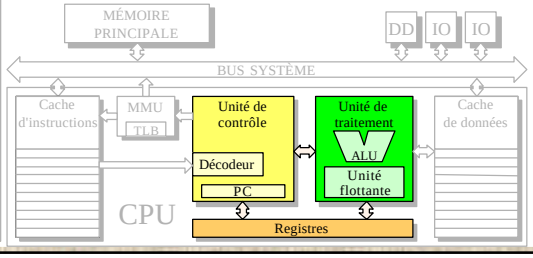
Parallélisme des instructions

- Ordonnancement dynamique
- Lancement multiple
- Compilation

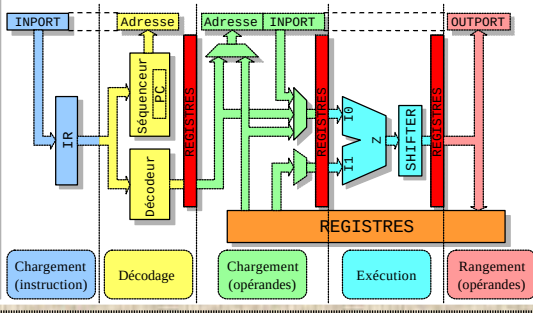


Boucle de traitement

Chargement (instruction)	L'instruction est cherchée en mémoire et chargée dans le registre d'instruction (IR)
Décodage	L'instruction est décodée pour en extraire les signaux de contrôle pour l'unité de traitement
Chargement (opérandes)	Les opérandes demandés par l'instruction sont cherchés dans les registres ou en mémoire
Exécution	L'opération demandée est effectuée sur les opérandes choisis
Rangement (opérandes)	Le résultat de l'opération est stocké dans un registre ou en mémoire

Pipeline - Vue d'ensemble

Le pipeline exécute plusieurs instructions en parallèle.



Aléas d'un pipeline

La présence d'un pipeline (et donc le partage de l'exécution d'une instruction en plusieurs étages) introduit des aléas de données: le résultat d'une opération dépend de celui d'une opération précédente qui n'est pas encore terminée.

Deux instructions contiguës `i` et `j` peuvent présenter des aléas de données (dépendances entre les opérandes des deux instructions). En particulier:

• RAW (*read-after-write*): `j` essaie de lire un registre avant que `i` ne l'ait modifié

```
i:  move R1, R2        {R2 ¬ R1}
j:  add  R2, R3, R4    {R4 ¬ R2+R3}
```

Ordonnancement des instructions

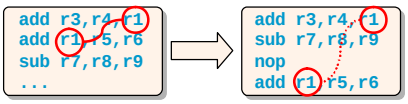
Deux des trois méthodes décrites pour "résoudre" le problème des aléas de données (*pipeline stall, bypassing*) n'essaient pas d'éliminer les aléas, mais cherchent tout simplement à limiter le délai ou à simplifier leur traitement.

La troisième méthode (le ré-ordonnancement des instructions par le compilateur, utilisée surtout pour les LOAD) essaie effectivement d'éliminer les aléas en modifiant l'ordre d'exécution des instructions. Les limitations de cette technique sont toutefois nombreuses.

Ces trois méthodes réalisent un ordonnancement statique des instructions: l'ordre d'exécution est fixé lors de la compilation, et ne peut pas être modifié par le processeur.

Aléas de données - Solution

Une solution peut être fournie par le compilateur, qui peut changer l'ordre d'exécution des instructions de façon à éliminer les aléas. Si nécessaire, les instructions intercalées peuvent être des NOP.



Malheureusement, l'utilité de cette méthode est limitée:

- le compilateur n'est pas toujours en mesure de détecter les aléas (par exemple, si les aléas concernent des pointeurs).
- le nombre d'instructions à intercaler dépend de la structure (nombre d'étages) du pipeline;
- la complexité du compilateur en est fortement augmentée.

Ordonnancement dynamique

Une méthode plus performante pour réduire le délai introduit par les aléas des données consiste à changer l'ordre des instructions pendant l'exécution du programme en utilisant du matériel dédié. Cette technique est appelée ordonnancement dynamique.

Dans un processeur à ordonnancement dynamique, les instructions sont exécutées dès que leurs opérandes sont disponibles. L'ordre établi lors de la compilation n'est donc pas nécessairement respecté.

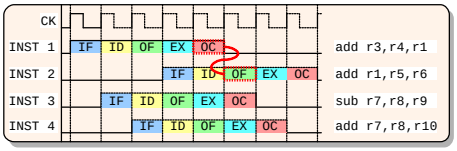
Cette exécution non-ordonnée des instructions a plusieurs avantages mais aussi plusieurs désavantages. Notamment, elle est plus performante dans l'élimination des aléas, mais nécessite une quantité très importante de logique de contrôle.

Ordonnancement dynamique

Dans un processeur à ordonnancement dynamique, les instructions sont exécutées dès que les opérandes sont disponibles.

```
add r3,r4,r1
add r1,r5,r6
sub r7,r8,r9
add r7,r8,r10
```

```
add r3,r4,r1
sub r7,r8,r9
add r7,r8,r10
add r1,r5,r6
```



Ordonnancement dynamique

L'ordonnancement dynamique essaie donc de modifier l'ordre d'exécution des instructions pour "éviter" les aléas de données. Son rôle est semblable à celui du compilateur lors des LOAD: il intercale des instructions exécutables entre deux instructions liées par un aléa. Toutefois, le processeur est potentiellement capable d'effectuer cette opération de façon beaucoup plus efficace:

- Les adresses sont explicites: même si l'aléa concerne un pointeur dans le code, lors de l'exécution la valeur de ce pointeur est connue est donc l'aléa peut être détecté;
- Le processeur connaît la profondeur du pipeline et donc sait exactement combien d'instructions il doit intercaler pour éviter un aléa;
- L'opération s'effectue sans l'aide du compilateur et donc sans augmenter sa complexité.

Ordonnancement dynamique

L'ordonnancement dynamique est une technique très performante, mais elle introduit plusieurs complications.

Tout d'abord, le changement de l'ordre d'exécution des instructions implique non seulement une exécution non -ordonnée, mais aussi une terminaison non-ordonnée, ce qui introduit des dépendances (et donc des aléas) des données qui n'existaient pas auparavant.

Ensuite, elle nécessite une quantité très importante de logique de contrôle, particulièrement en ce qui concerne le décodage des instructions.

Finalement, cette terminaison non-ordonnée complique ultérieurement le traitement des interruptions.

Aléas de données

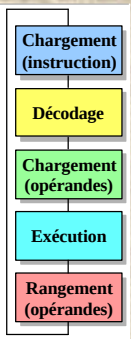
Lors que les instructions sont exécutées dans le désordre, deux instructions contiguës `i` et `j` peuvent présenter trois types d'aléas de données:

- RAW (*read-after-write*): `j` essaie de lire un registre avant que `i` ne l'ait modifié
- WAR (*write-after-read*): `j` essaie de modifier un registre avant que `i` ne l'ait lu

```
i: move R4, R2        {R2 ¬ R4}
j: add  R2, R3, R4    {R4 ¬ R2+R3}
```

- WAW (*write-after-write*): `j` essaie d'écrire dans un registre avant que `i` ne l'ait fait

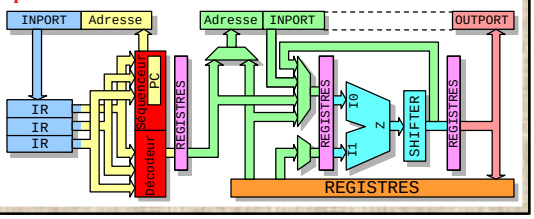
```
i: move R1, R3        {R2 ¬ R1}
   and  R2, R4, R5    {R5 ¬ R2&R4}
j: add  R3, R4, R2    {R2 ¬ R3+R4}
   and  R2, R7, R6    {R6 ¬ R2&R7}
```



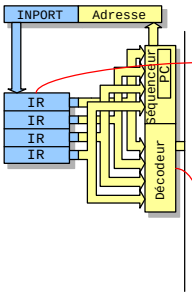
Implémentation

L'ordonnancement dynamique complique considérablement le contrôle du pipeline. En particulier, l'étage décode doit identifier tous les aléas dans le code pendant l'exécution et décider quelle instruction va être exécutée au coup d'horloge suivant.

En pratique, plusieurs instructions doivent être chargées et décodées en parallèle.



Ordonnancement dynamique - Implémentation



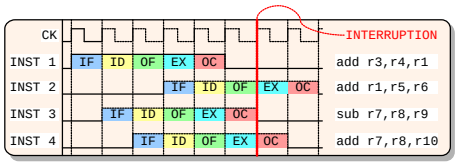
Queue d'instructions: quand l'instruction au sommet de la queue est suspendue, d'autres instructions peuvent être décodées et exécutées (si elles n'ont pas de dépendances).

Le décodeur doit être capable d'accueillir plusieurs instructions en parallèle, d'identifier les dépendances entre les instructions, et d'envoyer aux étages suivants les instructions qui peuvent être exécutées.

Interruptions

Le **traitement des interruptions**, déjà compliqué par la présence d'un pipeline (et donc par la nécessité de "vider" le pipeline des instructions à moitié exécutées), devient encore plus difficile dans un processeur à ordonnancement dynamique.

En particulier, la **terminaison non-ordonnée** des instructions rend terriblement complexe le **redémarrage du programme** après le traitement d'une interruption.



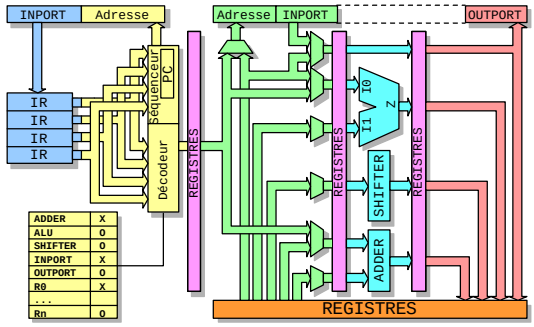
Lancement multiple

L'ordonnancement dynamique est souvent associé au **lancement multiple**: la nécessité de charger et de décoder plusieurs instructions en parallèle suggère immédiatement la possibilité d'**exécuter plusieurs instructions en parallèle**.

Pour ce faire, les processeurs à lancement multiple incluent **plusieurs unités de traitement** et peuvent donc effectuer des calculs sur plusieurs opérandes en parallèle.

Par rapport à l'exécution dans le désordre introduite par l'ordonnancement dynamique, le lancement multiple ne complique pas excessivement le contrôle (notamment, il n'introduit pas de nouveaux aléas). Par contre, il permet (théoriquement) d'obtenir des performances très élevées (**plusieurs instructions par cycle d'horloge**).

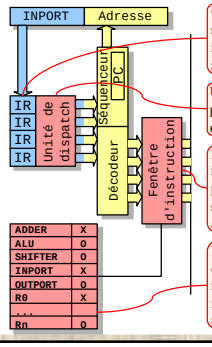
Lancement multiple - Implémentation



Politiques d'envoi des instructions

- *In-order issue with in-order completion*:
  Les instructions sont cherchées, décodées et envoyées aux unités de traitement dans l'ordre d'écriture du programme. Les résultats sont écrits dans le même ordre.
- *In-order issue with out-of-order completion*:
  Les instructions sont cherchées, décodées et envoyées aux unités de traitement dans l'ordre d'écriture du programme. Par contre, les résultats sont écrits dès qu'ils sont obtenus.
- *Out-of-order issue with out-of-order completion*:
  Les instructions sont cherchées, décodées et envoyées aux unités de traitement dès que les ressources nécessaires sont disponibles (sauf si des dépendances existent). Les résultats sont écrits dès qu'ils sont obtenus.

Out-of-order issue - Implémentation



Queue d'instructions: quand l'instruction au sommet de la queue est suspendue, d'autres instructions peuvent être exécutées (si elles n'ont pas de dépendances).

Unité de dispatch: dès leur arrivée dans le processeur, les instructions sont envoyées au décodeur, qui analyse les dépendances.

Fenêtre d'instruction: le décodeur envoie les instructions dans la fenêtre, tant qu'il y a de la place. Les instructions sont envoyées en exécution, sans regarder l'ordre, pour autant qu'il n'y ait pas de conflits de ressources.

Un tableau de marques (*scoreboard*) est une liste des ressources du processeur: dès qu'une instruction demande une (ou plusieurs) des ressources, cette dernière devient indisponible pour les instructions suivantes.

Politiques d'envoi des instructions

I1 a besoin de deux cycles d'exécution | I5 dépend du résultat de I4
I3 et I4 utilisent la même ressource | I5 et I6 utilisent la même ressource

In-order issue with in-order completion

decode		window	execute			writeback		cycle
I1	I2							1
I3	I4		I1	I2				2
	I4		I1					3
I5	I6				I3	I1	I2	4
	I6				I4	I3		5
				I5		I4		6
				I6		I5		7
							I6	8

Out-of-order issue with in-order completion

decode		window	execute			writeback		cycle
I1	I2							1
I3	I4		I1	I2				2
	I4		I1		I3	I2		3
I5	I6				I4	I1	I3	4
	I6			I5		I4		5
				I6		I5		6
						I6		7
								8

Out-of-order issue with out-of-order completion

decode		window	execute			writeback		cycle
I1	I2							1
I3	I4	I1, I2	I1	I2				2
I5	I6	I3, I4	I1		I3	I2		3
		I4, I5, I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6
								7
								8

Lancement multiple - Limitations

Quelles sont les limitations du lancement multiple? Pourquoi ne pas lancer 50 instructions par cycle?

Même si l'on ne considérait pas les difficultés à construire le matériel sous-jacent (hypothèse bien sûr irréaliste), les limitations les plus sévères du lancement multiple sont imposées par les limitations intrinsèques en parallélisme d'instructions: souvent, il n'y a tout simplement pas suffisamment d'instructions indépendantes pour occuper toutes les ressources d'un processeur.

En effet, la recherche d'instructions indépendantes est la motivation principale de la conception de nombreuses architectures de la dernière génération (VLIW, SMT), ainsi que du développement de nouvelles techniques de compilation.

Exemple: le processeur PowerPC 601

- Architecture 32 bits
- Instructions à taille fixe (32 bits)
- Alignement obligatoire des instructions
- *Big-endian* ou *little-endian*
- Architecture superscalaire à trois unités de traitement (entiers, virgule flottante et sauts)
- Cache interne: 32 KB, unifié, 8 voies *set-associative*, adressé physiquement
- Pipelines à différentes longueurs
- Bus de données à 64 bits
- Prédiction statique du saut
- 32 registres entiers et 32 registres virgule flottante
- Registre de conditions

Exemple: le processeur PowerPC 601

Unité des entiers (IU)

- Exécute toutes les opérations sur des données entières et tous les accès à la mémoire (*load/store*)
- Contient: une ALU
  un multiplicateur
  un diviseur
  32 registres généraux
- Peut recevoir une instruction par coup d'horloge

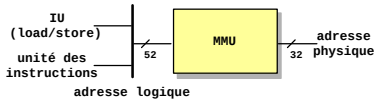
Unité virgule flottante (FPU)

- Contient: un multiplicateur/additionneur
  un diviseur
  32 registres généraux à 64 bits
- Implémentation complète du standard IEEE 754, en simple ou double précision

Exemple: le processeur PowerPC 601

Unité des instructions

- Contient deux parties:
  - une queue d'instructions à 8 niveaux
  - une unité de traitement des sauts (BPU)
- La queue d'instructions peut recevoir jusqu'à 8 instructions de la cache par coup d'horloge. Trois instructions peuvent être envoyées en parallèle aux différentes unités de traitement.
- Les instructions sur les entiers sont toujours exécutées dans l'ordre. En conséquence, une seule instruction accédant à la mémoire est retirée de la queue par cycle d'horloge.



Exemple: le processeur PowerPC 601

Caractéristiques du cache:

- 32 KBytes
- unifié
- adressé physiquement
- 8 voies, associatif par ensembles
- ensembles de 64 lignes, 2 secteurs par ligne, 8 mots par secteur
- algorithme LRU pour le remplacement
- écriture *copy-back* ou *write-through* au choix
- chaque ligne possède un tag et 4 bits d'état (2 par secteur)
- les 16 mots d'une ligne sont contigus dans la mémoire et ne peuvent pas croiser la limite d'une page
- les opérations du cache se font sur la base d'un secteur

Exemple: le processeur PowerPC 601

Modes d'adressage:

- Adressage des données:
  - Indirect de registre — `adresse = (RA)`
  - Indirect de registre avec index — `adresse = (RA) + (RB)`
  - Indirect de registre avec index immédiat — `adresse = (RA) + offset`

6	11	16	31
opcode	RB	RA	offset

- Adressage des sauts:
  - adressage absolu

6	29	30	31
opcode	adresse	1	LK

  - adressage relatif

6	29	30	31
opcode	offset	0	LK

  - saut au registre de lien (LR)
  - saut au registre compteur (CTR)

Exemple: le processeur PowerPC 601

Prédiction des sauts

- La BPU a besoin d'un seul cycle d'horloge pour décoder et exécuter une instruction de saut.
- Dans le cas d'un saut conditionnel, la BPU vérifie dans le pipeline si une instruction en cours peut modifier la condition. Si non, l'adresse de saut est calculée. Si oui, la branche est prédite, en fonction d'un bit y spécifié dans l'instruction.
- Si la prédiction s'avère fausse, on enlève les instructions de la branche prédite et on prend l'autre branche.
- Le *writeback* est interdit dans les instructions d'une branche tant que la condition n'est pas résolue.
- Il existe un seul niveau de prédiction: on ne fait pas de prédiction à l'intérieur d'une branche qui n'est pas encore résolue.

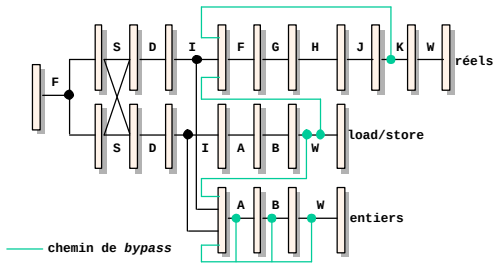
Exemple: le processeur Alpha

- Architecture 64 bits
- Taille unique des instructions (32 bits)
- Alignement obligatoire des instructions et des données
- *Little-endian*
- Architecture superscalaire avec des pipelines de longueur différente
- Adresse virtuelle sur 64 bits, sans segmentation
- Trois types de données supportés, en 32 (*longword*) ou 64 (*quadword*) bits:
  - entiers
  - virgule flottante IEEE
  - virgule flottante DEC
- Il n'y a pas de division pour les entiers
- Il n'y a pas d'opérations sur les bytes (par contre, il y a des instructions de manipulation des bytes)
- Les sauts sont prédits
- 32 registres entiers (R31 toujours égal à 0) et 32 registres virgule flottante (F31 toujours égal à 0), tous à 64 bits

Exemple: le processeur Alpha

- **Quatre unités de traitement:**
  - unité des entiers
  - unité des réels
  - unité des *load/store*
  - unité des sauts
- **Deux instructions peuvent être exécutées en parallèle**
- **Plusieurs types d'étage**
  - F (*fetch*): recherche de deux instructions dans la cache.
  - S (*swap*): envoi des deux instructions dans le bon pipeline. Prédiction du saut.
  - D (*decode*): identification des opérandes.
  - I (*issue*): lecture des opérandes. Analyse des dépendances. Au-delà de cette phase il est impossible de bloquer le pipeline.
  - A: première phase de l'ALU. Les opérations et les décalages courts sont terminés. L'adresse du *load/store* est calculée. Le PC est mis à jour.
  - B: deuxième phase de l'ALU. Les décalages longs sont terminés. La cache de données est lue pour les *loads*.
  - W (*write*): écriture des registres. Le succès/échec de l'accès à la cache est déterminé: lors d'un succès, on écrit dans le tampon d'écriture.

Exemple: le processeur Alpha



Exemple: le processeur Alpha

- Les instructions sont cherchées et envoyées à l'exécution dans l'ordre.
- Les registres entiers possèdent 4 ports de lecture et 2 d'écriture. Les registres réels possèdent 3 ports de lecture et 2 d'écriture.
- Le pipeline des réels utilise 5 phases pour les opérations d'addition et multiplication. La division utilise 31 ou 61 cycles, selon la précision.
- Il y a un total de 38 *bypasses*, pour permettre l'utilisation du résultat d'une opération comme opérande d'une instruction postérieure.

Prédiction des sauts:

- La première fois qu'un saut a lieu, la prédiction se fait de façon statique: les sauts en arrière sont pris. Par la suite, la prédiction se fait dynamiquement: un bit d'histoire est gardé dans la cache pour toutes les instructions de saut.
- La pénalité pour une mauvaise prédiction est de 4 cycles.