HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Prof. Romuald Mosqueron

Prof. Marina Zapater

# Laboratoire d'architecture des ordinateurs

semestre printemps 2022

## Microarchitecture PIPELINE - Partie 1

## Informations générales

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

Un rapport intermédiaire vous sera également demandé à la fin du troisième laboratoire (EXECUTE) ainsi qu'un rapport final à la fin du dernier laboratoire (PIPELINE)

Le rendu du laboratoire ne sera pas évalué comme indiqué dans la planification des laboratoires, cependant il permet aux professeurs/assistants d'assurer un suivi de votre travail. Tout retard impactera l'évaluation du rapport intermédiaire ou final.

⚠ **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 de comprendre le fonctionnement d'un processeur dit pipeliné. Vous recevrez le processeur complet en version pipeliné et des programmes à exécuter. Cette version du processeur ne comprend aucun mécanisme de détection des aléas et d'arrêt du pipeline. A la fin du laboratoire, vous devrez modifier le circuit pour ajouter ces mécanismes. Vous devez rendre le projet Logisim et **tous les codes assembleur** que vous écrivez ou modifiez.

## 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.

⚠ **L'installation de la machine virtuelle doit se faire en dehors des séances de laboratoire afin que vous puissiez profiter de poser des questions pendant le laboratoire. L'installation n'est pas comptée dans les périodes nécessaires à la réalisation de ce laboratoire.**

## Fichiers

Vous devez télécharger à partir du site Moodle (Cyberlearn) un .zip contenant :

- Le fichier de travail Logisim (contenant le processeur)
- Le fichier source du code assembleur main.S (contenant un programme)
- Le fichier Makefile contenant les directives d'assemblage

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 en deux parties. Chaque partie est notée séparément.

Pour la première partie vous devez réaliser :

- Analyse et test du processeur pipeliné
- Ajout de la détection d'aléas de contrôle

Pour la deuxième partie vous devrez réaliser lors du prochain laboratoire :

- Ajout de la détection d'aléas de donnée
- Optimisation du pipeline avec la technique du pipeline forwarding

# 1 Analyse et test du processeur

Vous devez avoir fini cette partie après la première séance.

## 1.1 Analyse du processeur

Le processeur qui vous a été fourni a été pipeliné à partir du processeur que vous aviez implémenté dans les laboratoires précédents. Certains changements ont du être opérés pour pouvoir supporter le pipeline. Pour pipeliner un processeur, il ne suffit pas d'ajouter des registres entre chaque bloc.

Il faut, par exemple, s'assurer que tous les signaux de contrôle arrivent au bon moment. Le signal execute_control_bus est généré au moment où l'instruction est décodée, mais il est utilisé au moment où l'instruction est executée. Il faut donc que le signal de contrôle arrive au même moment que l'instruction dans le bloc execute. Le schéma ci-dessus est un croquis du processeur pipeliné. Les registres sont en gris. Sur le schéma, il y a un grand registre entre les stages du pipeline, or dans Logisim, il y a un registre par signal. Register READ et Register WRITE sont implémentés dans Logisim dans bank_register.

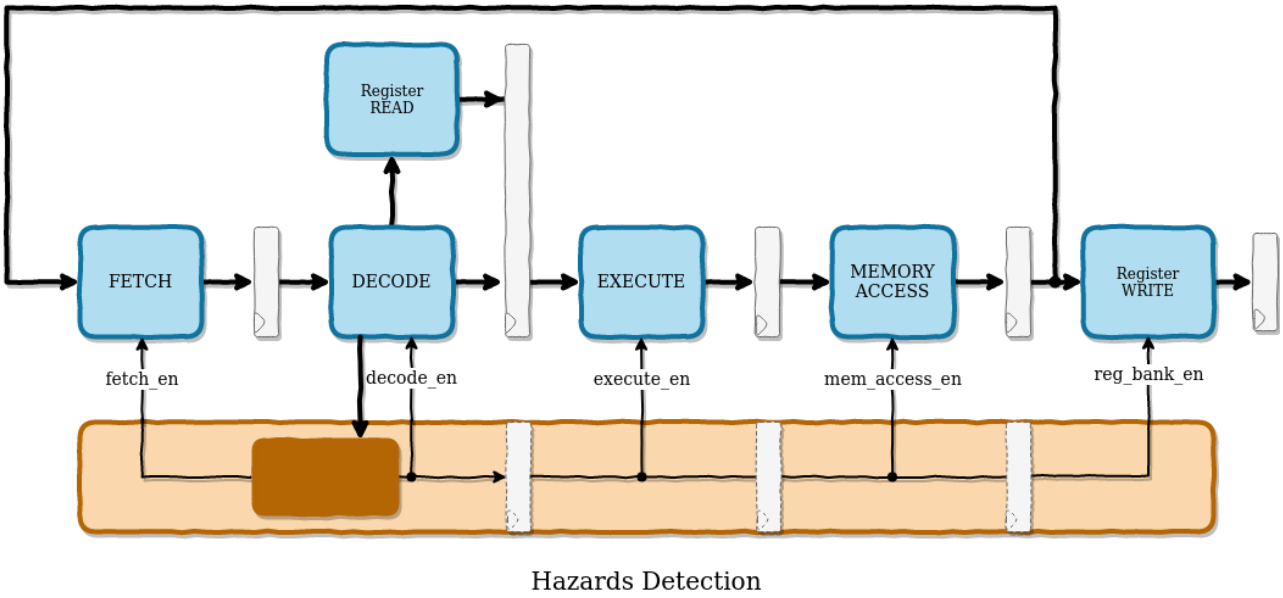


FIGURE 1 – Croquis du processeur pipeliné

Les changements qui ont été faits pour le processeur pipeliné :

- Dans le circuit mult_2, les offsets sont incrémentés de 1 au lieu d'être incrémenté de 2.
- Dans le circuit LR_manager, le signal link_en_i passe dans 3 registres au lieu de 1, pour que le signal link_en_d1_s soit généré au bon timing.
- Le signal branch_i est calculé dans memory_access au lieu de fetch car c'est dans ce bloc que les informations sont disponibles pour le calcul.
- Les signaux passent par tous les blocs même s'ils ne sont pas utilisés dans un bloc. Ceci pour assurer que les informations de contrôle arrive en même temps que les données dans le bloc qui les utilisent.

Dans Logisim, regardez les différents blocs et répondez aux questions suivantes.

### Questions

1. Dans le circuit mult_2, les offsets sont incrémentés de 1 au lieu d'être incrémenté de 2 dans le circuit non-pipeliné, pourquoi ?

2. Dans le circuit fetch, le signal LR_adr_o vient d'un registre et est connecté au bloc decode au lieu du bloc bank_register, pourquoi ?
3. Dans le circuit decode, le signal adr_reg_d_s est mis dans un registre alors que les signaux adr_reg_n_s, adr_reg_m_s et adr_reg_mem_s sont directement connecté à la sortie, pourquoi ?
4. Dans le circuit decode, les signaux des bus de contrôle sont connectés aux registres avec une porte MUX contrairement aux autres signaux, pourquoi ?
5. Si on voulait ajouter le multiplieur 5x3 pipeliné du laboratoire préparatoire, quelles seraient les conséquences sur le pipeline du processeur ? Comment ça pourrait être fait ?

### 1.2 Test du processeur

Compiler et tester le programme suivant.

```
@ programme 1
mov r0,#0x3E
mov r1,#3
mov r2,#0xCB
mov r3,#6
nop
@ Partie à analyser
add r4,r0,#2
strh r2,[r0,#4*2]
ldrh r1,[r0,#4*2]
b fin
nop
nop
nop
nop
nop
.org 0x40
fin:
and r1,r3
nop
nop
nop
nop
nop
@ fin de l'analyse
```

Relevez le chronogramme de l'exécution du code ci-dessus depuis le début du traitement de l'instruction *add r4, r0,#2* jusqu'à la fin du traitement de l'instruction *and r1,r3* Vous devez vous inspirer de l'exemple donné en cours. Votre chronogramme doit comprendre les signaux suivants : clock, PC, sortie du registre de chacun des 5 étages du pipeline.

**Questions**

1. Est-ce que le programme s'exécute correctement ? Est-ce que les registres prennent les bonnes valeurs ?
2. Combien de cycles sont nécessaires pour exécuter ce programme ?

### 1.3 Assembleur : dépendances de données

Dans le programme main.S qui vous a été fourni, indiquez en commentaire pour la première partie (depuis *MAIN_START* jusqu'à *B PART_2*), les dépendances de données pour chaque instruction. Relevez le chronogramme de l'exécution du code.

Ajoutez le nombre minimum d'instructions *NOP* pour résoudre les aléas de donnée. Relevez le chronogramme de l'exécution du code.

**Questions**

1. Quelles dépendances posent des problèmes d'aléas ?
2. Combien de cycles sont nécessaires pour résoudre un aléa de donnée ?
3. Quelle est l'IPC ? Le throughput si la clock vaut 4KHz ? La latence ?

### 1.4 Assembleur : aléas de contrôle

Dans la deuxième partie (depuis l'instruction *B PART_2*) du programme main.S qui vous a été fourni, ajoutez le nombre minumum d'instructions *NOP* pour résoudre les aléas de contrôle. Relevez le chronogramme de l'exécution du code.

**Questions**

1. Combien de cycles sont nécessaires pour résoudre un aléa de contrôle ?
2. Quelle est l'IPC ? Le throughput si la clock vaut 4KHz ? La latence ?

# 2 Aléas de contrôle

## 2.1 Circuit control_hazard

Ce circuit permet de détecter si le pipeline doit être bloqué à cause d'un aléa de contrôle. Vous devez compléter le circuit de ce bloc afin de générer un signal **no_ctl_hazard_o** qui indique qu'il n'y pas d'aléa de contrôle. Lorsqu'une instruction qui génère un aléa de contrôle est détectée (entrée instr_control_i), la sortie no_ctl_hazard_o doit être mise à 0 pendant un certain nombre de coups de clock. Puis le signal doit de nouveau être à 1 pendant au minimum 1 coup de clock pour laisser la prochaine instruction être fetch-ée.

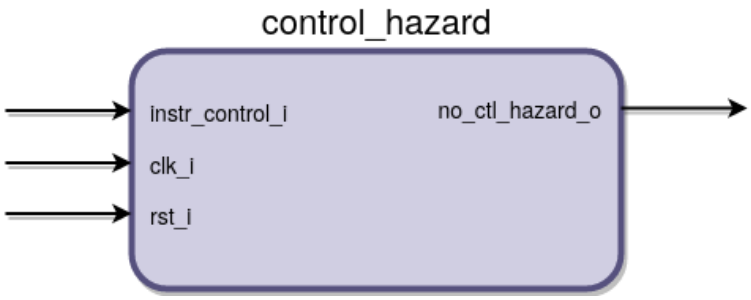


FIGURE 2 – Entrées/sorties du bloc control_hazard

Description des différentes entrées/sorties du bloc :

| **Nom I/O** | **Description** |
|---|---|
| instr_control_i | Indique que l'instruction en cours de décodage est une instruction de contrôle |
| clk_i | Clock du système |
| rst_i | Reset asynchrone du système |
| no_ctl_hazard_o | Indique s'il n'y a **pas** d'aléas de contrôle pour cette instruction |

Comme précédemment, répondez aux questions ci-dessous puis transposez vos réponses sur Logisim.

**Questions**

1. Combien de cycles le pipeline doit être bloqué dans le cas d'un aléa de contrôle ?
2. Pourquoi faut-il bloquer le pipeline lorsqu'il y a un aléa de contrôle ?
3. Quels sont les conditions pour qu'un aléa de contrôle ait lieu ?
4. Que se passe-t-il si une instruction génère un aléa de contrôle et un aléa de donnée ?

## 2.2 Circuit hazard_detection

Ce circuit est instancié dans main_control_unit qui est lui dans le bloc decode. La plupart des connections de ce bloc sont déjà effectuées.

Dans le circuit main_control_unit vous devez ajouter les connections pour le signal **instr_control_s**. Ce signal indique qu'une instruction va générer un aléa de contrôle. Les signaux du bloc hazard_detection sont décrits ici.

Description des différents signaux du bloc :

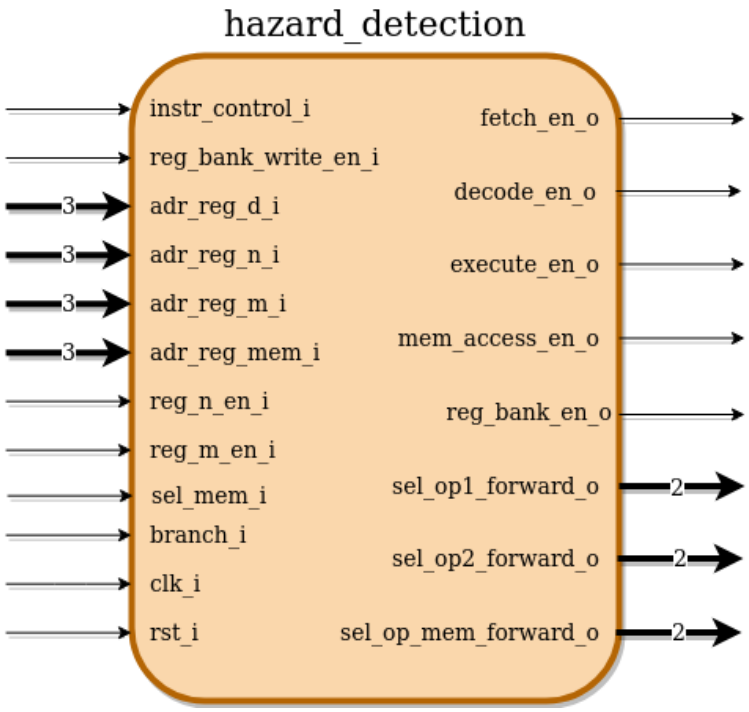


FIGURE 3 – Entrées/sorties du bloc hazard_detection

| **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 requiert l'écriture d'un registre |
| adr_reg_d_i | Registre à écrire |
| adr_reg_n_i | Registre à lire pour l'opérande 1 |
| adr_reg_m_i | Registre à lire pour l'opérande 2 |
| adr_reg_mem_i | Registre à lire pour l'opération mémoire |
| reg_n_en_i | Indique si l'opérande 1 est lue d'un registre |
| reg_m_en_i | Indique si l'opérande 2 est lue 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 |

Analyser le circuit hazard_detection, en particulier comment les signaux **fetch_en_s**, **decode_en_s** **execute_en_s**, **mem_access_en_s** et **reg_bank_en_s** sont générés à partir des signaux **no_ctl_hazard_s** et **branch_s** et répondre aux questions.

**Questions**

1. Quelles instructions génèrent un aléa de contrôle ?
2. Comment les aléas de contrôle influencent les différents enables ?
3. Que se passe-t-il dans le pipeline si un saut est pris ? Quelle est la prochaine instruction exécutée ?

4. Pourquoi branch_i est dans les entrées du circuit hazard_detection ?
5. Pourquoi instr_control_i du bloc control_hazard dépend de no_data_hazard_s ?

### 2.3 Test aléas de contrôle

Ecriver un programme qui contient des aléas de contrôle. Tester votre programme en faisant un chronogramme. Eviter pour le moment l'instruction *BL* car elle génère un aléa de donnée et un aléa de contrôle.

**Questions**

1. Est-ce que les valeurs dans les registres sont mises à jour correctement et au bon moment ?
2. Quel est l'IPC pour votre programme ?
3. Pourquoi l'instruction *BL* génère en même temps un aléa de contrôle et un aléa de donnée ?

## 3 Rendu

Votre rendu ne sera pas évalué comme indiqué dans la planification des laboratoires, cependant il permet aux professeurs/assistants d'assurer un suivi de votre travail. Tout retard impactera l'évaluation du rapport intermédiaire ou final.

Vous devez rendre les fichiers suivants :

— Le circuit processeur_ARO2.circ dûment complété.
— Le circuit mul_pipeline_1.circ réalisé lors du laboratoire de préparation au pipeline.
— Le code main.S modifié en suivant le point 1. de la donnée
— Le code que vous avez utilisé pour tester les aléas de contrôle.