Rapport : Première séquence de TP (1 à 5)

PROCESSEUR MONO−CYCLE: SIMULATION VHDL

Groupe :
Lounes ACHAB,
Maxime MARTELLI

Sommaire :

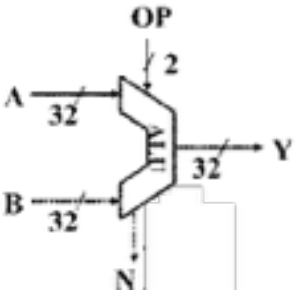
Introduction : aide à la compilation

Afin de faciliter la lecture, nous avons divisé les fichiers suivant les différentes parties. Il y a donc quatre dossiers, comprenant chacun un dossier SOURCES et un dossier TESTBENCHS. Il suffit donc de rajouter au fur et à mesure toutes les sources et testbenchs de chaque partie lue. En arrivant à la partie 4, on aura ainsi tous les fichiers nécessaires.

De plus, il sera souvent utile, dans les simulations, de mettre pour certaines variables le Radix en décimal ou en héxadécimal, afin d'augmenter la lisibilité.

Partie 1 - Unité de traitement

1. Unité Arithmétique Logique - UAL



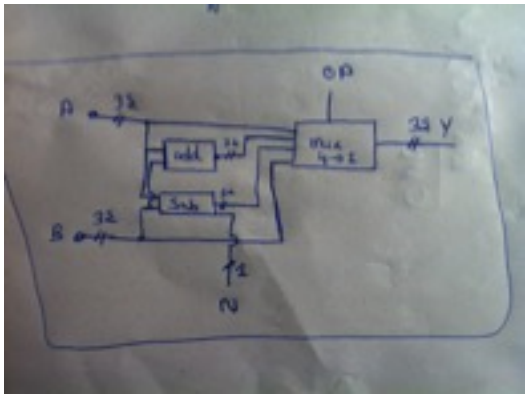


Figure 1 : schémas de l'UAL

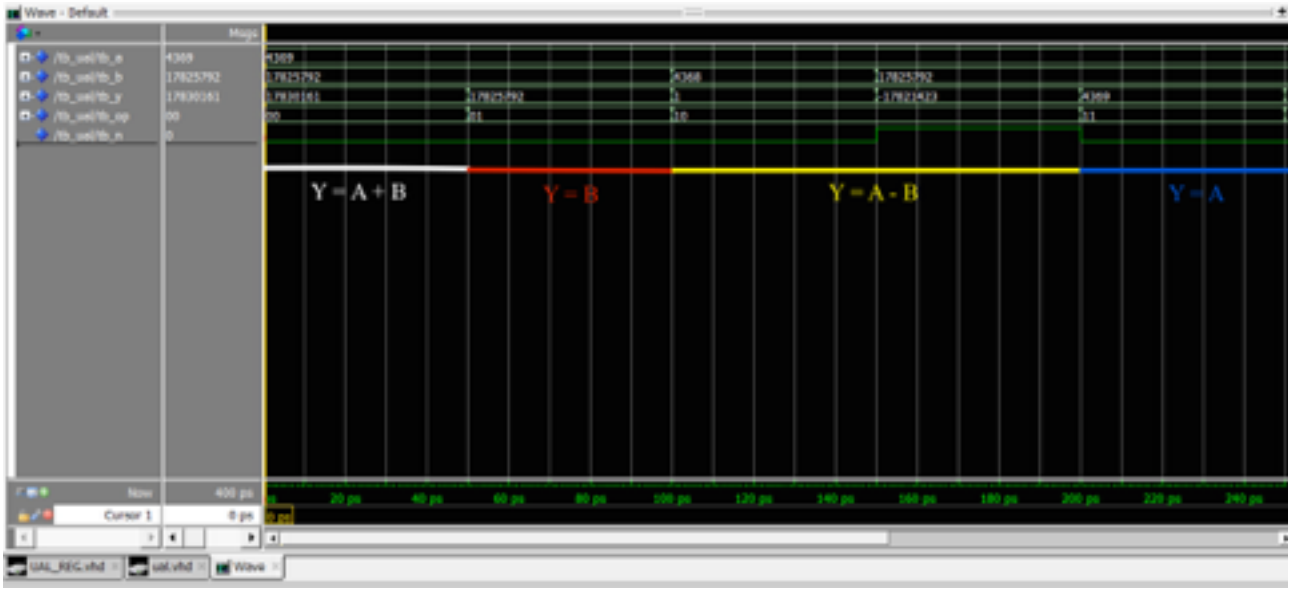


Figure 2 : simulation de l'UAL

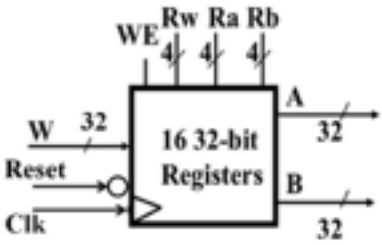
On remarque que le drapeau est bien égal à 1 lorsque l'on fait une soustraction dont le résultat est négatif.

Difficultés :

La principale difficulté de la création de l'UAL était de pouvoir effectuer facilement une comparaison de deux std_logic_vector. L'astuce trouvée est de convertir ceux-ci en entier :

to_integer(unsigned(A)) renvoie un entier (lorsque A est de type std_logic_vector)

2. Banc de registres - Banc_de_16_registres_32



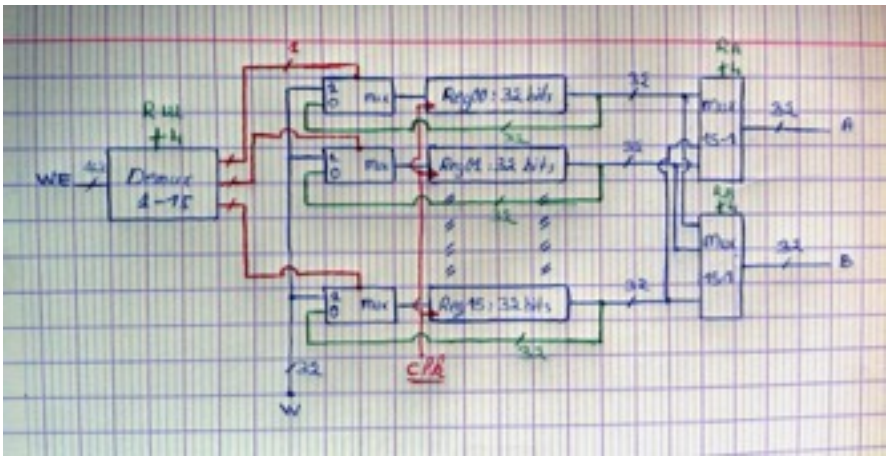


Figure 3 : schémas du banc de registres

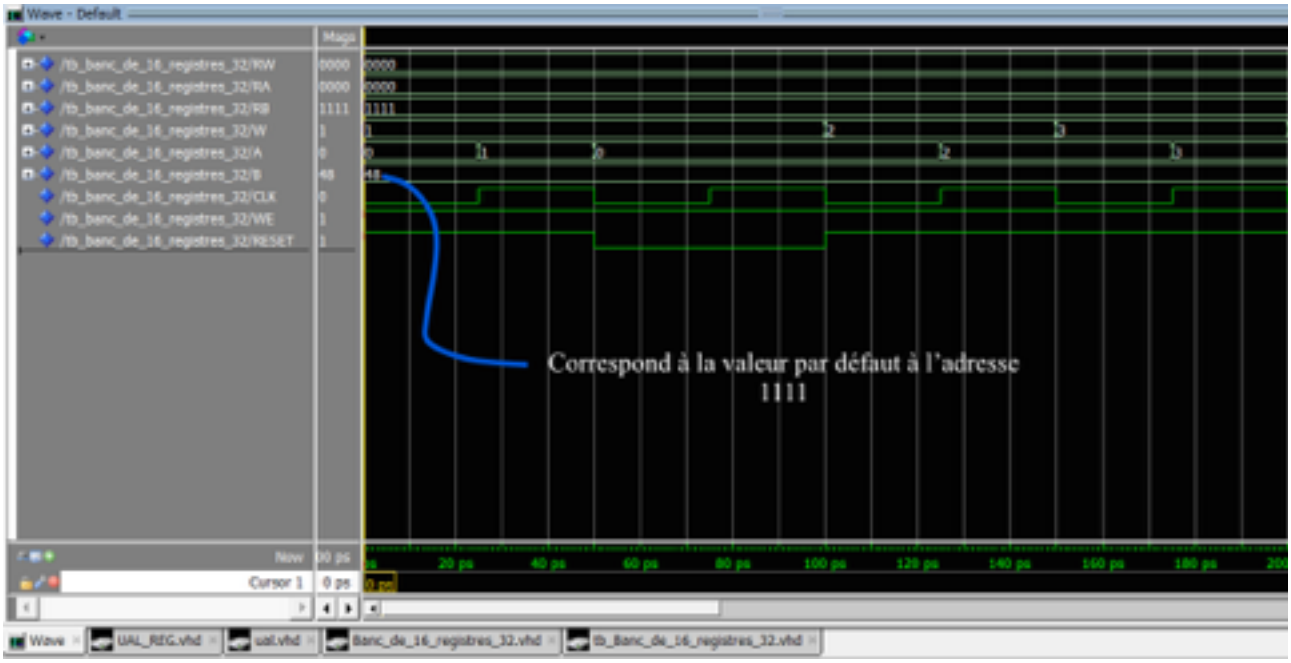


Figure 4 : simulation du banc de registres

On observe bien que l'écriture se fait de manière synchrone (à 100 ps, il faut attendre le prochain front montant pour pouvoir écrire W dans le registre RA), et la lecture se fait de manière combinatoire.

Difficultés :

La principale difficulté ici était de bien comprendre la différence entre l'écriture et la lecture. Étant synchrone, l'écriture devait se faire dans un process, alors que la lecture se décrit de manière combinatoire.

3. Assemblage banc de registres et UAL - UAL_REG

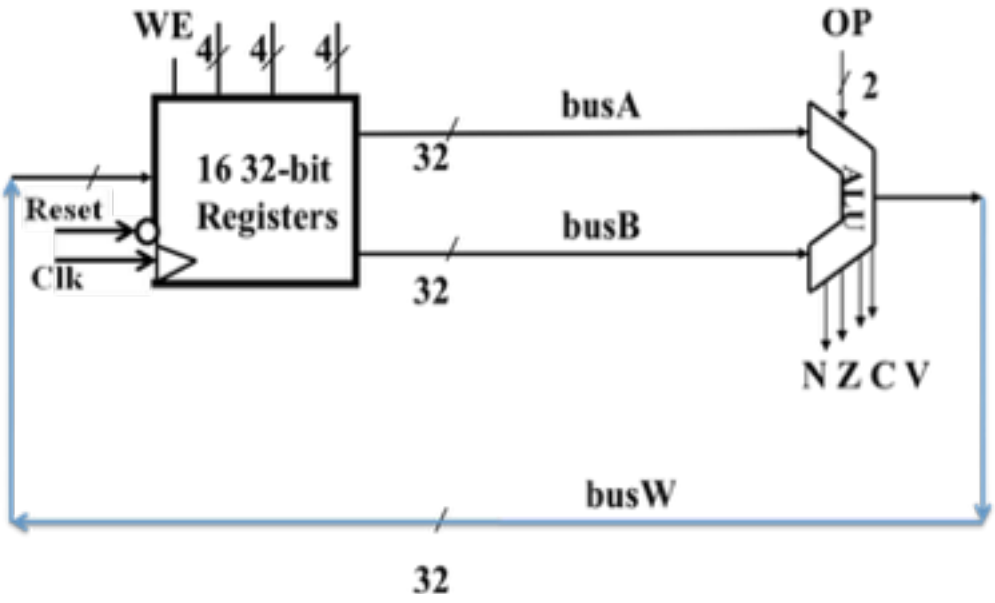


Figure 5 : schéma de l'assemblage UAL-banc de registre

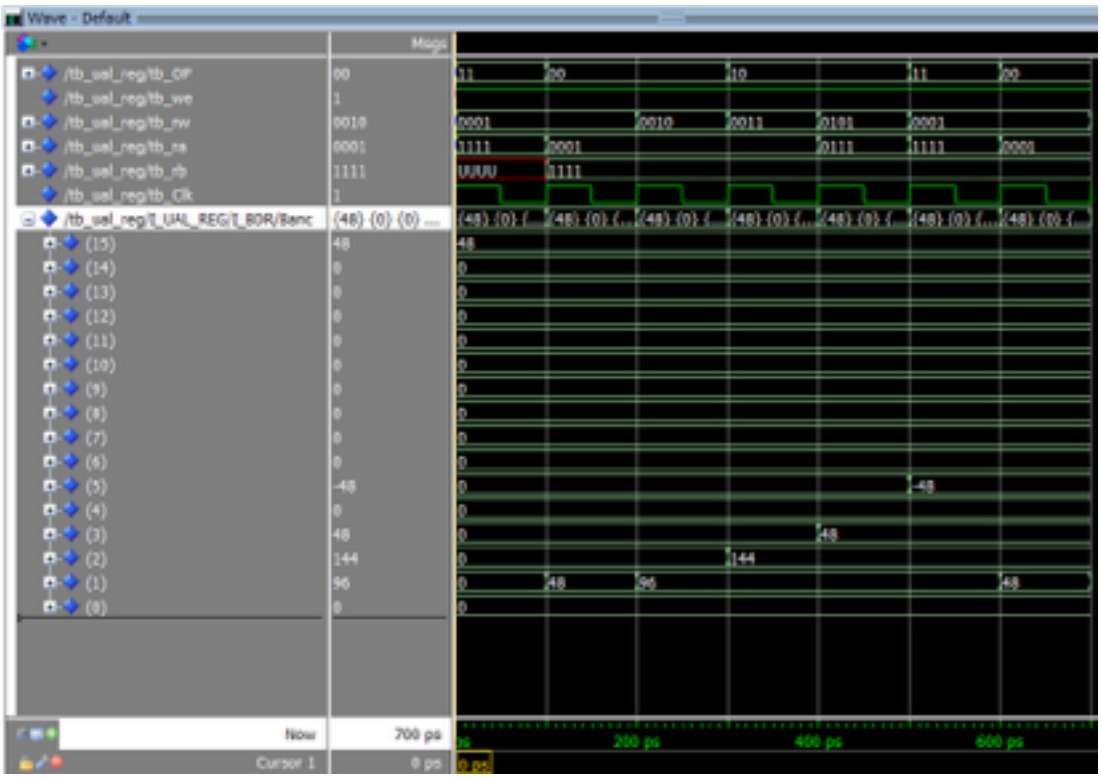


Figure 6 : simulation de l'assemblage entre l'UAL et le banc de registres

Le testbench demandé devait simuler :

R(1) = R(15) (= 48)
R(1) = R(1) + R(15) (= 48 + 48 = 96)
R(2) = R(1) + R(15) (= 96 + 48 = 144)
R(3) = R(1) - R(15) (= 96 - 48 = 48)
R(5) = R(7) - R(15) (= 0 - 48 = - 48)

On observe en figure 6 que nous avons exactement ce résultat. Le testbench valide bien l'assemblage du banc de registre avec l'UAL.

Difficultés :

Il n'y a pas de réelle difficulté dans cette étape, si on omet les erreurs de syntaxes habituelles (« ; » au lieu de « , » lors des instanciations, etc). Il n'y a qu'à relier deux composants entre eux. Nous pouvons tout de même noter l'utilisation des signaux internes afin de pouvoir relier les entrées et sorties facilement.

4. Multiplexeur 2-1 - mult2_1

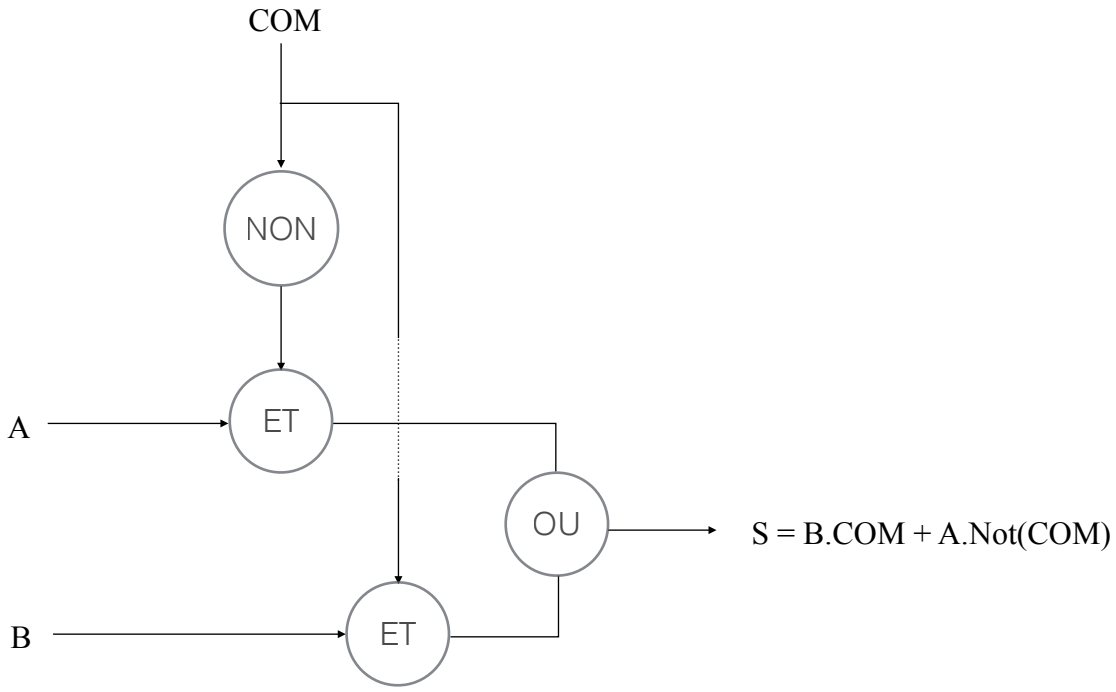


Figure 4 : schéma bloc du multiplexeur 2 vers 1

Figure 5 : simulation du multiplexage 2 vers 1

Comme demandé, lorsque COM = 0, S = A, et lorsque COM = 1, S = B.

Difficultés :

La présence d'un paramètre générique demande une syntaxe légèrement différente, et il ne faut pas oublier de l'initialiser dans le testbench.

5. Extension de signe - extension_signe

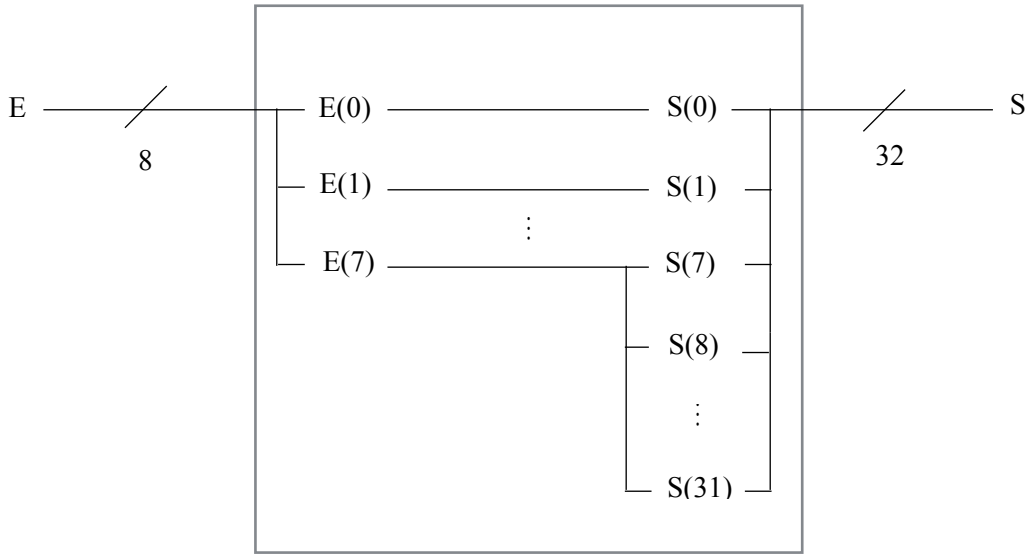


Figure 6 : schéma bloc de l'extension de signe

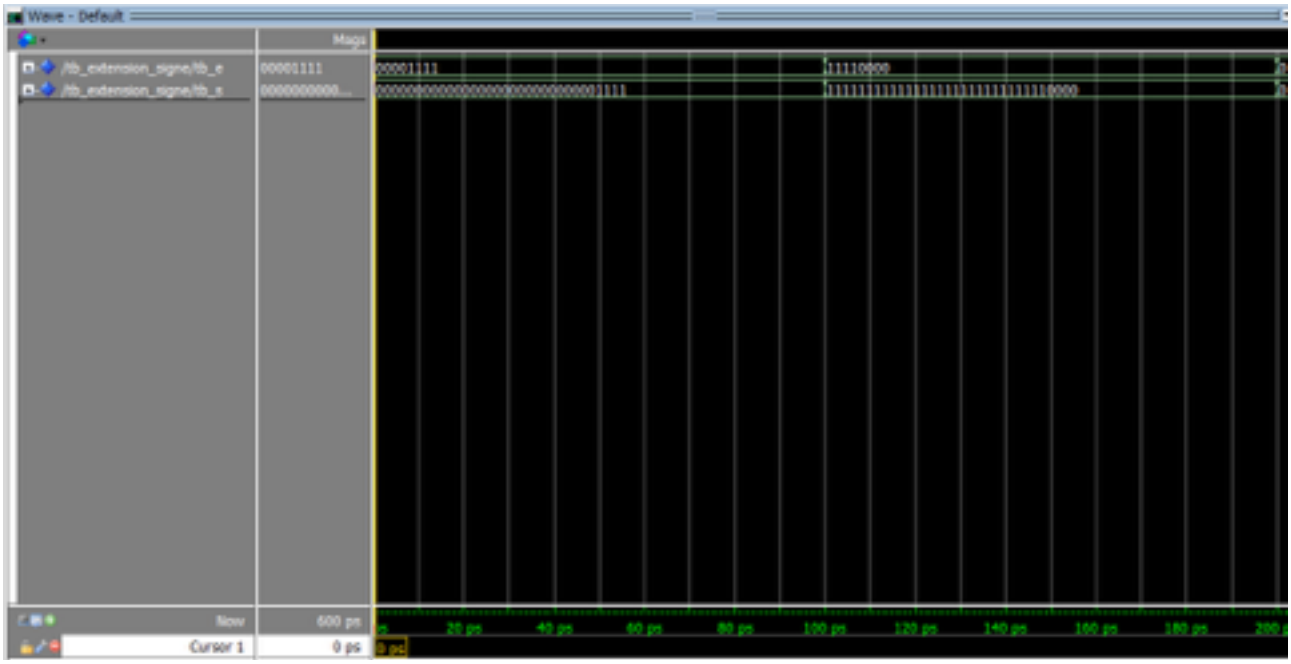


Figure 7 : simulation de l'extension de signe

Difficultés :

Au lieu de devoir faire une boucle pour compléter le signal d'entrée, une utilisation judicieuse de « others » permet de faciliter l'architecture de l'extension de signe.

6. Simulation de l'unité de traitement - unite_de_traitement

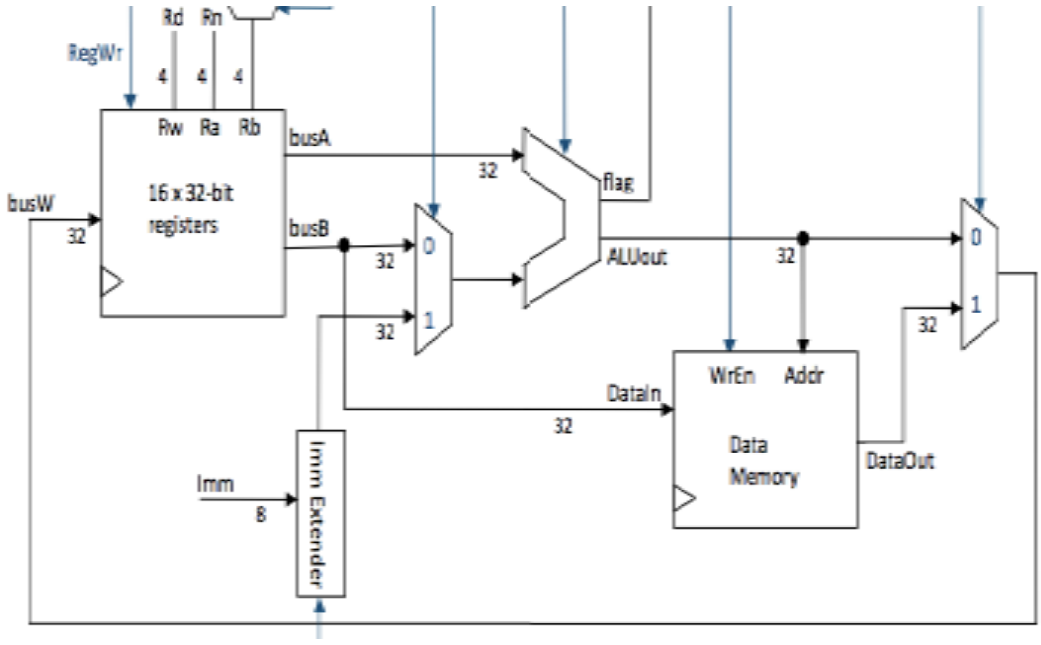


Figure 8 : schéma de l'unité de traitement

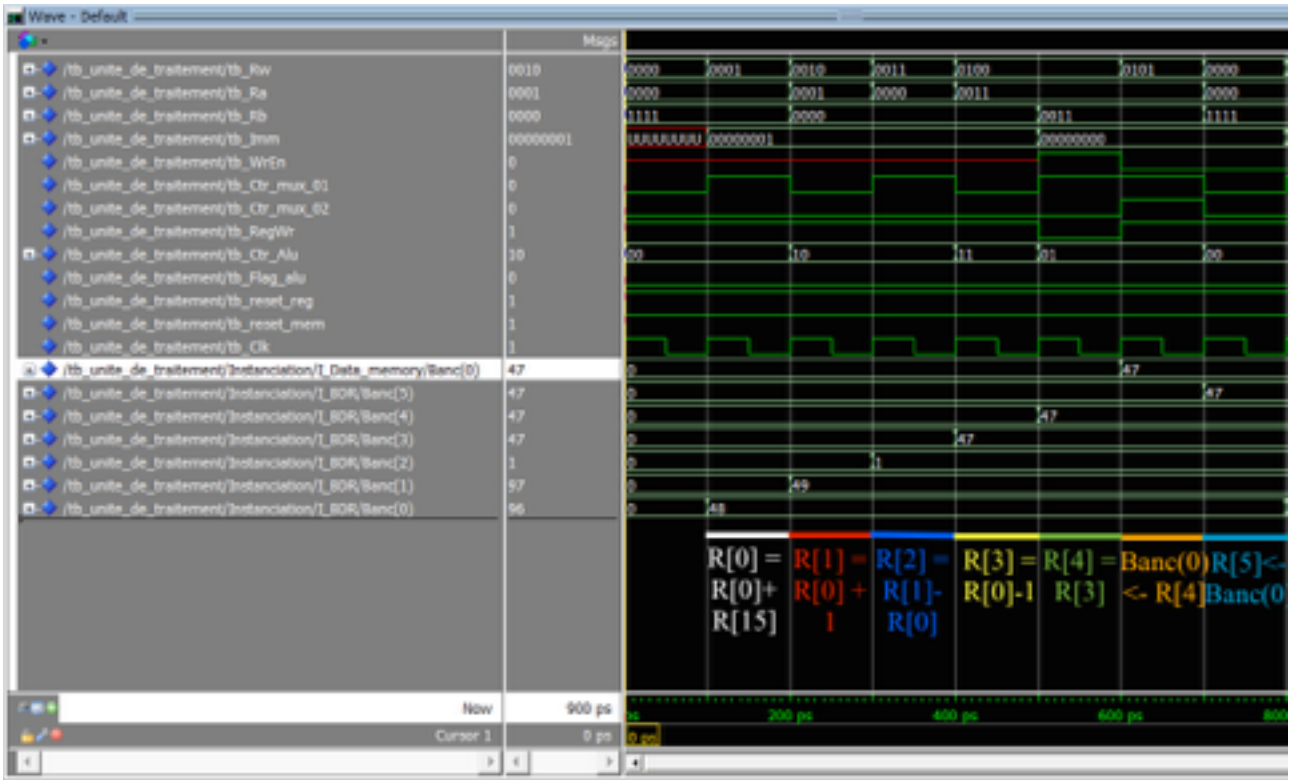


Figure 9 : simulation de l'unité de traitement

On remarque que les opérations annotées sur la figure 9 sont bien effectuées. Notre unité de traitement est donc opérationnelle .

Difficultés :

Tout d'abord, nous avons essayé d'intégrer le module déjà construit comprenant l'UAL ainsi que le Banc de registres. Malheureusement, l'ajout du multiplexeur au milieu rend cela impossible, et nous ne pouvons donc pas réutiliser ce module.

Encore une fois, l'utilisation des signaux internes est obligatoire afin de faciliter la jonction entre les différents composants, et une écriture papier de ceux-ci permettent d'éviter des erreurs d'étourderies.

Dernièrement, on notera que nous avons du faire tout particulièrement attention aux commandes d'écritures (WrEn et RegWr), afin de ne pas écrire dans la mémoire ou dans le banc de registre sans le vouloir.

Partie 2 - Unité de gestion des instructions

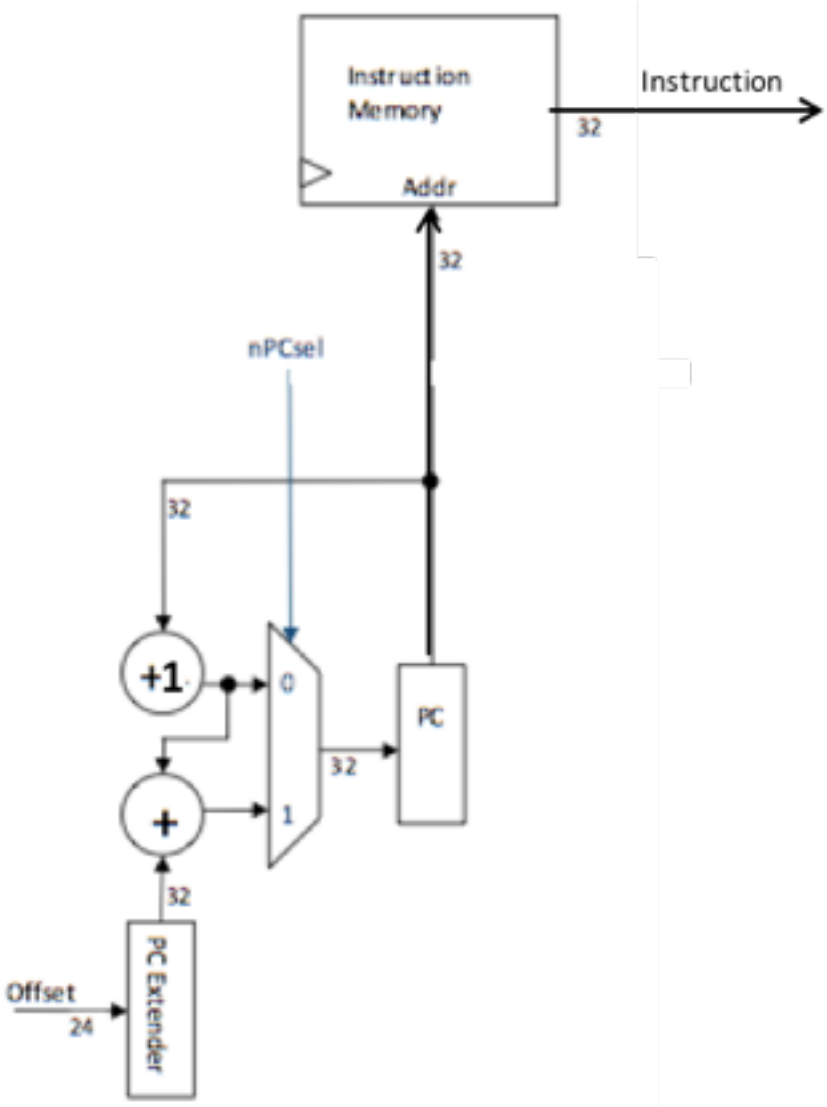


Figure 10 : schéma de l'unité de gestion des instructions

Cette partie va être divisée en plusieurs étapes.

1. Instruction-memory

   Ce bloc est très similaire au bloc data_memory, sauf que les dimensions sont différentes. On initialise les huit premières lignes avec les commandes en binaire du programme de test (c.f : Partie 4). On n'oublie pas de prendre en compte le reset. Contrairement à la data_memory, l'instruction est déterminée de manière combinatoire, il n'y a pas de Write Enable ni de clock !

2. PC

   Ce bloc est des plus simples : sur le front montant de l'horloge, il transfère l'entrée sur la sortie, et sur le reset, il met la sortie à X "00000000 ".

3. Unite_de_gestion_des_instructions

   Il s'agit de mettre en commun les composants créés précédemment, tout en rajoutant l'addition ainsi que le multiplexeur suivant la valeur de nPCSel. L'extension de signe se fait en réutilisant le bloc précédemment créé, tout en changeant la valeur du paramètre générique à 24 au lieu de 8. Il n'y a pas de difficulté en soi dans cette partie, sauf celle d'écrire au préalable sur papier tous les bus de l'unité afin de ne pas faire de faux raccords entre les composants.

4. Testbench de l'unité de gestion des instructions

- Remarquons tout d'abord que la valeur X"FFFFFD" de offset correspond à -3 en écriture C2
- On peut vérifier que le compteur PC marche bien lorsque nPCsel est égal à zéro. En effet, on lit bien les instructions à la suite dans l'ordre (et on a bien PC = PC + 1).
- De plus, lorsque nPCsel passe à 1, on observe que PC diminue de deux. En effet, on a alors bien PC = PC + 1 + offset (puisque offset = - 3, PC = PC - 2).
- Finalement, lorsqu'on effectue le reset, PC repasse bien à 0.

  Ce testbench permet donc de bien valider le fonctionnement de l'unité de gestion des instructions.

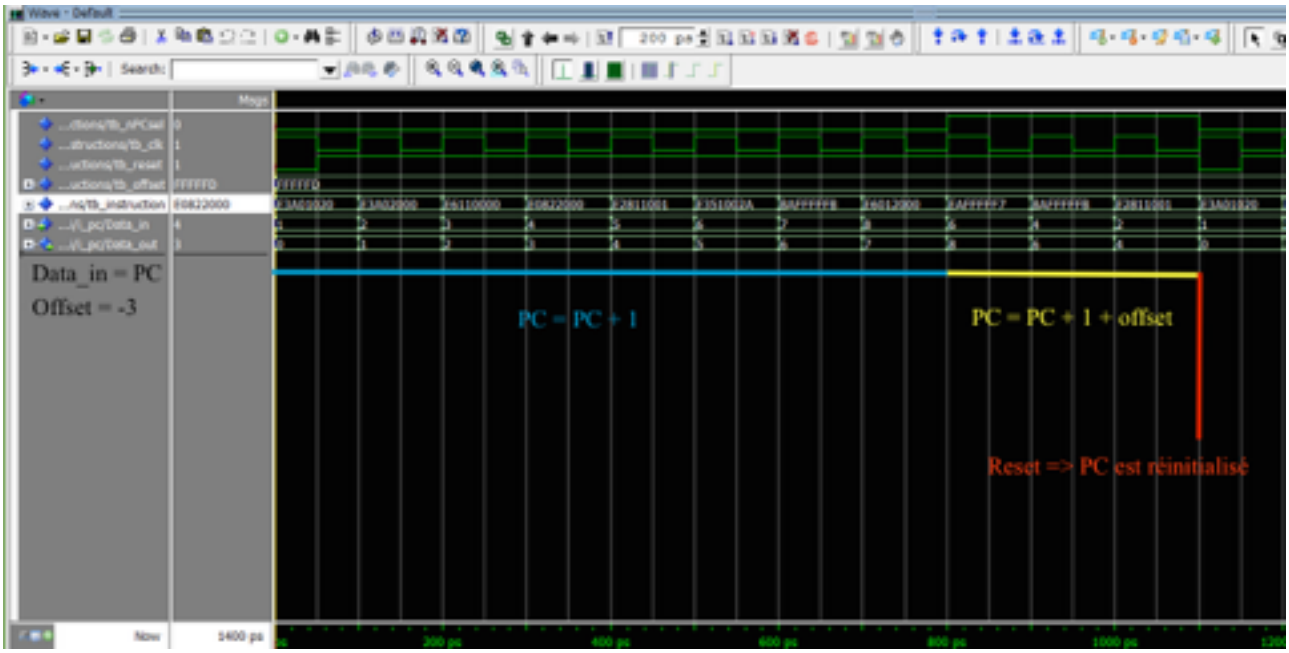


Figure 11 : simulation de l'unité de gestion des instructions

Partie 3 - Unité de contrôle

Instruction	nPCSel	RegWr	ALUSrc	ALUCtr	PSREn	MemWr	WrSrc	RegSel
ADDi	0	1	1	00	0	0	0	/
ADDr	0	1	0	00	0	0	0	0
BAL	1	0	/	/	/	0	/	/
BLT	PSRValue(0)	0	/	/	0	0	/	/
CMP	0	0	1	10	1	0	/	/
LDR	0	1	/	11	0	0	1	/
MOV	0	1	1	01	0	0	0	/
STR	0	0	/	11	0	1	/	1

Figure 11 : Valeur des commandes

Cette partie a deux blocs : le registre PSR et le décodeur d'instructions.

1. Registre PSR

   Ce bloc est simple : si on peut écrire dans Dataout (Write Enable = 1), alors on transfère la donnée de Datain dans Dataout. Sinon, si le reset est actif, on met Dataout à 0.

2. Décodeur d'instructions

   La difficulté est de bien dissocier les deux phases : fixer la valeur de l'instruction, et donner les valeurs des commandes des registres et des opérateurs du processeur.

- Fixer la valeur de l'instruction courante : à l'aide des fichiers en annexe, on trie les différentes instructions suivant les bits du signal "instruction" d'entrée, à l'aide de différents « case ».
- A l'aide de la figure 11, on donne les valeurs des commandes des registres suivant l'instruction courante.
  La principale difficulté d'avoir une très bonne rigueur afin de ne pas se tromper sur l'étude théorique : en effet, une erreur dans le tableau peut faire planter le programme par exemple.
- Une autre difficulté survenue est lors de la liste de sensibilité du process sur l'instruction. En effet, on fait des matchs de bouts de l'instruction sur des variables, et on avait donc fait le process sur les bouts d'instructions et non sur les variables créées. En raison de décalages dans la simulation VHDL (les deltas), cela ne marchait pas.

3. Testbench du décodeur

   On peut vérifier que la figure 12 n'est que la traduction visuelle du tableau de la figure 11, et que nous avons bien les instructions rentrées dans l'instruction_memory.

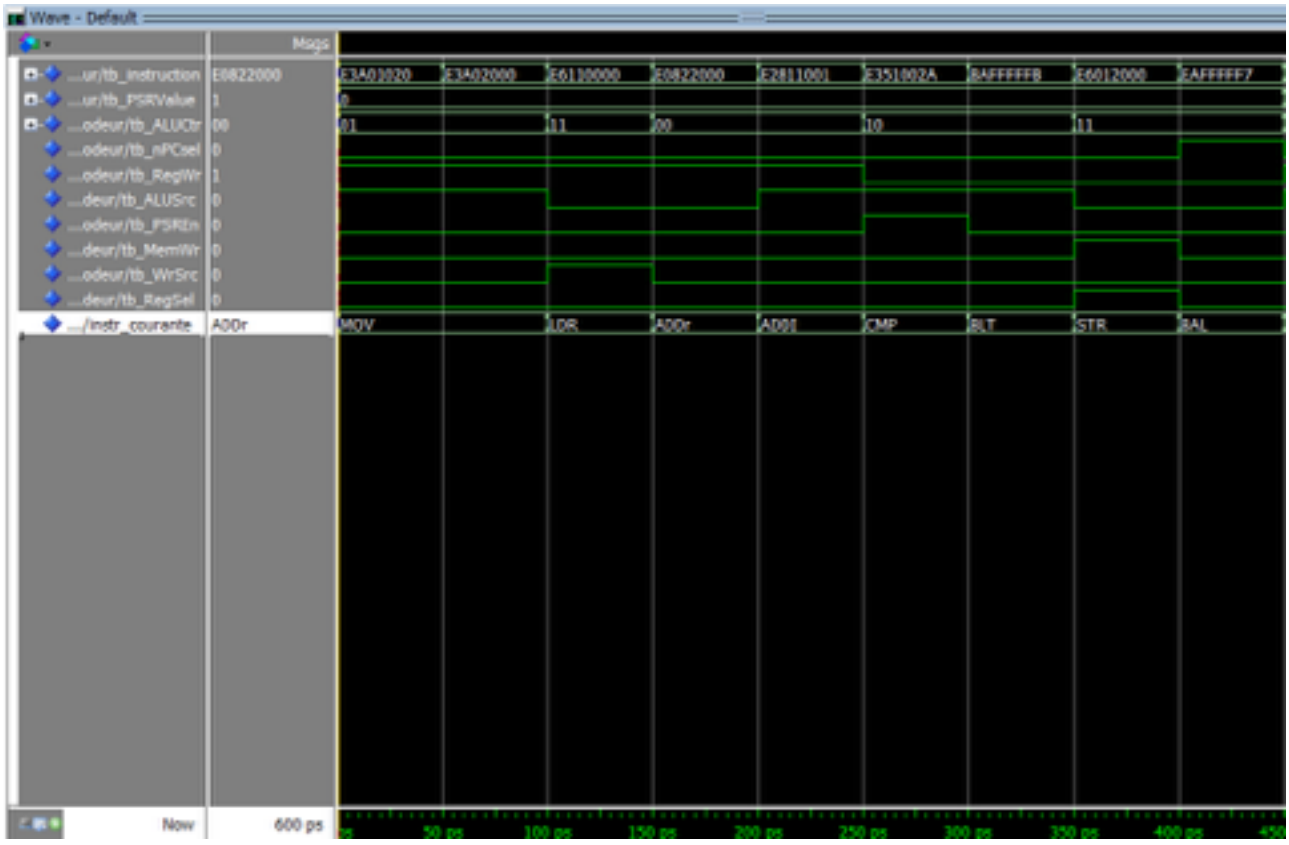


Figure 12 : Simulation du décodeur

Partie 4 - Assemblage et validation du processeur

Il est important de noter que nous utilisons ici un autre fichier VHDL pour l'unité de traitement (unité_de_traitement_final). Ce dernier est presque identique au premier, à ceci près qu'on lui a rajouté le multiplexeur sur Rb.

En utilisant les documents en annexe, on a donc pu définir la conversion binaire et hexadécimale des commandes du programme de test :

Commande : MOV R(1), #0x20
Binaire : 1110 0011 1010 0000 0001 0000 0010 0000
Hexadécimal : E3A01020

Commande : MOV R(2), #0
Binaire : 1110 0011 1010 0000 0010 0000 0000 0000
Hexadécimal : E3A02000

Commande : LDR R(0), 0(R1)
Binaire : 1110 0110 0001 0001 0000 0000 0000 0000
Hexadécimal : E6110000

Commande : ADD R(2), R(2), R(0)
Binaire : 1110 0000 1000 0010 0010 0000 0000 0000
Hexadécimal : E0822000

Commande : ADD R(1), R(1), #1
Binaire : 1110 0010 1000 0001 0001 0000 0000 0001
Hexadécimal : E2811001

Commande : CMP R(1), 0x2A
Binaire : 1110 0011 0101 0001 0000 0000 0010 1010
Hexadécimal : E351002A

Commande : BLT loop (l'adresse de _loop est 0x2)
Binaire : 1011 1010 1111 1111 1111 1111 1111 1001
Hexadécimal : BAFFFFFB

Commande : STR R(2), 0(R(1))
Binaire : 1110 0110 0000 0001 0010 0000 0000 0000
Hexadécimal : EAFFFFF7

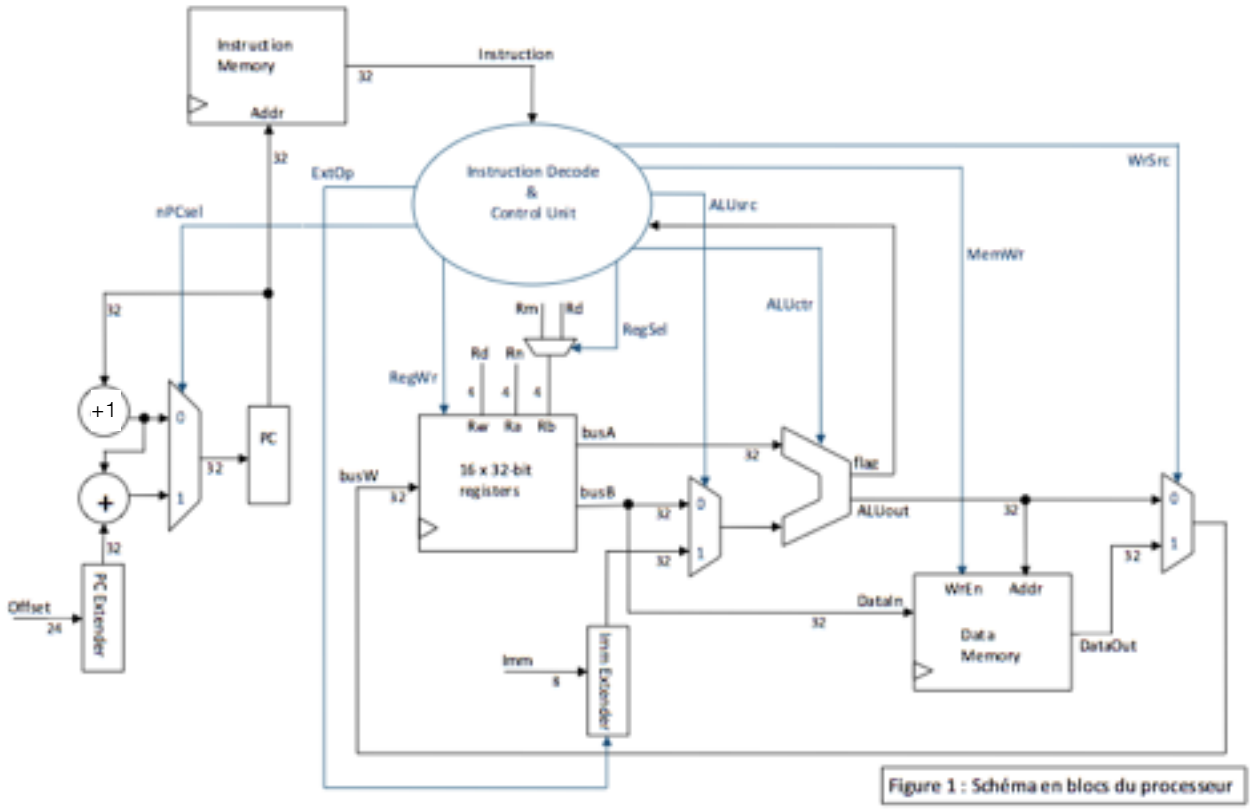


Figure 12 : Schéma en blocs du processeur

Ici encore, la difficulté majeure a été de ne pas se tromper dans les raccords : un schéma sur papier a permis de bien mettre les choses au clair avant de décrire le processeur en VHDL.

Testbench processeur :

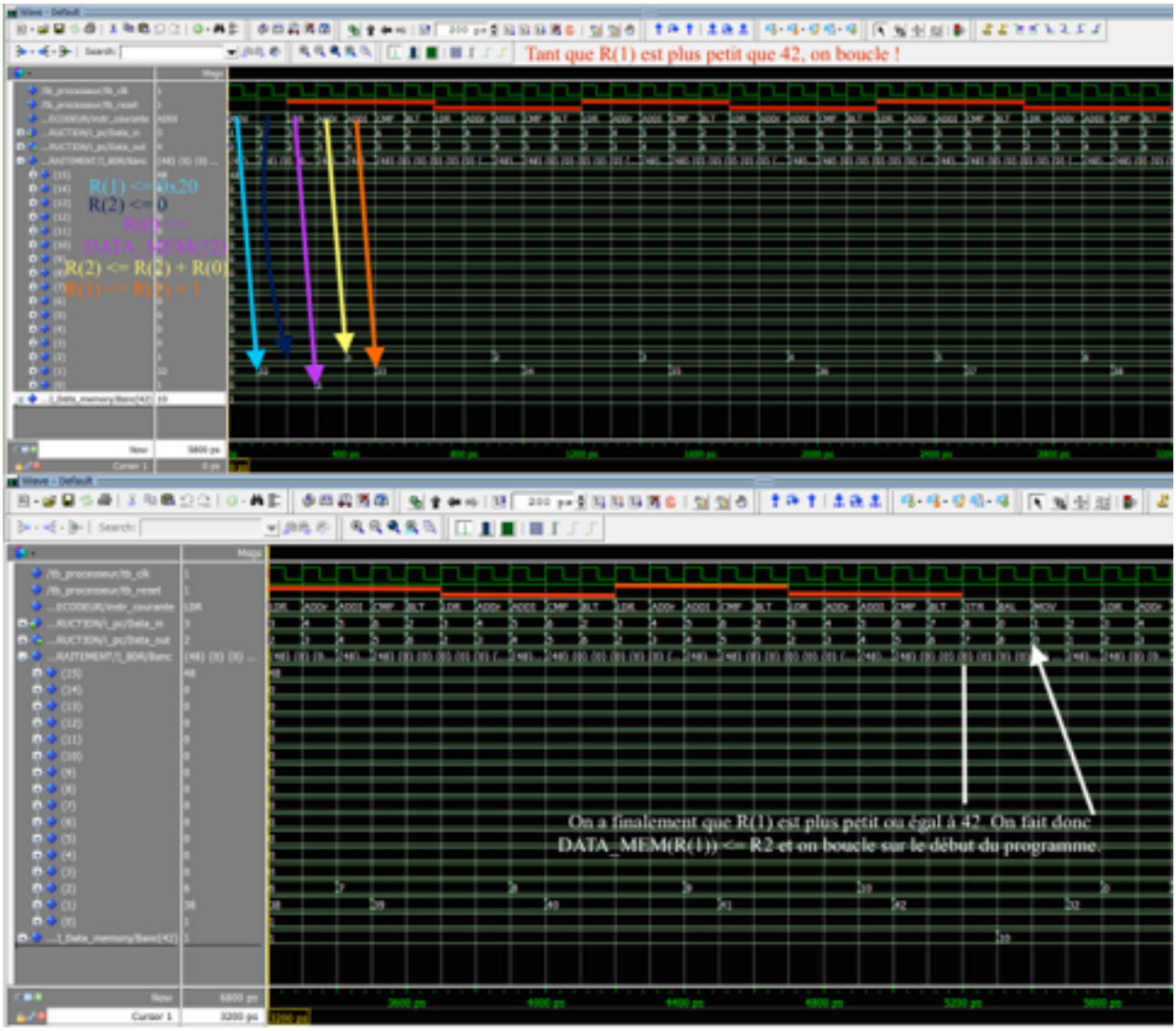


Figure 13 : Simulation du processeur

En disséquant encore plus ce dernier testbench, on observe aussi le bon fonctionnement du registre PSR qui permet de boucler correctement avec le BLT.

Conclusion :

Nous avons finalement décrit et simulé en VHDL un processeur mono-cycle. On remarquera qu'il a fallu une attention toute particulière pour la clarté des informations, et comme toujours, écrire sur papier en premier s'est avéré un gain de temps considérable. En plus de confirmer certaines notions de VHDL, ce tp nous a aussi permis de faire le lien avec le module d'architecture des ordinateurs de l'année dernière, nous apportant ainsi une encore meilleure compréhension des mécanismes inhérents au fonctionnement des systèmes embarqués.