

**LES PROBLEMES POSES PAR LES ALEAS DANS LES PIPELINES
INSTRUCTIONS
LES SOLUTIONS LOGICIELLES ET/OU MATERIELLES
ENVISAGEES**

INTRODUCTION

- Considérons l'exécution d'une séquence de code et son cheminement à travers le pipeline :

Exemple :

```
lw   R10, 9(R1)
sub  R11, R2, R3
and  R12, R4, R5
or   R13, R6, R7
add  R14, R8, R9
```

- Bien qu'une instruction démarre à **chaque cycle**, toute instruction **prend 5 cycles** pour être exécutée.
 - Les numéros des registres de destination **progressent** à chaque cycle **en direction** de l'étage **ER**.
 - A l'étage ER c'est le registre **MEM/ER** qui **fournit** le numéro du registre à écrire.
 - Cette séquence n'exhibe **aucune dépendance** entre les numéros des registres sources et destinations.
- Le séquençement du contrôle du pipeline est incorporé à sa structure même.
 - Toutes **les informations de contrôle** sont **déterminées** durant **le décodage** de l'instruction, puis transmises par les registres pipeline.
 - Lorsqu'un **étage est inactif**, les **lignes de contrôle** qui lui sont **associées** sont **désactivées**.

LES ALEAS DE DONNEES (1/3)

■ Dépendances et aléas :

- Observons une séquence avec un grand nombre de dépendances :

```
sub R4, R1, R3      # R4 écrit par sub
and R12, R4, R5     # 1er opérande (R4) dépend de sub
or  R13, R6, R4     # 2nd opérande(R4) dépend de sub
add R14, R4, R4     # 1er(R4) & 2nd (R4) dépendent de sub
sw  R15, 100(R4)    # Index(R4) dépend de sub
```

Les 4 dernières instructions sont toutes dépendantes du résultat que la première écrit dans R4.

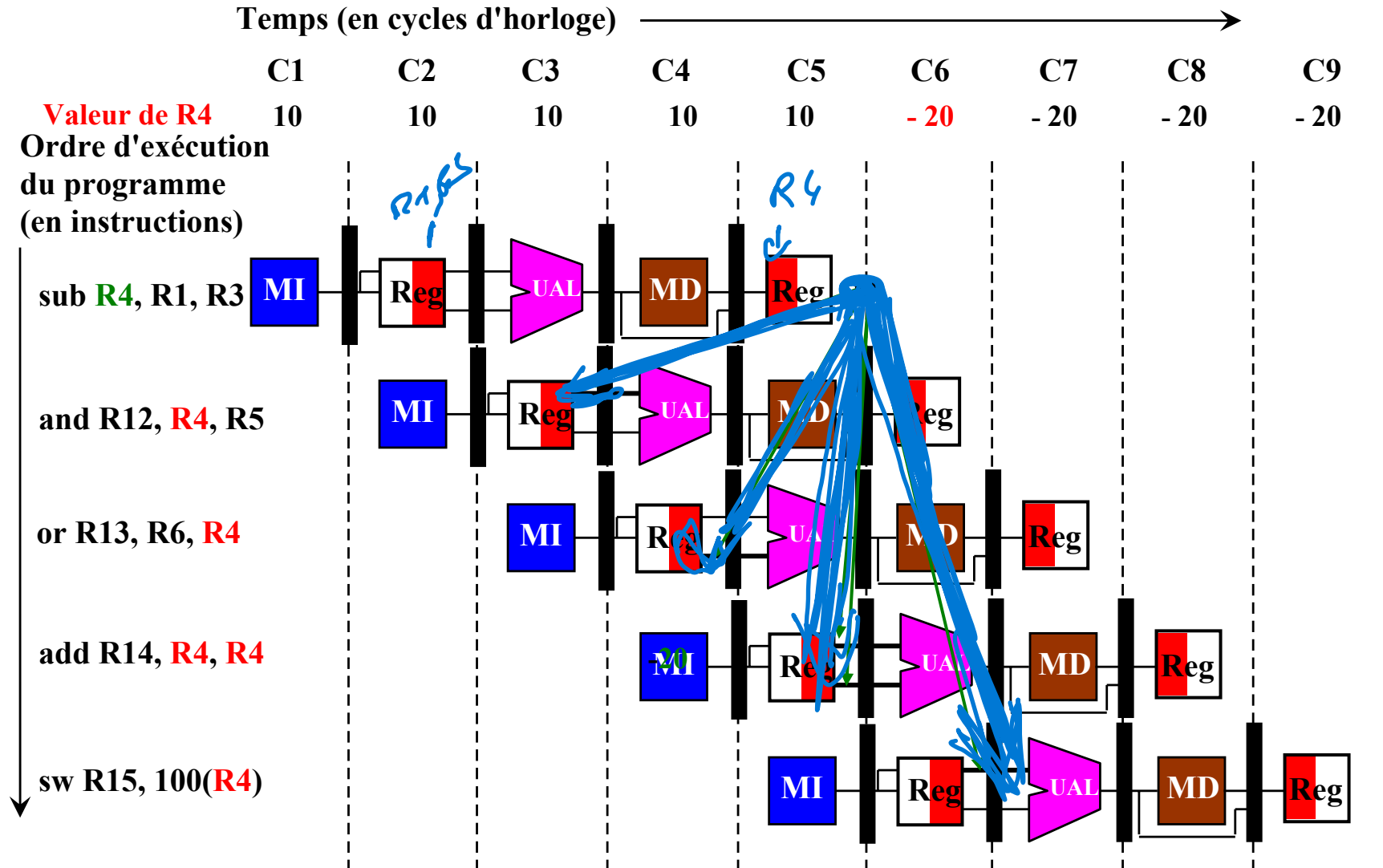
Supposons que R4 valait 10 avant **sub** et -20 après, le programmeur s'attend à ce que - 20 soit utilisé dans les instructions qui font référence au registre R4.

On va observer l'exécution de cette séquence dans une version simplifiée du chemin de données pour chaque instruction, chaque chemin de données étant aligné avec le cycle d'horloge approprié.

Pour conserver un ordre temporel correct, ce chemin de données divise le banc de registres en deux moitiés logiques : les registres lus pendant DI et les registres écrits pendant ER.

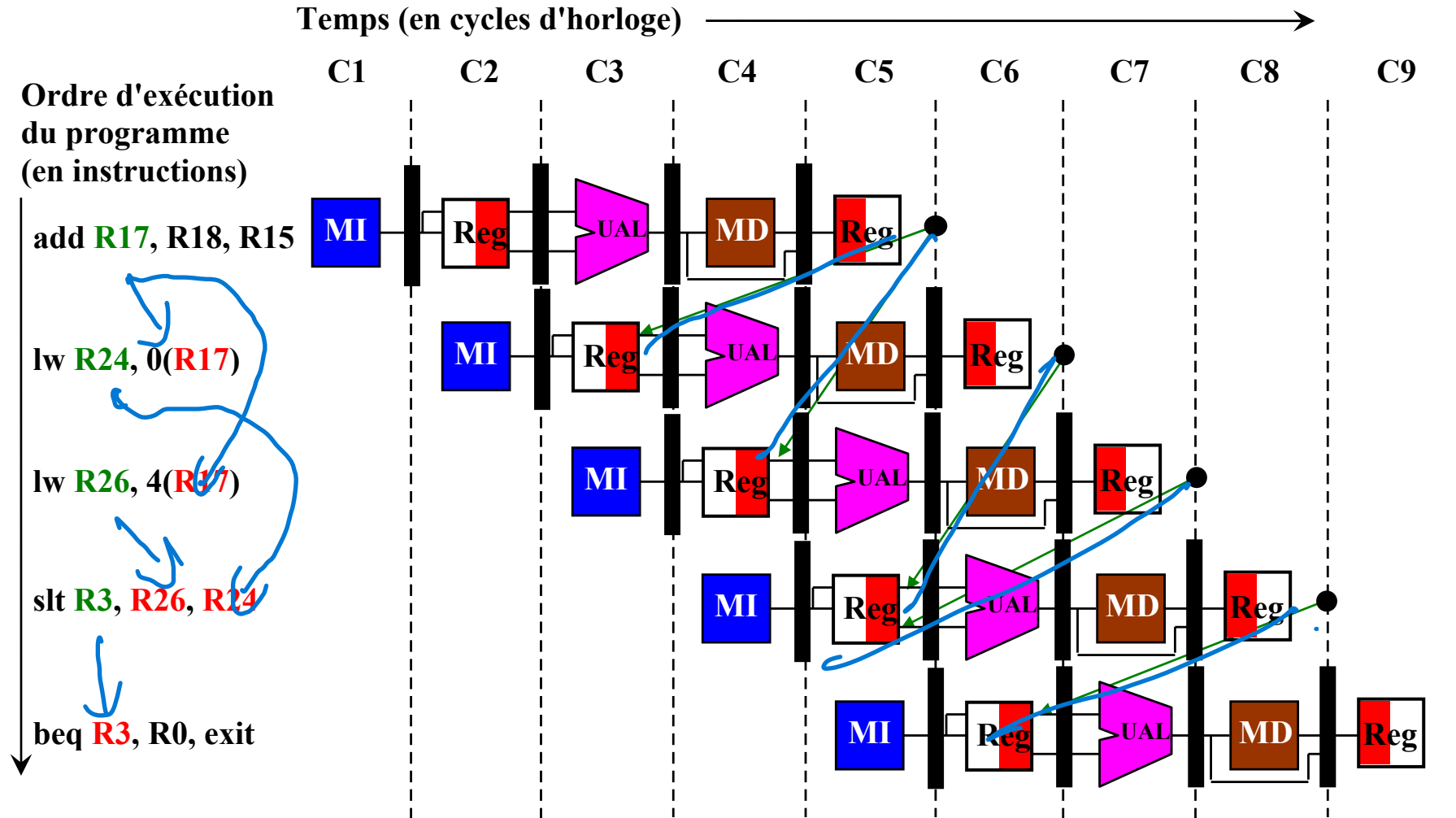
Cette division est justifiée, car les moitiés ne sont logiquement jointes que lorsqu'un même registre est lu et écrit. Il est utile de considérer pour l'instant la moitié lecture et la moitié écriture comme des sources séparées, mais on abordera ce point plus tard.

LES ALEAS DE DONNEES (2/3)



De tels aléas de données constituent une des raisons pour lesquelles les pipelines sont difficiles à concevoir : au niveau matériel, au niveau logiciel ou les deux.

LES ALEAS DE DONNEES (3/3)



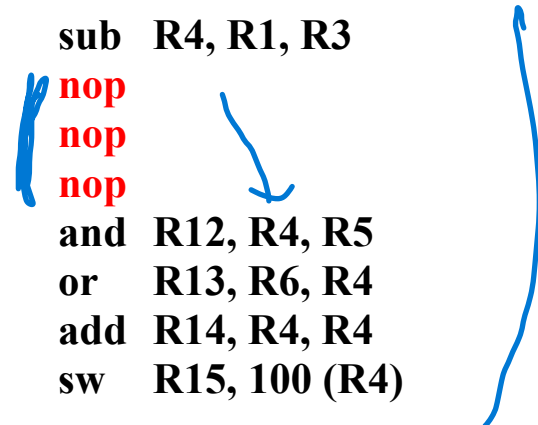
5 aléas de données :

- deux sur **R17** écrit par **add** et lu par les 2 **lw** ;
- un sur **R24** écrit par le 1^{er} **lw** et lu par **slt** ;
- un sur **R26** écrit par le 2^{ème} **lw** et lu par **slt** ;
- un sur **R3** écrit par **slt** et lu par **beq**.

LE CONTROLE POUR LES ALEAS DE DONNEES : LE COMPILATEUR (1/2)

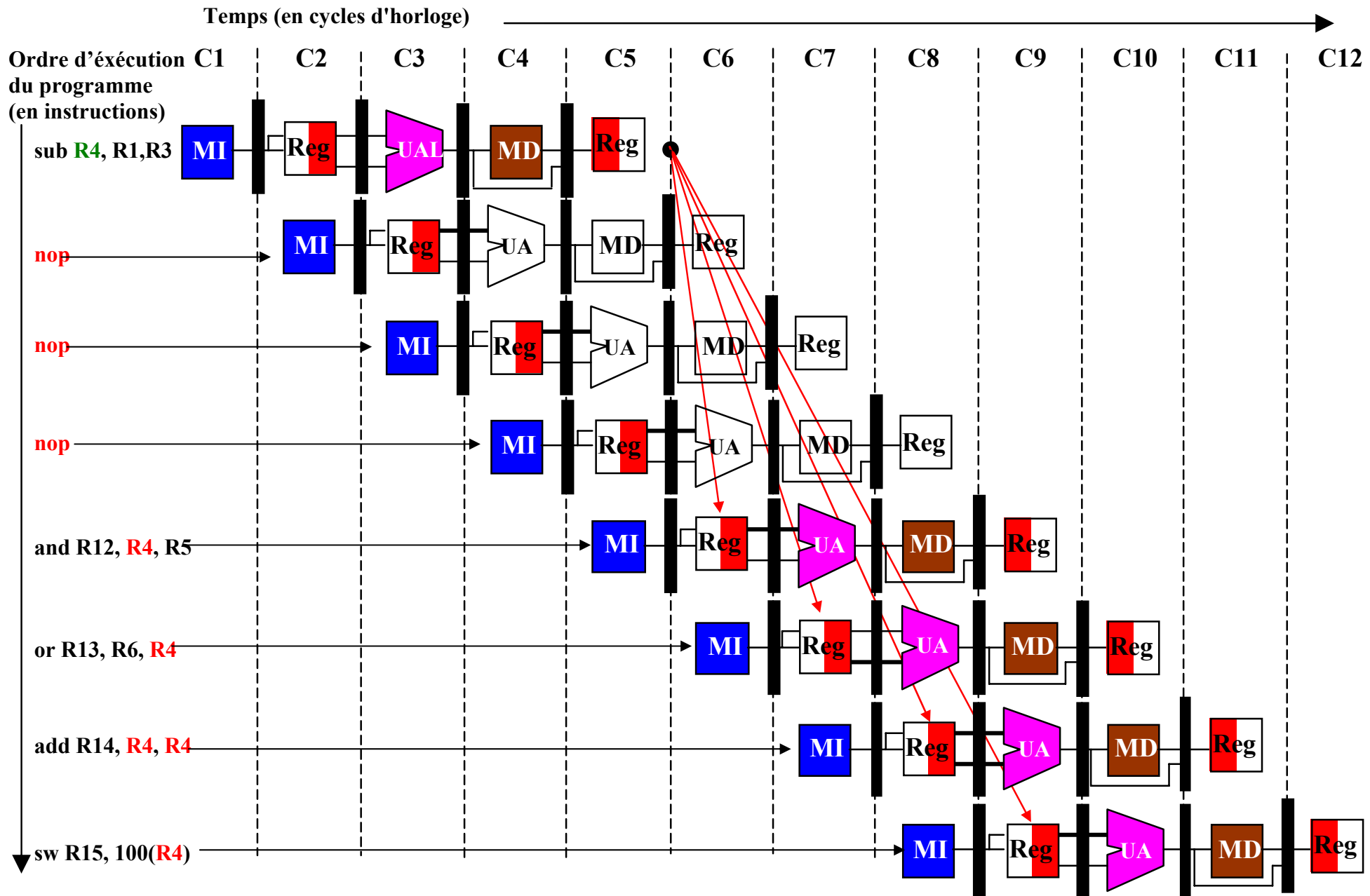
- Une des stratégies consiste *à forcer l'absence d'aléas de données : on interdit* au compilateur de générer des séquences d'instructions dépendantes rapprochées .
- Le compilateur insère par exemple trois instructions indépendantes entre les instructions **sub** et **and**, faisant ainsi disparaître l'aléa.
- Lorsqu'il ne peut trouver de telles instructions, il insère des instructions **nop**. Un **nop** n'effectue *ni lecture, ni écriture*. Le code de l'exemple précédent utilise des **nop** pour supprimer les aléas :

```
sub  R4, R1, R3
nop
nop
nop
and  R12, R4, R5
or   R13, R6, R4
add  R14, R4, R4
sw   R15, 100 (R4)
```



- Bien que *ce code fonctionne correctement pour ce pipeline à 5 étages*, ces 3 **nop** occupent 3 cycles qui ne correspondent à aucun travail utile.
- Dans le cas idéal, le compilateur trouvera des instructions utiles indépendantes à insérer pour aider les calculs, remplaçant ainsi ces instructions inactives.

LE CONTROLE POUR LES ALEAS DE DONNEES : LE COMPILATEUR (2/2)



LE CONTROLE POUR LES ALEAS DE DONNEES : LES SUSPENSIONS (1/8)

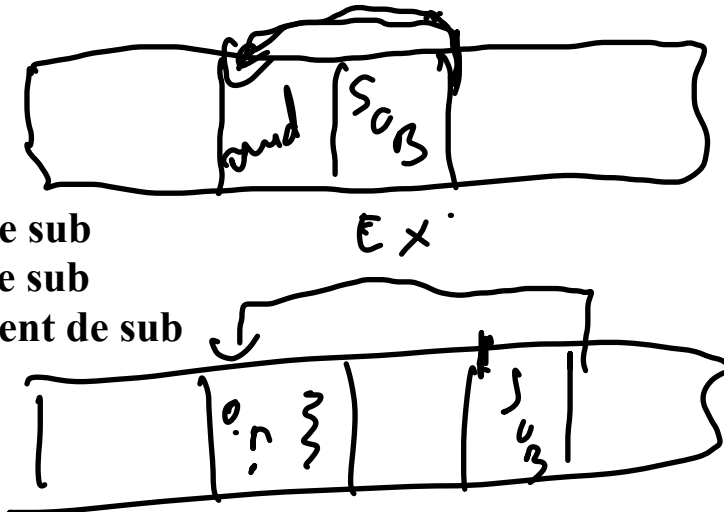
- ✚ L'approche la plus simple pour résoudre avec du matériel les aléas de données consiste à suspendre les instructions dans le pipeline jusqu'à ce que l'aléa soit résolu.
- ✚ Dans l'exemple de la première séquence, cela veut dire suspendre les instructions qui suivent l'instruction *sub* initiale jusqu'à ce que la lecture de la donnée soit possible, au cycle 6.
- ✚ Les concepteurs ont baptisé une suspension de pipeline d'instructions par le surnom « *bulle* » .
- ✚ Cette stratégie consiste *à détecter d'abord un aléa* ensuite *on suspend les instructions* dans le pipeline : (*on insère des bulles*) jusqu'à ce que l'aléa soit résolu.
- ✚ Un aléa a lieu exactement au moment où une instruction tente à l'étage DI de lire un registre qu'une instruction antérieure, à l'étage ER, s'apprête à écrire.
- ✚ Une notation faisant référence aux infos détenues dans les registres pipeline permet d'être plus précis.
- ✚ Exemple : le champ : «*EI/DI.RegistreLecture1*» fait référence au n° du registre1 (source 1) détenu par le registre *EI/DI*.
- ✚ En utilisant cette notation, les 3 paires de conditions d'aléas sont les suivantes :
 - (1a) *DI/EX.RegistreEcriture* = *EI/DI.RegistreLecture1*
 - (1b) *DI/EX.RegistreEcriture* = *EI/DI.RegistreLecture2*
 - (2a) *EX/MEM.RegistreEcriture* = *EI/DI.RegistreLecture1*
 - (2b) *EX/MEM.RegistreEcriture* = *EI/DI.RegistreLecture2*
 - (3a) *MEM/ER.RegistreEcriture* = *EI/DI.RegistreLecture1*
 - (3b) *MEM/ER.RegistreEcriture* = *EI/DI.RegistreLecture2*

LE CONTROLE POUR LES ALEAS DE DONNEES : LES SUSPENSIONS (2/8)

■ On va classer les aléas de données dans cette séquence :

```
sub  R4, R1, R3
and  R12, R4, R5
or   R13, R6, R4
add  R14, R4, R4
sw   R15, 100(R4)
```

```
# R4 écrit par sub
# 1er opérande (R4) dépend de sub
# 2nd opérande(R4) dépend de sub
# 1er (R4) & 2nd (R4) dépendent de sub
# index (R4) dépend de sub
```



- L'aléa **sub-and** : se produit sur R4 destination de sub, et source 1 de and
 ✓ L'aléa est détecté quand **and** est dans **DI** et **sub** est dans **EX** → il s'agit de la condition (1a) :
DI/EX.RegistreEcriture = EI/DI.RegistreLecture1 = R4.
- L'aléa **sub-or** est une condition (2b) : **EX/MEM.RegistreEcriture = EI/DI.RegistreLecture2 = R4 ;**
- Les aléas **sub-add** sont :
 - ✓ une condition (3a) : **MEM/ER.RegistreEcriture = EI/DI.RegistreLecture1 = R4 ;**
 - ✓ une condition (3b) : **MEM/ER.RegistreEcriture = EI/DI.RegistreLecture2 = R4.**

■ Pas d'aléa de données entre **sub** et **sw** : **sw** lit R4 après que **sub** l'ait écrit.

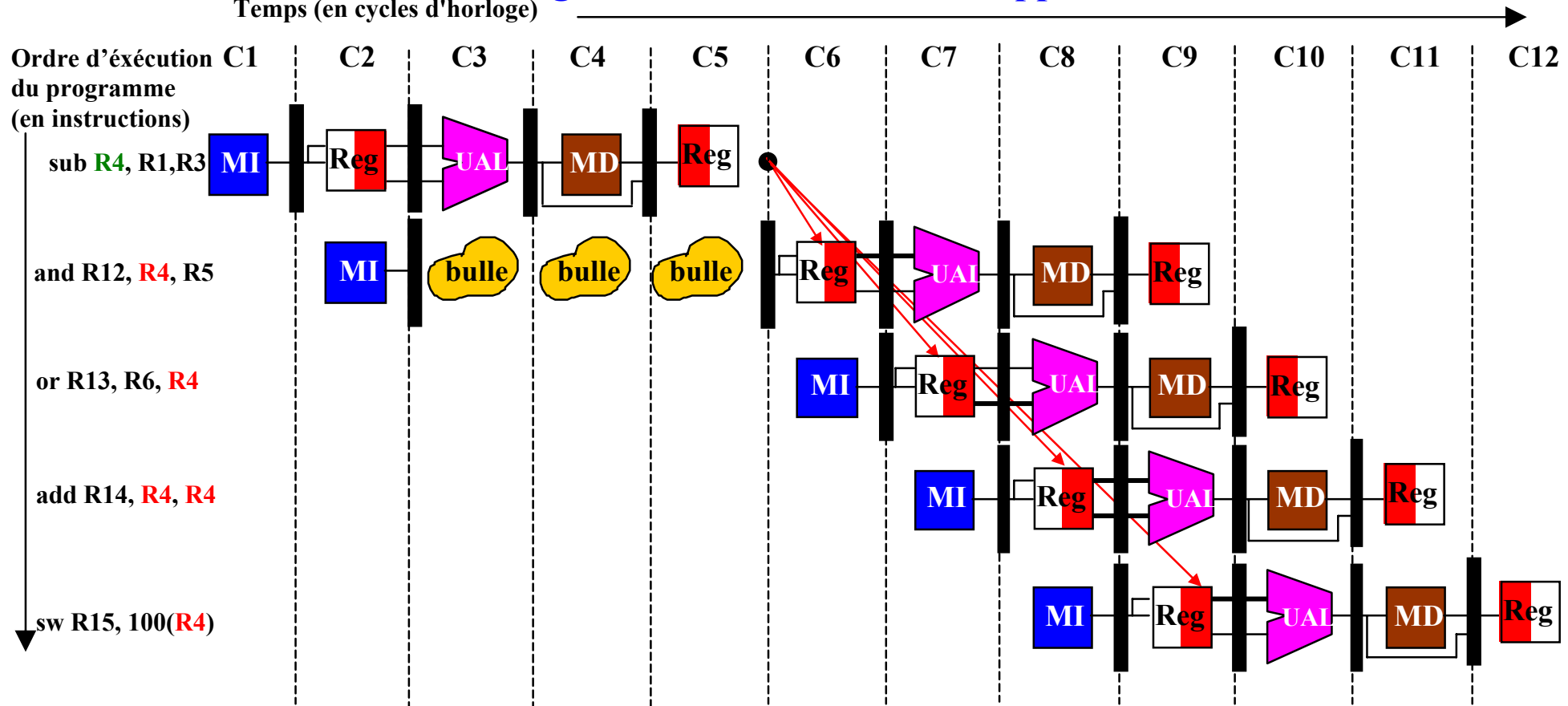
■ Si aléa alors suspendre à l'étage **DI** l'instruction dépendante jusqu'à ce que l'instruction ayant provoqué l'aléa soit achevée.

LE CONTROLE POUR LES ALEAS DE DONNEES : LES SUSPENSIONS (3/8)



L'insertion de 3 bulles à l'étage DI de l'instruction *and* supprime les aléas.

Temps (en cycles d'horloge)



Politique prudente : certaines instructions n'écrivent pas de résultat → parfois des suspensions inutiles.

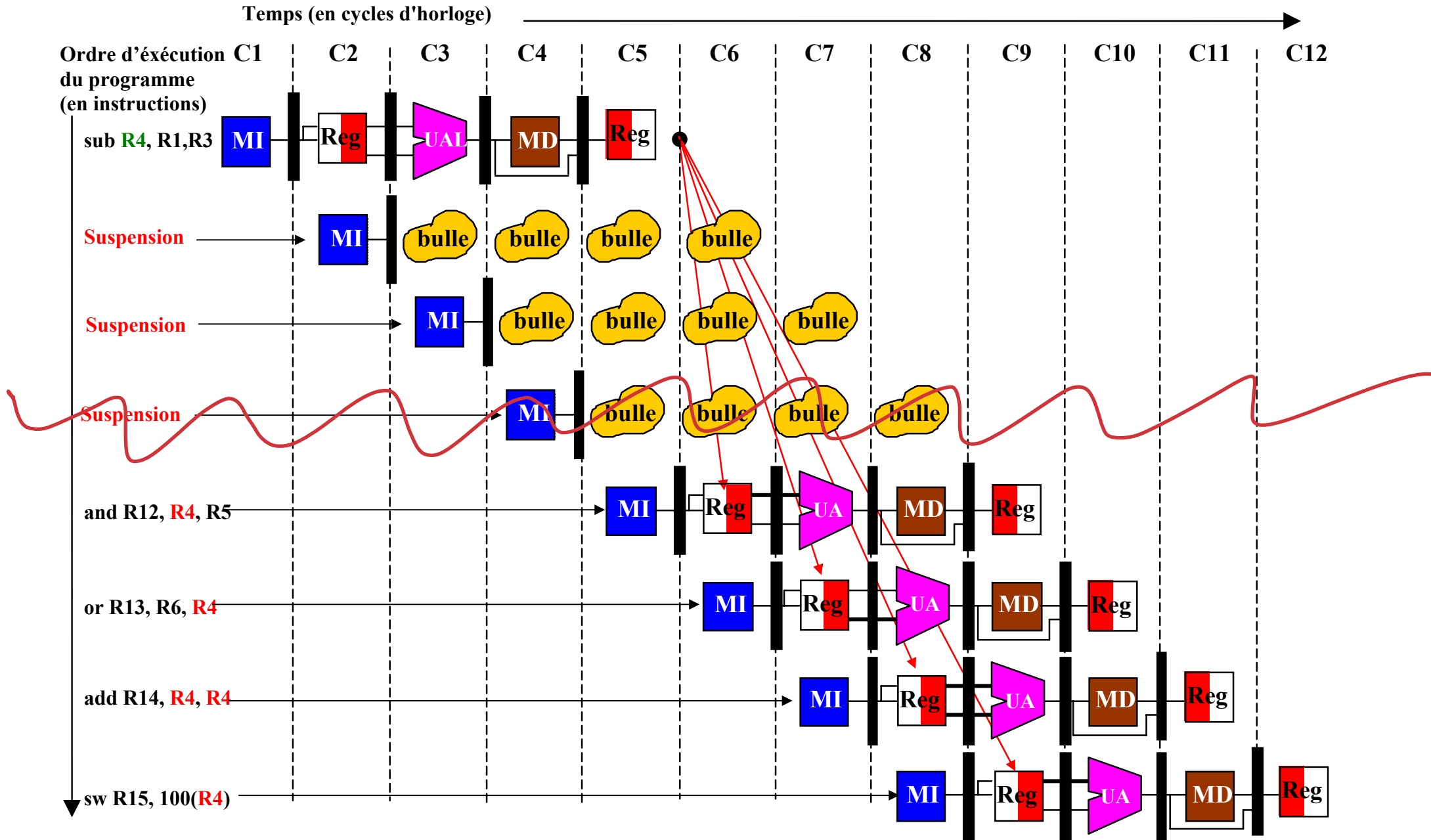
Solution : tester si *EcrireReg* est actif → il suffit d'examiner le contrôle de l'étage ER dans les registres pipeline pendant les étages EX et MEM, ER.

Le registre **DI/EX** a 2 champs *RegistreEcriture*, on doit aussi utiliser le signal *RegDst* à l'étage EX pour connaître le numéro approprié du registre à écrire.

LE CONTROLE POUR LES ALEAS DE DONNEES : LES SUSPENSIONS (4/8)

- On sait détecter les aléas potentiels, il faut savoir suspendre les instructions en présence d'aléa.
 - Si l'instruction à l'étage **DI** est suspendue, alors celle à l'étage **EI** doit aussi l'être.
- Empêcher ces 2 instructions de progresser : empêcher la modification de CP et du registre pipeline EI/DI.
 - l'instruction à l'étage **EI** continuera d'être relue en utilisant le même **CP**,
 - les registres à l'étage **DI** continueront d'être relus en utilisant la même instruction dans **EI/DI**.
- La suspension du pipeline, doit produire le même effet que le *nop*, mais cette fois l'effet du « *nop* » débute à l'étage EX.
 - Ceci revient à *désactiver* les signaux dans les étages **EX**, **MEM** et **ER** : générer un *nop*
- Donc *insérer une bulle* dans le pipeline revient à *mettre les champs de contrôle EX, MEM et ER du registre DI/EX à l'état inactif*.
 - Ces valeurs de contrôle sont passées vers l'avant à chaque cycle avec l'effet souhaité : ni registres ni mémoires ne sont écrits.
- Dans le cas pire l'instruction suspendue reste dans le registre pipeline EI/DI pendant 3 cycles et lance 3 bulles séparées dans le pipeline.

LE CONTROLE POUR LES ALEAS DE DONNEES : LES SUSPENSIONS (5/8)



LE CONTROLE POUR LES ALEAS DE DONNEES : LES SUSPENSIONS (6/8)

✚ On sait détecter les aléas et provoquer des suspensions → il reste à définir le matériel qui met en œuvre les suspensions. il s'appuie sur un petit nombre de portes logiques pour réaliser les tests de conditions d'aléas :

1. Aléa EX : ce test est le plus compliqué → on doit déterminer si l'instruction au format I ou au format R (RegDst = 0 ou 1). L'aléa se situe ici entre les instructions des étages DI et EX.

DI/EX.EcrireReg et ((DI/EX.RegDst = 0 et DI/EX.RegistreEcritureRt = EI/DI.RegistreLecture1) ou (DI/EX.RegDst = 1 et DI/EX.RegistreEcritureRd = EI/DI.RegistreLecture1) ou (DI/EX.RegDst = 0 et DI/EX.RegistreEcritureRt = EI/DI.RegistreLecture2) ou (DI/EX.RegDst = 1 et DI/EX.RegistreEcritureRd = EI/DI.RegistreLecture2)).

2. Aléa MEM : l'aléa se situe ici entre les instructions des étages DI et MEM.

EX/MEM.EcrireReg et ((EX/MEM.RegistreEcriture = EI/DI.RegistreLecture1) ou (EX/MEM.RegistreEcriture = EI/DI.RegistreLecture2)).

3. Aléa ER : l'aléa se situe ici entre les instructions des étages DI et ER.

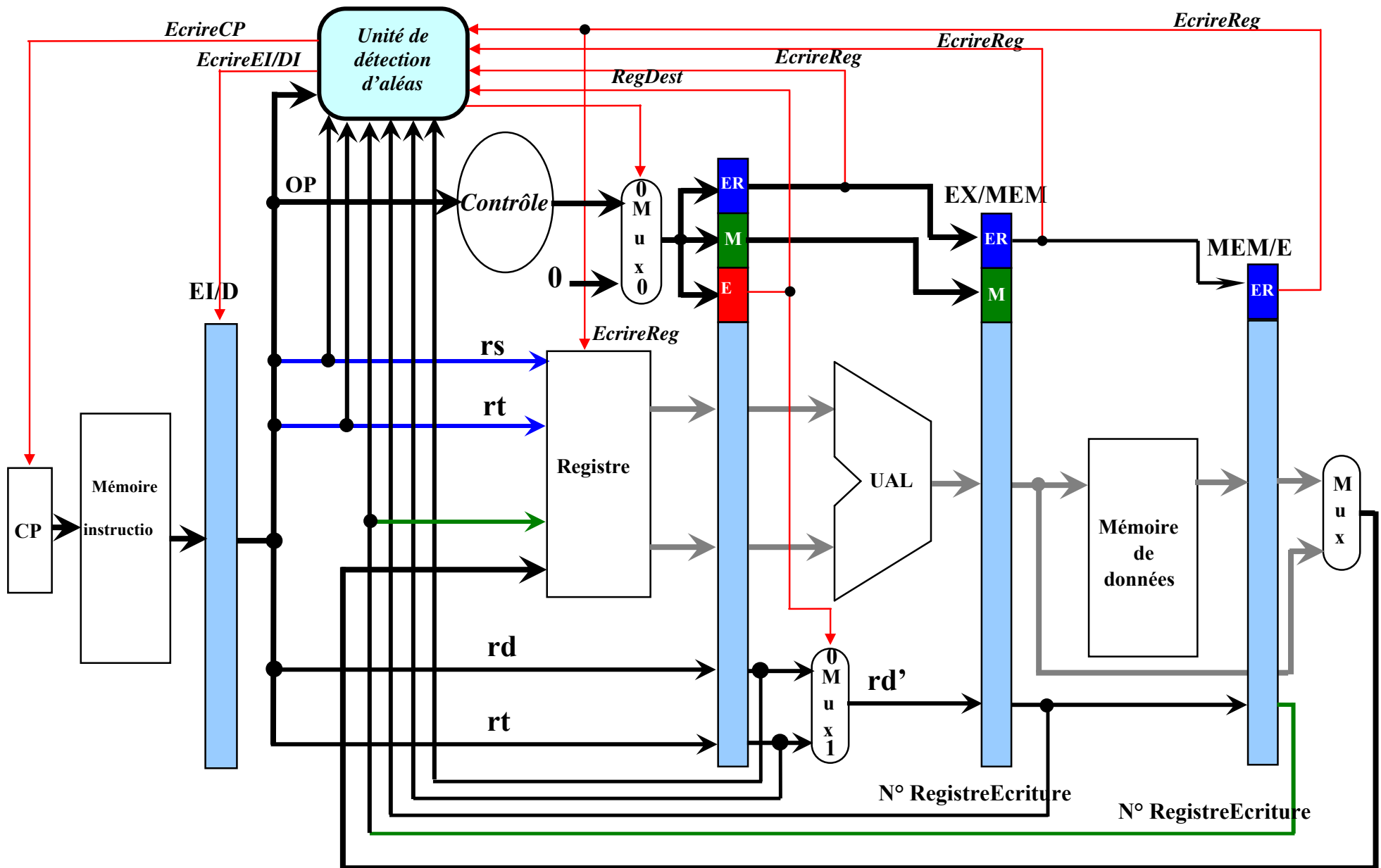
MEM/ER.EcrireReg et ((MEM/ER.RegistreEcriture = EI/DI.RegistreLecture1) ou (MEM/ER.RegistreEcriture = EI/DI.RegistreLecture2)).

✚ L'aléa ER peut être évité, si l'écriture d'un registre sa lecture dans le même cycle retourne ce qui a été écrit, ce qui est vrai pour beaucoup de mises en œuvre de bancs de registres, l'aléa sera éliminé.

✚ une nouvelle unité “*Unité de détection d'aléas*” contrôlera l'écriture de CP et de EI/DI ainsi que les multiplexeurs qui choisissent entre les valeurs de contrôle réelles et des valeurs qui correspondent à l'état inactif.

✚ L'Unité de détection d'aléas suspend et désactive les champs de contrôle si l'un des trois tests d'aléas est vrai.

LE CONTROLE POUR LES ALEAS DE DONNEES : LES SUSPENSIONS (7/8)



LE CONTROLE POUR LES ALEAS DE DONNEES : LES SUSPENSIONS (8/8)

Autres amélioration des conditions de test des aléas :

- Une autre amélioration concerne le registre R0 qui vaut toujours 0, les suspensions dues à l'écriture ou la lecture du registre R0 doivent donc aussi être évitées.
- une amélioration peut être apportée en évitant un test sur RegistreLecture2 dans le cas d'une instruction au format R suivie d'une instruction arithmétique ou logique au format immédiat ou d'un chargement, puisque ces instructions n'utilisent que RegistreLecture1.

Exemple : considérons les 2 instructions suivantes :
add R8, R1, R2
lw R8, 1200 (R5)

Ce qui donne

op	rs	rt	(rd)	(ValDec)	(fonct)/adresse
0	1	2	8	0	32
35	5	8	1200		

La valeur 8 du champ rt conduirait à la suspension du lw alors que son champ rt indique son registre de destination, et pas un registre source → cette combinaison ne doit pas être suspendue.

Il faut donc éviter les suspensions dues aux tests sur RegistreLecture2 pour les chargements et les instructions arithmétiques et logiques en immédiat.

Remarque : le test d'aléa se fait entre l'instruction à l'étage DI et celles aux étages EX, MEM et ER. Ce test a lieu au milieu du cycle et la machine doit décider de la valeur à placer sur les signaux de contrôle avant la fin du cycle par l'intermédiaire du multiplexeur.

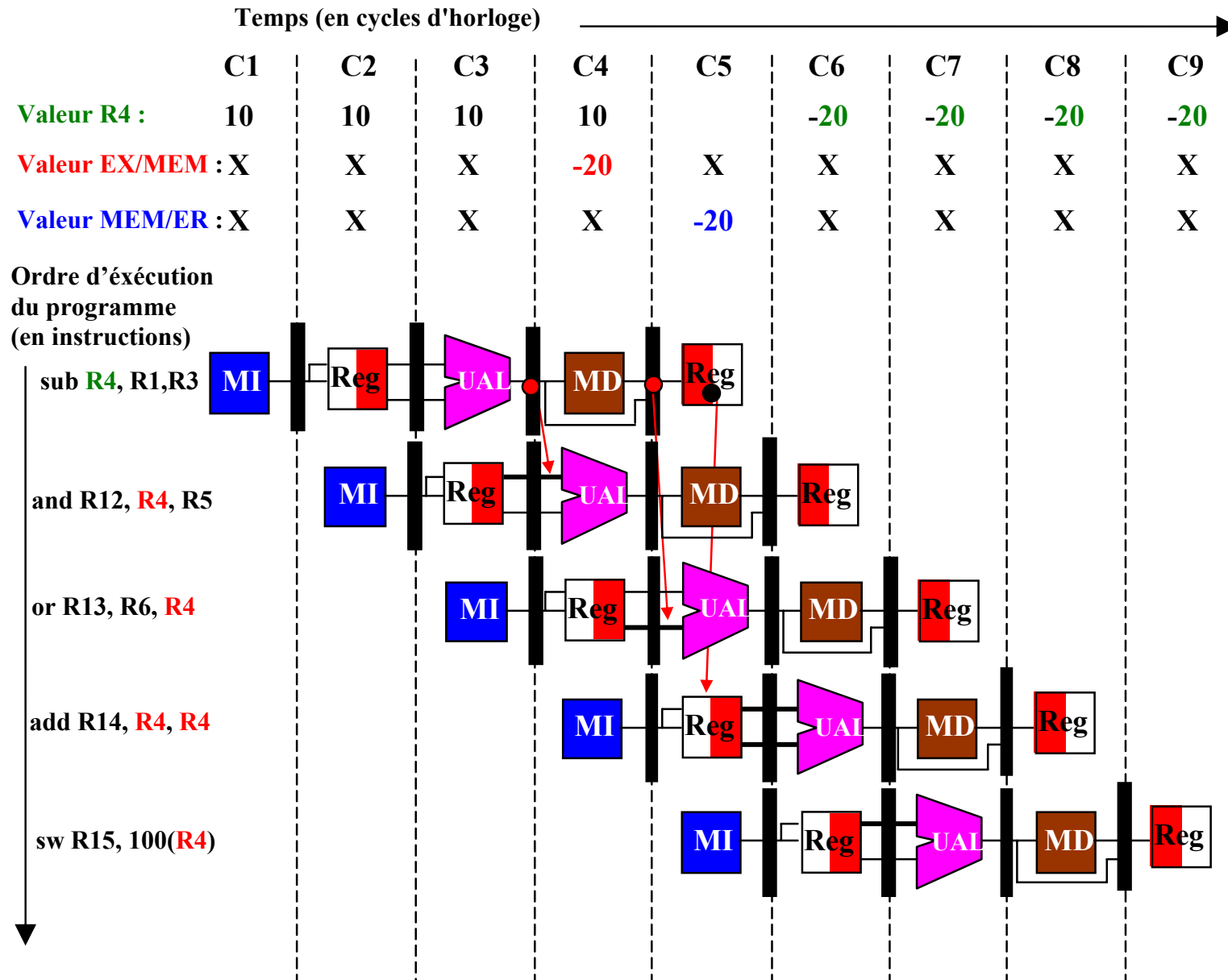
REDUIRE LES ALEAS DE DONNEES : L'ENVOI (1/7)

- Suspendre le pipeline garantit une exécution correcte lorsque le compilateur génère des instructions dépendantes proches les unes des autres, mais le prix à payer pour cette exactitude est une baisse des performances.
- Si on observe plus attentivement la même séquence de code, on constate que la valeur dont a besoin :
 - l'instruction *and* en entrée de l'UAL au cycle d'horloge 4 est en fait présente dans le champ RésultatUAL du registre pipeline EX/MEM de l'instruction *sub* ;
 - l'instruction *or* en entrée de l'UAL au cycle d'horloge 5 se trouve dans le registre pipeline MEM/ER de l'instruction *sub*.
- Si on modifie le banc de registres de façon à ce qu'il retourne la valeur écrite lorsqu'une lecture et une écriture ont lieu sur le même registre, le chemin de données pourra également fournir l'opérande de l'instruction *add* qui suit.
- Une dépendance prend sa source dans *un registre pipeline* plutôt que d'attendre l'écriture des vrais registres à l'étage ER.

Les données nécessaires sont présentes dans les registres pipeline au moment où elles doivent être utilisées par des instructions ultérieures, ce qui suggère *un raccourci* qui permettrait de réduire les baisses de performances auxquelles conduisent les suspensions.

REDUIRE LES ALEAS DE DONNEES : L'ENVOI (2/7)

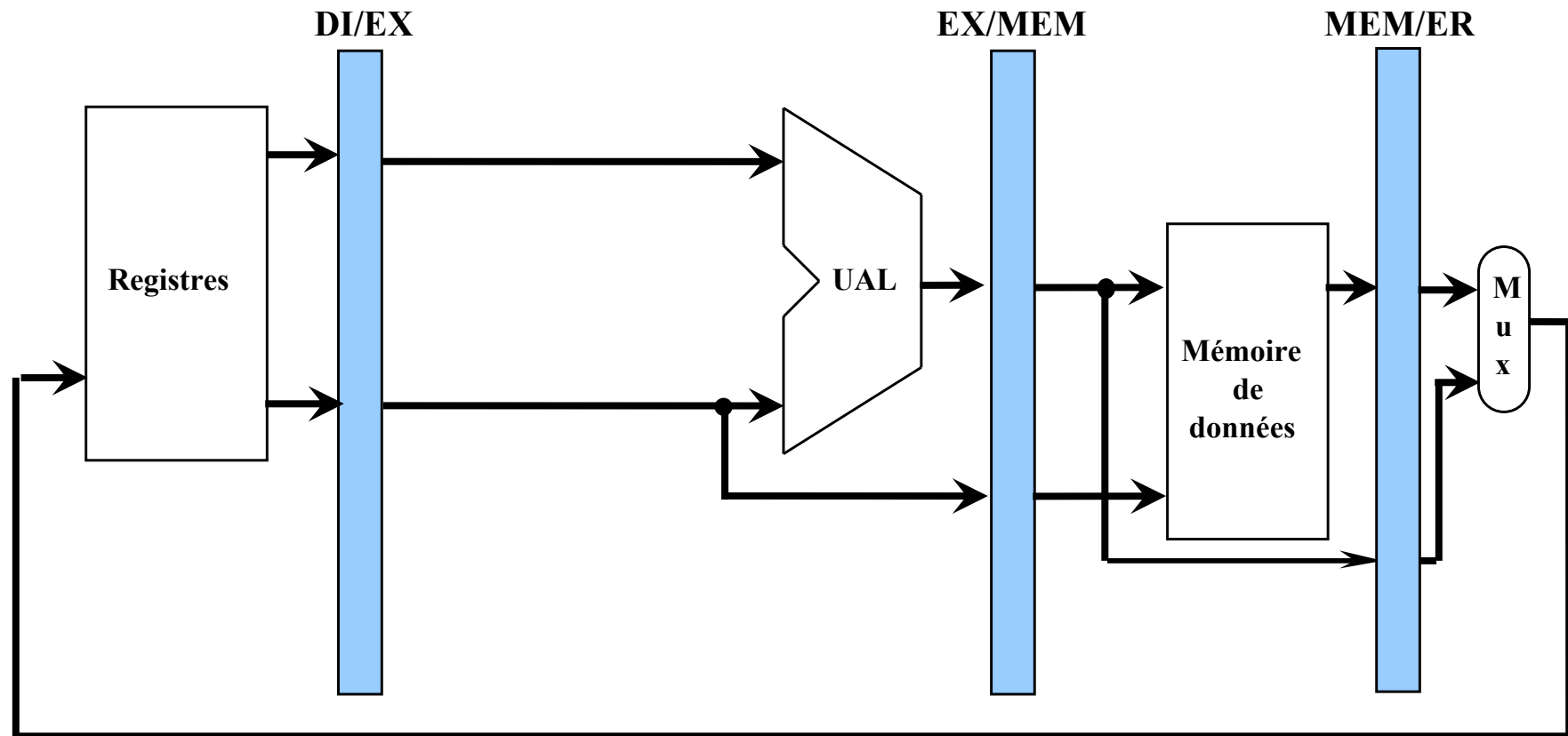
Pour la même séquence de code les dépendances entre *les registres pipeline* et *les entrées de l'UAL*.



REDUIRE LES ALEAS DE DONNEES : L'ENVOI (3/7)

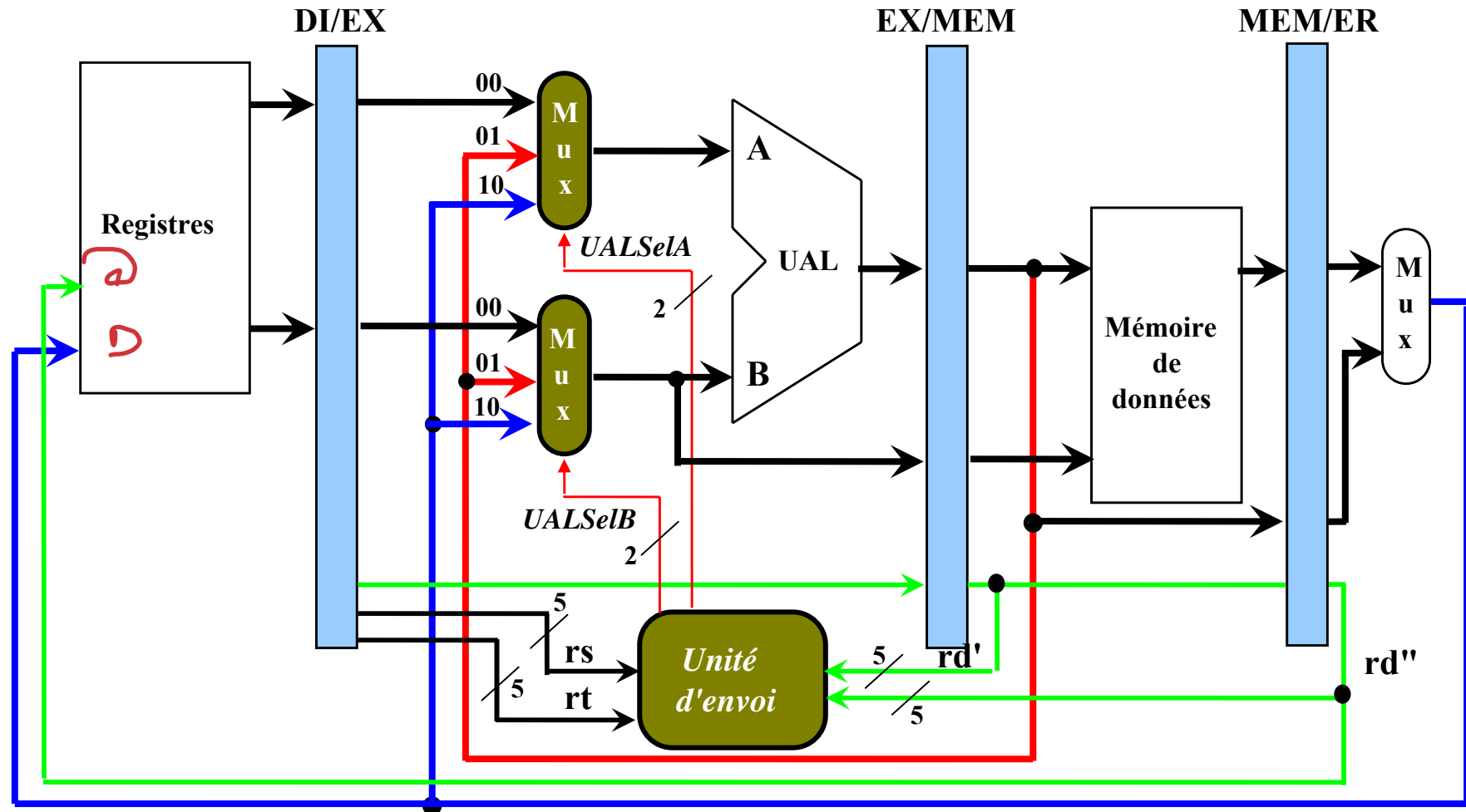
Pour le moment, on suppose que les seules instructions qu'ont besoin de faire un envoi sont les 4 instructions au format R : *add*, *sub*, *and*, et *or*.

Le chemin de données pour ces instructions comprend : l'UAL, le banc de registres, et les registres pipeline avant l'ajout de l'Unité d'envoi.



REDUIRE LES ALEAS DE DONNEES : L'ENVOI (4/7)

Le chemin de données après l'ajout de l'unité d'envoi et les multiplexeurs d'envoi.



L'unité d'envoi se trouve à l'étage EX, car les multiplexeurs d'envoi de l'UAL s'y trouvent. On doit donc lui transmettre par l'intermédiaire du registre pipeline DI/EX les numéros de registres de l'étage DI pour savoir s'il faut ou non envoyer des valeurs.

REDUIRE LES ALEAS DE DONNEES : L'ENVOI (5/7)

Les valeurs des lignes de contrôle pour les multiplexeurs de l'UAL qui sélectionnent soit les valeurs normales des registres soit les valeurs envoyées sont données dans le tableau suivant :

Contrôle Mux	Source	Explication
UALSelA = 00	DI/EX	Le 1er opérande de l'UAL provient des registres normaux.
UALSelA = 01	EX/MEM	Le 1er opérande de l'UAL provient de l'envoi du résultat précédent de l'UAL.
UALSelA = 10	MEM/ER	Le 1er opérande de l'UAL provient d'un envoi depuis la mém. de données ou de l'envoi d'un résultat antérieur de l'UAL.
UALSelB = 00	DI/EX	Le 2nd opérande de l'UAL provient des registres normaux.
UALSelB = 01	EX/MEM	Le 2nd opérande de l'UAL provient de l'envoi du résultat précédent de l'UAL.
UALSelB = 10	MEM/ER	Le 2nd opérande de l'UAL provient d'un envoi depuis la mém. de données ou de l'envoi d'un résultat antérieur de l'UAL.

Les deux conditions d'envoi et l'emplacement du résultat sont donnés par :

1. Aléa EX :

Si (EX/MEM.EcrireReg et (EX/MEM.RegistreEcriture = DI/EX.RegistreLecture1)) **Alors** **UALSelA = 01**

Si (EX/MEM.EcrireReg et (EX/MEM.RegistreEcriture = DI/EX.RegistreLecture2)) **Alors** **UALSelB = 01**

Dans ce cas le résultat de l'instruction précédente est envoyé à l'une ou l'autre des entrées de l'UAL.

2. Aléa MEM :

Si (MEM/ER.EcrireReg et (MEM/ER.RegistreEcriture = DI/EX.RegistreLecture1)) **Alors** **UALSelA = 10**

Si (MEM/ER.EcrireReg et (MEM/ER.RegistreEcriture = DI/EX.RegistreLecture2)) **Alors** **UALSelB = 10**

On teste ici les mêmes numéros de registres, mais c'est le fait que l'instruction dépende d'une opération de l'UAL (MemversReg =0) ou d'une instruction de chargement (MemversReg =1) qui définit la valeur envoyée.

Il n'y a pas de troisième aléa, car on suppose que le banc de registres fournit le résultat correct si l'instruction à l'étage DI lit le même registre que celui écrit par l'instruction à l'étage ER.

Ce banc modifié est une autre forme d'envoi, mais qui survient cette fois à l'intérieur du banc.

REDUIRE LES ALEAS DE DONNEES : L'ENVOI (6/7)

Problème : les aléas peuvent avoir lieu à la fois aux étages EX et MEM pour une même entrée de l'UAL.

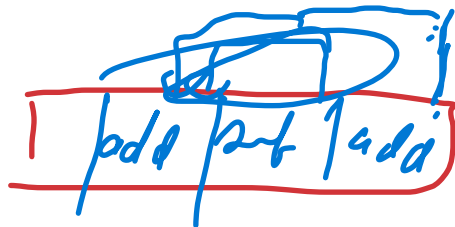
Exemple : si on somme les éléments d'un vecteur dans un *même* registre, une séquence d'instructions *lira et écrira toujours dans le même registre*.

Solution : dans ce cas, la priorité est à l'aléa se trouvant le plus proche de l'étage DI dans l'ordre d'exécution du programme : ici l'aléa EX.

Modification du contrôle : le contrôle pour l'aléa MEM est donc le suivant :

Si (MEM/ER.EcrireReg
et (EX/MEM.RegistreEcriture \neq DI/EX.RegistreLecture1)
et (MEM/ER.RegistreEcriture = DI/EX.RegistreLecture1)) Alors $UALSelA = 10$

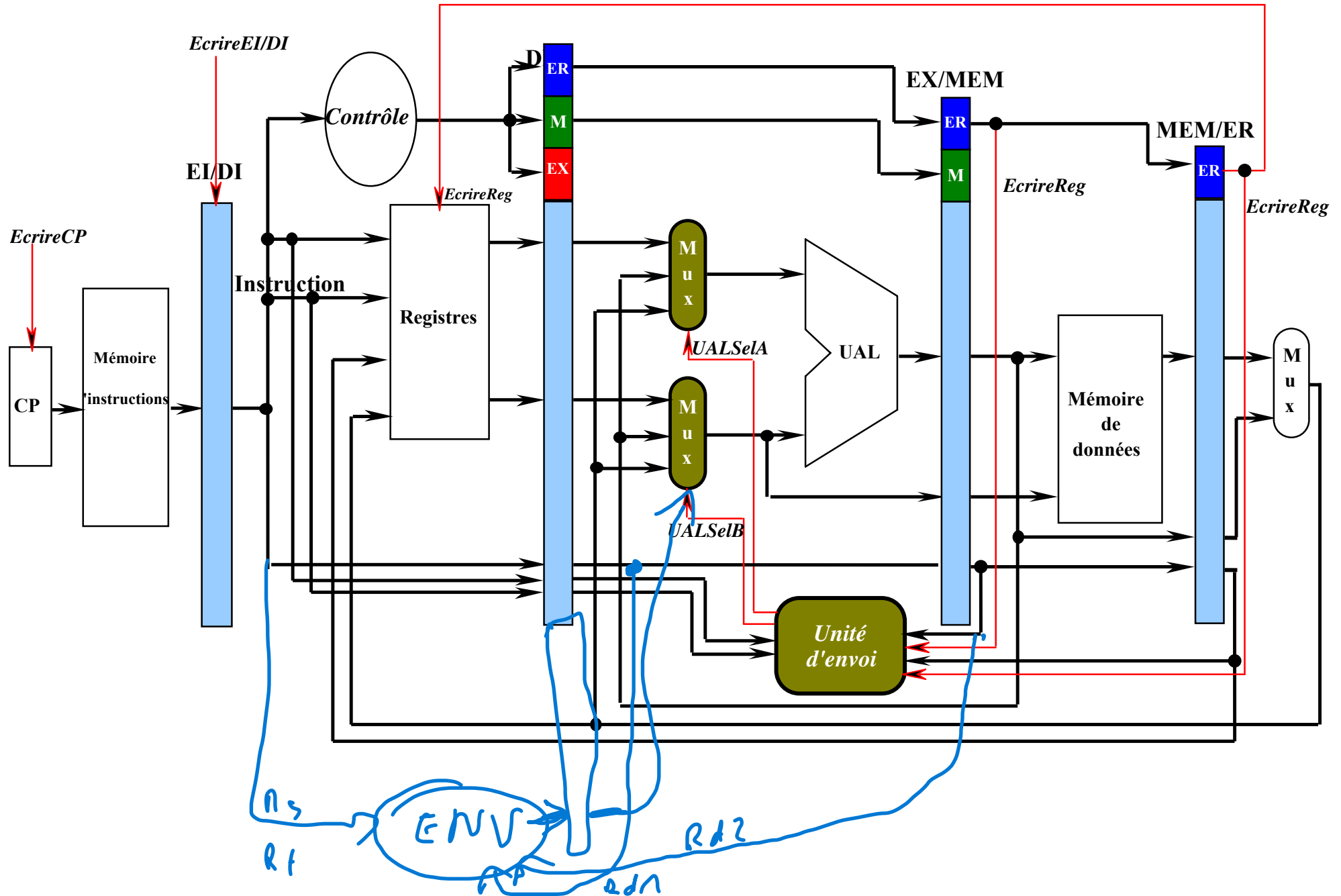
Si (MEM/ER.EcrireReg
et (EX/MEM.RegistreEcriture \neq DI/EX.RegistreLecture2)
et (MEM/ER.RegistreEcriture = DI/EX.RegistreLecture2)) $UALSelB = 10$



add r4, ...
sub r4, ...
add r1, r4

REDUIRE LES ALEAS DE DONNEES : L'ENVOI (7/7)

Le matériel nécessaire pour contrôler les mux. d'envoi aux entrées de l'UAL par l'Unité d'envoi est le suivant:



AMELIORATION DE L'UNITE D'ENVOI



Les conditions d'envoi de données sont en fait un peu plus compliquées :

- Le registre R0 n'est jamais modifié, donc toute utilisation de R0 doit fournir un 0 comme opérande.
- Les conditions sont donc valables tant que $DI/EX.RegistreLecture1 \neq 0$ et $DI/EX.RegistreLecture2 \neq 0$.



Il est probable que le temps pour sélectionner les entrées de l'UAL fasse partie du chemin critique et donc le temps de positionnement des signaux de contrôle des multiplexeurs d'envoi par l'Unité d'envoi serait limite.



Le matériel serait peut-être plus rapide avec une mise en œuvre différente de l'Unité d'envoi qui consisterait à déterminer pendant l'étage DI le contrôle des multiplexeurs d'envoi aux entrées de l'UAL.



Il faudrait donc :

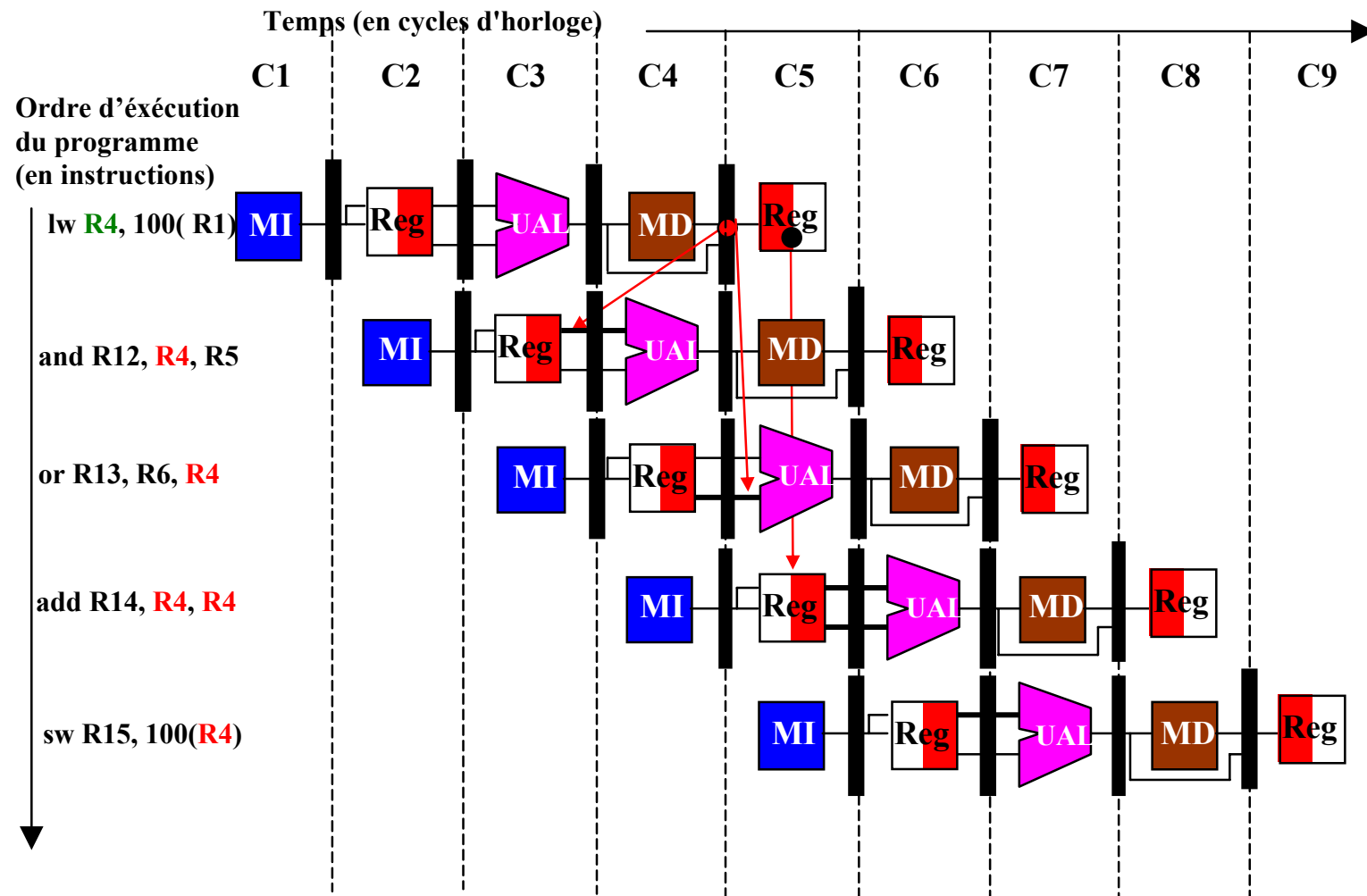
- placer l'Unité d'envoi dans l'étage DI afin d'anticiper le positionnement des signaux de contrôle ;
- positionner les valeurs de ces signaux dans de nouveaux champs de contrôle du registre DI/EX.

L'ENVOI POUR LES CHARGEMENTS ET LES RANGEMENTS (1/5)

Hélas, l'envoi ne fait pas l'affaire lorsqu'une instruction tente de lire un registre à la suite d'un chargement écrivant dans le même registre.

La donnée est toujours lue en mémoire au cycle 4 tandis que l'UAL effectue l'opération de l'instruction suivante.

Dans ce cas, on doit à nouveau suspendre le pipeline pour la combinaison d'instructions composée d'un chargement suivi d'une instruction lisant le résultat de celui-ci.



L'ENVOI POUR LES CHARGEMENTS ET LES RANGEMENTS (2/5)

Pour suspendre le pipeline pour le cas des instructions de référence mémoire, on va donc faire revivre l'unité de détection d'aléas.

- Elle opère pendant l'étage DI, et continue à fonctionner en présence de l'unité d'envoi.
- Elle recherche les instructions de chargement en testant si le signal de contrôle DI/EX.RegDst est à 0 et cop du chargement.

Le contrôle de l'Unité de détection d'aléas peut maintenant s'écrire avec la condition unique suivante :

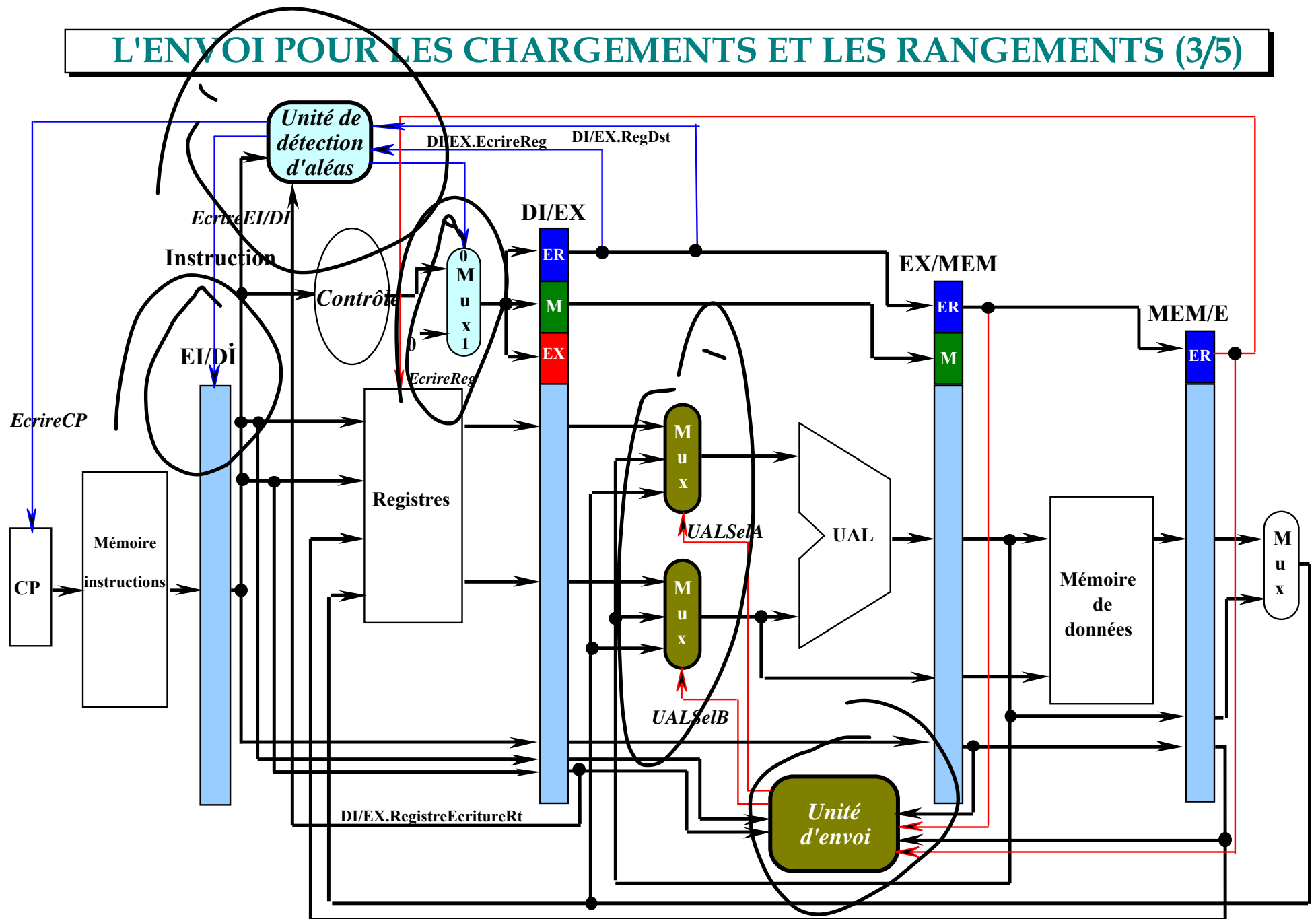
Si (DI/EX.EcrireReg et (DI/EX.RegDst = 0)
et (DI/EX.code-op=COP(lw)) et (LI/DI.code-op=COP(Format R))et
((DI/EX.RegistreEcritureRt = EI/DI.RegistreLecture1) ou
(DI/EX.RegistreEcritureRt = EI/DI.RegistreLecture2))

Alors *suspendre le pipeline*

Comme auparavant, l'unité d'envoi contrôle les multiplexeurs de l'UAL afin de remplacer la valeur issue d'un registre général par la valeur issue du registre pipeline approprié.

Autre solution : compromis entre compilateur et complexité du matériel, les premiers processeurs évitaient que le matériel suspende le pipeline en imposant au logiciel de trouver une instruction indépendante du chargement et de la placer après celui-ci. Au pire on plaçait une instruction *nop* après un chargement.

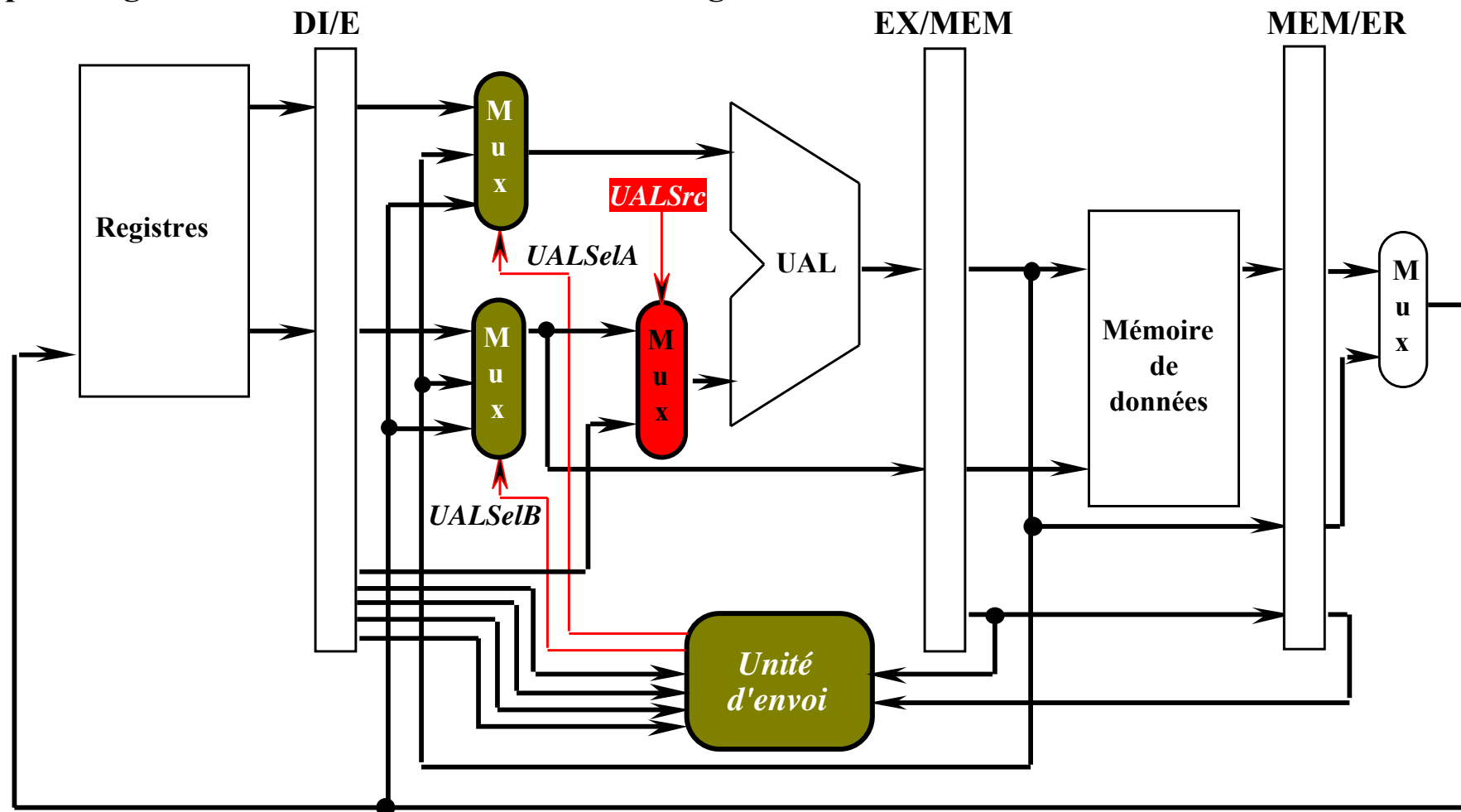
L'ENVOI POUR LES CHARGEMENTS ET LES RANGEMENTS (3/5)



L'ENVOI POUR LES RANGEMENTS ET LES RANGEMENTS (4/5)

Problème : Ajout de la valeur immédiate signée et étendue à l'entrée de l'UAL est nécessaire pour les instructions de référence mémoire. Le contrôle central **décide** entre **un registre** et **une valeur immédiate** alors que L'unité d'envoi **choisit un registre pipeline** dans le cas où l'entrée de l'UAL est **un registre** et qu'il y a **dépendance**.

Solution : Le plus simple est **d'ajouter un multiplexeur 2:1** qui choisit entre la sortie du multiplexeur d'envoi contrôlé par la ligne **UALSelB** et la valeur immédiate signée.

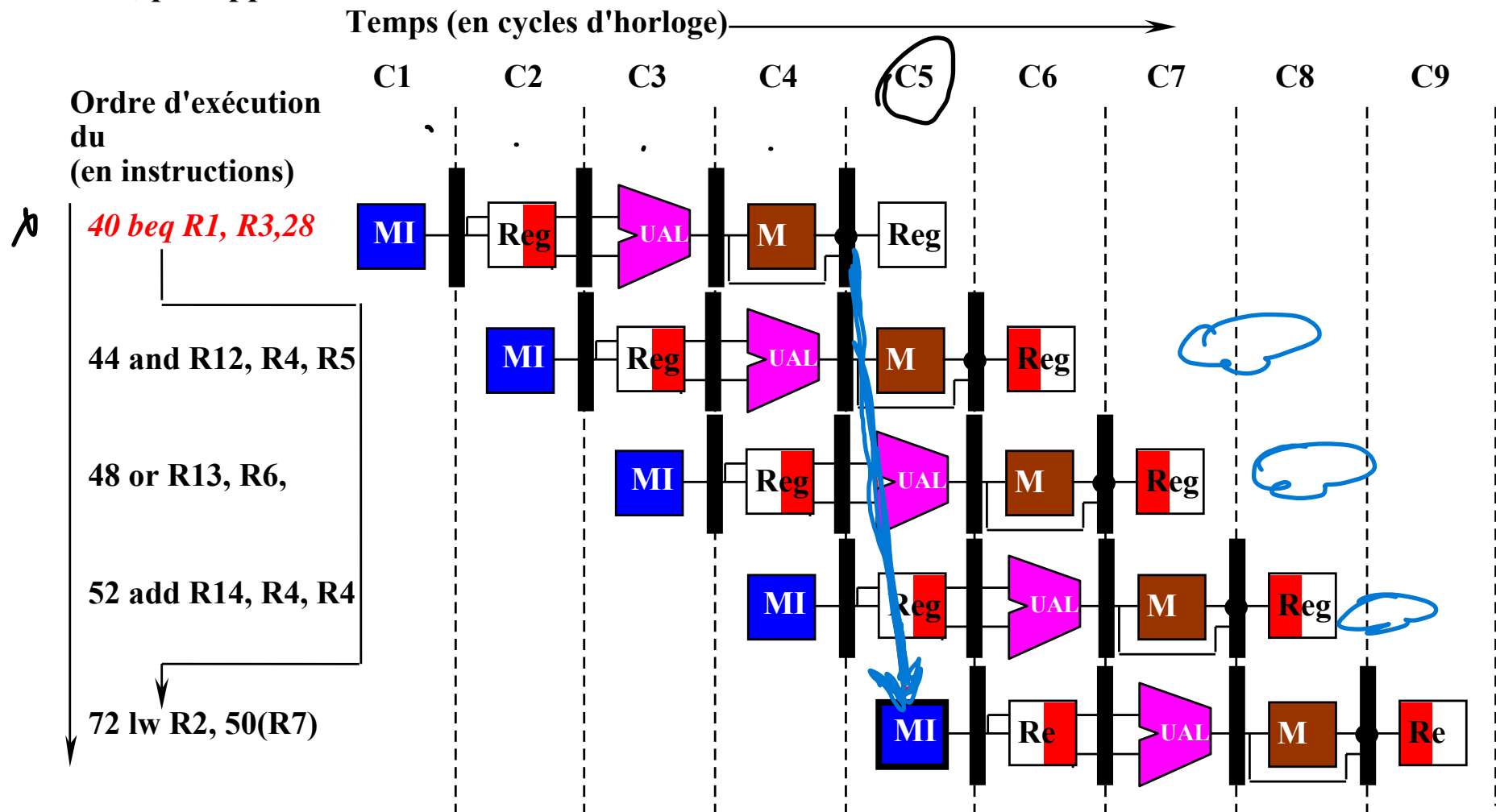


LES ALEAS DE BRANCHEMENT ou LES ALEAS DE CONTROLE

Une autre forme d'aléa de pipeline affecte les instructions de branchements.

Dans ce modèle de pipeline, la décision concernant un branchement éventuel ne survient qu'à l'étage MEM, alors que l'alimentation du pipeline nécessite l'arrivée d'une instruction à chaque cycle d'horloge.

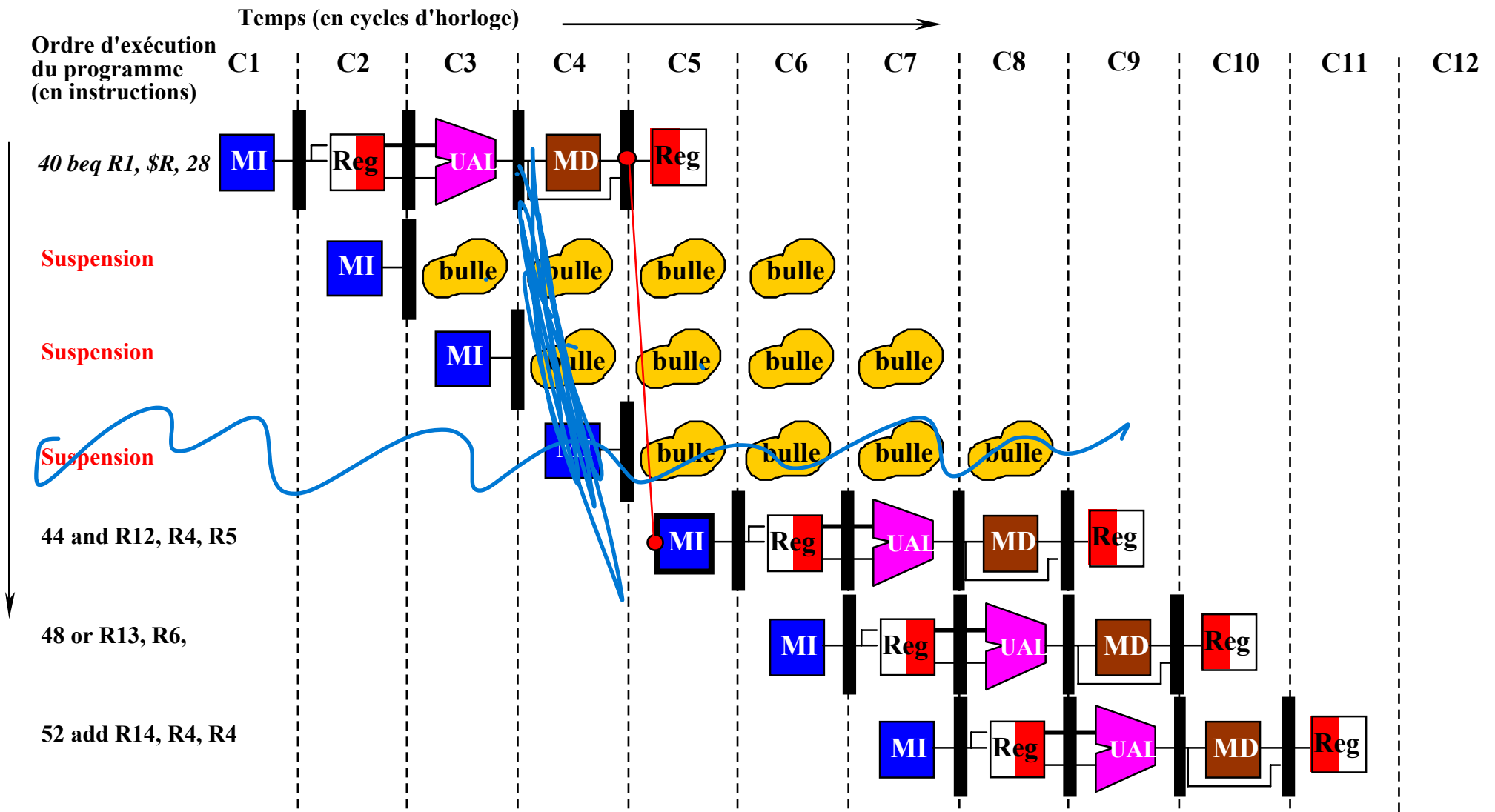
Ce délai pour déterminer quelle est l'instruction correcte à extraire est appelé un aléa de contrôle ou de branchement, par opposition aux aléas de données.



LES ALEAS DE BRANCHEMENT : TOUJOURS SUSPENDRE

Une solution consiste à suspendre le pipeline jusqu'à ce que le branchement soit achevé.

Cette solution subira une pénalité de plusieurs cycles d'horloge pour chaque branchement, alors qu'un branchement conditionnel décide assez souvent de ne pas effectuer le branchement.



LES ALEAS DE BRANCHEMENT : SUPPOSER BRANCHEMENT N'EST PAS PRIS

Prédire tout branchement *toujours pris* → Suspension systématique **!!! Solution trop pénalisante**

Amélioration : faire une prédiction inverse → Branchement *toujours non pris*.

Si mauvaise prédiction → Abandonner les instructions qui suivent le branchement
→ Poursuivre à partir de l'adresse de destination du branchement

Cette optimisation diminue environ de moitié le coût des aléas de contrôle

Il est plus coûteux de suspendre systématiquement que d'abandonner des instructions en cas de mauvaise prédiction

Abandonner des instructions → revient à remplacer simplement les valeurs actives des signaux contrôle par des valeurs inactives, comme pour la méthode des suspensions.

La différence est qu'on doit être capable en plus de vider les étages EI, DI, et EX. Pour les suspensions, on remplaçait simplement le contrôle par un 0 à l'étage DI et on le laissait se propager la bulle jusqu'à ER.

Vider l'étage EI : ajouter une ligne de contrôle, appelée **EI.Vider**, qui permet **la mise à zéro le registre EI/DI** afin de vider l'instruction extraite.

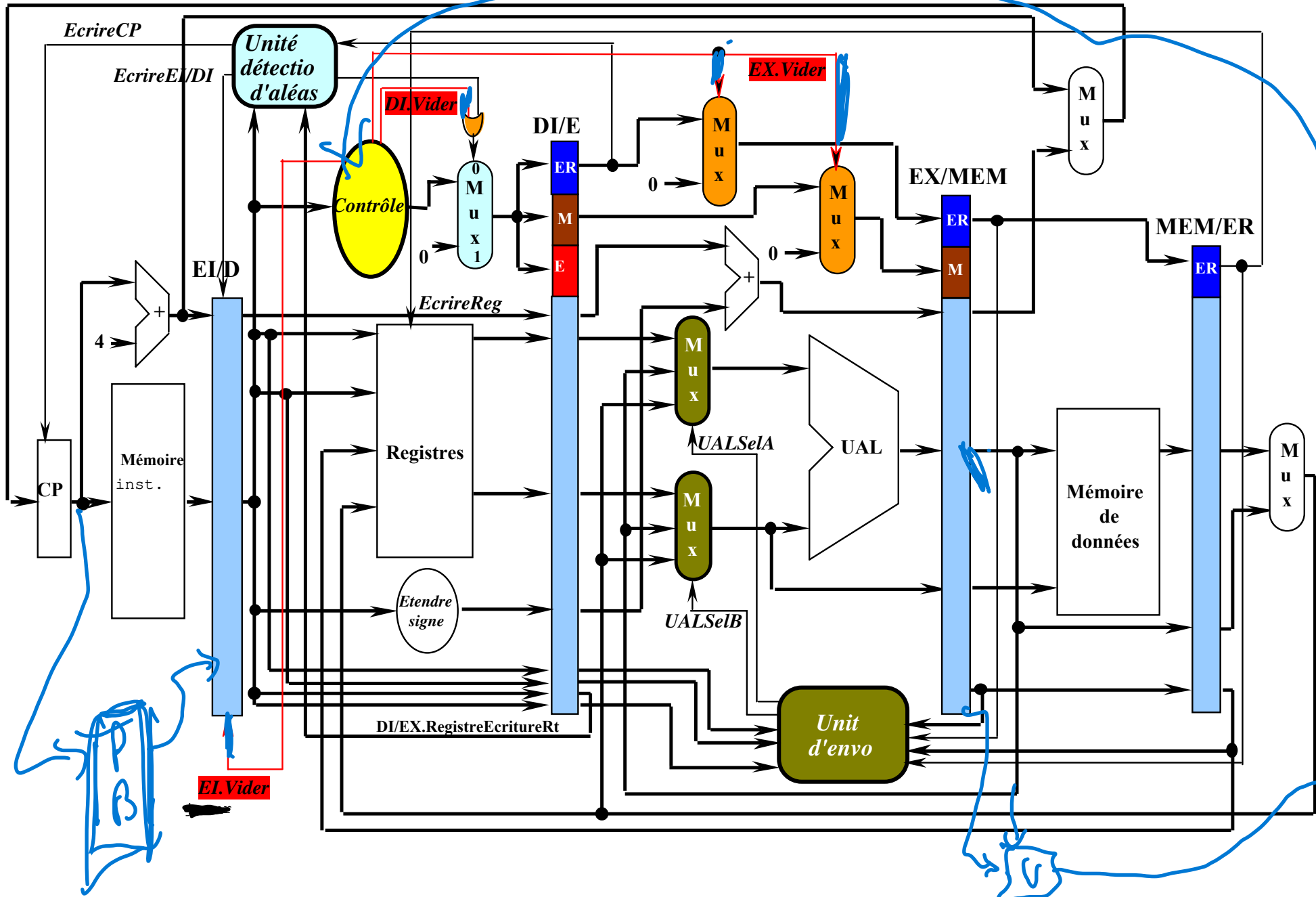
Vider l'étage DI : utiliser le MUX de l'UDA en place à l'étage DI pour effectuer les suspensions et ajouter un nouveau signal de contrôle, appelé **DI.Vider**, permettant de contrôler aussi le MUX de l'UDA.

Le contrôle du MUX de l'UDA est soumis alors au OU du signal **DI.vider** et du signal de suspension issu de l'UDA.

Vider l'étage EX : ajouter un signal appelé **EX.Vider** qui impose à 2 nouveaux MUX situés à l'étage EX, de mettre à zéro les lignes de contrôle.

Le contrôle décide d'envoyer un signal de vidage en fonction du code-op de l'instruction et de la valeur de la condition de branchement.

LES ALEAS DE CONTROLE : PREDICTION D'UN BRANCHEMENT NON PRIS



LES EXCEPTIONS (1/2)

Une autre forme d'aléa de contrôle concerne les exceptions.

Par exemple, supposons que l'instruction : ***add R1, R2, R1*** provoque un débordement arithmétique.

- Le contrôle doit être transféré à la routine d'exception immédiatement après cette instruction → ***ne pas contaminer les registres ou la mémoire ;***
- Comme pour les branchements pris : vider du pipeline les instructions qui suivent l'instruction ***add*** et ***extraire les instructions à partir de l'adresse d'exception***. L'adresse d'exception du R3000 se trouve à $(4000\ 0040)_{16}$.
- Même mécanisme que pour les branchements pris, mais cette fois c'est l'exception qui demandera à ce que les lignes de contrôle soient désactivées.
- Pour extraire les instructions à partir de l'adresse $(4000\ 0040)_{16}$, on ajoute une entrée au MUX du CP qui aiguille l'adresse $(4000\ 0040)_{16}$ au CP.
- Cet exemple souligne un problème qui peut survenir avec les exceptions si on ne stoppe pas l'exécution :
 - ***R1 sera contaminé par une valeur invalide.***
 - ***La valeur originelle de R1 qui a contribué à provoquer le débordement sera détruite.***
- L'exception de débordement est détectée à l'étage EX, et on peut donc utiliser le signal ***EX.Vider*** pour empêcher l'instruction d'écrire son résultat à l'étage ER.
- La dernière étape consiste à stocker l'adresse de l'instruction incriminée dans le Compteur d'Exceptions de Programme (CEP).

LES EXCEPTIONS (2/2)

