Architecture des Ordinateurs Avancée (L3)

Cours 5 - Mieux exploiter le parallélisme : la prédiction de branchements

Carine Pivoteau [1]

1. En collaboration avec X. Goaoc

Retour sur le TP 4 : ...

- Optimiser la récursion
- Combiner les optimisations
- Pourquoi `REP RET` ?

Retour sur le TP 4 : ...

- Optimiser la récursion
- Combiner les optimisations
- Pourquoi `REP RET` ?

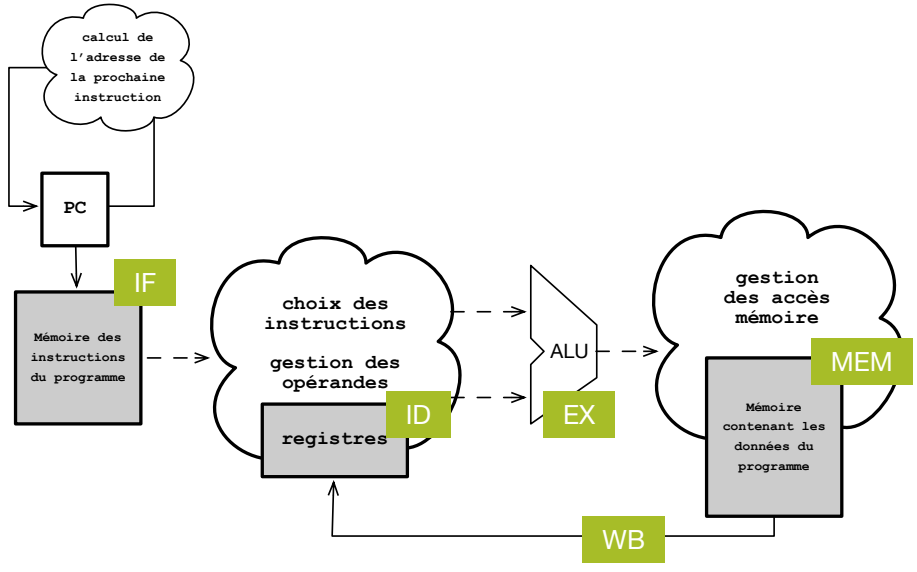
Retour sur le TP 4 : ...

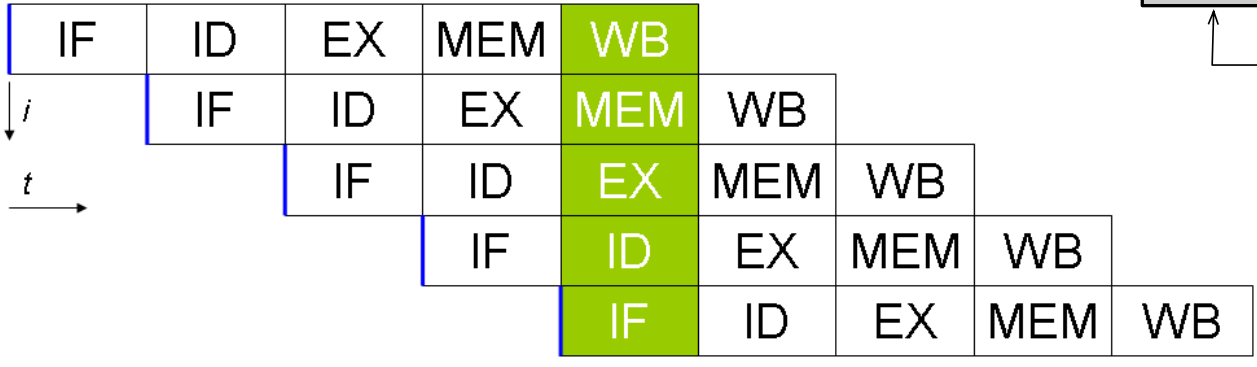
- Optimiser la récursion
- Combiner les optimisations
- Pourquoi `REP RET` ?
  ▷ Explication avant la fin du cours.

Le pipeline : rappel

Dans un processeur qui utlise un pipeline :

- Chaque étape utilise une portion de circuit différente (en gros ...).
- On suppose que chaque étape prend environ un cycle à s'exécuter.
- On cherche à introduire de l'**ILP** (*Instruction Level Parallelism*).
- On peut utiliser le principe d'une chaîne de montage.





Le problème des bulles...

C'est très efficace en théorie... s'il n'y a pas de **bulles** (*stall*) dans le pipeline. Reprenons l'exo précédent (et son code assembleur) :

```
b = a + 13;
a = c + 17;
```

```
lea r13, [r12+13]
lea r12, [r14+17]
```

```
b = a + 13;
a = b + 17;
```

```
lea r13, [r12+13]
lea r12, [r13+17]
```

Le code de droite produit une bulle qui ralentit l'exécution du pipeline... Heureusement, il y a des solutions :

- exécution *Out Of Order* (ré-ordonnancement des instructions),
- renommage des registres,
- propagation des opérandes (*operand forwarding*),
- exécution spéculative, ...

Réordonner les instructions : exécution *Out Of Order*

Exemple de traitement des bulles liées à la dépendance des données (peut être fait de façon logicielle ou matérielle).

Lorsque c'est possible : permuter/modifier les instructions de manière à éliminer les bulles sans changer le résultat du calcul. Par exemple :

```
mov     eax, [rbp-4]
imul    eax, 6 // <- edx:eax
mov     [rbp-4], eax
mov     ebx, [rbp-8]
add     ebx, 2
mov     [rbp-8], ebx
```

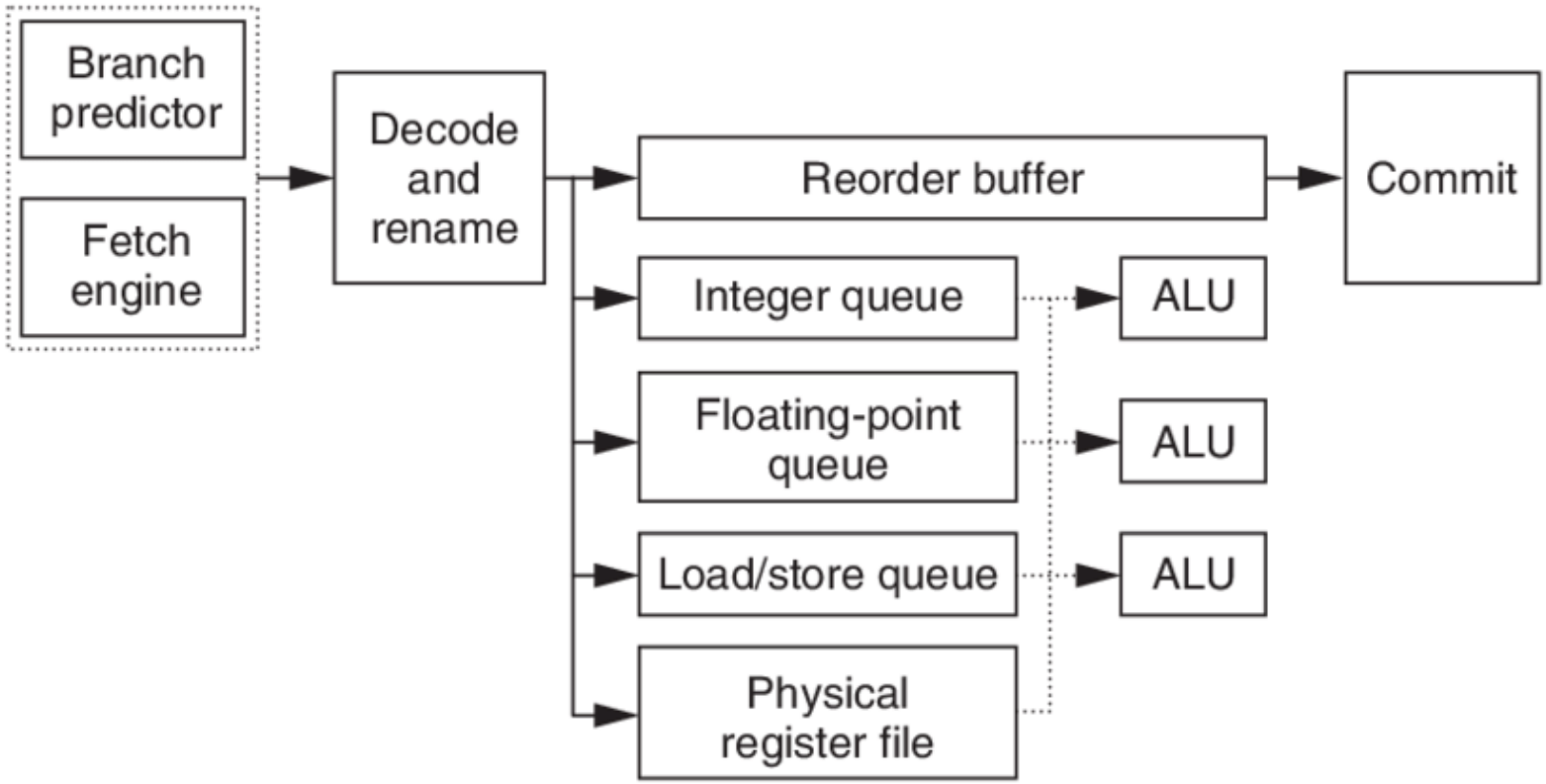
$\rightarrow$

```
mov     eax, [rbp-4]
mov     ebx, [rbp-8]
imul    eax, 6
add     ebx, 2
mov     [rbp-4], eax
mov     [rbp-8], ebx
```

Multiplication des unités de calcul

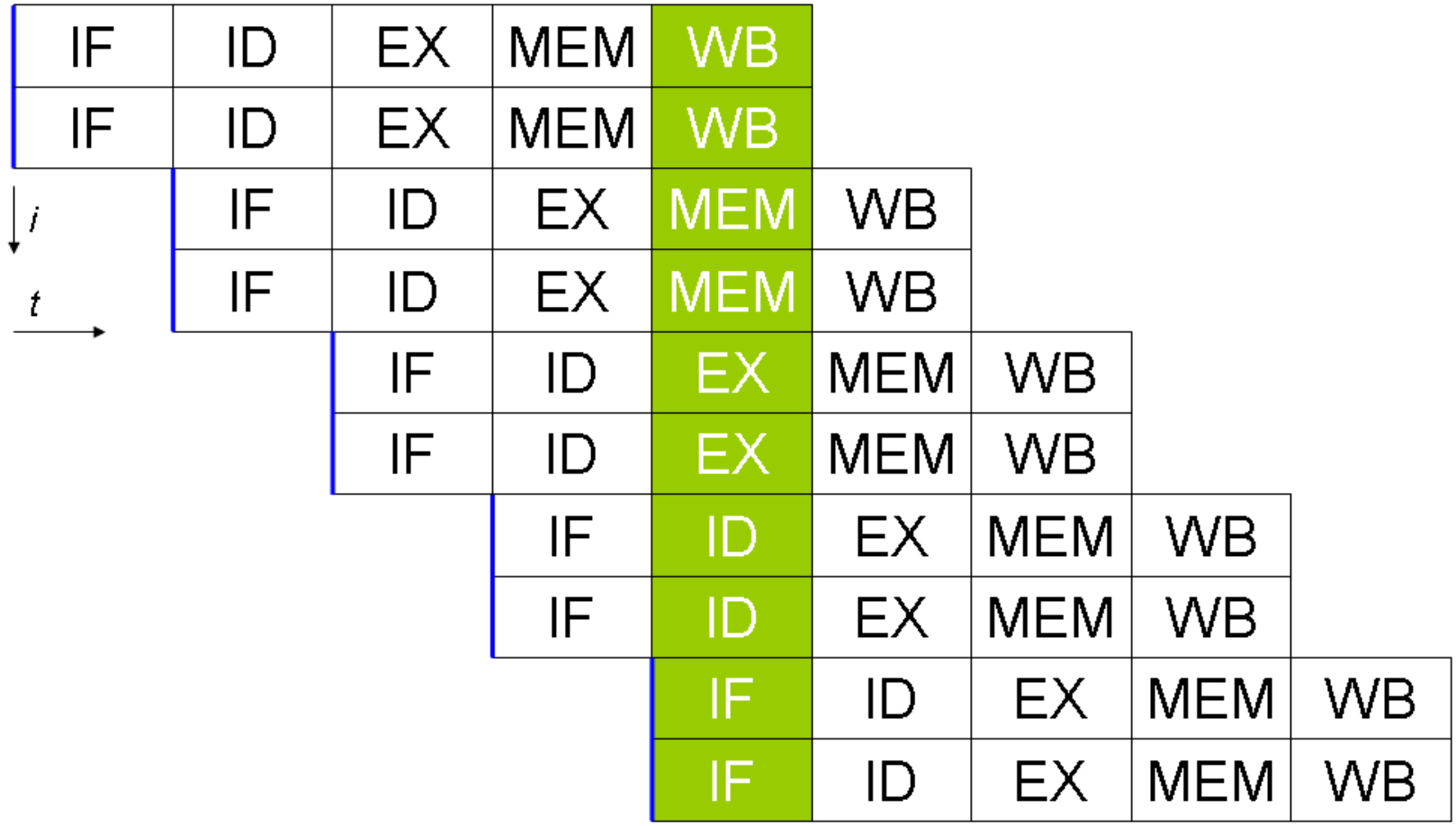
Un processeur *super scalaire*, exécute plusieurs instructions pendant un cycle en les dispatchant à des unités fonctionnelles redondantes (ALU, calcul flottant, unité de multiplication, ...).

Exemple d'architecture super-scalaire (out of order) :



Multiplication des unités de calcul

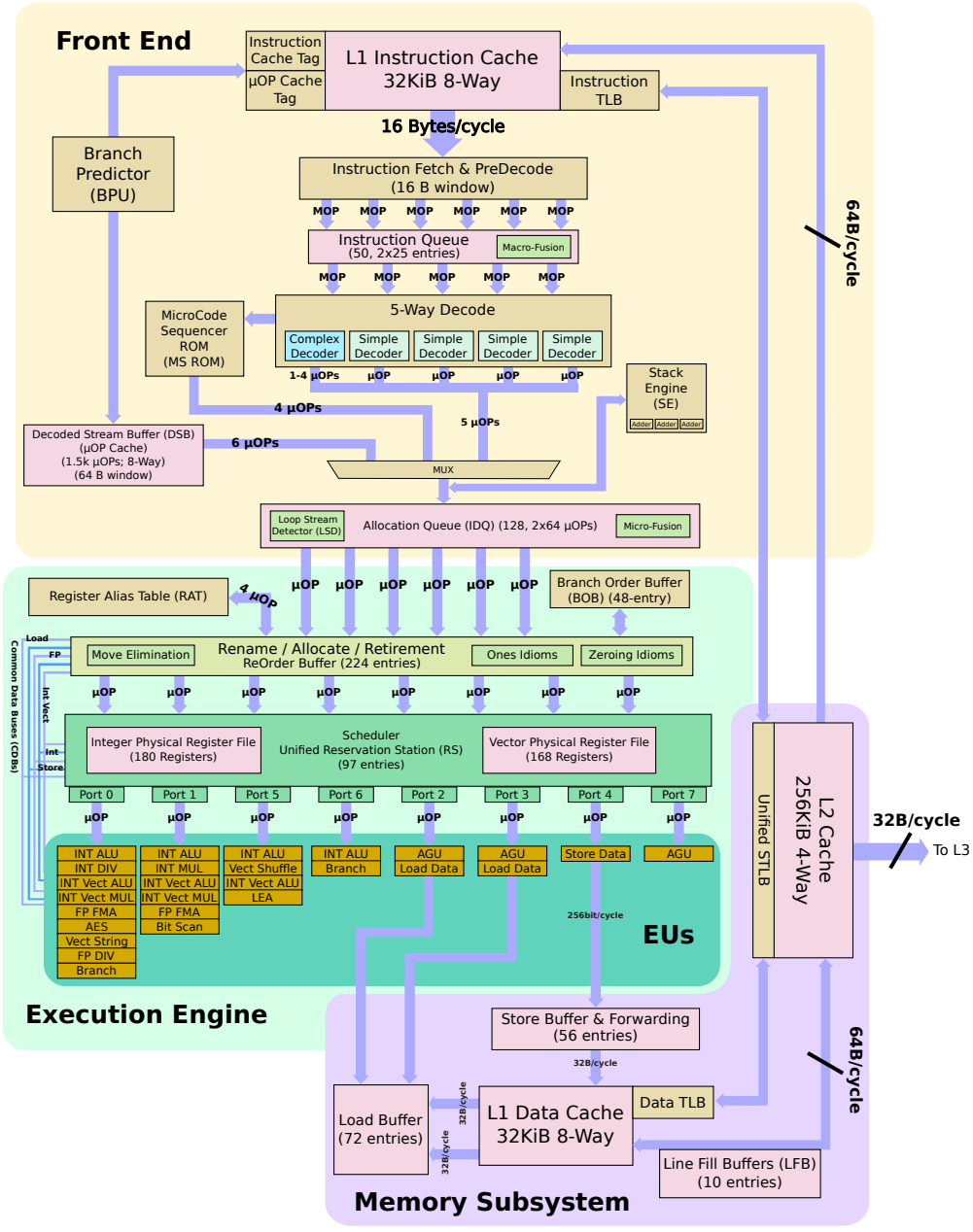
Exécution parallèle (9 cycles pour 10 instructions) :



Architecture Skylake
(Intel Core i3, i5, i7)

source :

`https://en.wikichip.org/wiki/skylake`



Le cas du saut conditionnel

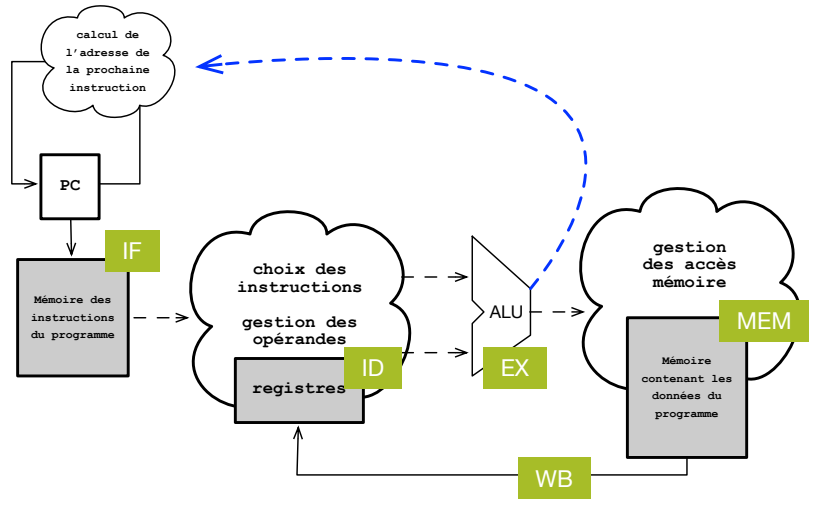
Un saut conditionnel est une instruction qui permet d'effectuer un `if` ou un tour de boucle, par exemple.

```
    mov  edx, 0
    mov  eax, 0
.L: add  edx, eax
    add  eax, 1
    cmp  eax, 10
    jne  .L
    mov  eax, 0
    ...
```

Le cas du saut conditionnel

Un saut conditionnel est une instruction qui permet d'effectuer un `if` ou un tour de boucle, par exemple.

```
    mov  edx, 0
    mov  eax, 0
.L: add  edx, eax
    add  eax, 1
    cmp  eax, 10
    jne  .L
    mov  eax, 0
    ...
```

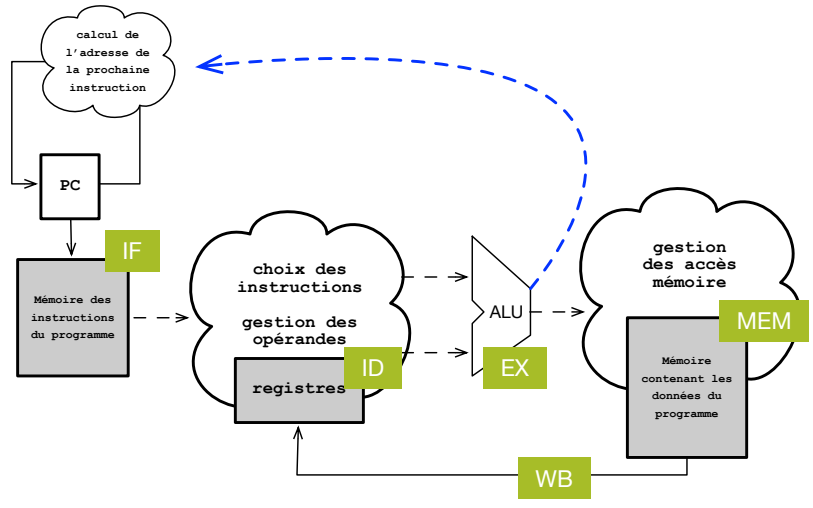


Il faut attendre la fin de l'instruction `jne .L` pour savoir quelle sera la prochaine instruction... Mais ça crée une grosse bulle !

Le cas du saut conditionnel

Un saut conditionnel est une instruction qui permet d'effectuer un `if` ou un tour de boucle, par exemple.

```
    mov  edx, 0
    mov  eax, 0
.L: add  edx, eax
    add  eax, 1
    cmp  eax, 10
    jne  .L
    mov  eax, 0
    ...
```



Il faut attendre la fin de l'instruction `jne .L` pour savoir quelle sera la prochaine instruction... Mais ça crée une grosse bulle !
Autre solution : l'execution spéculative. Et si on se trompe de branche ?

Exo : min et max

- Est-ce qu'on peut limiter l'impact des bulles créées par les sauts conditionnels ?

- On souhaite calculer simultanément le minimum et le maximum d'un tableau de n entiers aléatoires.
- Proposer un algorithme simple pour faire cela. Vous devriez pouvoir y arrivez en faisant de l'ordre de $2n$ comparaisons.
- En réalité, on peut faire mieux : il existe un algorithme qui ne fait que $n + \frac{n}{2}$ comparaisons. Essayer de trouver cet algo.
- Coder les deux algorithmes en C et mesurer le temps d'exécution pour un n assez grand. Qu'observe-t-on ?
- Essayer de trouver une explication au phénomène observé.

(fichier **demo-C5-minmax.c**)

Exo : min et max

- Est-ce qu'on peut limiter l'impact des bulles créées par les sauts conditionnels ?
- On souhaite calculer simultanément le minimum et le maximum d'un tableau de n entiers aléatoires.
- Proposer un algorithme simple pour faire cela. Vous devriez pouvoir y arrivez en faisant de l'ordre de $2n$ comparaisons.
- En réalité, on peut faire mieux : il existe un algorithme qui ne fait que $n + \frac{n}{2}$ comparaisons. Essayer de trouver cet algo.
- Coder les deux algorithmes en C et mesurer le temps d'exécution pour un n assez grand. Qu'observe-t-on ?
- Essayer de trouver une explication au phénomène observé.

(fichier **demo-C5-minmax.c**)

Exo : min et max

- Est-ce qu'on peut limiter l'impact des bulles créées par les sauts conditionnels ?
- On souhaite calculer simultanément le minimum et le maximum d'un tableau de n entiers aléatoires.
- Proposer un algorithme simple pour faire cela. Vous devriez pouvoir y arrivez en faisant de l'ordre de $2n$ comparaisons.
- En réalité, on peut faire mieux : il existe un algorithme qui ne fait que $n + \frac{n}{2}$ comparaisons. Essayer de trouver cet algo.
- Coder les deux algorithmes en C et mesurer le temps d'exécution pour un n assez grand. Qu'observe-t-on ?
- Essayer de trouver une explication au phénomène observé.

(fichier **demo-C5-minmax.c**)

Exo : min et max

- Est-ce qu'on peut limiter l'impact des bulles créées par les sauts conditionnels ?
- On souhaite calculer simultanément le minimum et le maximum d'un tableau de n entiers aléatoires.
- Proposer un algorithme simple pour faire cela. Vous devriez pouvoir y arrivez en faisant de l'ordre de $2n$ comparaisons.
- En réalité, on peut faire mieux : il existe un algorithme qui ne fait que $n + \frac{n}{2}$ comparaisons. Essayer de trouver cet algo.
- Coder les deux algorithmes en C et mesurer le temps d'exécution pour un n assez grand. Qu'observe-t-on ?
- Essayer de trouver une explication au phénomène observé.

(fichier **demo-C5-minmax.c**)

Exo : min et max

- Est-ce qu'on peut limiter l'impact des bulles créées par les sauts conditionnels ?
- On souhaite calculer simultanément le minimum et le maximum d'un tableau de n entiers aléatoires.
- Proposer un algorithme simple pour faire cela. Vous devriez pouvoir y arrivez en faisant de l'ordre de $2n$ comparaisons.
- En réalité, on peut faire mieux : il existe un algorithme qui ne fait que $n + \frac{n}{2}$ comparaisons. Essayer de trouver cet algo.
- Coder les deux algorithmes en C et mesurer le temps d'exécution pour un n assez grand. Qu'observe-t-on ?
- Essayer de trouver une explication au phénomène observé.

(fichier **demo-C5-minmax.c**)

Prédiction de branchement

Branch miss ?

- Lors d'**un saut conditionnel**, on peut déterminer rapidement l'adresse où il *faudrait* sauter dans le programme...
- ... mais il faut attendre la fin de l'instruction (au moins la comparaison) pour savoir si le saut est effectué ou non.
- Cette attente crée un bulle importante dans le pipeline.
- Pour éviter cela, on peut essayer de **"deviner"** le résultat du test.
- Ainsi, on anticipe la branche à prendre.
- Si on s'est trompé (erreur de prédiction), c'est un *branch miss* et on doit vider le pipeline (ça ne coûte pas beaucoup plus cher que la bulle qui aurait eu lieu sans anticipation).
- Sinon, on a profité pleinement du pipeline, en conservant un bon taux d'instructions par cycle.

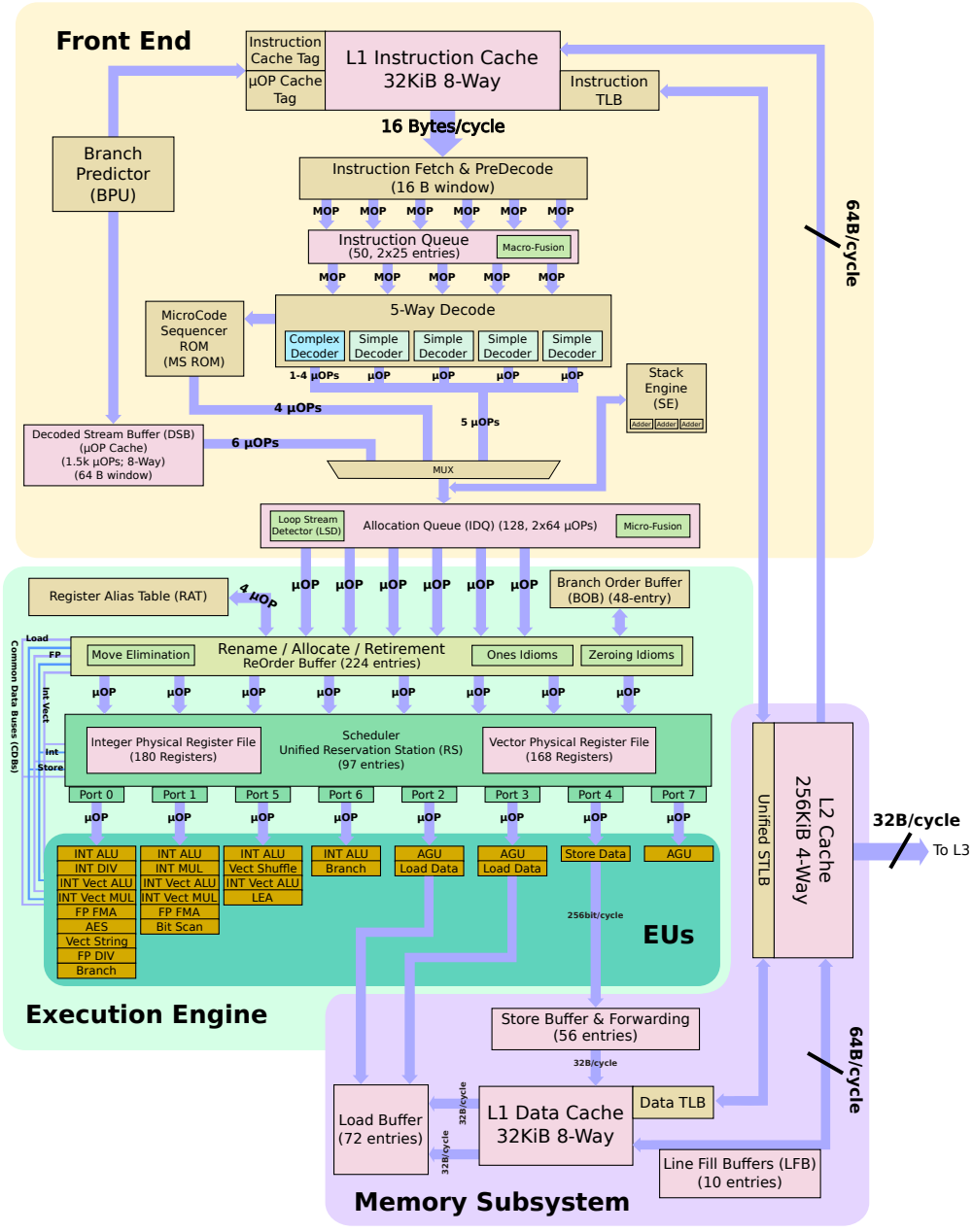
Les prédicteurs de branchement

- Prédicteur statique.
- Prédicteur dynamique : on utilise des informations sur le déroulement de l'exécution pour prédire.
- Une idée simple est d'utiliser les résultats du/des branchements précédents (système à états, table d'historique, ...).
- Le prédicteur peut être local (attaché à un seul branchement) ou global (il utilise l'information venant de l'ensemble des branchements du programme).
- Dans les processeur actuels, on ne sait pas précisément ce qui est implanté (secret industriel), mais c'est probablement une combinaison de plusieurs prédicteurs, avec à la fois du local et du global, le choix se faisant dynamiquement.

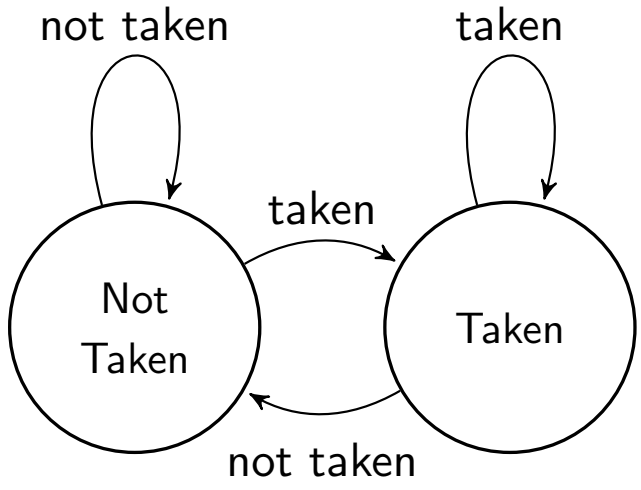
Architecture Skylake
(Intel Core i3, i5, i7)

source :

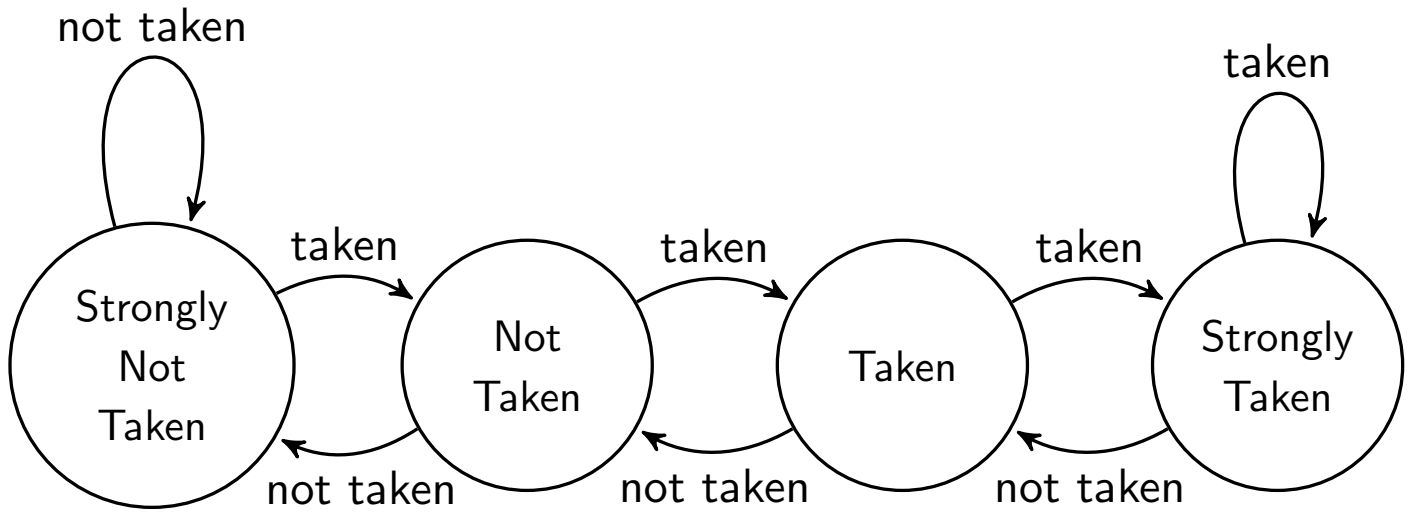
`https://en.wikichip.org/wiki/skylake`



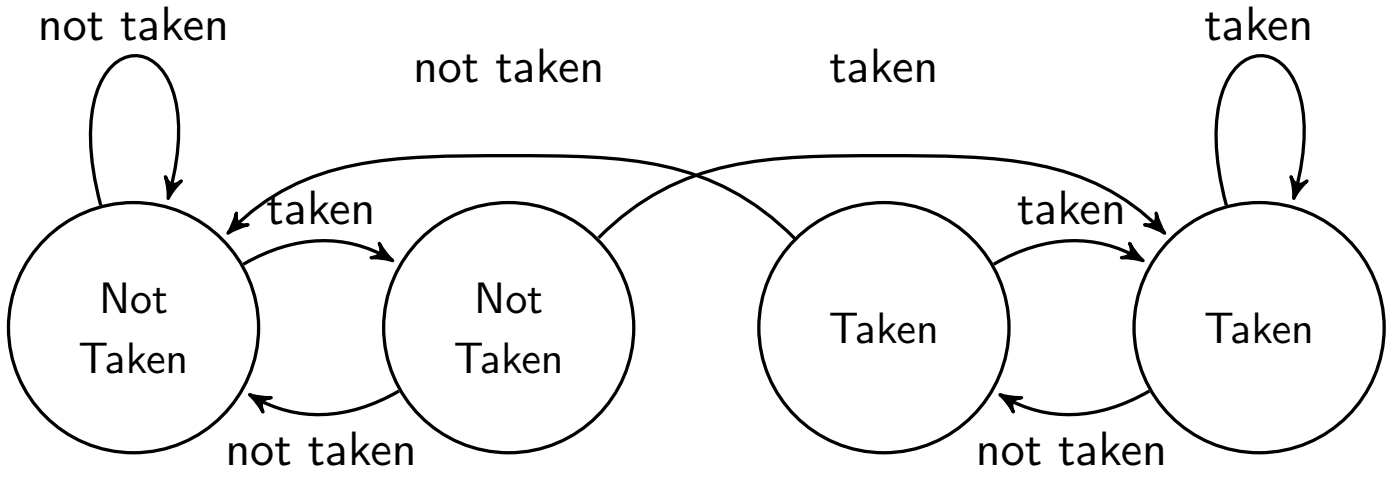
Prédicteur 1 bit



Prédicteur 2 bits saturé



Autre prédicteur 2 bits



Retour sur le calcul du min et du max

- Peut-on estimer le nombre d'erreurs de prédiction ?
- Ça dépend du prédicteur choisi. Prenons un 2 bits saturé.
- Pour l'algorithme naïf, avec un tableau d'entiers aléatoires, on fait de l'ordre de $log(n)$ erreurs de prédiction.
- Avec l'autre algorithme, c'est similaire pour les tests imbriqués... par contre le premier test crée une erreur de prédiction environ une fois sur deux, c'est à dire environ $n/2$ erreurs de prédiction.

(fichier **demo-C5-minmax-PAPI.c**)

- Que se passerait-il si le tableau était trié ? Faire le test.
- Pour aller plus loin : que se passe-t-il quand on utilise les optimisations de gcc ?

Retour sur le calcul du min et du max

- Peut-on estimer le nombre d'erreurs de prédiction ?
- Ça dépend du prédicteur choisi. Prenons un 2 bits saturé.
- Pour l'algorithme naïf, avec un tableau d'entiers aléatoires, on fait de l'ordre de $log(n)$ erreurs de prédiction.
- Avec l'autre algorithme, c'est similaire pour les tests imbriqués... par contre le premier test crée une erreur de prédiction environ une fois sur deux, c'est à dire environ $n/2$ erreurs de prédiction.

(fichier **demo-C5-minmax-PAPI.c**)

- Que se passerait-il si le tableau était trié ? Faire le test.
- Pour aller plus loin : que se passe-t-il quand on utilise les optimisations de gcc ?

Retour sur le calcul du min et du max

- Peut-on estimer le nombre d'erreurs de prédiction ?
- Ça dépend du prédicteur choisi. Prenons un 2 bits saturé.
- Pour l'algorithme naïf, avec un tableau d'entiers aléatoires, on fait de l'ordre de $log(n)$ erreurs de prédiction.
- Avec l'autre algorithme, c'est similaire pour les tests imbriqués... par contre le premier test crée une erreur de prédiction environ une fois sur deux, c'est à dire environ $n/2$ erreurs de prédiction.

  (fichier **demo-C5-minmax-PAPI.c**)

- Que se passerait-il si le tableau était trié ? Faire le test.
- Pour aller plus loin : que se passe-t-il quand on utilise les optimisations de gcc ?

Retour sur le calcul du min et du max

- Peut-on estimer le nombre d'erreurs de prédiction ?
- Ça dépend du prédicteur choisi. Prenons un 2 bits saturé.
- Pour l'algorithme naïf, avec un tableau d'entiers aléatoires, on fait de l'ordre de $log(n)$ erreurs de prédiction.
- Avec l'autre algorithme, c'est similaire pour les tests imbriqués... par contre le premier test crée une erreur de prédiction environ une fois sur deux, c'est à dire environ $n/2$ erreurs de prédiction.

(fichier **demo-C5-minmax-PAPI.c**)

- Que se passerait-il si le tableau était trié ? Faire le test.
- Pour aller plus loin : que se passe-t-il quand on utilise les optimisations de gcc ?

Retour sur le calcul du min et du max

- Peut-on estimer le nombre d'erreurs de prédiction ?
- Ça dépend du prédicteur choisi. Prenons un 2 bits saturé.
- Pour l'algorithme naïf, avec un tableau d'entiers aléatoires, on fait de l'ordre de $log(n)$ erreurs de prédiction.
- Avec l'autre algorithme, c'est similaire pour les tests imbriqués... par contre le premier test crée une erreur de prédiction environ une fois sur deux, c'est à dire environ $n/2$ erreurs de prédiction.

  (fichier **demo-C5-minmax-PAPI.c**)

- Que se passerait-il si le tableau était trié ? Faire le test.
- Pour aller plus loin : que se passe-t-il quand on utilise les optimisations de gcc ?

Retour sur le calcul du min et du max

- Peut-on estimer le nombre d'erreurs de prédiction ?
- Ça dépend du prédicteur choisi. Prenons un 2 bits saturé.
- Pour l'algorithme naïf, avec un tableau d'entiers aléatoires, on fait de l'ordre de $log(n)$ erreurs de prédiction.
- Avec l'autre algorithme, c'est similaire pour les tests imbriqués... par contre le premier test crée une erreur de prédiction environ une fois sur deux, c'est à dire environ $n/2$ erreurs de prédiction.

  (fichier **demo-C5-minmax-PAPI.c**)

- Que se passerait-il si le tableau était trié ? Faire le test.
- Pour aller plus loin : que se passe-t-il quand on utilise les optimisations de gcc ?

Autres modèles de predicteurs ?

T est un tableau de booléens (Faux si 0, Vrai sinon) de taille n. Chaque valeur a une probabilité $\frac{1}{2}$ d'être 0 et $\frac{1}{2}$ d'être autre chose.

```
1  ...
2  for(int i = 0; i < n-1; i++){
3    if (T[i] == 0)
4      ...
5    if (T[i+1] == 0)
6      ...
7    if (T[i] == T[i+1])
8      ...
9  }
```

- Que se passe-t-il avec un prédicteur local ?
- Peut-on faire mieux ?

Autres modèles de predicteurs ?

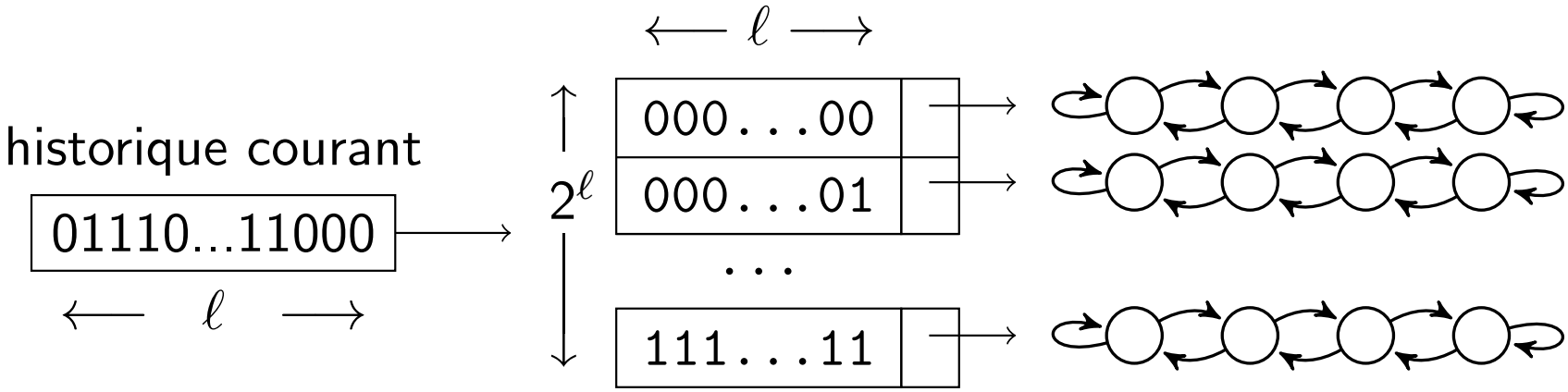
T est un tableau de booléens (Faux si 0, Vrai sinon) de taille n. Chaque valeur a une probabilité $\frac{1}{2}$ d'être 0 et $\frac{1}{2}$ d'être autre chose.

```
1  ...
2  for(int i = 0; i < n-1; i++){
3    if (T[i] == 0)
4      ...
5    if (T[i+1] == 0)
6      ...
7    if (T[i] == T[i+1])
8      ...
9  }
```

- Que se passe-t-il avec un prédicteur local ?
- Peut-on faire mieux ?
  - ▷ historique globale, prédiction de boucles, détection de corrélations, ...

Exemple : prédicteur avec table d'historique globale

- L'historique de taille ℓ represente les résultats des ℓ derniers branchements du programme.
- Pour un historique donné, seul le prédicteur associé donne la prédiction et est mis à jour.
- L'historique courant peut être unique et éventuelleemnt combiné avec le numéro de la branche. Il peut également y avoir un historique courant pour chaque branche (ou presque).



Et, au fait, ...

Pourquoi `REP RET` ? ?

- Pour les processeurs K8 d'AMD (1ère implém. 64 bits de x86).
- Grosso modo, on peut associer un prédicteur de branchement à chaque portion de code machine de 2 octets.
- `RET` est une instruction de branchement ; elle a donc un prédicteur.
- Mais son code machine tient sur 1 octet. Elle risque donc de "partager" le prédicteur de branchement de l'instruction qui la précède (si c'est un saut), ce qui fausse la prédiction.
- Solution : rajouter le complément d'instruction `REP` (sans effet dans ce cas), qui allonge le code machine pour que ça n'arrive pas.
- Remarque : les développeurs de gcc on donc estimé que la pénalité engendrée par l'erreur de prédiction était suffisamment importante pour justifier d'implémenter cette astuce.

`http://repzret.org/p/repzret/`