

## Laboratoire d'architecture des ordinateurs semestre printemps 2022

### Microarchitecture FETCH

---

#### 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 données présentes sont effacées.

#### Objectifs du laboratoire

---


L'objectif principal de ce laboratoire est la réalisation de la partie FETCH d'un processeur simplifié. L'idée sera de développer un système de A à Z afin que vous puissiez faire chaque étape vous-mêmes et ainsi bien comprendre les concepts vus dans la théorie du cours afin de les appliquer dans un cas pratique.

Vous devez rendre les projets Logisim ainsi que les codes en assembleur

## Outils

---

Pour ce laboratoire, 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 Cyberlearn un .zip contenant un répertoire «workspace» où vous trouverez :

- **labo\_processeur.circ** : Le fichier de travail Logisim
- **main.S** : fichier source du code assembleur
- **Makefile** : fichier contenant les directives d'assemblage

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

## Workspace fourni

---

Si tout se passe bien durant le laboratoire, vous devriez pouvoir réaliser tous les laboratoires dans le workspace téléchargé pour le premier laboratoire.

Vous allez recevoir un schéma qui contient :

- Mémoire d'instructions
- Mémoire de données
- Processeur\_ARO2
- Contrôleur mémoire

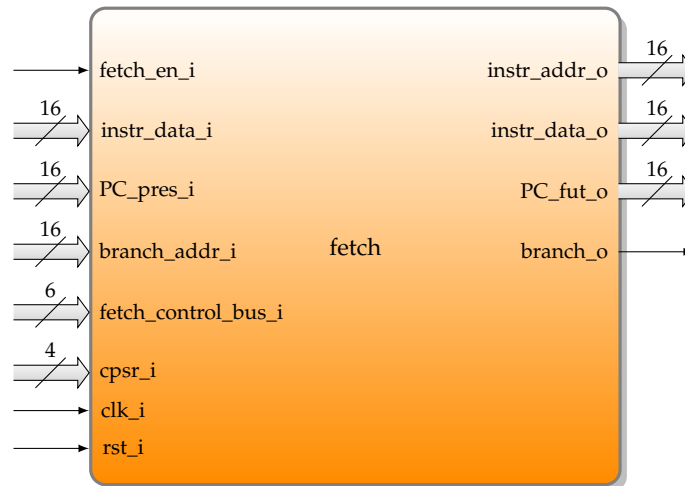
Ainsi que la plupart des entités que vous allez réaliser dans le cadre de ce cours. Certaines sont déjà complétées, ce sont soit des parties qui prennent trop de temps selon nous ou alors qui ne sont pas très constructives à réaliser. Cependant, lorsque vous allez en avoir besoin, nous vous expliquerons leur fonctionnement.

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

# 1 FETCH

## Travail à effectuer

### Entité du bloc FETCH



Nom I/O	Description
fetch_en_i	Permet de superviser le fonctionnement du FETCH
instr_data_i	Données mémoire d'instructions
PC_pres_i	Etat présent du Program Counter
branch_addr_i	Valeur à mettre dans PC lors d'un saut
fetch_control_bus_i	Bus de controle de la partie FETCH
cpsr_i	Conditions à vérifier
clk_i	Entrée d'horloge
rst_i	Entrée du reset asynchrone
instr_addr_o	Adresse pour la mémoire d'instructions
instr_data_o	Instructions courante à traiter
PC_fut_o	Valeur future du Program Counter
branch_o	Flag indiquant un saut validé (conditionnel ou non)

**NOTE :** L'enable du FETCH sera implémenté dans ce laboratoire et permettra de pouvoir réaliser le laboratoire PIPELINE à la fin du semestre. Il est donc normal que l'implémentation de cette fonctionnalité dans ce laboratoire puisse ne pas paraître utile pour la version actuelle mais permettra d'éviter de devoir modifier le bloc FETCH dans un autre laboratoire.

**NOTE :** Le fetch control bus transporte des informations qui viendront par la suite du bloc DECODE (prochain laboratoire). Il permet d'avoir une seule connexion, cela vous simplifiera la vie pour la suite des laboratoires. Le bus doit répondre à la configuration suivante :

Position	Taille	Description
0	1	Flag qui indiquera que l'instruction courant est un saut (conditionnel ou non)
1	1	Flag qui indiquera que l'instruction courante est conditionnelle
5-2	4	Permet de sélectionner la condition que vous voulez tester
6	1	Pas utilisé dans ce laboratoire

### Etape 1-a : Implémenter un registre représentant le PC

Par la suite, le PC sera un registre de la banque de registres. Mais pour l'instant, vous allez devoir

instancier un registre directement dans le circuit **processeur\_ARO2**.  
Connectez-le avec les entrées/sorties liées au Program Counter du bloc FETCH.

#### **Etape 1-b : Implémenter l'incrémentation du PC dans le bloc FETCH**

Vous allez devoir implémenter l'incrémentation du Program Counter. Comme vous êtes en train de réaliser une architecture 16 bits, vous devez incrémenter le Program Counter par le nombre de Byte que vous accédez. L'adresse allant jusqu'à la mémoire d'instructions est divisée par deux avant la ROM, splitter dans le circuit main.

#### **Etape 1-c : Implémenter les entrées/sorties de la mémoire d'instructions**

Câbler correctement les signaux qui sont connectés à la mémoire d'instructions (instr\_data\_i, instr\_data\_o et instr\_addr\_o).

### Etape 2-a : Ecrire un programme simple qui utilise les instructions supportées


Utiliser les fonctions *mov*, *add* et *and* afin de pouvoir compiler un programme (**au minimum 5 instructions**). Par exemple :

```
mov r1 , #5
add r0 , r2 , r3
```

### Etape 2-b : Compiler le programme écrit à l'étape 2-a

Vous allez compiler le programme que vous venez d'écrire. Vous avez deux options à partir de ce moment-là.

- **Geany** : Avec cet éditeur de texte, vous pouvez compiler avec le menu : Build -> Make.
- **terminal** : Placez-vous dans le dossier qui contient votre programme et tapez : make

 **Il ne faut absolument pas modifier le fichier Makefile**

### Etape 2-c : Charger le programme compilé dans la mémoire d'instructions

Aller dans le circuit main et sur la mémoire d'instructions :

1. clic-droit->clear contents : efface le contenu
2. clic-droit->load image : charge une image en mémoire
3. sélectionner le fichier main.raw

Vous devriez voir des valeurs dans la mémoire ROM d'instructions.

### Etape 2-d : Comparer le contenu de la mémoire avec les valeurs d'instructions compilées

Grâce au fichier main.lss, comparez les valeurs écrites dans la mémoire ROM d'instructions avec le programme que vous avez les instructions que vous avez écrites.

### Etape 3-a : Simuler pas à pas afin de vérifier le bon fonctionnement

Afin de vérifier le bon fonctionnement de cette première partie du bloc FETCH :

- Ajouter une sonde dans le bloc *processeur\_ARO2* sur la sortie *instr\_data\_o* afin de pouvoir observer que le bloc FETCH donne bien les bonnes instructions.
- Reset le PC (ctrl+r dans logisim ou alors simulate->reset simulation).
- Avancer pas à pas dans le programme en utilisant la touche F2 pour changer l'état de clk. Assurez-vous que F2 fait des sauts d'une période, et pas d'une demi période en allant dans Project > Options > Duration of Main tick (F2) Full period.
- Vérifier que les instructions qui sortent du bloc FETCH correspondent bien à ce qui attendu par rapport à la valeur du PC.

**Note** : Logisim possède un outil «**Assembly viewer**» (menu Simulate) qui permet d'afficher le fichier main.lss dans Logisim avec l'instruction courante automatiquement surlignée. Pour cela chargez le fichier main.lss (menu file de l'outil) et sélectionnez votre registre PC (votre registre doit avoir un nom). Vous devez être dans le bloc contenant le registre PC (pas dans le main). Essayez cet outil très pratique en faisant du pas à pas.

### Etape 3-b : Relever le chronogramme

- Simulate -> Chronogram
- Sélectionner les signaux intéressants pour cette étape
- Exécuter le programme pas à pas
- Analyser le chronogramme et veiller à bien comprendre ce que vous observez.

#### Etape 4-a : Ajout d'un mécanisme de saut dans le bloc fetch

En vous servant du bus de control *fetch\_control\_bus\_i* (qui sera commandée par le bloc decode plus tard). Vous allez devoir sauter à l'adresse donnée par l'entrée *branch\_addr\_i*. Il faut construire ce bus directement dans le circuit *processeur\_ARO2*. (voir description du bus présentée précédemment).

#### Etape 4-b : Ajout d'une partie de test dans le circuit *processeur\_ARO2*

Comme les blocs DECODE et EXECUTE n'ont pas encore été réalisés, vous allez faire des petites parties de chacun afin de pouvoir montrer les fonctionnalités les unes après les autres.

##### Etape 4-b-1 : Valeur pour le saut

Comme cette valeur sera calculée dans le bloc execute par la suite, vous allez simplement mettre une constante sur l'entrée (*wiring->constant*) afin de pouvoir tester manuellement le saut.

##### Etape 4-b-2 : Commande de saut

Cette commande sera fournie par le bloc decode, vous allez simplement placer une bouton sur l'entrée afin de pouvoir vérifier son bon fonctionnement.

##### Etape 4-b-3 : Tester votre commande de saut

Afin de vérifier que vous chargez correctement la valeur lorsque vous appuyez sur le bouton, simuler manuellement le fonctionnement.

#### Etape 5-a : Ecrire un programme avec des sauts inconditionnels

Ecrire un programme afin de faire des sauts inconditionnels. Par exemple :

```
ADR_SAUT_2:
@ instructions (2 ou 3)

B ADR_SAUT_1 @ saut inconditionnel en avant
.org 0x40
ADR_SAUT_1:
@ instructions (2 ou 3)

B ADR_SAUT_2 @ saut inconditionnel en arriere
```

**Note :** la directive d'assemblage **.org** permet de choisir l'adresse où sera placée dans la mémoire la partie du programme qui suit la directive.

#### Etape 5-b : compilez et vérifiez le code généré

- Analyser le code compilé contenu dans le fichier *main.lss*.
- Placer le code dans la mémoire ROM d'instructions du fichier *main.raw*.

#### Etape 6-a : Ajouter l'exécution des sauts à votre circuit

Dans cette partie, vous allez devoir concevoir des petites parties des blocs execute et decode afin de pouvoir exécuter le saut. Ces éléments seront directement placés dans le circuit *processeur\_ARO2* pour l'instant car ils seront réalisés dans les prochains laboratoires.

##### Etape 6-a-1 : Détection de l'instruction de saut

Comme vous n'avez pas encore réalisé le bloc DECODE, vous allez directement implémenter la détection de l'instruction de saut dans le circuit *processeur\_ARO2*. Veuillez vous référer au manuel d'instruction Thumb à disposition sur Cyberlearn afin de pouvoir décoder cette instruction.

### Etape 6-a-2 : Calcul de l'adresse de saut

Comme vous n'avez pas encore réalisé le bloc EXECUTE, vous allez directement implémenter le calcul de l'adresse de saut dans le circuit *processeur\_ARO2*. Veuillez vous référer au cours.

### Etape 6-a-3 : Câblage pour charger le PC

Câblez le système afin que vous puissiez exécuter une instruction de saut inconditionnel.

### Etape 6-b : Vérification de l'exécution

- Tester pas à pas votre programme et vérifier que le saut s'effectue correctement.
- Analyser le comportement du système dans un chronogramme.

### Etape 7-a : Implémentation de l'enable du fetch

- Cette partie est réalisée ici afin de pouvoir créer un bloc FETCH compatible avec une version pipelinée que vous verrez et réaliserez plus tard dans le cadre de ce cours. Il faut implémenter la fonctionnalité afin que lorsque l'enable n'est pas actif sur le bloc FETCH, ce dernier **ne doit plus** incrémenter le PC mais le maintenir à la même valeur.
- Vérifier le comportement de cette fonctionnalité.

### Etape 8-a : Implémentation du testeur de conditions

Un circuit *condition\_tester* vous est fourni. Il permet de tester les conditions telles qu'elles sont présentée dans le manuel d'instructions Thumb.

- Analyser le contenu de ce bloc et faire le lien avec le manuel d'instructions.
- Instancier ce bloc dans le FETCH.

Description du contenu du *CPSR* (Registre qui donne le status du programme) :

Position	Taille	Description
0	1	Z : equal zero flag
1	1	C : carry flag
2	1	N : negative number flag
3	1	V : overflow flag

**EXPLICATIONS** : Les flags qui permettent de tester les conditions qui sont nécessaires pour faire des saut conditionnels sont mis à jour dans le bloc EXECUTE. Vous allez le réaliser plus tard dans le laboratoire EXECUTE. Vous pouvez connecter une constante sur l'entrée *CPSR\_i* du bloc fetch. Vous pouvez regarder la connectique et comparer au manuel d'instructions.

Il vous faut maintenant considérer que lorsqu'il y a un saut, la sortie *cond\_test\_o* doit être à '1' afin que le saut ait lieu. Cette sortie est également à un lorsque le flag de l'enable sur la condition est à '0'. Ce qui permet de pouvoir l'utiliser directement dans une équation :

$$branch\_s = fetch\_control\_bus\_i(0) * cond\_test\_o$$

Ceci permet que le signal *branch\_s* soit inactif lorsqu'une instruction de saut conditionnel est détecté mais que la condition testée avec les flags du *CPSR* n'est pas respectée.

- Tester le fonctionnement du testeur de condition.

Pour se faire, vous pouvez mettre une constant sur l'entrée *CPSR* du votre bloc fetch et ainsi, choisir les flags *ZCNV* puis observer ce que cela implique en changeant les conditions testées à travers le bus de contrôle du FETCH.

## Etape 9 : Programme à rendre pour la validation

Voici le programme à réaliser. Vous devez écrire les instructions afin d'obtenir ce comportement. Ce programme sera celui qui valide votre laboratoire (autant le code que le circuit).

```
ADR_debut_prog :
@ charge 5 dans r0
@ charge 2 dans r1
@ r0 + r1 , resultat dans r2
@ r2 + 4 , resultat dans r3
@ Saut inconditionnel jusqu' a ADR_saut_1

ADR_saut_1 : (adresse 0x40)
@ charge 6 dans r2
@ r2 - 1 , resultat dans r3
@ saut conditionnel si egal zero jusqu' a ADR_saut_2 (condition si Z = '1')

ADR_saut_2 : (adresse 0x60)
@ Charge 4 dans r0
@ Charge 128 dans r1
@ Saut inconditionnel jusqu' a ADR_debut_prog
```

**REMARQUE :** Pour le saut conditionnel, vous allez devoir mettre le **bit 0** du *CPSR\_i* à '1' pour que le saut ait lieu et à '0' pour qu'il n'ait pas lieu. Vérifiez les deux situations afin d'être sûr(e)s que vous ayez tout câblé correctement.

## Rendu

---

Pour ce laboratoire, vous devez rendre :

- votre fichier *.circ*
- votre programme *main.S* de l'étape 9

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.

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