Architectures des processeurs

Support de cours EII 3

D.Chillet

Daniel.Chillet@enssat.fr

http://r2d2.enssat.fr

Plan

- ***Introduction***

  - *rappel processeurs CISC / RISC :*
    - *évolution des machines*
    - *concept de processeurs Risc*

  - *rappel pipeline des processeurs :*
    - *fonctionnement d'un pipeline*
    - *problèmes engendrés par le pipeline du contrôleur*

Plan

- ***Mécanismes permettant d'augmenter les performances des processeurs :***
  - *les tendances en termes d'évolution des processeurs :*
    - *allongement du pipeline*
    - *augmentation du nombre d'unités fonctionnelles*
    - *techniques avancées (renommage de registres, exécution dans le désordre, multithreading, etc etc)*

Plan

- ***Panorama de quelques processeurs :***
  - *itanium : processeur IA64*
  - *power PC*
  - *PTS : processeurs de traitement du signal*
  - *cœur de processeurs*
  - *etc*

Rappel :

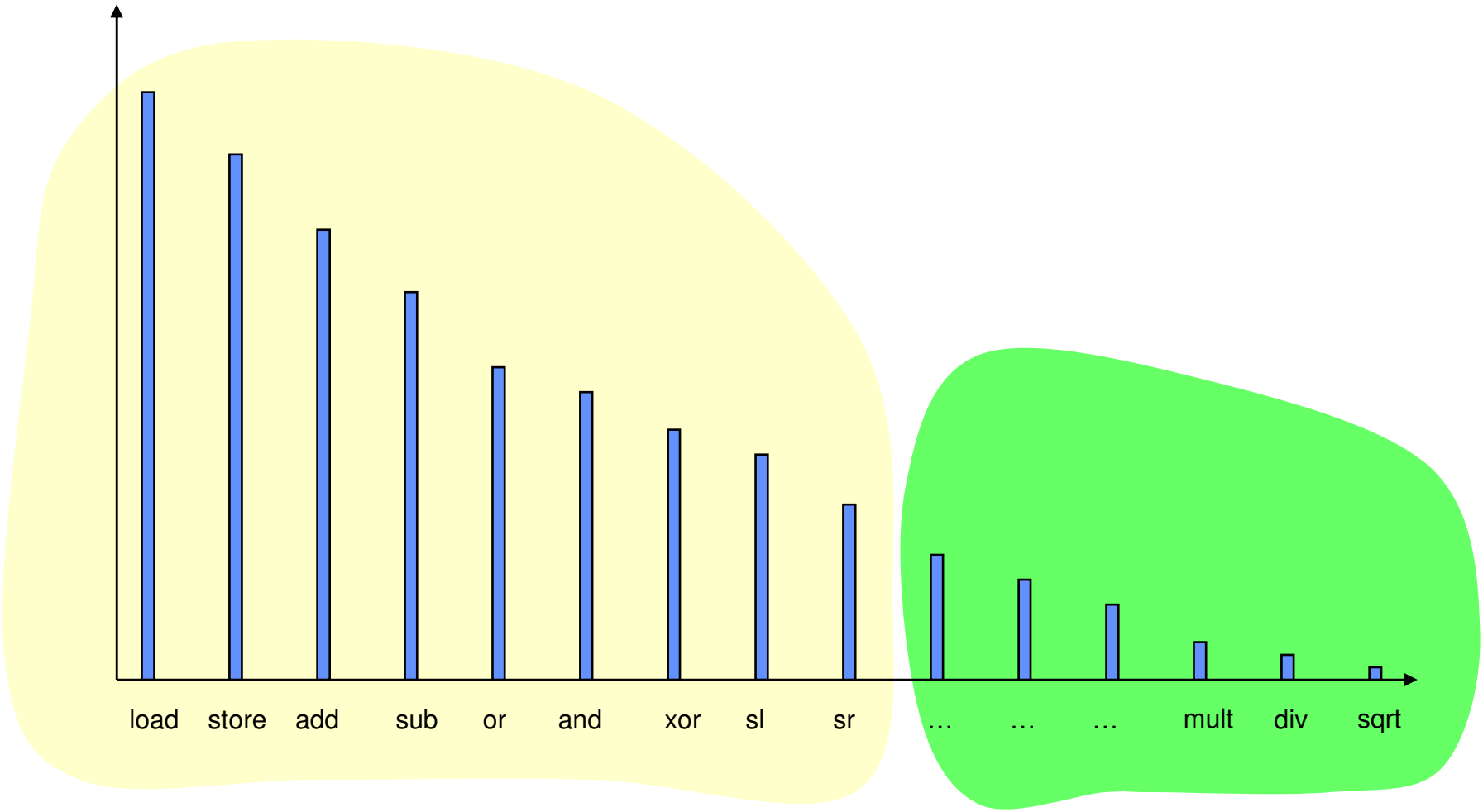
processeurs CISC RISC

Rappel : processeurs CISC

- Evolution des processeurs CISC :
  - tendance a aller vers un langage machine de haut niveau (==> C) :
    - instructions de plus en plus nombreuses
    - instructions de plus en plus complexes
    - difficulté pour décoder les instructions (tailles variables)
    - augmentation de la complexité de la logique de contrôle
    - limite l'augmentation de la fréquence d'horloge
    - contrôle très couteux : limite la place pour le reste :
      - taille des caches primaire, secondaire
      - nombre de registres limité
      - nombre d'unités de calcul limité

Rappel : processeurs CISC

– occurrence d'apparition des instructions



Rappel : processeurs CISC

- Observations : par analyse de code
  - *on passe 90 % du temps à exécuter 10 % du jeu d'instructions !*
  - les instructions les moins couramment utilisées sont celles qui engendrent le plus de contrôle :
    - adressages complexes (pré-post incrémenté, basé, indexé, ...)
    - tous les calculs peuvent s'exécuter avec des opérandes en mémoire (via tous les adressages)
    - latences des instructions très variables
    - difficulté à pipeliner l'exécution
    - jeux d'instructions non orthogonals
    - jeux d'instructions mal utilisés par les compilateurs

Rappel : processeurs RISC

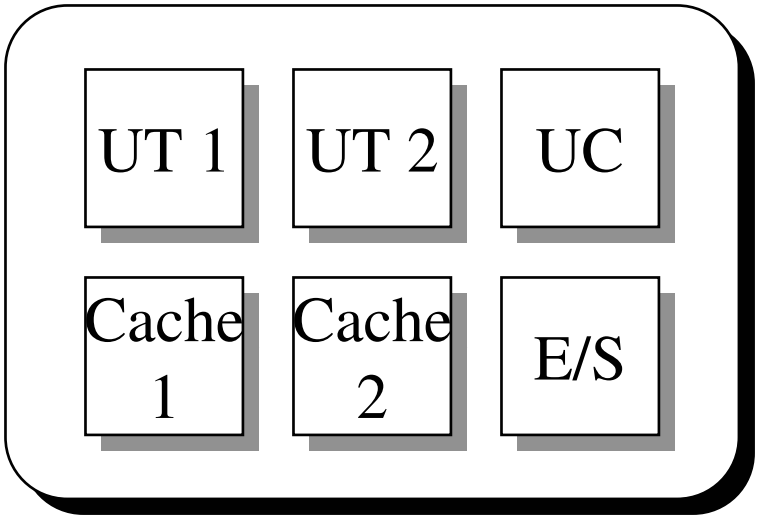
- réduire le jeu d'instructions aux instructions les plus couramment utilisées :
  - instructions simples :
    - faciles à décoder
    - taille fixe
    - latence d'exécution tendant vers 1 temps de cycle
    - facile à pipeliner, **contrôle simple,** processeur superpipeline
    - gain en surface, qui permet alors :
      - d'augmenter le nombre de registres (architectures load/store)
      - d'augmenter la taille des caches
      - de placer un ou deux caches primaire (mixte ou séparés), puis un cache secondaire en interne au processeur et gestion du 3 ième interne
      - d'augmenter le nombre d'unités de calcul, processeur superscalaire
      - d'augmenter la fréquence d'horloge

Rappel : processeurs RISC

- Les tendances des processeurs RISC :
  - pipelines profonds :
    - jusqu'à 20 étages pour le Pentium IV
  - superscalaire :
    - unités fonctionnelles indépendantes
    - unité de *disptaching*
    - jusqu'à 6 instructions lancées par cycle pour le R10000
  - fréquence d'horloge élevée :
    - jusqu'à 3 Ghz pour les prochaines versions du pentium
  - architectures 32 ou 64 bits
  - jeux d'instructions étendu au multi média

Rappel : processeurs RISC

- Les tendances des processeurs RISC : suite
  - hiérarchie mémoire complexe :
    - bus de données entre cache et buffer large
    - premier et second niveau de cache intégrés (on chip)
  - multi processeurs on chip :
    - décomposition du processeur en plusieurs chip
    - mais packaging dans un même boitier



Rappel : processeurs RISC

- Les problèmes liés à cette évolution :
  - parallélisme limité à quelques instructions par cycle
  - le nombre de ports sur le banc de registres est critique :
    - le pipeline engendre des accès simultané sur les mêmes ressources
    - goulot d'étranglement
  - gestion des ressources :
    - table d'occupation des ressources
    - dispatching des instructions
    - maintien de la cohérence du code (exécution dans le désordre, réordonnancement)
  - désamorçage des pipelines notamment pour les instructions de branchement

Rappel : processeurs RISC

- Mécanismes particuliers :
  - exécution dans le désordre : (ou exécution spéculative)
    - ***Objectif*** : ne pas désamorcer les pipes des différentes unités
    - exécution jusqu'au cycle d'écriture du résultat
    - exécution dès que possible :
      - disponibilité des opérandes et des unités fonctionnelles
    - mise en place de buffer d'instructions en entrée des pipelines des différentes unités et fonctionnement indépendant de ces unités
    - gestion dynamique du droit de lancer des instructions dans le désordre, gestion d'une table de ressources
    - il faut disposer d'un buffer d'instructions important (pour pouvoir piocher les instructions à exécuter à chaque instant)

Rappel : processeurs RISC

- Mécanismes particuliers :
  - renommage des registres :
    - il y a plus de registres physiques que de registres logiques
    - permet de casser les fausses dépendances entre des instructions successives
    - permet de paralléliser du code apparemment séquentiel
    - gestion dynamique du renommage

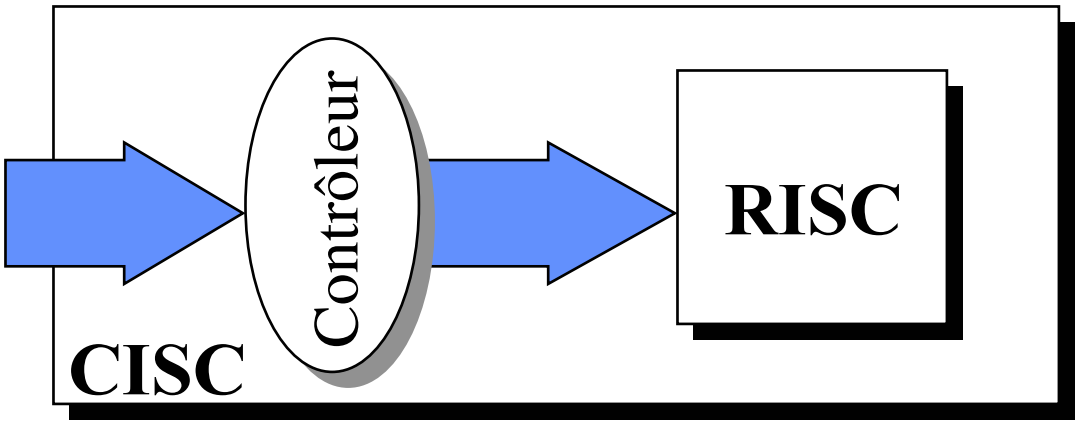
Rappel : processeurs RISC

– prédictions de branchement :

- **objectif** : éviter au mieux de désamorcer le pipeline contrôleur
- les applications comportent entre 15 et 30 % d'instructions de branchement, la décomposition est la suivante :
  - 1 tiers sont des instructions de branchement inconditionnel
  - 1 tiers sont des boucles (généralement des branchements arrière), facilement prédictible
  - 1 tiers sont des instructions de branchement conditionnel
- Solutions :
  - Prédécodage des instructions de branchement conditionnel au plus tôt : durant le cycle de lecture, prédiction au plus tôt
  - prédiciton statique : branchement arrière pris (intéressant pour boucles)
  - prédiction dynamique à 1 bit (table de branchements)
  - prédiction dynamique à 2 bits (algorithme de Smith)
  - prédiction dynamique hybride (dynamique avec 2 algorithmes)

Rappel : les CISC à coeur RISC

- Qu'est ce que c'est ?
  - le coeur du processeur travaille sur des instructions simples, réduites, de taille constante, avec peu d'adressage
  - le jeu d'instructions à l'apparence d'un jeu CISC
  - le contrôleur décompose les instructions très complexes en suites d'instructions RISC

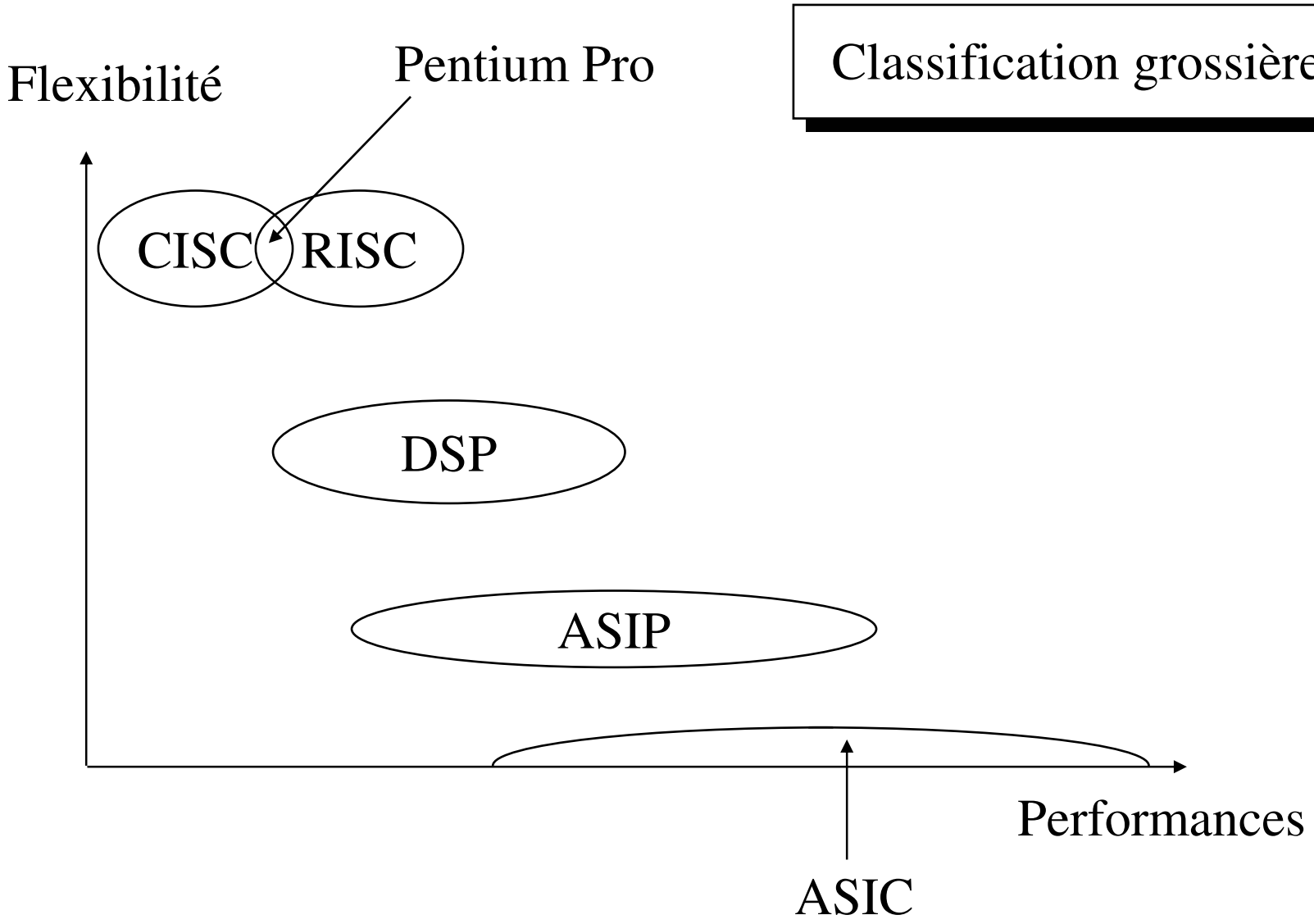


Cas du pentium

Rappel : les CISC à coeur RISC

- Intérêts et motivations :
  - permet de conserver la compatibilité ascendante du jeu d'instructions des générations de processeurs (notamment entre les processeurs 80xxx, Pentium)
  - coeur du processeur plus facilement pipelinable
  - contrôle moins couteux gain de surface utilisable pour :
    - plus de registres
    - cache plus important
    - mise en place d'un second (voir d'un troisième) niveau de cache

Rappel : processeurs RISC --- DSP ?



Rappel : processeurs RISC --- DSP ?

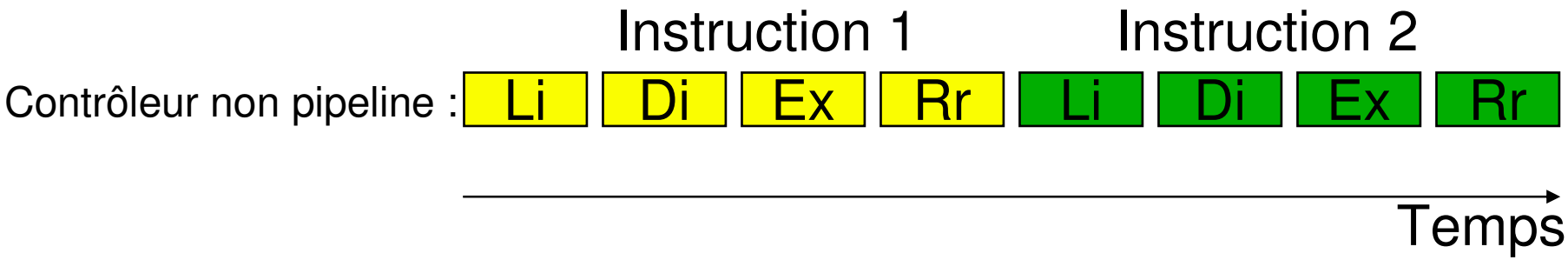
- Avantages et inconvénients du DSP :
  - prise en main **délicate**
  - chaine de développement **lourde**, développement croisé
  - retouche du code assembleur à la main pour les parties critiques (développement en C)
  - dans l'industrie, 80 % du code DSP est écrit en assembleur
  - gestion temps réel **précise** mais **délicate**
    - le temps réel correspond à la faculté d'assurer un temps de réponse maximal lors de l'arrivée d'un évènement (exemple : signal d'alarme)

Rappel :

Pipeline de processeurs

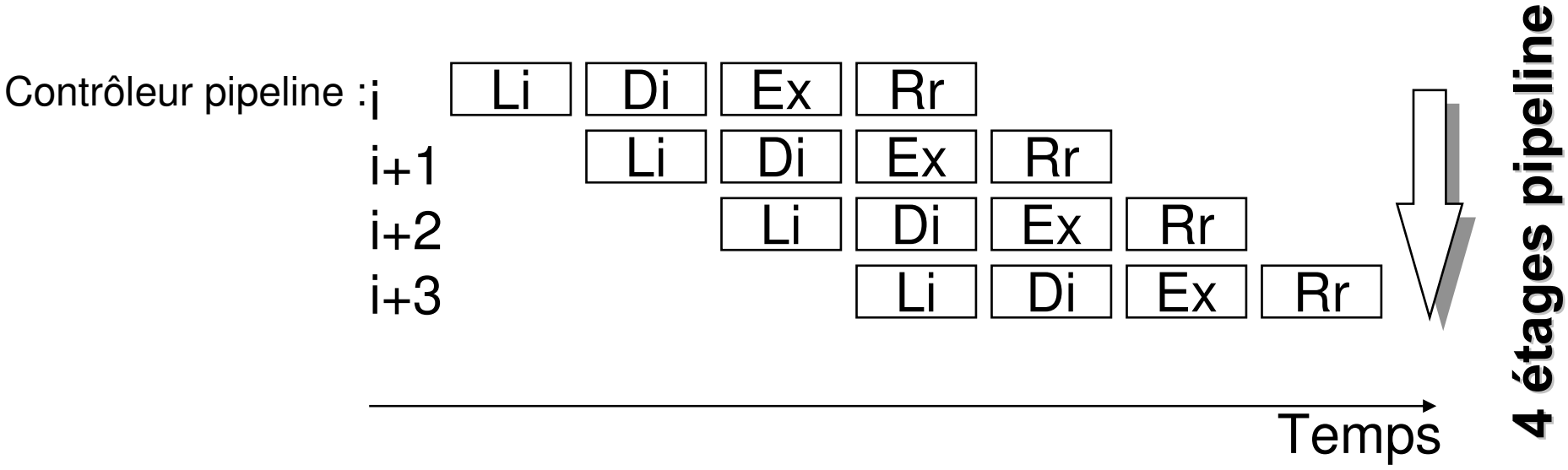
Rappel Pipeline des processeurs

- Pipelinage d'un contrôleur :
  - le cycle Von Neumann :
    - Li : lecture instruction
    - Di : décodage instruction
    - Ex : exécution d'instruction
    - Rr : rangement du résultat



Rappel Pipeline des processeurs

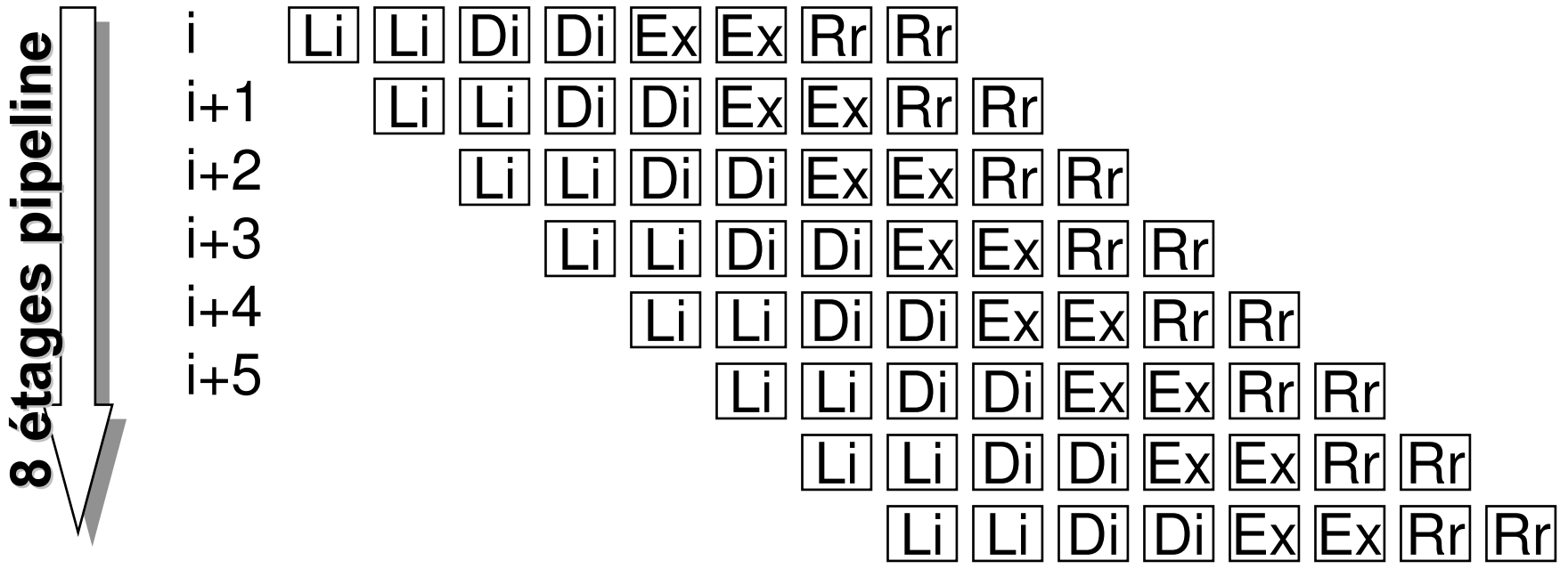
- Pipelinage d'un contrôleur, suite :



- le processeur consomme 1 instruction par cycle
- gain = *4
- fréquence d'horloge multipliée par 4

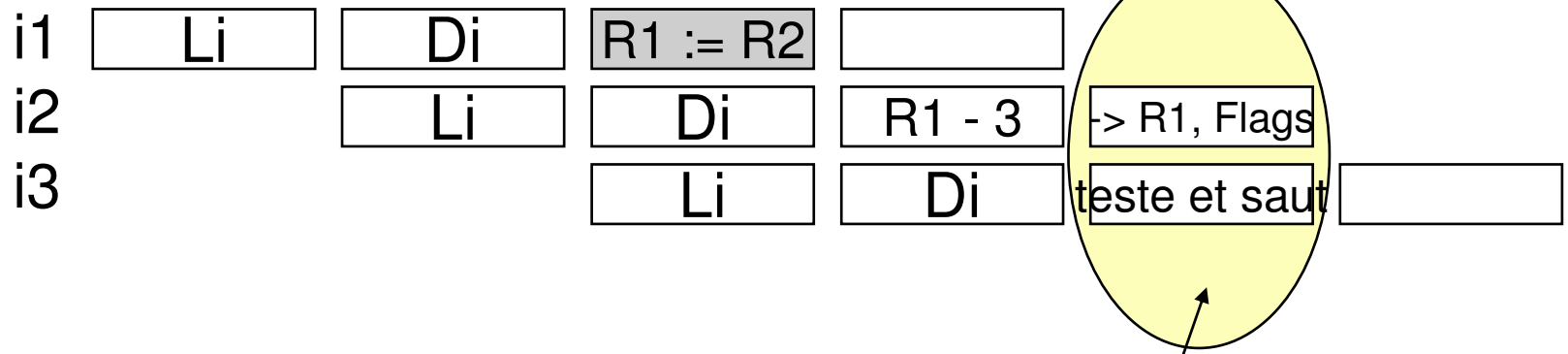
Rappel Pipeline des processeurs

- On fractionnant encore les tâches en sous tâches,
  1) on diminue le temps de cycle,
  2) on augmente le nombre d'étages de l'architecture



Rappel Pipeline des processeurs

- Enchaînement du cycle Von Neumann :
  - soient les instructions :
    - i1 : move R1, R2 R1 := R2
    - i2 : sub R1, 3 R1 := R1 -3
    - i3 : beq @adresse si R1 = 0 alors saut

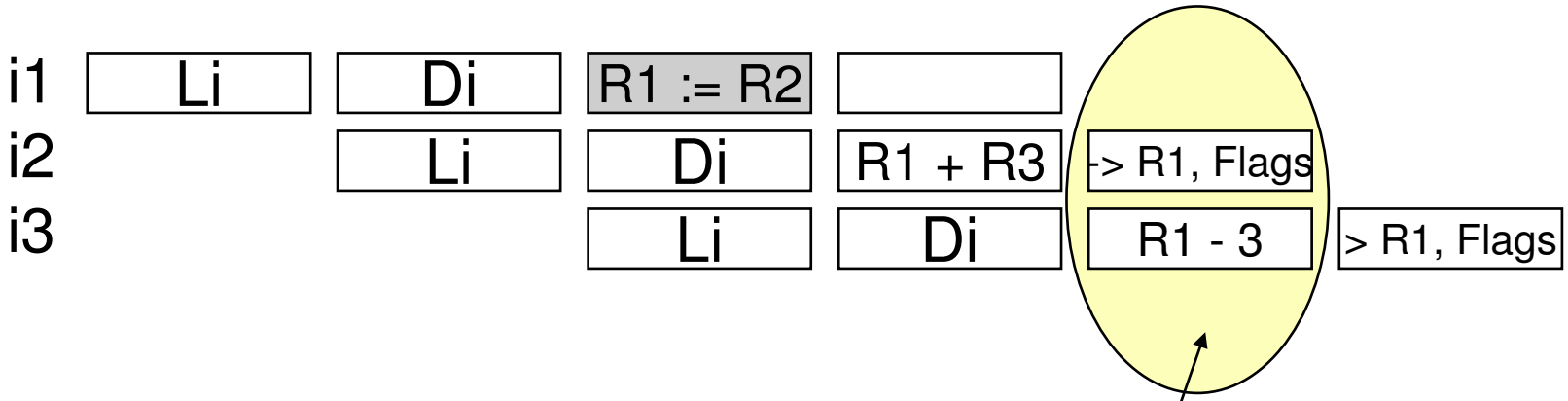


Les flags ne sont pas encore valides lorsqu'on les teste

Dépendance entre les instructions

Rappel Pipeline des processeurs

- Enchaînement du cycle Von Neumann :
  - soient les instructions :
    - i1 : move R1, R2 R1 := R2
    - i2 : add R1,R3 R1 := R1 + R3
    - i3 : sub R1, 3 R1 := R1 -3

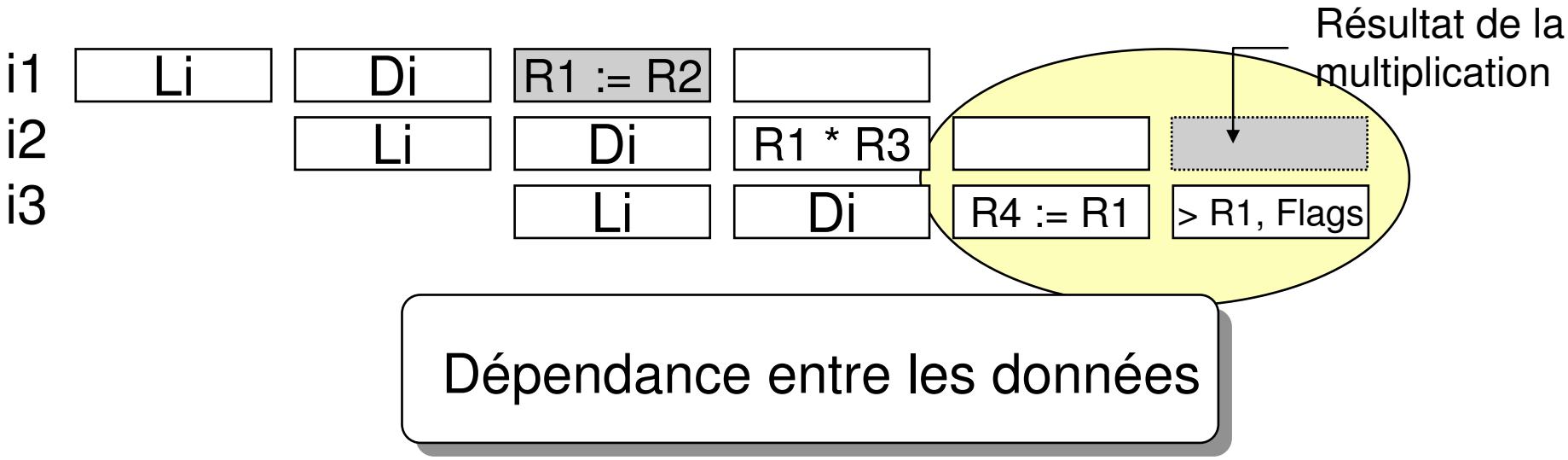


Le registre R1 n'est encore chargé

Dépendance entre les données

Rappel Pipeline des processeurs

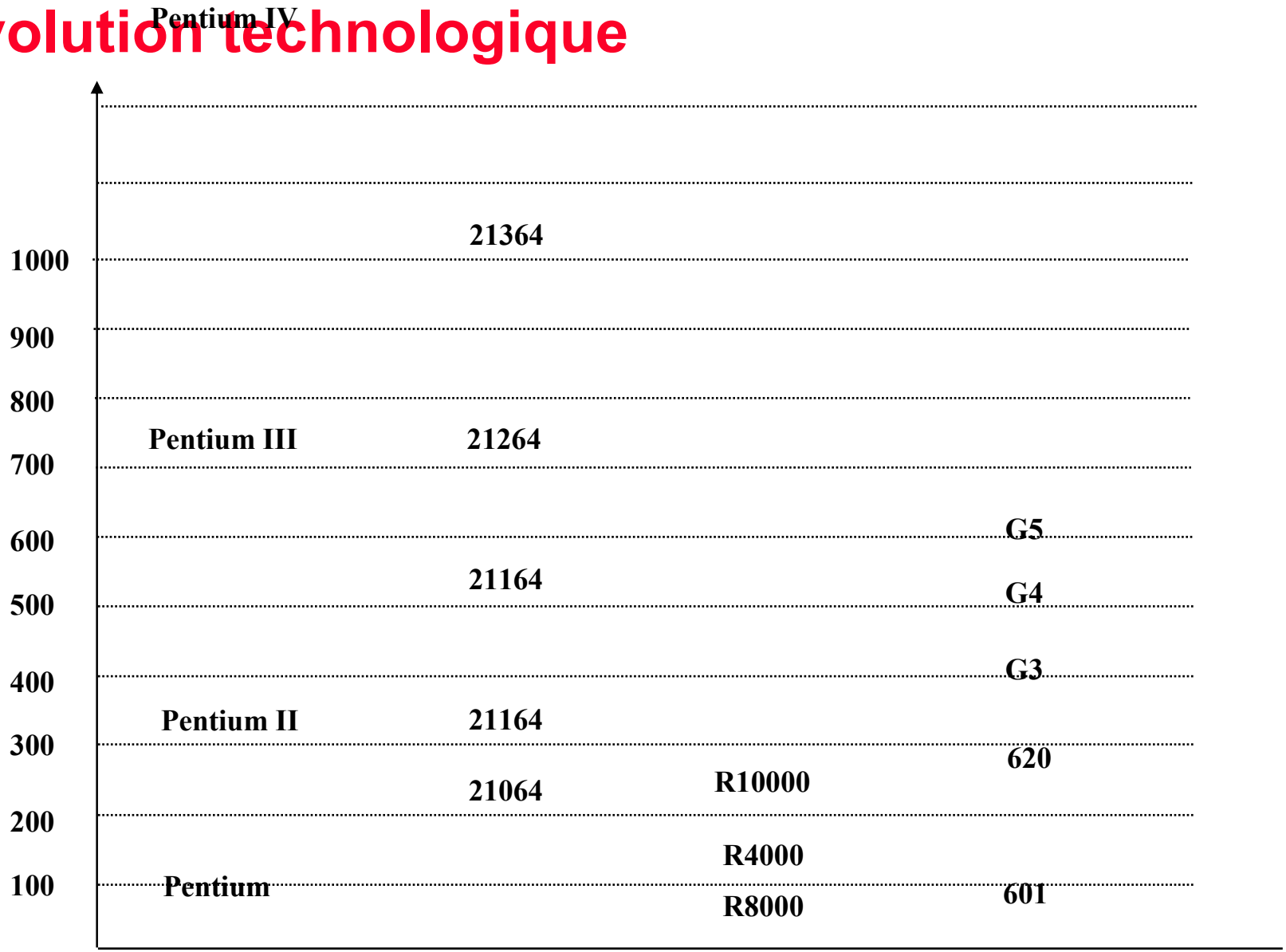
- Enchaînement du cycle Von Neumann :
  - soient les instructions :
    - i1 : move R1, R2 R1 := R2
    - i2 : mult R1,R3 R1 := R1 * R3
    - i3 : move R4, R1 R4 := R1
  - avec l'opération multiplication réalisée sur un opérateur pipeline à 2 étages



AUGMENTATION DE PERFORMANCES

Augmentation de performances

Evolution technologique



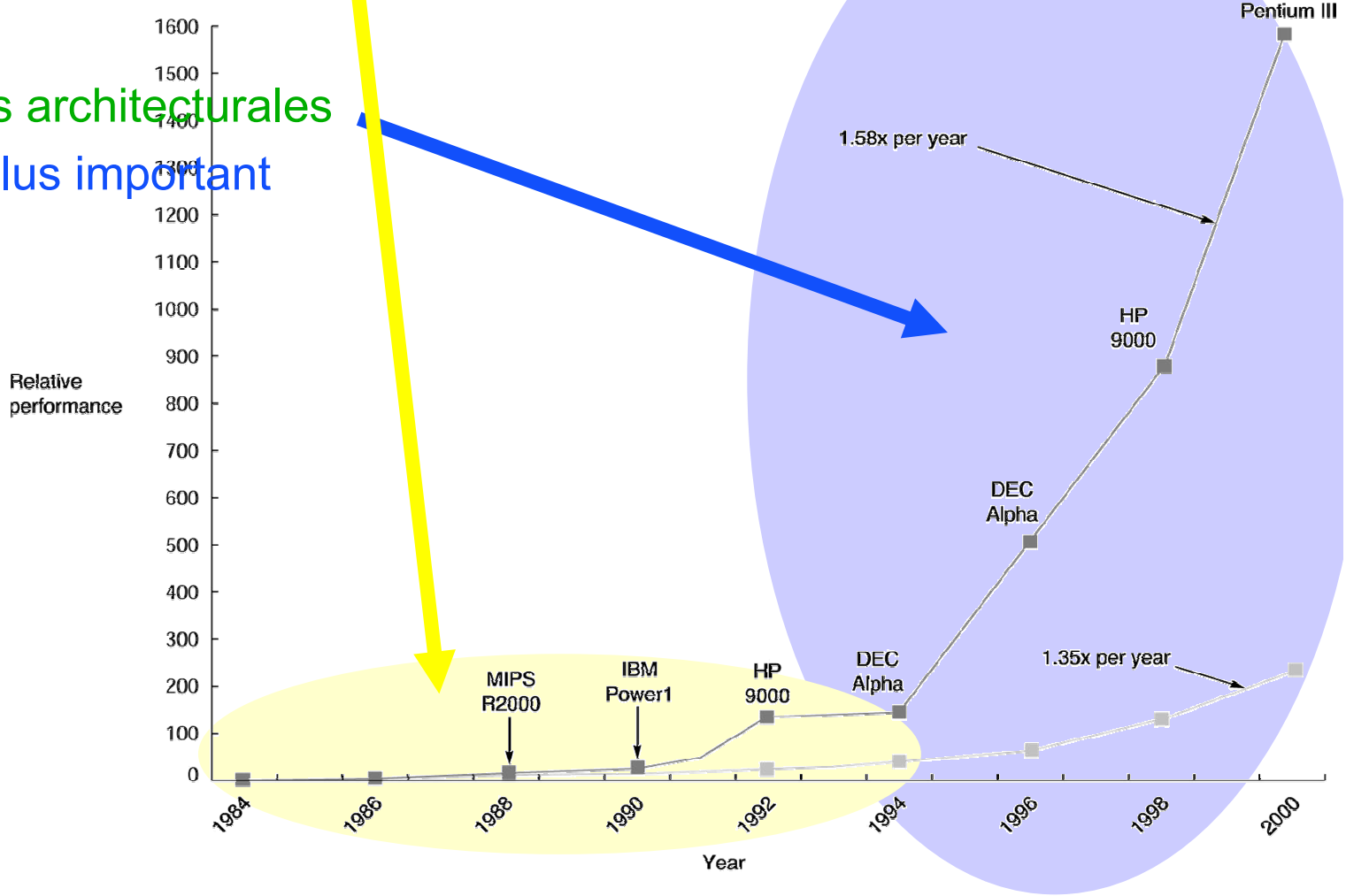
Augmentation de performances

❑ La course à la fréquence est finie ??

- ➢ Lors d'une conférence organisée en partie par Gartner et à laquelle près de 6500 professionnels de l'informatique assistaient, Craig Barrett l'actuel CEO d'Intel, a publiquement demandé pardon genou à terre, pour **ne pas être en mesure de lancer un processeur Pentium à 4 GHz** en déclarant qu'il aurait préféré être en mesure de tenir sa promesse.
- ➢ Barrett de souligner qu'Intel préfère dorénavant **se concentrer** non pas sur la fréquence pure mais sur les **fonctionnalités des processeurs pour augmenter leurs performances**.
- ➢ Octobre 2004

Augmentation de performances

- la techno ne fait pas tout :
  - gain du à la techno : 35% par an
- avancées architecturales
  - gain plus important



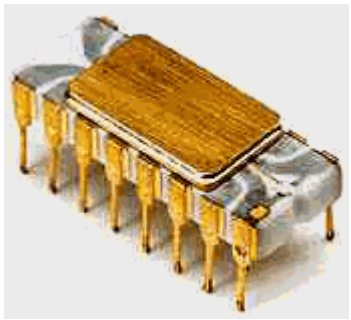
Augmentation de performances

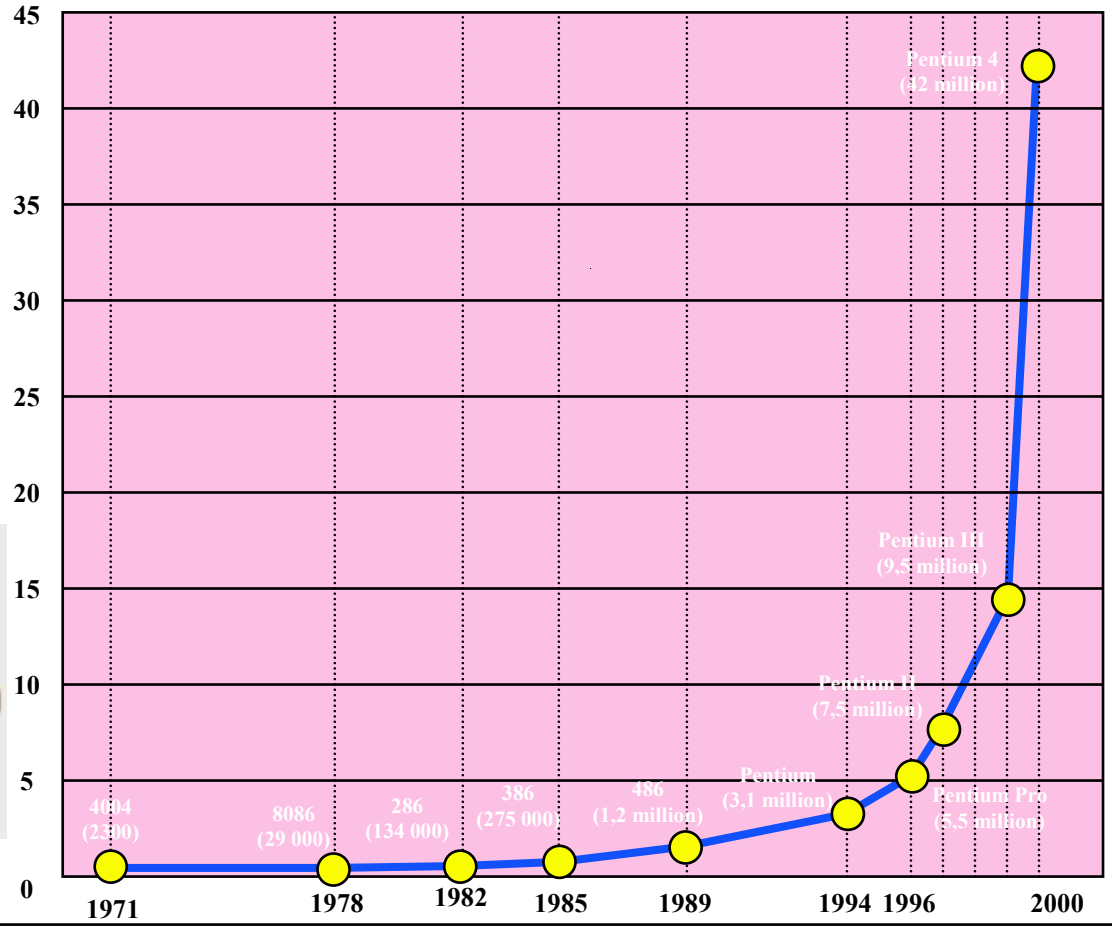
➢obtenue par :

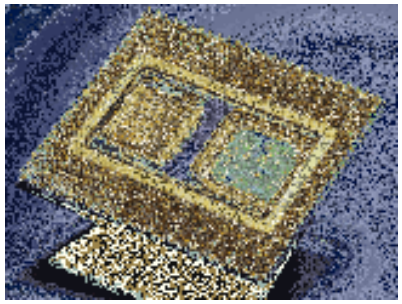
- ♦ l'apport technologique (1 µ m, 0.8 µ m, 0.5 µ m, 0.35 µ m, ...)
- ♦ le pipelinage :
  - – du contrôleur
  - – inter unités fonctionnelles
  - – intra unité fonctionnelle
- ♦ des fréquences élevées peuvent conduire à réduire la complexité du contrôleur :
  - – cas du 21164 dans lequel il n'existe pas d'exécution dans le désordre des instructions
- ♦ nécessite la mise en place de mémoires caches coûteuse :
  - – ayant le même temps de cycle que le processeur et placées dans le même boîtier que le processeur (cache interne)

Augmentation de performances

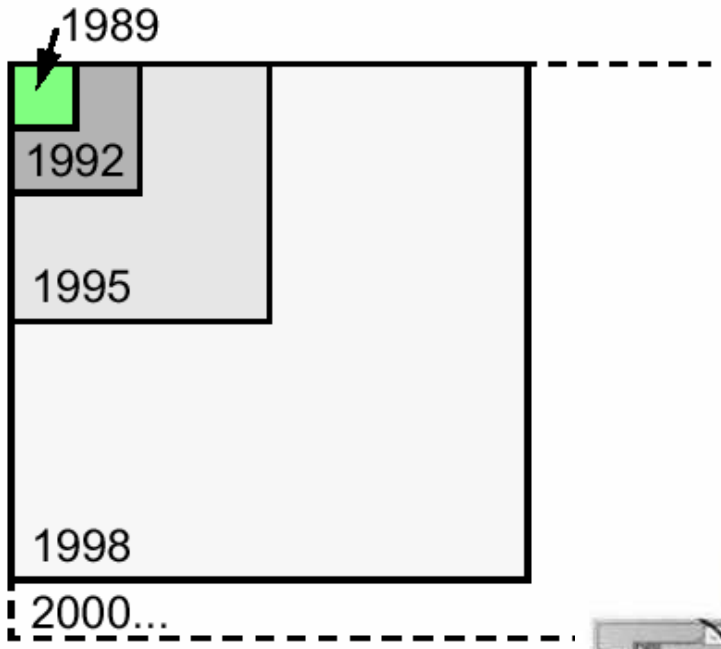
➢Augmentation du nombre de transistors par puces

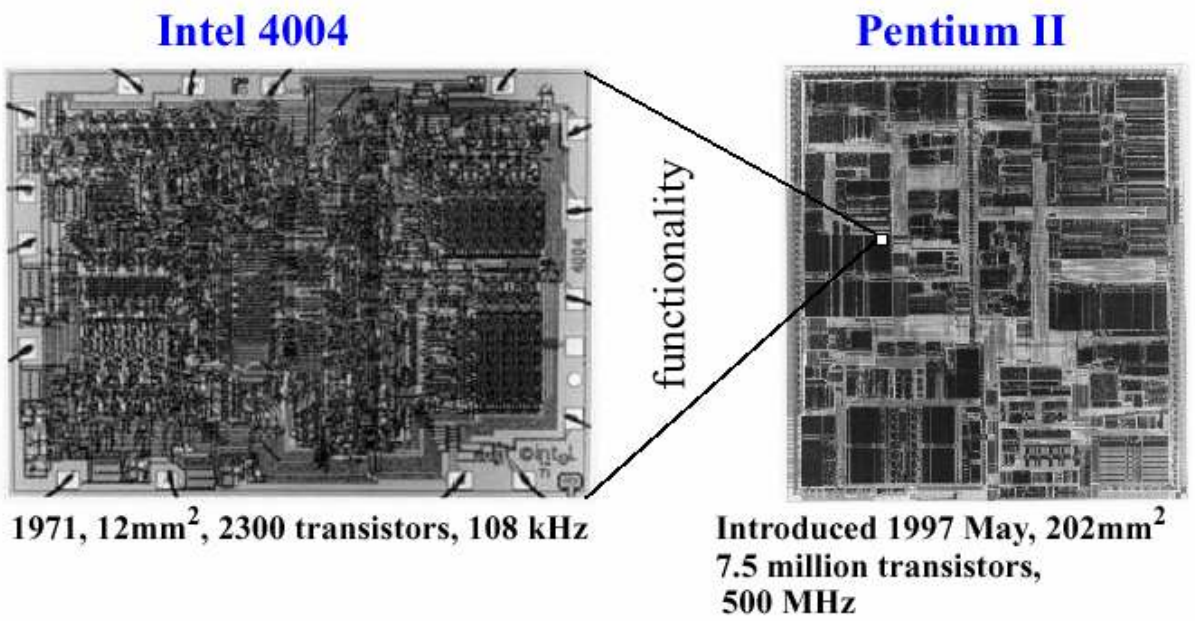






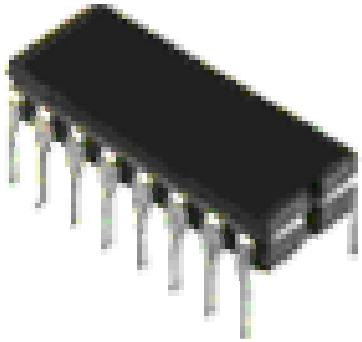
Augmentation de performances

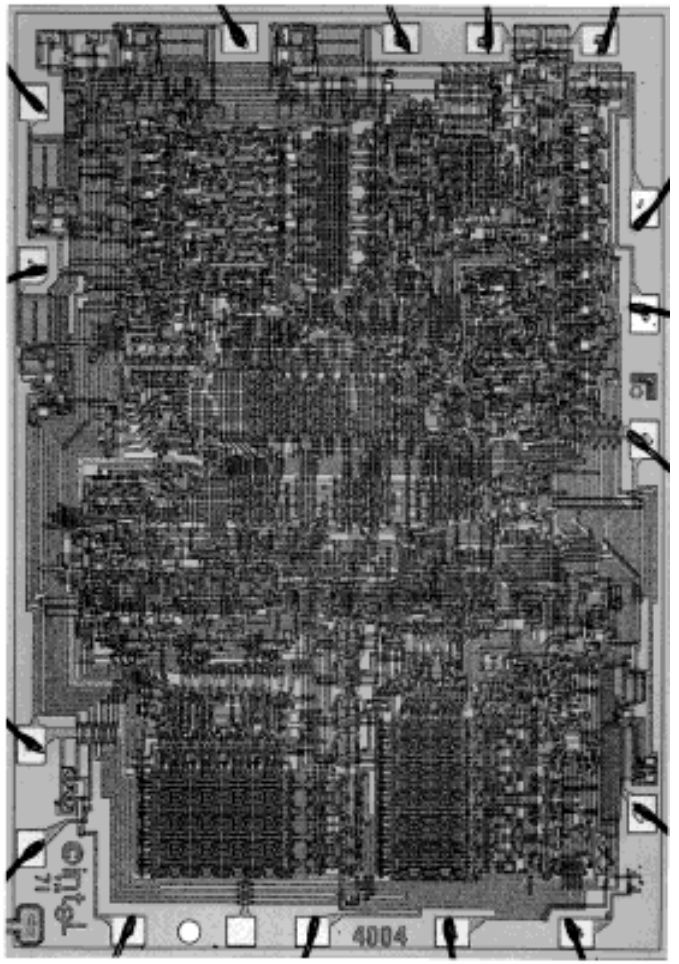




Augmentation de performances

- **Intel 4004**
  - ➢1971
  - ➢108 kHz
  - ➢4 bits
  - ➢1200 FF
  - ➢0,06 MOPS
  - ➢10 microns
  - ➢2300 transistors



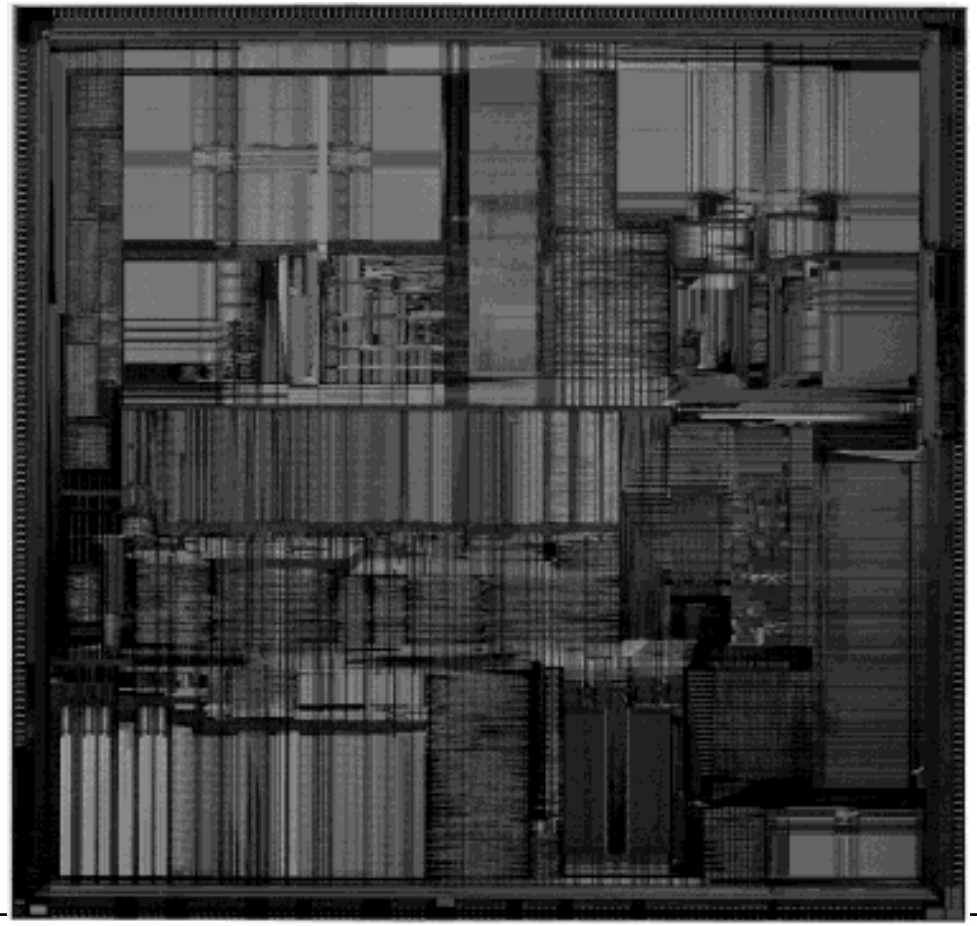


Augmentation de performances

- 1970 Mémoire 4Kbits MOS
- 1972 1er processeur : 4004 (Intel), techno. NMOS
- 1977 16K DRAM et 4K SRAM en production
- 1979 64K DRAM en production
- 1980 Processeur INTEL x86
- 1984 Processeur INTEL 80286 des PC AT
- 1986 1 mégabit DRAM
- 1988 TI/Hitachi 16-megabit DRAM
- 1990 Processeur INTEL 80286 avec fonctions multimédia
- 1990 Wafer de 20cm en production
- 1991 4 mégabit DRAM en production
- 1993 Processeur Pentium
- 1999 Processeur Pentium III
- 2000 Nouvelle architecture Intel, HP : IA 64 (Merced, Itanium)
- 2001 Processeur Pentium IV, 1,7 Ghz
- 2001 Successeur de l 'Itanium, 10 Ghz, 1 milliard de transistors, 100 milliards d 'opérations par seconde

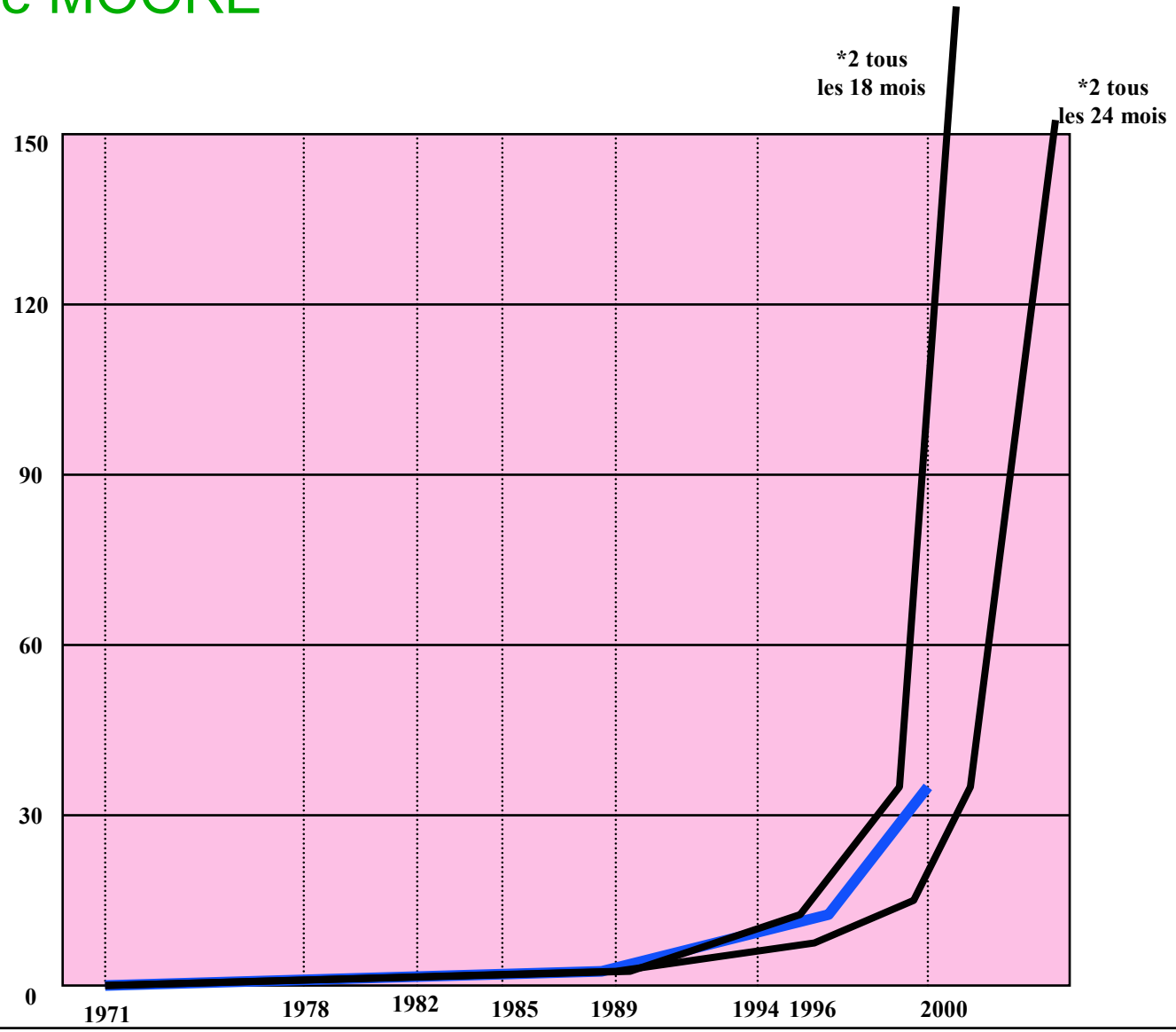
Augmentation de performances

- **INTEL Pentium II :**
  - ➢1996
  - ➢5.5 Millions de transistors de 0.35µ
  - ➢200 MHz
  - ➢2 cm^2
  - ➢Données 32 bits
  - ➢3.3V, 35W



Augmentation de performances

➢Loi de MOORE



Augmentation de performances

- **Tous les 3 ans une nouvelle technologie**

Année d'introduction	1997	1999	2001	2003	2006	2009	2012
Technologie (µm)	0,25	0,18	0,15	0,13	0,1	0,07	0,05
Fréquence (MHz)	750	1250	1500	2100	3500	6000	10000
Mémoire	256M	1G		4G	16G	64G	256G
Nombre de transistors	11M	21M	38M	77M	202M	520M	1350M
Coût par M Tr (Cent.)	500	290	166	97	42	18	8

- **Réduction d'un facteur k**
  - ➢vitesse augmente d'un facteur k
  - ➢mémoire augmente d'un facteur k^2

Augmentation du nombre de registres

- tendances :
  - les processeurs CISC consistait à proposer de plus en plus d'instructions travaillant avec la mémoire, donc peu de registres
  - avec les processeurs RISC, toutes les instructions s'exécutent sur les registres, donc il y a un besoin important en nombre de registres
- registres généraux ou dédiés :
  - pentium : CISC
    - 8 registres généraux (héritage de la famille Intel)
    - 8 registres dédiés aux opérations flottantes :
      - implémentés sous forme de PILE
      - toutes les opérations s'effectuent avec le registre du sommet de pile : goulot d'étranglement sur ce registre
  - Dec Alpha 21064 : RISC : architecture Load/Store
    - 32 registres entiers
    - 32 registres flottants

Augmentation de performances

- ♦ power PC 601: RISC
  - 32 registres entiers
  - 32 registres flottants

➢ la largeur registres augmente :

- ♦ Dec Alpha 21064 : 64 bits
- ♦ Power PC : 32 bits
- ♦ Pentium : 32 bits pour les registres entiers, 80 bits pour les flottants

➢ intégration de registres spécialisés : MMX

- ♦ Pentium : 8 registres 64 bits (MM0, ... MM7)
  - correspondent aux registres flottants, attention aux conflits lors du mixage d'instructions flottantes et MMX
- ♦ Autres processeurs :
  - Introduction de formats de données spécifiques
  - Introduction d'instructions spécifiques à ces formats de données

Augmentation de performances

Opérateurs et instructions particuliers

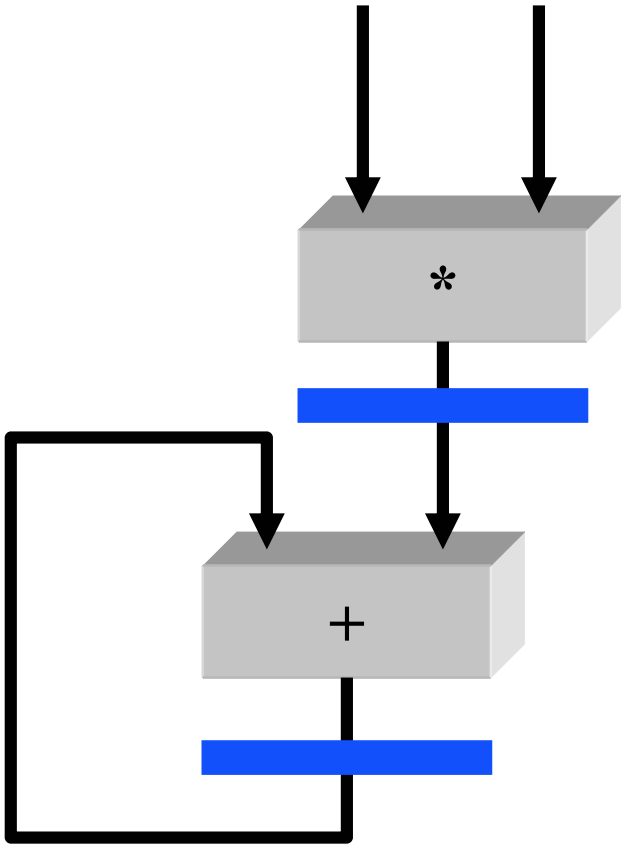
- architecture Power :
  - additionneur à 3 entrées
  - multiplicateur / diviseur
  - multiplicateur additionneur
- pentium :
  - diviseur
  - racine carrée
  - instructions transcendantales (trigonométriques, logarithmiques, hyperboliques)
- instructions multi média :
  - accélération des traitements du son et de l'image
  - nouveaux types de données : 8 octets, 4 mots, 2 doubles mots,

Augmentation de performances

➢Processeurs de traitement du signal :

♦ opération et opérateur MAC :

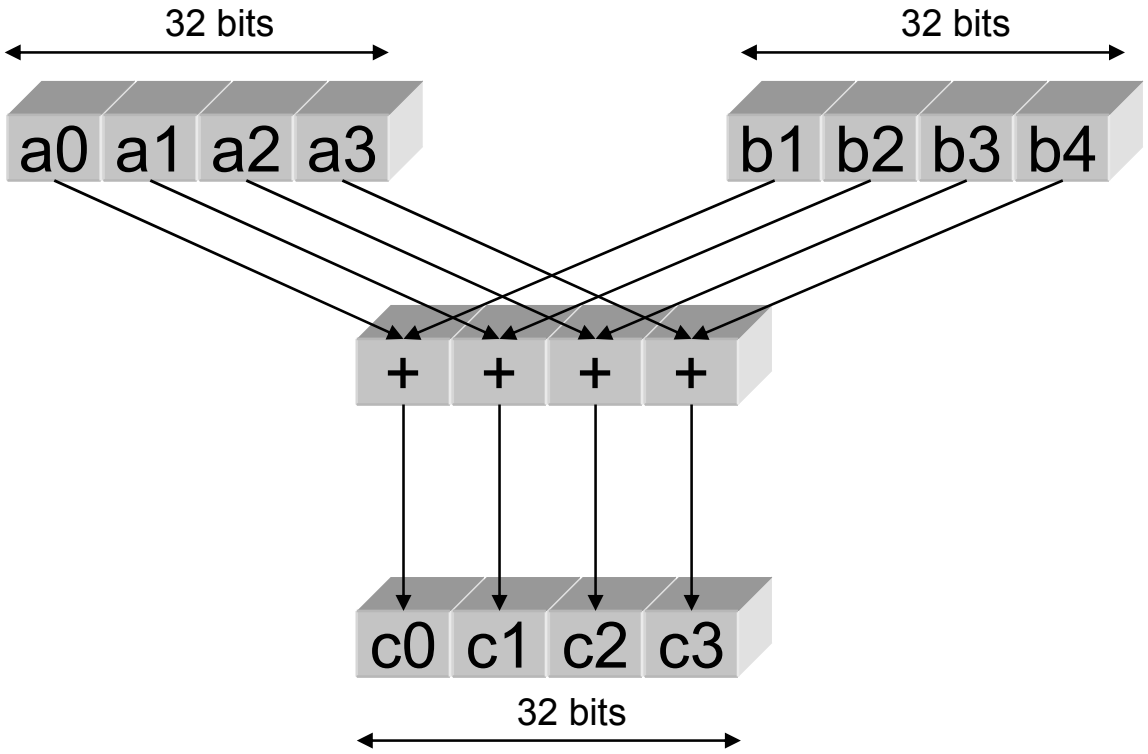
– s = a * b +c en 1 cycle



Augmentation de performances

➢Opérations SIMD ou SWP :

- ♦ SIMD : Single Instruction Multiple Data
- ♦ SWP : Sub Word Parallelism
- ♦ un opérateur * sur N bits peut faire :
  - 1 addition N bits par N bits (32 bits + 32 bits)
  - M additions sur N/M bits (4 * 8 bits + 4 * 8 bits)



Augmentation de performances

- Accélération des traitements vectoriels
- Exploitation du parallélisme de l'application

- Exemple de traitement pouvant bénéficier de ces opérations :
  - traitement d'images de bas niveau : par exemple rehaussement de contraste
  - multiplication de tous les pixels de l'image par un coefficient Beta

  - si on dispose d'un opérateur sur 32 bits pouvant réaliser des opérations sur 8 bits on peut aller 4 fois plus vite

Augmentation de performances

Nombre d'adressages disponibles

- tendances :
  - processeurs CISC : *augmentation*
    - beaucoup d'adressages complexes, langage machine proche des langages de programmation évolués
    - palier le manque de registres
    - problèmes :
      - décodage complexe des instructions
      - temps d'exécution des instructions variables
  - processeurs RISC : *limitation*
    - diminuer le temps de cycle, donc limiter la complexité des instructions
    - limitation du nombre d'instructions, et donc facilite le décodage, contrôleur plus simple

Augmentation de performances

❑ Latence versus débit

Avion	Paris/DC Heures	Vitesse Km/h	Passagers	Débit p*km/h
A380	7	890	555	493 950
Concorde	3	2 160	132	285 120

	Trajets X → Y	Vitesse Km/h	Passagers	Débit p*km/h
Train	8H	66,75	200	13 750
Voiture	5H	110	4	440

Augmentation de performances

❑ Définition du temps d'exécution :

$$\text{Tps exec} = \frac{\text{Nb d'instructions programme} * \text{Temps cycle}}{\text{Nb instr par cycle}}$$

1 programme	Instructions	Intr/cycle	Temps cycle	Temps d'exécution
68 030	1	0,19	60	31
Sparc	1,2	0,77	60	9,3

Augmentation de performances

❑ Sur quoi peut-on jouer ?

➢ diminuer le temps de cycle :

- ♦ la techno s'en occupe
- ♦ pas suffisant pour expliquer l'augmentation constatée

➢ organisation architecturale :

- ♦ augmenter le nombre d'étages du pipeline : pipeline profond
- ♦ élargir la taille du pipeline : superscalaire, Vliw

Augmentation de performances

Allongement des pipelines de contrôle

➢de Von Neumann sans pipeline : **LI, DI, EX, WR**

➢vers Von Neumann avec 4 étages pipelines :

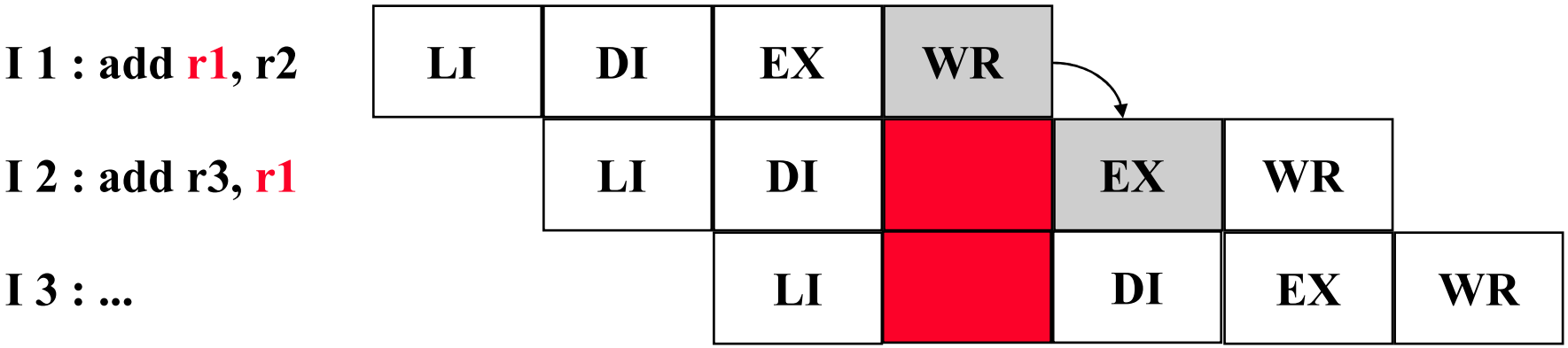
LI	DI	EX	WR

➢découpage en plusieurs sous tâches des phases :

- ♦ décodage :
  - – pré décodage pour les prédictions de branchement
  - – pré décodage instructions entière ou flottante
- ♦ exécution :
  - – les instructions flottantes sont très souvent pipelinées sur plus d'un étage

Augmentation de performances

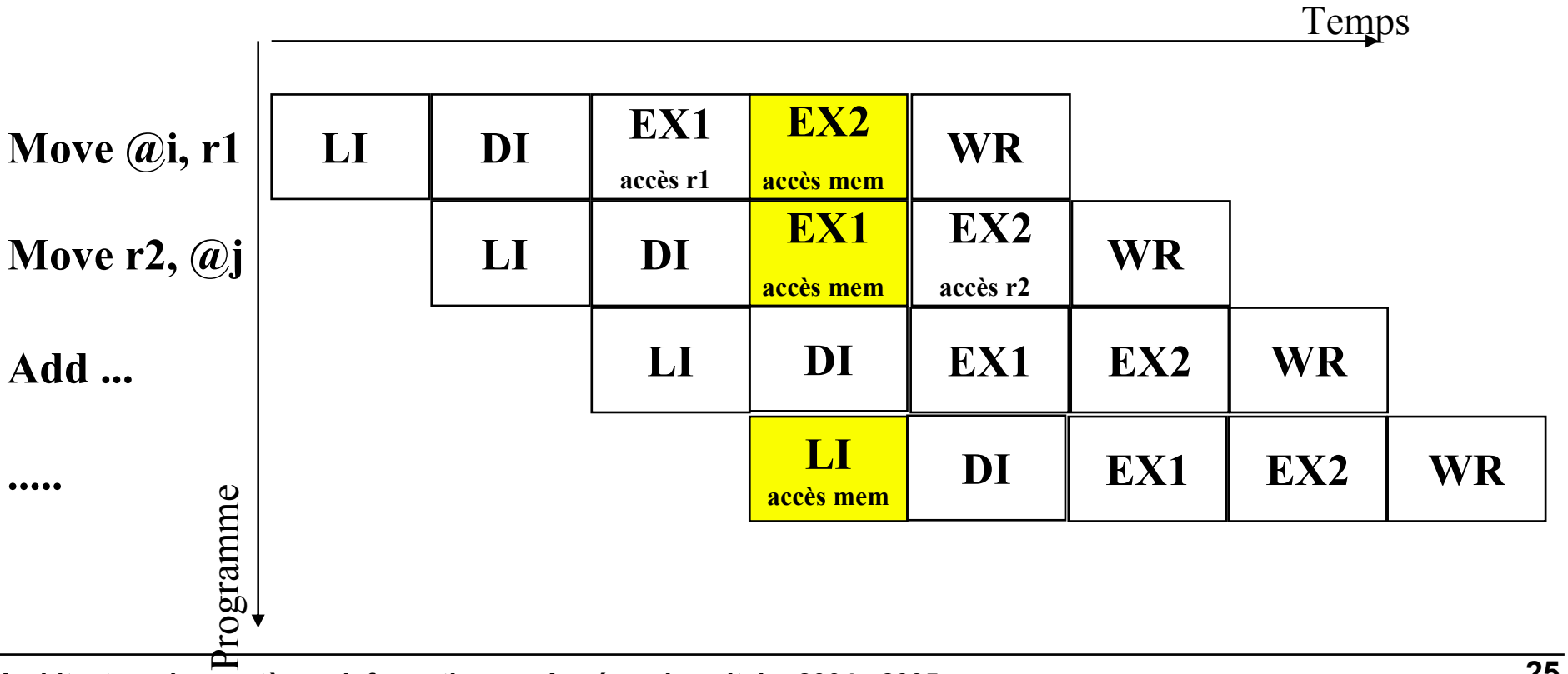
➢problèmes apportés par le pipeline de contrôle :



- ♦ dépendance : d'instructions et de données
- ♦ perte d'efficacité du pipeline pour cause de désamorçage
- ♦ plusieurs accès mémoire par cycle :
  - LI : accès IM en lecture
  - EX : accès DM en lecture ou écriture
  - WR accès DM en écriture

Augmentation de performances

- plusieurs accès mémoire par cycle (exemple) :
  - LI : accès mémoire instruction en lecture
  - EX1 : accès mémoire de données en lecture ou écriture
  - EX2 : accès mémoire de données en écriture



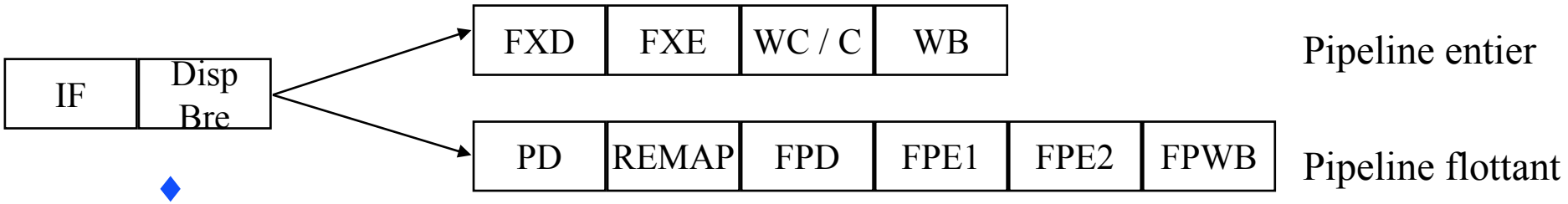
Augmentation de performances

➢ Exemples de pipelines :

♦ le power pc :

– pipeline entier sur 6 étages

– pipeline flottant sur 8 étages

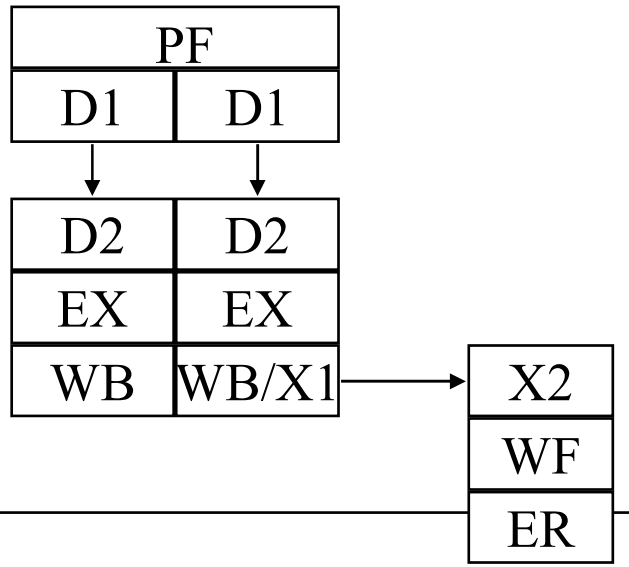


♦

♦ le pentium :

– pipeline entier sur 5 étages

– pipeline flottant sur 7 étages



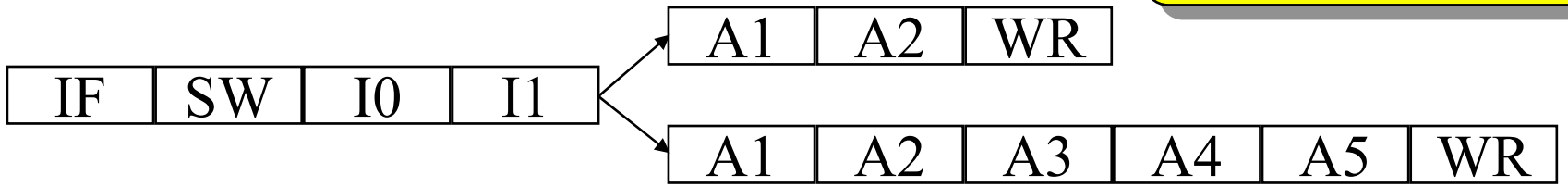
Augmentation de performances

♦ le Dec Alpha 21064 :

– pipeline entier sur 7 étages

– pipeline flottant jusqu'à 10 étages

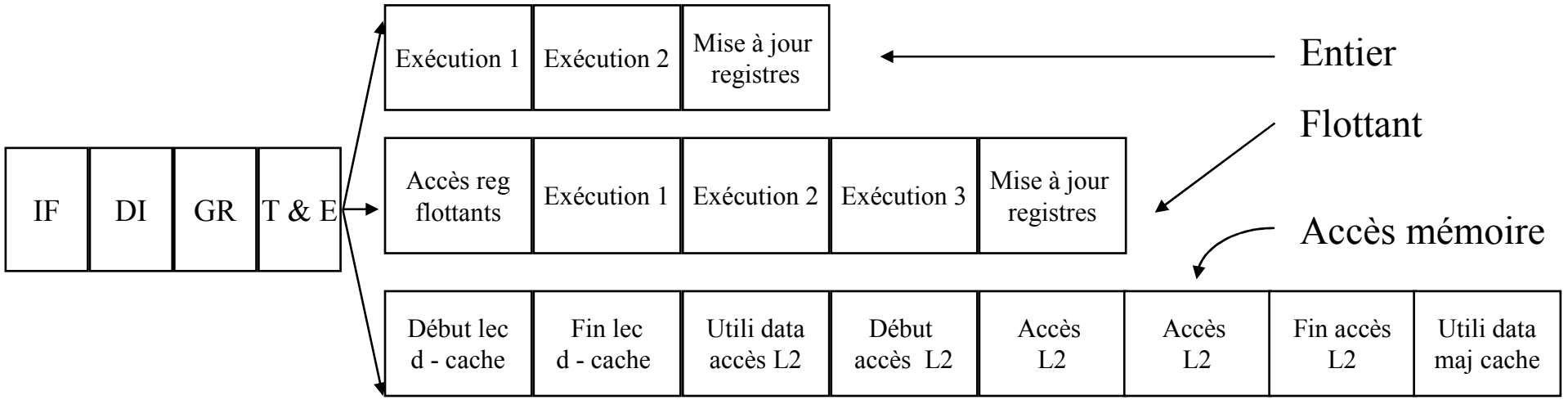
Processeurs super pipelines



- IF : instruction fetch
- SW : ??
- I0 : prédécodage
- I1 : fin décodage
- Ai : exécution
- WR : mise à jour état du processeur

Augmentation de performances

- le Dec Alpha 21164 :
  - pipeline entier sur 7 étages
  - pipeline flottant jusqu'à 9 étages



Augmentation de performances

♦ Le processeur MIPS R10000

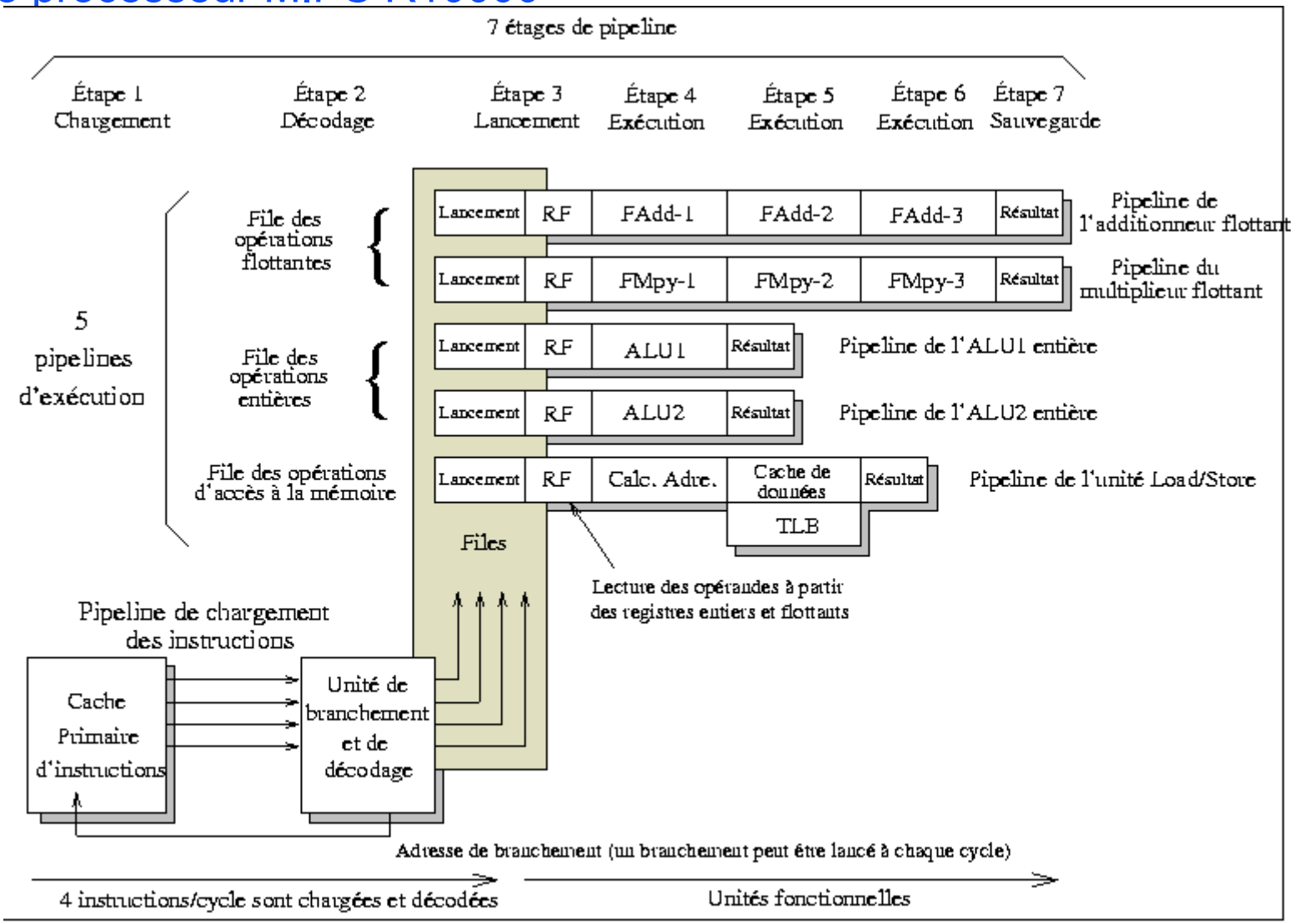


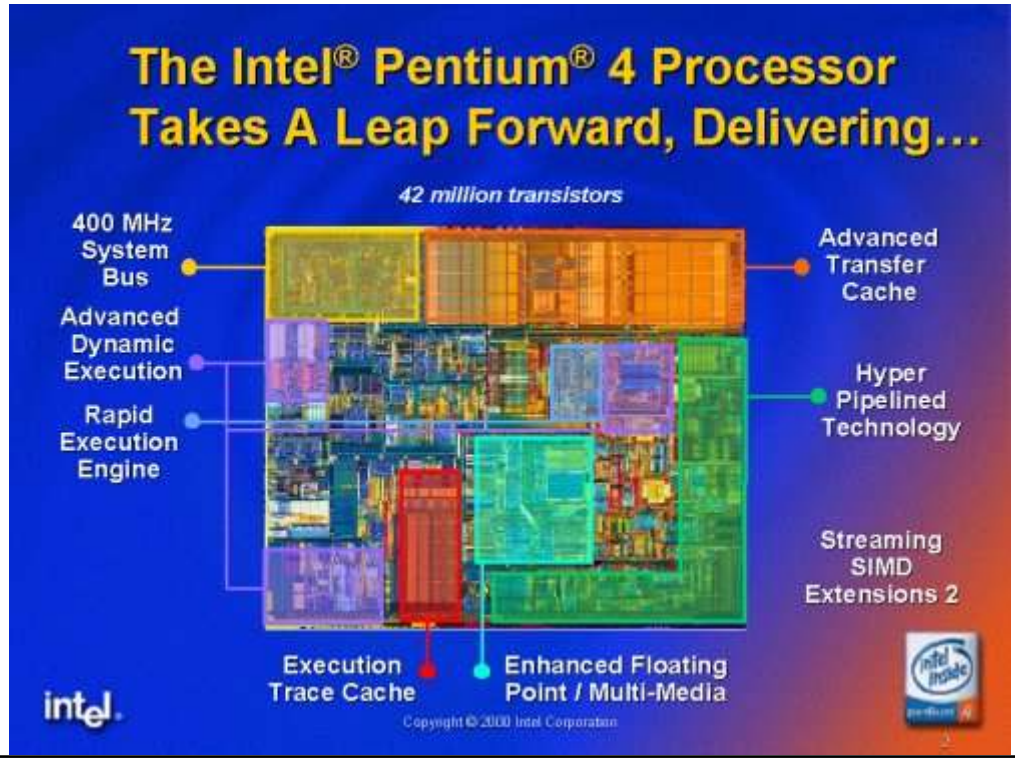
FIG. 3.1 – *Pipeline du MIPS R10000*

Augmentation de performances

♦ Le Pentium 4 :

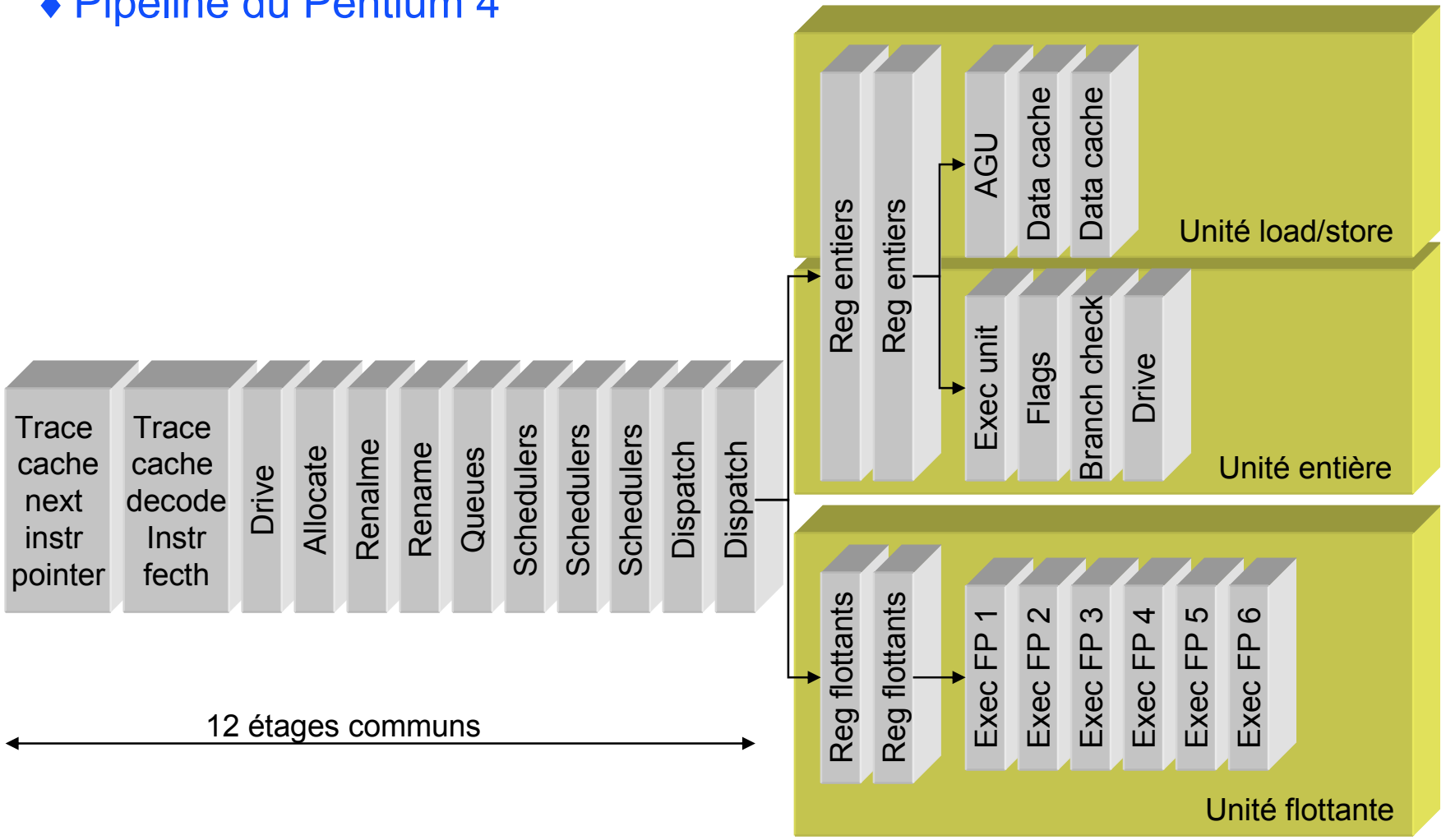
– 20 étages de pipeline :

- microarchitecture Intel NetBurst
- hyperpipeline
- l'augmentation du nombre d'étage permet d'augmenter la fréquence sans changement du procédé de gravage (0,13 micron)
- en contre partie, une mauvaise prédiction de branchement pénalise énormément le processeur : performances de la prédiction de branchement 93 % (90% pour le pentium 3)
- les UAL fonctionnent à vitesse double du processeur



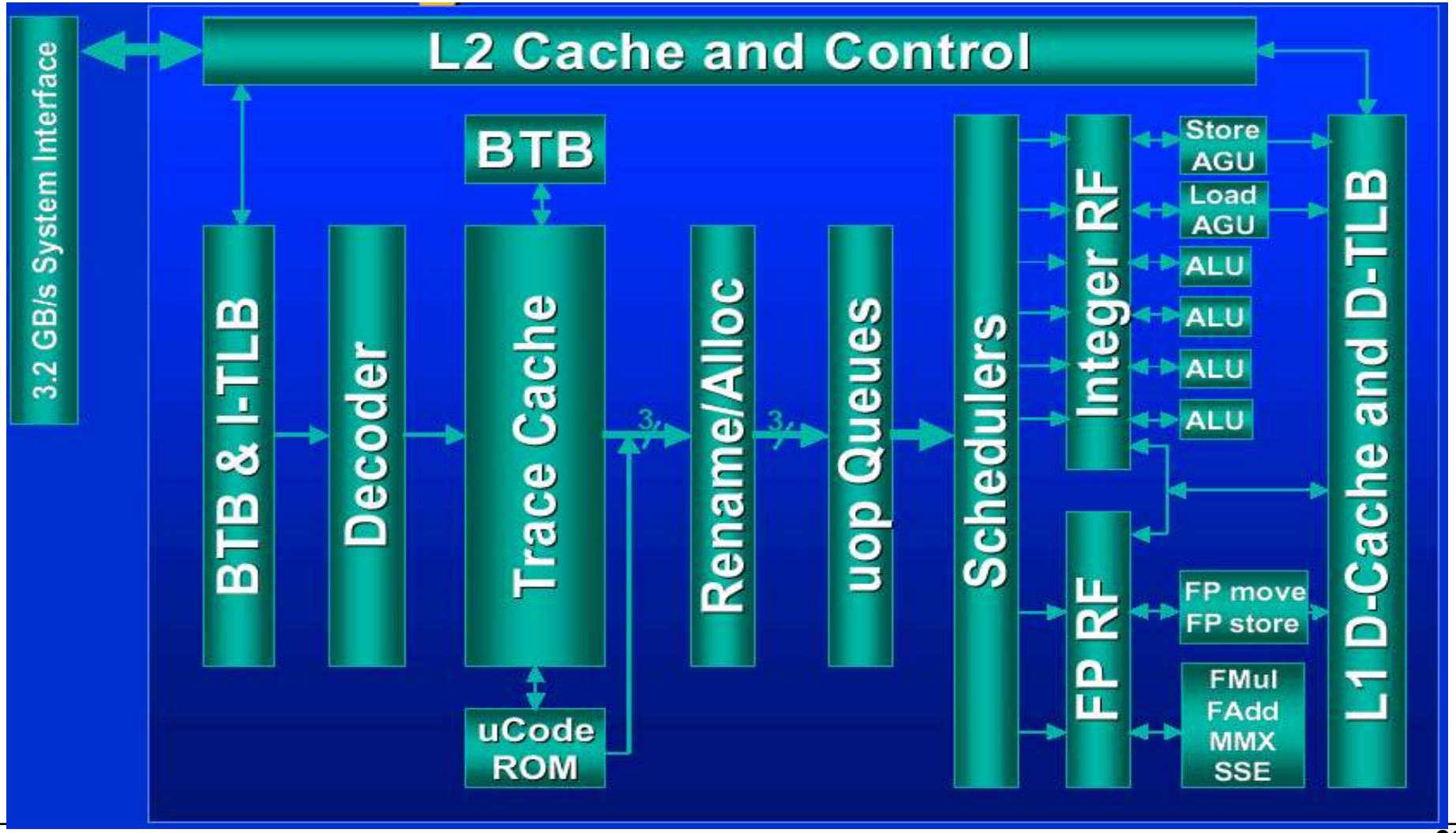
Augmentation de performances

♦ Pipeline du Pentium 4



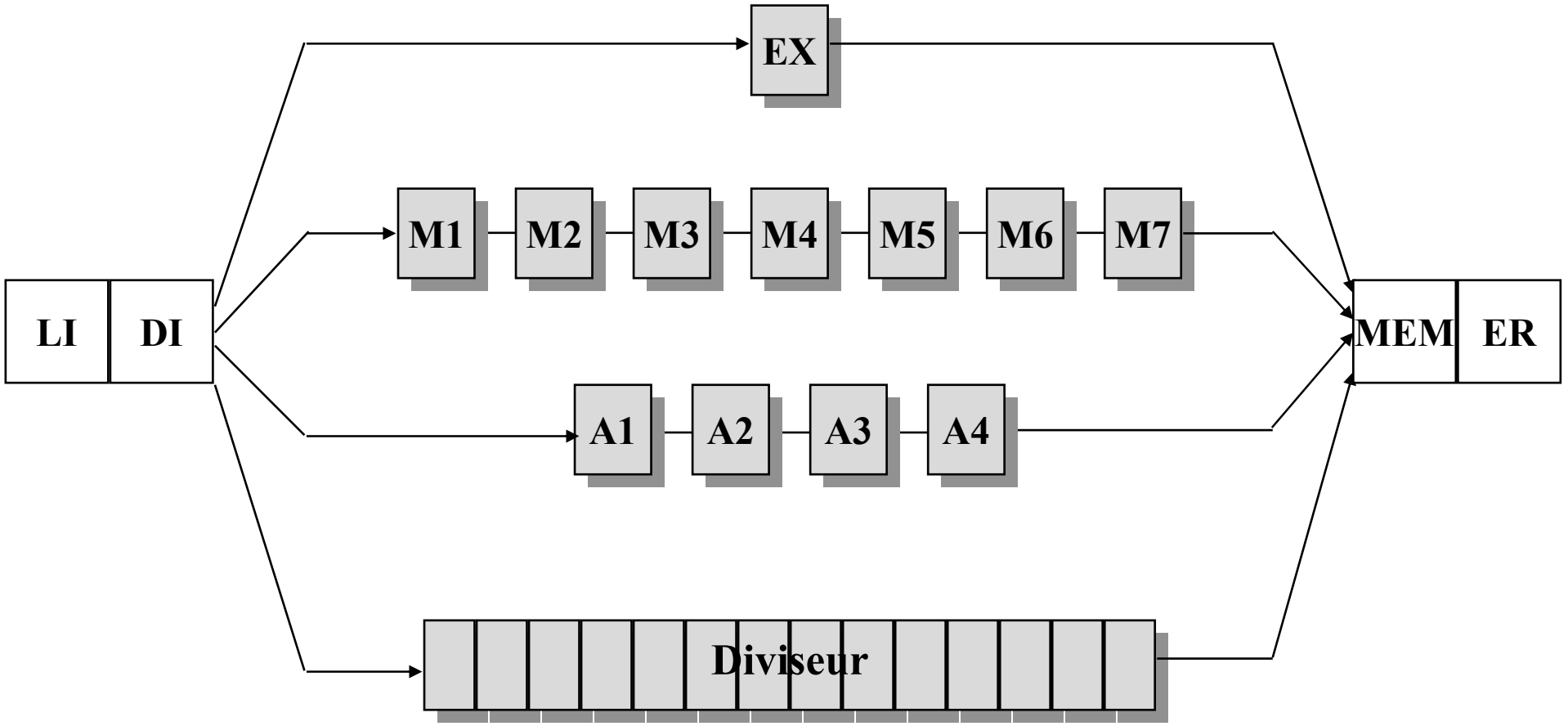
Augmentation de performances

♦ Autre présentation du pipeline du Pentium 4



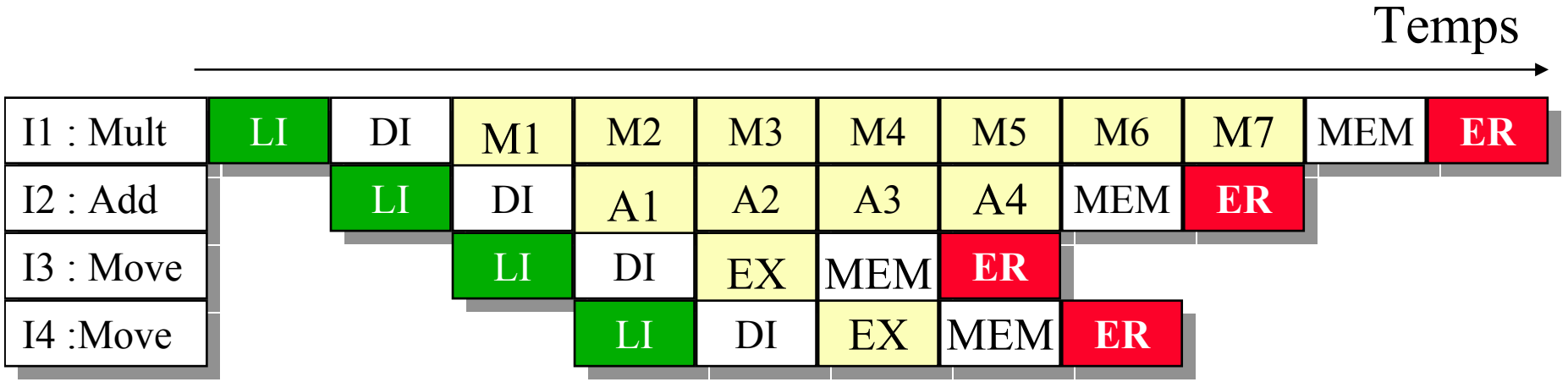
Augmentation de performances

♦ pipeline multi opérations :

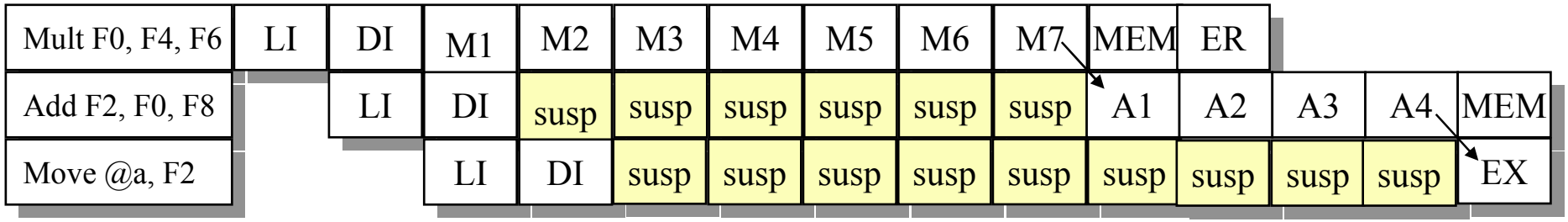


Augmentation de performances

– diagramme d'états de ce pipeline multi opérations :



– les instructions peuvent se terminer dans le désordre par rapport à leur lancement, ceci pose des problèmes avec les exceptions
– plus les pipelines sont longs, plus les pertes de cycles peuvent être importante

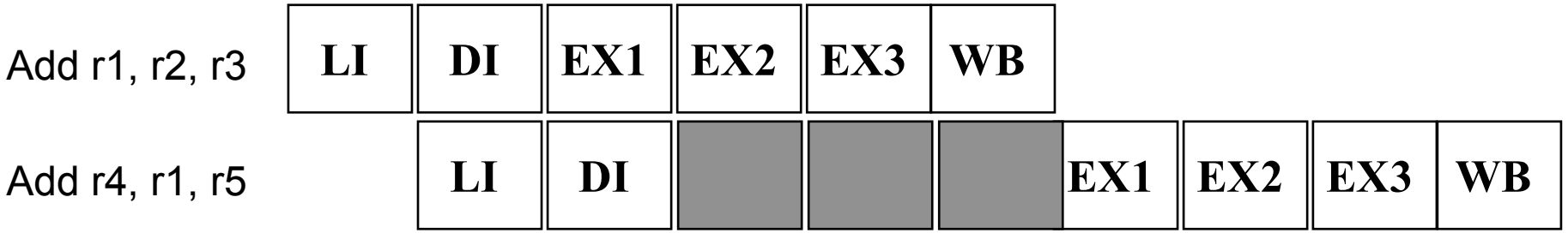


Augmentation de performances

❑ Mécanisme de bypass :

➢ objectif :

- ♦ faire passer une valeur attendue directement à une instruction sans attendre l'écriture en registre
- ♦ exemple de fonctionnement sans ByPass
  - soit le pipeline suivant :
    - lecture opérandes dans l'étage EX1
    - écriture opérande dans l'étage WB



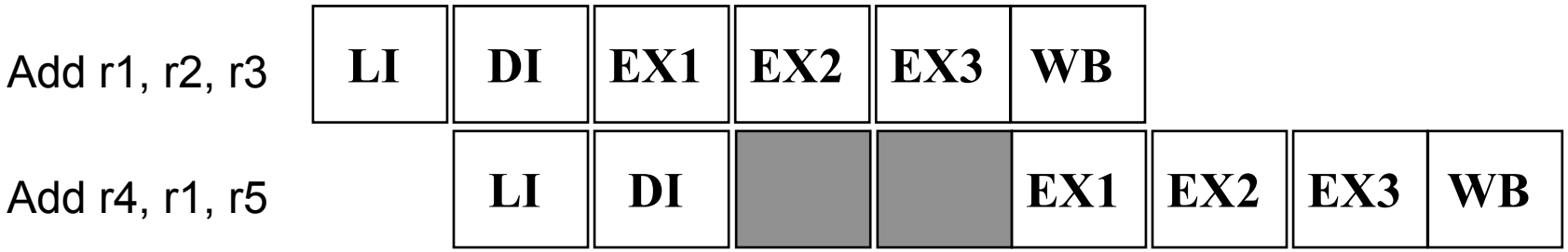
  - on perd 3 cycles

Augmentation de performances

♦ exemple de fonctionnement avec ByPass

– soit le pipeline suivant :

- Lecture opérandes dans l'étage EX1
- Écriture opérande dans l'étage WB

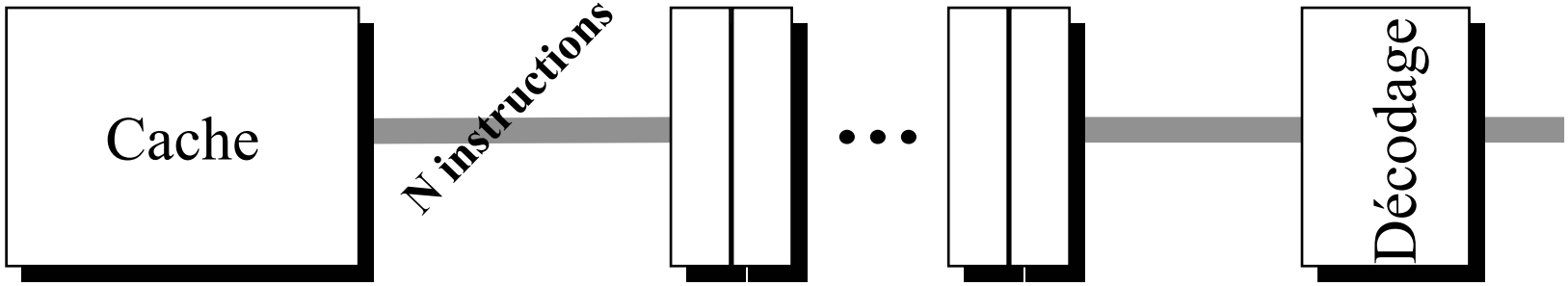


– on ne perd plus que 2 cycles

– le pipeline peut redémarrer plus rapidement

Augmentation de performances

❑ Buffer de préchargement des instructions

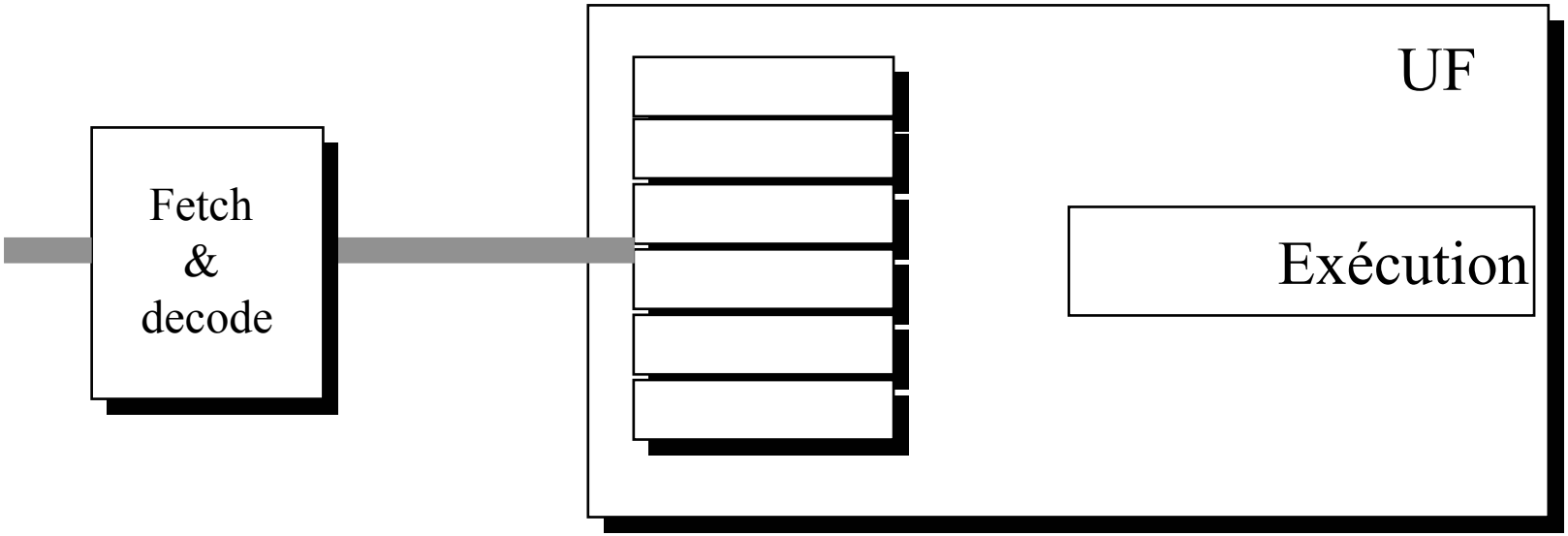


- ➢ permet de stocker un nombre important d'instructions
- ➢ nécessite souvent une augmentation de la largeur du bus entre le cache d'instructions et le buffer
- ➢ favorise le pré décodage des instructions (de branchement notamment)
- ➢ favorise l'exécution dans le désordre des instructions :
  - ♦ le processeur peut piocher dans le buffer d'instructions pour exécuter l'instruction i+1 avant l'instruction i
- ➢ favorise l'ILP (Instruction Level Parallelism) :
  - ♦ favorise la capacité a fournir des instructions au cœur du processeur qui est de plus en plus superscalaire

Augmentation de performances

➢Buffer d'instructions :

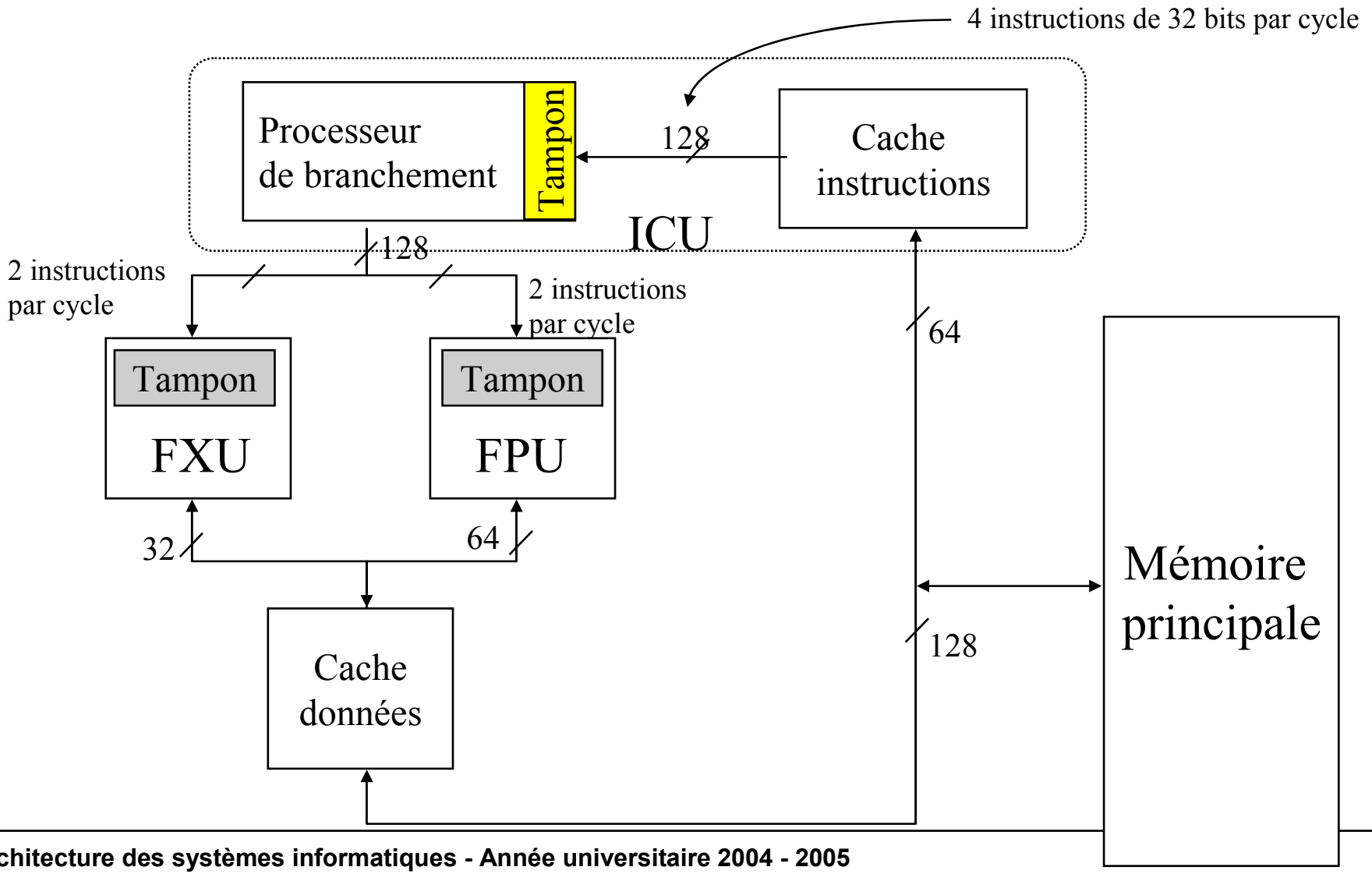
- ♦ découplage entre les étages de lecture / décodage et les étages d'exécution



- ♦ Etage fetch & decode : produit des instructions dans le buffer
- ♦ Etage exécution : consomme des instructions dans le buffer

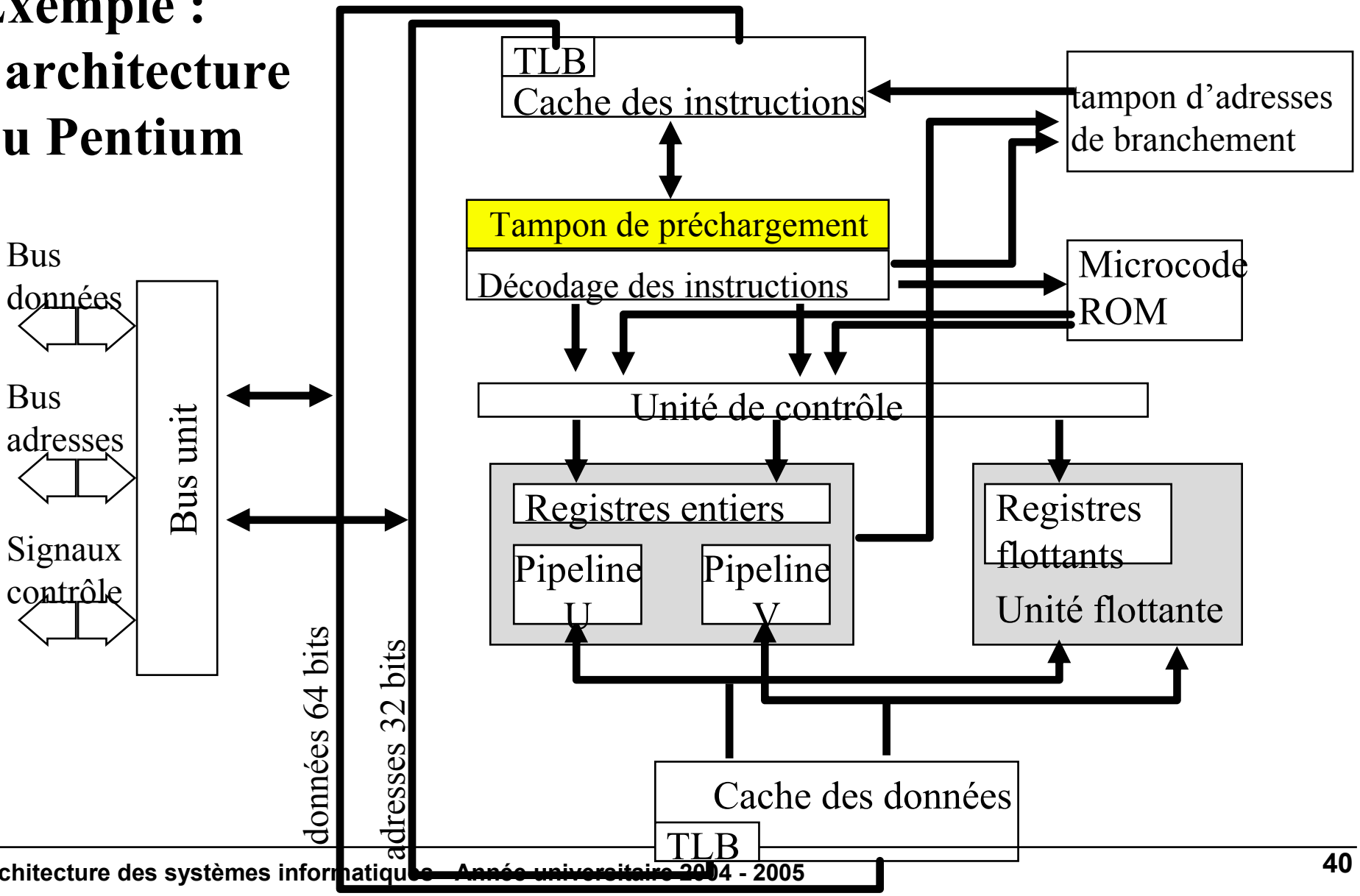
Augmentation de performances

Exemple : architecture Power PC



Augmentation de performances

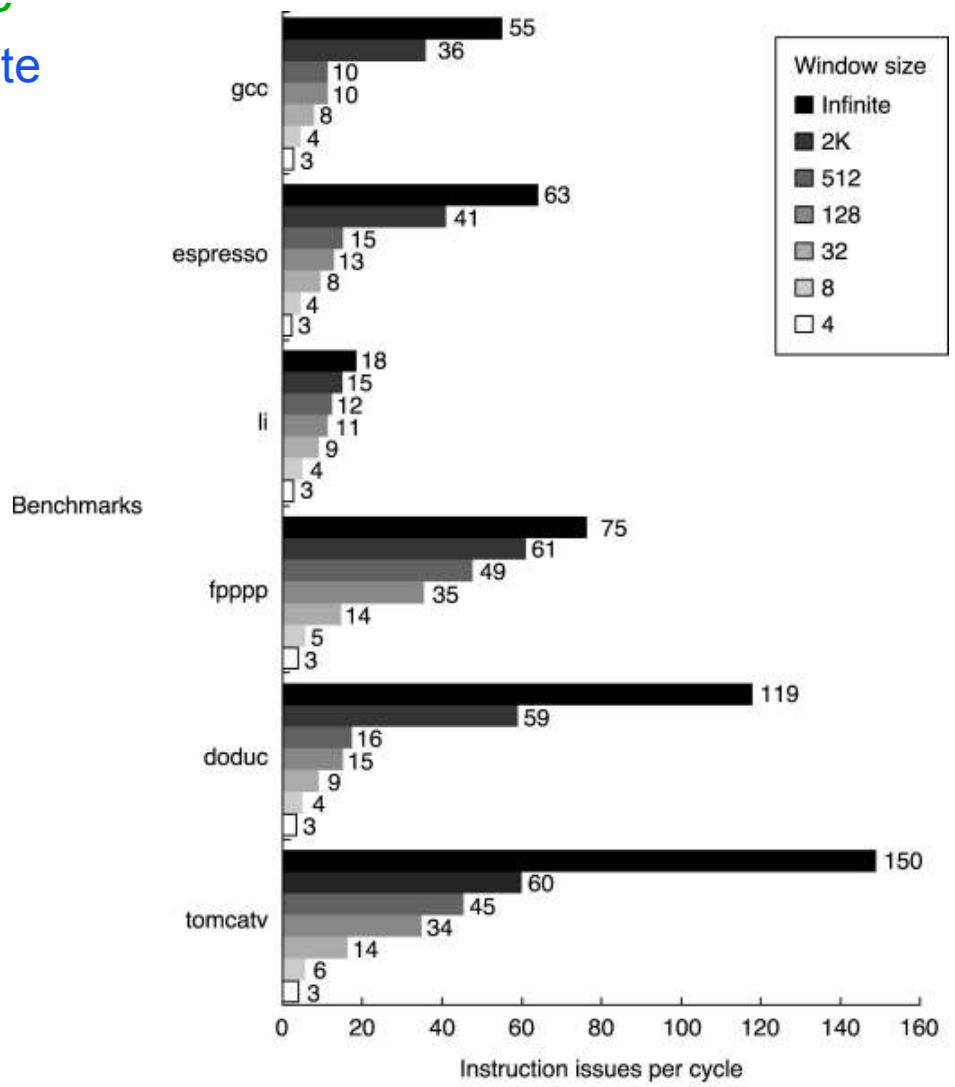
Exemple : l'architecture du Pentium



Augmentation de performances

➢ Effet de la taille de la fenêtre

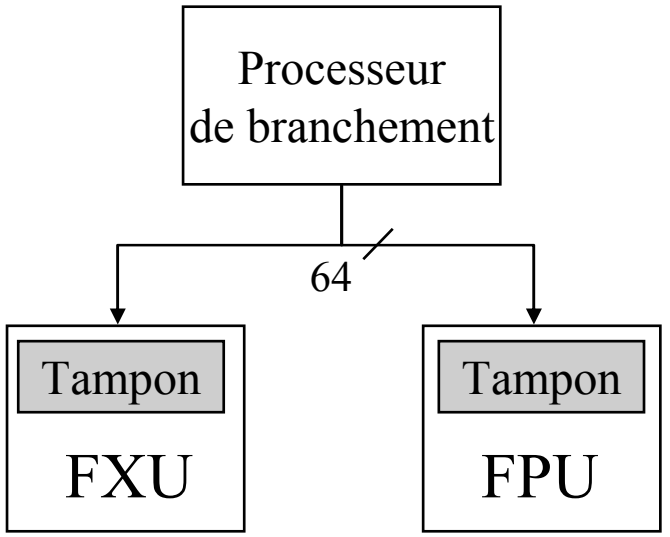
- ♦ plus la fenêtre est importante plus le processeur est en mesure de lancer des instructions en //



Augmentation de performances

➢ tamponnage des instructions au niveau de chaque unité :

- ♦ permet d'assurer une continuité dans l'alimentation du pipeline d'une unité même en l'absence d'instruction pour cette unité
- ♦ dans le Power PC : les instructions s'exécutent dans l'ordre

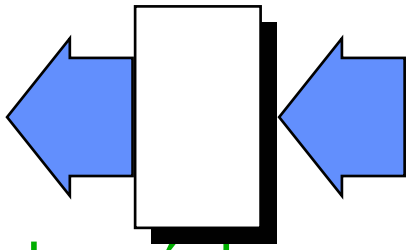


Augmentation de performances

❑ Module de dispatching

➢gestion de la cohérence entre les différentes unités et les différents types d'instructions :

- ♦ unités entières
- ♦ unité flottante
- ♦ unité de branchement



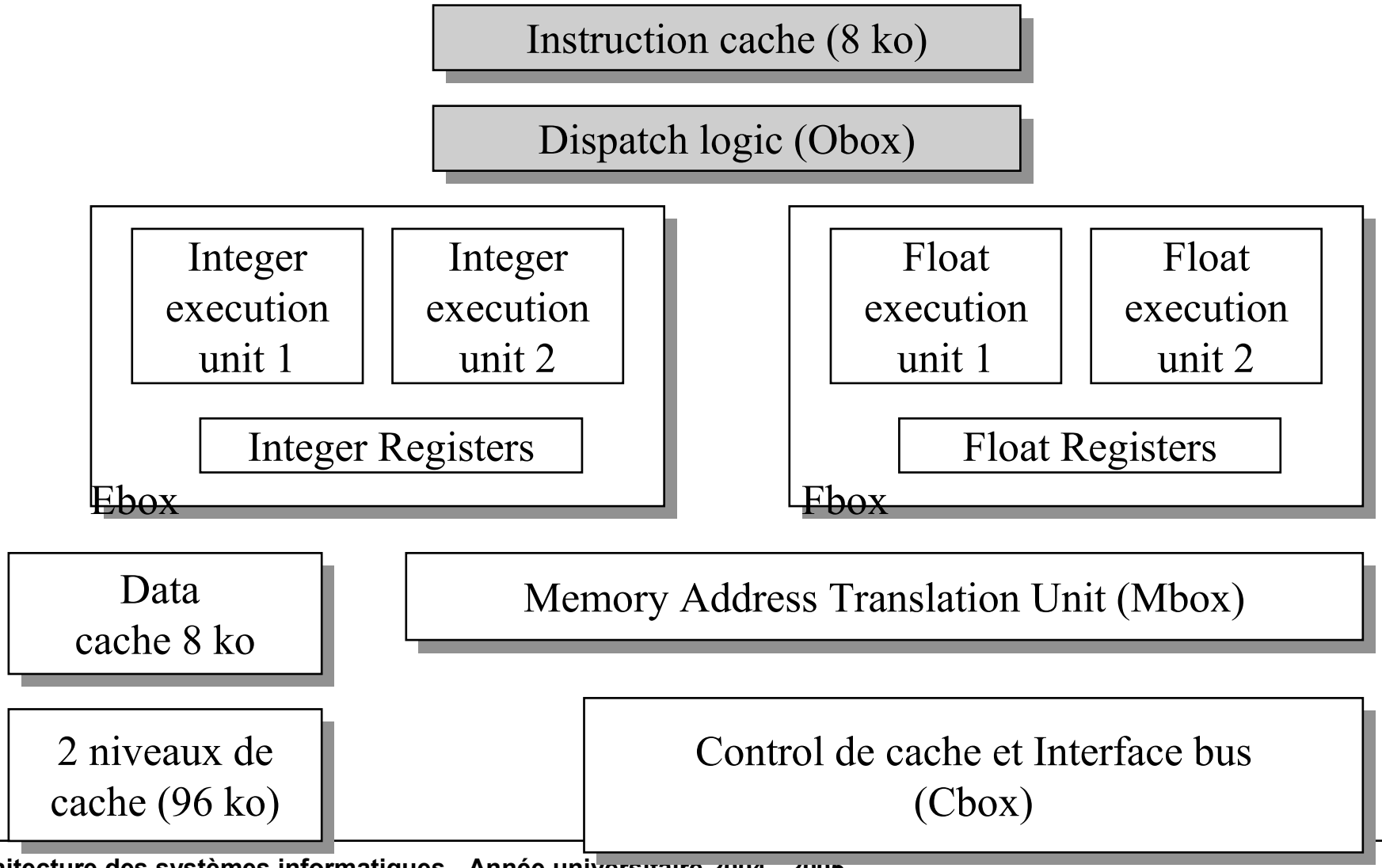
- instructions de branchement
- instructions flottantes
- instructions entières

➢puise dans le buffer de préchargement, pré décodage de l'instruction, et envoie vers le bon type d'unité

➢exemple : le power pc :

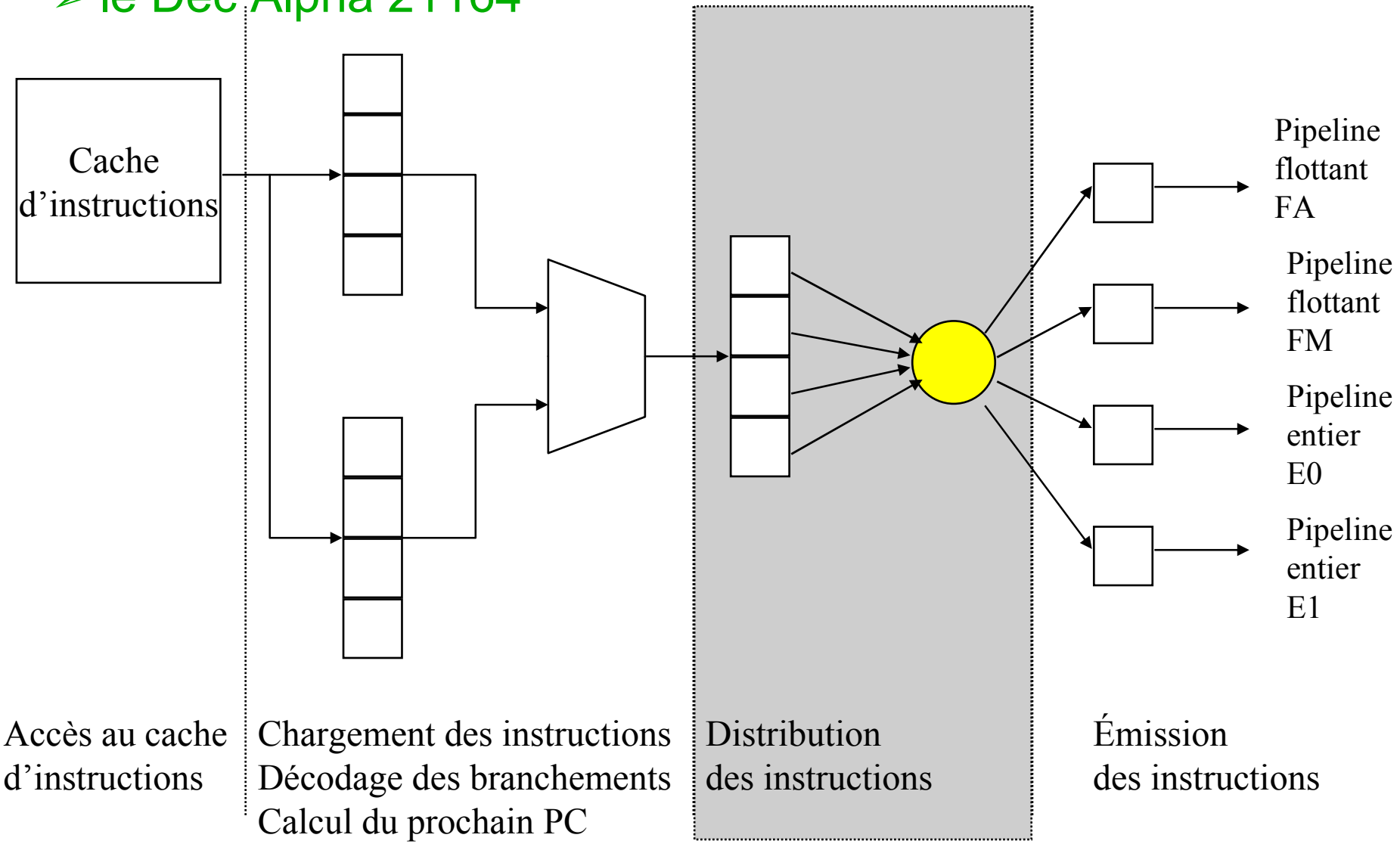
Augmentation de performances

➢exemple : le Dec Alpha 21164:



Augmentation de performances

➢le Dec Alpha 21164



Augmentation de performances

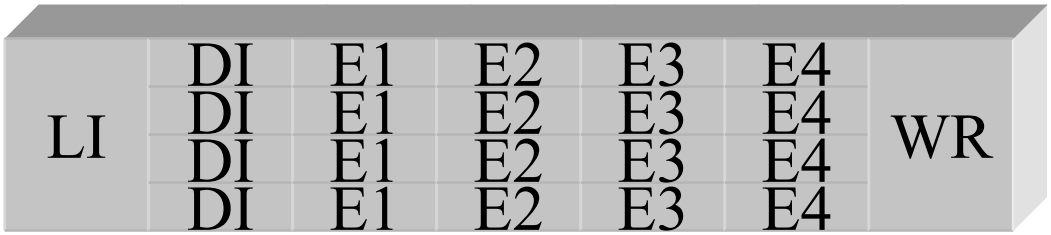
❑ Parallélisme dans les processeurs :

- ➢ plusieurs modèles :
  - ♦ pipeline :
    - – il s'agit d'un traitement à la chaîne
    - – plusieurs instructions sont exécutées en même temps mais elles sont à des niveaux de traitement différents
  - ♦ VLIW :
    - – plusieurs instructions sont exécutées en parallèle sur des unités différentes
    - – l'ordonnancement des instructions est déterminés à la compilation, statique
  - ♦ superscalaire :
    - – plusieurs instructions sont exécutées en parallèle sur des unités différentes
    - – l'ordonnancement est déterminé à l'exécution, dynamique

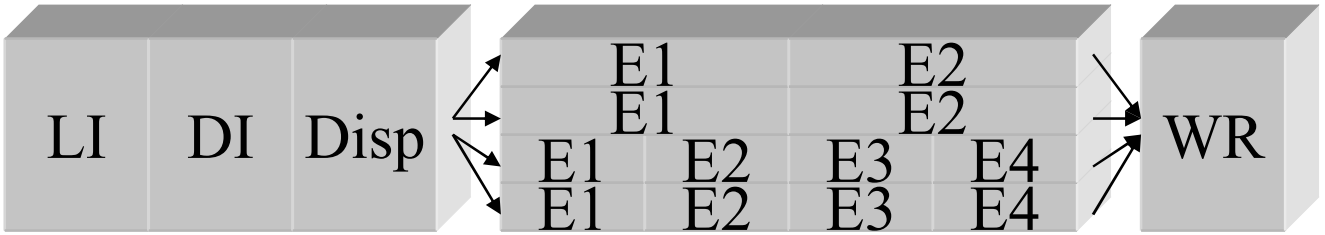
Augmentation de performances

➢pipeline

➢VLIW



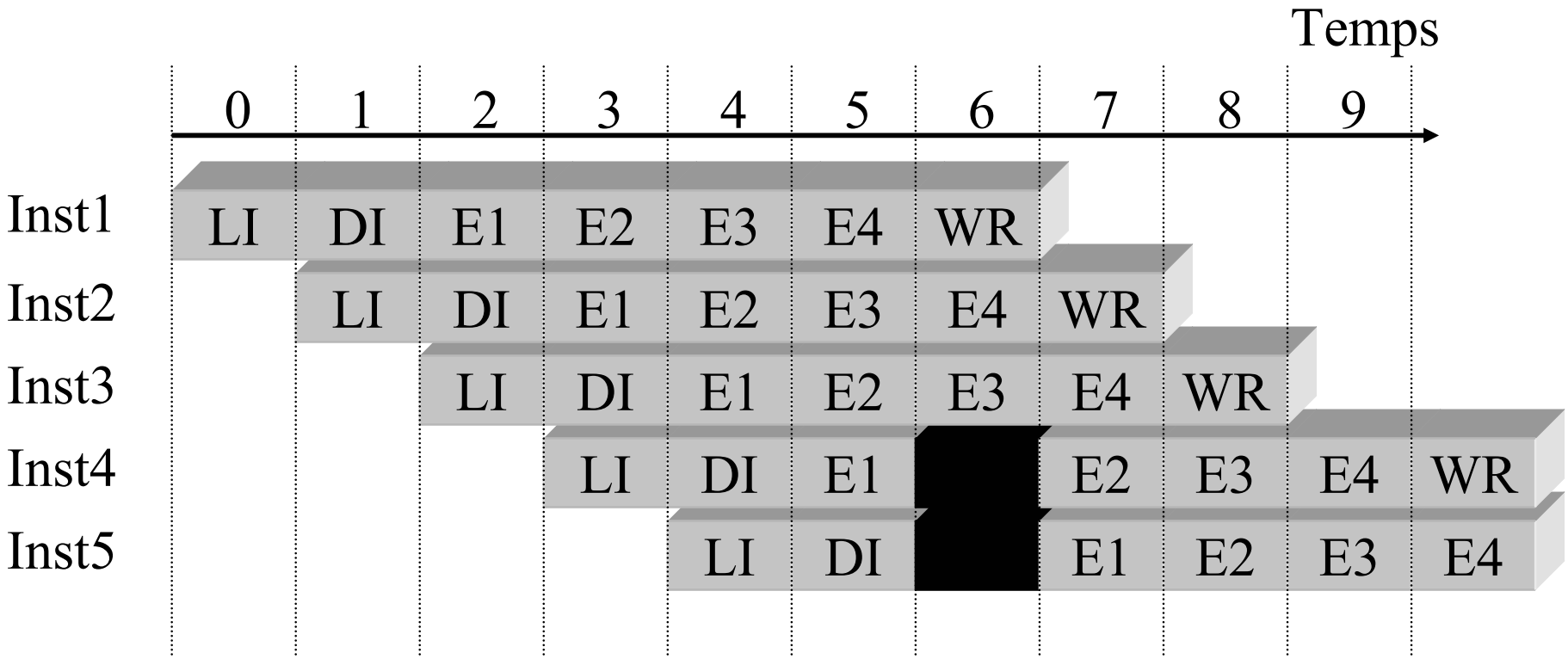
➢superscalaire



Augmentation de performances

➢Fonctionnement :

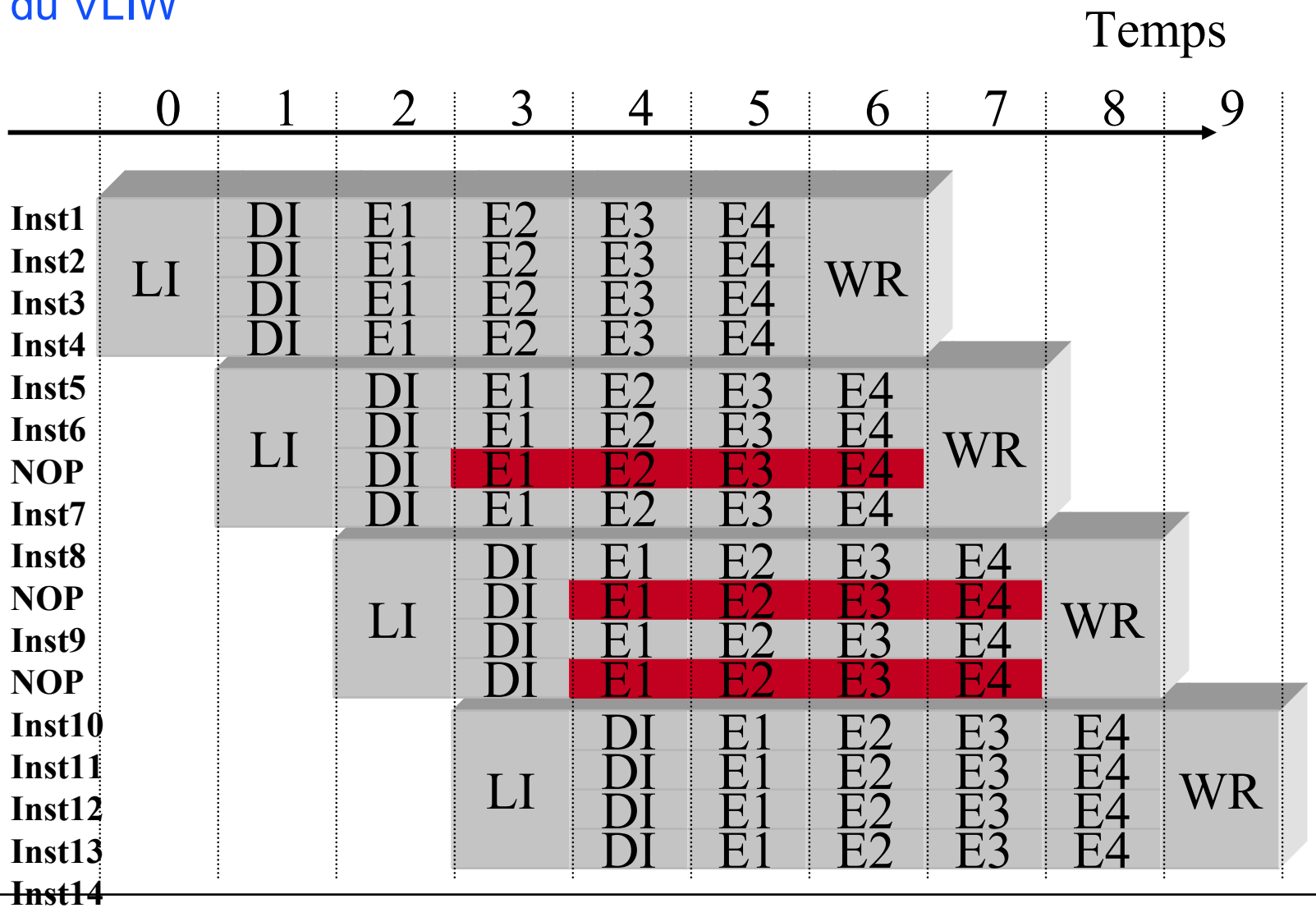
♦ du pipeline :



Augmentation de performances

➢Fonctionnement :

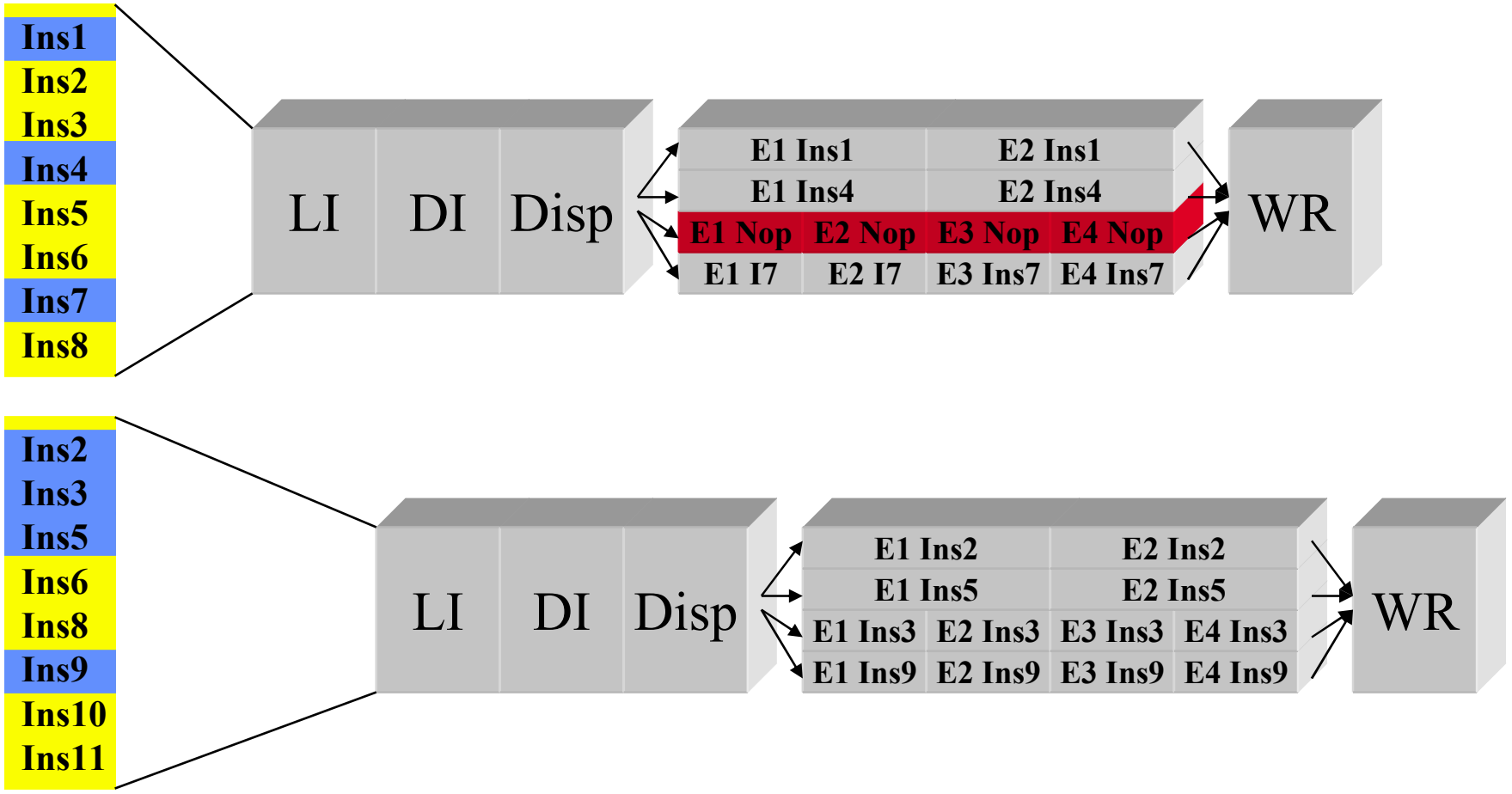
♦ du VLIW



Augmentation de performances

➢Fonctionnement :

♦ du superscalaire



Augmentation de performances

❑ Processeurs superscalaires

➢mise en place d'unités fonctionnelles indépendantes :

♦ fonctionnement parallèle des unités :

– donc les instructions s 'exécutant en parallèles ne doivent pas avoir de dépendance (données, instructions)

♦ unités équivalentes ou non :

– Pentium : 2 pipelines U et V, ne sont pas équivalentes, c'est le pipeline U qui réalise les opérations flottantes (en plus des instructions entières)

– Dec Alpha 21164 :

- 2 unités entières équivalentes
- 2 unités flottantes équivalentes

♦ unités fortement pipelinées :

– Pentium : 7 étages

– Power pc : 8 étages

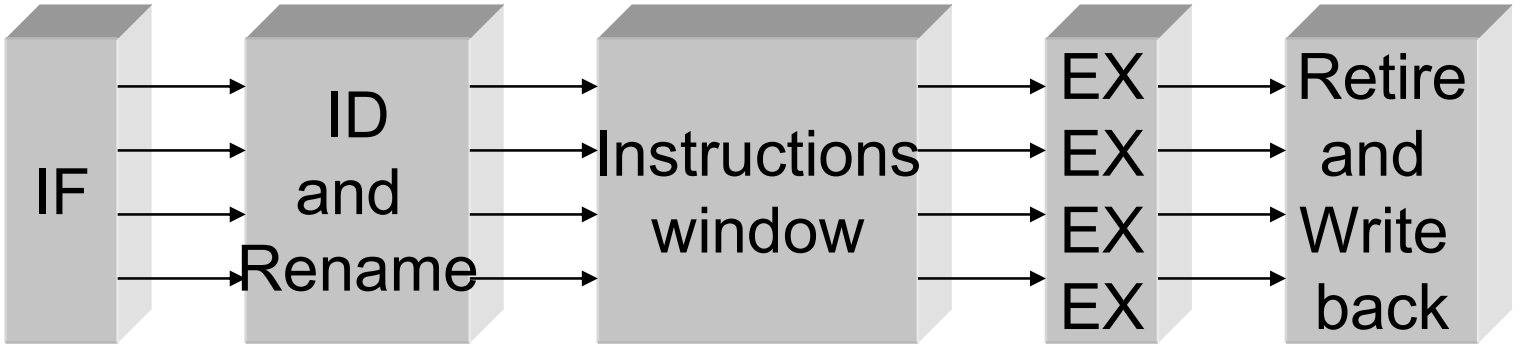
– Dec alpha : 10 étages

Augmentation de performances

- Associé à l'exécution dans le désordre
  - pour une utilisation optimale des pipelines l'ordonnancement des instructions est déterminé dynamiquement
  - logique de contrôle lourde
  - compilation simplifiée, toute (ou partie de) la complexité est rejetée sur le matériel
  - nécessite une gestion rigoureuse des ressources :
    - quelles instructions sont actives ?
    - quels registres vont être modifies par quelles instructions ?
    - quelles instructions peut être lancée à un temps de cycle précis ?
    - (fonctionnement détaillé par le tableau des marques, voir un peu plus loin)

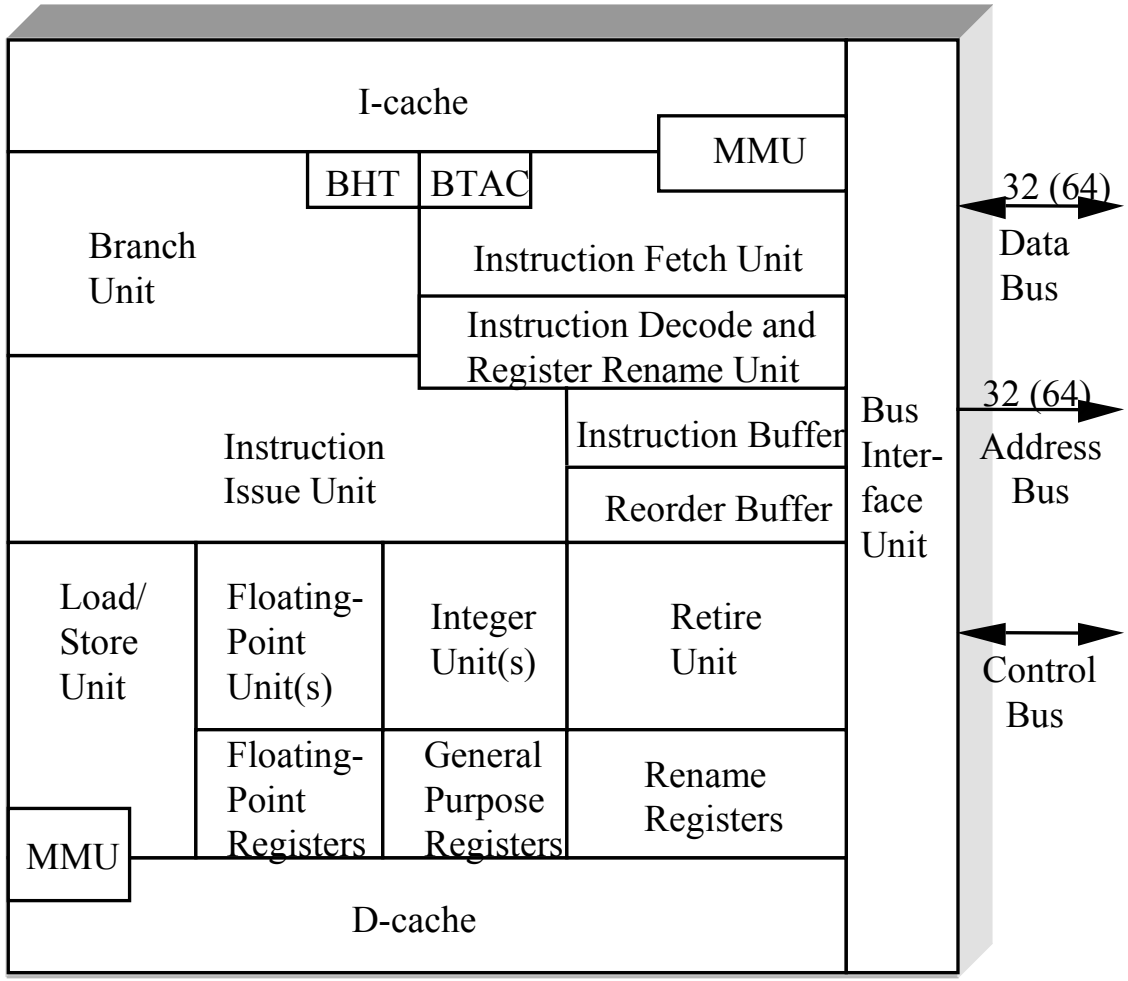
Augmentation de performances

➢Pipeline classique pour un superscalaire



Augmentation de performances

➢Architecture générale d'un processeur superscalaire



Augmentation de performances

➢Exemple du Pentium 4

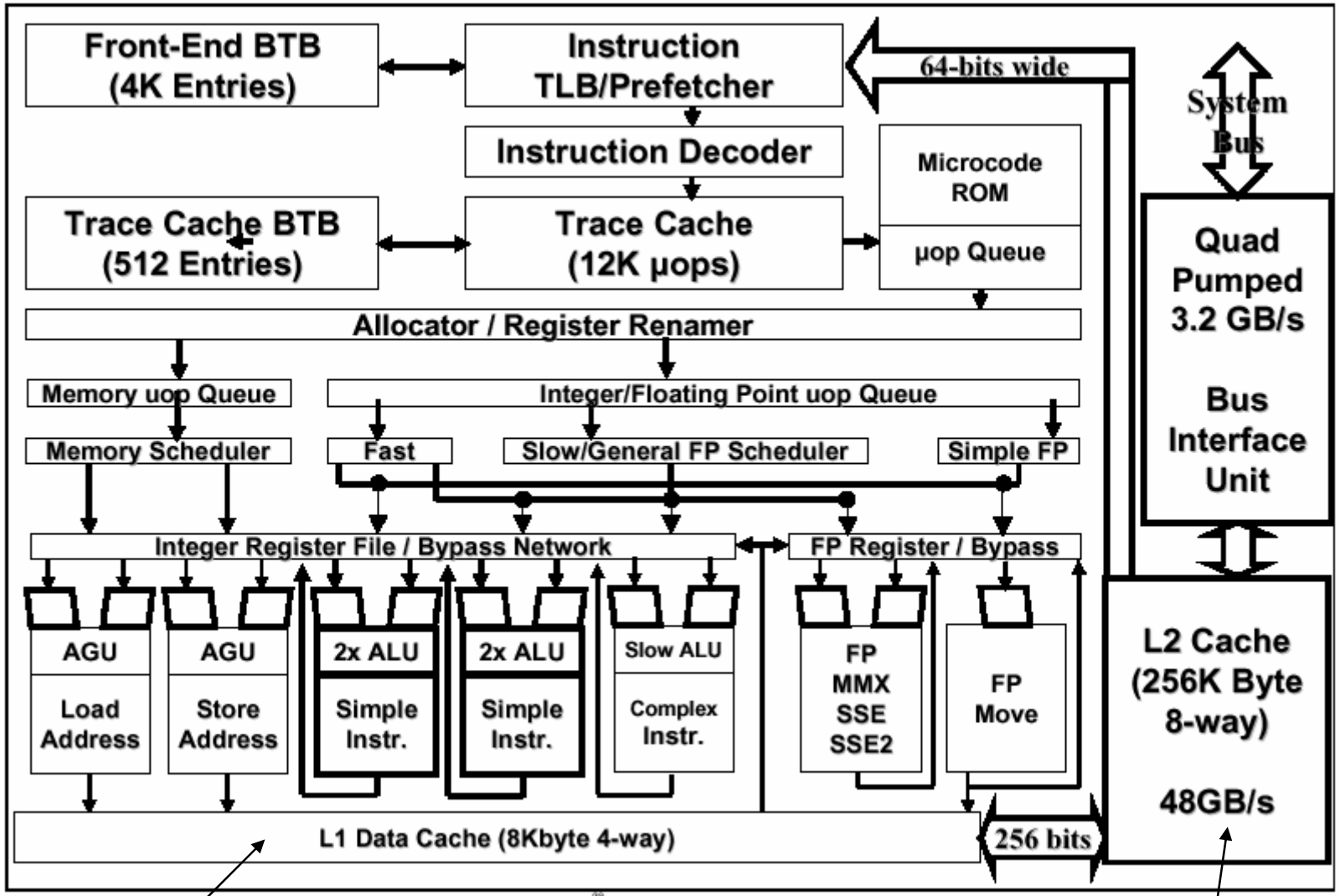


Figure 4: Pentium® 4 processor microarchitecture

Augmentation de performances

➢ Exemple du Pentium 4, suite :

♦ 7 unités indépendantes

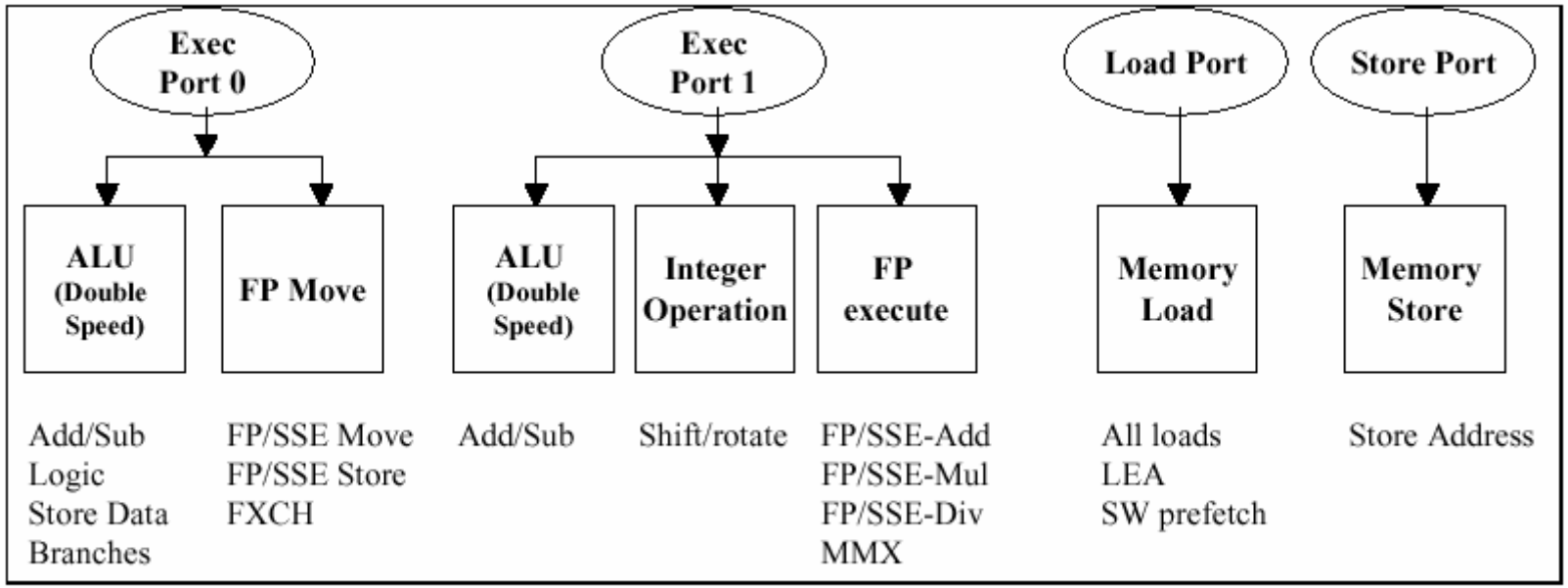


Figure 6: Dispatch ports in the Pentium® 4 processor

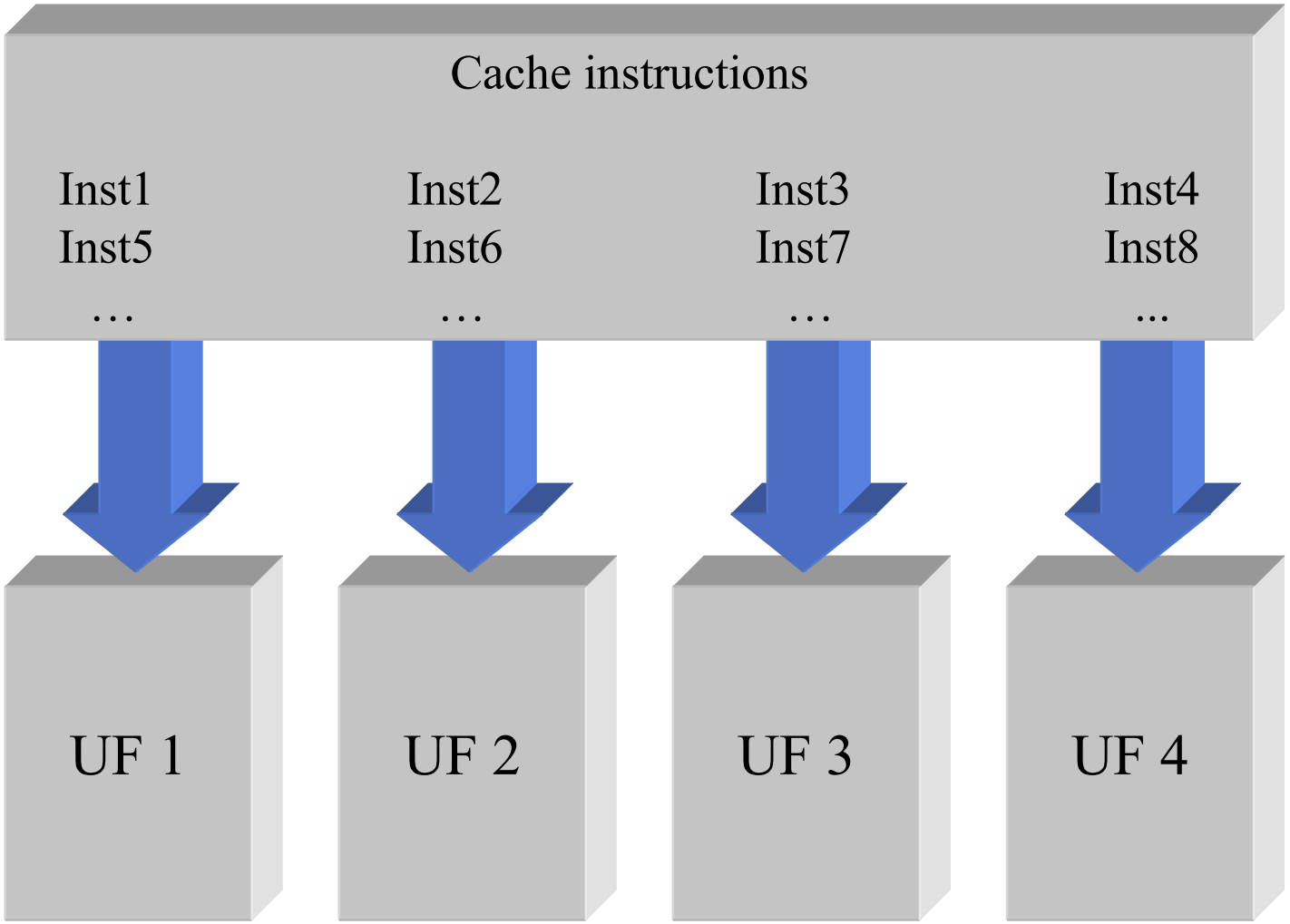
Augmentation de performances

❑ Processeurs VLIW

- ➢ pas d'exécution dans le désordre
- ➢ ordre des instructions déterminé à la compilation
- ➢ toute la complexité est rejetée sur le compilateur :
  - ♦ extraction du parallélisme de l'application
  - ♦ exploitation de celui-ci sur l'architecture
- ➢ a chaque cycle :
  - ♦ lecture d'une seule instruction longue :
    - – Very Long Instruction Word (exemple une instruction de 128 bits)
  - ♦ une instruction longue est composée de :
    - – N instructions élémentaires (exemple 4 instructions de 32 bits)
  - ♦ la position des instructions élémentaires définies l'unité fonctionnelle qui exécutera l'instruction

Augmentation de performances

➢Architecture VLIW typique

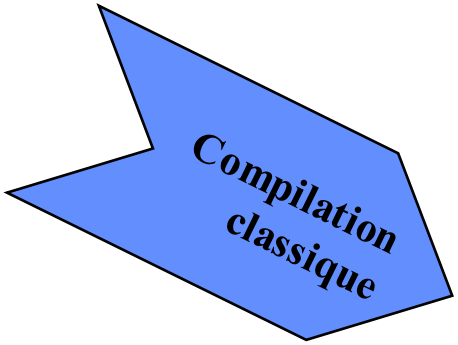


Augmentation de performances

➢ Exemple de code compilé :

♦ soit le programme réalisant le calcul du produit scalaire entre 2 vecteurs U et V :

```
P = 0.0;
for (i=0 ; i < Taille ; i++) {
        P = P + U[i] * V[i];
}
```

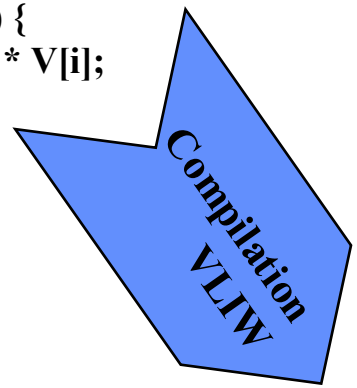



```
Debut
        Move    R0, @U
        Move    R1, @V
        Move    R2, @P
        Move    R3, Taille
        Move    F0, 0
Boucle
        Move    F1, (R0)
        Move    F2, (R1)
        Multf   F1, F2
        Addf    F0, F1
        Add     R0, 1
        Add     R1, 1
        Sub     R3, 1
        Bnez    Boucle
        Move    (R2), F0
```

Augmentation de performances

➢Exemple de code compilé (suite) :

```
P = 0.0;
for (i=0 ; i < Taille ; i++) {
        P = P + U[i] * V[i];
}
```



Debut	Move R0, @U	Move R1, @V	Move F0, 0	NOP
	Move R2, @P	Move R3, Taille	NOP	NOP
Boucle	NOP	NOP	Move F1, (R0)	Move F2, (R1)
	Add R0, 1	AddR1, 1	Multf F1, F2	NOP
	Sub R3, 1	NOP	Addf F0, F1	NOP
	Bnez Boucle	NOP	NOP	NOP
	NOP	NOP	Move (R2), F0	NOP

Augmentation de performances

➢ Exemple de code compilé (suite) :

- ♦ taille du code :
  - – compilation classique sur processeur RISC :
    - 14 instructions de format 32 bits = 448 bits
  - – compilation VLIW :
    - 7 instructions longues de format 4 * 32 bits = 896 bits

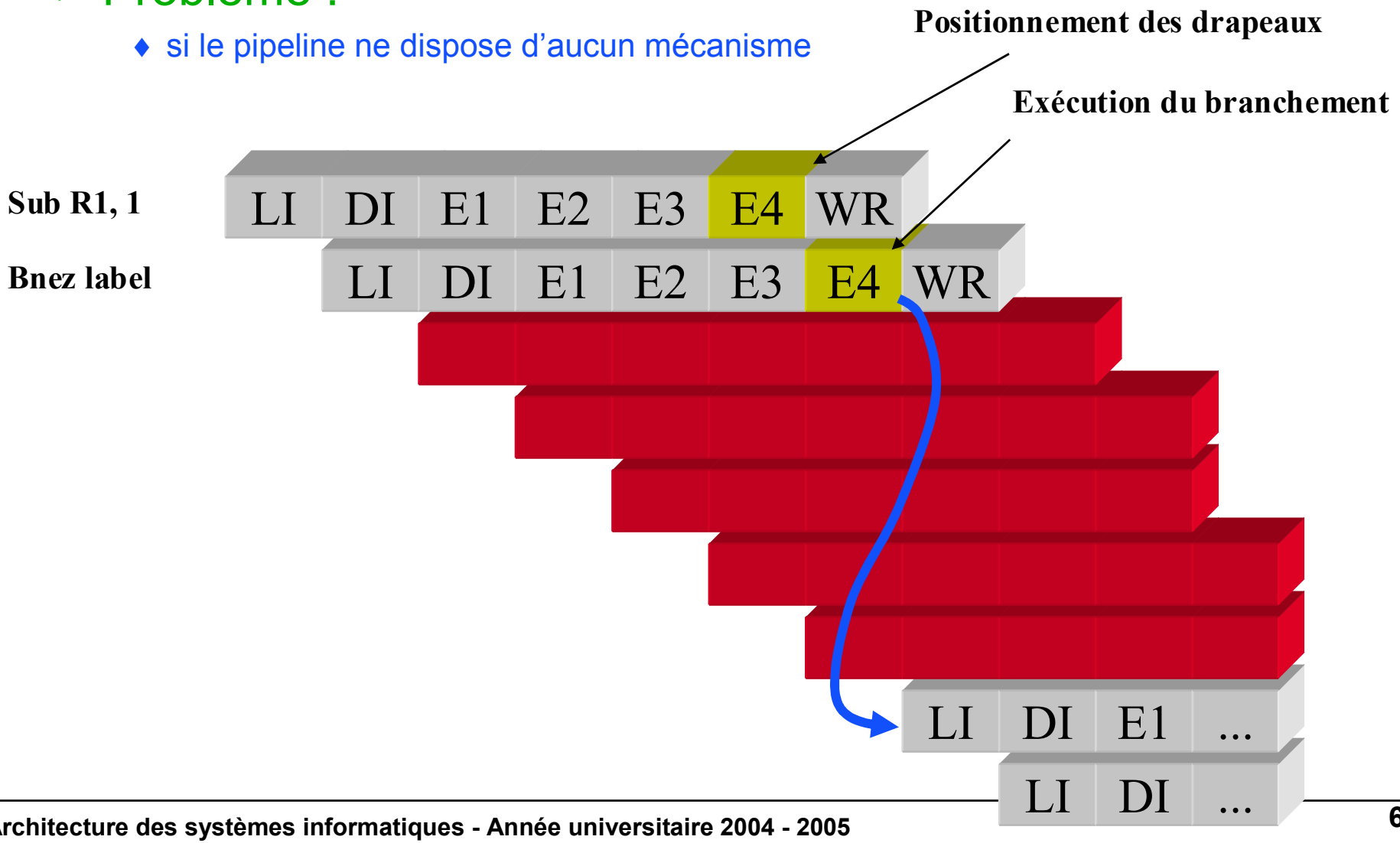
- ♦ pour cet exemple le code VLIW est 2 fois plus encombrant que le code RISC classique !!!
  - – problématique notamment pour les systèmes embarqués :
    - l 'espace mémoire peut être compté
    - la consommation doit être réduite
  - – solution :
    - taille de l'instruction longue variable, notion de bundle :
    - solution mise en œuvre dans l 'IA 64 (Itanium), Processeur Lx ST

Augmentation de performances

Prédiction de branchement

➢Problème :

- si le pipeline ne dispose d'aucun mécanisme



Augmentation de performances

➢Que faire ?

♦ limiter le nombre de cycles perdus dans les branchements, 3 solutions :

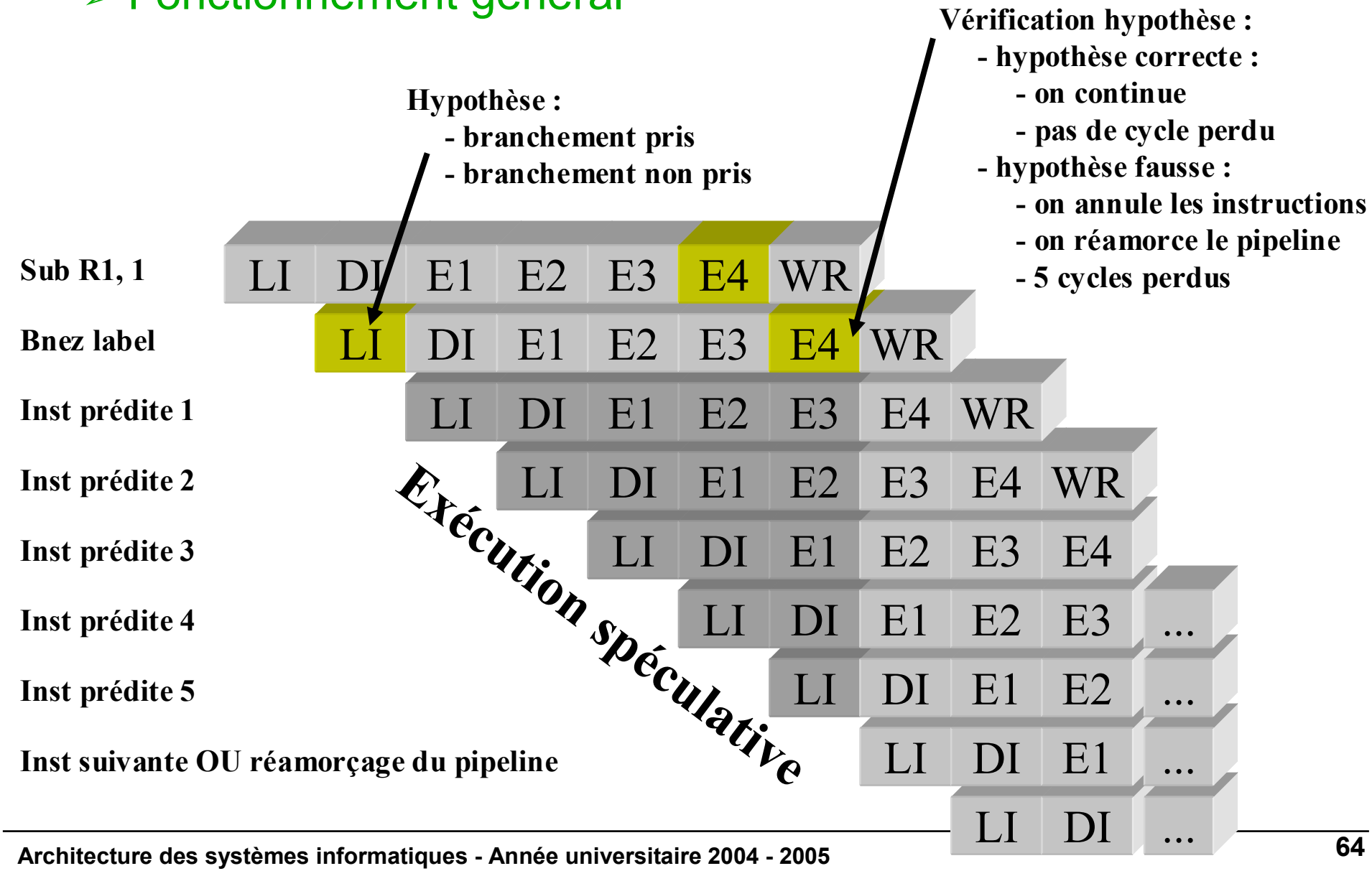
– limiter la longueur des pipelines :
  • c'est en contradiction avec l'augmentation de performance liée au travail à la chaîne

– limiter le nombre d'instructions de branchement :
  • certains processeurs proposent des instructions du type CMOVE

– tenter de prédire le comportement du branchement :
  • si la prédiction est correcte alors pas de perte de cycles

Augmentation de performances

➢Fonctionnement général



Augmentation de performances

♦ dans le cas ou l'hypothèse est fausse :

– les 5 instructions qui suivent le branchement doivent être annulées :

- on indique à l ’étage WR de ne pas réaliser les écritures (en registres ou en mémoire) et ce pendant 5 cycles
- les 5 instructions ont donc un comportement équivalent à des NOP
- on a perdu 5 cycles
- le processeur réamorce le pipeline avec l ’instruction située à l ’adresse label

♦ dans le cas ou l'hypothèse est correcte :

– les 5 instructions qui suivent le branchement poursuivent leur exécution

– pas de cycle perdu

Augmentation de performances

➢Coût d'une mauvaise prédiction :

- sur un Alpha 21264 :
  - 4 à 9 cycles
- sur un Pentium II
  - 11 cycles

Augmentation de performances

➢ Technique pour établir la prédiction (hypothèse) :

- ♦ doit se tromper le moins souvent possible
- ♦ n'influence pas le nombre de cycles perdus lors du mauvaise prédiction
- ♦ type de prédictions :
  - – prédiction statique :
    - prédiction globale au processeur quelque soit le branchement :
    - par exemple : branchements toujours prédit pris, ou branchements toujours prédits non pris
  - – prédiction statique :
    - parfois le compilateur a la possibilité de positionner un bit dans l'instruction pour indiquer au processeur comment le branchement doit être prédit
  - – prédiction statique dépendant du sens du branchement :
    - prédiction globale au processeur
    - si le branchement est arrière alors prédiction statique 1 (par exemple : prédit pris)
    - si le branchement est avant alors prédiction statique 2 (par exemple : prédit non pris)
  - – prédiction dynamique :
    - mémorisation du comportement de chaque branchement (historique)
    - décision par rapport aux comportements précédents

Augmentation de performances

➢prédiction statique simple

- utilise la technologie des compilateurs
- examen des exécutions précédentes du programme :
  - on observe les branchements : (voir figure), on en conclue que la plupart des branchements sont pris, la manière la plus simple de prédire le branchement est alors de dire que tout branchement sera pris

➢prédiction statique tenant compte du sens du branchement

♦ le comportement des branchements dépend souvent de leur sens :

- on appelle branchement arrière un branchement du type :
  - @i BR condition, @j avec @ j < @i
- on appelle branchement avant un branchement du type :
  - @i BR condition, @j avec @ j > @i

Augmentation de performances

♦ Les branchements arrières :

– sont souvent issues de la compilation d ’une boucle :

```
For (i = 0 ; i < N ; i++) {
        cœur de la boucle ;
}
```

```
        Move R1, N
Boucle
        Bloc d ’instructions
        Sub R1, 1
        Brnez Boucle
        …
```

– comportement d ’une boucle facilement prédictible :

- le branchement est pris N fois
- et non pris 1 fois (à la sortie de la boucle)

♦ Prédiction de branchement du processeur :

– les branchements **arrières** seront toujours **prédits pris**

Augmentation de performances

♦ Les branchements avants :

– ils sont issues (entre autres) de structures conditionnelles

```
If condition then
        Bloc instructions 1 ;
else
        Bloc instructions 2 ;
end if;
```

```
        Br condition, else
        Bloc instructions 1
        JUMP suite
else
        Bloc instructions 2
suite
```

– comportement plus délicat à prédire

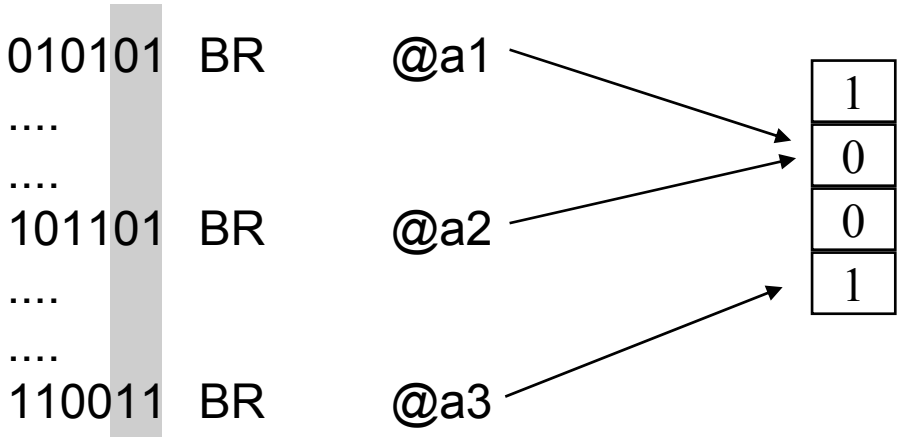
♦ Prédiction de branchement du processeur :

– les branchements **avants** seront toujours **prédits non pris**

Augmentation de performances

➢prédiction de branchement dynamique à 1 bit

- dépend du comportement des branchements au cours de l'exécution
- on conserve un historique du comportement des branchements
- la prédiction change si le comportement du branchement change
- gestion par une table d'historique des branchements :



Augmentation de performances

– efficacité de cette prédiction, soit le code suivant :

```
@a1     i = 0
@a2     ...
...     ...
@an     i = i + 1
@an+1   si i < N alors BR @a2
```

– au premier passage dans la boucle, 2 cas peuvent se produire :
  – le branchement est prédit pris, et il est effectivement pris
  – le branchement est prédit non pris, et il est pris

RUPTURE PIPELINE

– à la dernière itération de la boucle, le bit de prédiction est à 1 (indique que le branchement doit être pris), le branchement est donc prédit pris, mais il ne sera pas pris

RUPTURE PIPELINE

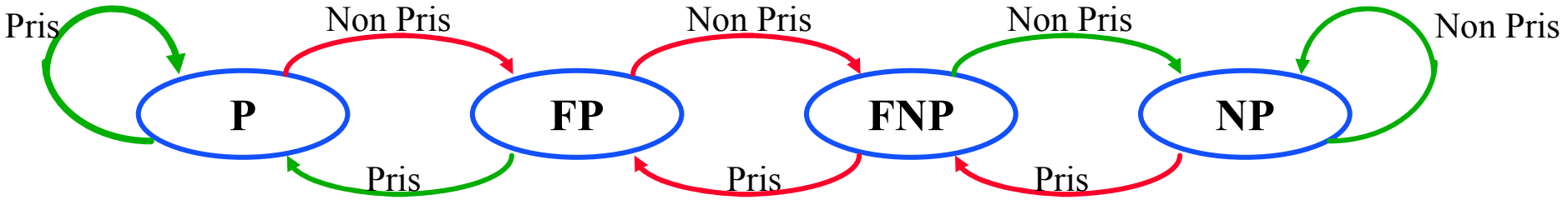
– au retour dans la boucle, lors de la première itération, le bit de prédiction est à 0 (dernier branchement non pris), donc le branchement est prédit non pris, mais il sera pris

RUPTURE PIPELINE

Augmentation de performances

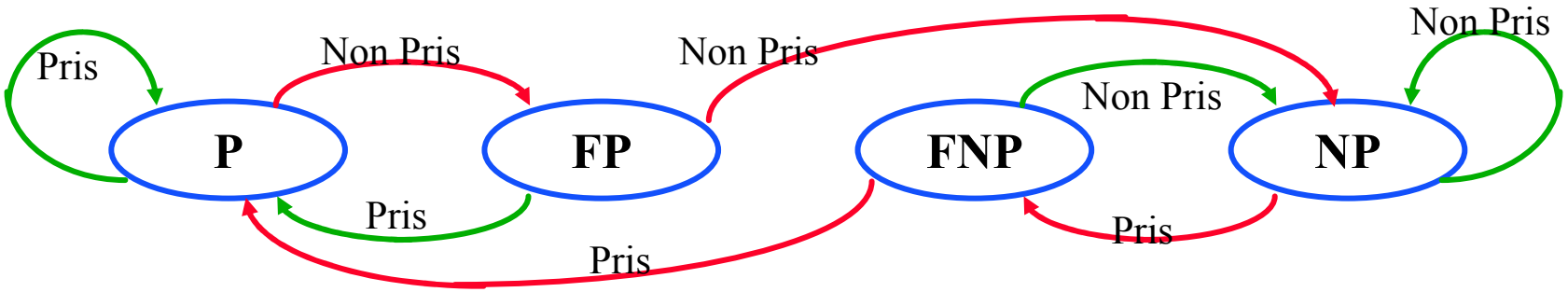
➢prédiction de branchement dynamique à 2 bits

- il s'agit d'un cas particulier d'un schéma plus général à base de compteur n bits à saturation
- un compteur n bits peut prendre ces valeurs entre 0 et $2^n - 1$
- si la valeur du compteur est supérieure à 2^{n-1} (la moitié de la valeur max) alors le branchement est prédit pris, sinon il est prédit non pris
- le compteur est incrémenté si le branchement est pris et décrémenté si le branchement est non pris
- des études ont montré que des prédicteurs à 2 bits sont presque aussi efficace que des prédicteurs n bits, et ils sont moins coûteux, donc la plupart des processeurs se limite à une prédiction à 2 bits
- ***Schéma 1 :***

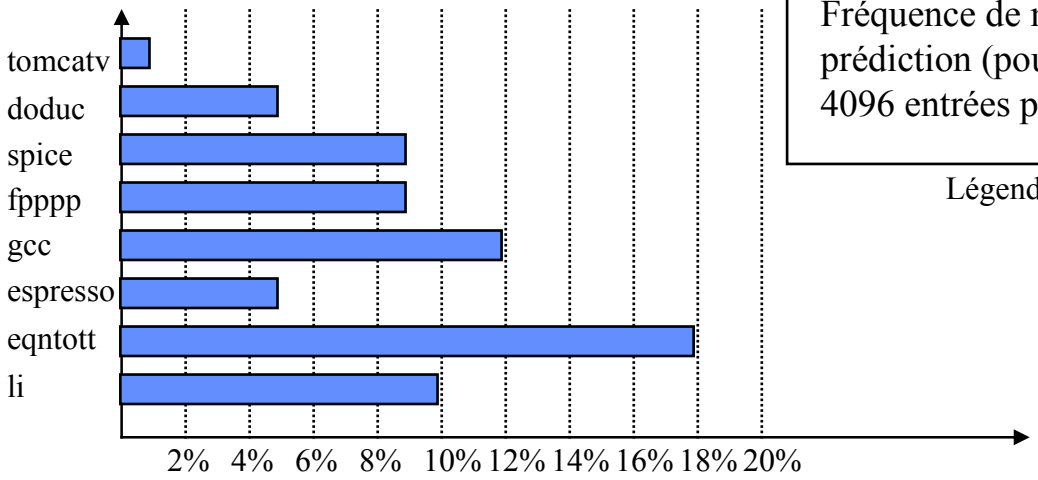


Augmentation de performances

– ***Schéma 2:***



– comportement vis à vis des boucles, très bonne prédiction :
  – une erreur de prédiction lors de la dernière itération de la boucle
  – pas d'erreur de prédiction au retour dans la boucle



Fréquence de mauvaise prédiction (pour un tampon de 2 bits à 4096 entrées pour les Spec89)

Légende :

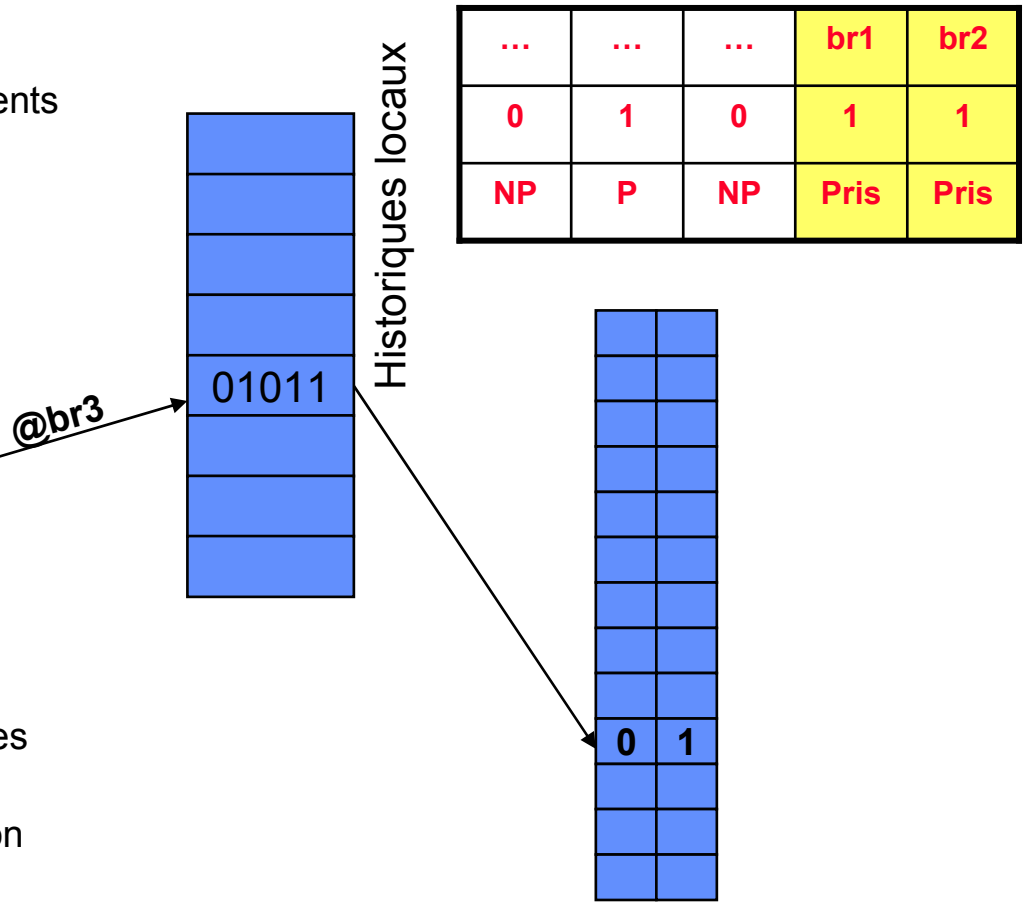
tomcatv : prg de génération de grille (vectorisable)
doduc : simulation monte carlo d'un réacteur nucléaire
spice : logiciel de simulation de circuit
fpppp : prg de chimie quantique calculant les dérivées intégrale de 2 électons
gcc : compilateur C GNU
espresso : minimisation de fonctions booleennes
eqntott : traduit une équation booléenne en table de vérité
li : interpréteur lisp résolvant le problème des 9 reines

Augmentation de performances

➢ Prédiction de branchement à corrélateur

- dynamique à 2 bits qui tient compte des autres branchements
- soit le code :

```
if ( a == 2 )                 // br1
        a = 0 ;
if ( b == 2 )                 // br 2
        b = 0;
if ( a != b )                 // br3
        ....
```



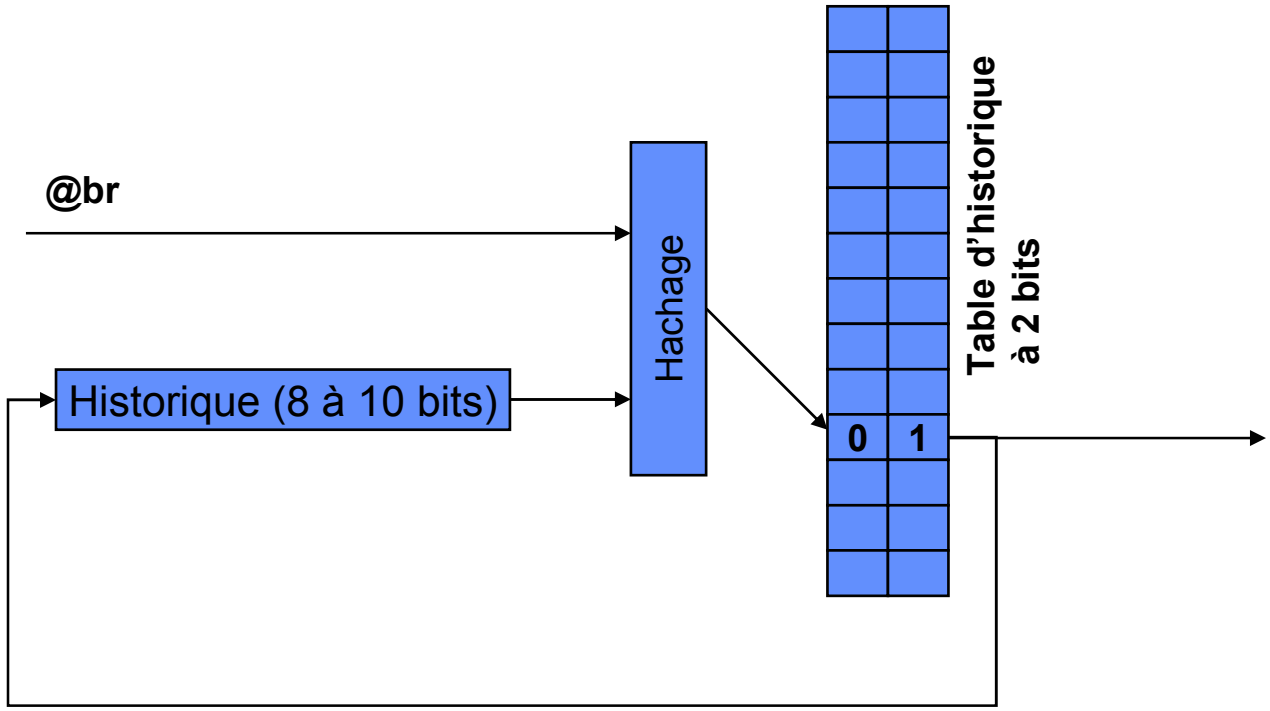
- pour le branchement br3, on mémorise le comportement des branchements précédents
- longueur de l'historique environ **10 bits**

Augmentation de performances

➢Prédiction de branchement à deux niveaux :

♦ on combine l'adresse du branchement avec le comportement des branchement précédent :

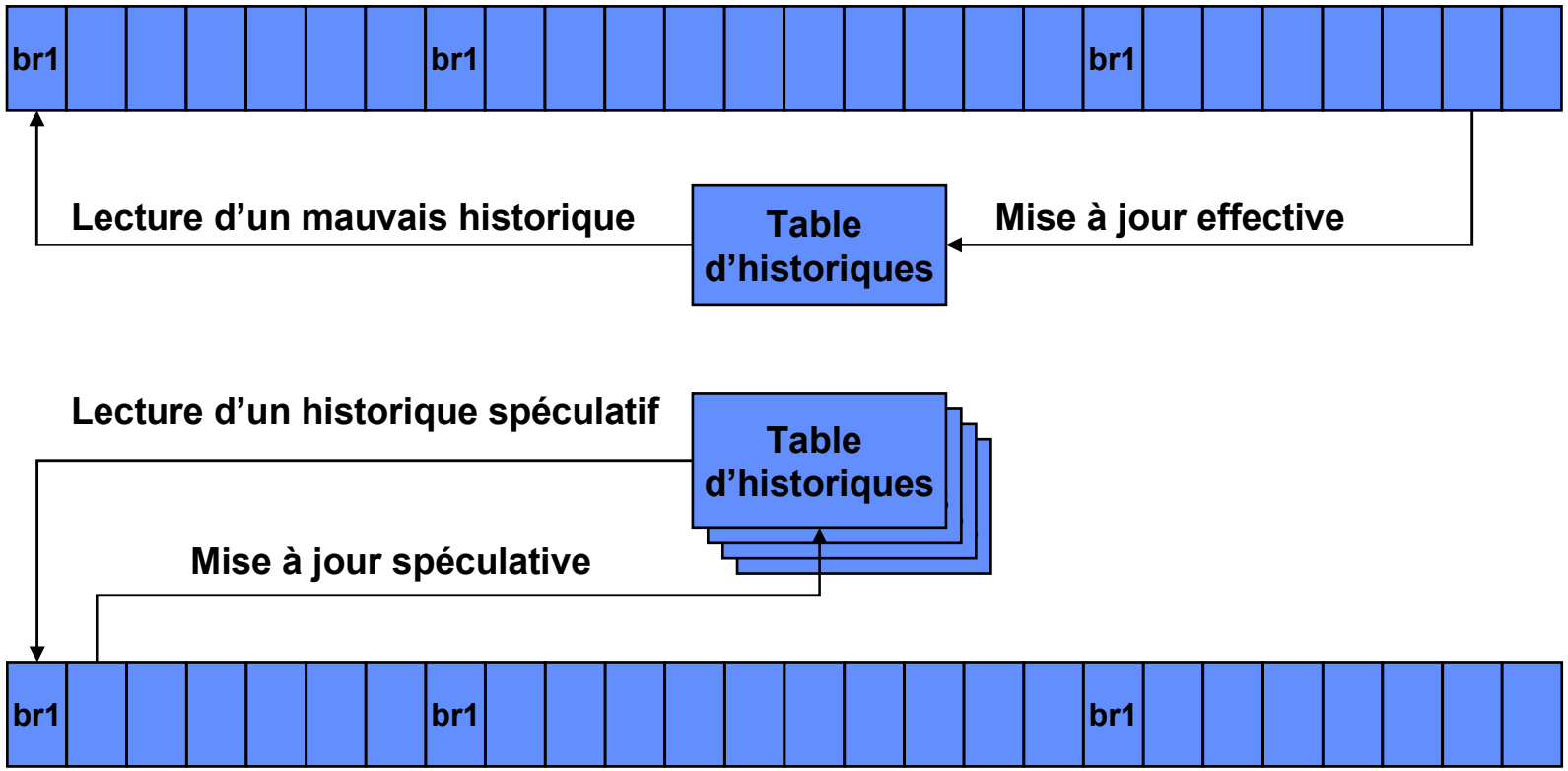
– on tombe sur un historique dynamique à 2 bits



Augmentation de performances

➢Quand mettre à jour la table d'historique ?

- ♦ si pipeline court, mise à jour lors de l'exécution ***effective*** de l'instruction
- ♦ si pipeline long (très long) mise à jour de façon ***spéculative***



- ♦ **il faut pouvoir défaire très vite les historiques**

Augmentation de performances

➢ Autre solution : déroulage de boucles lors de la compilation

- pour qu'un pipeline reste toujours plein, il faut exhiber le parallélisme
- dans un code de boucle, on peut dérouler (partiellement ou complètement) la boucle:

```
for (i=1 ; i<20 ; i=i+1) {
        x[i] = v1[i] + v2[i] ;
}
```

```
x[0] = v1[0] + v2[0] ;
x[1] = v1[1] + v2[1] ;
....
x[18] = v1[18] + v2[18] ;
x[19] = v1[19] + v2[19] ;
```

→ plus de branchement conditionnel, donc plus de perte de cycles

Augmentation de performances

➢ Autres solutions : branchement différé :

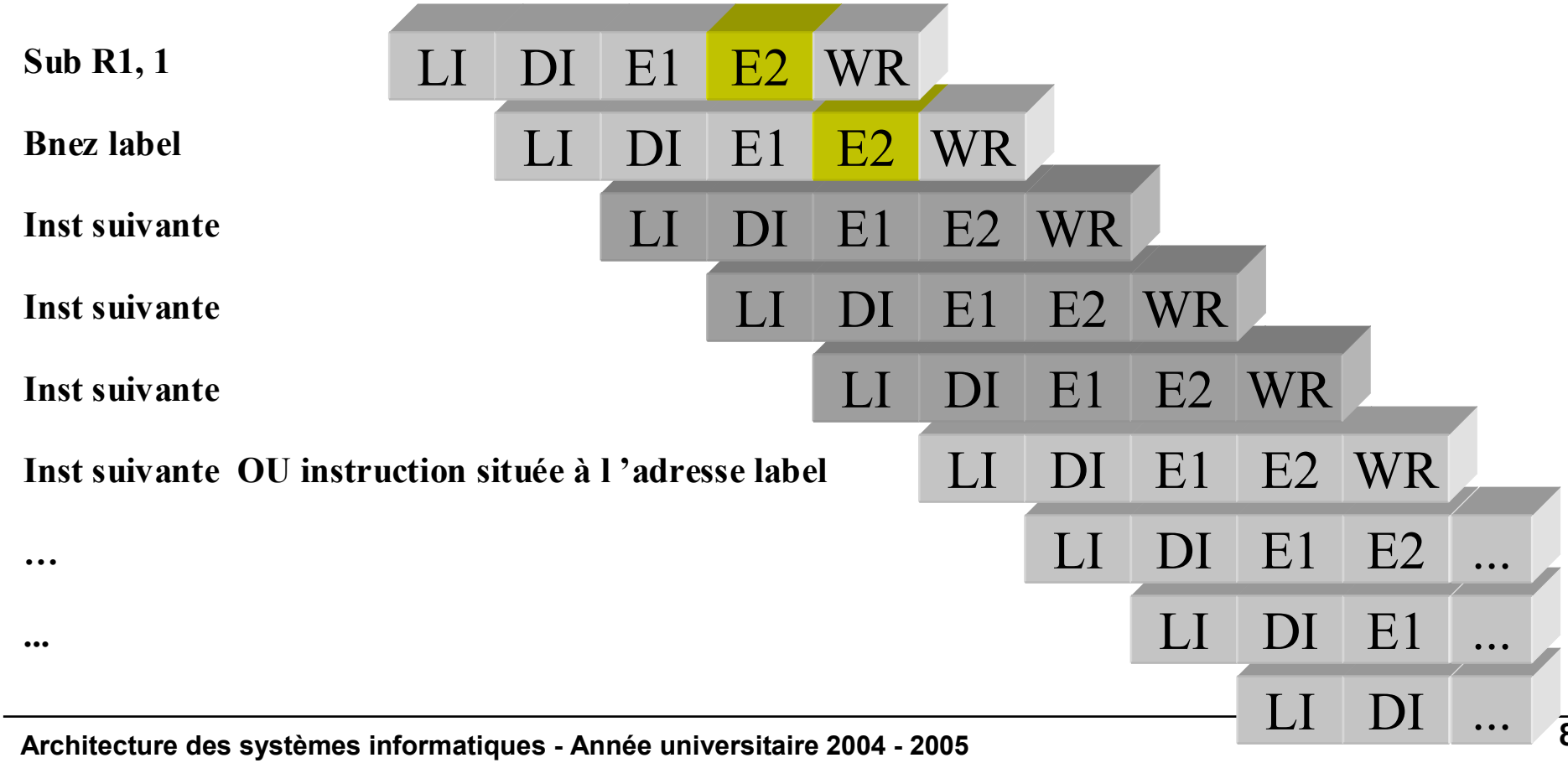
- ♦ notion de délai de branchement : n

  instruction de branchement
  successeur 1
  successeur 2
  ...
  successeur n

- ♦ les instructions successeurs sont exécutées que le branchement soit pris ou non
- ♦ en pratique toutes les machines à branchement différé ont un seul délai

Augmentation de performances

♦ Branchement différé, fonctionnement



Augmentation de performances

♦ Pour un branchement différé efficace :

– le délai de branchement doit être remplie efficacement :

• éviter les NOP

```
BI 1;
If condition then
            BI 2 ;
else
            BI 3 ;
end if;
BI 4;
```

Compilation non efficace

```
            BI 1
            Br condition, else
            NOP
            NOP
            NOP
            BI 2
            JUMP suite
else
            BI 3
suite
            BI 4
```

Compilation efficace

```
            BI 1
            Br condition, else
            Inst du BI 1
            Inst du BI 1
            Inst du BI 4
            BI 2
            JUMP suite
else
            BI 3
suite
            BI 4
```

Augmentation de performances

- Avantages des branchements différés :
  - mise en œuvre simple
  - ordonnancement par le compilateur assez simple à réaliser
  - convenait assez bien au premier processeur RISC, puisqu'il y avait des pipelines assez courts et des délais de branchement égaux à 1
- Inconvénients des branchements différés :
  - repose sur l'aspect implémentation (qui définit le délai de branchement)
  - l'allongement des pipelines ne permet plus d'ordonnancer simplement les opérations
  - perte de cycles importante

Augmentation de performances

➢Taille du tampon de prédiction :

- ♦ le taux d'efficacité de la prédiction dépend largement de la taille de ce tampon
- ♦ ce tampon fonctionne de la même manière qu'un cache :
  - – une adresse de branchement **@a1** qui n'a pas été utilisée depuis un certain temps est écrasée par une autre plus récente **@a2** (on perd donc la prédiction de branchement pour le branchement de l'adresse **@a1**)
- ♦ un tampon de faible taille augmente fortement le taux d'échec de la prédiction
- ♦ pour les programmes Spec89, avec un tampon de 4096 entrées, on obtient des taux d'échec variant entre 1 et 18 %.

Augmentation de performances

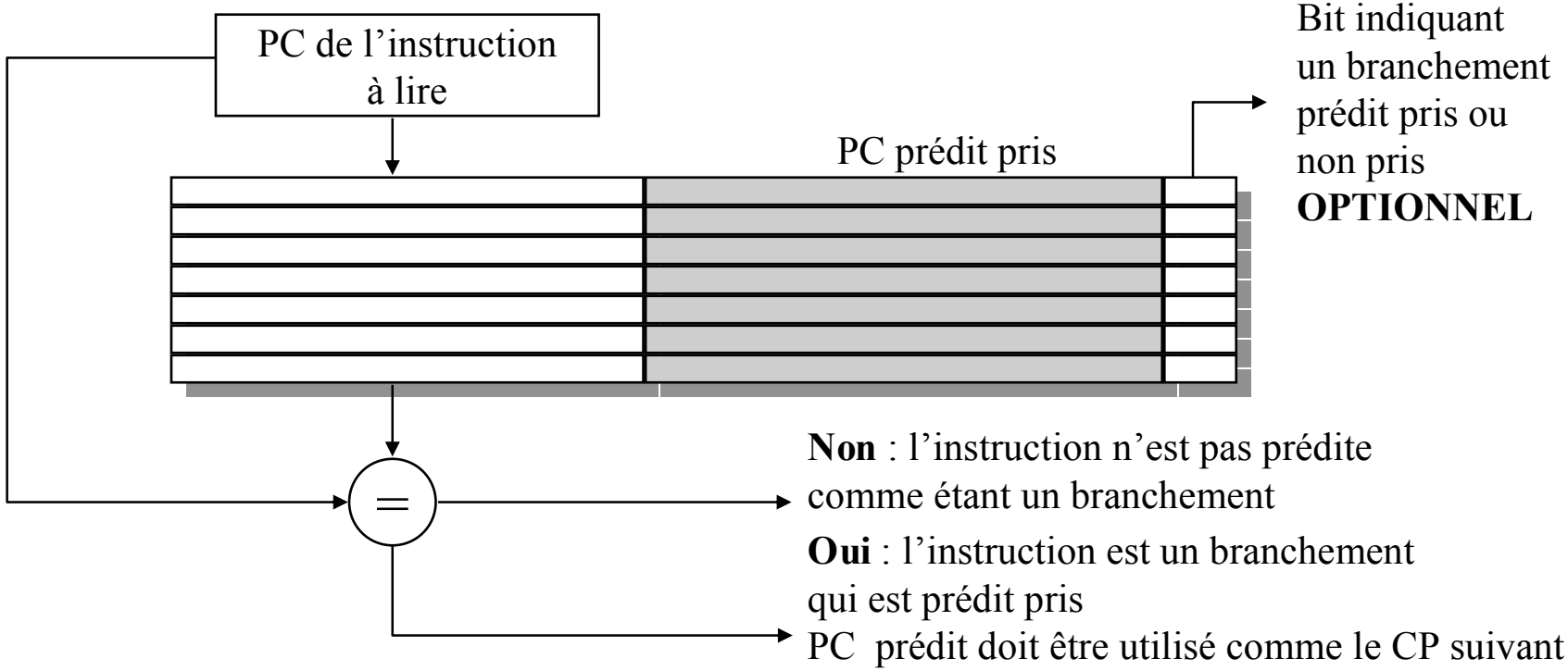
❑ Exemples d'implémentation :

- ➢ pas de prediction — Intel 8086
- ➢ prédiction statique
  - ♦ jamais pris — Intel i486
  - ♦ toujours pris — Sun SuperSPARC
  - ♦ arrière pris, avant non pris — HP PA-7x00
  - ♦ semi statique, par profiling — PowerPCs
- ➢ prédiction dynamique :
  - ♦ 1-bit — DEC Alpha 21064, AMD K5
  - ♦ 2-bit — PowerPC 604, MIPS R10000, Cyrix 6x86 and M2, NexGen 586
  - ♦ two-level adaptive — PentiumPro, Pentium II, AMD K6
- ➢ prédiction hybride — DEC Alpha 21264
- ➢ Predication — Intel/HP Itanium et les DSP ARM processors, TI TMS320C6201
- ➢ Eager execution (limited) — IBM mainframes: IBM 360/91, IBM 3090

Augmentation de performances

Tampon d'adresses de branchement

- l'intérêt est de connaître au plus tôt l'adresses de branchement
- fonctionnement similaire à celui d'un cache
- ce cache est indexé par l'adresse de l'instruction en cours de lecture (LI)
- on connaît donc l'adresse de l'instruction prédite dès la fin du cycle LI



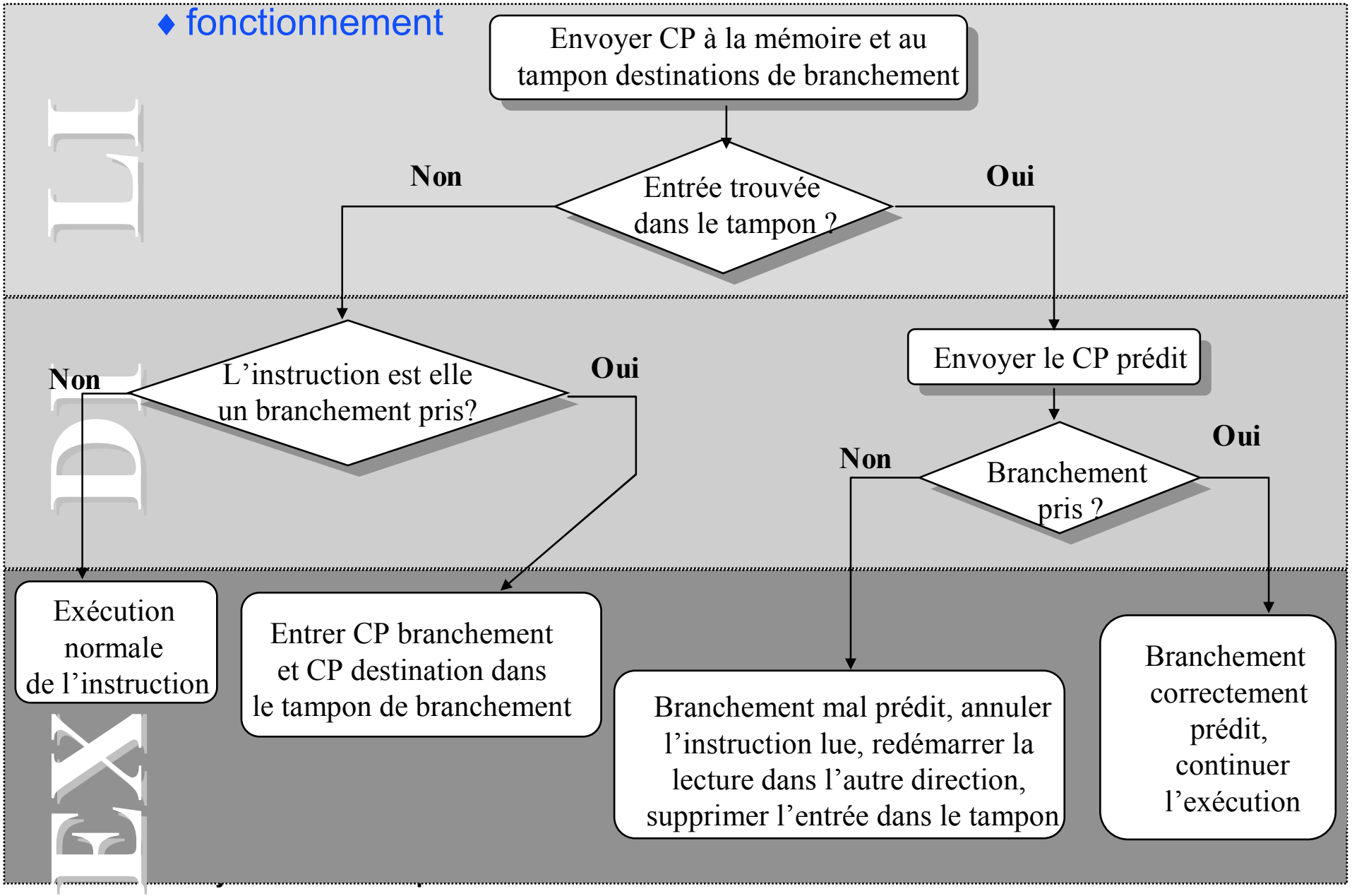
Augmentation de performances

➢ Fonctionnement :

- ♦ lorsque le processeur recherche l'instruction située à l'adresse @a, l'adresse @a est comparée à l'ensemble des adresses contenues dans le tampon.
- ♦ 2 cas peuvent alors se produire :
  - – l'adresse est présente dans le tampon, alors l'instruction que l'on est en train de lire est un branchement conditionnel qui a été pris lors de l'exécution précédent. Le branchement est alors prédit pris et l'instruction prédite suivante peut alors être chargée immédiatement.
  - – l'adresse n'est pas présente dans le tampon, l'instruction que l'on est en train de lire est une instruction classique ou une instruction de branchement conditionnel qui n'a pas été pris lors de l'exécution précédente.

Augmentation de performances

♦ fonctionnement



Augmentation de performances

❑ Predicated instructions :

- ♦ instructions conditionnées à la valeur d'un registre prédicat
- ♦ permet de supprimer des branchements conditionnels
- ♦ instructions classiques + :
  - – un registre valant vraie ou faux
  - – si le registre contient vraie alors l'instruction est exécutée
  - – sinon l'instruction se comporte comme un NOP

```
If (x==0) {
        a = b +c;
        d = e - f;
}
g = h *i;
```

```
(Pred = (x == 0))
if (Pred) then a = b + c;
if (Pred) then d = e - f;
g = h * i;
```

- ♦ complique l'exécution des instructions
- ♦ nécessite l'ajout de ports sur la file de registres
- ♦ les instructions qui se comporteront comme des NOP consomment des ressources processeur
- ♦ intéressant pour les petites structures conditionnelles if then else

Augmentation de performances

❑ Trace Cache :

➢allongement des pipelines :

♦ conséquence : probabilité d'avoir plusieurs instructions de branchement dans le pipeline importante :

– plusieurs prédictions de branchement en cours

➢objectif du cache de trace :

♦ capturer la séquence d'instructions exécutée :

– de façon dynamique

• suite de blocs exécutés par le processeur

♦ différence avec un cache d'instructions :

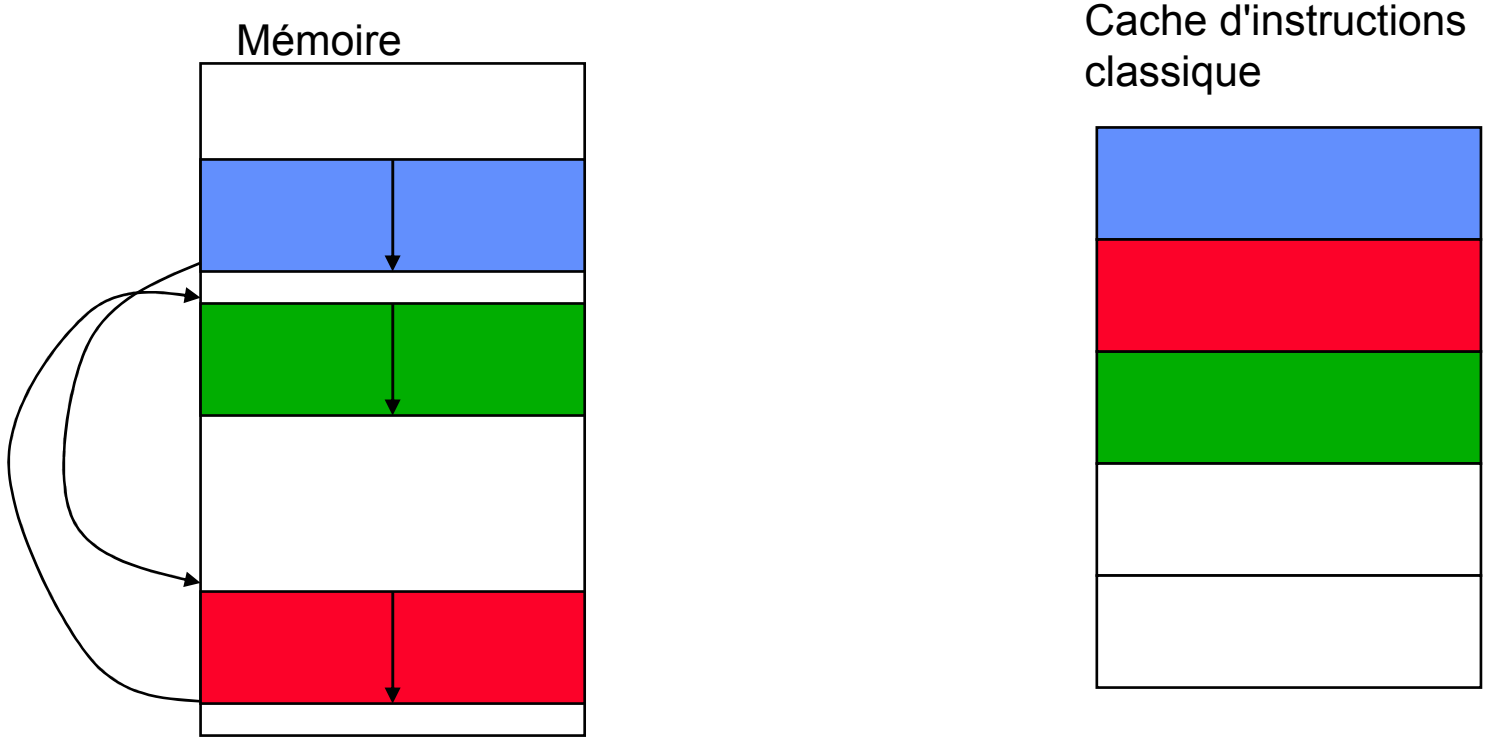
– le cache d'instructions mémorise une séquence statique d'instructions :

• suite de blocs consécutifs dans la mémoire

Augmentation de performances

➢Fonctionnement :

- ♦ soit le programme suivant en mémoire
- ♦ et soit l'exécution :
  - – bloc instructions 1
  - – appel à une fonction : bloc instructions 2
  - – bloc instructions 3

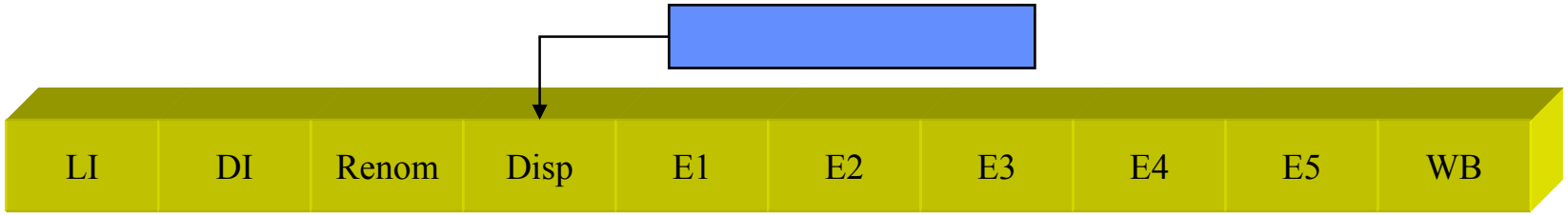


Augmentation de performances

- la ligne du cache contient la séquence des instructions exécutées :
  - y compris les différents branchements conditionnels ou inconditionnels

Cache de trace

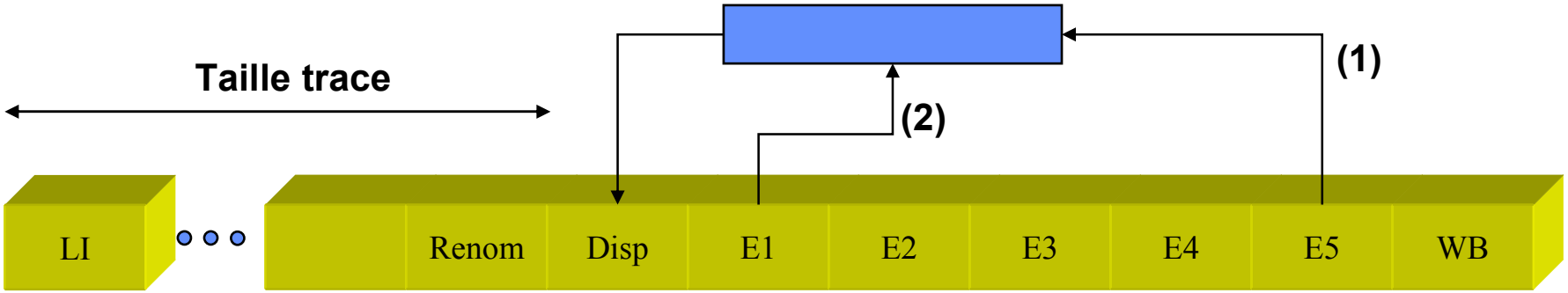
- si le processeur recommence la même séquence :
  - alors le trace de cache fournit la séquence d'instructions précédemment exécutée
  - pas de prédiction de branchement
  - si le processeur ne répète pas la même séquence, alors il faut stocker une nouvelle trace d'exécution



Augmentation de performances

➢ on cache les traces effectives (1) ou les traces spéculatives (2) ?

- ♦ plus le pipeline est long, plus on a intérêt à cacher des traces spéculatives
- ♦ si on attend des derniers étages du pipeline pour mémoriser la trace, alors il y a de fortes chance que l'on loupe la reprise d'une trace



➢Taille de la trace à stocker ?

- ♦ suffisante pour que le temps de traiter la trace permette de préparer la première instruction suivant la trace

Augmentation de performances

❑ Exécution spéculative :

➢idée générale :

- ♦ les instructions génèrent beaucoup de valeurs prédictibles
- ♦ le processeur est parfois bloqué par l'indisponibilité d'une opérande

➢exemple :

```
add     R1, R2, R3      // R1 = R2 + R3
mult    R4, R1, R6      // R4 = R1 * R6
```

- ♦ la dépendance sur R1 bloque l'exécution de la multiplication

➢solution :

- ♦ si l'addition a déjà été calculée :
  - – alors on peut poser comme hypothèse que le résultat de l'addition sera le même
  - – donc on peut exécuter spéculativement la multiplication
  - – lorsque l'addition sera terminée, deux possibilités :
    - le résultat est égal à celui spéculé ===> rien de particulier à faire
    - le résultat est différent de celui spéculé ===> il faut refaire le calcul

Renommage de registres

- Objectifs : casser les dépendances entre différentes données afin d'utiliser au mieux le pipeline
- Classification des aléas de données :

  - LAE : lecture après écriture :

```
add R1, R2, R3        // R1 := R2 + R3
add R5, R1, R8        // R5 := R1 + R8
```

  - la deuxième instruction lit l'ancienne valeur de R1
  - c'est le type d'aléas le plus courant

Augmentation de performances

♦ EAE : écriture après écriture :

– existe uniquement si le pipeline du ***mult*** est plus long que le pipeline de l'***add***
– ordre des écritures inversé, la valeur écrite dans R1 est le résultat du ***mult*** plutôt que le résultat du ***add***

```
mult   R1, R2, R3      // R1 :=R2 * R3
add    R1, R5, R8      // R1 := R5 +  R8
```

MOVE R1, 0(R2)	LI	DI	EX 1	EX 2	EX 3	MEM	ER
ADD R1, R5, R8		LI	DI	EX	MEM	ER	

Augmentation de performances

- EAL : écriture après lecture :
  - les pipelines :
    - lisent très tôt les données à traiter
    - écrivent très tard les données traitées

```
add   R2, R1, R3        // R2 := R1 + R3
add   R1, R4, R5        // R1 := R4 + R5
```

  - Le problème existe toutefois si le processeur à la possibilité d'exécution dans le désordre

- LAL : lecture après lecture, ce n'est pas un aléas de fonctionnement :

```
add   R2, R1, R3        // R2 := R1 + R3
add   R4, R1, R5        // R4 := R1 + R5
```

  - pas de soucis particuliers

Augmentation de performances

➢Solutions :

- LAE : pas de solution en effectuant du renommage de registres, la solution consiste alors à intercaler des instructions indépendantes là où c'est nécessaire
- EAE : on peut renommer le registre dans lequel est effectué la dernière écriture

```
mult   R1, R2, R3      // R1 :=R2 * R3
add    R'1, R5, R8     // R'1 := R5 +  R8
```

- EAL : idem précédent

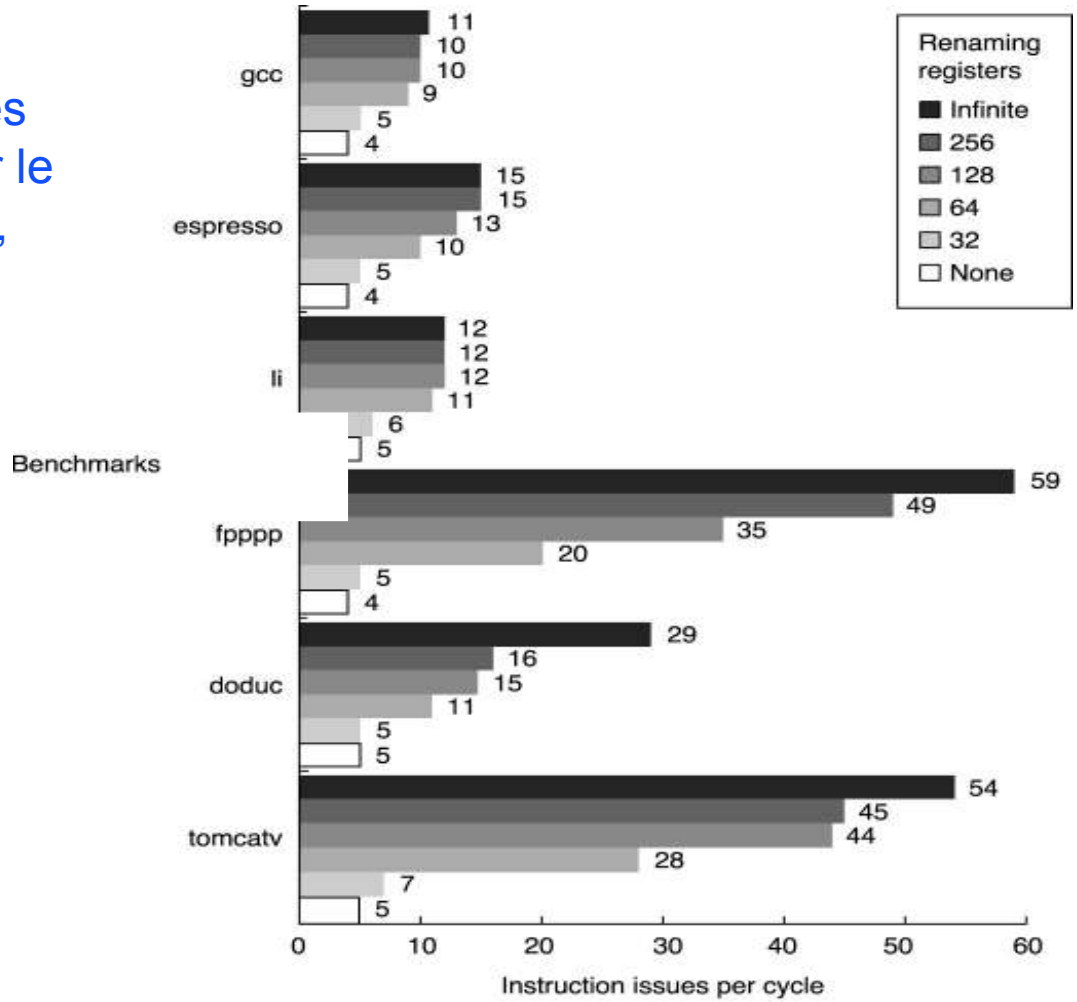
```
add   R2, R1, R3       // R2 :=  R1 + R3
add   R'1, R4, R5      //  R'1 := R4 + R5
```

- Solution idéale : disposer d'un nombre infini de registres virtuels
- Effets du nombre fini de registres : limite le nombre d'instructions qu'il est possible de lancer simultanément

Augmentation de performances

➢ Effet de nombre de registres sur l'ILP :

- ♦ plus le nombre de registres physique disponibles pour le renommage est important, plus l'ILP augmente



Augmentation de performances

Exécution dans le désordre :

- motivations :
  - assurer au mieux le remplissage des pipelines de toutes les unités disponibles
  - permettre le maximum de parallélisme
- Pour un processeur superscalaire de degré N, les dépendances de données ou d'instructions provoque des blocages des pipelines
- Il s'agit de réaliser un ***ordonnancement dynamique*** des opérations en fonction de la disponibilité des ressources
- Pour un processeur superscalaire de degré N, c'est donc la possibilité d'exécuter ***toujours*** N instructions par cycle
- 2 façons de réaliser l'exécution spéculative :
  - statique :
    - très souvent réaliser par le logiciel, le compilateur :
      - difficile à réaliser lorsqu'il y a manipulation de pointeurs
    - pas optimale, surtout lorsque le processeur supporte un système d'exploitation multitâches (chaque tâche est ordonnancée au mieux, mais l'ensemble !!!)

Augmentation de performances

- dynamique :
  - par le matériel, logique de contrôle très importante gérant les émission d'instructions vers les différentes unités
  - beaucoup plus souple

➢ D'une façon générale l'exécution dans le désordre est très liée à :

- la prédiction de branchement
  - très bons résultats si la prédiction matérielle de branchement est bonne
- la détection des dépendances :
  - renommage des registres, afin de casser les dépendances de données
- la taille des buffers (*locaux et globaux)* de préchargement d'instructions :
  - plus le buffer est important, plus le processeur a de possibilité pour lancer des instructions aux différents cycles

Augmentation de performances

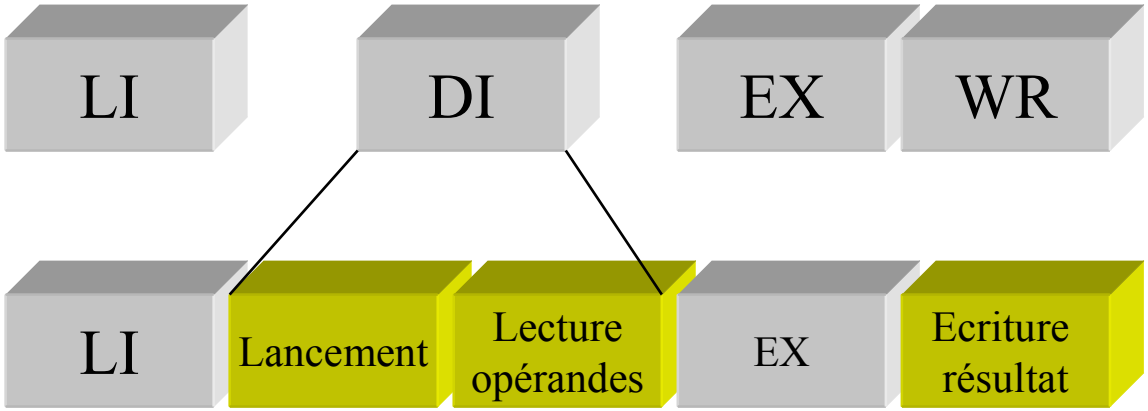
➢Tous les étages ne sont pas exécutés dans le désordre :

- ♦ dans l'ordre :
  - – Lecture d'instruction
  - – Décodage d'instruction
  - – Renommage de registres
- ♦ dans le désordre :
  - – exécution des instructions
- ♦ dans l'ordre :
  - – mise à jour état processeur
  - – écriture dans les registres destination et/ou mémoire

Augmentation de performances

➢Le tableau des marques :

- ♦ il s'agit d'une structure de données permettant :
  - – d'assurer l'exécution non ordonnée des instructions
  - – de détecter les aléas
- ♦ il détermine quand les unités fonctionnelles :
  - – peuvent lire leurs opérandes sources
  - – peuvent commencer leur exécution
  - – peuvent écrire leur résultat dans le registre destination



Pipeline classique

Pipeline avec tableau des marques

Augmentation de performances

♦ Les étapes du pipeline :

– **Lancement :**

- **Si** une unité fonctionnelle est libre pour exécuter l'instruction
- **et si** il n'y a pas d'autre instruction active qui a le même registre destination (aléas de type EAE) :

```
ADD     R0, R1, R2
SUB     R0, R3, R4
```

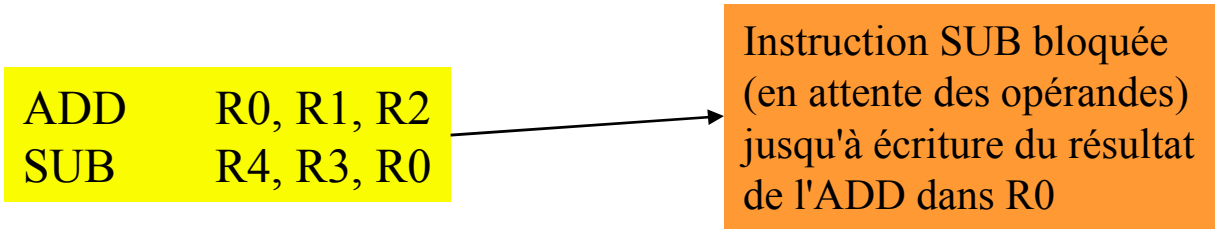
→ **Instruction SUB suspendue (non lançée) jusqu'à écriture du résultat de l'ADD dans R0**

- **alors** le tableau des marques lance l'instruction
- **et** effectue une mise à jour de sa structure de données
- **Sinon**, l'instruction est suspendue jusqu'à disparition de la dépendance

Augmentation de performances

– **Lecture opérandes :**

- un opérande source est disponible si aucune autre instruction active lancée auparavant ne va le modifier (détection des dépendances de type LAE)

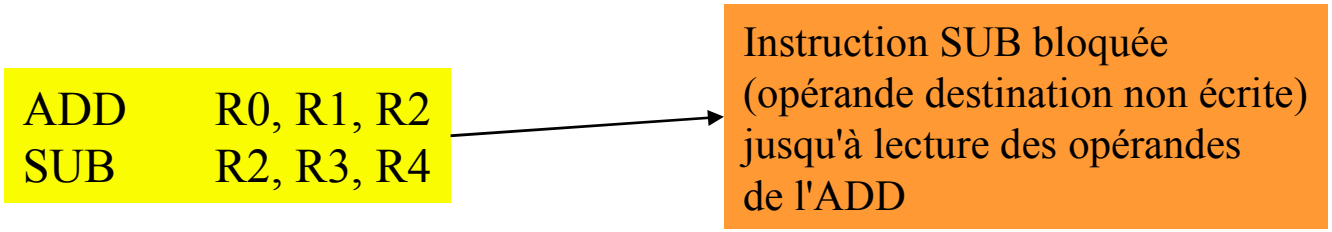


- **lorsque tous les opérandes sont disponibles**, l'instruction peut passer dans l'étage d'exécution

Augmentation de performances

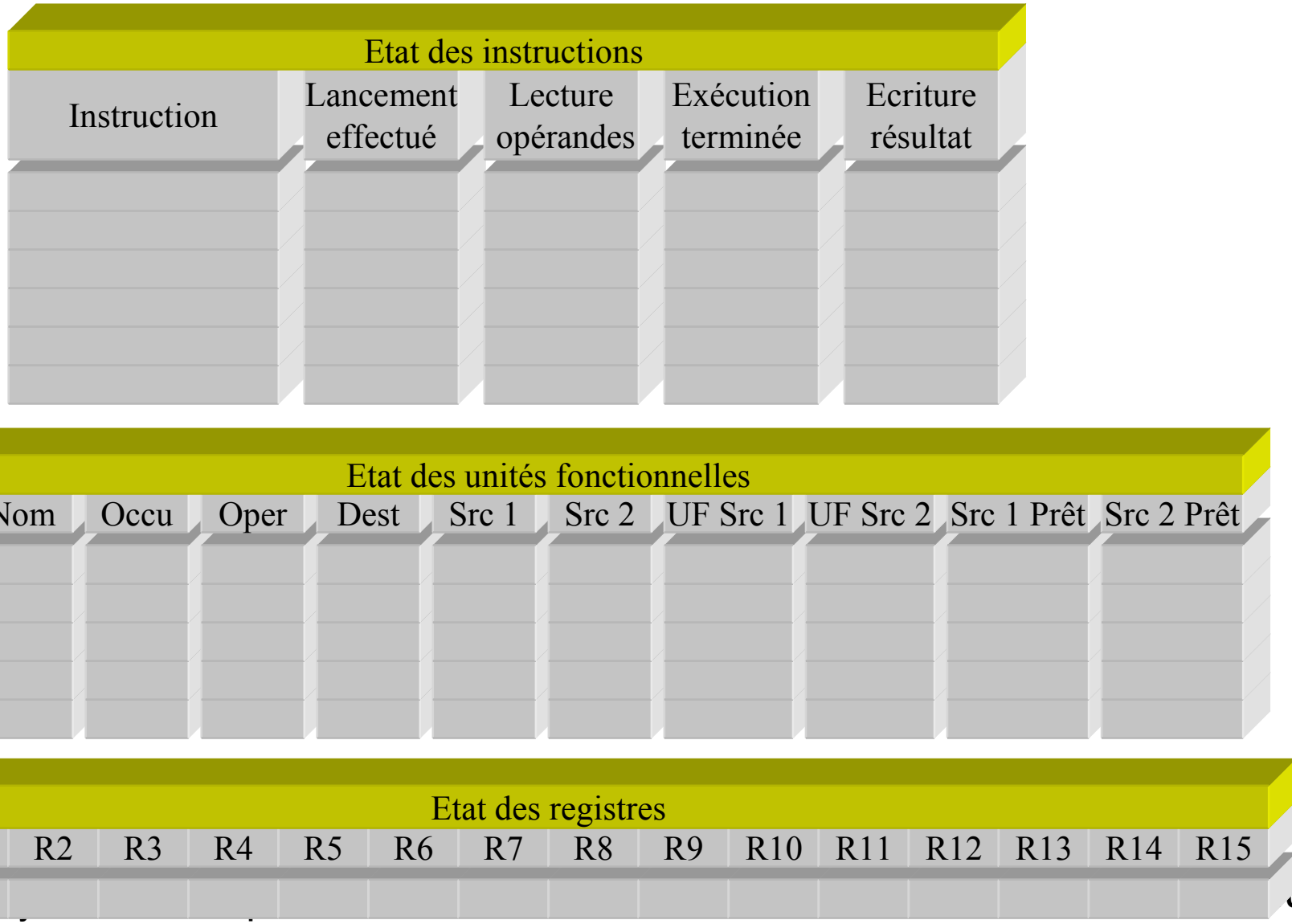
– **Exécution instruction :**
  - lorsque l'unité fonctionnelle a terminé, elle l'indique au tableau des marques
  - l'unité fonctionnelle **n'effectue pas** l'écriture dans l'opérande destination

– **Ecriture résultat :**
  - le tableau des marques **vérifie** qu'il n'existe pas une instruction lancée auparavant qui n'aurait pas encore lue ses opérandes (dépendances de type EAL)



Augmentation de performances

➢Représentation du tableau des marques



Augmentation de performances

➢Exemples :

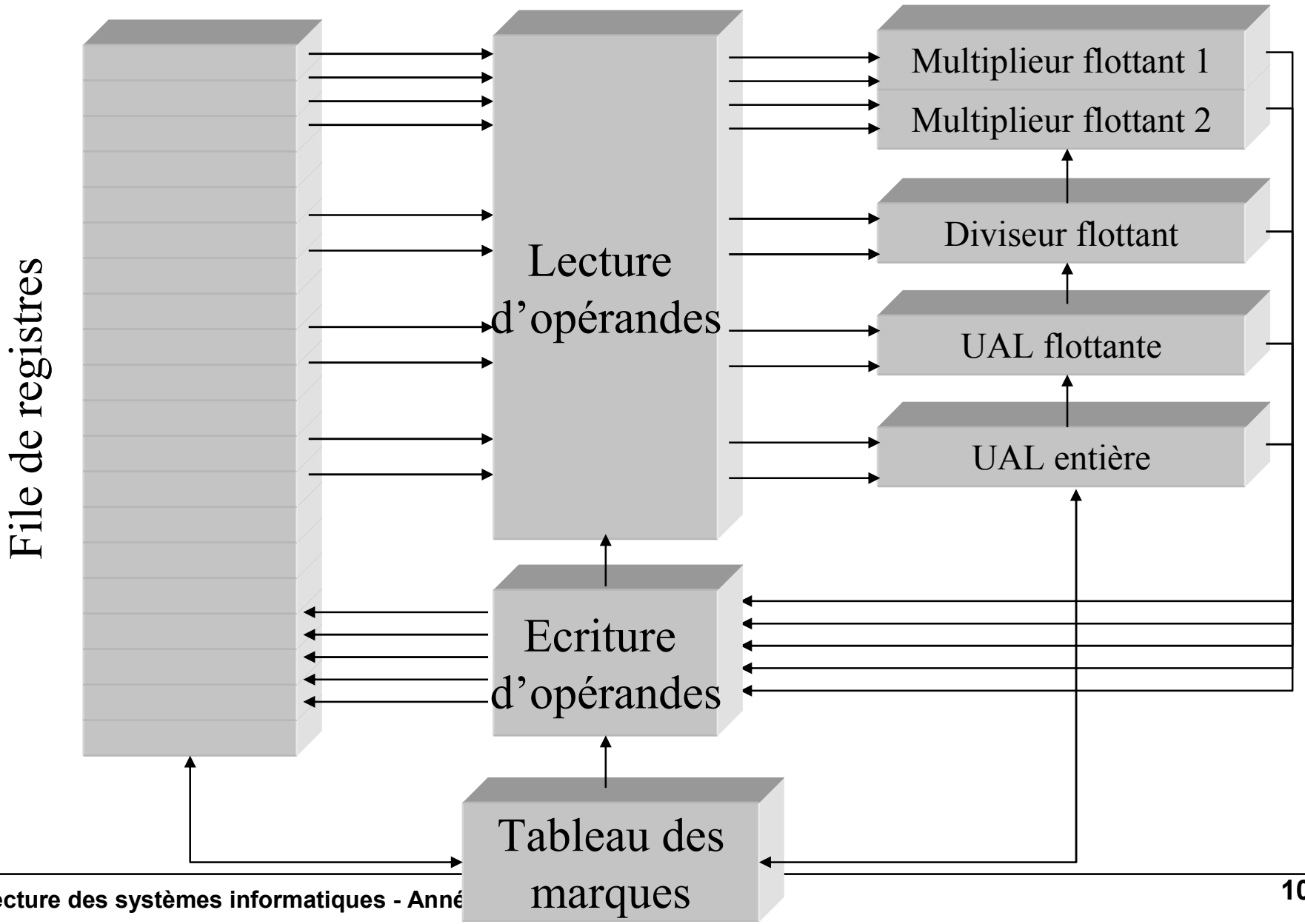
♦ on admettra que le processeur :

– analyse une fenêtre d'instruction de taille 6

– possède :

- 1 unité entière UAL (assurant aussi les accès mémoire)
- 1 unité flottante UAL
- 2 unités de multiplication flottante
- 1 unité de division flottante

Augmentation de performances



Augmentation de performances

➢ Exemple 1 : code avec dépendances

♦ soit le code suivant :

- LOAD **R2**, (R3)
- LOAD **R6**, (R2)
- MULTF **R0**, **R2**, R4
- SUBF R8, **R6**, **R2**
- DIVF R10, **R0**, **R6**
- ADDF **R6**, R8, R2

LAE sur :
- R6
- R2
- R0

EAL sur :
- R6

Augmentation de performances

♦ Temps de cycle 1 : analyse des 6 instructions

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#			
LOAD	**R6, (R2)**				
MULTF	**R0, R2, R4**	#			
SUBF	**R8, R6, R2**	#			
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> **Unité UAL libre**
> **Plus d'UAL dispo**
> **Unité Mult libre**
> **Unité UALF libre**
> **Unité Div libre**
> **Plus d'UALF dispo**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R2	R3				Oui	
2	Mult1	#	multf	R0	R2	R4	UAL		Non	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2		UAL	Non	Non
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1		UAL				UAL		UALF		Div					

Augmentation de performances

♦ Temps de cycle 2

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#		
LOAD	**R6, (R2)**				
MULTF	**R0, R2, R4**	#			
SUBF	**R8, R6, R2**	#			
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> Lecture opérandes
> Plus d'UAL dispo
> Opérandes non dispo
> Opérandes non dispo
> Opérandes non dispo
> Plus d'UALF dispo

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R2	R3				*Oui*	
2	Mult1	#	multf	R0	R2	R4			Non	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	Non
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1		UAL				UAL		UALF		Div					

Augmentation de performances

♦ Temps de cycle 3

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#		**> Exécution terminée**
LOAD	**R6, (R2)**					**> UAL dispo**
MULTF	**R0, R2, R4**	#				**> Opérandes non dispo**
SUBF	**R8, R6, R2**	#				**> Opérandes non dispo**
DIVF	**R10, R0, R6**	#				**> Opérandes non dispo**
ADDF	**R6, R8, R2**					**> Plus d'UALF dispo**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R2	R3				Oui	
2	Mult1	#	multf	R0	R2	R4			Non	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	Non
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1		UAL						UALF		Div					

Augmentation de performances

♦ Temps de cycle 4

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#	#	> **Ecriture résultat**
LOAD	**R6, (R2)**	#				> **Opérandes non dispo**
MULTF	**R0, R2, R4**	#				> **Opérandes non dispo**
SUBF	**R8, R6, R2**	#				> **Opérandes non dispo**
DIVF	**R10, R0, R6**	#				> **Opérandes non dispo**
ADDF	**R6, R8, R2**					> **Plus d'UALF dispo**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R6	R2				Oui	
2	Mult1	#	multf	R0	R2	R4			***Oui***	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	***Oui***
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1		***UAL***				UAL		UALF		Div					

Augmentation de performances

♦ Temps de cycle 5

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#	#	**> Instruction terminée**
LOAD	**R6, (R2)**	#	#			**> Opérandes disponibles**
MULTF	**R0, R2, R4**	#	#			**> Opérandes disponibles**
SUBF	**R8, R6, R2**	#				**> Opérandes non dispo**
DIVF	**R10, R0, R6**	#				**> Opérandes non dispo**
ADDF	**R6, R8, R2**					**> Plus d'UALF dispo**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R6	R2				Oui	
2	Mult1	#	multf	R0	R2	R4			Oui	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1						UAL		UALF		Div					

Augmentation de performances

♦ Temps de cycle 6

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#	#	#
LOAD	**R6, (R2)**	#	#	#	
MULTF	**R0, R2, R4**	#	#		
SUBF	**R8, R6, R2**	#			
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> Instruction terminée
> Exécution terminée
> Cycle 1 (3 cycles)
> Opérandes non dispo
> Opérandes non dispo
> Plus d'UALF dispo

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R6	R2				Oui	
2	Mult1	#	multf	R0	R2	R4			Oui	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1						UAL		UALF		Div					

Augmentation de performances

♦ Temps de cycle 7

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#	#	#
LOAD	**R6, (R2)**	#	#	#	#
MULTF	**R0, R2, R4**	#	#		
SUBF	**R8, R6, R2**	#			
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> Instruction terminée
> Ecriture résultat
> Cycle 2 (3 cycles)
> Opérandes non dispo
> Opérandes non dispo
> Plus d'UALF dispo

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R6	R2				Oui	
2	Mult1	#	multf	R0	R2	R4			Oui	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		***Oui***	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	***Oui***

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1						***UAL***		UALF		Div					

Augmentation de performances

♦ Temps de cycle 8

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#	#	#
LOAD	**R6, (R2)**	#	#	#	#
MULTF	**R0, R2, R4**	#	#	#	
SUBF	**R8, R6, R2**	#	#		
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> Instruction terminée
> Instruction terminée
> Exécution terminée
> Opérandes disponibles
> Opérandes non dispo
> Plus d'UALF dispo

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1	#	multf	R0	R2	R4			Oui	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Oui	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1								UALF		Div					

Augmentation de performances

♦ Temps de cycle 9

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#	#	**> Instruction terminée**
LOAD	**R6, (R2)**	#	#	#	#	**> Instruction terminée**
MULTF	**R0, R2, R4**	#	#	#	#	**> Ecriture résultat**
SUBF	**R8, R6, R2**	#	#	#		**> Exécution terminée**
DIVF	**R10, R0, R6**	#				**> Opérandes non dispo**
ADDF	**R6, R8, R2**					**> Plus d'UALF dispo**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1	#	multf	R0	R2	R4			Oui	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Oui	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	***Oui***	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1								UALF		Div					

Augmentation de performances

♦ Temps de cycle 10

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#	#	**> Instruction terminée**
LOAD	**R6, (R2)**	#	#	#	#	**> Instruction terminée**
MULTF	**R0, R2, R4**	#	#	#	#	**> Instruction terminée**
SUBF	**R8, R6, R2**	#	#	#	#	**> Ecriture résultat**
DIVF	**R10, R0, R6**	#	#			**> Opérandes disponibles**
ADDF	**R6, R8, R2**					**> Plus d'UALF dispo**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Oui	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	***Oui***	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
								UALF		Div					

Augmentation de performances

♦ Temps de cycle 11

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#	#	#
LOAD	**R6, (R2)**	#	#	#	#
MULTF	**R0, R2, R4**	#	#	#	#
SUBF	**R8, R6, R2**	#	#	#	#
DIVF	**R10, R0, R6**	#	#		
ADDF	**R6, R8, R2**	#			

> Instruction terminée
> Instruction terminée
> Instruction terminée
> Instruction terminée
> Cycle 1 (6 cycles)
> Unité UALF dispo

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2									
4	UALF	#	addf	R6	R8	R2			Oui	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Oui	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
						UALF				Div					

Augmentation de performances

♦ Temps de cycle 12

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#	#	#
LOAD	**R6, (R2)**	#	#	#	#
MULTF	**R0, R2, R4**	#	#	#	#
SUBF	**R8, R6, R2**	#	#	#	#
DIVF	**R10, R0, R6**	#	#		
ADDF	**R6, R8, R2**	#	#		

> Instruction terminée
> Instruction terminée
> Instruction terminée
> Instruction terminée
> Cycle 2 (6 cycles)
> Opérandes disponibles

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2									
4	UALF	#	addf	R6	R8	R2			Oui	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Oui	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
						UALF				Div					

Augmentation de performances

♦ Temps de cycle 13

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#	#	**> Instruction terminée**
LOAD	**R6, (R2)**	#	#	#	#	**> Instruction terminée**
MULTF	**R0, R2, R4**	#	#	#	#	**> Instruction terminée**
SUBF	**R8, R6, R2**	#	#	#	#	**> Instruction terminée**
DIVF	**R10, R0, R6**	#	#			**> Cycle 3 (6 cycles)**
ADDF	**R6, R8, R2**	#	#	#		**> Exécution terminée**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2									
4	UALF	#	addf	R6	R8	R2			Oui	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Oui	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
						UALF				Div					

Augmentation de performances

♦ Temps de cycle 14

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#	#	**> Instruction terminée**
LOAD	**R6, (R2)**	#	#	#	#	**> Instruction terminée**
MULTF	**R0, R2, R4**	#	#	#	#	**> Instruction terminée**
SUBF	**R8, R6, R2**	#	#	#	#	**> Instruction terminée**
DIVF	**R10, R0, R6**	#	#			**> Cycle 4 (6 cycles)**
ADDF	**R6, R8, R2**	#	#	#	#	**> Ecriture résultat**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2									
4	UALF	#	addf	R6	R8	R2			Oui	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Oui	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
						UALF				Div					

Augmentation de performances

♦ Temps de cycle 15

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#	#	**> Instruction terminée**
LOAD	**R6, (R2)**	#	#	#	#	**> Instruction terminée**
MULTF	**R0, R2, R4**	#	#	#	#	**> Instruction terminée**
SUBF	**R8, R6, R2**	#	#	#	#	**> Instruction terminée**
DIVF	**R10, R0, R6**	#	#			**> Cycle 5 (6 cycles)**
ADDF	**R6, R8, R2**	#	#	#	#	**> Instruction terminée**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2									
4	UALF	#								
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Oui	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
										Div					

Augmentation de performances

♦ Temps de cycle 16

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#	#	**> Instruction terminée**
LOAD	**R6, (R2)**	#	#	#	#	**> Instruction terminée**
MULTF	**R0, R2, R4**	#	#	#	#	**> Instruction terminée**
SUBF	**R8, R6, R2**	#	#	#	#	**> Instruction terminée**
DIVF	**R10, R0, R6**	#	#	#		**> Exécution terminée**
ADDF	**R6, R8, R2**	#	#	#	#	**> Instruction terminée**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2									
4	UALF									
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Oui	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
										Div					

Augmentation de performances

♦ Temps de cycle 17

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#	#	**> Instruction terminée**
LOAD	**R6, (R2)**	#	#	#	#	**> Instruction terminée**
MULTF	**R0, R2, R4**	#	#	#	#	**> Instruction terminée**
SUBF	**R8, R6, R2**	#	#	#	#	**> Instruction terminée**
DIVF	**R10, R0, R6**	#	#	#	#	**> Ecriture résultat**
ADDF	**R6, R8, R2**	#	#	#	#	**> Instruction terminée**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2									
4	UALF									
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Oui	Oui

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
										Div					

Augmentation de performances

➢Exemple 2 : code pouvant être exécuté dans le désordre

♦ soit le code suivant :

- LOAD R1, (R2)
- MULTF R0, R1, R3
- ADDF R4, R5, R6
- MULTF R7, R5, R6

LAE sur :
- R1

Augmentation de performances

♦ Temps de cycle 1

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R1, (R2)**	#			
MULTF	**R0, R1, R3**	#			
ADDF	**R4, R5, R6**	#			
MULTF	**R7, R5, R6**	#			

> **Unité UAL libre**
> **Unité Mult1 libre**
> **Unité UALF libre**
> **Unité Mult2 libre**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R1	R2				Oui	
2	Mult1	#	multf	R0	R1	R3	UAL		Non	Oui
3	Mult2	#	multf	R7	R5	R6			Oui	Oui
4	UALF	#	addf	R4	R5	R6			Oui	Oui
5	Div									

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1	UAL			UALF			Mult2								

Augmentation de performances

♦ Temps de cycle 2

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R1, (R2)**	#	#			> **Opérandes dispo**
MULTF	**R0, R1, R3**	#				> **Opérandes non disp**
ADDF	**R4, R5, R6**	#	#			> **Opérandes dispo**
MULTF	**R7, R5, R6**	#	#			> **Opérandes dispo**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R1	R2				Oui	
2	Mult1	#	multf	R0	R1	R3	UAL		Non	Oui
3	Mult2	#	multf	R7	R5	R6			Oui	Oui
4	UALF	#	addf	R4	R5	R6			Oui	Oui
5	Div									

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1	UAL			UALF			Mult2								

Augmentation de performances

♦ Temps de cycle 3

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R1, (R2)**	#	#	#	
MULTF	**R0, R1, R3**	#			
ADDF	**R4, R5, R6**	#	#	#	
MULTF	**R7, R5, R6**	#	#		

> **Exécution terminée**
> **Opérandes non dispo**
> **Exécution terminée**
> **Cycle 1 (3 cycles)**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R1	R2				Oui	
2	Mult1	#	multf	R0	R1	R3	UAL		Non	Oui
3	Mult2	#	multf	R7	R5	R6			Oui	Oui
4	UALF	#	addf	R4	R5	R6			Oui	Oui
5	Div									

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1	UAL			UALF			Mult2								

Augmentation de performances

♦ Temps de cycle 4

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R1, (R2)**	#	#	#	#
MULTF	**R0, R1, R3**	#			
ADDF	**R4, R5, R6**	#	#	#	#
MULTF	**R7, R5, R6**	#	#		

> **Ecriture résultat**
> **Opérandes non dispo**
> **Ecriture résultat**
> **Cycle 2 (3 cycles)**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R1	R2				Oui	
2	Mult1	#	multf	R0	R1	R3	UAL		***Oui***	Oui
3	Mult2	#	multf	R7	R5	R6			Oui	Oui
4	UALF	#	addf	R4	R5	R6			Oui	Oui
5	Div									

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1	***UAL***			***UALF***			Mult2								

Augmentation de performances

♦ Temps de cycle 5

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R1, (R2)**	#	#	#	#
MULTF	**R0, R1, R3**	#	#		
ADDF	**R4, R5, R6**	#	#	#	#
MULTF	**R7, R5, R6**	#	#	#	

> Instruction terminée
> Opérandes disponibles
> Instruction terminée
> Exécution terminée

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1	#	multf	R0	R1	R3			Oui	Oui
3	Mult2	#	multf	R7	R5	R6			Oui	Oui
4	UALF									
5	Div									

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1							Mult2								

Augmentation de performances

♦ Temps de cycle 6

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R1, (R2)**	#	#	#	#
MULTF	**R0, R1, R3**	#	#		
ADDF	**R4, R5, R6**	#	#	#	#
MULTF	**R7, R5, R6**	#	#	#	#

> Instruction terminée
> Cycle 1 (3 cycles)
> Instruction terminée
> Ecriture résultat

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1	#	multf	R0	R1	R3			Oui	Oui
3	Mult2	#	multf	R7	R5	R6			Oui	Oui
4	UALF									
5	Div									

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1							*Mult2*								

Augmentation de performances

- Temps de cycle 7

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R1, (R2)**	#	#	#	#	**> Instruction terminée**
MULTF	**R0, R1, R3**	#	#			**> Cycle 2 (3 cycles)**
ADDF	**R4, R5, R6**	#	#	#	#	**> Instruction terminée**
MULTF	**R7, R5, R6**	#	#	#	#	**> Instruction terminée**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2	#	multf	R7	R5	R6			Oui	Oui
4	UALF									
5	Div									

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1															

Augmentation de performances

♦ Temps de cycle 8

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R1, (R2)**	#	#	#	#	> **Instruction terminée**
MULTF	**R0, R1, R3**	#	#	#		> **Exécution terminée**
ADDF	**R4, R5, R6**	#	#	#	#	> **Instruction terminée**
MULTF	**R7, R5, R6**	#	#	#	#	> **Instruction terminée**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2	#	multf	R7	R5	R6			Oui	Oui
4	UALF									
5	Div									

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1															

Augmentation de performances

♦ Temps de cycle 9

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R1, (R2)**	#	#	#	#	**> Instruction terminée**
MULTF	**R0, R1, R3**	#	#	#	#	**> Ecriture résultat**
ADDF	**R4, R5, R6**	#	#	#	#	**> Instruction terminée**
MULTF	**R7, R5, R6**	#	#	#	#	**> Instruction terminée**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL									
2	Mult1									
3	Mult2	#	multf	R7	R5	R6			Oui	Oui
4	UALF									
5	Div									

Etat des registres

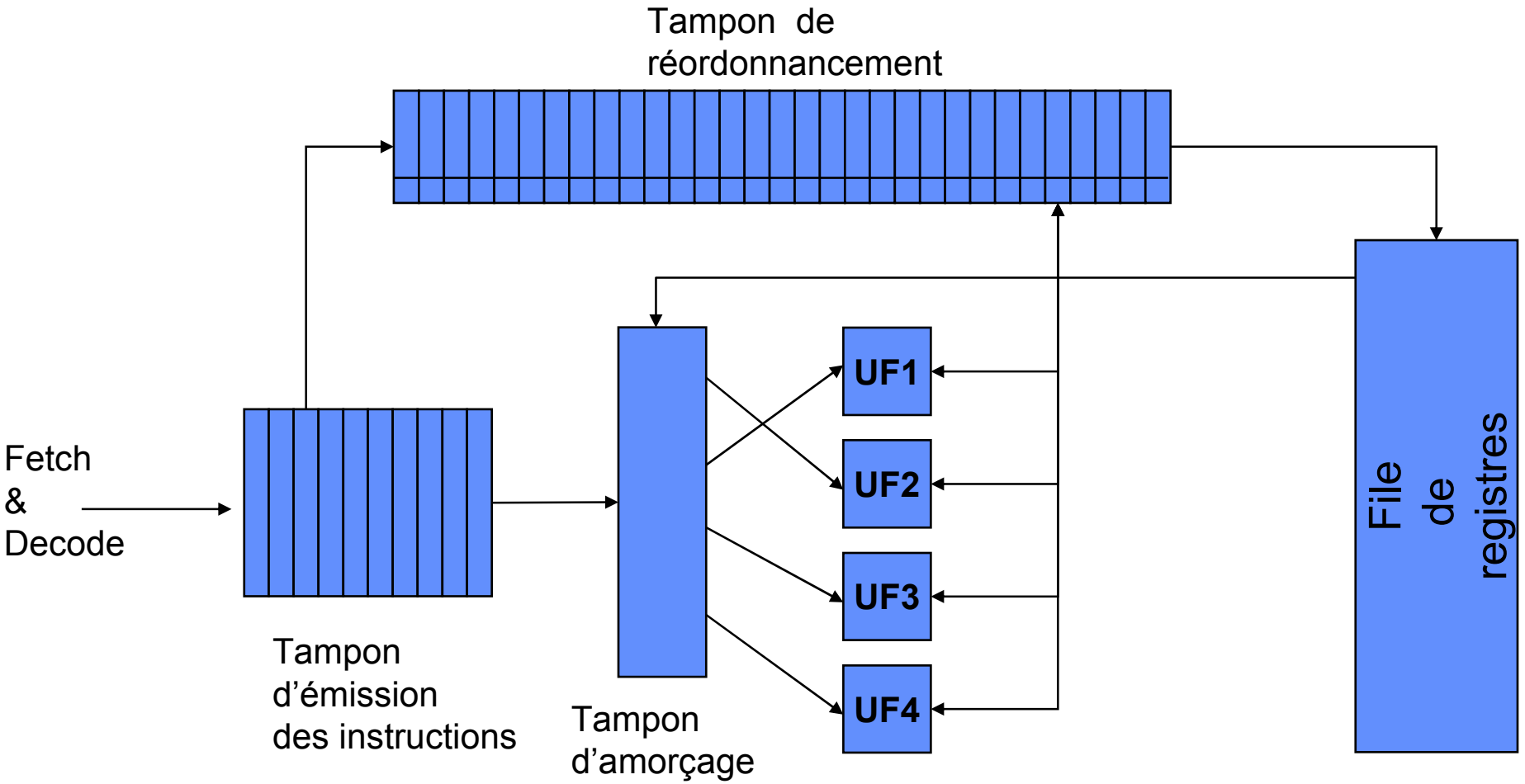
R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1															

Augmentation de performances

- Stations de réservation :
  - les instructions attendent leurs opérandes dans les stations de réservation
  - les stations peuvent être :
    - globales aux unités fonctionnelles
    - locales aux unités fonctionnelles

Augmentation de performances

❑ Renommage de registres et ordonnancement dynamique : ROB



Augmentation de performances

Le Rob : suite

- les instructions sont placées dans le ROB en respectant l'ordre
- on renomme les registres et on mémorise l'association entre registres physique et registres logiques
- une place est prévue pour la valeur résultante
- l'instruction est marquée prête dans le tampon d'amorçage si les opérandes sont prêts ou reste en attente d'opérandes
- l'écriture dans la file de registres n'est faite que lorsque l'instruction est en tête de ROB
- gestion exception : une exception générée est stockée dans le ROB et ne sera traitée que lorsque l'instruction sera en tête de ROB, au moment où l'on sait qu'elle ne peut plus être annulée

Augmentation de performances

❑ Modèle SMT ou Hyperthreading

➢ Simultaneous MultiThreading

➢ hyperthreading (terme utilisé par Intel)

➢« Exécution simultanée de plusieurs processus léger »

➢ une des innovations architecturale les plus intéressante de ces dernières années :

- ♦ gain en performances important par rapport au surcoût induit :
  - – 5% de complexité en plus
  - – 20 à 30% de performances supplémentaires

Augmentation de performances

➢Motivations :

- un processeur superscalaire de degré N, peut exécuter, en théorie, jusqu ’à N instructions par cycle, or on observe que ce type de machine n ’exécute guère plus de 2 ou 3 instructions par cycle (parallélisme limité) :
  - granularité de parallélisme fine

- une machine multiprocesseur exécute plusieurs processus sur différents processeurs :
  - granularité de parallélisme importante

Augmentation de performances

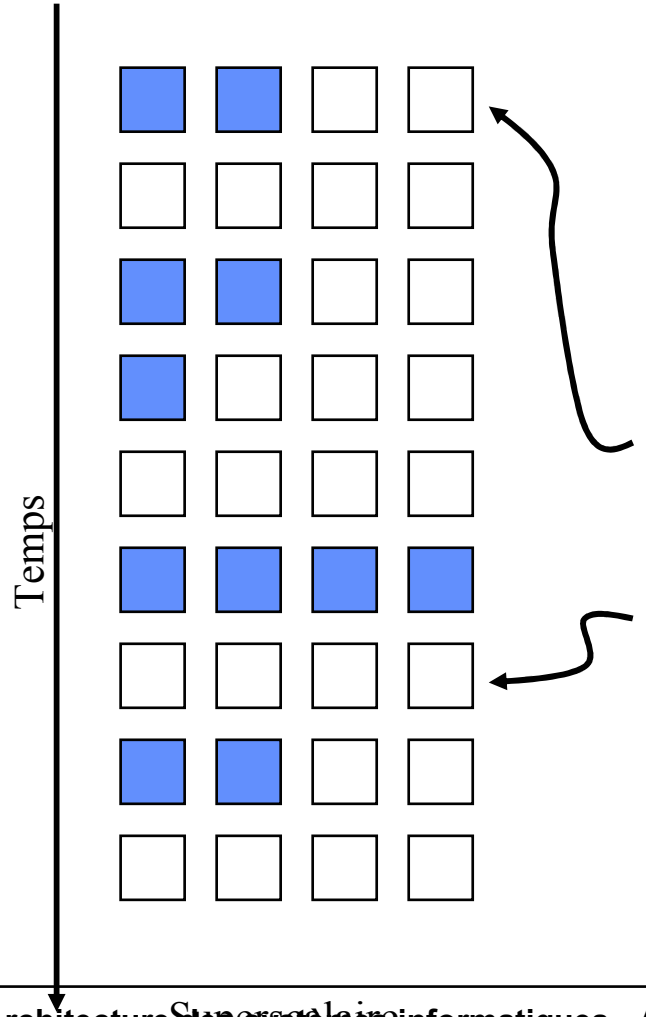
➢Objectif des SMT :

- ♦ exploiter tous les parallélismes disponibles dans les applications :
  - – grains fin ou gros grains
- ♦ utilisation efficace des ressources de la machine

Augmentation de performances

❑ Le SMT : comment ça marche ?

➢Rappel : fonctionnement d'une machine superscalaire



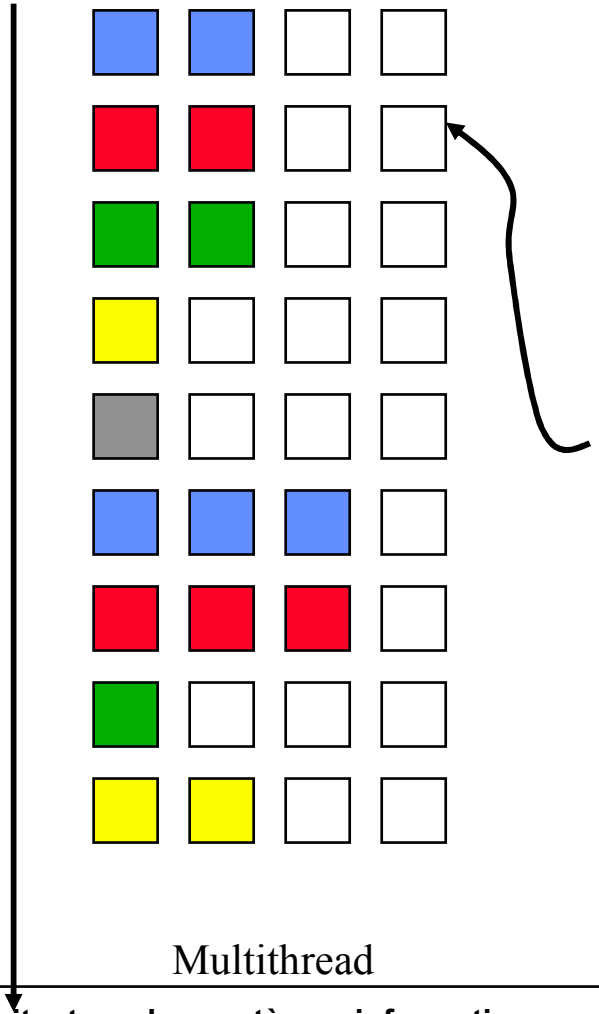
A chaque temps de cycle, le processeur tente de lancer un maximum d'instructions d'un processus.

Le faible parallélisme des applications provoque des trous dans les lignes horizontales.

De plus, certaines instructions demandent plus d'un cycle pour être exécutées, cela provoque des trous dans les lignes verticales

Augmentation de performances

➢Rappel : fonctionnement d'une machine multithread



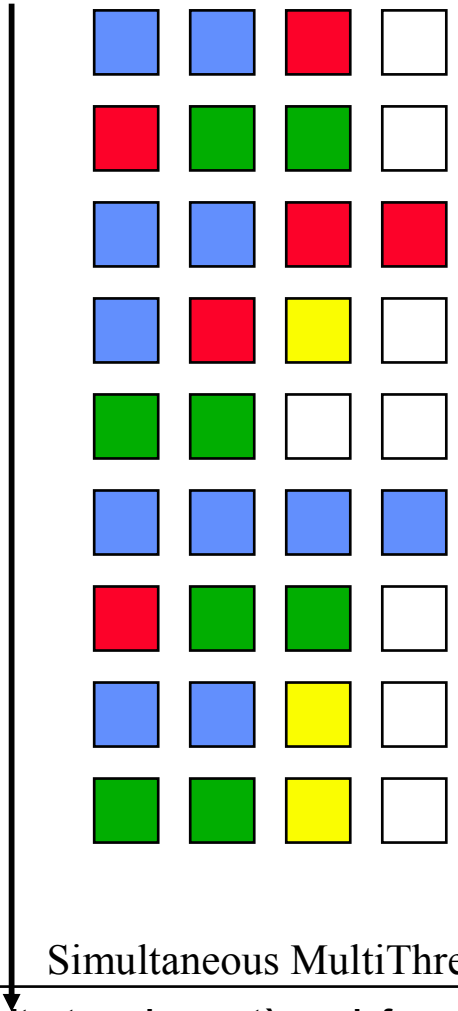
A chaque temps de cycle, le processeur tente de lancer un maximum d'instructions d'un processus.
Au temps de cycle suivant, il fait de même avec un autre processus.

Le faible parallélisme intra-processus provoque des trous dans les lignes horizontales.

Les trous dans les lignes verticales sont supprimés.

Augmentation de performances

➢Fonctionnement d'une machine multithread



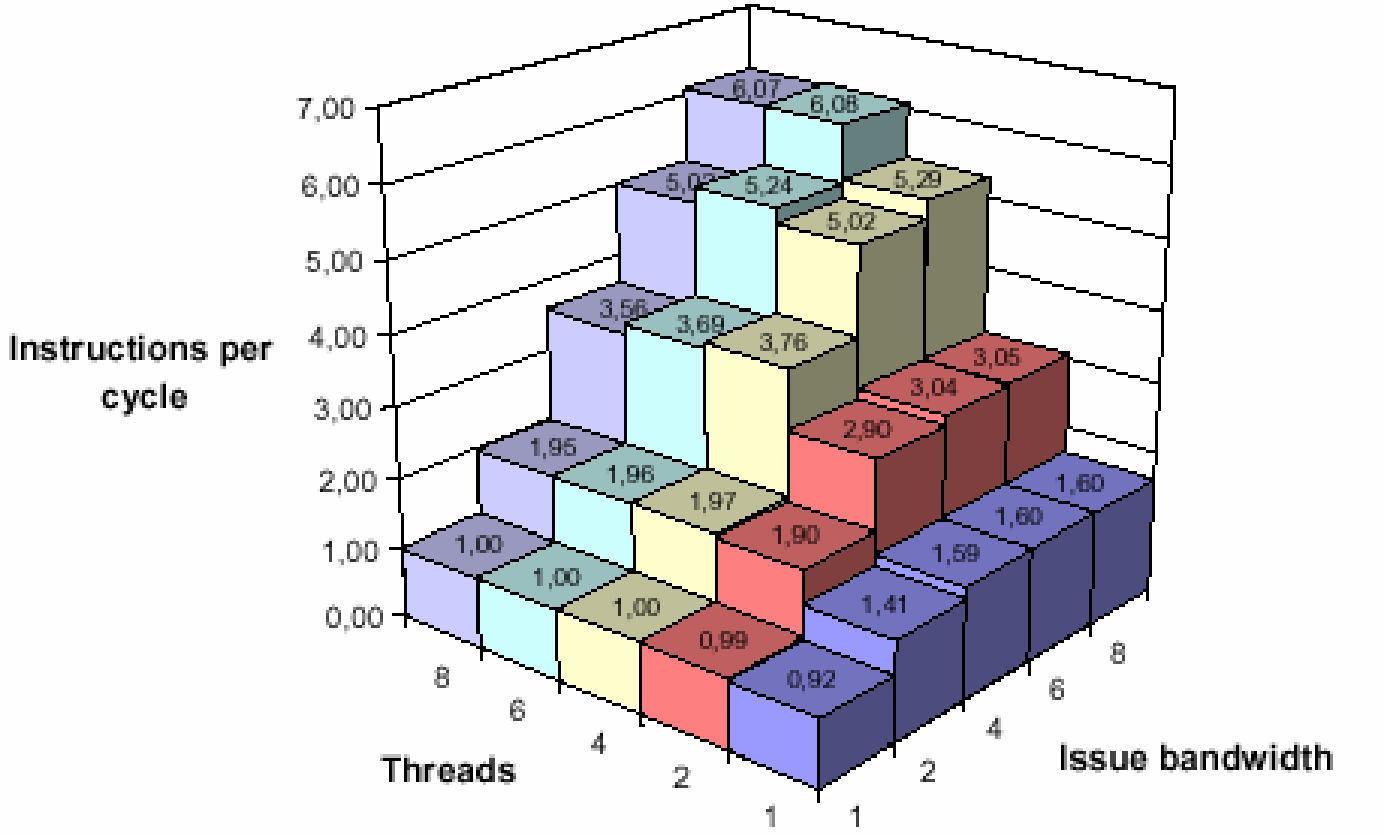
A chaque temps de cycle, le processeur tente d'occuper un maximum de ses ressources.
Il lance un maximum d'instructions parmi toutes les instructions de tous les processus de l'application

Les tous dans les lignes horizontales sont moins nombreux.

Les trous dans les lignes verticales sont supprimés.

Augmentation de performances

- **Performances :**
  - ➢ le modèle SMT offre de bonne performances pour les applications qui ne peuvent être parallélisée
  - ➢ meilleure utilisation des ressources du processeur



Augmentation de performances

- **Processeurs implémentant ce concept:**
  - Power PC G3
  - l'ex futur processeur Alpha 21464, prévu pour 2003
  - le Pentium 4 :
    - environ 5% de surface supplémentaire pour une augmentation de performance de 27 %)

- **Coût matériel :**
  - nécessite la mise en place de plusieurs contexte physique

- **Processeurs classiques :**
  - 1 contexte physique
  - plusieurs contextes d'exécution

- **Processeurs SMT :**
  - plusieurs contextes physiques
  - plusieurs contextes d'exécution

Augmentation de performances

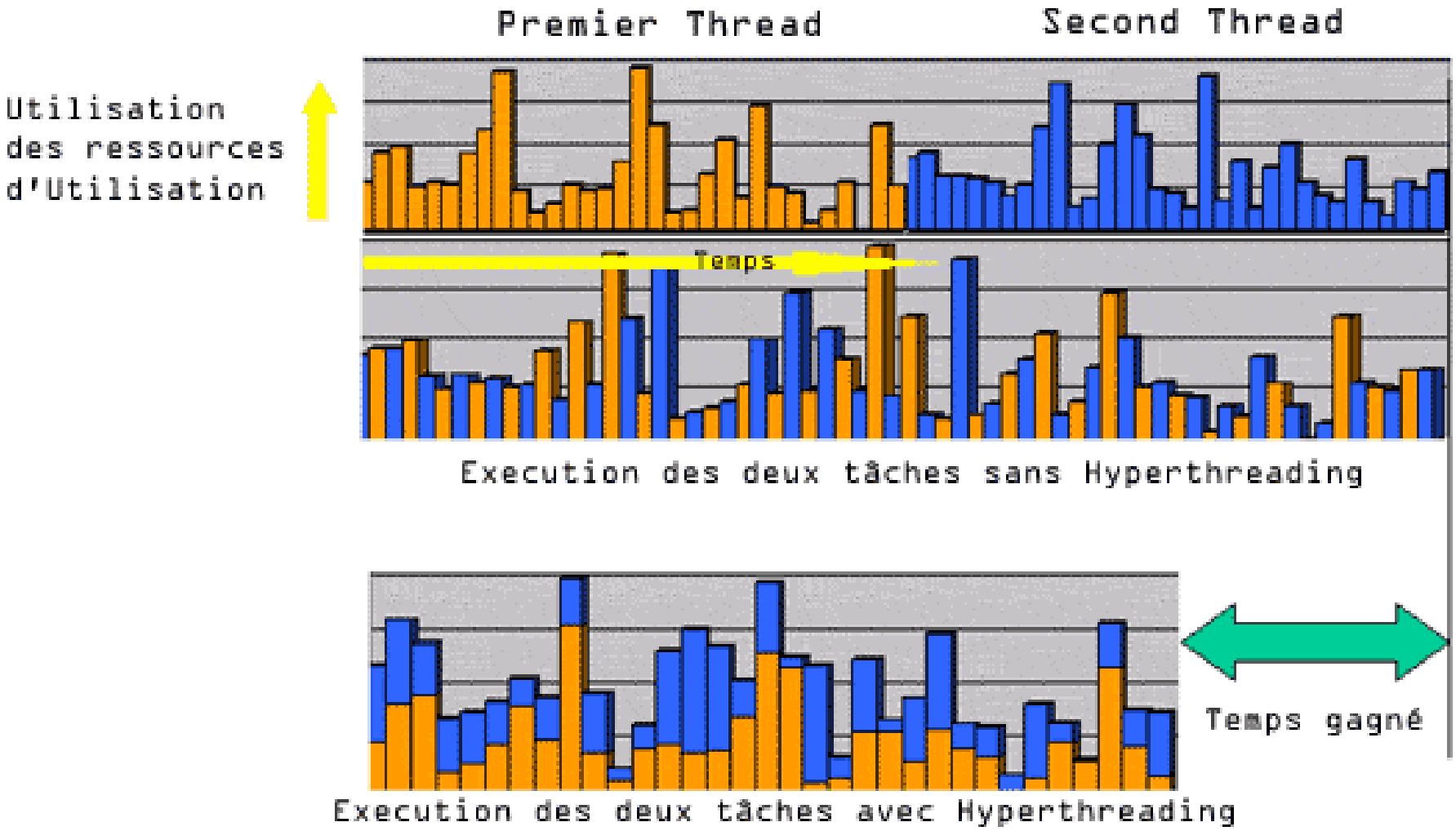
L'hyperthreading du Pentium 4

- à un cycle donné le processeur exécute 1 thread
- un seul thread ne parvient pas à occuper les 20 étages pipelines d'un Pentium 4

- objectif : permettre l'exécution de plusieurs threads
  - mise en place de plusieurs contextes physiques d'exécution

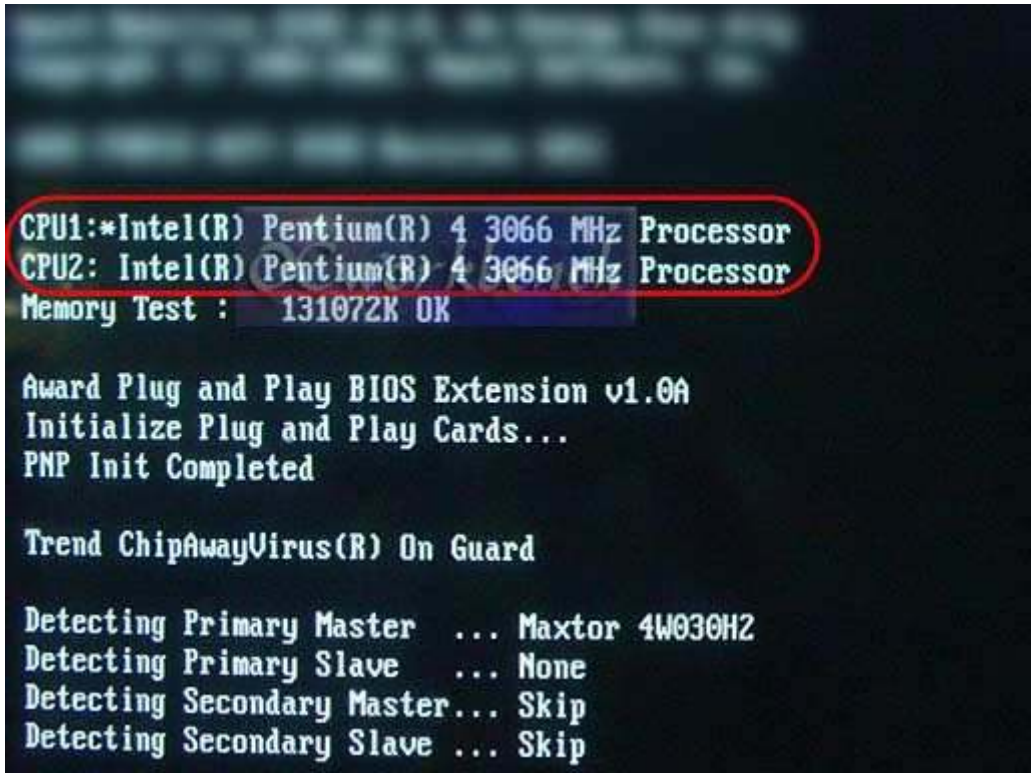
Augmentation de performances

- **L'hyper threading suite :**



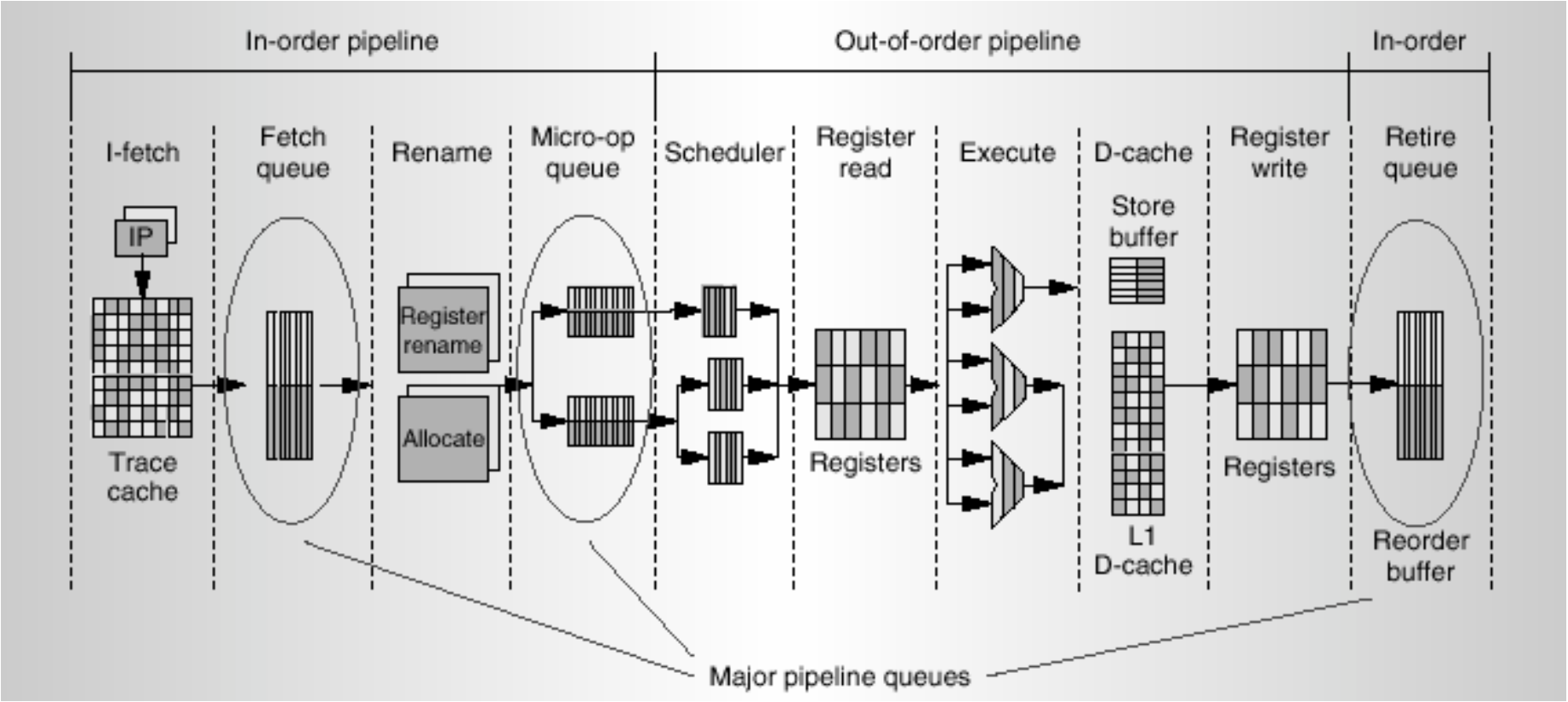
Augmentation de performances

- **L'hyper threading suite :**
  - ➢ le système d'exploitation voit 2 processeurs



Augmentation de performances

➢ les principales parties du pipeline du Pentium 4



Augmentation de performances

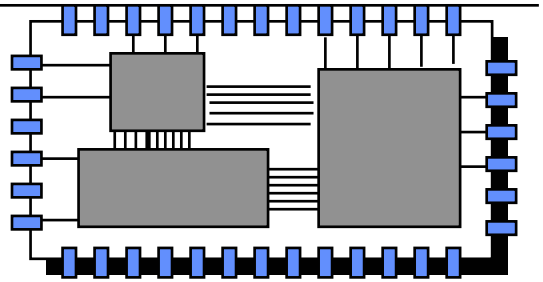
❑ Le CMT : Chip MultiThreading

➢ plusieurs cœurs dans le même chip

- ♦ Niagara :
  - 8 cœurs,
  - chaque cœur est un SMT pouvant traiter 4 threads
  - potentiellement : 32 threads pris en charge
  - prévu pour début 2006
  - 90 nm
- ♦ Rock :
  - prévu pour 2008
  - gravé en 65 nm

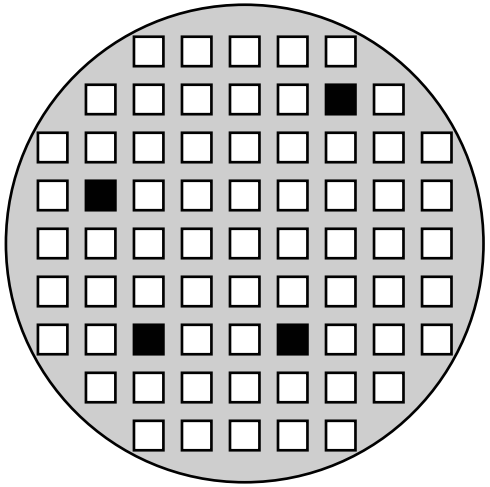
Augmentation de performances

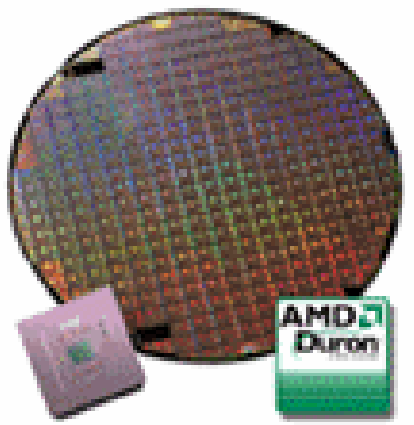
La technologie MCM



- Objectifs :
  - placer plus de chose dans le même package
  - réaliser des puces de tailles raisonnables, pour en limiter le coût
- Notion de coût d'un circuit intégré :

$$\text{Coût du circuit intégré} = \frac{\text{Coût de la puce} + \text{Coût du test} + \text{Coût de mise en boîtier}}{\text{Rendement après test final}}$$





$$\text{Coût d'une puce} = \frac{\text{Coût de la tranche}}{\text{Nb puces par tranche} * \text{Rendement des puces}}$$

Augmentation de performances

➢ estimation du nombre de puces par tranches :

$$\text{Nb puces par tranche} = \frac{\text{PI} * (\text{Diamètre tranches} / 2)^2}{\text{Surface puce}} - \frac{\text{PI} * \text{Diamètre tranches}}{\sqrt{2 * \text{Surface puce}}}$$

Compense le problème de la puce carrée dans une tranche ronde

➢ notion de rendement :

$$\text{Rendement par puce} = \left(1 * \frac{\text{Défaut par unité de surface} * \text{Surface puce}}{\text{alpha}}\right)^{-\text{alpha}}$$

Technologie de fabrication

➢ nb puces bonnes par tranches :

Nb Puces correctes = Nb Puces par tranches * Rendement par puce

Augmentation de performances

♦ valeurs typiques des différents termes :

– alpha = correspond en gros au nombre de niveaux de masques pour réaliser la puce, plus il y a de masques plus alpha est grand :
  – CMOS à plusieurs niveaux, alpha = 3 est une bonne estimation
– les défaut par unité de surface sont aléatoirement répartis, en 1995 les valeurs typiques étaient comprises entre 0,6 et 1,2 par cm^2

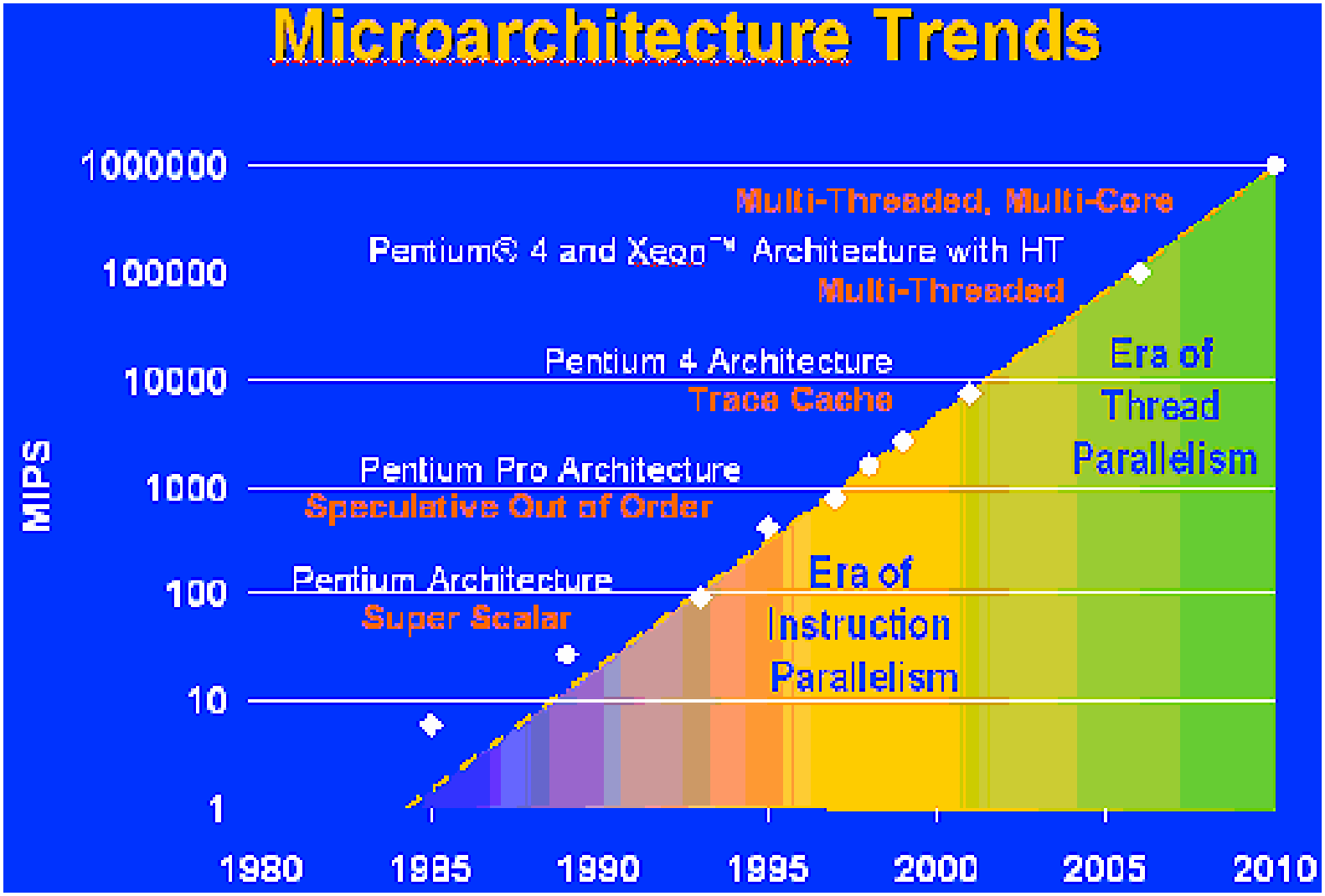
–Exemple :

•densité de défaut = 0,8 par cm^2

•puces de 1 cm de côté

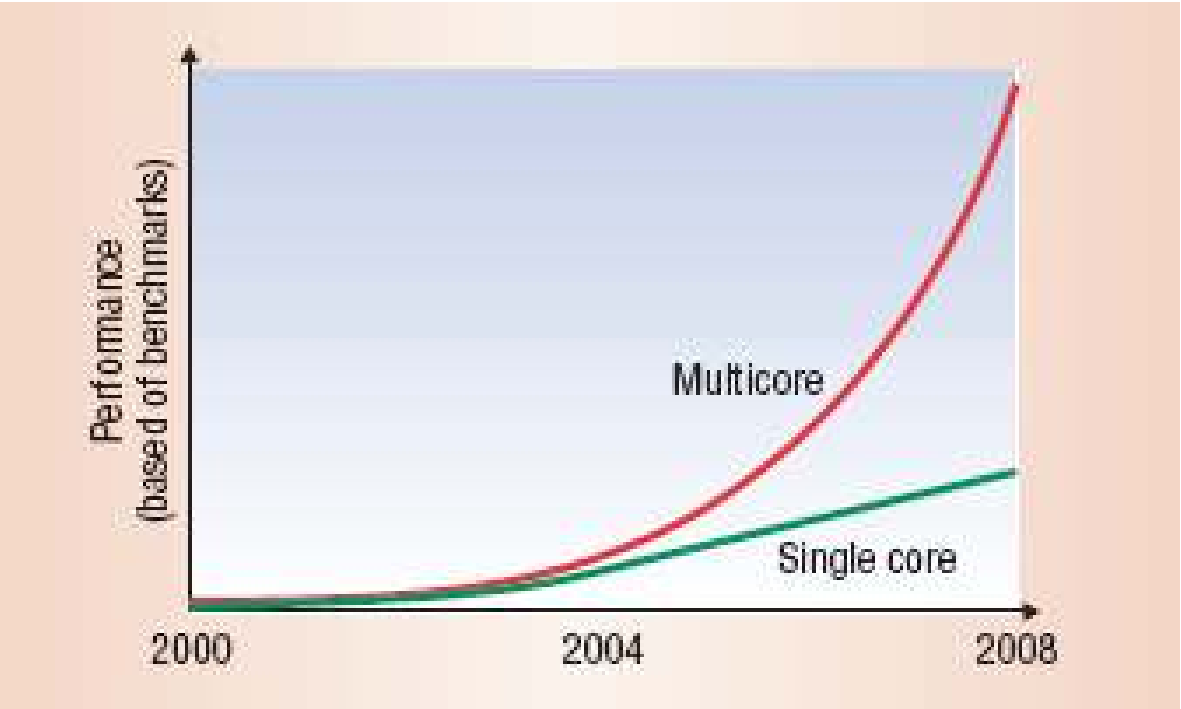
•waffer de 20 cm de diamètre

Augmentation de performances



Multi-core Architecture

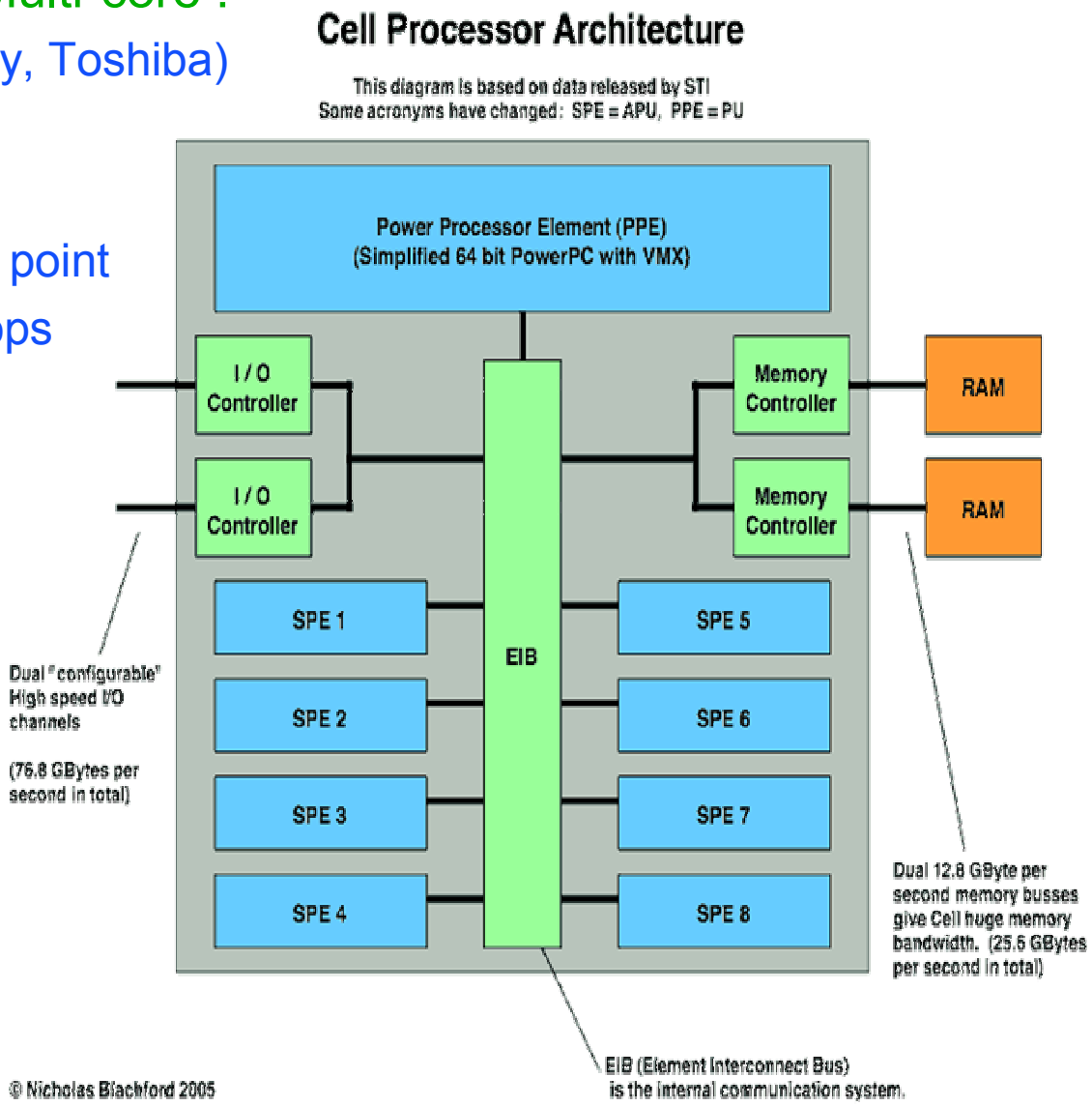
- Objectifs :
  - placer plusieurs cœurs de processeur dans le même package



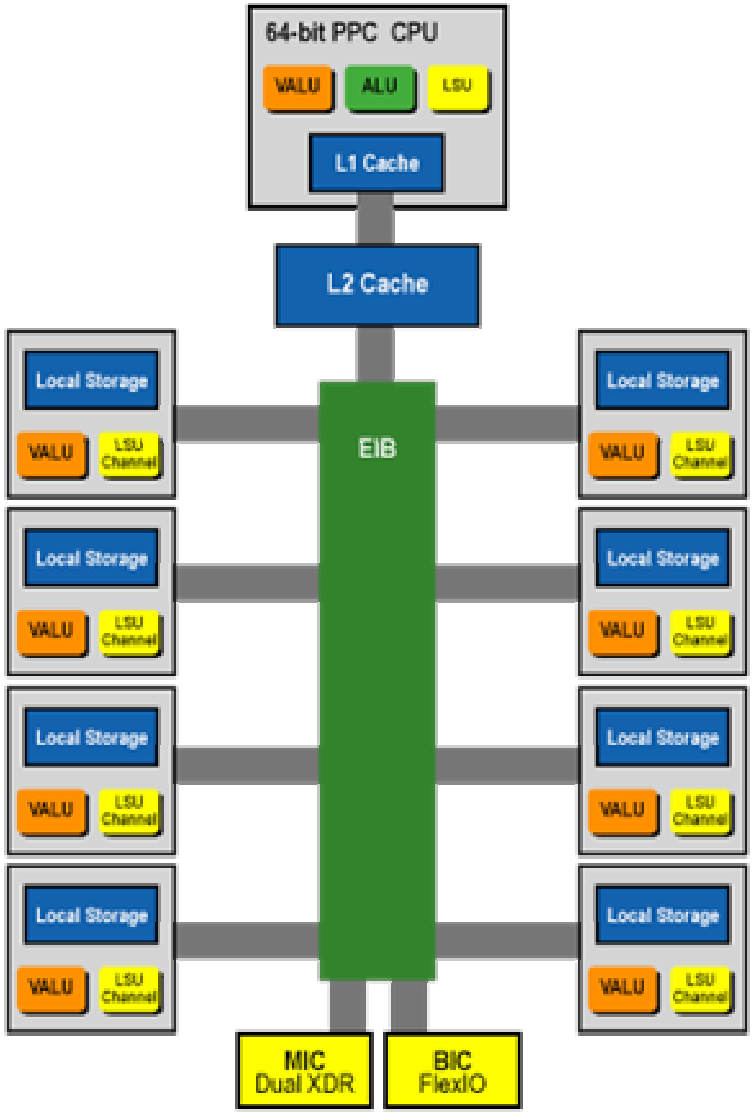
Augmentation de performances

➢ Exemple d'architecture Multi-core :

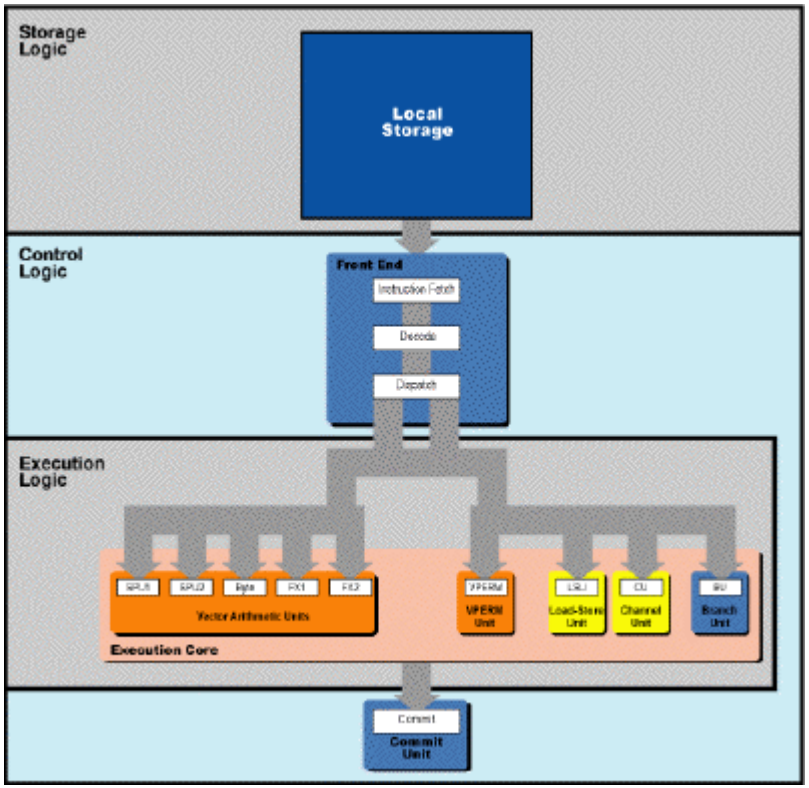
- ♦ Machine Cell (Ibm, Sony, Toshiba)
- ♦ Assimilable à un SoC
- ♦ Power core 64 bits
- ♦ 8 core 128 bits, floating point
- ♦ Perf : 4,6 Ghz, 256 Gflops



Augmentation de performances

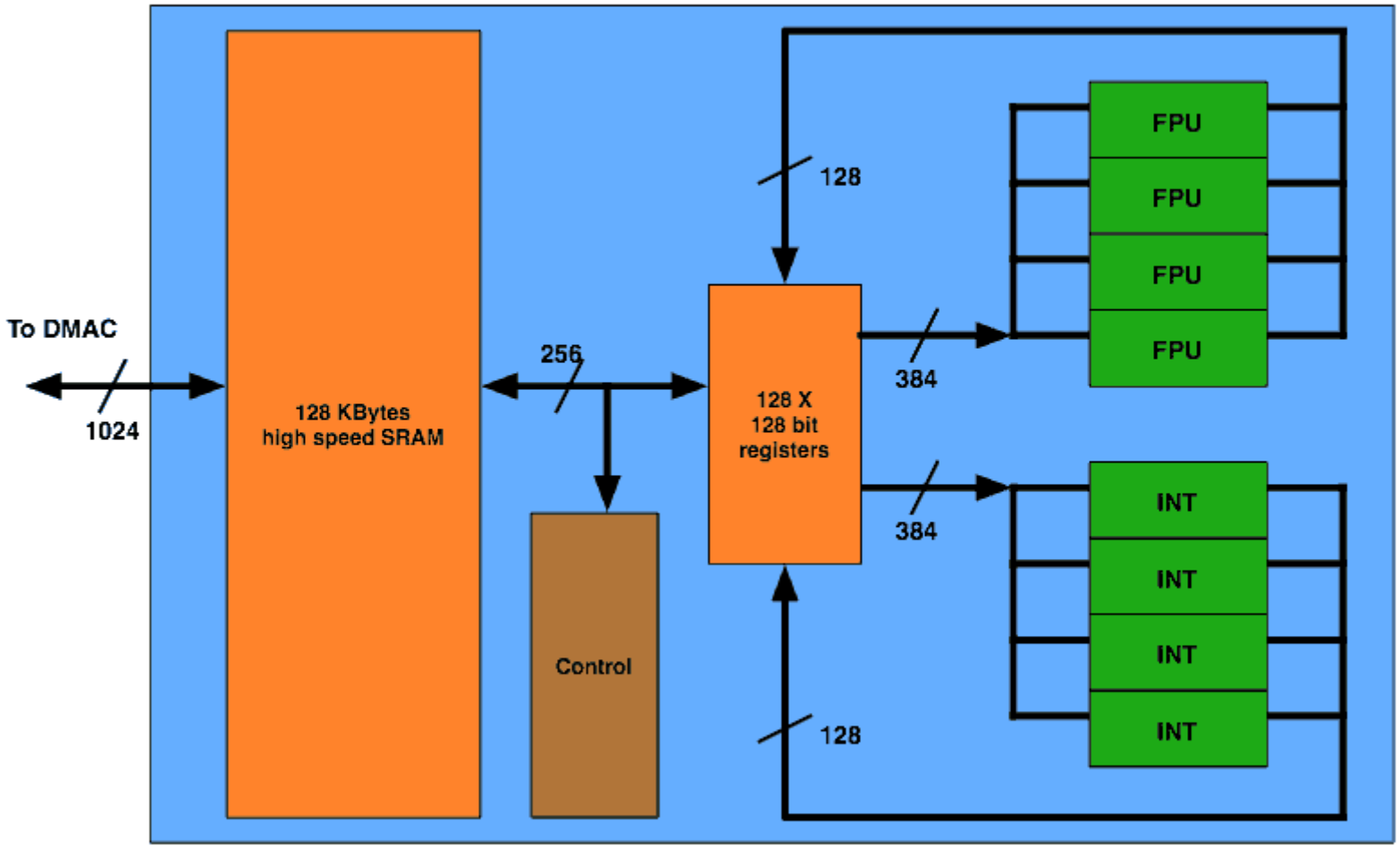


The CELL Architecture

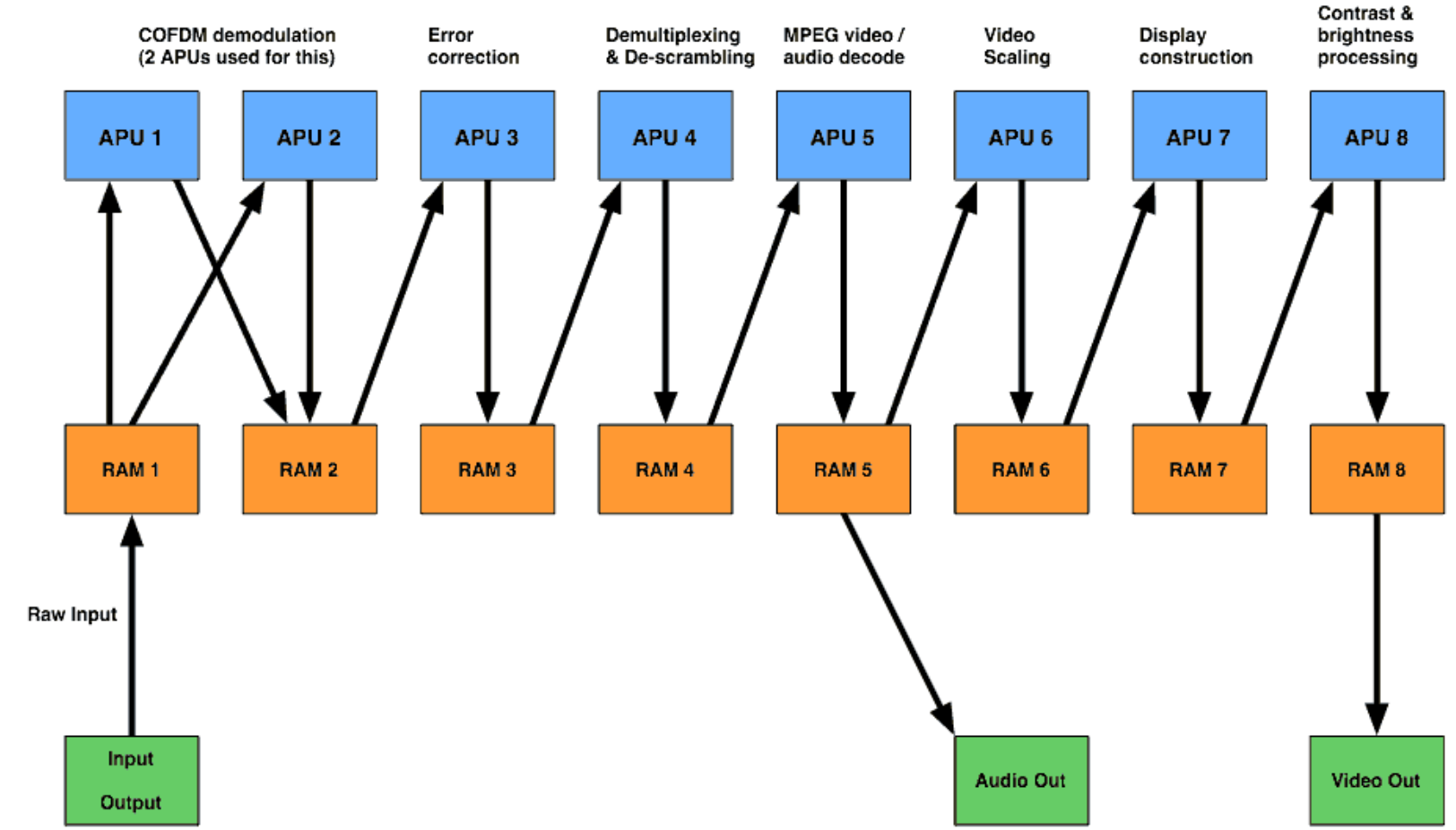

Storage Logic
Local Storage
Control Logic
Front End
Instruction Fetch
Decode
Dispatch
Execution Logic
Vector Arithmetic Units
Execution Core
Commit
Commit Unit

Cell APU Architecture

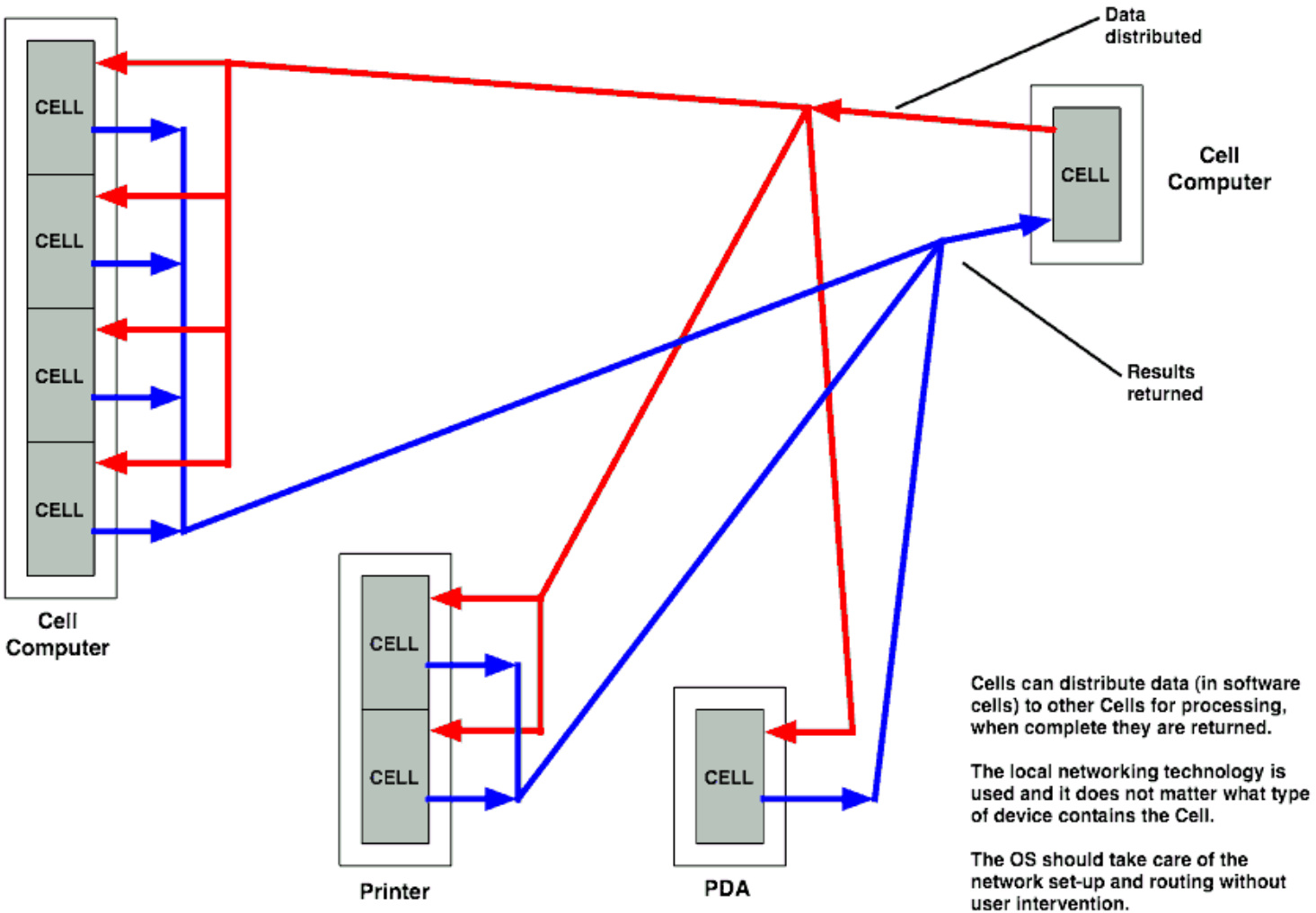
Each APU is an independent vector CPU capable of 32 GFLOPs or 32 GOPs.



Stream Processing



Distributed Processing with Cells



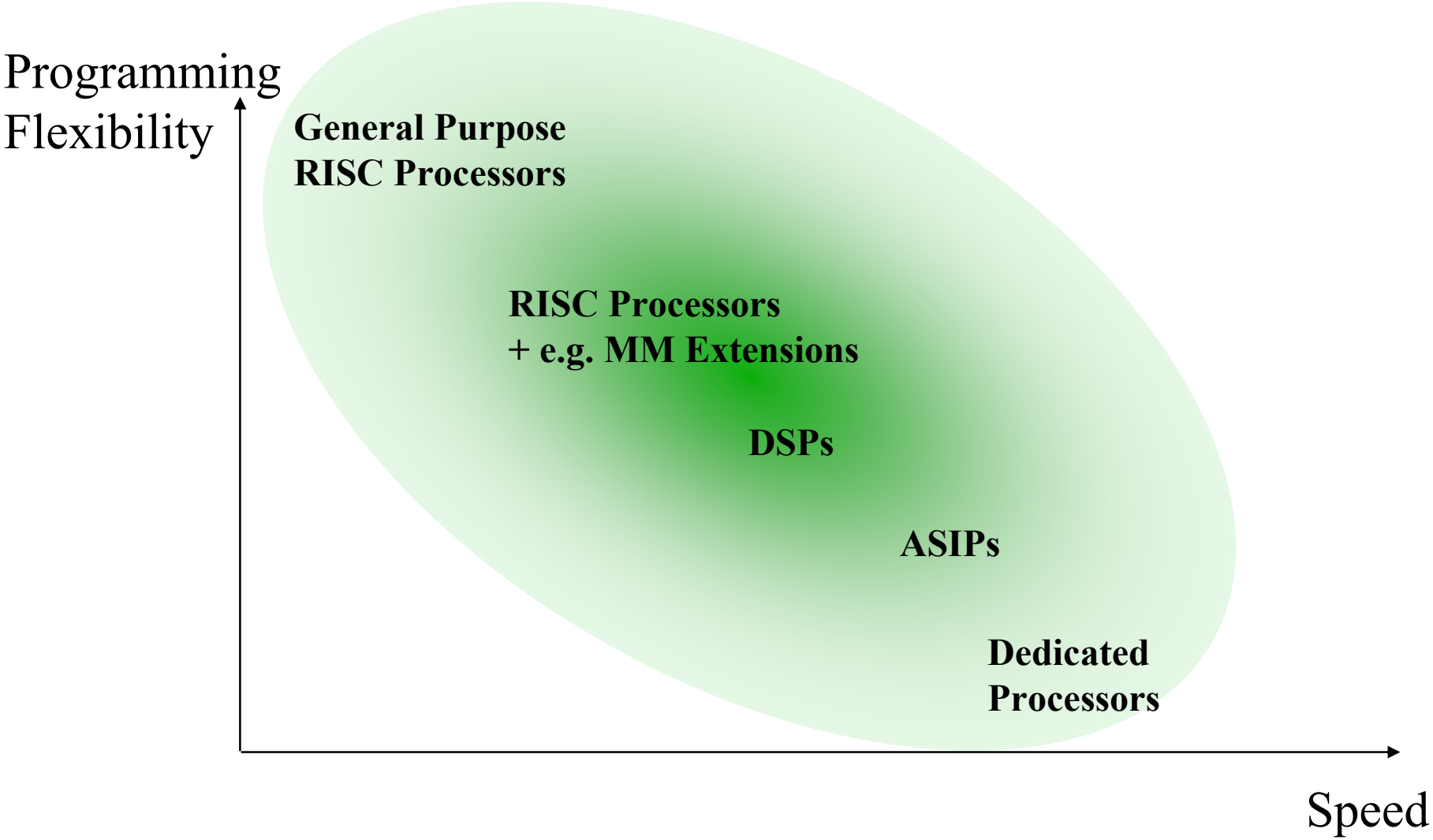
Augmentation de performances

Résumé

TECHNIQUES / MODELE D'EXECUTION	VLIW	Superscalaire	EPIC
Allongement des pipelines	+	+++	++
Plusieurs unités fonctionnelles en //	Oui	Oui	Oui
By pass	Oui	Oui	Oui
Buffer de préchargement d'instructions	?	+++	++
Module de dispatching	Oui/non	oui	Oui
Prédiction de branchement	Branchement retardé	Oui	Oui
Cache de trace	Non	Oui	?
Renommage de registres	?	Oui	Oui
Exécution dans le désordre	Non	Oui	Oui
Multi thread simulanné	Non ?	Oui	Oui ?
Move conditionnel	?	Oui	Oui
Compilation	Délicate	Pas d'extraction de //	Délicate
Exécution	Facile	Délicate	Relativement facile

PANORAMA

Panorama de processeurs



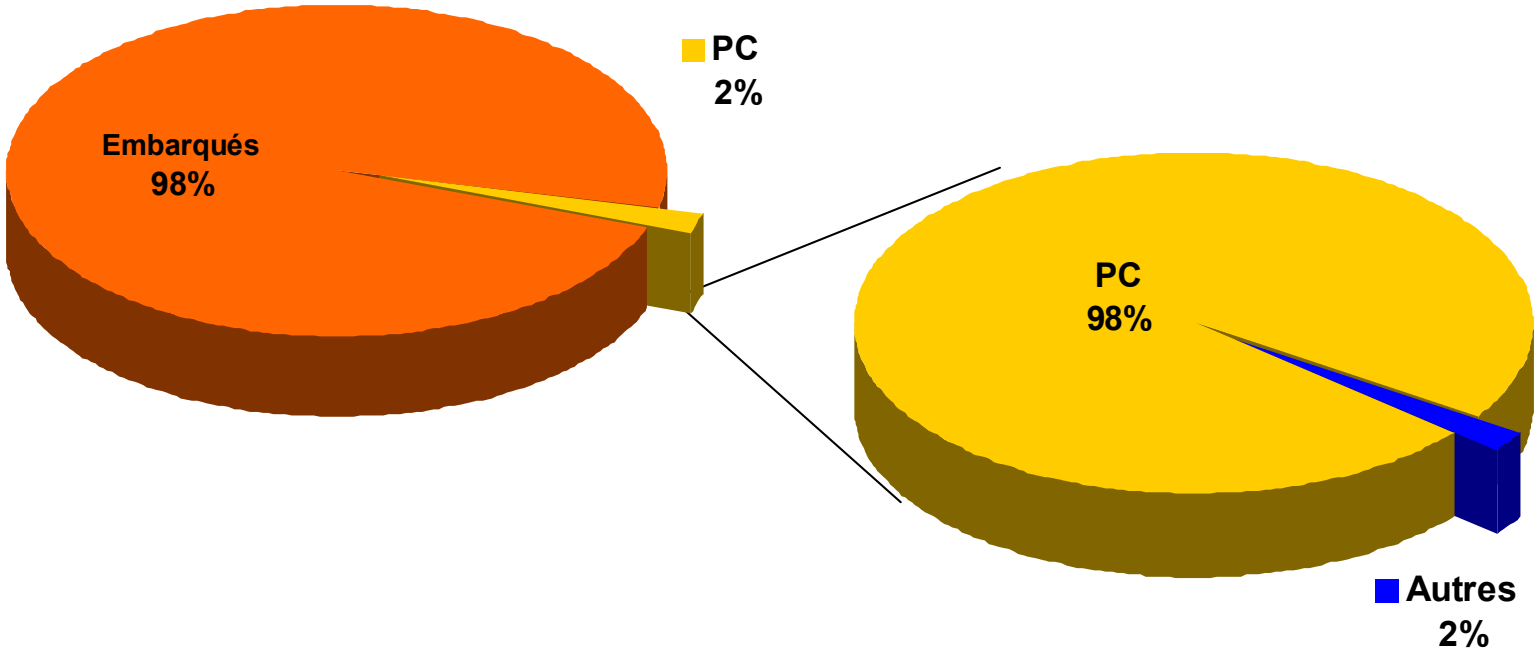
Panorama de processeurs

❑ Volume des ventes :

➢50 millions de microprocesseurs pour PC

➢4 milliards de microprocesseurs

➡ Le marché économique est dans les µP *embarqués*



Panorama de processeurs

Les processeurs généraux

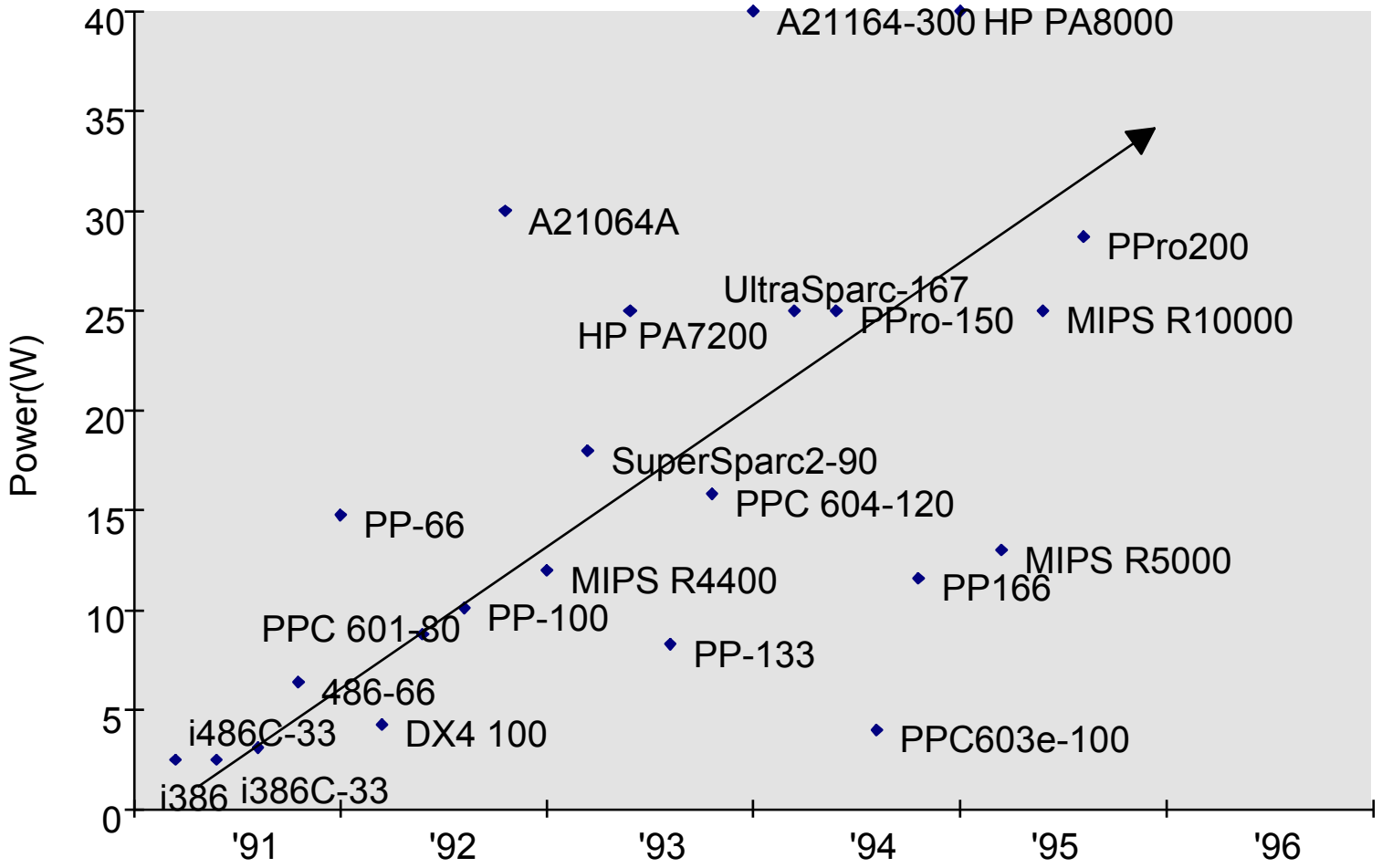
- Voir tableaux d'informations générales sur les différents constructeurs :
  - **http://infopad.EECS.Berkeley.EDU/CIC/summary/local/**
- Remarques générales :
  - augmentation de la consommation des processeurs :
    - c'est une conséquence de l'augmentation
      - de la fréquence d'horloge
      - du nombre de transistors
      - de la taille totale du circuit
    - des études de refroidissement ont été menées :
      - nouvelle technique de refroidissement (caloduque)
      - abaissement de température (jusqu'à - 40 °C)
  - les tensions d'alimentation diminuent :
    - pour limiter l'augmentation de consommation : (Pentium Pro = 30 Watts ====> 8 Watts)

$$P = \alpha \cdot C \cdot Vdd^2 \cdot f$$

Panorama de processeurs

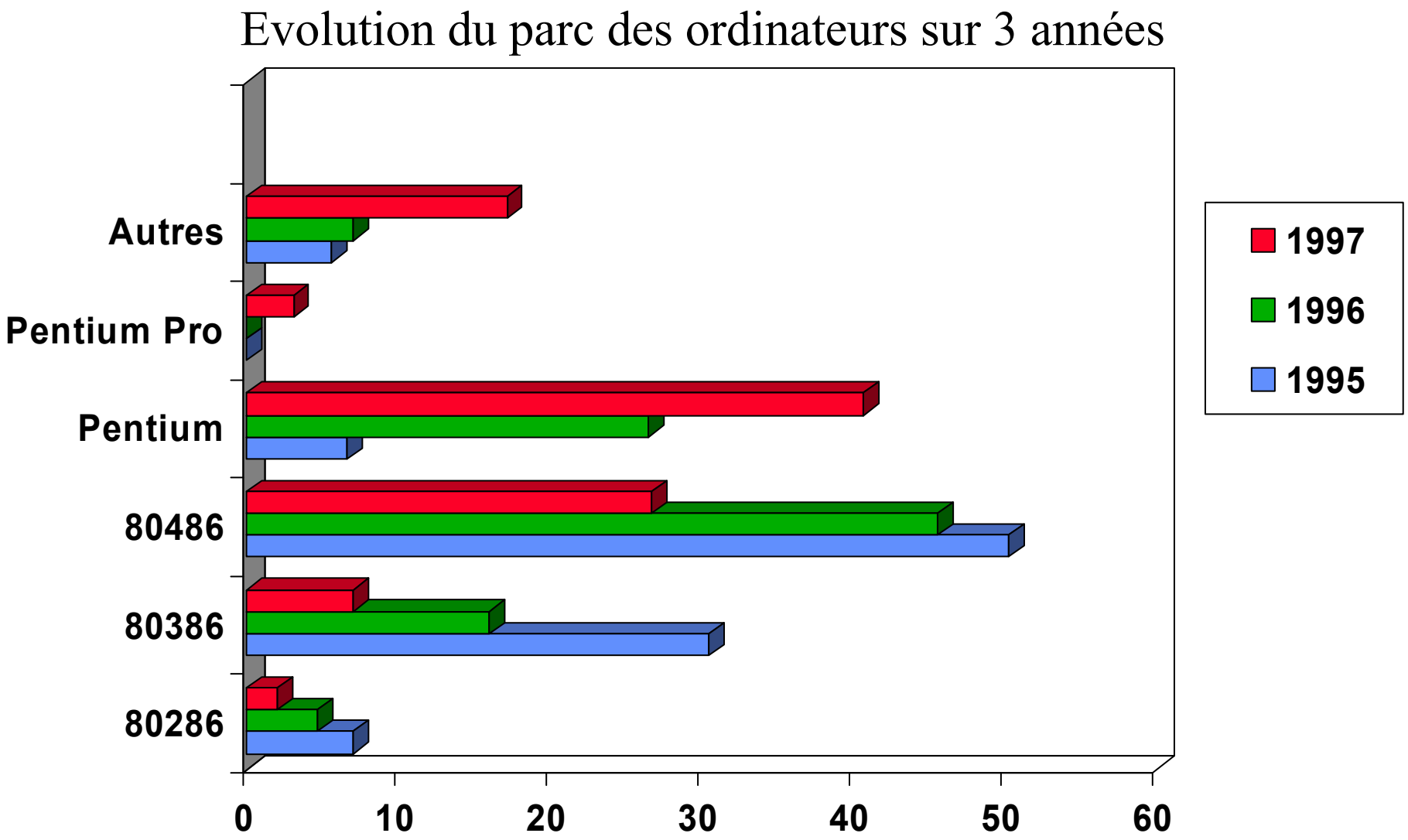
♦ Evolution de la consommation des microprocesseurs

– x4 tous les 3 ans



[Source: Microprocessor Report]

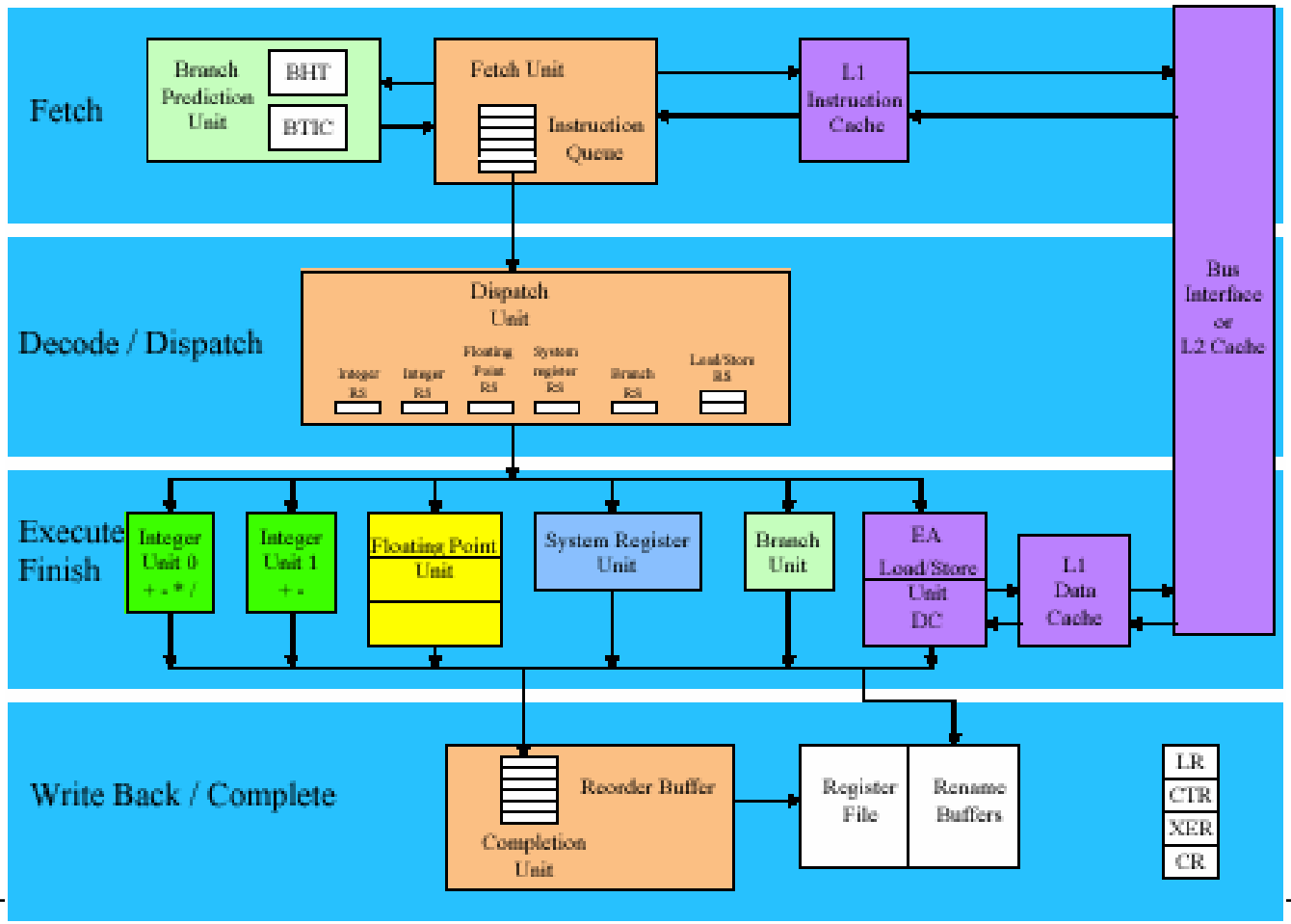
Panorama de processeurs



Panorama de processeurs

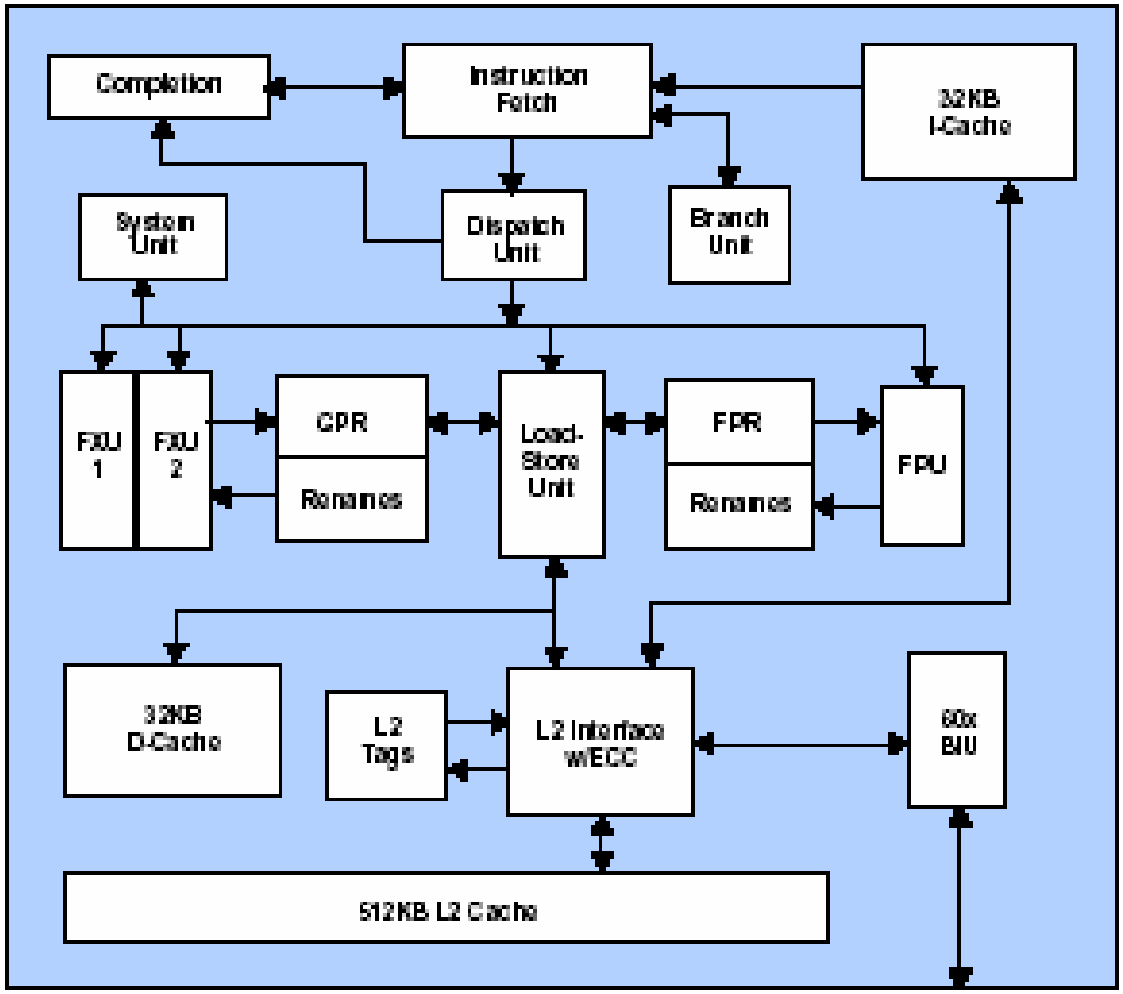
❑ Exemple de processeur à architecture power pc

➢le power pc 750



Panorama de processeurs

❑ Le power PC 750 FX



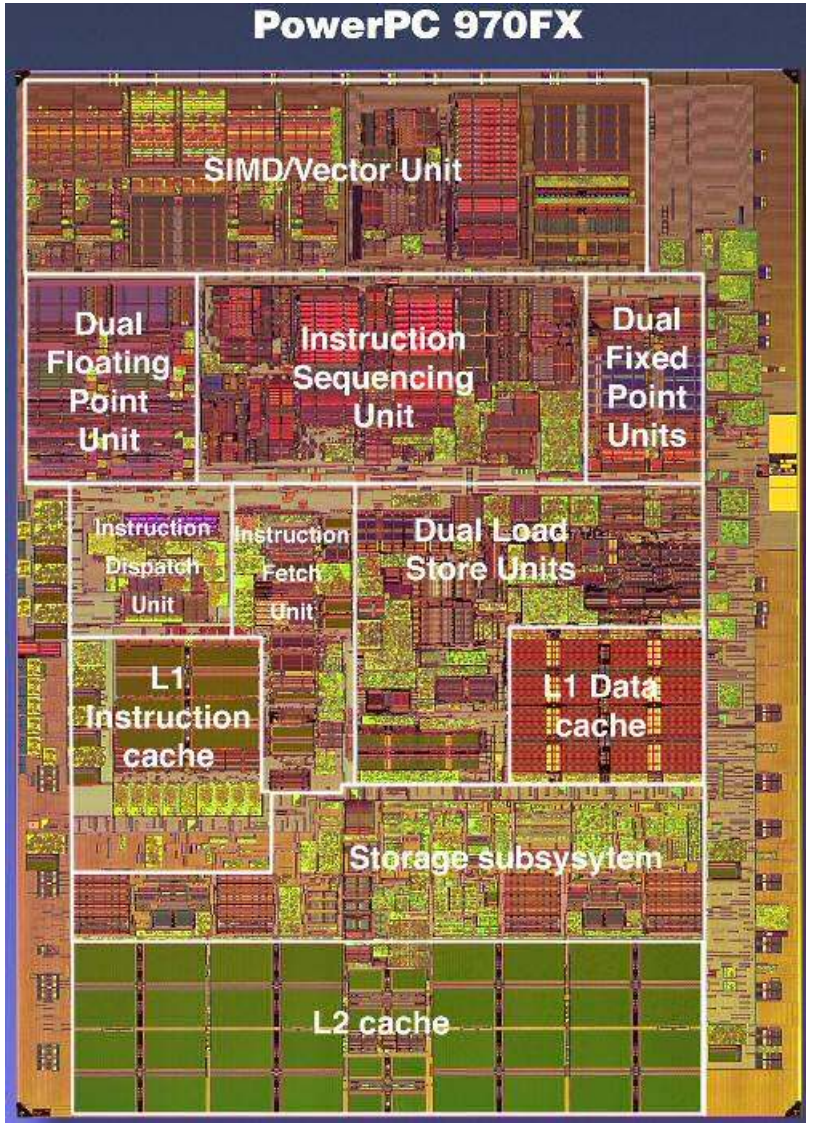
Panorama de processeurs

- Power PC 750 FX
  - fréquence : 600 à 1000 Mhz
  - cache L1 de 2 fois 32 Ko
  - cache L2 de 512 Ko interne
  - technologie : 0,13 micron
  - mémoire adressable : 64 Go (36 bits d'adresses)
  - tensions d'alimentation :
    - 1,4 V pour le cœur
    - 3,3 V pour les I/O
  - dissipation : 3,6 W (800 Mhz - 1,4 V)

Panorama de processeurs

➢PowerPC 970FX

- ♦ Superscalaire : 5 instructions par cycle
- ♦ Out-of-order execution
- ♦ Chargement de 8 instructions par cycle dans le cache L1
- ♦ Décodage de 8 instructions par cycle



Panorama de processeurs

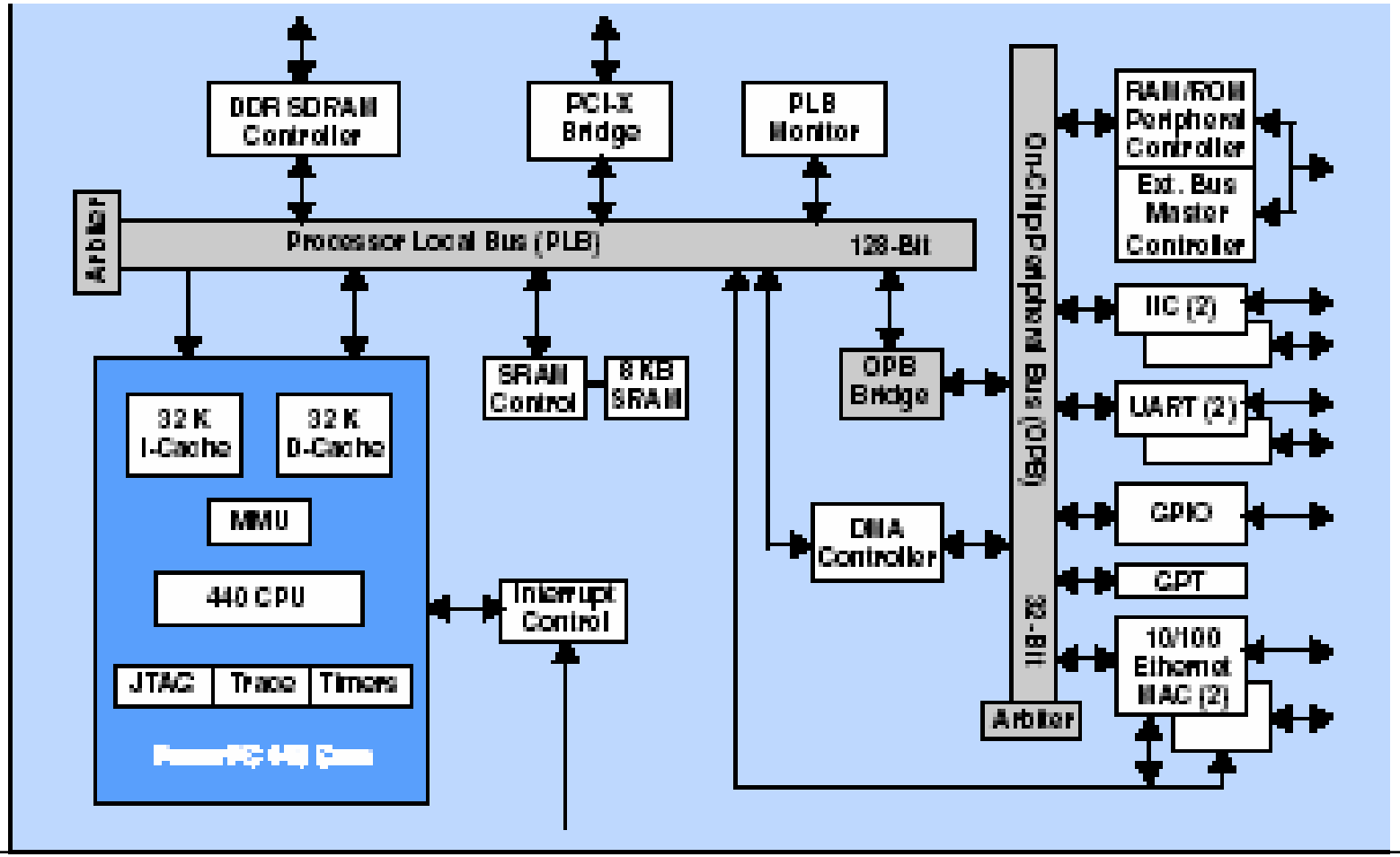
❑ Processeur Gekko : GameCube

➢Gekko Chip:

- IBM PowerPC Architecture
- Customized 405 MHz processor
- 0.18 micron copper wire technology
- 32-bit integer
- 64-bit floating point performance

Panorama de processeurs

- **System on chip processeur :**
  - ➢Le power pc 440 GP embedded

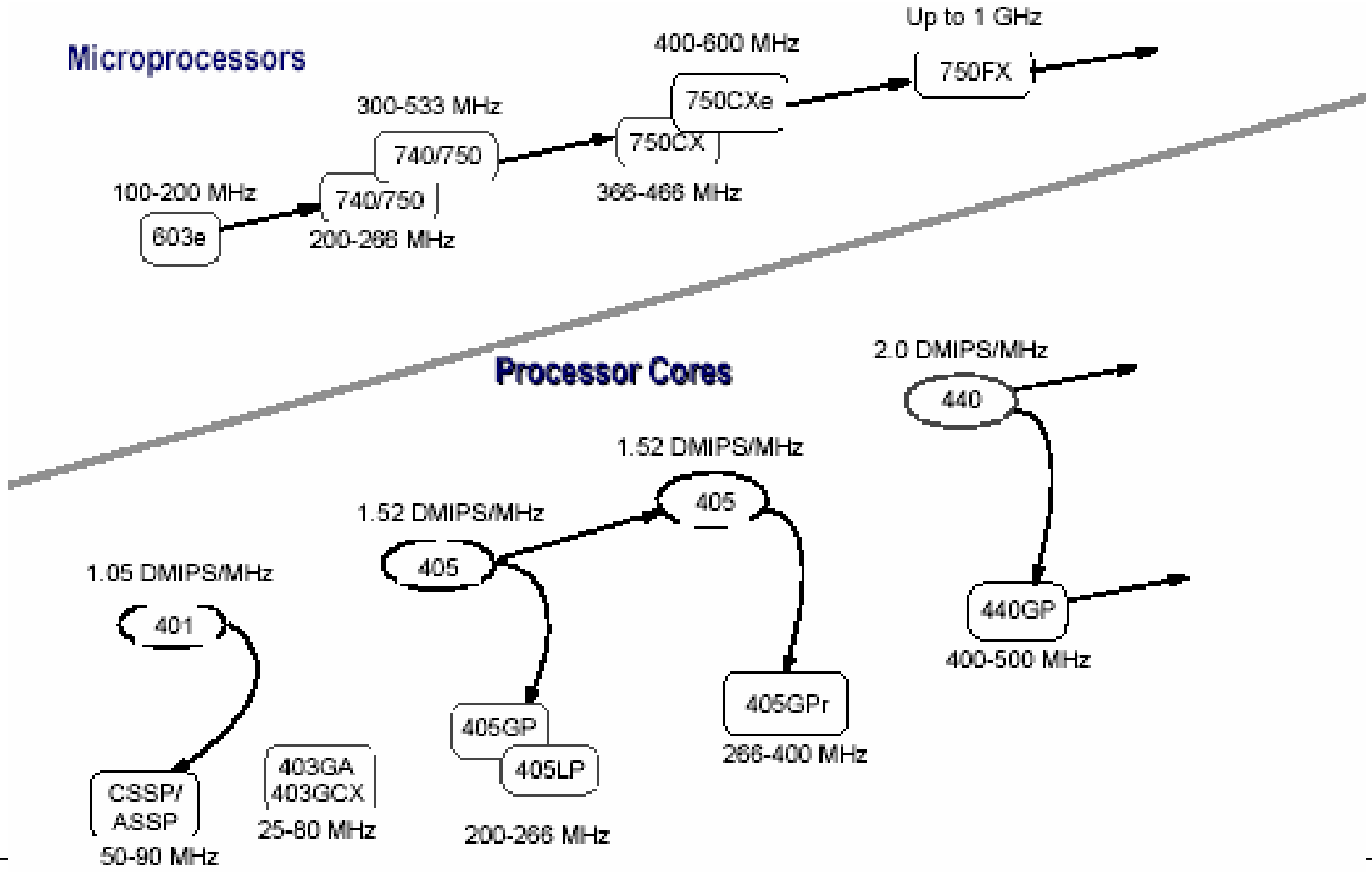


Panorama de processeurs

- Power PC 440 GP Embbeded
  - fréquence : 400 à 500 Mhz
  - 7 étages pipeline
  - Superscalaire de degré 2
  - cache instructions de 32 Ko, cache données de 32Ko
  - technologie : 0,18 micron
  - prédiction de branchement dynamique
  - 24 instructions de type DSP
  - mémoire adressable : 64 Go (36 bits d'adresses)
  - tensions d'alimentation :
    - 1,8 V pour la logique
    - 2,5 V pour la mémoire SDRAM
    - 3,3 V pour les I/O

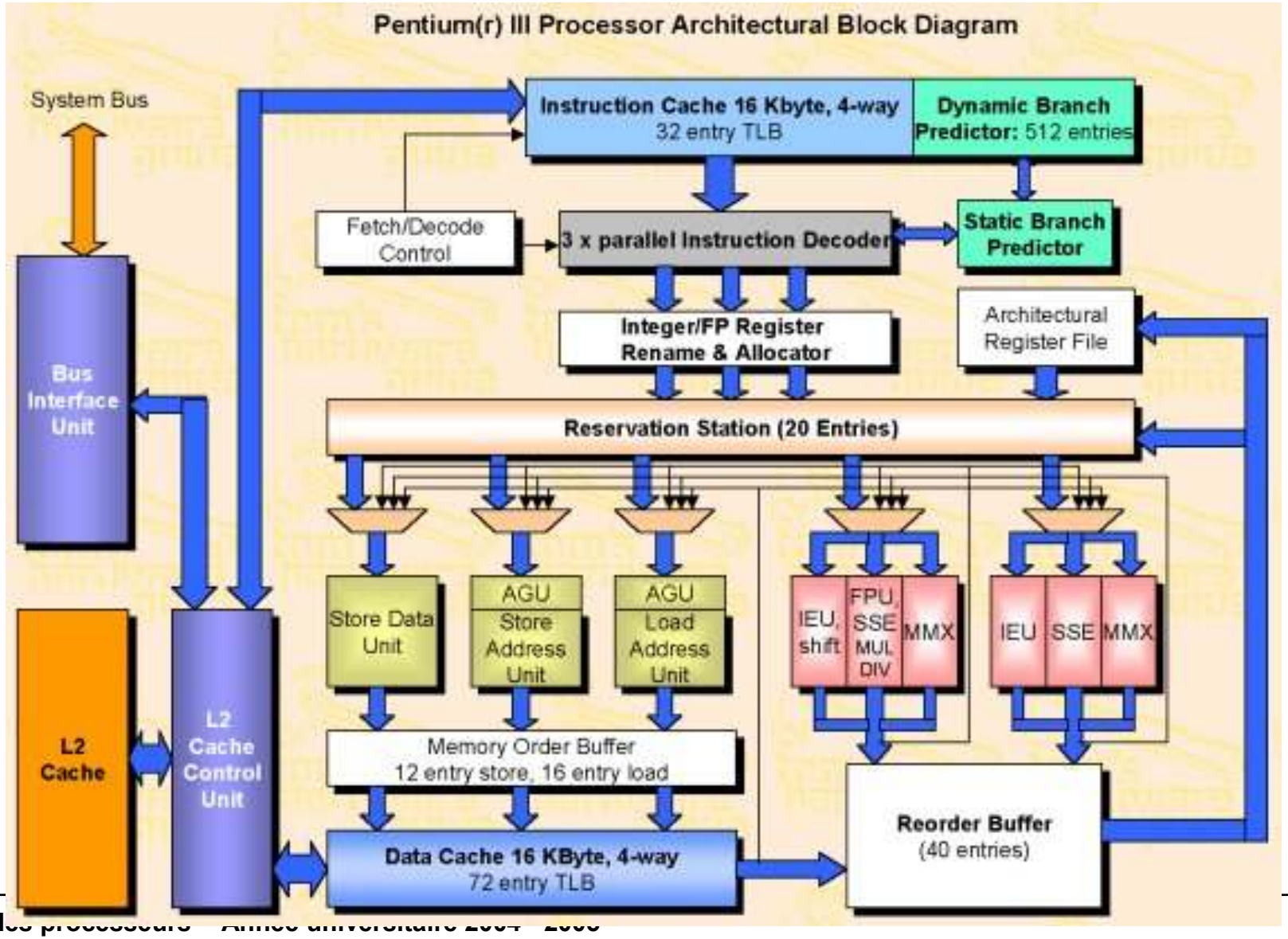
Panorama de processeurs

❑ Power Pc Road Map



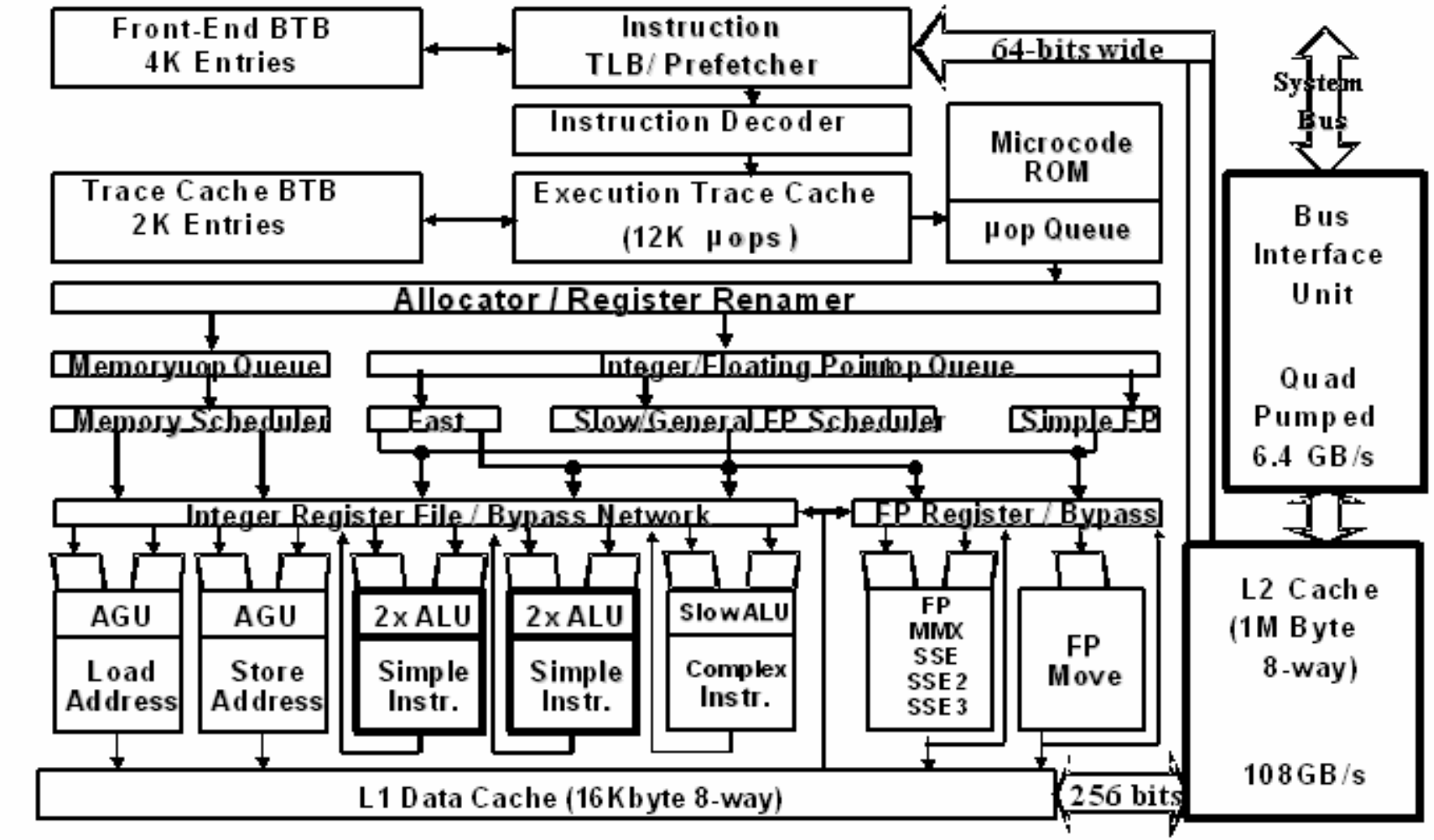
Panorama de processeurs

❑ Pentium III : architecture



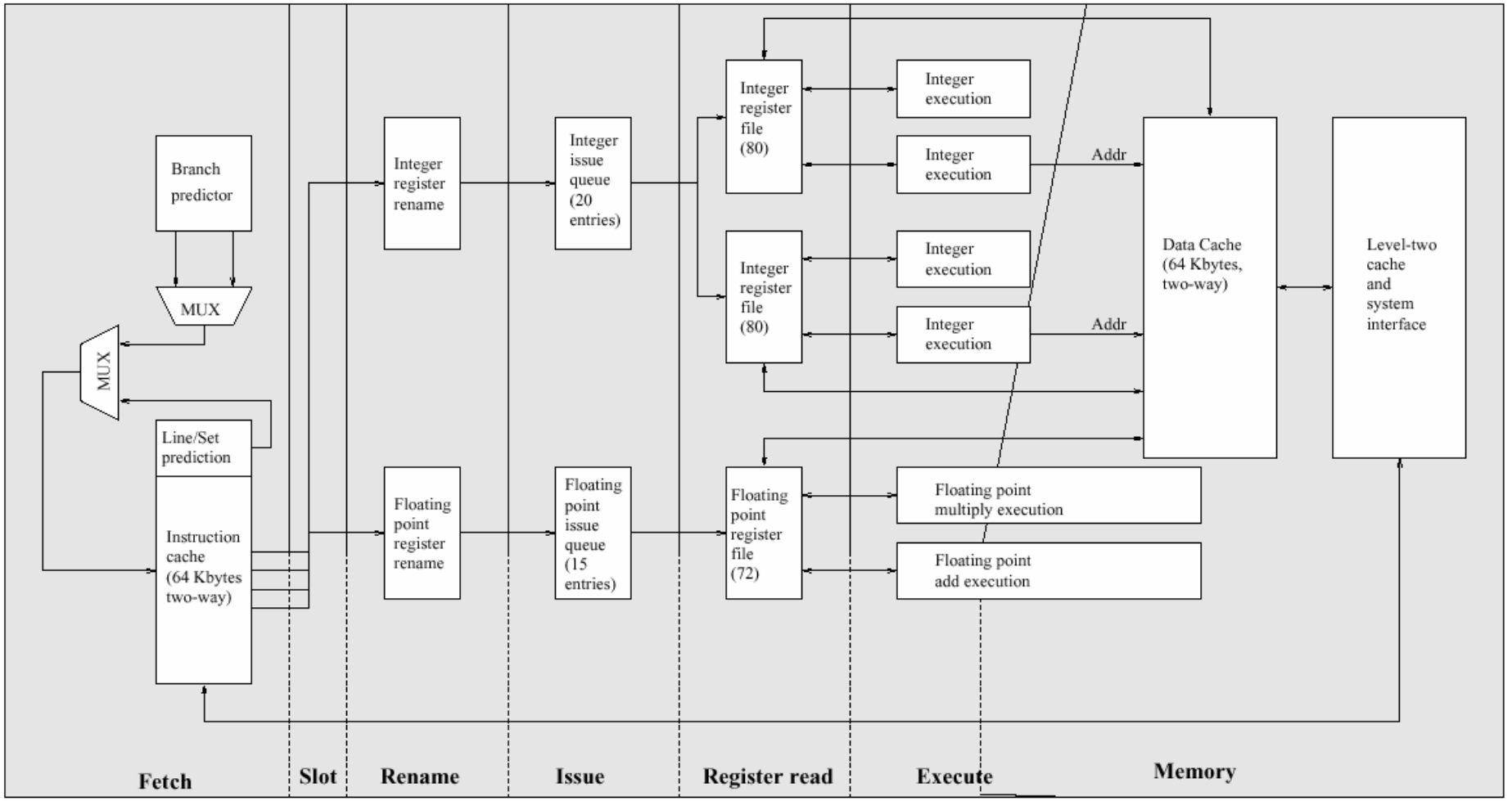
Panorama de processeurs

❑ Prescott's architecture (Pentium 4)



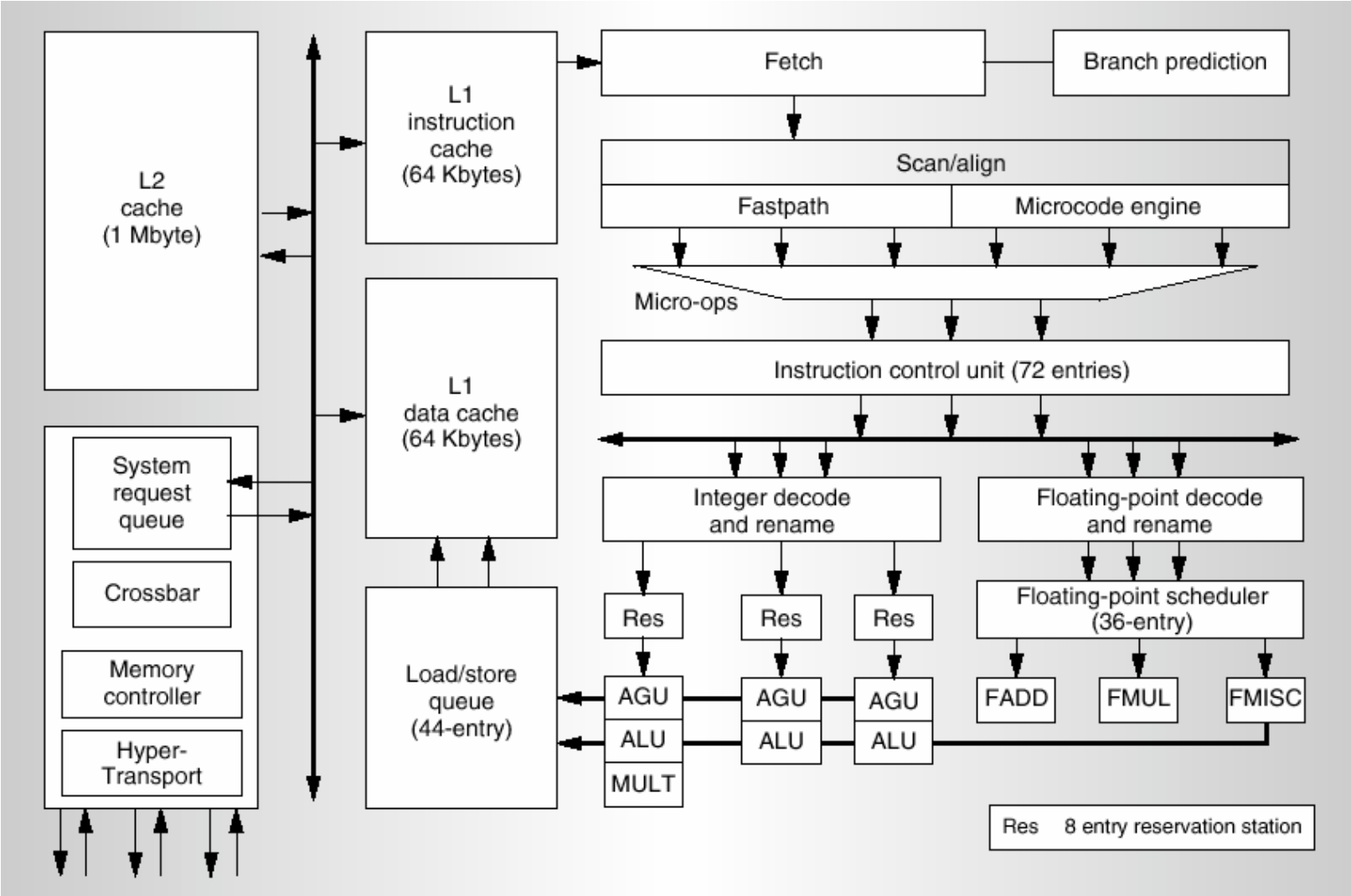
Panorama de processeurs

❑ Dec Alpha 21264 : architecture



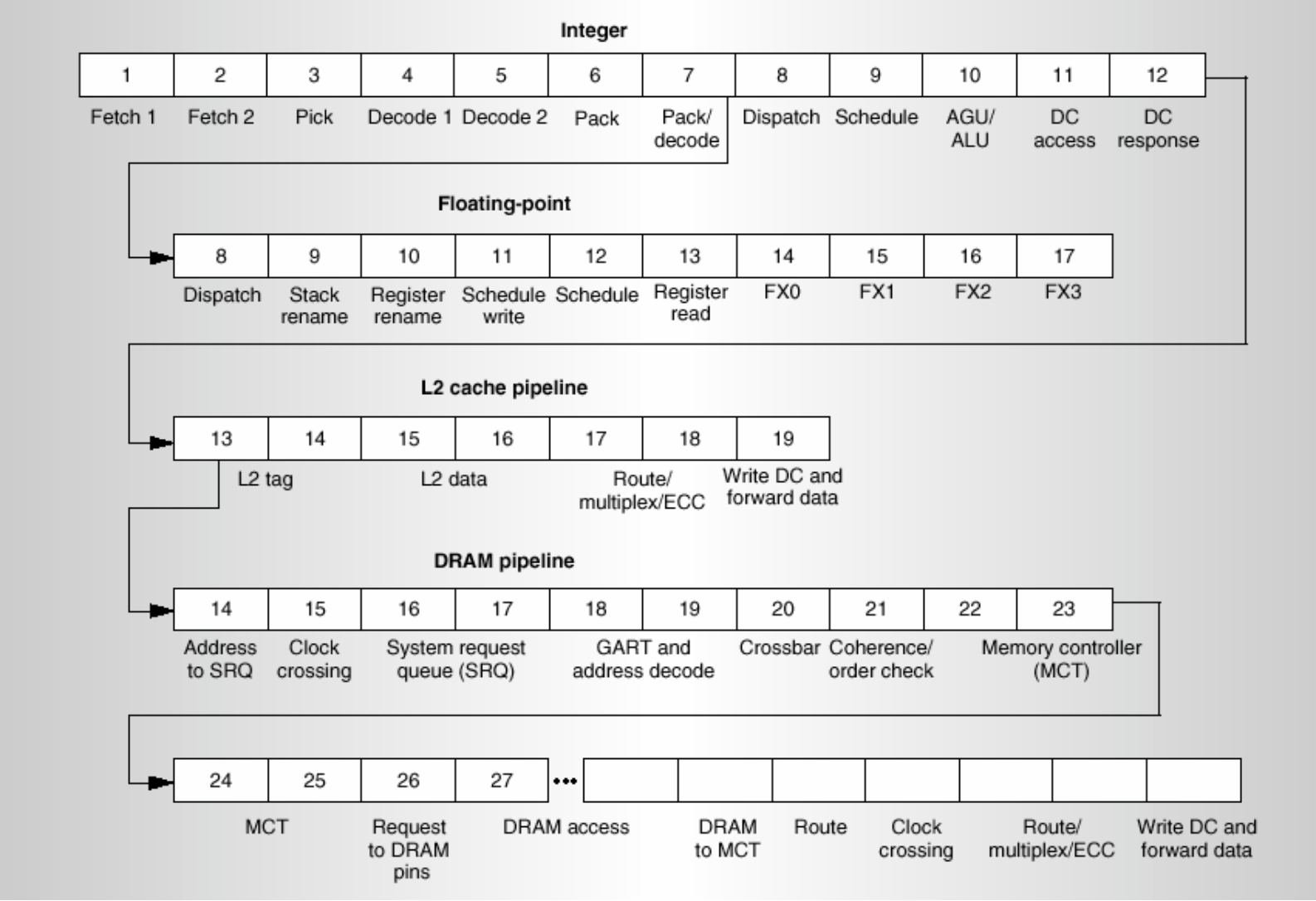
Panorama de processeurs

❑ L'Opteron de AMD



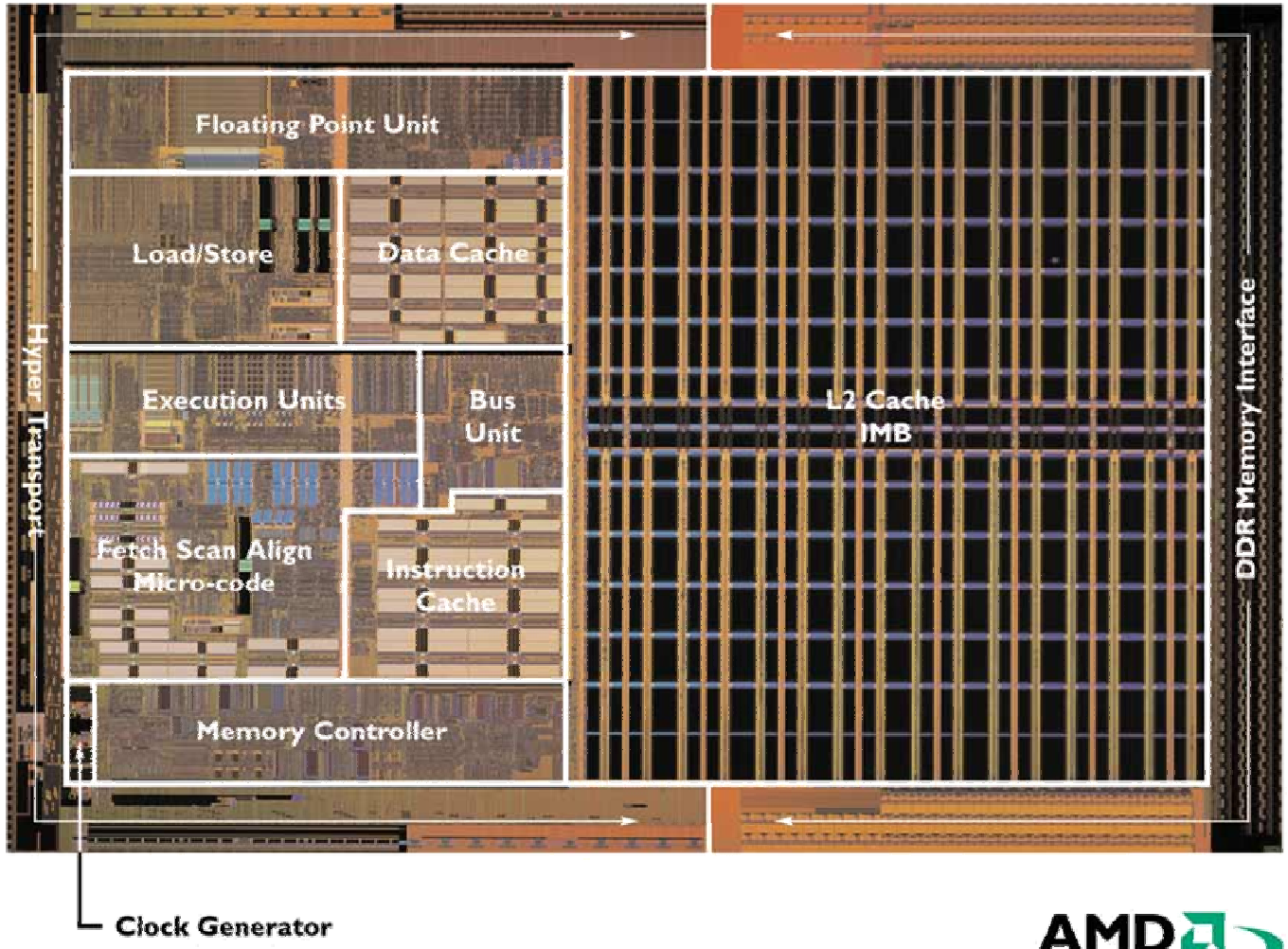
Panorama de processeurs

❑ Le pipeline de l'Opteron de AMD



Panorama de processeurs

❑ Le Layout de l'Opteron



Panorama de processeurs

- **L'architecture IA 64 : Processeur Itanium :**
  - ➢architecture développée conjointement entre Intel et Hewlett Packard
  - ➢c'est une machine RISC 64 bits
  - ➢la base de cette architecture est le processeur PA-RISC de chez HP
  - ➢capable de tourner du code IA 32, compatibilité oblige

Panorama de processeurs

➢ Problèmes avec le Pentium II :

- ♦ instructions de longueurs très variables
- ♦ énormément de formats différents
- ♦ les instructions peuvent référencer la mémoire

Conduit à un contrôleur très complexe

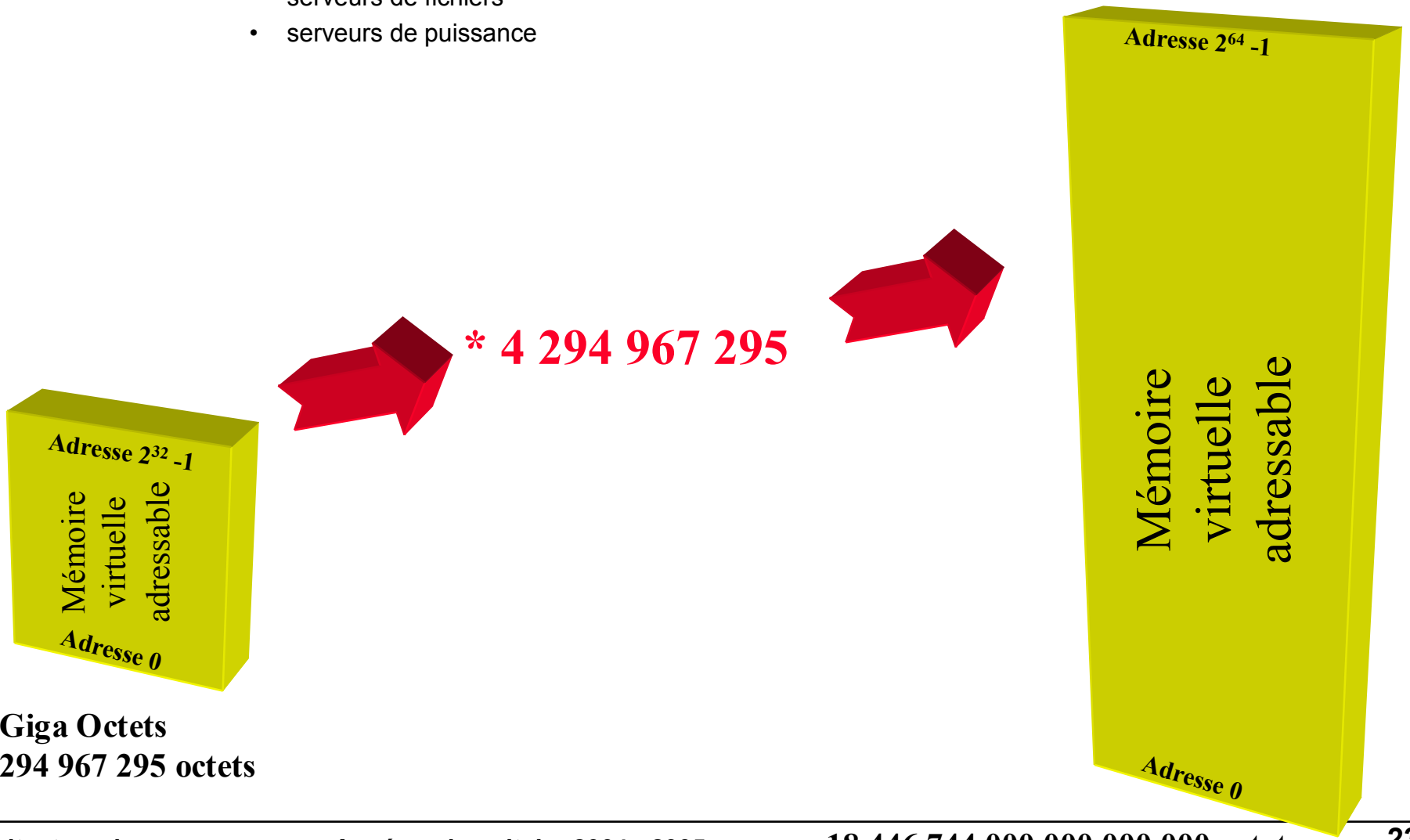
Grand nombre de transistors pour le décodage, le contrôle du pipeline

Alors qu'on pourrait utiliser la place pour d'autres choses, par exemple un cache L1 plus grand

- ♦ Pour arriver à tout faire rapidement :
  - nécessite la mise en place d'un pipeline long :
    - pipeline sur 12 étages
    - dans ce cas il faut une très bonne prédiction de branchement faute de quoi un grand nombre de cycles sont perdus lors des mauvaises prédictions

Panorama de processeurs

- l'architecture IA 32, ne peut adresser "que" 4 Goctets de mémoire :
  - ça commence à faire juste
    - serveurs de fichiers
    - serveurs de puissance



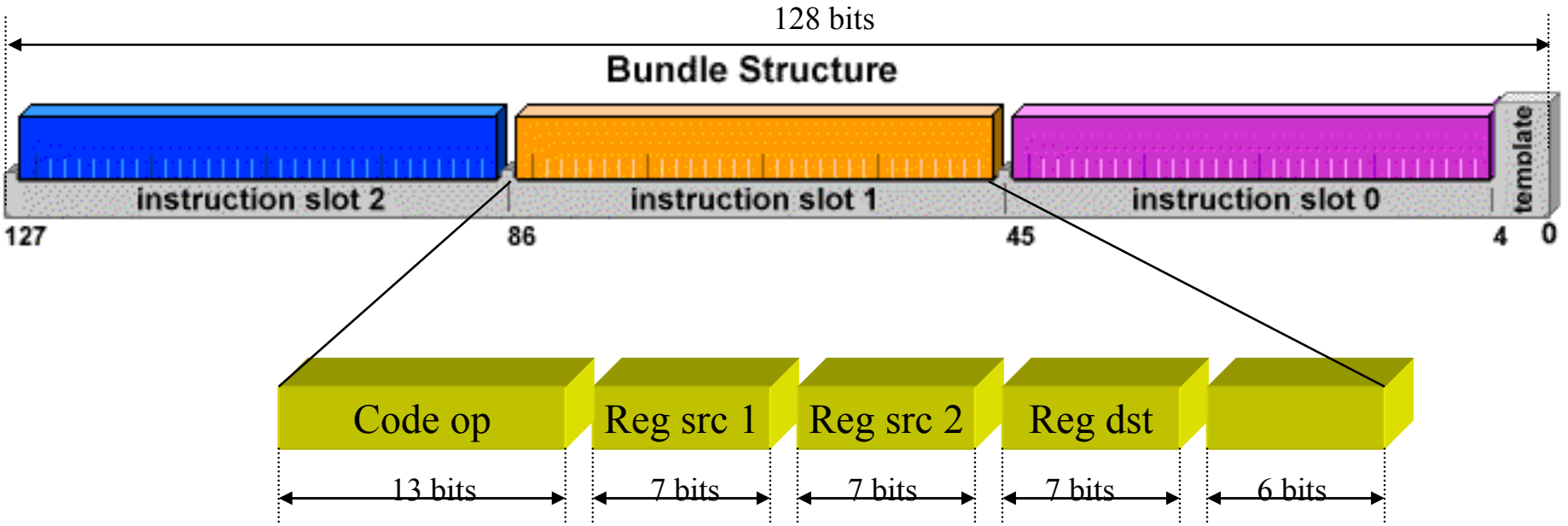
4 Giga Octets
4 294 967 295 octets

18 446 744 000 000 000 000 octets

Panorama de processeurs

➢Description de l'architecture :

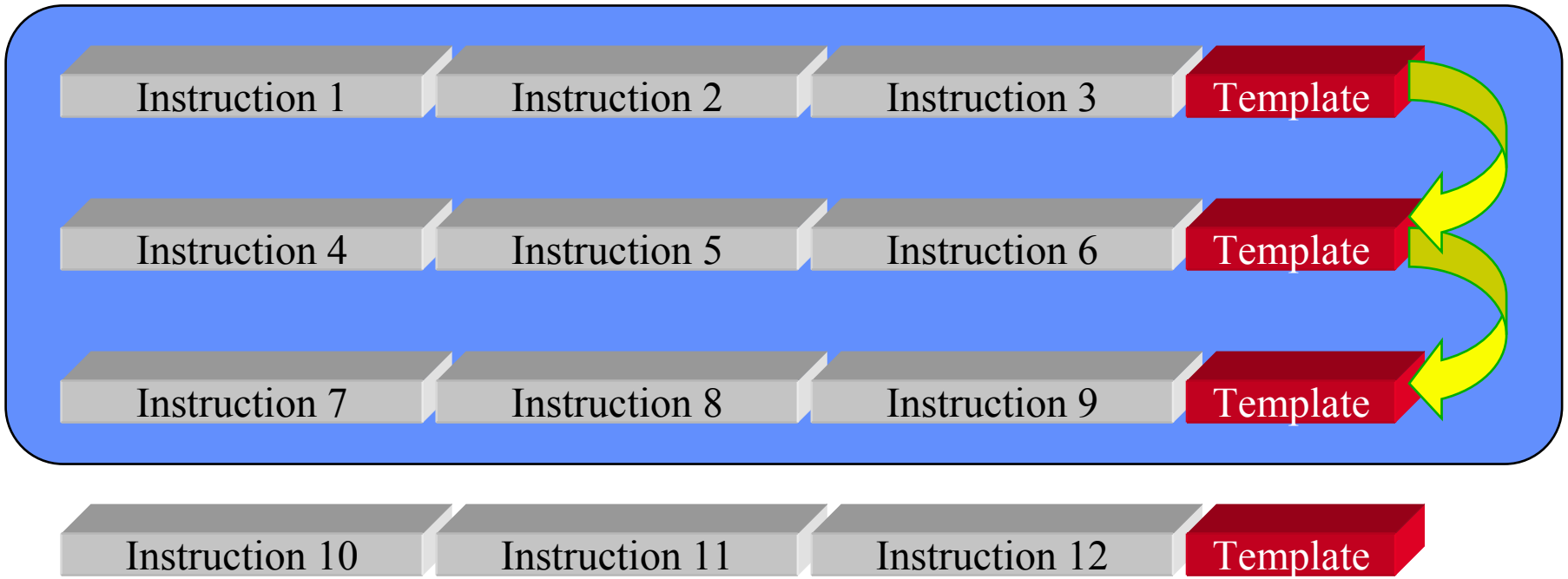
- ♦ 256 registres généraux 64 bits (128 entiers, 128 flottants)
- ♦ les instructions sont regroupées par 3, c'est un petit VLIW (LIV)



- ♦ les instructions peuvent comporter 3 opérandes :
  - – 2 opérandes sources, 1 opérande destination

Panorama de processeurs

- L'architecture définit des bundles :
  - ce sont des ensembles d'instructions traitées si possible ensemble
  - le champ template définit le parallélisme dans le groupe de 3 instructions
  - un bit du template indique si les bundles sont chaînés

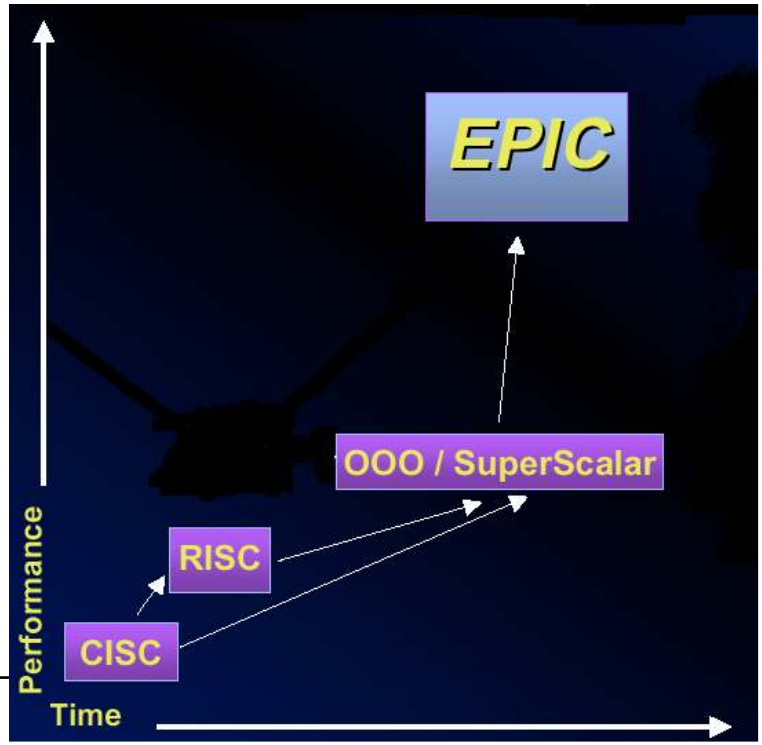


Panorama de processeurs

♦ Rôle du compilateur :

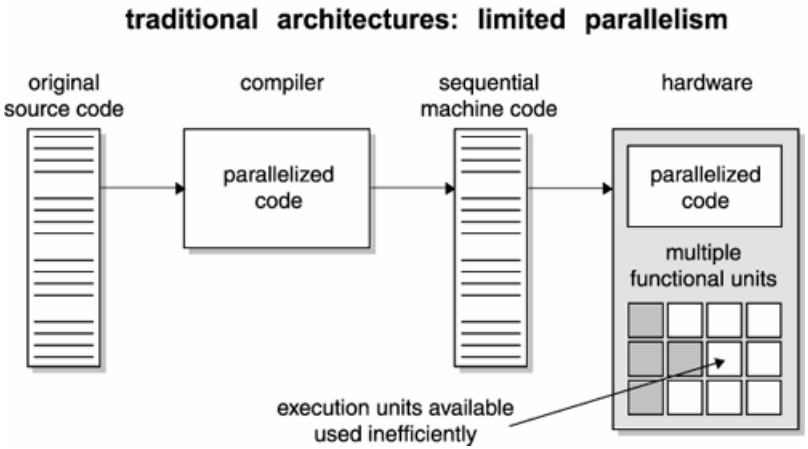
- regroupement des instructions de façon à exploiter le maximum de parallélisme
- toute la complexité est renvoyée vers le logiciel, ce qui décharge d'autant le matériel :
  - la surface peut alors être consacrée à autre chose
- modèle EPIC : Explicitly Parallel Instruction Computing

- actuellement une grosse partie de la complexité du processeur (superscalaire) est utilisée pour l'exécution dans le désordre

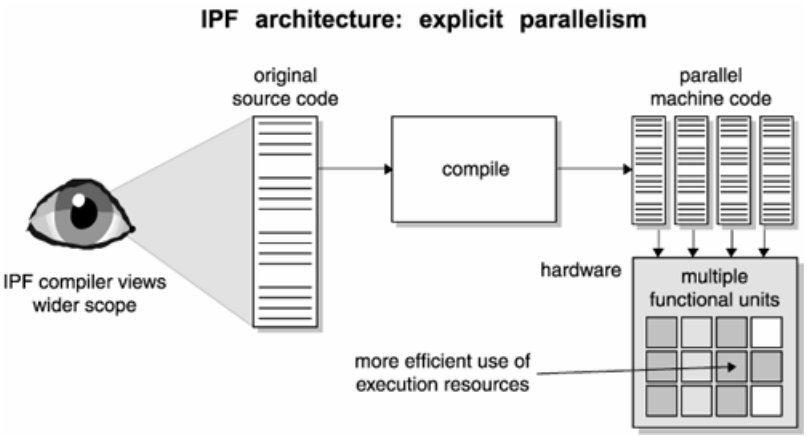


Panorama de processeurs

♦ Processeurs traditionnels



♦ Processeurs EPIC



Panorama de processeurs

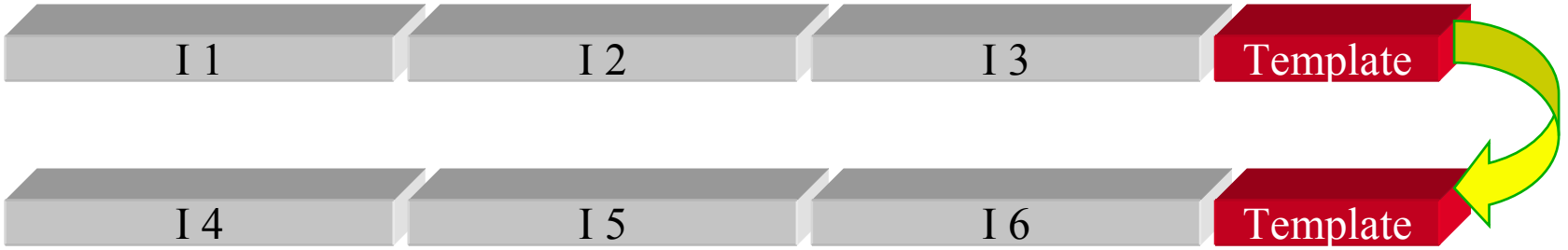
- 2 types de processeurs :
  - low-end CPU :
    - le processeur peut traiter 1 seul bundle par cycle
    - on ne passe au bundle suivant que lorsque le bundle courant est terminé
  - high-end CPU :
    - le processeur peut traiter plusieurs bundles par cycle :
      - prise en compte du chaînage des bundles
    - le processeur peut lancer un nouveau bundle alors que les précédents ne sont pas terminés :
      - type d'exécution superscalaire

Panorama de processeurs

♦ Parallélisme adapté à la machine :

– le champ template définit quelles opérations peuvent s'exécuter en parallèle indépendamment de la machine :

• exemple : le template indique que les instructions I1, I2, I3, I4 peuvent s'exécuter en parallèle



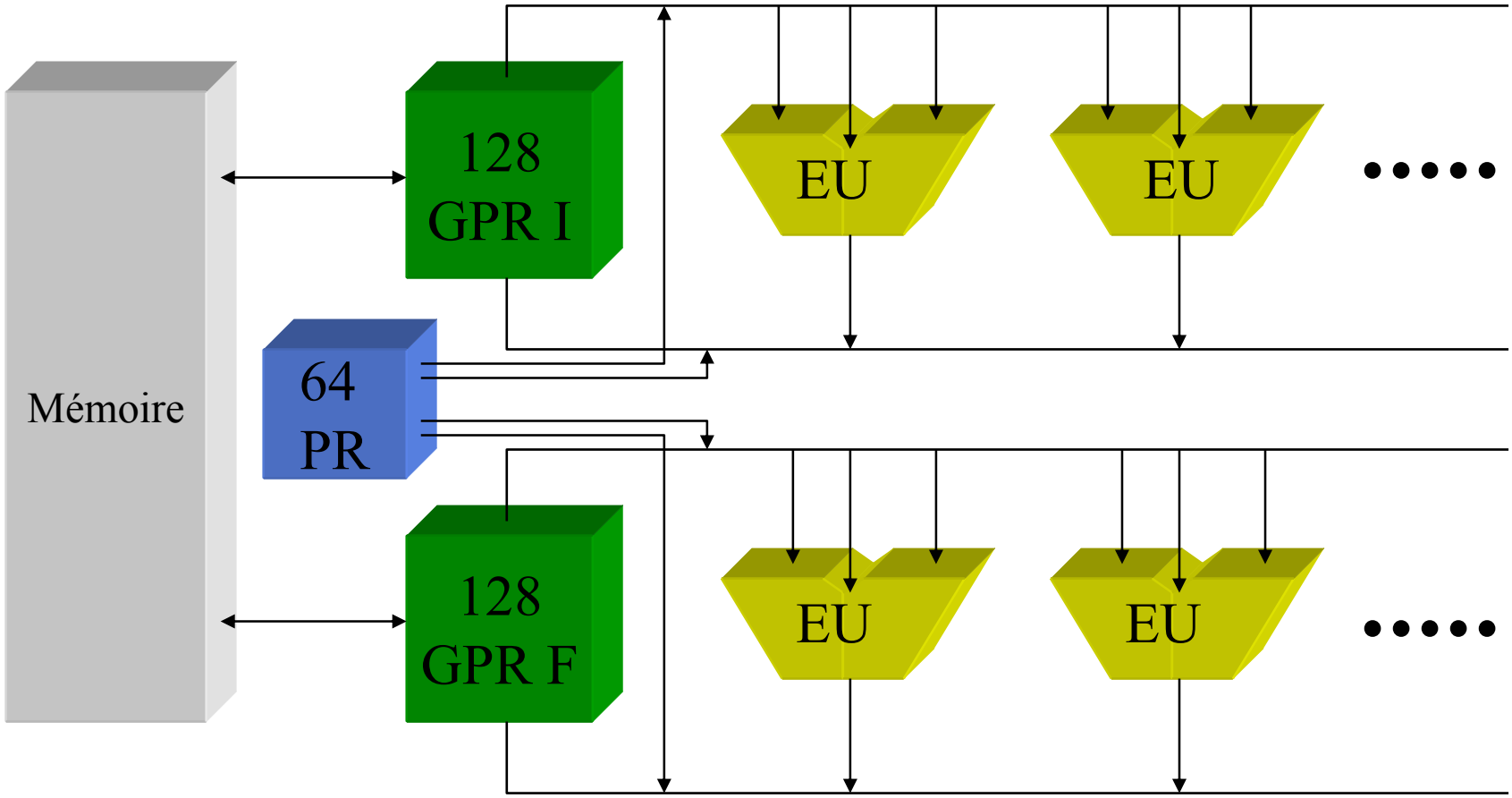
– en fonction du nombre d'unités fonctionnelles disponibles la machine lancera 1, 2, 3 ou 4 instructions en parallèle :

• exemple : si la machine possède 2 unités fonctionnelles, alors 2 instructions seront lancées

• si un nouveau Itanium est développé avec 4 unités fonctionnelles, alors le processeur lancera 4 instructions en parallèle

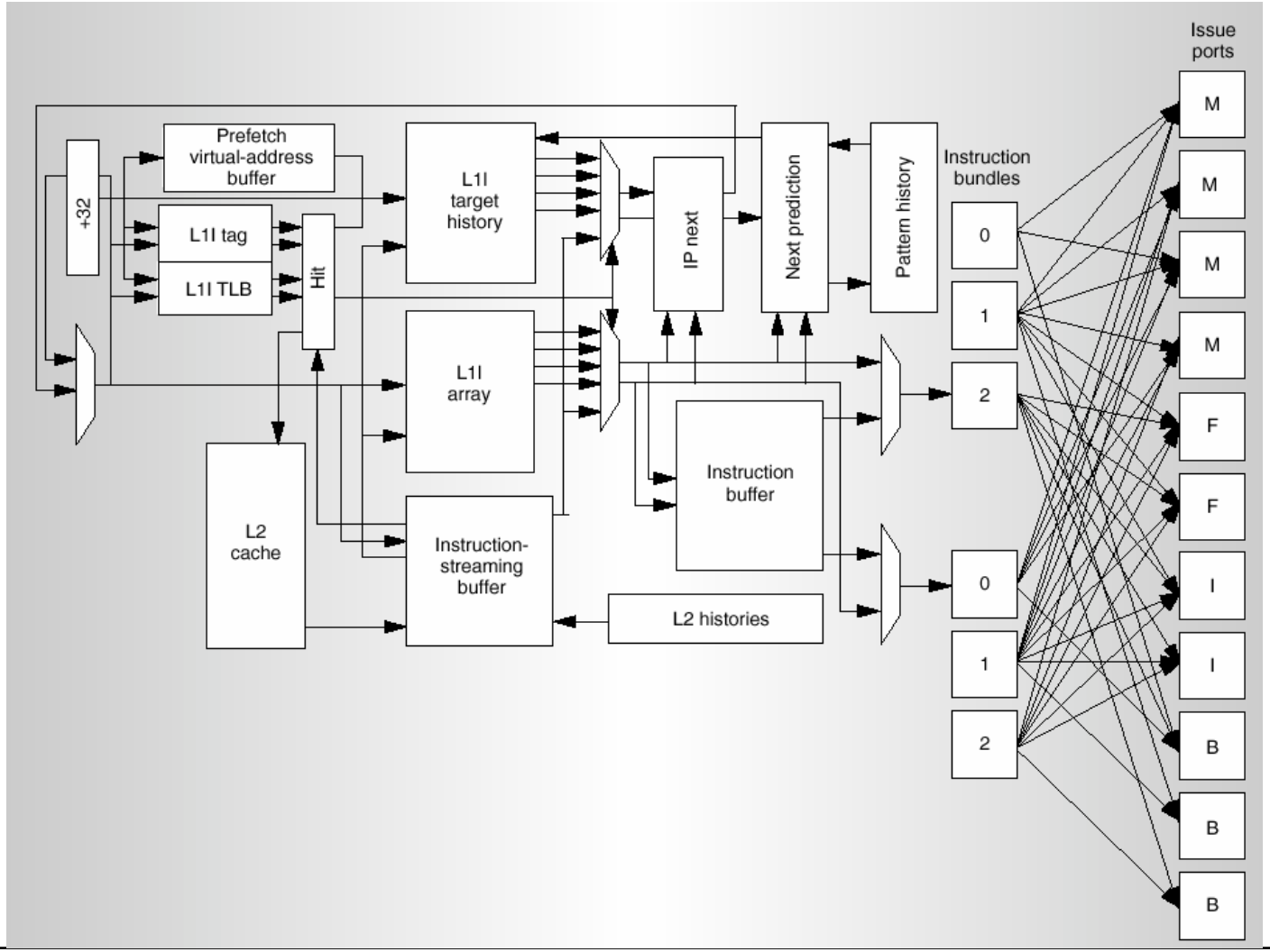
Panorama de processeurs

➢Exemple d'une version de processeur Itanium



Panorama de processeurs

➢Front end (Itanium 2) permettant de dispatcher les instructions

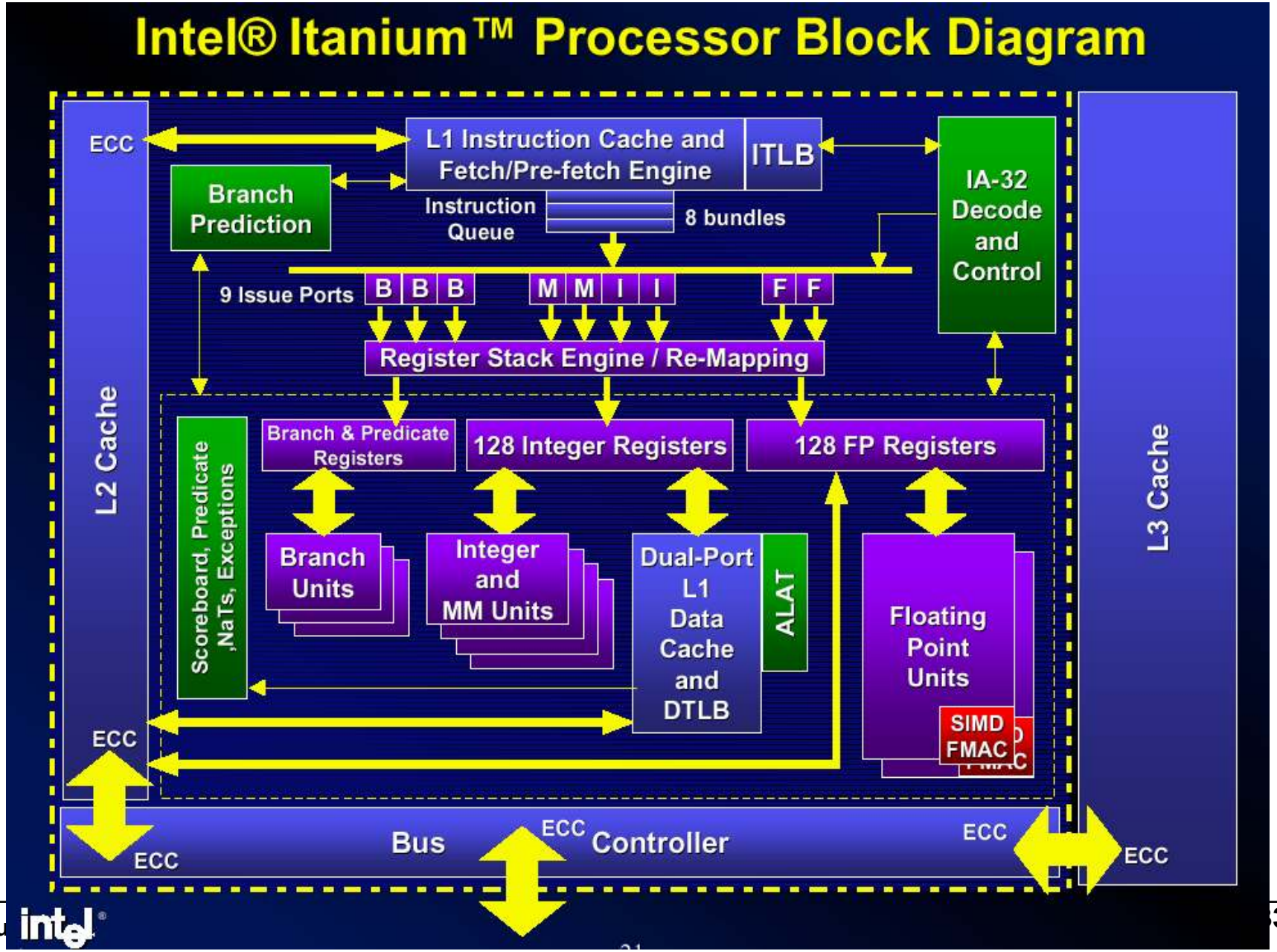


Panorama de processeurs

➢ Le pipeline du processeur Itanium :

♦ 10 étages

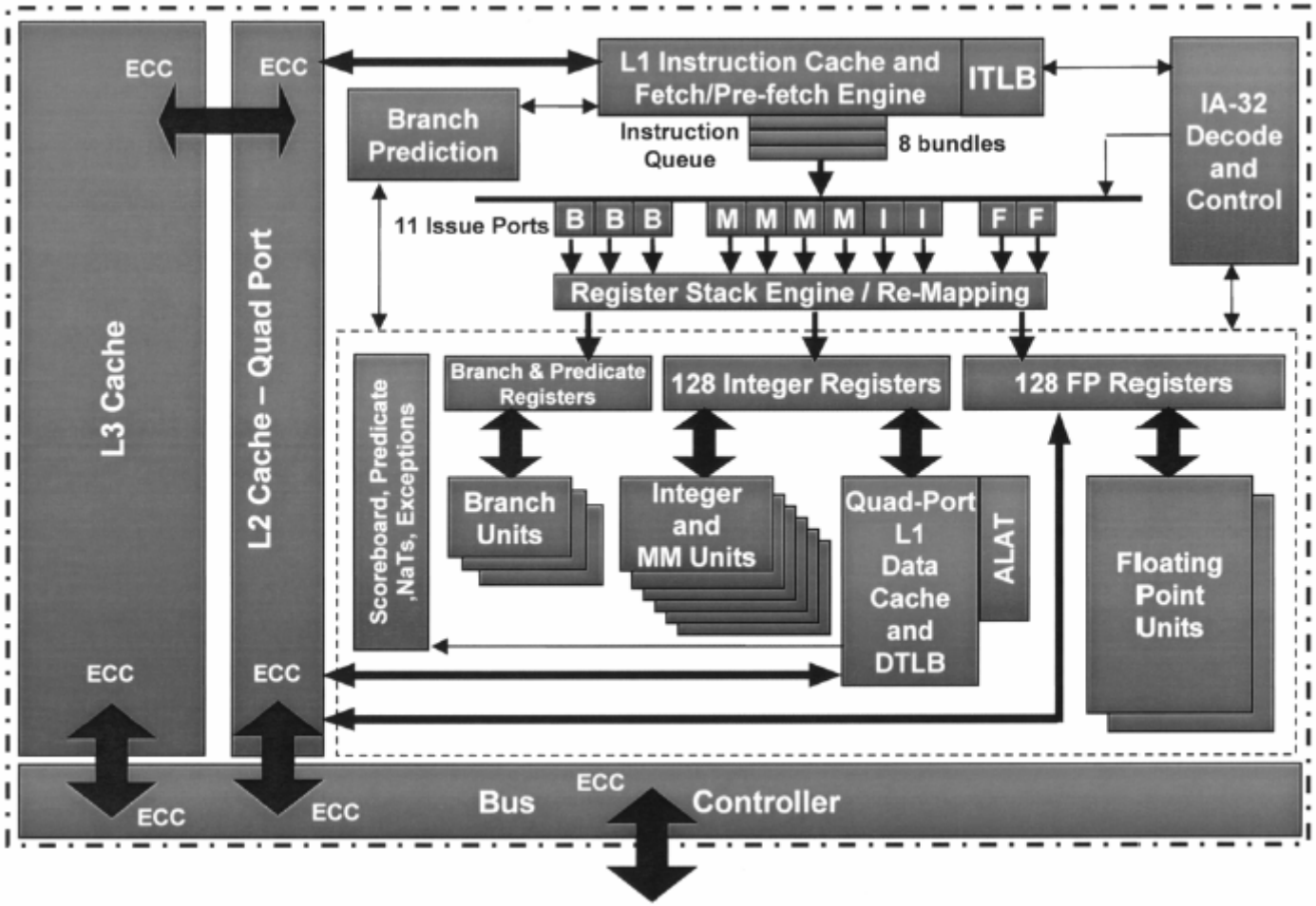
Panorama de processeurs



Panorama de processeurs

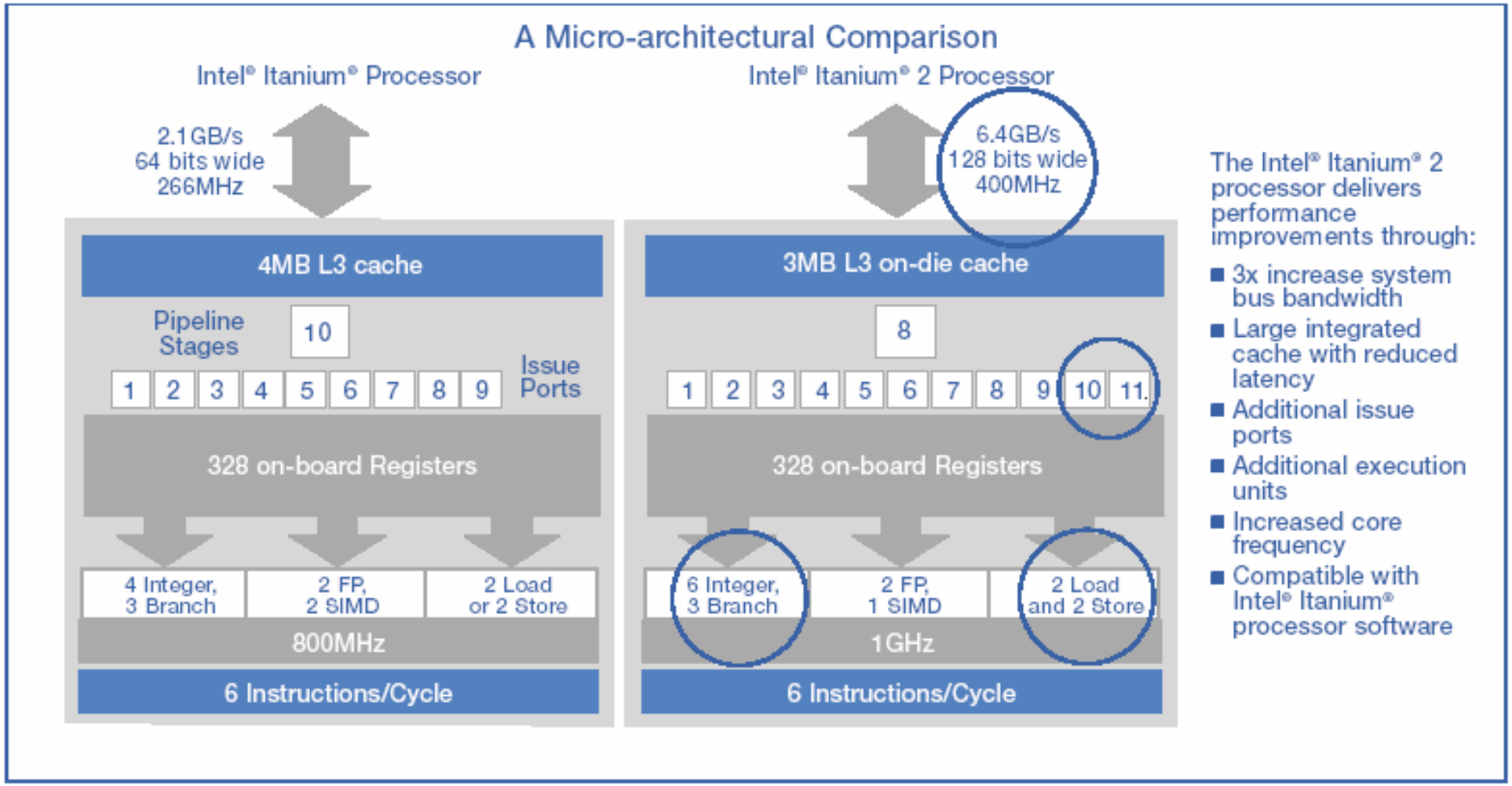
➢Exemple d'une version de processeur Itanium :

♦ Itanium 2



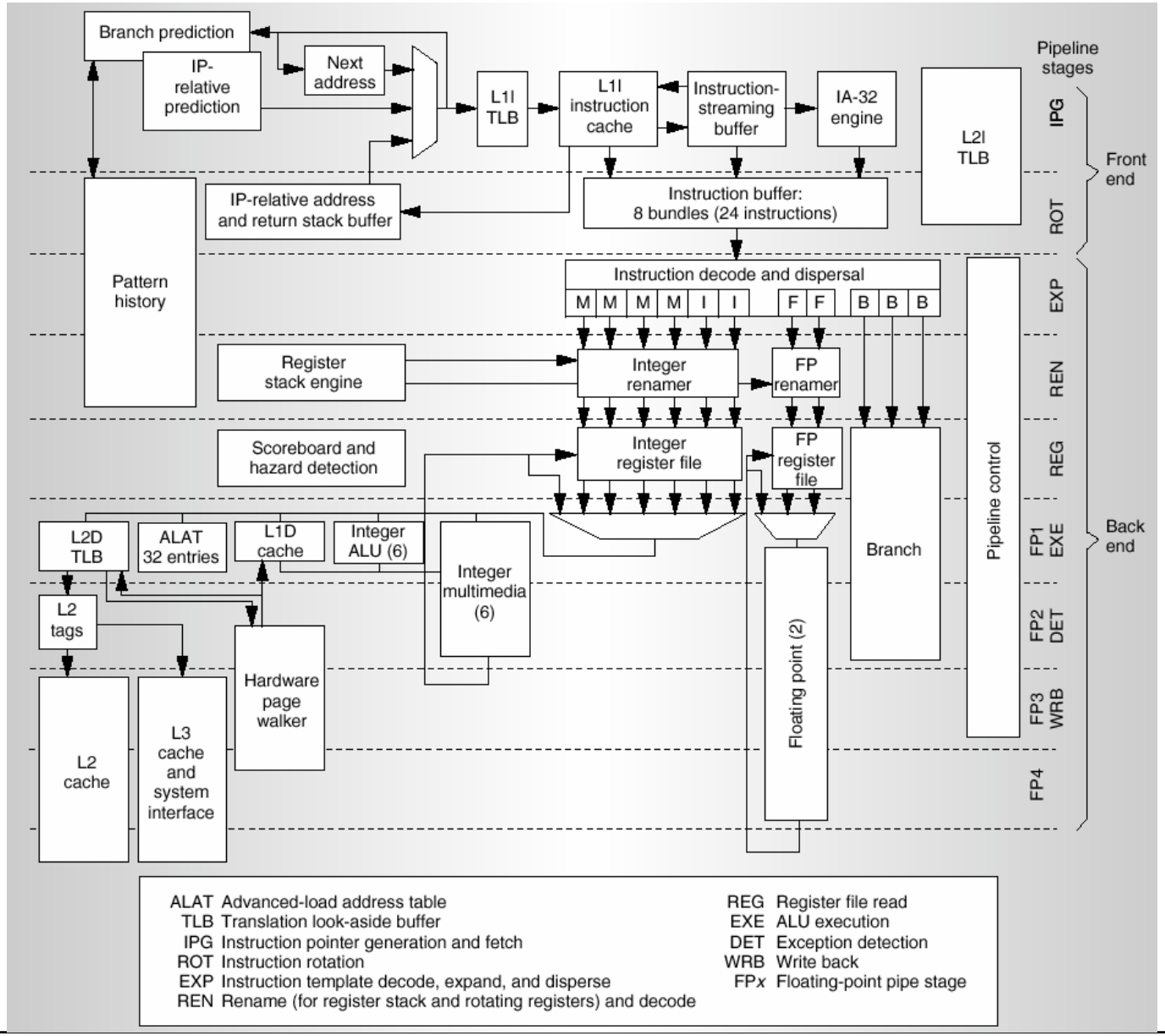
Panorama de processeurs

➢Itanium et Itanium 2

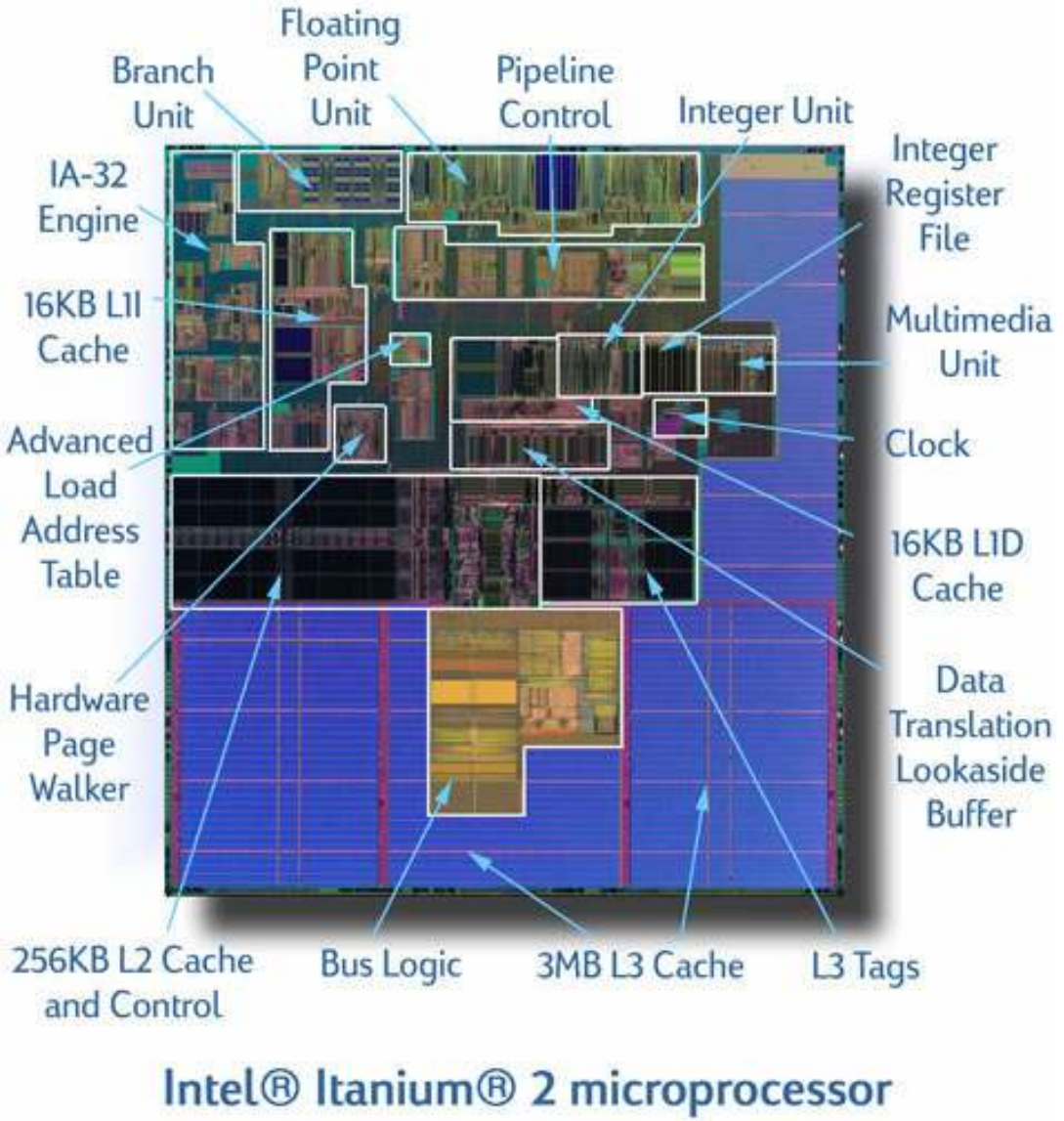


Source Intel Corporation

Panorama de processeurs



Panorama de processeurs



Panorama de processeurs

➢ Limitation des pertes de cycles :

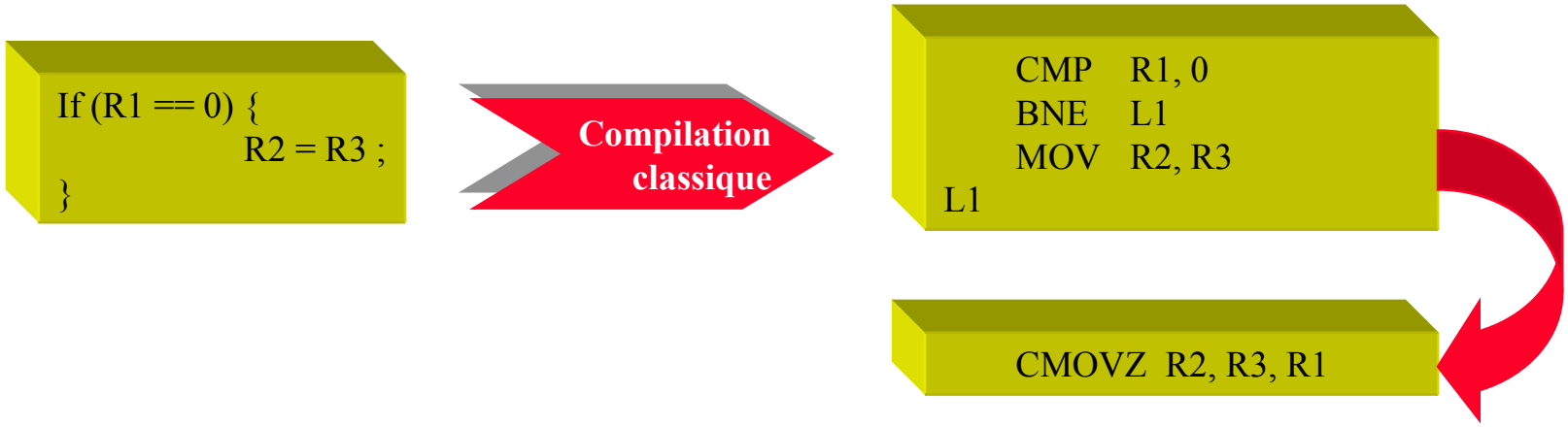
♦ la longueur du pipeline impose :

– soit une très bonne prédiction de branchement (intel estime que 20 à 30 % de la puissance de calcul des processeurs est consommée par les mauvaises prédictions)

– une autre solution consiste à supprimer certains branchements

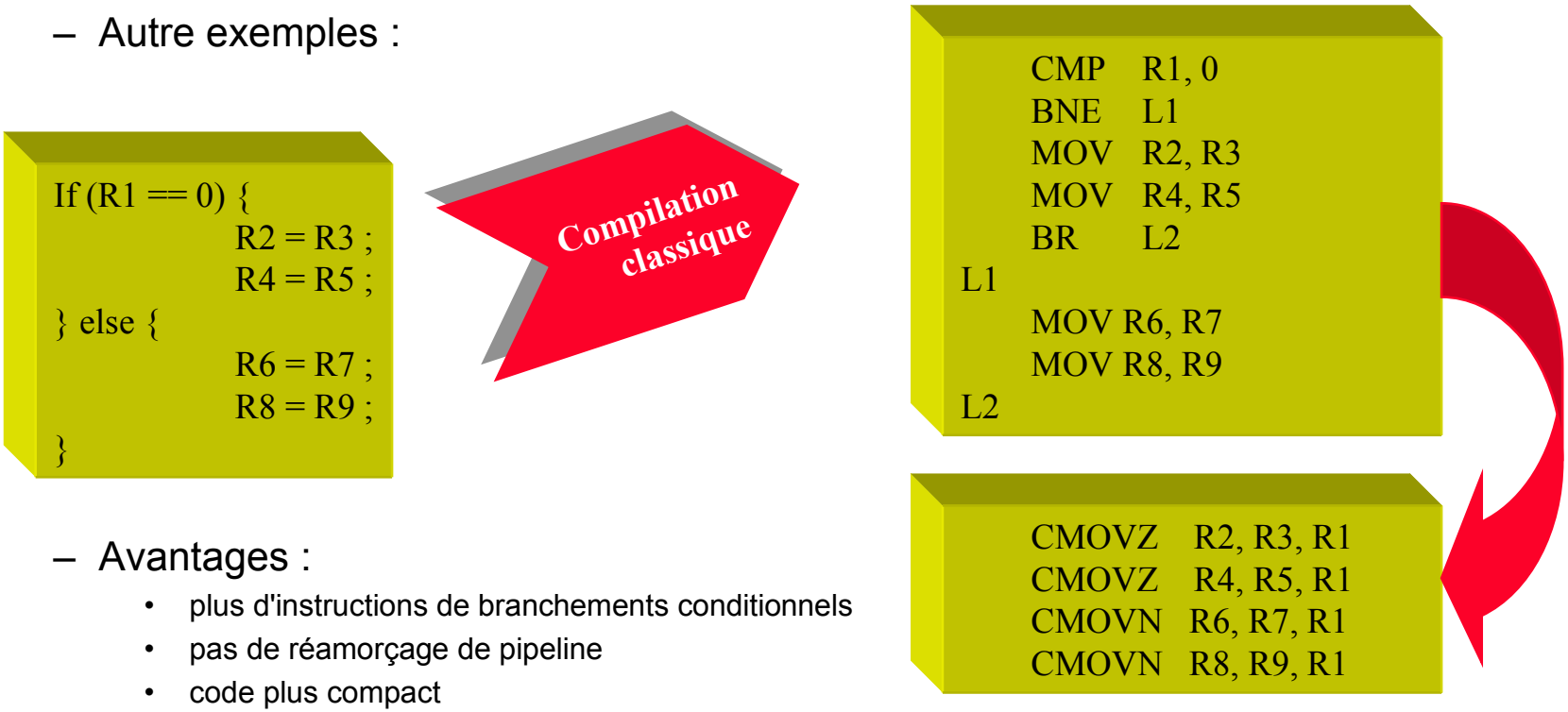
– Exemple :

• soit le code C suivant :



Panorama de processeurs

– Autre exemples :

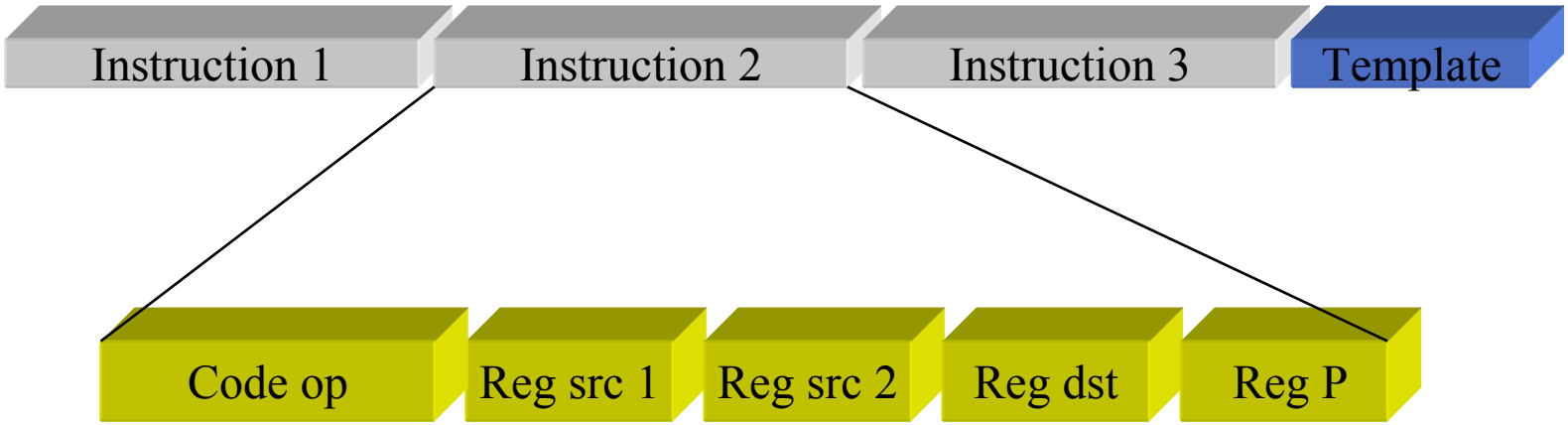


– Avantages :
  - plus d'instructions de branchements conditionnels
  - pas de réamorçage de pipeline
  - code plus compact
– Inconvénients :
  - deux instructions vont se comporter comme des NOP
  - pas utilisable dans tous les cas, mais quand même

– Remarque : les deux branches de la structure conditionnelles sont exécutées, mais une seule est retenue

Panorama de processeurs

- Le registre de "test" :
  - predicate register
  - est codé dans les 6 dernier bits



Panorama de processeurs

- Dans le processeur Itanium, toutes les instructions sont conditionnelles :
  - les instructions de comparaison positionnent les registres Predicate
  - exemple, soit le code :

P1, P2 = cmp (A == 0)

P1 = (A == 0)
P2 = (A != 0)

A	P1	P2
0	1	0
autre	0	1

Panorama de processeurs

♦ Utilisation de ces instructions :

```
If (a && b)
                j = j +1 ;
else
                if ( c )
                                k = k + 1 ;
                else
                                k = k -1 ;
i = i + 1 ;
```

Compilation classique

```
        beq     a, 0, L1
        beq     b, 0, L1
        add     j, j, 1
        jump    L3
L1
        beq     c, 0, L2
        add     k, k, 1
        jump    L3
L2
        sub     k, k, 1
L3
        add     i, i, 1
```

```
                P1, P2   = cmp   (a == 0)
<P2>            P1, P3   = cmp   (b == 0)
<P3>                       add   j, j, 1
<P1>            P4, P5   = cmp   (c != 0)
<P4>                       add   k, k, 1
<P5>                       sub   k, k, 1
                           add   i, i, 1
```

– Remarque : les deux branches de la structure conditionnelles sont exécutées, mais une seule est retenue

Panorama de processeurs

➢Format général des instructions :

1)	<Pi>			instruction
2)		Pj, Pk	= cmp	(relation)
3)	<Pi>	Pj, Pk	= cmp	(relation)

- ♦ format 1 :
  - instruction conditionnelle, l'instruction sera exécutée si et seulement si Pi est vraie :
    - en faite l'instruction s'exécute mais l'étage d'écriture de résultat est inhibé si la condition est fausse
- ♦ format 2 :
  - la comparaison positionne les registres *predicate* :
    - Pj = vraie si la relation est vraie, à faux sinon
    - Pk = faux si la relation est vraie, à vraie sinon
- ♦ format 3 :
  - la comparaison positionne les registres predicate si et seulement si Pi est vraie

Panorama de processeurs

- Les chargements spéculatifs :
  - ils ont pour rôle de demander la donnée à la mémoire avant que celle-ci ne soit éventuellement utiliser
- Un Load spéculatif provoque :
  - une fetch mémoire : le fetch peut avoir deux conséquences :
    - si le load échoue
- L'instruction Check vérifie la disponibilité de la donnée :
  - si la donnée est disponible alors le check se comporte comme une instruction nop
  - si la donnée n'est pas disponible, alors l'exception est exécutée

Panorama de processeurs

♦ Exemple :

```
If (b[j] == true) && (a[i+j] == true) && (c[i-j+7] == true)) then
                .....
else
                .....
end
```

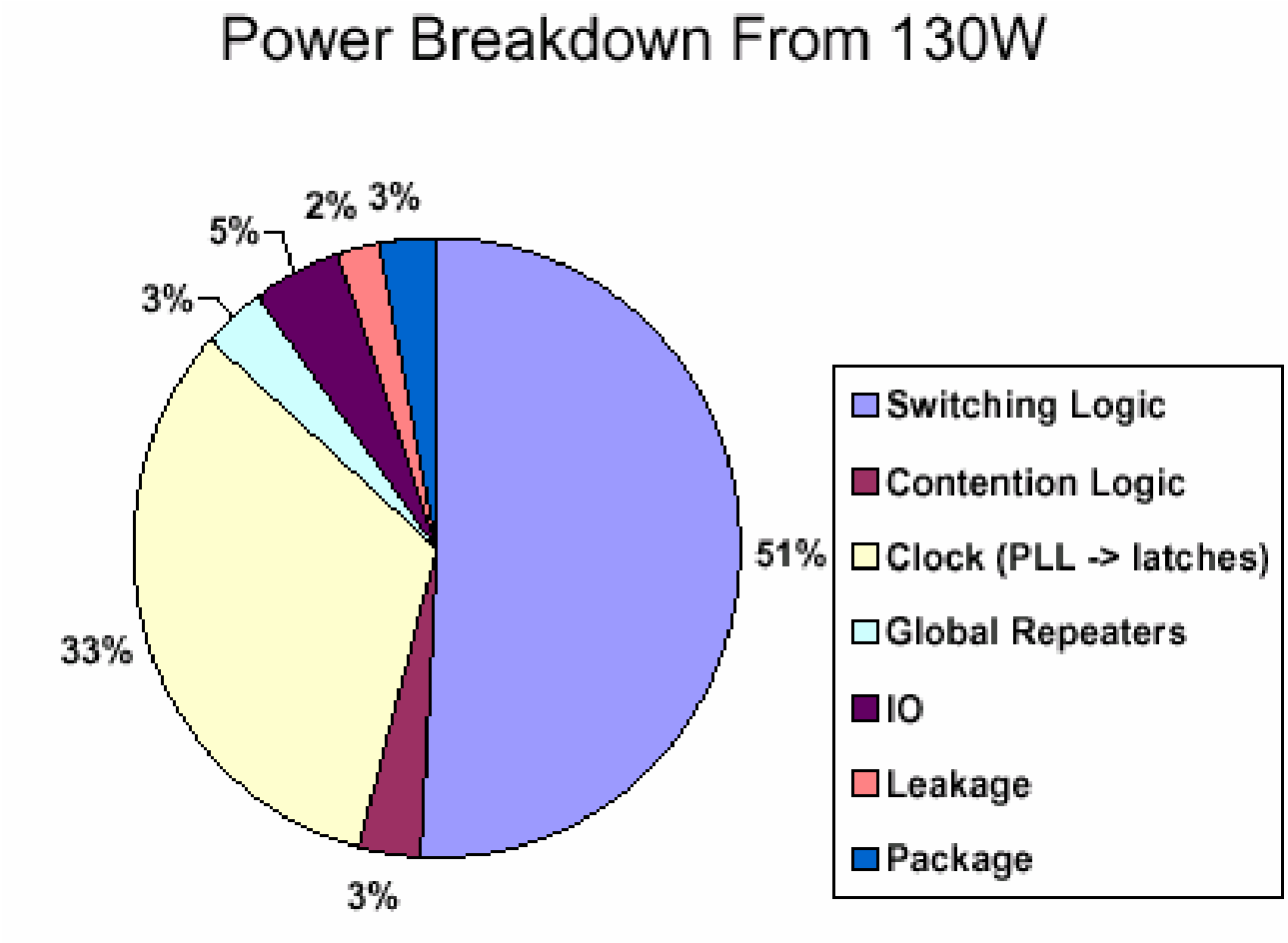
Compilation classique

```
        R1 = &b[j]
        ld  R2, (R1)
        bne R2, 1, L2
        R3 = &a[i+j]
        ld R4, (R3)
        bne R4, 1, L2
        R5 = &c[i-j+7]
        ld R6, (R5)
        bne R6, 1, L2
L1      .....
        ....
L2
        .....
        .....
```

```
                        R1 = &b[j]
                        R3 = &a[i + j]
                        R5 = &c[i - j + 7]
                        ld      R2, (R1)
                        ld.s    R4, (R3)
                        ld.s    R6, (R5)
              P1, P2    = cmp   (R2 == 1)
<P1>                    chk.s R4
<P1>          P1, P2    = cmp   (R4 == 1)
<P1>                    chk.s R6
<P1>          P1, P2    = cmp   (R5 == 1)
<P1>                    Partie then
<P1>                    .......
<P2>                    Partie else
<P2>                    .......
```

Panorama de processeurs

♦ Remarque : la consommation dans le processeur …



Roadmap INTEL

Panorama de processeurs

Les processeurs multi média

- Les objectifs : inclure l'ensemble des besoins multi média dans un seul boîtiers :
  - puissance de calcul : GSM, codage MPEG
  - haute résolution graphique (graphique 3D)
  - téléphonie, Fax, vidéo phone
  - courrier électronique
  - portable, donc faible consommation (3,3 volts)
- Applications concernées :
  - communication spécialisées transmission :
    - capture d'images et de sons : graphique, animation, photo numérique, caméra VT, TV
    - transmission : hertzien terrestre, satellite, câble, ATM
    - restitution : écran, projecteur, haute définition

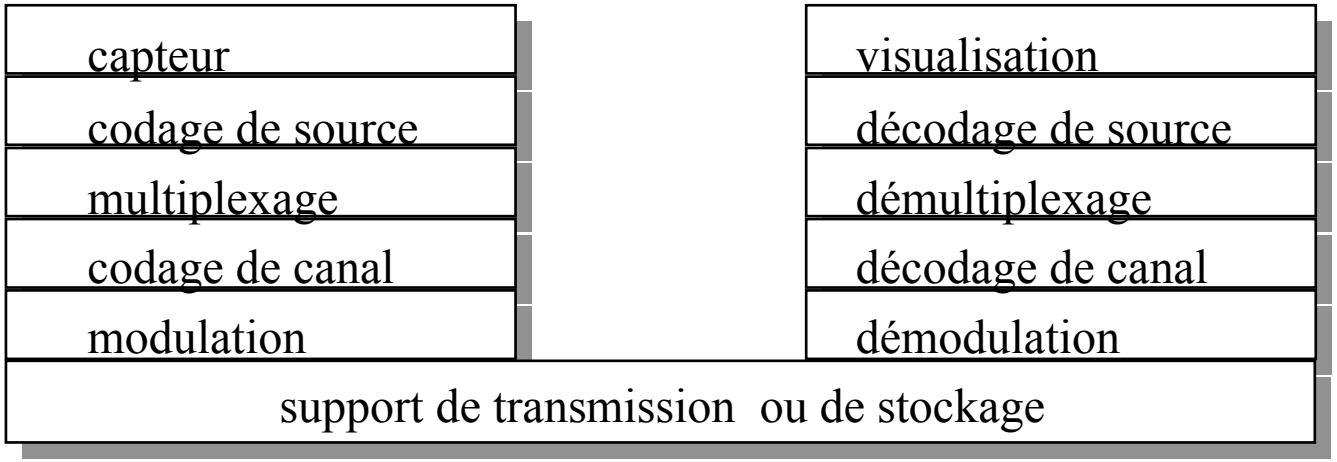
Panorama de processeurs

- stockage : RAM, disque dur, disque magnéto optique, CD rom, DAT

Convergence :

–audiovisuel	- communication
–informatique	- multi média globale,
–télécommunication	- intelligente, interactive

– Exemple :



Panorama de processeurs

Processeurs MMX

- pentium MMX : MultiMédia eXtensions
  - ajout d'instructions spécifiques au multi média (57 instructions supplémentaires)
  - augmentation de performances
  - ce n'est pas un nouveau processeur MAIS simplement une extension
  - les applications doivent être développées spécifiquement pour ce processeur (pour tirer le meilleur partit de ces capacités)
- DEC Alpha :
  - 13 instructions supplémentaires
  - augmentation de 0,6 % de la surface
  - fréquence d'horloge augmentée à 550 Mhz
  - codage MPEG 2 temps réel (complexité =
  - alimentation réduite (2.8 volts)
  - diminution de la consommation

Panorama de processeurs

- Ultra Sparc II :
  - 30 instructions spécifiques (Visual Instruction Set)
  - décodage MPEG2 temps réel
  - augmentation de la surface de 3 % du circuit
  - fréquence augmentée à 250 Mhz
  - 2,5 Gops en pic de traitement
  - tension d'alimentation à 2,6 volts
- MIPS V :
  - extension MMX
  - jusqu'à 8 multiplications 8 bits en parallèle
  - accumulateur sur 192 bits

Panorama de processeurs

Les *vrais* multi média

- Mpact :
  - puissance de calcul : 3 Bops (Gops)
  - fréquence de 125 MHz
  - graphique 2D et 3D
  - 5 ports d'entrées sorties
  - vidéo (VGA : million de couleurs, pixels codés sur 24 bits)
  - haute qualité audio (48 Khz), effets audio puissant (réverbération, atténuation du bruit, écho, etc), précision du contrôle
  - 8 voies simultanées d'enregistrement et d'écoute avec différents formats et différentes fréquences d'échantillonnages
  - interface MIDI (Musical Instrument Digital Interface)
  - Fax, Modem : supporte beaucoup de formats et de vitesse de transmission (jusqu'à 33600 bits par seconde)
  - téléphonie, vidéo phone : estimation de mouvements

Panorama de processeurs

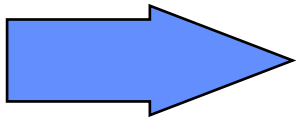
- MPEG temps réel : 30 images par seconde, codage 18 bits par pixel 352*240 (décodage MPEG 1 et 2, encodage MPEG 1) supporte windows 95
- faible tension (3,3 volts)
- accélération graphique :
  - dessins en fil de fer
  - génération de polygone
  - tracer de lignes
- instructions :
  - papillon FFT
  - mult, Add, Mad : algorithme de Booth, arbre de Wallace
- opérateurs :
  - 4 UALs
  - un bloc d'estimation de mouvements (pour le MPEG) :
    - 400 éléments arithmétique (pointe à 20 BOPS)

Panorama de processeurs

- Le Trimédia :
  - TM 1000 Philips
  - 4 milliard d'opérations par seconde (4 BOPS)
  - architecture VLIW :
    - 100 Mhz
    - jusqu'à 5 instructions par cycle
    - adressage sur 32 bits
    - 128 registres généraux sur 32 bits
    - 27 unités fonctionnelles
  - caches :
    - 32 Ko pour les instructions
    - 16 Ko pour les données
  - périphériques :
    - audio
    - vidéo
    - graphique

Panorama de processeurs

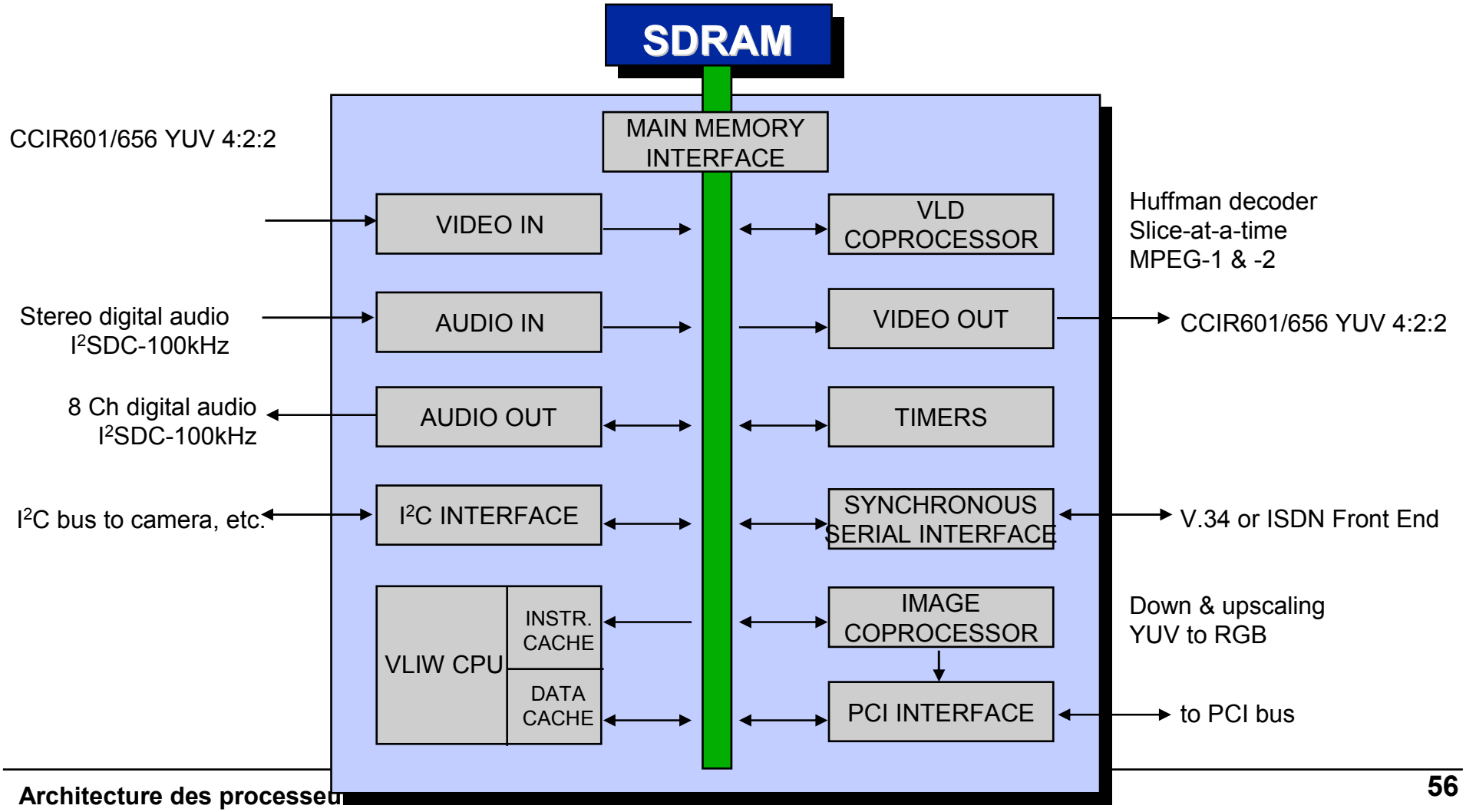
- Cœur du processeur : CPU temps réel (RTOS : real time operating system)
- développement en C ANSI ou en C++
- 8 canaux de sortie audio (100 Khz)
- 1 canal d'entrée audio 100 Khz
- 1 cana vidéo
- Coprocesseur MPEG 1 et 2
- supporte 37 opérations :
  - multimédia
  - DSP



  - estimation du mouvement
  - valeur absolue
  - addition
  - multiplication
  - moyenne
  - etc
- gestion du parallélisme :
  - lors de la compilation : détection du parallélisme durant la compilation, ordonnancement des opérations durant la compilation
  - simplifie énormément la logique de gestion par rapport à un processeur qui évalue le parallélisme durant l'exécution

Panorama de processeurs

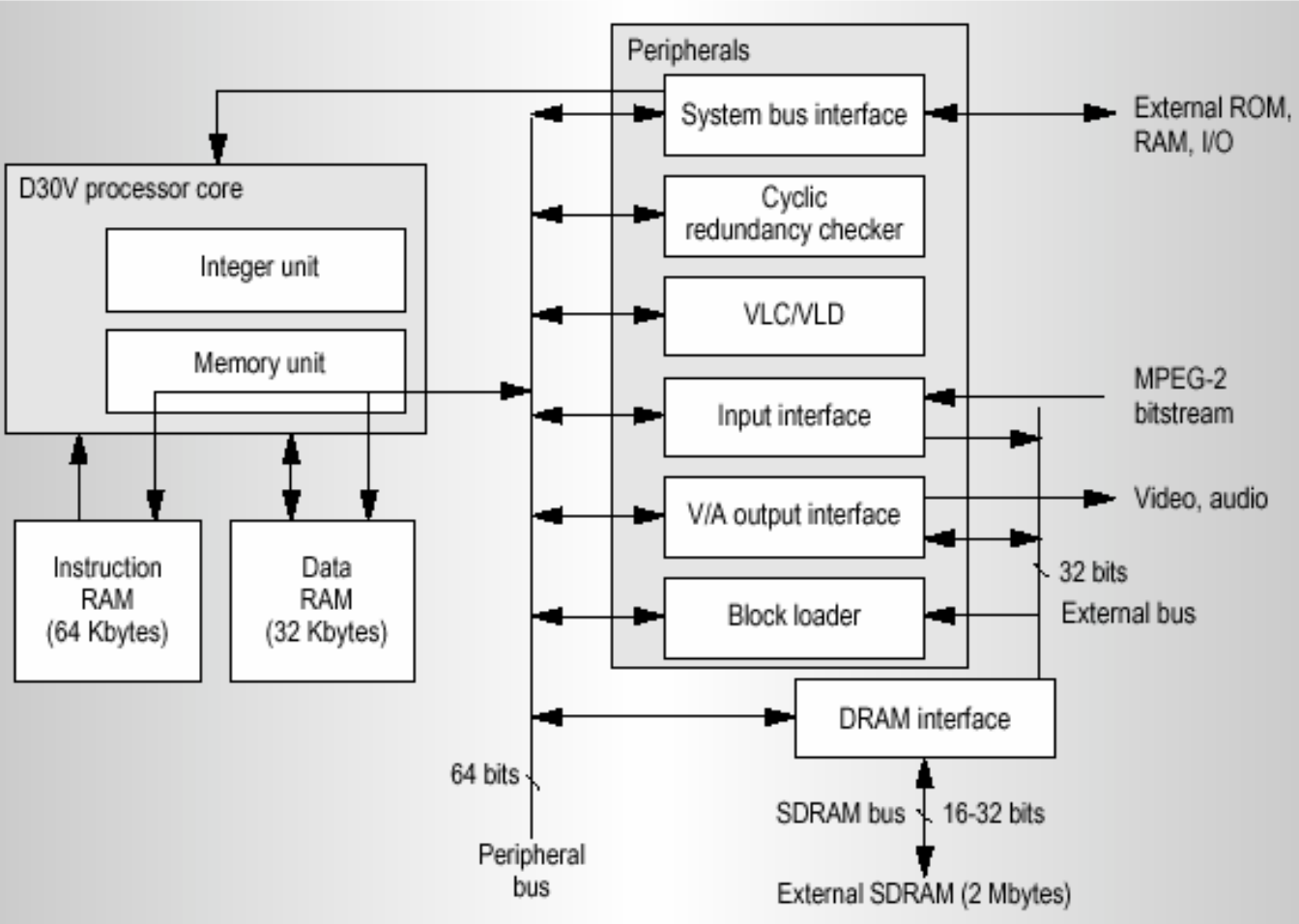
♦ Architecture du processeur Trimedia



Panorama de processeurs

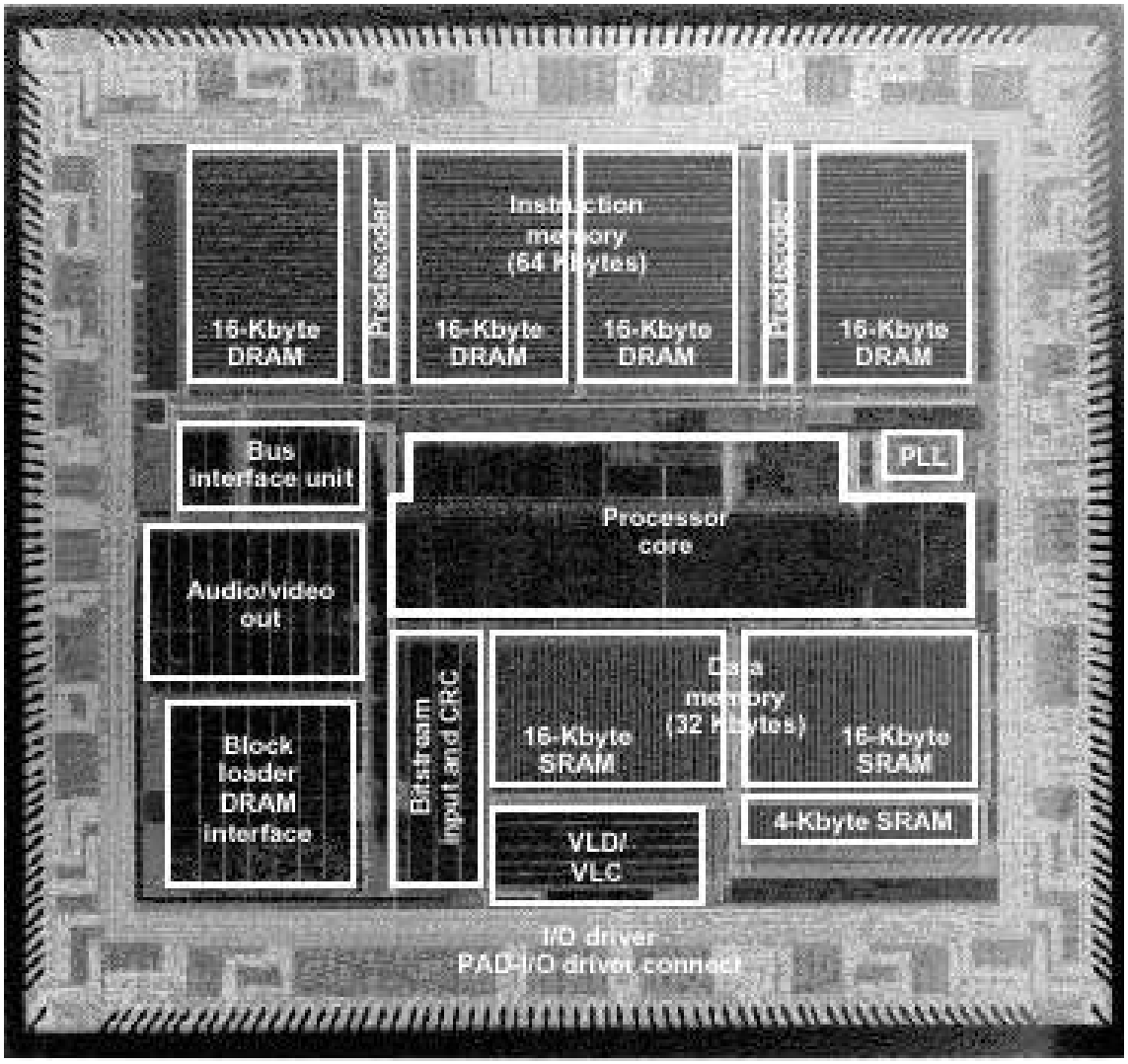
Le D30V/MPEG

➢ Schéma bloc du processeur



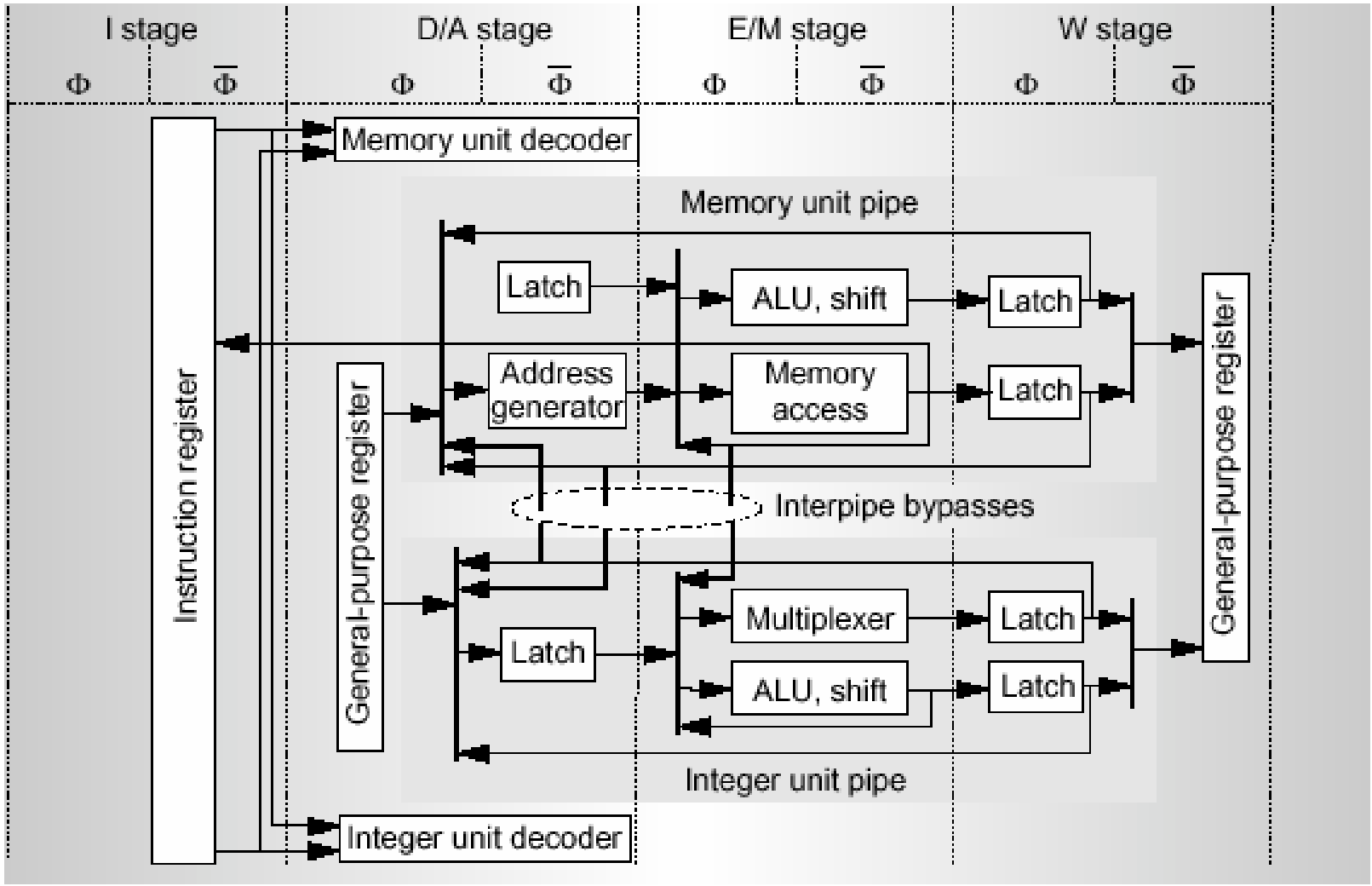
Panorama de processeurs

❑ Le layout du D30V/MPEG



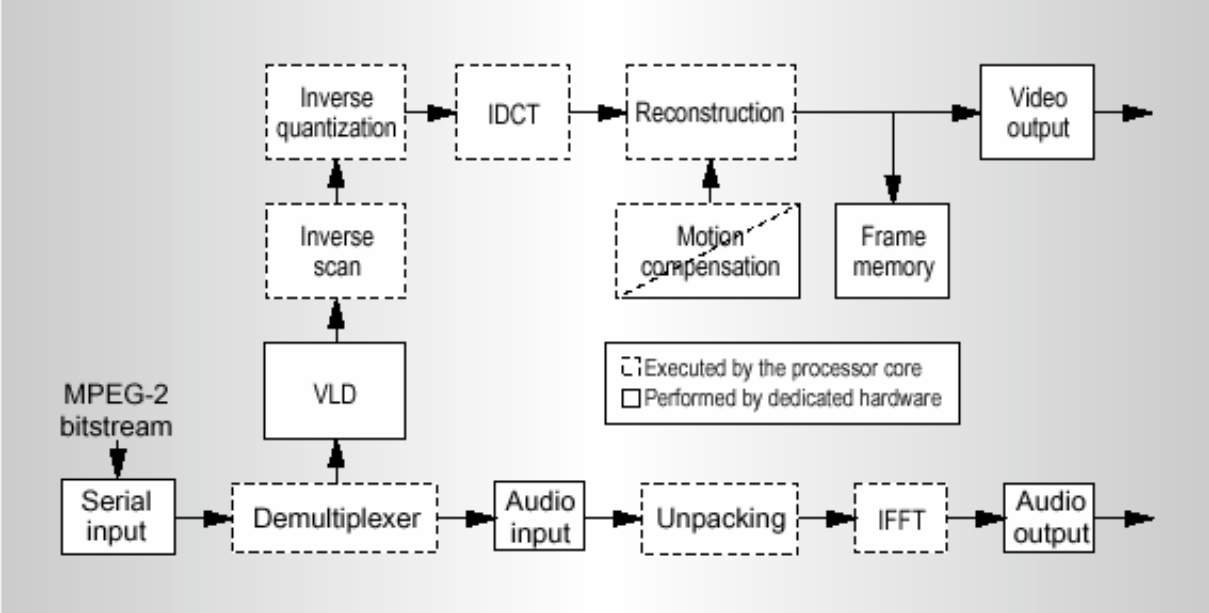
Panorama de processeurs

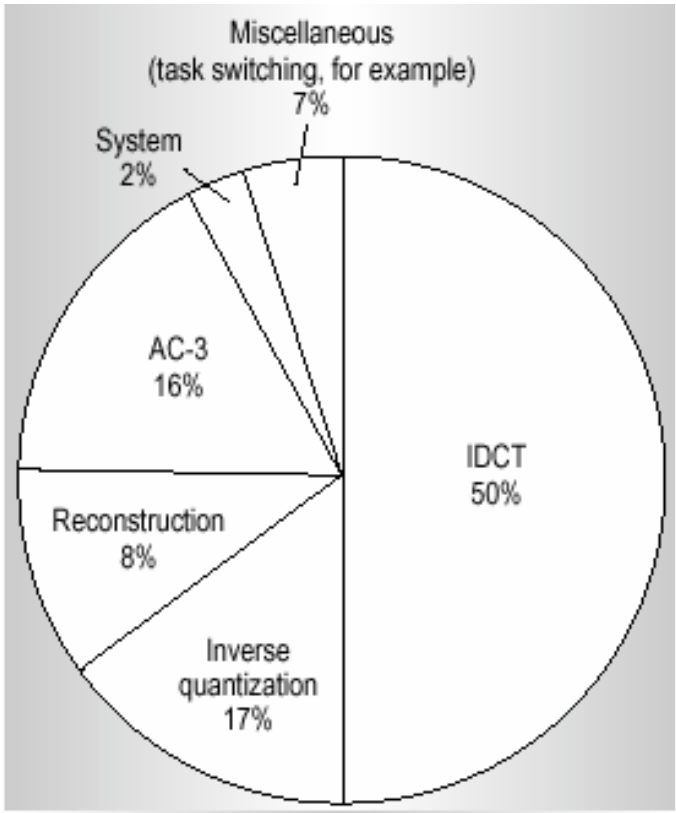
❑ Le pipeline D30V/MPEG



Panorama de processeurs

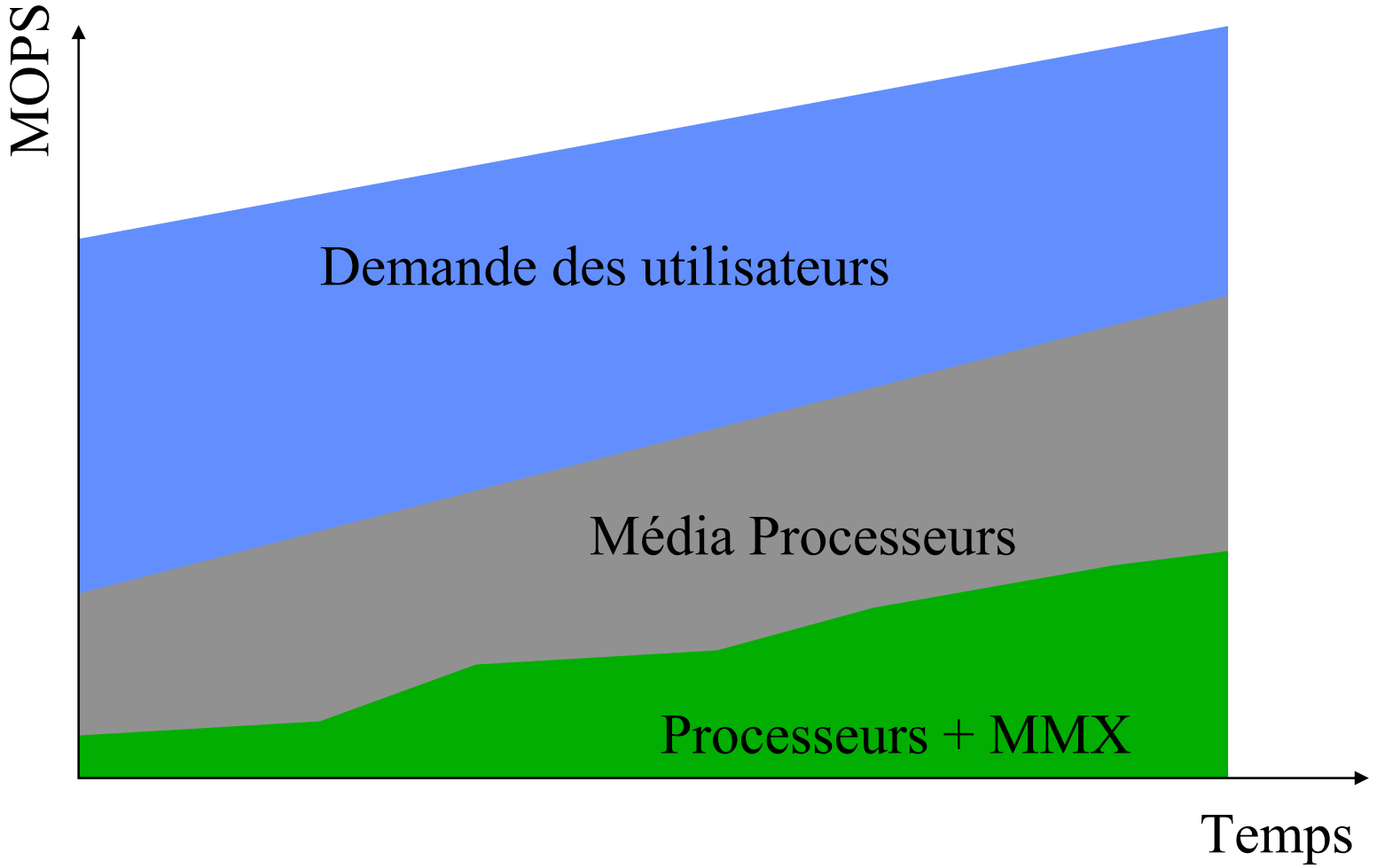
➢ Mpeg : schéma bloc et répartition en terme de puissance de calcul





Panorama de processeurs

MMX ou Média Processeurs



Panorama de processeurs

❑ Cœurs de processeurs :

➢La société Zoran développe des cœurs de *processeurs* de traitement du signal (audio et vidéo) :

- ♦ circuits disponibles sous forme logicielle (Vérilog ou VHDL)
- ♦ indépendant de la technologie
- ♦ fournis sous forme de modèles synthétisables
- ♦ interface générique permettant de les associer à d'autres fonctions
- ♦ prédiction de Zoran pour l'horizon 2001 : 20 % des circuits spécifiques seront développés à partir de composants virtuels
- ♦ la difficulté actuelle est liée aux problèmes de propriétés industrielles de ces circuits (*pouvoir les diffuser sans pour autant se les faire pirater*)

Panorama de processeurs

- Xilinx Alliance core :
  - propose un cœur de processeur "LavaCore" :
    - configurable Java Processor Core
    - exécution directe (hardware) du byte code Java
    - cœur implémentable dans les circuits de la famille :
      - Virtex-II
      - Virtex-E

Panorama de processeurs

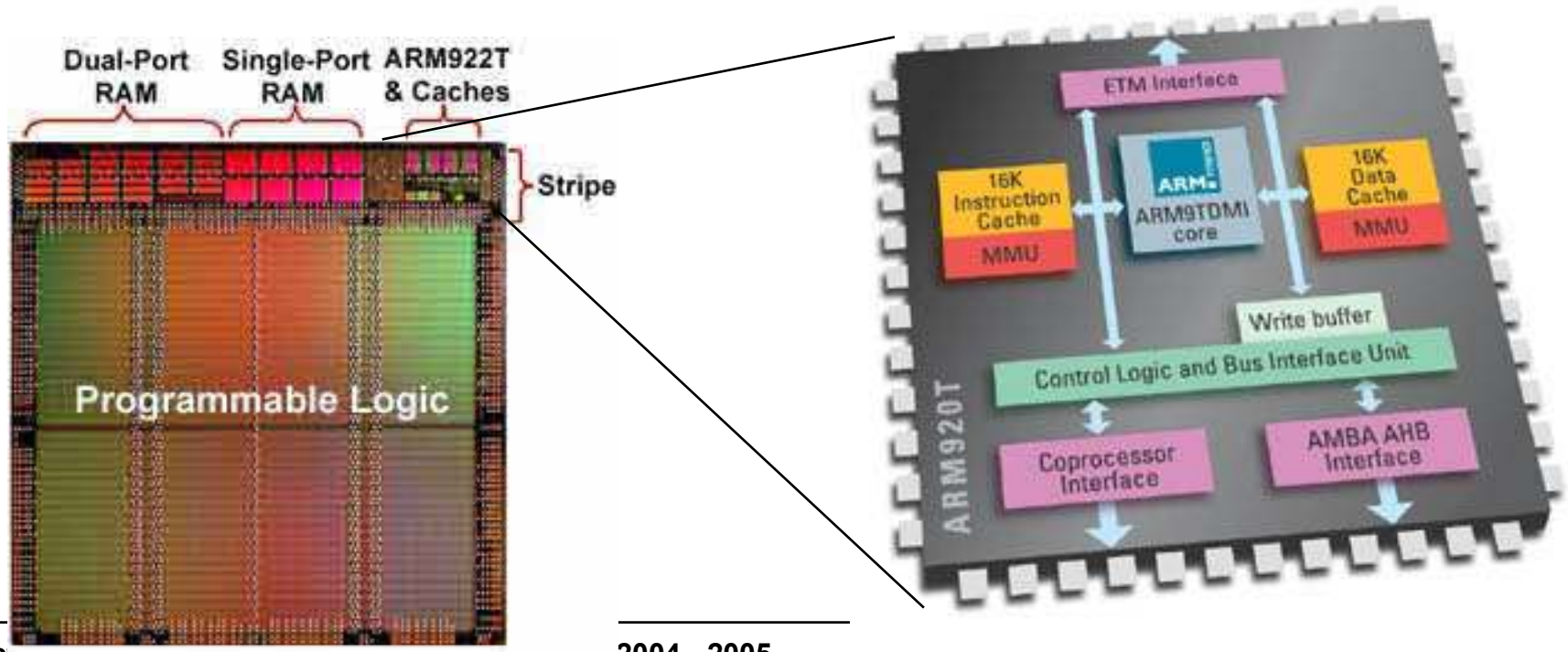
➢ Altera :

♦ propose des cœurs de processeur :

– Nios :

• processeur RISC

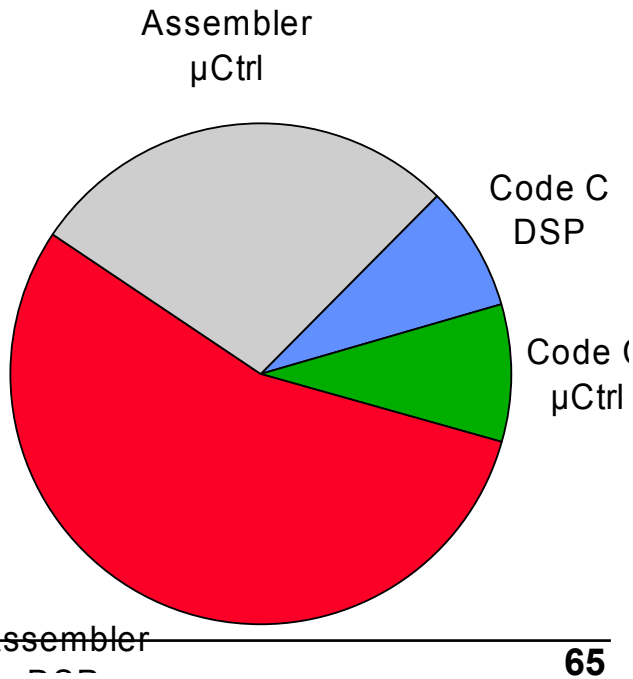
– ARM : Excalibur EPXA10 Device



Panorama de processeurs

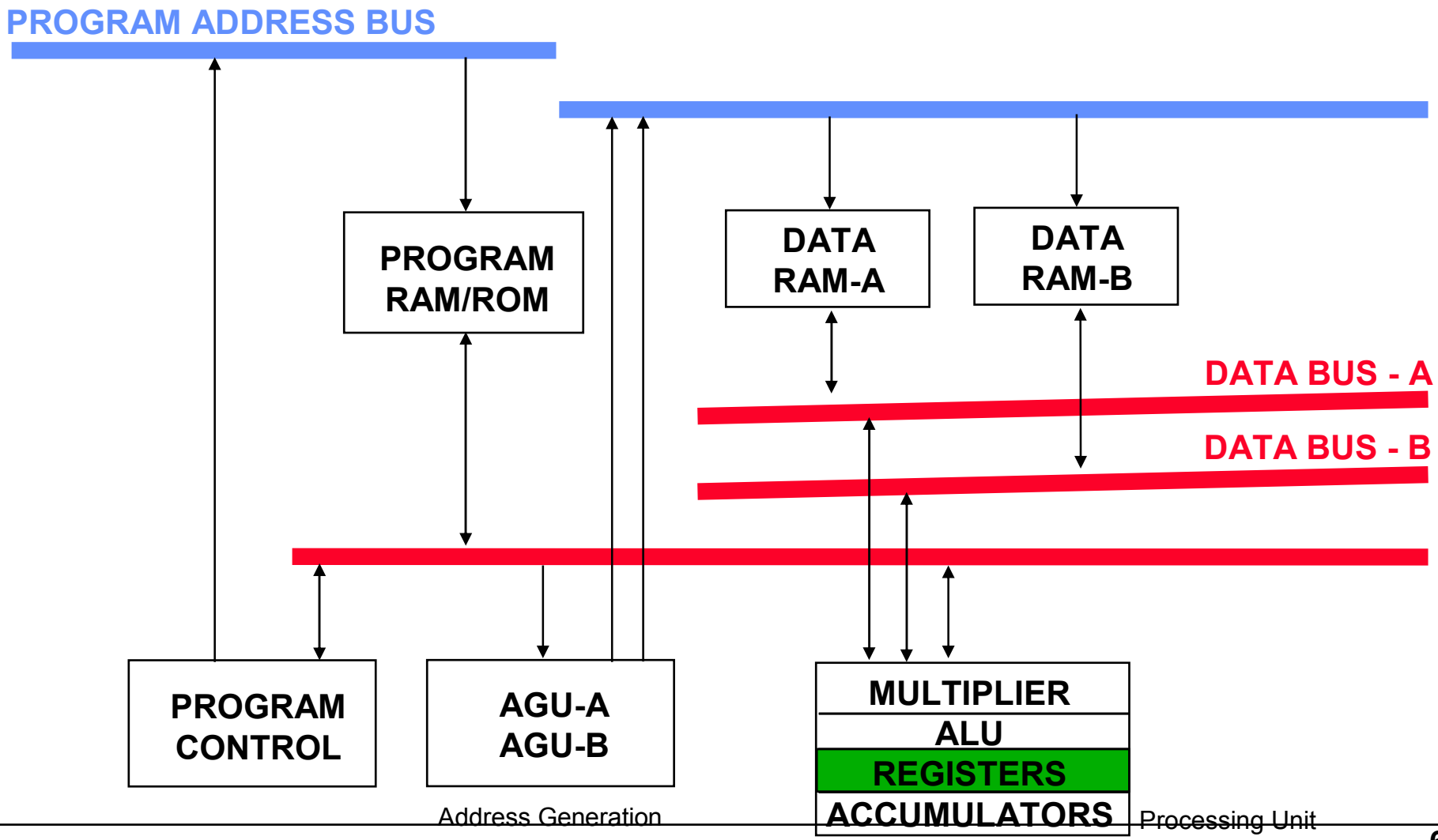
❑ Les processeurs de traitement du signal

- ➢ Ce sont les premiers processeurs a avoir adopté en masse l'architecture Harvard :
  - ♦ bus de données et bus d'adresses distincts
- ➢ Contrôleur RISC
- ➢ Ils possèdent des opérateurs spécifiques très performants pour les applications de traitement du signal :
  - ♦ multiplication - addition
  - ♦ manipulation de bits (très utile pour la FFT)
- ➢ Mise en place de plusieurs bancs mémoire parallèles : accès simultané à plusieurs données
- ➢ Traitements en 32 bits flottants ou 16 bits virgule fixe
- ➢ Traitements parallèles
- ➢ Nécessité de faible consommation :
  - ♦ applications portables : téléphonie, systèmes embarqués



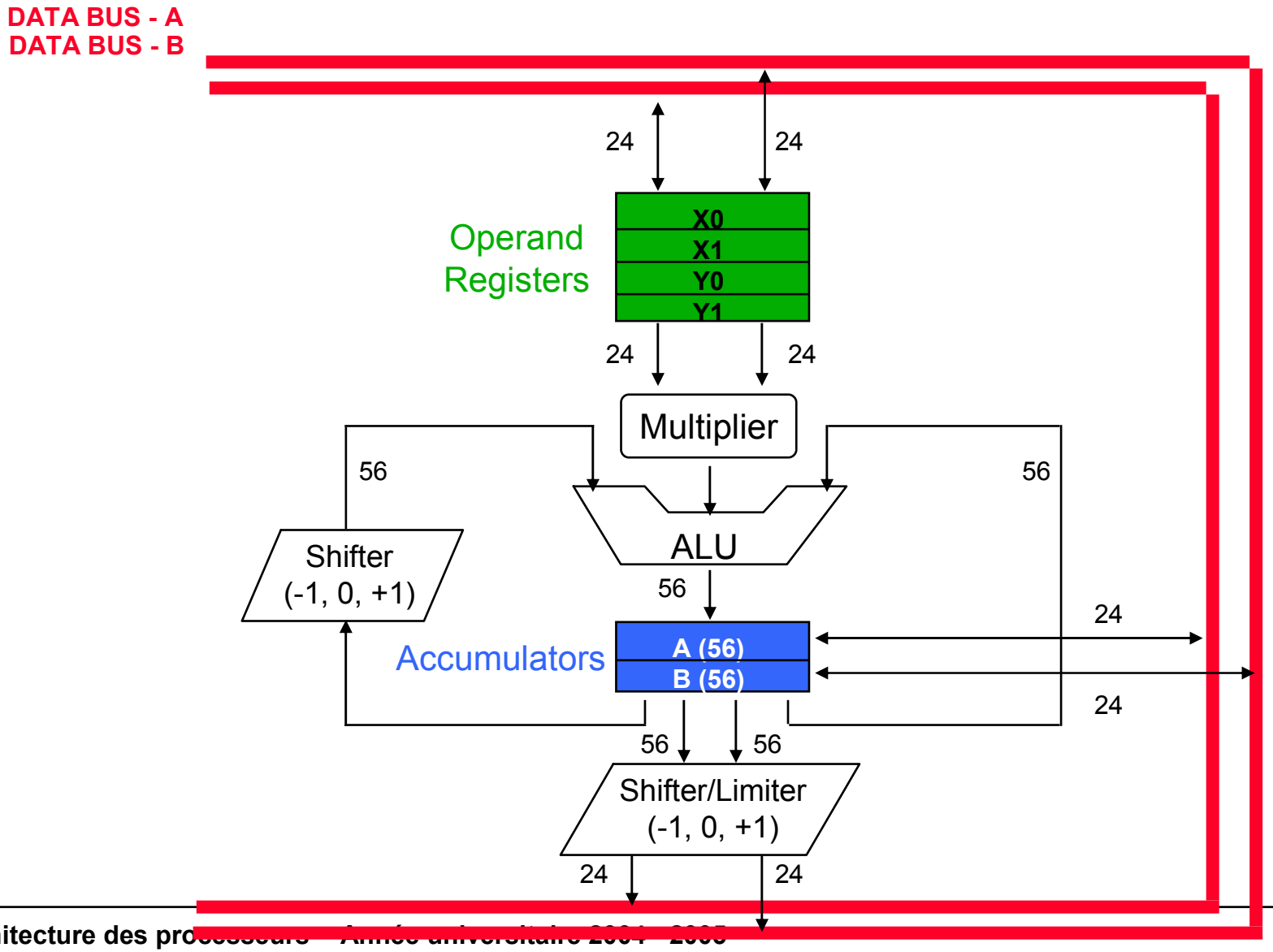
Panorama de processeurs

♦ Structure générale d'un DSP



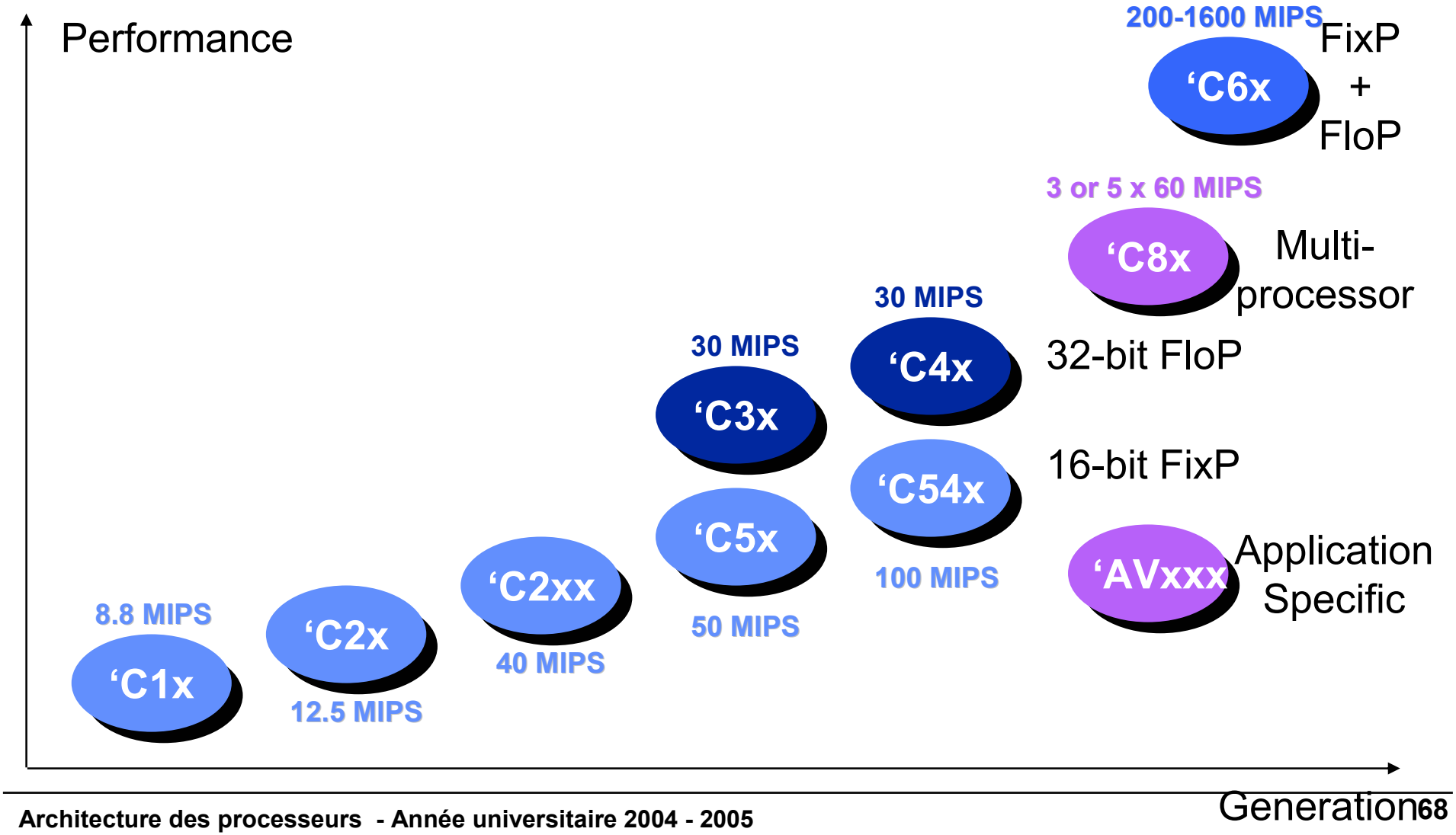
Panorama de processeurs

♦ Structure générale d'une unité de calcul de DSP



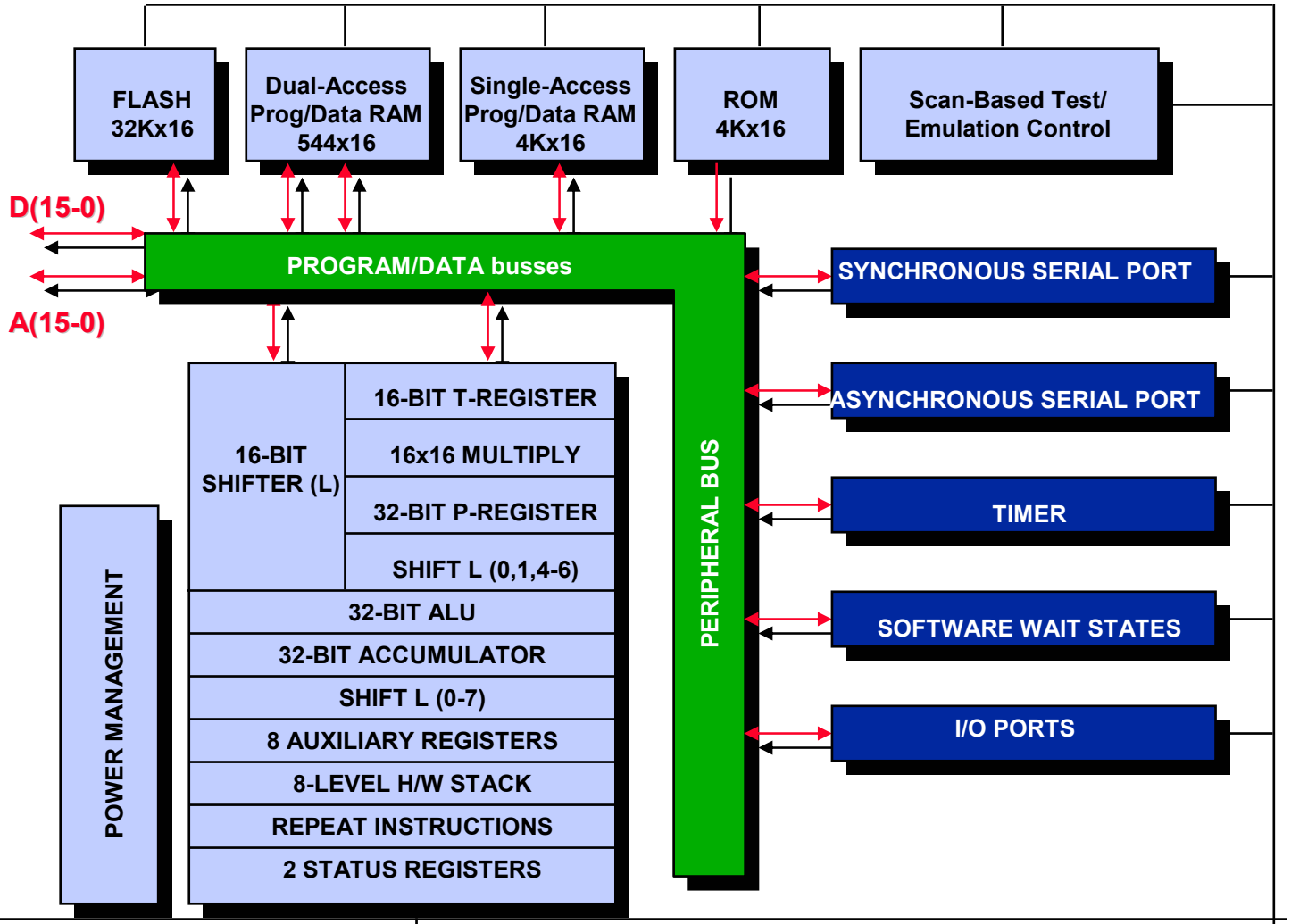
Panorama de processeurs

❑ La famille des processeurs TMS 320 C XX

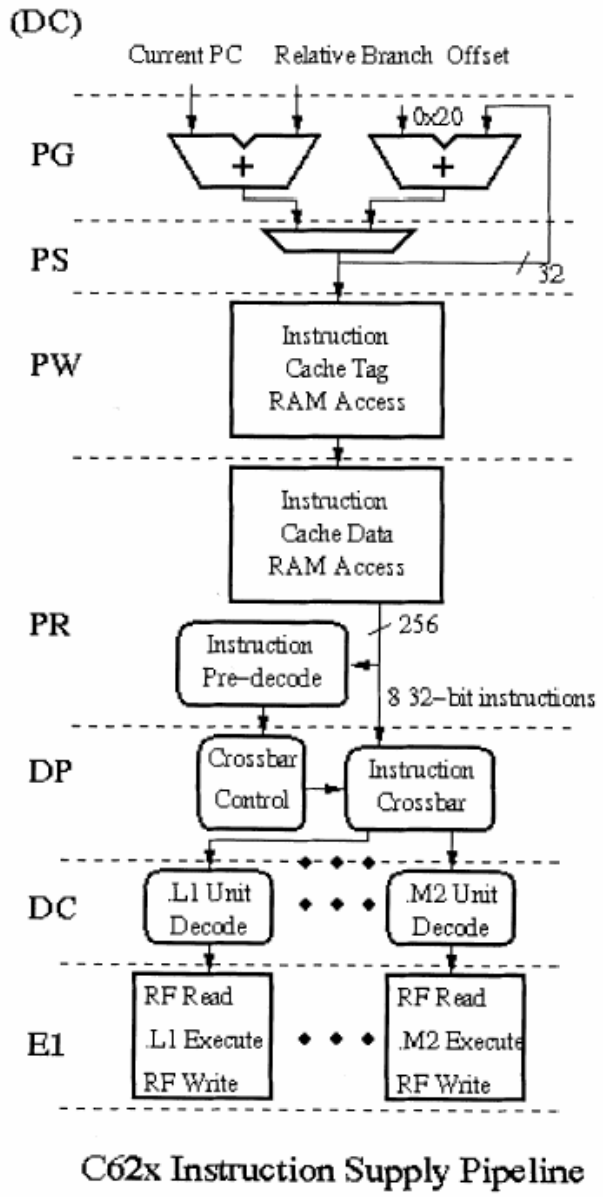


Panorama de processeurs

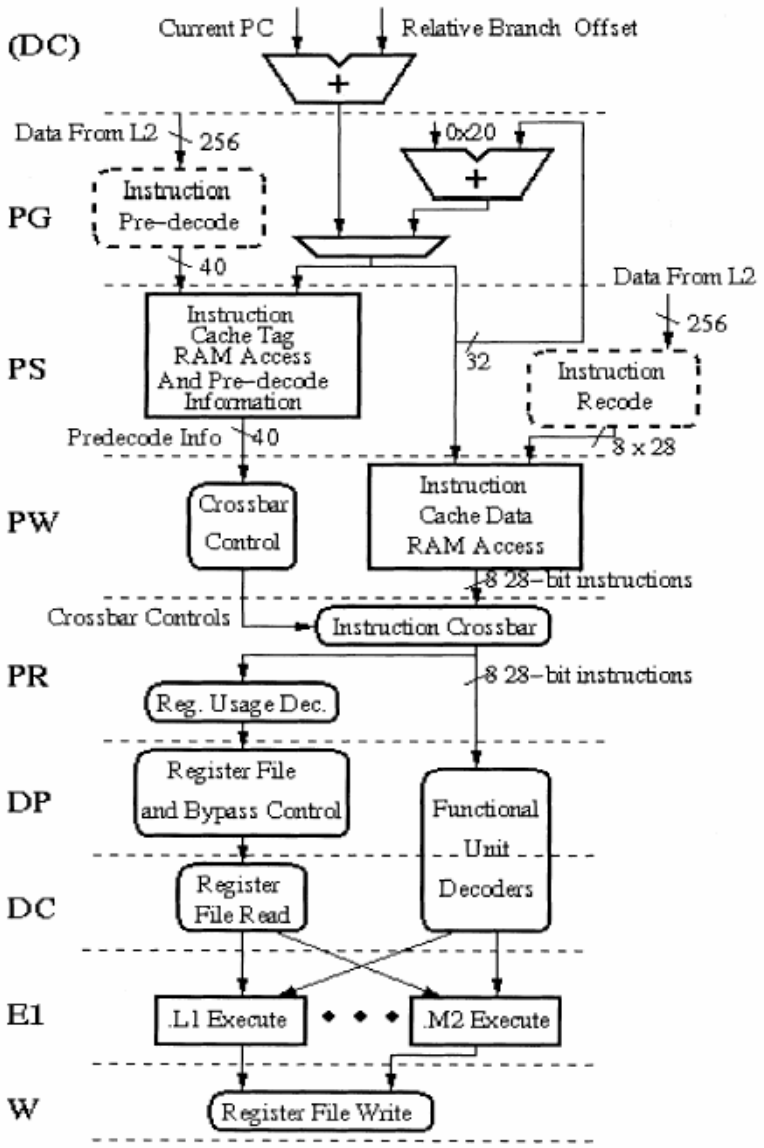
♦ Architecture des processeurs TMS 320 C XX



Panorama de processeurs



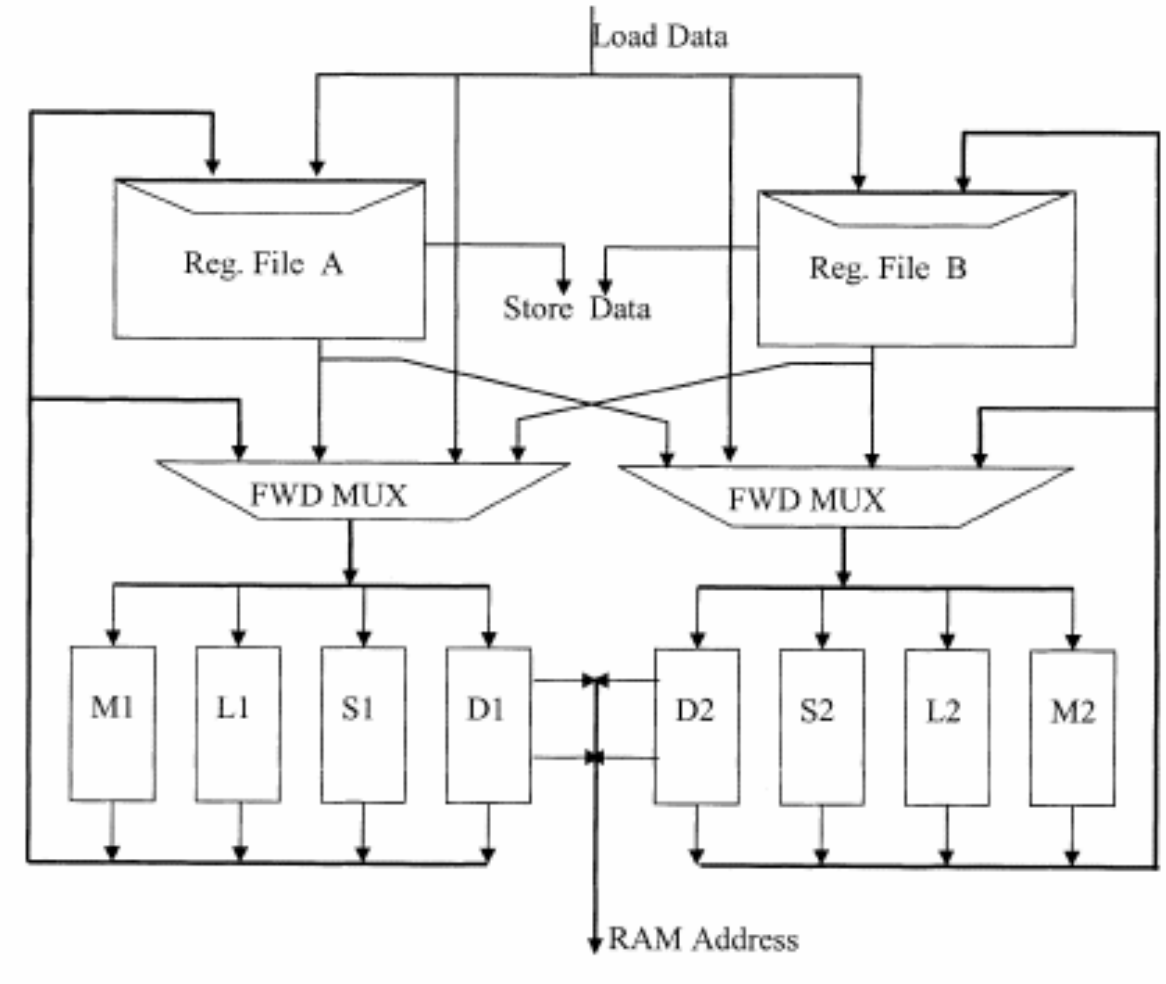
C62x Instruction Supply Pipeline



C64x Instruction Supply Pipeline

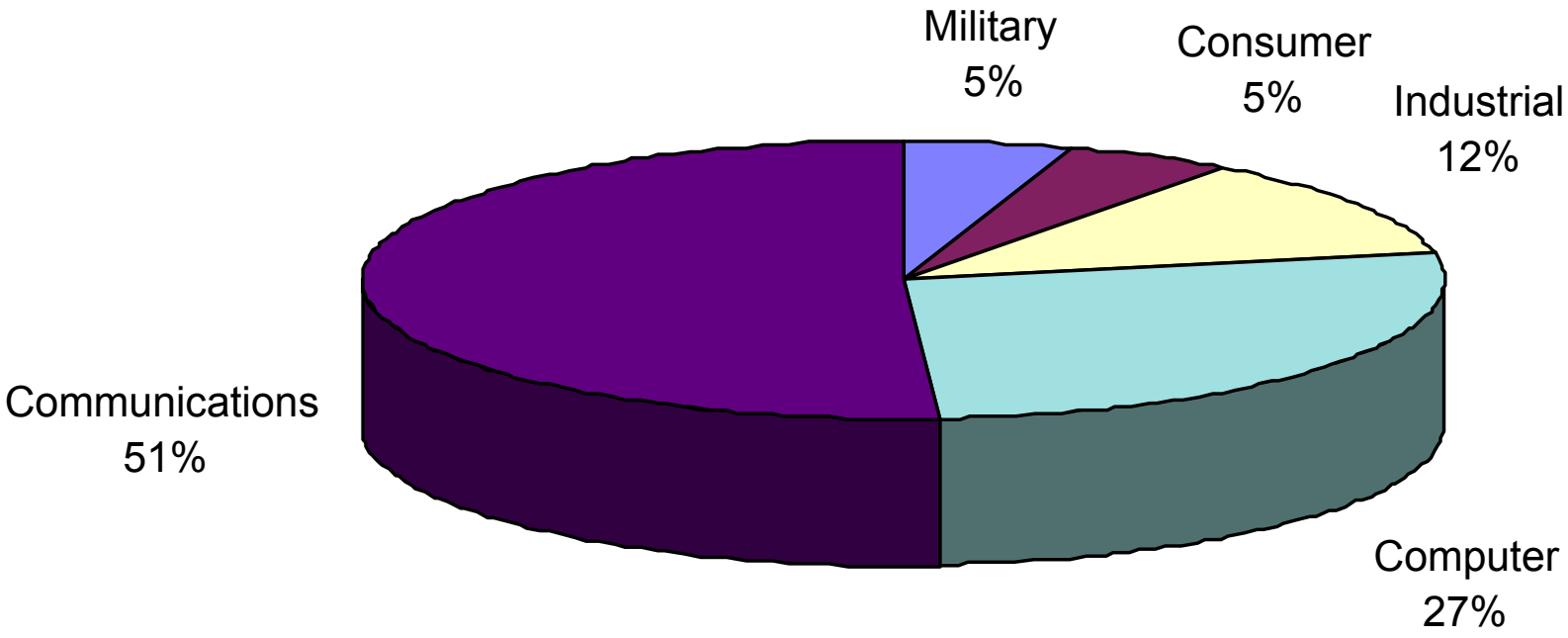
Panorama de processeurs

♦ Chemin de données du C64X



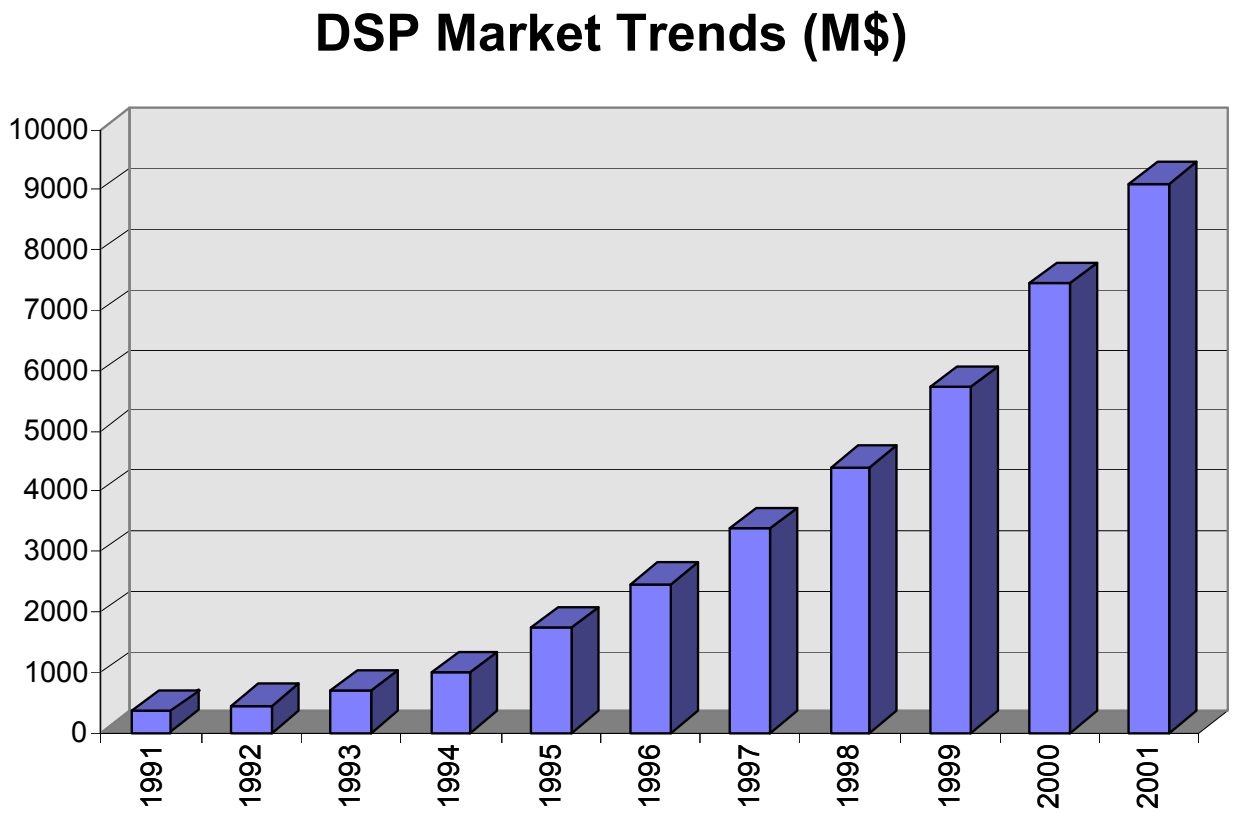
Panorama de processeurs

➢Utilisation des DSP (REE 10 / 97)



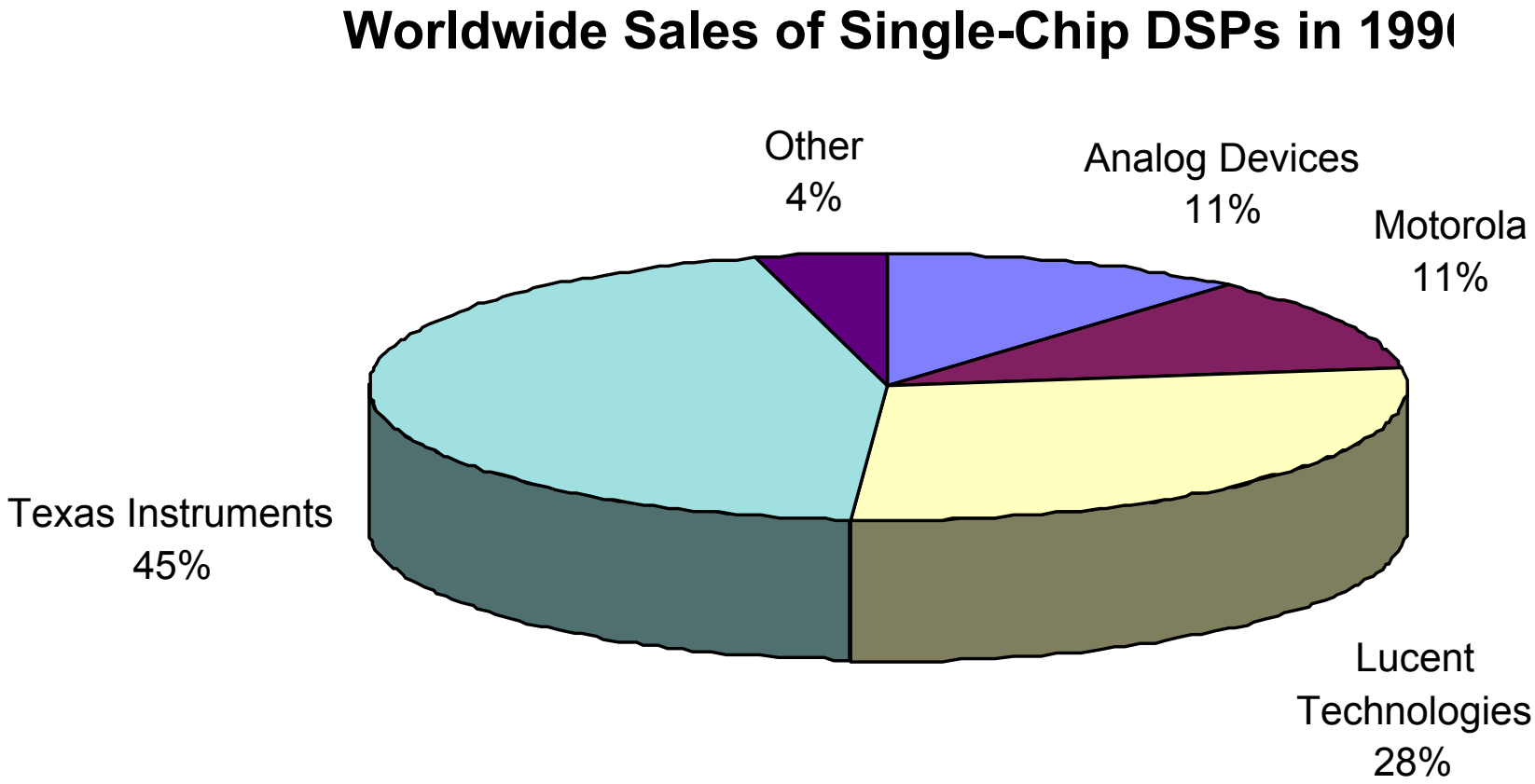
Panorama de processeurs

➢Evolution du marché des DSP



Panorama de processeurs

➢Les vendeurs de DSP



Marché de $395M