CY Cergy Paris Université

RAPPORT

pour le projet d'Architecture des ordinateurs
Licence d'Informatique troisième année

sur le sujet

Processeur en VHDL

rédigé par

DUCLY Hugo, CAMARA Almamy

Avril 2020

Contents

1 Introduction **2**
1.1 Objectifs 2

2 Cahier des charges **2**

3 Instructions **3**

4 Rôle des composants **5**

5 Réalisation des composants **8**

6 Processeur et système complet **9**

7 Tests du système **10**

8 Organisation **12**

9 Problèmes rencontrés **12**

10 Conclusion **12**

1 Introduction

1.1 Objectifs

Ce projet est réalisé dans le cadre de la troisième année de licence en informatique à l'Université de Cergy-Pontoise en cours d'architecture des ordinateurs. Il a pour but de réaliser un processeur en langage VHDL en utilisant les différentes notions vues en cours. Ce projet est réalisé sous la tutelle de J.Lorandel, professeur de l'Université de Cergy-Pontoise. Pour ce projet nous utilisons le logiciel Quartus pour la conception et le test du CPU en VHDL.

2 Cahier des charges

L'objectif est de créer un processeur en assemblant divers éléments réalisés en VHDL à l'aide du logiciel Quartus. Les principaux éléments à réaliser sont les suivants :

- Unité Arithmétique et Logique (ALU)
- Banc de 8 registres généraux
- Unité de contrôle

Diverses extensions devaient être réalisées durant ce projet. Nous avons choisies de réaliser :

- Ajout d'une mémoire d'instructions, connectée à l'entrée Din du processeur
- Ajout des opérations logiques ET, OU et NOT instructions réalisables par le processeur

3 Instructions

Notre processeur peut réaliser un jeu d'instructions composé des instructions suivantes :

- MOV - permet de passer une donnée d'un registre à un autre
- MOVI - permet de stocker une donnée de l'extérieur dans un registre
- ADD - permet d'addtionner les valeurs de deux registres et de placer le résultat dans le premier
- SUB - permet de soustraire les valeurs de deux registres et de placer le résultat dans le premier
- AND - permet de réaliser un ET logique sur les valeurs de deux registres et de placer le résultat dans le premier
- OR - permet de réaliser un OU logique sur les valeurs de deux registres et de placer le résultat dans le premier
- NOT - permet d'inverser logiquement la valeur d'un registre et d'actualiser ce registre avec le résultat

Les instructions sont transmises au registre d'instruction lorsque le processeur est dans l'état Fetch (recherche d'instruction). Elles sont codées sur 9 bits da la façon suivante :

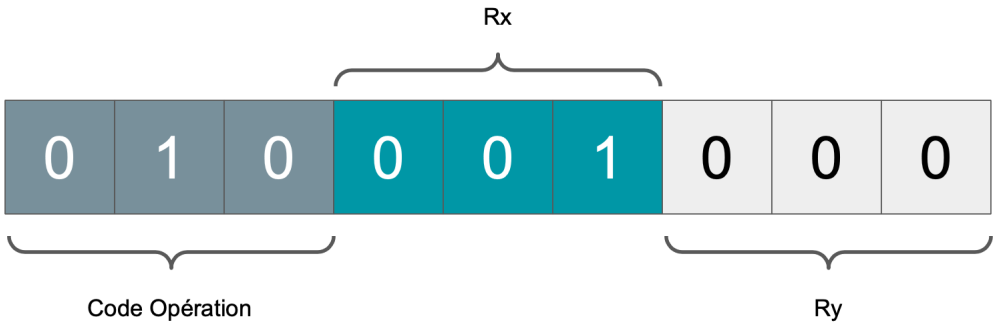

Figure 1: Structure d'une instruction

Le **Code opération** est sur 3 bits et permet de dire aux processeur quelle opération il devra effectuer. Ce code est utilisé dans la machine à état du contrôleur pour savoir quelle intruction lancer. **Rx** et **Ry**, sur 3 bits aussi sont les adresses des registres dont on veut utiliser les valeurs pour réaliser l'instruction. Les codes des opérations citées précédemment sont les suivants :

Opération	Code opératoire
MOV	000
MOVI	001
ADD	010
SUB	011
AND	100
OR	101
NOT	110

Figure 2: Codes opératoires des instructions du processeur

Certaines instructions, telles que MOV et MOVI, ne vont nécessiter qu'un seul cycle du processeur, et d'autres plusieurs. Voici un tableau récapitulatif des actions à effectuer pour chaque cycle d'horloge pour chaque instruction.

Opération	Cycle 1	Cycle 2	Cycle 3
MOV Rx Ry	Lecture de Ry Ecriture sur Rx		
MOVI Rx	Lecture de Din Ecriture sur Rx		
ADD Rx Ry	Lecture de Rx Ecriture de A	Lecture de Ry Ecriture de G	Lecture de G Ecriture de Rx
SUB Rx Ry	Lecture de Rx Ecriture de A	Lecture de Ry Ecriture de G	Lecture de G Ecriture de Rx
AND Rx Ry	Lecture de Rx Ecriture de A	Lecture de Ry Ecriture de G	Lecture de G Ecriture de Rx
OR Rx Ry	Lecture de Rx Ecriture de A	Lecture de Ry Ecriture de G	Lecture de G Ecriture de Rx
NOT Rx	Lecture de Rx Ecriture de A	Lecture de G Ecriture de Rx	

Figure 3: Réalisation des instructions du processeur

4 Rôle des composants

Dans cette partie, les trois composants principaux seront détaillés selon leur utilité et la façon de les réaliser.

ALU Cette unité est en charge de réaliser les calculs. Elle prend en entrées deux mots, A et B de 16 bits chacun. L'énoncé imposait deux opérations de base :

- ADD
- SUB

que nous aggrémentons comme extension de trois autres opérations logiques qui sont :

- OR
- AND
- NOT

Ces opérations seront détaillées dans la partie Instructions.

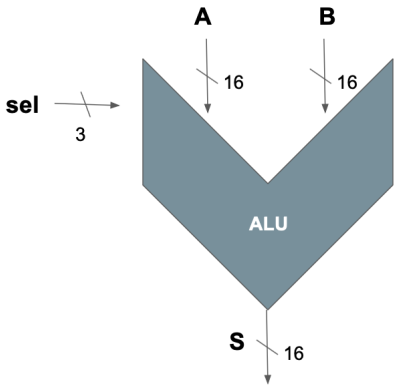

Figure 4: Structure ALU

- A : entrée du premier opérande
- B : entrée du second opérande
- sel : permet de choisir le type d'opération a effectuer par l'unité.
- S : sortie de l'unité qui renvoit le résultat de l'opération.

Banc de registres Il est composé de huit registres qui vont servir à stocker temporairement les données provenant de l'extérieur (mémoire). Ils servent aussi à stocker les résultats de l'ALU. C'est sur l'entrée Enable que pourront intervenir les signaux de contrôle émanant du contrôleur

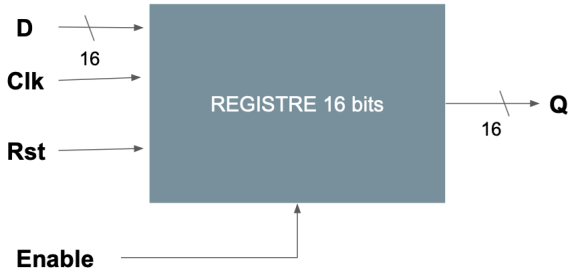

Figure 5: Structure Registre

- D : entrée de la donnée du registre
- Clk : entrée d'horloge
- Rst : entrée qui permet de réinitialiser le registre
- Enable : entrée qui autorise le registre à stocker la donnée située à son entrée dans sa sortie.
- Q : sortie du registre

Multiplexeur 10 vers 1 Ce composant est essentiel car il permet de placer la donnée dont on a besoin sur le bus de données. En effet, le contrôleur, via une sortie, communique au multiplexeur la donnée souhaitée. Les huit premières valeurs concernent les registres R0 à R7 (du banc de registres) , la neuvième est réservée à l'entrée du processeur et la dernière concernce la sortie du registre G (sortie de L'ALU).

Mémoire Ce composant va permettre de lire dans un fichier d'extension MIF (memory input file) toutes les entrées que l'on souhaite envoyer au processeur. Au niveau pratique, cela évite de devoir tapper à la main chaque opération que l'on souhaite dans le testbench. Et si l'on avait pu tester le processeur sur un FPGA, cela nous aurait éviter de manipuler les switchs (qui peuvent être la source de nombreux problèmes, notamment d'activation non volontaire) manuellement. Ce composant est lui-même constitué de deux choses :

- Compteur : qui est paramétré de façon à se remettre à zéro arrivé à 10 (c'est le nombre de cases que nous avons choisi pour notre fichier MIF).
- ROM : qui permet de lire un fichier MIF à l'adresse qu'on lui passe en entrée

Ces composants sont reliés à la même horloge, qui sera différente de celle de tous les autres composants du processeur.

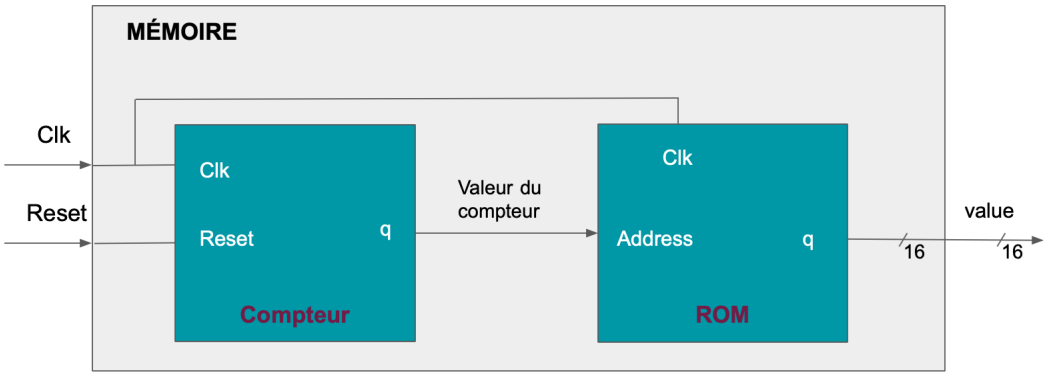

Figure 6: Structure Mémoire

- Clk : Entrée d'horloge de la mémoire
- Reset : Entrée qui permet de réinitialiser le compteur
- Value : Sortie qui vaut la valeur de la case mémoire située à l'adresse à l'entrée de la ROM

Unité de contrôle L'unité de contrôle est l'unité principale du CPU qui reçoit les instructions et pilote le chemin des données dans le reste du CPU grâce au pilotage de signaux de contrôle.

- IR : entrée de l'instruction, provenant du registre d'instruction
- Run : entrée permettant de lancer l'exécution de l'instruction en cours
- Reset : Remet le contrôleur dans son état initial
- IRs : signal qui permet au registre d'instruction de stocker la donnée venant de l'entrée
- BusSel : sortie qui permet, selon l'action à réaliser, de choisir quelle entrée du multiplexeur sera présente sur le bus de données.

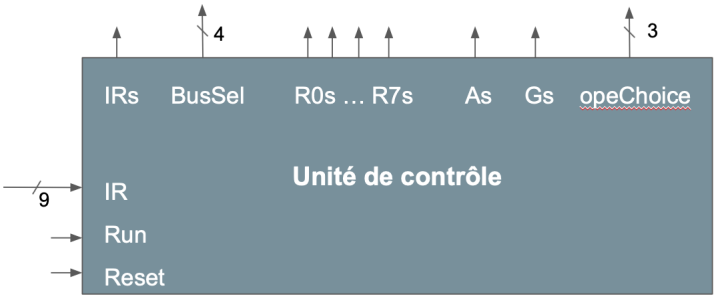

Figure 7: Structure Unité de contrôle

- R0s,...,R1s : signaux de contrôle permettant aux registres du banc de registre de stocker la donnée présente sur le bus de données.
- As : signal permettant au registre A - qui contient un des deux opérandes de l'ALU - de stocker la donnée du bus de données.
- Gs : Idem que pour As, mais G est le registre contenant le résultat de l'opération de l'ALU.
- opeChoice : sortie qui permet de transmettre à l'ALU le type d'opération que l'on souhaite.

5 Réalisation des composants

La majeure partie de la réalisation des composants du processeur détaillés précédemment a été faite en amont et ne comporte aucun points particuliers. L'essentiel de la complexité s'est trouvée être au niveau du contrôleur.

Pour le modéliser nous avons choisi une machine à états finis de type Moore. Les états que nous avons utilisés sont les suivants :

- Fetch : c'est l'état de base, durant lequel on recherche l'instruction qui sera réalisée.
- Decode : cet état permet de décoder l'instruction qui a été récupéré
- Tous les autres états concernent les différentes étapes d'exécution des opérations possibles pour le processeur. Il vont être détaillés dans le schéma ci-après.

Le schéma ne contient que les quatre opérations de base car les opérations en extensions, qui sont les opérations logiques suivent exactement le même schéma, excepté le code de l'opération qui sera différent.

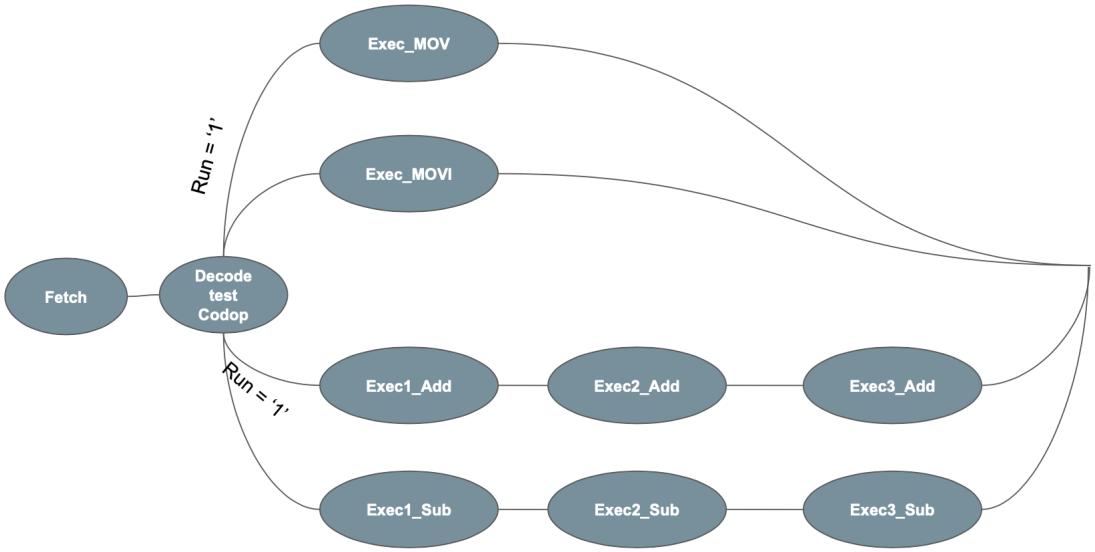

Figure 8: FSM Contrôleur

Les traits qui se rassemblent à la fin signifient que l'on retourne à l'état Fetch pour y attendre une nouvelle instruction.

Au niveau de la question de la réalisation en VHDL de la machine à états, nous avons choisi une représentation en trois PROCESS qui est la manière la plus simple et intuitive de représenter une FSM.

- Le premier PROCESS consiste à déterminer l'état futur à partir de l'état présent et des entrées, entre autre l'entré Run, détaillée précédemment. C'est dans ce process que l'on représente la structure même du schéma.
- Le second PROCESS consiste à déterminer les sorties selon l'état présent. Ici les sorties sont les différents signaux de contrôle du contrôleur, ainsi que le selecteur relié au multiplexeur. Les différents états des sorties seront détaillés dans un tableau dans la prochaine partie.
- Le troisième et dernier PROCESS permet de passer à l'état suivant à chaque front montant de l'horloge. C'est aussi dans celui-ci qu'est placé le reset asynchrone de la FSM.

6 Processeur et système complet

Tous les composants détaillés précédemment permettent, une fois assemblés, d'obtenir un processeur capable de réaliser les opérations décrites dans la partie instructions.

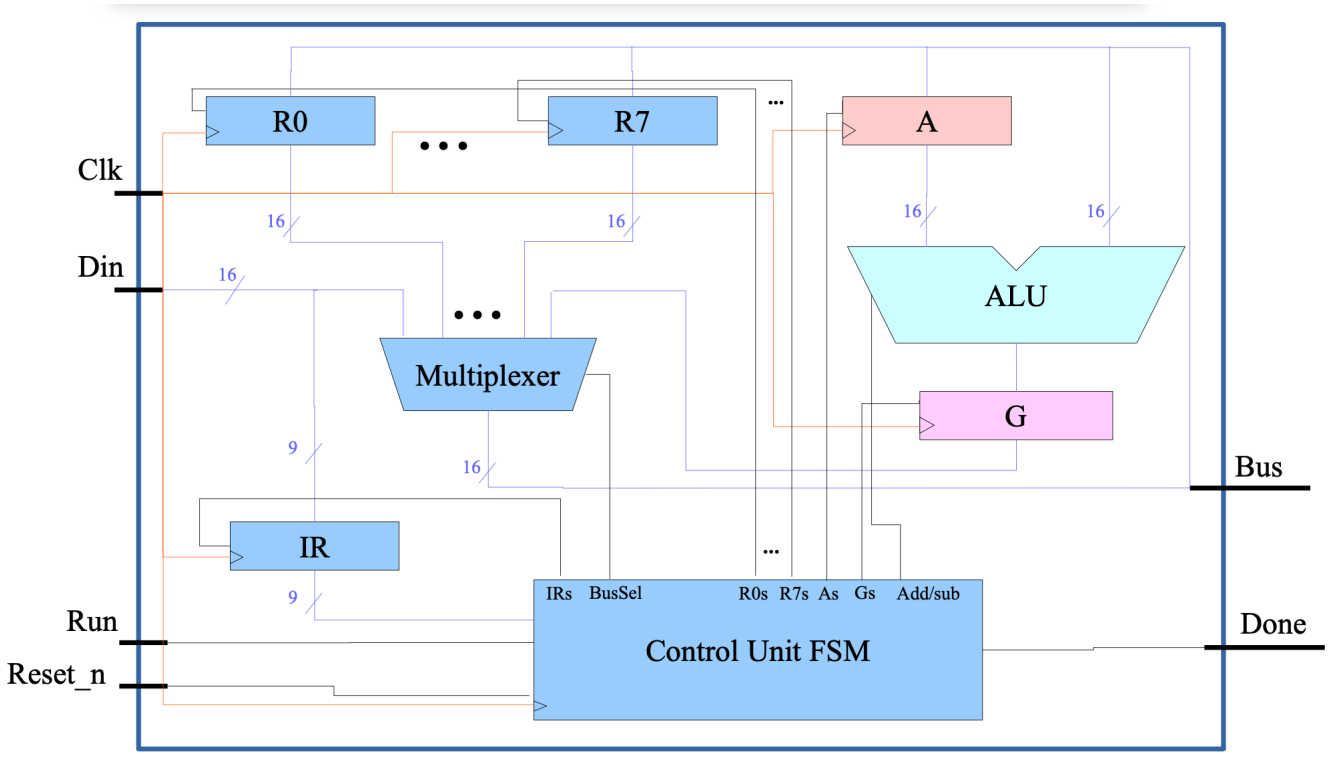

Figure 9: Structure du processeur - Schéma tiré du cours d'Architecture des ordinateurs intitulé "Séance 5 : Projet" réalisé par Jordane Lorandel, MCF ETIS-ENSEA-CYU

Une fois la mémoire ajoutée, on obtient le système suivant :

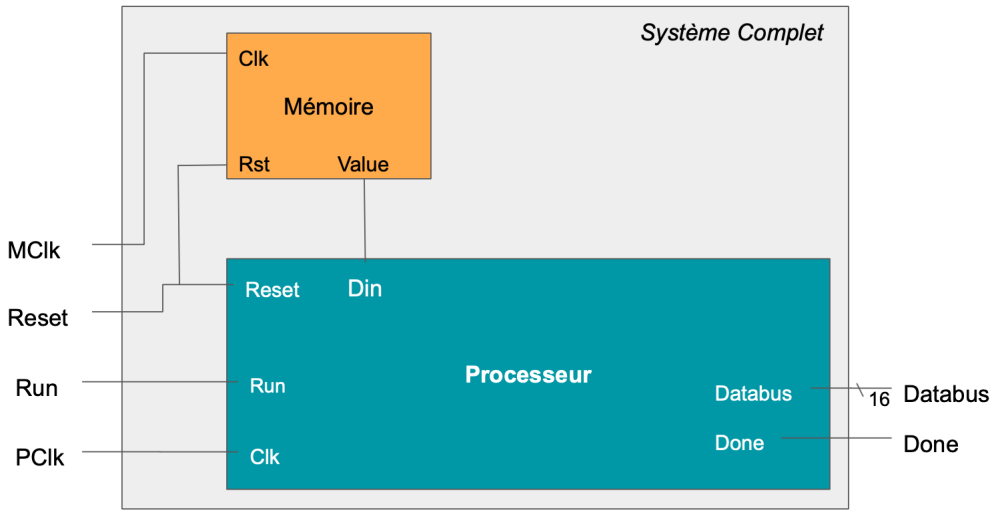

Figure 10: Structure du système complet

7 Tests du système

Des tests ont été pratiqué tout le long de la réalisation du projet. En effet, durant les séances de cours, nous pouvions tester les composants directement sur le FPGA en attribuant aux entrées et sorties de nos composants des switchs, leds et autres afficheurs 7-segments. Pour les tests de l'ALU, du processeur et de la mémoire, des testbenchs on été utilisé. Ces fichiers permettent d'utiliser l'outil ModelSim, via l'option tools -> RTL Simulation de Quartus. Il suffit, au lieu d'attribuer des éléments physiques, d'attribuer aux différentes entrées et sorties des signaux, auxquels on donne des valeurs dans le testbench. Une fois le testbench correctement compilé, on peut lancer la simulation pour une durée définie et observer les valeurs de tous nos signaux via la fenêtre Wave. Ainsi ci-après se trouvent quelques tests parmi tous ceux que nous avons pratiqué, juste pour illustrer cette partie.

Test du contrôleur Tout d'abord nous avons testé le contrôleur.

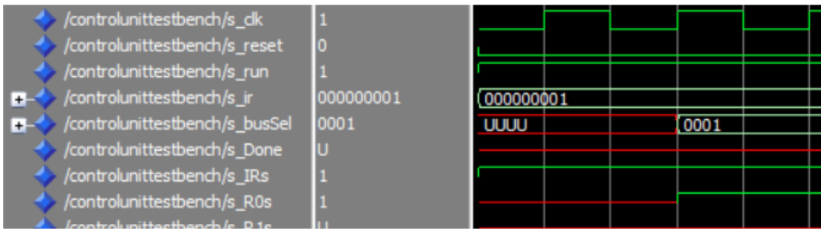

Figure 11: Résultat Simulation 1

On peut voir que l'opération a effectué est un déplacement simple de type MOVE et que l'on voulait déplacer la donnée du registre R1 dans le registre R0. Donc le contrôleur a activé le multiplexeur pour récupérer valeur de R1 (sbusSel = 0001) et a aussi activer R0s (sR0s = 1) pour stocker la valeur dans R0.

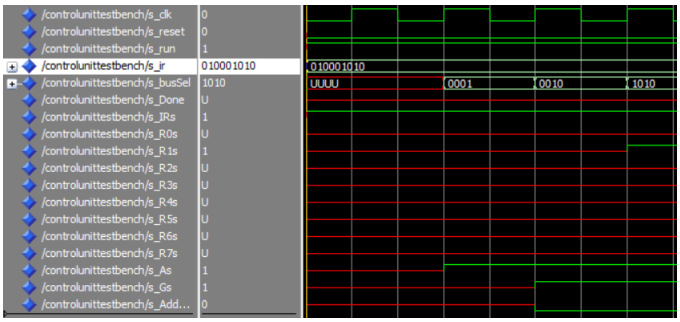

Figure 12: Résultat Simulation 2

Ici le test était une addition de type ADD pour les registres R1 et R2. On peut donc d'abord voir que le bus selectionne R1 pour mettre sa valeur dans A puis qu'il sélectionne R2 pour mettre la valeur de R1 + R2 dans G, ensuite le bus sélectionne G pour mettre sa valeur dans R1

Test du processeur On a ensuite testé le processeur

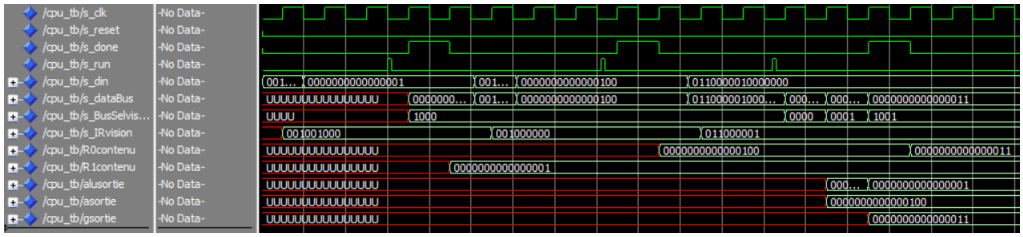

Figure 13: Résultat Simulation 3

Le résultat du test ci-dessus concerne deux actions de type MOVEI pour mettre la valeur 1 dans R1 puis 4 dans R0. La dernière instruction est un type SUB, et on retrouve bien à la fin la valeur 3 (4 - 1) dans R0.

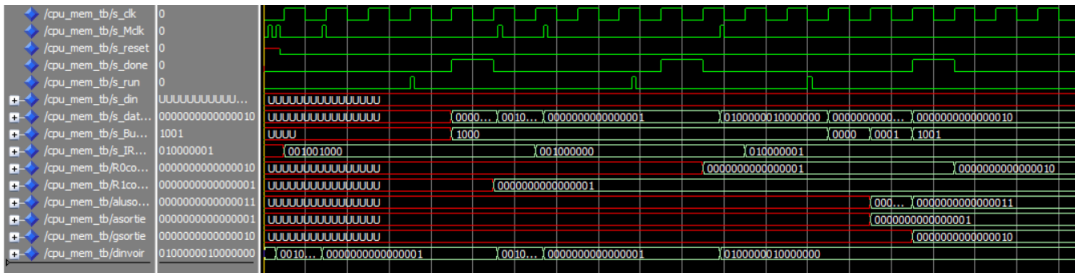

Figure 14: Résultat Simulation 3

Le résultat du test ci-dessus concerne deux actions de type MOVEI pour mettre la valeur 1 dans R1 puis 4 dans R0. La dernière instruction est un type SUB, et on retrouve bien à la fin la valeur 3 (4 - 1) dans R0.

8 Organisation

L'organisation du travail s'est faite de manière équitable. La version actuelle était gardée à jour via Google Drive et le rapport a été tappé via Overleaf, un site dédié à la rédaction en LaTex sur lequel on peut collaborer sur un même projet.

9 Problèmes rencontrés

Les problèmes rencontrés lors de ce projet furent surtout des erreurs d'étourderie pour certains composants et parfois un manque de tests. C'est ce qui nous a ammené à créer des testbench pour quasiment tous les composants, afin d'éliminer une à une les sources d'erreur potentielles. L'erreur d'étourderie la plus commise a été d'oublier l'ajout de certains signaux à la liste de sensibilité des process. Et pour le contrôleur, nous avions oublier de désactiver les registres après les avoir activé, ce qui fait qu'il changeait de valeur sans respecter ce qui était prévu. Mais nous avons su régler tous ces problèmes.

10 Conclusion

En conclusion, nous avons réalisé un processeur en VHDL fonctionnel et nous avons respecté le cahier des charges. Afin d'améliorer notre processeur, nous pourrions ajouter d'opérations logique dans l'ALU comme les opérations XOR et NOR, par exemple.