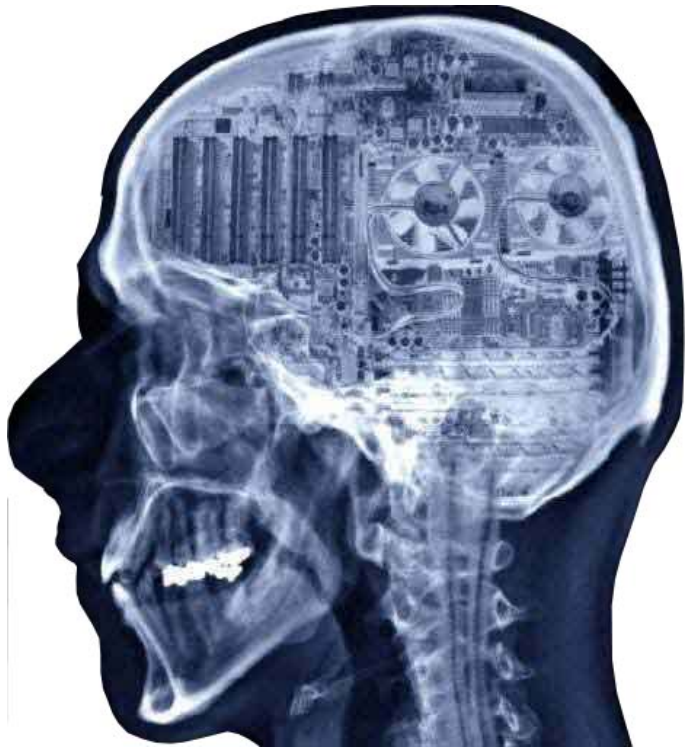
Unité de traitement



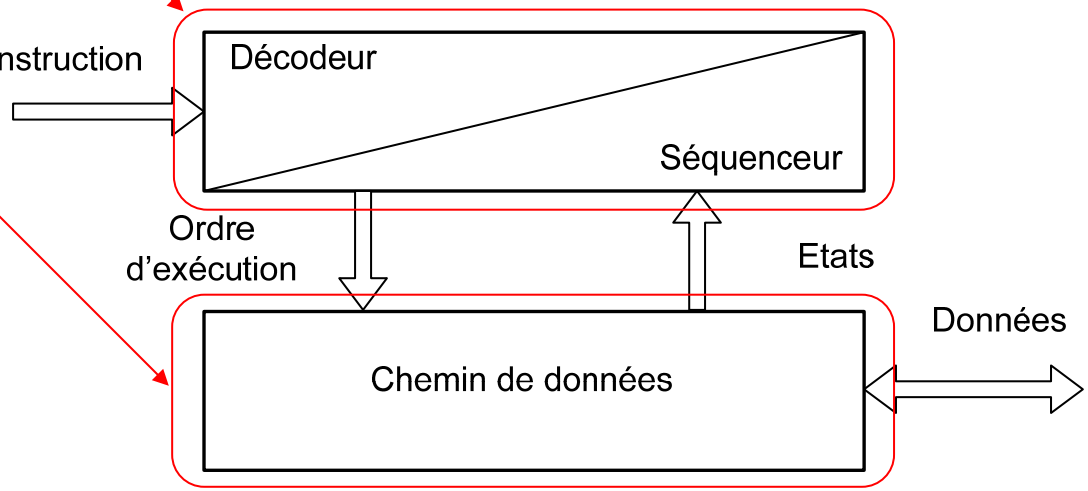
Décomposition fonctionnelle d'un microprocesseur

L'organisation fonctionnelle d'un processeur classique est typiquement découpée en deux unités fonctionnelles principales appelées :

le chemin de données et l'unité de contrôle

Réalisation du séquencement :
- enchaînement
- rupture de séquence
- décodage

Réalisation des :
- opérations arithmétiques
- tests
- indexations ...



Unité de contrôle

Chaque **cycle d’instruction** se compose d’un **ensemble de micro-opérations** générant des signaux de contrôle.

L’unité de contrôle contient un **décodeur d’instruction** et un **séquenceur**.
Le séquenceur peut être câblé ou microprogrammé

L'unité de contrôle dirige le fonctionnement de tous les autres éléments de **l'unité centrale de traitement** (CPU : Central Processing Unit) en leur envoyant des signaux de commande.

Unité de contrôle

Les principaux éléments de l'unité de contrôle sont :

le compteur de programme (PC : Program Counter) :

⇨ registre contenant l'adresse mémoire où se trouve l'instruction à chercher

le registre d'instruction (RI) :

⇨ registre contenant l'instruction qui doit être exécutée

le décodeur d'instruction :

⇨ détermine l'opération à effectuer et les opérandes impliquées

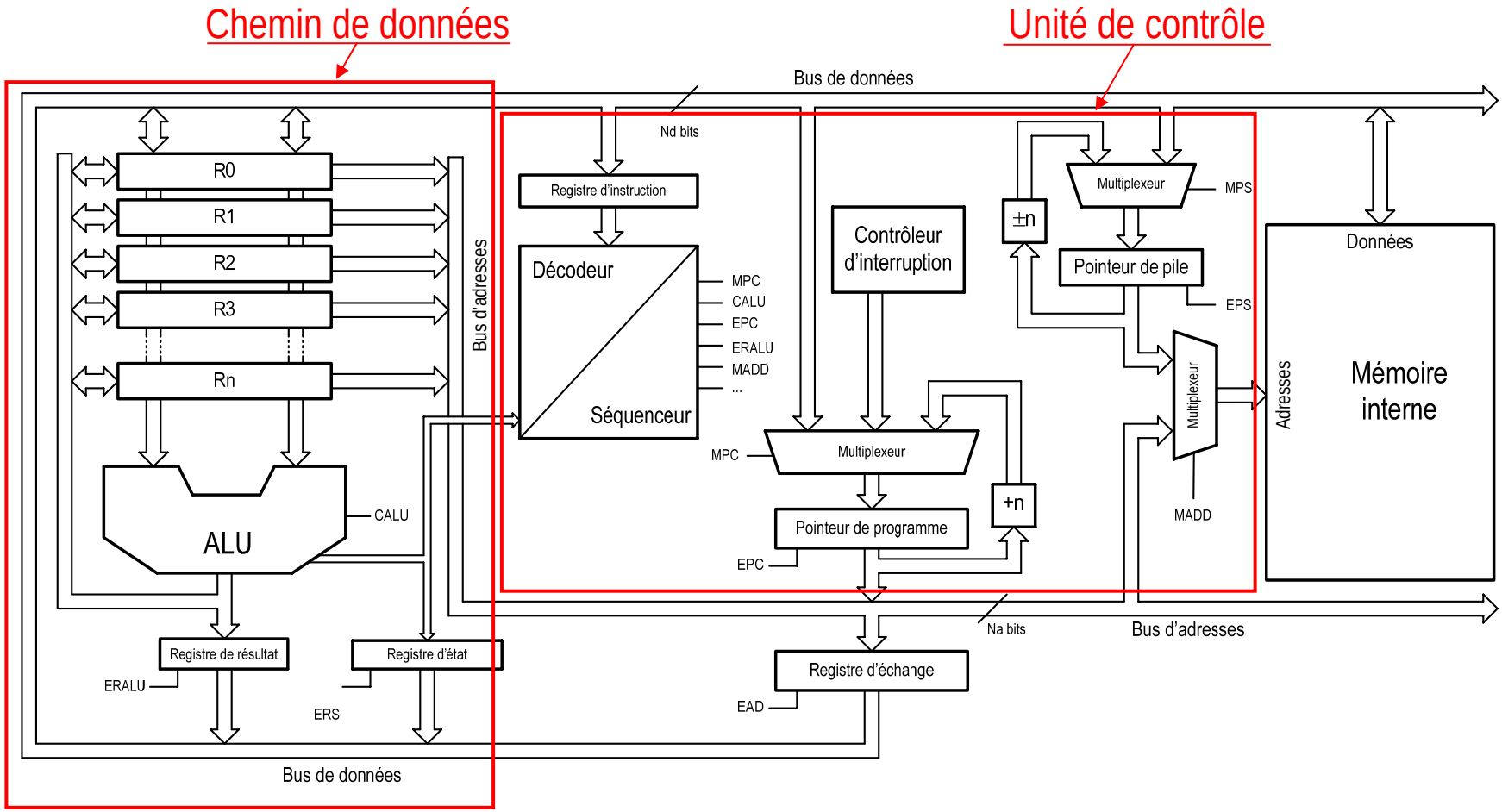
le séquenceur :

⇨ génère les signaux de commande des différents composants

l'horloge (interne ou externe) :

⇨ permet la synchronisation de tous les éléments de l'unité centrale de traitement

Architecture d'un processeur von Neumann



Cycle d'instruction

L'exécution d'un programme consiste en l'**exécution séquentielle d'instructions**. Chaque **instruction** s'exécute au cours d'un certain nombre d'**étapes** composées de **macro opérations**.

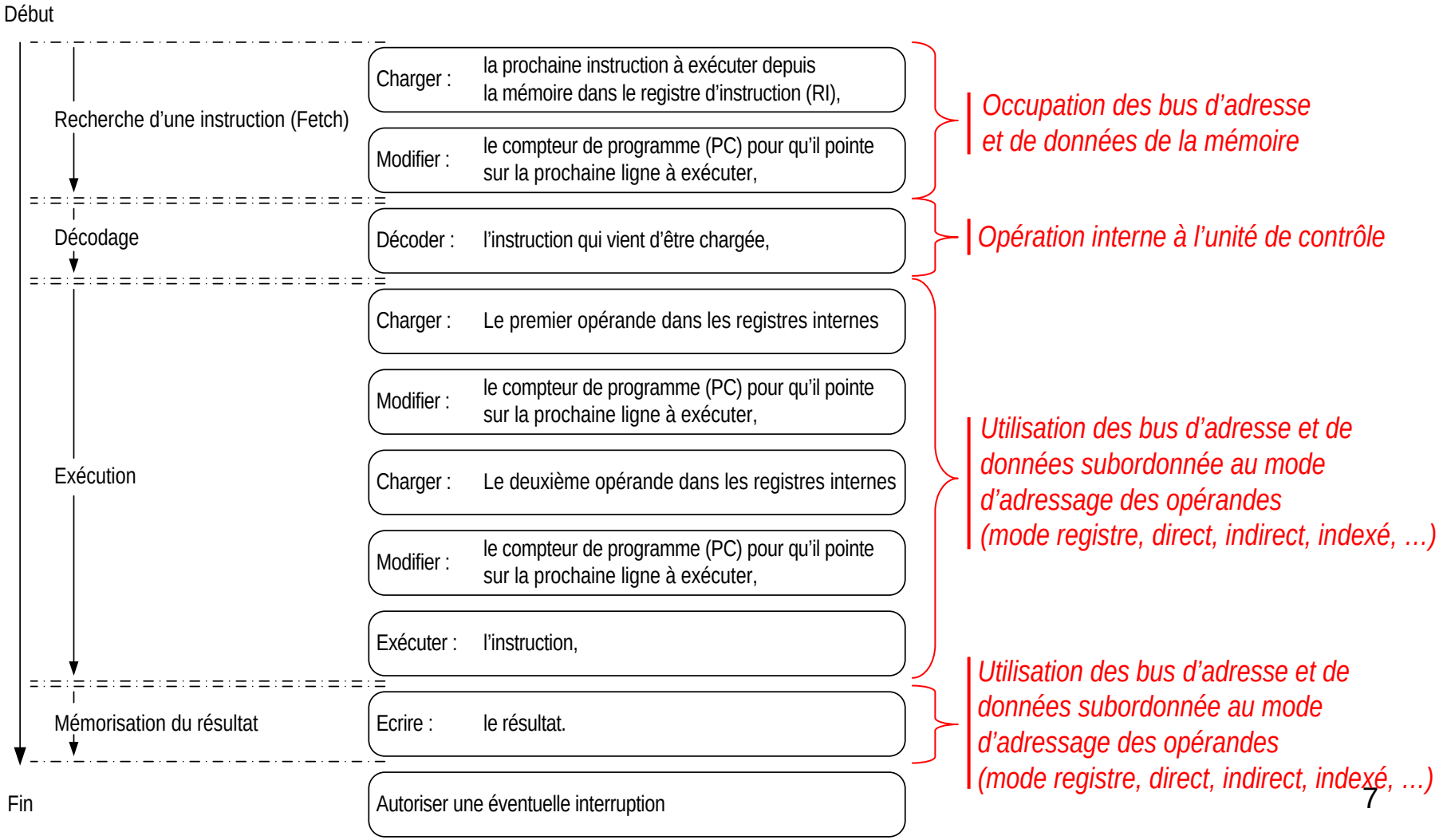
Les macro opérations les plus classiques sont :

- recherche de l'instruction,
- décodage,
- exécution (recherche d'opérandes y compris),
- écriture du résultat

L'exécution de chaque **macro opérations** requiert plusieurs opérations plus courtes, les **micro-opérations**.

Les micro-opérations représentent les **opérations fonctionnelles** d'un processeur

Etapes d'exécution d'une instruction



Exemple : instruction d'addition

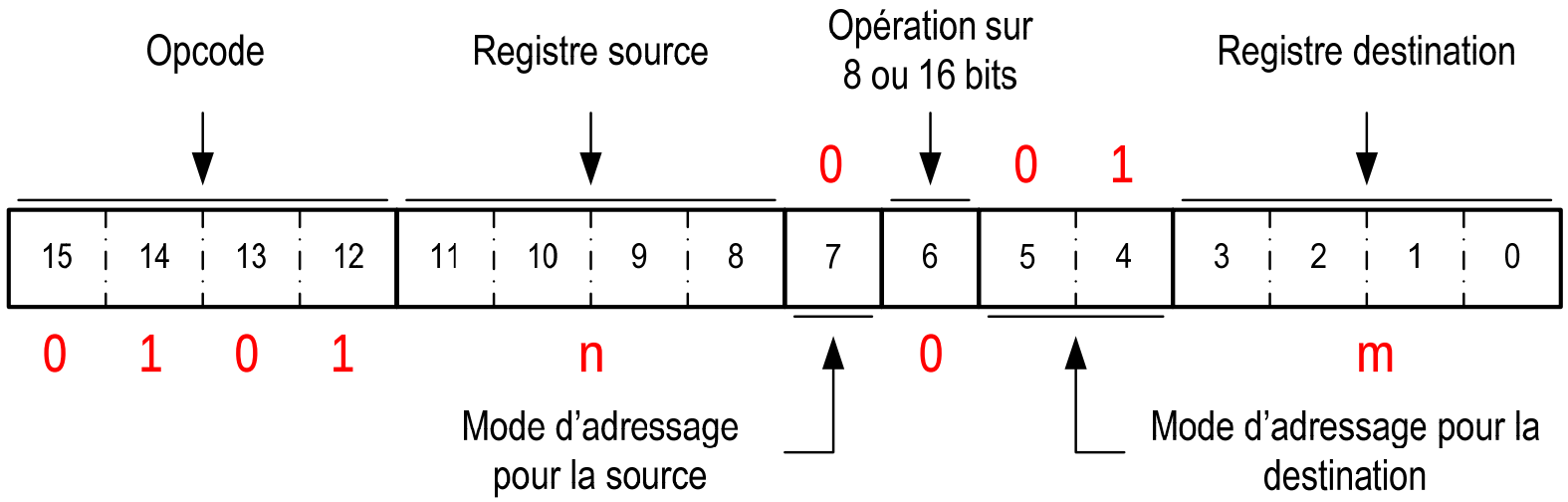
ADD @Rn,Rm ; source+destination → destination : [Rn]+Rm → Rm

registre de source (2ème opérande) et de destination (résultat)

registre de source (adresse de la 1ère opérande)

Type d'opération a effectuer (addition 16 bits + 16 bits)

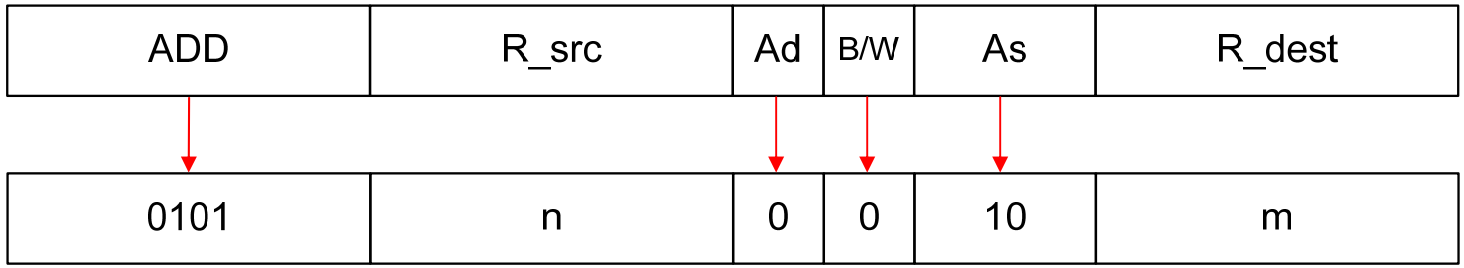
Le code machine correspondant à cette instruction tient sur 16 bits



Exemple : instruction d'addition

Rn et Rm étant deux registres internes du processeur, l'instruction consiste donc à additionner le contenu de Rm avec le contenu de l'emplacement mémoire spécifié par le contenu de Rn.
Le résultat de l'addition est implicitement copié dans Rm.
L'ancien contenu de Rm est écrasé par cette nouvelle valeur.

L'opcode de l'addition vaut 0101,
L'opération se fait sur des mots de 16 bits (B/W=0)
Le registre source est le registre n (R_rsc=n),
L'adressage sur la destination est un adressage direct (Ad=0),
L'adressage sur la source est indirecte (As=10)
Le registre de destination est le registre m (R_dest=m).



1. Recherche de l'instruction

La recherche d'une instruction est une macro-opération qui s'exécute en début de chaque cycle d'instruction et dont le but est d'acquérir une instruction située en mémoire

(1) ⇨ le contenu du pointeur de programme est mis sur le bus d'adresse interne,

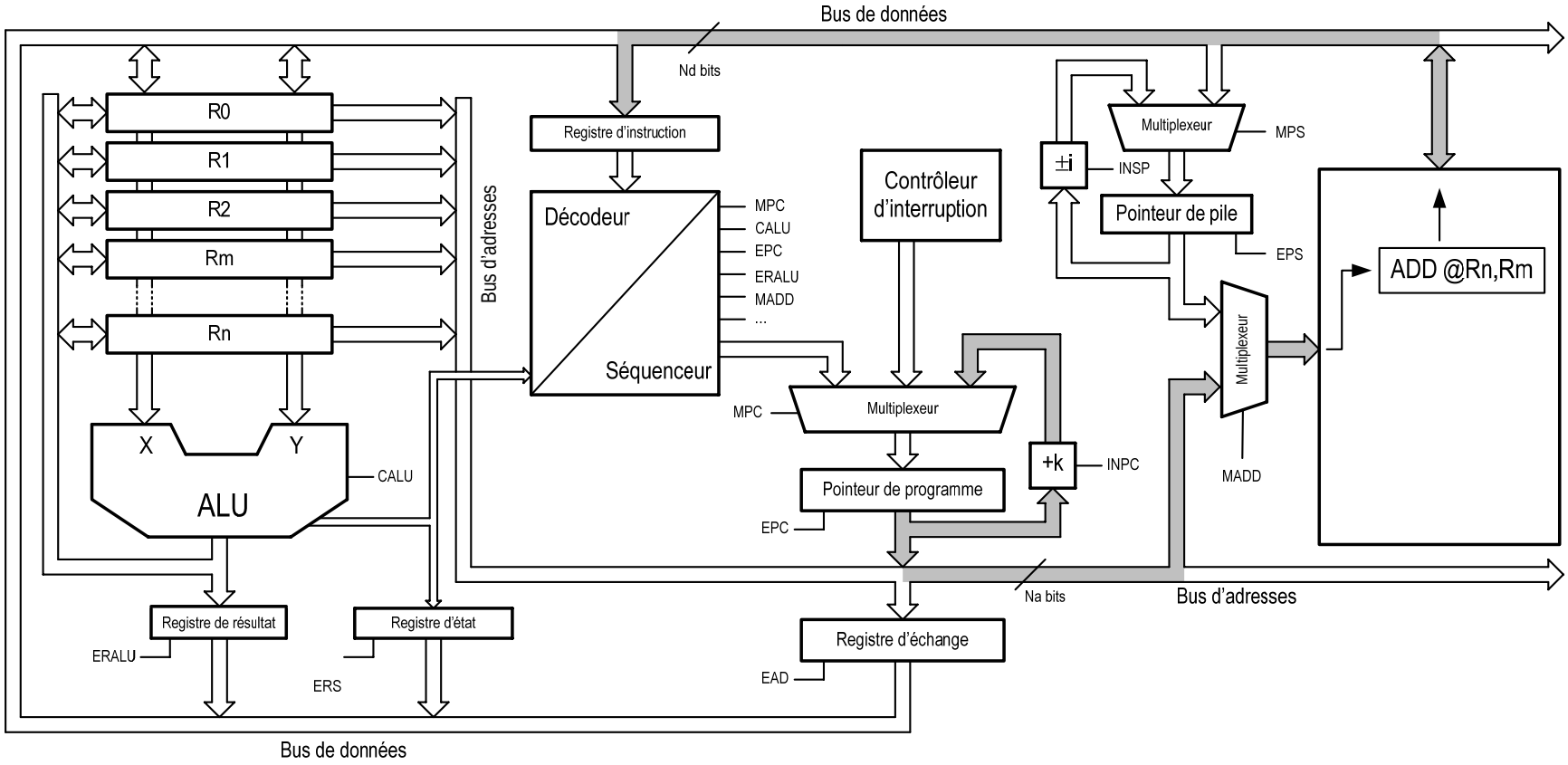
(2) ⇨ le bus d'adresse de la mémoire est lié (multiplexeur d'adresse) au bus d'adresse interne,

(3) ⇨ le contenu de la position mémoire définie par la valeur du bus d'adresse interne est mis sur le bus de données interne,

(4) ⇨ le registre d'instruction mémorise (latch) la valeur courant sur le bus de données,

(5) ⇨ le contenu du pointeur de programme est incrémenté de manière à pointer la ligne suivante, cette ligne peut être soit une nouvelle instruction soit la suite de l'instruction précédente c'est-à-dire un opérande

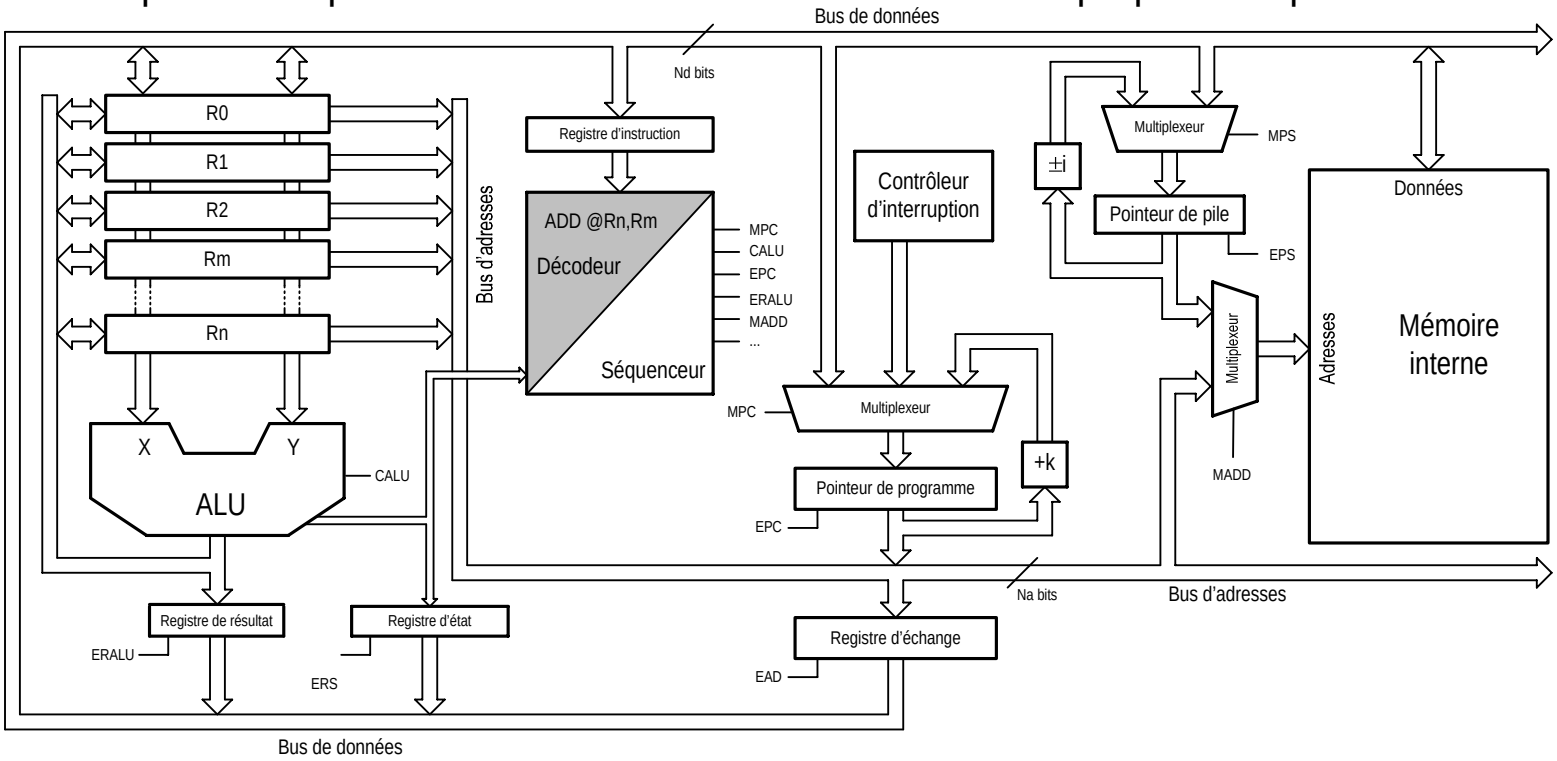
1. Recherche de l'instruction



A la fin de cette phase, l'instruction à exécuter est présente dans le registre d'instruction,

2. Décodage de l'instruction

Le code machine de l'instruction à exécuter se trouve dans le registre d'instruction RI. Le bloc décodeur peut donc analyser l'instruction afin de connaître la séquence de micro-opérations qu'il doit mettre en œuvre lors l'exécution à proprement parler



A la fin de cette phase, le type d'instruction à exécuter est connu et le séquenceur est prêt à envoyer la séquence de commande nécessaire à la réalisation de l'instruction à exécuter

3. Exécution de l'instruction

3.1 Chargement des opérandes

Les opérandes demandés par l'instruction sont cherchées dans les registres ou en mémoire

(1) ⇨ le contenu du registre Rn est mis sur le bus d'adresse,

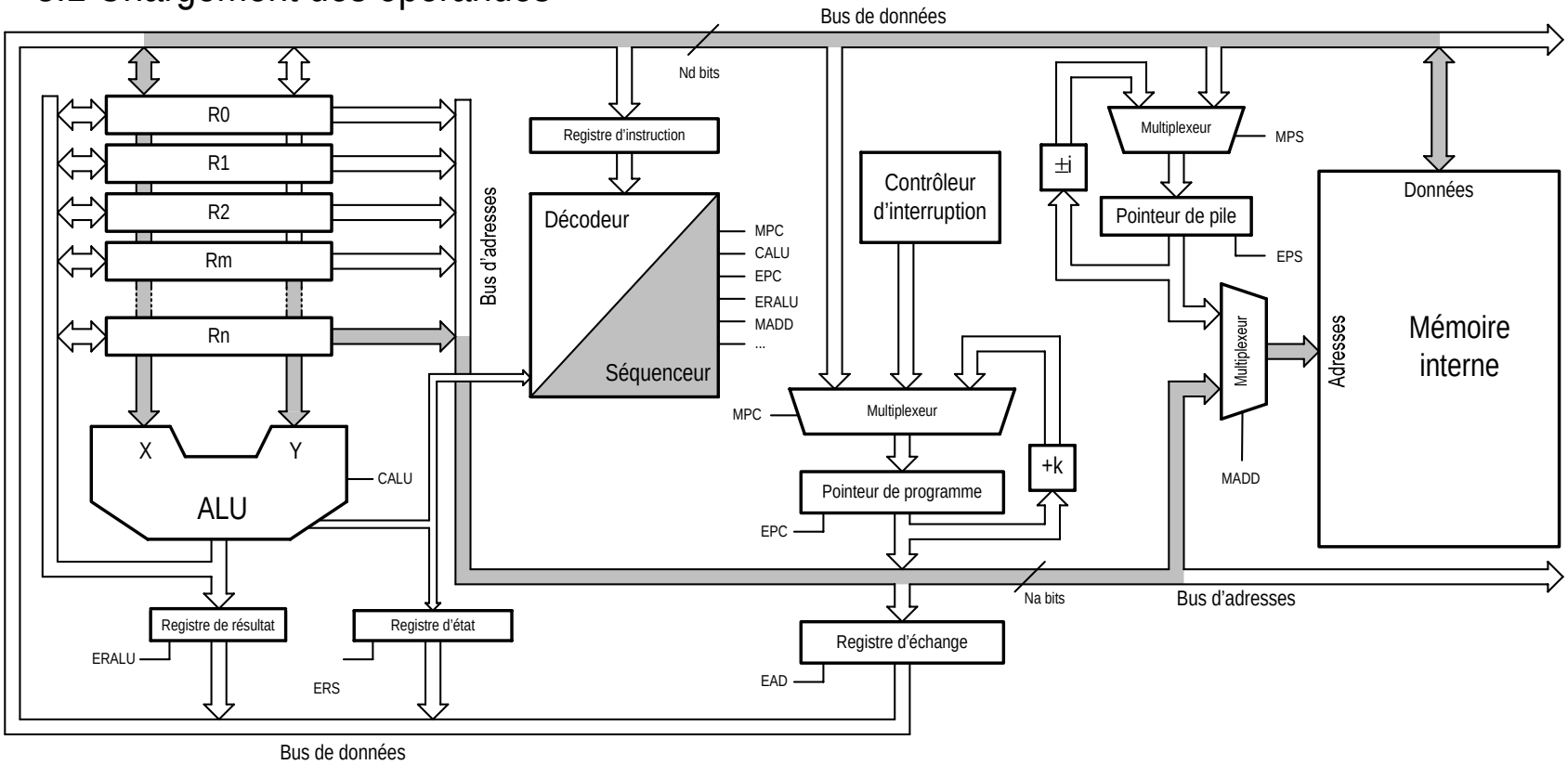
(2) ⇨ le multiplexeur du bus d'adresse de la mémoire alloue le bus d'adresse interne au bus d'adresse de la mémoire,

(3) ⇨ le contenu de la position mémoire définie par la valeur du bus d'adresse interne est mis sur le bus d'adresse interne et se retrouve sur l'entrée X de l'ALU,

(4) ⇨ le contenu du registre Rm est mis sur l'entrée Y de l'ALU.

3. Exécution de l'instruction

3.1 Chargement des opérandes



Si les deux opérandes doivent être lues en mémoire, il faut un cycle d'horloge de plus
A la fin de cette phase, le séquenceur est prêt à envoyer la séquence de commande nécessaire à la réalisation de l'instruction à exécuter

3. Exécution de l'instruction

3.2 Exécution de l'opération arithmétique

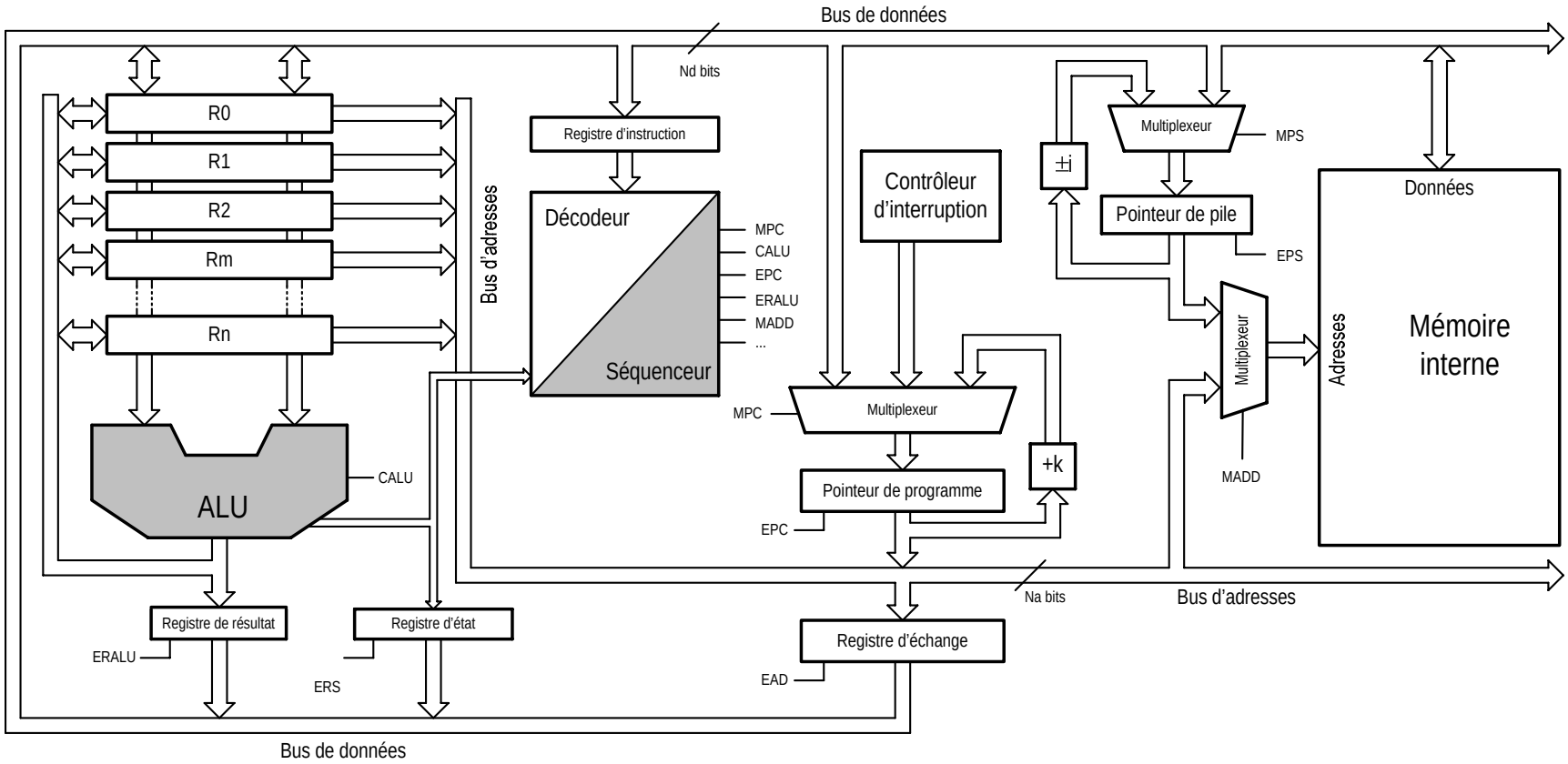
Une fois l'instruction lue et décodée, la phase suivante consiste à l'exécuter.
Pour un microcontrôleur dont le jeu d'instructions est composé de N instructions différentes, il existe N séquences différentes de micro-opérations.

(1) ⇨ l'ALU procède à l'addition des deux valeurs numériques présentes sur ses entrées X et Y

A la fin de cette phase, l'instruction est exécutée.
Le contenu du résultat est dans un registre interne à l'ALU, les bits d'états (C : report ,N : résultat négatif, Z : résultat nul, V : dépassement) sont mis à jour.
Le séquenceur peut donc commander l'écriture du résultat dans Rm (registre de destination)

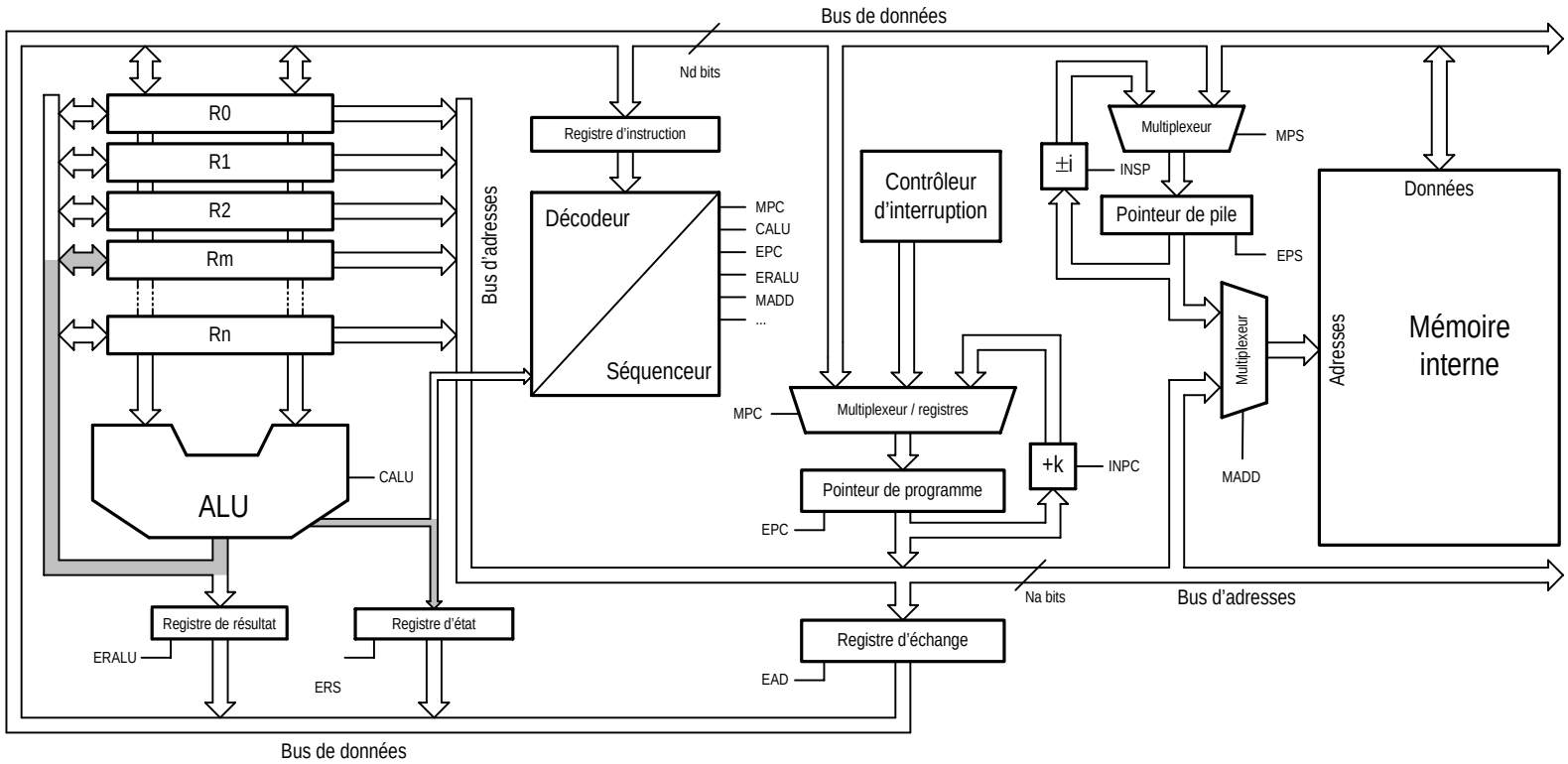
3. Exécution de l'instruction

3.2 Exécution de l'opération arithmétique



4. Ecriture du résultat (write back)

Le résultat contenu dans un registre interne de l'ALU est placé sur un bus interne au bloc de registres puis chargé dans le registre de destination Rm. Dans le même temps, les flags de l'ALU sont mis a jour et envoyé dans le registre d'état



La fin de cette phase correspond à la fin du cycle de l'instruction ADD @Rn,Rm.

Le séquenceur

Le séquenceur permet de réaliser une suite de microcommandes selon un ordre bien défini. A chaque phase d'une instruction (recherche, décodage, exécution et écriture du résultat) correspond un grand nombre de microcommandes dont la chronologie d'apparition est primordiale

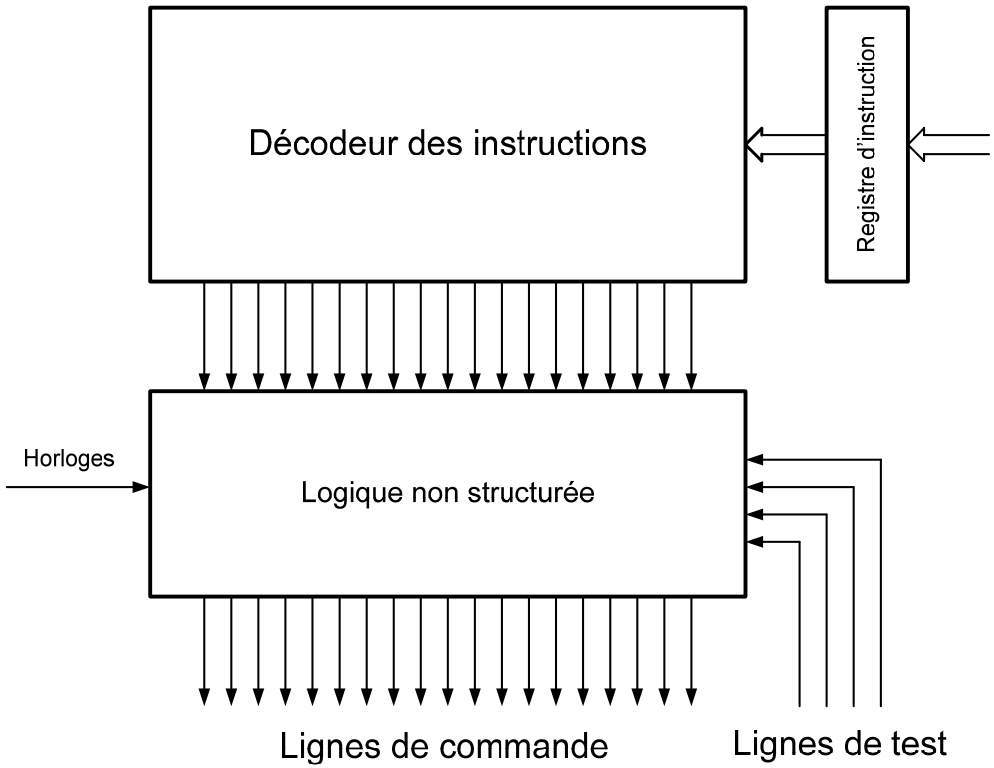
Le **séquenceur** d'un microprocesseur peut être **câblé** ou **micro programmé**.

⇨ Les **séquenceurs microprogrammés** sont mieux adaptés à l'exécution d'instructions complexes qui nécessite un nombre variables de cycles d'horloge. Cette technique est très utilisée par les **microprocesseurs CISC**.

⇨ Les **séquenceurs câblés** sont plus adaptées à l'exécution des instructions simples utilisant un nombre fixe de cycles d'horloges. Ils sont utilisés dans la conception des **microprocesseurs RISC**.

Le séquenceur câblé

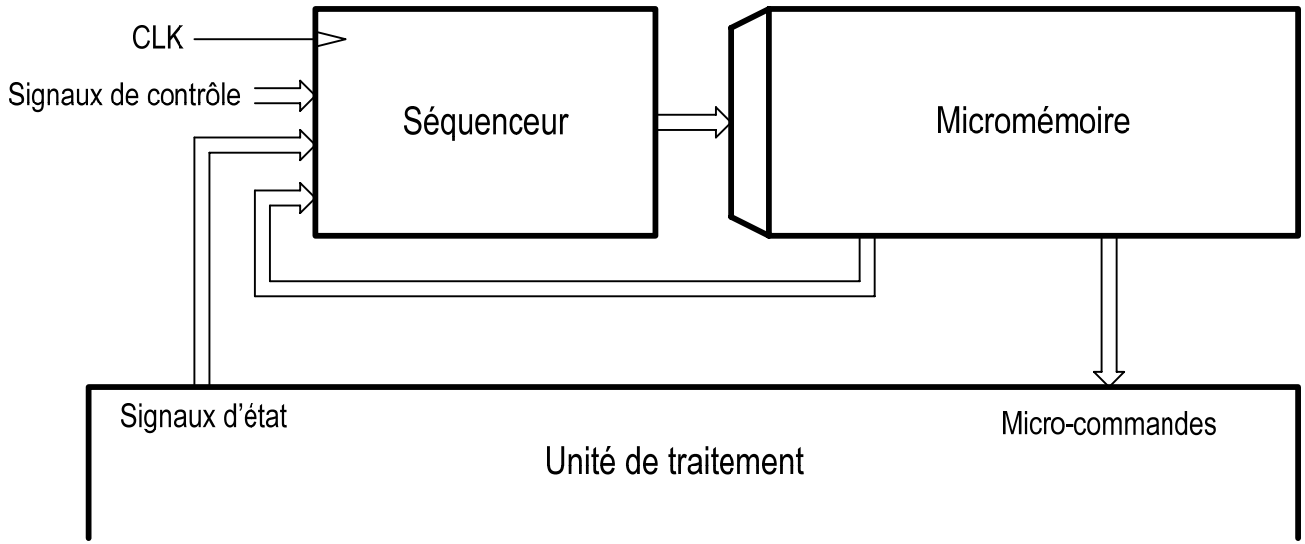
Un séquenceur câblé est conçu autour d'un bloc qui décode les instructions. Des chaînes de portes logiques sont utilisées pour générer les commandes du chemin de données à partir des sorties du décodeur combinées avec des signaux temporels représentant les différents instants de l'exécution d'une instruction.



Le séquenceur microprogrammé

Une unité de contrôle microprogrammée est composée de deux parties:

- une **micromémoire**, chargée de stocker les micro-instructions et de générer à chaque cycle d'horloge les micro-ordres contrôlant les micro-opérations du processeur
- un **séquenceur**, chargée de générer à chaque cycle d'horloge l'adresse de la micro-instruction à chercher dans la micromémoire



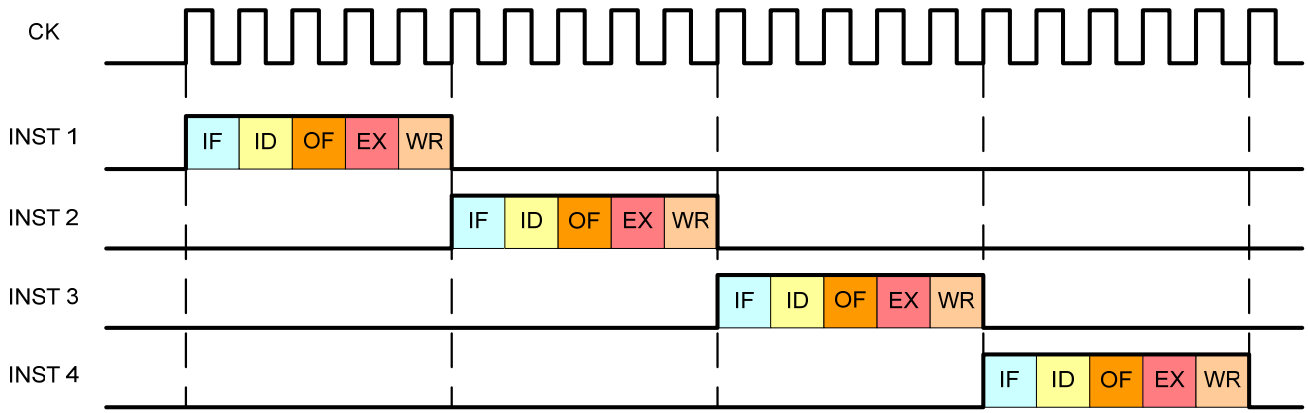
Exécution - Timing

L'exécution d'une instruction comporte au plus 5 phases:

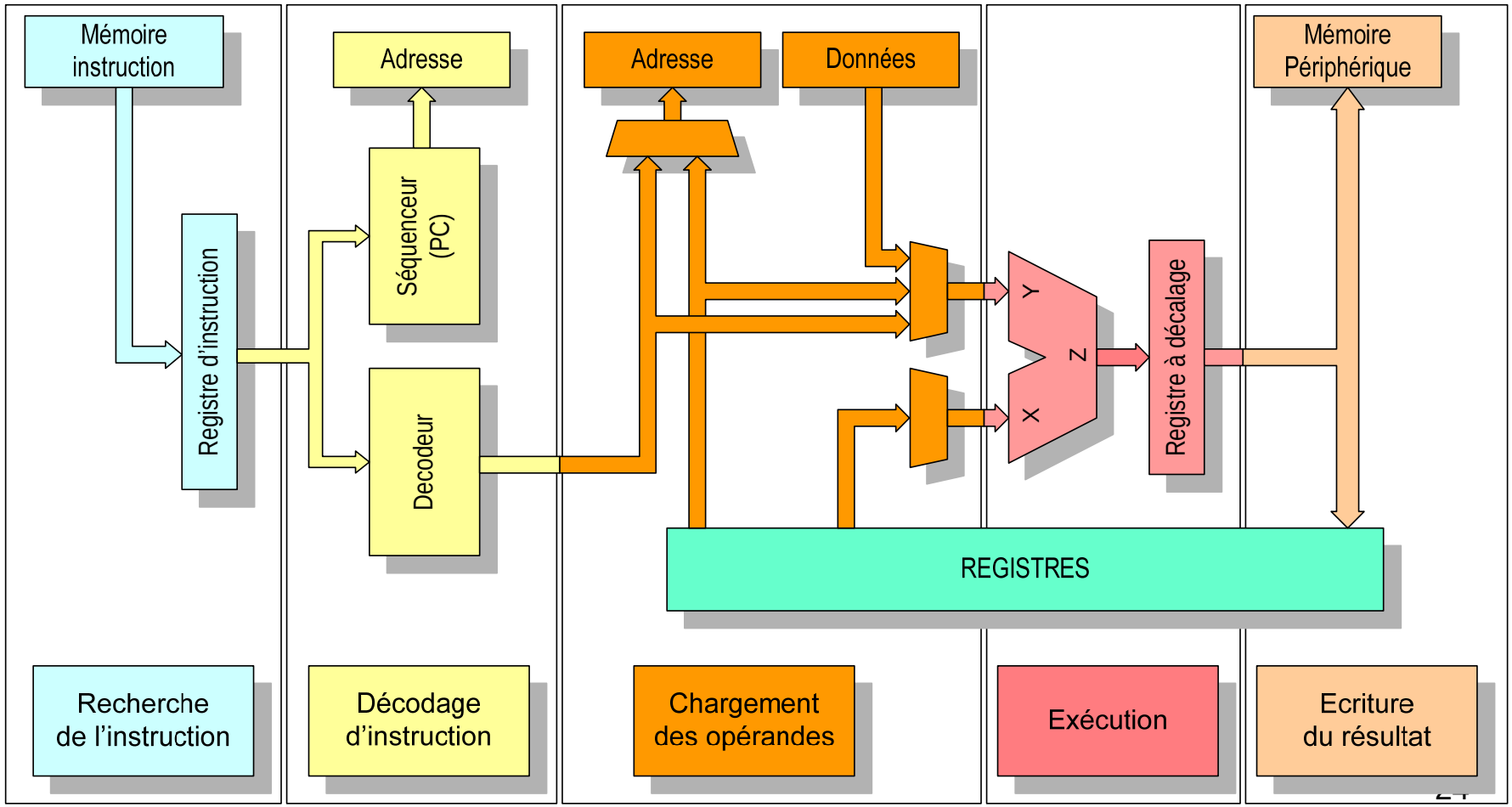
- la recherche et le chargement de l'instruction (*instruction fetch IF*)
- le décodage de l'instruction (*instruction decode ID*)
- le chargement des opérandes (*operandes fetch OF*)
- l'exécution de l'instruction (*execute* ou EX)
- la sauvegarde du résultat (*write back WB*)

Décomposition de l'exécution en deux phases

Toutes les phases d'exécution n'ont pas la même durée.

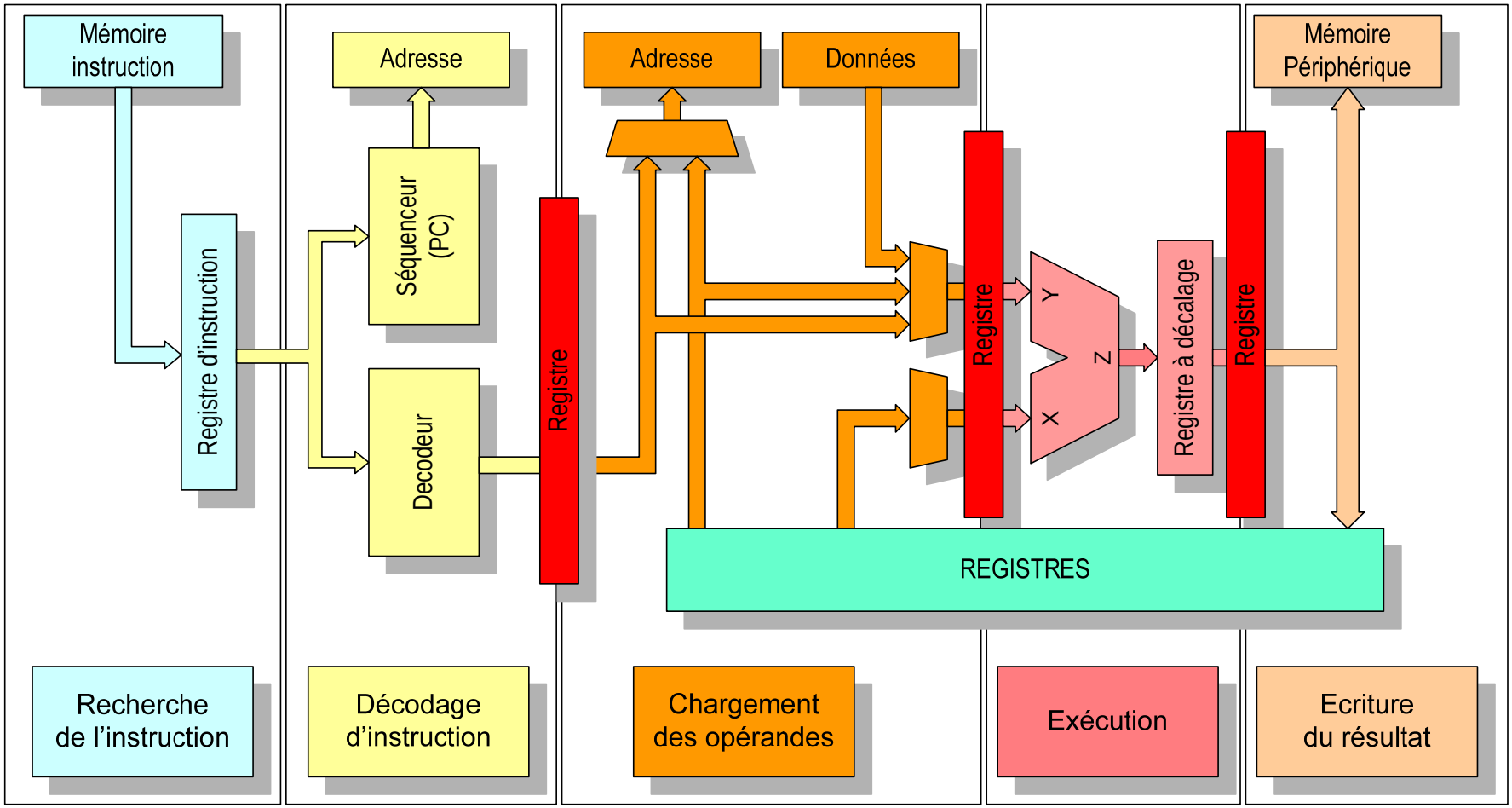


Exécution - Timing

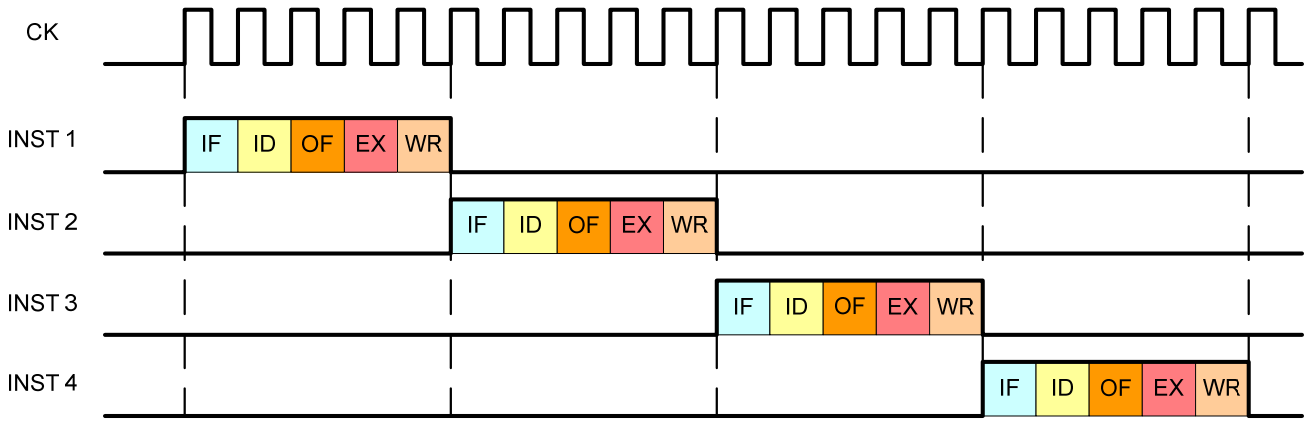


Exécution - Timing

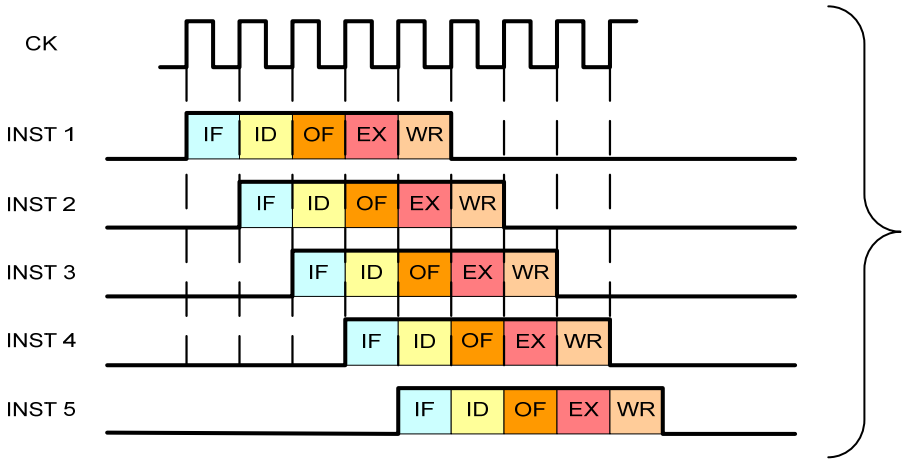
Les étages doivent être séparés par des registres (ou des *latches*).



Pipelining



Pipelining: l'exécution de chaque instruction est divisée en étages.



Flux d'instructions plus élevé

Définition du timing

Latence (*latency* ou *flow-through time*) d'un pipeline: temps (en nombre de coups d'horloge) nécessaire pour produire le premier résultat.

En pratique, la latence d'un pipeline correspond à sa profondeur, c'est-à-dire, au nombre d'étages.

Bande passante (*throughput*) d'un pipeline: nombre d'instructions par cycle d'horloge.

En pratique, pour la plupart des pipelines, la bande passante est d'une instruction par cycle d'horloge.

Définition du timing

Le temps d'exécution d'un programme sur un processeur sans pipeline est donné par:

Texec = # instructions x # cycles par instruction x Tcycle

Pour un processeur avec pipeline, le temps d'exécution devient:

Texec = ((# instructions-1) x (1 / Bande passante) + Latence) x Tcycle

Comme pour la plupart des processeurs avec pipeline la bande passante est d'une instruction par cycle d'horloge:

Texec = ((# instructions-1) + Latence) x Tcycle

Aléas

La présence d'un pipeline (et donc le partage de l'exécution d'une instruction en plusieurs étages) introduit des aléas:

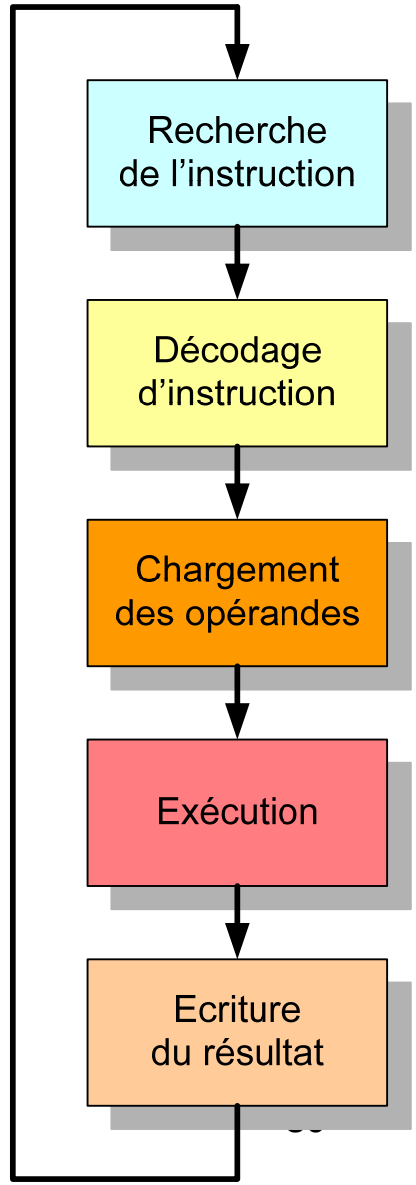
- aléas de structure : l'implémentation empêche une certaine combinaison d'opérations (lorsque des ressources sont partagées entre plusieurs étages)
- aléas de données : le résultat d'une opération dépend de celui d'une opération précédente qui n'est pas encore terminée
- aléas de contrôle : l'exécution d'un saut introduit un délai lorsque le saut est conditionnel et dépend donc du fanion généré par une opération précédente

Les aléas de structure peuvent être éliminés en agissant sur l'architecture du processeur lors de sa conception. Par contre, les aléas de données et de contrôle ne peuvent pas être éliminés.

Aléas de structures

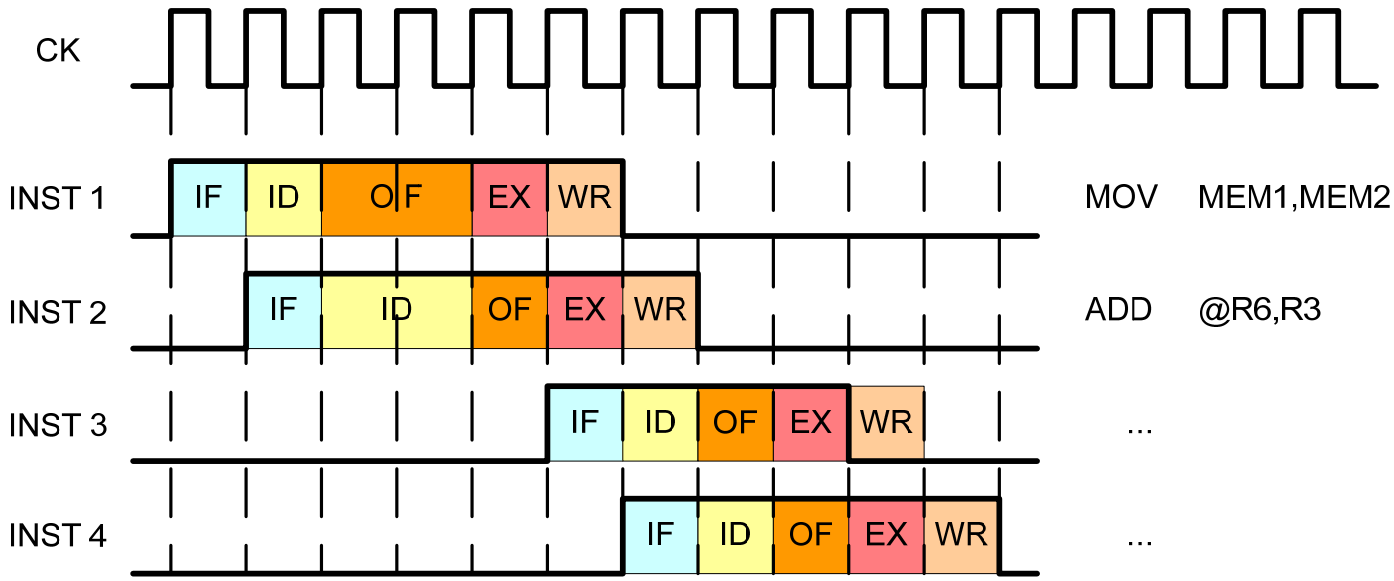
Deux instructions contiguës i et j peuvent présenter des aléas de structure (utilisation des mêmes bus par exemple).

```
i : MOV MEM1, MEM2      {MEM1→MEM2}
j : ADD  @R6, R3        {[R6]+R3→R3}
```



Aléas de structure

Une solution relativement simple pour traiter les aléas de structure est de retarder l'exécution des instructions concernées (*pipeline stall*) en les bloquant là ou il y a des conflits matériel.



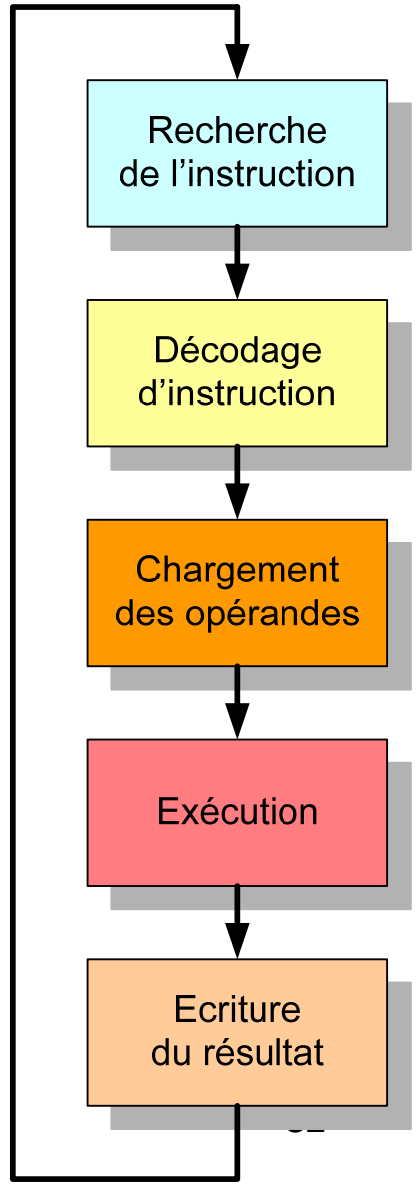
Les aléas de structure sont dépendant de l'architecture du processeur.

Aléas de données

Deux instructions contiguës i et j peuvent présenter des aléas de données (dépendances entres les opérandes des deux instructions). En particulier, pour notre pipeline:
RAW (*read-after-write*): j essaie de lire un registre avant que i ne l'ait modifié

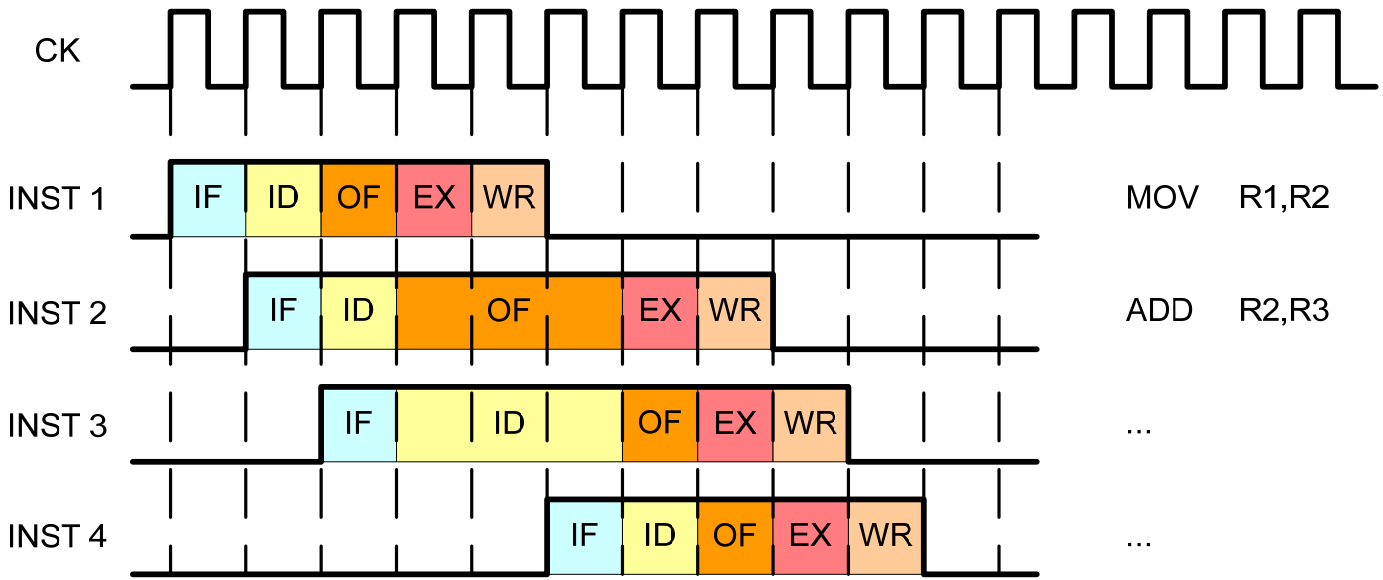
```
i: MOV R1, R2    {R1→R2}
j: ADD  R2, R3   {R2+R3→R3}
```

D'autres aléas sont possibles entre deux instructions (write-after-read, write-after-write), mais ils ne sont pas possibles dans un pipeline "standard".



Aléas de données

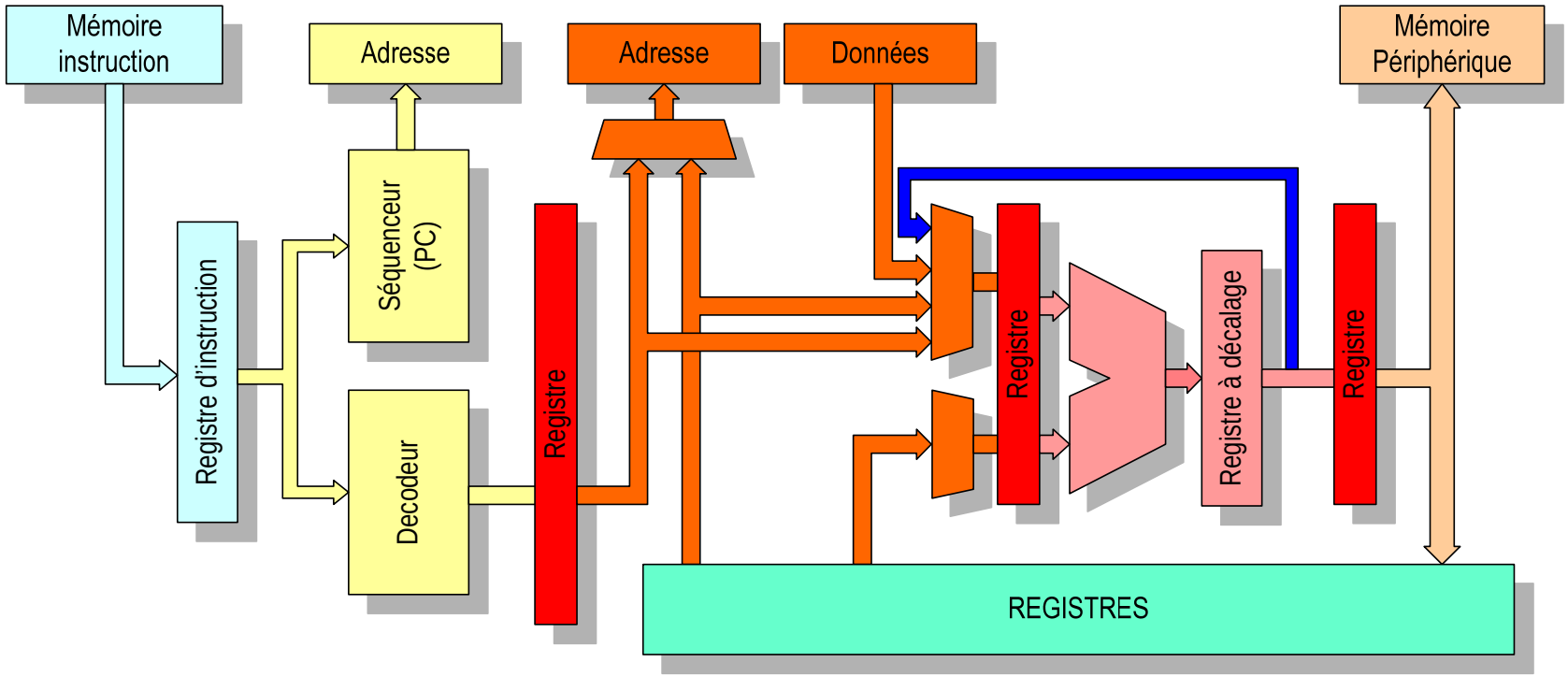
Une solution relativement simple pour traiter les aléas de données est de retarder l'exécution des instructions concernées (*pipeline stall*) en les bloquant à l'étage de chargement des opérandes.



Il faut toutefois remarquer que, les aléas de données (et surtout les RAW) étant relativement fréquents, le délai introduit par cette méthode est loin d'être négligeable.

Aléas de données

La fréquence élevée d'aléas de données peut justifier l'introduction de matériel supplémentaire. Notamment, la méthode appelée *forwarding* ou *bypassing* introduit une connexion directe entre la sortie de l'étage d'exécution et l'étage de chargement des opérandes, permettant au résultat d'une instruction d'être un opérande de l'instruction suivante.



Cette méthode est coûteuse en logique de contrôle supplémentaire.

Aléas de données

Une troisième solution peut être fournie par le compilateur, qui peut changer l'ordre d'exécution des instructions de façon à éliminer les aléas. Si nécessaire, les instructions intercalées peuvent être des NOP.

```
ADD   R5,R6
MOV   R6,&MEM
SUB    R4,R3
…
```

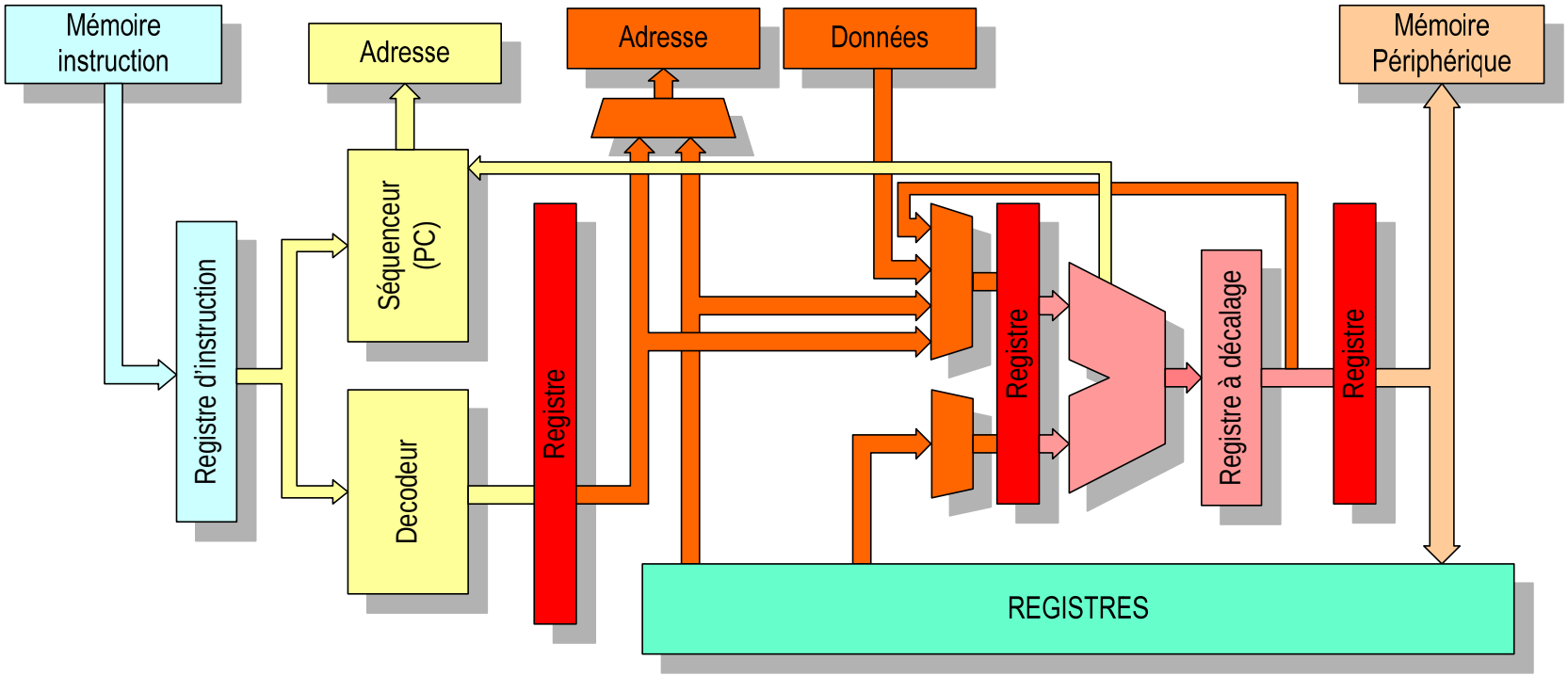
```
ADD   R5,R6
SUB   R4,R3
NOP
MOV   R6,&MEM
…
```

Malheureusement, l'utilité de cette méthode est limitée:

- ⇨ le compilateur n'est pas toujours en mesure de détecter les aléas (par exemple, si les aléas concernent des pointeurs).
- ⇨ le nombre d'instructions à intercaler dépend de la structure (nombre d'étages) du pipeline;
- ⇨ la complexité du compilateur en est fortement augmentée.

Aléas de contrôle

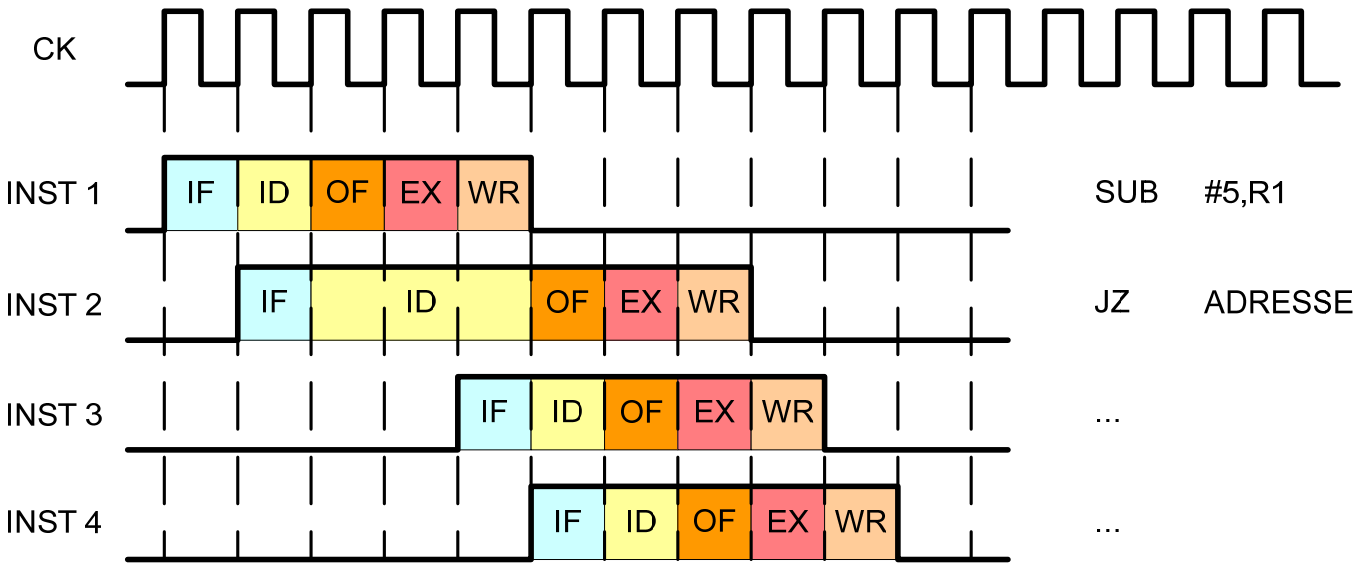
La présence d'un pipeline introduit des complications lors de l'exécution d'un saut conditionnel : le flag qui permet au séquenceur de déterminer si le saut est pris ou non n'est généré que lorsque l'instruction précédente arrive à l'étage d'exécution. L'étage de décodage de l'instruction n'est pas en mesure de calculer l'adresse de l'instruction suivante avant de connaître la valeur du fanion.



Aléas de contrôle

Comme pour les aléas de données, une solution possible aux aléas de contrôle est de retarder l'exécution de l'instruction de saut (*pipeline stall*).

Cette opération introduit un délai qui est relativement important cause du grand nombre d'instructions de saut dans un programme (les sauts conditionnels représentent environ 11-17% des instructions dans un programme).



Interruption

La présence d'un pipeline complique le traitement des interruptions.

Lors du déclenchement d'une interruption non-masquable, la routine de traitement doit parfois être lancée immédiatement.
Le pipeline contiendra alors des instructions partiellement exécutées.

La seule solution efficace est de "vider" le pipeline. Cette opération peut toutefois compliquer le redémarrage du programme après le traitement de l'interruption (par exemple, il faut retrouver l'adresse de la première instruction dont l'exécution n'a pas été achevée).