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 EXECUTE et MEMORY ACCESS

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 principal de ce laboratoire est la réalisation de la partie EXECUTE 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 de cette partie. Une documentation des étapes réalisées sera également présente dans le rapport qui est à rendre pour le processeur.

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

Comme précisé dans le laboratoire précédent, l'objectif sera de reprendre votre workspace à l'étape du précédente afin de construire un processeur de A à Z. Vous pouvez donc continuer de travailler directement dans le même dossier que le laboratoire précédent. Pensez cependant à créer une sauvegarde afin de pouvoir revenir au début de cette partie si quelque chose devient, selon vous, trop compliqué.

NOTE : Si un laboratoire s'est mal passé pour un groupe, prendre contact avec l'assistant afin de pouvoir récuperer une version "clean" du workspace.

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

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.**

Ce laboratoire est divisé en deux parties :

— Le bloc EXECUTE

— Le bloc MEMORY ACCESS

1 Bloc EXECUTE

Entité du bloc EXECUTE

Nom I/O	Description
clk_i	Entrée d'horloge
rst_i	Entrée du reset asynchrone
reg_data_n_i	Données du registre choisi par Rn
reg_data_m_i	Données du registre choisi par Rm
imm3_i	Valeur immediate sur 3 bits
imm5_i	Valeur immediate sur 5 bits
offset_incond_i	Valeur d'offset pour le saut inconditionnel
offset_cond_i	Valeur d'offset pour le saut conditionnel
imm8_i	Valeur immediate sur 8 bits
execute_control_bus_i	Bus de contrôle du bloc execute
execute_en_i	Signal enable du bloc execute
data_out_o	Résultat en sortie du bloc execute
cpsr_o	Registre contenant l'état actuel du programme

Construction du bus de contrôle du bloc execute :

Position	Taille	Description
2-0	3	Sélection de l'opération du shift
5-3	3	Sélection de l'opération du bloc ALU
7-6	2	Sélection de l'opérande 2
8	1	Sélection de l'opérande 1
9	1	Sélection de la valeur d'entrée pour le shift
10	1	Flag qui met à jour le CPSR
13-11	3	Sélection de l'opération du bloc Mult_2
14	1	Pas utilisé

1.1 Réalisation du bloc shift

Afin de faire le lien avec cette partie, vous devez revenir à la logique que vous avez dû implémenter afin de gérer ce bloc dans le bus de contrôle (depuis le bloc décode).

— Implémenter les differentes sortes de shift afin de pouvoir exécuter les opérations supportées.

Suivre la table de vérité suivante :

sel_op_shift_s	shift_data_out_s
0	operand_s (bypass)
1	shift arithmétique (ASR)
2	shift logique à gauche (LSL)
3	shift logique à droite (LSR)
4	shift rotatif à droite (ROR)
5	0x0000 (pas assigné)
6	0x0000 (pas assigné)
7	0x0000 (pas assigné)

1.2 Réalisation du bloc ALU

Afin de faire le lien avec cette partie, vous devez revenir à la logique que vous avez dû implémenter afin de gérer ce bloc dans le bus de contrôle (depuis le bloc décode).

1.2.1 Operations arithmétiques

Réaliser la partie arithmétique de votre processeur. Celle-ci devra donner le résultat de l'operation choisie correspondant au tableau suivant :

sel_op_alu_s	data_out_s
0	operand_1_s (bypass)
1	operand_1_s + operand_2_s
2	operand_1_s - operand_2_s
3	operand_1_s AND operand_2_s
4	operand_1_s OR operand_2_s
5	NOT operand_1_s
6	NEG operand_1_s (=> *-1)
7	operand_1_s XOR operand_2_s

1.2.2 Gestion du carry

Donnez l'information du carry de l'opération qui vient d'être exécutée. Cette information permettra de mettre à jour le CPSR et donc de pouvoir prendre des décisions par la suite (conditions).

1.2.3 Gestion de l'overflow

Réalisez la gestion de l'overflow du calcul afin de pouvoir également donner cette information au CPSR. La technique de calcul de l'overflow est libre.

1.3 Réalisation de la sélection de l'opérande 2

Dans le circuit *execute* vous allez devoir implémenter la logique qui permettra de donner la bonne valeur au bloc *alu*.

Cette valeur doit répondre à la table de vérité suivante :

sel_operand_2_s	operand_2_s
0	reg_data_m_s
1	ext16_unsigned(imm3_s)
2	mult_2_out_s
3	ext16_unsigned(imm8_s)

Remarque : Le signal *mult_2_out_s* correspond à la sortie du bloc *mult_2*.

1.4 Réalisation de la sélection de l'opérande 1

Dans le circuit *execute* vous allez devoir implémenter la logique qui permettra de donner la bonne valeur à décaler au bloc *shift*.

La valeur à décaler pour le shift doit répondre à la table de vérité suivante :

sel_operand_1_s	operand_1_s
0	reg_data_n_s
1	0x0000

1.5 Réalisation de la sélection de la valeur pour le shift

Dans le circuit *execute* vous allez devoir implémenter la logique qui permettra de donner le décalage correct au bloc *shift*.

La valeur du décalage pour le shift doit répondre à la table de vérité suivante :

sel_shift_i	sel_shift_data_s
0	imm5_i(3 :0)
1	reg_data_m_i(3 :0)

1.6 Implémentation du CPSR

Implémenter le CPSR (Current Program State Register). Ce registre est composé de 4 bits, définis :

Position	Nom	Description
0	Z	Comparaison avec zéro
1	C	Indication du carry
2	N	Indication d'une valeur négative
3	V	Indication de l'overflow

Nous vous avons déjà préparé un bloc *zcnv_unit* qui permet de faire les diverses comparaisons qui permettent de construire ces flags.

Remarque : Il vous faut analyser ce bloc et comprendre les éléments à l'intérieur afin de pouvoir faire le lien avec le reste du système.

A vous de bien connecter le bloc à un registre afin de construire le CSPR correctement. Il sera important de réfléchir quand ce registre doit être écrit en fonction des informations qui arriveront du bus de contrôle.

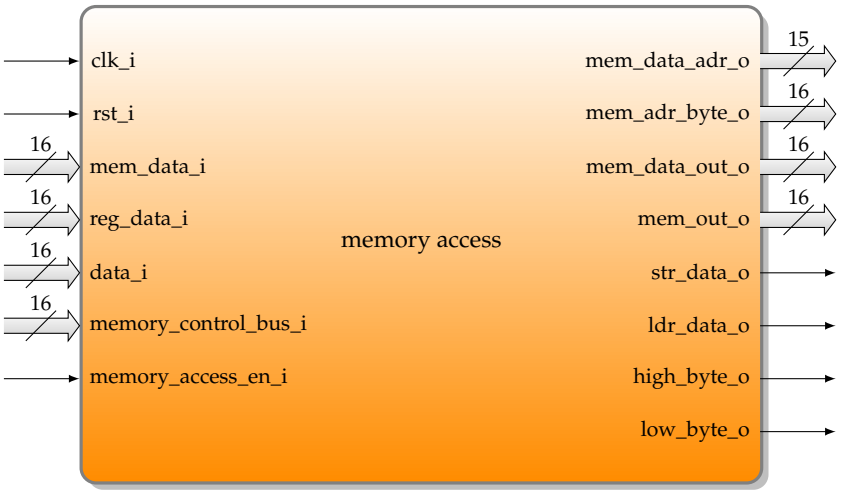
1.7 Implémentation du bloc mult_2

Remarque : Analyser et tester.

Cette partie n'a pas besoin d'être modifiée car elle a déjà réalisée mais il est important de comprendre que l'on doit donner une valeur cohérente en sortie lorsque l'on appelle la fonctionnalité de ce bloc. Testez le bloc.

2 MEMORY ACCESS

Entité du bloc MEMORY ACCESS



Nom I/O	Description
clk_i	Entrée d'horloge
rst_i	Entrée du reset asynchrone
mem_data_i	Donnée qui arrivent de la mémoire de données
reg_data_i	Donnée du registre choisi par sel_mem (Voir labo decode)
data_i	Donnée qui arrive depuis le bloc execute
memory_control_bus_i	Bus de contrôle du bloc Memory Access
memory_access_en_i	Signal enable du bloc Memory Access
mem_data_adr_o	Adresse dans la mémoire de données
mem_adr_byte_o	Adresse brute (uniquement pour DEBUG)
mem_data_out_o	Donnée qui part vers la mémoire de données
mem_out_o	Valeur de sortie qui va dans la banque de registres
str_data_o	Flag qui indique qu'on veut stocker une donnée en mémoire
ldr_data_o	Flag qui indique qu'on veut lire une donnée en mémoire
high_byte_o	Lors d'une écriture par byte on veut accéder la partie haute de la mémoire
low_byte_o	Lors d'une écriture par byte on veut accéder la partie basse de la mémoire

Construction du bus de contrôle du bloc memory access :

Position	Taille	Description
0	1	Flag qui indique que l'opération courante utilise memory access
1	1	Flag qui indique un stockage de la valeur en mémoire
2	1	Flag qui indique une lecture de la valeur en mémoire
3	1	Flag qui indique une opération en byte (halfword sinon)
4	1	Pas utilisé

2.1 Implémentation du bloc memory access

Remarque : Analyser et tester.

Cette partie n'a pas besoin d'être modifiée car elle a déjà réalisée mais il est important de **tester** que tout se passe bien avec le reste du design.

2.2 Test des instructions d'accès à la mémoire de données

Implémentez les instructions qui accèdent à la mémoire afin de tester son bon fonctionnement.
Il est important de valider que les valeurs que l'on souhaite accéder transitent correctement au travers du design.
Par exemple, la valeur du registre à lire/écrire arrive sur la bonne connexion et les deux autres valeurs de registres forment bien la bonne adresse source/destination.
Vous devez indiquer les instructions que vous allez utiliser pour faire les accès à la mémoire de données.

3 Programme complet pour le processeur que vous avez réalisé

3.1 La stack

Maintenant que vous avez testé les diverses fonctionnalités de votre processeur et que tout semble fonctionner, il faut implémenter une **stack** pour compléter l'ensemble des fonctionnalités. Comme notre processeur ne supporte pas les instructions *push* ou *pop*, il faut réaliser ces instructions autrement en découpant la séquence avec plusieurs instructions pour stocker les informations.

Indice 1 : L'instruction BL stocke dans le *LR*, l'information de l'adresse à laquelle revenir après un appel à fonction.

Indice 2 : Le registre SP est censé mémoriser l'adresse courante du sommet de la stack.

Indice 3 : regarder les numéros de registres dans les laboratoires précédents.

3.2 Programme complet

Faire un programme qui permette de faire au moins 8 sauts imbriqués et de revenir au point de départ. Après avoir réalisé les sauts imbriqués, utiliser des sauts conditionnels et inconditionnels en plus des instructions de calcul afin de vérifier que tout fonctionne.

Rendu

Pour ce laboratoire, vous devez rendre :

— votre fichier *.circ*

— votre programme *main.S* contenant le code de test que vous avez utilisé pour tester votre circuit.

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.