TD 6 : Pipeline, prédiction de saut

Exercice 1 - Aléas du pipeline

On considère le code assembleur [1] suivant.

```
ld  r2, 0(r0)
ld  r3, 4(r0)
add r3, r2, r3
st  r3, 0(r1)
sub r3, r3, r2
ld  r4, 8(r0)
add r4, r3, r4
st  r4, 4(r1)
```

1.1 On suppose que `r0` et `r1` contiennent des adresses vers deux tableaux d'entiers (type `int32_t`). Décrire ce que fait le code ci-dessus.

1.2 On veut utiliser le pipeline à 4 étages décrit en cours. Lister les problèmes d'aléas que présente le code ci-dessus.

1.3 Proposer un nouveau code où ces problèmes sont résolus grâce à l'ajout d'instructions `nop`.

1.4 On suppose que le processeur est muni d'un système de *bypass* permettant de récupérer la valeur d'un registre juste avant que celle-ci soit effectivement écrite dans le banc de registre.

Proposer un nouveau code assembleur, qui résout toujours les aléas grâce à des instructions `nop` tout en tenant compte de la présence du *bypass*.

1.5 Que peut-on faire pour améliorer encore les performances ?

Exercice 2 - Prédiction de sauts

On considère le morceau de code `C` suivant :

```
for (int i = 0; i<n; ++i) {
    if (i%2 == 0) printf("*");
    else          printf(" ");
}
```

2.1 Identifier les sauts conditionnels qui seront générés à la compilation de ce code.

2.2 Déterminer le nombre de prédictions correctes lorsqu'on utilise le prédicteur de sauts à 1 bit décrit en cours.

2.3 Déterminer le nombre de prédictions correctes lorsqu'on utilise le prédicteur de sauts à 2 bits décrit en cours.

Commenter.

1. Un descriptif du jeu d'instructions est fourni en annexe. Ignorer les latences/débits pour cet exercice.

Exercice 3 - Pipeline **(exam. 2021)**

On considère le code assembleur [2] suivant :

```
      ldi r4, 0
      ldi r5, N
loop: fld rf1, 0(r1)
      fmul rf1, rf1, rf1
      fld rf2, 0(r2)
      fmul rf2, rf2, rf2
      fadd rf1, rf1, rf2
      fst  rf1, 0(r3)
      addi r1, r1, 4
      addi r2, r2, 4
      addi r3, r3, 4
      addi r4, r4, 1
      jne  r4, r5, loop
```

3.1 On suppose que les valeurs initiales des registres $r1$, $r2$ et $r3$ sont respectivement les adresses des tableaux `A[N]`, `B[N]` et `C[N]` contenant des valeurs de type `float`.

Donner un code `C` équivalent au code assembleur.

3.2 Donner un ordonnancement du corps de boucle dans le cas d'un processeur de type VLIW à 4 voies avec les contraintes suivantes :

— Une instruction `fmul` doit forcément être lancée sur la voie 1 ;
— Une instruction `fadd` doit forcément être lancée sur la voie 2.

3.3 On suppose que N est divisible par 4, et on effectue un déroulage de boucle d'ordre 4.

Donner un nouvel ordonnancement sur le processeur VLIW, et estimer le gain par rapport à l'ordonnancement de la question précédente.

3.4 Expliquer comme faire pour dérouler davantage la boucle, et déterminer le nombre de cycles nécessaires pour réaliser K itérations du corps de boucle.

2. Un descriptif du jeu d'instructions est fourni en annexe.

Description du langage assembleur

instruction	latence	débit	effet
add a, b, c	1	1	$a \leftarrow b + c$
addi a, b, i	1	1	$a \leftarrow b + i$
sub a, b, c	1	1	$a \leftarrow b - c$
subi a, b, i	1	1	$a \leftarrow b - i$
mul a, b, c	1	1	$a \leftarrow b \times c$
div a, b, c	1	1	$a \leftarrow \lfloor b/c \rfloor$
ldi a, i	1	1	$a \leftarrow i$
ld a, d(c)	2	1	$a \leftarrow \text{MEM}[c + d]$
st a, d(c)	2	1	$\text{MEM}[c + d] \leftarrow a$
fadd x, y, z	3	1	$x \leftarrow y + z$
fsub x, y, z	3	1	$x \leftarrow y - z$
fmul x, y, z	3	1	$x \leftarrow y \times z$
fdiv x, y, z	6	6	$x \leftarrow y/z$
fld x, d(c)	2	1	$x \leftarrow \text{MEM}[c + d]$
fst x, d(c)	2	1	$\text{MEM}[c + d] \leftarrow x$
jmp l	1	1	saut à l'adresse l
jeq a, b, l	1	1	saut à l'adresse l quand $a = b$
jne a, b, l	1	1	saut à l'adresse l quand $a \neq b$
jlt a, b, l	1	1	saut à l'adresse l quand $a < b$

Légende :

- a, b, c = registres entiers
- x, y, z = registres flottants
- i, d = constantes immédiates entières
- l = adresse ou label