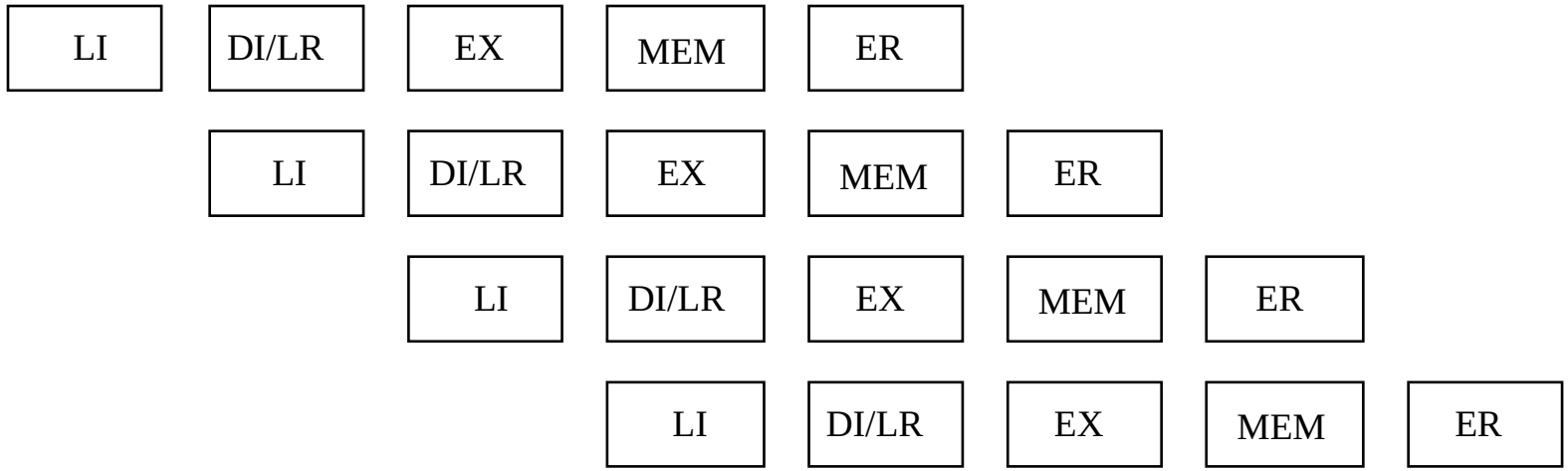
Architecture des ordinateurs
Problèmes du pipeline

Daniel Etiemble
de@lri.fr

Pipeline 1 instruction par cycle

Pipeline R2000-R3000



Latence : 5 cycles
Débit : 1 instruction par 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

LI DI LR EX_1 EX_2 ... EX_n ER

LI DI LR $EX_1$$EX_2$ ... Ex_n ER

Pipelinables

LI DI LR EX_1 EX_2 ...EX_n ER

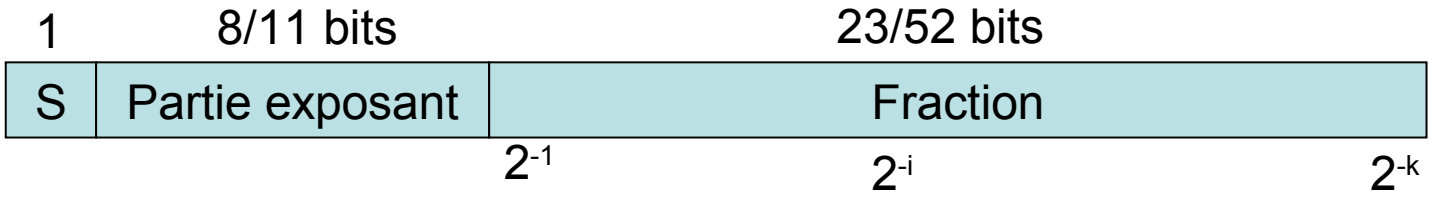
LI DI LR EX_1 EX_2 ... Ex_n ER

Non pipelinables

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 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 ≠ 0

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

DENORMALISES

La multiplication flottante

- Les différentes étapes
  - Multiplication des mantisses (PM = 1.f1 x 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 ≥ 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

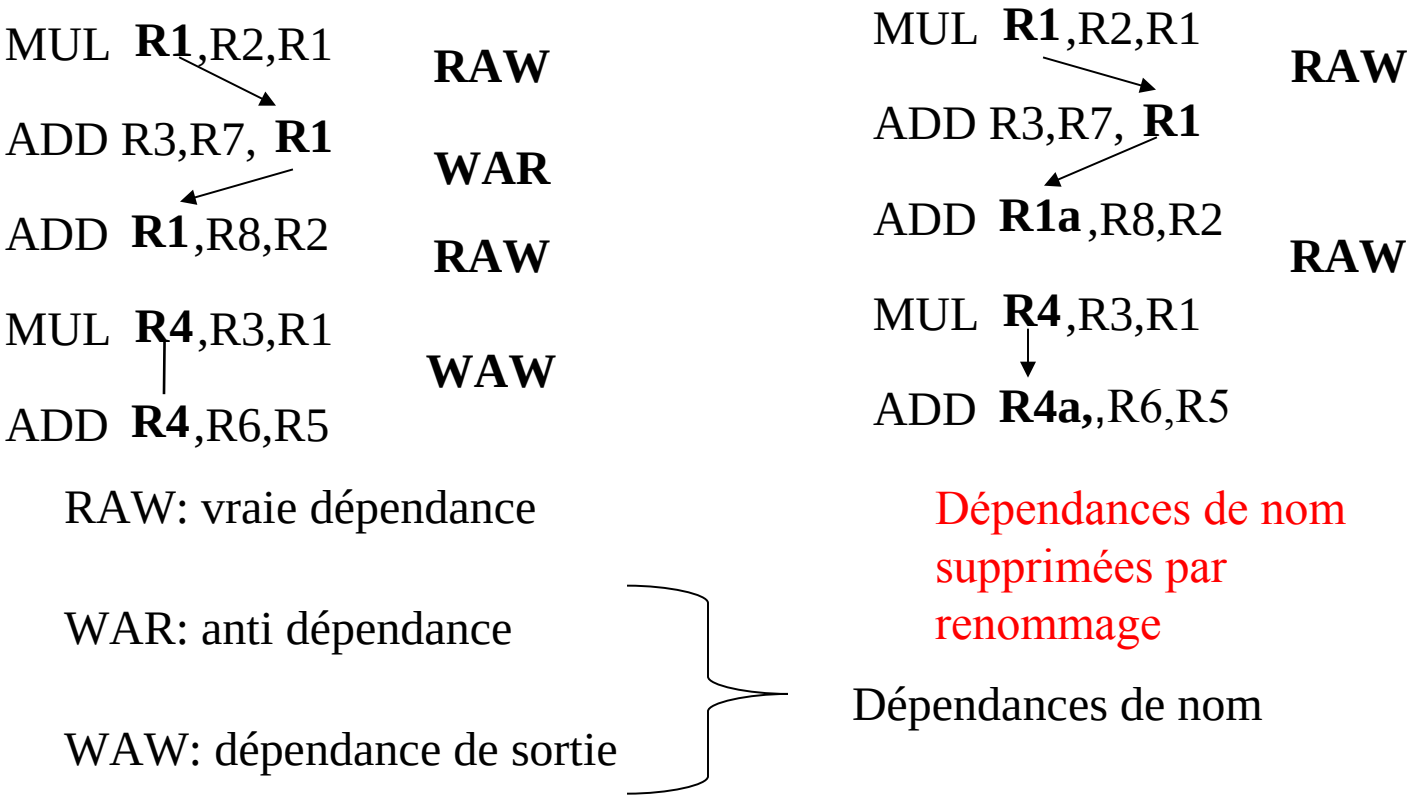
- Les différentes étapes
  - Comparer les parties exposant
  - Dénormaliser la mantisse de l'opérande avec le plus petit exposant pour obtenir des parties exposant égales
  - Faire addition/soustraction des mantisses
  - Renormalisation éventuelle
    - Addition
      - $1 \leq AM < 4$ car $1 \leq 1.f1 < 2$ 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

La division flottante

- Effectuée par itération successive
  - 1 ou 2 ou 4 bits calculés à chaque itération
    - Généralement algorithme SRT en base 4 (2 bits/itération)
  - Non pipelinable
- Opération « très » longue

Les dépendances de données

Exemple:



RAW: vraie dépendance

WAR: anti dépendance

WAW: dépendance de sortie

Dépendances de nom

Dépendances de nom supprimées par renommage

Traitement des dépendances

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

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)

Dest. \ *Source*	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

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

```
Boucle   1    LF F1, (R1)              ; charge X(i)
         2    LF 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,R1,4             ; adresse X(i+1)
         11   ADDI R2,R2,4             ; adresse Y(i+1)
         12   SF F2, -4(R2)            ; range Y(i)
         13   BNEQ R6, R0, Boucle      ;si I<N-1, branchement
```

13 cycles (4 cycles sont perdus)

Déroulage de boucle

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

```
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];
        }
```

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)

Pipeline logiciel (principe)

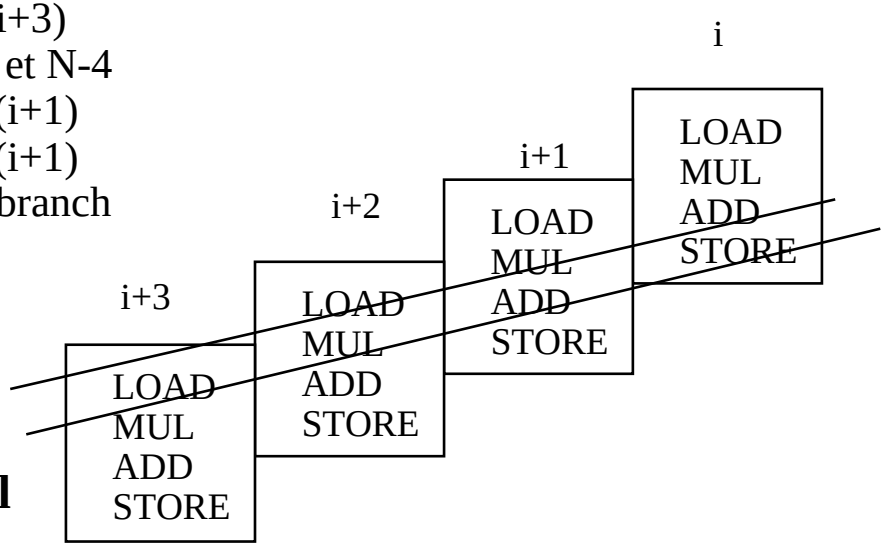
SAXPY **PROLOGUE**

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

```
LOOP    1    LF  F4, 0(R2)        ; range Y(i)
        2    ADDD F4,F2,F3        ; a * X(i+1) + Y(i+1)
        3    MULTD F3, F0, F1     ; a * X(i+2)
        4    LF  F1, 24(R1)       ; charge X(i+3)
        5    LF 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



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 »)

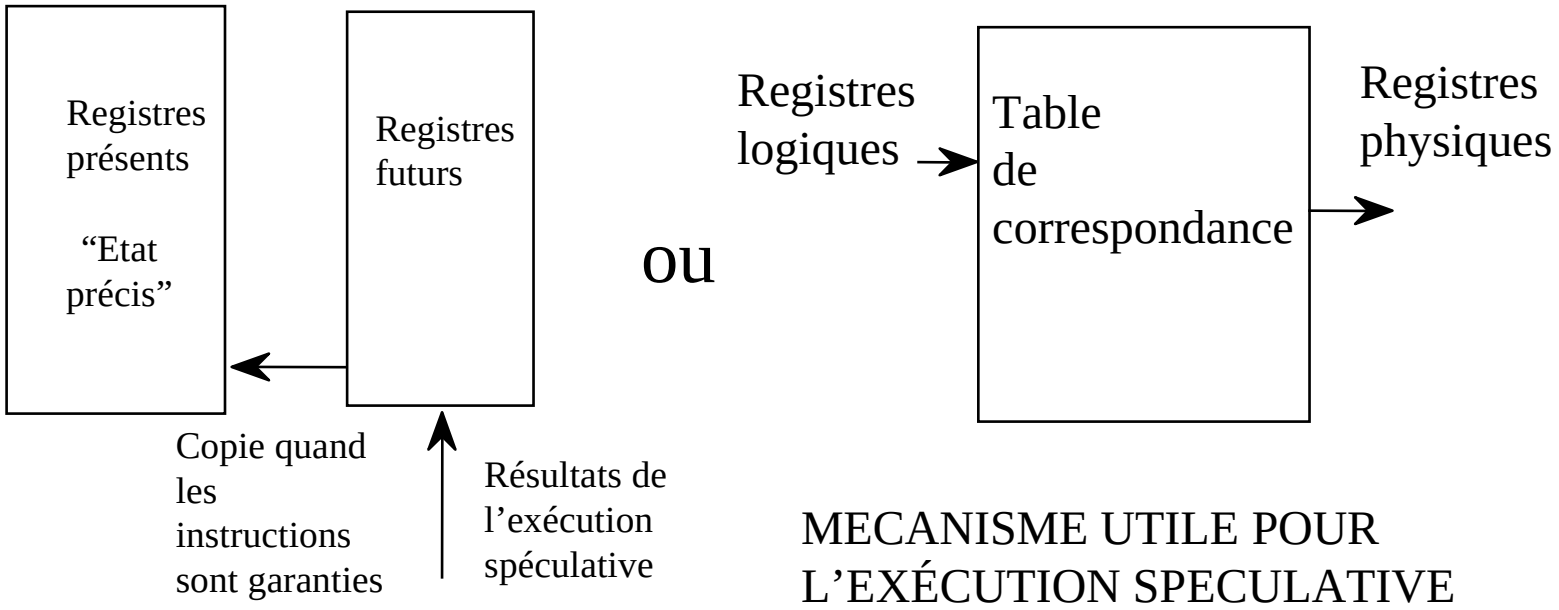
LI DI LR EX_1 EX_2 ...EX_n ER — DIV (instruction multi-cycles)

LI DI LR MEM ER — Instruction 1 cycle

Renommage des registres

Renommage de registres

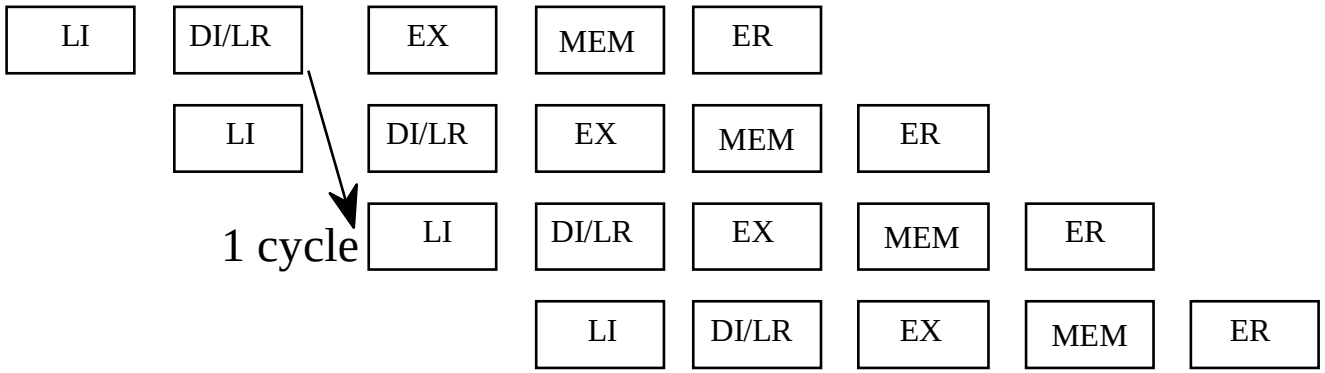
Registres logiques (Jeu d'instructions)
Registres physiques



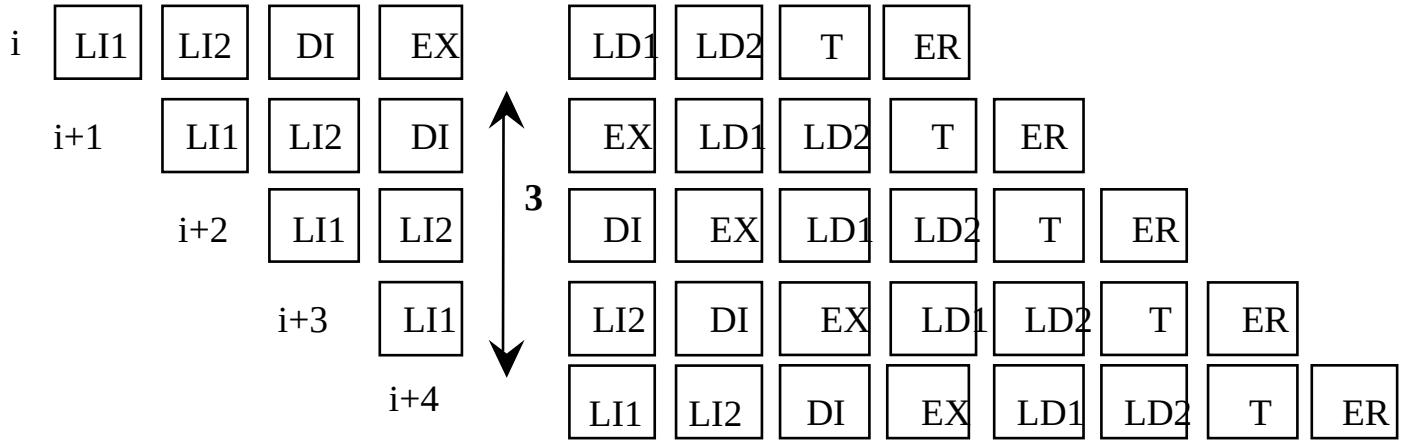
MECANISME UTILE POUR L'EXÉCUTION SPECULATIVE

Les délais de branchement

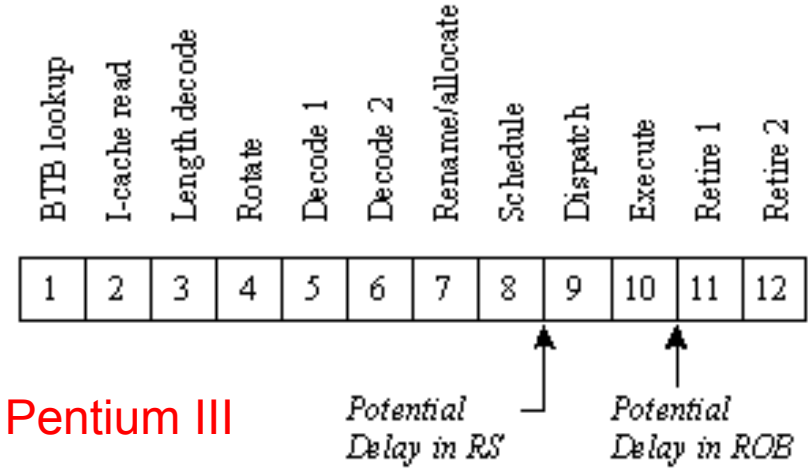
MIPS R2000

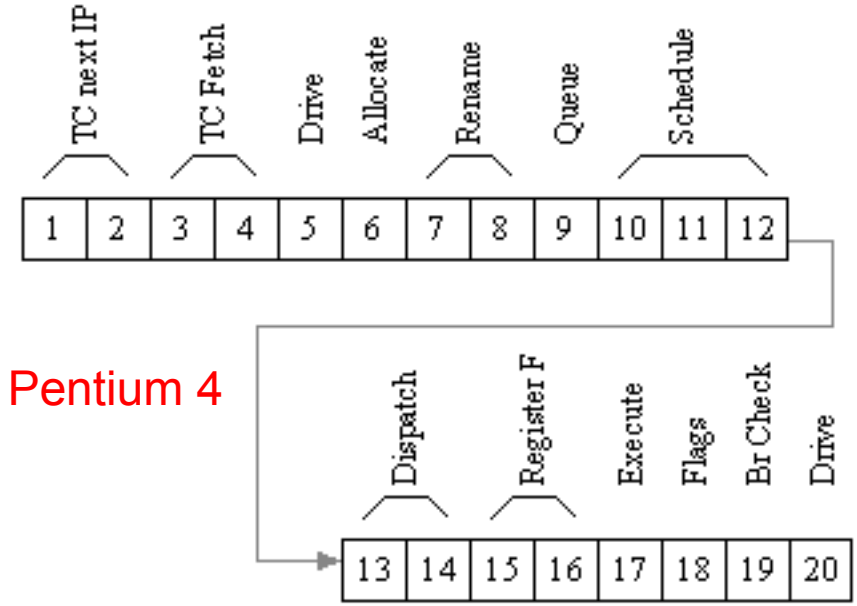


MIPS R4000



Pipelines Pentium III et Pentium 4





Les instructions de contrôle

- Fréquence
  - 10 à 25% de toutes les instructions exécutées
- Les différentes instructions de contrôle
  - Les branchements inconditionnels (B déplacement)
  - Les sauts indirects (JMP Ri)
  - Les retours de procédures (RET ou JMP R31)
  - Les branchements inconditionnels (B_{cond} déplacement)

Les branchements conditionnels représentent de l'ordre de 75% des instructions de contrôle

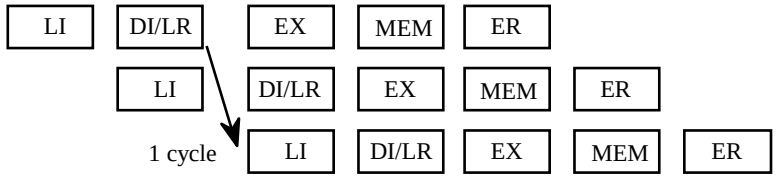
Le traitement des branchements conditionnels

- Ne rien faire
  - Pénalité de branchement si branchement pris
- Branchements retardés
  - Un cycle (MIPS, SPARC)
    - Non compatible avec le superpipeline
  - Plusieurs cycles
    - Utilisé dans les processeurs VLIW de traitement du signal (exemple : TMS 320C6x) avec technique de pipeline logiciel
- Prédiction de branchements
- Suppression (de la plupart) des branchements
  - Instructions avec prédicat.

Branchements retardés

- Branchements retardés
  - L'instruction qui suit le branchement est exécutée avant le branchement
  - Réordonnancement par le compilateur

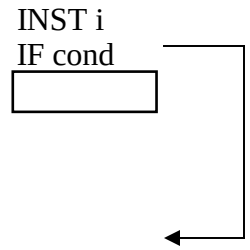
```
        LD R1, (R4+8)              LD R1, (R4+8)
        LD R2, (R5+8)              LD R2, (R5+8)
        ADD R3,R1,R2               ADD R3,R1,R2
        ST R3, (R6+8)              BR L1
        BR L1                      ST R3, (R6+8)
        NOP                        ...
        ...                        ....
        ....                       .....
L1 :    .....              L1 :    ......
        ......
```



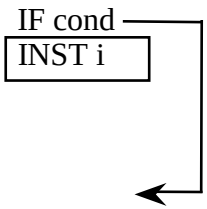
Branchements retardés (2)

Schémas de réordonnancement

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



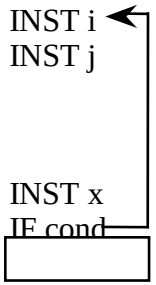
a) Initial



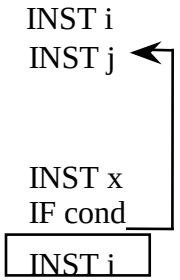
a) Final

b) **Instruction cible**
Branchement très probablement pris

(Branchement de boucle)

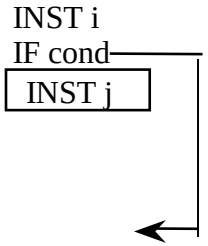


b) Initial

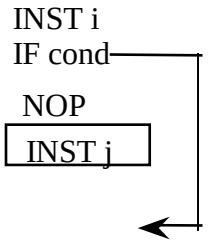


b) Final

c) **Instruction après le branchement**



c) Initial



c) Final

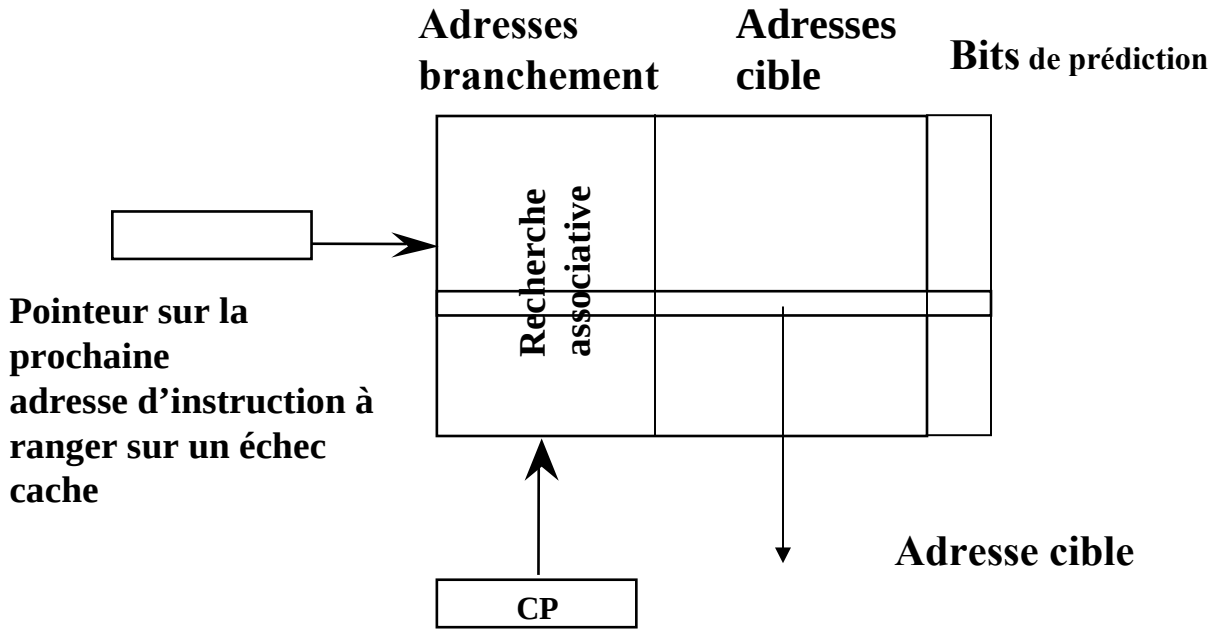
Branchements avec annulation

L'instruction suivante est exécutée si le branchement est pris

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

Les branchements non retardés (BTB)

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

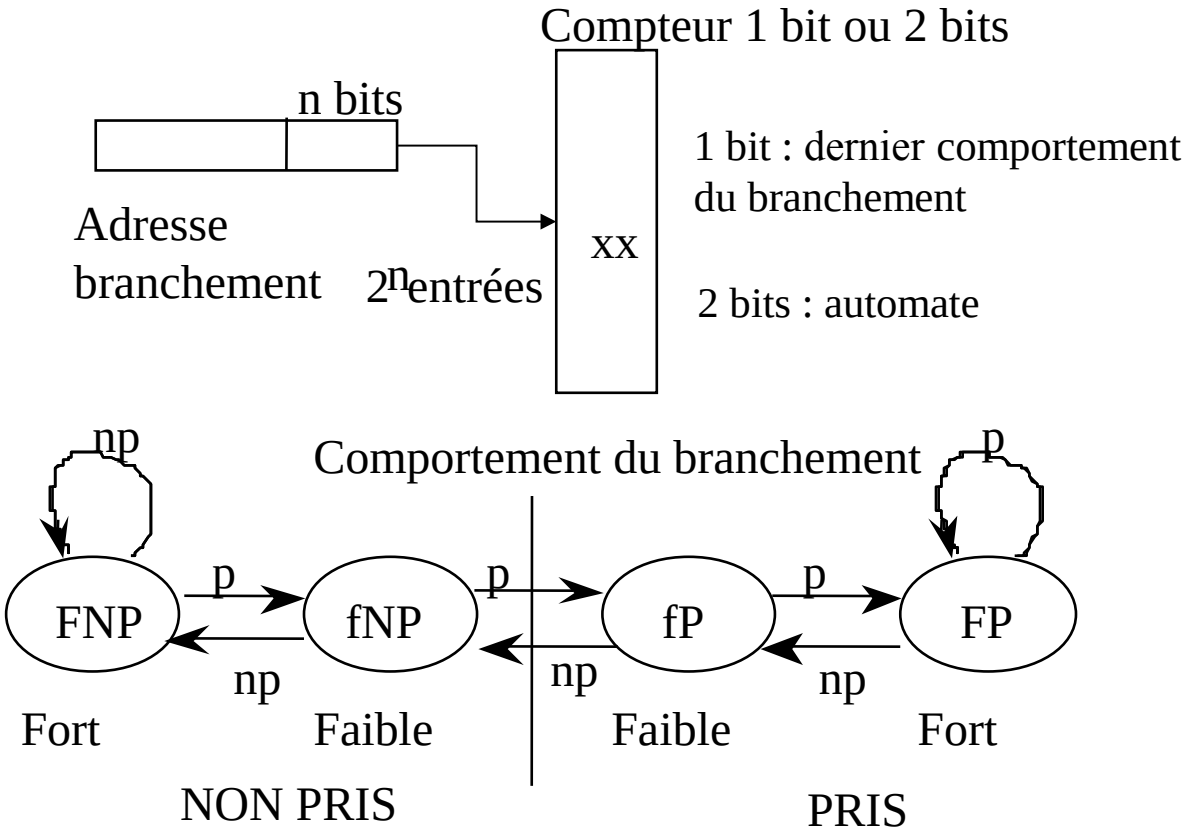


La prédiction de branchement

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

Les prédicteurs dynamiques locaux

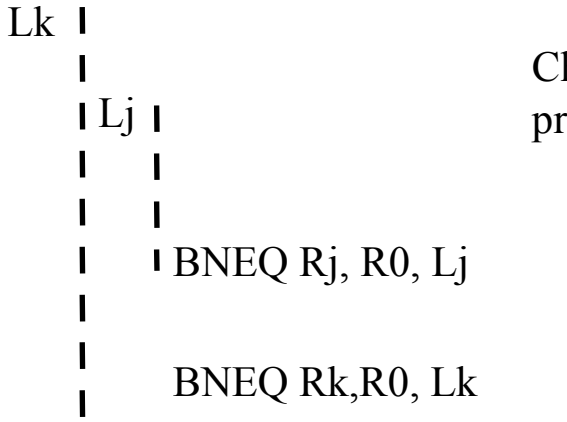
- Prédicteurs 1 bit et 2 bits



Prédiction des programmes flottants

- Exemple

```
For (k=0; k<N; k++)
    For (j=0; k<N; k++)
        C(j)  = C(j) + A(k) * B(k,j)
```



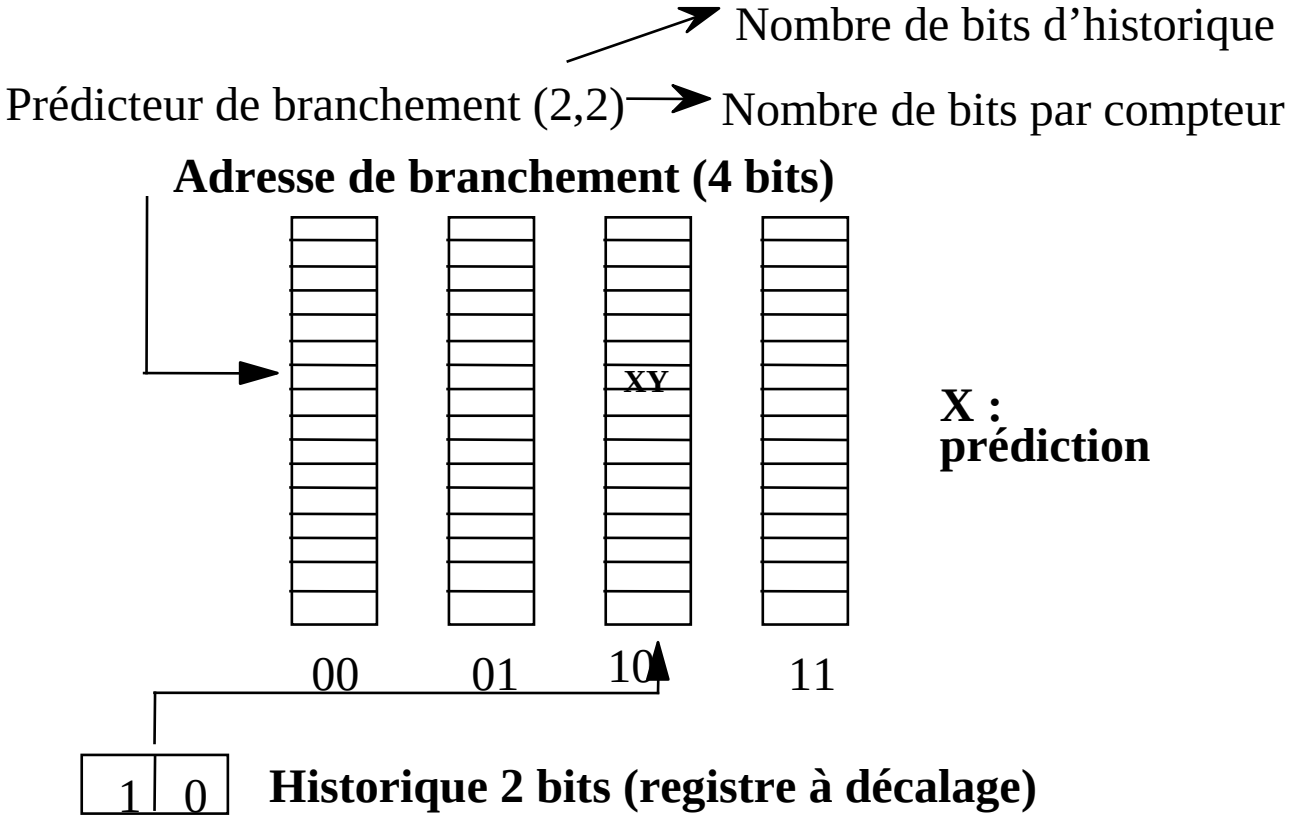
Chaque branchement est pris (n-1) fois et non pris 1 fois

Prédicteur 1 bit : 2 mauvaises prédictions sur n

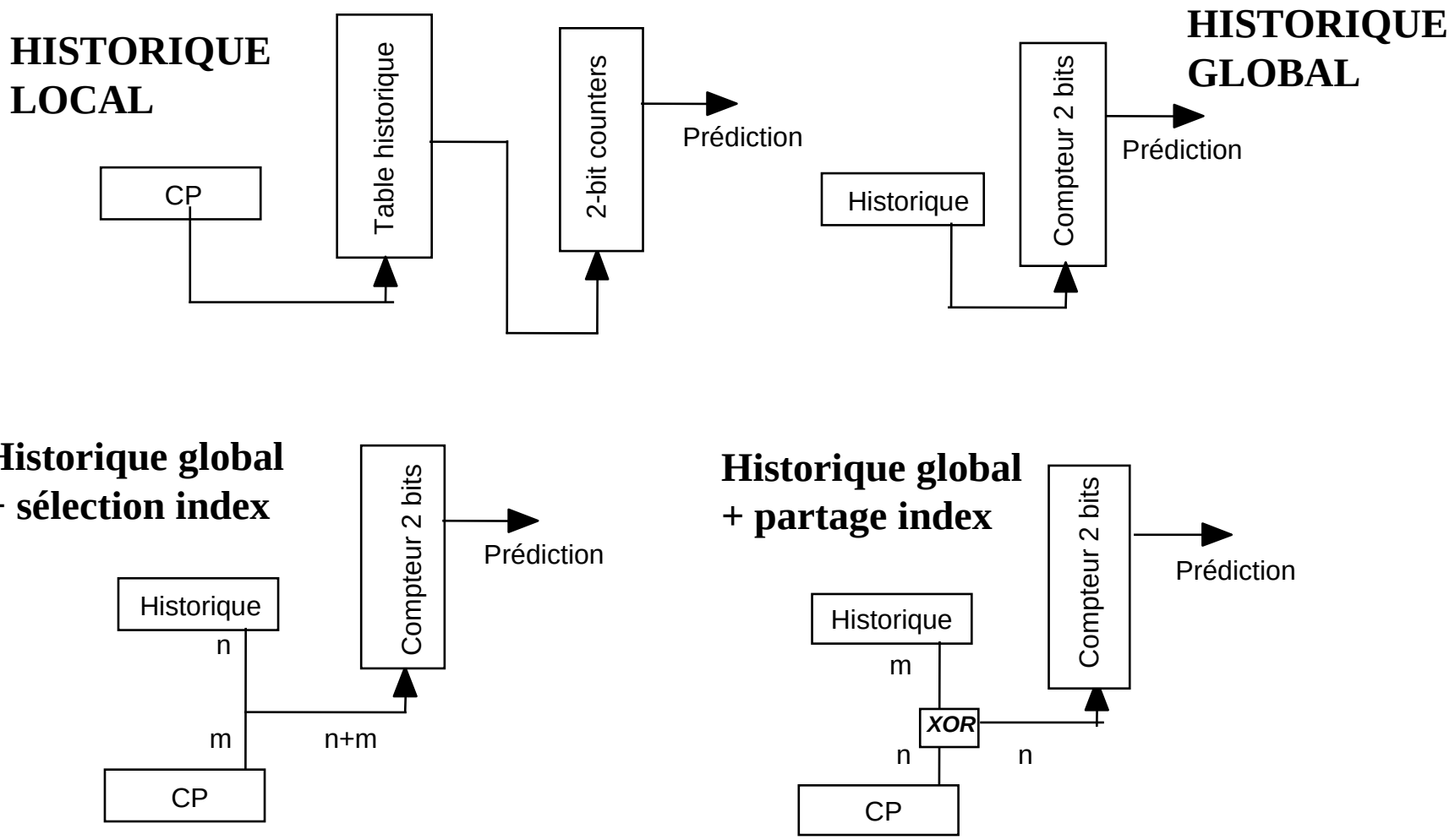
Prédicteur 2 bits : 1 mauvaise prédiction sur n

Les prédicteurs à deux niveaux

PREDICTEUR A 2 NIVEAUX

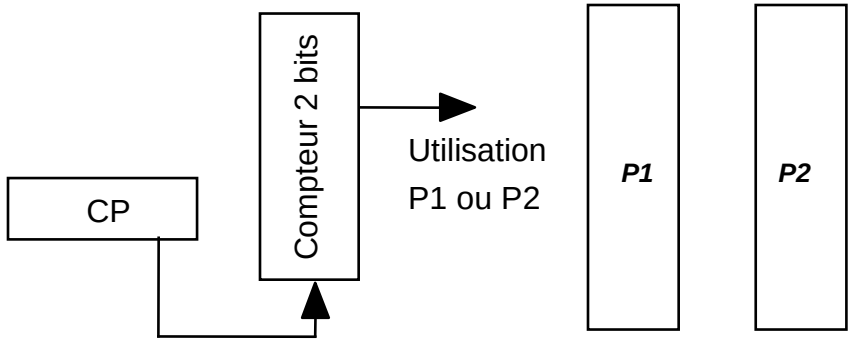


Local + Global



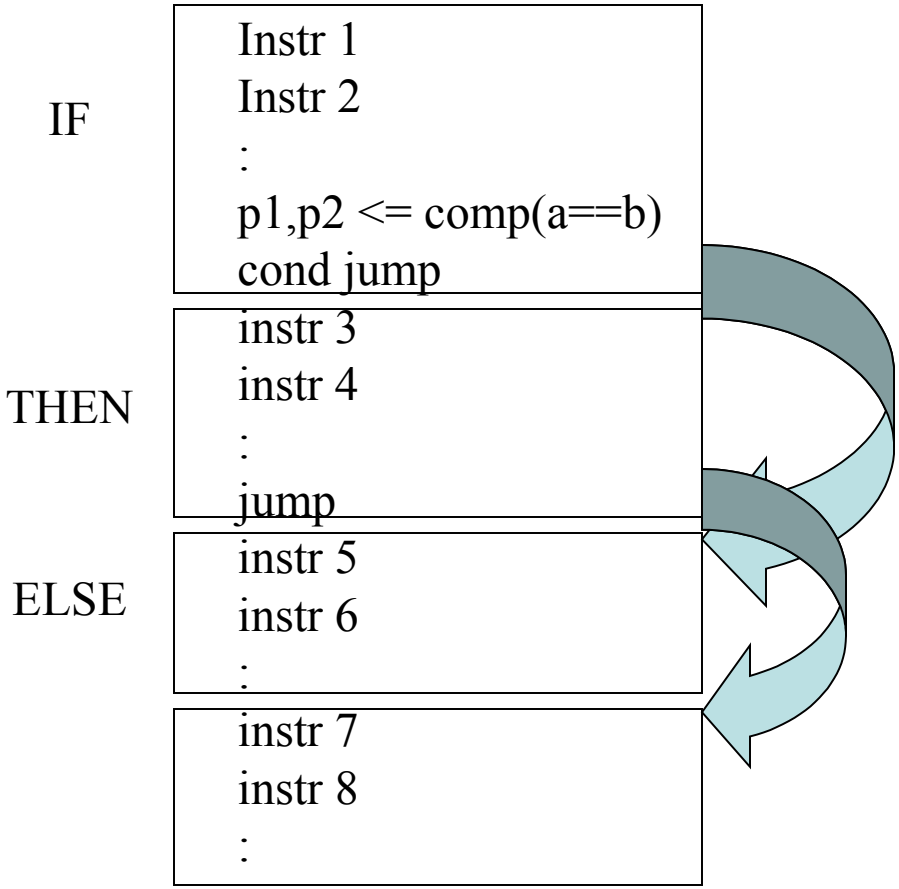
Les méta-prédicteurs

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



P_1c	P_2c	Action
0	0	Rien
0	1	Décremente
1	0	Incrémente
1	1	Rien

L'utilisation des prédicats



4 blocs de base

IF

```
Instr 1
Instr 2
:
p1,p2 <= comp(a==b)


(p1) instr 3    (p2)  instr 5
(p1) instr 4    (p2)  instr 6
:

instr 7
instr 8
:
```

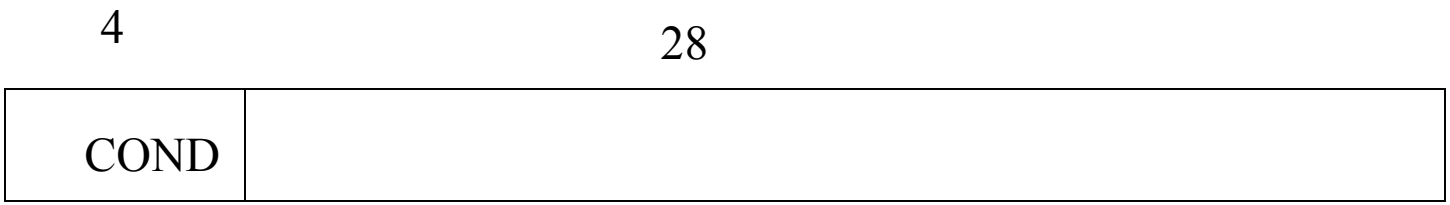
Nouveau bloc de base

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



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

CODE ARM

Calcul du PGCD (algorithme d'Euclide)

Programme C

```
while (a!=b)
ef (a>b) a-=b;
else b-=a
```

Code ARM

```
Boucle : CMP   Ra,Rb
         SUBGT Ra,Ra,Rb
         SUBLT Rb,Rb,Ra
         BNE   Boucle
```

Les branchements conditionnels ont disparu (et la prédiction de branchement)

Exécution successive des deux branches de l'alternative

Caractéristiques des instructions prédiquées

- Exécution
  - Prédicat « vrai » : instruction exécutée
  - Prédicat « faux » : instruction NOP exécutée
  - Dans les deux cas, une instruction est exécutée
- Intéressant si plusieurs instructions sont exécutables en parallèle
  - Sinon, exécution successive des branches « vrai » et « faux »

IF

```
instr 1
instr 2
:
p1,p2 <= comp(a==b)


(p1) instr 3    (p2)  instr 5
(p1) instr 4    (p2)  instr 6
:

instr 7
instr 8
:
```