Le PIPELINE

Henri-Pierre Charles & Frédéric Rousseau

Motivations et principes de base

Motivation

- Les performances d'un microprocesseur simple ne sont pas très bonnes
- Technique qui n'est pas nouvelle : les premières machines commerciales datent de 1964 avec l'IBM 360/91

idée

- Le pipeline est une technique de réalisation dans laquelle plusieurs instructions se supersposent pendant l'exécution

Difficultés

- Ce recouvrement est source de nombreuses difficultés dans l'exécution des instructions

Idée de fonctionnement

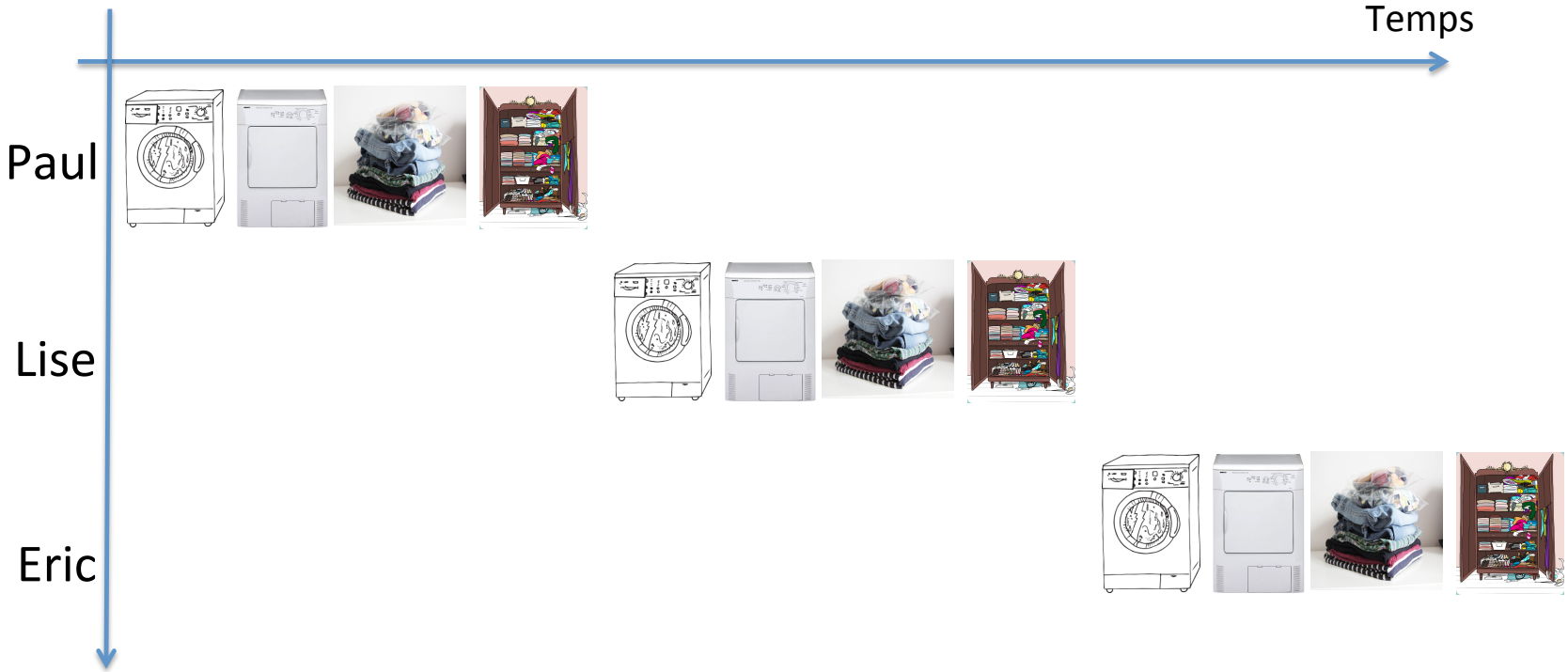
Principe appliqué aux processeurs

- Les activités au sein de l'unité centrale sont organisées comme une chaîne de montage : on décompose l'unité de commande chargée de la **lecture de instructions**, de leur **décodage**, de leur **exécution**, ... en plusieurs modules fonctionnels.
- Ces **modules travaillent en parallèle**
- On commence ainsi le traitement d'une nouvelle instruction avant que la précédente ne soit terminée, ce qui crée un flot continu en entrée de chacun des modules.

Exemple avec une laverie

Analogie avec le fonctionnement à la laverie

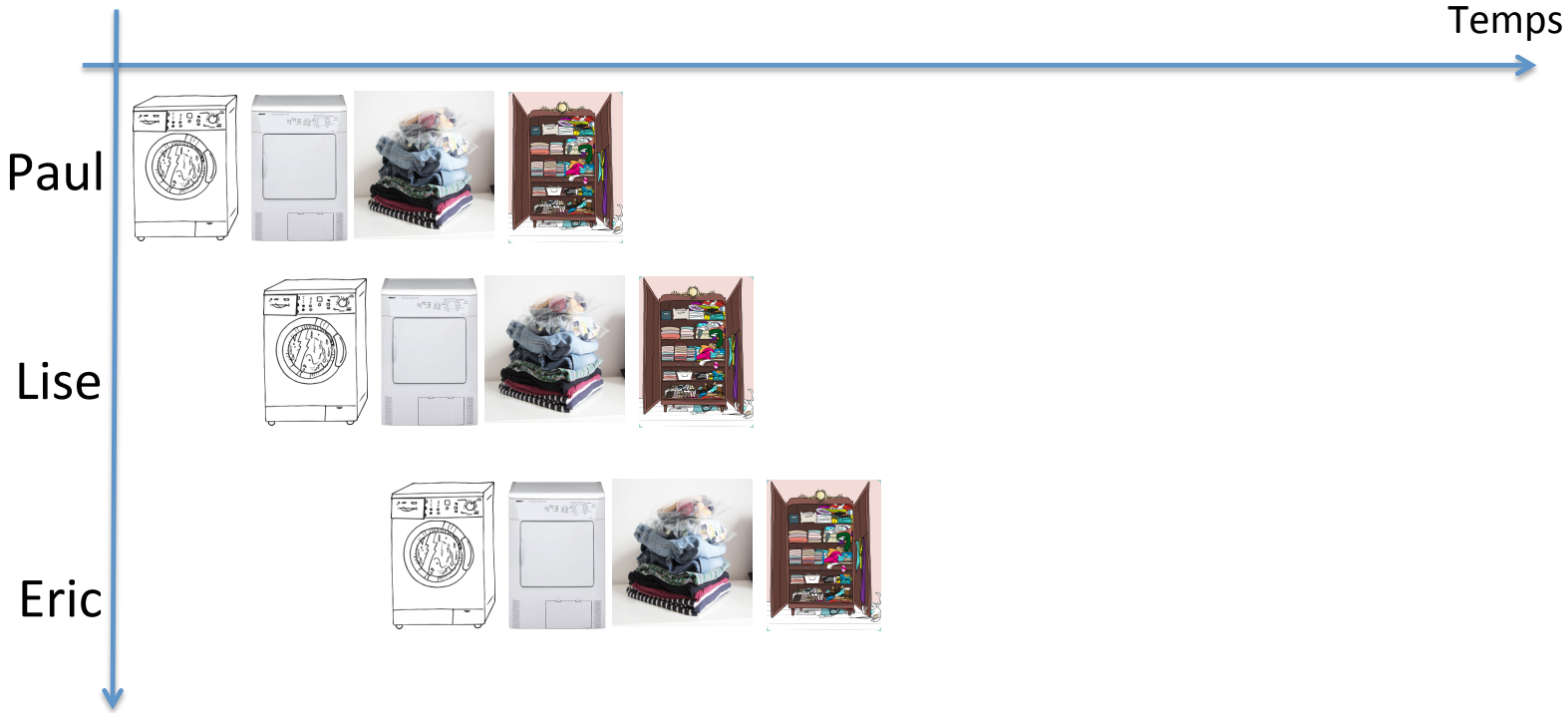
Sans pipeline



Exemple avec une laverie

Analogie avec le fonctionnement à la laverie

Avec pipeline



Intérêt - laverie

Laverie sans pipeline - 1h pour chaque action

- Pour exécuter 1 séquence lavage-séchage-pliage-rangement, il faut 4h.
- Si Paul commence à 8h, Eric termine à 20h ...

Laverie avec pipeline - 1h pour chaque action

- Pour exécuter 1 séquence lavage-séchage-pliage-rangement, il faut 4h.
- Si Paul commence à 8h, Eric termine à 14h ...

Application aux microprocesseurs - pipeline à 5 étages

On considère 5 étapes pour l'exécution d'une instruction (MIPS)

- Lecture en mémoire de l'instruction : Inst. Fetch - **IF**
- Décodage de l'instruction et lecture des registres : Inst. Decode - **DE**
- Exécution de l'instruction (ou calcul d'adresse) : Execution - **EXE**
- Accès à un opérande en mémoire : **MEM**
- Ecriture du résultat dans un registre : Write Back - **WB**

	Cycle								
instruction	1	2	3	4	5	6	7	8	9
i0	IF	DE	EXE	MEM	WB				
i1		IF	DE	EXE	MEM	WB			
i2			IF	DE	EXE	MEM	WB		
i3				IF	DE	EXE	MEM	WB	
i4					IF	DE	EXE	MEM	WB

Intérêt - Application aux microprocesseurs

Architecture sans pipeline - 1 cycle = 1 ns pour chaque étage

- Pour exécuter 10 instructions, il faut 10 cycles soit 10 ns
- Pour éxécuter m instructions, il faut m cycles

Architecture avec pipeline - 5 étages

- 1 instruction en 1 ns, soit 0,2 ns si tous les étages ont le même temps de traitement
- Pour exécuter 10 instructions, il faut 14 étapes soit 2,8 ns

Généralisation pour une architecture pipeline à n étages

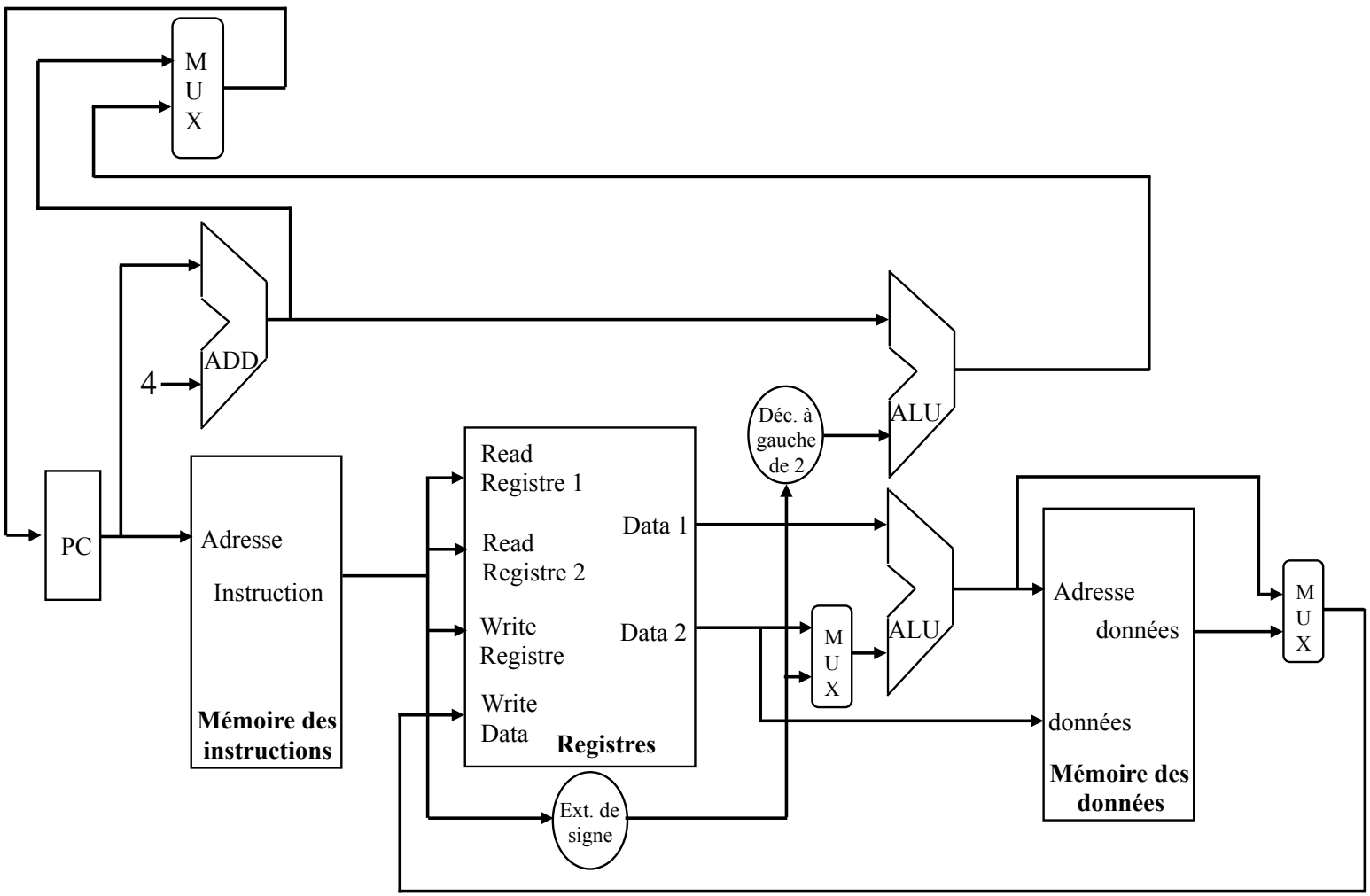
- Si m le nombre d'instructions est très grand, le temps d'exécution est $\frac{m+n-1}{n} = \frac{m}{n}$

Amélioration de performances

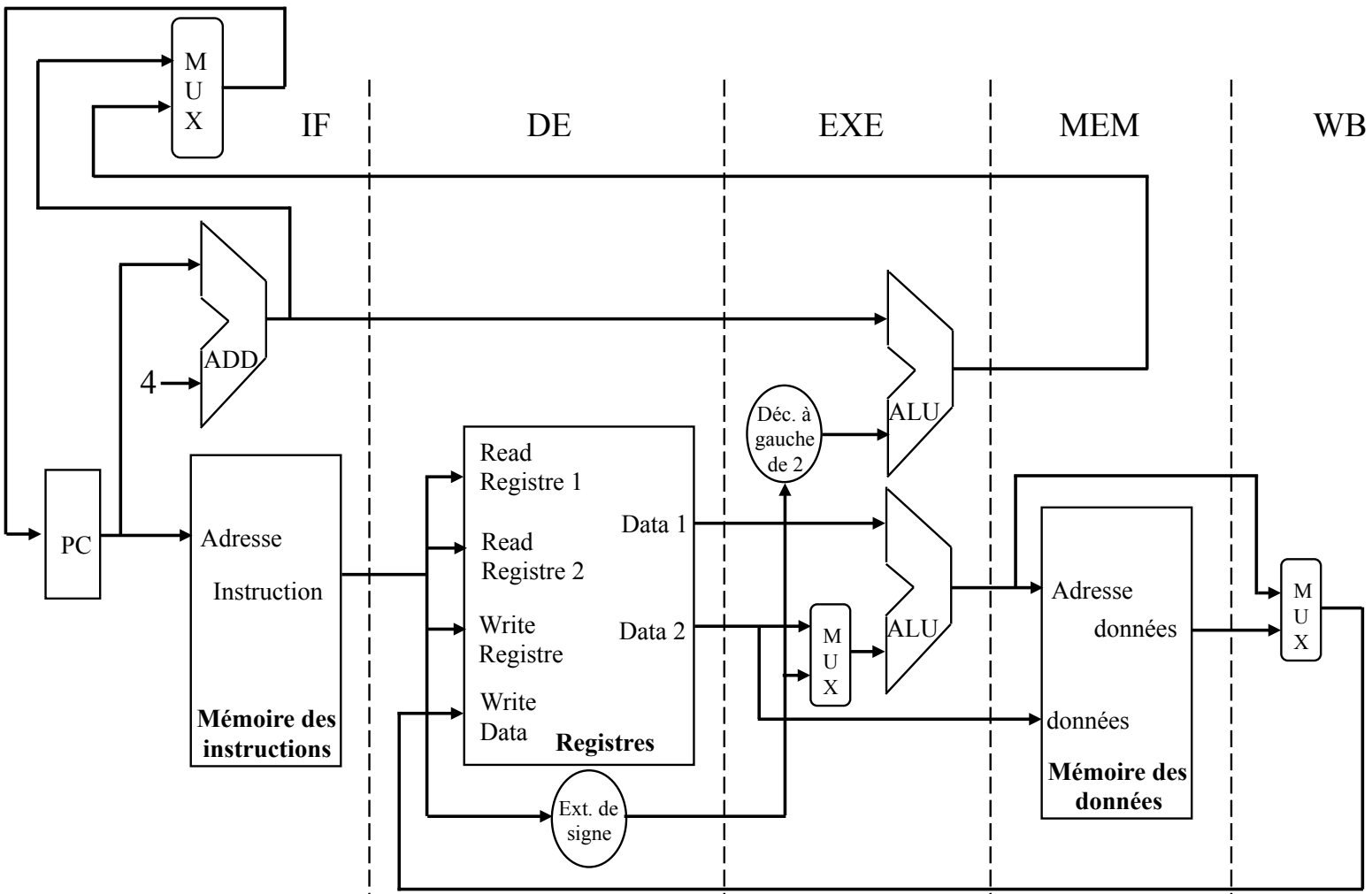
A retenir

- La technique du pipeline améliore les performances en *augmentant le débit de sortie des instructions, plutôt qu'en diminuant le temps d'exécution d'une instruction*.
- Le débit des instructions est la bonne *métrique* à considérer car les programmes réels exécutent des milliards d'instructions.

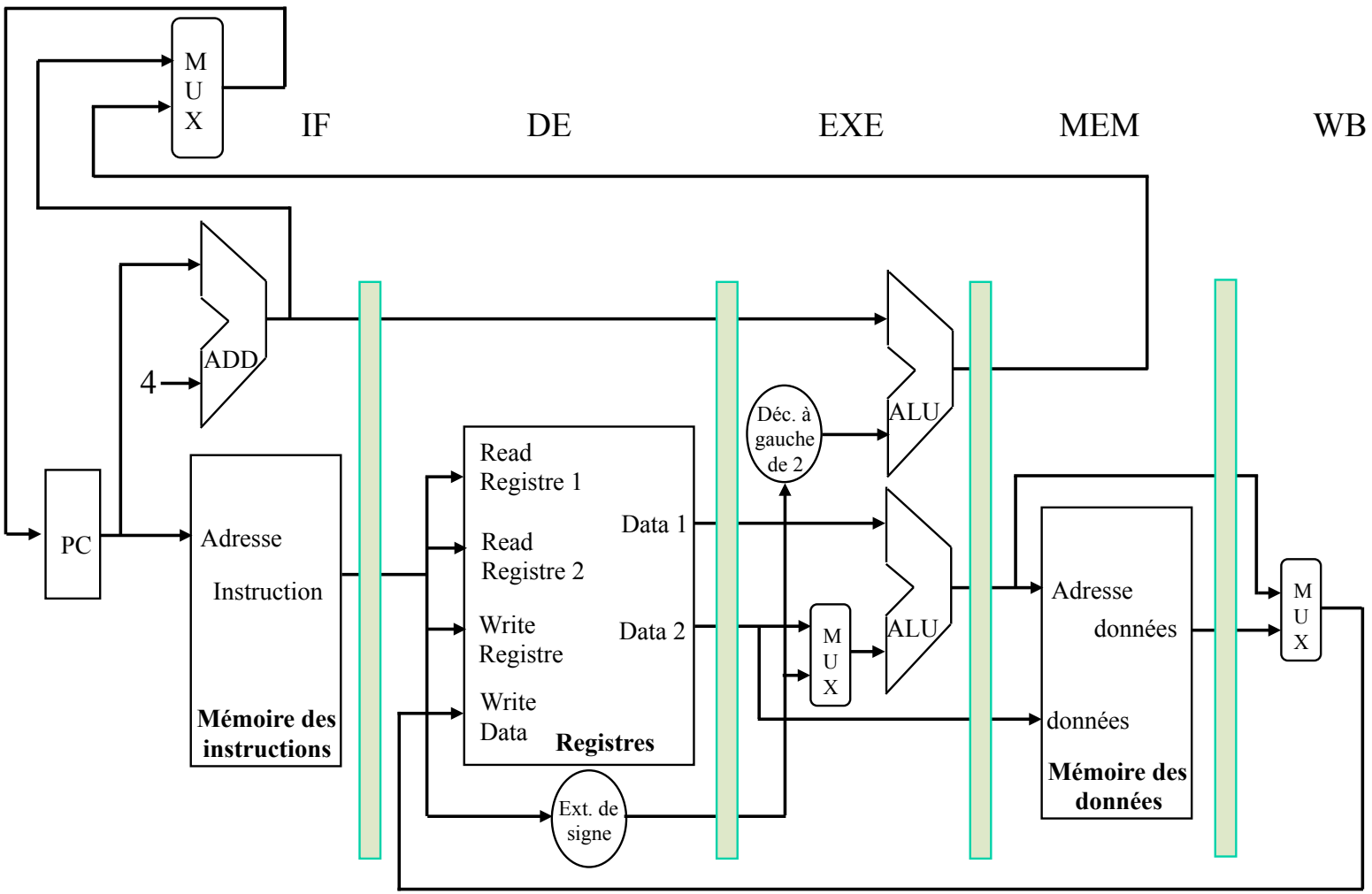
Retour sur l'architecture d'un processeur RISC - MIPS



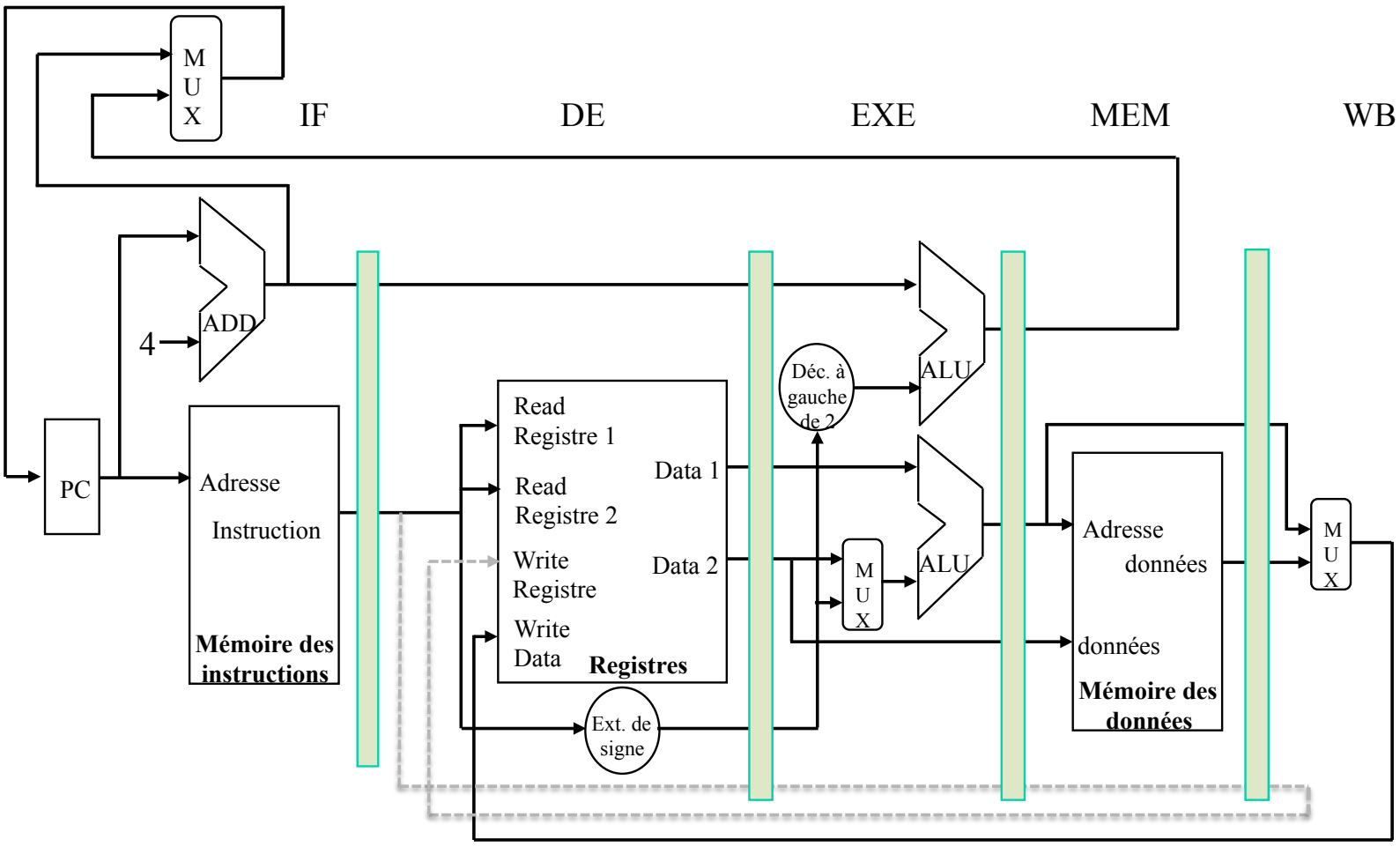
Découpage par étage d'un processeur RISC - MIPS



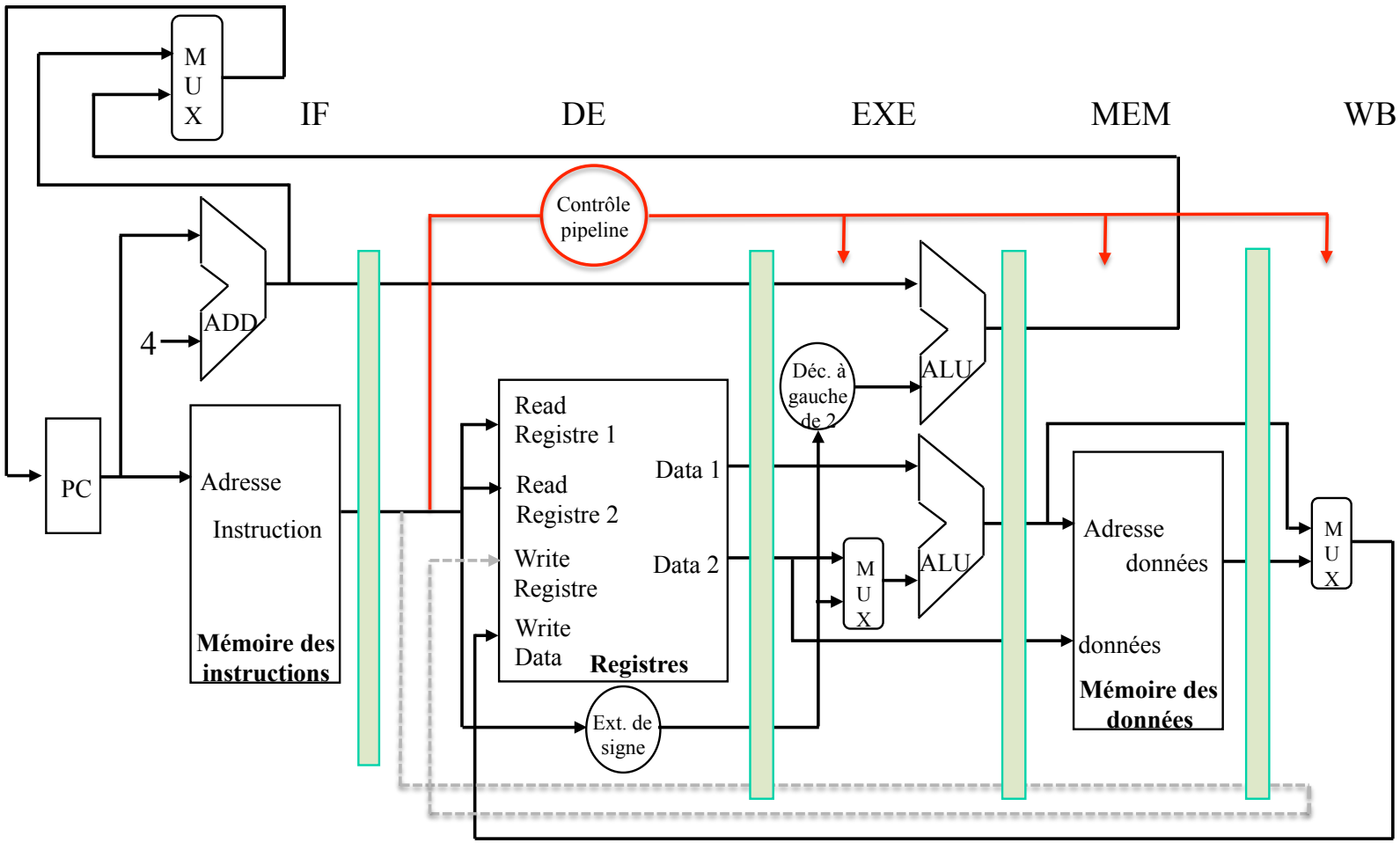
Architecture d'un processeur RISC - MIPS - Avec pipeline



Modification de l'architecture - registre d'écriture décalé



Modification de l'architecture - Contrôle du pipeline



Difficultés d'utilisation du pipeline

Les principaux problèmes rencontrés sont liés :

- aux *accès à la mémoire*
- aux conflits de *dépendance entre registres*
- aux *branchements* et sauts
- au *traitement des interruptions et exceptions*

Il existe 2 façons de résoudre ce problèmes

- Par l'ajout de matériel (connexion ou logique)
- Par le logiciel (le compilateur est en charge d'ajouter des instructions NOP ou de permuter des instructions)

Conséquence

- La programmation en langage d'assemblage devient trop difficile -> utilisation du C et d'un compilateur

Aléas de pipeline

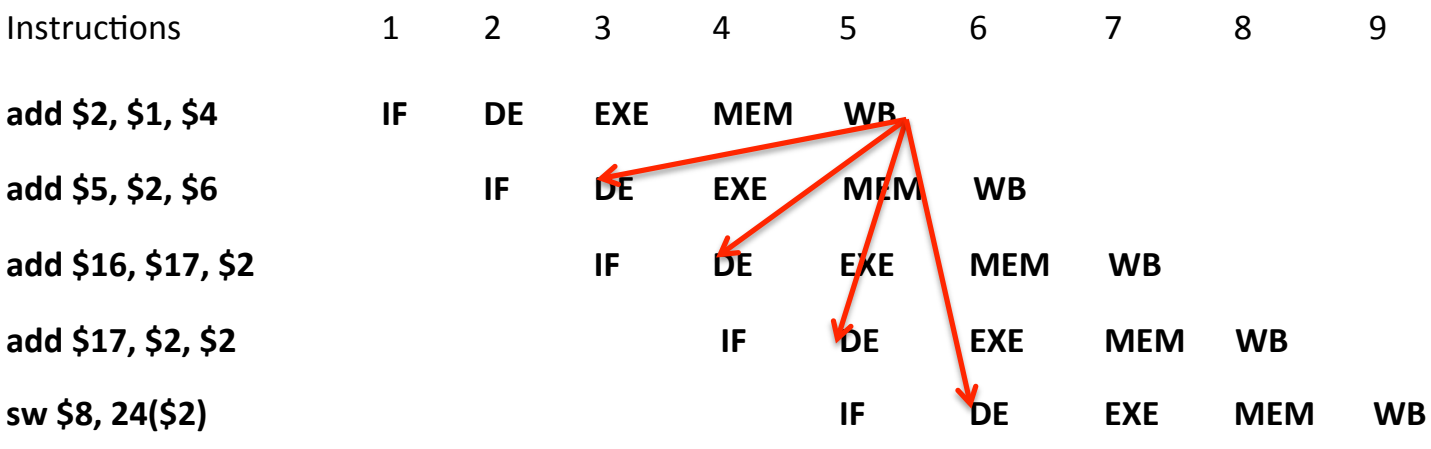
On ne peut pas toujours exécuter l'instruction suivante au cycle d'horloge suivant :

- Aléas de données (dépendances de données)
  - Instruction ayant besoin du résultat de l'instruction précédente
- Aléas de contrôle (branchements)
  - La prochaine instruction à exécuter n'est connue qu'à la fin du traitement - condition de branchement
- Aléas de contrôle (interruption ou exception)
  - Les interruptions ou exceptions viennent modifier le cours de l'exécution du programme.
- Aléas structurels (conflits de ressources)
  - Accès au même composant au même cycle (lecture-écriture dans le même registre par exemple)

Aléas de donnée

Considérons un exemple de 5 instructions ci-dessous

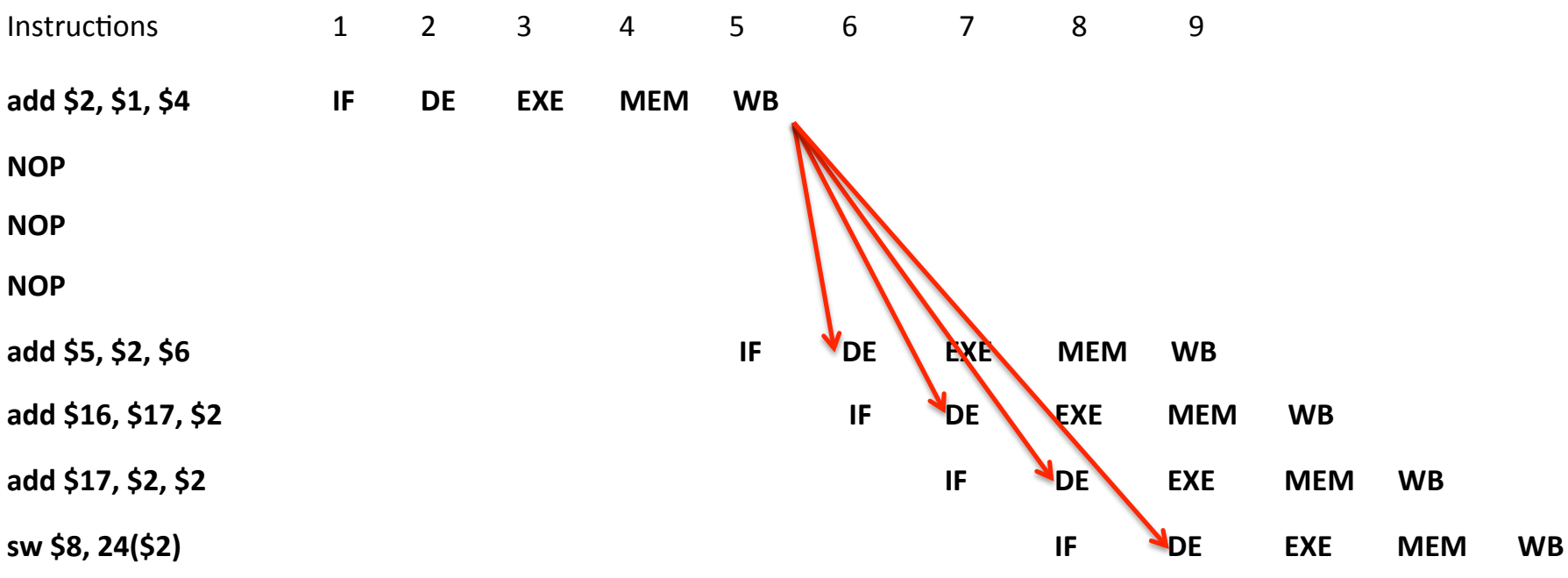
- Les 4 dernières instructions sont toutes dépendantes du résultat que la première instruction produit.
- Les valeurs lues dans le registre $2 ne sont pas les bonnes pour les 3 instructions add.



Aléas de donnée - Une solution logicielle

Insertion d'instructions indépendantes

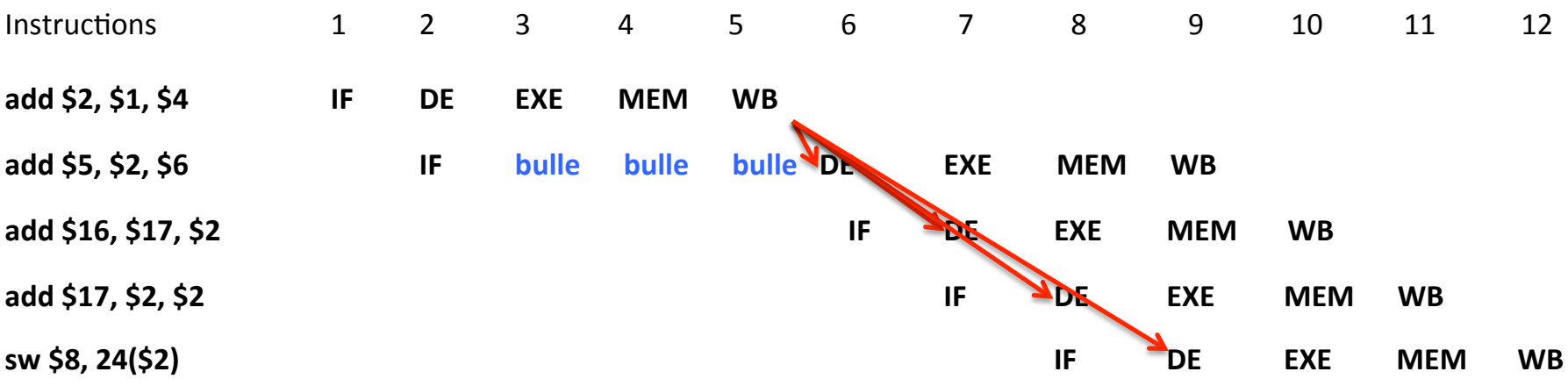
- On insère des NOP (No OPeration)
- Le code ci-dessous est fonctionnel, mais plutôt inefficace avec ces 3 instructions NOP



Aléas de donnée - Les solutions matérielles

La suspension

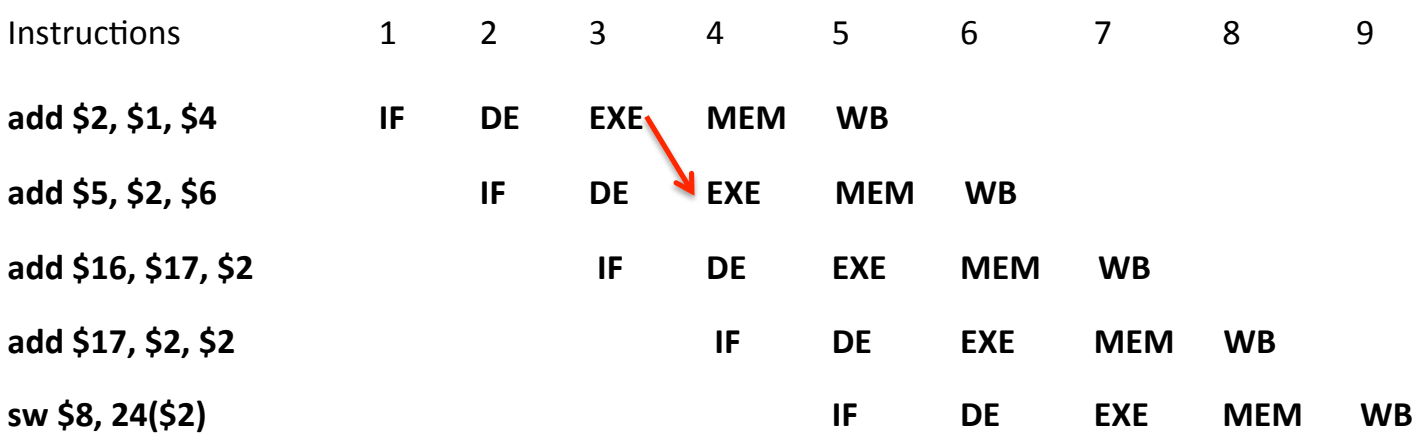
- On suspend les instructions dans le pipeline jusqu'à ce que l'aléa soit résolu
- On parle d'insertion de *bulle(s)*
- On suspend l'étape de décodage DE et on empêche de lire une nouvelle instruction



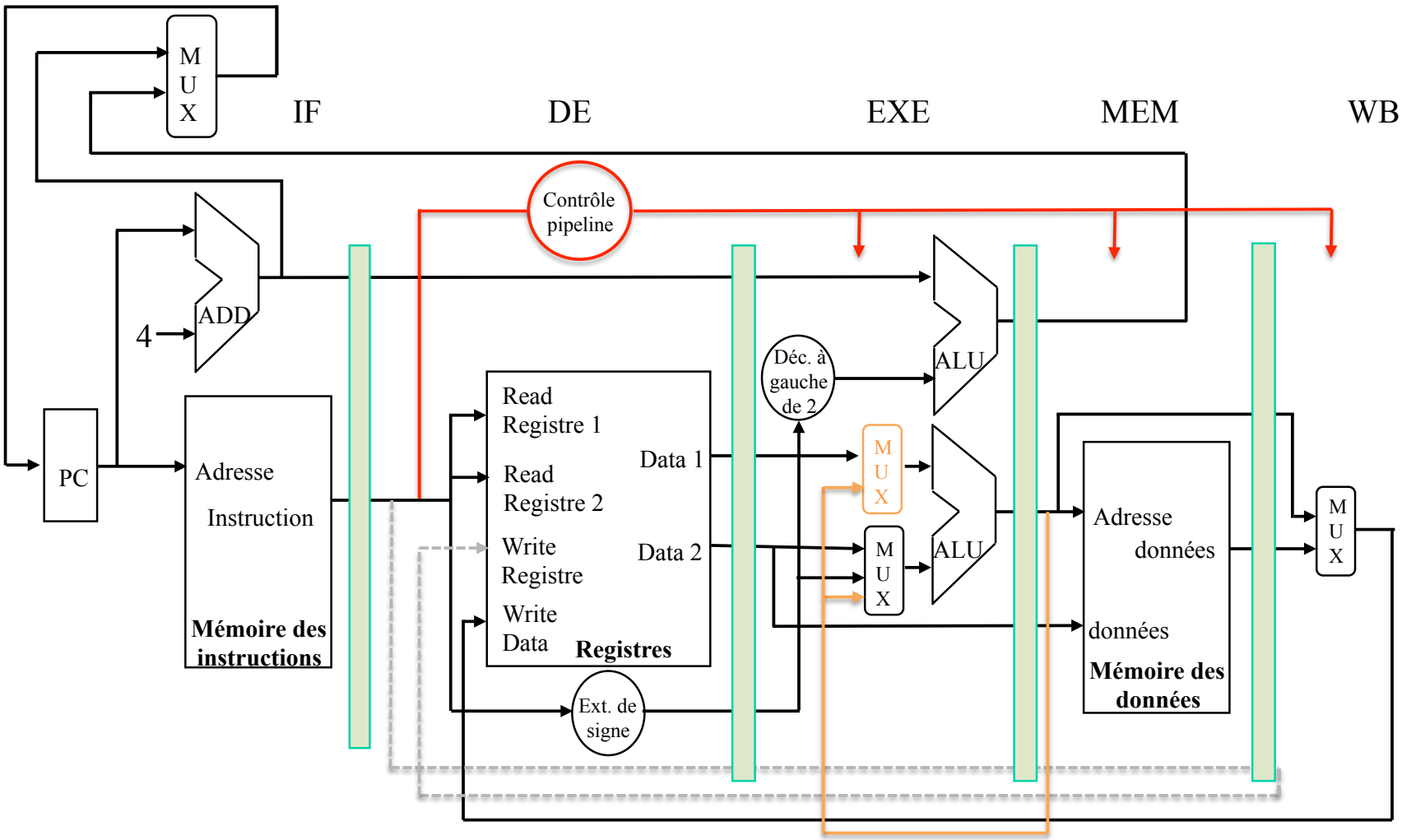
Aléas de donnée - Les solutions matérielles

L'envoi

- On constate sur l'exemple ci-dessous que la donnée $2 est disponible à la sortie de EXE, et utilisée à l'entrée de EXE de l'instruction suivante.
- On peut utiliser les résultats temporaires (disponibles dans les registres du pipeline) en entrée de l'ALU (EXE). Le pipeline peut alors s'exécuter sans suspension.



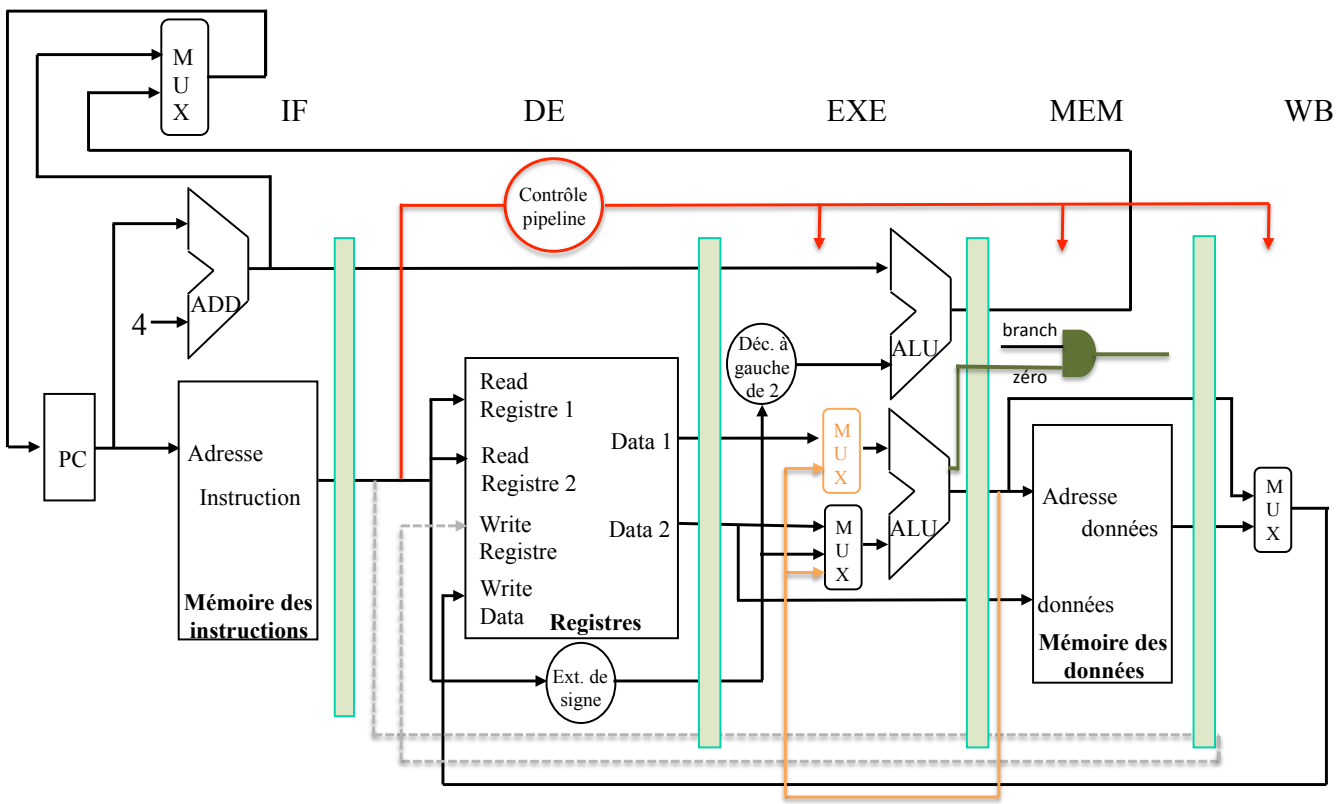
Aléas de donnée - architecture pour l'envoi



Aléas de branchements

Hypothèse

- La décision de branchement est prise à l'étage MEM (calcul de la condition ET pour valider ou non le branchement).



Aléas de branchements par l'exemple

Description du problème

- Quand la décision de branchement est prise, les 3 instructions suivantes sont dans le pipeline et s'exécuteront.

Instructions	1	2	3	4	5	6	7	8	9
beq $1, $3, Exit	**IF**	**DE**	**EXE**	**MEM**	**WB**				
add $5, $2, $6		**IF**	**DE**	**EXE**	**MEM**	**WB**			
add $16, $17, $2			**IF**	**DE**	**EXE**	**MEM**	**WB**		
add $17, $2, $2				**IF**	**DE**	**EXE**	**MEM**	**WB**	
.....									
Exit: sw $8, 24($2)					**IF**	**DE**	**EXE**	**MEM**	**WB**

Aléas de branchement - Les solutions

Quelques remarques

- Les aléas de contrôle (ou branchements) ont lieu moins souvent que les alées de données.
- Les solutions sont des méthodes simples car il n'existe rien d'aussi efficace que l'envoi pour les aléas de données.

La suspension

- On peut suspendre jusqu'à ce que le branchement soit achevé, mais avec une pénalité de plusieurs cycles d'horloge.

Prédiction de branchement - simple

- On suppose que le branchement ne sera pas effectué et on exécute les instructions suivantes. Si le branchement a lieu, on annule les effets du résultat des 3 instructions suivantes.

Aléas de contrôle - Interruptions et exceptions

Interruption

- La solution la plus simple est de terminer l'exécution des instructions qui sont dans le pipeline avant de traiter l'interruption.

Exception - add $1, $2, $2

- Supposons que l'instruction add provoque un débordement -> exception
- Il ne faut pas contaminer les registres des instructions suivantes qui sont dans le pipeline. Il faut vider le pipeline avec un méchanisme identique à celui des aléas de branchements. On peut ensuite exécuter la routine d'exception.

Conclusion

Performances

- Les processeurs pipelinés ont de meilleurs performances, en réduisant le temps d'exécution moyen par instruction. Mais les aléas diminuent les performances, même si des solutions existent pour en limiter la pénalité.
- Intel Xeon : 20 étages
- ARM Cortex A8 (processeur iPhone, iPad) : 13 étages

Lien architecture - compilateur

- Un compilateur doit être adapté à l'architecture pipelinée pour des performances optimales.

Exercice

Soit la séquance d'instructions ci-dessous

- Mettre en évidence tous les problèmes de dépendances dans ce code
- Dessiner les à l'aide sur la figure ci-dessous
- Quelles sont les aléas qui peuvent être résolus pas du matériel ? quels sont ceux qui ne le peuvent pas ?
- Comment gérer les dépendances qui ne le sont pas par du matériel ? Quel est alors le résultat sur le code ci-dessous ?

Instructions	1	2	3	4	5	6	7	8	9
addi $2, $4, 100	**IF**	**DE**	**EXE**	**MEM**	**WB**				
lw $4, 40($2)		**IF**	**DE**	**EXE**	**MEM**	**WB**			
subu $5, $4, $2			**IF**	**DE**	**EXE**	**MEM**	**WB**		
sw $5, 20($4)				**IF**	**DE**	**EXE**	**MEM**	**WB**	
beq $5, $2, Exit					**IF**	**DE**	**EXE**	**MEM**	**WB**