# 17. Réalisation d’un processeur

## Principes de fonctionnement des ordinateurs

Jonas Lätt

Centre Universitaire d’Informatique

# Contenu du cours

## Partie I: Introduction

## Partie II: Codage de l'information

## Partie III: Circuits logiques

## Partie IV: Architecture des ordinateurs

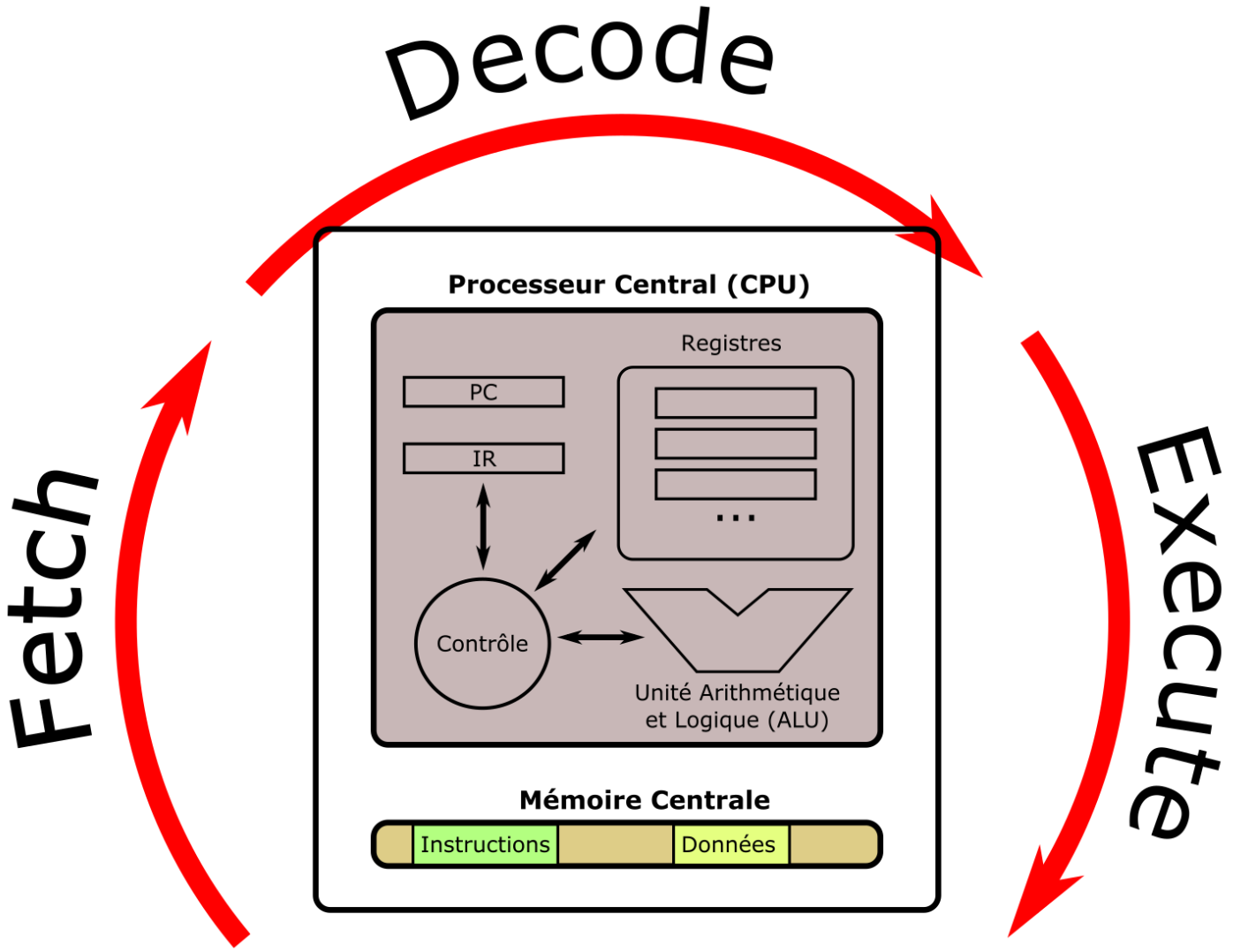
1. Introduction
2. Histoire de l'informatique
3. Information digitale et codage de l'information
4. Codage des nombres entiers naturels
5. Codage des nombres entiers relatifs
6. Codage des nombres réels
7. Codage de contenu média
8. Portes logiques
9. Circuits logiques combinatoires et algèbre de Boole
10. Réalisation d'un circuit combinatoire
11. Circuits combinatoires importants
12. Principes de logique séquentielle
13. Réalisation de la bascule DFF
14. Architecture de von Neumann
15. Réalisation des composants
16. Code machine et langage assembleur
17. **Architecture d'un processeur**
18. Performance et micro-architecture
19. Du processeur au système

# Déroulement universel d'un cycle RISC-V

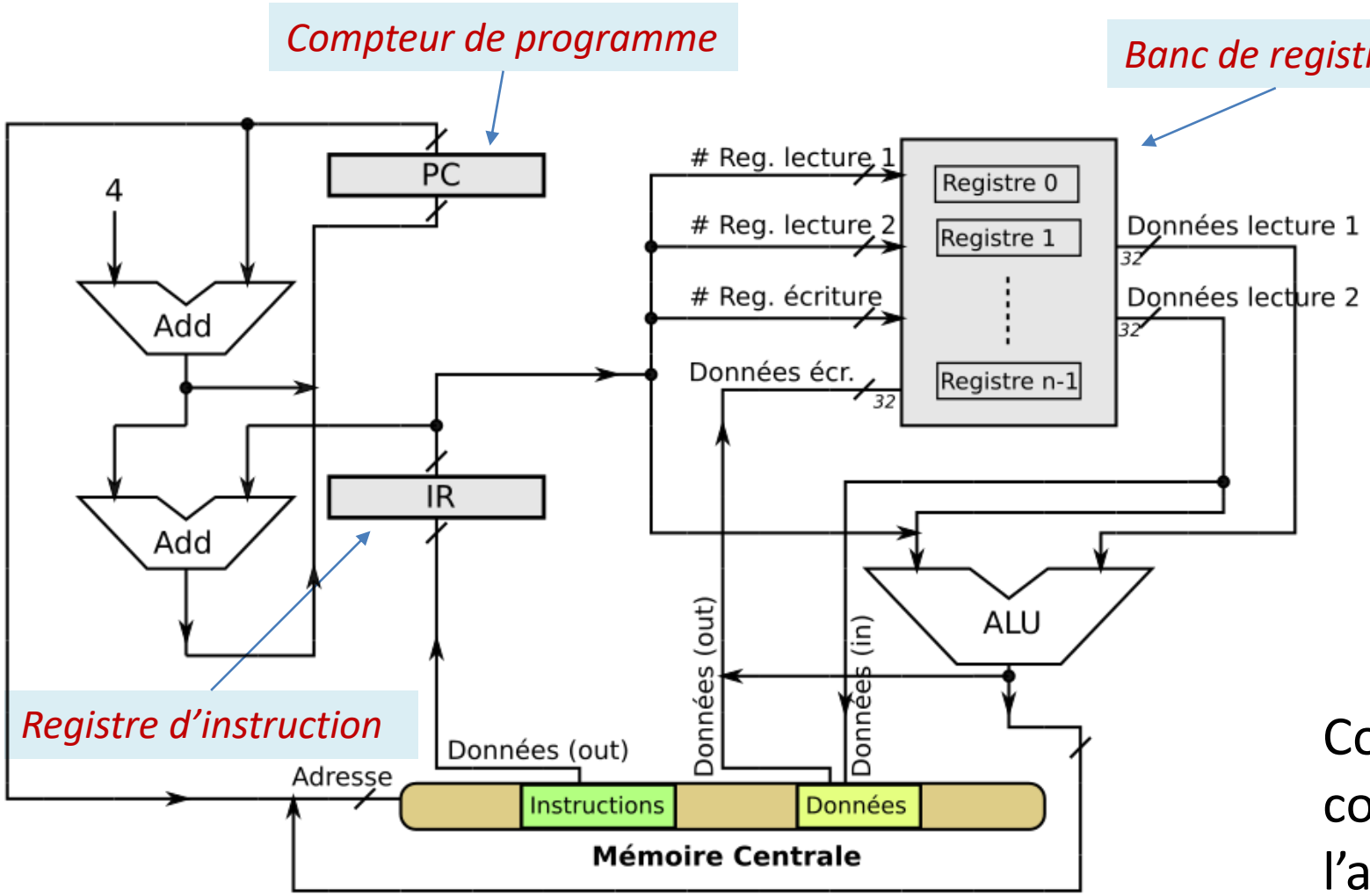
Simplicité et régularité de l'architecture RISC-V.

- **1ère partie**, identique pour toutes les instructions:
  1. Envoyer le PC à la mémoire et récupérer la prochaine instruction dans l'IR.
  2. Lecture d'un ou deux registres, identifiés par l'instruction.
- **2ème partie**. La plupart des instructions utilisent l'ALU:
  - Instruction Arithm/Logique: Pour effectuer l'opération demandée.
  - Instruction de transfert: Pour additionner un offset à l'adresse de base.
  - Branchement: Pour effectuer une comparaison.
- **3ème partie**. L'action entreprise dépend de la catégorie d'instruction:
  - Instruction Arithm/Logique: Ecriture dans le banc de registre.
  - Instruction de transfert: Accès mémoire.
  - Branchement: Manipulation du PC.

# Rappel: l'architecture de von Neumann



# Fonctionnement du processeur

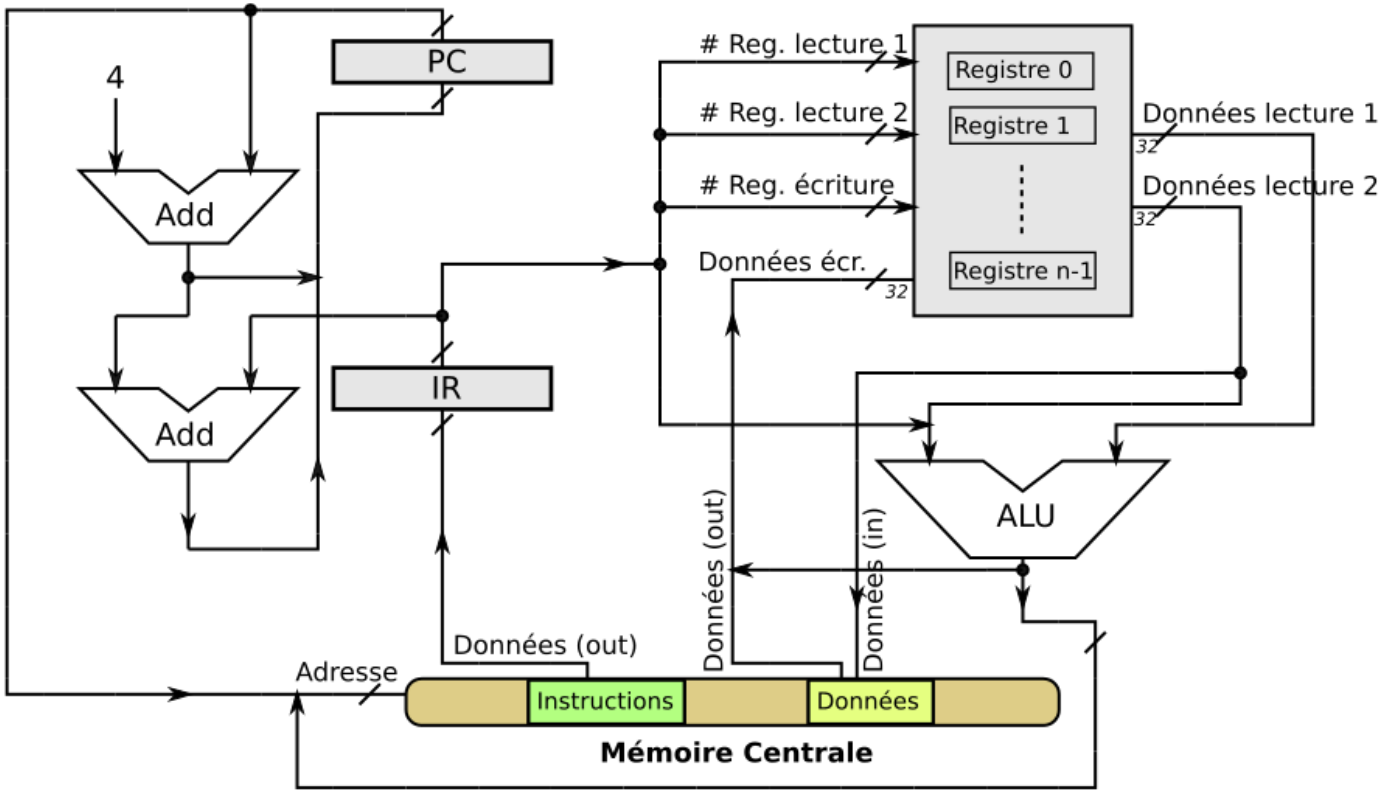



Connexions principales entre composants du CPU dans l'architecture RISC-V (32 bits)

# Etape fetch

Lors de l'étape *fetch*

- Le compteur de programme est communiqué à la mémoire.
- La prochaine instruction est déposée dans le registre d'instructions.



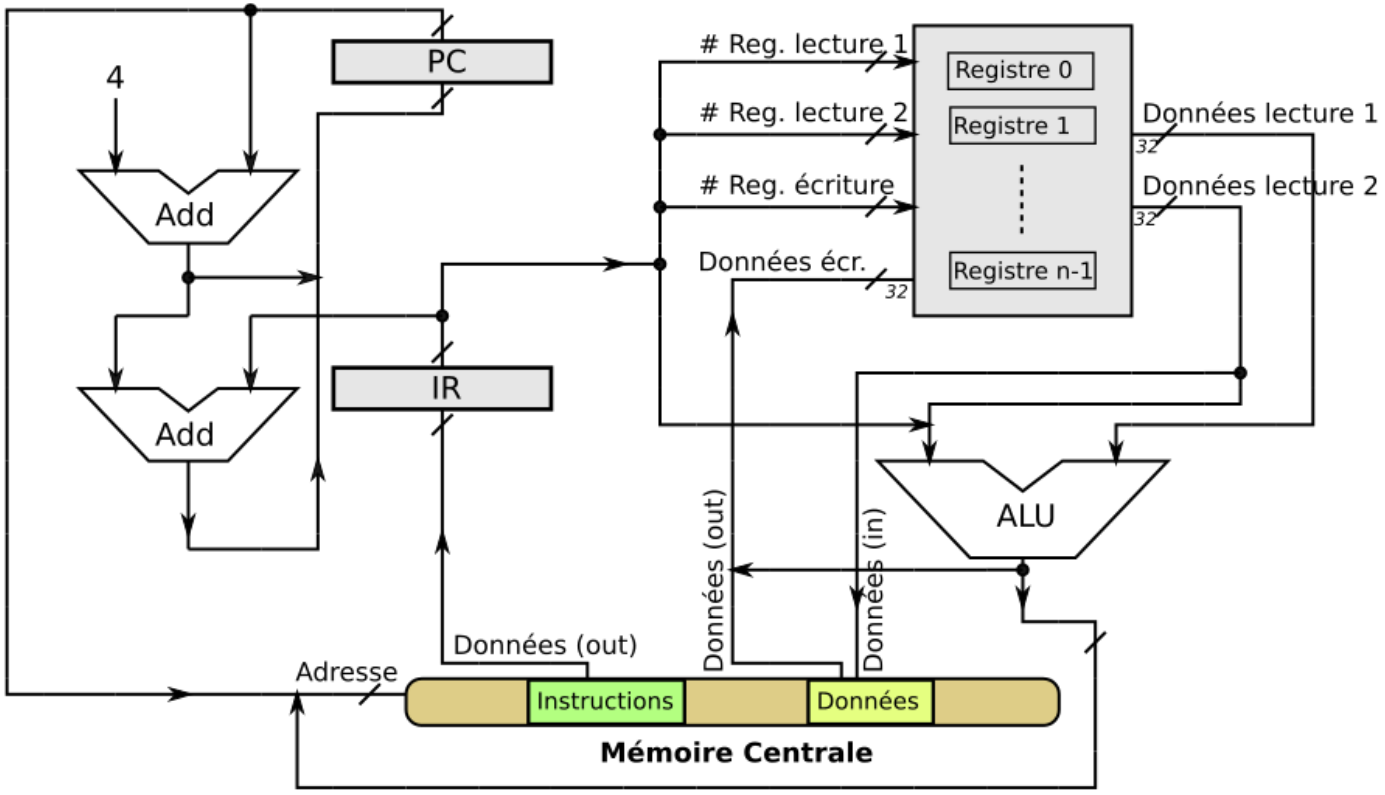

Le compteur de programme est alors mis à jour, pour passer à l'adresse de la prochaine instruction.

Dans le cas sans branchement (exécution séquentielle), l'adresse est simplement incrémentée de 4 (la taille d'une instruction).

# Etape decode

Lors de l'étape *decode*,

- L'unité de contrôle envoie des signaux de contrôle aux différents dispositifs.
- L'instruction est décomposée, et chaque groupe suit un parcours différent.



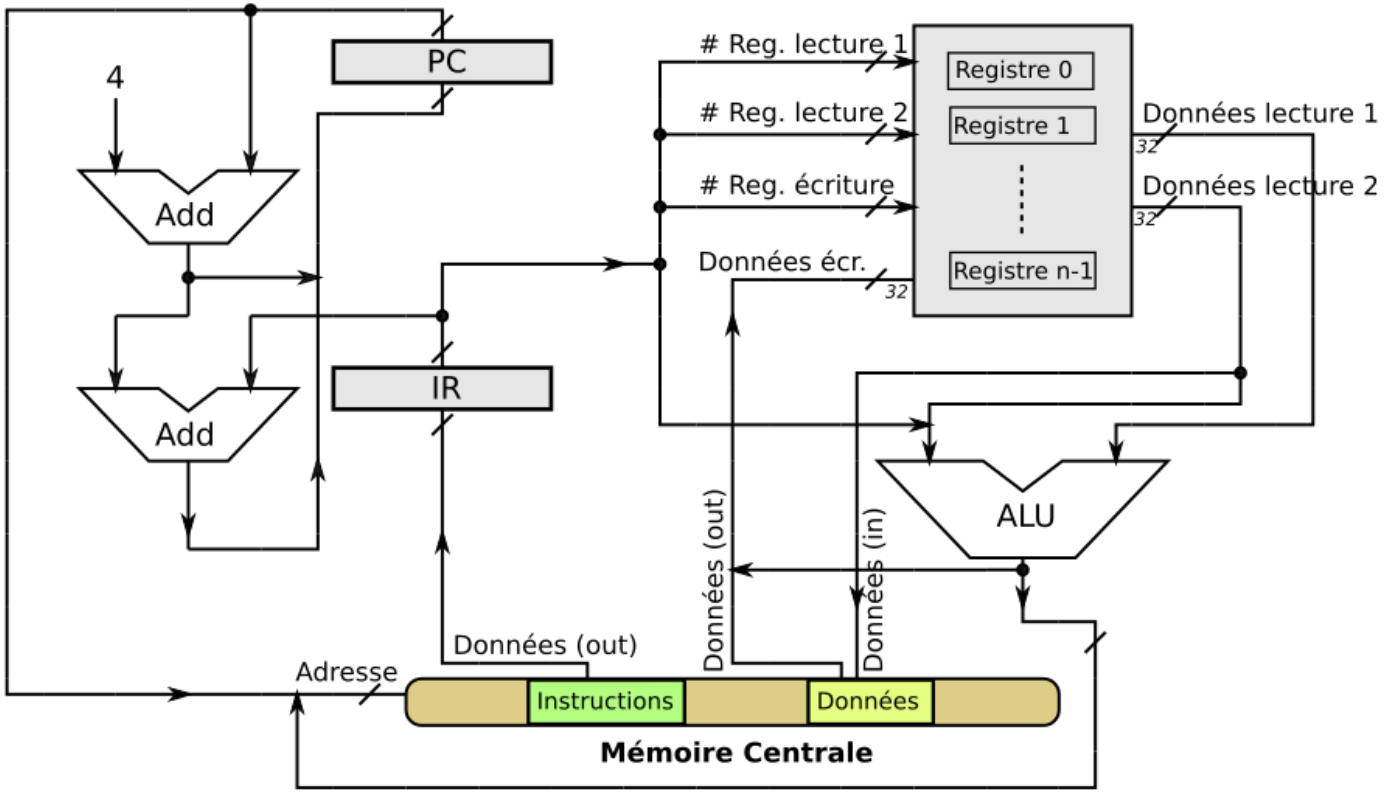

Pour un branchement, l'instruction contient l'adresse relative du branchement.

L'instruction peut contenir l'identifiant d'un ou deux registres en lecture, un registre en écriture, ou encore une valeur constante (instruction de type **I**).

# Etape execute (1): instruction au format R

```
add x1, x11, x12
```

Pour une opération arithmétique ou logique au format **R**, lors de l'étape *execute*

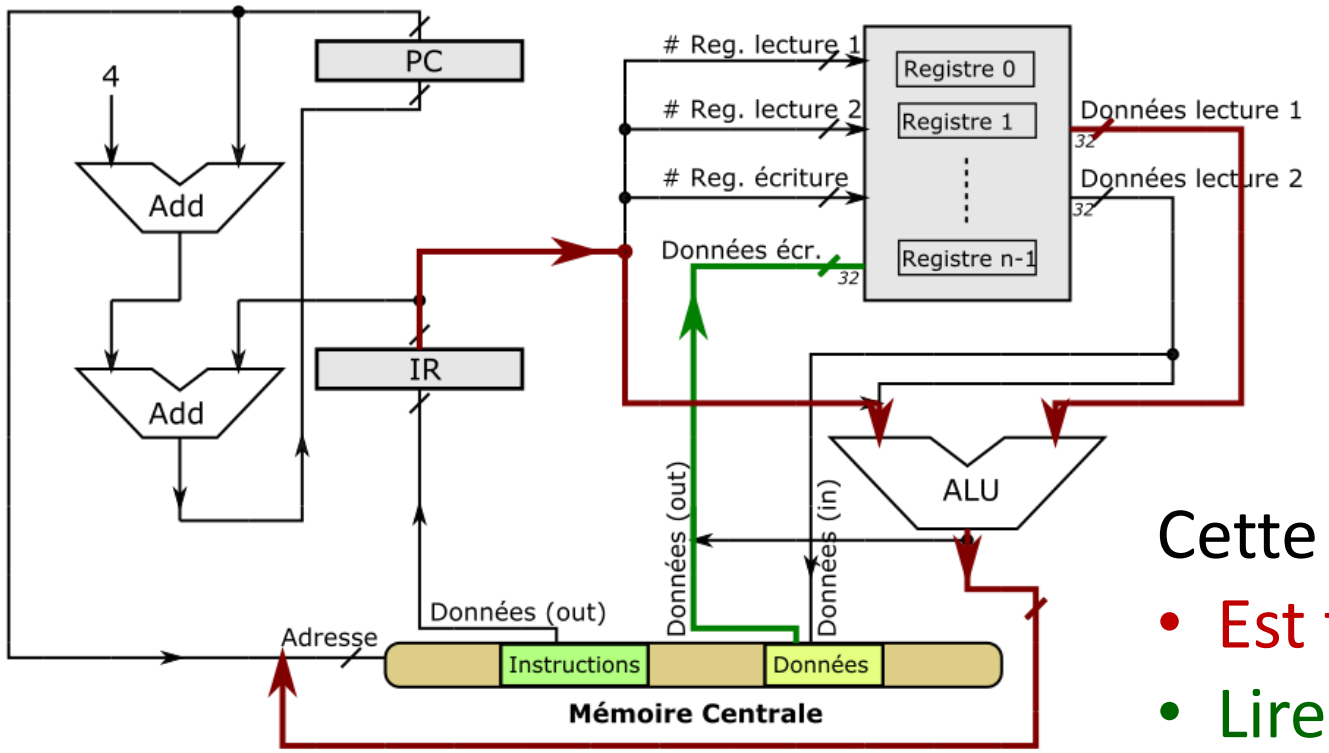



- Deux registres sont accédés en lecture
- L'opération est effectuée à l'aide de l'ALU
- Le résultat est sauvegardé dans un registre.

# Etape execute (2): accès à la mémoire

```
lw    x1, 8(x2)
```

Pour une opération de lecture d'un mot `lw` (format **I**) l'étape *execute*

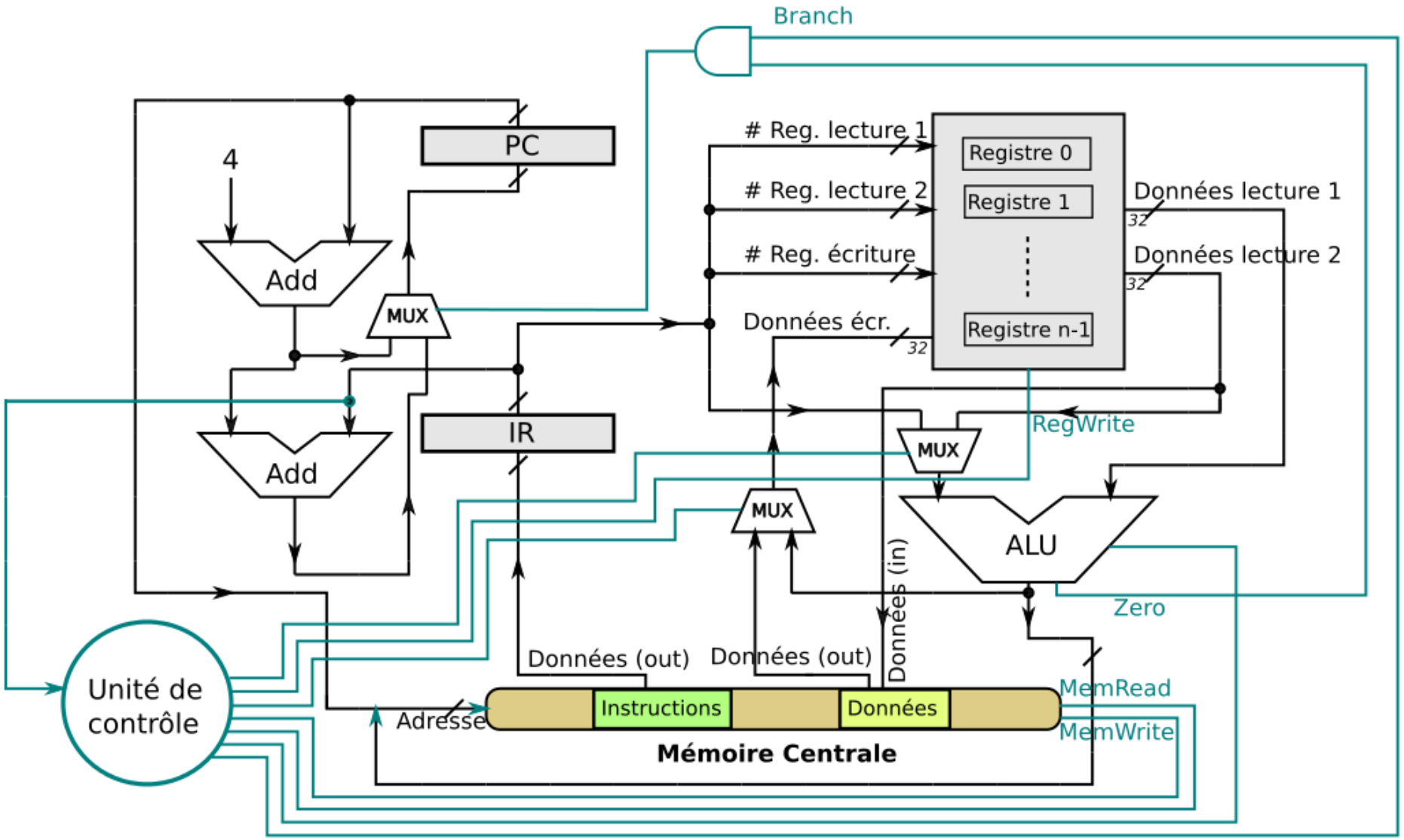



- Lit la valeur d'un registre (l'adresse de base) et récupère l'offset codé dans l'instruction.
- Calcule l'adresse de la donnée en mémoire à l'aide de l'ALU.

Cette adresse

- Est fournie à la mémoire, pour
- Lire la donnée et la déposer dans un registre.

# Architecture (presque) complète



# Architecture (presque) complète

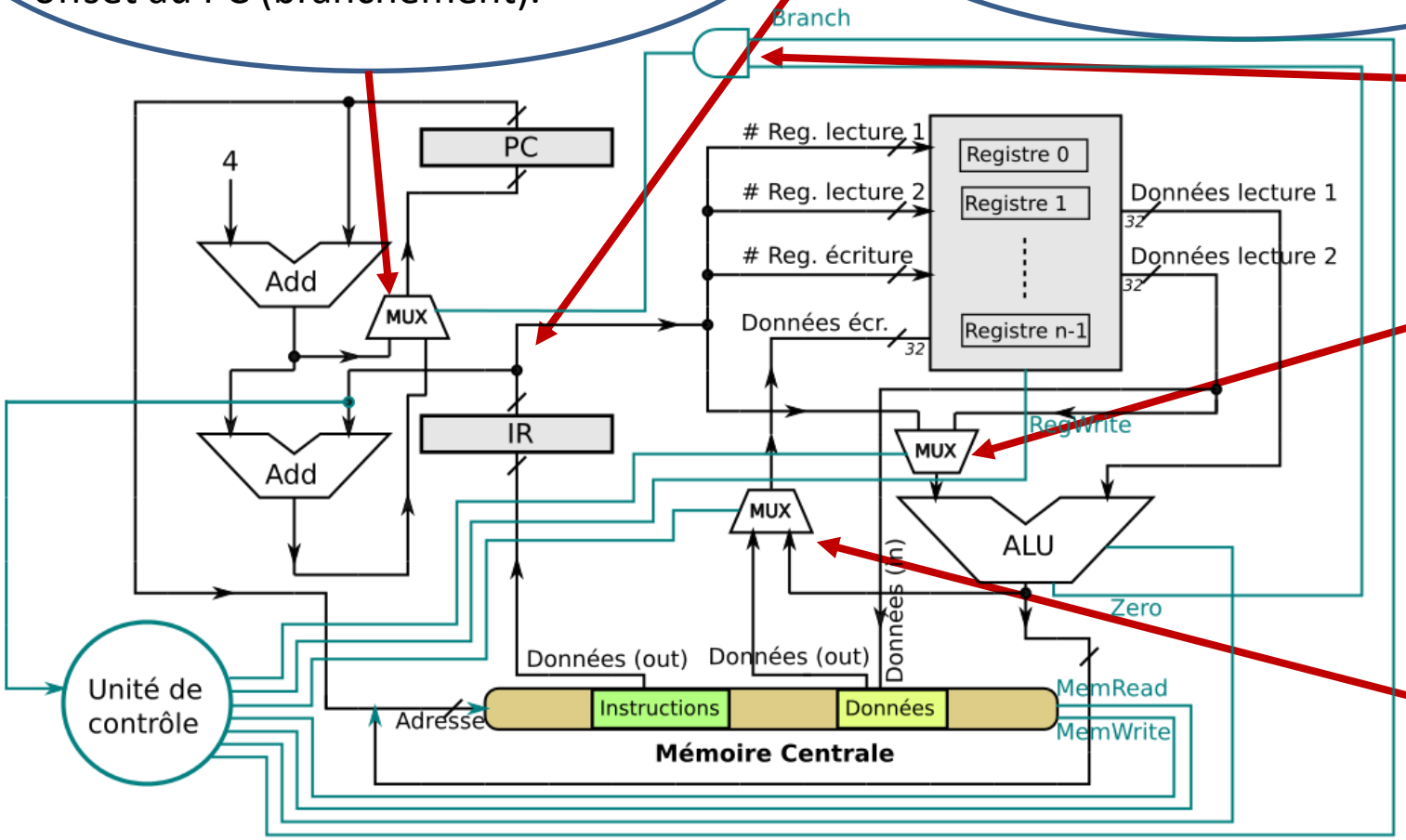
Ce multiplexeur sélectionne entre une incrémentation +4 du PC (flux d'instruction linéaire), ou l'ajout d'un offset au PC (branchement).

Par simplicité, on ne montre pas le circuit qui décompose l'instruction en ses différents groupes.

Test: L'instruction est un branchement ET le résultat de l'ALU vaut 0.

Sélection entre valeur d'un registre ou valeur constante de l'instruction.

Sélection entre données de la mémoire ou données en sortie de l'ALU.




Jonas Lätt

# Exemple: `add x1, x11, x12`

0000000 01100 01011 000 00001 0110011