TD 6 : Pipelines & Opérations Multicycles & Prédiction De Branchement

Prénom : **Nom :**

1 Pipelines

Un processeur *P1* dispose du pipeline à 5 étages pour les opérations entières et les chargements / rangements flottants et des pipelines suivants pour les instructions flottantes :

Multiplication : `fmul`	LI	DI	LR	EX1	EX2	EX3	EX4	ER
Addition : `fadd`	LI	DI	LR	EX1	EX2	ER		

Les instructions flottantes `lf` (Load Float) et `sf` (Store Float) ont les mêmes pipelines que les instructions entières LW (Load Word) et SW (Store Word).

Une instruction i a une latence de n si l'instruction suivante peut commencer au cycle $i+n$. Une latence de 1 signifie que l'instruction suivante peut commencer au cycle suivant.

1.1 Donner les latences des instructions flottantes `fmul` et `fadd`.

1.2 Donner les latences des instructions flottantes `fmul` et `fadd` pour le processeur *P2* suivant.

Multiplication : `fmul`	LI	DI	LR	EX1	EX2	EX3	EX4	EX5	EX6	ER
Addition : `fadd`	LI	DI	LR	EX1	EX2	EX3	EX4	ER		

2 Latences

Soit le jeu d'instructions suivant : (on dispose des registres entiers R0 à R31 et des registres flottants F0 à F31)

Instructions qui travaille sur des flottants :	
`lf Fdest, address`	Charge un flottant depuis l'adresse (l'adresse peut être de la forme `N(R)` avec `N` un immédiat et `R` un registre entier)
`sf Fsrc, address`	Stocke un flottant dans l'adresse
`fadd Fdest, Fsrc1, Fsrc2`	`Fdest` ← `Fsrc1` + `Fsrc2` (addition flottante simple précision)
`fsub Fdest, Fsrc1, Fsrc2`	`Fdest` ← `Fsrc1` − `Fsrc2` (soustraction flottante simple précision)
`fmul Fdest, Fsrc1, Fsrc2`	`Fdest` ← `Fsrc1` × `Fsrc2` (multiplication flottante simple précision)
`fdiv Fdest, Fsrc1, Fsrc2`	`Fdest` ← `Fsrc1` / `Fsrc2` (division flottante simple précision)
Instructions qui travaille sur des entiers :	
`lw Rdest, address`	Charge un entier depuis l'adresse (l'adresse peut être de la forme `N(R)` avec `N` un immédiat et `R` un registre entier)
`sf Rsrc, address`	Stocke un entier dans l'adresse
`add Rdest, Rsrc1, Rsrc2`	`Rdest` ← `Rsrc1` + `Rsrc2` (addition sur des entiers)
`addi Rdest, Rsrc1, Rsrc2`	`Rdest` ← `Rsrc1` + `Rsrc2` (addition sur des entiers non signés)
Branchements :	
`bne Ra, Rb, Label`	Si `Ra` ≠ `Rb` : `goto Label`
`bneq R, Label`	Si `R` ≠ 0 : `goto Label`

Les additions, soustractions et multiplications flottantes sont pipelinées. Une nouvelle instruction peut démarrer à chaque cycle. Les latences sont de 2 cycles pour `lf` et fournies par les résultats de l'exercice 1 pour les multiplications et additions flottantes.

La division a une latence de 15 cycles. Elle n'est pas pipelinée : une nouvelle division ne peut commencer que lorsque la division précédente est terminée.

```
float x[100];   float y[100];   float z[100];

for (size_t i = 0; i < 100; ++i)
{
    z[i] = x[i] * y[i];
}
```

2.1 Traduire ce code C en assembleur en énumerant les numéros de cycles à côté des instructions pour les deux processeurs de l'exercice 1. Les tableaux `x`, `y` et `z` sont rangés à la suite en mémoire. `R1` contient déjà l'adresse de `x[0]` et `R3` l'adresse de `x[100]` (l'adresse de fin de `x`).

2.2 Donner le code C de la boucle après déroulage d'ordre 2.

2.3 Donner le code C de la boucle après déroulage d'ordre 4.

2.4 Traduire le code C déroulé avec un facteur de 2 en assembleur en énumerant les numéros de cycles à côté des instructions pour les deux processeurs.

2.5 Traduire le code C déroulé avec un facteur de 4 en assembleur en énumerant les numéros de cycles à côté des instructions pour les deux processeurs.

2.6 Pour chaque code et chaque processeur : quel est le nombre de cycles par itération ?

2.7 Quel est le facteur de déroulage maximal qui permettrait d'améliorer le nombre de cycle par itération ? Quel serait le nombre des différentes instructions (chargements, multiplications, ...) ? le nombre de cycles ? Le nombre de cycles par itération ?

2.8 Donner le code C de la boucle après déroulage d'ordre 2 dans le cas général, c'est-à-dire en remplaçant 100 par `N`.

3 Somme Des Carrés (Bonus)

Soit le code C suivant :

```
float x[100];   float s = 0.f;

for (size_t i = 0; i < 100; ++i)
{
    s += x[i] * x[i];
}
```

3.1 Mêmes questions que pour l'exercice 2.

4 Multiplication Vs Division

Soit les code C suivants équivalents mathématiquement :

```
float x[100]; float y[100]; float z[100]; float a;
// Calcul de a


for (size_t i = 0; i < 100; ++i)
{
    z[i] = x[i] / a + y[i];
}
```

```
float x[100]; float y[100]  float z[100]; float a;
// Calcul de a
float a_inv = 1.f / a;

for (size_t i = 0; i < 100; ++i)
{
    z[i] = x[i] * a_inv + y[i];
}
```

Dans cet exercice, on utilise les latences suivantes :

Instructions	Latence	Pipelinée ?
`lf`	2	oui
`fadd`, `fsub`	3	oui
`fmul`	5	oui
`fdiv`	15	non

Les tableaux `x`, `y` et `z` sont rangés à la suite en mémoire.
`R1` contient déjà l'adresse de `x[0]` et `R3` l'adresse de `x[100]` (l'adresse de fin de `x`).
`a` est déjà calculé dans `F31` pour le premier code et `a_inv` est déjà calculé dans `F31` pour le deuxième.

4.1 Pour chaque code C sans déroulage et avec un déroulage de 2, écrire le code assembleur correspondant en énumerant les numéros de cycle.

5 Prédiction De Branchement

Soit le code C suivant :

```
int a = 0;   int b = 0;   int n = 0;   int p = 0;

for (size_t 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 (NP, P), soit un compteur 2 bits (4 états : FNP (fortement non pris), fNP (faiblement non pris), fP (faiblement pris), FP (fortement pris)). Les prédicteurs sont initialisés respectivement à NP et FNP.

5.1 Quel est le nombre de prédictions correctes avec :

· un prédicteur 1 bit ?
· un prédicteur 2 bits ?
· si on utilise l'historique des 2 derniers branchement avec un prédicteur 1 bit par configuration du registre d'historique ?
· si on utilise l'historique des 3 derniers branchements avec un prédicteur 1 bit par configuration

6 Place libre (remarques, impressions, fin de réponse...)