

Laboratoire d'architecture des ordinateurs semestre printemps 2023 - 2024

Microarchitecture FETCH

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'oublier 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-même 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, le tout sera noté. Vous pouvez également rendre les réponses aux questions qui seront corrigées mais pas évaluées.

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

Ce workspace sera utilisé uniquement lors de ce laboratoire FETCH.

Le schéma reçu contient :

- Mémoire d'instructions
- Mémoire de données
- Processeur_ARO2
- Contrôleur mémoire

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.

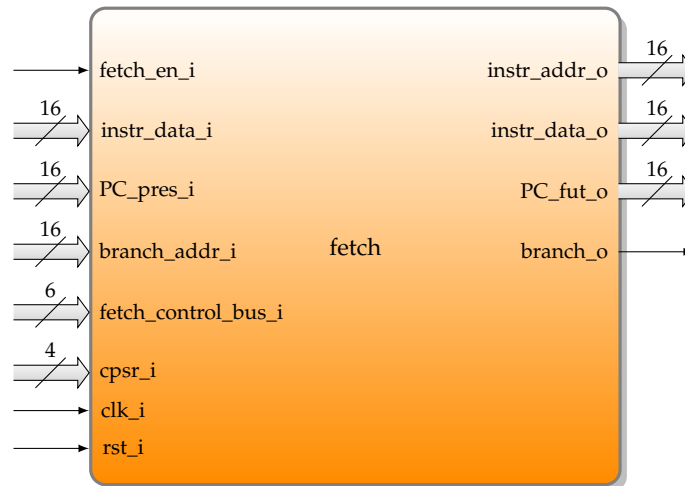
⚠ Respecter l'architecture hiérarchique présentée dans le cours.

⚠ Ne modifier pas les entrées/sorties des composants/entités fournis.

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 associée au saut conditionnel
6	1	Pas utilisé dans ce laboratoire

Etape 1-a : Connecter la mémoire d'instruction au processeur_ARO2

Dans le bloc main, connecter la mémoire d'instruction au processeur_ARO2 sur ces entrées/sorties : **instr_data_i** et **instr_addr_o**. L'adresse en sortie du composant processeur_ARO2 et qui va à la mémoire d'instructions doit être divisée par le nombre de byte de donnée accédé par le processeur. Un simple splitter permet de réaliser cette opération.

Etape 1-b : Implémenter un registre représentant le PC dans le bloc Processeur_ARO2

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**. Vous le nommerez PC. Connectez-le avec les entrées/sorties liées au Program Counter du bloc FETCH.

Etape 1-c : Implémenter l'incrémentation du PC dans le bloc FETCH

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 Bytes de données accédées. L'incrémentation doit être reliée aux signaux **PC_pres_s** et **PC_fut_s**.

Etape 1-d : Implémenter l'adressage de la mémoire d'instructions dans le bloc FETCH

Dans le bloc FETCH, câbler correctement le signal **instr_addr_o** qui est connecté à la mémoire d'instructions.

Question 1 :

Quel est la valeur de l'incrément du PC dans ce processeur? Justifier votre réponse.

Etape 2-a : Ecrire un programme simple


Dans le fichier main.S, 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, comparer les valeurs écrites dans la mémoire ROM d'instructions avec le programme que vous avez écrit. Les codes d'instructions doivent correspondre.

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

Vous allez simuler le circuit afin de vérifier le bon fonctionnement de cette première partie du bloc FETCH :

- Se placer dans le circuit main pour le simuler, puis utiliser le bouton "show simulation hierarchy" en haut à gauche de la fenêtre. Vous pouvez ensuite vous déplacer dans l'arborescence du circuit pendant la simulation.
- Visualiser la 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.
- Initialiser 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 (4 ou 5 signaux)
- Exécuter le programme pas à pas

Question 2 :

Relever et analyser le chronogramme. Veiller à bien expliquer ce que vous observez, vous pouvez annoter le chronogramme.

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

Dans le bloc FETCH, en vous servant du bus de control *fetch_control_bus_i* ajouter un mécanisme de saut. Le PC devra sauter à l'adresse donnée par l'entrée **branch_addr_i**.

Etape 4-b : Partie de test dans le bloc *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 : Ajout d'une valeur de saut dans le bloc *processeur_ARO2*

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

Etape 4-b-2 : Commande de saut dans le bloc *processeur_ARO2*

Le bus *fetch_control_bus* sera fourni plus tard par le bloc *decode*. Vous allez construire ce bus directement dans le circuit *processeur_ARO2*. (voir description du bus présentée précédemment). Placer un bouton sur l'entrée de commande de saut afin de pouvoir vérifier son bon fonctionnement.

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

Simuler manuellement le fonctionnement du saut, afin de vérifier que vous chargez correctement la valeur de saut dans le PC lorsque vous appuyez sur le bouton.

Etape 5-a : Écrire 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 inconditionnels 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 dans le bloc *processeur_ARO2*

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* et ainsi remplacer le bouton de l'étape 4-b-2. 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 dans le bloc *processeur_ARO2*

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* et ainsi remplacer la constante de l'étape 4-b-1. 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.

Question 3 :

Relever et analyser le chronogramme. Veiller à bien expliquer ce que vous observez, vous pouvez annoter le chronogramme.

Etape 7-a : Implémentation de l'enable du fetch dans le bloc 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 fournie. Il permet de tester les conditions telles qu'elles sont présentées 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 statut 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 sauts 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. Ce qui permet d'utiliser l'équation suivante :

$$branch_s = fetch_control_bus_i(0) \text{ AND } cond_test_o$$

Ceci permet que le signal *branch_s* soit actif lorsqu'une instruction de saut conditionnel ou incondi-tionnel est détectée. Dans le cas d'une instruction de saut conditionnel et que la valeur du CPSR ne respecte pas la condition de saut, le saut ne doit pas avoir lieu. L'équation utilisée respecte bien ce dernier cas car elle désactive le signal *branch_s* lorsque la sortie *cond_test_o* est à '0'.

- Tester le fonctionnement du testeur de condition.

Pour ce faire, vous pouvez mettre une constante sur l'entrée *CPSR* de 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-a : Ajouter l'exécution des sauts conditionnels à votre circuit

Comme pour les sauts inconditionnels de l'étape 6, vous allez ajouter les parties qui permettent de traiter des sauts conditionnels. Ces éléments seront placés dans le circuit *processeur_ARO2*.

Etape 9-a-1 : Détection de l'instruction de saut dans le bloc *processeur_ARO2*

Ajouter la détection de l'instruction de saut conditionnel à votre circuit. Puis, compléter la commande du bus de control *fetch_control_bus_i* avec les informations liées au saut conditionnel. (voir description du bus présentée précédemment).

Etape 9-a-2 : Calcul de l'adresse de saut dans le bloc *processeur_ARO2*

Modifier votre calcul de l'adresse de saut dans le circuit *processeur_ARO2* afin de supporter le calcul de l'adresse de saut pour un saut conditionnel et inconditionnel.

Etape 10 : 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.

Question 4 :

L'adresse de saut contenue dans les instructions de saut conditionnel et inconditionnel est une valeur signée ou non signée? Justifier votre réponse.

Rendu

Pour ce laboratoire, vous devez rendre une archive qui comporte les noms de votre binôme "**rendu_NomEtudiant1_NomEtudiant2**" et qui contient :

- votre fichier *.circ*
- votre programme *main.S* de l'étape 10
- les réponses aux questions (recommandé)

Votre rendu sera évalué sur les projets Logisim ainsi que les codes en assembleur. Les réponses aux questions seront corrigées mais pas évaluées.

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