# Micro-architecture d'un processeur mono-cycle ARM7TDMI simplifié

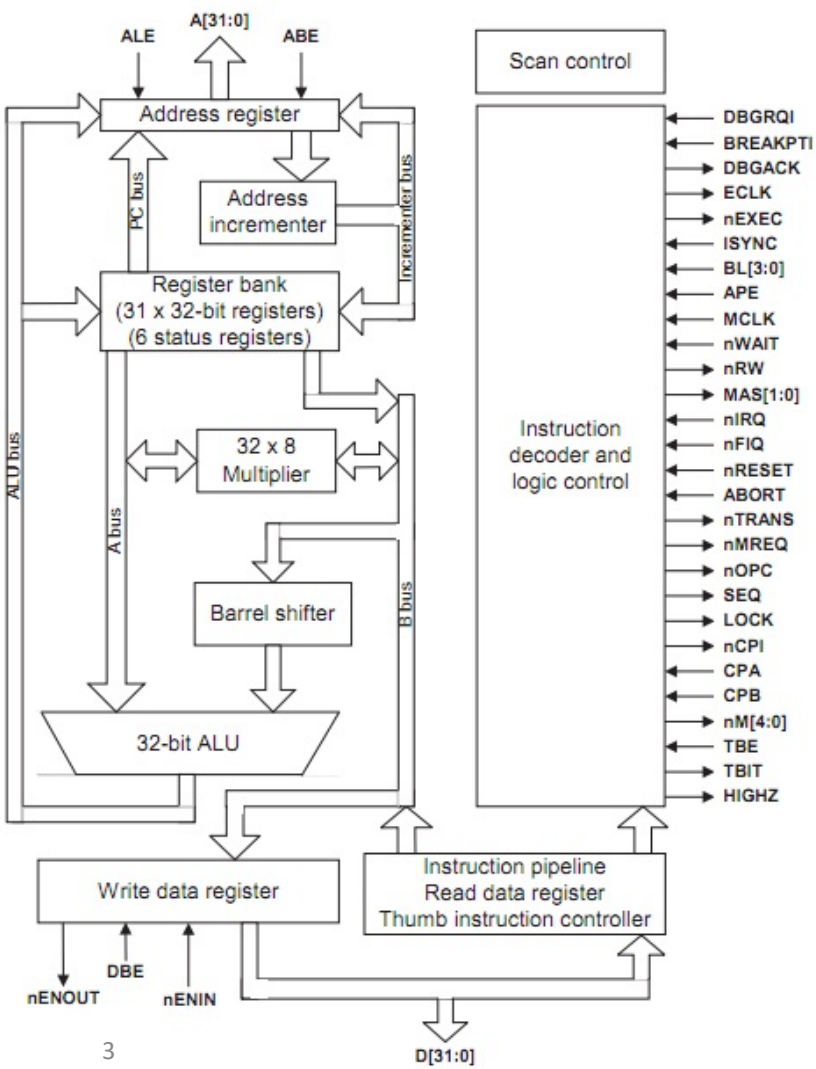
yann.douze@sorbonne-universite.fr

# Introduction

- Objectif du cours : Décrire l'architecture d'un processeur Monocycle
- On prend pour exemple le jeu d'instruction du processeur ARM7TDMI
- 1ère partie : présentation de l'architecture du processeur ARM7TDMI
- 2ème partie : comment concevoir un processeur étape par étape

# 1ère partie : Architecture ARM7TDMI

- Processeur 32-bit
- UAL 32-bit
- File de registres
- Registre à décalage
- Multiplieur 32x8

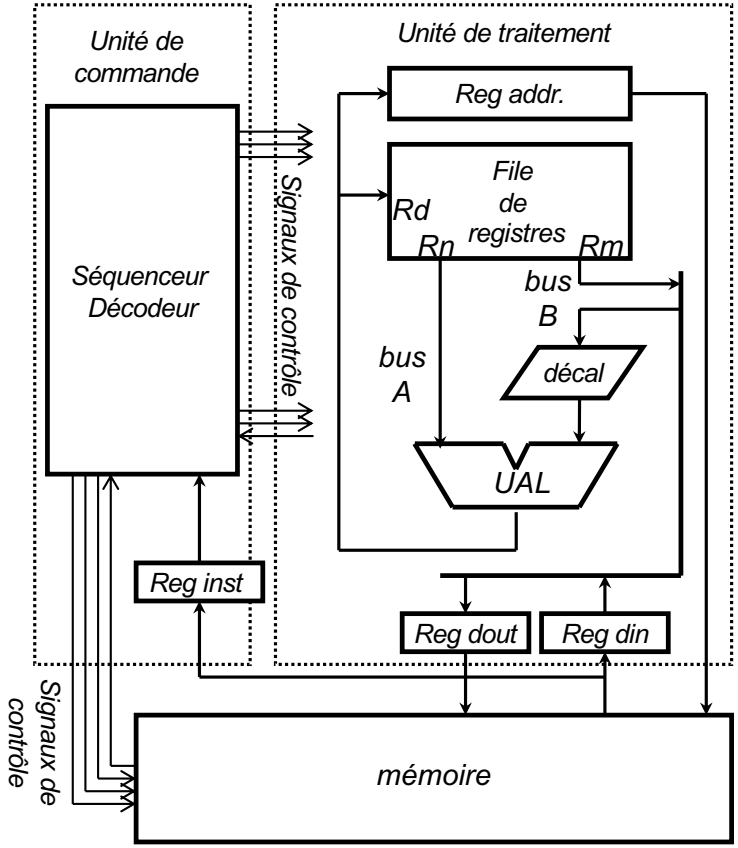


# Caractéristiques générales

- Architecture load-store
  - Les instructions ne traitent que des données en registre et placent les résultats en registre. Les seules opérations accédant à la mémoire sont celles qui copient une valeur mémoire vers un registre (load) et celles qui copient une valeur registre vers la mémoire (store).
- Format de codage fixe des instructions
  - Toutes les instructions sont codées sur 32 bits.
- Format 3 adresses des instructions de traitement
  - Deux registres opérandes et un registre résultat, qui peuvent être spécifiés indépendamment.
- Exécution conditionnelle
  - Chaque instruction peut s'exécuter conditionnellement
- UAL + shift
  - Possibilité d'effectuer une opération Arithmétique ou Logique et un décalage en une instruction (1 cycle), la ou elles sont realisées par des instructions séparées sur la plupart des autres processeurs

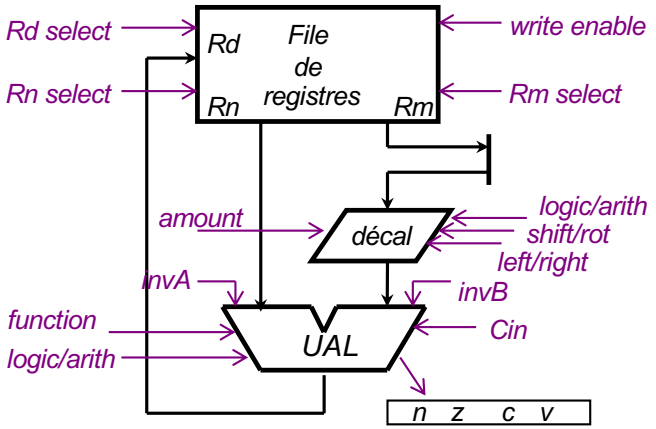
# Unité de traitement

- Unités fonctionnelles de l'UT:
  - File de registres
    - 16 registres pour permettre une manipulation souple des données et stockage des résultats de l'UAL
  - Unité Arithmétique et Logique (UAL)
    - 2 opérandes : entrée A donnée provenant d'un registre, entrée B donnée reliée au décaleur
    - Résultat de l'UAL: renvoyé dans un registre
  - Registre à décalage

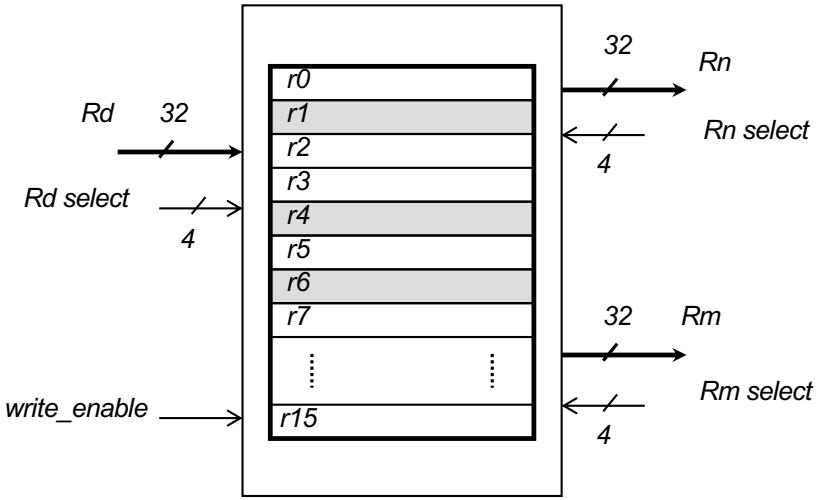


# Organisation de l'unité de traitement

- Signaux de commande de l'unité de traitement:
  - File de registres
    - **Rd, Rn, Rm select : sélection du registre cible dans la file des 16 registres**
    - **write enable: activation de la file de registres**
  - UAL
    - **Indicateurs: indiquent l'état de l'UAL après une opération (ex: retenue C)**
    - logic/arith+function: mode logique ou arthmétique, dans chaque mode choix de la fonction
    - InvA, invB: inversion des opérandes A et B (not)
    - Cin: injection du bit retenue C
  - Décaleur
    - shift/rot: opération de décalage ou rotation
    - logic/arith: opération logique ou arthmétique
    - Amount: nombre de bits de décalage/rotation



# Banc de registre ARM7TDMI




- 16 registres utilisateurs r0, …, r15
- Notation pour l'appel aux instructions:
  - INST Rd, Rn, Rm

# Instruction MOV

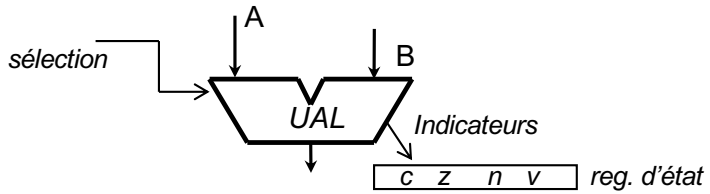
- Instructions de mouvements de données entre registres
  - MOV (Move), MVN (Move not)
    - MOV Rd, *#literal*
    - MOV Rd, Rn
    - MOV Rd, Rm, *shift*

    - *Mouvements de données entre registres, ou d'une constante vers registre, uniquement .*

    - *#literal*: valeur immédiate (constante)
    - *shift*: le deuxième opérande peut être sujet à un décalage

  - Exemples
    - `MOV r3, #2` `@ r3 ← 2`
    - `MOV r3, r4` `@ r3 ← r4`
    - `MOV r3, r4, LSL #2` `@ r3 ← r4<<2`

# UAL et registre d'état




- Unité Arithmétique et Logique (UAL) : effectue des opérations arithmétiques et logiques
  - (ADD, SUB, AND, OR, …)
- Le registre d'état (Status Register) fournit des indications sur les résultats d'opération:
  - C: Carry
    - bit indicateur de dépassement pour une opération arithmétique (ou de décalage)
  - Z: Zero
    - bit indicateur de résultat nul de l'UAL
  - N: Négatif
    - bit indicateur de résultat négatif de l'UAL
  - V: Débordement (oVerflow)
    - bit indicateur de dépassement de capacité du résultat de l'UAL (modification du bit de signe)

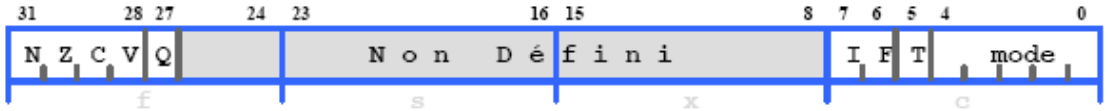
# Registre d'état (Status Register)

- Exemple sur 4 bits : 1 0 1 0 + 1 0 0 1 = (1) 0 0 1 1
  - Résultat de l'UAL: 0 0 1 1
  - C = 1
  - Z = 0, le résultat est différent de 0 0 0 0
  - N = 0, 0 0 1 1 est un nombre positif car le bit de signe ($4^{ème}$ bit) = 0
  - V = 1, car 1 0 1 0 et 1 0 0 1 sont des nombre négatifs et le résultat est positif

  - les bits C, V, N sont examinés ou ignorés en fonction de l'interprétation des nombres
    - Si les nombres sont en représentation non-signée, C est utile, V et N inutiles
    - Si les nombre sont en représentation signée, C est inutile, V et N sont utiles
    - Ces indicateurs sont positionnés par l'UAL, le programmeur les utilise ou non en fonction de ses besoins (par exemple pour effectuer une addition sur 8 bits à partir de l'addition 4 bits dans le cas de l'exemple)

# Le registre d'état : CPSR

- CPSR: Current Program Status Register
- Contient les indicateurs Z (zero), N (negative), C (carry), V (overflow)
- Donne des informations sur le résultat d'une opération arithmétique ou d'une comparaison
- Permet à une instruction de s'exécuter ou non en fonction de ce résultat (exécution conditionnelle)
- Permet de conserver la retenue pour effectuer des opérations sur plus de 32-bit



# Instructions conditionnelles

- Sur ARM, toutes les instructions peuvent s'exécuter de façon conditionnelle: une instruction sera exécutée ou non en fonction de l'état des bits N, Z, C, V
  - En assembleur ARM, il suffit d'ajouter au mot-clé de l'instruction un suffixe qui représente sa condition d'exécution

| Extension Mnémonique | Interprétation | Etat indicateur |
|---|---|---|
| EQ | Equal/equals zero | Z = 1 |
| NE | Not Equal | Z = 0 |
| **LT** | **Lesser Than** | **N = 1** |
| GE | Greater than or equal | N = 0 |
| VS | Overflow | V = 1 |
| VC | Not overflow | V = 0 |

# Instructions conditionnelles

- Exemple d'exécution conditionnelle en fonction du résultat d'une comparaison

```
CMP r4, r5                  @ comparer r4 et r5
SUBGT r4, r4, r5            @ si >, alors r4 ← r4 – r5
SUBLE r5, r5, r4            @ si ≤, alors r5 ← r5 – r4
```

- Exemple d'exécution conditionnelle en fonction du résultat d'un calcul

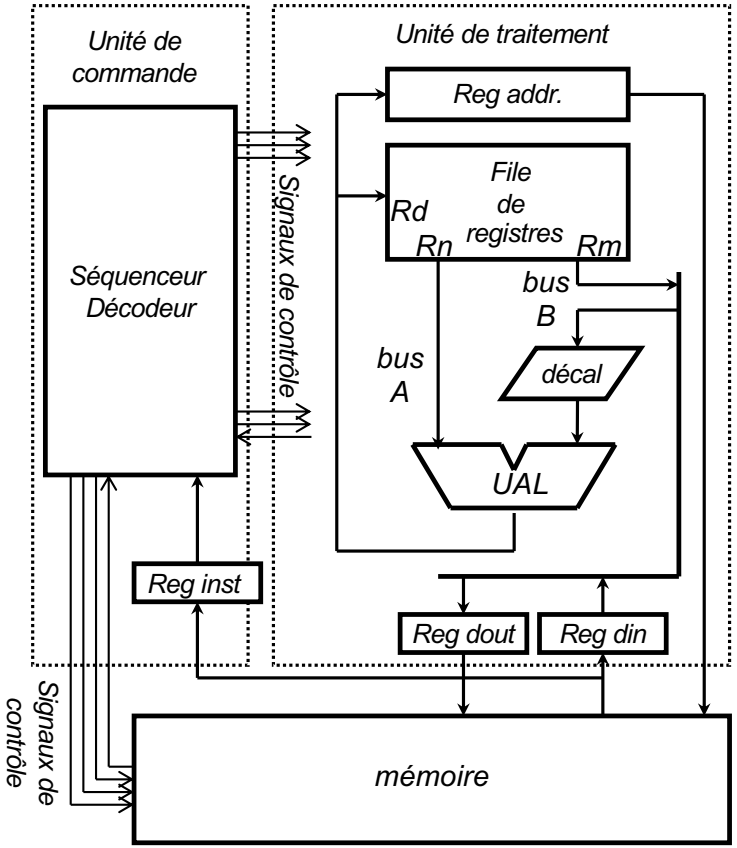
```
SUBS r4, r4, #10            @ r4 ← r4 – 10
MOVPL r5, #1                @ si r4>0, r5 ← +1
MOVMI r5, #-1               @ si r4<0, r5 ← -1
MOVEQ r5, #0                @ si r4=0, r5 ← 0
```

- Exemple de branchement conditionnel

```
ADDS r6, r4, r5             @ r6 ← r4 + r5
BLVS ErrDebordement         @ Branch and Link (BL) si VS
                            @ si débordement (VS),
                            @ appel du sous-programme
                            @ ErrDebordement
```

# 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



# 2ème partie :
# Comment concevoir un processeur étape par étape

1. Analyser le jeu d'instructions
2. Sélectionner un ensemble de composants pour réaliser le chemin des données et établir une méthodologie d'horloge
3. Assembler les composants afin de réaliser chemin de données
4. Etablir la logique de contrôle

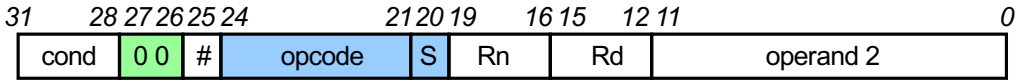
# Étape 1 : Instructions ARM7TDMI

- Traitement de données :
  - ADD Rd, Rn, RM → Rd = Rn + Rm (Addition)
  - ADD Rd, Rn, #Imm → Rd = Rn + Imm (Addition)
  - MOV Rd, #Imm → Rd = Imm (Move)
  - CMP Rn, #Imm → Flag CPSR = Rn – Imm (Compare)
- Accès mémoire :
  - LDR Rd, [Rn, #Offset] → Rd = Mem[Rn + Offset] (Load, lecture)
  - STR Rd, [Rn, #Offset] → Mem[Rn + Offset] := Rd (Store, écriture)
- Branchements :
  - B{AL} label → PC = PC + Offset (Branch always)
  - BLT label → Si Flag N =1 => PC = PC + Offset (Branch if Lesser Than)

# Codage binaire d'une instruction

- Une instruction de traitement est composée des champs
  - Registre résultat (destination)
    - **Rd**: R0…R15
  - Premier opérande (toujours un registre)
    - **Rn**: R0…R15
  - Deuxième opérande
    - **Rm**: R0…R15
    - Rm: R0…r15 avec décalage (log/arith/rot, quantité)
    - Une valeur immédiate (constante)
  - Opération à réaliser par l'UAL
    - Opcode

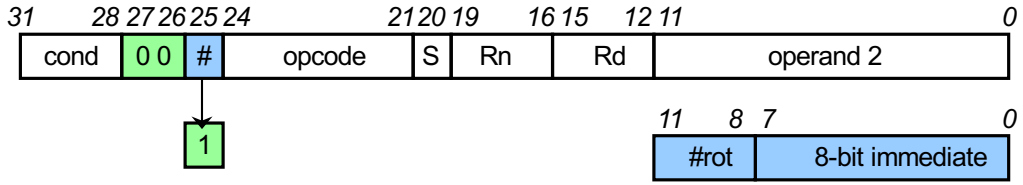
# Instructions Traitement de données ARM7TDMI



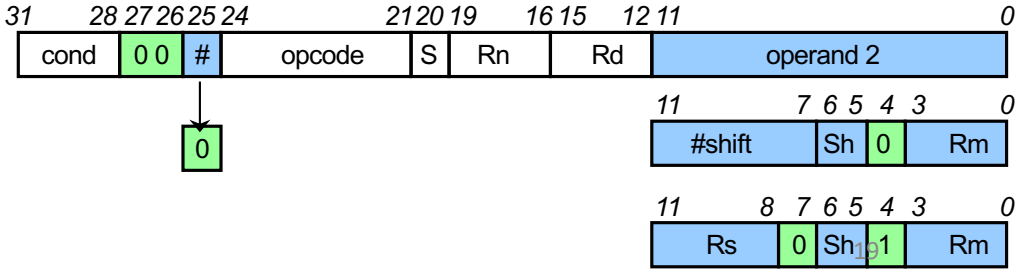

| Asm | Opcode |
|---|---|
| AND | 0000 |
| EOR | 0001 |
| SUB | 0010 |
| RSB | 0011 |
| ADD | 0100 |
| ADC | 0101 |

- *Rd* → numéro du registre de destinations
  - 4 bits pour sélection parmi les 16 registres possibles
- *Rn* → numéro du registre qui sert de premier opérande
  - 4 bits pour sélection parmi les 16 registres possibles
- *operand2* → Rm ou immédiat
- Exemple : ADD Rd, Rn, operand2 → Rd = Rn + operand2

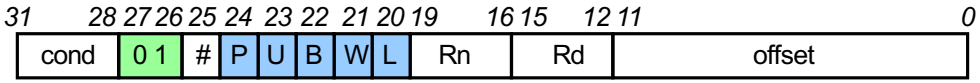
# Instructions Traitement de données ARM7TDMI




- Si le bit # est à 1, *operand2* est une valeur littérale
  - Exemple : ADD R0, R1, #10 → R0 = R1 + 10
- Si le bit # est à 0, *operand2* est un registre (*Rm*) sur lequel on applique (ou non) une rotation/décalage
  - Exemple : ADD R0, R2, R3 → R0 = R2 + R3



# Instructions de transfert de données (Load et Store)




- *B* : byte/word
  - 1 accès 8 bits, 0 accès 32 bits
- *L* : load/store
  - 1 load, 0 store
- Exemples :
  - LDR Rd, [Rn, #Offset] → Rd = Mem[Rn + Offset] (Load, lecture)
  - STR Rd, [Rn, #Offset] → Mem[Rn + Offset] := Rd (Store, écriture)

# Instructions de branchement

- Branch (B)


- Offset : déplacement signé sur 24 bits
- Cond : exécution conditionnelle
- BLT → Branch if lesser than
- Branchement si le flag N du CPSR = 1

| Asm | Cond |
|---|---|
| VS | 0110 |
| VC | 0111 |
| HI | 1000 |
| LS | 1001 |
| GE | 1010 |
| LT | 1011 |

# Récupération des instructions (Fetch)

- Tout commence par récupérer l'instruction
- PC : Program Counter

**op | rn | rd | rm = MEM[ PC ]**

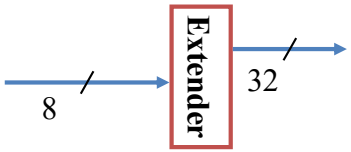
**op | rn | rd | Imm8 = MEM[ PC ]**

| inst | Register Transfers | |
|---|---|---|
| **ADD** | **R[rd] <– R[rn] + R[rm];** | **PC <– PC + 1** |
| **SUB** | **R[rd] <– R[rn] – R[rm];** | **PC <– PC + 1** |
| **LOAD** | **R[rd] <– MEM[ R[rn] + sign_ext(Imm8)];** | **PC <– PC + 1** |
| **STORE** | **MEM[ R[rn] + sign_ext(Imm8) ] <– R[rd];** | **PC <– PC + 1** |

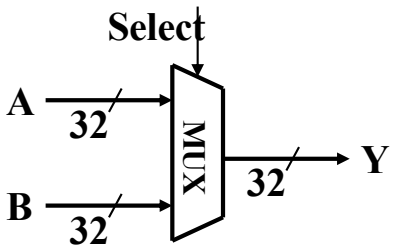
# Etape 2 : Eléments du chemin de données (datapath)

# Logique Combinatoire

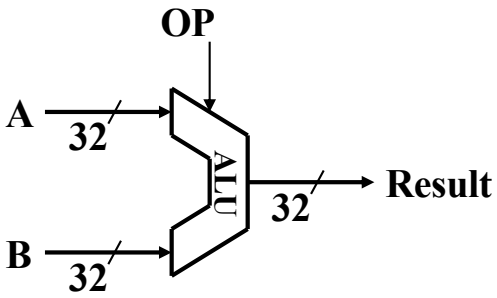
- Extender




- MUX



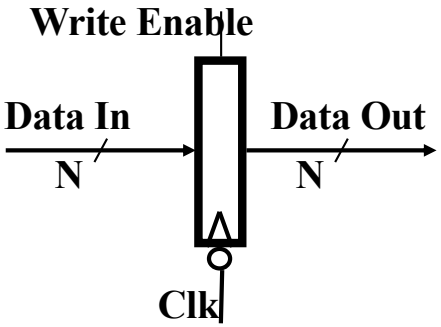

- ALU



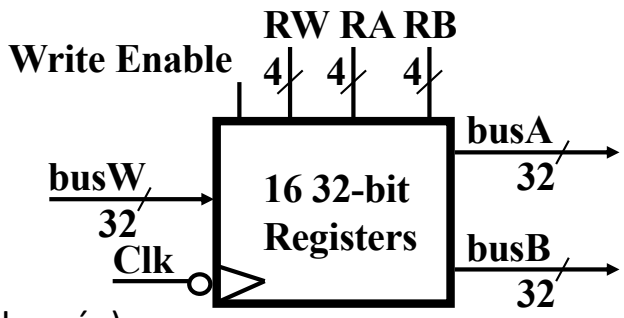

Carry
Zero
Sign
Overflow

# Elément de mémorisation : Registre

- Registre
  - Similaire à une bascule D Flip-Flop
    - N-bit en entrée et sortie
    - Entrée Write Enable
  - Write Enable:
    - (0): La donnée en sortie n'est pas modifiée
    - (1): Data Out recopie Data In

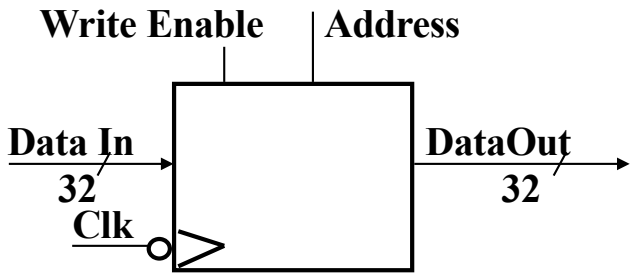


# Elément de memorisation : Banc de registre



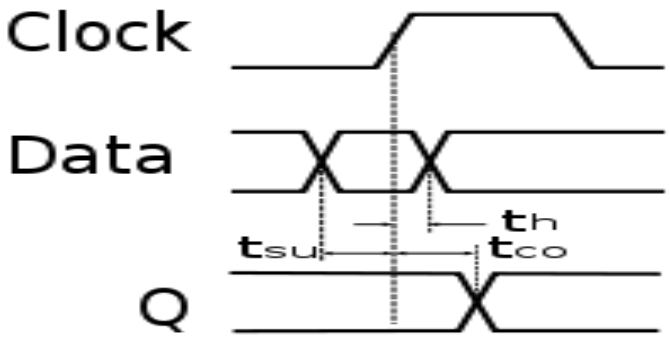

- Le banc de registre contient 16 registres:
  - Deux bus 32 bit en sorties : busA et busB
  - Un bus 32 bit en entrée : busW
- Les registres sont sélectionnées :
  - RA (nombre) sélectionne le registre à mettre sur le busA (donnée).
  - RB (nombre) sélectionne le registre à mettre sur le busB (donnée).
  - RW (nombre) sélectionne dans quel registre écrire à partir du busW lorsque Write Enable = 1
- Entrée d'horloge (CLK)
  - L'entrée CLK est évalué UNIQUEMENT pendant l'opération d'écriture
  - Pendant l'opération de lecture, se comporte comme un bloc logique combinatoire :
  - RA ou RB valide => busA ou busB valide après un temps d'accès. (Tp : temps de propagation)

# Elément de memorisation : Mémoire idéale




- Mémoire (idéalisé)
  - Un bus en entrée : Data In
  - Un bus en sortie : Data Out
- Le mot mémoire est sélectionné par :
  - L'adresse sélectionne le mot à mettre en sortie des données
  - Write Enable = 1: l'adresse sélectionne le mot mémoire à écrire via le bus Data In
- Entrée d'horloge (CLK)
  - L'entrée CLK est évalué UNIQUEMENT pendant l'opération d'écriture
  - Pendant l'opération de lecture, se comporte comme un bloc logique combinatoire :
  - Address valide => Data Out valide après un temps d'accés (Tp)

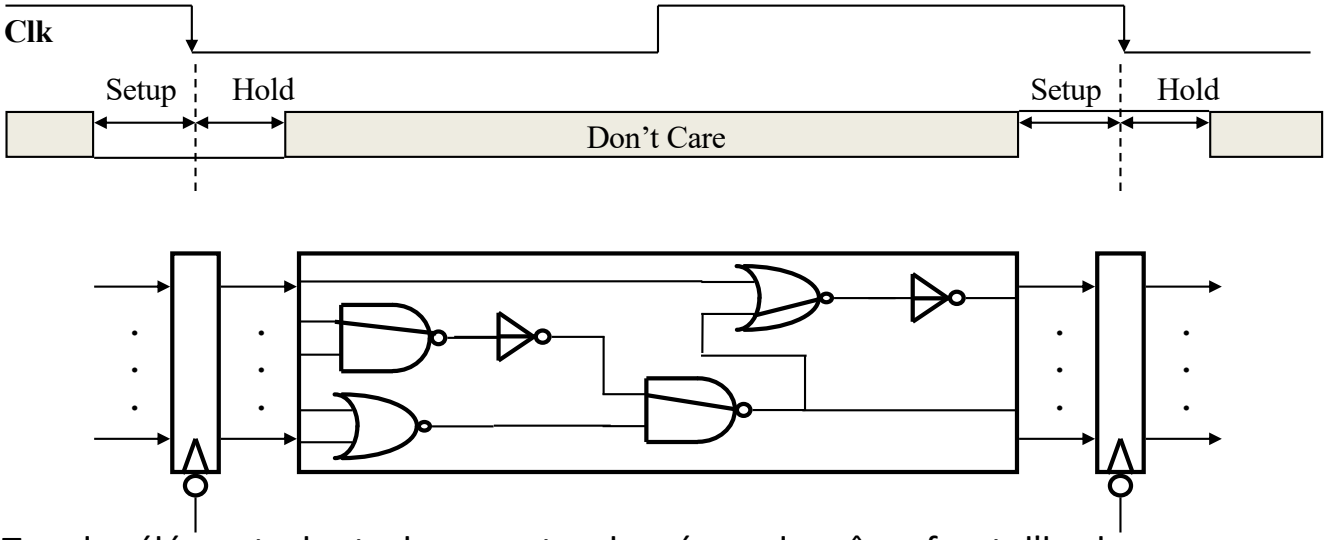
# Timing d'une Bascule D




- Tsu : Setup Time
  - Temps prépositionnement
- Th : Hold Time
  - Temps maintien

- Tco : Clock to Output Time
- Tp : Temps propagation

# RTL : Register Transfer Level




- Tous les éléments de stockage sont cadencés par le même front d'horloge

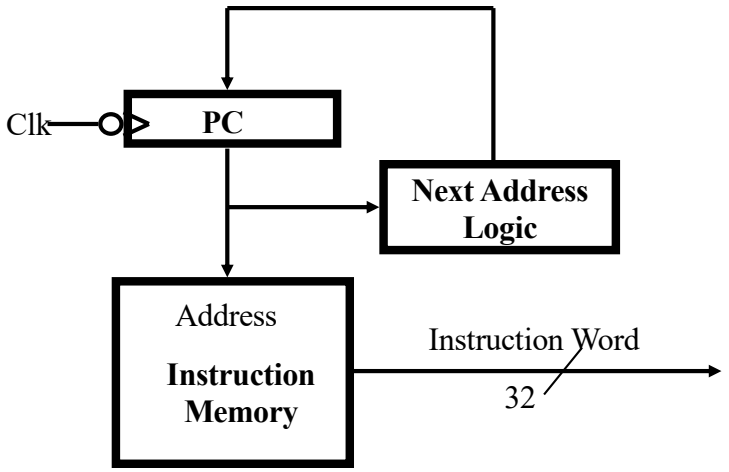
- Cycle Time = CLK-to-Q (Tco) + Longest Delay Path (Tp) + Setup (Tsu)
- Fmax du processeur = 1/ Cycle Time

# Etape 3 : Assembler les composants afin de réaliser chemin de données (datapath)

- Récupération des instructions (Fetch)
- Exécuter les instructions -> Assemblage du chemin de données

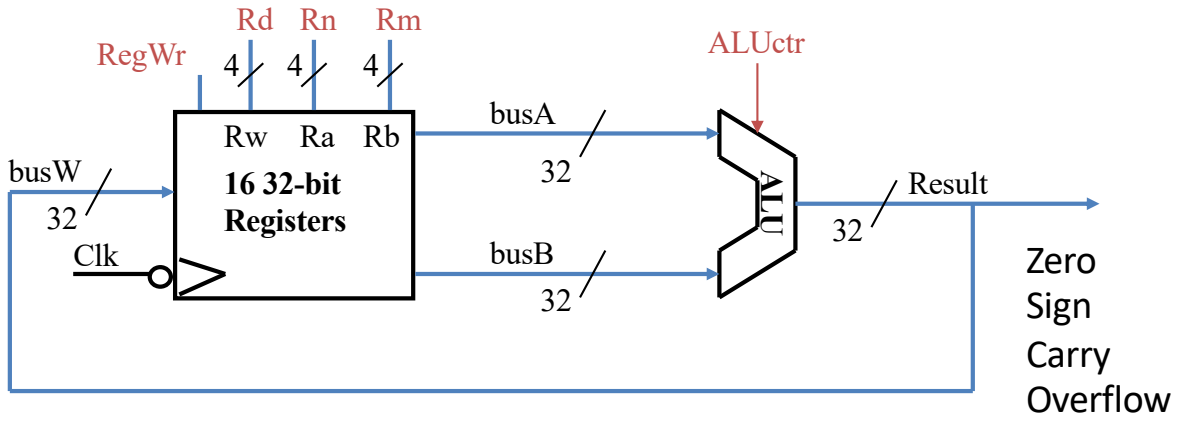
# 3a: Unité de récupération des instructions (Fetch)

- Objectif
  - Récupérer l'instruction: mem [PC]
  - Mes à jour le compteur de programme:
    - Code séquentiel : PC <- PC + 1
    - Branchement : PC <- “something else”

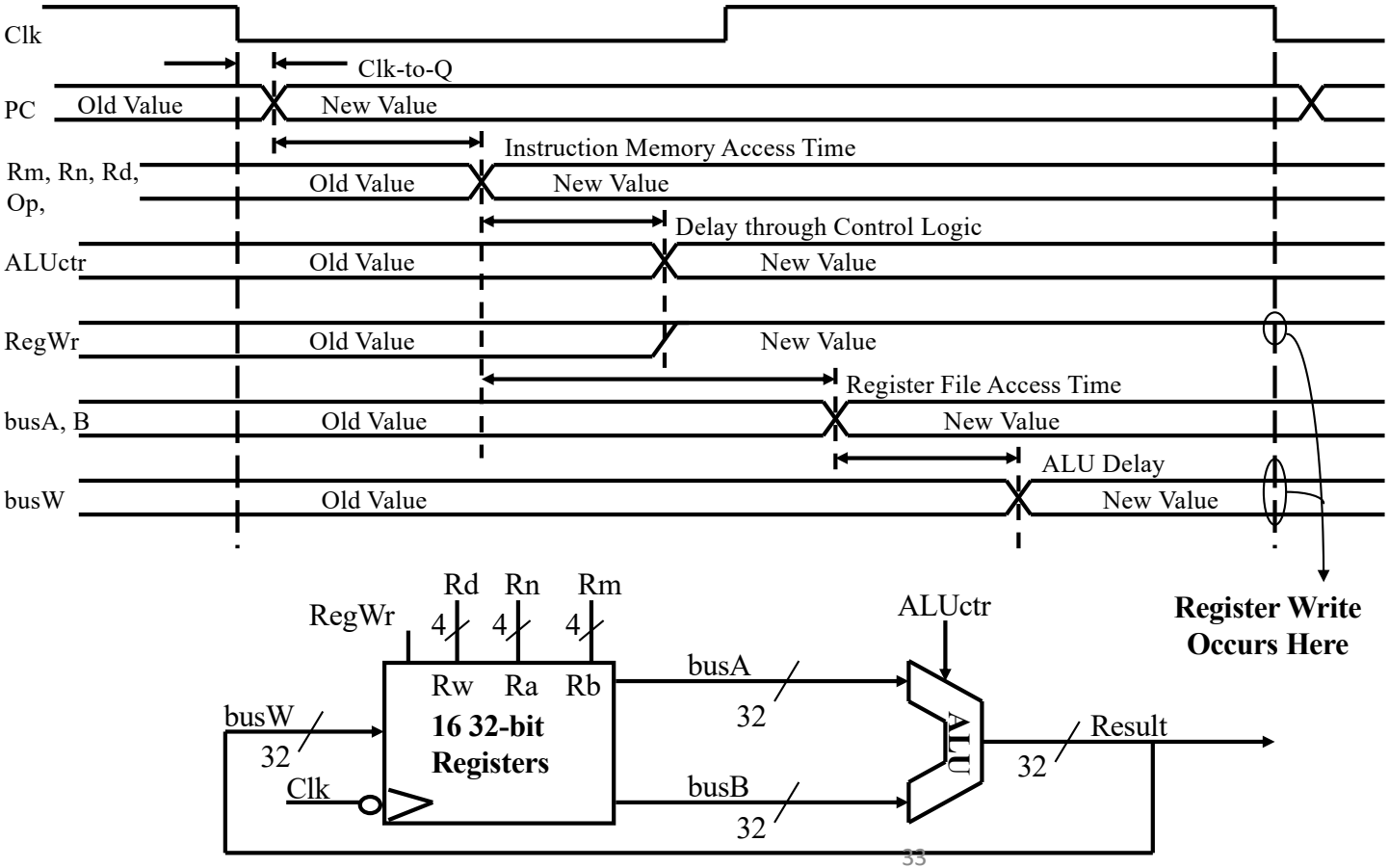


# 3b: Addition et soustraction

- **R[rd] <- R[rn] + R[rm]** Exemple: **add rd, rn, rm**
  - Ra, Rb, et Rw proviennent des champs rn, rm, et rd.
  - ALUctr et RegWr: proviennent de la logique de controle après le décodage de l'instruction.

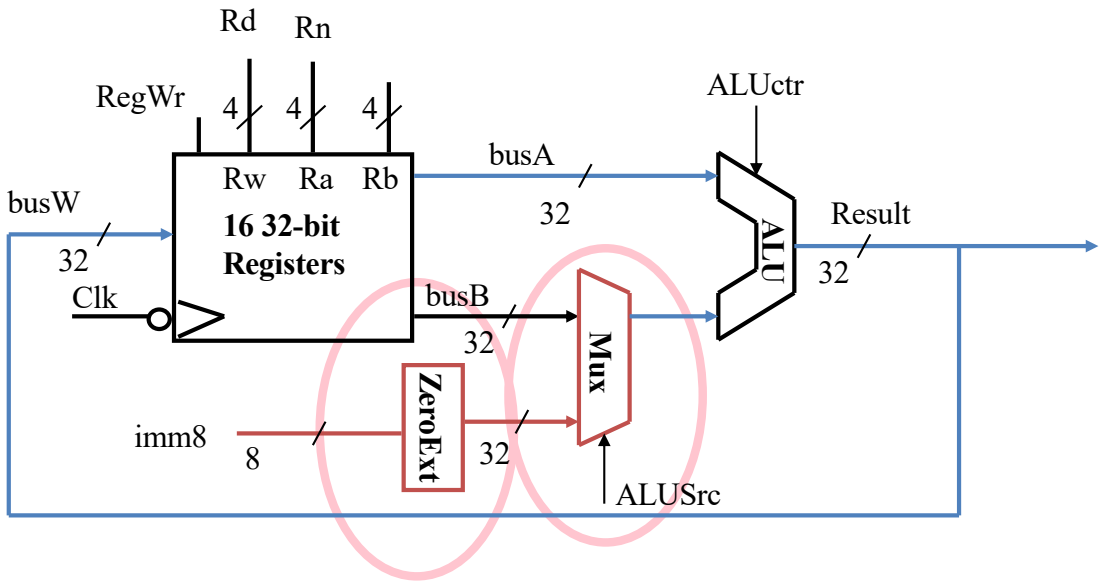


# Timing de registre à registre



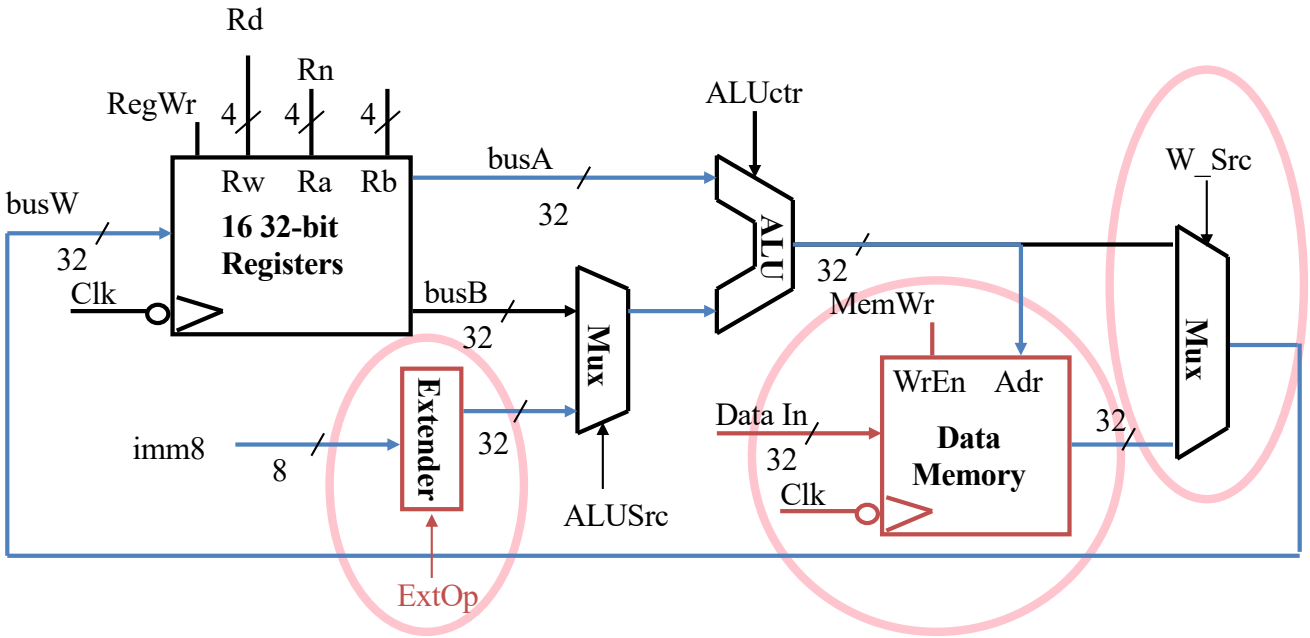
# 3c: Opérations logique avec un immédiat

- R[rd] <- R[rn] + ZeroExt[imm8]



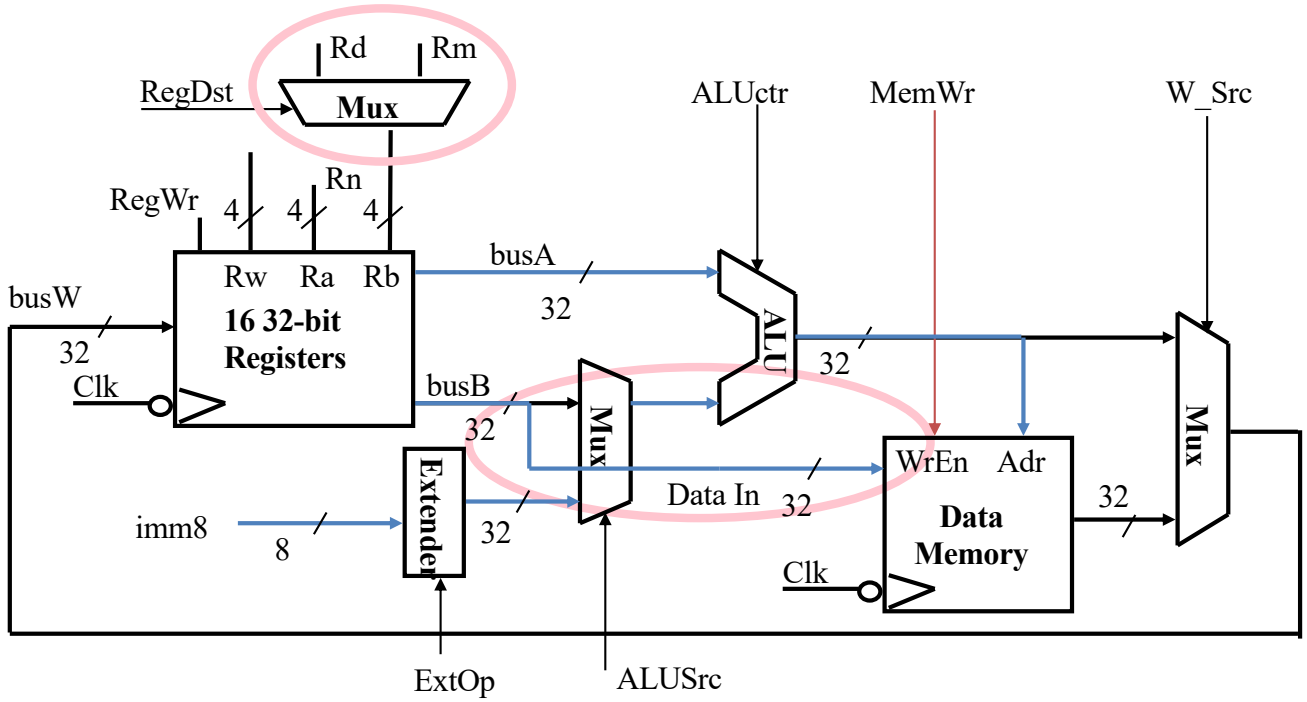
# 3d: Instructions Load

- R[rd] <- Mem[R[rn] + SignExt[imm8]]



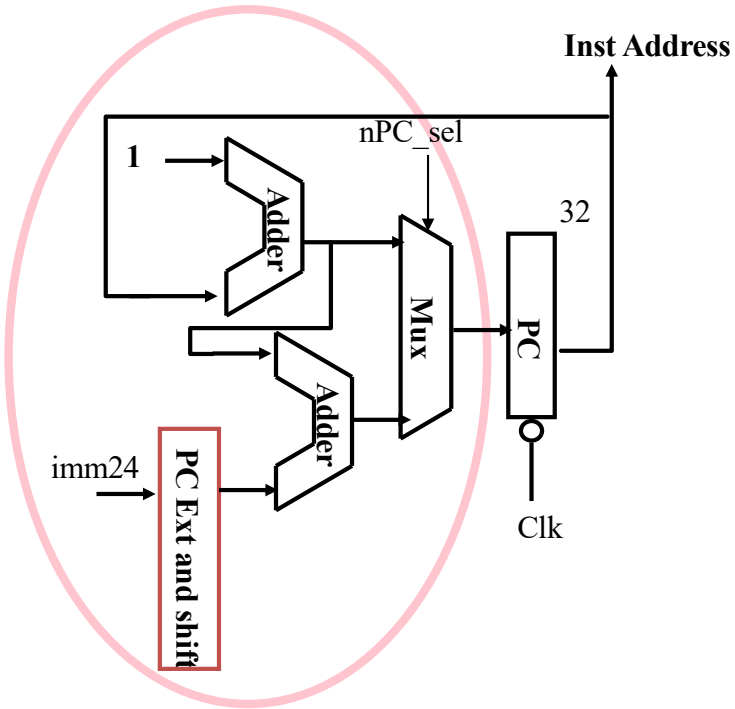
# 3e: Instructions Store

- Mem[ R[rn] + SignExt[imm8] ] <- R[rd]

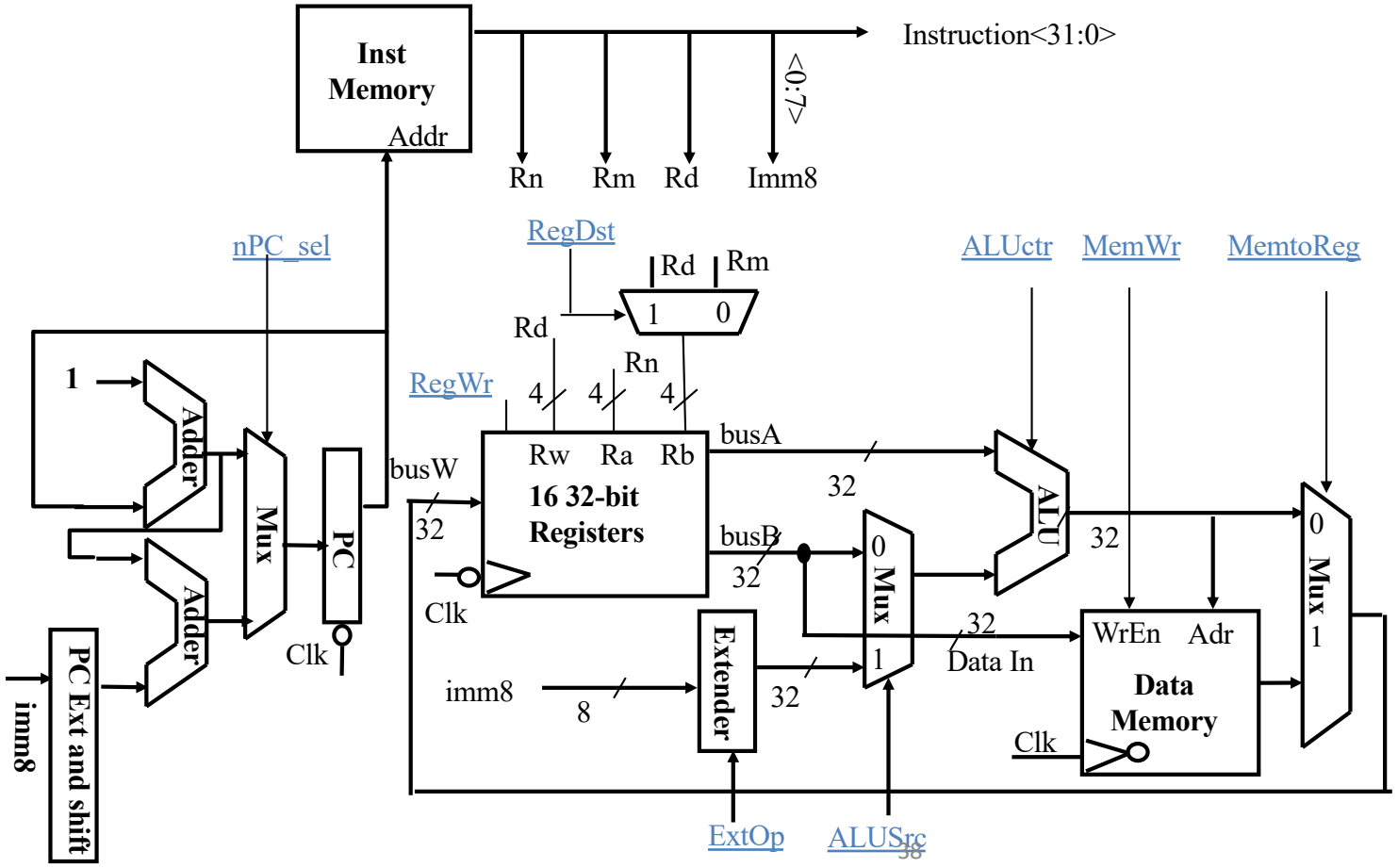


# Chemin de donnée pour l'opération de branchement

- CMP Rn, #Imm : Flag N du CPSR = Rn – #Imm
- BLT label : Branch Lesser Than, Branchement que si le flag N du CPSR = 1

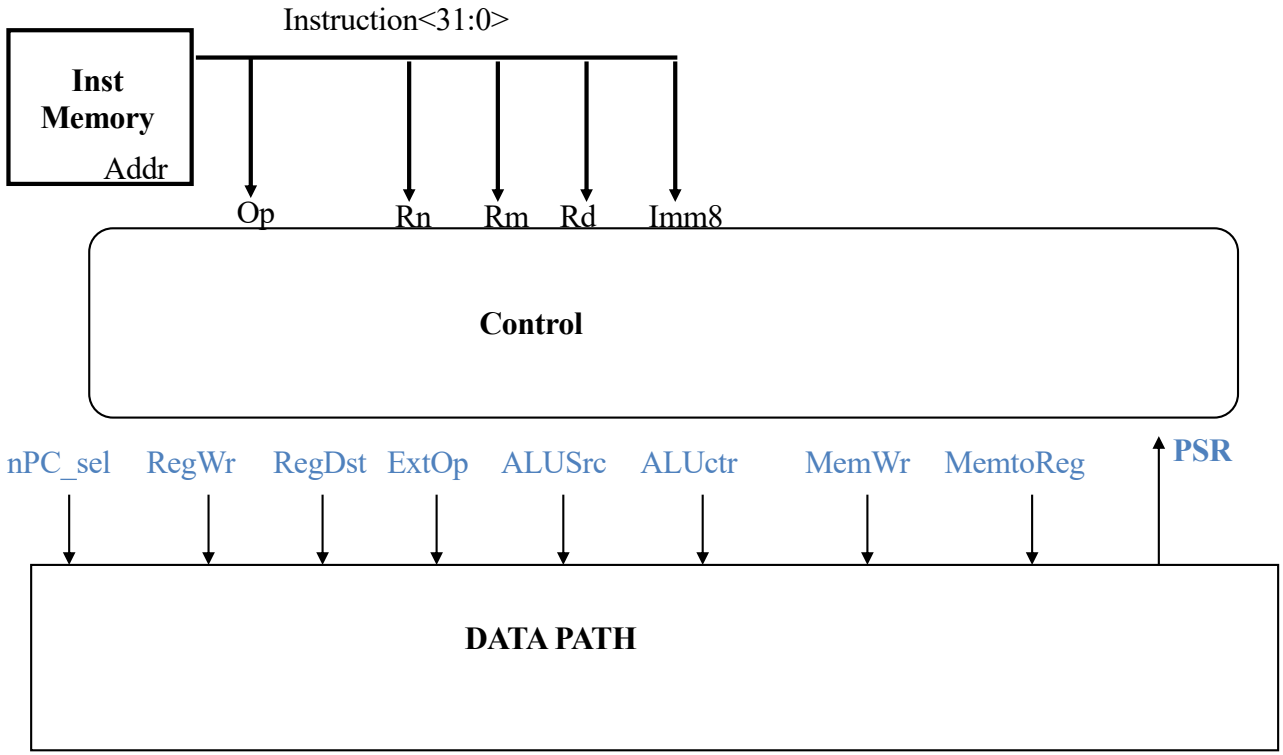


# Tout mettre ensemble: un chemin de données à 1 cycle machine



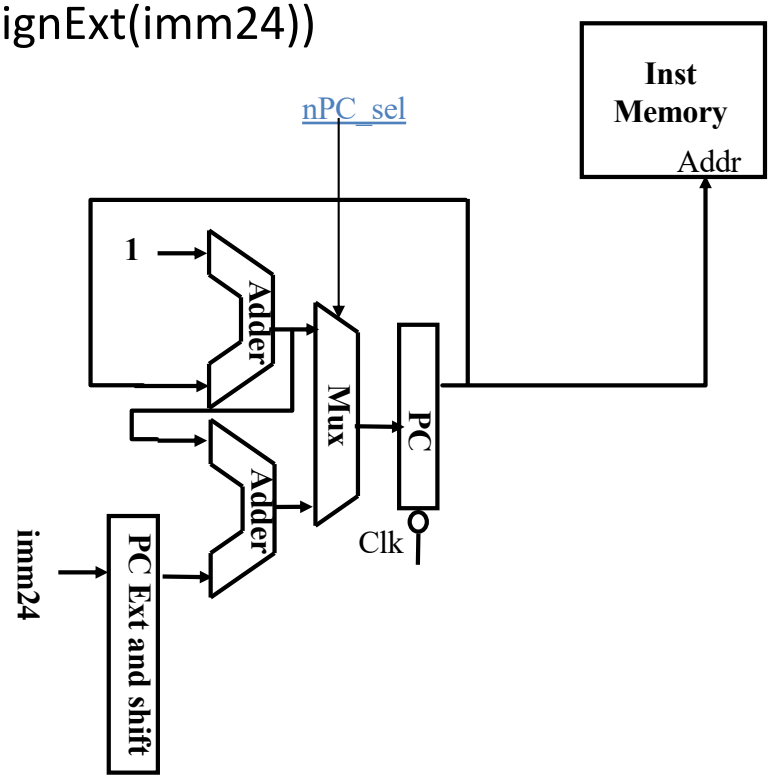
# 4/ Etablir la logique de contrôle

# Etape 4: Signaux de controle



# Signification des signaux de controle

- nPC_sel: 0 => PC <– PC + 1;
  1 => PC <– PC + 1 + (SignExt(imm24))



# Signification des signaux de controle

- ExtOp: “zero”, “sign”
- ALUsrc: 0 => regB; 1 => immed
- ALUctr: “add”, “sub”, “mov”

- **MemWr: write memory**
- **MemtoReg: 1 => Mem, 0 => ALU out**
- **RegDst: 0 => “rm”; 1 => “rd”**
- **RegWr: write dest register**