Rôle de la pile interne dans l'exécution d'un programme

1 Notions d'architecture des ordinateurs

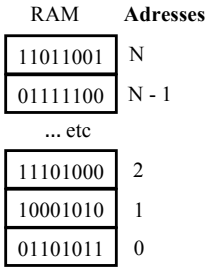
Le coeur d'un ordinateur comprend essentiellement :

- Une Unité Centrale (UC) ou microprocesseur. C'est le cerveau de la machine et il effectue tous les calculs. En revanche, il possède peu de mémoire et donc il ne peut stocker que peu d'informations à la fois.

- Une mémoire vive (RAM = *random access memory*) qui contient tous les programmes qui en train d'être exécuté sur la machine : système d'exploitation (Windows, OS X ou Linux), programmes tels que cette visionneuse de diapos.

- Une mémoire de masse : disque dur, clé USB, disque CD ou DVD. C'est ici que sont stockés tous les fichiers et programmes qui ne sont pas exécutés.

La mémoire vive est organisée comme une série de tiroirs placés les uns sur les autres (cases mémoires). Chaque tiroir est de taille fixe et peut contenir exactement 1 octet.



D'autre part, chaque case mémoire possède un *numéro unique* qui l'identifie complètement : ce numéro constitue l'**adresse** du tiroir [1]. La numérotation commence toujours à 0 pour la première case, puis 1 pour la seconde, etc..., jusqu'au numéro N de la dernière case, qui en général est très élevé.

Chaque case est relié à l'Unité Centrale par une série de fils de connexions (qui sont en fait gravés sur un circuit imprimé). L'ensemble de tous ces fils est appelé **BUS de transfert de données**. Ainsi, à chaque instant, l'UC peut envoyer des données informatiques vers chacune des cases de la RAM ou en recevoir de chacune d'entre elles.

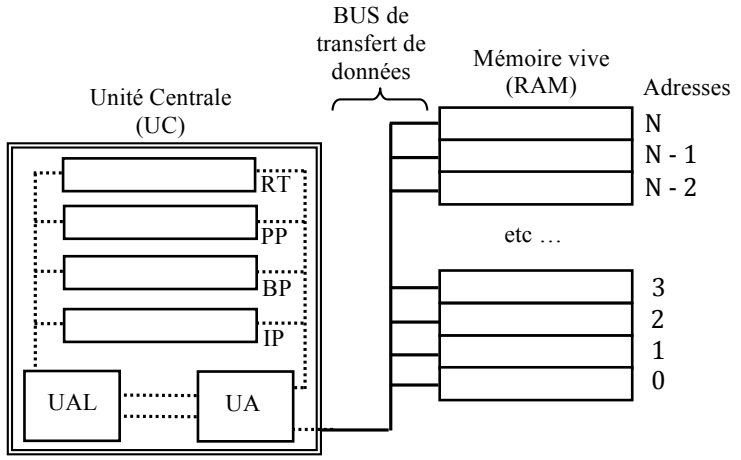


Figure 1 Architecture simplifiée de la RAM et de l'UC d'un ordinateur.

1. Tout comme une adresse postale qui identifie complètement une maison.

L'Unité Centrale (UC) est constituée :

- d'une **Unité de Traitement Arithmétique et Logique** (UAL) chargée d'exécuter les instructions, à un rythme cadencé par une horloge interne,
- d'une **Unité d'Adressage** (UA) chargée du transfert des données depuis et vers la RAM,
- de quelques cases mémoire (une dizaine environ) appelés **registres**, qui servent à stocker temporairement les informations reçues de la RAM ou destinées à être envoyée vers la RAM. Leurs noms exacts dépend du type de processeur utilisé (par exemple Intel ou Motorola). L'UAL ne travaille qu'avec les données contenues dans ces registres : elle les additionne, les soustrait, les divise, les comparent, les augmentent, etc...

Dans la suite, 4 registres particuliers vont nous intéresser (nous allons en décrire l'utilité et le fonctionnement plus tard). Il s'agit des registres :

- **BP** : pointeur de base.
- **PP** : pointeur de pile.
- **IP** : pointeur d'instructions.
- **RT** : registre de travail. Dans la réalité, il y a plusieurs registres de travail.

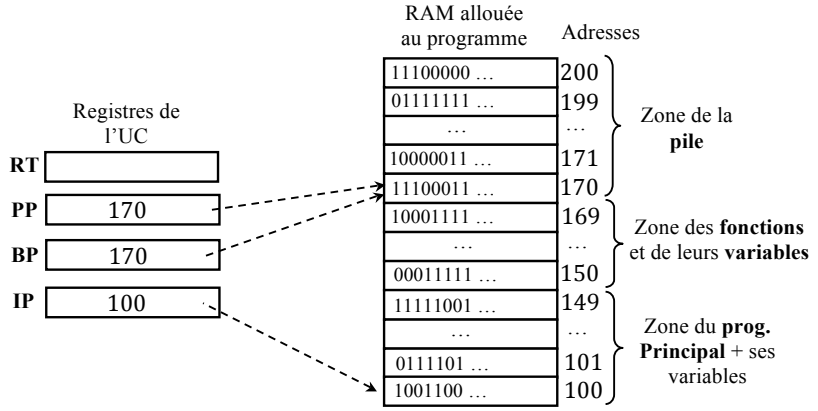
2 Chargement et exécution d'un programme en mémoire

Lorsque vous écrivez un programme dans un fichier et que vous lancez son exécution, tout le programme (instructions, variables utilisées, fonctions, ...) va être traduit dans le langage de la machine et chargé dans la RAM.

Le système d'exploitation alloue au programme un bloc contigu de tiroirs RAM, divisé en plusieurs zones.

- Une première zone contient tout le code du programme principal (celui du fichier qui est exécuté) ainsi que ses variables. Au tout début de l'exécution le registre **IP** contient l'adresse du première case mémoire de cette zone.
- Suivent plusieurs zones qui contiennent les codes de chaque fonction définie dans ce programme et de ses variables (celles qui sont définies dans le corps de la fonctions).
- Une dernière zone est allouée au programme et constitue une **pile**. L'adresse du première case mémoire de cette pile est contenant dans le registre **BP** et l'adresse du sommet de la pile est dans **PP**. Au début du programme, la pile est vide : les deux registres contiennent donc la même adresse.

Prenons l'exemple d'un programme chargé en mémoire RAM et qui occupe les tiroirs d'adresses 100 à 200. Le **code** du programme et ses variables est placé entre les adresses 100-149, la zone réservée aux fonctions s'étend des adresses 150 à 169 et la **pile** occupe la zone située entre les adresses 170 et 200.



Au début de l'exécution du programme, le pointeur d'instruction **IP** contient l'adresse de la première instruction du programme. Les pointeurs **BP** et **IP** définissent la pile : au début, celle-ci est vide (les contenus de BP et IP sont identiques).

Comment sont stockées les variables ?

Si vous avez défini une variable **Age** dans le corps de votre programme (ou encore dans le corps d'une fonction), avec par exemple une instruction de la forme **Age** = 21 :

- Au moment où vous demandez l'exécution du fichier, Python (ou n'importe quel autre langage de programmation) demande au système d'exploitation (Windows, OX S X, ...) de lui donner une adresse dans la RAM pour stocker cette variable.
- L'OS (operating system) regarde s'il existe encore de la place et, dans l'affirmative, alloue le nombre de cases mémoires nécessaires pour stocker le contenu de la variable : 1 octet, 2 octets, 8 ou plus si nécessaire. Ces cases sont jointives : elle sont "collées" les unes autre autres.
- L'adresse de votre variable est celle de la case mémoire la plus basse.
- Toute référence au nom de la variable a disparu : l'UC et la RAM ne connaissent pas **Age**. Ils ne connaissent qu'une adresse (3200 par exemple) et une suite de cases mémoires qui contiennent la valeur de Age (21 ici).

Comment s'exécute un programme ?

1. L'UAL demande à l'UA de lui transférer le contenu stocké dans le tiroir situé à l'adresse contenue dans **IP**. l'UA obéit et transfère ce contenu dans l'UAL.
2. l'UAL exécute l'instruction qui lui a été transmise.
3. Le contenu de **IP** est augmenté d'une unité et désigne donc l'adresse de la **prochaine instruction**.
4. Le cycle peut recommencer au point 1. L'UAL demande à l'UA de lui transférer l'instruction suivante, etc...

Une horloge interne impose **une cadence** : l'ensemble des étapes 1-2-3 est réalisée en 1 cycle, 2 ou 3 cycles d'horloge (selon la complexité de l'instruction). Un cycle d'horloge T est appelé **unité de temps machine** (utm). Actuellement, il est de l'ordre de la nano-seconde (les ordinateurs les plus rapides atteingnent 10^{-15} voire 10^{-17} secondes d'utm). L'inverse de l'utm $f = 1/\mathrm{T}$ est la **fréquence de l'horloge**, typiquement de l'ordre du GigaHz pour les ordinateurs personnels actuels.

3 À quoi sert la pile ?

La pile est utilisée pour stocker des informations temporaires nécessaires au bon déroulement du programme.

En particulier, elle est utilisée lors des **appels de fonction** :

- pour passer un paramètre à la fonction : celui-ci est empilé sur la pile.
- La fonction récupère ce paramètre en le dépilant.
- De même, si la fonction retourne une valeur, elle le fait en la déposant sur la pile. Le programme n'a plus qu'à la récupérer en dépilant cette valeur.

Illustration avec une animation que vous pourrez trouver sur le site de la MP1 sous le nom **AnimationProg.ppt** (c'est un fichier powerpoint). On considère le programme Python suivant :

```
1  def f(x) :
2    y = x + 3
3    return y
4
5  # Début du programme
6  a = 3
7  b = f(3)
```

Lors de la traduction dans le langage de la machine, les sauts de lignes et les commentaires sont purement et simplement ignorés.

- Le code et les variables **a** et **b** du programme principal sont placés dans un premier bloc de la mémoire.
- Le code de la **fonction** f et sa seule variable **y** sont placés dans un autre bloc de RAM.

Dans l'animation qui suit, on suppose pour simplifier que chaque variable et chaque instruction du programme en cours d'exécution était stockée sur 1 octet de mémoire.

En fait, les ordinateurs actuels sont capables de transférer plusieurs octets (jusqu'à 8 octets) dans les registres du processeur en 1 seul cycle d'horloge (les registres ont une taille de 4 ou 8 octets).

Point important

Lors du chargement d'un programme en mémoire vive (RAM), une **pile** est créée pour stocker provisoirement les informations. Cette pile est notamment utilisée pour :

- Transmettre les paramètres à une fonction lors de son appel.
- Transmettre au programme principal les valeurs retournées par une fonction.
- Contrairement aux variables, **aucun espace fixe** de la RAM n'est réservé à ces paramètres et valeurs retournées.