# Compte rendu Projet 2 - SE201

Sevin Adrien, Gaillard Vincent, Bias Steven October 7, 2016

## 3 - Pipelining

<u>3.1</u>

En observant les instructions, on remarque un data hazard lors de l'execution de l'instruction 3. En effet, le store (instruction 3) nécessite la valeur du registre 29, mais celle-ci n'est pas encore à jour. L'instruction précédent qui écrit dans ce registre n'est pas encore terminée.

Au cycle 17, la valeur de r29 n'est pas à jour mais elle devrait être chargée par le store qui est en ID. Au cycle 18, l'instruction 3 passe directement à l'execution sans avoir attentdu de pouvoir lire la valeur du registre 29. Il y a donc bien forwarding.

📋 cycles and pipelir	ie 💮																				
Instructions/Cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	2
nop	₩B	₩B	WB	₩B	₩B	₩B	₩B	₩B	WB	₩B	₩B	WB	₩B	₩B	₩B	₩B	₩B	₩B	WB	WB	WE
addui r29,r29,-32						IF	IF	IF	IF	IF	IF	ID	EΧ	ЕΧ	EΧ	ЕΧ	ЕΧ	EΧ	MEM	MEM	ME
sw 28(r29),r31												IF	ID	ID	ID	ID	ID	ID	ЕΧ	EX	
sw 24(r29),r30													IF	IF	IF	IF	IF	IF	ID	ID	I
or r30,r29,r0																			IF	IF	IF
lhi 29,3840																					
jal 0x1054																					П
nop																					L

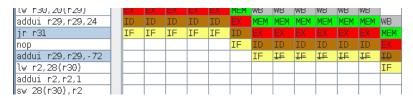
## <u>3.2</u>

## Premier exemple:

slti r2,r2,2												
begz r2,0xffffff50	WE	3	WB	WB	WB							
nop	ME	M	MEM	MEM	MEM	WB						
sw 28(r30),r0	E)		EX	ЕΧ	ЕΧ	MEM	WB					
beqz r0,0x78	I	)	ID	ID	ID	EΧ	MEM	WB				
nop	IF	=	IF	IF	IF	ID	ЕΧ	MEM	WB			
lw r2,28(r30)						IF	ID.	EΧ	M≣M	₩B		
lw r2,24(r30)							IF	ID	ЕΧ	MEM	WB	₩B
addui r3,r2,-l								IF	ID	ID	EΧ	MEM
lw r2,28(r30)									IF	IF	ID	ЕΧ
slt r2,r2,r3											IF	ID
0 0 00000000												7.5

Dans le cas d'une condition (beqz, bnez) à cause d'une "misprediction" le processeur va "flush" les instructions qui avaient été commencé. Celles-ci ne se finiront jamais.

### Deuxième exemple:



Dans le cas d'un jump (jr) les instructions qui suivent vont être "flushed" et ne seront pas finies.

<u>3.3</u>

Notons que chaque jump et condition est suivie d'une instruction nop. Cette instruction n'est pas "flushed".

Cette technique permet de ne jamais exécuter trop longtemps une instruction qui sera stoppée ensuite.

Concrètement, une instruction "flushed" n'atteind jamais l'état ID.

## 4 - Branch Prediction

### 4.1

1-bit saturation counting branch predictor : misprediction rate = 19.14% 2-bit saturation counting branch predictor : misprediction rate = 18.09% Donc le branch predictor sur 2 bits est meilleur que celui sur 1 bit.

On peut aussi voir la meilleure prédiction dans le nombre de "fetches" qui ont due être faits. 2499 contre 2503 pour le 1-bit prediction.

Cela est due au fait que le 2-bit prediction permet de supprimer le bruit en quelque sorte. Effectivement, si dans un programme il y a beaucoup de boucles (ce qui est le cas ici), même si de temps en temps les boucles seront finies et il y aura une misprediction, on peut supposer que la condition est toujours vérifiée.

Dans ce cas, la méthode 1-bit prediction entraine deux mispredictions, contre une seule pour celle de textit2-bit prediction.

#### 4.2

Selon les branches, on remarque quelques différences entre les branch predictors.

Les seules branches où il y a une différences sont :

1-bit saturation counting branch predictor:

bpc: 0x00001104 [4] a:56 t/nt: 10/46 mp/cp: 9/47 mp-ratio: 0.16 bpc: 0x00001158 [88] a:27 t/nt: 21/6 mp/cp: 12/15 mp-ratio: 0.44

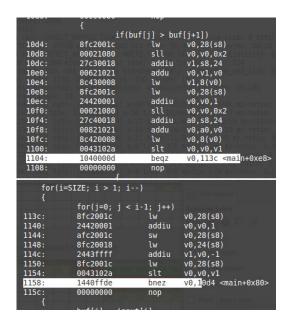
2-bit saturation counting branch predictor:

```
Number of unique jumps: 10
bpc: 0x0000104c [76] tgts: [0x0000113c] a:66 t/nt: 66/0 mp/cp: 5/61 mp-ratio: 0,08
bpc: 0x00001104 [4] tgts: [0x0000113c] a:46 t/nt: 10/36 mp/cp: 10/36 mp-ratio: 0,22
bpc: 0x00001158 [88] tgts: [0x0000104] a:27 t/nt: 21/6 mp/cp: 7/20 mp-ratio: 0,26
bpc: 0x00001134 [52] tgts: [0x00001000] a:16 t/nt: 16/0 mp/cp: 1/15 mp-ratio: 0,06
bpc: 0x000010cc [76] tgts: [0x00001148] a:11 t/nt: 11/0 mp/cp: 6/5 mp-ratio: 0,55
```

On remarque donc que pour la branche [4] le branch predictor sur 1 bit est légèrement meilleur, mais sur la branche [88], le branch predictor sur 2 bit est bien meilleur.

On peut en conclure qu'il est difficile de classer les prédicteurs par performance car elles dépendent du code exécuté.

4.3



La différence à noter est que la branche [4] découle d'une condition (if) alors que la branche [88] découle d'une boucle (for).

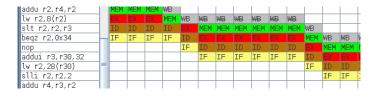
Leur comportement est très différent car la condition d'une boucle for n'est fausse que pendant une itération (généralement), ce qui est bien géré par une 2-bit prediction, alors qu'une condition (if) ne suit pas cette règle.

Ici, la condition fonctionne plutôt par "paquets" de conditions vérifiés/fausses. Le 2-bit prediction prend bien en charge les boucles for mais entraine régulièrement deux misprédictions.

Le 1-bit prediction est moins intéressante pour les for mais gère mieux les conditions comme décrit juste au dessus.

### 4.4

Dans le cas où la boucle ou condition de nécessite pas de jump, l'instruction suivante provoque un point (un nop est inséré) de pénalité alors qu'il n'y en a pas dans le cours.



Aussi, si un jump est effectué l'une des deux instructions qui est flushed génère plus de pénalités. C'est une autre différence par rapport au cours.

slt1 r2,r2,7	IF	IF	IF	ΙD	ID	ЕΧ	MEM	WB			
bnez r2,0xffffffbc				IF	IF	ID	ΕX	MEM	WB	WB	W
nop						IF	ID	EX	MEM	MEM	М
lhi 2,1							IF	ID.	ΕX	EΧ	E
addui r2,r0,7								IF	ID	ID	Ι
out 04/5001 50									TE	TE	Т

	_			_			_	_				_	_	_	_	_	_
slt r2,r2,r3		ID	ID	ID	ID	EX	MEM	WB	WB	WB	WB	WB	WB				
begz r2,0x34		IF	IF	IF	IF	ID	EX	MEM	MEM	MEM	MEM	MEM	MEM	₩B			
nop						IF	ID	EΧ	ΕX					MEM	WB	WB	WB
lw r2,28(r30)							IF	ID.	ID.	ID	ID	ID.	ID	EΧ	MEM	MEM	MEM
addui r3,r30,32								IF	IF	IF	IF	IF	IF	ID	EΧ		ΕX
lw r2,28(r30)														IF	ID	ID	ID
slli r2,r2,2															IF	IF	IF
- 11 1 - 0 - 0																	

## 5 - Data Caches

<u>5.1</u>

Nombre de ligne dans le cache : 4 Taille d'un ligne dans le cache : 4 Taille totale du cache : 16 octets Accesses : 488 hits : 364 misses : 124 hit rate : 74,59loaded words : 124

<u>5.2</u>

Nombre de ligne dans le cache : 8 Taille d'un ligne dans le cache : 4 Taille totale du cache : 32 octets Accesses : 613 hits : 557 misses : 56 hit rate : 90,86Nombre de ligne dans le cache : 16 Taille d'un ligne dans le cache : 4 Taille totale du cache : 64 octets Accesses : 688 hits : 673 misses : 15 hit rate : 97,82Nombre de ligne dans le cache : 32 Taille d'un ligne dans le cache : 4 Taille totale du cache : 128 octets Accesses : 728 hits : 721 misses : 7 hit rate : 99,04Nombre de ligne dans le cache : 64 Taille d'un ligne dans le cache : 4 Taille totale du cache : 256 octets Accesses : 728 hits : 721 misses : 7 hit rate : 99,04La taille du cache est égale à le taille d'une ligne multipliée par le nombre de ligne. Plus le nombre de ligne augmante, donc la taille du cache augmente, plus le hit rate augmente. Les configurations avec 32 lignes et 64 lignes offrent le meilleurs hit rate. Mais on peut dire que la meilleure configuration est celle avec 32 lignes car la taille du cache est plus petite que celle avec 64 lignes.