# Réponse aux questions Processeur pipeliné Partie 1

Départements : TIN

Unité d'enseignement ARO

Auteurs : Timothée Van Hove
Benoit Delay

Professeur : Romuald Mosqueron
Assistant : Mike Meury

Classe : A
Salle de labo : A07

Date : lundi, 30 mai 2022

# Table des matières

1 Analyse et test du processeur ..... 3
1.1 Analyse du processeur - Donnée ..... 3
1.2 Test du processeur – Donnée ..... 5
1.3 Assembleur : dépendances de données - Donnée ..... 8
1.4 Assembleur : aléas de contrôle - Donnée ..... 9
2 Aléas de contrôle ..... 10
2.1 Circuit control_hazard - Donnée ..... 10
2.2 Circuit hazard_detection - Donnée ..... 12
2.3 Test aléas de contrôle - donnée ..... 13

# 1 Analyse et test du processeur

## 1.1 Analyse du processeur - Donnée

Le processeur qui vous a été fourni a été pipeliné à partir du processeur que vous aviez implémenté dans les laboratoires précédents. Certains changements ont dû être opérés pour pouvoir supporter le pipeline. Pour pipeliner un processeur, il ne suffit pas d'ajouter des registres entre chaque bloc. Il faut, par exemple, s'assurer que tous les signaux de contrôle arrivent au bon moment. Le signal execute_control_bus est généré au moment où l'instruction est décodée, mais il est utilisé au moment où l'instruction est exécutée. Il faut donc que le signal de contrôle arrive au même moment que l'instruction dans le bloc execute. Le schéma ci-dessus est un croquis du processeur pipeliné. Les registres sont en gris. Sur le schéma, il y a un grand registre entre les stages du pipeline, or dans Logisim, il y a un registre par signal. Register READ et Register WRITE sont implémentés dans Logisim dans bank_register.

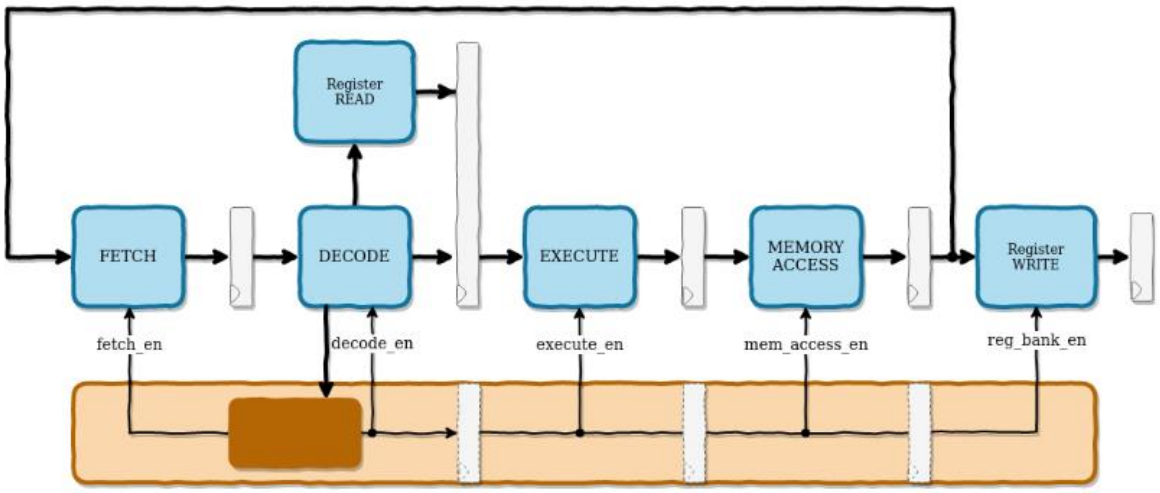



FIGURE 1 – Croquis du processeur pipeliné

Les changements qui ont été faits pour le processeur pipeliné :

- Dans le circuit mult_2, les offsets sont incrémentés de 1 au lieu d'être incrémenté de 2
- Dans le circuit LR_manager, le signal link_en_i passe dans 3 registres au lieu de 1, pour que le signal link_en_d1_s soit généré au bon timing.
- Le signal branch_i est calculé dans memory_access au lieu de fetch car c'est dans ce bloc que les informations sont disponibles pour le calcul.
- Les signaux passent par tous les blocs même s'ils ne sont pas utilisés dans un bloc. Ceci pour assurer que les informations de contrôle arrive en même temps que les données dans le bloc qui les utilisent.

### 1.1.1 Réponse aux 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 ?**

On a mis un registre dans la Bank Register il nous faut 2x plus de temps pour sortir donc on doit faire + 1 pour temporiser vu qu'on ne part pas plus 4 parce qu'on part une instruction après

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 ?**

   Pour pouvoir temporiser un coup de clock vu qu'on est en parallèle. Cela nous permet de ne pas a attendre de devoir attendre que le registre se met a jour lors de l'etape write back et donc de sauvegarder des temps de cycle

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 ?**

Car adr_reg_d est utilisé pour écrire dans un registre. Il faut donc que ce signal passe par les étages suivants du pipeline.

En revanche adr_reg_n, adr_reg_m et adr_reg_mem sont utilisé pour lire des registres dans ce même bloc Decode, donc dans ce même étage du pipeline.

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

   On utilise un MUX pour pouvoir temporiser l'arrêt du processeur et pas que cela gêne le bon fonctionnement si on a un saut conditionnel.

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 ?**

   Vu qu'on a besoin de 3 étages de pipeline pour la multiplication et que pour le processeur on a besoin de 5 étages, si l'on ajoute le multiplicateur pipeliné on additionne les étages ce qui nous donnerait 8 étages de pipeline.

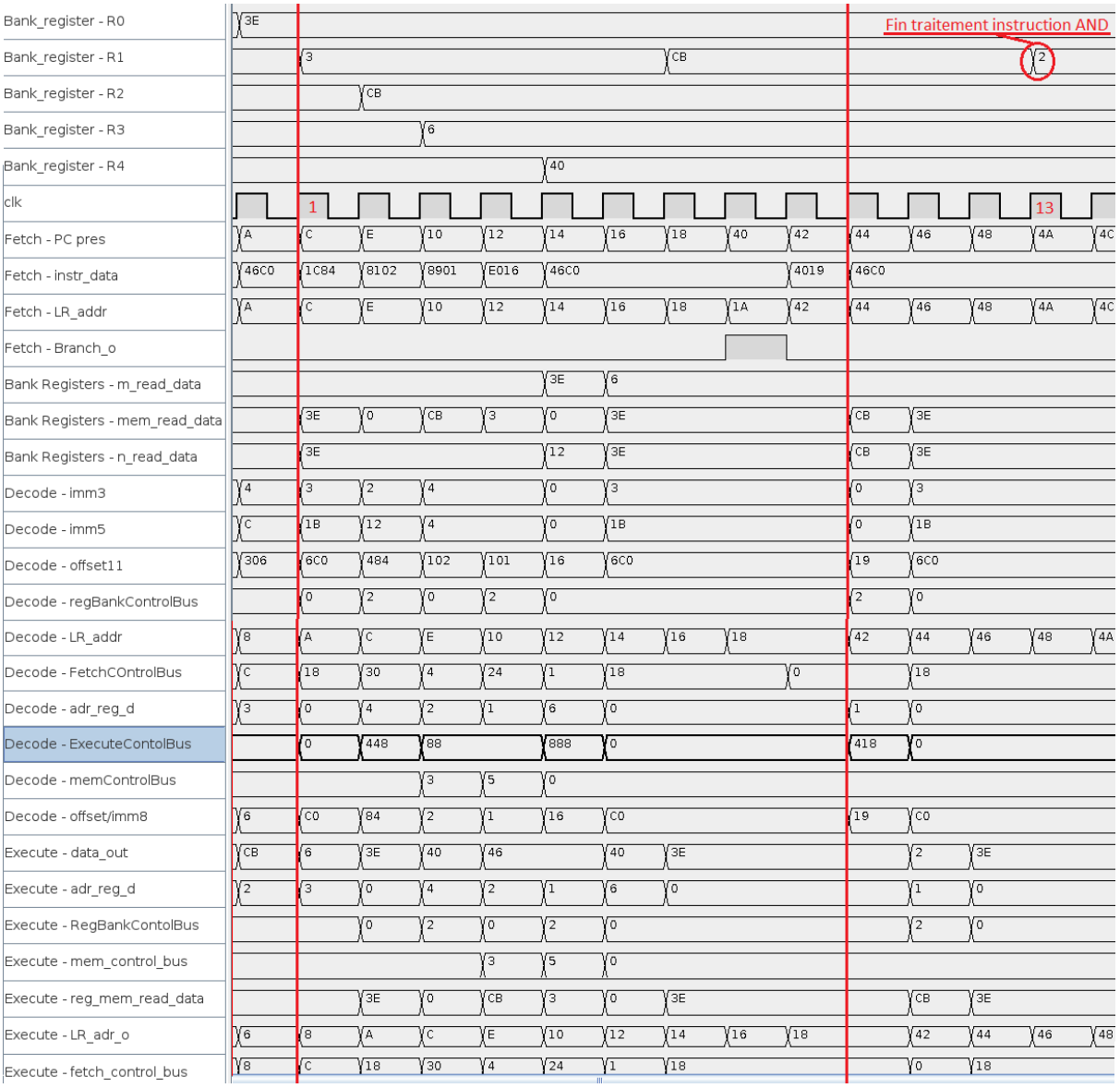
## 1.2 Test du processeur – Donnée

Compilez et testez le programme suivant :

```
@ programme 1
mov r0,#0x3E
mov r1,#3
mov r2,#0xCB
mov r3,#6
nop
@ Partie à analyser
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
@ fin de l'analyse
```

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.

### 1.2.1 Chronogramme complet du programme à analyser



### 1.2.2 Réponse aux questions

1. **Est-ce que le programme s'exécute correctement ? Est-ce que les registres prennent les bonnes valeurs**

Hormis la 5e instruction nop après b fin qui est inutile, les registres prennent les bonnes valeurs :

- `add r4,r0,#2`: R4 prend bien la valeur de 40
- `strh r2,[r0,#4*2] / ldrh r1,[r0,#4*2]`: La Valeur CB best bien récupérée dans r1 après avoir été stockée dans la mémoire de données
- `b fin`: Le saut se fait, et au bon endoit dans la mémoire d'instruction
- `and r1,r3`: L'opération 0011 AND 0110 est bien 0010 => 2 dans R1

2. **Combien de cycles sont nécessaires pour exécuter ce programme ?**

Depuis l'instruction `add r4,r0,#2` jusqu'à la fin de l'exécution de l'instruction `and r1,r3`, Il faut compter 13 cycles (voir chronogramme)

## 1.3 Assembleur : dépendances de données - Donnée

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.

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

### 1.3.1 Réponse aux questions

**3. Quelles dépendances posent des problèmes d'aléas ?**

Comme vu en cours, dans notre architecture, uniquement les dépendances de type RAW posent des problèmes. Voici le digramme des dépendances :

**4. Combien de cycles sont nécessaires pour résoudre un aléa de donnée ?**

En mode arret il faut 4 cycles pour que le write back s'effectue et qu'on puisse utiliser le registre.

**5. Quelle est l'IPC ? Le throughput si la clock vaut 4KHz ? La latence ?**

Latence = niveau pipeline * 1/clock = 5 * 1/4000 = 1,25ms

Débit = 1/ latence = 800 opérations par cycles

9 instructions qui prennent 1 cycle

1 qui prend 3 cycles

1 qui prend 4 cycles

$$IPC = \frac{1}{(0.09 * 3 + 0.09 * 4 + 0.81 * 1} = 0.69\, IPC$$

## 1.4 Assembleur : aléas de contrôle - Donnée

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.

### 1.4.1 Réponse aux questions

1. **Combien de cycles sont nécessaires pour résoudre un aléa de contrôle ?**

On ajoute 3 nop donc il nous faut 3 cycles en plus !

2. **Quelle est l'IPC ? Le throughput si la clock vaut 4KHz ? La latence ?**

Le debit et la latence sont les mêmes qu'au points 1.3. L'ipc en revanche est différentes ! Voici le calcul

10 instructions qui veulent 1 cycle

2 qui prennent 4 cycles

$$IPC = \frac{1}{0.16 * 4 + 0.83 * 1} = 0.68\, IPC$$

# 2 Aléas de contrôle

## 2.1 Circuit control_hazard - Donnée

Ce circuit permet de détecter si le pipeline doit être bloqué à cause d'un aléa de contrôle. Vous devez compléter le circuit de ce bloc afin de générer un signal no_ctl_hazard_o qui indique qu'il n'y pas d'aléa de contrôle. Lorsqu'une instruction qui génère un aléa de contrôle est détectée (entrée instr_control_i), la sortie o_ctl_hazard_o doit être mise à 0 pendant un certain nombre de coups de clock. Puis le signal doit de nouveau être à 1 pendant au minimum 1 coup de clock pour laisser la prochaine instruction être fetch-ée.

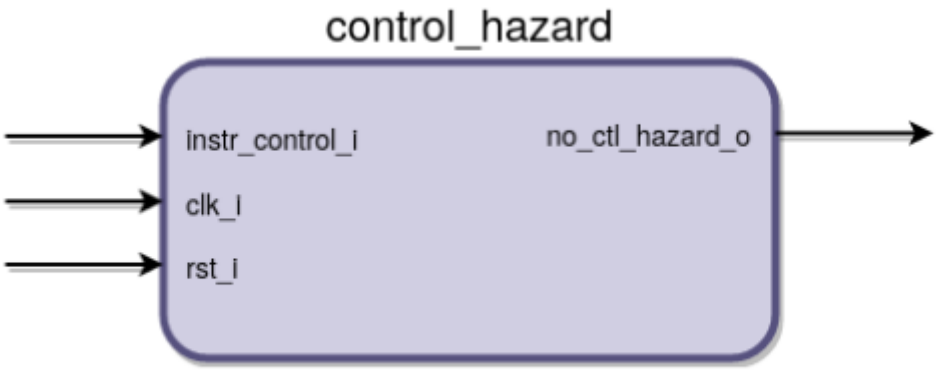



FIGURE 2 – Entrées/sorties du bloc control_hazard

Description des différentes entrées/sorties du bloc :

| Nom I/O | Description |
|---|---|
| instr_control_i | Indique que l'instruction en cours de décodage est une instruction de contrôle |
| clk_i | Clock du système |
| rst_i | Reset asynchrone du système |
| no_ctl_hazard_o | Indique s'il n'y a pas d'aléas de contrôle pour cette instruction |

Comme précédemment, répondez aux questions ci-dessous puis transposez vos réponses sur Logisim.

### 2.1.1 Réponse aux questions

1. **Combien de cycles le pipeline doit être bloqué dans le cas d'un aléa de contrôle ?**

3 cycles, le temps que le pc soit mis a jour dans le pc et que le saut puisse être effectué.

2. **Pourquoi faut-il bloquer le pipeline lorsqu'il y a un aléa de contrôle ?**

Le CPSR est dans le bloc Execute. Le calcul de l'adresse du saut se fait aussi dans le bloc Execute. Donc dans le cas d'un saut, le bloc fetch ne peut pas savoir :

1. Quelle sera l'adresse du saut
2. Est-ce que le saut sera effectué (si saut conditionnel)

Donc le blocage pipeline permet de laisser les coups d'horloge nécessaires à la résolution du saut.

3. **Quels sont les conditions pour qu'un aléa de contrôle ait lieu ?**

Il faut qu'il y ait un saut qui soit pris et des instructions a la suite du saut.

4. **Que se passe-t-il si une instruction génère un aléa de contrôle et un aléa de donnée ?**

Comme nous devons avoir les bonnes valeurs dans les registres pour calculer l'adresse, l'aléa de donnée doit être résolu en premier. Ensuite, on peut résoudre l'aléa de contrôle.

## 2.2 Circuit hazard_detection - Donnée

Ce circuit est instancié dans main_control_unit qui est lui dans le bloc decode. La plupart des connections de ce bloc sont déjà effectuées. Dans le circuit main_control_unit vous devez ajouter les connections pour le signal instr_control_s. Ce signal indique qu'une instruction va générer un aléa de contrôle. Les signaux du bloc hazard_detection sont décrits ici.

Description des différents signaux du bloc :

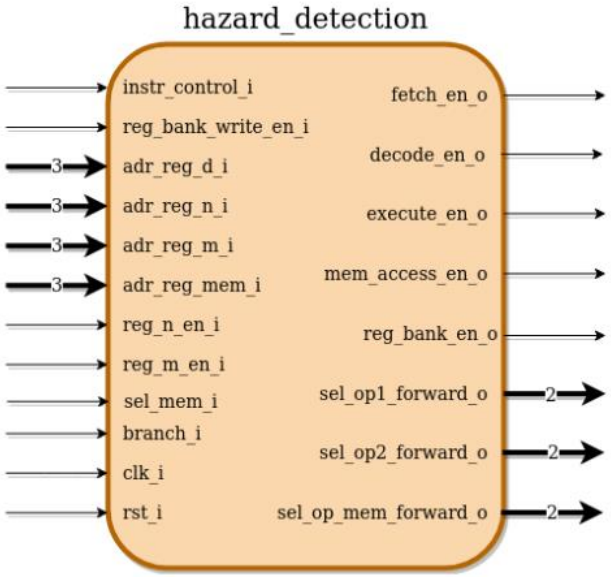



FIGURE 3 – Entrées/sorties du bloc hazard_detection

| Nom I/O | Description |
|---|---|
| instr_control_i | Indique que l'instruction en cours de décodage est une instruction de contrôle. |
| reg_bank_write_en_i | Indique si l'instruction requiert l'écriture d'un registre |
| adr_reg_d_i | Registre à écrire |
| adr_reg_n_i | Registre à lire pour l'opérande 1 |
| adr_reg_m_i | Registre à lire pour l'opérande 2 |
| adr_reg_mem_i | Registre à lire pour l'opération mémoire |
| reg_n_en_i | Indique si l'opérande 1 est lue d'un registre |
| reg_m_en_i | Indique si l'opérande 2 est lue d'un registre |
| reg_mem_en_i | Indique si l'opération mémoire lit une valeur d'un registre |
| sel_mem_i | Indique si cette instruction est une instruction mémoire |
| branch_i | Indique si l'instruction en cours de décodage suit un saut qui a été pris |
| clk_i | Clock du système |
| rst_i | Reset asynchrone du système |
| fetch_en_o | Enable du bloc fetch |
| decode_en_o | Enable du bloc decode |
| execute_en_o | Enable du bloc execute |
| mem_access_en_o | Enable du bloc memory_access |
| reg_bank_en_o | Enable du bloc bank_register |
| sel_op1_forward_o | Sélection d'une donnée forward-ée pour l'opérande 1 |
| sel_op2_forward_o | Sélection d'une donnée forward-ée pour l'opérande 2 |
| sel_op_mem_forward_o | Sélection d'une donnée forward-ée pour l'opérande des instructions mémoire |

### 2.2.1 Réponse aux questions

1. **Quelles instructions génèrent un aléa de contrôle ?**

B, BEQ, BL sont les instructions qui génèrent des aléas.

2. **Comment les aléas de contrôle influencent les différents enables ?**

Les aléas de contrôle sont résolus quand il n'y a pas des aléas de données.

3. **Que se passe-t-il dans le pipeline si un saut est pris ? Quelle est la prochaine instruction exécutée ?**

S'il y a un saut la prochaine instruction est fetch jusqu'à ce que le pc a été mis a jour puis les instructions fetch jusque-là sont abandonnées.

4. **Pourquoi branch_i est dans les entrées du circuit hazard_detection ?**

Pour savoir si on doit sauter ou non est comme ça on sait si on doit avoir un saut.

5. **Pourquoi instr_control_i du bloc control_hazard dépend de no_data_hazard_s ?**

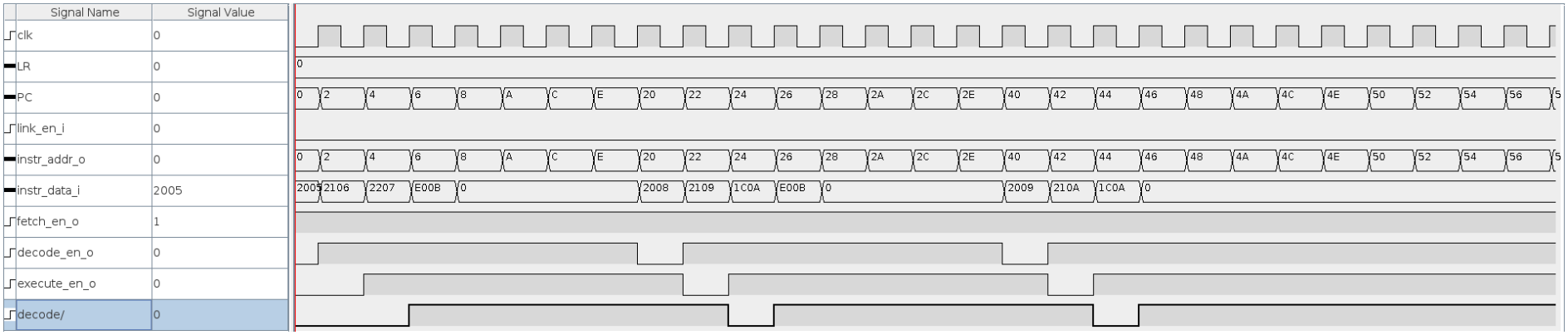
Comme on l'a vu au point 2, on évalue les aléas de contrôle s'il n'y a pas eu de d'alea de donnée. Il nous faut donc vérifier qu'il n'y a pas de contrôle de donnée préalable.

## 2.3 Test aléas de contrôle - donnée

Ecrivez un programme qui contient des aléas de contrôle. Tester votre programme en faisant un chronogramme. Eviter pour le moment l'instruction BL car elle génère un aléa de donnée et un aléa de contrôle.

### 2.3.1 Réponse aux questions

1. **Est-ce que les valeurs dans les registres sont mises à jour correctement et au bon moment ?**




On peut voir que l'on fetch bien les instructions et qu'au bout de 4 nop on jump ! Ce qui confirme qu'il y a bien une détection d'aléa de contrôle !

2. **Quel est l'IPC pour votre programme ?**

On a : 9 instructions à 1 cycle, 2 instructions à 4 cycles

$$IPC = \frac{1}{0.18 * 4 + 0.81 * 1} = 0.65\ IPC$$

3. **Pourquoi l'instruction BL génère en même temps un aléa de contrôle et un aléa de donnée ?**

L'instruction BL est composée de 2 instructions : bl_msb et bl_lsb. bl_msb calcule une partie de l'adresse du saut et l'écrit dans le LR, puis bl_lsb lit le LR et calcule d'adresse du saut complète. Il y a donc un aléa de données, car il faut que bl_msb ait fini d'écrire avant que bl_lsb vienne lire dans le LR Il y a aussi un aléa de contrôle, car il faut attendre que la première partie de l'adresse du saut soit calculée via le bloc execute.