Architecture des microprocesseurs (GIF-3000)
Département de génie électrique et de génie informatique
Automne 2023

UNIVERSITÉ LAVAL

Projet 2

Instructions : – Ce TP est à réaliser en équipe de **2** (max).
– Déposez sur l'ENA une archive **zip** de votre projet.
– Date limite : le lundi **27 novembre**, à `23h59`.

Pondération : Ce projet compte pour 10% de la note finale.

1 Objectifs

Ce projet comporte les objectifs suivants :

1. Comprendre le fonctionnement d'un microprocesseur implanté en pipeline ;
2. Réaliser les défis présents lors de la conception d'un microprocesseur ;
3. Approfondir les connaissances de circuits numériques avec microprocesseur.

2 Énoncé

Pour ce deuxième TP, vous devez ajouter à votre pipeline du TP précédent des chemins de traverse (forwarding) et une mémoire cache pour les données, toujours en utilisant le logiciel Logisim-Evolution [1].

Ce 2e TP étant une continuation du premier, toutes les instructions du 1er TP restent valides. Si vous aviez une solution fonctionnelle, vous serez en mesure de la réutiliser en grande partie.

L'objectif principal de ce TP est d'améliorer les performances de votre première implantation. Pour arriver à cette fin, vous explorerez deux pistes de solution. La première a pour but d'augmenter l'efficacité du pipeline en évitant l'insertion de bulles lorsque c'est possible. La seconde, a pour but d'augmenter en efficacité la gestion des opérations faites dans la mémoire des données. Vous serez amenés après chacune de ces étapes à mesurer la performance de votre solution améliorée. La métrique utilisée pour évaluer objectivement les performances d'une solution est la même que précédemment, à savoir le nombre de cycles d'horloge nécessaires pour exécuter le programme de test `performance.o`.

2.1 Chemins de traverse (forwarding)

Votre solution du TP1 forçait le pipeline à s'arrêter lorsqu'il détectait un aléas de type *RAW*. Un tel aléas se produit lorsqu'une opération doit être faite avec une mémoire qui a été modifiée mais qui n'a

1. https://github.com/logisim-evolution/logisim-evolution

pas encore eu le temps d'être écrite dans l'un des registres de la banque de registres. La solution était alors d'attendre que le résultat soit disponible dans la banque de registres afin de pouvoir aller chercher la valeur désirée et poursuivre l'exécution du programme.

Cependant, il est souvent possible de faire mieux. En effet, bien que la valeur du registre désirée ne soit pas encore disponible dans la banque de registres cette valeur est bien souvent connue à l'intérieur du pipeline. Cette valeur est contenue, comme vous le savez, soit dans le registre de phase de l'instruction précédente ou celle qui précède l'instruction précédente. Vous avez eu à utiliser le registre d'instruction *IR* des deux instructions précédentes à l'instruction présente dans la phase *ID* afin de déterminer les bulles à insérer. Cette fois, il faut déterminer où se trouve la donnée recherchée et de l'amener en utilisant les chemins de traverse pour éviter l'insertion de bulles.

Malheureusement, il n'est pas toujours possible d'éviter l'ajout de bulle dans un pipeline. En effet, si une instruction de type `load` cherche en mémoire une donnée dont l'instruction suivante à besoin, il est impossible de prévenir l'insertion de bulle. En effet, il s'agit du seul cas où la donnée n'est simplement pas connue à l'intérieur du pipeline au moment où il faudrait l'utiliser.

2.2 Cache

La cache est un mécanisme très utile pour augmenter les performances d'un système. En effet, comparativement à la vitesse du microprocesseur, les opérations en mémoires sont très lentes. Afin de rendre la simulation plus réaliste, les opérations faites dans la mémoire de données subiront une latence de 8 cycles d'horloge. Veuillez prendre note que durant le premier TP la mémoire des données répondait en moins d'un cycle d'horloge. Ceci avait pour but de simplifier le débogage de votre solution.

Une autre caractéristique du nouveau fonctionnement de la mémoire des données est son débit. Alors que la latence se mesure par le nombre de coups d'horloge que prend une mémoire pour arriver au microprocesseur depuis le moment où l'appel est fait, le débit se mesure par le ratio entre le nombre de mémoires appelées et le temps utilisé pour effectuer cette tâche(mesuré en coups d'horloge). Et ce, pour un très grand nombre d'opérations. La nouvelle mémoire de données prend 8 cycles d'horloge pour commencer à répondre mais peut fournir 1 mémoire par coup d'horloge supplémentaire. Par exemple, il faut 16 coups d'horloge pour effectuer deux lectures séparées. Cependant, 9 coups d'horloge suffisent si les appels sont effectués consécutivement. Pour un milliard de lecture en mémoire, il faudrait un milliard sept coups d'horloge. Ainsi, le ratio pour un très grand nombre d'opérations tend vers un. Le débit est dit d'être de 1 mémoire par cycle d'horloge.

Une hypothèse très importante est implicitement utilisée. Les données utilisées doivent être organisées séquentiellement en mémoire pour que cette stratégie puisse augmenter les performances du système. Dans le pire scénario, les performances pourraient se détériorer. En effet, chercher d'avance des valeurs qui ne seront pas utilisées avant que l'espace mémoire ne soit remplacé augmente la latence sans en améliorer la performance (9 coups d'horloge au lieu de seulement 8). Dans le cadre de ce TP, le programme `performance.o` permet de tester ceci.

2.2.1 Exigences de la cache

— Votre cache doit comporter exactement 8 mots de 12 bits, avec des blocs de 2 mots. L'objectif étant de simuler un microprocesseur à l'intérieur d'un contexte réaliste, ce nombre est choisi de manière à ce que la cache contribue à l'augmentation de la performance, mais sans pour autant que les programmes de test puissent entrer en entier dans la cache.
— Votre cache doit utiliser la stratégie du *write-back*, c'est-à-dire qu'elle doit retardez le plus longtemps possible les écritures en mémoire.
— Et votre cache doit être associative par groupe de deux blocs, en employant une stratégie LRU (*Least Recently Used*).

En résumé, votre cache doit comporter 4 blocs de 2 mots avec une stratégie LRU de remplacement par groupe de 2.

2.2.2 Conseils pour implémenter la cache

L'implantation d'une telle cache n'est pas simple. Beaucoup de détails sont à considérer en même temps. Il vous est *suggéré* de procéder à l'élaboration de votre cache en quatre phases :

1. **Commencez par implémenter une cache simple en write-through.** Aucune associativité, une seule lecture à la fois. Cette cache est la plus simple possible. Notez que ce type de cache fait en sorte que la cache est toujours cohérente avec la mémoire des données. Cette cache vous permet d'identifier les principaux modules à construire ainsi que vous familiariser avec les entrées-sorties et la manière d'interfacer avec la mémoire des données. Cette cache vous permet finalement de tester rapidement votre solution pour corriger les bogues avant que la solution soit complexe.
2. **Changer la cache pour un mode write-back**. Toujours une seule lecture à la fois et aucune associativité. Ce deuxième niveau de cache vous permet d'économiser beaucoup de coups d'horloge en évitant de devoir synchroniser la cache à la mémoire des données inutilement.
3. **Débit de 2.** Ce troisième niveau de cache vient profiter de l'effet du débit sur la latence et vient chercher une deuxième valeur immédiatement sachant que cette valeur sera probablement demandée bientôt.
4. **Associativité par groupe de 2.** Finalement, ce dernier niveau de cache ajoute l'associativité afin de préserver la mémoire la plus souvent utilisée.

2.3 Consignes supplémentaires

Plusieurs éléments ont été ajoutés dans la coquille Logisim qui vous est fournie. Notamment à l'intérieur du module *microprocesseur*, où certains registres de phases du pipeline ont été ajoutés pour permettre l'insertion des chemins de traverse. Il vous est donc possible de réutiliser votre solution en grande partie mais il sera nécessaire de la modifier pour ajouter les chemins de traverse. Faites attention si vous copier-coller votre solution d'un fichier à l'autre. Les modules peuvent différer même s'ils portent le même nom. Il est conseillé de refaire les branchements en vous servant de votre solution comme plan pour vous assurer de ne pas remplacer un module par un autre.

Vous devez exclusivement travailler à l'intérieur des blocs `microprocesseur` et `cache`.

3 Tâches à réaliser

Pour réaliser votre projet, vous devez :

1. Prendre le temps de bien lire le manuel de **Logisim**, car vous en ferez un usage **intensif**.
2. Maîtriser les différents composants de la **solution** partielle que nous vous fournissons. Assurez-vous de bien comprendre leur fonctionnement et n'hésitez pas à poser des questions lorsque des détails vous échappent.
3. Travailler dans le **cadre** que nous vous fournissons, et non tout refaire. Si vous refaites des choses, faites-les proprement, car cela pourrait rendre la correction **plus** difficile et, par conséquent, affecter votre note à la baisse.
4. Implanter **tous** les détails qui manquent, soit **beaucoup** de branchements entre les composants, la **logique** de contrôle du pipeline et sa mécanique de **blocage**, ainsi que la détection de **tous** les aléas, de même que la sélection des **chemins de traverse** qui permettent de les minimiser.
5. Implanter une **cache** pour votre processeur. La cache que nous vous fournissons ne fait rien, c'est-à-dire qu'elle se contente de relier ses entrées avec ses sorties correspondantes. On vous conseille **d'abord** de faire fonctionner votre processeur avant d'implanter une cache.
6. Corriger **tous** les bogues qui pourraient exister dans ce que nous vous fournissons.
7. Rédiger un rapport qui met en valeur la **qualité** de votre travail et de vos résultats.
8. Votre rapport devrait fournir la performance du programme `performance.o` à chacune des étapes de votre solution.
9. Finalement, indiquer la performance finale qu'obtient votre programme compteur que vous avez codé lors du TP1, et comparez-là avec votre performance précédemment obtenue.

4 Critères d'évaluation et rapport

Le projet sera évalué en utilisant les critères suivants :

1. **Fonctionnalité** (50%) : votre solution produit les bons résultats pour tous les programmes de test.
2. **Performance** (10%) : votre solution minimise le nombre de cycles nécessaires pour l'exécution de `performance.o`
3. **Cache** (10%) : Une cache associative par deux à correctement été implémentée.
4. **Esthétisme** (10%) : clarté et élégance de votre solution.
5. **Rapport** (20%) : Un rapport en format PDF présente clairement votre solution et vos résultats.

Pour votre rapport, **évitez** les banalités et le verbiage inutile. Utilisez la structure suivante :

1. **Solution** : autant que possible, illustrez votre solution afin de **faciliter** son évaluation. Qu'avez-vous fait de spécial qui **mérite** notre attention et qui n'est pas a priori évident ?

2. **Résultats et analyse** : présentez les résultats que vous avez obtenus avec nos programmes de test. Le cas échéant, présenter aussi d'autres résultats intéressants que vous auriez pu obtenir avec d'autres programmes. Comment votre microprocesseur se comporte-t-il ? Est-il sans bogue ? Est-il efficace ?
3. **Conclusion** : que doit-on retenir de votre travail ? Comporte-t-il des limitations ? En êtes-vous satisfait ? Si c'était à refaire, que changeriez-vous ?
4. **Documents à remettre** : vous devez remettre votre rapport en format pdf, votre solution logisim ainsi que la banque de programmes incluant votre programme compteur du TP1.

Bonne conception !

8/11/2023