Réponse aux questions Processeur pipeliné Partie 2

Départements : TIN

Unité d'enseignement ARO

Auteurs : Timothée Van Hove
Benoit Delay

Professeur : Romuald Mosqueron
Assistant : Mike Meury

Classe : A
Salle de labo : A07

Date : lundi, 30 mai 2022

1 Aléas de données

1.1 Circuit data_hazard

1.1.1 Questions

1. Comment savoir si une instruction est dépendante d'une instruction qui est pour le moment dans le stage EXECUTE ? dans le stage MEMORY_ACCESS ? Dans le stage WRITE_BACK ?

Il suffit de mémoriser pour les 3 derniers cycles l'adresse du registre de destination. Si pour l'instruction en cours, l'adresse des registres n, m ou mem en lecture est le même que l'un des registres de destination mémorisé, on sait qu'il y aura un aléa de type RAW.

2. Est-ce que ça pose un problème si une instruction dépend du résultat d'une instruction qui est au stage WRITE_BACK ?

Techniquement, sur Logisim ça fonctionne, le WRITE BACK et la lecture du registre sont faites de manière instantanée, mais en réalité, il faudrait ajouter un coup de clock, à cause du délai de propagation.

3. Quelles informations doivent être mémorisées pour chaque instruction ?
    a. L'adresse du registre de destination
    b. L'information de si pour cette instruction on veut écrire dans la banque de registre
    c. L'information de si pour cette instruction on veut écrire dans la mémoire de données
4. Quelles informations permettent de savoir si le registre D est utilisé ?

Il suffit de savoir si une écriture est demandée dans la banque de registres, donc on doit connaître le signal bank_wr_s qui vient du reg_bank_cotrol_unit dans le main_ctrl_unit dans le bloc decode.

1.2 Commande des signaux dans main_control_unit

1.2.1 Questions

1. Quelles informations permettent de savoir si le registre N, M ou mem sont utilisés ?
    a. Pour N : Si le bloc Execute a sélectionné l'opérande 1, on sait que la lecture se fait sur le registre N
    b. De même, si le bus de sélection de l'opérande 2 est à 0, on sait qu'on va lire le registre M
    c. Finalement, si on sait que l'instruction est une instruction de type STR, STRH ou STRB (signal str_data du bloc memory_access_control_unit dans le bloc decode), on sait que le registre MEM sera lu

1.2.2 Questions Commande des signaux dans hazard_detection

1. Quelles informations permettent de savoir si le registre D est utilisé ?

Il suffit de savoir si une écriture est demandée dans la banque de registres, donc on doit connaître le signal bank_wr_s qui vient du reg_bank_cotrol_unit dans le main_ctrl_unit dans le bloc decode.

2. Une détection d'aléa de donnée va influencer quel(s) enable(s) ? A quel moment ? Pourquoi ?

Lors de la détection d'un aléa de donnée, et si aucun aléa de contrôle n'est détecté, chaque enable des blocs sera désactivé en cascade : DECODE => EXECUTE => M_ACCESS => WRITE_BACK (REG_BANK). La désactivation en cascade permet de terminer de traiter l'instruction en cours. Cependant, ce n'est pas un comportement normal pour un pipeline en forwarding, car les aléas ne sont pas sensés désactiver les blocs => les données devraient être mémorisées dans le bloc execute pour pouvoir être réutilisées directement lors de la prochaine instruction nécessitant une lecture après écriture (RAW)

1.3 Test aléas de donnée

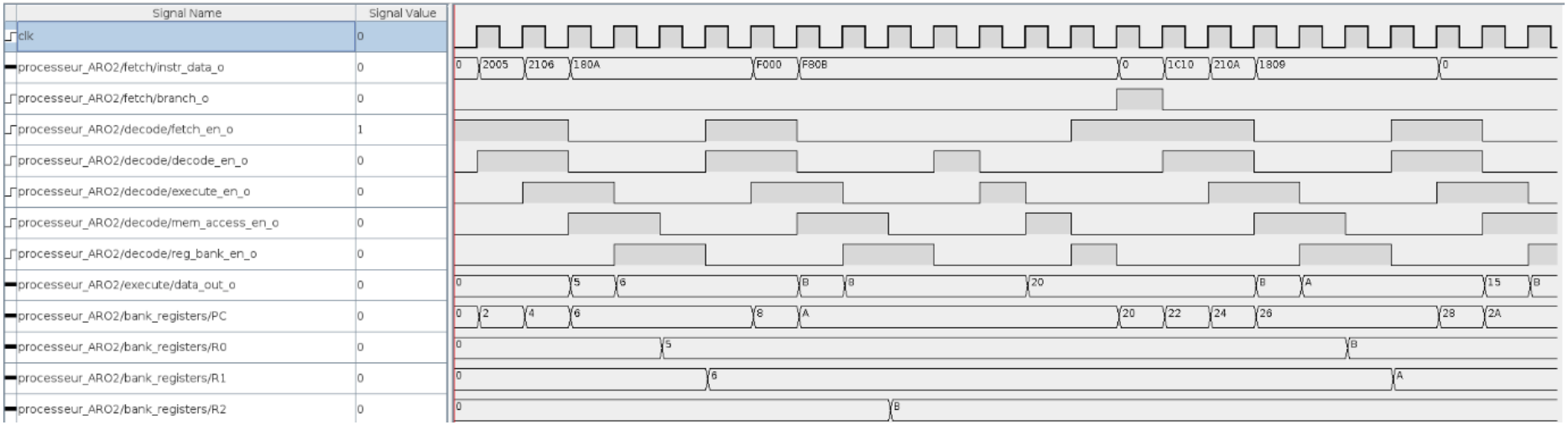
1.4 Programme et fonctionnement théorique

```
@ Programme de test des aléas de données

mov r0, #5
mov r1, #6
add r2, r1, r0  @RAW I2(R1), I1(R0)
bl test_1

.org 0x20
test_1:
mov r0, r2      @RAW I3(R2), I1(R0)
mov r1, #10
add r1, r0      @RAW I5(R0), I6(R1)
```

			Dep RAW	1	2	3	4	5	6	7	8	9	10	11	12
		Arret Pipeline / Forwarding													
1	MOV	r0, #5		F	D	E	M	W							
2	MOV	r1, #6			F	D	E	M	W						
3	ADD	r2, r1, r0	I2(R1), I1(r0)			F	D	E	M	W					
4	BL	0x20													
0x20															
5	MOV	r0, r2	I3(R2)						F	D	E	M	W		
6	MOV	r1, #10								F	D	E	M	W	
7	ADD	r1, r0	I5(R0)								F	D	E	M	W

En principe, avec un pipeline forwardé, il nous faudrait au minimum 12 cycles pour terminer notre programme, peut-être un peu plus car l'instruction BL ne se comporte pas comme une instruction de contrôle standard.

1.4.1 Chronogramme



Nous voyons dans ce chronogramme Que notre processeur ne fonctionne pas comme en théorie avec le forwarding. En effet, lors d'aléas (instructions 180A et 1809), le processeur met à 0 les enable de chaque étage du pipeline : le fetch, le décode, l'execute. Le memory access et le writeBack (RegBank). Or en théorie, ces aléas devraient ne pas stopper le pipeline, pas prendre de coups d'horloge supplémentaires grâce au forwarding.

A cause de cela, notre programme prend 22 coups de clock pour s'exécuter. A noter que l'instruction BL s'exécute en 8 coups de clock. Tout vient à penser que ce circuit de forwarding ne fonctionne pas correctement.

1.4.2 Questions

1. Est-ce que les valeurs dans les registres sont mises à jour correctement et au bon moment ?

Sans aller chercher trop loin, vu que nous constatons que le forwarding ne fonctionne pas, nous pouvons sans risque dire que non. Cependant, nous n'avons pas eu le temps de chercher d'où vient ce problème de forwarding. Peut-être vient-il du fait que les 3 dernières valeurs calculées par l'execute ne sont pas mémorisées ?

2. Pourquoi l'instruction BL génère un aléa de contrôle et un aléa de donnée ?

L'instruction BL est composée de 2 instructions : bl_msb et bl_lsb.

bl_msb calcule une partie de l'adresse du saut et l'écrit dans le LR, puis bl_lsb lit le LR et calcule d'adresse du saut complète.

Il y a donc un aléa de données, car il faut que bl_msb ait fini d'écrire avant que bl_lsb vienne lire dans le LR

Il y a aussi un aléa de contrôle, car il faut attendre que la première partie de l'adresse du saut soit calculée via le bloc execute.

3. Combien de cycles sont nécessaires pour résoudre les aléas de l'instruction BL ?

Comme vu dans le chronogramme, 8 cycles. Cela correspond à 2x4 cycles,donc la résolution de d'un aléa de donnée et un aléa de contrôle. Cependant ce n'est pas normal dans un pipeline forwardé.

4. Quel est l'IPC pour votre programme ?
   Nous avons 7 instructions et un arrêt de 4 cycles pour 2 instructions et un arrêt de 8 cycles pour 1 instruction. On peut donc dire que nous avons un arrêt de 5.3 cycles pour 3 instructions.
   3 instructions = 43% du nombre d'instructions total

$$IPC = \frac{1}{0.57 + 0.43 * (5.3 + 1)} = 0.3\ IPC$$

2 Pipeline Forwarding

2.1 Circuit data_hazard

1. A quoi sert le signal sel_mem_i ?

Sel_mem_i nous informe qu'une instruction d'accès à la mémoire de données est décodée. Typiquement les instructions LDRH et STRH

2. Est-il possible/utile de faire un data forwarding depuis le stage WRITE_BACK ? (L'écriture dans le registre dans la banque de registres). Comment pourrait-il être ajouté au circuit ?

Si on veut garder le fonctionnement du forwarding, il faudrait mémoriser les données dans le WRITE_BACK (dans notre cas Reg Bank). Il faudrait repasser la donnée mémorisée dans le RegBank au bloc execute en cas d'aléa de type RAW, pour que l'execute calcule le résultat de l'instruction précédente.

3. Quelles sont les conditions pour que le forwarding puisse avoir lieu ? Quelles sont les conditions pour que le forwarding soit utile ?

Pour que le forwarding ait lieu, il faut que le résultat écrit dans un registre de la banque de données soit disponible immédiatement à l'étage de l'execute lors de l'instruction suivante. Pour que le Forwarding soit utile, il faut que le programme à exécuter comporte des aléas de type RAW. Dans ce cas, nous pouvons gagner des coups de clock.

4. Quelles sont les conséquences du forwarding sur la gestion des aléas de données? Quelles sont les conséquences du forwarding sur la gestion des aléas de contrôle?

En cas d'aléas de données, sauf dans le cas d'une instruction de type LDR, le forwarding permet de ne pas arrêter les blocks du pipeline, car le bloc execute a accès directement aux données nécessaire pour faire le calcul de la prochaine instruction. On économise.

Il n'y a pas de conséquences pour la gestion des aléas de contrôle, car de toutes manière, le calcul du saut doit se faire dans le bloc execute. Nous perdons donc 2 cycles avec ou sans forwarding.

2.2 Circuit Execute

1. Pourquoi doit-on faire ça ?

Pour la gestion des aléas de données, il faut prendre la valeur du dernier résultat de l'execute pour pouvoir effectuer l'instruction courante

2. Pourquoi doit-on faire ça pour le signal reg_mem_data_s ?

On doit stocker l'adresse de la mémoire de donnée pour les instructions LRDH et STRH pour aller la chercher au cycle d'après dans le cas d'un aléa de donnée.

3. Que devrait-on faire si on avait un data forwarding venant du WRITE_BACK ?

Il faudrait ajouter un registre supplémentaire dans le WRITE_BACK permettant de fournir la valeur calculée de l'instruction précédent au bloc execute

2.3 Test : pipeline forwarding

1. Est-ce que votre processeur fonctionne correctement ? Est-ce que les timings sont respectés ? Est-ce que les registres contiennent les bonnes valeurs si on regarde étape par étape l'exécution des instructions ?

Comme on nous a fourni le circuit déjà terminé, nous avons fait ce programme au point 1.3

2. Quel est l'IPC de votre programme ? et le throughput si on considère une clock à 4KHz ?

Déjà effectué au point 1.3

3. Combien de cycles sont nécessaires pour que l'instruction BL soit complétée ?

Déjà effectué au point 1.3

4. Avez-vous d'autres idées d'optimisation de ce processeur ?

De la prédiction de branchement, pour éviter de perdre 2 cycles lors d'un saut.

Nous fournir un processeur forwardé qui fonctionne =).