TD 8 - Pipeline & Opérations multicycles

1. Pipeline

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 lf (load float) et sf (strore float) ont les mêmes pipelines que les instruction 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. Donner les latences des instructions flottantes fmul et fadd.
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 R21 et des registres flottants F0 à F31).

Instructions sur les flottans :	
lf Fdest, address	Charge un flottant depuis l'adresse (l'address est au format N(R) avec N un immédiat et R un registre entier
sf Fsrc, address	Stocker 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 sur les entiers :	
Lw Rdest, address	Charge un entier depuis l'adresse
sf Rsrc, address	Stocke un entier dans l'adresse
add Rdest, Rsrc1, Rsrc2	Rdest ← Rsrc1 + Rsrc2 (addition avec des entiers)
addi Rdest, Rsrc1, Imm	Rdest ← Rsrc1 + Imm (addition avec des entiers)
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.
On considère le code C qui suit

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

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

1. Traduire ce code C en assembleur en énumérant les numéros de cycles à côté des instructions pour les deux processeurs de l'exercice 1. Les tableaux x, y et z sont implémentés en mémoire à la suite. R1 contient déjà l'adresse de x[0] et R3 l'adresse de x[100] (l'adresse de fin de x).
2. Donner le code C de la boucle après déroulage d'ordre 2.
3. Donner le code C de la boucle après déroulage d'ordre 4.
4. Traduire le code C déroulé avec un facteur de 2 en assembleur et en énumérant les numéros de cycles à côté des instructions pour les deux processeurs.
5. Traduire le code C déroulé avec un facteur de 4 en assembleur et en énumérant les numéros de cycles à côté des instructions pour les deux processeurs.
6. Pour chaque code et chaque processeur : quel est le nomber de cycles par itération ?

3. Multiplication versus Division

On considère le code C qui suit

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

for (int i=0 ; i<100 ; i++)
{
        z[i] = x[i] / a + 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 implémentés en mémoire à la suite. 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 dabs F31.

1. Pour chaque code C sans déroulage et avec un déroulage de 2, écrire le code assembleur correspondant en énumérant les numéros de cycle.
2. Comment serait-il possible d'améliorer grandement le nombre de cycle par itération ?