Youssef LAKHDHAR
EISE3

RAPPORT PROCESSEUR MONO-CYCLE: SIMULATION VHDL

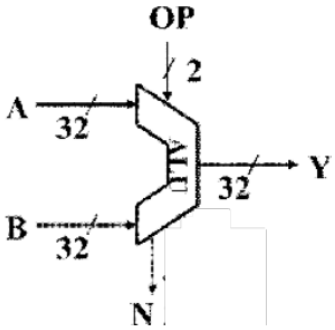
Partie 1 - Unité de traitement

La première partie de ce projet consistait à réaliser l'unité de traitement du processeur: cette partie avait pour fonction de réaliser les différentes opérations arithmétiques sur les registres.

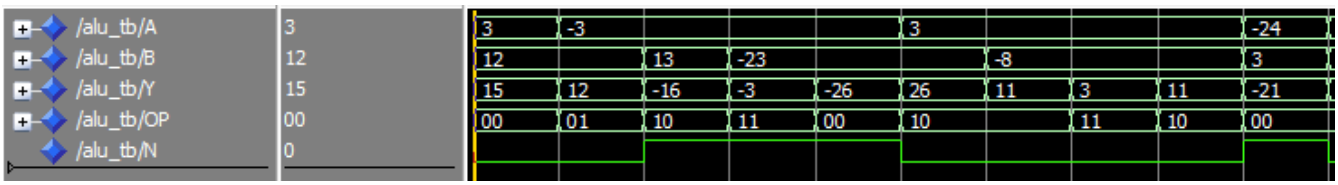
Unité Arithmétique Logique – UAL

Le premier composant à décrire est une unité arithmétique et logique (désignée UAL ci-après). Elle permet, au moyen de 2 entrées A et B sur 32 bits et d'un code opératoire OP sur 2 bits d'effectuer une addition, une soustraction ou simplement jouer le rôle de multiplexeur 2 vers 1.
Ci-dessous un schéma ainsi qu'un récapitulatif des actions.

ADD :	Y = A + B;	OP = "00";
B :	Y = B;	OP = "01";
SUB :	Y = A - B;	OP = "10";
A :	Y = A;	OP = "11";



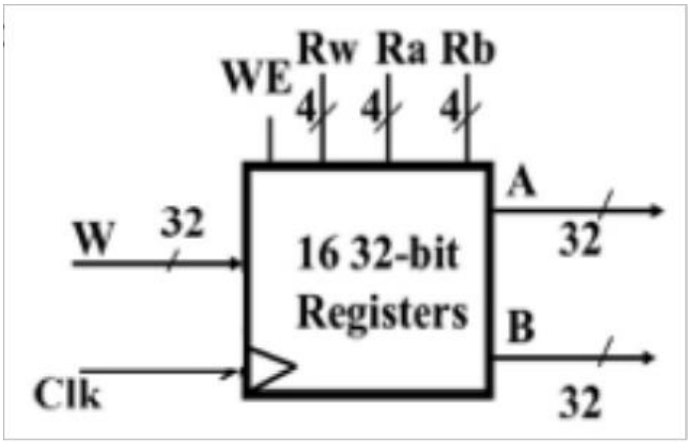
On réalise alors le test de ce composant pour les 4 valeurs possibles du code OP ainsi que des différents cas de figure pour A et B.
On obtient alors les résultats ci-dessous. (simu_alu.do)



On constate que tous les signaux correspondent bien au résultat attendu : le composant est donc validé.

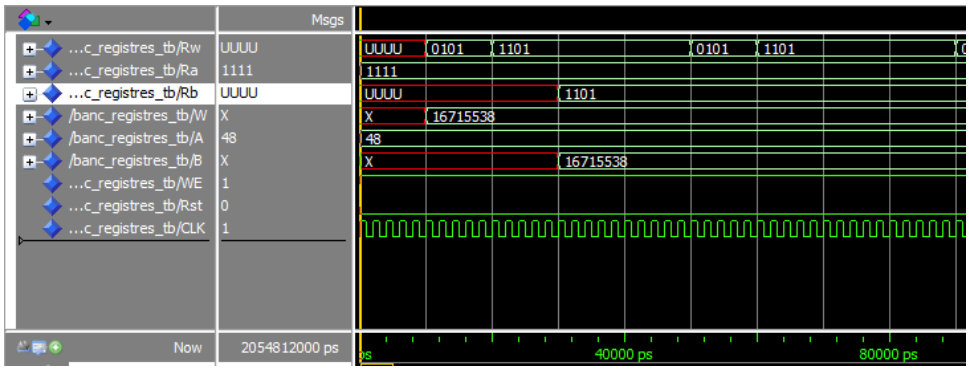
Banc de Registres

Ce banc de registres consiste en un tableau de 16 registres auxquels on inclut une commande de chargement WE synchrone.
Ainsi, si WE = 1 et que l'on se trouve sur un front montant d'horloge, alors on recopie l'entrée W dans le registre d'adresse Rw.
Sinon, les registres restent inchangés.
Enfin, les sorties sont affectées de manière combinatoire et simultanée : à tout instant, les bus A et B prennent respectivement les valeurs stockées dans les registres d'adresses Ra et Rb.
Ci-dessous un schéma du composant :



Afin de vérifier le bon fonctionnement du composant, on réalise successivement les opérations suivantes : (simu_registres.do)

A = R(15)
R(6) = 16715538
R(13) = 16715538
B = R(13)

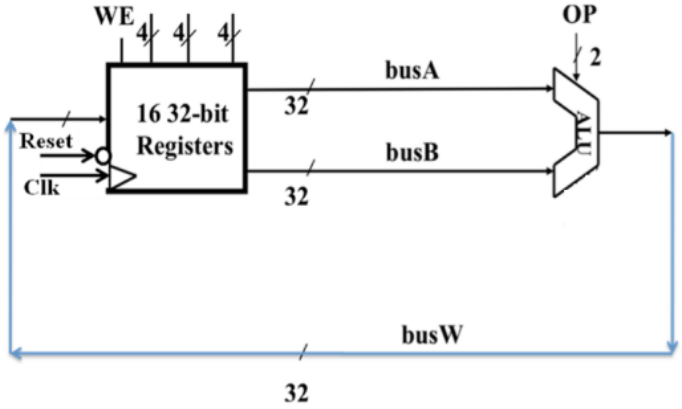


On observe alors que les sorties attendues sont bien celles attendues, malgrè la présence de signaux rouges car non initiailisés : le banc de registres est donc opérationnel.

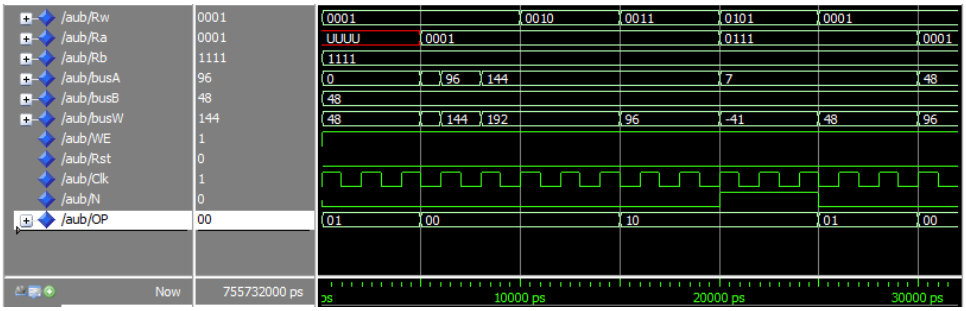
Test des deux premiers composants

Afin de valider les deux premiers composants, on les assemble de la manière suivante et tentons de réaliser quelques opérations.

R(1) = R(15)
R(1) = R(1) + R(15)
R(2) = R(1) + R(15)
R(3) = R(1) – R(15)
R(5) = R(7) – R(15)



Ci-dessous les résultats de la simulation.

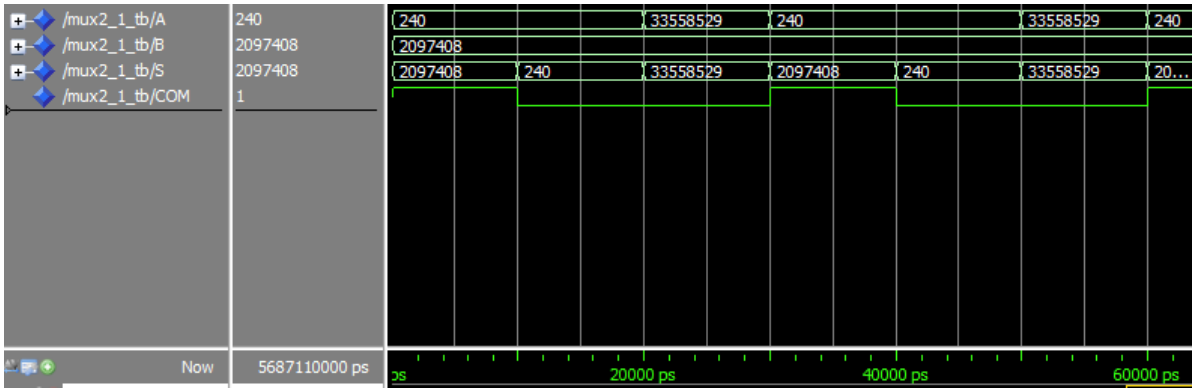


Certaines opérations sont exécutées plusieurs fois alors qu'elles devraient l'être une seule fois: cela est dû à un mauvais choix de clock.
Cependant, les résultats semblent cohérents : les composants sont donc validés.

Multiplexeur 2 vers 1

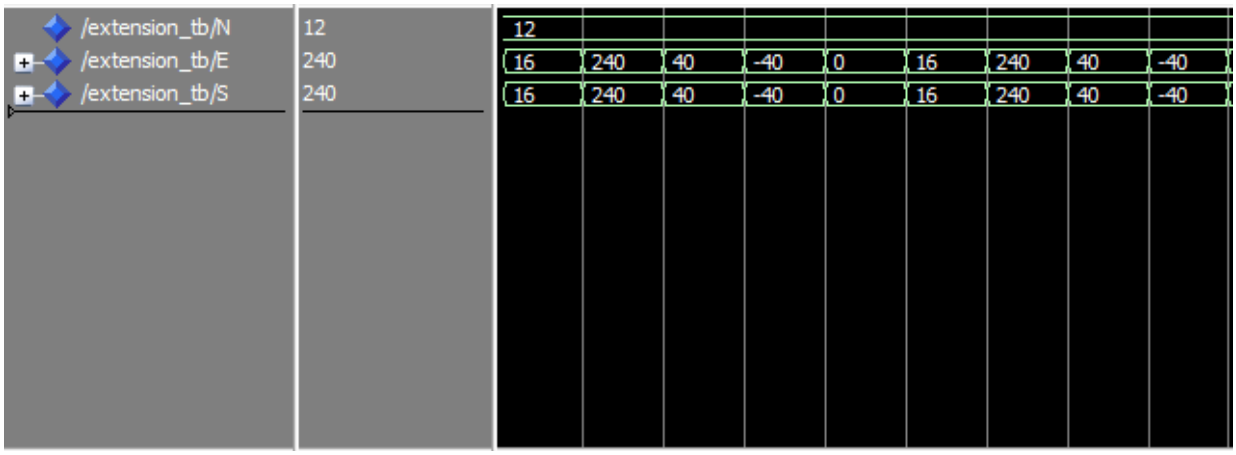
Le multiplexeur 2 vers 1 est tout ce qu'il y a de plus classique si ce n'est qu'on y introduit un paramètre générique N qui définit la taille des vecteurs d'entrée et de sortie.

Le banc de test suivant nous donne alors les résultats ci-dessous. (simu_mux.do)



Extension de signe

Le module d'extension de signe permet d'étendre la taille d'un vecteur. Après quelques brefs calculs, on se rend compte que peu importe le signe du nombre, il suffit de le compléter par son MSB jusqu'à obtenir le nombre de bits requis, ici N.



On peut alors vérifier le bon fonctionnement du fonctionnement en faisant varier la valeur de l'entrée. (simu_sign.do)
On obtient les mêmes nombres (en écriture décimale) en entrée et en sortie : on en déduit que le composant est fonctionnel.

Mémoire de données

La mémoire de données est très similaire au banc de registres, hormis le fait que les registres Ra,Rb,Rw sont remplacées par une unique adresse Addr qui sert de pointeur pour la lecture ainsi que pour l'écriture.

On effectue alors un banc de test rapide similaire à celui du banc de registres afin de vérifier le bon fonctionnement de la mémoire. (simu_mem_d.do)
On obtient des signaux cohérents: la mémoire est donc opérationnelle.

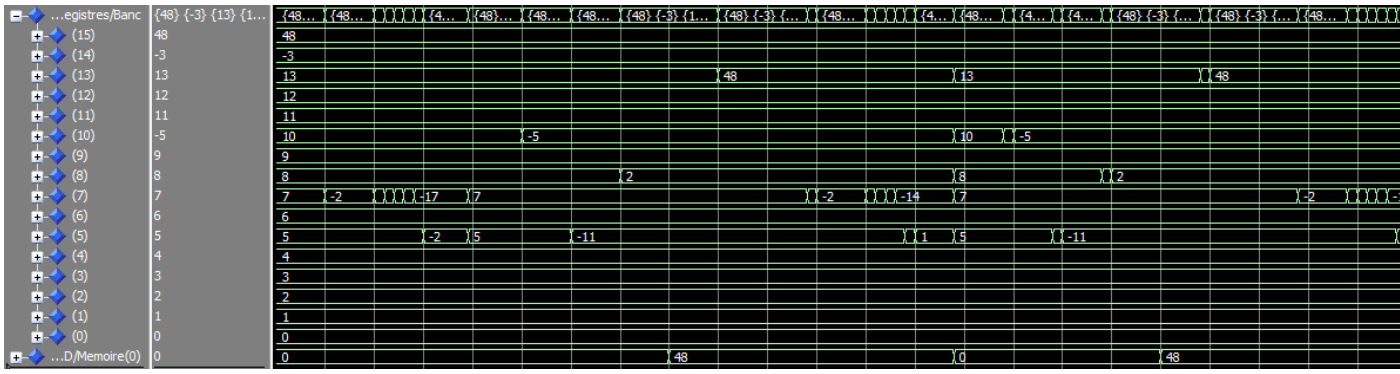
Assemblage Unité de Traitement

On finit par assembler tous les composants.

On simule alors les opérations suivantes:

- L'addition de 2 registres: R(7) = R(1) + R(14) puis R(7) = R(7) + R(14)
- L'addition d'1 registre avec une valeur immédiate: R(5) = R(7) + Imm et Imm = 16
- La soustraction de 2 registres: R(10) = R(4) - R(9)
- La soustraction d'1 valeur immédiate à 1 registre R(5) = R(4) - Imm et Imm = 15
- La copie de la valeur d'un registre dans un autre registre: R(8) = R(2)
- L'écriture d'un registre dans un mot de la mémoire: R(15) dans Memoire(0)
- La lecture d'un mot de la mémoire dans un registre: Memoire(0) dans R(13)

Le banc de test nous retourne alors ces valeurs. (simun_trait.do)



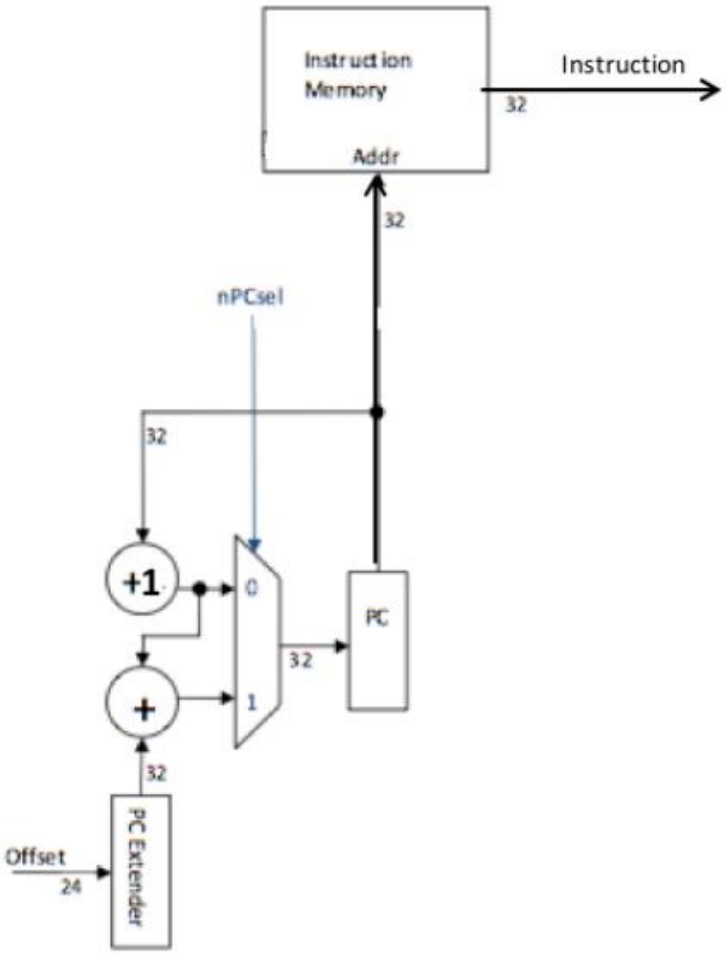
On rencontre une fois des opérations successives non désirées (en R(7) par exemple dûes à la clock) mais elles ne sont pas réellement dérangeantes dans la mesure où les instructions sont tout de même parfaitement exécutées.
On constate que les opérations sont bien réalisées : l'unité de traitement est donc opérationnelle.

PARTIE 2 – UNITE DE GESTION DES INSTRUCTIONS

Afin de pouvoir exécuter différentes actions au sein du processeur, il faut qu'il ait des instructions à exécuter. Parmi les instructions : on distingue 2 types :

- celles qui nécessitent un branchement et pour lesquelles on doit "sauter" d'une instruction à l'autre dans un ordre non linéaire ;
- les autres qui se lisent de manière séquentielle de haut en bas.

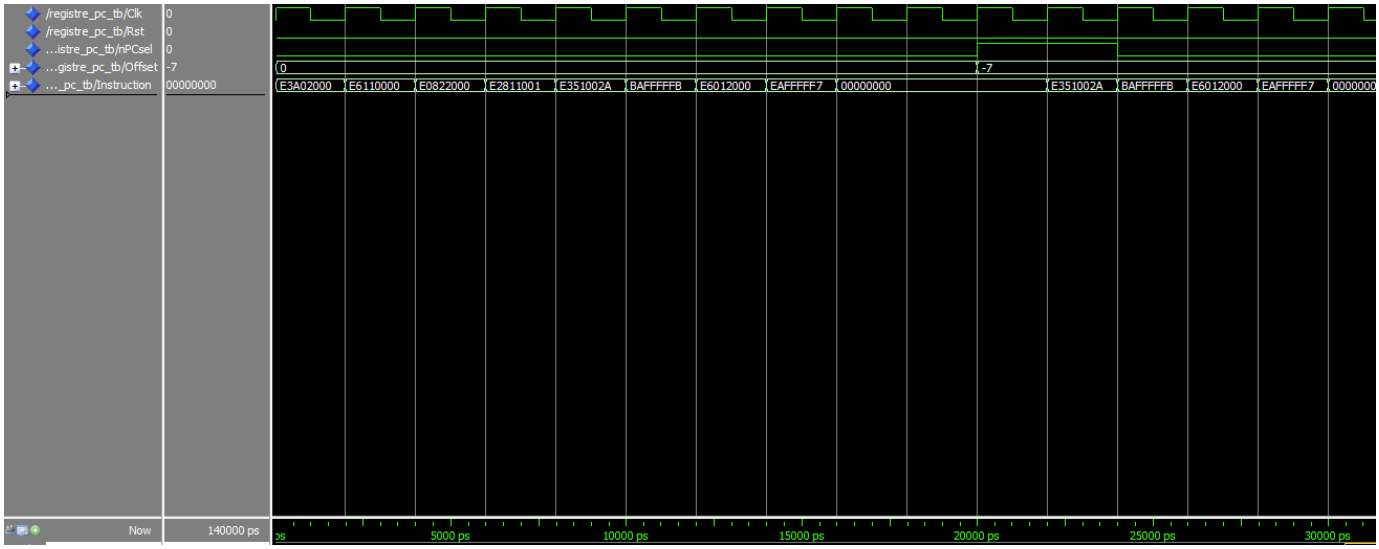
Pour gérer les sauts, on fait donc appel à un Offset qui permet de quantifie l'écart entre l'instruction juste après et l'instruction souhaitée.
Ci-dessous un schéma du dispositif.



Le registre PC joue alors le rôle d'adresse : l'instruction en sortie de l'unité de gestion des instructions sera celle l'instruction de la mémoire d'instructions dont l'adresse sera égale à la valeur contenue dans le registre PC.

On peut alors, au moyen d'un test-bench, réaliser différentes lectures séquentielles ainsi qu'un saut pour s'assurer du bon fonctionnement de ce module.

Ci-dessous le résultat de cette simulation. (simu_PC.do)

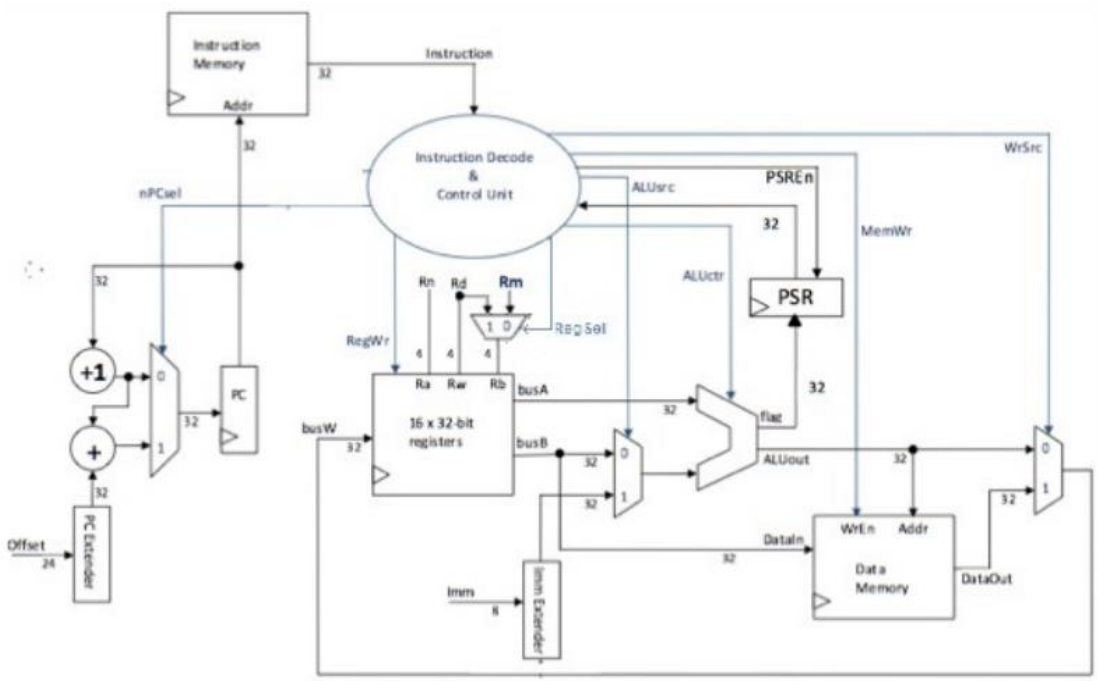


La lecture des instructions est bien réalisée à chaque fois et le saut de 6 instructions en arrière (donc un Offset de –7) est correctement effectué : l'unité de gestion des instructions est donc opérationnelle.

PARTIE 3 – UNITE DE CONTROLE

L'unité de contrôle est la partie la plus importante du processeur : c'est elle qui va traduire les instructions afin de générer les commandes et permettre à l'unité de traitement d'effectuer les bons traitements en générant les différents bits de contrôle des composants tels que les multiplexeurs, l'UAL, les commandes d'écriture et de lecture des bancs de registre...
Cependant, certaines instructions sont conditionnelles et dépendent notamment du signe des opérations réalisées au sein de l'UAL. C'est pourquoi il est nécessaire d'insérer un registre 32 bits, dit registre PSR, à commande de chargement afin d'actualiser la sortie seulement si nécessaire et permettre au décodeur d'agir en fonction du signe.

Une fois les entrées du décodeur gérées (l'instruction et le flag), il convient de gérer les sorties. Pour cela, il faut se servir du schéma général afin de savoir l'état de chaque bit pour chaque type d'instruction.



Par exemple, dans le cas d'une opération nécessitant une addition, on sait que OP sera à "00". Une opération d'écriture dans le banc de mémoire nécessitera d'avoir le bit MemWr à 1 et d'avoir le bit à RegWr à 1 dans le cas d'une écriture dans le banc de registres. Enfin, une opération nécessitant à un branchement nécessitera l'intervention de l'Offset et donc de mettre le bit nPCsel à 1.
Ci-dessous un tableau résumant l'état de chaque bit de commande pour chaque instruction.

INSTRUCTION	nPCSel	RegWr	ALUSrc	ALUCtr	PSREn	MemWr	WrSrc	RegSel
ADDi	0	1	1	00	0	0	0	1
ADDr	0	1	0	00	0	0	0	0
BAL	1	0	0	0	0	0	0	1
BLT	1	0	0	10	0	0	0	1
CMP	0	0	1	10	1	0	0	1
LDR	0	1	1	00	0	0	1	1
MOV	0	1	1	01	0	0	0	1
STR	0	0	1	00	0	1	0	0

PARTIE 4 – ASSEMBLAGE ET VALIDATION DU PROCESSEUR

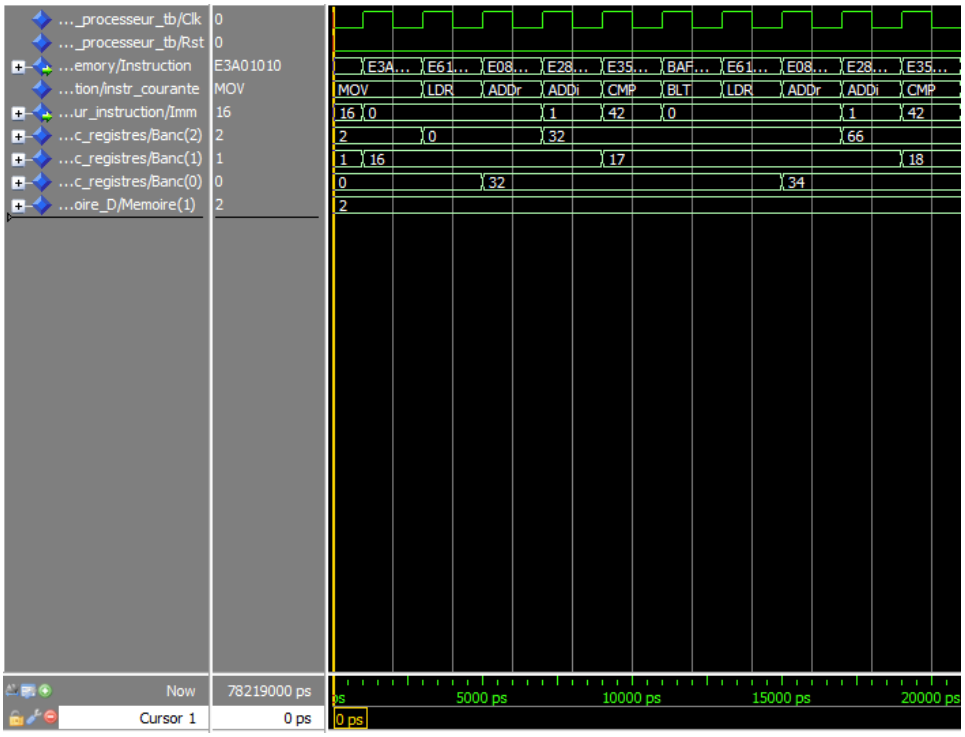
Afin de compléter le processeur, il convient d'ajouter un dernier multiplexeur 4 bits 2 vers 1 en entrée du banc de registres. En effet, pour certaines opérations nécessitant un second opérande, il convient d'utiliser un registre nommé Rm.
On instancie alors le multiplexeur générique codé au sein de l'unité de traitement.

On assemble alors l'ensemble des composants conformément au schéma.

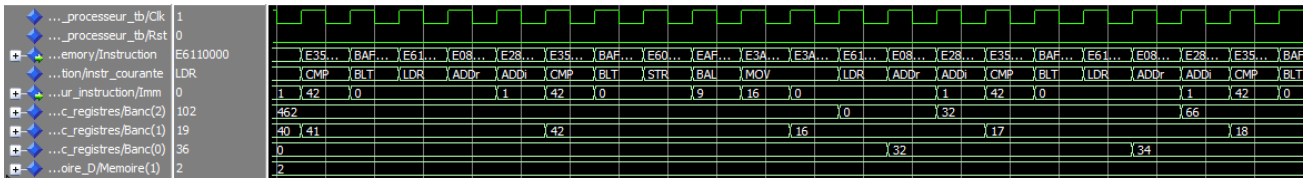
Les entrées se résument à une clock Clk et un Reset Rst.

Ci-dessous les résultats de la simulation. (simu_assemblage_4.do)

Il ne faut pas oublier de décommenter la ligne instruction_memory1 dans le code de registre PC et de commenter les 2 autres.
Cette contrainte aurait pu être évitée en créant 3 architectures différentes mais faute de temps, il a fallu imposer cela a l'utilisateur et je m'excuse par avance pour la gêne occasionnée.



On peut constater que le comptage dans R1 s'exécute parfaitement.
Observons ce qu'il se passe lorsque l'on arrive à 42.



On observe un saut au label main et donc un retour à R1 = 0x10.
Les instructions sont donc bel et bien exécutées correctement : le fonctionnement du processeur est donc validé.

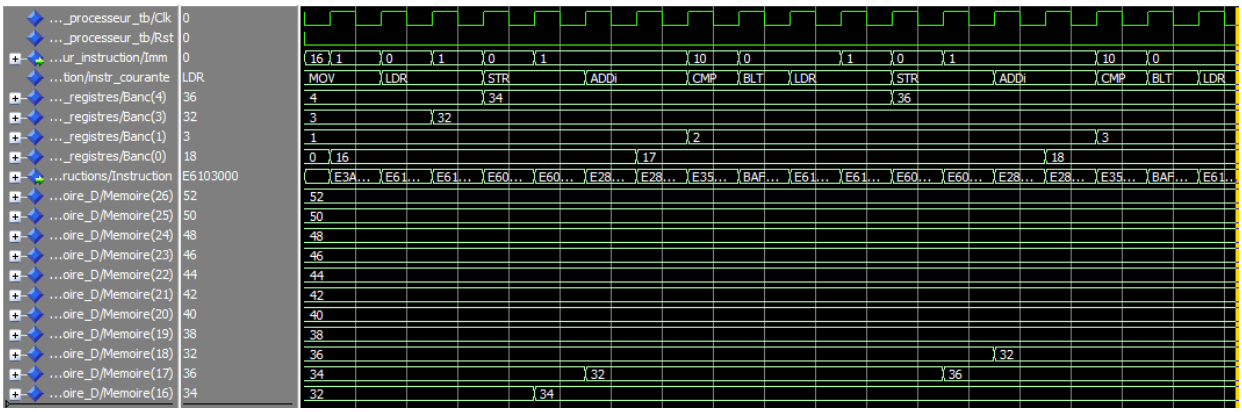
PARTIE 5 – TEST COMPLET DU PROCESSEUR

Une fois le processeur validé, on le teste avec un nouveau programme qui fait intervenir des Offset.
Il ne reste plus qu'à décoder les instructions à l'aide des annexes.
On en déduit que les instructions en binaire sont :

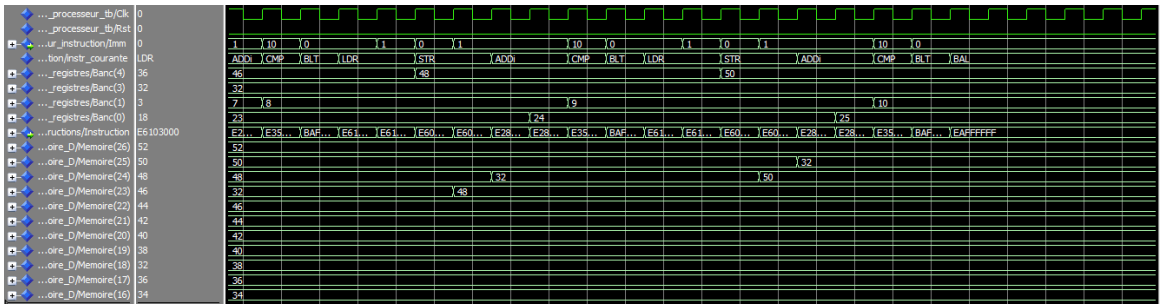
MOV R0, #0x10: 0xE3A00010
MOV R1, #1: 0xE3A01001
FOR: LDR R3, [R0]: 0xE6103000
LDR R4, [R0,#1]: 0xE6104001
STR R4, [R0]: 0xE6004000

STR R3, [R0,#1]: 0xE6003001
ADD R0,R0,#1: 0xE2800001
ADD R1,R1,#1: 0xE2811001
CMP R1,#0xA: 0xE351000A
BLT FOR: 0xBAFFFFF8 (on remonte de 7 instruction donc OFFSET = -8)
wait: BAL wait: 0xEAFFFFFF

Il ne faut pas oublier de décommenter la ligne instruction_memory2 dans le code de registre PC et de commenter la ligne instruction_memory1.



On en déduit que les instructions s'effectuent conformément à ce qui est demandé et que l'on boucle bien car la valeur contenue dans R1 est inférieure à 10.
Regardons ce qui se passe quand R1 dépasse 9. (simu_assemblage_5.do)



On constate que le processeur boucle sur l'instruction wait et rien ne se passe : le processeur exécute donc bien le programme souhaité et son fonctionnement est donc encore une fois validé.

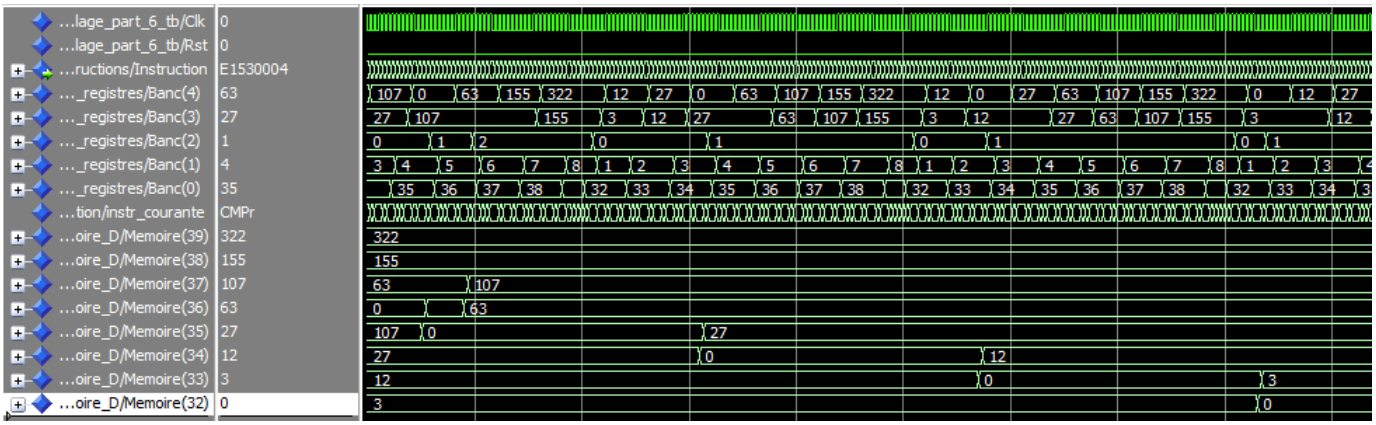
PARTIE 6 – TEST COMPLET DU PROCESSEUR

Cette dernière partie consistait à élargir le jeu d'instructions. Il s'agissait essentiellement de rajouter des conditions à nos instructions (de Addi à ADDGT, de STR à STRGT et de BAL à BNE). Il a également fallu rajouter l'instruction CMPr, qui effectuait la comparaison entre 2 registres et non plus entre un registre et un immédiat.
On en déduit alors le code binaire associé à chacune des instructions du programme du tri à bulles.

start: MOV R0, #0x20: 0xE3A00020
MOV R2, #1: 0xE3A02001
WHILE: MOV R2, #0: 0xE3A02000
MOV R1, #1: 0xE3A01001
FOR: LDR R3,[R0]: 0xE6103000
LDR R4, [R0,#1]: 0xE6104001
CMPr R3,R4: 0xE1530004
STRGT R4,[R0]: 0xC6004000
STRGT R3,[R0+1]: 0xC6003001
ADDGT R2,R2,#1: 0xC2822001
ADD R0,R0,#1: 0xE2800001
ADD R1,R1,#1: 0xE0811001
CMP R1,#0x7: 0xE3510007
BLT FOR: 0xBAFFFFF6 (Saut de -10)
CMPi R2,#0: 0xE3520000
MOV R0,#0x20: 0xE3A00020
BNE WHILE: 0x1AFFFFF1 (Saut de -15)
wait: BAL wait: 0xEAFFFFFF

Il ne faut pas oublier de décommenter la ligne instruction_memory3 dans le registrePC et commenter la ligne du dessus.

Ci-dessous le résultat de la simulation. (simu_part_6.do)



On remarque les valeurs de 32 a 39 de la mémoire sont bien rangées dans l'ordre croissant : le tri a donc été correctement effectué.

CONCLUSION

La conception du processeur a donc été finalisé. Évidemment, beaucoup d'autres instructions auraient pu être codées afin de le rendre plus complet mais l'on possède déjà un jeu d'une douzaine d'instructions ce qui permet un bon nombre de possibilités.

Ce projet a été l’occasion de mettre en pratique l'assembleur vu au cours du premier semestre et m’a permis d’avoir une meilleure compréhension du fonctionnement d’un processeur.
Une perspective d’amélioration pourrait cependant être d’ajouter d’autres cycles afin de le rendre plus efficace.