Cours

Processeur - Pipeline

Plan du chapitre

1. *Introduction.*
2. *Les pipelines.*
3. *Les conflits dans un pipeline et leur résolution.*
4. *Principes d'ordonnancement pour un pipeline.*

❑ **Les mots importants sont :**

- **<u>Parallèle</u> qui a trait à <u>l'espace;</u>**
- **<u>Simultané</u> qui a trait au <u>temps</u>.**

☞ Les vocabulaires de l'espace et du temps sont étroitement liés comme le relevait déjà Bergson**, juxtaposition et succession**.

☞ En informatique, il est question de parallélisme et de simultanéité.

✓ **Le parallélisme est relatif à l'espace;**

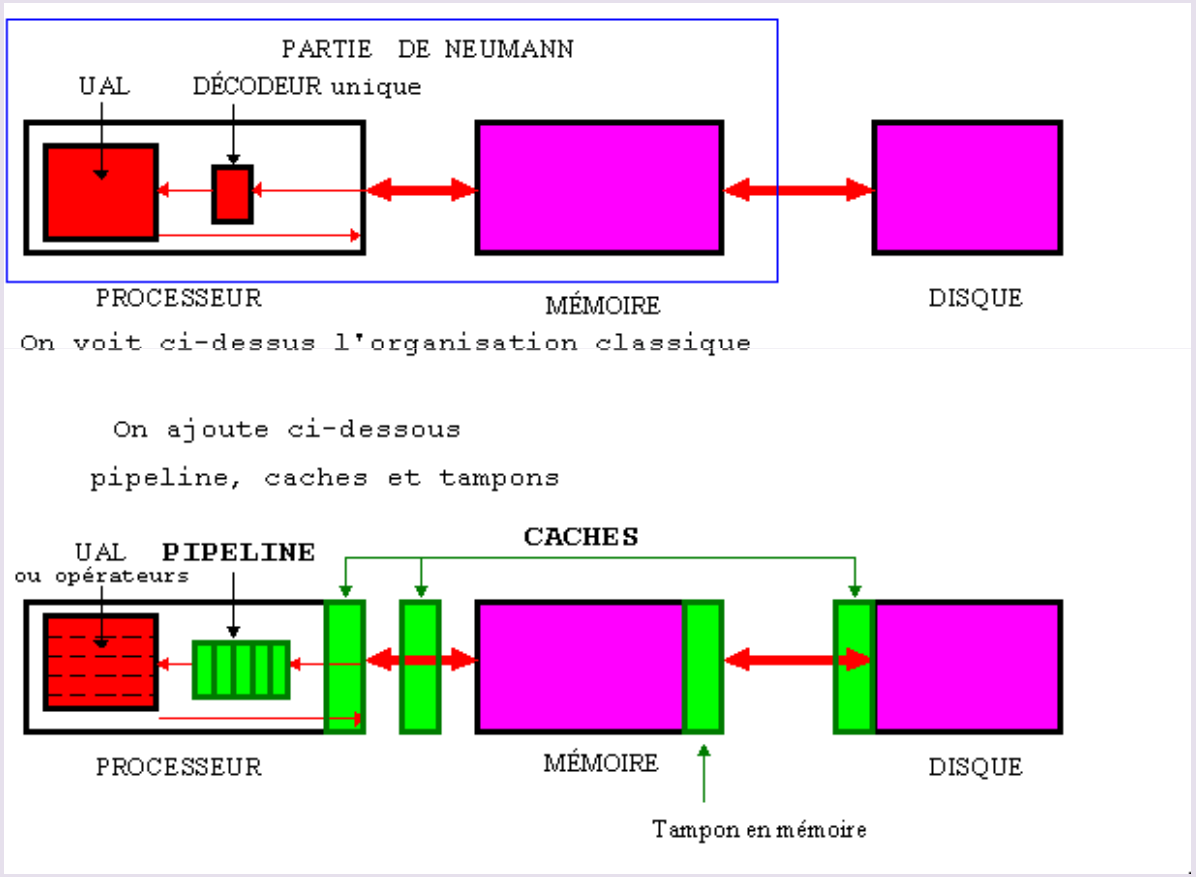
✓ **La simultanéité est relative au temps;**

❑ **Pipelines, caches et modèle de base** : Parmi les prescriptions de Neumann, on trouve :

- ♦ **La deuxième :**
  - ☞ « Le programme et les données sont logés dans une **même mémoire découpée en cellules.** »
  - • **On s'affranchit de cette règle par le schéma d'Aïken (⇔ architecture de Harvard) entre instructions et données et par la technique des caches entre les données.**
    - ➢ On est ainsi certain de maintenir la cohérence d'exécution par l'unicité de la mémoire de données.
- ♦ **La cinquième :**
  - ☞ « L'exécution de chaque instruction est achevée **avant que la suivante soit prise** en compte. »
  - • **On est ainsi certain d'éviter des collisions dommageables entre données.**
  - • **On s'affranchit de cette règle en installant un pipeline.**

• **Pour Info** : architecture de Harvard **est une conception de microprocesseurs qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chacune des deux mémoires s'effectue via deux bus distincts**

- Les places des deux techniques des caches et des pipelines sont montrées dans le schéma suivant :



- ***La simultanéité*** **(temporelle) consiste à décomposer le décodage des instructions**
  - Ces **structures** qui apportent une **simultanéité, anticipation et recouvrement**, ont été nommées chaîne d'assemblage, on dit aujourd'hui **pipeline**.

« Le pipeline augmente les performances du seul processeur »

- ***Le parallélisme*** **(spatial) consiste à dupliquer les supports des données, pour y accéder dans un même temps ou pour disposer de mémoires aux caractéristiques meilleures.**

« Ce sont les caches »

→ **L'analogie usuelle est celle de la logistique à entrepôts** :

- un entrepôt central contient le stock, des entrepôts régionaux contiennent des stocks intermédiaires en plus petites quantités, enfin le stock “arrière” de chaque magasin contient un approvisionnement minime.

Un cache comble en partie les différences de performances temporelles entre deux voisins

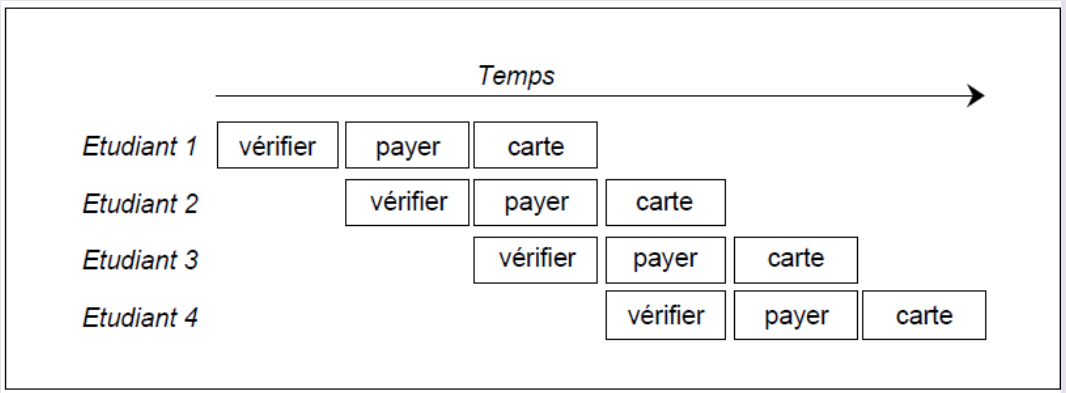
LES PIPELINES

- **<u>Le principe du pipeline est celui de la chaine de montage</u>**
  - fractionner la tâche en sous tâches de durées égales appelées étages
  - exécuter simultanément les déférentes sous tâches de plusieurs tâches

- Exemple : lors de son inscription un étudiant doit exécuter trois actions
  1. faire vérifier des diplômes
  2. puis payer
  3. enfin recevoir sa carte

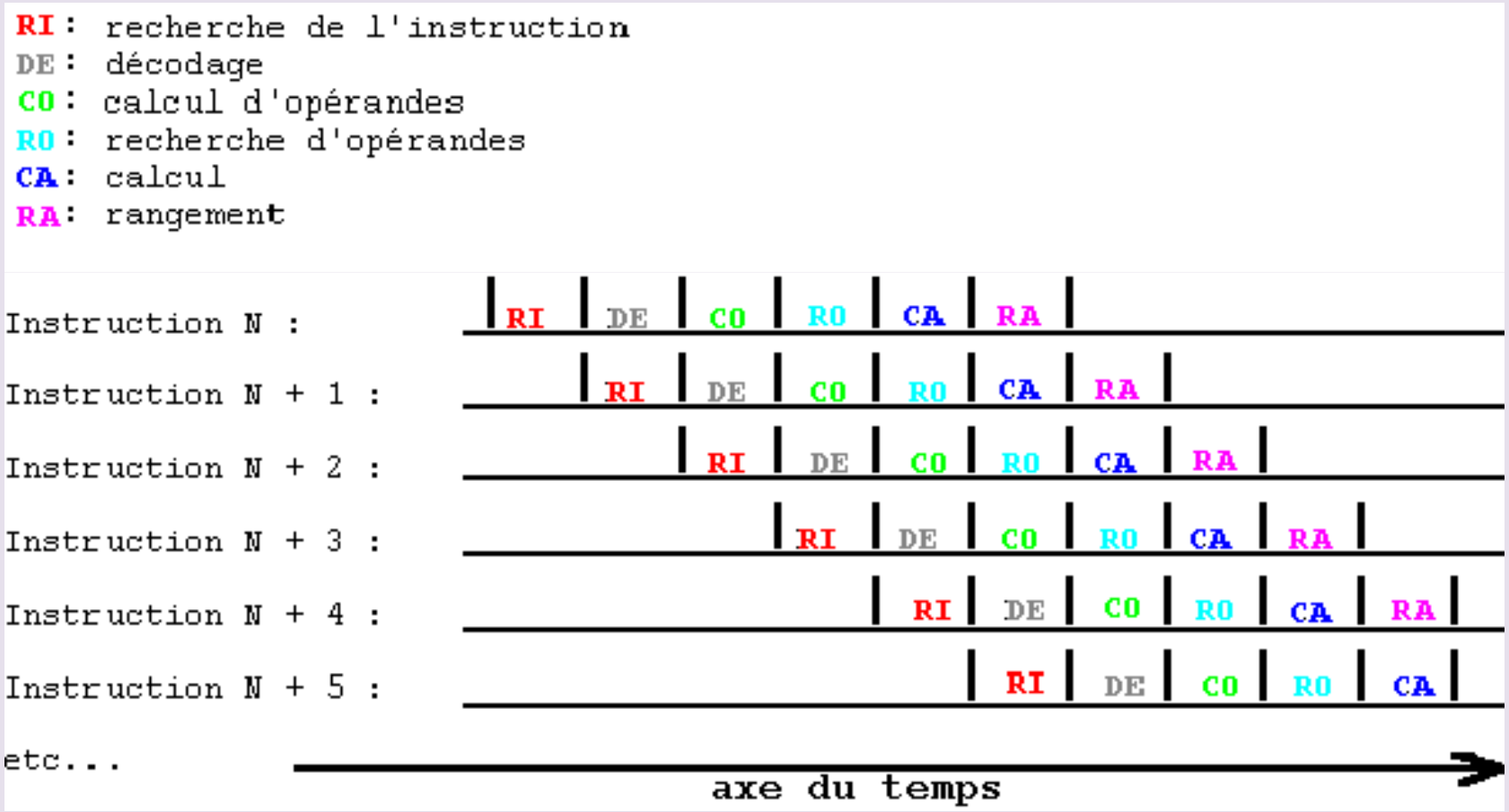
  - Ces trois actions dépendent l'une de l'autre et doivent donc être exécutées en séquence
  - En admettant que chaque action consomme une unité de temps le temps total pour chaque étudiant **est trois unités** et ne peut pas être améliore
  - Si chaque étudiant est traité complètement par un seul employé d'administration le débit est **1/3 d'étudiant par unité de temps**

- Avec des ressources plus importantes et si beaucoup d'étudiants doivent s'inscrire le débit du système peut être triplé :
  - pendant que le premier étudiant effectue le paiement un deuxième étudiant entre dans le système et voit ses diplômes vérifiés.
- Au temps suivant trois étudiants occupent chacun un poste de traitement
- Au temps suivant le premier étudiant est sorti du système à partir de là un étudiant sort du système à chaque unité de temps
  - le débit est donc d'un étudiant par unité de temps bien que le temps individuel vécu par chaque étudiant reste trois unités de temps



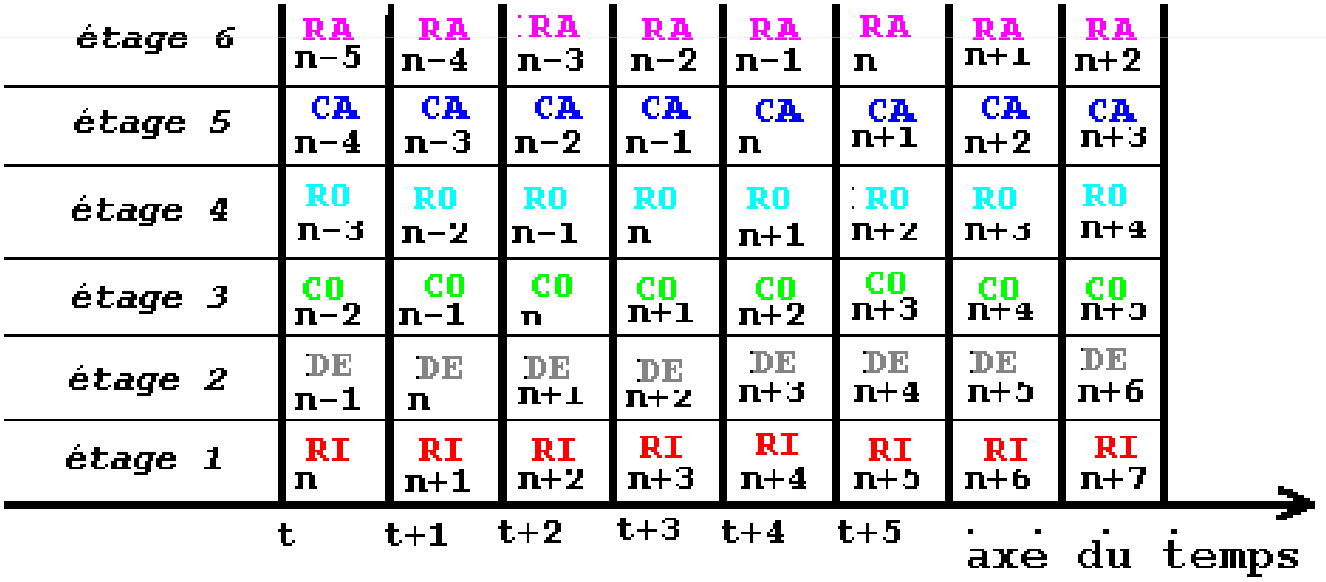
- **Anticiper est commencer un travail alors que le précédent n'est pas terminé.**
- **Recouvrir s'entend ici par poursuivre un traitement alors que le suivant est entamé.**
- **L'anticipation et le recouvrement sont issus d'un constat :**
  - le chargement d'une instruction et le début au moins de son décodage ne dépendent pas, sauf branchement, **du résultat de l'exécution de l'instruction précédente**.

- **Exemples : Les deux figures suivantes présentent deux vues du fonctionnement d'un pipeline de principe à six étages.**
- **Exemple 1 : en suivant les instructions dans le temps ⇔ vue diachronique.**



♦ **Exemple 2 : En suivant les étages dans le temps ⇔ vue synchronique :**

RI : recherche de l'instruction
DE : décodage
CO : calcul d'opérandes
RO : recherche d'opérandes
CA : calcul
RA : rangement



- L'exécution d'une instruction : Les étapes fondamentales :

Instructions UAL	*Instructions Mémoire*	*Instructions Branchement*
Lecture instruction	Lecture instruction	Lecture instruction
Incrémentation CP	Incrémentation CP	Incrémentation CP
Décodage de l'instruction	Décodage de l'instruction	Décodage de l'instruction
Lecture des opérandes	Calcul de l'adresse	Calcul de l'adresse de
Exécution	mémoire	branchement
Ecriture du résultat	Accès mémoire	Exécution
	Rangement du résultat	

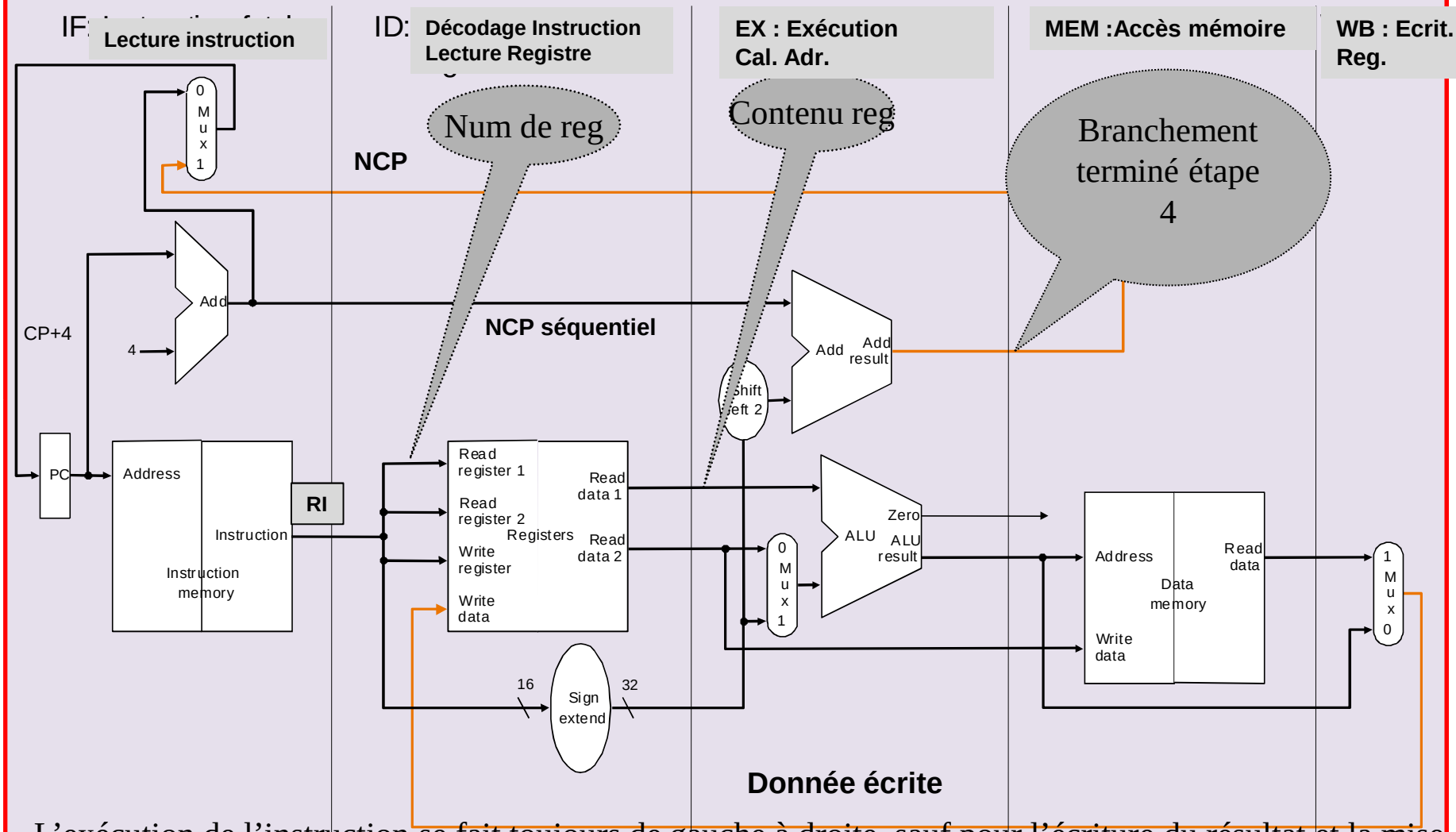
- **Instructions entières**
  - LI/CP DI/LR EX ER
- **Instructions flottantes**
  - LI/CP DI/LR EX1 EX2 ... ER
- **Instructions mémoire**
  - LI/CP DI/LR CA AM ER
- **Instructions de branchement**
  - LI/CP DI/CAB/EX

- ***Rappel :***

L'exécution de l'instruction se fait toujours en 5 étapes

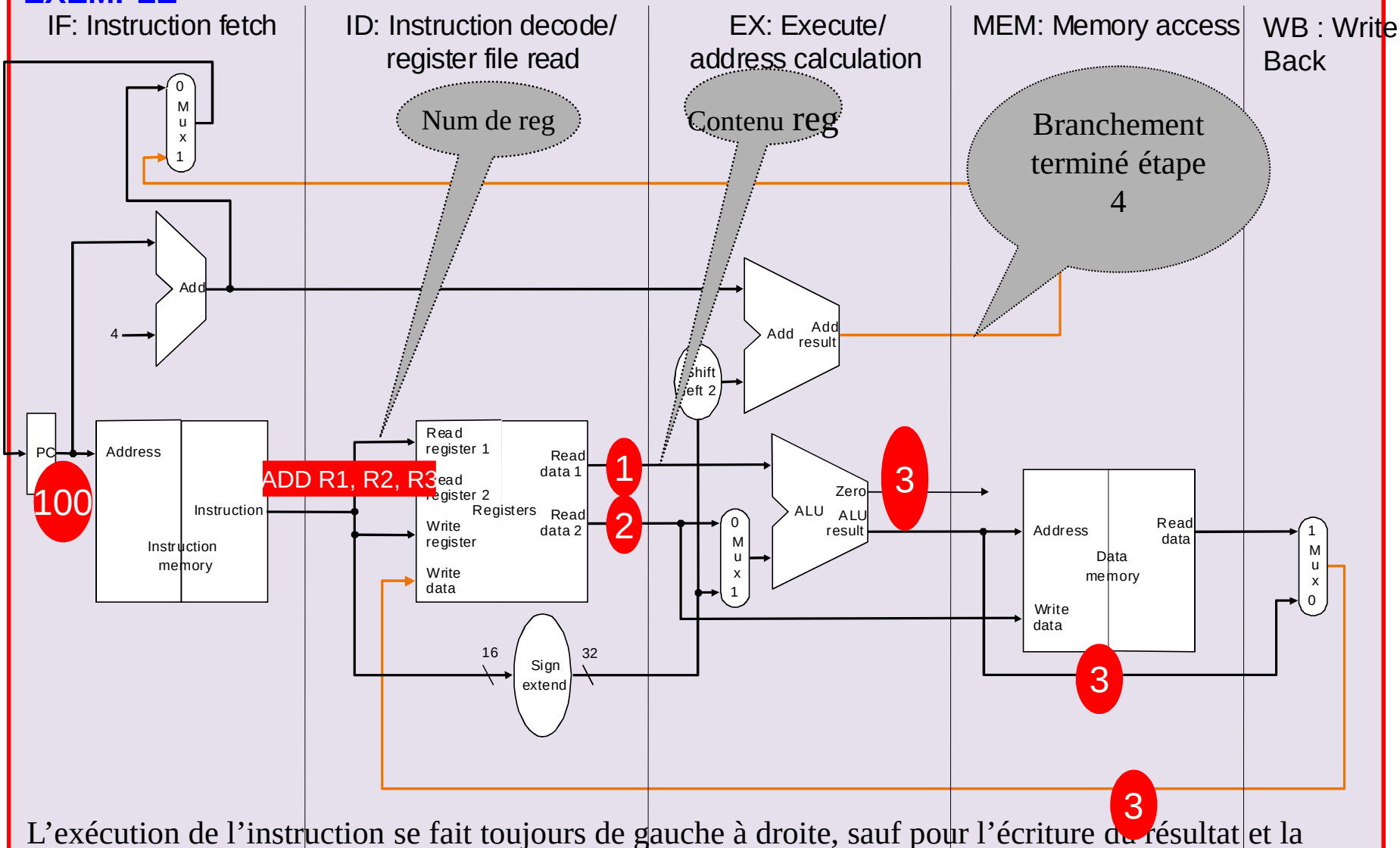
1. Lecture de l'instruction
2. Décodage et en même temps lecture des registres
3. Calcul dans l'**UAL**
   - Instruction UAL : appliquer l'opération
   - Instruction Mémoire : Calculer l'adresse
   - Instruction de branchement : Calculer l'adresse de branchement et évaluer la condition
4. Accès mémoire
5. Mise à jour du registre destination (pour les **UAL** et **loads**) ou **CP** (pour les branchements)

Chemin de données de la machine MIPS sans Pipeline



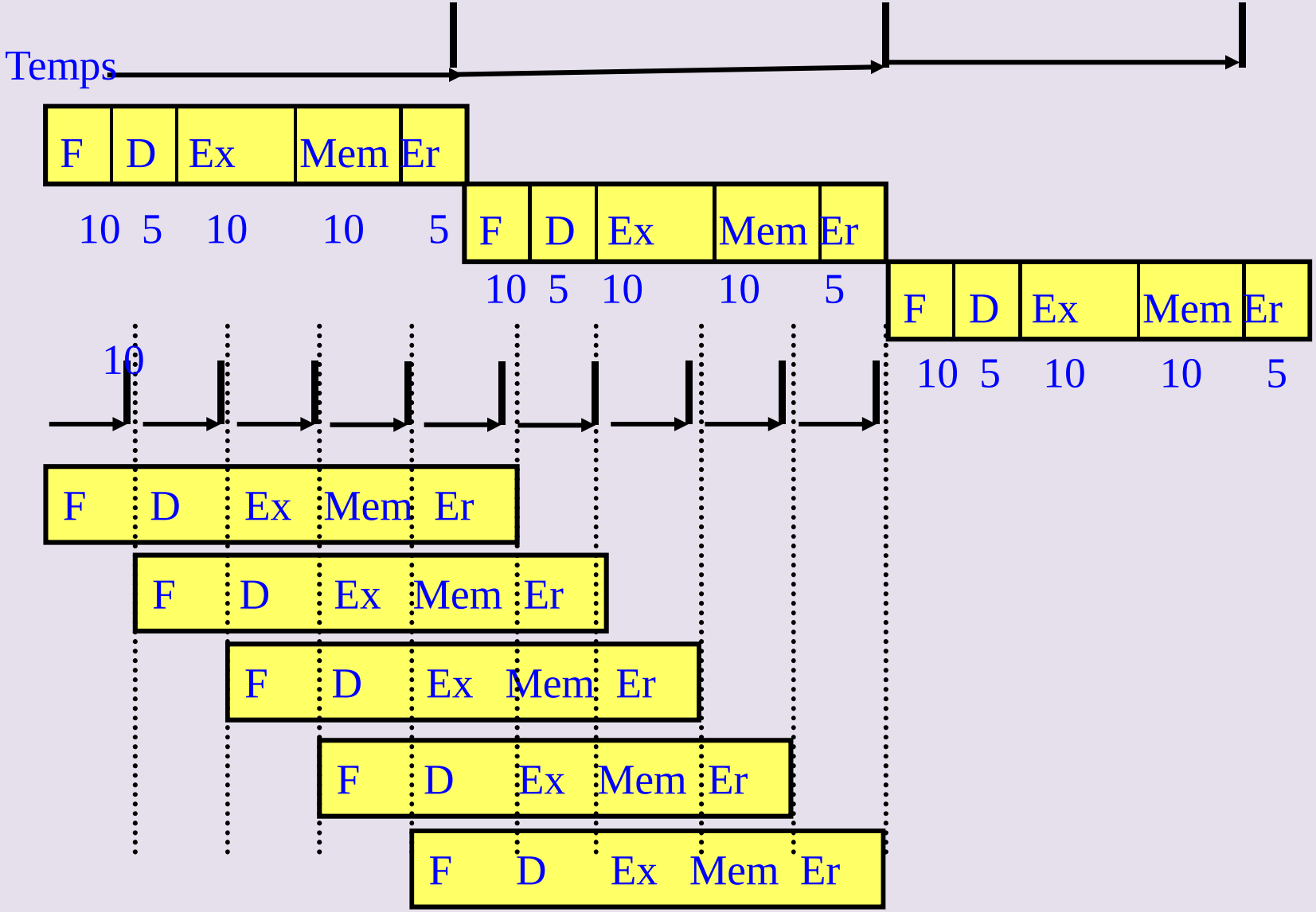
L'exécution de l'instruction se fait toujours de gauche à droite, sauf pour l'écriture du résultat et la mise à jour de CP en cas de branchement pris.

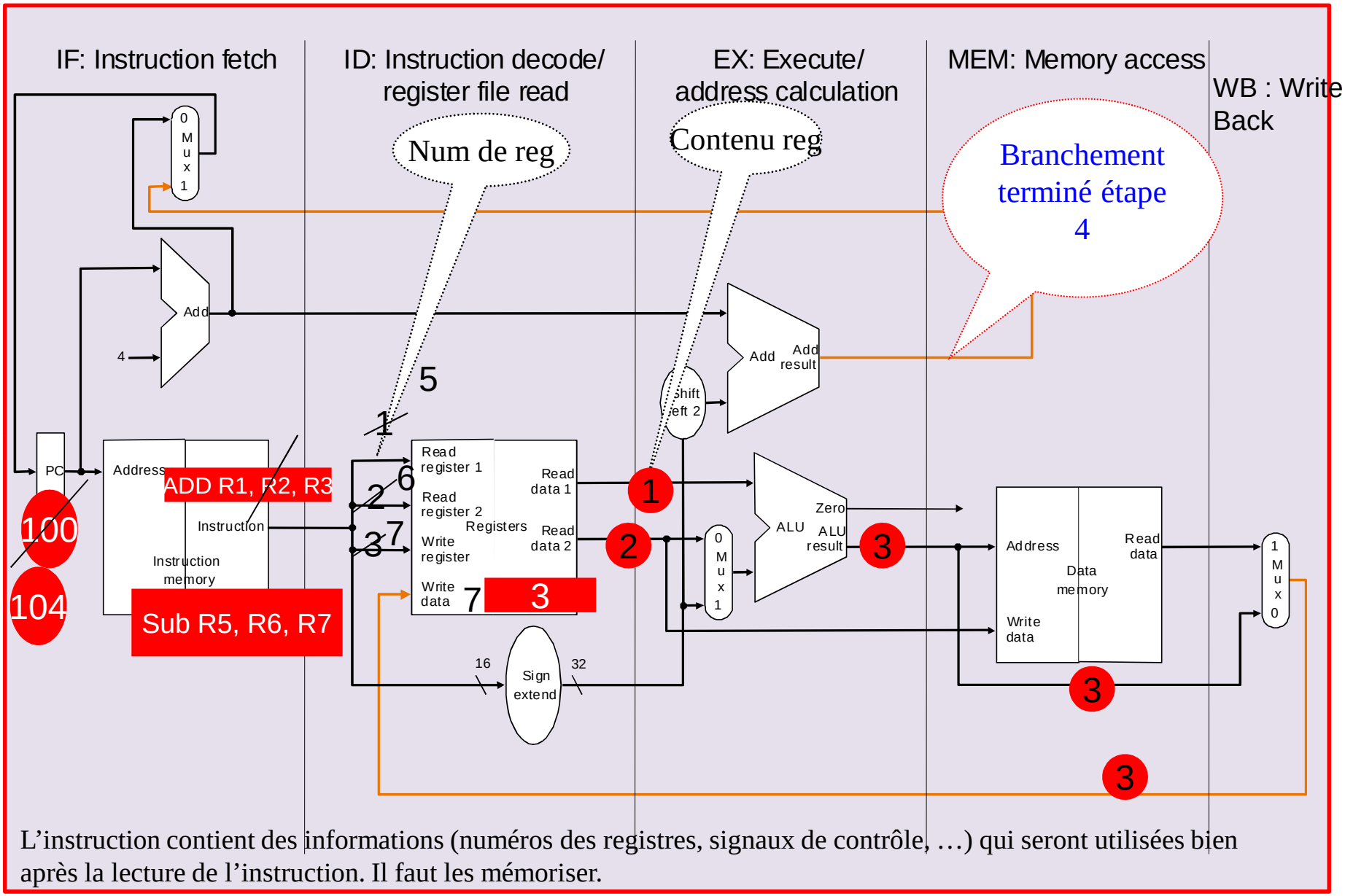
EXEMPLE



L'exécution de l'instruction se fait toujours de gauche à droite, sauf pour l'écriture du résultat et la mise à jour de CP en cas de branchement pris.

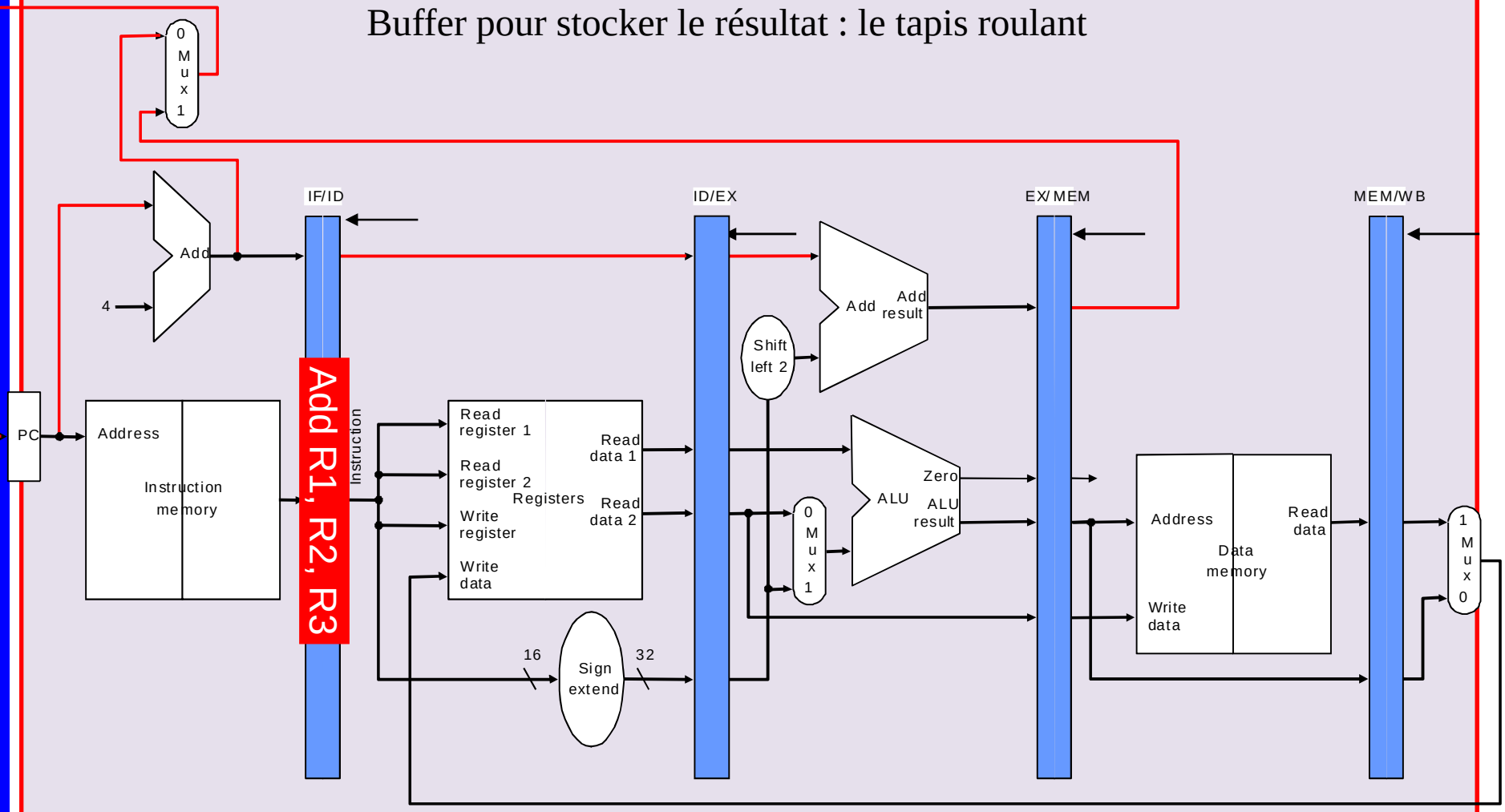
Exécution des instructions sans et avec pipeline





L'instruction contient des informations (numéros des registres, signaux de contrôle, …) qui seront utilisées bien après la lecture de l'instruction. Il faut les mémoriser.

Buffer pour stocker le résultat : le tapis roulant

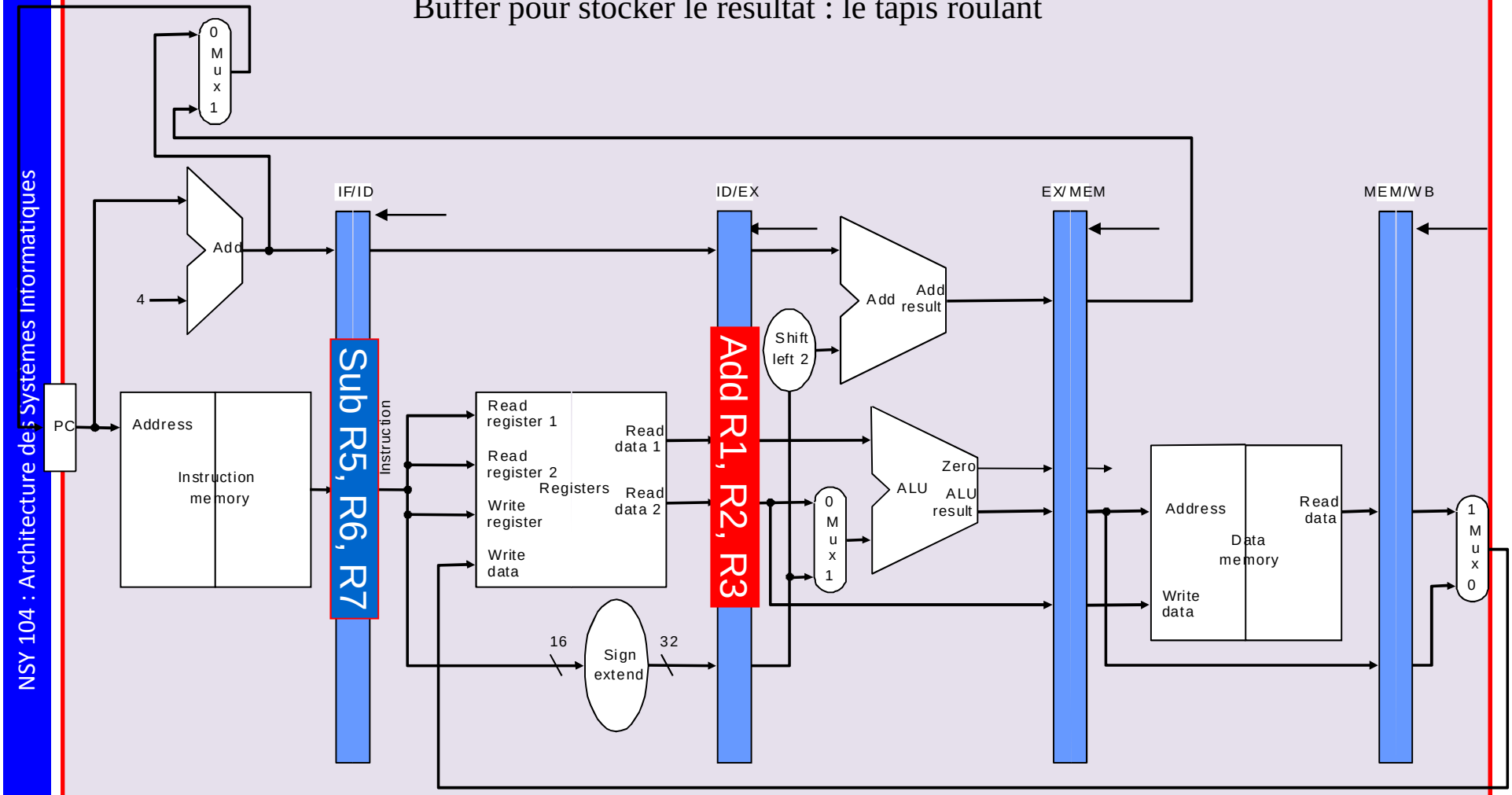


Ces « buffers » sont utilisés non seulement pour stocker les données mais aussi les signaux de contrôle.

L'écriture dans les buffers est faite ***<u>à chaque fin de cycle par les signaux :</u>***

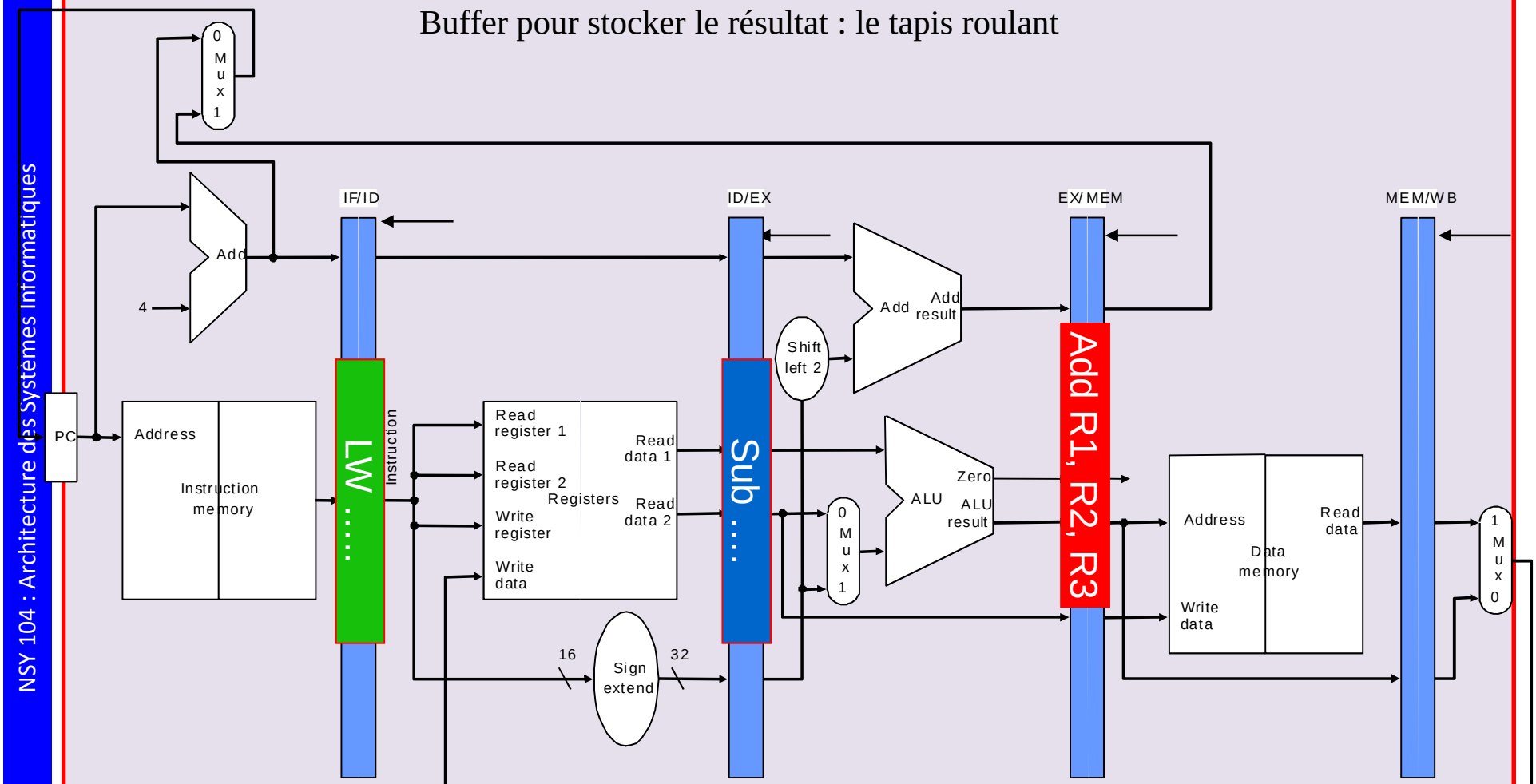
FI/IDWrite, ID/ExWrite, EX/MemWrite, Mem/RbWrite

Buffer pour stocker le résultat : le tapis roulant



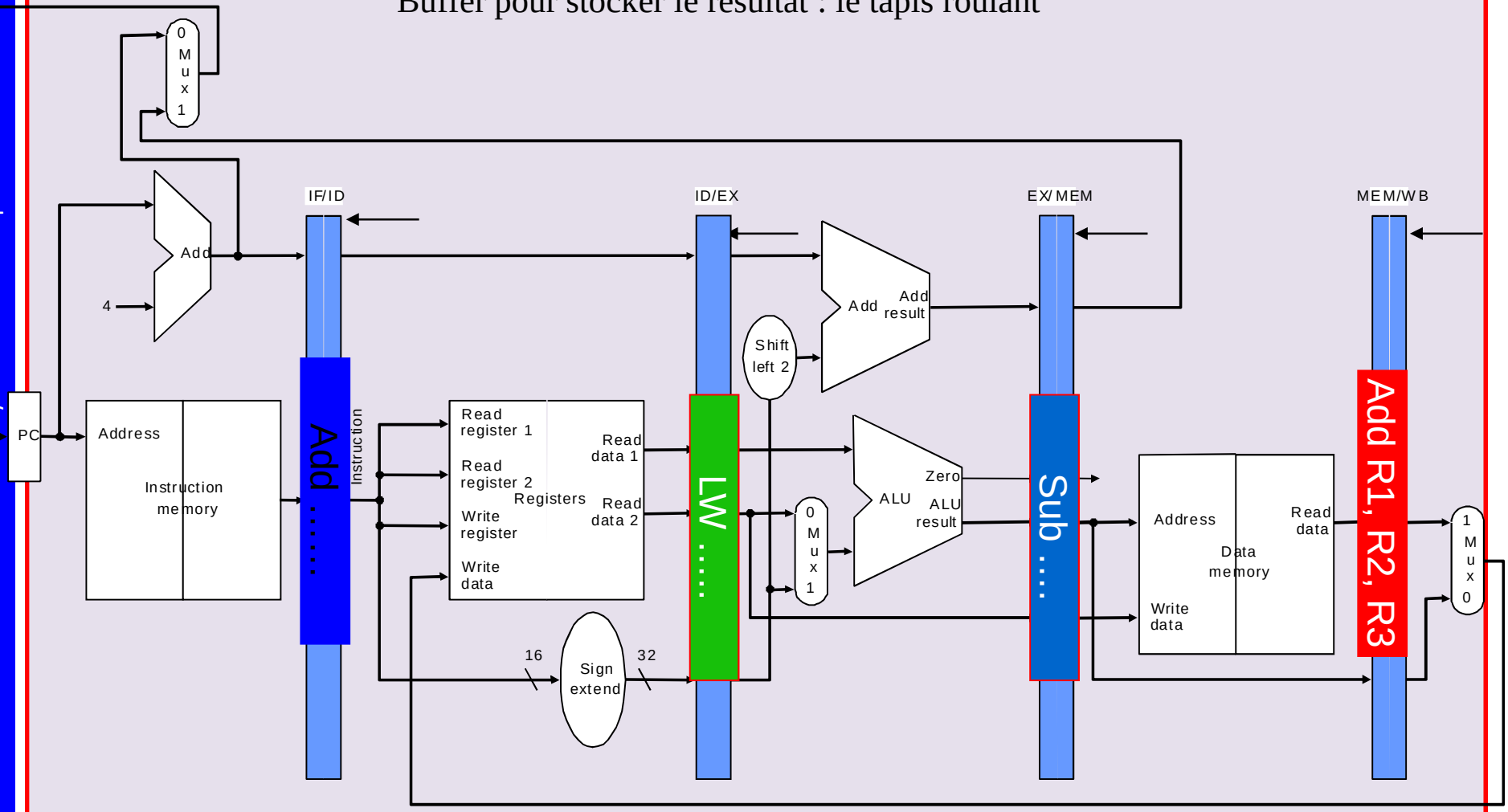
Ces « buffers » sont utilisés non seulement pour stocker les données mais aussi les signaux de contrôle.
L'écriture dans les buffers est faite à chaque fin de cycle par les signaux :
FI/IDWrite, ID/ExWrite, EX/MemWrite, Mem/RbWrite

Buffer pour stocker le résultat : le tapis roulant



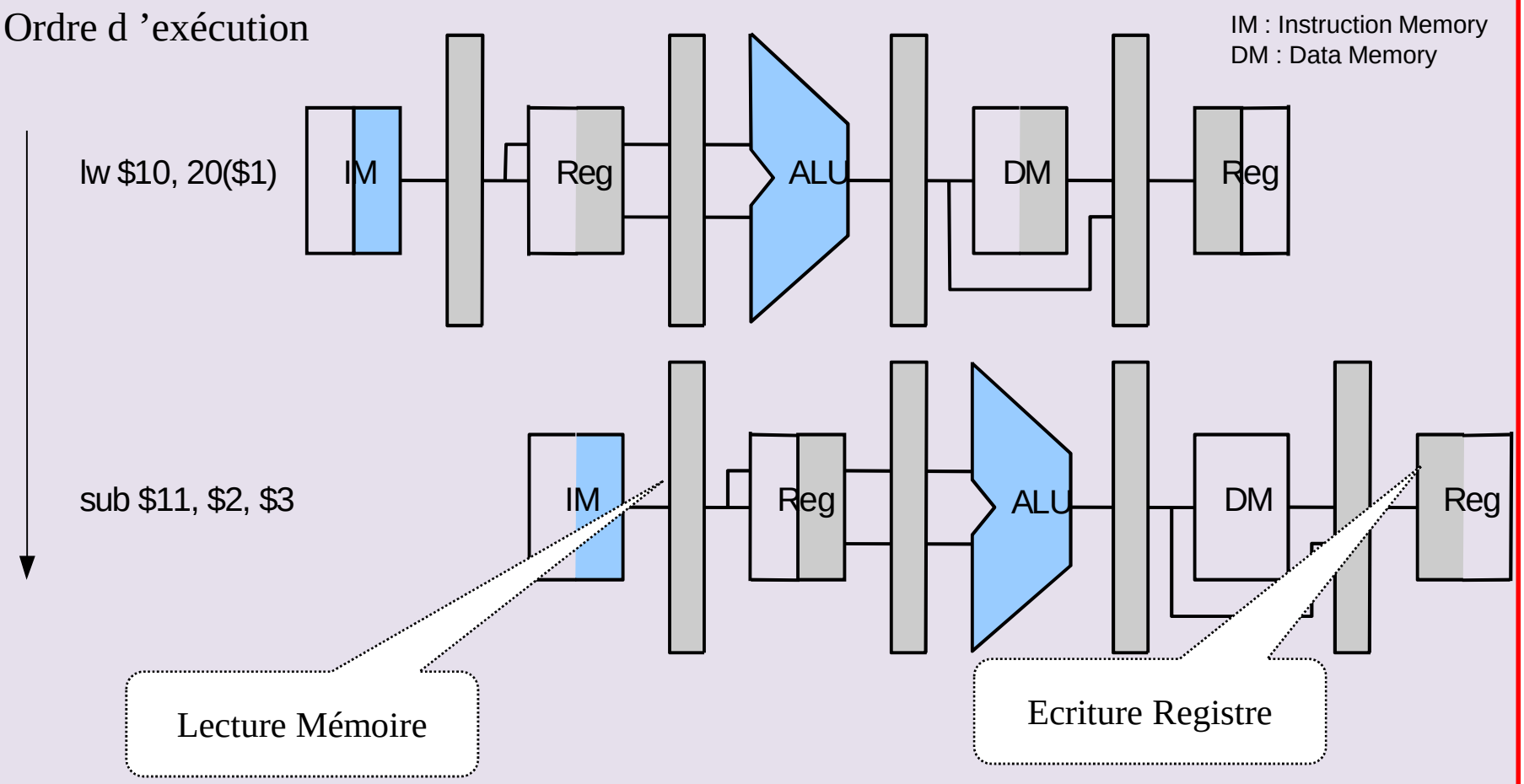
Ces « buffers » sont utilisés non seulement pour stocker les données mais aussi les signaux de contrôle.
L'écriture dans les buffers est faite à chaque fin de cycle par les signaux :
FI/IDWrite, ID/ExWrite, EX/MemWrite, Mem/RbWrite

Buffer pour stocker le résultat : le tapis roulant



Ces « buffers » sont utilisés non seulement pour stocker les données mais aussi les signaux de contrôle.
L'écriture dans les buffers est faite à chaque fin de cycle par les signaux :
FI/IDWrite, ID/ExWrite, EX/MemWrite, Mem/RbWrite

lw (load word) : cherche un mot en mémoire [instruction relatif] : implique une lecture mémoire
Lw $10, 20($1) : lire la donné se trouvant dans l'adresse : @[$1 + 20] et la mettre dans l'adresse @10
Sw : (store word) : pour un stockage, implique une écriture mémoire



Voir : http://people.ee.duke.edu/~sorin/prior-courses/ece152-spring2009/lectures/

- Un pipeline ne serait pas efficace **s'il était géré par programme**, il doit être géré par le **matériel,** ainsi :
  - Le programmeur ne peut ni ne doit voir le pipeline.
    - ☞ Le pipeline ne fait jamais partie de l'architecture induite par le jeu d'instructions
  - La justification du pipeline est propre au processeur lui même.
    - ☞ À technique constante, un pipeline produit une augmentation des performances **du seul processeur**, qui ne serait pas atteignable par d'autres moyens.

- Autres utilisations : Cette technique est applicable à toute tâche longue, ***aux deux conditions :***
  1. Qu'elle soit décomposable en tâches plus petites;
  2. Qu'elle soit répétitive pour conserver le pipeline plein.
     - En particulier, elle est applicable à l'enchaînement des opérations en virgule flottante qui doivent traiter successivement
       - les exposants,
       - l'alignement des mantisses,
       - les calculs
       - et la normalisation des résultats.

- Chaque étage fonctionne indépendamment des autres sous les réserves ultérieures.
- Un signal d'horloge unique est utilisé pour la commande des transferts entre étages.
  - Sa période est égale au ***délai propre à l'étage le plus lent***.
  - ☞ On observe couramment des réalisations de 5 à 12 étages, le Pentium IV fait exception avec 20 étages.

1. Dans un *étage* est réalisée une *étape* du traitement.
2. Le *nombre d'étages* est aussi nommé ***profondeur*** du pipeline. Un pipeline de *profondeur k*, contient au plus k instructions successives.
3. **Dans le cas idéal,** chaque étage opère sur une instruction différente à tout instant.

❑ **Nous présentons les calculs sous plusieurs hypothèses.**

- Soit **t** : le temps total de décodage sans pipeline.
- Soit t_1 : le temps de traitement de l'étape la plus longue.
- Soit **k** : le nombre d'étages.

- Le **temps de transit** d'une instruction dans le **décodeur sans pipeline** est : **t**
- Le **temps de transit** d'une instruction dans **le pipeline est** : kxt_1

- Supposons le pipeline **vide au début**, il sera rempli pendant le **temps de transit de la première instruction soit** : : kxt_1.
- Dès que le pipeline est plein, un résultat est obtenu toutes les : t_1 **secondes**
- Si le nombre d'instructions à exécuter est **n**, le temps total d'exécution est :

$$k x t_1 + (n-1) x t_1 = (n+k-1) x t_1$$

- **L'accélération** est alors $S = n x t / (k+n-1) x t_1$
- Si **n** est très grand **S** devient proche de t/t_1

♦ **Application** :

- Supposons les étapes équi-réparties sans perte : $t_1 = t/k$,
- l'accélération est alors : $S = n \times k / (k + n - 1)$.

Elle tend vers **k** quand **n** augmente indéfiniment.

- **Conclusion : L'accélération apportée par un pipeline idéal équi-réparti tend vers le nombre de ses étages.**

♦ Exemple d'un pipeline pour un additionneur simplifié en virgule flottante.

♦ Soit un flot indéfini d'additions. Une fois le pipeline rempli :

- L'addition demande 10 cycles : un étage
- L'alignement 5 cycles : un étage
- La normalisation 25 cycles : un étage

➢ Sans pipeline chaque exécution nécessite **10+5+25 = 40 cycles;**

➢ Avec un pipeline **à quatre étages** où l'opération la plus longue est faite en **25 cycles;**

➢ Chaque addition est faite en : **25+25+25 = 75 cycles;**

➢ Une addition est achevée tous les : **25 cycles;**

➢ **Le facteur d'augmentation du débit** est : **40/25 = 1,6**.

▪ Si l'on décompose l'étage de normalisation en trois étages chacun de **10 cycles**, en tout **30 cycles** au lieu de **25**, l'étape la plus longue dure 10 cycles,

➢ **le facteur d'augmentation du débit** est **40/10 = 4.**

Instruction N	**AD**	**AL**	**NOR**			
Instruction N+1		**AD**	**AL**	**NOR**		
Instruction N+2			**AD**	**AL**	**NOR**	
Instruction N+3				**AD**	**AL**	**NOR**

Les conflits dans un pipeline et leur résolution -1-

- **Rappels :**
  - Les conflits tiennent **aux instructions** et **non au pipeline**.
  - Ce sont des conflits **dans** le pipeline et non des conflits **du** pipeline.
  - Les mêmes sources de conflits existent dans les processeurs **VLIW** pour les exécutions simultanées des instructions

- **Les instructions utilisent des ressources de la machine pour leur exécution.**
  - **Elles manipulent des données.**
  - **Les branchements structurent la commande du programme.**

- **Les trois causes de conflits tiennent alors :**

1. ***Aux ressources de la machine autres que le pipeline.***
   - **Cas suivant : Deux instructions présentes dans le pipeline qui doivent utiliser la même ressource (matérielle ou logicielle).**
   - **Exemples :** le bus pour deux transferts, un registre pour deux écritures, une UAL unique qui doit faire une addition et calculer une adresse.

2. ***Aux données manipulées,*** par leur **dépendance** et les différences de **durées d'exécution des instructions.**
   - **Exemple classique :** un opérande de I_{n+1} est le résultat de I_n, mais I_n n'est pas terminée alors que I_{n+1} commence. .

3. ***À la commande programmée***. Les branchements rompent l'ordre de la séquence.

   - Le branchement inconditionnel sera traité **très tôt** par son code d'opération.
   - Le branchement conditionnel ne peut être décidé **qu'après exécution de l'instruction.**

     Ceci amène :

     - ☞ **une *obligation*** : pouvoir annuler les travaux en cours qui doivent donc être réversibles, déchargement du pipeline
     - ☞ **et une *nécessité :*** prévoir au mieux la branche qui sera prise.

Phénomènes :

- Ce conflit apparaît quand **deux opérations** devraient utiliser **la même ressource** au **même instant**.
- Pour prévoir cet état, il faut connaître **l'état d'occupation des ressources** à chaque instant et faire une **détection préventive**.
- Comment ?
  - On entretient pour cela une **table des réservations** qui contient **l'état d'occupation** des ressources pour chaque cycle d'horloge.
  - Chaque **colonne** correspond à **un étage du pipeline**.
  - Chaque ressource **dispose d'une ligne**. La case **(i,j)** est à 1 si la ressource **i** est utilisée à l'instant **j**, à zéro sinon.

Solutions :

- Duplication des ressources en cause;
- Nouvel ordonnancement des instructions;
- Attente de libération de la ressource.

	Etage 1	Etage 2	.....	Etage n
R 1	1	0		
R2	0			
R3		1		
:				
:				
:				
:				
Rn				

- **Exemple :**
  - Dans un pipeline quelconque qui ***a un accès unique à la mémoire***, toute lecture d'opérande, toute écriture de résultat sera **en conflit** avec la lecture d'une instruction suivante.

Phénomènes :

- La dépendance des données apparaît dans les pipelines. Elle est d'autant plus critique que des instructions peuvent **être exécutées dans le désordre**.
- La dépendance provient des deux opérations de lecture et d'écriture, il y en a donc 4, nommées en anglo-saxon :

1.	**RAR**	**L**ecture après lecture,	**« read after read »;**
2.	**WAR**	écriture après lecture,	**« write after read »;**
3.	**RAW**	Lecture après écriture,	**« read after write »;**
4.	**WAW**	écriture après écriture,	**« write after write ».**

1. ***La dépendance RAR :***
   - **L**ectures successives de la même donnée, n'apporte pas de conflit.

2. ***Dans la dépendance WAR :***
   - Une instruction de rang **I+n** écrit **dans un registre lu** par une instruction de rang **I**.
   - Si ces instructions sont exécutées dans le désordre, il ne faut pas que la **I+n** se termine avant que **I** n'ait lu le registre.
   - ***Cette dépendance n'apparaît qu'en cas d'exécution dans le désordre.***

3. ***Dans la dépendance RAW,***
   - une instruction de rang **I+n** **lit un registre écrit par** une instruction de rang **I.**
   - Il faut attendre que l'instruction **I** écrive son résultat avant de lire l'opérande de l'instruction **I+n**.
   - **Cette dépendance existe dans un exécution séquentielle**

4. *Dans la dépendance* WAW :

- une instruction de rang **I+n** écrit dans **un registre lui-même écrit par** l'instruction de rang **I**.
- Si ces instructions **sont exécutées dans le désordre**, il ne faut pas que la **I+n** se termine avant que la **I** ne se termine.

- **Cette fausse dépendance n'apparaît qu'en cas d'exécution dans le désordre.**

Exemples sur un petit programme :

- int a := 3;
- int b := 2;
- int c := 1;
- a := b + c;
- b := a + b;
- b := a + c;

Peut être compilé en :

```
LDI r31,3                    ; initialise a, chargement immédiat.
LDI r30,2                    ; initialise b, idem.
LDI r29,1                    ; initialise c, idem.
i1 : ADD r31,r30,r29  i1     ; lit r30 et r29, écrit dans r31
i2 : ADD r30,r31,r30  i2     ; lit r31 et r30, écrit dans r30
i3 : ADD r30,r31,r29  i3     ; lit r31 et r29, écrit dans r30
```

Exemples sur un petit programme :

- int a := 3;
- int b := 2;
- int c := 1;
- a := b + c; (i1)
- b := a + b; (i2)
- b := a + c; (i3°

Peut être compilé en :

```
LDI r31,3                    ; initialise a, chargement immédiat.
LDI r30,2                    ; initialise b, idem.
LDI r29,1                    ; initialise c, idem.
i1 : ADD r31,r30,r29   i1    ; lit r30 et r29, écrit dans r31
i2 : ADD r30,r31,r30   i2    ; lit r31 et r30, écrit dans r30
i3 : ADD r30,r31,r29   i3    ; lit r31 et r29, écrit dans r30
```

- i1 dépend de i2 en **WAR** car i2 écrit dans r30, lu par i1.
- i2 dépend de i3 en **WAW** car i2 écrit dans r30, écrit par i3
- i2 dépend de i1 en **RAW,** car i2 lit r31, écrit par i1.

➢ **En exécution séquentielle,** la dépendance RAW apparaît seule.

➢ **En exécution dans le désordre**, les deux autres peuvent survenir.

♦ **Solutions,** toujours préventives :

1. **Le pipeline étant alimenté en séquence,**

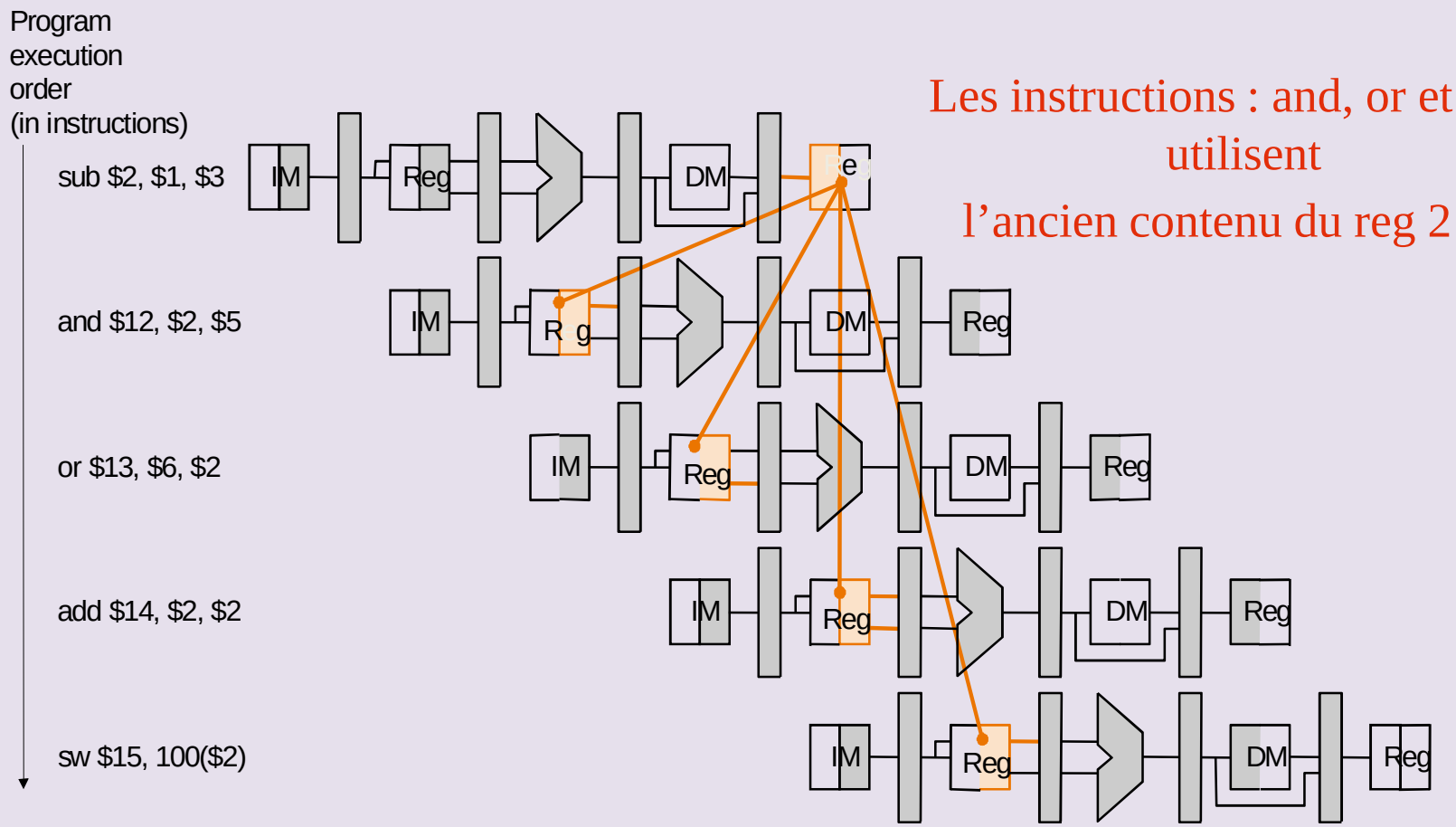
   - on **détecte les situations** précédentes par **examen de son contenu.**
   - Dans tous les cas, si un couple d'instructions est en cause, la seconde sera bloquée dans le pipeline jusqu'à la fin de la première.

     ➢ On dit que **le pipeline contient *une bulle***.

2. **Le travail de prévention peut être confié au compilateur ou à un post processeur de compilation, avec tous les inconvénients associés.**

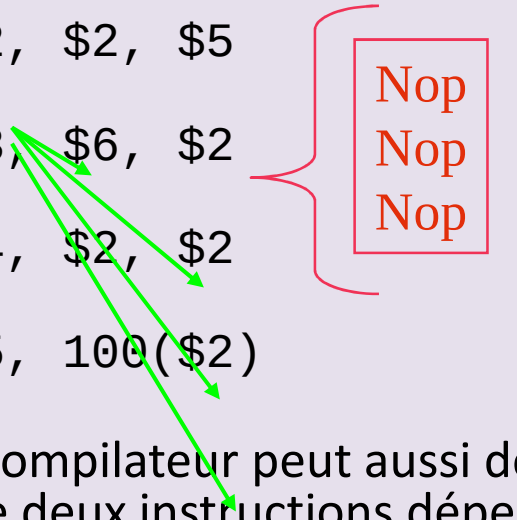
   - La **détection est alors statique** et,
     - ***soit l'ordre des instructions est modifié,***
     - soit ***une instruction de non opération*** est insérée qui provoque un effet de *bulle* dans le pipeline.
   - Le traitement des instructions qui suivent un branchement conditionnel est particulièrement pénalisante. On dit que l'on a fait de l'ordonnancement de pipeline.
   - Pour les accès à la mémoire, on utilise des retards au chargement (stalls).
     - L'utilisation des caches compliquent ces situations.

Les aléas de données-Exemple



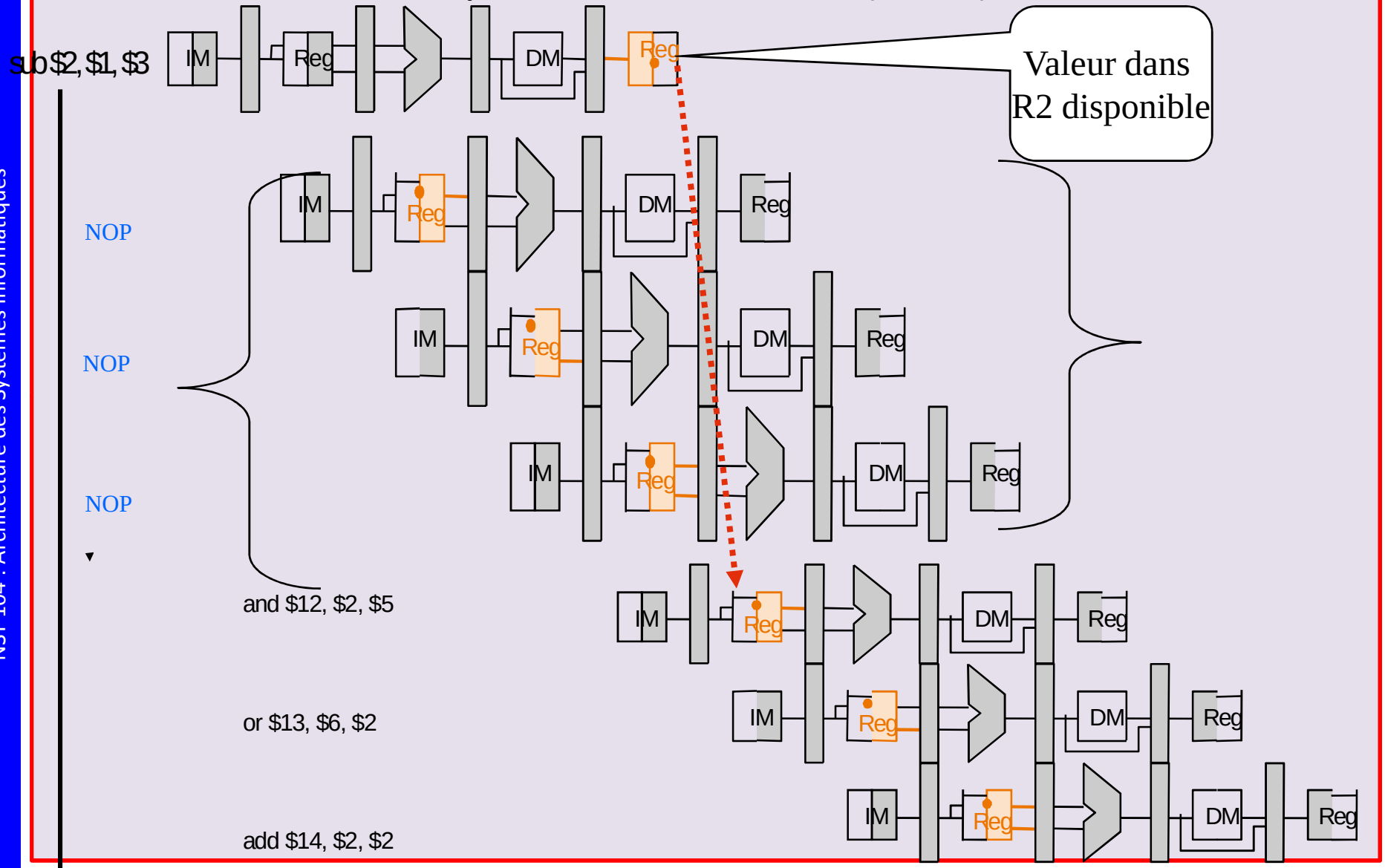
- Compter sur le compilateur pour détecter les DD, et insérer les instructions NOP. Inconv. : perte de temps
  - Dans notre cas à chaque dépendance entre deux instructions :
    Il faut insérer 1, 2 ou 3 Nop suivant la distance entre les instructions.

```
sub    $2, $1, $3
and    $12, $2, $5
 or    $13, $6, $2
add    $14, $2, $2
sw     $15, 100($2)
```



  - Remarque le compilateur peut aussi déplacer du code pour l'insérer entre deux instructions dépendantes.

Résoudre les aléas de données par des instructions sans effet (les NOP)

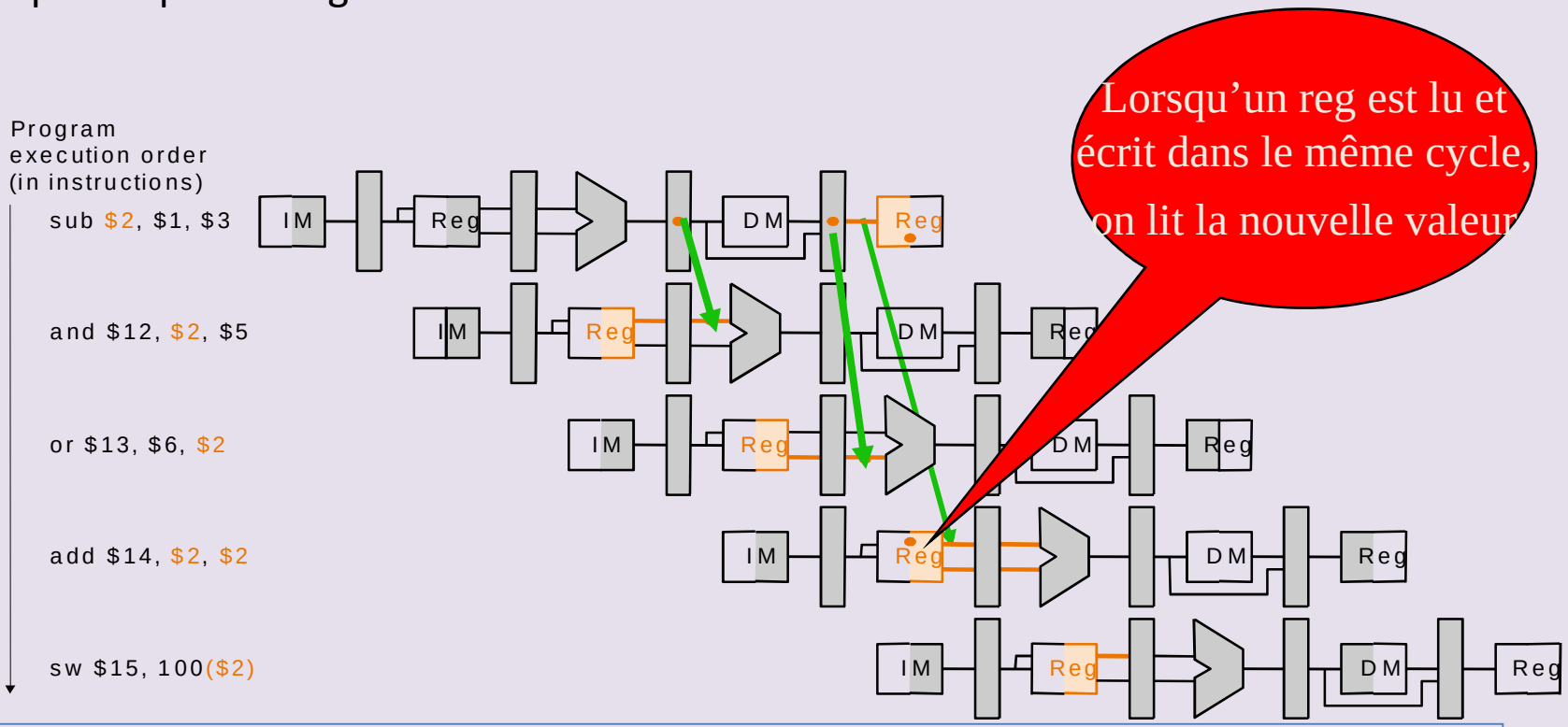


3. Le **court-circuit matériel**

- consiste à prendre le résultat d'une instruction et à le fournir en entrée de l'instruction suivante **sans dépôt intermédiaire dans un registre.**
- Ce n'est pas toujours efficace,
- Elle appelée : **technique de « forwarding ».**

3. Le **court-circuit matériel – exemple**

- Le résultat est **envoyé directement** vers l'instruction destination **avant qu'il ne soit écrit dans le registre destination**: Il y a transmission directe entre instructions sans passer par les regs



lw (load word) : cherche un mot en mémoire [instruction relatif] : implique une lecture mémoire
Lw $10, 20($1) : lire la donné se trouvant dans l'adresse : @[$1 + 20] et la mettre dans l'adresse @10
Sw : (store word) : pour un stockage, implique une écriture mémoire

4. Le **renommage des registres** utilise des **registres non visibles**, **non accessibles** au programmeur. On y range des **résultats intermédiaires.**

- **Exemple :** Dans l'exemple ci-dessous, si on peut placer le résultat de **i2**, prévu pour être dans **r30**, dans un autre registre

  ☞ **les deux instructions peuvent être exécutées *simultanément*.**

➢ **Exemples sur un petit programme :**

Peut être compilé en :

```
LDI r31,3                ; initialise a, chargement immédiat.
LDI r30,2                ; initialise b, idem.
LDI r29,1                ; initialise c, idem.
i1 : ADD r31,r30,r29  i1 ; lit r30 et r29, écrit dans r31
i2 : ADD r30,r31,r30  i2 ; lit r31 et r30, écrit dans r30
i3 : ADD r30,r31,r29  i3 ; lit r31 et r29, écrit dans r30
```

- **En exécution séquentielle :**
  - i1 dépend de i2 en **WAR** car i2 écrit dans r30, lu par i1.
  - i2 dépend de i3 en **WAW** car i2 écrit dans r30, écrit par i3
  - i2 dépend de i1 en **RAW,** car i2 lit r31, écrit par i1.

➢ **En exécution séquentielle,** la dépendance RAW apparaît seule.

➢ **En exécution dans le désordre**, les deux autres peuvent survenir.

- Le **renommage** de registre peut être **systématique** :
  - Les instructions sont ***décodées dans l'ordre*** et **à chaque fois qu'une instruction part à l'exécution**, on lui attribue un ***registre de renommage*** pour ranger son résultat.
  - A la fin de l'exécution, les registres de renommage sont recopiés d**ans l'ordre du programme** dans les ***registres officiels*** pour conserver la sémantique du programme.

- **Phénomènes :**
  - Les instructions de ***branchement inconditionnels*** (les sauts) et les ***branchements conditionnels*** rompent la **séquentialité** d'exécution.
  - Du simple au complexe, correspondant **à une résolution** de plus en plus éloignée de l'entrée du pipeline, nous avons :
    A. le saut à **adresse explicite**;
    B. le saut à **adresse calculée**, via un registre d'index par exemple;
    C. le branchement conditionnel **à adresse fixe**;
    D. le branchement conditionnel **à adresse calculée**.

A. Les sauts

– Leur traitement est relativement simple car même **si l'adresse est calculée**, ***elle l'est dans les tout premiers étages***.

B. Les branchements conditionnels

– Ils sont des : ***exécutions conditionnelles*** et des ***tests de boucles***.

– Un branchement est **spécifié** par :

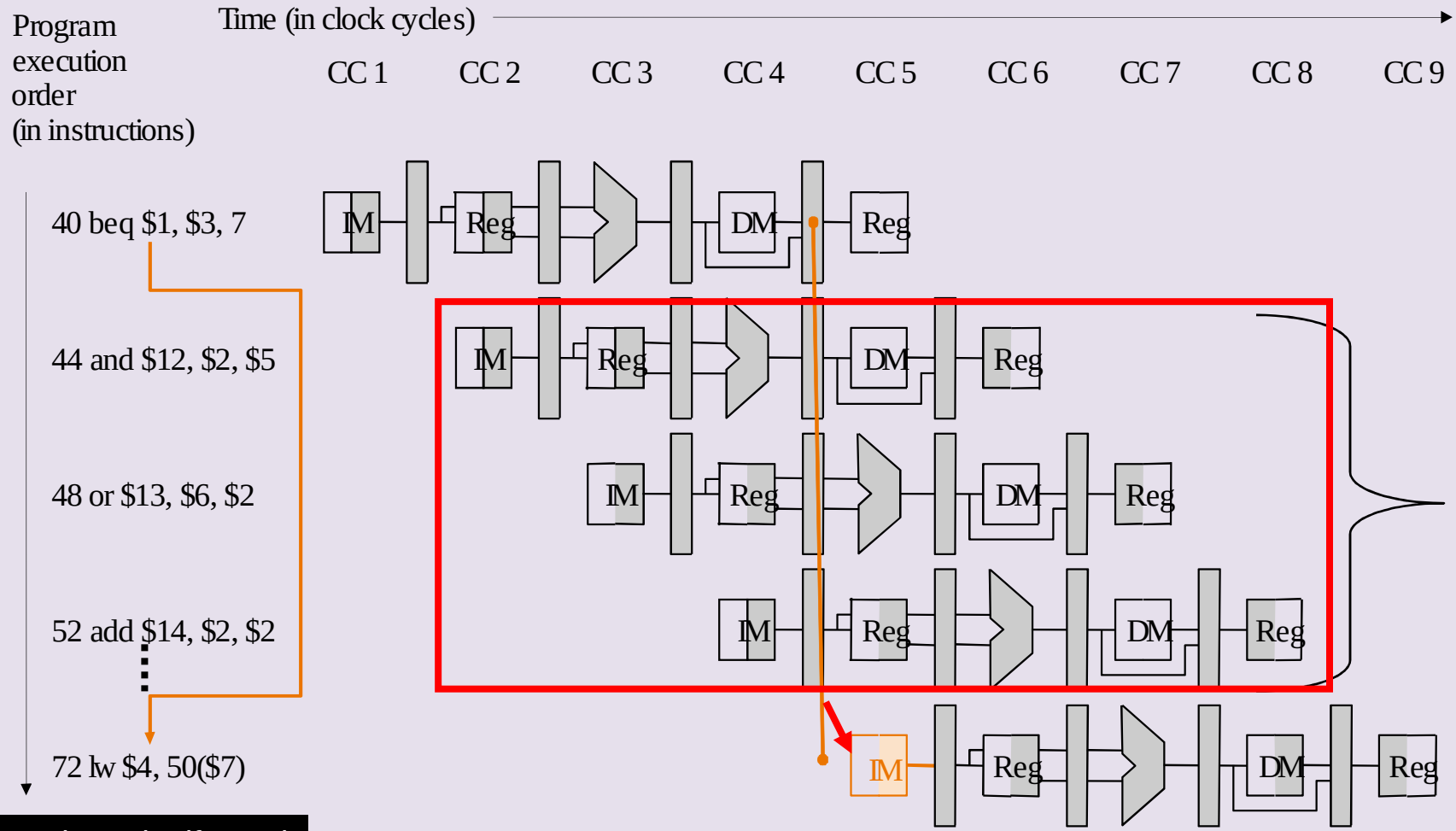
- Le test du contenu d'un **registre général**,
- Le test du contenu d'un **registre spécial**, **registre de code de condition,**
- Une **condition spécifiée** dans l'instruction.

– **La valeur testée est en général : 0, test plus rapide que, plus grand que.**

– **L'étage où le résultat est connu est assez haut** :

- ***4 dans un MIPS,***
- ***6 dans un Alpha 21164,***
- ***11 dans un Pentium IV.***

Exemple : Le résultat de la condition du branchement n'est connu **qu'à la fin de la phase 4.**
Pendant la phase 2, et 3 et 4, les trois instructions suivantes ont été chargées
Problème si la condition est vérifiée , ***On écrase des registres !!!!!!***



beq : branche if equal

- **Exemple 1 :**
- une exécution conditionnelle (**if - then - else**) qui affecte **1** ou **-1** à la variable **b** selon que la variable **a** contient **0** ou non :

```
int a;
int b;
if (a == 0)
   b = 1;
else
   b = -1;
```

0008 : CMPWI r3,0	compare **a** à 0
000C: **BNE** *+12	si différent, va a l'adresse **00000018**, **branchement conditionnel** «branch if not equal»
0010: LDI r31,1 ;	sinon affecte 1 dans b
0014: BR *+8 ;	va à $0000001C, branchement inconditionnel
0018: LDI r31,-1 ;	affecte -1 à b
001C: ...	

- Le résultat de **CMPWI** dit si le branchement est pris et enchaîne sur **LDI r31,-1** ou s'il n'est pas pris et enchaîne sur **LDI r31,1**.
- Ce test peut demander 5 à 6 cycles. Ces attentes et ces indécisions diminuent les performances.

- **Exemple 2 :** une boucle qui calcule la somme des 10 premiers nombres entiers :
  - int i;
  - int **somme = 0;**
  - **for (i = 1; i <= 10; i++)**
    - **somme = somme + i;**

0008: LDI r30,0	charge 0 dans **somme**
000C: LDI r31,1	charge 1 dans i
0010: BR *+12;	va en **$0000001C**
0014: **ADD** r30,r30,r31	ajoute i à somme
0018: ADD r31,r31,1	ajoute 1 à i
001C: CMPWI r31,10 ;	compare i à 10
0020: BLE *-12;	si <=, va en **$00000014**, branchement conditionnel : **« branch if lower or equal »**

- Dans le programme ci-dessus **ADD r30,r30,r31** serait déjà lu quand le processeur détecte que **BR *+12** est un branchement et si le branchement est pris, une ***instruction a été lancée à tort***.
-
- Certains processeurs lisent plusieurs instructions simultanément.

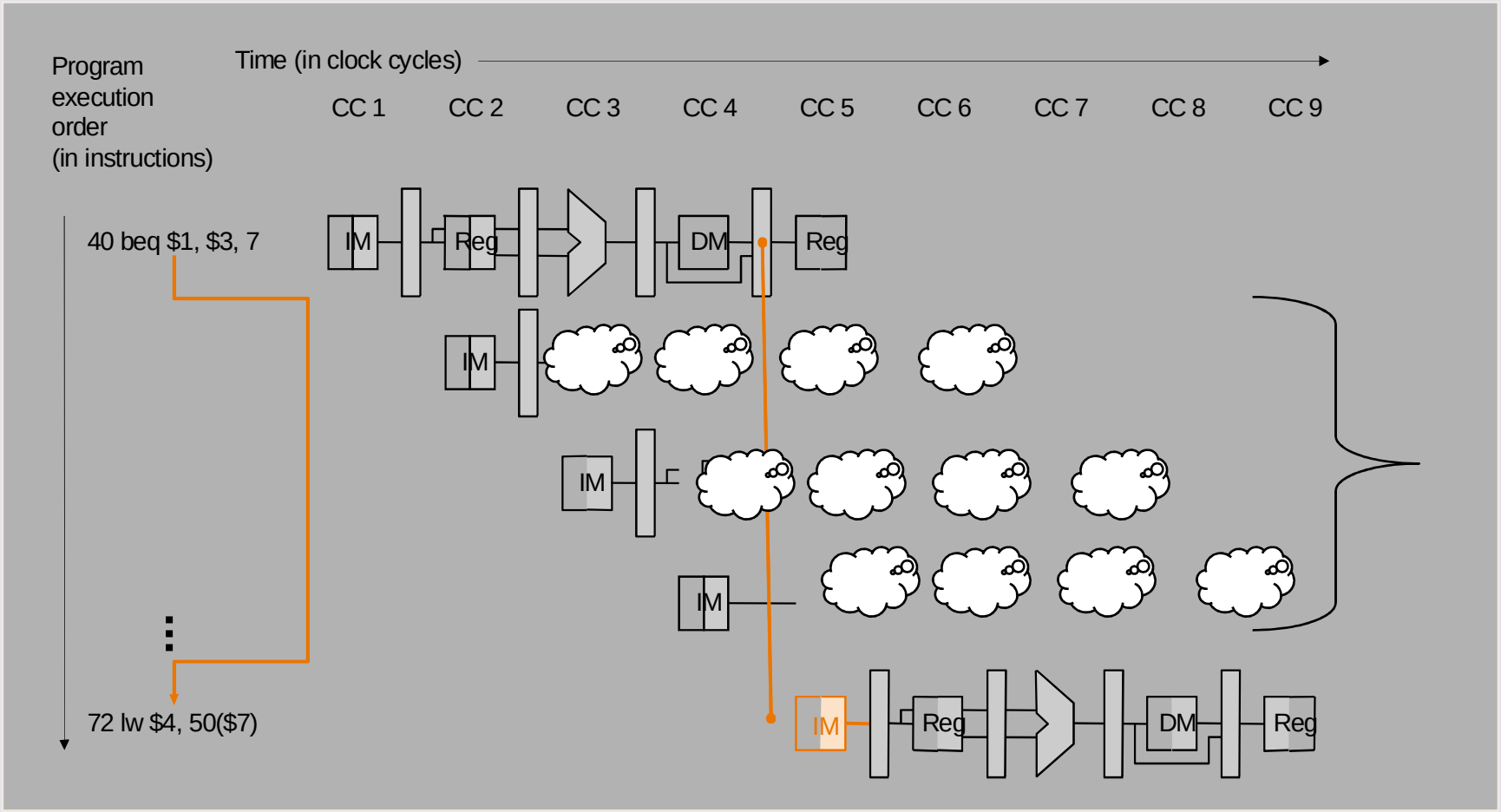
Les trois grands types de solutions

1. La neutralisation.

- **Principe :** Une fois détectée une instruction de branchement, le pipeline ***cesse d'être alimenté jusqu'à la complète exécution de l'instruction***.
- Cette solution **prévient toute difficulté**.
- Le ***temps perdu*** augmente avec la **profondeur du pipeline**.
- Cette solution est couramment employée pour **les sauts**.

La neutralisation-exemple

- Dès qu'une instruction de branchement est chargée dans IF/D, on bloque l'arrivée de nouvelle instruction jusqu'à ce que la condition soit évaluée: Perte de 3 cycles



2. Le retard de branchement (delayed branch)

- **Principe :**
  - En fonction de ***la profondeur du pipeline***, on remplit si possible les étages situés au **dessous de l'instruction de branchement** par des ***instructions indépendantes du résultat du branchement***, éventuellement par des **non opérations (NOP).**
  - Cette solution est couramment **employée pour les sauts.**

♦ Cette opération de remplissage peut être **faite de trois façons**.

- ***Les deux premières relèvent de la compilation ou d'un analyse préalable à l'exécution.***

A. ***Transporter au delà du branchement des instructions qui le précèdent*** **à condition que la condition de branchement *n'utilise pas leurs résultats*.**

B. *Exécuter* ***des instructions postérieures*** *à l'instruction de branchement qui doivent être exécutées* ***dans les deux cas.***

C. *Exécuter **les instructions** d'une branche **qui suit le branchement**.*

⇔ *On nomme cela **exécution spéculative**.*

➢ **Remarque :** Cette exécution « ***exécution spéculative »*** doit impérativement être **réversible**. Pour cela, toutes les modifications apportées aux registres, à la mémoire et à l'état interne de la machine doivent être notées pour **pouvoir annuler ces modifications si besoin est.**

Rappel : Exécution spéculative

L'exécution spéculative d'une instruction signifie que l'exécution de cette instruction est ***lancée de manière anticipée*** (i.e. sans être certain que l'instruction doit réellement être exécutée).

C'est par exemple le cas lors qu'il y a prédiction de branchements.

3. La prédiction de branchement :

- Pour limiter l'impact des branchements dans le pipeline instruction on essaie de prédire l'adresse de la prochaine instruction

- **L'importance du sujet est établie par les évaluations suivantes :**
  - L'observation d'un branchement dans 10 instructions successives est de **50%** à **99%** selon les programmes.
  - L'observation **de plus d'un branchement** dans un pipeline de grande taille est de **30%** à **99%**.

- **Soient :**
  - **C** le nombre de ***cycles perdus par mauvaise prédiction***,
  - **p** la probabilité de ***bonne prédiction***,
  - **r** le rapport entre le nombre de branchements et le nombre total d'instructions,
  - **i** le nombre d'instructions par cycle.

  - La perte **P** en nombre de cycles, due aux branchements est :

$$P = C \times ([(1-p) \times r \times i])$$

- Exemple : Pour des valeurs courantes :

  C = 5 et **r x i = 1**,

  obtenir une perte de moins de 10% demanderait une probabilité de bonne prédiction :

  (5*(1-p)*1) <0,1

  soit **p > 0,98 ou p > 98%**

  ☞ *La prédiction est alors faite quand le branchement est détecté ou à la compilation*

- **Quand on prédit le branchement, on passe dans la phase spéculative.**

  - On ne sait pas si la décision prise est bonne ou mauvaise et donc si les résultats le sont aussi ⇔ **Il faut attendre le résultat.**

  - Tant que l'on est en phase spéculative, **les registres de renommage** ne sont pas recopiés dans les registres officiels pour pouvoir défaire ce qui a été fait à tort.

  - Si la prédiction est fausse, les registres de renommage sont oubliés, sinon ils sont recopiés dans les vrais registres.

- **Les algorithmes de prédiction relèvent de deux classes:**
- **Classe 1 : la prédiction statique :** C'est la plus simple. Elle est connue depuis longtemps.
  - **la direction du branchement est fixe, définie en matériel au moment de la conception du processeur;**
  1. **Une première politique** consiste à prédire **« le branchement n'est jamais pris**". **(*predict not taken*).** L'adresse de destination est connue. Le succès n'est que de 40%.
     - **Cette stratégie (prédit non pris) est la plus courante (p.ex., dans la famille 486 de Intel).**
     - **La raison est d'ordre pratique: dans cette stratégie, les instructions exécutées spéculativement sont les instructions qui suivent immédiatement l'instruction de saut et qui sont donc vraisemblablement déjà stockées dans le cache.**
  2. **Une deuxième politique : Prédit pris (*predict taken*) :**
     - **Les sauts sont, en moyenne, pris environ 50% des fois. Les deux stratégies sont donc équivalentes par rapport au pourcentage de prédictions correctes**

- Remarque :
  - ➢ Les sauts sont, en moyenne, pris environ 50% des fois.
  - ➢ Par contre, les **sauts en arrière** (boucles) sont plus souvent pris que pas pris,
  - ➢ et inversement pour les **sauts en avant.**
  - Cette observation est à l'origine d'une troisième stratégie de prédiction :

3. **Prédiction selon la direction:** si le saut est en arrière, il est pris, s'il est en avant, il n'est pas pris.

⇨ **A distinguer avant et arrière, et si les deux destinations sont dans le même sens , à distinguer la portée courte ou longue.**

- Dans le cas **Avant-Arrière,** on prédit arrière (**BTFN : B**ackward **T**aken, **F**orward **N**ot taken).
- Dans le cas **Avant-Avant** ou **Arrière-Arrière**, on prédit celui dont la portée est la plus courte.
- Cette prédiction est statistiquement correcte dans **70 % des cas** au plus.

- Remarques :
  - Cette stratégie donne des très bons résultats (70-80%) avec une augmentation relativement restreinte de la logique de contrôle: elle est utilisée dans plusieurs processeurs (p.ex., MicroSparc, HP-PA).
  - Le choix de la direction du saut peut aussi être définie par le compilateur sur la base d'une analyse (statique) du code (p.ex. MIPS, PowerPC). Dans ce cas, on parle de prédiction semi-statique.

Classe 2 : la prédiction dynamique :

→ Dans son principe, la prédiction dynamique consiste à prédire le comportement d'un branchement en fonction du passé

→ (⇔ la direction du branchement est définie au moment de **l'exécution** du programme, sur la base d'une analyse du code)

→ Pour réaliser une prédiction dynamique le processeur mémorise le comportement du programme lors de l'exécution des sauts.

- À chaque exécution d'un branchement dans un programme, le processeur **mémorise** si le saut **était pris** ou **pas pris** dans un tampon de prédiction de branchement (**BPB** :*branch prediction buffer*) ou table historique de branchement (**BHT** : *branch history table*).
- Sur la base du comportement passé du programme pour un branchement donné, le processeur prédit son comportement pour l'exécution suivante du même saut.

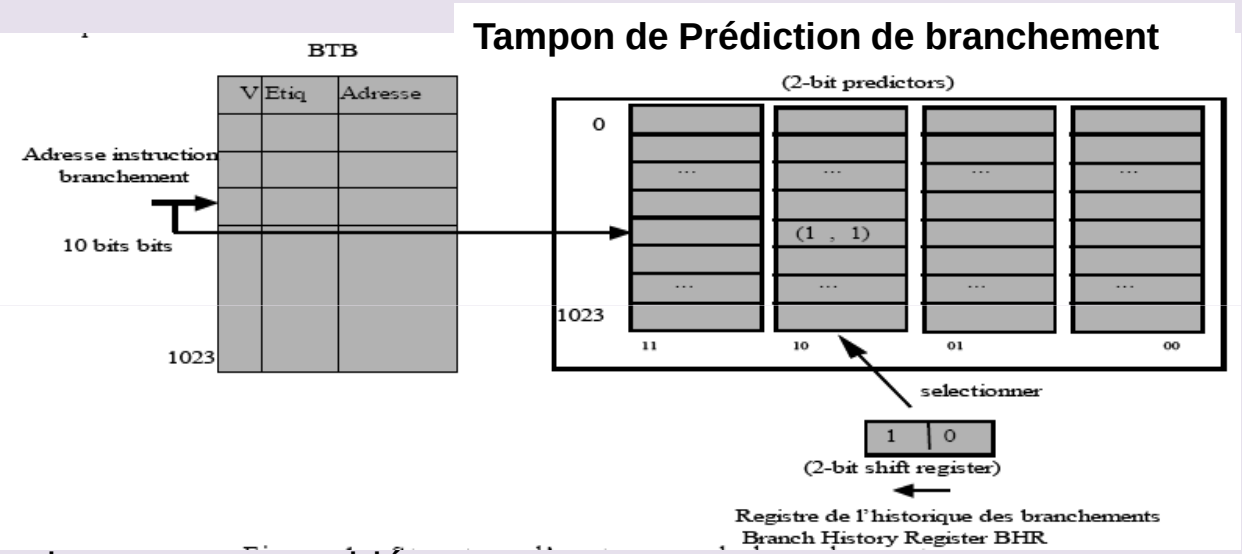
❑ ***Remarque : Par rapport à la prédiction statique, la prédiction dynamique est plus performante, mais nécessite une quantité très importante de logique de contrôle.***

- **Vocabulaires spécialisés :**
  - **Table des valeurs prédites dans le passé** (**PHT** : **P**attern **H**istory **T**able).
  - Tampon des cibles ou adresses de branchements (**BTB** : **B**ranch **T**arget **B**uffer).
  - Registre des comportements des derniers branchements (**BHR** : **B**ranch **H**istory **R**egister).

- **Identification de la nature du branchement :** Elle est faite dans le ***premier ou au pire, le second étage***.

Vocabulaires spécialisés :

- **Exemple :** structure d'un tampon de branchement avec deux bits de corrélation et deux bits de branchement



Dans cet exemple, on peut considérer que

- la PHT est un tableau à deux dimensions de 1024 entrées par 4 éléments de deux bits.
- PHT[X, Y], désigne la prédiction pour une adresse de branchement X et un BHR égale à Y,avec X<1024 et Y <4.
- De même la BTB est considérée comme une table de structures contenant trois éléments (***Valid, Etiquette, adresseBranchement***).

- ***La méthode du compteur*** **est due à Smith puis à Hwu.**
  - **L'indicateur a au moins deux bits qui fonctionnent en compteur-décompteur à double saturation.**
    - ⇨ Il est incrémenté à chaque bonne prédiction et se bloque à sa valeur maximum.
    - ⇨ Il est décrémenté à chaque mauvaise prédiction et se bloque à 0.
    - ⇨ On définit un seuil de décision.
  - **Lors d'une demande de prédiction**
    - ⇨ si la valeur courante est supérieure au seuil, on prédit le branchement pris,
    - ⇨ sinon on prédit non-pris.

– **00** fortement prédit non-pris.
– **01** faiblement prédit non-pris.
– **10** faiblement prédit pris.
– **11** fortement prédit pris.