# TD-2 : Pipelines scalaires et prédiction de branchement

## Exercice 1

On considère une architecture RISC possédant 4 types d'instructions :

- A = arithmétiques et logiques : registre- registre
- C = chargement : lecture mémoire
- R = rangement : écriture mémoire
- B = branchement. Les branchements conditionnels se font sur code condition

Cette architecture a deux implantations, l'une avec le pipeline P1 dont les phases sont décrites table I et l'autre avec le pipeline P2 qui est décrit table II. Dans les deux cas, la mémoire d'instructions et la mémoire de données sont séparées.

| | |
|---|---|
| LI | Lecture de l'instruction |
| LR | Pour toutes les instructions, décodage de l'instruction et lecture des opérandes registre.<br>Pour les types C et R, calcul de l'adresse mémoire<br>Pour le type B, calcul de l'adresse de branchement |
| EX | Pour le type A, opération UAL.<br>Pour le type R, lecture du 3ème opérande.<br>Pour les types C et R, accès mémoire |
| RR | Rangement du résultat arithmétique (A) ou de la lecture (C) dans le banc de registres. |

*Table I: le pipeline P1*

| | |
|---|---|
| LI1 | Début lecture de l'instruction |
| LI2 | Fin lecture de l'instruction |
| LR | Décodage de l'instruction et lecture des opérandes registre |
| EX | Pour le type A, opération UAL.<br>Pour le type R, lecture du 3ème opérande.<br>Pour les types C et R, calcul de l'adresse mémoire.<br>Pour le type B, calcul de l'adresse de branchement |
| M1 | Pour les types C et R, début accès mémoire |
| M2 | Pour les types C et R, fin accès mémoire |
| RR | Rangement du résultat arithmétique (A) ou de la lecture (C) dans le banc de registres. |

*Table II : le pipeline P2*

1) Définir l'ensemble des dépendances qui peuvent exister entre une instruction qui génère une donnée ou une adresse et une instruction qui utilise cette donnée ou cette adresse.

2) Pour toutes les dépendances définies dans la question précédente, on introduit tous les mécanismes d'anticipation nécessaires (bypass) pour supprimer ou réduire les suspensions. Définir dans chaque cas le « bypass » nécessaire et en déduire le nombre de cycles de suspension (avec 0 quand il n'y a pas de suspension).

3) Quelle est la taille du délai de branchement pour P1 et pour P2 ?

4) Quelles ont les ressources matérielles nécessaires aux deux pipelines ?

## Exercice 2(optionnel)

Dans cet exercice, on regarde l'impact des modèles différents du modèle (3,0) pour lesquels les instructions arithmétiques et logiques vont chercher un opérande en mémoire.
On ajoute le support pour les opérations UAL registre-mémoire au pipeline RISC classique à cinq étages. Pour compenser cette augmentation en complexité, l'adressage mémoire sera limité au mode indirect par registre (i.e., toutes les adresses sont simplement une valeur contenue dans un registre: Aucun déplacement ne peut être ajouté à la valeur du registre). Par exemple, l'instruction registre-mémoire ADD R4, R5, (R1) signifie ajouter le contenu du registre R5 au contenu de la case mémoire dont l'adresse est égale à la valeur dans le registre R1 et ranger la somme dans le registre R4. Les opérations UAL registre-registre. sont inchangées Répondre aux questions suivantes pour le pipeline RISC entier.

a) Donner un ordre réarrangé des cinq étages traditionnels du pipeline RISC qui pourrait supporter les opérations registre- mémoire implémentées en utilisant exclusivement l'adressage indirect par registre.

b) Décrire les nouveaux chemins d'envoi nécessaires pour le pipeline réarrangé en spécifiant la source, la destination et l'information transférée sur chaque nouveau chemin nécessaire.

c) Pour les étages réordonnés du pipeline RISC, quels nouveaux aléas de données sont créés par ce mode d'adressage? Donner une séquence d'instructions illustrant chaque nouvel aléa.

d) Lister toutes les cas pour lesquelles le pipeline avec les opérations UAL registre-mémoire peut avoir un nombre d'instructions différent pour un programme donné de celui du pipeline RISC originel. Donner deux séquences spécifiques d'instructions, une pour le pipeline originel et une pour le pipeline réarrangé, pour illustrer chaque cas

# Exercice 3 : prédicteurs de branchement

Soit le programme C suivant :

```
long a=0, b=0, n=0, p=0;
main()
{
int i, n, p;
for (i=0; i<24; i++)
      {
      a = (a+1)%2
      b= (b+1)%3
      if (a>=b)
            n++;
            else p++;
      }
}
```

On considère le branchement conditionnel correspondant au if(a>=b). Le branchement est pris (P) si (a<b) et non pris (NP) autrement.
On associe un prédicteur à ce branchement. On utilise soit un prédicteur 1 bit, soit un compteur 2 bits. Le compteur 2 bits a 4 états : fortement non pris (FNP), faiblement non pris (fNP), faiblement pris (fP) et fortement pris (FP) auxquels on peut associer les valeurs 0, 1, 2 et 3.
Dans tous les cas, les prédicteurs sont initialisés à NP (ou FNP pour le compteur 2 bits)

Pour les 24 itérations de la boucle, quel est le nombre de prédictions correctes dans les cas suivants :

a) on utilise un prédicteur 1 bit par branchement
b) on utilise un prédicteur 2 bits par branchement
c) on utilise l'historique local des 2 derniers branchements, avec un prédicteur 1 bit par configuration du registre d'historique
d) on utilise l'historique local des 3 derniers branchements, avec un prédicteur 1 bit par configuration.

# Exercice 4 : prédicteurs de branchement (optionnel)

Soit le programme C suivant, auquel on associe des branchements B1, B2, B3 et B4 :

```
for (m=0;m<10;m++)
  {
        if (a[m]!=0)                                    // B1 pris si la condition est fausse
              {if ((b[m]*b[m]-4*a[m]*c[m])>=0)          // B2 pris si la condition est fausse
                    i++;}
              else
                    if (b[m]!=0)                        //B3 pris si condition fausse
                          j++;
                          else k++;
  }                                                     // B4 : branchement de fin de boucle pris si m <10
```

On suppose que le contenu des vecteurs a, b et c est le suivant

| | *a[m]* | *b[m]* | *c[m]* | *B1* | *B2* | *B3* | *B4* |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 5 | 2 | | | | |
| **1** | 1 | 4 | 3 | | | | |
| **2** | 2 | 3 | 4 | | | | |
| **3** | 0 | 0 | 5 | | | | |
| **4** | 1 | 1 | 3 | | | | |
| **5** | 2 | 2 | 4 | | | | |
| **6** | 0 | 3 | 3 | | | | |
| **7** | 1 | 4 | 2 | | | | |
| **8** | 2 | 5 | 1 | | | | |
| **9** | 0 | 0 | 0 | | | | |

1) Donner le comportement des branchements dans le tableau ci-dessus, sous la forme P (pris) et NP (non pris)

2) Donner le nombre de mauvaises prédictions de branchements en supposant une prédiction statique, où tous les branchements avant sont prédits Non pris (NP) et les branchements arrière sont prédits Pris (P)

3) On suppose que chacun des branchements B1 à B4 a un prédicteur 1 bit, initialisé à NP pour les branchements avant, et P pour les branchements arrière. Donner le nombre de mauvaises prédictions

4) On suppose que chacun des branchements B1 a B4 a un prédicteur 2 bits, initialisé à FNP pour les branchements avant (fortement NP) et FP pour les branchements arrière (fortement P). Donner le nombre de mauvaises prédictions.

5) On utilise un prédicteur 2 niveaux, avec un bit d'historique global et un prédicteur 1 bit par branchement, initialisé à NP (avant) ou P (arrière). Donner le nombre de mauvaises prédictions. On supposera que le branchement précédent le début d'exécution était NP