
Pipelines scalaires

Daniel Etiemble
de@lri.fr

PERFORMANCE

- L'équation

$$\text{Temps exécution CPU} = \text{NI} * \text{CPI} * \text{T}_C$$

↓ ↓ ↓
Nombre d'instructions Cycles/ Instruction Temps de cycle

- Les composantes de la performance

- Nombre d'instructions
 - Jeu d'instructions et compilateur
- CPI
 - Microarchitecture
- T_C
 - Technologie CMOS et Microarchitecture

L'exécution d'une instruction

- Les étapes fondamentales

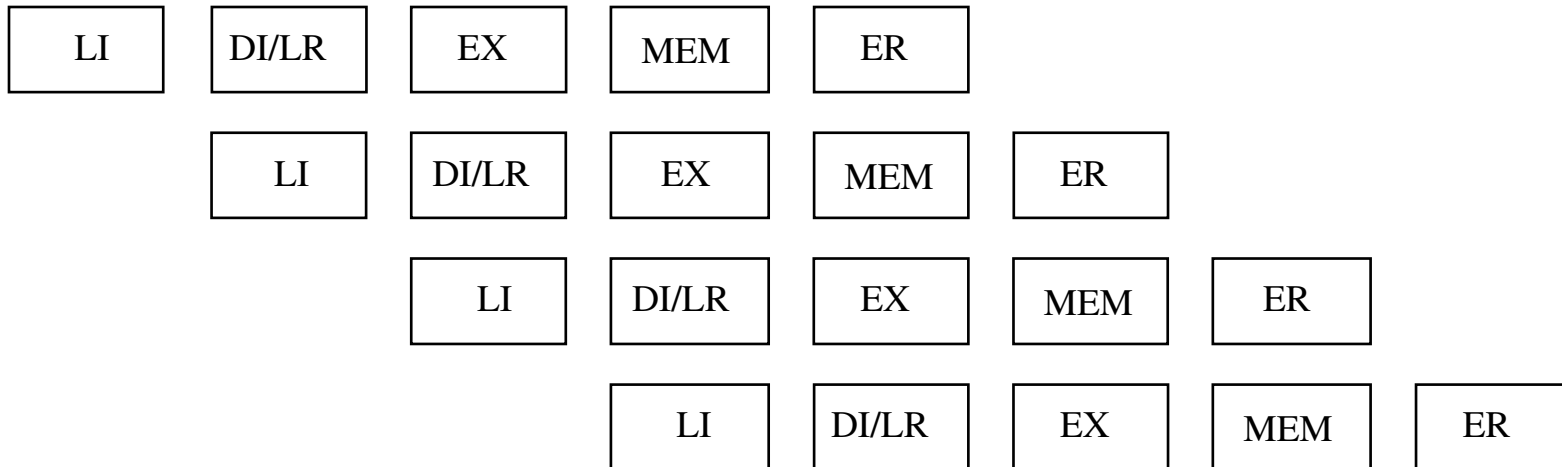
<i>Instructions UAL</i>	<i>Instructions Mémoire</i>	<i>Instructions Branchement</i>
Lecture instruction	Lecture instruction	Lecture instruction
Incrémentation CP	Incrémentation CP	Incrémentation CP
Décodage de l'instruction	Décodage de l'instruction	Décodage de l'instruction
Lecture des opérandes	Calcul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

Les différentes étapes

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

Pipeline 1 instruction par cycle

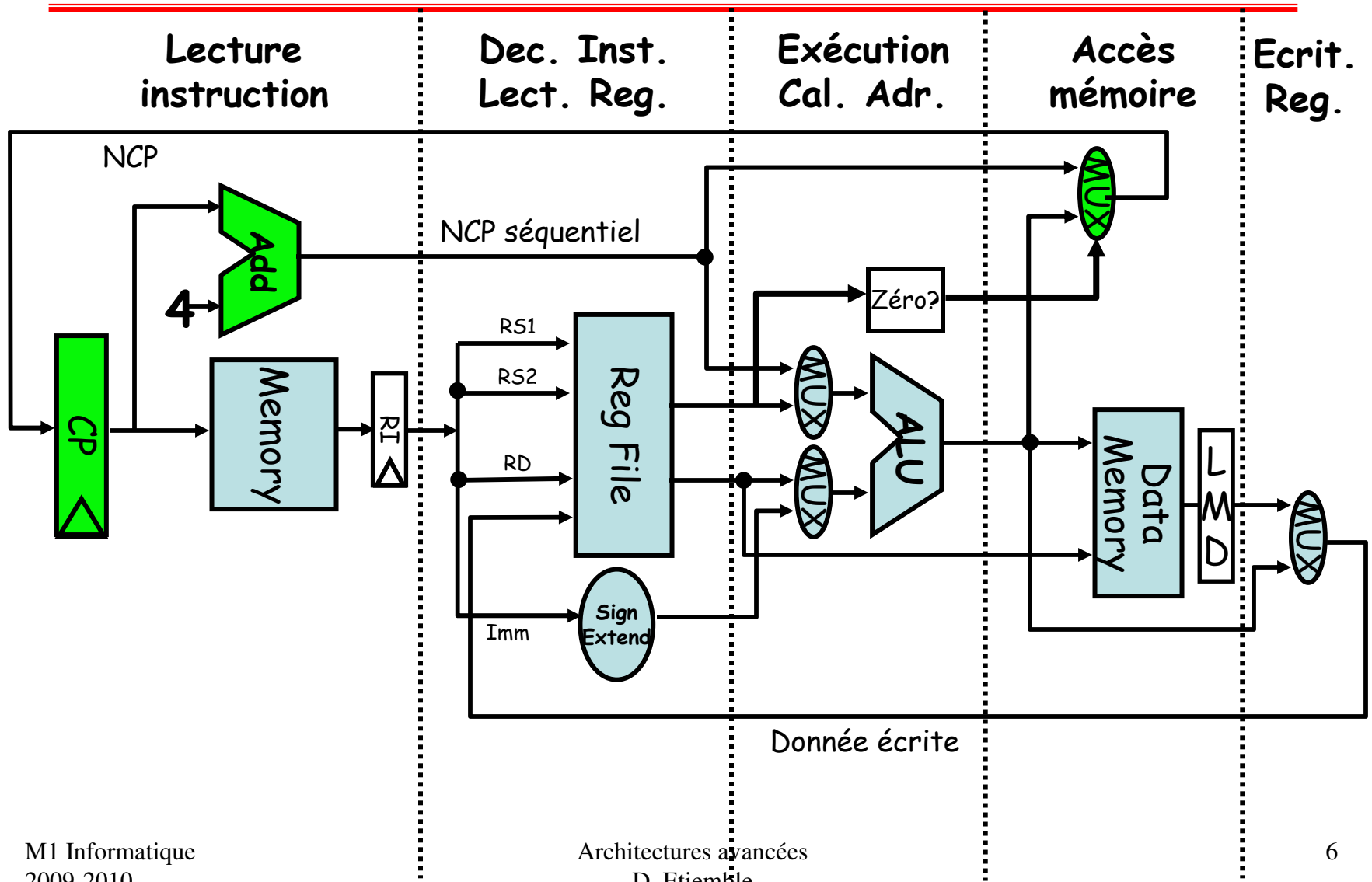
Pipeline R2000-R3000



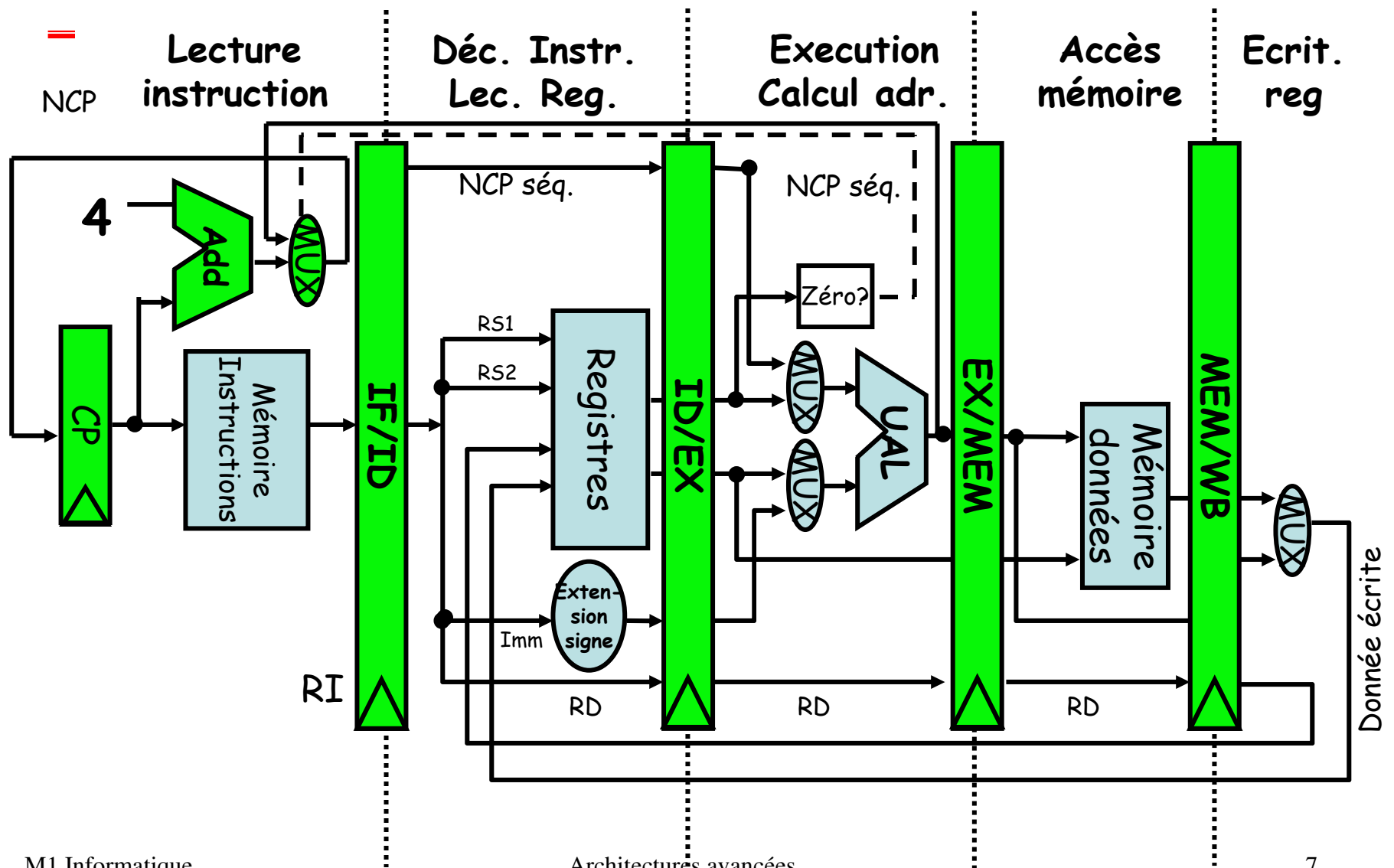
Latence : 5 cycles

Débit : 1 instruction par cycle

Exécution scalaire MIPS (non pipelinée)

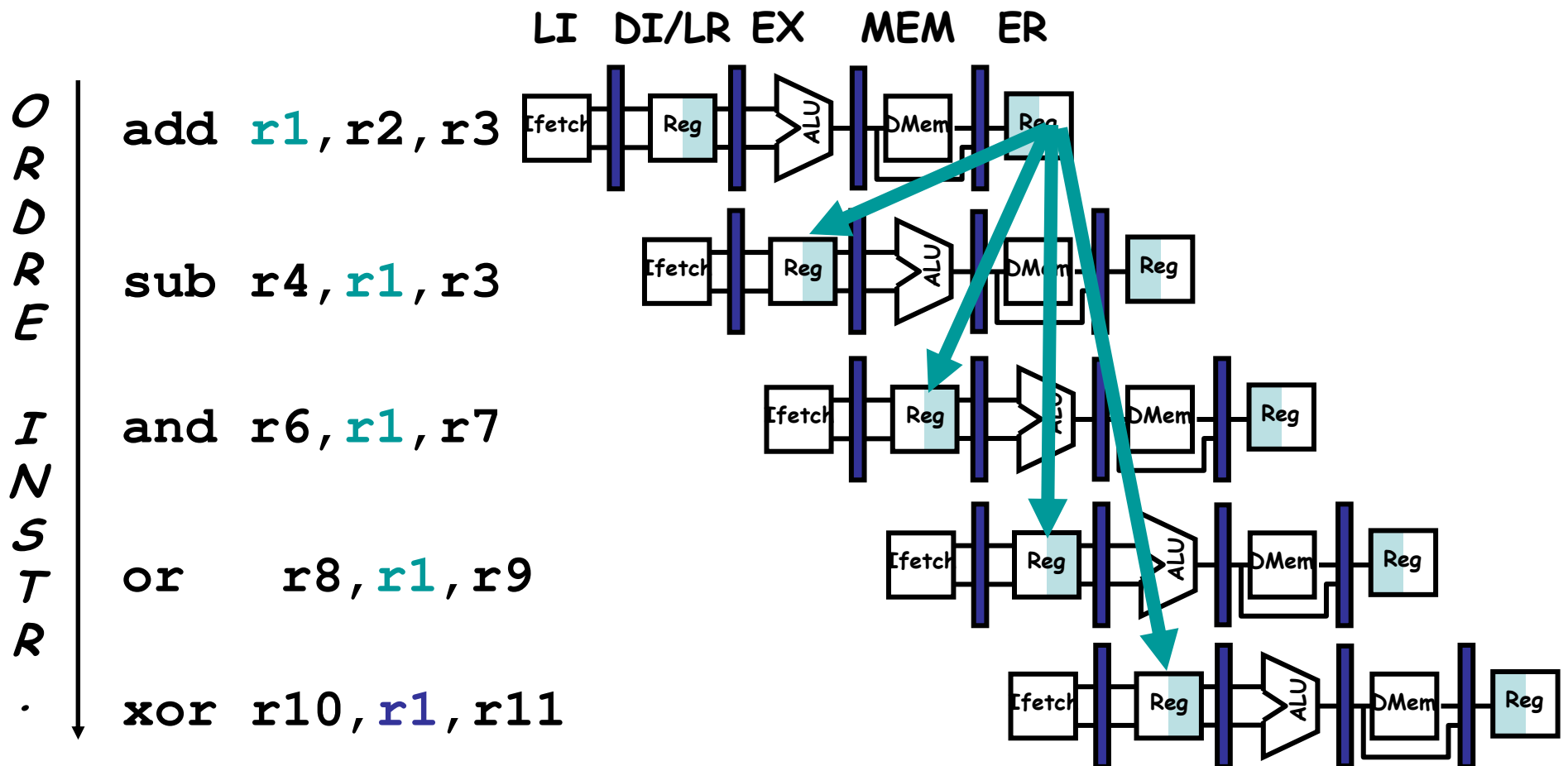


Exécution scalaire MIPS (pipelinée)



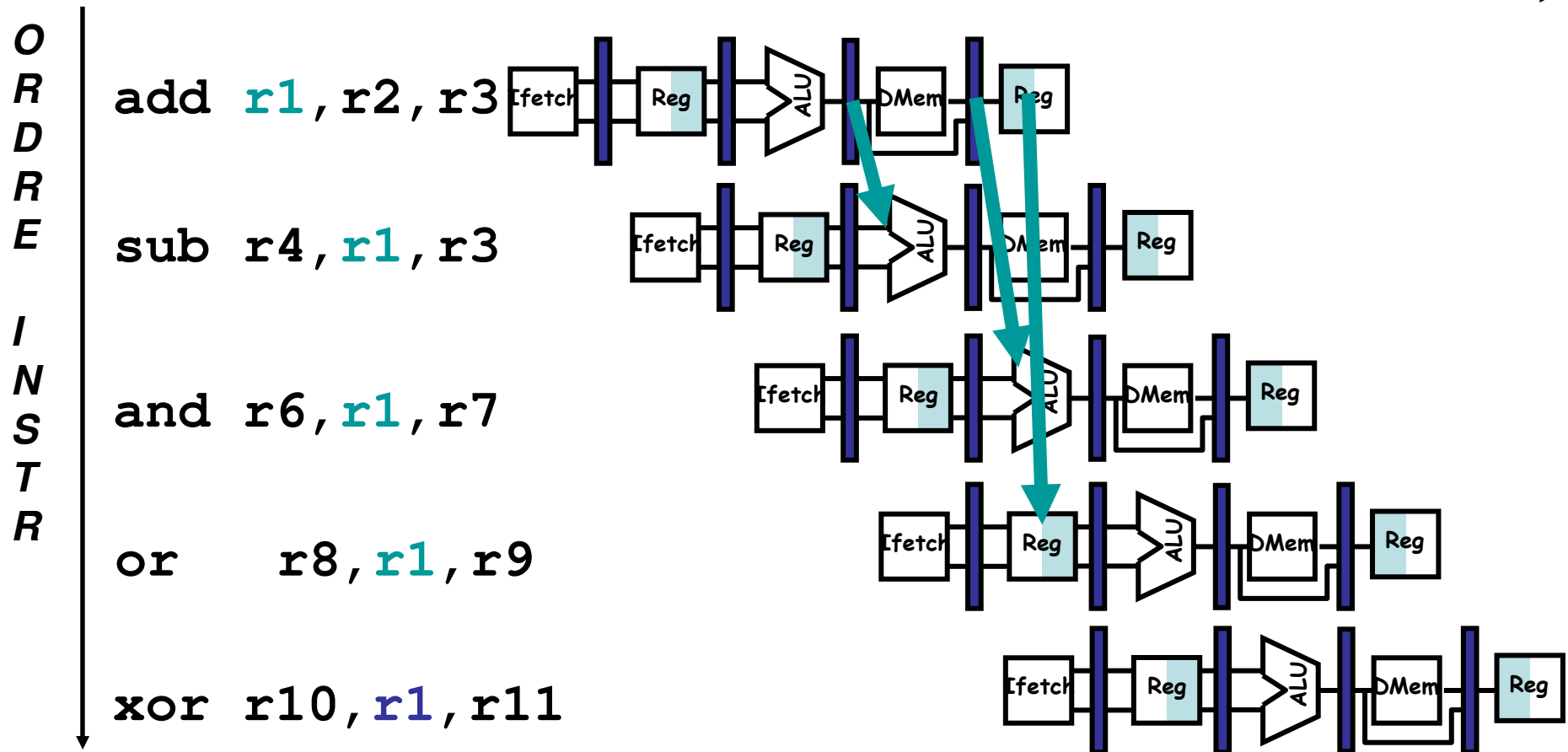
Aléas de données

Temps (cycles d'horloge)

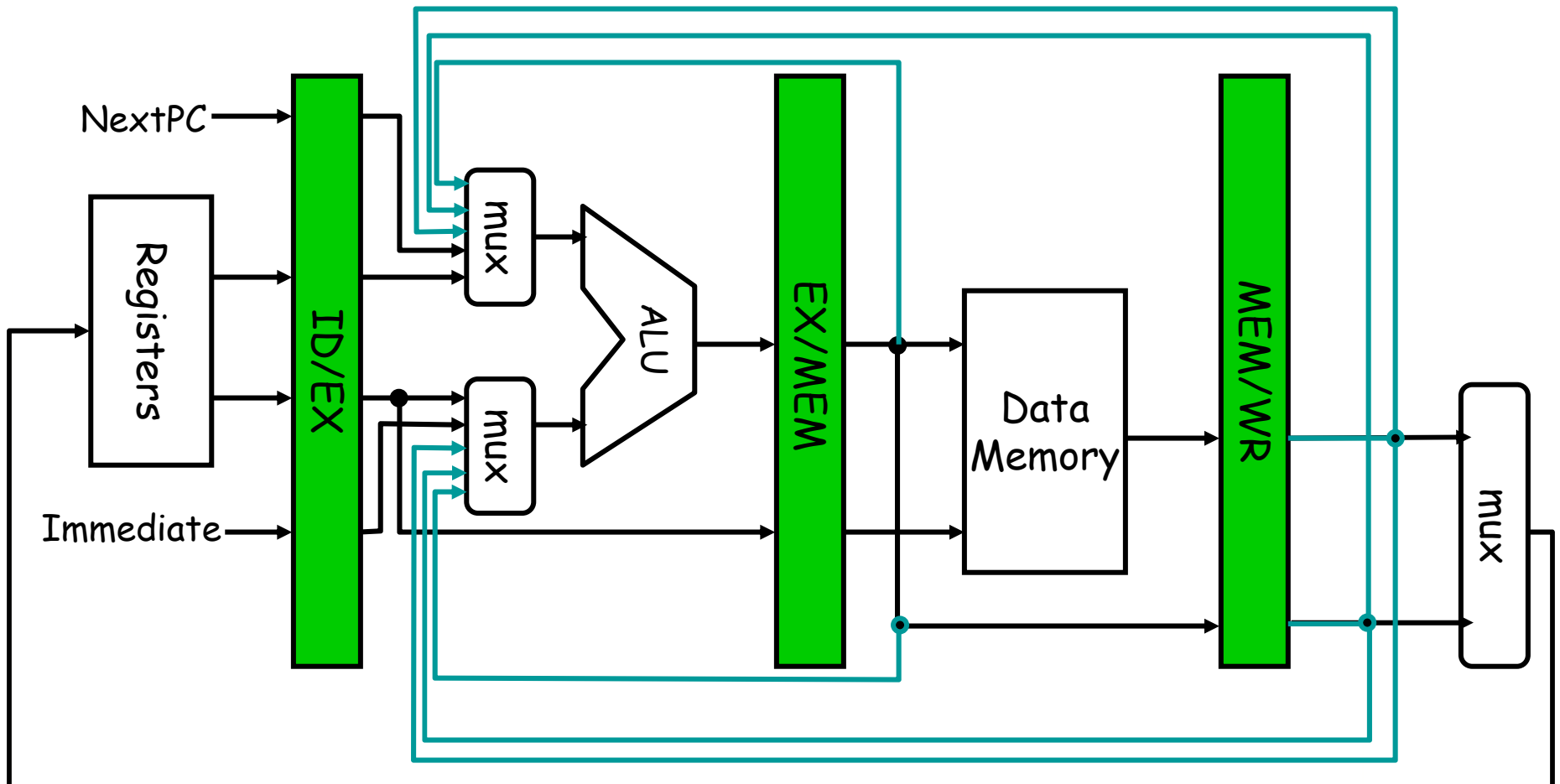


Envoi pour éviter les aléas

Temps (cycles d'horloge)

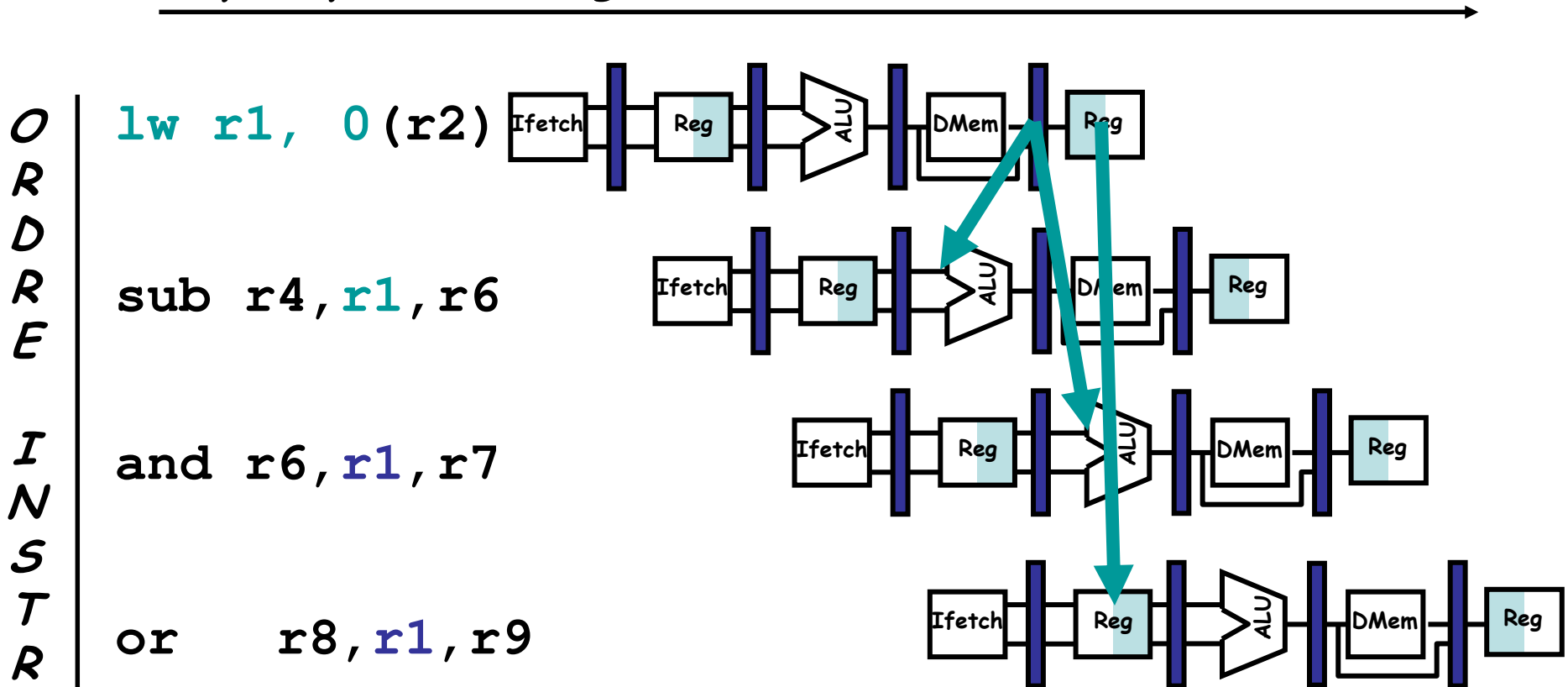


Matériel pour l'envoi



Aléas de données incontournables

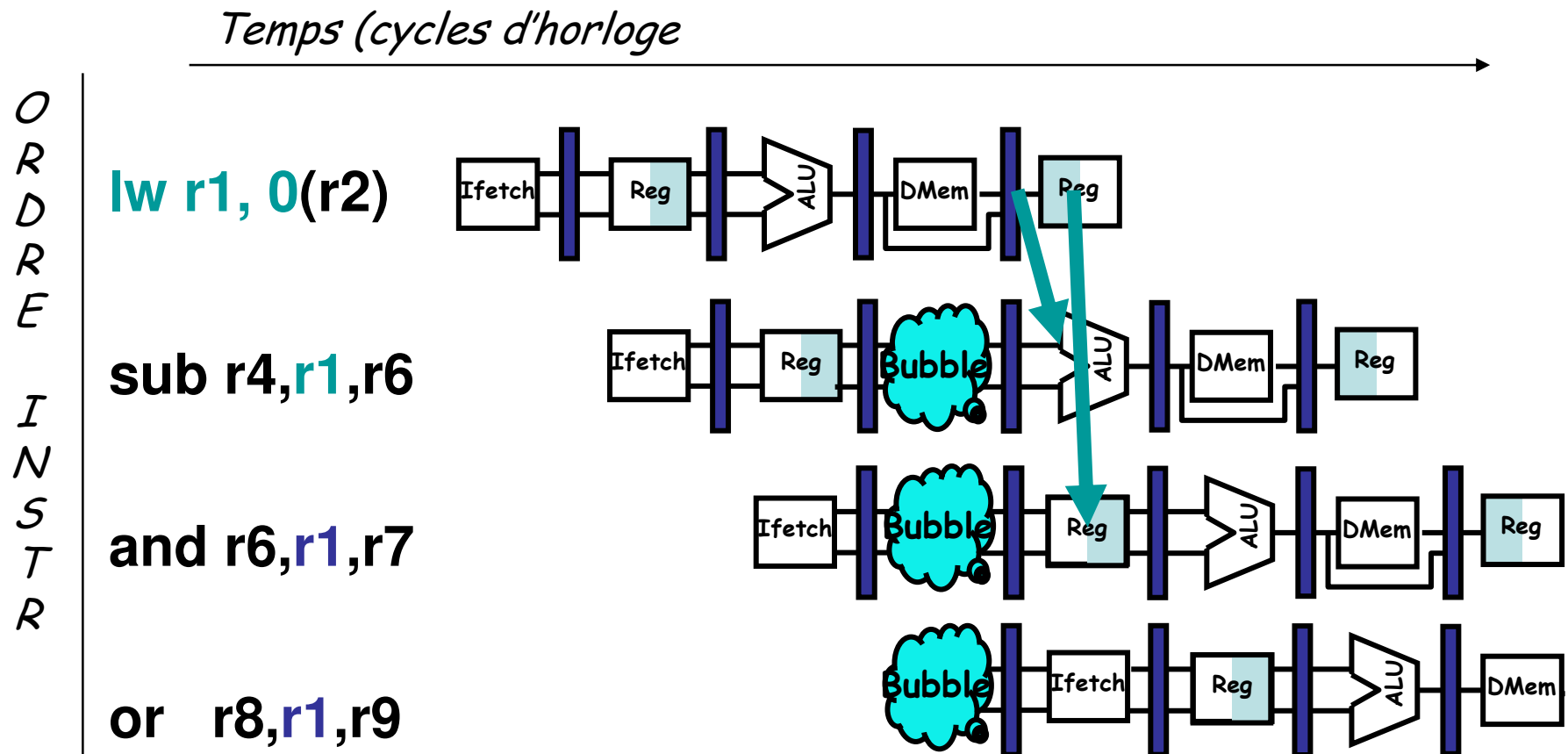
Temps (cycles d'horloge)



O
R
D
R
E

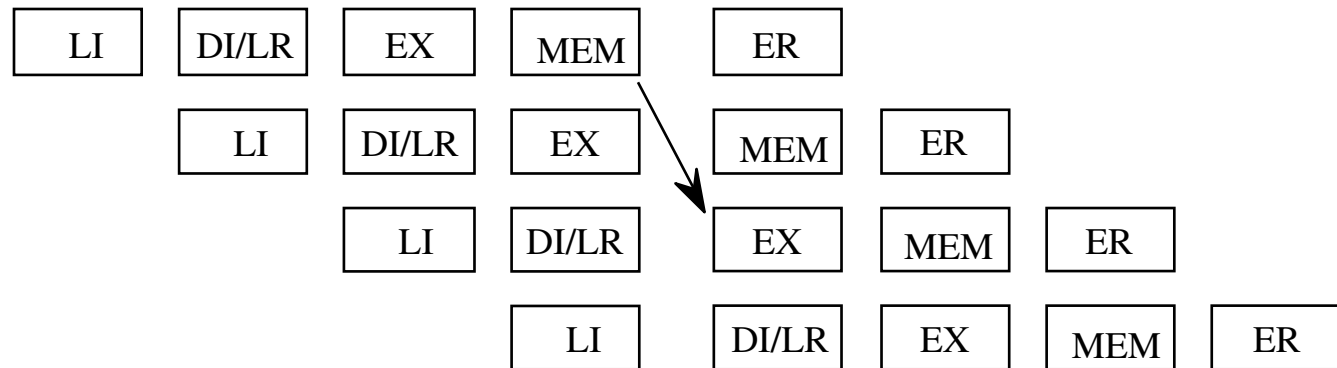
I
N
S
T
R

Aléas de données incontournables



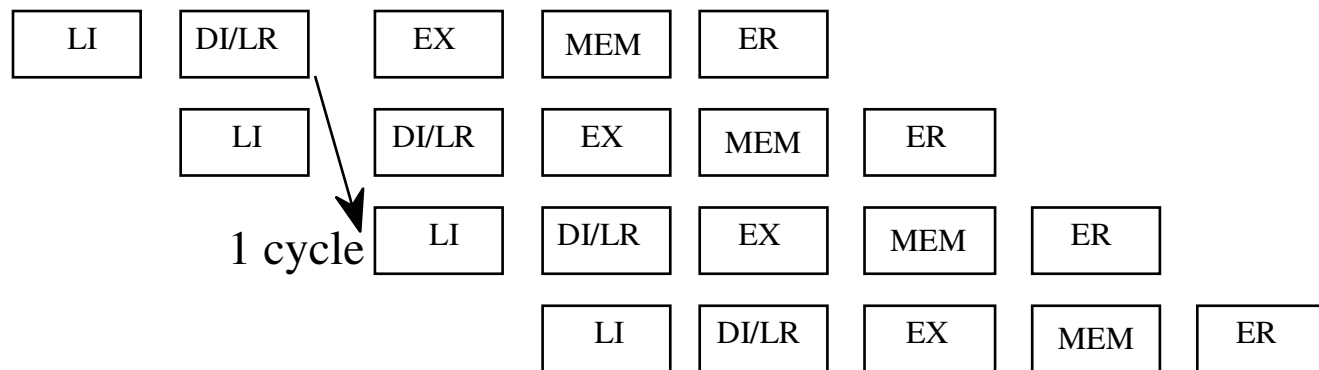
Les délais liés au pipeline

DELAI DE CHARGEMENT



DELAI DE BRANCHEMENT

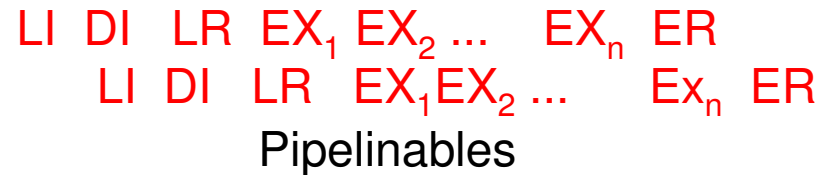
1 cycle



Plusieurs cycles pour l'étape EX

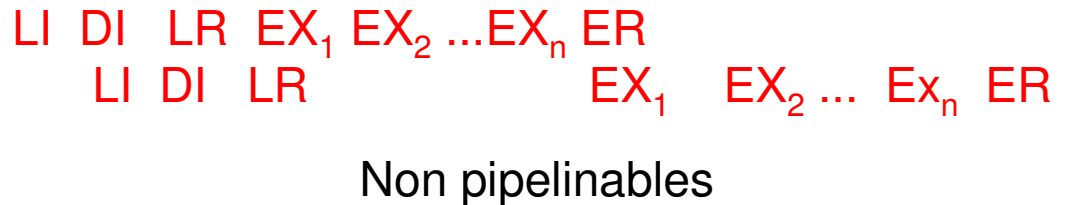
- Opérations entières

- Pipelinables
 - Multiplication
- Non pipelinable
 - Division



- Opérations flottantes

- Pipelinables
 - Addition/Soustraction
 - Multiplication
- Non pipelinable
 - Division
 - Racine carrée



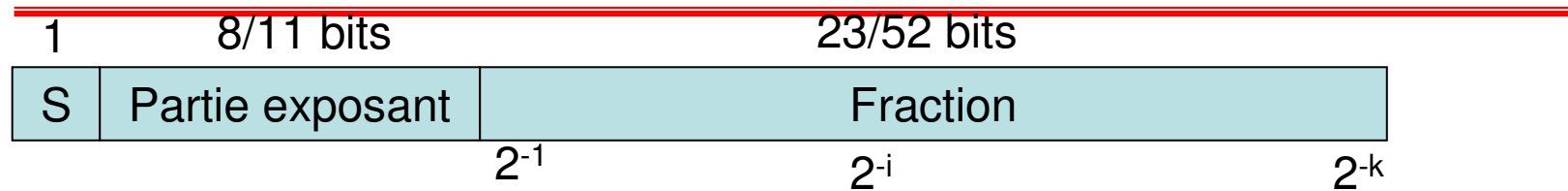
Pentium 4 – Flottant double précision

	+-	*	/	√	+-	*	/	√
P4-x87	5	7	38	38	1	2	38	38
P4-SSE2	4	6	35		2	2	35	

Latence

Débit
démarrage

La représentation flottante



$$N = -1^S \times 1, fraction \times 2^{PE - biais}$$

Si $0 < PE < PE \text{ max}$

NORMALISES

$$fraction = \sum_{i=1}^{i=k} 2^{-i}$$

PE = entier ≥ 0

Biais = 127 (SP) ; 1023 (DP)

S = bit de signe

N = 0 si PE = 0 et Fraction = 0
 N = ∞ si PE = PE max et Fraction = 0
 N = NaN si PE = PE max et Fraction $\neq 0$

$$N = -1^S \times 0, fraction \times 2^{1 - biais}$$

DENORMALISES

La multiplication flottante

- Les différentes étapes
 - Multiplication des mantisses ($PM = 1.f1 \times 1.f2$)
 - $1 \leq PM < 4$ car $1 \leq 1.f1 < 2$ et $1 \leq 1.f2 < 2$
 - Addition des Parties exposants
 - $PE = PE1 + PE2 - 127$
 - Renormalisation éventuelle
 - Si $PM \geq 2$, il faut faire $PM = PM/2$ (décalage) et $PE = PE + 1$
- Opération « longue »
 - Plusieurs cycles d'horloge
 - Pipelinable
 - Implémentation combinatoire (et non séquentielle)
 - La multiplication entière a les mêmes caractéristiques, sauf pour les processeurs de traitement du signal (1 cycle)

L'addition/soustraction flottante

- Les différentes étapes
 - Comparer les parties exposant
 - Dénormaliser la mantisse de l'opérande avec le plus petit exposant pour obtenir des parties exposant égales
 - Faire addition/soustraction des mantisses
 - Renormalisation éventuelle
 - Addition
 - $1 \leq AM < 4$ car $1 \leq 1.f1 < 2$ et $1 \leq 1.f2 < 2$
 - Si $AM \geq 2$, il faut faire $PM = PM/2$ (décalage) et $PE = PE + 1$
 - Soustraction
 - Le résultat est généralement dénormalisé
 - Recherche du premier 1 du résultat et renormalisation
- Opération « longue »
 - Plusieurs cycles d'horloge
 - Pipelinable

Les dépendances de données

Exemple:

MUL **R1**,R2,R1 RAW
 ↘
ADD R3,R7, **R1** WAR
 ↙
ADD **R1**,R8,R2 RAW
 |
MUL **R4**,R3,R1 WAW
 |
ADD **R4**,R6,R5

MUL **R1**,R2,R1 RAW
 ↘
ADD R3,R7, **R1** RAW
 ↙
ADD **R1a**,R8,R2 RAW
 |
MUL **R4**,R3,R1
 ↓
ADD **R4a**,R6,R5

RAW: vraie dépendance

WAR: anti dépendance

WAW: dépendance de sortie

Dépendances de nom
supprimées par
renommage

Dépendances de nom

Traitement des dépendances

- Contrôle des dépendances de données
 - Réalisé par matériel
 - Scoreboard
 - Tomasulo
- Suppression des dépendances de nom
 - Renommage de registres
 - Tomasulo
- Techniques logicielles pour supprimer les suspensions
 - Déroulage de boucle
 - Pipeline logiciel

Problème des exceptions/interruptions

- Exceptions
 - Situation où un évènement extérieur au CPU (demande d'interruption) ou interne au CPU (ex : division par 0, accès mémoire non aligné, interruptions logicielles...) provoque l'arrêt de l'exécution du programme et l'appel à une procédure de traitement de l'interruption
- Exceptions « propres »
 - Toutes les instructions **avant** celle ayant provoqué l'exception se terminent
 - Toutes les instructions après celle ayant provoqué l'exception n'ont pas commencé leur exécution
 - Pas de modification des registres ou de la mémoire.

Terminaison des instructions

- Pipeline et exceptions propres.
- Terminaison dans l'ordre des instructions
 - Les instructions multi-cycles obligent les instructions suivantes 1 cycle à attendre, même
 - s'il n'y a pas de dépendances de données
 - si elles ne provoquent pas d'exceptions
- Terminaison non ordonnées des instructions
 - Sans dépendance de données, si on autorise les instructions suivantes à se terminer, elles vont modifier les registres (et la mémoire)
 - Des exceptions « propres » sont alors impossibles (les instructions suivantes ne doivent pas avoir commencé)
- Exécution « spéculative » des instructions
 - Rangement temporaire des résultats des instructions
 - On ne modifie les registres et la mémoire que lorsqu'on est sûr qu'il n'y aura pas d'exception (instructions « garanties »)

LI DI LR EX₁ EX₂ ...EX_n ER
LI DI LR MEM ER

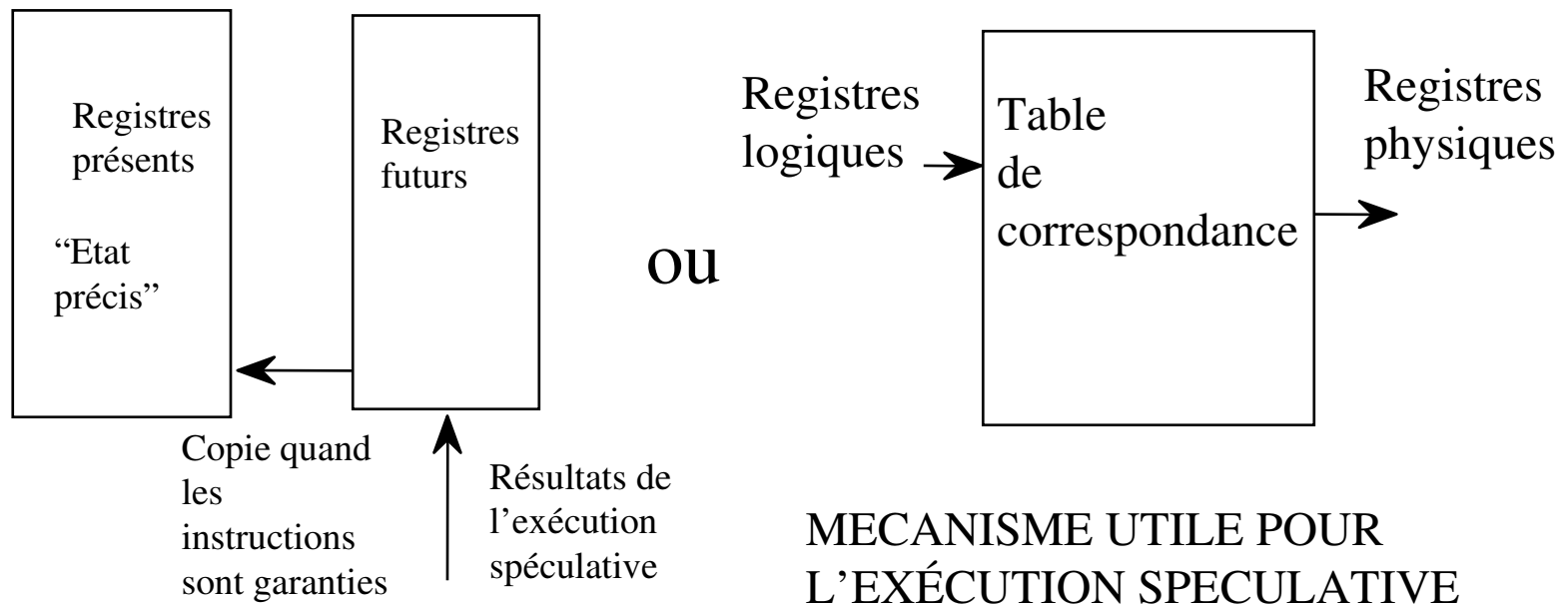
DIV (instruction multi-cycles
Instruction 1 cycle

Renommage des registres

Renommage de registres

Registres logiques (Jeu d'instructions)

Registres physiques



Multiplication de matrices : SAXPY

IJK

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++){
    x[i][j]=0;
    for (k=0; k<N; k++)
      x[i][j]+=y[i][k]*z[k][j];}
```

IKJ

```
for (i=0; i<N; i++)
  for (k=0; k<N; k++)
    for (j=0; j<N; j++)
      x[i][j]+=y[i][k]*z[k][j];
```

Produit scalaire

```
for (k=0; k<N; k++)
  r+=Y[k]*Z[k];
```

SAXPY ou DAXPY

```
for (j=0; j<N; j++)
  X[j]+=Y*Z[j];
```

Dépendances de données et suspensions

EXEMPLE

```
For (i=0; i<N; i++)  
  Y[i]+= A*X[i];
```

SAXPY

Latence des instructions (cycles)

	<i>Source</i>		
<i>Dest.</i>	UAL	LD/ST (données)	Opérations FP
UAL	2	2	
LD/ST (adresses)	2	2	
ST (données)	1	1	4
Opérations flottantes		2	5

Performance SAXPY

Boucle non optimisée

Boucle	1	LD F1 , (R1)	; charge X(i)
	2	LD F2 , (R2)	; charge Y(i)
	3	FMUL F1 ,F0, F1	; a * X(i)
	4		
	5		
	6		
	7		
	8	FADD F2 ,F2, F1	; a * X(i) + Y(i)
	9	SUB R6,R7,R1	; compare i et N-1
	10	ADDI R1,R1,8	; adresse X(i+1)
	11	ADDI R2,R2,8	; adresse Y(i+1)
	12	SD F2 , -8(R2)	; range Y(i)
	13	BNE R6, R0, Boucle	;si I<N-1, branchement

13 cycles (4 cycles sont perdus)

Déroulage de boucle

Loop	1	LD F1 , (R1)	; charge X(i)	14	FADD F2 ,F2 , F1	; a * X(i) + Y(i)
	2	LD F3 , 8(R1)	; charge X(i+1)	15	FADD F4 ,F4 , F3	; a * X(i+1) + Y(i+1)
	3	LD F5 , 16(R1)	; charge X(i+2)	16	FADD F6 ,F6 , F5	; a * X(i+2) + Y(i+2)
	4	LD F7 , 24(R1)	; charge X(i+3)	17	FADD F8 ,F8 , F7	; a * X(i+3) + Y(i+3)
	5	LD F2, (R2)	; charge Y(i)	18	SD F2 , (R2)	; range Y(i)
	6	LD F4, 8(R2)	; charge Y(i+1)	19	SD F4 , 8(R2)	; range Y(i+1)
	7	LD F4, 16(R2)	; charge Y(i+2)	20	SD F6 , 16(R2)	; range Y(i+2)
	8	LD F4, 24(R2)	; charge Y(i+2)	21	SD F8 , 24(R2)	; range Y(i+3)
	9	FMUL F1 ,F0, F1	; a * X(i)	22	ADDI R1,R1,32	; adresse X(i+4)
	10	FMUL F3 ,F0, F3	; a * X(i+1)	23	ADDI R2,R2,32	; adresse Y(i+4)
	11	FMUL F5 ,F0, F5	; a * X(i+2)	24	BNEQ R6, Loop	; si I<N-4, branch
	12	FMUL F7 ,F0, F7	; a * X(i+3)			
	13	SUB R6,R7,R1	; compare i et N-4			

**6 cycles/itération
au lieu de 13**

- * Plus de suspensions (4 cycles)
- * 1 cycle de gestion de boucle
(4 inst./4) au lieu de 4 (- 3 cycles)

```

For (i=0; i<N; i+=4) {
    Y[i]+=A*X[i];
    Y[i+1]+=A*X[i+1];
    Y[i+2]+=A*X[i+2];
    Y[i+3]+=A*X[i+3];
}
    
```

Pipeline logiciel

SAXPY

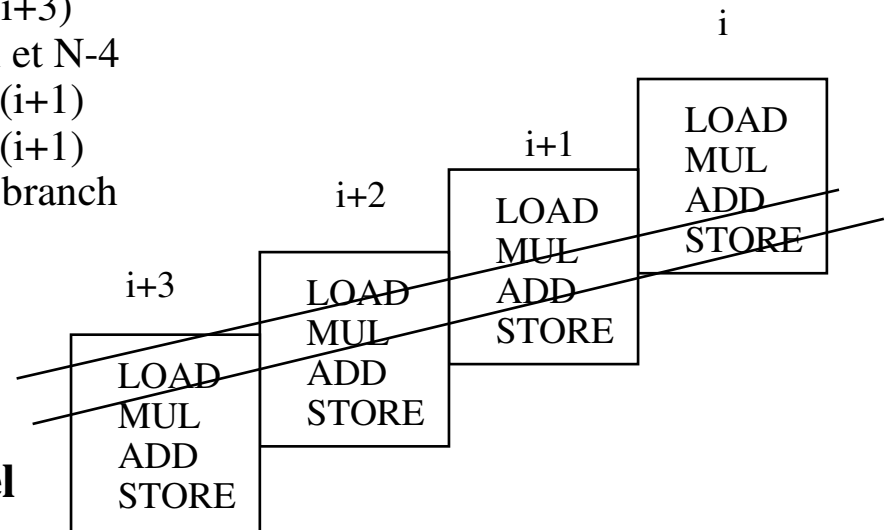
PROLOGUE

9 cycles (pas de suspension)
+ surcoût (prologue + épilogue)

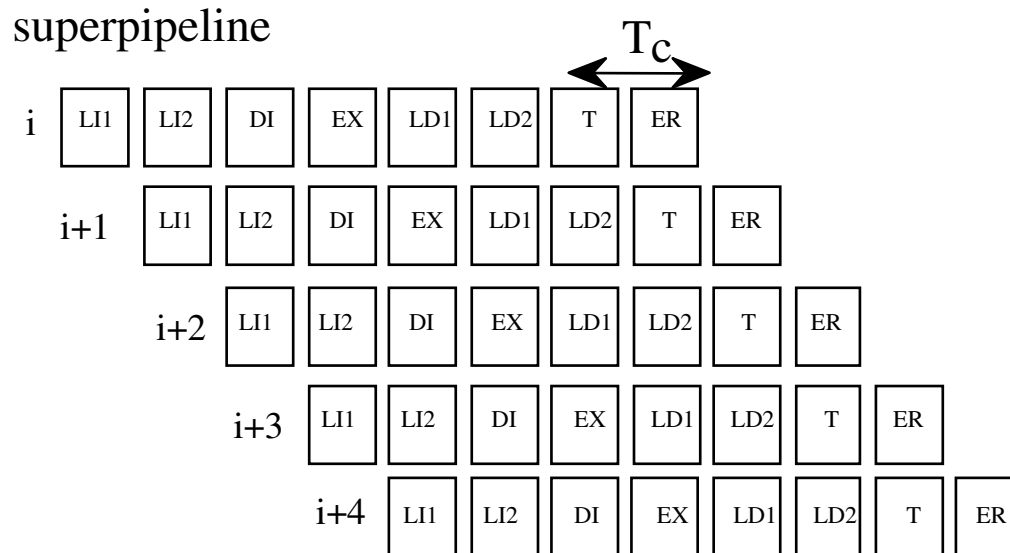
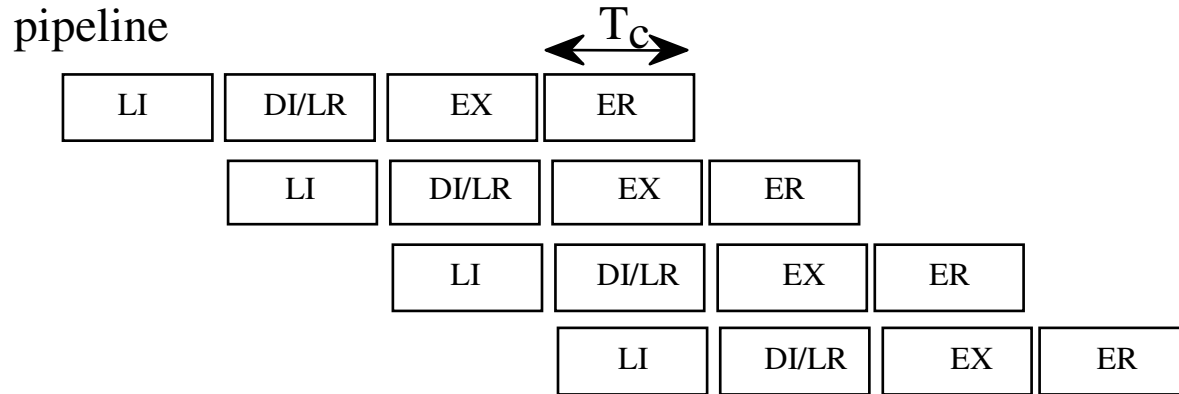
LOOP	1	SD F4, 0(R2)	; range Y(i)
	2	FADD F4,F2,F3	; a * X(i+1) + Y(i+1)
	3	FMUL F3, F0, F1	; a * X(i+2)
	4	LD F1, 24(R1)	; charge X(i+3)
	5	LD F2, 24(R2)	; charge Y(i+3)
	6	SUB R6,R7,R1	; compare i et N-4
	7	ADDI R1,R1,8	; adresse X(i+1)
	8	ADDI R2,R2,8	; adresse Y(i+1)
	9	BNEQ R6, LOOP	; si I<N-4, branch

EPILOGUE

**Boucle
pipelinée
par logiciel**



Approche superpipeline

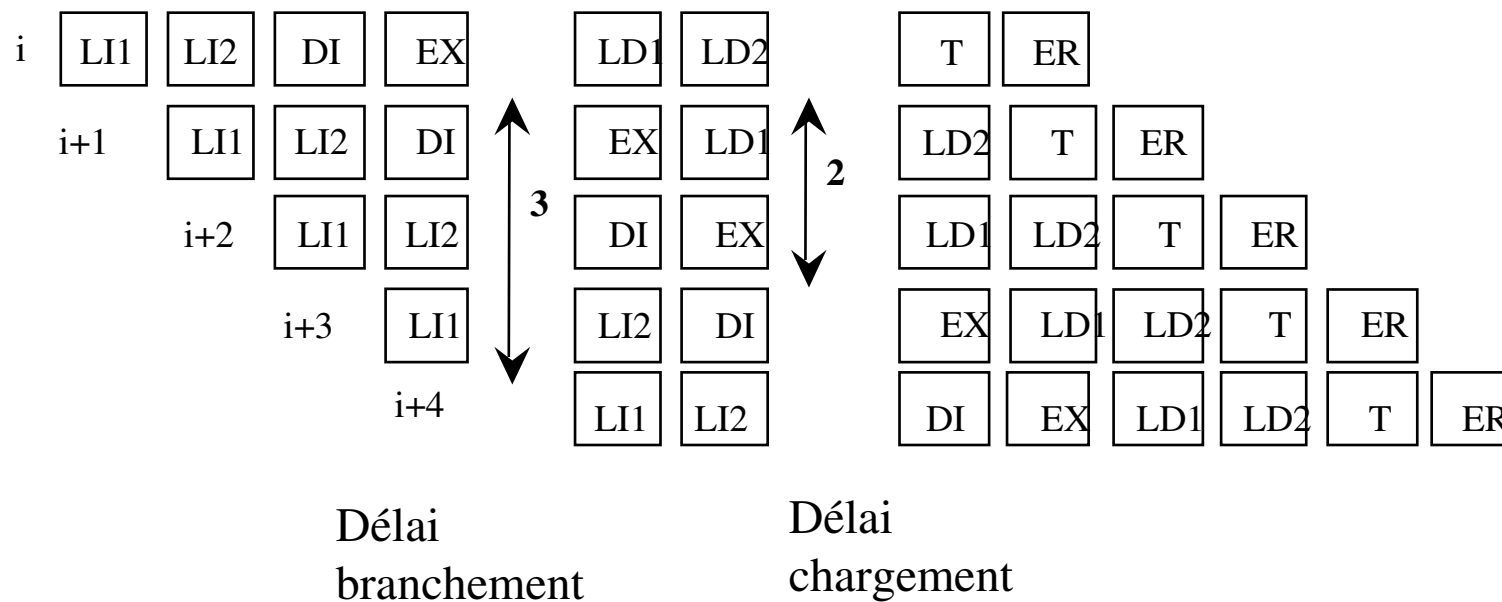


Les étages critiques
(accès cache) sont pipelinés

$$T_{cs} = T_c / 2$$

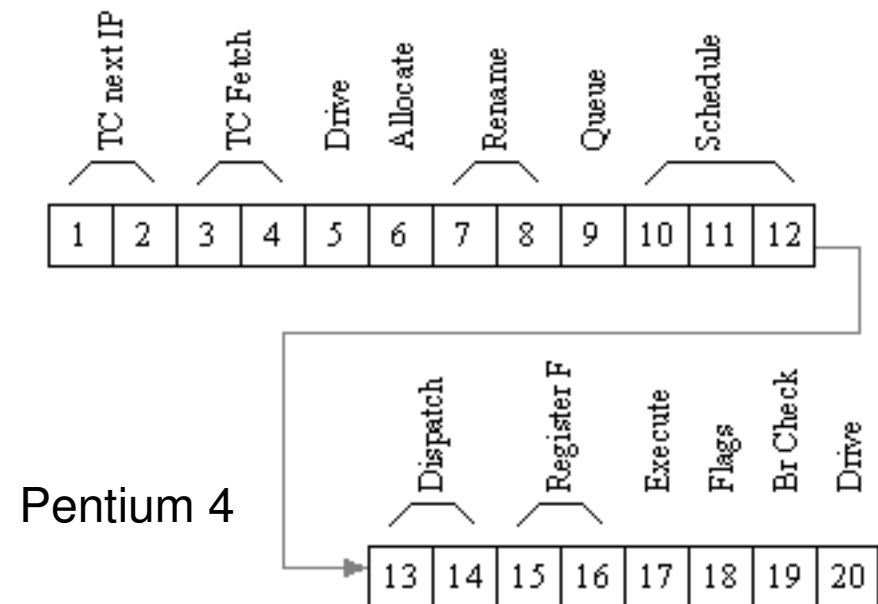
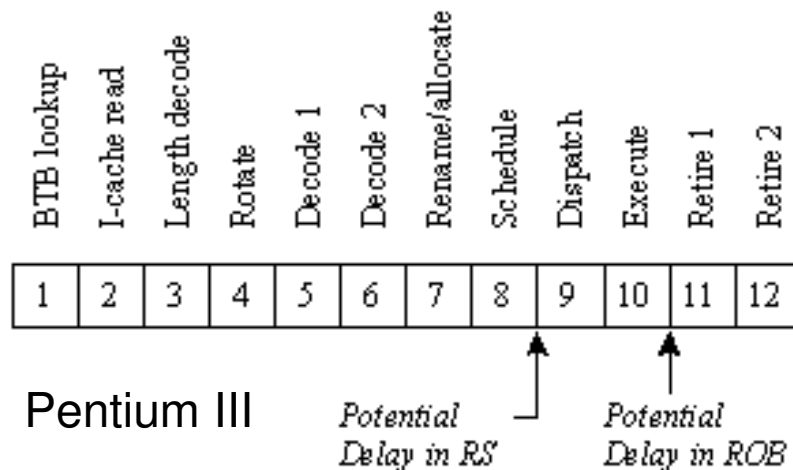
Superpipelines : chargements et branchements

Superpipeline MIPS R4000



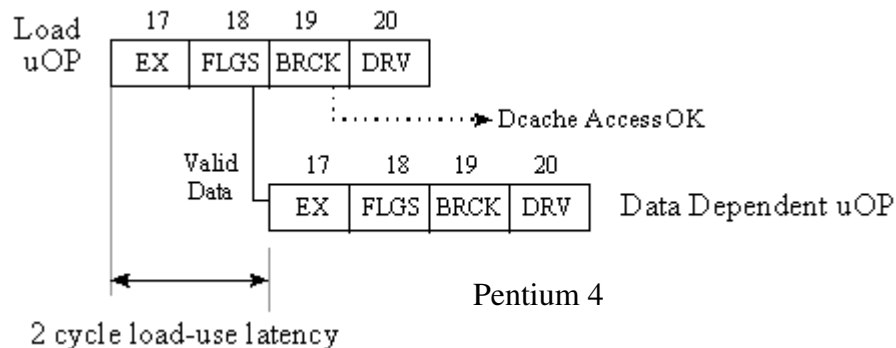
Pipelines Pentium III et Pentium 4

- Superpipeline : technique permettant d'utiliser des fréquences d'horloge élevée (2 à 3 GHz en 2003)



Latences chargement/branchement

Latence de chargement



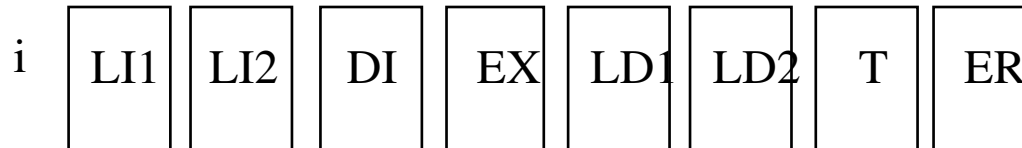
Pénalité de mauvaise prédiction

Basic Pentium® III Processor Misprediction Pipeline									
1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Basic Pentium® 4 Processor Misprediction Pipeline																			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	Br Ck	Drive				

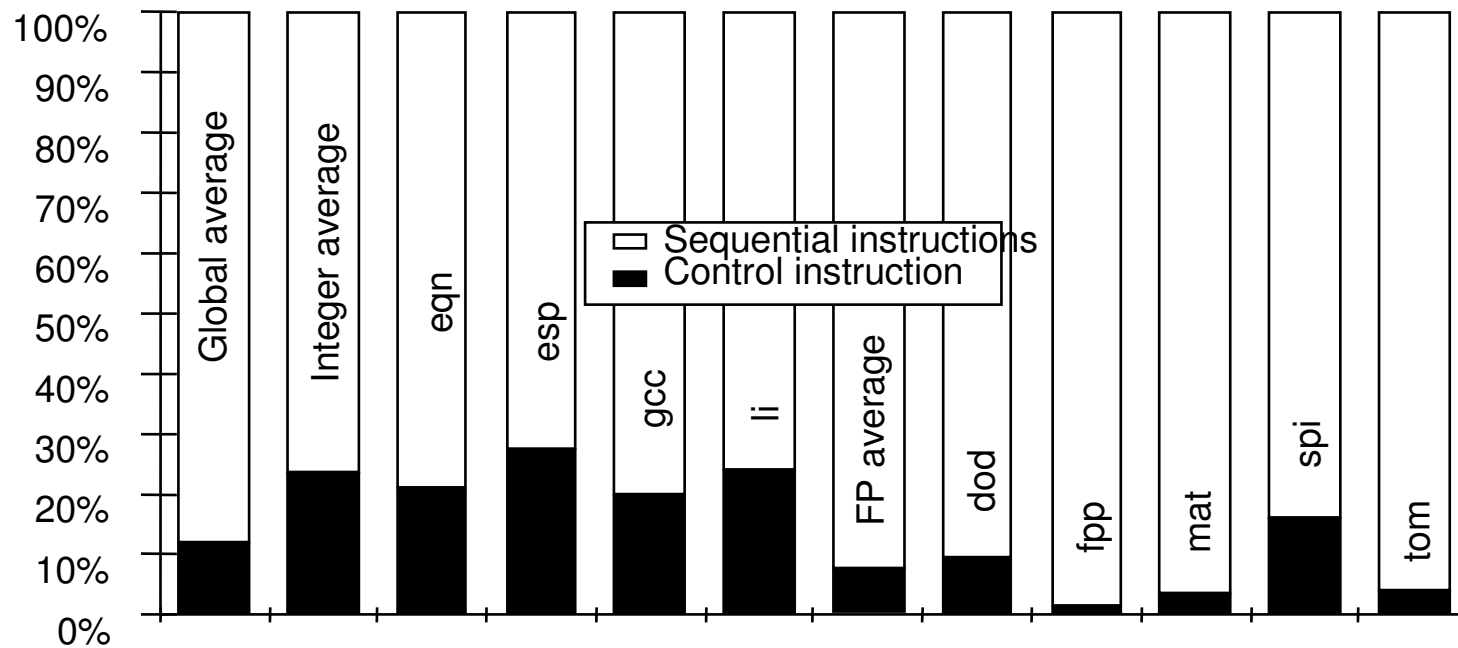
- Optimisation matérielle
 - Prédiction de branchement
- Optimisation programmeur/compilateur
 - Ordonnancement des instructions
 - Conversion SI
 - Utilisation des instructions de transfert conditionnel pour supprimer des branchements conditionnels

Caractéristiques des superpipelines



- **Avantages**
 - Simple extension du concept RISC
- **Inconvénients**
 - fréquence d'horloge plus élevée
 - circuits d'anticipation (forwarding) plus complexes
 - Délais de branchement et chargement plus élevés,
- **Remarque**
 - La compatibilité binaire doit être préservée : Pb des branchements retardés (SPARC, MIPS)

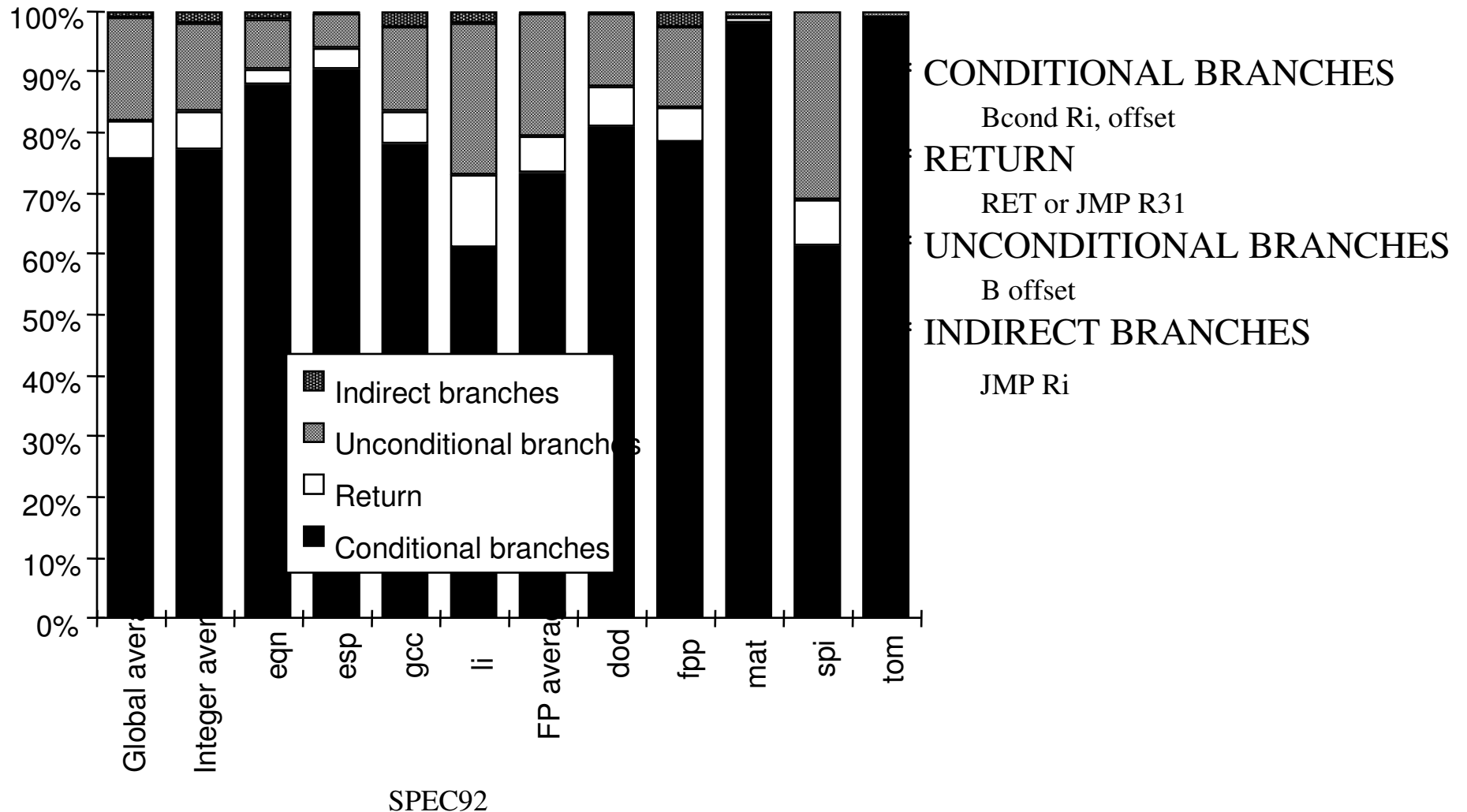
Les instructions de contrôle



SPEC92

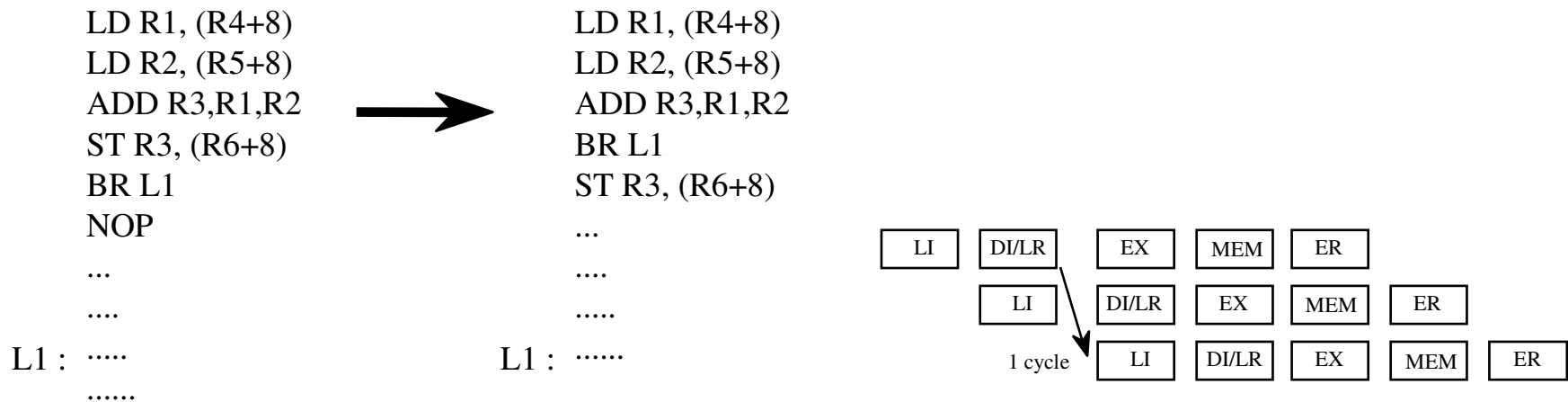
Yeh and Patt

Répartition des branchements



Techniques pour aléas de contrôle

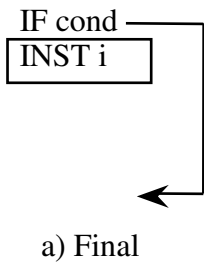
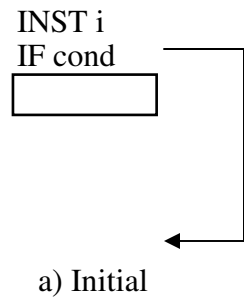
- INSTRUCTIONS SAUT/BRANCHEMENT
 - Annulation par matériel de l'instruction qui suit.
 - Toute instruction de contrôle prend 2 cycles.
 - Insérer une instruction NOP
 - Branchements retardés
 - Réordonnancement par le compilateur



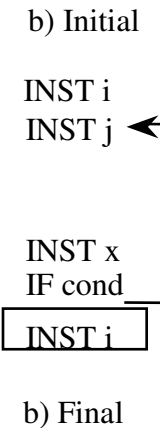
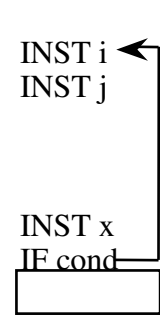
Les branchements conditionnels retardés

Schémas de réordonnancement

a) L'instruction avant le branchement ne calcule pas la condition



b) Instruction cible
Branchement très probablement pris



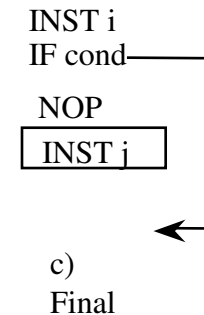
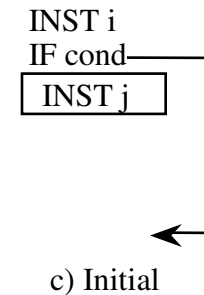
(Branchement de boucle)

Branchements avec annulation

L'instruction suivante est exécutée si le branchement est pris

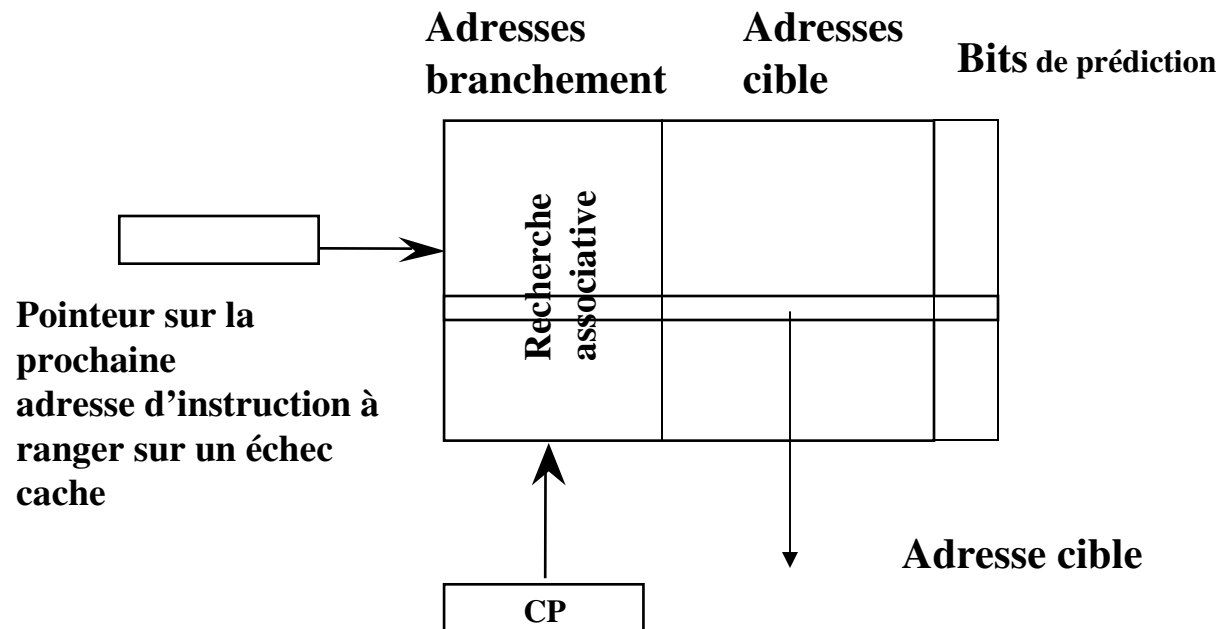
L'instruction suivante est annulée si le branchement n'est pas pris

c) Instruction après le branchement



Les branchements non retardés (BTB)

- Les caches d'adresse de branchement (BTB ou BTC)

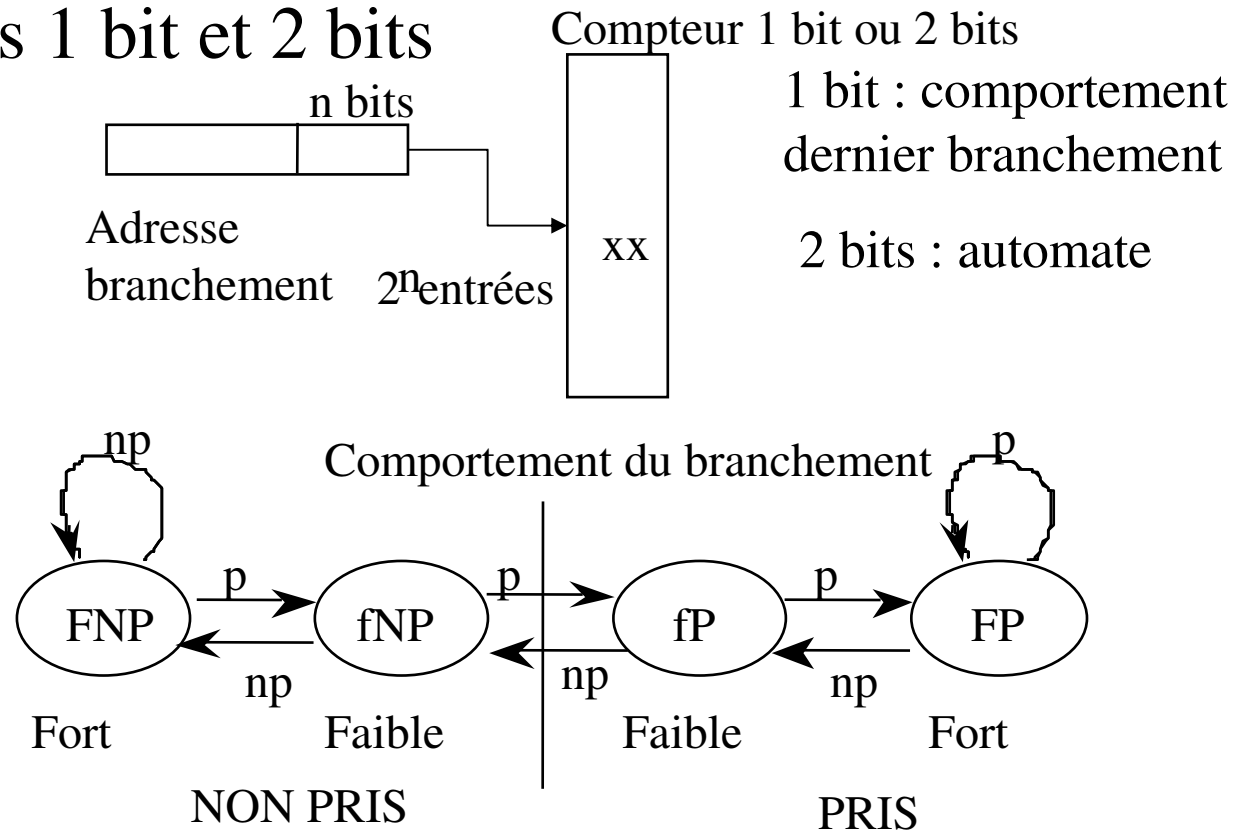


La prédiction de branchement

- Prédiction statique
 - Informations connues à la compilation
 - Branchements arrière pris (boucles)
 - Profilage des programmes
- Prédiction dynamiques
 - Prédicteurs locaux
 - Prédicteurs 1 bit et 2 bits
 - Prédicteurs globaux
 - Historique des branchements

Les prédicteurs dynamiques locaux

- Prédicteurs 1 bit et 2 bits



Prédiction des programmes flottants

- Exemple

```
FOR K = 1 to N
  FOR J = 1 to N
    C(J) = C(J) + A(K) x B(K,J)
```

```
Lk |
  |
  | Lj |
  | |
  | |
  | BNEZ Rj, Lj
  |
  |
  | BNEZ Rk, Lk
```

Chaque branchement est pris (n-1) fois et non pris 1 fois

Prédicteur 1 bit : 2 mauvaises prédictions sur n

Prédicteur 2 bits : 1 mauvaise prédiction sur n

Les branchements corrélés

- Exemple : eqntott (SPEC92)

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

b1 PRIS si $aa \neq 2$

b2 PRIS si $bb \neq 2$

b1 et b2 NON PRIS ($aa = bb = 0$) => b3 PRIS

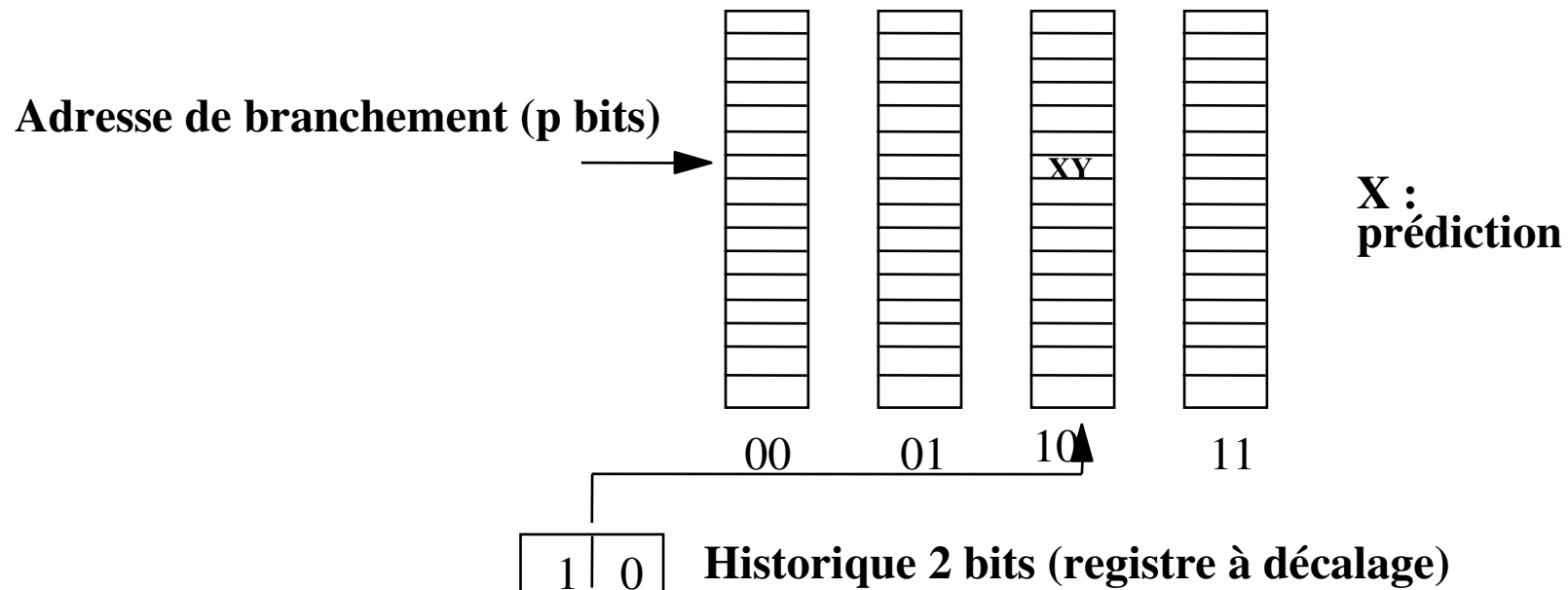
```
          SUBI    R3,R1,#2
    b1     BNEZ   R3,L1
          ADD    R1,R0,R0
L1 :      SUBI    R3,R2,#2
    b2     BNEZ   R3,L2
          ADD    R2,R0,R0
L2 :      SUB    R3,R1,R2
    b3     BEQZ  R3,L3
```

Corrélation entre les comportements de b1, b2 et b3

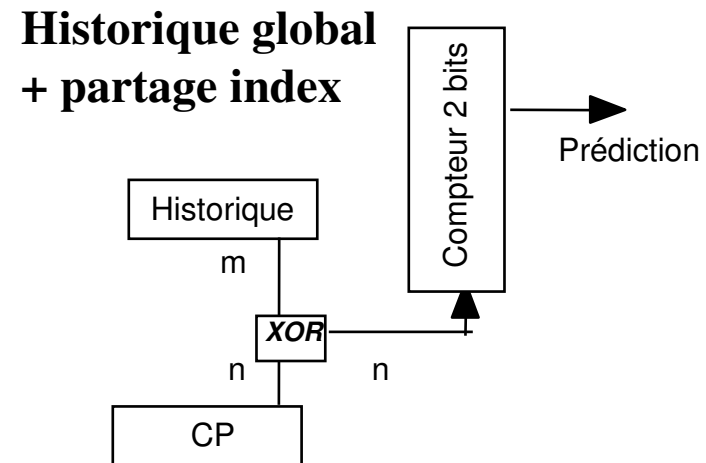
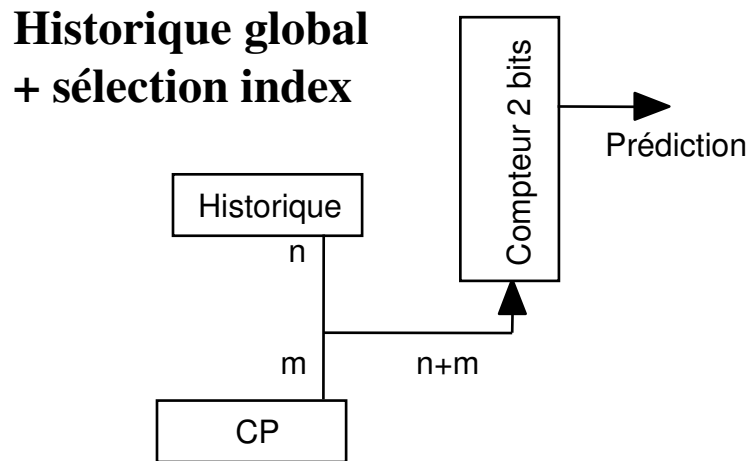
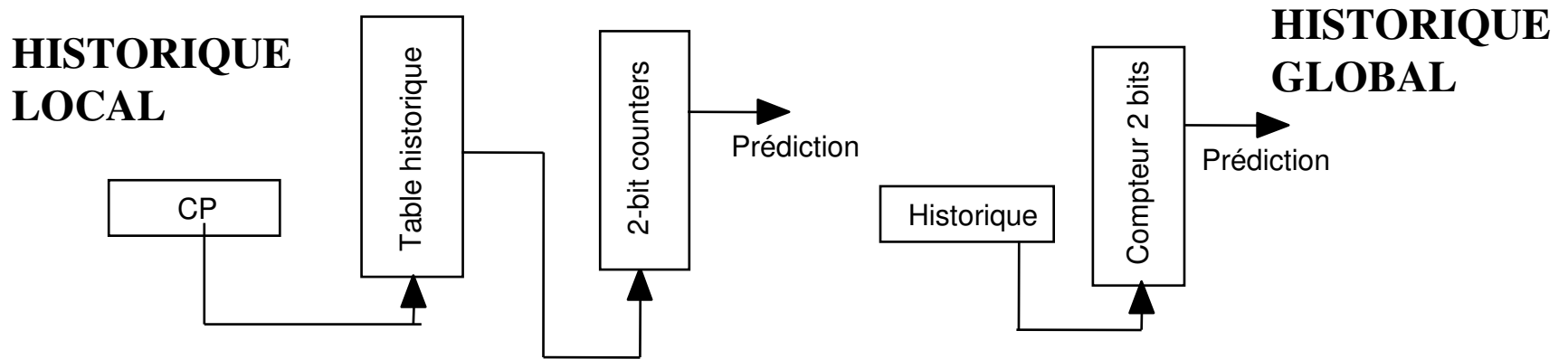
Les prédicteurs à deux niveaux

PREDICTEUR A 2 NIVEAUX

Prédicteur de branchement (2,2) → Nombre de bits d'historique
→ Nombre de bits par compteur

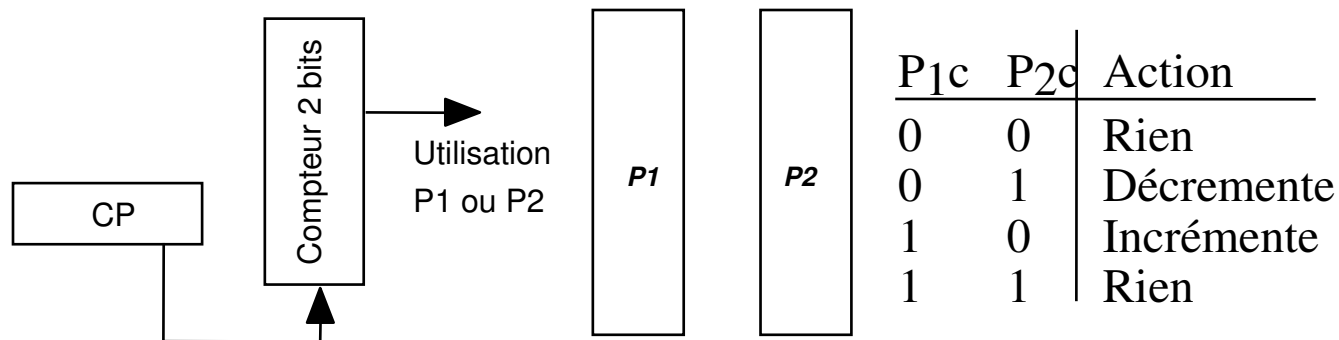


Local + Global



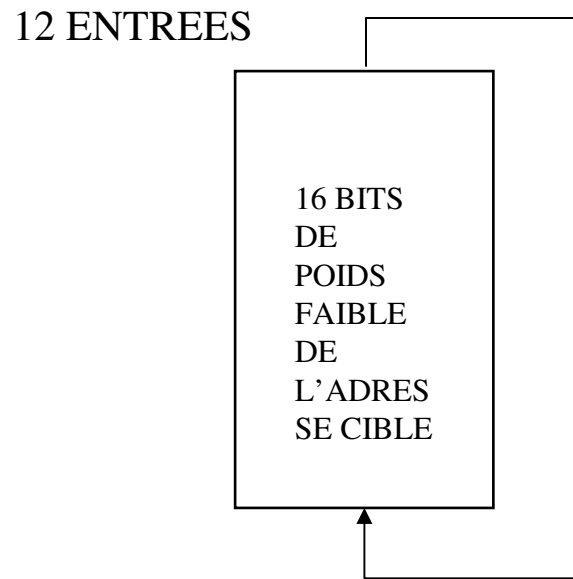
Les méta - prédicteurs

- Utilisation de deux prédicteurs différents
 - efficacité d'un prédicteur dépend de la nature du code
- Choix en dynamique du meilleur



Prédiction de l'adresse de retour

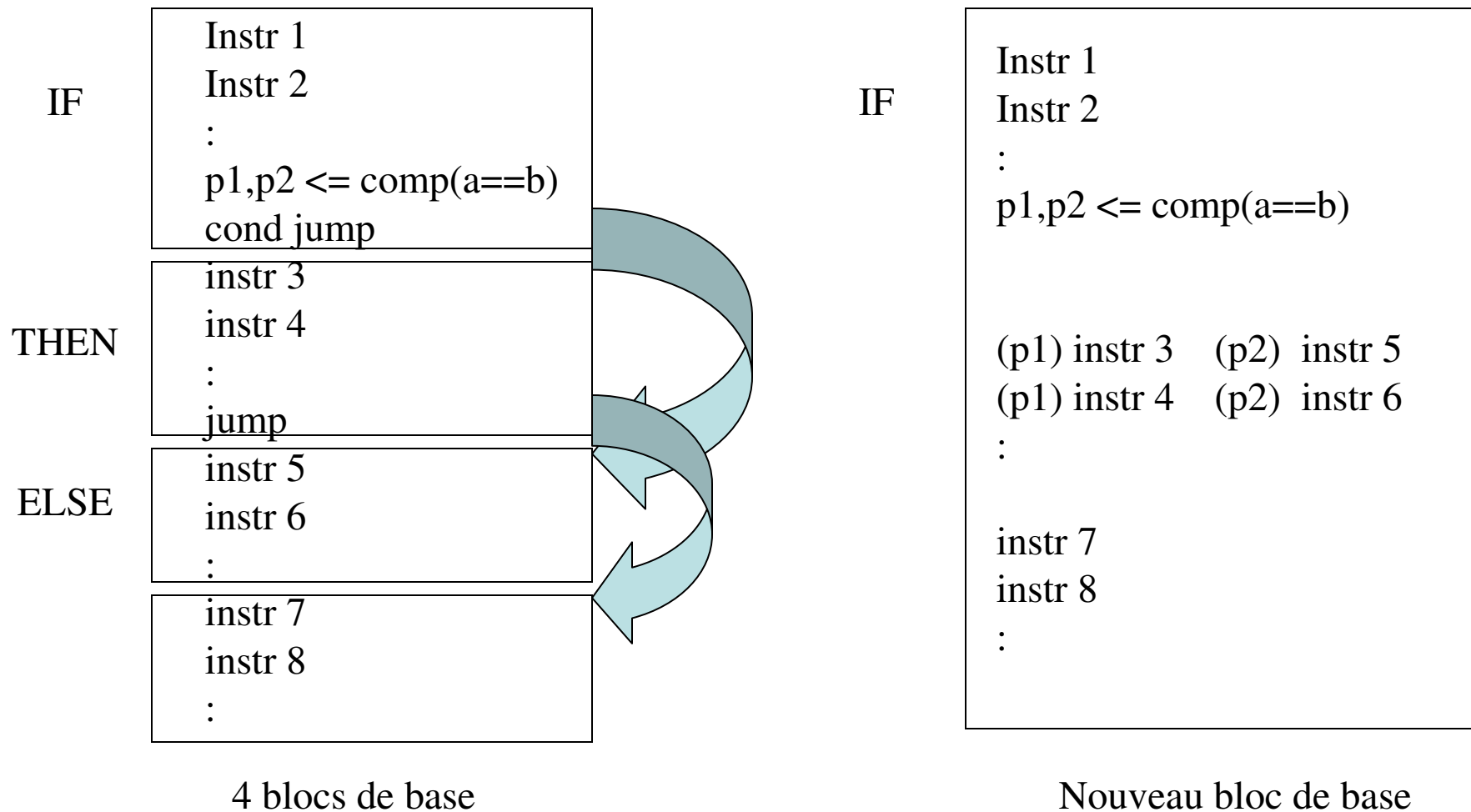
- Pile d'adresse de retour
ALPHA



Off.(15:14)	Cible prédite	Opération sur la pile
00 JMP	PC+4xdep(13:0)	
01 JSR	PC+4xdep(13:0)	PUSH PC
10 RET	Prediction stack	POP PC
11 JSR Co-routine	Prediction stack	POP ET PUSH PC

Les adresses de retour dépendent de l'endroit d'où les procédures sont appelées. Elles sont donc difficilement prévisibles.

L'utilisation des prédicats



Les jeux d'instructions avec prédicat

- ARM
 - Processeur RISC 32 bits (non VLIW)
- C6x
 - Processeur VLIW traitement du signal (Texas)
- Trimedia
 - Processeur VLIW traitement du signal (Philips)
- IA64
 - Processeur Intel usage général

Jeu d'instructions ARM

EXECUTION CONDITIONNELLE DE TOUTE INSTRUCTION

4

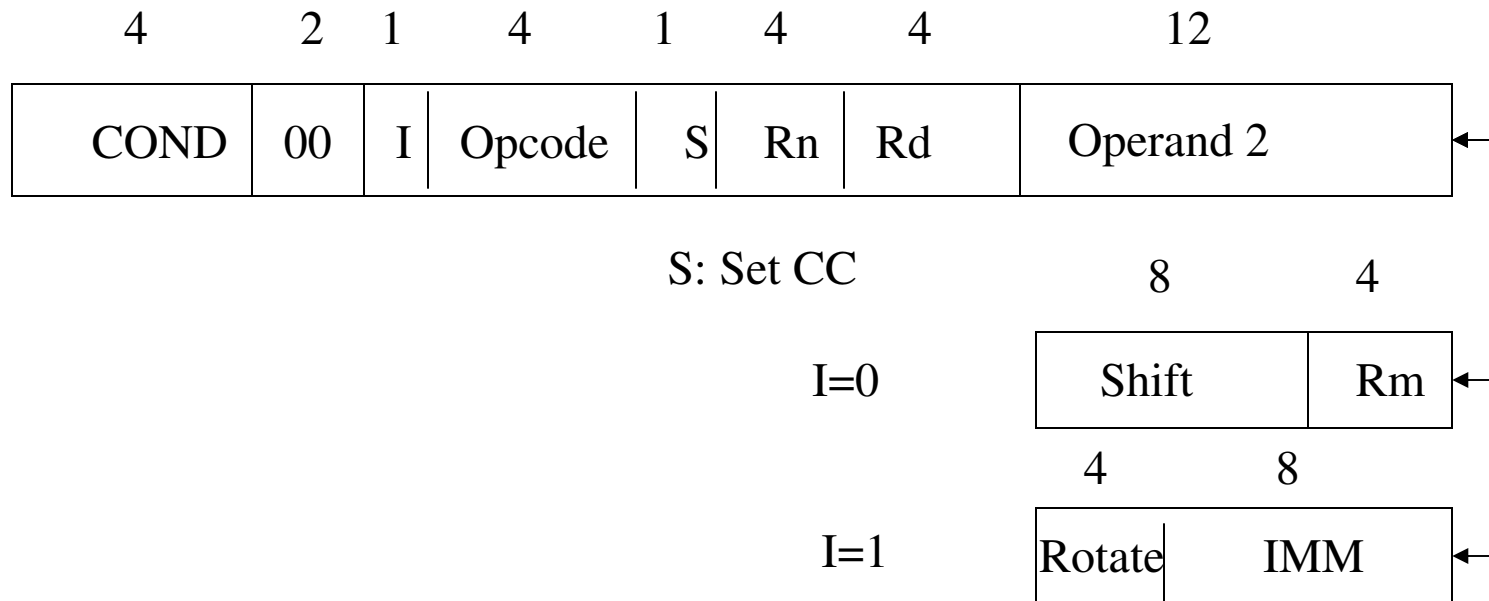
28

COND	
------	--

0000	EQ	1000	unsigned >
0001	NE	1001	unsigned <=
0010	CS	1010	≥
0011	CC	1011	<
0100	<0	1100	>
0101	>0 or =0	1101	≤
0110	overflow	1110	ALWAYS
0111	No overf	1111	NEVER

Instructions ARM

Opérations UAL



Branchement – Branchement avec lien



Code ARM

1. Valeur absolue – l'exécution conditionnelle évite un branchement

```
CMP  Rn,#0          ; compare with zero
RSBLT Rn,Rn,#0      ; and negate if Less Than
```

2. Convertir les lettres en minuscules sans changer le reste – évite deux branchements

```
CMP  Ra,'#A'        ; 'A' or later in ASCII?
RSBGES Rb,Ra,'#Z'   ; and 'Z' or before?
ORRGE Ra,Ra,#32     ; if so - make lower case (ORR=OR Register)
```

3. Décoder un nombre décimal

```
MOV  Rt,#0          ; start with zero
loop BL next_digit_Ra ; subroutine call for next digit in Ra
SUB  Ra,Ra,'#0'
CMP  Ra,#10         ; if unsigned Lower than 10 then 0...9
ADDLO Rt,Rt,Rt,LSL#2 ; if valid - multiply Rt by 5
ADDLO Rt,Ra,Rt,LSL#1 ; complete multiply by 10 and add in next digit
BLO  loop
```

CODE ARM (2)

. Calcul du PGCD avec l'algorithme d'Euclide.

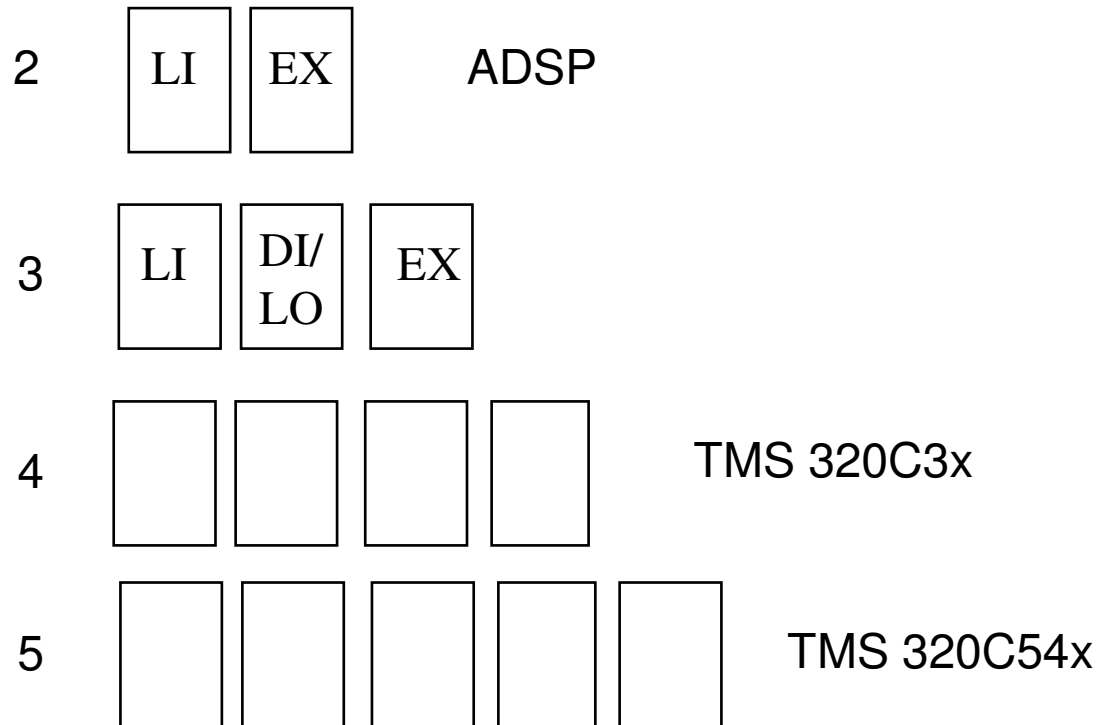
Programme C :

```
while (a != b)
{ if (a>b) a-=b;
  else  b-=a;
}
```

Codage ARM:

```
gcd CMP  Ra,Rb
    SUBGT Ra,Ra,Rb
    SUBLT Rb,Rb,Ra
    BNE  gcd
```

Les pipelines des DSP

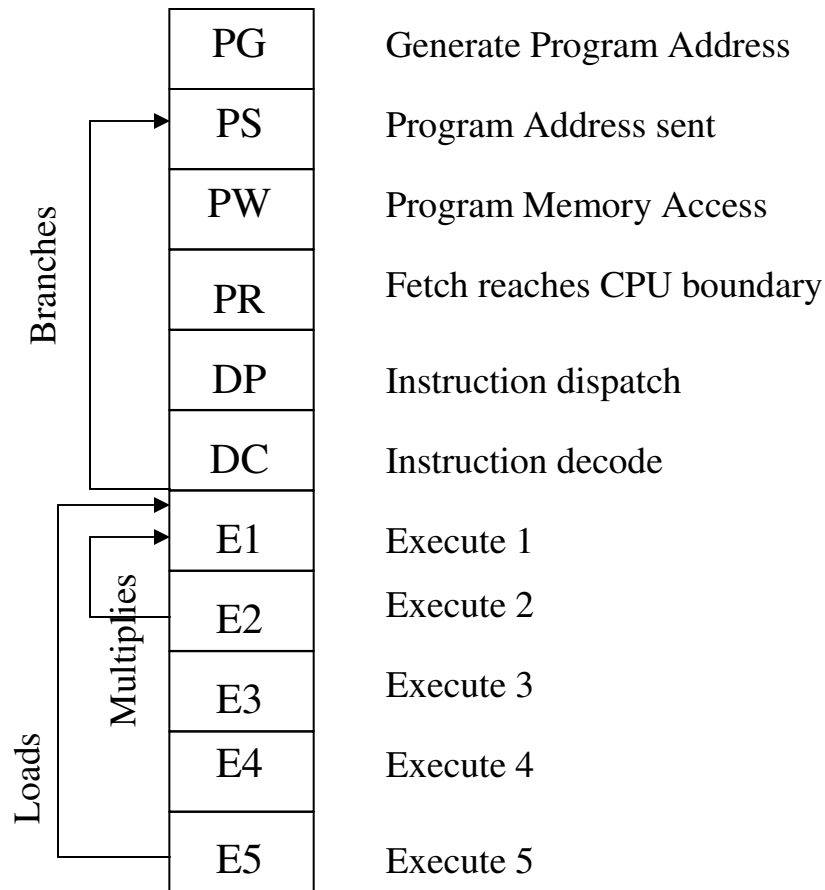


Pipeline du C54x



- PR : calcul adresse de l'instruction
- LI : Lecture de l'instruction
- DI : Décodage de l'instruction
- CA : Calcul adresse de l'opérande
- LD : Lecture de l'opérande
- EX : Exécution

Pipeline du C6x (TI)



BRANCH DELAY SLOT: 5 cycles
 LOAD DELAY SLOT : 4 cycles
 MULTIPLY DELAY SLOT : 1