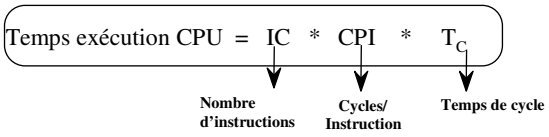
Pipelines scalaires

Daniel Etiemble
de@lri.fr

PERFORMANCE

- L'équation



- 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

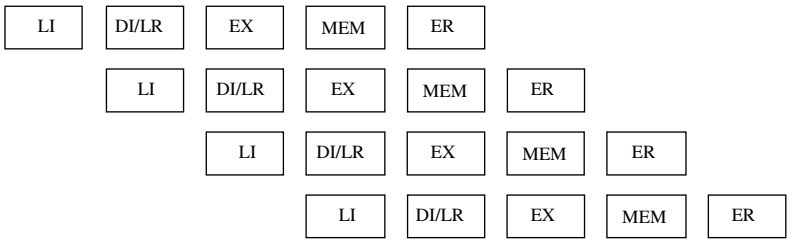
Instructions UAL	*Instructions Mémoire*	*Instructions Branchement*
Lecture instruction	Lecture instruction	Lecture instruction
Incrémentation CP	Incrémentation CP	Incrémentation CP
Décodage de l'instruction	Décodage de l'instruction	Décodage de l'instruction
Lecture des opérandes	Calcul de l'adresse mémoire	Calcul de l'adresse de branchement
Exécution	Accès mémoire	Exécution
Ecriture du résultat	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

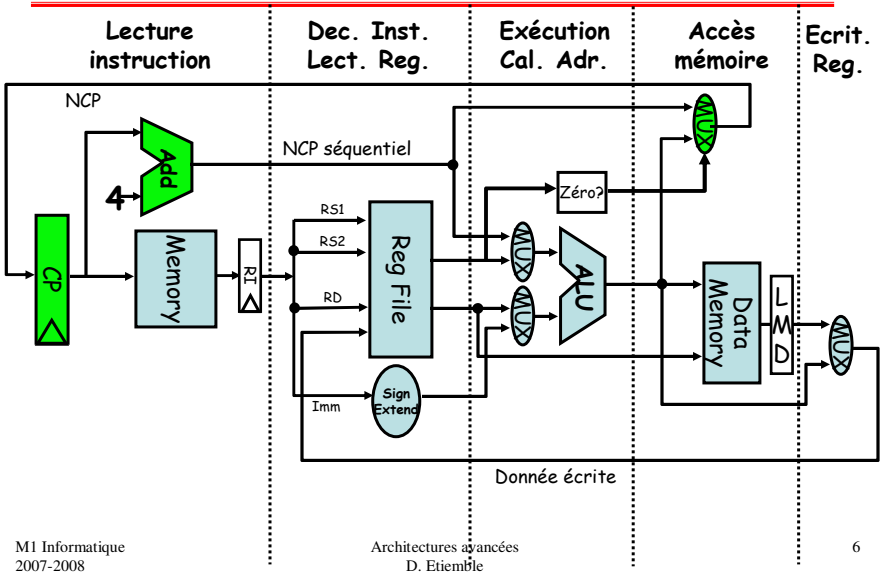
Pipeline 1 instruction par cycle

Pipeline R2000-R3000

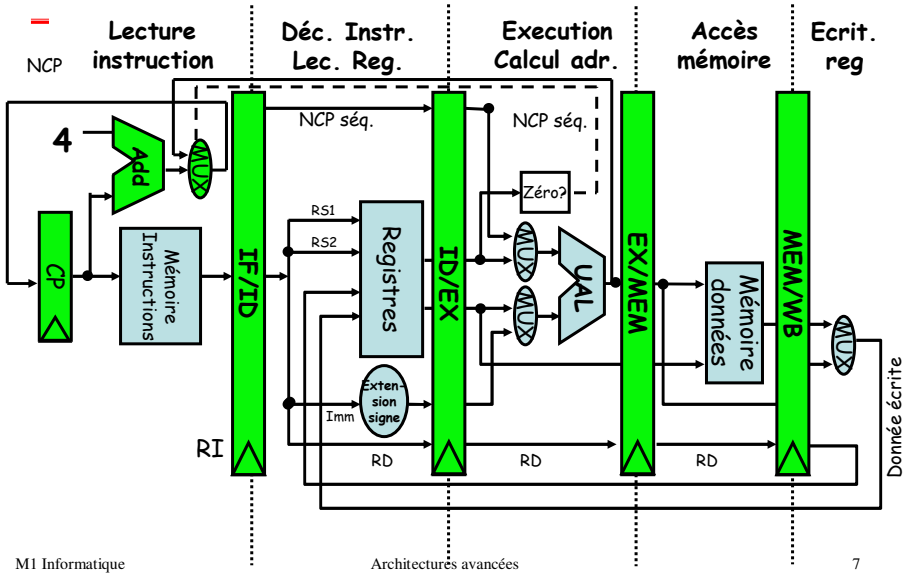


Latence : 5 cycles
Débit : 1 instruction par cycle

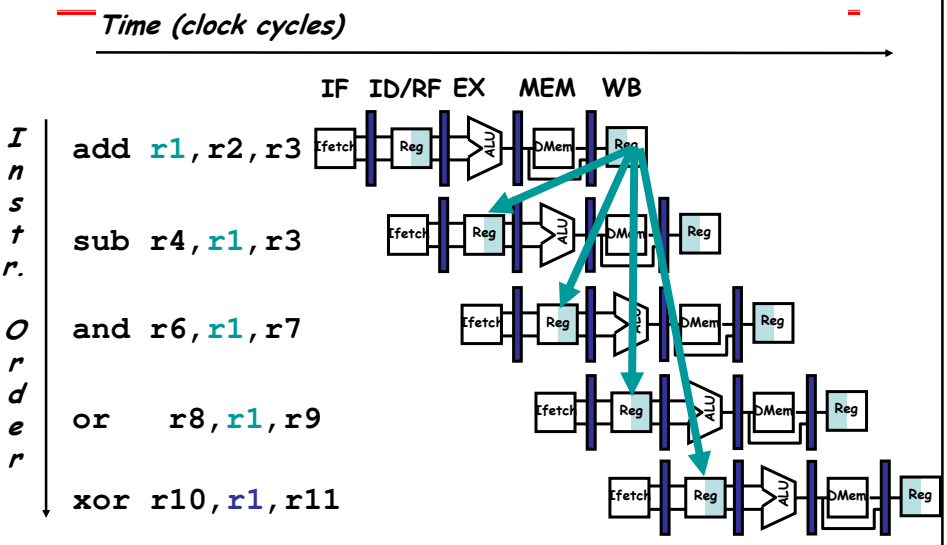
Exécution scalaire MIPS (non pipelinée)



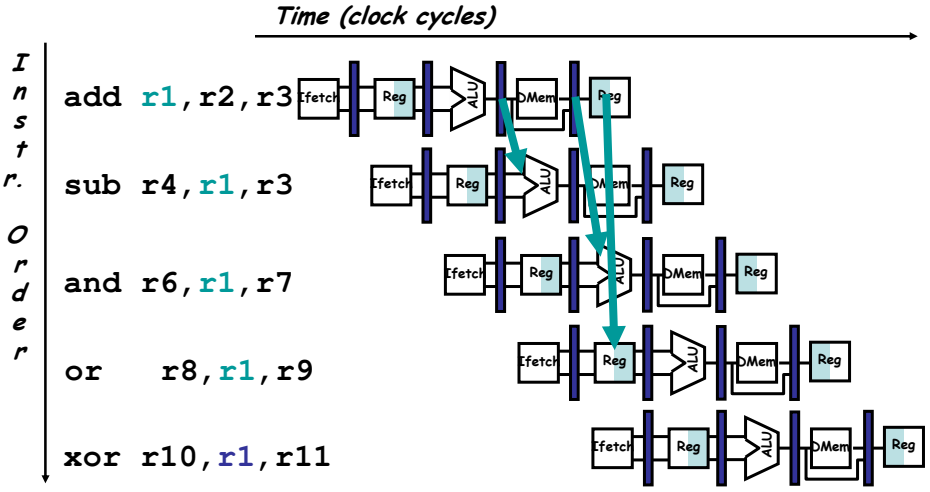
Exécution scalaire MIPS (pipelinée)



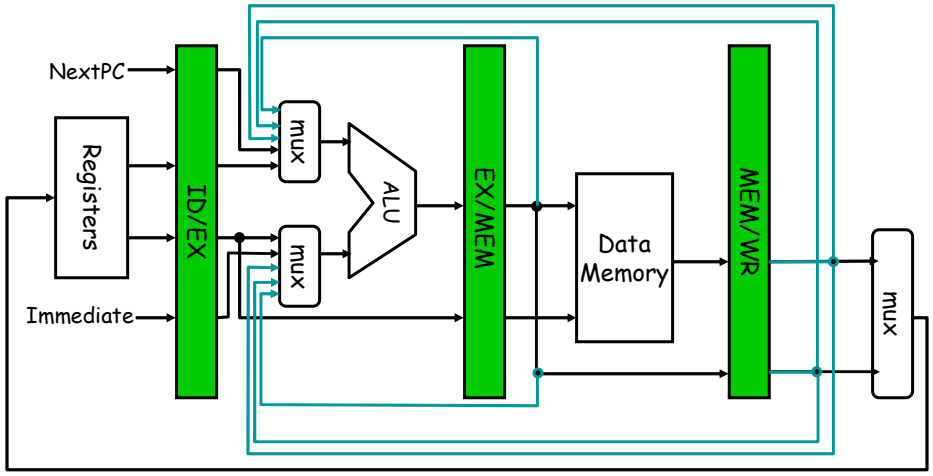
Aléas de données



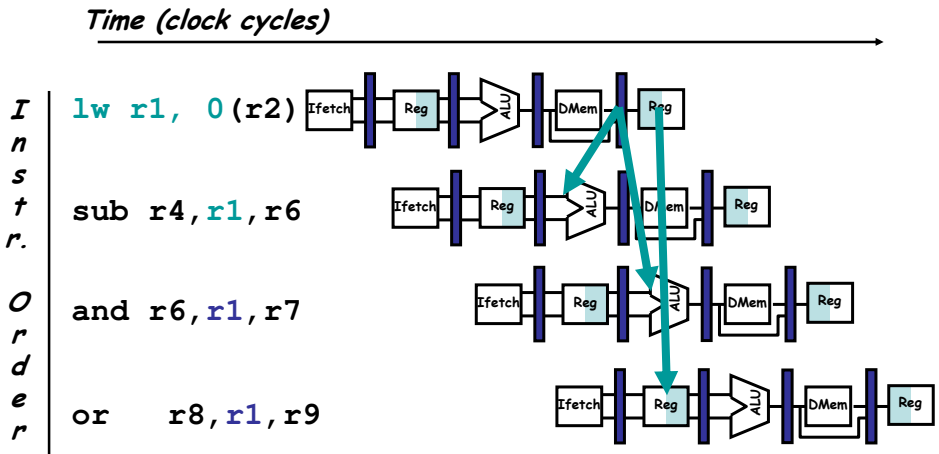
Envoi pour éviter les aléas



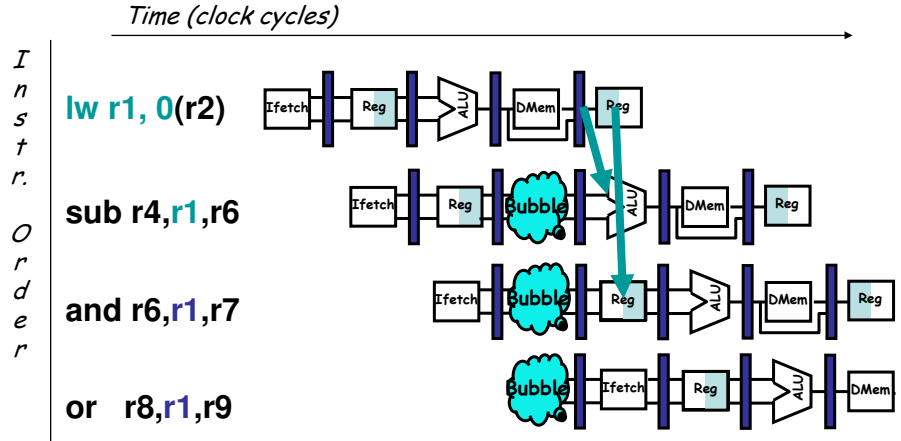
Matériel pour l'envoi



Aléas de données incontournables

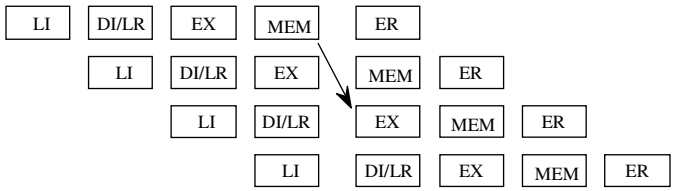


Aléas de données incontournables

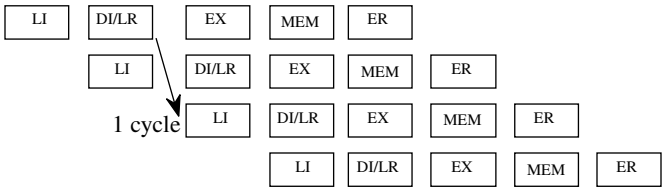


Les délais liés au pipeline

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

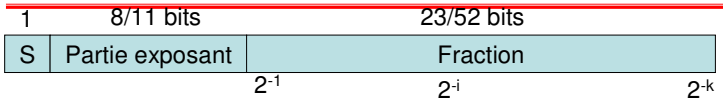
LI DI LR EX_1 EX_2 ... EX_n ER
LI DI LR EX_1 EX_2 ... Ex_n ER
Pipelinables

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

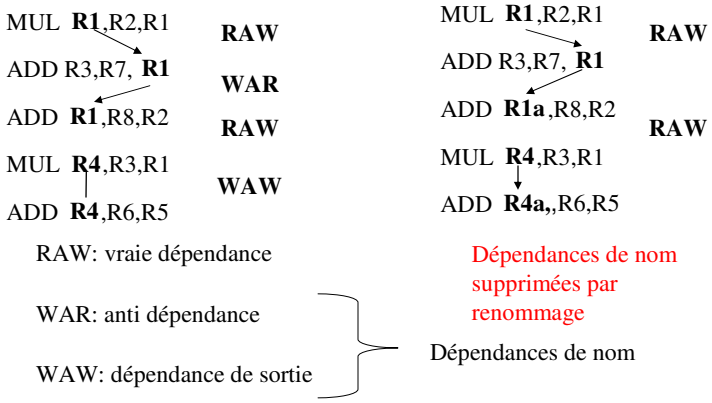
- Les différentes étapes
  - Multiplication des mantisses (PM = 1.f1 x 1.f2)
    - 1≤ PM < 4 car 1≤ 1.f1 < 2 et 1≤ 1.f2 < 2
  - Addition des Parties exposants
    - PE = PE1 + PE2 -127
  - Renormalisation éventuelle
    - Si PM ≥ 2, il faut faire PM =PM/2 (décalage) et PE=PE+1
- Opération « longue »
  - 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

Les dépendances de données

Exemple:



RAW: vraie dépendance

WAR: anti dépendance

WAW: dépendance de sortie

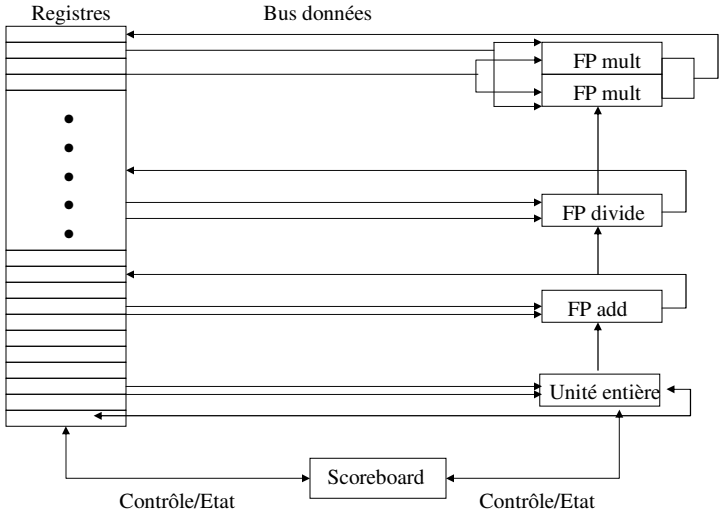
Dépendances de nom

Dépendances de nom supprimées par renommage

Traitement des dépendances

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

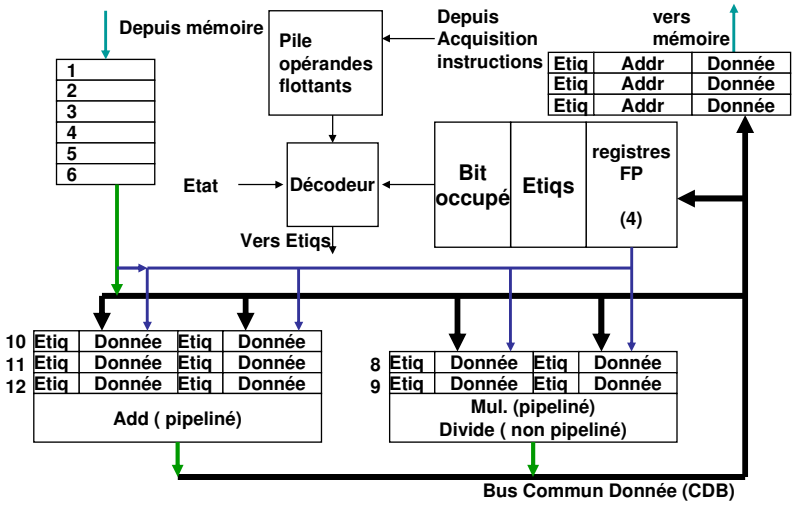
Le tableau de marques



Les trois étapes du « tableau de marques »

1. Etat de l'instruction : indique la phase en cours de l'instruction
   1. Lancement
   2. Lecture des opérandes
   3. Exécution
   4. Ecriture du résultats
2. Etat des unités fonctionnelles : indique l'état de l'unité fonctionnelle (UF). Il y a 9 champs dans chaque unité fonctionnelle
   - Occupé : indique si l'unité est occupée ou non.
   - Op : Opération à réaliser dans l'unité (par ex: + ou –)
   - Fi : Registre destination
   - Fj, Fk : Numéros des registres source
   - Qj, Qk: Unités fonctionnelles qui alimentent les registres sources Fj, Fk
   - Rj, Rk : Drapeaux indiquant quand Fj, Fk sont prêts
3. Etat des registres résultat : indique quelle unité fonctionnelle écrira dans chaque registre, s'il y a une écriture. Un blanc indique qu'aucune instruction en cours n'écrira dans ce registre.

Le contrôle des aléas : Tomasulo



Les composantes des stations de réservation

- Les stations de réservation
  - Op : opération à effectuer dans l'unité (par ex: + or –)
  - Vj, Vk : valeurs des opérandes source
    - Les tampons de rangement ont un champ V : le résultat à ranger
  - Qj, Qk : Stations de réservation alimentant les registres source (valeur à écrire)
    - Note : Pas de drapeaux Prêt comme dans le tableau de marques ; Qj,Qk=0 => prêt
    - Les tampons de rangement ont seulement Qi pour la SR produisant le résultat
  - Occupé : indique que la station de réservation ou l'UF est occupée
- Banc de registre et tampon de rangement
  - Qi : champ étiquette
  - occupé : actuellement utilisé
- Etat des résultats dans les registres :
  - Indique quelle unité fonctionnelle va écrire dans chaque registre. Un blanc lorsque aucune instruction en cours n'écrira dans le registre.

Les trois étapes de l'algorithme de Tomasulo

- 1. Lancement : obtenir l'instruction de la file des instructions flottantes (dans l'ordre)
  - Si une station de réservation est disponible (pas d'aléa structurel), le contrôle lance l'instruction et envoie les opérandes (renomme les registres) depuis le banc de registres (si les opérandes y sont) à la SR. Autrement, suspension à cause de l'aléa structurel
- 2. Exécution : opération sur les opérandes (EX) (peut être non ordonné)
  - Quand les deux opérandes sont prêts, exécution
  - S'ils ne sont pas prêts, attendre le résultat sur le Bus Commun de Données (CDB)
  - Les chargements effectuent ces deux étapes
  - Traite les aléas RAW
- 3. Ecriture du résultat : termine l'exécution (ER) (peut être non ordonné).
  - Ecrire sur le CDB pour toutes les unités en attente. Marquer la SR comme disponible.
  - Le renommage évite les aléas WAW et WAR.

- Les bus
  - Bus de données normaux : données + destination (bus "vers")
  - Bus commun de données : données + source (bus "depuis")
    - 64 bits de données et 4 bits pour l'adresse source de l'Unité Fonctionnelle
    - Ecriture si l'Unité Fonctionnelle espérée est là (produit le résultat)
    - Effectue la diffusion

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

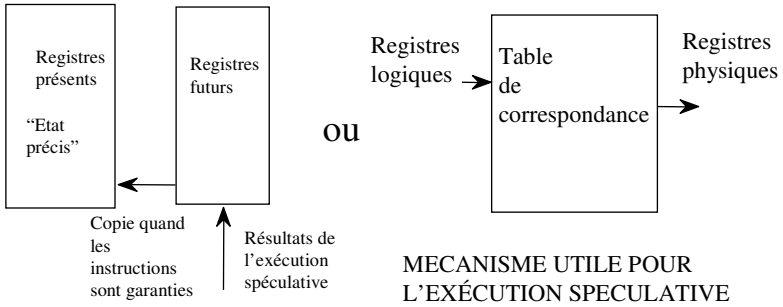
- Pipeline et exceptions propres.
- Terminaison dans l'ordre des instructions
  - Les instructions multi-cycles obligent les instructions suivantes 1 cycle à attendre, même
    - s'il n'y a pas de dépendances de données
    - si elles ne provoquent pas d'exceptions
- Terminaison non ordonnées des instructions
  - 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

Renommage des registres (2)

a)

```
L2 : move r3,r7
WAW  lw   r8, (r3)
     add  r3, r3, 4
     lw   r9, (r3)
     ble  r8, r9, L3
```

RAW

WAR

b)

```
L2 : move  r3,r7
     lw    r8, (r3)
     add   r3a, r3, 4
     lw    r9, (r3)
     ble   r8, r9, L3
```

RAW

Avant : add r3,r3,4

r0	R8
r1	R7
r2	R5
r3	R1
r4	R9

Après : add R2,R1,4

r0	R8
r1	R7
r2	R5
r3	R2
r4	R9

Liste des registres disponibles

R2 R6 R13

R6 R13

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)

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

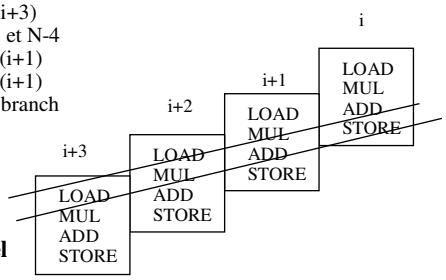
SAXPY **PROLOGUE**

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

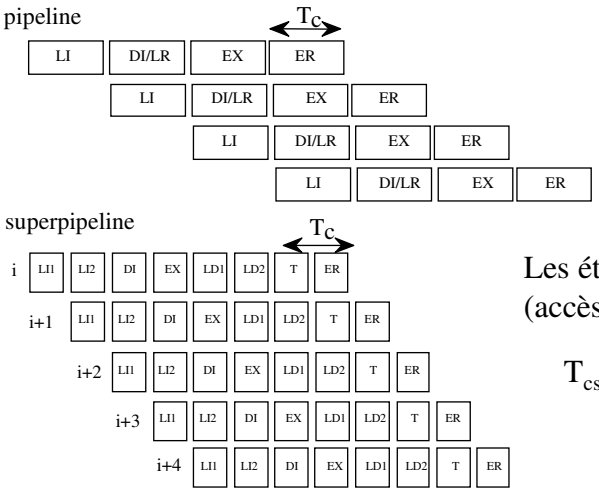
```
LOOP   1   SD F4, 0(R2)       ; range Y(i)
       2   FADD  F4,F2,F3     ; a * X(i+1) + Y(i+1)
       3   FMUL F3, F0, F1    ; a * X(i+2)
       4   LD F1, 24(R1)      ; charge X(i+3)
       5   LD F2, 24(R2)      ; charge Y(i+3)
       6   SUB R6,R7,R1       ; compare i et N-4
       7   ADDI R1,R1,8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

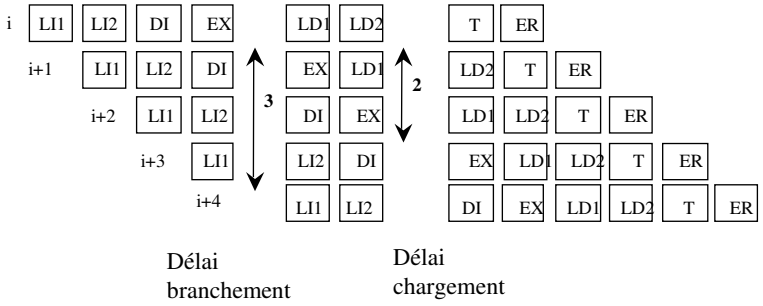


Les étages critiques (accès cache) sont pipelinés

$T_{cs} = T_c/2$

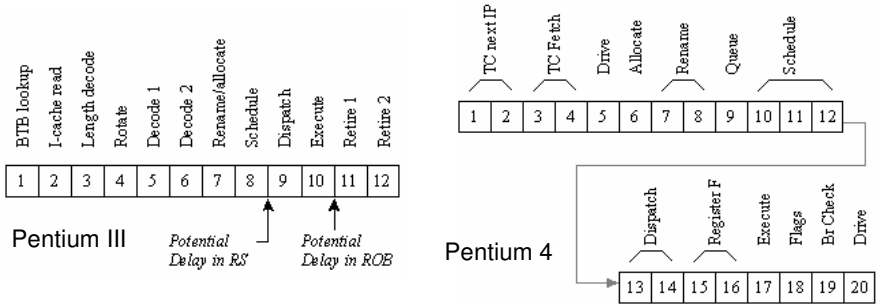
Superpipelines : chargements et branchements

Superpipeline MIPS R4000



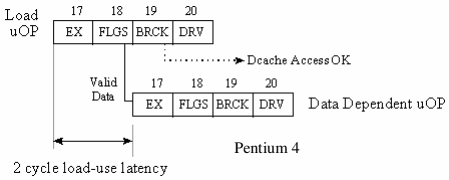
Pipelines Pentium III et Pentium 4

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

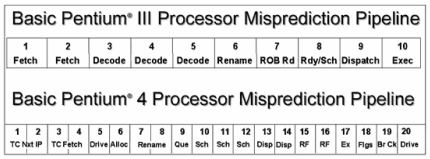


Latences chargement/branchement

Latence de chargement

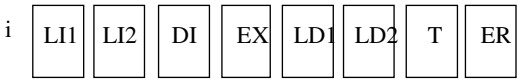


Pénalité de mauvaise prédiction



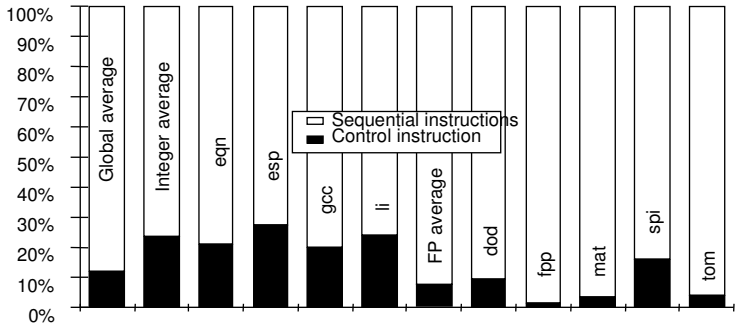
- Optimisation matérielle
  - Prédiction de branchement
- Optimisation programmeur/compilateur
  - Ordonnancement des instructions
  - Conversion SI
    - Utilisation des instructions de transfert conditionnel pour supprimer des branchements conditionnels

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)

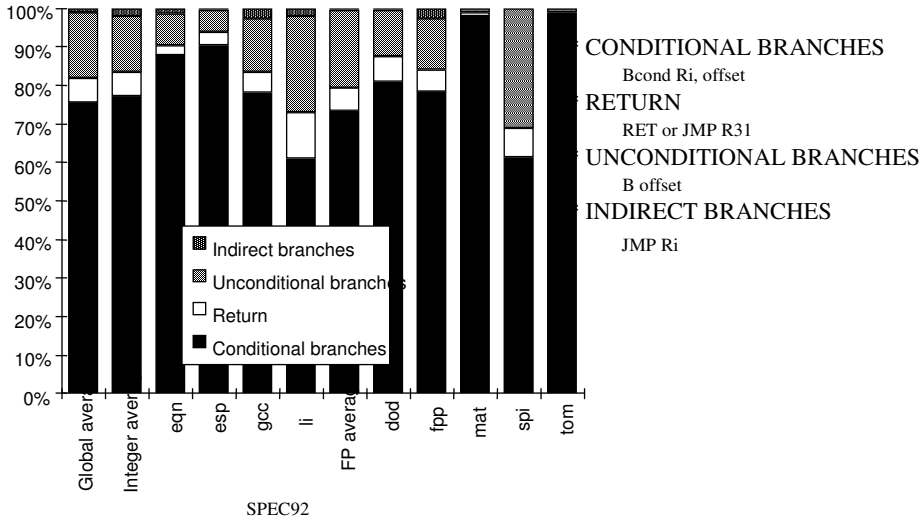
Les instructions de contrôle



SPEC92

Yeh and Patt

Répartition des branchements

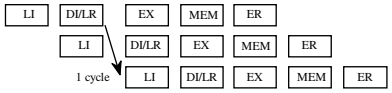


SPEC92

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

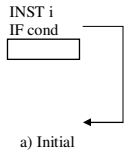
```
LD R1, (R4+8)              LD R1, (R4+8)
LD R2, (R5+8)              LD R2, (R5+8)
ADD R3,R1,R2               ADD R3,R1,R2
ST R3, (R6+8)              BR L1
BR L1                      ST R3, (R6+8)
NOP                        ...
...                        ....
....                       .....
L1 :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



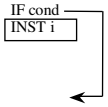
Les branchements conditionnels retardés

Schémas de réordonnancement

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



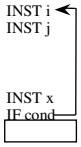
a) Initial



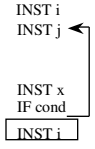
a) Final

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

(Branchement de boucle)

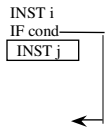


b) Initial

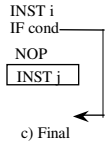


b) Final

c) **Instruction après le branchement**



c) Initial



c) Final

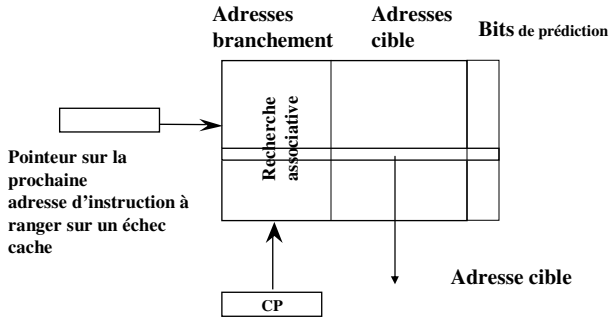
Branchements avec annulation

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)

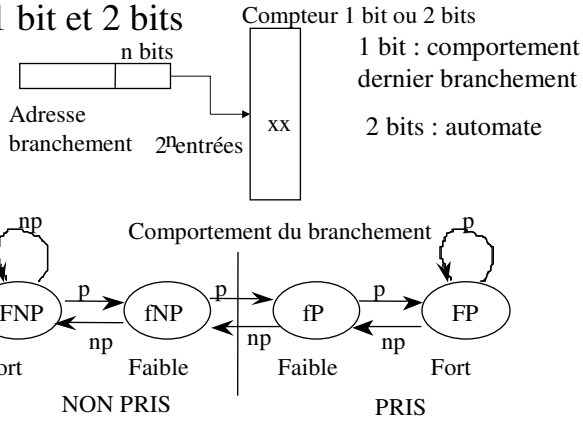


La prédiction de branchement

- 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 = 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 {
```

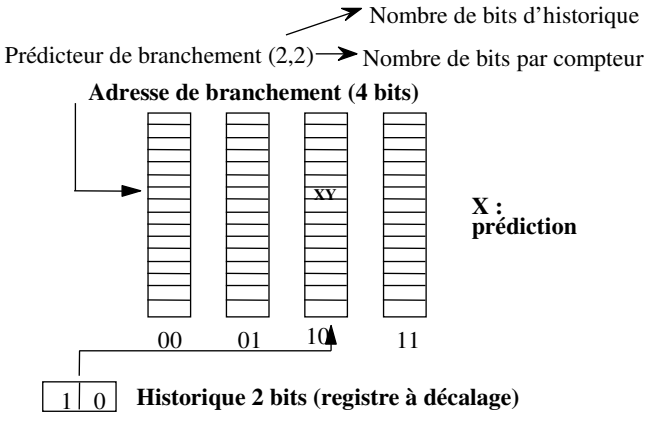
```
        SUBI    R3,R1,#2
b1      BNEZ    R3,L1
        ADD     R1,R0,R0
L1  :   SUBI    R3,R2,#2
b2      BNEZ    R3,L2
        ADD     R2,R0,R0
L2:     SUB     R3,R1,R2
b3      BEQZ R3,L3
```

b1 PRIS si aa ≠ 2
b2 PRIS si bb ≠ 2
b1 et b2 NON PRIS (aa = bb = 0) => b3 PRIS

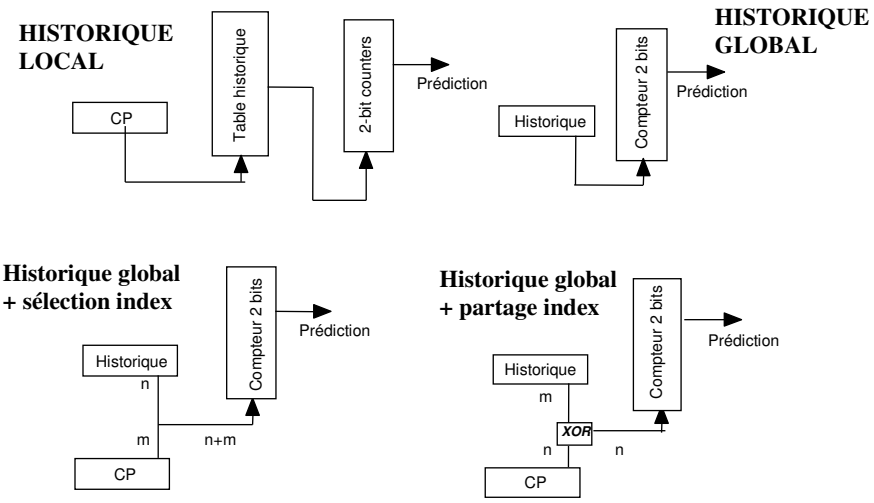
Corrélation entre les comportements de b1, b2 et b3

Les prédicteurs à deux niveaux

PREDICTEUR A 2 NIVEAUX

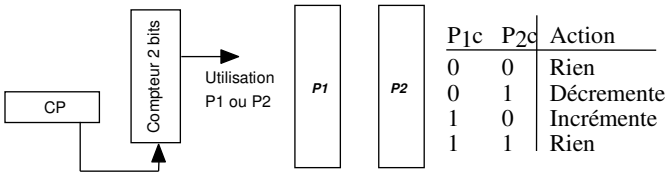


Local + Global



Les méta - prédicteurs

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

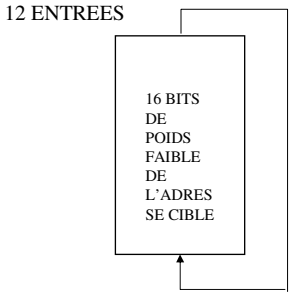


P1c	P2c	Action
0	0	Rien
0	1	Décremente
1	0	Incrémente
1	1	Rien

Prédiction de l'adresse de retour

- Pile d'adresse de retour

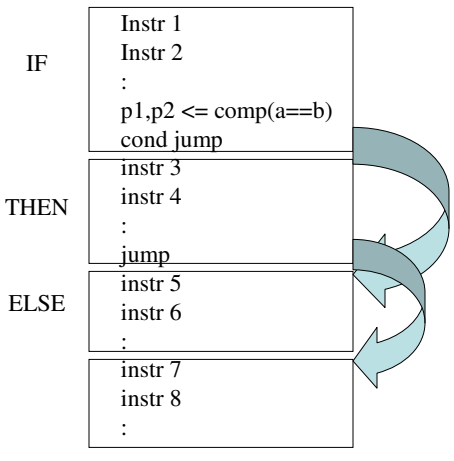
ALPHA



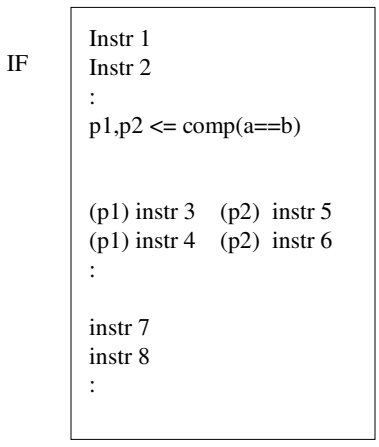
Off.(15:14)	Cible prédite	Opération sur la pile
00 JMP	PC+4xdep(13:0)	
01 JSR	PC+4xdep(13:0)	PUSH PC
10 RET	Prediction stack	POP PC
11 JSR Co-routine	Prediction stack	POP ET PUSH PC

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

L'utilisation des prédicats



4 blocs de base



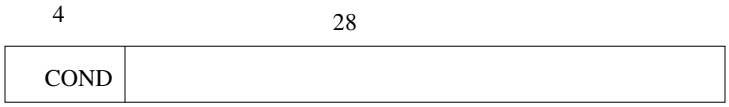
Nouveau bloc de base

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

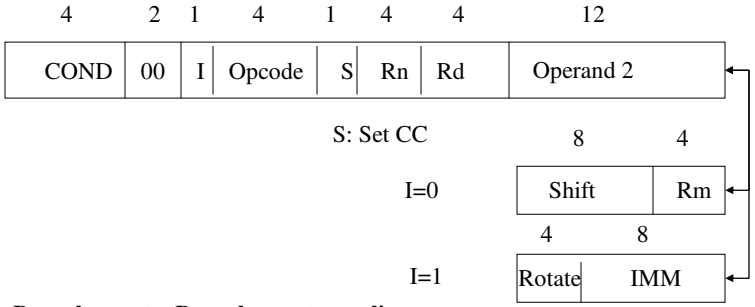
EXECUTION CONDITIONNELLE DE TOUTE INSTRUCTION



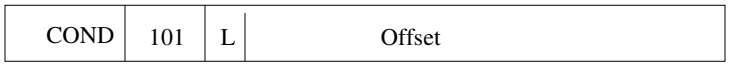
0000	EQ	1000	unsigned >
0001	NE	1001	unsigned <=
0010	CS	1010	≥
0011	CC	1011	<
0100	<0	1100	>
0101	>0 or =0	1101	≤
0110	overflow	1110	ALWAYS
0111	No overf	1111	NEVER

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 letters 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.

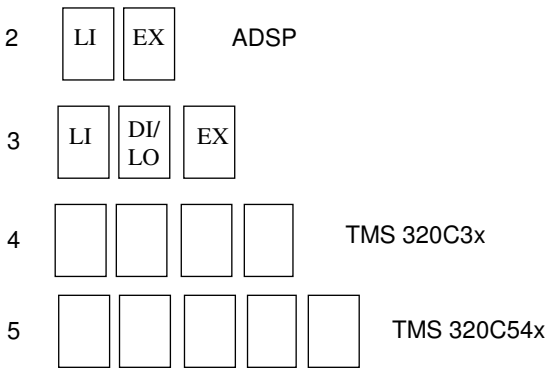
Programme C :

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

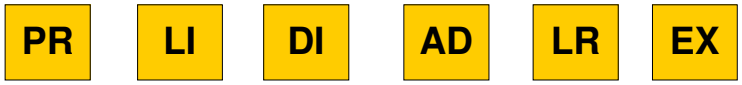
Codage ARM:

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


Les pipelines des DSP

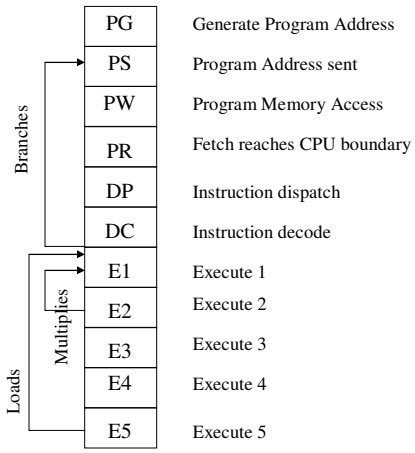


Pipeline du C54x



- 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