Pipeline P1

Alexandre Delétraz, Michaël Cheneval

▼ Partie 1

1.1

Questions

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 ?

3

2. Dans le circuit fetch, le signal LR_adr_o vient d'un registre et est connecté au bloc decode au lieu du bloc bank_register, pourquoi ?
3. Dans le circuit decode, le signal adr_reg_d_s est mis dans un registre alors que les signaux adr_reg_n_s, adr_reg_m_s et adr_reg_mem_s sont directement connecté à la sortie, pourquoi ?
4. Dans le circuit decode, les signaux des bus de contrôle sont connectés aux registres avec une porte MUX contrairement aux autres signaux, pourquoi ?
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 ?

1. Il est de +1 car il y a un autre +1 dans le bank register (là où se situent les 3 registres MUX vers la droite).
   Quand on passe du decode à l'execute, on a un coup de clk en plus pour lire ce qui se trouve dans le decode, qui est donc la valeur de PC+2.
2. Pour que le link_register suivre l'instruction. Comme l'architecture permet d'avoir une instruction par coup de clock, il faut que la valeur de LR passe par tous les blocs jusqu'au bank register.
   Cela permet de savoir s'il faut écrire dans le registre en cas de saut ou non. Si la condition est respectée dans l'execute, on sait que l'on va écrire dans le registre LR une fois arrivé au write back.

3. Car on doit pouvoir lire et écrire dans ce registre. On doit donc garder en tampon l'adresse du registre d pour que l'on puisse ensuite écrire dans le bon registre au moment du write back. Quant aux autres registres, on les passe directement dans l'execute pour faire l'opération souhaitée. Le résultat passe ensuite par des tampons jusqu'au write back où l'on va écrire dans le registre d.
4. Car il faut vérifier que l'enable du decode soit bien à 1 avant de pouvoir continuer le processus. Cela permet d'éviter que l'on écrive quand on ne devrait pas, par exemple en cas d'aléa dans les dépendances de données.
5. On ne sait pas.

1.2

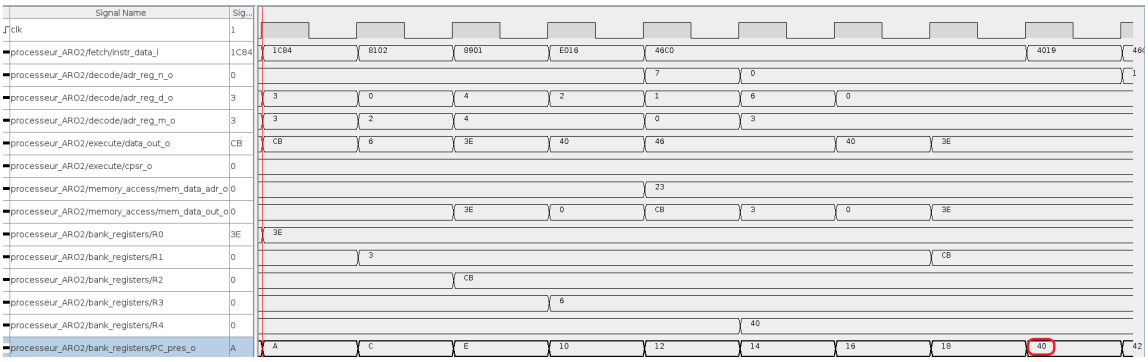
Relevez le chronogramme de l'exécution du code ci-dessus depuis le début du traitement de l'instruction *add r4, r0,#2* jusqu'à la fin du traitement de l'instruction *and r1,r3* Vous devez vous inspirer de l'exemple donné en cours. Votre chronogramme doit comprendre les signaux suivants : clock, PC, sortie du registre de chacun des 5 étages du pipeline.

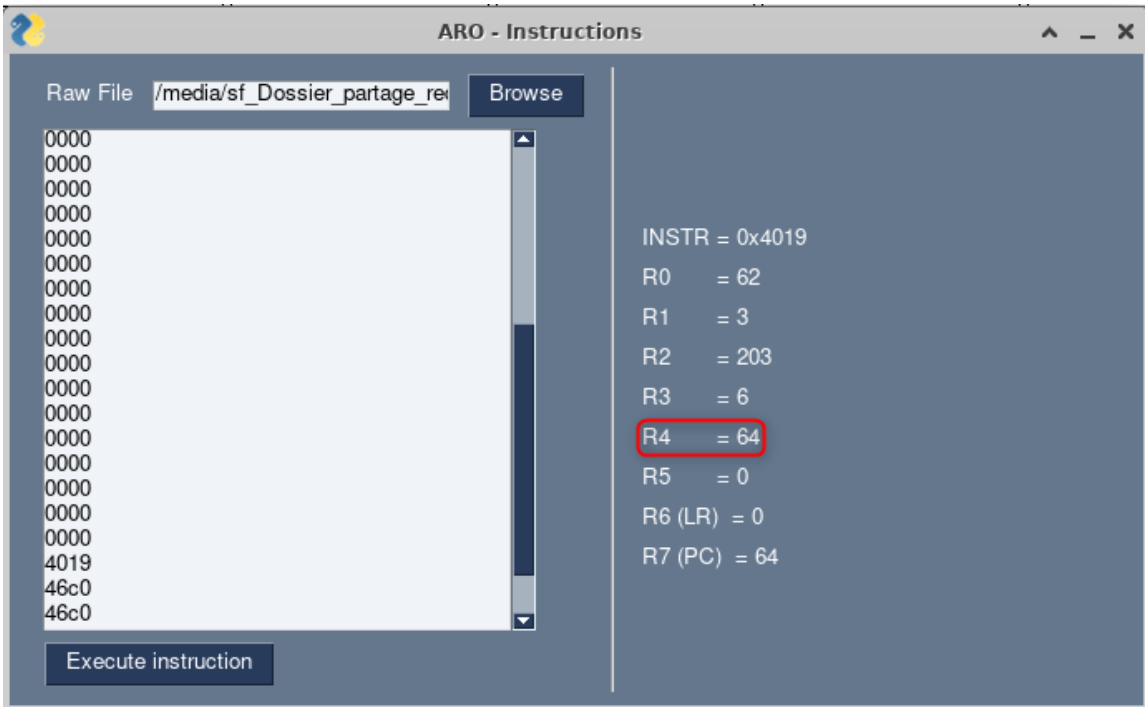
Questions

1. Est-ce que le programme s'exécute correctement ? Est-ce que les registres prennent les bonnes valeurs ?
2. Combien de cycles sont nécessaires pour exécuter ce programme ?

```
add r4, r0, #2
strh r2, [r0, #4*2]
ldrh r1, [r0, #4*2]
b fin
nop
nop
nop
nop
nop
.org 0x40
fin:
and r1, r3
nop
nop
nop
nop
nop
```

1. Le programme s’exécute correctement, mais les registres ne prennent pas les bonnes valeurs (registre R4 incorrect).





2. 8 cycles

1.3

Dans le programme main.S qui vous a été fourni, indiquez en commentaire pour la première partie (depuis *MAIN_START* jusqu'à *B PART_2*), les dépendances de données pour chaque instruction. Relevez le chronogramme de l'exécution du code.

4

Ajoutez le nombre minimum d'instructions *NOP* pour résoudre les aléas de donnée. Relevez le chronogramme de l'exécution du code.

Questions

1. Quelles dépendances posent des problèmes d'aléas ?
2. Combien de cycles sont nécessaires pour résoudre un aléa de donnée ?
3. Quelle est l'IPC ? Le throughput si la clock vaut 4KHz ? La latence ?

```
1  MAIN_START:
2  MOV r0, #1
3  MOV r1, #2
4  MOV r2, #6
5  STRH r0, [r1, #4]   ; r0 @2
6  ADD r4, r2, #1      ; r2 @4
7  ADD r3, r2, #4      ; r2 @4
8  SUB r4, r1, r0
9  ADD r0, r0, #5
10 LSL r2, r2, #1
11 LSL r2, r2, #1      ; r2 @10
12 B PART_2
```

```
MAIN_START:
MOV r0, #1
MOV r1, #2
MOV r2, #6
NOP
STRH r0, [r1, #4]
NOP
NOP
ADD r4, r2, #1
ADD r3, r2, #4
SUB r4, r1, r0
ADD r0, r0, #5
LSL r2, r2, #1
NOP
NOP
```

```
NOP
LSL r2, r2, #1
```

1. Les dépendances de données ne peuvent être que de type RAW. Nous ne prenons donc en compte que celles-ci. Il y a 3 dépendances causant des aléas :
    a. STRH r0, [r1, #4] → I1(R0)
    b. ADD r4, r2, #1 → I3(R2)
    c. LSL r2, r2, #1 → I9(R2)
2. Cela dépend de l'aléa :
    a. 1 cycle en plus
    b. 2 cycles en plus
    c. 3 cycles en plus
3. Réponses :
    a. $IPC = \frac{nbrInstr}{nbrClk} = \frac{10}{20} = 0.5$
    b. $Débit = Fréquence = 4KHz$
    c. $Latence = 5 * \frac{1}{Débit} = \frac{5}{4000} = 0.00125$

1.4

Dans la deuxième partie (depuis l'instruction *B PART_2*) du programme main.S qui vous a été fourni, ajoutez le nombre minumum d'instructions *NOP* pour résoudre les aléas de contrôle. Relevez le chronogramme de l'exécution du code.

Questions

1. Combien de cycles sont nécessaires pour résoudre un aléa de contrôle ?
2. Quelle est l'IPC ? Le throughput si la clock vaut 4KHz ? La latence ?

```
B PART_2

.org 0x40
PART_2:
MOV r0, #3
```

```
MOV r1, #4
MOV r2, #8
b SAUTZERO
ADD r0, r1, r2

.org 0x60
SAUTZERO:
MOV r0, #255
BNE SAUTC
MOV r1, #5

.org 0x80
SAUTC:
MOV r0, #0
BNE NOT_TAKEN
MOV r1, #0
BEQ MAIN_START

.org 0xA0
NOT_TAKEN:
B MAIN_START
MOV r4, #6
```

1. 2 cycles
2. Réponses :
   a. $IPC = \frac{nbrInstr}{nbrClk} = \frac{10}{20} = 0.5$
   b. $Débit = Fréquence = 4KHz$
   c. $Latence = 5 * \frac{1}{Débit} = \frac{5}{4000} = 0.00125$

▼ Partie 2

2.1

Questions

1. Combien de cycles le pipeline doit être bloqué dans le cas d'un aléa de contrôle ?
2. Pourquoi faut-il bloquer le pipeline lorsqu'il y a un aléa de contrôle ?
3. Quels sont les conditions pour qu'un aléa de contrôle ait lieu ?
4. Que se passe-t-il si une instruction génère un aléa de contrôle et un aléa de donnée ?

1. 2 cycles

2. Car lors d'un saut, nous stoppons le programme en cours pour exécuter un sous-programme
3. Il faut qu'il y ait un saut conditionnel dans le programme.
4. Cela dépend de l'aléa prenant le plus de coups de clk à se résoudre. Si l'aléa de donnée prend par exemple 3 coups de clk, une fois qu'il sera résolu, l'aléa de contrôle le sera aussi.

2.2

Questions

1. Quelles instructions génèrent un aléa de contrôle ?
2. Comment les aléas de contrôle influencent les différents enables ?
3. Que se passe-t-il dans le pipeline si un saut est pris ? Quelle est la prochaine instruction exécutée ?

7

4. Pourquoi branch_i est dans les entrées du circuit hazard_detection ?
5. Pourquoi instr_control_i du bloc control_hazard dépend de no_data_hazard_s ?

1. Les instruction de contrôles telles que BEQ ou BNE
2. Cela fait passer les enables à 0 (d'abord decode, puis execute, etc).
3. Il y a une interruption du pipeline de 2 coups de clk. La prochaine instructions sera celle à l'adresse du saut si celui-ci est pris.
4. Afin de pouvoir stopper le pipeline en cas de saut.
5. Si une donnée présente un problème, il faut pouvoir stopper le processus et ne pas transmettre ces données.

2.3

Questions

1. Est-ce que les valeurs dans les registres sont mises à jour correctement et au bon moment ?
2. Quel est l'IPC pour votre programme ?
3. Pourquoi l'instruction *BL* génère en même temps un aléa de contrôle et un aléa de donnée ?

1. Oui
2. $IPC = \frac{nbrInstr}{nbrClk} = \frac{15}{29} = 0.5172$
3. Car elle fait un saut et elle demande à écrire dans un registre.