Conception avancée de processeurs: pipelines

IFT1225

Dép. d'IRO, Université de Montréal
(professeur E. Cerny)

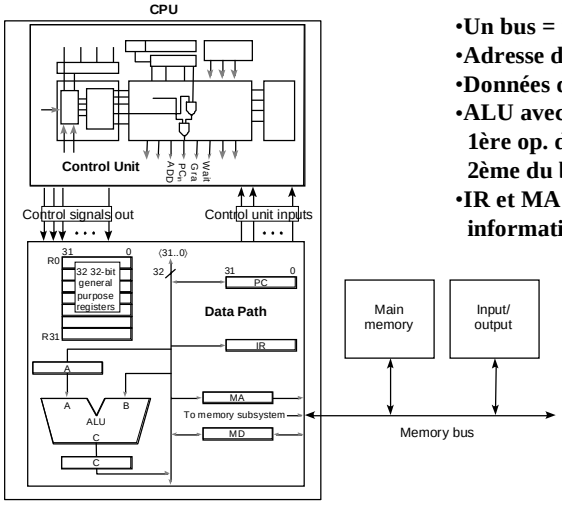
Résumé

- Organisation de processeurs sans pipeline (revision)
  - chemin de données à 1, 2 et 3 bus
- Conception à pipeline
  - étages d'un pipeline d'instructions
  - dépendances, aléas (*hazards*)
  - solutions:
    - calage
    - sauts avec retard (*branch delay*)
    - transit des données (*data forwarding*)
    - chargement avec retard (*load delay)*

Organisation de processeurs

- Description RT abstraite - l'effet des instructions sur les registres et mémoire visibles au programmeur
- Description RT concrète - organisation du chemin de données et les transferts exactes nécessaires pour exécuter des instructions sur cette organisation particulière
- Plusieurs organisation réelles différentes peuvent exécuter le même jeux d'instructions

SRC à bus unique (sans pipeline d'instructions)



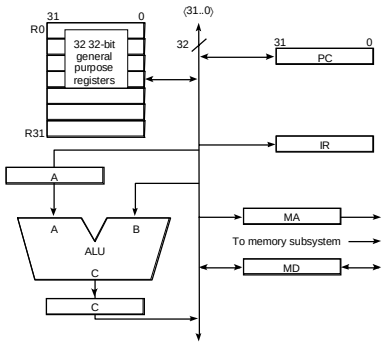
- **Un bus = un transfert à la fois**
- **Adresse de mémoire via MA**
- **Données de/à mémoire via MD**
- **ALU avec 2 tampons:**
  1ère op. dans A,
  2ème du bus, résultat dans C
- **IR et MA seulement reçoivent information du bus**

Description RT abstraite et concrète d'*add*

RT abstraite: **(IR ← M[PC]: PC ← PC + 4; instruction_execution);**
instruction_execution := (• • •
add (:= op= 12) → R[ra] ← R[rb] + R[rc]:

RT concrète

Step	**RTN**
T0	**MA ← PC: C ← PC + 4;**
T1	**MD ← M[MA]: PC ← C;**
T2	**IR ← MD;**
T3	**A ← R[rb];**
T4	**C ← A + R[rc];**
T5	**R[ra] ← C;**

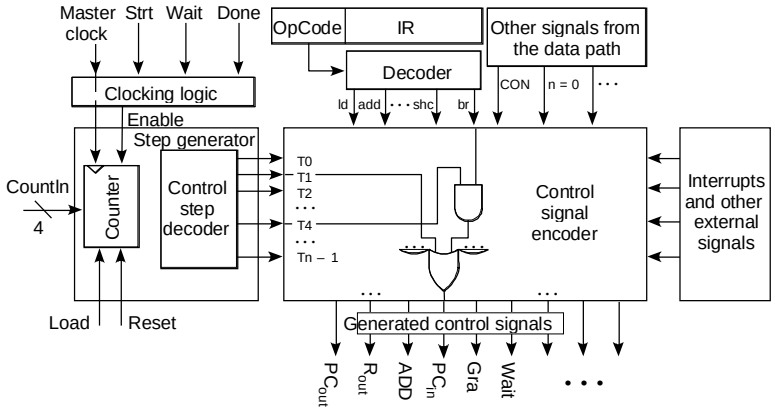


- Portions de 2 RTs (IR ← M[PC]: PC ← PC + 4;) effectuées dans T0
- l'adition demande 3 transferts (T3, T4, T5)

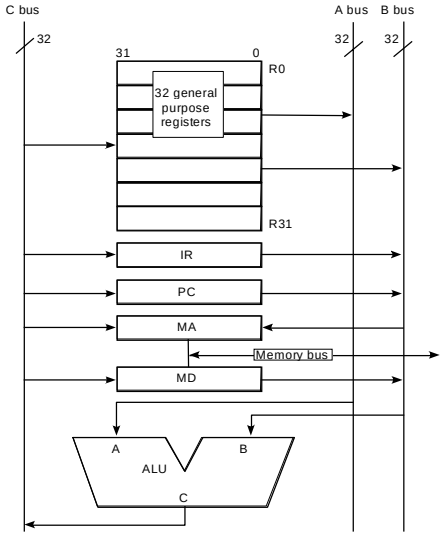
L'unité de contrôle

- Génère des signaux de contrôle dans la bonne séquence
- L'unité de contrôle dépend de
  - Le pas (ou cycle mineur) de contrôle, Ti
  - Le code d'opération de l'instruction (pour autres que T0, T2, T2)
  - Signaux de statut de l'UAL (CON, n = 0, etc.)
  - Signaux externes: reset, interrupt, etc.
- Les composants de l'unité de contrôle:
  - Générateur des pas d'instructions (T0, T1, T2, …)
  - Décodeur d'instructions
  - Logique combinatoire pour produire les signaux de contrôle

L'unité de contrôle (suite)



Organisation à 3 bus



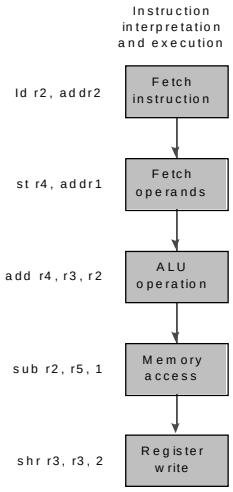
Exemple: addition

Pas	**RT concrète**
T0	**MA ← PC: MD ← M[MA];**
	PC ← PC + 4:
T1	**IR ← MD;**
T2	**R[ra] ← R[rb] + R[rc];**

Performance:
T d'horloge 1.1x plus longue
mais 3 cycles seulements
pour toute instruction

accélération?

Exécution en pipeline



Programme:

```
...
shr r3, r3, #2
sub r2, r5, #1
add r4, r3, r2
st  r4, addr1
ld  r2, addr2
...
```

Un top d'horloge par instruction

Une instruction se termine à chaque top d'horloge

Latence vs. débit

Dépendances entre instructions

- Résultat d'une instruction est l'opérande de l'instruction suivante
- Condition de branchement n'est connue qu'après la recherche d'instruction suivante
- Chargement de registre (ld) de mémoire arrive trop tard

Solutions:

Une triviale, mais lente...

- Bloquer l'exécution (calage, *stall*) en attendant le résultat - défaut!

Et pour accélérer…:

- Avancer le résultat sans passer par le registre (transit, *forward*)
- Exécuter toujours l'instruction qui suit un branchement (*delay slot*)
- Prédiction de branchement
- Reorganiser le programme pour éliminer la dépendance (*load delay*)

Positions de retard (*delay slots*)

De branchement

```
brz r2, r3
add r6, r7, r8
st  r6, addr1
```

Cette instruction est toujours exécutée

Que si r2 ≠ 0

De chargement

```
ld  r2, addr
add r5, r1, r2
shr r1,r1,#4
sub r6, r8, r2
```

Cette instr. Obtient l'ancienne valeur de r2

Elle obtient la valeur de r2 chargée d'*addr*

- L'opération de chaque instruction individuelle n'est pas affectée, mais l'interaction entre les instructions peut changer! La sémantique du programme n'est pas maintenu.

Caractéristiques des processeurs à pipeline

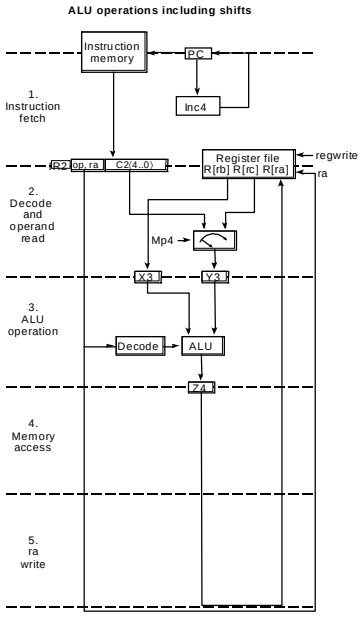
- Mémoire doit répondre en un cycle d'horloge (temps d'accès)
  - Par une technologie très chère et rapide
  - Habituellement plutôt avec un cache (voir Chap. 7)
- Les mémoires d'instructions et de données doivent apparaître séparées
  - Architecture "Harvard" plutôt que "von Neumann"
  - Obtenue avec des caches séparés (et mémoire commune)
- Peu de bus utilisés
  - La plupart de connexions sont point-à-point (moins de délai)
  - Quelques multiplexeurs utilisés
- Données et partie d'instructions mémorisées à l'entrée de chaque étage de pipeline (registres tampons) - "*pipeline registers*"
- Les opérations de l'UAL seulement un cycle (v. flottante traitée à part)

Structure de pipeline de SRC

- Toute instruction passe par tout étage
- Pipeline à 5 étages (assez commun)
  - F: Recherche d'instruction (*Fetch*)
  - D: Décodage et accès aux opérandes (*Decode*)
  - E: Opération de l'UAL (*Execute*)
  - M: Accès à la mémoire de données (*Memory*)
  - W: Écriture de registre (*Write-back*)
- Chaque instruction prend 5 tops d'horloge
- Mais, en absence d'aléa (calage), une instruction complète à chaque top d'horloge

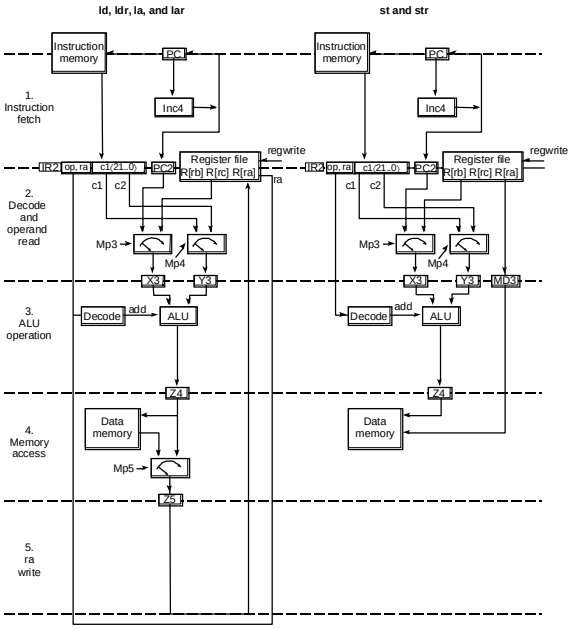
Instructions arithmétiques et logiques

- **2ème opérande de l'UAL vient d'un registre ou du champ c2 d'IR**
- **Le code d'opération demandé dans l'étage 3**
- **Registre de résultat ra est écrit dans l'étage 5**
- **Pas d'accès à la mémoire**
- **Faut-il mémoriser le code d'opération et le résultat dans les étages suivants??**



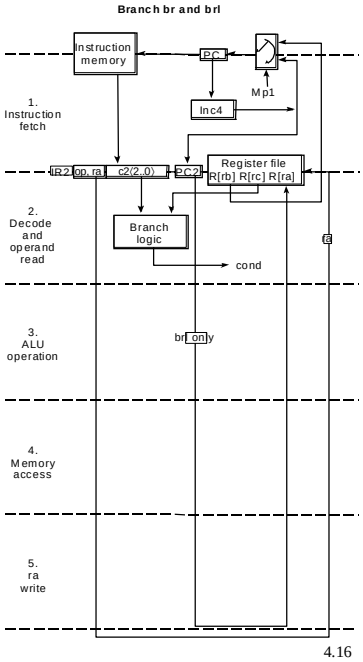
Chargement et Rangement ld, la, ldr, lar, st, str

- L'UAL calcule l'adresse effective
- Étage 4 effectue la lecture ou écriture de la mémoire de données
- Registre - le résultat n'est écrit qu'avec des instructions de chargement (ld, la, ldr, lar)



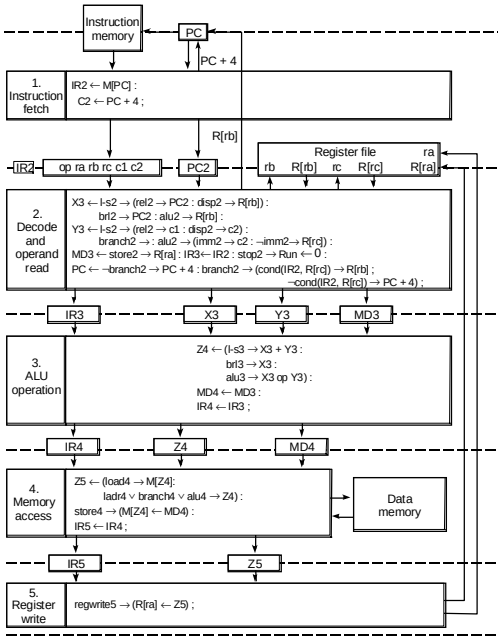
Branchements

- La nouvelle valeur du PC n'est connue que dans l'étage 2, mais pas dans 1!
- Seulement "*branch and link*" écrit un registre dans l'étage 5
- Il n'y a pas d'opération UAL ou d'accès à la mémoire



Intégration avec les registres tampons de pipeline

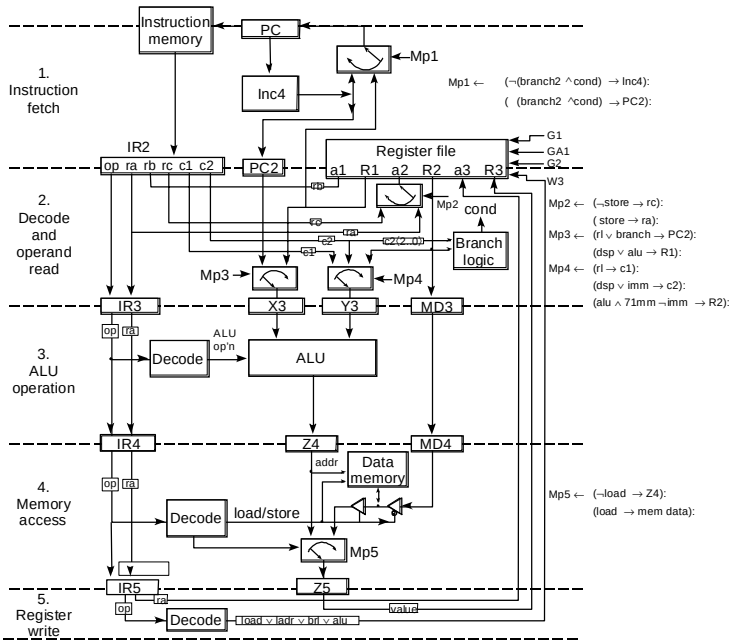
- Registres de pipeline isolent les étages
- RT spécifie les valeurs de registres en sortie de l'étage en terme des opérations sur les registres en entrée de cet étage
- PC accédé dans étages 1 et 2
- Registres lus dans étage 2 et écrits dans 5
- Mémoire d'instructions dans 1, de données dans 4



Accès à l'état du processeur

- L'état global de la machine: PC, les registres généraux, la mémoire d'instructions, et la mémoire de données
- Restrictions de l'accès à l'état global:
  - Instruction accédée dans étage 1 et données dans étage 4 en même temps - demande les mémoires de données et d'instructions séparées pour éviter de conflit d'accès (calage)
  - Deux opérandes lus dans étage 2 et le résultat écrit dans étage 5 - demande une banque de registres qui peut effectuer ces 3 opérations au même cycle d'horloge:
    il y a 2 ports de lecture et un port d'écriture
  - La valeur incrémentée du PC (dans étage 1) peut être modifiée par la suite par l'adresse de branchement dans étage 2 (si le branchement est pris)

Chemin de données à pipeline



- **Majorité de signaux de contrôle est montrée**
- **Contrôle de MUXs indiqué**

Fonctions des registres du pipeline SRC

- Registres entre étages 1 et 2:
  - I2 retient l'instruction au complet
  - PC2 retient le PC incrémenté de 4
- Registres entre étages 2 et 3:
  - I3 retient le code d'opération et *ra* (pour étage 5)
  - X3 retient le PC ou la valeur d'un registre (pour *link* ou 1er opérande de l'UAL)
  - Y3 retient *c1* ou *c2* ou la valeur d'un registre (pour 2ème opérande de l'UAL)
  - MD3 est utilisé pour la valeur d'un registre à être rangée en mémoire de données

Fonctions des registres du pipeline SRC (suite)

- Registres entre étages 3 et 4:
  - I4 retient le code d'opération et le numéro du registre (*ra*)
  - Z4 a l'adresse de mémoire ou la valeur du résultat à être rangé dans le registre indiqué par *ra*
  - MD4 retient la valeur à être rangée en mémoire
- Registres entre étages 4 et 5:
  - I5 a le code d'opération et le numéro du registre de destination *ra*
  - Z5 a la valeur à être rangée dans le registre destination (indiqué par *ra*): le résultat de l'UAL ou le PC de *link* ou les données en provenance de la mémoire

Fonctions des étages du pipeline SRC

- Étage 1: recherche d'instruction
  - PC est incrémenté ou remplacé par l'adresse de branchement de l'étage 2
- Étage 2: décode l'instruction et recherche les opérandes
  - *Load* ou *Store* obtient les opérandes pour le calcul d'adresse effective
  - *Store* obtient la valeur d'un registre à être rangée, comme 3e opérande
  - l'UAL obtient 2 registres ou un registre et une constante
- Étage 3: performe l'opération de l'UAL
  - Calcule l'adresse effective ou arithmétique / logique
  - Peut laisser passer le PC de *link* ou la valeur à être rangée en mémoire

Fonctions des étages du pipeline SRC (suite)

- Étage 4: accès à la mémoire de données
  - Passe Z4 à Z5 inchangé pour les instructions sans accès de mémoire
  - *Load* charge Z5 par la donnée en provenance de la mémoire
  - *Store* utilise l'adresse de Z4 et la donnée de MD4 (ne sera plus utile)
- Étage 5: écrit le registre de résultat *ra*
  - Z5 contient la valeur à être écrite qui peut être le résultat de l'UAL, l'adresse effective, la valeur de PC *link*, ou les données de la mémoire
  - le champ *ra* toujours spécifie le registre de résultat dans le SRC

Exemple de propagation d'instructions dans le pipeline

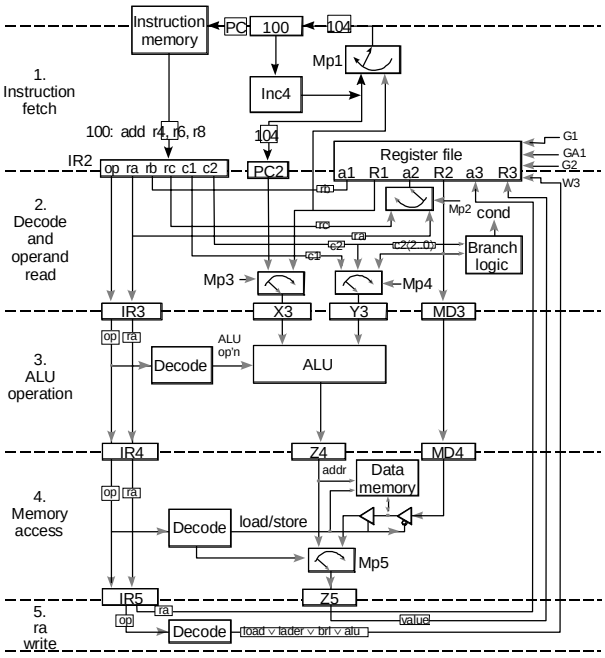
```
100: add  r4, r6, r8;    R[4] ← R[6] + R[8]
104: ld   r7, 128(r5);   R[7] ← M[R[5]+128]
108: brl  r9, r11, 001;  PC ← R[11]: R[9] ← PC
112: str  r12, 32;       M[PC+32] ← R[12]
     . . . . . .
512: sub  ...            next instr. ...
```

- On suppose que R[11] contient 512 quand *brl* est exécuté
- R[6] = 4 et R[8] = 5 sont les opérandes d' *add*
- R[5] =16 pour *ld* et R[12] = 23 pour *str*

1er cycle: *add* entre dans le pipeline

- PC est incrémenté de 100 à 104

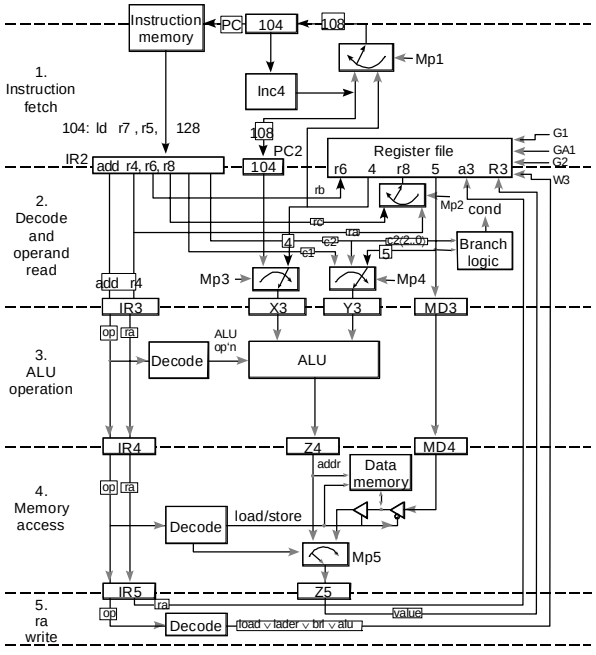
```
512: sub  ...
     . . . . . .
112: str  r12, #32
108: brl  r9, r11, 001
104: ld   r7, r5, #128
100: add  r4, r6, r8
```



2e cycle: *add* entre étage 2, *ld* entre étage 1

- Opérandes d'*add* sont recherchés dans étage 2

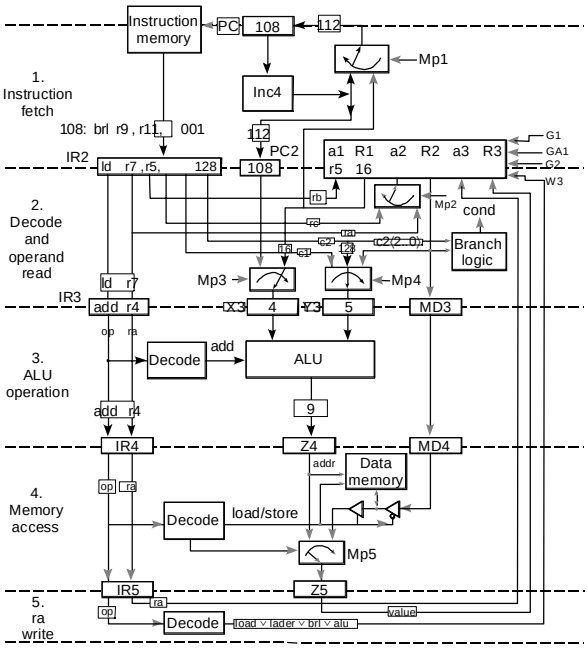
```
512: sub  ...
     . . . . . .
112: str  r12, #32
108: brl  r9, r11, 001
104: ld   r7, r5, #128
100: add  r4, r6, r8
```



3e cycle: *brl* entre étage 1, *add* et *ld* progressent

- *Add* **est effectué dans étage 3**

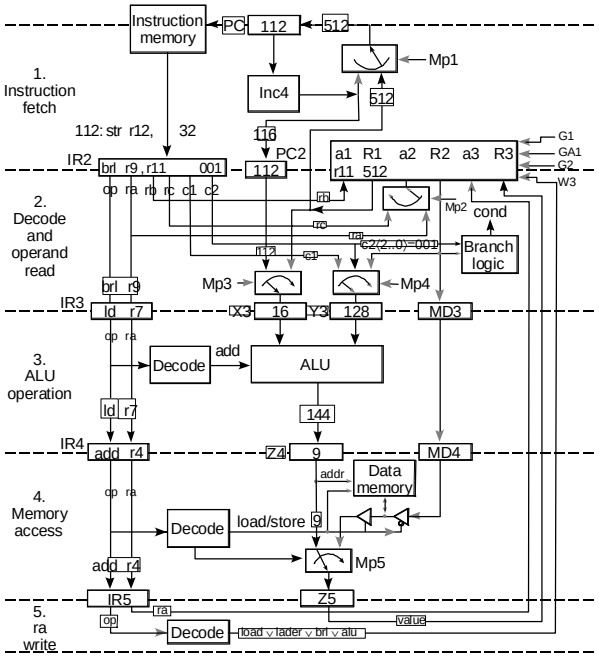
```
512: sub  ...
     . . . . . .
112: str  r12, #32
108: brl  r9, r11, 001
104: ld   r7, r5, #128
100: add  r4, r6, r8
```



4e cycle: *str* entre étage 1 ...

- *add* ne fait rien dans étage 4
- Succès de *brl* change le PC à 512

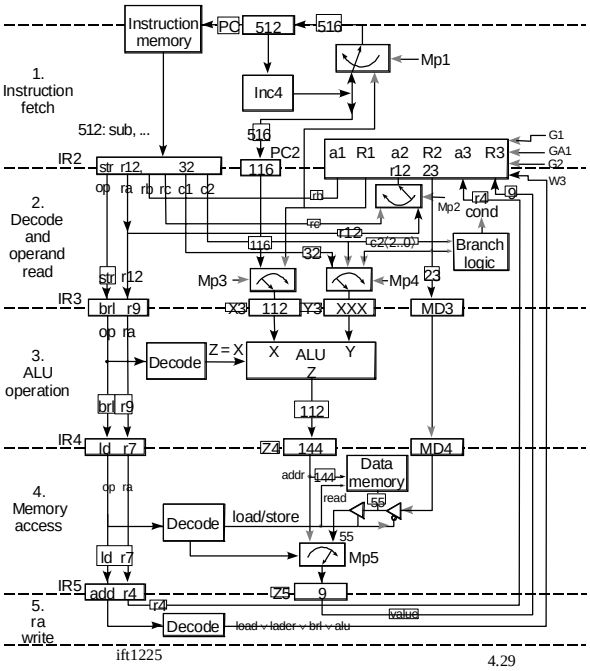
```
512: sub  ...
     . . . . . .
112: str  r12, #32
108: brl  r9, r11, 001
104: ld   r7, r5, #128
100: add  r4, r6, r8
```



5e cycle: *add* complète et *sub* entre dans le pipeline

- *add* finit dans étage 5
- *sub* est recherché de l'adresse 512 à cause de *brl*

```
512:  sub   ...
      . . . . . .
112:  str   r12, #32
108:  brl   r9, r11, 001
104:  ld    r7, r5, #128
100:  add   r4, r6, r8
```



Dépendances entre instructions dans le pipeline: Les aléas

- Les instructions dans le pipeline exécutent toutes en parallèle
- les opérandes d'une peuvent dépendre du résultat de l'autre instruction dans le pipeline qui n'a pas encore complété son exécution
- Deux types des aléas:
  - Aléa de données: Utilisation incorrecte de données passées ou futures
  - Aléa de branchement: Une instruction incorrecte est recherchée suite à un changement de PC dû à un branchement

Aléas de données

- Lecture après écriture: (*Read after write* - RAW) provient de la dépendance de flux de données (*flow dependence*) où une instruction utilise les données produites par une instruction précédente
- Écriture après lecture: (*Write after read* - WAR) provient de l'anti-dépendance où une instruction écrit une nouvelle valeur avant qu'une instruction précédente puisse lire l'ancienne valeur
- Écriture après écriture (Write after write - WAW) provient de la dépendance de sortie (*output dependence*), où deux instructions parallèles écrivent le même registre et devraient le faire dans l'ordre d'émission de ces instructions

?? Est-ce qu'une lecture après une autre lecture peut produire un aléa…??

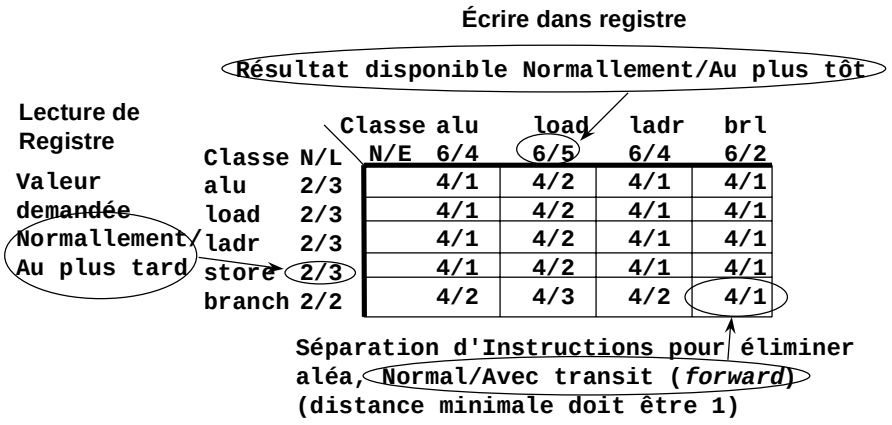
Aléas de données dans SRC

- Tout accès à la mémoire de données est dans l'étage 4, donc les écritures et lectures de la mémoire sont séquentielles - pas d'aléa ici
- Tout écriture de registre dans l'étage 5, donc pas d'aléa WAW et WAR
  - Deux écritures sont toujours effectuées dans l'ordre d'émission et toute écriture suit une lecture émise préalablement
- Tout aléa dans SRC est lié avec accès aux registres généraux et est limité à RAW dû à la dépendance de flux de données
- Écriture dans un registre est dans l'étage 5, mais la valeur peut être demandée par l'instruction suivante au début de l'étage 2!
- Pourrait être évitée par le programmeur - peu pratique
- Besoin de solutions automatiques (détection facile):
  - calage (perte de performance)
  - transit direct de données (*data forwarding*) à l'étage 2 sans passer par le registre

Détection des aléas et la distance de dépendance

- Pour détecter les aléas il faut considérer des paires d'instructions
- Données sont normalement disponibles après être écrite dans le registre, **mais**
- peuvent être envoyées dès leur production dans un étage
  - étage 3 sort les résultats de toute opération UAL,
  - étage 4 produit toute donnée recherchée dans la mémoire
- Opérandes normalement demandés dans l'étage 2, **mais**
- il suffit de les recevoir dans l'étage de leur utilisation:
  - étage 3 pour les opérandes UAL et les modificateurs d'adresse,
  - étage 4 pour ranger le contenu d'un registre, et
  - étage 2 pour la cible de branchement

Interaction entre paires d'instructions



- Étage 3 pour *store* est dû au calcul d'adresse effective; la valeur à ranger n'est nécessaire que dans l'étage 4
- *Store* aussi demande un operande de *ra*

Délais avec le transit de données

- La colonne *Load* : La valeur du résultat ne peut pas être disponible à l'instruction suivante!
  - Compilateur ne devrait pas mettre une instruction dépendante dans cette position (pour éviter le calage du pipeline)
- Le registre cible du branchement ne peut pas être modifié par l'instruction précédente
  - Programme est restreint à ne pas modifier le registre cible par l'instruction juste avant le branchement ou même 2 instructions s'il s'agit d'un chargement (*load*) de la mémoire
  - NE PAS CONFONDRE avec le retard de branchement qui est une dépendance de la recherche d'instruction suivante sur le calcul de l'adresse de branchement (et non une dépendance de branchement sur une instruction précédente)

Calage du pipeline dû à un aléa

- Lorsque l'on détecte un aléa, le contrôle peut caler les étages en avant, en attendant les étages suivants à compléter leurs opérations (et ainsi éliminer l'aléa)
- Une façon simple de caler un étage est d'inhiber l'horloge vers les registres destinations de l'étage - aucun de ces registres n'est modifié
- P.e., si les étages 1 et 2 calent, alors il faut donner quelqe chose à l'étage 3 pour continuer à vider le pipeline - l'insertion d'un NOP est un choix naturel

Exemple de détection d'aléa et de calage

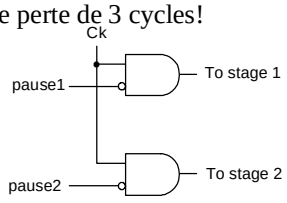
- Détection d'aléa et calage à cause de 2 instructions arithmétiques dans les étages 2 et 3

  (alu3 ∧ alu2 ∧ ((ra3 = rb2) ∨ (ra3 = rc2) ∧¬imm2)) →
  (pause2: pause1: op3 ← 0): … op = 0 est NOP
- Après ce calage, l'aléa apparaîtra entre les étages 2 et 4:

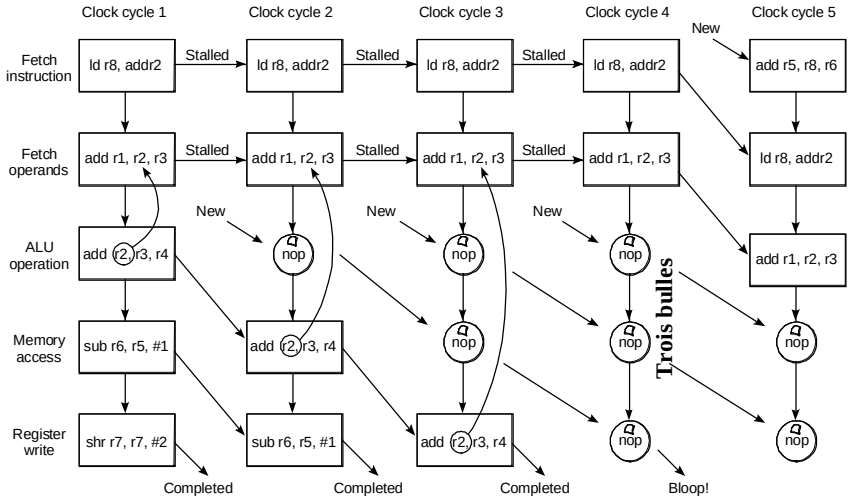
  (alu4 ∧ alu2 ∧ ((ra4 = rb2) ∨ (ra4 = rc2) ∧¬imm2)) →
  (pause2: pause1: op3 ← 0): ...
- Les aléas entre étages 2 et 5 requièrent:

  (alu5 ∧ alu2 ∧ ((ra5 = rb2) ∨ (ra5 = rc2) ∧¬imm2)) →
  (pause2: pause1: op3 ← 0): … une perte de 3 cycles!

Commande d'horloge pour caller les étages 1 et 2 du pipeline



Exemple de détection d'aléa et de calage (suite)



Transit de données (UAL à UAL)

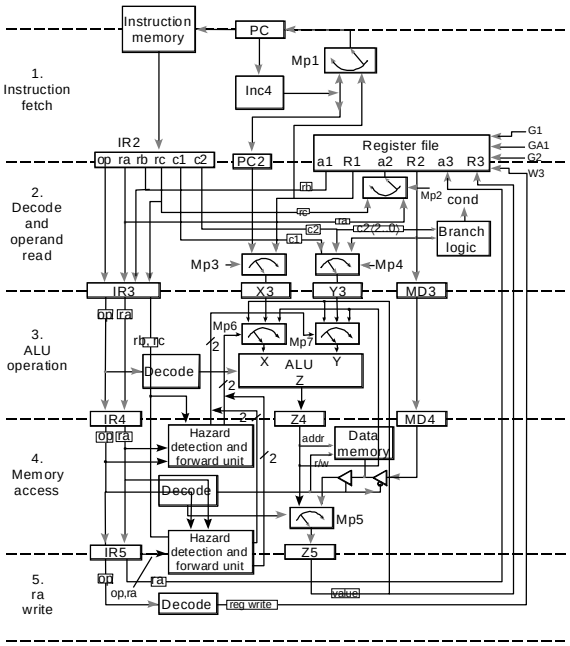
- La table de dépendances indique que 2 instructions dépendantes peuvent être adjacentes (plutôt que 4 cycles à part) si le transit de données de l'étage source à l'étage 3 est réalisé. Il faut passer les données de leur origine dans le registre X2 ou Y2 de l'étage 3
- L'équation décrivant la dépendance d'une instruction UAL dans l'étage 3 sur le résultat d'une autre instruction dans l'étage 5 et le transfert devient comme suit:

  $alu5 \wedge alu3 \rightarrow ((ra5 = rb3) \rightarrow X \leftarrow Z5:$
  $(ra5 = rc3) \wedge \neg imm3 \rightarrow Y \leftarrow Z5):$
- Et pour l'étage 4 à 3:

  $alu4 \wedge alu3 \rightarrow ((ra4 = rb3) \rightarrow X \leftarrow Z4:$
  $(ra4 = rc3) \wedge \neg imm3 \rightarrow Y \leftarrow Z4):$
- *rb* et *rc* doivent être visibles dans l'étage 3 pour détecter l'aléa
- Multiplexeurs sur les entrées des registres X and Y de l'UAL pour remplacer leur contenus par les données en transit des autres étages

Transit de données (UAL à UAL) (suite)

- De Z4 ou Z5 à X ou Y, entrée de l'UAL
- *rb* et *rc* sont nécessaires dans l'étage 3 pour la détection des aléas



Restriction après la réalisation du transit

- **(1) Position du retard de branchement (*branch delay slot*):**
  - **L'instruction après branchement est toujours exécutée**
- **(2) Position du retard de chargement (*load delay slot*):**
  - **Le registre chargée de la mémoire ne peut pas être utilisé dans l'instruction suivante comme opérande**
  - **Et il ne peut pas être utilisé comme la cible de branchement pendant les 2 instructions suivantes**
- **(3) Cible de branchement (*branch target*)**
  - **Le registre résultat d'une instruction UAL ne peut pas être utilisé comme la cible de branchement par l'instruction suivante**

```
br r4
add . . .
• • •

ld r4, 4(r5)
nop
neg r6, r4

ld r0, 1000
nop
nop
br r0

not r0, r1
nop
br r0
```


Parallélisme au niveau d'instructions

- Un pipeline même si rempli ne peut completer qu'une instruction par cycle d'horloge
  - Appelée quelquefois la limite de Flynn
- Si l'on recherche plus d'une instruction à la fois et s'il y a plusieurs unités fonctionnelles (UAL, multipliers, diviseurs, décaleurs, etc.), alors il est possible d'émettre plusieurs instructions en même temps
- Deux approches:
  - **Superscalaire**: Émettre dynamiquement autant d'instructions prérecherchées que possible aux unités fonctionnelles en attente
  - **Mot d'instruction très large** (***Very Long Instruction Word (VLIW)***): Statiquement, compiler plusieurs instructions dans un seul mot, avec une opération par unité disponible (autant que possible). Toutes les opérations sont exécutées en parallèle

Machines à émission d'instructions multiples

- Types des unités fonctionnelles
  - Virgule flottante
  - Entier
  - Branchement
- Plus d'une unité de chaque type; peuvent être elles mêmes à pipeline
- Problème restant - brise du flux d'instruction par des branchements
  - Peu de cycles d'horloge (instructions) entre deux branchements pour obtenir un haut degré du parallélisme
  - Aide par logiciel - déroulement des boucles d'itération
  - Aide par matériel - unité de branchement avec prédiction de la direction du branchement
    - Instructions à la cible de branchement peuvent être prérecherchées et même spéculativement exécutées (sans ranger le résultat dans un registre ISA visible au programmeur)