Formation en Calcul Scientifique - LEM2I

Architecture et programmation

Violaine Louvet [1]

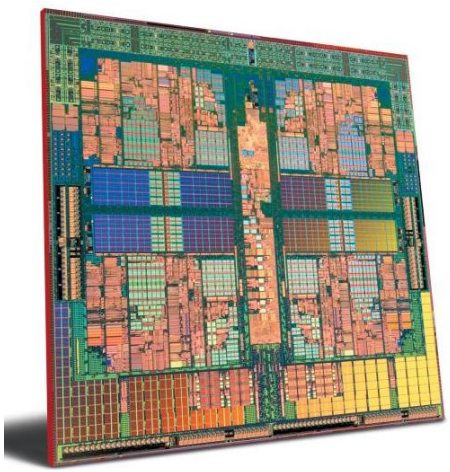
[1] Institut Camille jordan - CNRS

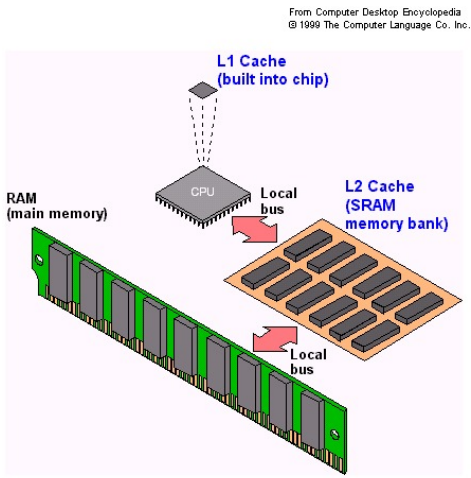
12-14/12/2011

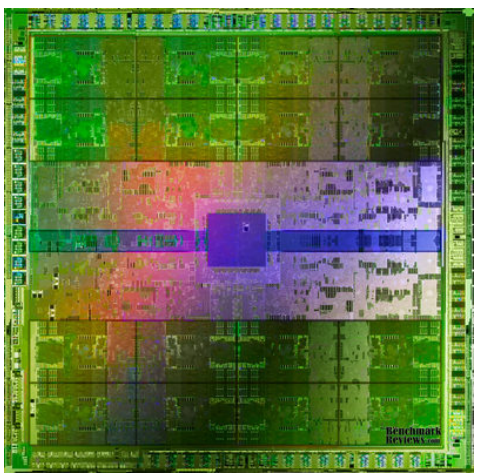
Introduction

Décoder la relation entre l'architecture et les applications :

- Adapter les méthodes numériques, les algorithmes et la programmation
- Comprendre le comportement d'un programme
- Optimiser les codes de calcul en fonction de l'architecture

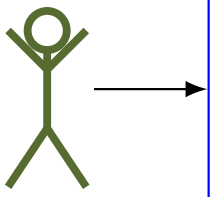






Du programme au hardware

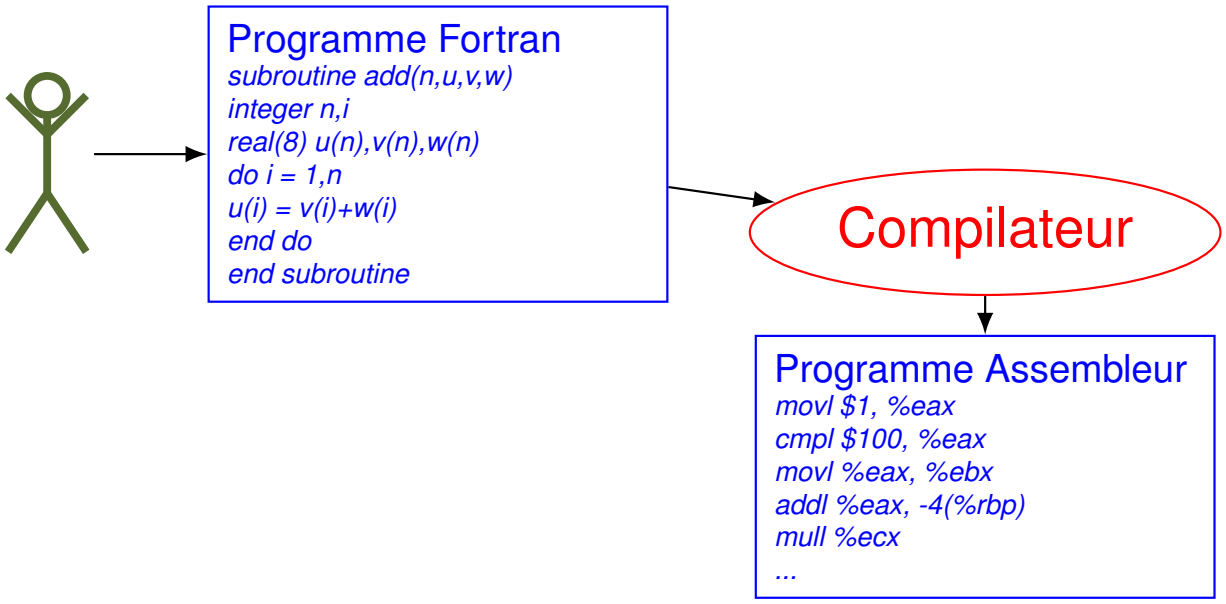
Du programme au hardware



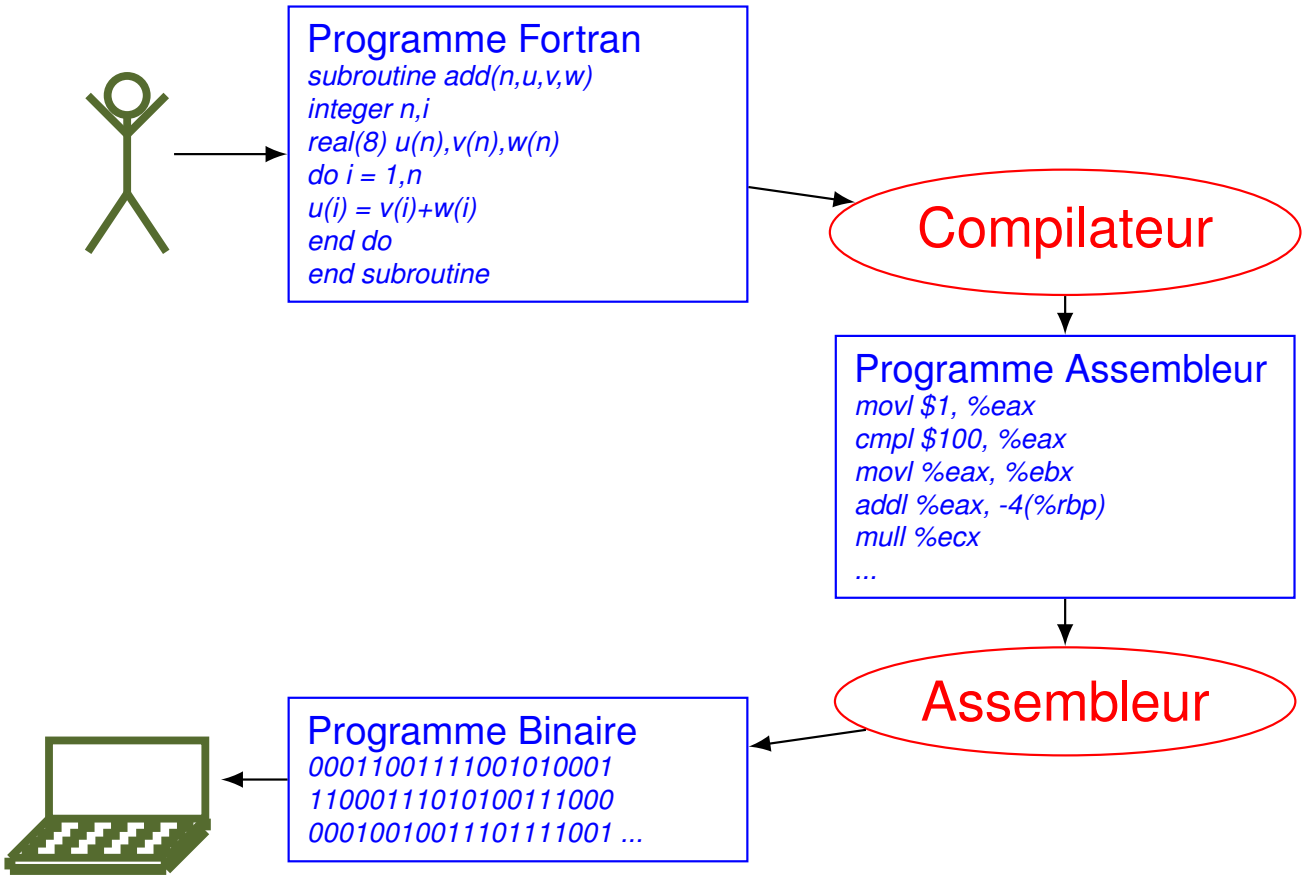
Programme Fortran

```
subroutine add(n,u,v,w)
integer n,i
real(8) u(n),v(n),w(n)
do i = 1,n
u(i) = v(i)+w(i)
end do
end subroutine
```

Du programme au hardware



Du programme au hardware



Binaire

- Données représentées physiquement par 2 niveaux de tensions différents.
- L'information élémentaire est appelée le bit et ne peut prendre que 2 valeurs : 0 ou 1
  - 1 bit permet donc d'avoir 2 états (0 ou 1), 2 bits permettent d'avoir 4 (2^2) états (00,01,10,11),... , n bits permettent ainsi d'avoir 2^n états
- Une information plus complexe sera codée sur plusieurs bit : cet ensemble est appelé mot
- Un mot de 8 bits est appelé un octet (ou byte)

Exemple de représentation d'un nombre entier en binaire

$(101)_2 <> 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (5)_{10}$

Exemple de représentation d'un nombre entier en hexdécimal

Lorsqu'une donnée est représentée sur plus de 4 bits, on préfère souvent l'exprimer en hexadécimal, à l'aide des 16 caratères :

0 1 2 3 4 5 6 7 8 9 A B C D E F

$(9A)_{16} <> 9 \times 16^1 + A \times 16^0 = 9 \times 16^1 + 10 \times 16^0 = (154)_{10}$

Retour à l'assembleur

- Langage symbolique très proche du CPU, plus lisible que des suites de bits

Exemple : additionner 2 et 3

- En assembleur :

```
mov AX,2
add AX,3
```

- En langage machine :

```
10111000000000100000000000000101000000110000000
```

Liens entre code assembleur et architecture

Que manipule-t-on en assembleur ?

- des séquences d'instructions $\longrightarrow$ traitement d'informations
  - $\rightarrow$ Branchements : *CMP* compare deux registres ...
  - $\rightarrow$ Opérations arithmétiques et logiques : *ADD*, *MUL*, *AND*, *XOR* ...
  - $\rightarrow$ Accès à la mémoire : *LOAD*, *STORE*
- des registres $\longrightarrow$ mémoire
- des données de différents types $\longrightarrow$ représentativité de ces données

Sommaire

Sommaire

Sommaire

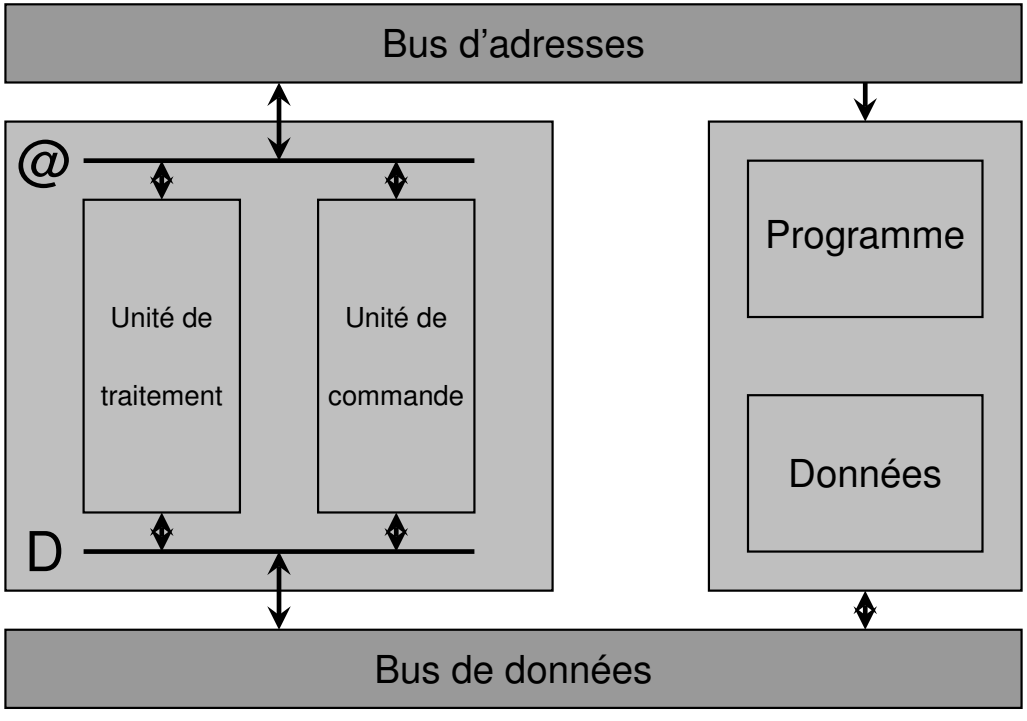
Architecture de base

Le processeur est construit autour de deux éléments principaux :

- Une unité de commande
- Une unité de traitement

associés à des registres chargés de stocker les informations à traiter :

- des registres de données d'usage général d'accès très rapide
- des registres d'adresses (pointeurs)



Unité de commande

Permet de séquencer le déroulement des instructions :

- recherche de l'instruction en mémoire
- décodage de l'instruction (stockée sous forme binaire)
- exécution de l'instruction
- préparation de l'instruction suivante

Composition

- Compteur de programme (PC) : registre dont le contenu est initialisé avec l'adresse de la 1ère instruction du programme. Contient toujours l'instruction à exécuter
- Registre d'instruction et décodeur d'instruction : l'instruction à exécuter est rangée dans le registre d'instruction et décodée par le décodeur d'instruction
- Bloc logique de commande (séquenceur) : organise les instructions au rythme d'une horloge en fonction de l'état de tous les autres composants

Unité de traitement

Le coeur du CPU : assure les traitements nécessaires à l'exécution des instructions

Composition

- Unité Arithmétique et Logique (UAL) : assure les fonctions logiques (ET, OU, ...) et arithmétiques (addition,...)
- Registre d'état : indicateur de l'état de la dernière opération. Conditionne souvent la suite du programme.
- Accumulateurs : registres de travail qui servent à stocker une opérande au début d'une opération arithmétique et le résultat à la fin de l'opération

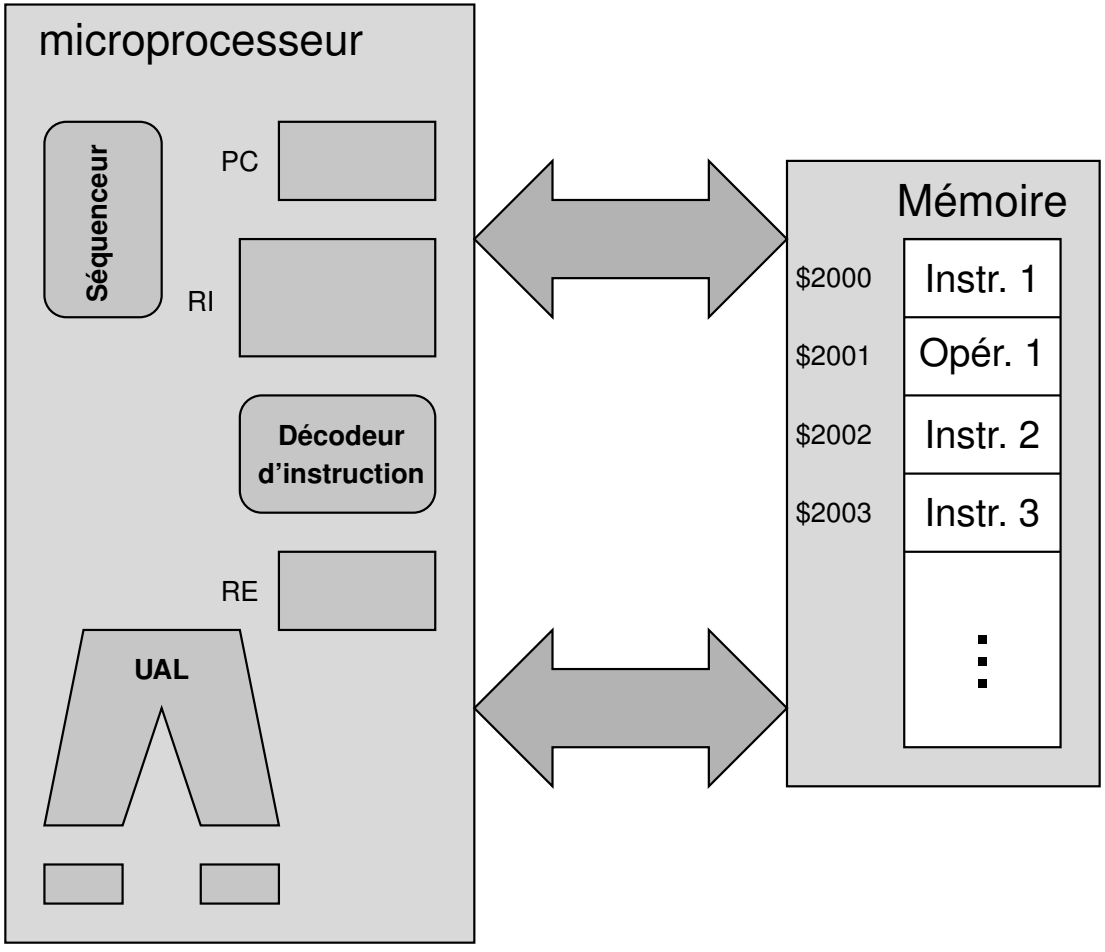
Caractéristiques d'un processeur

- Fréquence d'horloge (MHz) : vitesse de fonctionnement du processeur = nombre de millions de cycles que le processeur est capable d'effectuer par seconde
  - Cycle = plus petite unité de temps au niveau du processeur. Chaque opération/instruction nécessite au minimum un cycle, et plus souvent plusieurs
  - $1 GHz = 10^9 Hz = 10^9 \text{cycle/s}$
- Largeur (32 ou 64 bits) : notamment du bus de données et des registres internes. Bon indicateur de la quantité d'information que celui-ci peut gérer en un temps donné
- Jeu d'instructions : ensemble des opérations qu'un processeur peut exécuter, plus ou moins complexes

Adressage mémoire

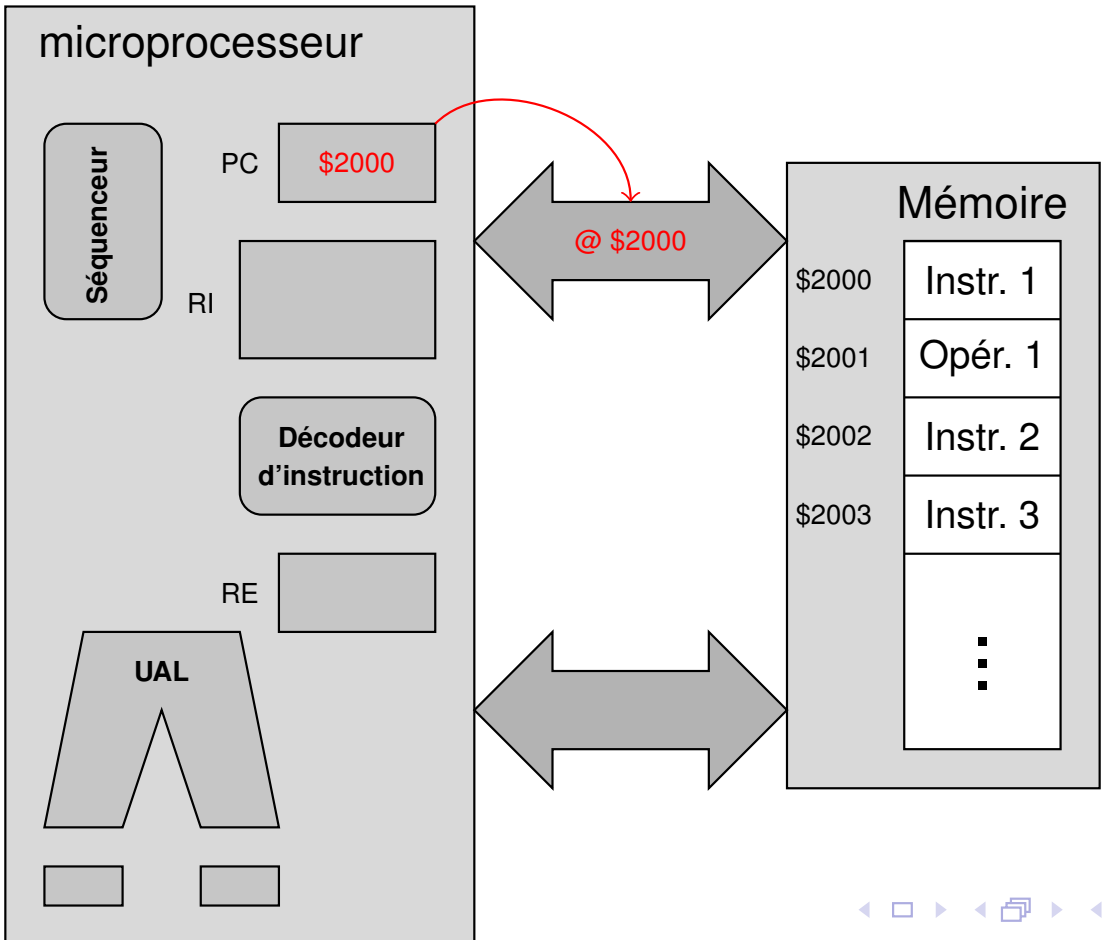
- Par définition, un processeur 32 bits utilise 32 bits pour se référer à la position de chaque octet de la mémoire.
- On a donc $2^{32} = 4.2$ milliards, ce qui signifie que des adresses mémoires de 32 bits de longs peuvent se référer à seulement 4,2 milliards de positions uniques.
- 1 position = 1 mot = 1 octet, donc 4 Go de mémoire vive adressable maximale

Cycle d'exécution d'une instruction



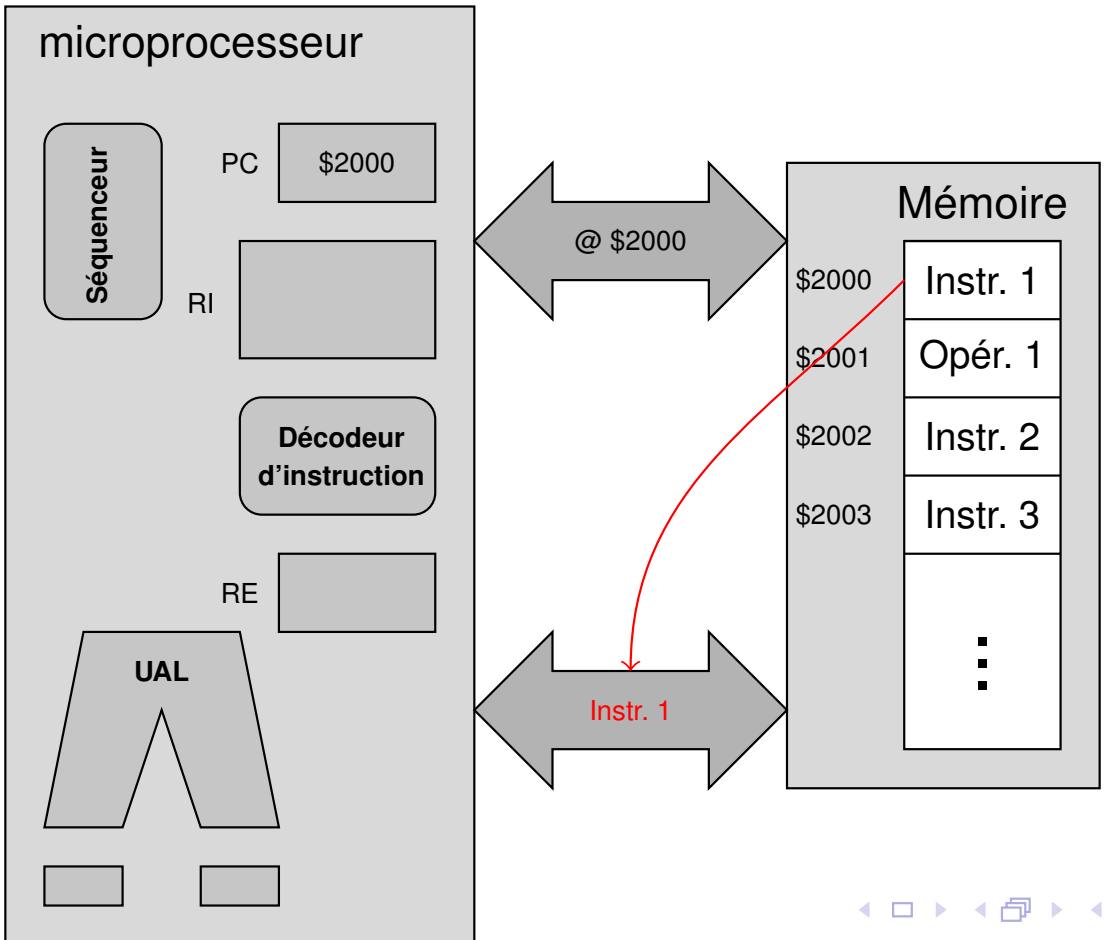
Cycle d'exécution d'une instruction

1. Recherche de l'instruction à traiter :
   1. Le PC contient l'adresse de l'instruction à traiter : cette valeur est placé dans le bus d'adresse avec un ordre de lecture



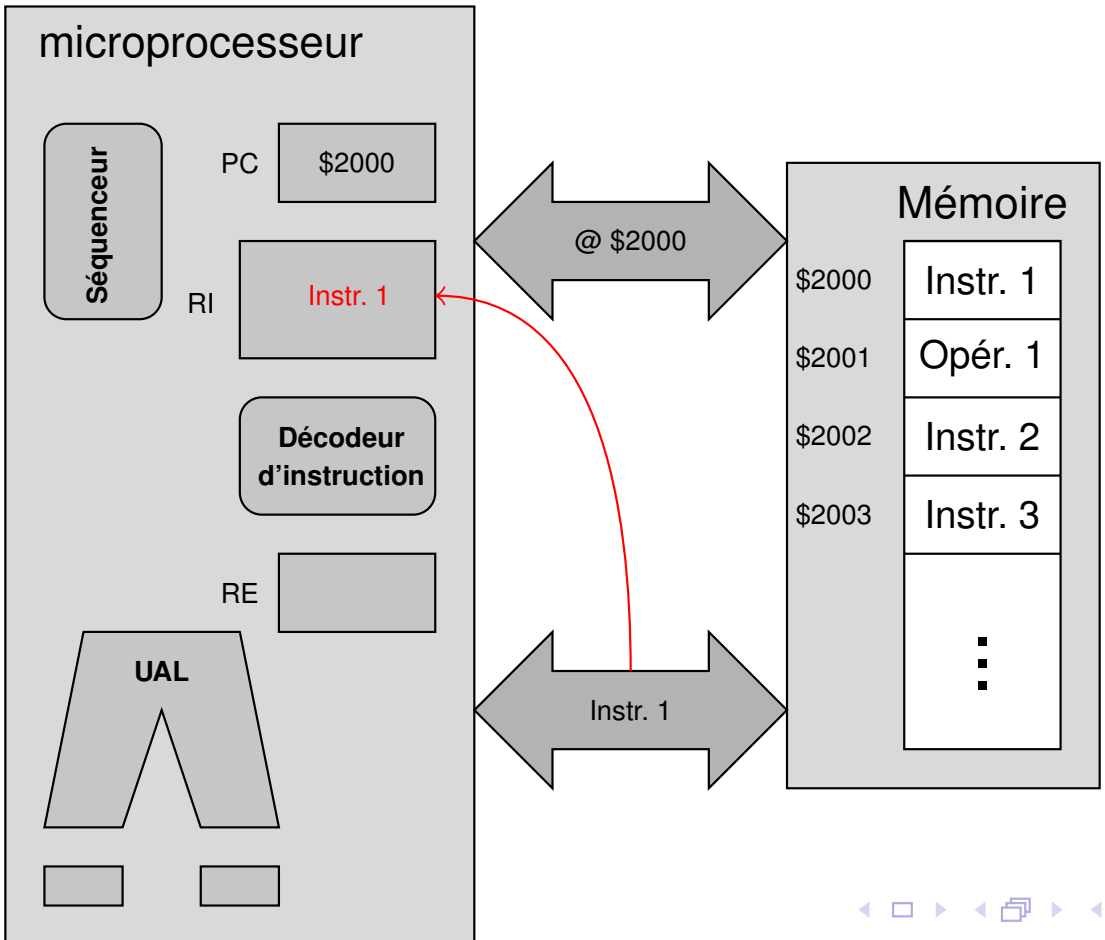
Cycle d'exécution d'une instruction

1. Recherche de l'instruction à traiter :
   1. Le PC contient l'adresse de l'instruction à traiter : cette valeur est placé dans le bus d'adresse avec un ordre de lecture
   2. Après un temps d'accès à la mémoire, le contenu de la case mémoire est mise dans le bus de données



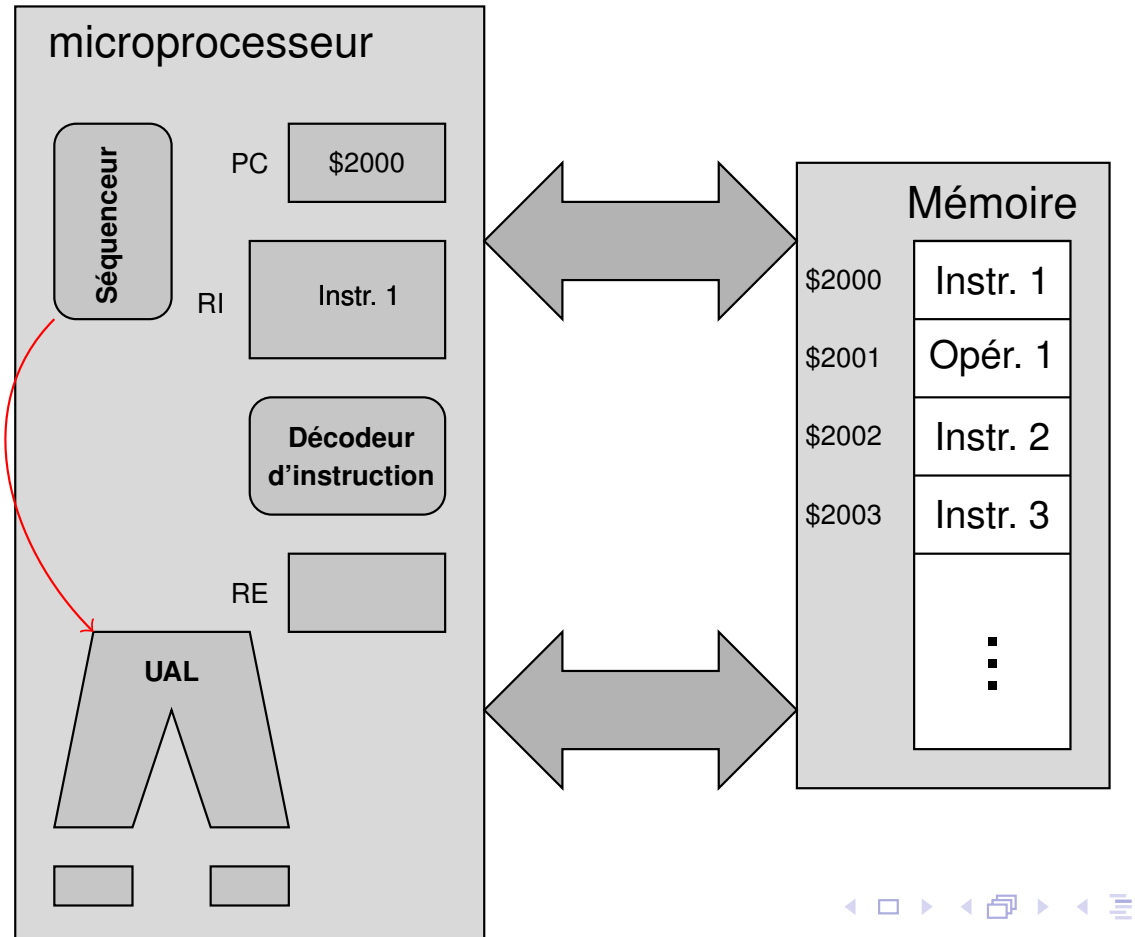
Cycle d'exécution d'une instruction

1. Recherche de l'instruction à traiter :
   1. Le PC contient l'adresse de l'instruction à traiter : cette valeur est placé dans le bus d'adresse avec un ordre de lecture
   2. Après un temps d'accès à la mémoire, le contenu de la case mémoire est mise dans le bus de données
   3. L'instruction est stockée dans le RI



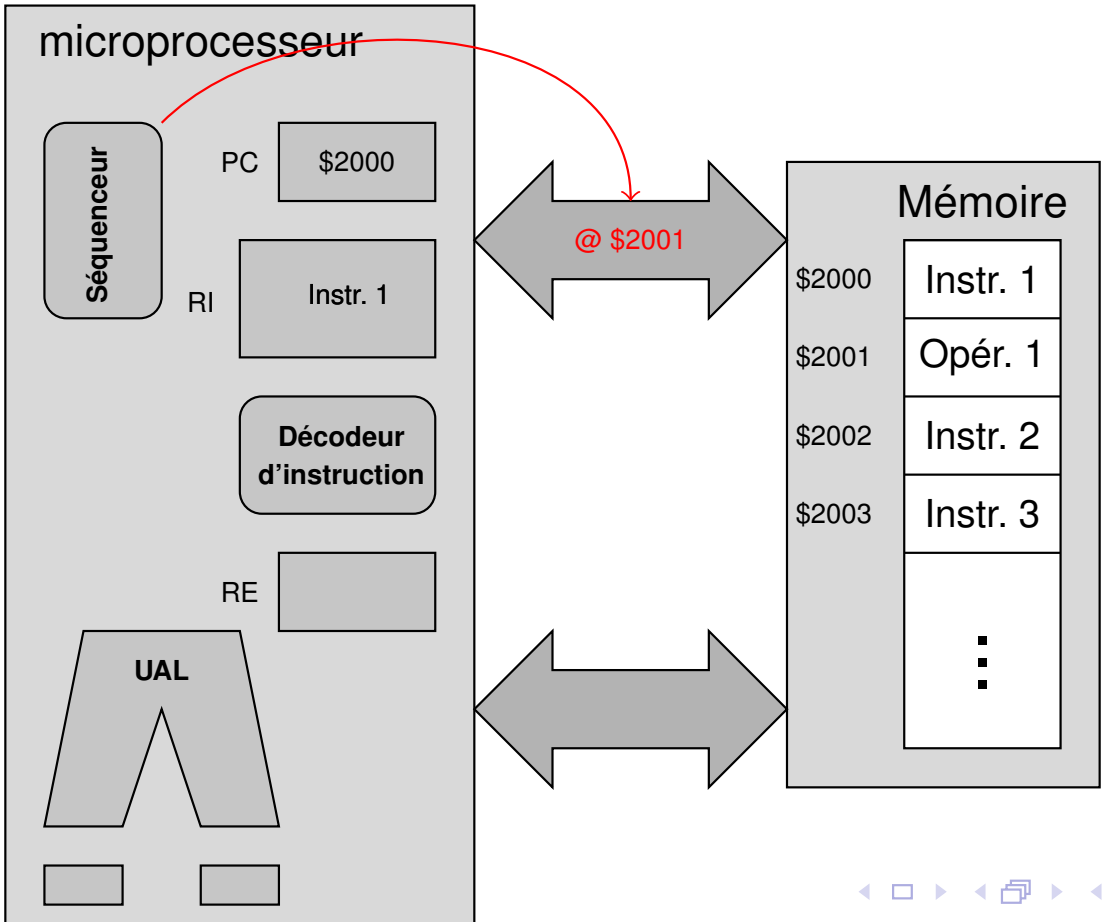
Cycle d'exécution d'une instruction

2. Décodage de l'instruction et recherche de l'opérande :
   1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires



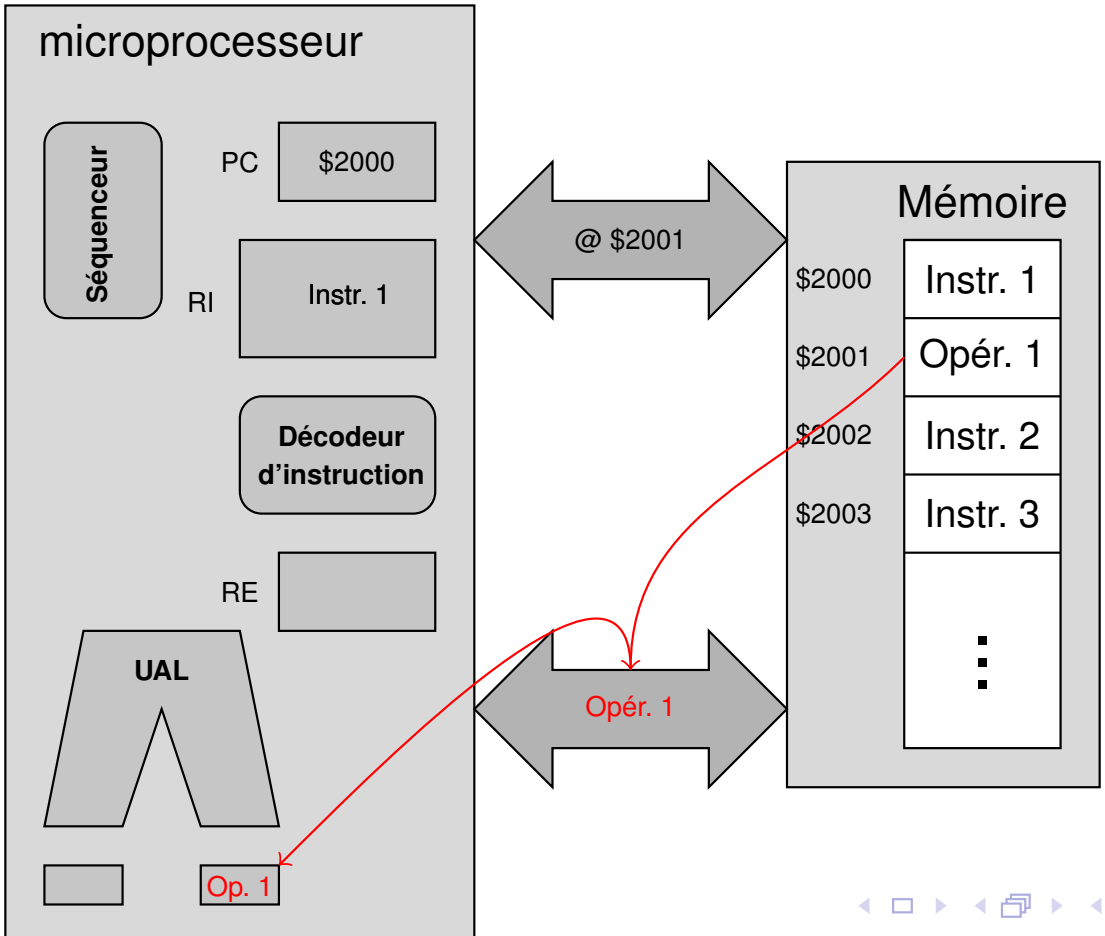
Cycle d'exécution d'une instruction

2. Décodage de l'instruction et recherche de l'opérande :
   1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires
   2. Elle récupére la donnée (opérande) sur laquelle doit opérer l'instruction



Cycle d'exécution d'une instruction

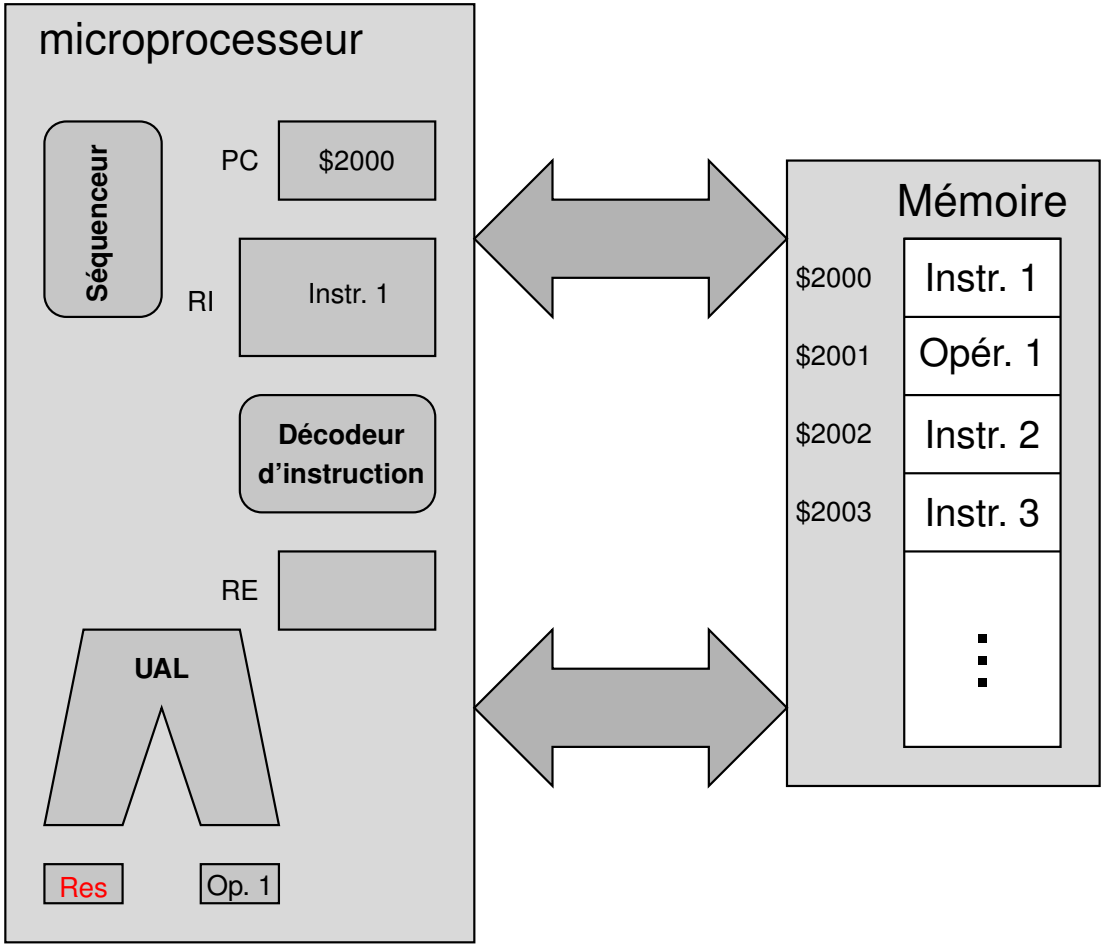
2. Décodage de l'instruction et recherche de l'opérande :
   1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires
   2. Elle récupére la donnée (opérande) sur laquelle doit opérer l'instruction
   3. Cette opérande est placée dans un registre



Cycle d'exécution d'une instruction

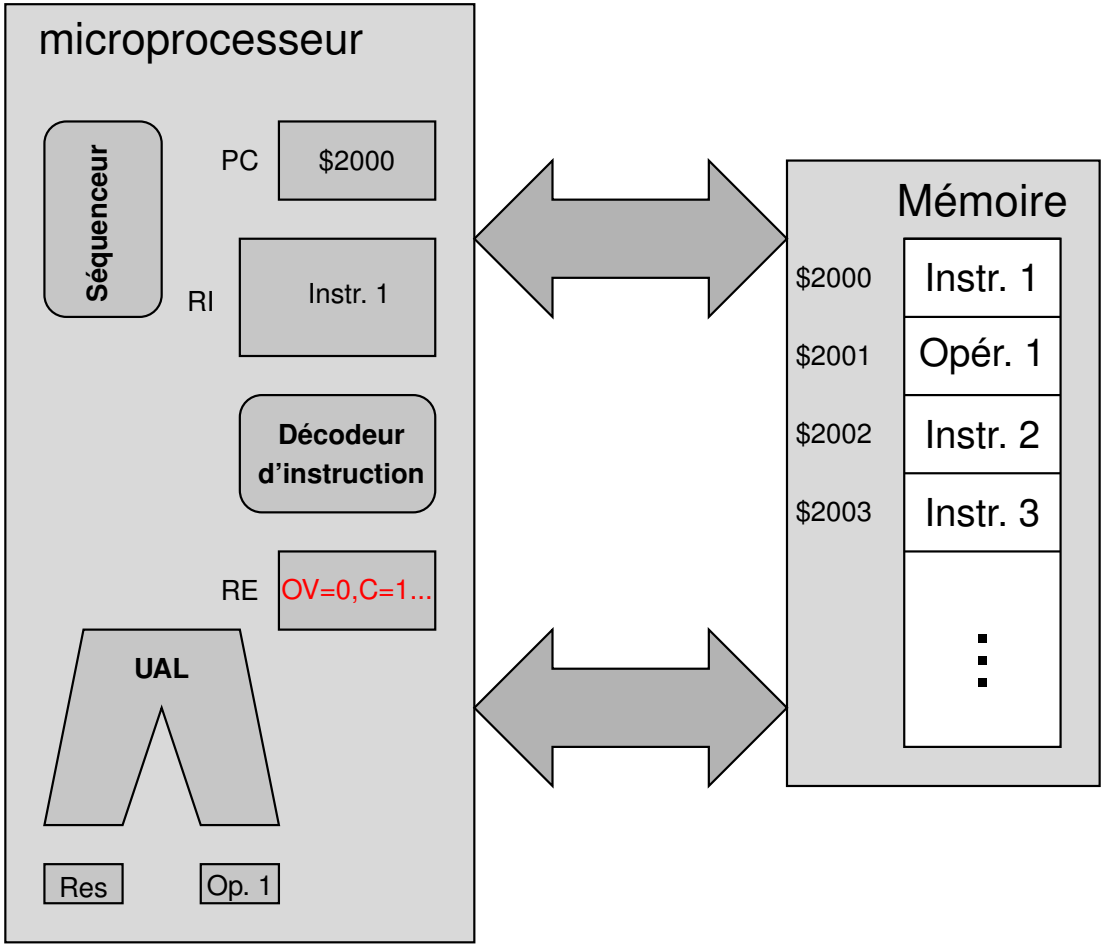
3 Exécution de l'instruction :

1 Le micro-programme réalisant l'instruction est exécuté



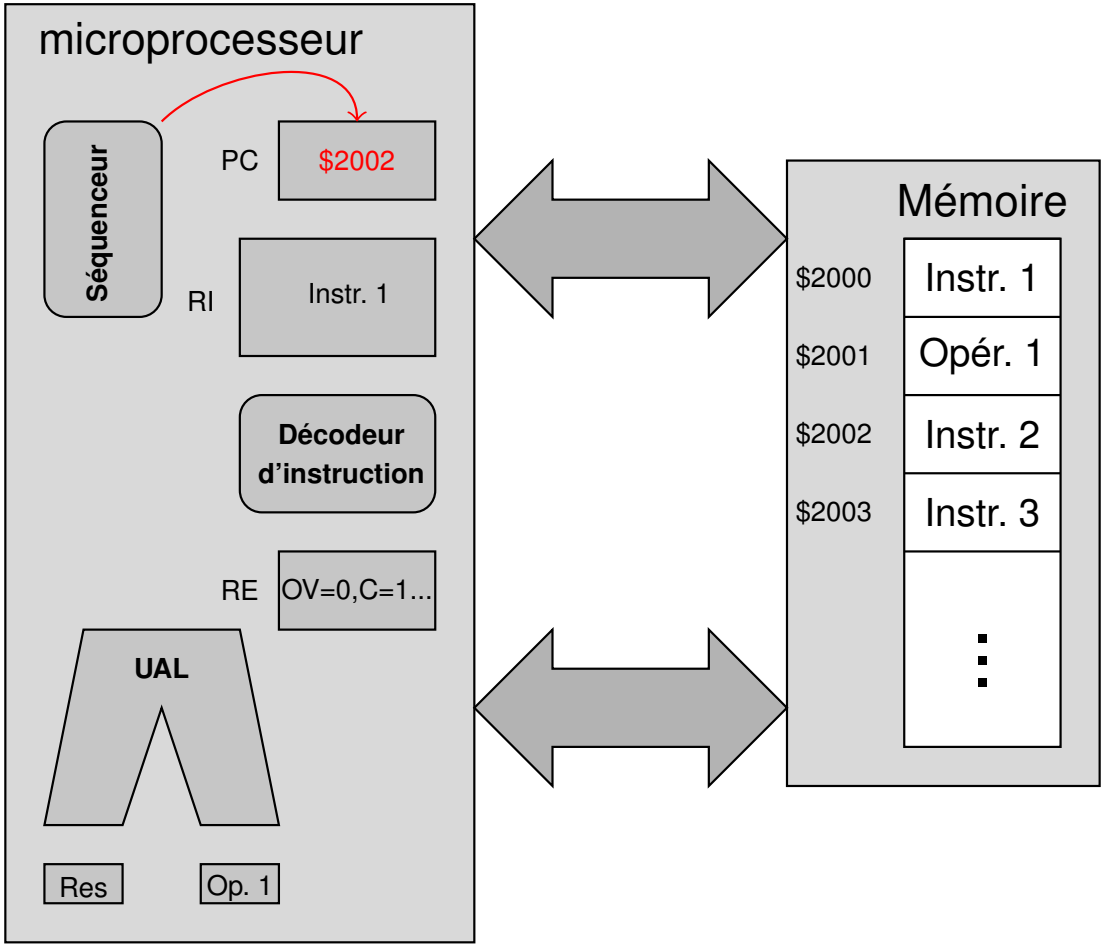
Cycle d'exécution d'une instruction

3. Exécution de l'instruction :
   1. Le micro-programme réalisant l'instruction est exécuté
   2. Les drapeaux sont positionnés (registre d'état)



Cycle d'exécution d'une instruction

3. Exécution de l'instruction :
   1. Le micro-programme réalisant l'instruction est exécuté
   2. Les drapeaux sont positionnés (registre d'état)
   3. L'unité de commande positionne le compteur d'instructions sur l'instruction suivante



Performances

- Chaque instruction nécessite un certain nombre de cycles d'horloge pour s'effectuer.
- Le nombre de cycles dépend de la complexité de l'instruction.
- La durée d'un cycle dépend de la fréquence d'horloge du séquenceur.

Puissance d'un microprocesseur

Nombre d'instructions qu'il est capable de traiter par seconde

- CPI (Cycle par Instruction) : nombre moyen de cycles d'horloge nécessaire pour l'exécution d'une instruction pour un microprocesseur donné.
- MIPS (Millions d'Instructions Par Seconde) : puissance de traitement du microprocesseur.

$$MIPS = \frac{\text{Fréquence (en MHz)}}{CPI}$$

Augmenter les performances

- augmenter la fréquence d'horloge (limites matérielles)
- diminuer le CPI

Sommaire

Pipeline

- L'exécution d'une instruction est décomposée en une succession d'étapes.
- Chaque étape correspond à l'utilisation d'une des fonctions du micro-processeur.
- Lorsqu'une instruction se trouve dans l'une des étapes, les composants associés aux autres étapes ne sont pas utilisés.

Principe du pipeline

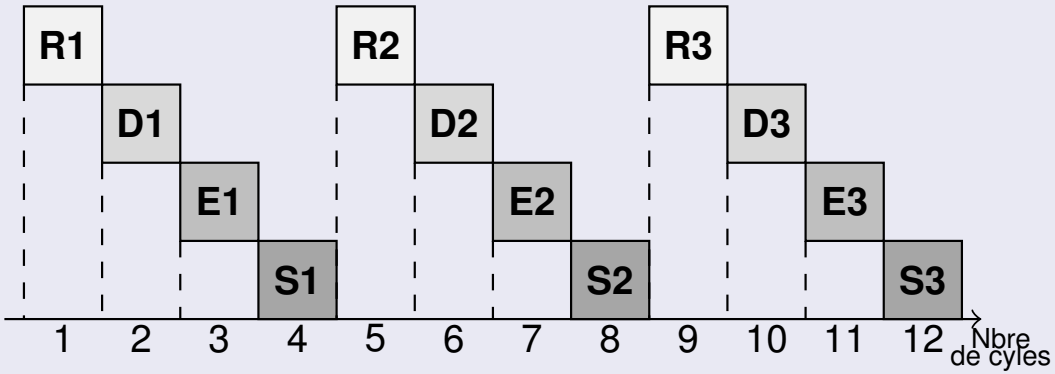
- Une instruction en cours dans chacune des étapes
- Utilisation de chacun des composants du micro-processeur à chaque cycle d'horloge

Une machine pipeline se caractérise par le nombre d'étapes utilisées pour l'exécution d'une instruction (on parle d'étages).

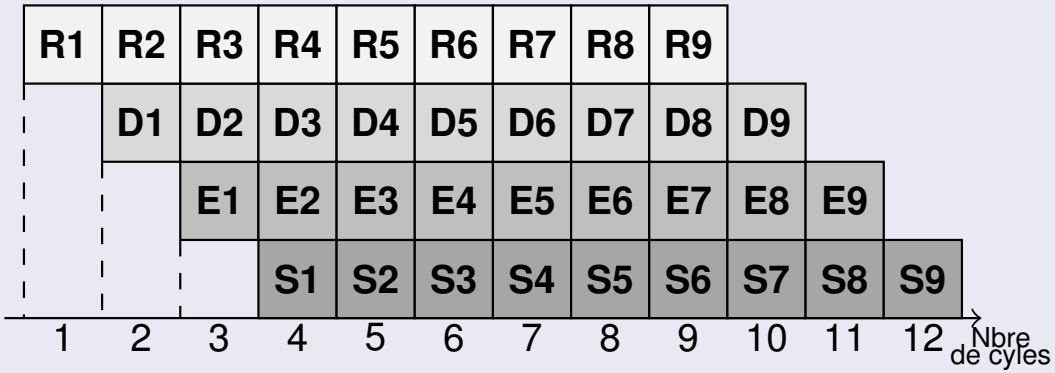
Exemple de pipeline à 4 étages

Recherche	Décodage	Exécution	Sauv. résultat

Modèle classique



Modèle pipeliné à 4 étages



Intérêts du pipeline

Gain en performances

- Pour exécuter n instructions nécessitant chacune k cycles d'horloge, il faut :
  - $n \times k$ cycles d'horloge pour une exécution séquentielle
  - k cycles pour la 1ère instruction, puis $n-1$ cycles pour les $n-1$ instructions restantes si on utilise un pipeline à k étages
- Gain :

$$G = \frac{n \times k}{k + n - 1}$$

- Si n est grand par rapport à k, on divise le temps d'exécution par k.

Inconvénients du pipeline

Problèmes potentiels

- Le temps de traitement de chaque étape doit être à peu près égal sinon les unités rapides doivent attendre les plus lentes.
- Si le pipeline est long :
  - aléa structurel quand 2 instructions ont besoin d'utiliser la même ressource du processeur
  - aléa de données quand 1 instruction produit un résultat utilisé par l'instruction suivante
  - aléa de contrôle lors des instructions de branchement. Il faut alors normalement attendre de connaître l'adresse de destination du branchement.

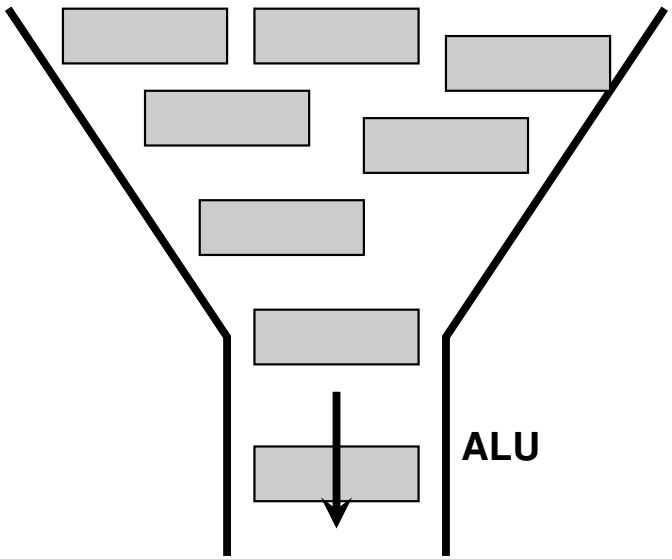
Prédiction de branchement ou prefetching

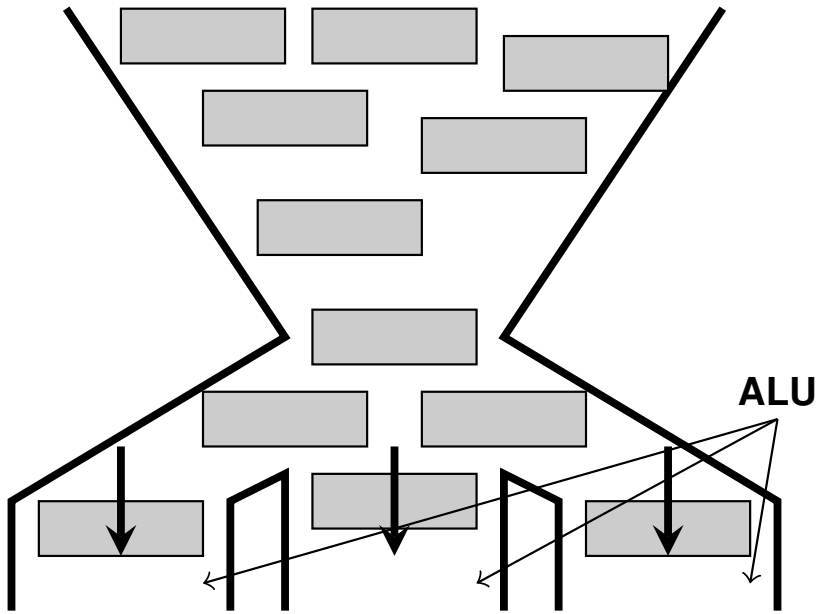
- Recenser lors des branchements le comportement le plus probable
- Fiabilité d'environ 90 à 95%

Architecture superscalaire

Principe

- Exécuter plusieurs instructions en même temps.
- Doter le micro-processeur de plusieurs unités de traitement travaillant en parallèle
- Nécessite de disposer d'un flot de données/instructions important





Retour aux performances

Rappel

- CPI (Cycle par Instruction) : nombre moyen de cycles d'horloge nécessaire pour l'exécution d'une instruction pour un microprocesseur donné.
- MIPS (Millions d'Instructions Par Seconde) : puissance de traitement du microprocesseur.

$$MIPS = \frac{\text{Fréquence (en MHz)}}{CPI}$$

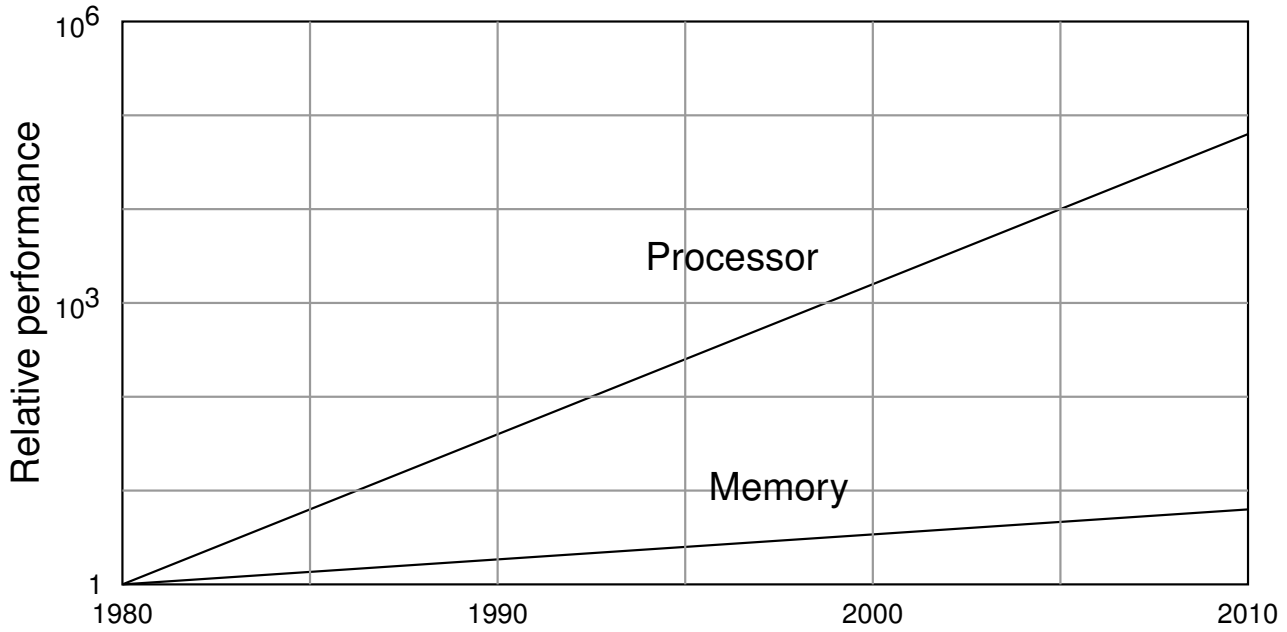
- Ce qu'on vient de voir permet d'améliorer le débit du traitement des flots d'instructions : on peut traiter plus d'instructions en même temps
  - Exemple
- Mais le nombre de cycles par instruction dépend aussi du mode d'adressage : la façon dont le micro-processeur va accéder aux opérandes, donc à la mémoire.

Sommaire

Problématique

« Memory Wall »

$$Vitesse_{CPU} \gg Vitesse_{Memory}$$



Sommaire

Principe

- Mémoire divisée en emplacements de taille fixe (en général 8 bits)
- Chaque emplacement est repéré par un identifiant unique : l'adresse, le plus souvent écrite en hexadécimal.

Adresse	Case mémoire
7=111	
6=110	
5=101	
4=100	
3=011	
2=010	
1=001	
0=000	0001 1010

Liaison processeur-mémoire : les bus

- Circulation des informations entre mémoire et processeur sur un bus
- Bus = ensemble de *n* fils conducteurs utilisés pour transporter *n* signaux binaires
- Bus d'adresse unidirectionnel : seul le processeur envoie des adresses
- Bus de données bidirectionnel :
  - Lecture : la mémoire envoie le mot sur le bus
  - Ecriture : le processeur envoie la donnée

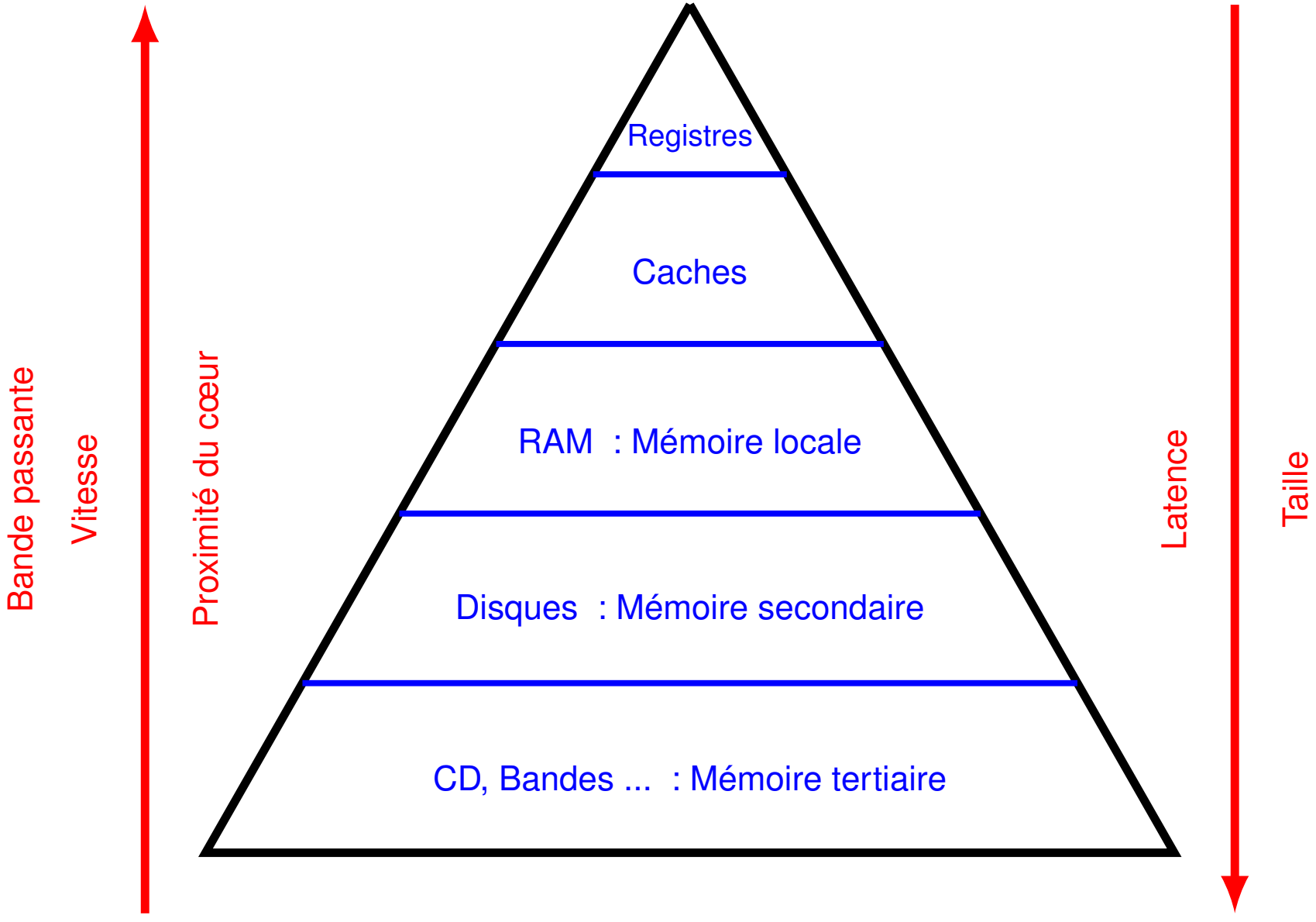
Sommaire

Caractéristiques

- Capacité : nombre total de bits que contient la mémoire
- Format des données : nombre de bits que l'on peut mémoriser par case mémoire. C'est la largeur du mot mémorisable.
- Temps d'accès : Temps qui s'écoule entre l'instant où a été lancée une opération de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données
- Temps de cycle : intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture
- Débit : nombre maximum d'informations lues ou écrites par seconde
- Volatilité : caractérise la permanence des informations dans la mémoire. L'information est volatile si elle risque d'être altérée par un défaut d'alimentation électrique.

Sommaire

Hiérarchie Mémoire



Sommaire

Principes de localité

Localité spatiale

Lorsqu'un programme accède à une donnée ou à une instruction, il est probable qu'il accédera ensuite aux données ou instructions voisines

Localité temporelle

Lorsqu'un programme accède à une donnée ou à une instruction, il est probable qu'il y accédera à nouveau dans un futur proche

Exemple

```
subroutine sumVec(vec,n)
    integer :: n
    integer :: vec(n)
    integer :: i,sum=0
    do i = 1,n
        sum = sum + vec(i)
    end do
end subroutine
```

- *vec(i), vec(i+1), vec(i+2)* : localité spatiale, accès en séquence
- *n, vec, i* : localité temporelle, accès fréquent

Mémoire cache

Principe

- Le processeur a besoin d'un débit soutenu en lecture d'instructions et de données
- Pour ne pas devoir attendre sans rien faire

Problème

La mémoire centrale qui stocke ces instructions et données est beaucoup trop lente pour assurer ce débit

Idée

- Utiliser une mémoire très rapide intermédiaire entre la mémoire centrale et le processeur : la mémoire cache
- Exploiter la localité mémoire

Fonctionnement des caches

- Le cache est divisé en lignes (ou blocs) de mots
- 2 niveaux de granularité :
  - le CPU travaille sur des mots (par ex 32 ou 64 bits)
  - les transferts mémoire se font par ligne (ou bloc, par ex 256 octets)
- Les lignes de caches sont organisées en ensembles à l'intérieur du cache, la taille de ces ensembles est constante et appelée le degré d'associativité.
- Exploitation de la localité spatiale : le cache contient des copies des mots par lignes de cache
- Exploitation de la localité temporelle : choix judicieux des lignes de cache à retirer lorsqu'il faut rajouter une ligne à un cache déjà plein

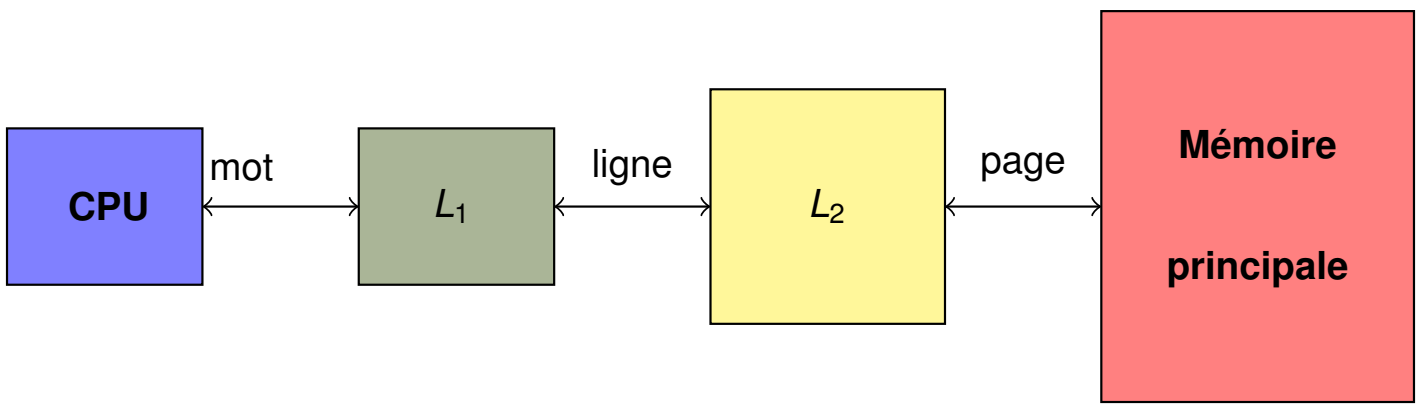
Lorsque le processeur tente d'accéder à une information (instruction ou donnée)

Si l'information se trouve dans le cache (hit), le processeur y accède sans état d'attente, sinon (miss) le cache est chargé avec un bloc d'informations de la mémoire

Organisation des caches

Niveaux de caches

- Cache L_1
  - le plus rapide, mais le plus petit
  - scindé en 2 parties : données et instructions (notion de voisinage s'applique différemment pour les instructions et pour les données)
- Le cache L_{i+1} jour le rôle de cache pour le niveau L_i



Caches miss

- Défaut de cache (cache miss) : lorsque le processeur essaie d'accéder une donnée qui n'est pas dans le cache
  - Raté obligatoire (compulsory miss) : premier accès au bloc pendant l'exécution du programme. Inévitable.
  - Raté par capacité insuffisante (capacity miss) : accès à un bloc qui a été remplacé par un autre dans le cache car la capacité du cache est insuffisante.

Exemple : Balayage d'un grand tableau

```
do  i=1,n
    do  j=1,m
        ...  a(i,j)  ...
    end  do
end  do
```

- Hypothèse : architecture 32 bits, cache L1 de 16 Ko avec des lignes de 64 octets.
- Quel est le taux d'échec si $n = m = 256$?
- Taux d'échec = nombre total d'échecs divisé par le nombre total d'accès.

Sommaire

Exemple

Problème

On considère la suite :

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_n = 111 - \dfrac{1130}{u_{n-1}} + \dfrac{3000}{u_{n-1}u_{n-2}} \end{cases}$$

Elle converge vers ?

Programme

```
for (i = 3 ; i <= max ; i++)
  {
    w = 111. - 1130./v + 3000./(v*u);
    u = v;
    v = w;
    printf("u%d = %1.17g\n",i,v);
  }
```

Exemple

Problème

On considère la suite :

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_n = 111 - \dfrac{1130}{u_{n-1}} + \dfrac{3000}{u_{n-1}u_{n-2}} \end{cases}$$

Elle converge vers ? 6

Programme

```
for (i = 3 ; i <= max ; i++)
  {
    w = 111. - 1130./v + 3000./(v*u);
    u = v;
    v = w;
    printf("u%d = %1.17g\n",i,v);
  }
```

Vérifions !!

Erreurs de calcul

Quand on cherche à résoudre un problème physique, biologique, ... avec un ordinateur, plusieurs sources d'erreur :

- Erreur sur les données : imprécision de mesures physiques
- Erreur sur les modèles : simplification de la complexité d'un phénomène
- Erreur d'approximation ou de discrétisation : l'analyse numérique nous permet de la quantifier
- Erreur due à l'algorithme si il est itératif : accumulation d'erreurs d'arrondi
- Erreur due à la précision finie :
  - Représenter les nombres réels : Passer de l'infini au fini

Erreurs de calcul

Quand on cherche à résoudre un problème physique, biologique, ... avec un ordinateur, plusieurs sources d'erreur :

- Erreur sur les données : imprécision de mesures physiques
- Erreur sur les modèles : simplification de la complexité d'un phénomène
- Erreur d'approximation ou de discrétisation : l'analyse numérique nous permet de la quantifier
- Erreur due à l'algorithme si il est itératif : accumulation d'erreurs d'arrondi
- Erreur due à la précision finie :
  - Représenter les nombres réels : Passer de l'infini au fini

On s'intéresse ici aux erreurs liées à la représentation finie des nombres réels

Sommaire

Représentation flottante

En virgule flottante, en base β, un nombre réel x est représenté par :

- un signe $s \in \{0, 1\}$
  - 0 : positif
  - 1 : négatif
- une mantisse m, écrite en virgule fixe en base β sur p chiffres appelés digit
- un exposant $e \in \{e_{min}, ..., e_{max}\}$

$$x = (-1)^s \times m \times \beta^e$$

avec pour $k \in \{0, \ldots - 1\}$:

$$m = m_0 \cdots m_i . m_{i+1} \cdots m_{p-1} \text{ et } m_k \in \{0, \ldots, \beta - 1\}$$

- On dit que le nombre flottant x est de précision p (avec $p \geq 1$)

Limites

Propriétés et définitions

- Plus petit nombre représentable : $\epsilon = \pm 0.1\beta^{e_{min}}$
- Plus grand nombre représentable : $M = \pm 0.(\beta - 1)(\beta - 1)\ldots(\beta - 1)\beta^{e_{max}}$
- Dépassement de capacité : Si l'on cherche à représenter un nombre plus petit que ϵ, on produit un débordement par valeur inférieure (underflow). Si l'on cherche à représenter un nombre plus grand que M, on produit un débordement par valeur supérieure (overflow).

Limites

- Certains réels sont par définition impossibles à représenter en numération classique : $1/3$, π ...
- La représentation en un nombre fixe d'octets oblige le processeur de faire appel à des approximations afin de représenter les réels.
- Le degré de précision de la représentation par virgule flottante des réels est directement proportionnel au nombre de bits alloué à la mantisse, alors que le nombre de bits alloué à l'exposant conditionnera l'amplitude de l'intervalle des nombres représentables.

Opérations flottantes

Opérations élémentaires

- $x + y \longrightarrow fl(fl(x) + fl(y))$
- $x - y \longrightarrow fl(fl(x) - fl(y))$ ...

Attention

Plusieurs propriétés de l'arithmétique (associativité, distributivité, ...) ne sont plus valides en arithmétique flottante !

Exemple

```
real   :: x,y,y1,z,z1,w
data x/77777/, y/7/
data w /.01/
y1 = 1/y
z = x/y
z1 = x*y1
if (z .ne. z1) print *, "Not Equal !"
if ( w*100.d0 .ne. 1.0) then
   print *, "Many systems print this surpringly !!"
else
   print *, "Some systems print this"
end if
```

Sommaire

Problématique

- Jusque dans les années 80 : chaque constructeur avait sa propre implantation de l'arithmétique flottante
  - quelle base β était utilisée ? quelle plage d'exposant $[e_{min}, e_{max}]$?
  - un programme : différents résultats sur différentes architectures
- Besoin de standardiser et d'homogénéiser l'implantation l'arithmétique virgule flottante en base 2
  - fixer précisement le format des données et leur encodage en machine
  - définir le comportement et la précision des opérations de base
  - définir les valeurs spéciales, les modes d'arrondis, et la gestion des exceptions
- 1985 : publication du standard IEEE 754-1985, initié par Prof. William Kahan
  - 2008 : révision de la norme : IEEE 754-2008

Formats de représentation standards des données

	Simple Précision	Double Précision
Précision p	24	53
Taille de l'exposant e	8	11
Taille de la représentation $(p+e)$	32	64
e_{min}, e_{max}	-126, 127	-1022, 1023
Type C	*float*	*double*

Autres apports de la norme IEEE 754

- Flottants spéciaux : $-\infty$, $+\infty$, *NaN*, ...
- Modes d'arrondis : vers $+\infty$, vers $-\infty$, vers 0, vers le nombre flottant le plus proche
- Exceptions : *Invalid operation*, *Overflow*, *Underflow*, *Division par zéro*, ...

Sommaire

Conclusions

Ce qu'il faut retenir

L'écriture d'un code efficace nécessite :

- d'avoir une bonne idée de la façon dont le processeur et la mémoire fonctionnent
- de connaître les optimisations que le processeur permet
- de savoir comment la mémoire est agencée
- d'être conscient des limites de l'arithmétique flottante

Ces notions sont essentielles pour mieux appréhender le comportement d'un code, en matière de temps CPU, d'usage mémoire et de résultats.