Fonctionnement et performance des processeurs

Eric Cariou

Université de Pau et des Pays de l'Adour
UFR Sciences Pau - Département Informatique

Eric.Cariou@univ-pau.fr

Plan

- Fonctionnement des processeurs
  - Unités de calcul et de commande
  - Registres
  - Modes d'adressage mémoire
  - Bus internes
  - Architecture X bits
- Optimisation des performances
  - Pipeline
  - Prédiction de branchement
  - Jeux d'instructions
  - Architectures superscalaire, multi-core
- Exemple avec Athlon 64
- Conclusion sur évolution des performances

Unités de calculs

- Unités réalisant des calculs : 3 types
  - Calculs logiques et arithmétiques sur les entiers : ALU
  - Calculs sur les flottants : FPU
  - Calculs multimédia ou vectoriels
- UAL (ALU : Arithmetic and Logic Unit)
  - La plus importante, utilisée par tous les programmes
  - Calculs simples sur les entiers
  - Calculs logiques (comparaison, OR, NOT, ...)
- FPU (Floating Point Unit)
  - Calculs sur des flottants
  - Fonctions mathématiques avancées : sqrt, sin ...

Unités de calcul

- Unité multimédia
  - Diffère selon le type et la marque : Intel MMX et SSE, AMD 3D Now ! ...
  - Fait principalement des calculs vectoriels
    - Exécution en parallèle d'une même instruction sur plusieurs données
- Un processeur peut intégrer plus d'une unité de chaque type
  - Exemple : AMD Athlon 64
    - 3 ALU et 3 FPU : calculs en parallèle possibles

Unité de commande

- Unité qui coordonne le fonctionnement des autres éléments
  - Dans le but d'exécuter une séquence d'instructions (le programme)
- Pour exécuter une instruction, 2 cycles se succèdent
  - Cycle de recherche de l'instruction
    - Lecture en mémoire de l'instruction à exécuter puis décodage de son contenu
  - Cycle d'exécution de l'instruction
    - Lecture en mémoire ou des registres pour envoyer les opérandes à l'unité de calcul ou d'accés en mémoire
    - Exécution du calcul ou de l'accès mémoire et enregistrement du potentiel résultat en mémoire ou dans un registre

Unité de commande

- Constituée de plusieurs éléments
  - Compteur Ordinal (CO) : registre contenant l'adresse du mot mémoire stockant le code de la prochaine instruction
  - Registre d'Instruction (RI) : reçoit le code de la prochaine instruction à exécuter
  - Le décodeur : à partir du code de l'instruction, détermine l'opération à exécuter
  - L'horloge : pour synchroniser les éléments
  - Le séquenceur : coordonne le tout
- Utilise également pour l'accès en mémoire
  - Registre d'Adresse (RA) : registre contenant l'adresse du mot à accéder en mémoire
  - Registre Mémoire (RM) : registre contenant le mot lu ou à écrire en mémoire

Unité de commande : horloge

- Horloge
  - Définit le cycle de base : cycle machine (clock cycle)
  - Utilisée pour synchroniser chaque étape des cycles de recherche et d'exécution
- Temps exécution d'un cycle de recherche ou d'exécution
  - Prend un certain nombre de cycles de base
- Cycle CPU = temps d'exécution minimal d'une instruction (recherche + exécution)

Unité de commande : séquenceur

- Séquenceur
  - Automate générant les signaux de commande contrôlant les différentes unités
- 2 façons de réaliser cette automate
  - Séquenceur câblé
  - Séquenceur micro-programmé (firmware)
- Avantages
  - Câblé : un peu plus rapide
  - Micro-programmé : plus souple et plus simple à réaliser

Unité de commande : séquenceur

- Séquenceur cablé
  - Circuit séquentiel réalisé « en dur » avec des portes de base
  - Pour chaque instruction exécutable
    - Un sous-circuit réalisant le contrôle des éléments pour réaliser l'opération
  - Un seul des sous-circuits est activé selon le code renvoyé par le décodeur
- Séquenceur micro-programmé
  - Une ROM contient des micro-programmes constitués de micro-instructions
  - Le séquenceur sait exécuter en séquence des micro-instructions
  - Nouvelle version du séquenceur = nouvelle ROM seulement, pas besoin de reconcevoir entièrement le circuit

Registres

- Registre = mots mémoire internes au processeur
- Les registres de fonctionnement
  - Compteur Ordinal (CO), Registre Instruction (RI), ...
  - Accumulateur
    - Registres internes à l'UAL
    - Stockent les opérandes et le résultat d'un calcul
- Registres généraux
  - Registres accessibles par le programme
  - Servent à stocker
    - Des valeurs souvent utilisées
    - Des résultats intermédiaires
  - Sans avoir besoin de repasser par la mémoire
    - Gain de performance et de temps

Registres

- Registres d'indice ou d'index (XR : indeX Registers)
  - Utilisés pour parcourir plus rapidement des tableaux
  - Jeu d'instruction particulier pour incrémenter et décrémenter leurs valeurs
  - Possibilité d'incrémentation ou décrémentation automatique de ces registres après leur utilisation
- Registres de pile (SP : Stack Pointer)
  - Pour simuler une pile dans la mémoire centrale

Registres

- Registres d'état (PSW : Program Status Word)
  - Ensemble de bits représentant chacun un état particulier (drapeau ou flag)
    - C : dépassement de capacité après un calcul de l'UAL
    - Z : résultat de l'opération est égal à 0
    - ...
  - Jeu d'instruction pour tester ces drapeaux
    - Exemple : fait une comparaison entre deux nombres puis fait un saut si Z = 0 (si les 2 nombres sont identifiques)
    - Permet de faire des sauts dans la séquence d'instructions et d'implémenter des boucles ou des sauts conditionnels (if ... then ... else)

Modèles d'accès mémoire

- Le processeur exécute des opérations
  - Avec des opérandes comme paramètres
- Plusieurs combinaisons possibles
  - Exemple sur une opération d'addition : A = B + C
  - A, B et C étant des valeurs se trouvant en mémoire centrale
    - Peut-on y accéder directement ?
    - Doit-on les placer avant dans des registres généraux ?
    - Doit-on les placer dans l'accumulateur de l'UAL ?
  - Plusieurs modèles généraux d'accès mémoire
    - Notation : (m,n) avec
      - m : nombre maximum d'opérandes par opération (addition ...)
      - n : nombre d'opérandes accessibles directement en mémoire centrale pour une opération de calcul

Modèles d'accès mémoire

- Modèle mémoire-mémoire (3,3)
  - `ADD @A, @B, @C`
  - 3 opérandes, toutes accessibles directement en mémoire
- Modèle mémoire-accumulateur (1,1)
  - ```
    LOAD @B
    ADD @C
    STORE @A
    ```
  - L'instruction LOAD place le contenu lu dans le registre accumulateur de l'UAL
  - L'accumulateur contient ce résultat après le calcul, le STORE le place à l'adresse mémoire précisée
  - Fonctionnement des premiers CPU 8bits
    ```

Modèles d'accès mémoire

- Modèle mémoire-registre (2,1)
  - ```
    LOAD R1, @B     ; R1 = contenu adr. B
    ADD R1, @C      ; R1 = R1 + contenu adr. C
    STORE R1, @A    ; place contenu R1 à adr. A
    ```
  - Variante (2,0) possible avec `ADD R1,R2`
  - Fonctionnement classique des processeurs 16 bits
- Modèle pile (0,0)
  - Utilisation d'une pile pour stocker les opérandes et les résultats
  - ```
    PUSH @B
    PUSH @C
    ADD
    POP @A
    ```

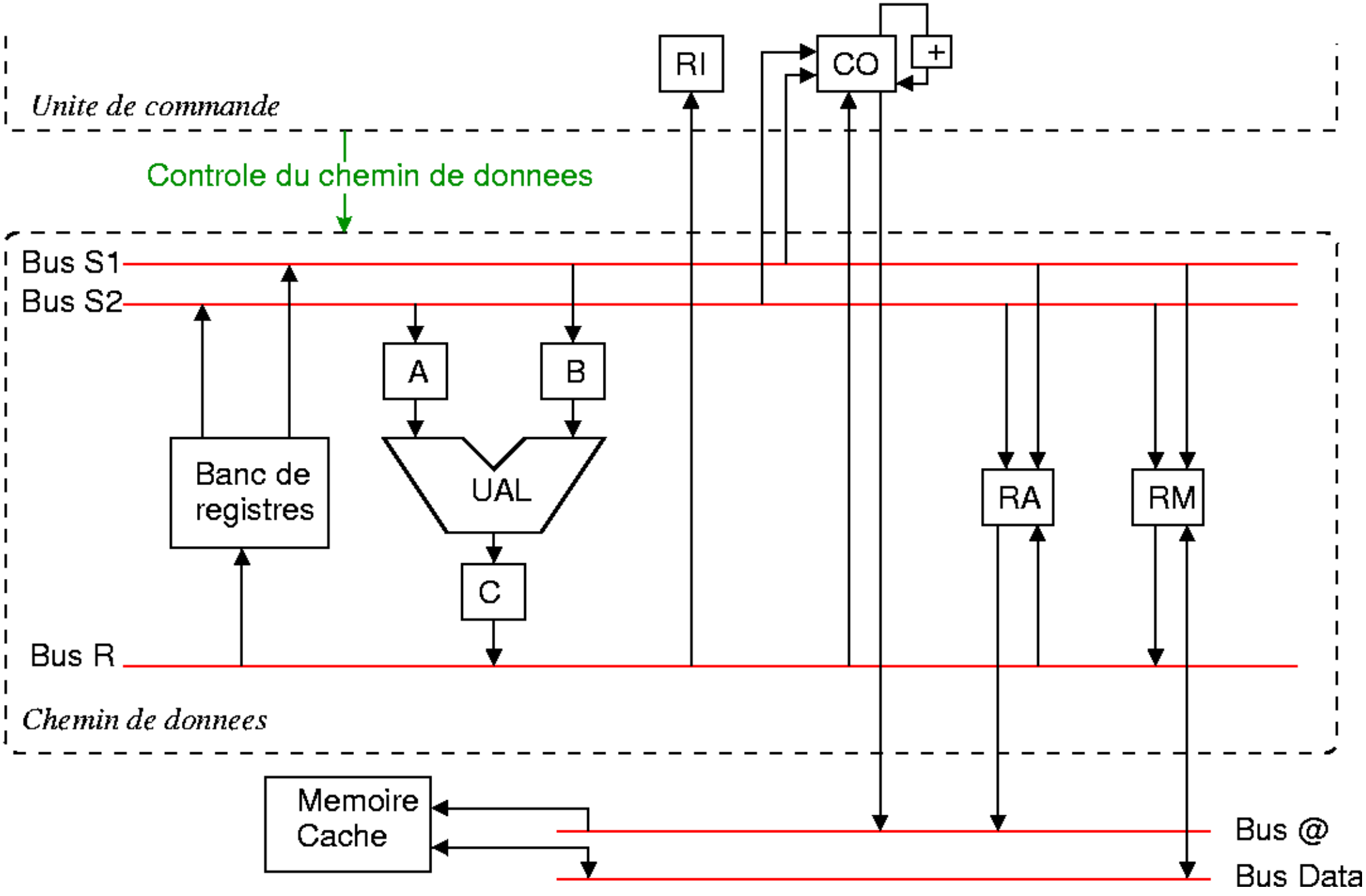
Modèles d'accès mémoire

- Modèle registre-registre (3,0)
  - Aucun accès direct en mémoire pour une opération de calcul
    - Tout passe par des registres
  - ```
    LOAD R1, @B
    LOAD R2, @C
    ADD R3, R1, R2
    STORE R3, @A
    ```
  - Architecture typique des processeurs RISC
  - Architecture dominante actuellement
  - Appelé aussi modèle chargement-rangement
    ```

Bus internes au CPU

- Au coeur du CPU, utilisation de bus
  - Pour envoi de données entre les éléments
    - Contrôleur mémoire, unités de calculs, registres ...
  - Pour envoi d'adresse
    - Lecture/écriture en mémoire ou dans un registre ...
  - Pour commander, coordonner les éléments
- Chemin de données
  - Unités gérant le traitement des données
    - Unités de calculs
    - Registres de tout type
  - Pour modèle accès mémoire (3,0) on aura 3 bus de données dans ce chemin

Bus internes au CPU



Bus internes au CPU

- Figure transparent précédent
  - Fonctionnement très simplifié et schématique d'un CPU de type chargement-rangement
- Les bus
  - S1 et S2 : entrées de l'UAL
  - R : résultat, sortie de l'UAL
  - @ : bus d'adresse pour accès en mémoire centrale (via le cache)
  - Data : bus de données à écrire ou lues dans mémoire centrale
  - Manque le bus de commande entre tous les éléments
- Autres éléments
  - + : incrémentation automatique du CO pour pointer sur la prochaine instruction
  - A, B et C : registres internes de l'UAL

Architecture X bits

- Processeurs sont souvent différenciés selon leur architecture 16, 32 ou 64 bits
- Historiquement : taille des registres (8, 16 bits...)
  - Mais dans processeurs récents : registres de toute taille (16, 32, 64, 80 ou 128 bits)
    - Selon que l'on manipule des entiers, des adresses, des flottants, des vecteurs ...
- Norme de fait de nos jours
  - Taille des registres généraux
    - Un processeur 64 bits a des registres généraux de 64 bits

Architecture X bits

- Conséquences
  - Unités de calcul entier doivent gérer des nombres de même taille que les registres généraux
  - Bus internes doivent avoir aussi cette même taille
- Registre général peut contenir une adresse mémoire
  - Définit alors aussi la taille maximale de mémoire adressable par le processeur
    - 32 bits : 2^{32} octets = 4 Go
      64 bits : 2^{64} octets = 18 Millions de To
    - En pratique : codage d'adresses sur moins de bits
      - AMD Athlon 64 : 48 bits = 256 To théorique mais 1 To en pratique

Optimisation et augmentation des performances des processeurs

Augmentation des performances

- Recherche permanente de l'augmentation des performances des CPU
  - Évolution des architectures
- Points principaux de cette évolution
  - Fréquence de fonctionnement
  - Mémoire cache
  - Parallélisation et optimisation des séquences d'instructions
    - Pipeline
    - Architectures superscalaires et multi-core
  - Jeu d'instructions
- Chaque point influence en bien ou en mal un autre
  - Recherche du meilleur compromis

Fréquence

- Temps d'exécution d'une instruction
  - Cycle CPU
- Idée : diminuer ce cycle
  - Augmentation de la fréquence de fonctionnement
- Avantages
  - Plus d'instructions exécutées en moins de temps
- Problèmes tchnologiques et physiques
  - Dégagement de chaleur
    - Nécessite un refroidissement du processeur mais qui a ses limites

Fréquence

- Problèmes (suite)
  - Temps de propagation
    - Les signaux électriques mettent un certain temps à traverser les éléments du processeur ou à circuler sur un conducteur
    - Certaines opérations effectuées par le processeur ne sont plus réalisables en un temps trop court
      - Limite la possibilité de monter en fréquence
- Solutions
  - Diminuer la finesse de gravure des transistors
    - Conducteurs plus courts, propagation plus rapide
    - Diminue le dégagement de chaleur
  - Chercher d'autres technologies de réalisation de transistors et des conducteurs

Fréquence

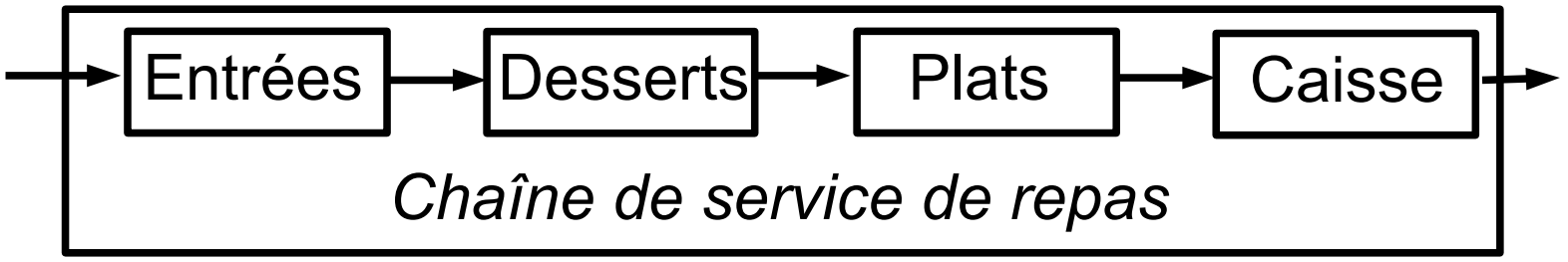
- Problèmes (suite)
  - Architecture, fonctionnement interne influence également la montée en fréquence
  - En général, pour une génération/architecture de processeur, on a une plage de fréquences utilisables
  - Exemple avec CPU Intel : passage du P3 au P4
    - Intel P3 : de 450 Mhz à 1.3 Ghz
    - Intel P4 : de 1.5 Ghz à 3.8 Ghz
      - Avec modification intermédiaire de l'architecture du P4
    - Changement important d'architecture pour monter en fréquence
  - Aujourd'hui, on atteint des limites à la montée en fréquence
    - Intel pensait arriver à 10 Ghz avec le P4 : pas réussi
    - Processeurs les plus rapides : pas plus que 4 à 5 Ghz

Mémoire cache

- Processeur a besoin d'un débit soutenu en lecture d'instructions et de données
  - Pour ne pas devoir attendre sans rien faire
- Utilisation de mémoire cache
  - Mémoire intermédiaire très rapide entre la mémoire centrale et le processeur
    - Avec algorithmes pour « deviner » et mettre dans le cache les données/instructions avant que le CPU en ait besoin
  - Recherche bon compromis entre tailles, types de caches, niveaux de cache, technique d'accès au cache ... pour optimiser les performances du cache
- *Voir le cours sur les mémoires*

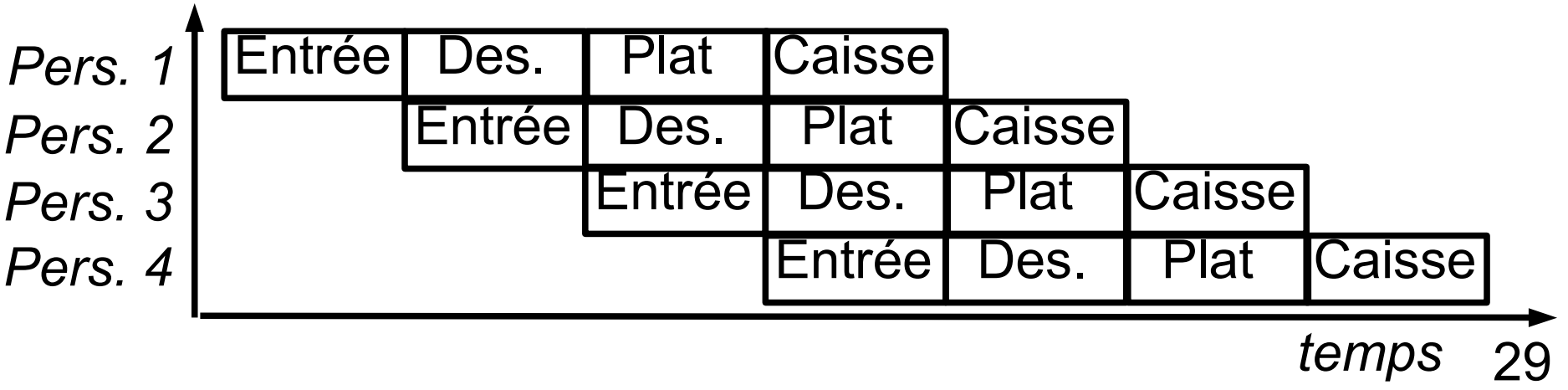
Pipeline

- Principe du pipeline par l'exemple
- Restaurant Universitaire
  - On passe, dans l'ordre devant 4 éléments
    - Un présentoir pour les entrées
    - Un présentoir pour les desserts
    - Un présentoir pour les plats de résistance
    - Une caisse



Pipeline

- 2 modes d'utilisation pour se servir un repas
  - Une seule personne à la fois dans toute la chaîne de service
    - Quand elle a passé toute la chaîne et est sortie, une autre personne entre se servir
  - Plusieurs personnes à la fois, en décalé
    - Une personne à chaque présentoir/élément
    - Une personne passe à l'élément suivant quand il est libre et qu'elle en a fini avec son élément courant



Pipeline

- Intérêts du deuxième mode
  - Plusieurs personnes se servent en même temps
  - Gain de temps : plus de personnes passent pendant une même durée
  - Meilleure gestion des éléments : toujours utilisés
- Inconvénients du deuxième mode
  - Plus difficile de faire « demi-tour » dans la chaîne d'éléments
  - Nécessite des synchronisations supplémentaires et des éléments dont le parcours prend un temps proche pour une bonne optimisation

Pipeline

- Dans un processeur, utilisation d'un pipeline pour exécution d'une opération
- Une opération comporte plusieurs sous-opérations
  - Pipeline pour exécution de ces sous-opérations
  - Une sous-opération utilise une sous-unité du processeur qui n'est pas utilisée par d'autres sous-opérations (si possible...)
- Exemple de pipeline simple (fictif mais proche des premiers pipelines développés)
  - LE : Lecture de l'instruction en mémoire
  - DE : Décodage de l'instruction
  - CH : Chargement des registres sources dans l'unité de calcul
  - EX : Exécution du calcul
  - ENR : Enregistrement du résultat dans le registre destination

Pipeline fictif – détail des étapes

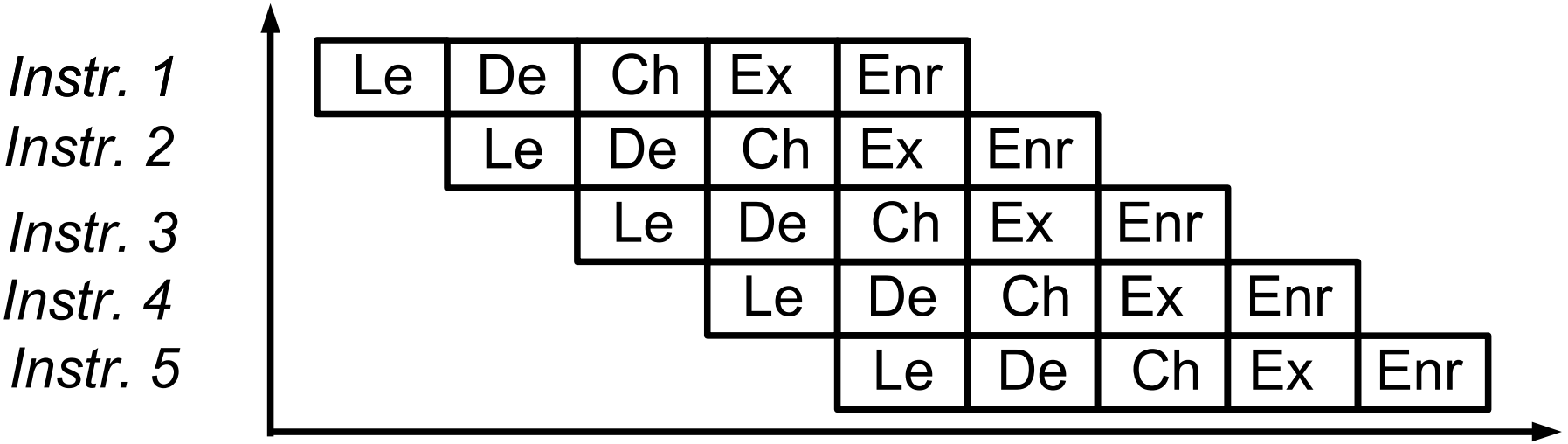
- En reprenant le chemin de donnée précédent, en modèle chargement-rangement (aucun accès direct en mémoire pour les opérations de calcul)
  - LE : lecture instruction
    - Accès en mémoire pour lire le mot mémoire placé à l'adresse contenue dans CO et correspondant à la prochaine instruction
    - Résultat placé dans RM
  - DE : décodage instruction
    - Copie de RM dans RI puis décodage de l'opération
  - CH : chargement des registres
    - Pour un calcul
      - Chargement des registres A et B à partir des registres du banc
    - Pour un accès en mémoire
      - Chargement de RA à partir d'un registre du banc
      - Dans le cas d'une écriture en mémoire, chargement en plus de RM à partir d'un registre du banc

Pipeline fictif – détail des étapes

- EX : exécution de l'instruction
  - Pour un calcul
    - UAL effectue le calcul
    - Résultat disponible dans C
  - Pour un accès en mémoire
    - Accès à la mémoire via les bus mémoire, RA et/ou RM
    - Si opération de lecture, RM contient le mot lu à l'adresse contenue dans RA
- ENR : enregistrement du résultat
  - Pour un calcul
    - Copie du registre C de l'UAL dans le registre destination du banc
  - Pour un accès en mémoire
    - Ecriture : rien à faire
    - Lecture : copie de RM dans le registre destination du banc

Pipeline

- Exécutions décalées de plusieurs instructions en même temps



  - Gain important en utilisant le pipeline
    - Sans : exécution séquentielle de 2 instructions en 10 cycles
    - Avec : exécution parallèle de 5 instructions en 9 cycles
    - Gain théorique car nombreux problèmes en pratique
  - Pour optimisation : temps de passage dans chaque étape identique (ou très proche)

Pipeline – profondeur

- En pratique actuellement : autour de 15 étages
- Exemples de profondeur de pipeline (nombre d'étages)
  - Processeurs Intel
    - i3, i5, i7 : 14
    - Core 2 Duo et Mobile : 14 et 12
    - P4 Prescott : 31
    - P4 (avant architecture Prescott) : 20
    - Intel P3 : 10
  - Processeurs AMD
    - K10 : 16
    - Athlon 64 : 12
    - AMD Athlon XP : 10
  - Processeurs de type RISC
    - Sun UltraSparc IV : 14
    - IBM Power PC 970 : 16

Pipeline – profondeur

- Intérêts d'avoir un pipeline plus profond
  - Plus d'instructions en cours d'exécution simultanée
  - Permet une montée en fréquence du processeur
- Limite de la montée en fréquence
  - Temps de propagation des signaux
    - A travers une unité et entre les unités du CPU
  - Plus d'unités plus petites = temps de propagation plus courts entre les unités
    - On peut raccourcir le temps de réalisation d'un cycle
    - Et donc augmenter la fréquence du processeur
- Mais un pipeline profond pose plus de problèmes qu'un pipeline court
  - Avec les aléas de contrôles principalement

Pipeline – aléas

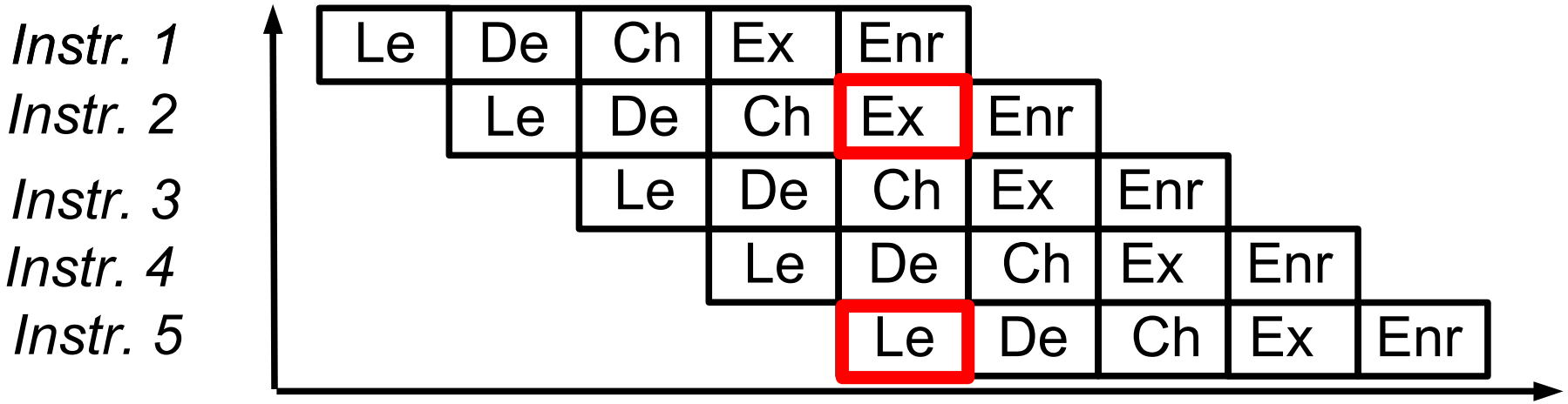
- Aléas
  - Problèmes rencontrés lors de l'exécution d'instructions par le pipeline
- 3 familles d'aléas
  - Aléas structurels
    - Des sous-unités du CPU doivent être utilisées simultanément par plusieurs étages du pipeline
  - Aléa de données
    - Une instruction de calcul en cours d'exécution dépend d'une valeur non encore calculée
  - Aléas de contrôle
    - L'instruction suivante dépend du résultat d'une instruction pas encore connu (test)

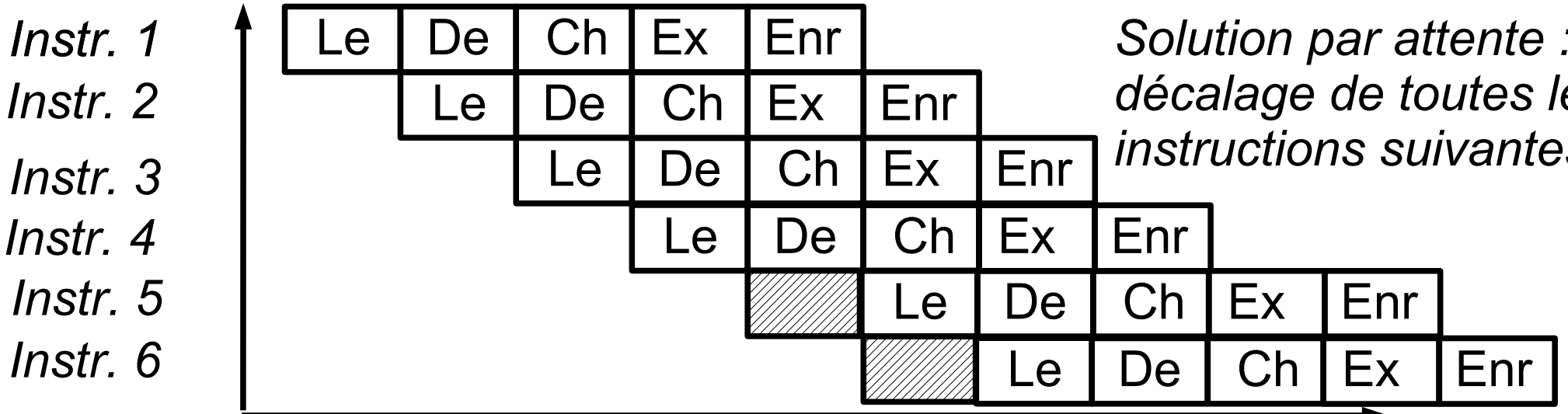
Pipeline – aléas structurels

- Exemple d'aléa structurel, pour notre pipeline simple
  - Accès à la mémoire dans les étapes
    - LE : lecture de l'instruction suivante en mémoire
    - EX dans le cas d'une opération de lecture/écriture en mémoire
  - Utilise une même sous-unité (accès mémoire) du processeur
- Solutions
  - Attendre pour une instruction que l'unité soit disponible
    - Peu efficace
  - Dupliquer dans le processeur ces sous-unités
    - Accès mémoire : intérêt de découper le cache L1 en deux parties
      - Partie « données » avec accès via RM et RA
      - Partie « instructions » avec accès via CO et RI
    - Peut alors faire un EX d'accès mémoire et un LE en même temps : 2 accès mémoires en parallèle sur les 2 parties différentes du cache L1
    - De plus le cache d'instructions est en lecture seule car les instructions ne sont pas modifiées en mémoire : plus simple et plus rapide car pas besoin de gérer la cohérence cache / mémoire centrale

Pipeline – aléas structurels

- Aléa structurel
  - *EX* de instr. 2 et *LE* de instr. 5 : accès à la mémoire

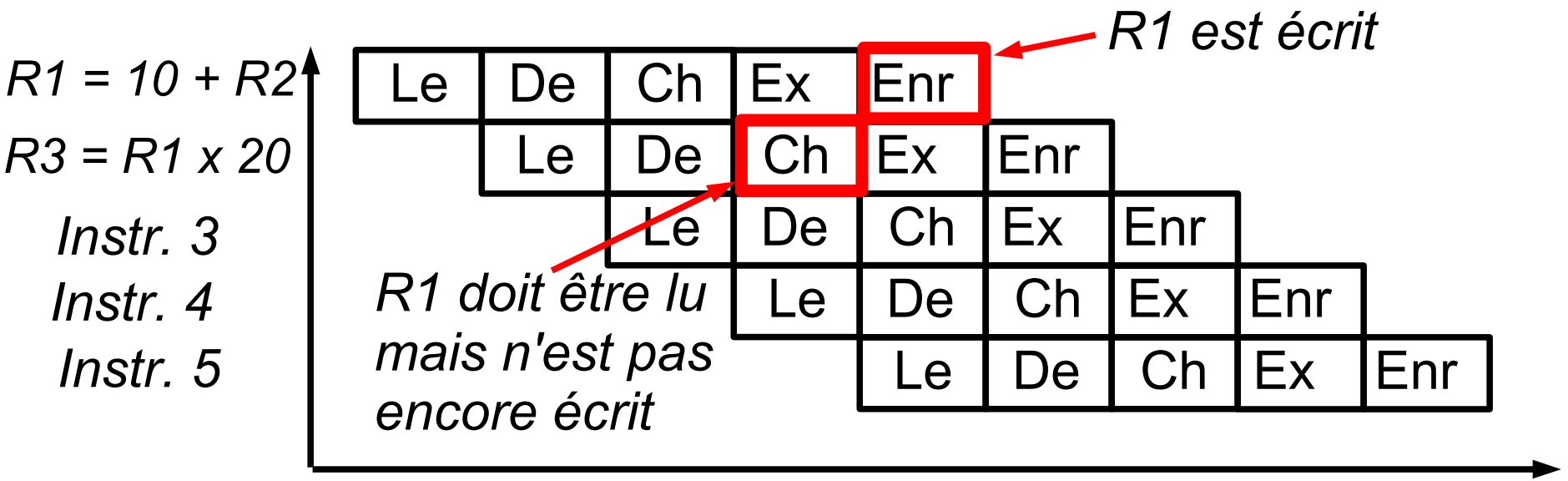




Solution par attente : décalage de toutes les instructions suivantes

Pipeline – aléas de données

- Aléa de données
  - `R1 = 10 + R2`
    `R3 = R1 x 20` (R1, R2 et R3 sont des registres)
  - Problème
    - Le calcul de R3 ne peut se faire que quand R1 est connu

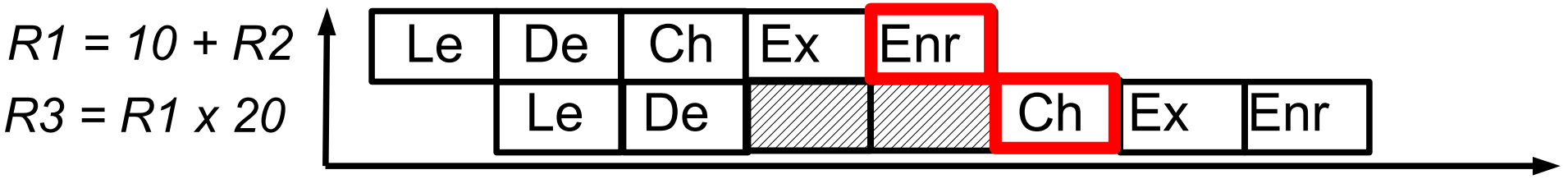


Pipeline – aléas de données

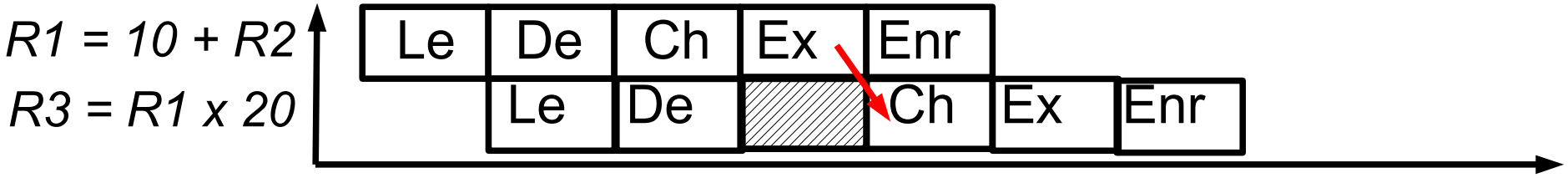
- Aléa de données : solutions
  - Arrêter l'exécution du calcul de R3 tant que R1 n'est pas connu : peu efficace
  - Changer l'ordre d'exécution des opérations pour éviter ou réduire le problème
  - Court-circuiter au plus tôt le pipeline quand la valeur de R1 est connue
    - Le résultat du dernier calcul est dans le registre C de l'UAL
    - On peut le réinjecter au cycle suivant dans le registre A ou B de l'UAL

Pipeline – aléas de données

- Aléa de données
  - Suspension du pipeline

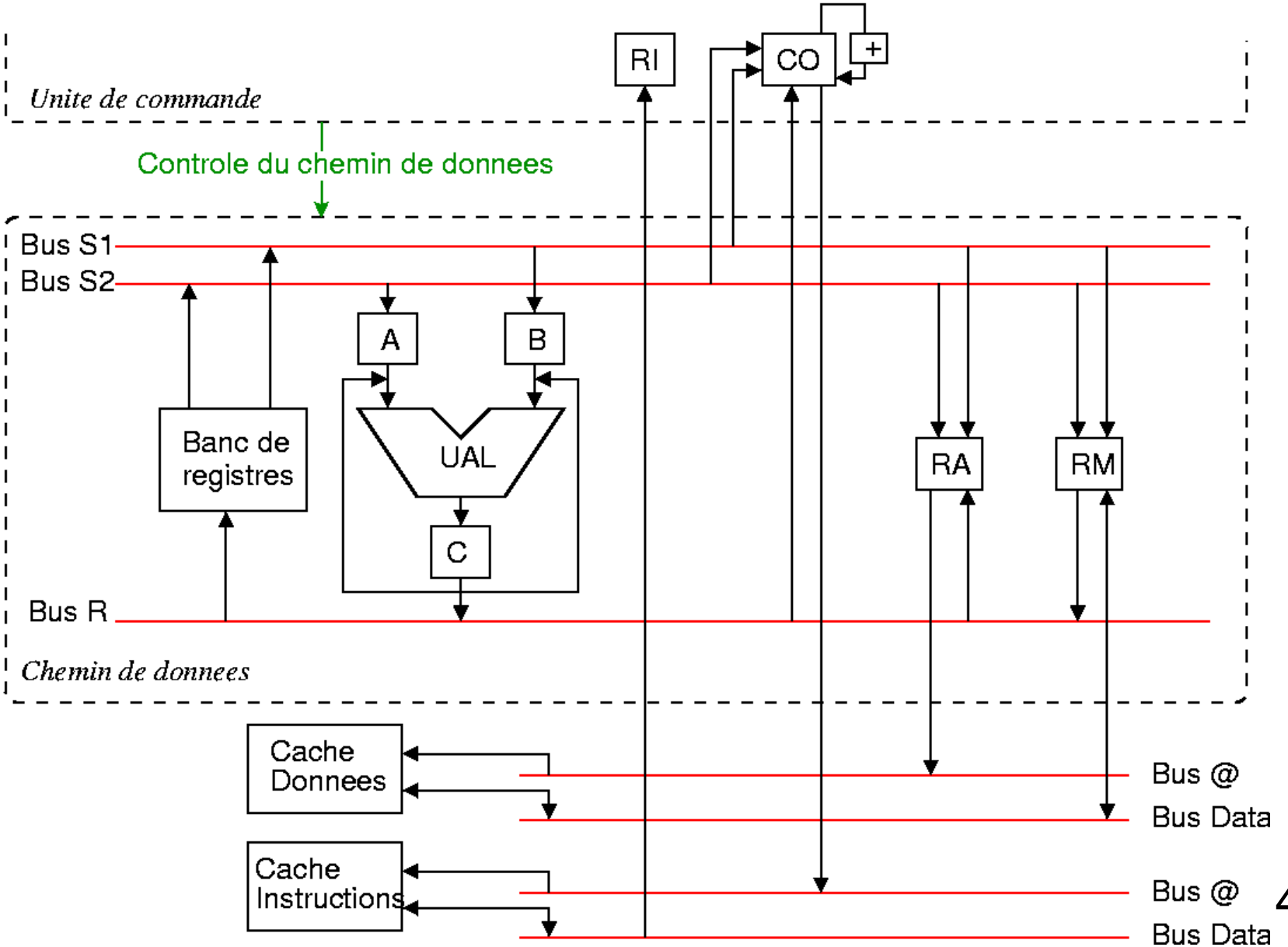


    - La deuxième instruction est suspendue tant que R1 n'est pas écrit
  - Court-circuit du pipeline
    - Après l'étape *EX* de la première instruction, on connaît la valeur de R1 : on la réinjecte directement dans l'UAL sans attendre son écriture au niveau du banc de registre



Pipeline – nouveau chemin de données

- Nouveau chemin de données avec court-circuit du pipeline et accès mémoire via 2 parties du cache

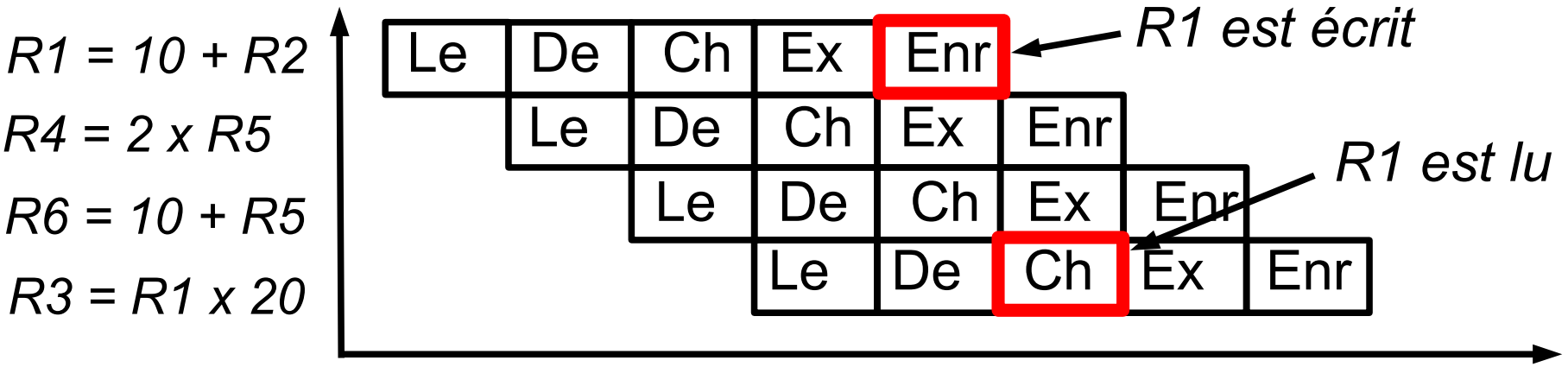


Pipeline – aléas de données

- Aléa de données : cas avec réordonnancement
  - **R1** = 10 + R2
    R3 = **R1** x 20
    R4 = 2 x R5
    R6 = 10 + R5
    - Dépendance de données entre les 2 premières instructions : aléa de données dans le pipeline
  - Réordonnancement pour éviter cet aléa
    - On place les 2 autres instructions entre ces 2 instructions
    - **R1** = 10 + R2
      R4 = 2 x R5
      R6 = 10 + R5
      R3 = **R1** x 20

Pipeline – aléas de données

- Aléa de données : réordonnancement



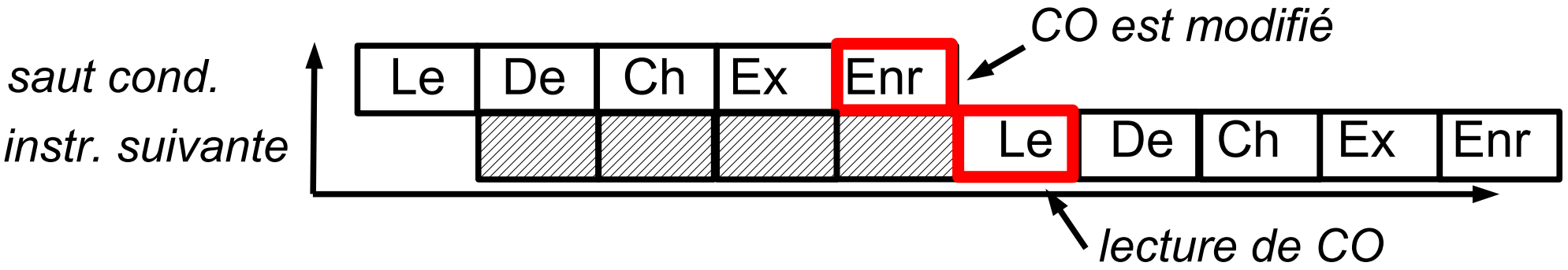
  - Grâce à ce réordonnancement : pipeline non suspendu pour aucune des 4 instructions
  - Utilisation optimale du pipeline
- 2 types de réordonnancement
  - Logiciel : fait par le compilateur
  - Matériel : fait par le processeur en interne

Pipeline – aléas de contrôle

- Aléas de contrôle
  - ```
    if (R1 > 30)
        then R3 = 10 + R1
        else R3 = 20 + R1
    ```
  - Fonctionnement du saut conditionnel
    - En fonction du résultat du test, le contenu de CO est modifié avec l'adresse de la prochaine instruction
    - Phase EX : exécution de la comparaison par l'UAL
    - Phase ENR : écriture de CO en fonction du résultat du test
  - Problème
    - Doit connaître la valeur du test de valeur de R1 pour savoir quelle est l'instruction suivante à exécuter
    ```

Pipeline – aléas de contrôle

- Aléas de contrôle (suite)
  - Solutions
    - Attendre que le bon CO soit connu : peu efficace
    - Réordonnancer le code : pas toujours suffisant et possible
    - Prédire quelle sera la valeur de R1 et commencer le calcul suivant selon cette prédiction
  - Solution avec attente



    - Doit attendre le ENR précédent avant de faire le LE : on passe en exécution purement séquentielle !

Prédiction de branchement

- Aléas de contrôle : prédictions de branchement pour en limiter les conséquences
  - Indispensable pour efficacité du pipeline
  - A l'aide de tables statistiques dynamiques
    - Prédit le résultat d'un test
    - On commence ensuite l'instruction suivante prédite
  - Problème si prédiction erronée
    - On a commencé des calculs inutiles
    - Vidage du pipeline pour reprendre dans un état correct
      - Trés couteux en temps
      - Très pénalisant pour des pipelines profonds

Prédiction de branchement

- Principes de la prédiction
  - Mémoriser les adresses des branches du programme et regarder celles qui sont souvent atteintes
  - Exemple
    - ```
      1    R0 = R2 - 3
      2    if R1 = 0 jump suite
      3    R3 = 2 x R1
      4    R4 = R3 + R1
           suite:
      5    R3 = 0
      ```
    - Deux branches : adresses 3 et 5
    - Prédiction : lors du saut conditionnel à l'adresse 2, on prendra la branche la plus souvent atteinte
      ```

Prédiction de branchement

- Deux éléments pour fonctionnement
  - Tampon des branches cibles (BTB : Branch Target Buffer)
    - Contient les adresses des branches du programme
  - Table de l'historique des branchements (BHT : Branch History Table)
    - Mémoriser l'historique des choix de branchements faits précédemment pour faire des prédictions
    - Fonctionnement dépend de l'algorithme utilisé
    - Exemple basique : 2 bits associés à chaque branche
      - 00 : branchement jamais pris jusqu'à présent
      - 01 : branchement parfois pris jusqu'à présent
      - 10 : branchement souvent pris jusqu'à présent
      - 11 : branchement toujours pris jusqu'à présent
  - Mise à jour des BTB et BHT pendant l'exécution du programme

Prédiction de branchement

- Pour plus d'efficacité des prédictions
  - Augmenter la taille du BTB et du BTH
    - Pour pouvoir gérer plus de branches (BTB)
      - Si BTB trop petit, il ne stocke pas toutes les branches : pas de prédictions possibles pour toutes les branches
    - Pour avoir un historique plus long et précis (BHT)
    - Pb : temps d'accès plus long car tables plus grandes
  - Augmenter la qualité de la prédiction avec des algorithmes plus efficaces
    - Pb : prend un temps plus long qu'avec des algorithmes plus simples
  - Dans les 2 cas : augmentation du temps de la prédiction
    - Contraire au besoin de connaître au plus tôt la prédiction
    - Limite la montée en fréquence du processeur
- Efficacité des prédictions
  - En moyenne, autour de 90% des prédictions sont correctes

Pipeline – conclusion

- Influence de la profondeur du pipeline
  - Avantage d'un pipeline long
    - Plus d'instructions en exécution parallèle
    - Montée en fréquence du processeur facilitée
    - Donc gain en nombre d'instructions exécutées en un temps donné
  - Inconvénient d'un pipeline long
    - Une erreur de prédiction est plus coûteuse
      - Plus d'opérations en cours d'exécution à annuler
  - Solution globale
    - Trouver le bon compromis entre gain d'un coté et perte de l'autre
    - Améliorer les algorithmes et unités de prédiction de branchement

Parallélisation

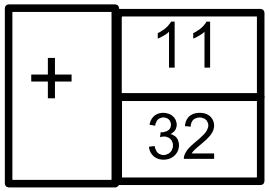
- Généralisation de la parallélisation d'exécution d'instructions
  - On a vu le système du pipeline : plusieurs instructions simultanées en exécution décalée
- Approches complémentaires
  - Ajouter des unités (approche superscalaire)
    - Unités de calculs, de commandes ou même pipeline complet
      - Exemple de l'AMD Athlon 64 : 3 UAL, 3 FPU et 3 unités de décodage d'instructions
    - Permettre l'exécution complète ou partielle de plusieurs séquences instructions en parallèle : multi-thread
      - Jusqu'à 8 threads en même temps (IBM Power8, SPARC M7)
  - Mettre plusieurs processeurs sur la même puce
    - Approche « multi-core »

Parallélisation

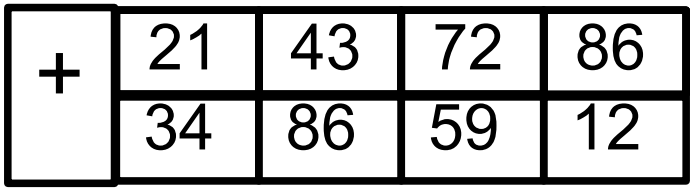
- Intérêts de l'ajout d'unités
  - Peut faire plusieurs calculs/traitements en même temps
- Problèmes de l'ajout d'unité
  - Nécessite plus de synchronisation entre unités
    - Prend du temps
  - Pas toujours pleine utilisation de toutes les unités
    - Exemple : on n'a pas forcément des séries de X additions en permanence même après réordonnancement
  - Coût important en terme de transistors et de place
    - Allongement des distances : temps de propagation
    - Coût de fabrication plus important
  - Au final : rarement plus de 2 ou 3 unités d'un type

Parallélisation : calcul vectoriel

- Autre technique
  - Unités de calcul vectoriel
  - Effectuer un même calcul avec plusieurs valeurs en parallèle (vecteur de 4, 2 ou 1 valeurs)
    - Parallélisme SIMD : Single Instruction, Multiple Data



Addition standard



Addition vectorielle à 4 valeurs

- Avantages
  - Généralement plus simple et moins lourd de paralléliser
    - En ayant une unité vectorielle avec vecteurs de taille X
    - Que X unités complètes standard non vectorielles

Parallélisation : multi-core

- Approche multi-core
  - Mettre plusieurs coeurs de processeurs sur la même puce (le même « die »)
    - Jusqu'à 32 avec le SPARC M7
    - Mais plus couramment entre 4 et 8
  - Utilité : faire du multi-processeur mais avec un seul
    - Possibilité de multi-processeur sur des cartes mères ayant un seul support physique de processeur
    - Au niveau logiciel, permet l'exécution de threads en parallèle
      - Plusieurs applications en parallèle
      - Plusieurs threads d'une même application

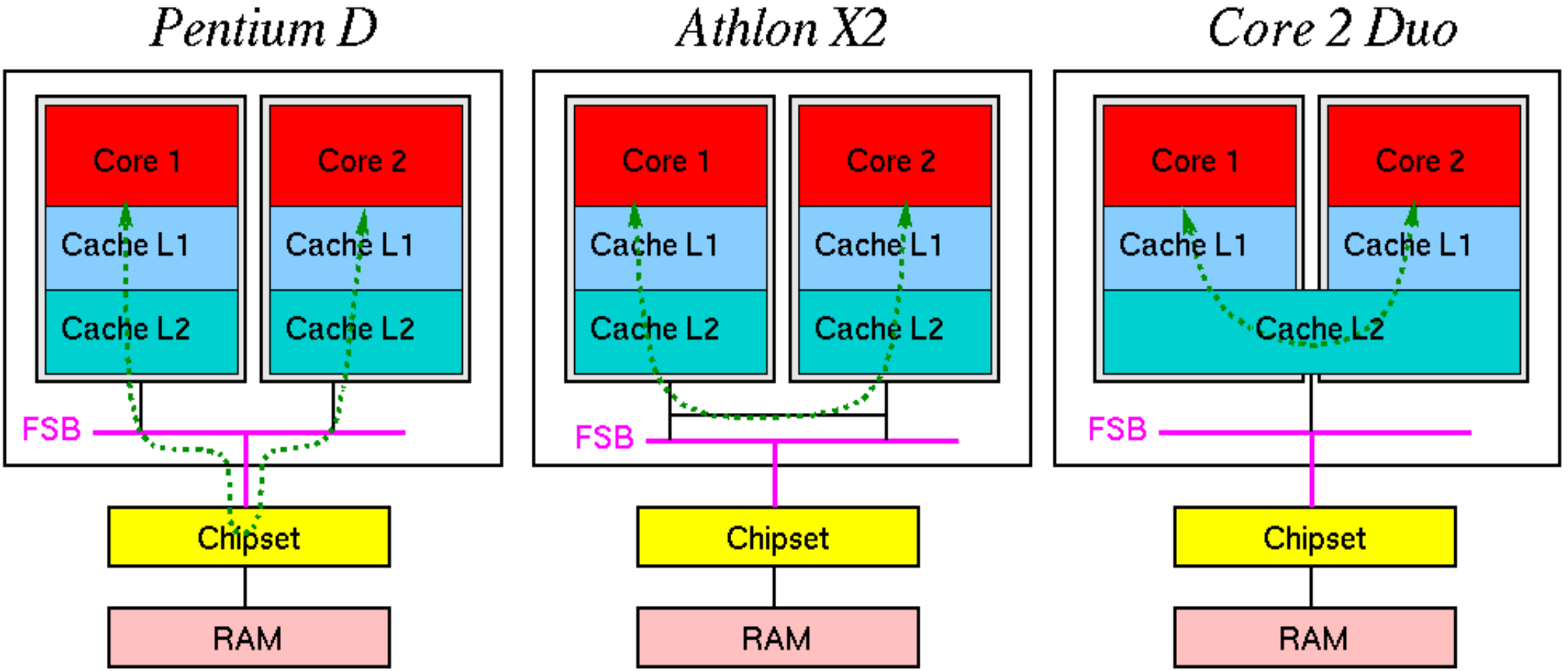
Parallélisation : multi-core

- Premières approches du multi-core
  - Intel Pentium D / XE : approche « basique »
    - Deux processeurs entièrement dupliqués sur la même puce, avec chacun embarquant
      - Unités de commande
      - Unités de calculs
      - Mémoire cache (niveaux L1 et L2)
    - Chaque core communique directement et uniquement avec le chipset via le FSB
      - Pb de performances pour communication entre 2 cores car on passe par le chipset
      - Occupation inutile du FSB avec les communications entre les cores

Parallélisation : dual-core

- Approche AMD Athlon 64 X2
  - 2 cores entièrement dupliqués également mais avec
    - Un bus de communication interne au CPU très rapide entre les 2 cores
    - Améliore les performances de communication entre les 2 cores
- Approche Intel Core 2 Duo
  - Duplication des 2 cores mais avec un cache L2 commun
  - Meilleure performance pour communication entre 2 cores
    - On passe par le cache directement sans passer par un bus dédié ou le FSB
  - Généralisation aujourd'hui du cache commun
    - Mais au niveau L3
    - Chaque core embarque son L1 et L2

Parallélisation : dual-core



- Problème dans tous les cas
  - Gérer la cohérence des données entre les caches et la mémoire centrale

Jeux d'instruction

- Deux grands types de jeux d'instructions
  - CISC : Complex Instruction Set Computing
    - Exemples : processeurs Intel et AMD : familles x86
    - Fonctionne en modèle mémoire (3,3) généralement
  - RISC : Reduced Instruction Set Computing
    - Exemples : Oracle Sparc et IBM PowerPC
      - Et les disparus : DEC Alpha, MIPS, HP PA ...
    - Fonctionne en modèle mémoire (3,0) généralement
- Différences fondamentales
  - Intructions en nombre réduit pour le RISC
  - Instructions de taille fixe (code opération + opérande) pour le RISC

Jeux d'instruction

- Exemple de calcul
  - Faire la somme du registre R1 et de la valeur à l'adresse 100 en mémoire et placer le résultat à l'adresse 200
  - En CISC
    - Plusieurs types d'additions gérant de multiples modes d'accès en mémoire et aux registres
    - Exemples : R = accès registre, A = accès adresse mémoire
      - ADD R, R, R
      - ADD R, A, A
      - ADD A, A, A ....
    - Une seule opération nécessaire :
      - `ADD R1, @100, @200`

Jeux d'instructions

- En RISC
  - Calculs uniquement avec des registres
  - Accès en mémoire : uniquement 2 types
    - Lecture mémoire vers un registre
    - Écriture mémoire à partir d'un registre
  - Exemple précédent : besoin de 3 instructions
    - ```
      LOAD R2, @100
      ADD R3, R1, R2        (R3 = R1 + R2)
      STORE R3, @200
      ```
  - Conséquence sur nombre de registres d'un mode (3,0)
    - Un processeur RISC possèdent beaucoup plus de registres généraux qu'un processeur CISC
      - Premier CPU x86 (CISC) : 8 registres
      - Architecture SPARC (RISC) : entre 72 et 640
        - Généralement 160
      ```

Jeux d'instructions

- Codage d'une instruction
  - Sur un nombre de bits, on place le code de l'opération et ses paramètres (adresse mémoire ou registre)
- Exemples (fictifs)
  - CISC
    - 8 bits instruction (256 instructions)
    - Adresse mémoire : 16 bits
    - Registre : 4 bits (16 registres généraux)
    - `ADD R1, R2 , @200` : 8 + 4 + 4 + 16 = 32 bits
    - `ADD R1, @100, @200` : 8 + 4 + 16 + 16 = 44 bits
  - RISC
    - 6 bits instruction (64 instructions)
    - Adresse mémoire : 16 bits
    - Registre : 6 bits (64 registres généraux)
    - `LOAD R2, @100` : 6 + 6 + 16 = 28 bits
    - `ADD R3, R1, R2` : 6 + 6 + 6 + 6 = 24 bits

Jeux d'instructions

- Taille de codage des instructions
  - RISC : taille fixe
    - Tout tient en 32 bits pour notre exemple
  - CISC : taille variable
    - Selon le nombre d'adresses mémoires codées dans l'instruction
- Parallélisation facilitée en RISC
  - On sait que tous les 32 bits il y a une instruction
  - Peut commencer à lire une nouvelle instruction sans avoir fini de décoder la précédente
- Parallélisation plus compliquée en CISC
  - Doit décoder les premiers 32 bits pour savoir si l'instruction est complète ou si on doit lire les 32 bits suivants pour l'avoir en entier
  - Durée d'exécution de l'instruction variable (selon nombre accès mémoire, registres …)
- RISC compense l'augmentation du nombre d'instructions pour un même programme par une optimisation de leur exécution

Jeux d'instructions

- L'architecture RISC a montré qu'elle était plus performante
- Tous les processeurs fonctionnent en RISC de nos jours
  - Pour un processeur avec un jeu d'instruction CISC
    - Traduction interne des instructions CISC en micro-opérations de type RISC
    - C'est le cas pour tous les processeurs AMD et Intel x86
      - Apparition de ce fonctionnement avec le PentiumPro chez Intel
- Autre avantage de la traduction en micro-instructions
  - Permet une meilleure gestion des unités de calculs et d'optimiser le code grâce à un réordonnancement des micro-instructions
  - Certains processeurs RISC transforment aussi les instructions natives RISC en micro-instructions internes
    - Exemple : PowerPC d'IBM

Conclusion sur performances

- Techniques pour augmenter les performances
  - Augmentation de la fréquence de fonctionnement
  - Parallélisation pour exécuter plusieurs instructions simultanément
    - Pipeline
      - Prédiction de branchement
      - Réordonnancement des séquences d'instructions
    - Duplication des unités de calculs et commandes
    - Unités de calculs vectoriels
    - Architecture multi-core
  - Mémoire cache
  - Jeu d'instruction (interne) de type RISC

Conclusion sur les performances

- Problèmes
  - Ne peut pas jouer sur tous les points en même temps car interdépendances
  - Exemples
    - Augmentation de la taille du pipeline
      - Permet montée en fréquence
      - Mais réduit efficacité de la parallélisation
    - Augmentation de la prédiction de branchement
      - Diminue les aléas de contrôle
      - Mais limite la montée en fréquence
  - Recherche du meilleur compromis entre tous ces points

Exemple de l'Athlon 64
&
Evolution des performances

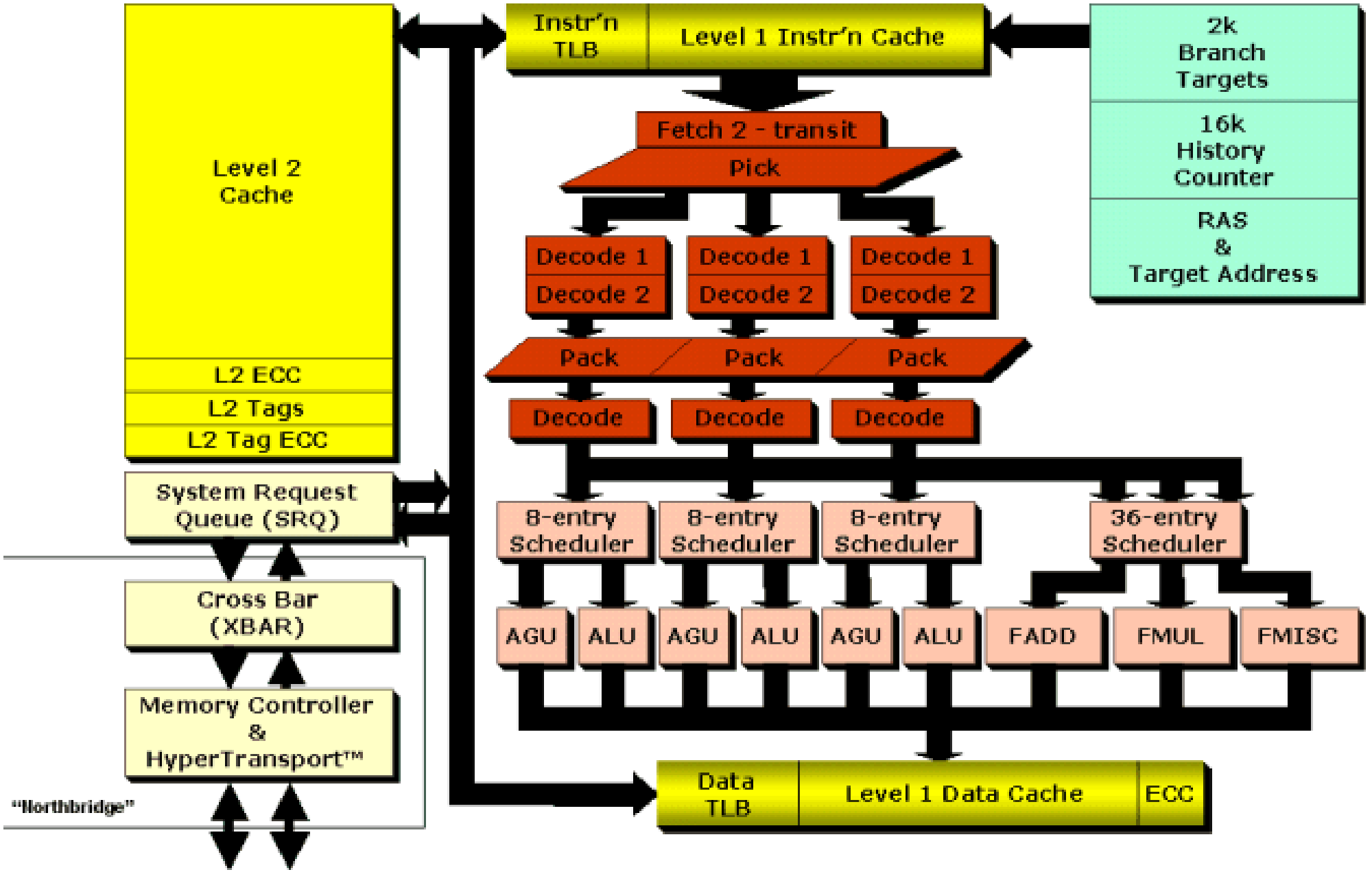
Exemple : AMD Athlon 64

- Étude des éléments d'un processeur (relativement) récent
  - AMD Athlon 64
  - Commercialisé entre 2003 et 2008
  - Décliné en plusieurs versions et modèles
    - Y compris une version dual-core
- Caractéristiques générales
  - Processeur avec jeu d'instructions x86
  - Fréquence : de 1,8 à 3,2 Ghz
  - Cache L1 de taille 2 x 64 Ko
  - Cache L2 de taille 512 Ko ou 1024 Ko selon modèle
    - L1 et L2 sont exclusifs

Exemple : AMD Athlon 64

- Cache L1
  - 64 Ko pour données et 64 Ko pour instructions
  - Taille d'une ligne : 64 octets
  - Associatif par ensemble de 2 (2-way associative)
  - Ecriture différée : write-back
  - Latence : 3 cycles d'horloge
- Cache L2
  - 512 Ko ou 1024 Ko
  - Taille d'une ligne : 64 octets
  - Associatif par ensemble de 16 (16-way associative)
  - Latence : 5 cycles d'horloge
  - Pre-fetching pour charger en avance des données/instructions dans le cache L2

Exemple : AMD Athlon 64



Exemple : AMD Athlon 64

- Mémoire cache
  - *Level 1 instruction cache* : cache L1 pour instructions
  - *Level 1 data cache* : cache L1 pour données
  - *Level 2 cache* : cache L2
  - *TLB* : tampon de traduction anticipée
    - Cache pour correspondance entre adresses virtuelles et physiques
  - *ECC* : contrôle de parité pour vérifier l'intégrité des données

Exemple : AMD Athlon 64

- Prédiction de branchement
  - *2k branch targets* : tampon des branches cibles d'une taille de 2 Ko
  - *16k history counter* : table de l'historique des branchements, taille de 16 Ko
- Communication avec mémoire
  - *Memory Controller, System Request Queue, Cross Bar*
  - Le contrôleur mémoire est intégré au processeur pour l'Athlon 64 au lieu d'être dans le pont nord du chipset

Exemple : AMD Athlon 64

- Réalisation du pipeline, exécution des instructions
  - Eléments en plusieurs exemplaires
    - 3 unités de calculs entiers : ALU
      - Calculs simples (addition, ou, décalage ...) en 1 cycle d'horloge
      - Multiplication : 3 cycles d'horloge en 32 bits et 5 en 64 bits
    - 3 unités de calculs d'adresses mémoire : AGU
    - 3 unités de calculs flottants : FADD, FMUL, FMISC
      - Qui font aussi les calculs scalaire MMX, 3D Now!, SSE1 et 2
    - 3 unités de décodage et gestion de l'exécution des instructions
      - 3 blocs *Decode1/Decode2* et éléments suivants

Exemple : AMD Athlon 64

	ALU	FPU
1	Fetch1	
2	Fetch2	
3	Pick	
4	Decode 1	
5	Decode 2	
6	Pack	
7	Pack/Decode	
8	Dispatch	Dispatch
9	Schedule	Stack Rename
10	Exec	Register Rename
11	Data Cache 1	Write Schedule
12	Data Cache 2	Schedule
13		Register Read
14		FX 0
15		FX 1
16		FX 2
17		FX 3

- Description du pipeline
  - 12 étages en calculs entiers
  - 17 étages en flottants

Exemple : AMD Athlon 64

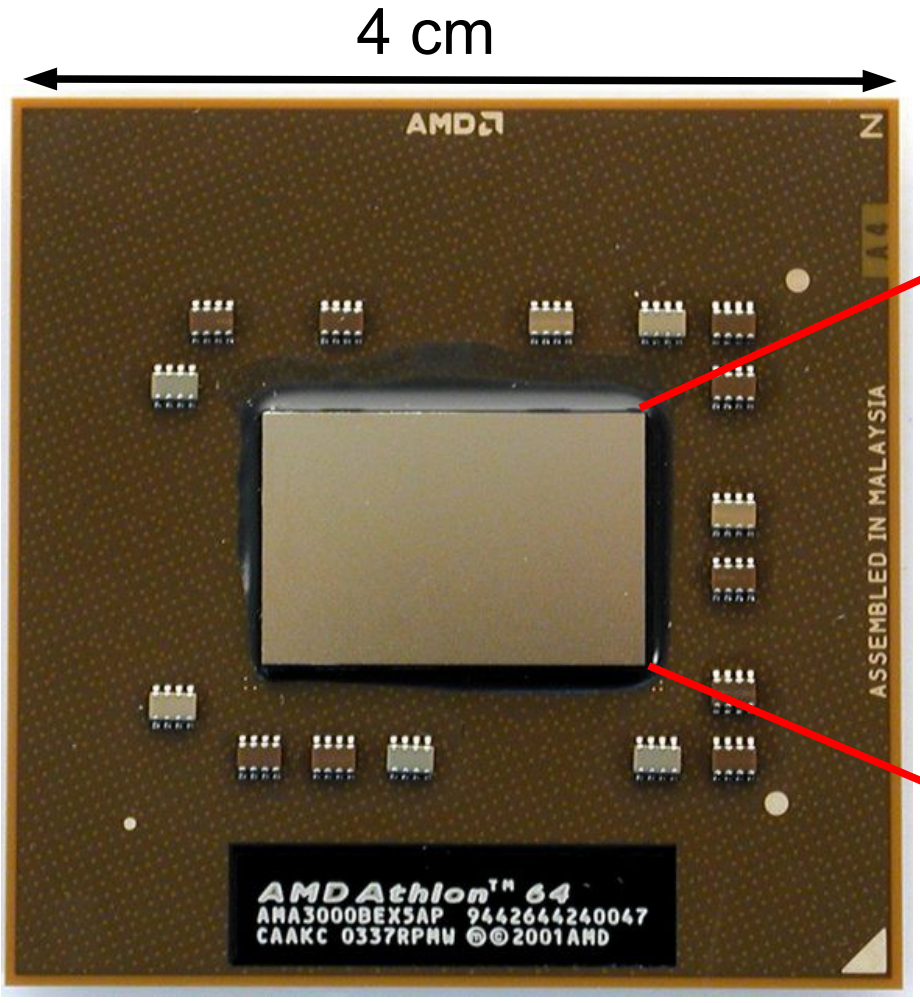
- Étapes du pipeline, pour calcul entier
  - (1, 2 & 3) *fetch* et *pick*
    - Récupère les instructions suivantes via le cache L1 d'instructions et la prédiction de branchement
    - Fournit une instruction à chacune des 3 unités de décodage
  - (4 & 5) : *decode 1* et *2*
    - Décodent les instructions x86 et les transforment en micro-opérations (µOPs) de longueur fixe
      - Pour fonctionnement interne en mode RISC
  - (6 & 7) : *pack* et *pack/decode*
    - Continue décodage pour instructions longues
    - Regroupement de µOPs en pack

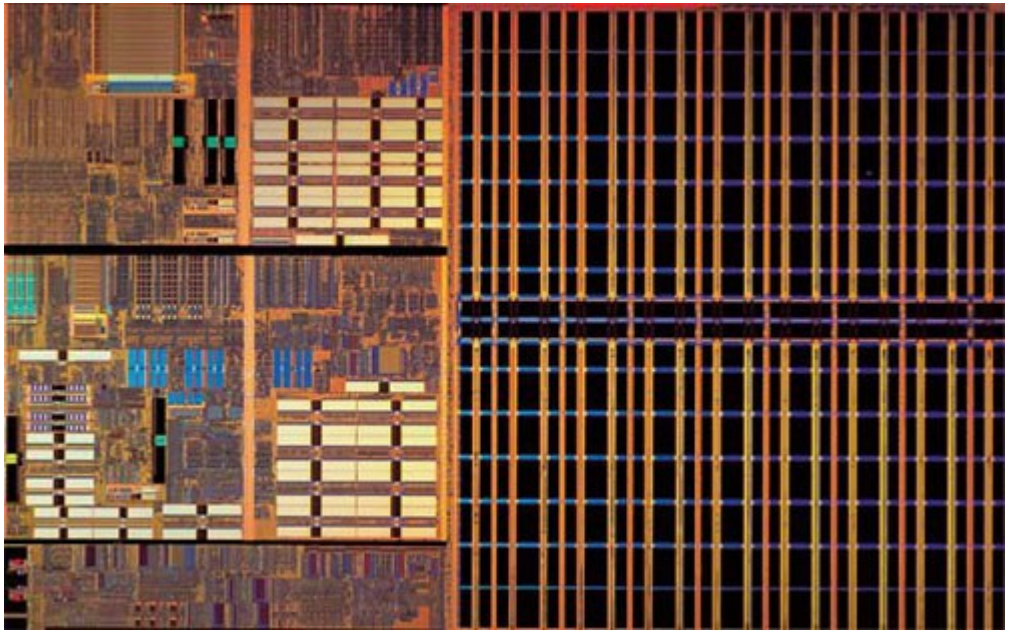
Exemple : AMD Athlon 64

- Étapes du pipeline, pour calcul entier (suite)
  - (8 & 9) *dispatch* et *schedule*
    - Les packs sont distribués aux unités de calcul (ALU, FPU selon les µOPs)
  - (10) *exec*
    - Exécution des µOPs
  - (11 & 12) *data cache 1* et *2*
    - Accès en cache L1 de données, en lecture et/ou écriture selon les µOPs
      - Lecture pour chargement dans un registre
      - Écriture résultat d'un calcul

Exemple : AMD Athlon 64

- 105 millions de transistors
- Consommation électrique : 90 Watts
- « Die » de 193 mm2





Mesures de performances

- Plusieurs types de calcul des performances
  - IPC : Instruction Per Cycle
    - Nombre moyen d'instructions exécutées par cycle d'horloge
    - Dépend du programme utilisé
      - Nombre d'accès mémoire, entrée/sortie, aléas générés pour le pipeline ...
    - Exemple
      - AMD Athlon XP : 2 à 2,5 instructions par cycle d'horloge
      - Intel P4 : 1,5 à 2 instructions par cycle d'horloge
    - IPC > 1 car parallélisation généralisée dans le processeur
  - MIPS : Millions d'Instructions Par Seconde
    - MIPS = Fréquence (en Mhz) x IPC

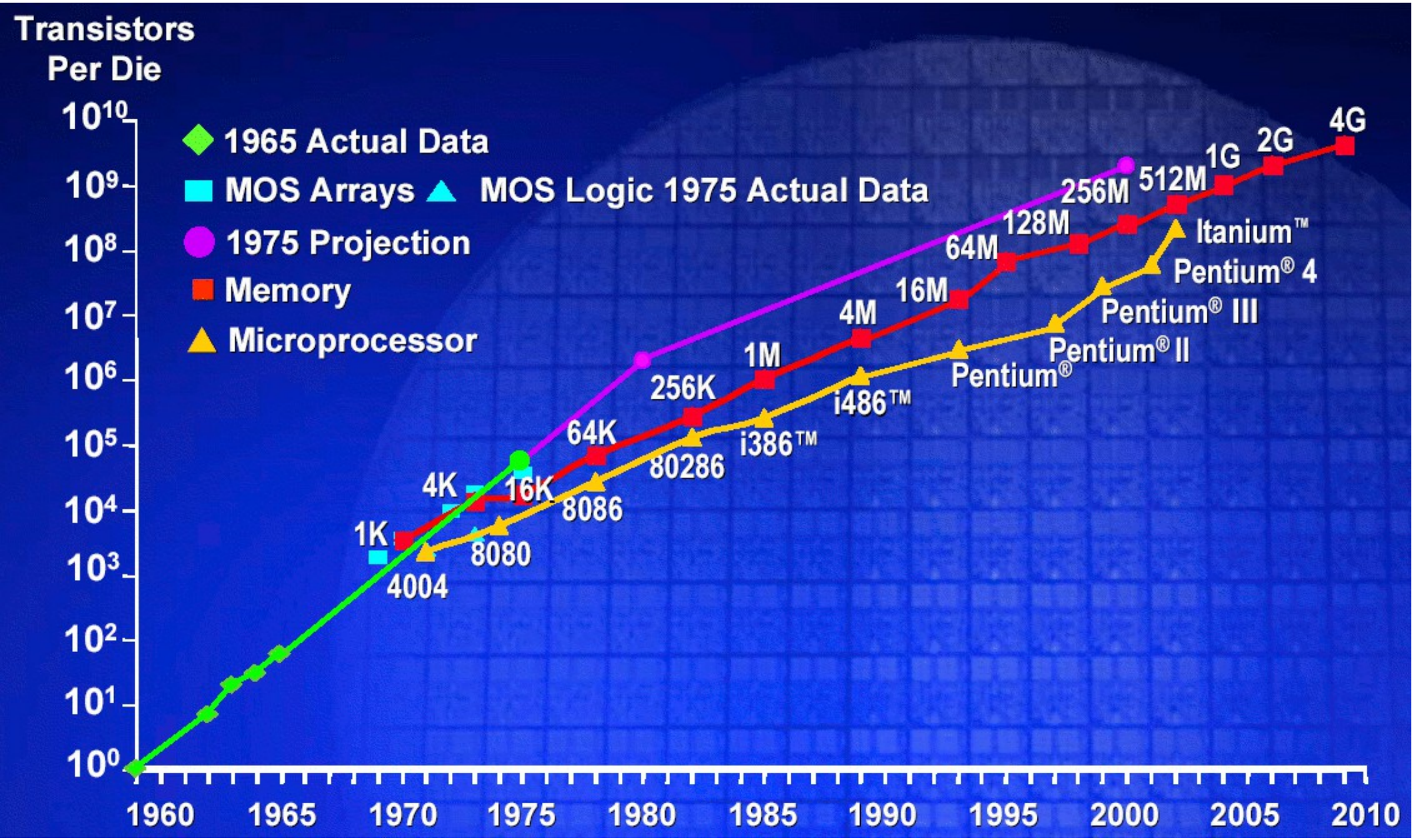
Mesures de performances

- Pour comparer les performances des processeurs
  - Utilise un benchmark
    - Logiciel réalisant un certain calcul/traitement
    - On exécute ce même traitement sur tous les processeurs pour pouvoir comparer le temps d'exécution
  - 2 benchmarks pour calculs
    - Spec Int 2000 : calculs sur des entiers
    - Spec Fp 2000 : calculs sur des flottants
  - Il existe de très nombreux autres types de benchmarks

Loi de Moore

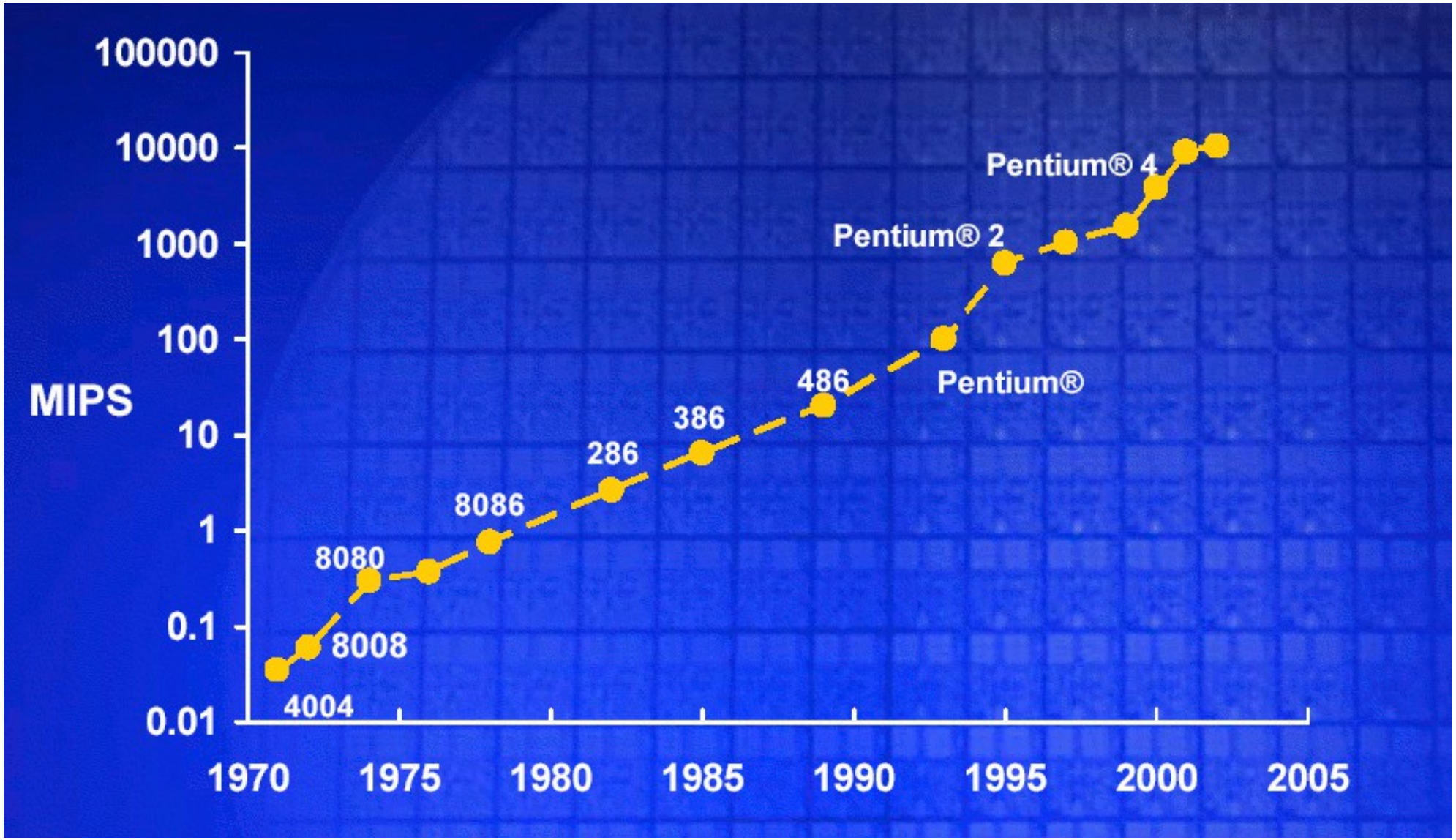
- En 1965, Gordon Moore (PDG d'Intel) prédit
  - Que le nombre de transistors utilisés dans les circuits intégrés doublerait tous les ans
    - En prenant en compte
      - Les avancées technologiques
      - Les coûts de fabrication
  - Prédiction réactualisée en 1975
    - Doublement tous les 2 ans au lieu de tous les ans
      - Complexité grandissante des circuits
  - Doublement de la puissance de calcul tous les 18 mois

Loi de Moore



Source : www.intel.com

Loi de Moore



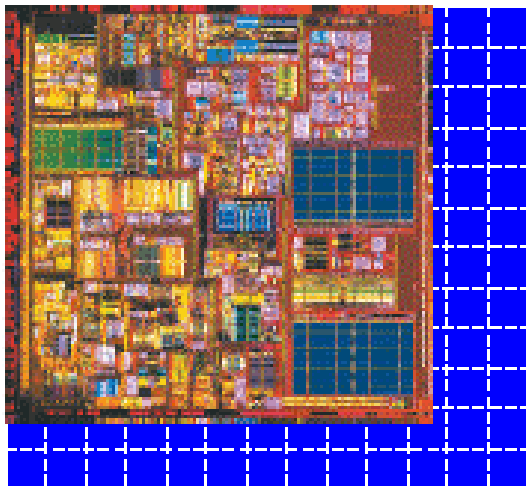
Source : www.intel.com

Visualisation évolution CPU

- 3 processeurs Intel pour ordinateurs portables
  - Toujours plus de transistors, de cache
  - Mais toujours diminution de la finesse de gravure

X = 13 mm
Y = 12 mm

2002

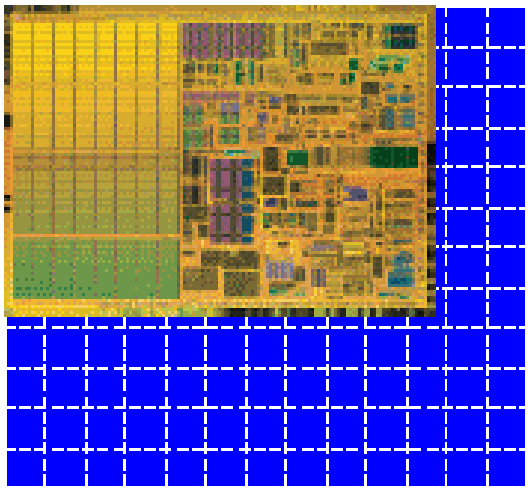


Pentium 4-M (0.13 µm / 512K)
11.62 mm * 11.34 mm =
131.8mm²

55 Millions de Transistors

Cache : 28.3 M
Core : 55-28.3 = 26.7 M

2003

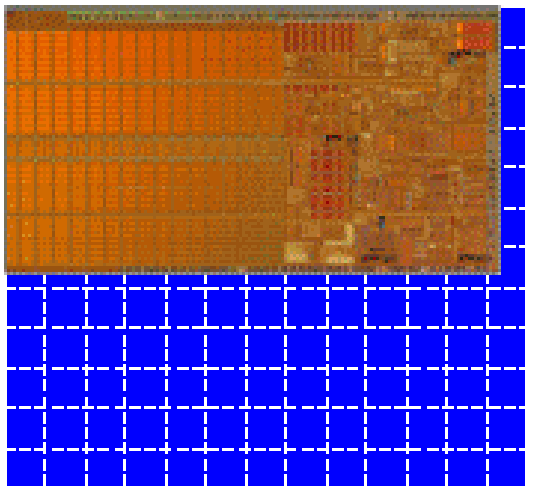


Pentium M (0.13 µm / 1M)
10.56 mm * 7.84 mm =
82.8 mm²

***77** Millions de Transistors*

Cache : 56.6 M
Core : 77-56.6 = 20.4 M

2004



Pentium M (0.09 µm / 2M)
12.33 mm * 6.78 mm =
83.6mm²

***140** Millions de Transistors*

Cache : 113.2 M
Core : 140-113.2 = 26.5 M