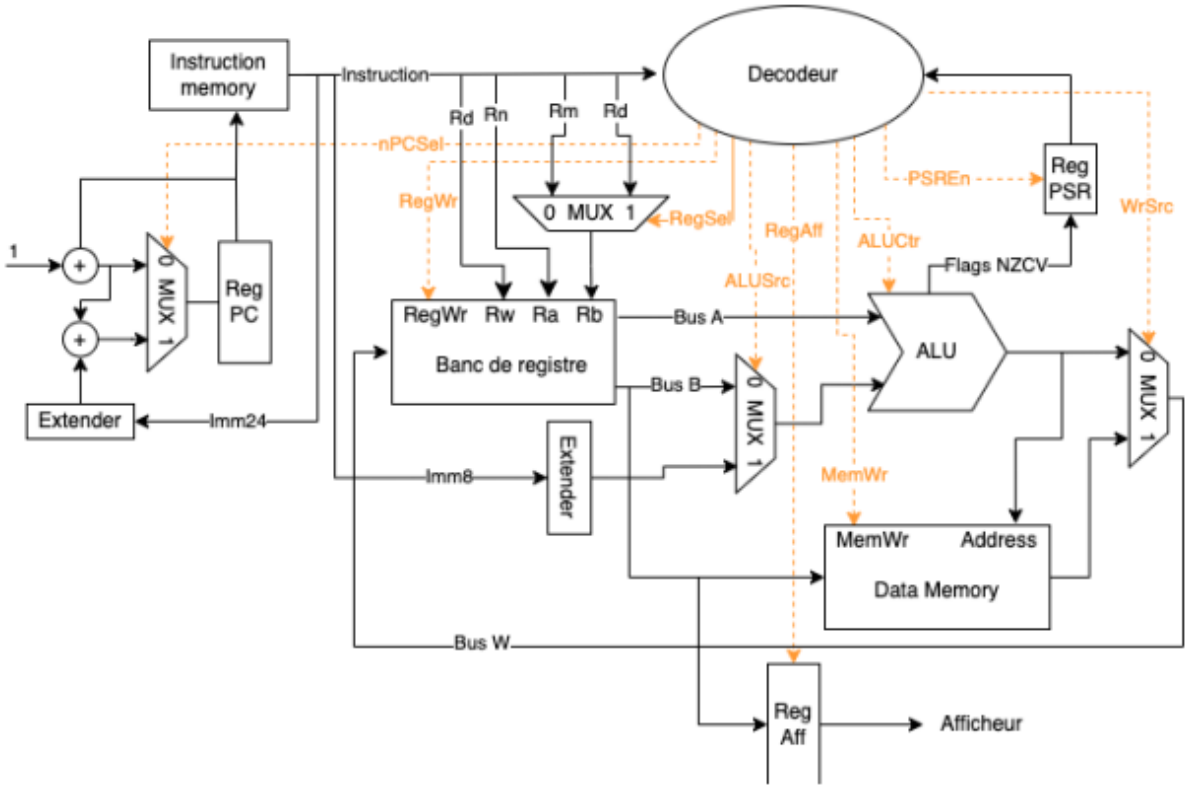
NAVRATIL Adrien
THIBERT Adam

GISTRE 2024

VHDL - CPU ARM7TDMI

Objectif:

- Concevoir, synthétiser et valider une IP numérique décrite en VHDL RTL



Bancs de tests:

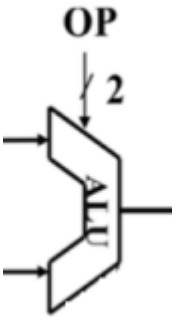
Les bancs de tests (qui se situent dans le dossier *simu/* de l'architecture et finissants par l'extension de fichier *vhd*) ont été écrits en utilisant des directives du framework *VUnit*, il faut donc que le logiciel soit installé afin de les compiler et de les simuler.
Un fichier *README.md* est fourni avec des instructions plus précises.

NAVRATIL Adrien **GISTRE 2024**
THIBERT Adam

1. Unité de Traitement

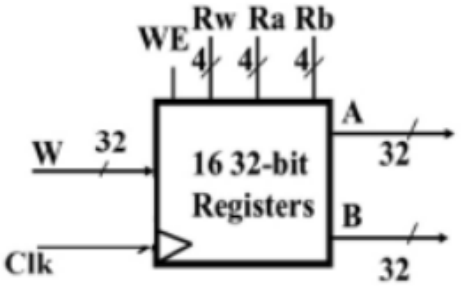
L'unité de traitement d'un processeur **(src/processing_unit.vhd)** comprend une Unité Arithmétique et Logique (UAL) pour les opérations mathématiques et logiques, un banc de registres, une mémoire de données, ainsi que des composants auxiliaires tels que des extensions de signe et des multiplexeurs afin de sélectionner l'une des multiples entrées possibles et la diriger vers une sortie spécifique en fonction des signaux de commande. Ces éléments travaillent ensemble pour exécuter les instructions du programme et effectuer les calculs nécessaires au fonctionnement du processeur.

Composants :

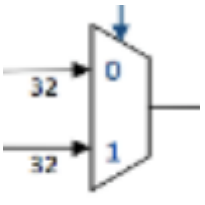


- **Unité Arithmétique et Logique (src/components/alu.vhd)**
  L'unité Arithmétique et Logique se compose de 2 entrées de données et une entrée de contrôle qui spécifie l'opération à effectuer entre les 2 opérandes.
  En sortie, on retrouve le résultat de l'opération effectuée ainsi que 4 signaux appelés "flags", donnant des informations sur l'opération qui a été effectuée, tel que le signe si la donnée est interprétée comme un nombre signé, etc…

- **Banc de registre (src/components/file_register.vhd)**
  Le banc de registres contient 16 registres de 32 bits. Il dispose d'un signal d'écriture pour écrire des données dans un registre sélectionné, ainsi que de deux signaux en lecture pour récupérer les données stockées dans les registres sélectionnés. Les signaux de sélection permettent de spécifier le registre à lire ou à écrire et d'activer ou non l'écriture. En utilisant ces signaux, les données peuvent être déplacées entre les registres et les autres composants du processeur, tels que l'UAL et la mémoire de données.
  Ce composant possède un signal d'horloge afin de réaliser les opérations de manière synchrone. Néanmoins il possède un signal de Reset asynchrone.

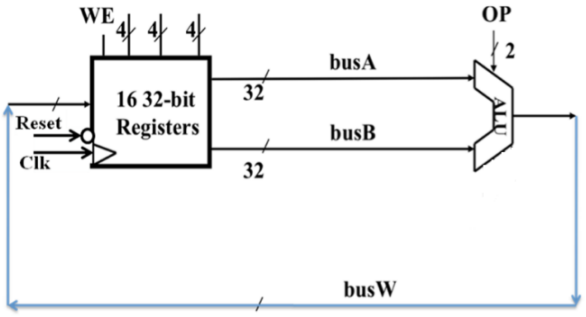


- **Multiplexeur 2 vers 1 (src/components/mux_21.vhd)**
  Un multiplexeur 2 vers 1 est un composant qui permet de choisir entre deux entrées de données et de diriger celle sélectionnée vers une sortie. Son fonctionnement est basé sur un signal de commande qui détermine quelle entrée sera transmise à la sortie.
  Ce multiplexeur sert à choisir si l'on veut une valeur immédiate ou une valeur provenant d'un registre comme seconde opérande de l'UAL.



- **Extension de signe (src/components/sign_extender.vhd)**
  Une extension de signe est utilisée pour étendre la taille d'un nombre binaire signé tout en préservant son signe. Elle fonctionne en répétant le bit de signe lors de l'extension vers des bits supplémentaires. Cela permet d'assurer la cohérence du signe lors des opérations ultérieures.
  Dans le projet, on utilisera plusieurs extensions de signe afin de ramener différents signaux sur 32 bits.

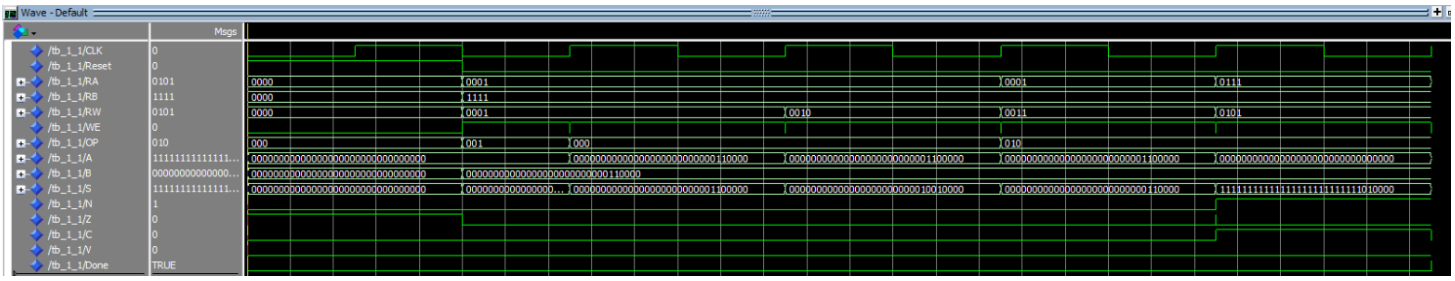
- **Mémoire de données (src/components/data_memory.vhd)**
  La mémoire de données se comporte de la même manière que le Banc de registre mais possède cette fois-ci 64 espaces mémoire de 32 bits et seulement un signal de sortie qui prend la valeur de la donnée à l'adresse spécifiée en entrée.
  Ce composant sert à stocker des données de manière générale contrairement aux registres qui sont là pour stocker des valeurs temporaires pour des opérations courantes lors de l'exécution d'un programme.



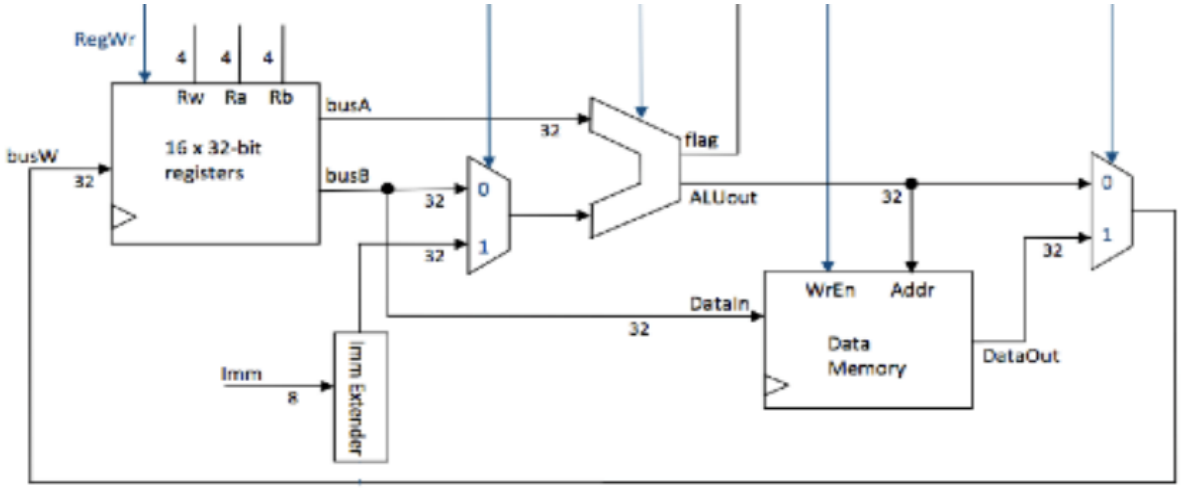
<u>Simulation:</u>

- **UAL et Banc de registre**
  Tout d'abord, nous avons réalisé un banc de test **(simu/tests_1_1.vhd)** simple uniquement sur l'UAL et le banc de registre afin de tester les différentes opérations facilement. Le signal de clock s'alterne toutes les 5 millisecondes, ainsi un cycle dure 10 millisecondes.

  Voici les différents tests réalisés, à chaque fois nous vérifions d'abord la valeur présente dans le registre, puis nous modifions nos signaux afin de sélectionner les bons registres et la bonne opération puis nous vérifions que la valeur écrite dans le banc de registre est correcte:
  - 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) [95 - 48 = 48]
  - R(5) = R(7) – R(15) [0 - 48 = 4294967248]



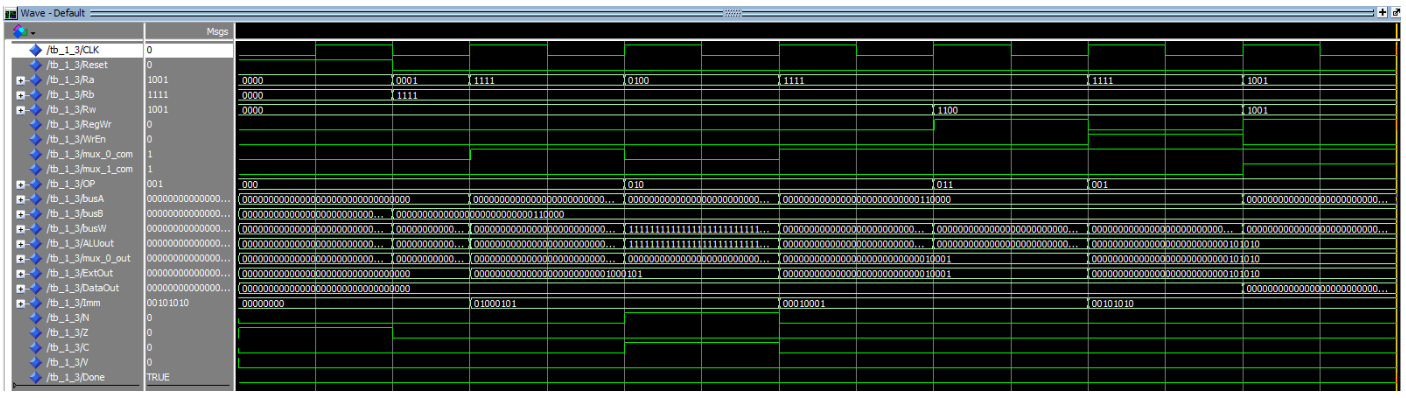
- **Unité de Traitement complète**
  Ensuite nous avons assemblé tous les composants nécessaires à l'unité de traitement (UAL, Banc de registre, Mémoire de données, Valeur immédiate, etc…) et avons réalisé un banc de test **(simu/tests_1_3.vhd)** simulant un usage plus complexe et complet de l'unité de traitement.



Afin de tester correctement l'unité de traitement, voici les tests que nous avons effectué (toutes les valeurs valent 0 initialement, sauf R(15) qui est initialisé à 48):

- L'addition des registres R(1) et R(15)
  - R(1) = 0 + 48 = 48
- L'addition du registre R(1) avec la valeur immédiate 69
  - R(1) = 48 + 69 = 117
- La soustraction des registres R(4) et R(15)
  - R(4) = 0 - 48 = 4294967248
- La soustraction de la valeur immédiate 17 au registre R(15)
  - R(15) = 48 - 17 = 31
- La copie de la valeur du registre R(15) dans le registre R(12)
- L'écriture du registre R(15) dans la mémoire à l'adresse 42
- La lecture d'un mot de la mémoire à l'adresse 42 dans le registre R(9)

On observe aussi sur le graphique que les valeurs des flags N, Z, C et V se mettent correctement à jour.

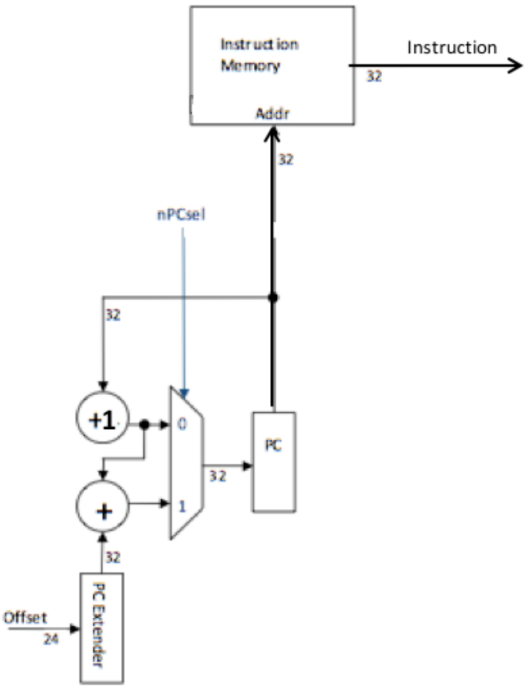


2. Unité de gestion des instructions

L'unité de gestion des instructions **(src/operation_unit.vhd)** possède une mémoire d'instructions contenant un programme d'exemple, et un registre PC de 32 bits sur lequel un offset de 24 bits peut-être appliqué, contrôlé par un multiplexeur.

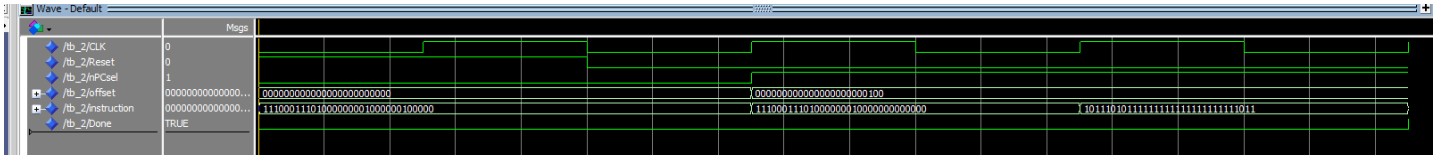
Composants:

- **Mémoire d'instruction (src/components/instruction_memory.vhd)**
  Contient un programme d'exemple de 9 instructions additionnant les cases 32 à 42 de la mémoire de données, avant de stocker le résultat.
- **Registre PC**
  Registre de 32 bits contenant l'index de l'instruction courante du programme. Ce composant est contrôlé par une clock et un reset.
- **Multiplexeur 2 vers 1 (src/components/mux_21.vhd)**
  Ce multiplexeur est à 0 lorsque le PC doit être incrémenté de 1, ou 1 lorsqu'il doit être augmenté de 1 + l'offset. C'est le registre nPCsel qui le contrôle.
- **Sign extender (src/components/sign_extender.vhd)**
  L'offset reçu est sur 24 bits, le sign extender est utilisé pour le passer sur 32 bits pour pouvoir l'ajouter au PC.



Simulation:

Pour cette partie, nous avons écrit un banc de test **(simu/tests_2.vhd)** qui vérifie la première instruction, puis la deuxième après un cycle avec nPCsel à 0. Ensuite, on définit nPCsel à 1 et l'offset à 4, et on vérifie que l'on se trouve bien à la 7ème instruction après un cycle. On peut voir le résultat sur ModelSim :



3. Unité de contrôle

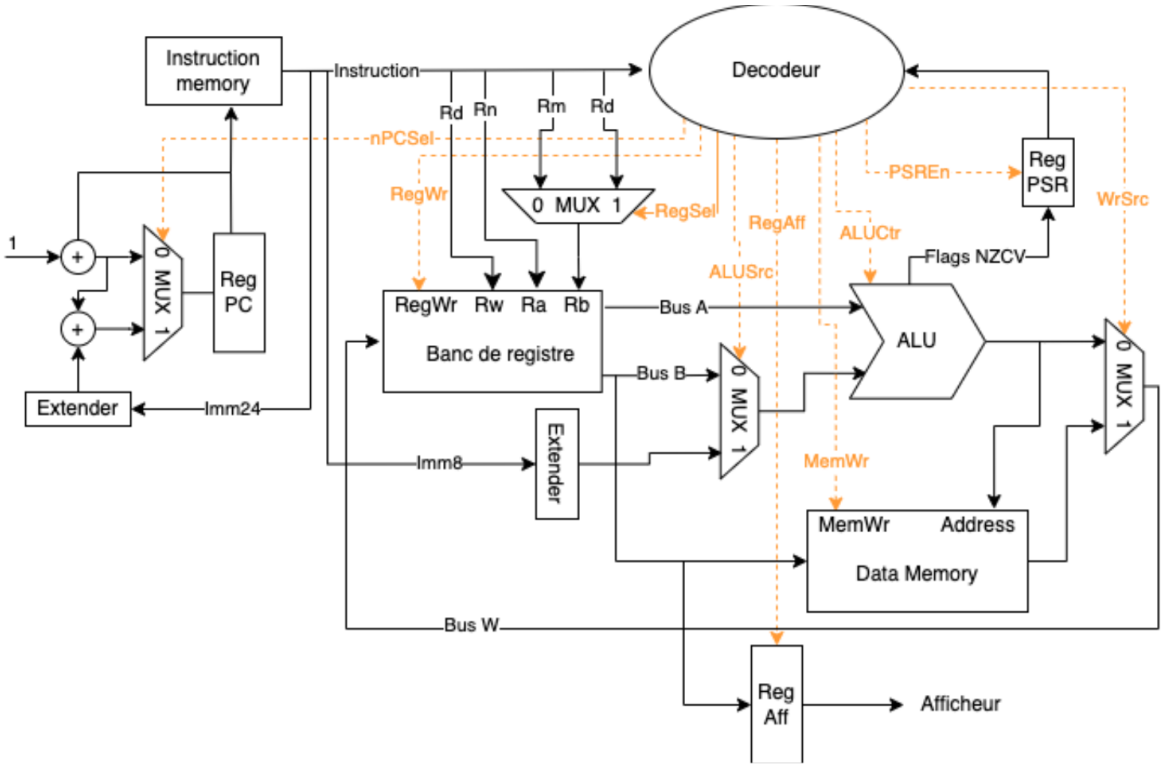
L'unité de contrôle **(src/control_unit.vhd)** comprend le registre PSR ainsi que le décodeur d'instructions.

Composants:

- **Registre PSR (src/components/state_register.vhd)**
  Un registre 32 bits qui contient les valeurs des flags N, Z, C et V.
- **Décodeur d'instruction (src/components/instruction_decoder.vhd)**
  L'entité responsable du décodage des instructions du programme : il va définir pour une instruction donnée les différents signaux passés aux autres composants :

INSTRUCTION	nPCSel	RegWr	ALUSrc	ALUCtr	PSREn	MemWr	WrSrc	RegSel	RegAff
ADDi	0	1	1	000	1	0	0	X	0
ADDr	0	1	0	000	1	0	0	0	0
BAL	1	0	X	X	0	0	X	X	0
BLT	N	0	X	X	0	0	X	X	0
CMP	0	0	1	010	1	0	X	X	0
LDR	0	1	1	000	0	0	1	X	0
MOV	0	1	1	001	0	0	0	X	0
STR	0	0	1	000	0	1	X	1	1

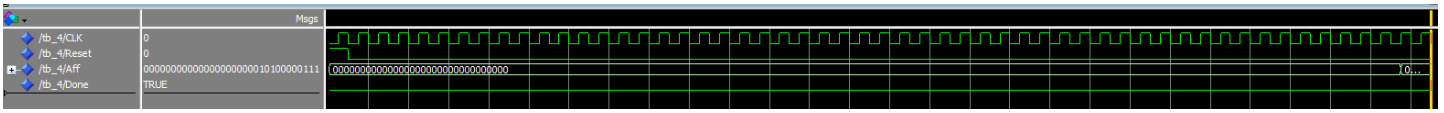
Le schéma global du processeur est le suivant :



4. Assemblage du processeur

Pour l'assemblage du processeur **(src/cpu.vhd)** nous avons rassemblé les unités précédentes, ainsi qu'un registre 32 bits 'RegAff' recevant des données via l'instruction STR.

Simulation :
Pour la simulation, nous avons mis quelques nombres dans les cases 0x27, 0x28 et 0x29 de la mémoire de données avec respectivement 0x1, 0x7 et 0x4FF, puis créé un banc de test **(simu/tests_4.vhd)** attendant 53 cycles pour que le programme s'exécute complètement, puis vérifiant que le contenu de l'afficheur est bien 0x507 :



5. Programmation sur FPGA

Pour programmer la carte FPGA, nous avons créé une entité top level **(src/top_level.vhd)** ainsi qu'un décodeur BCD **(src/components/seven_seg_decoder.vhd)**.

L'entité top level instancie le CPU et 4 décodeurs BCD, et se lie aux pins de la FPGA, avec notamment le bouton 0 comme reset. Le résultat est ensuite affiché sur les 4 premiers chiffres de l'afficheur 7-segment de la carte : 0x507 ou 1287 en décimal :