

ARCHITECTURE HAUTES PERFORMANCES SE RAPPROCHER D'UN CPI DE 1

Pascal SAINRAT

M1 SIAME – 2018/2019
EMINS2G1 : AHP

Objectifs de l'UE

- Vous faire comprendre l'évolution de l'architecture des processeurs
 - Passée mais aussi future
- Que vous soyez capable de comparer les caractéristiques de différents processeurs
- Que vous soyez capable d'évaluer la performance d'un processeur



Evaluation

- Contrôle terminal



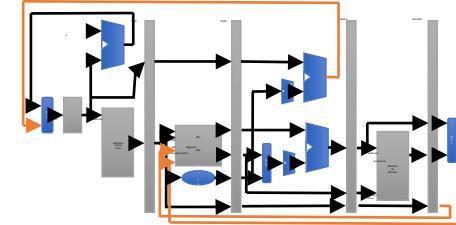
Abstraction

- **Architecture** : vue pour le programmeur
- **Microarchitecture** : Comment c'est fait matériellement
- Il faut connaître les **fonctions logiques combinatoires et séquentielles**
 - Qui sont une abstraction d'un assemblage de **portes logiques**
 - Qui sont une abstraction d'un assemblage de **transistors**
 - Dont le fonctionnement est fondé sur des lois **physiques**

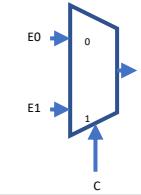
Architecture

LDR R3, [R6]
ADD R4, R3, R2
ADD R2, R2, #1

Microarchitecture



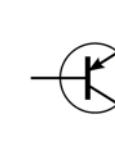
Fonctions logiques



Portes logiques



Transistors



Physique



Abstraction

Architecture ARM

- Cf. ressources sur Moodle, module archi3 de L2 info.
 - Architecture load/store : LDR, STR
 - ADD, SUB, AND, ...
 - Bcc
- L'architecture est définie par le jeu d'instructions et l'état architectural (registres généraux, registre d'état, mémoire d'instructions, mémoire de données)

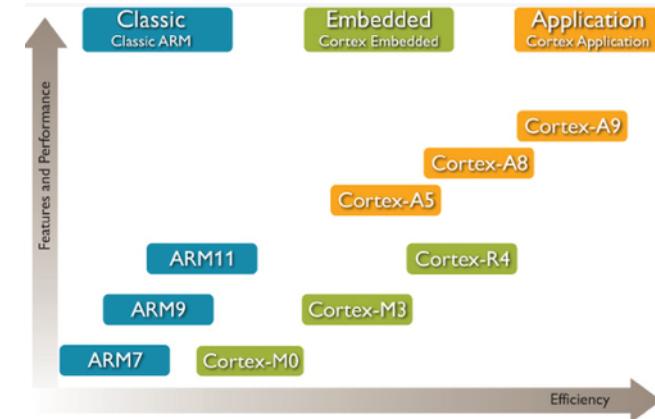


Microarchitecture

- Connexion entre la logique (combinatoire et séquentielle) et l'architecture
- Agencement des registres, ALUs, mémoires et autre logique
- Plusieurs microarchitectures sont possibles pour une architecture
 - Compromis performance, énergie, coût, ...

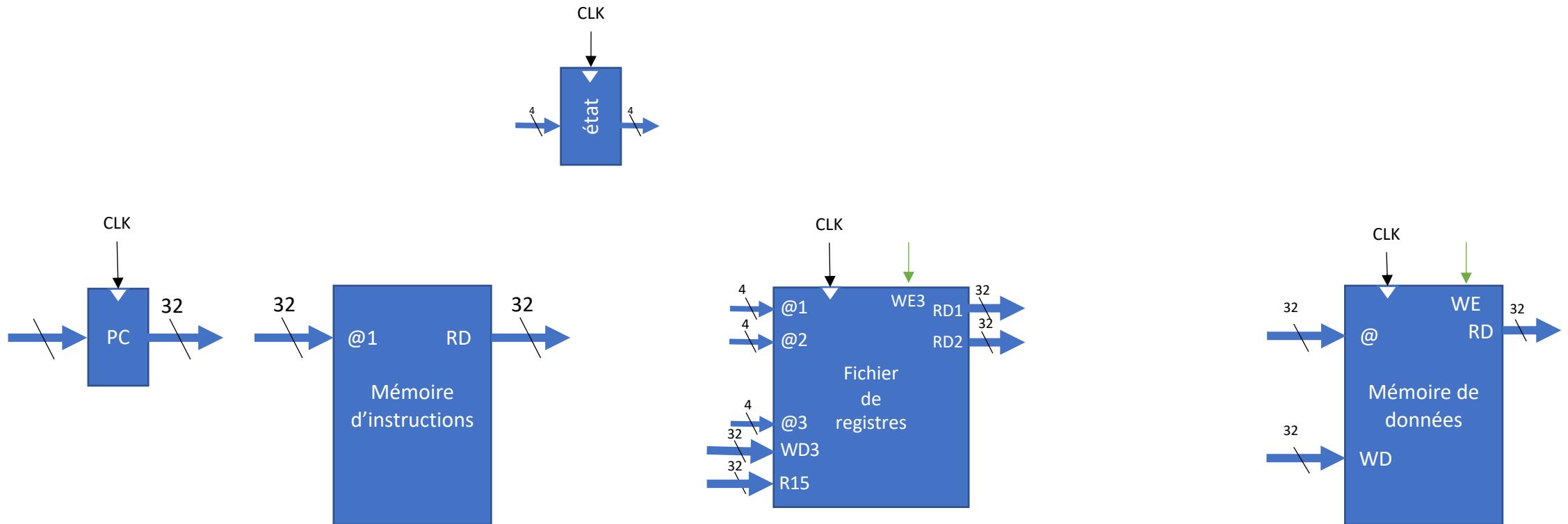


- Chemins de données
 - 32 bits pour l'ARM
 - Contient les mémoires, les registres, les ALUs, des multiplexeurs
- Unité de contrôle
 - Contrôle le chemin de données (sens des multiplexeurs, signaux d'écriture des mémoires, contrôle de l'ALU, ...)



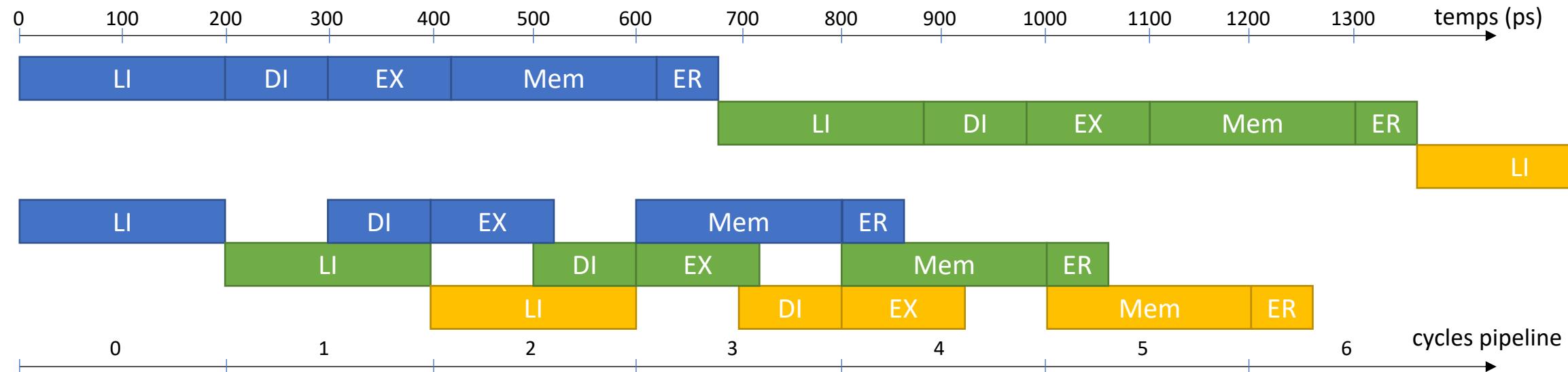
Mémorisation de l'état de la microarchitecture

- Signal noir : l'horloge de cadencement (CLK)
- Signaux **verts** : signaux de contrôle
- Signaux **bleus** : chemin de données



Processeur pipeline

- Le traitement d'une instruction est divisé en cinq étages. 5 instructions peuvent être en cours de traitement, chacune dans un étage.
 - Chargement de l'instruction depuis la mémoire (LI)
 - Lecture des opérandes source (dans le banc de registre ou dans l'instruction elle-même pour les valeurs immédiates) et décodage pour produire les signaux de contrôle (DI)
 - Calcul dans l'ALU (EX)
 - Lecture ou écriture de la mémoire de données (Mem)
 - Écriture du résultat dans le banc de registres (ER)



Exercice pipeline idéal et temps de cycle (1)

- On suppose que les étages du chemin de données ont les durées suivantes :

LI	DI	EX	Mem	ER
250 ps	350 ps	150 ps	300 ps	200 ps

- On suppose que les instructions exécutées par le processeur sont décomposées comme suit:

ALU	Branch	LDR	STR
45%	20%	20%	15%

- Quel est le temps de cycle d'un processeur pipeliné et d'un processeur non pipeline ? Quelles sont les fréquences respectives ?
- Quelle est la latence totale d'une instruction LDR sur un processeur pipeline et un processeur non pipeline ?
- Supposons que l'on peut couper un étage du pipeline en 2. Lequel couper ? Nouveau temps de cycle du processeur pipeline ? Nouvelle fréquence ?
- Quel est le taux d'utilisation de la mémoire de données ?
 - En supposant que les accès à la mémoire de données durent un cycle
- Quel est le taux d'utilisation du registre en écriture du banc de registres ?

Exercice rendement pipeline idéal

- Nombre de cycles minimum pour exécuter n instructions sur un pipeline de k étages ?

Exécution de LDR R2, [R0, #40] (1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
cond				0	1	0	P	U	B	W	L		Rn		Rd																	addr_mode
1110				0	1	0	1	1	0	0	1		0000		0010															0000 0100 0000		

Always

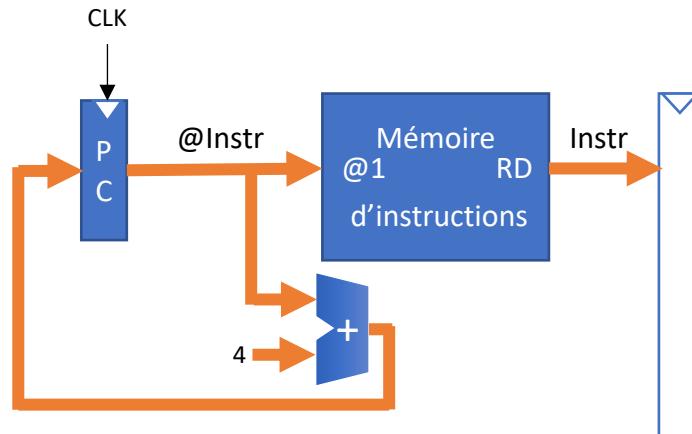
LDR indexé immédiat

0

2

40

Chargement de l'instruction



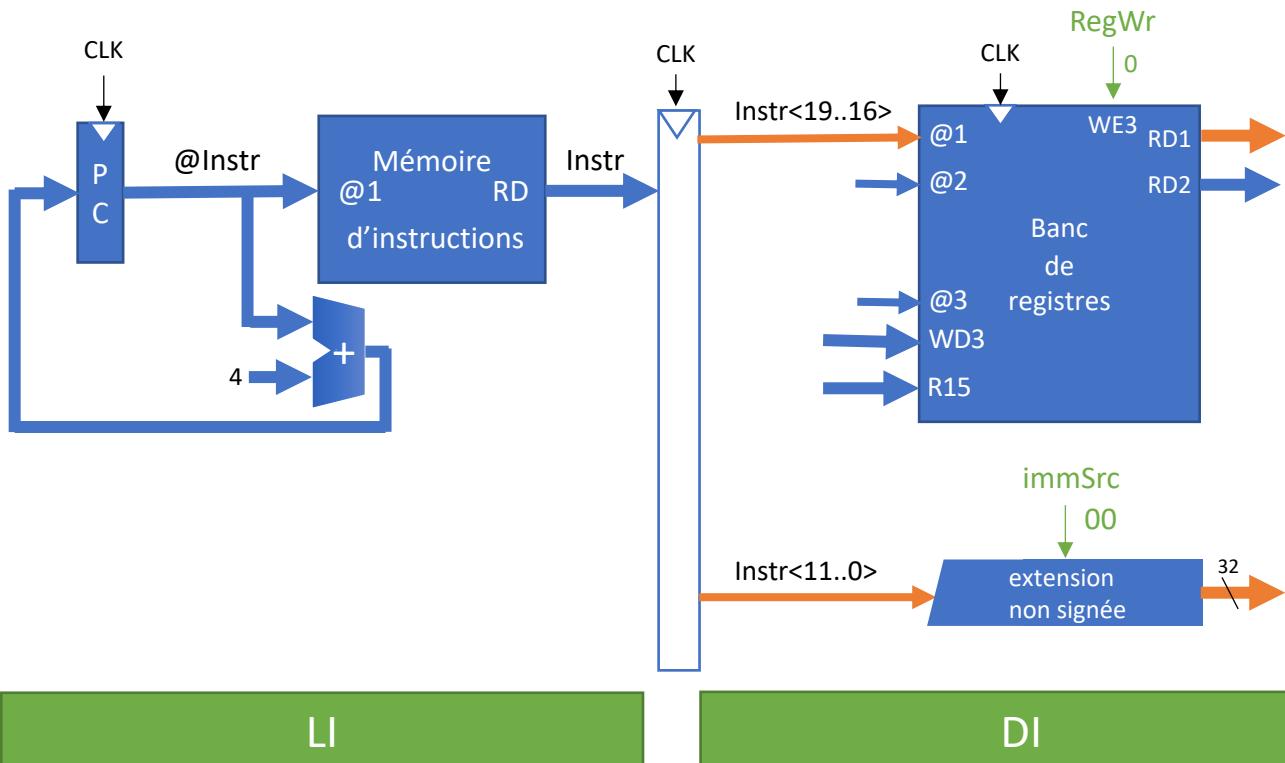
LI

Exécution de LDR R2, [R0, #40] (1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1110		0	1	0	1	1	0	0	0	1		0000		0000	0010							0000	0100	0000							

Always LDR indexé immédiat 0 2 40

Lecture registre d'adresse
Extension valeur immédiate



Exécution de LDR R2, [R0, #40] (2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1110				0	1	0	1	1	0	0	1		0000			0010					0000	0100	0000								

Always

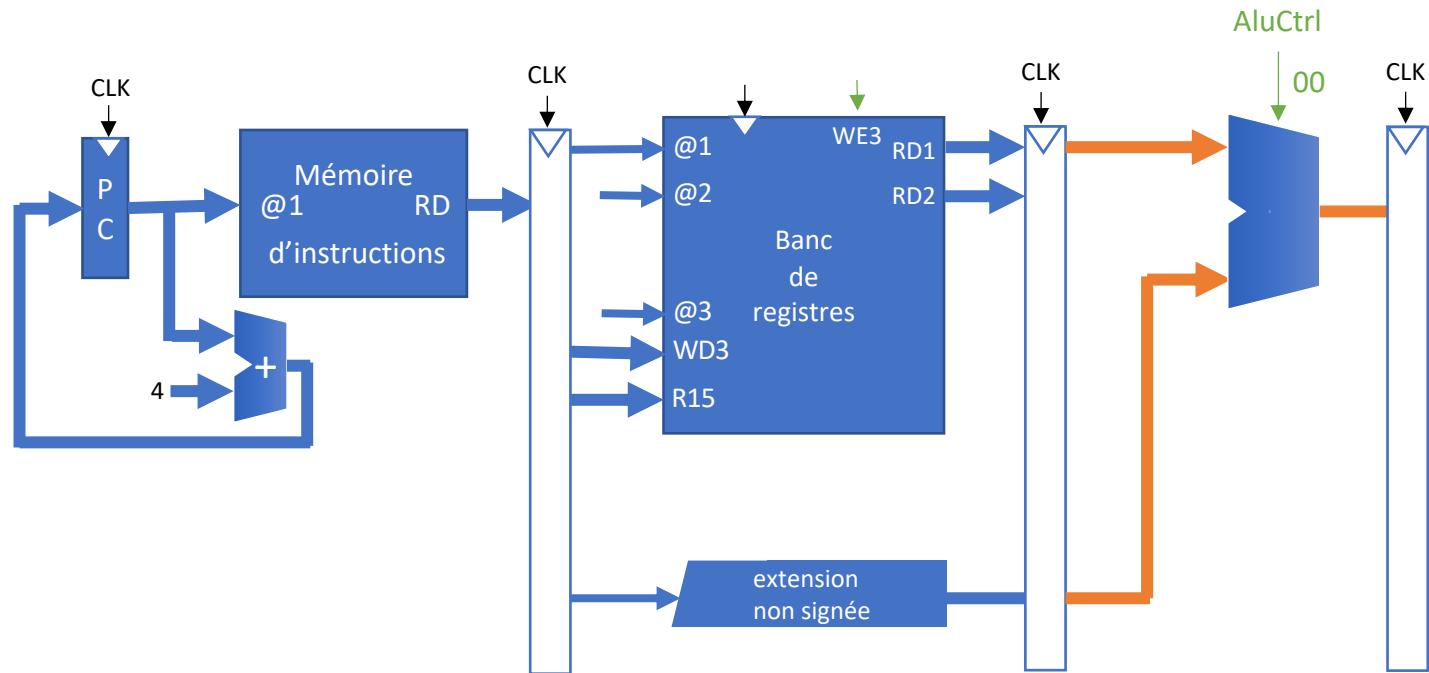
LDR indexé immédiat

0

2

40

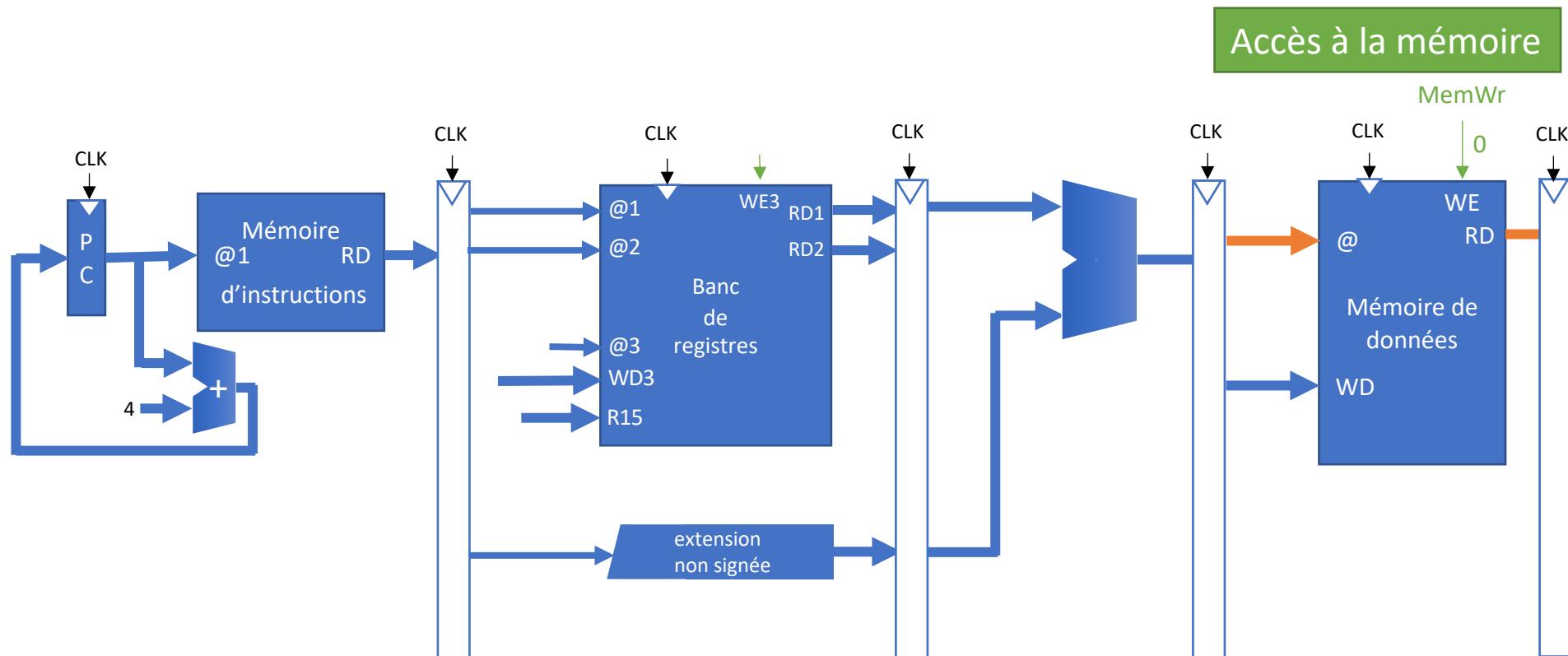
Calcul de l'adresse



Exécution de LDR R2, [R0, #40] (3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1110				0	1	0	1	1	0	0	1	0000		0010									0000	0100	0000						

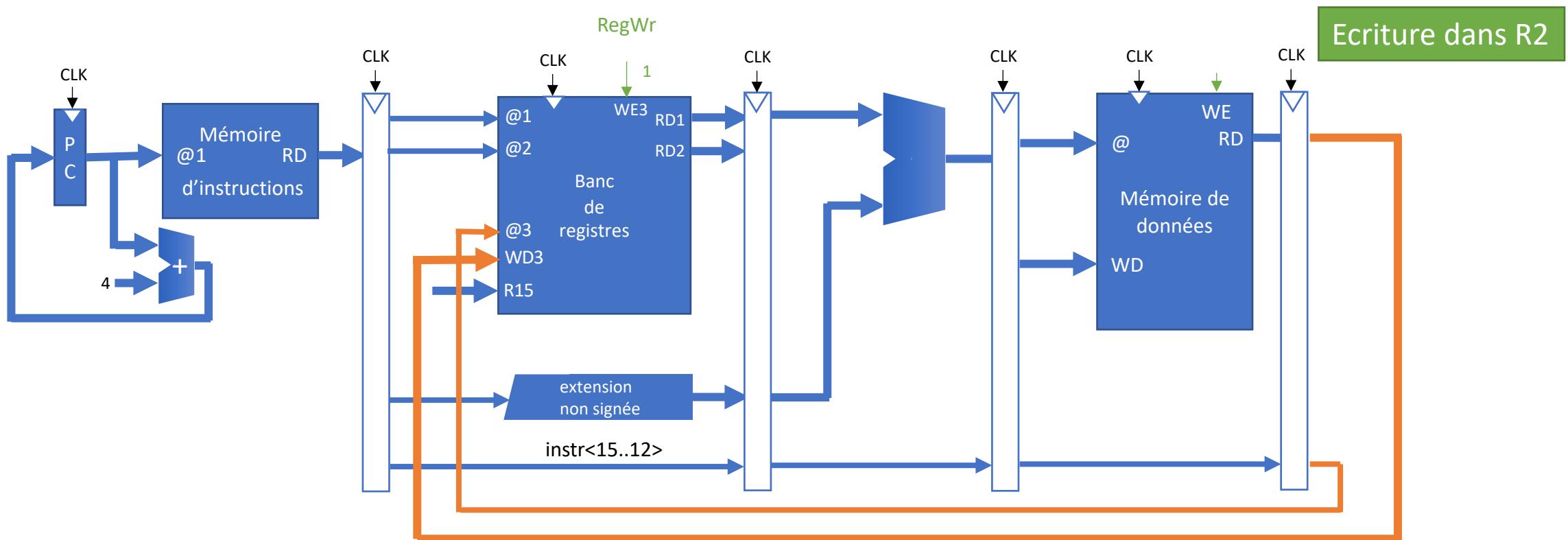
Always LDR indexé immédiat 0 2 40



Exécution de LDR R2, [R0, #40] (3)

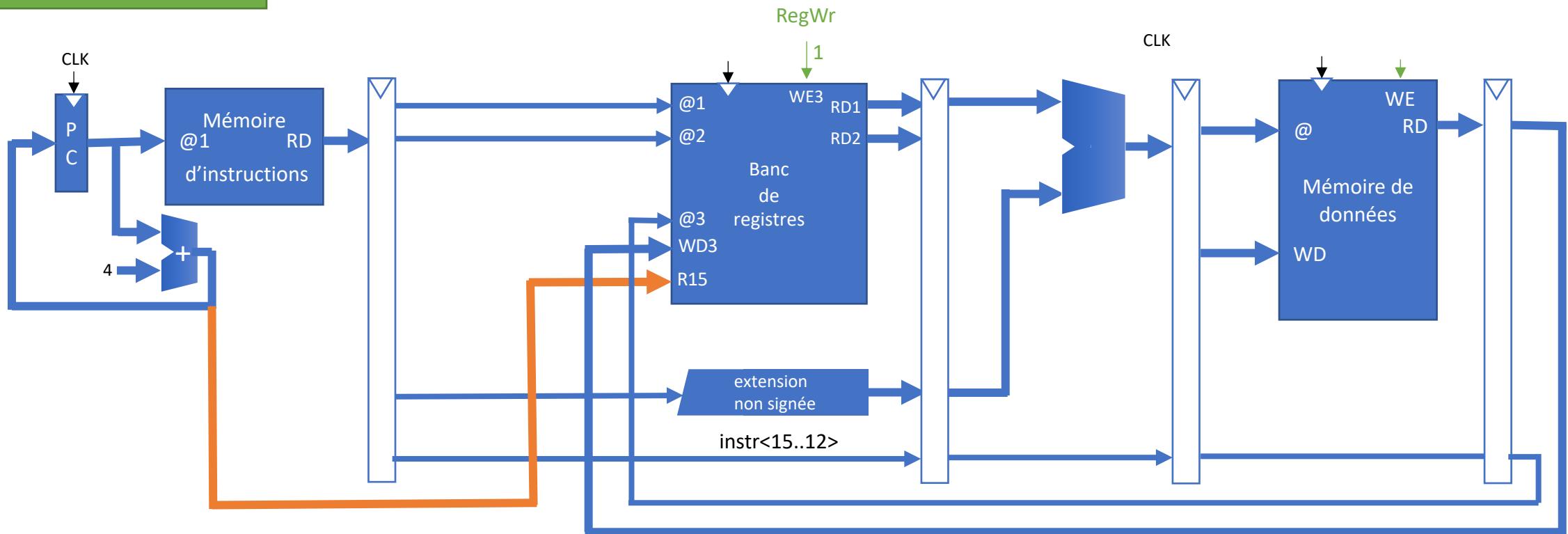
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1110				0	1	0	1	1	0	0	1	0000			0010							0000	0100	0000							

Always LDR indexé immédiat 0 2 40

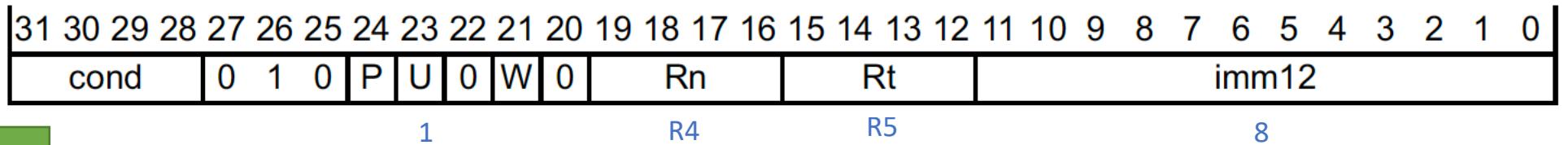


Exécution de LDR R2, [R0, #40] (5)

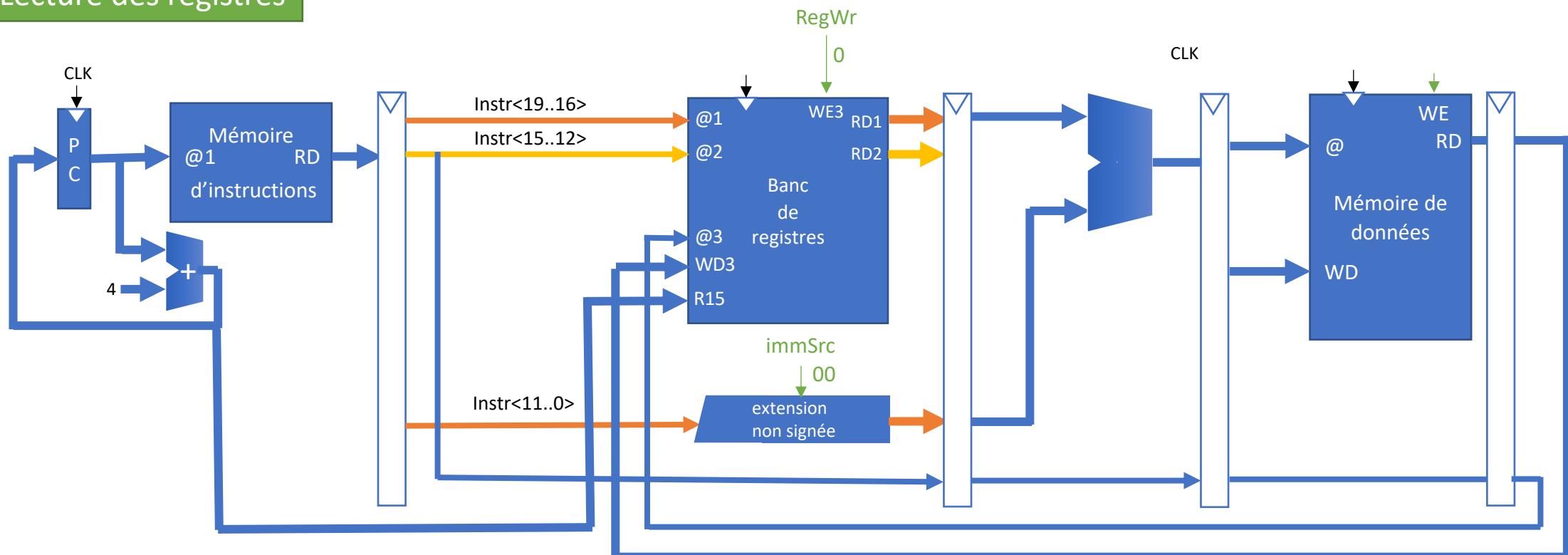
Ecriture de R15 (2)



Exécution d'un STR (1) – Ex : STR R5, [R4, #8]

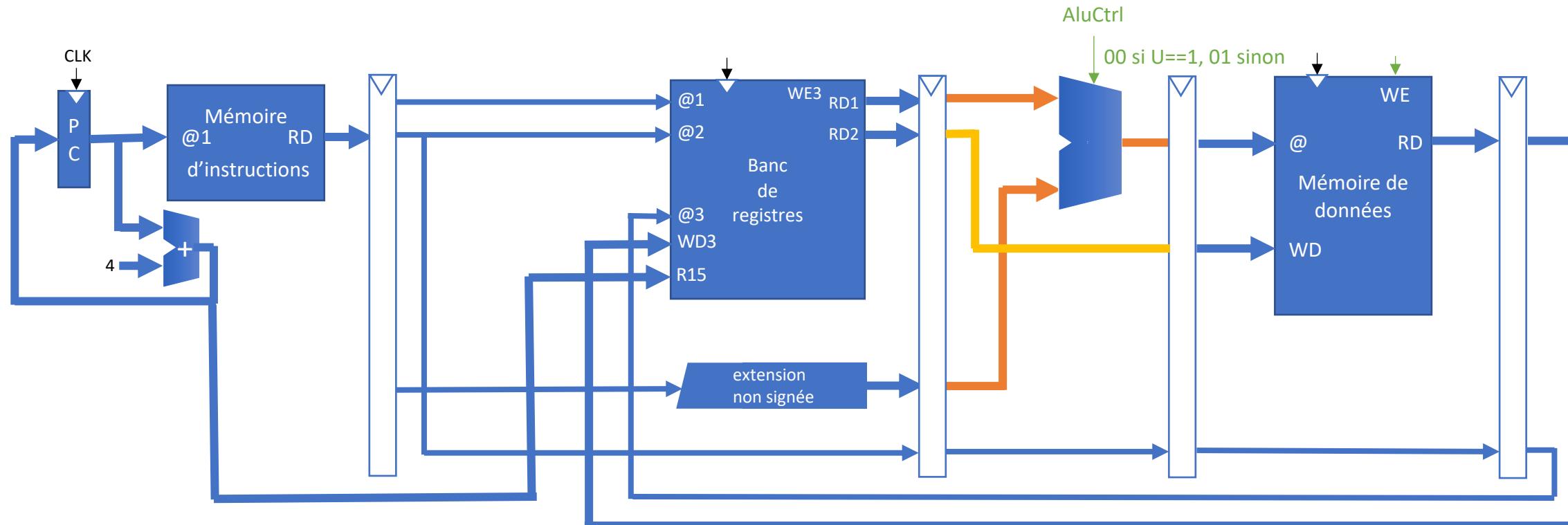


Lecture des registres

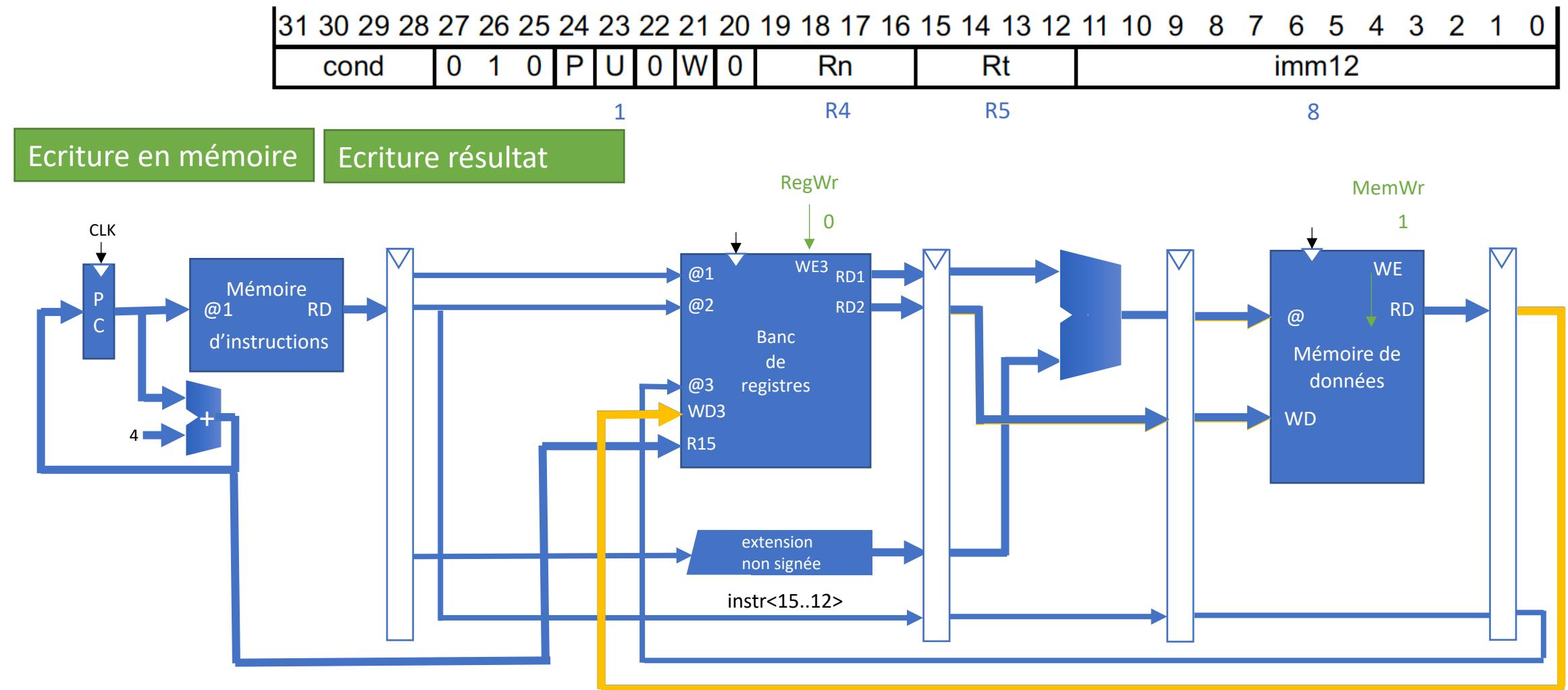


Exécution d'un STR (2) – Ex : STR R5, [R4, #8]

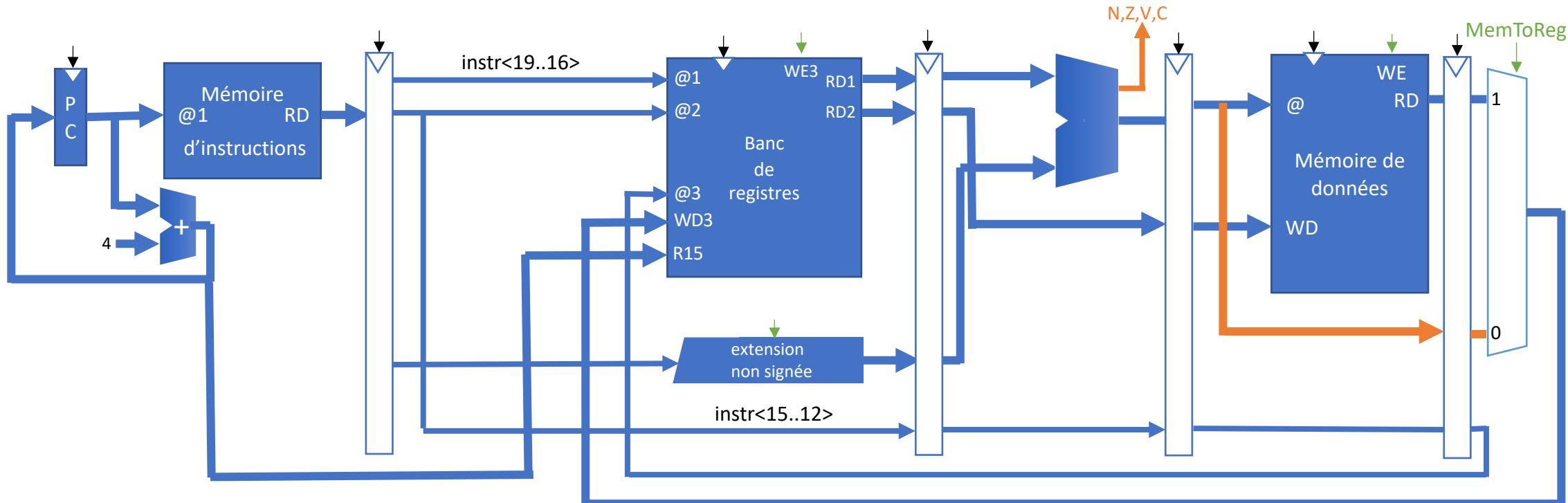
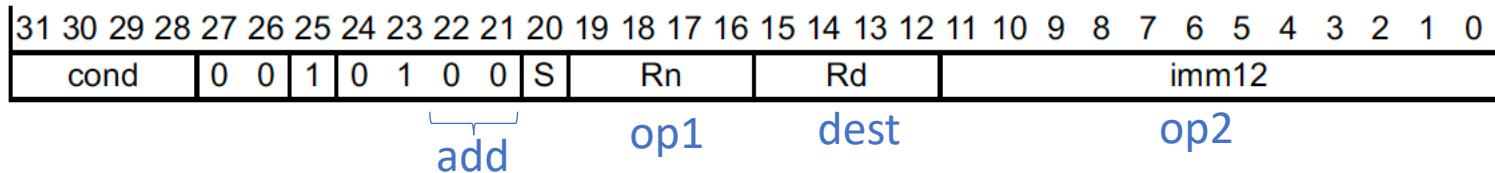
Calcul	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
	cond	0	1	0	P	U	0	W	0	Rn	Rt																								
	1									R4	R5																								8



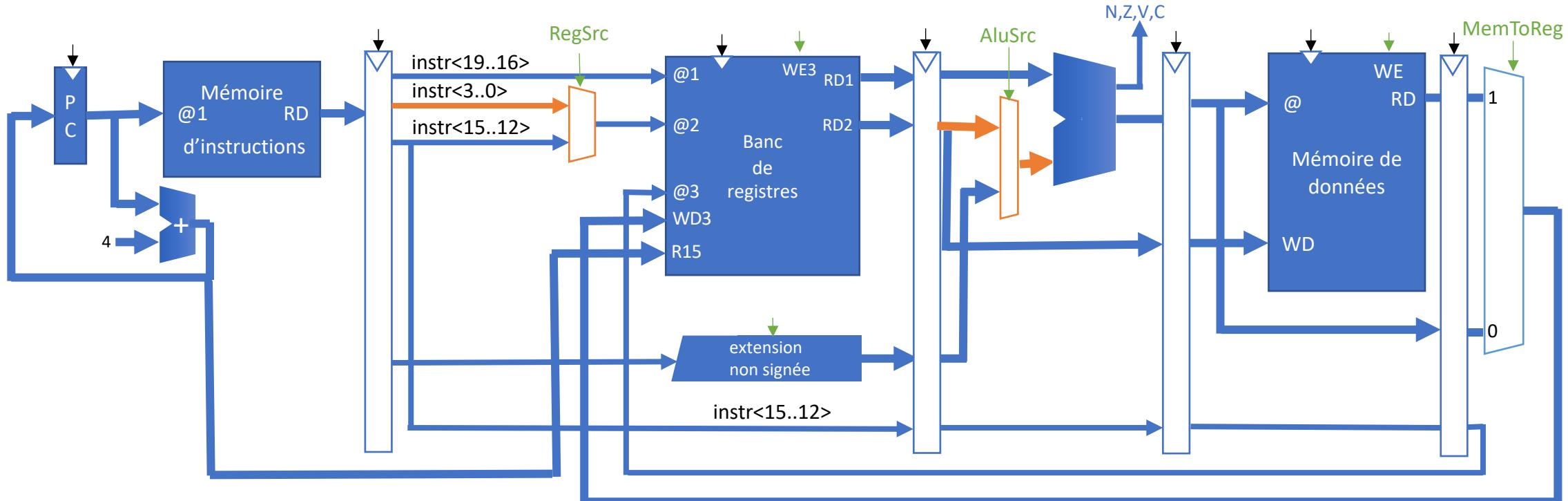
Exécution d'un STR (3) – Ex : STR R5, [R4, #8]



Instruction de calcul registre/immédiat



Instruction de calcul registre/registre

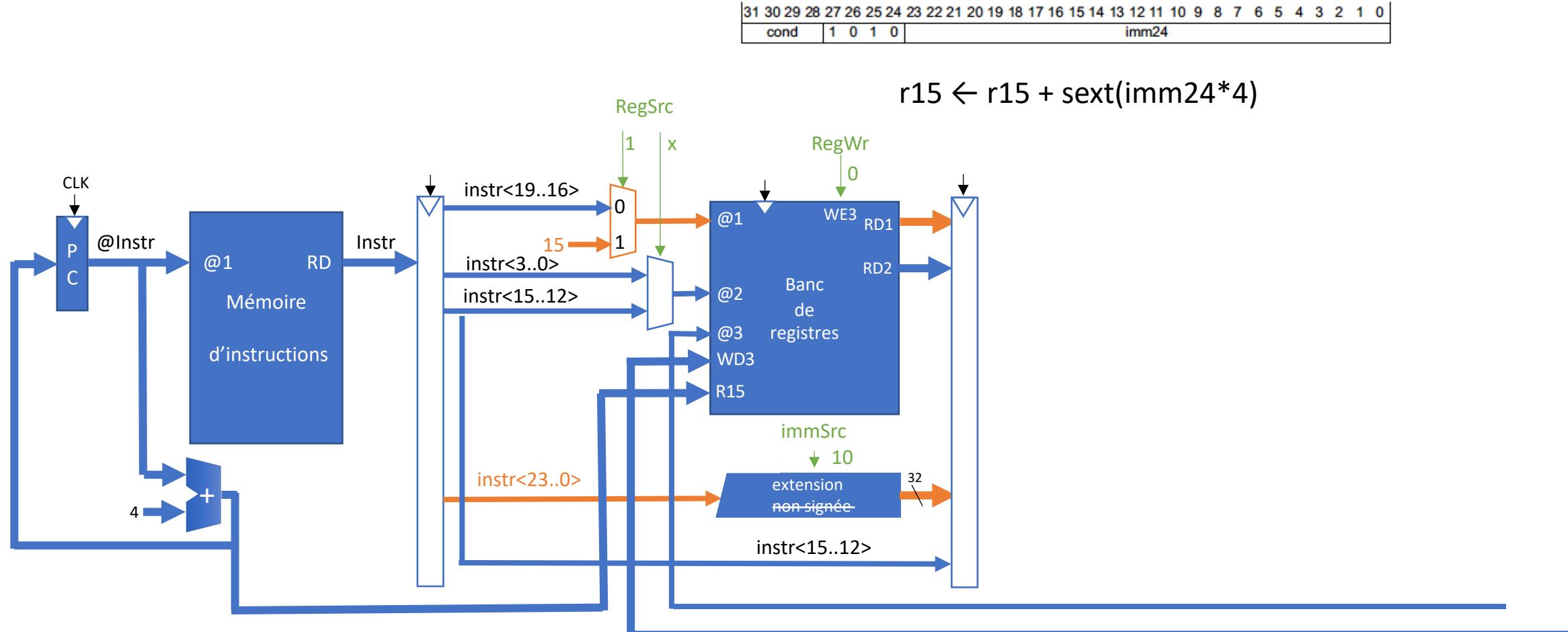


Instruction de branchement (étage LI)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond	1	0	1	0																										imm24	

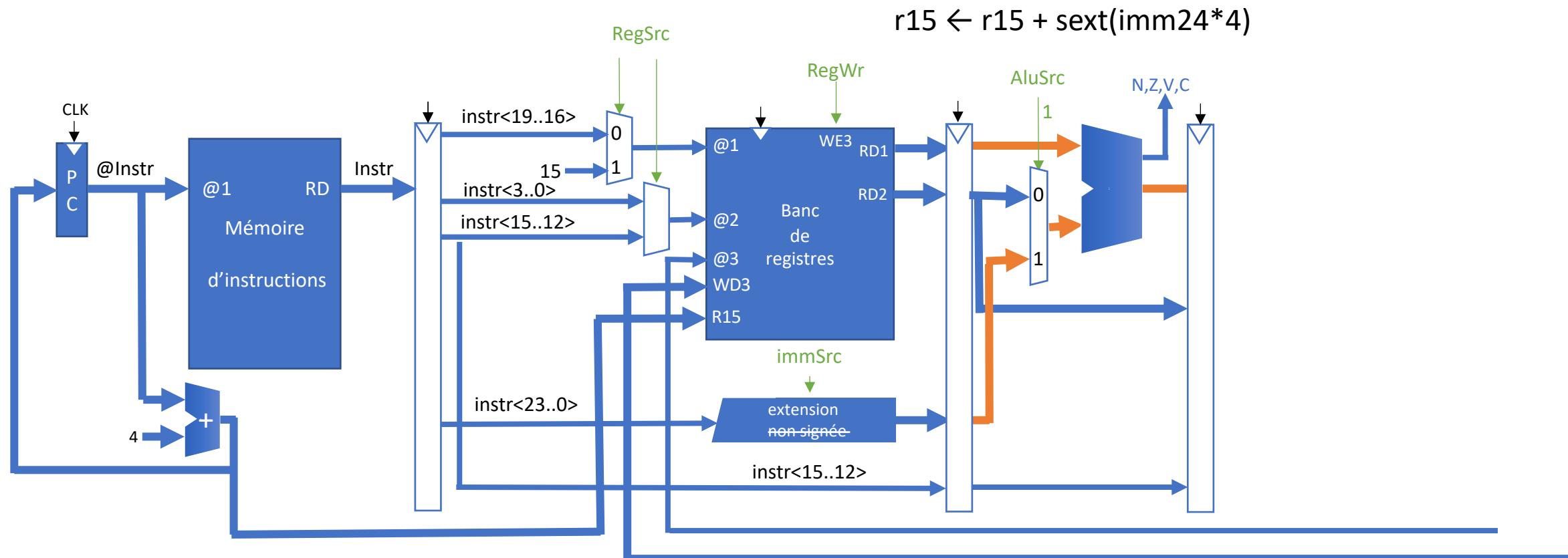


Instruction de branchement (étage DI)

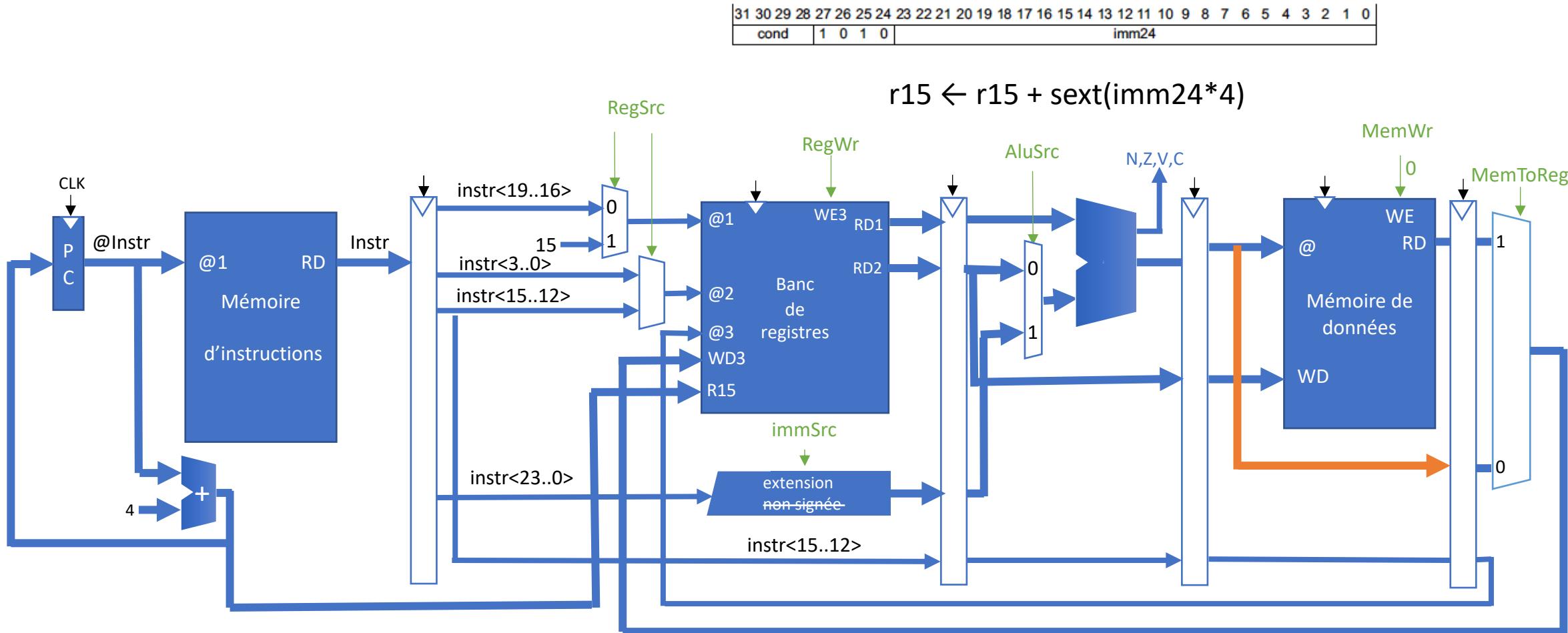


Instruction de branchement (étage EX)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond	1	0	1	0																										imm24	

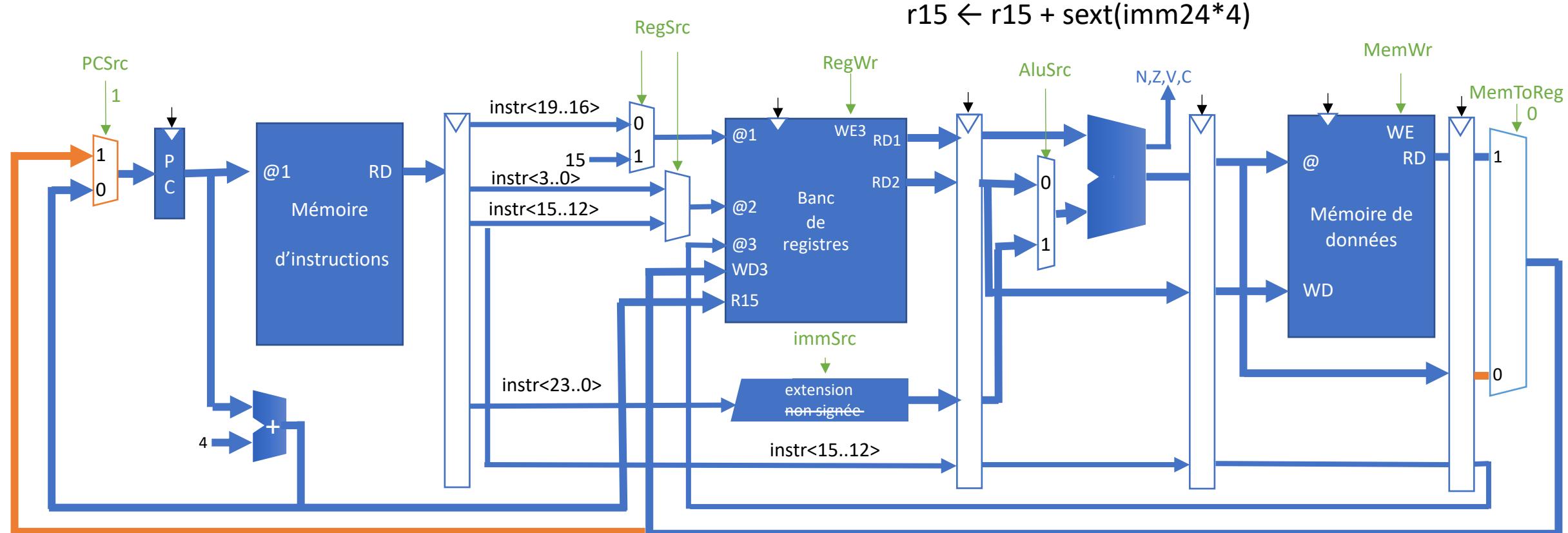


Instruction de branchement (étage Mem)



Instruction de branchement (étage ER)

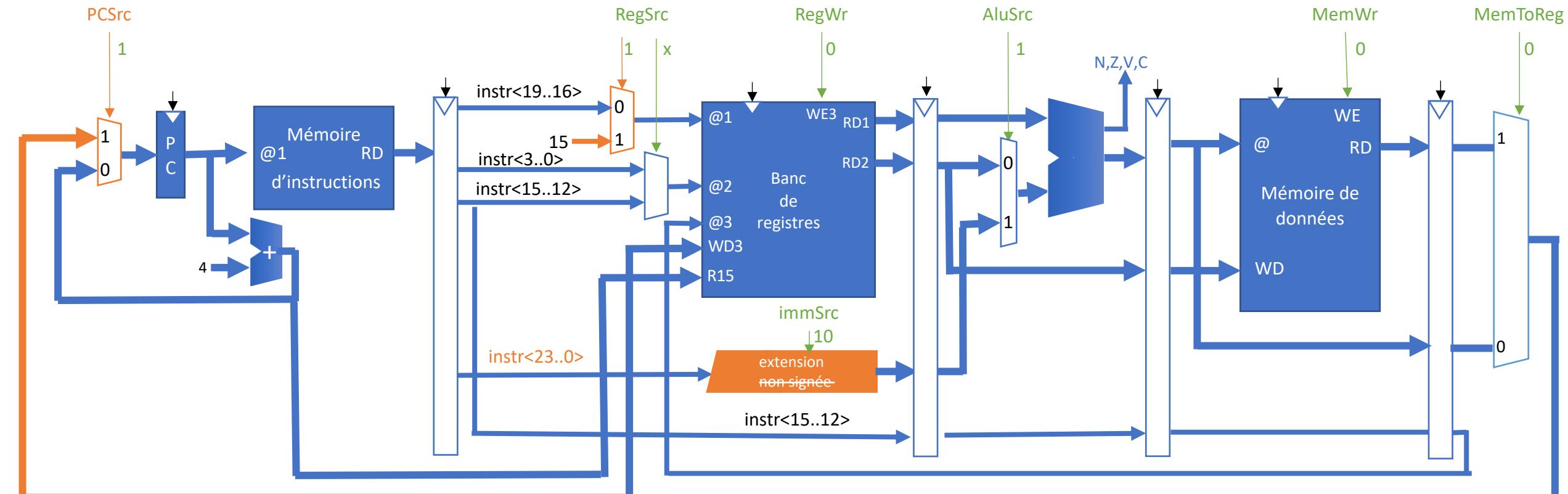
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond	1	0	1	0																									imm24		



Instruction de branchement (résumé des ajouts)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
cond	1	0	1	0																													imm24

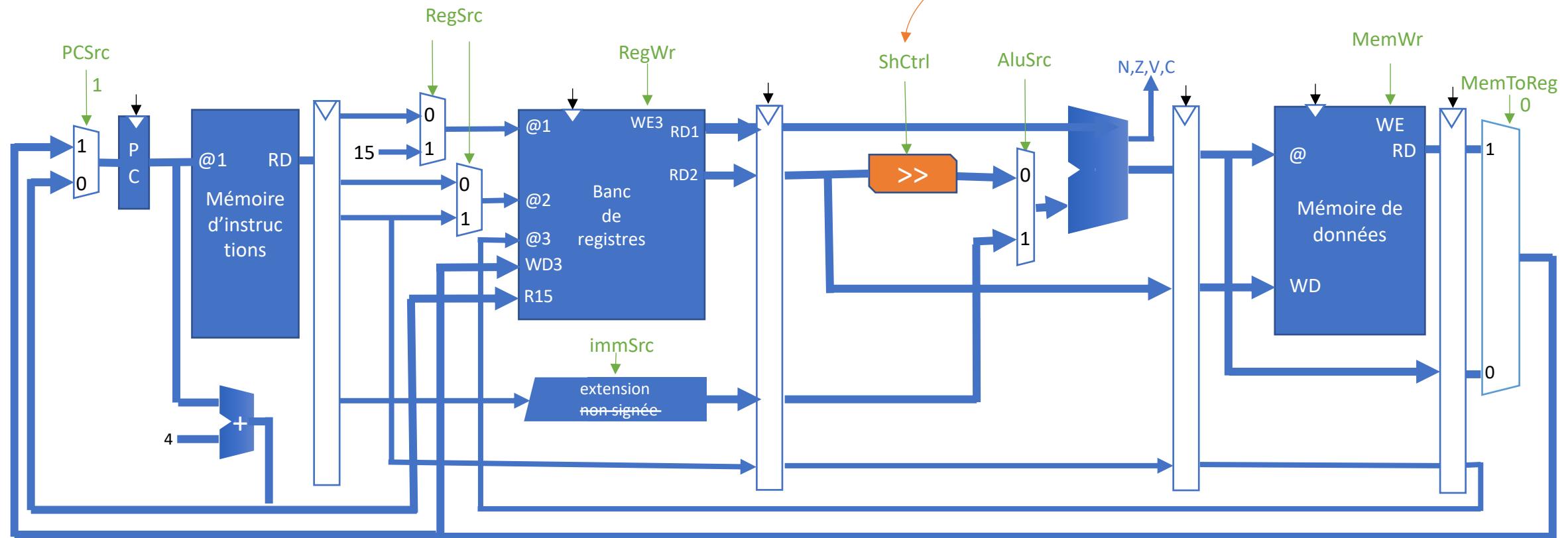
$$r15 \leftarrow r15 + \text{sext}(\text{imm24} * 4)$$



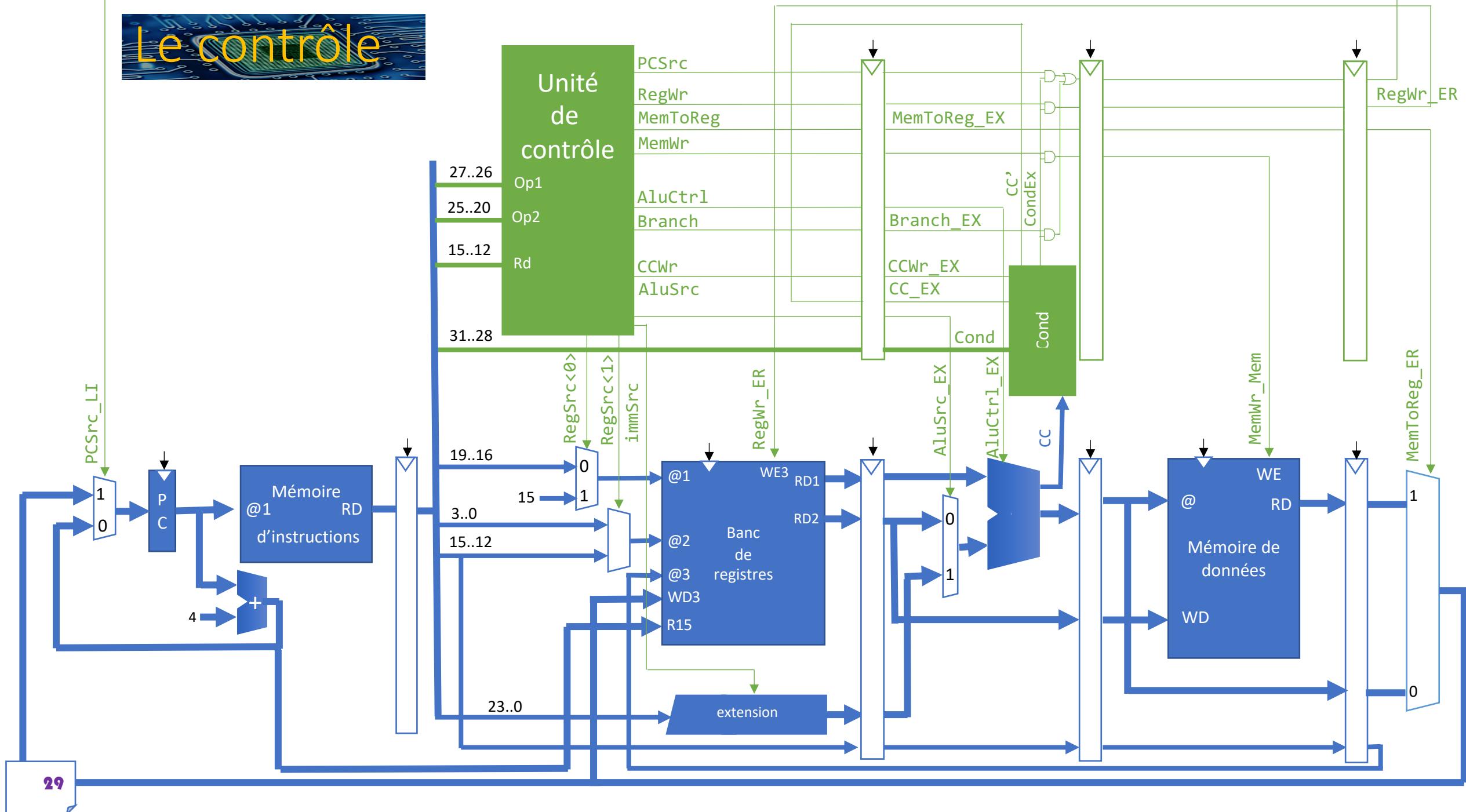
Ajout d'un mode d'adressage : ADD R3, R2, R1, LSL #2

$R3 \leftarrow R2 + (R1 \ll 2)$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
cond	0	0	0	0	1	0	0	S	Rn	Rd	imm5	type	0	Rm																		



Le contrôle



Génération du signal AluCtrl

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 0 0 0 1 0 0 S Rn Rd Rm

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 0 1 0 1 0 0 S Rn Rd imm12

Le bit S indique si les flags (N, Z, V, C) doivent être mémorisés.

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 0 1 1 0 1 0 Rn (0)(0)(0)(0) imm12

Calcul reg/reg

Calcul reg/imm

CMP

STR

LDR

Branchement

Les bits 26 et 27 indiquent que c'est une opération de calcul.
Le bit 25 indique le mode d'adressage (reg+reg ou reg+imm).
Les bits 21 à 24 indique l'opération que doit faire l'ALU :
0100: ADD – 0010 : SUB – 0000 : AND – 1100 : ORR

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 1 0 P U 0 W 0 Rn Rt imm12

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 1 0 P U 0 W 0 Rn Rt imm12

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 1 0 1 0 imm24

sauf si c'est un
branchement.

Instr<27..26>	Instr<24..21>	AluCtrl<1..0>
00 (calcul)	10 (Branch)	xxxx
	0100	00 (+)
	0010	01 (-)
	0000	10 (Et)
	1100	11 (Ou)
	1010	01(-)

LDR ou STR		
Instr<27..26>	Instr<23>	AluCtrl<1..0>
01	0	00 (+)
01	1	01 (-)

Génération des autres signaux (1)

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 0 0 0 1 0 0 S Rn Rd _____ Rm

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 0 1 0 1 0 0 S Rn Rd _____ imm12

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 0 0 1 0 1 0 Rn (0)(0)(0)(0) imm5 type 0 Rm

AluSrc

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 1 0 P U 0 W 0 Rn Rt imm12

MemWr

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 0 1 0 P U 0 W 1 Rn Rt imm12

MemToReg

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond 1 0 1 0 imm24

Branch, RegSrc<0>

Calcul reg/reg

Rn+Rm → RegSrc<1>=0, RegSrc<0>=0, RegWr=1

Calcul reg/imm

Rn+Rm → RegSrc<1>=0, RegSrc<0>=0, RegWr=1
ImmSrc<1..0>=00

CMP

Rn-Rm → RegSrc<1>=0, RegSrc<0>=0
RegWr=0

STR

Rn+imm → RegSrc<0>=0
Rt à écrire en mémoire → RegSrc<1>=1
RegWr=0

LDR

Rn+imm → RegSrc<0>=0
RegWr=1

Branchement

R15+ext(imm24) → RegSrc<0>=1
ImmSrc<1..0>=01, RegWr=0

ImmSrc<1..0>=01

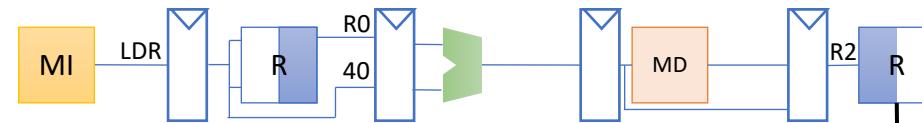
	Instr<27..26>	Instr<25>	Instr<20>	Branch	MemToReg	MemWr	AluSrc	ImmSrc<1..0>	RegWr	RegSrc<1..0>
Calcul reg/reg	00	0	x	0	0	0	0	xx	1	00
Calcul reg/imm	00	1	x	0	0	0	1	00	1	x0
LDR	01	x	0	0	x	1	1	01	1	x0
STR	01	x	1	0	1	0	1	01	0	10
B	10	x	x	1	0	0	1	10	0	x1

Génération des autres signaux (2)

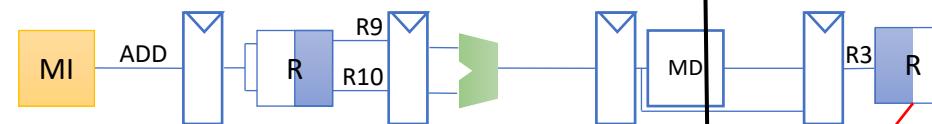
- Il manque PCSrc et tous les signaux pour gérer les codes conditions...
 - On a pas le temps mais rien ne vous empêche de tenter de le faire.
- Vous pouvez vous amuser à prendre une autre instruction de l'ARM (ou un autre mode d'adressage de LDR, STR) et voir comment modifier le pipeline pour que votre processeur soit capable de l'exécuter.

Vue (un peu) plus abstraite du pipeline

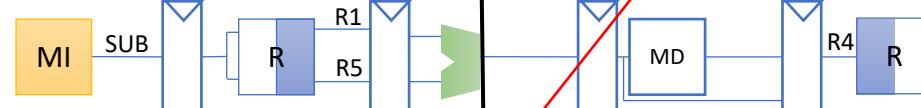
LDR R2, [R0, #40]



ADD R3, R9, R10



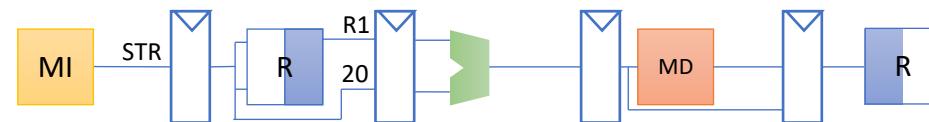
SUB R4, R1, R5



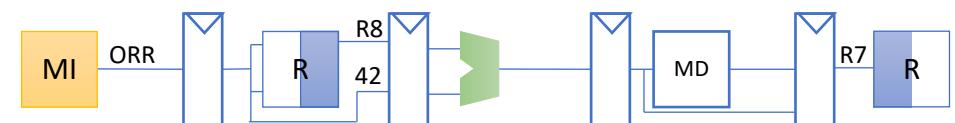
AND R5, R2, R3



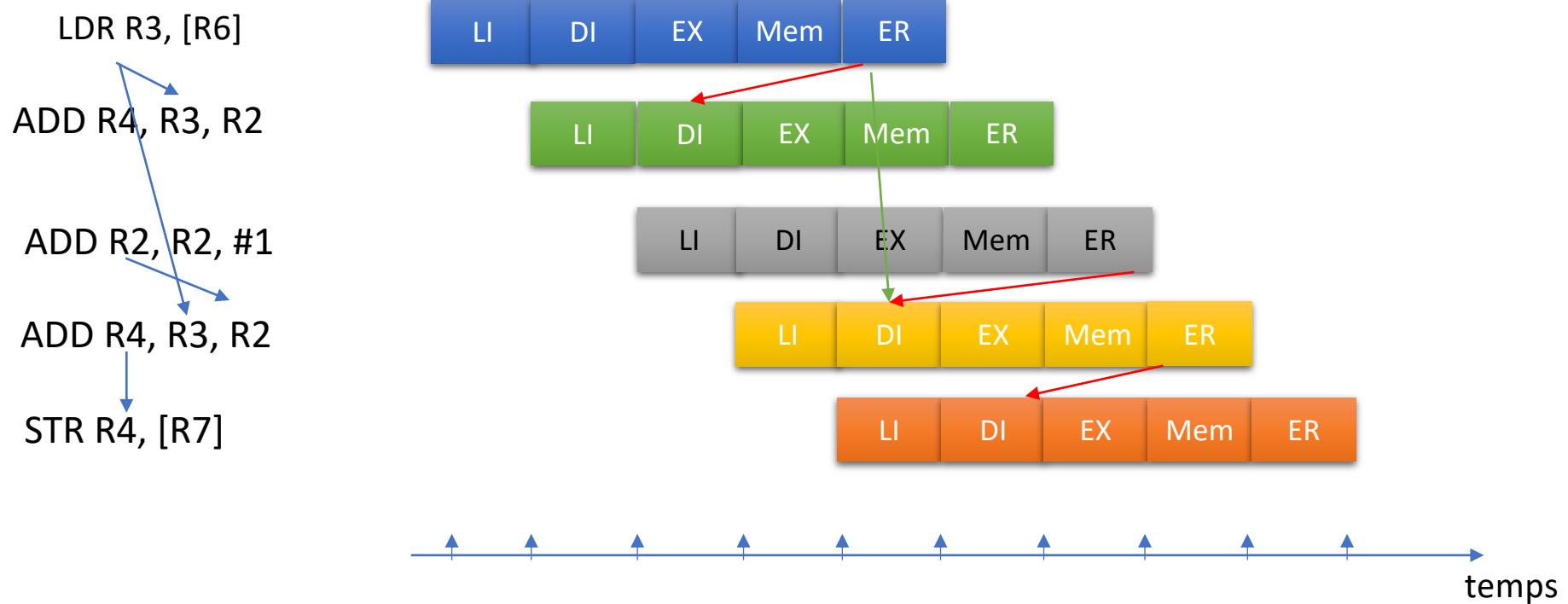
STR R6, [R1, #20]



ORR R7, R8, #42



Dépendances et pipeline – Les aléas de données



Les données ne peuvent pas remonter le temps !

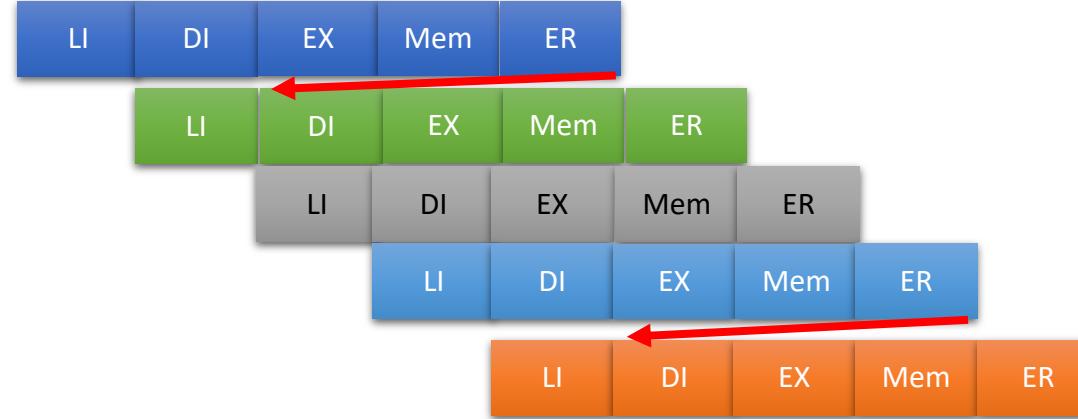
Exercice problème des dépendances

- Supposons que R1 contient 11 et que R2 contient 22.
- Supposons que l'on exécute le code suivant sur le processeur défini au transparent précédent. Quelle est la valeur finale des registres R3 et R4 ?

```
ADD R1, R2, #5
ADD R3, R1, R2
ADD R4, R1, #15
ADD R3,R1, R3
```

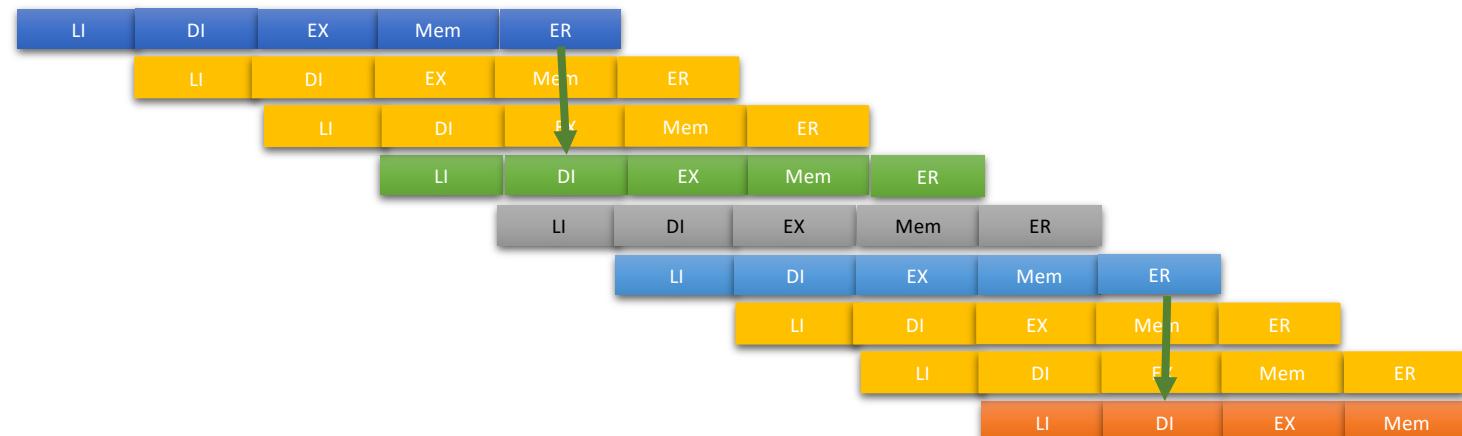
Solution logicielle aux aléas de données

LDR R3, [R6]
ADD R4, R3, R2



ADD R2, R2, #1
ADD R4, R3, R2
STR R4, [R7]

LDR R3, [R6]
NOP
NOP
ADD R4, R3, R2
ADD R2, R2, #1
ADD R4, R3, R2
NOP
NOP
STR R4, [R7]



Exercice

Peut-on écrire le programme ci-dessous de sorte à n'avoir ni NOP ni aléa dans le pipeline dû à la dépendance sur R3 ?

LDR R3, [R6]



NOP



ADD R5, R6, R7



ADD R4, R3, R2



ADD R8, R7, R2

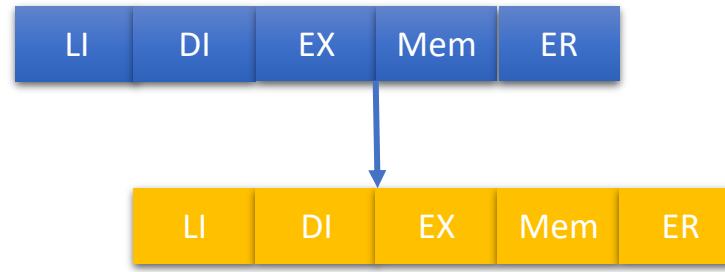


Solution matérielle pour éviter les ALEAS (1)

- Unité d'envoi (ou de court-circuit)

ADD R5, R6, R7

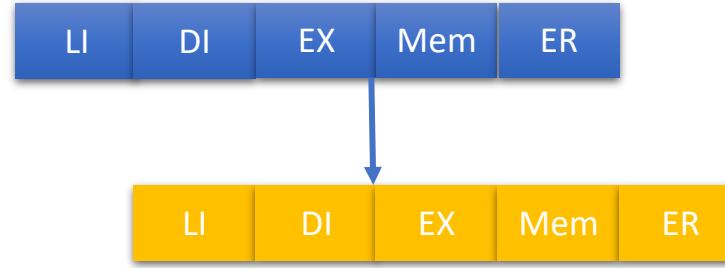
ADD R4, R3, R5



Besoin d'envoyer la sortie de l'ALU
vers l'entrée 2 de l'ALU

ADD R5, R6, R7

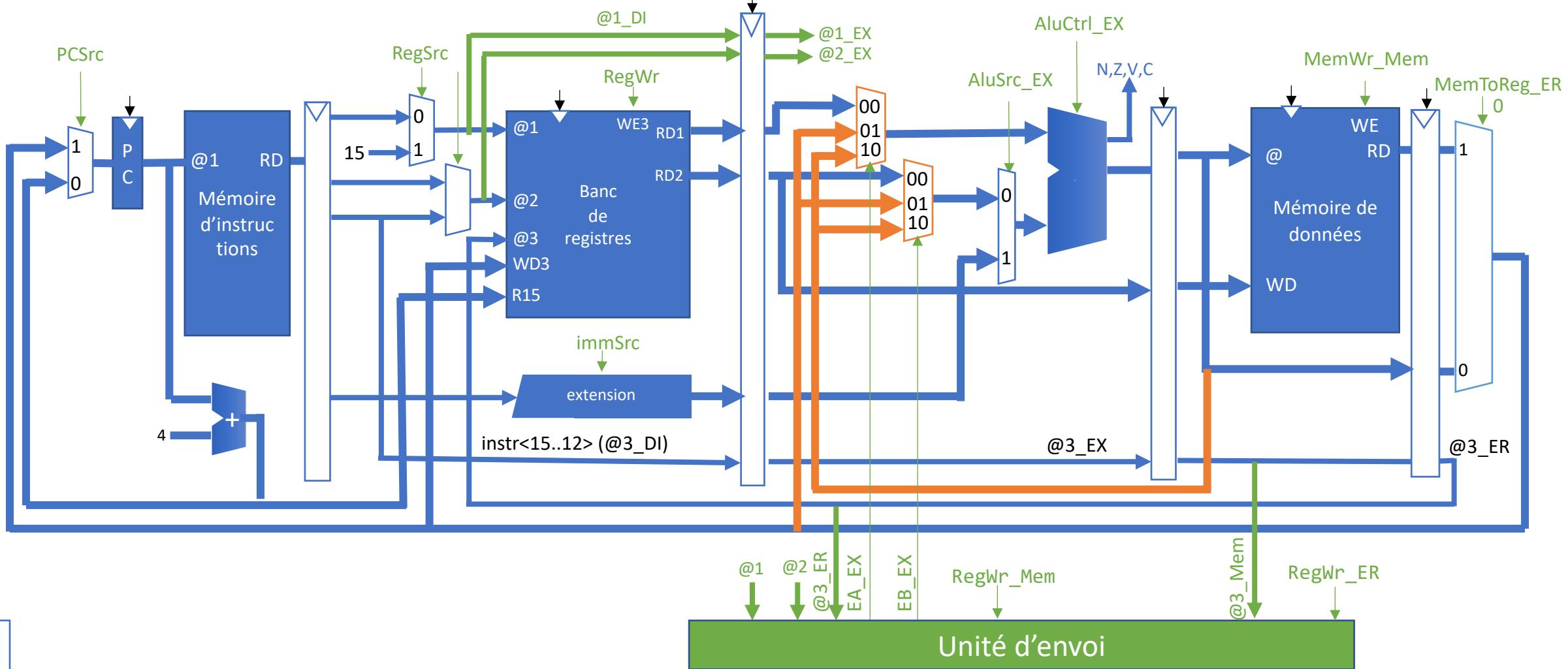
ADD R4, R5, R3



Besoin d'envoyer la sortie de l'ALU
vers l'entrée 1 de l'ALU

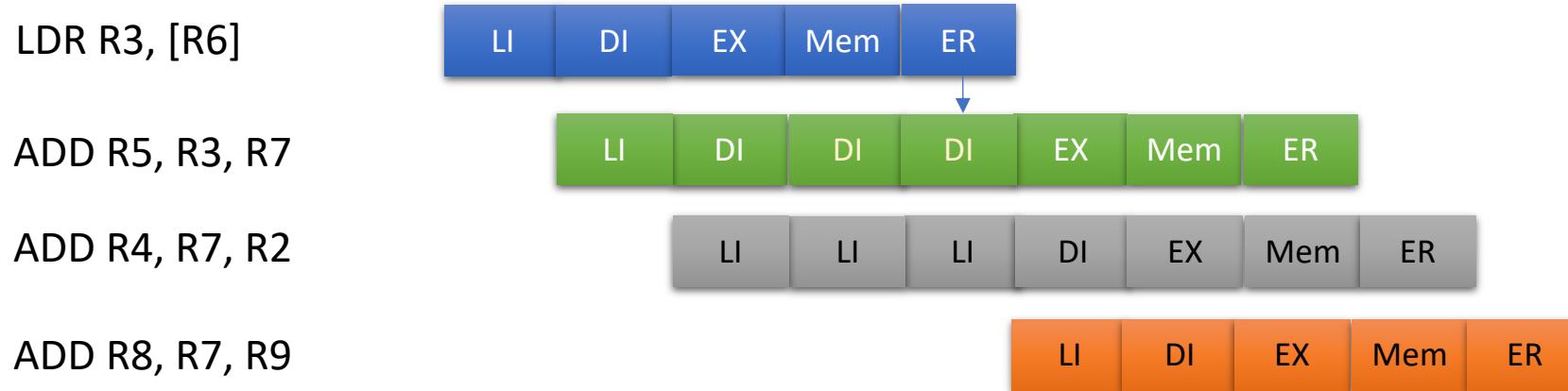
Solution matérielle pour éviter les ALEAS (2)

- Insérer les NOP de manière dynamique

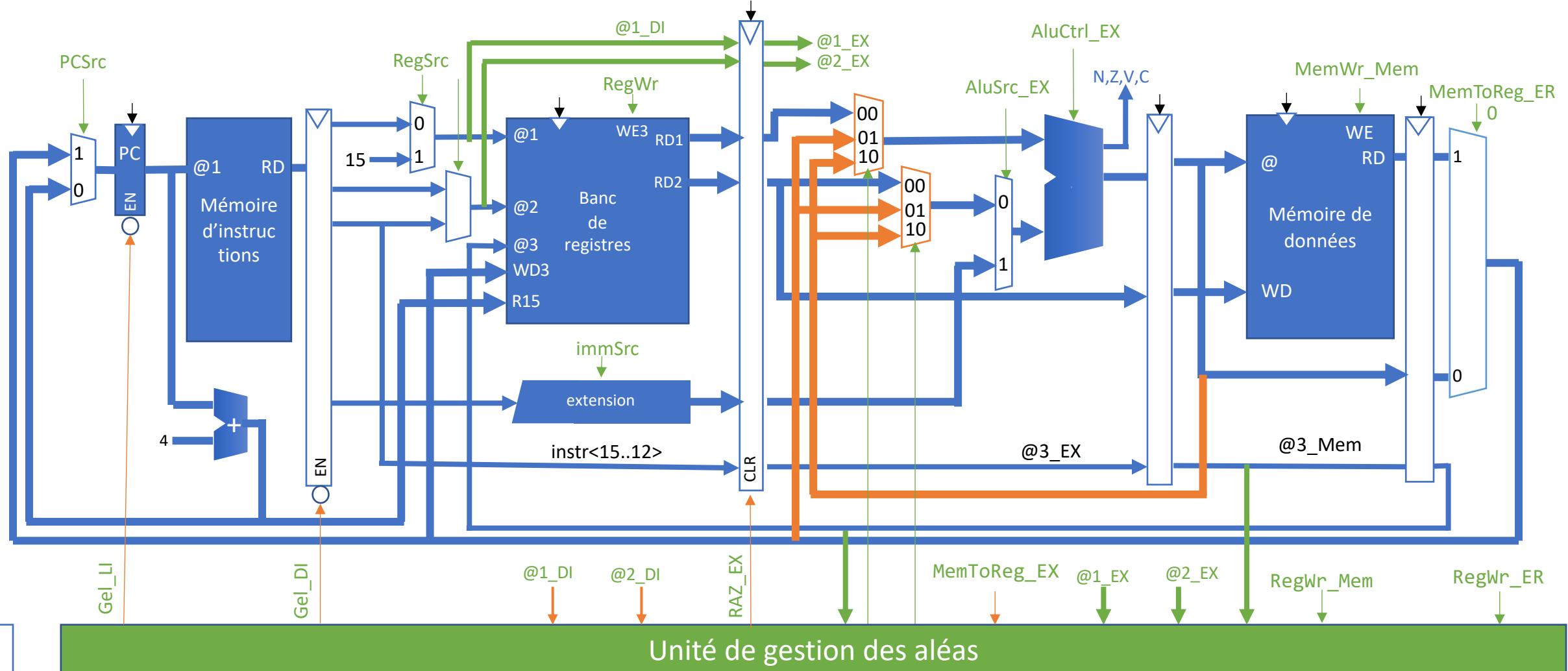


Solution matérielle pour éviter les ALEAS (3)

- Gel du pipeline



Solution matérielle pour éviter les ALEAS (2)



Exercice – simulation pipeline 1 (1)

- Soit le pipeline à 5 étages vu précédemment et en supposant les latences suivantes (valables pour tous les exercices à venir sauf si indication contraire)
 - Opérations entières : 0 cycle, MUL : 6 cycles, DIV : 24 cycles
 - Soit la séquence de code suivante :

LDR R1, [R2] @ $R1 \leftarrow \text{mem}(R2)$

ADD R1, R1, #1 @ $R1 \leftarrow R1 + 1$

STR R1, [R2] @ $\text{mem}(R2) \leftarrow R1$

ADD R2, R2, #4 @ $R2 \leftarrow R2 + 4$

SUB R4, R3, R2 @ $R4 \leftarrow R3 - R2$

Exercice – simulation pipeline 1 (2)

Représenter l'exécution de cette séquence pour le processeur avec gestion des aléas

Brouillon

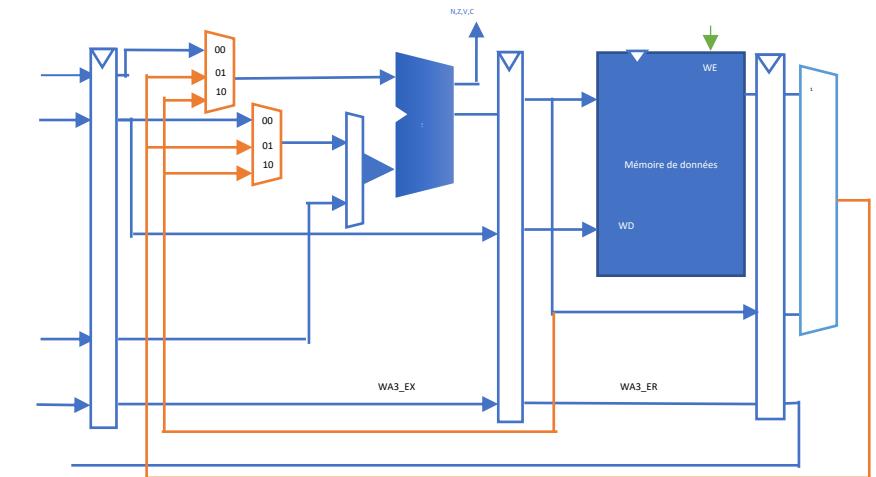
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LDR R1, 0[R2]	LI	DI	EX	M	ER												
ADD R1, R1, #1																	
STR R1, 0[R2]																	
ADD R2, R2, #4																	
SUB R4, R3, R2																	

Solution

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LDR R1, 0[R2]	LI	DI	EX	M	ER												
ADDI R1, R1, #1																	
STR R1, 0[R2]																	
ADD R2, R2, #4																	
SUB R4, R3, R2																	

Solution – simulation pipeline 1 (3)

LDR	R1, [R2]	$\text{@ R1} \leftarrow \text{mem(R2)}$
ADD	R1, R1, #1	$\text{@ R1} \leftarrow \text{R1 + 1}$
STR	R1, [R2]	$\text{@ mem(R2)} \leftarrow \text{R1}$
ADD	R2, R2, #4	$\text{@ R2} \leftarrow \text{R2 + 4}$
SUB	R4, R3, R2	$\text{@ R4} \leftarrow \text{R3 - R2}$



Solution

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LDR R1, [R2]	Li	DI	EX	M	ER												
ADD R1, R1, #1		LI	DI	DI	EX	M	ER										
STR R1, [R2]			LI	LI	DI	EX	M	ER									
ADD R2, R2, #4				LI	DI	EX	M	ER									
SUB R4, R3, R2					LI	DI	EX	M	ER								

Exercice – simulation pipeline 2 (1)

- Soit le pipeline à 5 étages vu précédemment et en supposant les latences suivantes (valables pour tous les exercices à venir sauf si indication contraire)
 - Opérations entières : 0 cycle, MUL : 6 cycles, DIV : 24 cycles
 - Soit la séquence de code suivante :

LDR R2, [R0] @ R2 \leftarrow mem(R0)

LDR R3, [R0, #4] @ R3 \leftarrow mem(R0+4)

MUL R5, R3, R2 @ R5 \leftarrow R3 * R2

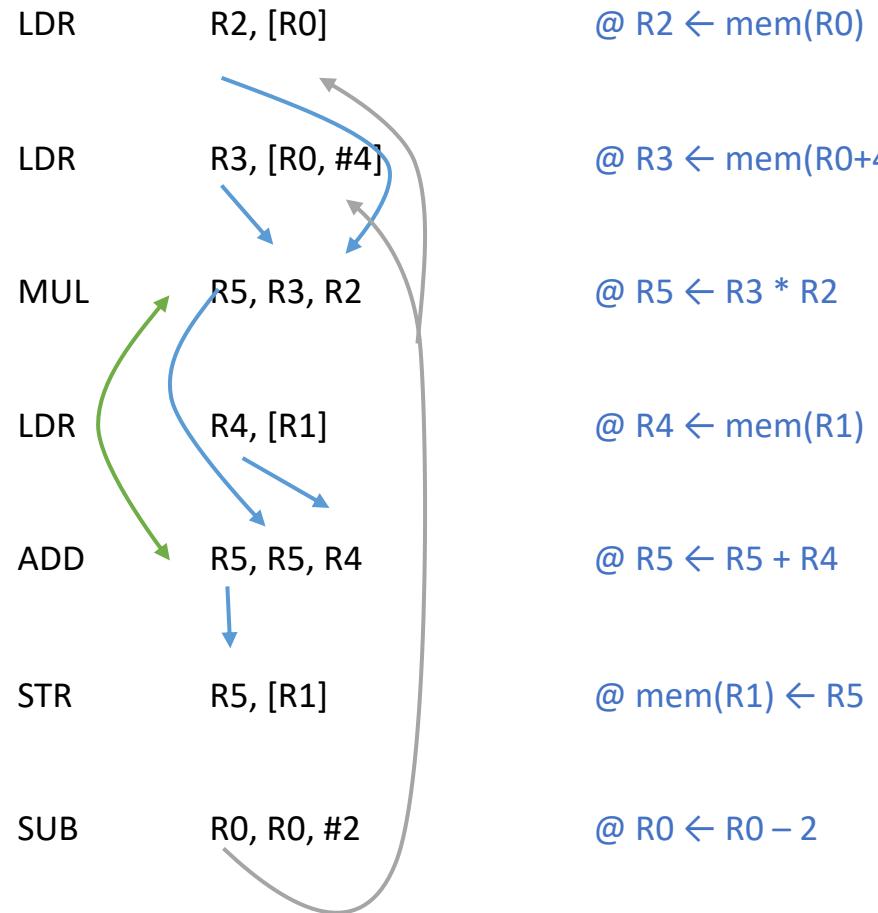
LDR R4, [R1] @ R4 \leftarrow mem(R1)

ADD R5, R5, R4 @ R5 \leftarrow R5 + R4

STR R5, [R1] @ mem(R1) \leftarrow R5

SUB R0, R0, #2 @ R0 \leftarrow R0 - 2

Solution – simulation pipeline 2 dépendances (1)

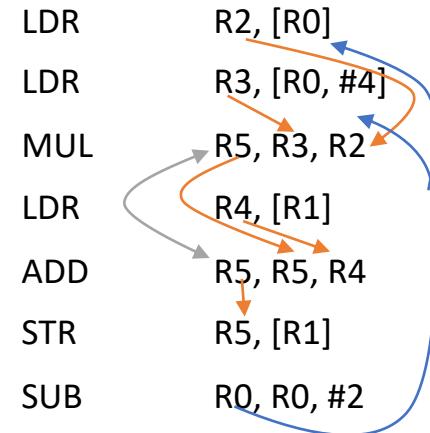


Exercice – simulation pipeline 2 (2)

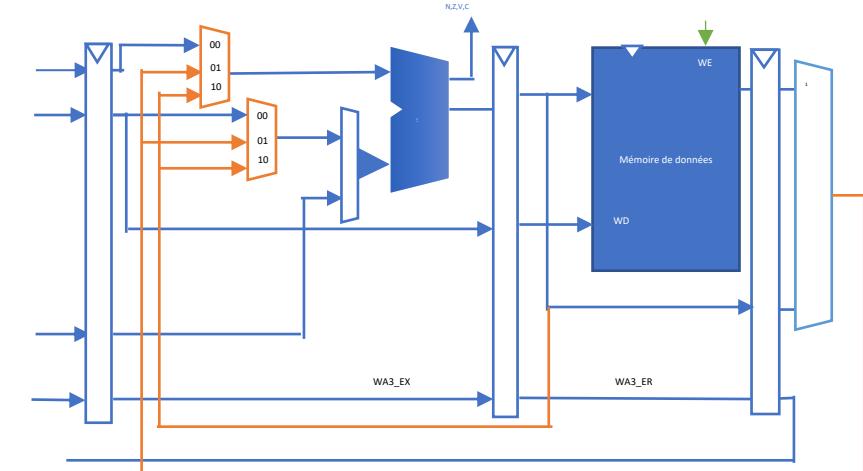
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
<i>Brouillon</i>	LDR R2, [R0]	LI	DI	EX	M	ER												
	LDR R3, [R0, #4]																	
	MUL R5, R3, R2																	
	LDR R4, [R1]																	
	ADD R5, R5, R4																	
	STR R5, [R1]																	
	SUB R0, R0, #2																	

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
<i>Solution</i>	LDR R2, [R0]	LI	DI	EX	M	ER												
	LDR R3, [R0, #4]																	
	MUL R5, R3, R2																	
	LDR R4, [R1]																	
	ADD R5, R5, R4																	
	STR R5, [R1]																	
	SUB R0, R0, #2																	

Solution – simulation pipeline 2 avec forwarding (2)



$\text{@ R2} \leftarrow \text{mem(R0)}$
 $\text{@ R3} \leftarrow \text{mem(R0+4)}$
 $\text{@ R5} \leftarrow \text{R3 * R2}$
 $\text{@ R4} \leftarrow \text{mem(R1)}$
 $\text{@ R5} \leftarrow \text{R5 + R4}$
 $\text{@ mem(R1)} \leftarrow \text{R5}$
 $\text{@ R0} \leftarrow \text{R0 - 2}$



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LDR R2, [R0]	LI	DI	EX	M	ER													
LDR R3, [R0, #4]		LI	DI	EX	M	ER												
MUL R5, R3, R2			LI	DI	DI	EX	EX	EX	EX	EX								
LDR R4, [R1]				LI	LI	DI	DI	DI	DI	DI	EX	M	ER					
ADD R5, R5, R4					LI	LI	LI	LI	LI	DI	DI	EX	M	ER				
STR R5, [R1]										LI	LI	DI	EX	M	ER			
SUB R0, R0, #2												LI	DI	EX	M	ER		

Solution

Supprimer la suspension - Réordonnancement

A la charge du compilateur

```
LDR R3, [R6]  
ADD R4, R3, R2  
ADD R5, R5, #1
```



```
LDR R3, [R6]  
ADD R5, R5, #1  
ADD R4, R3, R2
```

Pas toujours possible à cause des dépendances

Dépendance
vraie

```
LDR R3, [R6]
```

```
ADD R4, R3, R2
```

Anti-
dépendance

```
ADD R2, R2, #1
```

```
ADD R4, R3, R2
```

```
STR R4, [R7]
```

Dépendance
de
sortie

Il ne faut violer aucune dépendance.
Le compilateur peut modifier les registres pour
supprimer des fausses dépendances. Encore
faut-il que des registres soient disponibles...

Renommage de registres

```
LDR R3, [R6]
```

```
ADD R4, R3, R2
```

```
ADD R8, R2, #1
```

```
ADD R9, R3, R8
```

```
STR R9, [R7]
```

Exercice – simulation pipeline 2 (3)

Réordonner le code pour minimiser les suspensions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
<i>Brouillon</i>																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
<i>Solution</i>																	

Solution– simulation pipeline 2 (3)

Réordonner le code pour minimiser les suspensions

Solution

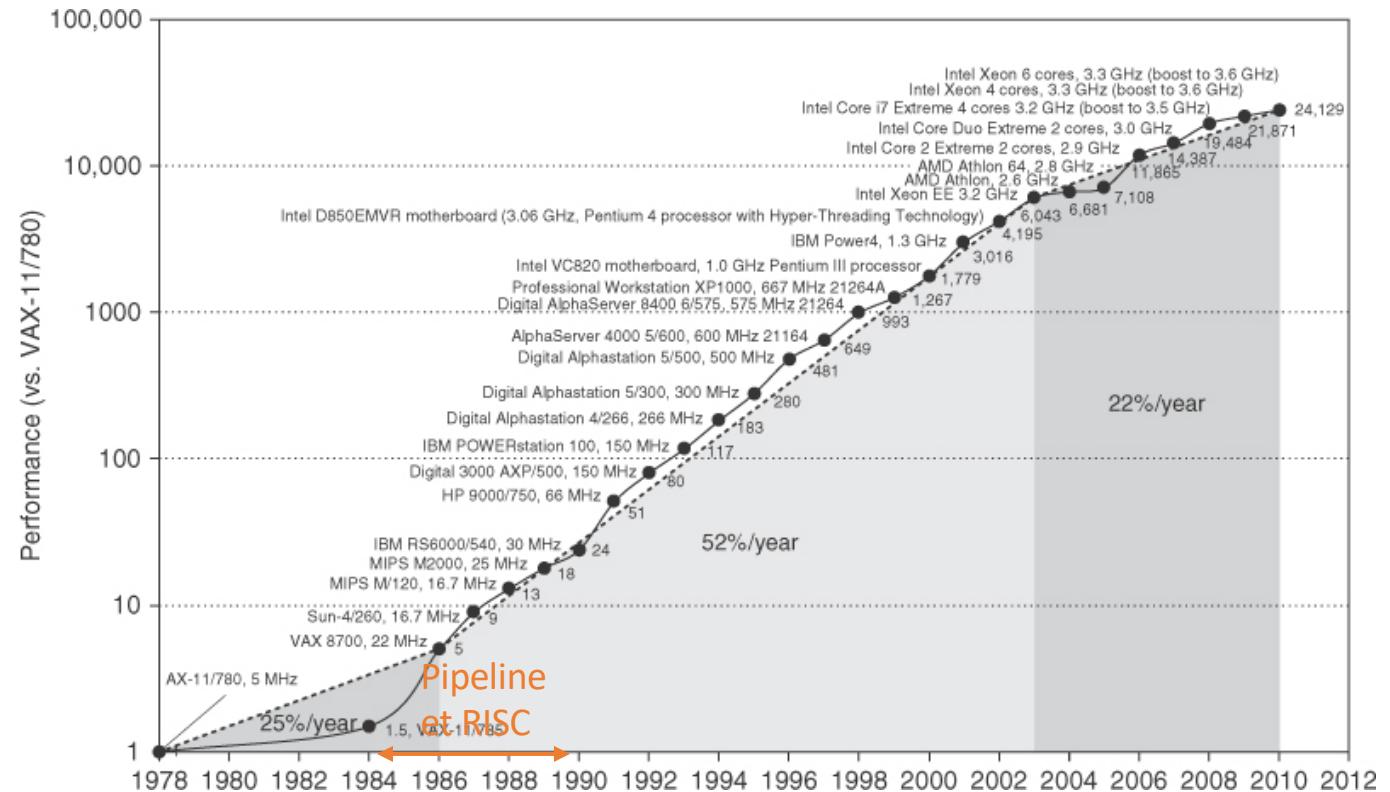
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LDR R2, [R0]	LI	DI	EX	M	ER												
LDR R3, [R0, #4]		LI	DI	EX	M	ER											
LDR R4, [R1]			LI	DI	EX	M	ER										
MUL R5, R3, R2				LI	DI	EX	EX	EX	EX	EX	EX	M	ER				
ADD R5, R5, R4					LI	DI	DI	DI	DI	DI	DI	EX	M	ER			
STR R5, [R1]						LI	LI	LI	LI	LI	LI	DI	EX	M	ER		
SUB R0, R0, #2												LI	DI	EX	M	ER	

En résumé

- Le pipeline permet d'augmenter la fréquence de fonctionnement et le débit d'instructions

La performance ne se mesure plus en CPI mais en IPC.

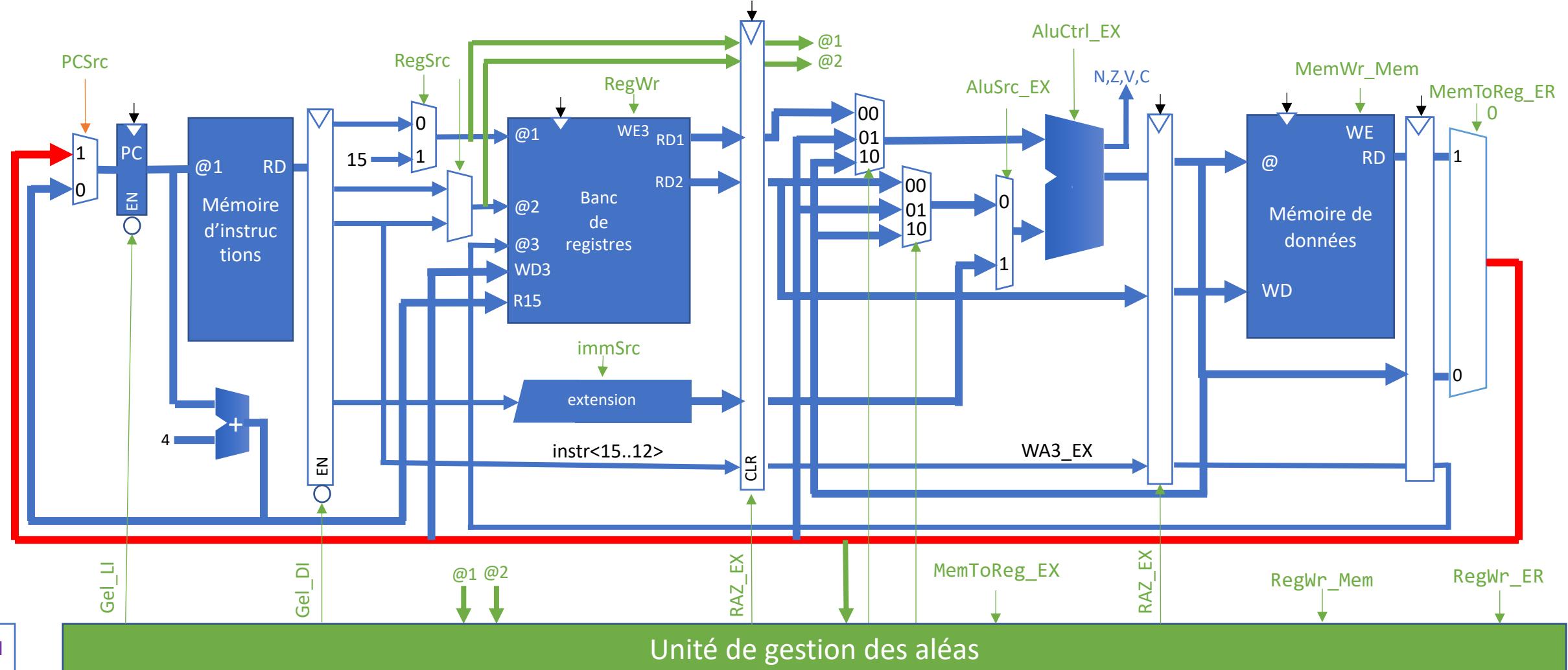
Pipeline et RISC datent de 1985 (début de la pente à 52%).



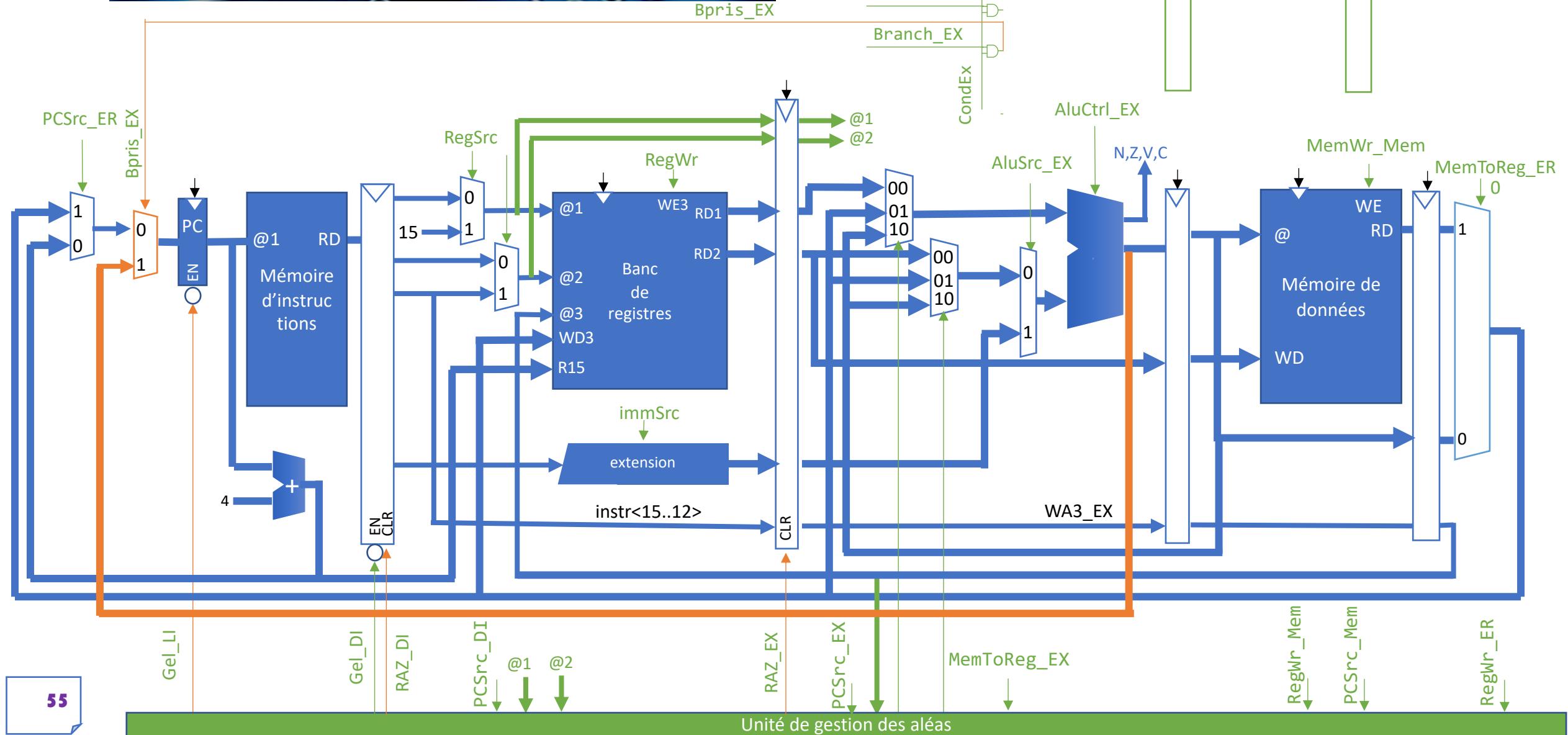
Rappels sur les ruptures du flux de contrôle

1. Par défaut, le processeur doit exécuter l'instruction qui suit
2. 2 types de branchement
 - Branchement non conditionnels
B, BL, écriture dans R15 (PC)
 - Branchements conditionnels
BEQ, BNE, BHI, BLO, ... Le branchement est pris si le registre CPSR indique que la condition est vérifiée.
CPSR est mis à jour par CMP, TST, xxxS

Aléa de contrôle



Aléas de contrôle réduit

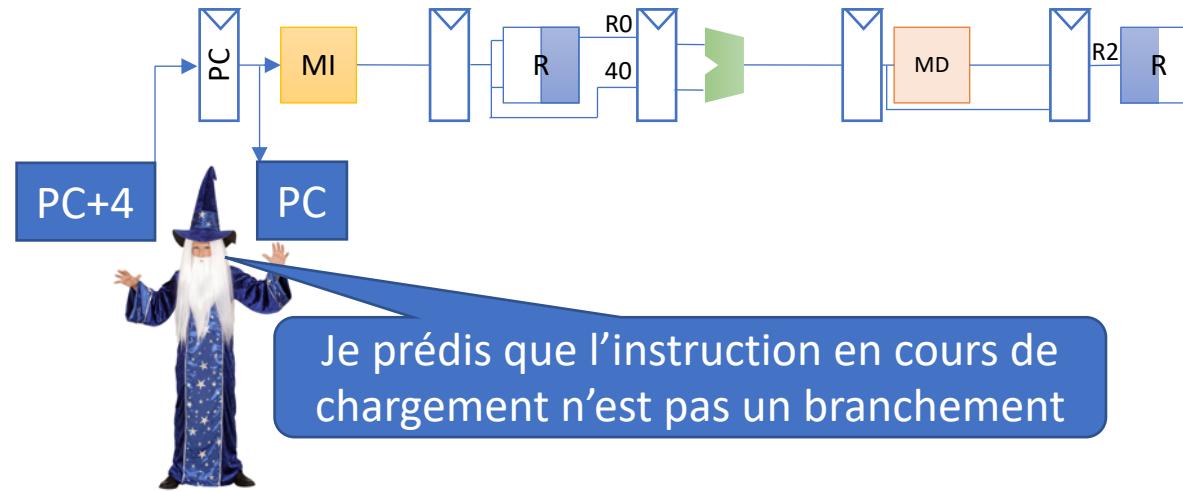


Quelques exemples de pipeline

- Cortex A5
 - 8 étages
- Cortex A8
 - LI sur 3 étages, DI sur 5 étages, EX sur 5 étages, ER sur 1 étage

Prédiction de branchement

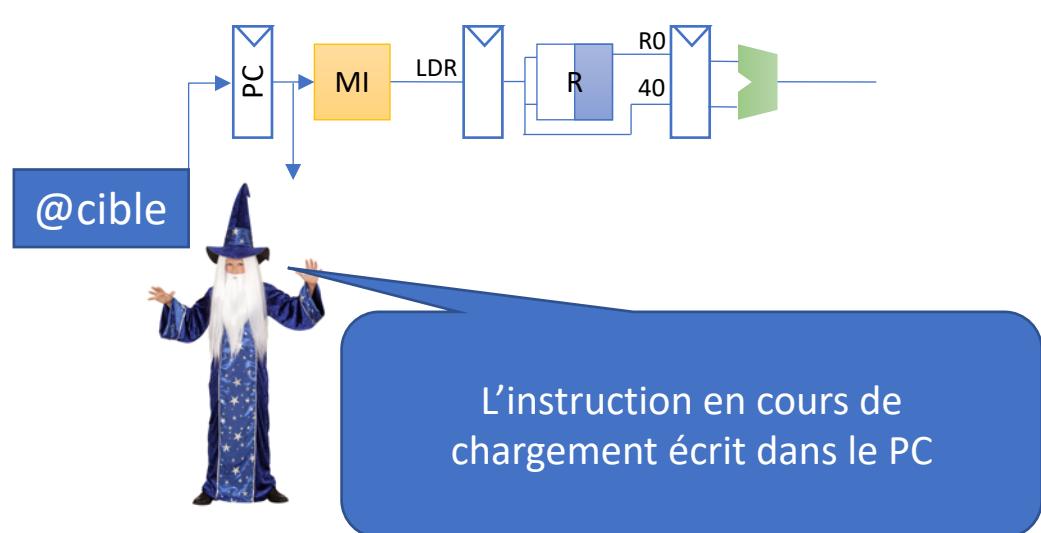
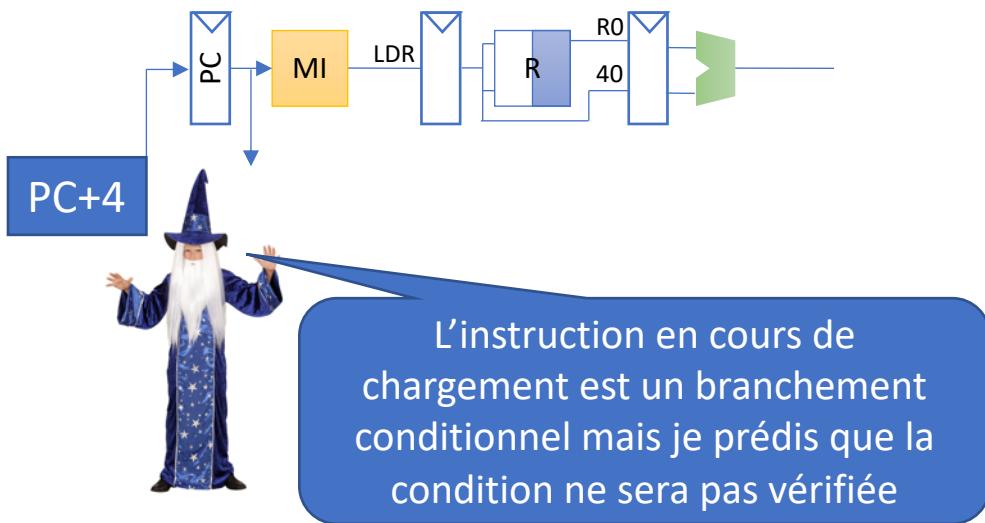
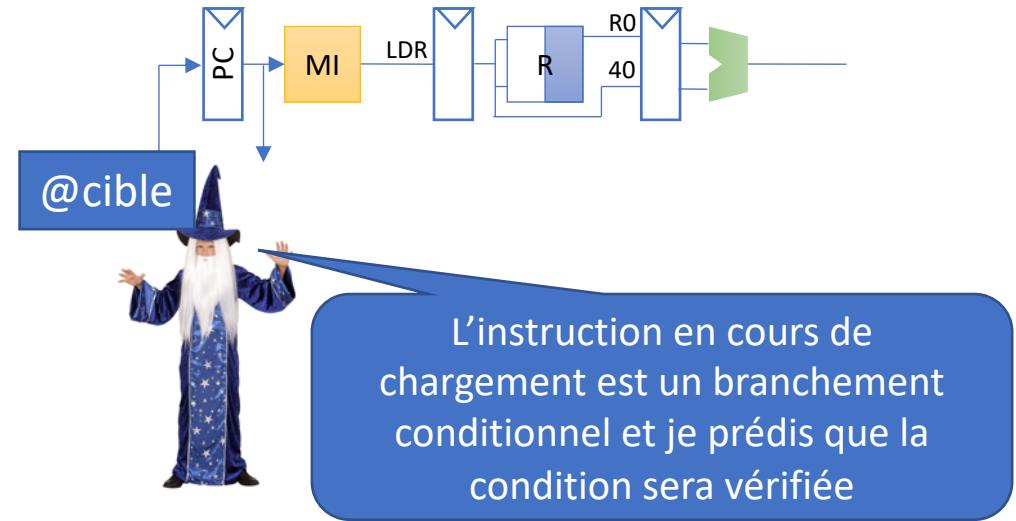
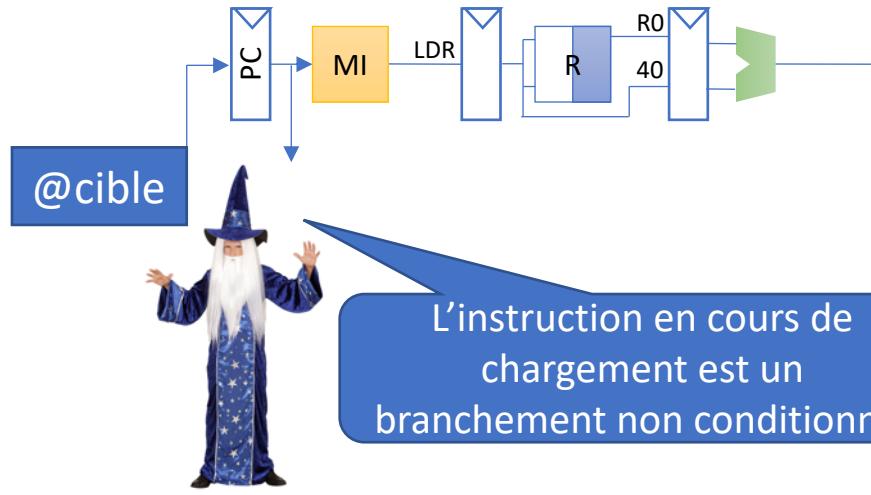
- Le pipeline précédent prédit systématiquement que les branchements ne sont pas pris et on a une pénalité de 3 cycles chaque fois qu'un branchement est pris.
- Pour éviter cette pénalité, il faudrait calculer les branchements dès le premier cycle
 - Impossible
- A défaut, prédire les branchements dès le premier cycle, vérifier à l'exécution que la prédiction était correcte. On ne paye la pénalité que si la prédiction est incorrecte.



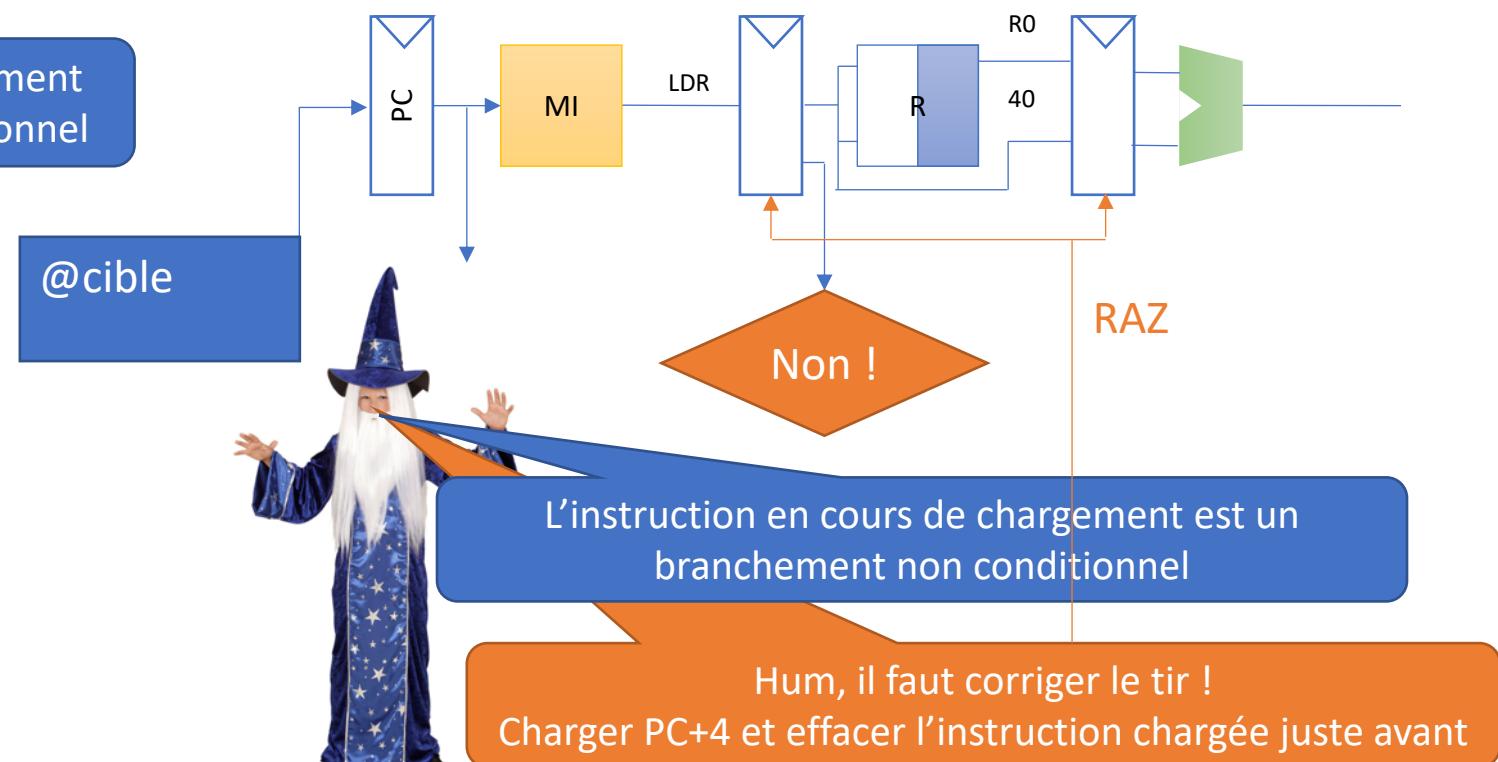
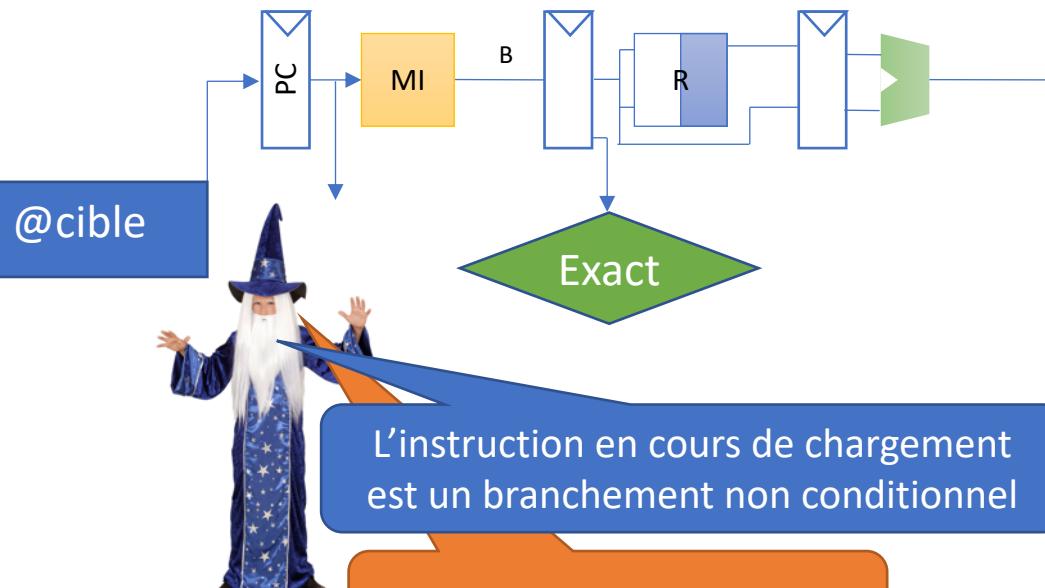
Prédiction de branchement: exercice 1

- Le processeur non pipeline a un débit d'une instruction toutes les 680ps.
- Le processeur pipeline (5 étages) a un temps de cycle de 200ps.
 - Chaque branchement pris implique une pénalité de 3 cycles.
- Un programme de 100 instructions comporte 20 branchements.
- Quel est le débit du processeur pipeline ?

Prédiction de branchement



Prédiction de branchement



Prédiction statique de la direction des branchements

```
for (i=0; i<256; i++) {  
    if (x==y) ...  
    else ...  
}
```

for:	mov r2, #0	
	cmp r2, #256	
	beq finfor	← Branchement en avant : généralement non pris
	cmp r3, r4	
	bne sinon	← Branchement en avant : ? généralement non pris
	...	
	b suite	← Branchement en avant : toujours pris
sinon:		
	add r2, r2, #1	
	b for	← Branchement en arrière : toujours pris
suite:	...	
finfor:	...	

BTFNT : Backward Taken, Forward not taken

Problème de la prédiction statique

- Il faut savoir que l'instruction que l'on est en train de charger est un branchement
 - On ne le sait qu'à l'étage de décodage
 - Perte d'un cycle chaque fois qu'un branchement est pris.
- NB : la prédiction de la direction ne concerne que les branchements conditionnels
 - Une fois décodés, on sait que les branchements non conditionnels sont pris

Prédiction statique de la direction des branchements

```
        mov  r2, #0
for:    cmp  r2, #256
        beq  finfor
        cmp  r3, r4
        bne  sinon
        ...
        // 4 cycles
        b    suite
sinon:
        ...
        // 5 cycles
suite: add  r2, r2, #1
        b    for
finfor: ...
```

1. Quel est le taux minimal et maximal de mauvaises prédictions de ce programme sans prédiction de branchement (prédiction toujours non pris)
2. Quel est le taux minimal et maximal de mauvaises prédictions de ce programme avec un prédicteur statique BTFNT ?

Prédiction dynamique

- Prédiction qui prend en compte le comportement passé des branchements
- Prédiction dès le premier cycle
 - Prédiction de la présence d'un branchement
 - Prédiction du type du branchement
 - Prédiction de l'adresse cible
- Vérification lors de l'exécution du branchement et annulation si mauvaise prédiction

Prédiction de la direction dynamique

Compteurs 1 bit

- Prédiction du comportement
 - Table d'historique des branchements:
 - petite mémoire indexée par l'adresse de l'instruction
 - Moins d'entrée que la mémoire principale, accès direct
 - interférences entre branchements
 - Entrée = un bit qui indique si le branchement était pris ou non la dernière fois qu'on l'a exécuté
 - Si prédiction fausse, inversion du bit
 - Pb : branchement de fin de boucle exécutée plusieurs fois
 - 2 erreurs chaque fois qu'un branchement change de comportement

Boucles imbriquées

BCL1: ...

...

BCL2: ...

...

BLS

BCL2

...

BLS

BCL2

Comportement B2	P	P	P	P	NP	P	P	P
Prédiction B2	?	P	P	P	P	NP	NP	P

Exercice prédition statique et dynamique 1 bit

- Soit les 3 branchements ayant les comportements suivants

- B1

P P P N N N N N N P P P P N N N N N N N P P P N P

- B2

P P P P N P P P P N P P P N N N N P N N N P N N N N

- B3

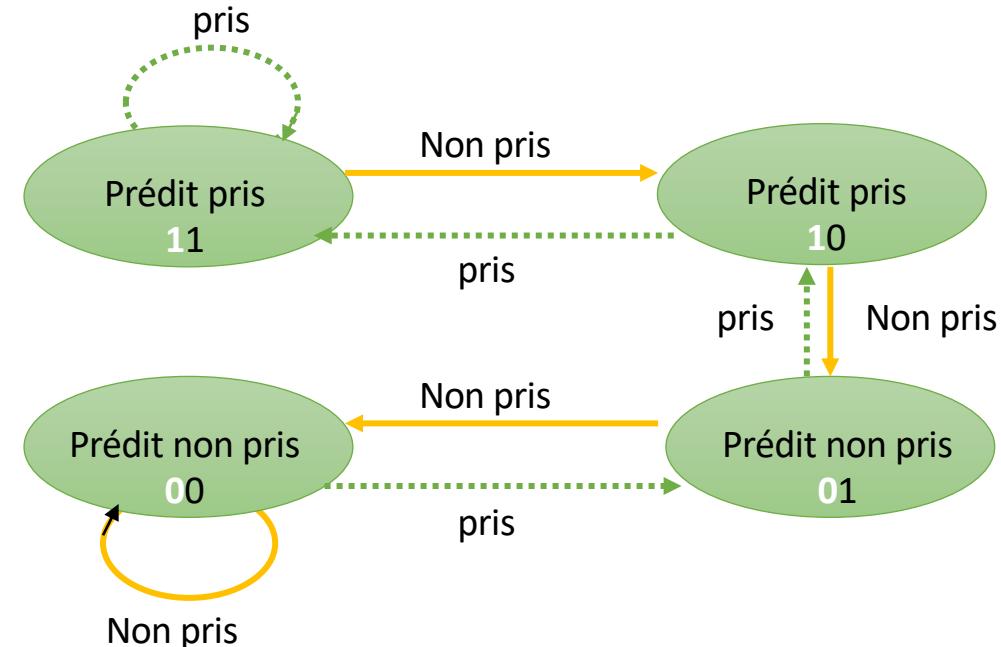
N N N N N P N N N N N P N N N N N P N N N N N P N N

- Donner le nombre d'erreurs de prédition pour
 - Prédition statique pris
 - Prédition statique non pris
 - Prédition dynamique 1 bit (état initial à 0)

Prédiction de la direction dynamique

compteurs 2 bits

- La prédiction ne change que si un branchement a 2 fois la même prédiction
- Initialisation de la table à 10 ou 01 suivant la direction majoritaire (dépend du compilateur).
- Taille de la table : 4096 entrées apporte une précision quasiment identique à une table infinie (de 0 à 15% de mauvaises prédictions)
- La différence entre un compteur 3 bits et un 2 bits est très faible.



Comportement B2	P	P	P	P	NP	P	P	P
Compteur	10	11	11	11	11	10	11	11
Prédiction	P	P	P	P	P	P	P	P

Exercice1 prédition dynamique 2 bits

- Soit les 3 branchements ayant les comportements suivants

- B1

P P P N N N N N N P P P P N N N N N N N P P P N P

- B2

P P P P N P P P P N P P P N N N N P N N N P N N N N

- B3

N N N N N P N N N N N P N N N N N P N N N N N P N N

- Donner le nombre d'erreurs de prédition pour

- Prédition dynamique 2 bits, compteurs initialisés à 10

Exercice 2 : prédiction dynamique 2 bits

- On suppose que l'état initial des compteurs 2 bits est 10 et qu'il n'y a pas de conflits dans la table des compteurs.
- Soit le programme suivant

```
for (i=0; i<5; i++)  
    ...
```

```
for:   CMP      MOV      R1, #0  
        R1, #5  BEQ      finfor  
        ...  
        B       for  
finfor:
```

- Quel est le taux de bonnes prédictions du BEQ `finfor` ?

Compteur	10						
Prédiction	P						
Comportement							

Solution exercice 2 : prédition dynamique 2 bits

Compteur	10	01	00	00	00	00
Prédiction	P	NP	NP	NP	NP	NP
Comportement	NP	NP	NP	NP	NP	P

Tx bonnes prédictions = 4/6 = 66,7%

Exercice 3 : prédiction dynamique 2 bits

- Soit le programme suivant compilé de la même façon que dans l'exemple précédent

```
for (i=0; i<100; i++) // b1
    for (j=0; j<5; j++) // b2
        tab[i] = 0;
...
...
```

- Quels sont les taux de bonnes prédictions des branchements conditionnels b1 et b2 ? Quel est le taux global de bonnes prédictions des branchements conditionnels ?

Compteur b1	10					
Prédiction b1	P					
Comportement b1						

Compteur b2	10					
Prédiction b2	P					
Comportement b2						

Solution exercice 3 : prédiction dynamique 2 bits

Compteur b1	10	01	00	00	...	00
Prédiction b1	P	NP	NP	NP	...	NP
Comportement b1	NP	NP	NP	NP	...	P

$$Taux\ bonnes\ prédictions\ b1 = \frac{99}{101} = 98\%$$

Compteur b2	10	01	00	00	00	00	01	00	00	00	00	00
Prédiction b2	P	NP										
Comportement b2	NP	NP	NP	NP	NP	P	NP	NP	NP	NP	NP	P

$$Taux\ bonnes\ prédictions\ b2 = \frac{1 * 4 + 99 * 5}{600} = 83\%$$

$$Taux\ global\ bonnes\ prédictions = \frac{99 + 1 * 4 + 99 * 5}{101 + 600} = \frac{598}{701} = 85\%$$

Exercice 4 : prédition dynamique 2 bits

- Soit le programme suivant compilé de la même façon que dans l'exemple précédent. On suppose pour b3 et b4 que les branchements sont pris lorsque la condition est fausse.

```
for (i=0; i<100; i++)                                // b1
    for (j=0; j<10; j++)                            // b2
        if (j%2 == 0)                                // b3
            if (i%2 == 0)      // b4
                a[j] = 0;
...

```

- Quels sont les taux de bonnes prédictions des branchements conditionnels b1, b2, b3 et b4 ? Quel est le taux global de bonnes prédictions des branchements conditionnels ?

Compteur b3	10							...
Prédiction b3	P							...
Comportement réel B3								...

Compteur b4	10																		...
Prédiction b4	P																		...
Comportement réel B4																			...

Solution exercice 4 : prédition dynamique 2 bits

- $b1 = 99/101 = 98\%$
- $b2 = (1*9 + 99*10)/1100 = 999/1100 = 91\%$

Compteur b3	10	01	10	01	10	01	...
Prédiction b3	P	NP	P	NP	P	NP	...
Comportement réel B3	NP	P	NP	P	NP	P	...

- $b3 = 0/1000 = 0\%$

Compteur b4	10	01	00	00	00	00	01	10	11	11	11	10	01	...
Prédiction b4	P	NP	NP	NP	NP	NP	NP	P	P	P	P	P	NP	...
Comportement réel B4	NP	NP	NP	NP	NP	P	P	P	P	P	NP	NP	NP	...

- $b4 = (1*4+99*3)/500 = 301/500 = 60\%$

$$Taux\ global = \frac{99 + 999 + 0 + 301}{101 + 1100 + 1000 + 500} = \frac{1399}{2701} = 51,8\%$$

Prédiction de la direction dynamique

Historique global - Introduction

- Un branchement peut avoir un comportement qui dépend des branchements précédents

Exemple de programme

```
if (aa=2)
    aa = 0;
if (bb=2)
    bb = 0;
if (aa != bb)
    { ... }
```

Traduction en assembleur
aa dans R1, bb dans R2

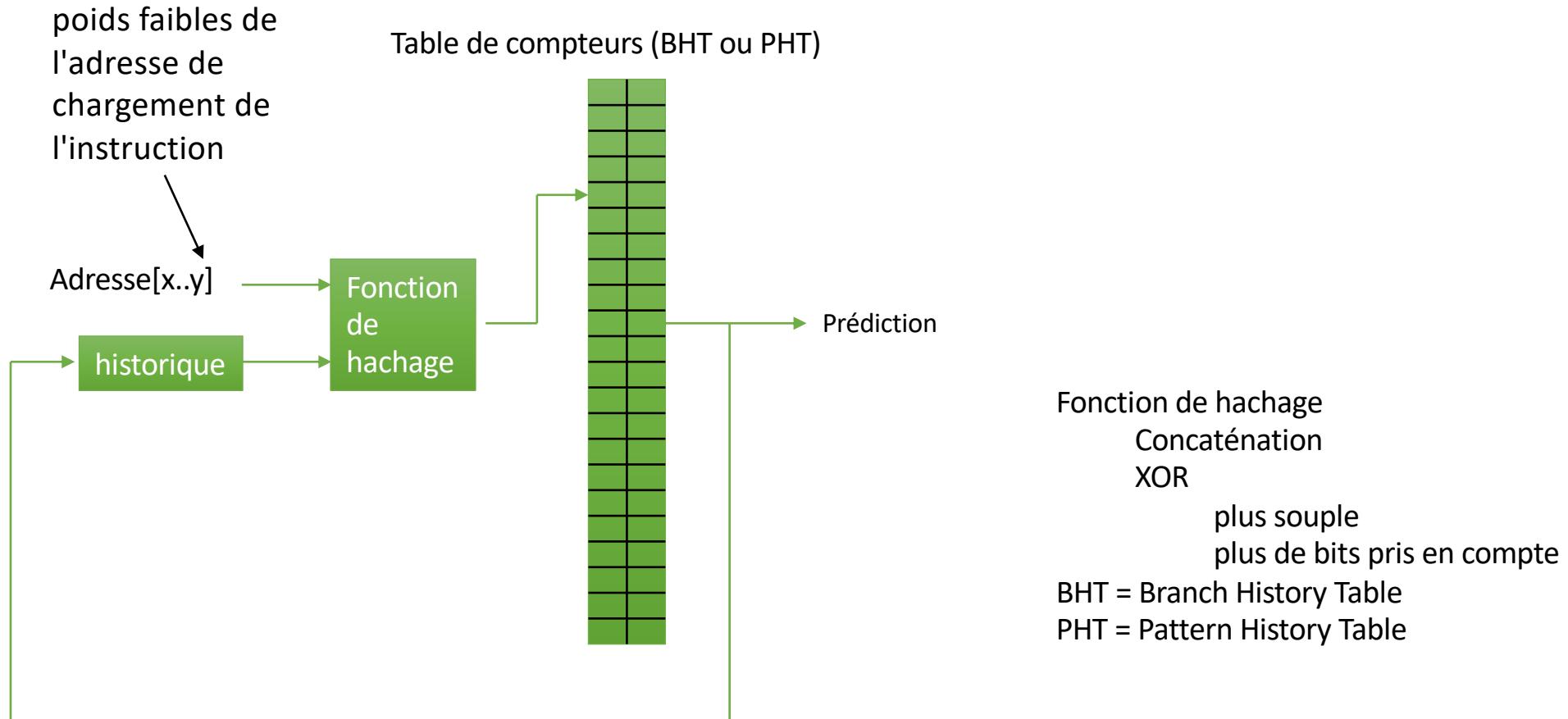
```
SUBS      R3, R1, #2
b1: BNE     L1
      MOV      R1, #0
L1: SUBS R3, R2, #2
b2: BNE     L2
      MOV R2,#0
L2: SUBS      R3, R1, R2
b3: BNE     L3
```

Si b1 et b2 ne sont pas pris,
b3 le sera car aa = bb = 0.
b3 est corrélé à b1 et b2.

Prédicteurs à corrélation ou
prédicteurs à deux niveaux.

Prédiction de la direction dynamique

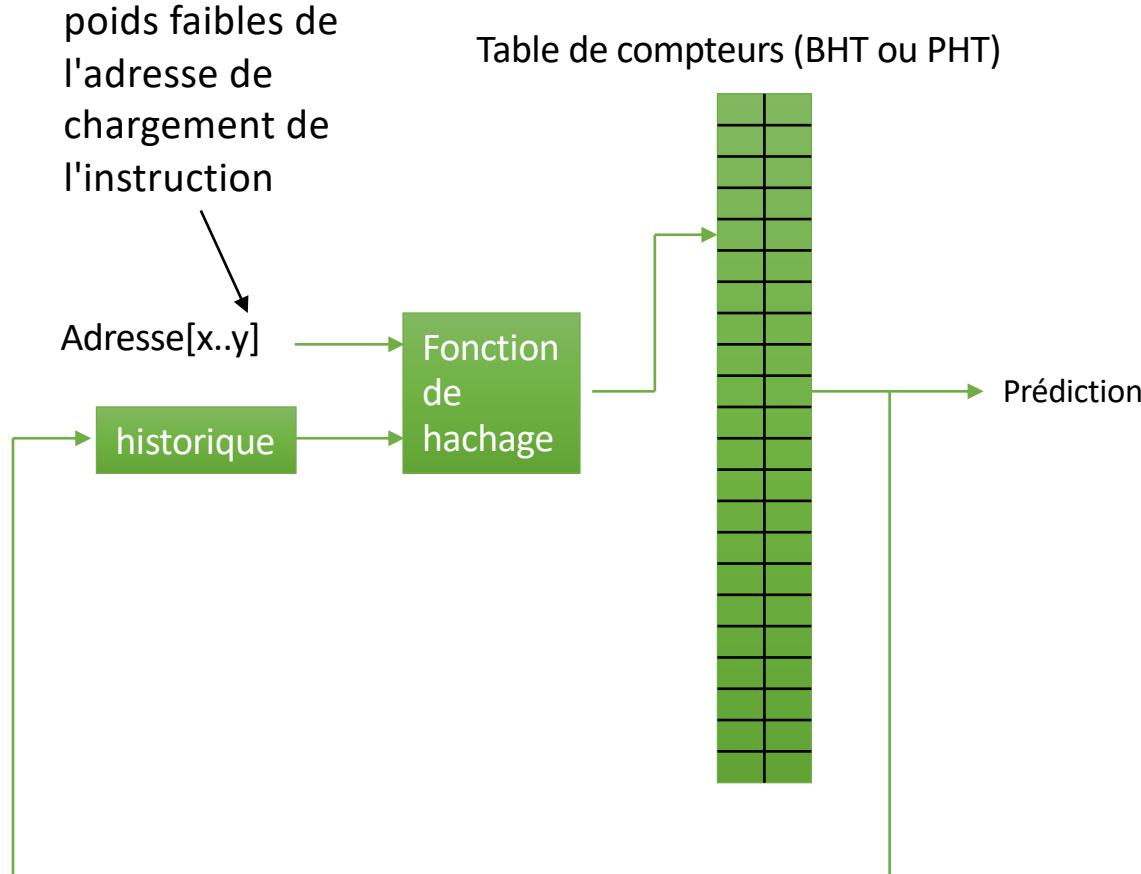
Historique global (1)



Prédiction de la direction dynamique

Historique global (2)

Prédicteur caractérisé par (nombre de bits d'historique, nombre de bits d'un compteur)



Exemple de concaténation (4,2)

8 bits d'adresse
4 bits d'historique
donne accès à 2^{12} prédicteurs 2 bits
2 branchements se partagent un prédicteur s'ils sont éloignés de 2^8 instructions et qu'il y a le même historique global.

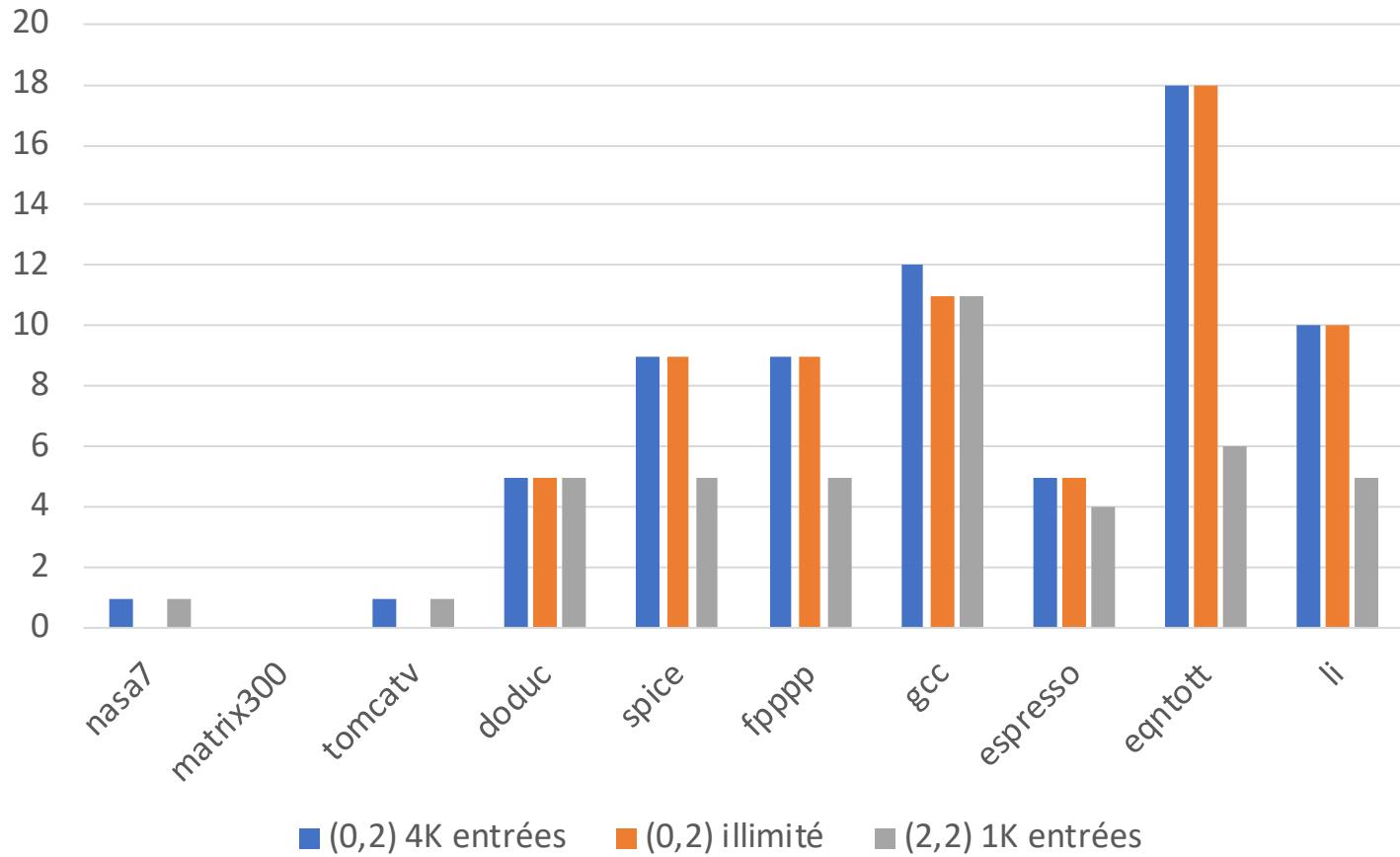
Exemple de XOR (12, 2)

12 bits d'adresse
12 bits d'historique
donne accès à 2^{12} prédicteurs 2 bits
Certains branchements distants de moins de 2^{12} instructions peuvent partager un prédicteur...

Prédiction de la direction dynamique

Historique global (3)

Taux de mauvaises prédictions en %



Exercice prédicteur à historique global (1)

- Soit le code suivant

```
int tab[9] = {8, 9, 10, 11, 12, 20, 29, 30, 31};  
int a=0, b=0;  
  
for (i=0 ; i<9 ; i++) {  
    if (tab[i]%2 == 0)           branchement b1  
        a++;                   [b1 pris]  
    if (tab[i]%10 == 0)          branchement b2  
        b++;                   [b2 pris]  
}
```

- **Question 1.**

Les branchements sont prédits à partir de compteurs 1-bit, initialisés à 0 (*non pris*). Il n'y a pas de conflit dans la table de compteurs entre b_1 et b_2 . Chaque branchemenmt met à jour son compteur avant que le branchemenmt suivant ne soit exécuté.

Exercice prédicteur à historique global (2)

- **Question 1.**

Les branchements sont prédits à partir de compteurs 1-bit, initialisés à 0 (*non pris*). Il n'y a pas de conflit dans la table de compteurs entre $b1$ et $b2$. Chaque branchement met à jour son compteur avant que le branchement suivant ne soit exécuté.

Indiquer, dans le tableau suivant, comment les deux branchements sont prédits et finalement exécutés puis indiquer le taux de bonnes prédictions de chacun des deux branchements puis le taux global.

tab[i]	8	9	10	11	12	20	29	30	31
Préd. b1	NP	P	N	P	N	(P)	P	N	P
Réel b1	P	N	P	N	P	P	N	P	N
Préd b2	NP	N	P	N	P	N	P	N	P
Réel b2	NP	N	P	N	P	P	N	P	N

$$\frac{1}{9}$$

$$\frac{3}{9}$$

$$\frac{5}{9}$$

tab[i]	8	9	10	11	12	20	29	30	31
Préd. b1									
Réel b1									
Préd b2									
Réel b2									

$$\frac{4}{18}$$

$$\approx 22\%$$

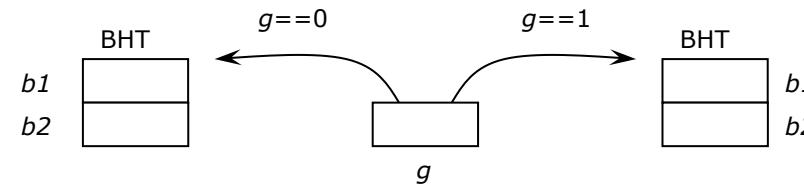
Solution prédicteur à historique global (q1)

tab[i]	8	9	10	11	12	20	29	30	31
Préd. b1	NP	P	NP	P	NP	P	P	NP	P
Réel b1	P	NP	P	NP	P	P	NP	P	NP
Préd b2	NP	NP	NP	P	NP	NP	P	NP	P
Réel b2	NP	NP	P	NP	NP	P	NP	P	NP

- Taux bonnes prédictions B1 : $1/9 = 11,1\%$
- Taux bonnes prédictions B2 : $3/9 = 33,3\%$
- Taux bonnes prédictions global : $4/18 = 22,2\%$

Exercice prédicteur à historique global (3)

- On suppose maintenant qu'on a un prédicteur à deux niveaux : un historique global g (1 bit) mémorise la direction réelle du dernier branchement exécuté. Ce bit est utilisé pour choisir parmi deux tables de compteurs 1-bit (initialisés à 0 : *non pris*), comme montré ci-dessous :



- Selon la valeur de g (initialisé à 0), une des deux tables est utilisée pour prédire le branchement suivant (c'est cette même table qui est mise à jour lorsque le branchement est résolu). Il n'y a pas de conflit dans les tables de compteurs entre $b1$ et $b2$. Chaque branchement met à jour son compteur avant que le branchement suivant ne soit exécuté.
- Indiquer, dans le tableau diapo suivante, comment les deux branchements sont prédis et finalement exécutés (pour chaque branchement, ne remplir qu'une seule des deux tables, en fonction de la valeur de g)
- Quel est le taux de bonnes prédictions pour chacun des deux branchements ?
- Quel est le taux global de bonnes prédictions ?
- Quel est le taux de bonnes prédictions de $b2$ lorsque $g==0$? Expliquer pourquoi.

Exercice prédicteur à historique global (4)

	tab[i]	8	9	10	11	12	20	29	30	31
Brouillon	Préd b1	NP	P	NP						
	Réel b1	P	NP	P						
	Préd b2		NP							
	Réel b2		NP							
g==1	Préd b1									
	Réel b1									
	Préd b2	NP								
	Réel b2	NP								
	tab[i]	8	9	10	11	12	20	29	30	31
Solution	Préd b1									
	Réel b1									
	Préd b2									
	Réel b2									
g==0	Préd b1									
	Réel b1									
	Préd b2									
	Réel b2									
g==1	Préd b1									
	Réel b1									
	Préd b2									
	Réel b2									

Solution prédicteur à historique global (q2)

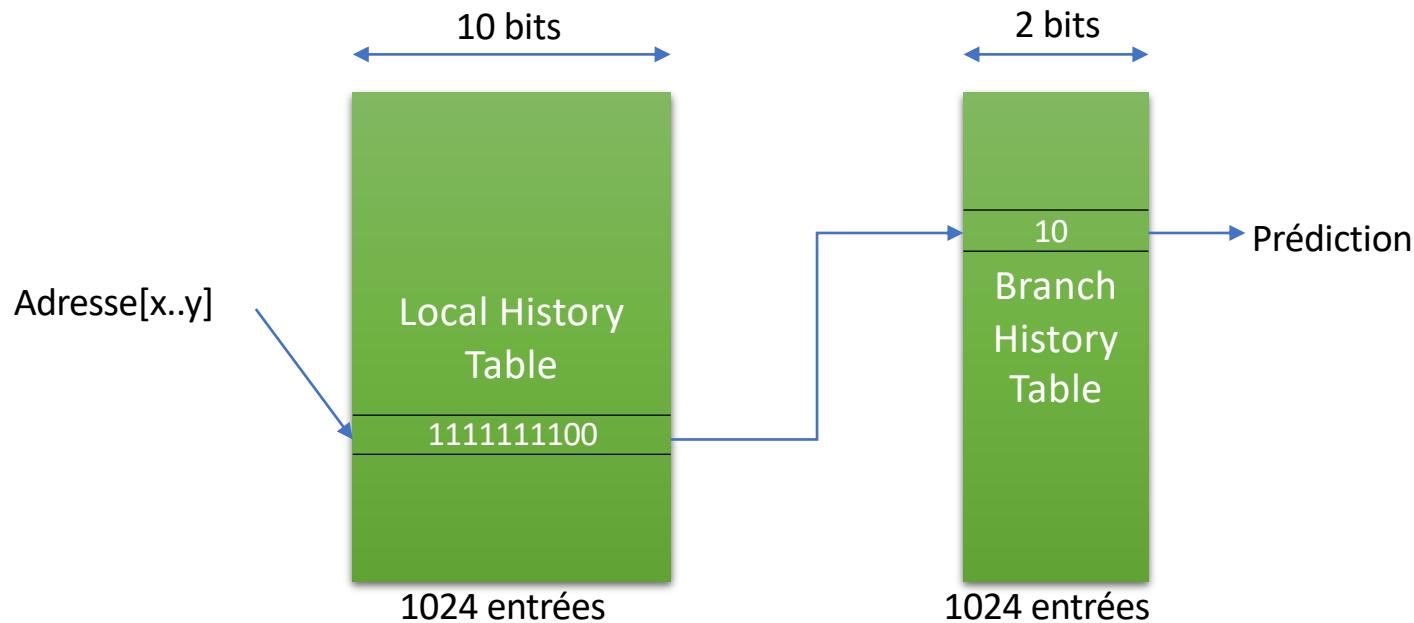
tab[i]	8	9	10	11	12	20	29	30	31
g==0	Préd b1	NP	P	NP		P	P	P	
	Réel b1	P	NP	P		P	P	P	
	Préd b2		NP		NP		NP		NP
	Réel b2		NP		NP		NP		NP
g==1	Préd b1				NP		NP		NP
	Réel b1				NP		NP		NP
	Préd b2	NP		NP		P	NP	P	
	Réel b2	NP		P		NP	P	P	

- b1 : 6/9 - b2 : 6/9 - Global : 6/9
- b2 pour g==0 : 100% parce que si b1 est non pris (nombre non divisible par 2) alors b2 est forcément non pris (nombre non divisible par 10)

Prédiction de la direction dynamique

Historique local

- Au lieu d'utiliser un historique représentant le comportement des derniers branchements, on utilise un historique représentant le comportement du branchement que l'on souhaite prédire. Il faut donc une table d'historiques qui va déterminer à quel compteur 2 bit on accède.



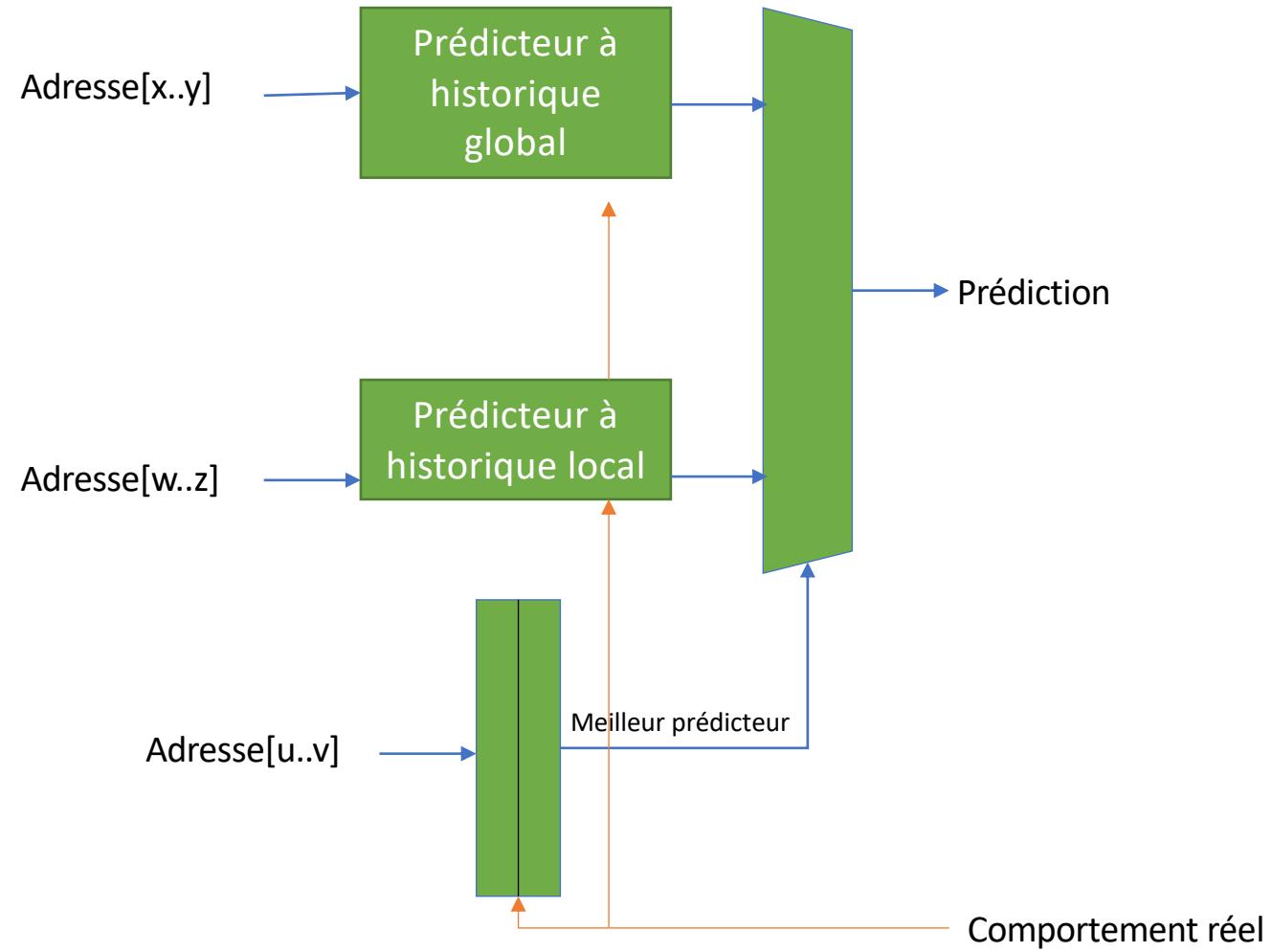
Prédiction de la direction dynamique

Table à accès direct

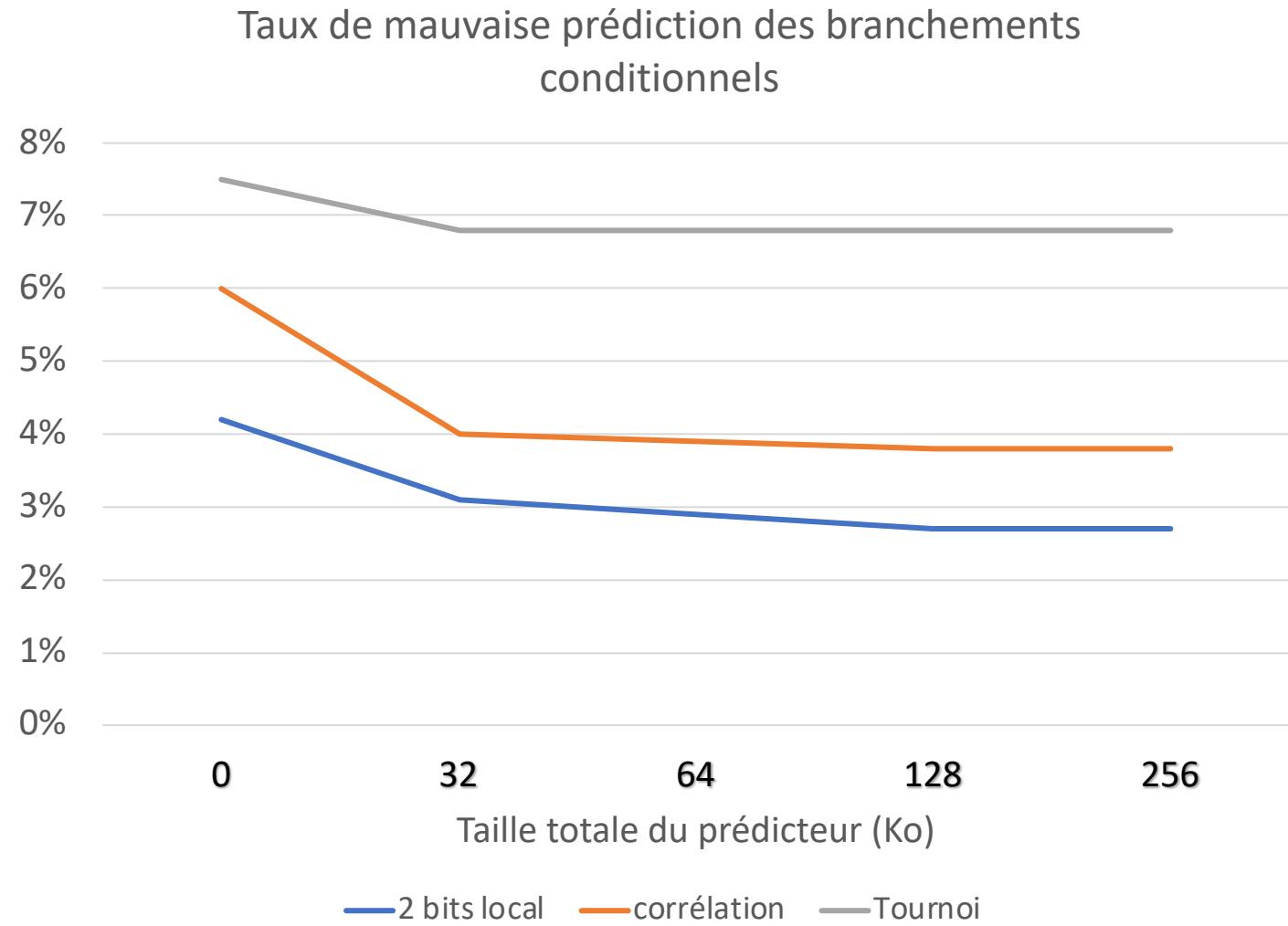
- La BHT (ou PHT) est à accès direct.
 - Le résultat étant 0 ou 1, ce n'est pas très gênant de partager des entrées entre plusieurs branchements. On a moins de 50% de chances de se tromper.
 - Il faut tout de même éviter de partager trop les entrées en adoptant une taille de BHT suffisamment grande (couramment 1024 voire 4096 entrées). On peut également essayer de trouver une fonction de hachage qui mélange mieux que le XOR mais elle doit rester simple pour que l'accès à la BHT se fasse en un cycle.
 - N'utiliser (mise à jour) la BHT que pour les branchements conditionnels. Les autres étant tout le temps pris, leur prédiction peut se faire différemment (voir plus loin la prédiction de l'adresse).
 - Des recherches continuent pour trouver de meilleures façons de prédire la direction des branchements conditionnels mais la qualité de prédiction étant déjà très bonne, il est difficile de l'améliorer sans ajouter un surcoût matériel important.

Prédiction de la direction dynamique

Prédicteur à tournoi

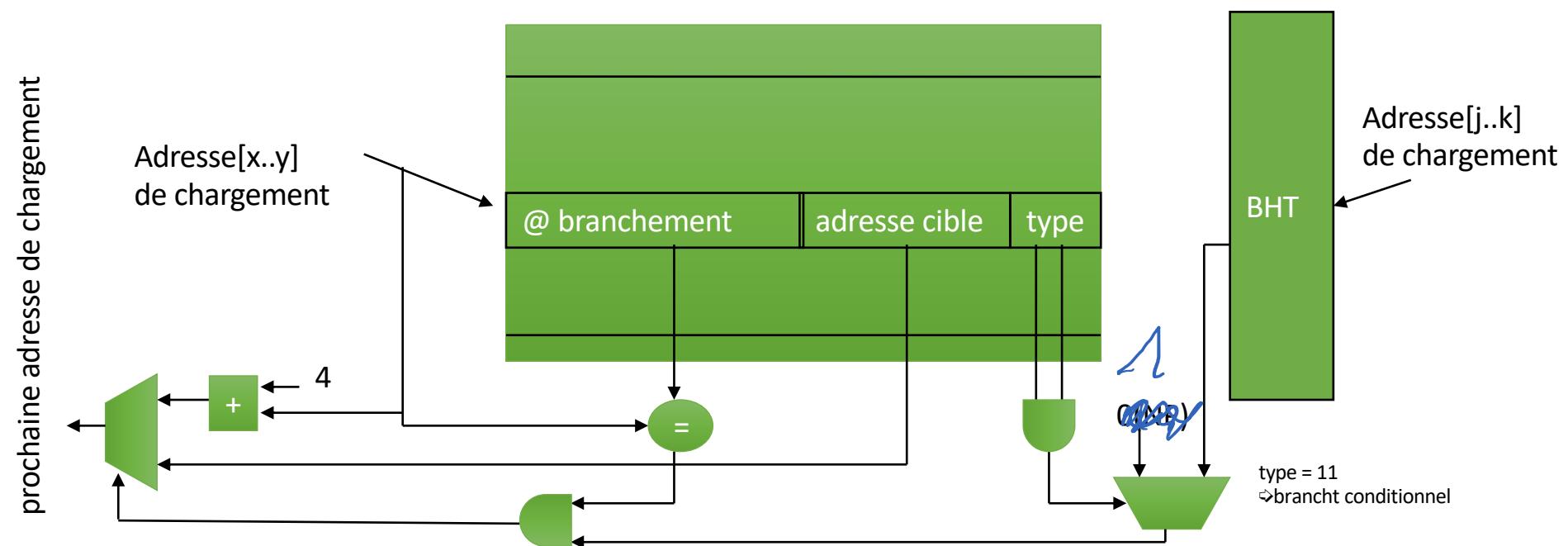


Performance prédicteurs de branchement



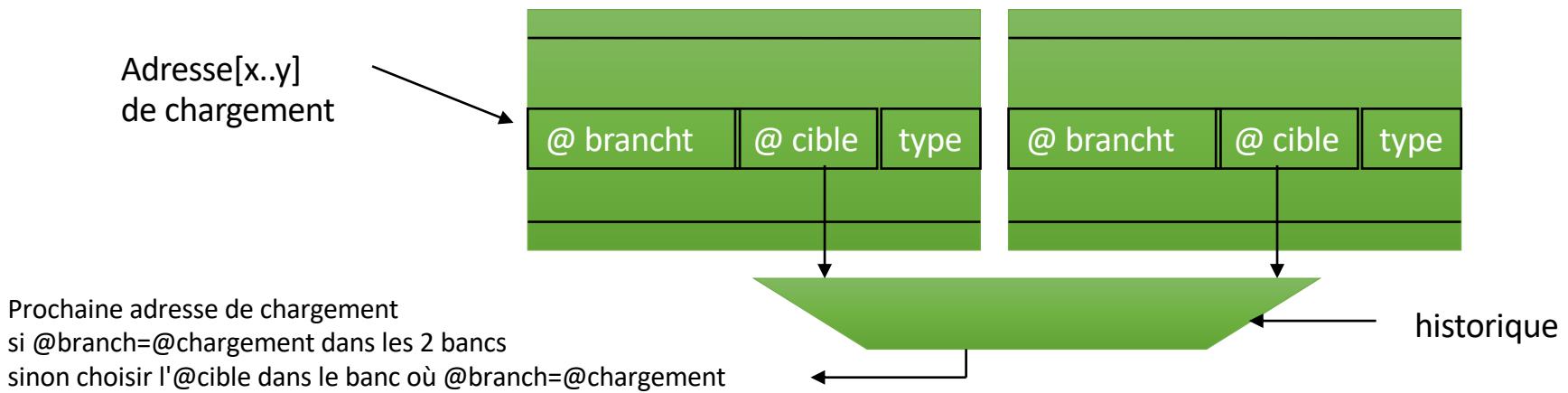
Prédiction de l'adresse

- Effectué avec une table à accès direct ou associatif par ensembles avec un tag (BTB pour Branch Target Buffer)
 - Tandis que l'on peut partager les entrées de la BHT car il n'y a que deux résultats possibles, prendre l'adresse cible d'un autre branchement conduit presque inévitablement à une erreur. Il faut donc s'assurer que l'adresse cible fournie correspond bien à l'adresse du branchement que l'on veut prédire.



Branchements indirects

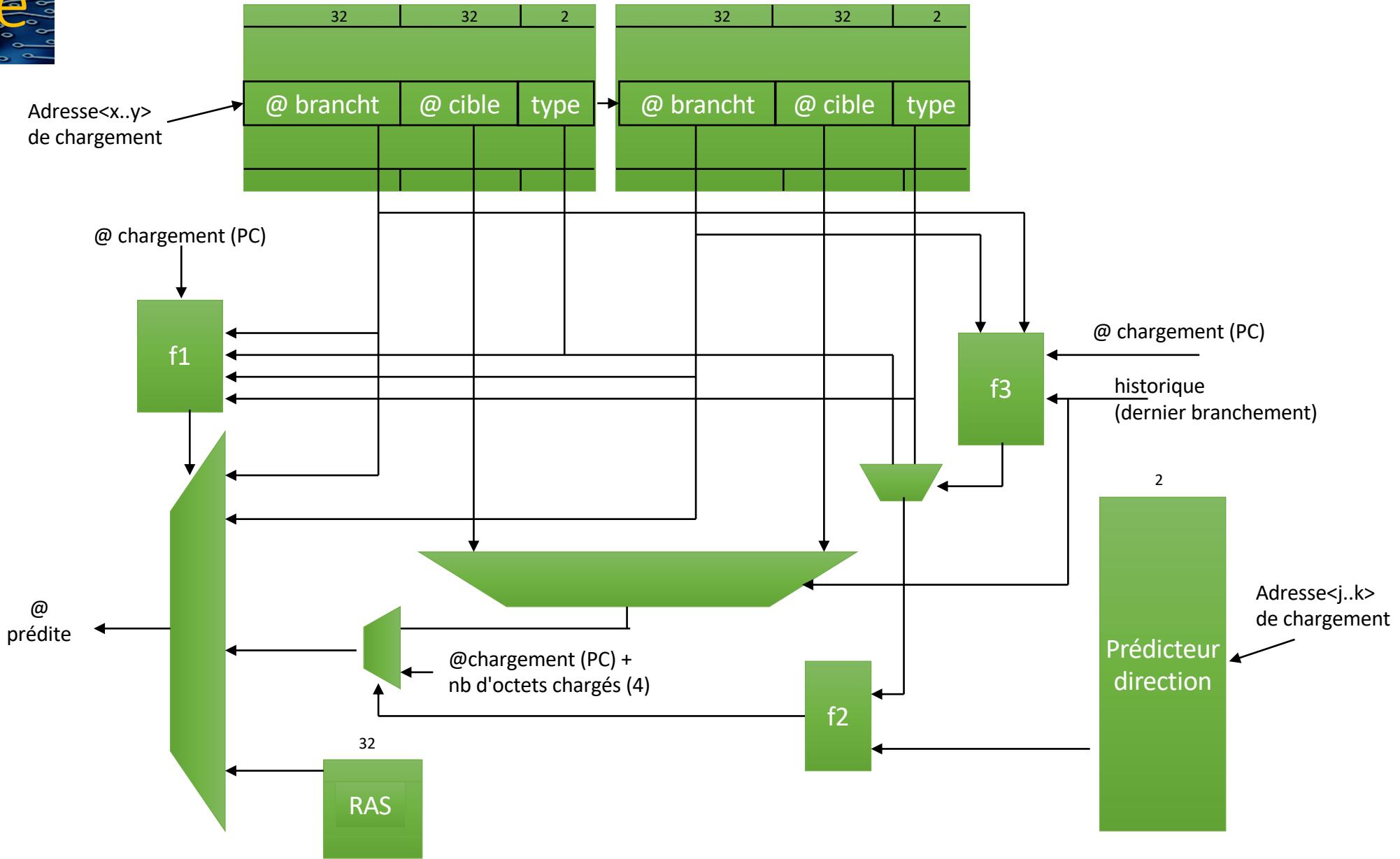
- Les branchements indirects trouvent l'adresse cible dans un registre.
 - Plusieurs adresses cibles possibles. Corrélée au chemin parcouru précédemment (direction des branchements précédents).
- BTB associative par ensembles
 - Le banc est choisi en fonction de l'historique global des branchements.



Sous-programmes

- Les appels de sous-programme se comportent comme des branchements inconditionnels.
- A l'exécution, les retours de sous-programme retournent à l'adresse du dernier BSR qui a été mémorisée en mémoire (pile).
- Pile interne au processeur
 - RAS: Return Address Stack
 - Exécution BSR: empiler dans la RAS adresse de retour (@BSR+4)
 - Chargement RTS: dépiler @ de chargement suivante de la RAS.

Vue globale



Exemples de prédicteurs de branchements réels

- Cortex A5
 - historique global, PHT 256 entrées, BTB de 4 entrées pour les branchements indirects, RAS de 4 entrées
- Cortex A9
 - historique global, PHT 1024 compteurs 2 bits, BTB 2*256 entrées, RAS de 8 entrées

Exercice speedup BTB

- Supposons que l'on a un pipeline assez profond et on a une BTB pour les branchements conditionnels uniquement.
- Supposons que la pénalité de mauvaise prédition est de 4 cycles et que la pénalité d'échec d'accès au BTB est de 3 cycles.
- Supposons un taux de succès de 90%, un taux de bonne prédition de 90%, et une fréquence de branchements de 15%.
- Speedup de ce processeur par rapport à un processeur n'ayant pas de BTB et une pénalité de 2 cycles par branchement ?
- Supposons que le CPI d'un programme sans pénalité de branchement est de 1.

Solution speedup BTB

$$speedup = \frac{CPI_{sansBTB}}{CPI_{avecBTB}} = \frac{CPI_{base} + Stalls_{sans\ BTB}}{CPI_{base} + Stalls_{avec\ BTB}}$$

$$Stall_{sansBTB} = PourcentageBranchements * PénalitéBranchement = 15\% * 2 = 0,3$$

Résultat accès BTB	Prédiction BTB	Fréquence (par instruction)	Pénalité (cycles)
Echec		15%*10%=1,5%	3
Succès	Correcte	15%*90%*90%=12,1%	0
Succès	Incorrecte	15%*90%*10%=1,3%	4

$$Stall_{avecBTB} = (1,5\% * 3) + (12,1\% * 0) + (1,3\% * 4) = 0,097$$

$$Speedup = \frac{1 + 0,3}{1 + 0,097} \approx 1,2$$

Pipeline et opérations multi-cycles



- M : multiplication entière et flottante
- A : additionneur flottant,
- Div rarement pipeliné

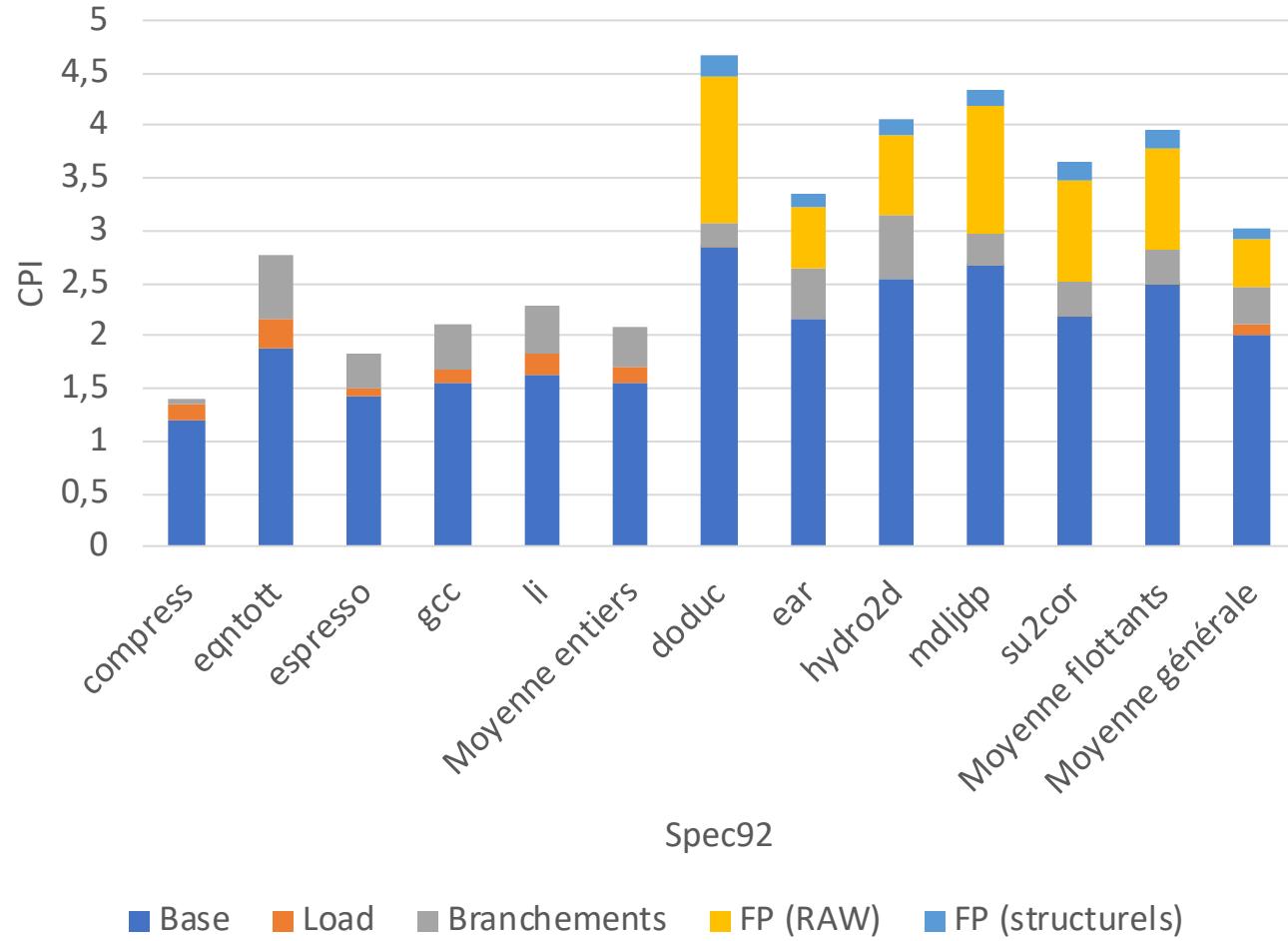
Unité fonctionnelle	Latence	Intervalle d'initiation
ALU entière (EX)	0	1
LDR	1	1
add flottant	3	1
mult	6	1
div	24	25

Pipeline et opérations multi-cycles (exemple aléas structurels)

	1	2	3	4	5	6	7	8	9	10	11
VMUL.F64 D0, D4, D6	LI	DI	M1	M2	M3	M4	M5	M6	M7	M	ER
ADD R0, R0, R2		LI	DI	EX	M	ER					
ADD R1, R0, R3			LI	DI	EX	M	ER				
VADD.F64 D2, D4, D6				LI	DI	A1	A2	A3	A4	M	ER
ADD R0, R0, R2					LI	DI	EX	M	ER		
ADD R1, R0, R3						LI	DI	EX	M	ER	
VLDR .64 D3, [R2]							LI	DI	EX	M	ER

- 3 instructions se terminent en même temps
 - Plus de ports sur le banc de registres mais nombre moyen = 1
 - Priorité à l'instruction la plus ancienne
- Les instructions se terminent dans le désordre
 - Pb pour les exceptions

Pipeline et opérations multi-cycles (performance)



Pipeline et opérations multi-cycles (ordonnancement)

- L'ordonnancement statique (par le compilateur) parvient difficilement à masquer les suspensions dues aux opérations à latency longue car l'ordonnancement statique ne doit pas violer les dépendances de contrôle (doit donc se faire à l'intérieur d'un bloc de base).
- En moyenne, 20% de branchements
 - Bloc de base = 5 instructions
- Solution logicielle : déroulage de boucles
- Solution matérielle : ordonnancement dynamique

Déroulage de boucle (1)

```
for (i=9999; i>=0; i--)  
    X[i] = X[i] +S;
```

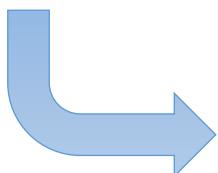
```
for:      VLDR.64  D0, [R1]  
          VADD.F64 D1, D0, D2  
          VSTR.64  D1, [R1]  
          SUBS     R1, R1, #8  
          BNE      for
```

```
for (i=9999; i>0; i-=2){  
    X[i] = X[i] +S;  
    X[i-1] = X[i-1] +S;  
}
```

Déroulage et renommage

for:

```
VLDR.64  D0, [R1]  
VADD.F64 D1, D0, D2  
VSTR.64  D1, [R1]  
VLDR.64  D3, [R1, #-8]  
VADD.F64 D4, D3, D2  
VSTR.64  D4, [R1, #-8]  
SUBS     R1, R1, #16  
BNE      for
```



for:

```
VLDR.64  D0, [R1]  
VLDR.64  D3, [R1, #-8]  
VADD.F64 D1, D0, D2  
VADD.F64 D4, D3, D2  
SUBS     R1, R1, #16  
VSTR.64  D1, [R1, #16]  
VSTR.64  D4, [R1, #8]  
BNE      for
```

Ordonnancement

Déroulage de boucle (2)

Hypothèses de latence

Instruction productrice	Instruction utilisatrice	Latence
opération UAL flottante	opération UAL flottante	3
opération UAL flottante	rangement double	2
chargement double	opération UAL flottante	1
chargement double	rangement double	0

for: VLDR.64 D0, [R1] 1
VADD.F64 D1, D0, D2 s3
VSTR.64 D1, [R1] ss6
SUBS R1, R1, #8 7
BNE for 8

10 000 * 9 = 80 000 cycles (idéal=50 000 cycles)

Ordonnancement simple

for: VLDR.64 D0, [R1] 1
VADD.F64 D1, D0, D2 s3
SUBS R1, R1, #8 4
VSTR.64 D1, [R1] s6
BNE for 7

10
I

10 000 * 7 = 70 000 cycles

Déroulage et réordonnement

for: VLDR.64 D0, [R1] 1
VLDR.64 D3, [R1, #-8] 2
VADD.F64 D1, D0, D2 3
VADD.F64 D4, D3, D2 4
SUBS R1, R1, #16 5
VSTR.64 D1, [R1, #16] 6
VSTR.64 D4, [R1, #8] 7
BNE for 8

5 000 * 8 = 40 000 cycles

Exercice 1 : déroulage de boucle

Hypothèses de latence

Instruction productrice	Instruction utilisatrice	Latence
opération UAL flottante	opération UAL flottante	3
opération UAL flottante	rangement double	2
chargement double	opération UAL flottante	1
chargement double	rangement double	0

- On considère la séquence de code suivante où X est un vecteur de nombres flottants 64 bits

```
for (i=N-2; i>0; i--)  
    x[i+1] = x[i]+A;
```

- En supposant que le vecteur est rangé à l'adresse 0, que le registre R1 est initialisé avec l'adresse de l'avant dernier élément du tableau ($X[N-2]$) et que D2 contient la valeur A, donner le code ARM de cette séquence.
- Avec les hypothèses de latence, calculer le nombre de cycles nécessaires pour exécuter une itération de la boucle. On suppose qu'il n'y a pas d'aléas structurels.
- Dérouler la boucle 4 fois, réordonner. Durée moyenne d'une itération ?

Exercice 1 : réponse q1

Exercice 1 : réponse q2

Exercice 1 : réponse q3

Exercice 1 : solution q1, q2

1

```
for: VLDR.64      D0,[R1]
      VADD.F64     D0, D0, D2
      VSTR.64      D0, [R1, #8]
      SUBS          R1, R1, #8
      BNE           for
```

2

```
for: VLDR.64  D0,[R1]      1
      VADD.F64  D0, D0, D2      3
      VSTR.64   D0, [R1, #8]    6
      SUBS      R1, R1, #8.    7
      BNE       for             8
```

susp.
susp.
susp.

Exercice 1 : solution q3

for:	VLDR.64	D0,[R1]	1
	VADD.F64	D0, D0, D2	s3
	VSTR.64	D0, [R1, #8]	ss6
	VLDR.64	D3,[R1, #-8]	7
	VADD.F64	D3, D3, D2	s9
	VSTR.64	D3, [R1, #0]	ss12
	VLDR.64	D4,[R1, #-16]	13
	VADD.F64	D4, D4, D2	s15
	VSTR.64	D4, [R1, #-8]	ss18
	VLDR.64	D5,[R1, #-24]	19
	VADD.F64	D5, D5, D2	s21
	VSTR.64	D5, [R1, #-16]	ss24
	SUBS	R1, R1, #32	25
	BNE	for	26

- 26 cycles pour 4 itérations
soit **6,5 cycles par itération**

for:	VLDR.64	D0,[R1]	
	VLDR.64	D3,[R1, #-8]	
	VLDR.64	D4,[R1, #-16]	
	VLDR.64	D5,[R1, #-24]	
	VADD.F64	D0, D0, D2	
	VADD.F64	D3, D3, D2	
	VADD.F64	D4, D4, D2	
	VADD.F64	D5, D5, D2	
	VSTR.64	D0, [R1, #8]	
	VSTR.64	D3, [R1, #0]	
	VSTR.64	D4, [R1, #-8]	
	VSTR.64	D5, [R1, #-16]	
	SUBS	R1, R1, #32	
	BNE	for	

- 14 cycles pour 4 itérations
soit **3,5 cycles par itération**

Exercice 2 : déroulage de boucle

Hypothèses de latence

Instruction productrice	Instruction utilisatrice	Latence
opération UAL flottante	opération UAL flottante	3
opération UAL flottante	rangement double	2
chargement double	opération UAL flottante	1
chargement double	rangement double	0

- On considère la séquence de code suivante où X et Y sont des vecteur de nombres flottants 64 bits

```
for (i=N; i>0; i--)  
    Y[i] = A*X[i]+ Y[i];
```

- En supposant que le vecteur X est rangé à l'adresse 0, que les registres R1 et R2 sont initialisés avec l'adresse du dernier élément des tableaux X et Y et que D4 contient la valeur A, donner le code ARM de cette boucle (sans exploiter le branchement retardé).
- Avec les hypothèses de latence, calculer le nombre de cycles nécessaire pour exécuter une itération de la boucle. On suppose qu'il n'y a pas d'aléas structurels.
- Réordonner le code pour minimiser les suspensions.
- Dérouler la boucle 2 fois, réordonner. Durée moyenne d'une itération ?

Exercice 2 : réponse q1

Exercice 2 : réponse q2



Exercice 2 : solution q1, q2, q3

```
bcl:    VLDR.64 D0, [R1]
        VMUL.F64 D0, D0, D4
        VLDR.64 D1, [R2]
        VADD.F64 D1, D0, D1
        VSTR.64 D1, [R2]
        SUBS    R1, R1, #8
        SUB     R2, R2, #8
        BNE    bcl
```

```
bcl:    VLDR.64 D0, [R1]      1   s
        VMUL.F64 D0, D0, D4      3
        VLDR.64 D1, [R2]      4   ss
        VADD.F64 D1, D0, D1      7   ss
        VSTR.64 D1, [R2]      10
        SUBS    R1, R1, #8      11
        SUB     R2, R2, #8      12
        BNE    bcl      13
```

```
bcl:    VLDR.64 D0, [R1]      1
        SUBS    R1, R1, #8      2
        VMUL.F64 D0, D0, D4      3
        VLDR.64 D1, [R2]      4
        VADD.F64 D1, D0, D1      7
        SUB     R2, R2, #8      8
        VSTR.64 D1, [R2, #8]    s10
        BNE    bcl      11
```

Exercice 2 : réponse q4

Exercice 2 : solution q4

```
bcl:    VLDR.64 D0, [R1]
        VMUL.F64 D0, D0, D4
        VLDR.64 D1, [R2]
        VADD.F64 D1, D0, D1
        VSTR.64 D1, [R2]
        VLDR.64 D2, [R1, #-8]
        VMUL.F64 D2, D2, D4
        VLDR.64 D3, [R2, #-8]
        VADD.F64 D3, D2, D3
        VSTR.64 D3, [R2, #-8]
        SUBS    R1, R1, #16
        SUB     R2, R2, #16
        BNE    R1, bcl
```

s

ss

ss

s

ss

ss

```
bcl:    VLDR.64 D0, [R1]
        VLDR.64 D1, [R2]
        VMUL.F64 D0, D0, D4
        VLDR.64 D2, [R1, #-8]
        VLDR.64 D3, [R2, #-8]
        VMUL.F64 D2, D2, D4
        VADD.F64 D1, D0, D1
        SUBS    R1, R1, #16
        SUB     R2, R2, #16
        VADD.F64 D3, D2, D3
        VSTR.64 D1, [R2, #16]
        VSTR.64 D3, [R2, #8]
        BNE    R1, bcl
```

23 cycles pour 2 itérations soit
11,5 cycles par itération

13 cycles pour 2 itérations
soit 6,5 cycles par itération

Exercice 3 : déroulage de boucle

Hypothèses de latence

Instruction productrice	Instruction utilisatrice	Latence
opération UAL flottante	opération UAL flottante	3
opération UAL flottante	rangement double	2
chargement double	opération UAL flottante	1
chargement double	rangement double	0

- On considère la séquence de code suivante où A, B et C sont des vecteurs de nombres flottants 64 bits

```
for (i=0; i<N-1; i++){ A[i+1] = A[i]+ C[i]; B[i+1] = B[i] + A[i+1]; }
```

- En supposant que les registres R1, R2 et R3 sont initialisés avec les adresses des tableaux A, B et C et que l'adresse du dernier élément est dans R4, donner le code ARM de cette boucle (sans exploiter le branchement retardé).
- Avec les hypothèses de latence, calculer le nombre de cycles nécessaire pour exécuter une itération de la boucle. On suppose qu'il n'y a pas d'aléas structurels.
- Modifier le code pour supprimer les suspensions et exploiter le branchement retardé.
- Dérouler la boucle 2 fois, réordonner. Durée moyenne d'une itération ?

Exercice 3 : réponse q1

Exercice 3 : réponse q2

Exercice 3 : réponse q3

Exercice 3 : solution q1, q2, q3

bcl:	VLDR.64	D0, [R1]	1
	VLDR.64	D2, [R3]	2s
	VADD.F64	D0, D0, D2	4ss
	VSTR.64	D0, [R1, #8]	7
	VLDR.64	D1, [R2]	8s
	VADD.F64	D1, D0, D1	10ss
	VSTR.64	D1, [R2, #8]	13
	ADD	R1, R1, #8	14
	ADD	R2, R2, #8	15
	ADD	R3, R3, #8	16
	CMP	R1, R4	17
	BNE	bcl	18

bcl:	VLDR.64	D0, [R1]
	VLDR.64	D2, [R3]
	VADD.F64	D1, [R2]
	VADD.F64	D0, D0, D2
	ADD	R1, R1, #8
	ADD	R2, R2, #8
	VSTR.64	D0, [R1,
		#8]
	VADD.F64	D1, D0, D1
	ADD	R3, R3, #8
	CMP	R1, R4
	BNE	R1, bcl
	VSTR.64	D1, [R2]

12 cycles par tour

Exercice 3 : réponse q4

Exercice 3 : solution q4

bcl:	VLDR.64	D0, [R1]	1	bcl:	VLDR.64	D0, [R1]
	VLDR.64	D2, [R3]	2s		VLDR.64	D2, [R3]
	VADD.F64	D0, D0, D2	4ss		VLDR.64	D1, [R2]
	VSTR.64	D0, [R1, #8]	7		VADD.F64	D0, D0, D2
	VLDR.64	D1, [R2]	8s		VLDR.64	D5, [R3, #8]
	VADD.F64	D1, D0, D1	10ss		VLDR.64	D4, [R2, #8]
	VSTR.64	D1, [R2, #8]	13		ADD	R1, R1, #16
	VLDR.64	D3, [R1, #8]	14		VADD.F64	D1, D0, D1
	VLDR.64	D5, [R3, #8]	15s		VSTR.64	D0, [R1, #-8]
	VADD.F64	D3, D3, D5	17ss		VLDR.64	D3, [R1, #-8] @ MOV.64 D3, D0
	VSTR.64	D3, [R1, #16]	20		ADD	R2, R2, #16
	VLDR.64	D4, [R2, #8]	21s		VADD.F64	D3, D3, D5
	VADD.F64	D4, D3, D4	22ss		VSTR.64	D1, [R2, #-8]
	VSTR.64	D4, [R2, #16]	26		ADD	R3, R3, #16
ADD	R1, R1, #16	27			VSTR.64	D3, [R1]
ADD	R2, R2, #16	28			VADD.F64	D4, D3, D4
ADD	R3, R3, #16	29			CMP	R1, R4
CMP	R1, R4	30			VSTR.64	D4, [R2]
BNE	bcl	31			BNE	bcl

Ordonnancement dynamique

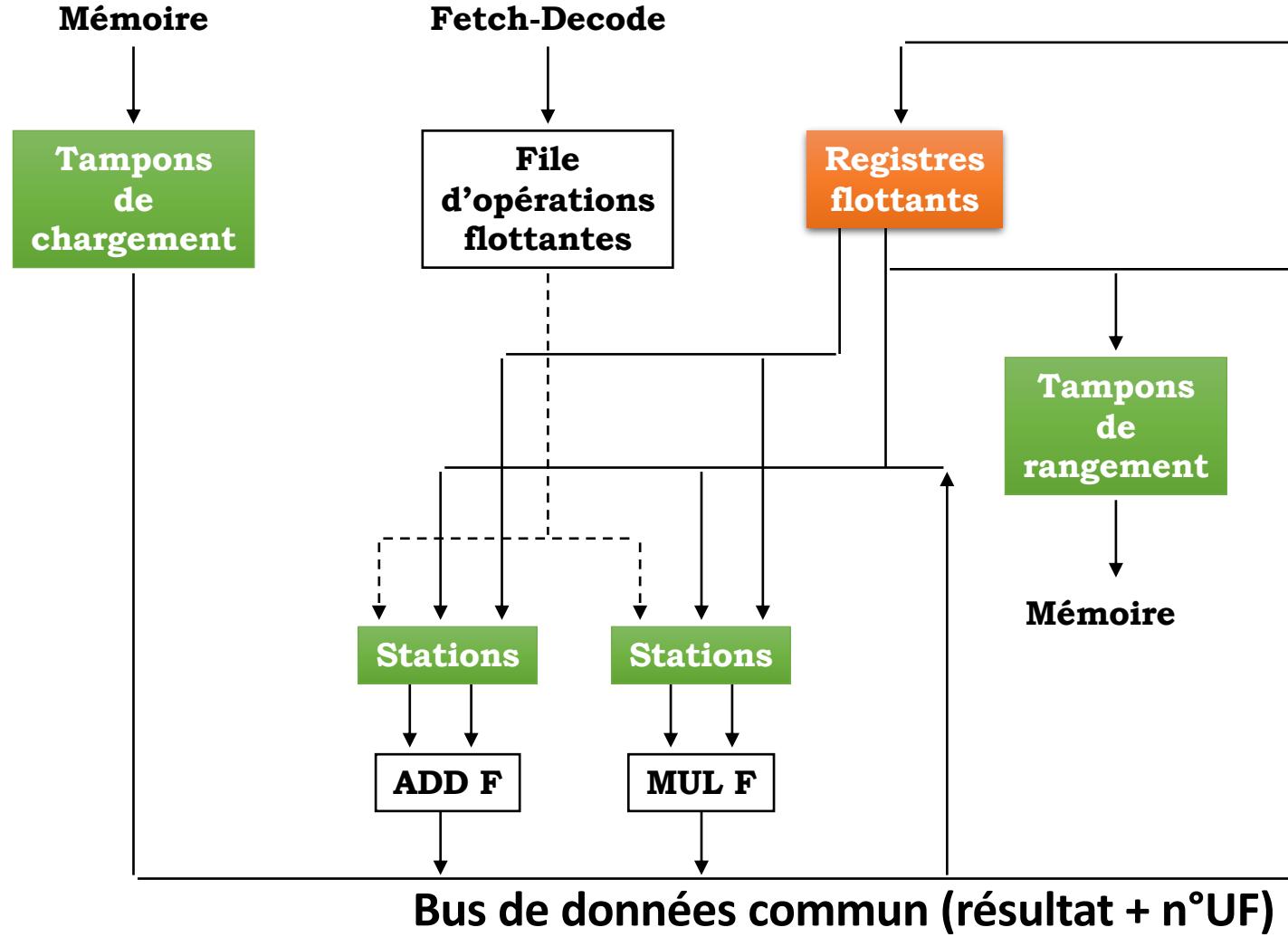
- Idée :
 - Le pipeline simple bloque une instruction **et les suivantes** dès qu'il y a un aléa RAW
 - Pourquoi bloquer une instruction suivante alors qu'elle n'a pas d'aléa RAW ?
 - Dépendances de données à travers la mémoire pas toujours connues du compilateur (adressages indirects) mais connue du processeur si les registres ne sont pas en cours de calcul
 - Indépendance compilateur/matériel
 - Division de l'étage de décodage en 2 étapes
 - décodage et détection des aléas structurels
 - attente d'absence d'aléas puis lecture des opérandes
- 2 catégories de solutions
 - Tableau de marques
 - Algorithme de Tomasulo

L'ALGORITHME DE TOMASULO

L'algorithme de Tomasulo

- Conçu pour l'IBM 360/91 en 1968
- But : Haute performance avec compilateurs spéciaux
- Différences entre l'algo de Tomasulo & le tableau de marques
 - Le contrôle & les tampons sont distribués auprès des UFs ; appelés “stations de réservation”
 - Les numéros de registres dans les instructions sont remplacés par des pointeurs vers les stations de réservation
 - Les registres sont renommés matériellement pour éviter les EAL et les EAE alors que le tableau de marques ne fait pas de renommage
 - Des bus de données communs diffusent les résultats à toutes les UFs
 - Les Load et les Store sont traités de la même façon que les UFs
- A la base du fonctionnement des processeurs d'aujourd'hui

Tomasulo – Structures matérielles



Constitution d'une station de réservation

Op	SRs1	SRs2	V s1	V s2	Q s1	Q s2	Occupé
----	------	------	------	------	------	------	--------

- Op Opération à effectuer dans l'UF (par exemple, +)
- SRs1, SRs2 Stations de réservation produisant les registres source
- Vs1, Vs2 Valeurs des opérandes sources
- Qs1, Qs2 Drapeaux indiquant que Vs1, Vs2 sont prêts
- Occupé indique que la station et l'UF sont occupées

Constitution du banc de registres



Etat : Indique si une UF est en train de calculer le registre et laquelle

Les 3 étages de l'algorithme de Tomasulo

1. Lancement — Lire l'instruction depuis la file d'opérations flottantes

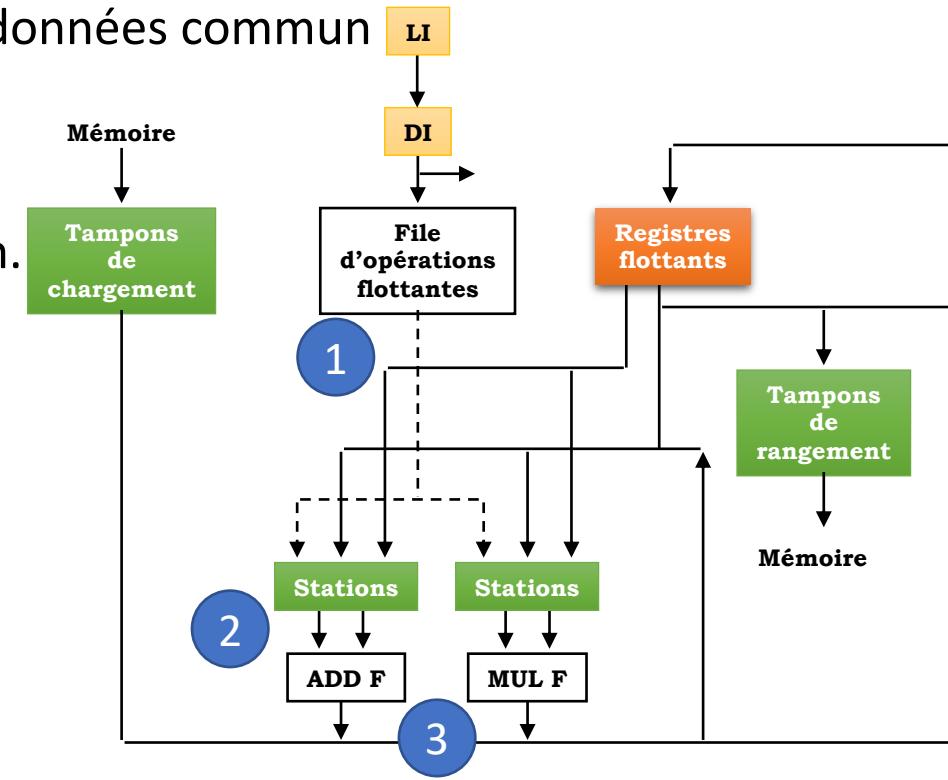
- Si une station de réservation est vide, envoi des opérandes à la station s'ils sont disponibles.

2. Exécution — opération sur les opérandes (EX)

- Quand les 2 opérandes sont prêts, exécution;
s'ils ne sont pas prêts, attendre le résultat sur le bus de données commun

3. Ecriture du résultat — fin de l'exécution (WB)

- Ecriture sur le bus de données commun
vers toutes les stations et libérer la station de réservation.



Tomasulo : les étapes de l'algorithme

Etat de l'instruction	Attente jusqu'à	Actions à effectuer
Lancement	Station ou tampon disponible	<p>Si ($\text{Reg}[S1].Qi \neq 0$) $\text{SR}[r].Qs1 \leftarrow \text{Reg}[S1].Qi;$ Sinon $\{ \text{SR}[r].Vs1 \leftarrow \text{Reg}[S1]; \text{SR}[r].Qs1 \leftarrow 0; \}$ Si ($\text{Reg}[S2].Qi \neq 0$) $\{ \text{SR}[r].Qs2 \leftarrow \text{Reg}[S2].Qi \}$ Sinon $\{ \text{SR}[r].Vs2 \leftarrow \text{Reg}[S2]; \text{SR}[r].Qs2 \leftarrow 0; \}$ $\text{SR}[r].occupé \leftarrow \text{oui};$ $\text{Registre}[D].Qi \leftarrow r;$</p>
Exécution	$(\text{SR}[r].Qs1=0)$ et $\text{SR}[r].Qs2=0$	Aucune. Les opérandes sont dans Vs1 et Vs2.
Ecriture résultat	Exécution terminée et bus résultat disponible.	$\forall x, (\text{si } \text{Reg}[x].Qi = r)$ $\{ Fx \leftarrow \text{résultat}; \text{Reg}[x].Qi \leftarrow 0; \}$ $\forall x, (\text{si } \text{SR}[x].Qs1 = r)$ $\{ \text{SR}[x].Vs1 \leftarrow \text{résultat}; \text{SR}[x].Qs1 \leftarrow 0; \}$ $\forall x, (\text{si } \text{SR}[x].Qs2 = r)$ $\{ \text{SR}[x].Vs2 \leftarrow \text{résultat}; \text{SR}[x].Qs2 \leftarrow 0; \}$ $\forall x, (\text{si } \text{Range}[x].Qs1 = r)$ $\{ \text{Range}[x].V \leftarrow \text{résultat}; \text{Range}[x].Qs1 \leftarrow 0; \}$ $\text{SR}[r].occupé \leftarrow \text{non} ;$

Tomasulo - Exemple

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat	Tampon de chargement et de rangement	
Instruction		S1	S2				Occupé	Adresse
VLDR.64	D0	32	R2				Load1	Non
VLDR.64	D1	36	R3				Load2	Non
VMUL.F64	D2	D1	D4				Load3	Non
VSUB.F64	D3	D0	D1				Store1	Non
VDIV.F64	D5	D2	D0				Store2	Non
VADD.F64	D0	D3	D1				Store3	Non

Stations de réservation							
Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Non					
	Mul2	Non					

Cycle	Etat registres									
		D0	D1	D2	D3	D4	D5	D6	...	D15
	Qi									

Hypothèses:
l'addition dure 2 cycles
la multiplication 10 cycles
la division 40 cycles

Tomasulo – Cycle 1

Etat instruction				Lancée	Exécution finie	Ecriture résultat	Tampon de chargement et de rangement	
Instruction		S1	S2				Occupé	Adresse
VLDR.64	D0	32	R2	1			Load1	Oui R2+32
VLDR.64	D1	36	R3				Load2	Non
VMUL.F64	D2	D1	D4				Load3	Non
VSUB.F64	D3	D0	D1				Store1	Non
VDIV.F64	D5	D2	D0				Store2	Non
VADD.F64	D0	D3	D1				Store3	Non

Stations de réservation							
Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Non					
	Mul2	Non					

Cycle	Etat registres									
		D0	D1	D2	D3	D4	D5	D6	...	D15
1	Qi	Load1								

Hypothèses:

- l'addition dure 2 cycles*
- la multiplication 10 cycles*
- la division 40 cycles*

Tomasulo – Cycle 2

Etat instruction				Lancée	Exécution finie	Ecriture résultat	Tampon de chargement et de rangement	
Instruction		S1	S2				Occupé	Adresse
VLDR.64	D0	32	R2	1			Load1	Oui R2+32
VLDR.64	D1	36	R3	2			Load2	Oui R3+36
VMUL.F64	D2	D1	D4				Load3	Non
VSUB.F64	D3	D0	D1				Store1	Non
VDIV.F64	D5	D2	D0				Store2	Non
VADD.F64	D0	D3	D1				Store3	Non

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Non					
	Mul2	Non					

Cycle

2

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Load1	Load2							

Hypothèses:

- l'addition dure 2 cycles*
- la multiplication 10 cycles*
- la division 40 cycles*

Tomasulo – Cycle 3

Etat instruction				Lancée	Exécution finie	Ecriture résultat	Tampon de chargement et de rangement	
Instruction		S1	S2				Occupé	Adresse
VLDR.64	D0	32	R2	1	3		Load1	Oui R2+32
VLDR.64	D1	36	R3	2			Load2	Oui R3+36
VMUL.F64	D2	D1	D4	3			Load3	Non
VSUB.F64	D3	D0	D1				Store1	Non
VDIV.F64	D5	D2	D0				Store2	Non
VADD.F64	D0	D3	D1				Store3	Non

Stations de réservation								
Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2	
0	Add1	Non						
	Add2	Non						
	Add3	Non						
	Mul1	Oui	VMUL.F64		(D4)	Load2		
	Mul2	Non						

Cycle	Etat registres									
3		D0	D1	D2	D3	D4	D5	D6	...	D15
	Qi	Load1	Load2	Mul1						

Hypothèses:
*l'addition dure 2 cycles
la multiplication 10 cycles
la division 40 cycles*

Tomasulo – Cycle 4

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2		
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4		
VDIV.F64	D5	D2	D0			
VADD.F64	D0	D3	D1			

Tampon de chargement et de rangement	
Occupé	Adresse
Non	R2+32
Oui	R3+36
Non	
Non	
Non	
Non	

Qs1 V

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
0	Add1	Oui	VSUB.F64		M(R2+32)			Load2
0	Add2	Non						
0	Add3	Non						
0	Mul1	Oui	VMUL.F64			(D4)	Load2	
0	Mul2	Non						

Cycle

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Lead1	Load2	Mul1	Add1					

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10
 cycles
 la division 40 cycles*

Tomasulo – Cycle 5

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4		
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1			

Tampon de chargement et de rangement	
Occupé	Adresse
Non	
Oui	R3+36
Non	
Non	
Non	
Non	

Qs1 V

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
0	Add1	Oui	VSUB.F64	M(R2+32)				Load2
0	Add2	Non						
0	Add3	Non						
0	Mul1	Oui	VMUL.F64		(D4)	Load2		
0	Mul2	Oui	VDIV.F64		M(R2+32)	Mul1		

Cycle

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi		Load2	Mul1	Add1		Mul2			

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10
 cycles
 la division 40 cycles*

Tomasulo – Cycle 6

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4		
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6		

Tampon de chargement et de rangement	
Occupé	Adresse
Non	
Non	R3+36
Non	
Non	
Non	
Non	

Qs1 V

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
2	Add1	Oui	VSUB.F64	M(R2+32)	M(R3+36)			
0	Add2	Oui	VADD.F64		M(R3+36)	Add1		
	Add3	Non						
10	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)			
0	Mul2	Oui	VDIV.F64		M(R2+32)	Mul1		

Cycle Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Add2	Lead2	Mul1	Add1		Mul2			

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10
 cycles
 la division 40 cycles*

Tomasulo – Cycle 7

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4		
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6		

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
Cycle	Add1	Oui	VSUB.F64		M(R2+32)	M(R3+36)		
	Add2	Oui	VADD.F64			M(R3+36)	Add1	
	Add3	Non						
	Mul1	Oui	VMUL.F64		M(R3+36)	(D4)		
	Mul2	Oui	VDIV.F64			M(R2+32)	Mul1	

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10 cycles
 la division 40 cycles*

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Add2		Mul1	Add1		Mul2			

Tomasulo – Cycle 8

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4	8	
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6		

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
0	Add1	Oui	VSUB.F64	M(R2+32)	M(R3+36)			
0	Add2	Oui	VADD.F64		M(R3+36)	Add1		
	Add3	Non						
8	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)			
0	Mul2	Oui	VDIV.F64		M(R2+32)	Mul1		

Cycle

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Add2		Mul1	Add1		Mul2			

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10 cycles
 la division 40 cycles*

Tomasulo – Cycle 9

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6		

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
0	Add1	Non	VSUB.F64	M(R2+32)	M(R3+36)	M(R3+36)	Add1	
	Add2	Oui	VADD.F64	M(...)-M(...)	M(R3+36)			
	Add3	Non						
7	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)	M(R2+32)	Mul1	
	Mul2	Oui	VDIV.F64					

Cycle

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Add2		Mul1	Add1		Mul2			

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10 cycles
 la division 40 cycles*

Tomasulo – Cycle 10

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6		

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
2	Add1	Non						
	Add2	Oui	VADD.F64	M(...)-M(...)	M(R3+36)			
	Add3	Non						
6	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)			
	Mul2	Oui	VDIV.F64		M(R2+32)	Mul1		

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Add2		Mul1			Mul2			

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10 cycles
 la division 40 cycles*

Tomasulo – Cycle 11

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6		

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
1	Add1	Non						
	Add2	Oui	VADD.F64	M(...)-M(...)	M(R3+36)			
	Add3	Non						
5	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)			
	Mul2	Oui	VDIV.F64		M(R2+32)	Mul1		

Cycle

11

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Add2		Mul1			Mul2			

Hypothèses:

- l'addition dure 2 cycles*
- la multiplication 10 cycles*
- la division 40 cycles*

Tomasulo – Cycle 12

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6	12	

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
0	Add1	Non						
	Add2	Oui	VADD.F64	M(...)-M(...)	M(R3+36)			
	Add3	Non						
4	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)			
	Mul2	Oui	VDIV.F64		M(R2+32)	Mul1		

Cycle

12

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Add2		Mul1			Mul2			

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10 cycles
 la division 40 cycles*

Tomasulo – Cycle 13

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6	12	13

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2	Qs1	Qs2
3	Add1	Non						
	Add2	Non	VADD.F64	M(...)	M(...)	M(R3+36)		
	Add3	Non						
	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)			
	Mul2	Oui	VDIV.F64			M(R2+32)	Mul1	

Cycle

13

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi	Add2			Mul1			Mul2		

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10 cycles
 la division 40 cycles*

Tomasulo – Cycle 14

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6	12	13

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Cycle	Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
		Add1	Non					
2	2	Add2	Non					
	0	Add3	Non					
0	2	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)		
	0	Mul2	Oui	VDIV.F64		M(R2+32)	Mul1	

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10 cycles
 la division 40 cycles*

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi			Mul1			Mul2			

Tomasulo – Cycle 15

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3		
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6	12	13

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2		Qs1	Qs2
1	Add1	Non							
	Add2	Non							
	Add3	Non							
	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)				
	Mul2	Oui	VDIV.F64			M(R2+32)	Mul1		

Cycle

15

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi				Mul1			Mul2		

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10
 cycles
 la division 40 cycles*

Tomasulo – Cycle 16

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3	16	
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6	12	13

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2		Qs1	Qs2
0	Add1	Non							
	Add2	Non							
	Add3	Non							
	Mul1	Oui	VMUL.F64	M(R3+36)	(D4)				
	Mul2	Oui	VDIV.F64			M(R2+32)	Mul1		

Cycle

16

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi			Mul1			Mul2			

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10 cycles
 la division 40 cycles*

Tomasulo – Cycle 17

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3	16	17
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6	12	13

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2		Qs1	Qs2
0	Add1	Non							
	Add2	Non							
	Add3	Non							
	Mul1	Non	VMUL.F64	M(R3+36)	(D4)				
	Mul2	Oui	VDIV.F64	M*D2	M(R2+32)				

Cycle

17	D0	D1	D2	D3	D4	D5	D6	...	D15
	Qi			Mul1			Mul2		

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10 cycles
 la division 40 cycles*

Tomasulo – Cycle 18

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	<i>Ecriture résultat</i>
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3	16	17
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6	12	13

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2		Qs1	Qs2
	Add1	Non							
	Add2	Non							
	Add3	Non							
	Mul1	Non							
40	Mul2	Oui	VDIV.F64		M*D2	M(R2+32)			

Cycle

18

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi						Mul2			

Hypothèses:

*l'addition dure 2 cycles
la multiplication 10 cycles
la division 40 cycles*

Tomasulo – Cycle 57

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	<i>Ecriture résultat</i>
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3	16	17
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5		
VADD.F64	D0	D3	D1	6	12	13

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2		Qs1	Qs2
	Add1	Non							
	Add2	Non							
	Add3	Non							
	Mul1	Non							
1	Mul2	Oui	VDIV.F64		M*D2	M(R2+32)			

Cycle

57

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi						Mul2			

Hypothèses:

*l'addition dure 2 cycles
la multiplication 10 cycles
la division 40 cycles*

Tomasulo – Cycle 58

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3	16	17
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5	58	
VADD.F64	D0	D3	D1	6	12	13

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2		Qs1	Qs2
0	Add1	Non							
	Add2	Non							
	Add3	Non							
	Mul1	Non							
	Mul2	Oui	VDIV.F64		M*D2	M(R2+32)			

Cycle

58

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi						Mul2			

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10
 cycles
 la division 40 cycles*

Tomasulo – Cycle 59

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	32	R2	1	3	4
VLDR.64	D1	36	R3	2	5	6
VMUL.F64	D2	D1	D4	3	16	17
VSUB.F64	D3	D0	D1	4	8	9
VDIV.F64	D5	D2	D0	5	58	59
VADD.F64	D0	D3	D1	6	12	13

Tampon de chargement et de rangement	
Occupé	Adresse
Non	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2		Qs1	Qs2
0	Add1	Non							
	Add2	Non							
	Add3	Non							
	Mul1	Non							
	Mul2	Non	VDIV.F64	M*D2	M(R2+32)				

Cycle

59

Etat registres

	D0	D1	D2	D3	D4	D5	D6	...	D15
Qi						Mul2			

Hypothèses:

*l'addition dure 2 cycles
 la multiplication 10
 cycles
 la division 40 cycles*

Tomasulo – Exemple de boucle

```
Loop: VLDR.64      D0,[R1]  
        VMUL.F64    D4,D0,D2  
        VSTR.F64    D4,[R1]  
        SUBS         R1,R1,#8  
        BNE          Loop
```

- Hypothèses
 - La multiplication dure 4 cycles
 - Le premier load provoque un échec dans le cache (8 cycles)

Exemple de boucle Cycle 1

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1		
VMUL.F64	D4	D0	D2			
VSTR.F64	D4	0	R1			
VLDR.64	D0	0	R1			
VMUL.F64	D4	D0	D2			
VSTR.F64	D4	0	R1			

Tampon de chargement et de rangement

	Occupé	Adresse		
Load1	Oui	80		
Load2	Non			
Load3	Non		Qs1	V
Store1	Non			
Store2	Non			
Store3	Non			

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Non					
	Mul2	Non					

Cycle

Etat registres

1	R1	D0	D1	D2	D3	D4	D5	D6	...	D15
	80	Qi	Load1							

Exemple de boucle Cycle 2

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat	Tampon de chargement et de rangement	
Instruction		S1	S2				Occupé	Adresse
VLDR.64	D0	0	R1	1			Load1	Oui 80
VMUL.F64	D4	D0	D2	2			Load2	Non
VSTR.F64	D4	0	R1				Load3	Non
VLDR.64	D0	0	R1				Store1	Non
VMUL.F64	D4	D0	D2				Store2	Non
VSTR.F64	D4	0	R1				Store3	Non

Stations de réservation								
Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2	
	Add1	Non						
	Add2	Non						
	Add3	Non						
	Mul1	Oui	VMUL.F64		R(D2)	Load1		
	Mul2	Non						

Cycle	Etat registres																			
	R1		D0		D1		D2		D3		D4		D5		D6		...		D15	
	80	Qi	Load1								Mul1									

Exemple de boucle Cycle 3

Etat instruction

Instruction		S1	S2	Lancée	Exécution finie	Ecriture résultat
VLDR.64	D0	0	R1	1		
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1			
VMUL.F64	D4	D0	D2			
VSTR.F64	D4	0	R1			

Tampon de chargement et de rangement

Occupé	Adresse		
Load1	Oui	80	Qs1 V
Load2	Non		
Load3	Non		
Store1	Oui	80	
Store2	Non		
Store3	Non		

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64		R(D2)	Load1	
	Mul2	Non					

Cycle

Etat registres

3	R1		D0	D1	D2	D3	D4	D5	D6	...	D15
	80	Qi	Load1					Mul1			

Exemple de boucle Cycle 4

Etat instruction				Lancée	Exécution finie	Ecriture résultat	Tampon de chargement et de rangement	
Instruction		S1	S2				Occupé	Adresse
VLDR.64	D0	0	R1	1			Load1	Oui 80
VMUL.F64	D4	D0	D2	2			Load2	Non
VSTR.F64	D4	0	R1	3			Load3	Non
VLDR.64	D0	0	R1				Store1	Qs1 V
VMUL.F64	D4	D0	D2				Store2	Oui 80 Mul1
VSTR.F64	D4	0	R1				Store3	Non
								Non

Stations de réservation							
Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64		R(D2)	Load1	
	Mul2	Non					

Cycle	Etat registres									
	R1		D0		D1		D2		D3	
	4	72	Qi	Load1					...	D15

Exemple de boucle Cycle 5

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1		
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1			
VMUL.F64	D4	D0	D2			
VSTR.F64	D4	0	R1			

Tampon de chargement et de rangement

Occupé Adresse

Load1	Oui	80		
Load2	Non			
Load3	Non		Qs1	V
Store1	Oui	80	Mul1	
Store2	Non			
Store3	Non			

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64		R(D2)	Load1	
	Mul2	Non					

Cycle

Etat registres

5	R1		D0	D1	D2	D3	D4	D5	D6	...	D15
	72	Qi	Load1				Mul1				

Exemple de boucle Cycle 6

Etat instruction				Lancée	Exécution finie	Ecriture résultat	Tampon de chargement et de rangement	
Instruction		S1	S2				Occupé	Adresse
VLDR.64	D0	0	R1	1			Load1	Oui 80
VMUL.F64	D4	D0	D2	2			Load2	Oui 72
VSTR.F64	D4	0	R1	3			Load3	Non
VLDR.64	D0	0	R1	6			Store1	Qs1 V
VMUL.F64	D4	D0	D2				Store2	Oui 80 Mul1
VSTR.F64	D4	0	R1				Store3	Non
								Non

Stations de réservation							
Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64		R(D2)	Load1	
	Mul2	Non					

Cycle	Etat registres										
6	R1		D0	D1	D2	D3	D4	D5	D6	...	D15
	72	Qi	Load2				Mul1				

Exemple de boucle Cycle 7

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1		
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6		
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1			

Tampon de chargement et de rangement

Occupé Adresse

Load1	Oui	80		
Load2	Oui	72		
Load3	Non		Qs1	V
Store1	Oui	80	Mul1	
Store2	Non			
Store3	Non			

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64		R(D2)	Load1	
	Mul2	Oui	VMUL.F64		R(D2)	Load2	

Cycle

Etat registres

7

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
72	Qi	Load2				Mul2				

Exemple de boucle Cycle 8

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1		
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6		
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

Occupé Adresse

Load1	Oui	80		
Load2	Oui	72		
Load3	Non		Qs1	V
Store1	Oui	80	Mul1	
Store2	Oui	72	Mul2	
Store3	Non			

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64		R(D2)	Load1	
	Mul2	Oui	VMUL.F64		R(D2)	Load2	

Cycle

Etat registres

8	R1		D0	D1	D2	D3	D4	D5	D6	...	D15
	72	Qi	Load2				Mul2				

Exemple de boucle Cycle 9

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6		
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

Occupé Adresse

Load1	Oui	80		
Load2	Oui	72		
Load3	Non		Qs1	V
Store1	Oui	80	Mul1	
Store2	Oui	72	Mul2	
Store3	Non			

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64		R(D2)	Load1	
	Mul2	Oui	VMUL.F64		R(D2)	Load2	

Cycle

Etat registres

9	R1		D0	D1	D2	D3	D4	D5	D6	...	D15
	64	Qi	Load2				Mul2				

Exemple de boucle Cycle 10

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

Occupé Adresse

Load1	Non		
Load2	Oui	72	
Load3	Non		Qs1 V
Store1	Oui	80	Mul1
Store2	Oui	72	Mul2
Store3	Non		

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64	Mem(80)	R(D2)	Load1	
	Mul2	Oui	VMUL.F64		R(D2)	Load2	

Cycle

Etat registres

10	R1		D0	D1	D2	D3	D4	D5	D6	...	D15
	64	Qi	Load2					Mul2			

Exemple de boucle Cycle 11

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

Occupé Adresse

Load1	Non			
Load2	Non	72		
Load3	Oui	64	Qs1	V
Store1	Oui	80	Mul1	
Store2	Oui	72	Mul2	
Store3	Non			

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2		Qs1		Qs2	
	Add1	Non									
	Add2	Non									
	Add3	Non									
	Mul1	Oui	VMUL.F64	Mem(80)	R(D2)						
	Mul2	Oui	VMUL.F64	Mem(72)	R(D2)	Load2					

Cycle

Etat registres

11	R1		D0	D1	D2	D3	D4	D5	D6	...	D15
	64	Qi	Load2					Mul2			

Exemple de boucle Cycle 12

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

Occupé Adresse

Load1	Non			
Load2	Non			
Load3	Oui	64	Qs1	V
Store1	Oui	80	Mul1	
Store2	Oui	72	Mul2	
Store3	Non			

Stations de réservation

Temps	Nom	Occupé	Op	Vs1		Vs2		Qs1		Qs2	
	Add1	Non									
	Add2	Non									
	Add3	Non									
	Mul1	Oui	VMUL.F64	Mem(80)	R(D2)						
	Mul2	Oui	VMUL.F64	Mem(72)	R(D2)						

Cycle

Etat registres

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
64	Qi						Mul2			

Exemple de boucle Cycle 13

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Etat instruction				<i>Lancée</i>	<i>Exécution finie</i>	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2		
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

Occupé Adresse

Load1	Non		
Load2	Non		
Load3	Oui	64	Qs1 V
Store1	Oui	80	Mul1
Store2	Oui	72	Mul2
Store3	Non		

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
1	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64	Mem(80)	R(D2)		
	Mul2	Oui	VMUL.F64	Mem(72)	R(D2)		

Cycle

Etat registres

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
64	Qi						Mul2			

Exemple de boucle Cycle 14

Etat instruction

Instruction		S1	S2	Lancée	Exécution finie	Ecriture résultat
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2	14	
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7		
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

Occupé	Adresse		
Non			
Non			
Oui	64	Qs1	V
Oui	80	Mul1	
Oui	72	Mul2	
Non			

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
0	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64	Mem(80)	R(D2)		
	Mul2	Oui	VMUL.F64	Mem(72)	R(D2)		

Cycle

14

Etat registres

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
64	Qi					Mul2				

Exemple de boucle Cycle 15

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2	14	15
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7	15	
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

	Occupé	Adresse		
Load1	Non			
Load2	Non			
Load3	Oui	64	Qs1	V
Store1	Oui	80	Mul1	Mem(80)*D2
Store2	Oui	72	Mul2	
Store3	Non			

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
0	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Non	VMUL.F64	Mem(80)	R(D2)		
	Mul2	Oui	VMUL.F64	Mem(72)	R(D2)		

Cycle

Etat registres											
15	R1		D0	D1	D2	D3	D4	D5	D6	...	D15
	64	Qi						Mul2			

Exemple de boucle Cycle 16

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2	14	15
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7	15	16
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

Occupé Adresse

Load1	Non			
Load2	Non			
Load3	Oui	64	Qs1	V
Store1	Oui	80	Mul1	Mem(80)*D2
Store2	Oui	72	Mul2	Mem(72)*D2
Store3	Non			

Stations de réservation

Temps	0	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
		Add1	Non					
		Add2	Non					
		Add3	Non					
		Mul1	Oui	VMUL.F64		R(D2)	Load3	
		Mul2	Non	VMUL.F64	Mem(72)	R(D2)		

Cycle

16

Etat registres

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
64	Qi					Mul1				

Exemple de boucle Cycle 17

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2	14	15
VSTR.F64	D4	0	R1	3		
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7	15	16
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

	Occupé	Adresse		
Load1	Non			
Load2	Non			
Load3	Oui	64	Qs1	V
Store1	Oui	80	Mul1	Mem(80)*D2
Store2	Oui	72	Mul2	Mem(72)*D2
Store3	Non	64	Mul1	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
0	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64		R(D2)	Load3	
	Mul2	Non					

Cycle

17

Etat registres

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
64	Qi					Mul1				

Exemple de boucle Cycle 18

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2	14	15
VSTR.F64	D4	0	R1	3	18	
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7	15	16
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

	Occupé	Adresse		
Load1	Non			
Load2	Non			
Load3	Oui	64	Qs1	V
Store1	Oui	80	Mul1	Mem(80)*D2
Store2	Oui	72	Mul2	Mem(72)*D2
Store3	Oui	64	Mul1	

Stations de réservation

Cycle 18	0	Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
			Add1	Non					
			Add2	Non					
			Add3	Non					
			Mul1	Oui	VMUL.F64		R(D2)	Load3	
			Mul2	Non					

Etat registres

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
56	Qi					Mul1				

Exemple de boucle Cycle 19

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2	14	15
VSTR.F64	D4	0	R1	3	18	19
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7	15	16
VSTR.F64	D4	0	R1	8		

Tampon de chargement et de rangement

Occupé Adresse

Load1	Non			
Load2	Non			
Load3	Oui	64	Qs1	V
Store1	Non	80	Mul1	Mem(80)*D2
Store2	Oui	72	Mul2	Mem(72)*D2
Store3	Oui	64	Mul1	

Stations de réservation

Cycle 19	Temps 0	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
		Add1	Non					
		Add2	Non					
		Add3	Non					
		Mul1	Oui	VMUL.F64		R(D2)	Load3	
		Mul2	Non					

Etat registres

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
56	Qi					Mul1				

Exemple de boucle Cycle 20

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2	14	15
VSTR.F64	D4	0	R1	3	18	19
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7	15	16
VSTR.F64	D4	0	R1	8	20	

Tampon de chargement et de rangement

	Occupé	Adresse		
Load1	Non			
Load2	Non			
Load3	Oui	64	Qs1	V
Store1	Non			
Store2	Oui	72	Mul2	Mem(72)*D2
Store3	Oui	64	Mul1	

Stations de réservation

Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
0	Add1	Non					
	Add2	Non					
	Add3	Non					
	Mul1	Oui	VMUL.F64		R(D2)	Load3	
	Mul2	Non					

Cycle

20

Etat registres

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
56	Qi					Mul1				

Exemple de boucle Cycle 21

Etat instruction				Lancée	Exécution finie	Ecriture résultat
Instruction		S1	S2			
VLDR.64	D0	0	R1	1	9	10
VMUL.F64	D4	D0	D2	2	14	15
VSTR.F64	D4	0	R1	3	18	19
VLDR.64	D0	0	R1	6	10	11
VMUL.F64	D4	D0	D2	7	15	16
VSTR.F64	D4	0	R1	8	20	21

Tampon de chargement et de rangement

	Occupé	Adresse		
Load1	Non			
Load2	Non			
Load3	Oui	64	Qs1	V
Store1	Non			
Store2	Non	72	Mul2	Mem(72)*D2
Store3	Oui	64	Mul1	

Stations de réservation

Cycle 21	0	Temps	Nom	Occupé	Op	Vs1	Vs2	Qs1	Qs2
			Add1	Non					
			Add2	Non					
			Add3	Non					
			Mul1	Oui	VMUL.F64		R(D2)	Load3	
			Mul2	Non					

Etat registres

R1		D0	D1	D2	D3	D4	D5	D6	...	D15
56	Qi							Mul1		

Exercice 1 - Tomasulo

- On considère une boucle dont le code est le suivant :

```
bcl:    vldr.64      D1, [R1], #-8 @ D1 <- mem(R1) ; R1 <- R1-8  
        vmul.F64     D2, D1, D0  
        vadd.F64     D4, D2, D3  
        vstr.64      D4, [R2], #-8  
        cmp           R1, #0  
        bne          bcl
```

1. Déterminer les dépendances entre les instructions
2. Analyser l'exécution de deux itérations successives en ignorant le temps d'exécution des instructions entières

Exercice – Tomasulo

Hypothèses latence

:

Add: 3 cycles,
Mult: 6cycles,
Div: 24 cycles,
load et store: 1
cycle

Instruction	Lancée	Exécution	Ecriture résultat
VLDR.64 D1, [R1], #-8	1	2-3	4
VMUL.F64 D2, D1, D0			
VADD.F64 D4, D2, D3			
VSTR.64 D4, [R2], #-8			
VLDR.64 D1, [R1], #-8			
VMUL.F64 D2, D1, D0			
VADD.F64 D4, D2, D3			
VSTR.64 D4, [R2], #-8			

N°	Nom	Occupée	Op	Valeur S1 / @	Valeur S2 / D	SR S1	SR S2
1	Add1						
2	Add2						
3	Mul1						
4	Mul2						
5	Div						
6	LD1	O ₁ N ₄	VLDR.64	R1	0		
7	LD2						

Etat des registres

	D0	D1	D2	D3	D4	D5	D6
Nom UF		LD1 ₁₋₄					

Solution – Tomasulo

Hypothèses latence

:

Add: 3 cycles,
Mult: 6cycles,
Div: 24 cycles,
load et store: 1
cycle

Etat des instructions

Instruction	Lancée	Exécution	Ecriture résultat
VLDR.64 D1, [R1], #-8	1	2-3	4
VMUL.F64 D2, D1, D0	2	5-11	12
VADD.F64 D4, D2, D3	3	13-16	17
VSTR.64 D4, [R2], #-8	4	18-19	20
VLDR.64 D1, [R1], #-8	5	6-7	8
VMUL.F64 D2, D1, D0	6	9-15	16
VADD.F64 D4, D2, D3	7	17-20	21
VSTR.64 D4, [R2], #-8	21	22-23	24

N°	Nom	Occupée	Op	Valeur S1 / @	Valeur S2 / D	SR S1	SR S2
1	Add1	O ₃ N ₁₇	VADD.F64	D2	D3	Mul1 ₃₋₁₂	
2	Add2	O ₇ N ₂₁	VADD.F64	D2	D3	Mul2 ₆₋₁₆	
3	Mul1	O ₂ N ₁₂	VMUL.F64	D1	D0	LD1 ₂₋₄	
4	Mul2	O ₆ N ₁₆	VMUL.F64	D1	D0	LD1 ₆₋₈	
5	Div						
6	LD1	O ₁ N ₄ O ₅ N ₈	VLDR.64	R1-8			
7	LD2						
8	ST	O ₄ N ₂₀ O ₂₁ N ₂₄	VSTR.64	R2-8	D4	Add1 ₃₋₁₂ Add2 ₂₁₋₂₁	

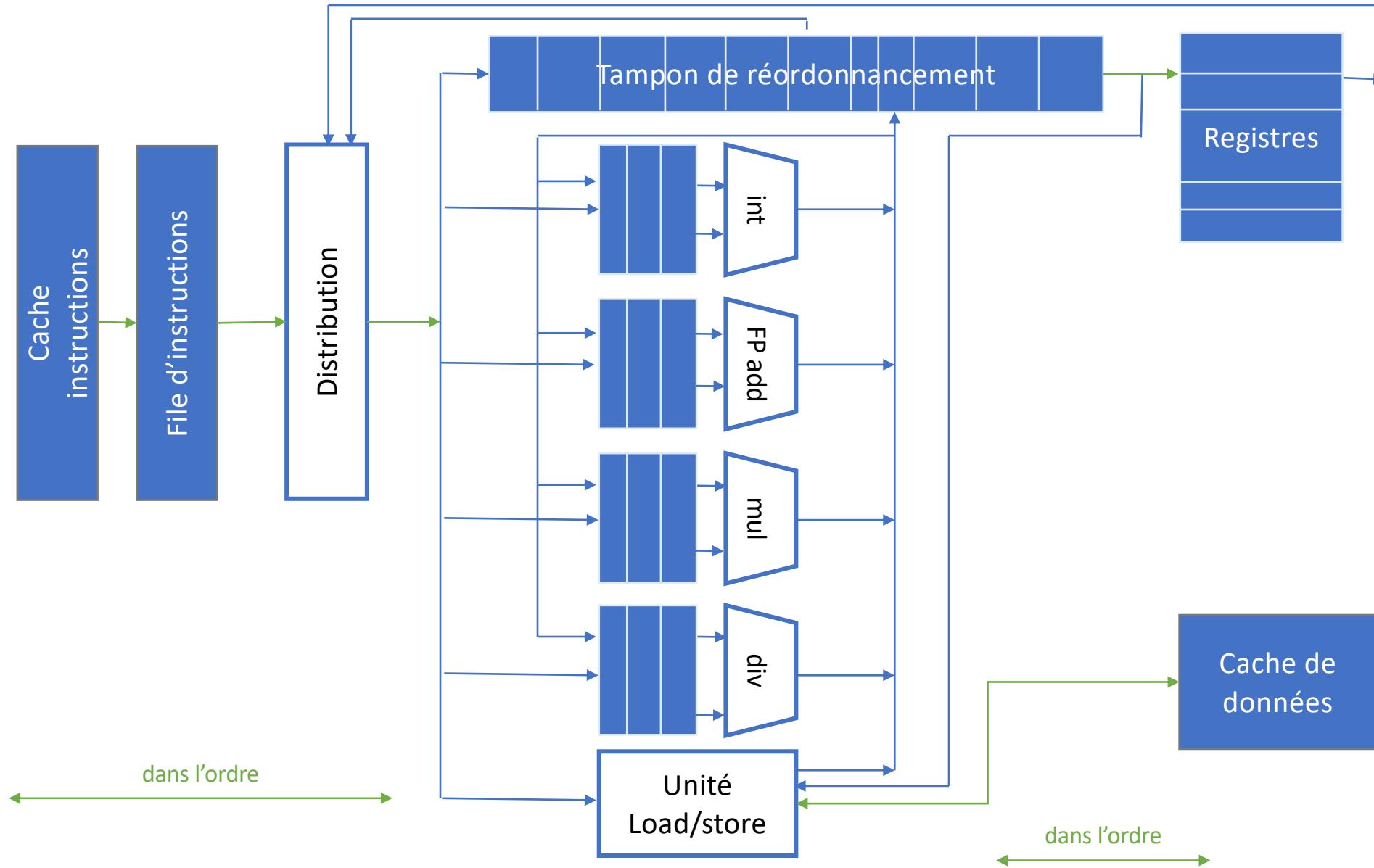
Etat des registres

	D0	D1	D2	D3	D4	D5	D6
Nom UF		LD1 ₁₋₄ LD1 ₅₋₈	Mul1 ₂₋₁₂ Mul1 ₆₋₁₆		Add1 ₃₋₁₇ Add1 ₇₋₂₁		

Conclusion Tomasulo

- Empêche les aléas de ressources de registre grâce au renommage
- Evite donc les aléas EAL et EAE
- Bypass automatique des registres
- Réalise un déroulage de boucle dans le matériel
- Non limité aux blocs de base si prédition de branchement mais problème avec les écritures dans le désordre des registres
 - Une instruction située derrière un branchement prédit peut modifier un registre alors que le branchement a été mal prédit...
- Reste aussi le problème des exceptions précises
 - Une instruction peut modifier un registre alors qu'une instruction ultérieure peut lever une exception

Ordonnancement dynamique et spéculatif



La spéculation matérielle

Les étapes de l'algorithme (non superscalaire)

Etat de l'instruction	Attente jusqu'à	Actions à effectuer
Lancement	Station et ROB disponibles	Si ($\text{Reg}[S1].\text{occupé}$) $\{\text{SR}[r].Qs1 \Leftarrow \text{Reg}[S1].\text{ROB}\}$ Sinon $\{\text{SR}[r].Vs1 \Leftarrow \text{Reg}[S1]; \text{SR}[r].Qs1 \Leftarrow 0\}$ Si ($\text{Reg}[S2].\text{occupé}$) $\{\text{SR}[r].Qs2 \Leftarrow \text{Reg}[S2].\text{ROB}\}$ Sinon $\{\text{SR}[r].Vs2 \Leftarrow \text{Reg}[S2]; \text{SR}[r].Qs2 \Leftarrow 0\}$ $\text{SR}[r].\text{occupé} \Leftarrow \text{oui}; \text{SR}[r].\text{dest} \Leftarrow b;$ Registre[D].ROB=b; Registre[D].occupé=oui; ROB[b].dest=D;
Exécution	$(\text{SR}[r].Qs1=0)$ et $(\text{SR}[r].Qs2=0)$	Aucune. Les opérandes sont dans Vs1 et Vs2.
Ecriture résultat	Exécution terminée de r avec numéro du ROB b et bus résultat disponible.	$\forall x, (\text{si } \text{SR}[x].Qs1=b) \{\text{SR}[x].Vs1 \Leftarrow \text{résultat}, \text{SR}[x].Qs1 \Leftarrow 0\}$ $\forall x, (\text{si } \text{SR}[x].Qs2=b) \{\text{SR}[x].Vs2 \Leftarrow \text{résultat}, \text{SR}[x].Qs2 \Leftarrow 0\}$ $\text{SR}[r].\text{occupé} \Leftarrow \text{non} ; \text{ROB}[b].\text{valeur} \Leftarrow \text{résultat}$
Garantie	Instruction en tête du ROB (entrée t) et écriture du résultat effectuée.	Si ($\text{ROB}[t].\text{instruction} = \text{branchement}$) et ($\text{branchement mal prédit}$) {vider le ROB ; redémarrer la lecture des instructions} Sinon Si $\text{ROB}[t].\text{instruction} = \text{rangement}$ $\{\text{Mem}(\text{ROB}[t].\text{dest}) \Leftarrow \text{ROB}[t].\text{valeur}\}$ sinon Registre[ROB[t].dest]=ROB[t].valeur $\text{Reg}[\text{ROB}[t]].\text{occupé} \Leftarrow \text{non} ;$ $\text{ROB}[t].\text{occupé} \Leftarrow \text{non} ;$