INSA-3IF-3TC

Pipeline des instructions RISC

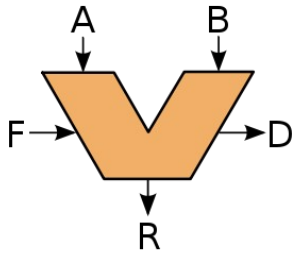
Christian Wolf,
INSA-Lyon, Dép. IF

Cycle de von Neumann : exemple

- IF (*Instruction Fetch*)
  - Lecture de l'instruction à partir du PC
- ID (*Instruction Decode*)
  - Accès au registres; calcul de la destination d'un saut conditionnel
  - Décode + accès aux registres : requiert orthogonalité!
- EX (*Execute*)
  - Travail ALU: op. ALU ou calcul adresse pour accès MEM
  - Requiert une architecture « chargement-rangement » !
- MEM (*Memory access*)
  - Chargement ou rangement mémoire
- WB (*Write Back)*
  - Ecrit dans le banc de registres (résultat ALU ou retour mémoire)

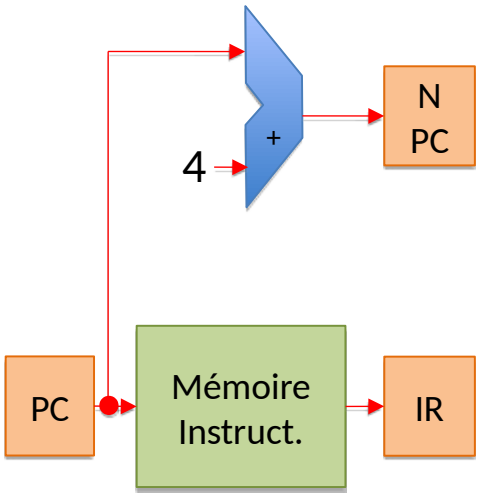
Rappel du cycle de von Neumann

- Le cycle est exécuté par l'automate, contrôlant le chemins de données au sein du processeur



- Durée d'une instruction = période horloge * nombre de cycles (exemple ci-dessus : 5 cycles = 5 CPI « cycles par instruction »)

Cycle« IF »



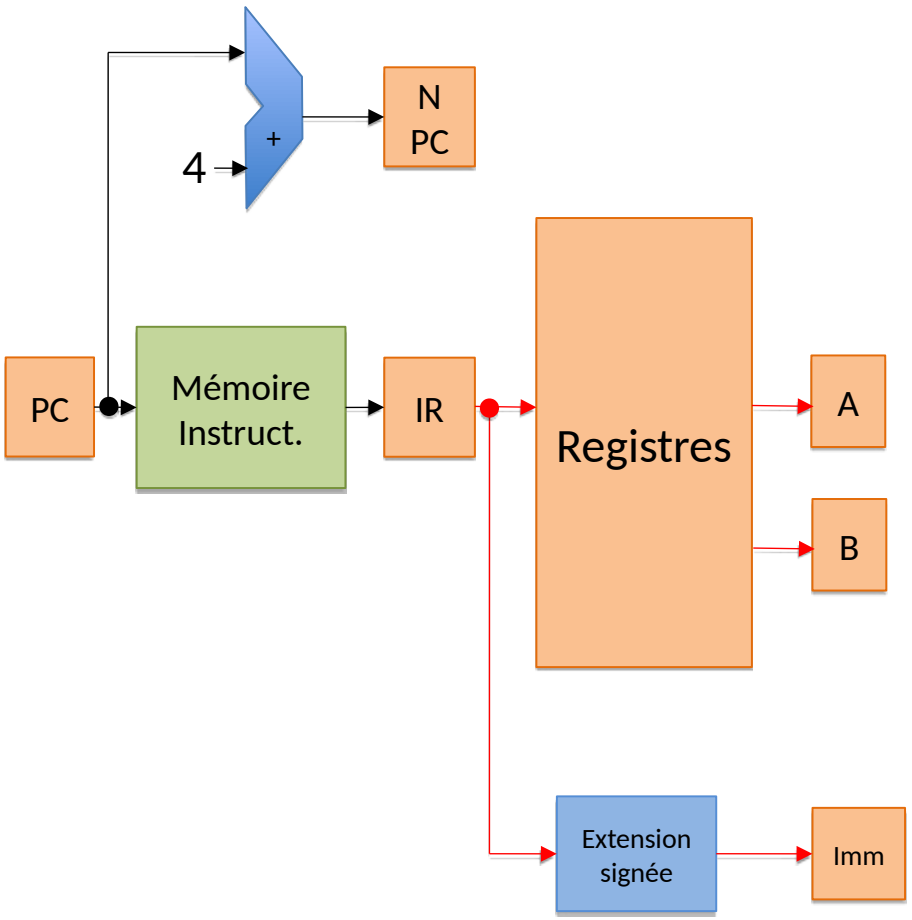
« Instruction fetch »

IR ⩵ Mem[PC]

NPC ⩵ PC+4

On garde la nouvelle valeur du PC dans un registre dédié

Cycle« ID »



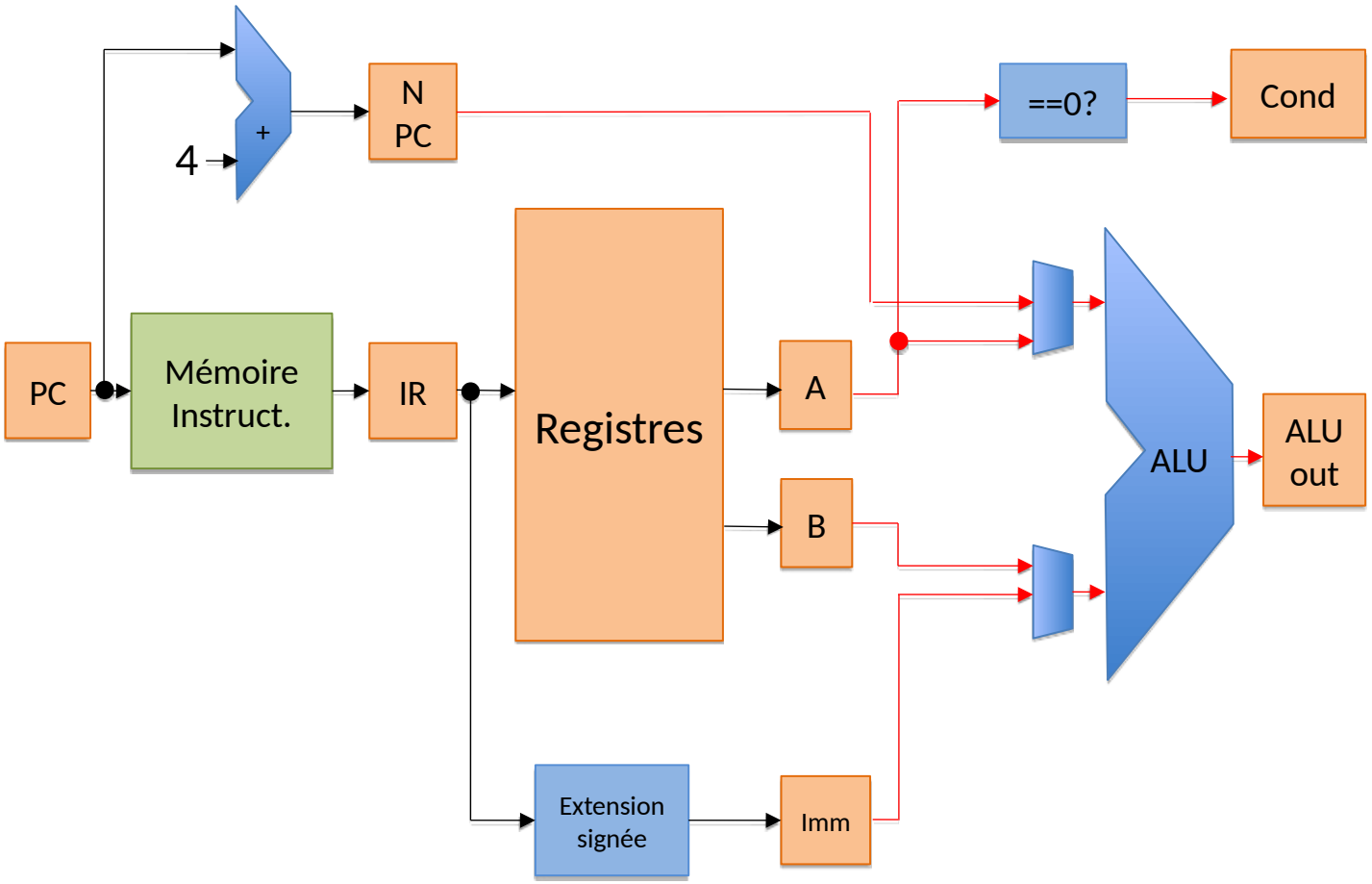
« Instruction decode »

A ≔ Regs[rs]
B ≔ Regs[rt]
Imm ≔ extension/mask[IR]

Calcul en parallèle :
- Décodage
- accès aux registres

Orthogonalité des instructions (encodage fixe des choix de registres) !!

Cycle« Ex »



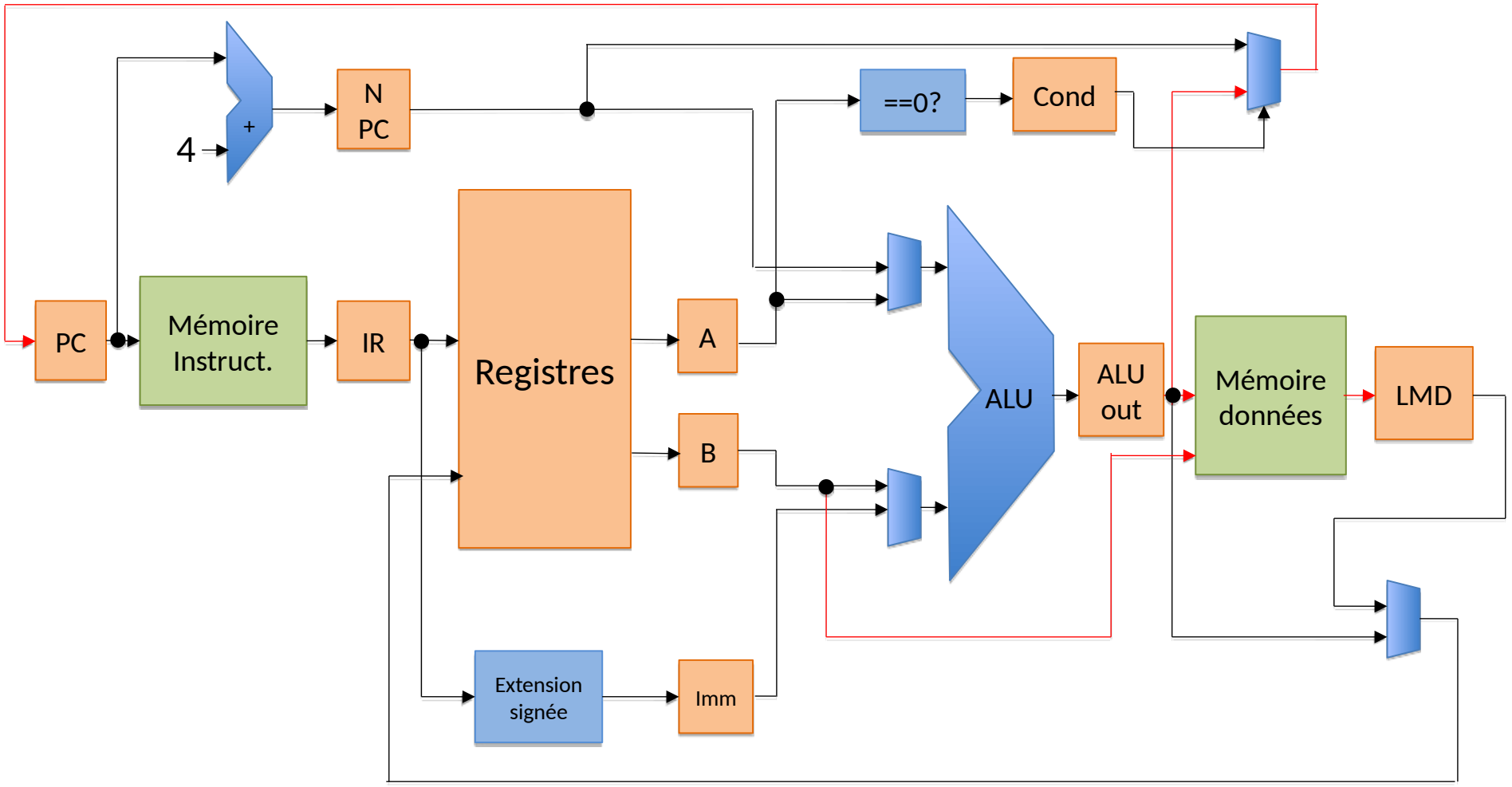
Cas 1 : add r1, r2, r3

Cas 2 : add r1, r2, #12

Cas 3 : ldr r3, [r4]

Cas 4 : beq .L1

Cycle « MEM »

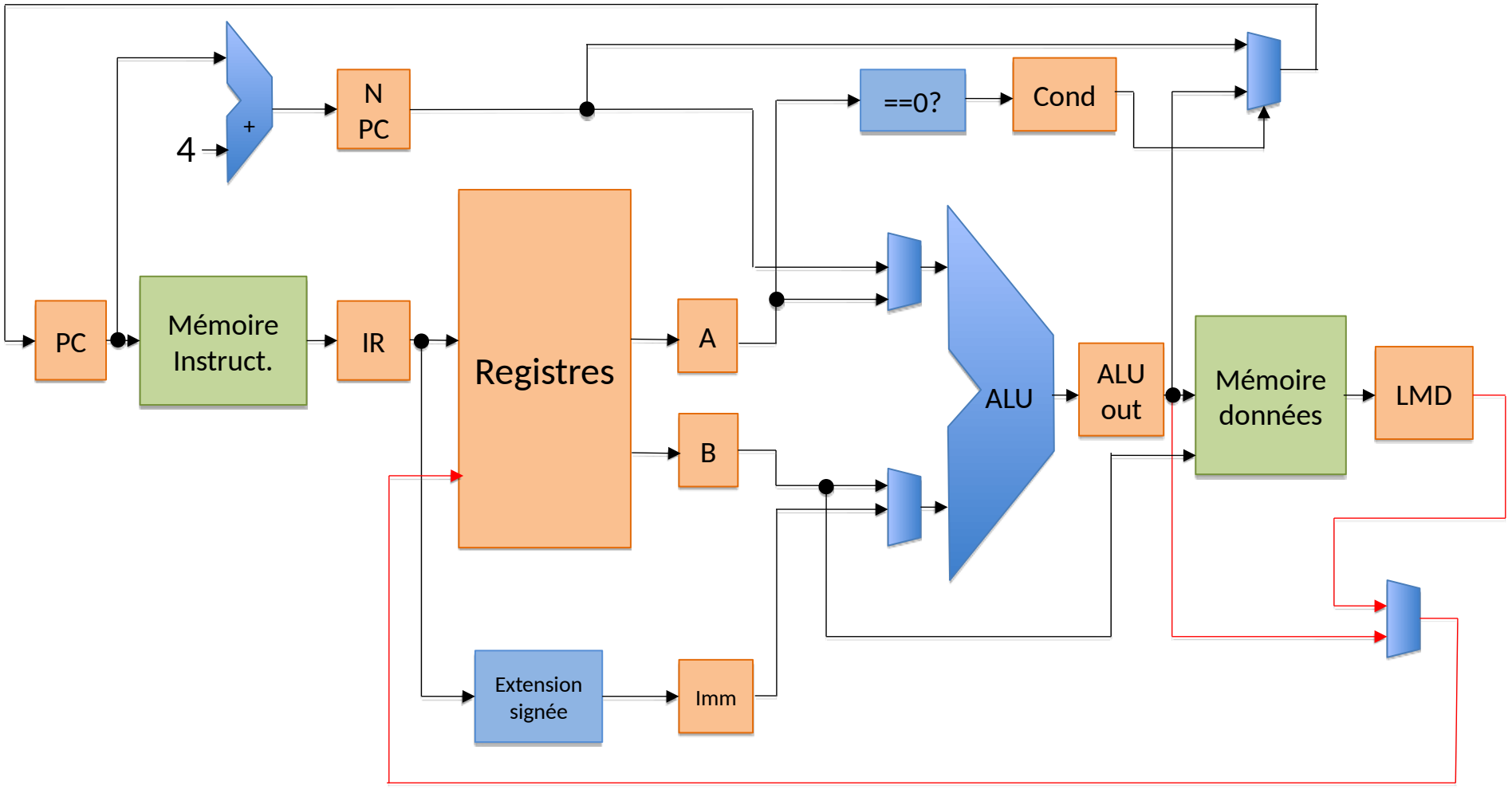


Cas 1 : ldr r1, #8001

Cas 2 : ldr r3, [r4]

Cas 3 : beq .L1

Cycle « WB »



Cas 1 : ldr r1, #8001

Cas 2 : add r1, r2, r3

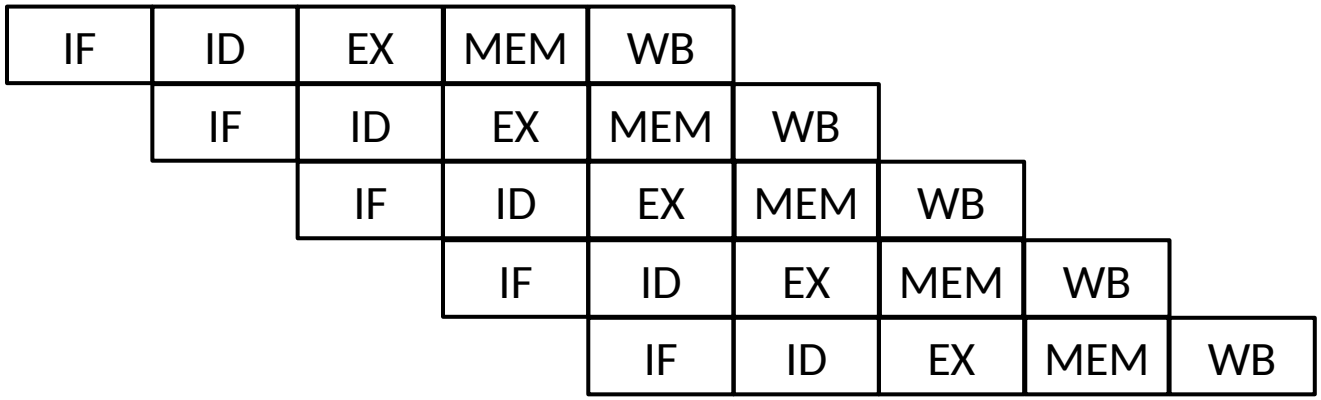
Performance

- Mesurée en instructions par unité de temps (sec)
- Dépend de
  - Fréquence horloge (Hz)
  - CPI (cycles par instruction)

- Compromis :
  - Augmenter la fréquence peut nécessiter des étapes plus « courtes », donc augmenter CPI
  - Au lieu d'exécuter les 5 etapes (IF,ID,EX,MEM,WB) en un cycle, on va les exécuter en 5 cycles

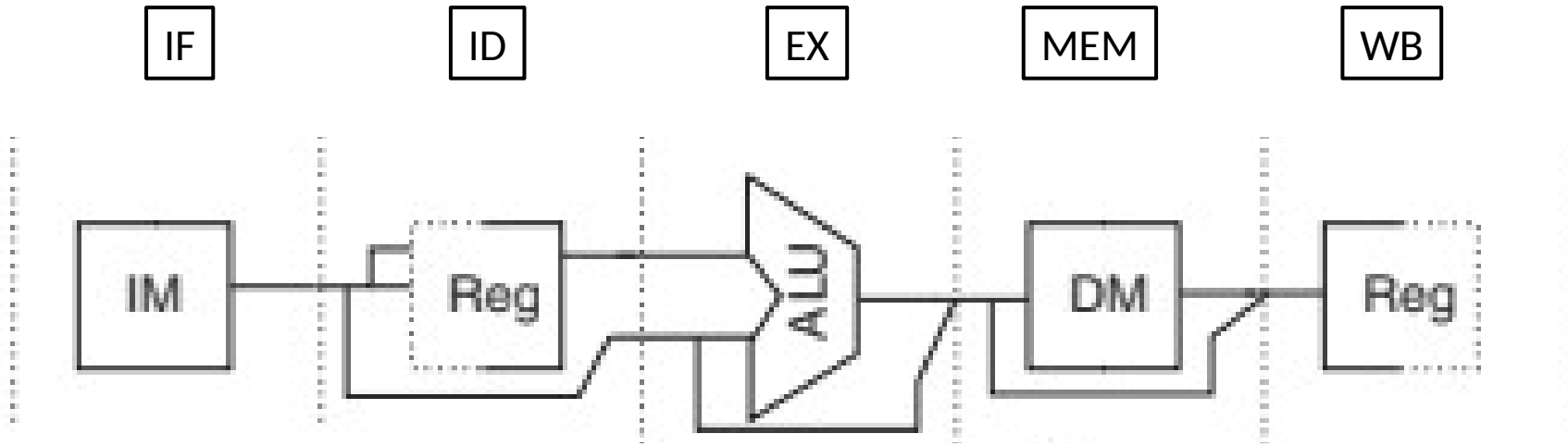
Un pipeline

- Fonctionnement similaire à une chaine d'assemblage
- Le processeur travaille sur plusieurs instructions en parallèle, en étant dans un état différent pour chaque instruction



- Durée d'une instruction : égale à la version non pipelinée
- Débit (max) :
  - version non pipelinée * nombre étages
  - Egale à la fréquence de l'horloge (une instruction par cycle!)

Chemin de données : utilisation



Instruction Memory

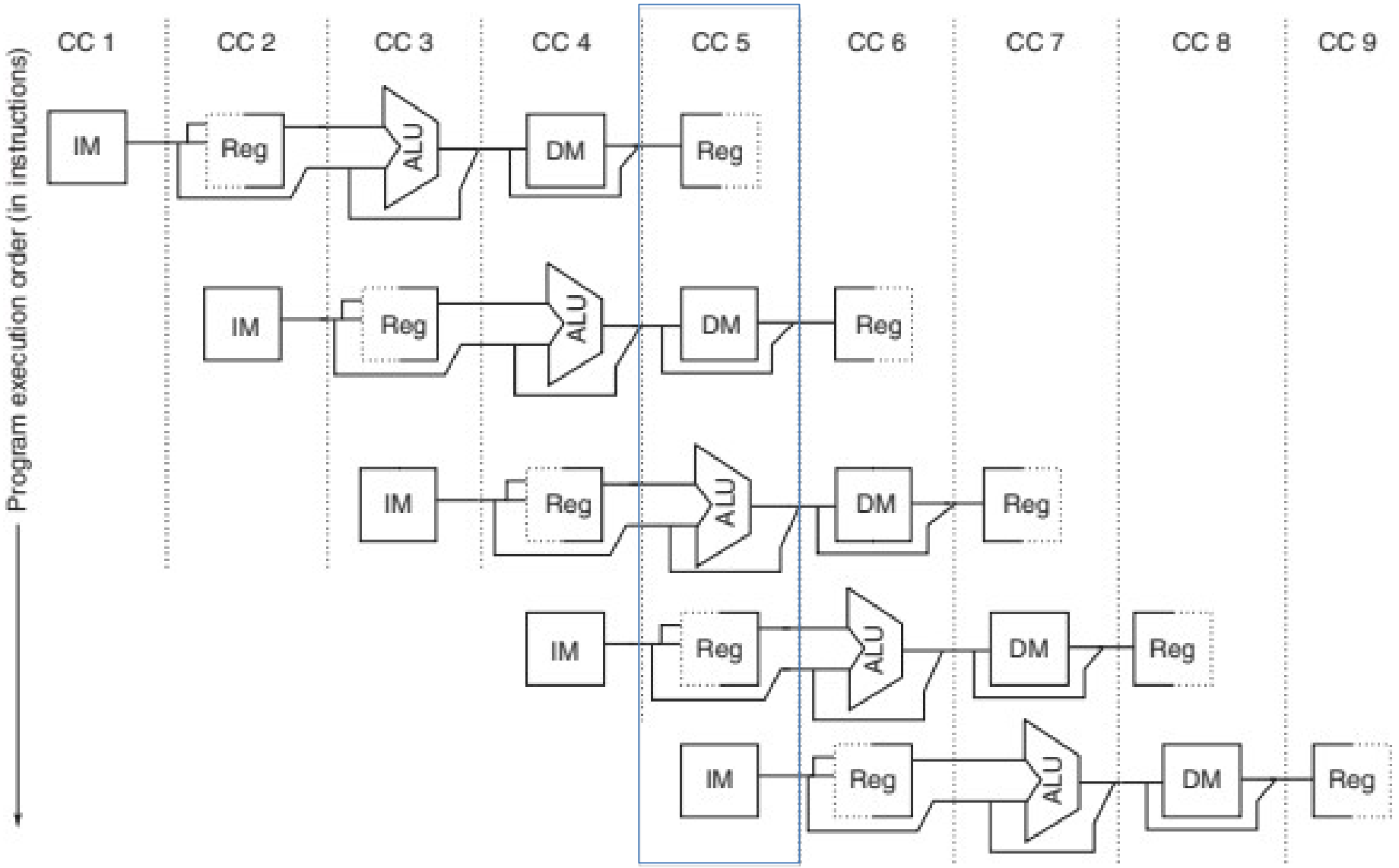
Lecture registres

Data memory

Ecriture registres

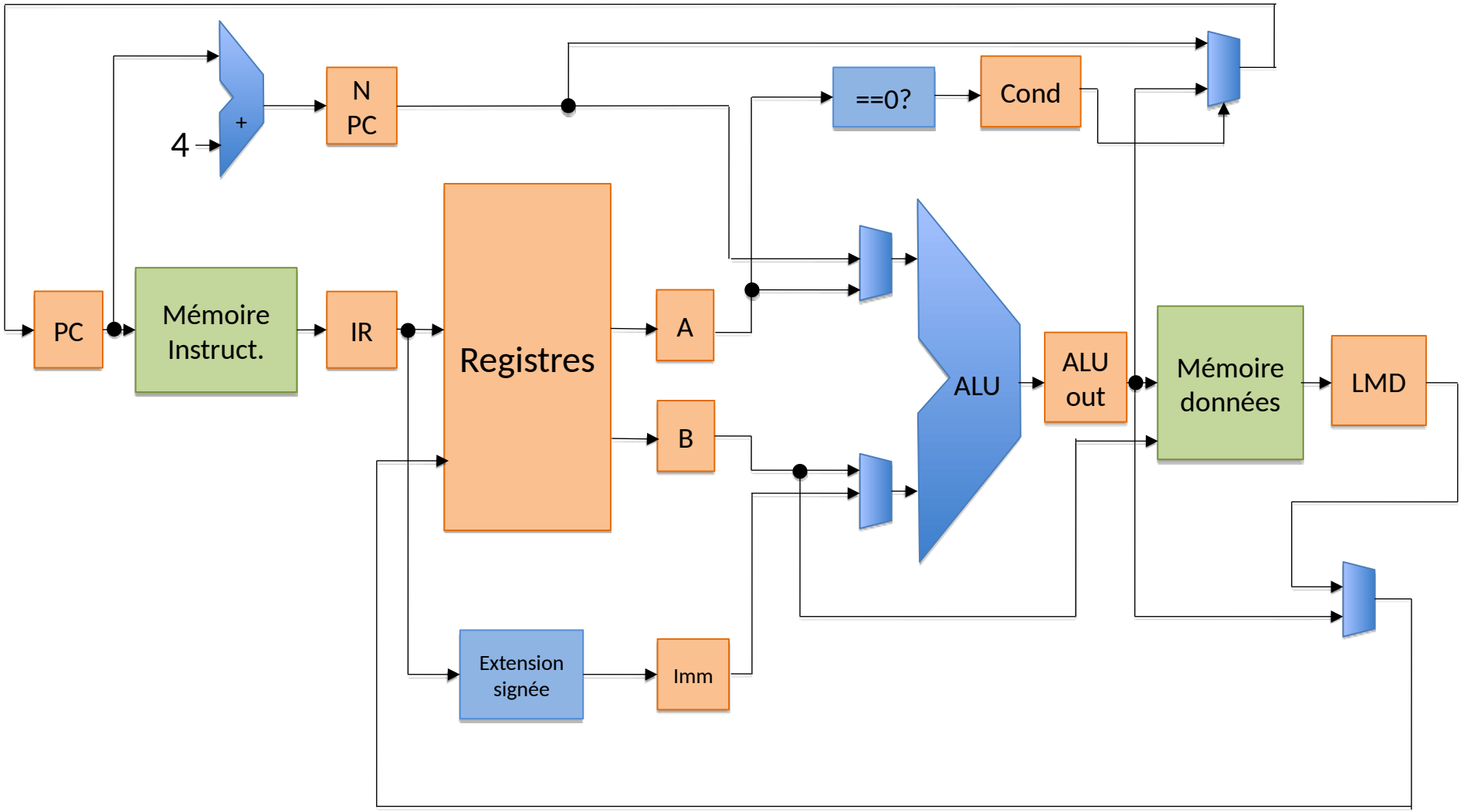
- Séparation en IM/DM ev. réalisée par cache
- Ecriture / lecture registres en demi-cycles différents

Cas général

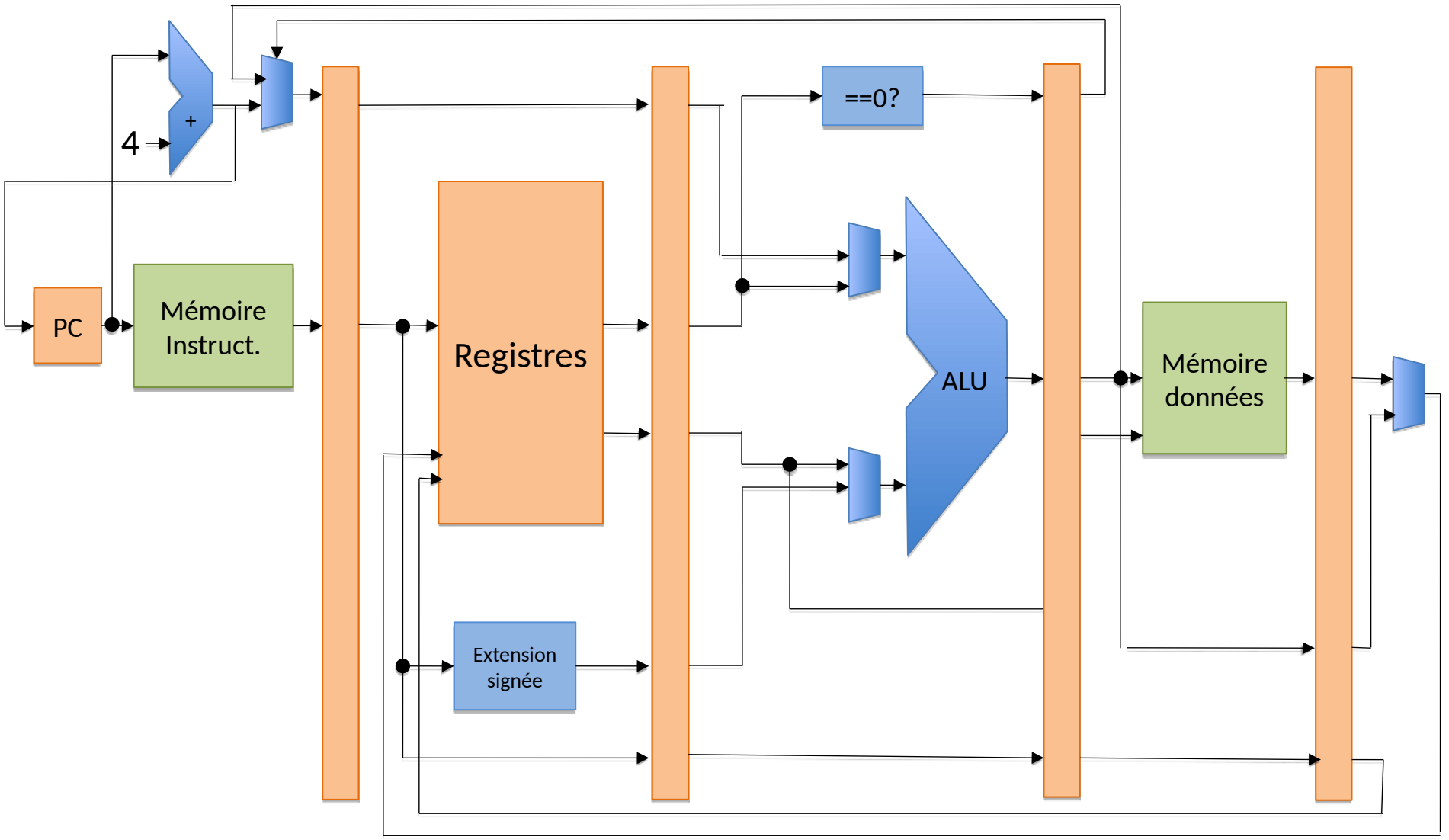


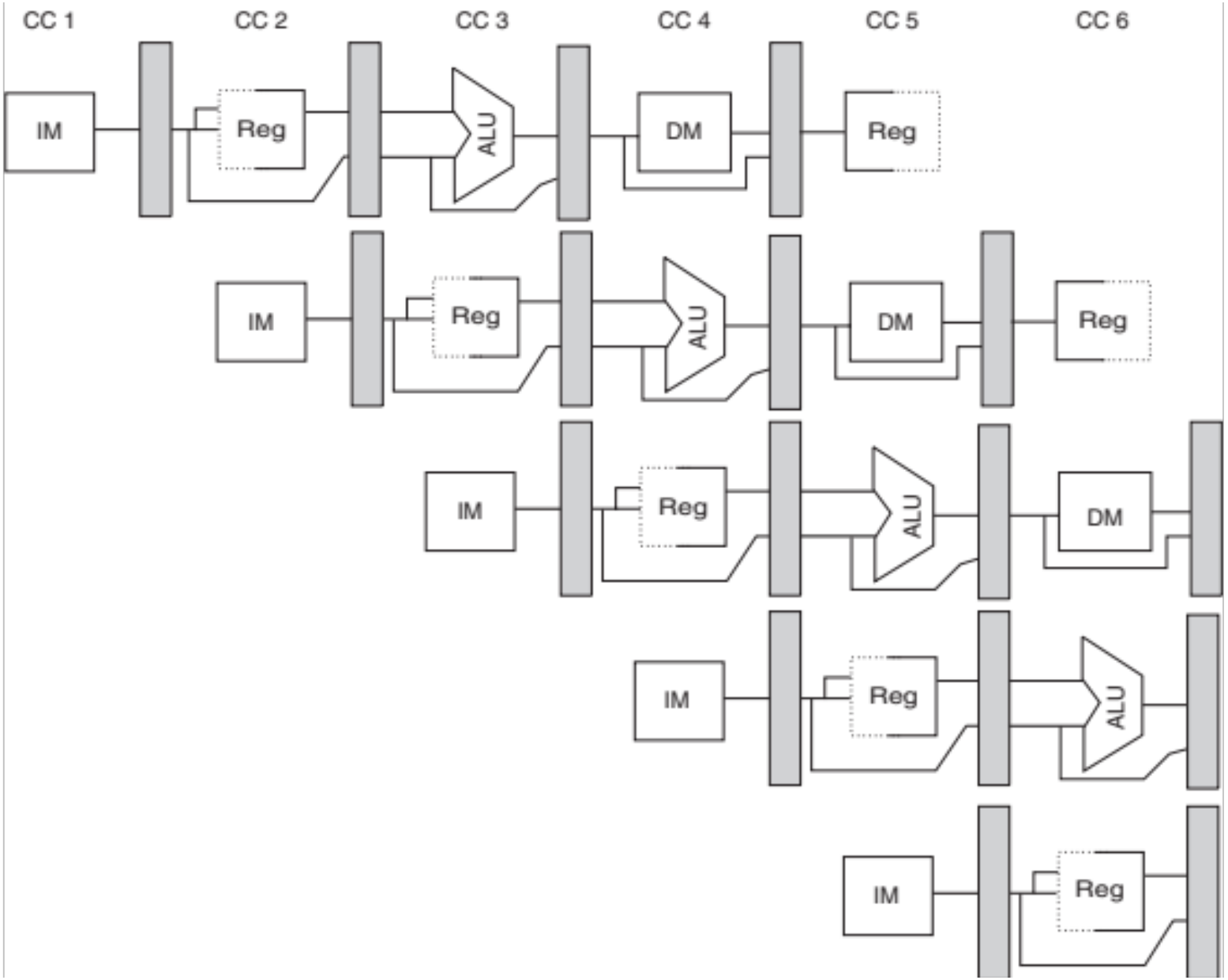
Quelques problèmes persistent … à traiter.

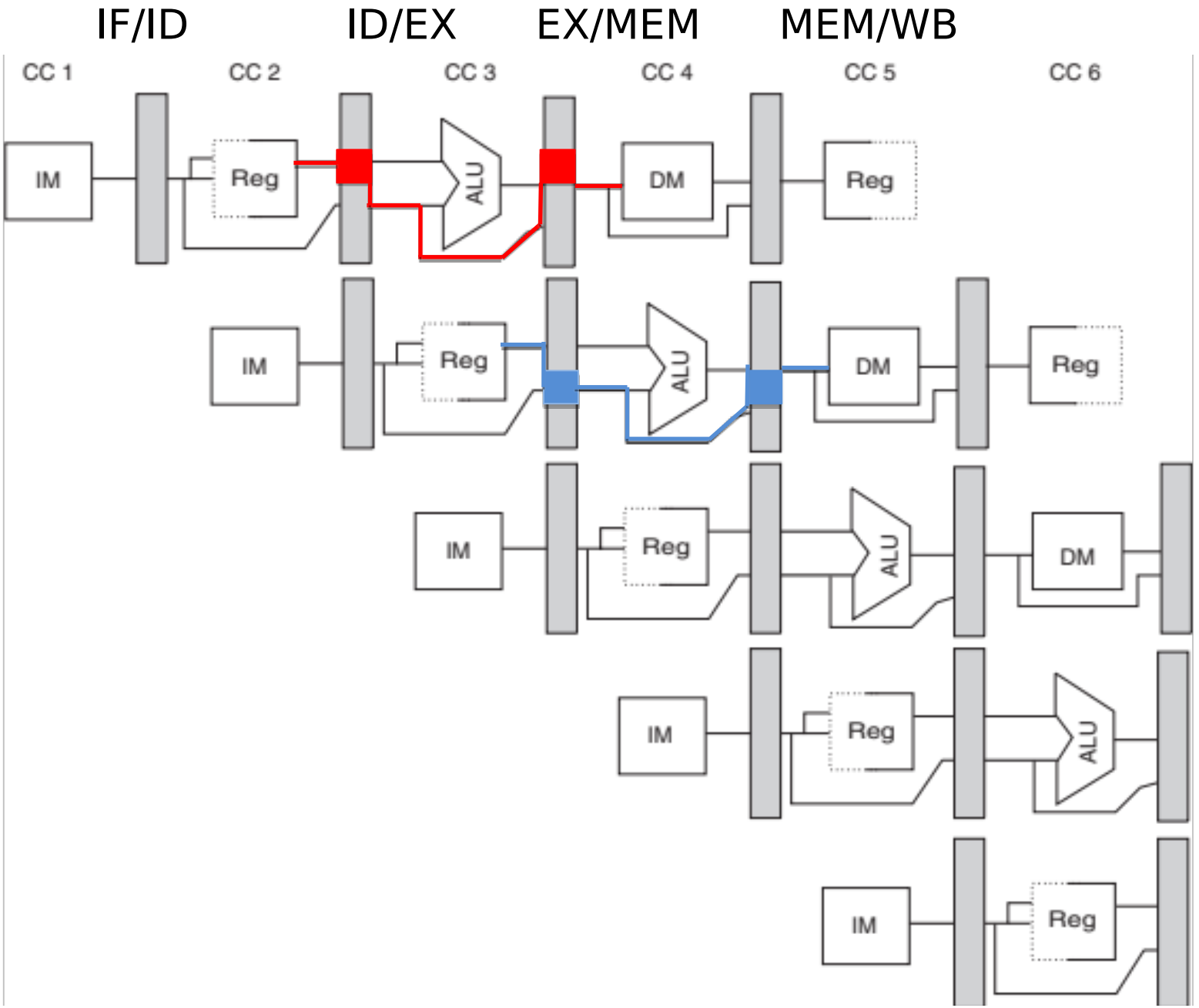
Rappel : chemin de données



Registres supplémentaires




CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
IM
Reg
ALU
DM
Reg
IM
Reg
ALU
DM
Reg
IM
Reg
ALU
DM
IM
Reg
ALU
IM
Reg


IF/ID
ID/EX
EX/MEM
MEM/WB
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
IM
Reg
ALU
DM
Reg

Premier bilan : limitations

- Le débit augmente, mais ...
- ... aucune instruction sera exécutée plus rapidement!

Limitations :

- Latence
- Problèmes d'équilibre entre les étapes (l'étape la plus lente détermine la fréquence)
- Surcout du à la gestion du pipeline (registres etc.)
- Dépendances entre les instructions

Conflits

3 types de conflits

- Structurels : quand il n'y a pas assez de matériel pour toutes les combinaisons d'instructions

- Liés aux données : quand une instruction utilise un résultat d'une instruction précédente (encore en exécution)

- Liés aux contrôle : quand le PC est modifié par une instruction

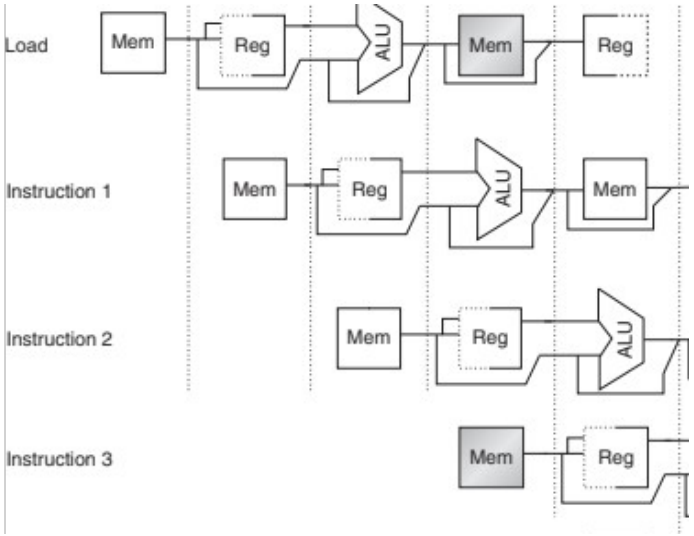
« Solution » :

  – raccourcir les chemins, ou
  – suspendre l'activité d'une partie du pipeline

Conflits liés à la structure

Exemples :

- Un seul chemin d'accès à la mémoire

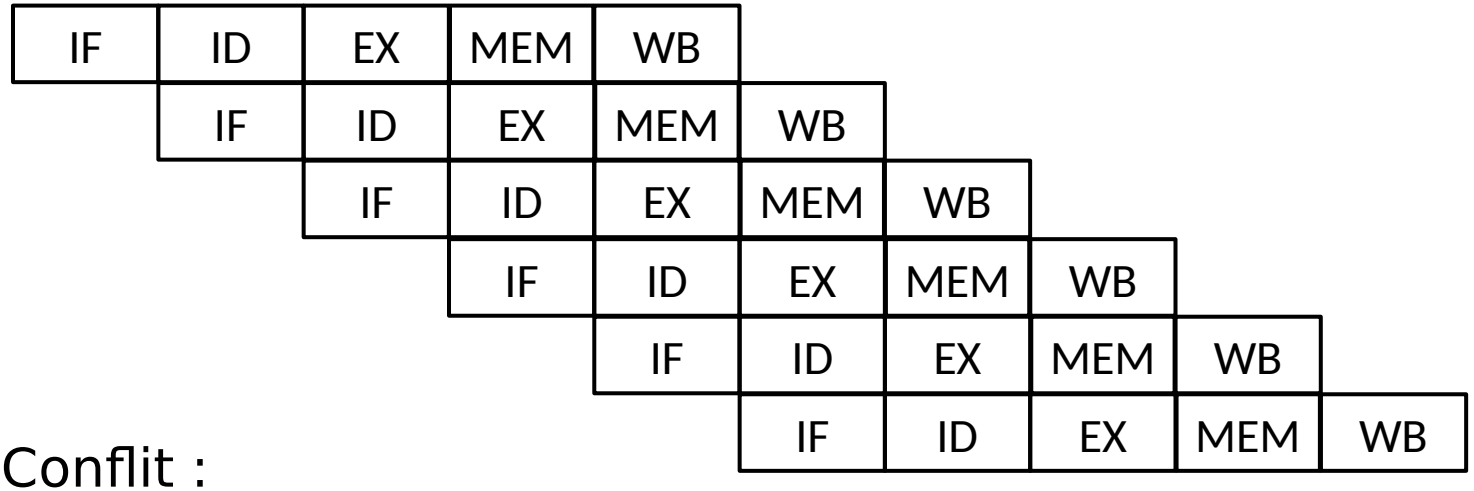


- Opérations A/L plus longues qu'un cycle (float?)

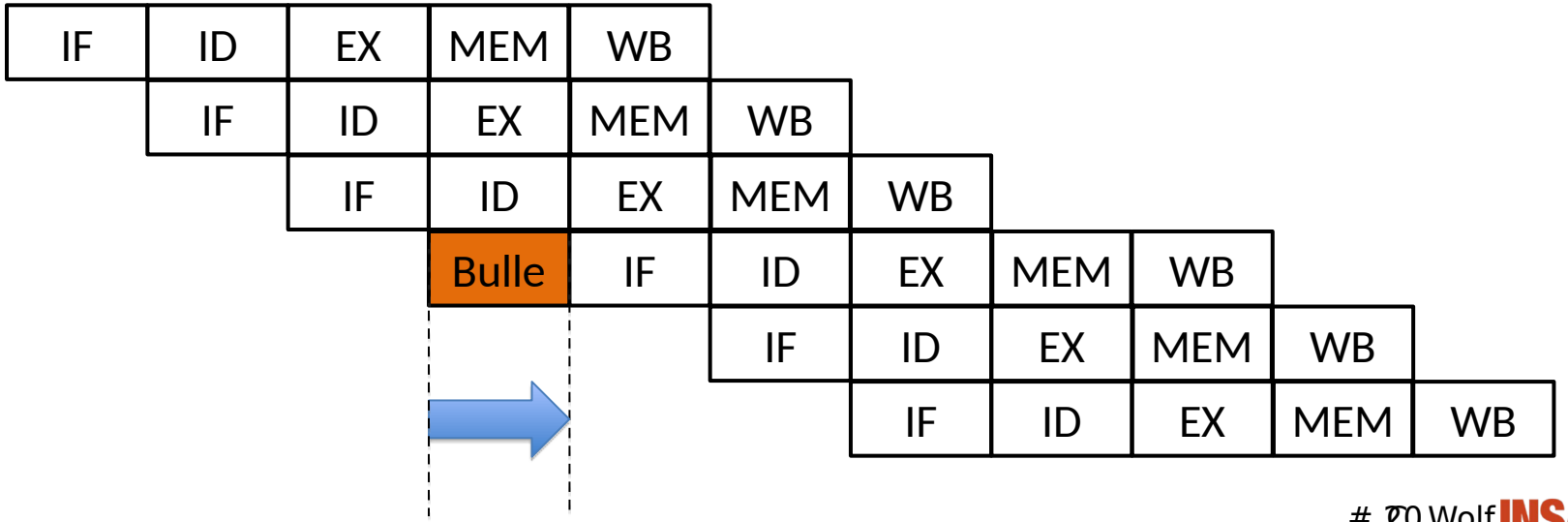
IF	ID	EX	EX	EX	MEM	WB

Les bulles

Exécution optimale :



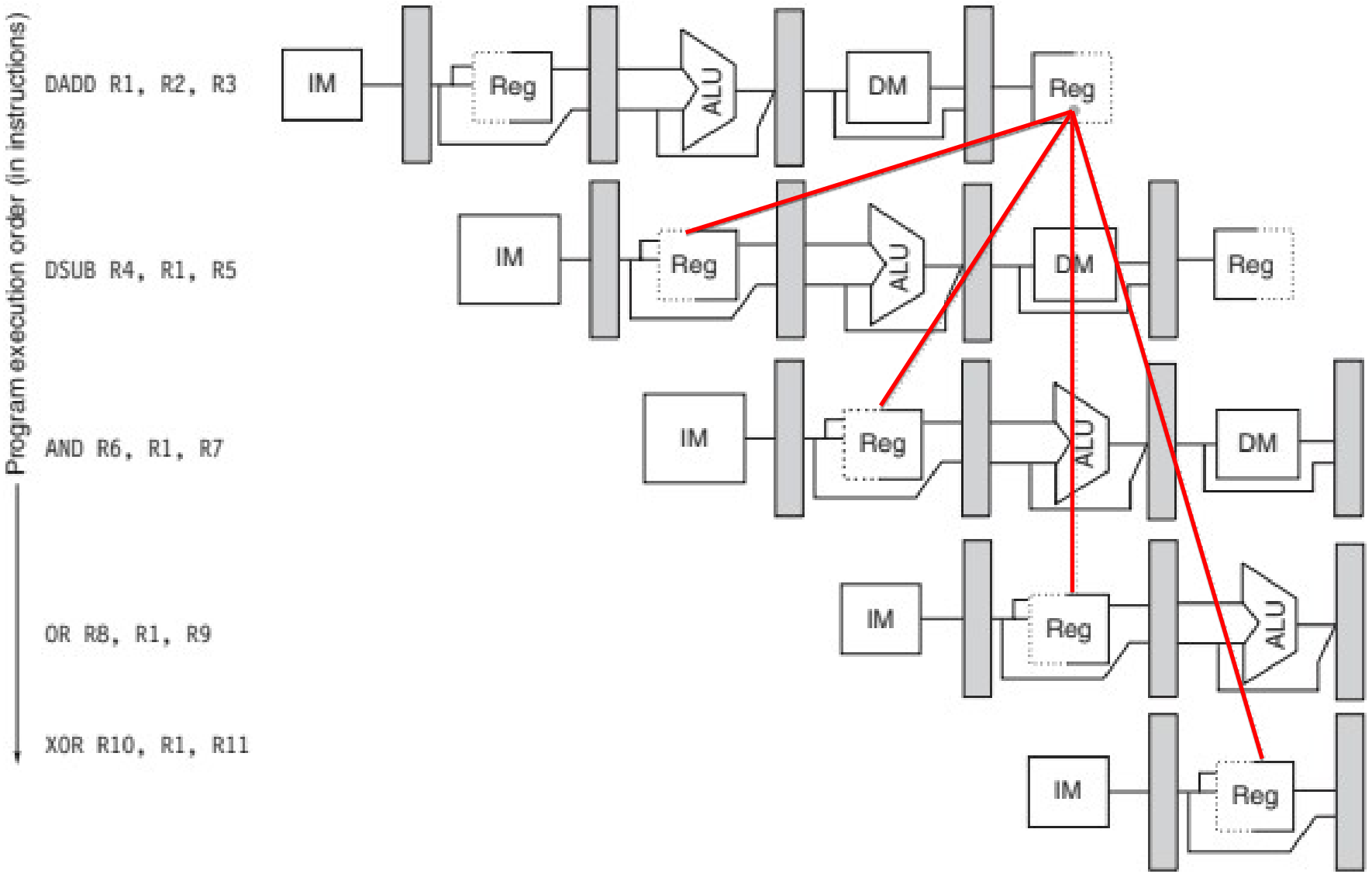
Bulle / Conflit :



Conflits liés aux données (1)

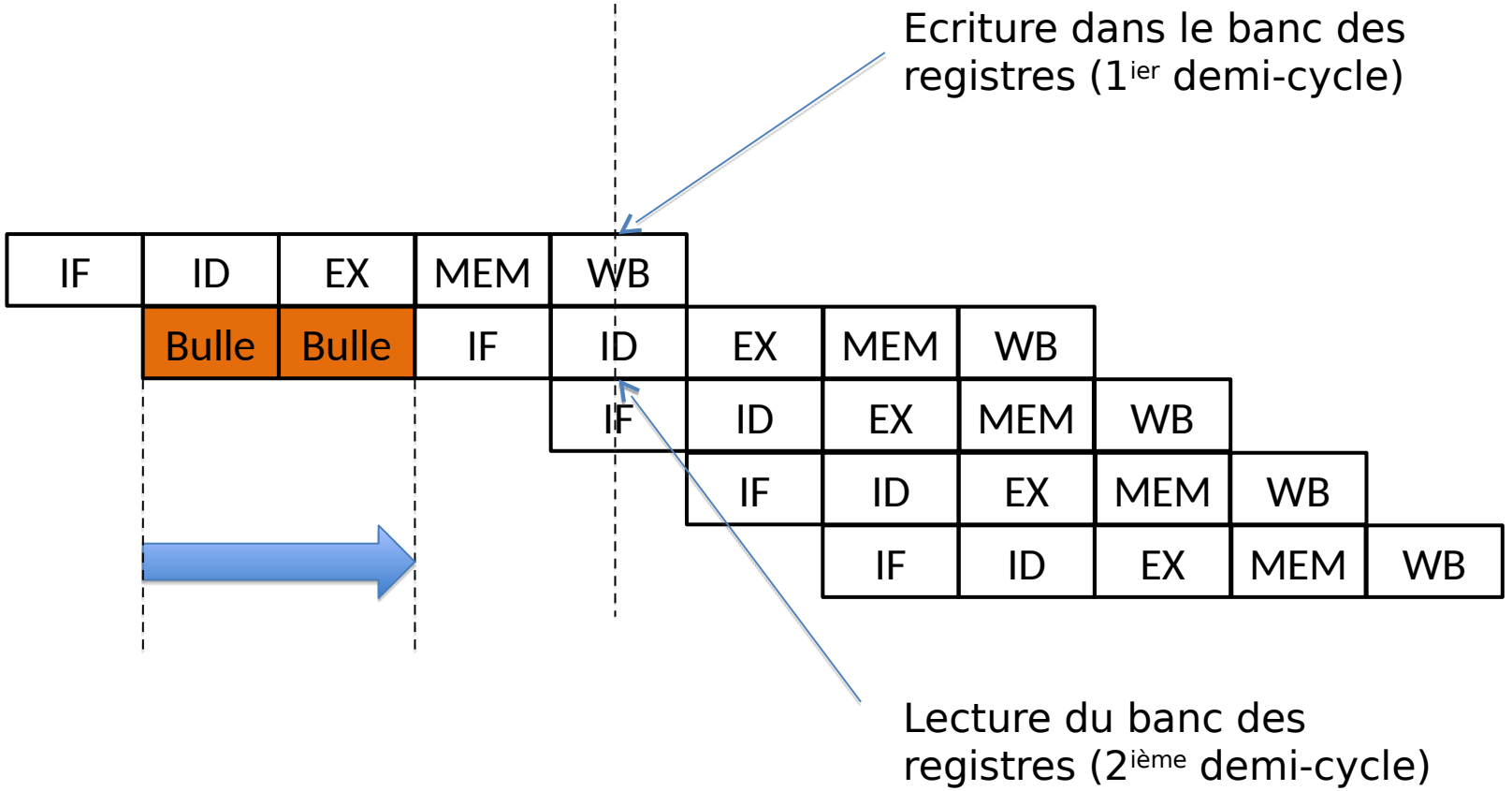
```
add r1, r2, r3
sub r4, r1, r5
and r6, r1, r7
or r8, r1, r9
xor r10, r1, r11
```

Toutes les opérations après « add » utilisent son résultat.



Instruction « OR » : ok si lecture en deuxième demi-cycle

Bulles (sans contre-mesures)



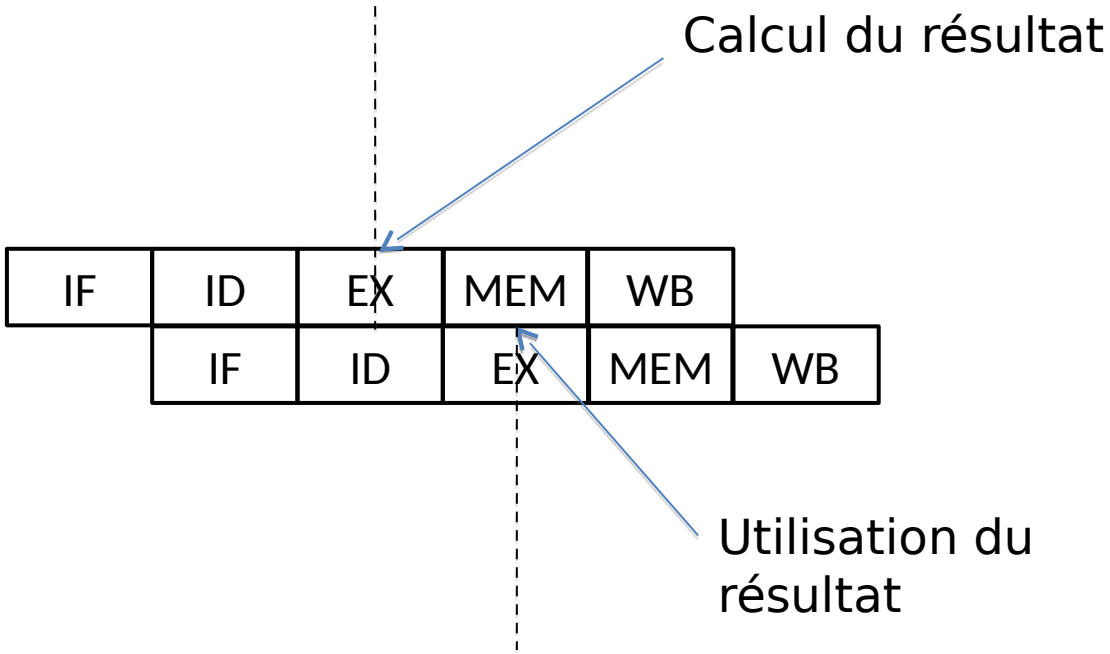
Fin de l’insertion dans ARC

ARC 2018 → Retour aux slides de TRI

- Le reste des slides de CWO détaille les optimisation proposées par Hennessy et Patterson pour éviter les bulles
- Livre en ligne « Computer Organization and Design: The Hardware/Software Interface » :

http://ac.aua.am/Arm/Public/2017-Spring-Computer-Organization/Textbooks/ComputerOrganizationAndDesign5thEdition2014.pdf

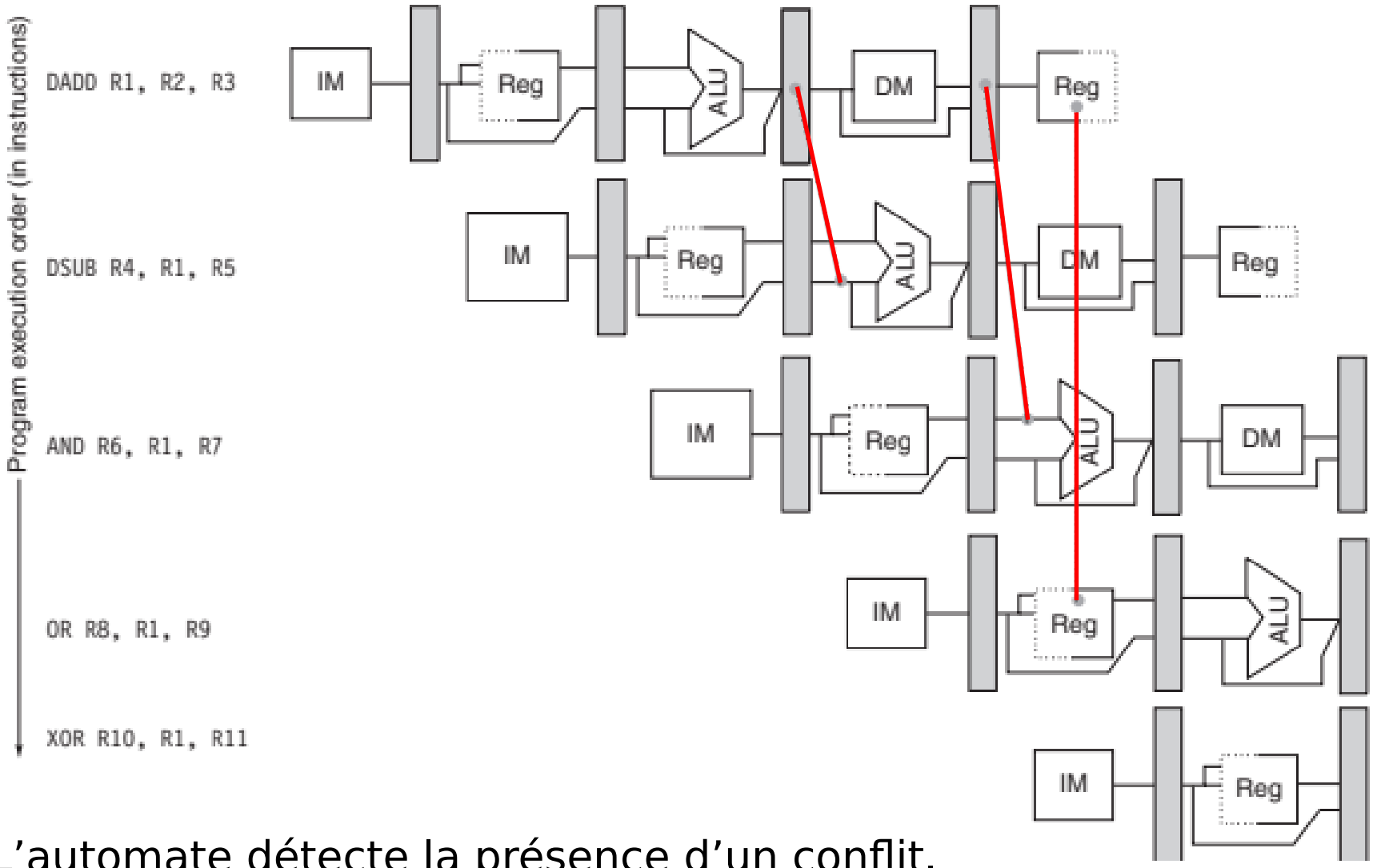
Bulles : contre-mesures?



Il n'y a pas de raison « théorique » pour un conflit, car la valeur demandée et déjà disponible.

Elle n'a juste pas encore été écrite dans le banc des registres

Transmission directe de valeurs (1)



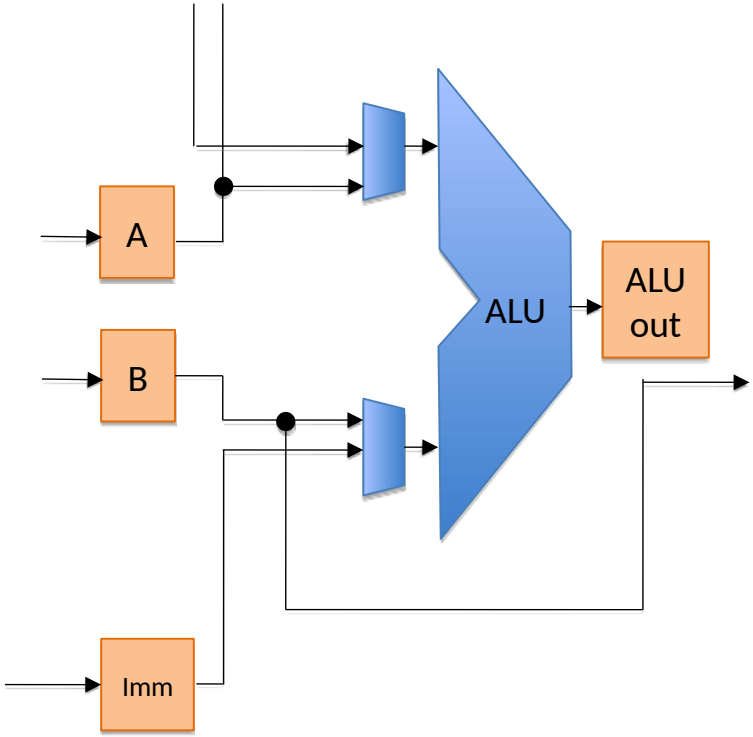
L'automate détecte la présence d'un conflit.

Choix de la source de calcul.

Pipeline : implémentation

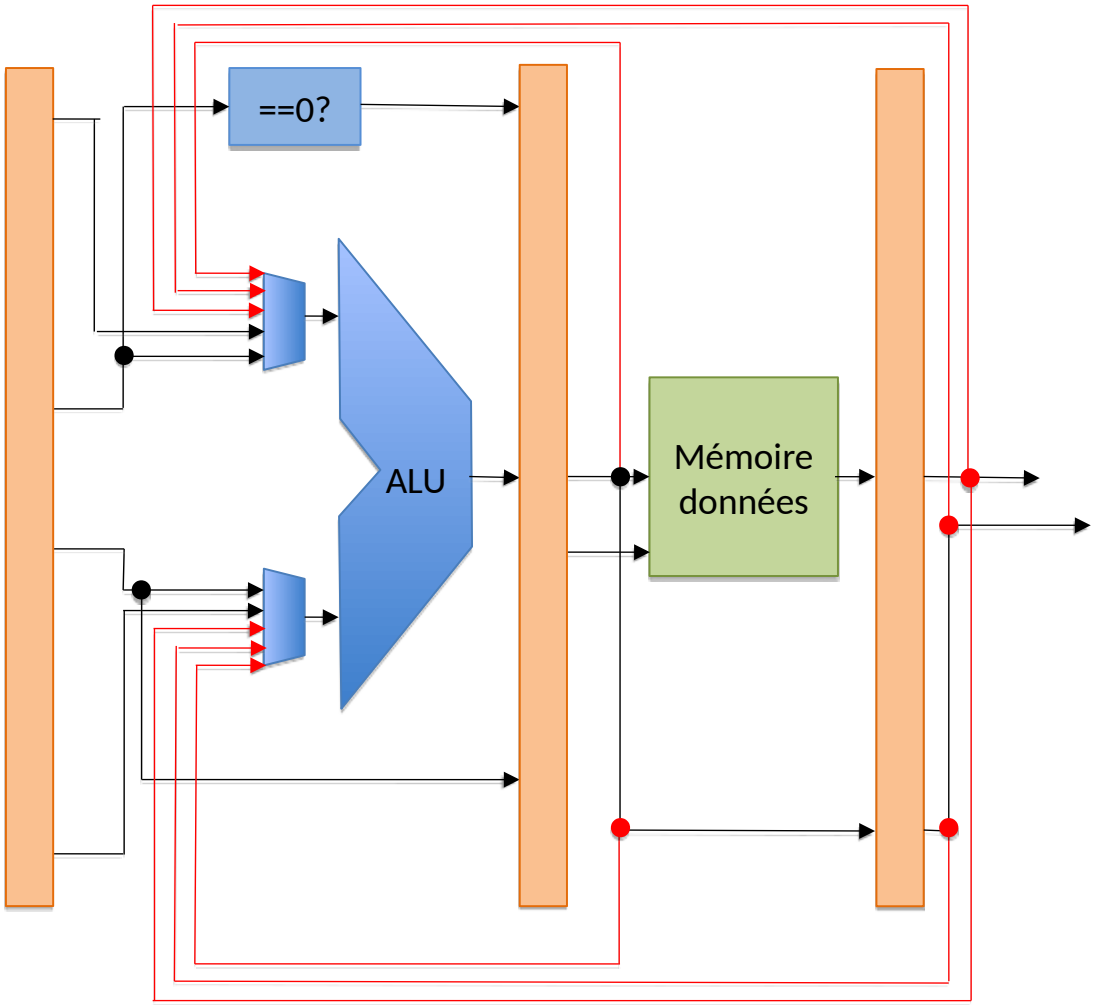
Rappel : version non pipelinée

Deux entrées pour chaque MUX devant l'ALU



Pipeline : implémentation

La transmission directe des valeurs demande des entrées supplémentaires.

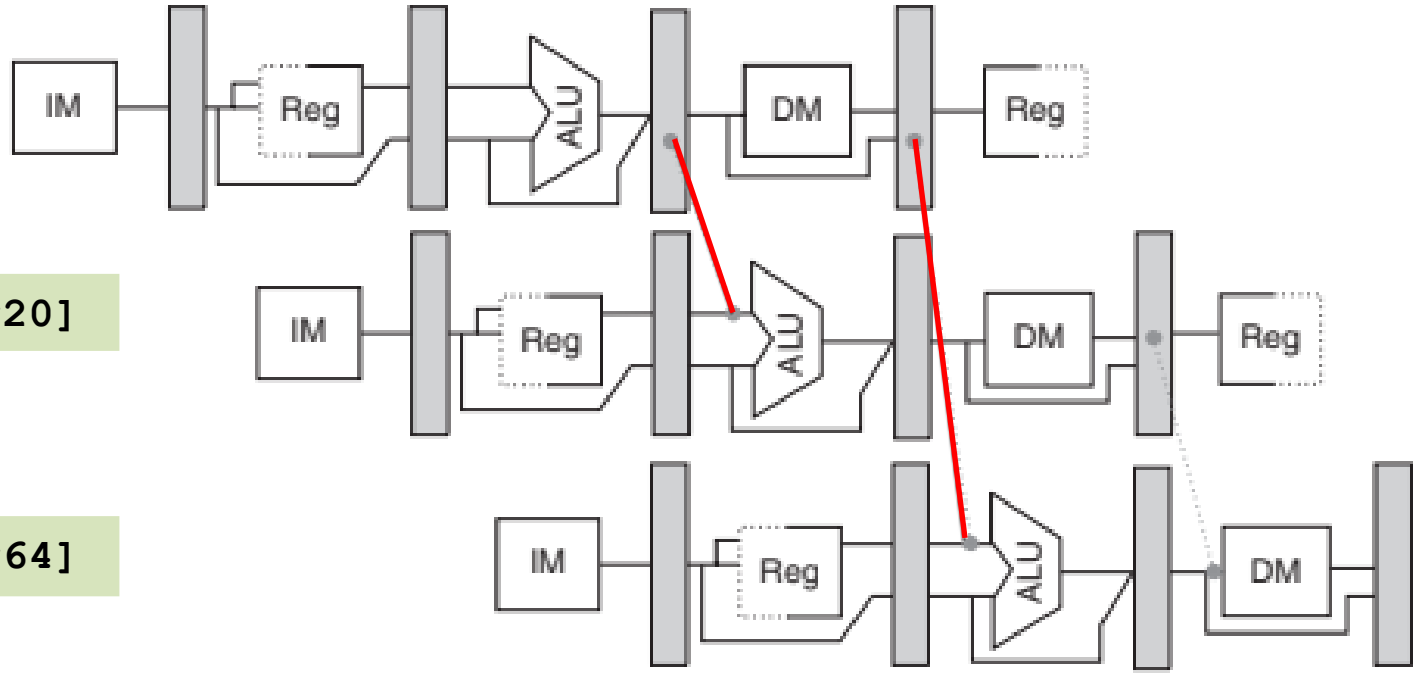


Conflits liés aux données (2)

```
add r1, r2, r3
ldr r4, [r1, #20]
str r4, [r1, #64]
```

Transmission directe de valeurs (2)

```
add r1,r2,r3
ldr r4, [r1, #20]
str r4, [r1, #64]
```



Conflits liés aux données (3)

```
ldr   r1,[r2,#0]
sub   r4, r1, r5
and   r6, r1, r7
or    r8, r1, r9
```

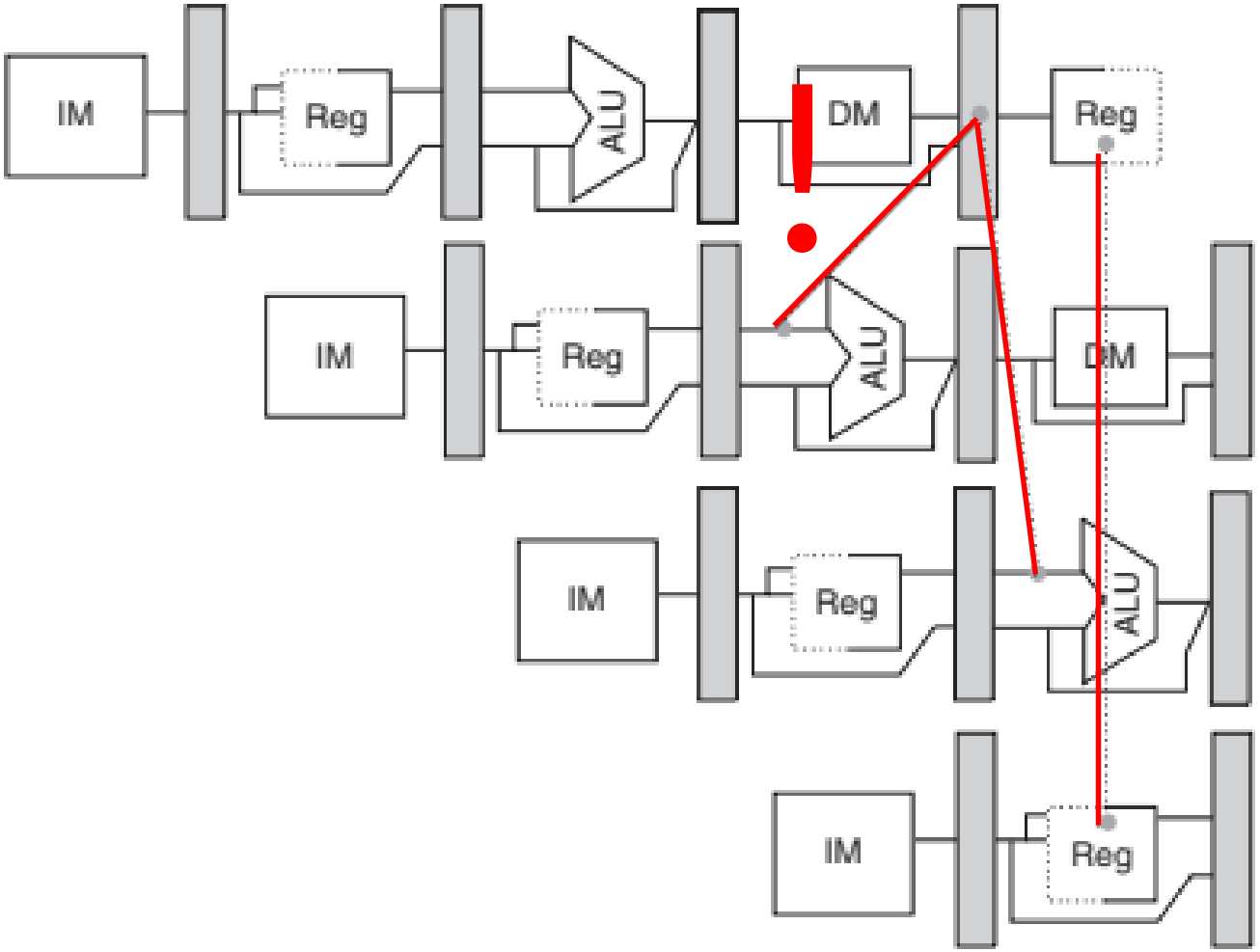
Conflit réel lié aux données

```
ldr r1,
[r2,#0]
```

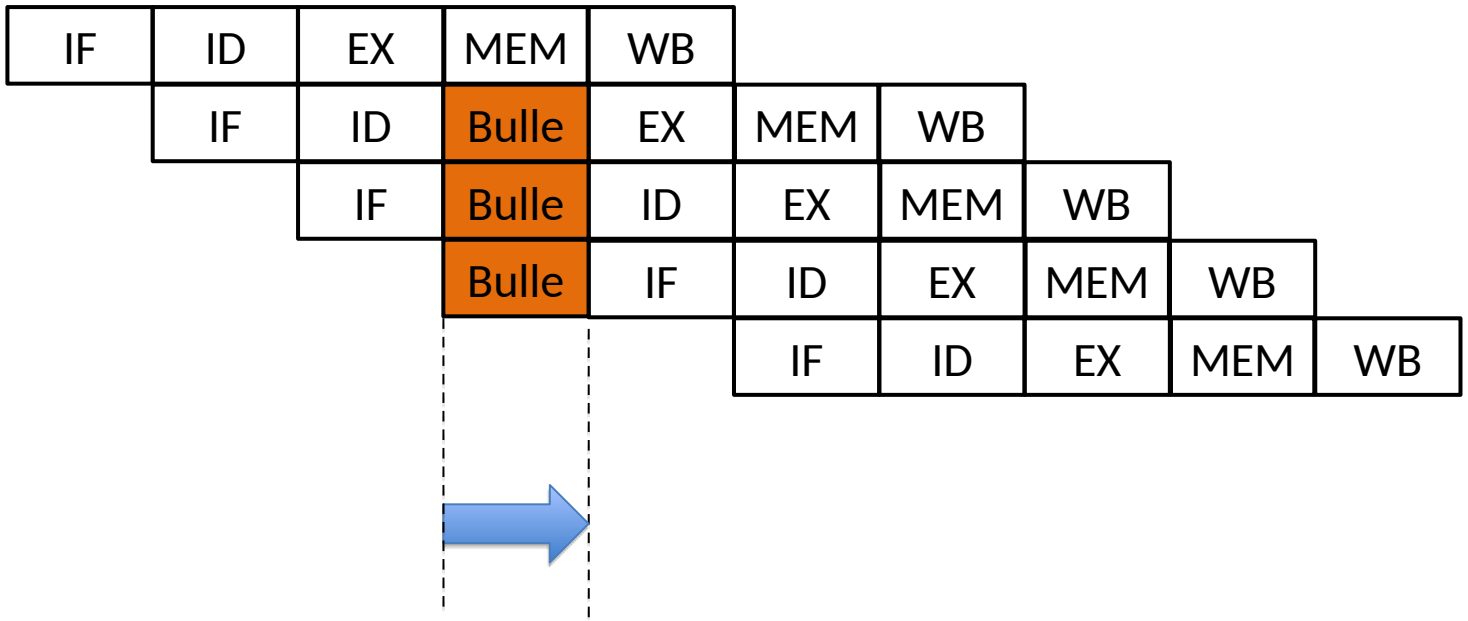
```
sub r4, r1, r5
```

```
and r6, r1, r7
```

```
or  r8, r1, r9
```



Bulle


IF ID EX MEM WB
IF ID Bulle EX MEM WB
IF Bulle ID EX MEM WB
Bulle IF ID EX MEM WB
IF ID EX MEM WB

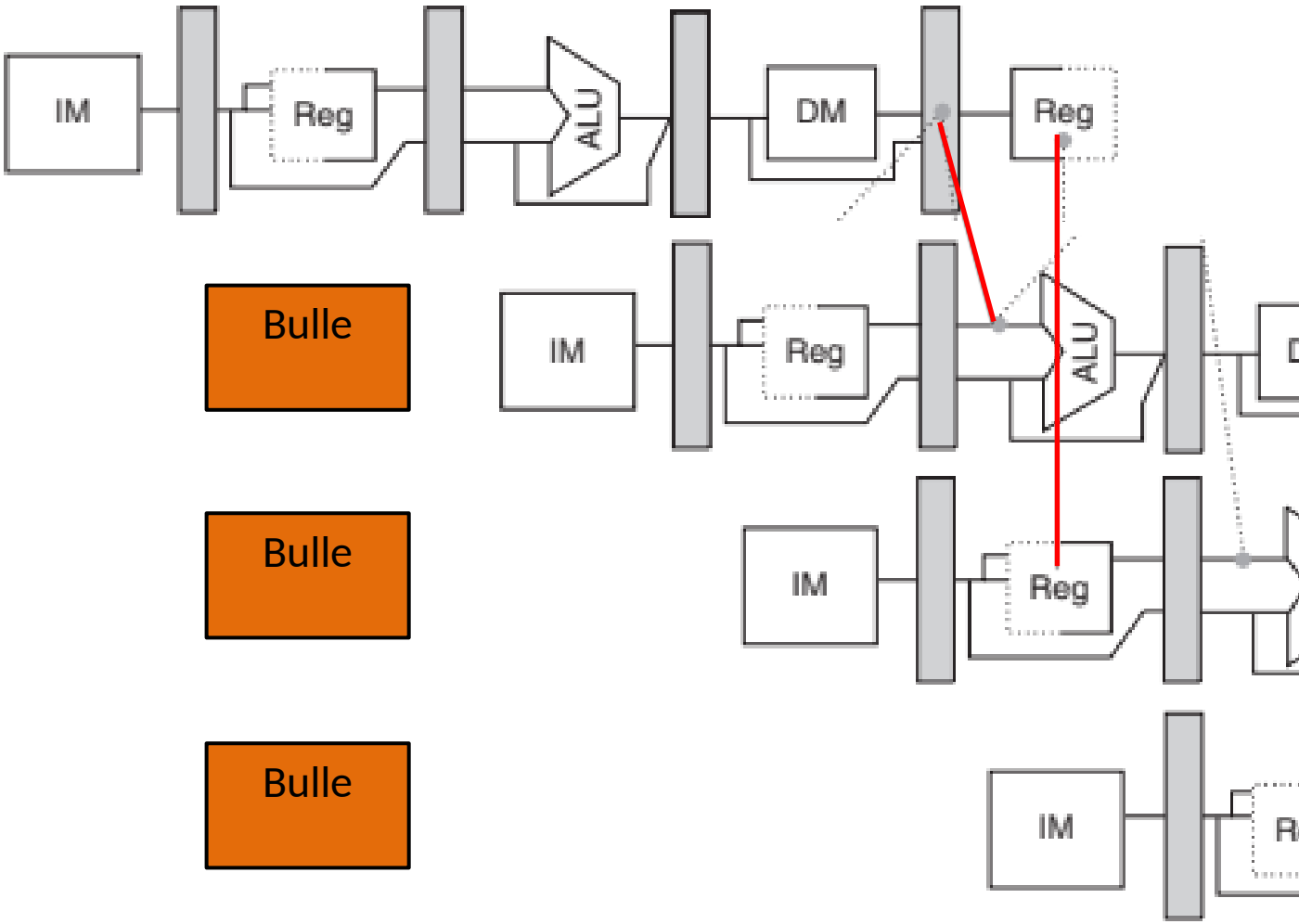
Conflit réel lié aux données

```
ldr r1,
[r2,#0]
```

```
sub r4, r1, r5
```

```
and r6, r1, r7
```

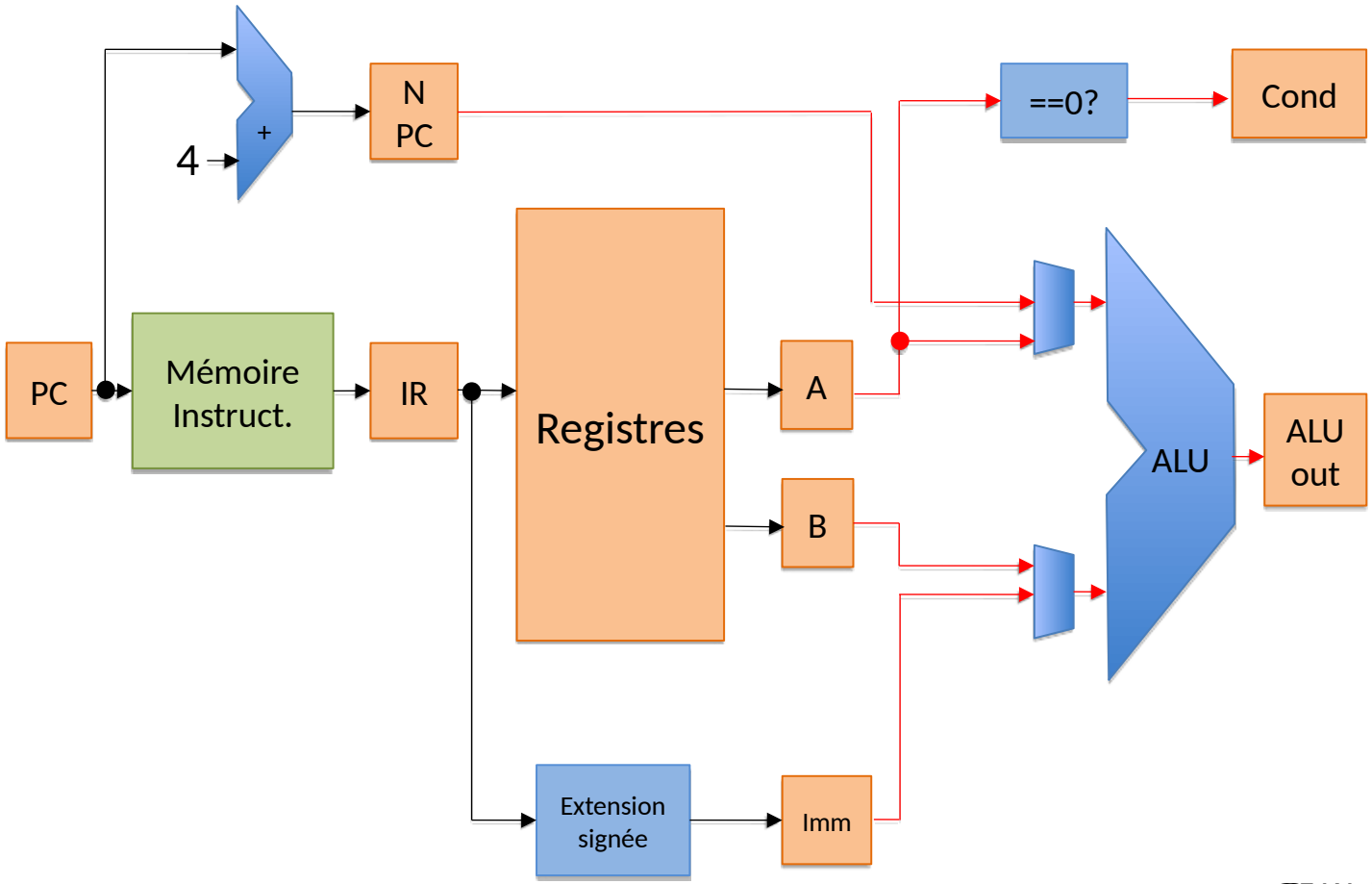
```
or  r8, r1, r9
```



Conflits liés aux branchements

```
bne .L1
add r1, r2, r3
.L1:
```

La destination du saut est
- disponible en fin de cycle « EX »
- écrit dans le PC en fin de cycle « MEM »

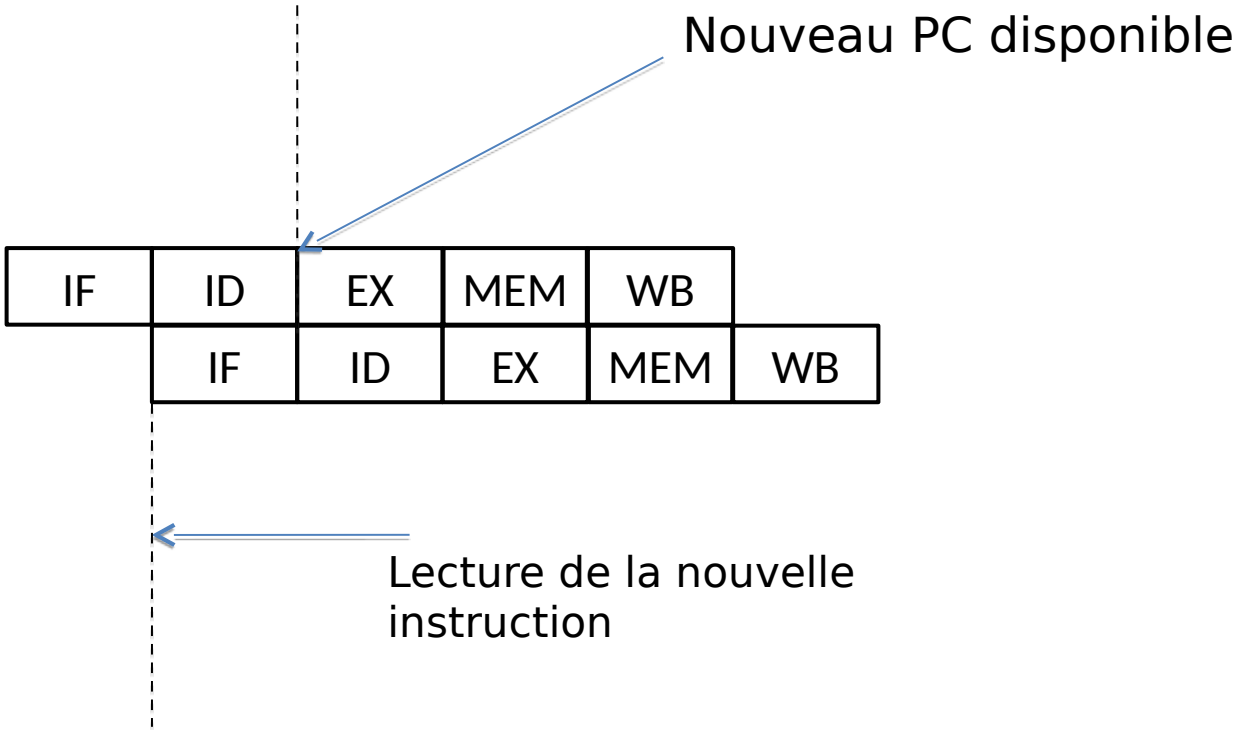


Conflits liés aux branchements

```
bne .L1
add r1, r2, r3
.L1:
```

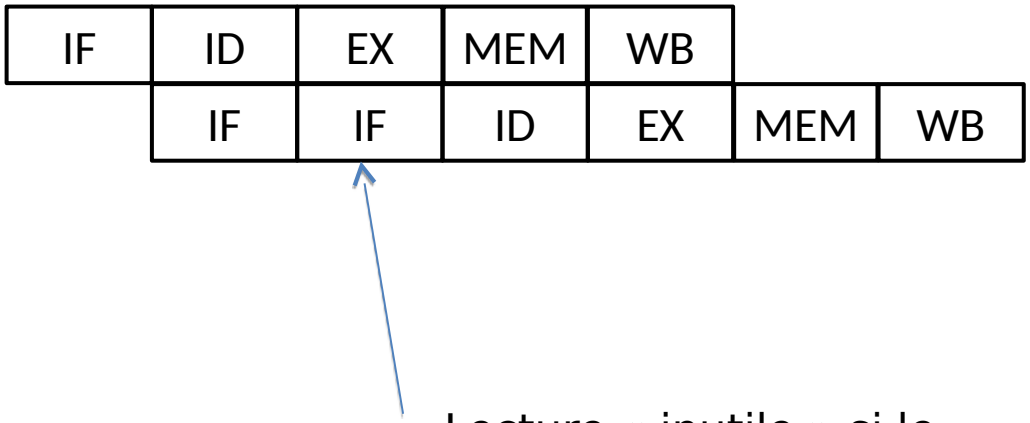
Une implémentation « très agressive » permet d'obtenir le saut en fin de cycle « ID » (non détaillé ici).

Il reste donc un problème …



Solution (2) : simple

Systématiquement répéter le cycle « IF » après un branchement.



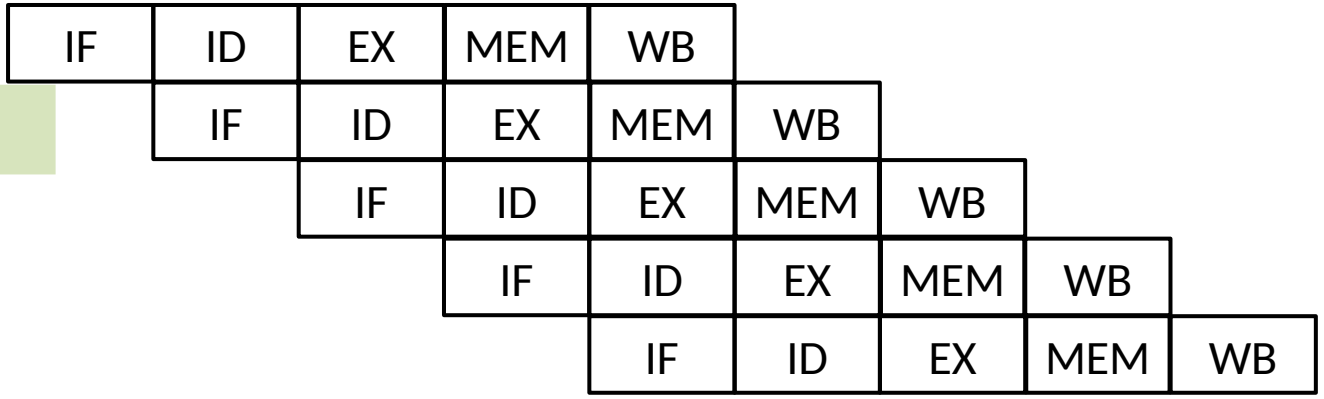
Lecture « inutile » si le branchement a été effectué.

Equivalent à une bulle.

Solution (2) : prédiction constante

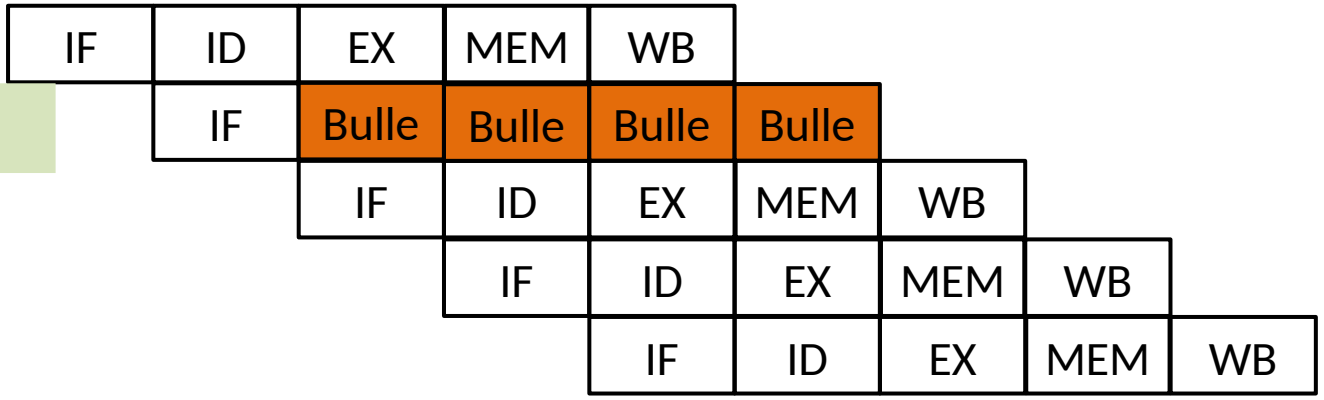
Branchement non-exécuté :

```
bne .L1
add r1, r2, r3
```



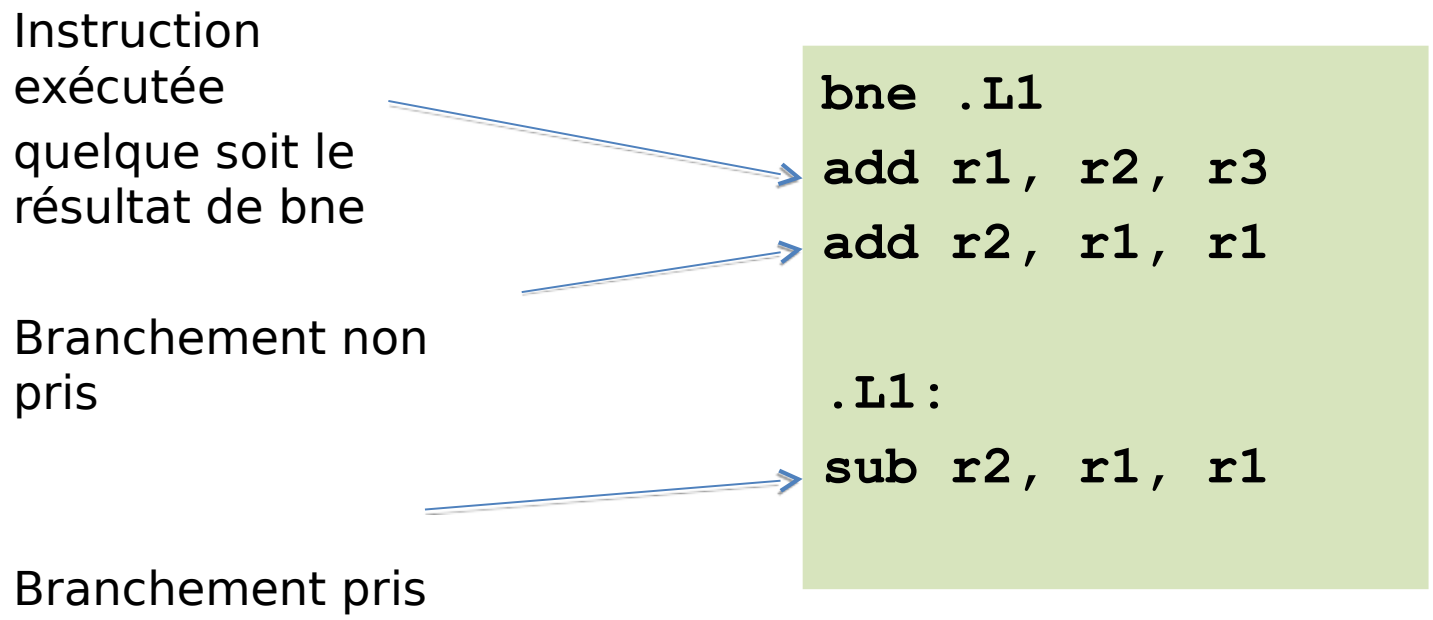
Branchement exécuté :

```
bne .L1
add r1, r2, r3
.L1:
```



Solution (3) : retardement

Systématiquement configurer une instruction entre l'instruction du branchement et son exécution :



Solution (3) : retardement

Le compilateur optimisera le code pour placer des instructions « utiles » dans le créneau de retardement.

```
add r1, r2, r3
Cmp r4, r5
bne .L1
<créneau>

.L1:
```

```
Cmp r4, r5
bne .L1
add r1, r2, r3

.L1:
```

Remarques : les processeurs actuels utilisent un pipeline « *out-of-order* » où les instructions sont réordonnées pour maximiser la performance.

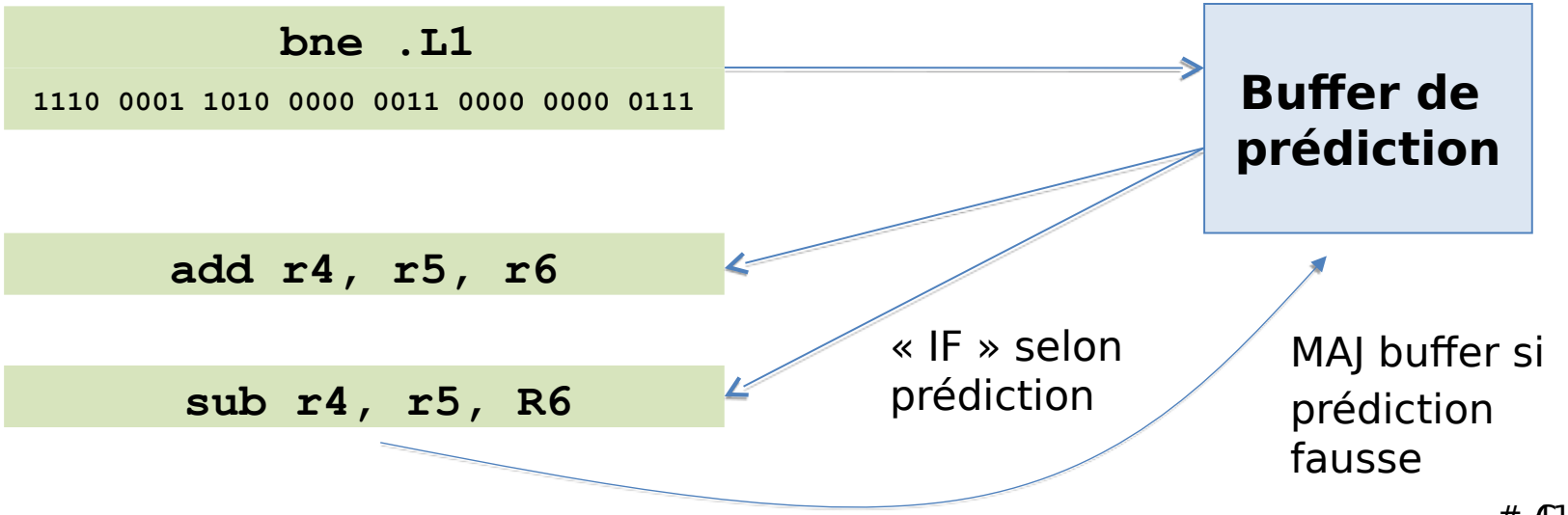
Prédiction dynamique

Pour certaines architectures avec des pipelines profonds, les branchements peuvent couter plusieurs cycles.

Solution : tentative de prédire la destination du saut.

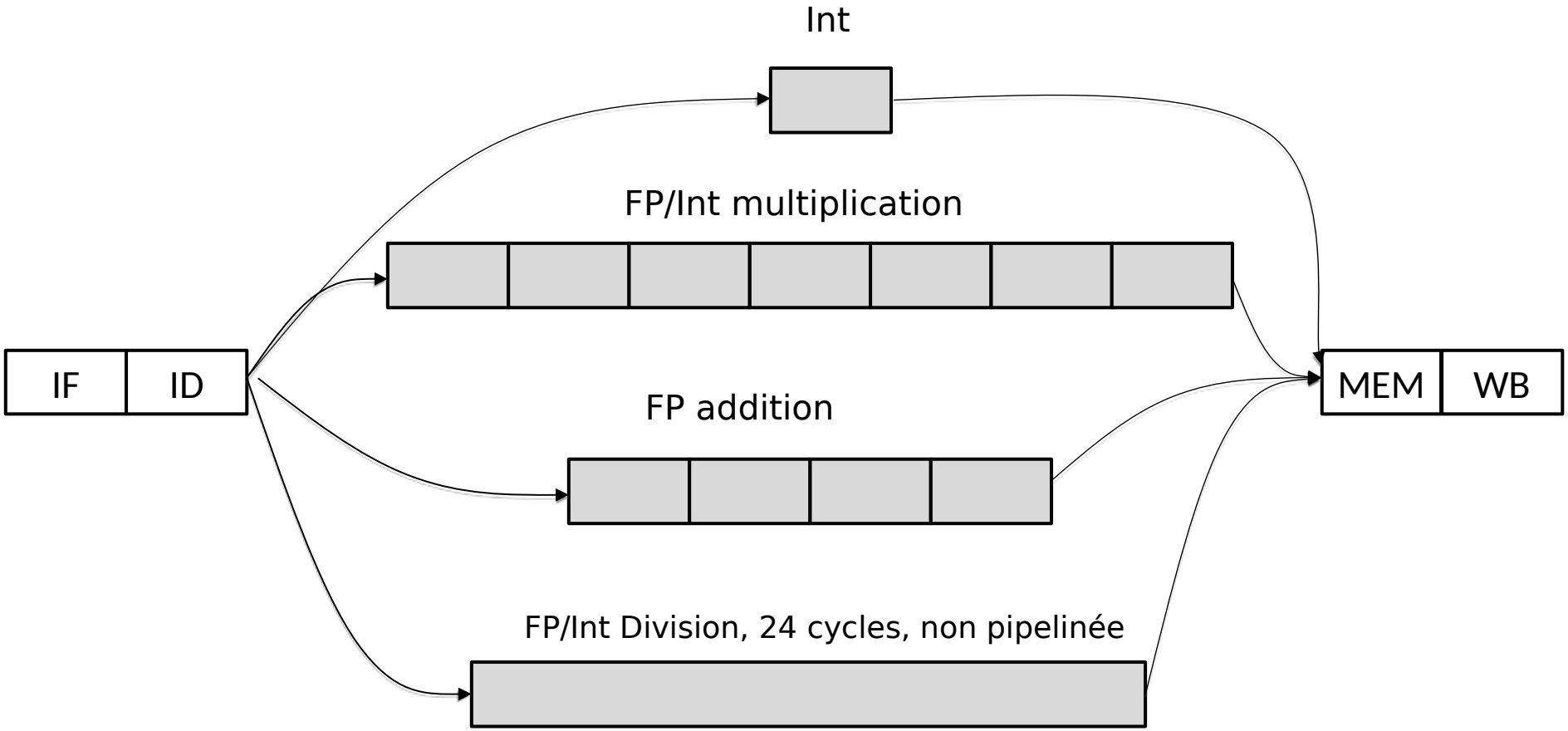
Un tableau sauvegarde si un branchement a été pris précédemment pour une adresse donnée.

Plusieurs adresses partagent une entrée, selon leurs bits de poids faible.



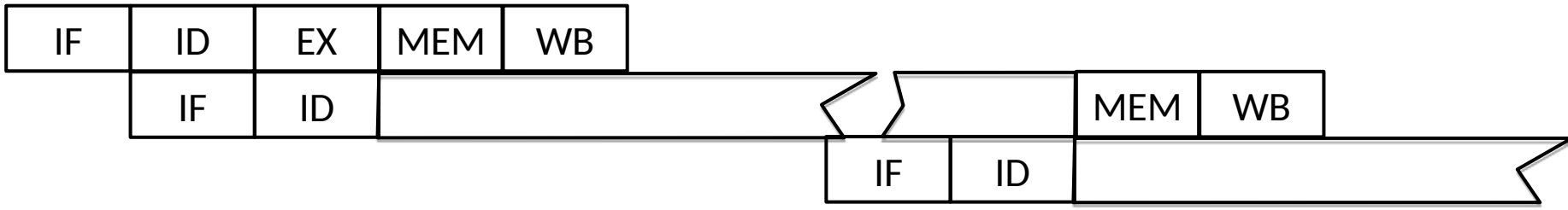
Les opérations multi-cycles

Certaines opérations durent plusieurs cycles, par exemple les opérations sur les nombres en virgules flottants



Les opérations multi-cycles

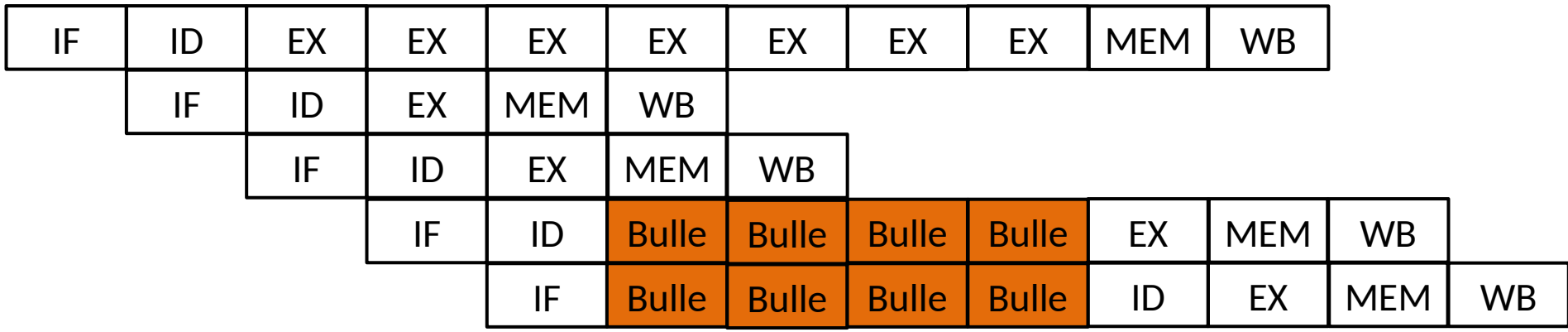
Les opérations non-pipelinées peuvent causer des bulles de durées très importantes.



Les opérations multi-cycles

Les opérations peuvent terminer dans un ordre différent

Problèmes de lecture après écriture



Problème des interruptions

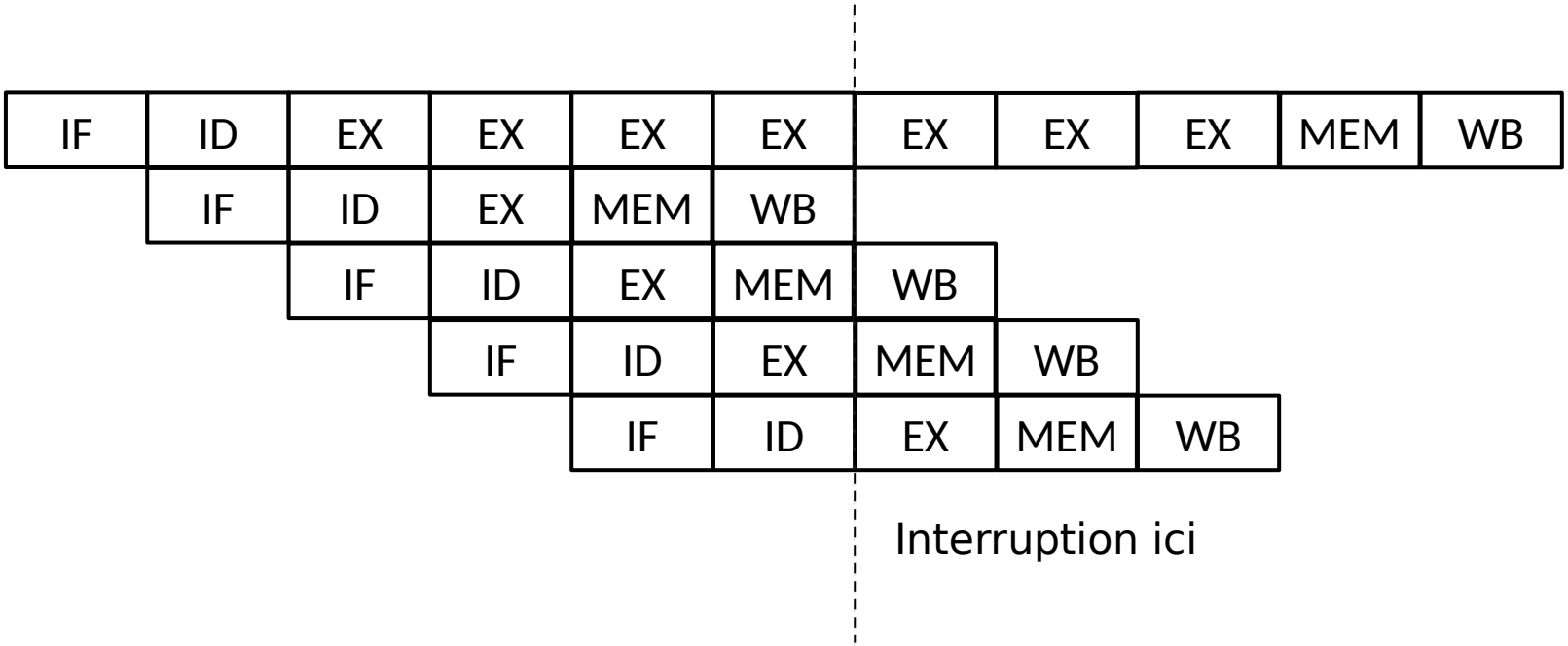
- Réactions sur des évènements
  - internes (exceptions mathématiques, accès mémoire interdite)
  - externes (matériel, changement de tâche etc.)
- Le programme actuel est interrompu, suivi par l'appel d'une sous-routine
- Des l'arrivée d'une interruption, toutes les écritures (mémoire, registres) sont désactivés.
- Les instructions non-terminées seront redémarrées après le retour de la routine d'interruption.

Interruptions et multi-cycles

Une instruction a déjà terminée, alors qu'une instruction précédente est encore en cours d'exécution.

Si les « interruptions précises » sont demandées, il faut absolument éviter ce cas de figure.

Solution : retard des écritures



Bilan : développement historique

- 1980 : les pipelines sont utilisées dans les super-ordinateurs uniquement
- ~1985 : arrivée des pipelines dans les micro-processeurs (desktop)
- Maintenant : les micro-controleurs à 1-2€ sont pipelinés