**Chapitre 1**

# STRUCTURE INTERNE d'un ORDINATEUR

## 1 INTRODUCTION

Chaque fois que nous utilisons une simple calculatrice à quatre opérations, nous agissons très exactement comme le fait un ordinateur soit disant beaucoup plus élaboré. La structure interne d'un ordinateur est en effet, contrairement à certaines idées reçues, relativement simple à comprendre.

Notre but n'est évidemment pas de décrire, ni détailler la structure interne de telle ou telle machine, fabriquée par tel ou tel constructeur, mais de présenter les concepts fondamentaux de l'organisation et du fonctionnement logique de ces machines.

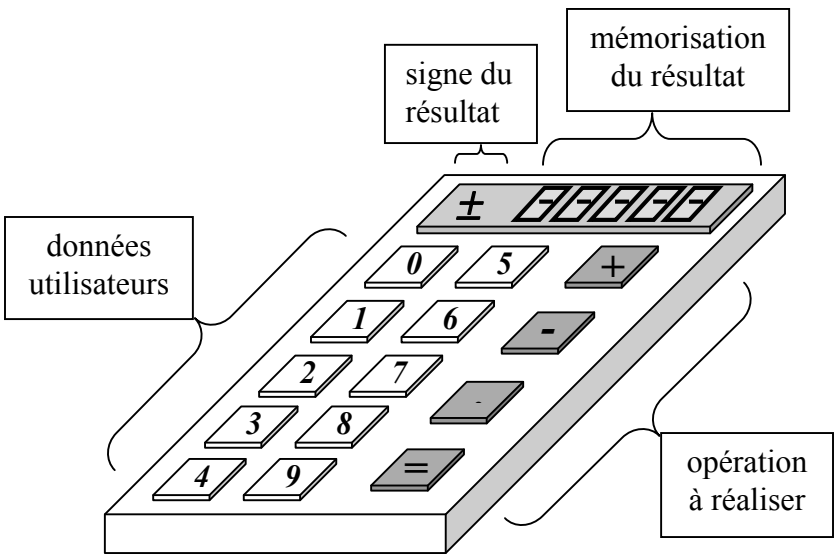
Les **principes** décrits sont applicables à pratiquement toutes les machines actuelles de type VON NEUMANN et WILKES. Il s'agit donc d'une **modélisation** qui pourra être utilisée quelle que soit la marque ou bien le type de machine étudiée.

Les circuits logiques, combinatoires et/ou séquentiels, nécessaires à cette modélisation ont été définis et synthétisés, dans les deux volumes : 'bit après bit' et 'séquence après séquence', publiés dans cette même collection.

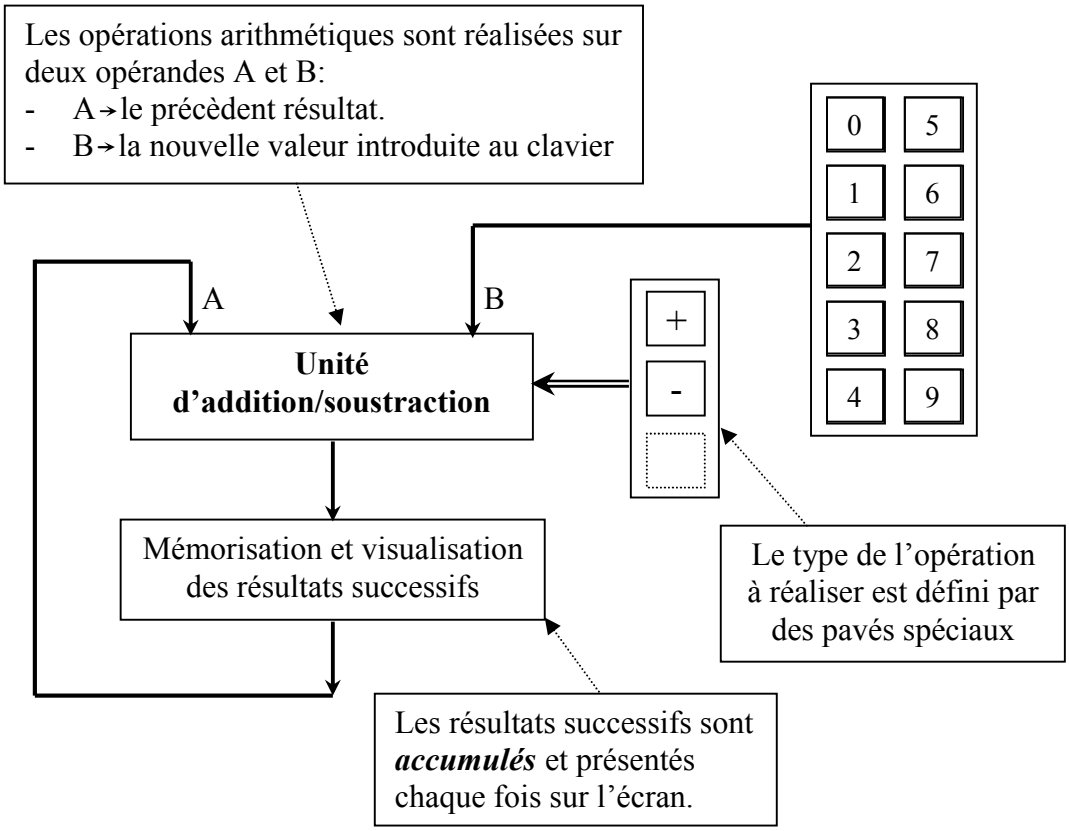
Après un rappel des fonctions logiques réalisées par ces circuits nous les représenterons par des schémas blocs afin de clarifier le modèle.

Dans une première partie nous présentons les **circuits logiques du modèle** ; dans la deuxième partie nous nous intéresserons au **fonctionnement temporel** de ces circuits.

On peut facilement identifier les éléments fondamentaux composant un ordinateur en observant une simple calculette.



On peut de même présenter une structure interne très simplifiée de cette calculette :




Il est évident que cette organisation, quoique opérationnelle au niveau des opérations élémentaires, manque totalement « *d'intelligence* », si tant est que l'on puisse parler d'intelligence pour une machine.

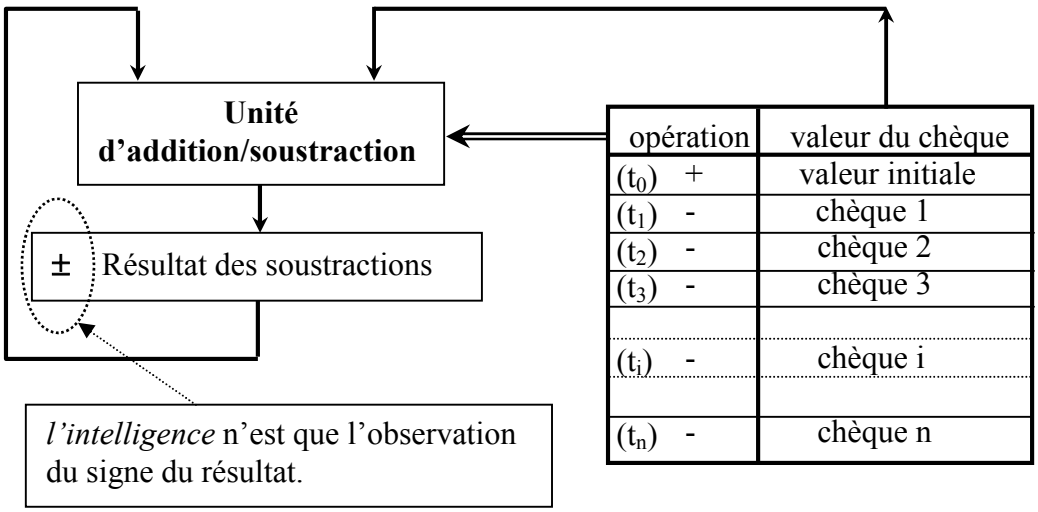
On peut constater en effet que cette machine ne peut en aucun moment signaler par exemple un changement de signe du résultat. Seule l'observation de l'écran par l'opérateur peut alerter ce dernier.

*Exemple :* Etant en vacances je ne peux pas consulter mon compte bancaire, mais prudemment j'ai noté la dernière valeur de mon compte avant de partir. Ayant réalisé beaucoup d'opérations de débit grâce à mon carnet de chèques, je désire savoir le solde de mon compte.
A l'aide de ma calculette, ayant introduit la dernière valeur de mon compte, je soustrais successivement les montants des chèques que j'ai notés sur les talons de mon carnet.
Ayant malheureusement dépensé beaucoup d'argent, je crains fort que mon compte soit négatif avant la fin du mois. A chaque soustraction, la question posée est donc :

- le signe affiché sur l'écran est-il toujours positif ?

Si le solde de mon compte devient négatif, je devrais arrêter mes vacances sinon je pourrais rester quelques jours de plus.

Le signe du résultat des soustractions représente « ***l'état*** » de mon compte : positif ou négatif.

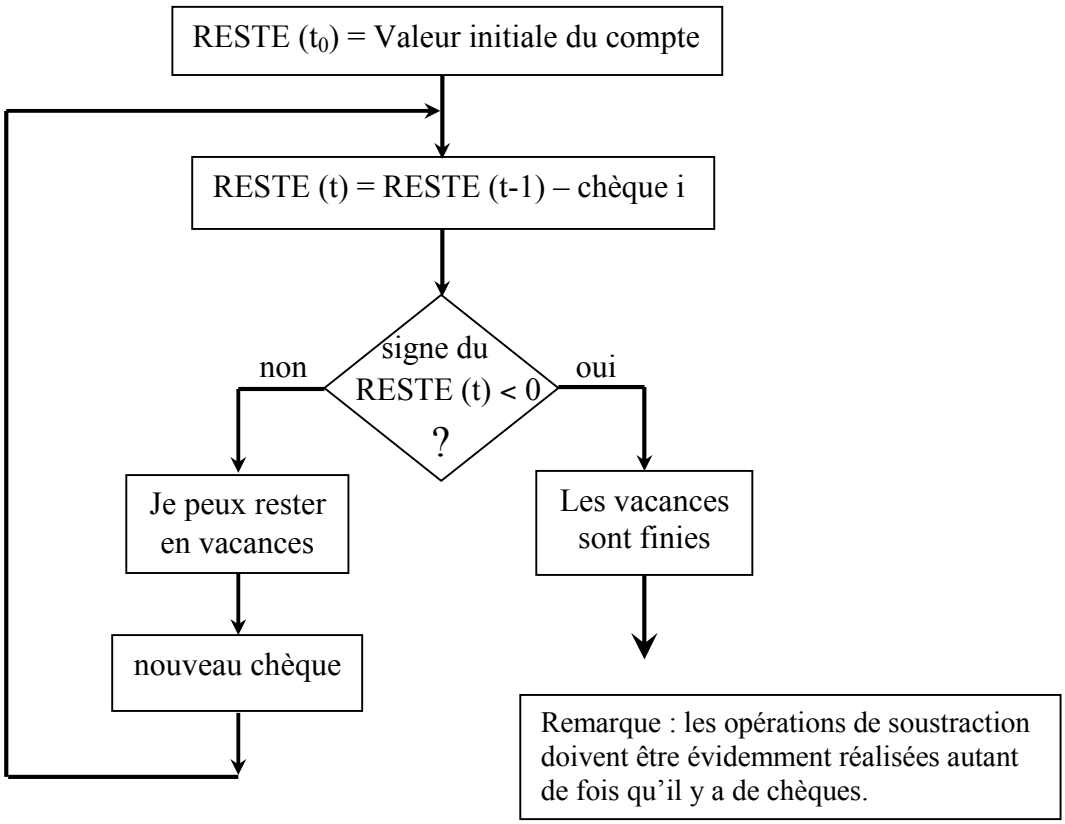



Comme on peut le constater sur cet exemple, la calculette a réalisé un calcul (succession de soustractions), mais c'est l'analyse du signe *par l'homme* qui permet de prendre une décision : rester en vacances ou bien rentrer.

Un ordinateur n'est guère capable de faire mieux, nous verrons qu'il existe plusieurs indicateurs *d'états* qui permettent de changer la séquence des événements.

La valeur de ces indicateurs d'états est binaire, les choix ou les décisions qu'ils permettent de prendre sont donc extrêmement simples : SI - ALORS - SINON

On peut aussi représenter ces indicateurs sous forme d'organigramme à l'aide d'un losange.



Dans un ordinateur, les données de l'utilisateur (valeur des chèques), comme la succession des opérations (soustractions successives) sont contenues (mémorisées) dans une partie spéciale de la machine que l'on appelle mémoire. Il existe différents types de mémoires que nous détaillerons ultérieurement, mais nous pouvons en donner une modélisation simple

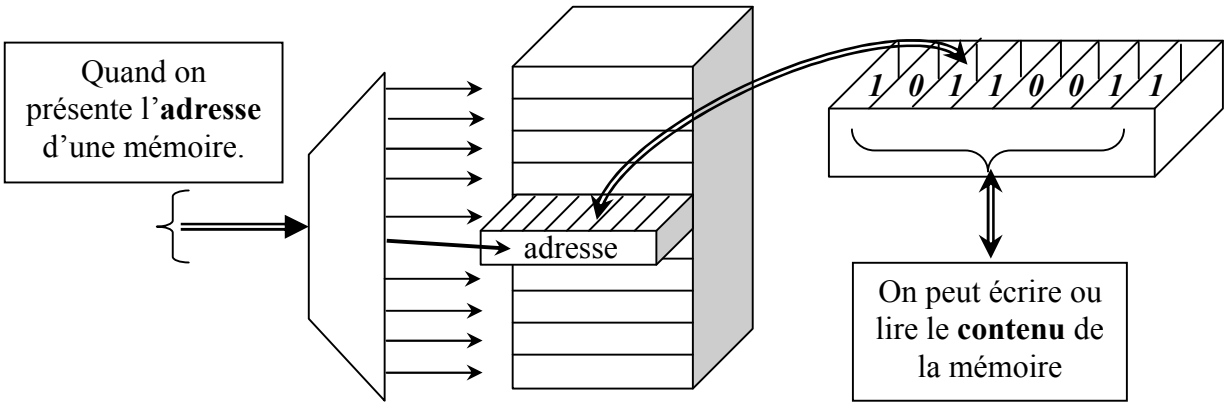
## 2 – MODELISATION d'une MEMOIRE

Nous avons présenté dans le volume intitulé 'séquence après séquence' combien il était facile de réaliser une mémoire élémentaire à l'aide de portes ET/OU/PAS.

Les technologies actuelles d'intégration à grande échelle permettent de réaliser des millions de cellules élémentaires organisées en 'mots-mémoire'.

Ces mots-mémoire contiennent une information binaire de 8, 16, 32 (ou plus) bits, dont la lecture et l'écriture sont simultanées. Nous devons cependant remarquer que dans certains micro-contrôleurs, il est possible de lire bit à bit ces mots-mémoire.

Le principe d'accès à une information en mémoire est dans son principe excessivement simple :




**Attention :** **Ne jamais confondre *adresse* et *contenu* d'une mémoire. L'adresse mémoire a toujours beaucoup plus de bits que le contenu lui-même.**
exemple : 24 bits peuvent adresser 1,6 millions de cases mémoires d'un octet chacune.

Pour adresser un élément d'une mémoire il faut évidemment 'ouvrir' cette case mémoire sans aucune ambiguïté. L'adresse doit donc comporter **n bits** pour adresser $\mathbf{2^n}$ cases mémoires.

Nous présenterons ultérieurement comment peut être structurée une mémoire et en particulier les notions de blocs, de segmentation, de pagination des mémoires virtuelles.

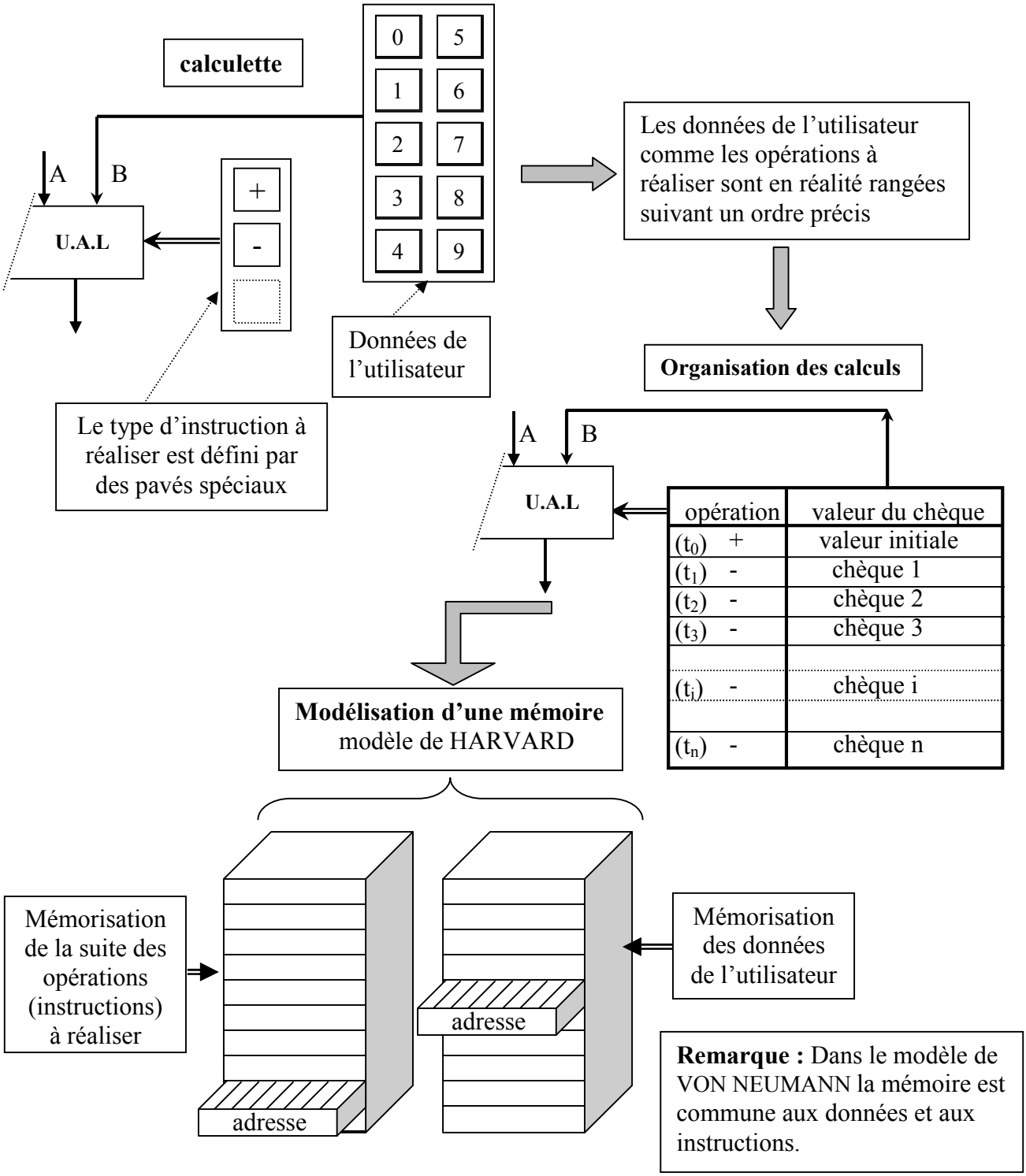
Dans une première étape, et pour simplifier, nous considérons qu'il est toujours possible d'accéder à **tous les éléments de la mémoire**.

Si le principe (ou le modèle) d'accès au contenu d'une mémoire est très simple, il est par contre beaucoup plus délicat de générer *la bonne adresse* au *bon moment.*

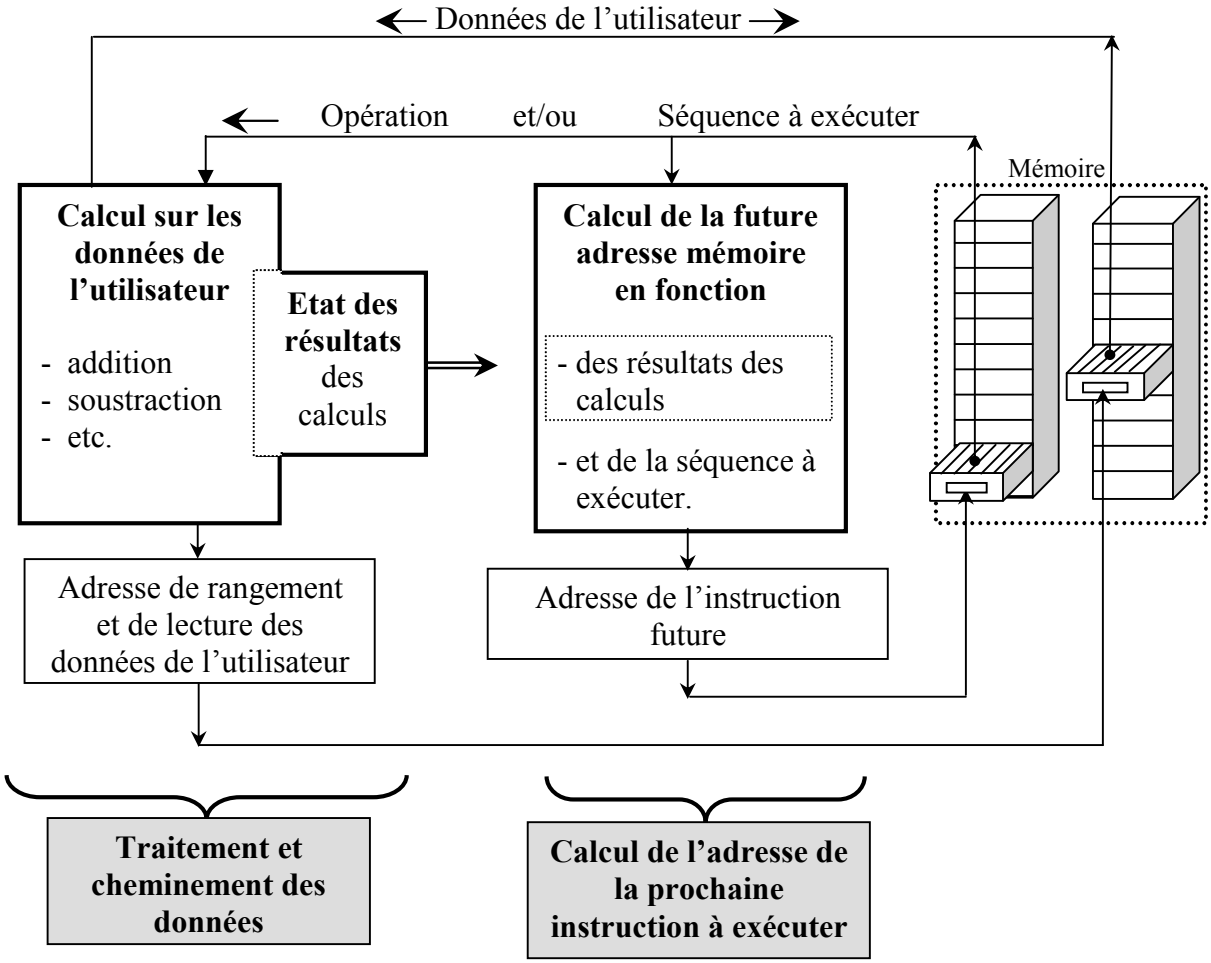
Nous avons vu au début de ce chapitre que le séquencement des opérations successives était parfois dépendant du résultat de certaines opérations. Il est donc nécessaire de pouvoir accéder non seulement aux données de l'utilisateur, mais aussi aux différentes séquences, prévues par le programmeur ou bien nécessaires au bon fonctionnement de la machine.

Nous pouvons donc modéliser la mémoire d'un ordinateur à partir des deux fonctions principales :

1- mémoriser les données de l'utilisateur.
2- mémoriser la suite des instructions du programme lui-même.



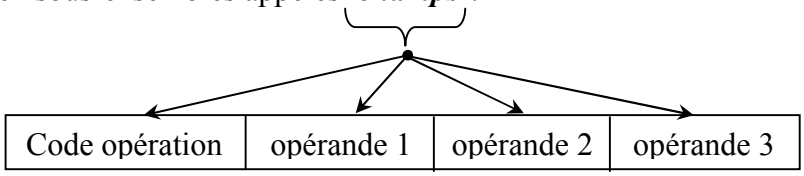
Outre la mémoire, on peut diviser un ordinateur en deux grandes parties.




Toutes les machines actuelles, du plus petit microprocesseur à l'ordinateur le plus élaboré, intègrent ces deux circuits.

Nous appelons 'séquence' la succession des ***'instructions'*** élémentaires que doit réaliser la machine pour accomplir une tâche complète, une addition par exemple.

La structure d'une instruction peut être représentée par un assemblage (concaténation) de bits groupés en sous-ensembles appelés ***'champs'***.




*exemple théorique*: ADD 18 R2 Mémoire 45

qui se lit : additionner 18 au registre R2 et ranger le résultat dans la mémoire 45

Le nombre des champs dépend :
- d'une part de l'instruction elle-même,
- d'autre part de la structure de la machine, en particulier du nombre de registres de travail de la machine.

Les opérandes peuvent représenter :
- des valeurs immédiates,
- des adresses de données contenues en mémoire ou dans un registre de travail,
- des adresses mémoires de branchement dans le cas de rupture de séquence.

Toutes les instructions n'ont pas le même nombre de champs, elles sont directement fonction de la séquence définie par le programme.

Ces séquences peuvent être définies par le programmeur lui-même, par exemple lorsqu'il travaille dans un langage très proche de la structure de la machine comme *'l'assembleur'*.

Ces séquences peuvent aussi être prédéfinies au sein de logiciels existants afin de faciliter la mise en œuvre de programmes dits en langages évolués.

Dans ce chapitre nous ne nous intéresserons qu'aux instructions les plus proches de la structure de la machine, car ce sont elles qui *activent* les circuits décrits.

Nous présentons successivement une modélisation des trois fonctions de base réalisées dans un ordinateur, fonctions que tout utilisateur de ces machines se doit de connaître :

- **l'adressage des données,**
- **le traitement et le cheminement des données,**
- **l'adressage des instructions du programme lui-même.**

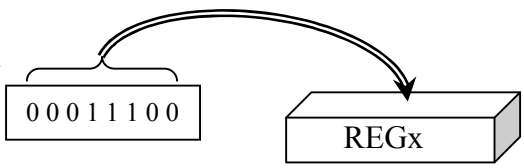
## 2- 1 Adressage des Données (rangement et/ou lecture)

Il existe de nombreuses possibilités pour un programmeur de définir l'adresse de la mémoire où il désire ranger ses propres données. C'est essentiellement en fonction de son application qu'il choisira tel ou tel type d'adressage.
Nous présentons les types d'adressage les plus courants en s'appuyant sur des exemples en langage *assembleur*.
Il faut cependant noter que les concepts d'adressage indirect en particulier peuvent faire l'objet de développements spécifiques en langages *évolués*.

♦ adressage **immédiat**

Le champ opérande contient la valeur désirée.

*exemple :* MOV REGx, 28d

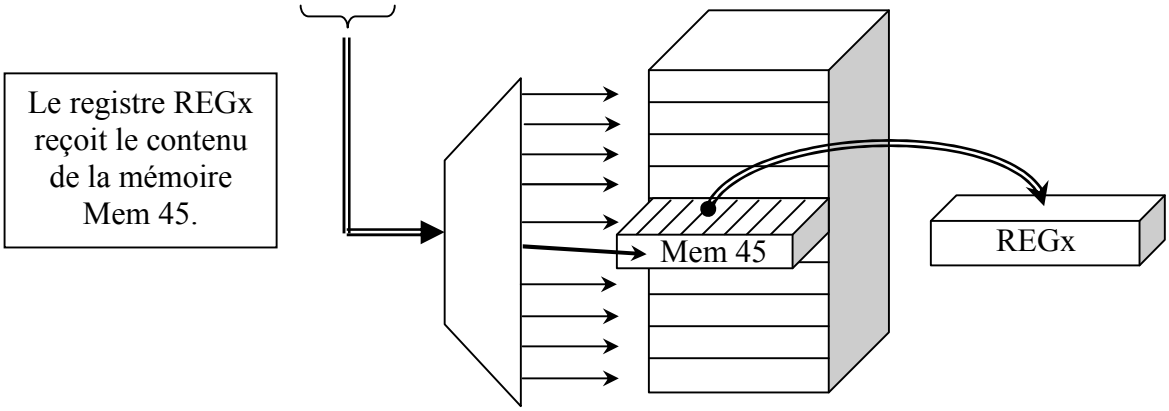



Le registre REGx reçoit immédiatement la valeur décimale 28.

♦ adressage **direct**

Le champ opérande contient l'adresse effective de la mémoire concernée.

*exemple :* MOV REGx, Mem 45




♦ adressage **implicite**

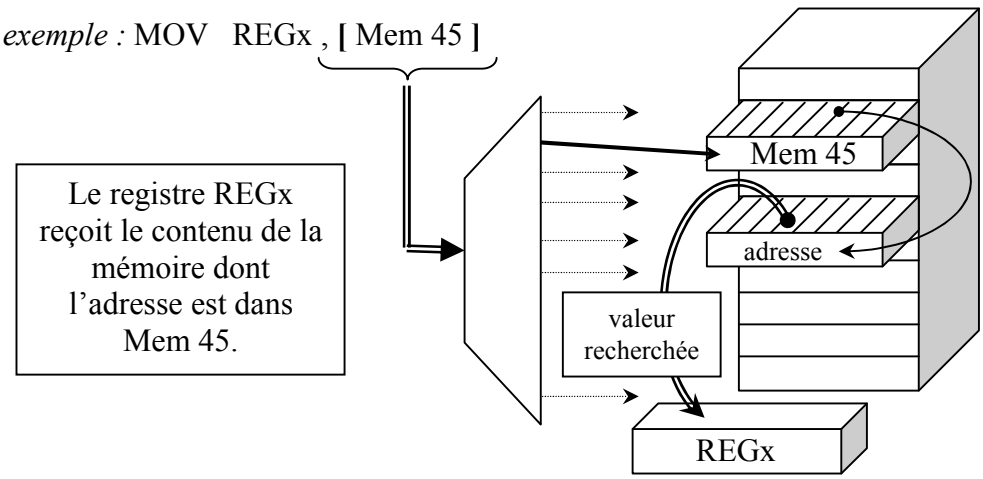
Il ne s'agit pas vraiment d'adresser une mémoire, mais plutôt un registre de travail.
Le champ code opération est alors suffisant pour définir l'opérande.

*exemple :* INC La machine incrémente l'accumulateur.

♦ adressage **indirect**

Le champ opérande contient l'adresse de ma mémoire **où** se trouve l'adresse effective de la mémoire concernée.

*exemple :* MOV REGx , **[** Mem 45 **]**




***Remarque :***

Nous avons écrit la mémoire Mem 45 entre crochets **[ ]** , cette symbolique très utilisée permet de spécifier sans ambiguïté un adressage indirect.

L'instruction écrite se prononce :
*Le **contenu** de l'**adresse mémorisée** dans Mem 45 **va** dans le REGx.*