Microarchitecture des Processeurs Généralistes

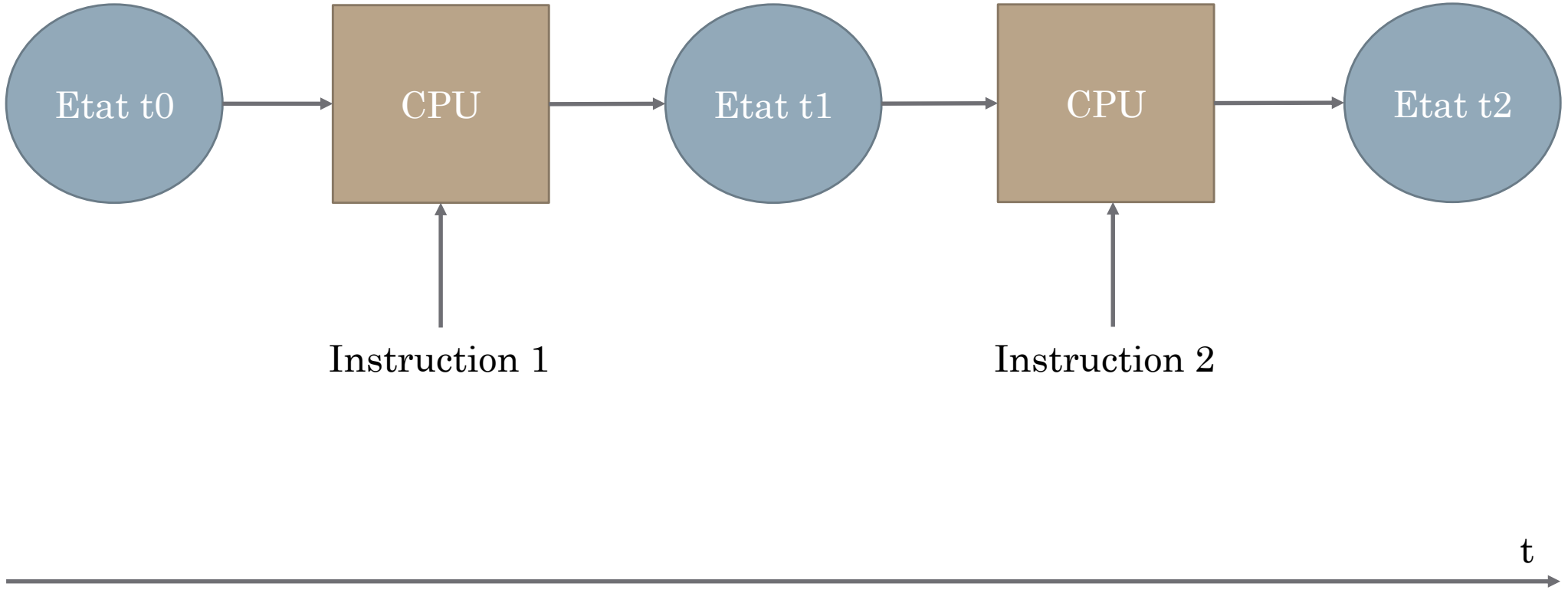
SEOC3A – CEAMC

Arthur Perais (arthur.perais@univ-grenoble-alpes.fr)

Préambule

- Le cours utilise le jeu d'instructions RISC-V dans les exemple de programmes
  - https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf
  - Ressemble beaucoup à MIPS

Processeur central (CPU)



Microarchitecture ?

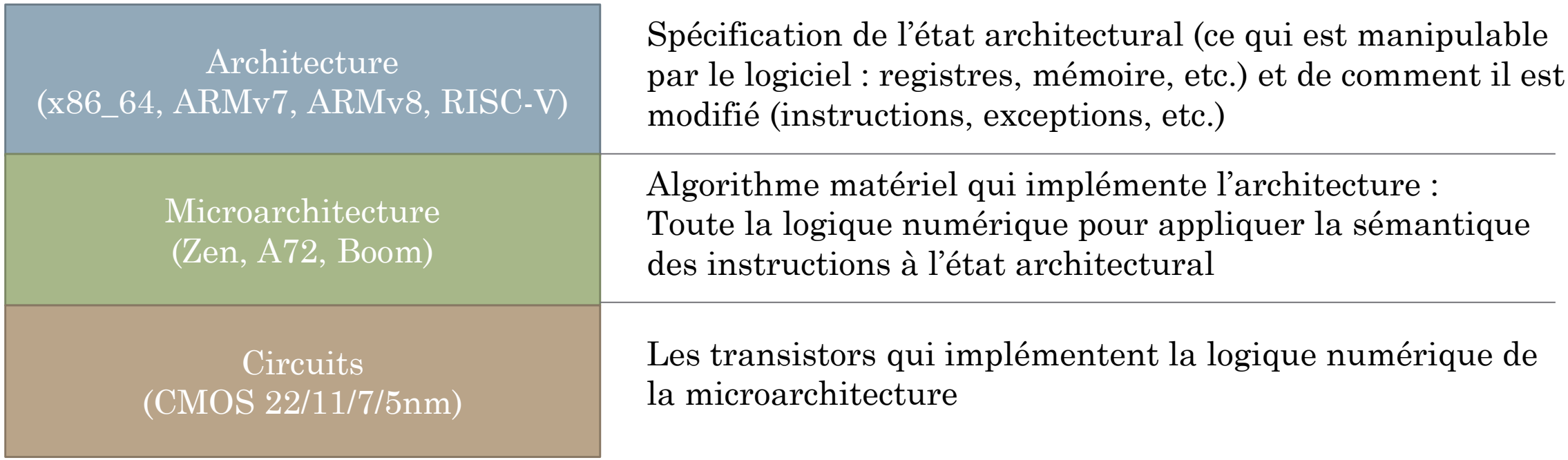
Architecture
(x86_64, ARMv7, ARMv8, RISC-V)

Spécification de l'état architectural (ce qui est manipulable par le logiciel : registres, mémoire, etc.) et de comment il est modifié (instructions, exceptions, etc.)

Microarchitecture ?

Architecture (x86_64, ARMv7, ARMv8, RISC-V)	Spécification de l'état architectural (ce qui est manipulable par le logiciel : registres, mémoire, etc.) et de comment il est modifié (instructions, exceptions, etc.)
Microarchitecture (Zen, A72, Boom)	Algorithme matériel qui implémente l'architecture : Toute la logique numérique pour appliquer la sémantique des instructions à l'état architectural

Microarchitecture ?

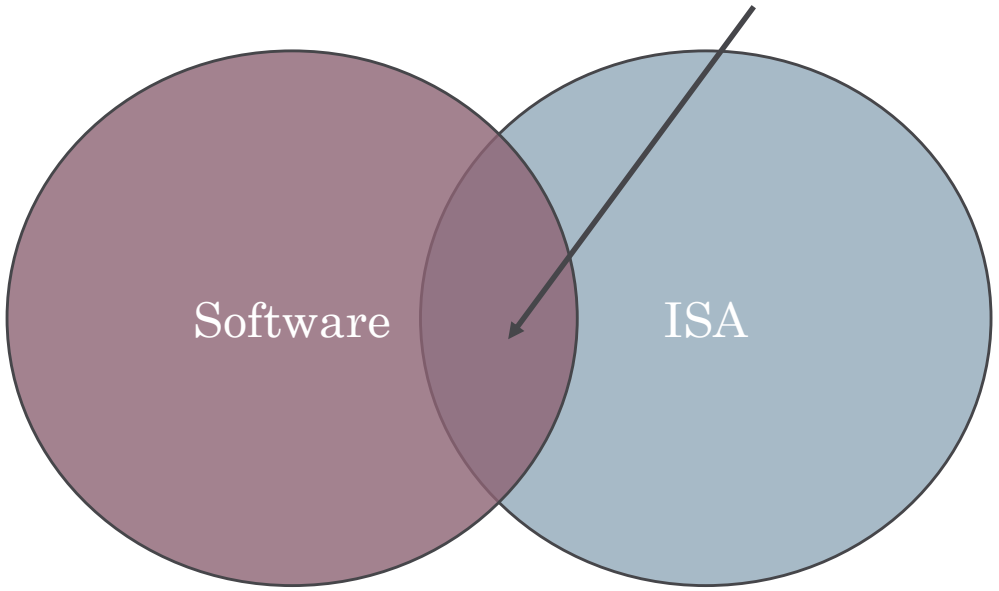


Performance – « Iron Law »

$$\frac{Temps}{Programme} = \frac{Instructions}{Programme} * \frac{Cycles}{Instruction} * \frac{Temps}{Cycle}$$

Performance – « Iron Law »

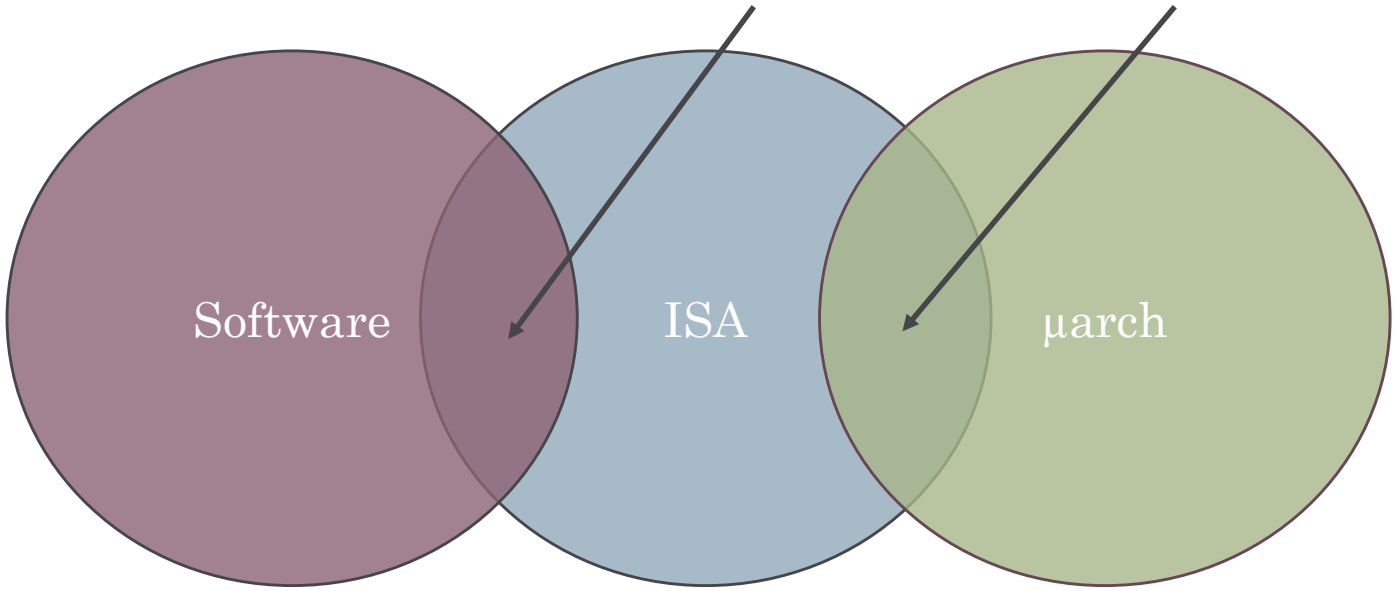
$$\frac{Temps}{Programme} = \frac{Instructions}{Programme} * \frac{Cycles}{Instruction} * \frac{Temps}{Cycle}$$



SW : Qualité de l'écriture du programme
ISA : Instructions disponibles

Performance – « Iron Law »

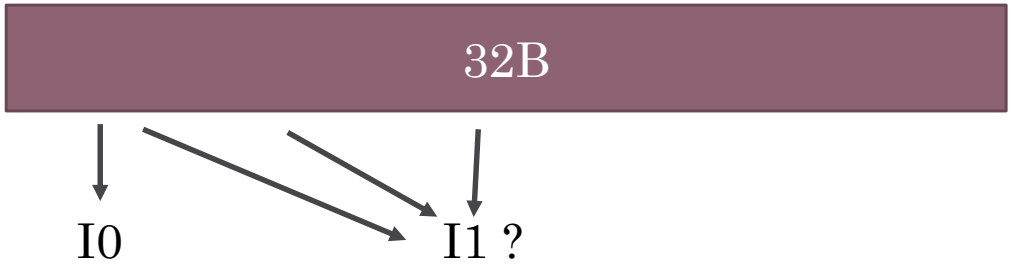
$$\frac{Temps}{Programme} = \frac{Instructions}{Programme} * \frac{Cycles}{Instruction} * \frac{Temps}{Cycle}$$



SW : Qualité de l'écriture du programme
ISA : Instructions disponibles
µarch : Efficacité de l'implémentation

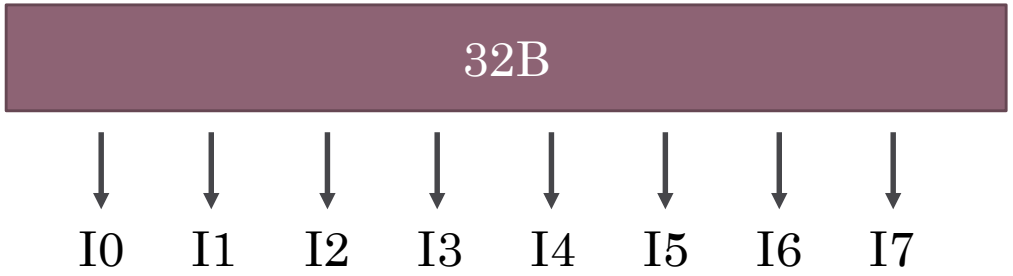
Un exemple de dépendance entre les couches

x86 : Une instruction occupe 1 à 15 octets



Processus séquentiel afin de déterminer **où** sont les instructions, puis envoyer les octets au décodeur

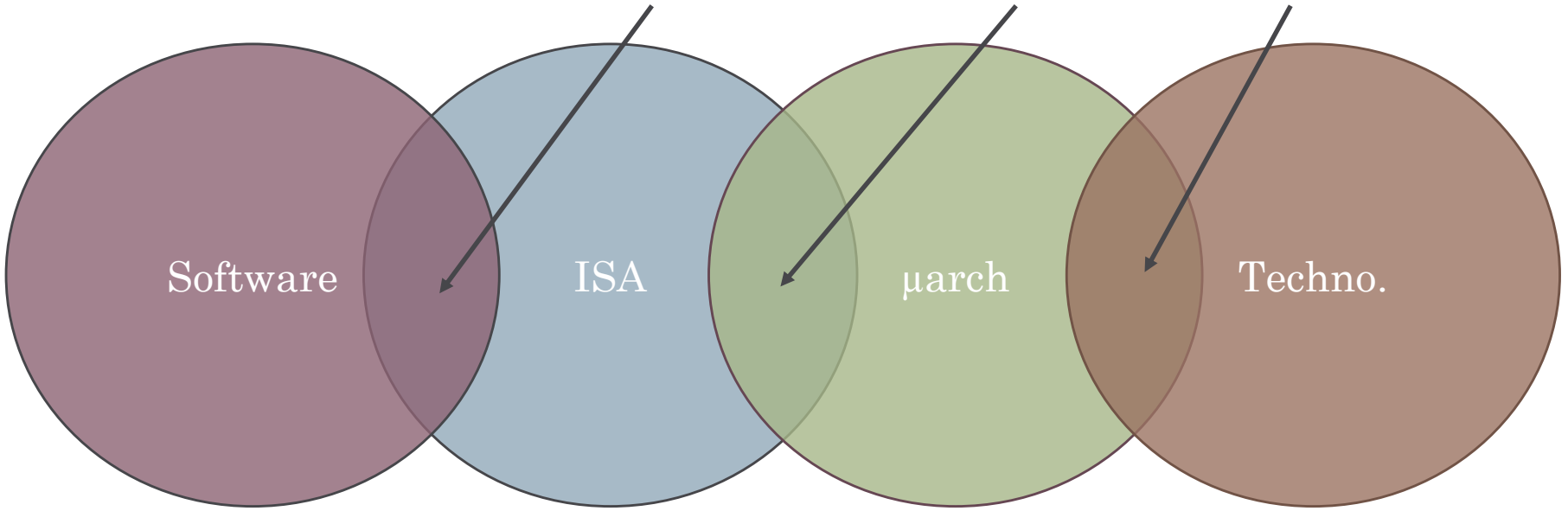
RISC-V : Une instruction occupe 4 octets



L'emplacement des instructions est déjà connu, on peut directement envoyer les octets au décodeur

Performance – « Iron Law »

$$\frac{Temps}{Programme} = \frac{Instructions}{Programme} * \frac{Cycles}{Instruction} * \frac{Temps}{Cycle}$$



SW : Qualité de l'écriture du programme
ISA : Instructions disponibles
µarch : Efficacité de l'implémentation
Techno. : Fréquence max

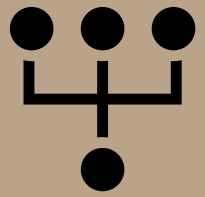
Trois « lois » intéréssantes

Moore (empirique) : Toujours plus de transistors à implémenter sur une puce

- Encore valide aujourd'hui. Pour combien de temps ?

Dennard (empirique) : A mesure que la taille d'un transistor réduit, sa consommation réduit de façon proportionnelle (densité de puissance constante)

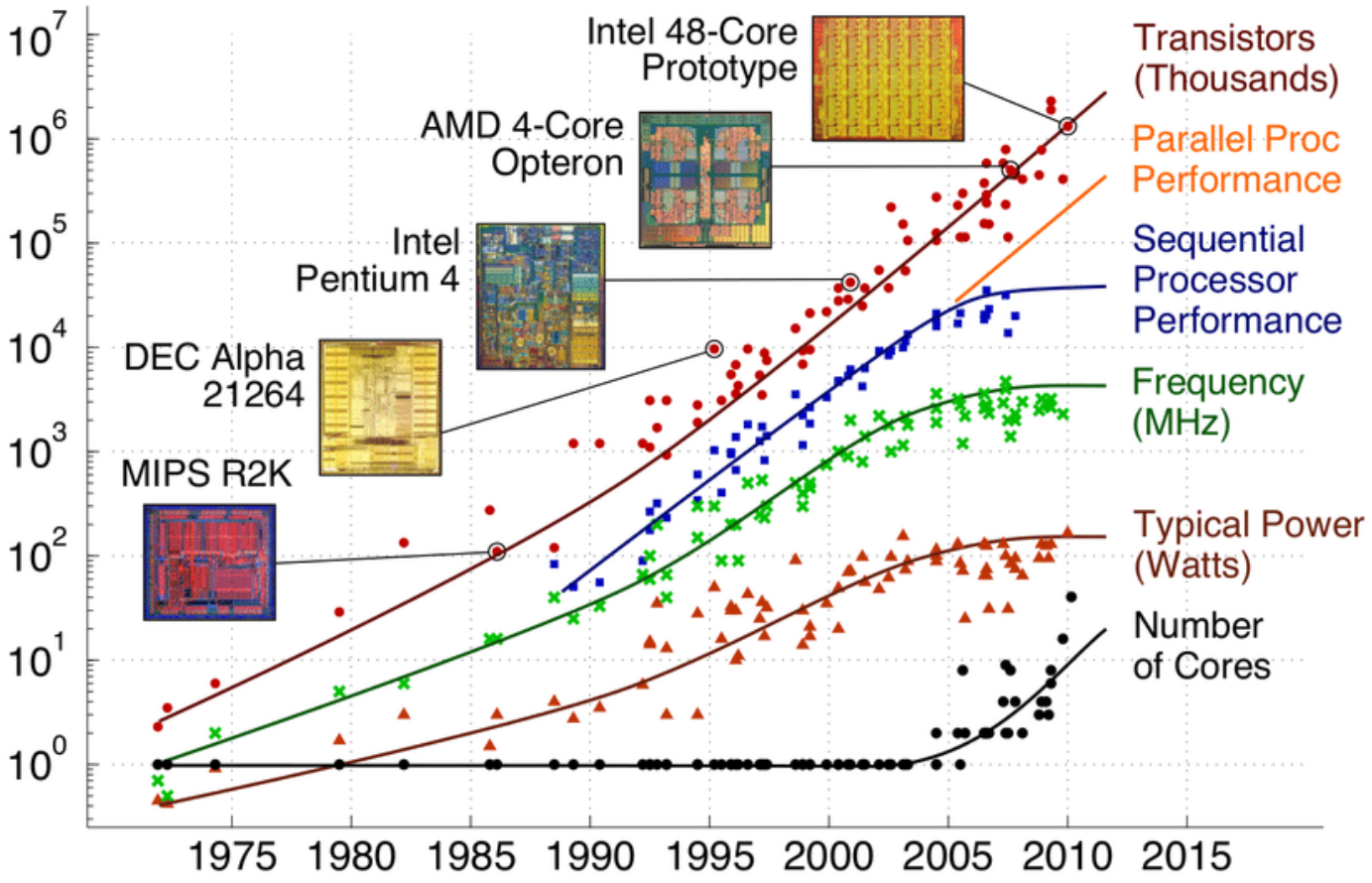
- Faux depuis 2006/2007 -> « Dark Silicon »



Amdahl (Théorique): Si on a une infinité de processeurs (cœurs), la performance d'un programme sera limitée par la partie séquentielle du programme, et donc la performance d'un seul processeur (cœur).

- Toujours vrai.

Quelques tendances



De Christopher Batten (dans ECE5950 Complex Digital ASIC Design Course Overview. URL: http://www.csl.cornell.edu/courses/ece5950)

Comment utiliser les transistors supplémentaires pour améliorer la performance séquentielle malgré la fin de la loi de Dennard?

Augmenter la performance séquentielle via la microarchitecture

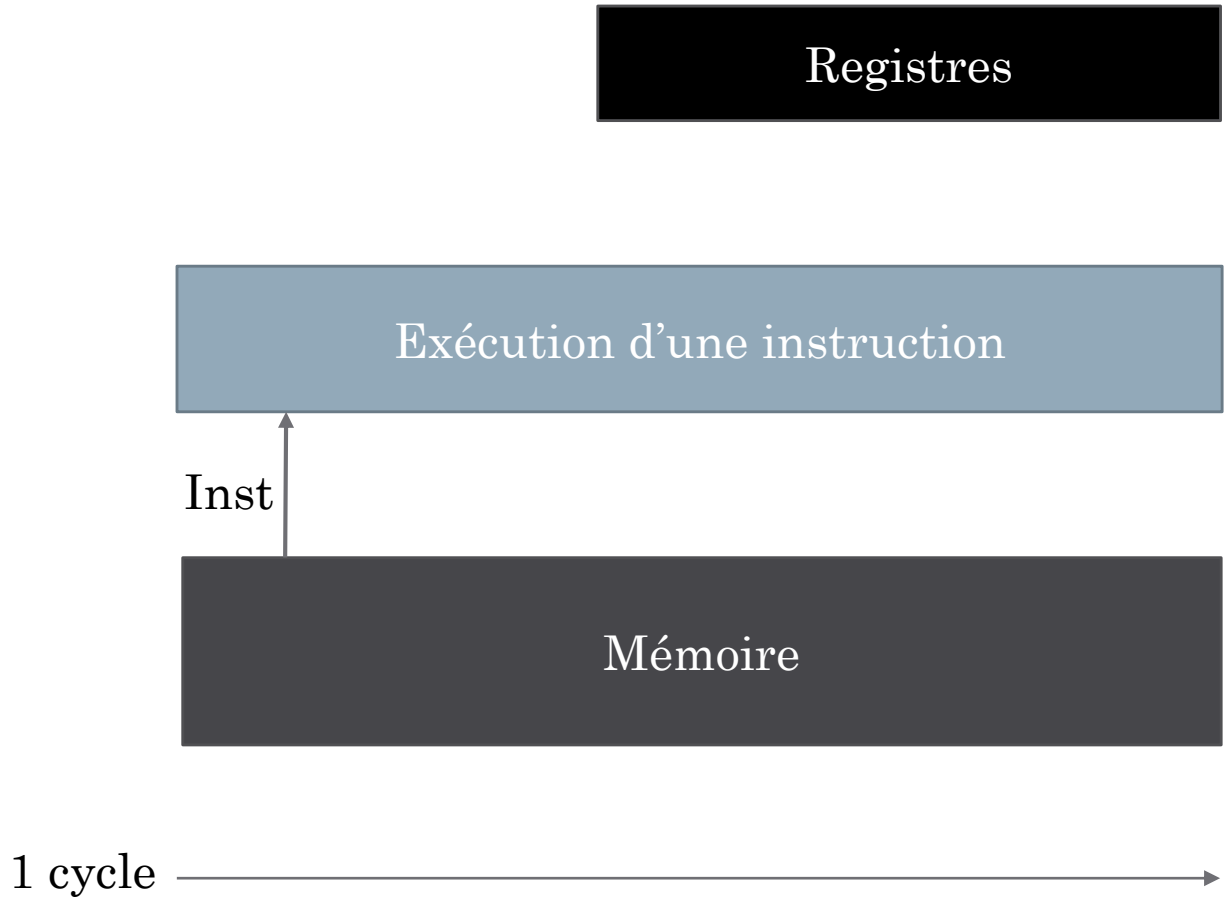
Pipelining

Spéculation

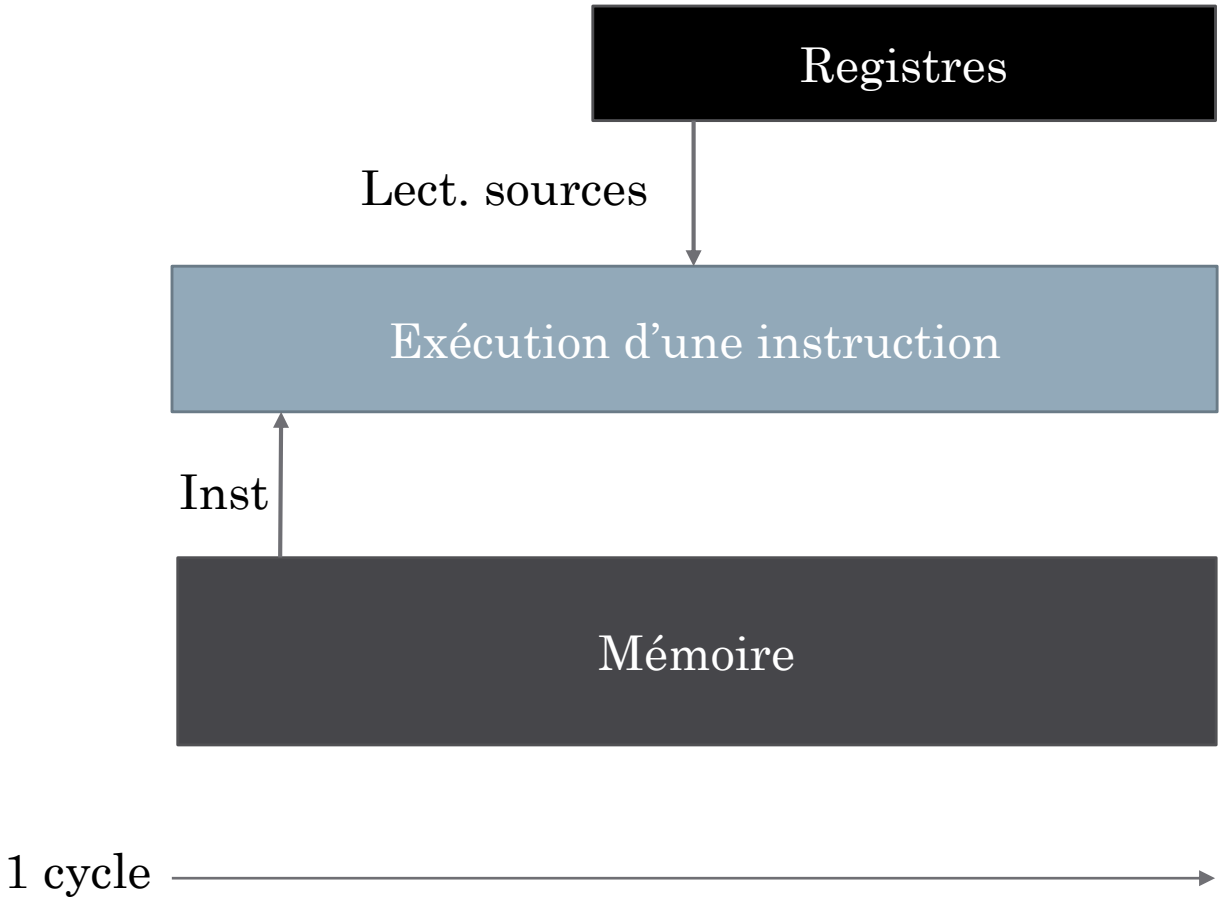
Exécution superscalaire

Ordonnancement dynamique

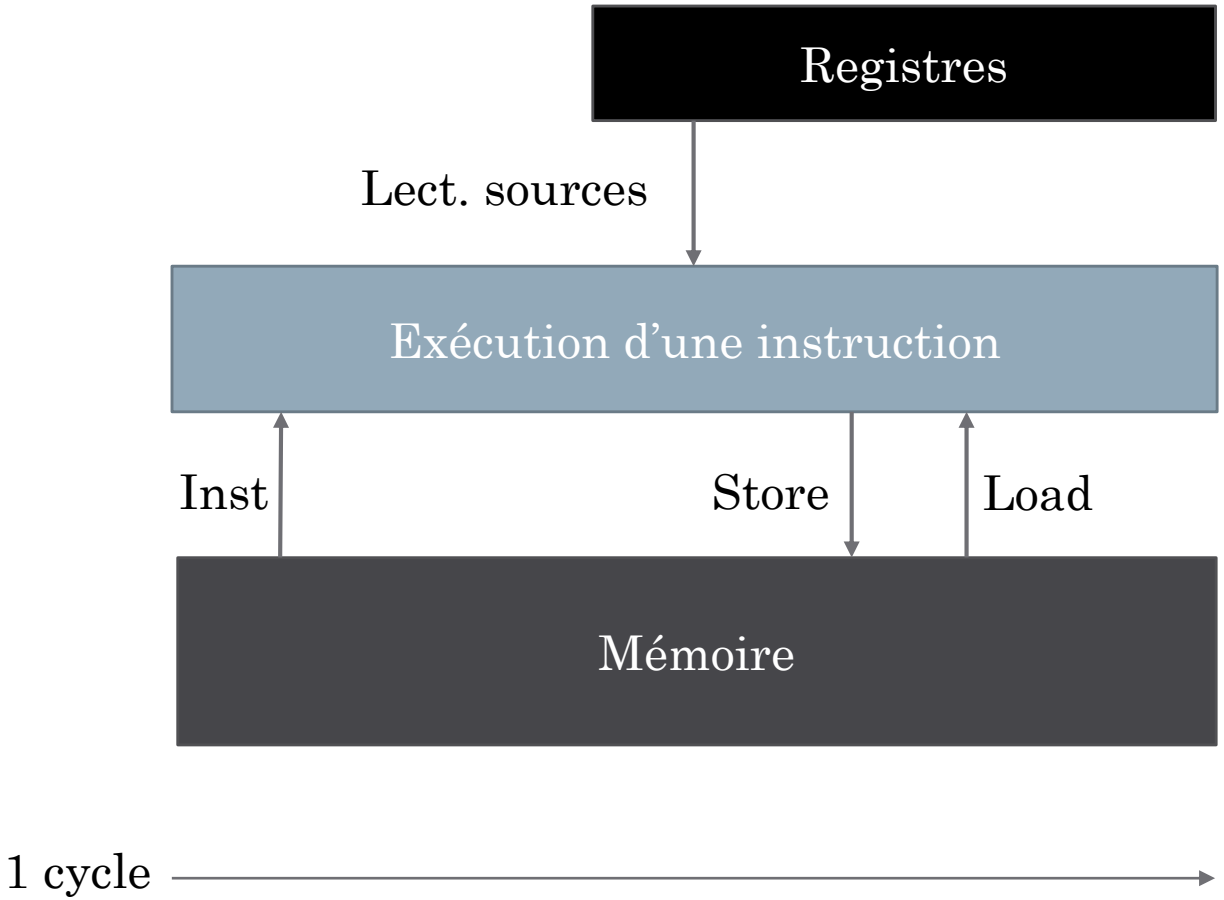
Exécuter une instruction ?



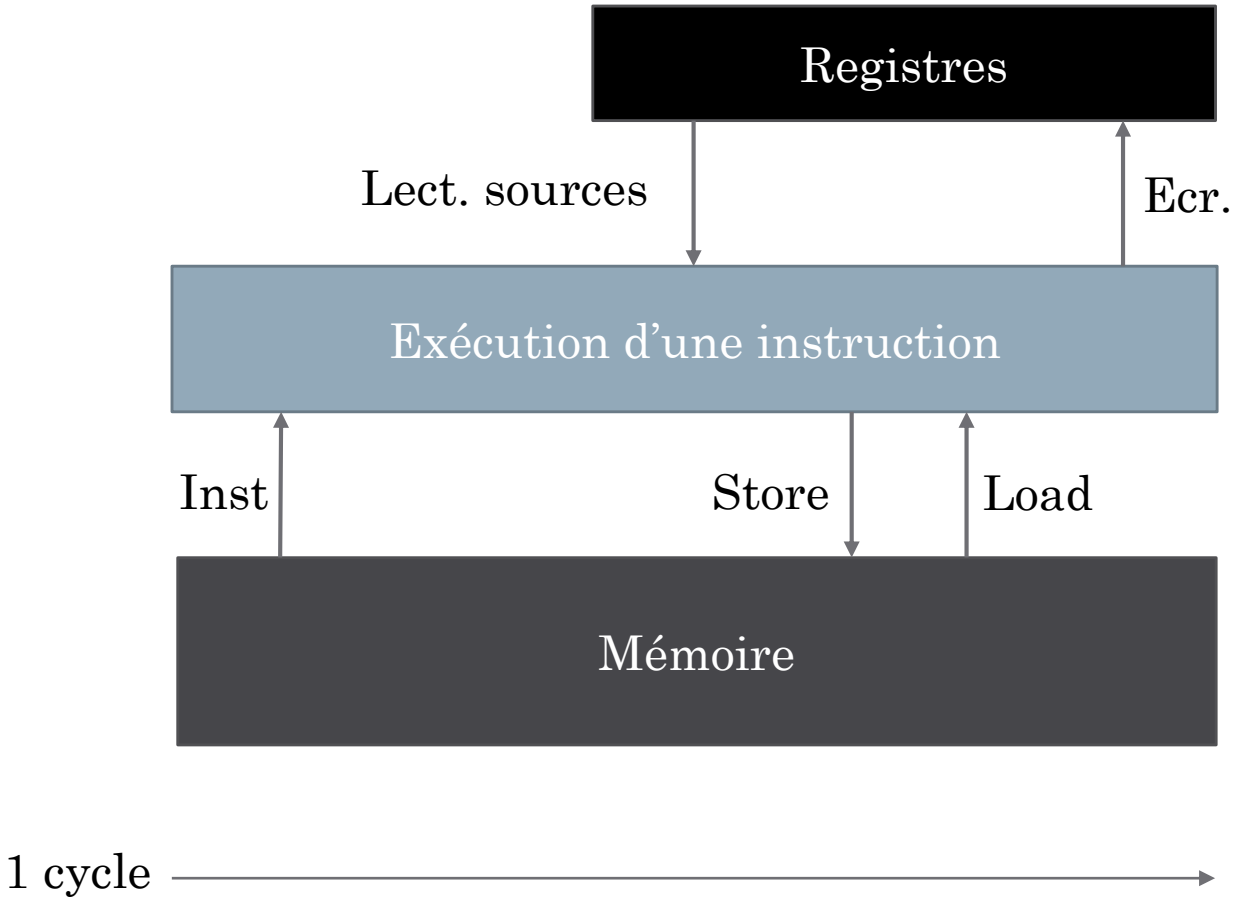
Exécuter une instruction ?



Exécuter une instruction ?



Exécuter une instruction ?

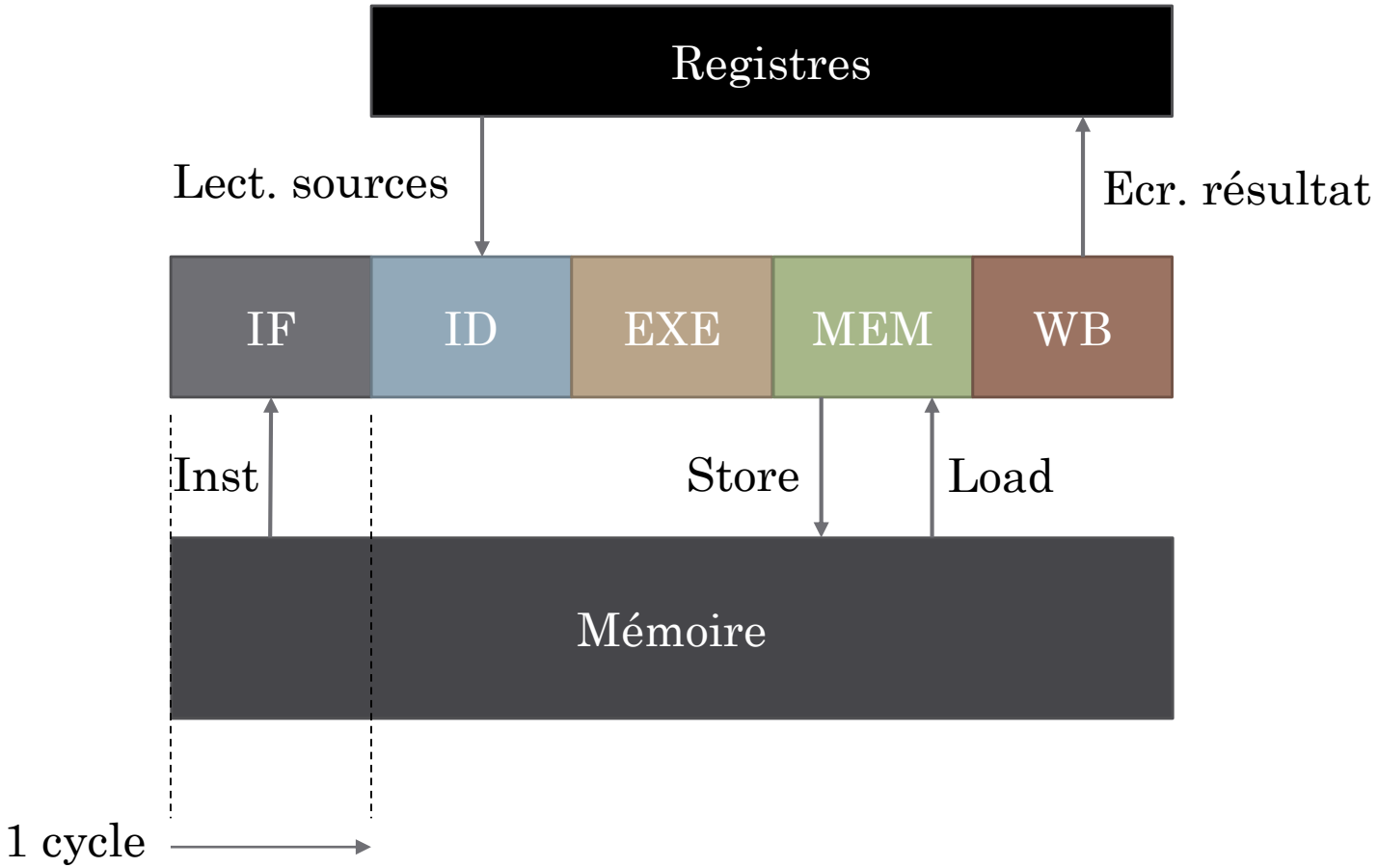


Latence : 1 Cycle Par Instruction
Débit : 1 Instruction Par Cycle

Pipelining

- Exécuter une instruction requiert plusieurs étapes indépendantes :
  - Lire l'instruction depuis la mémoire
  - Décoder pour générer le contrôle & lire les opérandes
  - Calcul du résultat
  - Accès mémoire si nécessaire
  - Ecrire le résultat dans un registre

Pipelining


Registres
Lect. sources
Ecr. résultat
IF
ID
EXE
MEM
WB
Inst
Store
Load
Mémoire
1 cycle

Pipelining

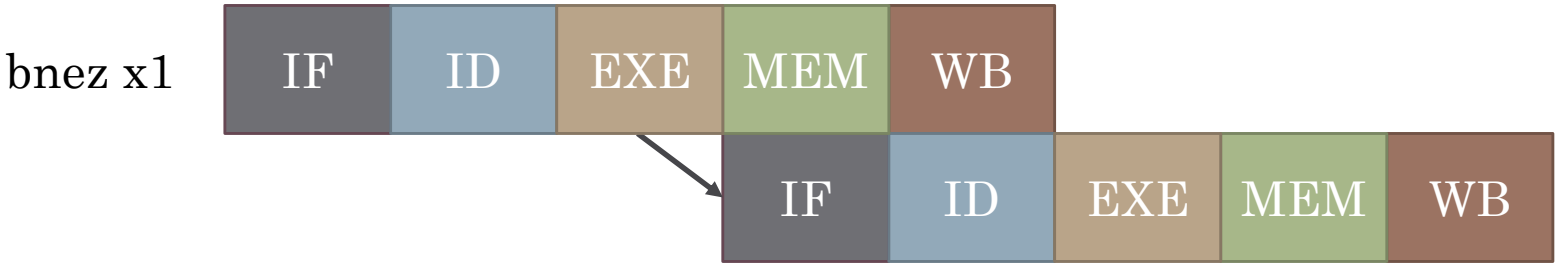
- Diviser le processeur en étages

Latence : 5 Cycles Par Instruction
Débit : 1 Instruction Par Cycle

- Avantages
  - Moins de « travail » par cycle, on peut augmenter la fréquence
- Inconvénients?

Spéculation

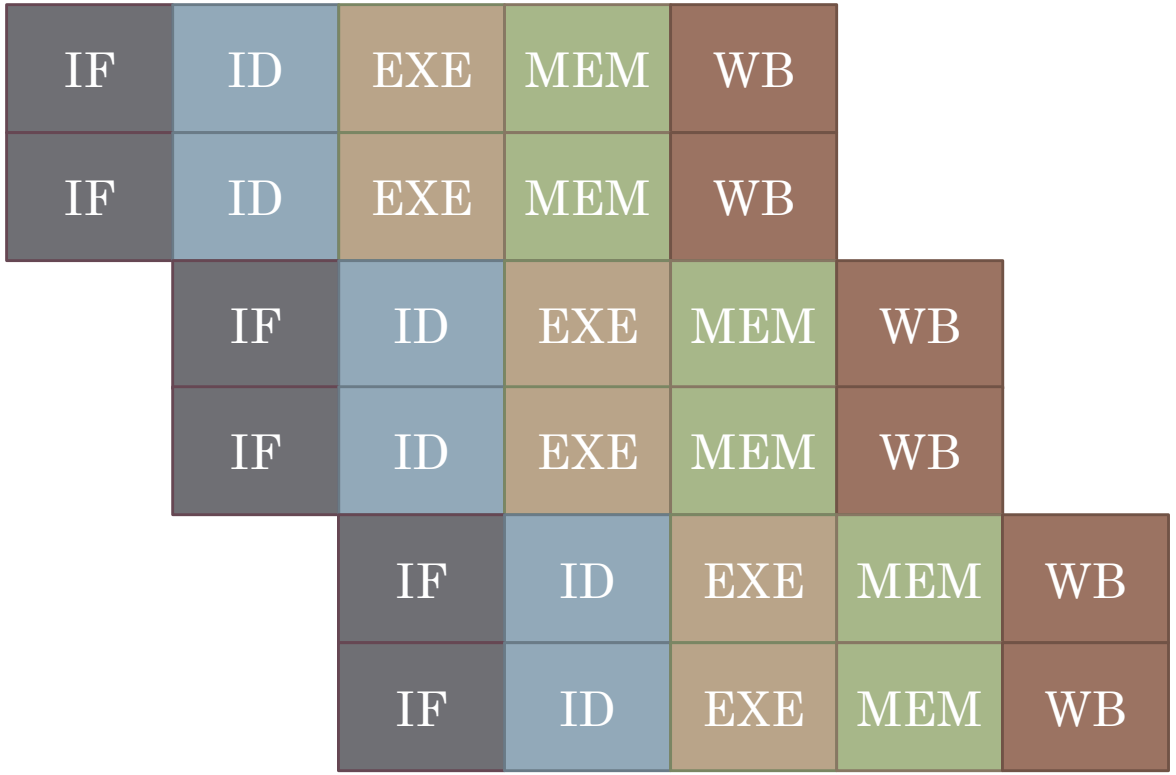
- L'efficacité du pipelining décroît à mesure qu'il s'allonge
  - **Délai de résolution des branchements conditionnels (if, else, boucles…)**



  - Direction de bnez connue au début du cycle 3, mais nécessaire au cycle 1 (dépendance de **contrôle**)
    - Prédire la direction pour garder le pipeline rempli

Exécution superscalaire

- Dupliquer les ressources *n* fois, traiter *n* instrs/cycle/étage (au mieux)



Latence : 5 CPI
Débit : **2 IPC**

Exécution superscalaire

- Les dépendances de données entre instructions peuvent causer des bulles

- IFetch/Idecode en parallèle, mais I1 a besoin du résultat de I0

Ordonnancement dynamique

- Hypothèse : Superscalaire degré 2

- I4 et I5 sont bloquées, mais est-ce nécéssaire ?

Ordonnancement dynamique

- Idée : Autoriser l'exécution des instructions dans le désordre

I0 : add x2, x1, x1

I1 : add x3, x2, x2

I2 : add x4, x3, x3

I3 : add x5, x4, x4

I4 : add x6, x8, x8

I5 : add x7, x9, x9

Pipelining, spéculation, exécution superscalaires, ordonnancement dynamique

Les processeurs généralistes modernes orchestrent ces différentes techniques ***microarchitecturales*** pour maximiser la performance