# Unité centrale

## 1. Introduction

Aussi appelée CPU (**C**entral **P**rocessing **U**nit) ou processeur central, l'unité centrale est le cœur de l'ordinateur. C'est elle qui exécute les instructions des programmes d'application et qui traite les données contenues dans la mémoire centrale. L'unité centrale se compose d'un ensemble d'unités chargées de mémoriser, de transformer et de traiter les instructions, les adresses et les données.

## 2. Les principales composantes du C.P.U.

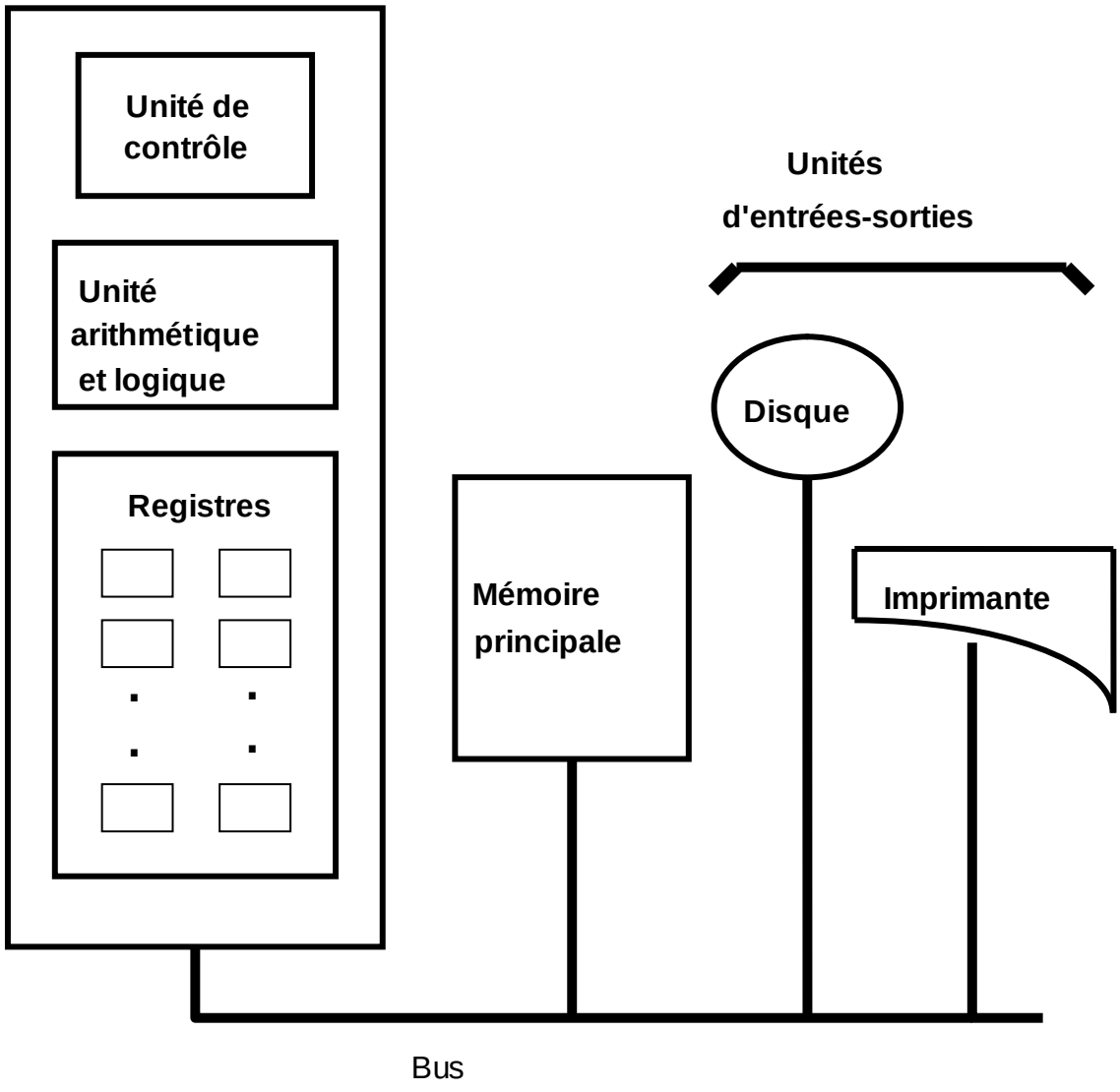
Comme première partie, on trouve l'unité de commande (ou de contrôle ou séquenceur), est la partie du CPU qui déclenche les différentes phases de l'exécution des instructions. C'est pour ainsi dire le cerveau de l'ordinateur.

La seconde partie est l'unité arithmétique et logique ou ALU (Arithmetic and Logic Unit), ou encore unité de calcul, est la partie du CPU qui s'occupe d'exécuter les opérations logiques et les calculs arithmétiques commandés par le ou les programmes. C'est le "tâcheron" par excellence.

Lorsque l'unité de commande (ou de contrôle) et l'unité arithmétique et logique sont intégrées sur une même "puce" ou microcircuit, on désigne l'ensemble par le terme processeur ou microprocesseur pour les ordinateurs de petite taille. Le processeur est également muni de registres qui mémorisent temporairement l'information significative nécessaire. Plusieurs types de registres sont présents dans le CPU. Il y a des registres à adressage explicite, qui sont utilisés directement par les programmes, et des registres à adressage implicite, qui sont nécessaires au fonctionnement interne du processeur. Selon le type d'information qu'un registre est destiné à contenir, on parlera de registre d'adresse, de registre arithmétique fixe ou flottant, de registre d'instruction, de registre à décalage etc.

Le schéma suivant montre l'architecture simplifiée d'un ordinateur traditionnel.

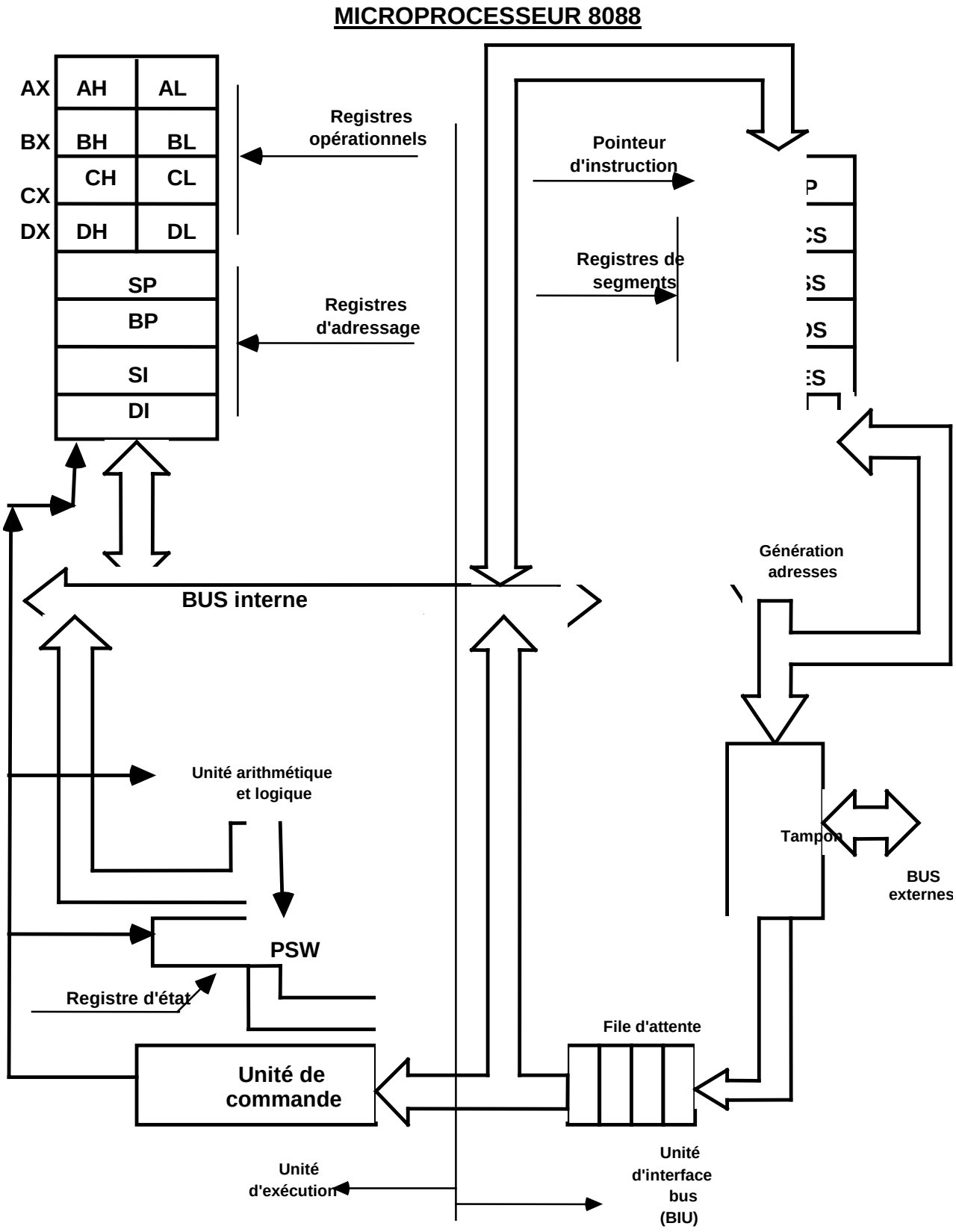
**Figure 1 : Organisation d'un ordinateur simple**




Sur les ordinateurs de petite taille, il arrive que les unités d'échange et les interfaces d'unités périphériques soient intégrés au CPU. Les interfaces sont des éléments de périphériques (ou de calculateurs). C'est à l'interface qu'aboutissent les lignes de transmission. L'interface effectue certaines opérations préliminaires à l'envoi d'information.

Par exemple, le schéma suivant montre le microprocesseur d'un IBM-PC où l'on peut apercevoir la présence des différents registres, interfaces et unités. Nous verrons plus en détails le rôle de ces différentes composantes du PC dans un chapitre ultérieur.

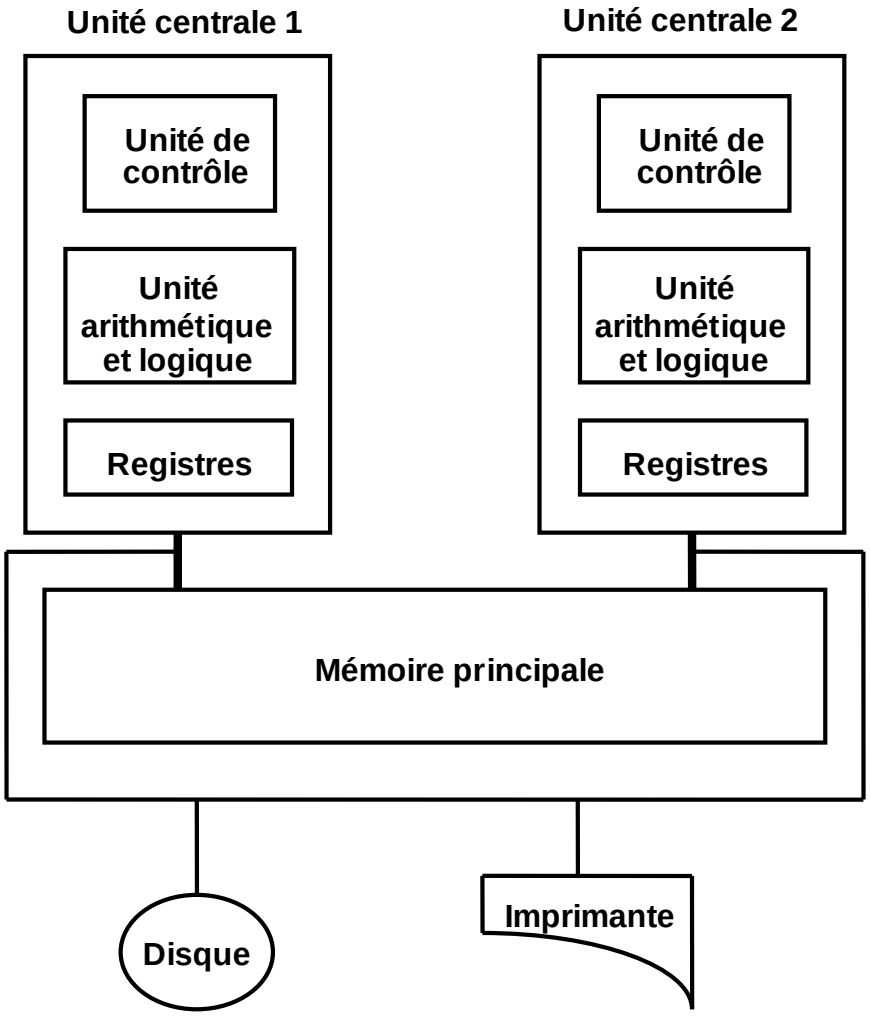
**Figure 2 :Microprosseur 8088**



Dans la structure du processeur 8088 du PC, on trouve des registres opérationnels réservés aux diverses opérations arithmétiques et logiques (AX, BX, CX et DX), des registres d'adressage servant à remiser et à calculer les adresses (SP, BP, SI et DI), des registres de segments remisant le pointeur de début des divers segments (CS, SS, DS et ES), et des registres spéciaux pour le pointeur d'instruction (IP), le vecteur d'état (PSW), …, etc.

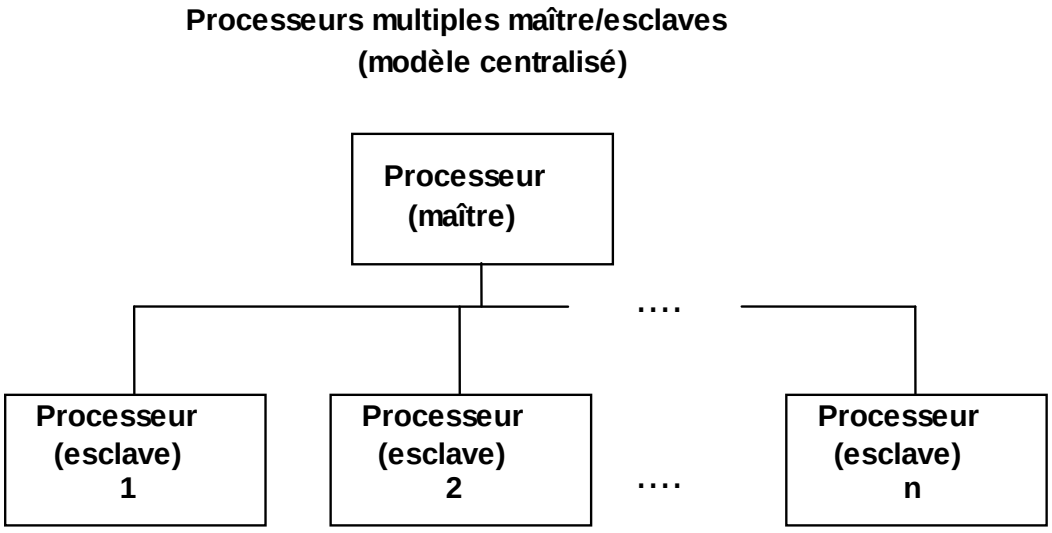
Sur les gros ordinateurs, le CPU dispose souvent d'un ou plusieurs blocs de mémoire et d'organes d'entrées/sorties. Il arrive aussi souvent que plusieurs processeurs soient mis en commun dans des architectures dites bi, tri, quadri ou multiprocesseurs. Le schéma suivant montre une organisation bi-processeurs.

**Figure 3 : Organisation d'un ordinateur à unités centrales multiples**



Le terme processeur désigne alors une unité d'échange sophistiquée et ayant une large autonomie de fonctionnement. Les processeurs peuvent être indépendants les uns des autres (architecture dite monoprocesseur) ou non, selon le cas. S'ils sont dépendants, le processeur chargé de gérer le système d'exploitation et de distribuer les tâches aux autres processeurs est appelé le processeur maître, alors que les autres processeurs sont appelés processeurs esclaves.

**Figure 4 : Processeurs multiples maîtres et esclaves**

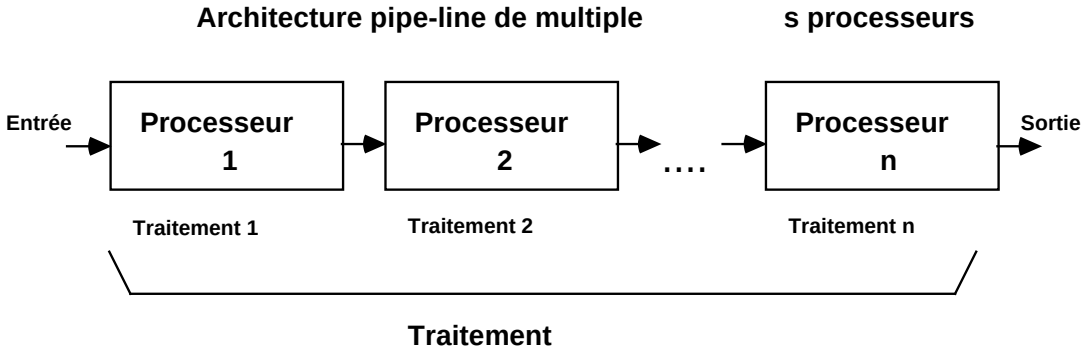



Le VAX 11/782, par exemple, est constitué de deux VAX 11/780 fonctionnant en mode maître/esclave.

## 2. 1. Autres architectures

Toujours sur les gros ordinateurs, on parle aussi de traitement en chaîne ou d'architecture pipe-line, qui consiste en une série de processeurs exécutant chacun une partie du travail. Ce type d'architecture s'adresse à des traitements spécialisés.

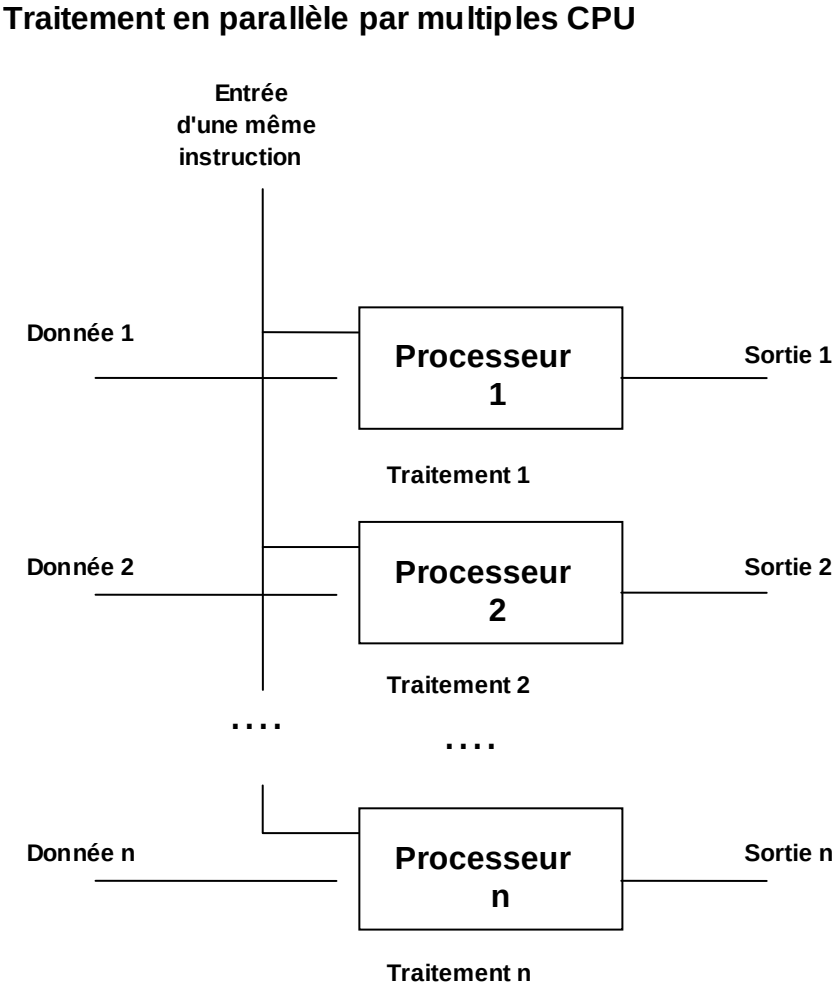
**Figure 5: Architecture pipe-line**



Dans l'architecture en pipe-line, l'information qui sort d'un processeur est nécessaire au processeur suivant qui exécute la suite du travail.

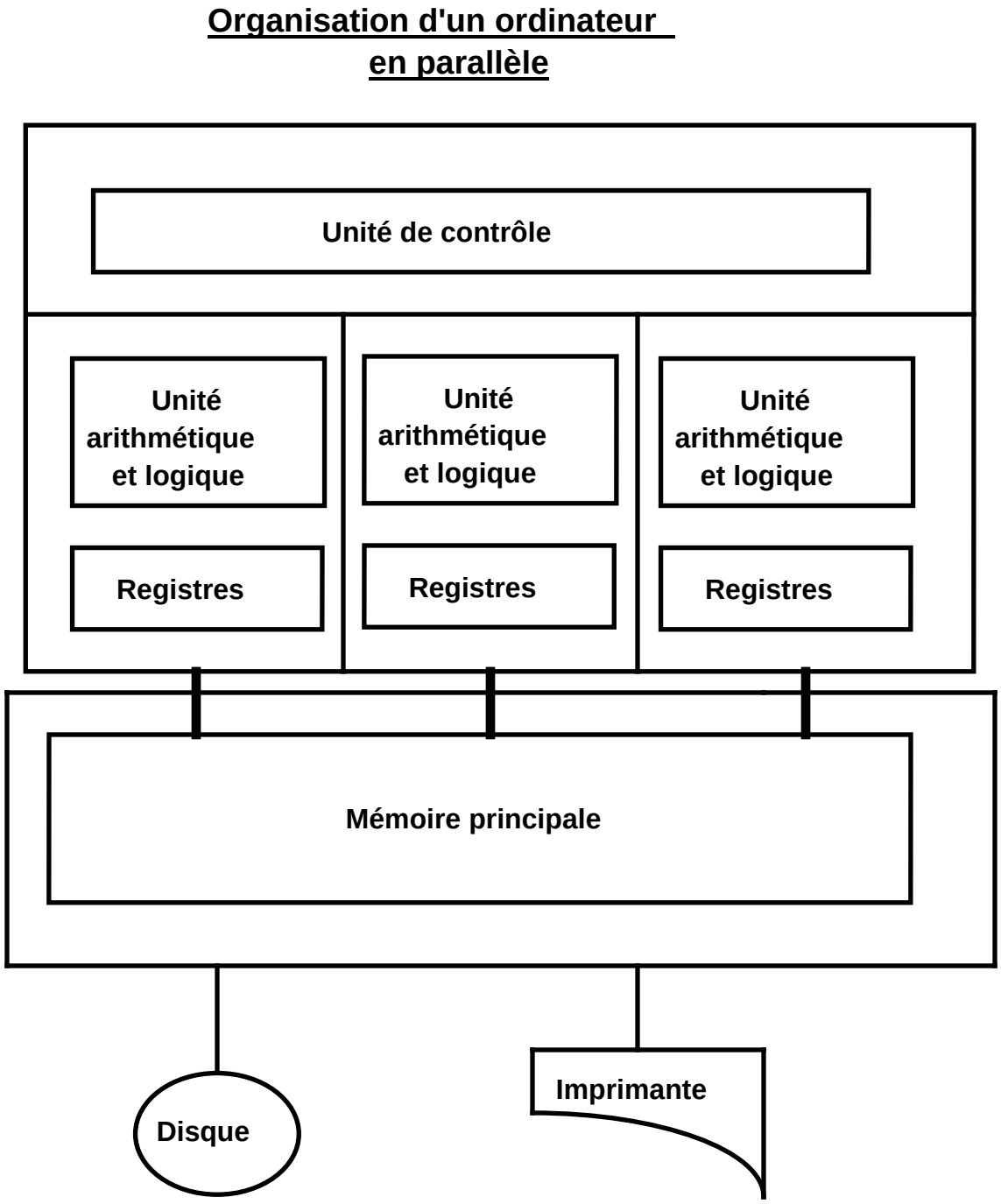
Ce concept diffère de l'architecture en parallèle qui suppose que plusieurs processeurs travaillent indépendamment mais en même temps sur des tâches identiques.

**Figure 6 : Traitement en parallèle**



Le schéma montre l'organisation interne d'un ordinateur faisant appel à plusieurs unités arithmétique et logique montées en parallèle sous le contrôle d'une même unité de commande.

**Figure 7 : Organisation d'un ordinateur en parallèle**




Enfin, on parle aussi d'architecture matricielle, qui se compose, en gros, de plusieurs processeurs, et où chaque processeur joue le rôle d'un élément d'une matrice et est interconnecté à ses processeurs voisins.

## 3. Unité arithmétique et logique (ALU)

L'ALU est la partie de l'ordinateur qui effectue la plupart des calculs. On peut résumer les opérations effectuées par l'ALU ainsi:

- Les opérations d'ordre logique comme la conjonction logique OU, la disjonction ET, la négation NON, les opérations de comparaison (< , > , = , …, etc).

- Les opérations arithmétiques comme l'addition, la soustraction, la multiplication, la division, les opérations en point flottant.

La puissance de traitement d'un ordinateur ne découle donc pas de la capacité à exécuter des opérations complexes, mais plutôt de la capacité d'exécuter un très grand nombre d'opérations simples en un court laps de temps.

Les opérations arithmétiques impliquent généralement deux nombres à l'entrée, comme par exemple les opérations de comparaison et la plupart des opérations logiques. On appelle opérandes les deux nombres impliqués dans une opération parce qu'on agit sur eux à l'aide d'un opérateur. Lorsque l'opérateur agit sur deux opérandes, on dit qu'il s'agit d'un opérateur binaire ou dyadique. S'il agit sur un seul opérande, on dit qu'il est unaire ou monoadique. La négation (NON) est un exemple d'opérateur monoadique.

### 3.1. Déroulement d'une opération dans l'ALU

L'unité de calcul n'effectue pas ses opérations directement sur les données contenues dans les emplacements de la mémoire centrale. Plutôt, elle dispose de registres où sont copiées les opérandes. L'opération terminée, le résultat est lui aussi inscrit dans un registre; ce résultat pourra éventuellement servir à une opération suivante. S'il s'agit d'un résultat final destiné à la mémoire centrale, c'est le contenu du registre qui sera copié dans une cellule de la mémoire.

Le nombre de registres dans une unité de calcul varie selon la taille et le prix de l'ordinateur. Dans l'ALU d'un petit ordinateur, il peut n'y en avoir qu'un seul. Cela est toutefois de plus en plus rare. On l'appelle alors de registre accumulateur, ou simplement accumulateur. L'accumulateur enregistre les données ou résultats instantanés d'opérations arithmétiques ou logiques. Les ALU d'ordinateurs plus importants ont plusieurs registres.

La longueur des registres varie elle aussi selon la taille et la puissance de l'ordinateur. Souvent, ils sont de longueur simple, c'est à dire de la même longueur que les mots mémoire. On retrouve aussi plusieurs registres à double longueur.

Les registres de l'ALU doivent obligatoirement avoir une très grande vitesse de fonctionnement pour ne pas ralentir le rythme de travail de l'unité.

### 3.2. n exemple du fonctionnement de l'ALU

Les figures suivantes montrent une unité de calcul à 4 registres et l'état de la mémoire principale. L'opération à effectuer (instruction) est:

$$[29] \leftarrow [12] + [18]$$

où [ n ] désigne le contenu de la cellule de mémoire d'adresse n. L'opération consiste donc à additionner les nombres contenus dans les emplacements 12 et 18 de la mémoire, et à inscrire le résultat de l'opération dans l'emplacement 29.

Nous verrons plus loin comment peuvent être codées les instructions à l'interne. Pour simplifier, on suppose que les registres R2 et R3 sont réservés aux opérandes, R1 aux instructions et R4 au résultat final.

**Figure 8 : État initial avant le début de l'opération**

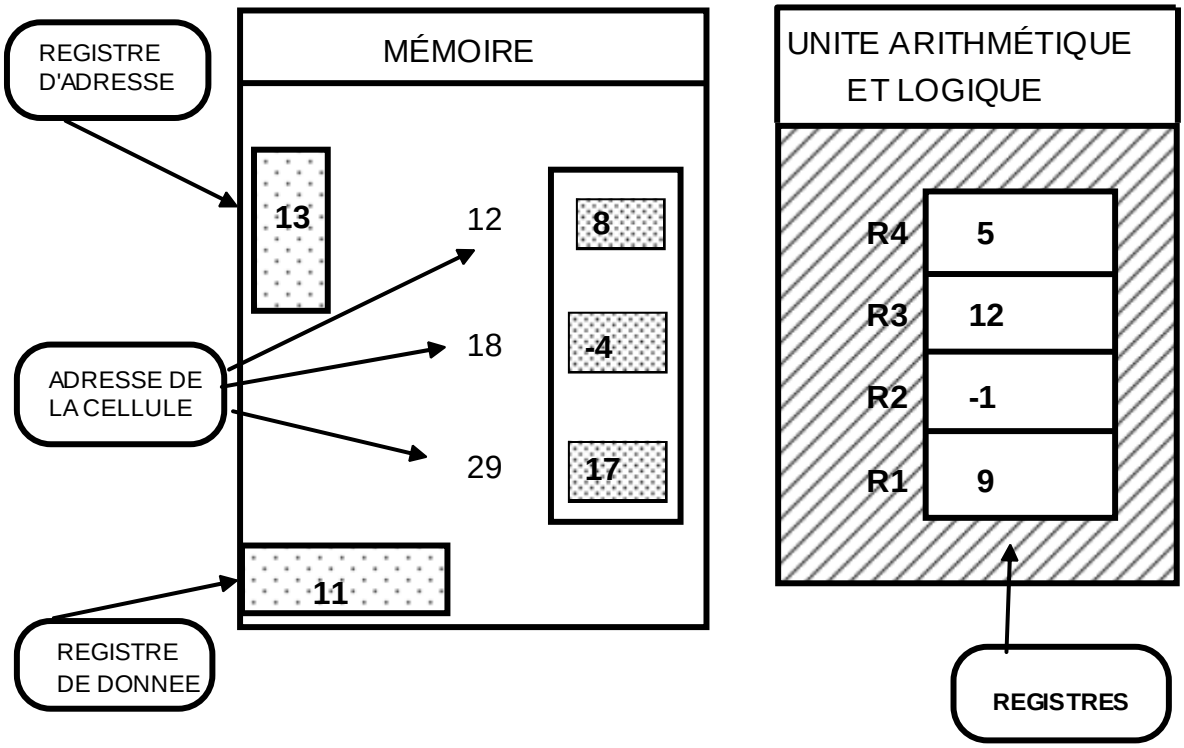


**Figure 9 : L'instruction est copiée dans le registre R1 de l'ALU**

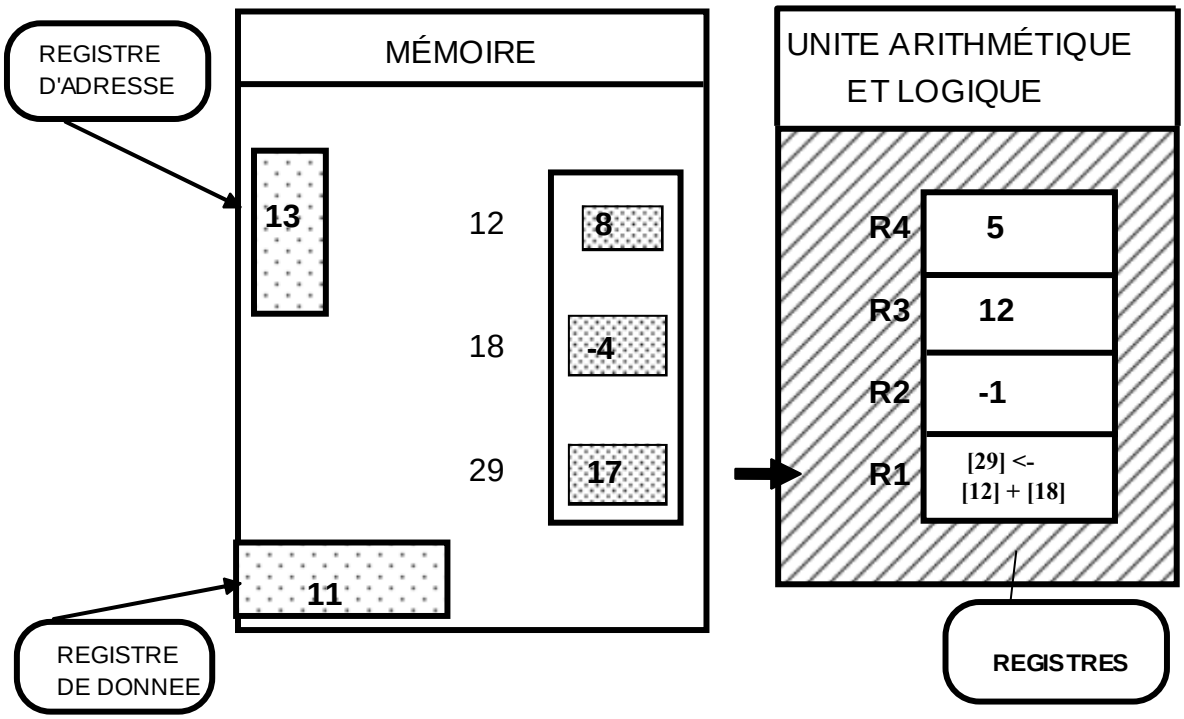



**Figure 10 : Le premier opérande est copié dans le registre R2 de l'ALU**

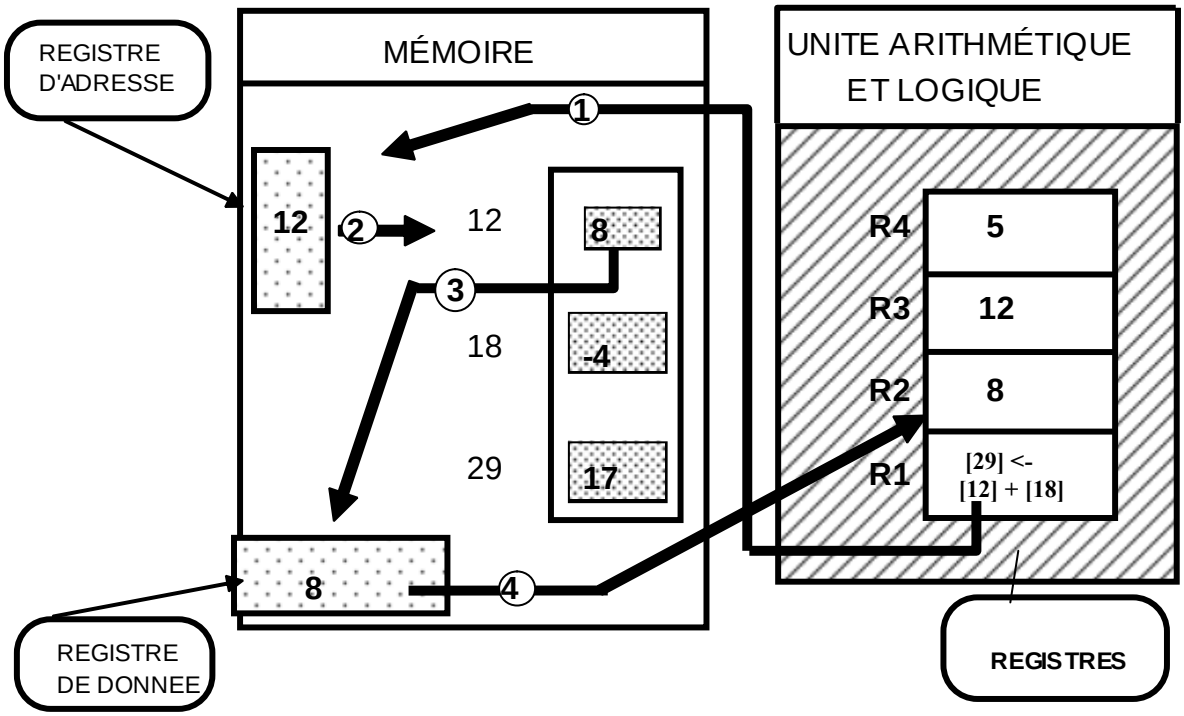


**Figure 11 : Le 2e opérande est copié dans le registre R3 de l'ALU**

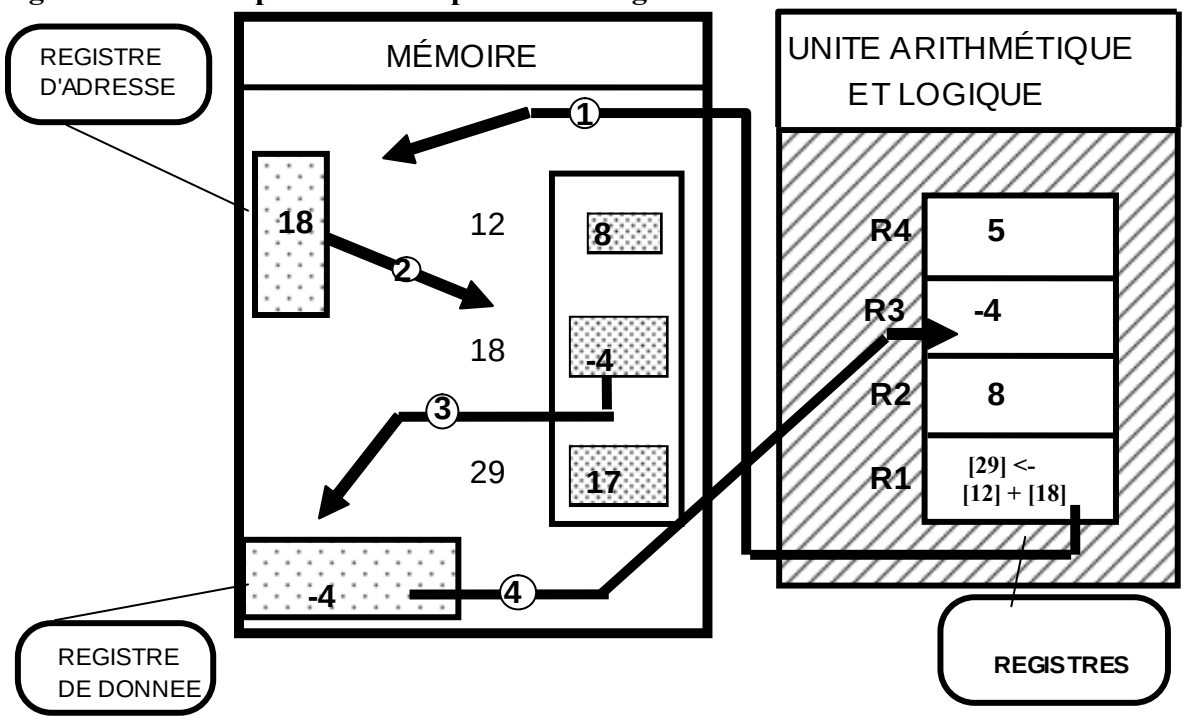



**Figure 12 : L'addition est effectuée. Le résultat va dans le registre R4**

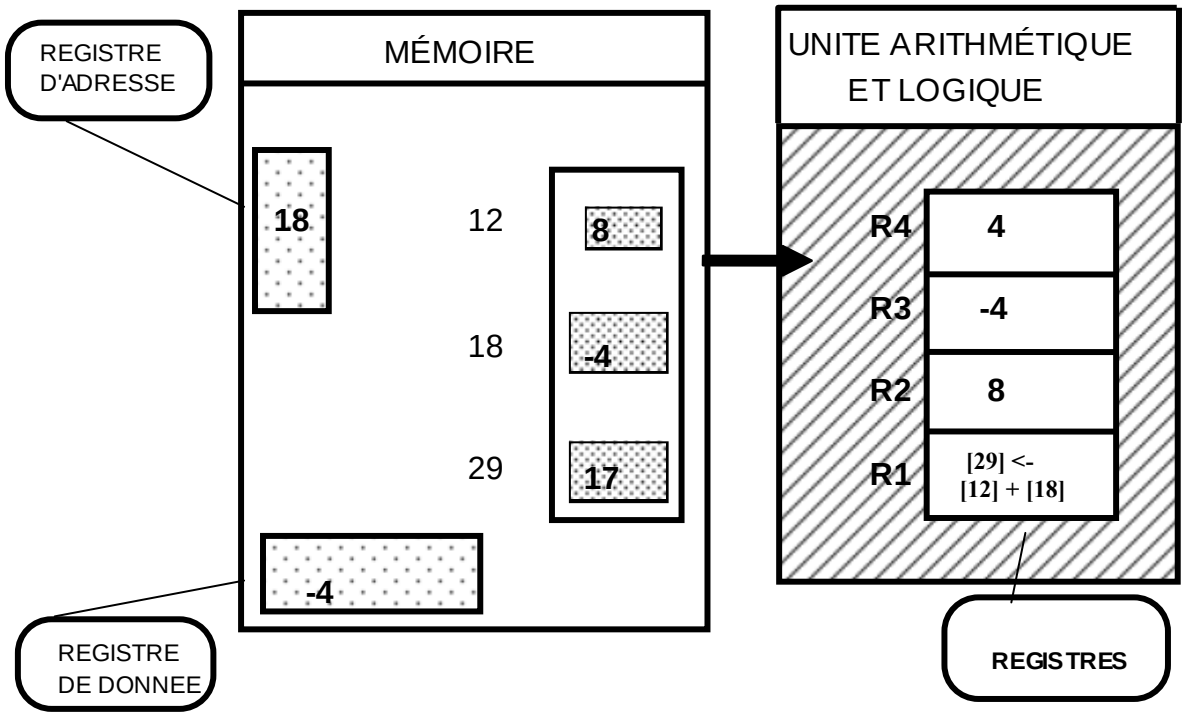
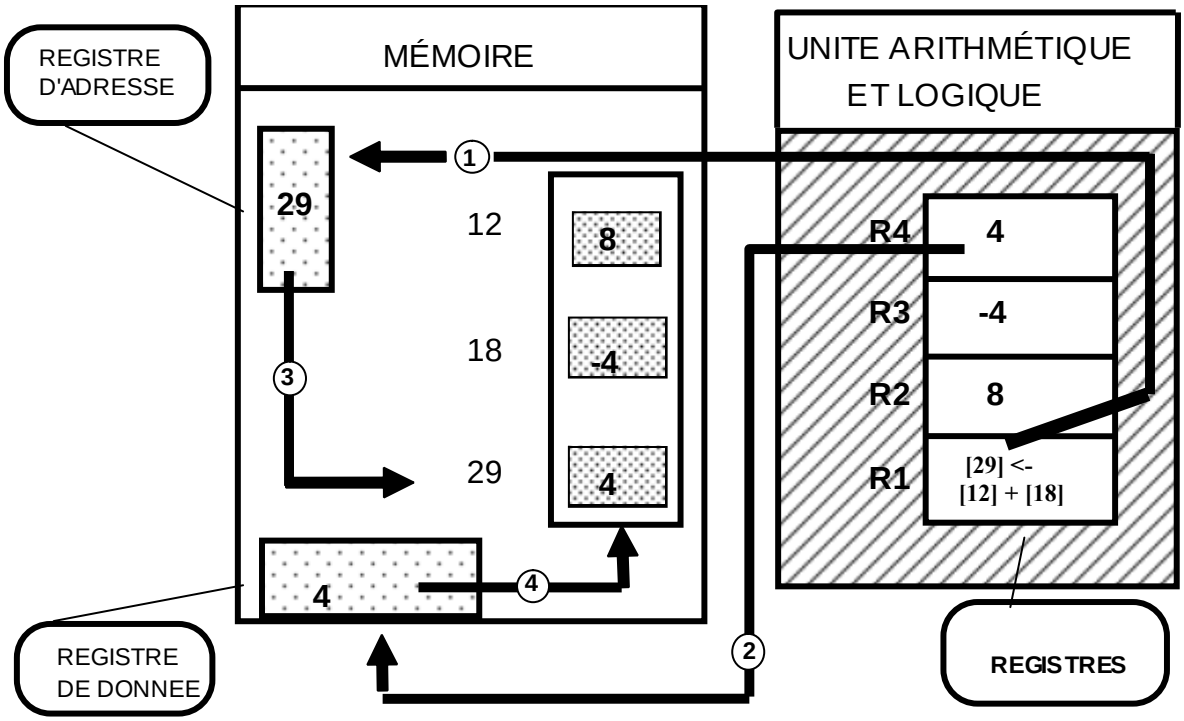


**Figure 13 : Le résultat est copié dans l'emplacement 29 de la mémoire:**




## 4. Unité de contrôle

Jusqu'à présent, nous avons vu que la mémoire principale sert à conserver les instructions des programmes à exécuter et les données nécessaires, et que l'unité arithmétique et logique exécute un certain nombre d'opérations logiques et arithmétiques. Le rôle de l'unité de contrôle (ou unité de commande ) est de coordonner le travail de toutes les autres unités et d'assurer la direction de l'ensemble. L'unité de contrôle est donc "la patronne" de l'ordinateur.

C'est l'unité de contrôle qui se charge d'appeler les instructions, de les décoder, de rechercher les opérandes et qui s'assure que les différentes phases de l'instruction sont exécutées au bon moment par l'unité concernée.

Un programme comporte des indications très précises, entre autres:

- quelles opérations effectuer
- sur quels opérandes effectuer ces opérations
- dans quel ordre effectuer les opérations

L'unité de contrôle veille à ce que ce que l'exécution du programme se fasse en respectant ces indications.

Le travail de l'unité de contrôle est en fait la répétition continuelle des trois étapes suivantes:

1. Chercher la prochaine instruction en mémoire
2. Exécuter cette instruction en envoyant les signaux appropriés aux autres unités de l'ordinateur
3. Retourner à l'étape 1.

Afin de bien comprendre le travail de l'unité de contrôle, nous allons devoir regarder de plus près les instructions machine, puisque ce sont les instructions qui "alimentent" ce travail. Mais d'abord, regardons d'un peu plus près les tâches que doit effectuer l'unité de contrôle, et les outils dont elle dispose.

### 4.1 Exécution d'une instruction

Le cycle d'exécution d'une instruction peut être résumé ainsi:

1. Appel d'une instruction
2. Décodage de l'instruction reçue
3. Recherche des opérandes impliqués dans l'instruction
4. Exécution de l'instruction
5. Retour à l'étape 1.

Le même cycle recommence jusqu'à l'épuisement des instructions du programme.

### 4.2. Action de la prochaine instruction

Une des tâches de l'unité de contrôle est d'extraire les instructions de la mémoire. Mais encore faut-il trouver la bonne instruction, c'est à dire celle qui respecte l'ordre indiqué par le programme. Pour ce faire, l'unité de contrôle dispose d'un registre spécial, appelé compteur ordinal. Le compteur ordinal contient l'adresse de l'instruction à extraire de la mémoire. Comme les instructions d'un programme sont le plus souvent stockées dans des cellules de mémoires d'adresses consécutives, il suffit d'initialiser le compteur ordinal à l'adresse de la première instruction du programme et de l'incrémenter à chaque instruction, sauf avis contraire, pour connaître l'adresse de l'instruction suivante.

Les premiers ordinateurs ne disposaient pas de compteur ordinal. Il fallait alors inclure l'adresse de la prochaine instruction dans chaque instruction, ce qui augmentait la longueur des instructions.
Une fois extraite de la mémoire, l'instruction est conservée dans un autre registre de l'unité de contrôle, le registre d'instruction. Le compteur ordinal est ensuite modifié afin qu'il pointe sur l'instruction suivante, soit en étant incrémenté, ou encore par un branchement demandé par l'instruction en cours. Notons que le compteur ordinal ne peut contenir qu'une adresse.

### 4.3. Décodage de l'instruction et recherche des opérandes

Une fois extraite de la mémoire et copiée dans le registre d'instruction de l'unité de contrôle, l'instruction est alors décodée. En effet, les instructions du programme sont stockées en mémoire sous forme de chaînes binaires. Cette transformation a été faite par le compilateur et l'éditeur de liens qui ont produit la version exécutable ne comportant que du code binaire. Notons que les données sont elles aussi stockées sous forme de chaînes binaires. Cela signifie que:

1. une cellule de mémoire peut contenir aussi bien une donnée qu'une instruction.

2. en mémoire, il n'y a pas vraiment de différence entre une donnée, une instruction ou une adresse d'opérande, puisque tout est sous forme de chaîne binaire. C'est la façon dont la chaîne binaire est décodée qui fait la différence.

Il s'agit là d'une des caractéristiques fondamentales des ordinateurs de "Von Neumann", dont tous les ordinateurs modernes sont. On parle d'interchangeabilité des instructions et des données.

L'unité de contrôle comporte donc un décodeur d'instruction qui lui permet de décoder l'instruction. Comme nous le verrons lorsque nous regarderons le format des instructions, la chaîne binaire que forme une instruction comporte un code d'opération et l'adresse des opérandes impliqués de l'opération. C'est pourquoi le décodeur d'instruction se compose en fait d'un décodeur d'opération qui permet d'identifier l'opération à effectuer et d'un décodeur d'adresse qui sert à retrouver les opérandes impliqués dans l'instruction à partir de leurs adresses.

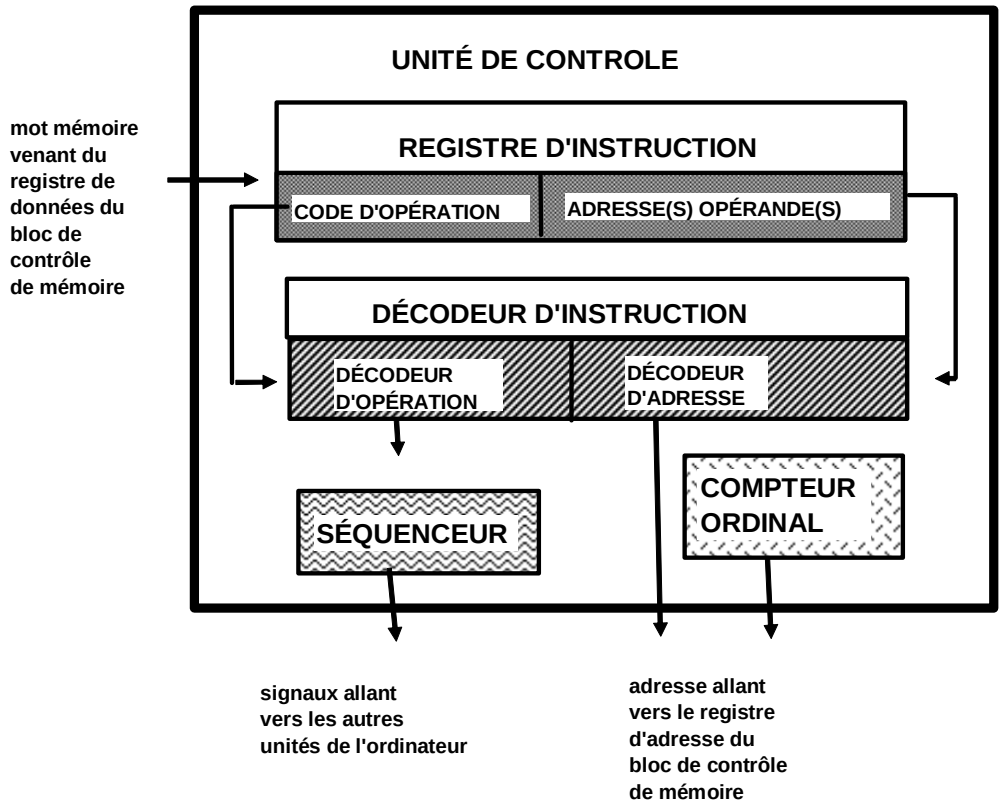
### 4.4. Exécution de l'instruction

Une fois l'opération et les adresses des opérandes décodées, l'unité de contrôle doit veiller à ce que l'instruction soit exécutée. Il faut pour cela coordonner le travail des autres unités de l'ordinateur impliquées. Des signaux sont envoyés par le séquenceur. Ces signaux indiquent à l'unité concernée qu'elle a un travail à accomplir. Il peut s'agir, par exemple, d'avertir la mémoire qu'il faut effectuer la lecture d'une cellule, ou de prévenir l'ALU qu'il faut effectuer l'addition de deux données, etc. Les signaux servent aussi de "Q" (signal de "timing") pour que le travail soit effectué au moment précis où il doit l'être. La séquence des différents signaux est déterminée par rapport à l'horloge interne.

#### 4.4.1. Schéma d'une unité de contrôle.

Le schéma suivant montre les éléments centraux de l'unité de contrôle de l'ordinateur classique ainsi que la source et la destination des informations qu'il traite. Les flèches montre la direction des échanges entre ces différents éléments.

**Figure 144**


UNITÉ DE CONTROLE
mot mémoire venant du registre de données du bloc de contrôle de mémoire
REGISTRE D'INSTRUCTION
CODE D'OPÉRATION
ADRESSE(S) OPÉRANDE(S)
DÉCODEUR D'INSTRUCTION
DÉCODEUR D'OPÉRATION
DÉCODEUR D'ADRESSE
SÉQUENCEUR
COMPTEUR ORDINAL
signaux allant vers les autres unités de l'ordinateur
adresse allant vers le registre d'adresse du bloc de contrôle de mémoire

## 4.5. Les instructions machine.

L'ordinateur exécute des instructions codées en langage machine. La syntaxe de ce langage permet de constituer des chaînes de valeurs binaires qui sont analysées puis exécutées par l'unité centrale. Le langage machine est le langage formé des instructions constituant le répertoire d'une machine donnée. C'est donc dire que chaque machine a son langage machine.

Le langage machine est le seul langage directement assimilable par un ordinateur.

Les difficultés de programmation directe en langage machine ont cependant conduit à concevoir des langages s'éloignant peu à peu de la machine réelle: langages d'assemblage d'abord (de bas niveau), jusqu'aux langages très évolués (FORTRAN, COBOL, PASCAL...). Ces langages évolués sont traduits par des compilateurs et des éditeurs de lien pour former des versions exécutables en langage machine, donc des suites d'instructions machine.

Les instructions machine sont réparties selon les types suivants:

- instructions de traitement ayant pour fonction d'effectuer les opérations arithmétiques et logiques
- instructions d'échange permettant le transfert des données entre la mémoire et les unités périphériques
- instructions de rupture de séquence ayant pour rôle de modifier la séquence de déroulement des instructions.

En général les instructions renferment les informations suivantes:

- le code opération
- un ou des opérandes
- l'endroit où doit être rangé le résultat.

Elles peuvent aussi indiquer l'adresse de la prochaine instruction.

Si le langage machine peut être différent d'une machine à une autre, c'est que certaines caractéristiques des instructions peuvent varier, entre autres:

- la longueur de l'instruction (nombre de bits)
- le nombre d'opérandes
- la longueur de chaque opérande
- la longueur du code d'opération
- la nature des opérations possibles

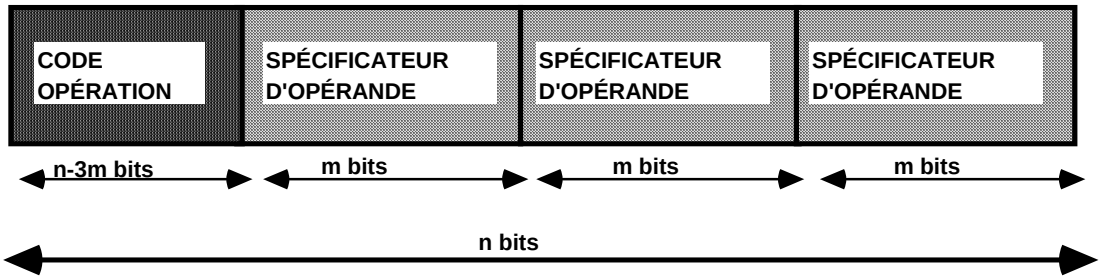
Le constructeur de la machine doit donc faire plusieurs choix. Ces choix sont en relation avec d'autres choix qui déterminent d'autres caractéristiques d'une machine, par exemple la capacité de la mémoire, la longueur des mots mémoire, le débit binaire des unités, etc. Ainsi, la longueur de l'instruction détermine le nombre de mots machine qui seront nécessaires pour la stocker.

La façon dont le nombre de bits qui forment l'instruction est réparti entre le code opération et les opérandes est aussi capitale: chaque opérande doit avoir une longueur suffisante pour que toutes les cellules de mémoire soient inaccessibles et la longueur du code d'opération détermine le nombre d'instructions disponibles. Nous verrons quelles sont les répercussions de cette répartition des bits sur la puissance et la capacité d'une machine.

Enfin, le nombre d'opérandes peut varier, ce qui suppose des différences de fonctionnement considérables. Nous examinerons des exemples d'instructions à 4, 3, 2, 1 et même 0 opérandes. Notons qu'une même machine peut avoir plusieurs formats d'instruction différents. Le VAX/780, par exemple, renferme dans son répertoire des instructions allant de 0 à 6 opérandes.

### 4.5.1. Format général d'une instruction.

Le schéma suivant montre le format général d'une instruction à trois opérandes




.

Cette instruction comporte un code d'opération, et 3 "spécificateurs d'opérande". Une instruction peut en contenir 0, 1 ou plusieurs spécificateurs d'opérande, en général des adresses d'opérande. C'est donc dire que cette instruction contient 3 adresses et un code d'opération. Toutes les adresses doivent avoir la même longueur (m bits dans le schéma). Si l'instruction est codée sur n bits, il reste donc:

n bits - 3m bits (l'espace occupé par les adresses) pour le code d'opération.

Cela signifie que:

- si chaque adresse est codée sur m bits, une adresse peut faire référence à $2^m$ cellules différentes (en mode d'adressage direct), soient les cellules dont l'adresse est comprise entre 0 et $2^{m-1}$.
- si le code opération est codé sur k bits (n - 3m bits), l'ordinateur pourra avoir $2^k$ opérations différentes.

### 4.5.2. Répertoire d'instructions.

L'ensemble des instructions qu'une machine peut exécuter constitue son répertoire d'instructions ou jeu d'instructions . Le choix des instructions et leur nombre dépend naturellement du type de travail que l'ordinateur aura à exécuter: plus le travail est diversifié et complexe, plus la liste a de

chances d'être longue et l'ordinateur coûteux car l'implantation de la plupart des opérations est faite de façon câblée (électroniquement).

La longueur du code opération détermine le nombre d'opérations différentes qui constituent le répertoire. Plus le code opération est court, plus le répertoire sera limité. Il doit cependant contenir au moins une instruction pour

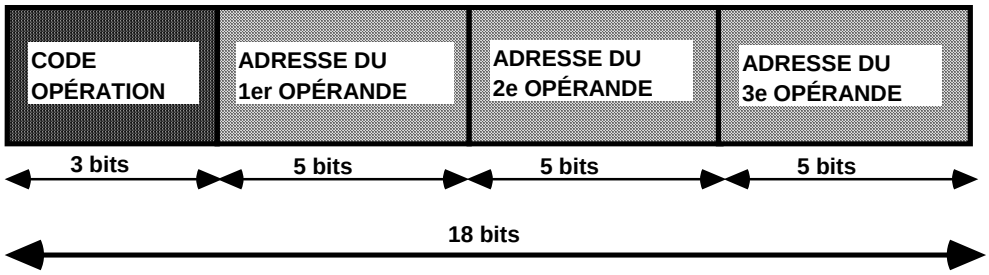
- communiquer avec l'extérieur
- entrer et sortir des données
- effectuer une rupture de séquence ou un test

À l'inverse, plus le code opération est long, plus le répertoire d'instruction de la machine est grand, ce qui permet d'ajouter au répertoire de base d'autres instructions plus sophistiquées, et par le fait-même, d'en accroître la complexité.

Le constructeur se retrouve ici encore devant des choix à faire: il doit trouver un équilibre entre la sophistication et la simplicité. Ces choix ont des répercussions énormes sur l'architecture de la machine et sur son coût. En effet, le choix des instructions et leur nombre fixe de façon définitive la structure de l'unité de contrôle et celle de l'unité de calcul; on ne peut modifier le répertoire d'instructions une fois l'ordinateur construit.

### 4.5.2.1. Exemple d'un répertoire d'instructions.

Voici un répertoire d'instructions très simple. Les instructions de cette machine fictive sont codées sur 18 bits répartis de la façon suivante:




.

Les 3 bits du code d'opération permettent de disposer de 8 instructions différentes. Le répertoire pourrait être le suivant:

**Tableau 1**

| CODE | OPÉRATION | MNÉMONIQUE | SIGNIFICATION |
|---|---|---|---|
| 000 | Addition | ADD | C ← (A) + (B) |
| 001 | Soustraction | SOU | C ← (A) - (B) |
| 010 | Initialisation | INI | Si A ≠ 0 alors A ← C |
| | | | Si B ≠ 0 alors B ← 1 |
| | | | Si C ≠ 0 alors C ← 2 |
| 011 | Rupture Conditionnelle | EGU | Si (A) = (B) alors |
| | | | compteur ordinal ← C |
| 100 | Rupture Inconditionnelle | RUP | compteur ordinal ← C |
| 101 | Lecture | LIR | Lire un nombre octal et le ranger dans A; |
| | | | si B ≠ 0 en lire un second |
| | | | si C ≠ 0 en lire un troisième |
| 110 | Écriture | ECR | Imprimer (A); |
| | | | si B ≠ 0 alors imprimer (B) |
| | | | si C ≠ 0 alors imprimer (C) |
| 111 | Arrêt | ARR | Ne plus exécuter d'instructions |

Dans ce tableau, la première colonne donne le code d'opération binaire (sur 3 bits); chaque opération est symbolisée par le mnémonique correspondant. Les lettres A, B et C désignent des adresses symboliques et (A), (B) et (C) désignent le contenu des cellules d'adresses A, B et C (les adresses effectives). Les affectations sont représentées par ← : C ← (A) signifie de placer à l'adresse C le contenu de la cellule d'adresse A.

**Remarque:** La longueur en bits de l'instruction , des opérandes et du code opération définissent déjà notre ordinateur simulé. En effet, sa mémoire ne pourra contenir plus de 25 = 32 mots (en mode d'adressage direct). La longueur de ces mots devra être de 18 bits si on veut qu'un mot puisse contenir une instruction complète. Enfin, son répertoire d'instruction ne peut contenir ni moins, ni plus que 8 instructions différentes.

### 4.5.2.2. Exemple d'une instruction machine et de son exécution.

Référons-nous à l'instruction déjà rencontrée au chapitre précédent, lorsque nous avons étudié le fonctionnement de l'ALU.

$$[29] \leftarrow [12] + [18]$$

Il s'agissait de faire l'addition du contenu de la cellule d'adresse 12 et du contenu de la cellule d'adresse 18 et de ranger le résultat dans la cellule d'adresse 29. Nous pouvons écrire cette instruction avec le répertoire d'instructions.

| **Code opération** | **Opérandes** |
|---|---|
| ADD | A, B, SOMME |

À partir des adresses des opérandes et du code opération, nous pouvons construire l'instruction machine de la façon suivante:

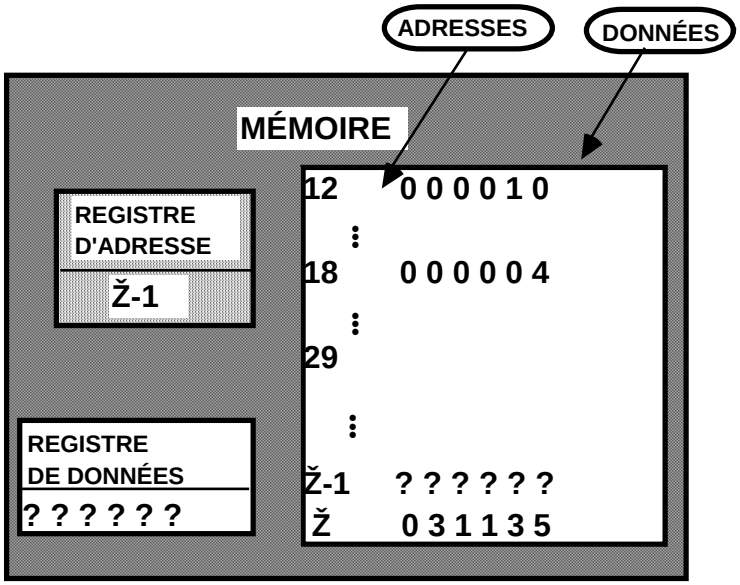
| | **ADD** | **A,** | **B,** | **Somme** | | |
|---|---|---|---|---|---|---|
| adresse décimale | | 12 | 18 | 29 | | |
| binaire | 000 | 01100 | 10010 | 11101 | | |
| | 000 | 011 | 001 | 001 | 011 | 101 |
| octal | 0 | 3 | 1 | 1 | 3 | 5 |

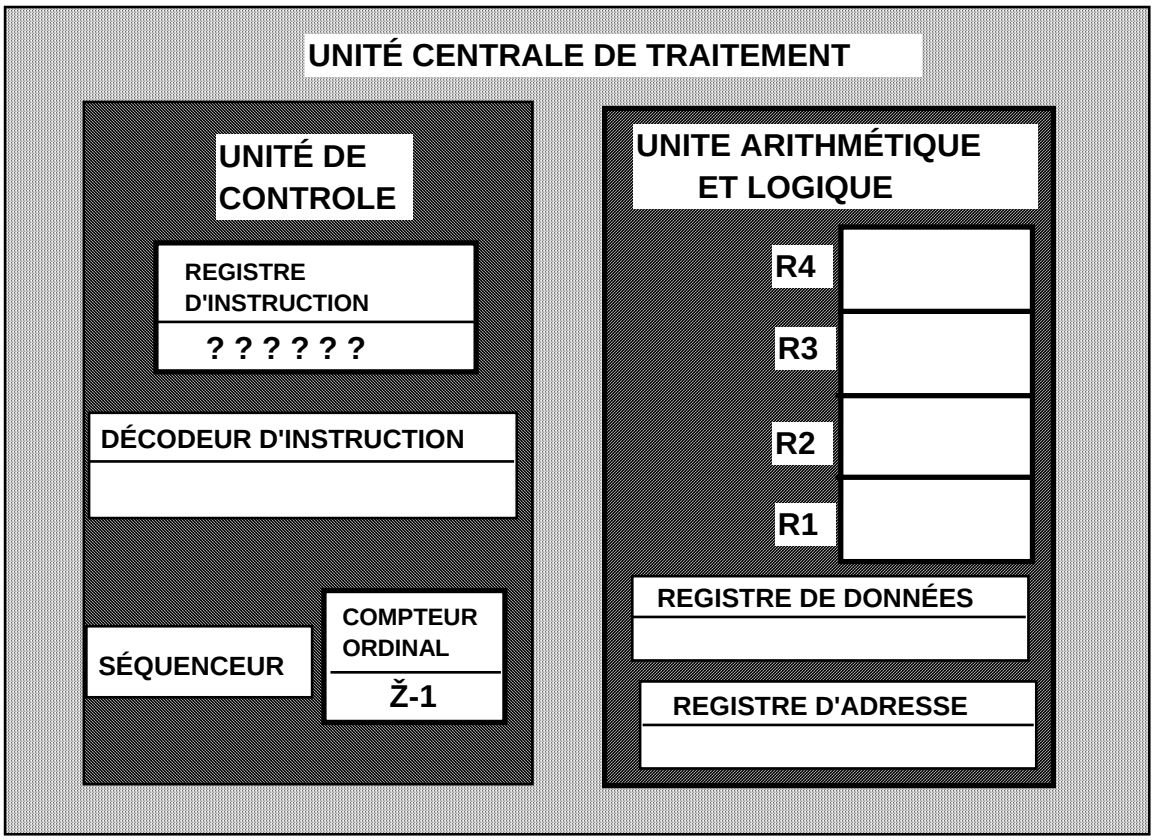
Remarque: L'instruction machine est ici codée en octal. L'instruction octale 031135 est l'équivalent de l'instruction binaire 000 011 001 001 011 101. L'octal et l'hexadécimal sont les plus utilisés pour coder les instructions machine, car les chaînes binaires pures sont longues et difficiles à manipuler.

Imaginons que l'instruction machine soit logée à l'adresse $\partial$ de la mémoire centrale. Pour que cette instruction soit exécutée, le travail de l'unité de contrôle s'effectuerait selon la séquence d'événements suivante:

Au temps $T_0$:
L'unité centrale de traitement traite l'instruction logée à l'adresse $\partial$-1 de la mémoire

**Figure 15**

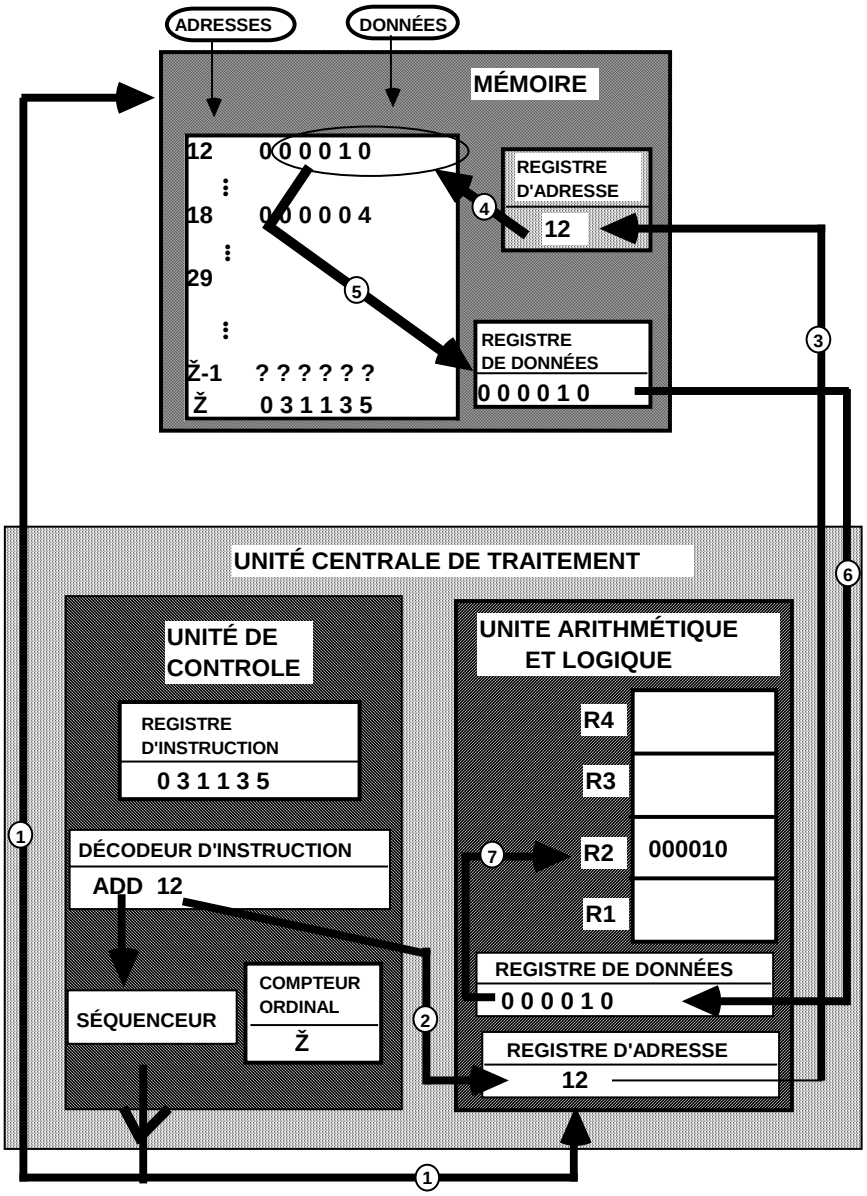

ADRESSES
DONNÉES
MÉMOIRE
REGISTRE
D'ADRESSE
Ž-1
REGISTRE
DE DONNÉES
? ? ? ? ? ?
12 0 0 0 0 1 0
18 0 0 0 0 0 4
29
Ž-1 ? ? ? ? ? ?
Ž 0 3 1 1 3 5


UNITÉ CENTRALE DE TRAITEMENT
UNITÉ DE
CONTROLE
REGISTRE
D'INSTRUCTION
? ? ? ? ? ?
DÉCODEUR D'INSTRUCTION
SÉQUENCEUR
COMPTEUR
ORDINAL
Ž-1
UNITE ARITHMÉTIQUE
ET LOGIQUE
R4
R3
R2
R1
REGISTRE DE DONNÉES
REGISTRE D'ADRESSE

Au temps T1:

- ✓ Le compteur ordinal passe à l'adresse ∂.
- ✓ L'instruction logée à l'adresse ∂ est chargée dans le registre d'instruction de l'unité de contrôle.
- ✓ L'unité de contrôle commence le décodage.
- ✓ Le séquenceur avertit les unités concernées (ALU, mémoire) et orchestre les opérations (voir 1 sur figure 15).
- ✓ Dans un premier temps, le décodeur d'adresses transmet la première adresse déchiffrée (A) à l'ALU (voir 2 sur figure 15)
- ✓ cela permettant à l'ALU de lire le contenu de l'adresse A (3, 4 et 5 sur figure 15)
- ✓ ce contenu est transféré dans le registre de données de l'ALU (6 sur figure 15)
- ✓ pour finalement être rangé dans le registre R2 de l'ALU (7 sur figure 15).

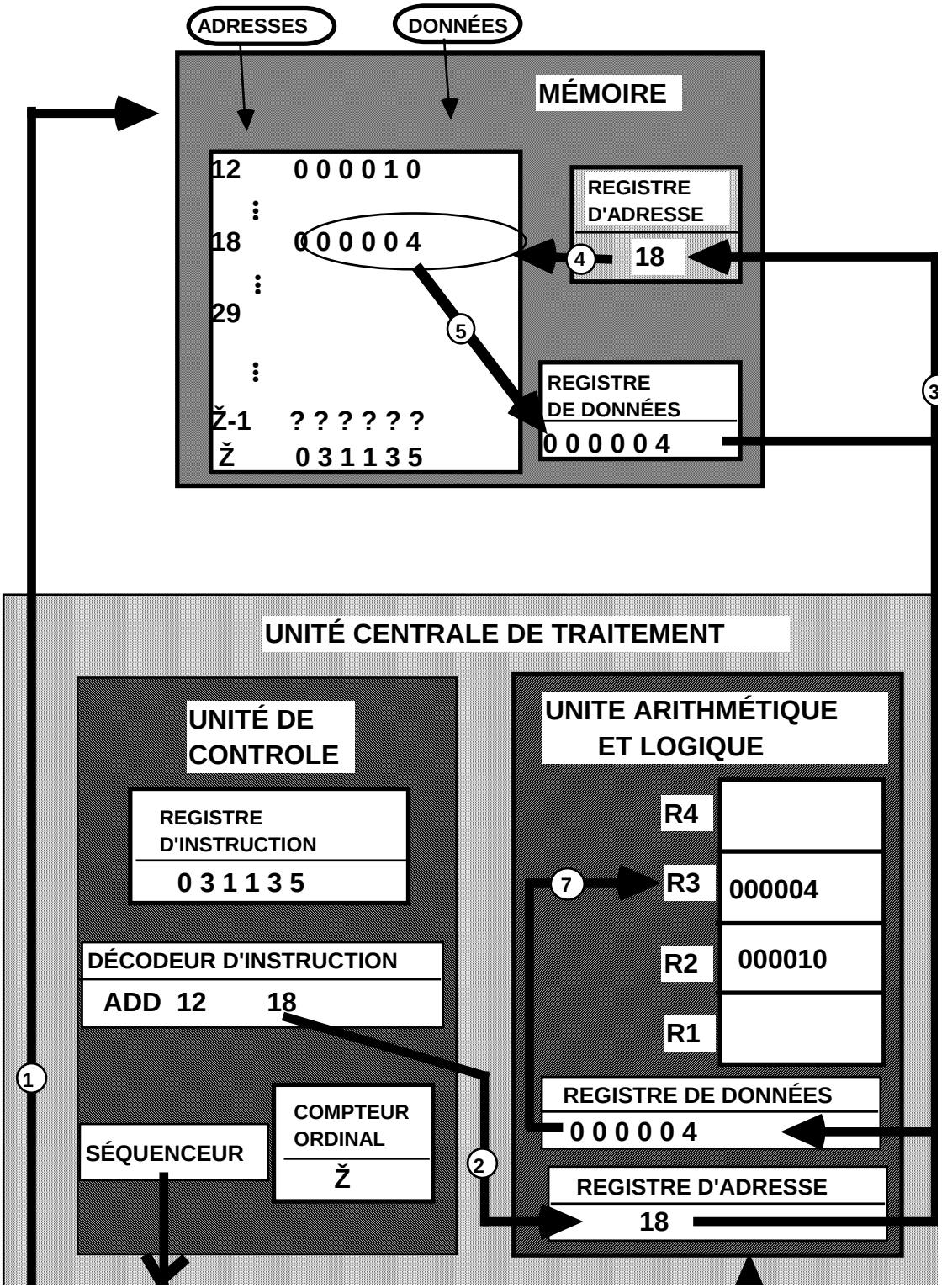
**Figure 16**


ADRESSES
DONNÉES
MÉMOIRE
12 000010
18 000004
29
Ž-1 ??????
Ž 031135
REGISTRE D'ADRESSE
12
REGISTRE DE DONNÉES
000010
UNITÉ CENTRALE DE TRAITEMENT
UNITÉ DE CONTROLE
REGISTRE D'INSTRUCTION
031135
DÉCODEUR D'INSTRUCTION
ADD 12
SÉQUENCEUR
COMPTEUR ORDINAL
Ž
UNITE ARITHMÉTIQUE ET LOGIQUE
R4
R3
R2
000010
R1
REGISTRE DE DONNÉES
000010
REGISTRE D'ADRESSE
12
1
2
3
4
5
6
7

Au temps T2:

- ✓ L'unité de contrôle coordonne le chargement du contenu de l'adresse B dans le registre R3 de l'ALU:

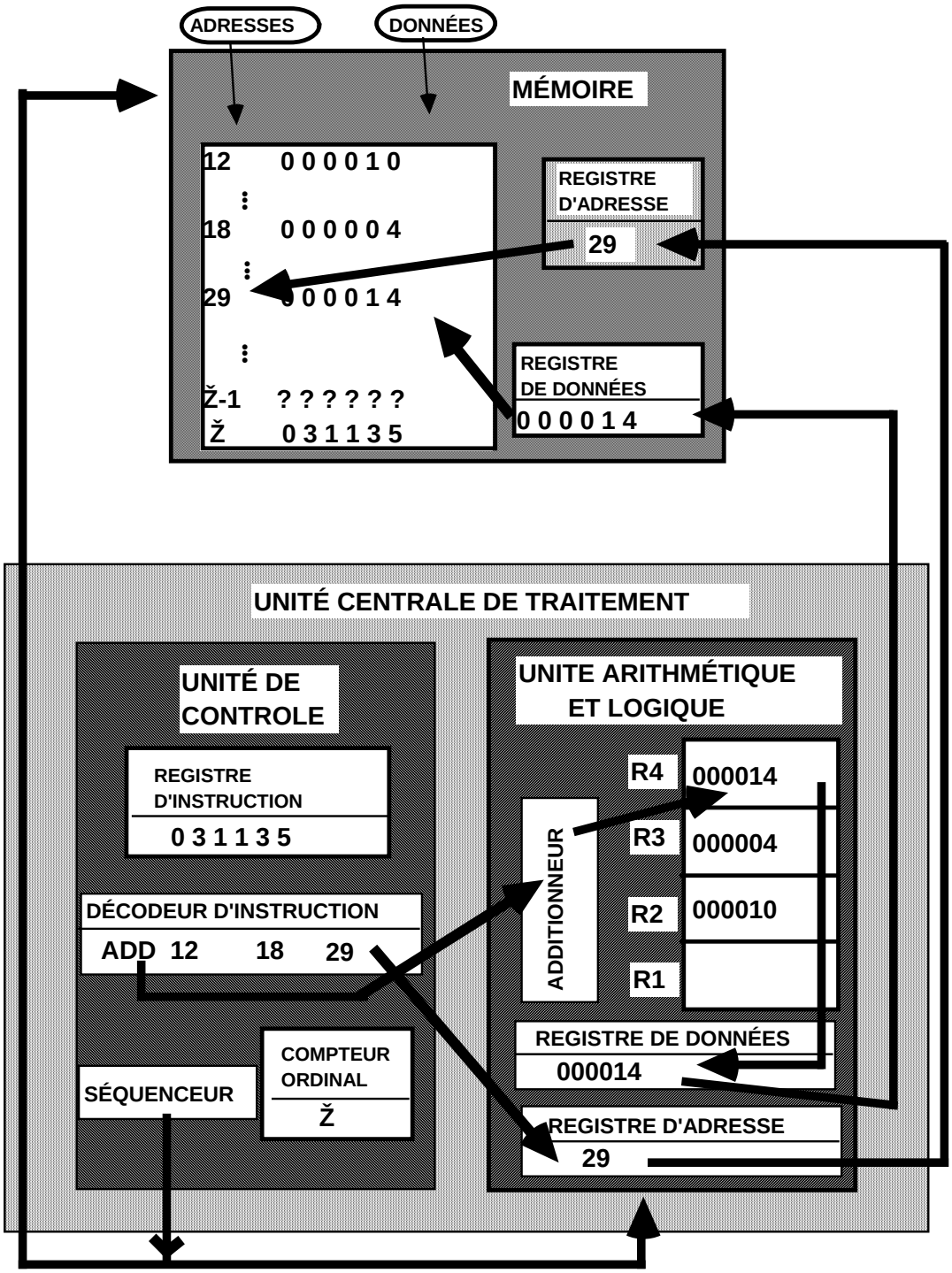
**Figure 17**


ADRESSES
DONNÉES
MÉMOIRE
12 0 0 0 0 1 0
18 0 0 0 0 0 4
29
Ž-1 ? ? ? ? ? ?
Ž 0 3 1 1 3 5
REGISTRE D'ADRESSE
18
REGISTRE DE DONNÉES
0 0 0 0 0 4
UNITÉ CENTRALE DE TRAITEMENT
UNITÉ DE CONTROLE
REGISTRE D'INSTRUCTION
0 3 1 1 3 5
DÉCODEUR D'INSTRUCTION
ADD 12 18
SÉQUENCEUR
COMPTEUR ORDINAL
Ž
UNITE ARITHMÉTIQUE ET LOGIQUE
R4
R3
000004
R2
000010
R1
REGISTRE DE DONNÉES
0 0 0 0 0 4
REGISTRE D'ADRESSE
18
1
2
3
4
5
7

Au temps T3: (voir figure 18)

- ✓ l'unité de contrôle décode l'adresse de SOMME,
- ✓ l'adresse de SOMME ainsi que le code d'opération sont transmis à l'ALU.
- ✓ L'ALU exécute l'opération à l'aide de son additionneur et range la valeur du résultat dans le registre R4,
- ✓ la valeur du résultat est copiée dans le registre de données,
- ✓ la valeur du résultat est copiée à l'adresse SOMME de la mémoire.

**Figure 18**


ADRESSES
DONNÉES
MÉMOIRE
12 0 0 0 0 1 0
18 0 0 0 0 0 4
29 0 0 0 0 1 4
Ž-1 ? ? ? ? ? ?
Ž 0 3 1 1 3 5
REGISTRE D'ADRESSE
29
REGISTRE DE DONNÉES
0 0 0 0 1 4
UNITÉ CENTRALE DE TRAITEMENT
UNITÉ DE CONTROLE
REGISTRE D'INSTRUCTION
0 3 1 1 3 5
DÉCODEUR D'INSTRUCTION
ADD 12 18 29
SÉQUENCEUR
COMPTEUR ORDINAL
Ž
UNITE ARITHMÉTIQUE ET LOGIQUE
ADDITIONNEUR
R4 000014
R3 000004
R2 000010
R1
REGISTRE DE DONNÉES
000014
REGISTRE D'ADRESSE
29

### 4.5.3. Exemple d'un programme en langage machine.

À l'aide du répertoire d'instructions du tableau 17, écrivons un programme permettant de lire à l'unité d'entrée (clavier) trois nombres entiers octaux (par exemple: 103, 411 et 1000), qu'on logera respectivement aux adresses octales 15, 16 et 21. Le résultat de l'addition sera rangé dans la cellule de mémoire d'adresse octale 22, puis livré à l'unité de sortie (écran ou imprimante). Supposons que A, B, C et TOTAL représentent symboliquement ces adresses. Voici le programme effectuant ce problème:

| **OPÉRATIONS** | **INSTRUCTIONS MACHINE** (en octal) |
|---|---|
| LIR A, B, C | 532721 |
| ADD A, B, TOTAL | 032722 |
| ADD TOTAL, C, TOTAL | 045062 |
| ECR 0, 0, C | 600022 |
| ARR 0, 0, 0 | 700000 |

La séquence revient à :

- Lire à l'unité d'entrée 3 nombres et les loger respectivement aux adresses A, B et C.
- Additionner le contenu de la cellule d'adresse 15 à celui de la cellule d'adresse 16 et de placer le résultat à la cellule d'adresse 22.
- Additionner le contenu de la cellule 22 à celui de la cellule 21 et placer le résultat à la cellule 22.
- Écrire à l'unité de sortie le nombre situé à l'adresse TOTAL.
- Arrêter

Ici encore, les instructions machine sont codées en octal. Les instructions en octal sont déduites à partir des adresses et du code d'opération en binaire, comme nous l'avons vu dans l'exemple précédent, ce qui donne, pour la deuxième instruction de notre programme:

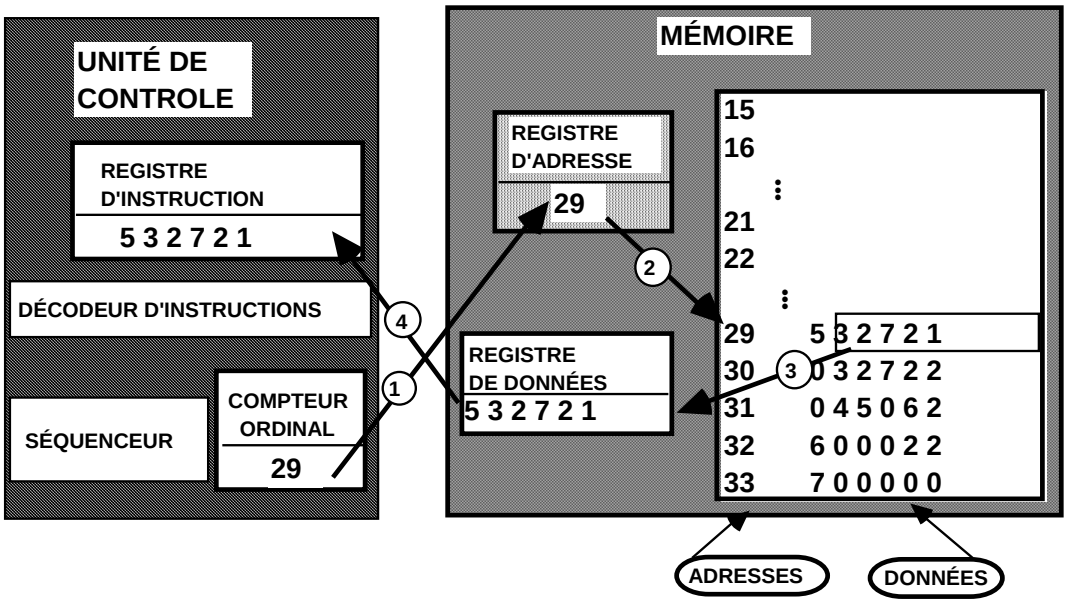
| | **ADD** | **A1,** | **A2** | **Somme** | | |
|---|---|---|---|---|---|---|
| adresse octale | | **15** | **15** | **22** | | |
| adresse décimale | | 13 | 14 | 18 | | |
| binaire | 000 | 01101 | 01110 | 10010 | | |
| | 000 | 011 | 010 | 111 | 010 | 010 |
| octal | 0 | 3 | 2 | 7 | 2 | 2 |

L'instruction 032722 en octal est l'équivalent de l'instruction binaire 0000110101110100 10. Les autres instructions octales sont déduites de la même façon.

### 4.5.4. Séquence d'exécution.

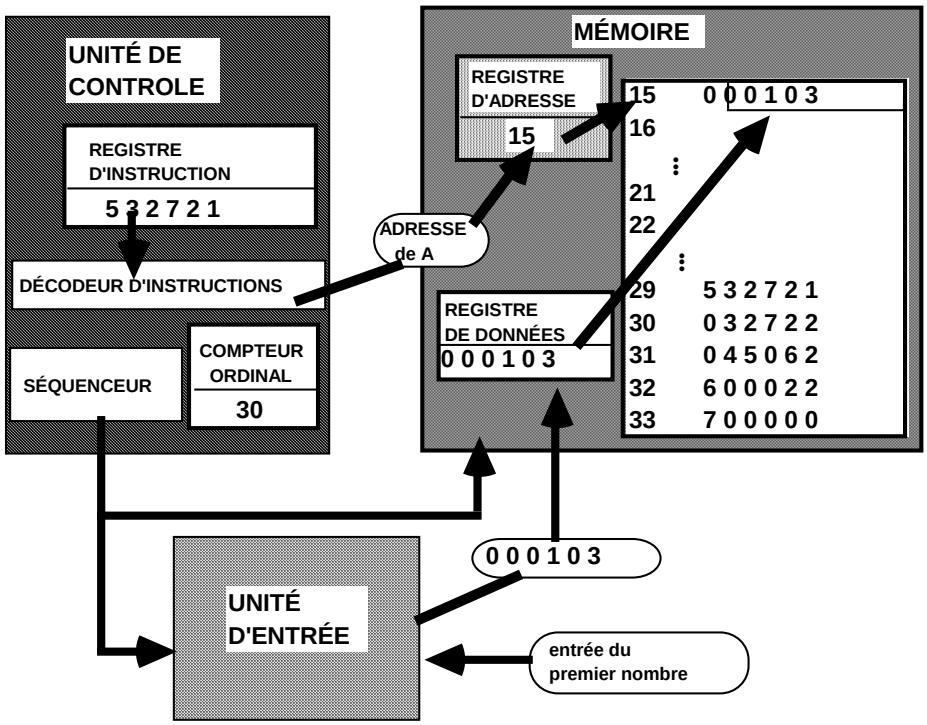
Pour des fins d'illustration, nous supposons que notre programme a été chargé en mémoire et qu'il occupe les adresses consécutives 27 à 33. Au début du programme, le compteur ordinal est initialisé à l'adresse de la première instruction. Cette adresse est copiée dans le registre d'adresse de la mémoire et l'instruction contenue dans la cellule de mémoire correspondante est copiée au registre de données de la mémoire, et de là, au registre d'instruction de l'unité de contrôle, comme le montre la figure suivante:

**Figure 19**




L'instruction est ensuite décodée. L'opération à effectuer, les adresses des opérandes et les signaux de synchronisation appropriés sont envoyés à l'unité d'entrée et à la mémoire. Le premier nombre est lu et rangé en mémoire.

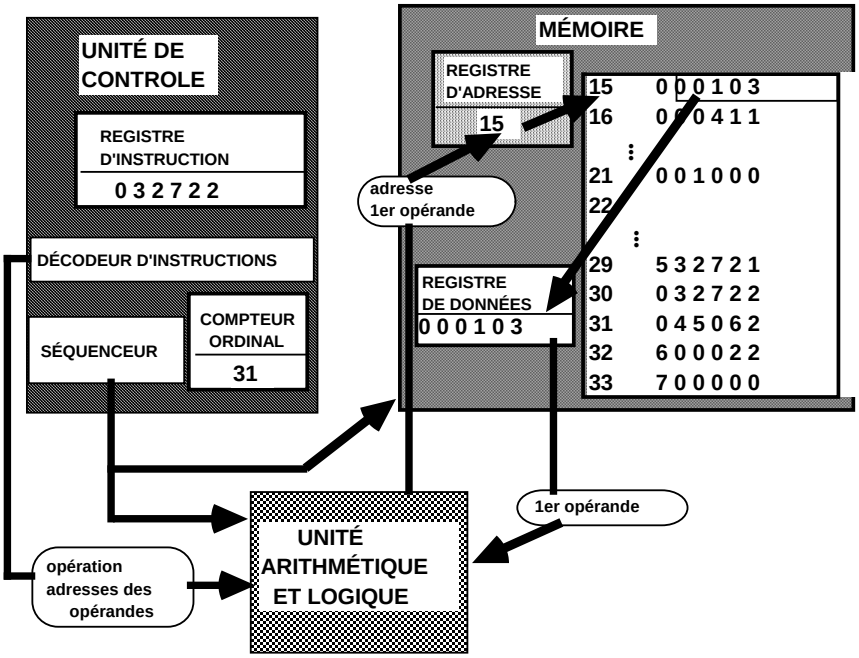
**Figure 20**


UNITÉ DE CONTROLE
REGISTRE D'INSTRUCTION
5 3 2 7 2 1
DÉCODEUR D'INSTRUCTIONS
SÉQUENCEUR
COMPTEUR ORDINAL
30
ADRESSE de A
MÉMOIRE
REGISTRE D'ADRESSE
15
REGISTRE DE DONNÉES
0 0 0 1 0 3
15 0 0 0 1 0 3
16
21
22
29 5 3 2 7 2 1
30 0 3 2 7 2 2
31 0 4 5 0 6 2
32 6 0 0 0 2 2
33 7 0 0 0 0 0
0 0 0 1 0 3
UNITÉ D'ENTRÉE
entrée du premier nombre

Une fois que les trois nombres sont lus et rangés en mémoire de la même façon, la deuxième instruction (ADD A,B, TOTAL) est copiée dans le registre d'instruction de l'unité de contrôle et décodée à son tour. Des signaux de synchronisation sont envoyés à l'ALU et à la mémoire.

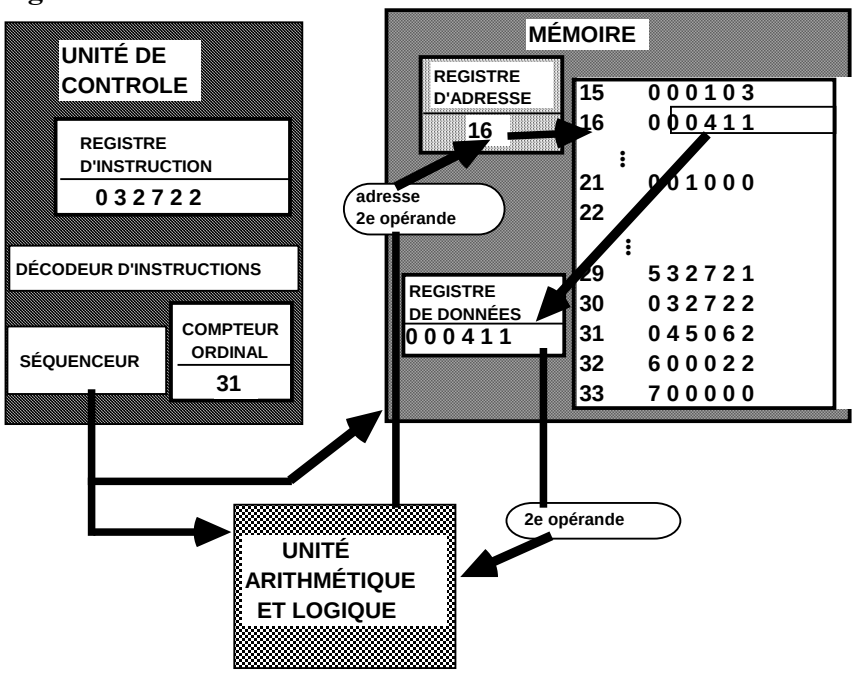
L'ALU va chercher le premier opérande en mémoire et le copie dans un de ses registres:

**Figure 21**



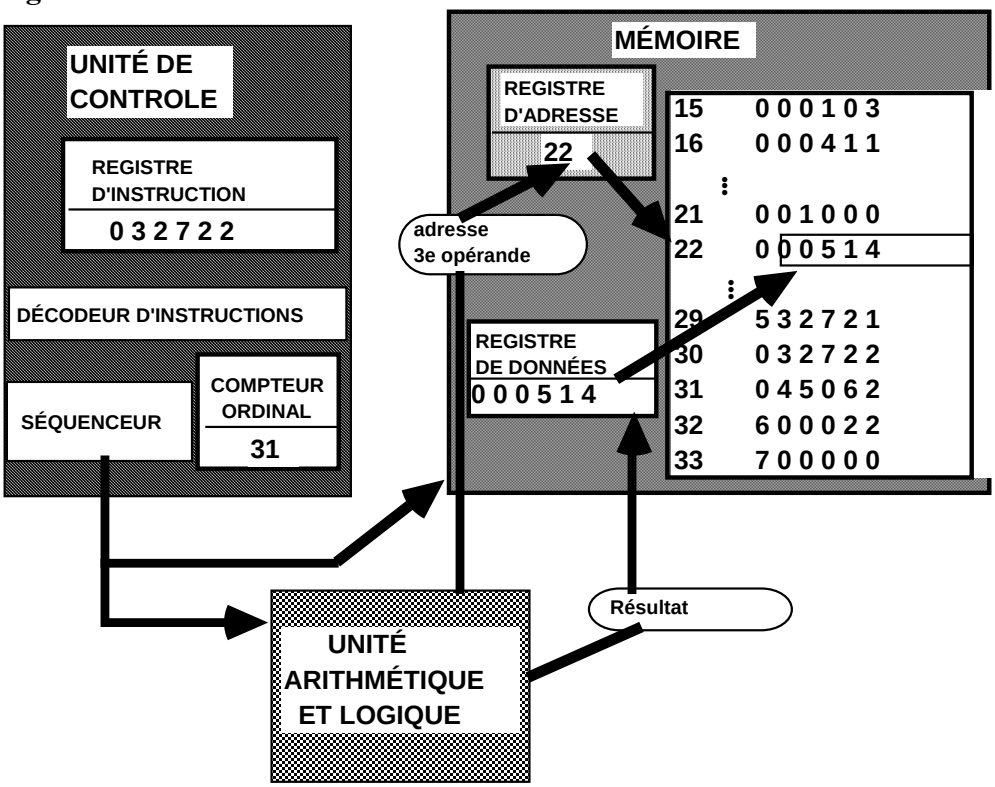

Le deuxième opérande est ensuite copié de la même façon dans un des registres de l'ALU:

**Figure 15**



Une fois l'opération effectuée, le résultat est copié en mémoire. Le compteur ordinal a été incrémenté et indique l'adresse de l'instruction suivante:

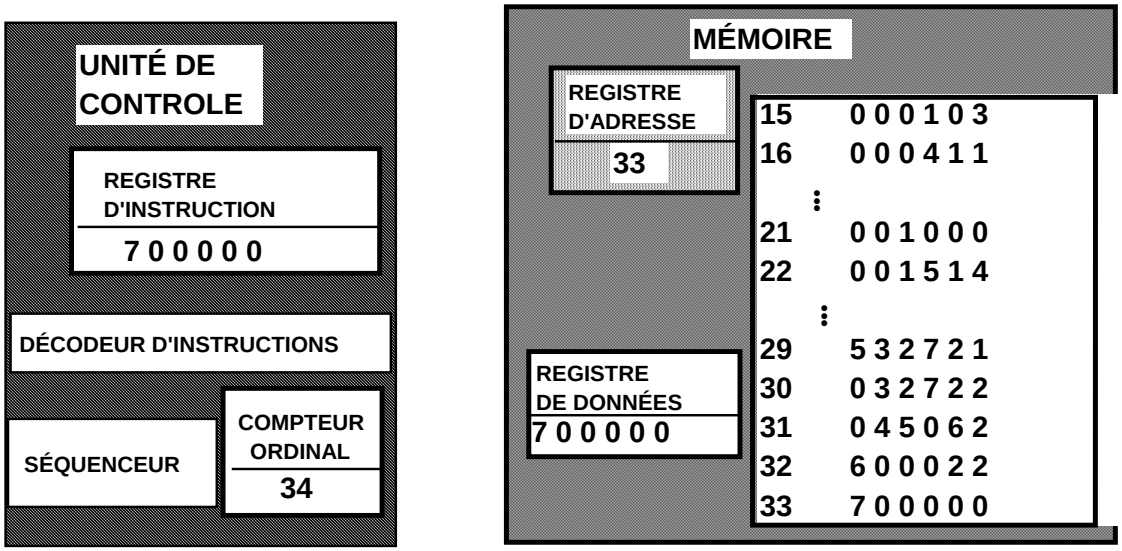
**Figure 22**




Notons qu'il aura fallu 4 cycles mémoire pour exécuter cette instruction: un premier pour obtenir l'instruction, deux autres pour obtenir les opérandes et un dernier pour ranger le résultat.

Les autres instructions sont exécutées de la même façon. A la fin du programme, la situation pourrait être schématisée ainsi:

**Figure 23**




Rupture de séquence et registre résultat test.

Dans certains cas, l'unité de contrôle doit conserver des résultats intermédiaires ou le résultats de tests qui rompent la séquence d'instructions. Un registre spécial de l'unité de contrôle est utilisé à cette fin: le registre résultat test. Voyons un exemple. Considérons le programme suivant:

| **No.** | **Étiquettes** | **Mnémoniques** | **Opérandes** |
|---|---|---|---|
| 1 | | EGU | A, B, TØ |
| 2 | | SOU | A, B, RESUL |
| 3 | | ARR | 0, 0, 0 |
| 4 | TØ | ADD | A, B. RESUL |
| 5 | | ARR | 0, 0, 0 |

Le programme compare les nombres dont les adresses sont symbolisées par A et B. Si les nombres sont égaux, il en fait la somme, sinon, le second est soustrait du premier. TØ désigne l'adresse symbolique de l'instruction à exécuter en cas d'égalité des contenus des mots d'adresses A et B.

Supposons que A et B désignent les adresses octales 25 et 26, alors que TØ désigne l'adresse octale 23 (respectivement 21, 22 et 23 en décimal), et que les adresses octales 20 à 27 de la mémoire ont les contenus suivants:

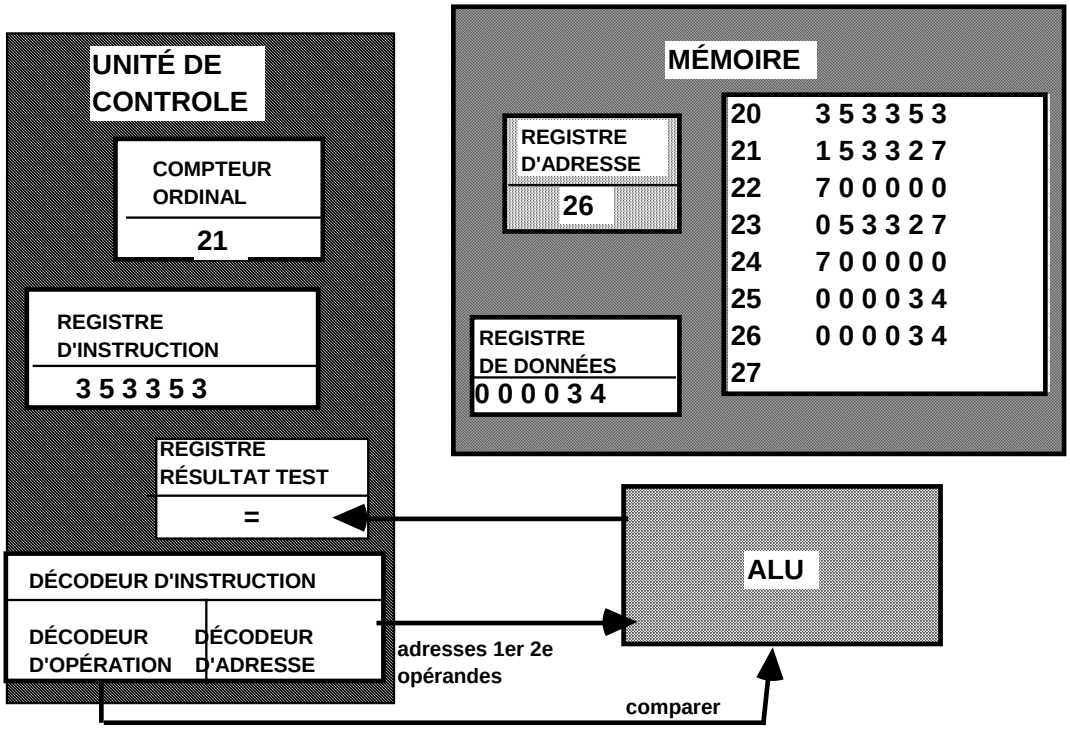
**Tableau 2**

| ADRESSES | Étiquette | INSTRUCTIONS | CONTENU | |
|---|---|---|---|---|
| | | | BINAIRE | OCTAL |
| 20 | | EGU A, B, TØ | 011 10101 10110 10011 | 353353 |
| 21 | | SOU A, B, RESUL | 001 10101 10110 10111 | 153327 |
| 22 | | ARR | 111 00000 00000 00000 | 700000 |
| 23 | TØ | ADD A, B, RESUL | 000 10101 10110 10111 | 53327 |
| 24 | | ARR | 111 00000 00000 00000 | 700000 |
| 25 | | 34 | 000 00000 00000 11100 | 34 |
| 26 | | 34 | 000 00000 00000 11100 | 34 |
| 27 | | | | |

Dans ce cas-ci, comme les deux valeurs sont égales, c'est l'instruction d'addition qui devait être exécutée. Le déroulement se fera comme dans l'exemple précédent pour l'extraction de l'instruction, son décodage, la recherche des opérandes.

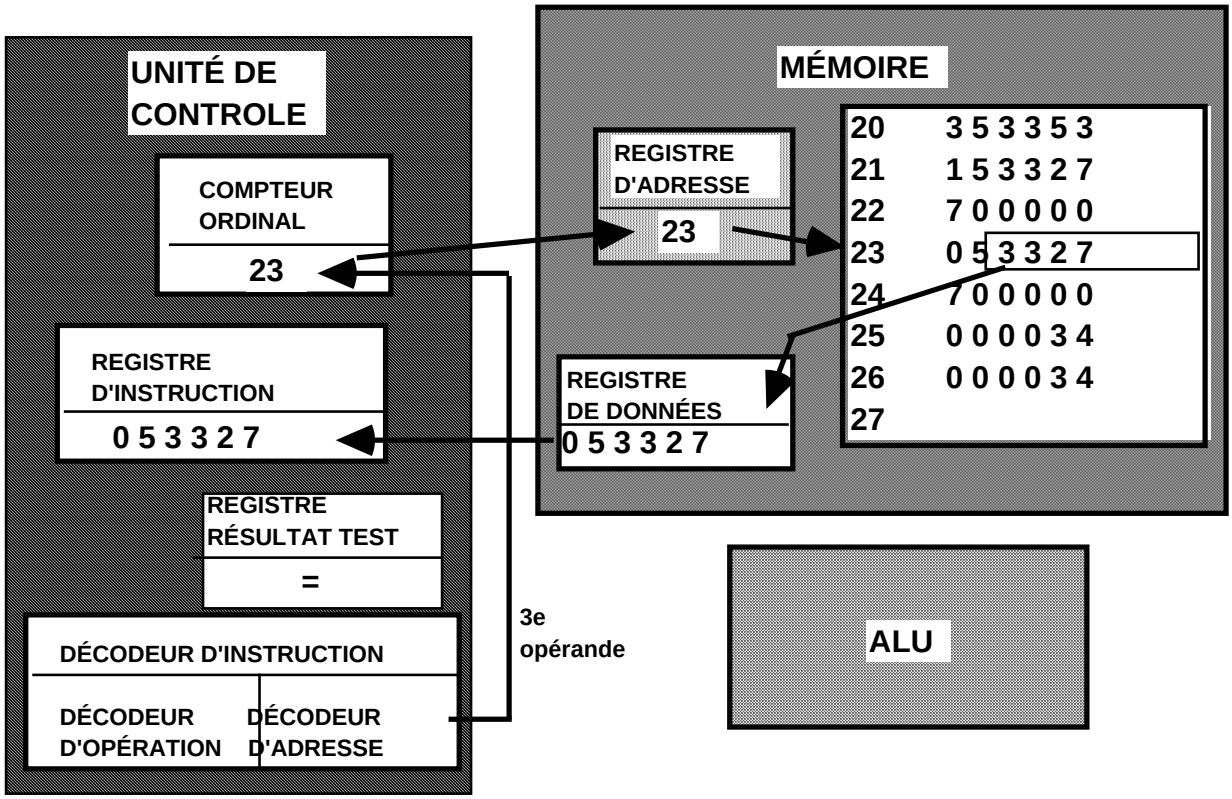
Sauf qu'une fois la comparaison effectuée par l'ALU, le résultat de cette comparaison sera copié dans le registre résultat test de l'unité de contrôle:

**Figure 24**




Ici, comme le résultat est égal, on doit passer à l'instruction 4. Le compteur ordinal prendra donc la valeur de l'adresse contenue dans le troisième opérande de l'instruction:

**Figure 16**



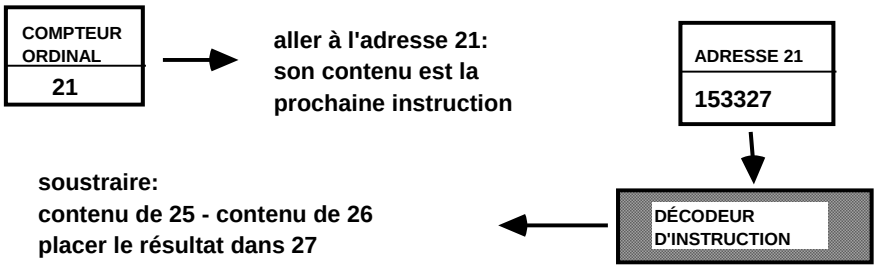

L'instruction d'addition sera exécutée, puis le compteur ordinal sera incrémenté à 24 et l'instruction d'arrêt sera exécutée.

Si au contraire, les cellules de mémoire d'adresse 25 et 26 contenaient deux nombres différents, c'est l'instruction 2 qui serait effectuée après la comparaison de l'instruction 1. Il n'y aurait alors pas de rupture de séquence. On note que les instructions du programme qui doivent être exécutées séquentiellement sont rangées dans des cellules de mémoires d'adresses consécutives; par contre, l'instructions à exécuter en cas de rupture de séquence peut être rangée ailleurs. Ainsi, dans notre exemple, l'instruction 4 aurait pu ne pas être rangée dans la cellule suivant celle de l'instruction 3, puisque ces deux instructions ne sont pas exécutées séquentiellement.

### 4.5.5. Interprétation du contenu d'une cellule de mémoire.

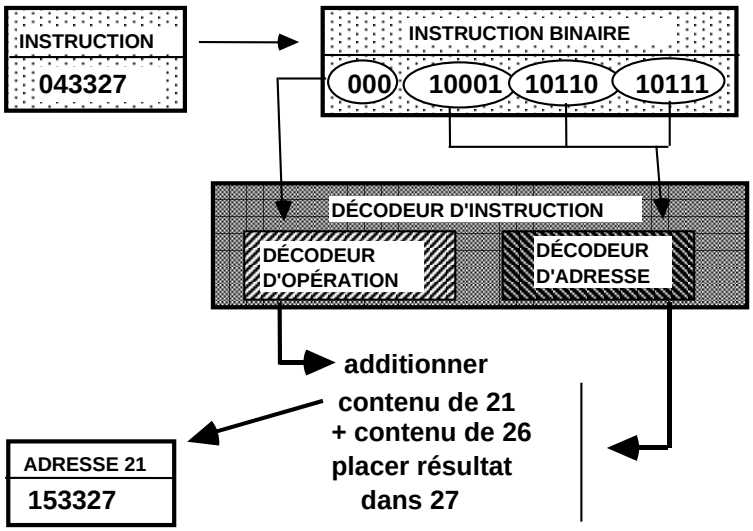
Encore une fois, on peut noter que le contenu d'une cellule de mémoire ne peut être interprété correctement qu'au cours d'une action. Par exemple, la cellule d'adresse 21 contient la chaîne octale 153327. Lorsque le compteur ordinal vaut 21 au moment d'exécuter une nouvelle instruction, cette chaîne sera interprétée comme l'instruction de soustraire le nombre contenu dans la cellule d'adresse 26 du nombre contenu dans la cellule d'adresse 25:

**Figure 26**




Si par contre, on faisait référence à la même chaîne par l'instruction 043327, elle serait interprétée comme étant le nombre octal 153327. En effet, dans cette nouvelle instruction, l'adresse 21 serait un opérande de l'instruction d'addition:

**Figure 27**

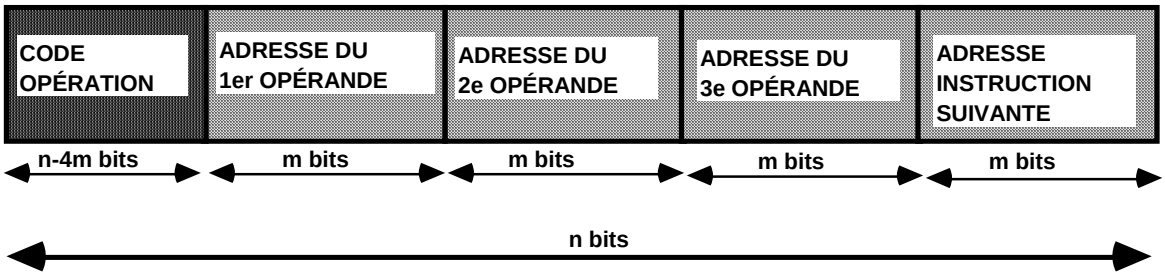


### 4.5.6. Nombre d'opérandes.

Voyons maintenant les différences de fonctionnement qu'implique un changement du nombre d'opérandes. Le format des instructions d'une machine peut en effet comporter 4, 3, 2, 1 ou même 0 opérandes. On parle d'instruction à 4, 3, 2, 1 ou 0 adresse.

#### 4.5.6.1. Instruction à 4 adresses.

**Figure 28**




.

**Exemple** : ADD X, Y, Z, W

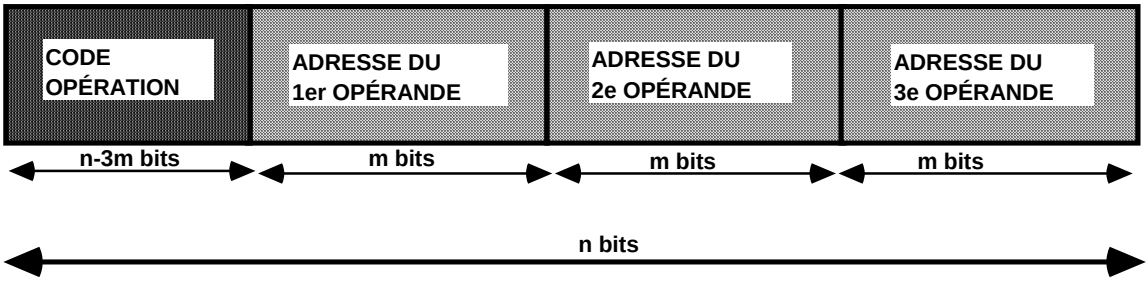
L'instruction signifie: (Z) ← (X) + (Y) & (W) c'est à dire:

- ✓ additionner les contenus des cellules de mémoire d'adresse X et Y,
- ✓ placer le résultat de l'addition dans la cellule d'adresse Z,
- ✓ aller chercher la prochaine instruction à la cellule d'adresse W.

Les instructions à 4 adresses sont très rares, car elles sont très longues. Les constructeurs préfèrent prévoir un compteur ordinal dans l'unité de contrôle qu'un opérande servant à indiquer l'adresse de la prochaine instruction.

#### 4.5.6.2.Instruction à 3 adresses.

**Figure 29**



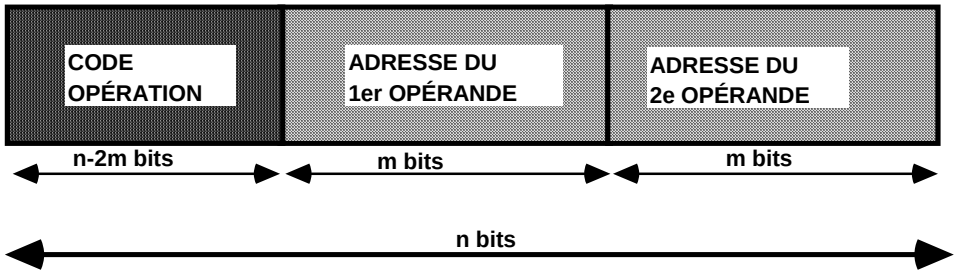

.

Ces instructions ont le même format que celles utilisées dans les exemples de programmes vus précédemment. Un processeur fonctionnant avec de telles instructions localise l'instruction

suivante à l'aide d'un registre compteur ordinal. Le compteur ordinal doit comporter m bits de position, car il est destiné à recevoir des adresses.

### 4.5.6.3. Instruction à 2 adresses

**Figure 17**




.

Les instructions à 2 adresses sont les plus courantes. L'adresse du résultat est implicitement choisie: celui-ci est rangé à l'adresse du 2ième opérande ou à celle du premier opérande selon le choix fait par le constructeur.

Exemple: L'expression **(X) = (A + B * C) / D** peut être programmée de la façon suivante:

| **INSTRUCTION** | **PSEUDO-CODE** |
|---|---|
| MUL B, C | (C) ← (B) * (C) |
| ADD A, C | (C) ← (A) * (C) |
| DIV C, D | (D) ← (C) / (D) |
| MOV D, X | (X) ← (D) |

Avantages sur les instructions à 3 ou 4 adresses:

1. puissance accrue de la machine, car pour une même longueur d'instruction on a:
   a) des opérandes plus longs (donc possibilité de faire référence à plus de cellules de mémoire) et/ou
   b) un code d'opération plus long (donc plus d'instructions différentes dans le répertoire)

2. baisse du coût relié au décodage
3. moins d'accès à la mémoire requis

Pour illustrer ces avantages, reprenons la machine imaginée à la section 2.2. Cette machine avait des instructions sur 18 bits. Chaque instruction avait 3 opérandes de 5 bits et un code d'opération sur 3 bits, ce qui lui donnait les caractéristiques suivantes:

nombre de cellules de mémoire adressables: $2^5 = 32$ cellules
nombre d'instructions du répertoire: $2^3 = 8$ instructions

Imaginons la même machine, mais avec 2 opérandes au lieu de 3. Les 18 bits de l'instruction pourraient, par exemple, se répartir ainsi: 2 opérandes de 7 bits et un code d'opération de 4 bits, ce qui donnerait les caractéristiques suivantes:

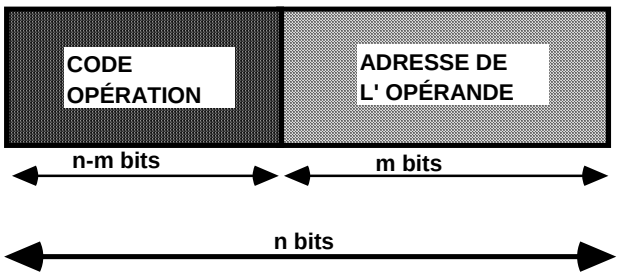
nombre de cellules de mémoire adressables: $2^7 = 128$ cellules
nombre d'instructions du répertoire: $2^4 = 16$ instructions

Comme on le constate, le répertoire d'instruction a doublé et le nombre de cellules de mémoire adressables a quadruplé par rapport à l'instruction à trois adresses.

En fait, dans les instructions à 3 adresses, le 3ième opérande ne sert qu'à indiquer l'adresse de rangement du résultat, puisque les opérations logiques et arithmétiques s'effectuent sur deux opérandes à la fois. La suppression du 3$^{ème}$ opérande ne nuit donc pas au travail de l'ALU. Évidemment, il peut être ennuyeux de perdre la valeur du 2ième opérande en lui substituant le résultat, mais on peut remédier à cela en faisant une copie du deuxième opérande dans une autre cellule de mémoire si cette valeur doit être conservée.

### 4.5.6.4. Instruction à une adresse

**Figure 18**




.

Une instruction à une adresse nécessite non seulement un compteur ordinal, mais aussi un registre supplémentaire pour stocker temporairement les valeurs intermédiaires pendant le calcul. Ce registre, appelé accumulateur (ACC.) peut contenir la donnée source ou le résultat du calcul.

**Exemple:** Le calcul de l'expression **(X) = (A + B * C) / D** peut être programmé de la façon suivante:

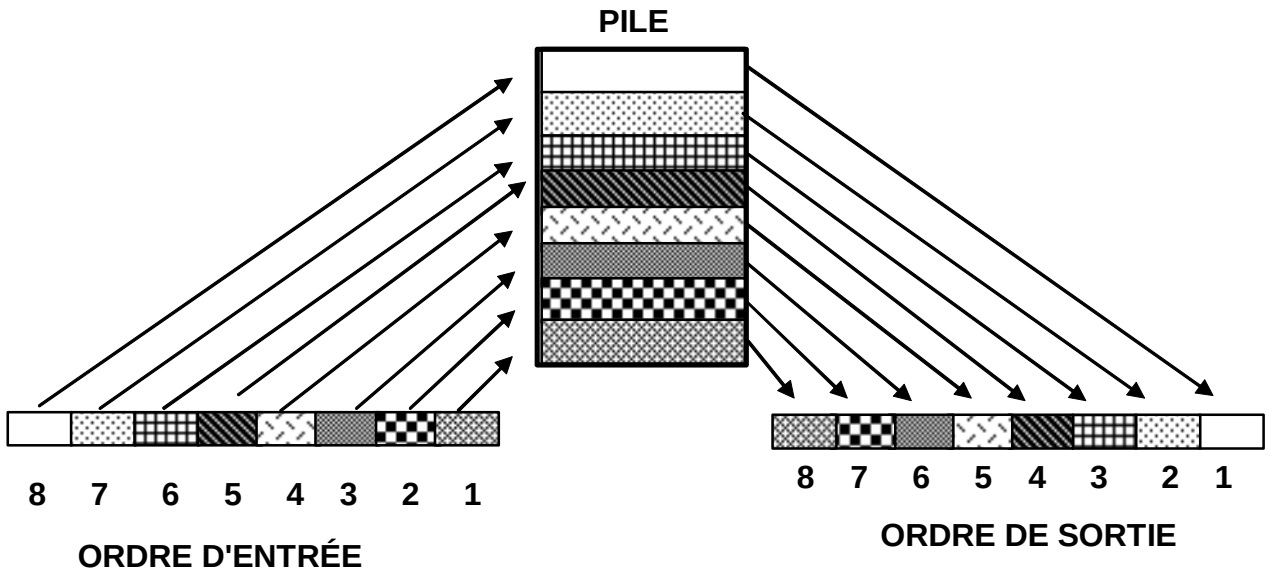
| **INSTRUCTIONS** | **SIGNIFICATION** |
|---|---|
| LOAD B | ACC. ← (B) |
| MUL C | ACC ← Contenu de l'ACC. + (C) |
| ADD A | ACC ← Contenu de l'ACC. + (A) |
| DIV D | ACC ← Contenu de l'ACC. / (D) |
| STORE X | (X) ← Contenu de l'ACC. |

**Remarque:** Les instructions de rupture inconditionnelle sont un exemple d'instruction à 1 opérande.

### 4.5.6.5. Instruction à 0 adresse

Le processeur à 0 adresse possède un accumulateur particulier qu'on nomme pile. Une pile est un rangement par ordre chronologique. Contrairement à un accumulateur ordinaire, placer une donnée dans une pile ne détruit pas son contenu précédent: celui-ci sera plutôt repoussé au fond de la pile. Généralement, l'accès aux données se fait dans l'ordre inverse où celles-ci ont été rangées dans la pile; on désigne souvent ce type d'accès par le sigle anglais LIFO (Last In, First Out). Cela implique qu'on a accès qu'à la donnée qui est au sommet de la pile (soit la dernière entrée) ou aux données se trouvant dans des emplacements consécutifs à partir du sommet.

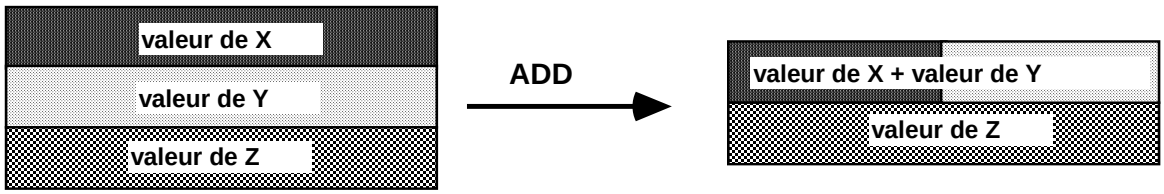
**Figure 31**




L'instruction à 0 adresse permet de modifier la pile en effectuant une opération sur les deux données à partir du sommet, et en plaçant le résultat de l'opération au sommet de la pile.

**Exemple:** ADD signifie: additionner les valeurs du sommet de la pile et placer le résultat dans la pile.

**Figure 32**



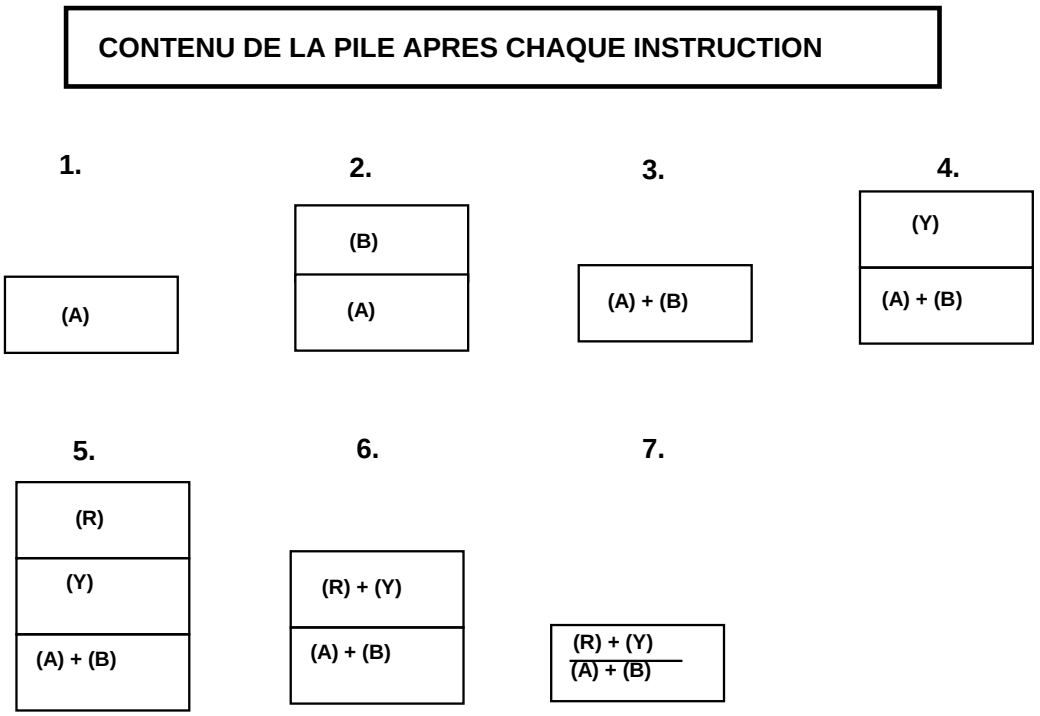

Pour ranger des données dans la pile ou pour les en retirer, on utilise deux instructions :
**PUSH A:** place le contenu de A dans la pile (et pousse les contenus précédents vers le fond de la pile)

**POP A:** retire la valeur située au sommet de la pile et la range à l'adresse A (et fait "remonter" les contenus précédents d'une position vers le sommet)

**Exemple:** Le calcul de l'expression (R) = ((R) + (Y))/((A) + (B)) peut être programmé de la façon suivante:

| NO. | INSTRUCTIONS | SIGNIFICATION |
|---|---|---|
| 1 | A | Entrer la valeur de A |
| 2 | PUSH | Placer la valeur de A dans la pile |
| 3 | B | Entrer la valeur de B |
| 4 | PUSH | Placer la valeur de B dans la pile |
| 5 | ADD | Sommet de la pile ← (A) + (B) |
| 6 | Y | Entrer la valeur de Y |
| 7 | PUSH | Placer la valeur de Y dans la pile |
| 8 | R | Entrer la valeur de R |
| 9 | PUSH | Placer la valeur de R dans la pile |
| 10 | ADD | Sommet de la pile ← (Y) + (R) |
| 11 | DIV | Sommet de la pile ← Résultat de la 2ième addition divisé par Résultat de la 1ière addition |
| 12 | POP | Sortir la valeur de Sommet de la pile |
| 13 | R | R ← Sommet de la pile |

La figure suivante montre quel est le contenu de la pile après chacune des opérations:




## 4.6. L'adressage

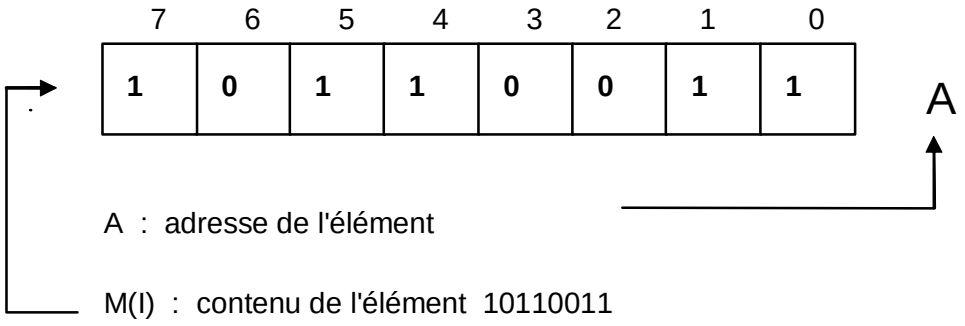
### 4.6.1. Généralités.

Toute information traitée par l'ordinateur, quelque soit le format utilisé, est désignée et référée par une adresse. C'est par l'adresse de l'information à traiter que l'ordinateur accède à celle-ci.

Comme on l'a vu, la mémoire centrale est découpée en emplacements ou éléments. Ces éléments de mémoire sont des regroupements de bits élémentaires de même longueur, soit des octets, des mots, des double-mots et autres. Chaque emplacement est numéroté et désigné par un numéro unique qu'on appelle l'adresse de l'emplacement.

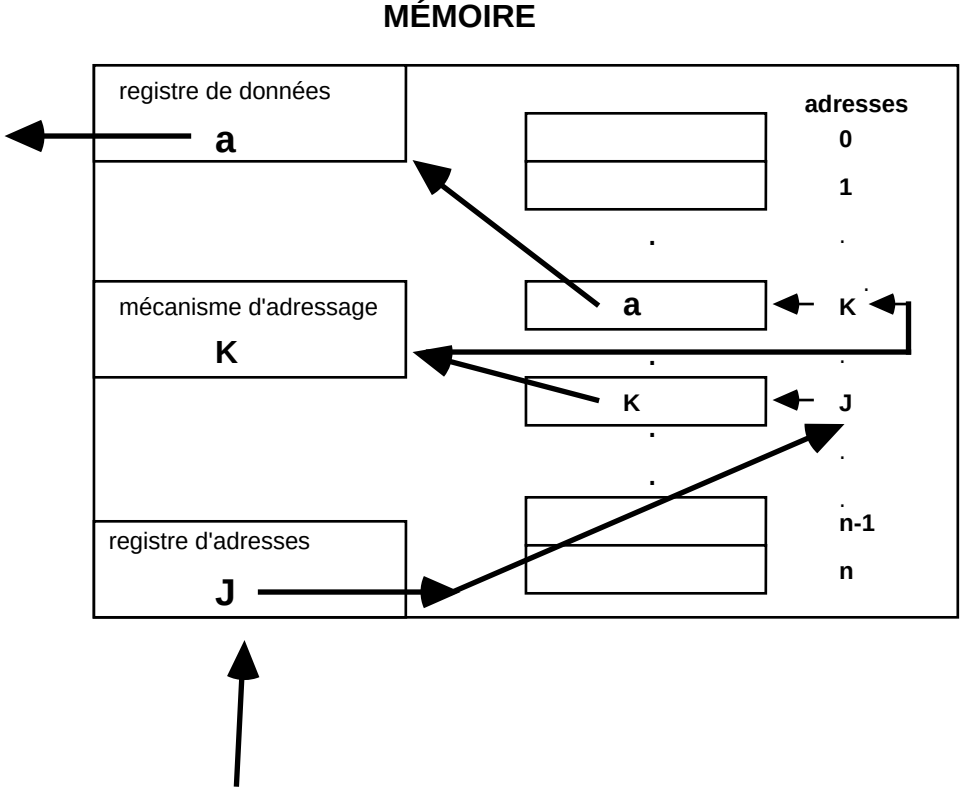
Par commodité, notons par I l'adresse d'un élément de la mémoire et par M(I) son contenu.

Exemple d'un emplacement octet.



Même le nom de l'élément peut être conservé en mémoire dans un autre élément devenant la valeur de ce dernier. Par exemple, la valeur a peut être conservée dans l'élément d'adresse K , on a alors selon notre notation M(K) = a ; et l'adresse K , qui est un entier, peut être elle-même conservée dans un autre élément d'adresse J , c'est-à-dire que M(J) = K. La composition des deux noms entraîne alors l'identification M(M(J)) = a.

C'est là d'ailleurs la notion fondamentale de pointeur. En désignant l'adresse J , un mécanisme approprié d'accès trouvera dans l'élément d'adresse J une nouvelle adresse K , pour livrer finalement la valeur désirée a , contenu de l'élément d'adresse K .

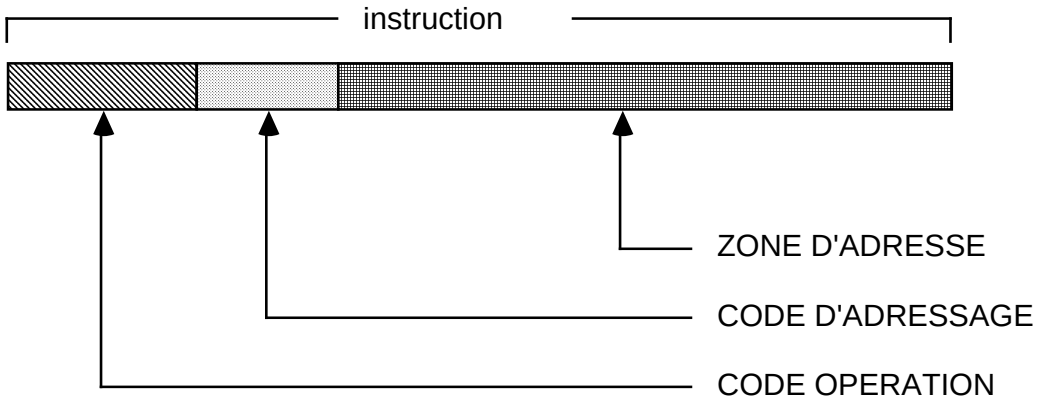



Notons qu'on pourrait poursuivre ce mécanisme d'adressage indirect à des niveaux plus élevés. Le problème de l'adressage est donc celui du moyen qui permet d'indiquer dans les instructions quelles sont les adresses des opérandes concernés.

Le type de traitement que doit subir le contenu de la zone d'adresse dans les instructions est généralement spécifié soit par le code opération lui-même de l'instruction, soit par une configuration binaire occupant une partie de l'instruction qui précise la formule d'adressage à utiliser. En effet, dans le répertoire d'instructions de l'ordinateur, on trouve souvent certaines opérations qui s'adressent directement au contenu de la zone d'adresse d'une instruction, opération déclenchée par l'unité de contrôle au décodage de l'instruction.

Une instruction renferme donc généralement trois zones en général: une zone pour le code d'opération, une autre pour le code d'adressage et une dernière pour les adresses des opérandes.

### 4.6.2. Format d'une instruction à plusieurs modes d'adressage




Le code d'adressage est formé d'indicateurs binaires spécifiant les modes d'adressage ou recette à suivre pour calculer à partir de l'information contenue dans la zone d'adresse de l'instruction, l'adresse effective de l'élément appelée l'adresse physique de l'élément qui est impliqué dans l'instruction.

Plusieurs modes d'adressage des informations sont possibles dans un même ordinateur. Nous allons présentés dans ce qui suit les modes ou techniques d'adressage les plus fréquemment utilisées.

En résumé, on peut définir l'adressage comme une fonction logicielle ou câblée (s'il s'agit d'une implantation physique à l'aide de circuits électroniques) permettant de sélectionner un élément d'information parmi un ensemble d'éléments de même type par un calcul d'adresse. . Le résultat de tout mode d'adressage est une adresse physique ou effective et opérationnelle d'un élément. Dépendant du contexte, l'élément en question pourra être un mot mémoire, un registre, une unité périphérique, une page de la mémoire, un numéro de piste sur un disque magnétique, un numéro d'enregistrement physique sur une bande magnétique, etc. L'adresse physique de l'élément est obtenue après l'exécution de la fonction d'adressage ou algorithme d'adressage.

Pourquoi a-t-on implanté plusieurs modes d'adressage qui semblent à première vue compliquer les choses plutôt que de les simplifier?

Cette question comporte plusieurs volets mais une des raisons fondamentales réside au niveau des instructions. En fait, il n'est pas toujours avantageux de mettre directement l'adresse réelle ou physique des opérandes dans les instructions. Les ordinateurs de moyenne et grande puissance d'aujourd'hui possèdent des mémoires volumineuses de plusieurs millions, parfois même de quelques centaines de millions d'emplacements adressables (en mémoire physique ou virtuelle). Pour les disques magnétiques, ce nombre atteint facilement quelques milliards. Par exemple, une mémoire de 16 Megs (méga-octets), donne $16* 2^{10}$ K ou $16*2^{10}*2^{10}$ octets ou $2^{24}$ octets . Ceci veut dire que les adresses varient de 0 à $2^{24-1}$. Par conséquent, une adresse occupe 24 bits à elle seule dans le registre d'instruction. Pour une instruction à deux opérandes, la zone d'adresse exige à elle seule 48 bit de position du registre d'instruction. Les techniques d'adressage permettent entre autres de réduire au besoin l'espace occupé par les adresses particulièrement

dans les instructions machines. On se sert même des techniques d'adressage pour raccourcir la longueur du registre d'instruction dans certains petits ordinateurs.
La réduction de l'espace d'adresse des instructions n'est pas l'unique raison d'être des techniques d'adressage, une autre tout aussi fondamentale est reliée aux problèmes de gestion de la mémoire, de gestion des processus, de protection et autres; ces problèmes interviennent au niveau du système d'exploitation de l'ordinateur. On abordera un peu plus loin tous ces problèmes de gestion.

## 4.7. Les modes d'adressage.

Dans le répertoire d'instructions de l'ordinateur, on peut diviser les instructions en plusieurs sous-groupes dépendant à quelle unité l'instruction s'adresse, cette classification peut être faite aussi selon la fonction de l'instruction. Par exemple, les instructions de l'unité arithmétique forment un sous-groupe, celles des unités d'entrées/sorties en forment un autre, etc. Une autre classification possible peut se réaliser aussi selon le nombre d'opérandes impliqués dans l'instruction. On distingue généralement des instructions à zéro, un, deux et trois opérandes chez la plupart des microprocesseurs. Le VAX en a qui ont jusqu'à six opérandes.

Dépendant de leurs fonctions, les adresses des instructions peuvent apparaître sous plusieurs modes d'adressage. Par exemple, l'adressage sera direct dans la plupart des instructions de l'ALU mais relatif ou indexé dans les instructions de lecture/écriture en mémoire centrale. Voyons plus en détails la signification de ces divers modes d'adressage. Évidemment, des indicateurs de mode et de format sont nécessaires lorsque plusieurs modes sont présents dans la même machine ayant des formats multiple d'instructions.

### 4.7.1. L'adressage immédiat

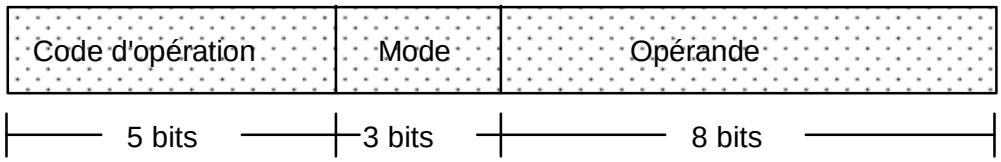
La façon la plus simple de spécifier un opérande dans une instruction, c'est d'écrire directement sa valeur dans la zone d'adresse de l'instruction. On appelle ce type d'opérande opérande immédiat. Celui-ci est chargé automatiquement dans la mémoire registre de l'unité de traitement lors du cycle de décodage de l'instruction, après bien sûr l'identification de ce mode d'adressage.

Le terme opérande immédiat est par conséquent plus adéquat que celui d'adressage immédiat car il n'y a pas de référence à la mémoire centrale de l'ordinateur, l'instruction étant elle-même porteuse de la valeur de l'opérande.

Cette technique d'adressage est particulièrement utile dans les instructions de décalage où la valeur inscrite est présente pour indiquer la valeur du décalage à effectuer. Ce mode intervient principalement dans les instructions de transfert d'information dans les registres du microprocesseur de l'unité de traitement. En fait, cette technique est utilisée là où l'usage d'une constante numérique entière de valeur fixe est requise. Cette valeur est cependant limitée au nombre de bits de la partie adresse, réduisant sensiblement le pouvoir de ce mode d'adressage.

Ce mode d'adressage en plus d'économiser de la place en mémoire principale, diminue le temps d'exécution car il n'y a pas de requête à la mémoire principale.
Exemple :

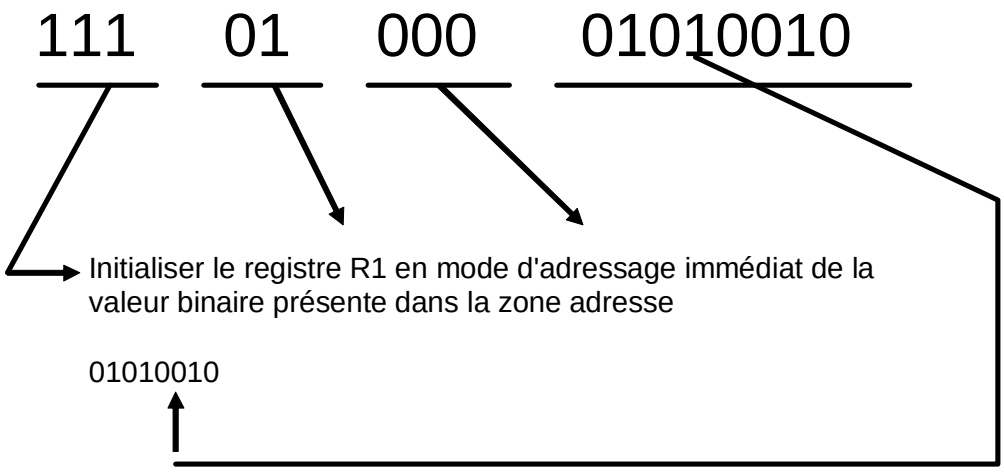
Supposons un ordinateur muni de quatre registres internes à son unité de commande identifiés R0 à R1 et numérotés en binaire 00 à 11 respectivement. Supposons de plus qu'un des formats d'instruction est à un opérande de la forme




où MODE indique le mode d'adressage utilisé dans l'instruction codé sur 3 bits. Supposons que la signature du mode d'adressage immédiat est le code 000. Si le répertoire d'instruction comprend l'opération d'initialisation suivante des registres internes: INIxx où xx indique le numéro du registre impliqué, l'opération étant codée sur cinq bits sous la forme binaire 111xx, alors l'instruction machine

1110100001010010

sera interprétée par l'unité de commande de la façon suivante:




**Question:** Quelle serait l'instruction machine permettant d'initialiser le registre R3 de la valeur décimale 1 en utilisant l'adressage immédiat?

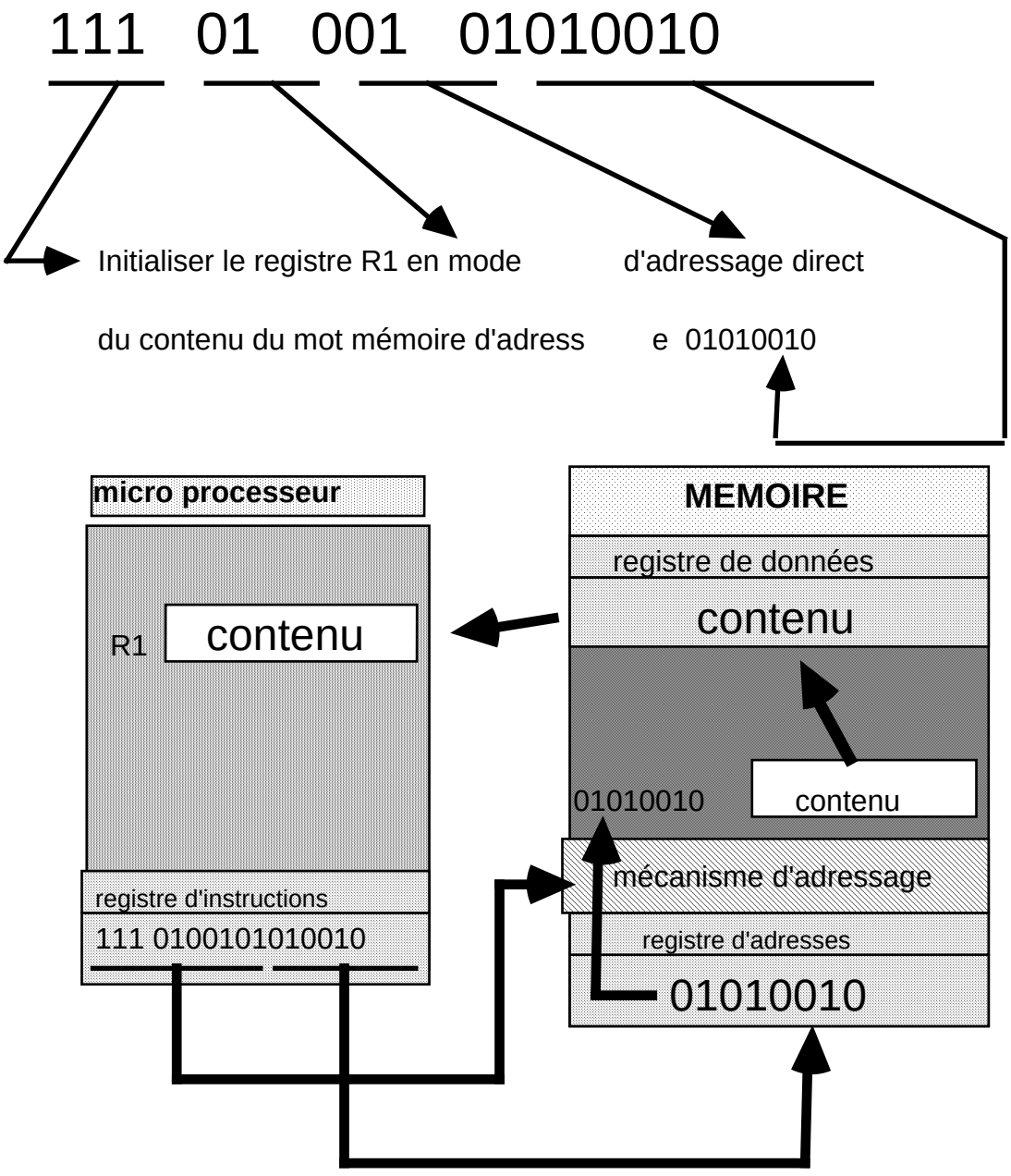
### 4.7.2. L'adressage direct

C'est le mode d'adressage le plus simple et le plus utilisé. Ce mode permet de référer à un mot mémoire en faisant figurer dans la zone adresse, directement l'adresse du mot désiré sans plus de complication, mot contenant la valeur de l'opérande impliqué. Cela nécessite exactement un cycle de mémoire pour que la valeur désirée soit livrée. La majorité des instructions de nature arithmétique utilisent ce mode d'adressage.

**Exemple:** Supposons le même ordinateur qu'à l'exemple 1 , ayant le même format d'instruction à un opérande. Supposons que le code pour l'adressage direct est la chaîne binaire 001. Alors l'instruction d'initialisation suivante

1110100101010010

aura pour signification:



### 4.7.3. L'adressage registre.

C'est conceptuellement la même chose que l'adressage direct, excepté que les adresses sont celles de registres plutôt que de mots mémoire. Étant donné qu'il n'y a qu'un nombre restreint de registres internes, généralement entre quatre et 16 registres, rarement plus, enfin tout au plus 32 à l'heure actuelle, le nombre de bits nécessaires pour spécifier le registre impliqué dans une adresse est très petit, 5 bits tout au plus s'il y a 32 registres. Le format de l'instruction est donc beaucoup plus court.

On peut supposer que la différenciation entre les formats pour l'espace adressable mémoire et celui de l'espace registre se réalise automatiquement au niveau du code d'opération de l'instruction. Voyons l'illustration de ce mode d'adressage registre dans l'exemple suivant.

**Exemple:** Supposons l'opération d'addition selon un mode d'adressage de l'espace mémoire dont la mnémonique est ADD1 avec le code 0000 et celui de l'espace registre de mnémonique ADD0 avec le code 0001 . Considérons les deux instructions suivantes:

ADD0, R0,R1 signifiant R0 ← R0 + R1

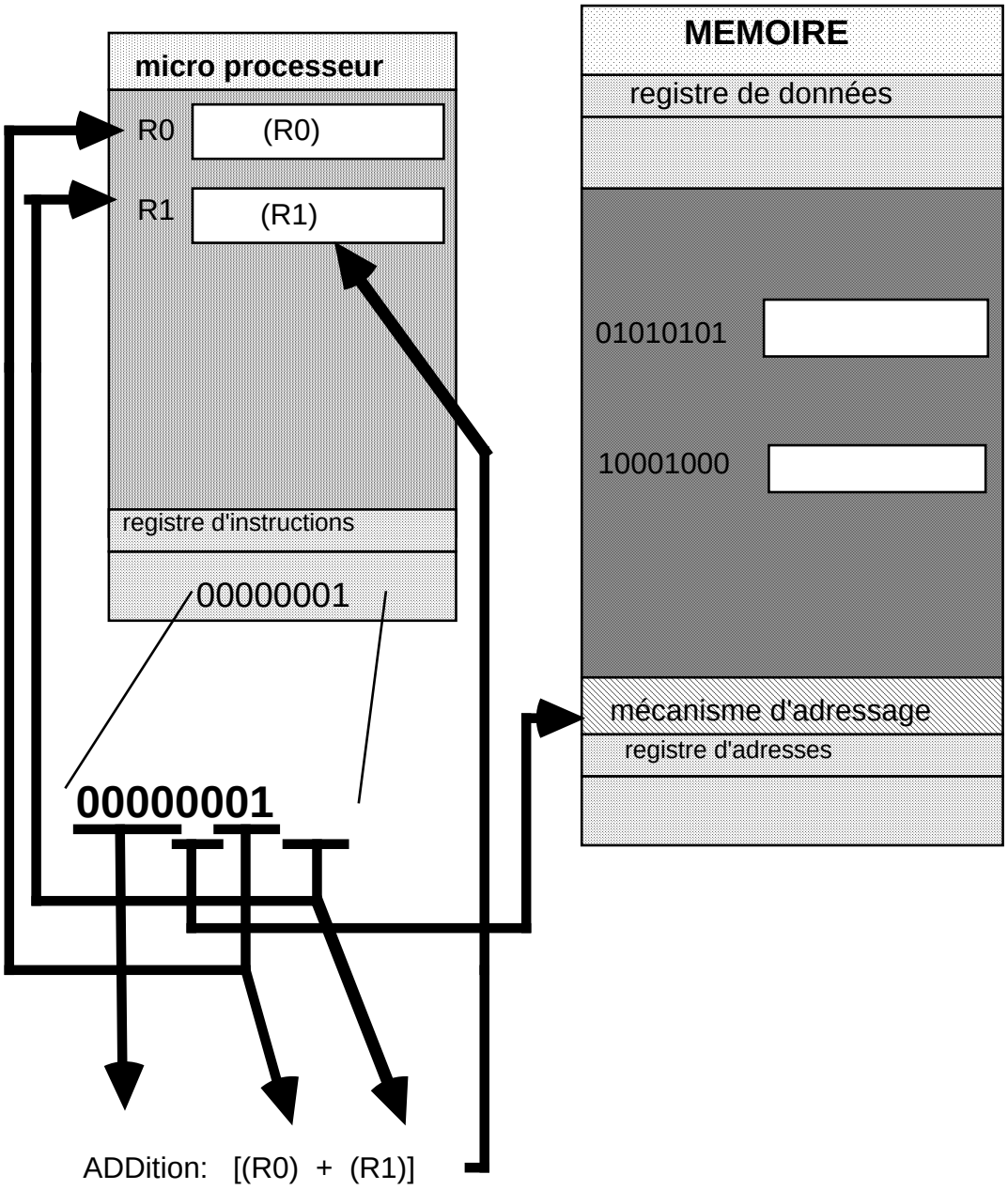
et

ADD1,A,B signifiant A ← A + B.

Si 00 est l'adresse du registre R0 et 01 celui du registre R1 comme dans l'exemple 1, si A représente l'adresse 00110111 et B l'adresse 11000111, les deux instructions mentionnées s'écrivent selon ces codes:

00000001 et 0001001101111111000111.

On remarque immédiatement une différence dans la longueur des instructions, le format de l'instruction est donc plus court pour le mode d'adressage registre. De plus, celle des registres ne nécessite pas de référence à la mémoire directement, donc le temps d'exécution est nettement plus court que celui en adressage direct qui nécessite deux accès à la mémoire principale. Les deux situations sont illustrées dans les schémas suivants: adressage registre


micro processeur
R0
(R0)
R1
(R1)
registre d'instructions
00000001
00000001
MEMOIRE
registre de données
01010101
10001000
mécanisme d'adressage
registre d'adresses
ADDition: [(R0) + (R1)]

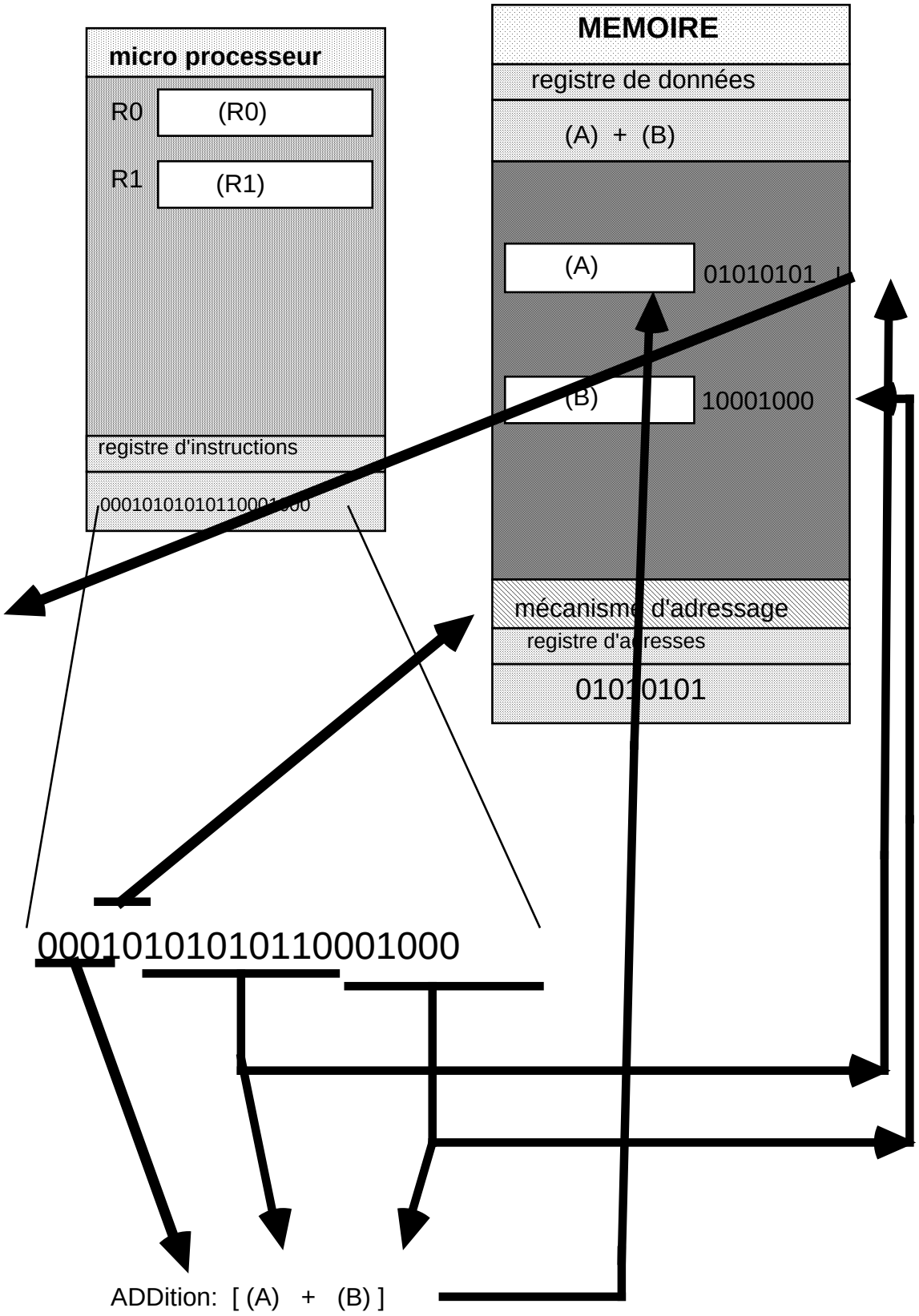

micro processeur
R0
(R0)
R1
(R1)
registre d'instructions
MEMOIRE
registre de données
(A) + (B)
(A)
01010101
(B)
10001000
mécanisme d'adressage
registre d'adresses
01010101
0001010101011000 1000
0001010101011000 1000
ADDition: [ (A) + (B) ]

**FIGURE 5-3-E**

**Exemple :** Voici deux séquences produisant le même résultat au niveau du registre R0, séquences écrites en utilisant les différents formats d'instructions précédents, agissant sur les mêmes mémoires et registres:

**Séquence 1**

```
ADD1, A,B       0001010101011000 1000
INI,R0,A        1110000101010101
```

**Séquence 2**

```
INI,R0,A        1110000101010101
INI,R1,B        1110000110001000
ADD0,R0,R1      00000001
```

On vérifie facilement par la trace mémoire l'équivalence entre les deux séquences au niveau du registre R0, cependant on note que l'effet n'est pas le même au niveau des autres mémoires impliquées.

**TRACE MEMOIRE DE LA SEQUENCE 1**

| R0 | R1 | A | B |
|---|---|---|---|
| [ R0 ] | [ R1 ] | [ A ] | [ B ] |
| [ R0 ] | [ R1 ] | [A]+[B] | [ B ] |
| [A]+[B] | [ R1 ] | [A]+[B] | [ B ] |

**TRACE MEMOIRE DE LA SEQUENCE 2**

| R0 | R1 | A | B |
|---|---|---|---|
| [ R0 ] | [ R1 ] | [ A ] | [ B ] |
| [ A ] | [ R1 ] | [ A ] | [ B ] |
| [ A ] | [ B ] | [ A ] | [ B ] |
| [A]+[B] | [ B ] | [ A ] | [ B ] |

### 4.7.4. L'adressage indirect

Dans ce mode d'adressage, une instruction donnée se réfère toujours au même mot de la mémoire. Pour rendre l'adresse variable, la seule solution est de modifier dans son entier l'instruction elle-même avant l'exécution. Ce procédé est lourd et parfois dangereux, surtout si le programmeur manque d'expérience. L'adressage indirect est une solution simple qui permet de rendre variable l'adresse à laquelle se réfère l'instruction.

Soit A, la partie adresse d'une instruction. Selon notre notation, M(A) et le contenu du mot d'adresse A. En mode d'adressage direct, l'adresse de l'opérande est A et la valeur de l'opérande

est M(A). Pour l'adressage indirect, l'adresse effective ou physique de l'opérande est M(A). L'adresse de l'opérande est donc le contenu du mot mémoire repéré par A. En fait, la valeur M(A) est un pointeur (une adresse).

En mode d'adressage indirect, on va donc chercher l'adresse de l'opérande en mémoire principale. Dans le cas où le mot mémoire est plus grand que la longueur en bits d'une adresse, une convention doit être prise par le constructeur pour déterminer quelle partie du mot servira à représenter l'adresse.

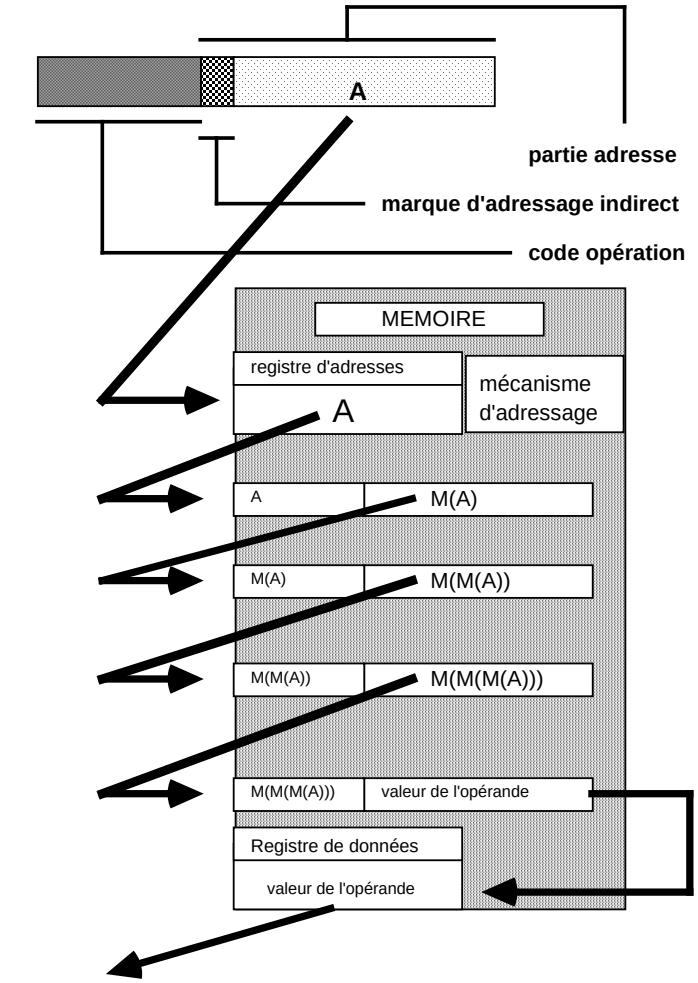
L'adressage indirect, lorsqu'il est utilisé, sera marqué par un code dans la partie mode de l'instruction ou par tout autre moyen finalement équivalent, un bit réservé à cette fin dans une position précise de l'instruction pourrait servir à l'indiquer par exemple. Ce bit pourrait aussi être logé dans la partie adresse même. Un tel choix permettrait alors des adressages indirects de profondeur quelconque.

En effet, l'adresse M(A) ainsi trouvée en mémoire à partir de A pourrait-elle aussi contenir la marque d'adressage indirect conduisant au contenu M(M(A)) comme adresse physique potentielle; et ainsi de suite de pointeur en pointeur jusqu'à la découverte d'un contenu qui ne contient pas cette marque. On obtient ainsi un chaînage d'adresses indirectes de profondeur quelconque. Certaines machines, bien que rares, admettent l'adressage indirect de profondeur quelconque.

À la profondeur égale à 2, le contenu M(M(A)) est l'adresse physique de l'opérande, c'est-à-dire que ce contenu n'est pas marqué du mode d'adressage indirect. Il faut noter qu'une erreur de programmation peut entraîner un bouclage. Par exemple, si A pointe sur B, si B pointe sur C et si, à son tour C pointe sur A, on se trouve en situation de bouclage infini qui devra être décelé et interrompu par la machine ou encore par une intervention venant de l'extérieur .

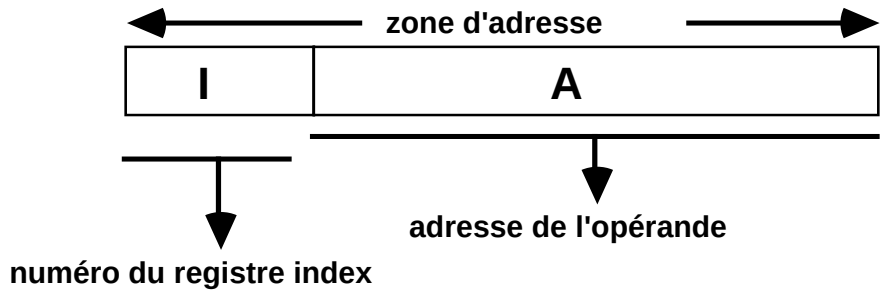
L'adressage indirect de profondeur 1 est le plus courrant et demande deux références à la mémoire principale. Rappelons-nous que l'adressage direct n'en demandait qu'une seule, tandis que l'adressage immédiat n'en demandait aucune, trouvant la valeur cherchée dans l'instruction même. En général, l'adressage indirect de profondeur n exigera n+1 requêtes à la mémoire principale pour trouver la valeur désirée de l'opérande. Notons que l'indirection de profondeur plus grande que l'unité ne se trouve que très rarement chez les ordinateurs conventionnels.

**Exemple:** (un adressage indirect de profondeur 3)

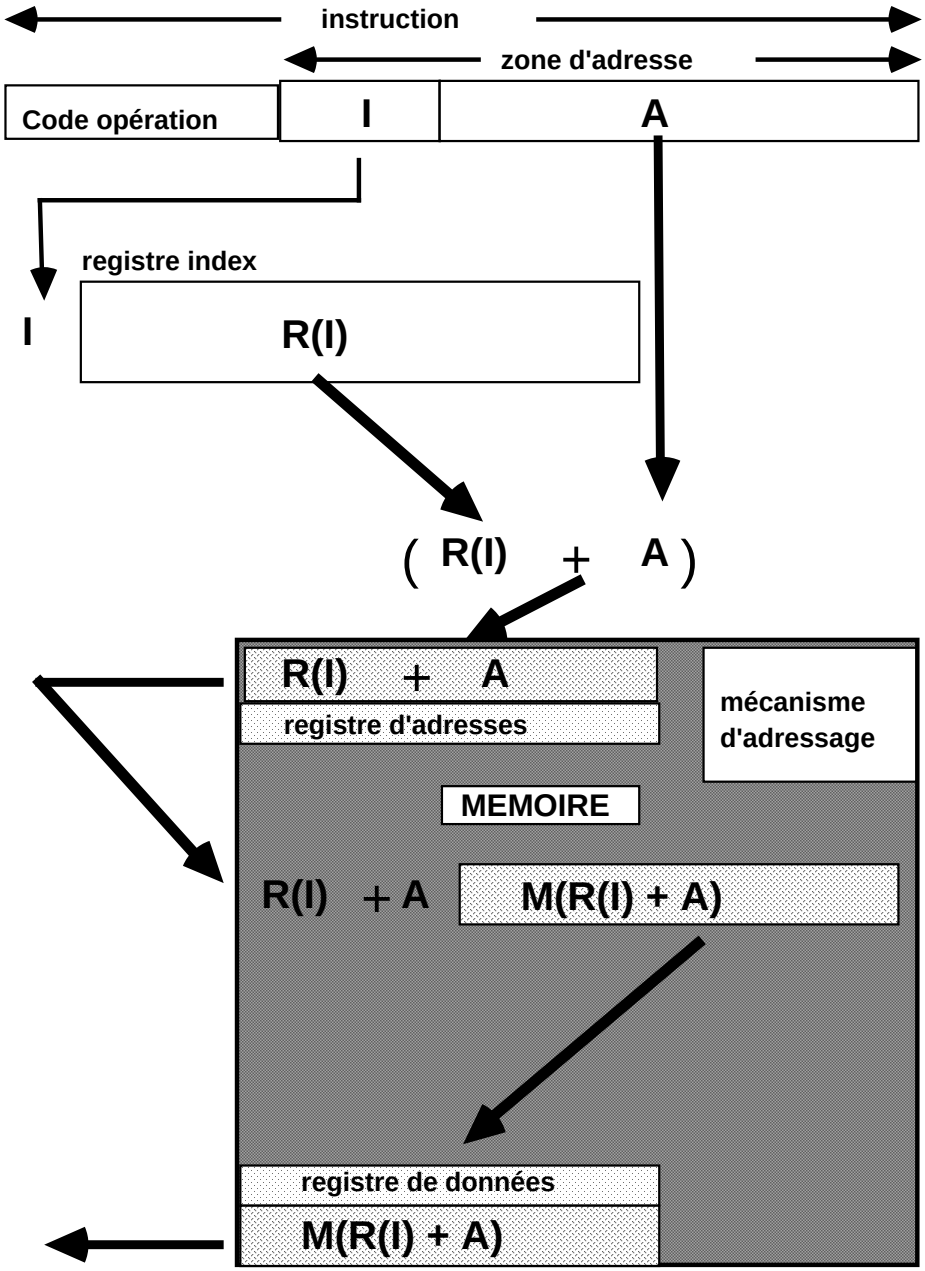



### 4.7.5. L'adressage indexé

Dans ce mode d'adressage, la zone d'un adresse est constituée de deux parties. Une première partie, appelée partie index, renferme le numéro I d'un registre spécial de la machine appelé registre index. La deuxième partie contient une adresse d'opérande.



À l'exécution, le contenu du registre index noté R(I) est ajouté à l'adresse A pour obtenir l'adresse effective de l'opérande soit A + R(I) en mémoire principale. Le contenu M(A+R(I)) donne la valeur de l'opérande impliqué. Notons que le registre index peut aussi être un mot mémoire. La fonction d'adressage est illustrée dans le schéma suivant:




Comme on peut le voir, il faut plusieurs opérations pour référer au mot mémoire désiré:

- accès au registre I contenant la valeur d'indexage
- addition de la partie adresse A
- accès à la mémoire à l'adresse résultante R(I) + A

Notons qu'il peut y avoir plusieurs registres internes à accès très rapide pouvant servir de registre index. Seul le numéro du registre servant à l'indexation apparaît dans l'instruction machine. Ce numéro peut aussi apparaître dans la zone indicatrice du mode d'adressage utilisé. L'indexation est l'un des modes les plus utilisés car il offre une très grande souplesse pour la gestion de la mémoire principale.

### 4.7.6. L'adressage indirect-indexation.

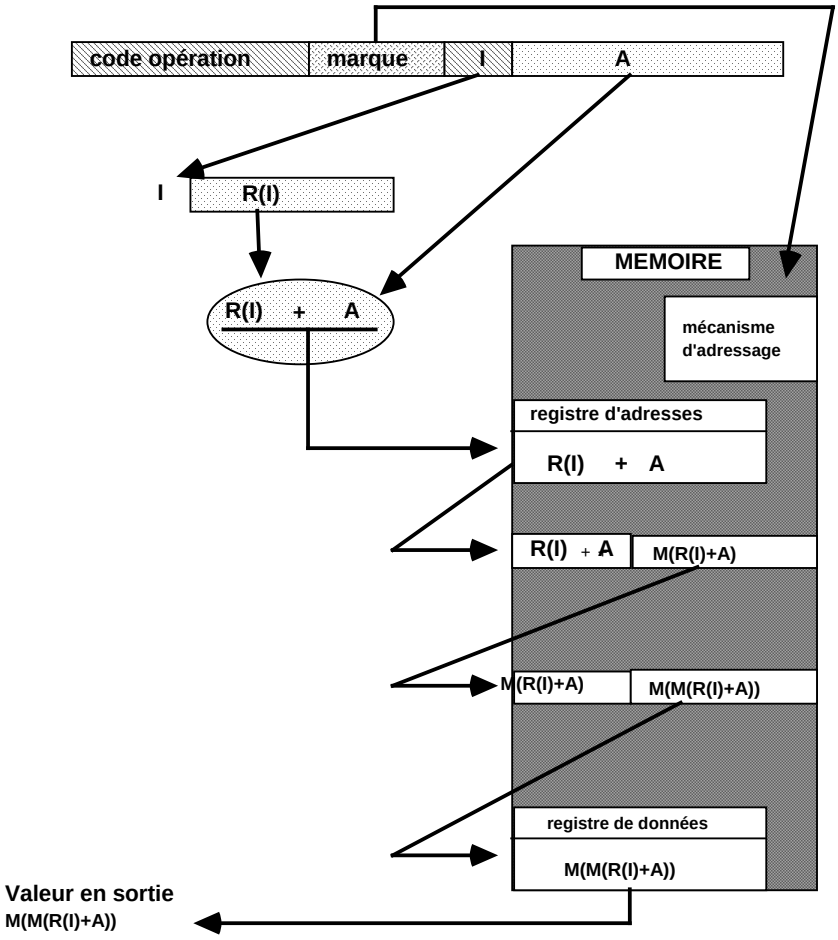
Comme son nom l'indique, ce mode d'adressage utilise à la fois deux modes d'adressage dans la même instruction: l'indirect et l'indexé. La zone d'adresse est donc dans ce cas subdivisée en trois éléments: la partie adresse A, la partie indexe I et la marque indiquant l'adressage indirect. Mais dépendant de l'ordre d'application de ces deux techniques d'adressage, nous aboutissons en mémoire principale à des opérandes effectifs différents. C'est pour cette raison qu'on distinguera ces modes par la terminologie suivante:

**Pré indexation - Post indirection**
indiquant que le mode d'adressage indexé est suivi du mode indirect

**Pré indirection - Post indexation** spécifiant que l'indirection est suivie de l'indexation.

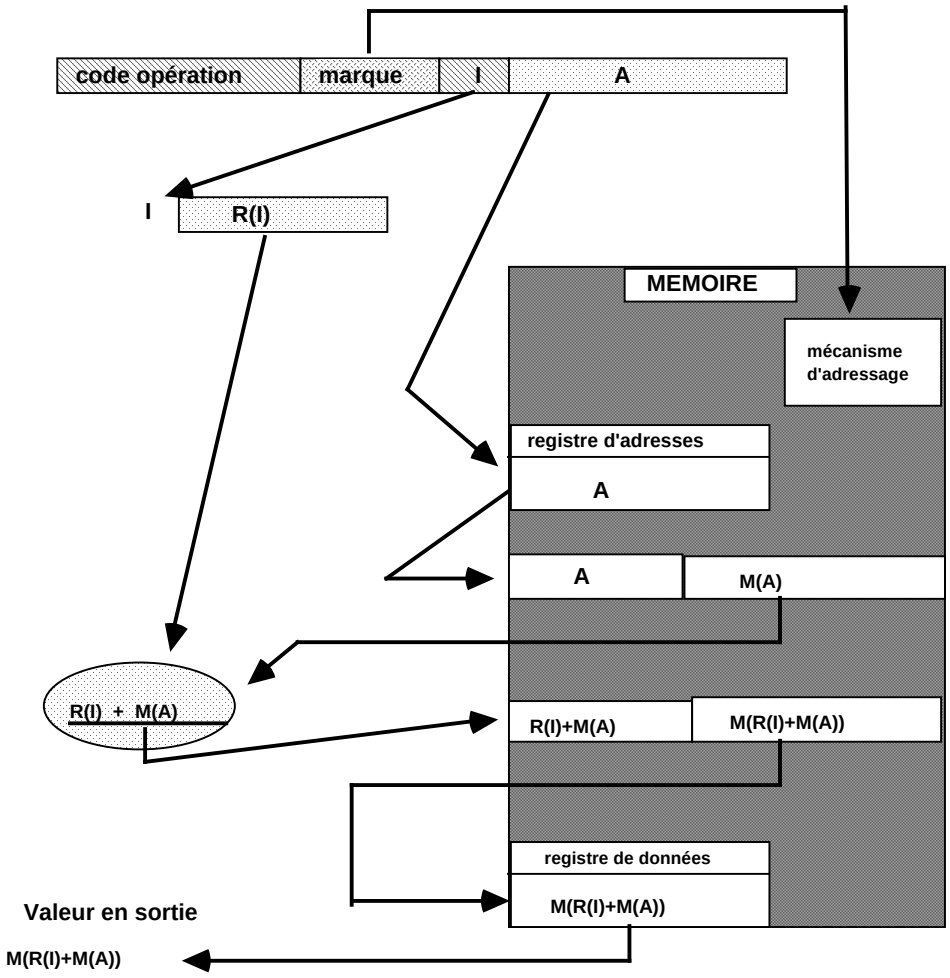
#### 4.7.6.1. L'adressage pré indexation - post indirection.

Dans ce mode, l'indexation est suivie de l'adressage indirect. Aussi, à partir du couple A et I contenu dans la partie adresse de l'instruction, l'adresse indexée est calculée donnant A + R(I). Cette adresse nous conduit par la suite au contenu M(A+R(I)), l'adresse physique de l'opérande désiré. La valeur de l'opérande M(M(A+R(I))) est alors livrée dans le registre de données de la mémoire principale. Toutes ces opérations sont évidemment synchronisées sur l'horloge interne de l'ordinateur et sous le contrôle de l'unité de commande et du séquenceur. La technique est illustrée dans le schéma suivant:




### 4.7.6.2. L'adressage pré indirection - post indexation.

Ayant le couple A et I, la marque indique d'effectuer d'abord l'indirection donnant comme résultat M(A). Ensuite vient l'indexation avec le registre I donnant l'adresse effective de l'opérande R(I) + M(A). Cette adresse fournit à la mémoire principale permet d'obtenir la valeur de l'opérande désirée M(R(I) + M(A)). Cette méthode est très utilisée par les compilateurs pour l'adressage d'éléments de tableaux de dimension connue uniquement à l'exécution entre autres.


code opération
marque
I
A
I
R(I)
MEMOIRE
mécanisme
d'adressage
registre d'adresses
A
A
M(A)
R(I) + M(A)
R(I)+M(A)
M(R(I)+M(A))
registre de données
M(R(I)+M(A))
Valeur en sortie
M(R(I)+M(A))

### 4.7.7. L'adressage relatif.

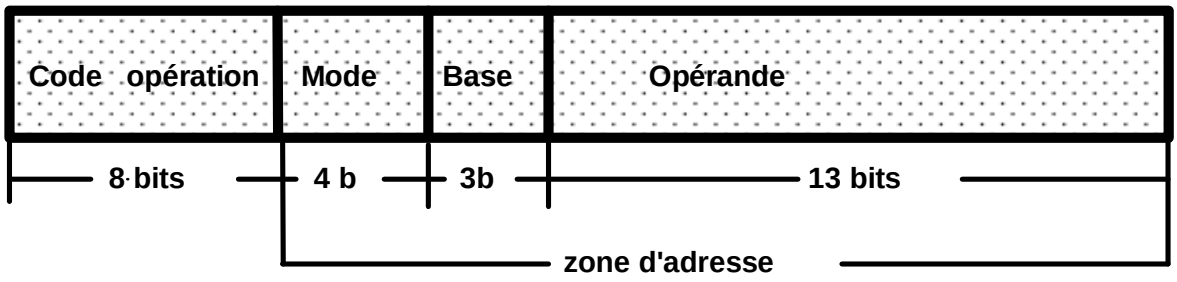
Ici encore, la zone d'adresse est divisée en deux partie appelées la base et le déplacement. Notons respectivement ces deux secteurs, B et D, D étant toujours positif. À l'exécution de la fonction d'adressage, l'adresse effective est donnée par R(B) + D, où B représente le numéro d'un registre spécial de contenu R(B), toujours selon notre notation. B est appelé registre de base.

L'intérêt de ce mode d'adressage est premièrement de pouvoir représenter une adresse d'une façon plus courte qu'une adresse en mode d'adressage direct. Donnons un exemple en supposant un ordinateur muni de 8 registres de base de 28 bits de long. Supposons aussi que le déplacement D est étendu sur 13 bits. La longueur des registres permet alors d'adresser une mémoire de $2^{28}$ mots, soit de 256 Mbits, qui demanderait 28 bits de position en adressage direct par adresse. Par contre, en adressage relatif, chaque adresse ne demandera que

| | | |
|---|---|---|
| | 13 bits | pour le déplacement |
| + | 3 bits | pour le numéro du registre de base |
| soit | 16 bits | par adresse, |

donc, une économie de 12 bits par adresse.

Pour fins pédagogiques, nous allons supposer que notre ordinateur ne dispose que d'instructions à une opérande et que le format unique est comme suit:




Dans cette configuration, son répertoire d'instructions comprend 256 opérations différentes et notre ordinateur permet 16 possibilités de mode d'adressage si on suppose qu'elles sont toutes identifiées dans la zone de marquage de mode.

Supposons que l'opération d'affectation d'un registre INI a pour code opération 83 en hexadécimal et que l'indicateur du mode d'adressage relatif est 7 (en hexadécimal). Supposons de plus que le registre de base porte le numéro 6 , que son contenu est A079BC6 et que le déplacement a pour valeur 08EB. Alors l'instruction s'écrirait selon ces initialisations 837C8EB ou

10000011011111001000111010 11 en binaire.

Le cheminement des données à l'exécution de la fonction d'adressage est montré dans la figure suivante:

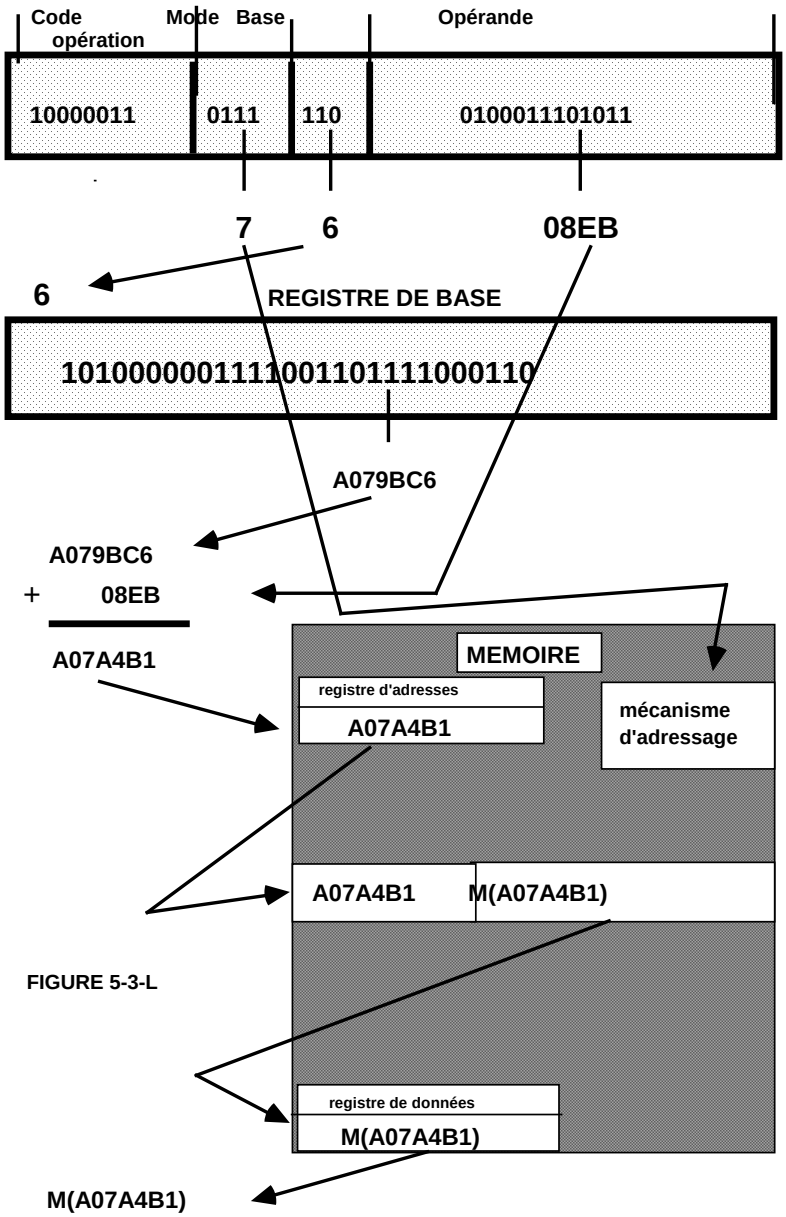


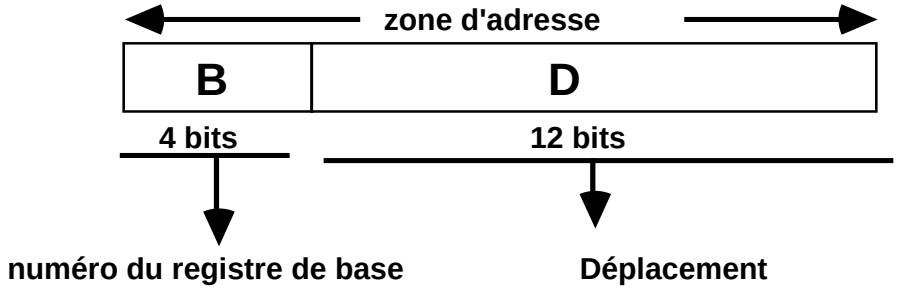

FIGURE 5-3-L

D'une autre façon, on peut voir B comme pointant sur une zone de longueur égale à l'étendue possible du déplacement D, c'est l'espace adressable au moyen du registre B. Comme dans notre exemple, le déplacement étant sur 13 bits, la longueur de cette zone est 213 adresses ou 8K .

De plus, ceci constitue une protection automatique si le contenu du registre B est sous le contrôle du système d'exploitation qui donne l'accès au programmeur sous réserve d'un mot de passe. En effet, dans ces conditions, la zone entre l'adresse 0 et l'adresse R(B), de même que celle entre l'adresse R(B) + 1 et l'adresse maximale de la mémoire adressable, ces zones sont inaccessibles. Notons que l'adresse relative peut être aussi indexée. On a alors des modes d'adressage mixtes. Si c'est le cas, on a alors une adresse formée de trois parties, soient la base, l'index et le déplacement.

Dans le cas où le contenu du registre de base B ,de longueur égale à N bits de position, est sous le contrôle du superviseur, c'est-à-dire sous le contrôle de la procédure du système d'exploitation chargé de contrôler l'enchaînement et la gestion des processus en optimisant les ressources disponibles, on peut dégager des propriétés intéressantes. L'espace accessible défini par le déplacement D étendu sur K bits, donc de longueur de 2K adresses, est fini et cet espace est dénommé une page. Le superviseur peut alors effectuer une affectation de la mémoire disponible de façon plus efficace en allouant à chaque programme ou partie de programme, une ou plusieurs pages. Pour cela, il lui suffira de laisser les K derniers bits du registre B à zéro. Ainsi, la mémoire apparaîtra comme formée de blocs de taille unique et égale à celle d'une page. La gestion de la mémoire est alors beaucoup plus souple à l'aide du superviseur qui contrôle les N - K premiers bits du registre de base B. Cette façon fournit aussi au superviseur une méthode facile de protection de la mémoire contre les accès du programme à des zones interdites de la mémoire principale comme celles entre autres occupées par le superviseur lui-même.

**Exemple:** Supposons une partie adresse de 16 bits dont 4 servent à indiquer le numéro d'un des 16 registres de base et les 12 autres à contenir la valeur du déplacement




Considérons l'opérande 51FF en notation hexadécimale exprimée en adressage relatif. Les quatre bits de plus fort poids indiquent alors le registre portant le numéro 5 tandis que le déplacement a pour valeur 1FF. Supposons qu' on a comme contenu au registre numéro 5, la valeur suivante:

| Numéro du registre de base | 5 | 0 0 0 2 F 5 |
|---|---|---|

L'adresse physique sera alors donnée par l'addition de l'adresse de base 0002F5 et du déplacement 1FF, ce qui donne 0004F4.

Comme on peut s'en rendre compte, l'adresse effective réfère à une mémoire de $2^{24}$ mots, soit 16 Mbits, malgré le fait que l'opérande n'a qu'une étendue de 16 bits, ce qui ne permettrait, en adressage direct, qu' un espace mémoire de $2^{16}$ mots, soit de 64 K .

**Exemple:** Comme dans l'exemple précédent, on désire adresser le mot d'adresse physique 00F3BA dans une instruction et utiliser comme registre de base, le registre numéro 10, sachant que l'adresse de base est 00F000 . Puisque l'adresse effective est égale à l'adresse de base R(B) plus le déplacement D, on doit avoir 00F3BA = 00F000 + D. On en déduit que D = 3BA. Aussi l'opérande sera A3BA en adressage relatif puisque la référence doit se faire par rapport au registre de base B numéro 10, c'est-à-dire A en notation hexadécimale.

Cette économie de bits de position, raccourcissant d'autant les instructions, s'accompagne d'un gain marqué de souplesse dans la gestion de la mémoire. Pour illustrer cela, considérons un programme dont la taille maximale est inférieure 4K, soit 4096 octets. Cette restriction dans la taille du programme n'est là que pour simplifier la discussion. Rappelons qu'on doit loger ce programme à quelque part dans la mémoire principale. L'objectif maintenant est de former les instructions du programme en langage machine sans faire référence à des adresses réelles mais plutôt en faisant appel à des adresses relatives. Pour ce faire, on construit les instructions en prenant pour acquis que la première instruction du programme occupera le mot d'adresse 0. Les opérandes sont alors établis par rapport à l'adresse de base 0.

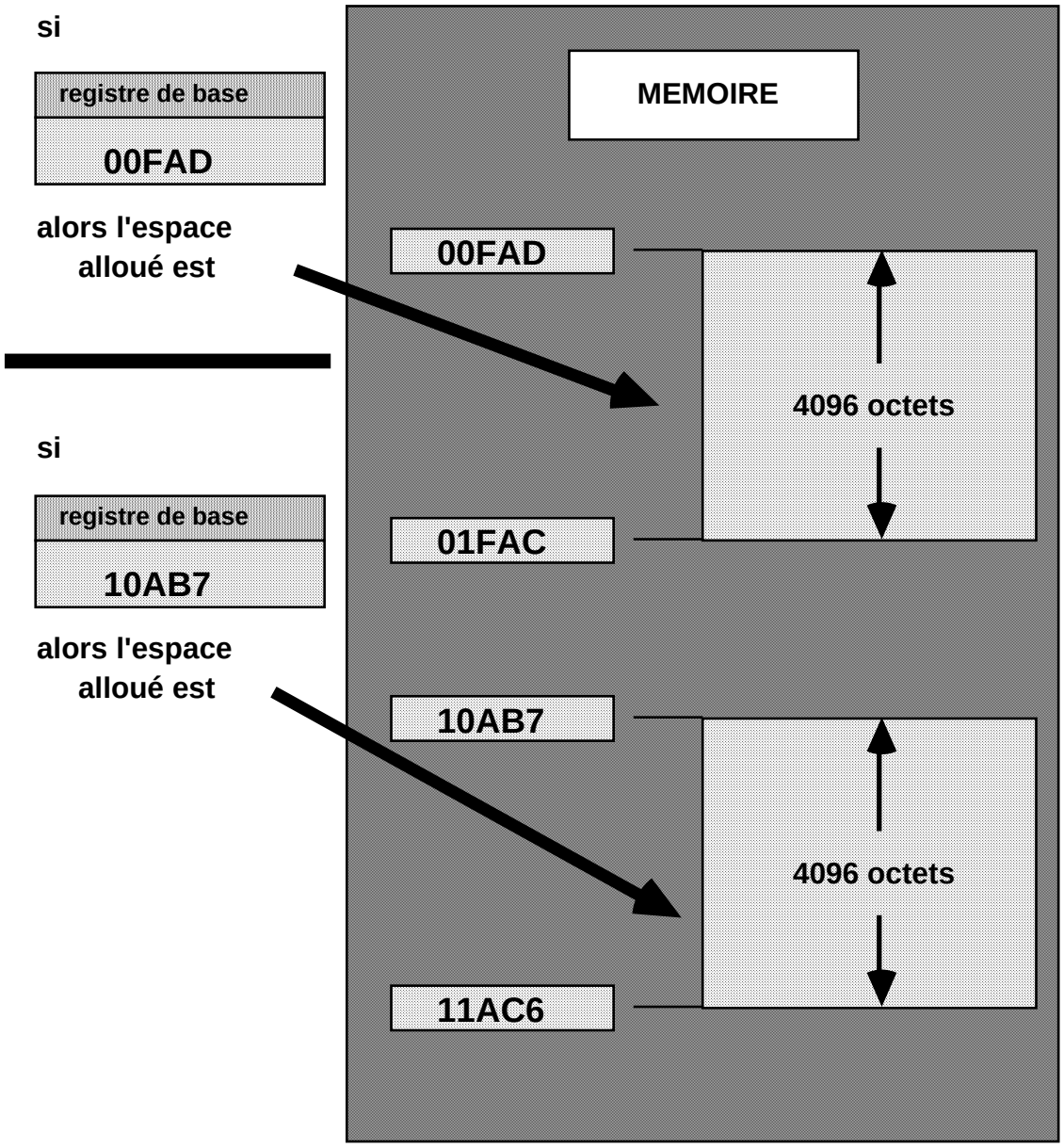
À titre d'exemple, si le programmeur a indiqué le registre numéro 12, ou C en hexadécimal, comme registre de base, le déplacement d'un opérande se référant au $30^{ième}$ octet du programme sera $30_{10} = 01E_{16}$ et l'adresse relative sera C01E exprimée en notation hexadécimale. Le déplacement admissible pour le dernier mot du programme sera $FFF_{16}$, puisqu'on dispose de 4096 octets dont les adresses relatives vont de 0 à $2^{12}$ - 1. L'opérande se référant à ce mot sera CFFF, où le registre numéro 12 est toujours le registre de base.

Ce travail d'adressage relatif pour nos programmes écrits en langage évolué, on entend ici par évolué tout langage n'étant pas le langage machine étant le seul que l'ordinateur peut reconnaître directement, est réalisé par le compilateur et l'éditeur de liens. Le programme ainsi obtenu sera chargé à son exécution, non pas à partir de l'adresse réelle 0, mais à une adresse quelconque, servant d'adresse de base, connue seulement au moment du chargement du programme en mémoire. Pour nos programmes en langage évolué, cette adresse de base est déterminée par le système d'exploitation, le gérant de l'espace mémoire.

Supposons qu'il doit être chargé à partir de l'adresse 200FA7. Cette adresse réelle devient alors l'adresse de base, c'est-à-dire qu'elle devient le contenu du registre de base numéro 12 dans cet exemple au moment de l'exécution. Ce chargement du registre de base est assumé pour nos programmes écrits en langage évolué par le système d'exploitation qui connaît en tout temps les espaces disponibles de sa mémoire physique.

Le même programme machine pourra être éventuellement exécuté à nouveau à partir d'une autre adresse (nouvelle adresse de base) sans aucune modification préalable des instructions du programme. La figure suivante résume cette idée de translation de programme en mémoire principale. Si l'adresse de base est 00FAD, la zone réservée au programme sera entre les adresses réelles 00FAD et 0FAD + FFF, soit 01FAC. D'autre part, si l'adresse de base est fixée par le

système d'exploitation à la valeur 10AB7, l'espace alloué au programme ira de l'adresse réelle 10AB7 à l'adresse réelle 10AB7 + FFF, soit 11AC6.




Notons que le contrôle de l'allocation de la mémoire peut être fait par le programmeur au niveau du langage machine ou de façon automatique par le superviseur pour nos programmes écrits en langage évolué. La technique d'adressage relatif procure une méthode efficace pour la protection de la mémoire, particulièrement pour les zones interdites aux programmes externes comme celles occupées et réservées au système d'exploitation.

Habituellement, l'unité de commande est munie de deux registres spéciaux contenant l'adresse de début et l'adresse de fin de la zone permise ( ou autorisée). Si une adresse réfère à un mot situé à

l'extérieur de la zone autorisée, un message d'interruption est alors généré automatiquement et émis à l'unité de contrôle et l'exécution du programme est immédiatement suspendue.

### 4.7.8. L'adressage progressif

Certains ordinateurs possèdent des registres d'index spéciaux ayant la propriété de s'incrémenter ou de se décrémenter de 1 automatiquement à chaque fois qu'ils sont utilisés pour l'adressage. Le VAX entre autres en possède de chaque sorte. Supposons, pour illustrer, une adresse A et un index I dont le registre qu'il réfère possède la propriété de s'incrémenter. Le premier nom de mémoire ou adresse physique atteint par ce couple sera A + R(I), d'après la technique de l'adressage relatif. Par la suite, le mécanisme d'incrémentation sera actionné changeant automatiquement le contenu R(I) en R(I) + 1. Par conséquent, à la prochaine utilisation de ce registre de base en mode d'adressage relatif, l'adresse physique calculée sera A + R(I) + 1, et ainsi de suite. On peut réaliser de cette façon des balayages séquentiels à l'aide d'un registre d'index auto incrémenté dans les structures répétitives entre autres utilisant un pas de 1. On peut facilement imaginer une foule d'autres usages possibles.

### 4.7.9. L'adressage associatif

Dans ce mode d'adressage, on recherche en mémoire par comparaison d'éléments d'information dont on dispose, qu'on appelle descripteur, avec le contenu de chaque emplacement mémoire d'une zone donnée. Ce mode d'adressage est aussi appelé adressage par contenu.

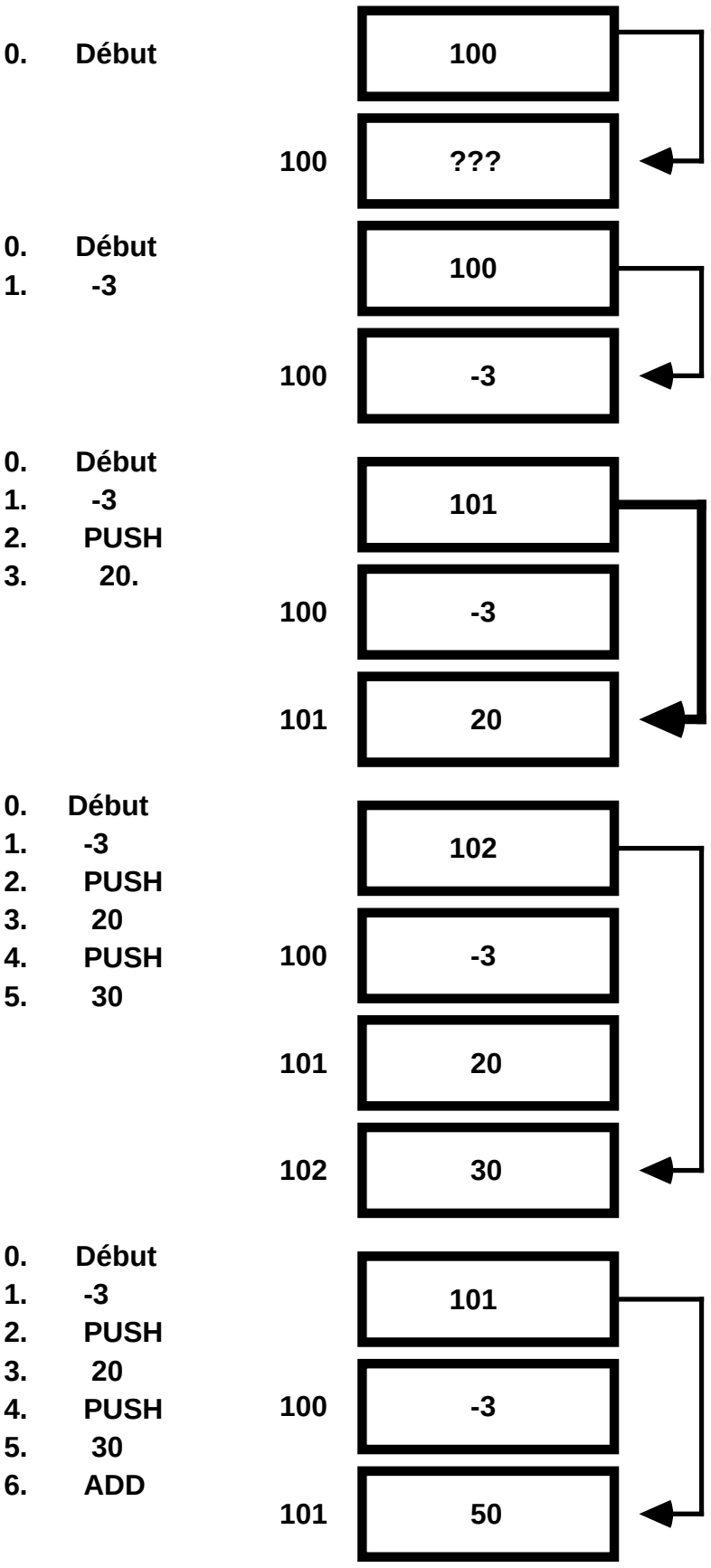
### 4.7.10. L'adressage par pile.

Les opérations associées à ce mode d'adressage sont sans opérande. Elles se réfèrent à un registre spécial appelé registre de pile qui contient un pointeur de pile, correspondant à l'adresse du sommet de la pile. Ce registre est matériellement associé à une adresse de début de pile ou de base. La plupart des ordinateurs possède cette structure particulière d'adressage. Les opérations nécessairement présentes en adressage de pile sont celle d'entrée de données (PUSH) et celle de sortie (POP). Voyons un exemple.

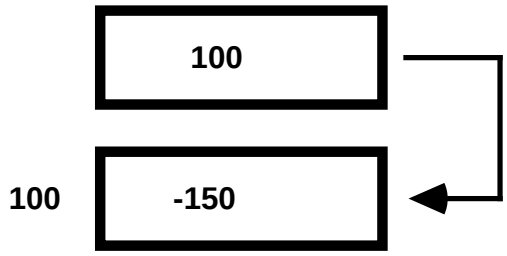
Supposons un registre de pile associé à l'adresse de base 100. Initialement, la pile est vide. Le pointeur a la valeur 100 et le contenu du mot d'adresse 100 est indéterminé. Voyons l'effet qu'aura le programme suivant sur cette pile:

```
0.  Début
1.  -3
2.  PUSH
3.  20
4.  PUSH
5.  30
6.  ADD
7.  MULT
8.  POP
```

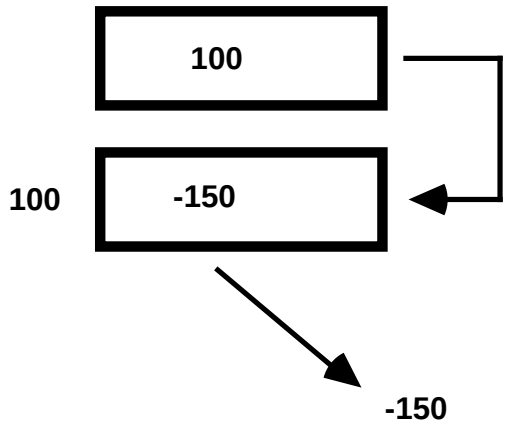
Les schémas suivants montrent l'état de la pile après chaque instruction de la séquence.


0. Début
100
100
???
0. Début
1. -3
100
100
-3
0. Début
1. -3
2. PUSH
3. 20.
101
100
-3
101
20
0. Début
1. -3
2. PUSH
3. 20
4. PUSH
5. 30
102
100
-3
101
20
102
30
0. Début
1. -3
2. PUSH
3. 20
4. PUSH
5. 30
6. ADD
101
100
-3
101
50

```
0.   Début
1.    -3
2.    PUSH
3.    20
4.    PUSH
5.    30
6.    ADD
7.    MULT
```




```
0.   Début
1.    -3
2.    PUSH
3.    20
4.    PUSH
5.    30
6.    ADD
7.    MULT
8.    POP
```




### 4.7.11. L'adressage implicite

Dans ce mode d'adressage, l'opérande est fixe et connue implicitement par la machine. Il s'adresse donc aux instructions sans opérande . Aussitôt que le code opération détecté, l'ordinateur agit instantanément sur la mémoire impliquée implicitement. Les opérations d'entrée et de sortie d'une donnée dans une pile (PUSH et POP) sont des instructions de ce type. Certaines opérations de chargement de registres, de mise à zéro (RAZ) et de décalage sont de ce type et utilise l'adressage implicite.