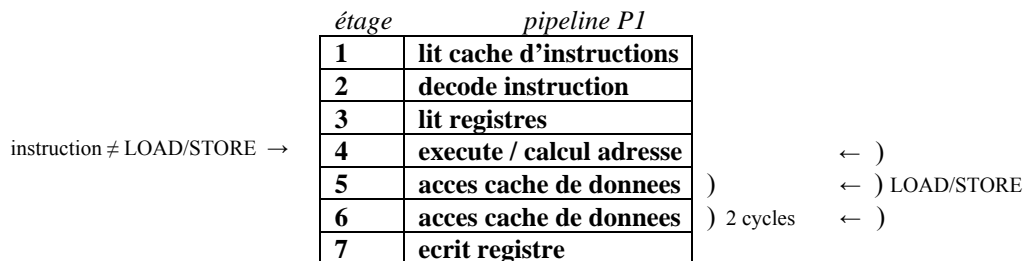


TD 4. Les Processeurs actuels. Pipelining

1. Pipelining

On considère le pipeline d'instructions (P1) représenté ci-dessous :



Le débit théorique maximum de ce pipeline est de 1 instruction/cycle.

On suppose que toutes les instructions, sauf les LOAD/STORE, sont exécutées à l'étage 4. Pour les instructions LOAD/STORE, le calcul d'adresse du/des opérandes du LOAD/STORE se fait à l'étage 4 et l'accès au cache de données est pipeliné sur 2 cycles (étages 5 et 6). Pour les instructions qui ne sont pas des LOAD/STORE, les étages 5 et 6 se comportent comme des étages vides.

On suppose qu'il y a un mécanisme de *bypass* permettant de transmettre le résultat d'une instruction aux instructions suivantes sans attendre l'écriture registre. Si un opérande source d'une instruction n'est pas disponible au moment où celle-ci va rentrer dans l'étage 4, les étages 1 à 3 sont bloqués jusqu'à ce que l'opérande indisponible soit accessible via le mécanisme de *bypass*, ce qui conduit à l'insertion de *bulles* dans le pipeline.

On supposera un prédicteur de branchement parfait. On supposera également que toutes les lectures d'instructions font des *hits* dans le cache d'instructions et que tous les LOAD/STORE font des *hits* dans le cache de données (cela signifie qu'il n'y a que des *cache-hits* et pas de *cache-miss*, c'est-à-dire que les accès cache sont toujours tels que le cache respectivement d'instructions / données contient les cibles, instructions / données).

1.

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

2.

On considère le programme asm ci-dessous, qui calcule la somme des N éléments d'un tableau. Le registre R4 est initialisé avec N , le registre R1 est initialisé avec l'adresse A du 1^{er} élément du tableau et le registre R3 contenant en fin d'exécution le résultat de la somme est initialisé à 0. Les éléments du tableau sont stockés sur 4 octets.

```

1:      boucle:  R2 = LOAD R1+0          // lit element du tableau
2:                      R3 = R3 ADD R2    // ajoute a la somme
3:                      R1 = R1 ADD 4      // R1 = R1 + 4
4:                      R4 = R4 SUB 1      // R4 = R4 - 1
5:                      BNZ R4, boucle     // BNZ (Branch Not Zero) : boucle si R4 ≠ 0

```

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

3.

Changez l'ordre des instructions dans la boucle afin d'obtenir un débit de 1 instruction/cycle.

4.

On considère le programme ci-dessous qui calcule la somme des éléments du tableau mais en parcourant le tableau dans l'autre sens. R4 est initialisé à 4N, R1 est initialisé à A-4N et R3 toujours initialisé à 0.

```

1:      boucle: R2 = LOAD R1+R4      // lit element du tableau
2:      R4 = R4 SUB 4              // R4 = R4 - 4
3:      R3 = R3 ADD R2             // ajoute a la somme
4:      BNZ R4, boucle            // BNZ (Branch Not Zero) : boucle si R4 ≠ 0

```

a) Toujours en supposant N grand, quel est le débit d'exécution du programme en instructions par cycle ?

b) Ce programme est-il plus performant que le programme de la question 2 ? Est-il plus performant que le programme modifié à la question 3 ?

5.

On modifie le pipeline d'instructions comme représenté ci-dessous (P2), c'est-à-dire en déplaçant l'étage d'exécution de l'étage 4 à l'étage 6. Le calcul d'adresse continue à se faire à l'étage 4.

étage	pipeline P2
1	lit cache d'instructions
2	decode instruction
3	lit registres
4	calcul adresse
5	Acces cache de donnees
6	execute / acces cache de donnees
7	ecrit registre

La performance du programme de la question 4 sur le nouveau pipeline (P2) est-elle supérieure ou inférieure à celle sur l'ancien pipeline (P1) ?
