*M1 - SEM*

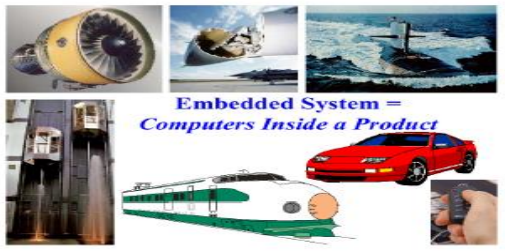
# Processeurs Embarqués

## Cours 3.3

# Conception de processeur : Unité de contrôle

**Année : 2022-2023**

*Pr R. BOUDOUR*



# Aspects performance

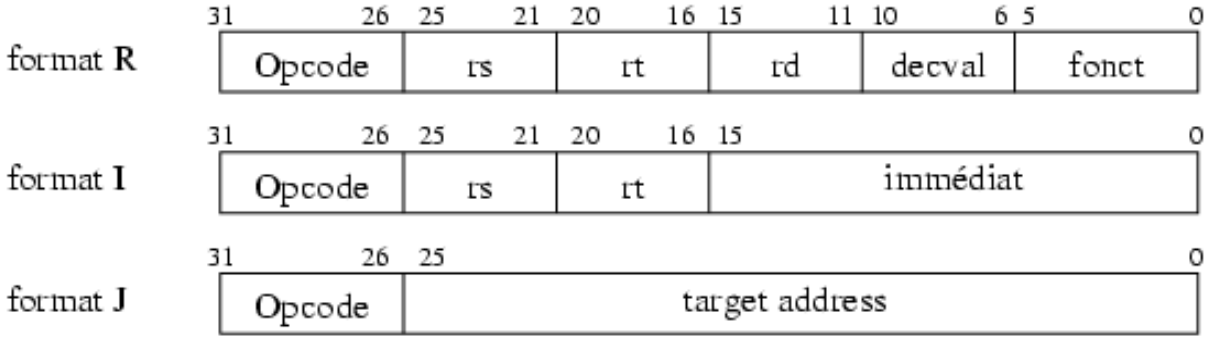


- Les performances d'une machine sont déterminées par :
  - le temps de cycle de l'horloge
  - le nombre de cycles par instruction
- La conception du chemin de données et du contrôle détermine :
  - le temps de cycle de l'horloge
  - le nombre de cycles par instruction

# Les formats des instructions MIPS



- Toutes les instructions MIPS ont 32 bits de long




- Les différents champs sont :
  - **opcode : opération de l'instruction**
  - **rs, rt, rd : registres sources et destination**
  - **dec : le nombre de décalages**
  - **fonct : les variantes des opérations**
  - **immédiat : valeur immédiate ou de déplacement d'adresse**
  - **target address : adresse cible pour les jump**

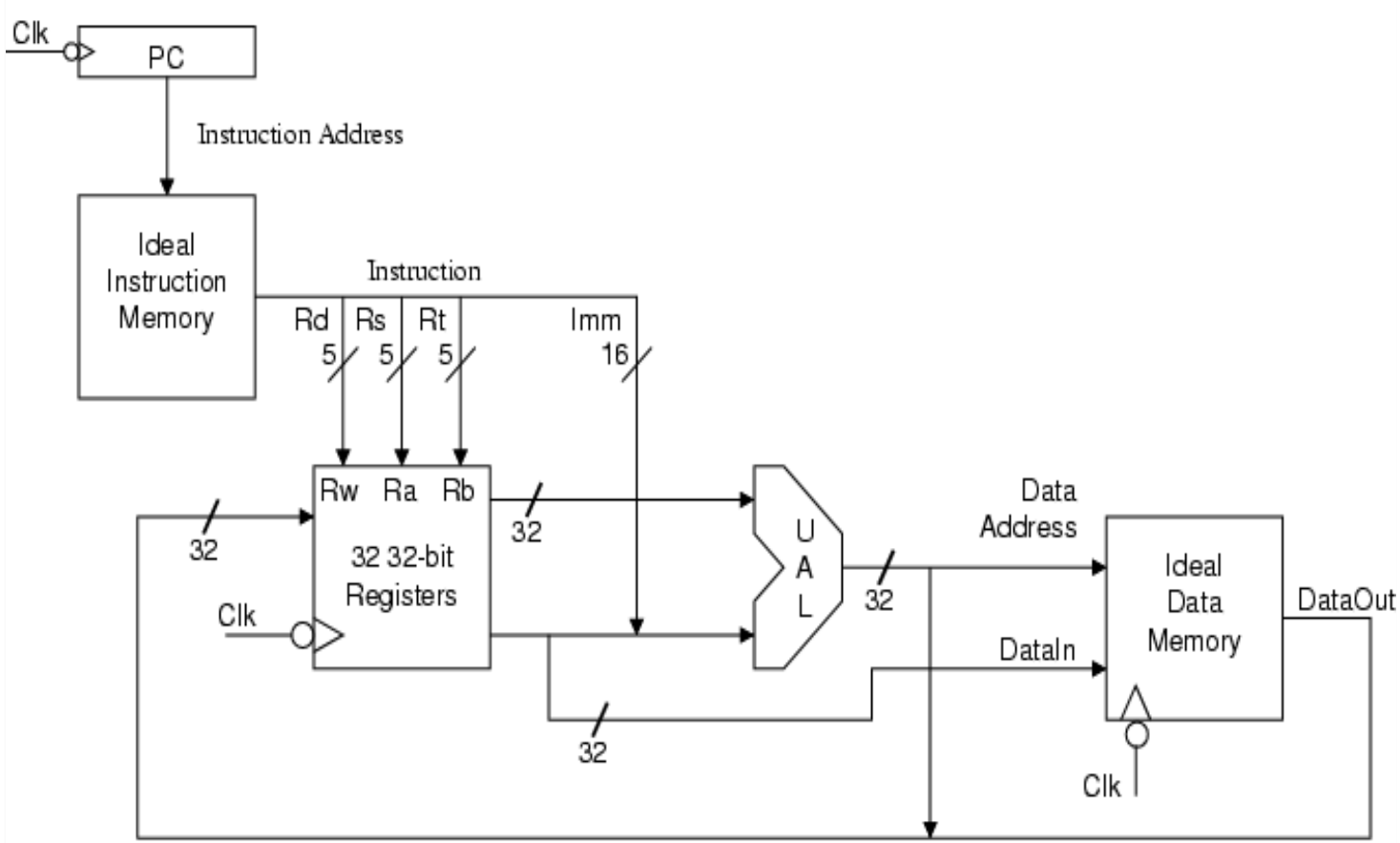
# Sous-ensemble de MIPS mis en œuvre



- **Addition et Soustraction :**
  - `add rd, rs, rt`
  - `sub rd, rs, rt`
- **Ou immédiat :**
  - `or rt, rs, imm16`
- **Chargement et Rangement :**
  - `lw rt, rs, imm16`
  - `sw rt, rs, imm16`
- **Branchement :**
  - `beq rs, rt, imm16`
- **Saut :**
  - `j target`

# Vue abstraite de l'implémentation





# Etapes de conception d'un processeur



- Architecture du jeu d'instruction ⇒ Langage "Transfert de registres" (RTL)
- Langage "Transfert de registres" ⇒
  - composants du chemin de données
  - interconnection des composants
- composants du chemin de données ⇒ Signaux de contrôle
- Signaux de contrôle ⇒ Logique de contrôle

# RTL : Instruction ADD

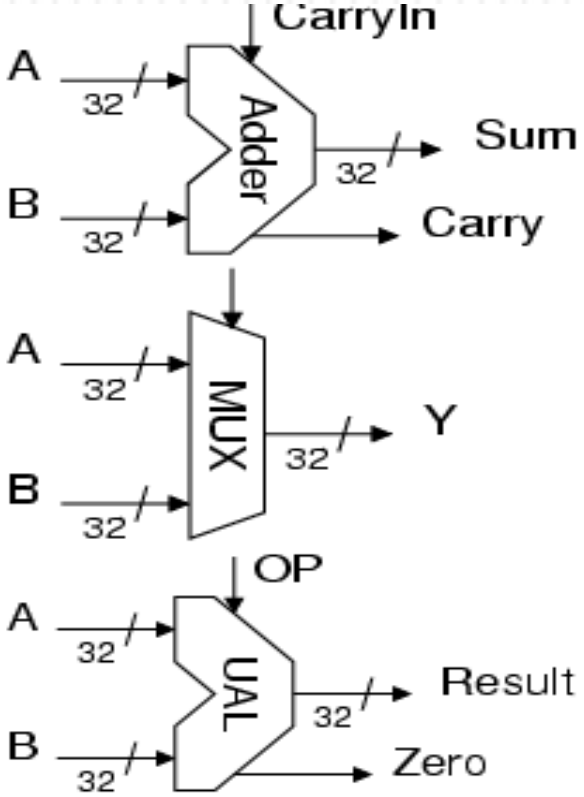


- **add rd, rs, rt**
  - `mem[PC]` — Extraire l'instruction de la mémoire
  - `R[rd] ← R[rs] + R[rt]` — Opération d'addition
  - `PC ← PC + 4` — Calcul de l'adresse de la prochaine instruction

# Eléments combinatoires



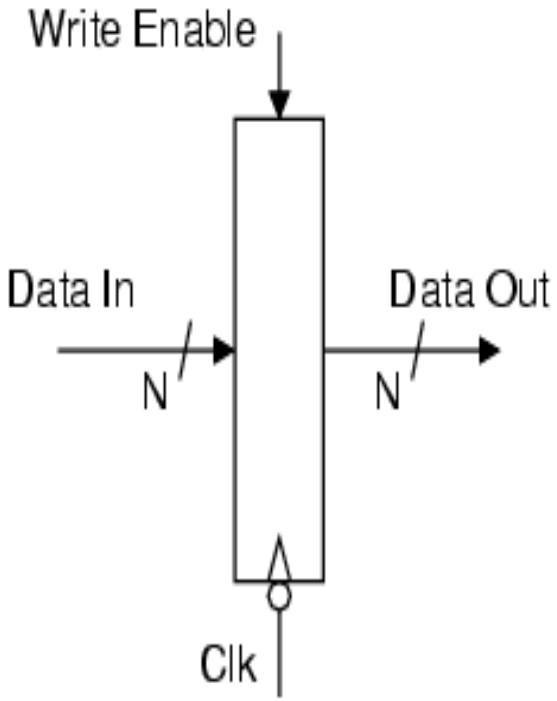
- **Adder**
- **MUX**
- **UAL**



# Elément de mémoire : Registre



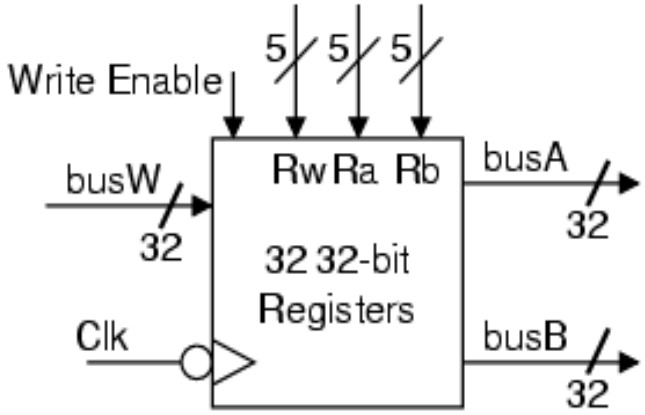
- Registre
  - Similaire aux bascules D mais
    - N-bits en entrée et en sortie
    - une entrée “Write enable”
  - Write enable :
    - 0 : **DataOut** ne change pas
    - 1 : **DataOut** devient **DataIn**



# Elément de mémoire : banc de registres



- Banc de registres = 32 Registres
  - 2 bus de sortie de 32 bits ; **busA** et **busB**
  - un bus d'entrée de 32 bits ; **busW**

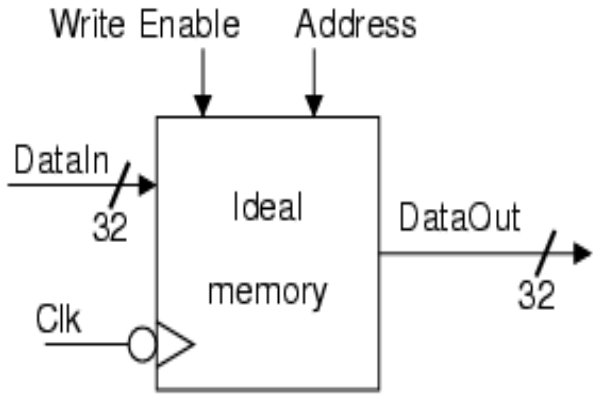



- Registre sélectionné par :
  - **RA** selectionne le registre pour le **busA**
  - **RB** selectionne le registre pour le **busB**
  - **RW** selectionne le registre à écrire via le **busW** quant **WriteEnable** est à 1

# Elément de mémoire : Mémoire idéale



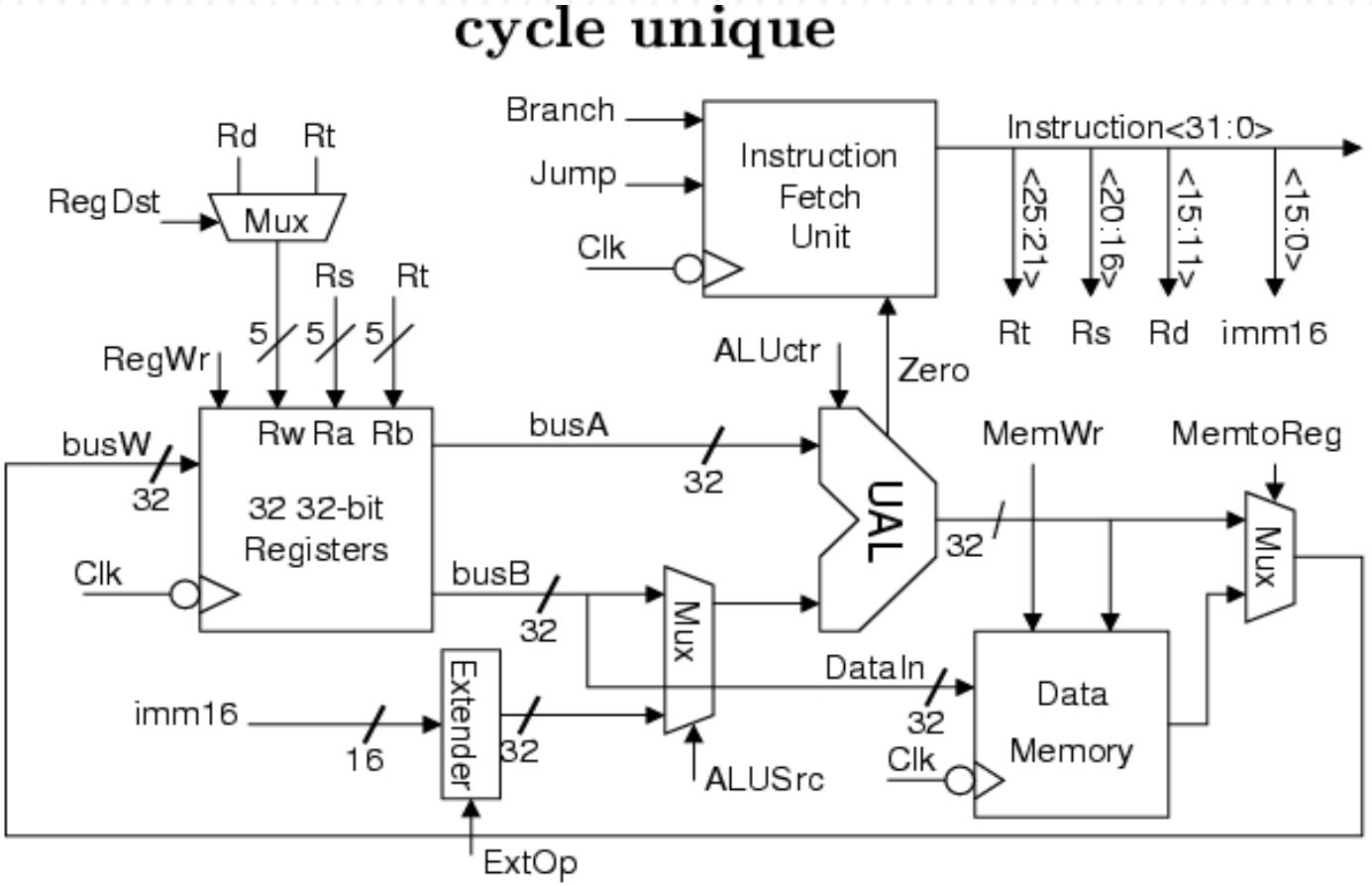
- Mémoire (idéale)
  - Un bus de sortie de 32 bits ; **DataOut**
  - Un bus d'entrée de 32 bits ; **DataIn**




- Un mot mémoire est sélectionné par :
  - **Address** sélectionne le mot pour le bus **DataOut**
  - Si **WriteEnable** $= 1$ ; **Address** sélectionne le mot mémoire à écrire via le bus **DataIn**

# Le tout : un chemin de données à cycle unique





# Que faut-il faire ?

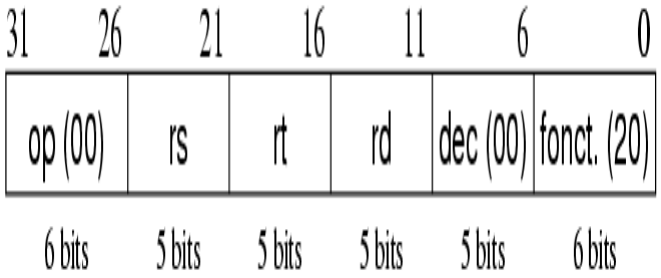


A partir des entrées, il faut générer :

- Un signal d'écriture pour chaque élément d'état
- Un signal de contrôle des multiplexeurs
- Un signal de contrôle pour l'UAL

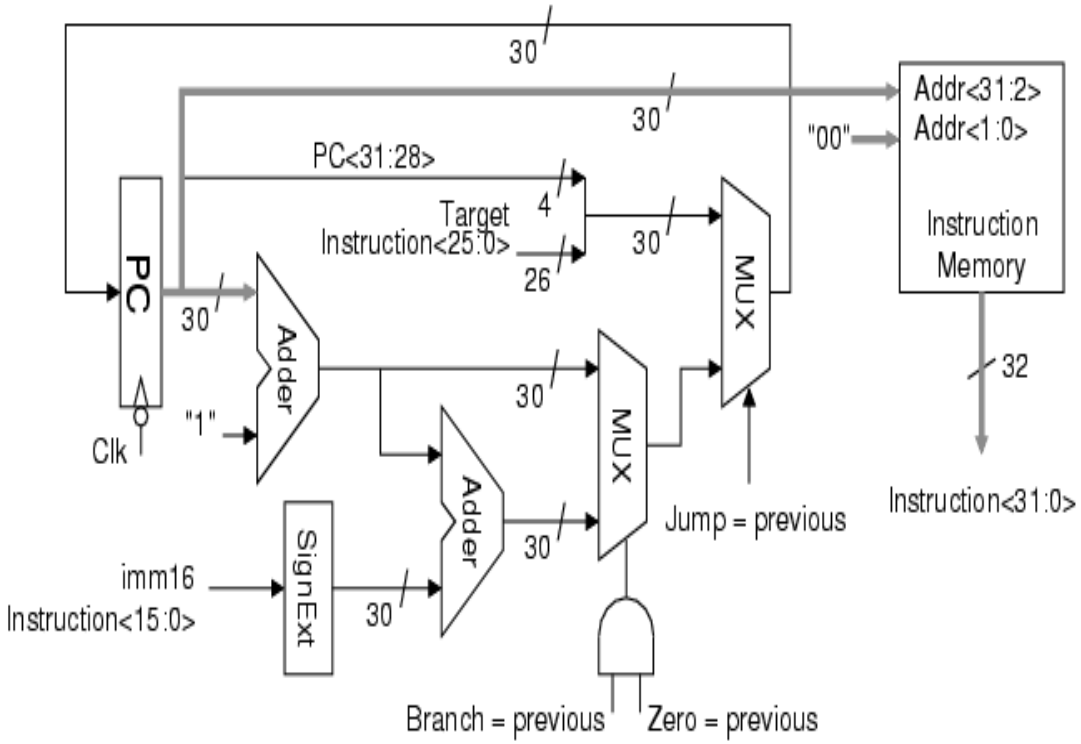
# Unité fetch au début de ADD et SUB





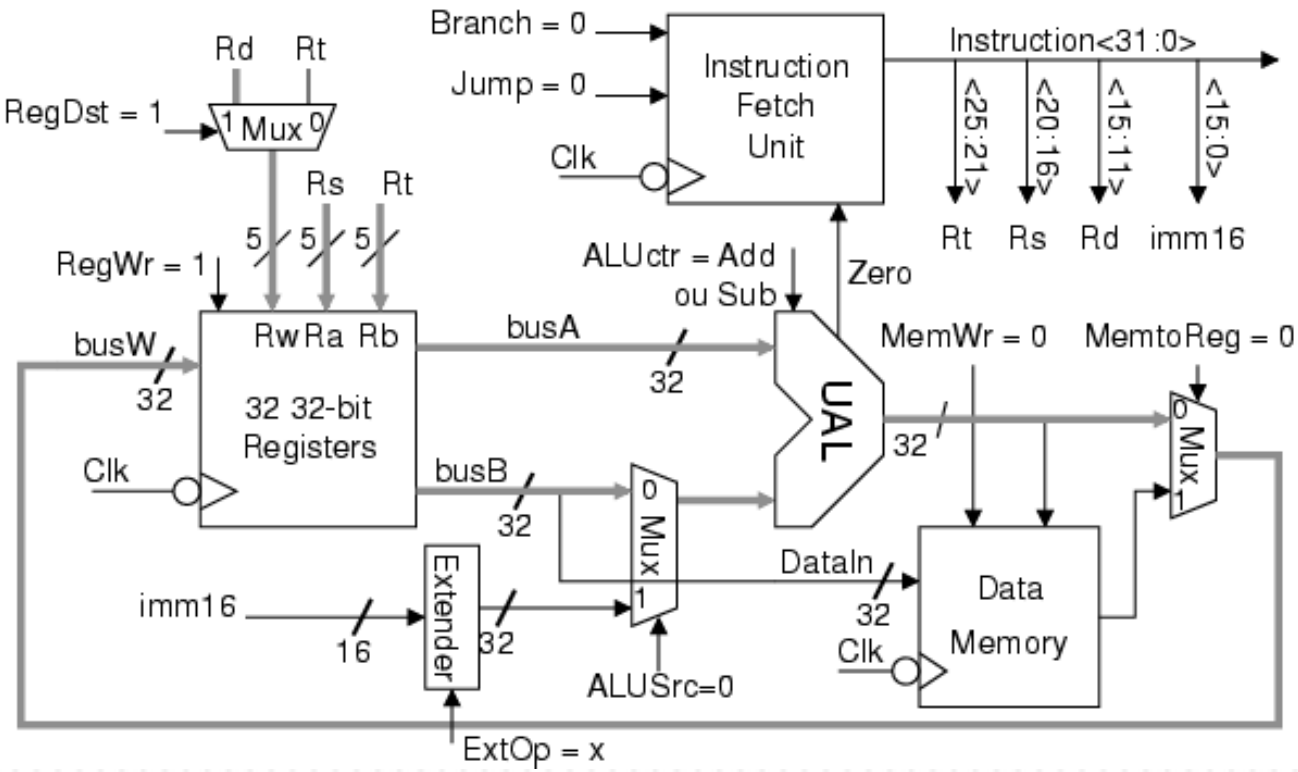

- **add rd, rs, rt**
  - mem[PC] — Extraire l'instruction
  - R[rd] ← R[rs] + R[rt] — Opération d'addition
  - PC ← PC + 4 — Calcul de l'adresse de la prochaine instruction

(identique pour toutes les instructions)



# Chemin de données pendant ADD ou SUB

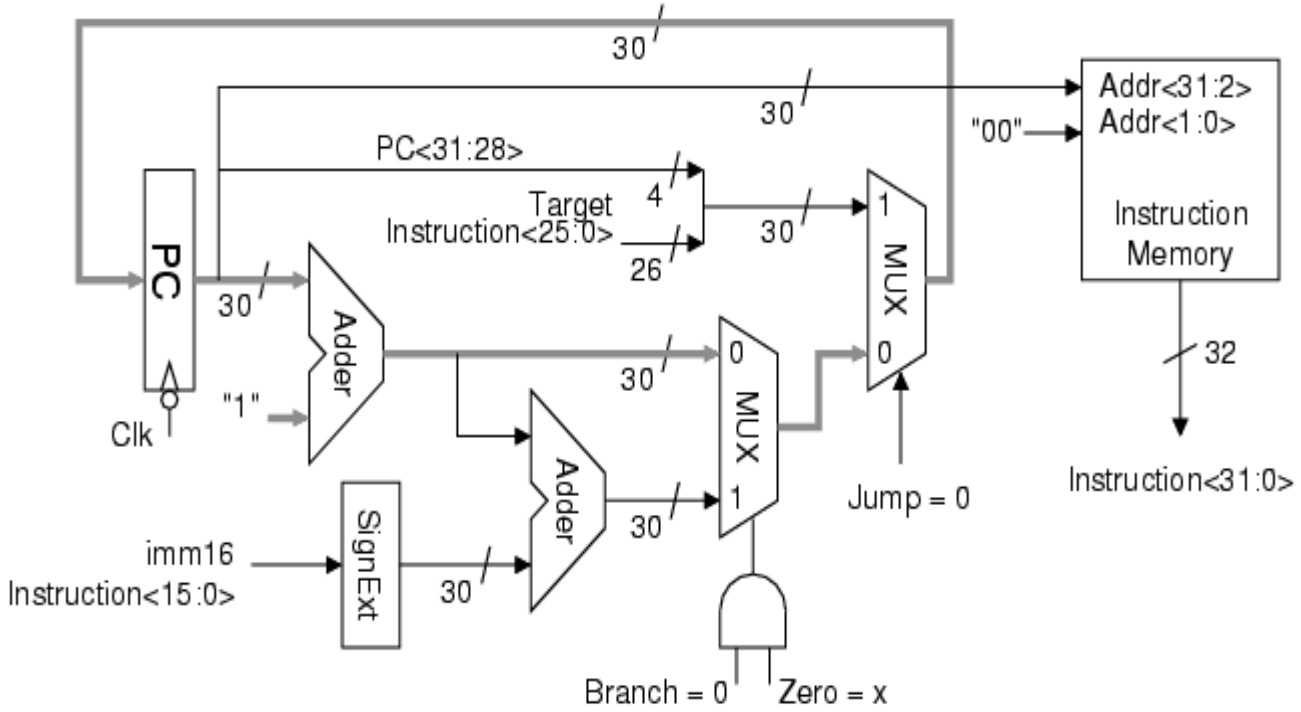




# Chemin de données à la fin de ADD ou SUB

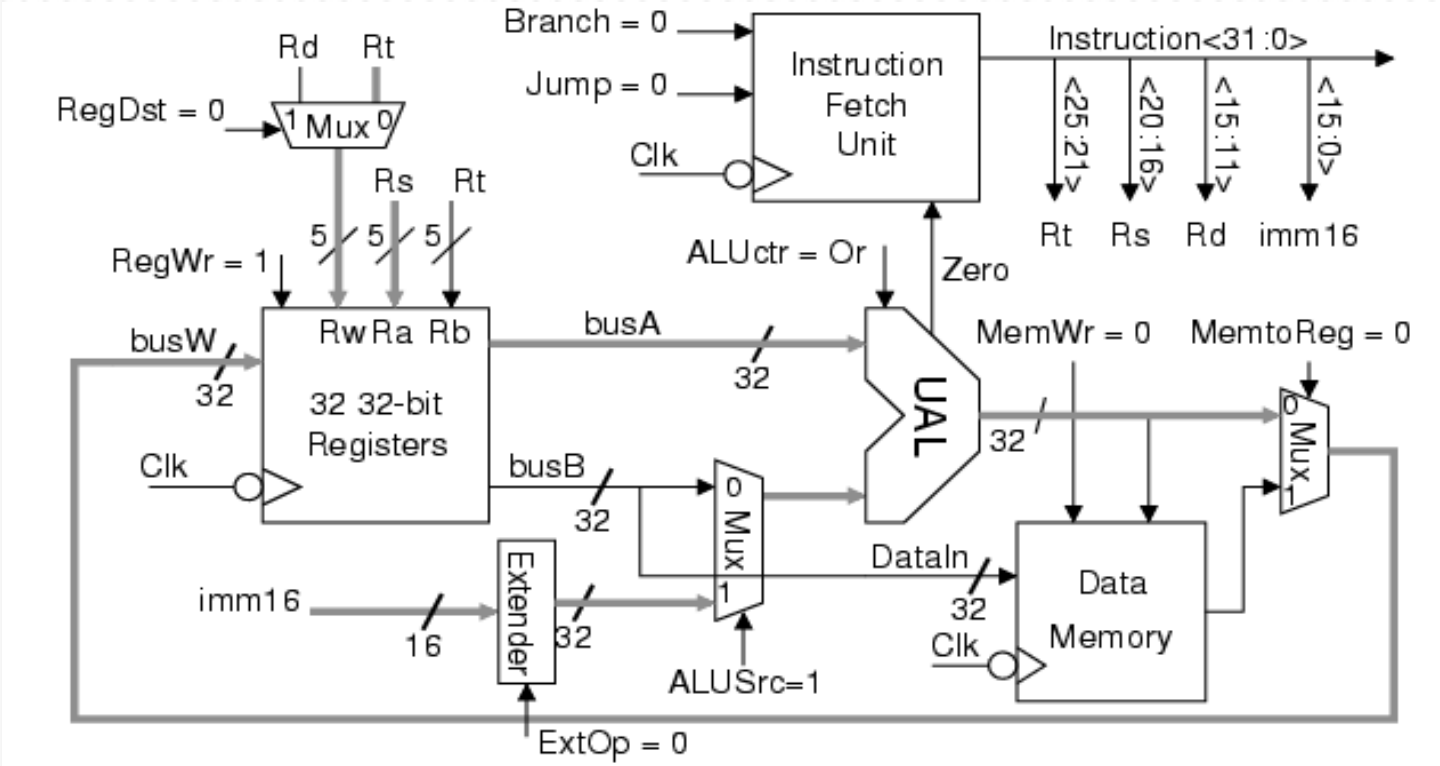


(identique pour toutes les instructions sauf branch et jump)



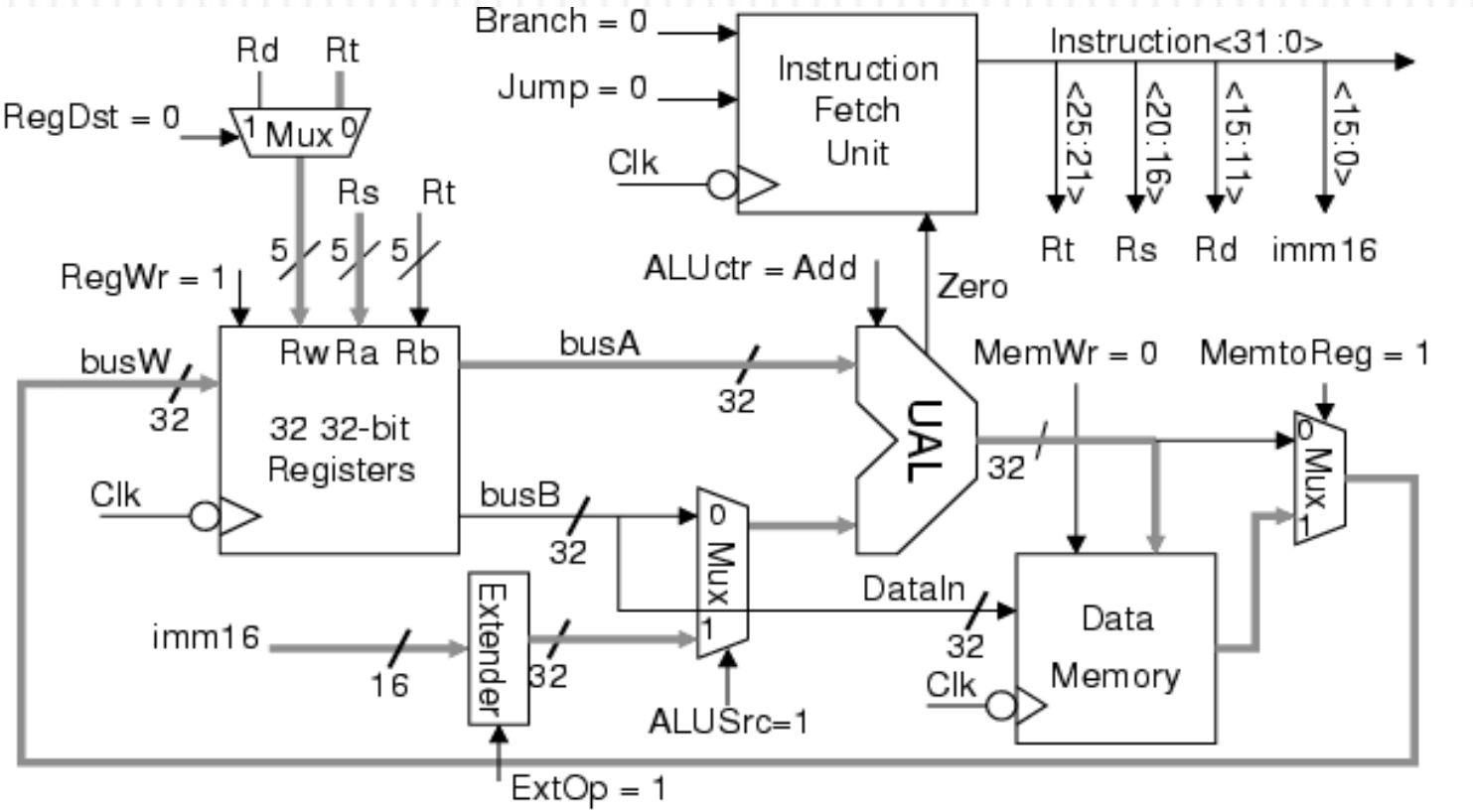
# Chemin de données pendant ORI (immédiat)





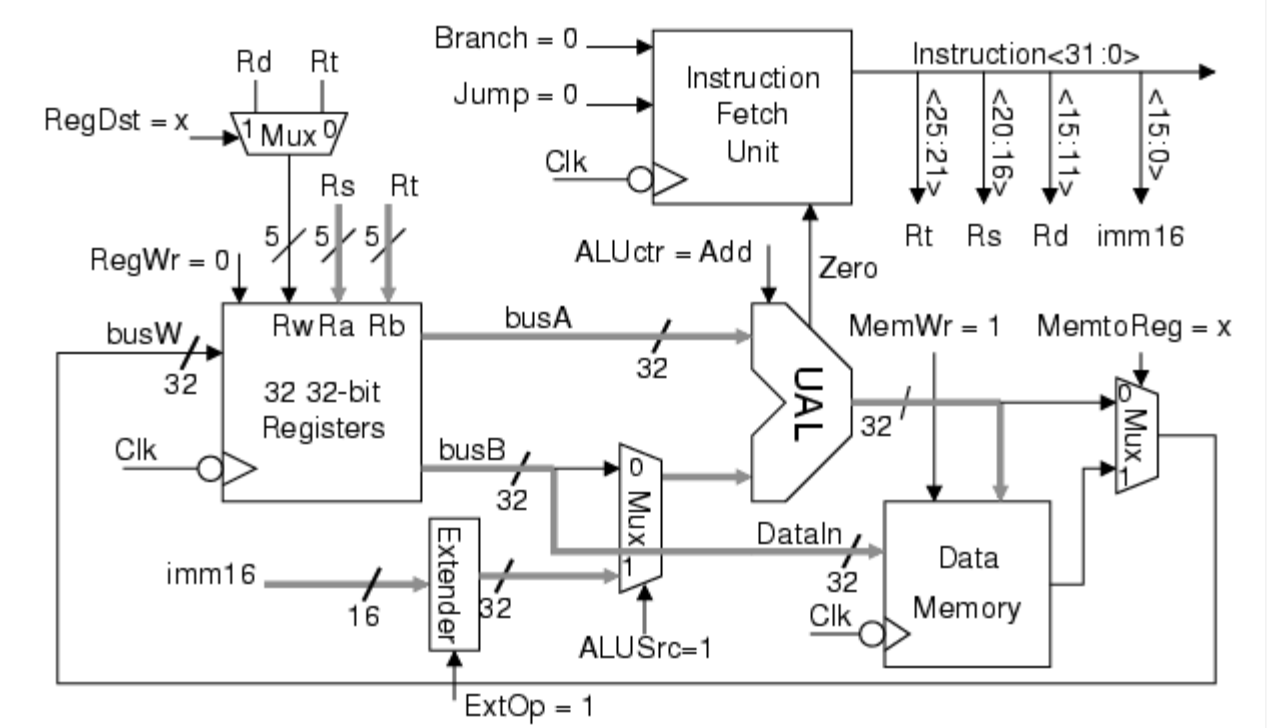
# Chemin de données pendant Load





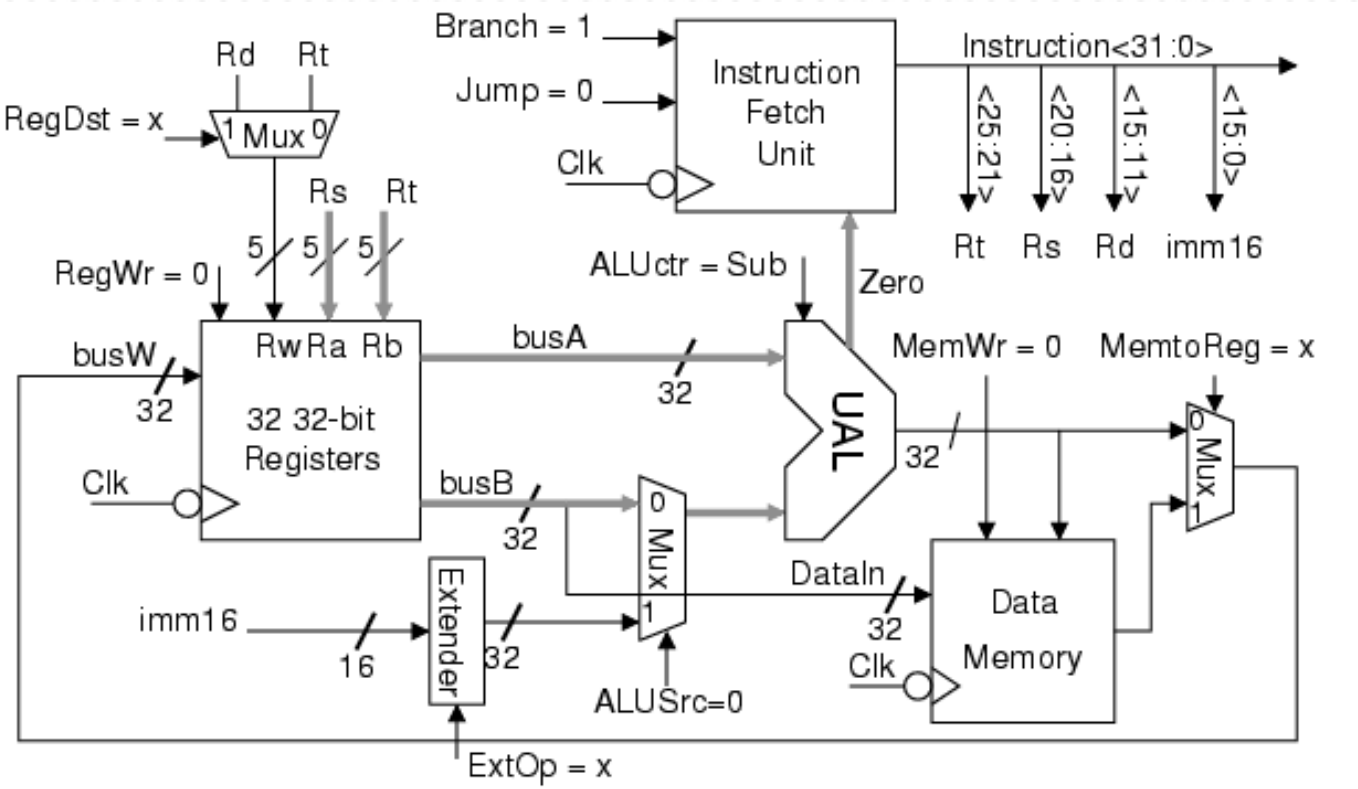
# Chemin de données pendant Store





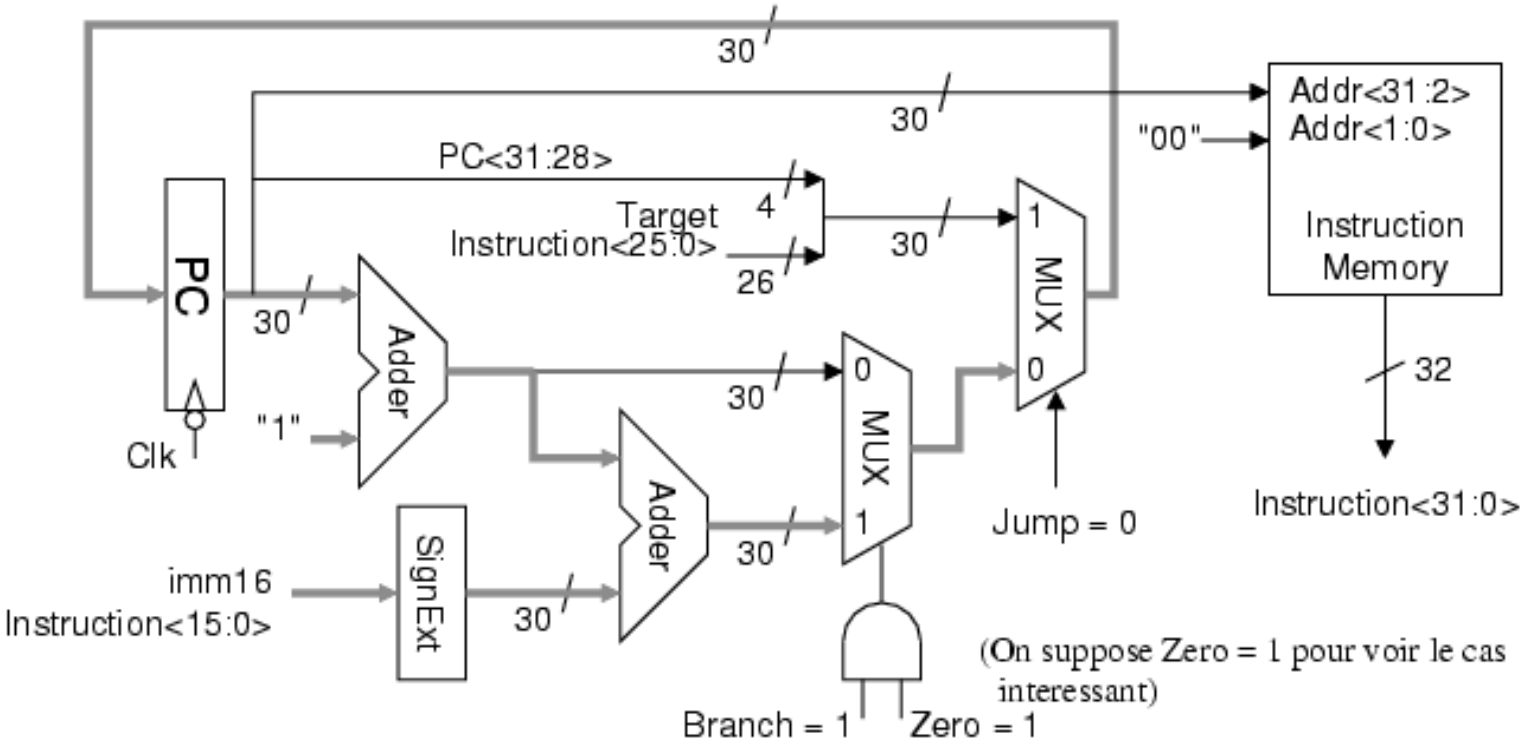
# Chemin de données pendant branch





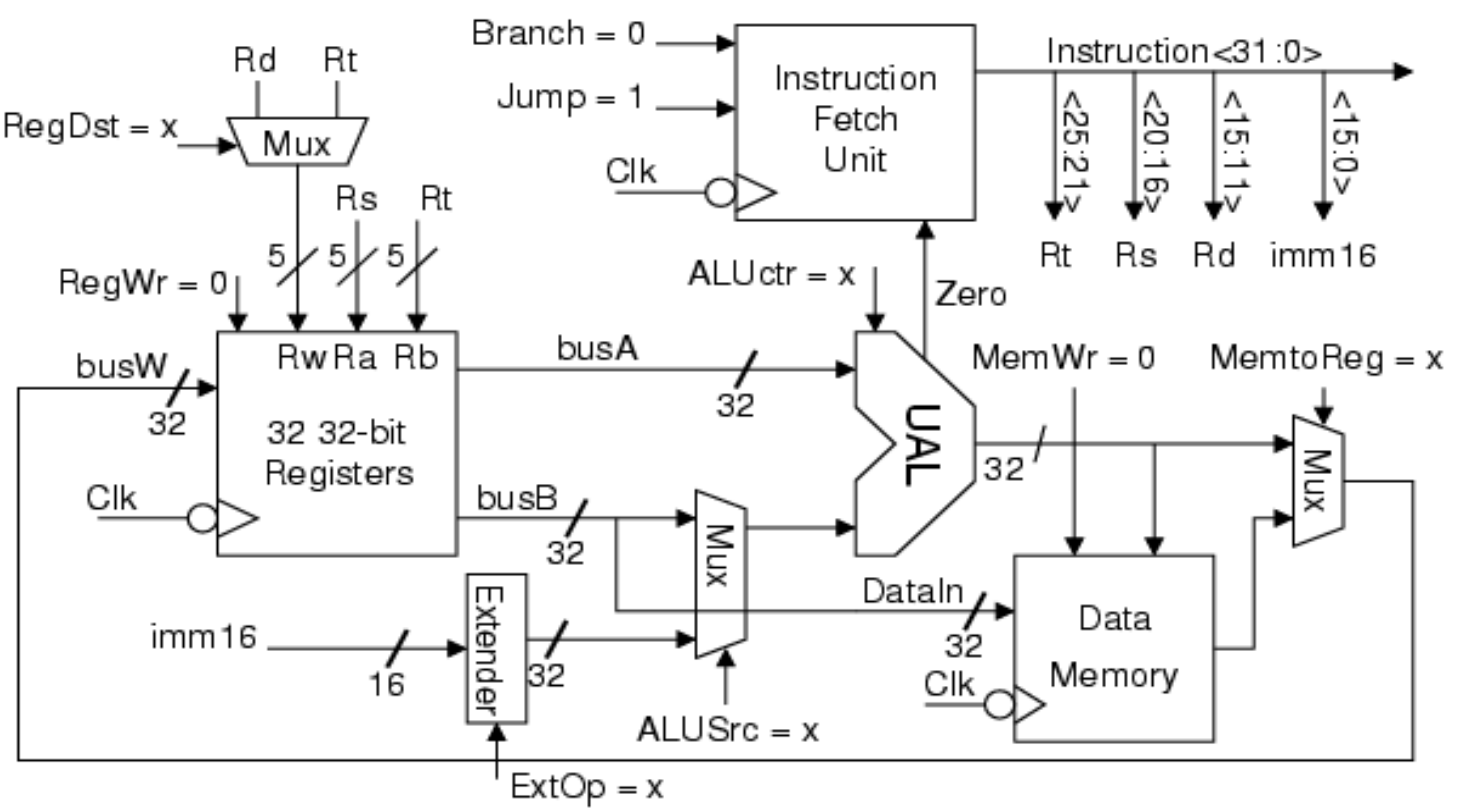
# Unité Fetch à la fin de Branch





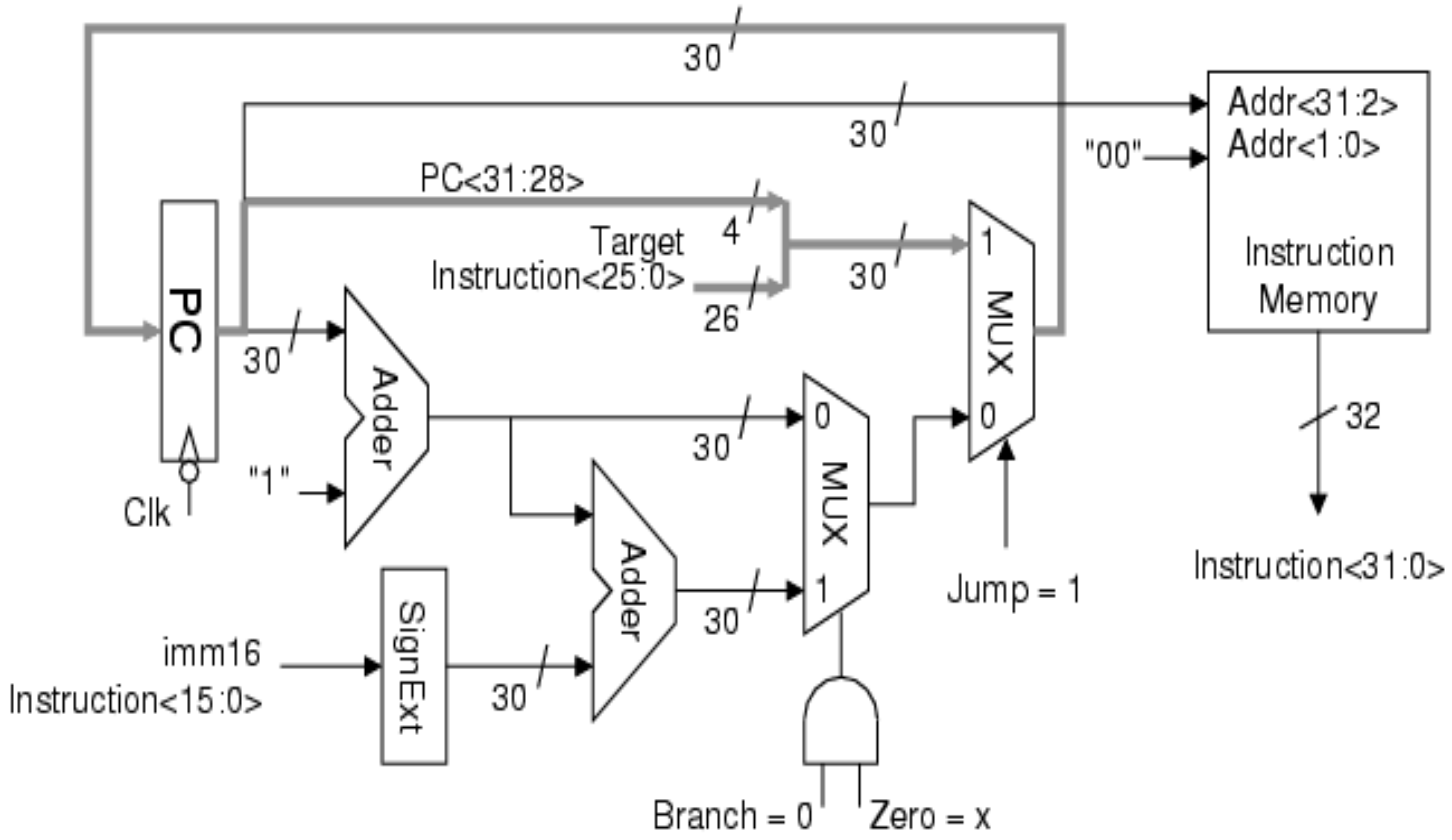
# Chemin de données pendant Jump





# Unité Fetch à la fin de Jump





# Résumé des signaux de contrôle

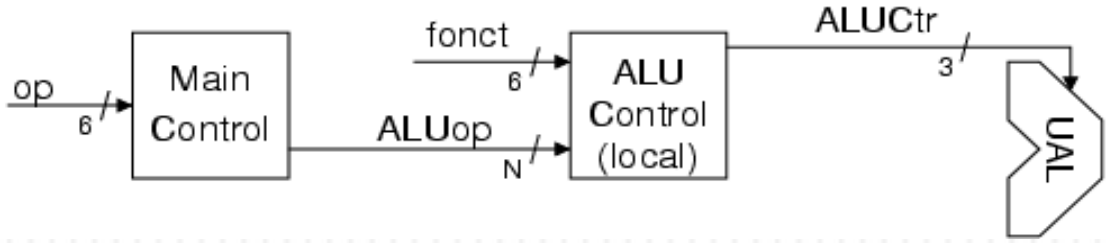


| fonct | 100000 | 100010 | Don't Care | | | | |
|---|---|---|---|---|---|---|---|
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 000010 |
| | add | sub | ori | lw | sw | beq | jump |
| RegDst | 1 | 1 | 0 | 0 | x | x | x |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | x |
| MemtoReg | 0 | 0 | 0 | 1 | x | x | x |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Branch | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Jump | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ExtOp | x | x | 0 | 1 | 1 | x | x |
| ALUctr | Add | Sub | Or | Add | Add | Sub | xxx |

# Concept de décodage local

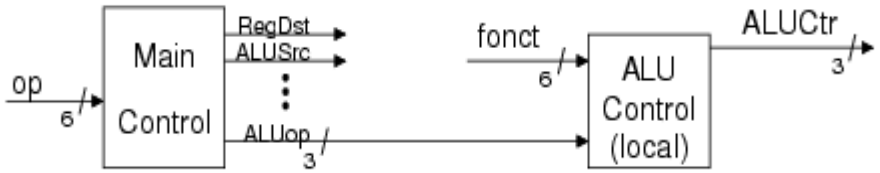


| op | 000000 | 001101 | 100011 | 101011 | 000100 | 000010 |
|---|---|---|---|---|---|---|
| | R-Type | ori | lw | sw | beq | jump |
| RegDst | 1 | 0 | 0 | x | x | x |
| ALUSrc | 0 | 1 | 1 | 1 | 0 | x |
| MemtoReg | 0 | 0 | 1 | x | x | x |
| RegWrite | 1 | 1 | 1 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 1 | 0 | 0 |
| Branch | 0 | 0 | 0 | 0 | 1 | 0 |
| Jump | 0 | 0 | 0 | 0 | 0 | 1 |
| ExtOp | x | 0 | 1 | 1 | x | x |
| ALUctr | "R-Type" | Or | Add | Add | Sub | xxx |



# Table de vérité pour le contrôle principal






| op | 000000 | 001101 | 100011 | 101011 | 000100 | 000010 |
|---|---|---|---|---|---|---|
| | R-Type | ori | lw | sw | beq | jump |
| RegDst | 1 | 0 | 0 | x | x | x |
| ALUSrc | 0 | 1 | 1 | 1 | 0 | x |
| MemtoReg | 0 | 0 | 1 | x | x | x |
| RegWrite | 1 | 1 | 1 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 1 | 0 | 0 |
| Branch | 0 | 0 | 0 | 0 | 1 | 0 |
| Jump | 0 | 0 | 0 | 0 | 0 | 1 |
| ExtOp | x | 0 | 1 | 1 | x | x |
| ALUop (symbol) | "R-Type" | Or | Add | Add | Sub | xxx |
| $ALUop_1$ | 1 | 0 | 0 | 0 | 0 | x |
| $ALUop_2$ | 0 | 1 | 0 | 0 | 0 | x |
| $ALUop_3$ | 0 | 0 | 0 | 0 | 1 | x |

# Logique pour RegWrite

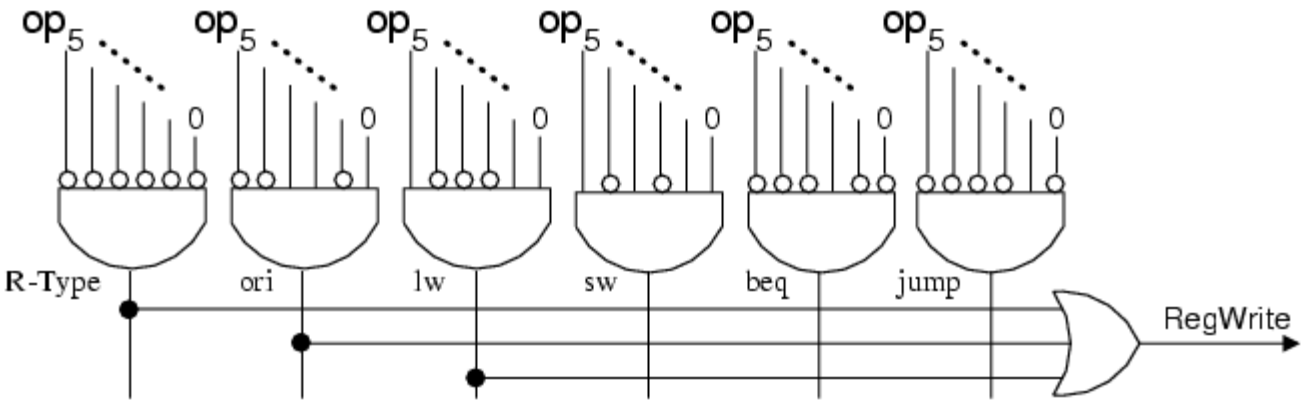


**RegWrite** = **R-Type** ∨ **ori** ∨ **lw**

$$\overline{op_5} \wedge \overline{op_4} \wedge \overline{op_3} \wedge \overline{op_2} \wedge \overline{op_1} \wedge \overline{op_0} \quad (\textbf{R-Type})$$

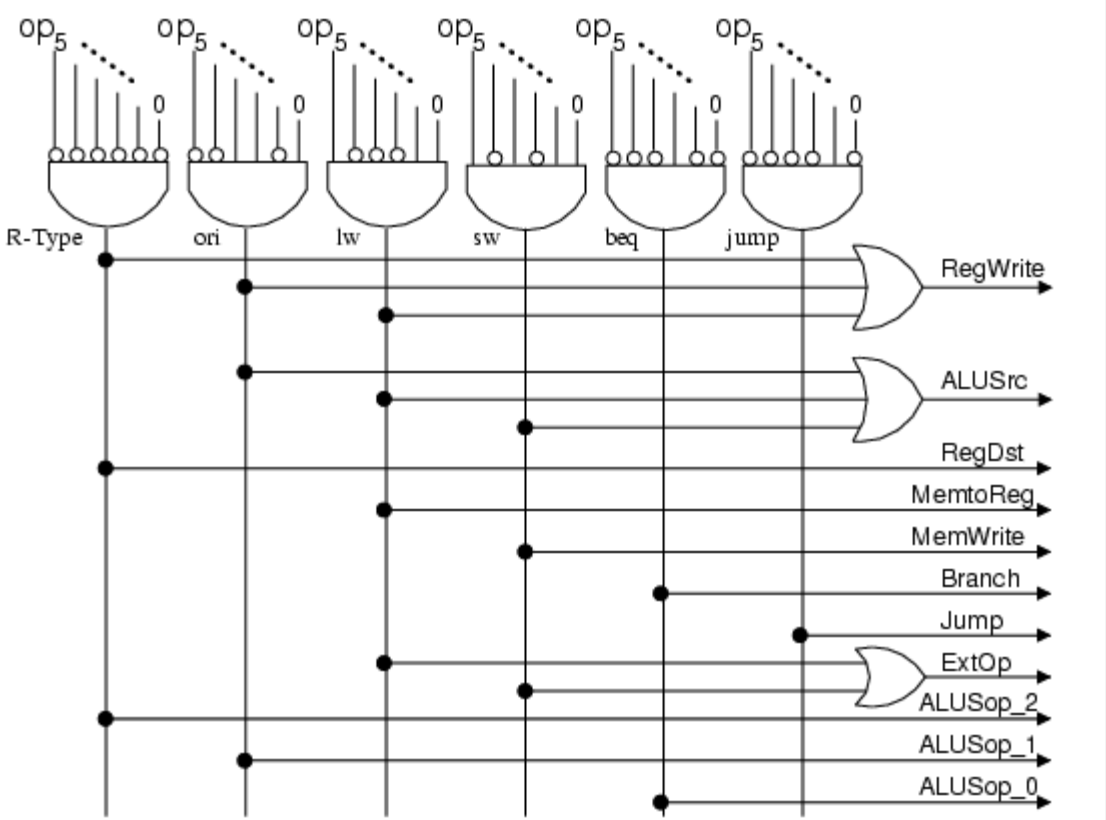
$$\vee \overline{op_5} \wedge \overline{op_4} \wedge op_3 \wedge op_2 \wedge \overline{op_1} \wedge op_0 \quad (\textbf{ori})$$

$$\vee op_5 \wedge \overline{op_4} \wedge \overline{op_3} \wedge \overline{op_2} \wedge op_1 \wedge op_0 \quad (\textbf{lw})$$



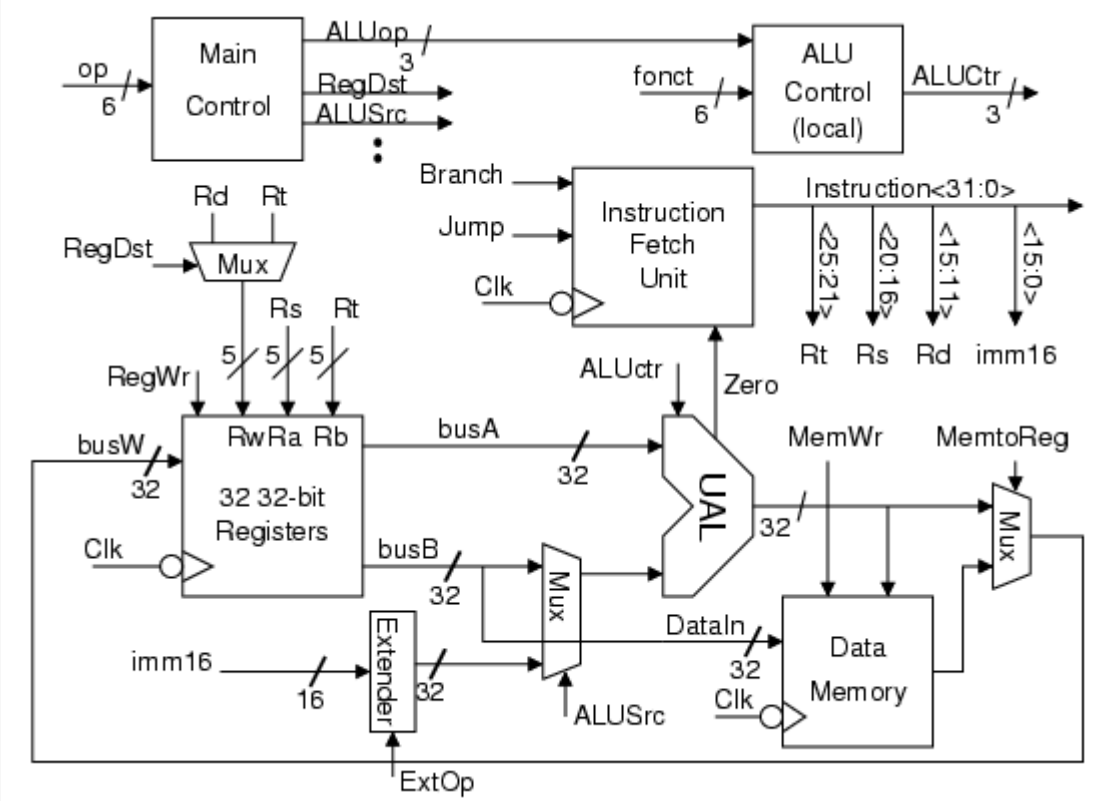
# PLA du contrôle principal





# Un processeur monocycle !





# Inconvénients d'un processeur monocycle



- Temps de cycle trop long (il doit être suffisant pour les "loads")
  - Extraction instruction
  - Accès au banc de registres (access time)
  - calcul UAL
  - Accès à la mémoire
  - Accès au banc de registres (setup time)
- Ce temps de cycle est trop long pour les autres instructions !

# Comparaison des chemins critiques



| Type instruction | Unités fonctionnelles utilisées | | | | |
|---|---|---|---|---|---|
| Format R | Extraction | Acces banc de registres | UAL | Acces banc de registres | |
| Chargement | Extraction | Acces banc de registres | UAL | Acces à la mémoire | Acces banc de registres |
| Rangement | Extraction | Acces banc de registres | UAL | Acces à la mémoire | |
| Branchement | Extraction | Acces banc de registres | UAL | | |
| Saut | Extraction | | | | |

# Implémentation multi-cycles



- Pb monocycle :

  le cycle doit être assez long pour l'instruction la plus lente
- Solution :
  - séparer l'instruction en petites étapes exécutées en un cycle

    $\Rightarrow$ temps de cycle = la plus longue des étapes

    $\Rightarrow$ essayer d'avoir des étapes de longueur similaire
- Avantages :
  - temps de cycle plus court
  - instructions différentes $\rightarrow$ nombre de cycles différent

    (5 pour un load, 3 pour un branchement)
  - les unités fonctionnelles peuvent être utilisées plus d'une fois par instruction

# Unité centrale de contrôle



- La définition du contrôle est plus complexe parce que les signaux dépendent du code d'opération et de l'étape en cours d'exécution (état courant).
- Etat = étape d'exécution de l'instruction ;
- Actions d'un état = signaux devant être activés pour réaliser les actions ;
- Transitions entre états : Déterminés par le code opération et par l'étape en cours d'exécution.

# Avantages de l'approche à plusieurs cycles



- Réduction du temps moyen d'exécution d'une instruction : le CPI moyen est plus grand ( exemple : 4), donc supérieur au CPI de 1 obtenu dans la mise en oeuvre monocycle mais la période d'horloge est plus petite.
- Permet le partage de certaines unités fonctionnelles, lorsqu'elles sont utilisées à des étapes distinctes.
  - Exemple : Une seule UAL est nécessaire pour calculer PC+4, adresse effective, opération R, etc.

# Microprogrammation : simplifier la conception du contrôle



- Microprogrammation = Technique pour la réalisation du contrôle dans le cas de la mise en oeuvre à plusieurs cycles :
  - l'unité de contrôle est conçue comme un microprogramme.
  - La fonction de ce micro-programme est d'interpréter les instructions du programme objet (en langage d'assemblage):
    - une instruction objet (MIPS) s'exécute via l'interprétation d'une séquence de micro-instructions.
    - Chaque micro-instruction peut être vue comme l'encodage (direct ou complexe) des signaux devant être activés pour réaliser les tâches associées à une étape de l'exécution d'une instruction.

# Questions



Q1. Comment obtient-on le cycle horloge dans une implémentation multi cycle ?

Q2. Quels sont les inconvénients de l'implémentation multi cycle ?

Q3. Discuter les deux implémentations du contrôle puis donner une ébauche pour un contrôle microprogrammé


Thank's