CHAPITRE 2

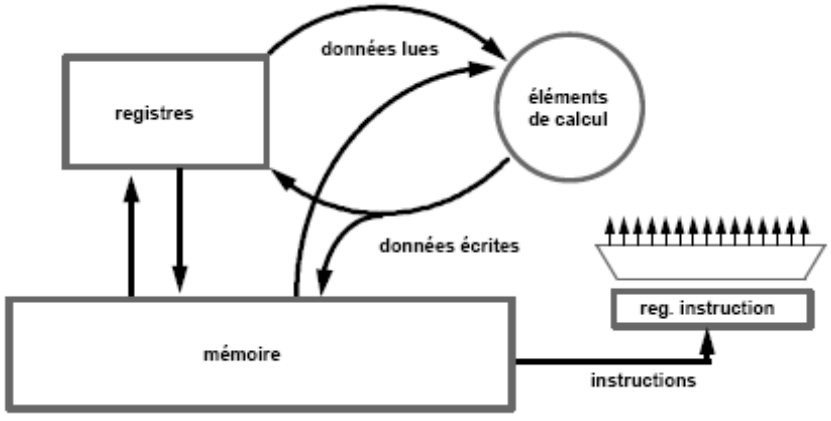
MECANISME D'EXECUTION D'UN PROGRAMME

1. Les instructions :

En informatique, une instruction machine est une opération élémentaire qu'un programme demande à un processeur d'effectuer. C'est l'ordre le plus basique que peut comprendre un ordinateur.

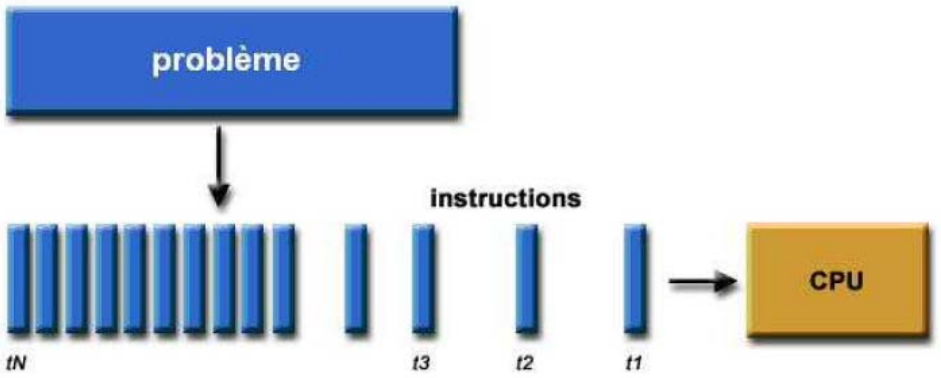
La collection d'instructions machine qui peuvent être données à un processeur est son **jeu d'instructions**. Les instructions machine sont codées en binaire. Un champ de l'instruction appelé « **code opération** » ou « opcode » désigne l'opération à effectuer. Puisque sa valeur numérique n'a pas de sens pour les humains, le programmeur utilise une abréviation désignant le code opération fourni par le langage assembleur pour ce processeur.

Depuis Von Neumann les instructions sont rangées en mémoire centrale. Ce sont des mots mémoire dont le profil décrit les opérations à effectuer dans un ou plusieurs **cycles de la machine**. La **succession des instructions** qui décrivent un calcul complexe constitue un **programme**. Il faut donc un mécanisme particulier pour les lire les unes à la suite des autres et pour les décoder.



Les principaux flux dans un ordinateur.

Traditionnellement, le logiciel a été écrit pour le calcul périodique. Il est couru sur un ordinateur simple ayant une unité centrale de traitement simple. Un problème est cassé dans une série discrète d'instructions. Des instructions sont exécutées l'un après l'autre et seulement une instruction peut s'exécuter dans un moment.



Le calcul périodique.

2. Taille des instructions :

La taille d'une instruction dépend de l'architecture de la plateforme, mais elle est usuellement comprise entre 4 et 64 bits.

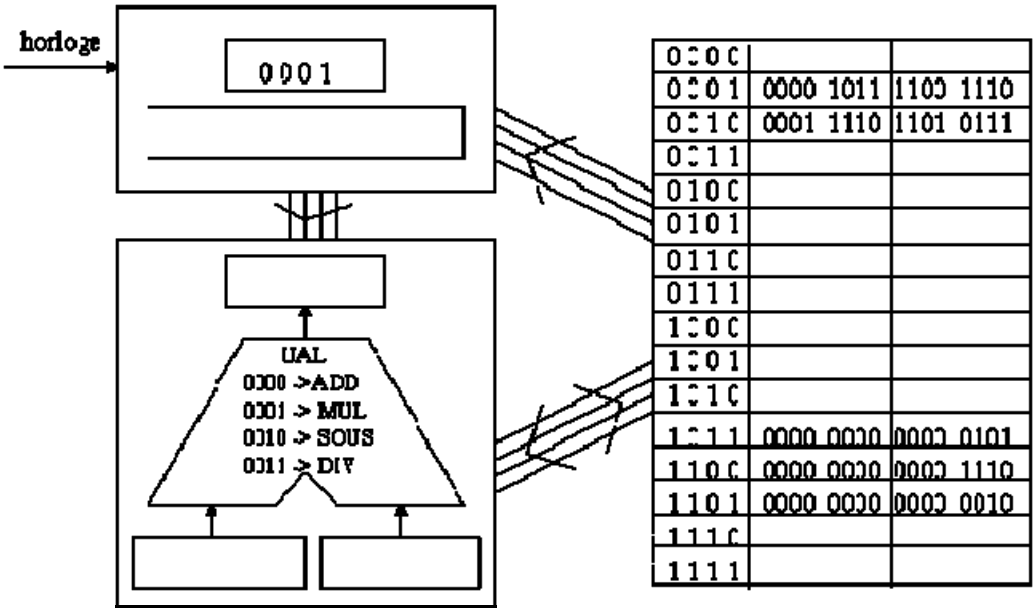
3. Le cycle d'exécution d'une instruction :

Supposons maintenant que la mémoire centrale de notre ordinateur contienne un programme et des données, et que l'on souhaite exécuter ce programme sur ces données. Lancer cette exécution revient à mettre dans le compteur ordinal (CO) l'adresse où se trouve stockée la première instruction du programme. A partir de là, le programme est exécuté étape par étape, instruction par instruction. L'exécution d'une instruction élémentaire fait suivant un cycle comprenant 3 phases :

- **Phase 1 :** L'instruction courante, dont l'adresse est stockée dans le CO, est recopiée dans le registre d'instruction (RI) en transitant par le bus «instructions» ;
- **Phase 2 :** cette instruction courante est décodée à destination de l'UAL ; ainsi le bus « ordres » transfère le code de l'opération (les 4 premiers bits) et le bus « données/résultats » transfère dans les registres appelés « donnée 1 » et « donnée 2 » le contenu des mots mémoire se trouvant aux adresses référencées dans l'instruction ;
- **Phase 3 :** l'UAL exécute l'opération qui lui est demandée en mettant à jour son registre « résultat » et transfère ce résultat dans la mémoire centrale, à l'adresse référencée dans l'instruction, en utilisant le bus « données/résultats »; par ailleurs le CO est automatiquement incrémentée (c'est-à-dire qu'il est augmenté de 1), pour signifier que l'instruction suivante à exécuter doit se trouver normalement à l'adresse qui suit immédiatement la précédente. Un nouveau cycle peut commencer alors pour la nouvelle instruction courante.

Ces cycles sont rythmés par les tops d'**horloge**, chaque phase correspondant à un nombre fixe de « tops » successifs. Dans notre exemple, pour la phase 1, qui nécessite de faire transiter l'instruction courante de la mémoire vers le RI en utilisant le bus d'instruction, 4 tops d'horloge seront nécessaires (car un mot mémoire fait 16 bits et le bus n'a une capacité que de 4 bits).

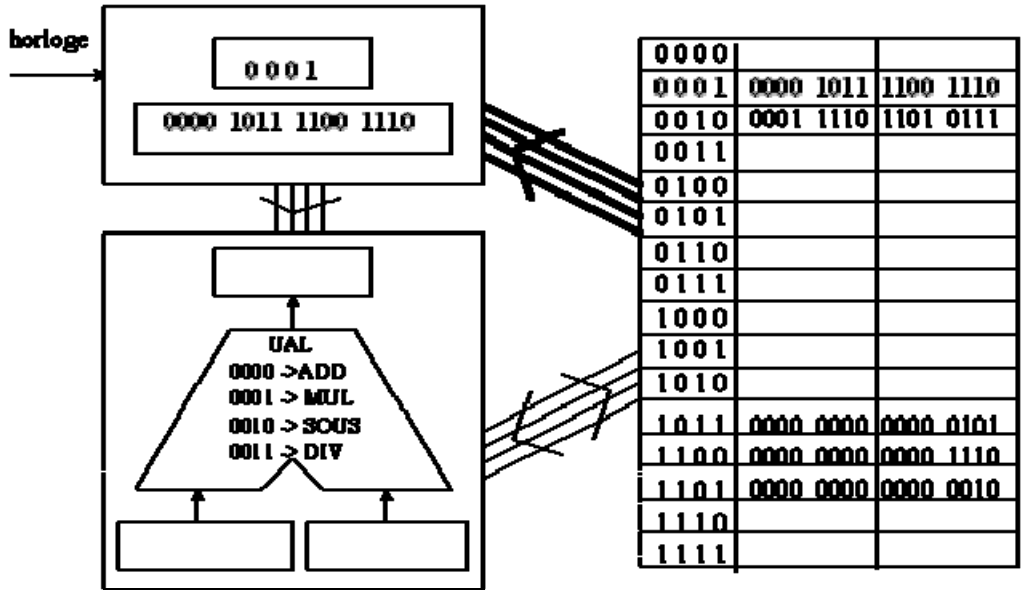
Illustrons ce fonctionnement à l'aide d'un exemple complet sur notre ordinateur miniature (seuls les bits « utiles » de la mémoire et des registres sont donnés). La figure suivante montre la situation de départ, les trois suivantes montrent l'état de l'ordinateur après l'exécution de chaque phase d'un cycle.



Situation de départ.

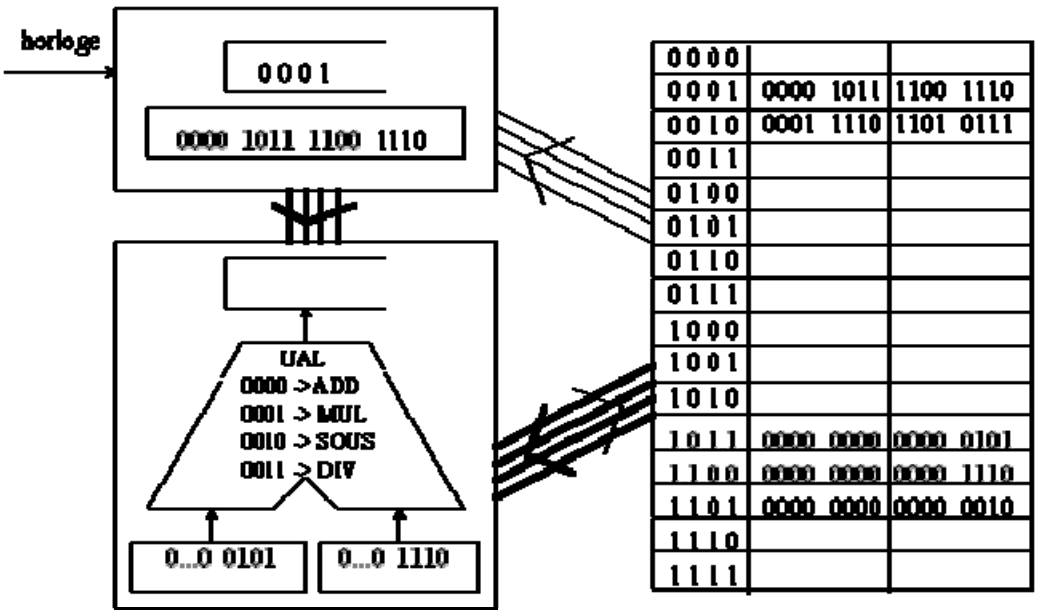
Pour aider à la compréhension, on fait à chaque étape figurer en gras les composants actifs à cette étape.

La figure suivante montre l'état de l'ordinateur à l'issue de la première phase du premier cycle. L'instruction courante a transité à travers le bus « instructions » et se retrouve désormais dans le registre d'instruction. Notons que, si le bus « instructions » est composé de 4 fils, comme sur notre schéma, alors qu'une instruction dans un mot mémoire est stockée sur 2 octets, alors ce transfert a dû se faire, au mieux, en 4 passages successifs (correspondant chacun à un top d'horloge).



Premier cycle, phase 1.

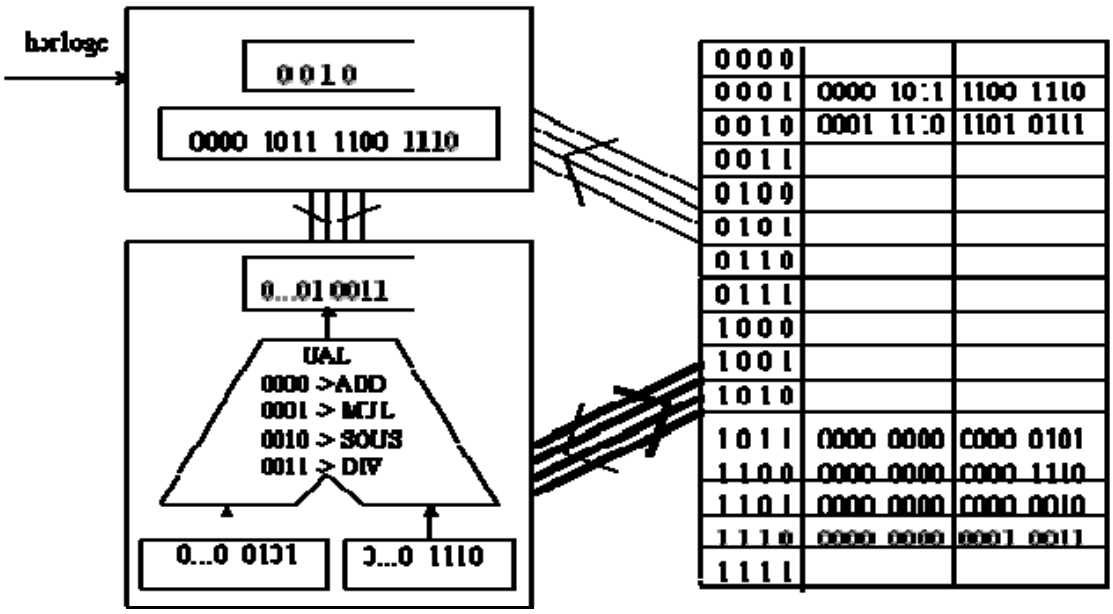
La figure suivante montre l'état de l'ordinateur à l'issue de la deuxième phase du premier cycle. Le bus « ordres » a fait tout d'abord transiter les 4 premiers bits de l'instruction courante à l'UAL, qui a reconnu que c'était le code d'une addition (l'opération active dans l'UAL est donc l'addition). Les deux suites de 4 bits suivantes correspondent aux adresses en mémoire où l'UAL doit aller chercher les données sur lesquelles effectuer cette opération. Ces données viennent remplir les registres « donnée 1 » et « donnée 2 » de l'UAL en passant par le bus « données/résultats ».



Premier cycle, phase 2.

La figure suivante montre l'état de l'ordinateur à l'issue de la troisième et dernière phase du premier cycle. L'UAL a réalisé l'opération qui lui était demandée et a rempli avec le résultat

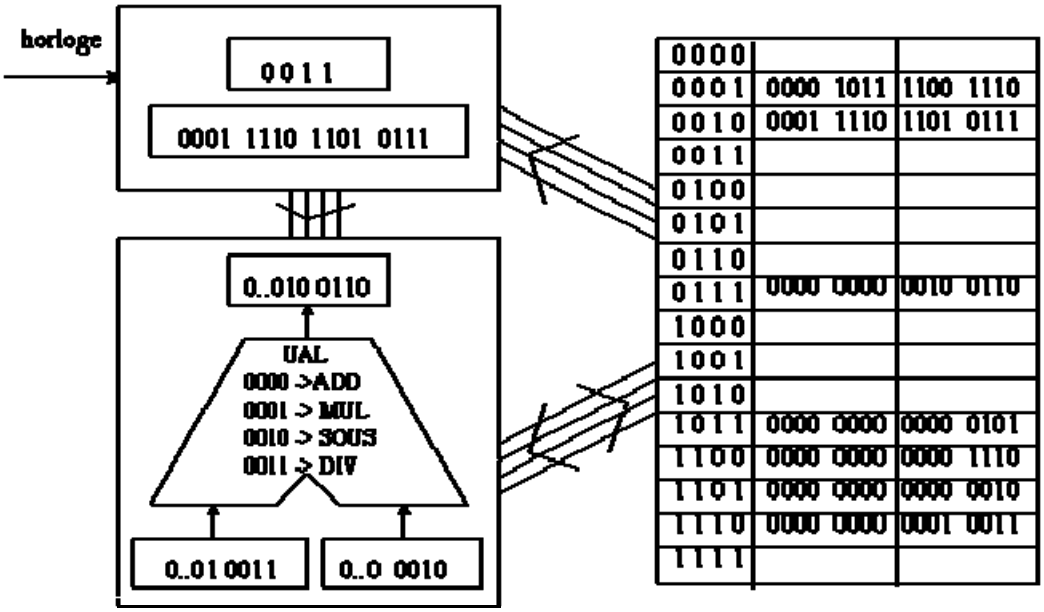
son registre « résultat ». La dernière partie de l'instruction courante indique l'adresse du mot mémoire où ce résultat doit être stocké. C'est ce qui est fait en utilisant de nouveau le bus « données/résultats ». Par ailleurs, pour préparer le cycle suivant, le compteur ordinal est augmenté de 1.



Premier cycle, phase 3.

A l'issue de ce cycle, la première instruction élémentaire a été entièrement exécutée et la machine est prête à démarrer un nouveau cycle pour exécuter la deuxième (et dernière) instruction du programme.

La figure suivante montre la situation de l'ordinateur à la fin de l'exécution de ce deuxième cycle (sans détailler les 3 phases, cette fois). Un nouveau cycle peut commencer, mais nous supposons qu'il n'y a plus d'information pertinente à l'adresse référencée par le CO (ou il y en a une signifiant « FIN ») : l'exécution du programme est donc terminée.



Situation à l'issue du deuxième cycle.

Mais quel programme, finalement, notre ordinateur a-t-il effectué, sur quelles données, et pour trouver quel résultat ? Reprenons le fil de cet exemple.

La première instruction commandait l'exécution d'une addition entre 2 nombres stockés en mémoire que nous appellerons respectivement « nombre 1 » et « nombre 2 » ; le résultat étant « nombre 3 » ; la deuxième instruction demandait de multiplier « nombre 3 » (donc, le résultat intermédiaire précédent) par un nouveau nombre, disons « nombre 4 », pour donner le résultat final.

L'algorithme réalisait donc l'opération composée suivante :

nombre 1 + nombre 2 = nombre 3

*nombre 3 * nombre 4 = résultat*

Ou encore :

*résultat = (nombre 1 + nombre 2) * nombre 4*

La valeur réelle des nombres sur lesquels le calcul a été effectué dépend de ce qui est stocké dans la mémoire aux adresses utilisées par le programme. Dans notre exemple, elles valaient respectivement (en décimal) : x=5, y=14 et z=2. La valeur du résultat final de notre exécution est donc : (5 + 14) * 2 = 38.

Sur la figure « situation de départ », les deux mots mémoire présents aux adresses 0001 et 0010 respectivement contenaient des instructions, tandis que les mots mémoire dont les adresses étaient 1011, 1100 et 1101 contenaient des données. L'exemple a aussi montré comment l'enchaînement de plusieurs opérations élémentaires pouvait permettre de réaliser une opération plus complexe.

4. Les types des Instructions :

On peut classer les instructions qu'un processeur est capable d'effectuer en quelques groupes :

a. Instructions de transfert :

Un processeur passe une grande partie de son temps à transférer des octets d'un endroit à l'autre du système : d'un périphérique vers un registre interne ou vice-versa, d'un registre interne vers la mémoire RAM ou vice-versa. Parfois, on ne peut pas effectuer de transfert direct d'une case mémoire vers une autre ou vers un périphérique, ou une écriture en mémoire ROM : il faut dans ce cas faire transiter les informations par l'un des registres internes. Remarquons que, sauf exception, il s'agit plutôt d'une copie que d'un transfert puisque la case mémoire d'origine garde son information (tant qu'on n'a pas écrit autre chose à la place).

b. Instructions arithmétiques :

Les processeurs les plus simples ne permettent que d'effectuer des additions et des soustractions, des multiplications et des divisions sur des nombres entiers de la taille d'un mot. C'est notamment le cas des processeurs. Cependant, les processeurs modernes disposent généralement d'une unité de calcul en virgule flottante capable d'effectuer des calculs sur les nombres à virgule. En l'absence d'une telle unité, les nombres à virgulent doivent être traitées en logiciel.

De même, certains anciens processeurs étaient capable d'effectuer des opérations mathématiques complexes telles que le traitement des grands nombres, des nombres fractionnaires, des puissances, des racines carrées, des fonctions trigonométriques, logarithmiques et exponentielles. Sur les processeurs modernes, ces opérations sont généralement réalisées en logiciel à l'aide des opérations mathématiques de base.

c. Instructions logiques :

Les processeurs sont capables d'effectuer des opérations logiques : ET, OU, XOU (XOR), NON (inverseur), rotations, décalages. Les opérations sont opérées simultanément sur les bits correspondant des deux registres.

La comparaison des octets A et B, qui est considérée comme une opération logique, est réalisée comme une soustraction dont on néglige le résultat ; on s'intéresse simplement au fait de savoir s'il est nul (ce qui signifie que A = B), positif (A > B) ou négatif (A < B). Ces

indications sont inscrites dans des indicateurs d'états (petites mémoires d'un bit situées dans le processeur).

d. Instructions d'entrées/sorties :

Ces instructions permettent de s'interfacer avec des dispositifs extérieurs, via des ports d'entrée/sortie. Dans certaines architectures, les ports sont considérés simplement comme des cases de mémoire et ils sont gérés par les instructions de transfert (entrées/sorties intégrées mémoire). D'autres architectures disposent d'instructions spécifiques pour les entrées/sorties (entrées/sorties indépendantes).

e. Instructions de branchement :

Il s'agit d'instructions qui altèrent le déroulement normal du programme. On distingue les sauts et les sous-routines :

- Les **sauts** provoquent un branchement du programme vers une adresse mémoire qui n'est pas contiguë à l'endroit où l'on se trouve ;
- Une **sous-routine** ou un **sous-programme** est une partie de programme dont on a besoin à plusieurs endroits dans l'exécution du programme principal. Plutôt que de répéter ce sous-programme à tous les endroits où l'on en a besoin, on le place en un endroit donné (par exemple à la fin du programme principal) et on opère un branchement du programme principal vers le sous-programme chaque fois que nécessaire. La grande différence par rapport au saut, c'est qu'au moment du branchement il faut mémoriser l'adresse d'où l'on vient, afin de pouvoir y revenir une fois le sous-programme terminé. Ceci est effectué en mémorisant l'adresse de départ dans un registre ad hoc (la pile) du processeur.

5. Ordre d'exécution des instructions :

Sauf instruction contraire, l'ordre d'exécution des instructions est celui de la lecture du code source : de la gauche vers la droite, de haut en bas.

Exemple :

```
A ← 5;
B ← A + 2;
A ← 6;
B ← A + 4;
```

L'ordre d'exécution des instructions de cet exemple est le suivant :

1. $A \leftarrow 5$; La variable A contient la valeur 5.
2. $B \leftarrow A + 2$; La variable B contient la valeur 7 (5+2).
3. $A \leftarrow 6$; La variable A contient maintenant la valeur 6.
4. $B \leftarrow A + 4$; La variable B contient maintenant la valeur 10 (6+4).

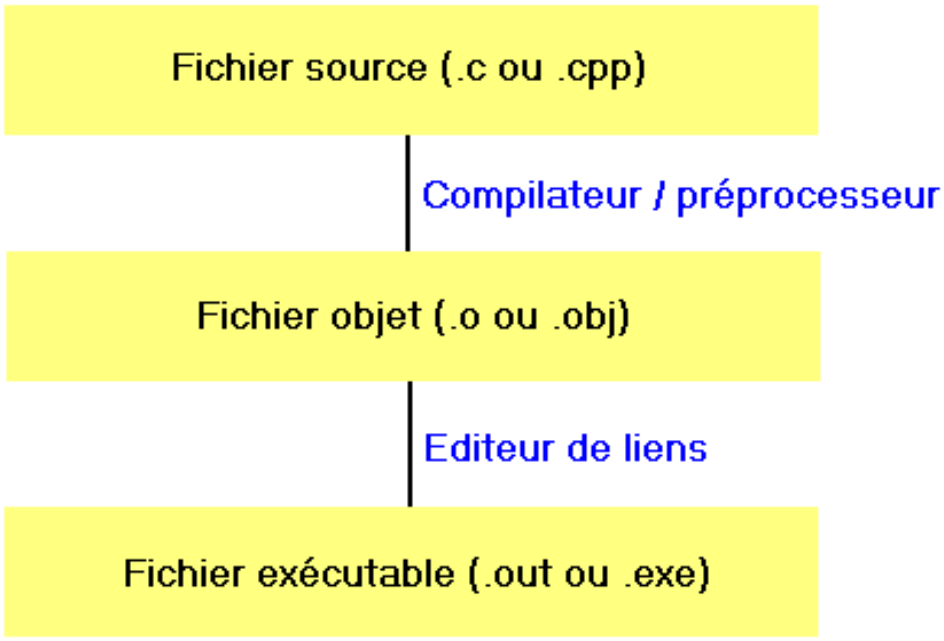
Les instructions qui utilisent un ordre d'exécution différent sont des instructions de contrôle d'exécution.

6. Phase d'élaboration d'un programme :

a. Phases de la compilation :

La compilation passe par différentes phases, produisant ou non des fichiers intermédiaires :

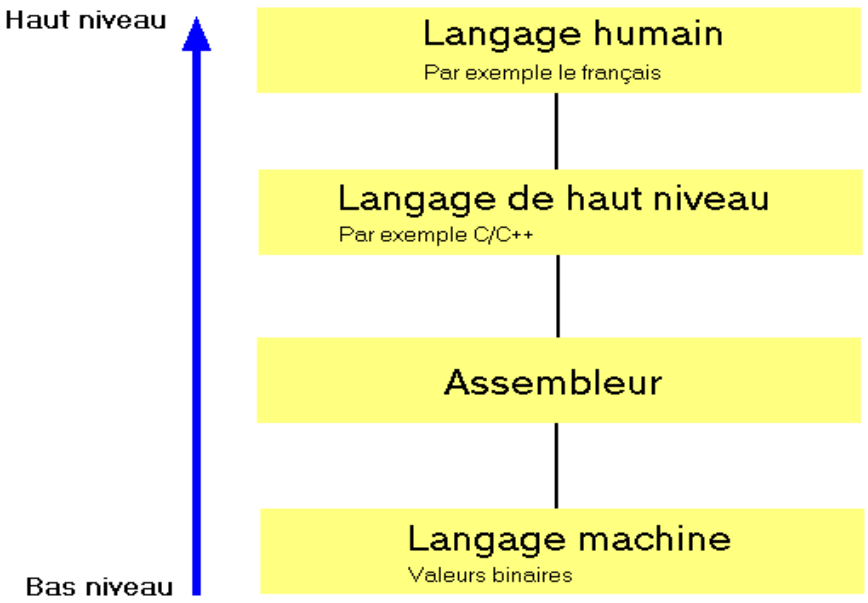
1. **Préprocessing :** Le **code source** original est transformé en code source brut. Les commentaires sont enlevés et les directives de compilation commençant par # sont d'abord traités pour obtenir le code source brut.
2. **Compilation en fichier objet :** les fichiers de **code source** brut sont transformés en un fichier dit **objet**, c'est-à-dire un fichier contenant du **code machine** ainsi que toutes les informations nécessaires pour l'étape suivante (édition des liens). Généralement, ces fichiers portent l'extension **.obj** ou **.o**.
3. **Edition de liens :** dans cette phase, l'éditeur de liens (linker) s'occupe d'assembler les fichiers objet en une entité exécutable.



Processus de compilation et d'édition de liens

b. Niveaux des langages :

Nous parlons de langage de bas niveau lorsque celui-ci est composé de valeurs binaires (langage machine) ou en est proche. Au contraire, un langage de haut niveau est compréhensible pour nous et peut être travaillé. Dans le schéma ci-dessous, vous pouvez observer la « hiérarchie » des niveaux de langages.



Niveaux des langages