TD 6 : Pipelines et opérations multi-cycles

Exercice 1

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 et SF ont les mêmes pipelines que les instructions entières LW et SW.
Donner les latences des instructions flottantes FMUL et FADD (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).

Mêmes questions avec 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

Exercice 2

On ajoute au jeu d'instructions MIPS32 des instructions flottantes simple précision (32 bits) (Figure 1) et 32 registres flottants $f0 à $f31 ($f0 est un registre normal).
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 instructions flottantes de multiplication et d'addition.
La division a une latence de 30 cycles. Elle n'est pas pipelinée : une nouvelle division ne peut commencer que lorsque la division précédente est terminée.

L.S	I	L.S $ft, déplac(rs)	$ft ← MEM [$rs +SIMM]
S.S	I	S.S $ft, déplac.(rs)	$ft → MEM [$rs + SIMM]
ADD.S	R	ADD.S $fd, $fs,$ft	$fd ← $fs + $ft (addition flottante simple précision)
MUL.S	R	MUL.S $fd, $fs,$ft	$fd ← $fs * $ft (multiplication flottante simple précision)
SUB.S	R	SUB.S $fd, $fs,$ft	$fd ← $fs - $ft (soustraction flottante simple précision)
DIV.S	R	DIV.S $fd,$fs,$ft	$fd ← $fs / $ft (division flottante simple précision)

Figure 1 : Instructions flottantes

Soit le programme suivant :

float X[100], Y[100], Z[100] ;

```
main(){
int i ;
for (i=0 ; i<100 ; i++)
        Z[i] = X[i]* Y[i];}
```

Quels sont les temps d'exécution par itération de la boucle avec le processeur P1 ? Avec P2 ?
On utilisera les hypothèses suivantes :
Les trois vecteurs X, Y et Z sont rangés successivement en mémoire. R1 contient l'adresse de X[0]. R3 contient l'adresse de X[100], c'est-à-dire l'adresse du mot suivant le dernier élément du tableau X

Soient des déroulages de boucle d'ordre 2 et 4

```
main(){
int i ;
for (i=0 ; i<100 ; i+=2){
      Z[i] = X[i]* Y[i];
      Z[i+1]= X[i+1]* Y[i+1];}

main(){
int i ;
for (i=0 ; i<100 ; i+=4){
      Z[i] = X[i]* Y[i];
      Z[i+1]= X[i+1]* Y[i+1];
      Z[i+2] = X[i+2]* Y[i+2];
      Z[i+3]= X[i+3]* Y[i+3];}
```

Quels sont les temps d'exécution par itération de la boucle initiale pour les 4 cas

- déroulage d'ordre 2 avec P1
- déroulage d'ordre 4 avec P1
- déroulage d'ordre 2 avec P2
- déroulage d'ordre 4 avec P2

Quels sont les temps minimum par itération de la boucle initiale que l'on peut obtenir par déroulage avec chaque processeur ?
Comment procède-t-on si le nombre d'itérations n'est pas un multiple du facteur de déroulage ?

Somme des carrés

Soit le programme suivant

```
float X[100], s ;
main(){
int i ;
for (i=1 ; i<100 ; i++)
        s+=X[i]*X[i];}
```

Quels sont les temps d'exécution par itération de la boucle avec le processeur P1 ? Avec P2 ?

S est initialement dans le registre F0. R1 contient l'adresse de X[0]. R3 contient l'adresse de X[100], c'est-à-dire l'adresse du mot suivant le dernier élément du tableau X

Quels sont les temps minimum par itération de la boucle initiale que l'on peut obtenir par déroulage avec chaque processeur ?

Exercice 3 (optionnel)

Soient les programmes P1et P2 :

P1	P2
float X[N], Y[N], Z[N], a ; int i ; for (i=0 ; i<N ; i++) Z[i]= X[i] /a + Y[i] ;	int i ; float X[N], Y[N], Z[N], a, c ; c =1.0/a ; for (i=0 ; i<N ; i++) Z[i]= X[i] *c + Y[i] ;

Les instructions flottantes utilisables sont

Instructions	Latence	Pipelinée ?
L.S	2	OUI
ADD.S, SUB.S	3	OUI
MUL.S	5	OUI
DIV.S	15	NON

Quelle est la version la plus rapide entre les programmes P1 et P2 (justifier). Pour la version la plus rapide,

- écrire la version optimisée sans déroulage de boucle, donner le nombre de cycles par itération et le nombre de cycles total si N=100
- écrire la version optimisée avec déroulage de boucle d'ordre 2, donner le nombre de cycles par itération et le nombre de cycles total si N=100.

Exercice 4 (optionnel)

Soit le code

```
Float X[1000], Y[999];
For (i=1; i<1000; i++)
    Y[i-1] = X[i]) + X[i-1];
```

Avec les latences de l'exercice 3, quel est le temps d'exécution total du programme pour la version optimisée sans déroulage de boucle ?