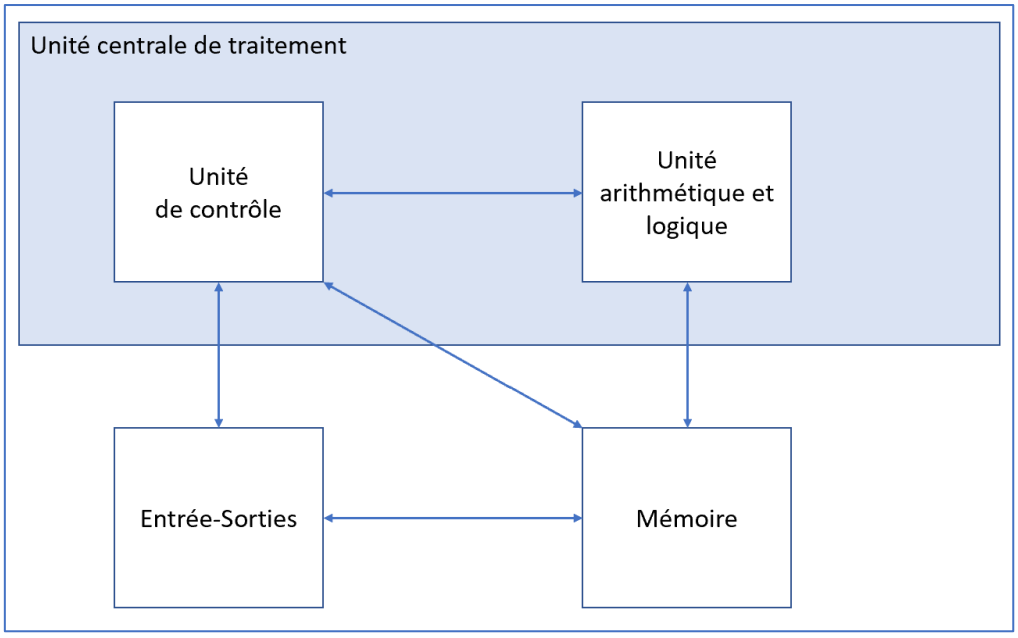
Architectures Von Neumann & Organisation de la mémoire

Eléments du programme

Contenus	Capacités attendues
Modèle d'architecture séquentielle (von Neumann)	Distinguer les rôles et les caractéristiques des différents constituants d'une machine.

I. Architectures Von Neumann

Les grands principes de fonctionnement des ordinateurs reposent sur des travaux réalisés au milieu des années 1940 par une équipe de chercheurs de l'université de Pennsylvanie.
Ces travaux concernaient la conception d'un ordinateur dans lequel les programmes à exécuter étaient stockés au même endroit que les données qu'ils devaient manipuler, à savoir dans la mémoire de l'ordinateur. Cette idée d'utiliser une zone de stockage unique pour les programmes et les données est toujours utilisé aujourd'hui. Cette architecture est appelée *modèle de von Neumann*, en l'honneur du mathématicien et physicien John von Neumann qui participa à ces travaux et qui publia en 1945 un rapport sur la conception de l'EDVAC[1], ce nouvel ordinateur basé sur ce modèle de calcul.



Architecture de von Neumann

[1] L'EDVAC (Electronic Discrete Variable Automatic Computer) est le successeur direct de l'ENIAC. Contrairement à l'ENIAC, l'EDVAC fonctionnait en binaire.

Ce schéma décrit l'organisation des principaux composants d'un ordinateur selon l'architecture de von Neumann. Ce modèle comporte quatre types de composants : ***une unité arithmétique et logique***, **une unité de contrôle**, ***la mémoire*** de l'ordinateur, et ***les périphériques d'entrée-sortie***.
Les deux premiers composants sont habituellement rassemblés dans un ensemble de circuits électroniques que l'on appelle *Unité Centrale de Traitement* ou plus simplement **processeur** (*CPU* en anglais, pour *Control Processing Unit*). Lorsqu'un processeur rassemble ces deux unités dans un seul et même circuit, on parle alors de ***microprocesseur***.

- L'unité *arithmétique et logique* (*Arithmetic Logic Unit* en anglais ou ***ALU***) est un circuit électronique qui effectue des opérations arithmétiques sur les nombres entiers et flottants, et des opérations logiques sur les *bits*.
- L'unité de *contrôle* (*Control Unit* en anglais ou ***CU***) joue le rôle de chef d'orchestre de l'ordinateur. C'est ce composant qui se charge de récupérer en mémoire la prochaine instruction à exécuter et les données sur lesquelles elle doit opérer, puis les envoie à l'unité arithmétique et logique.

Mémoire
La mémoire d'une machine permet de stocker des données. Sa capacité est exprimée en : kilooctet, mégaoctet, gigaoctet, téraoctet, pétaoctet. Conversions : 1 ko = 1000 octets, 1 Mo= 1000 ko, 1 Go = 1000 Mo, 1 To = 1000 Go et 1 Po = 1000 To.

La *mémoire* de l'ordinateur contient à la fois les programmes et les données. On distingue habituellement deux types de mémoire :

- La mémoire *vive* où *volatile* est celle qui perd son contenu dès que l'ordinateur est éteint. Les données stockées dans la mémoire vive d'un ordinateur peuvent être lues, effacées ou déplacées comme on le souhaite. Le principal avantage de cette mémoire est la *rapidité* d'accès aux données qu'elle contient, quel que soit l'emplacement mémoire de ces données. On parle souvent de mémoire *RAM* en anglais, pour *Random-access Memory*.

Mémoire vive
La mémoire vive où RAM (*Random-access Memory*) elle la mémoire dans laquelle sont enregistrées les informations traitées par la machine. Dans les ordinateurs personnels, la mémoire vive a une capacité de 4 à 64 Go. Elle s'efface quand la machine n'est plus alimentée en électricité. On dit qu'elle est *volatile*. Ordres de grandeur : ▪ capacité : jusqu'à 64 Go ▪ vitesse : jusqu'à 2 Go/s

- La mémoire *non volatile* est celle qui conserve ces données quand on coupe l'alimentation électrique de l'ordinateur. Il existe plusieurs types de telles mémoires. Par exemple **la ROM**, pour *Read-only Memory* en anglais, est une mémoire non modifiable qui contient habituellement des données nécessaires au démarrage d'un ordinateur ou tout autre

information dont l'ordinateur a besoin pour fonctionner. **La mémoire *flash*** est un autre exemple de mémoire non volatile. Contrairement à la ROM, cette mémoire est modifiable (un certain nombre de fois). Contrairement à la RAM, ces mémoires sont souvent beaucoup plus lentes soit pour lire les données, soit pour les modifier.

Mémoire morte
La mémoire morte ou ROM (de l'anglais *Read-Only Memory*) est une mémoire persistante de quelques mégaoctets, qui ne s'efface pas lorsque la machine n'est plus alimentée en électricité. Son contenu est fixé lors de la fabrication de la machine. Elle peut être lue plusieurs fois mais n'est pas prévue pour être modifiée.

Il existe un très grand nombre de périphériques d'*entrées/sorties* pour un ordinateur. On peut tenter de les classer par familles. Tout d'abord, ***les périphériques d'entrée*** :

- Les dispositifs de saisie comme les claviers ou les souris,
- Les manettes de jeu, les lecteurs de code barre,
- Les scanners, les appareils photos, des webcams, etc.

Ensuite, ***les périphériques de sortie*** comme :

- Les écrans et vidéoprojecteurs,
- Les imprimantes,
- Les hauts parleurs, etc.

Enfin certains périphériques sont à la fois des dispositifs d'entrée et de sortie comme :

- Les lecteurs de disques (CD, Blue Ray, etc.)
- Les disques durs, les clés USB ou les cartes SD,
- Les cartes réseaux (modems), etc.

Exercice 1 Dans la feuille d'exercices (Exercices Von Neumann), **compléter le schéma** des principaux composants de l'ordinateur.

Les flèches du diagramme (de l'Architecture de von Neumann) décrivent les différentes interactions entre ces composants. Pour les comprendre, nous allons passer rapidement en revue le fonctionnement de chaque composant et ses interactions avec les autres composants de l'ordinateur.

Unité de contrôle.

Cette unité est essentiellement constituée de trois sous-composants. Tout d'abord, deux ***registres*** (mémoires internes très rapides).

- Le premier est ***le registre d'instruction***, dénommé **IR** (car en anglais il se nomme *Instruction Registrer*), qui contient l'instruction courante à décoder et exécuter.
- Le second registre est ***le pointeur d'instruction***, dénommé **IP** (car en anglais il se nomme *Instruction Pointer*), qui indique l'emplacement mémoire de la prochaine instruction à exécuter.
- Le troisième sous-composant est un programme particulier, appelé ***micro-programme***, qui est exécuté par le **CU** et qui contrôle presque tous les mouvements de données de la mémoire vers l'**ALU** (et réciproquement) ou les périphériques d'entrée-sortie. L'unité de contrôle est donc tout naturellement connectée à tous les autres composants de l'ordinateur.

L'unité arithmétique et logique (UAL, en anglais Arithmetic Logic Unit, ALU).

L'unité arithmétique et logique est le cœur de l'ordinateur. Cette unité est composée de plusieurs registres dits *registre de données,* et d'un registre spécial, appelé ***accumulateur,*** dans lequel vont s'effectuer tous les calculs.

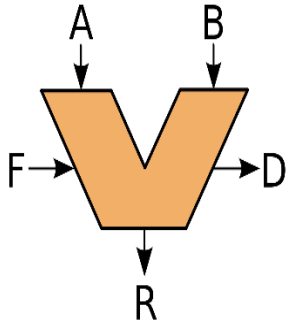
Les Registres
Un registre est un emplacement mémoire interne au processeur pour stocker des opérandes et des résultats intermédiaires lors des opérations effectuées dans l'UAL notamment. Leur capacité, leur nombre et leurs rôles varient selon les processeurs. A noter : La plupart des PC actuels ont des registres de taille 64 bits. Ils sont accessibles via le jeu d'instructions. Ordres de grandeur : ▪ capacité : quelques dizaines d'octets ▪ vitesse : jusqu'à 30 Go/s

À ces registres s'ajoutent tout un tas de circuits électroniques pour réaliser des opérations arithmétiques (addition, soustraction, etc.), des opérations logiques (ET, OU, Complément à un[2], etc…), des comparaisons (égalité, inférieur, supérieur, etc.) des opérations de sur les *bits* (décalages, rotations) ou des opérations de déplacements mémoire (copie de ou vers la mémoire). Les entrées d'une ALU sont les données sur lesquelles elle va effectuer une opération (on parle d'*opérandes*).

Ces registres sont chargés avec des valeurs venant de la mémoire de l'ordinateur et c'est l'unité de contrôle qui indique quelle opération doit être effectuée. Le résultat d'un calcul (arithmétique ou logique) se trouve dans l'accumulateur. Cependant, l'ALU peut également envoyer des signaux pour indiquer des erreurs de calcul (division par zéro, dépassement de la mémoire, etc.) ou des résultats de comparaison (inférieur, supérieur, etc.).

[2] Le complément à un d'un nombre binaire est la valeur obtenue en inversant tous les bits de ce nombre (en permutant les 0 par des 1 et inversement).

On le représente habituellement par ce schéma :



- A et B sont les opérandes (les entrées) ;
- R est le résultat ;
- F est une fonction binaire (l'opération à effectuer) ;
- D est un drapeau indiquant un résultat secondaire de la fonction (signe, erreur, division par zéro, …).

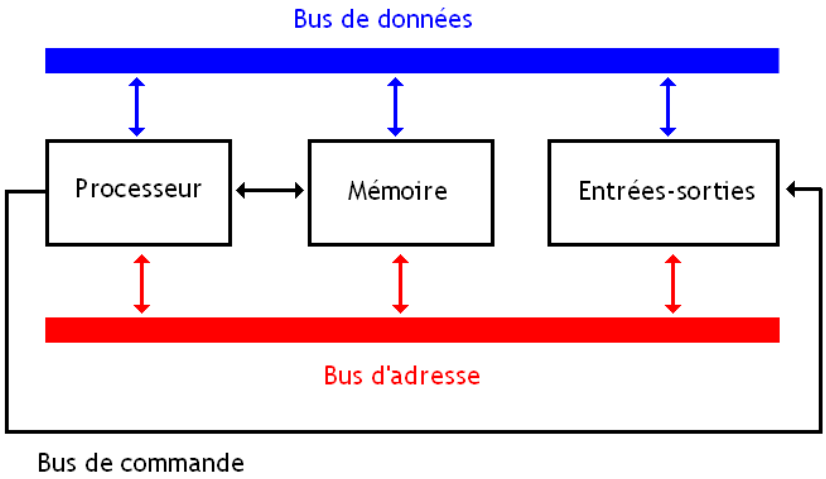
Les opérations que peuvent réaliser une UAL de base sont :

- Les additions, soustractions, multiplications et divisions ;
- Les opérations logiques (ET, OU, NON) ;
- Les opérations de comparaison (>, < et =)

La mémoire.

Nous avons vu les mouvements de données entre l'unité centrale de traitement et la mémoire de l'ordinateur. Ces échanges se font à travers un médium de communication appelé ***bus***. Mais les périphériques d'entrée-sortie peuvent également lire et écrire directement dans la mémoire à travers ce bus, sans passer par le CPU. Cet accès direct à la mémoire est réalisé par un circuit électronique spécialisé appelé contrôleur DMA, pour *Direct Memory Access* en anglais.

Les bus
Pour que les données circulent entre les différentes parties d'un ordinateur (mémoire, CPU et les entrées/sorties), il existe des systèmes de communication appelés bus. Il en existe de 3 grands types :



- **Le bus d'adresse** permet de faire circuler des adresses. *Par exemple l'adresse d'une donnée à aller chercher en mémoire.*
- **Le bus de données** permet de faire circuler des données.
- **Le bus de contrôle** permet de spécifier le type d'action. *Exemples : écriture d'une donnée en mémoire, lecture d'une donnée en mémoire.*

Les dispositifs d'entrée-sortie.

Ces composants sont connectés à l'ordinateur par des circuits électroniques appelés *ports* d'entrée-sortie sur lesquels il est possible d'envoyer ou recevoir des données. L'accès à ces ports se fait habituellement à travers des emplacements mémoires à des adresses prédéfinies. Ainsi, l'envoi ou la réception de données revient simplement à lire où écrire dans ces emplacements réservés. Pour connaître l'état d'un périphérique, le CPU peut soit *périodiquement* lire dans ces emplacements mémoires, mais il peut aussi être directement prévenu par un périphérique d'un changement à travers un mécanisme spécial *d'interruption* prévu à cet effet. Une fois interrompu, le CPU peut aller lire le contenu des ports.

Exercice 2 Construire **une carte mentale** détaillée des éléments constitutifs de l'Architecture Von Neumann.

Le rôle de l'horloge

Les programmes stockés dans la mémoire centrale d'un ordinateur sont constituées d'instructions de bas niveau, directement compréhensible par le CPU. Il s'agit des instructions du *langage machine*. Lorsqu'elles sont stockées dans la mémoire, ces instructions ne sont ni plus ni moins que de simples nombres binaires, comme les données manipulées par les programmes.

Pour progresser dans l'exécution d'un programme, l'unité de contrôle de l'ordinateur réalise de manière continue, à un rythme imposé par une *horloge* globale, la boucle suivante, appelé *cycle d'exécution d'une instruction*, qui est constituée de 3 étapes.

1. **Chargement**. A l'adresse mémoire indiquée par son registre **IP** l'unité de contrôle va récupérer le mot binaire qui contient la prochaine instruction à exécuter et le stocker dans son registre IR.
2. **Décodage**. La suite de *bits* contenue dans le registre **IR** et décodée afin de déduire quelle instruction doit être exécutée et sur quelles données. Cette étape peut nécessiter de lire d'autres mots binaires depuis la mémoire si le format de l'instruction en cours de décodage le nécessite. C'est également à cette étape que sont chargées les données (on dit aussi *opérandes*) sur lesquelles l'opération va porter (ces données pouvant être dans des registres ou en mémoire).
3. **Exécution**. L'instruction est exécutée, soit par l'ALU s'il s'agit d'une opération arithmétique où logique, soit par l'unité de contrôle s'il s'agit d'une opération de branchement qui va donc modifier la valeur du registre **IP**.

Le rôle de l'horloge

- Le CPU dispose d'une horloge qui cadence l'accomplissement des instructions. L'unité est appelée **cycle**.
- Lorsqu'on parle d'un processeur cadencé à 3 GHz, cela signifie qu'il y a 3 milliards de cycles d'horloge par seconde.
- La fréquence d'horloge caractérise (grossièrement) le nombre d'opérations qu'il peut effectuer en une seconde.

Le hertz (symbole : Hz) est l'unité de fréquence. Un hertz est la mesure de la fréquence de répétition d'un événement qui se répète une fois par seconde (1⁄s).

Multiples

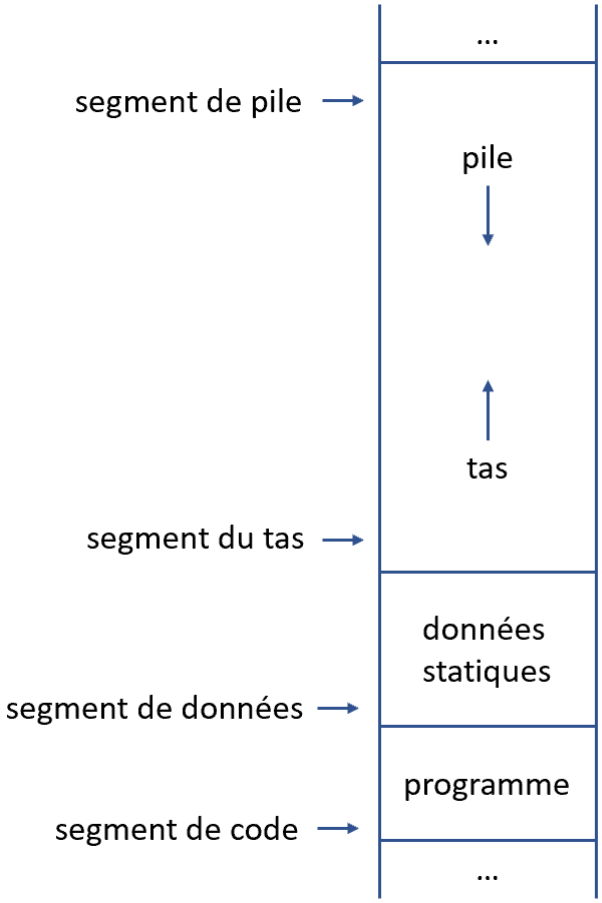
Coefficient	Nom	Symbole
10	Décahertz	daHz
10^2	Hectohertz	hHz
10^3	Kilohertz	kHz
10^6	Mégahertz	MHz
10^9	Gigahertz	GHz

Exercice 3 Répondre au questionnaire à choix multiples de la feuille d'exercices (Exercices Von Neumann).

II. Organisation de la mémoire

La mémoire d'un ordinateur (à ne pas confondre avec les périphériques de stockage comme les disques durs ou les clés USB) contient à la fois les programmes à exécuter et les données que ces programmes doivent manipuler.

La mémoire d'un ordinateur peut être vue comme un tableau de cases mémoires élémentaires, appelées ***mot mémoire***. Selon les ordinateurs, la taille de ces mots peut varier de 8 à 64 *bits*. Chaque case possède une adresse unique à laquelle on se réfère pour accéder à son contenu (en écriture ou en lecture). Traditionnellement, ce tableau mémoire est représenté verticalement comme dans la figure *ci-dessous*, avec les premières adresses de la mémoire en bas du schéma.



Organisation de la mémoire

Le tableau décrit l'organisation de l'espace mémoire d'un programme *actif*, plus généralement appelé *processus,* c'est à dire un programme en train d'être exécuté par la machine (sachant qu'il peut y avoir plusieurs programmes actifs en même temps dans la mémoire). Cet espace est découpé en quatre parties, on dit aussi ***segments de mémoire*** :

- **Le segment de code** qui contient les instructions du programme
- **Le segment de données** qui contient les données dont l'adresse en mémoire et la valeur sont connues au moment de l'initialisation de l'espace mémoire du programme. On parle alors de données *statiques,* par opposition aux données dont l'espace mémoire est alloué *dynamiquement,* c'est à dire pendant l'exécution de programme. La taille du segment de

données statiques est *fixe*, il n'est donc pas possible d'allouer de nouvelles cases mémoires dans cet espace de à l'exécution.

- **Le segment de pile**. Ce segment, comme le suivant, contient l'espace mémoire alloué *dynamiquement* par un programme. La pile est utilisée au moment de l'appel de fonctions d'un programme pour stocker les paramètres mais également les variables locales des fonctions. La gestion en pile de ce segment mémoire facilite l'exécution de fonctions récursives [3] (où chaque appel a besoin d'un espace mémoire propre pour être exécuté) mais également la libération de la mémoire au moment où la fonction se termine.

- **Le segment du tas.** Il s'agit de la zone mémoire qui contient toutes les données allouées dynamiquement par un programme. Il peut s'agir de celles dont la durée de vie n'est pas liée à l'exécution des fonctions, ou simplement celles dont le type impose qu'elles soient allouées dans cette zone mémoire, par exemple parce que leur taille peut évoluer (comme des tableaux Python).

Accès aux mots de la mémoire

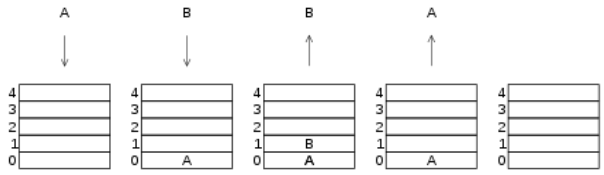
L'adresse d'une case mémoire s'obtient en additionnant l'adresse du début du segment mémoire où elle se trouve (code, donnée, tas ou pile), qu'on appelle *base*, et la position de ce mot dans le segment, qu'on appelle *déplacement.* Ce mode d'adressage *relatif* à une base permet à un programme d'être exécuté dans n'importe quelle partie de la mémoire d'un ordinateur, sans changer ses instructions d'accès en lecture où écriture.

Gestion du segment de pile

Le segment de pile permet de gérer facilement l'espace mémoire des fonctions, en empilant successivement leur contexte d'exécution (par exemple pour des fonctions récursives ou des fonctions imbriquées) et en libérant *rapidement* et *automatiquement* cet espace lorsque les fonctions se terminent.

C'est quoi une pile ?

Une pile est une structure de données fondée sur le principe « dernier arrivé, premier sorti », ce qui veut dire : le dernier élément ajouté à la pile est le premier à en sortir.



Schémas d'une pile

[3] Une fonction récursive est une fonction qui s'appelle elle-même. Cette notion sera abordée dans le programme de terminale uniquement.

Exercice 4 Gestion du segment de pile - **Compléter les textes à trous, puis compléter le schéma représentant la pile** de la feuille d'exercices (Exercices Von Neumann).

Exercice 5 **Testez par vous-même** : Utilisez Python Tutor et déroulez l'exemple de gestion du segment de pile

Gestion du tas.

La gestion de la mémoire avec une pile n'est pas toujours possible ou souhaitable. Par exemple, si les valeurs allouées dans le corps d'une fonction ont une durée de vie plus grande que l'appel de cette fonction, il n'est pas possible de les allouer dans la pile, sans quoi elles disparaîtraient au retour de la fonction. Prenons par exemple la fonction suivante **allouer** qui renvoie une paire constituée de deux tableaux construits par la fonction.

```
def allouer(x, y) :
    t = [y] * x
    u = [x] * y

    return (t, u)
```

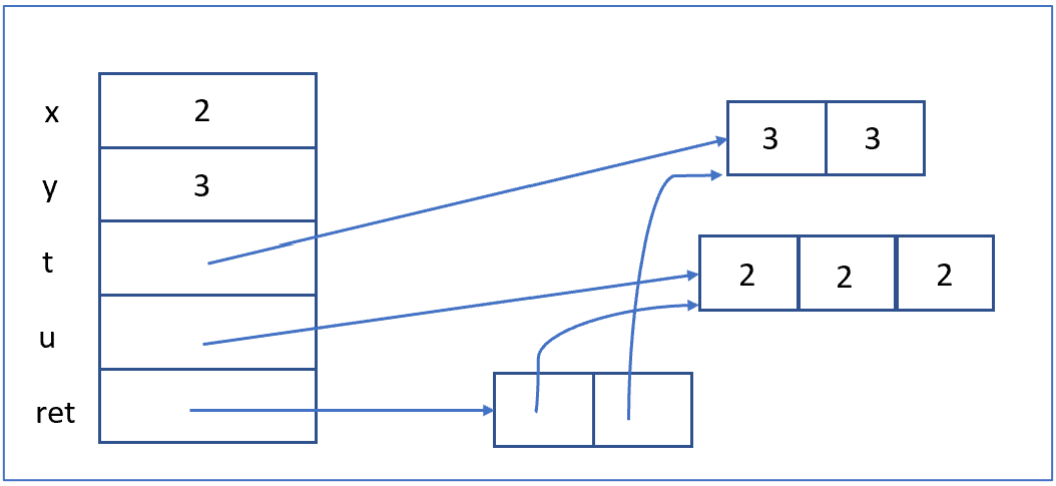
Il n'est pas possible d'allouer dans le segment de pile les tableaux t et u, ni la paire (t, u) renvoyée par la fonction, car toutes ces zones mémoires vont disparaître à la fin de l'appel.

Pour cela, ces valeurs doivent être allouées dans le segment du tas[4] qui sert à stocker des valeurs dans une zone mémoire qui n'est jamais effacée. Le tas est simplement vu comme un tableau contigu de cases mémoires. La politique de gestion de cet espace mémoire est beaucoup plus libre que la pile. Pour allouer de la mémoire dans le tas, le programmeur utilise dans certains langages une instruction *explicite* d'allocation (qui porte souvent le nom **malloc**, pour *memory allocation*) ou dans d'autres langages, comme Python, l'allocation est faite automatiquement, sans que le programmeur ne s'en rende compte.

Par exemple, la configuration de la mémoire après l'appel allouer(2,3), juste au moment où la fonction s'apprête à renvoyer sa valeur, est décrite dans le schéma ci-dessous. Les valeurs des variables x et y sont allouées dans la pile, mais les tableaux t et u sont eux alloués dans le tas. Seuls les pointeurs vers ces tableaux sont stockés dans la pile, aux emplacements réservés pour t et u. La cellule mémoire de la pile qui contient la valeur de retour (ret) est elle aussi un pointeur vers une paire dans le tas. Chaque valeur de cette paire est un pointeur vers les zones mémoires allouées pour t et u.

Ainsi **lorsque la fonction allouer termine, toutes les valeurs allouées dans le tas survivent à l'effacement des valeurs dans la pile.** La fonction renvoie alors simplement le pointeur de la paire (t, u) qui contient bien l'adresse d'une zone mémoire toujours allouée.

[4] Tas (heap en anglais) : structure de données de type arbre qui permet de retrouver directement l'élément que l'on veut traiter en priorité.



Segments de pile et de tas

En Python, la libération des zones mémoires est faite automatiquement par l'interpréteur. Il utilise pour cela un algorithme appelé *Glaneur de cellules* ou simplement GC (on dit *Garbage collector* en anglais) qui agit pendant l'exécution d'un programme. Le GC détermine quelles zones mémoires dans le tas sont inutiles et il les libère automatiquement. Ainsi, le programmeur Python n'a jamais à se soucier de la gestion mémoire du tas (allocation où libération). Tout se fait automatiquement et de manière transparente.

Exercice 6 **Testez par vous-même** : Utilisez Python Tutor et déroulez l'exemple de gestion du tas