1.1 Analyse du processeur

Question 1

Dans le circuit mult_2, les offsets sont incrémentés de 1 au lieu d'être incrémenté de 2 dans le circuit non-pipeliné, pourquoi?

La formules pour calculer une nouvelle adresse après un saut est : PC + ext_16b(offset x 2) + 4. Pour le circuit pipeliné les choses sont faites étapes par étapes et plusieurs instruction peuvent être faites simultanément. Le FETCH va incrémenter de +2 pour avoir la bonne adresse du ROM pour pouvoir lancer l'instruction suivante au prochain coup de clocks. Il nous reste donc maintenant à faire le +2 restants au lieu de +4 et donc +1 * 2 font +2.

Question 2

Dans le circuit fetch_decode_reg, le signal LR_adr_o vient d'un registre et est connecté au bloc decode au lieu du bloc bank_register, pourquoi?

Car depuis que nous somme pipeliné il, faut que la valeur de LR suive le chemin du pipeliné pour pouvoir être envoyé au bank_register quand il a été détermine s'il fallait envoyer l'adr dans le registre LR.

Question 3

Dans le circuit decode, le signal adr_reg_d_s va dans le circuit decode_execute_reg dans un registre alors que les signaux adr_reg_n_s, adr_reg_m_s et adr_reg_mem_s ne sont pas stockés dans des registres, pourquoi ?

Les registres opérandes et mémoires sont directement envoyés à EXECUTE et sont enregistrés dans ce bloc-là. Cependant, dans le bloc DECODE, on doit pouvoir décoder l'instruction suivante, c'est pourquoi il faut sauvegarder l'instruction précédente pour ensuite envoyer les données venant du bloc exécute.

Question 4

Dans le circuit decode_execute_reg, les signaux des bus de contrôle sont connectés aux registres avec une porte MUX contrairement aux autres signaux, pourquoi?

Car il ne faut pas stocker un mauvais bus et l'envoyer sans réfléchir, car les informations de ces bus pourrait changer la suite des exécutions, et donc le mux permet de reset à 0 le registre. Si dans le coup de clock on ne se situe pas dans le DECODE alors on envoit pas le bus

Question 5

Si on voulait ajouter le multiplieur 5x3 pipeliné du laboratoire préparatoire, quelles seraient les conséquences sur le pipeline du processeur? Comment ça pourrait être fait?

Une multiplication nécessite 3 coups de clocks à elle seule. Pour commencer, on cherche les dépendances afin d'éliminer les potentiels aléas. Dans le meilleure, des cas, il y a trois instructions entre le moment où on effectue la multiplication et le moment où on lit le résultat de la multiplication. Dans le pire des cas, on le fait à l'instruction d'après, il faut donc placer 5 nop pour ne pas avoir d'aléas. 3 nop pour effectuer la multiplication et 2 pour écrire le résultat en mémoire.

1.2 Test du processeur

Question 1

Est-ce que le programme s'exécute correctement? Est-ce que les registres et la mémoire prennent les bonnes valeurs?

Le programme s'exécute correctement en exécutant les instructions dans l'ordre et de façon pipeliné.

Les registres prennent les bonnes valeurs.

Question 2

Combien de cycles sont nécessaires pour exécuter ce programme?

15 cycles sont nécessaire pour l'exécution de ce code.

1.3 Assembleur: dépendances de données

Question 1

Quelles dépendances posent des problèmes d'aléas?

Instruction STRH r0, [r1, #4] RAW ligne 2 r1
Instruction SUB r4, r1, r0 RAW ligne 5 r0
Instruction LSL r2, r2, #1 RAW ligne 10 r2

Question 2

Combien de cycles sont nécessaires pour résoudre un aléa de donnée?

Il faut 4 - nombre de lignes (où se trouve le registre avec la dépendance), pour savoir le nombre de cycles nécessaires pour résoudre un aléa de données.

Question 3

Quel est l'IPC sur le programme fourni ? Le throughput si la clock vaut 4KHz? La latence?

IPC = 1 / (0.8 *1* + *0.2* (2 + 1)) = 0.6667

Débit = 1/fréquence --> 1/4000 = 0.00025

Latence = 5 * 1/fréquence --> 5/4000 = 1/800 = 0.00125

2.1 Circuit data_hazard

Question 1

Comment savoir si une instruction est dépendante d'une instruction qui est pour le moment dans le stage EXECUTE? dans le stage MEMORY_ACCESS? Dans le stage WRITE_BACK?

En comparant les dépendances des différents registres et en calculant combien de temps après l'instruction suivante est lancé après. Par exemple si l'instruction dépend de la précédente, elle sera dépendante de cette instruction qui sera dans le stage Execute.

Question 2

Est-ce que ça pose un problème si une instruction dépend du résultat d'une instruction qui est au stage WRITE_BACK?

Oui, c'est le dernier stage qui peut créer un aléa.

Question 3

Quelles informations doivent être mémorisées pour chaque instruction?

Le registre destination et dans quel stage l'instruction se trouve

Question 4

Quelles informations permettent de savoir si le registre D est utilisé?

La condition de saut ne sera peut-être pas respectée si l'aléa n'est pas traîté correctement.
Si les aléas sont traités correctement cela rajoutera entre 1 et 3 cycles supplémentaires.

2.2 Circuit hazard_detection

Question 1

Quelles informations permettent de savoir si le registre N, M ou mem sont utilisés?

Les signaux reg*n_en_i, reg_m_en_i et reg_mem_en*

Question 1

Quelles informations permettent de savoir si le registre D est utilisé?

Les signaux reg_bank_write_en_i et adr_reg_d_i

Question 2

Une détection d'aléa de donnée va influencer quel(s) enable(s)? A quel moment ? Pourquoi?

Il va influencer tout les enable lorsqu'il y a une détection. Il est nécessaire de tous les arrêter car il ne fait pas quie le Fetch avance sans que le Decode fasse sans travail. Les stage qui suivent le decode sont arrêter un à un après chque coup de clock pour permettre aux instructions précédentes de terminer leur travail et ainsi repartir avec les registres qui sont à jours pour ne plus avoir de problème d'aléa.