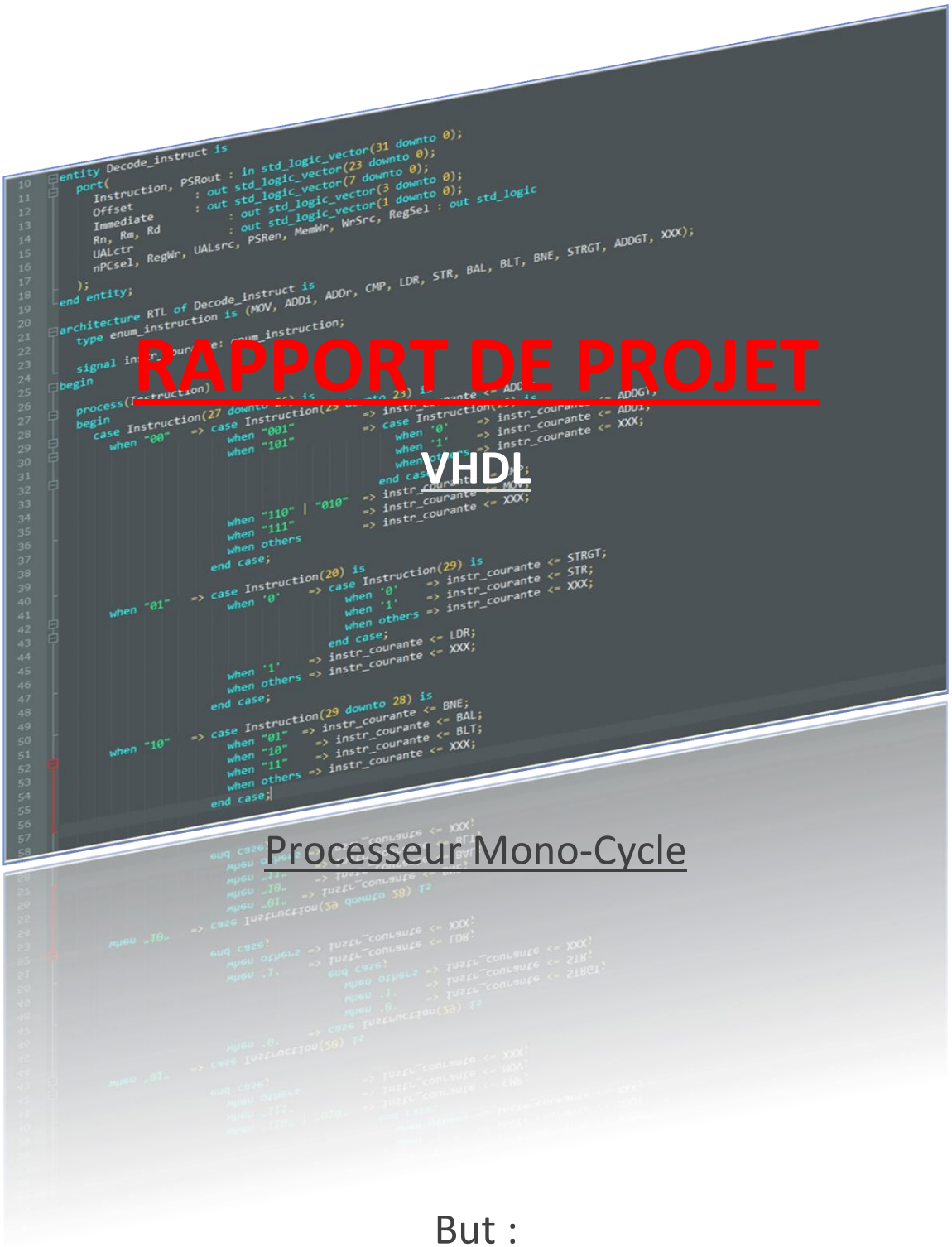
SCIENCES SORBONNE UNIVERSITÉ

POLYTECH SORBONNE



RAPPORT DE PROJET

VHDL

Processeur Mono-Cycle

But :

Conception et Simulation d'un cœur de processeur

ROYER Jérémy EI-SE 3

Introduction

Ce projet consistait à réaliser un processeur monocycle grâce à des briques de base comme des registres, des multiplexeurs ou encore des bancs de mémoire. La description des briques se faisait en VHDL comportemental. Chaque brique était ensuite assemblée avec d'autres afin de créer trois unités principales. L'unité de Traitement permettant de traité une instruction donnée, l'unité de gestion des instructions permettant de choisir l'instruction à traiter et la dernière unité est l'unité de contrôle permettant de décoder l'instruction pour l'unité de traitement. Ainsi le projet se décline en plusieurs parties, les trois premières sont dédiées à la réalisation de ces trois unités puis une quatrième partie permet l'assemblage des unités pour donner vie au processeur puis deux parties suivent permettant de perfectionner le processeur en ajoutant des instructions.

Partie 1 – Unité de Traitement

Dans cette partie je vais réaliser l'unité de traitement. Cette unité sera composée d'un banc de registres, d'une unité arithmétique et logique et d'une mémoire de données. Il y aura bien évidemment quelques composants de liaison qui sont deux multiplexeurs et une extension de signe.

Unité Arithmétique et Logique - UAL

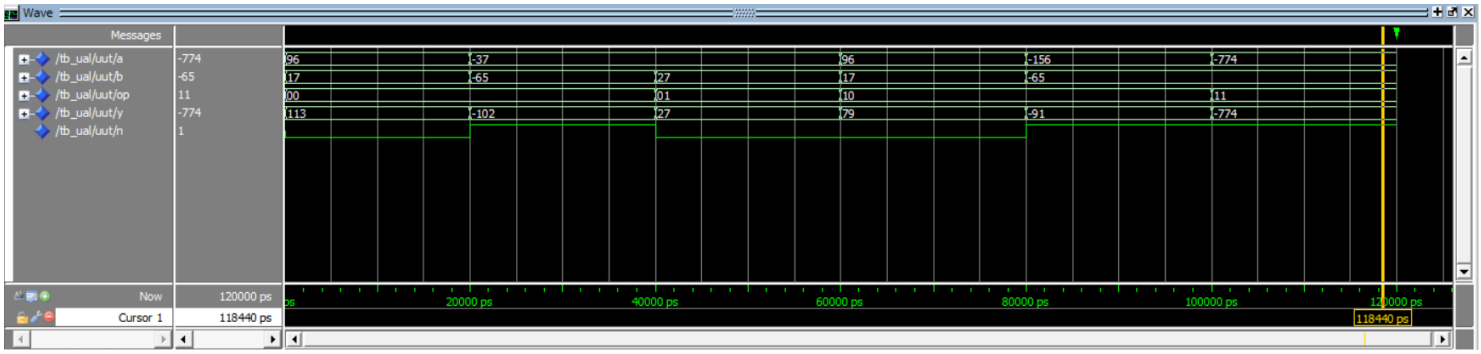


Figure 1 : Banc de Test de l'Unité Arithmétique et Logique

On observe sur la figure 1 que l'UAL fonctionne correctement autant pour faire une addition, qu'une recopie ou qu'une soustraction.

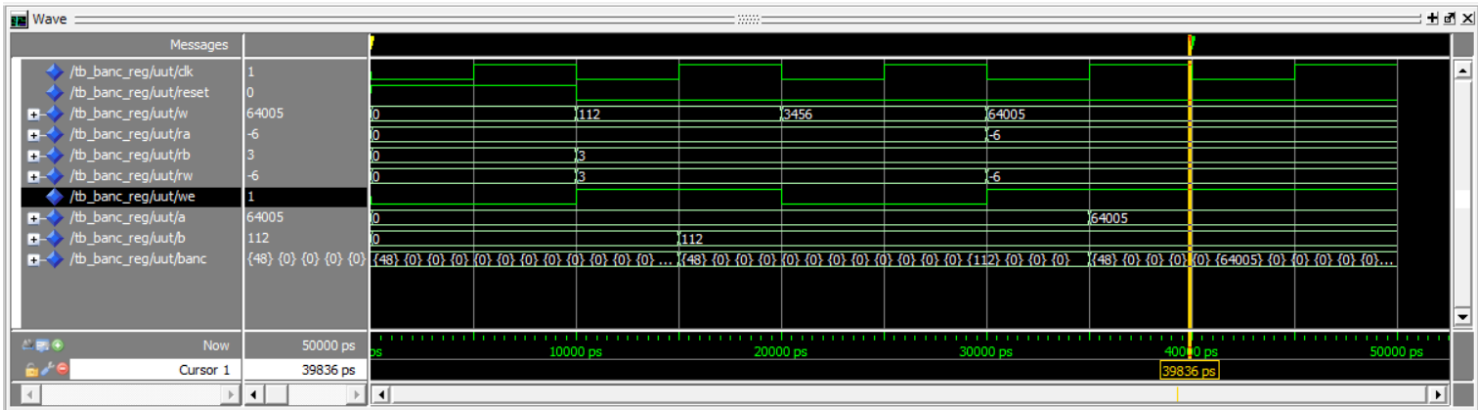


Figure 2 : Banc de Test du banc de registres

On peut voir sur la figure 2 la simulation du banc de 16 registres de 32 bits. On remarque que le banc fonctionne correctement.

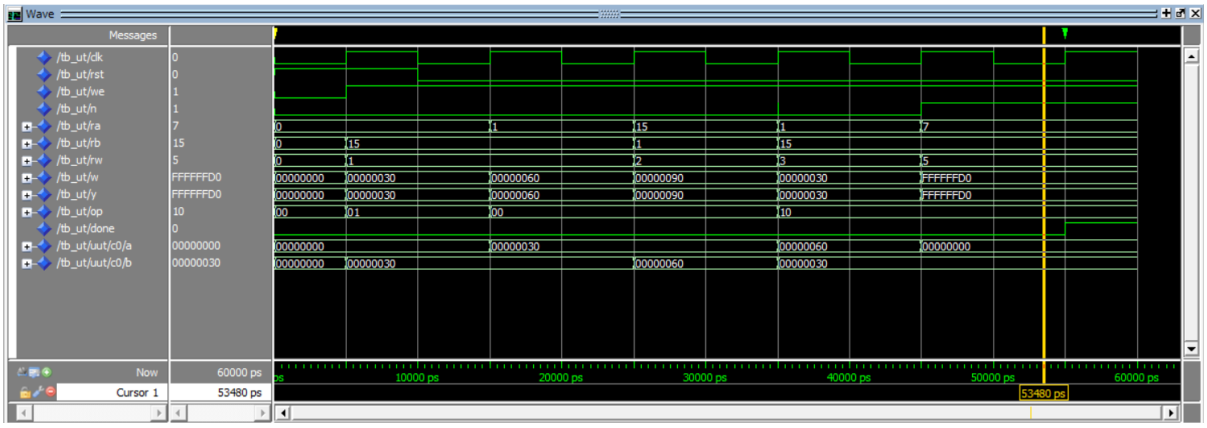


Figure 3 : Banc de Test de l'Unité de traitement intermédiaire

Sur la figure 3, on observe la simulation de l'unité de traitement intermédiaire avec son fonctionnement correct.

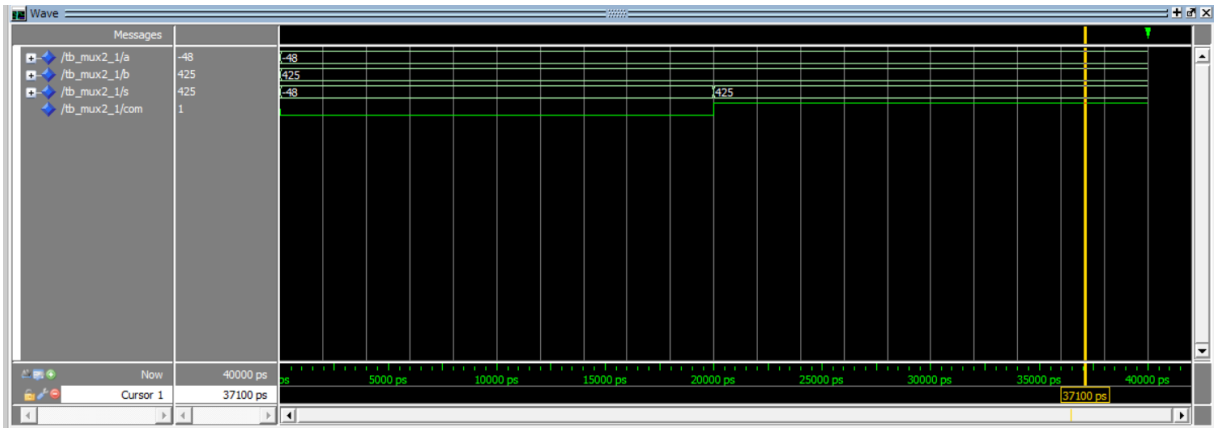


Figure 4 : Banc de Test d'un Multiplexeur 2 vers 1

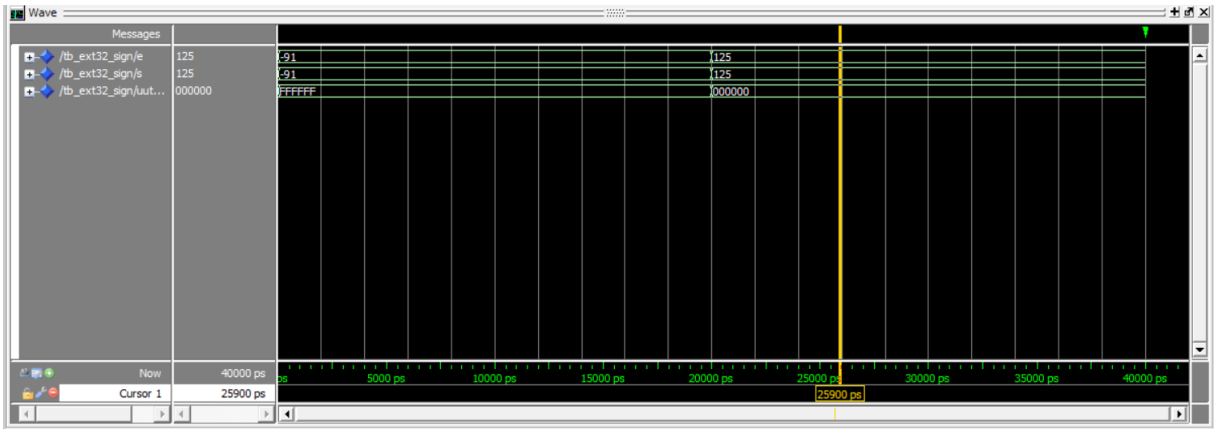


Figure 5 : Banc de Test d'un Extendeur de signe

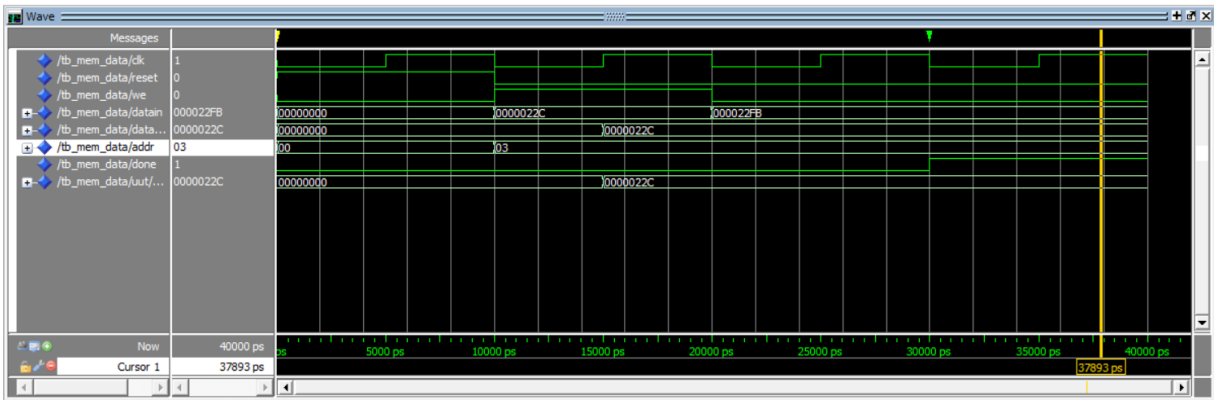


Figure 6 : Banc de Test d'une Mémoire de données

Grâce à la figure 6, on observe que cette mémoire fonctionne correctement et permet de charger et stocker des mots sur 32 bits.

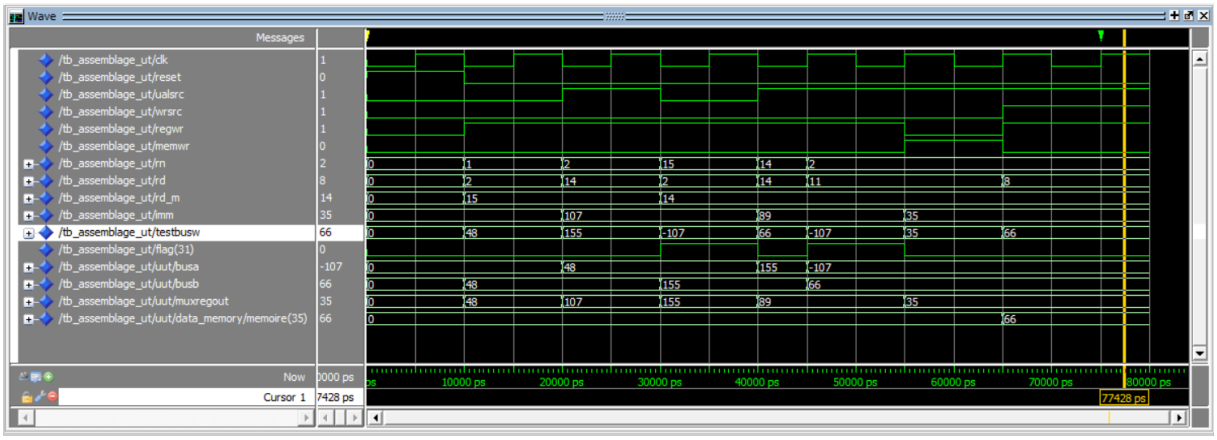


Figure 7 : Banc de Test de l'Assemblage de l'Unité de Traitement

La figure 7 représente le dernier Banc de Test de la partie 1. Il permet d'observer le bon fonctionnement de l'unité de Traitement en entier avec le bon comportement en fonction des signaux d'entrées.

PARTIE 2 – UNITE DE GESTION DES INSTRUCTIONS

Figure 8 : Banc de Test de l'Unité de gestion des instructions

Grâce à la figure 8, on observe le fonctionnement correct de l'unité de gestion qui permet d'envoyer la bonne instruction en fonction du signal nPCsel et de l'offset.

PARTIE 3 – UNITE DE CONTROLE

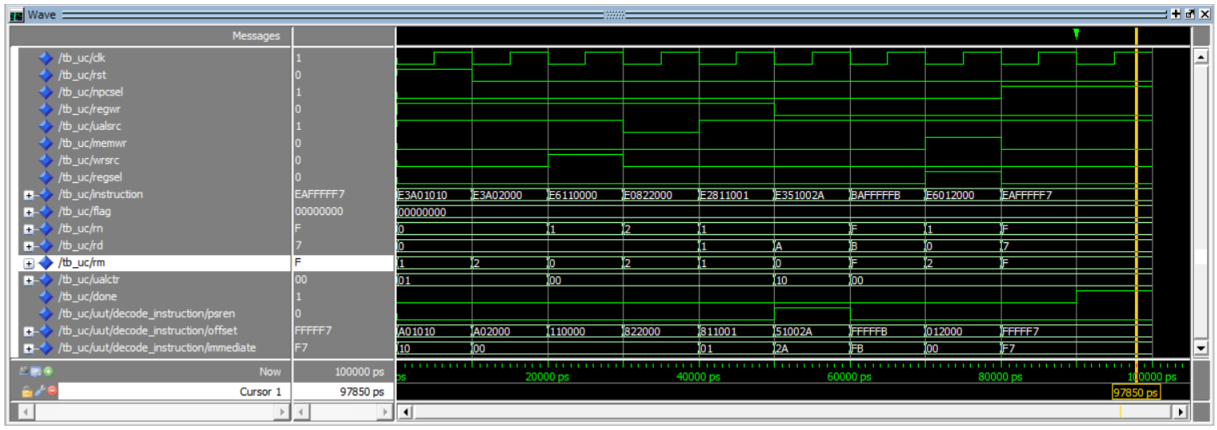


Figure 9 : Banc de Test de l'Unité de Contrôle

Cette figure représente la simulation de l'unité de contrôle qui permet en fonction de l'instruction courante générer les signaux correspondant pour l'unité de traitement. On peut remarquer que cette simulation fonctionne correctement.

PARTIE 4 – ASSEMBLAGE ET VALIDATION DU PROCESSEUR

```
begin
   for i in 63 downto 0 loop
      result (i):=(others=>'0');
   end loop;              -- PC        -- INSTRUCTION   -- COMMENTAIRE
      result (0):=x"E3A01010";-- 0x0 _main -- MOV R1,#0x10 -- R1 = 0x10
      result (1):=x"E3A02000";-- 0x1       -- MOV R2,#0x00 -- R2 = 0
      result (2):=x"E6110000";-- 0x2 _loop -- LDR R0,0(R1) -- R0 = DATAMEM[R1]
      result (3):=x"E0822000";-- 0x3       -- ADD R2,R2,R0 -- R2 = R2 + R0
      result (4):=x"E2811001";-- 0x4       -- ADD R1,R1,#1 -- R1 = R1 + 1
      result (5):=x"E351002A";-- 0x5       -- CMP R1,0x1A  -- si R1 >= 0x1A
      result (6):=x"BAFFFFFB";-- 0x6       -- BLT loop     -- PC = PC + 1 + (-5) si N = 1
      result (7):=x"E6012000";-- 0x7       -- STR R2,0(R1) -- DATAMEM[R1] = R2
      result (8):=x"EAFFFFF7";-- 0x8       -- BAL main  -- PC = PC + 1 + (-9)
   return result;
end init_mem;
```

Figure 10 : Code hexadécimal de la mémoire d'instruction

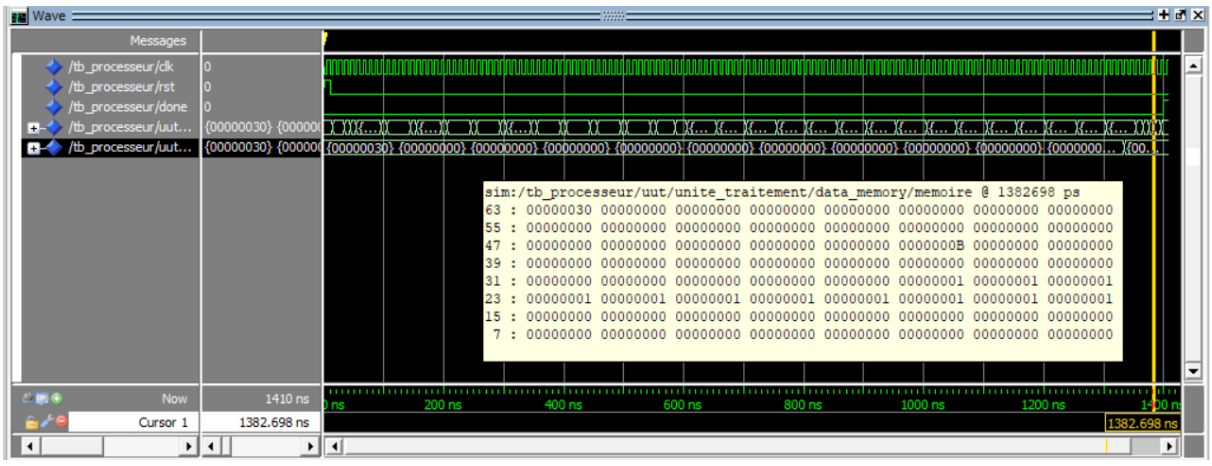


Figure 11 : Banc de Test de l'Assemblage Processeur

Après la traduction en binaire puis en hexadécimal du petit programme en commentaire permettant d'additionner dans le registre 42 la somme des 16 à 26. On peut voir le bon fonctionnement en simulation de ce programme.

PARTIE 5 – TEST COMPLET DU PROCESSEUR

```
for i in 63 downto 0 loop
   result (i):=(others=>'0');
end loop;                    -- PC          -- INSTRUCTION   -- COMMENTAIRE
   result (0) :=x"E3A00010";-- 0x0 _main -- MOV R0,#0x10 -- R0 = 0x10
   result (1) :=x"E3A01001";-- 0x1       -- MOV R1,#1    -- R1 = 0
   result (2) :=x"E6103000";-- 0x2 _for  -- LDR R3,0(R1) -- R3 = DATAMEM[R1]
   result (3) :=x"E6104001";-- 0x3       -- LDR R4,R0,#1 -- R4 = DATAMEM[R0+1]
   result (4) :=x"E6004000";-- 0x4       -- STR R4,R0    -- DATAMEM[R0] = R4
   result (5) :=x"E6003001";-- 0x5       -- STR R3,R0,#1 -- DATAMEM[R0+1] = R3
   result (6) :=x"E2800001";-- 0x6       -- ADD R0,R0,#1 -- R0 = R0 + 1
   result (7) :=x"E2811001";-- 0x7       -- ADD R1,R1,#1 -- R1 = R1 + 1
   result (8) :=x"E351000A";-- 0x8       -- CMP R1, #0xA -- Si R1 < 10
   result (9) :=x"BAFFFFF8";-- 0x9       -- BLT FOR   -- PC = PC + 1 + (-8)
   result (10):=x"EAFFFFFF";-- 0xA _wait -- BAL wait   -- PC = PC + 1 + (-1)
return result;
```

Figure 12 : Code hexadécimal de la deuxième mémoire d'instructions

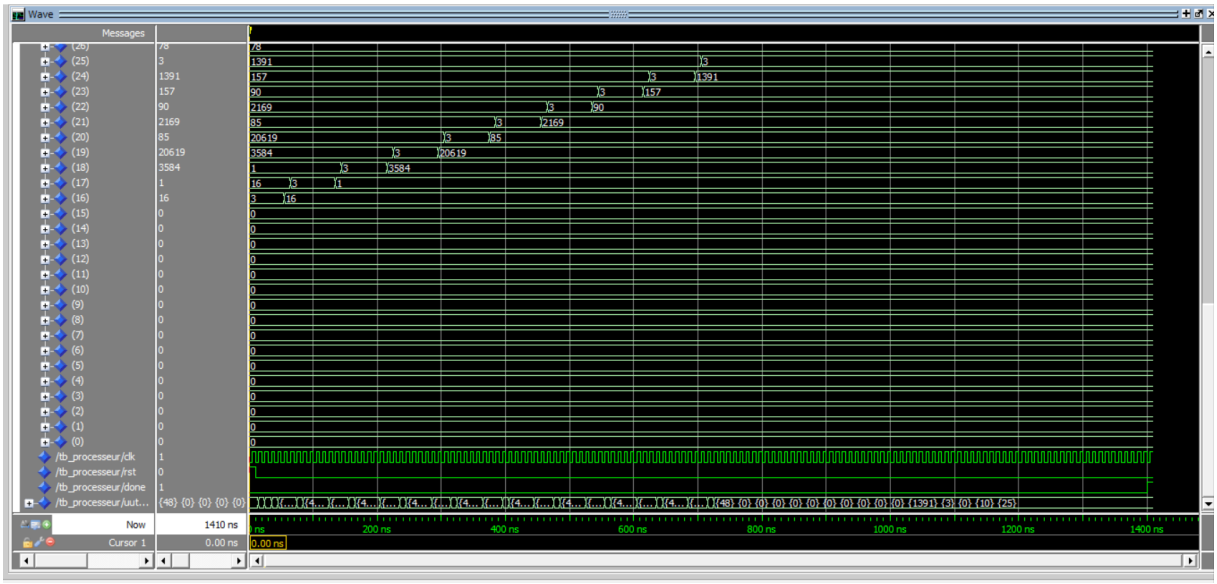


Figure 13 : Banc de Test de la deuxième mémoire d'instruction

Après la traduction du second programme en binaire puis en hexadécimal permettant de faire remonter la valeur du « premier » registre (n°16) dans le « dernier » registre (n°25). On observe que le résultat du banc de test est correct.

PARTIE 6 – AUGMENTATION DU JEU D'INSTRUCTION

```
for i in 63 downto 0 loop
   result (i):=(others=>'0');
end loop;                  -- PC          -- INSTRUCTION       -- COMMENTAIRE
   result (0) :=x"E3A00020";-- 0x0 _start-- MOV R0,#0x20       -- R0 = 0x20
   result (1) :=x"E3A02001";-- 0x1        -- MOV R2,#1         -- R2 = 1
   result (2) :=x"E3A02000";-- 0x2 _while-- MOV R2,#0          -- R2 = 0
   result (3) :=x"E3A01001";-- 0x3        -- MOV R1,#1         -- R1 = 1
   result (4) :=x"E6103000";-- 0x4 _for   -- LDR R3,[R0]       -- R3 = DATAMEM[R0]
   result (5) :=x"E6104001";-- 0x5        -- LDR R4,R0,#1      -- R4 = DATAMEM[R0+1]
   result (6) :=x"E1530004";-- 0x6        -- CMP R3, R4        -- si R3 < R4
   result (7) :=x"C6004000";-- 0x7        -- STRGT R4,[R0]     -- DATAMEM[R0] = R4
   result (8) :=x"C6003001";-- 0x8        -- STRGT R3,[R0,#1] -- DATAMEM[R0+1] = R3
   result (9) :=x"C2822001";-- 0x9        -- ADDGT R2,R2,#1    -- R2 = R2 + 1
   result (10):=x"E2800001";-- 0xA         -- ADD R0, R0, #1    -- R0 = R0 + 1
   result (11):=x"E2811001";-- 0xB         -- ADD R1, R1, #1    -- R1 = R1 + 1
   result (12):=x"E3510007";-- 0xC         -- CMP R1, #0x07     -- si R1 < 7
   result (13):=x"BAFFFFF6";-- 0xD         -- BLT FOR          -- PC = PC + 1 + (-10)
   result (14):=x"E3520000";-- 0xE         -- CMP R2, #0         -- si R2 < 0
   result (15):=x"E3A00020";-- 0xF         -- MOV R0, #0x20     -- R0 = 0x20
   result (16):=x"1AFFFFF1";-- 0x10        -- BNE WHILE         -- PC = PC + 1 + (-15)
   result (17):=x"EAFFFFFF";-- 0x11 _wait-- BAL wait          -- PC = PC + 1 + (-1)
return result;
```

Figure 14 : Code hexadécimal de la dernière mémoire d'instructions

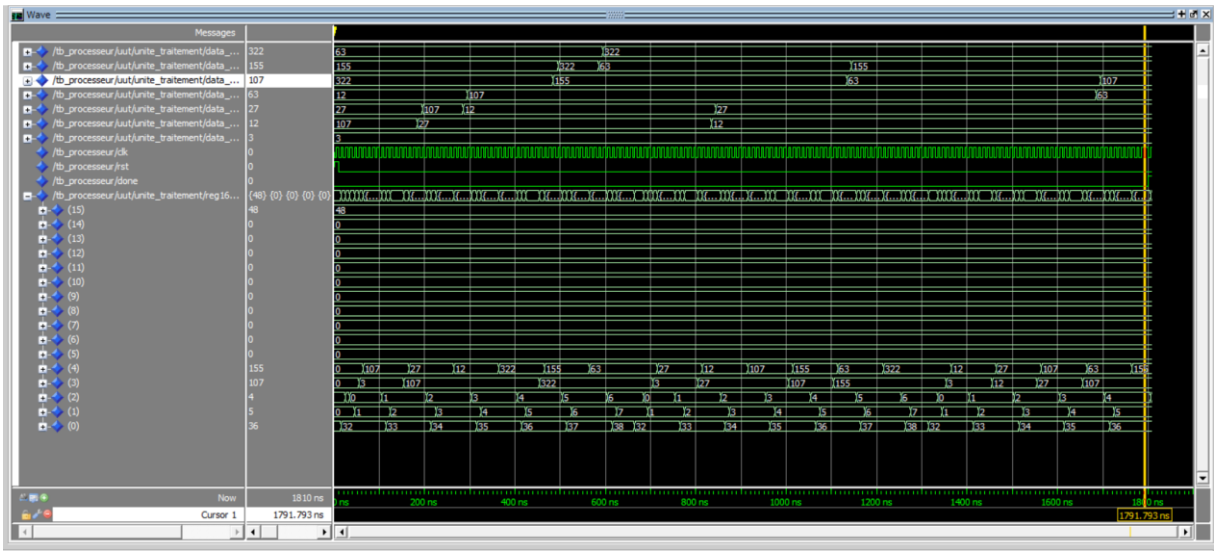


Figure 15 : Banc de Test de la dernière mémoire d'instruction

Ces deux dernières figures montrent tout d'abord la traduction en hexadécimal d'un programme permettant d'effectuer un tri à bulles entre 7 registres (n°32 -> n°38), puis la deuxième figure montre la bonne exécution de ce programme.

Conclusion

À travers ce projet, j'ai pu construire un processeur monocycle grâce à trois sous unités, elles-mêmes composées de sous bloques. Grâce à l'augmentation du jeu d'instruction, le processeur a été capable de réaliser un algorithme de tri. Pour le rendre encore plus efficace et capable d'accueillir un plus grand nombre de programme, on pourrait de nouveau ajouter des instructions.