Ecole thématique
Conception d'architectures de systèmes informatiques dédiés
à des applications spécifiques de type « enfoui »
Seix – 20-23 novembre 2000

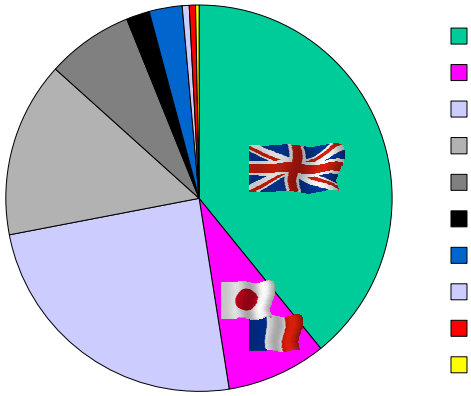
Architecture des processeurs superscalaires

Pascal Sainrat

Introduction
Les applications des systèmes enfouis

- **Hier**
  - **Contrôle de processus**
  - **Temps réel**
  - **Traitement de signal**
- **Aujourd'hui**
  - **Idem**
  - **Appareils portables**
    - Téléphones
    - Liaisons internet
    - Applications ludiques (lecteur MP3, mini-DVD, …)
  - **Infrastructure réseau**
- **Demain**
  - **?**

Parts de marché



Introduction
Les besoins des systèmes enfouis

- **Hier**
  - **Besoins en calcul régulier**
  - **Problématique temps réel**
    - Savoir calculer le WCET
- **Aujourd'hui**
  - ↗ **Besoins en performance**
  - ↗ **Besoins en calcul régulier**
  - ↗ **Interface réseau ou télécom**
  - ↗ **Problématique temps réel**
- **Demain**
  - ↗ **?**

Introduction
Les architectures

Processeurs haute performance

- Avant-hier
  - CISC
- Hier
  - RISC scalaire
  - Superscalaire exécution ordonnée
- Aujourd'hui
  - Superscalaire à exécution non ordonnée
  - VLIW ou EPIC
- Demain
  - Multithread
  - SMP on-chip

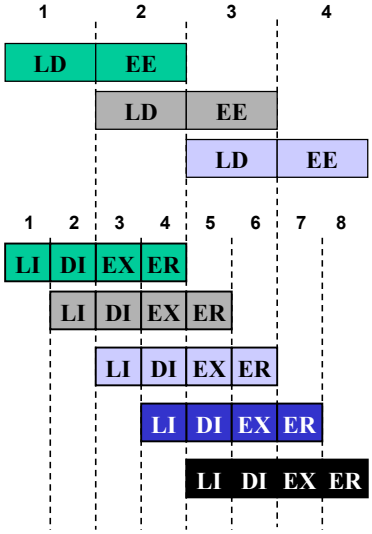
Systèmes enfouis

- Avant-hier
  - CISC
  - VLIW (DSPs)
- Hier
  - CISC
  - VLIW (DSPs)
- Aujourd'hui
  - CISC
    - DragonBall
  - RISC & superscalaire
    - PowerPC, ARM
  - VLIW
    - DSPs, multimedia
- Demain
  - SMP ?
  - Multihread ?

Architecture des processeurs haute performance aujourd'hui

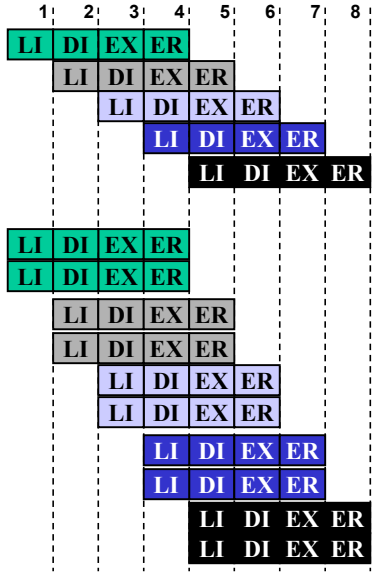
- Pipeline et dépendances
- Superscalaire
  - Exécution ordonnée
  - Exécution non ordonnée
- Exécution spéculative
  - Prédiction de branchement
  - Prédiction de dépendances, d'adresses, de valeur
- Cache de traces
- Multithreading
- Adéquation aux systèmes enfouis
  - Et la consommation dans tout cela ?
    - Crusoe

Exécution pipeline



- Forte tendance à augmenter la longueur du pipeline
- Performance * 2 ?
  - Dépendances de données
  - Dépendances de contrôle

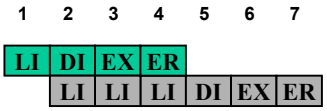
Exécution superscalaire



- Performance * 2
  - ↗ Dépendances de contrôle
  - ↗ Dépendances de données

Dépendances de données

```
LD    F0,0(R1)
ADD   F2,F0,F4
SUB   F2,F3,F2
ADD   F3,F3,#1
```



- **Dépendances vraies**
  - Lecture après écriture
- **Dépendances de sortie**
  - Ecriture après écriture
- **Anti-dépendances**
  - Ecriture après lecture

Solution :

Le compilateur ou le processeur doit éloigner les instructions ayant des dépendances vraies mais sans violer les autres dépendances (renommer au mieux)

Dépendances de contrôle

- Toute instruction qui suit un branchement dépend de celui-ci
- Compilateur
  - ☹ Réordonnancement à l'intérieur d'un bloc de base
  - ☹ Code de compensation pour exécution spéculative
- Matériel
  - ☺ Prédiction de branchement et exécution spéculative
  - ☺ Réordonnancement à l'intérieur de la fenêtre des instructions présentes dans le processeur

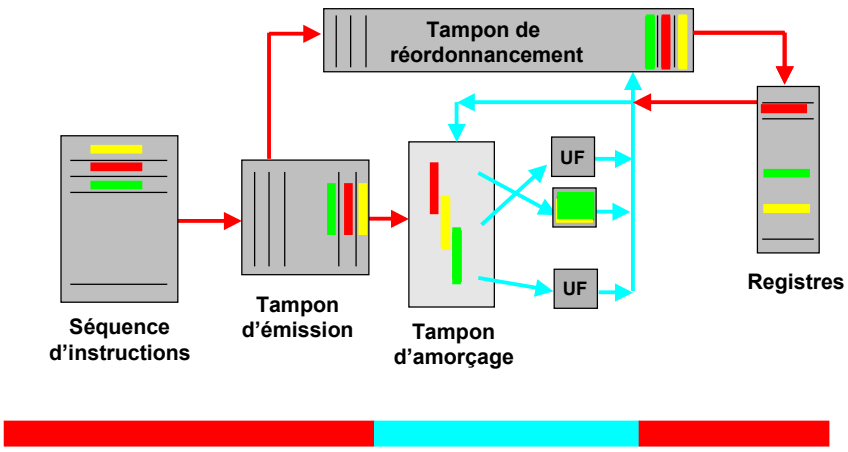
Plan

- Pipeline, superscalaire et dépendances
- **Processeur à exécution non ordonnée**
  - **Chargement des instructions**
    - prédiction de branchement
    - Cache de traces
  - **Renommage**
  - **Amorçage**
  - **Retrait**
  - **Accès à la mémoire**
  - **Réutilisation et prédiction de valeur**
- **VLIW et EPIC**

Exécution non ordonnée et spéculative

- **Combinaison de 3 idées :**
  - **Ordonnancement dynamique.**
    - Exécution data-flow
  - **Prédiction dynamique de branchement (prédiction au moment du chargement des instructions)**
  - **Exécution spéculative (avant que les dépendances de contrôle soient connues)**
- **Par rapport à la spéculation logicielle**
  - **Possibilité de lever les ambiguités mémoires dynamiquement**
  - **Prédiction de branchement matérielle nettement meilleure**
  - **Modèle d'exception précis même pour les instructions spéculées**
  - **Code indépendant de l'architecture et pas de code de compensation**
- **complexité**

Processeur à exécution non ordonnée
Schéma de principe

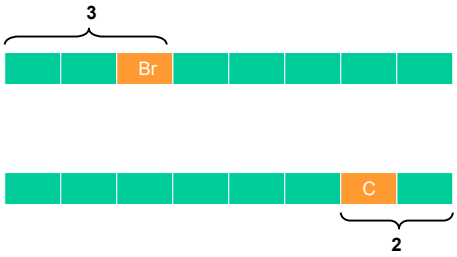


Plan

- Pipeline, superscalaire et dépendances
- **Processeur à exécution non ordonnée**
  - **Chargement des instructions**
    - prédiction de branchement
    - Cache de traces
  - **Renommage**
  - **Amorçage**
  - **Retrait**
  - **Accès à la mémoire**
  - **Réutilisation et prédiction de valeur**
- **VLIW et EPIC**

Chargement des instructions

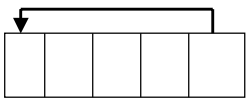
- Chargement des instructions depuis le cache d'instructions
- Doit satisfaire la bande passante requise
  - 3 à 4 instructions par cycle en moyenne



  - Charger deux blocs à la fois
  - Tampon d'émission pour les à-coups
- Besoin de savoir au cycle suivant la cible d'un branchement

Prédiction de branchement
Nécessité

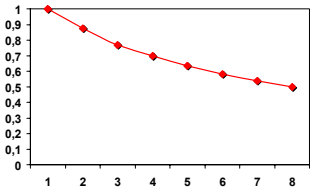
- En absence de prédiction, le processeur charge en séquence. Lorsque le branchement est exécuté (dans l'étage d'exécution), il redirige l'acquisition des instructions si le branchement est pris.
  - Les instructions acquises entre le branchement et le moment où le branchement est exécuté sont annulées (bulles).



Hyp: 1 saut toutes les 7 instructions

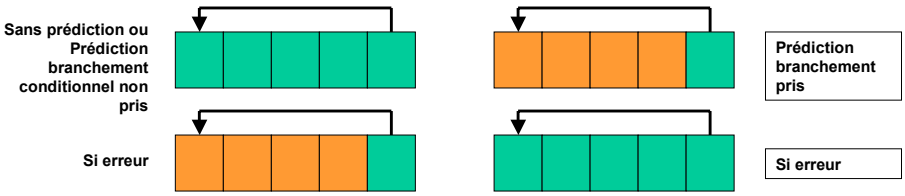
7 cycles pleins suivis
De N-1 cycles vides

$$débit = \frac{7}{7+(N-1)} inst/cycles$$

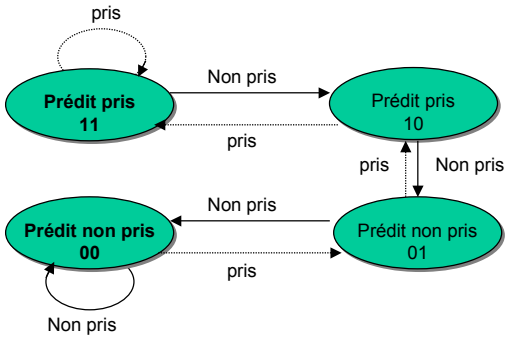


Prédiction de branchement
Principe

- Prédiction qui prend en compte le comportement passé des branchements
- Prédiction dès le premier cycle
  - Prédiction de la présence d'un branchement
  - Prédiction du type du branchement
  - Prédiction de l'adresse cible
- Vérification lors de l'exécution du branchement et annulation des instructions qui suivent le branchement si mauvaise prédiction



Prédiction de la direction
Compteur 2 bits



Itération	1	2	3	4	5	1	2	3
Comportement	P	P	P	P	NP	P	P	P
Compteur	10	11	11	11	11	10	11	11

Prédiction de la direction
Historique global

Exemple de programme

```
if  (aa=2)

      aa = 0;
if (bb=2)

      bb = 0;
if (aa != bb)

      { …}
```

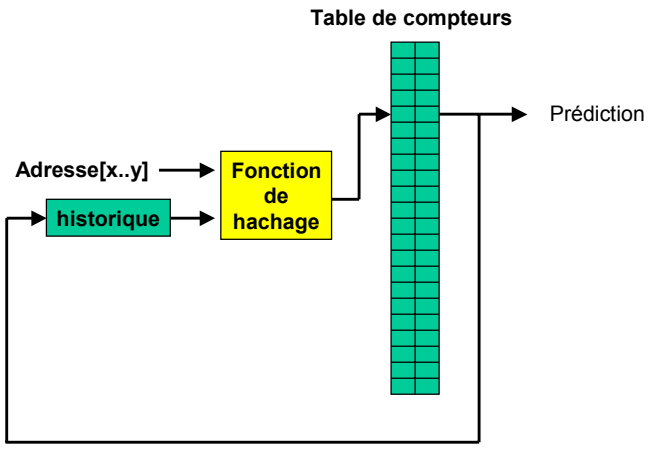
Traduction en assembleur
aa dans R1, bb dans R2

```
        SUBI R3, R1, #2
b1:     BNEZ R3, L1
        ADD R1, R0, R0
L1:     SUBI R3, R2, #2
b2:     BNEZ R3, L2
        ADD R2,R0,R0
L2:     SUBI R3, R1, R2
b3:     BNEZ R3, L3
```

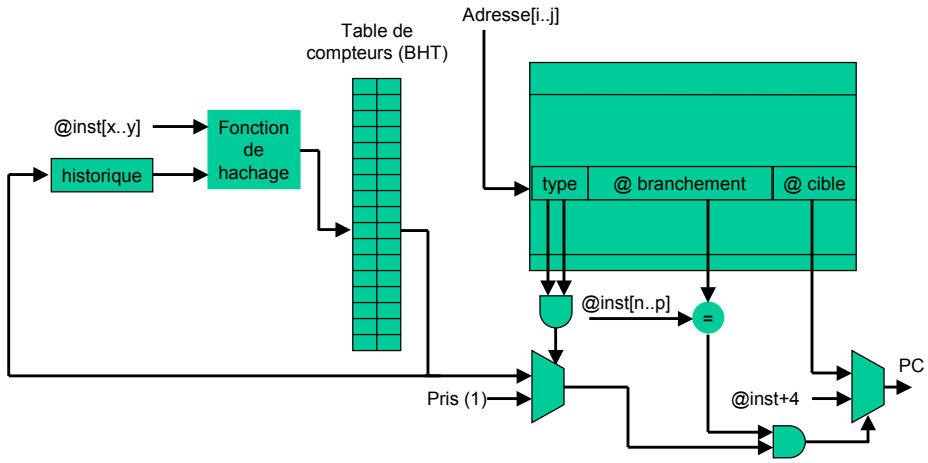
Si b1 et b2 ne sont pas pris, b3 le sera car aa = bb = 0. b3 est corrélé à b1 et b2.

➡**Prédicteurs à corrélation**

Prédiction de la direction
gshare

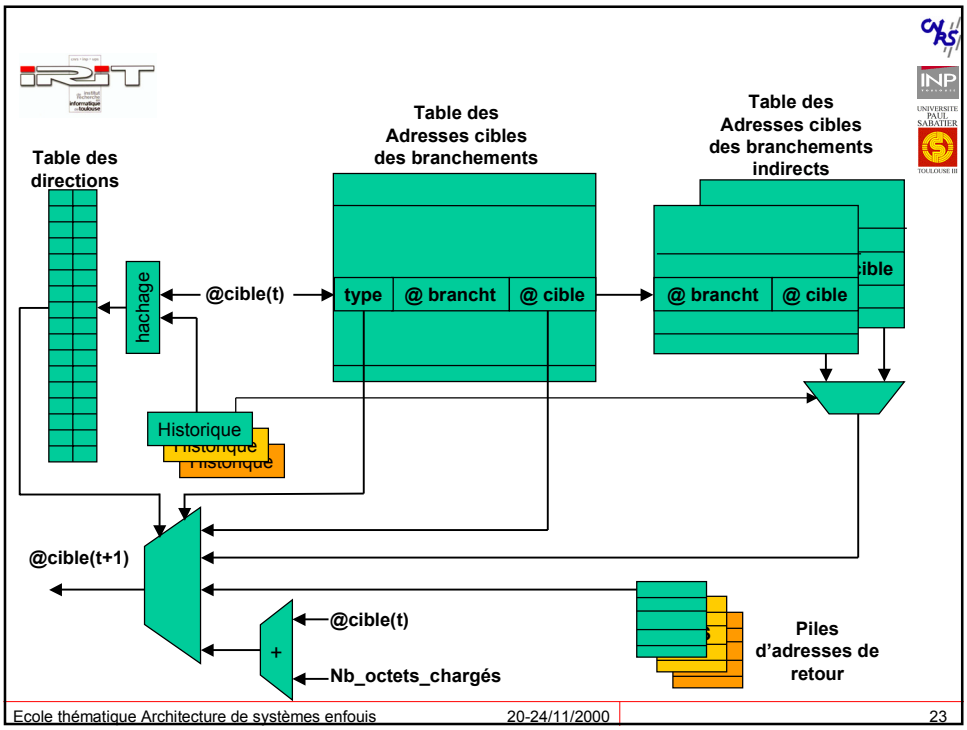


Prédiction de branchement
Schéma de base



Prédiction de branchement
Compléments

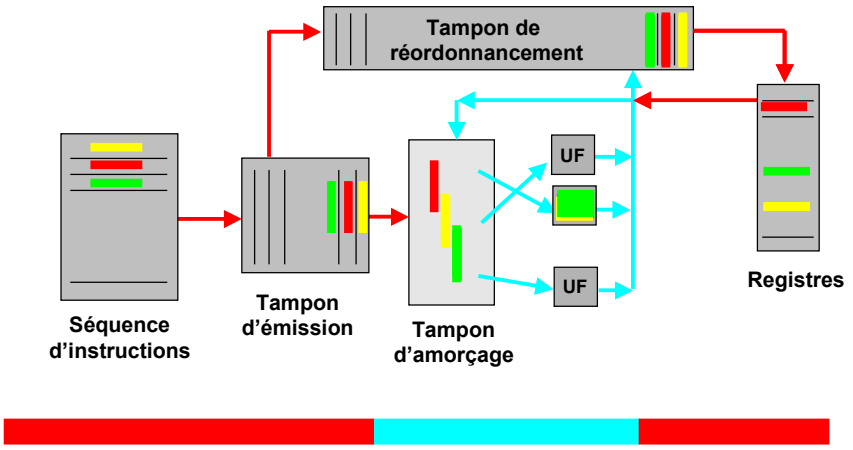
- **Branchements indirects**
  - **Cibles multiples ➡ table associative par ensembles**
- **Retours de sous-programmes**
  - **Cible = @ dernier appel ➡ pile d'adresses de retour**
- **Mise à jour spéculative de l'historique et de la pile**
  - **Sauvegardes**
- **Processeur superscalaire**
  - **Seule adresse connue = adresse de bloc**
  - **Si plusieurs branchements chargés, seul le premier branchement est prédit**
    - Prédire plusieurs branchements
  - **Besoin de l'adresse en séquence**



Instruction de prédiction de branchement

- Instruction de prédiction de l'adresse d'un branchement
- Mise à jour d'une petite table prioritaire dans le mécanisme de prédiction d'adresse
- Permet une économie sur la table des adresses cibles ?
  - Non sauf si le branchement prédit a un code spécial indiquant que la table des adresses cibles ne doit pas être mise à jour
- L'instruction de prédiction doit être suffisamment éloignée pour avoir le temps de mettre à jour la table avant que le branchement soit chargé

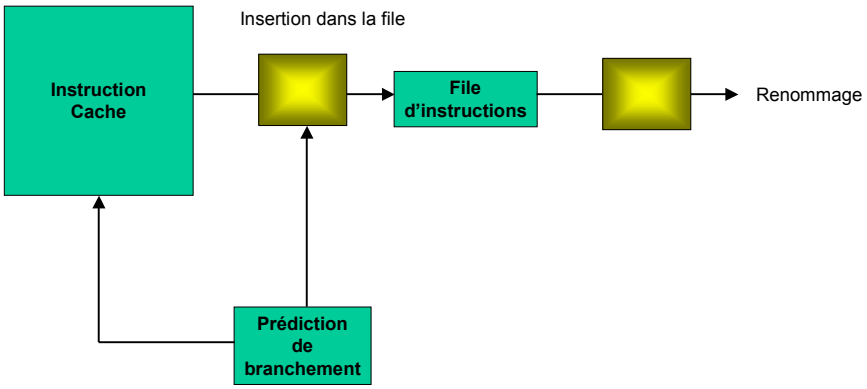
Processeur à exécution non ordonnée
Schéma de principe



Décodage

Adressage du cache
Chargement
Sélection
Insertion dans la file

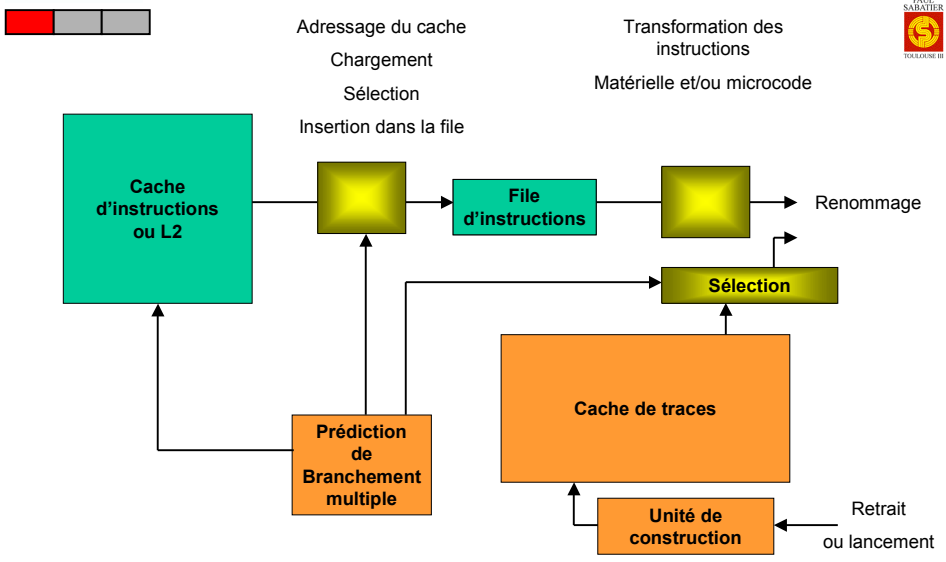
Transformation des instructions
Matérielle et/ou microcode



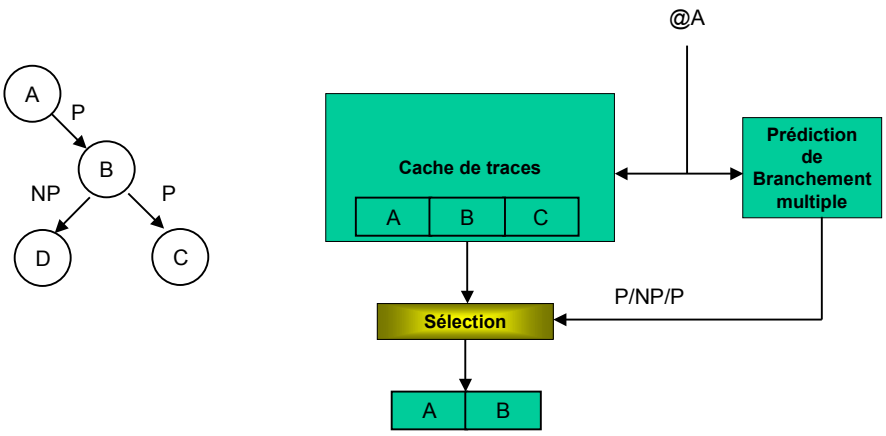
Augmenter le nombre d'instructions chargées

- Nombre d'instructions chargées limité par:
  - Taille d'une ligne du cache
  - Instructions de rupture du flot de contrôle
- Solutions:
  - Charger plusieurs lignes en même temps
    - Cache multi-bancs avec plusieurs ports de lecture ➡$$$
    - Cache de traces
  - Prédire plusieurs branchements simultanément

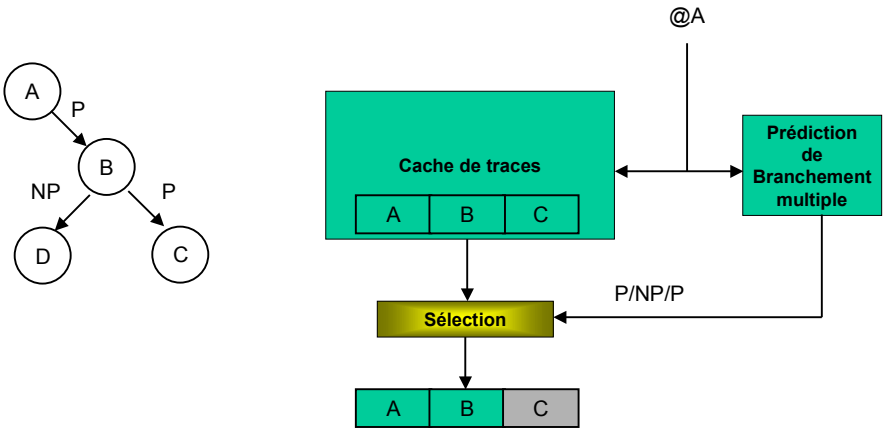
Cache de traces
Mécanisme



Cache de traces
Correspondance partielle



Cache de traces
Lancement inactif

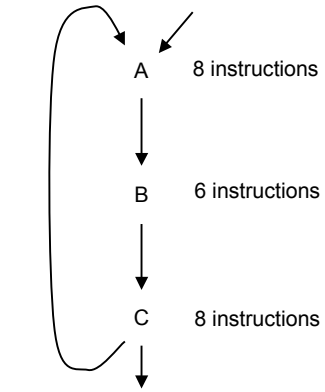


Cache de traces
Branchements biaisés

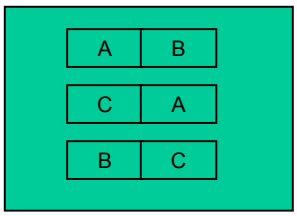
- Beaucoup de traces n'atteignent pas leur taille maximum parce qu'elles ont trop de branchements conditionnels
- Branchements conditionnels fortement biaisés

  If (erreur==vrai) then {afficher message; quitter}
- Ces branchements peuvent être prédits statiquement
  - Table mémorisant le comportement identique de n occurrences d'un branchement
  - Si comportement identique n fois, la prédiction statique de ce branchement est incorporée dans la trace.
    - Un branchement conditionnel supplémentaire peut être incorporé dans la trace.

Cache de traces
Redondance



Traces de 16 instructions



- Faut-il autoriser la redondance ?
- Compléter une trace avec une partie d'un bloc de base ?

Plan

- Pipeline, superscalaire et dépendances
- Processeur à exécution non ordonnée
  - Chargement des instructions
    - prédiction de branchement
    - Cache de traces
  - Renommage
  - Amorçage
  - Retrait
  - Accès à la mémoire
  - Réutilisation et prédiction de valeur
- VLIW et EPIC

Renommage

- Pour éviter des attentes dues à la réutilisation de registres

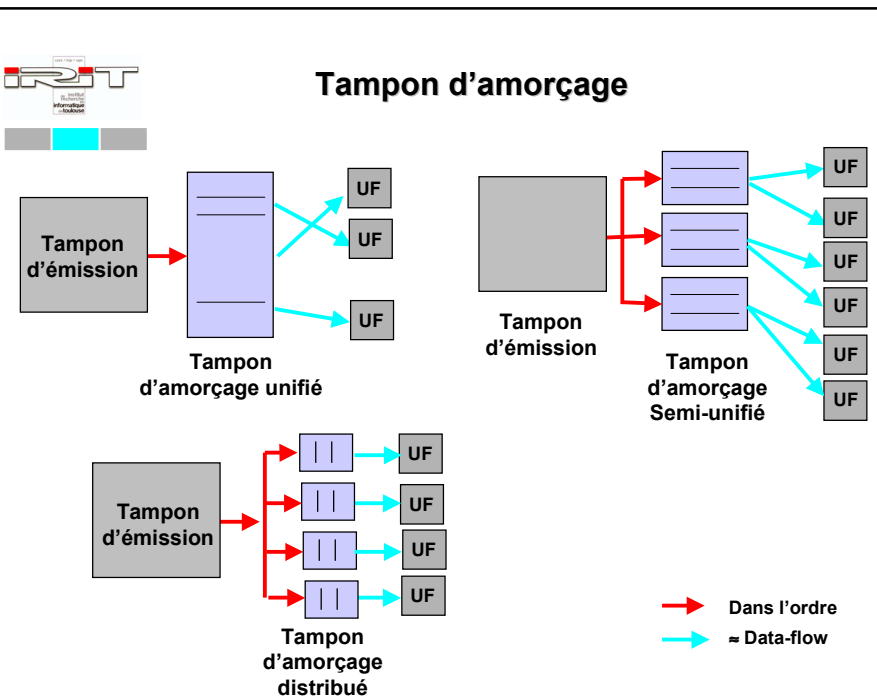
```
LD R1,0(R2)          LD R1,0(R2)
…                    …
ADD R1,R3,R4         ADD R2,R3,R4
```

- Solution:
  - Avoir autant de registres que d'instructions présentes dans le processeur. Ces registres sont mis à jour dès qu'une instruction a fini son exécution (dans le désordre).
  - Après le décodage, tout registre destination est renommé en un autre registre libre. La correspondance est enregistrée dans une table.
  - Tout opérande est renommé à partir de la table de correspondance sauf si pas de correspondance.

Renommage et spéculation

- Le renommage est effectué de manière spéculative.
- Problème en cas de mauvaise prédiction d'un branchement.
- Solution :
  - Sauvegarde de la table à chaque branchement prédit dans une pile de tables
  - Dépilement d'une table quand un branchement est vérifié
    - Exécution des branchements dans l'ordre = FIFO
    - Exécution des branchements dans le désordre = tampon associatif
  - ➢ Nombre maxi de branchements spéculatifs

Tampon d'amorçage

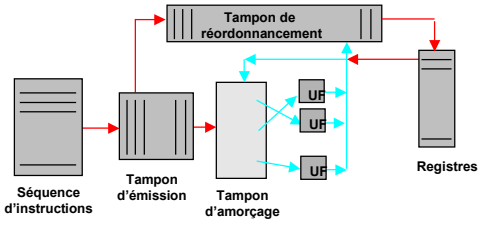


Unités fonctionnelles

- Entière, flottante, load/store
  - Sous-ensemble des opérations entières
  - Résolution branchements
    - Branchements exécutés dans l'ordre
  - Flottante capable de traiter des opérations entières
  - Load et store séparées
- SIMD
  - Entier
  - Flottant

Tampon de réordonnancement

- En charge de garder l'ordre des instructions de manière à les retirer dans l'ordre pour avoir des exceptions précises et une mise à jour des registres logiques dans l'ordre.
- Contient donc des informations d'état sur chaque instruction
  - Dans quel état, exception ?, type d'opération, …
  - Instructions insérées après le renommage
  - Instructions sortent dans le même ordre à la fin du pipeline



Retrait

- Tampon de réordonnancement = FIFO
  - Écritures dans les registres
    - Recopie du registre physique dans le registre logique
    - Numéro du registre physique dans le tampon
    - Dé-allocation de l'entrée correspondante dans la table de correspondance
  - Traitement des exceptions
  - Écritures en mémoire
- Une instruction ne sort que lorsque son résultat est calculé
  - Information sur l'état de l'instruction dans le tampon
  - Ce qui n'empêche pas les instructions de continuer à entrer dans le tampon tant qu'il n'est pas plein

Plan

- Pipeline, superscalaire et dépendances
- Processeur à exécution non ordonnée
  - Chargement des instructions
    - prédiction de branchement
    - Cache de traces
  - Renommage
  - Amorçage
  - Retrait
  - Accès à la mémoire
  - Réutilisation et prédiction de valeur
- VLIW et EPIC

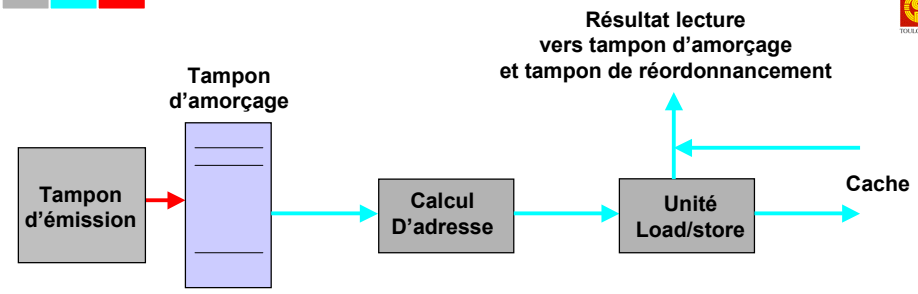
Accès aux données en mémoire

- Distinguer le calcul d'adresse de l'accès à la mémoire
- On ne peut pas renommer la mémoire comme on renomme les registres
  - Nombre d'emplacements trop grand.
- Les écritures ne sont pas effectuées en mémoire spéculativement
  - Écriture en mémoire au retrait
- Les lectures peuvent être effectuées spéculativement
  - Dépendances par rapport aux écritures en attente
- Les dépendances sont plus complexes à détecter
  - adresse sur 32 bits calculée vs 5 bits statique

Ordre des accès à la mémoire

- Lectures et écritures non spéculatives
  - Nécessaire pour les E/S (ex: FIFO)
  - Détection lors du calcul d'adresse
    - Accès à la MMU, détection de l'espace adressé
  - Instructions spéciales pour les lectures d'E/S
- Lectures spéculatives
  - Ordre total
  - Dépassement des écritures par les lecture si pas de dépendances
  - Dépassement avec court-circuit
    - Dépassement si indépendance
    - Court-circuit si dépendance dès que la donnée à écrire est connue

Accès à la mémoire
Mécanisme



- L'unité load/store ressemble à un tampon de réordonnancement.
  - Elle garde l'ordre des lectures et des écritures.
  - Recherche associative pour détecter les dépendances lorsqu'une lecture arrive.
  - Recherche inverse lorsqu'une écriture reçoit sa donnée pour le court circuit.
  - Exécution des écritures dans l'ordre quand le tampon de réordonnancement le permet.
  - Prise en compte de l'espace d'adressage pour les lectures dans l'espace des E/S lors du calcul d'adresse.

Accès mémoires spéculatifs
Prédiction de dépendances

- **Nécessité d'attendre que toutes les adresses des écritures antérieures soient connues ?**
- **Dépassement spéculatif seul**
  - **Suppose que les adresses sont différentes**
    - Systématiquement
    - Prédicteur de dépendances
  - **Permet seulement de masquer le temps d'accès au cache**
- **Dépassement spéculatif avec transmission**
  - **Les instructions qui dépendent du load sont exécutées avec la valeur chargée spéculativement**
  - **Nécessité d'un mécanisme pour relancer les instructions fautives**
    - Total
    - Fautives en un cycle
    - Fautives en chaîne
- **Dépassement et court-circuit spéculatifs**
  - **Prédicteur de dépendances**

Accès mémoire spéculatifs
Prédiction de dépendances

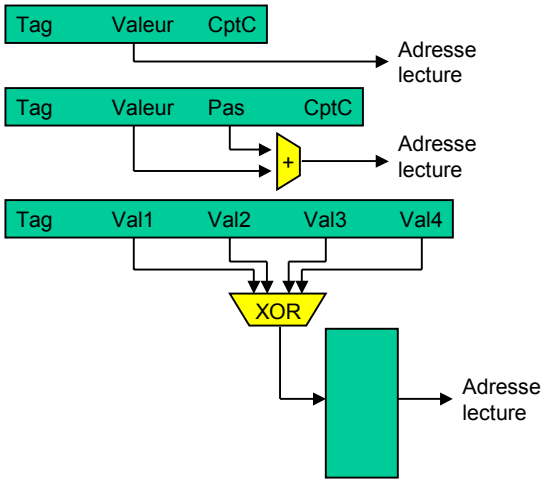
- **Détection d'erreur de prédiction**
  - **Lorsqu'une écriture reçoit son adresse**
  - **lecture relancée si mauvaise prédiction**
    - Une lecture peut avoir utilisé une valeur trouvée dans le cache
    - Une lecture peut avoir utilisé la valeur d'une écriture à la même adresse
  - **Possibilité d'erreurs multiples**
- **Prédiction aveugle**
  - **Une lecture est supposée indépendante des toutes les écritures en attente**
- **Prédiction par bit d'attente**
  - **A chaque instruction du cache est associé un bit d'attente.**
  - **Bit mis à 1 si dépendance observée à l'exécution précédente**
  - **Bit remis à 0 à intervalles réguliers**
    - Les adresses changent ➡ phénomène de saturation

Accès mémoire spéculatifs
Prédiction de dépendances (2)

- **Prédiction par ensembles d'alias**
  - **Une table (indexée par l'adresse de l'instruction) associe aux instructions un identificateur.**
  - **Deux instructions accédant à la même adresse ont le même identificateur.**
  - **Quand une lecture est chargée, la table renvoie son identificateur qui est utilisé pour trouver dans une autre table la dernière écriture ayant le même identificateur.**
  - **Table remise à zéro à intervalles réguliers.**
    - Les adresses changent… Saturation.

Accès mémoire spéculatifs
Prédiction de l'adresse

- Prédiction pour ne pas attendre le calcul de l'adresse
  - Dernière valeur
  - Pas
  - Contexte



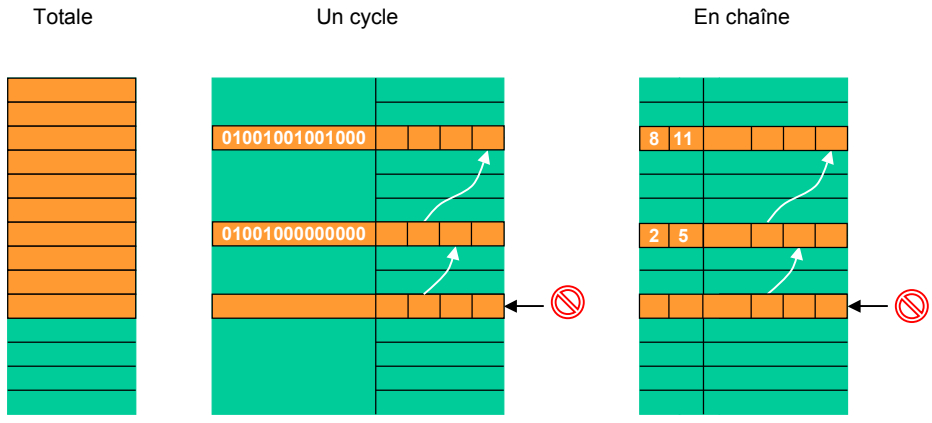
Plan

- Pipeline, superscalaire et dépendances
- Processeur à exécution non ordonnée
  - Chargement des instructions
    - prédiction de branchement
    - Cache de traces
  - Renommage
  - Amorçage
  - Retrait
  - Accès à la mémoire
  - Réutilisation et prédiction de valeur
- VLIW et EPIC

Spéculation sur les données
Prédiction de valeur

- Extension de la prédiction d'adresse d'une lecture
- Prédiction de la valeur produite par une instruction
  - Dernière valeur
  - Pas
  - Contexte
- Les instructions dépendantes peuvent être exécutées plus tôt
  - avec une valeur spéculative
- Mécanisme de récupération en cas de mauvaise prédiction
  - Totale
  - Fautives en un cycle
  - Fautives en chaîne

Spéculation
Mécanismes de récupération d'erreurs



Réutilisation d'instructions

- Table associant deux opérandes et une opération à un résultat
- Accès dès que l'instruction est décodée
- Évite d'occuper une unité fonctionelle
- Table accédée
  - Par l'adresse de l'instruction
  - Les opérandes et le type de l'opération
    - Deux instructions identiques mais à une adresse différente utilisent la même entrée
- Réutilisation d'instructions
  - ☹ Faible gain
- Réutilisation de traces
  - ☹ Faible taux car beaucoup d'opérandes donc non prêts
- Réutilisation de chaînes interne à un bloc de base
  - ☺ Moins d'entrées car instructions dépendantes

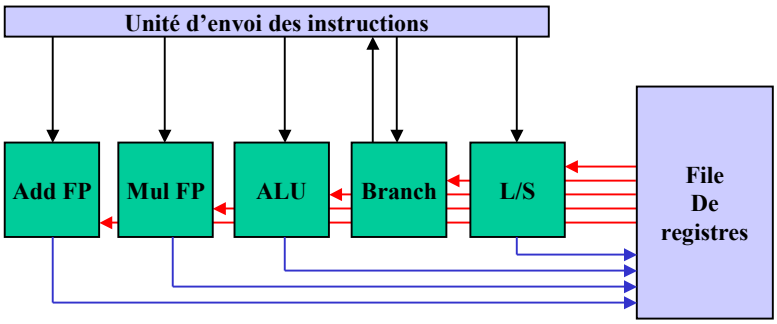
Plan

- Pipeline, superscalaire et dépendances
- Processeur à exécution non ordonnée
  - Chargement des instructions
    - prédiction de branchement
    - Cache de traces
  - Renommage
  - Amorçage
  - Retrait
  - Accès à la mémoire
  - Réutilisation et prédiction de valeur
- VLIW et EPIC

Architecture VLIW de base

Format instruction

Add FP	Mul FP	ALU	Branch	L/S



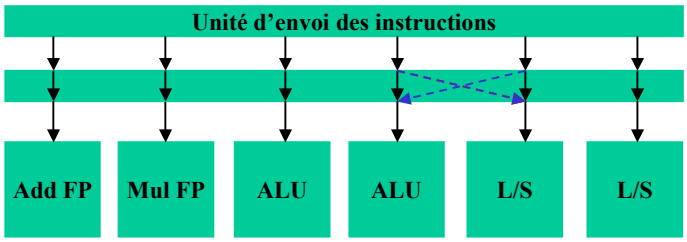
Le modèle VLIW
Avantages et inconvénients

- Le compilateur gère les dépendances entre instructions
- Matériel plus simple ➡ fréquence d'horloge plus grande
- Moins de problèmes pour avoir plus d'unités fonctionnelles

- Le compilateur doit être très intelligent et complexe
- Évolution et Taille du code

Le modèle EPIC

- **Ordonnancement statique et allocation dynamique**
  - **Préfixe associé à chaque instruction élémentaire indiquant quelle unité fonctionnelle doit être utilisée**
  - **Lancement des instructions plus complexe mais limitations possibles (Itanium)**
  - **Recombinaison des opérations**



Multithreading

- **Parallélisme d'instructions limité**
  - **Multiprocesseur**
    - Plusieurs processeurs complets et indépendants sur une puce
    - Facilité d'implémentation
  - **Multithreading**
    - Plusieurs contextes (registres)
    - Unités fonctionnelles partagées par les threads
    - Tolére les latences mémoire
    - Trois approches
      - Entrelaçage cycle par cycle
        - Autant de threads que d'étage dans le pipeline (scalaire)
        - Evite les aléas dus aux dépendances ➡simplifie le matériel
        - Nombre de threads important ➡autant de ressources nécessaires
      - Entrelaçage par bloc
        - Changement de thread sur opération à longue latence (cache miss)
      - Threads simultanés
        - Superscalaire large (degré 8 ➡ 5,4 insts/cycle)

Threads simultanés
L'exemple de l'Alpha 21464

- Exécution non ordonnée – degré 8
- 4 threads simultanés
- Ressources dupliquées
  - Table de correspondance
  - PC
- Ressources partagées
  - Caches L1 et L2 (taille L2 augmentée) et TLBs
  - Registres (nombre accru)
  - File d'instructions
  - Unités fonctionnelles
  - Prédicteur de branchement

"It looks like four chips, performs like two and is actually one chip with about 5 percent more transistors than a non-multithreaded device"

Joel Emer

Aujourd'hui c'est déjà demain

- Exemples de processeurs réseau
  - Xstream = multithread – sortie prévue en 2001
    - 8 threads – degré 8 (jusqu'à 4 instructions d'un même thread)
  - Intel IXP1200
    - 6 cœurs RISC multithread sur une puce
    - 1 strongARM
- Les embarqués deviennent moteur du développement de nouvelles architectures car il y a moins de contraintes.
  - Moins de dépendances au jeu d'instructions
  - Application spécifique