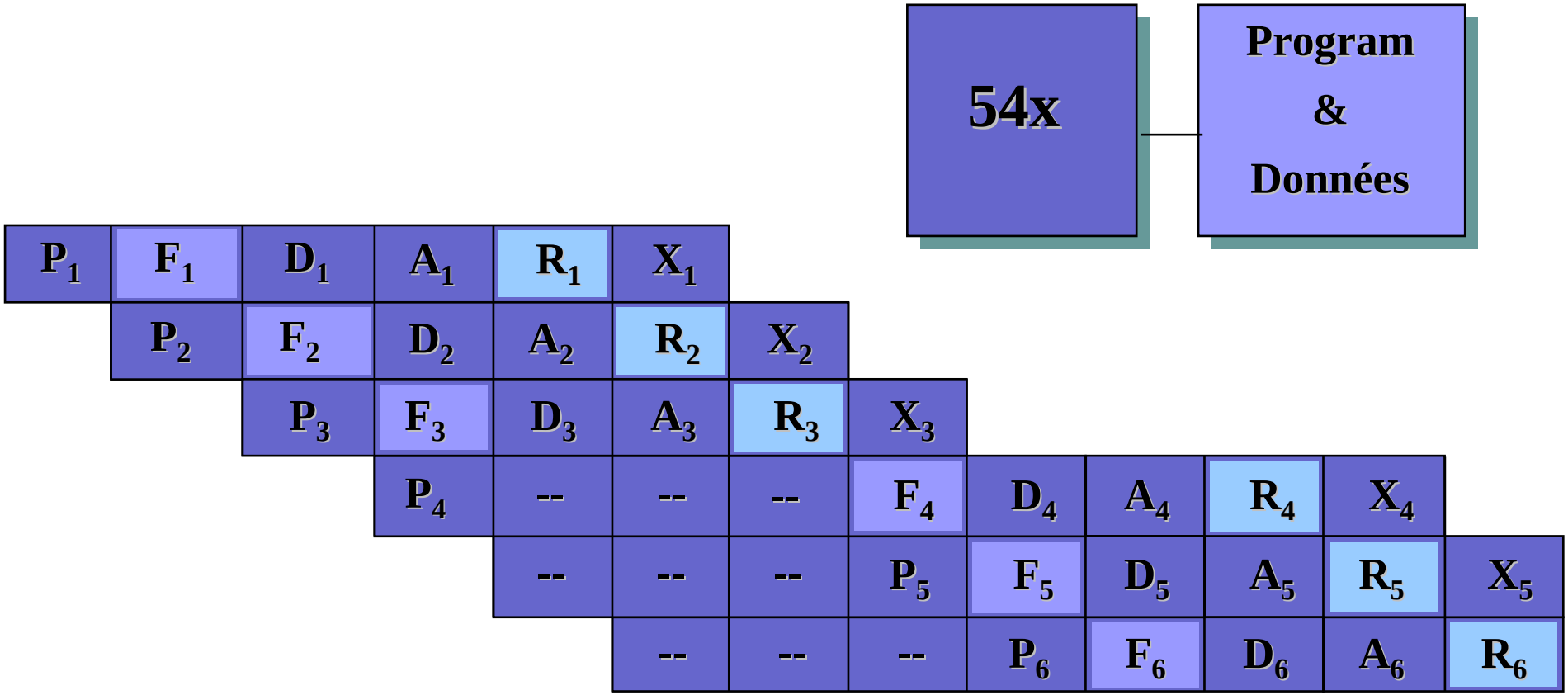
ARAU = Auxiliary Register Arithmetic Unit

Les étages d'un pipeline
Les retards

- Le pipeline atteint son plein rendement une fois qu'il est "rempli"
- Un retard peut se produire
  - S'il existe un conflit de ressources (retard ponctuel)
    - accès à la mémoire
    - utilisation des bus
  - En cas de rupture de séquence (vidange du pipeline)
    - branchement non prévu
    - appel de sous-programme
    - interruption

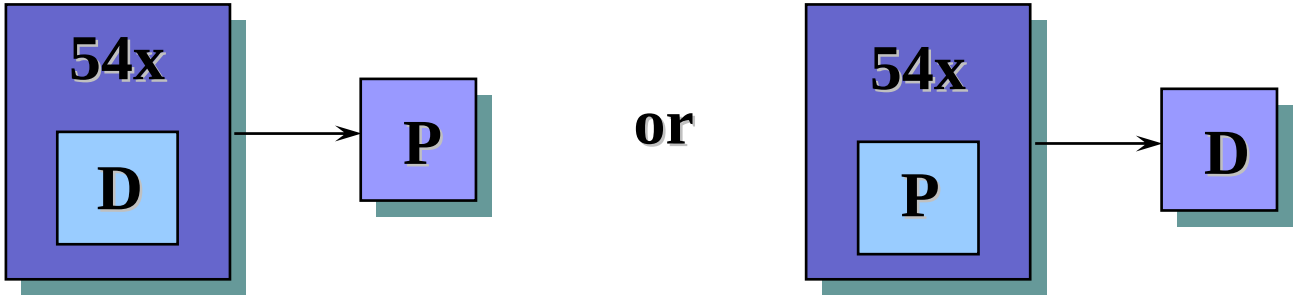
Les étages d'un pipeline

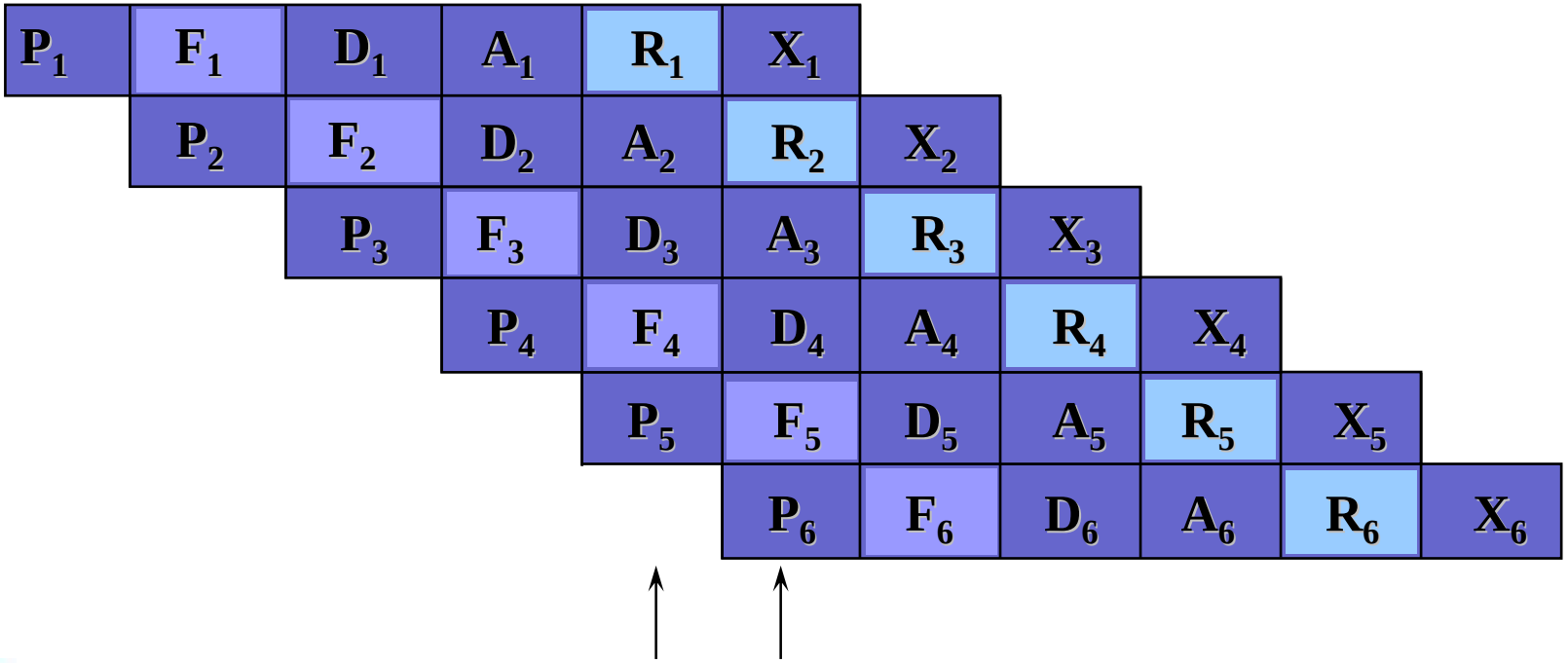
Exemple de rupture



Les étages d'un pipeline

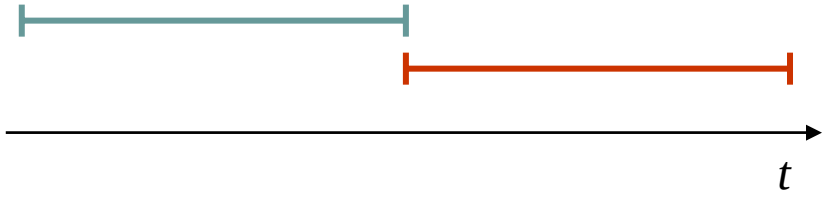
Solution par l'organisation du code



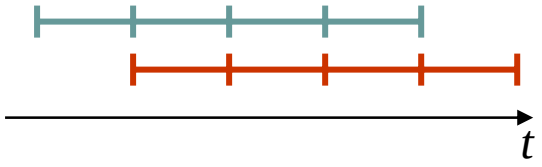


Types de pipelining

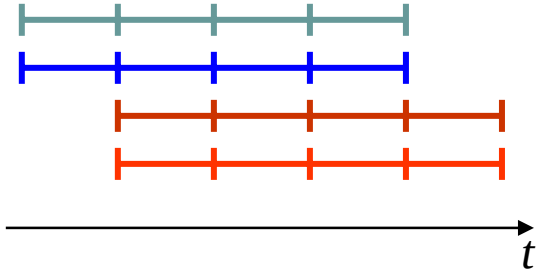
Séquentiel :
Pas de pipeline
(ex: Motorola 56000)



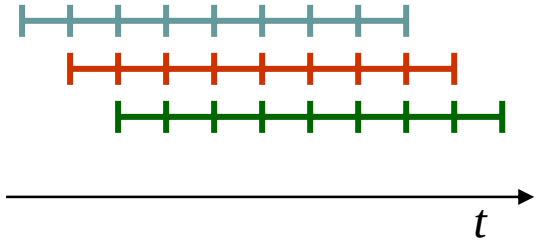
Pipeline simple
(plupart des DSP)



Double pipeline
(ex: Pentium)



Superpipeliné :
Nombre d'étages plus élevé
(ex: TMS320C6000)



Les étages d’un pipeline
Remarques sur les performances

Certaines phases sont inutiles pour certaines instructions (p.ex. un LOAD ne nécessite pas d'exécution), mais toutes les instructions doivent traverser tout le pipeline. Ce "gaspillage" est nécessaire pour simplifier le contrôle.

Les étages d'un pipeline
Exemple de profondeur

Processeur	***Profondeur du pipeline***
Intel Pentium 4 Prescott	31
Intel Pentium 4	20
AMD K10	16
Intel Pentium III	10
AMD Athlon	12
PowerPC G4 (PPC 7450)	7
IBM POWER5	16
IBM PowerPC 970	16
Intel Itanium	10

Chapitre 2 : Le pipeline

- 2.1 Définition d'un pipeline
- 2.2 Les étages d'un pipeline
- 2.3 Les aléas dans le pipeline

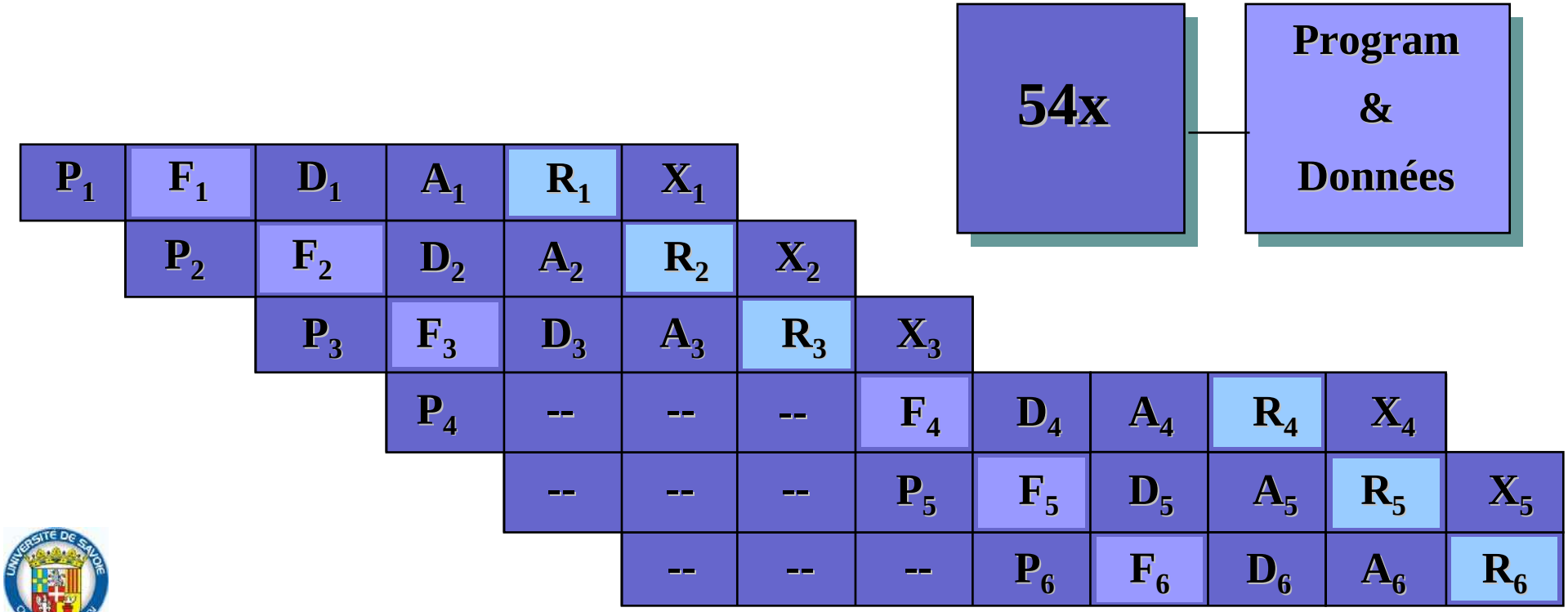
Aléas d'un pipeline
Les types d'aléas

- La présence d'un pipeline (et donc le partage de l'exécution d'une instruction en plusieurs étages) introduit des aléas :
  - Aléas de structure : L'implémentation empêche une certaine combinaison d'opérations (lorsque des ressources matériels sont accédées par plusieurs étages).
  - Aléas de données : Le résultat d'une opération dépend de celui d'une opération précédente qui n'est pas encore terminée.
  - Aléas de contrôle : L'exécution d'un saut conditionnel ne permet pas de savoir quelle instruction il faut charger dans le pipeline puisque deux choix sont possibles.

Les aléas dans le pipeline

Aléas de structures

Les aléas de structure peuvent être éliminés en agissant sur l'architecture du processeur lors de sa conception.



Les aléas dans le pipeline
Aléas de données (1)

- Une instruction ne peut récupérer le résultat de la précédente car celui-ci n'est pas encore disponible.

  Exemple :
  ADD R1, R2, R3 // R1 = R2 + R3
  STORE R1, 1000 // C(1000) = R1

- Cette séquence ne stocke pas à l'emplacement mémoire 1000 la valeur de R1 contenant la somme R2 + R3, mais la valeur de R1 contenue avant l'instruction ADD.

Les aléas dans le pipeline
Aléas de données (2)

- Prenons par exemple la séquence suivante. Cette suite d'instruction possède une dépendance directe simple. En effet A ne peut pas être disponible pour la partie droite de la seconde instruction, puisqu'elle n'est pas encore exécuter lorsque les opérandes de la seconde instruction sont chargés dans le pipeline.
  1. A = B + C
  2. D = A + C
  3. E = F + B

- Une solution consiste à réarranger les instructions. Dans cet exemple, l'opération de la ligne 3 n'a aucune interdépendance avec les deux précédentes. Le code modifié sera :
  1. A = B + C
  2. E = F + B
  3. D = A + C

Les aléas dans le pipeline
Aléas de données (3)

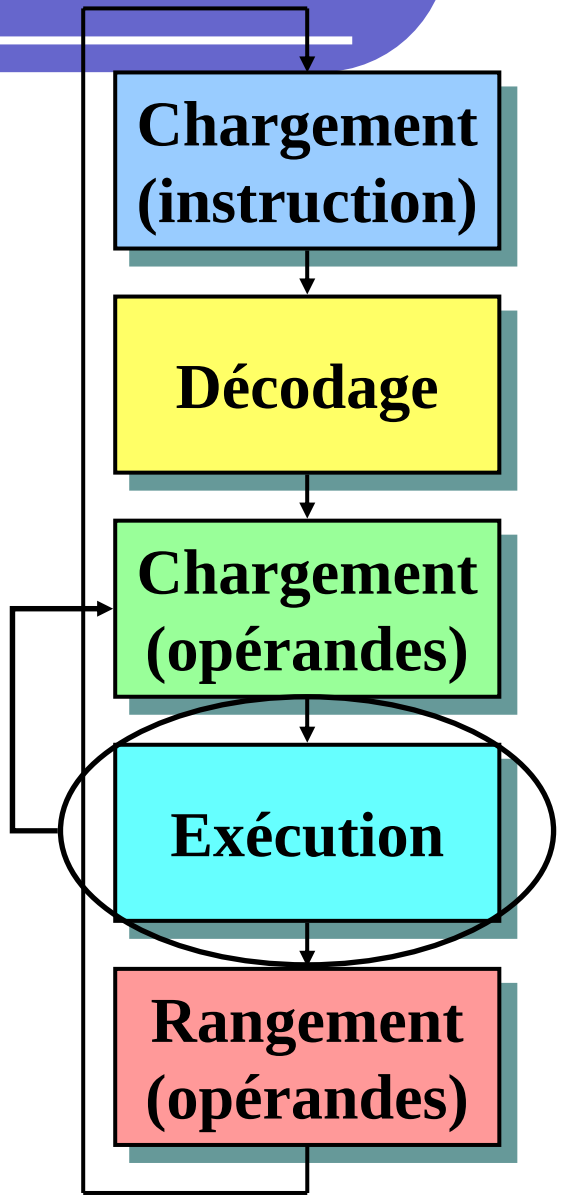
- Si nécessaire, les instructions intercalées peuvent être des NOP.
  1. A = B + C
  2. NOP
  3. D = A + C
  4. E = F + B

- Remarques :
  - Le compilateur n'est pas toujours en mesure de détecter les aléas (par exemple, si les aléas concernent des pointeurs).
  - Le nombre d'instructions à intercaler dépend de la structure (nombre d'étages) du pipeline.
  - La complexité du compilateur en est fortement augmentée.

Les aléas dans le pipeline
Aléas de données (4)

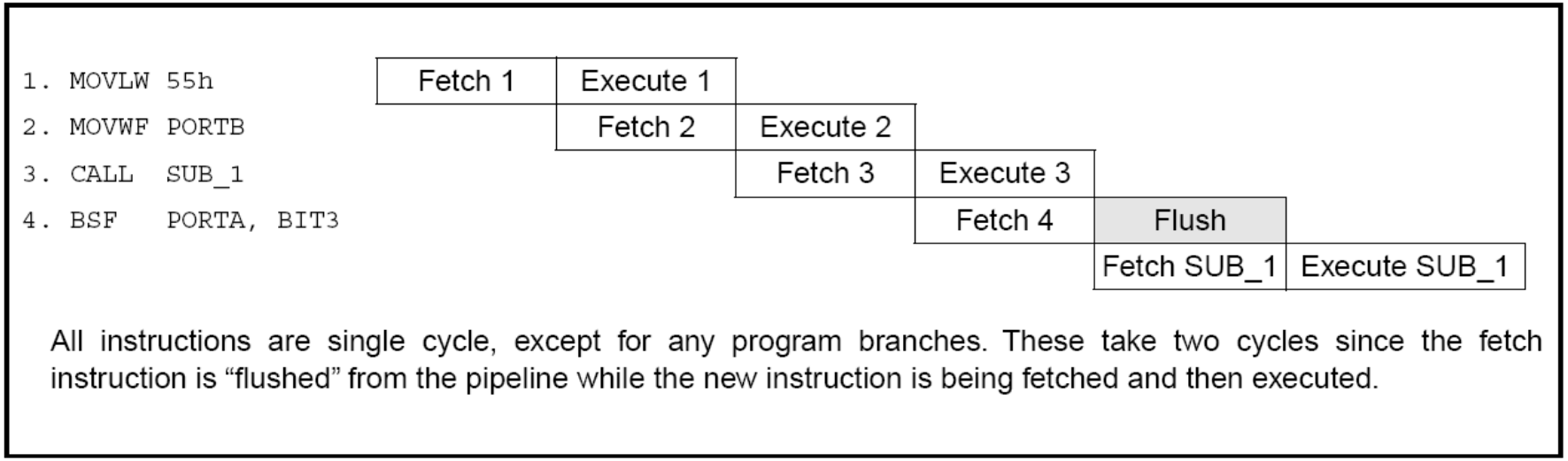
La fréquence élevée d'aléas de données peut justifier l'introduction de matériel supplémentaire. On peut par exemple introduire une connexion directe entre la sortie de l'étage d'exécution et l'étage de chargement des opérandes. Ceci permet au résultat d'une instruction d'être un opérande de l'instruction suivante.

Ci contre un pipeline pouvant réaliser cette solution.



Les aléas dans le pipeline
Aléas de contrôle (1)

La présence d'un pipeline introduit des complications lors de l'exécution d'un saut ou d'un saut conditionnel. L'étage de décodage de l'instruction n'est pas en mesure de calculer l'adresse de l'instruction suivante avant de connaître le résultat de l'instruction précédente.



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

Les aléas dans le pipeline
Aléas de contrôle (2)

- Une solution possible est de faire en sorte que le processeur devine si le branchement sera pris ou pas pris (branch prediction) et commencer à exécuter les instructions correspondant à cette décision.
  - Si le choix se révèle correct, la pénalité de branchement est éliminée.
  - Si le choix se révèle incorrect, il faudra vider le pipeline et charger l'instruction correcte.

- Pour faire de la prédiction de branchement il y a deux possibilités :
  - Solution statique : La direction du branchement est fixe, définie en matériel au moment de la conception du processeur.
  - Solution dynamique : La direction du branchement est définie au moment de l'exécution du programme, sur la base d'une analyse du code.

Les aléas dans le pipeline
Aléas de contrôle (3)

Solution statique :

- Les sauts en arrière (boucles) sont plus souvent pris que pas pris. En effet, une boucle est souvent réaliser avec plus que 2 itérations.

=>On peut donc faire une prédiction selon la direction:

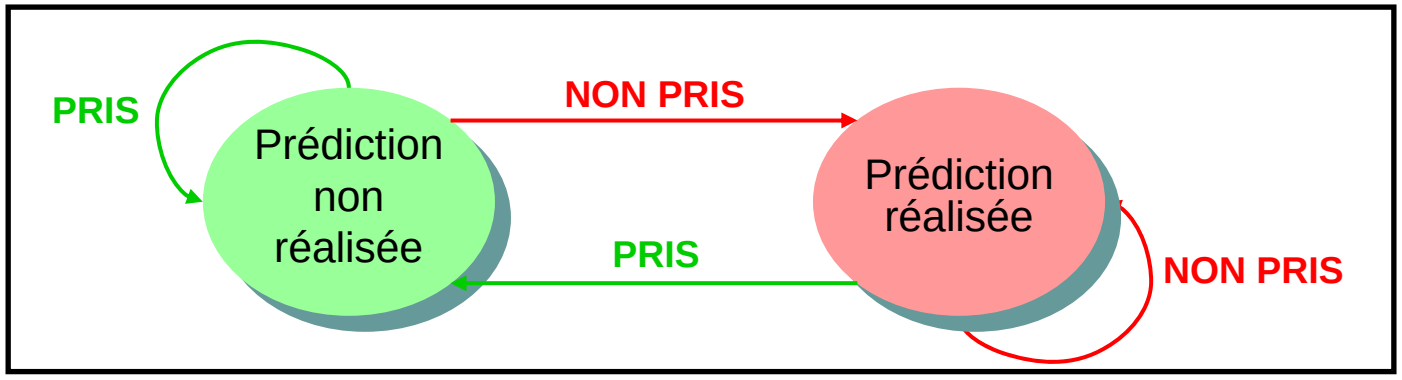
- si le saut est en arrière, il est pris,
- s'il est en avant, il n'est pas pris.

Cette stratégie donne des très bons résultats (70-80%) avec une augmentation relativement restreinte de la logique de contrôle: elle est utilisée dans plusieurs processeurs (p.ex., MicroSparc, HP-PA).

Les aléas dans le pipeline
Aléas de contrôle (4)

Solution dynamique

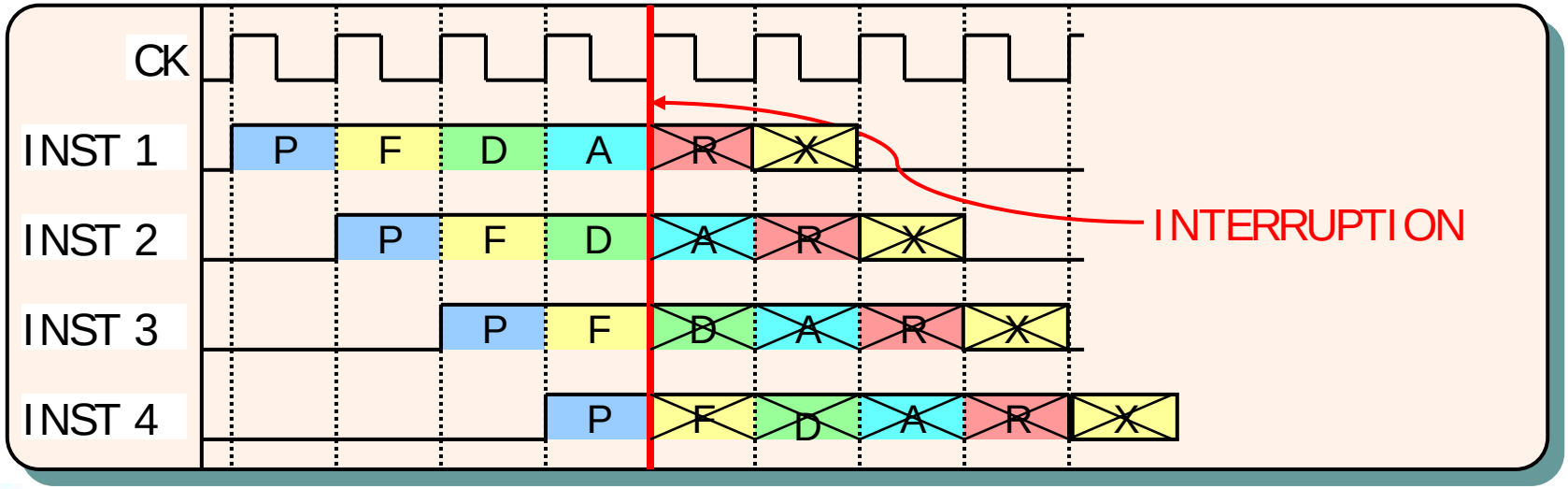
- Pour réaliser une prédiction dynamique le processeur mémorise le comportement du programme lors de l'exécution des sauts. À chaque exécution d'un branchement dans un programme, le processeur mémorise si le saut était pris ou pas pris dans un tampon de prédiction de branchement. Sur la base du comportement passé du programme pour un branchement donné, le processeur prédit son comportement pour l'exécution suivante du même saut.
- Par rapport à la prédiction statique, la prédiction dynamique est plus performante, mais nécessite une quantité très importante de logique de contrôle.



Machine d'état présente dans le processeur pour la prédiction de branchement statistique

Les aléas dans le pipeline
Gestion des interruptions

La présence d'un pipeline complique le traitement des interruptions: lors du déclenchement d'une interruption non-masquable, la routine de traitement doit parfois être lancée immédiatement. Le pipeline contiendra alors des instructions partiellement exécutées.



Les aléas dans le pipeline
Résumé

- Le pipeline améliore le débit mais pas le temps par instruction : il faut toujours cinq cycles à une instruction d'un pipeline à cinq étages pour s'exécuter.

- Les dépendances de données et de contrôle dans les programmes imposent une limite supérieure au gain que peut générer le pipeline car le processeur doit parfois attendre la fin d'une instruction pour que les dépendances soit résolues.

- On peut élever cette limite, mais pas l'éliminer, en réduisant les aléas de contrôle par des optimisations, et les aléas de données par un ordonnancement des instructions par le compilateur.

Chapitre 3 : Les mémoires caches

- 3.1 Objectif et principe d'une mémoire cache
- 3.2 Où placer un bloc?
- 3.3 Comment un bloc est-il trouvé?
- 3.4 Quel bloc remplacé lors d'un défaut?
- 3.5 Comment sont traitées les écritures?

Objectifs et principes du cache

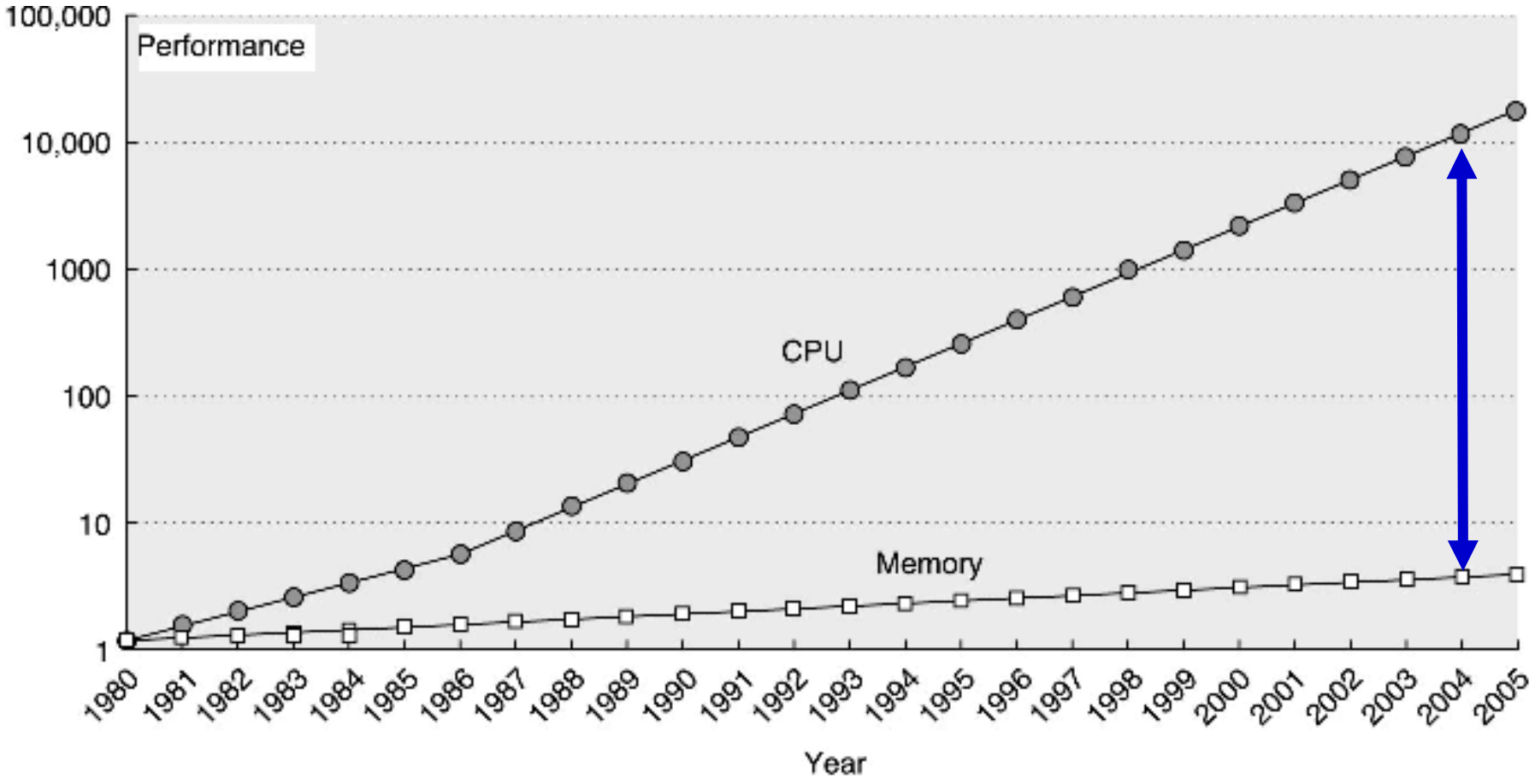
- Les mémoires doivent répondre à deux contraintes contradictoires :
  - Taille importante
  - Temps d'accès court
- Principe de base du cache :
  - Les mots mémoires les plus fréquemment utilisés sont conservés dans une mémoire rapide (cache) plutôt que dans une mémoire lente (mémoire centrale).

Objectifs et principes du cache
Vitesse des mémoires et des processeurs (1)

- **Évolution**

Année	Temps de cycle processeur	Temps de cycle mémoire
1990	~100ns	~140ns
1998	~4ns	~60ns
2002	~0.6ns	~50ns

Objectifs et principes du cache
Vitesse des mémoires et des processeurs (2)



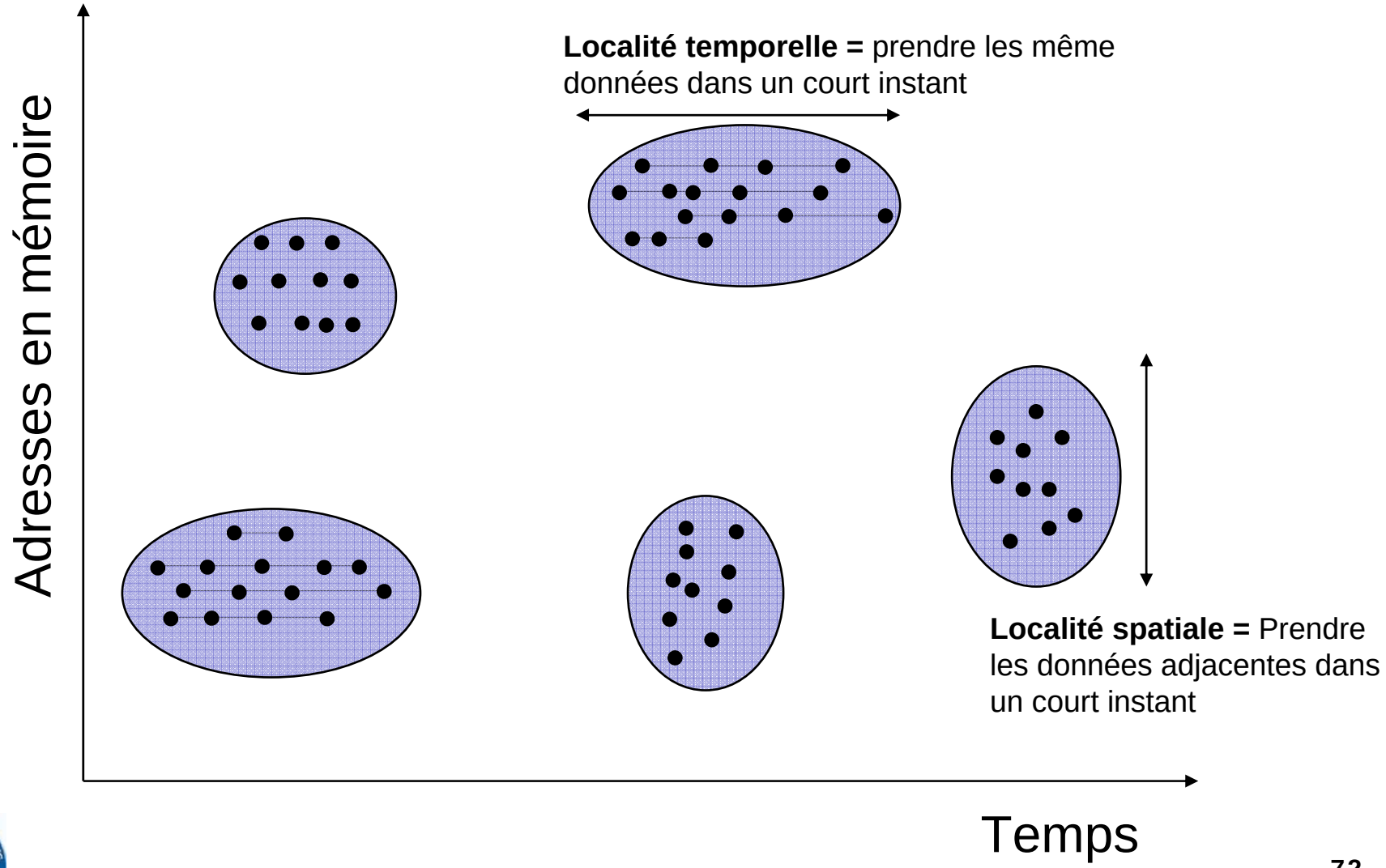
Objectifs et principes du cache
Principe de localité (1)

- Localité spatiale :
  - Tendance à accéder aux données qui sont proches de celles récemment utilisées

- Localité temporelle :
  - Tendance à réutiliser des données récemment utilisées

Objectifs et principes du cache

Principe de localité (2)



Objectifs et principes du cache
Principe de localité (3)

- Les données

```
for (i=0; i<N; i++) {
  for (j=0; j<N; j++) {
    y[i] = y[i] + a[i][j] * x[j]
  }
}
```

- y[i]: propriétés de localités temporelle et spatiale.
- a[i][j]: propriété de localité spatiale.
- x[j]: propriété de localité temporelle et spatiale.

- Le programme

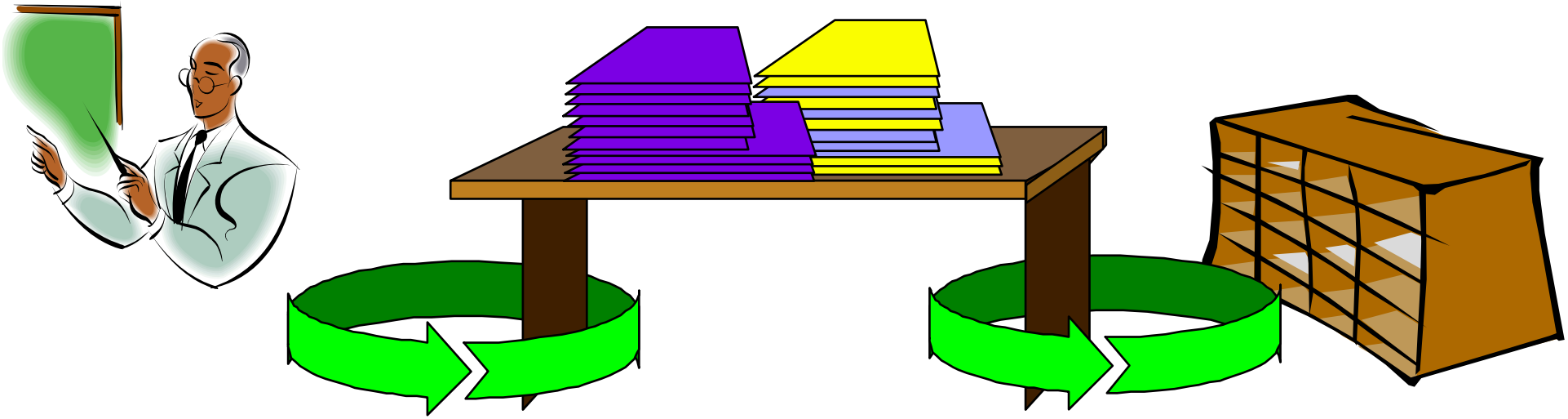
```
                ...
05  LOOP        LDR R1, R0, #3
06              ADD R1, R1, #5
07              STR R1, R0, #30
08              ADD R0, R0, #1
09              ADD R3, R0, R2
0A              BRn LOOP
                ...
```

Boucle : réutilisation des instructions : localité temporelle

Instructions consécutives en mémoire : localité spatiale

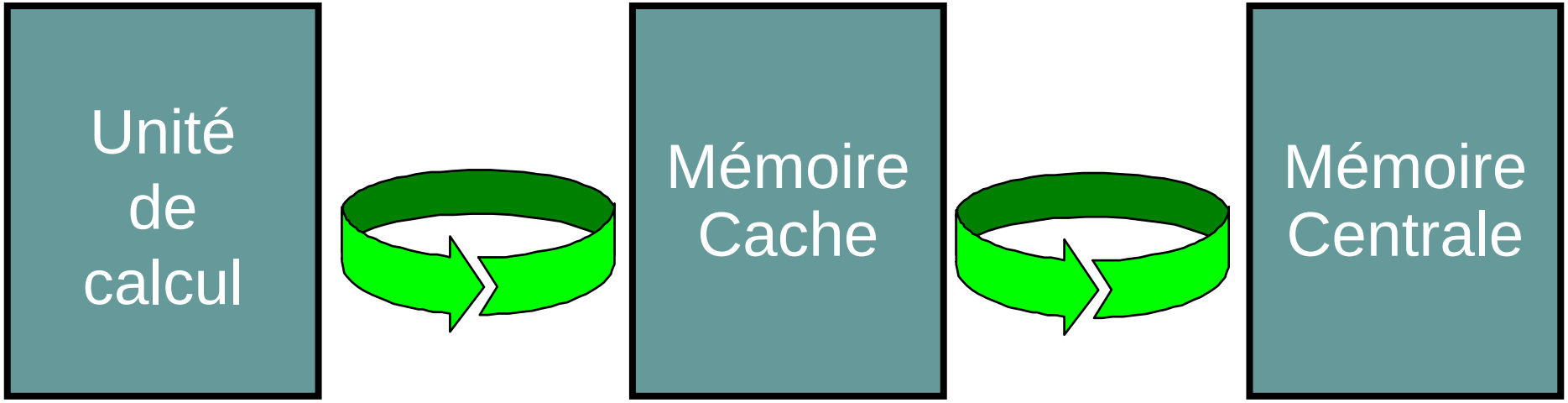
Objectifs et principes du cache
Analogie

- Homme = Unité de calcul
- Le bureau = Mémoire cache
- La bibliothèque = Mémoire centrale



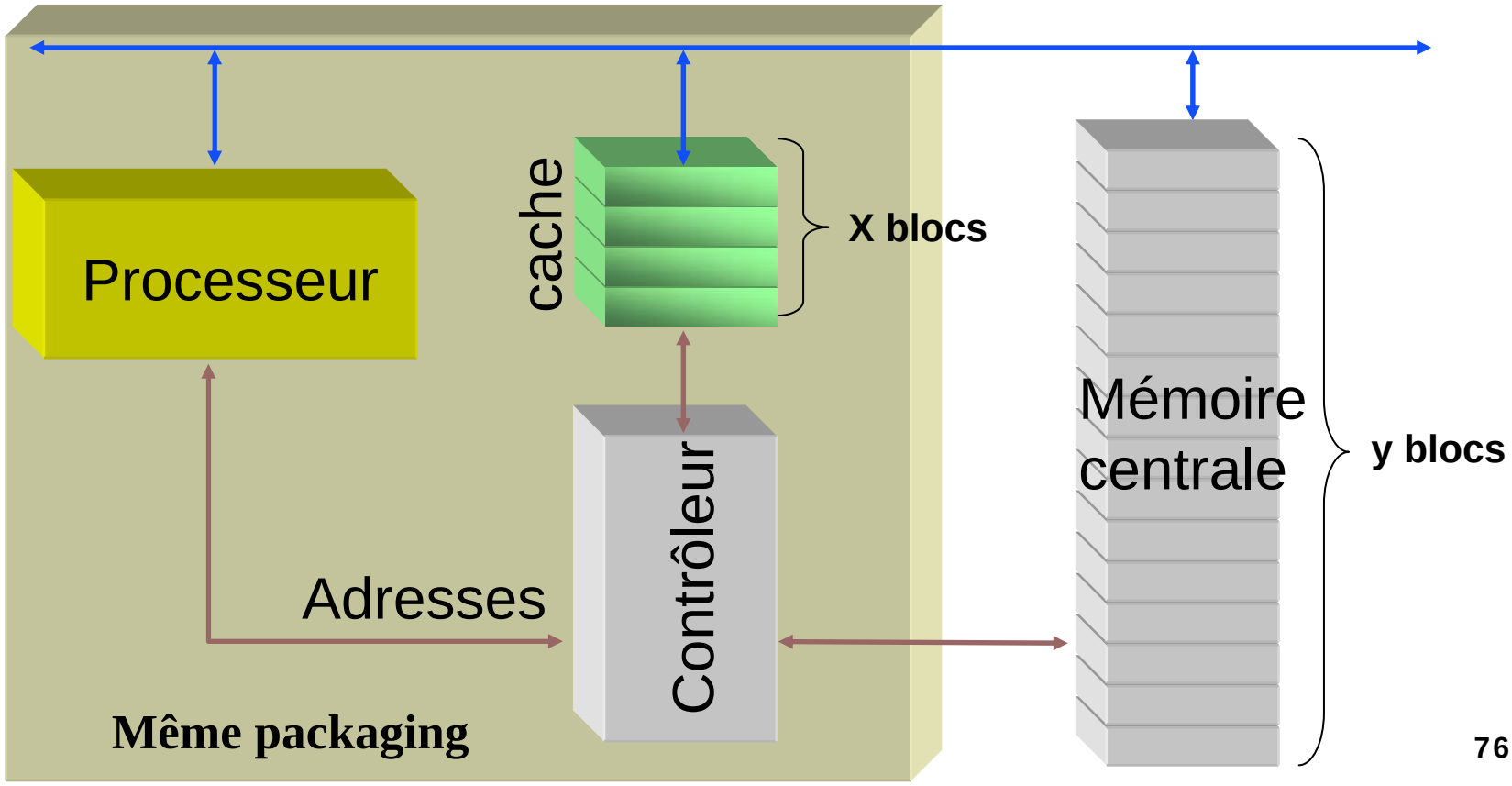
Objectifs et principes du cache

L'élément de base est le bloc
4 octets < Nbre d'octet par blocs < 128 octets)



Objectifs et principes du cache
Organisation mémoire

- La mémoire cache possède x blocs
- La mémoire centrale possède y blocs
- y >> x

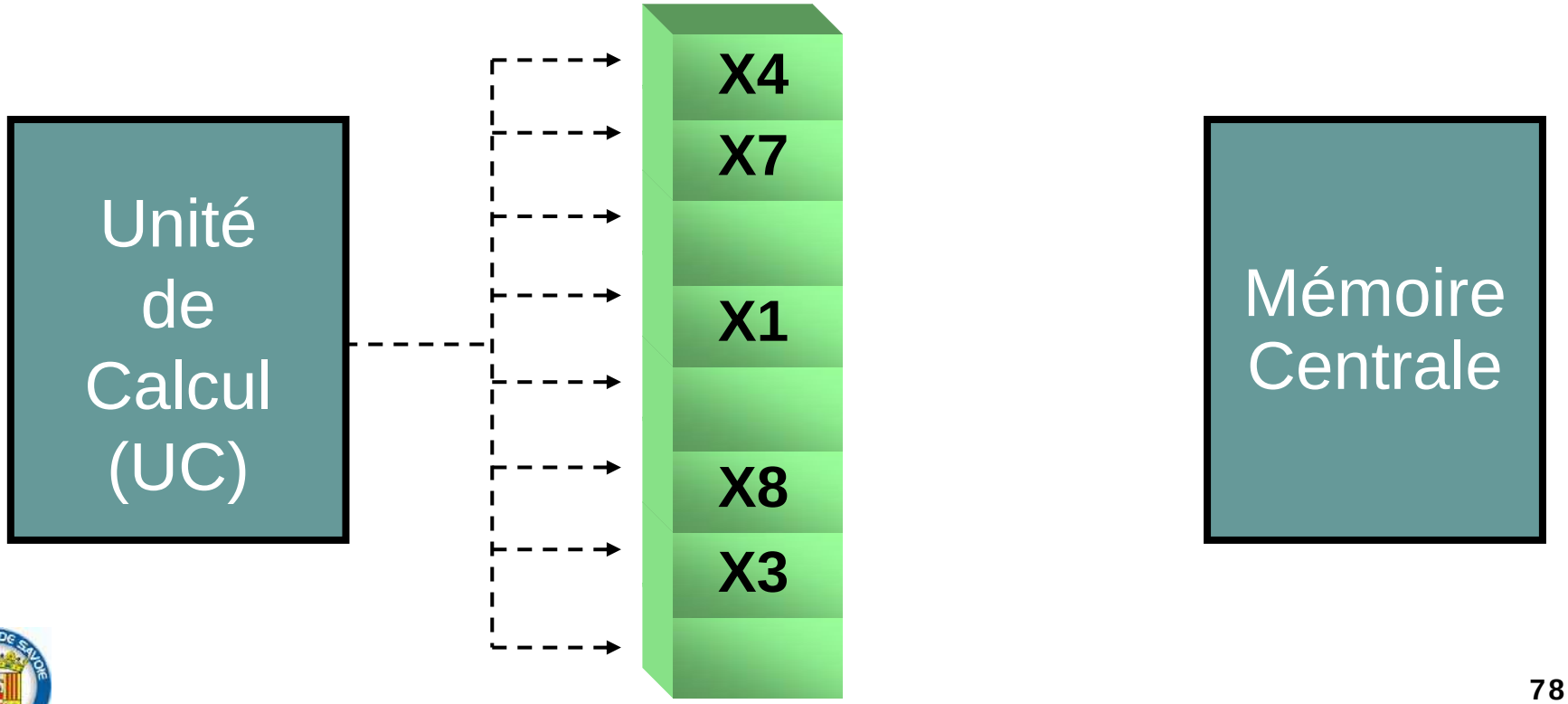


Objectifs et principes du cache
Placement des données dans le cache

- Le placement des données dans le cache est géré par le matériel :
  - le programmeur n'a pas à se soucier du placement des données dans le cache
  - En revanche le programmeur devra prendre en considération la présence du cache pour optimiser les performances.
  - le fonctionnement du cache est transparent pour le programmeur.

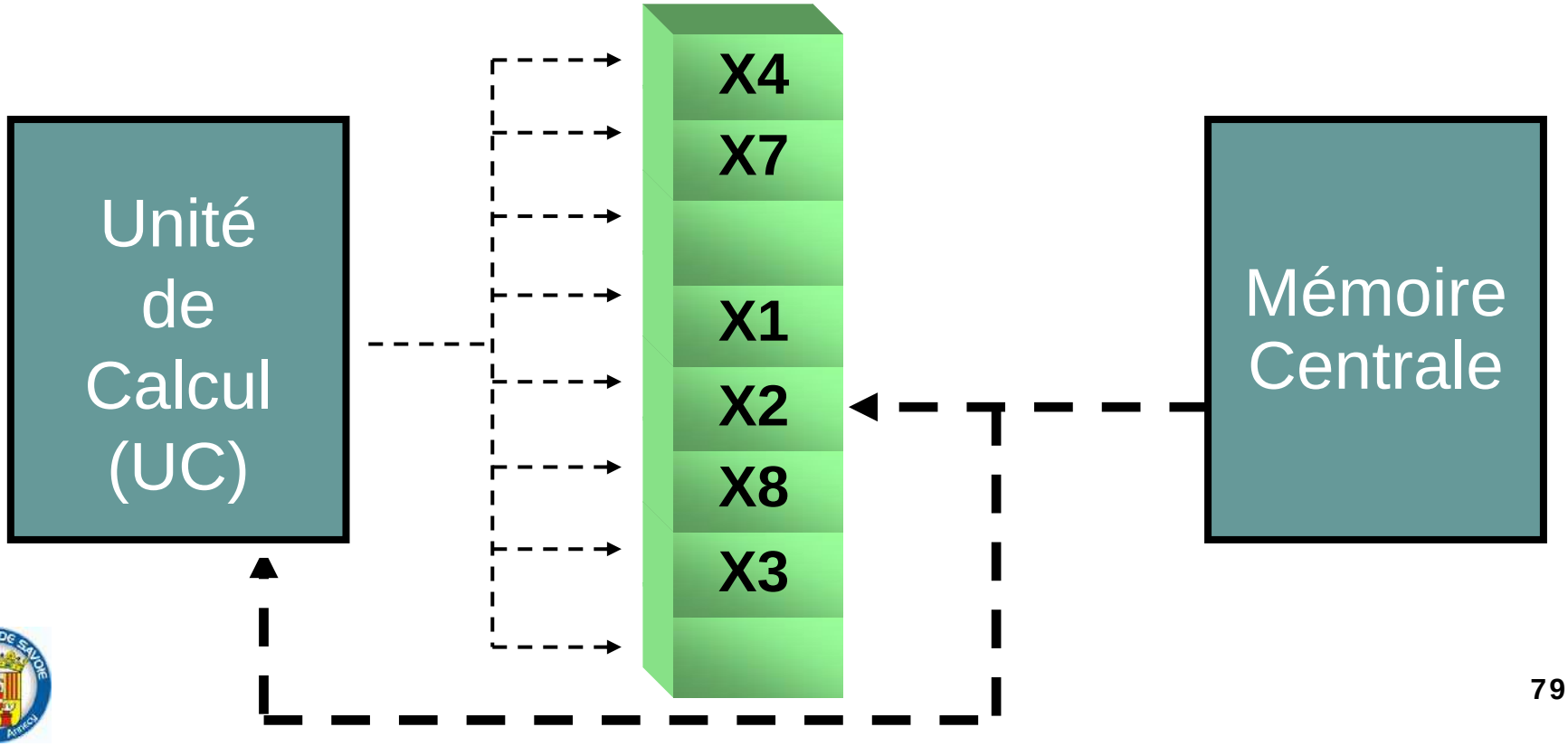
Objectifs et principes du cache
Principe général (1)

- L'UC veut faire référence à un bloc X2 dans le cache
  - Recherche de X2 dans le cache
  - => Défaut de cache



Objectifs et principes du cache
Principe général (2)

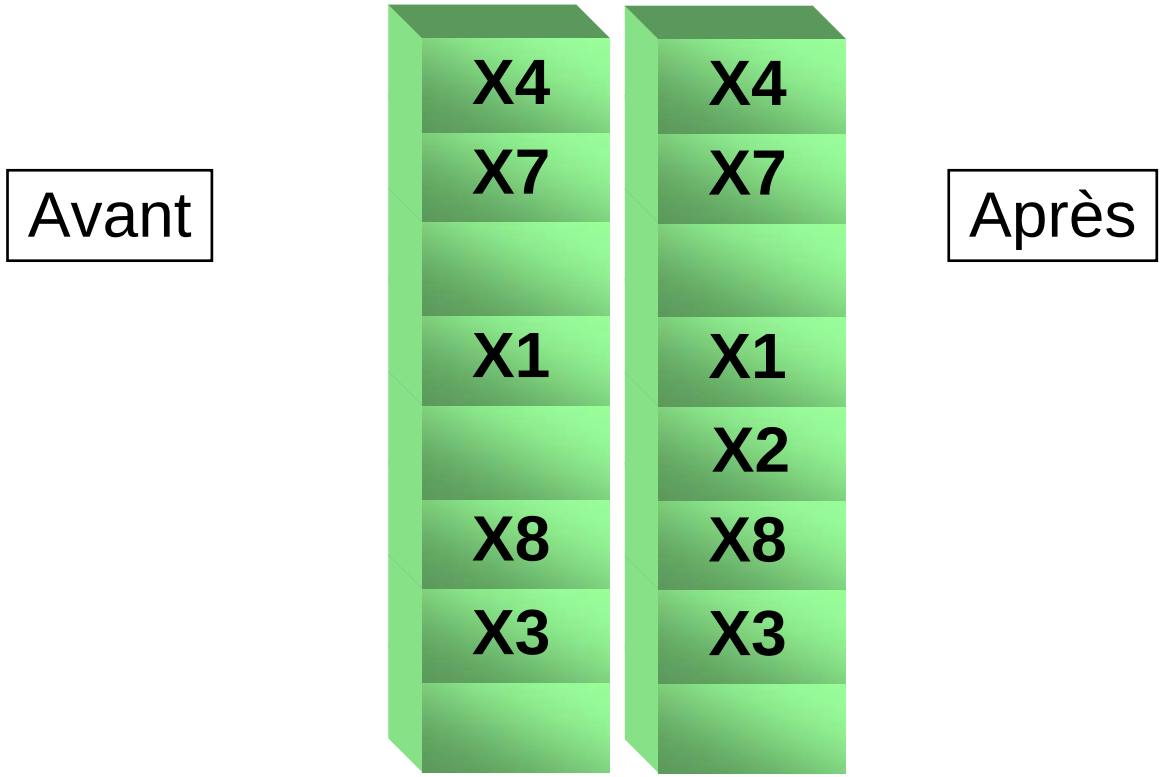
- L'UC veut faire référence à un bloc X2 dans le cache
  - Extraction de X2 de la mémoire centrale



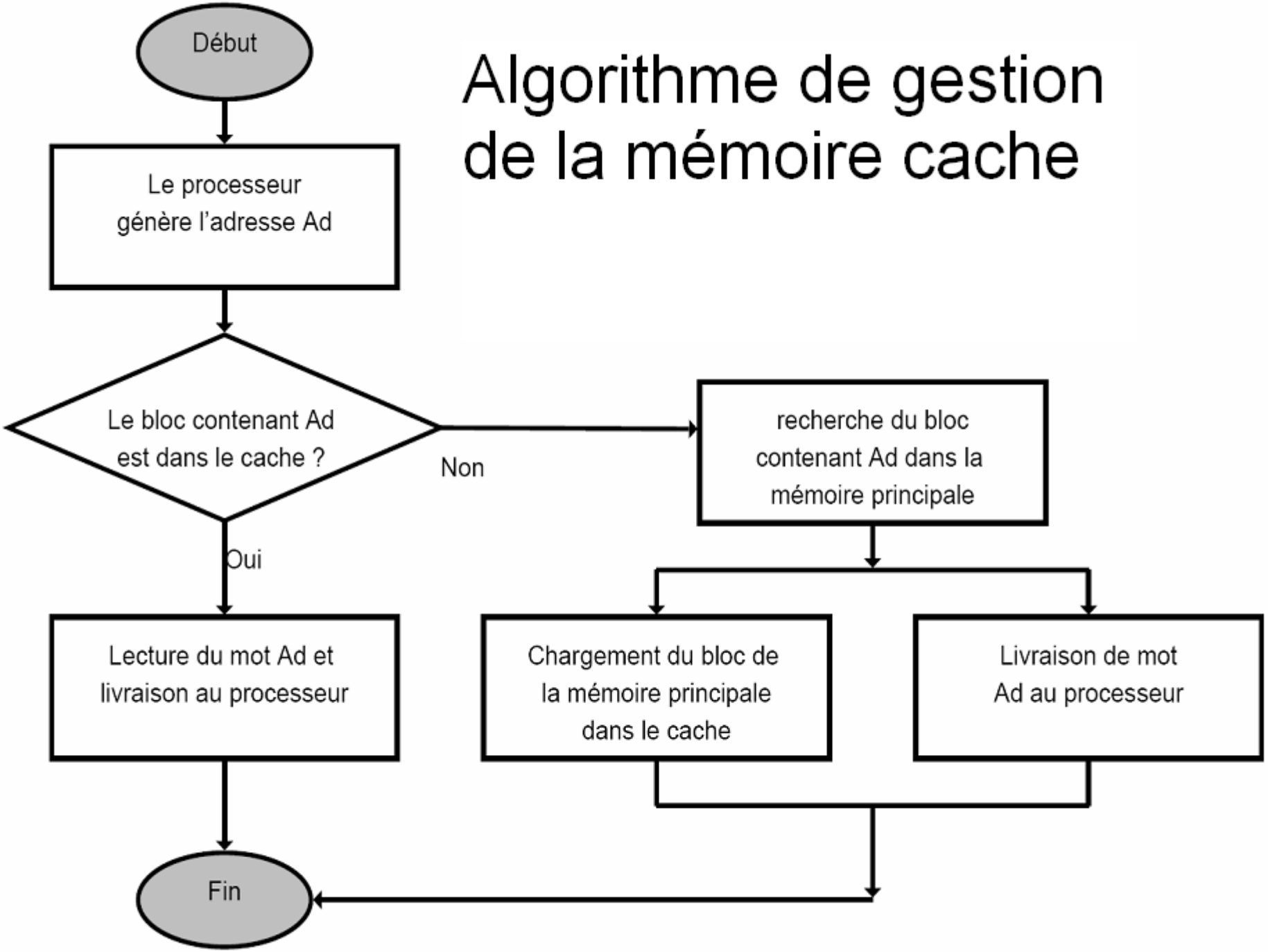
Objectifs et principes du cache

Principe général (3)

- Il y a eu transfert d'un nouveau bloc (X2) de la mémoire centrale, dans la mémoire cache.



Algorithme de gestion de la mémoire cache



Objectifs et principes du cache

Bloc ou ligne de cache (1)

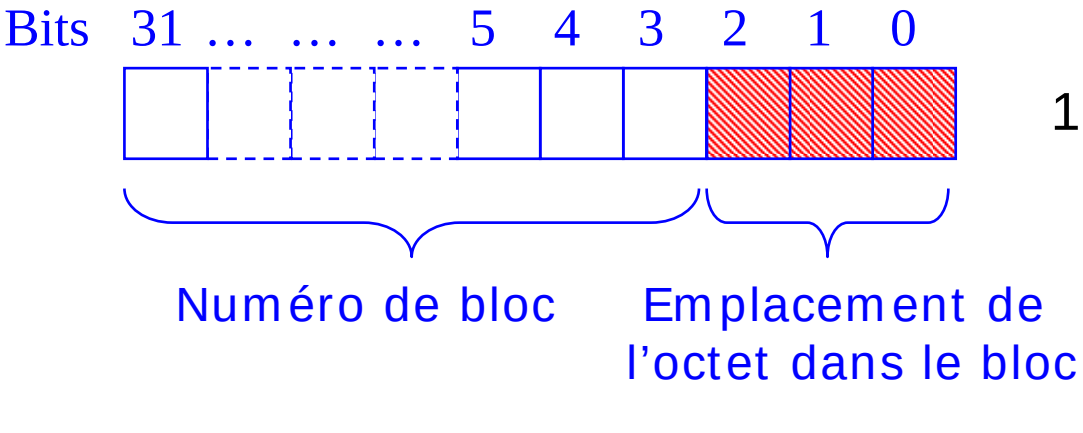
- L'unité d'information qui peut être présente ou non dans le cache est appelée un bloc, qui constitue une ligne (ou rangée) du cache. Les blocs ont généralement entre 4 et 128 octets et sont tous de même taille dans un cache donné.

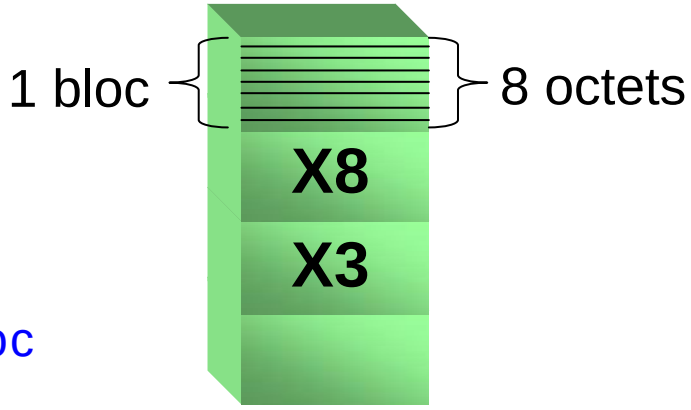
- La mémoire centrale et la mémoire cache ont impérativement les même tailles de blocs.

Objectifs et principes du cache
Bloc ou ligne de cache (2)

- L'adresse fournie par le processeur peut être scindée en deux parties : le n° de bloc et l'adresse dans le bloc.

- Exemple : @ sur 32 bits et blocs de 8 octets





Objectifs et principes du cache
Étiquettes

- À chaque bloc on associe
  - Une étiquette :
    - La valeur de cette étiquette permettra de décider si un bloc donné est effectivement dans le cache ou non.
  - Un bit de validité :
    - Il permet de savoir si les données du bloc sont obsolètes ou pas.

Chapitre 4 : Les mémoires caches

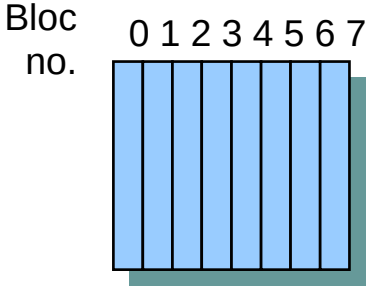
- 4.1 Objectif et principe d'une mémoire cache
- 4.2 Où placer un bloc?
  - **Caches totalement associatifs**
  - **Caches à correspondance directe**
- 4.3 Comment un bloc est-il trouvé?
- 4.4 Quel bloc remplacé lors d'un défaut?
- 4.5 Comment sont traitées les écritures?

Où placer un bloc?
Différentes organisations de cache

Où placer une ligne de la mémoire principale dans le cache?

Complètement associatif

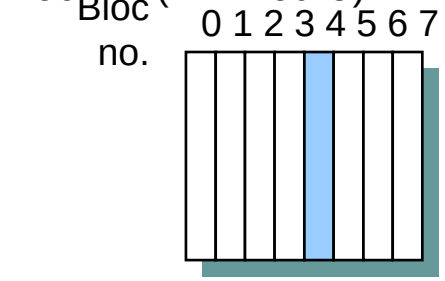
Bloc no.



Cache à correspondance direct

Bloc : 4 (12 mod 8)

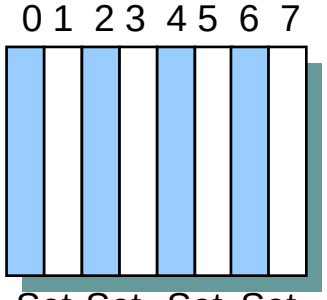
Bloc no.



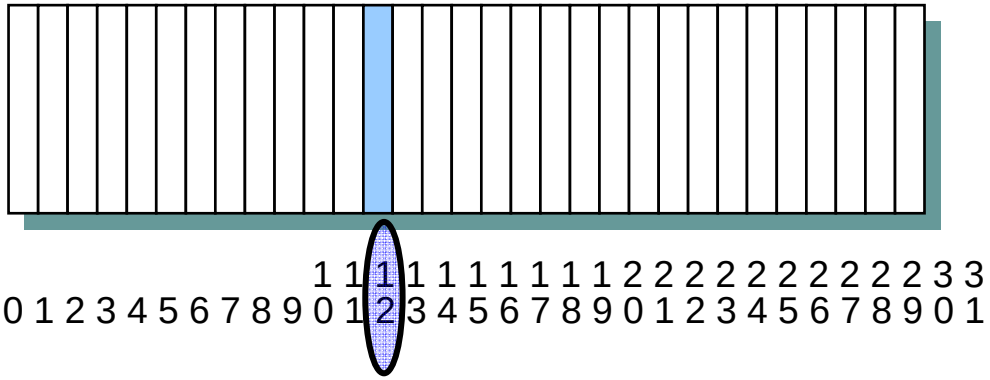
Cache associatif par ensemble de bloc

0 (12 mod 2)

Bloc no.

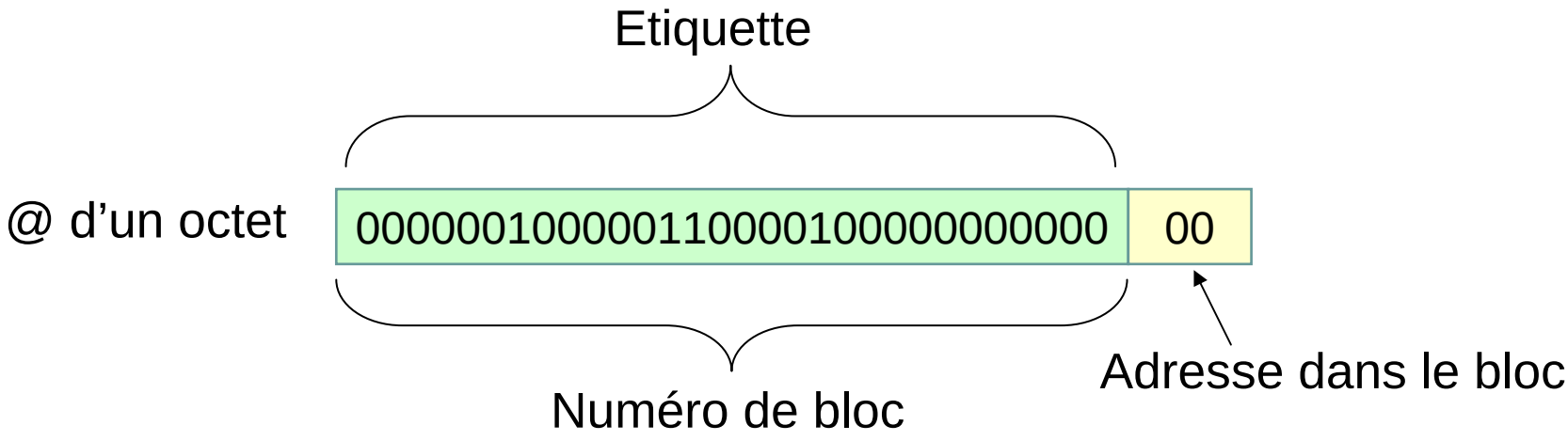


Block-frame address

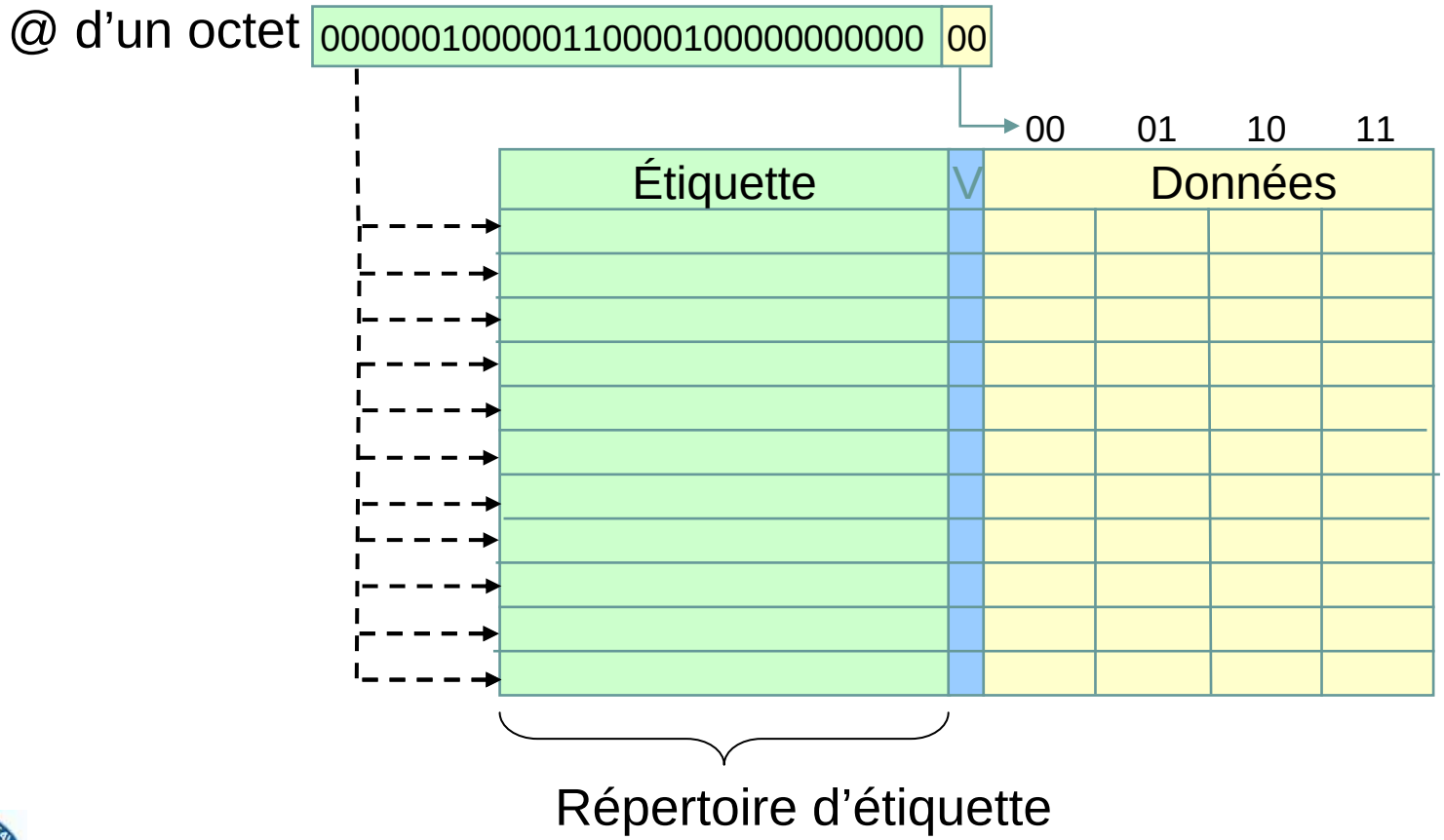


Cache associatif

- Le numéro de bloc est utilisé comme étiquette. Les étiquettes sont stockées dans un répertoire en même temps que les données
- Un bloc peut être placé n'importe où dans la mémoire cache.

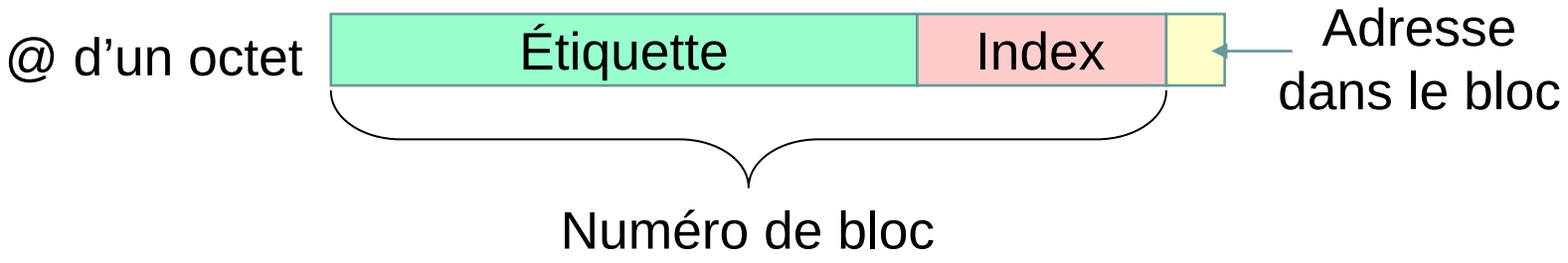


Cache associatif



Cache à correspondance directe

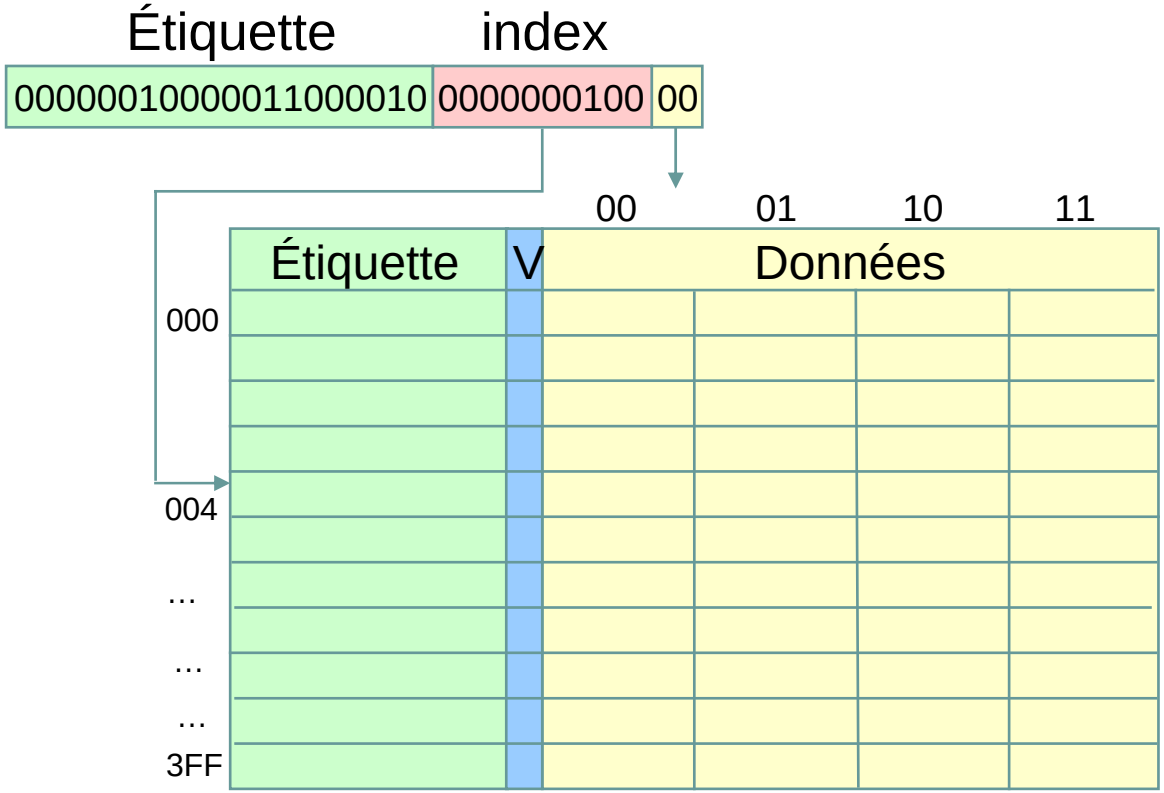
- Dans le cache à accès direct, le champ « numéro de bloc » est scindé en deux parties : l'étiquette et l'index. L'étiquette et les données correspondantes sont toujours enregistrées dans la rangée donnée par le champ index.



Cache à correspondance directe

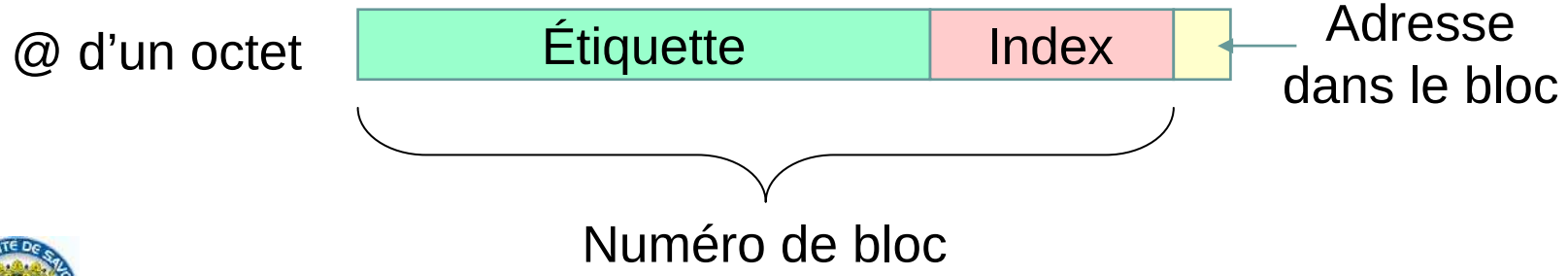
- L'index donne directement la position dans le cache

$$\text{Nbr ligne de cache} = 2^{\text{Nbre bits d'index}}$$



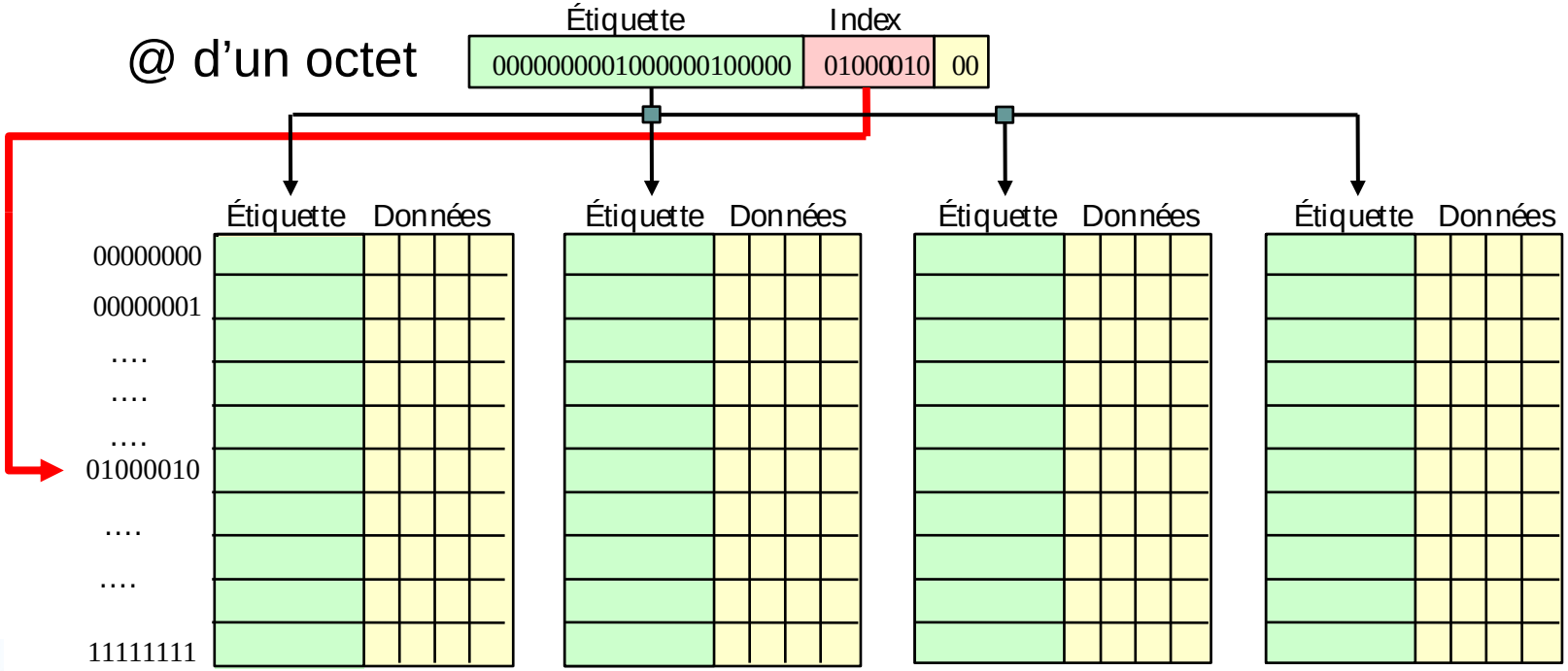
Cache associatif par ensemble de blocs

- Le cache associatif par ensemble de blocs est un compromis entre le cache purement associatif et le cache à correspondance directe. Le choix d'un ensemble est associatif. Cependant, chaque ensemble est géré comme dans le cache à correspondance direct.



Cache associatif par ensemble de blocs

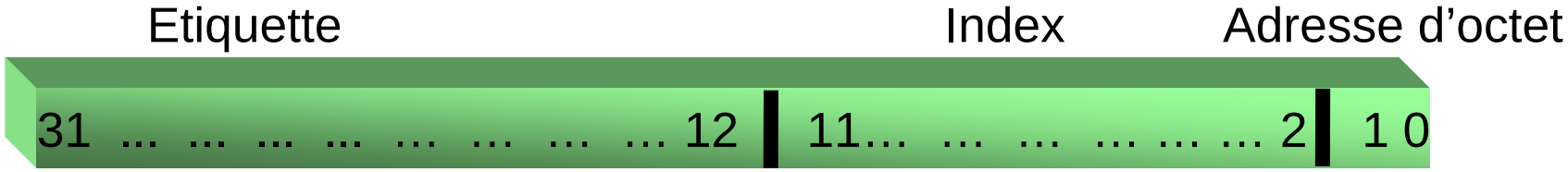
- Les données peuvent être rangées dans n'importe quelle ensemble (associatif)
- Par contre l'index indique la ligne à laquelle on stocke la donnée (correspondance directe)



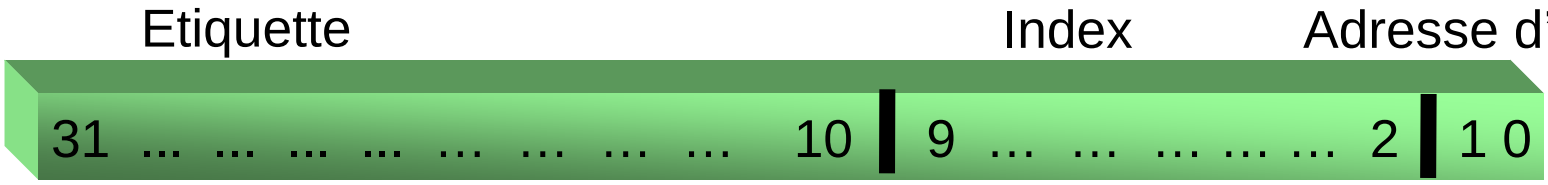
Les étiquettes en fonction du type de caches

- Cache associatif

- Cache à correspondance directe



- Cache associatif par ensemble de bloc



Chapitre 4 : Les mémoires caches

- 4.1 Objectif et principe d'une mémoire cache
- 4.2 Où placer un bloc?
- 4.3 Comment un bloc est-il trouvé?
  - **Caches totalement associatifs**
  - **Caches à correspondance directe**
- 4.4 Quel bloc remplacé lors d'un défaut?
- 4.5 Comment sont traitées les écritures?

Comment un bloc est-il trouvé?
Recherche d'un bloc dans le cache

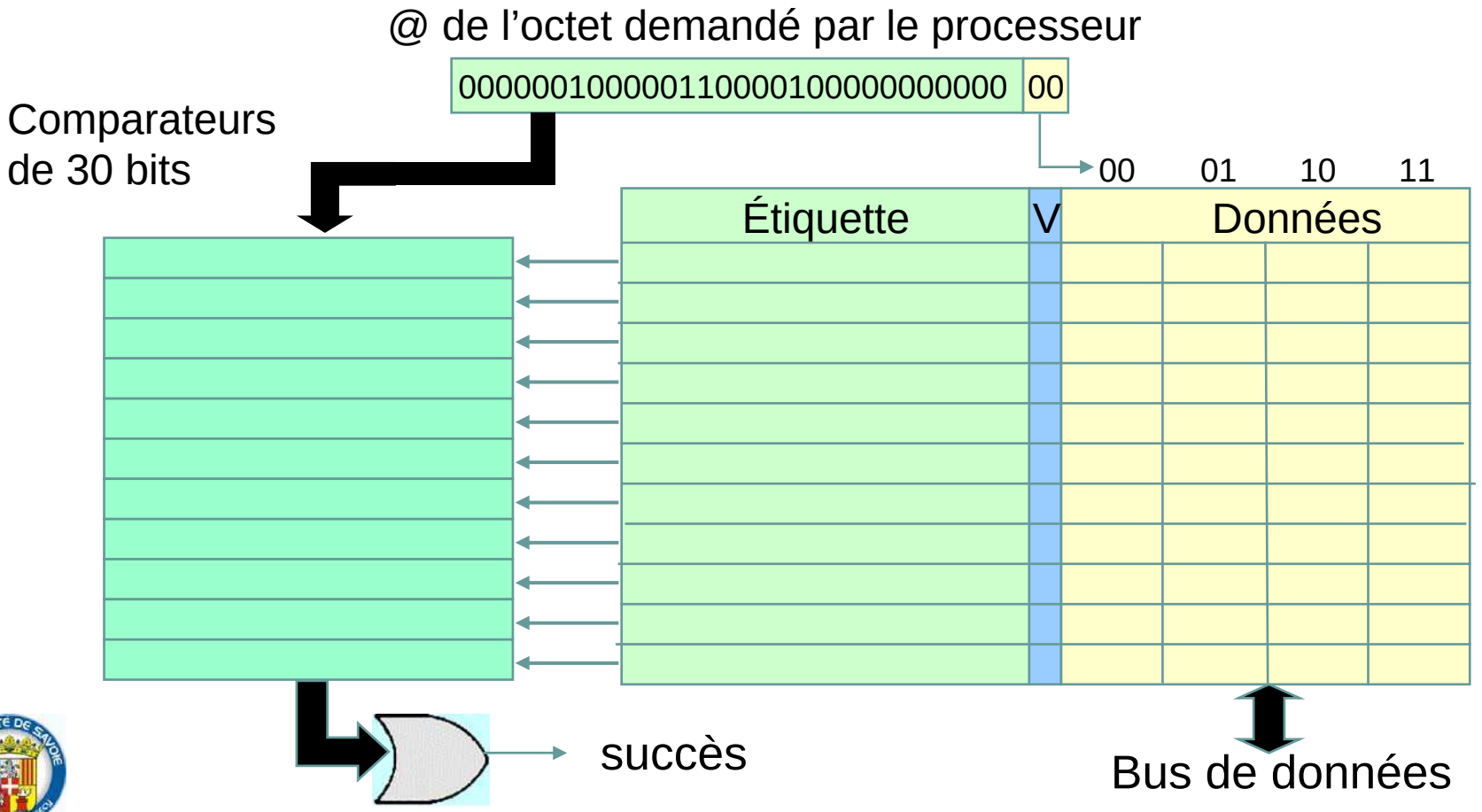
- Quand une donnée est placée dans le cache, le numéro de bloc de l'adresse fournie par le processeur est placé dans le champ étiquette de la rangée où la donnée est placée.

- À chaque accès mémoire, les comparateurs comparent simultanément le n°de bloc de l'adresse demandée avec toutes les étiquettes se trouvant dans le cache.

- Si égalité => Succès Sinon => Défaut

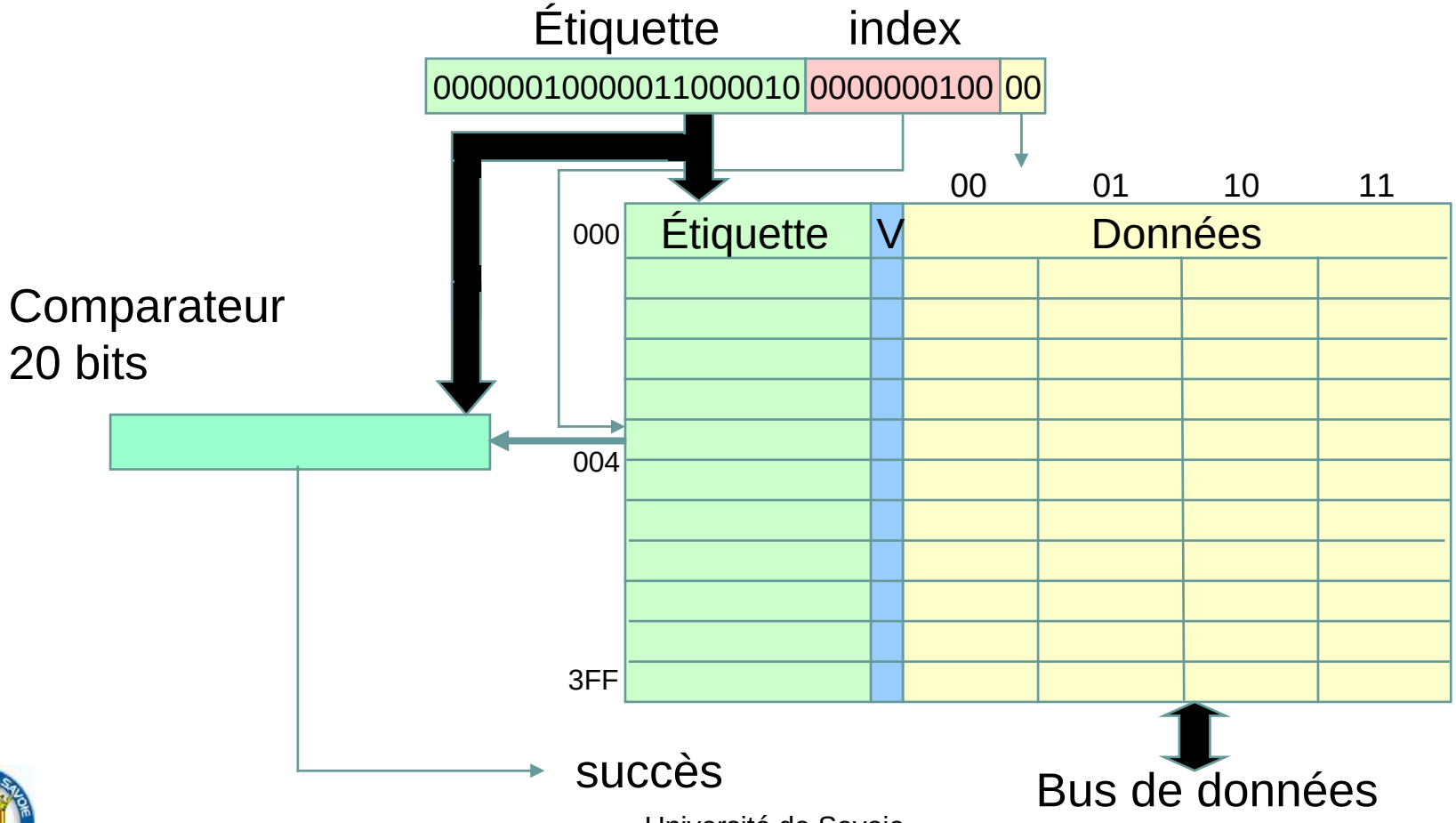
Comment un bloc est-il trouvé?
Cas du Cache associatif

- Comparaison entre l'étiquette de l'adresse et celle rangé en mémoire cache. Succès est valide que si V=1.



Comment un bloc est-il trouvé?
Cas du Cache à correspondance directe

- Comparaison entre l'étiquette de l'adresse et celle rangée en mémoire cache. Succès est valide que si V=1.



Chapitre 4 : Les mémoires caches

- 4.1 Objectif et principe d'une mémoire cache
- 4.2 Où placer un bloc?
- 4.3 Comment un bloc est-il trouvé?
- 4.4 Quel bloc remplacé lors d'un défaut?
- 4.5 Comment sont traitées les écritures?

Quel bloc remplacé lors d'un défaut ?

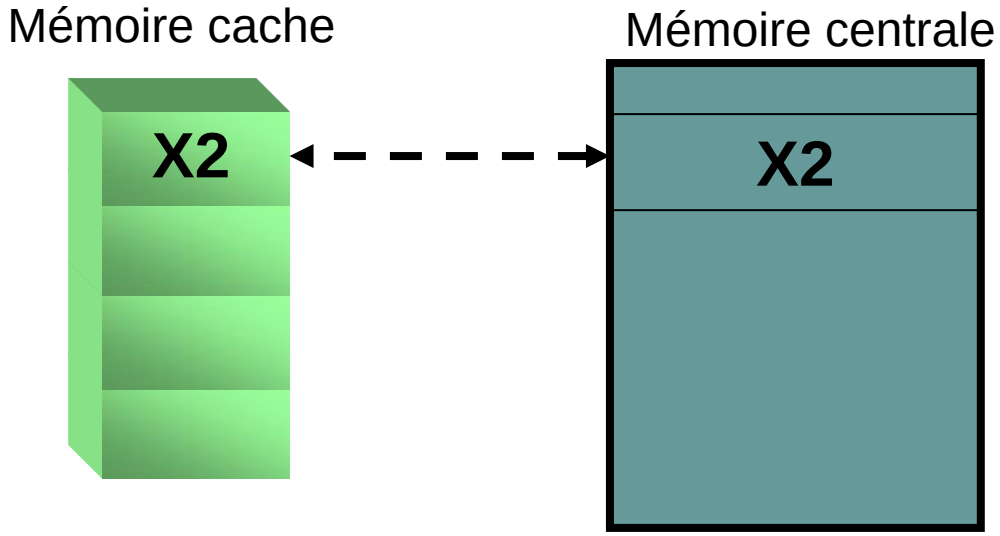
- Remplacement aléatoire :
  - Simplicité de l'algorithme
- FIFO (First In First Out)
  - Simplicité de conception
- LRU (Least Recently Used)
  - Doit mémoriser la liste des derniers éléments accédés, circuits complexes.

Chapitre 4 : Les mémoires caches

- 4.1 Objectif et principe d'une mémoire cache
- 4.2 Où placer un bloc?
- 4.3 Comment un bloc est-il trouvé?
- 4.4 Quel bloc remplacé lors d'un défaut?
- 4.5 Comment sont traitées les écritures?

Gestion des écritures (1)

- Quand une donnée se situe dans le cache, le système en possède deux copies :
  - une dans la mémoire principale
  - une dans la mémoire cache
- Comment gérer les mises à jour lorsque la donnée est modifiée localement?



Gestion des écritures (2)

- *write-through* : La donnée est écrite à la fois dans le cache et dans la mémoire principale. La mémoire principale et le cache ont à tout moment une valeur identique
- *write-back* : L'information n'est écrite dans la mémoire principale que lorsque la ligne disparaît du cache. Cette technique est la plus répandue car elle permet d'éviter de nombreuses écritures mémoires inutiles. Pour ne pas avoir à écrire des informations qui n'ont pas été modifiées (et ainsi éviter d'encombrer inutilement le bus), chaque ligne de la mémoire cache est pourvue d'un bit *dirty*. Lorsque la ligne est modifiée dans le cache, ce bit est positionné à 1, indiquant qu'il faudra réécrire la donnée dans la mémoire principale.

Chapitre 3 : Les interruptions

- 3.1 Problématique & définition
- 3.2 Rôle de la pile
- 3.3 Organisation logicielle

Les interruptions
Problématique & définition

- Un système informatique n'est utile que s'il communique avec l'extérieur. L'objectif est de pouvoir prendre connaissance que le périphérique sollicite le processeur. Cette sollicitation arrive de façon totalement asynchrone.

Deux modes sont possibles :

- Une méthode par scrutation (polling) permet d'interroger régulièrement les périphériques afin de savoir si une nouvelle donnée est présente.
- Une méthode par interruption permet au périphérique lui-même de faire signe au processeur de sa présence.

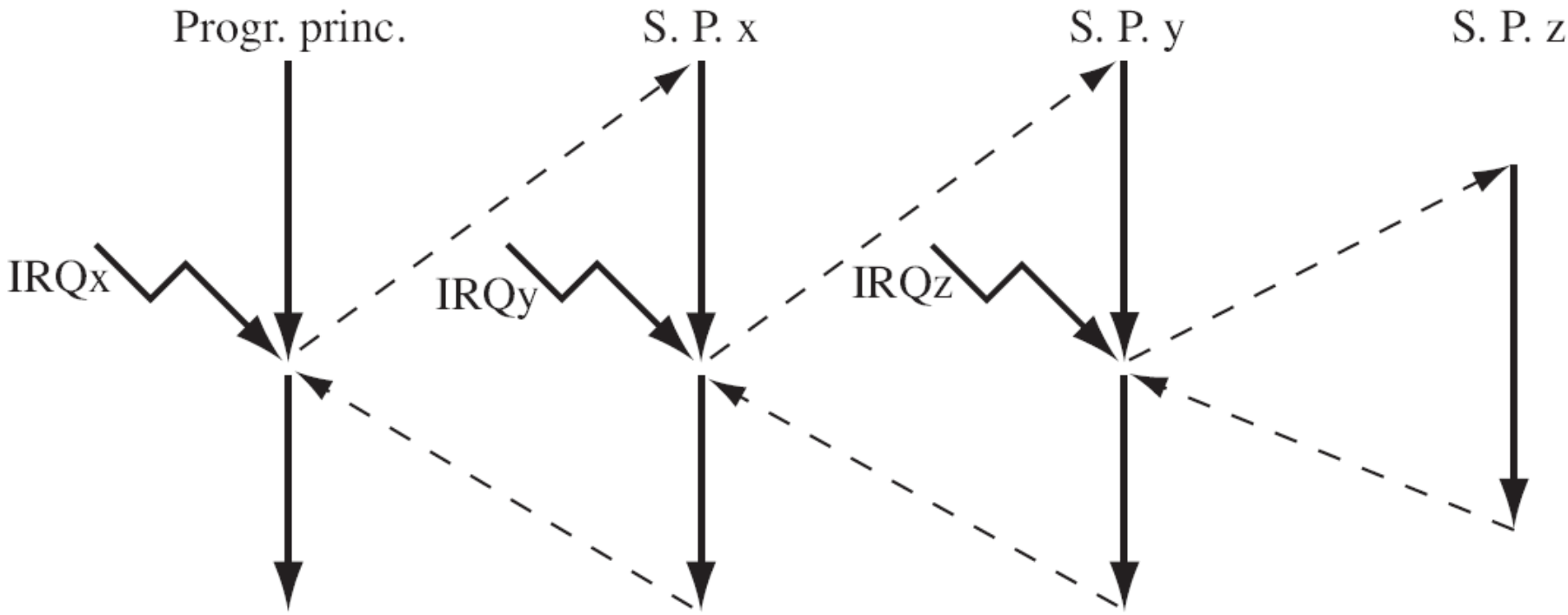
Les interruptions
Scrutation Vs interruption

- Scrutation (polling)
  - Coûteux en temps (multiplier par le nombre de périphérique à interroger)
  - Implémentation : Appel classique à une fonction dans le programme

- Interruption
  - Demande à l'initiative du périphérique
  - Prise en compte rapide de l'évènement
  - Implémentation : Interruption asynchrone d'un programme puis retour au même endroit à la fin du traitement

Les interruptions
Schéma

- Une **interruption** est un arrêt temporaire de l'exécution normale d'un programme informatique par le microprocesseur afin d'exécuter un autre programme (appelé routine d'interruption).



Les interruptions
Types d'interruption

- Interruption masquable
  - Un masque d'interruption est un mot binaire de configuration du microprocesseur qui permet de choisir **(démasquer)** quels modules pourront interrompre le processeur parmi les interruptions disponibles.

- Interruption non masquable
  - Elles s 'exécutent quoi qu'il arrive, souvent avec une priorité élevé (ex : Reset)

Les interruptions
Configuration

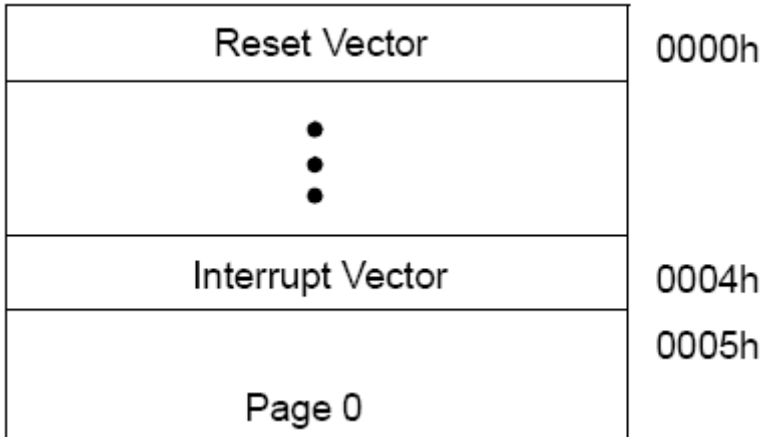
- Un système peut accepter plusieurs sources d'interruption. Chacune est configurable par registre (registre d'interruption).

- Méthode de configuration des interruptions
  - Sélectionner les interruptions qui nous intéressent
  - Valider les interruptions de façon globale
  - Ecrire le/les sous programme d'interruption
  - Définir les priorités entres interruptions

Les interruptions
Configuration

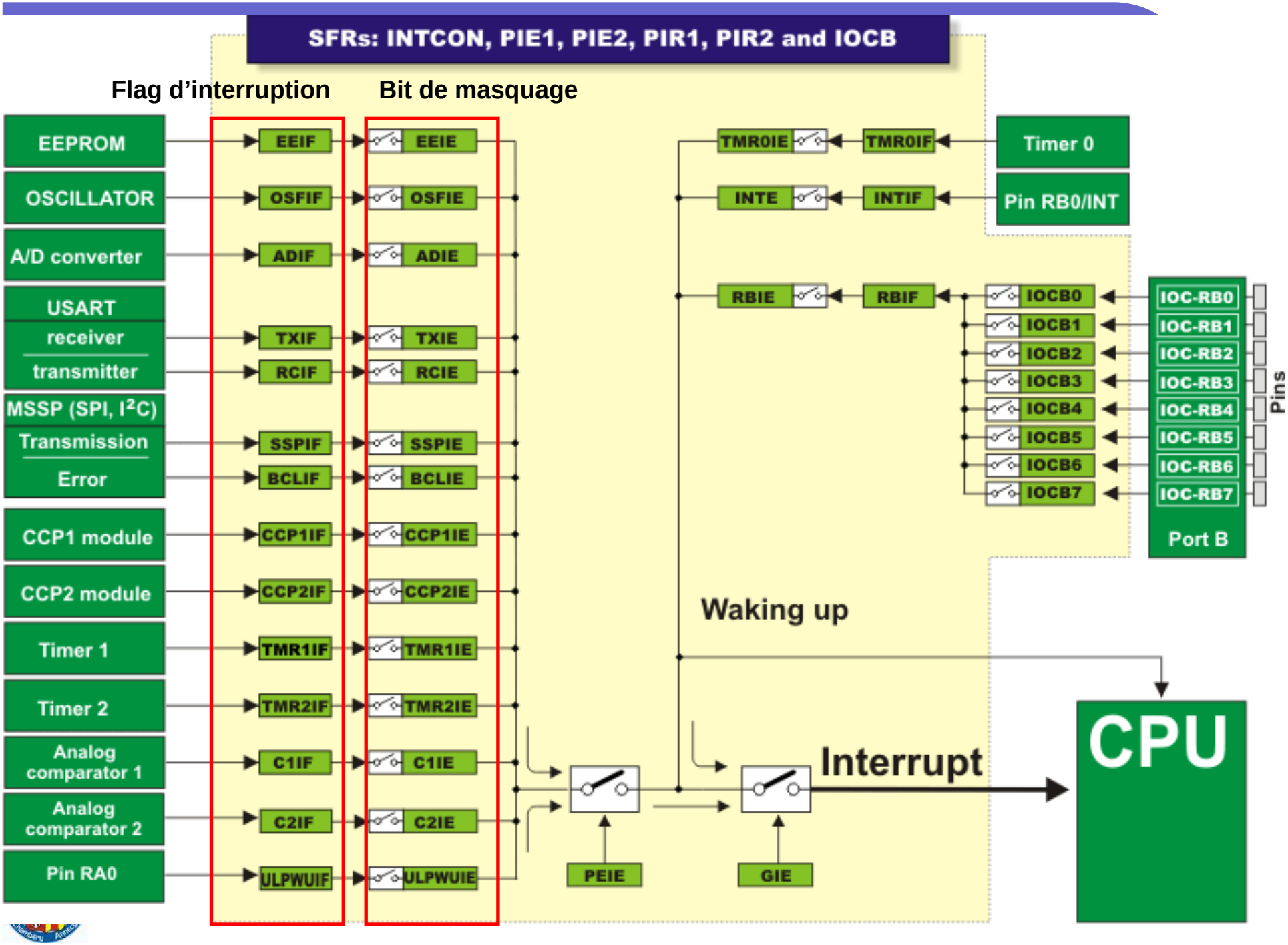
- Dans le sous programme d'interruption
  - Sauvegarder le contexte (fait automatique en langage C)
  - Définir la source d'interruption (si le sous programme est commun entres plusieurs sources d'interruption)
  - Réinitialiser les flags d'interruption
  - Ecrire le code relatif à l'application
  - Restituer le contexte (fait automatique en langage C)

Cas du PIC 16F877 (microchip)



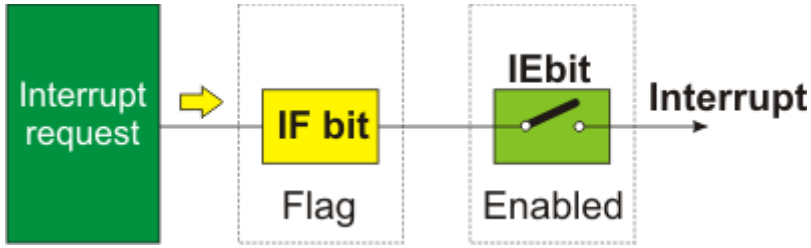
Cas du 80C51 (intel)

Interrupt Source	Interrupt Request Bits	Cleared by Hardware	Vector Address
$\overline{\text{INT0}}$	IE0	No (level) Yes (trans.)	0003H
TIMER 0	TF0	Yes	000BH
$\overline{\text{INT1}}$	IE1	No (level) Yes (trans.)	0013H
TIMER 1	TF1	Yes	001BH
SERIAL PORT	RI, TI	No	0023H
TIMER 2	TF2, EXF2	No	002BH


SFRs: INTCON, PIE1, PIE2, PIR1, PIR2 and IOCB
Flag d'interruption
Bit de masquage
EEPROM
OSCILLATOR
A/D converter
USART
receiver
transmitter
MSSP (SPI, I²C)
Transmission
Error
CCP1 module
CCP2 module
Timer 1
Timer 2
Analog comparator 1
Analog comparator 2
Pin RA0
EEIF
OSFIF
ADIF
TXIF
RCIF
SSPIF
BCLIF
CCP1IF
CCP2IF
TMR1IF
TMR2IF
C1IF
C2IF
ULPWUIF
EEIE
OSFIE
ADIE
TXIE
RCIE
SSPIE
BCLIE
CCP1IE
CCP2IE
TMR1IE
TMR2IE
C1IE
C2IE
ULPWUIE
TMR0IE
TMR0IF
Timer 0
INTE
INTIF
Pin RB0/INT
RBIE
RBIF
IOCB0
IOCB1
IOCB2
IOCB3
IOCB4
IOCB5
IOCB6
IOCB7
IOC-RB0
IOC-RB1
IOC-RB2
IOC-RB3
IOC-RB4
IOC-RB5
IOC-RB6
IOC-RB7
Pins
Port B
Waking up
PEIE
GIE
Interrupt
CPU

Les interruptions
Démasquage des interruptions

- Autorisation des interruptions
  - L'autorisation globale des interruptions



  - Démasquage des interruptions

INTCON	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)				
	GIE	PEIE	T0IE	INTE	RBIE				
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3				
PIE1		R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Features
	-	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
PIE2	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)		R/W (0)	Features
	OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	-	CCP2IE	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Les interruptions
Les flags d'interruption

- Visualisation des flags d'interruption

INTCON								
						R/W (0)	R/W (0)	R/W (x)
						T0IF	INTF	RBIF
						Bit 2	Bit 1	Bit 0

Features: Bit name

PIR1								
	R/W (0)	R (0)	R (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	
	-	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

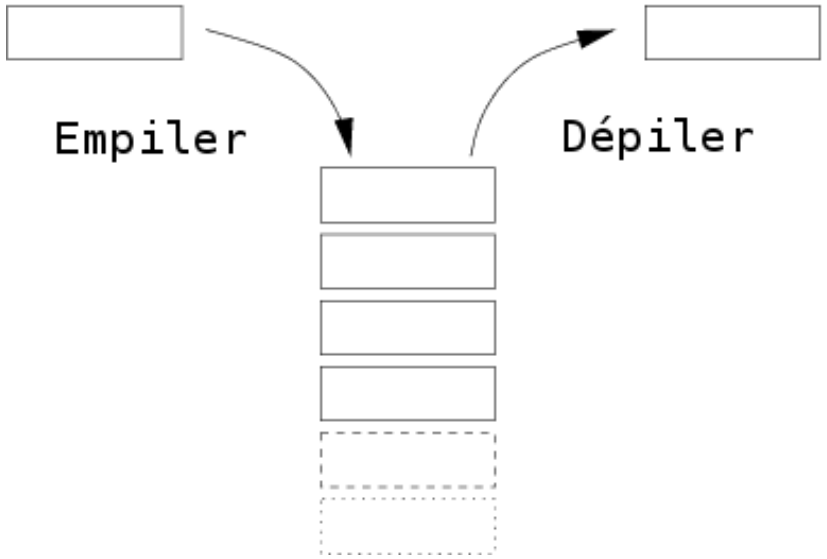
Features: Bit name

PIR2								
	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)		R/W (0)
	OSFIF	C2IF	C1IF	EEIF	BCLIF	ULPWUIF	-	CCP2IF
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Features: Bit name

Les interruptions
Le rôle de la pile

- La pile est une mémoire LIFO (Last In First Out) dans laquelle on stoke des variable temporaire (donnée ou adresse). Le haut de la pile est pointé par le registre SP (Stack Pointer).



Les interruptions
Rôle de la pile

- Elle va servir à :
  - **sauvegarder le contexte** l'environnement (adresse du programme et valeur des registres au moment de l'interruption).
  - **restituer le contexte** à la fin de l'interruption

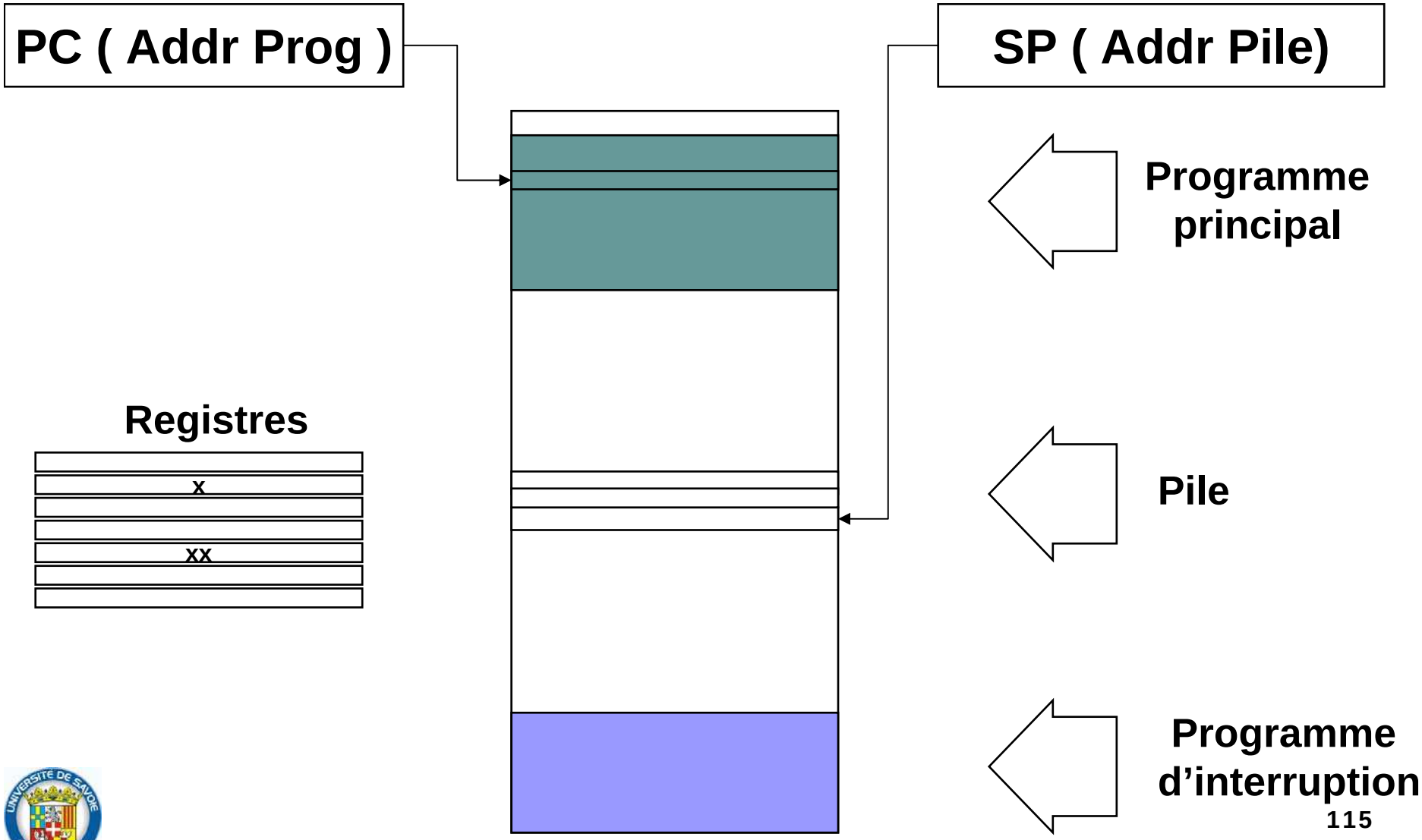
Note 1 : La sauvegarde et la restitution est faite implicitement en langage C.

Note 2 : Une fonction d'interruption est noté spécifiquement. Exemple du PIC qui ne possède qu'un seul vecteur d'interruption :

```
void interrupt() {
}
```

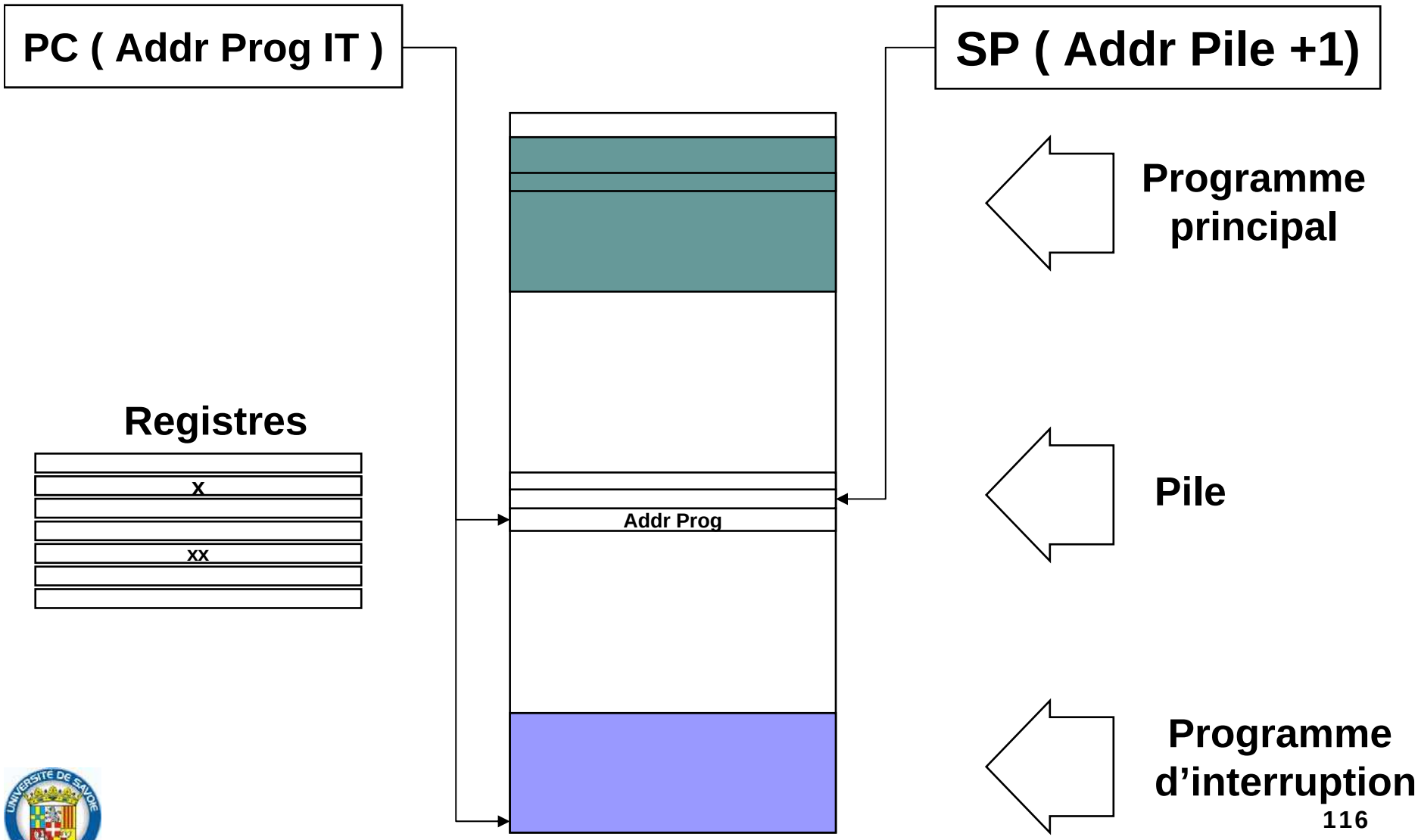
Les interruptions

Avant l'interruption



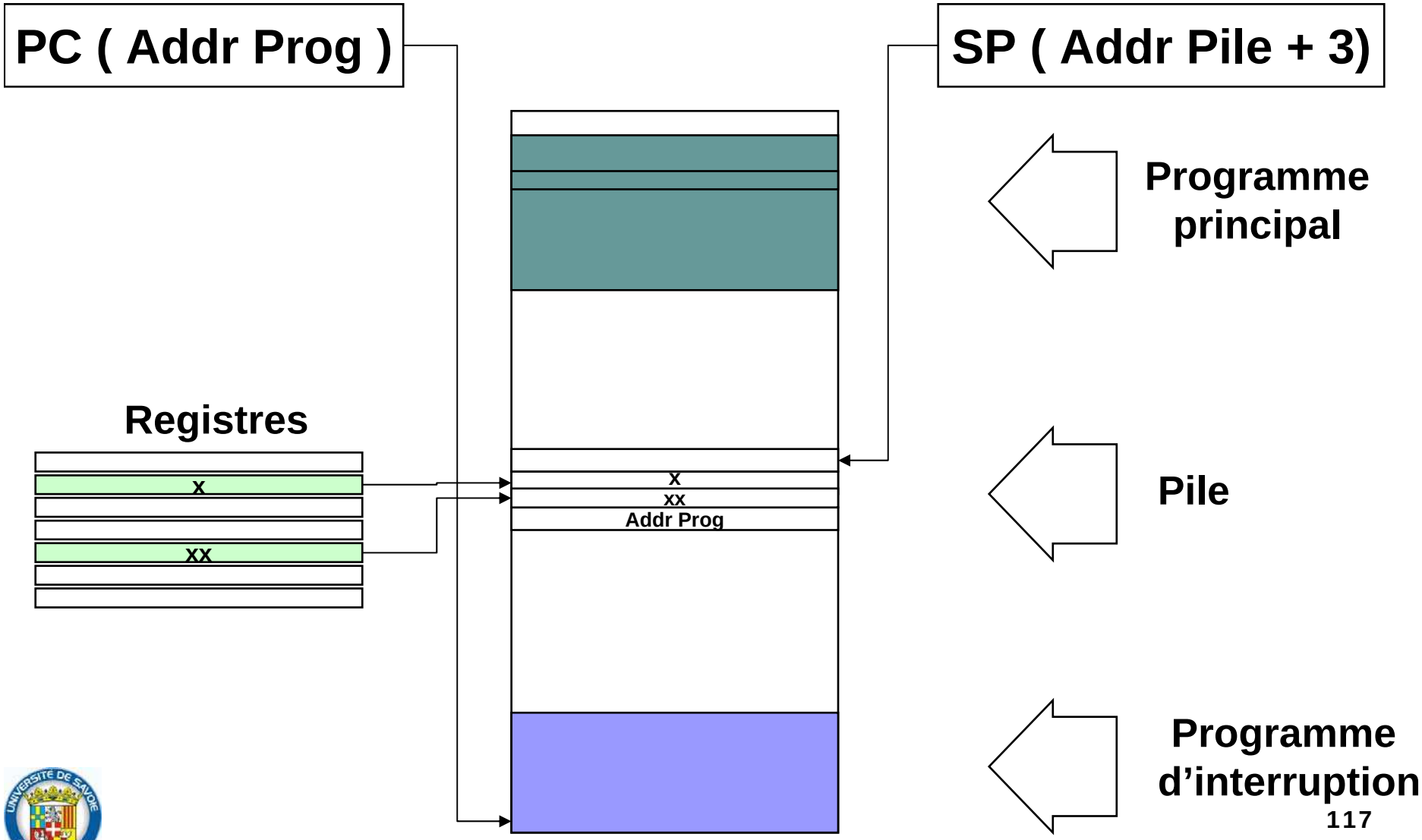
Les interruptions

Arrivée d'une interruption



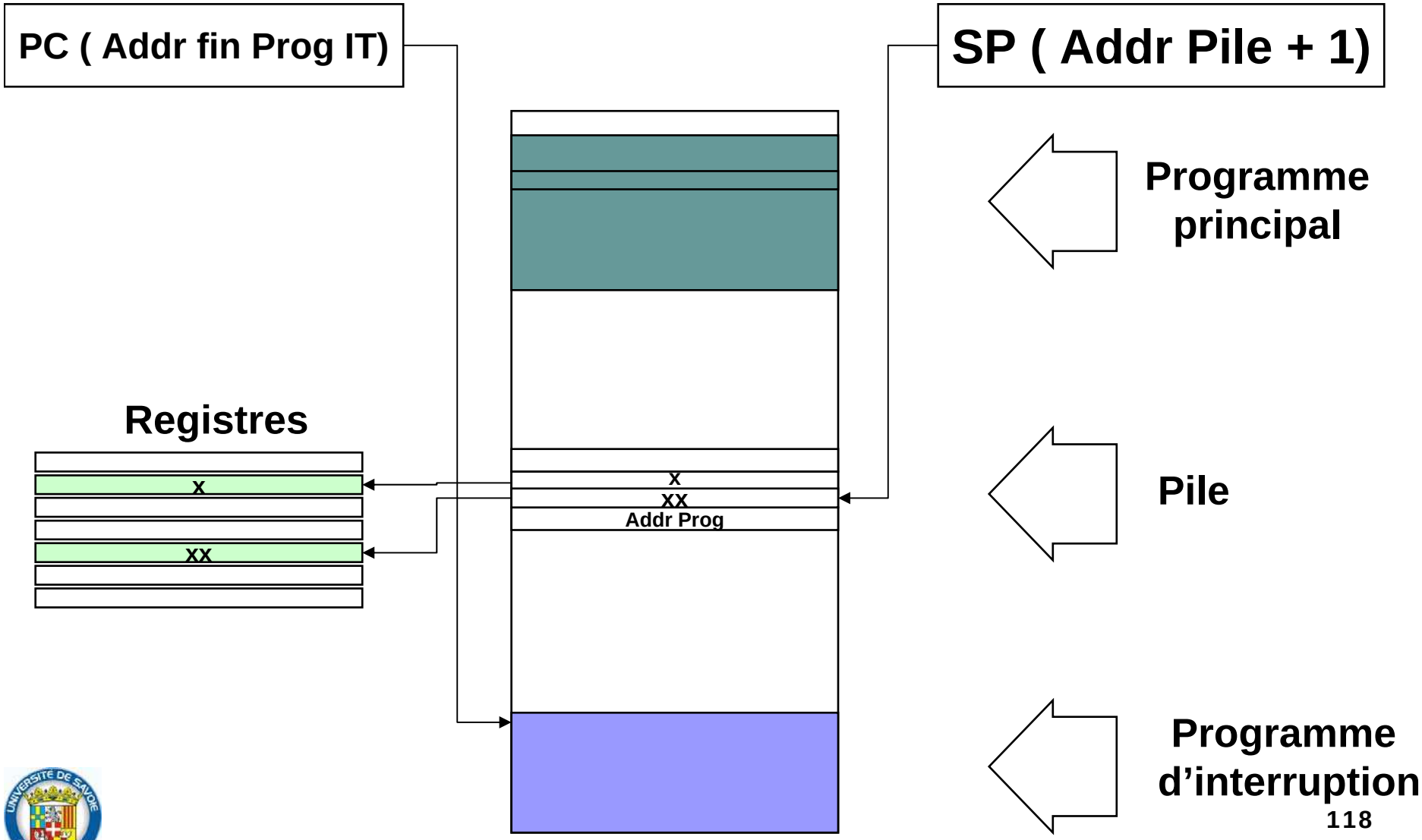
Les interruptions

Arrivée d'une interruption : Sauvegarde contexte



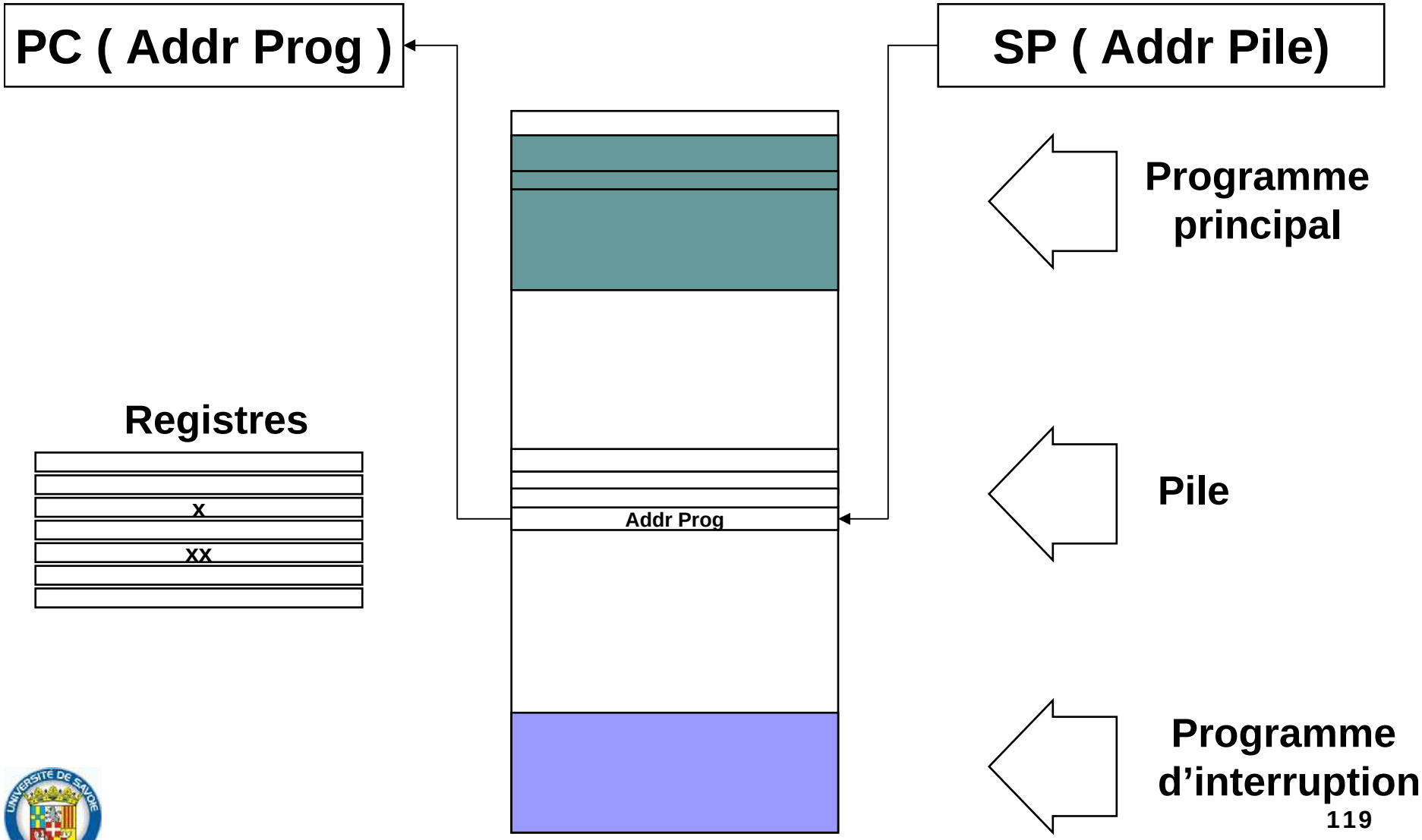
Les interruptions

Fin d'une interruption : Restitution contexte



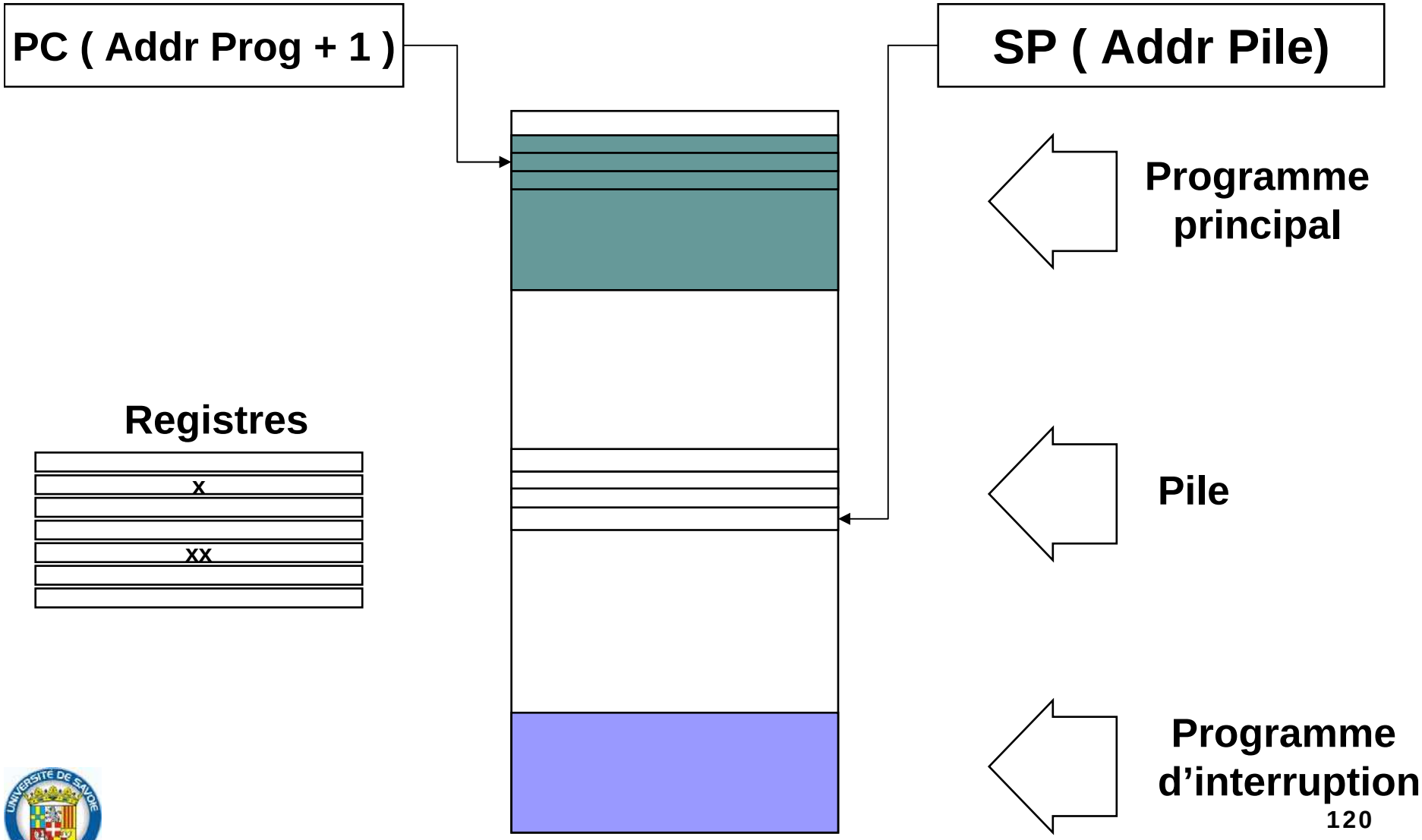
Les interruptions

Fin d'une interruption



Les interruptions

Retour au programme principal



Chapitre 4 : Les accès DMA

- 4.1 Définitions et problématiques
- 4.2 Les interfaces des disques de stockages
- 4.3 Les méthodes d'accès directs
- 4.4 Etude d'un système

Définitions et problématiques
Pourquoi les accès DMA?

Le système doit récupérer des données en provenance de ces périphériques externes. Plusieurs méthodes sont possibles :

- Une méthode par scrutation (polling) permet d'interroger régulièrement les périphériques afin de savoir si une nouvelle donnée est présente.
- Une méthode par interruption permet au périphérique lui-même de faire signe au processeur de sa présence.
- Une méthode par Accès Direct à la Mémoire (DMA) permet de gérer le transfert de façon autonome.

Définitions et problématiques

L'accès direct à la mémoire ou DMA est un procédé où des données circulant de ou vers un périphérique (port de communication, disque dur) sont transférées directement par un contrôleur adapté vers la mémoire centrale de la machine, sans intervention du microprocesseur. Le micro interviendra seulement pour initier et conclure le transfert. La conclusion du transfert ou la disponibilité du périphérique peuvent être signalés par interruption.

(Source : Wikipédia)

Définitions et problématiques
Utilisations

- Carte graphique
- Carte son
- Disque dur
- Lecteur CD…
- Et beaucoup d'autres périphériques internes…

Chapitre 4 : Les accès DMA

- 4.1 Définitions et problématiques
- 4.2 Les interfaces des disques de stockages
- 4.3 Les méthodes d'accès directs
- 4.4 Etude d'un système

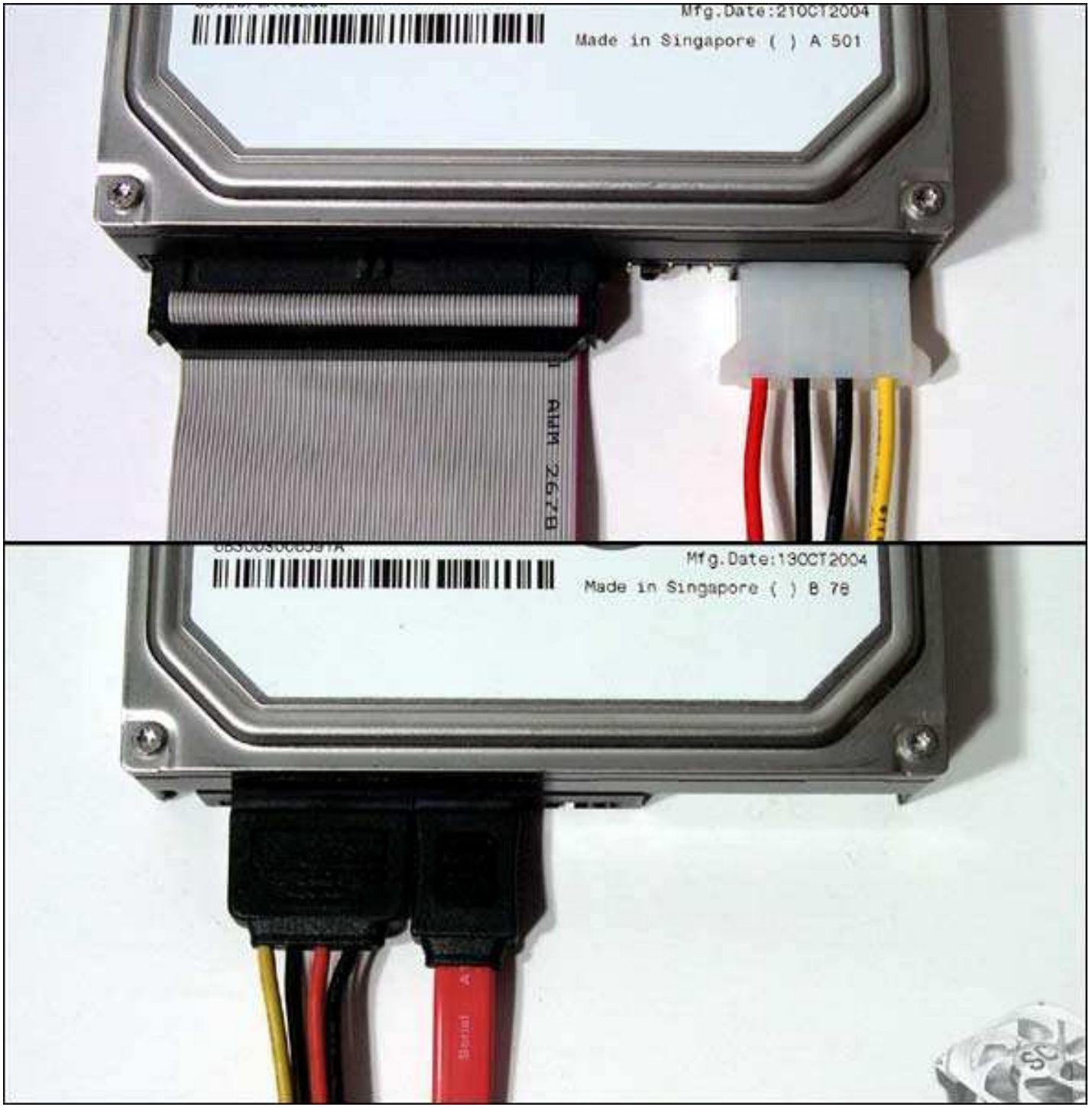
Les interfaces des disques

Les interfaces ATA

- Le standart ATA (Advanced Tecnology Attachment) est une interface permettant la connexion de périphérique de stockage sur les ordinateurs de type PC. Ce standard appararu en 1994 tend à disparaître au profit du SATA. Il est aussi connu sous le nom IDE (Integrated Drive Elelectronics) ou E-IDE (Enhanced IDE)
- Initialement pour connecter les disques dur, il a été étendu pour pouvoir interfacer d'autre périphériques de stockage (Interface ATAPI=ATA-Packet Interface)

Les interfaces des disques
Les interfaces SATA

- Les interfaces SATA (Serial ATA), permettent de transférer les données en série.
  - Gain de place
  - Branchement à chaud
  - Résolution de problème de CEM (compatibilité Electromagnétique)


Mfg.Date:21OCT2004
Made in Singapore ( ) A 501
AWM 2678
Mfg.Date:13OCT2004
Made in Singapore ( ) B 78

Chapitre 4 : Les accès DMA

- 4.1 Définitions et problématiques
- 4.2 Les interfaces des disques de stockages
- 4.3 Les méthodes d'accès directs
- 4.4 Etude d'un système

Les méthodes d'accès directs

Les modes de transfert

- Mode PIO

  PIO : Programmed Input Output.

  Permet d'échanger des données avec la mémoire vive. Ces transferts sont gérés entièrement par le processeur.

- Mode DMA

  La technique du DMA (Direct Memory Access) permet de désengorger le processeur en permettant à chacun des périphériques d'accéder directement à la mémoire.

Les méthodes d'accès directs

Mode PIO

Des commandes gérées directement par le processeur permette la gestion du transfert. Toutefois, de gros transferts de données peuvent rapidement imposer une grosse charge de travail au processeur et ralentir l'ensemble du système. Il existe 5 modes PIO définissant le taux de transfert maximal.

Mode PIO	Débit (Mo/s)
Mode 0	3.3
Mode 1	5.2
Mode 2	8.3
Mode 3	11.1
Mode 4	16.7

Les méthodes d'accès directs
Mode DMA

- La technique du DMA (Direct Memory Access) permet de désengorger le processeur en permettant à chacun des périphériques d'accéder directement à la mémoire. Deux types de DMA existent:
  - Le DMA dit "single word" permet de transmettre un mot simple à chaque session de transfert,
  - Le DMA dit "multi-word" permet de transmettre successivement plusieurs mots à chaque session de transfert.
- Le tableau suivant liste les différents modes DMA et les taux de transfert associés :

Mode DMA	Débit (Mo/s)
0 (Single word)	2.1
1 (Single word)	4.2
2 (Single word)	8.3
0 (Multiword)	4.2
1 (Multiword)	13.3
2 (Multiword)	16.7

Les méthodes d'accès directs

Mode Ultra DMA (1)

L'idée est d'augmenter la fréquence du signal d'horloge pour augmenter la rapidité. Toutefois sur une interface où les données sont envoyées en parallèle l'augmentation de la fréquence pose des problèmes d'interférence électromagnétiques. Deux solutions ont été apporté qui vont être en étroite relation :

- Augmentation de la fréquence : Utilisation des front montants et descendant.
- Amélioration du connecteur ATA (a partir de l'Ultra DMA mode 4 un nouveau type de nappe a été introduit afin de limiter les interférences ; il s'agit d'une nappe ajoutant 40 fils de masse entrelacés avec les fils de données.
- Apparition du CRC

Les méthodes d'accès directs
Mode Ultra DMA (2)

- Fonctionnement :
  - La fréquence de transfert augmente tant que les données transmises se font sans erreur.
  - Lorsque qu'une erreur est rencontrée, le transfert passe dans un mode Ultra DMA inférieur (voire sans Ultra DMA).

Mode Ultra DMA	Débit (Mo/s)
UDMA 0	16.7
UDMA 1	25.0
UDMA 2 (Ultra-ATA/33)	33.3
UDMA 3	44.4
UDMA 4 (Ultra-ATA/66)	66.7
UDMA 5 (Ultra-ATA/100)	100
UDMA 6 (Ultra-ATA/133)	133

Les méthodes d'accès directs

Les vitesses de transfert

Architecture	Name	Time	Transfer Speed	Note
Serial	Serial ATA[2]	Mid-2007	600 MB/S (generation 3)	CRC / Package transfer (bits of data together)
		Mid-2004	300 MB/S (generation 2)	
		Fall-2002	150 MB/S (generation 1)	
Parallel	Ultra DMA	Current mainstream	16.7/25.0/33.3/44.4/66.7 100.0/133.0 MB/S	CRC / Multi-word
	DMA	1990	4.2/13.3/16.7MB/S	Multi-word
		1980	2.1/4.2/8.3 MB/S	Single word

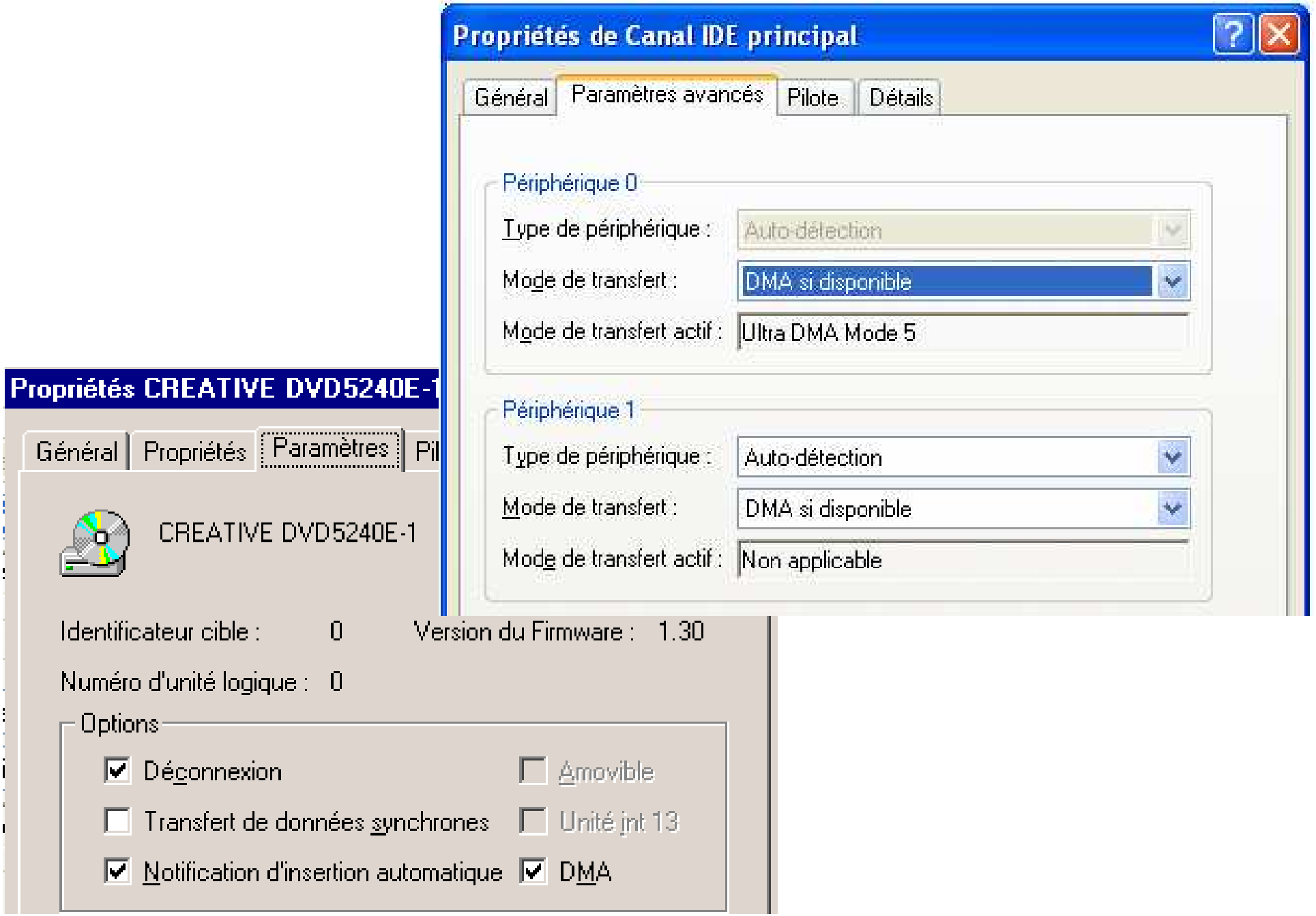
Les vitesses de transfert (mode DMA ou Ultra DMA restent donc toujours en étroite relation avec l'architecture utilisée (ATA, Serial ATA, …)

Les méthodes d'accès directs
Les canaux DMA

- Un ordinateur de type PC possède 8 canaux DMA.
  Les canaux DMA sont généralement assignés comme suit :

DMA0 -	System Use : Memory (DRAM) Refresh
DMA1 -	Libre
DMA2 -	contrôleur de disquettes
DMA3 -	port parallèle
DMA4 -	contrôleur d'accès direct à la mémoire (renvoi vers DMA0)
DMA5 -	(carte son)/ libre
DMA6 -	(SCSI)/ libre
DMA7 -	disponible

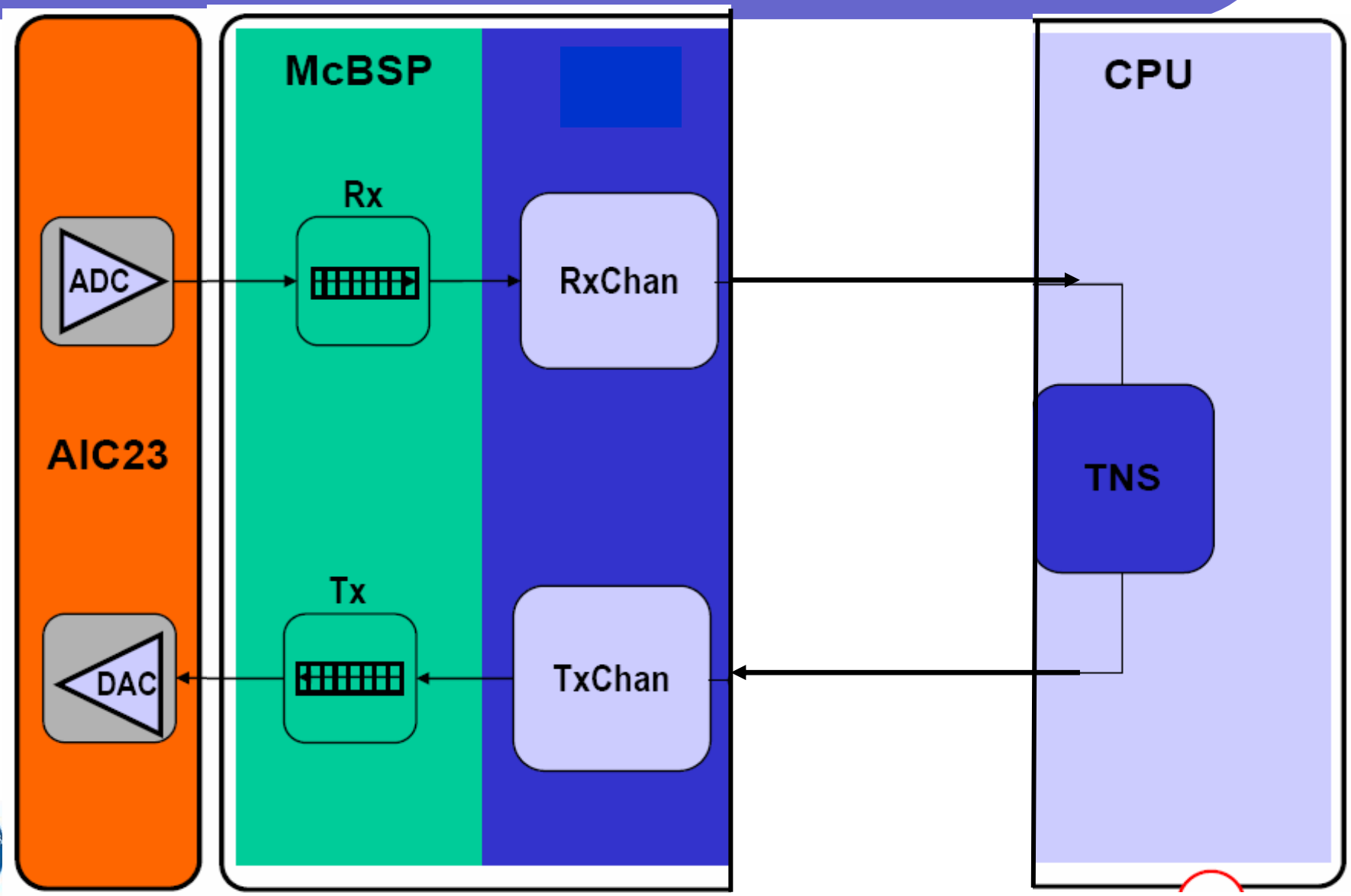
Démarrer>Programmes>Accessoires>Outils Système>informations Système



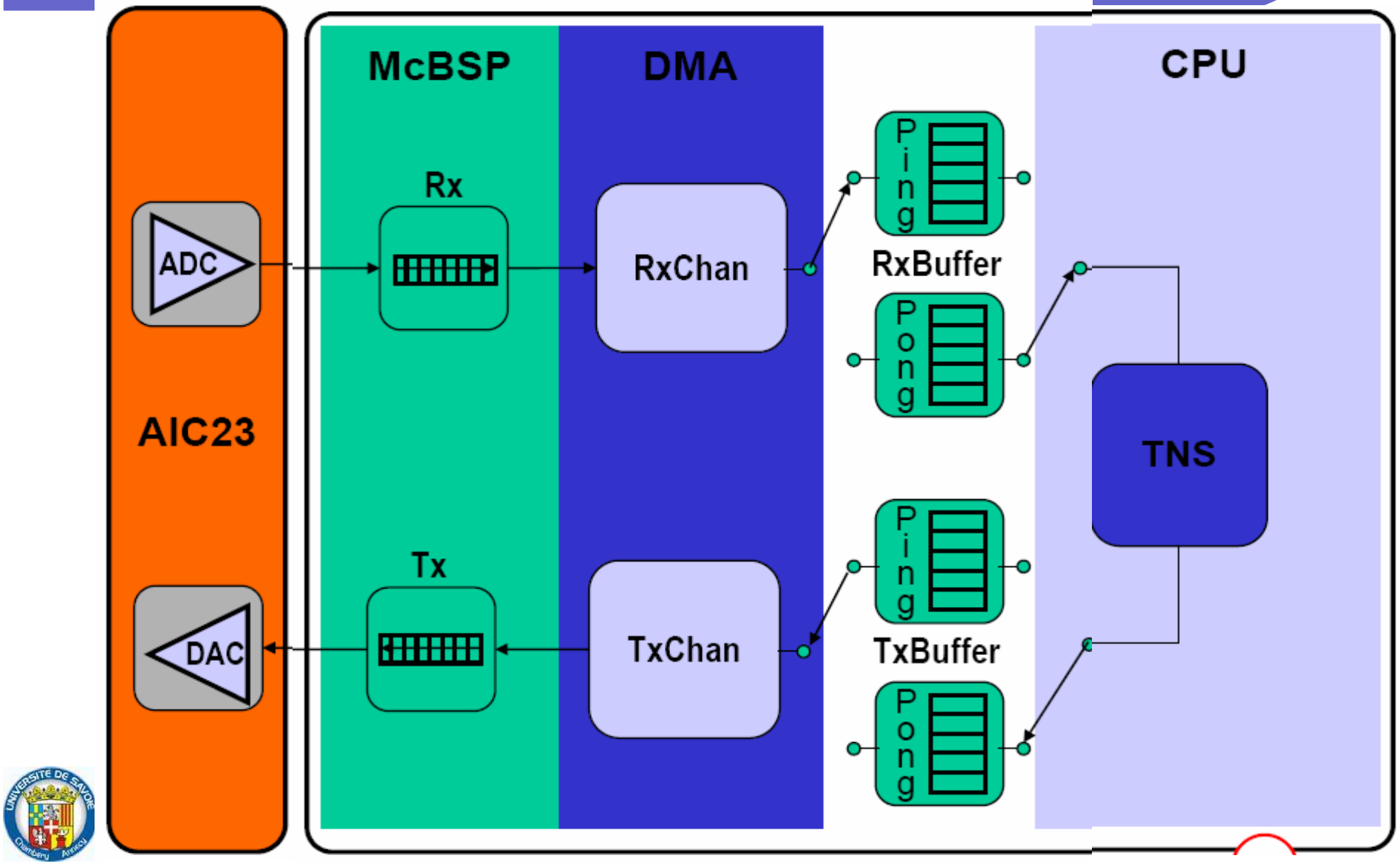
Chapitre 4 : Les accès DMA

- 4.1 Définitions et problématiques
- 4.2 Les interfaces des disques de stockages
- 4.3 Les méthodes d'accès directs
- 4.4 Etude d'un système

Etude d’un système (1)



Etude d’un système (2)



Etude d'un système
Programmation logicielle (1)

- Nous avons toujours les trois possibilités pour le transfert d'information de l'extérieur vers l'intérieur du système.
  - Polling (scrutation)
  - Interruption
  - DMA

- Pour la programmation des systèmes embarqués, on utilise des librairies :
  - BSL : Board Support Library
  - CSL : Chip Support Library

Polling (scrutation)

Cas du TP 2 sur DSP TMS320

```
while (!DSK5416_PCM3002_read16(hCodec, &left_input));
while (!DSK5416_PCM3002_write16(hCodec, left_output));
while (!DSK5416_PCM3002_read16(hCodec, &right_input));
while (!DSK5416_PCM3002_write16(hCodec, right_output));
```

Ces fonctions font parties des librairies de la carte (Board Support Library)

>> Voir C:\CCStudio_v3.1\docs\hlp\C5416DSK.HLP

Etude d'un système
Programmation logicielle (3)

Interruptions

- On active les interruptions sur la réception et l'émission d'une donnée.

```
IRQ_enable(IRQ_EVT_RINT0); //Enables Reception event (IMR register flag)
IRQ_enable(IRQ_EVT_XINT0); //Enables Transmission event (IMR register flag)
IRQ_globalEnable();              //Enables all Unmask Events
IRQ_clear(IRQ_EVT_RINT0) // Clear the specified Interrupt Flag (IFR Register)
```

Table 6–19. TMS320C541 Interrupt Locations and Priorities

TRAP/INTR Number (K)	Priority	Name	Location (Hex)	Function
20	7	RINT0/SINT4	50	Serial port 0 receive interrupt
21	8	XINT0/SINT5	54	Serial port 0 transmit interrupt

- Ces fonctions font parties des librairies de la carte (Chip Support Library)
- >> Voir aide > IRQ functions

Etude d'un système
Programmation logicielle (4)

DMA

1. Déclaration et réservation buffers ping pong

2. Configuration des canaux DMA 0 et 1et
   Configuration interruption (DMA0)

3. Création des fonctions d'interruption pour traitement

Etude d'un système
Programmation logicielle (5)

DMA

1

```
#define BUFFSIZE 20
...
Int16 RxBufferPing[BUFFSIZE];
...
Int16 RxBufferPong[BUFFSIZE];
```

Primary Functions

2

Function	Purpose
DMA_close()	Closes a DMA channel
DMA_config()	Sets up the DMA channel using the configuration structure
DMA_configArgs()	Sets up the DMA channel using the register values passed in
DMA_open()	Opens a DMA channel
DMA_pause()	Pauses a DMA channel. Identical to DMA_stop().
DMA_reset()	Resets DMA channel register to their power-on reset value
DMA_start()	Starts a DMA channel
DMA_stop()	Disables a DMA channel

3

```
void dmaRxHwi(void)
{
```