5- Pipeline

Hypothèse précédente

- Une instruction = un cycle

=> Le cycle doit correspondre au temps d'exécution de l'instruction la plus lente

Temps d'exécution du programme

Temps d'exécution = Σ cycles par instruction / fréquence d'exécution

Quelle différence de temps d'exécution entre instructions ?

5- Pipeline

Chaque instruction a une **latence**

- Temps de traversée des unités:
  - ALU: 2ns
  - Instruction memory: 2ns
  - Register Bank: 1ns
  - Data memory: 2ns

5- Pipeline

Temps de quelques instructions:

- Instruction arithmétique: instruction memory (2ns), register read (1ns), ALU (2ns), register write (1ns) = **6ns**
- Instruction accès mémoire **mov (store)**: instruction memory (2ns), register read (1ns), ALU (2ns), data memory (2ns), register write (1ns) = **8ns**
- Instruction j**eq**: instruction memory (2ns), ALU (2ns) = **5ns**

5- Pipeline

Comment améliorer efficacité ?

- ALU utilisée 33% du temps sur instruction arithmétique, 25% du temps sur **mov**, idem autres unités
- Principe du pipeline:
  - Segmenter l'exécution d'une instruction en plus petites étapes qui se suivent, prenant chacune 1 cycle

    => exécution multicycles
  - Dès qu'on a passé la première étape d'exécution d'une instruction , on peut lancer la première étape d'exécution de l'instruction suivante

Principe est valable pour toute activité: utilisé pour la construction automobile (chaines de montage), dans la fabrication de sandwichs, ...

5-a Pipeline: exemple

Etapes de fabrication de sandwichs

- Prise de l'ordre de commande (ORD), 8s
- Cuisson, toast du steak éventuel (TOS), entre 0 et 10s
- Ajout éventuel de frites (ADD), entre 0 et 10s
- Paiement de la commande (PAY), 5s

Un sandwich = entre 13s et 33s de fabrication

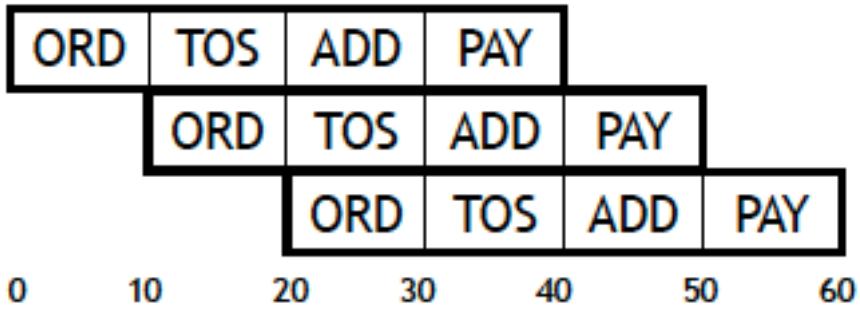
Pour aller plus vite:

- Mettre plusieurs chaines de fabrication en **parallèle** (duplique les unités de fabrication, ex. caisses, plaques de cuisson, ...)
- **Pipeliner** la fabrication

5-a Pipeline: exemple

En pipelinant:

- Un sandwitch toutes les 10s (diminue latence fabrication sandwich)
- Utilisation plus efficace de chacune des unités existantes



Contraintes:

- Tous les sandwichs passent par toutes les étapes (même si inutile)
- Chaque étape doit prendre le même temps (= l'étape la plus longue, 10s)

*Pour les pipelines en architecture, une étape s'appelle un **étage** (stage)*

5-b Pipeline du chemin d'exécution

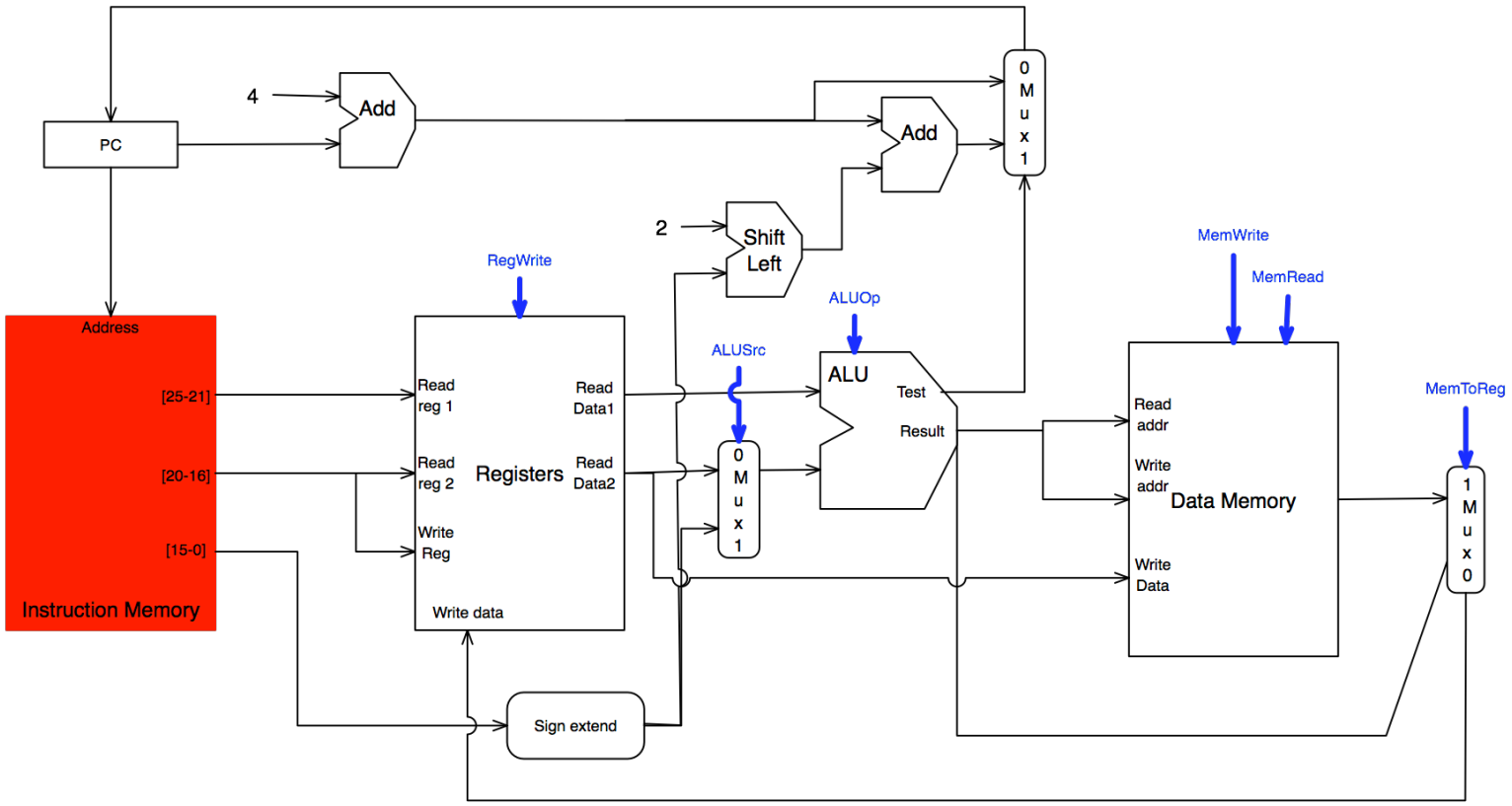
Découpage possible de l'exécution MIPS en étages de pipeline:

- Longueur du plus long étage: 2ns

Etage	Abbréviation	Description
Chercher l'instruction	IF	Incrémenter PC, lire instruction de la mémoire
Décoder l'instruction	ID	Lecture des registres sources et génération signaux de contrôle
Exécute	EX	Faire le calcul (ALU)
Mémoire	Mem	Faire les accès mémoire de données
Stockage résultat	WB	Ecriture dans registre destination

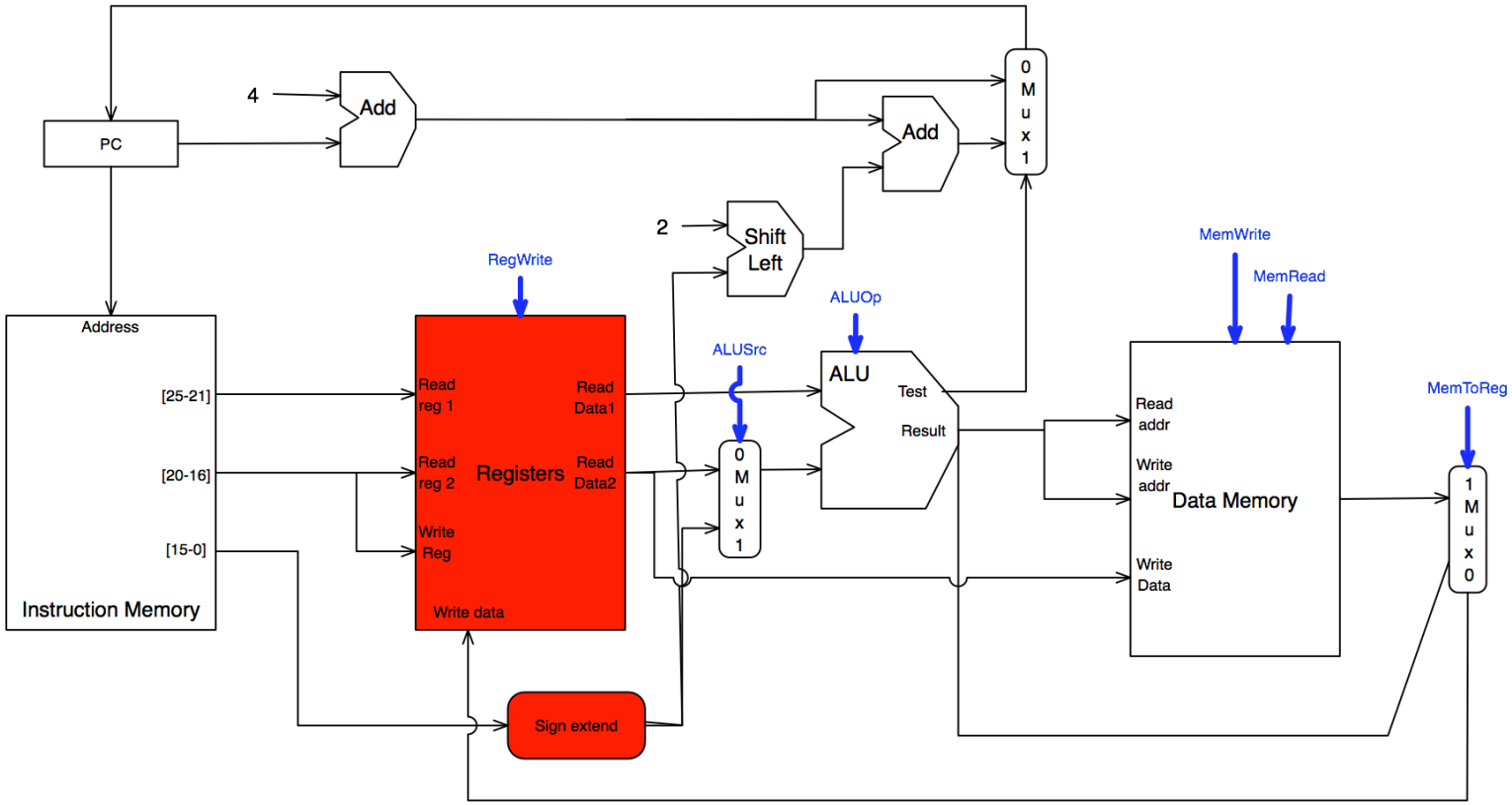
5-b Etages du pipeline

- **Chercher l'instruction (IF)**



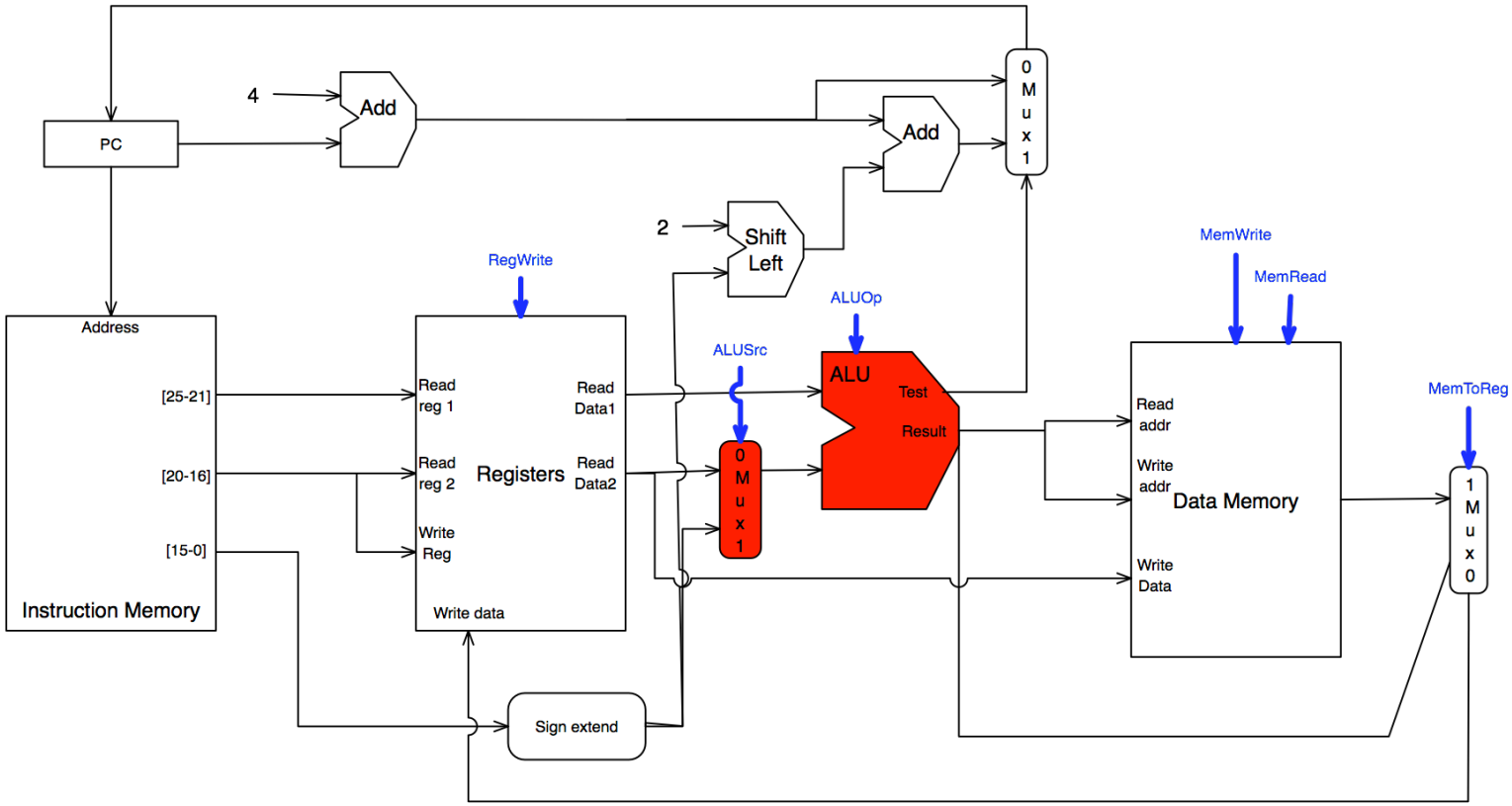
5-b Etages du pipeline

- **Décoder l'instruction (ID)**



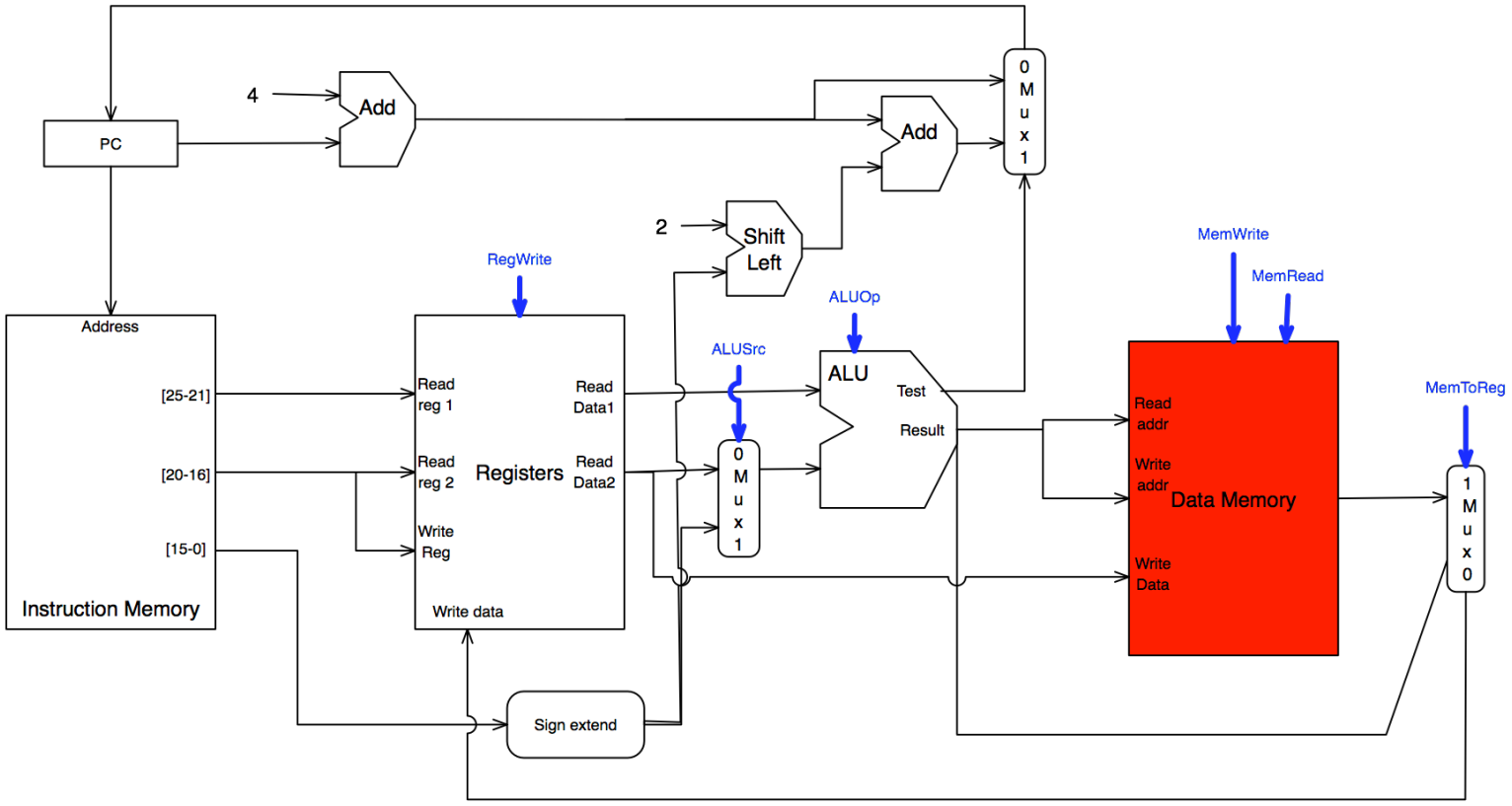
5-b Etages du pipeline

- **Exécuter, calculer avec l'ALU (EX)**



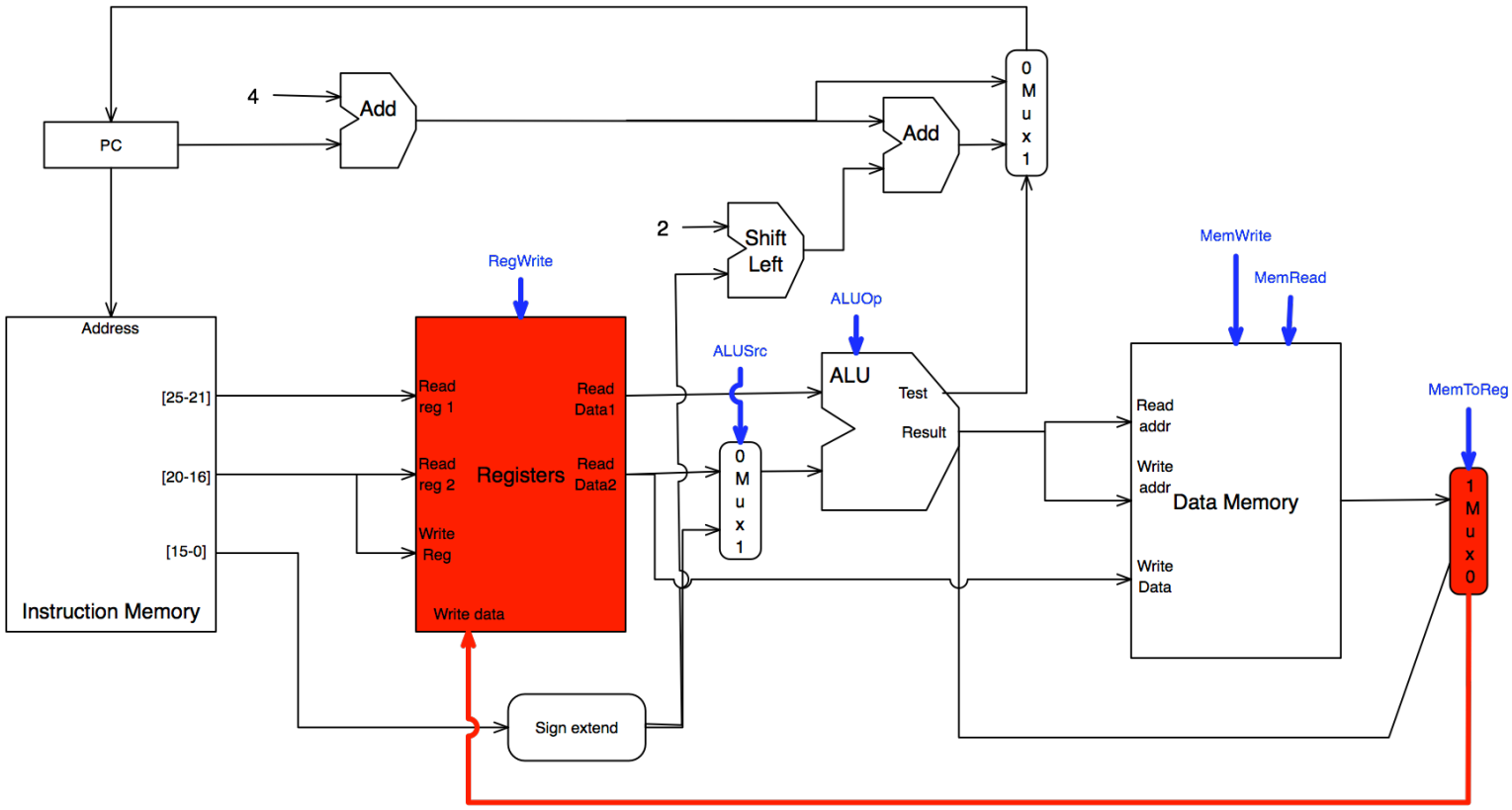
5-b Etages du pipeline

- **Accès mémoire (Mem)**



5-b Etages du pipeline

- **Ecrire dans le banc de registres (WB, write back)**



5-b Etage du pipeline

Utilisation des étages par chaque instruction:

	Arithmétique, 3 registres ou 2 registres et imm.	Load lw	Store sw	Branchement conditionnel beq	Branchement inconditionnel j
IF	Lit instruction, incrémente PC	Lit instruction, incrémente PC	Lit instruction, incrémente PC	Lit instruction, incrémente PC	Lit instruction, incrémente PC
ID	Décode registres de données et immédiat	Décode registre adresse et déplacement	Décode registre adresse, donnée et déplacement	Décode registres comparaison et label	Décode adresse cible et met à jour PC
EX	Fait le calcul arithmétique	Calcule l'adresse	Calcule l'adresse	Fait la comparaison	RIEN
Mem	RIEN	Lit la valeur	Ecrit la valeur	RIEN	RIEN
WB	Ecrit la valeur dans le registre destination	Ecrit la valeur dans le registre destination	RIEN	RIEN	RIEN

5-b Etage du pipeline

Unification des temps de chaque étage, utilisation de tous les étages

- Temps d'exécution d'une instruction non pipeliné: ID et WB ne prennent que 1ns

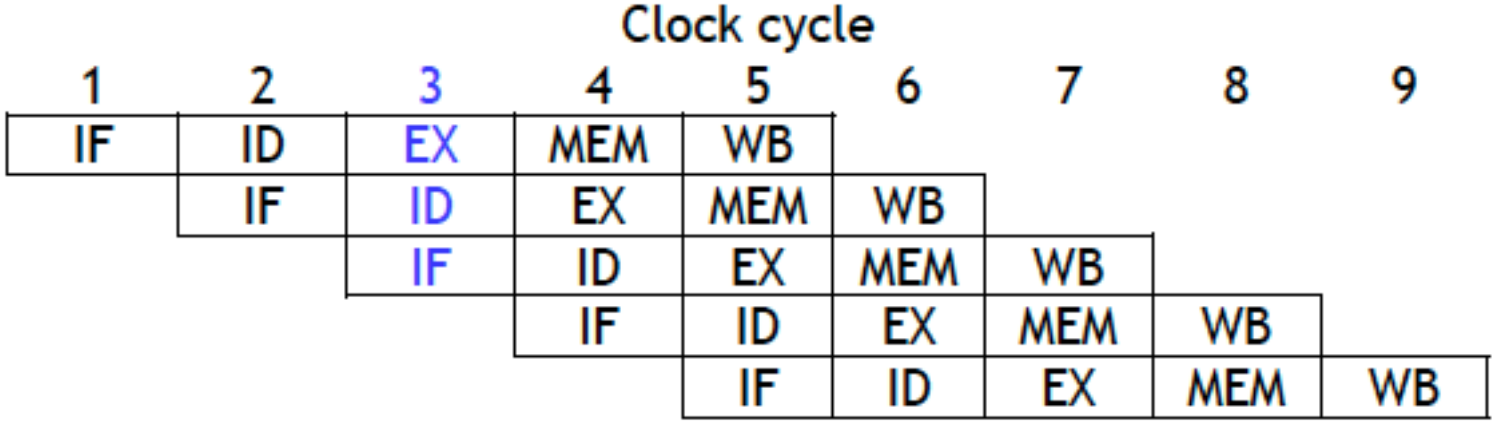
- Temps d'exécution d'une instruction complète, pipelinée: ID et WB prennent 2ns

Mais on lance une nouvelle instruction toutes les 2ns

5-c Exécution pipelinée

- Au maximum, 5 instructions sont en cours d'exécution simultanément
- Exemple de programme et état du pipeline
  - L'exécution des instructions se recouvre

```
add rax,rbx
add rbx, 3
or rax,rbx
and rcx,rdx
mov [rbx],rcx
```

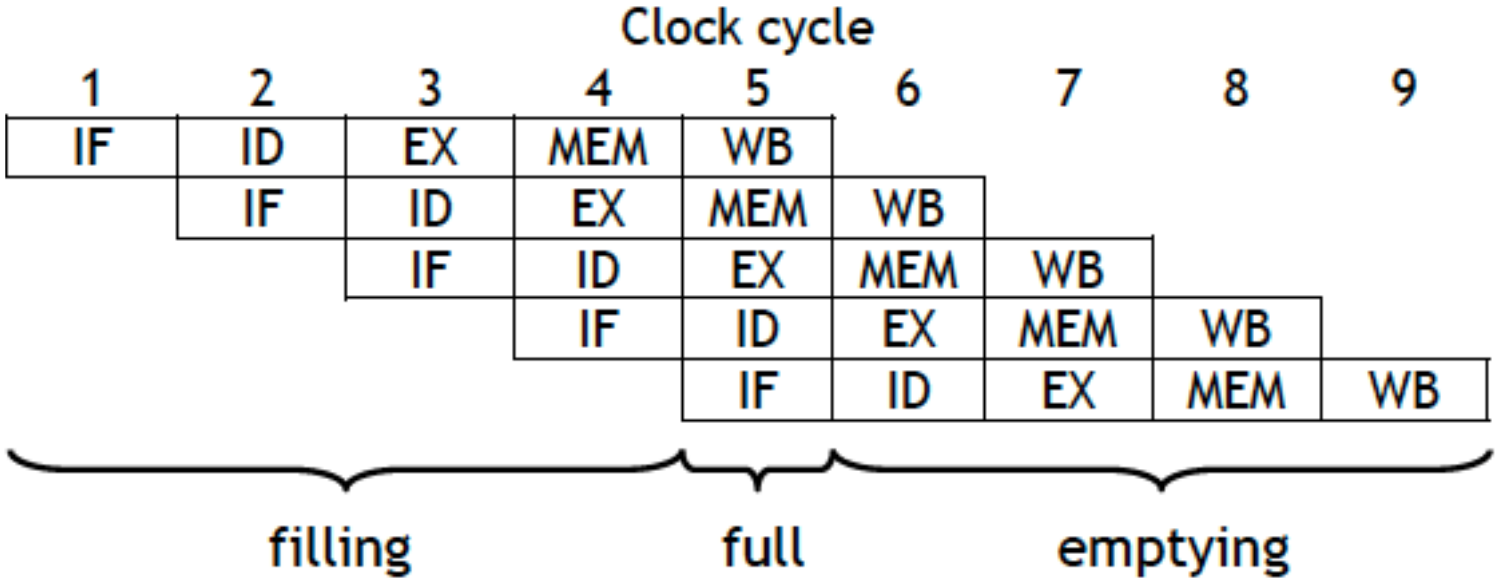


5-c Exécution pipelinée

Phases de l'exécution:

- Remplissage du pipeline: 4 cycles (pour 5 étages)
- Pipeline plein: les 5 étages sont occupés
- Vidage du pipeline: 4 cycles

```
add rax,rbx
add rbx, 3
or rax,rbx
and rcx,rdx
mov [rbx],rcx
```



5-c Exécution pipelinée

Formules générales

Pour un pipeline à *n* étages, 1 cycle/étage

- Vidage et remplissage: *n-1* cycles chacun
- Exécution non pipelinée de p instructions:

$$pn \text{ cycles}$$

- Exécution pipelinée de p instructions:

$$p + n-1 \text{ cycles}$$

Asymptotiquement, si p est grand, on gagne un facteur n de vitesse !!

(tant que le pipeline reste plein)

5-c Exécution pipelinée

Contraintes pour l'exécution:

- Certaines valeurs sont décodées de l'instruction mais utilisées à un étage plus tard
- Il faut éviter qu'elles soient écrasées par les valeurs décodée après
  - Valeur à écrire dans un registre
  - Bits de contrôle des différentes unités

Stage	Control signals needed		
EX	ALUSrc	ALUOp	RegDst
MEM	MemRead	MemWrite	PCSrc
WB	RegWrite	MemToReg	

5-c Exécution pipelinée

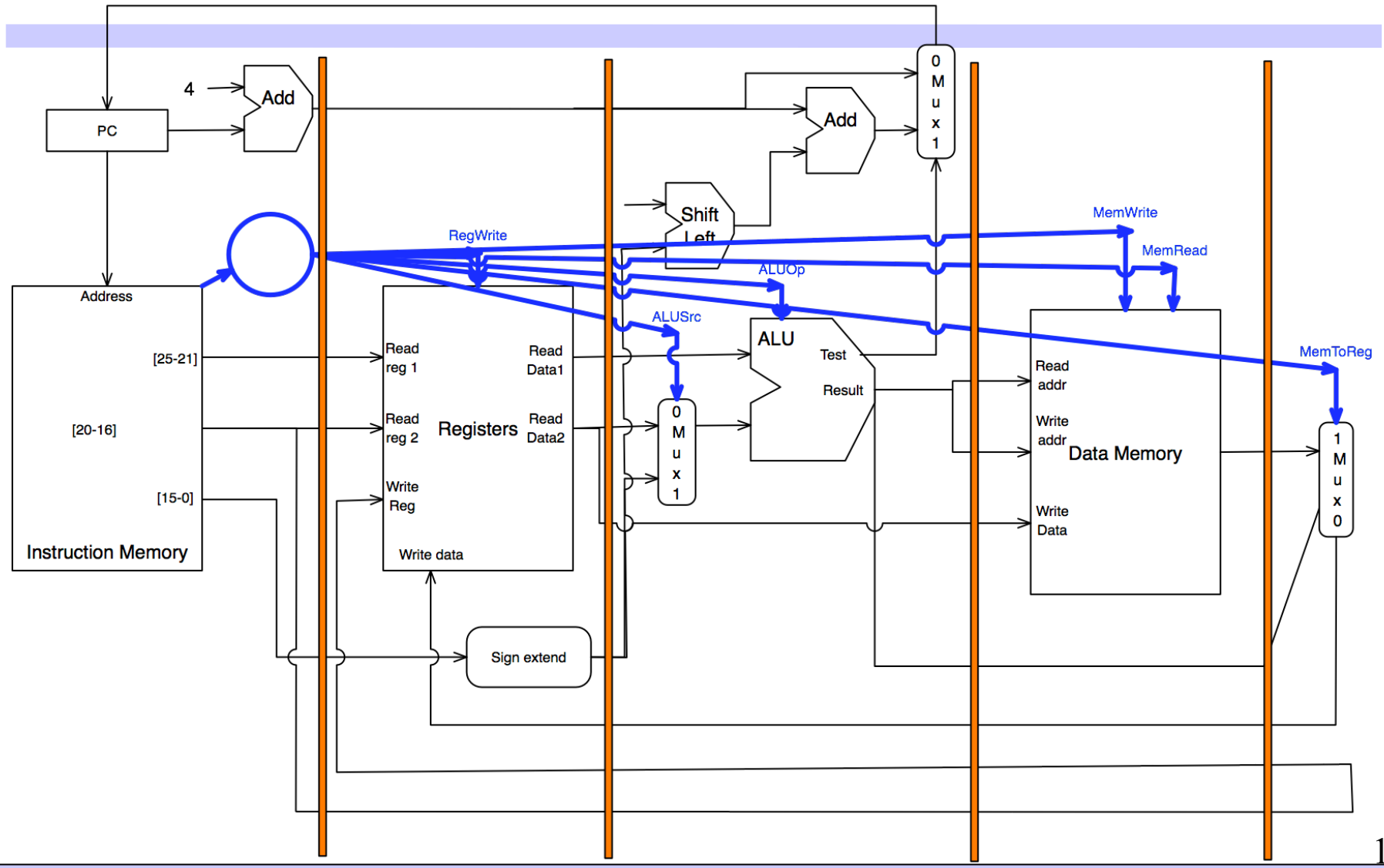
Registres entre étages (register latch)

- On insère des registres entre les étages pour stocker les valeurs produites à la fin du cycle par un étage.
- Ces valeurs sont transmises au cycle d'après à l'étage suivant
- Registres nommés par leur position entre étages:

IF/ID, ID/EX, EX/Mem, Mem/WB

Les registres servent à cloisonner les étages.

5-c Exécution pipelinée: latches



5-c Exécution pipelinée

Avantage du pipeline:

- Meilleure utilisation des différentes unités
- Utilise du parallélisme entre instructions (5 instructions simultanément dans le pipeline MIPS)
- Fait gagner en performances, en augmentant la fréquence

Inconvénient du pipeline

- Nécessite de raccourcir le temps d'un cycle → augmente la fréquence donc la dissipation d'énergie
- Les instructions qui sont simultanément dans le pipeline ne doivent pas dépendre les unes des autres → **difficile à imposer a priori !**

Solution 1: le matériel gère ce problème.

Solution 2: le compilateur gère.

5-c Problèmes d'exécution pipelinée

Difficultés pour un pipeline:

- Dépendances de données: la valeur calculée par une instruction doit être utilisée par une autre instruction, mais l'utilisation a lieu avant le calcul... **5-d**
- Branchements: que fait-on pendant que le calcul du branchement est effectué ? **5-e**

Les solutions générales pour ces difficultés

- Forwarding
- Stall
- Spéculation

5-d Dépendances

Dépendances de données

Exemple:

```
sub rax,rbx
add rcx,rax
or rdx,rax
add  r8,rax
mov r10,[rax]
```

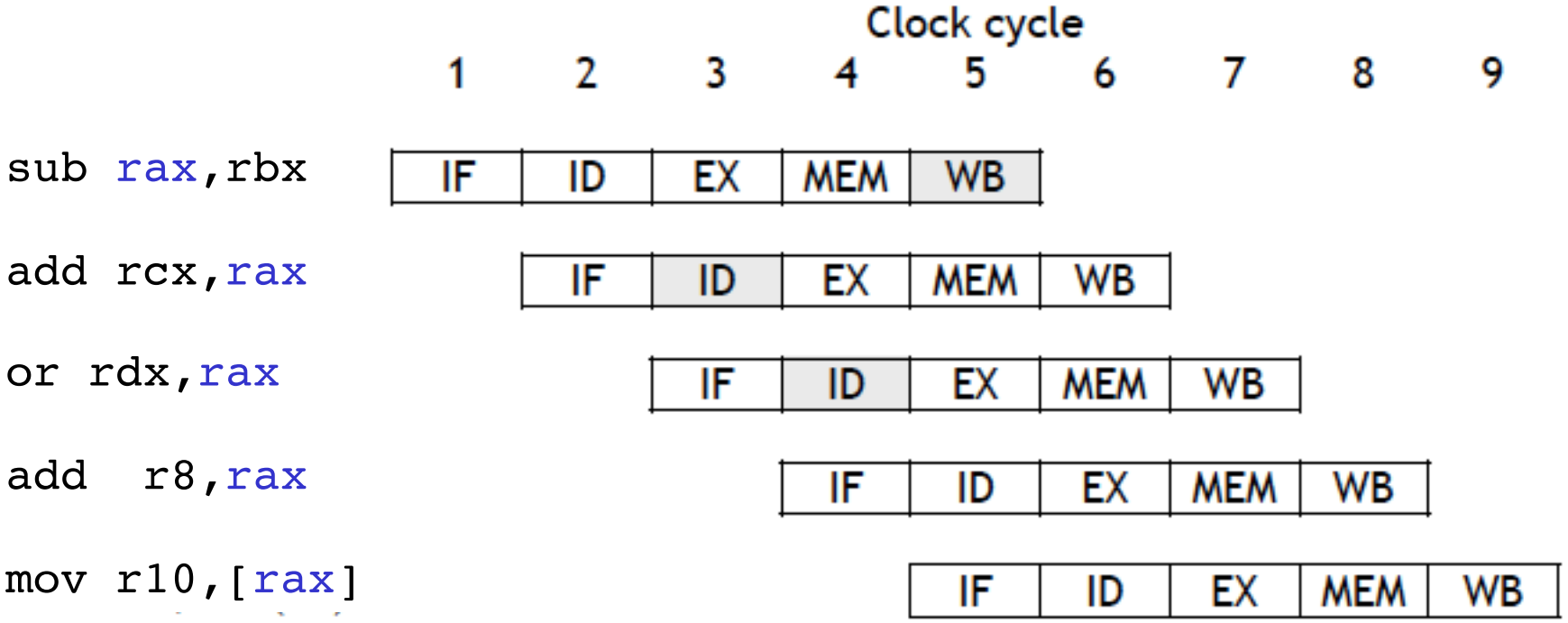
Le registre rax est écrit par **sub** puis lu par les instructions suivantes.

- Pas un problème lorsque 1 instruction = 1 cycle : **sub** fini avant **add**
- Pour le pipeline, **sub** pas fini avant **add**

5-d Dépendances

Détail: 2 dépendances de données posent problème

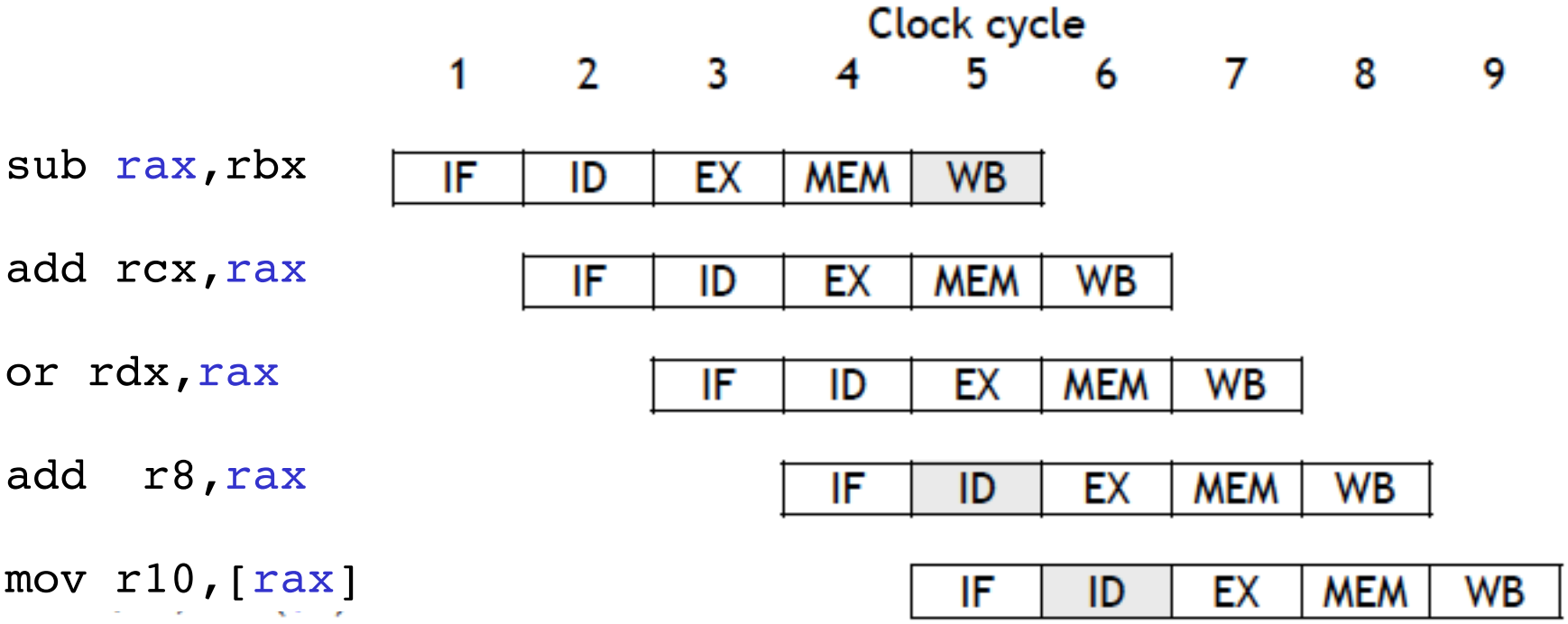
- **rax** est écrit à l'étage WB, à la fin du cycle 5 par le **sub**
- **rax** est lu à l'étage ID, fin des cycles 3 et 4, par le **add** et **or** et utilisé par l'étage EX le cycle d'après, au début des cycles 4 et 5.



5-d Dépendances

Détail: 2 dépendances ne posent pas de problème

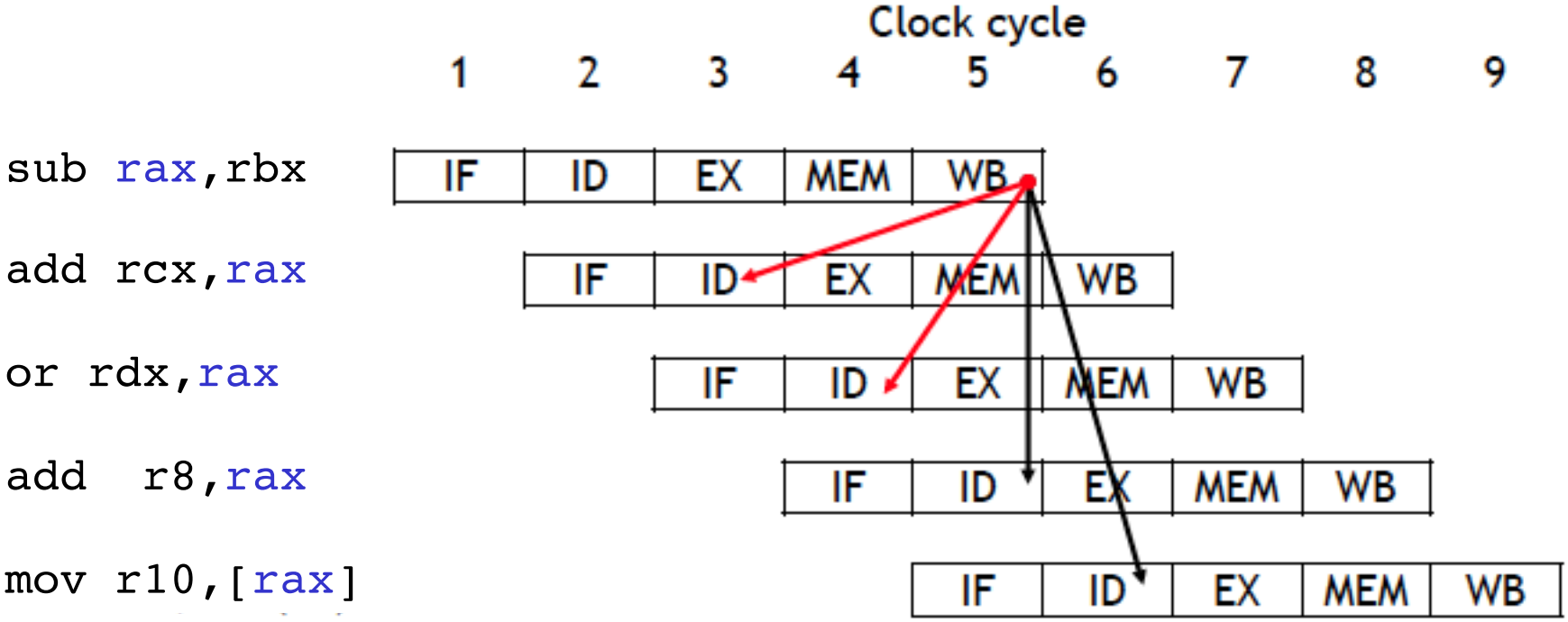
- Les deux dernières instructions **add** et **mov** ne posent pas de problème
- Le registre est écrit en début de cycle 5 par **sub**, lu en fin des cycles 5 et 6 par **add** et **mov**



5-d Dépendances

Schéma des dépendances:

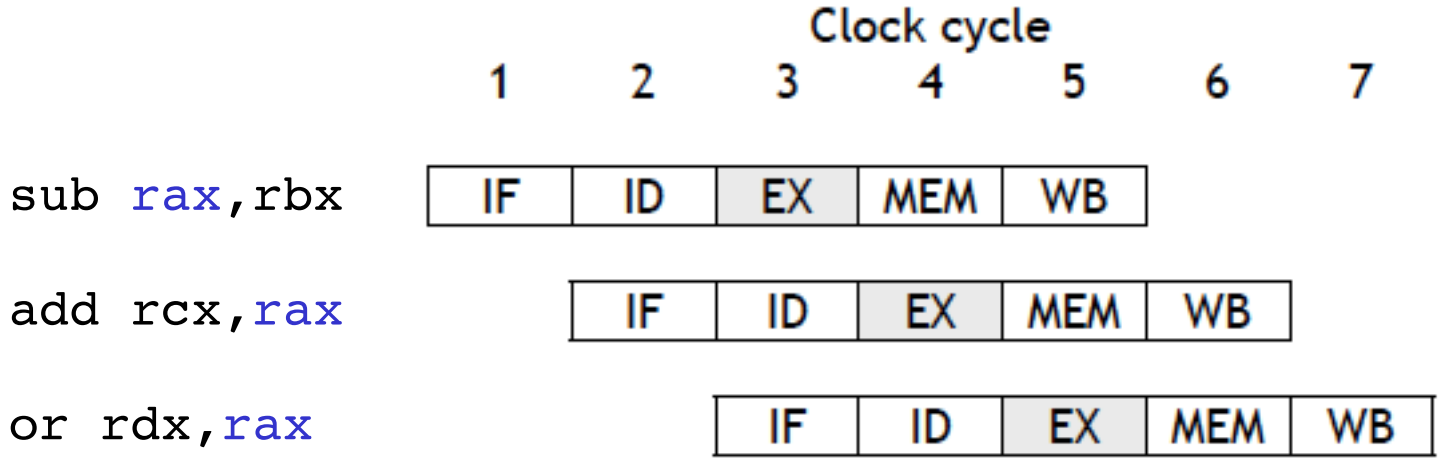
- Flèches sur le diagramme, de l'écriture de la valeur jusqu'à sa lecture
- Les flèches qui vont en arrière: problèmes d'interdépendances



5-d Dépendances et forwarding

1ere solution: forwarding

- La valeur requise est calculée en fait dès la fin du cycle 3, à l'étage EX, et peut être stockée dans le registre de latch EX/Mem
- On la propage (forward) à l'entrée de l'ALU pour les autres instructions en début des cycles 4 et 5.

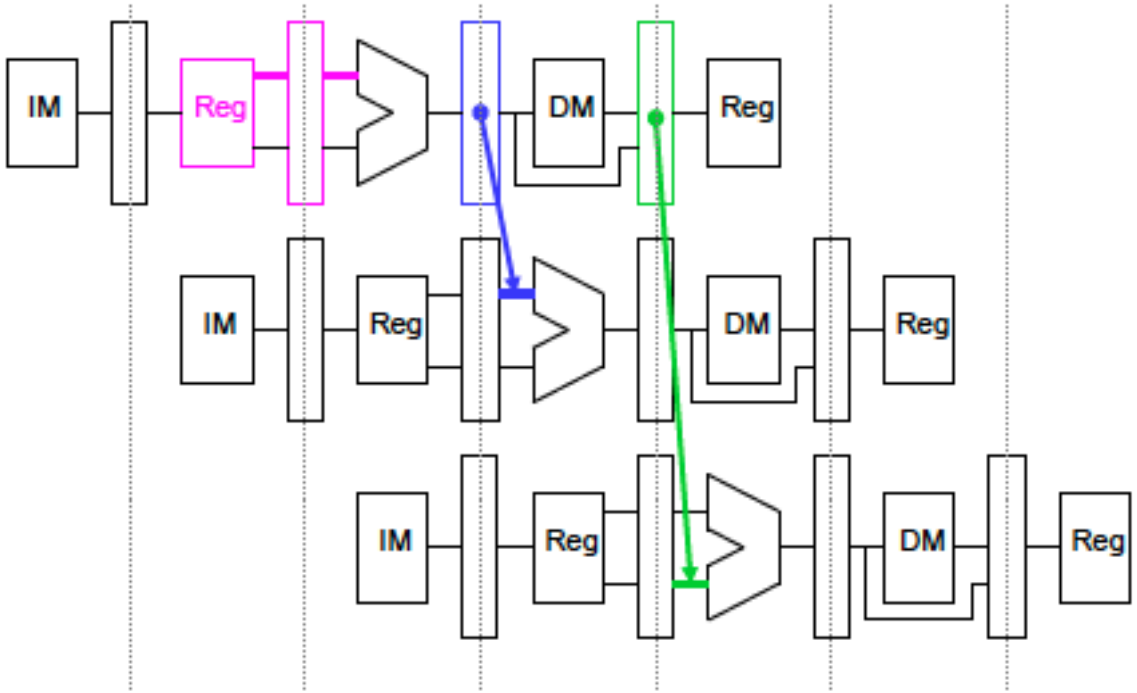


5-d Dépendances et forwarding

Forwarding

- Lecture de la valeur de $2 pour le **and** dans EX/Mem
- Lecture de la valeur de $2 pour le **or** dans Mem/WB

```
sub rax,rbx

add rcx,rax

or rdx,rax
```

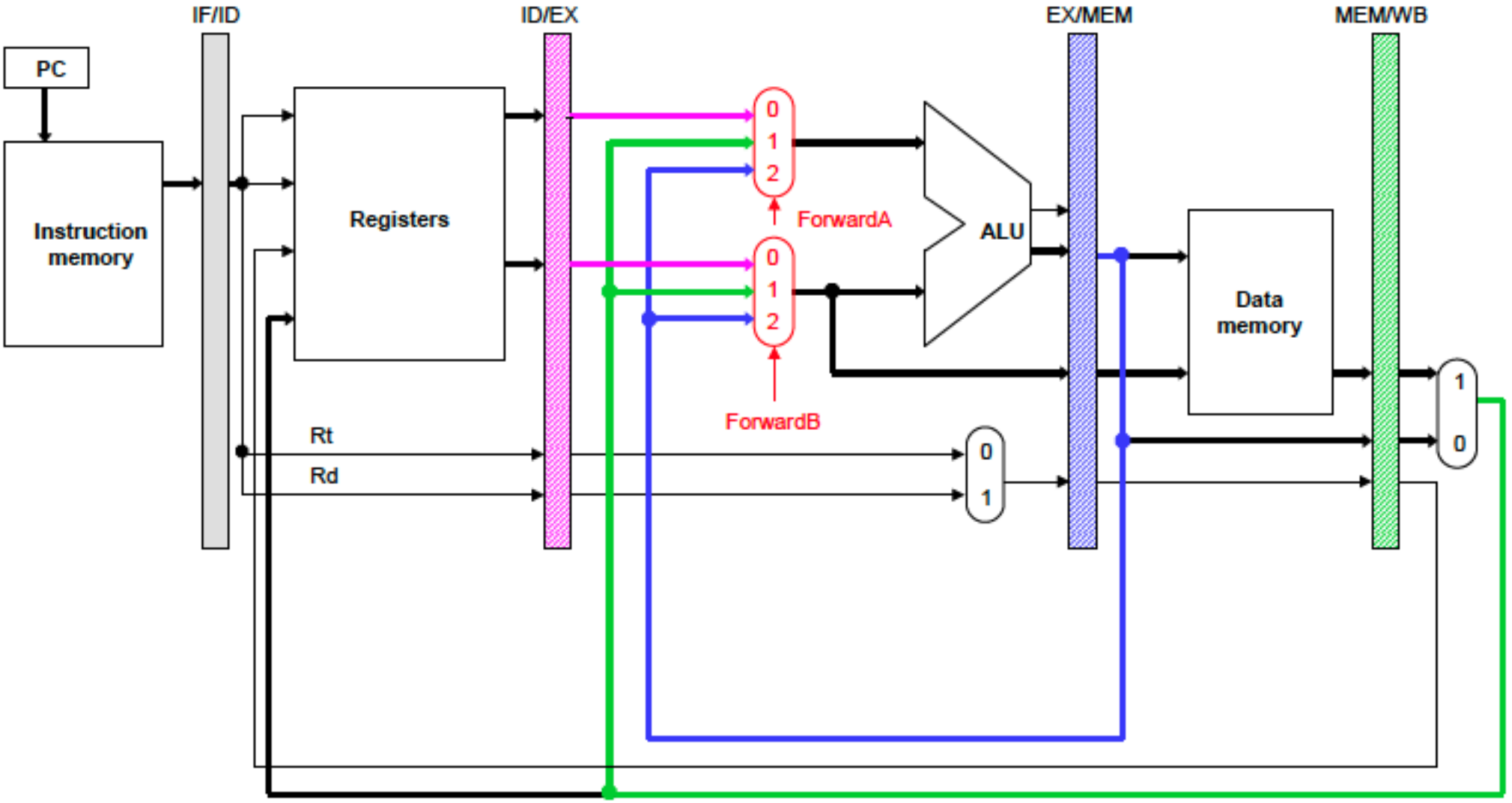


5-d Dépendances et forwarding

Modification de l'architecture pour le forwarding

- Une unité de forwarding choisit la bonne source pour l'ALU
  - Si pas de dépendances, la valeur vient du banc de registres ou de la constante de l'instruction
  - Si dépendance, la valeur vient soit de EX/Mem, soit de Mem/WB
- L'unité de forwarding contrôle un multiplexeur sur les entrées de l'ALU

5-d Dépendances et forwarding



5-d Dépendances et forwarding

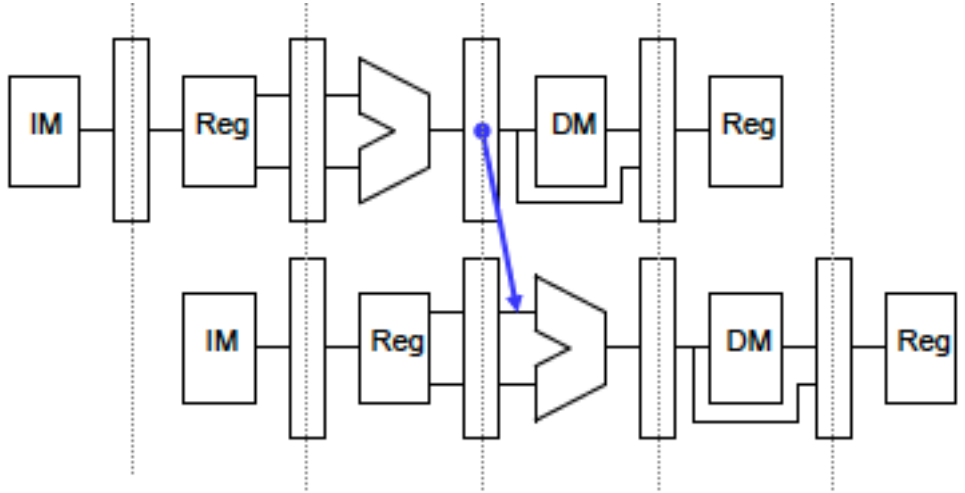
Conditions pour appliquer le forwarding

- Une instruction est dans EX et une autre dans l'étage ID réutilise son résultat

ID/EX.rd == IF/ID.rs (premier opérande)

ou ID/EX.rd == IF/ID.rt (second opérande)

```
sub rax,rbx
add rcx,rax
```

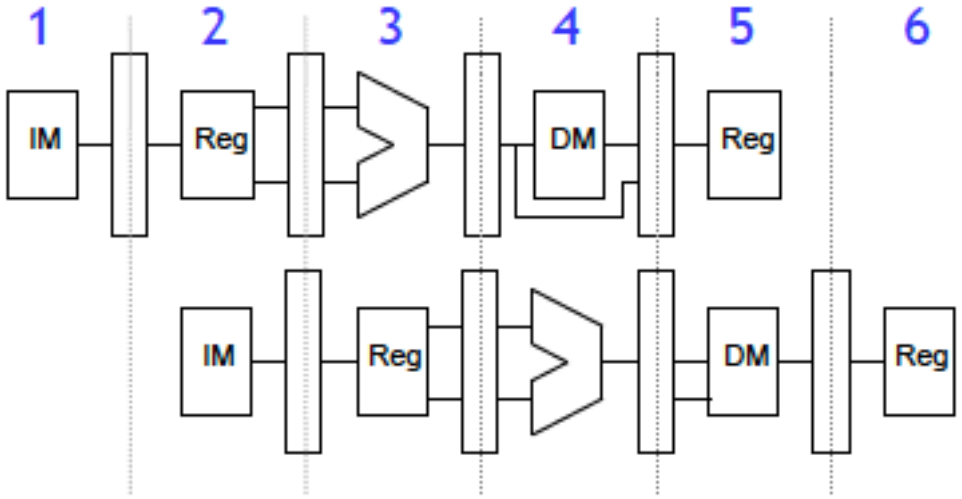


5-d Dépendances et forwarding

Un autre exemple

- A quel cycle la valeur de rax est calculée par le **add** ?
- A quel cycle la valeur de rax est lue par le **mov**?
- Que rajouter au chemin des données ?

```
sub rax,rbx

mov [rax+8],rcx
```

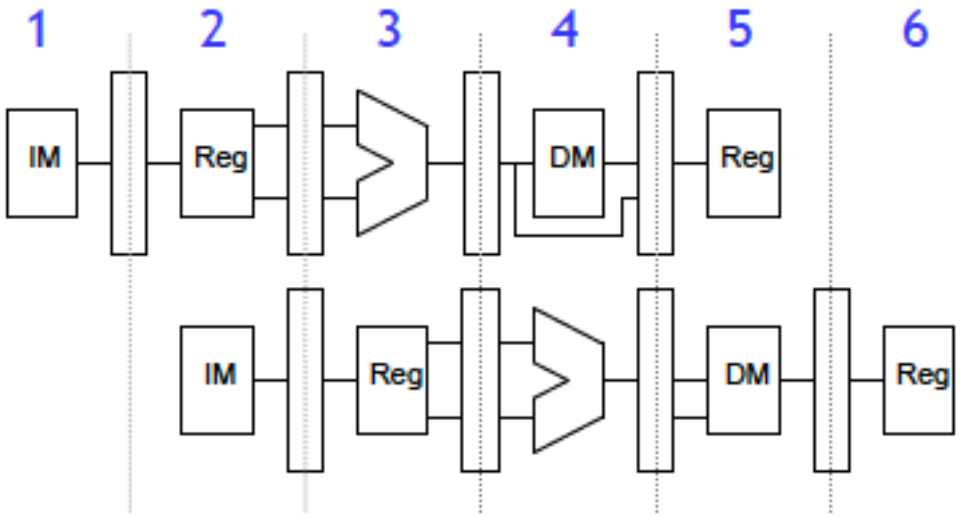


5-d Dépendances et forwarding

Un autre exemple

- A quel cycle la valeur de $1 est calculée par le **add** ?
- A quel cycle la valeur de $1 est utilisée par le **sw** ?
- Que rajouter au chemin des données ?

```
sub rax,rbx

mov [rcx+8],rax
```

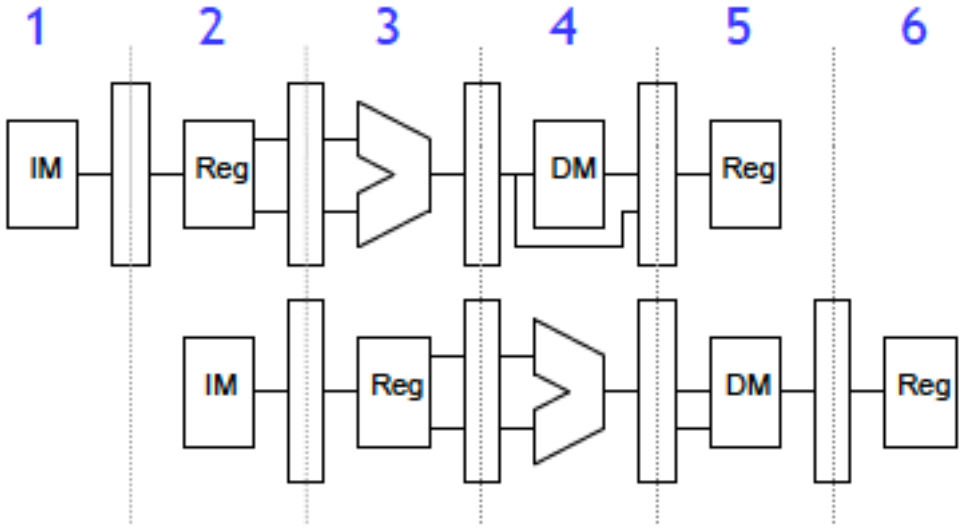


5-d Dépendances et forwarding

Un autre exemple

- A quel cycle la valeur de $1 est chargée par le **lw** ?
- A quel cycle la valeur de $1 est utilisée par le **sw** ?
- Que rajouter au chemin des données ?

```
mov [rax],rbx

mov rcx,[rax+8]
```

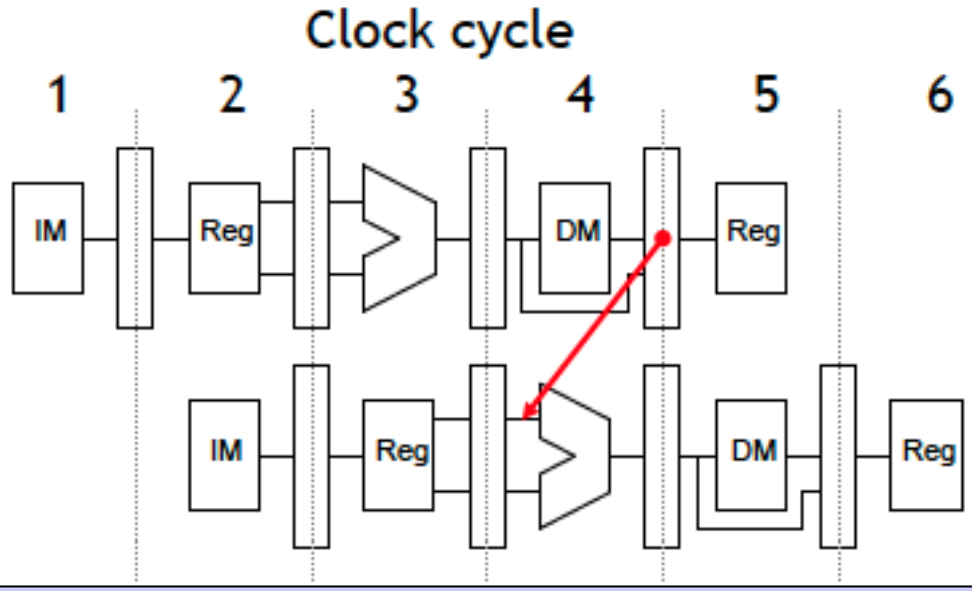


5-d Dépendances et bulles

Le Forwarding ne marche pas dans tous les cas

- La dépendance est vers l'arrière: il y a une dépendance posant problème
- La valeur n'est pas disponible avant son utilisation par le **and**

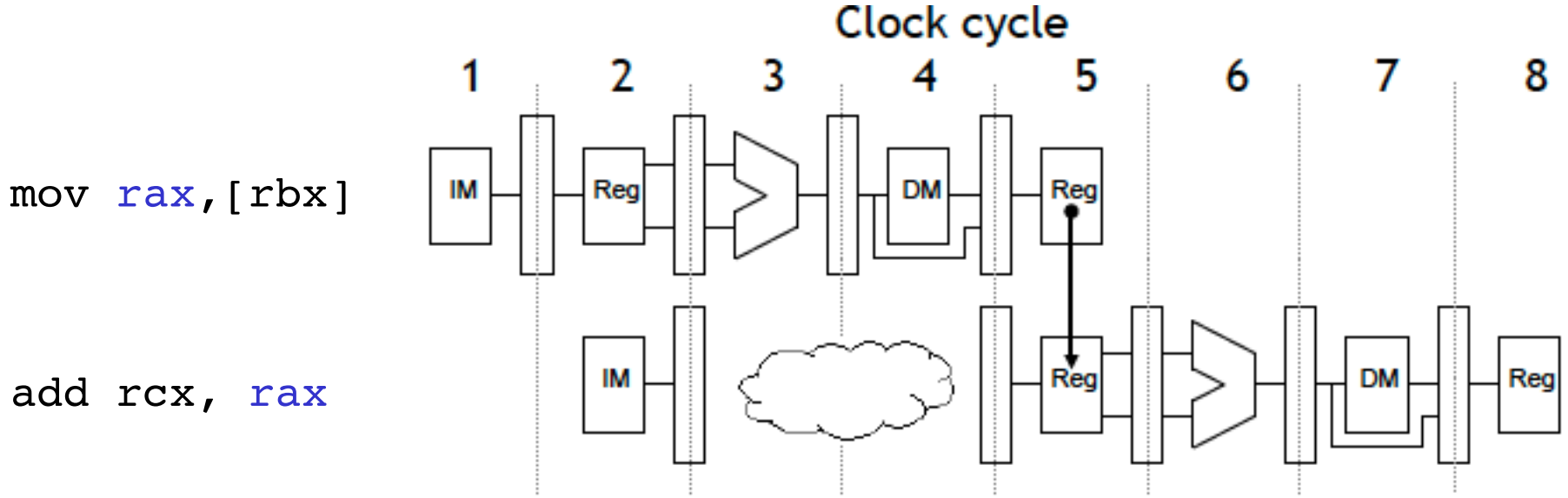
```
mov rax,[rbx]
add rcx, rax
```



5-d Dépendances et stall

Solution:

- Arrêter la progression des instructions dans le pipeline avant l'étage EX en attendant que le résultat soit prêt
- Met une **bulle** de 2 cycles dans le pipeline, il y a un **stall** (arrêt partiel du pipeline)

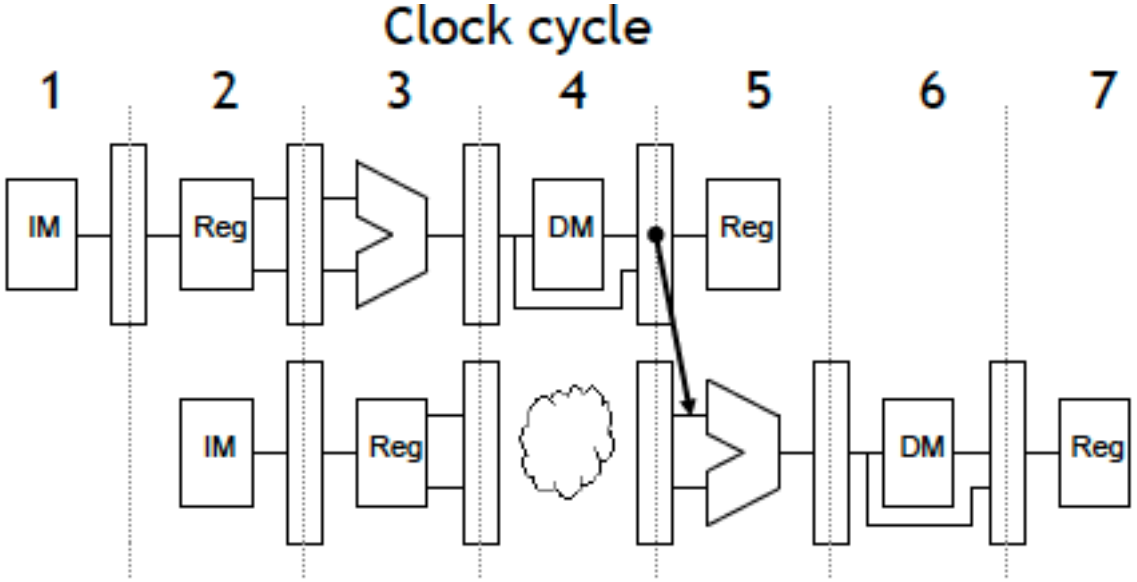


5-d Dépendances, forwarding et stalls

Meilleure solution

- La valeur est prête dès la fin du cycle 4 → forwarding de Mem/WB vers l'ALU
- Economise un cycle.

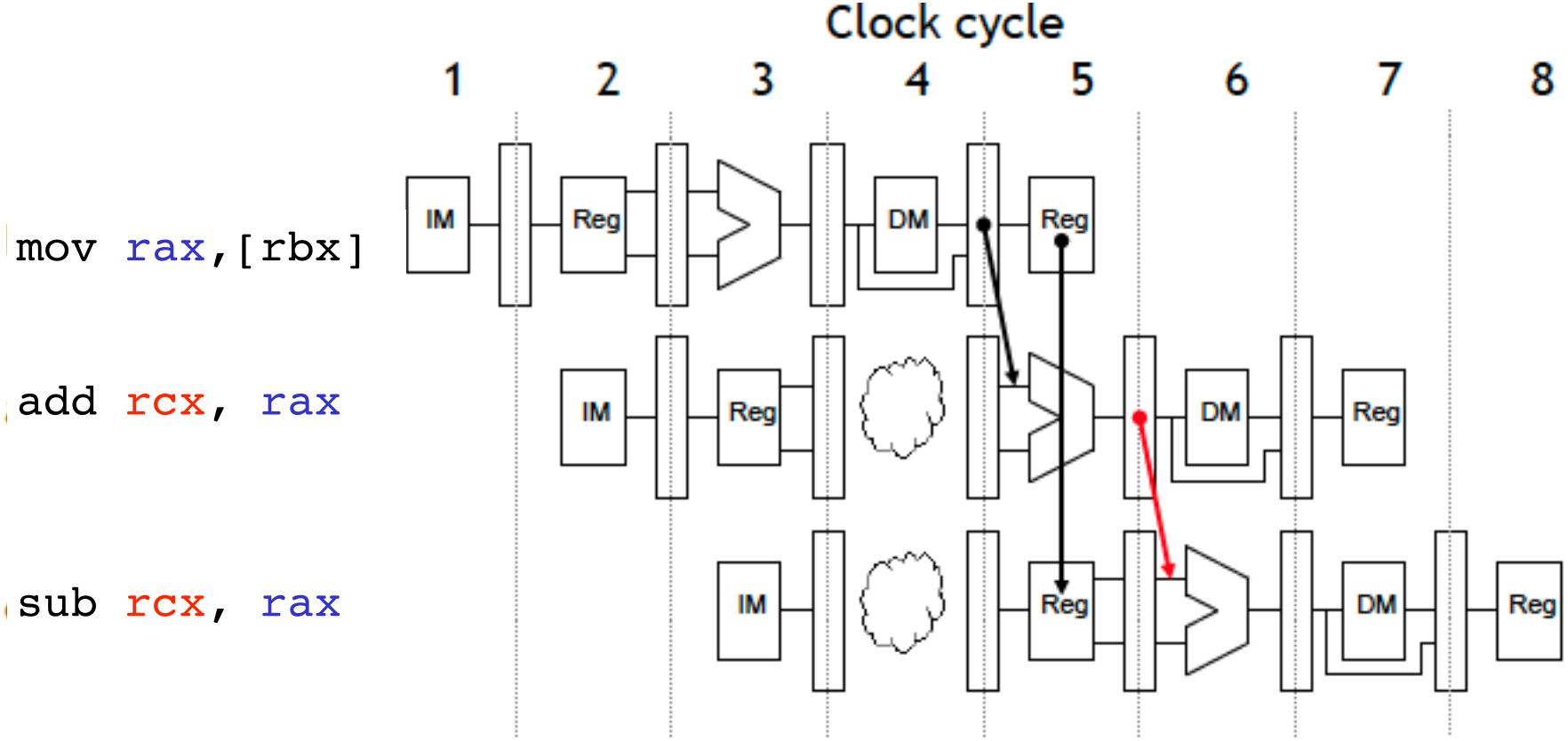
```
mov rax,[rbx]
add rcx, rax
```



5-d Dépendances, forwarding et stalls

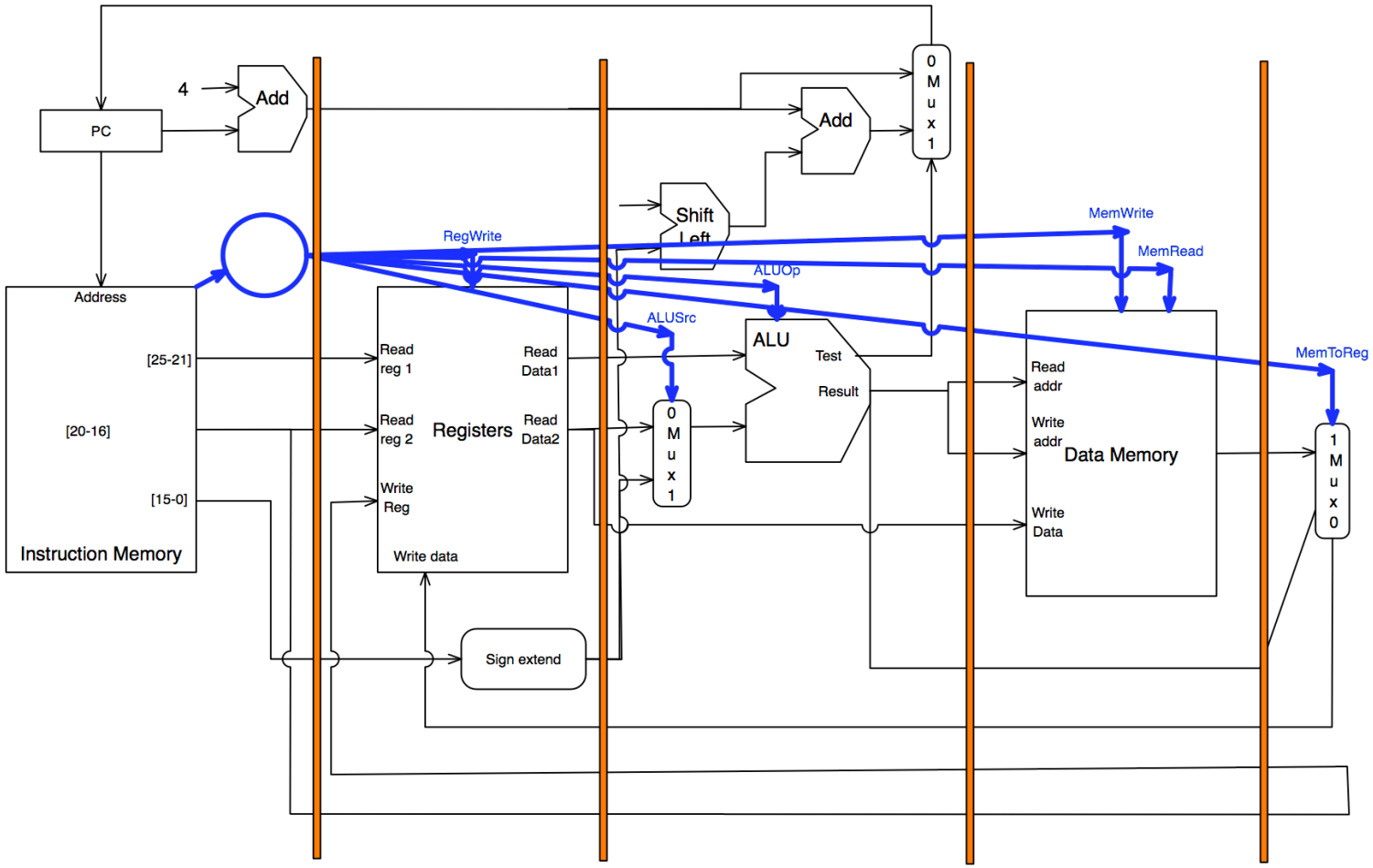
Stall: stoppe **tous** les étages précédant celui qui est à l'origine de la dépendance

- Coute d'autant plus cher en cycles que le pipeline a beaucoup d'étages !



5-e Branchements

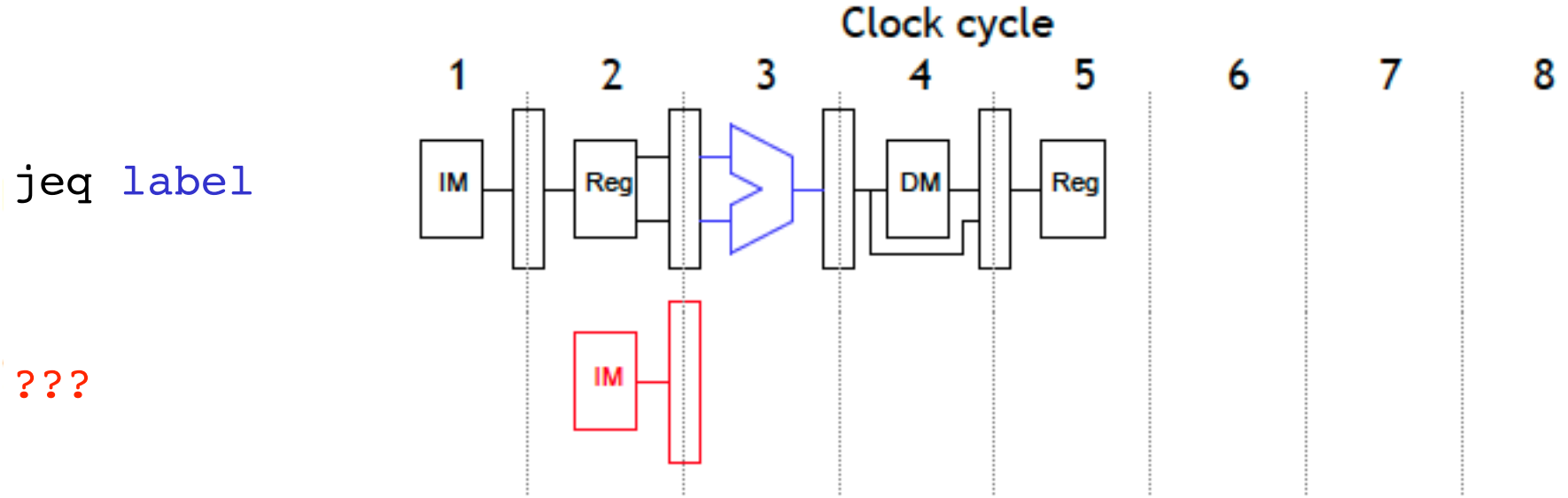
Quand sont-ils calculés ??



5-e Branchements et stalls

Difficulté

- On sait si on fait un branchement à la fin du cycle 3
- Il faut exécuter l'instruction suivante au début du cycle 2...
- ...le forwarding ne peut pas aider → stall !

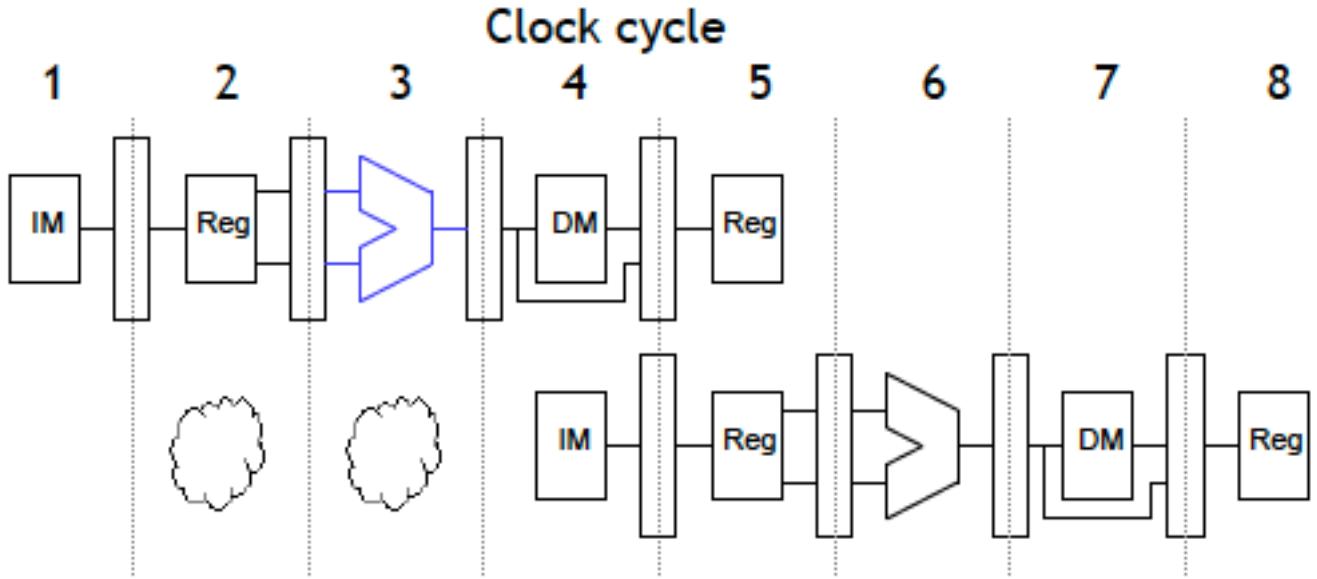


5-e Branchements et stalls

Stall de deux cycles

- L'instruction suivante ne commence qu'au cycle 4

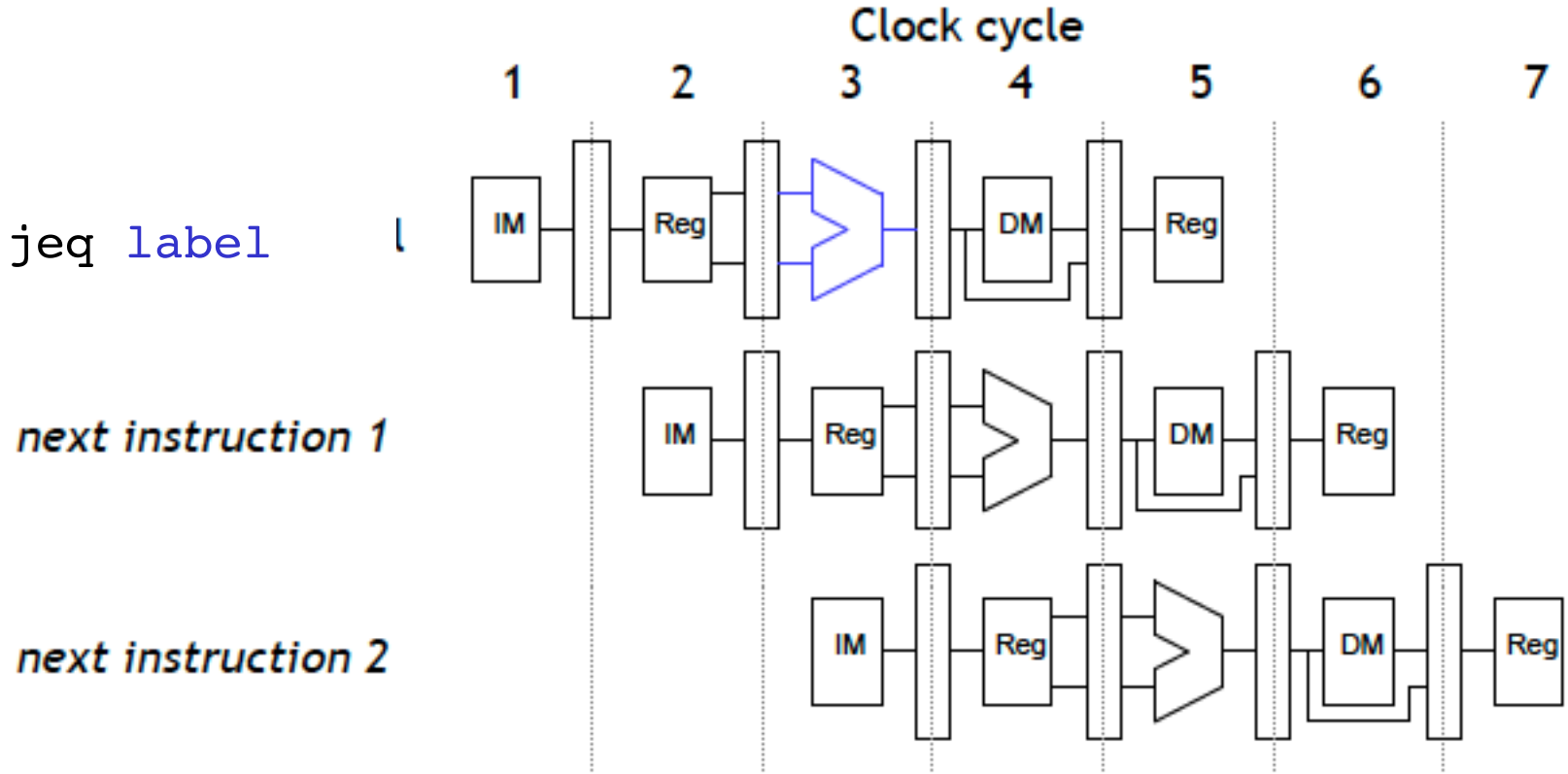
```
jeq label

???
```



5-e Branchement et spéculation

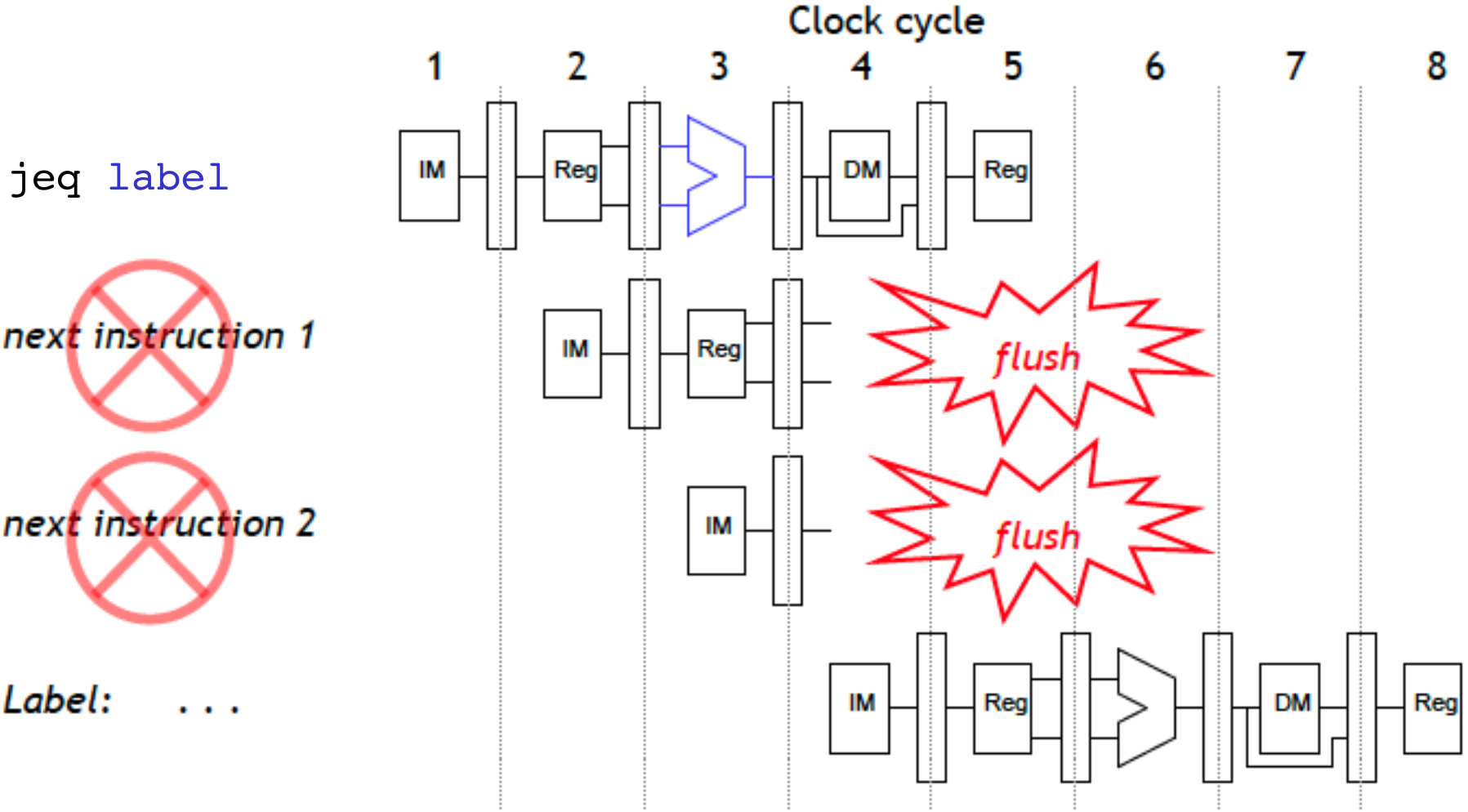
On spécule qu'en fait, on ne fera pas le branchement !

- Très avantageux: les instructions suivantes sont exécutée sans attendre
- Si on s'est trompé, on le sait au cycle 4



5-e Branchement et spéculation

- Si on s'est trompé, on **réinitialise** le pipeline, on recommence avec la bonne instruction (flush).



5-e Branchement

Gain de la spéculation

- Perd 2 cycles quand la spéculation n'est pas la bonne
- Au lieu de perdre 2 cycles à chaque branchement

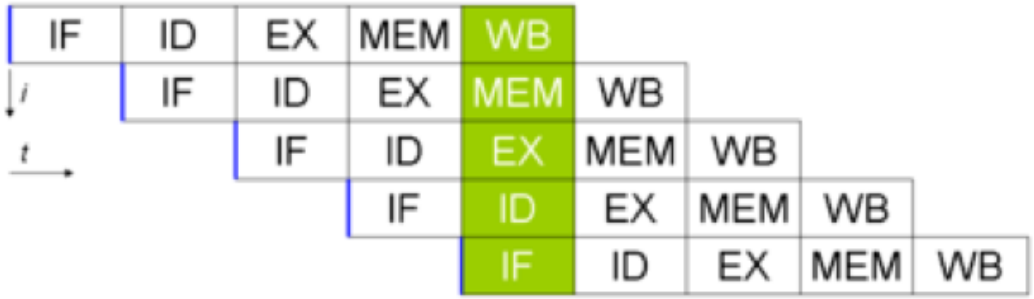
Spéculation très présente dans processeurs modernes

- Flush a un coût important pour long pipelines
- Amélioration de la spéculation, diminution probabilité flush:
  - Calcule des statistiques pour chaque branchement
  - Utilise des caches de branchement pour prédire prochaine adresse instruction (cf après)
- !! Les instructions à flusher ne doivent avoir eu aucun effet sur la mémoire !!

5- Pipeline: conclusion

Sans pipeline

Avec pipeline



Quelques longueurs de pipelines

Processeur	Longueur pipeline	Processeur	Longueur pipeline
MIPS R4400	8	IBM Power5	16
Intel Itanium	10	Intel Core	14
Sun UltraSparc IV	14	Intel Pentium4 Prescott	31
ARMv6	8	ARMv7A (Cortex A8)	13

5- Pipeline: conclusion

Exécution pipelinée

- Permet meilleure utilisation du hardware
- Amélioration des performances (au max, gain d'un facteur de la longueur du pipeline)
- Nécessite indépendance entre instructions pour que le pipeline soit toujours plein

Dépendances et branchements

- Read After Write (RAW), Write after Read (WAR), Write after Write (WAW) pour les dépendances sur les registres
- **Solutions**: **stall (insertion de bulles), forwarding pour les dépendances, speculation pour branchements et flush éventuel**