HEIG VD | REDS Institut Reconfigurable and Embedded Digital Systems

Prof. Romuald Mosqueron

Prof. Olivier Auberson

Laboratoire d'architecture des ordinateurs

semestre printemps 2023 - 2024

Microarchitecture PIPELINE

Informations générales

Le rendu pour ce laboratoire sera **par groupe de deux**, chaque groupe devra rendre son travail sur Cyberlearn.

Le rendu du laboratoire sera évalué comme indiqué dans la planification des laboratoires. Tout retard impactera la note obtenue.

> ⚠ **N'oubliez pas de sauvegarder et d'archiver votre projet à chaque séance de laboratoire**

NOTE : Nous vous rappelons que si vous utilisez les machines de laboratoire situées au niveau A, il ne faut pas considérer les données qui sont dessus comme sauvegardées. Si les machines ont un problème nous les remettons dans leur état d'origine et toutes les donnés présentes sont effacées.

Objectifs du laboratoire

L'objectif est d'implémenter dans le processeur pipeliné un mécanisme permettant de supporter les aléas de données. Pour ceci, il vous sera demandé de réaliser la logique pour la détection des aléa de données. Puis ensuite en fonction des aléas de données, il faudra stopper les étages du pipeline nécessaire pour résoudre les aléas et ainsi exécuter correctement les instructions du programme en cours. Dans le projet Logisim rien n'est encore mise en place concernant l'arrêt du pipeline.

Vous devez rendre les projets Logisim ainsi que les codes en assembleur, le tout sera noté. Vous pouvez également rendre les réponses aux questions qui seront corrigées mais pas évaluées.

Outils

Pour ce labo, vous devez utiliser les outils disponibles sur les machines de laboratoire (A07/A09) ou votre ordinateur personnel avec la machine virtuelle fournie par le REDS.

Fichiers

Vous devez télécharger à partir du site Cyberlearn un ".zip" contenant un répertoire «workspace» où vous trouverez :

- **processeur_ARO2_pipeline_student.circ** : Le fichier de travail Logisim
- **main.S** : fichier source du code assembleur pour tester le circuit.
- **Makefile** : fichier contenant les directives d'assemblage
- **0X_main.S** : Les différents programmes à tester.

Attention : Vous ne devez pas utiliser les fichiers du précédent laboratoire. Créez un nouveau répertoire.

⚠ **Le fichier Makefile ne doit pas être modifié !**

⚠ **Respectez l'architecture hiérarchique présentée dans le cours.**

Travail demandé

Ce laboratoire est divisé deux parties.

- Ajout du mécanisme de détection des aléas de données
- Gestion des signaux permettant de stopper le pipeline

1 Aléas de donnée

Vous pouvez modifier les entrées/sorties des différents blocs si vous le désirez. Dans ce cas, vous devez rendre un fichier readme.txt qui décrit brièvement les différents changements effectués, en spécifiant les blocs et les signaux.

⚠ Si vous changez les entrées/sorties, la taille du bloc va changer et les signaux peuvent ne plus être connectés correctement. C'est votre responsabilité de contrôler que les blocs soient connectés correctement.

1.1 Circuit data_hazard

Ce circuit permet de détecter si une instruction en cours de décodage est dépendante d'une instruction précédente dont le résultat n'a pas encore été écrit dans un registre. Vous le trouverez instancié dans : decode -> main_control_unit -> hazard_detection.

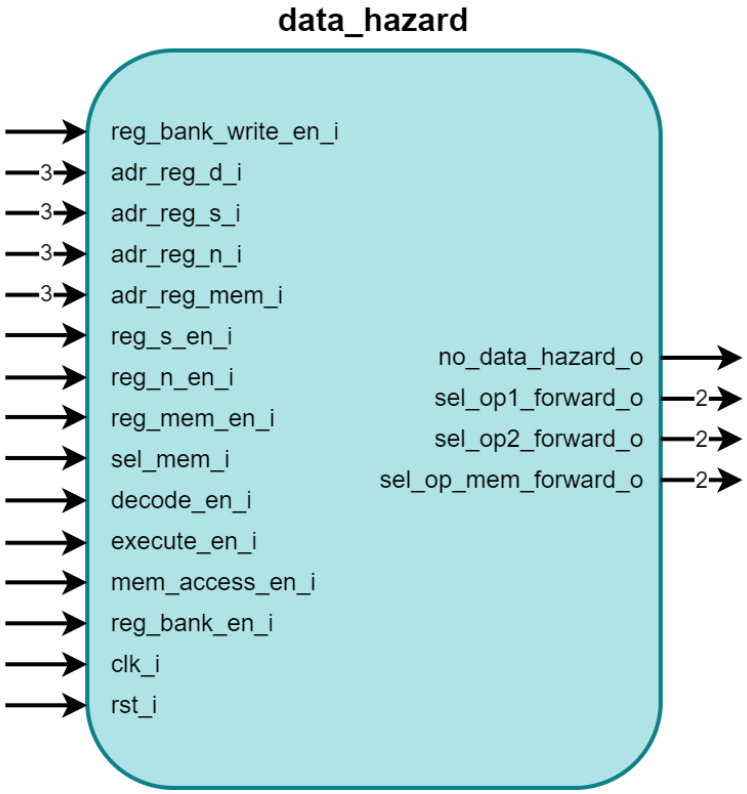

FIGURE 1 – Signaux du bloc data_hazard

Description des différents signaux du bloc :

Nom I/O	Description
reg_bank_write_en_i	Indique si l'instruction en cours de décodage écrit son résultat dans un registre
adr_reg_d_i	Registre à écrire
adr_reg_s_i	Registre à lire pour l'opérande 1
adr_reg_n_i	Registre à lire pour l'opérande 2
adr_reg_mem_i	Registre à lire pour l'opération mémoire
reg_s_en_i	Indique si l'opérande 1 lit une valeur d'un registre
reg_n_en_i	Indique si l'opérande 2 lit une valeur d'un registre
reg_mem_en_i	Indique si l'opération mémoire lit une valeur d'un registre
sel_mem_i	Indique si cette instruction est une instruction mémoire
decode_en_i	Enable du bloc DECODE
execute_en_i	Enable du bloc EXECUTE
mem_access_en_i	Enable du bloc MEMORY_ACCESS
reg_bank_en_i	Enable du bloc BANK_REGISTER
clk_i	Clock du système
rst_i	Reset asynchrone du système
no_data_hazard_o	Indique qu'il n'y **pas** d'aléa de donnée pour cette instruction
sel_op1_forward_o	Sélection d'une donnée forward-ée pour l'opérande 1
sel_op2_forward_o	Sélection d'une donnée forward-ée pour l'opérande 2
sel_op_mem_forward_o	Sélection d'une donnée forward-ée pour l'opérande des instructions mémoire

Pour le moment, nous nous intéressons uniquement à commander la sortie **no_data_hazard_o**. Répondez aux questions ci-dessous puis servez-vous de ces réponses pour créer le schéma Logisim permettant de détecter un aléa de donnée et de commander cette sortie.

Questions

1. Comment savoir si une instruction en cours de décodage est dépendante d'une instruction qui est pour le moment dans le stage EXECUTE ? dans le stage MEMORY_ACCESS ? Dans le stage WRITE_BACK ?
2. Est-ce que cela pose un problème si une instruction en cours de décodage dépend du résultat d'une instruction qui est au stage WRITE_BACK ?
3. Quels informations doivent être mémorisées pour chaque instruction ?
4. Quelles informations permettent de savoir si le registre D est utilisé ?

Vous devez compléter le contenu de ce bloc.

Indications

Un aléa de données est détecté si le ou les registres qui doivent être lus pour l'instruction courante correspond à un registre qui va être écrit par une instruction précédente. On peut donc séparer l'aléa de donnée en trois aléas distincts :

— Un aléa à cause du registre S

— Un aléa à cause du registre N

— Un aléa à cause du registre MEM

Il vous faudra comparer la valeur des signaux adr_reg_s_i, adr_reg_n_i, adr_reg_mem_i (si ils sont utilisés) de l'instruction en cours de décodage par rapport à l'adresse du registre D (s'il est utilisé) des instructions précédentes. Pour se faire il faut mémoriser à l'aide de registres, l'utilisation et l'adresse du registre D des instructions précédente qui sont dans les blocs EXECUTE, MEMORY_ACCESS et WRITE_BACK.

Attention, certaines instructions n'utilisent pas de registre en lecture ou pas tous les registres en lecture. C'est pourquoi il faut contrôler si les valeurs dans adr_reg_s_i, adr_reg_n_i et adr_reg_mem_i contiennent la valeur d'un registre à lire ou si leur valeur doit être ignorée.

De la même manière, certaines instructions ne font pas d'écriture dans le registre D, donc il faudra vérifier qu'il est bien utilisé. Ceci est aussi le cas, lorsque certains blocs du pipeline sont désactivés.

1.2 Circuit hazard_detection

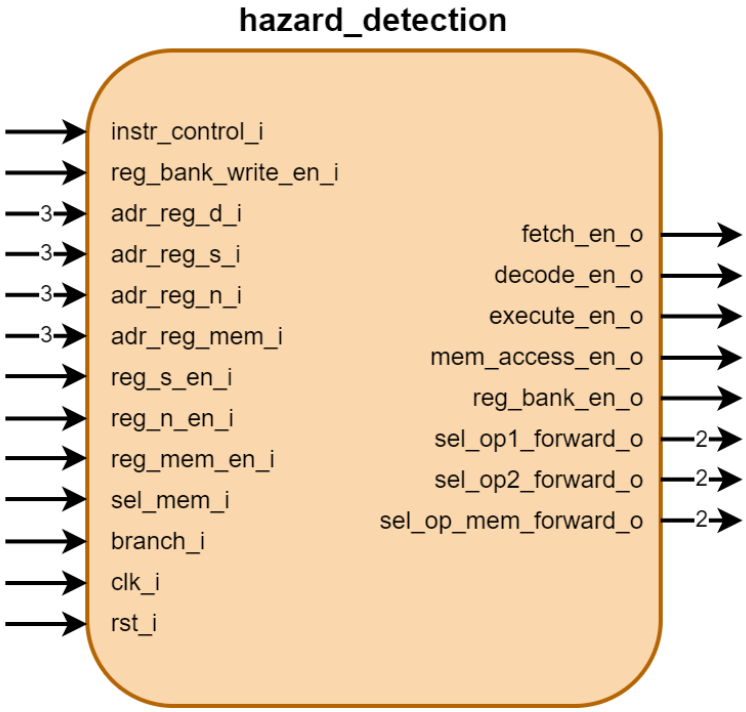

FIGURE 2 – Entrées/sorties du bloc hazard_detection

Description des différents signaux du bloc :

Nom I/O	**Description**
instr_control_i	Indique que l'instruction en cours de décodage est une instruction de contrôle
reg_bank_write_en_i	Indique si l'instruction en cours de décodage écrit son résultat dans un registre
adr_reg_d_i	Registre à écrire
adr_reg_s_i	Registre à lire pour l'opérande 1
adr_reg_n_i	Registre à lire pour l'opérande 2
adr_reg_mem_i	Registre à lire pour l'opération mémoire
reg_s_en_i	Indique si l'opérande 1 lit une valeur d'un registre
reg_n_en_i	Indique si l'opérande 2 lit une valeur d'un registre
reg_mem_en_i	Indique si l'opération mémoire lit une valeur d'un registre
sel_mem_i	Indique si cette instruction est une instruction mémoire
branch_i	Indique si l'instruction en cours de décodage suit un saut qui a été pris
clk_i	Clock du système
rst_i	Reset asynchrone du système
fetch_en_o	Enable du bloc fetch
decode_en_o	Enable du bloc decode
execute_en_o	Enable du bloc execute
mem_access_en_o	Enable du bloc memory_access
reg_bank_en_o	Enable du bloc bank_register
sel_op1_forward_o	Sélection d'une donnée forward-ée pour l'opérande 1
sel_op2_forward_o	Sélection d'une donnée forward-ée pour l'opérande 2
sel_op_mem_forward_o	Sélection d'une donnée forward-ée pour l'opérande des instructions mémoire

Ce circuit est instancié dans main_control_unit qui est instancié dans le bloc decode. Les connections des signaux d'entrées et sorties sont déjà faites. Mais dans le circuit main_control_unit, vous devez commander les 3 signaux **reg_s_en_s**, **reg_n_en_s**, **reg_mem_en_s**.

Commande des signaux dans main_control_unit

Répondez à la question ci-dessous puis servez-vous de la réponse pour commander les 3 signaux dans le circuit Logisim.

Question

5. Quelles informations permettent de savoir si le registre S, N ou MEM sont utilisés ?

Dans le circuit Logisim, vous devez commander les 3 signaux **reg_s_en_s**, **reg_n_en_s**, **reg_mem_en_s**.

Commande des signaux dans hazard_detection

Le circuit hazard_detection permet de stopper les étages du pipeline nécessaire pour résoudre l'aléa detecté. Lorsque l'aléa sera résolu, les étages du pipeline pourront reprendre leur exécution normal.

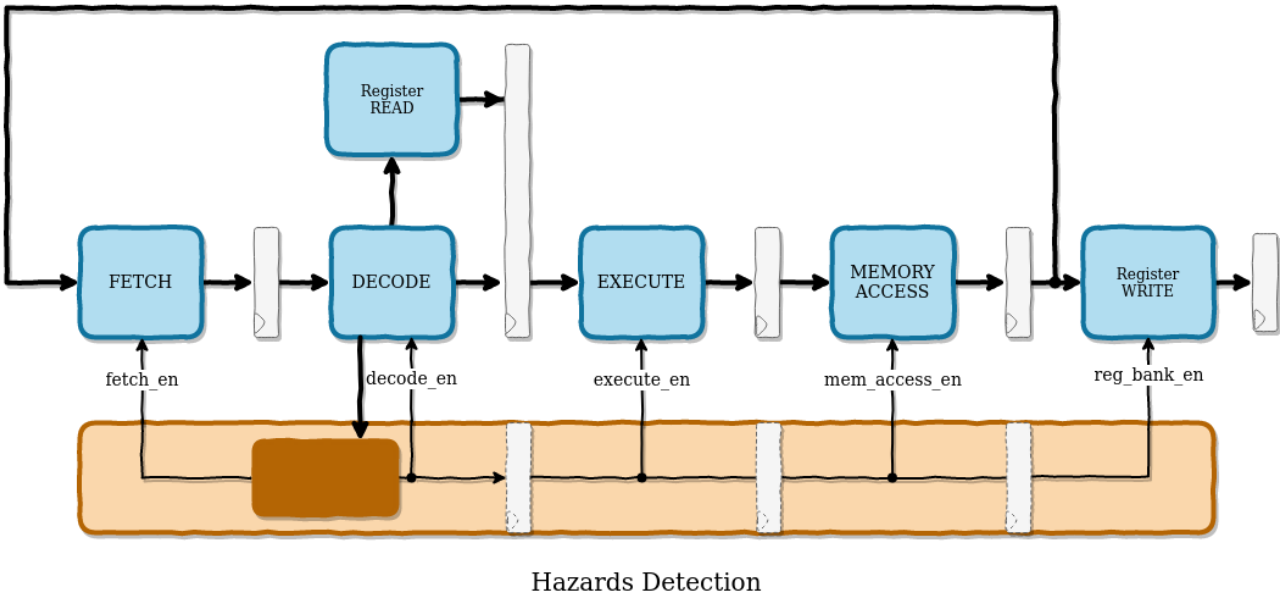

FIGURE 3 – Croquis du processeur pipeliné

Répondez à la question ci-dessous puis servez-vous de la réponse pour commander les signaux d'enable du cicuit hazard_detection.

Question

6. Une détection d'aléa de donnée va influencer quel(s) enable(s) d'étage du pipeline ? A quel moment ? Pourquoi ?

Dans le circuit hazard_detection, réaliser la commande des signaux **fetch_en_s**, **decode_en_s execute_en_s**, **mem_access_en_s** et **reg_bank_en_s**. Ces signaux dépendent du signal **no_data_hazard_s**. Vous pouvez vous aider du schéma 3 et de la réponse à la question précédente.

1.3 Test aléas de donnée

Tester le programme (01_main.S) que vous avez vu au laboratoire précédent et qui contient des aléas de données. Tester votre programme en faisant un chronogramme. Analysez l'exécution et déterminez si votre implémentation est juste.

Tester le programme (02_main.S) qui est un autre programme et qui contient des aléas de données. Tester votre programme en faisant un chronogramme. Analysez l'exécution et déterminez si votre implémentation est juste.

Question

7. Quel est l'IPC du programme (01_main.S) dans votre circuit logisim ? Pour ce calcul vous ne tiendrez pas compte des instructions *NOP* à la fin du programme ?
8. Quel est l'IPC du programme (02_main.S) dans votre circuit logisim ? Pour ce calcul vous ne tiendrez pas compte des instructions *NOP* à la fin du programme ?
9. Quel est le rôle des intructions *NOP* placées à la fin des programmes ? Tester avec et sans les instructions *NOP*.

Rendu

Pour ce laboratoire, vous devez rendre une archive qui comporte les 2 noms de votre binôme (ex : **rendu_nom1_nom2.zip**) et qui est composée de :

— votre fichier *.circ*

— les réponses aux questions (recommandé)

Votre rendu sera évalué sur le projet Logisim. Les réponses aux questions seront corrigées mais pas évaluées.

Tout retard ou un nom manquant dans le nom de l'archive rendu impactera l'évaluation du laboratoire.

CONSEIL : Faire une petite documentation sur cette partie vous fera directement un résumé pour l'examen.