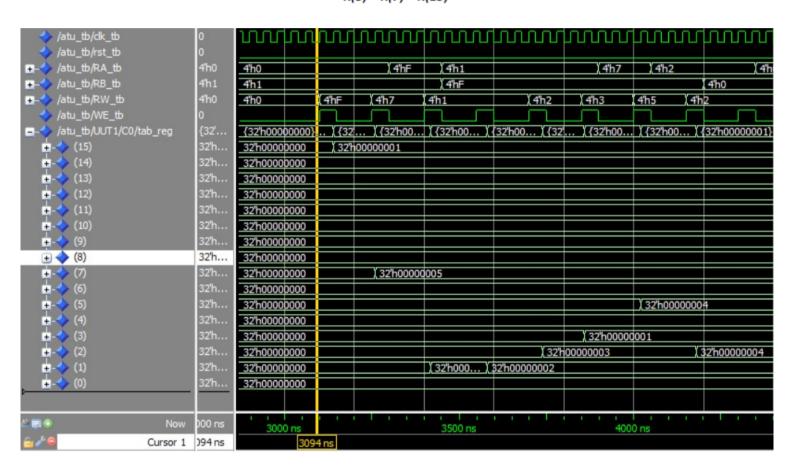
Architecture des processeurs

Processeur Monocycle Simulation en VHDL

Test de L'Unité de Logique (ALU)

Nous exécutons les commandes suivantes en ayant préalablement initialisé les registres R(1) et R(7) aux valeurs 1 et 5 respectivement :

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)



Test de du banc de Registre (REG)

Afin de tester le banc de registre, on vient simplement écrire dans les registre R(0), R(1), et R(5) les valeurs respectives 1, 5, et 55. Puis on remplace la valeur du registre R(1) par 11.

reg_tb/W_tb	32'h00000011	32'h000000000	32'h00000000	32'h000	00005			32'h0000	0055			32'h0000	0011		
+ /reg_tb/RA_tb	4'h0	4'h0					4h1			4h5	4'h0				
+ /reg_tb/RB_tb	4h1	4'h0				4h1	4'h0			4'h5	4h1				
+ /reg_tb/RW_tb	4'h0	4'h0		4h1				4'h5				4'h0			
<pre>/reg_tb/WE_tb</pre>	0														
+ /reg_tb/A_tb	32'h00000011	32'h00000000	(32'h00000000	1			32'h000	0005		32'h0	32'h0	32'h000	00011		j
reg_tb/B_tb	32'h00000005	32'h000000000	(32'h0000000	1		32'h	32'h000	0001		32'h0	32'h0000	0005			
/reg_tb/UUT1/tab_reg	{32'h000000000}	{32'h000000	({32'h0000000	0 ({32'h00	000000} {	32'h00000	0000}	({32'h000	00000} {	32'h00000	000} {	{32'h00	(00000	{32'h00000	000} {3
<u>+</u> > (15)	32'h00000000	32'h000000000													j
±- (14)	32'h00000000	32'h000000000													į
±- (13)	32'h00000000	32'h000000000													į –
±- (12)	32'h00000000	32'h00000000													j
±- (11)	32'h00000000	32'h000000000													j
±- (10)	32'h00000000	32'h00000000													j
÷	32'h00000000	32'h00000000													j
±- (8)	32'h00000000	32'h000000000													
±- → (7)	32'h00000000	32'h000000000													j
±- (6)	32'h00000000	32'h000000000													j
+> (5)	32'h00000055	32'h000000000						(32'h0000	0055						
+ (4)	32'h00000000	32'h000000000													
+ > (3)	32'h00000000	32'h000000000													
±> (2)	32'h00000000	32'h00000000													
+> (1)	32'h00000005	32'h000000000		(32'h000	00005										i –
1 − (0)	32'h00000011	32'h000000000	(32'h0000000	1								32'h000	00011		
△ 🛒 ④ Now	29000 ns	2000 ns	4000 ns	600	00 ns	800	0 ns	1000	00 ns	1200	00 ns	1400	00 ns	1600	Ons
€ Cursor 1	16158 ns														158 ns

Test du Multiplexeur (MUX)



Test de l'Extension de signe (EXT)

La valeur en entrée sur 8 bits est bien transposée sur 32 bits en sortie.



Test de l'Unité de Traitement (ATU)

Pour tester l'assemblage de l'unité des traitement, on procède au test successif de plusieurs opérations comme décrit ci-dessous en ayant préalablement initialisé les registres R(0) et R(1) avec les valeurs respectives 1 et 3 :

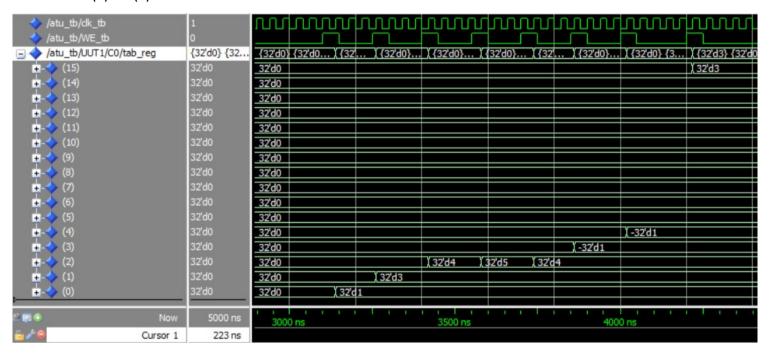
$$R(2) = R(0) + R(1) = 4$$

$$R(2) = R(2) + 1 = 5$$

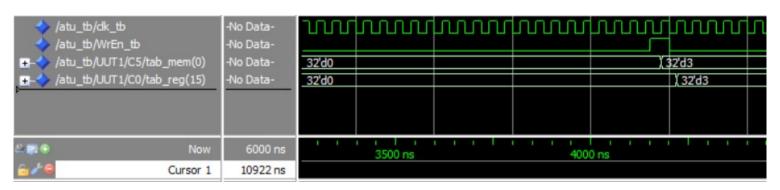
$$R(2) = R(2) - R(0) = 4$$

$$R(3) = R(0) - 2 = -1$$

$$R(4) = R(3) = -1$$

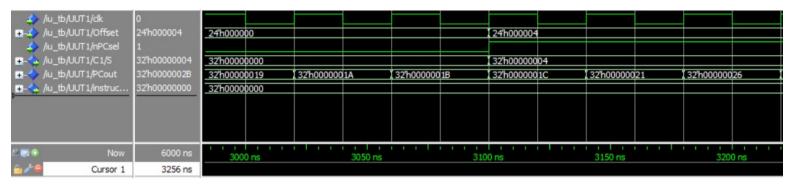


On effectue ensuite une écriture dans la mémoire de données de R(1) = 3 à l'adresse 0. Puis on lit cette donnée qui vient d'être écrite et on la stocke dans R(15).



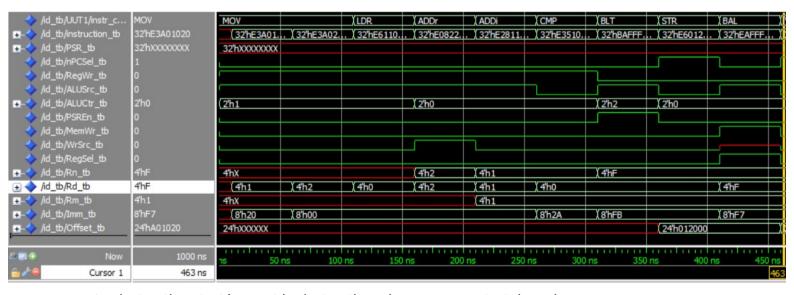
Test de l'Unité de Gestion des Instruction (IU)

Ici on vient vérifier que le pointeur d'instruction en entrée de la mémoire (PCout) d'instruction s'incrémente bien à chaque coup d'horloge. Puis on applique un offset de 4 en entrée :



Test du Décodeur d'Instruction (ID)

On vérifie bien que le décodeur reconnaît la trame de chaque instruction grâce au premier signal qui affiche le nom de l'instruction se trouvant actuellement en entrée (instruction_tb):



Les instructions testées sont les instructions du programme test du main.

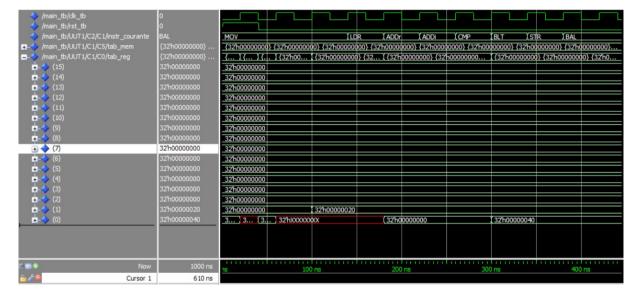
Simulation du Processeur

On a maintenant modifié la mémoire d'instruction pour qu'elle contienne le code binaire des instructions suivantes :

```
MOV R1, #0x20
0 \times 0
     _main :
                MOV R2, #0
0x1
0x2 _loop:
                LDR R0,0(R1)
0x3
                ADD R2, R2, R0
0x4
                 ADD R1, R1, #1
                 CMP R1,0x2A
0x5
0x6
                 BLT loop
R1 inferieur a 0x2A
      _end :
                 STR R2,0(R1)
0x7
0x8
                 BAL main
```

On a aussi initialisé la mémoire de données entre les adresses 0x20 et 0x2A comme suit :

```
result (32):=x"00000001"; -- 0x20
result (33):=x"0000002";
result (34):=x"00000003";
result (35):=x"00000004";
result (36):=x"00000005";
result (37):=x"00000006";
result (38):=x"00000007";
result (39):=x"00000008";
result (40):=x"00000009";
result (41):=x"0000000A";
result (42):=x"0000000B"; -- 0x2A
```



On constate que les instructions sont bien lues mais que la boucle ne s'effectue pas au niveau de l'instruction BLT. Le problème vient d'une erreur dans l'entité Décodeur (ID) ou du fait que l'exécution des instructions est en retard par rapport à leur lecture.

■ Wave - Default :======															/////									
<u>~</u> .	Msgs																							
♦ /main tb/rst tb	0																							
/main_tb/clk_tb	0																							
/main_tb/Done	FALSE																							
/main_tb/UUT1/C1/N	1																							
■→ /main_tb/UUT1/C1/ALUout	-9	0	32				0		32		0		33		-9		34		33		34			
/main_tb/UUT1/C2/C1/instr	CMP	MOV							LDR		ADDr		ADDi		CMP		BLT		STR		BAL			
■	42		32				0					\longrightarrow	1		42				0					
/main_tb/UUT1/C1/C0/tab_reg	{000000000	{0000	000000	000	{0000	000000	00000	000000	000000	000	{0000	000000	000000	0000	{000000	0000	00000	000000	000000	00000}	{0000	000000	000000	00000
₽ ◆ (15)	0	0																						
(14)	0	0																						
‡ → (13)	0	0														▙							_	
□ ♦ (12)	0	0														▙								
± → (11)	0	0																						
	0	0				_										⊨								
(9)	0	0																					_	
(8)	0	0																					=	
★ ◆ (7)	0	0								_						⊨							=	
<u>+</u> → (6)	0	0				_				_														
(5)	0	-				_				_													=	
→ (4) → (3)	0																						=	
(3)	0																							
+ (2) + (1)	0	-			32										33									
(1) (0)	1	-			32					=	,				33									
/main_tb/UUT1/C1/C0/A	33	0							32	=	0		32	_	33	⊨								
	1	_							32		1		32		33				0		1			
	1	0									1								10		1			

On voit que maintenant après modification du code, cela fonctionne, mem[R1] = 0, donc l'addition n'a aucun effet, l'addition en immédiat a été réalisé avec succès et comme R1 < 42 (0x2A), alors on quitte la boucle. Et on a plus ce décalage, donc tout ce qui a été fait précédemment permet de valider.