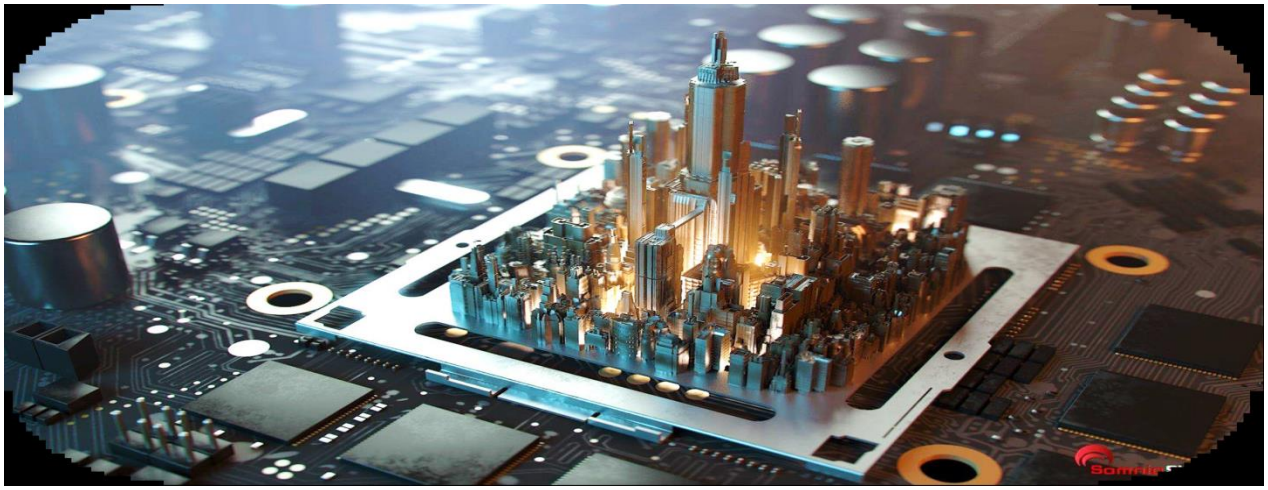
Faculté des sciences exactess
Département d'informatique

Cours 2: Architecture interne des processeurs



NIVEAU: INGÉNIEUR I ANNÉE
PRÉSENTÉE PAR: DR. SAAD NARIMANE

2024/2025

Introduction

- Le **microprocesseur** exécute les fonctions d'unité centrale « principale » d'ordinateur **(CU),**
- **CPU** : l'unité de **traitement** de l'ordinateur ,**exécute les instructions** du programme et **les données** à traiter,
- Nous expliquerons son **rôle** ainsi que ses **éléments internes** et de leur **fonction**
- Nous allons étudier la programmation des différentes conventions relatives à l'écriture des **programmes en langage d'assemblage**

I. Processeur

- CPU (*Central Processing Unit*), c'est un **circuit électronique** cadencé au rythme d'une **horloge** interne, elle envoie des impulsions, appelées « **top** ».

- Les **avancées technologiques** se concentrent sur CPU, qui travaille toujours **plus vite** et effectue des **opérations de plus en plus compliquées**.

I.1. Les principales caractéristiques d'un processeur

- ✓ **Le format des mots de données** : 8 bits, 16 bits, etc.
- ✓ **La taille de l'espace adressable** : dépend du nombre de bits d'adresses (ex: 65536 emplacements pour 16 bits).
- ✓ **La puissance de traitement** : caractérisée par le nombre d'instructions capable de traiter par seconde, **CPI** (**Cycle Per Instruction**) ou en **MIPS** (**Millions d'Instructions Par Seconde**)
- ✓ **Le jeu d'instructions** :
- Etendu (**CISC**)
- Réduit (**RISC**)

- Le microprocesseur exécute le **programme** « suite d'instructions »
- Une **instruction** est une opération **SIMPLE** sur **un (ou plusieurs) mot(s) de données** :

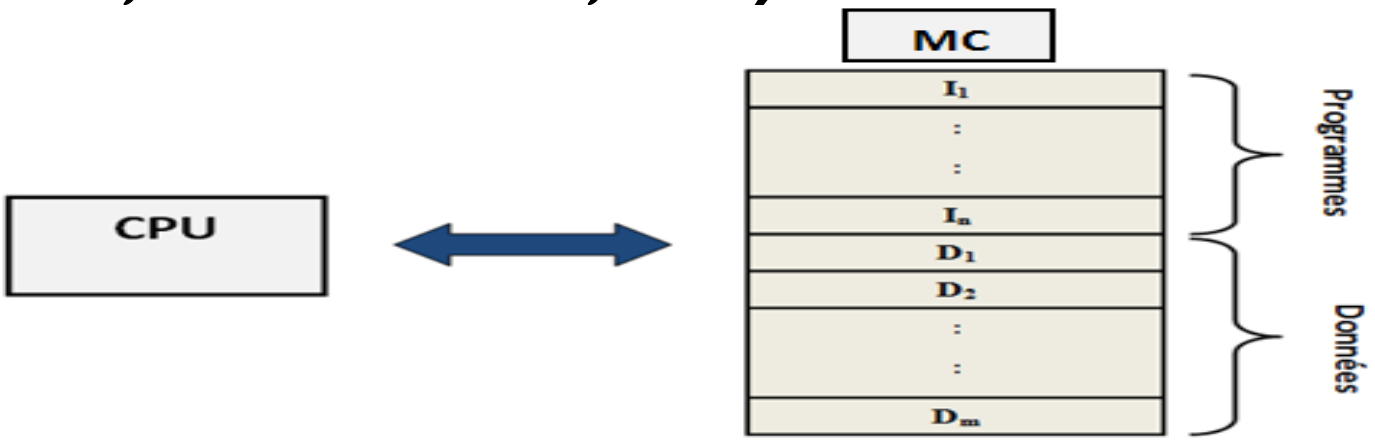
✓ **Lecture (LOAD)**

✓ **Ecriture (STORE) en mémoire**

✓ **Opération logique (ET, OU, ..)**

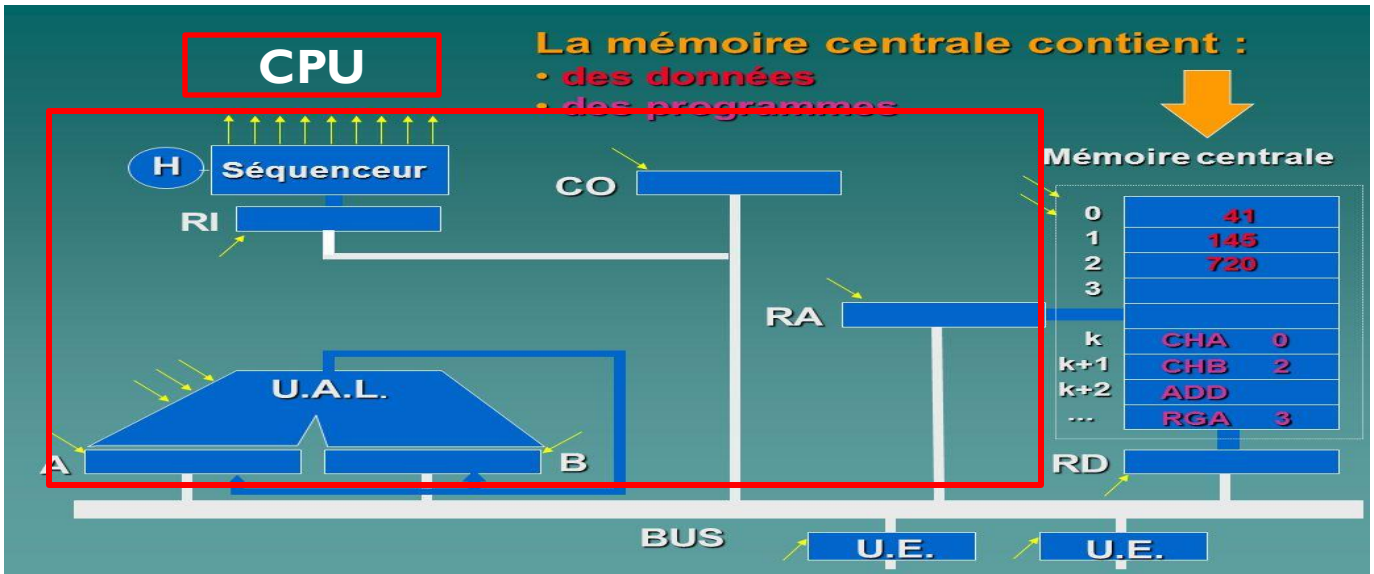
✓ **Opération arithmétique (addition, soustraction, etc.)**

```
Assembler MASM DOS.asm
1  .model small
2  .stack
3  .data
4      message  db "Hello World", "$"
5  .code
6      main   proc
7          mov   ax,seg message
8          mov   ds,ax
9          mov   ah,09
10         lea   dx,message
11         int   21h
12
13         mov   ax,4c00h
14         int   21h
15     main   endp
16 end main
17
```



1.2. Rôle du processeur

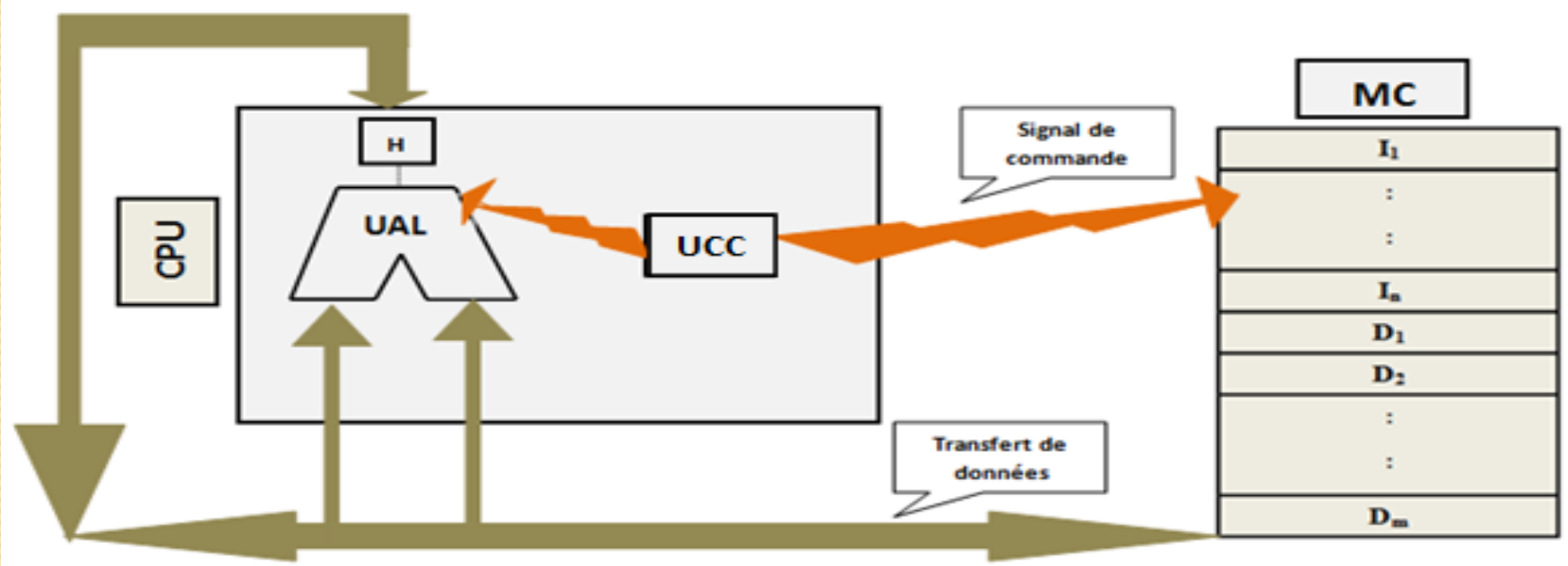
- **Exécuter les instructions** composant le programme.
- Les **calculs mathématiques** et des **transferts de données internes et externes**.
- **Prendre des décisions**
- **Manipuler** des informations numériques(**codées sous forme binaire)**
- **Exécuter** les instructions stockées en mémoire.



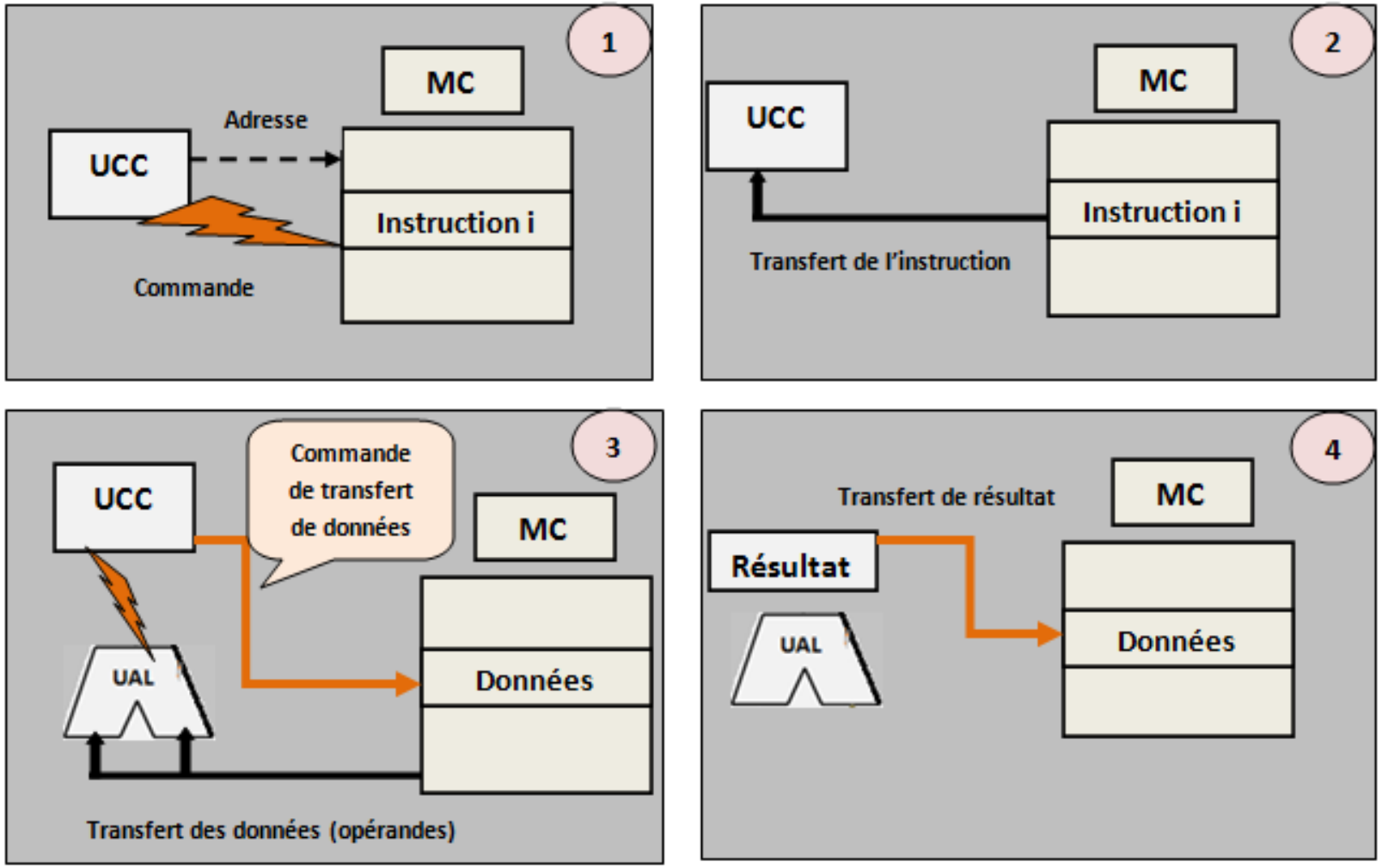
I.3. Les composants internes d'un processeur et leur fonction

CPU composée de 02 unités fonctionnelles:

- **1.L'unité arithmétique et logique (UAL):** s'occupe de toutes les **opérations arithmétiques et logiques**
- **2. L'unité de contrôle et de commande (UCC) :** **commande l'exécution** de toutes les opérations (L'UAL, mémoire et l'unité d'entrée /sortie), et **le contrôle** de leurs déroulement.



- *1.3. Fonctionnement de l'ensemble UAL et UCC*
- **Programme en cours d'exécution ➡ (Les instructions + les données) @ en MC**



2. Les Registres

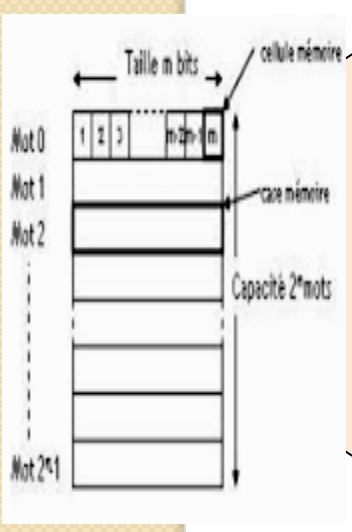
- **Cellule mémoire** la plus petite ayant une fonction particulière,
- Elle peut **mémoriser des instructions, des adresses, des résultats intermédiaires**,…etc.
- Un registre permet de mémoriser une information sur **n bits**.
- Le **nombre de registres** différent et dépend de **l'architecture de l'ordinateur** ,
- Certains **registres principaux** existent dans la plupart des machines

2.1. Caractéristiques des registres

- Un registre est une **mémoire locale** et privée **du processeur**, elle est caractérisé par :
- **Mémorise** une Information **temporaire**
- **Capacité limitée** (1 à 128bits)
- Les **informations mémorisés** peuvent être **Visibles** ou **invisibles** (accessibles par l'utilisateur ou uniquement par le processeur).

2.2. Les types de registres

- **Le registre d'adresse:** l'adresse d'un mot-mémoire
- **Le registre mot :** le contenu d'un mot-mémoire
- **Un registre de contrôle :** il contrôle le processeur ou autre.



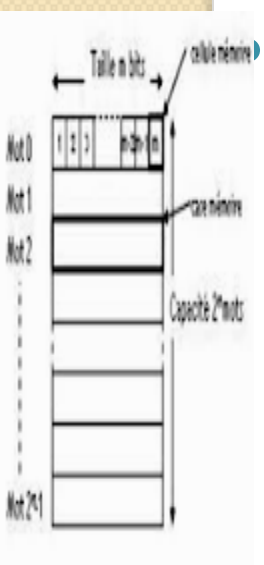
Remarque : *Un* ***registre mot*** *a la même* ***taille*** *qu'un* ***mot-mémoire****, alors qu'un* ***registre d'adresse*** *doit permettre* ***d'adresser tous les mots de la mémoire****.*

- **Les registres principaux :**
- **1. Compteur ordinal (CO):** Compter les instructions contient l'adresse mémoire de la prochaine instruction
- **2. Le registre d'instruction (RI):** Ce registre contient l'instruction en cours d'exécution « = taille mot mémoire »
- **3. L'accumulateur (ACC) :** Un registre de l'UAL, contient (le résultat de l'opération ou l'un des deux opérandes avant l'exécution « = taille mot mémoire »
- **4. Les registres généraux (R0, …, Rn):** Registres de **stockage intermédiaire** pour **les calculs**
- Exemple : MOV C,B: transfert contenu du B dans C.

- **6. Registre de base :** Ce type de registre sert **à calculer l'adresse effective**,

- **7. Le registre d'état : (PSW : Program Status Word) : registre de condition** pointer sur le début des zones mémoires), il contient des bits indicateurs : **drapeaux (ou flags**)

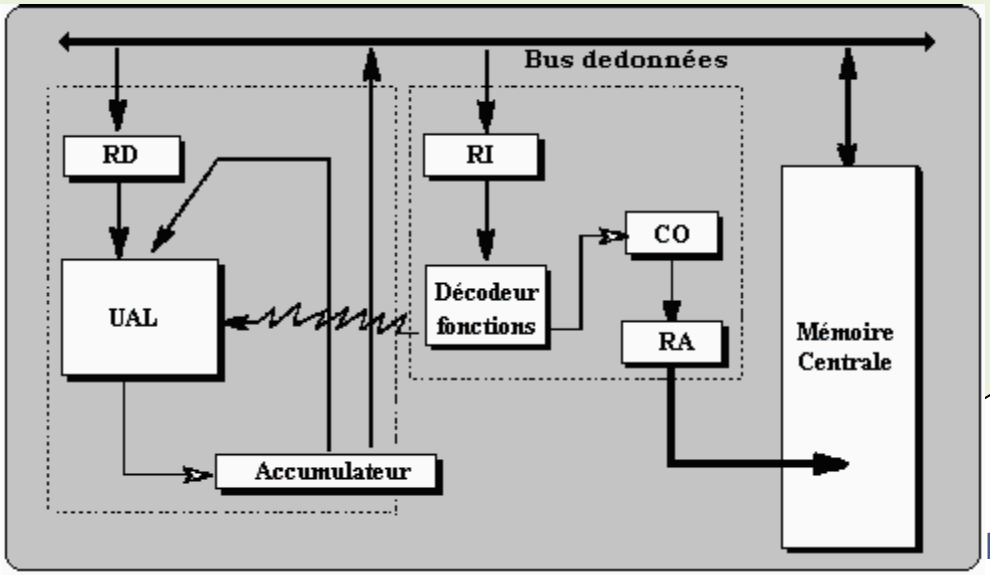
Exemple: Signe de résultat (N** pour **Négatif, P** pour **Positif)



- **8. Le registre pointeur de pile (SP : Stack pointer) :** Contient **l'adresse du sommet de la pile** en MC pour gérer une **zone de mémoire** sous forme **d'une pile (LIFO)**

Pour un traitement rapide, l'exécution d'une instruction par le processeur passe par ces étapes :

1) ***Charger*** l'***instruction*** à exécuter dans ***RI***.
2) ***Modifier*** le (**CO**) se pointe sur ***l'instruction suivante***.
3) ***Décoder*** (analyser) ***l'instruction chargée***.
4) ***Localiser*** en mémoire les **données nécessaires** à l'instruction.
5) **Charger** les **données** nécessaire **dans les registres généraux**.
6) **Exécuter l'instruction**.
7) **Stockage des résultats** à leurs destinations respectives.
8) **Recommencer** à l'étape 1



3. Unité Arithmétique et Logique (UAL)

- Réalise **opérations arithmétiques** et **logiques**,
- Possède **les registres** suivants :

✓ **L'accumulateur (ACC) :** réservé pour l'UAL **stocke le résultat** de l'opération

✓ **Les registres arithmétiques :** destinés pour les opérations arithmétiques (+, -, *, /) ou logiques (NOT, AND, OR, XOR),

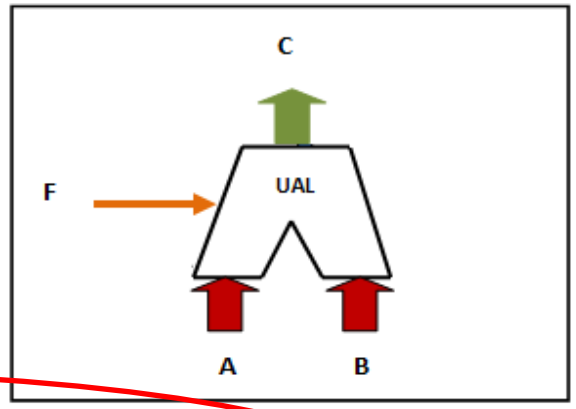
✓ **Les registres de base et d'index:** (calcul d'adresse par rapport à **une base ou index**).

✓ **Les registres généraux:** (ex : **stockage de résultats intermédiaires**).

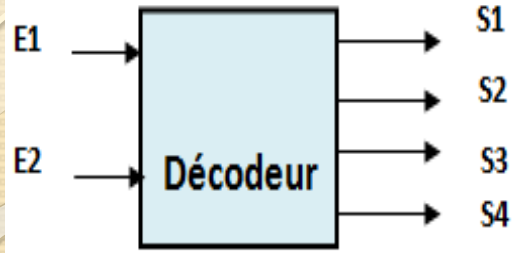
✓ **Le registre d'état PSW :** qui indique **l'état du système**

- A et B sont les opérandes (les données en entrée) et C est le résultat de l'opération, F est la fonction à exécuter.
- Pour réaliser une UAL permet de faire les opérations logiques de base (ET, OU, NON) et la somme de deux nombres binaires ⇨ quelle est la composition de l'UAL ?

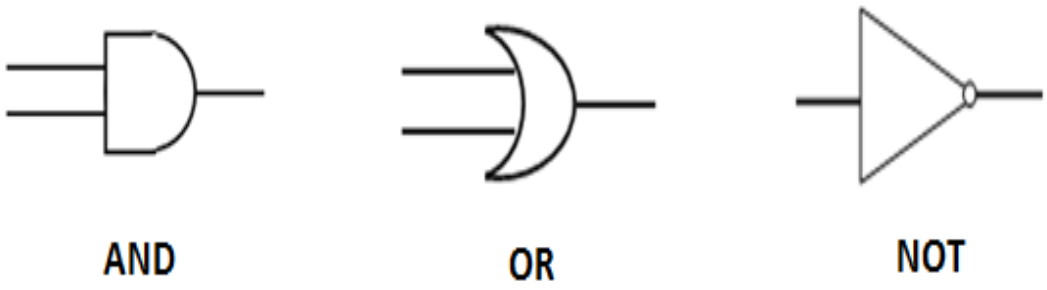
- Cette UAL traite 04 opérations : A ou B, A et B, non B, A+B
- Ce circuit sélectionne l'opération à exécuter,
- Ce circuit doit avoir 2 entrées E1, E2
- Pour effectuer 4 choix : 00, 01, 10, 11



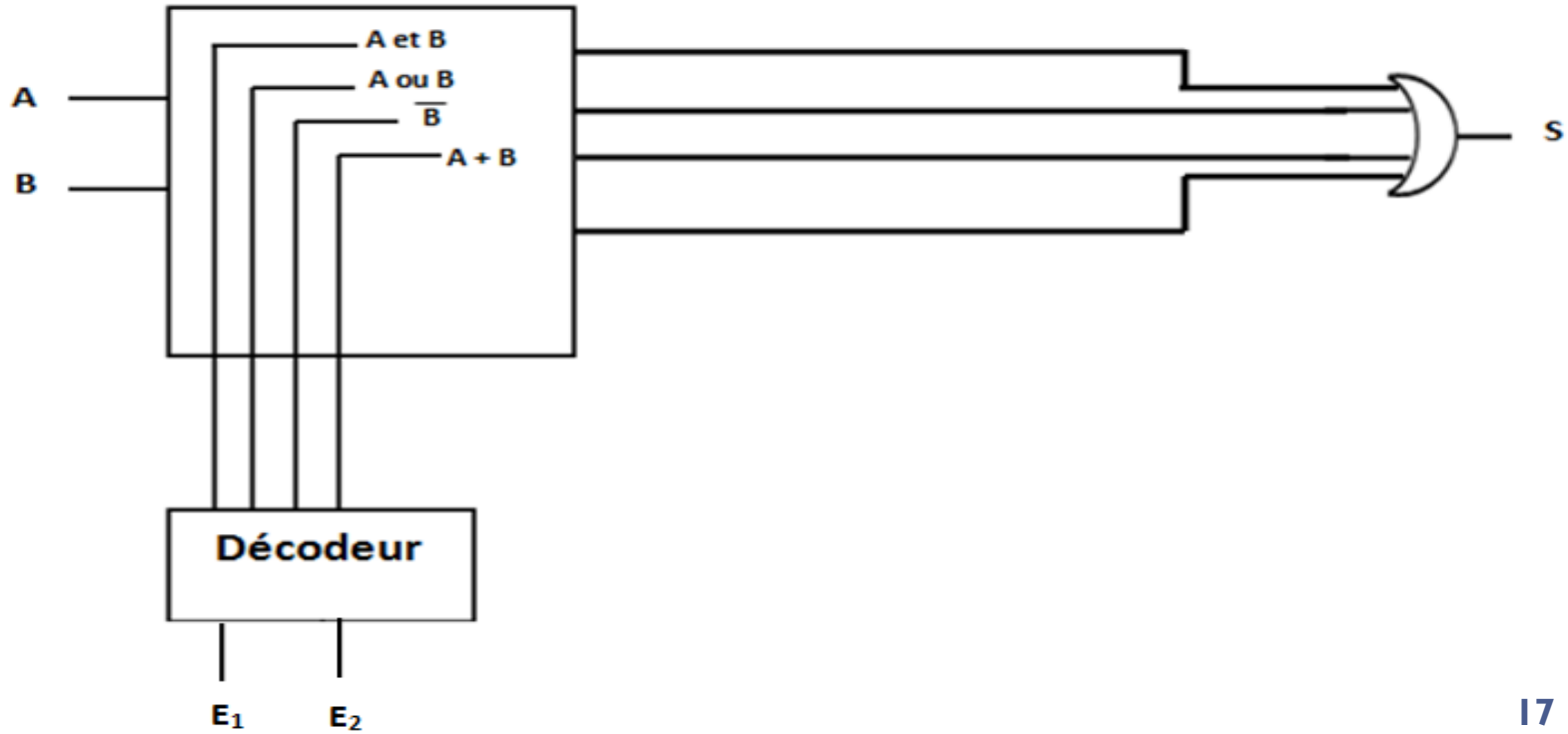
Ce circuit (UAL) : Est un décodeur à 2bits

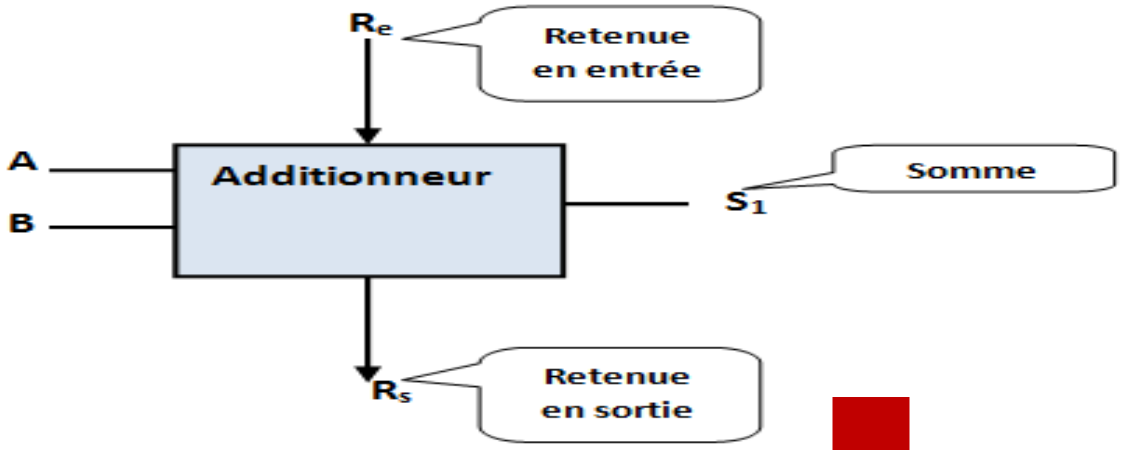


Pour réaliser les 03 opérations logiques A et B, A ou B et Non B ⇨ on a besoin de 3 portes logiques

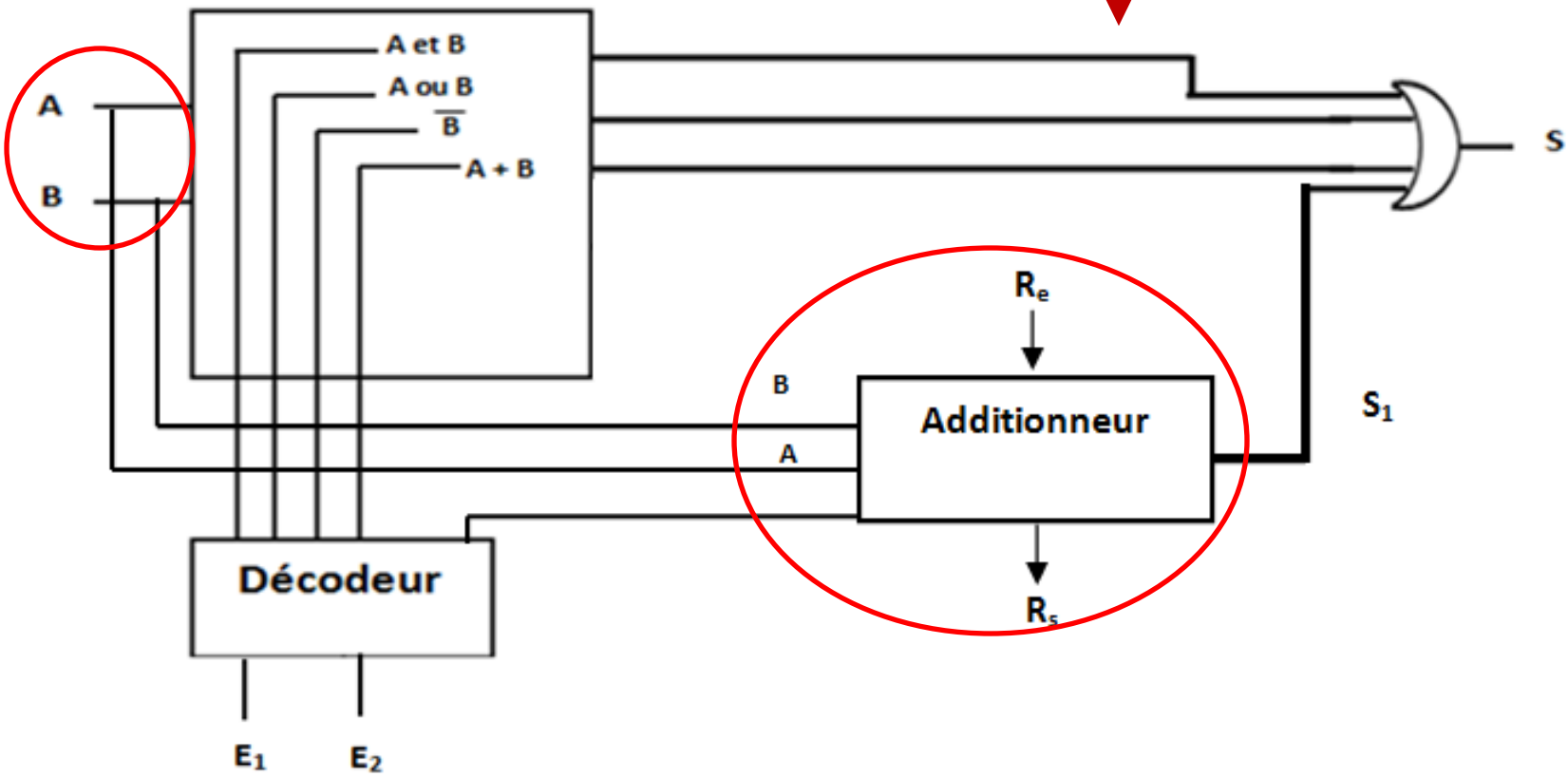


Le résultat d'une seule opération de l'UAL est transmis à un instant donné vers la sortie S.





Le schéma de cette UAL sera comme suit:

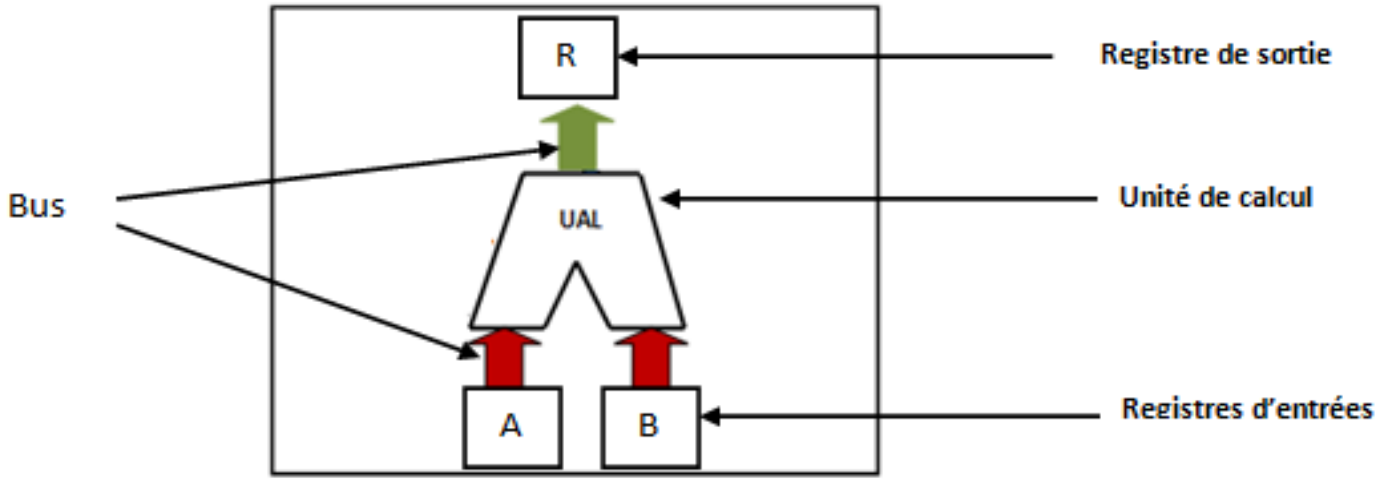


- Pour construire une UAL à **n bits**, il faut relier **n circuits** d'UAL à **1 bit** (Sortie)
- Les **circuits qui réalisent les opérations arithmétiques et logiques** constitue **l'unité de calcul** de l'UAL,

➢ **L'unité de calcul** comporte :

✓ **Des registres** (**mémoriser** les **données et le résultats**)

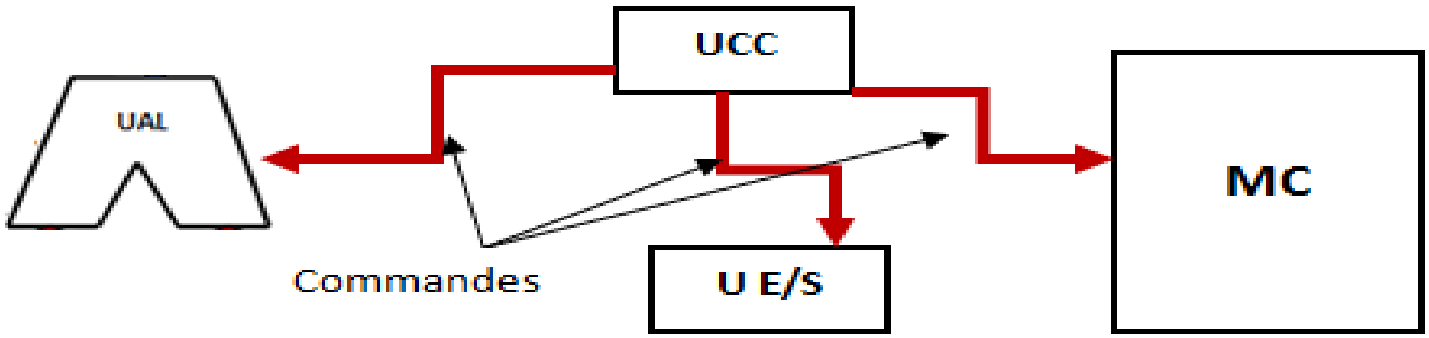
✓ **Des bus** qui **transmettent les données** entre **registres et l'unité de calcul**



4. Unité de commande et de contrôle (UCC)

4.1. Rôle de l'UCC

- Deux fonctions: **Commander et contrôler**,
- Dirige le fonctionnement des unités de l'ordinateur **(UAL, Mémoire, Entrée/Sortie)**
- Elle utilise des **circuits pour décoder les instructions** du programme et les transformer en signaux de commandes vers toutes ces unités



➢ *4.2. Les composants de l'UCC*

- **Des registres,** stocker des résultats temporaires ou des informations de commande:

✓ **Le compteur ordinal (CO) :** Le registre le plus important pointe sur la prochaine instruction à exécuter.

✓ **Le registre instruction (RI):** qui contient l'instruction en cours d'exécution. La taille du **RI** = taille du mot mémoire.

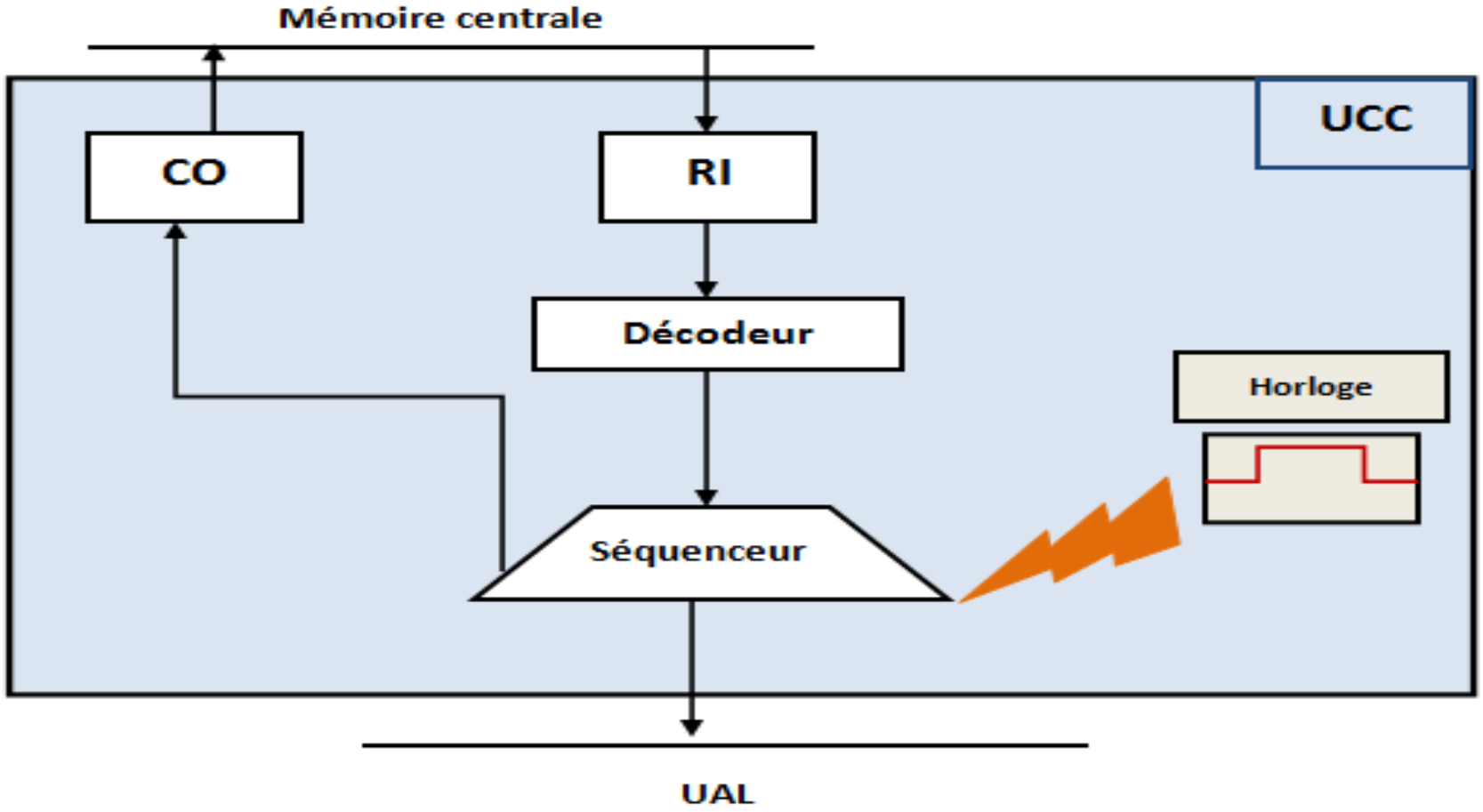
✓ **Le décodeur :** Un circuit combinatoire qui détermine quelle opération doit être effectuée.

✓ **Le séquenceur :** Le séquenceur est **un automate** générant les **signaux de commande** pour **actionner et contrôler** les unités participant à l'exécution d'une instruction donnée.

➢ Cet automate peut être réalisé de deux façons :

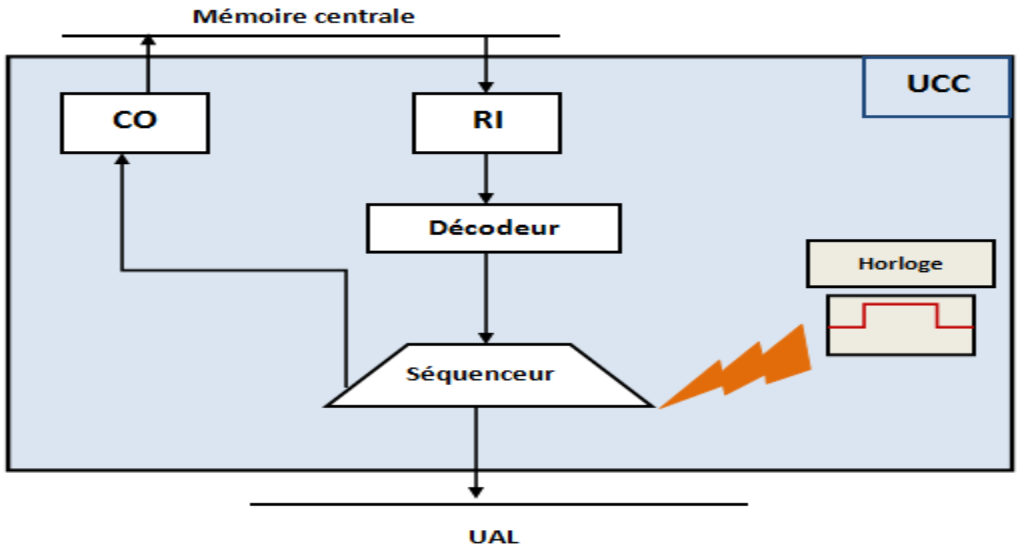
- Câblé : Circuit séquentiel
- Microprogramme : stockées dans ROM

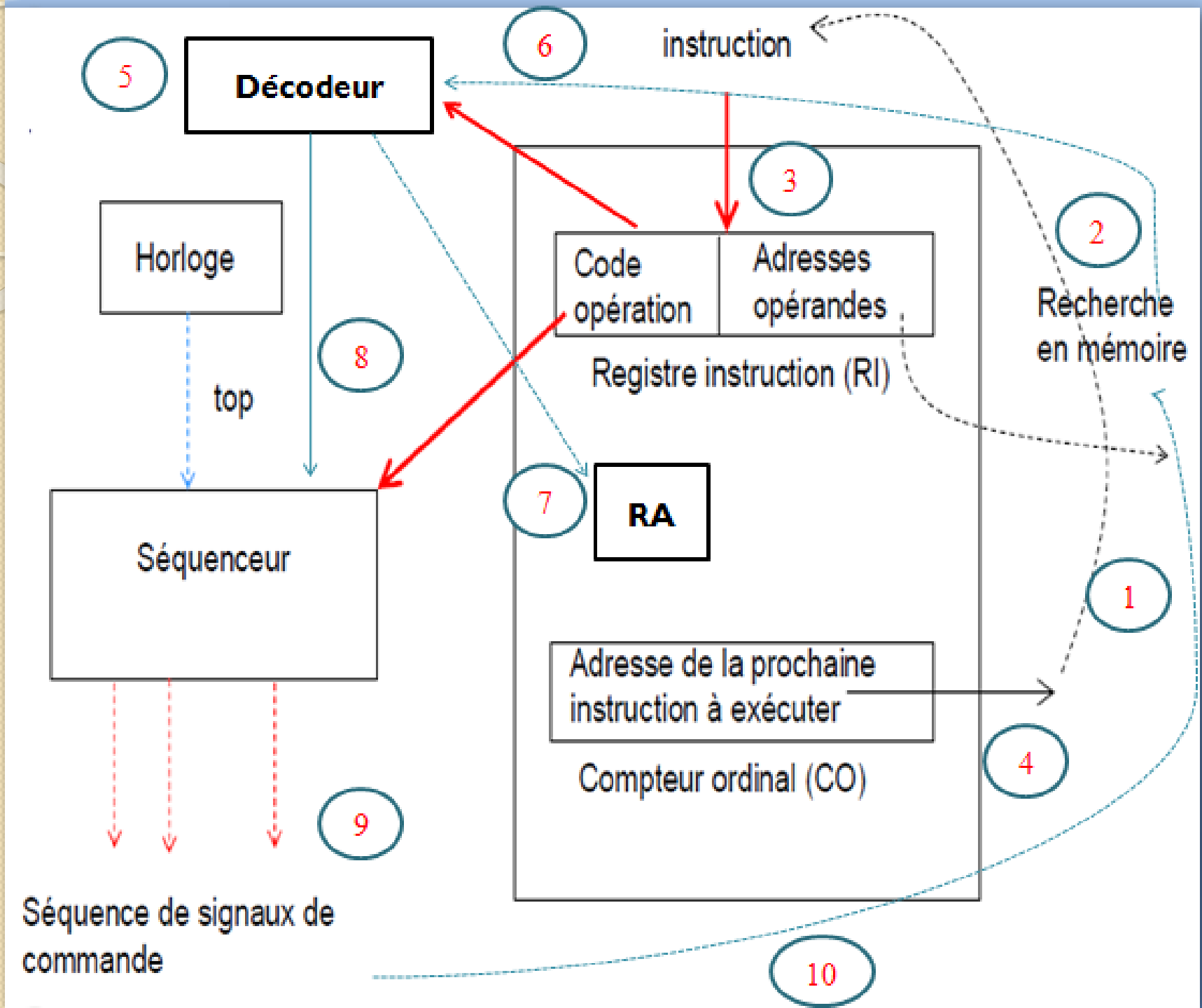
- **L'horloge :** Le rôle de l'horloge dans l'UCC est de générer des signaux périodiques définissent la durée de l'opération effectuée. Elle synchronise toutes les actions du processeur.



4.3. Le fonctionnement de l'UCC

- **Instruction=** code opération **+** adresse de l'opérande
- L'**UCC** cherchera ces instructions dans la mémoire ,
- Les instructions sont placées en mémoire à des adresses séquentielles, sauf autre cas (branchement, saut,……)




5
Décodeur
6
instruction
Horloge
3
2
Code opération
Adresses opérandes
Recherche en mémoire
Registre instruction (RI)
8
top
7
RA
Séquenceur
1
Adresse de la prochaine instruction à exécuter
4
Compteur ordinal (CO)
9
Séquence de signaux de commande
10

4.3. Le fonctionnement de l'UCC

- **L'exécution d'une instruction par l'UCC passe par les étapes suivantes:**

✓ **1. Le compteur ordinale (CO):** donne l'adresse de la **nouvelle instruction** à exécuter et l'envoie vers la mémoire centrale.

✓ **2. Une commande de lecture** générée par l'UCC permet de lire l'instruction dans la mémoire MC.

✓ **3. Chargement de l'instruction** à exécuter depuis la mémoire dans le registre instruction **(RI).**

✓ **4. Modification du compteur ordinal** pour qu'il pointe sur l'instruction suivante.

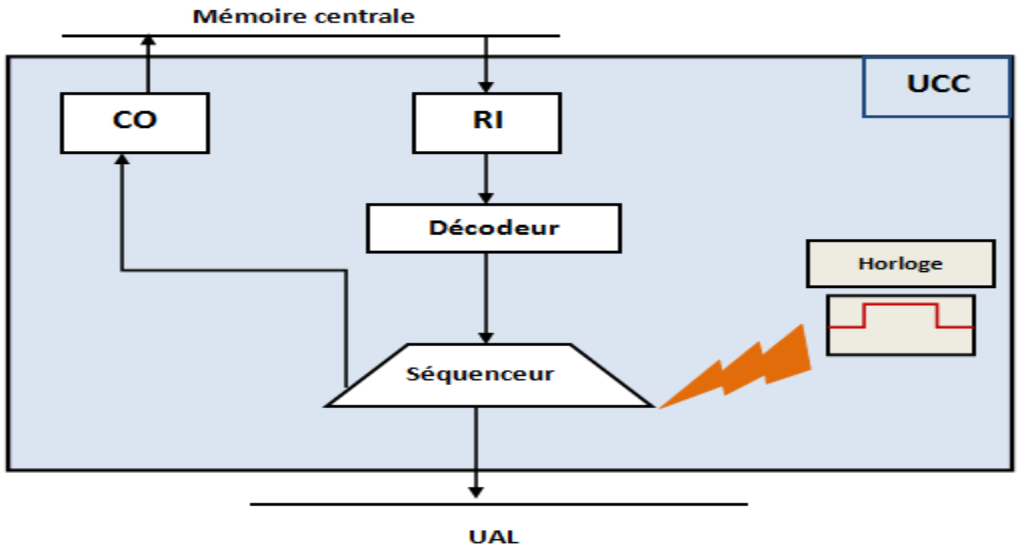
✓ **5. Décodage de l'instruction** que l'on vient de charger.

✓ 6. **Localisation dans la mémoire** des éventuelles **données utilisées** par l'instruction.

- **7.Chargement des données**, si nécessaire, dans les registres internes.
- 8. **Le séquenceur envoie un signal de commande** vers l'UAL pour **l'exécution de l'instruction.**
- **9.Pour le stockage des résultats** à leurs destinations respectives, le séquenceur envoie un signal de commande d'écriture en mémoire,
- 10. Le résultat est transféré de l'UAL vers la mémoire.
- **Retour** à l'étape 1 pour exécuter l'instruction suivante.

- **Instruction=** code opération **+** adresse de l'opérande

- L'**UCC** cherchera ces instructions dans la mémoire ,
- Les instructions sont placées en mémoire à des adresses séquentielles, sauf autre cas (branchement, saut,……)

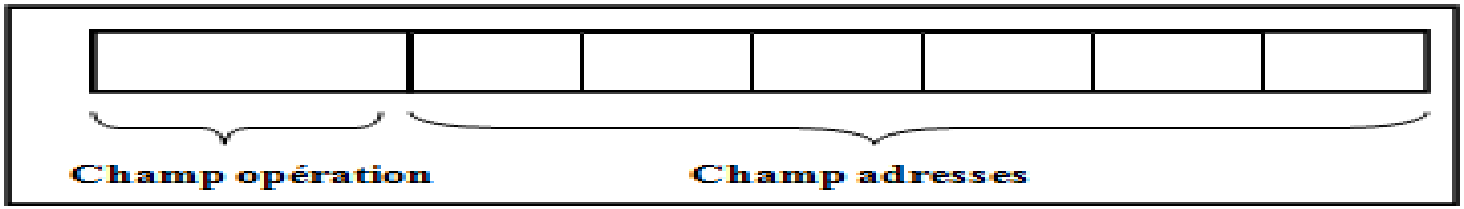


5. Jeu d'instruction

- **Un programme = suite d'instructions** (écrites en un langage de programmation),
- **Instructions** ensuite introduites en langage machine seront **traitées** par la machine avec le **microprocesseur** ⇒ **Instruction machine.**
- **Une instruction machine** comporte:

✓ Un champ **code opération** (champ obligatoire),

✓ Un ou plusieurs champs **d'adresses (adresse de l'opérande, adresse du résultat, adresse de l'instruction suivante)**



Instruction machine

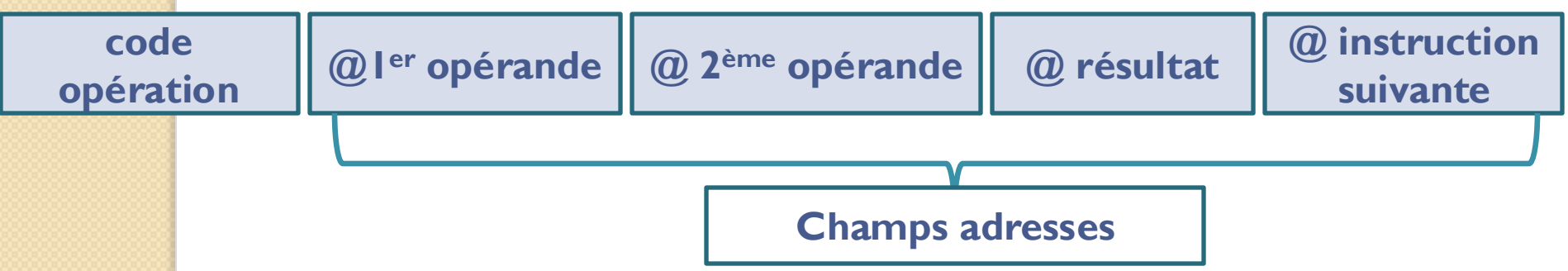
5.2. Les différents formats d'instructions machine :

- Le ordinateurs actuels, on instructions à **une seule adresse :**
  - ✓ Le **compteur ordinale (CO)** joue le rôle du **champ adresse** ,
  - ✓ **Champ adresse du résultat**, il est remplacé par un registre spécial de l'**UAL** appelé **accumulateur**
- Machine **à zéro adresse**: utilise une mémoire **LIFO (Last In First Out)** appelée aussi **Pile.**

- Les formats d'instructions ⟺ l'évolution des générations d'ordinateurs,

(n =0, ……4) adresses le format de l'instruction machine

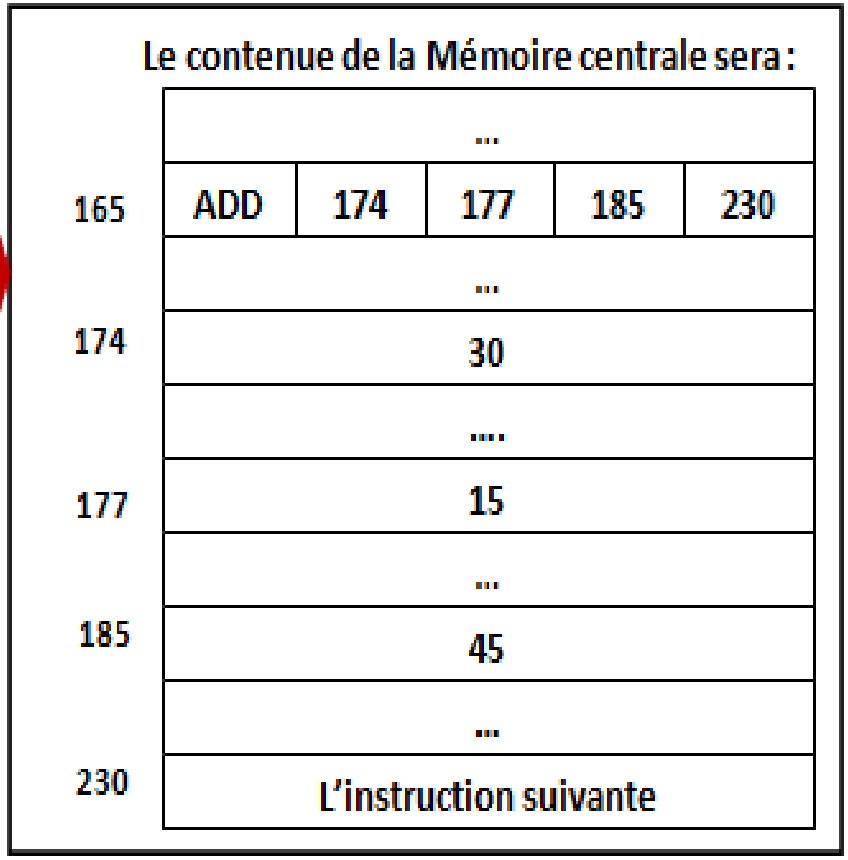
✓ ***Machine à 4 adresses*** **: Dans les premiers ordinateurs**



- Exemple : 30+15
- L'adresse de l'instruction addition est 185,
- L'adresse première opérande (30) est 174, L'adresse deuxième opérande (15) est 177
- L'adresse de l'instruction suivante est 230

Le contenu du registre instruction RI :

ADD	174	177	185	230



- L'adresse de résultat est 185,
- L'adresse première opérande (30) est 174, L'adresse deuxième opérande (15) est 177
- L'adresse de l'instruction suivante est 230

✓ *Machine à 3 adresses :*

- Le **champ de l'instruction suivante est supprimé** ⇒ **Compteur ordinal (CO)** qui est chargé de calculer cette adresse

code opération	@1er opérande	@ 2ème opérande	@ résultat

Champs adresses

Le contenu du registre instruction RI :

ADD	174	177	185

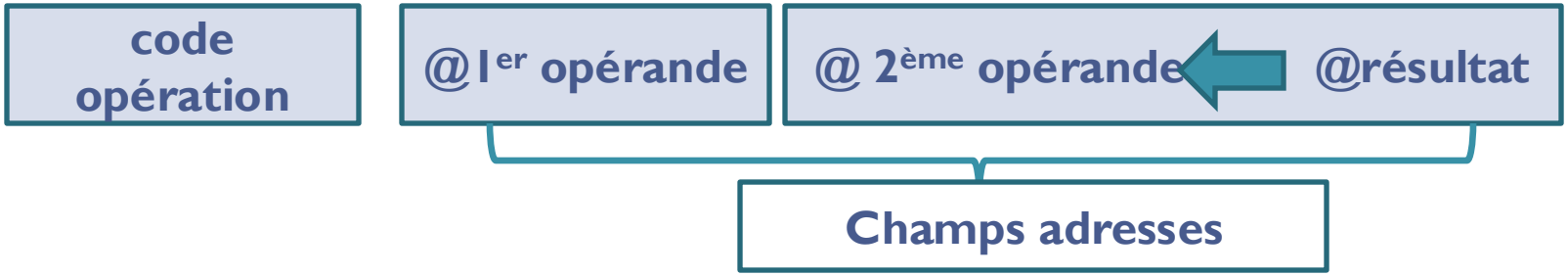
Le contenue de la Mémoire centrale sera :

	...			
165	ADD	174	177	185
	.....			
174	30			
	....			
177	15			
	...			
185	45			
	...			

Le contenu du compteur ordinal (CO) est : 230

✓ *Machine à 2 adresses :*

- Le champ, **adresse du résultat** a été supprimée, l'adresse du résultat sera rangée **à la place du deuxième opérande**. (Champ adresses à 2 adresses)



L'instruction contient deux adresses :

ADD	174	177

Le contenue de la Mémoire avant l'exécution:

	...		
165	ADD	174	177
	...		
174	30		
	....		
177	15		
	...		

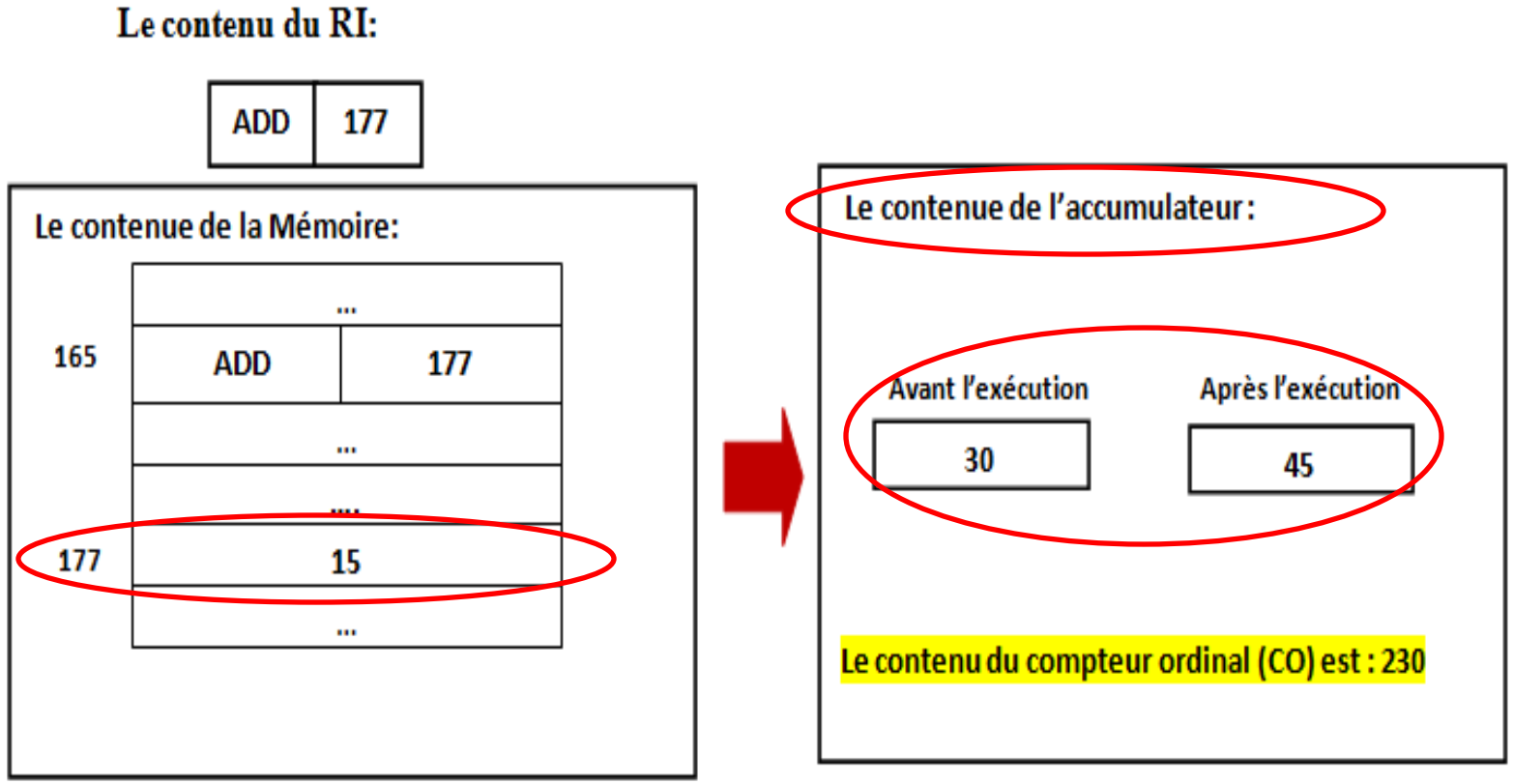
Le contenue de la Mémoire centrale sera :

	...		
165	ADD	174	177
	.....		
174	30		
	....		
	...		
177	45		
	...		

Le contenu du compteur ordinal (CO) est : 230

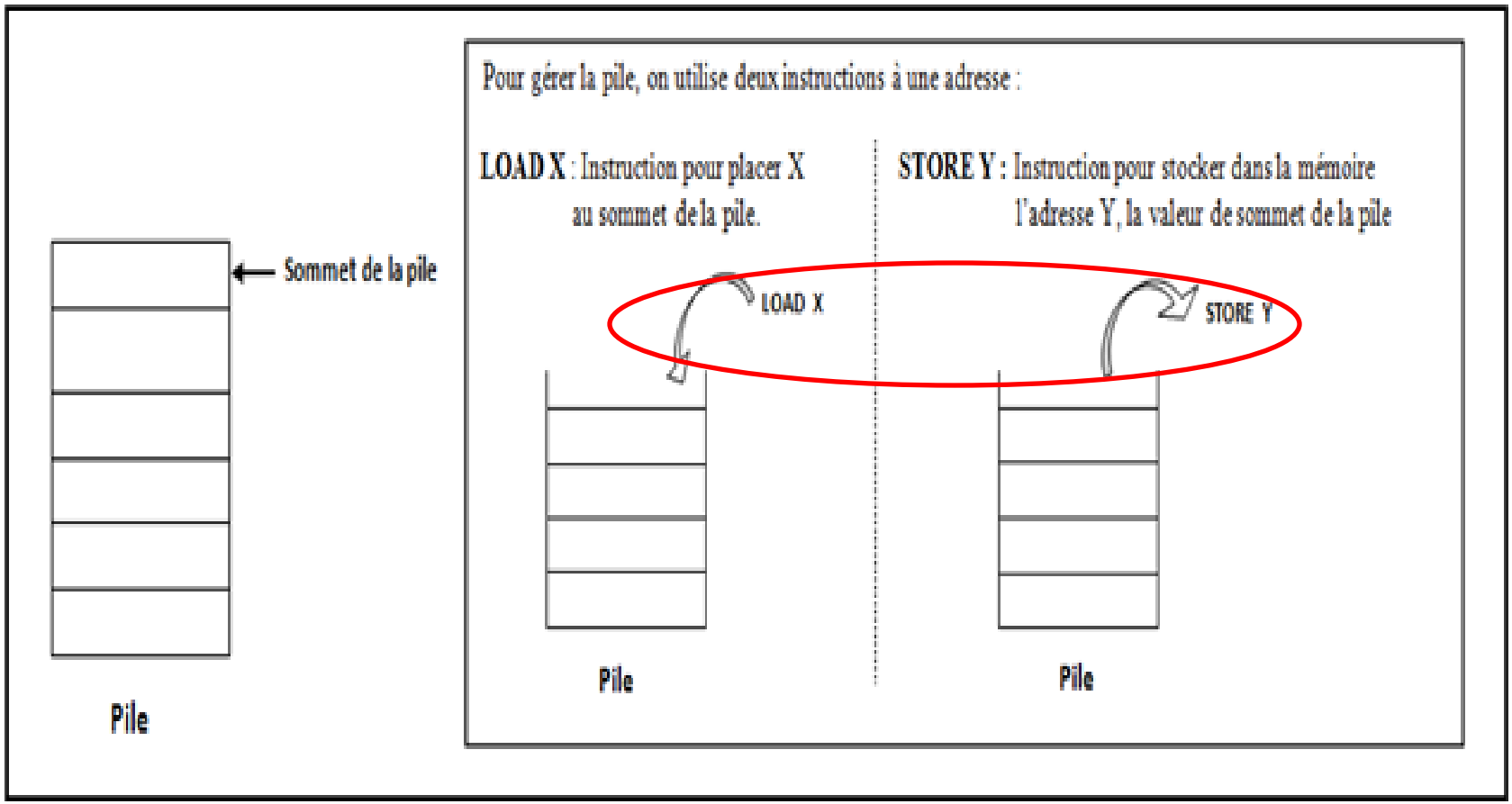
✓ *Machine à 1 adresse*

- Le format de l'instruction contient qu'une **seule adresse** = L'adresse de premier opérande,
- Le **deuxième opérande** se trouve **dans l'accumulateur** (donc pas besoin de son adresse).



✓ *Machine à zéro adresse (ou à pile)*

- D'abord : Le fonctionnement du pile???
- Pour gérer une Pile : Deux instructions à une adresse « LOAD, STORE » **(Charger, Stocker)**



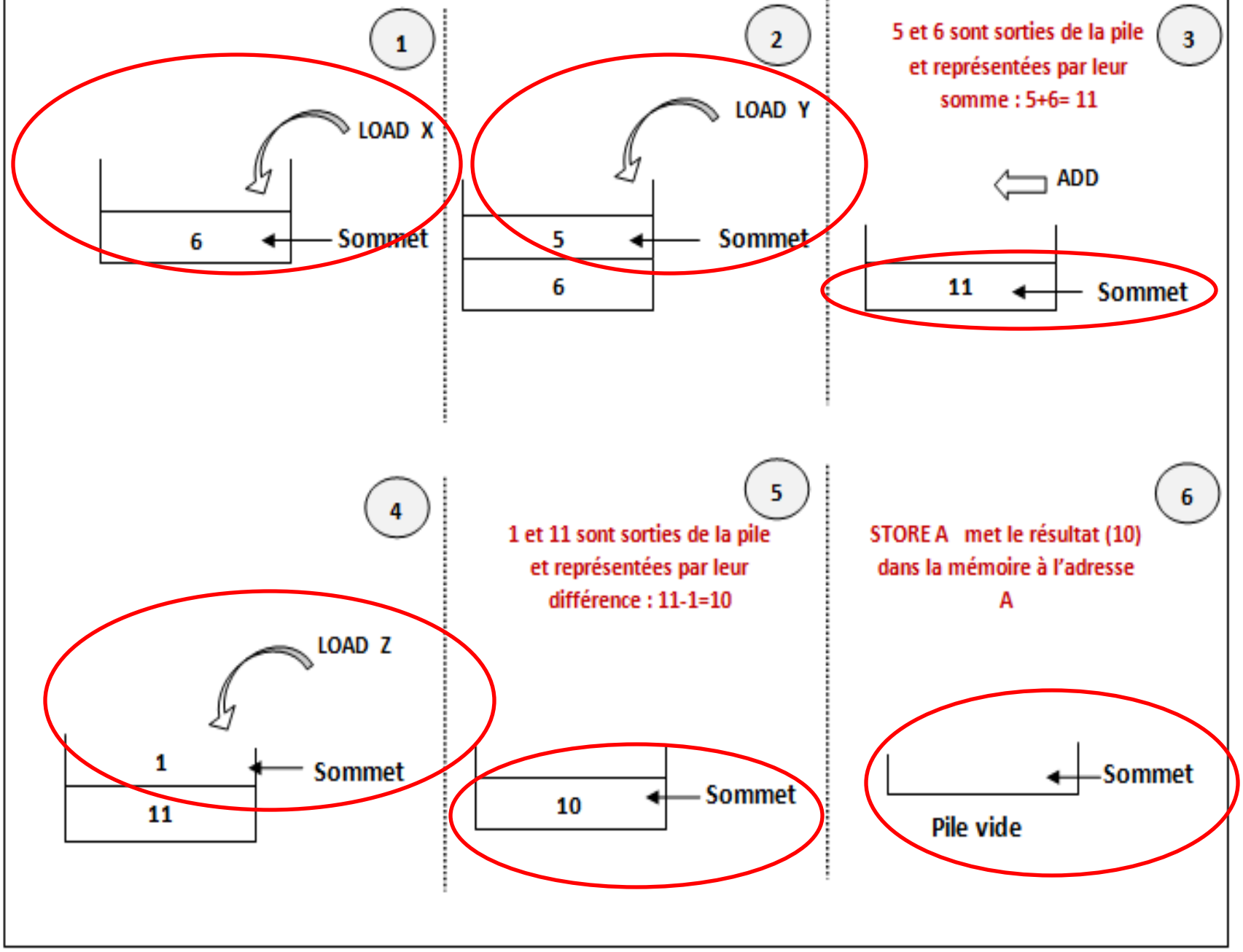
- Quelques instructions du **machine à zéro et à une adresse** :

Instruction	Description
ADD	Addition
MULL	Multiplication
DIV	Division
SUB	Soustraction
STA	Rangement
LOAD	Chargement
STORE	Stockage

  - **Exemple :** Soit à exécuter l'expression suivante:
    A=X+Y-Z ou : X=6, Y=5, Z=1
- La suite d'instruction correspondante à l'évaluation de cette expression est la suivante :

LOAD X , LOAD Y, ADD, LOAD Z, SUB, STORE A

Le contenu de la pile :



5.3. Architecture du jeu d'instructions:

- **Jeu d'instructions = un ensemble d'instructions utilisés par le processeur pour exécuter ses traitements**

- Chaque machine ⇨ son propre jeu d'instruction.
- Il existe deux architectures du jeu d'instructions des machines :

✓ **Architecture RISC (Reduced Instruction Set Computer) :** qui signifie ***processeur à nombre d'instructions réduit.***

✓ **Architecture CISC (Complexe Instruction Set Computer) :** qui signifie ***processeur à jeu d'instructions complexes.***

Architecture CISC	Architecture RISC
- Jeu d'instructions large - Instructions complexes - Instructions de tailles différentes - Instructions de durées différentes - Séquenceur micro-programmé - **Richesse du langage machine**	- Jeu d'instructions réduit - Instructions simples - Instructions de mêmes tailles - Instructions de mêmes durées - Séquenceur câblé - **Efficacité du langage machine**

- A présent presque tous les processeurs proposent des **jeux d'instructions très similaires**. Les instructions qu'on retrouve dans toute machine:

✓ **Les instructions de transfert des données :**
- **Entre la mémoire et les registres (ex : Load, Store)**
- **Entre registre et registre (ex : Move)**

✓ **Les instructions de traitement des données :**
- **Opérations arithmétiques (ex: ADD, SUB, MUL, DIV,….)**
- **Opérations logiques (ex : AND, OR, NOT, XOR, ……)**

✓ **Les instructions de contrôle de l'exécution :**
- **Branchements conditionnels**
- **Branchements inconditionnels ou sauts**
- **Appel de procédure**

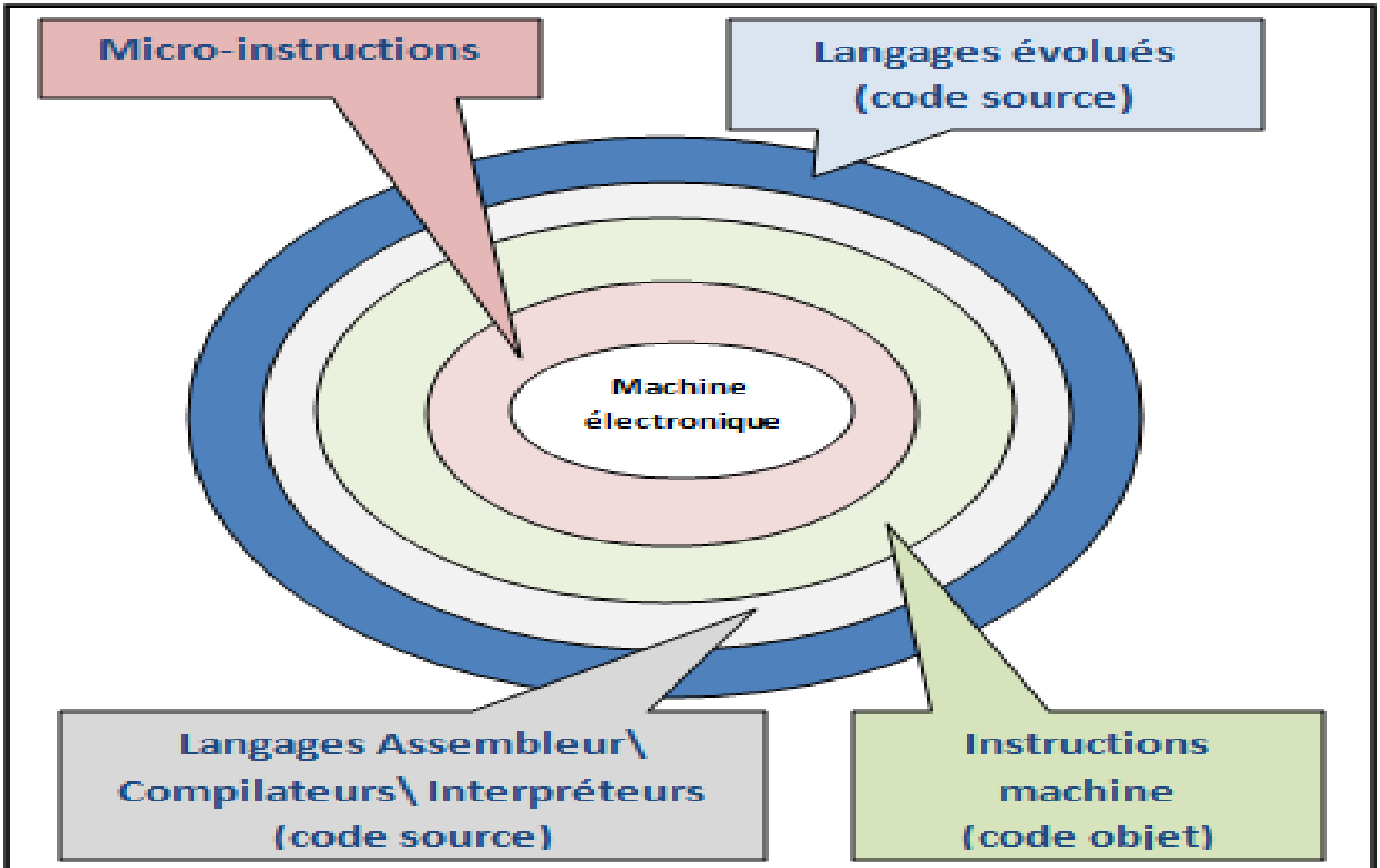
✓ **Les instructions d'entrées / sorties.**

5.4. Architecture du jeu d'instructions:

- Le **modèle de programmation** d'un processeur définit la manière dont les instructions accèdent à leurs opérandes et leur **façon d'être décrites dans le langage assembleur du processeur.**
- Sur la plupart des machines, on retrouve deux **modèles de programmation** utilisées sur deux types de processeurs :

✓ **Les architectures à registres généraux**

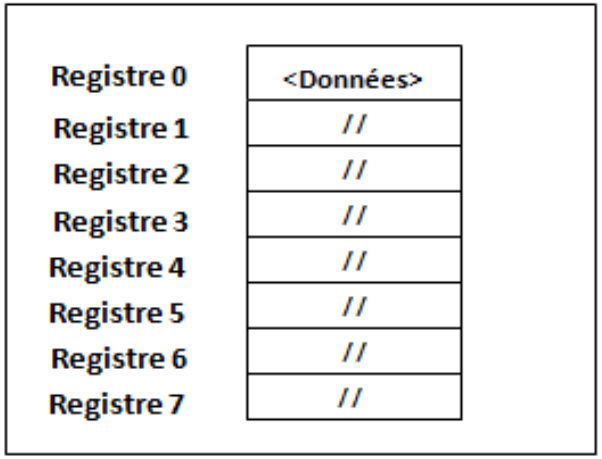
✓ **Les architectures à Pile**

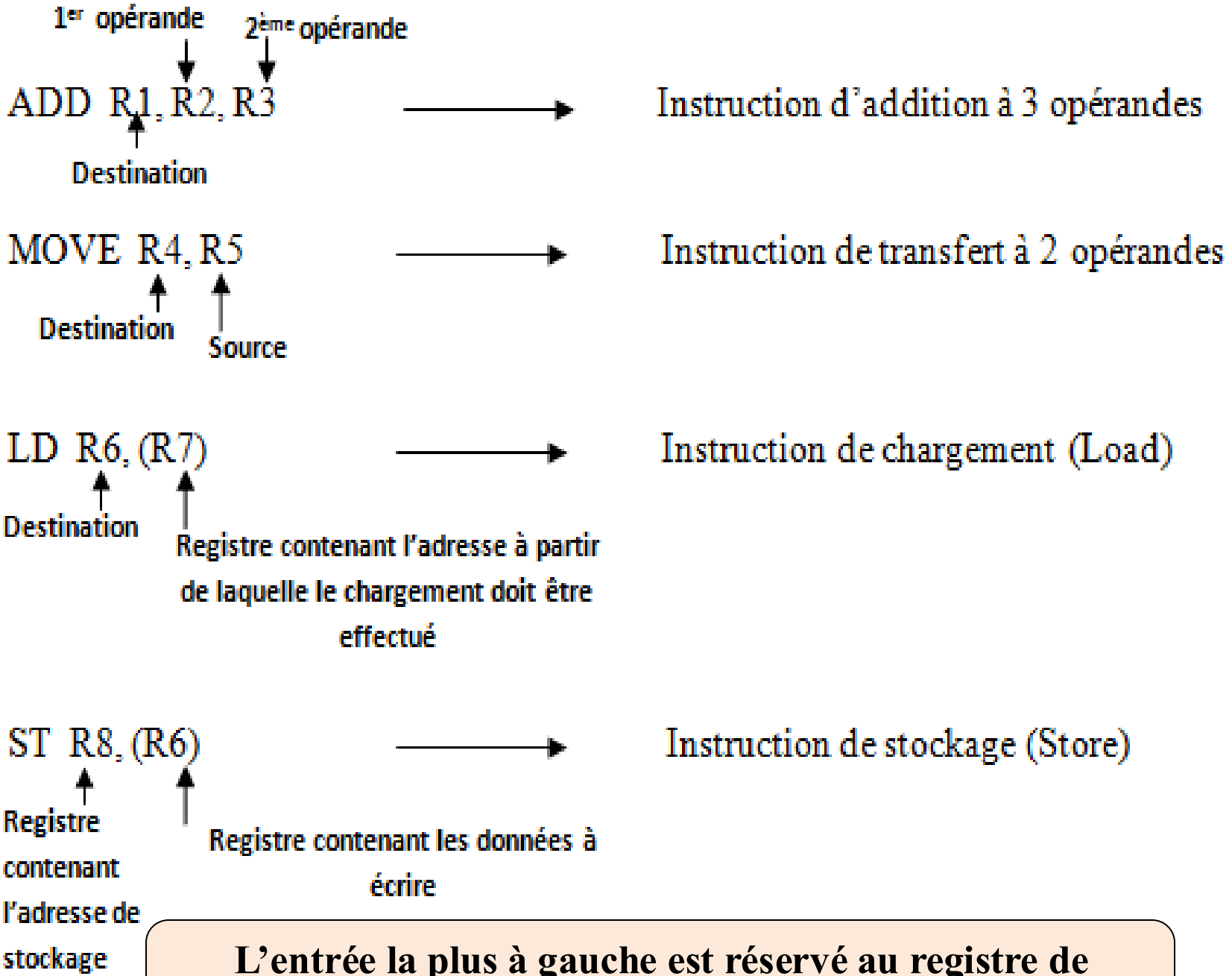


Les niveaux de programmation

✓ Architecture à registres généraux

- Les instructions **lisent leurs opérandes** dans **des registres** et y **écrivent leurs résultats**.
- Cet ensemble de registres est appelé **fichier de registres généraux**
- L'accès au registre est **aléatoire**, chaque registre possède **un identificateur**.
- Les instructions doivent spécifier les registres qui contiennent leurs opérandes **d'entrée** et celui dans lequel **leur résultat** doit être écrit.
- On utilise en général le format d'instruction à 3entrées





L'entrée la plus à gauche est réservé au registre de destination pour mettre le résultat

- **Exemple:** Un programme pour un processeur à registres généraux: **(a+b) x c**

Avec **a, b et c des constantes**.

Le programme sera comme suit :

```
MOVE   R1, # a
MOVE   R2, # b      # X: signifie que X est une constante
ADD     R3, R1, R2
MOVE   R4, #c
MUL     R5, R3, R4
```

- **MOVE** placent les constantes **a** et **b** dans registres **R1,R2**
- **ADD** addition le contenu des registres **R1** et **R2**et **le résultat** dans **R3.**
- La quatrième instruction met la constante **c** dans le registre **R4**,
- l'instruction **MUL** multiple le contenu de **R3** et **R4** et met le **résultat** dans **R5.**

- ***- Le programme précédent peut s'écrire comme suit :***

```
MOVE   R1, # a
MOVE   R2, # b
ADD    R2, R1, R2
MOVE   R1, #c
MUL    R3, R2, R1
```

On a utilisé 3 registres au lieu 5 registres.

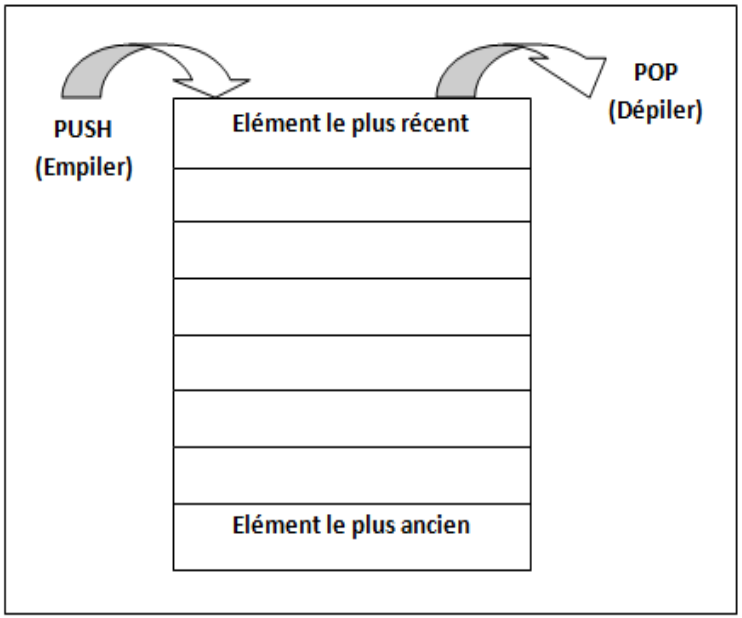
✓**L'architecture à registres généraux :**
L'instruction de chargement (LOAD) et de stockage (STORE) sont *les seules qui peuvent accéder à la mémoire*.

✓ Architecture à Pile

- Les instructions lisent leurs opérandes et écrivent leurs résultats dans une pile (**LIFO** : **Last In First Out**).
- Avec « Machine à zéro adresse » ; une pile est une structure de données qui obéit au principe « le dernier qui entre est le premier qui sort (PEPS) »

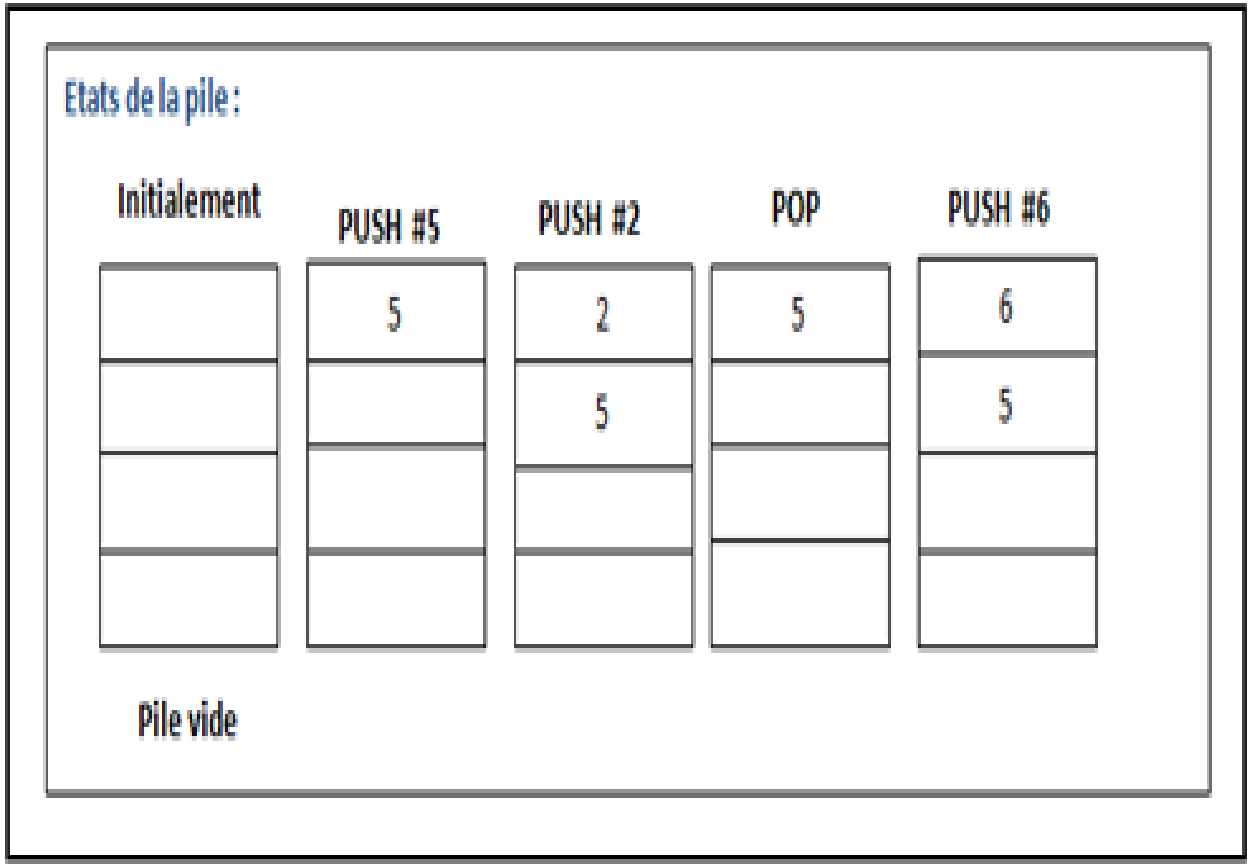
✓**Avec le jeu d'instructions *à zéro adresse* on a cité deux instructions *(LOAD, STORE)* qui permettent de *gérer la pile et d'accéder à la mémoire*.**

- L'architecture Pile utilise deux instructions de base, **PUSH** et **POP**
- **L'opération PUSH :** permet de placer une donnée au sommet de la pile en poussant les données précédentes vers le bas.
- **L'opération POP:** permet de retirer la valeur du haut de la pile pour être utilisée comme entrée d'une instruction.



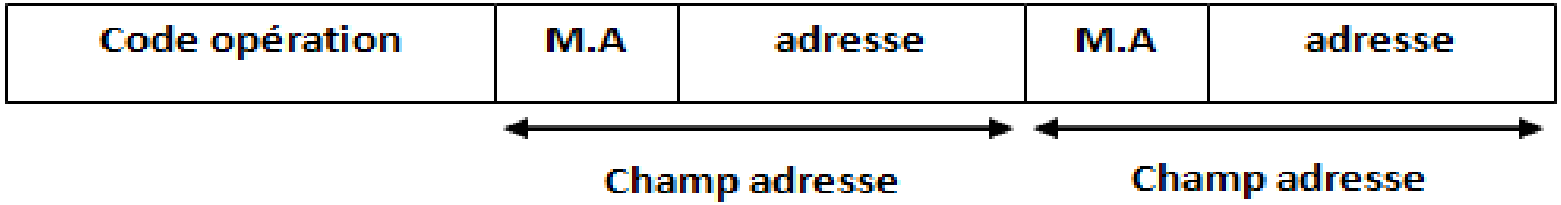
Exemple1 : On considère le jeu d'instruction suivant :

PUSH #5
PUSH #2
POP
PUSH #6



6. Mode d'adressage

- Un **champ d'adresse** d'une instruction peut référencer un mot mémoire ou un registre.
- **Le mode d'adressage fait partie du champ d'adresse**



- **Le mode d'adressage** défini par le code opération celui-ci impose un type déterminé

- On présentera les modes d'adressage et on expliquera la manière dont ils opèrent.

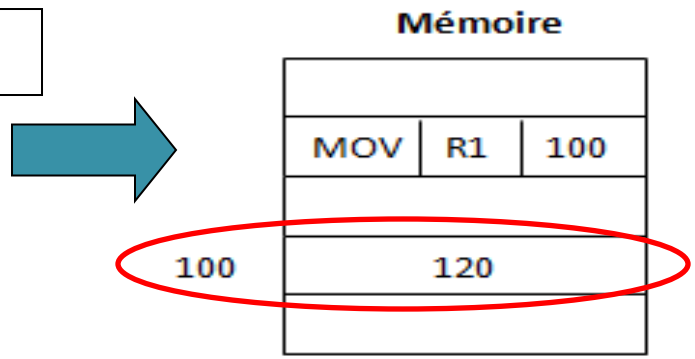
6.1. Adressage immédiat

- Dans ce cas, **l'opérande** est contenu **dans le champ adresse** de l'instruction.
- Exemple : MOV R2, #100
  Le contenu de R2 : **100**

6.2. Adressage direct

- Le **champ adresse** de l'instruction contient **l'adresse effective de l'opérande** en mémoire centrale.
- Exemple : MOV R1, 100

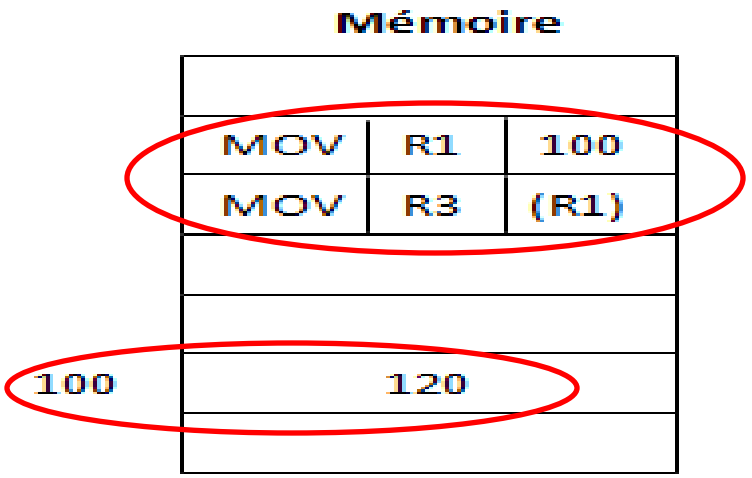
- Cette instruction met dans le registre R1, l'opérande qui se situe à l'adresse 100 en mémoire.
- Le contenu de R1 : **120**



6.3. Adressage indirect

- **Le champ adresse** contient l'adresse d'un pointeur, c'est-à-dire l'adresse effective de l'opérande
- Exemple : MOV R3, (R1)
  - ✓Si l'on suppose que R1 contient la valeur 100
  - ✓Alors, après l'instruction MOV R3, (R1)
  - ✓**R3 contiendra la valeur 120**

R1	**100**
R3	**120**



6.4. Adressage relatif

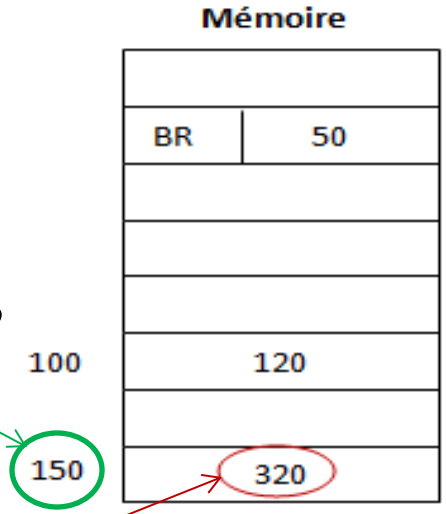
- **L'adresse de l'opérande** est obtenue en **additionnant** le contenu du compteur ordinal (CO) au contenu du champ adresse de l'instruction.
- Ce type d'adressage est utilisé, en général dans les **instructions de branchement**.

- Exemple : BR, 50 / On suppose que le contenu du compteur ordinal est 100.

CO | 100 |

✓ Adresse de l'opérande est **100+50**,

✓ Alors l'opérande est **320**



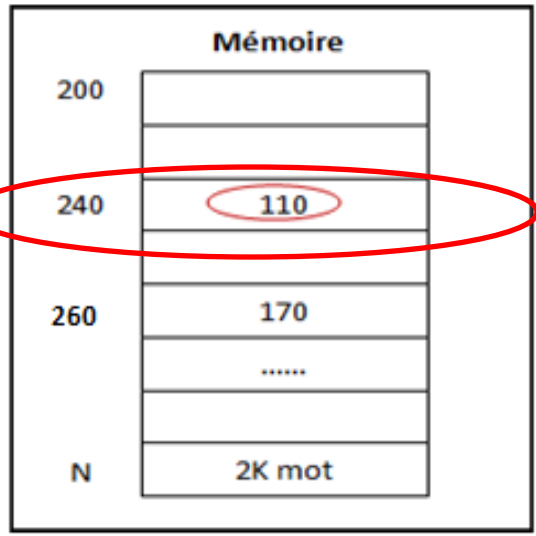
6.5. *Adressage basé*

- **L'adresse de l'opérande** est obtenue en additionnant le contenu du registre de base (RB) au contenu du champ adresse de l'instruction.
- Le registre de base contient l'adresse du début **d'un bloc de 2K mots**.
- Le champ adresse de l'instruction contient le déplacement dans le bloc.

- Exemple : R1, 40

RB | 200 |

Adresse de l'opérande est 200+40

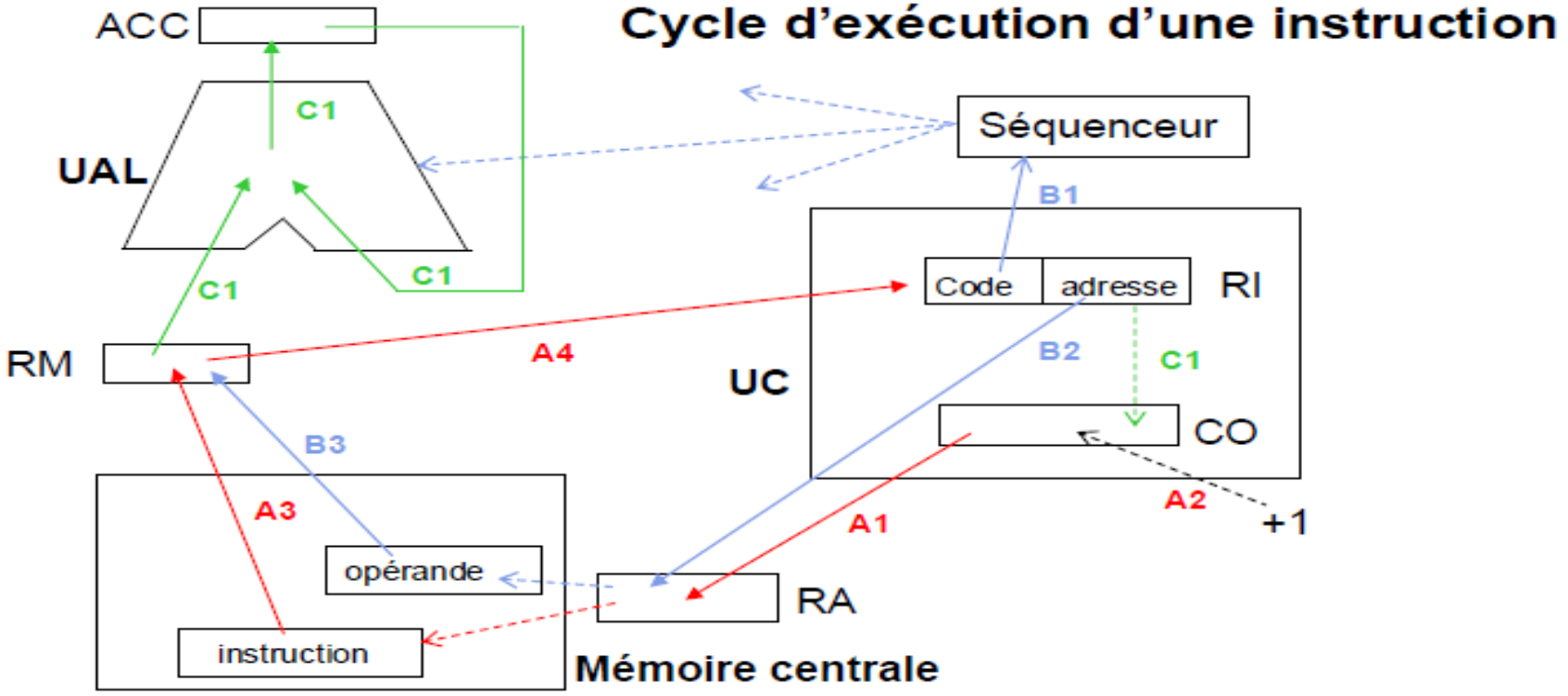


6.6. Adressage indexé

- L'adresse de l'opérande est calculée en **additionnant** le contenu du champ adresse de l'instruction au contenu du registre d'index.
- Ce type d'adressage est utilisé pour les tableaux, l'index permet de référencer un élément dans un tableau.

L'adresse de l'opérande est: 120+20=140

7. Etapes d'exécution d'une instruction



A1 : CO dans RA (adresse prochaine instruction dans RA)
A2 : incrémentation de 1 de CO (prochaine instruction à l'adresse suivante)
A3 : lecture de l'instruction et rangement dans RM — CHERCHER
A4 : transfert de l'instruction de RM à RI pour décodage
B1 : analyse du code instruction; envoi des signaux de commande — DECODER
B2 : adresse de l'opérande dans RA
B3 : lecture de l'opérande en mémoire et rangement dans RM — EXECUTER
C1 : **calcul** sur opérandes dans RM et ACC (a été précédé par un transfert du 1er opérande de la mémoire vers ACC et est suivie d'un transfert du résultat de ACC vers la mémoire).

OU

C1 : **branchement** adresse est rangée dans CO (devient la prochaine instruction à exécuter).