TD2
Gestion des dépendances

Michel Agoyan

Novembre 2025

1 Introduction

Il vous est demandé de rédiger un compte rendu pour ce TD dans lequel vous ferez figurer toutes les remarques pertinentes qui vous semblent nécessaires à la bonne compréhension du problème posé et de la solution apportée. Nous vous rappelons qu'il important pour une bonne compréhension et une bonne assimilation de ce cours de réaliser un travail personnel.

Dans ce deuxième TD, nous allons expérimenter les effets de la dépendance sur une microarchitecture de type **pipeline**.
Nous essaierons plusieurs méthodes pour tenter de résoudre ces dépendances :
- insérer des instructions NOP en logiciel
- implémenter un interlock

2 Problème de dépendances

Le programme assembleur **exo2.S** dans le répertoire **firmware** comporte des dépendances .

```
.section .start;
.globl start;



start:
      addi x5,x0,15
      add  x6,x0,x0
      add  x7,x0,x0
      add  x8,x0,x0
lab1:
      j lab2
      addi x7,x7,1

lab2:
      addi x6,x6,1
      bge x5,x6,lab1
      addi x8,x8,1
      addi x8,x8,1
lab3 :  j lab3
      nop
      nop

.end start
```

3 Correction du problème

3.1 Correction logicielle

Q1

Comment peut-on coder une instruction **NOP** sur l'architecture RISC-V ?

Q2

Utilisez cette instruction pour éliminer les dépendances. De quel type de dépendance s'agit-il ? Combien d'instructions NOP devez-vous utiliser ?

Q3

Vérifiez, en utilisant la simulation, que le programme fonctionne comme attendu.

4 Correction matérielle : Ilock

4.1 Dépendances de données

Q4

En examinant le sous-circuit **control_path**, que représentent les signaux de l'étage ID :
— **rs1_dec_w**
— **rs2_dec_w**

Q5

Que représente le signal **rd_add_XXX_w** des étages EXE, MEM et WB ?

Q6

Ecrire l'équation booléenne générant un signal **stall_o** à partir des signaux précédemment identifiés.
Dans un premier temps, nous allons écrire une version simple de cette équation en ne tenant compte que des registres sources de l'instruction à l'étage DEC et du registre destination dans les étages qui suivent.
On calculera séparement les 2 conditions de dépendance sur **rs1** et **rs2** avant de les combiner pour obtenir **stall_o**.
On notera **hazard_rs1** et **hazard_rs2** les signaux intermédiaires correspondants.
Que se passe-t-il quand le signal **stall_o** est actif dans le **control_path** ?

Q7

Est-ce que toutes les instructions utilisent (lisent) **rs1** ?
Construire un signal **rs1_re_dec_w**, actif si l'instruction à l'étage décodage utilise **rs1**. Utilisez ce signal pour qualifier en conséquence la partie concernée de l'équation booléenne de la question **Q6**.

Q8

Est-ce que toutes les instructions utilisent (lisent) **rs2** ?
Construire un signal **rs2_re_dec_w**, actif si l'instruction à l'étage décodage utilise **rs2**. Utilisez ce signal pour qualifier en conséquence la partie concernée de l'équation booléenne de la question **Q6**.

Q9

Est-ce que toutes les instructions utilisent (évrivent) **rd** ?
Qualifiez en conséquence la partie concernée de l'équation booléenne de la question **Q6**.
Pour factoriser le travail, écrire une fonction SystemVerilog **rd_we** qui renvoie un booleen pour dire si l'instruction écrit dans rd.
Cette fonction est ensuite utilisée pour affecter les signaux **rd_we_exec_w**, **rd_we_mem_w**, **rd_we_wb_w** qui seront utilisés dans l'équation de **Q6**.

Q10

Testez votre solution en utilisant le programme d'origine sans les instructions NOP insérées pour résoudre les dépendances de données mais en maintenant celles qui ont été insérées pour résoudre les dépendances de contrôle et comparez la séquence d'intructions avec celle issue du programme de la question **Q3**.
Pour cela observez dans le **contro_path** :

- **inst_wb_r**
- **stall_o**

4.2 Dépendances de contrôle : sauts

Q11

Gestion des sauts : Ajoutez la logique nécessaire dans le control_path pour la gestion des sauts (voir cours) ?
Faut-il modifier l'équation de la question **Q6** ?

Q12

Testez votre solution en utilisant le programme d'origine sans les instructions NOP insérées pour résoudre les dépendances de contrôle relatives aux sauts et comparez la séquence d'intructions avec celle issue du programme de la question **Q3**.
Pour cela observez dans le **contro_path** :

- **inst_wb_r**
- **stall_o**

4.3 Dépendances de contrôle : branchements

Q13

Gestion des branchements : Ajoutez la logique nécessaire dans le control_path et data_path pour la gestion des banchements (voir cours) ?
Faut-il modifier l'équation de la question **Q6** pour **stall_o** ?

Q14

Testez votre solution en utilisant le programme d'origine sans les instructions NOP insérées pour résoudre les dépendances de contrôle relatives aux branchements et comparez la séquence d'intructions avec celle issue du programme de la question **Q3**.
Pour cela observez dans le **contro_path** :

- **inst_wb_r**
- **stall_o** lz

Q15

Avant de poursuivre , remplacer le programme de test utilisé au début du TD par le programme assembleur fourni avec ce TD : **ilock_test.S.bck**.
Renommez le fichier assembleur utilisé depuis le début du cours en **exo2.S.bck** puis renommez **ilock_test.S.bck** en **exo2.S** (le fichier build.sh compile par défaut ce fichier) Ce test est constitué de plusieurs tests unitaires vérifiant chachun une partie de l'interlock. Il n'est pas complet et vous pouvez le compléter. Lisez les commentaires qui indiquent les résultats attendus et à vérifie de visu sur les chronogrammes de simulation.

5 Dépendances de contrôle : WS

Cette partie du TD , va nous permetrre de maintenir le mécanisme d'interlock en présence de temps d'attente (Wait State) lors d'accès mémoire ce qui nous permettra par la suite d'ajouter des contrôleurs de mémoires caches.

Changez les temps d'attente d'accès mémoire de la mémoire instructions et de la mémoire données en changeant le paramètre WS et en le fixant à 3.

5.1 Gestion des branchements

Q16

Comme vu dans le cours, modifiez le signal **imem_re_o** présent dans le fichier **RV32i_pipeline_top.sv** pour l'effacer en cas de branchement.
On pourra se servir de **pc_next_sel_w** pour détecter un branchement.

Q17

Comme également vu dans le cours il faut maintenir le signal **inst_dec_w**, présent dans le controlpath , effacé (= NOP) en cas de branchement.
Pour cela on combinera au signal **stall_dec_i**, présent dans l'équation de **inst_dec_w**, un signal **stall_branch_w** qui sera actif lors d'un branchement et sera maintenu jusqu'a que la nouvelle transaction déclenchée par l'effacement de **imem_re_o** soit terminée.
Indication : quelle est la relation entre l'entrée **stall_dec_i** du controlpath et l'entrée **imem_valid_i** au top du design ?

Q17

Testez la solution obtenue , et résoudre la gestion des sauts le cas échéant .