6- Parallélisme au niveau des instructions

Parallélisme d'instructions (Instruction Level Parallelism, ILP)

- Pour améliorer les performances par rapport au pipeline, lancement simultanée de plusieurs instructions
- Nécessite plusieurs unités de calcul (ALUs, unités de calcul flottant, unités pour calcul vectoriel)

Instructions scalaires multiples exécutées en même temps

- Architecture superscalaire
- Architectures Very Long Instruction Word (VLIW)

Parallélisme de données

Une instruction à la fois, mais opère sur un vecteur de données (Single Instruction, Multiple Data, SIMD).

- Architecture Vectorielle

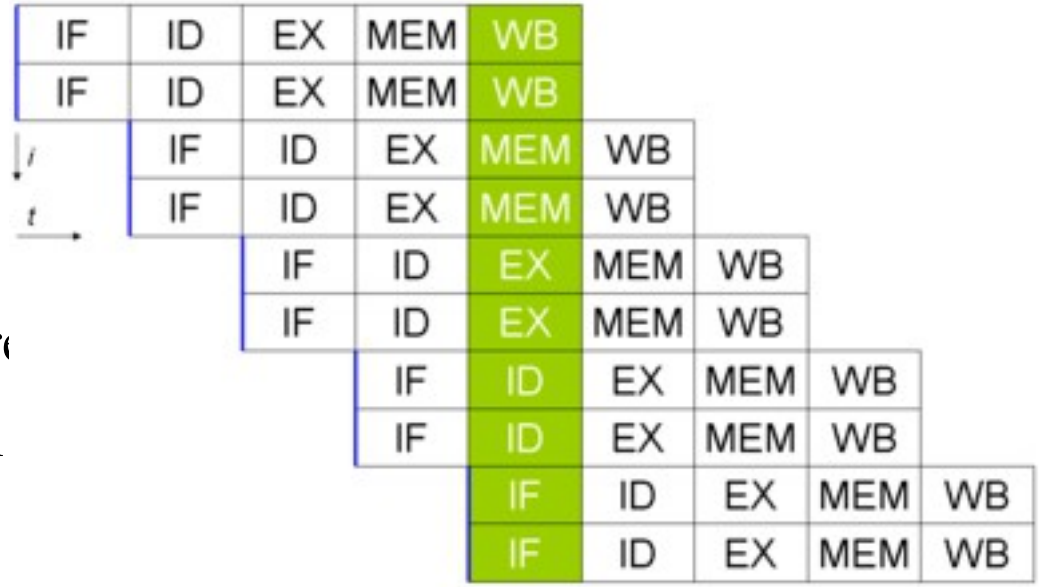
6-a Architecture Superscalaire

Caractéristiques

- Plusieurs instructions lancées en même temps
- Multiples unités fonctionnelles (y compris décodage)

Adaptations par rapport pipeline simple

- **Risques élevés de dépendances**
- Tout est plus compliqué !
- Mécanismes additionnels
  - Renommage de registr
  - Out of order execution



6-a Architectures superscalaires: renommage de registres

Objectif:

- Eviter des dépendances entre étages dues à des réutilisations de registres

Principe:

- Faire la distinction entre les registres rééls de la machine (registres physiques) et les registres utilisés dans le code assembleur (registres virtuels)
- Après décodage, assigne un registre physique pour un registre virtuel
- Le matériel garde la correspondance pour préserver le sens du code

Il peut y avoir plus de registres physiques que de registres virtuels !

Le compilateur ne sait pas combien de registres physiques il y a:

- Le matériel fait une partie de l'optimisation, ou corrige ce qu'a fait le compilateur !

6-a Architectures superscalaires: renommage de registres

Sans renommage

```
1: lw r2,(r1)
2: mul r3,r2,r2
3: sw r3, (r4)
4: lw r2, 4(r1)
5: mul r3, r2,r2
6: sw r3, (r4)
```

Dépendances WAR entre 2 et 4, et WAW entre 3 et 5 dues à des réutilisations de registre

Impossible de lancer 1, 2, 3 et 4,5,6 simultanément

→ **stall du pipeline superscalaire**

Avec renommage

```
1: lw pr2,(pr1)
2: mul pr3,pr2,pr2
3: sw pr3, (pr4)
4: lw pr12, 4(pr1)
5: mul pr13, pr12,pr12
6: sw pr13, (pr4)
```

On peut lancer les deux blocs de 3 instruction en parallèle

→ **exécution parallèle de 1,2,3 et 4,5,6**

6-a Architectures superscalaires: prédiction de branchements

Source importante de stall: les branchements

- Pipeline simple: Spéculation. On sait si on fait un branchement à l'étage EX
- Pipeline parallèle: Prédicteur de branchement

Principe prédicteur de branchement (BTB, branch target predictor)

- Utiliser le passé pour prédire l'avenir
- Garder dans une table, pour les dernières instructions de branchement décodées, le dernier résultat du branchement: pris, pas pris et cible du branchement
- Entrées de la table indexé par PC: on peut savoir très tôt (étage IF) dans le pipeline si on fait un branchement

6-a Architectures superscalaires: prédiction de branchements

Algorithmes de prédiction

- **Simple**: se base sur le dernier branchement.
  - Si branchement pris la dernière fois, prédit qu'on le prendra.
  - Si branchement pas pris, alors prédit pas pris.
- Prédiction très tot: étage IF !
- Si prédiction échoue: flush (quand même nécessaire), mise à jour BTB.

6-a Architectures superscalaires: prédiction de branchements

Algorithmes de prédiction

- **A 2bits**: se base sur les 2 derniers branchements.
- Prédiction suivant valeur des deux bits:
  - 0 et 1: branchement pris
  - 2 et 3: branchement pas pris
- Mise à jour
  - Si pris, décrémente compteur (si 0, reste à 0)
  - Si pas pris: incrémente compteur (si 3, reste à 3)

En moyenne, prédiction juste dans plus de 90% des cas

6-a Architectures superscalaires: prédiction de branchements

Corrélation entre branchements (Yeh, Patt, 1992)

If (x[i]>5) y=y+4;

If (x[i]>3) c=2;

- Si le premier branchement est pris, le deuxième aussi

→ les 2 branchements sont corrélés

Historique des branchements: garde la trace des résultats (pris ou pas pris, adresse de branchement) des séquences des derniers branchements

- Indexé par le PC (program counter)
- Sur pentium Pro: garde sur 2 bits les 2 derniers branchements
- 95% des prédictions correctes

Doit arbitrer avec méthode précédente de prédiction

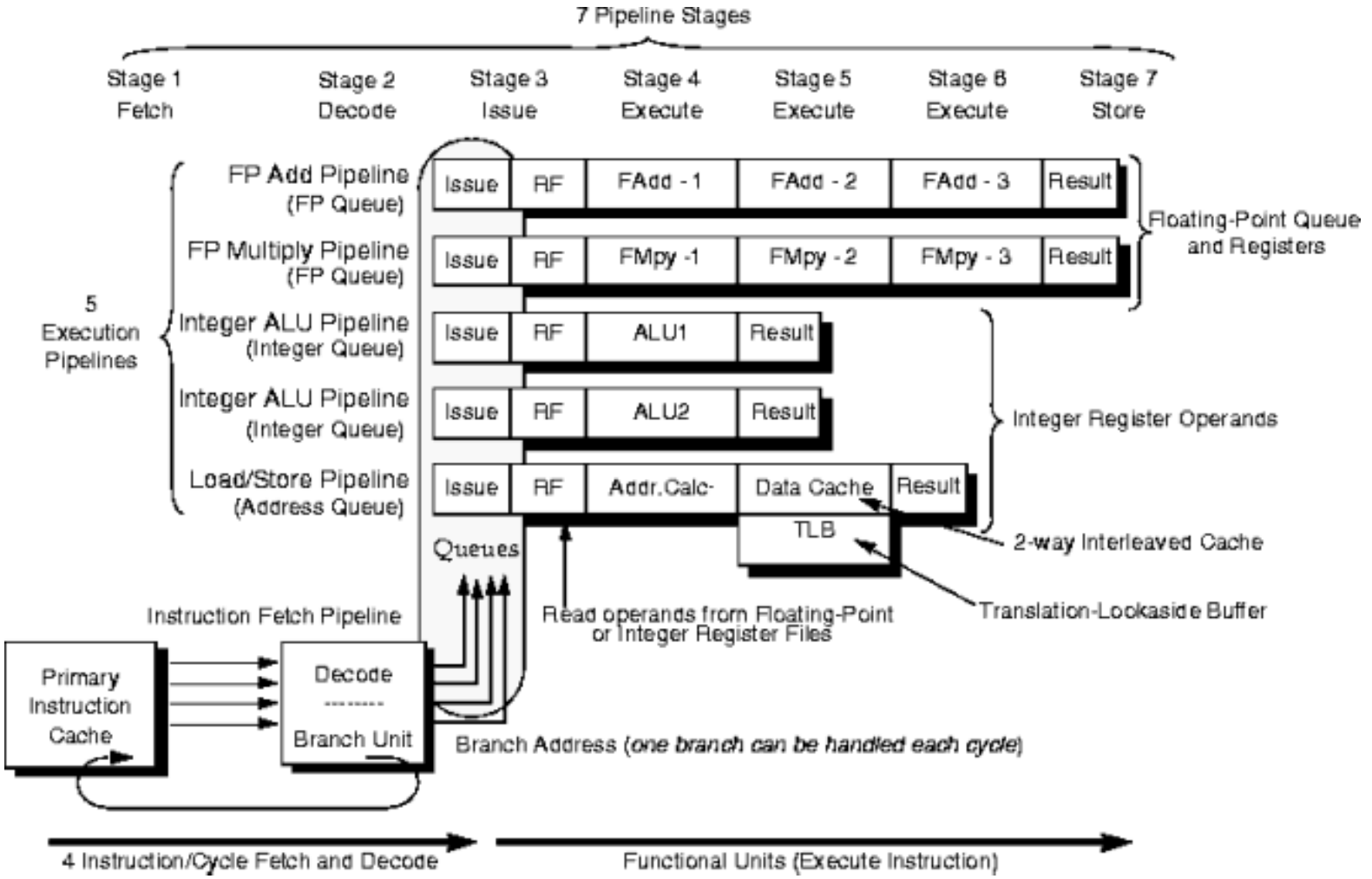
6-a Architectures superscalaires: exécution dans le désordre

Exécution dans le désordre = Out of order (OoO) execution

Principe:

- **Des instructions prêtes ne sont pas bloquées par des instructions en attente**
  - Evite qu'un stall d'une instruction ne bloque tout le pipeline
- **Les instructions peuvent avoir un pipeline spécialisé**
  - Les instructions n'utilisent que les unités dont elles ont besoin
  - Evite qu'une opération arithmétique ne traverse l'étage de la mémoire des données

6-a Architecture Superscalaire: exemple du MIPS 10000



6-a Architectures superscalaires: exécution dans le désordre

Files d'attentes (buffer)

- Les instructions sont placées après décodage dans une file d'attente et sont exécutées dès que leurs opérandes sont prêts
- Les instructions doivent écrire dans les registres dans l'ordre d'exécution d'origine (sinon risque de changer sens du programme)
    - Les instructions sont placées dans un buffer avant l'étape WB et remises dans l'ordre.

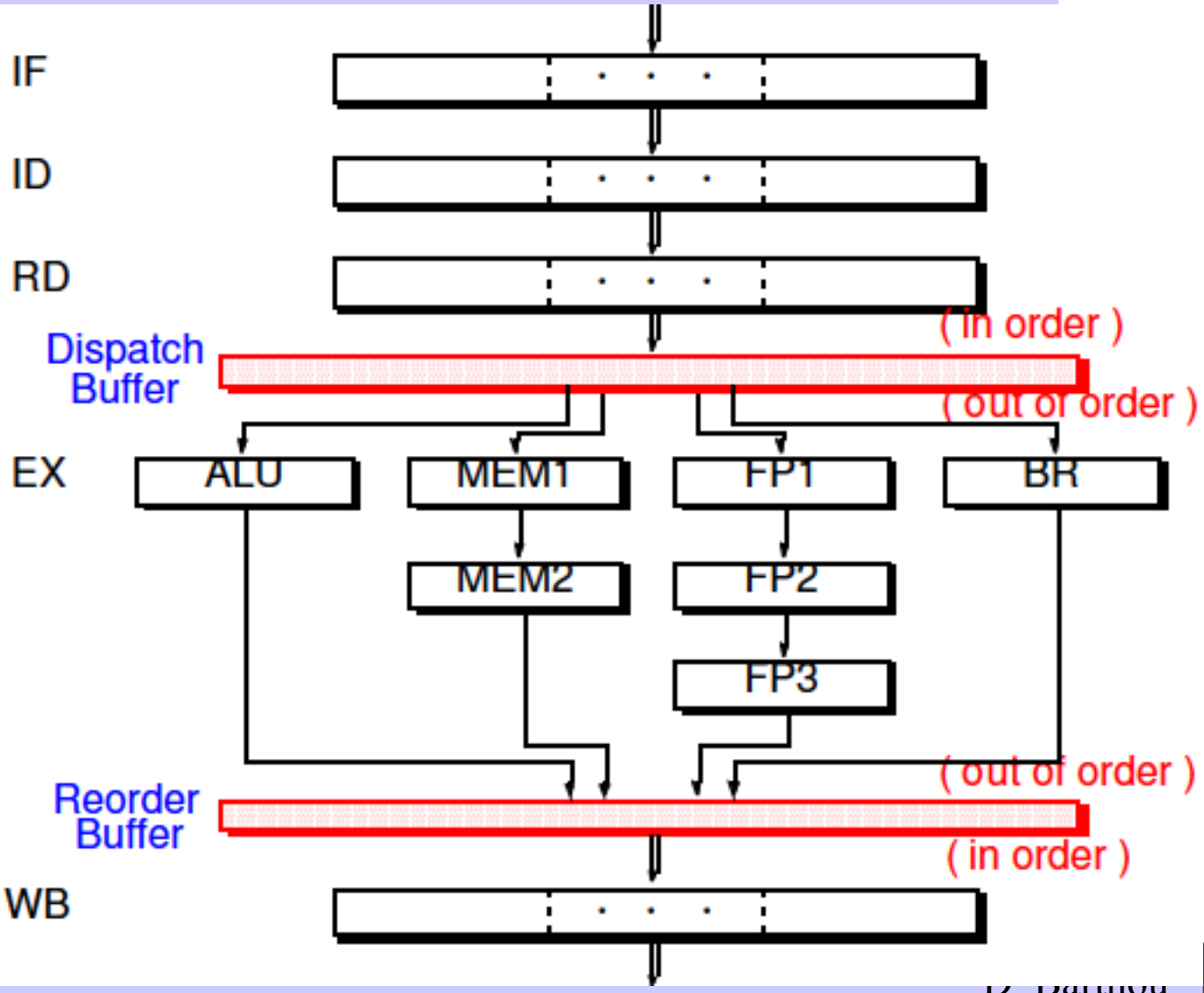
6-a Architectures superscalaires: exécution dans le désordre

Les instructions non prêtes sont mises en attente

Les instructions prêtes vont vers leur ALU ou autre pipeline

Deux files d'attente (buffer

- Dispatch buffer: buffer des instructions en attent d'opérandes ou d'unités disponibles
- Reorder buffer: écriture des registres dans l'ordre



6-a Architectures superscalaires: exécution dans le désordre

Pipeline parallèle

- Instruction fetch centralisé
- Décodage d'instructions centralisé
- Renommage de registres centralisé (supprime dépendances WAW, WAR)
- Exécution des instructions distribuée
  - Les pipelines d'exécution sont parallèles, de longueur différente (Mem, ALU, calculs flottants, vectoriels, branchements)
  - Evite l'uniformisation des longueurs des étages du pipeline

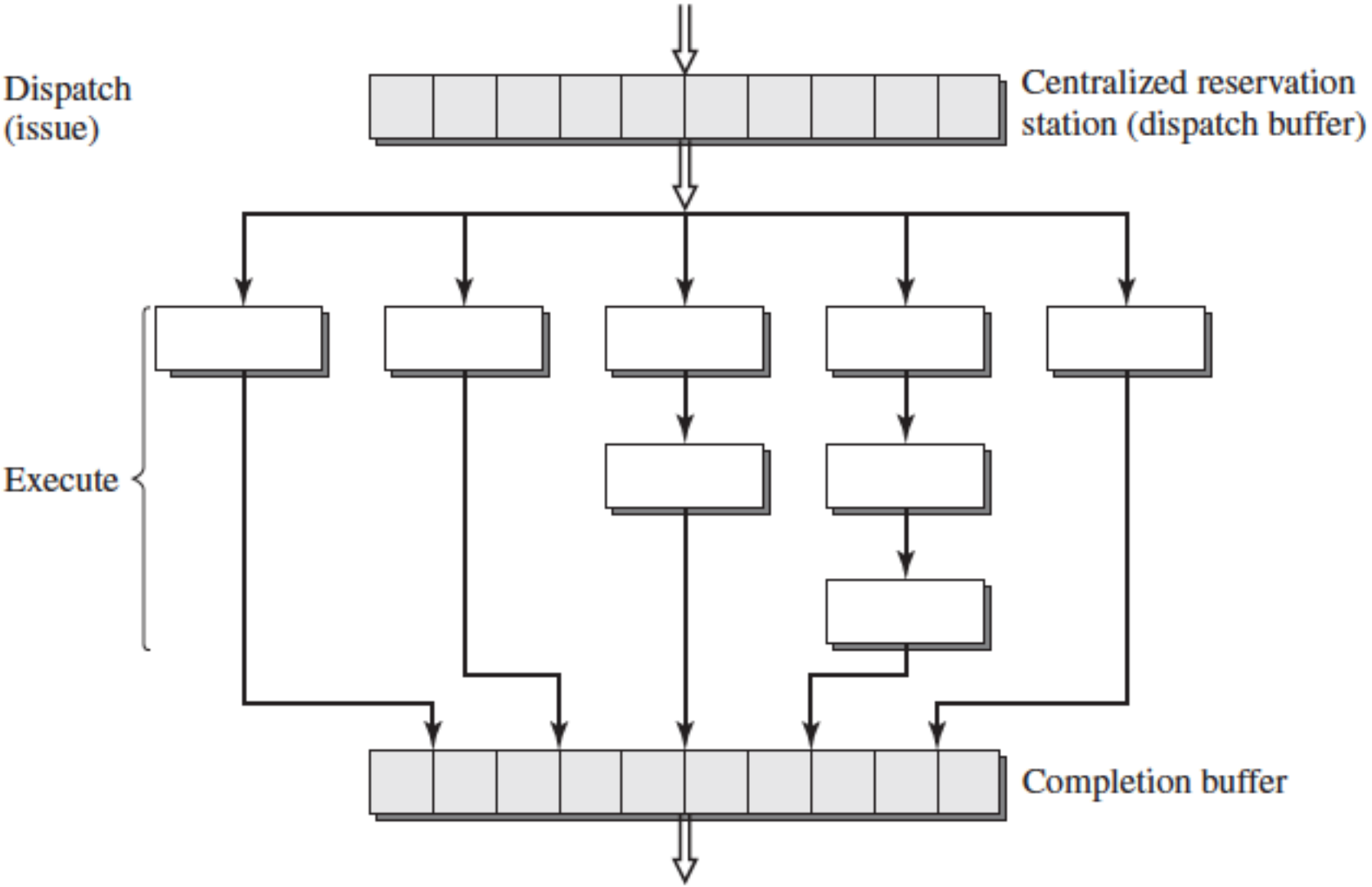
Buffer pour l'attente des opérandes et des unités (dispatch buffer)

- Un pour toutes les unités
- Ou un par type d'unités

6-a Architectures superscalaires: exécution dans le désordre

Exemple

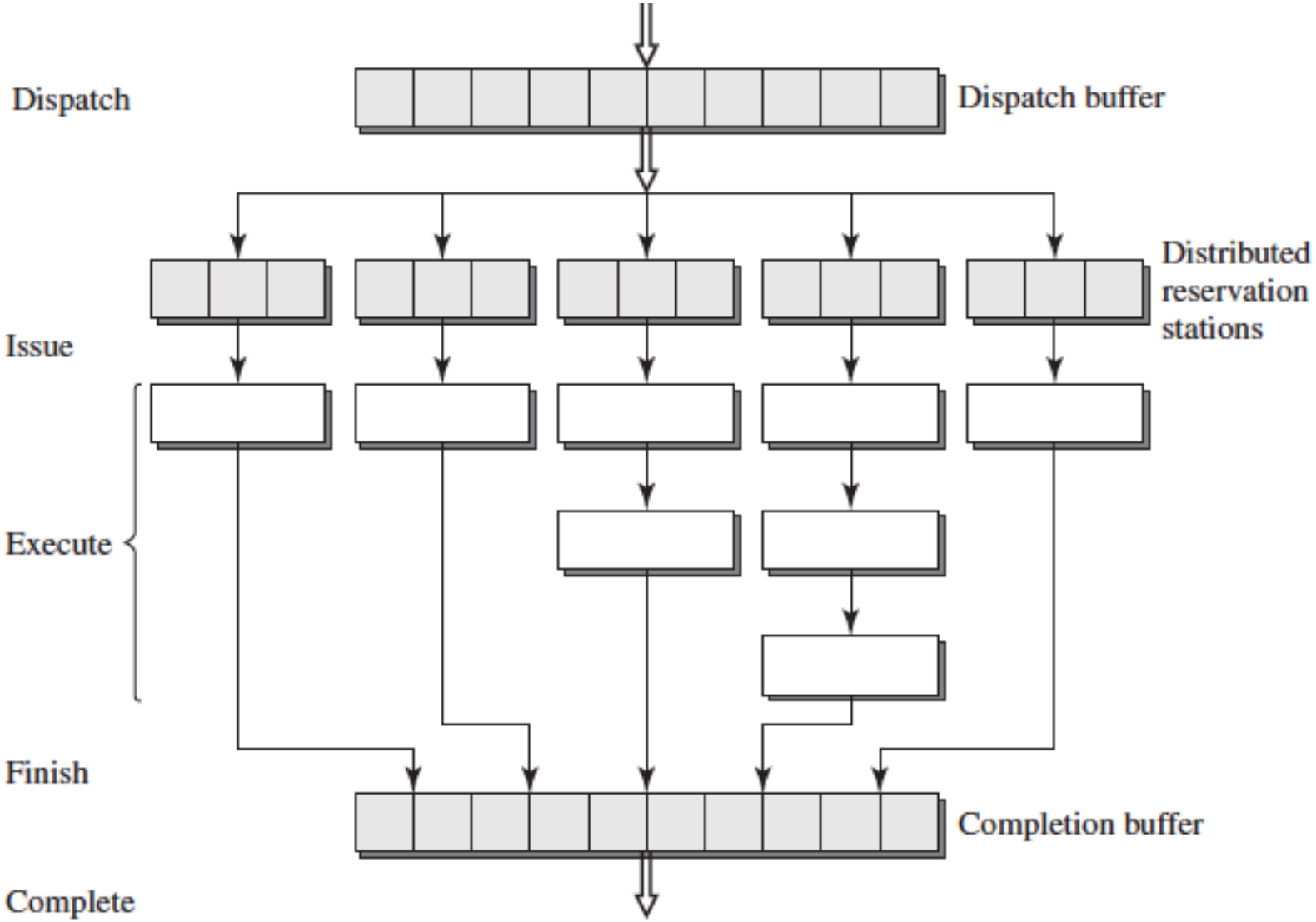
Pentium



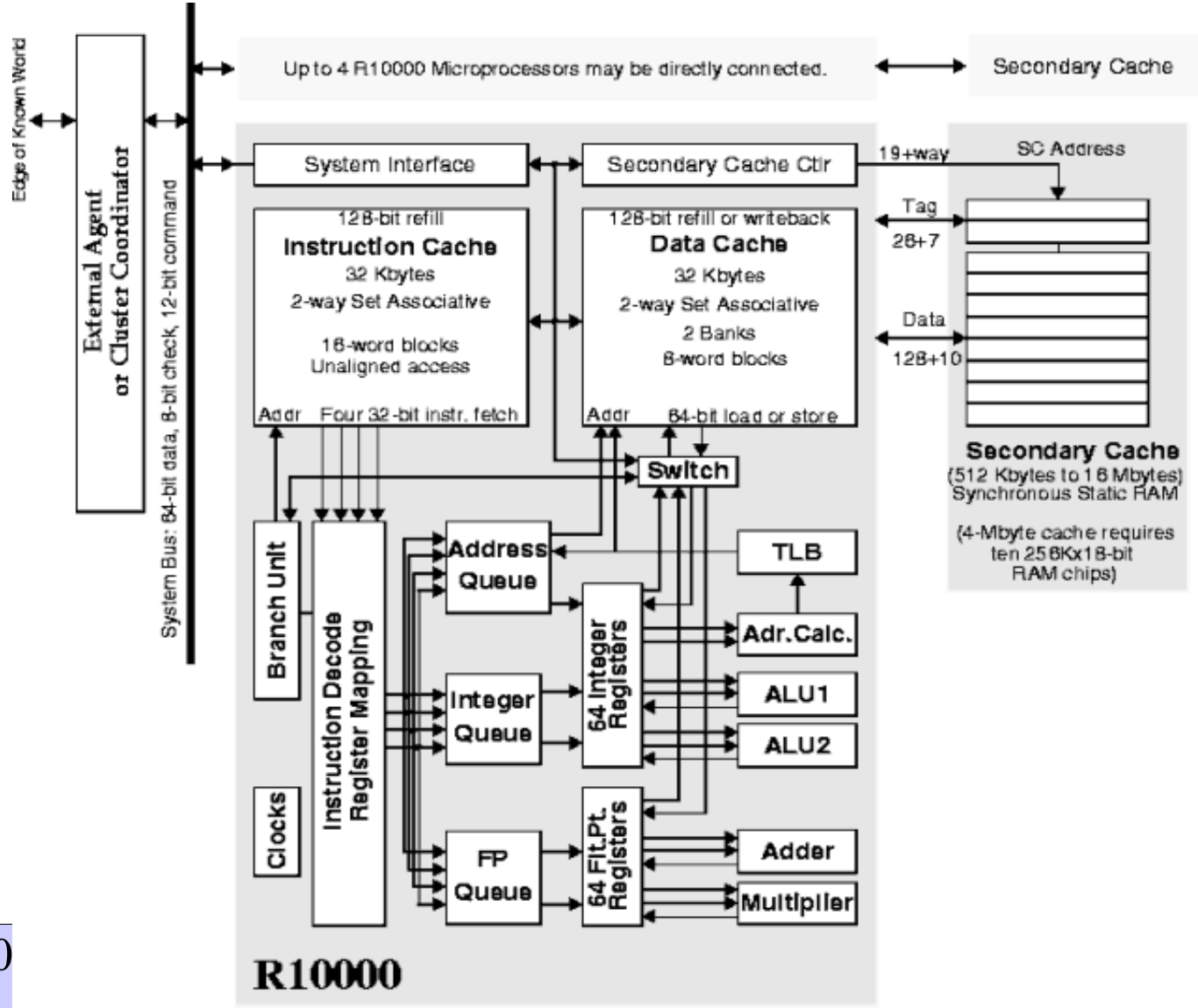
6-a Architectures superscalaires: exécutions dans le désordre

Exemple

MIPS



6-a Architectures superscalaires: exemple MIPS 10000



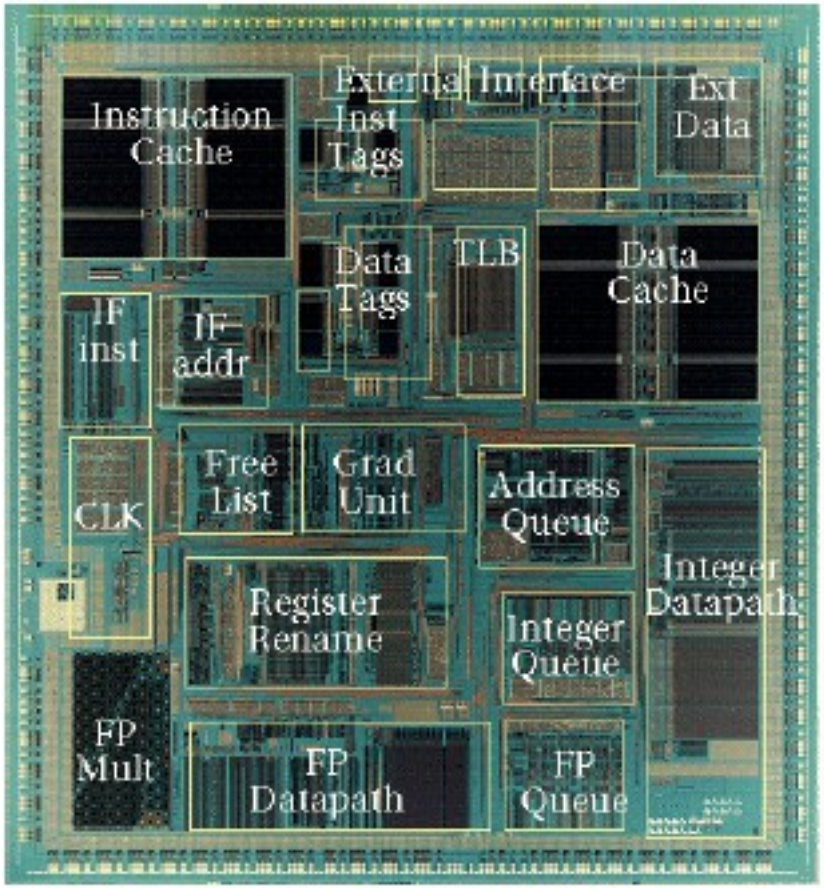
6-a Architectures superscalaires: exécutions dans le désordre

Coût de l'OoO (Patterson) et performances

MIPS	R5000	R10000	Rapport
Horloge	200Mhz	195Mhz	1x
Instructions/cycle	1	4	4x
Etages de pipeline	5	5-7	1.2x
Modèle d'exécution	Dans l'ordre	Dans le désordre	
Taille de la puce (sans les mémoires caches), mm2	32	205	6.3x
Temps de développement (h.m)	60	300	5x
Performances SPECint95	5,7	8,8	1.6x

6-a Architectures superscalaires: exécutions dans le désordre

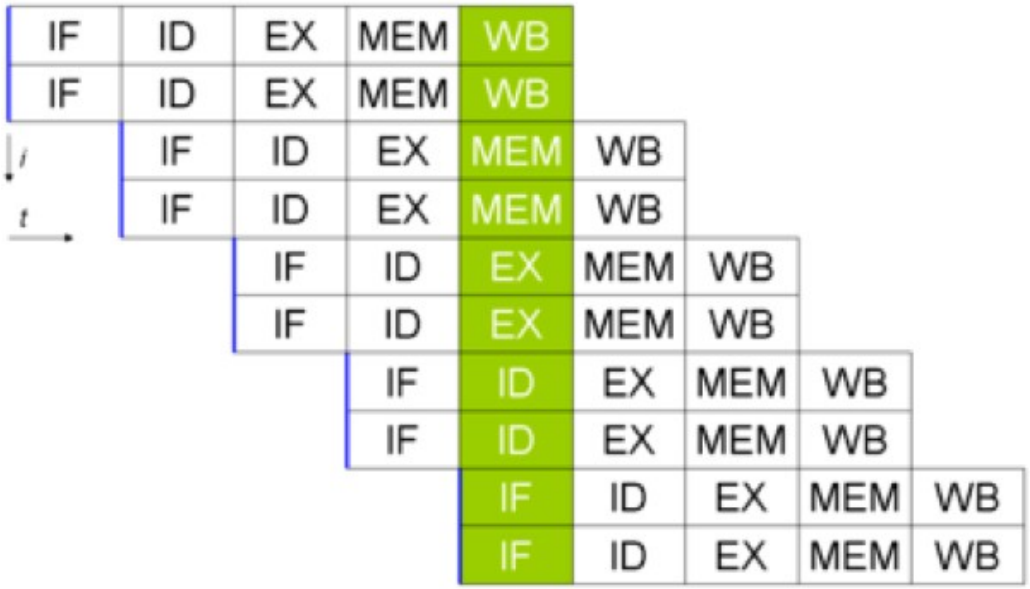
- MIPS R10000



6-b Very Large Instruction Word

Caractéristiques

- Les instructions sont regroupées statiquement, dans le code asm, en paquet, un paquet est exécuté par cycle
- Le nombre d'instruction par paquet est fixé (au max)
- Le compilateur fait les paquets

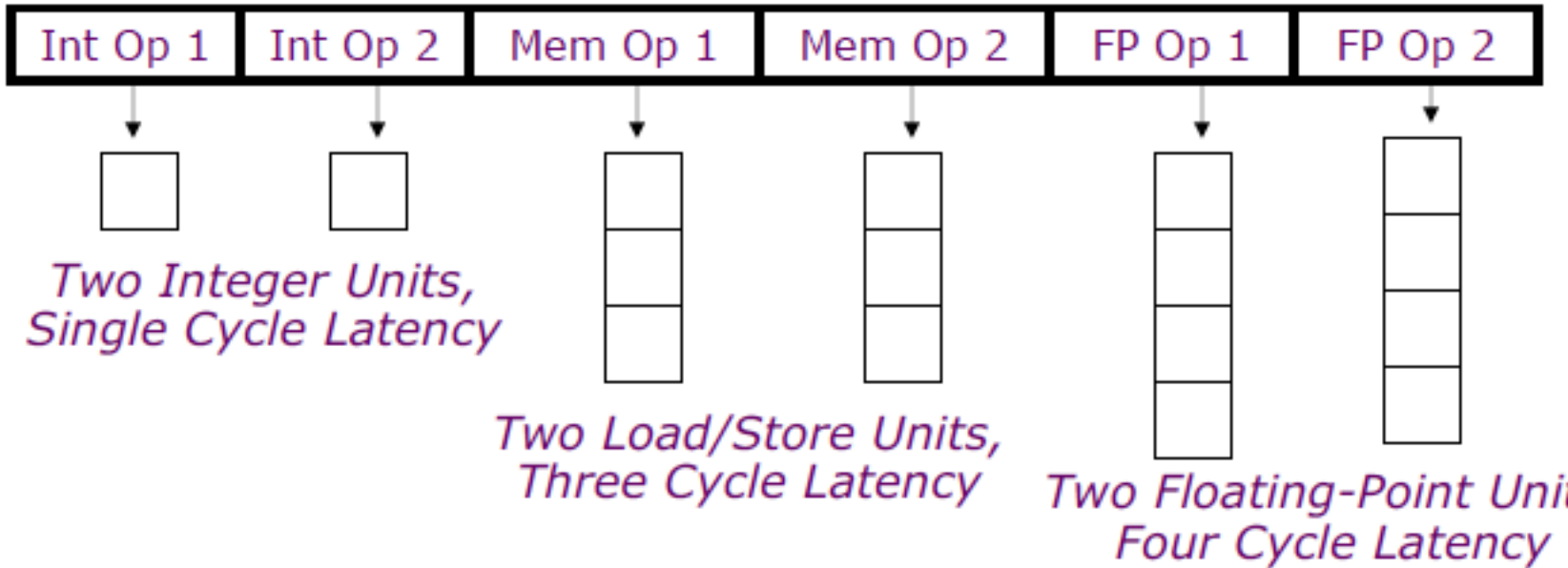


6-b Very Large Instruction Word

Les instructions ont une latence fixée (ici, 4)

Propriétés attendues par le hardware

- Parallélisme dans l'instruction → Pas de dépendances dans l'instruction entre opérations (ni RAW, WAR, WAW). Le hardware ne vérifie pas!
- Les données sont prêtes avant d'être utilisées → Pas de stall inséré entre instructions, pas de détection faite par le hardware.



6-b Very Large Instruction Word

Responsabilité du compilateur

- Trouver du parallélisme pour les opérations d'une instruction
  - → garantir qu'il n'y a pas de dépendance
  - → optimiser le code suffisamment pour détecter du parallélisme, l'exprimer
  - → bien choisir les registres (allocation des registres) pour éviter des dépendances dues à des réutilisations (pas de renommage par le hardware)
- Garantir que les instructions inter dépendantes sont séparées par une latence suffisante
  - → insertion de **nop**

Le compilateur a un impact très important sur le code et ses performances!

6-b Very Large Instruction Word: exemple de l'Itanium

Itanium: jeu d'instruction EPIC (Explicit Parallel Instruction Computing)

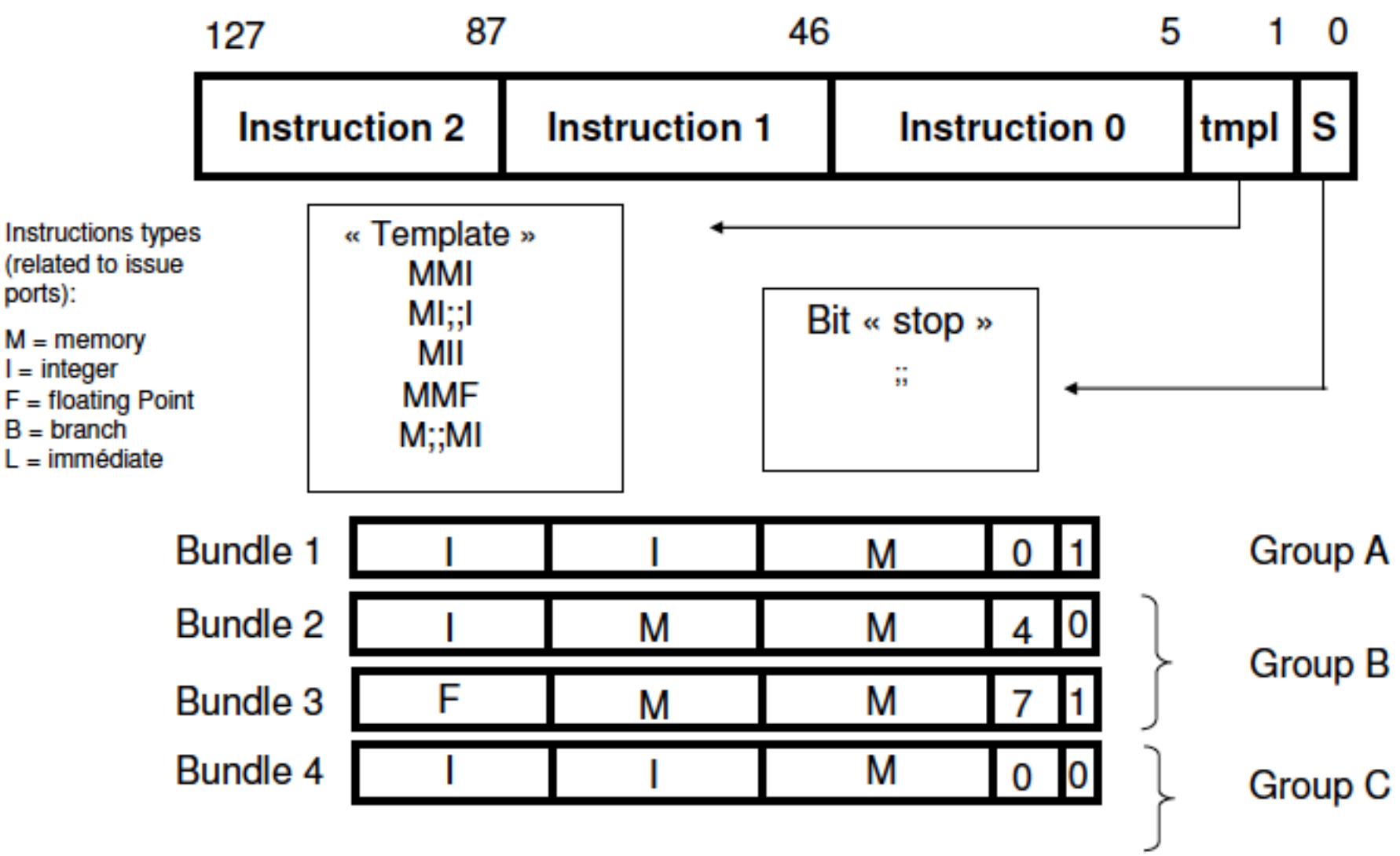
Unités fonctionnelles: (Itanium2)

- 6 ALUs (calcul entier)
- 2 unités pour le calcul flottant (avec Fused Multiplied Add en 1 cycle)
- 2 unités mémoire: capables de faire chacune 2 loads et 1 store / cycle

Instructions:

- **Groupe d'instructions**: séquence d'instructions parallèles, séparées par ;;
- **Bundle d'instructions**: les instructions viennent par paquet (bundle) de 3. Un groupe peut s'étaler sur plusieurs bundles, un bundle peut s'étaler sur 2 groupes. Un bundle = 128 bits

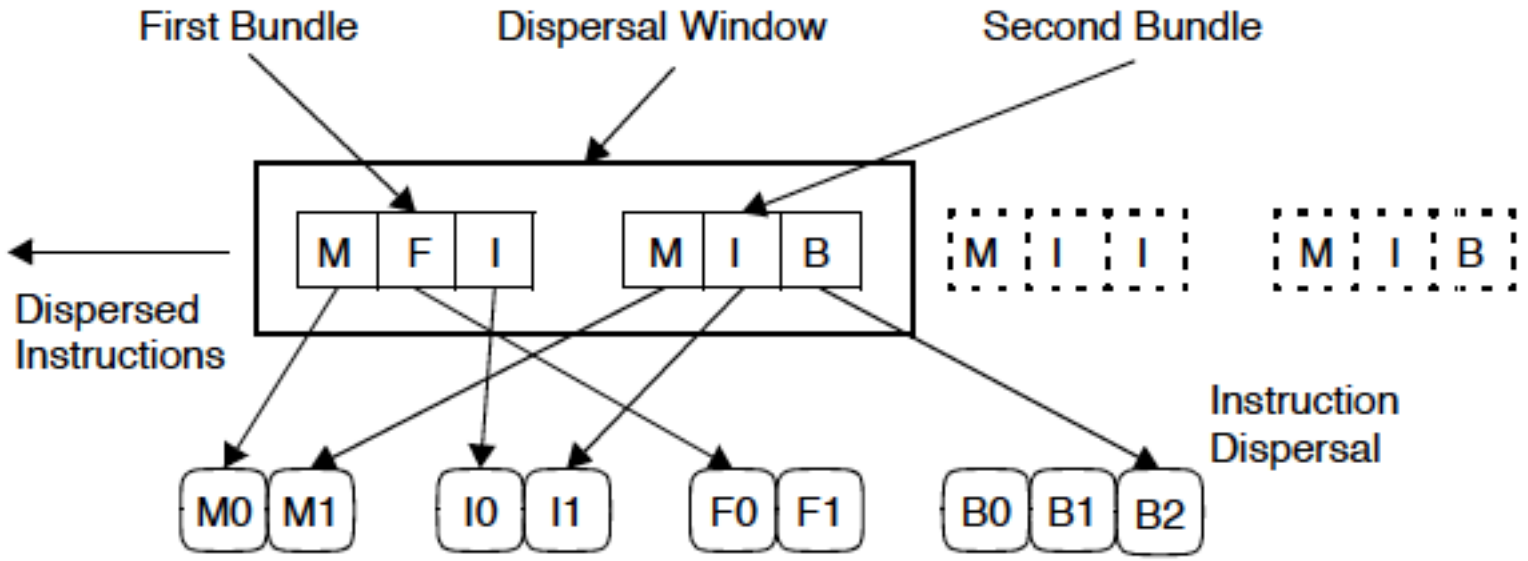
6-b Very Large Instruction Word: exemple de l'Itanium



6-b Very Large Instruction Word: exemple de l'Itanium

Exécution:

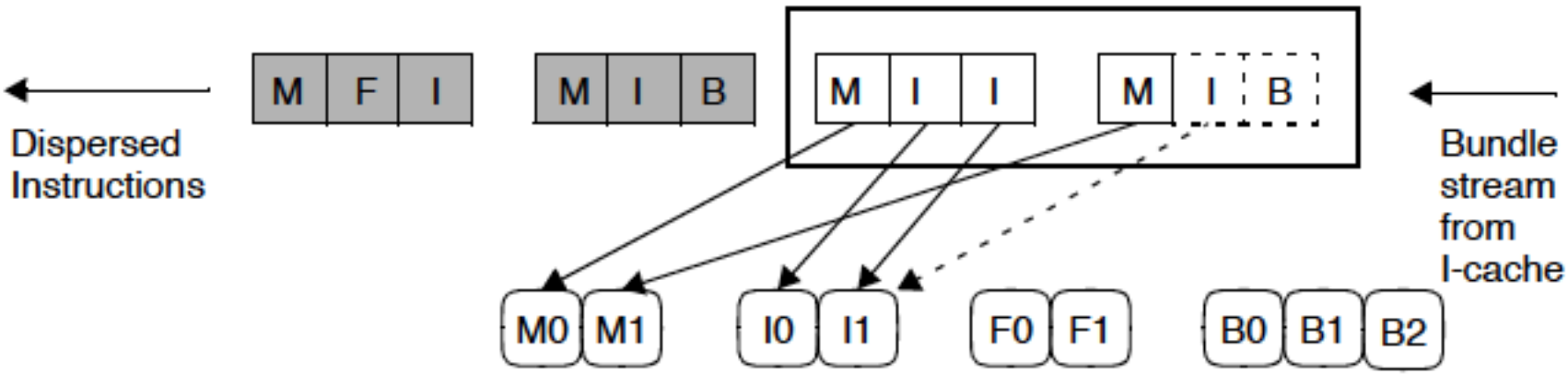
- Les bundles sont lus jusqu'à saturation des unités fonctionnelles
- Au max, 2 bundles exécutés en parallèle (taille max fenêtre d'exécution)



6-b Very Large Instruction Word: exemple de l'Itanium

Exécution

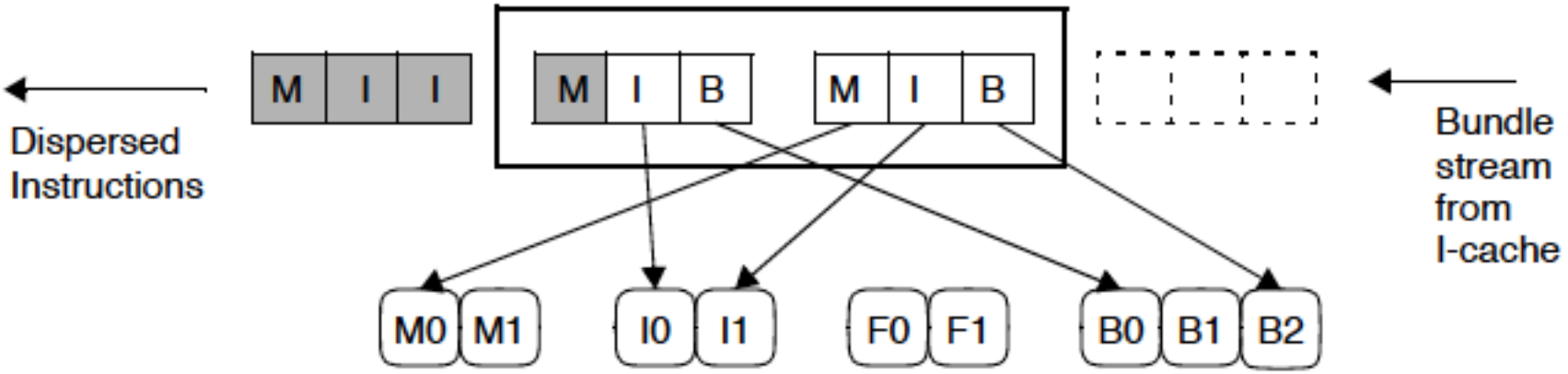
- Tous les cycles: une nouvelle fenêtre
- Limitation du parallélisme:
  - Nombre d'unités fonctionnelles
  - Longueur du groupe d'instructions



6-b Very Large Instruction Word: exemple de l'Itanium

Exécution

- Tous les cycles: une nouvelle fenêtre
- Limitation du parallélisme:
  - Nombre d'unités fonctionnelles
  - Longueur du groupe d'instructions



6-b Very Large Instruction Word: exemple de l'Itanium

Mécanismes pour assurer un bon niveau de parallélisme

Pas les mêmes que le superscalaire, ni OoO ni renommage de registres

Les mécanismes proposés permettent au compilateur d'exprimer du parallélisme

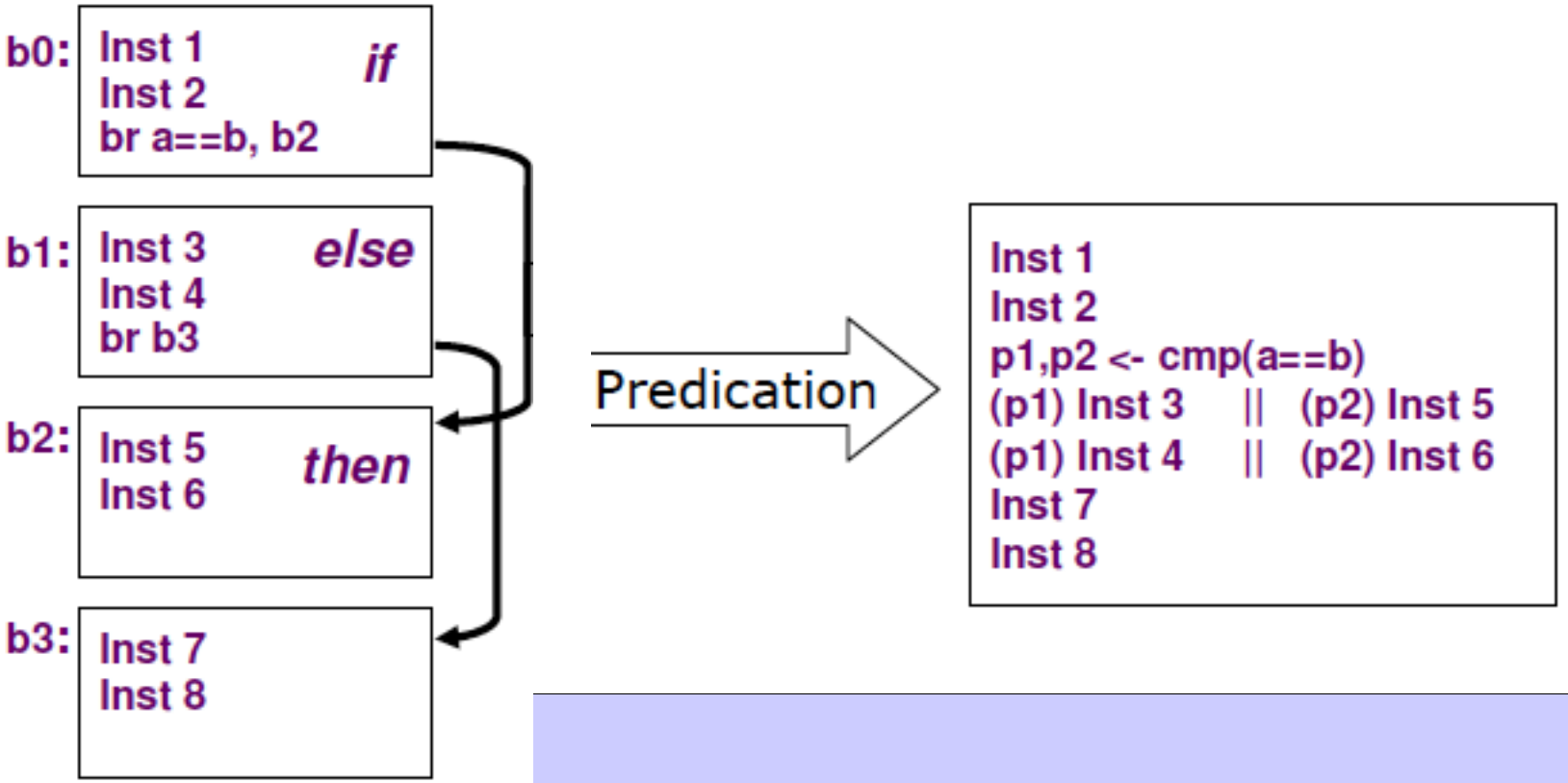
- Problèmes de branchement:
  - Prédiction statique des branchements: instructions différentes de branchement suivant probabilité de faire le branchement
  - Prédication
- Dépendances (entre cases mémoire, entre un load et un store par ex.)
  - Spéculation

D'autres mécanismes complètent la panoplie: registres tournants, grand nombre de registres, ...

6-b Very Large Instruction Word: Prédication sur Itanium

Principe:

- Remplacer les branchements par instructions conditionnées (prédicatées)
- Les deux branches du if..then..else sont exécutées, en parallèle !



6-b Very Large Instruction Word: Prédication sur Itanium

Prédication

- + Plus de stall dus aux branchements ! Plus de branchement !
- + Augmente le parallélisme, très bien pour un VLIW
- - Beaucoup d'instructions sont des nop au final...
  - Pas rentable pour des if..then..else contenant de longues séquences d'instructions

6-b Very Large Instruction Word: Spéculation

Problème de dépendances:

store (r1), r2 ← stocke la valeur de r2 à l'adresse r1

load r3, (r4) ← lit la valeur à l'adresse r4 et la place dans r3

Possible dépendance si r1 == r4

- Les instructions ne peuvent pas être mises en parallèle, stall...

Spéculation

load.a r3,(r4) ← load spéculatif: on suppose qu'il n'y a pas de dépendance

store (r1), r2 le load peut être fait bien avant le store

...

check.a r4 ← vérifie que la spéculation était légale

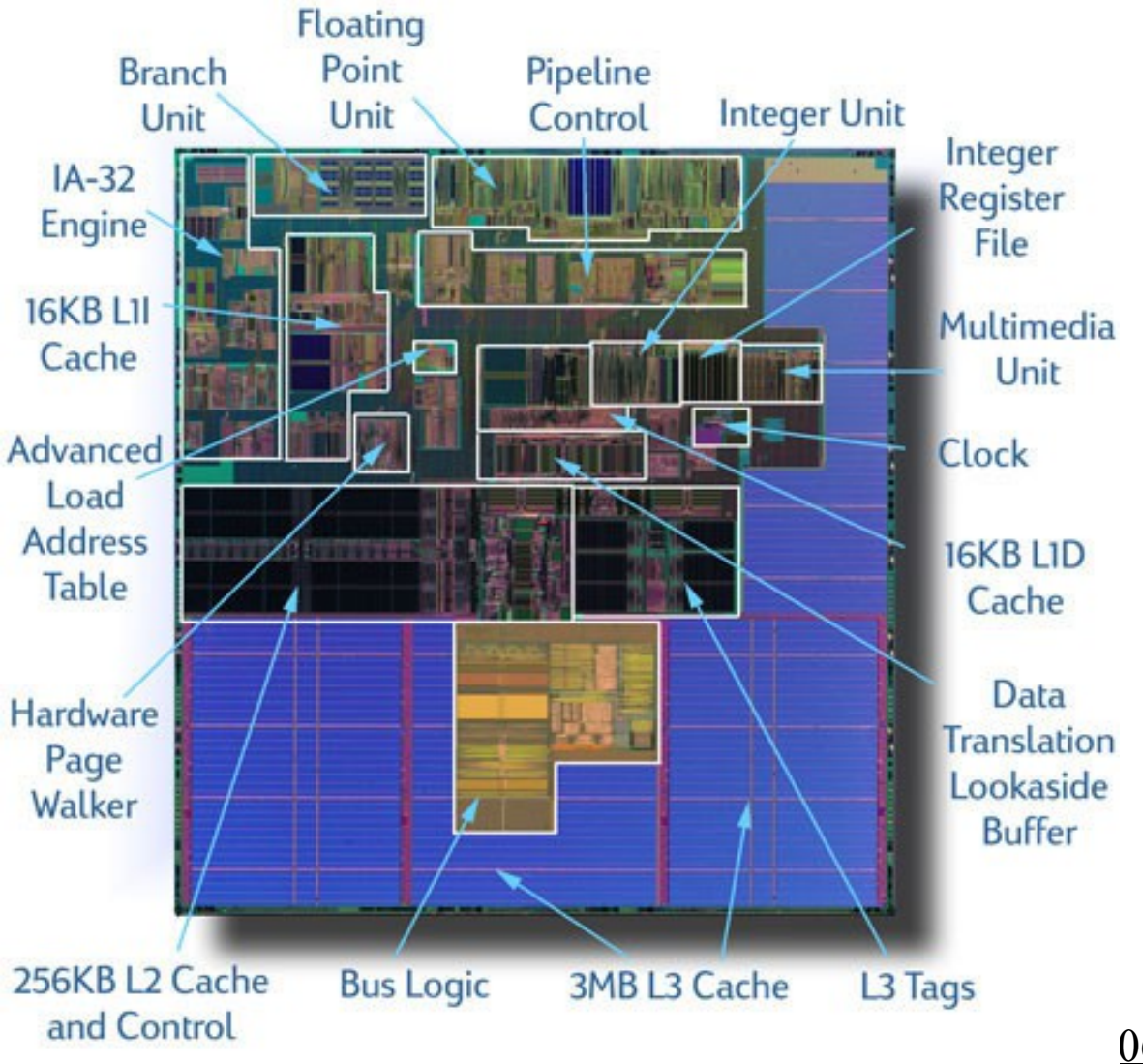
6-b Very Large Instruction Word: Spéculation

Spéculation

- \+ Permet d'exprimer plus de parallélisme et diminuer les dépendances couteuses
- \- Si spéculation erronée, exécute code de correction...
- \- Au final, difficulté du compilateur et des optimisations à bien utiliser ce type de spéculation

6-b Very Large Instruction Word: exemple de l'Itanium

Itanium2



6-a/b Parallélisme d'instructions

Conclusion VLIW et superscalaire

Limites au parallélisme d'instruction (ILP)

- Matériel: pour le OoO, nombre de transistors croit exponentiellement avec nombre d'instructions à mettre en parallèles
- Logiciel: **difficile de trouver des séquences de plus de 6 instructions parallèles dans les codes** (même en optimisant)

Gains de performances apportés par l'ILP

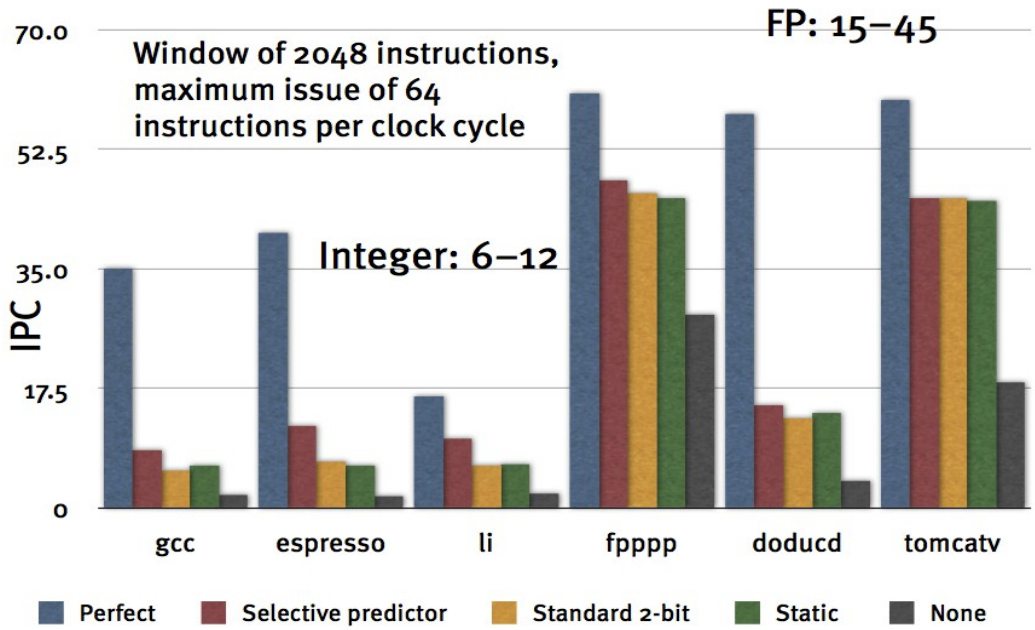
- “Out of order can buy you 5 cycles, not 200 cycles” (D.Levinthal, Intel)
- Gains au max d'un facteur 6, sur des codes très simples
- Ne permet pas de résoudre les longues latences mémoire (cf apres)

6-a/b Limites au parallélisme d'instructions

En simulant une machine avec beaucoup de parallélisme d'instruction (IPC):

- Fenêtre limitée (2048 instructions)
- Prédicteur de branchement
- 64 instructions/cycle max.

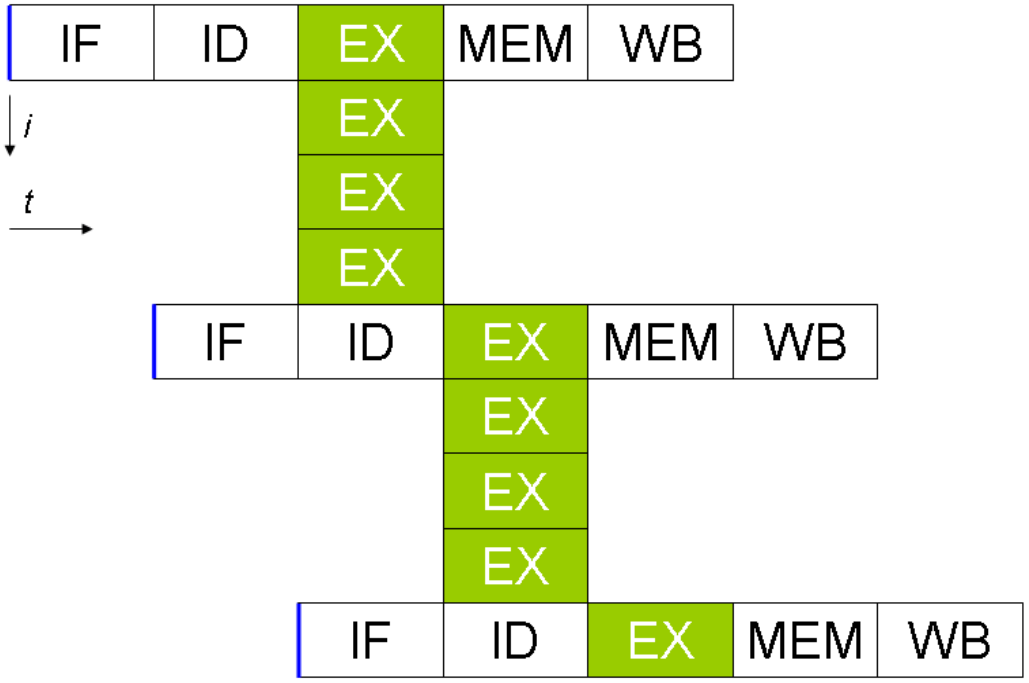
Sur des programmes tests, combien d'instructions en parallèle, en moyenne ?



6-c Vectoriel

Caractéristiques:

- Une instruction (vectorielle), appliquée sur plusieurs données (un vecteur de données)
- SIMD: single instruction, multiple data



6-c Vectoriel

Caractéristiques

- Ne convient pas à tous les codes, toutes les instructions
- Instructions types: multimédia, calcul 3D (cartes graphiques), calcul scientifique, simulations physiques

  Multimédia = marché de masse (rentabilise développements)

- Extensions jeu d'instructions vectorielles pour toutes les architectures
  - Intel: SSE, MMX, AVX (pour Nehalem)
  - IBM: Altivec
  - ARM: Neon
  - Sparc: Visual Instruction Set
- Certains processeurs vectoriels: Cray, GPUs, ...

6-c Vectoriel

Exemple type de code vectorisable

```
For (i=0; i<N; i++)
   A[i] = B[i] + C[i];
```

Toutes les itérations peuvent être faites simultanément.

Addition élément par élément des vecteurs B et C

6-c Vectoriel

Calculs vectoriels et longueur du vecteur

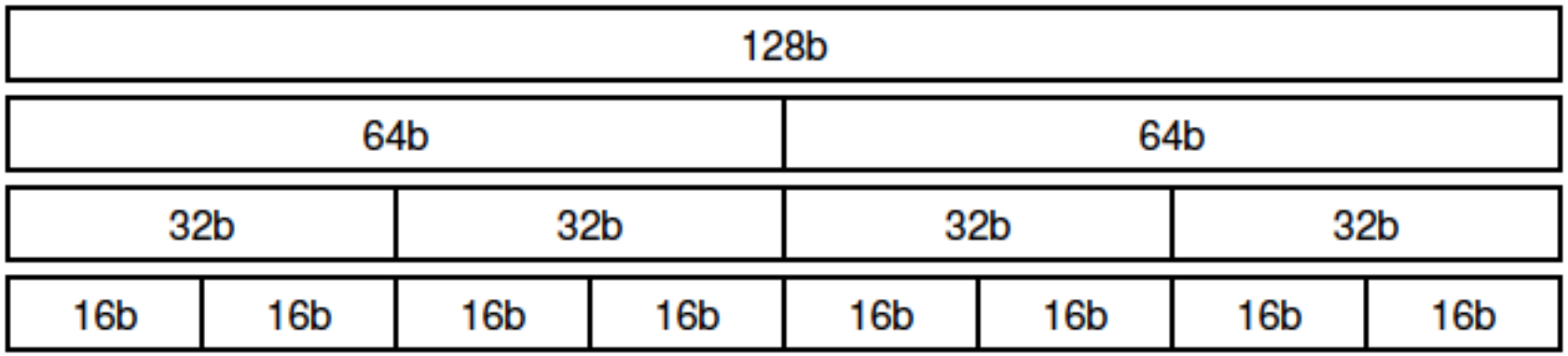
Kernel	Vector length
Matrix transpose/multiply	# vertices at once
DCT (video, communication)	image width
FFT (audio)	256-1024
Motion estimation (video)	image width, iw/16
Gamma correction (video)	image width
Haar transform (media mining)	image width
Median filter (image processing)	image width
Separable convolution (img. proc.)	image width

(from Pradeep Dubey - IBM,
http://www.research.ibm.com/people/p/pradeep/tutor.html)

6-c Vectoriel: exemple SSE

Registre vectoriel (SSE) %xmm

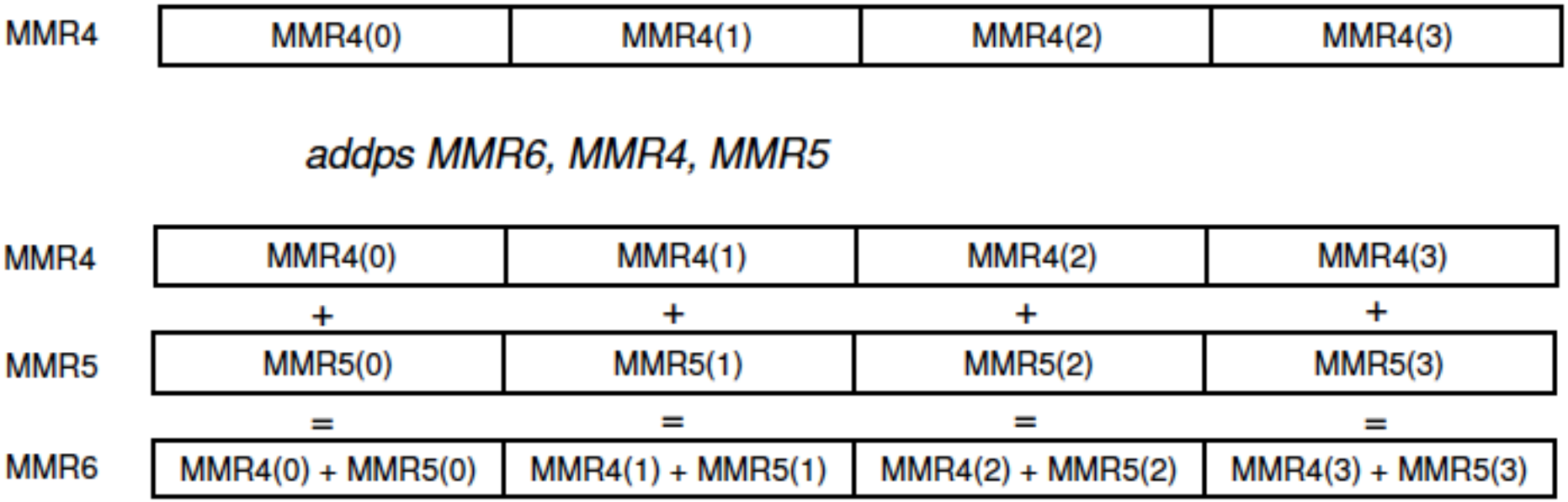
- Peut être vu comme 1 valeur 128 bits, 2 valeurs 64 bits, 4 valeur 32 bits...
- Longueur des valeurs spécifié par l'instruction
- La longueur du vecteur est **toujours** de 128 bits
- Accès mémoire en 1 instruction (en load et store)



6-c Vectoriel: exemple SSE

Exemple type d'instruction vectorielle (SSE)

- Registres vectoriel: contient différentes valeurs indépendantes (ici, 4)
- Les opérations sont les mêmes sur les 4 valeurs

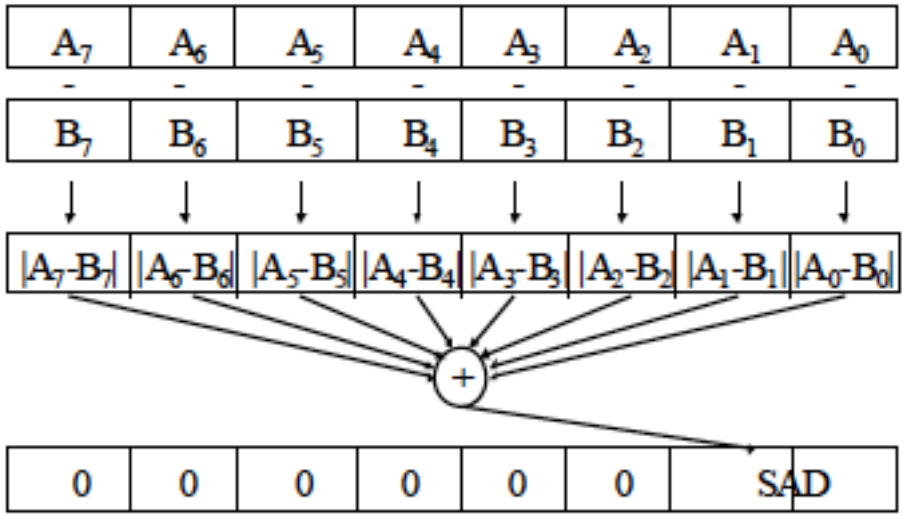


6-c Vectoriel: exemple Itanium

Exemple d'instruction plus compliqué: psad

for (i=0; i<7; i++) s+=abs(A[i]-B[i]);

- Calcule $\sum |Ai-Bi|$
- Aucune chance pour le compilateur de la sélectionner !



6-c Vectoriel

Jeu d'instruction vectoriel

- Actuellement, sur petits registres (128, 256 bits). Avant, sur Cray, registres très longs.
- Jeux d'instructions très riche (1400 pages juste pour décrire SSE)
- Utilisation des instructions vectorielles
  - Responsabilité compilateur: nécessite vectoriseur. Difficile prise en compte de la richesse des instructions
  - Responsabilité utilisateur: codage en utilisant l'assembleur, ou les fonctions intrinsèques (correspond à une instruction asm)

Le hardware ne vectorise pas !

6- Conclusion parallélisme au niveau des instructions

Parallélisme d'instructions:

- Extraction automatique: architectures superscalaires
  - Hardware très complexe
  - Mécanismes architecturaux possibles: prédiction branchement, OoO, renommage registres
- Expression par le compilateur: architectures VLIW
  - Besoin de compilateur optimisants
  - Mécanismes architecturaux possibles: spéculation, prédication

Parallélisme de données, architectures vectorielles

- Des petits vecteurs dans toutes les architectures. Necessite compilateur vectorisant

6- Cartographie du parallélisme au niveau des instructions

- A noter que certaines architectures sont multicoeurs ou ont un support pour le multithread

Superscalaires	VLIW	Scalaires et SIMD
MIPS R16000 (2004)	Intel Itanium Tukwila (2008). Partiellement OoO pour accès mémoire	Intel ATOM (2007)
Intel Nehalem (2009)	PNX1005, Trimedia NXP (2009)	Sun T2 (2006)
Sun UltraSparc IV	ST 231, ST Microelectronics	ARMv6
ARMv7A (Cortex A8)	TMS 320DSP, Texas Instrument	Puces GPU