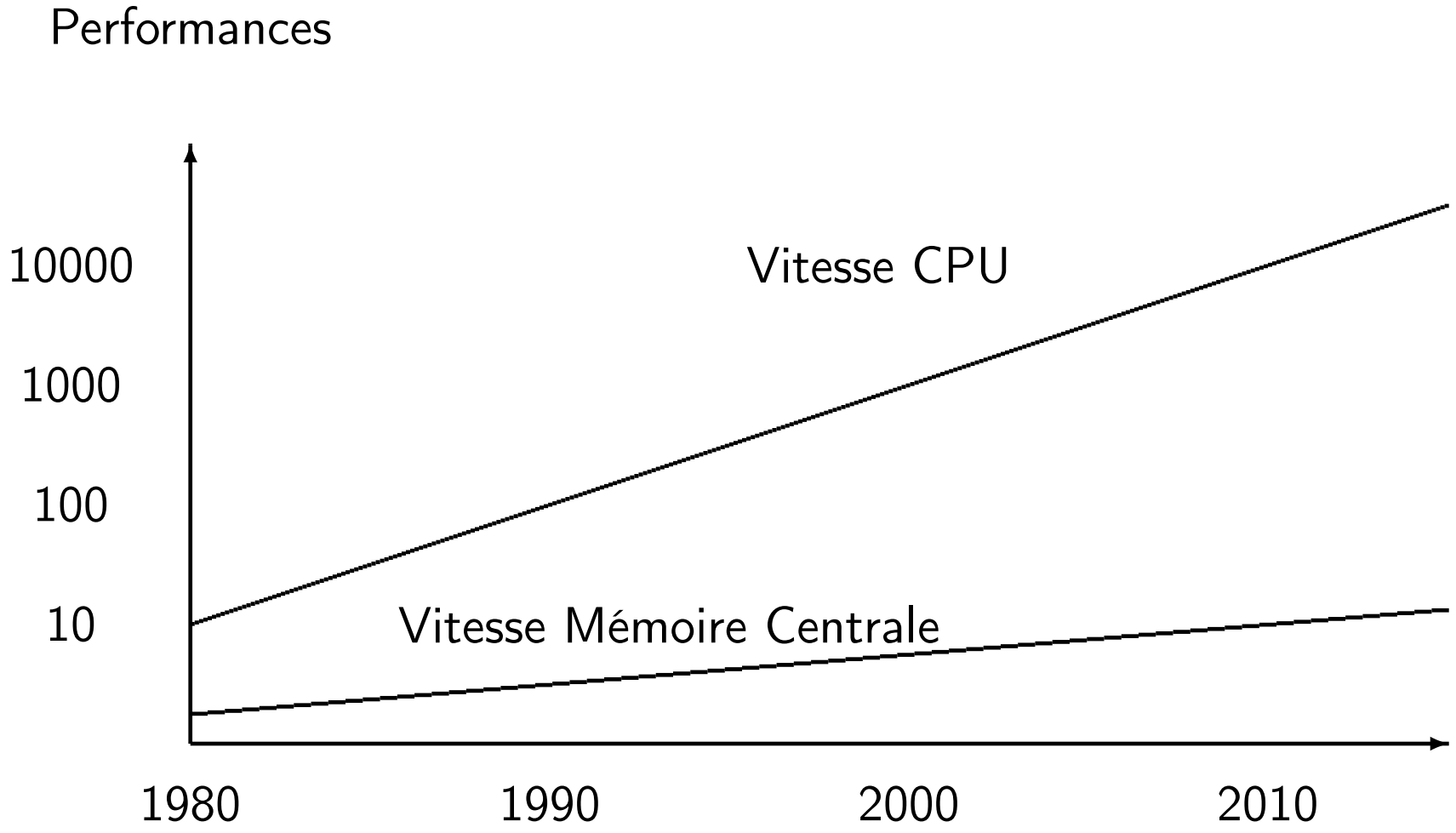
Architecture des ordinateurs - Accélération

Didier Teifreto

Université de Franche Comté

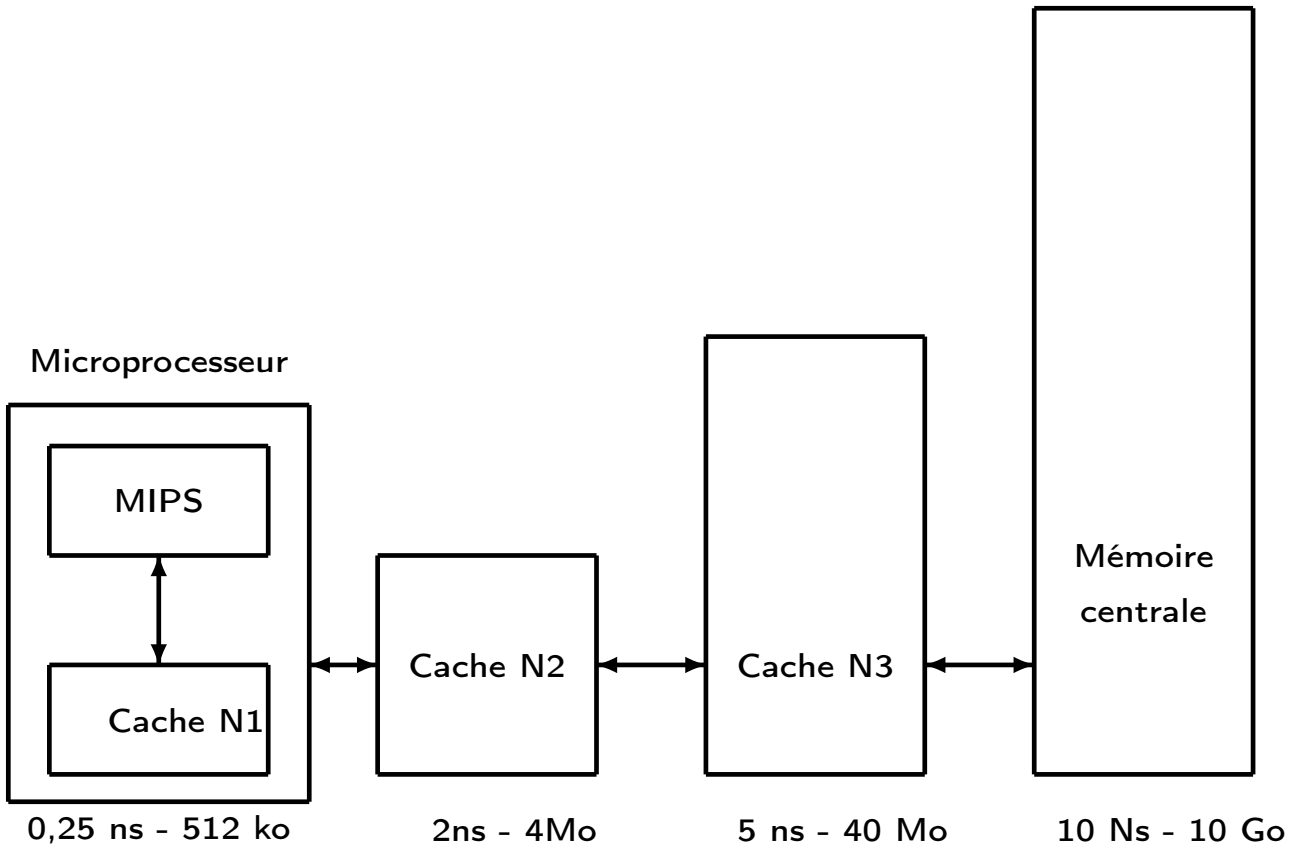
16 novembre 2015

Hiérarchie mémoire- utilité



Écart augmente de 50 % par an.

principe


Microprocesseur
MIPS
Cache N1
Cache N2
Cache N3
Mémoire centrale
0,25 ns - 512 ko
2ns - 4Mo
5 ns - 40 Mo
10 Ns - 10 Go

Localité

1. Localité temporelle : données ou instructions utilisées le seront probablement bientôt.
2. Localité spatiale : données ou instructions "autour" de celles utilisées le seront bientôt. Notion de blocs d'information
3. Trois fonctions de correspondance : direct, associative, associative par ensemble

- Le nombre d'octets par bloc : NOPB,
- La taille de la mémoire centrale (en octet) : TMC,
- La taille de la mémoire cache (en octet) : TC,
- Le nombre de blocs dans mémoire centrale NBMC=$\frac{\text{TMC}}{\text{NOPB}}$,
- Le nombre de blocs dans mémoire cache NBC=$\frac{\text{TC}}{\text{NOPB}}$.

Position d'un bloc dans le cache : Correspondance directe

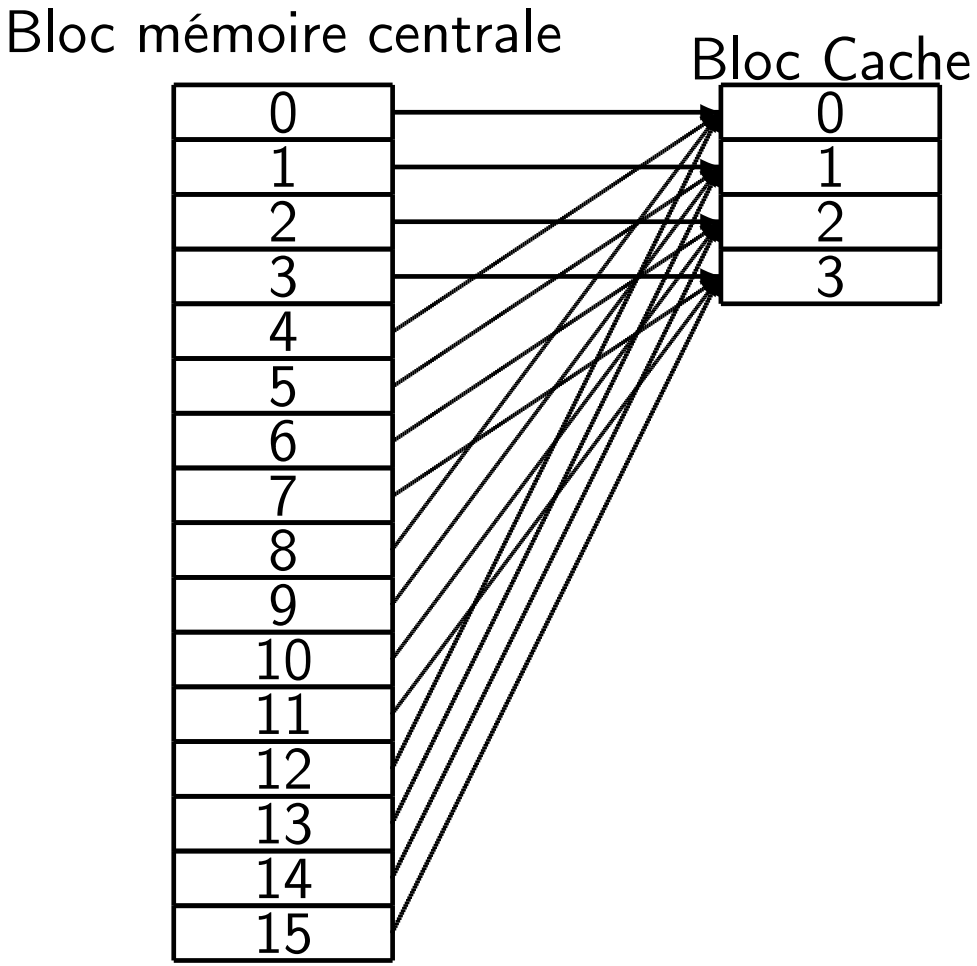
- Un bloc de la mémoire a une place dans le cache.
- bloc x de la mémoire placé en bloc y dans le cache
- $y = fd(x, NBC) = x \mod NBC = x\&(NBC - 1)$

Inconvénient : Si nous prenons deux blocs concurrents de la mémoire centrale, accédés séquentiellement, par exemple le bloc numéro x=0 et le bloc x=4,

$$\begin{aligned} y &= 0 \mod 4 = 0 \\ y &= 4 \mod 4 = 0 \end{aligned}$$

Avantage : Pour savoir si une information se trouve dans la mémoire cache, nous savons précisément où la chercher.

Correspondance directe - 2



Correspondance associative

- Un bloc de la mémoire peut être n'importe ou le cache
- Cache plein algorithme LRU (less recent used) ou random

Inconvénient : Pour savoir si un bloc est dans le cache, nous devons parcourir (dans le pire des cas) tous les blocs de celui-ci, ce qui peut être très long. En moyenne, nous devrons parcourir la moitié des blocs du cache pour rechercher l'information recherchée.

Avantage : N'importe quel bloc de la mémoire cache pouvant contenir n'importe quel bloc de la mémoire centrale simultanément, il n'y a plus de problème de concurrence de bloc.

Correspondance associative par ensemble

- Un bloc de la mémoire dans un ensemble de bloc du cache.
- Le nombre de blocs par ensemble NBPE,
- Le nombre d'ensembles dans la mémoire centrale NEMC=$\frac{\text{NBMC}}{\text{NBPE}}$,
- Le nombre d'ensembles dans la mémoire cache NEC=$\frac{\text{NBC}}{\text{NBPE}}$.

Nous devons chercher l'information x dans tous les blocs de l'ensemble y. La relation est de la forme :

$$y = f_e(\text{x, NEC}) = x \mod NEC$$

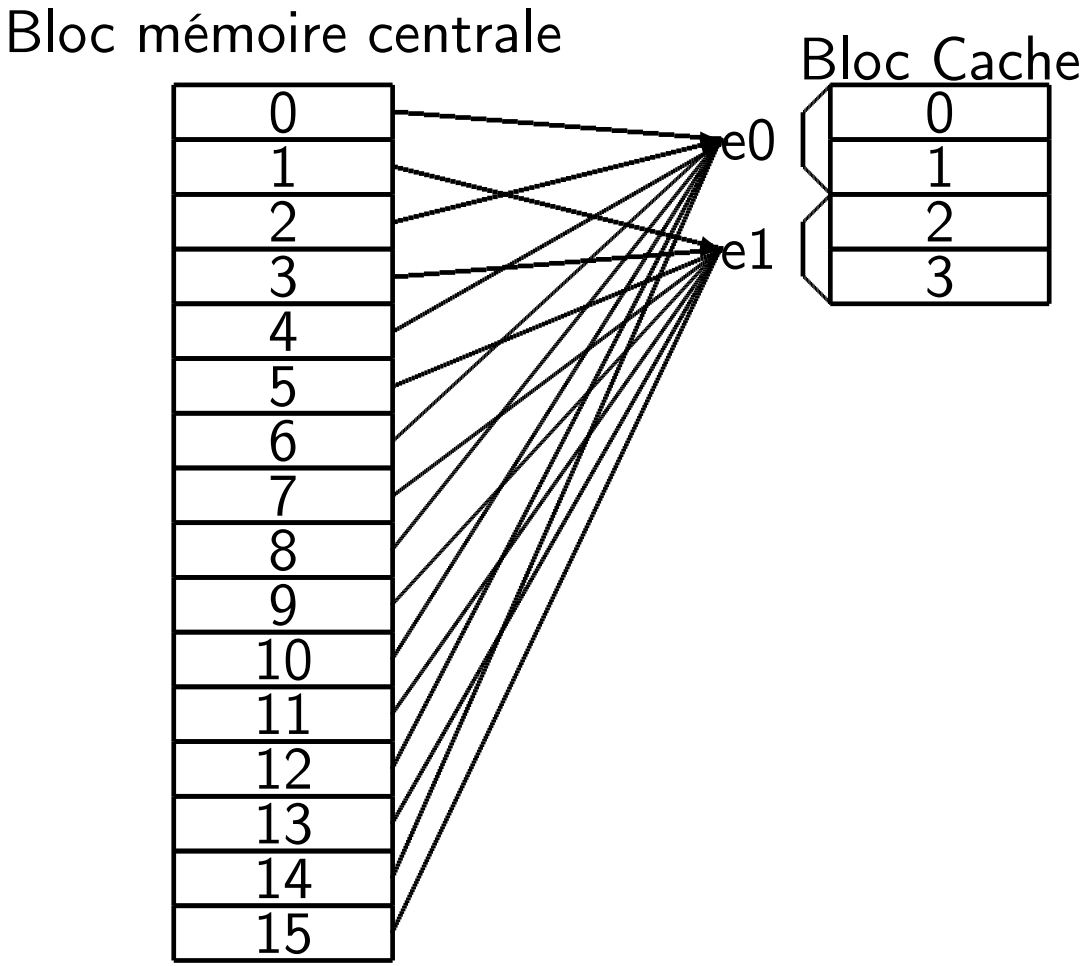
Correspondance associative par ensemble - 2

Avantage : Prenons deux blocs concurrents de la mémoire centrale accédés séquentiellement, le bloc numéro x=0 et le bloc x=4.

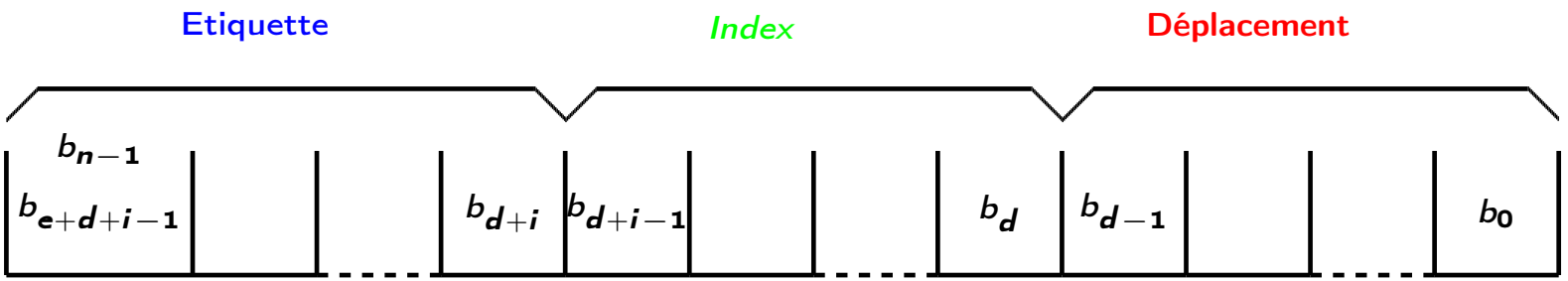
$$y = 0 \mod 4 = 0$$
$$y = 4 \mod 4 = 0$$

Ces deux blocs se trouveront tous les deux dans l'ensemble 0 de la mémoire cache, mais dans des blocs différents. Nous devrons alors chercher dans l'ensemble pour savoir si le bloc recherché est présent.

Correspondance associative par ensemble - 3



Trouver un bloc dans le cacje



Le déplacement Trouver une information à l'intérieur d'un bloc,

L'index Savoir dans quel bloc ou ensemble de la mémoire cache se trouve l'information recherchée (dépend de la fonction de correspondance)

L'étiquette Savoir quel bloc de la mémoire centrale se trouve dans un bloc de la mémoire cache (stockée dans le cache, en plus des données et pourra ainsi être comparée à l'étiquette du bloc recherché. Si égales, le bloc est dans la mémoire cache : c'est un succès ; sinon : c'est un défaut de cache)

Trouver un bloc - 2

Soit un système ayant des adresses mémoire 15 bits, une mémoire cache contenant 4 kilos octets (2^{12} octets). La taille des blocs est fixée à 128 octets (2^7 octets) déplacement sur 7 bits.
Avec la fonction de correspondance directe

L'index 5 bits car le cache contient 32 blocs (2^5),
L'étiquette est donc codée sur $15 - 5 - 7 = 3$ bits

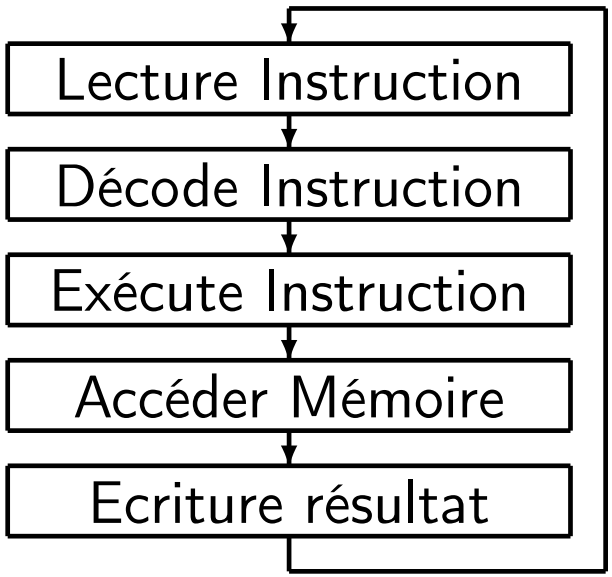
Avec la fonction de correspondance associative

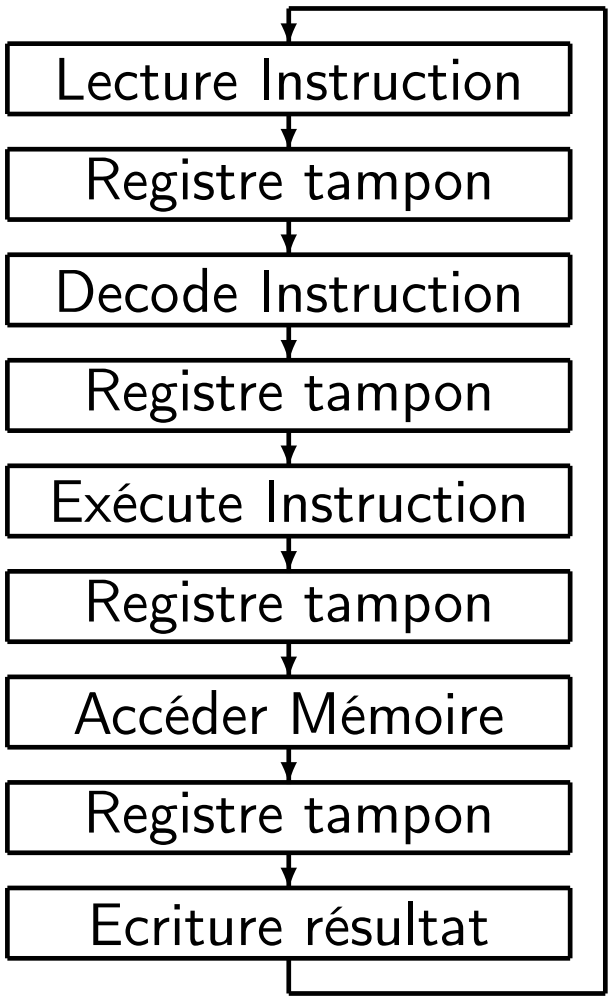
L'index est codé sur 0 bit car non utilisé,
L'étiquette est donc codée sur $15 - 7 = 8$

Avec la fonction associative par ensemble de 2 blocs :
Dans la mémoire cache il y a $\frac{32}{2} = 16$ ensembles

L'index est codé sur 4 bits car le cache contient 16 ensembles (2^4),
L'étiquette est donc codée sur $15 - 4 - 7 = 4$ bits

Pipeline d'instruction - sans /avec pipeline





Étage du pipeline

Etage LI : Lire l'instruction dans la mémoire de code et calculer l'adresse de l'instruction suivante.

Etage DI : Décoder l'instruction et rechercher les opérandes dans le banc de registres du MIPS.

Etage EX : Effectuer l'opération dans l'UAL : calculer le résultat ou calculer l'adresse ou tester deux valeurs suivant l'instruction à traiter. En cas de branchement, calculer l'adresse de l'instruction suivante.

Etage M : Accéder à la mémoire de données en lecture ou en écriture. En cas de branchement préparer l'adresse de l'instruction suivante.

Etage ER : Écriture du résultat dans le jeu de registres et en cas de branchement charger l'instruction cible.

Accélération

Si une version du processeur non pipeliné traite une instruction en un temps nommé TSP , le temps de traitement d'une instruction sur un processeur pipeliné TP contenant n étages sera calculé en utilisant la relation suivante :

$$TP = \frac{TSP}{n}$$

En tenant compte du temps passé dans le registre tampon TT, la relation devient donc :

$$TP = \frac{TSP}{n} + TT$$

Diagramme temporel

Instruction / Cycle	1	2	3	4	5	6	7	8	9
addi $t0, $zero, 1	LI	DI	EX	M	ER				
addi $t1, $zero, 2		LI	DI	EX	M	ER			
addi $t2, $zero, 3			LI	DI	EX	M	ER		
addi $t3, $zero, 4				LI	DI	EX	M	ER	
addi $t4, $zero, 5					LI	DI	EX	M	ER

Aléa de données

1 : addi *$t0*, $zero, 1

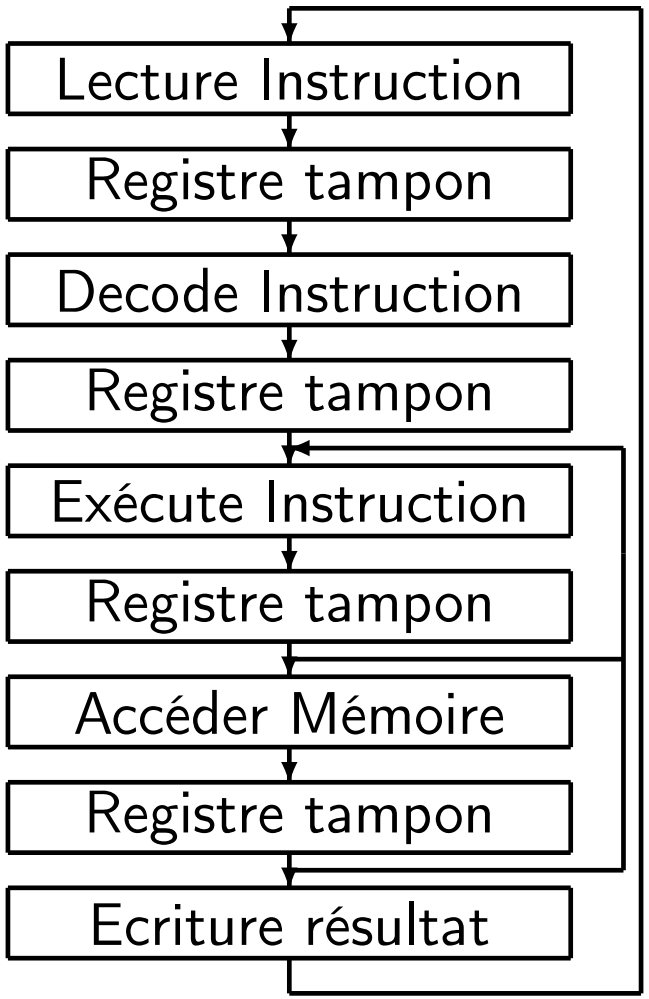
2 : addi **$t1**, $zero, 2

3 : add $t2, **$t1**, *$t0*

Instruction / Cycle	1	2	3	4	5	6	7	8	9
addi $t0, $zero, 1	LI	DI	EX	M	ER				
addi $t1, $zero, 2		LI	DI	EX	M	ER			
add $t2, $t1 ,$t0			LI	**DI**	**DI**	DI	EX	M	ER

Remarque : Pas d'aléa pour $t0 donnée disponible dans l'étage ER

Envoi


Lecture Instruction
Registre tampon
Decode Instruction
Registre tampon
Exécute Instruction
Registre tampon
Accéder Mémoire
Registre tampon
Ecriture résultat

Aléa avec envoi puis avec envoi et réorganisation

Instruction / Cycle	1	2	3	4	5	6	7
1 :addi $t0, $zero, 1	LI	DI	EX	M	ER		
2 :addi $t1, $zero, 2		LI	DI	EX	**M**	ER	
3 :add $t2, $t1, $t0			LI	**DI**	**EX**	M	ER

Instruction / Cycle	1	2	3	4	5	6	7
1 :lw $t1, 0($t0)	LI	DI	EX	M	**ER**		
2 :addi $t1, $t1, 1		LI	**DI**	**EX**	**EX**	M	ER

Avec réorganisation du code

Instruction / Cycle	1	2	3	4	5	6	7
1 :lw $t1, 0($t0)	LI	DI	EX	M	**ER**		
2 :addi $t0,$t0,4		LI	DI	EX	M	ER	
3 :addi $t1,$t1,1			LI	**DI**	**EX**	M	ER

Aléa de branchement

Instruction / Cycle	1	2	3	4	5	6	7	8	9
bne $t1,$zero,boucle	LI	DI	EX	M	ER				
inst +1		LI	DI	ER					
inst +2			LI	DI					
inst +3				LI					
addi $t1,$t1,-1					LI	DI	EX	M	ER

Instruction / Cycle	1	2	3	4	5	6	7	8	9
bne $t1,$zero,boucle	LI	DI	EX	M	ER				
inst +1		LI	DI	ER	M	ER			
inst +2			LI	DI	EX	M			
inst +3				LI	DI	EX	M	ER	
inst +4					LI	DI	EX	M	ER

Aléa de branchement

```
1  boucle: andi $t2,$a0,1      3  srl $a0,$a0,1
2  add $t1,$t1,$t2             4  bne $t0,$zero,boucle
```

Instruction / Cycle	1	2	3	4	5	6	7	8
1 : andi $t2, $a0, 1	LI	DI	EX	M	ER			
add $t1, $t1,$t2		LI	DI	EX	M	ER		
srl $a0, $a0, 1			LI	DI	EX	**M**	ER	
bne $a0, $zero, 1 :				LI	DI	**EX**	M	ER
andi $t2, $a0, 1								LI

Instruction / Cycle	1	2	3	4	5	6	7	8
bne $a0, $zero, 1 :				LI	DI	EX	M	ER
andi $t2, $a0, 1					LI	DI	EX	M

Optimisation matérielle

branchement au plus tôt : (logique de décision placée en DI)

Instruction / Cycle	1	2	3	4	5	6	7	8
1 : andi $t2, $a0, 1	LI	DI	EX	M	ER			
add $t1, $t1,$t2		LI	DI	EX	M	ER		
srl $a0, $a0, 1			LI	DI	EX	**M**	ER	
bne $a0, $zero, 1 :				LI	DI	**DI**	EX	M
andi $t2, $a0, 1							LI	DI

Instruction / Cycle	1	2	3	4	5	6	7	8
bne $t0, $zero, 1 :				LI	DI	DI	EX	M
andi $t2, $a0, 1					LI	LI	DI	EX

Optimisation logicielle

branchement au plus tôt : (logique de décision placée en DI)

Instruction / Cycle	1	2	3	4	5	6	7
1 : andi $t2, $a0, 1	LI	DI	EX	M	ER		
srl $a0, $a0, 1		LI	DI	EX	**M**	ER	
add $t1, $t1,$t2			LI	DI	EX	M	ER
bne $a0, $zero, 1 :				LI	**DI**	EX	M
andi $t2, $a0, 1						LI	DI

Instruction / Cycle	1	2	3	4	5	6	7	
bne $t0, $zero, 1 :				LI	DI	EX	M	ER
andi $t2, $a0, 1					LI	DI	EX	M

Optimisation logicielle et matérielle

branchement retardé d'un cyle : traitement instruction suivant le branchement dans tous les cas.

Instruction / Cycle	1	2	3	4	5	6	7	
andi $t2, $a0, 1	LI	DI	EX	M	ER			
1 : srl $a0, $a0, 1		LI	DI	EX	**M**	ER		
add $t1, $t1,$t2			LI	DI	EX	M	ER	
bne $a0, $zero, 1 :				LI	**DI**	EX	M	
andi $t2, $a0, 1					LI	DI	EX	M