Principe du pipeline par l'exemple

Sequential

Parallel

Pipelined

Un autre exemple

Restaurant Universitaire On passe, dans l'ordre devant 4 éléments

- Un présentoir pour les entrées
- Un présentoir pour les desserts
- Un présentoir pour les plats de résistance
- Une caisse

2 modes d'utilisation

1. Une seule personne à la fois dans toute la chaîne de service
   - Quand elle a passé toute la chaîne et est sortie, une autre personne entre se servir
2. Plusieurs personnes à la fois, en décalé
   - Une personne à chaque présentoir/élément
   - Une personne passe à l'élément suivant quand il est libre et qu'elle en a fini avec son élément courant

2 modes d'utilisation

1. Une seule personne à la fois dans toute la chaîne de service
   - Quand elle a passé toute la chaîne et est sortie, une autre personne entre se servir
2. Plusieurs personnes à la fois, en décalé
   - Une personne à chaque présentoir/élément
   - Une personne passe à l'élément suivant quand il est libre et qu'elle en a fini avec son élément courant

2 modes d'utilisation

1. Une seule personne à la fois dans toute la chaîne de service
   - Quand elle a passé toute la chaîne et est sortie, une autre personne entre se servir
2. Plusieurs personnes à la fois, en décalé
   - Une personne à chaque présentoir/élément
   - Une personne passe à l'élément suivant quand il est libre et qu'elle en a fini avec son élément courant

2 modes d'utilisation

1. Une seule personne à la fois dans toute la chaîne de service
   - Quand elle a passé toute la chaîne et est sortie, une autre personne entre se servir
2. Plusieurs personnes à la fois, en décalé
   - Une personne à chaque présentoir/élément
   - Une personne passe à l'élément suivant quand il est libre et qu'elle en a fini avec son élément courant

2 modes d'utilisation

1. Une seule personne à la fois dans toute la chaîne de service
   - Quand elle a passé toute la chaîne et est sortie, une autre personne entre se servir
2. Plusieurs personnes à la fois, en décalé
   - Une personne à chaque présentoir/élément
   - Une personne passe à l'élément suivant quand il est libre et qu'elle en a fini avec son élément courant

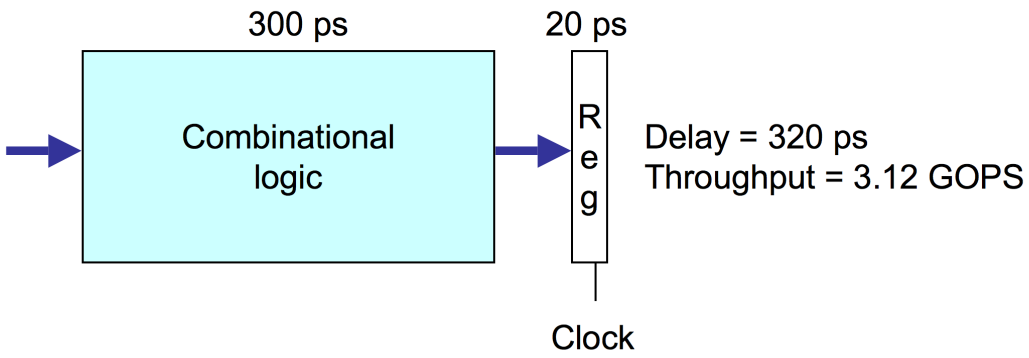
Intérêts du deuxième mode

- Plusieurs personnes se servent en même temps
- Gain de temps : plus de personnes passent pendant une même durée
- Meilleure gestion des éléments : toujours utilisés

Inconvénients du deuxième mode

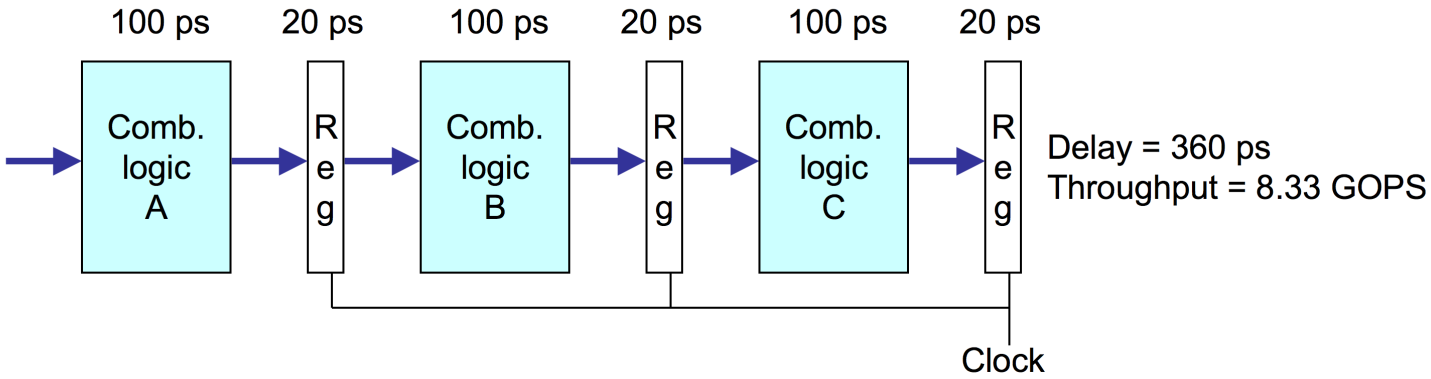
- Plus difficile de faire « demi-tour » dans la chaîne d'éléments
- Nécessite des synchronisations supplémentaires et des éléments dont le parcours prend un temps proche pour une bonne optimisation

Exemple calculatoire



- Le calcul nécessite 300 picosecondes
- On rajoute 20 picosecondes pour sauver le résultat dans le registre

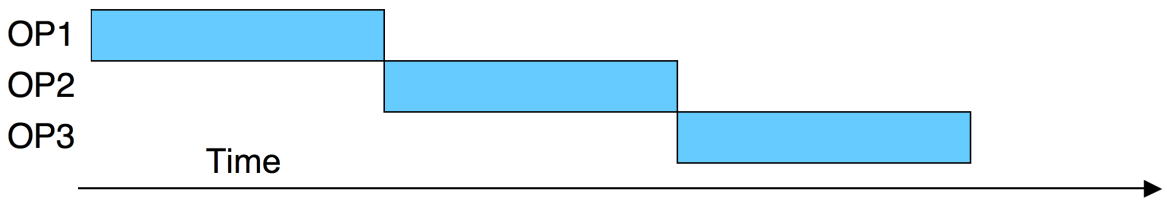
Un pipeline à 3 étages



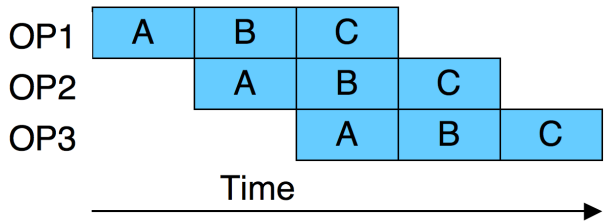
- On divise la séquence combinatoire en 3 blocs de 100 ps
- On commence une nouvelle opération dès que la précédente a terminée l'étape A. On commence une opération toutes les 120 ps.
- La latence totale augmente : 360 ps registre

Diagramme du Pipeline

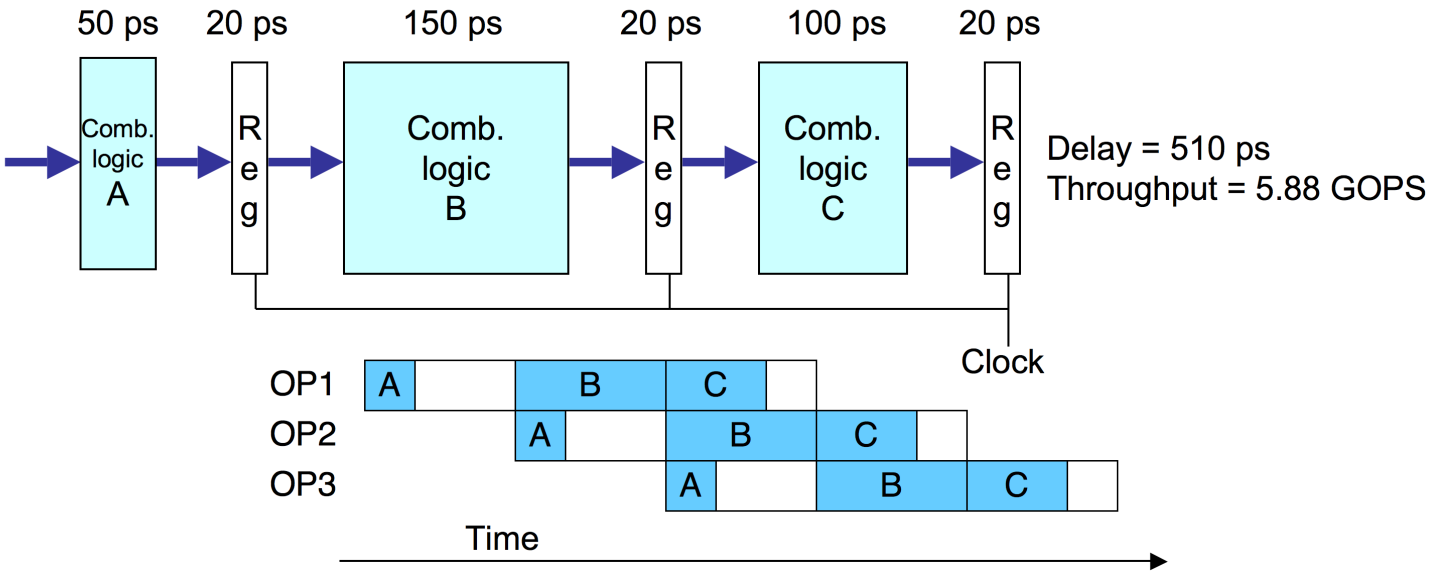
Sans Pipeline : une nouvelle opération ne peut pas commencer tant que les précédentes n'est pas terminée.



Pipeline à 3 étages : simultanément jusqu'à 3 opérations dans le processus



Limitations : délais non uniformes



- La traversée est limitée par l'étage
- Modifier la partition pour équilibrer les étages

Principes

Dans un processeur, utilisation d'un pipeline pour exécution d'une opération

- Une opération comporte plusieurs sous-opérations
  - Pipeline pour exécution de ces sous-opérations
  - Une sous-opération utilise une sous-unité du processeur qui n'est pas utilisée par d'autres sous-opérations (si possible...)
- Exemple de pipeline simple (fictif)
  - LE : Lecture de l'instruction en mémoire
  - DE : Décodage de l'instruction
  - CH : Chargement des registres sources dans l'unité de calcul
  - EX : Exécution du calcul
  - ENR : Enregistrement du résultat dans le registre destination

Détail des étapes

En reprenant le chemin de donnée précédent, en modèle chargement-rangement :

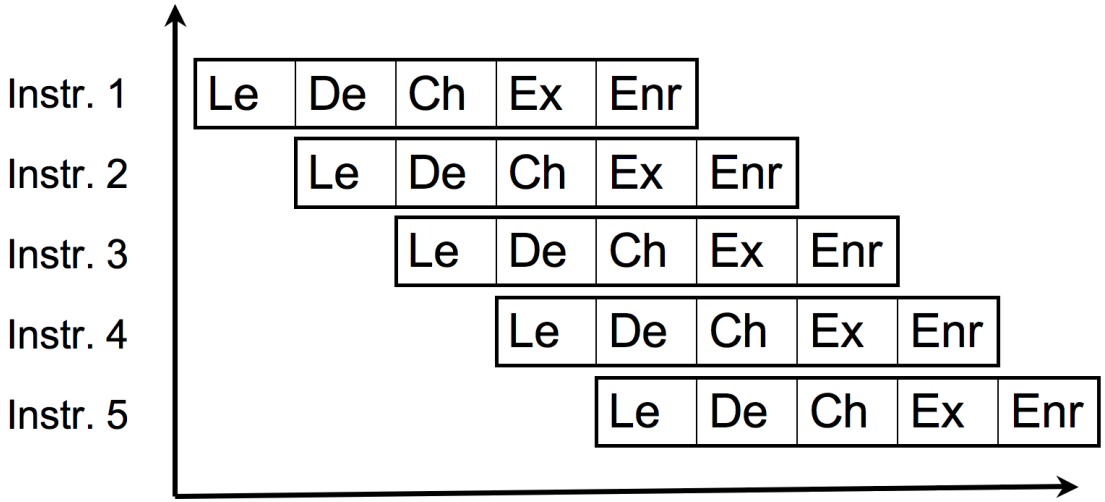
- LE : lecture instruction
  - Accès en mémoire pour lire le mot mémoire placé à l'adresse contenue dans CO et correspondant à la prochaine instruction
  - Résultat placé dans RM
- DE : décodage instruction
  Copie de RM dans RI puis décodage de l'opération
- CH : chargement des registres
  - Pour un calcul : Chargement des registres A et B à partir des registres du banc
  - Pour un accès en mémoire : Chargement de RA à partir d'un registre du banc
  - Dans le cas d'une écriture en mémoire, chargement en plus de RM à partir d'un registre du banc

Détail des étapes

- EX : exécution de l'instruction
  - Pour un calcul: l'UAL effectue le calcul
  - Résultat disponible dans C
  - Pour un accès en mémoire : Accès à la mémoire via les bus mémoire, RA et/ou RM
  - Si opération de lecture, RM contient le mot lu à l'adresse contenue dans RA
- ENR : enregistrement du résultat
  - Pour un calcul : Copie du registre C de l'UAL dans le registre destination du banc
  - Pour un accès en mémoire
    - Ecriture : rien à faire
    - Lecture : copie de RM dans le registre destination du banc

Pipeline

Exécutions décalées de plusieurs instructions en même temps



- Gain important en utilisant le pipeline :
  - Sans : exécution séquentielle de 2 instructions en 10 cycles
  - Avec : exécution parallèle de 5 instructions en 9 cycles
  - Gain théorique car nombreux problèmes en pratique ...
- Pour optimisation : temps de passage dans chaque étape identique (ou très proche)

Profondeur

En pratique actuellement : autour de 12/15 étages
Exemples de profondeur de pipeline (nombre d'étages) pour processeurs actuels

- Processeurs Intel
  - Core 2 Duo et Mobile : 14 et 12
  - P4 Prescott : 30
  - P4 (avant architecture Prescott) : 20
  - Génération précédente : Intel P3 : 10
- Processeurs AMD
  - Athlon 64 : 12
  - Génération précédente : AMD Athlon XP : 10
- Processeurs de type RISC
  - Sun UltraSparc IV : 14
  - IBM Power PC 970 : 16

Profondeur

Intérêts d'avoir un pipeline plus profond

- Plus d'instructions en cours d'exécution simultanée
- Permet une montée en fréquence du processeur
- Limite de la montée en fréquence
- Temps de propagation des signaux
- A travers une unité et entre les unités du CPU
- Plus d'unités plus petites = temps de propagation plus courts entre les unités
- On peut raccourcir le temps de réalisation d'un cycle
- Et donc augmenter la fréquence du processeur
- Mais un pipeline profond pose plus de problèmes qu'un pipeline court
- Avec les aléas de contrôles principalement

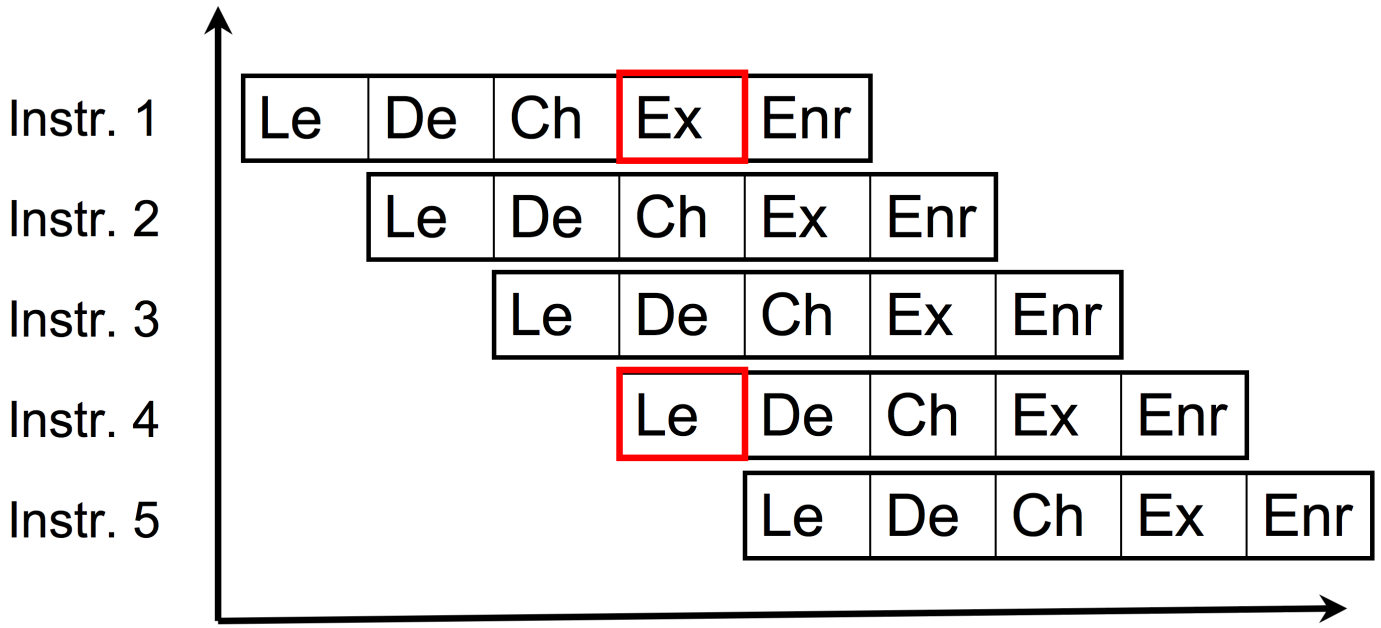
Aléas

- Problèmes rencontrés lors de l'exécution d'instructions par le pipeline
- 3 familles d'aléas
  - Aléas structurels :
    Des sous-unités du CPU doivent être utilisées simultanément par plusieurs étages du pipeline
  - Aléa de données :
    Une instruction de calcul en cours d'exécution dépend d'une valeur non encore calculée
  - Aléas de contrôle
    L'instruction suivante dépend du résultat d'une instruction pas encore connu (test)

Aléas structurels

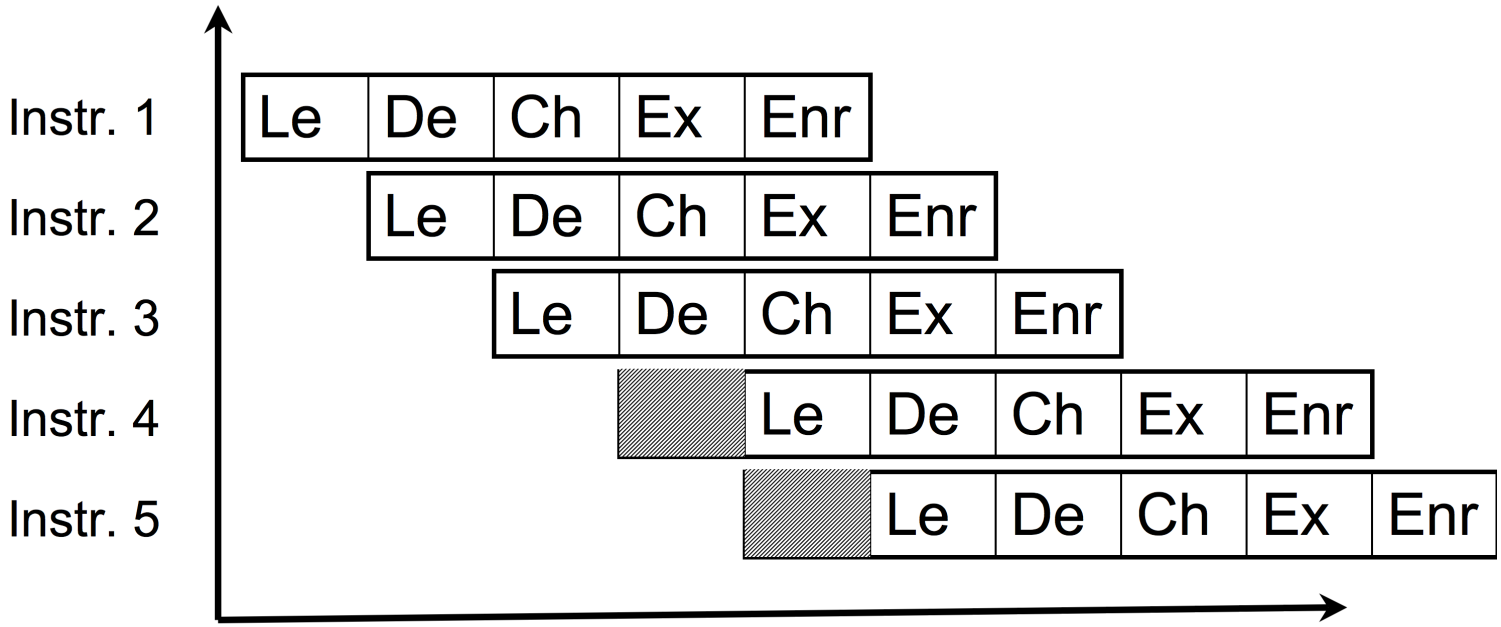
- Exemple d'aléa structurel, pour notre pipeline simple
  - Accès à la mémoire dans les étapes
  - LE : lecture de l'instruction suivante en mémoire
  - EX dans le cas d'une opération de lecture/écriture en mémoire
  - Utilise une même sous-unité (accès mémoire) du processeur
- Solutions
  - Attendre pour une instruction que l'unité soit disponible
  - Peu efficace
  - Dupliquer dans le processeur ces sous-unités
  - Accès mémoire : intérêt de découper le cache L1 en deux parties
  - Partie « données » avec accès via RM et RA
  - Partie « instructions » avec accès via CO et RI
  - Peut alors faire un EX d'accès mémoire et un LE en même temps : 2 accès mémoires en parallèle sur les 2 parties différentes du cache L1

Exemple d'Aléas structurels



Ex de l'instruction 1 et Le de l'instruction 4: accède au même valeur en mémoire.

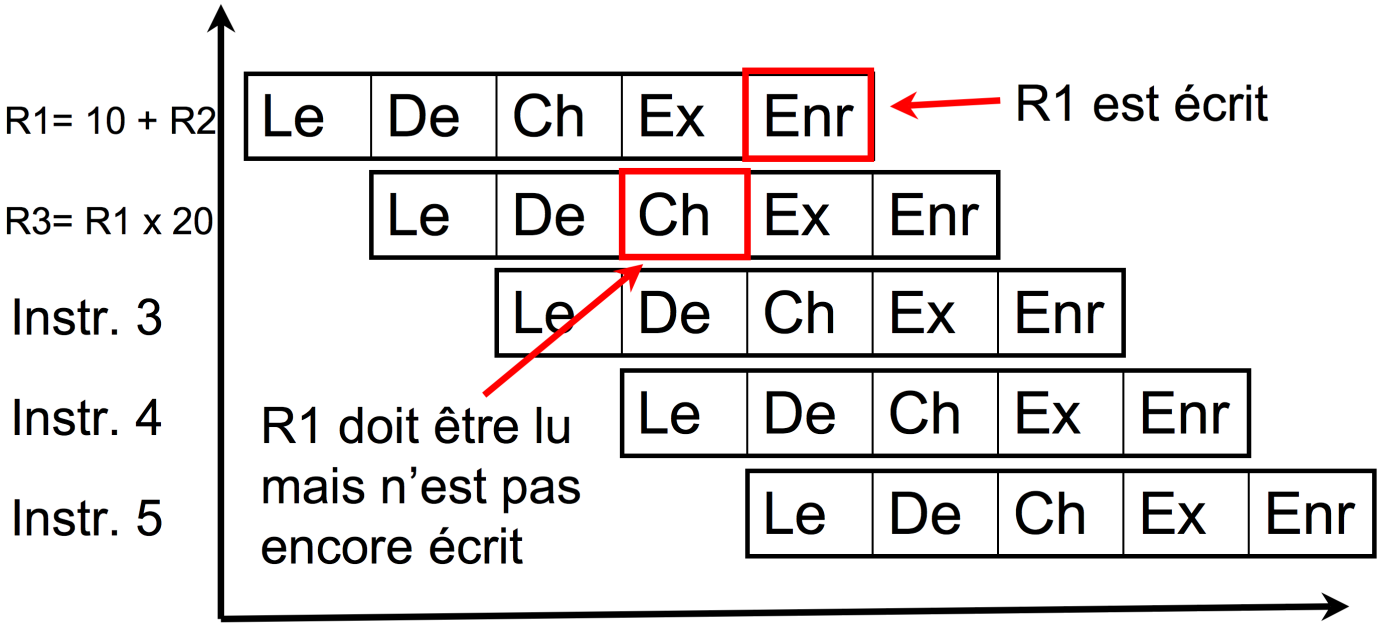
Exemple d'Aléas structurels



Solution par attente : décalage de toutes les instructions suivantes

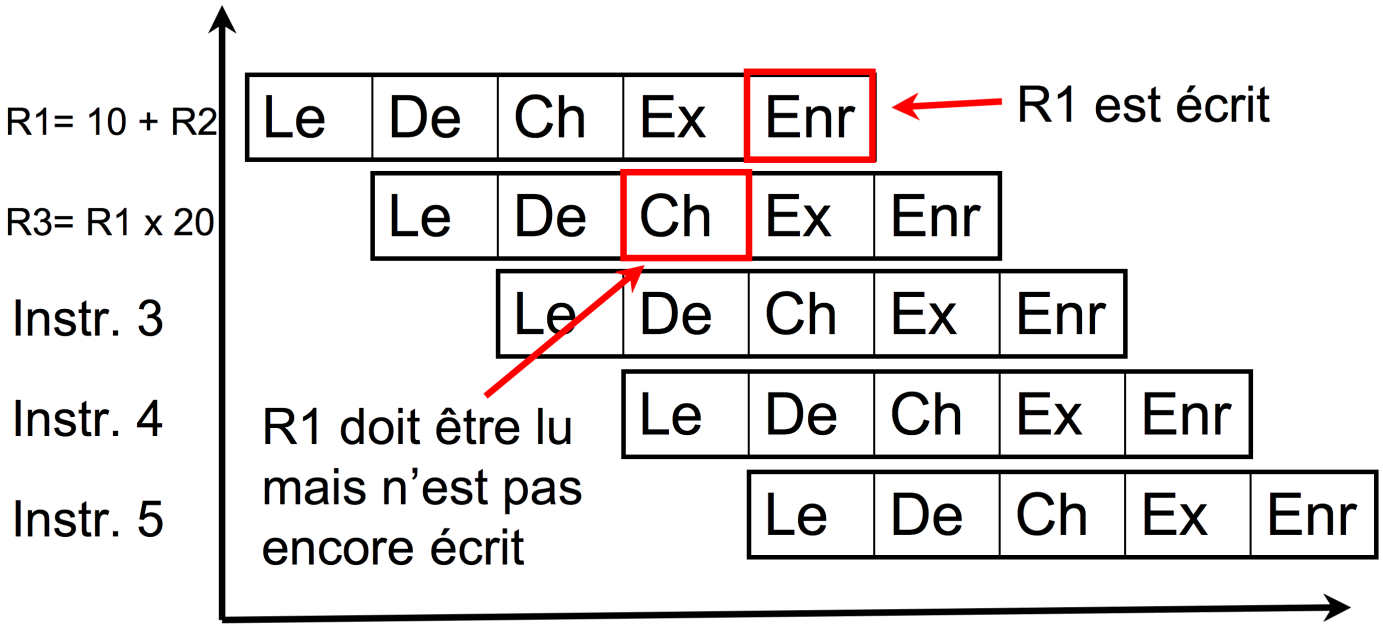
Aléas de données

- `R1 = 10 + R2`
  `R3 = R1 x 20` (R1, R2 et R3 sont des registres)
- Problème : Le calcul de R3 ne peut se faire que quand R1 est connu



Aléas de données

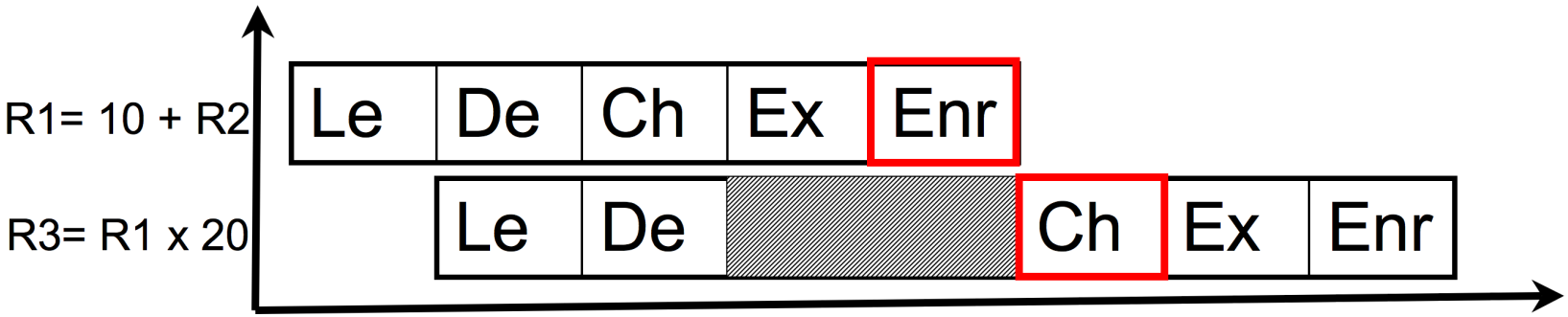
- `R1 = 10 + R2`
  `R3 = R1 x 20` (R1, R2 et R3 sont des registres)
- Problème : Le calcul de R3 ne peut se faire que quand R1 est connu



Solutions

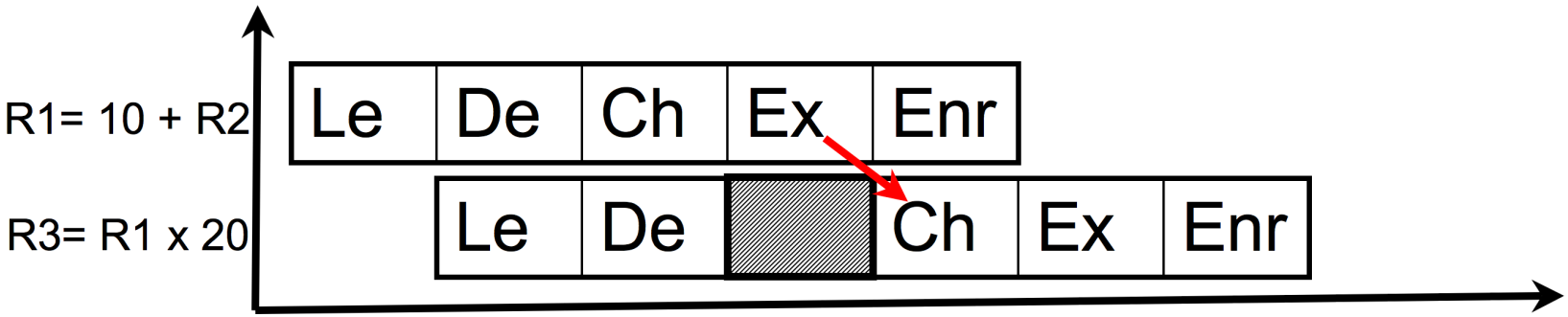
- Arrêter l'exécution du calcul de R3 tant que R1 n'est pas connu : peu efficace
- Changer l'ordre d'exécution des opérations pour éviter ou réduire le problème
- Court-circuiter au plus tôt le pipeline quand la valeur de R1 est connue
- Le résultat du dernier calcul est dans le registre C de l'UAL
- On peut le réinjecter au cycle suivant dans le registre A ou B de l'UAL

Suspension du pipeline



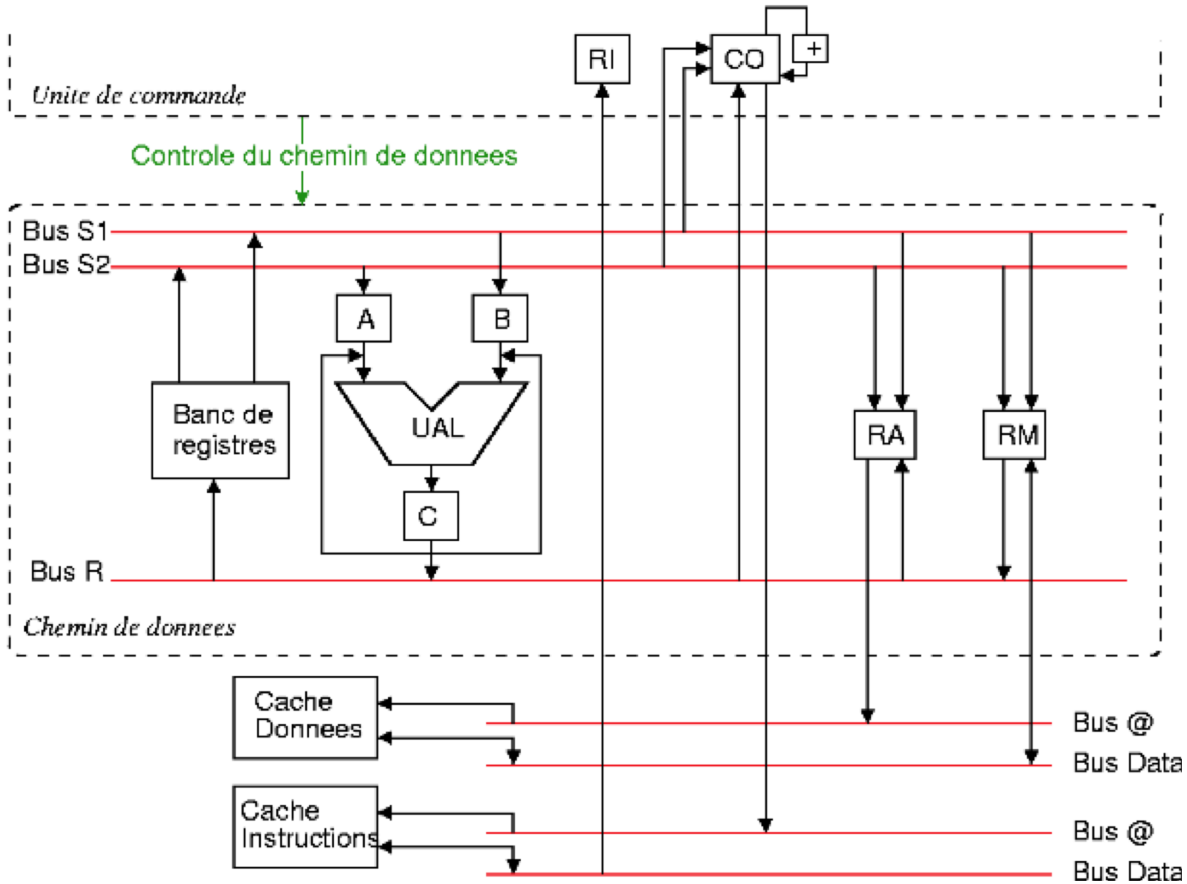
La deuxième instruction est suspendue tant que R1 n'est pas écrit

Court-circuit du pipeline



Après l'étape Ex de la 1'ere instruction, on connaît l avaleur de R1 : on la réinjecte directement dans l'UAL sans attendre son écriture au niveau du banc de registre.

Nouveau chemin de données



Nouveau chemin de données avec court-circuit du pipeline et accès mémoire via 2 parties du cache

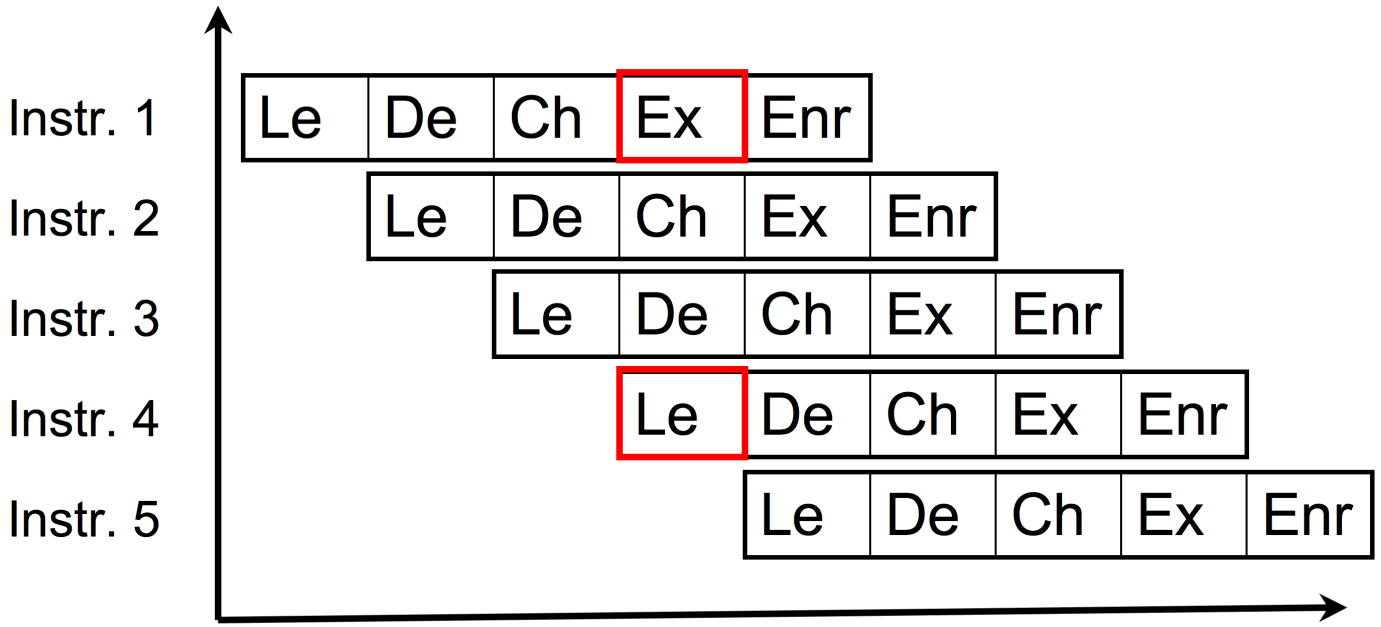
Réordonnancement

```
R1 = 10 + R2
R3 = R1 x 20
R4 = 2 x R5
R6 = 10 + R5
```

- Dépendance de données entre les 2 premières instructions : aléa de données dans le pipeline
- Réordonnancement pour éviter cet aléa
- On place les 2 autres instructions entre ces 2 instructions:

```
R1 = 10 + R2
R4 = 2 x R5
R6 = 10 + R5
R3 =R1 x 20
```

Réordonnancement



- Grâce à ce réordonnancement : pipeline non suspendu pour aucune des 4 instructions
- Utilisation optimale du pipeline
- Deux types de réordonnancement :
  - Logiciel (compilateur)
  - Matériel (fait par le processeur)

Aléas de contrôle

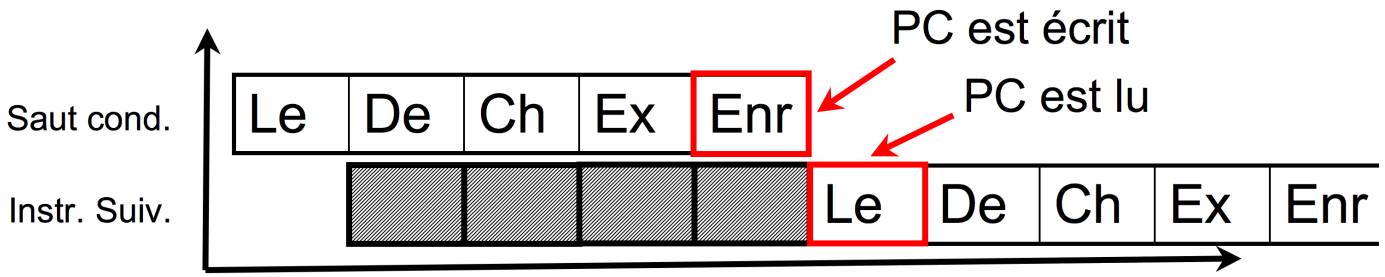
```
 if (R1 > 30)
then R3 = 10 + R1
else R3 = 20 + R1
```

- Fonctionnement du saut conditionnel
  - En fonction du résultat du test, le contenu de PC est modifié avec l'adresse de la prochaine instruction
  - Phase EX : exécution de la comparaison par l'UAL
  - Phase ENR : écriture de PC en fonction du résultat du test
- Problème
  - Doit connaître la valeur du test de valeur de R1 pour savoir quelle est l'instruction suivante à exécuter

Solutions

Attendre que le bon PC soit connu : peu efficace

- Réordonnancer le code : pas toujours suffisant et possible
- Prédire quelle sera la valeur de R1 et commencer le calcul suivant selon cette prédiction
- Solution avec attente :



- Doit attendre le ENR précédent avant de faire le LE : on passe en exécution purement séquentielle !

Prédiction de branchement

Aléas de contrôle : prédictions de branchement pour en limiter les conséquences

- Indispensable pour efficacité du pipeline
- A l'aide de tables statistiques dynamiques
- Prédit le résultat d'un test
- On commence ensuite l'instruction suivante prédite
- Problème si prédiction erronée
- On a commencé des calculs inutiles
- Vidage du pipeline pour reprendre dans un état correct
- Trés couteux en temps
- Très pénalisant pour des pipelines profonds

Principes de prédiction

Mémoriser les adresses des branches du programme et regarder celles qui sont souvent atteintes

- Exemple:

```
1 R0 = R2 - 3
2 if R1 = 0 jump suite
3 R3 = 2 x R1
4 R4 = R3 + R1
suite:
5 R3 = 0
```

- Deux branches : adresses 3 et 5
- Prédiction : lors du saut conditionnel à l'adresse 2, on prendra la branche la plus souvent atteinte

Prédiction de branchement

- Deux éléments pour fonctionnement
  1. Tampon des branches cibles (BTB : Branch Target Buffer) : Contient les adresses des branches du programme
  2. Table de l'historique des branchements (BHT : Branch History Table) : Mémoriser l'historique des choix de branchements faits précédemment pour faire des prédictions
- Fonctionnement dépend de l'algorithme utilisé
- Exemple basique : 2 bits associés à chaque branche
  - 00 : branchement jamais pris jusqu'à présent
  - 01 : branchement parfois pris jusqu'à présent
  - 10 : branchement souvent pris jusqu'à présent
  - 11 : branchement toujours pris jusqu'à présent
- Mise à jour des BTB et BHT pendant l'exécution du programme

Amélioration de l'efficacité

Pour plus d'efficacité des prédictions

- Augmenter la taille du BTB et du BTH
  - Pour pouvoir gérer plus de branches (BTB)
  - Si BTB trop petit, il ne stocke pas toutes les branches : pas de prédictions possibles pour toutes les branches
  - Pour avoir un historique plus long et précis (BHT)
  - Pb : temps d'accès plus long car tables plus grandes
- Augmenter la qualité de la prédiction avec des algorithmes plus efficaces
- Pb : prend un temps plus long qu'avec des algorithmes plus simples
- Dans les 2 cas : augmentation du temps de la prédiction
  - Contraire au besoin de connaître au plus tôt la prédiction
  - Limite la montée en fréquence du processeur
- Efficacité des prédictions
- En moyenne, autour de 90% des prédictions sont correctes

Conclusion

Influence de la profondeur du pipeline

- Avantage d'un pipeline long
  - Plus d'instructions en exécution parallèle
  - Montée en fréquence du processeur facilitée
  - Donc gain en nombre d'instructions exécutées en un temps donné
- Inconvénient d'un pipeline long
  - Une erreur de prédiction est plus coûteuse
  - Plus d'opérations en cours d'exécution à annuler
- Solution globale
  - Trouver le bon compromis entre gain d'un coté et perte de l'autre
  - Améliorer les algorithmes et unités de prédiction de branchement