ÉCOLE POLYTECHNIQUE MONTRÉAL

Solution des exercices pour la préparation de l'examen final : bloc processeur
INF3610 Systèmes Embarqués

Chapitre 3

QUESTION #1 Architecture RISC et aléas

a)

Instructions pouvant être pipelinées	**Signification**	**Nombre de cycles dans EX**	**Cycle du pipeline où l'opération termine (le résultat étant disponible 1 cycle plus tard)**
LW R1, 10(R2)	R1 ← Mem[10 + R2]	1	ER (pas d'accumulateur)
ADDI R1, R1, #1	R1 ← R1 + 1	1	ER "
SW R1, 10(R2)	Mem[10 + R2] ← R1	1	MEM
SUB R4, R3, R2	R4 ← R3 – R2	1	ER "
BNZ R3, etiq	Branch. si non zéro	1	EX

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LW	R1, 10(R2)	LI	DI	EX	MEM	ER											
ADDI	R1, R1, #1		LI	DI	SU	SU	EX	MEM	ER								
SW	R1, 10(R2)			LI	SU	SU	DI	EX	SU	MEM							
ADDI	R2, R2, #4						LI	DI	SU	EX	MEM	ER					
SUB	R4, R3, R2							LI	SU	DI	SU	SU	EX	MEM	ER		
LW	R5, 10(R6)									LI	SU	SU	DI	EX	MEM	ER	
BNZ	R4, etiq												LI	DI	SU	EX	
														LI**	SU	SU	LI*

*Si le test donne pas 0 on aura LI (de LW R1, 10(R2) sinon DI (de l'instruction suivante à BNZ i.e. CP +1)

** On aurait pu aussi mettre un SU, mais la réalité c'est qu'on lit la prochaine instruction (CP+1) et dès qu'on voit qu'on a une instruction de branchement (fin du cycle 13), on arrête (donc équivalent à un SU).

Nombre total d'instructions requises est de 15 cycles et on débute la prochaine itération au 16e cycle.

b)

Instructions pouvant être pipelinées	**Signification**	**Cycle du pipeline où l'opération termine et entre parenthèse cycle ou la donnée est disponible grâce aux chemins d'envoi et d'un accumulateur.**
LW R1, 10(R2)	R1 ← Mem[10 + R2]	ER (MEM)
ADDI R1, R1, #1	R1 ← R1 + 1	ER (EX)
SW R1, 10(R2)	Mem[10 + R2] ← R1	MEM
SUB R4, R3, R2	R4 ← R3 – R2	ER (EX)
BNZ R3, etiq	Branch. si non zéro	EX

		1	2	3	4	5	6	7	8	9	10	11	12	
LW	R1, 10(R2)	LI	DI	EX	MEM	ER								
ADDI	R1, R1, #1		LI	DI	SU	EX	MEM	ER						
SW	R1, 10(R2)			LI	SU	DI	EX	MEM						
ADDI	R2, R2, #4					LI	DI	EX	MEM	ER				
SUB	R4, R3, R2						LI	DI	EX	MEM	ER			
LW	R5, 10(R6)							LI	DI	EX	MEM	ER		
BNZ	R4, etiq								LI	DI	EX			
										LI*		LI**		

*LI de l'instruction du branchement pris (LW R1, 10(R2))
**LI de l'instruction suivante à BNZ (CP + 1)

Le nombre total d'instructions requises est de 11 cycles mais on peut débuter la prochaine itération (en espérant que l'anticipation est bonne) au 9^{e} cycle (en ce sens on peut 8 cycles par itération).

Si on se trompe (i.e. on branche pas), le problème n'est pas temps le 2 cycles perdus mais plutôt qu'il faut tout remettre le système dans l'état du cycle 9. Or imaginez qu'il y est eu interruption au cours du cycle 10…

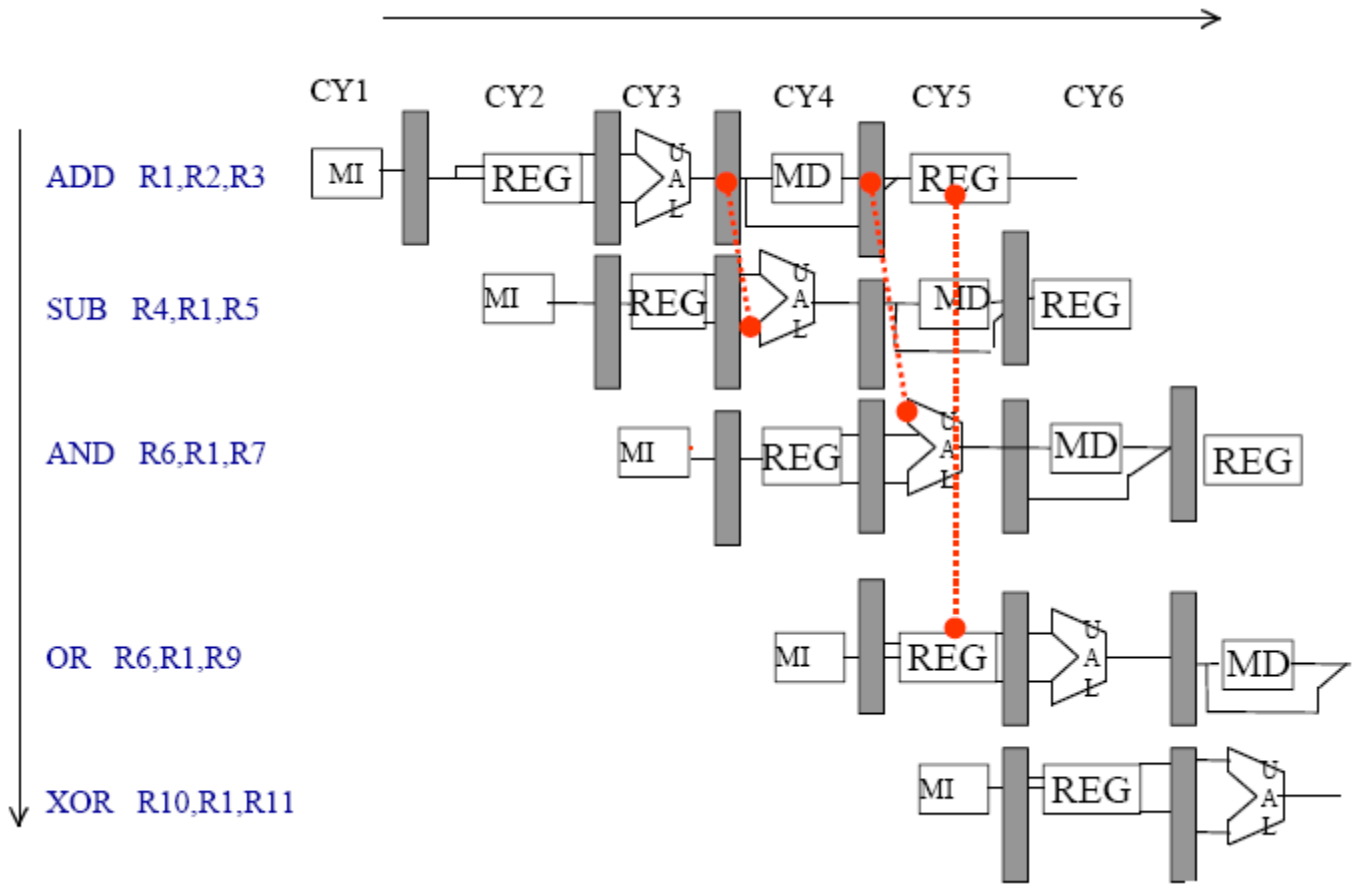


Figure 1.1 : Exemples de chemin d'envoie pour éviter les aléas

QUESTION #2 Architecture RISC, aléas et superscalaire

a)

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
LD	F10, 0(R1)	LI	DI	EX	ME	ER																			
LD	F20, 0(R2)		LI	DI	EX	ME	ER																		
SUBF	F30, F10, F20			LI	DI	SU	E1	E2	E3	E4	ME	ER													
LD	F40, 0(R3)				LI	SU	DI	EX	MEM	ER															
ADDF	F50, F40, F30						LI	DI	SU	SU	E1	E2	E3	E4	ME	ER									
SD	0(R3), F50							LI	SU	SU	DI	EX	SU	SU	SU	ME	ER								
ADDI	R1, R1, #8										LI	DI	SU	SU	SU	EX	ME	ER							
ADDI	R2, R2, #8											LI	SU	SU	SU	DI	EX	ME	ER						
ADDI	R3, R3, #8															LI	DI	EX	ME	ER					
SGTI	R5, R1, R4																LI	DI	EX	ME	ER				
BEQZ	R5, etiq																	LI	DI	EX					
																			SU	SU	LI				

Le nombre total de cycles est de 20 et on peut recommencer une nouvelle itération au 20e cycle, donc on peut dire 19 cycles par itération 16*19= 304 cycles[1].

[1] Évidemment on fait ici l'hypothèse qu'il n'y a pas de dépendance entre 2 itérations de boucle. À titre d'exemple d'une telle dépendance: F10 est utilisé dans une longue division (p.e. après ADDI R3, R3, #8) juste avant le branchement et on fait un LD sur F10 au début de l'itération suivante. Dans un tel cas il faudrait attendre que la division soit terminée. Ça repousserait à plus que 9 cycles…

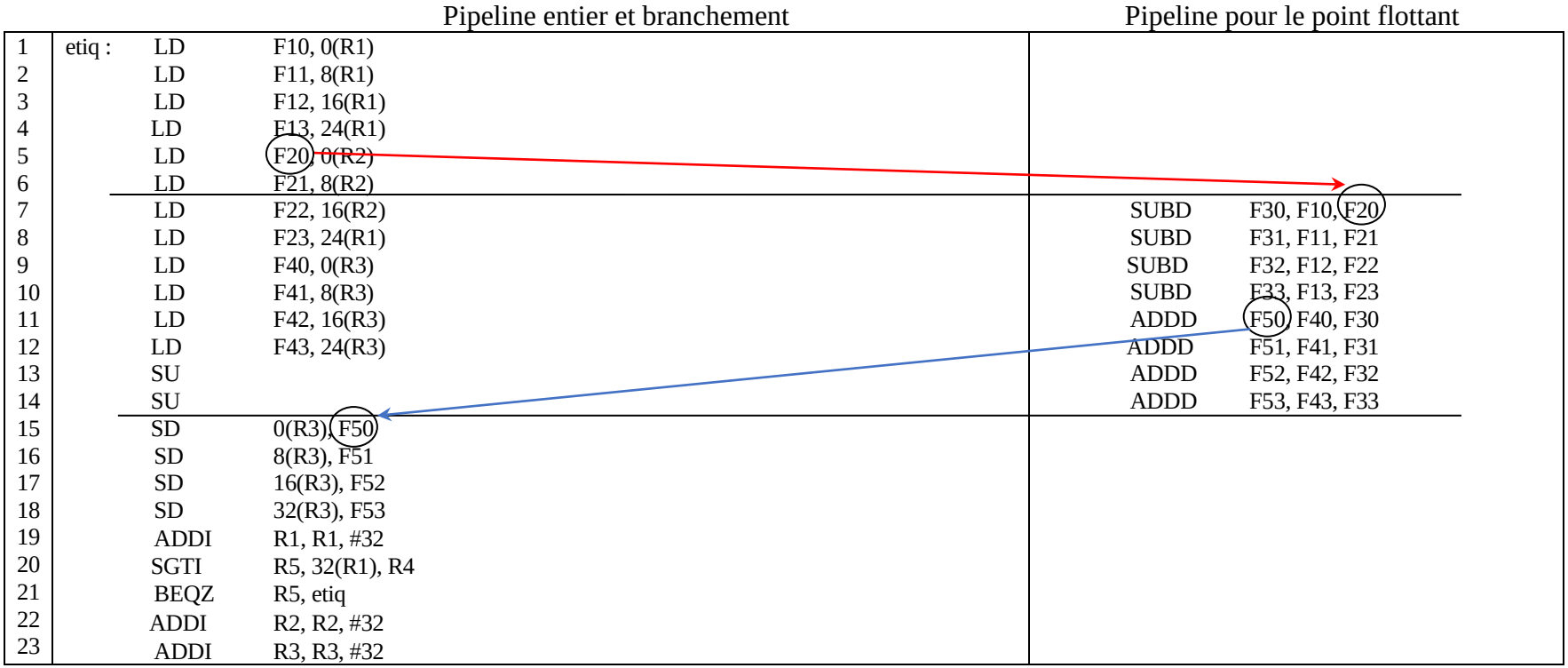
b)

```
etiq :  LD      F10, 0(R1)
        LD      F11, 8(R1)
        LD      F12, 16(R1)
        LD      F13, 24(R1)
        LD      F20, 0(R2)
        LD      F21, 8(R2)
        LD      F23, 16(R2)
        LD      F24, 24(R2)
        LD      F40, 0(R3)   //on serait arriver au même résultat si
        LD      F41, 8(R3)   // les loads de F40 à F43 étaient restés
        LD      F42, 16(R3)  // entre SUBD et ADDD
        LD      F43, 24(R3)
        SUBD    F30, F10, F20
        SUBD    F31, F11, F21
        SUBD    F32, F12, F22
        SUBD    F33, F13, F23
        ADDD    F50, F40, F30
        ADDD    F51, F41, F31
        ADDD    F52, F42, F32
        ADDD    F53, F43, F33
        SD      0(R3), F50
        SD      8(R3), F51
        SD      16(R3), F52
        SD      24(R3), F53
        ADDI    R1, R1, #32
        SGTI    R5, 0(R1), R4
        BEQZ    R5, etiq
        ADDI    R2, R2, #32
        ADDI    R3, R3, #32
```

1 itération = 29 cycles
4 itérations = 116 cycles donc 304/116 = 2,62 d'accélération

c) On suppose un superscalaire à avec 2 étages donc on peut supposer être capable de lire 2 instructions par cycle et produire 2 instructions par cycles (quand le pipeline est plein). Attention! Il peut y avoir alors des dépendances **intra pipeline** (i.e. à l'intérieur d'un même pipeline comme pour le scalaire) ou **entre pipelines** comme le montre les 2 flèches ci-après :

- la flèche rouge représente la dépendance entre les 2 pipelines de 1 cycle à respecter pour F20 entre LD et SUBD (i.e. le SU de la colonne 5 en a). Ici on va simplement débuter le calcul sur le pipeline point flottant au cycle 7 (et non au cycle 6). Du coté du pipeline entier, on va simplement poursuivre les loads (pas de suspension).
- la flèche bleu représente la dépendance entre pipeline de 3 cycles pour F50 entre ADDD et SD. Notez ici que ADDD prends 4 cycles i.e. E1 à E4 (et non 3 cycles comme les slides du cours). Or ADDD F51, F41, F31, ADDD F52, F42, F32 et ADDD F53, F43, F33 comblent cette suspension du côté de l'unité flottante, mais on ne peut repartir le pipeline entier qu'au cycle 15. Il faut donc ajouter 2 suspensions dans le pipeline entier (lignes 13 et 14).



1 itération = 23 cycles
4 itérations = 92 cycles donc 304/92 = 3.30 d'accélération

Autres commentaires :

- Si on regarde le tableau de la page précédente, il est normal que le pipeline flottant (ou parfois le pipeline entier) soit vide par moment. Car même si on peut ici lire 2 instructions par cycle, si durant une certaine séquence de code on a que des instructions entières (ou inversement flottantes) seulement un des 2 pipelines sera alors occupé.
- Comme j'ai dit au cours, dans la vraie vie même si on peut avoir des modèles M5 avec plusieurs unités de calculs de chargement, de calculs entiers et calculs flottants (p.e. 8 sur le Cortex A72 que vous retrouvez aujourd'hui sur un Raspberry Pi 4[2]) ça ne veut pas dire pour autant que vous pourrez lire et décoder 8 instructions en parallèle (il y a une limite technologique et économique qui est simplement qu'une mémoire à 8 ports parallèles va couter très cher!). Typiquement (comme le A-72) on peut lire et décoder 4 instructions de 32 bits en parallèle (ou 2 instructions de 64 bits ou encore 1 instruction de 128 bits). Réf. voir slide 40 du bloc processeur.

[2] À moins de 100$ avec en prime un GPU

QUESTION #3 Pipeline et parallélisme d'instructions

a)

##		Instruction/Cycl	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	L:	LD F10, 0(R1)	LI	DI	EX	ME	ER																					
2		MULTF F20,F10,F0		LI	DI	SU	E1	E2	E3	ME	ER																	
3		LD F30,0(R2)			LI	SU	DI	EX	ME	ER																		
4		DIVF F40,F30,F1					LI	DI	SU	E1	E2	E3	E4	ME	ER													
5		ADDF F40, F40, F20						LI	SU	DI	SU	SU	SU	E1	E2	ME	ER											
6		SD 0(R2),F40								LI	SU	SU	SU	DI	EX	SU	ME											
7		ADDI R1, R1, 8												LI	DI	SU	EX	ME	ER									
8		ADDI R2, R2, 8													LI	SU	DI	EX	ME	ER								
9		STGI R3,R1,fait															LI	DI	EX									
10		BEQZ R3, L																LI	DI	EX								
																			SU	SU	LI							

Tableau 2.2

Au total 8 suspensions. On débute une nouvelle itération au cycle 19. Donc, le nombre de cycles par itération est de 18.

b) Il y a très peu de gain à dérouler avec M3 :

```
L:          LD F10, 0(R1)
            LD F11, 8(R1)
            LD F12, 16(R1)
            LD F13, 32(R1)
            MULTF    F20, F10, F0
            Suspension
            suspension
            suspension
            MULTF    F21, F11, F0
            suspension
            suspension
            suspension
            MULTF    F22, F12, F0
            Etc.
```

N.B. Les supensions du haut ne sont pas dues à des dépendances mais simplement parce que le multiplieur n'est pas pipeliner (on dit que le multiplieur est multicycles). On doit donc attendre 3 cycles avant de commencer une autre opération de multiplication.

Par contre, avec un modèle comme M4 on aurait :

```
L:          LD F10, 0(R1)
            LD F11, 8(R1)
            LD F12, 16(R1)
            LD F13, 32(R1)
            MULTF    F20, F10, F0
            MULTF    F21, F11, F0
            MULTF    F22, F12, F0
            MULTF    F23, F13, F0
            LD F30, 0(R2)
            LD F31, 8(R2)
            LD F32, 16(R2)
            LD F33, 24(R2)
            DIVF F40, F30, F1
            DIVF F41, F31, F1
            DIVF F42, F32, F1
            DIVF F43, F33, F1
            ADDF F40, F40, F20
            ADDF F41, F41, F21
            ADDF F42, F42, F22
            ADDF F43, F43, F23
            SD 0(R2), F40,
            SD 8(R2), F41,
            SD 16(R2), F42,
            SD 32(R2), F43,
            STGI R3,R1,fait
            BEQZ R3, L
            ADDI R2, R2, 32
            ADDI R1, R1, 32
```

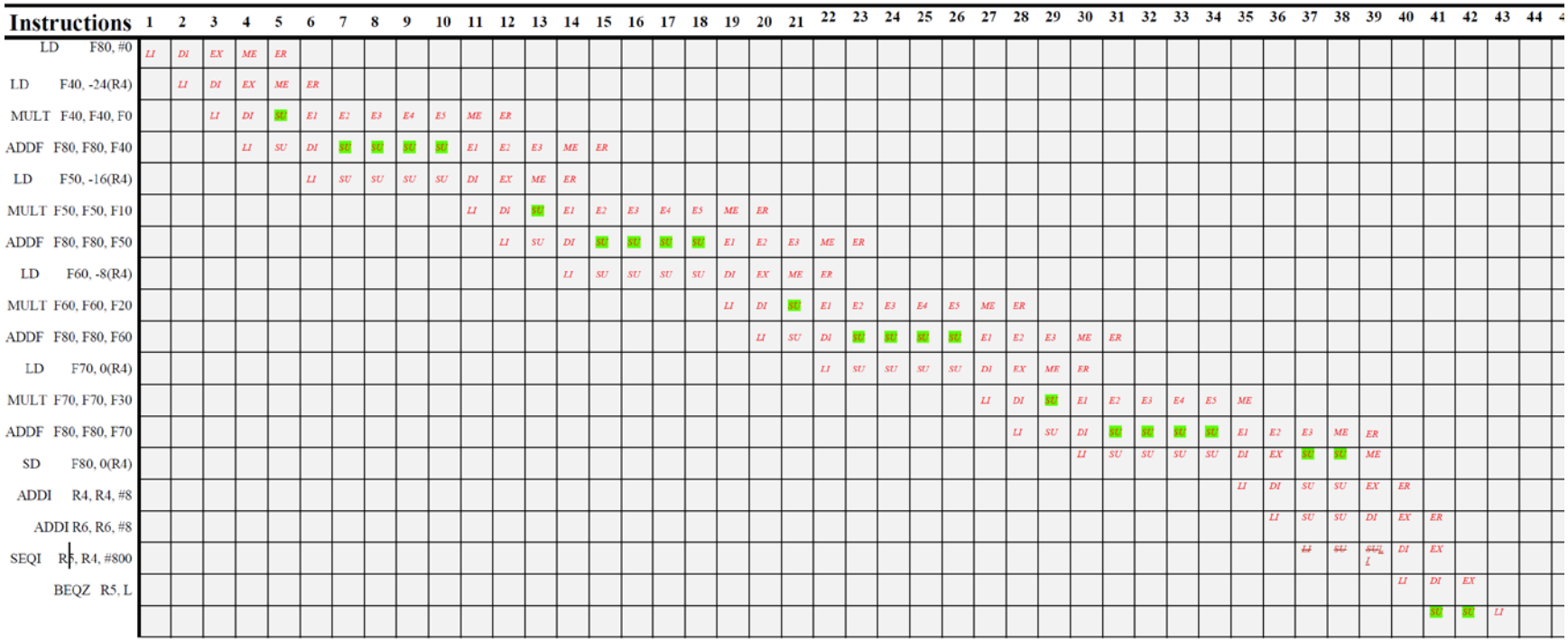
N.B. Contrairement à M3 on peut ici on peut débuter une 2e opération de (même chose pour ADDF, SUBF et DIVF) au cycle suivant en autant bien sur qu'il n'y est pas de dépendance entre 2 multiplications consécutives. On dit alors que

la multiplication est pipelinable (et donc II=1 en régime permanent) plutôt que multicycles…

*En a) on a 18 cycles * 100 = 1800 cycles au total*
*En b) on a 28 cycles * 25 = 700 cycles au total*
Donc, en déroulant la boucle on obtient une accélération de 1800/700 ≅ 2.57

QUESTION #4 Architectures RISC de base et ordonnacement

a)



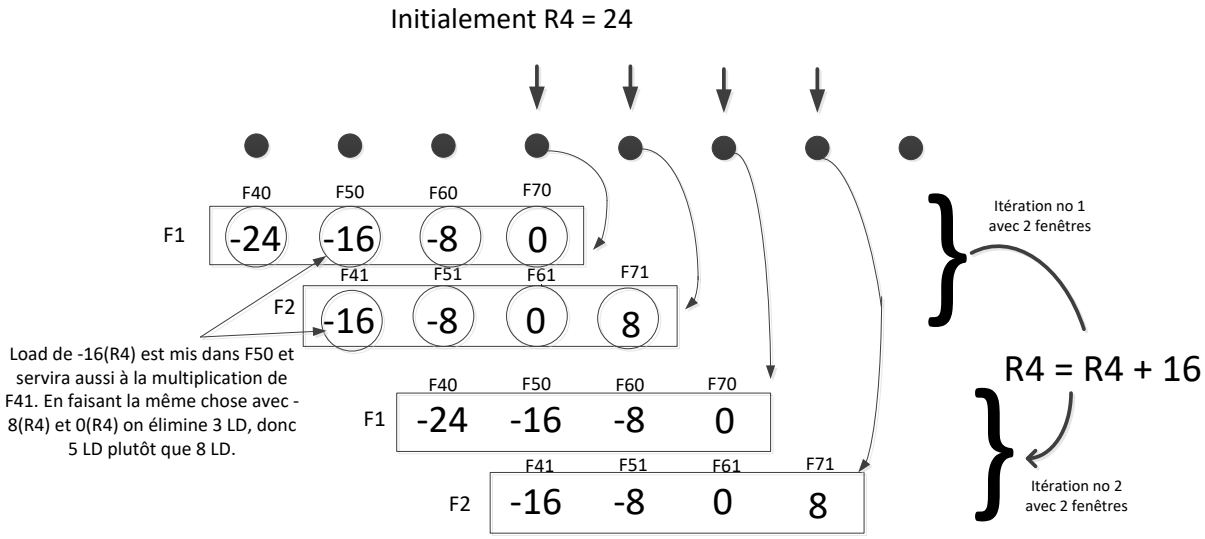
Instructions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
LD F80, #0	LI	DI	EX	ME	ER																																							
LD F40, -24(R4)		LI	DI	EX	ME	ER																																					
MULT F40, F40, F0			LI	DI	ST	E1	E2	E3	E4	E5	ME	ER																														
ADDF F80, F80, F40				LI	SU	DI	ST	ST	ST	ST	E1	E2	E3	ME	ER																												
LD F50, -16(R4)						LI	SU	SU	SU	SU	DI	EX	ME	ER																														
MULT F50, F50, F10											LI	DI	ST	E1	E2	E3	E4	E5	ME	ER																						
ADDF F80, F80, F50												LI	SU	DI	ST	ST	ST	ST	E1	E2	E3	ME	ER																				
LD F60, -8(R4)														LI	SU	SU	SU	SU	DI	EX	ME	ER																				
MULT F60, F60, F20																			LI	DI	ST	E1	E2	E3	E4	E5	ME	ER														
ADDF F80, F80, F60																				LI	SU	DI	ST	ST	ST	ST	E1	E2	E3	ME	ER											
LD F70, 0(R4)																						LI	SU	SU	SU	SU	DI	EX	ME	ER												
MULT F70, F70, F30																											LI	DI	ST	E1	E2	E3	E4	E5	ME							
ADDF F80, F80, F70																												LI	SU	DI	ST	ST	ST	ST	E1	E2	E3	ME	ER			
SD F80, 0(R4)																														LI	SU	SU	SU	SU	DI	EX	ST	ST	ME			
ADDI R4, R4, #8																																			LI	DI	SU	SU	EX	ER			
ADDI R6, R6, #8																																				LI	SU	SU	DI	EX	ER		
SEQI R5, R4, #800																																					LI	SU	STL L	DI	EX			
BEQZ R5, L																																							LI	DI	EX		
																																								ST	ST	LI	

Tableau 4.2

*On a au total 24 suspensions. On a 18 instructions. On voit bien que la prochaine itération débutera à 43 cycles. Donc au total 42 * 96 = 4032 cycles*

b)

Ici l'idée est de travailler sur deux fenêtres (F1 et F2) durant la même itération. Toutefois contrairement à l'opération vectorielle vue en classe ici il faudra maintenir 2 accumulateurs. Voici le schéma. À la première itération on fait des multiplications/accumulation sur 2 fenêtres en parallèle (fenêtre 1 et fenêtre 2), puis à la prochaine itération on fera fenêtre 2 et fenêtre 3, etc. Vous me remarquerez que ici seulement 5 loads sont requis.



P.S. Dans le cas de l'exemple en classe, je vous ai mentionné que dans le cas du produit scalaire, on pouvait imaginer un compilateur faire les optimisations (dérouler les boucles). Ici, ça devient un peu plus compliqué, mais encore un grand nombre programmeur DSP va se rendre à ce niveau d'optimisation.

Cycle	Pipeline 1
1 L :	LD F80, #0
2	LD F81, #0
3	LD F40, -24(R4)
4	LD F50, -16(R4)
5	LD F60, -8(R4)
6	LD F70, 0(R4)
7	LD F71, 8(R4)
8	MULT F40, F40, F0
9	MULT F41, **F50**, F0
10	MULT F50, F50, F10
11	MULT F51, **F60**, F10
12	MULT F60, F60, F20
13	MULT F61, **F70**, F20
14	MULT F70, F70, F30
15	MULT F71, F71, F30
16	ADDF F80, F80, F40
17	ADDF F81, F81, F41
18	suspension
19	ADDF F80, F80, F50
20	ADDF F81, F81, F51
21	suspension
22	ADDF F80, F80, F60
23	ADDF F81, F81, F61
24	suspension
25	ADDF F80, F80, F70
26	ADDF F81, F81, F71
27	ADDI R4, R4, #16
28	ADDI R6, R6, #16
29	SEQI R5, -8(R4), #800
30	BEQZ R5, L
31	SD -16(R6), F80
32	SD -8(R6), F81

Tableau 4.3

- On obtient 32 cycles * 50 = 1600 cycles
- Ici, le gain aurait-il été meilleur si on avait fait 3 fenêtres par itération… Pourquoi? Expliquez.
- Autre question : pourrait-on sauver davantage en éliminant des loads? Hint : peut-on imaginer un load par itération? Est-ce vraiment un gain si on a une cache?

c)

Cycle	Pipeline 1	Pipeline 2
1 L :	*LD F40, -24(R4)*	
2	*LD F50, -16(R4)*	
3	*LD F60, -8(R4)*	*MULT F40, F40, F0*
4	*LD F70, 0(R4)*	*MULT F41, **F50**, F0*
5	*LD F71, 8(R4)*	*MULT F50, F50, F10*
6	*LD F80, #0*	*MULT F51, **F60**, F10*
7	*LD F81, #0*	*MULT F60, F60, F20*
8		*MULT F61, **F70**, F20*
9		*MULT F70, F70, F30*
10		*MULT F71, F71, F30*
11		*ADDF F80, F80, F40*
12		*ADDF F81, F81, F41*
13		*suspension*
14		*ADDF F80, F80, F50*
15		*ADDF F81, F81, F51*
16		*suspension*
17		*ADDF F80, F80, F60*
18		*ADDF F81, F81, F61*
19	*ADDI R4, R4, #16*	*suspension*
20	*ADDI R6, R6, #16*	*ADDF F80, F80, F70*
21	*SEQI R5, 8(R4), #800*	*ADDF F81, F81, F71*
22	*BEQZ R5, L*	
23	*SD -16(R6), F80*	
24	*SD -8(R6), F81*	

Tableau 4.4

On obtient 24 cycles * 50 = 1200 cycles

d)

	UNITÉ TRANSFERT 1	UNITÉ TRANSFERT 2	UNITÉ FLOTTANTE 1	UNITÉ FLOTTANTE 2	UNITÉ ENTIÈRE
1L:	*LD F41, +8(R4)*	*LD F40, 0(R4)*			
2	*LD F50, -8(R4)*	*LD F60, -16(R4)*	*MULT F40, F40, F0*		
3	*LD F70, -24(R4)*		*MULT F50, F50, F10*	*MULT F41, **F50**, F0*	
4	*LD F80, #0*	*LD F81, #0*	*MULT F60, F60, F20*	*MULT F51, **F60**, F10*	
5			*MULT F70, F70, F30*	*MULT F61, **F70**, F20*	
6			*suspension*	*MULT F71, F71, F30*	
7			*ADDF F81, F81, F41*	*suspension*	
8			*suspension*	*ADDF F80, F80, F40*	
9			*suspension*	*suspension*	
10			*ADDF F81, F81, F51*	*suspension*	
11			*suspension*	*ADDF F80, F80, F50*	
12			*suspension*	*suspension*	
13			*ADDF F81, F81, F61*	*suspension*	
14			*suspension*	*ADDF F80, F80, F60*	
15			*suspension*	*suspension*	*ADDI R4, R4, #16*
16			*ADDF F81, F81, F71*	*suspension*	*ADDI R6, R6, #16*
17				*ADDF F80, F80, F70*	*SEQI R5, 8(R4), #800*
18					*BEQZ R5, L*
19	*SD -16(R6), F80*	*SD -8(R6), F81*			*suspension*
20					*suspension*

Tableau 4.4

On obtient 20 cycles * 50 = 1000 cycles

QUESTION #5 FIR et architecture VLIW

Soit l'application du filtre 1D FIR N TAP :

$y(n) = a0*x(n) + a1*x(n-1) + a2*x(n-2) + ... + aN-1*x(n-N-1)$

où on a[3]

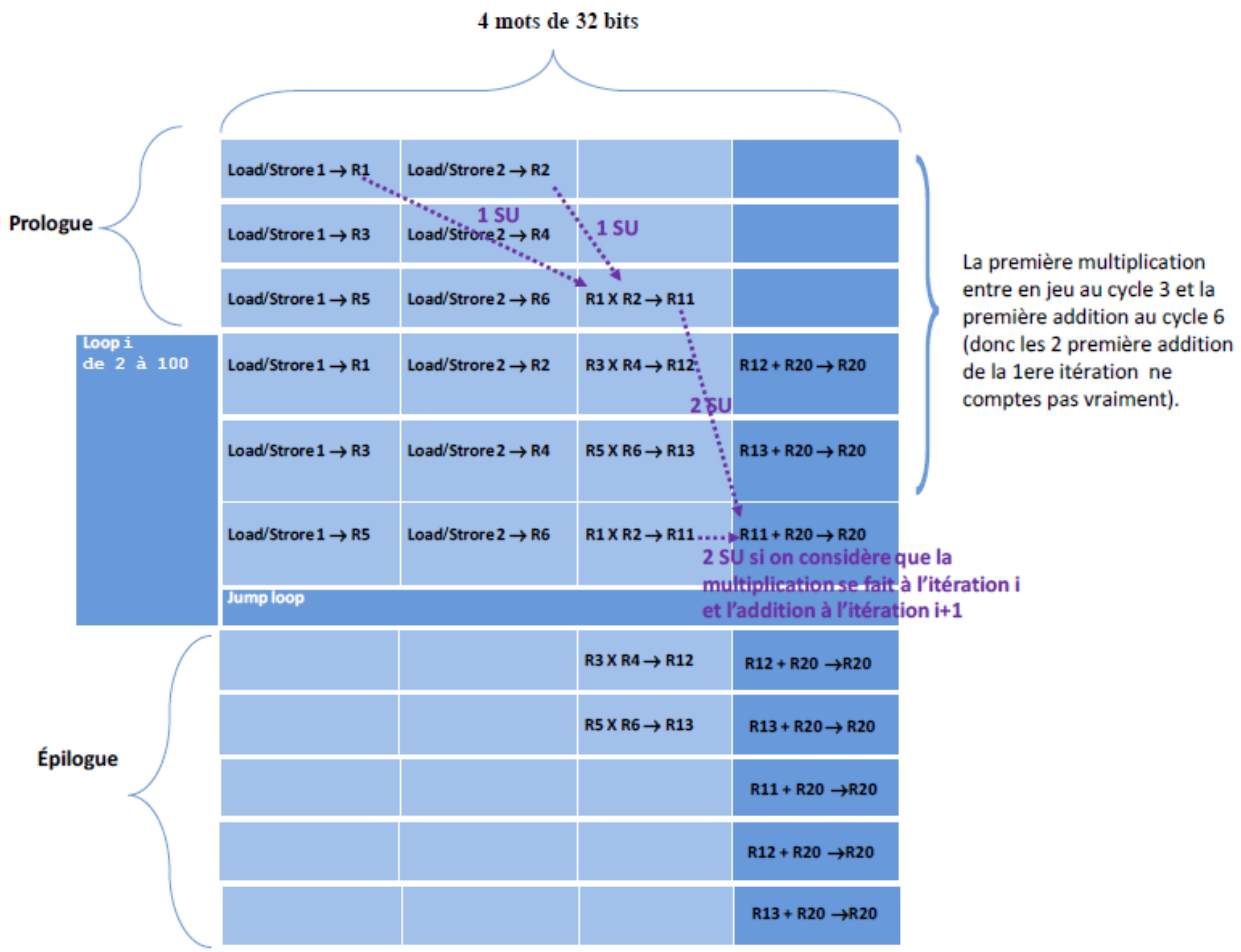
- une table de N coefficients, a0 à aN-1
- une table de N échantillons, allant de l'échantillon courant x(n) à l'échantillon le plus ancien x(n-N-1)
- pour chaque échantillon résultat y(n) courant, la nécessitée d'effectuer N opérations MAC.
- une structure itérative, l'échantillon d'entrée x(n) courant devenant l'échantillon précédant à chaque calcul du nouvel échantillon de sortie y(n)

Soit d'autre part une machine VLIW composé de 2 unités pour lecture/écriture (1 cycle pour EX), 1 unité de multiplication (3 cycles pour EX) et 1 unité d'addition/soustraction (1 cycle pour EX). Expliquez ce que représente le code de la Figure 5.1 et la technique de programmation employée. Indiquez aussi l'endroit où se trouvent le prologue et l'épilogue ainsi que leurs raisons d'être.

Finalement, on suppose les latences d'exécution suivantes :

1) 1 cycle pour le load
2) 3 cycles pour la mult (flottante)
3) 3 cycles pour le add (flottant)

[3] Pour plus d'information référez à http://b.l.free.fr/Page7.html


4 mots de 32 bits
Prologue
Load/Strore 1 → R1
Load/Strore 2 → R2
Load/Strore 1 → R3
Load/Strore 2 → R4
Load/Strore 1 → R5
Load/Strore 2 → R6
R1 X R2 → R11
1 SU
1 SU
Loop i
de 2 à 100
Load/Strore 1 → R1
Load/Strore 2 → R2
R3 X R4 → R12
R12 + R20 → R20
2 SU
Load/Strore 1 → R3
Load/Strore 2 → R4
R5 X R6 → R13
R13 + R20 → R20
Load/Strore 1 → R5
Load/Strore 2 → R6
R1 X R2 → R11
R11 + R20 → R20
2 SU si on considère que la multiplication se fait à l'itération i et l'addition à l'itération i+1
Jump loop
Épilogue
R3 X R4 → R12
R12 + R20 →R20
R5 X R6 → R13
R13 + R20 → R20
R11 + R20 →R20
R12 + R20 →R20
R13 + R20 →R20
La première multiplication entre en jeu au cycle 3 et la première addition au cycle 6 (donc les 2 première addition de la 1ere itération ne comptes pas vraiment).