

## Laboratoire d'architecture des ordinateurs semestre printemps 2022 - 2023

### Microarchitecture PIPELINE

---

#### Informations générales

---

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

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 données 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. Il vous sera demandé d'implémenter la logique pour la détection des aléas de données. Rien n'est encore en place concernant l'arrêt du pipeline. Pour finir, vous implémenterez le forwarding.

Vous devez rendre deux projets Logisim (avec et sans forwarding) ainsi que **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.


## 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 plusieurs parties.

- Analyse et compréhension du processeur pipeline
- Ajout du mécanisme de détection des aléas de données
- Gestion des signaux permettant de stopper le pipeline
- Implémentation du mécanisme de forwarding

# 1 Analyse et test du processeur

## 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 dû 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 exécutée. 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 blocs du pipeline, or dans Logisim, il y a un registre par signal. Register READ et Register WRITE sont implémentés dans Logisim dans `bank_register`.

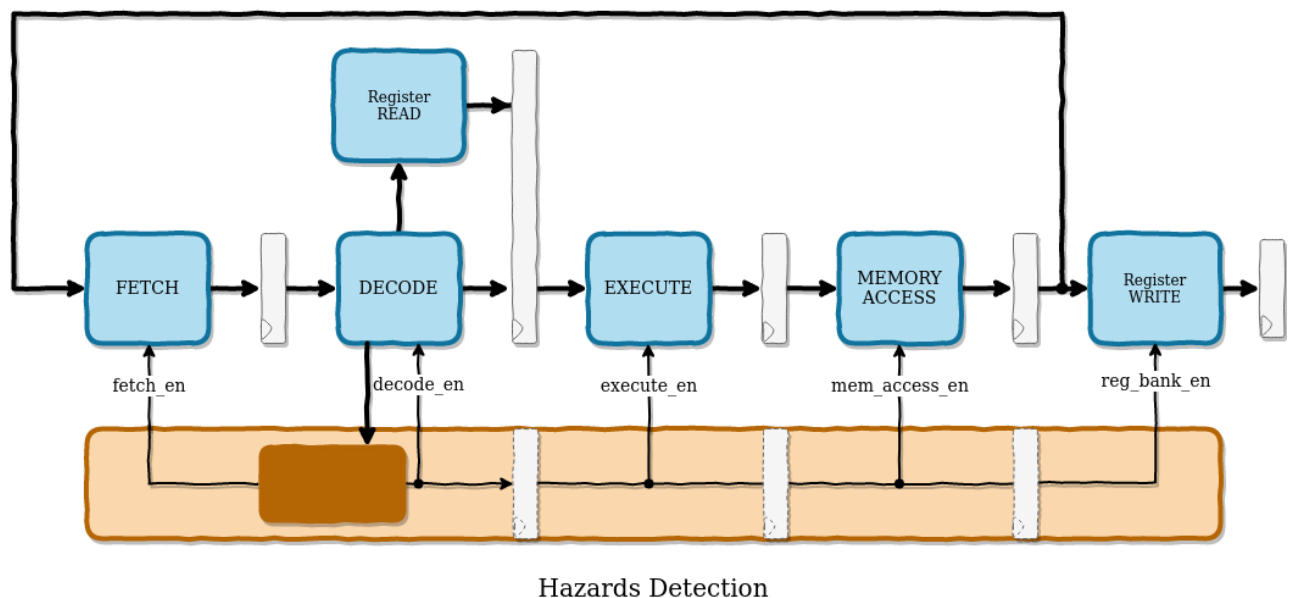


Figure 1: Croquis du processeur pipeliné

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.
- 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_decode_reg`, 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` va dans le circuit `decode_execute_reg` dans un registre alors que les signaux `adr_reg_n_s`, `adr_reg_m_s` et `adr_reg_mem_s` ne sont pas stockés dans des registres, pourquoi ?
4. Dans le circuit `decode_execute_reg`, 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.

Ce programme est fourni dans l'archive que vous avez téléchargé (01\_main.S). Pour le compiler, il vous suffit de copier le contenu de ce fichier dans `main.S` et de compiler avec la commande "make".

```
@ programme 1
mov r0,#0x3E
mov r1,#3
nop
nop
mov r2,#0xCB
add r3,r0,r1
add r3,r2,#2
nop
sub r1,r2,r1
strh r3,[r0,#4*1]
strh r1,[r0,#4*2]
```

Analysez le comportement du système en réalisant un chronogramme de l'exécution du programme.

### Questions

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

## 1.3 Assembleur: dépendances de données

Dans le programme 02\_main.S qui vous a été fourni, indiquez en commentaire les dépendances de données pour chaque instruction. Relevez le chronogramme de l'exécution du code. Pour le compiler, il vous suffit de copier le contenu de ce fichier dans `main.S` et de compiler avec la commande "make".

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. Quel est l'IPC sur le programme fourni ? Le throughput si la clock vaut 4KHz? La latence?

## 2 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.

### 2.1 Circuit data\_hazard

Ce circuit permet de détecter si une instruction en cours de décodage est dépendante du résultat d'une instruction qui n'est pas encore écrit dans un registre. Vous le trouverez instancié dans : decode -> main\_control\_unit -> hazard\_detection.

Vous devez compléter le contenu de ce bloc.

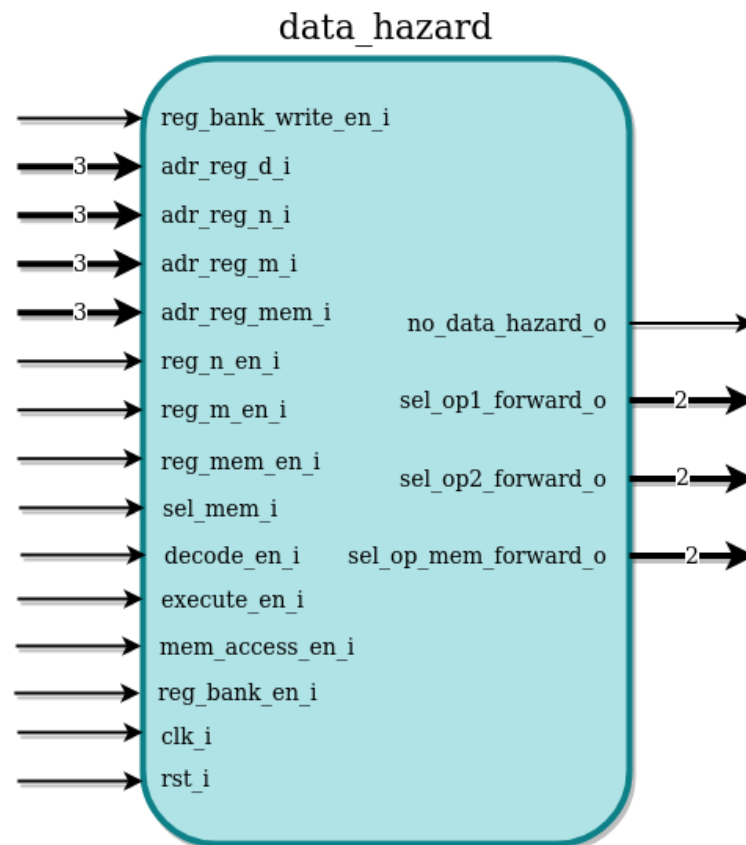


Figure 2: Signaux du bloc data\_hazard

Description des différents signaux du bloc:

Nom I/O	Description
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
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 <b>pas</b> 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 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 ça pose un problème si une instruction dépend du résultat d'une instruction qui est au stage WRITE\_BACK?
3. Quelles informations doivent être mémorisées pour chaque instruction?
4. Quelles informations permettent de savoir si le registre D est utilisé?

### 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 N
- Un aléa à cause du registre M
- Un aléa à cause du registre MEM

Il vous faudra comparer la valeur des signaux `adr_reg_n_i`, `adr_reg_m_i`, `adr_reg_mem_i` avec l'adresse du registre D (s'il est utilisé) de l'instruction qui est dans le bloc EXECUTE, MEMORY\_ACCESS et WRITE\_BACK. Pour se faire il faut mémoriser à l'aide de registres, l'utilisation et l'adresse du registre D des instructions qui passent dans le pipeline.

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_n_i`, `adr_reg_m_i` et `adr_reg_mem_i` contiennent la valeur d'un registre qui va être lu ou si leur valeur doit être ignorée.

De la même manière, certaines instructions ne font pas d'écriture dans le registre D. Ceci est aussi le cas si par exemple, certains blocs du pipeline sont désactivés.

## 2.2 Circuit hazard\_detection

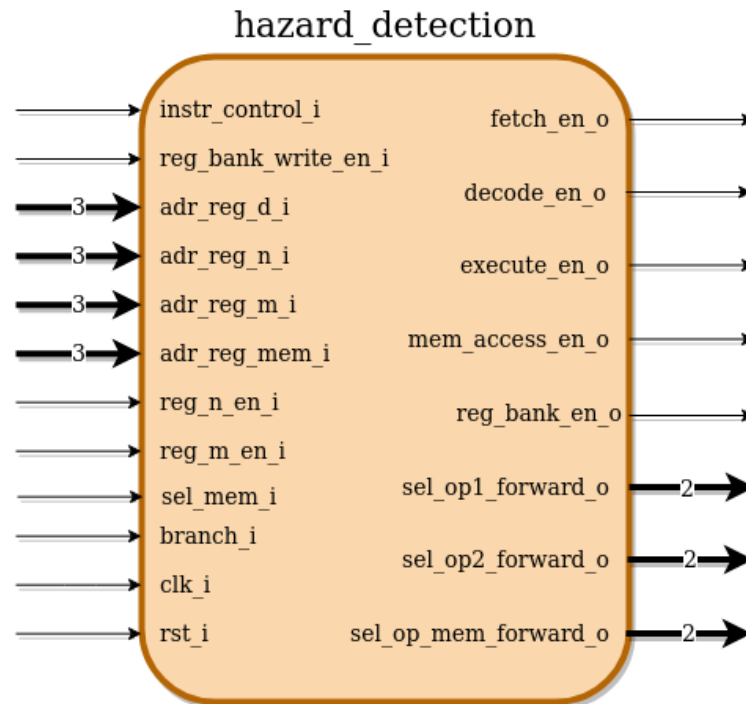


Figure 3: Entrées/sorties du bloc hazard\_detection

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 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

Ce circuit est instancié dans main\_control\_unit qui est instancié dans le bloc decode. La plupart des

connections sont déjà faites. Vous devez ajouter dans le circuit `main_control_unit` les connections pour les signaux `reg_n_en_s`, `reg_m_en_s`, `reg_mem_en_s`. Les signaux du bloc `hazard_detection` sont décrits dans le tableau précédent.

### Commande des signaux dans `main_control_unit`

Répondez aux questions ci-dessous puis transposez sur Logisim vos réponses et complétez le circuit.

#### Questions

1. Quelles informations permettent de savoir si le registre N, M ou mem sont utilisés?

### Commande des signaux dans `hazard_detection`

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 de `no_data_hazard_s`. Vous pouvez vous aider du schéma 4.

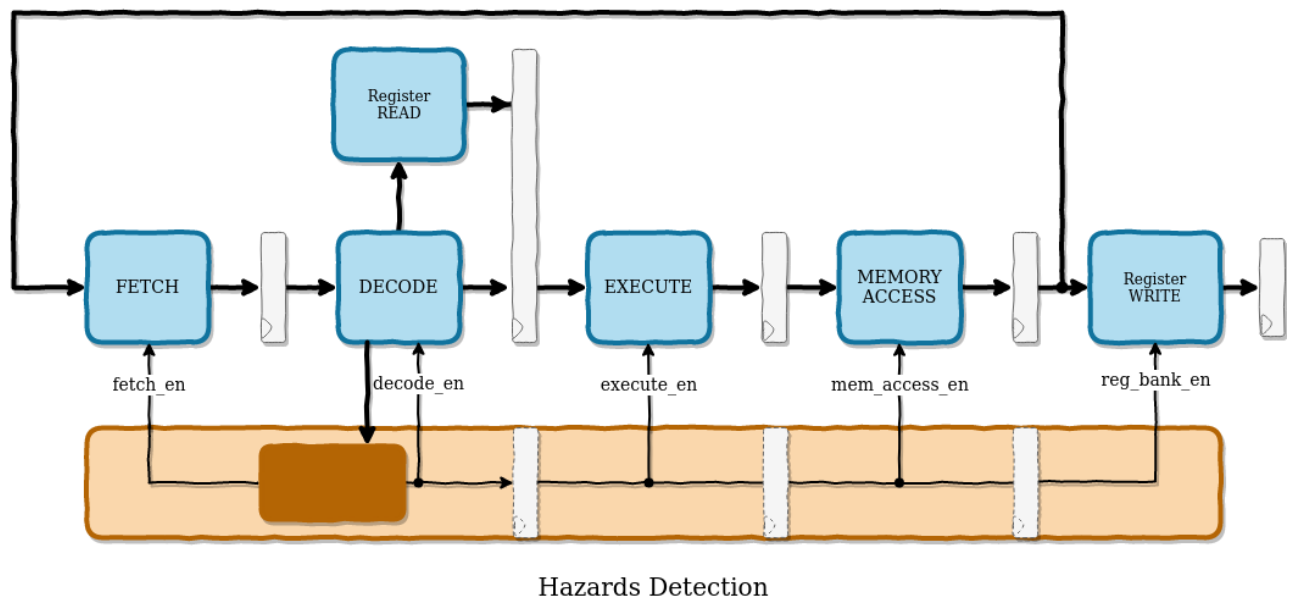


Figure 4: Croquis du processeur pipeliné

#### Questions

1. Quelles informations permettent de savoir si le registre D est utilisé?
2. Une détection d'aléa de donnée va influencer quel(s) enable(s)? A quel moment ? Pourquoi?

### 2.3 Test aléas de donnée

Ecrire un programme qui contient des aléas de donnée. Tester votre programme en faisant un chronogramme. Analysez l'exécution et déterminez si votre implémentation est robuste !



### 3 Pipeline Forwarding

⚠️ **Faites une copie de votre workspace qui contient la gestion des aléas et travailler dans la copie du workspace.**

Vous devez rendre un circuit séparé pour le pipeline forwarding.

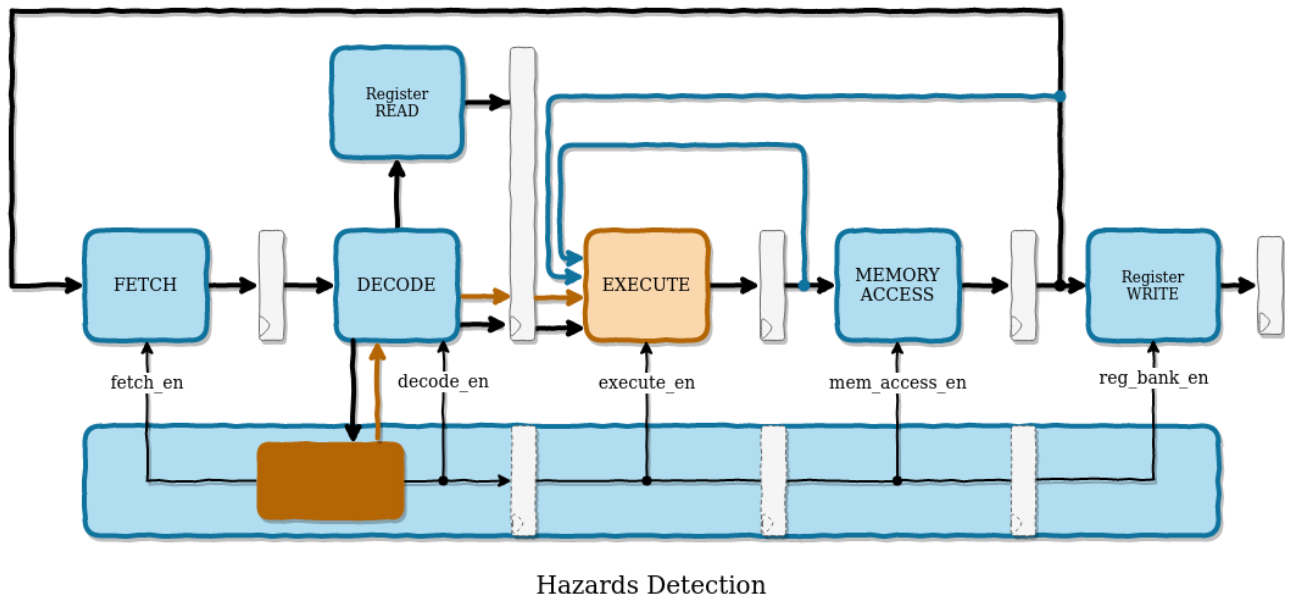


Figure 5: Croquis du processeur optimisé avec du pipeline forwarding

Afin d'optimiser la gestion des aléas, deux chemins pour les données ont été ajoutés au circuit afin d'obtenir la nouvelle valeur des registres avant qu'elle soit écrite dans les registres comme sur la Figure ci-dessus.

#### 3.1 Circuit data\_hazard

Vous devez faire des modifications au circuit `data_hazard` pour créer les signaux `sel_op1_forward_s`, `sel_op2_forward_s` et `sel_op_mem_s`. Ces signaux doivent prendre les valeurs suivantes.

Valeur	Description
0	Il n'y pas de data forwarding possible ou nécessaire pour cette instruction
1	Data forwarding depuis le bloc EXECUTE
2	Data forwarding depuis le bloc MEMORY_ACCESS

`sel_op1_forward_s` fait la sélection pour l'opérande 1. `sel_op2_forward_s` fait la sélection pour l'opérande 2 et `sel_op_mem_forward_s` fait la sélection de la donnée pour les instructions mémoire.

#### Questions

1. A quoi sert le signal `sel_mem_i`?
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?
3. Quelles sont les conditions pour que le forwarding puisse avoir lieu? Quelles sont les conditions pour que le forwarding soit utile?
4. Quelles sont les conséquences du forwarding sur la gestion des aléas de données?

## Indications

Les valeurs qui proviennent d'une instruction mémoire ne peuvent pas être forwardées.

### 3.2 Circuit execute

Vous devez modifier le circuit Execute pour que **operand\_1\_s** et **operand\_2\_s** soient sélectionnés avec les signaux **sel\_op1\_forward\_s** et **sel\_op2\_forward\_s** respectivement. Ils doivent pouvoir garder leur valeur si le forwarding n'est pas effectué ou prendre la valeur **forward\_execute\_data\_s** ou **forward\_mem\_data\_s** dépendant d'où le forwarding est effectué.

La même chose doit être fait pour le signal **reg\_mem\_data\_s** grâce au signal **sel\_op\_mem\_forward\_s**

#### Questions

1. Pourquoi doit-on faire ça?
2. Pourquoi doit-on faire ça pour le signal **reg\_mem\_data\_s**?
3. Que devrait-on faire si on avait un data forwarding venant du WRITE\_BACK?

### 3.3 Test: pipeline forwarding

Vous pouvez réutiliser le programme que vous aviez fait pour la gestion des aléas pour tester votre pipeline avec le forwarding. Faites un chronogramme et regardez si tout est correct.

#### Questions

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?
2. Quel est l'IPC de votre programme? et le throughput si on considère une clock à 4KHz?
3. Avez-vous d'autres idées d'optimisation de ce processeur?

## 4 Rendu

---

Tout retard impactera l'évaluation labo.

Vous devez rendre les fichiers suivants:

- Le circuit processeur\_ARO2.circ avec gestion des aléas de donnée sans forwarding que vous renommerez : processeur\_ARO2\_p1.circ.
- Le circuit processeur\_ARO2.circ avec gestion des aléas de donnée sans forwarding que vous renommerez : processeur\_ARO2\_p2.circ.
- Les différents codes assembleurs que vous aurez utilisés, modifiés ou écrits.