

L'architecture pipeline 000000 Les dépendances 0000 Architectures superscalaires 000000 Architecture pipeline — 1 / 24

# Architecture pipeline

## Chapitre 7

J.-C. Dubacq

IUT de Villeteuse  
Université Paris 13

S1D 2009

L'architecture pipeline 000000 Les dépendances 0000 Architectures superscalaires 000000 Architecture pipeline — 5 / 24

## La chaîne de travail du processeur

Imaginons que le travail du processeur soit découpé en cinq *étages*.

- 1 IF (Instruction fetch) charge l'instruction.
- 2 ID (Instruction decode) décode l'instruction, lit les registres.
- 3 EX (Execute) utilise l'UAL pour exécuter l'instruction.
- 4 MEM (Memory) écrit/lit le résultat en mémoire si Store ou Load.
- 5 WB (WriteBack) écrit le résultat dans les registres (le cas échéant).

Le nombre d'étages est variable selon les processeurs. Quatre est le minimum (cycle vital du processeur), mais des découpages plus fins existent (ici, découpage classique RISC).

L'architecture pipeline 000000 Les dépendances 0000 Architectures superscalaires 000000 Architecture pipeline — 4 / 24

## Un processeur à 1000 Ghz ?

- Dans la guerre entre IBM et Intel (années 90), croyance commune : la fréquence d'horloge est le principal facteur de performances du CPU malgré des contre-exemples déjà flagrants à l'époque.
- Accélération de l'horloge : tenir compte de l'évacuation de la chaleur et de la stabilisation des valeurs.
- Ce qui compte (plus) : instructions par secondes (IPS).
- Nombre de cycles par instructions (CPI) : 1 au maximum.
- Durée du cycle CCT en (n)s, fréquence  $f$  en (G)Hz.

**Théorème (Performances du processeur)**

$$IPS = \frac{f}{CPI} = \frac{1}{CCT \times CPI}$$

L'architecture pipeline 000000 Les dépendances 0000 Architectures superscalaires 000000 Architecture pipeline — 6 / 24

## Combien de cycles pour exécuter une instruction ?

- Registres intercalés dans le chemin de données, 1 cycle d'horloge – informations passent 1 étage.
- Chaque instruction doit donc utiliser cinq cycles d'horloge pour être exécutée complètement.

- On a donc  $CPI' = 5$ .
- En revanche, chaque étage prend un temps qui est un cinquième de l'ancien temps de cycle.
- On a donc  $CCT' = \frac{1}{5}CCT$ .
- Performances : exactement pareil !

## L'analogie de la laverie

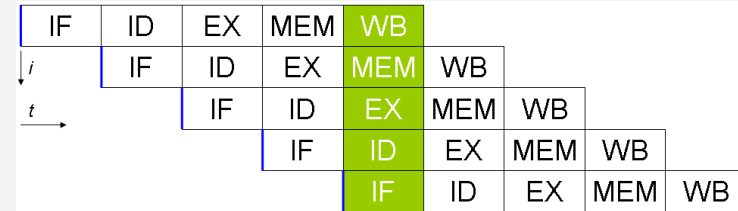
- Dans une laverie, il faut 1h pour laver, 1h pour sécher et 1h pour repasser/plier le linge.
- Des clients arrivent régulièrement.
- Organisation classique : on ne sert qu'un client à la fois, on lave, sèche et repasse avant de prendre le client suivant.
- Un client = 3h, un client toutes les 3h
- Organisation intelligente : on commence à servir un client pendant qu'on finit le précédent.
- Un client = 3h, un client toutes les heures !

## L'amorçage du pipeline

- Une instruction met toujours le même temps à être exécuté.
- Le débit d'instructions est multiplié par le nombre d'étages.
- La première instruction ne sort pas au premier cycle.
- Trous dans le pipeline quand il est vide.
- Dépendances entre les instructions posent des problèmes.

## Le modèle pipeline idéal

- On commence une instruction avant la fin de la précédente.



- On a donc  $CPI'' = 1$ ,  $CCT'' = \frac{1}{5}CCT$ .
- Performances : cinq fois plus rapide !

## Le problème des sauts et branchements

- Les branchements conditionnels dépendent du résultat d'opérations.
- La récupération de l'instruction suivante ne peut pas intervenir tant que l'étage 3 (exécution) n'a pas été faite complètement.
- Impossibilité d'assurer un débit maximal.
- Solution du délai : introduire des retards artificiel (NOP).
- Possibilité de faire du branchement *anticipé*, suivi d'une *purge* si on s'est trompé.
- En C, primitives pour dire si on pense qu'un test est plutôt vrai ou faux.

L'architecture pipeline 000000 Les dépendances 000000 Architectures superscalaires 000000  
 Dépendance d'instructions Architecture pipeline — 13 / 24

## Les solutions à la dépendance d'instructions

- Introduction de NOP fictifs pour attendre.
- Autre solution : toujours supposé qu'il n'est pas pris, et vider le pipeline si on s'est trompé.
- La purge nécessite une infrastructure matérielle plus complexe, et n'est possible que si aucune écriture n'a été faite.
- Amélioration : tenir compte du dernier choix effectué et supposé qu'on aura le même.
- Nécessite un chemin de données encore plus complexe (utilisation d'un BTB).

L'architecture pipeline 000000 Les dépendances 000000 Architectures superscalaires 000000  
 Dépendance de données Architecture pipeline — 16 / 24

## D'une variable à l'autre

- Introduction de NOP fictifs pour attendre.
- $A = 1; B = 1; C = 1; NOP; NOP; NOP; A = A + C; NOP; NOP; NOP; C = B + A.$
- Autre solution : faire du réordonnancement de code.
- Exemple :  $A = 1; C = 1; B = 1; NOP; NOP; A = A + C; NOP; NOP; NOP; C = B + A.$
- Nécessite une infrastructure matérielle très complexe pour le faire de façon dynamique.
- Souvent fait au niveau du compilateur.

L'architecture pipeline 000000 Les dépendances 000000 Architectures superscalaires 000000  
 Dépendance de données Architecture pipeline — 15 / 24

## D'une variable à l'autre

- Que se passe-t-il si on fait  $A = 1; B = 1; C = 1; A = A + C; C = B + A?$
- On suppose qu'au départ,  $A = B = C = D = 0.$
- En théorie,  $A = 2$  et  $C = 3.$
- Si on a un pipeline à cinq étages, la lecture des registres se fait à l'étape 2 et l'écriture à l'étape 5.
- Quand on fait  $A = A + C$ ,  $A$  et  $C$  valent encore 0.
- Quand on fait  $C = A + B$ ,  $A$  vaut 1 et  $B$  vaut 0.
- Donc  $A$  vaut 0 et  $C$  vaut 1 à la fin !

L'architecture pipeline 000000 Les dépendances 000000 Architectures superscalaires 000000  
 Performances d'un pipeline Architecture pipeline — 18 / 24

## D'une variable à l'autre

- Accélération : 
$$S = \frac{\text{temps sans pipeline}}{\text{temps avec pipeline}}$$
- En pratique, jamais atteint à cause des dépendances.
- Mécanisme de *purge* du pipeline.
- Réordonnancement et insertion de délais.
- Prédiction de branchements.
- Compilateur spécial, code balisé.

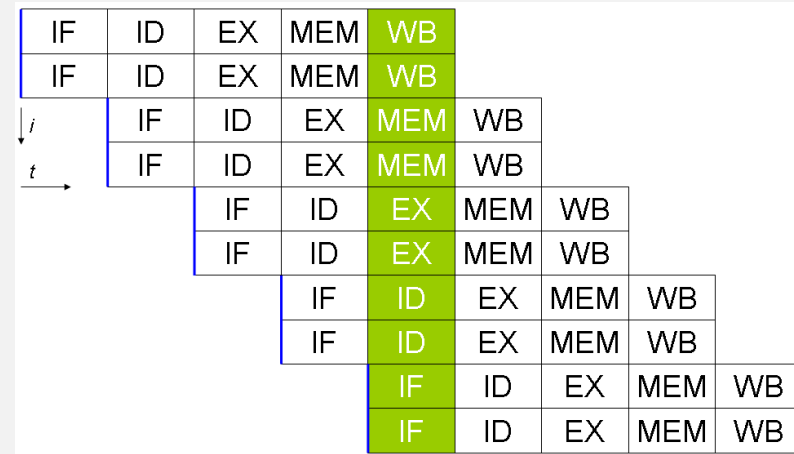
## Processeur superscalaire

- Duplication du matériel du chemin de données.
- Exécution de deux instructions simultanées, si elles n'ont pas de conflit de données.
- Double le débit d'instructions avec le même cycle d'horloge.
- Conflits plus fréquents.
- Parfois duplication seulement des éléments les plus lents (les plus rapide traitent plus vite leur tâche).
- Aussi duplication du chemin de données pour traiter calculs flottants et entiers indépendamment.
- Jusqu'à huit unités juxtaposées.

## Architecture VLIW et superpipeline

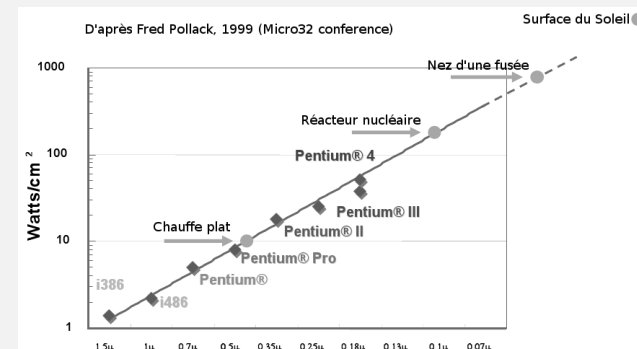
- Very large instruction word.
- Évolution du superscalaire.
- Une instruction donne du travail à toutes les unités d'exécution.
- Solution pratique aux problèmes de dépendances dans les superscalaires.
- Superpipeline : découpages en des étages beaucoup plus nombreux (et division du temps de cycle).
- Pentium 4 Prescott : 31 étages différents.
- Très grande granularité.

## Architecture superscalaire



## Deux directions de développement

- Ne pas augmenter la fréquence d'horloge.
- Problème de la dissipation de chaleur.
- Lenteur de la mémoire : pentium 4, jusqu'à 600 cycles processeurs pour un cycle mémoire.



## Le multi-threading et le multi-processing

- Simultaneous multi-threading (SMT) : solution non-superscalaire, alterne entre plusieurs fils d'exécution (2, parfois 4, plus à venir) dans un même pipeline.
- Partage du cache de niveau 1, des registres.
- Multiprocesseurs (Symmetric multiprocessing, SMP) : deux (ou plus) processeurs qui communiquent.
- Très ancien (dès 1961).
- Multicœurs (Core multiprocessing, CMP) : techniques superscalaires pour créer plusieurs cœurs dans un processeur, réduction d'énergie (partage du cache de niveau 2, alimentation commune...)
- Comme deux processeurs, quasiment indépendants.