Compte-rendu du TD1 :

I) Analyse de lôarchitecture de type « Banc de registres » :

1) Le diagramme décrivant lôentité « register_file » est le suivant :

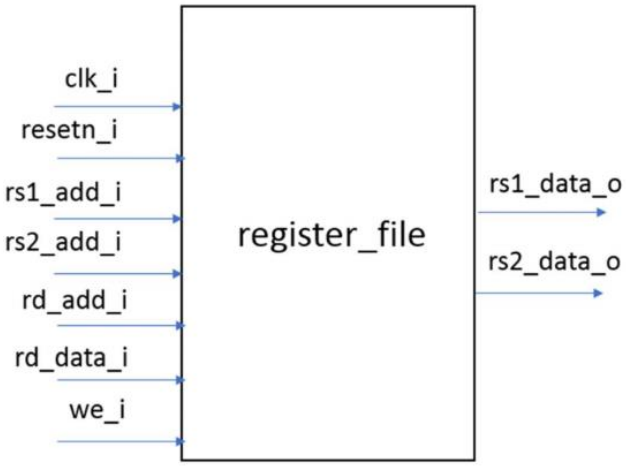
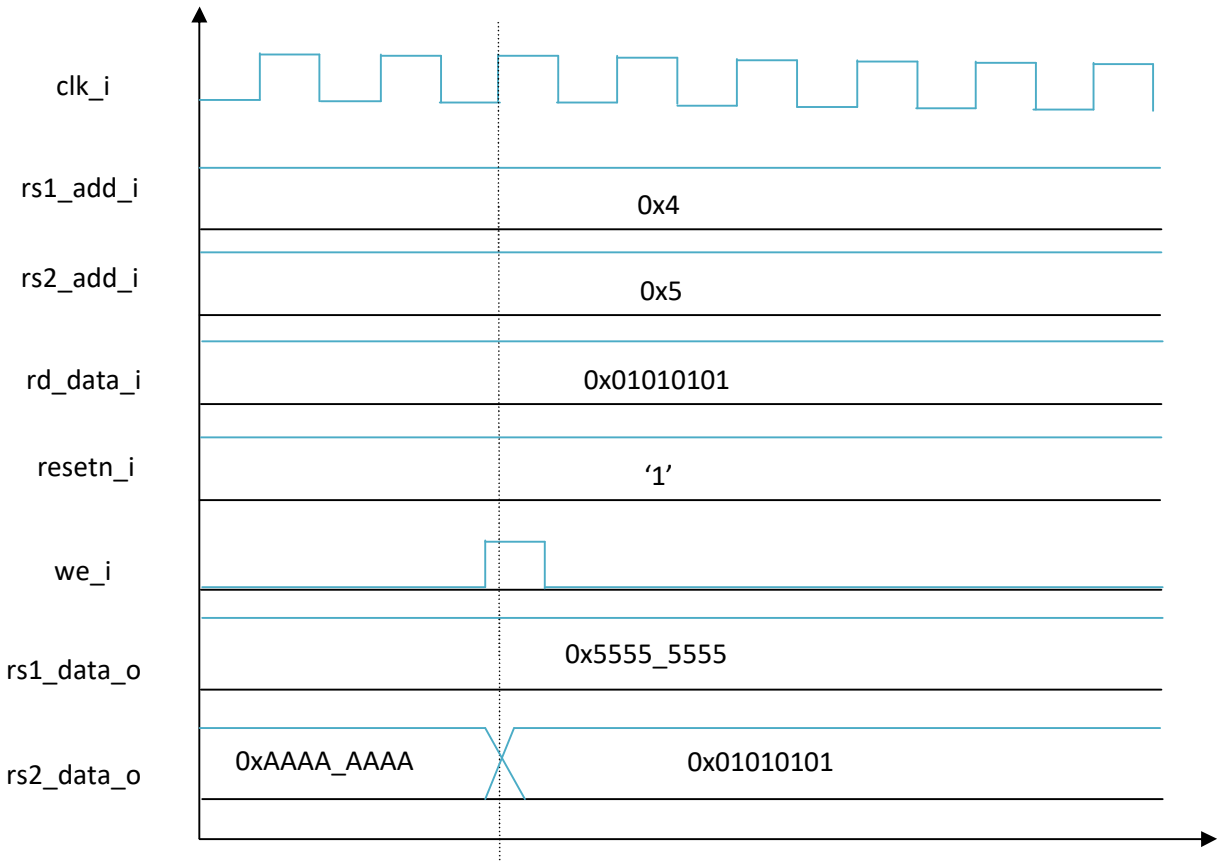


Figure 1 : entité register_file

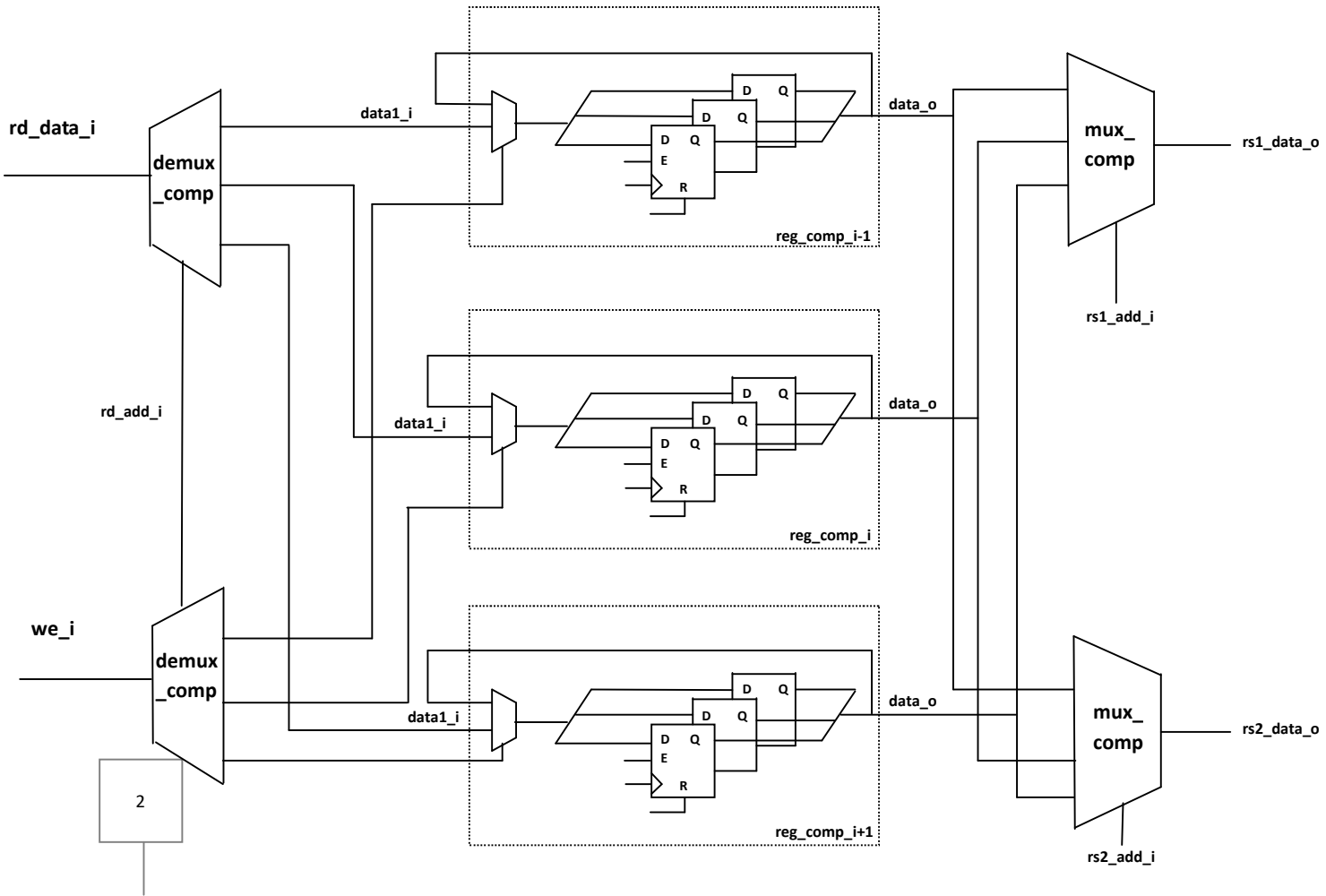
La définition de chacune des entrées et sorties de lôentité est la suivante :

- rs1_add_i et rs2_add_i sont deux entrées correspondant aux adresses respectives des deux registres sources
- rd_add_i correspond à lôadresse du registre de destination
- rd_data_i correspond à la donnée de 32 bits que lôon souhaite écrire sur le registre de destination
- clk_i correspond au signal dôhorloge de lôentité
- resetn_i est le signal de reset (lorsquôil est actif, tout est mis à zéro)
- we_i est le bit dôautorisation dôécriture sur le registre de destination (write enable). On ne peut écrire sur le registre que si ce bit est actif
- rs1_data_o et rs2_data_o correspondent aux contenus des registres (respectivement rs1 et rs2) obtenus en sortie du register_file

Pour lôexemple donné sur le sujet de TD, le chronogramme du register_file sera le suivant :

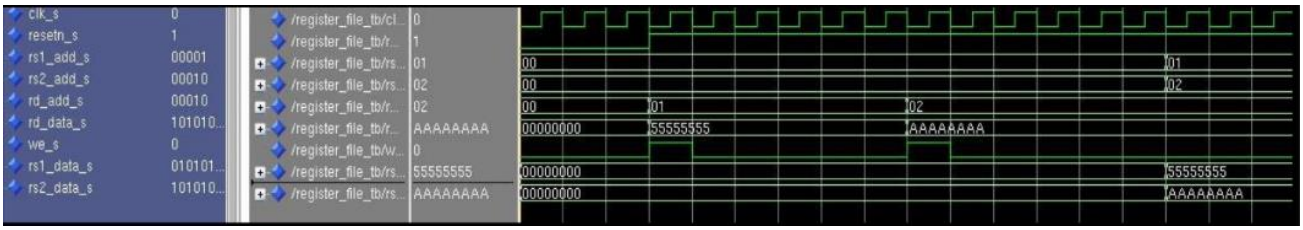


2) Le schéma du banc de registre donné sur 3 registres successifs (i-1, i et i+1) est le suivant :



3) Pour compléter le corps de lôarchitecture du composant register_file, on commence dôabord par inclure les composants utilisés dans le banc de registres (demux_comp, mux_comp et reg_comp), que lôon va port mapper correctement selon les connexions prévues sur le schéma réalisé à la question précédente. Pour cela, il va falloir créer deux composants DEMUX_1 et DEMUX_2 (pour lôun, lôentrée sera reliée à rd_data_i, pour lôautre ça sera we_i), un generate de taille 32 dans lequel on crée 32 reg_comp que lôon mappera aux sorties des deux demux, ainsi que deux multiplexeurs MUX_1 et MUX_2 dont les sorties respectives sont rs1_data_o et rs2_data_o.

4) Les résultats obtenus avec le testbench sont les suivants :



II) Banc de registres et ALU :

1) Les deux sorties des deux registres vont être connectées aux entrées de lôALU ; elles sont utilisées pour effectuer le calcul et déterminer lôentrée définissant la fonction à réaliser.
2) Sorties ALU :
3) Les quatre lignes du code, de la ligne 100 à 103, permettent dôavoir la sélection des bons bits dans lôinstruction. En effet, les bits 31 à 25 et 14 à 12 déterminent la nature de lôopération arithmétique, tandis que dôautres plages de bits comme les bits 24 à 20, déterminent les registres utilisés.
4) Pour vérifier la validité du code, le testbench fonctionne comme suit :

- On met dôabord la valeur de R2 à 1 et celle de R3 à 0
- Quelques ns plus tard, on met R3 à 1
- Quelques ns plus tard, on calcule la somme R2 + R3 et on la met dans R1

La traduction en langage assembleur RV32I ainsi que sa compilation du programme donné est la suivante :

Langage assembleur	Compilation
LOAD R1, 6	X''000000B3''
LOAD R2, 2	X''00000133''
LOAD R3, 3	X''000001B3''
LOAD R4, 1	X''00000233''
ADD R1, R1, R2	X''003080B3''
ADD R3, R3, R4	X''004181B3''
SUB R5, R1, R3	