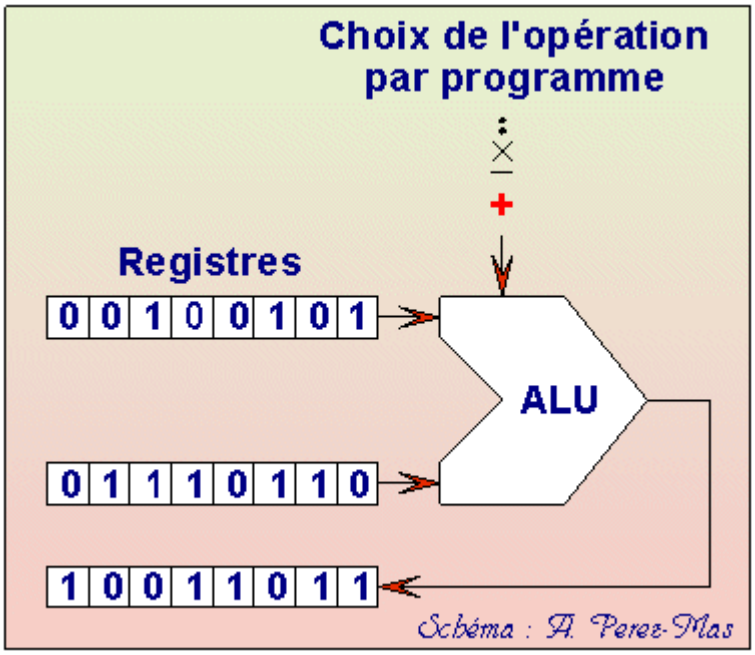
Introduction à la programmation

Rappels sur le fonctionnement d'un programme informatique

Programme informatique ?

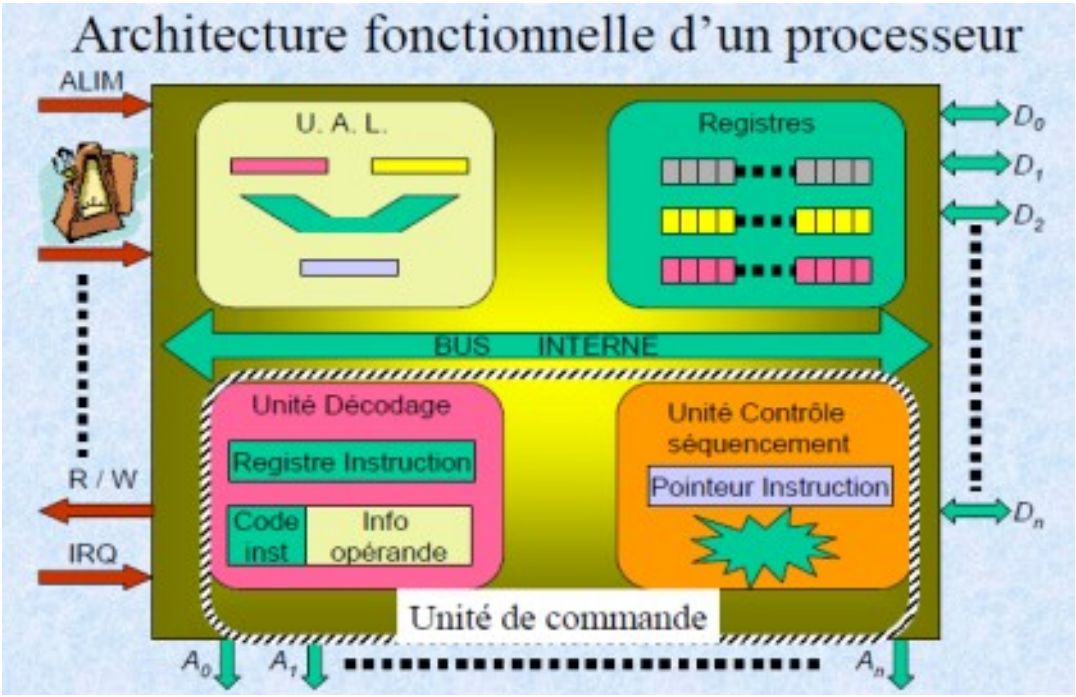
C'est un ensemble rigoureusement organisé d'ordres donnés à un ordinateur pour lui indiquer une tâche à accomplir. Plus précisément c'est le microprocesseur qui traite les ordres, que l'on appelle instructions.

Voici un exemple d'instruction élémentaire :



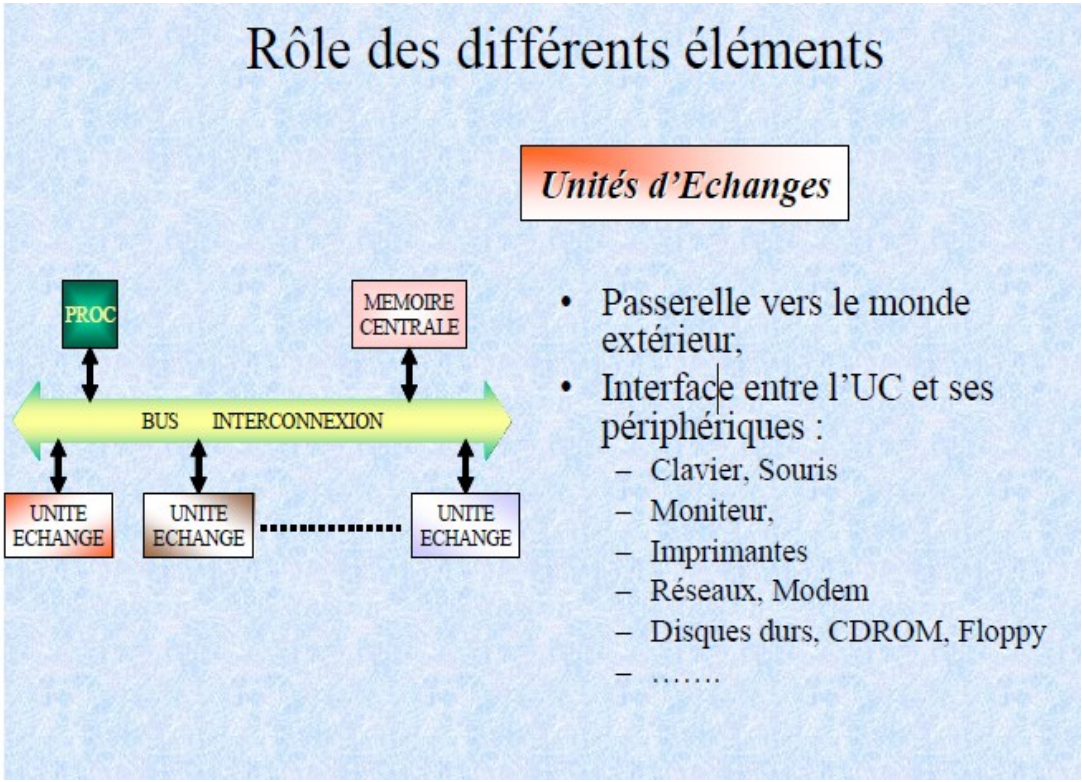
(origine du diagramme : M. Boughanem, Université Paul Sabatier, Toulouse)

Voir sur le diagramme ci-dessous l'architecture générale d'un processeur :



(origine du diagramme : M. Boughanem, Université Paul Sabatier, Toulouse)

et ci-dessous l'architecture globale d'un ordinateur :



(origine du diagramme : M. Boughanem, Université Paul Sabatier, Toulouse)

Les différentes parties de la mémoire utilisées par un programme

Le processeur exécute des instructions qui sont lues en mémoire centrale sur des données qui sont également lues en mémoire centrale.

Lors de l'exécution d'un programme, celui-ci est d'abord chargé en mémoire (par l'intermédiaire du système d'exploitation et au travers d'une “unité d'échange” permettant d'accéder au disque dur ou à une clé USB par ex.).

Ensuite le programme va placer des données en mémoire en exploitant trois types de zones :

1/ une **zone statique** qui est directement liée à l'espace mémoire occupé par le programme lui-même (pour les variables globales déjà connues au moment du chargement du programme en mémoire)

2/ une **zone dynamique appelée « tas »** (ou « heap » en anglais) pour la réservation dynamique d'espace mémoire au moment de l'exécution. Cette zone partagée par l'ensemble des programmes chargés en mémoire à un instant donné est gérée par un “gestionnaire de mémoire dynamique” lié au système d'exploitation.

3/ une **zone dynamique appelée « pile »**, propre à chaque programme s'exécutant en mémoire et où sont placées des données temporaires : les variables locales et les arguments des sous-programmes.

Remarque : pour être plus précis il faudrait distinguer dans le vocabulaire le programme qui est la suite d'instruction écrite par le programmeur, et le processus qui correspond à l'exécution d'un programme en mémoire à un instant donné. C'est au processus que sont associées les zones mémoires.

Voici une illustration de la répartition mémoire pour deux programmes (ou plus précisément deux processus) :

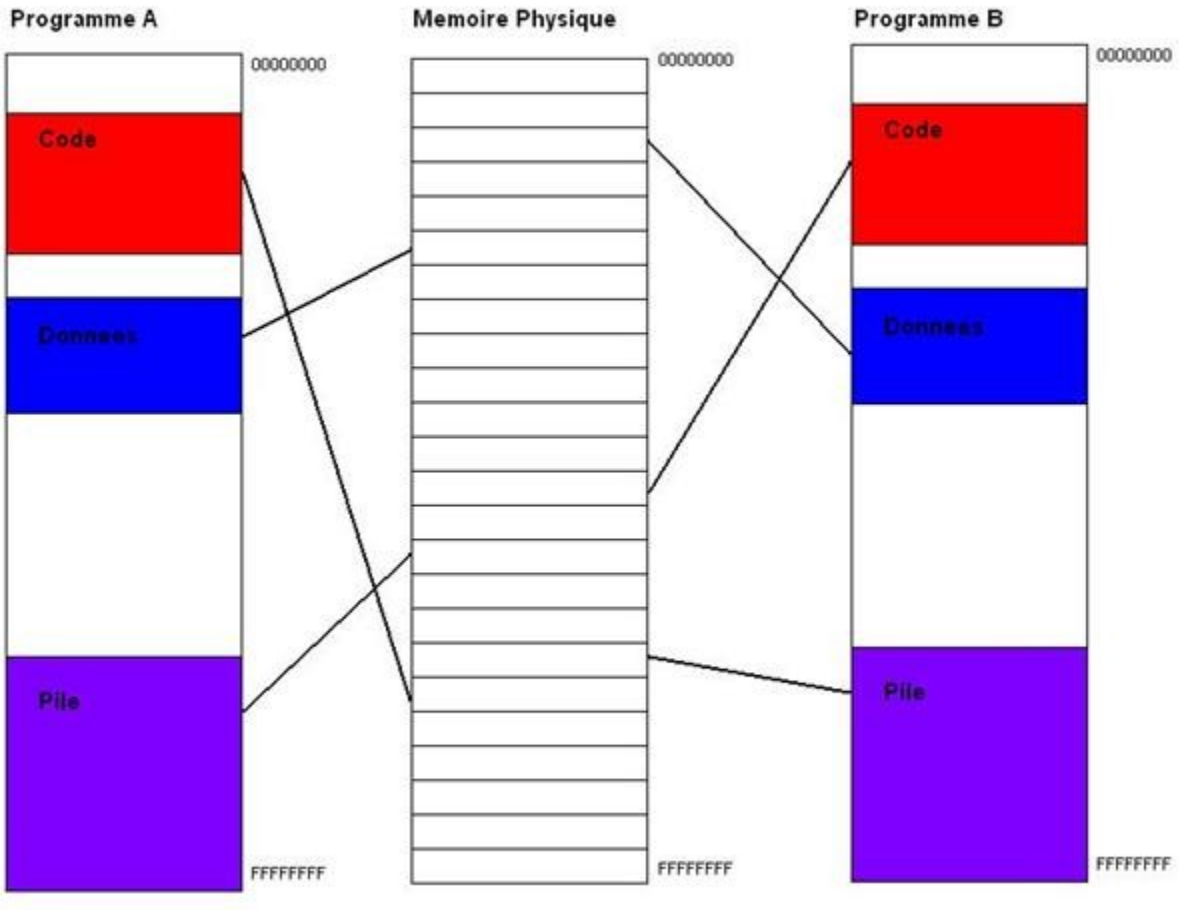


diagramme : différents types de gestion mémoire

1/ En Assembleur

Dans un programmes informatique en langage machine ou assembleur, le programmeur gère lui-même la mémoire en accédant à des adresses concrètes de la mémoire physique.

Ex :

```
start
        BL      func1                   ; Branch to first subroutine
        BL      func2                   ; Branch to second subroutine
stop
        MOV     r0, #0x18               ; angel_SWIreason_ReportException
        LDR     r1, =0x20026            ; ADP_Stopped_ApplicationExit
        SVC     #0x123456               ; ARM semihosting (formerly SWI)
func1
        LDR     r0, =42                 ; => MOV R0, #42
        LDR     r1, =0x55555555         ; => LDR R1, [PC, #offset to
                                        ; Literal Pool 1]
        LDR     r2, =0xFFFFFFFF         ; => MVN R2, #0
        BX      lr
```

```
        LTORG                           ; Literal Pool 1 contains
                                       ; literal 0x55555555
func2
        LDR      r3, =0x55555555        ; => LDR R3, [PC, #offset to
                                        ; Literal Pool 1]
        ; LDR r4, =0x66666666           ; If this is uncommented it
                                        ; fails, because Literal Pool 2
                                        ; is out of reach
```

Dans l'exemple précédent, les instructions LDR accèdent directement à des adresses en mémoire.

2/ En langage de 3ème génération comme le langage C

Le programmeur accède à la mémoire par l'intermédiaire de variables symboliques.

Chaque variable a un type précis qui doit être explicitement indiqué par le programmeur. Le type indique au compilateur (le logiciel qui transforme le programme source écrit par le programmeur en langage machine exécutable par le processeur) comment il devra traiter la donnée correspondante (comme un nombre, comme un caractère textuel,etc)

ex :

int i; // déclaration d'un integer (par exemple 32 bits)

long val; // déclaration d'un entier long (par exemple 64 bits)

char maChaine[5] ; //déclaration d'un tableau de 5 caractères et réservation de
//l'espace mémoire (statique) correspondant

int * pt; // déclaration d'un pointeur représentant une adresse en mémoire

pt = malloc (sizeof(int)*1000) // réservation de place mémoire pour 1000 entiers

la ligne ci-dessus correspond à une instruction (les autres étaient des déclarations) une zone de 1000 fois la taille d'un entier est réservée, par le gestionnaire de mémoire dynamique du système, au moment où le processeur rencontre cette instruction (réservation dynamique).

pt[0] = 125 ; // on place la valeur entière 125 au début de la zone précédente
pt[4] = i; // la valeur de i est placée en cinquième postion dans la zone précédente

free(pt); // quand le programmeur a fini d'utiliser la zone de mémoire dynamique, il
// doit ordonner au gestionnaire de mémoire dynamique de libérer l'espace.

3/ Avec les langages dynamiques ou langages de scripts (Php, Python, …)

Ce sont des langages interprétés (les langages de scripts sont aujourd'hui pré-compilés en “bytecode” lors de leur première exécution mais ils restent des langages interprétés car le bytecode n'est pas du langage machine, c'est un langage intermédiaire).

Dans le cas des langages interprétés, il n'y a pas de phase de traduction préalable du programme en code machine. L'interpréteur du langage est un logiciel qui lit un programme écrit dans ce langage et traduit les instructions « à la volée », une à une, et les exécute directement.

Les langages dynamiques modernes sont très évolués. En particulier le programmeur est dispensé complètement de la gestion de la mémoire. Toute réservation de mémoire est effectuée dynamiquement par l'interpréteur au moment où une nouvelle variable est utilisée.
Le programmeur ne déclare pas non plus les types des variables. C'est l'affectation des variables qui détermine leur type.

De plus, ces langages bénéficient de types tableaux (ou collections) très évolués et simples à manipuler.

La contrepartie de ces facilités offertes au programmeur est une moindre rapidité d'exécution par rapport aux langages compilés comme le C. Cela est compensé en partie par la toujours plus grande rapidité des processeurs.

Exemple en Python

```
maChaine = "une quelconque string";
maListe = ["chaine de caracteres 1", "autre string", "une troisième chaine"]
maListe.append(maChaine)
nbElem = len(maListe)
print("Ma liste comporte %d éléments\n" % nbElem)
for elem in maListe :
        print(elem+"\n")
```

On remarquera la facilité de créer un tableau (une liste) de chaines de caractères en l'initialisant avec trois chaines puis ensuite d'y rajouter une nouvelle chaine sans aucune réservation mémoire préalable par l'instruction :
maListe.append(maChaine)

C'est l'interpréteur Python qui se charge de tout !
Une règle cependant : on ne peut pas utiliser de variable avant de l'avoir affectée.