POLYTECH
NICE-SOPHIA

Systèmes à Microprocesseurs

Cycle Ingénieur Troisième Année

Sébastien Bilavarn

Plan

- Ch1 – Représentation de l'information
- Ch2 – ARM Instruction Set Architecture
- Ch3 – Accès aux données
- Ch4 – Programmation structurée
- Ch5 – Cycle d'exécution
- Ch6 – Codage binaire
- Ch7 – Microcontrôleur ARM Cortex-M

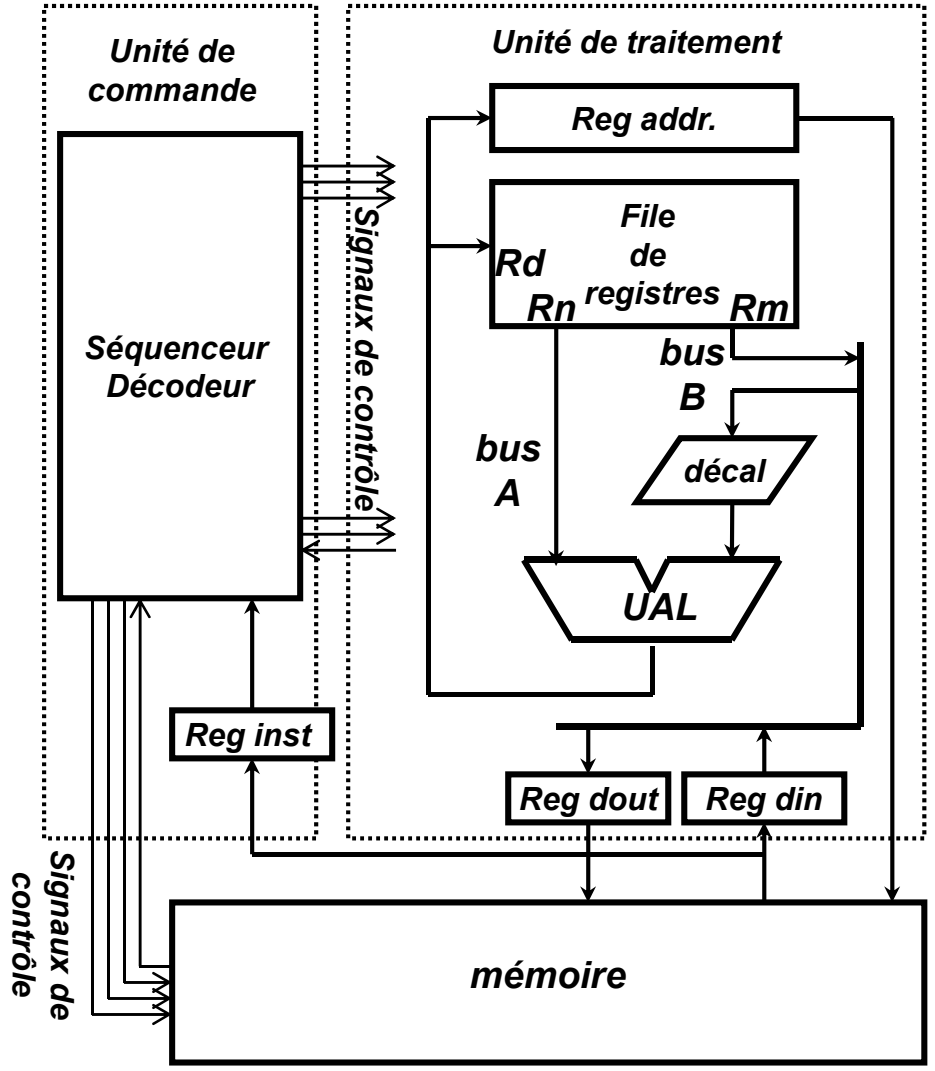
POLYTECH
NICE-SOPHIA

Exécution du processeur ARM

- Rôle de l'unité de commande
  - Cycle instruction
  - Exécution pipeline

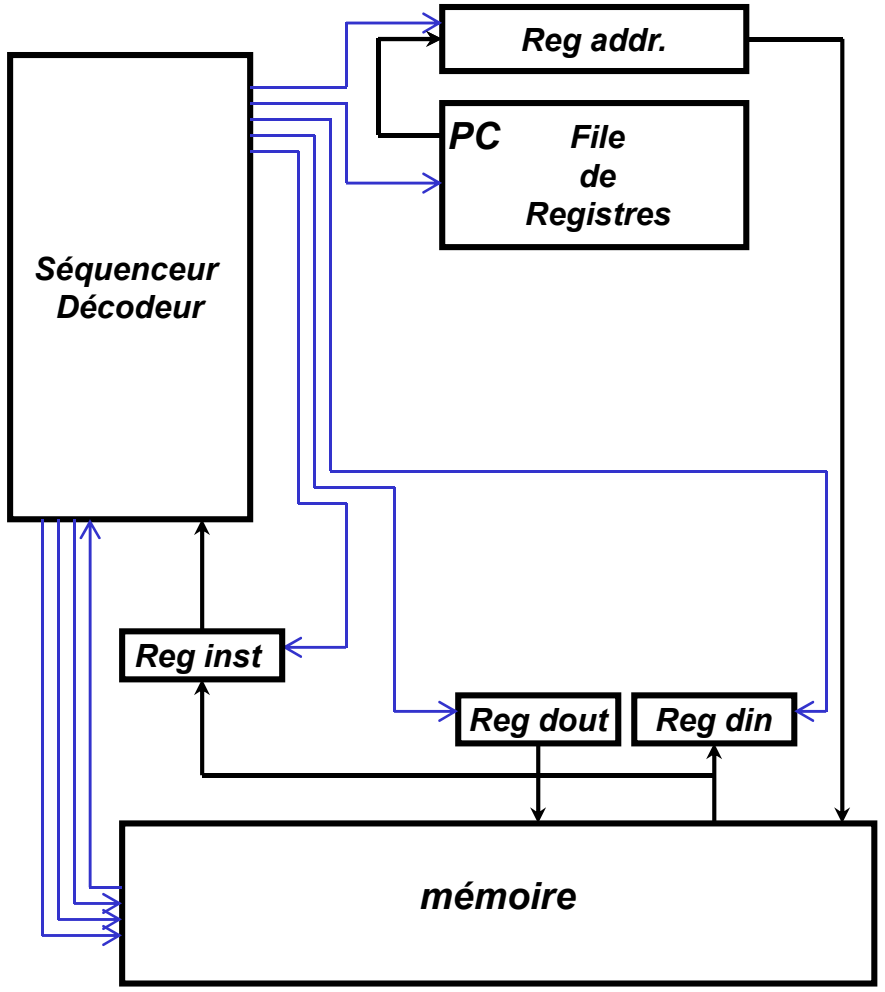
Rôle de l'unité de commande

- Contrôle des accès mémoire
  - Pour l'accès aux instructions/données
    - Positionne les signaux nécessaires pour un accès aux instructions/données en mémoire
- Décodage des instructions
  - Interprétation des instructions
    - Processus de transformation d'une instruction en signaux de commande
- Contrôle de l'unité de traitement
  - Pour l'exécution d'une instruction
    - Positionne les signaux nécessaires pour l'exécution d'une instruction



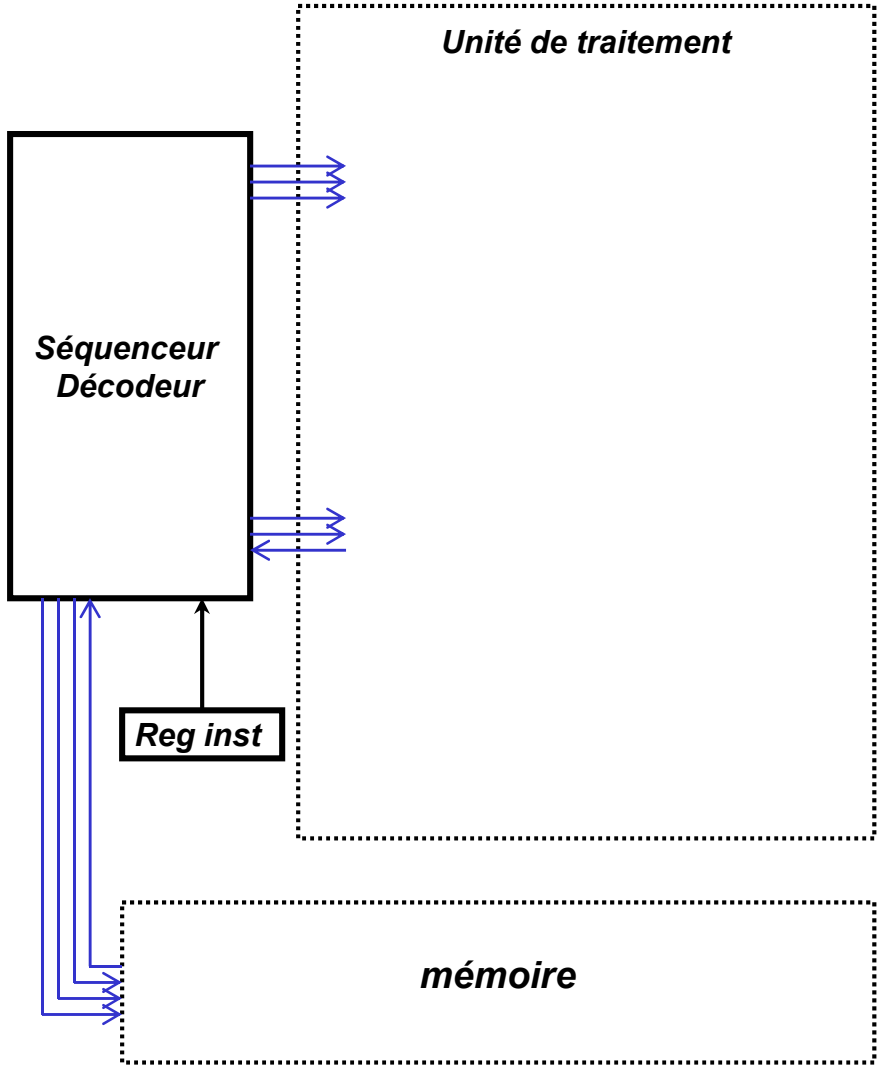
Rôle de l'unité de commande

- **Contrôle des accès mémoire**
  - **Accès aux instructions**
    - **Pour l'exécution du programme, pendant la phase de recherche d'une instruction en mémoire**
    - **Registre d'adresses: contient l'adresse de l'instruction à exécuter**
    - **Registre d'instructions: l'instruction correspondante est renvoyée dans le registre d'instruction pour décodage**
  - **Accès aux données**
    - **Pendant l'exécution d'une instruction load/store.**
    - **Registres de données (Reg din, Reg dout)**
  - **Génération des signaux**
    - **positionne tous les signaux de contrôle mémoire nécessaire (nMREQ, nRW, MAS, registres)**



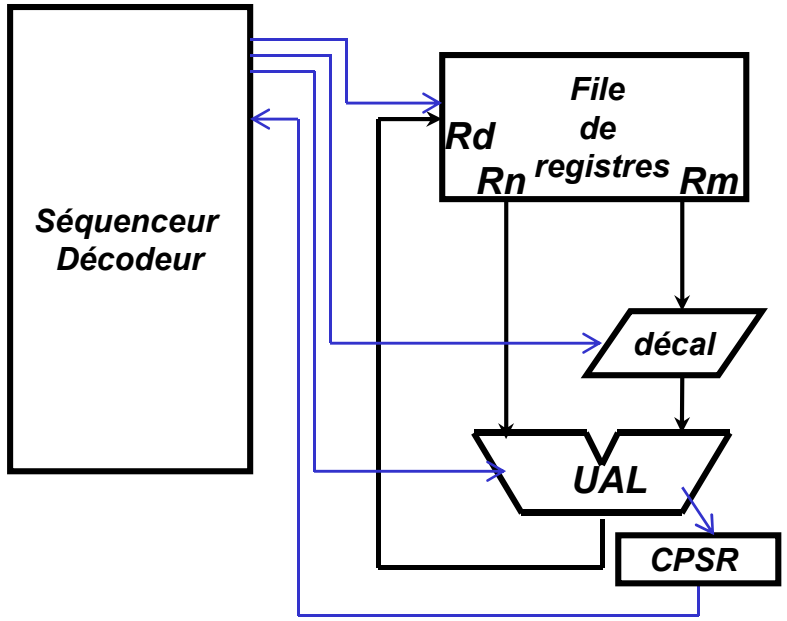
Rôle de l'unité de commande

- **Décodage des instructions**
  - **Convertir une instruction en signaux de commande**
    - **Le décodage consiste à transformer la représentation binaire 32 bits d'une instruction en signaux synchronisés**
    - **Pour le séquencement de l'unité de traitement**
    - **Pour le séquencement de la mémoire**
  - **Processus complexe qui dépend de nombreux paramètres**
    - **Type de l'instruction (traitement, transfert, etc)**
    - **Opérandes (valeur immédiate, registre, registre avec décalage)**
    - **Bits du registre d'état pour évaluation d'une condition d'exécution (ex: branchement si egal)**
    - **Mode d'adressage**
    - **Type de données**



Rôle de l'unité de commande

- **Contrôle de l'unité de traitement**
  - **Circulation des données dans l'unité de traitement**
    - **Sélection des registres dans la file de registres pour chaque opérande (Rn, Rm) et pour le résultat (Rd)**
  - **Sélection des opérations**
    - **Décalage (arithmétique, logique, rotation, nombre de décalages)**
    - **UAL (arithmétique, logique, fonction)**
  - **Exécution conditionnelle des instructions**
    - **Registre d'état (Current Program Status Register)**



Cycle instruction du ARM7

- 1. Phase de recherche ("fetch")
  - Recherche de l'instruction en mémoire
- 2. Phase d'analyse ("decode")
  - Décodage de l'instruction: analyse de la représentation binaire 32 bit de l'instruction et de ses opérandes pour mise en œuvre des signaux par l'unité de commande
  - Pour les phases suivantes, 2 cas possibles selon qu'il s'agit d'une instruction de traitement (sollicite l'unité de traitement) ou d'une instruction d'accès mémoire (sollicite les unités de traitement et mémoire)

Pour une instruction de traitement de données:

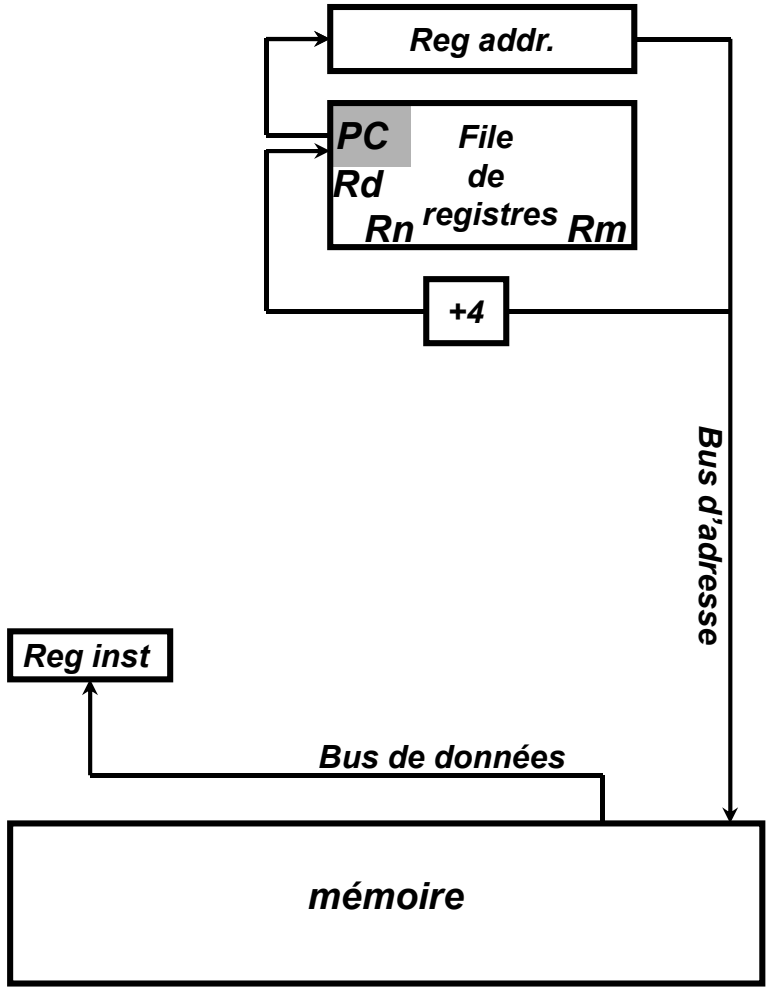
- 3. Phase d'exécution ("execute")
  - Circulation des données, sélection des opérations, exécution conditionnelle

Pour une instruction d'accès mémoire:

- 3. Phase de calcul d'adresse ("address calc")
  - L'adresse est calculée par l'UAL et stockée dans le registre d'adresse
- 4. Phase d'accès mémoire ("data transfer")
  - L'adresse est envoyée à la mémoire qui lit/écrit la donnée

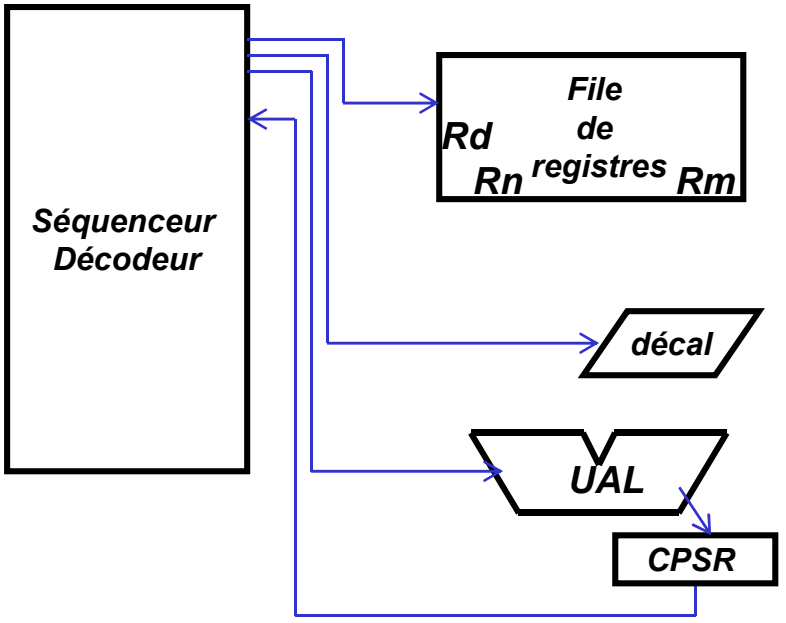
Cycle d'instruction

- Phase de recherche ("fetch")
  - Chargement du registre d'adresse
    - L'adresse de l'instruction à exécuter est contenue dans le registre PC (Program Counter)
    - Cette adresse est copiée dans le registre d'adresse
  - Lecture mémoire
    - L'adresse est placée sur le bus d'adresse
    - On calcule l'adresse de l'instruction suivante (PC+4)
  - Chargement du registre d'instructions
    - L'instruction est disponible sur le bus de données et copiée dans le registre d'instruction
    - PC est mis à jour (adresse + 4)



Cycle d'instruction

- Phase d'analyse ("decode")
  - Lecture du registre d'instruction
    - Le mot de 32 bits représentant l'instruction est extrait du registre d'instruction
    - L'instruction est décodée pour déterminer la séquence d'action à effectuer et avec quelles données.
  - Prise en compte du registre d'état
    - Instructions conditionnelles
  - Sélection opérandes source / destination
    - Sélection opérandes source (valeur immédiate ou sélection des registres Rn et Rm)
    - Résultat (sélection de Rd)
  - Sélection des opérations à effectuer
    - UAL (logic/arith, function, invA, invB, Cin)
    - Décaleur (logic/arith, shift/rot, left/right, amount)



Cycle instruction du ARM7

- 1. Phase de recherche ("fetch")
- 2. Phase d'analyse ("decode")

Pour une instruction de traitement de données:

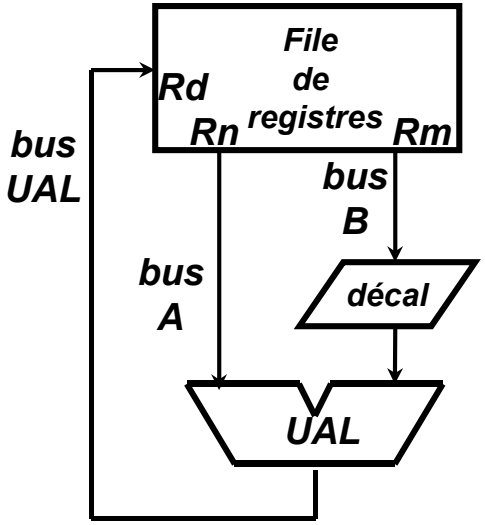
- 3. Phase d'exécution ("execute")

Pour une instruction d'accès mémoire:

- 3. Phase de calcul d'adresse ("address calc")
- 4. Phase d'accès mémoire ("data transfer")

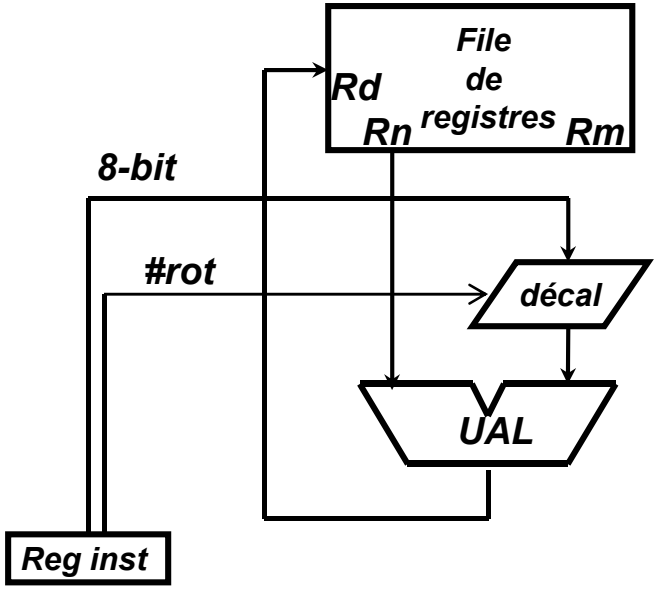
Cycle d'instruction

- Instruction de traitement: phase d'exécution
  - Cas d'une instruction de type ADD Rd, Rn, Rm, *shift*
    - Les opérandes sont lus dans la file de registres. Rn est présenté sur la première entrée de l'UAL (bus A). Rm est présenté sur l'entrée du décaleur (bus B).
    - Une opération de décalage est réalisée ainsi qu'une opération arithmétique ou logique.
    - Le résultat est renvoyé dans le registre de destination (Rd)



Cycle d'instruction

- Instruction de traitement: phase d'exécution
  - Cas d'une instruction de type ADD Rd, Rn, *#litteral*
    - Rn est lu dans la file de registres
    - *#litteral* est obtenu à partir de l''instruction (il est codé dans l'appel à l'instruction sur 8 bits+rotation)
    - Calcul de l'opération arithmétique ou logique.
    - Le résultat est renvoyé dans le registre de destination (Rd)



Cycle instruction du ARM7

- 1. Phase de recherche ("fetch")
- 2. Phase d'analyse ("decode")

Pour une instruction de traitement de données:

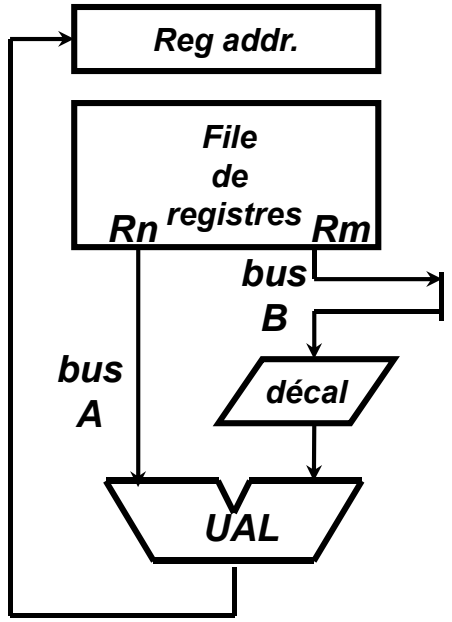
- 3. Phase d'exécution ("execute")

Pour une instruction d'accès mémoire:

- 3. Phase de calcul d'adresse ("address calc")
- 4. Phase d'accès mémoire ("data transfer")

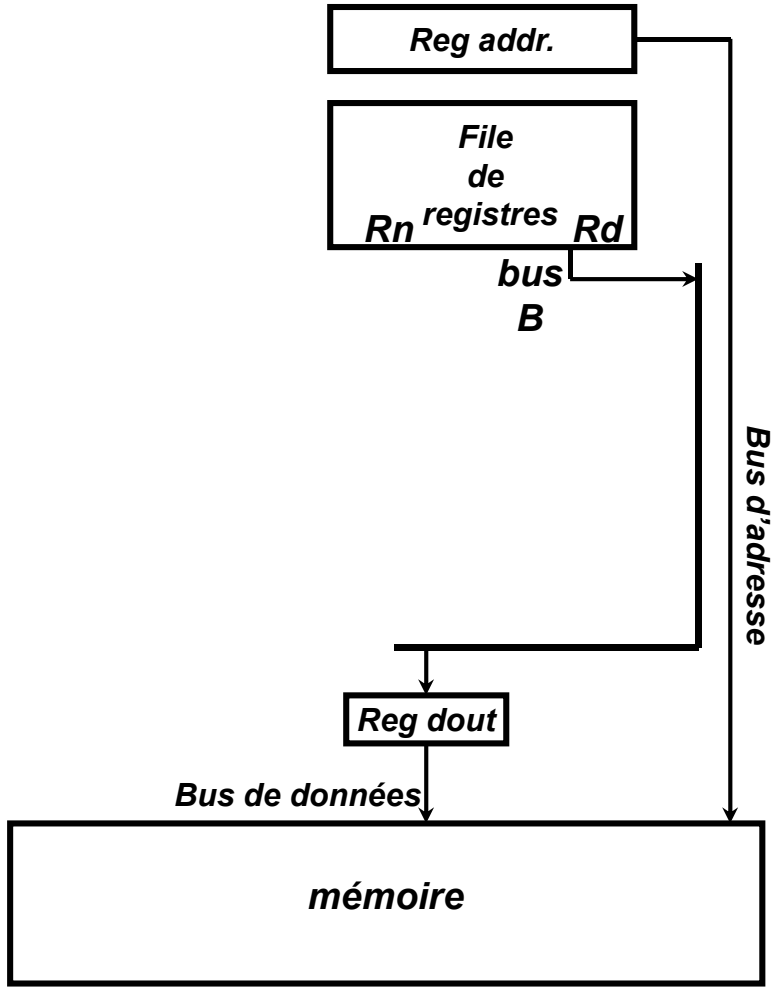
Cycle d'instruction

- Cas d'une instruction de type STR Rd, [Rn, +/-Rm, *shift*]
  - Phase de calcul d'adresse
    - Le calcul d'adresse est effectué par l'UAL. Le résultat du calcul est renvoyé dans le registre d'adresse.



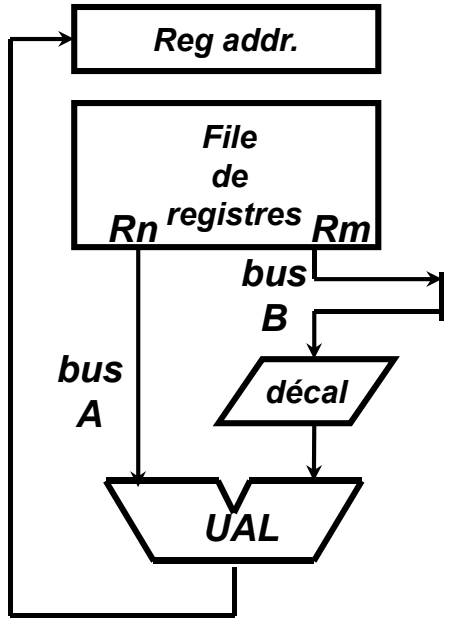
Cycle d'instruction

- Cas d'une instruction de type STR Rd, [Rn, +/-Rm, *shift*]
  - Phase d'accès
    - L'adresse d'écriture contenue dans le registre d'adresse est envoyée sur le bus d'adresse
    - La valeur du registre à copier (Rd) en mémoire est copiée dans le registre Reg dout
    - La donnée est envoyée sur le bus de données et écrite en mémoire



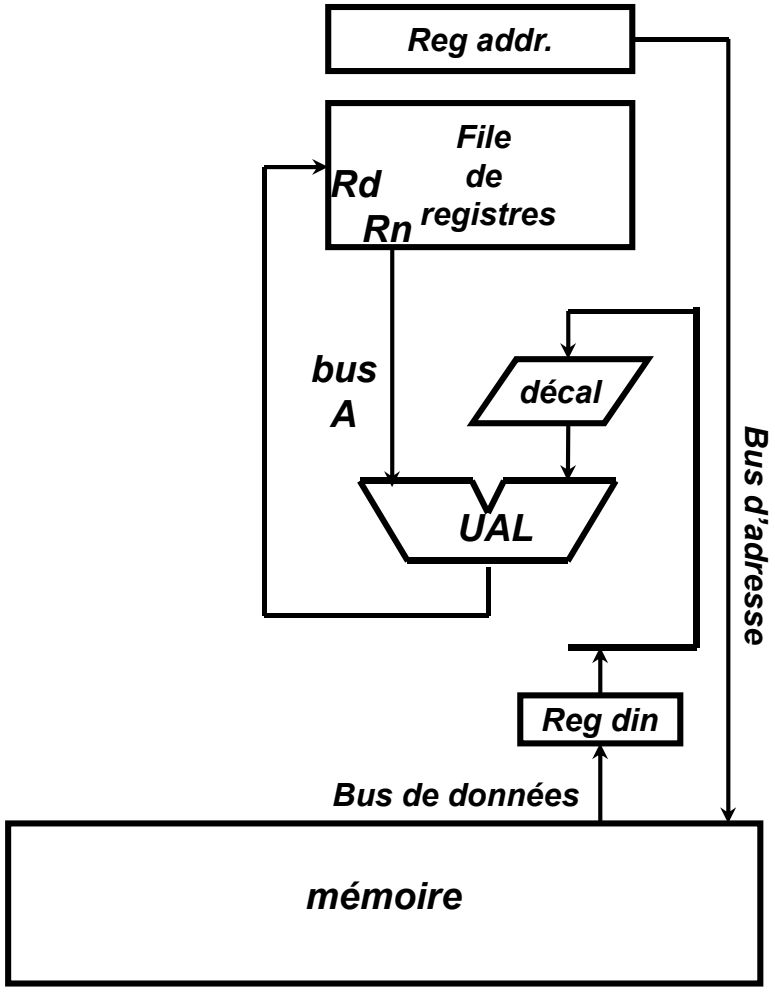
Cycle d'instruction

- Cas d'une instruction de type LDR Rd, [Rn, +/-Rm, *shift*]
  - Phase de calcul d'adresse
    - Le calcul d'adresse est effectué par l'UAL. Le résultat du calcul est renvoyé dans le registre d'adresse.



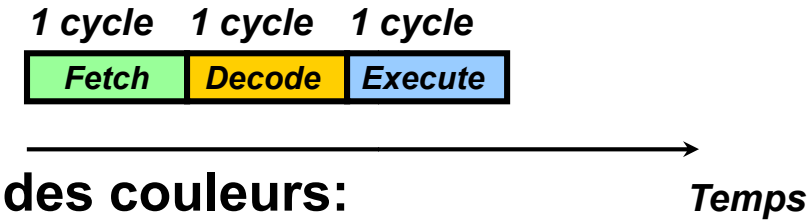
Cycle d'instruction

- Cas d'une instruction de type LDR Rd, [Rn, +/-Rm, *shift*]
  - Phase d'accès
    - L'adresse de lecture est envoyée sur le bus d'adresse
    - La donnée est lue en mémoire
    - La donnée est envoyée sur le bus de données
    - La donnée est acheminée vers le registre de destination (Rd)



Exécution pipeline

- Sur l'ARM7, les phases "fetch", "decode" et "execute" prennent chacune un cycle d'horloge
- Chaque instruction de traitement s'exécute donc en trois cycles d'horloge



Convention des couleurs:

- **Vert: utilisation de la mémoire (fetch lit les instructions en mémoire)**
- **Orange: utilisation du décodeur/séquenceur (analyse des instructions)**
- **Bleu: utilisation de l'unité de traitement (exécution d'une instruction de traitement)**

Exécution pipeline

- Combien de temps pour exécuter la séquence d'instructions suivante ?

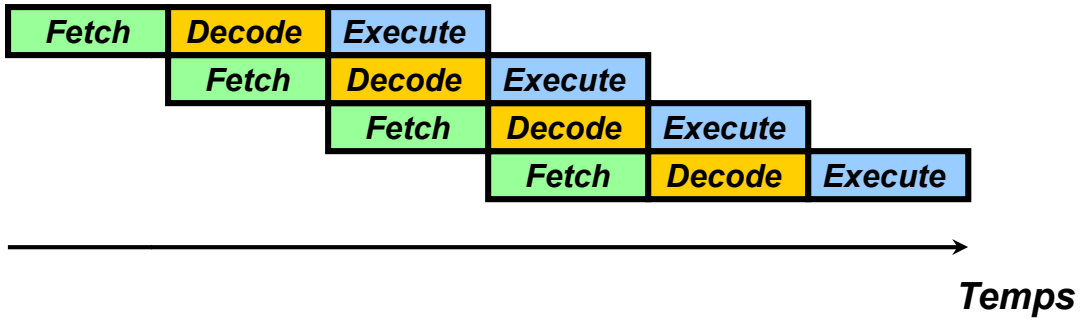
```
MOV r5, r3, LSL #1
ADD r5, r5, r3, LSL #3
MOV r7, r5, LSL #1
ADD r5, r5, r3, LSL #3
```

- 4 instructions x 3 = 12 cycles d'horloge ?

Exécution pipeline

- Réponse: 6 cycles d'horloge !!

```
MOV r5, r3, LSL #1
ADD r5, r5, r3, LSL #3
MOV r7, r5, LSL #1
ADD r5, r5, r3, LSL #3
```

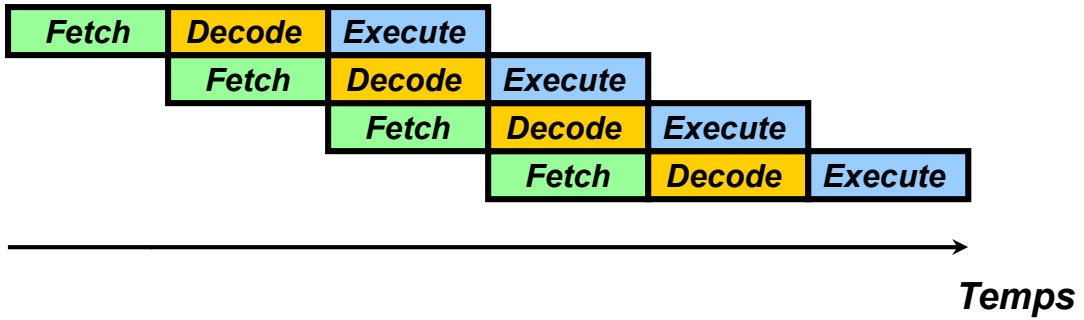


- **Le pipeline permet de recouvrir les phases d'exécution**
  - **La division en phases indépendantes (fetch, decode, execute) permet de démarrer l'exécution d'une instruction avant que la précédente ne soit terminée**
  - **On peut démarrer la phase fetch dès la phase decode de l'instr précédente**
  - **On peut démarrer la phase decode dès la phase execute de l'instr précédente**

Exécution pipeline

- Réponse: 6 cycles d'horloge !!

```
MOV r5, r3, LSL #1
ADD r5, r5, r3, LSL #3
MOV r7, r5, LSL #1
ADD r5, r5, r3, LSL #3
```

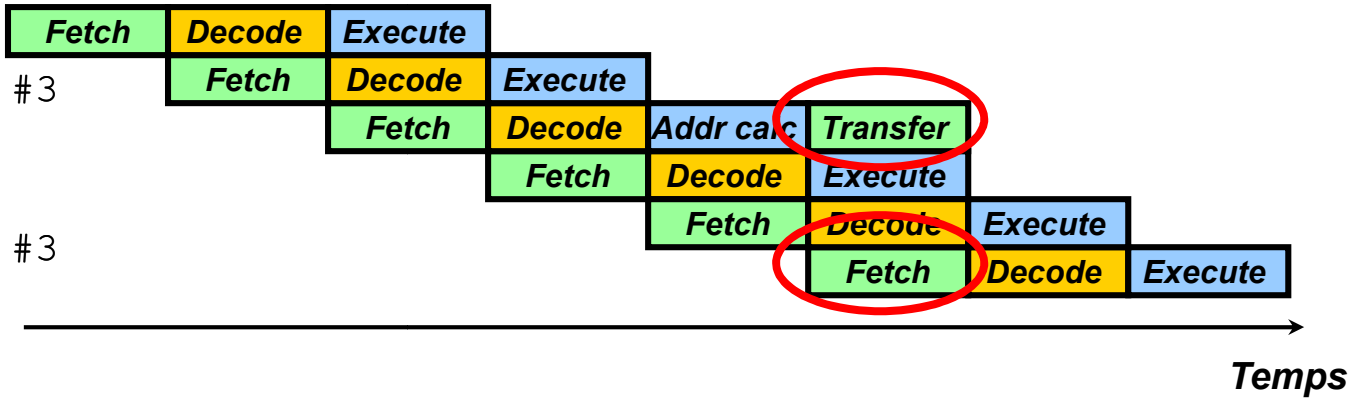


- **On peut exécuter une instruction par cycle d'horloge**
  - **A partir du cycle 3, une phase d'exécution se termine à chaque cycle d'horloge**
  - **On utilise au mieux les ressources: 3 phases s'exécutent parfois simultanément**
  - **Condition: ne pas avoir 2 fois la même couleur dans le même cycle!!**

Exécution pipeline

Ralentissement du pipeline: cas des instructions d'accès mémoire (1)

```
MOV r5, r3, LSL #1
ADD r5, r5, r3, LSL #3
LDR r8, [r2, #12]
MOV r7, r8, LSL #1
ADD r7, r7, r8, LSL #3
SUB r0, r7, r5
```



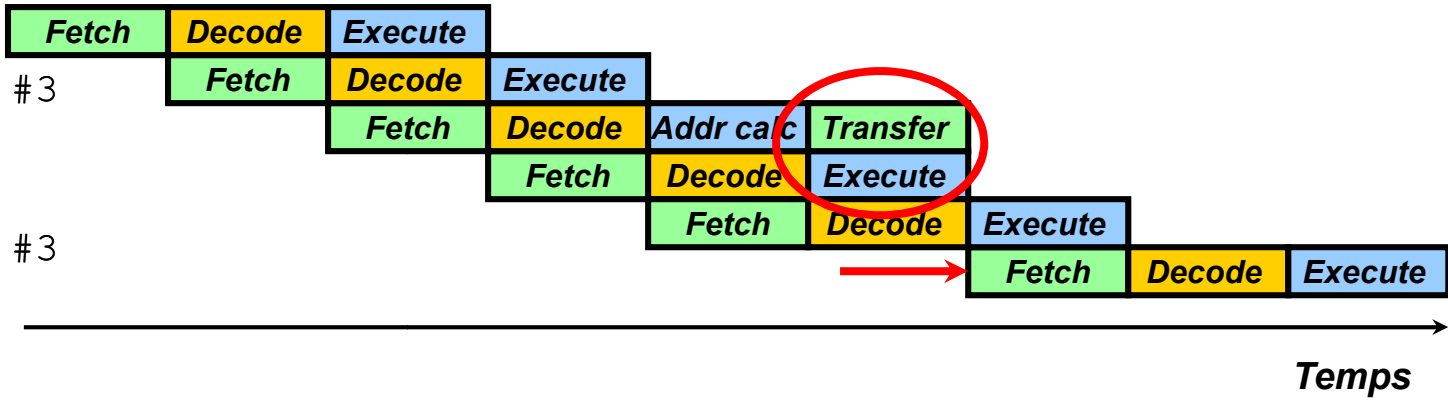
On insère une instruction d'accès mémoire (LDR r8, [r2, #12])

- La phase addr calc utilise l'unité de traitement (bleu)
- La phase de transfert utilise la mémoire (vert)
- Les deux étapes transfert (LDR) et fetch (SUB) ne peuvent pas utiliser la mémoire simultanément

Exécution pipeline

- **Ralentissement du pipeline: cas des instructions d'accès mémoire (2)**

```
MOV r5, r3, LSL #1
ADD r5, r5, r3, LSL #3
LDR r8, [r2, #12]
MOV r7, r8, LSL #1
ADD r7, r7, r8, LSL #3
SUB r0, r7, r5
```



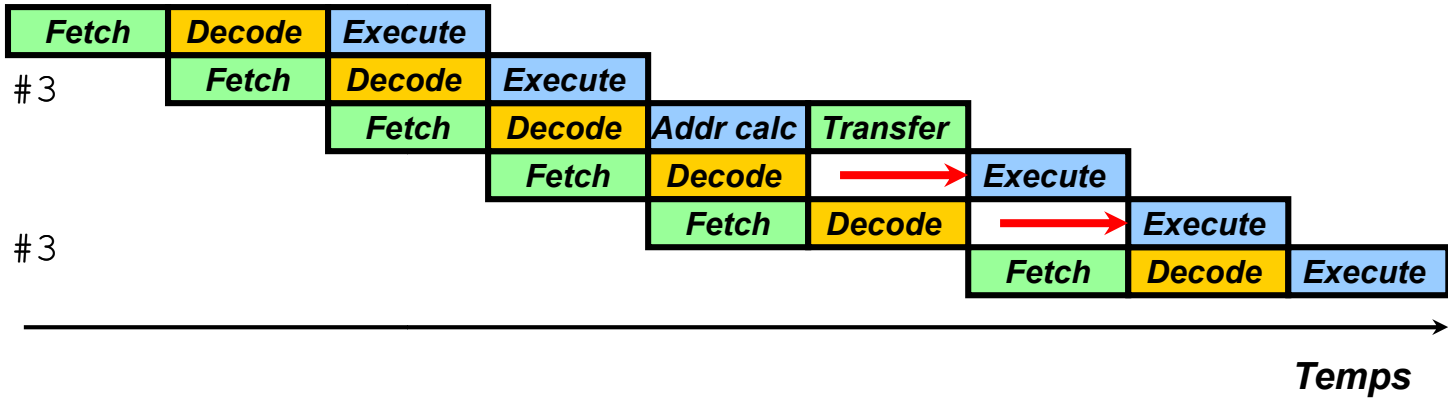
Solution: décaler la dernière instruction de 1 cycle

- **Décaler les phases de l'instruction SUB de 1 cycle**
- **MAIS: l'instruction suivant LDR (MOV) a besoin de la valeur du registre r8 qui n'est pas disponible (transfert non terminé), elle ne peut donc pas s'exécuter.**
- **Il faut à nouveau décaler les phases d'exécution de l'instruction MOV, et celle de l'instruction ADD**

Exécution pipeline

Ralentissement du pipeline: cas des instructions d'accès mémoire (3)

```
MOV r5, r3, LSL #1
ADD r5, r5, r3, LSL #3
LDR r8, [r2, #12]
MOV r7, r8, LSL #1
ADD r7, r7, r8, LSL #3
SUB r0, r7, r5
```

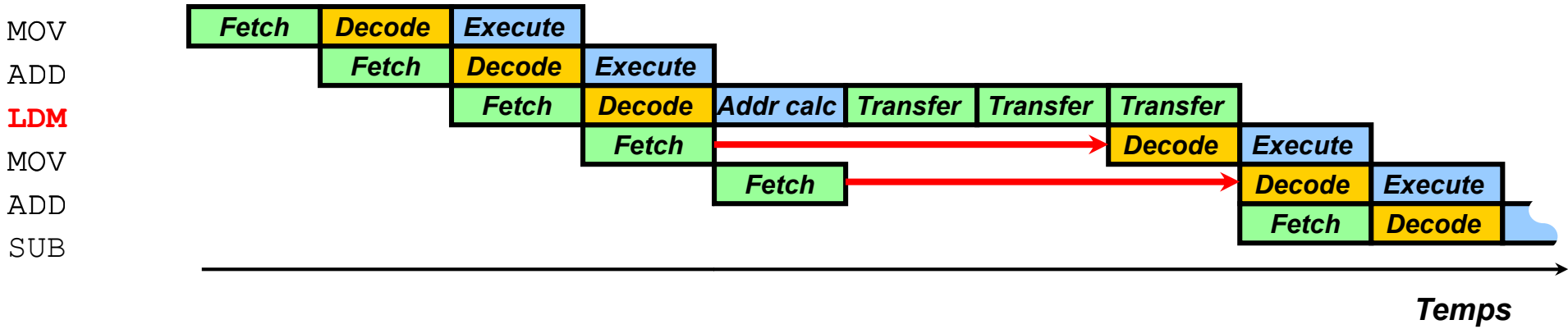


Solution: décaler les phases d'exécution des 2 avant dernières instructions

- Ralentissement du pipeline: le cycle de l'instruction en cours est arrêté jusqu'à ce que la donnée soit disponible (géré par le processeur)
- Apparition d'interruptions ("bulles") dans le pipeline
- Le pipeline implique un surcoût en complexité pour sa mise en œuvre

Exécution pipeline

- **Ralentissement du pipeline: cas des instructions d'accès mémoire multiples**



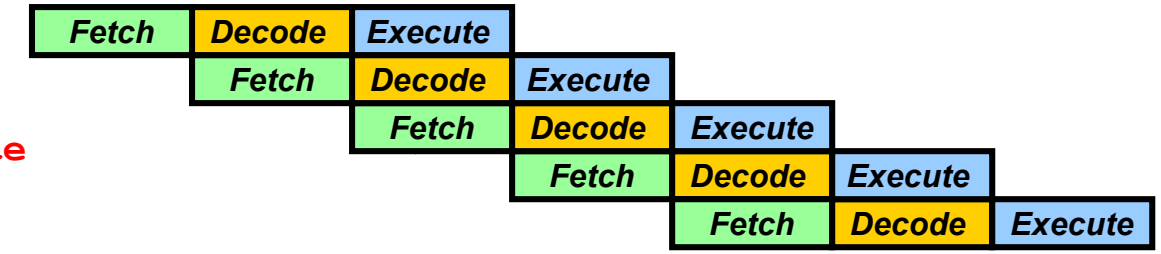
Autres cas de ralentissement du pipeline

- **L'exécution est ralentie par la durée de la phase de transfert**

Exécution pipeline

- **Ralentissement du pipeline: cas des instructions de branchement (1)**

```
MOV
ADD
BNE suite
MOV
ADD
…
Suite: MUL
```



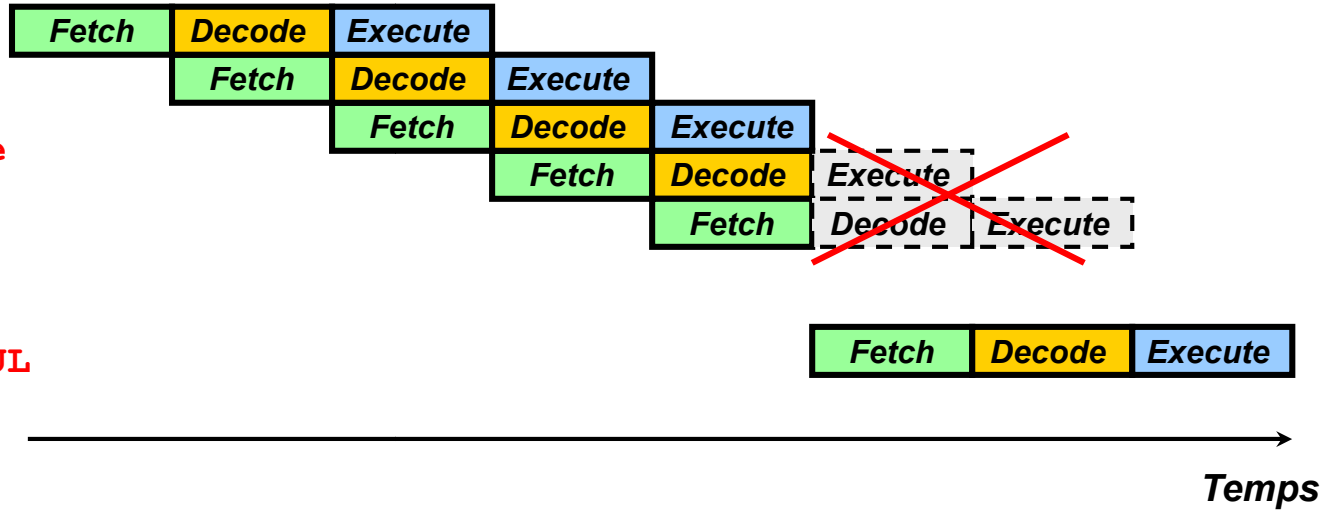
Temps

  - **Si la condition de branchement (BNE) est fausse, on poursuit la séquence d'instruction (MOV, ADD). Il n'y a pas d'interruption du pipeline.**

Exécution pipeline

- **Ralentissement du pipeline: cas des instructions de branchement (2)**

```
MOV
ADD
BNE suite
MOV
ADD
…
Suite: MUL
```



- **Si la condition de branchement (BNE) est vraie, on annule l'exécution des deux instructions suivantes, on poursuit à l'étiquette `Suite` (réinitialisation du pipeline).**
- **C'est la tâche du programmeur d'écrire du code assembleur qui utilise efficacement le pipeline pour optimiser l'exécution d'un programme**
- **Par exemple en utilisant l'exécution conditionnelle au lieu d'instructions de branchement (quand c'est possible), ou en ré-agençant les instructions pour éviter les ralentissement dus aux données non disponibles.**