TD 4 4MMARCA, Ensimag 2A/S6, ISI/SEOC

Architecture *pipeline* : aléas

Architecture avancée, 2018-2019

Le but de l'exercice est de mettre en évidence les problèmes liés à l'existence du *pipeline*. On dispose d'un processeur qui implante le jeu d'instructions du MIPS avec un pipeline à 5 étages, obtenu simplement en insérant des registres pour former les 5 étages (passage du processeur monocycle au processeur à 5 étages). Un schéma symbolique du processeur est donné en annexe, ainsi que le sous-ensemble du jeu d'instructions utilisé.

L'exécution d'une instruction se déroule en 5 cycles, et l'exécution d'instructions successives est pipelinée. L'exécution d'une séquence d'instruction i_i se déroule donc conceptuellement comme décrit ci-dessous :

instruction \ cycle	1	2	3	4	5	6	7	8	9
i_0	IF	DE	EXE	MEM	WB				
i_1		IF	DE	EXE	MEM	WB			
i_2			IF	DE	EXE	MEM	WB		
i_3				IF	DE	EXE	MEM	WB	
i_4					IF	DE	EXE	MEM	WB

Ex. 1 : Dépendances de données

On s'intéresse aux aléas créés par des dépendances de type lecture après écriture (RaW, Read after Write), c'est à dire les cas où il faut lire une valeur après l'avoir écrite. Par exemple :

```
sub $1, $2, $3
add $4, $1, $1
```

Dans la séquence d'instructions précédentes, on voit qu'il y a une dépendance RaW entre le résultat de l'instruction `sub` et le premier opérande de l'instruction `add`, ainsi que entre le résultat de l'instruction `sub` et le deuxième opérande de l'instruction `add`.

Pour différents couples d'instructions et de situations, nous allons étudier ces aléas, en indiquant :
— s'il y a un aléa,
— si oui, où est la valeur nécessaire à l'instruction dépendante et où elle devrait être.
Pour matérialiser la distance entre les instructions en dépendances, nous insérerons des instructions `nop` entre ces dernières.

Question 1 – Remplissez le tableau ci-dessous.

Séquence	Aléa RaW	où on a besoin de la valeur	la valeur est encore dans

`op $1,$2,$3` `op $4,$1,$5`	oui	à l'entrée 1 de l'ALU	le registre EXE/MEM

`op $1,$2,$3`
`nop`
`op $4,$1,$5`
	oui	à l'entrée 1 de l'ALU	le registre MEM/WB

`op $1,$2,$3`
`nop nop`
`op $4,$1,$5`			

`op $1,$2,$3`
`nop nop nop`
`op $4,$1,$5`			

`op $1,$2,$3` `op $4,$5,$1`			

`op $1,$2,$3`
`nop`
`op $4,$5,$1`			

`op $1,$2,$3`
`nop nop`
`op $4,$5,$1`			

`op $1,$2,$3`
`nop nop nop`
`op $4,$5,$1`			

`op $1,$2,$3` `opi $4,$1,imm`			

`op $1,$2,$3`
`nop`
`opi $4,$1,imm`			

`op $1,$2,$3`
`nop nop`
`opi $4,$1,imm`			

`op $1,$2,$3`
`nop nop nop`
`opi $4,$1,imm`			

`op $1,$2,$3` `lw $4, imm($1)`			

`op $1,$2,$3`
`nop`
`lw $4, imm($1)`			

`op $1,$2,$3`
`nop nop`
`lw $4, imm($1)`			

`op $1,$2,$3`
`nop nop nop`
`lw $4, imm($1)`			

`lw $2, imm($4)` `op $1,$2,$3`			

`lw $2, imm($4)`
`nop`
`op $1,$2,$3`			

`lw $2, imm($4)`
`nop nop`
`op $1,$2,$3`			

`lw $2, imm($4)`
`nop nop nop`
`op $1,$2,$3`			

`lw $2, imm($4)` `sw $2, imm($3)`			
`lw $2, imm($4)`			

`nop` `sw $2, imm($3)`			

`lw $2, imm($4)`
`nop nop`
`sw $2, imm($3)`			

`lw $2, imm($4)`
`nop nop nop`
`sw $2, imm($3)`			

Ex. 2 : Dépendances de contrôle

Dans cet exercice, nous allons travailler sur les séquences d'instructions ci-dessous.

a	b	c

`label :`
`  Ins1`
`  Ins2`
`  j label`
`  Ins3`
`  Ins4`
`  Ins5`
`  Ins6`
	`label :`
	`  Ins1`
	`  Ins2`
	`  beq $1, $2, Label`
	`  Ins3`
	`  Ins4`
	`  Ins5`
	`  Ins6`
		`label :`
		`  Ins1`
		`  slt $1, $2, $3`
		`  beq $1, $2, Label`
		`  Ins3`
		`  Ins4`
		`  Ins5`
		`  Ins6`

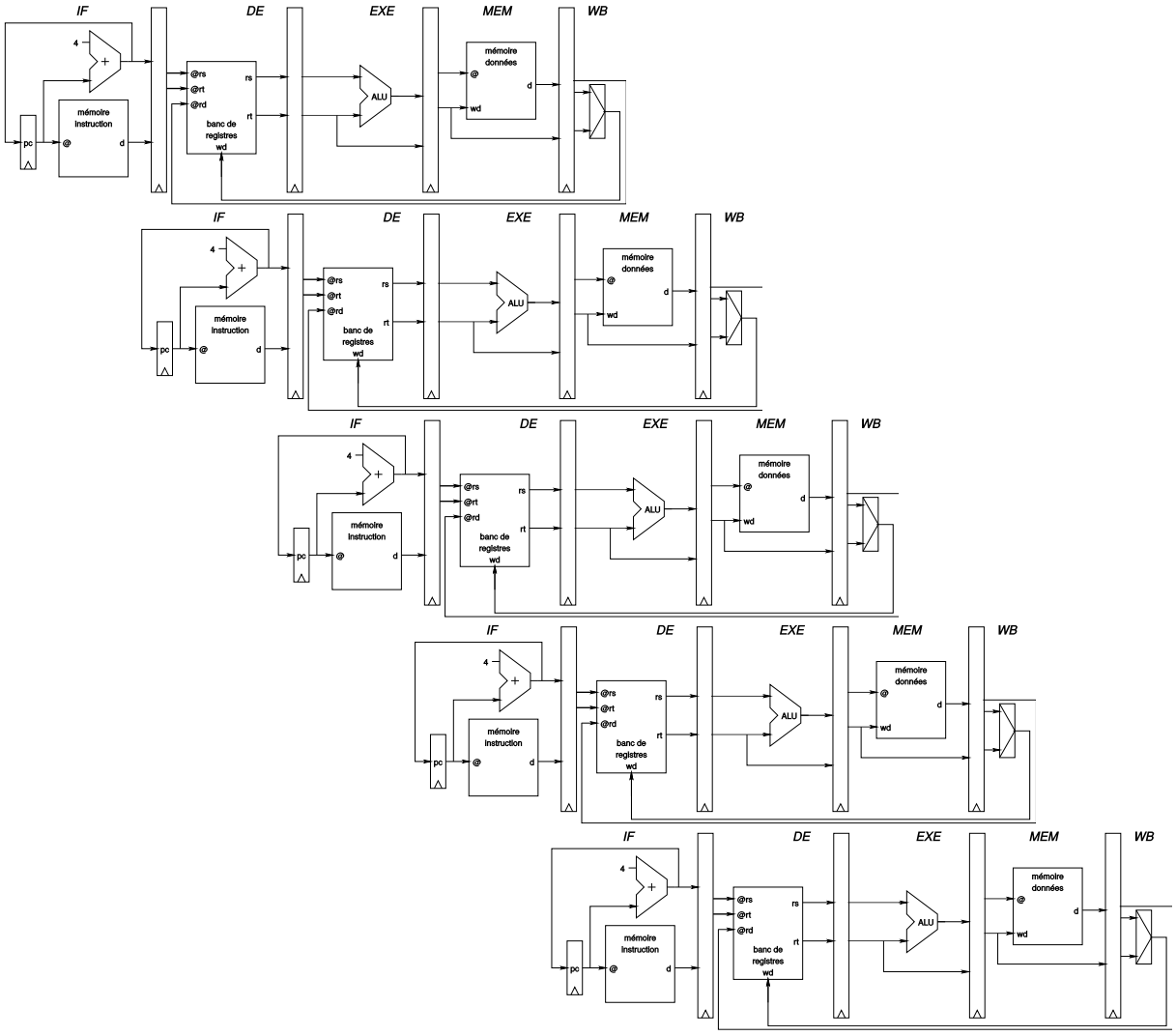
Question 1 – Les aléas de contrôle sont aussi causés par des dépendances RaW. Quel registre est impliqué dans ce cas ?

Question 2 – On suppose que le calcul de l'adresse de branchement et son affectation dans PC se font dans l'étage EXE. Pour l'exemple a, donnez la suite d'instructions normalement exécutée et celle exécutée avec l'hypothèse précédente.

Question 3 – On rajoute des instructions `nop` pour assurer le bon fonctionnement de cet exemple. Quel est le CPI équivalent de l'instruction j ? Quel est le CPI moyen pour l'exécution de la boucle ? Comment pourrait-on améliorer ce chiffre ?

Question 4 – Que faut-il rajouter comme matériel pour gérer l'exemple b ?

Question 5 – Qu'apporte l'exemple c ?



Nmémonic/ syntaxe	For-mat	Opcode/ Func	Opération	Nom Complet
`addiu $rt, $rs, imm`	I	0x9	R[rt]=R[rs]+SignExtImm [1]	addition non signée avec un immédiat
`addu $rd, $rs, $rt`	R	0x0/0x21	R[rd]=R[rs]+R[rt]	addition non signée entre registres
`and $rd, $rs, $rt`	R	0x0/0x24	R[rd]=R[rs] & R[rt]	ET bit à bit entre registres
`andi $rt, $rs, imm`	I	0xC	R[rt]=R[rs] & ZeroExtImm [2]	ET bit à bit avec un immédiat
`beq $rs, $rt, label`	I	0x4	if(R[rs]== R[rt]) PC=PC+4+BranchAddr [3]	Branchement si égalité
`bne $rs, $rt, label`	I	0x5	if(R[rs] != R[rt]) PC=PC+4+BranchAddr [3]	Branchement si inégalité
`j label`	J	0x2	PC = JumpAddr [4]	Saut inconditionnel
`lw $rt, imm($rs)`	I	0x23	R[rt] = mem[R[rs]+SignExtImm [1]]	chargement registre
`slt $rd, $rs, $rt`	R	0x0/0x2A	R[rd] = (R[rs] < R[rt]) ? 1 : 0	Set Less Than registre à registre
`sw $rt, imm($rs)`	I	0x2B	mem[R[rs]+SignExtImm [1]] = R[rt]	stockage registre

1. SignExtImm = $\text{IMM16}_{15}^{16}||\text{IMM16}$
2. ZeroExtImm = $\mathbf{0^{16}}||\text{IMM16}$
3. BranchAddr = $\text{IMM16}_{15}^{14}||\text{IMM16}||\mathbf{0^2}$
4. JumpAddr = $(\text{PC+4})_{31\cdots28}||\text{IMM26}||\mathbf{0^2}$