RAPPORT

Tronc commun EIDIA
Systèmes embarqués

Processeur 8 bits avec langage VHDL.

2022-2023

Réalisé par:

MEZZOUR Omar
AIT OUBRAHIM Hassna
OUCHKER Mohamed Ayman

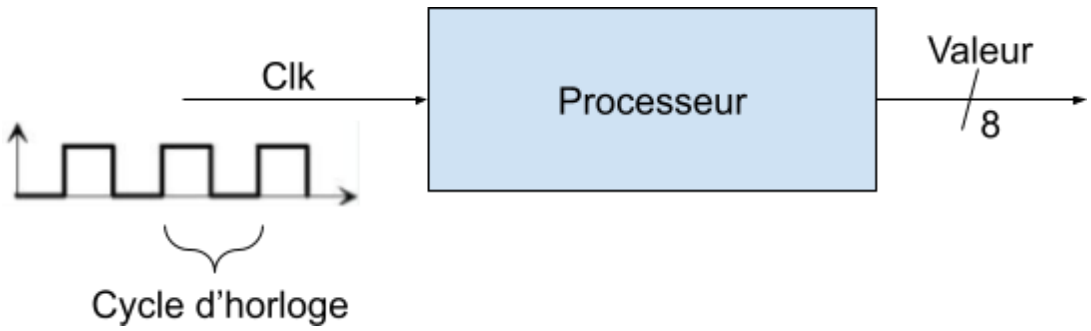
Encadré par:
Monsieur le professeur :
Dr Alae ammour

I. Introduction

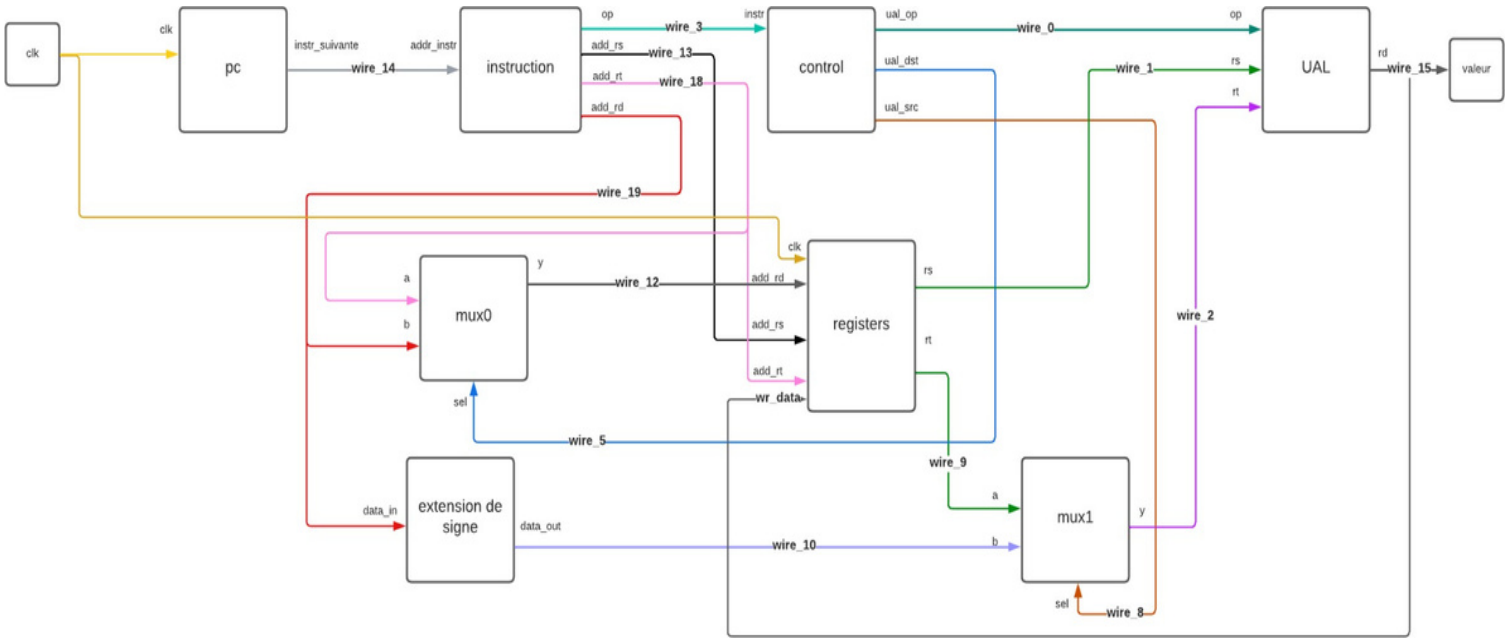
L'enjeu de ce projet est de concevoir et de réaliser un processeur 8 bits sous langage VHDL. Pour se faire plusieurs étapes ont été suivies allant de la conception de l'architecture globale du projet jusqu'à la réalisation de la simulation du comportement du processeur par le biais de Modelsim afin de valider son comportement et l'ensemble des opérations réalisées.

II. Fonctionnement et Architecture globale

Le processeur 8 bits permettra d'obtenir une valeur en sortie codée sur 8 bits pendant chaque cycle d'horloge à l'aide des différentes instructions effectuées par différents blocs qui seront présentés par la suite.

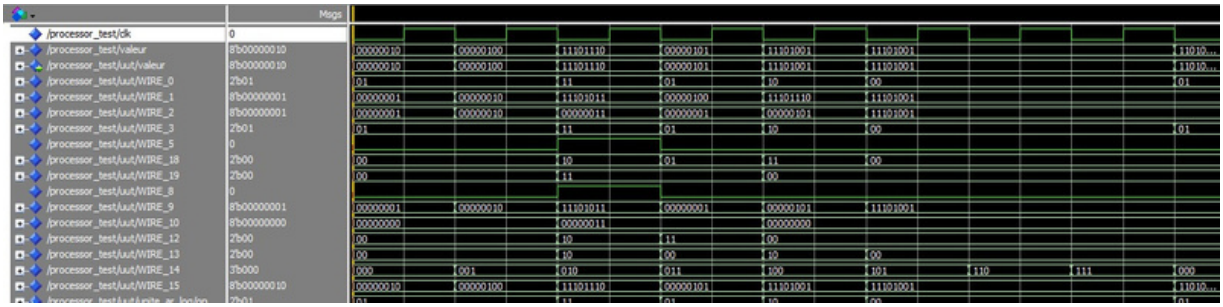


Le schéma suivant représente l'architecture globale du processeur 8 bits dont l'entrée est l'horloge clk et la sortie est valeur.



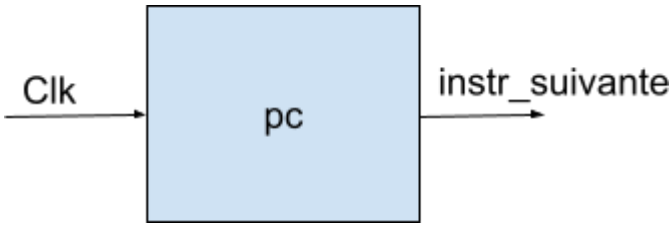
III. Simulation du comportement du processeur 8 bits

Le schéma suivant montre les résultats trouvés lors de la simulation du comportement du processeur 8 bits réalisé, dont l'architecture globale est représentée ci-dessus.
On représente la variation de l'entrée CLk, la sortie valeur et les signaux internes qui servent comme entrées/sorties des blocs internes du processeur.
Les résultats de la simulation seront interprétés dans la suite du rapport.

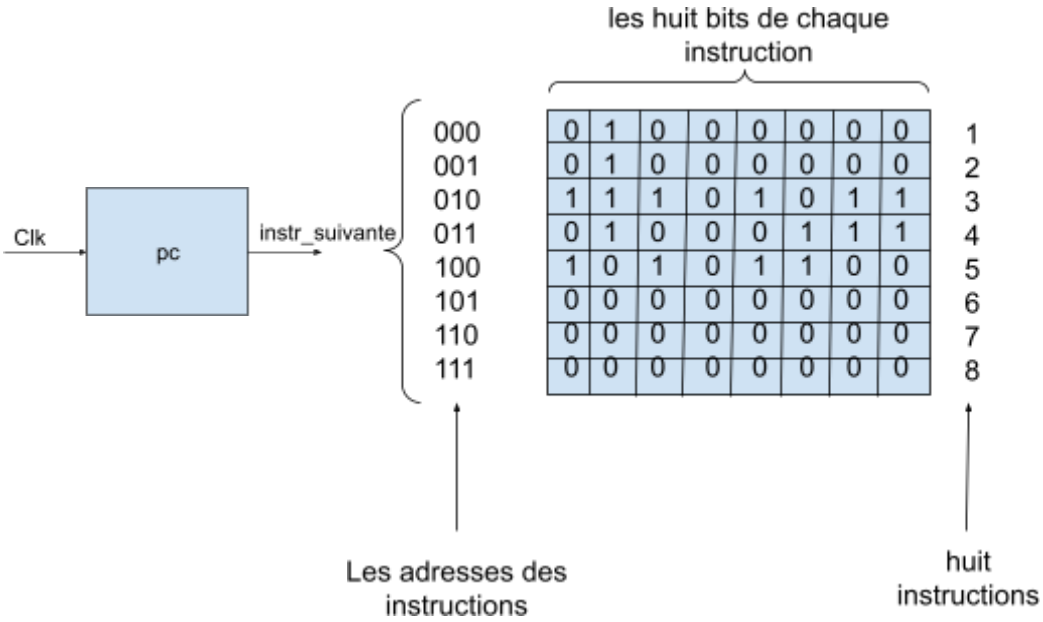


IV. Composants de l'architecture

a) Program Counter : pc



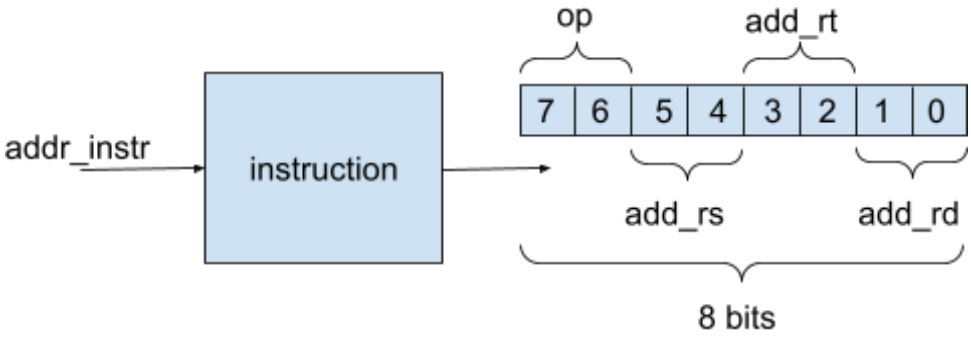
La sortie de ce bloc sert comme entrée au bloc "instruction", ce dernier stocke huit instructions qui doivent s'exécuter c'est pourquoi on a besoin du bloc "pc" pour attribuer une adresse à chaque instruction, donc intr_suivante est codée sur 3 bits.



Ce composant permet de compter les cycles d'horloge en incrémentant la valeur du compteur par un à la fin de chaque cycle. Il s'agit d'un compteur modulo 8 qui se réinitialise dès qu'il arrive à 7.
En effet, lorsque la valeur du compteur change, le processeur entame la nouvelle instruction pour calculer la nouvelle valeur de sortie.
Le résultat de la simulation wire_14 donne le résultat attendu.

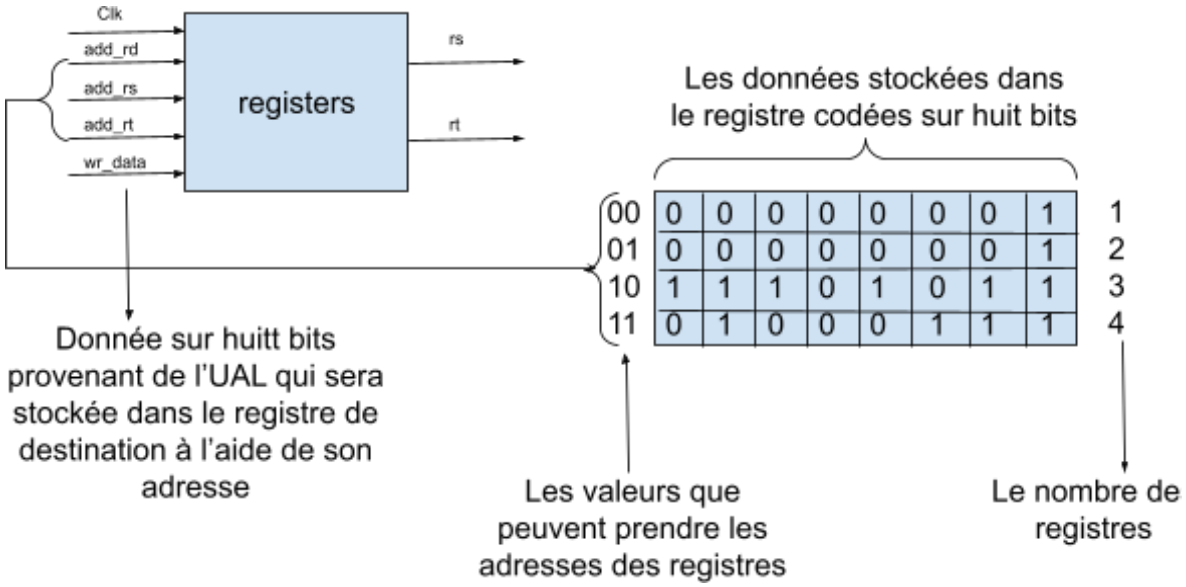
b) instruction

A la sortie du bloc "pc" on récupère l'adresse de l'instruction mémoire. Comme c'est expliqué auparavant.
Par la suite, on récupère à la sortie du composant "instruction" l'instruction codée sur huit bits qui servira à connaître l'opération et l'adresse des registres sources contenant les données sur lesquelles l'opération va s'exécuter (add_rs et add_rt) ainsi que l'adresse du registre de destination stockant le résultat de l'opération (add_rd). Chacun de ces éléments est codé sur deux bits.



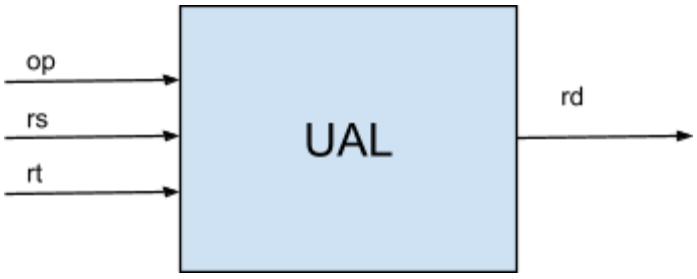
c) registers

Ce bloc permettra de récupérer du registerfile (quatre registres puisque les adresses sont codées sur deux bits) la donnée codée sur huit bits correspondante à l'adresse fournie par le bloc "instruction". Ainsi à l'aide des adresses des registres add_rs et add_rt on récupère les valeurs de rs et rt.
Il permet également de mettre à jour la valeur du registre de destination pendant chaque cycle d'horloge. En effet, après le calcul effectué par l'unité arithmétique et logique, le résultat wr_data est stocké dans le registre correspondant à l'adresse rd_addr.



Pour vérifier ces résultats, on s'intéresse aux signaux wire_1, wire_9, wire_3, wire_13, wire_18 et wire_19 dans la simulation.
En effet, pour le premier cycle d'horloge, la valeur de sortie du bloc "pc" instr_suivante vaut 000 ce qui correspond à l'adresse mémoire de la valeur 01000000 stockée dans "instruction". Donc le code de l'opération est 01 (voir wire_3) et la valeur des adresses add_rs, add_rt et add_rd est 00 (voir wire_13, wire_18 et wire_19 respectivement).
Comme le registre correspondant à l'adresse 00 est le premier registre, alors la valeur de rs et rt est 00000001. C'est ce qu'on peut vérifier en regardant les signaux wire_1 et wire_9 respectivement. Même chose pour les autres cycles.
Pour le troisième cycle d'horloge par exemple: la valeur de instr_suivante est 010 (2 sur 3 bits, voir wire_14). 010 correspond à la troisième instruction dans la mémoire qui est 11101011. Donc 11, 10, 10, 11 sont respectivement les valeurs op, add_rs, add_rt et add_rd.
Le wire_3 représente le code de l'opération qui est 11, le wire_13 représente la valeur de add_rs qui vaut 10, le wire_18 représente l'adresse add_rt qui vaut 10 et le wire_19 représente la valeur de add_rd qui vaut 11.
Pour vérifier les valeurs de rs et rt on vérifie les signaux wire_1 et wire_9, on trouve 11101011 pour rs et rt. Ce qui correspond bien à la valeur du troisième registre dont l'adresse est 10. On déduit que les résultats sont conformes.

d) UAL: unité arithmétique et logique



Maintenant qu'on a récupéré les valeurs rs et rt des registres on effectue des opérations sur ces données à l'aide du bloc UAL: unité arithmétique et logique.
Comme l'opération op est codée sur 2 bits on peut réaliser 4 opérations distinctes suivant le code de l'opération.
On choisit de réaliser les opérations d'addition (lorsque op = "01"), de soustraction (lorsque op = "10"), le ET logique (lorsque op = "00"), le OU logique (lorsque op = "11").
On décide de remplacer l'opération de OU logique par une autre opération ADDI afin de mettre en oeuvre des blocs davantage: unité de contrôle, deux multiplexeurs et extension de signe (voir après).
En effet, l'opération addi est une addition mais qui consiste à faire une substitution des membres de l'opération; c'est à dire au lieu de faire A+B et stocker le résultat dans le registre de destination d'adresse add_rd, on fait A+add_rd et on stocke le résultat dans le le deuxième registre (registre contenant B).
On vérifie à ce stade le fonctionnement des trois premières opérations et on laisse addi dans la suite du rapport.
Pour cela, on s'intéresse aux signaux wire_0, wire_1, wire_2 et wire_15.
Pendant le premier cycle d'horloge, le code de l'opération est "01" (voir wire_0) donc il s'agit d'une addition entre rs="00000001" (voir wire_1) et rt="00000001" (voir wire_2).
Le résultat "rd" sur wire_15, qui correspond à la valeur prise par la sortie valeur du processeur est: "00000010". Ce qui est conforme au résultat attendu.
On peut également vérifier le bon fonctionnement des opérations AND et soustraction en regardant le 6ème et 5ème cycle d'horloge respectivement.

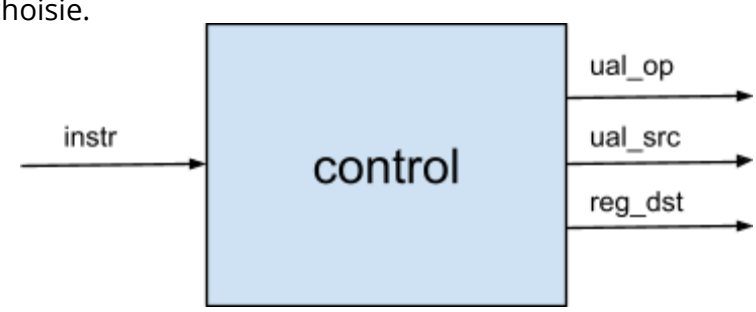
e) Control

L'intérêt de ce bloc ainsi que la suite des blocs ci-dessous et d'assurer le

fonctionnement de l'opération ADDI expliquée auparavant: valeur du premier registre

+ adresse du registre de destination et stocker le résultat dans le deuxième registre.
En effet, le bloc s'agit d'une unité de contrôle qui intervient avant l'unité arithmétique et logique afin de bien fournir à cette dernière les éléments convenables et corrects pour

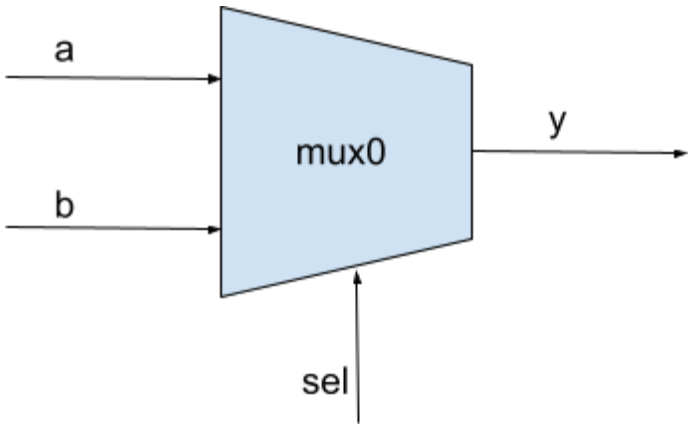
faire l'opération choisie.



L'entrée instr correspond à la sortie du bloc "instruction", qui correspond au code de l'opération à exécuter.
La sortie ual_op correspond exactement au code de l'opération reçu en entrée et sera transmis au bloc UAL (wire_0 vaut wire_3 tout le temps).

La sortie ual_src est la ligne de sélection du multiplexeur 1 qui permet de choisir quelle donnée à sommer avec la valeur du premier registre (donnée “A”). Elle est mise à 1 lorsque l’opération est “11” (qui correspond à addi).
Sur le schéma de la simulation wire_8 vaut 1 lorsque cette condition est vérifiée.Sinon elle vaut 0.
La sortie ual_dst est la ligne de sélection du multiplexeur 0 qui permet de choisir quelle sera le registre de destination pour stocker le résultat de l’opération. Elle est mise à 1 lorsque l’opération est “11” (qui correspond à addi).
Sur le schéma de la simulation wire_5 vaut 1 lorsque cette condition est vérifiée.Sinon elle vaut 0.

f) mux0



L’entrée “a” correspond à la sortie add_rt du bloc “instruction” et l’entrée b correspond à la sortie add-rd qui est l’adresse du registre de destination par défaut.
La sortie y est reliée à l’entrée add_rd du bloc “registers” pour indiquer si le registre de destination est celui par défaut ou celui qui contient la donnée “B” de l’addition, c'est-à- dire le deuxième registre d’adresse add_rt.

Pour résumer, ce bloc permet de choisir le registre de destination qui stockera la valeur du résultat de l’opération. Si l’opération est addi, la ligne de sélection sel va prendre la valeur 1 et le registre de destination sera le deuxième registre contenant la valeur de B. Sinon c’est le registre de destination par défaut.
Pour vérifier ce comportement, on s'intéresse aux signaux wire_5, wire_12, wire_18 et wire_19.

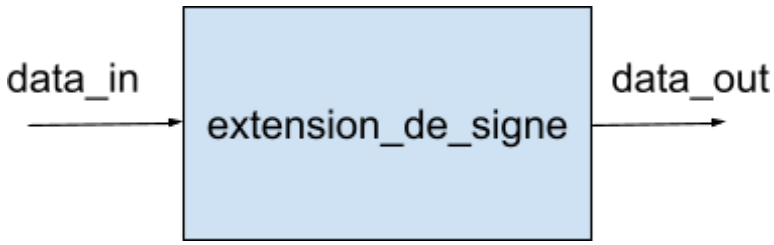
Au cours du cinquième cycle d’horloge, a= “11” (wire_18), b=”00” (wire_19) et l’opération est “10” donc différente de “11” (addi). Par suite, la ligne de sélection vaut 0 (voir wire_5). Comme sel=’0’ alors le registre de destination est le registre de destination par défaut d’adresse add_rd, qui est sur l’entrée b du multiplexeur. En regardant la sortie y sur le wire_12, qui doit être égale à b dans ce cas, on trouve qu’elle égale à “00”. Ce qui est bien attendu.
On fait la même chose pour le troisième cycle, la valeur de a=”10” ,b=”11”. L’opération vaut “11” donc il s’agit de l’opération addi. Par suite, la valeur de la ligne de sélection est 1, et la sortie y doit être égale à “a”, on trouve bien que le wire_12 vaut “10”.

En conclusion on réussit à l'aide de mux0 de changer le registre de destination de celui par défaut au registre 2.

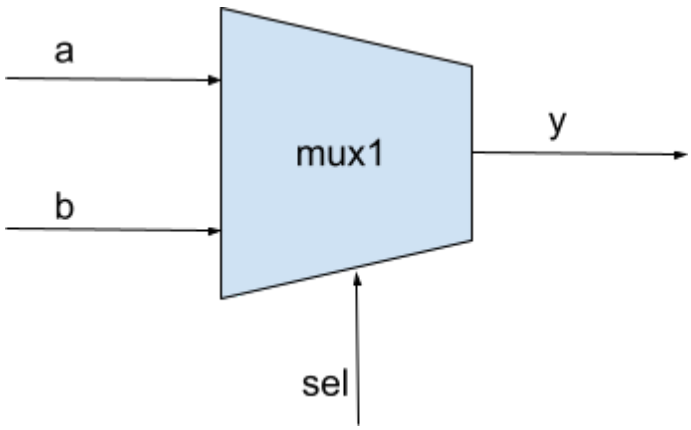
g) extension de signe

Ce bloc permet de convertir la donnée d'entrée data_in sur 2 bits en une donnée data_out sur 8 bits en ajoutant des 0 à gauche de la valeur data_in.

En effet, pour assurer l'opération addi, on a besoin de convertir l'adresse add_rd reçue en entrée sur 8 bits afin de réussir à la sommer avec la valeur du registre 1 qui est codée sur 8 bits.
Pour visualiser le comportement de ce bloc, il suffit de regarder la valeur d'entrée sur wire_19 et la valeur de sortie sur wire_10.



h) mux1



Maintenant que le multiplexeur 0 indique quel sera le registre de destination pour stocker le résultat de l'opération.
Le multiplexeur 1 permet à l'aide de la ligne de sélection de choisir le deuxième membre de l'addition. En effet, lorsque l'opération vaut "11" c'est à dire qu'on est face à l'opération addi sel vaut 1 et le signal y de sortie sera égal au signal b d'entrée.
Ce dernier est relié à la sortie du bloc "extension_de_signe" pour assurer la compatibilité de l'addition sur 8 bits.
Sinon lorsque sel vaut 0 c'est à dire que l'opération est autre que addi le signal de sortie bascule sur l'entrée a qui correspond à la valeur du deuxième registre (donnée B).

Le signal de sortie y est servi comme entrée du bloc UAL pour effectuer l'opération souhaitée. Ainsi, on assure que ce bloc reçoit les données convenables, valeurs du registre 1 et registre 2 dans le cas de ET logique, soustraction et addition. et valeur du premier registre et adresse du deuxième registre convertis sur 8 bits par le bloc extension_de_signe dans le cas de l'opération addi.
Pour vérifier le comportement du bloc mux1 on s'intéresse aux signaux wire_2, wire_8, wire_9 et wire_10. On remarque bien que wire_2 (sortie y) prend la valeur de wire_9 (entrée a) lorsque wire_8(sel) vaut 0 et bascule sur le wire_10 (entrée b) dans le cas contraire.
Maintenant que tous les blocs sont expliqués, il reste à vérifier le bon fonctionnement de l'opération addi.
Pour cela on regarde le troisième cycle d'horloge où le code de l'opération est "11, il suffit juste de comprendre et vérifier les valeurs prises par les signaux wire_1 et wire_2.
Le résultat de l'addition effectuée par l'UAL,obtenu sur le wire_15, est déjà correct.
Comme il s'agit de l'opération addi les lignes de sélection ual_src et ual_op des multiplexeur 1 et 0 respectivement sont à 1.
Donc à la sortie du multiplexeur 0 sur le wire_12 on doit avoir l'adresse du deuxième registre add_rt (et pas l'adresse du registre par défaut: add_rd = "11") on obtient ce qu'on souhaite wire_12 = wire_18 = "10".
La valeur de add_rs est "10" qui sert comme entrée au bloc "registers" au niveau de du port add_rs ainsi la valeur du sortie rs sur le wire_1 correspond à la valeur "11101011" stockée dans le troisième registre.
A la sortie du multiplexeur 1, sur le wire_2 on doit avoir add_rd sur 8 bits, on trouve "00000011", ce qui est prévu.
Par addition du wire_1 et wire_2 on trouve sur le wire_15 le résultat attendu qui est "11101110". Ce résultat est donc stocké dans le registre d'adresse add_rt = "10". Pour vérifier ceci on cherche dans les cycles qui suivent une adresse qui vaut "10" sur le wire_13 ou sur la wire_18 et on vérifie la valeur du registre correspondant. Dans le cinquième cycle d'horloge la valeur de add_rs est "01" est la valeur du registre rs sur le wire_1 est "11101110".

V. Conclusion

Durant ce projet, on a pu modéliser et simuler un processeur 8 bits basé sur une architecture simple.
La définition de l'architecture globale permet d'avoir une idée générale sur le fonctionnement du processeur, en spécifiant les différents composants et modules nécessaires à la réalisation des différentes instructions.
On a pu vérifier le comportement du processeur à travers l'interprétation des résultats de simulation, réalisée par le biais de Modelsim, et l'explication du fonctionnement de chaque bloc. Les scénarios de test permettent de couvrir les différentes instructions et conditions et mène à valider les différentes opérations définies: ET logique, l'addition, la soustraction et ADDI définie auparavant.
En conclusion, dans ce rapport on présente de façon structurelle le travail réalisé, allant de la présentation du fonctionnement général,passant par la présentation de l'architecture globale du processeur et les modules utilisés, jusqu'à la vérification et l'interprétation des résultats obtenus lors de la simulation pour chaque instruction et scénario de test.