**UE104 - Architecture des systèmes**

**HAUTE ÉCOLE DE NAMUR-LIÈGE-LUXEMBOURG**

**Bloc 1**

Atelier 5 : Pipeline

Objectifs

* Comprendre la différence entre l’exécution séquentielle et en pipeline des instructions du CPU
* Expliquer les dépendances de données et les aléas de données
* Décrire une technique du pipeline pour éliminer les aléas de données
* Montrer les bénéfices de l’optimisation du compilateur “loop unrolling” sur le pipeline d’instructions
* Décrire comment le réarangement des instructions par le compilateur minimise les dépendances de données

Montrer l’utilisation de la table de prédiction de branchements dans l’optimisation du pipeline

# Introduction

Cet atelier regroupe une série d’activités à réaliser avec CPU OS Simulator afin de découvrir divers aspects du pipeline d’instructions d’un CPU. Diverses instructions sont fournies afin d’identifier et de localiser les informations permettant de compléter ce document. N'oubliez pas de prendre des notes de vos observations.

Afin d’être capable de faire ces activités, vous devriez consulter les informations fournies dans les notes du cours de théorie associé. Le but est de vous permettre de saisir le fonctionnement d'un CPU, lisez donc attentivement et veillez à comprendre les instructions au lieu de faire des essais-erreurs qui ne vous amèneront pas nécessairement à la solution/réponse idéale.

# Différence entre l’exécution en séquentiel ou en pipeline

Écrivez le code source suivant dans l’éditeur de code source de la fenêtre **Program Compiler** et compilez-le.

program pipeline1

for n = 1 to 20

p = p + 1

next

end

Chargez le code assembleur qui en résulte dans la mémoire du simulateur via le bouton **LOAD IN MEMORY…** mis en évidence sur la Figure 1.

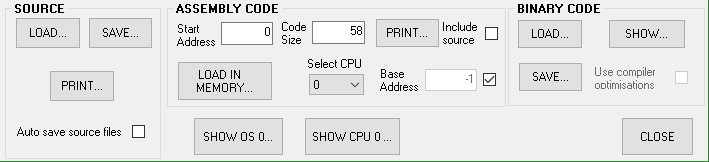


Figure 1 - Passage du compilateur vers le CPU

De retour dans la fenêtre principale du simulateur, cliquez sur le bouton **SHOW PIPELINE**… de l’onglet **Cache-Pipeline** de la Figure 2.

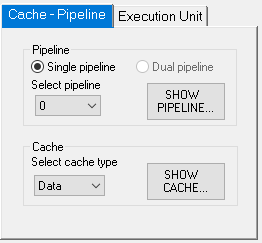


Figure 2 - Ouvrir le pipeline

La fenêtre **Instruction Pipeline** présentée à la Figure 3 s’ouvre et vous permet de visualiser les étapes du pipeline.

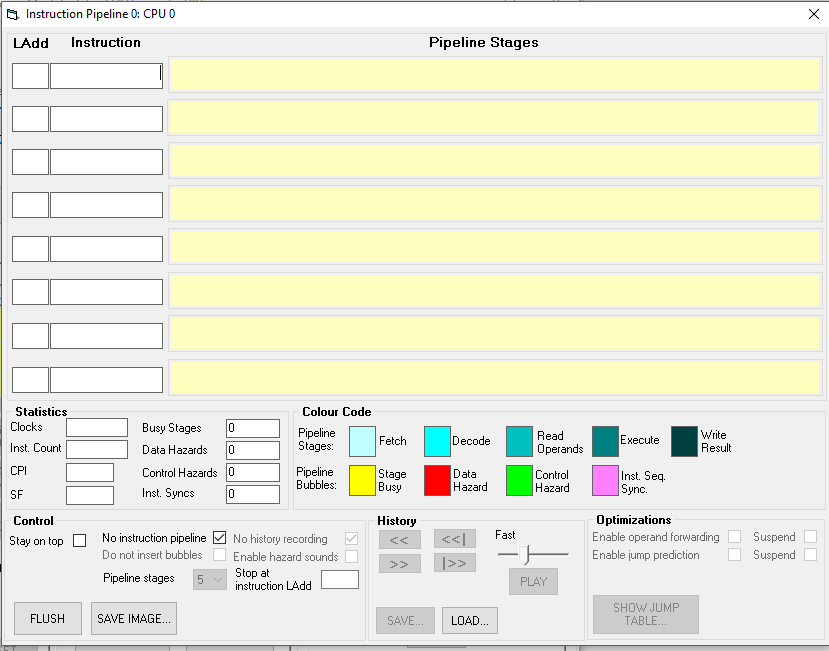


Figure 3 - Instruction Pipeline

Cette fenêtre simule le comportement du pipeline. Vous pouvez y observer les différentes étapes du pipeline au fur et à mesure du traitement des instructions du programme. Ce pipeline comporte cinq étapes. Chacune a un code couleur comme le montre Figure 4. Les différents événements mettant à mal l’efficacité du pipeline ont également un code couleur.



Figure 4 - Code couleur des étapes du pipeline

Listez les noms des 5 étapes du pipeline en les décrivant brièvement selon ce qui a été vu en théorie (elles ne portent pas exactement le même nom que dans le cours de théorie) :

Lors de l’exécution du programme, les instructions en cours de traitement dans le pipeline sont répertoriées sur le côté gauche de la Figure 5 (sous le label **Instruction**). L’instruction la plus récente du pipeline est en bas et la plus ancien en haut.

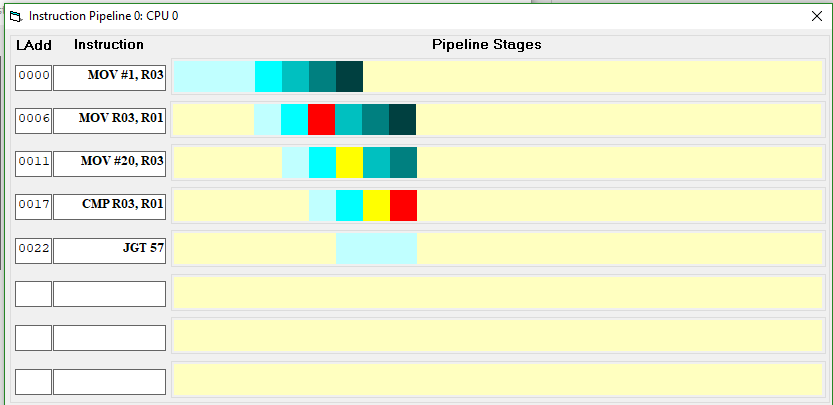


Figure 5 - Exemple d'instructions dans le pipeline

Chaque instruction passe par les différentes étapes du pipeline. Celles-ci sont mises en évidence au moyen du code couleur dans la zone de droite de la Figure 5 (sous le label **Pipeline Stages**). Lorsqu’un aléa ou autre souci se présente, il se distingue au moyen de la couleur qui lui est associée.

Dans la zone **Statistics**, visible sur la Figure 6, sont affichées des informations concernant les instructions exécutées.

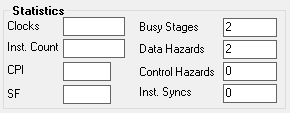


Figure 6 - Statitiques sur les instructions

Dans la zone **Controls**, mise en évidence sur la Figure 3, cochez la case **Stay on top** et cochez également la case **No instruction pipeline**. Dans la fenêtre principale du simulateur, faites-en sorte que le *slider* indiquant la vitesse d’exécution des instructions se trouve aux alentours de 30. Exécutez le programme et observez le pipeline.

Si vous désirez conserver l’historique de toutes les instructions qui défilent dans le pipeline, vous pouvez décocher l’option **No History recording** de la zone **Control** de la Figure 3.

Attendez que le programme soit terminé et notez ensuite les valeurs correspondant aux valeurs suivantes de la zone **Statistics** :

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |

Ensuite, décochez la case **No instruction pipeline**, faites un **RESET PROGRAM** et relancez-le. Lorsque son exécution est terminée, notez vos observations sur les différences que vous avez constatées par rapport à l’exécution précédente.

Prenez à nouveau note des valeurs suivantes :

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |

Expliquez brièvement à quoi sont dues ces différences entre les valeurs sans et avec pipeline :

# Aléas de données, bulles et instruction NOP

Dans les CPU, le pipeline doit souvent faire face à divers aléas, comme on l’a vu au cours de théorie. Un aléa de données est provoqué par le fait que la valeur d’un opérande soit indisponible (parce qu’en cours de modification ou pas encore chargée par l’instruction précédente), alors qu’elle est nécessaire. Pour observer cela, dans la fenêtre principale du simulateur, créez un programme nommé pipeline2 et entrez-y les instructions suivantes :

MOV #1, R01

MOV #5, R03

MOV #3, R01

ADD R01, R03

HLT

Avant de continuer, notez ce que vous pensez voir apparaître dans le registre général R03 à la fin des instructions ci-dessus.

R03 =

Assurez-vous que la case **No instruction pipeline** est décochée et que la case **Do not insert bubbles** est cochée. Faites un **RESET PROGRAM** et relancez-le. Prenez note de ce qui apparaître dans le registre général R03 à la fin des instructions ci-dessus.

R03 =

Maintenant, insérez une instruction NOP (dans l’onglet **Miscellaneous**) après la troisième instruction, afin d’obtenir le code suivant :

MOV #1, R01

MOV #5, R03

MOV #3, R01

NOP

ADD R01, R03

HLT

« *Resetez* » le programme et relancez-le. Prenez note de ce qui apparaître dans le registre général R03 à la fin des instructions ci-dessus.

R03 =

Vous avez pris note de 3 valeurs pour le registre R03. Commentez brièvement ces 3 valeurs :

Effacez l’instruction NOP du programme ci-dessus et décochez la case **Do not insert bubbles**. Faites un **RESET PROGRAM** et relancez-le. Prenez note de ce qui apparaître dans le registre général R03 à la fin des instructions.

R03 =

La valeur ci-dessus devrait être la même que celle de la version dans laquelle vous avez inséré l’instruction NOP. Toutefois cette valeur est obtenue sans l’instruction NOP dans ce cas. Expliquez brièvement :

Avez-vous vu la « bulle » ? De quelle couleur est-elle ?

Enfin, notez les différentes valeurs ci-dessous.

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |
| Data Hazards |  |

# Une façon d’éliminer les aléas de données

Une façon de gérer les aléas de données est de permettre au CPU de prendre connaissance des opérandes déjà chargées en ajoutant des « courts-circuits » au niveau matériel, ce qu’on appelle « *forwarding* ».

Pour mieux comprendre cette méthode, cochez l’option **Enable operand forwarding** dans la zone **Optimizations** de la Figure 3 et, après avoir fait un **RESET PROGRAM**, exécutez à nouveau le code ci-dessus.

La bulle de l’exercice précèdent a-t-elle disparu ?

Le simulateur compte le nombre d’aléas qu’il détecte lors du passage des instructions dans le pipeline. Ces informations sont affichées dans la zone **Statistics**.

Notez les valeurs suivantes :

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |
| Data Hazards |  |

Y a-t-il eu une amélioration par rapport à la version sans « *forwarding* » ?

# Déroulement de boucle pour minimiser les aléas de contrôle

Dans l’atelier 6 sur les optimisations du compilateur, vous avez examiné une méthode d’optimisation appelée « déroulage de boucle ». Cette méthode consiste essentiellement à dupliquer le code intérieur d’une boucle autant de fois qu’il y a d’itérations, permettant de supprimer certaines instructions redondantes, mais surtout de supprimer les instructions de comparaison et de saut. Cette méthode comporte certains défauts dont l’augmentation de la taille du code.

Toutefois, il est démontré que cette technique est intéressante dans l’architecture avec pipeline. En effet, le fait de dérouler les boucles permet d’exploiter les avantages du traitement en parallèle de plusieurs instructions en évitant les aléas de contrôle dus aux branchements… et ainsi d’améliorer les performances du processeur.

Cet exercice devrait vous en convaincre.

Créez un programme nommé pipeline3\_1 et entrez le code suivant. Compilez celui-ci en n’oubliant pas de cocher l'option **Remove redundant code** avant de compiler.

program pipeline3\_1

for n = 1 to 8

t = t + 1

next

end

Notez la taille du code généré pour le programme pipeline3\_1 ci-dessous :

Maintenant, chargez ce code dans la mémoire du simulateur via le bouton **LOAD IN MEMORY…** mis en évidence sur la Figure 1.

Ensuite, cochez l’option d’optimisation **Apply loop unrolling on loops** en plus de **Remove redundant code**. Changez le nom du programme en pipeline3\_2 et compilez à nouveau ce programme. Chargez cette seconde version dans la mémoire du simulateur.

Vous devriez avoir deux versions, l’une sans et l’autre avec l’option « déroulage de boucle ».

Notez la taille du code généré pour le programme pipeline3\_2 ci-dessous :

Cochez la case **Stay on top** dans la zone **Control** de la fenêtre **Instruction Pipeline** afin que celle-ci reste au-dessus.

Décochez l’option **Enable operand forwarding** et l’option **Enable jump prediction** dans la zone **Optimizations** de la Figure 3 et exécutez à nouveau le code ci-dessus.

Sélectionnez le programme pipeline3\_1 dans la zone **PROGRAM LIST** de la fenêtre principale du simulateur et faites un **RESET PROGRAM**. Faites-en sorte que le *slider* indiquant la vitesse d’exécution des instructions se trouve au maximum.

Cliquez sur le bouton **RUN** pour exécuter ce programme. Regardez le pipeline et lorsque le programme est terminé, prenez note des valeurs suivantes :

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |
| Inst. Count |  |

Procédez de la même façon pour le programme pipeline3\_2 :

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |
| Inst. Count |  |

Commentez brièvement ce que vous observez en considérant la taille du code et le nombre d’instructions exécutées :

# Réarrangement de la séquence d’instruction par le compilateur pour minimiser les aléas de données

Le « déroulage de boucle » est une des techniques permettant de mieux tirer parti du pipeline. Un autre exemple est le réarrangement des instructions, tout en gardant la logique du programme. Cette solution permet de minimiser les aléas de données…

Dans la fenêtre **Program Compiler**, dans la zone **COMPILER** et l’onglet **Options**, cochez l’option **Show data dependencies** comme présenté à la Figure 7.

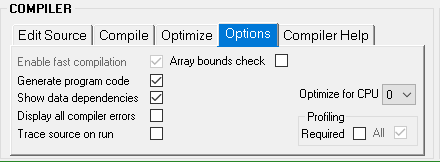


Figure 7 - Option Data dependencies

Cochez UNIQUEMENT l’optimisation **Remove redundant** **code** dans l’onglet **Optimize**.

Créez un programme nommé pipeline4\_1 et écrivez le code suivant. Compilez-le et chargez-le dans la mémoire du simulateur.

program pipeline4\_1

a = 1

b = a

c = 2

end

Copiez la séquence d’instructions générée ci-dessous (sans les adresses des instructions).

Ensuite, cochez l’optimisation **Optimize code dependencies** comme indiqué à laFigure 8. Changez le nom du programme en pipeline4\_2, compilez-le et chargez-le dans la mémoire du simulateur.

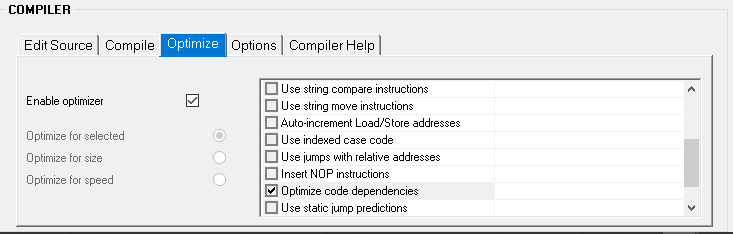


Figure 8 - Optimize code dependencies

Copiez la séquence d’instructions générée ci-dessous (sans les adresses des instructions).

En quoi ces deux séquences d’instructions diffèrent-elles ? Ce changement affecte-il la logique du programme ? Expliquez brièvement les raisons du changement :

Essayez maintenant de voir si la méthode du « réarrangement de la séquence d’instructions » appliquée améliore la situation.

Reprenez le programme précédent, ajoutez une boucle comme indiqué ci-dessous et renommé-le pipeline5\_1 avant de le sauver.

program pipeline5\_1

for n = 1 to 50

a = 1

b = a

c = 2

next

end

Compilez-le deux fois, une fois en cochant l’optimisation **Optimize code dependencies** et l’autre, après l’avoir renommé en pipeline5\_2 et en décochant cette même option.

Exécutez le programme pipeline5\_1 et prenez note des valeurs suivantes :

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |

Exécutez le programme pipeline5\_2 et prenez note des valeurs suivantes :

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |

Y a-t-il une amélioration entre les deux version (exprimez-la en pourcentage) ?

# Table de prédictions de branchement

Le pipeline de CPU utilise une table pour mémoriser les prédictions de branchement et ainsi l’adresses vers laquelle le branchement est prévu. À chaque fois qu’une instruction de saut conditionnel est exécutée cette table est consultée afin de voir à quelle adresse le saut est prévu. Si cette prédiction est fausse, l’adresse est alors calculée et utilisée à la place de celle prévue.

Il s’avère que dans de nombreux cas, l’adresse prévue est correcte. Seules quelques prévisions sont erronées. Les prédictions de branchement permettent donc une amélioration des performances du processeur.

Pour visualizer cette amelioration, entrez le programme ci-dessous et compilez-le en cochant UNIQUEMENT l’option d’optimisation **Remove redundant code**.

program pipeline6

i = 0

for p = 1 to 40

i = i + 1

if i = 10 then

i = 0

r = i

end if

next

end

Chargez le résultat dans la mémoire du simulateur.

Exécutez le programme et prenez note des valeurs suivantes :

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |

Cochez l’option **Enable jump prediction** dans la zone **Optimizations** de la Figure 3 et compilez-le. Faites un **RESET PROGRAM**, **e**xécutez-le et prenez note des valeurs suivantes :

|  |  |
| --- | --- |
| CPI (Clocks Per Instruction) |  |
| SF (Speed‐up Factor) |  |

Voyez-vous une différence ? Y a-t-il une amélioration ?

Dans la fenêtre **Instruction Pipeline, c**liquez sur le bouton **SHOW JUMP TABLE…** comme mis en évidence sur la Figure 9.

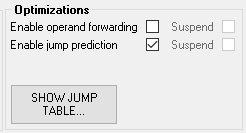


Figure 9 – Enable jump prediction

La fenêtre **Jump Predict Table** apparaît comme indiqué à la Figure 10.

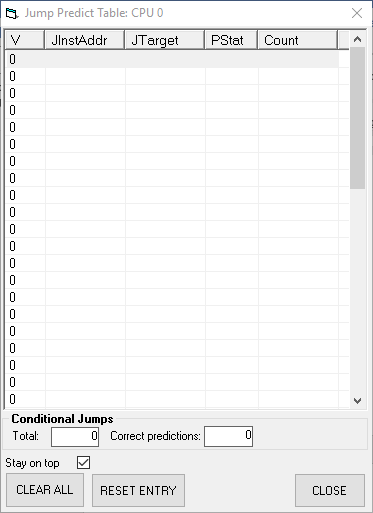


Figure 10 - Jump Predict Table

Cette table conserve une entrée pour chaque instruction de saut conditionnel. Les informations contenues comportent les champs suivants. Pouvez-vous suggérer ce que représente chaque champ ? Inscrivez ces suggestions dans le tableau ci-dessous.

|  |  |
| --- | --- |
| V |  |
| JInstAddr |  |
| JTarget |  |
| PStat |  |
| Count |  |