Architecture des ordinateurs

Cours (12x2h00) :

- Frédéric Mallet - fmallet@unice.fr

TP (12x2h00 - 2 groupes) :

- Jean-Pierre Lips - Jean-Pierre.LIPS@unice.fr
- Christophe Delage – Christophe.Delage@sophia.inria.fr

http://deptinfo.unice.fr/~fmallet/archi

Structure du cours

- Performances : évolution et comparaison
- Codage de l'information
- Fonctions logiques et éléments mémoires
- Systèmes à microprocesseurs
- La famille Intel - 80x86
- Conception d'architectures numériques – VHDL
- Éléments avancés (parallélisme, mémoire, ...)

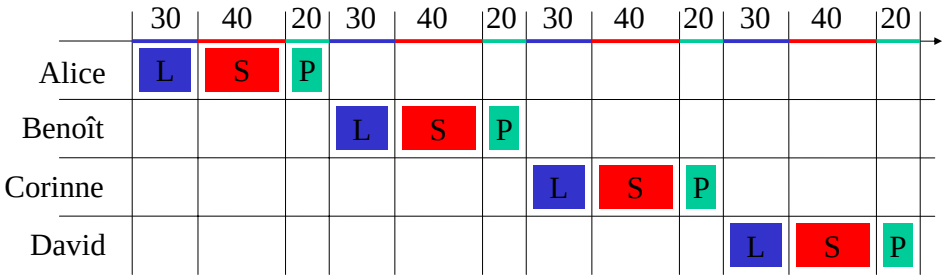
Améliorer les performances

- Solutions **technologiques**
  - Plus de transistors dans moins de surface
  - Des portes plus rapides
  - Des temps d'accès à la mémoire de + en + court
- Solutions **architecturales**
  - Banc de registres au lieu d'un accumulateur
    - Réduit le nombre d'accès à la mémoire
  - 2 bus de données au lieu d'un
    - Permet de lire **2** opérandes par cycle au lieu d'**1**
  - Quoi d'autre ?
    - **Parallélisme d'instructions**, prédiction de branchement, prédication, cache d'instructions ou de données, instructions délivrées dans le désordre, DMA, co-processeurs

Exemple de la laverie automatique

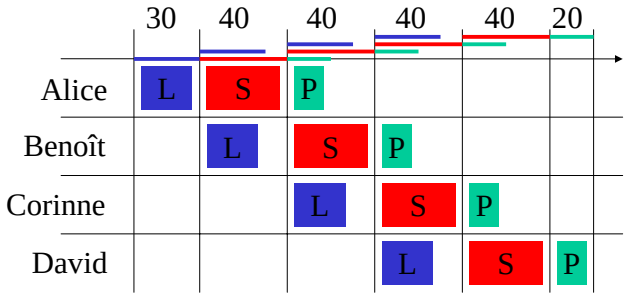
- **A**lice, **B**enoît, **C**orinne et **D**avid ont du linge à laver
  - Le lavage prend 30 minutes
  - Le séchage prend 40 minutes
  - Le pliage prend 20 minutes
- On dispose d'une machine à laver, d'un sèche-linge et d'une table à plier.

Le lavage séquentiel



- Durée totale : (30+40+20)*4 = 6h
- **Latence :** 1h30
  - Temps pour exécuter une tâche complète
- **Débit :** 6h / 4 = 1h30
  - Temps pour exécuter N tâches / nombre de tâches

Le lavage en *pipeline*



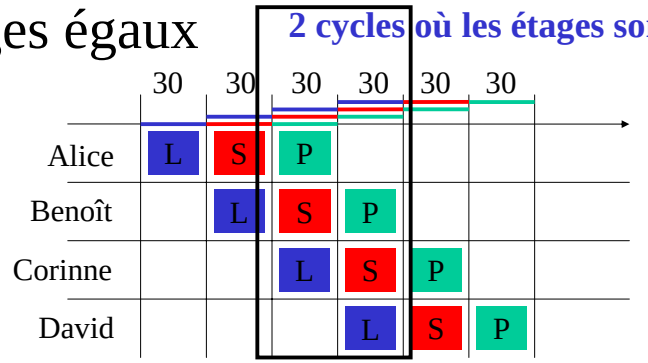
- Durée totale : 30+40*4+20 = 3h30
- **Amélioration :** 6h00/3h30 = 1.74
- **Latence :** 1h30
  - Temps pour exécuter une tâche complète
- **Débit :** 3h30 / 4 = 52.5 mn (tend vers 40 mn)
  - Temps pour exécuter N tâches / nombre de tâches

Les leçons à tirer

- Le *pipeline* améliore **seulement le débit** pas la latence
- Décomposition en **étages** exécutés en parallèle
- Le *taux* de pipeline est **limité par l'étage le plus lent**
- Amélioration : $T_{séquentiel}$ / $T_{pipeline}$
  - Dépend du nombre d'étages
  - Réduite si les étages ont des durées inégales
  - Réduite par le temps de **remplissage** et de vidange du **pipeline**

3 étages égaux

2 cycles où les étages sont pleins



- Durée totale : 30+30*4+30 = 3h00
- **Amélioration :** 6h00/3h00 = 2
- **Latence :** 1h30
  - Temps pour exécuter une tâche complète
- **Débit :** 3h00 / 4 = 45 mn (tend vers 30 mn)
  - Temps pour exécuter N tâches / nombre de tâches

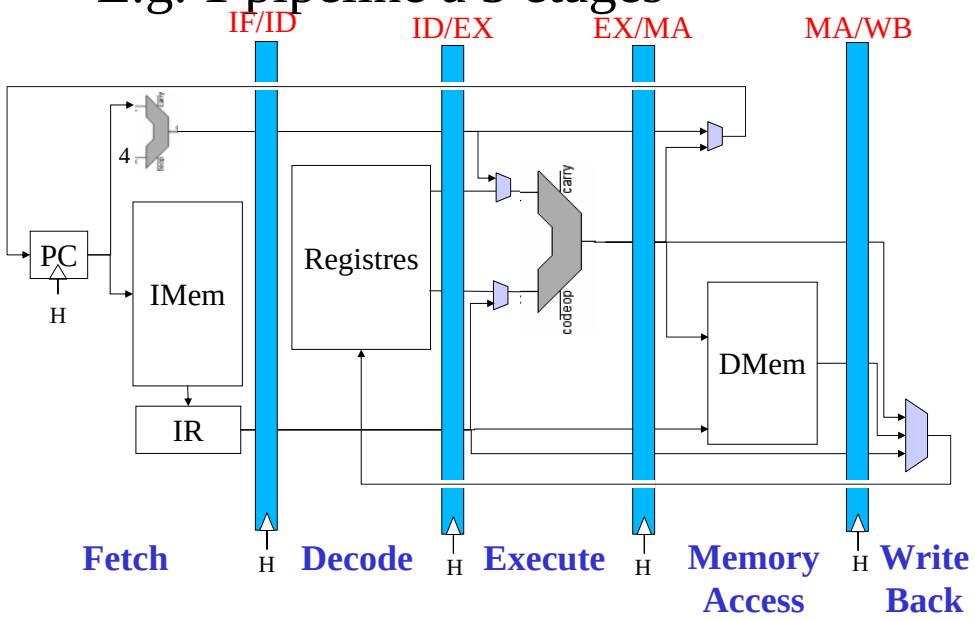
Parallélisme d'instructions : **pipeline**

- Similaire au travail à la chaîne
  - L'exécution d'une instruction se décompose en étapes « indépendantes » (**étages**) :
    - Fetch, Decode, Execute, Write Back
  - On crée des ateliers spécialisés pour chaque étape/étage
    - Lorsque l'atelier 1 a fini avec l'instruction **`i`**, il l'envoie à l'atelier 2 et commence le traitement sur l'instruction **`i+1`** …
- Difficultés
  - Les étages doivent avoir des durées à peu près égales
    - Tout le monde marche au rythme du plus lent
  - Certaines instructions brûlent certains étages (**`jmp`**, …)
  - On ne peut pas toujours commencer l'instruction 2 avant d'avoir terminé complètement l'instruction précédente (**`jmp`**)

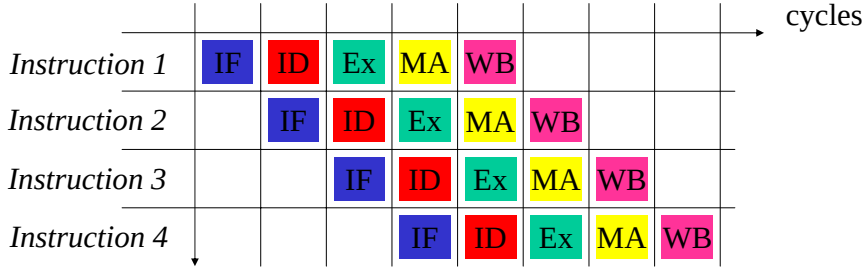
Jeu d'instruction et *pipeline*

- Comment doivent être les instructions pour que le pipeline soit possible et efficace ?
  - Instructions de longueur constante (*Fetch* de durée constante)
  - Accès réduits à la mémoire (LOAD/STORE)
  - Nombre de modes d'adressage réduit
    - Simplifie le décodage
    - Opérations sur les registres
    - Toutes les instructions ont des phases d'exécution similaires
- Si les étages sont courts, on peut augmenter la fréquence d'horloge :
  - **près de N fois pour que ce soit efficace !**
  - Où N est le nombre d'étages du pipeline
- ⇒ **RISC**

E.g. 1 pipeline à 5 étages

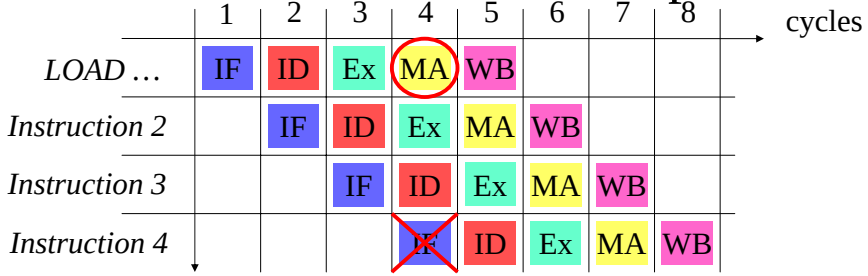


Visualisation du *pipeline*



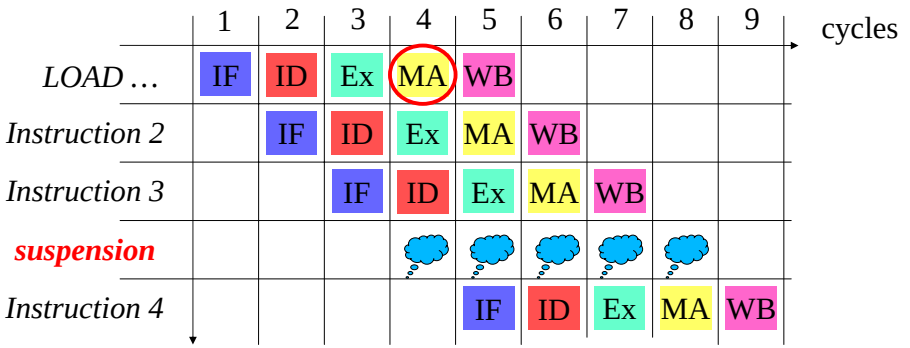
- Ce n'est malheureusement pas aussi simple
  - **Aléas structurels :** Le matériel ne permet pas certaines combinaisons (une machine à laver/sèche linge)
  - **Aléas de données :** Une instruction dépend d'une autre instruction en cours
  - **Aléas de contrôle :** Lors d'un saut, on ne sait pas quelle instruction il faut lire avant que le saut soit terminé

Aléa structurel / mémoire 1-port



- Si mémoire Donnée/instruction avec 1 seul port
  - L'accès mémoire (MA) d'une instruction LOAD empêche un *Fetch* (IF) simultané 3 instructions plus tard
  - Détection par le matériel ⇒ **insérer une bulle** pour retarder le IF de l'instruction 4.
- Si l'ALU est utilisée dans le *Fetch* : PC=PC+4

Aléa structurel / insertion de bulles

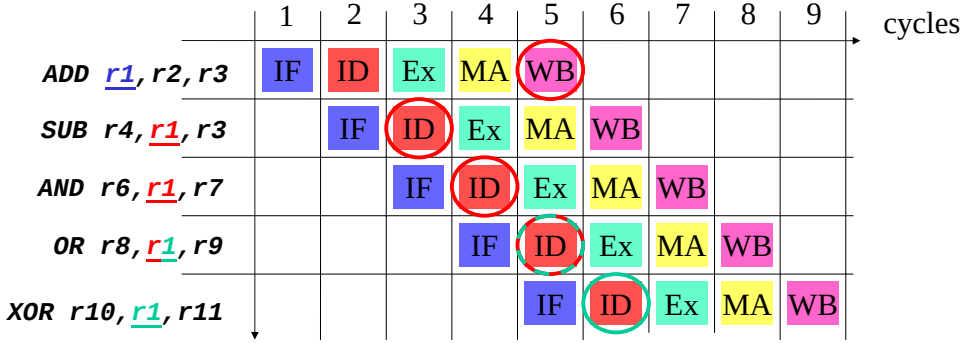


- Sans *pipeline* : 1 CPI H=100 MHz ⇒ 100 MIPS

Avec 1 *pipeline* 5 étages, on devrait pouvoir multiplier H par 5:

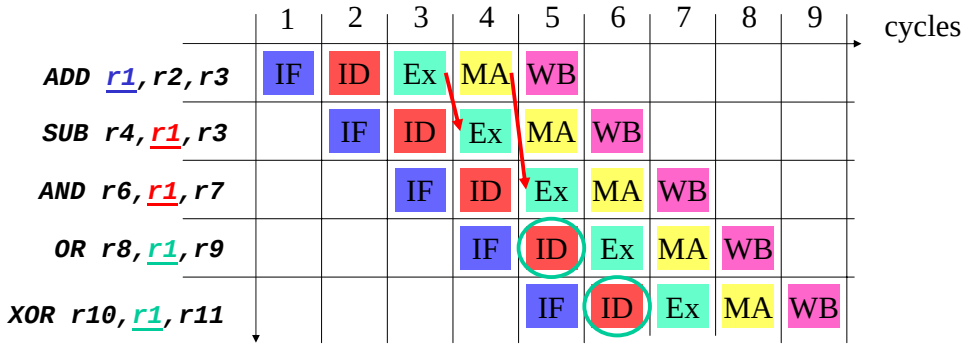
- *Pipeline* plein : 8/4=2 CPI H=450 MHz ⇒ 225 MIPS
- 1 bulle : 9/4 = 2.25 CPI H=450 MHz ⇒ 200 MIPS
- *Pipeline* idéal : 1 CPI H=500 MHz ⇒ 500 MIPS

Aléa de données / RAW



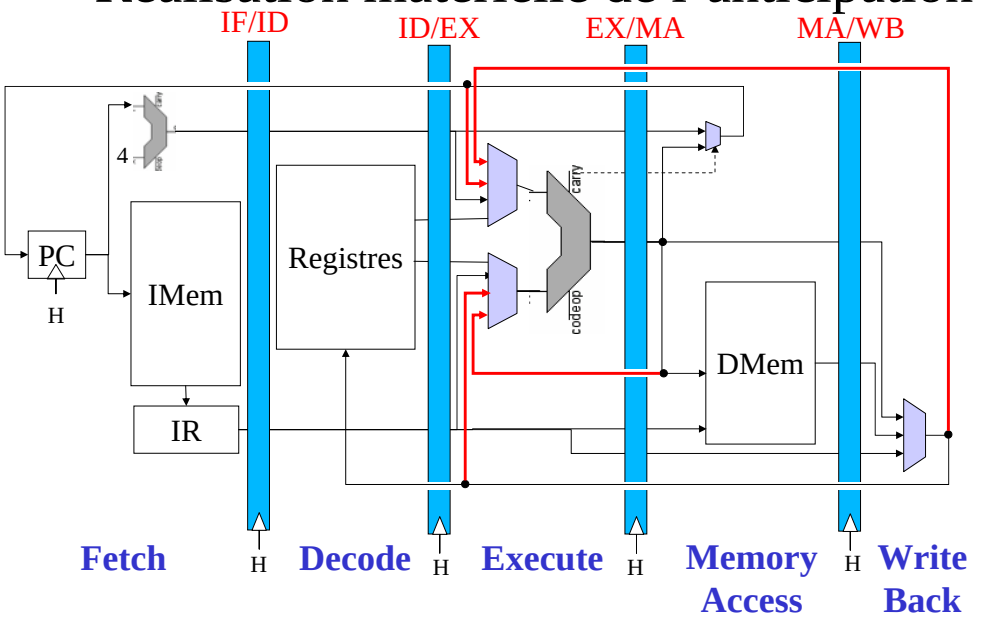
- 3 types d'aléas de données
  - **RAW : Read After Write** (**dépendance**)
  - **WAR : Write After Read** (ssi les étages sont variables)
  - **WAW : Write After Write** (ssi exécution dans le désordre)

Aléa de données / anticipation

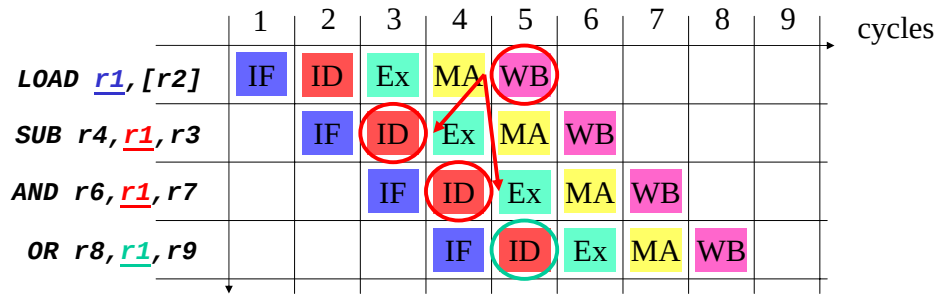


- Pour éviter les aléas RAW
  - On propage les résultats dans l'architecture au plus tôt
  - Il faut faire des **modifications matérielles**

Réalisation matérielle de l'anticipation

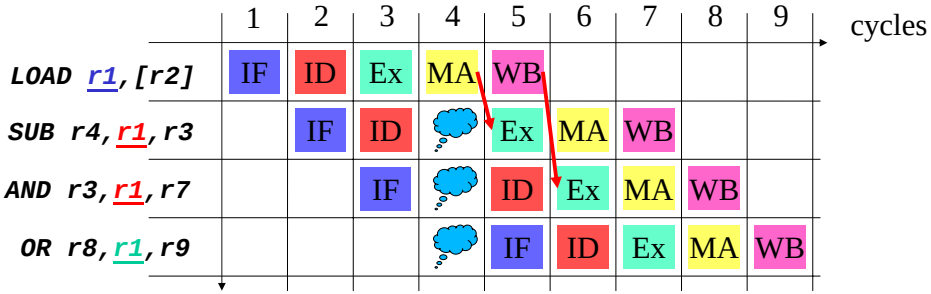


RAW malgré l'anticipation



- RAW sur R1
  - Produit par le LOAD au cycle 5
  - Utilisé par le SUB au cycle 3 (**anticipation impossible!**)
    - **Il faut insérer une bulle !**
  - Utilisé par le AND au cycle 4 (**anticipation possible!**)
    - Nouveau chemin dans l'architecture

RAW malgré l'anticipation



- RAW sur R1
  - On insère une bulle **dans les étages qui précèdent MA**
  - L'**anticipation est toujours nécessaire**
    - De MA vers EX
    - De WB vers EX
  - Sinon il faudrait insérer 2 bulles

limiter RAW : Astuce de compilation

- On veut produire du code pour réaliser
  - a = b + c et d = e – f a,c,b,d,e,f résident en mémoire

- Version 1

```
LOAD Rb,[b]
LOAD Rc,[c]
ADD Ra,Rb,Rc
STORE [a],Ra
LOAD Re,[e]
LOAD Rf,[f]
SUB Rd,Re,Rf
STORE [d],Rd
```

PAUSE

PAUSE

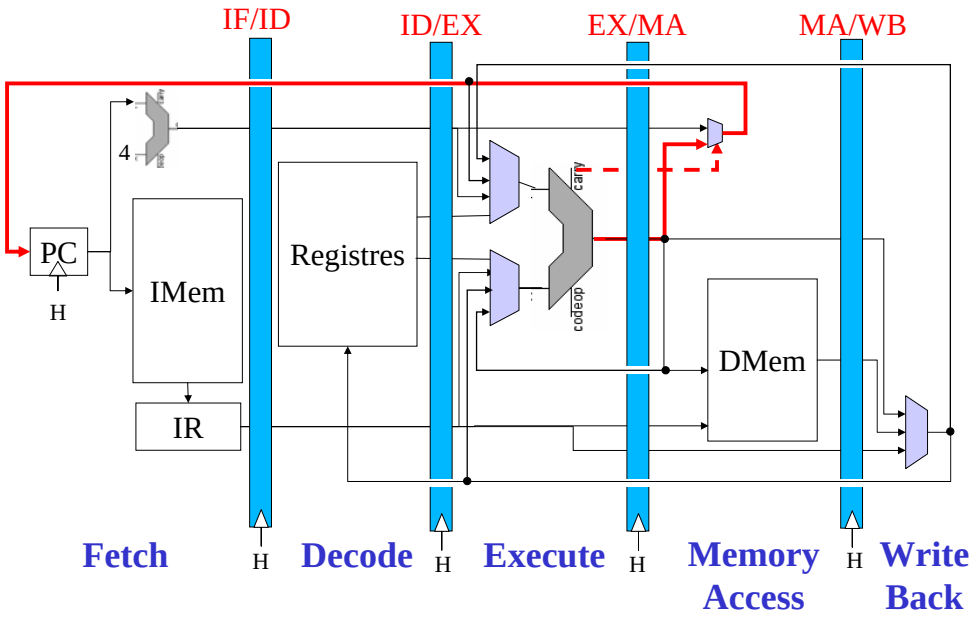
- Version 2

```
LOAD Rb,[b]
LOAD Rc,[c]
LOAD Re,[e]
ADD Ra,Rb,Rc
LOAD Rf,[f]
STORE [a],Ra
SUB Rd,Re,Rf
STORE [d],Rd
```

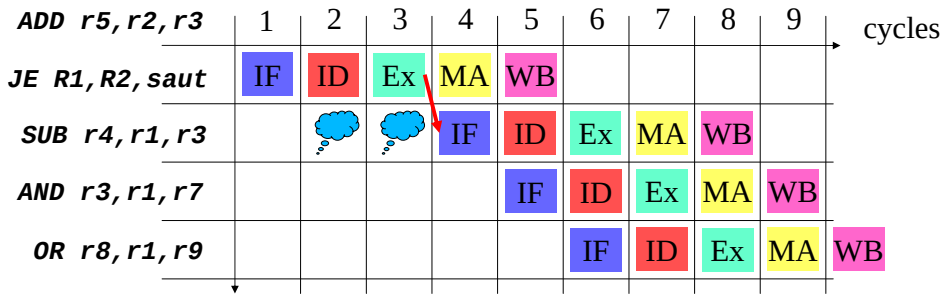
ANTICIPE

2 cycles perdus !

Aléa de contrôle : branchement



Aléa de contrôle : branchement



- Aléas de contrôle causé par le JE
  - Exécuter le SUB que si le saut n'est pas effectué
    - Il faut *a priori* insérer 4 bulles (vidanger le pipeline)
- On saura si le saut doit être pris **seulement après EX**
  - On peut utiliser l'anticipation
  - Il faut alors insérer 2 bulles, malgrè tout

Aléa de contrôle : Une solution partielle

- On peut utiliser à la place deux instructions
  - CMP R1,R2
  - JE saut
  - Reste une bulle ⇒ même coût qu'avant

- Déplacer le test de zéro? à **l'étage de décode**
- Ajouter un additionneur à **l'étage de décode** pour calculer le nouveau PC si c'est un branchement relatif
- On gagne ainsi une BULLE

Aléa de branchement : solutions ?

- **1.** Insérer suffisamment de bulles
- **2. Supposer que le branchement n'est pas pris**
  - Exécute les instructions suivantes en séquence
  - Annule les instructions si le branchement est pris
    - Possible ici car le pipeline, met à jour très tard (WB, MA)
    - Il faut également faire attention au registre d'état FLAGS
  - Coûte comme des bulles en cas d'annulation
  - On peut faire des statistiques (SPEC) pour savoir combien de branchements sont pris en moyenne
- **3. Supposer que le branchement est pris**
  - Nécessite une bulle pour calculer l'adresse de saut

4. délai de branchement

- On peut sinon décider que le branchement n'est effectif qu'après N cycles
  - Il y a alors N **délais de branchement**
  - Il suffit de le savoir dans les compilateurs

- E.g. N = 1

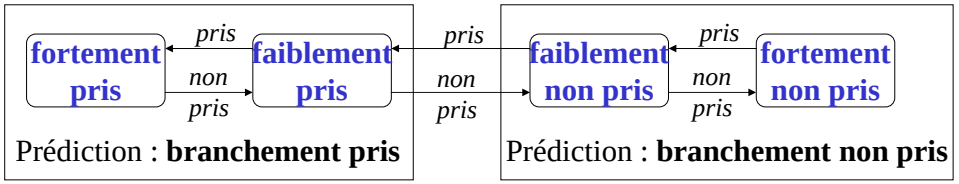
```
ADD R5,R2,R3
JE R1,R2,saut
NOP  ; délai de branchement
SUB R4,R1,R3
AND R3,R1,R7
```

- Ou mieux

```
JE R1,R2,saut
ADD R5,R2,R3 ; délai
SUB R4,R1,R3
AND R3,R1,R7
```

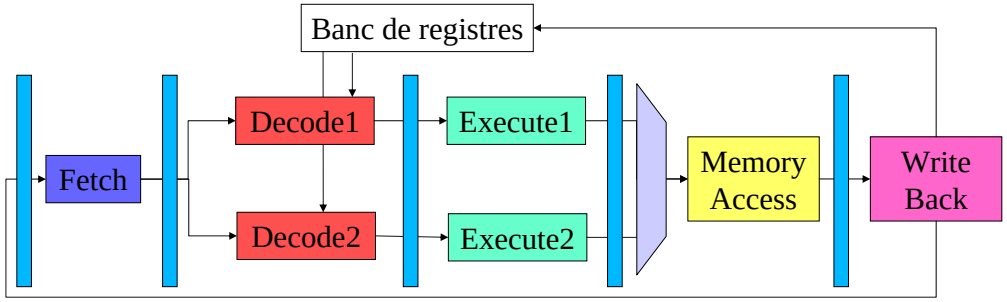
Mais, il faut avoir des instructions à insérer dans les délais !
Sinon, on peut toujours insérer des NOP

5. Prédiction de branchement



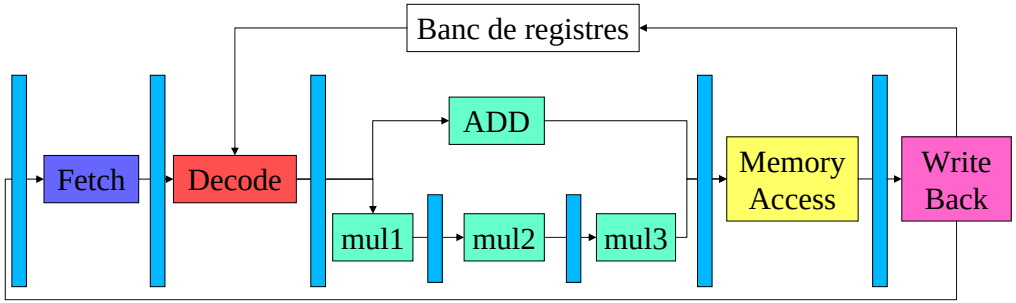
- E.g. prédiction sur 2 bits : 4 états
  - On observe les branchement pris ou non pris
  - On choisit **dynamiquement** en fonction de l'histoire
  - Les bons *prédicteurs* de branchement ont des résultats de 95 à 99% de réussite

6. Exécution par prédicats



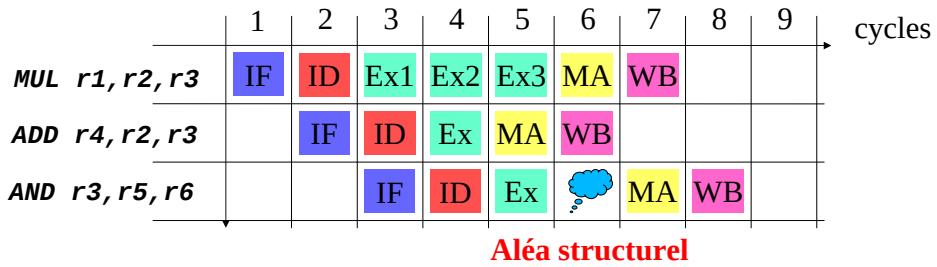
- Remplace les branchements par des instructions conditionnelles
  - Il suffit de confirmer ou infirmer l'exécution au MA/WB
  - Cela coûte du matériel en plus, mais pas de pénalité
  - E.g. `ADD r2,r1,r3  if (cc)`

Cas de plusieurs pipelines



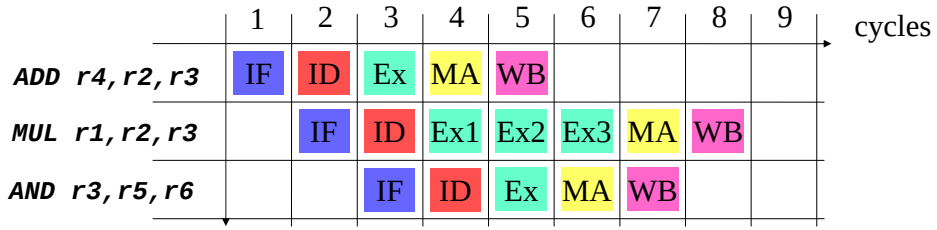
- Dans le cas de processeurs superscalaires
  - Les unités de calculs n'ont pas nécessairement le même temps d'exécution
  - On construit des pipelines séparés pour ne pas être pénalisé par la plus lente des exécutions

Processeurs superscalaires

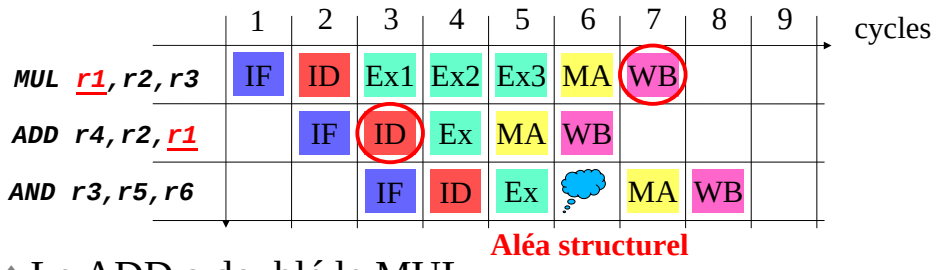


Aléa structurel

- Le ADD a doublé le MUL
  - Cela peut provoquer d'autres formes d'aléas de données

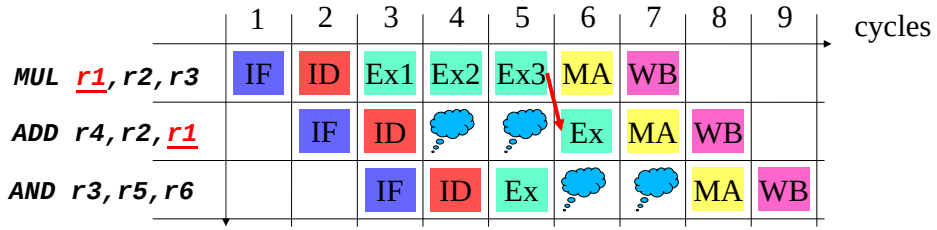


Aléa de donnée : Write After Read

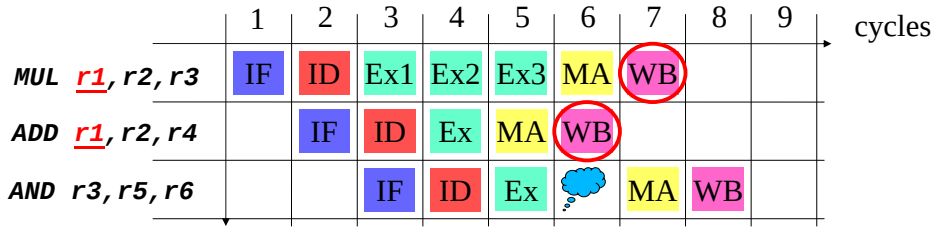


Aléa structurel

- Le ADD a doublé le MUL
  - Au lieu d'avoir r4=r2+r2*r3 on obtient r4=r2+$r1_{old}$

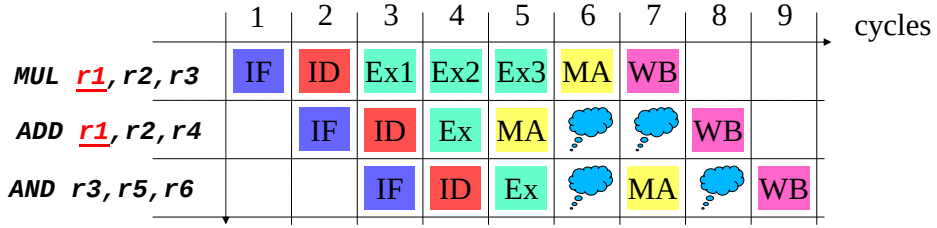


Aléa de donnée : Write After Write



Aléa structurel

- Le ADD a doublé le MUL
  - Au lieu d'avoir r1=r2+r4 on obtient r1=r2*r3



Conclusions et compléments

- L'utilisation d'un pipeline permet d'**augmenter le débit**
- Mais de nombreux aléas sont possibles
  - **Aléas de contrôle**
    - Délai de branchement, Prédiction de branchement, bulles, prédication
  - **Aléas structurels**
    - Un travail sur le pipeline **statique** peut empêcher la situation d'arriver
  - **Aléas de données**
    - RAW : anticipation, travail sur la compilation (vraie dépendances)
    - WAR : bulles, renommage de registres : Tomasulo (anti-dépendances)
    - WAW : renommage de registres (dépendances de sortie)
- Les **exécutions dans le désordre**
  - Évitent de bloquer le pipeline, mais augmente les WAR et WAW