TD 3 - Pipeline In-Order

Hugo Bolloré[1] and Mohammed Salah Ibnamar[2]

[1]hugo.bollore@uvsq.fr
[2]mohammed-salah.ibnamar@uvsq.fr

Pour l'ensemble de ce TD nous utiliserons le jeu d'instruction suivant :

Instruction	Description
LOAD *%Rx, (%Ry, %Rz)*	Charge la valeur pointé par l'opérande mémoire dans le registre Rx
MUL *%Rx, %Ry, %Rz*	Multiple la valeur du registre Ry avec la valeur du registre Rz et écrit le résultat dans le registre Rx
ADD *%Rx, %Ry, %Rz*	Additionne la valeur du registre Ry avec la valeur du registre Rz et écrit le résultat dans le registre Rx
ADDI *%Rx, %Ry, #N*	Additionne la valeur du registre Ry avec la valeur N et écrit le résultat dans le registre Rx
SUBI *%Rx, %Ry, #N*	Soustrait la valeur du registre Ry avec la valeur N et écrit le résultat dans le registre Rx
JNZ *%Rx, label*	Jump au label si la valeur du registre Rx est différente de 0

Exercice 3.1

On considère le pipeline d'instructions représenté ci-dessous :

Lecture du cache d'instructions
Décodage de l'instruction
Lecture des registres
Exécution / calcul de l'adresse
Accès au cache de données
Accès au cache de données
Ecriture des registres

Avec les caractéristiques suivantes :

1. Le débit théorique maximum de ce pipeline est de 1 instruction/cycle,
2. Les instructions, sauf les *LOAD/STORE*, sont exécutées à l'étage 4,
3. Pour les *LOAD/STORE*, le calcul d'adresse se fait à l'étage 4 et l'accès au cache de données est pipeliné sur 2 cycles (étages 5 et 6),
4. Pour les instructions qui ne sont pas des LOAD/STORE, les étages 5 et 6 se comportent comme des étages vides,
5. Il y a un mécanisme de bypass permettant de transmettre le résultat d'une instruction aux instructions suivantes sans attendre l'écriture registre.
6. Si un opérande source d'une instruction n'est pas disponible au moment où celle-ci va rentrer dans l'étage 4, les étages 3 est bloqué.
7. On supposera un prédicteur de branchement parfait.
8. On supposera également que toutes les lectures d'instruction font des hits dans le cache d'instructions et que tous les LOAD/STORE font des hits dans le cache de données.

Question 3.1.1

Lorsqu'un LOAD est immédiatement suivi d'une instruction utilisant le résultat du LOAD, combien de *bulles* sont insérées dans le pipeline ?

Question 3.1.2

On considère le programme ci-dessous, qui calcule la somme des N éléments d'un tableau. RSI est initialisé avec N, RAX est initialisé avec l'adresse du tableau, RBX est initialisé à 0.

```
boucle: LOAD %RCX, (%RAX, %RBX)
        ADD %RDX, %RDX, %RCX
        ADDI %RBX, %RBX, #4
        SUBI %RSI, %RSI, #1
        JNZ %RSI, boucle
```

En supposant que N est grand, quel est le débit d'exécution de cette boucle en instructions par cycle ?

Question 3.1.3

Pourrions-nous changer l'ordre des instructions dans la boucle afin d'obtenir un débit de une instruction par cycle ?

Question 3.1.4

On considère le programme ci-dessous qui calcule la somme des élément du tableau mais en parcourant le tableau dans l'autre sens. RBX est initialisé à $4 * N$ et RAX est initialisé à l'adresse du tableau - 4.

```
boucle: LOAD %RCX, (%RAX, %RBX)
        SUBI %RBX, %RBX, #4
        ADD %RDX, %RDX, %RCX
        JNZ %RSI, boucle
```

Ce programme est-il plus performant que le programme de la question 1.2 ? Est-il plus performant que le programme modifié à la question 1.3 ?

Question 3.1.5

En prenant en compte que l'on s'autorise à initialiser les registres différement, pourrions-nous changer l'ordre des instructions dans la boucle du programme de la question 1.4 afin d'améliorer les performances ?

Question 3.1.6

En déroulant 2 fois la boucle du programme de la question 1.2, on obtient le programme ci-dessous. RAX est initialisé avec l'adresse du tableau, RBX est initialisé avec l'adresse du tableau + 4 et RSI est initialisé avec $N/2$ (on suppose que N est pair) :

```
boucle : LOAD %RCX, (%RAX, %R9)
         LOAD %R8, (%RBX, %R9)
         ADDI %R9, %R9, #8
         ADD %RDX, %RDX, %RCX
         ADD %RDX, %R8, %R8
         SUBI %RSI, %RSI, #1
         JNZ %RSI, boucle
```

Ce programme est-il plus performant que le programme trouvé à la question 1.5, et si oui, quel est le facteur d'accélération ?

Question 3.1.7

On considère un autre processeur qui a le pipeline d'instructions représenté ci-dessous, c'est-à-dire qui déplaçe l'étage d'exécution de l'étage 4 à l'étage 6. Le calcul d'adresse continue à se faire à l'étage 4.

Lecture du cache d'instructions
Décodage de l'instruction
Lecture des registres
Calcul de l'adresse
Accès au cache de données
Exécution / Accès au cache de données
Ecriture des registres

La performance du programme de la question 1.4 sur le nouveau pipeline est-elle supérieure ou inférieure à celle sur l'ancien pipeline ?

Exercice 3.2

On considère le pipeline d'instructions ci-dessous :

Lecture du cache d'instructions
Décodage de l'instruction
Lecture des registres
Exécution de l'instruction
Exécution de l'instruction
Exécution de l'instruction
Ecriture des registres

On suppose de nouveau un prédicteur de branchements parfait et on suppose que les instructions sont déjà dans le cache d'instruction. On veut calculer $ax^3 + bx^2 + cx + d$. On propose deux méthodes possibles pour faire ce calcul :

- Méthode de Horner : $d + x(c + x(b + xa))$
- Méthode du TD : $(d + cx) + ((x * x) * (ax + b))$

On remarquera que la méthode du TD demande une multiplication de plus que la méthode de Horner.

Pour simplifier, on considérera que les instructions définies précédemment peuvent faire les calculs sur des nombres flottants et que les registres standard peuvent stocker des valeurs flottantes (ce n'est normalement pas le cas). On supposera que x, a, b, c, d se trouvent déjà dans les registres RAX, RBX, RCX, RDX, RSI. On veut le résultat dans le registre RDI.

Question 3.2.1

Quelle méthode est la plus performante sur le pipeline considéré ?