Architecture des processeurs

Compte rendu

Auteur : Axel Gannerie
Institution : École des Mines de Saint-Étienne
Encadrant : M. Olivier Potin

Table des matières

1 TD 1 **2**
1.1 Organisation des registres 2
1.1.1 Banc de registres : 4 x 8 bits 2
1.1.2 Banc de registres : 32 bits 4

2 TD 2 **7**
2.1 RV32I architecture monocycle 7
2.1.1 Intégration du banc de registres 32 bits 7
2.1.2 Ajout d'instructions 10

3 TD 3 **13**
3.1 Instruction de branchement 13
3.1.1 Modification du `data_path` 13
3.1.2 Modification du `control_path` 13

4 TD 4 **15**
4.1 CRT & édition de lien 15
4.1.1 Construction du programme 15
4.1.2 Éxécution du programme 15

Chapitre 1

TD 1

1.1 Organisation des registres

1.1.1 Banc de registres : 4 x 8 bits

Q1 : Quelle est la largeur en nombre de bits des entrées et des sorties : *rd_data_i, rs1_data_o, rs2_data_o* ?

Réponse : *rd_data_i, rs1_data_o, rs2_data_o* correspondent à des entrées et sorties de données, elles font la même taille qu'une case mémoire, c'est-à-dire 8 bits.

Q2 : Combien de bits sont nécessaires pour pouvoir adresser 4 registres ? En déduire la largeur en nombre de bits des entrées *rd_data_i, rs1_data_o, rs2_data_o.*

Réponse : 2 bits sont nécessaires pour pouvoir adresser 4 registres. Les entrées *rd_add_i, rs1_add_i, rs2_add_i* font donc toutes 4 bits, car elles correspondent respectivement aux adresses des registres de destination, du registre de source numéro 1 et du registre source 2.

Q3 : Construction d'un registre sur 8 bits sur le logiciel logisim-evolution.

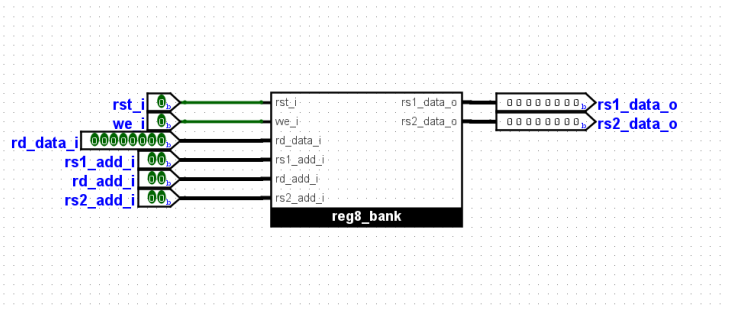

FIGURE 1.1 – Entrées et sorties du banc de registres 4 x 8 bits

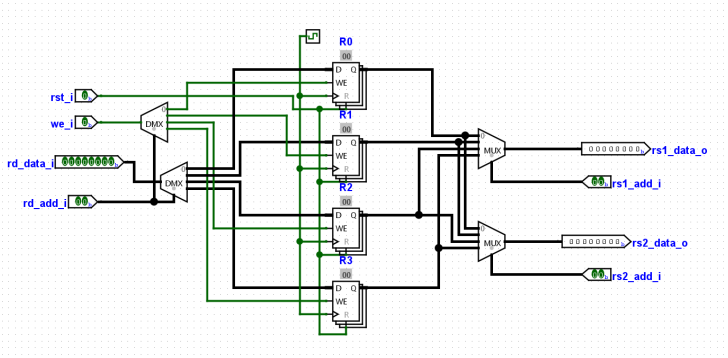

FIGURE 1.2 – Structure du banc de registre 4 x 8 bits

Q4 : Test d'écriture et de lecture du registre.

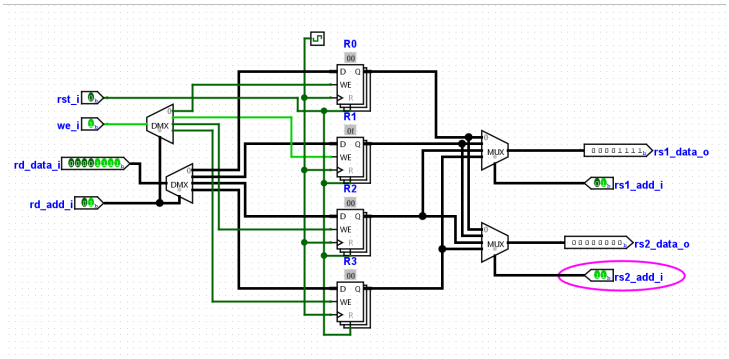

FIGURE 1.3 – Test du registre 4 x 8 bits

Q5 / Q6 : Dessinez le chronogramme correspondant au scénario suivant :

— adresse du registre source 1, 2 et destination à 0, valeur des données à inscrire dans le registre de destination à 0.
— remise à zéro.
— écriture de la valeur 0x11, 0x22, 0x33, 0x44 respectivement dans les registres 0, 1, 2 et 3.
— lecture des registres 1 et 2.
— remise à zéro.
— lecture des registres 1 et 2.

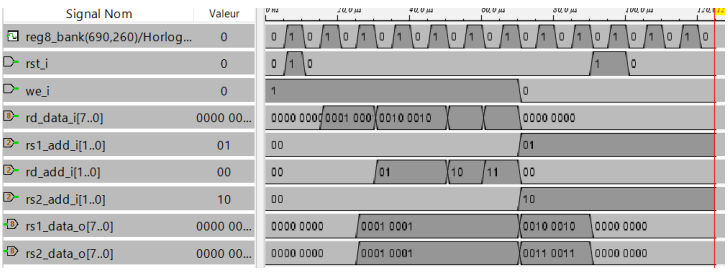

FIGURE 1.4 – Chronogramme correspondant à la séquence décrite précédemment

1.1.2 Banc de registres : 32 bits

Q7 : Complétez le circuit `registers_bank_incomplete.circ`.

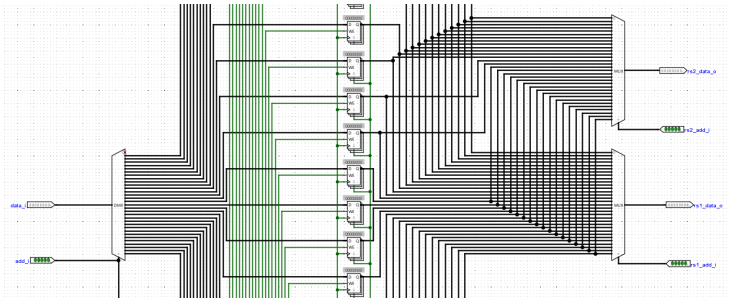

FIGURE 1.5 – Banc de registres 32 bits

Q8 : Complétez le circuit `registers_bank_incomplete.circ` pour qu'il soit conforme à la spécification RISC-V.

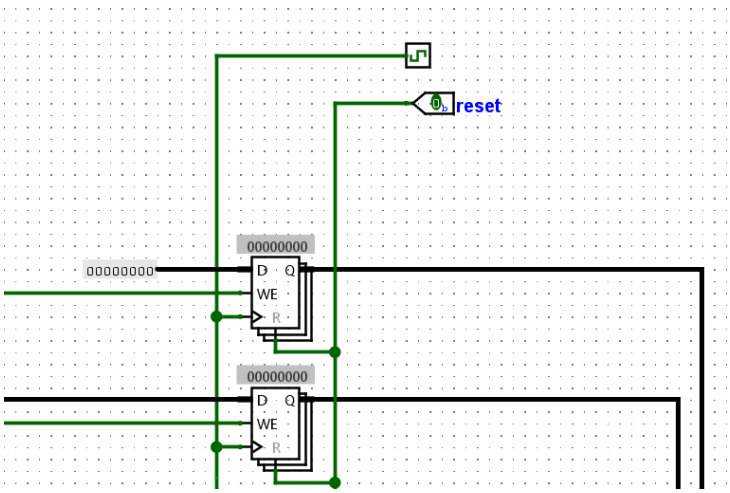

FIGURE 1.6 – Registre numéro 0 forcé à la valeur 0

Q9 : Reliez le banc de registres 32 bits à l'Alu dans le circuit `test`.

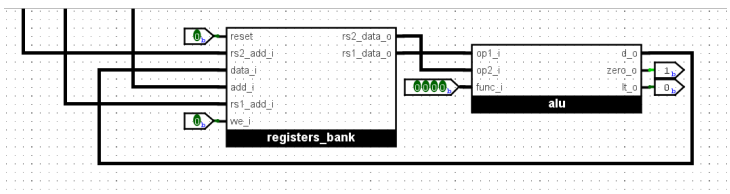

FIGURE 1.7 – Cablâge du banc de registres 32 bits à l'ALU

Q10 : Créez une entrée de 32bits de largeur que l'on nommera instruction Supposez que les seules instructions qui vont être présentées sur cette entrée soient des instructions de type registre et soient des additions. Quelle valeur doit être appliquée sur l'entrée `func_i` de l'ALU ? Utilisez le composant Répartiteur (splitter) pour extraire les champs de bits nécessaires pour piloter le reste des entrées de l'ALU.

Réponse : Après avoir regardé le détail du circuit `ALU`, l'entrée doit être réglée à 0001 pour effectuer une addition d'après la configuration du multiplexeur.

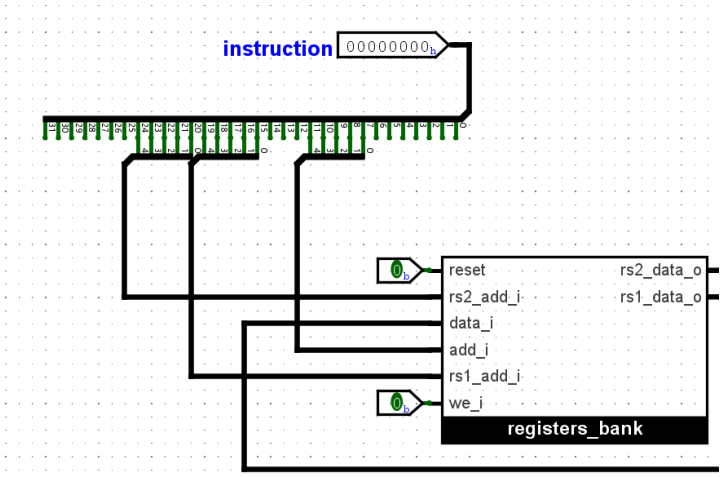

FIGURE 1.8 – Cablâge de l'entrée des instructions avec le registre 32 bits

Q11 : Compilez (traduire en langage assembleur RV32I) le programme suivant :

— 1 - Unsigned int a = 6
— 2 - Unsigned int b = 2
— 3 - Unsigned int c = 3
— 4 - Unsigned int d =1
— 5 - Unsigned int e
— 6 - e = (a+b)+(c+d)

Assemblez le programme assembleur obtenu.

Réponse :
— 1 - ADDI R1, R0, 6
— 2 - ADDI R2, R0, 2
— 3 - ADDI R3, R0, 3
— 4 - ADDI R4, R0, 1
— 5 - ADD R5, R1, R2
— 6 - ADD R6, R3, R4
— 7 - ADD R7, R5, R6

Programme assemblé

— 1 - 0x00600093
— 2 - 0x00200113

— 3 - 0x00300193
— 4 - 0x00100213
— 5 - 0x002082B3
— 6 - 0x00418333
— 7 - 0x006283B3

Chapitre 2

TD 2

2.1 RV32I architecture monocycle

2.1.1 Intégration du banc de registres 32 bits

Q1 : Dans quel sous-circuit faut-il instancier le banc de registres ?

Réponse : Le banc de registres doit être instancié dans le sous-circuit `data_path`.

Q2 : Quelle est le type d'instruction actuellement supporté ?

Réponse : Le circuit supporte actellement seulement les instructions de type R c'est-à-dire registre à registre d'après l'architecture du `data_path`. En effet, dans le sous-circuit `control_path` on trouve seulement de quoi écrire dans les registres. Une structure similaire à celle vu en cours jointe ci-dessous.

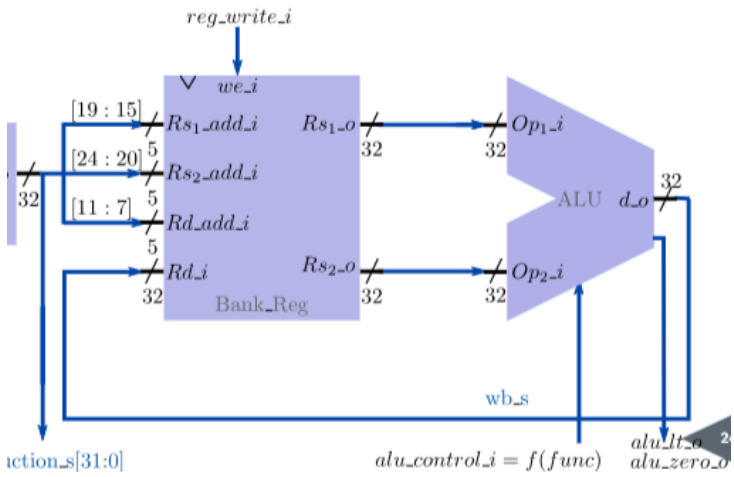

FIGURE 2.1 – Structure supportant exclusivement les instructions de type R

Q3 : Quel circuit contient les instruction du programme à exécuter sur cette micro-architecture ?

Réponse : Sur cette micro-architecture de type Harvard, les instructions du programme sont contenues exclusivement dans la mémoire c'est-à-dire dans le sous-circuit `memory`.

Q4 : Décomposition de l'instruction : 001080B3

Réponse : En décomposant l'instruction en octets puis en distinguant la sémentique de chaque groupe de bits d'une instruction de type R à l'aide de la greendcard RV32I, on obtient la segmentation suivante : 0000000|00001|00001|000|00001|011001. Cette segmentation resprésente respectivement funct7, rs2, rs1, funct3, rd, opcode. On reconnaît à l'aide de la greencard la fonction ADD qui additionne le contenu de deux registres (rs1 et rs2) et insére le résultat dans un autre registre (rd). Il faudra donc initialiser l'instruction dans la mémoire, manuellement initialiser le contenu des registres que l'on veut additionner et activer le write enable.

Q5 : Désassemblez l'instruction de la question précédente.

Réponse : En se référant à la greencard, on peut en déduire le sens de l'instruction qui est le suivant en assembleur : ADD R1, R1, R1.

Q6 : Simuler les deux instructions après avoir initialisé le système de la manière décrite précédemment.

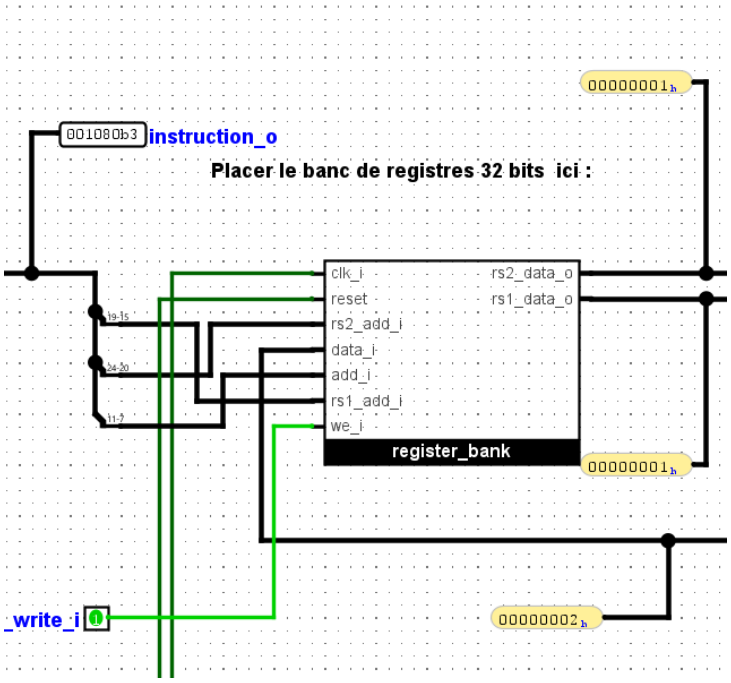

FIGURE 2.2 – Résultat après la 1ère instruction

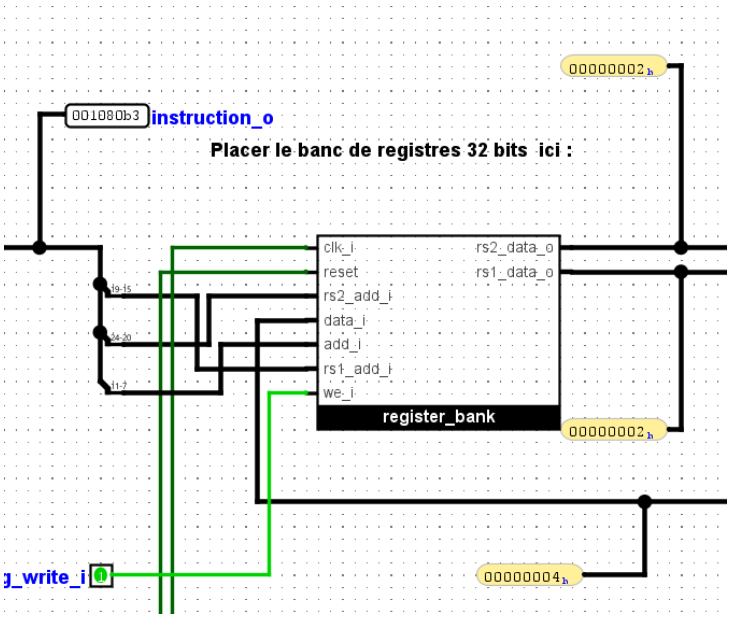

FIGURE 2.3 – Résultat après la seconde instruction

Réponse : Le résultat est donc 4. En effet, dans un premier temps le registre R1 est initialisé à 1 puis après la première instruction il passe à 2 et enfin après la deuxième instruction il passe à 4.

Q7 : Ajoutez à l'architecture actuelle le support de l'instruction OR.

Réponse : Pour cela on vient reprogrammer le PLA dans le `control_path` auquel on ajoute le code funct_3 de la fonction OR c'est-à-dire 110 d'après la greencard. Enfin, il faut faire correspondre la sortie du PLA avec la bonne adresse de redirection pour le multiplexeur en aval qui pilote l'ALU.

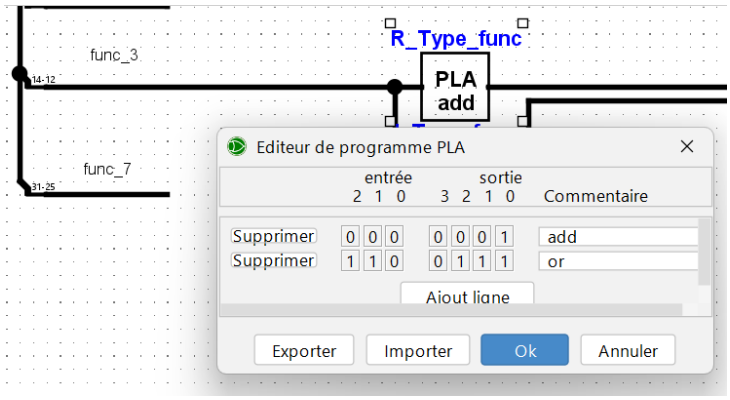

FIGURE 2.4 – Reprogrammation du PLA pour que l'architecture supporte l'opération OR

2.1.2 Ajout d'instructions

Q8 : Ajoutez à l'architecture actuelle le support des instructions
— LUI
— ADDI

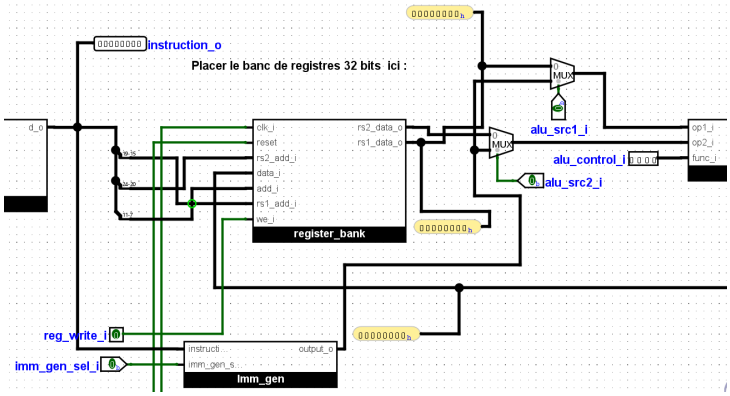

FIGURE 2.5 – Ajout du support des instructions ADDI (type I) et LUI (type U) dans l'architecture du control-path

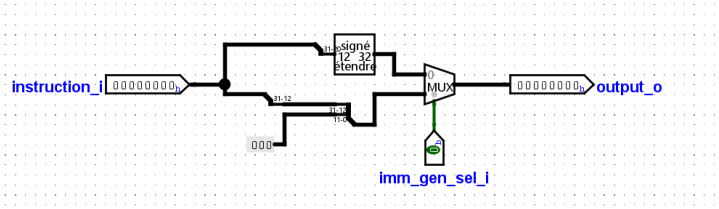

FIGURE 2.6 – Structure interne du générateur d'immediate

Q9 : Assemblez à la main, le programme suivant : **LI R1, 0xDEADBEEF**

Réponse : L'instrution **LI R1, 0xDEADBEEF** correspond aux 2 instructions suivantes pour notre architecture : **LUI R1, 0xDEADC** (il faut prendre en compte le fait que l'on va ajouter l'entier négatif 0xeef d'où le décalage à DEADC au lieu de DEADB) puis **ADDI R1, R1, 0xEEF**. Ces deux instructions se traduise en langage machine par respectivement $0xDEADC0B7$ et $EEF08093$

Q10 : Ajoutez à l'architecture actuelle le support des instructions

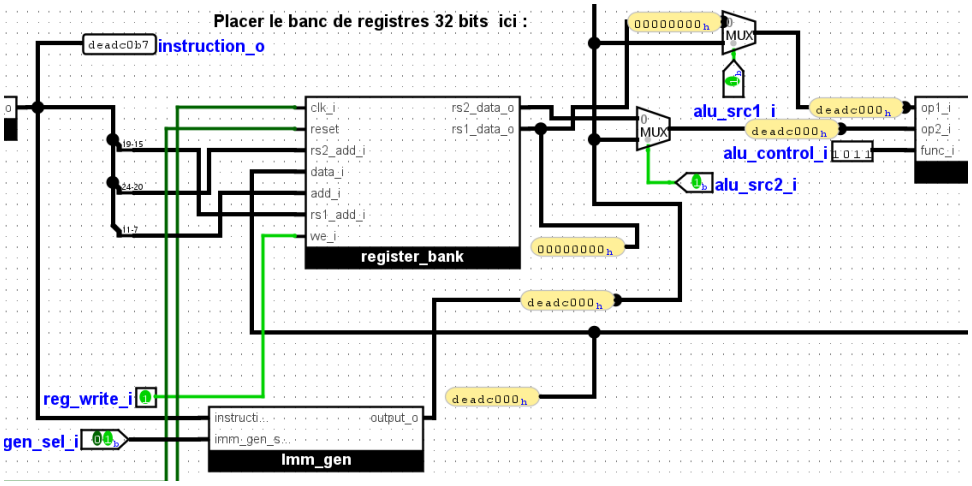

FIGURE 2.7 – État du control-path quand il reçoit l'instruction DEADC0B7

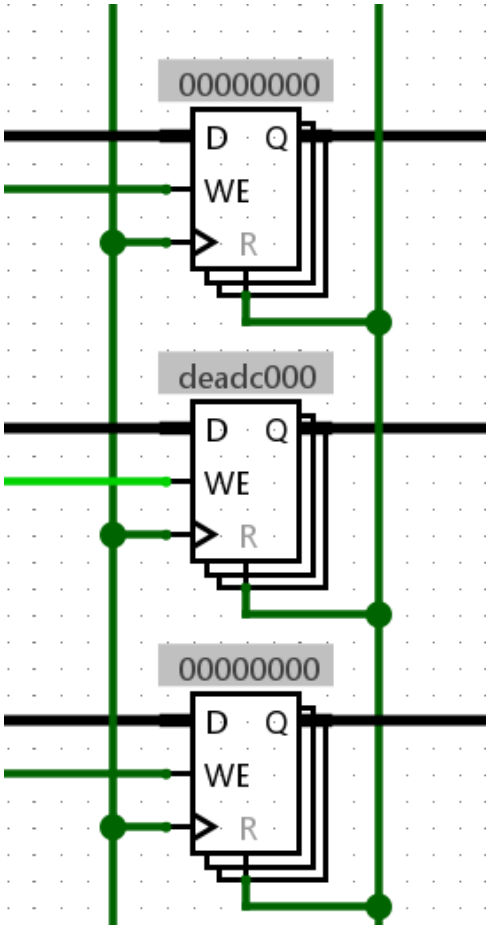

FIGURE 2.8 – Écriture dans le registre R1 après une période d'horloge

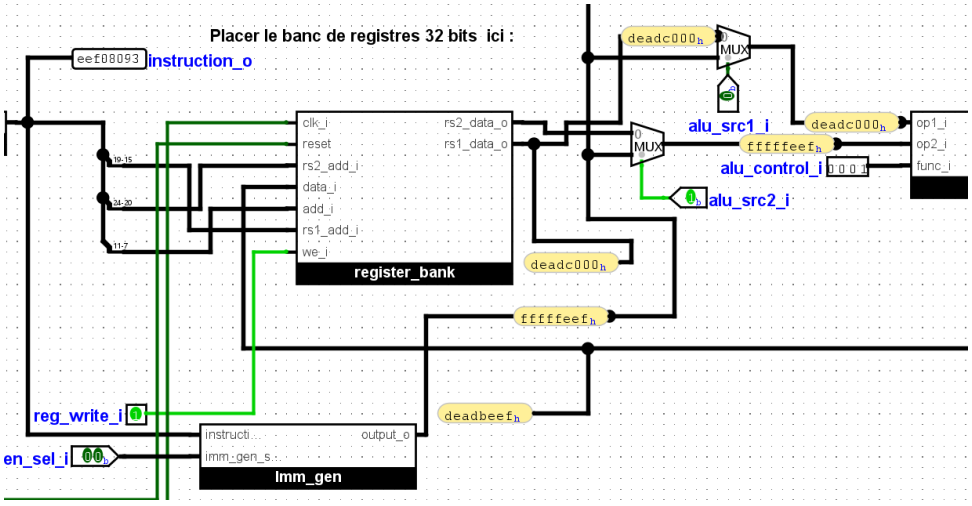

FIGURE 2.9 – État du control-path quand il reçoit l'instruction EEF08093

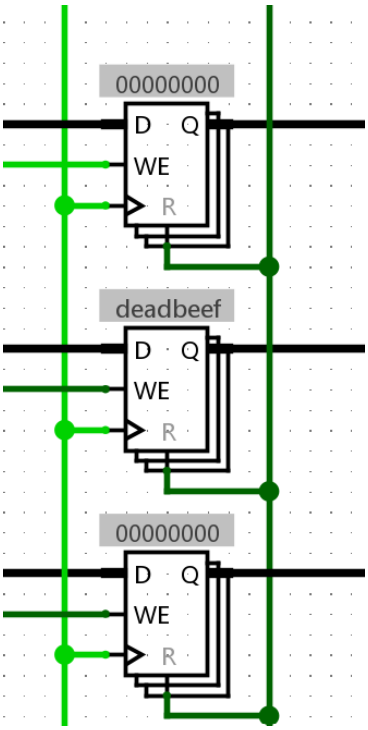

FIGURE 2.10 – Écriture dans le registre R1 après une deuxième période d'horloge

Chapitre 3

TD 3

3.1 Instruction de branchement

3.1.1 Modification du `data_path`

Q1 : Comment doit-on traiter la valeur immédiate pour une instruction de branchement

Réponse : À l'aide d'un splitter on récupère les 8 bits de point faible de la valeur immédiate puisque la mémoire à notre disposition peut-être adressée avec 256 adresses différentes.

Q2 : Quel signal en sortie de l'ALU allons nous utiliser pour déterminer si la condition de branchement est remplie ?

Réponse : Le signal `zero_o` en sortie de l'ALU nous permet de savoir si le résultat est nul. Il suffit donc de réaliser une différence entre les deux signaux en entrée de l'ALU et de vérifier la valeur de `zero_o` pour savoir s'ils sont différents ou égaux, si `zero_o` est différent de 0 alors les signaux en entrée sont différents et inversement.

Q3 : Quel opération de l'ALU va permettre d'activer ce signal suivant la condition requise par cette instruction de branchement ?

Réponse : C'est la soustraction qui va permettre de d'activer le signal `zero_o` comme décrit dans la question précédente.

3.1.2 Modification du `control_path`

Q4 : Écrire le programme assembleur correspondant, de combien d'instructions a-t-on besoin ?

Réponse : Le programme nécessite 3 instructions qui sont les suivantes dans l'ordre de la mémoire :
- ADDI R2, R0, 10
- ADDI R1, R1, 1
- BNE R1, R2, -4

Q5 : Assemblez ce programme.

Réponse : Les 3 instructions assemblées donnent dans l'ordre :
— 00A00113
— 00108093
— FE209E63

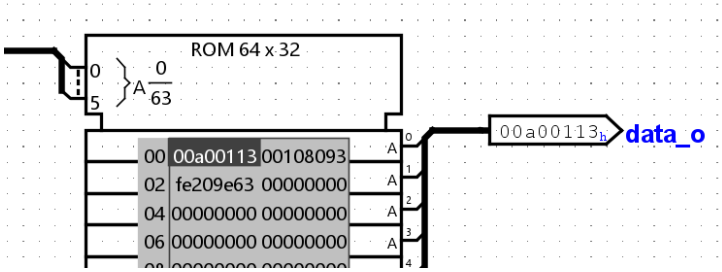

FIGURE 3.1 – Instructions du programme écritent dans la ROM.

Q6 : Éxécutez et vérifiez votre programme. Capturez un chronogramme lors de l'éxécution de votre programme.

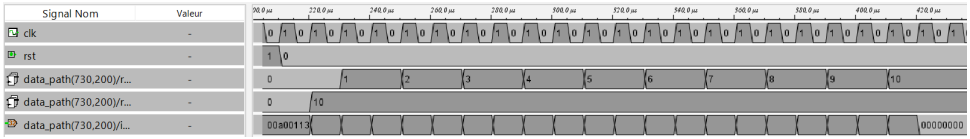

FIGURE 3.2 – Chronogramme suivant l'éxécution du programme.

Q bonus : Ajout de la lecture des instructions de type J.

Chapitre 4

TD 4

4.1 CRT & édition de lien

4.1.1 Construction du programme

4.1.2 Éxécution du programme