Comment fonctionne un ordinateur ?

Une des questions que je me suis posée à propos des ordinateurs, c'est tout simplement : comment ça marche ?

Comment des composants matériels reliés par des fils et des prises arrivent à afficher une vidéo ou permettent à des gens de communiquer d'un bout à l'autre de la Terre ?
Bref, comment un objet peut faire tout ça ?

Je vais essayer d'expliquer dans cette page, en partant de zéro comment fonctionne un ordinateur. J'expliquerai ce qu'est un ordinateur, le fonctionnement de ses composants de base puis je remonterai progressivement au langage binaire, puis l'assembleur et les langages de programmation de haut niveau, et enfin jusqu'à l'utilisateur qui clic sur la souris.

Certaines explications sont volontairement simplifiées afin de ne pas vous embrouiller dans les détails.

Note : depuis l'écriture de ce tutoriel, j'ai également rédigé une série de 3 articles sur les semi-conducteurs, les transistors, et la mémoire informatique. Ces articles peuvent compléter le présent document et vous pourriez préférer les lire avant :

1. C'est quoi, un semi-conducteur ?
2. Comment fonctionne un transistor ?
3. Comment fonctionne la mémoire flash d'un lecteur SSD ?

Sommaire

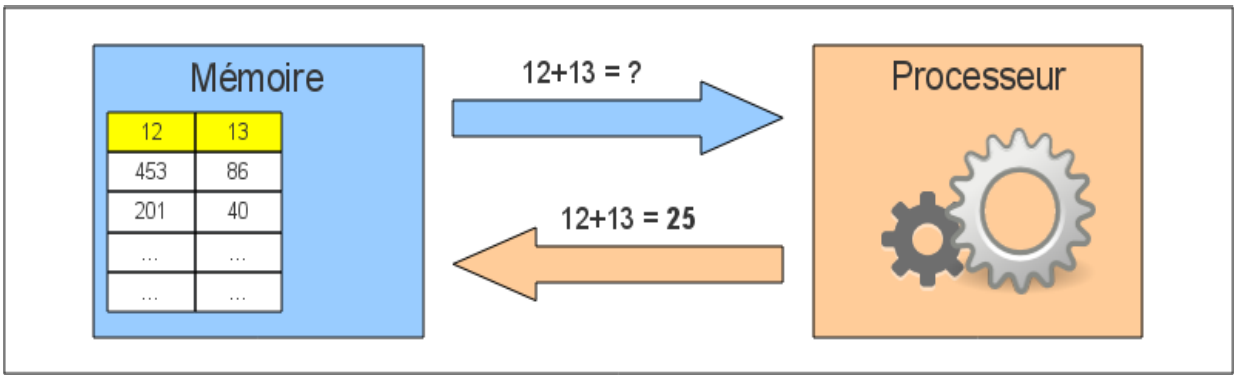
1 - L'ordinateur minimal

1.1 - La mémoire et le processeur

Un ordinateur comme le votre ou le mien est relativement complexe. Il est fait de plusieurs composants (carte réseau, disque dur, mémoire, moniteur…) mais en fait, un

ordinateur minimal n'a besoin que de deux choses pour calculer : de **la mémoire** et **d'un processeur**.

La mémoire contient des nombres et le processeur va effectuer des calculs sur ces nombres. Le processeur est donc le « cerveau » de l'ordinateur.



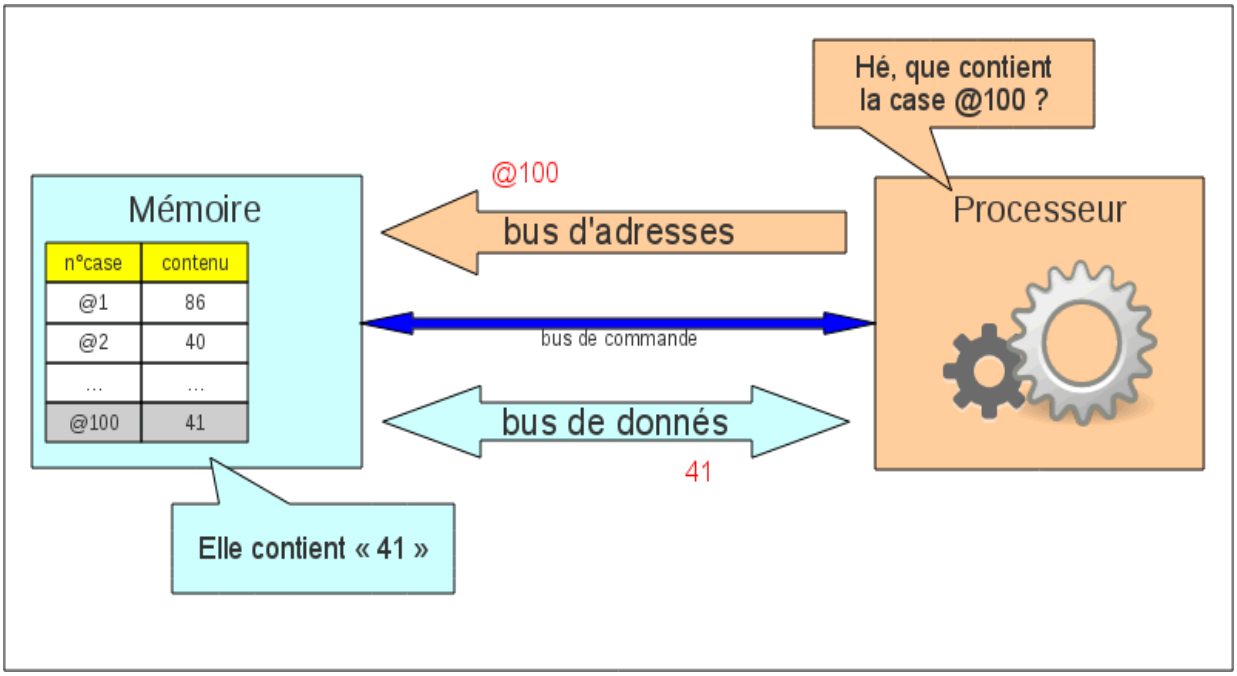
Comme vous voyez, les données se déplacent : ils transitent de la mémoire vers le processeur et retournent ensuite dans la mémoire.
Le chemin emprunté par les données est appelé le **bus**. Si vous ouvrez votre ordinateur, vous pouvez voir des nappes de câbles : ce sont un type de bus.

1.2 - Les bus de connexion

Parlons de **la mémoire** : on peut voir la mémoire comme un tableau de deux colonnes. La première contient **les données** (les nombres) et second contient **le numéro de la case**, on parle aussi de l'adresse.
L'adresse permet de repérer une case : par exemple, si on demande à la mémoire « *Hé, que contient la case n°100 ?* » il nous répondra son contenu : « *Il contient le nombre 41 !* ».
Vous l'aurez compris : c'est le processeur qui va demander le contenu des cases mémoire et ce dernier va les lui donner.

Nous avons parlé **du bus**. On peut les voir comme des fils ou des câbles.
Il y en a de plusieurs types :

- Le **bus de données** : il transporte les valeurs, les données.
- Le **bus d'adresse** : c'est ici que passe le numéro de la case dont on veux le contenu.
- Le **bus de commande** : ce bus sert à synchroniser l'ensemble afin que tout se passe dans le bon ordre.



Pour le moment, il faut retenir :

- Que le **processeur** effectue des calculs sur des nombres ;
- Que ces nombres sont stockés dans la **mémoire** ;
- Que les nombres transitent entre le processeur et la mémoire en empruntant un chemin appelé **bus**.

Il est temps de voir comment techniquement il est possible de faire les calculs : comment on peut faire 1+1=2 avec des composants électriques.

Sommaire

2 - Les transistors et les portes logiques

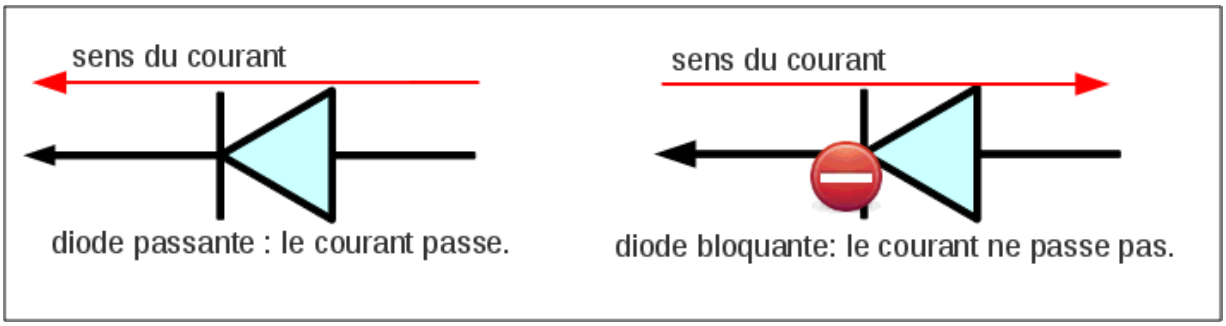
2.1 - Les transistors

Ce sont des composants électroniques en **silicium**. Le silicium est un métal dit semi-conducteur. Ça signifie qu'il n'est pas un isolant comme le plastique mais pas tout à fait

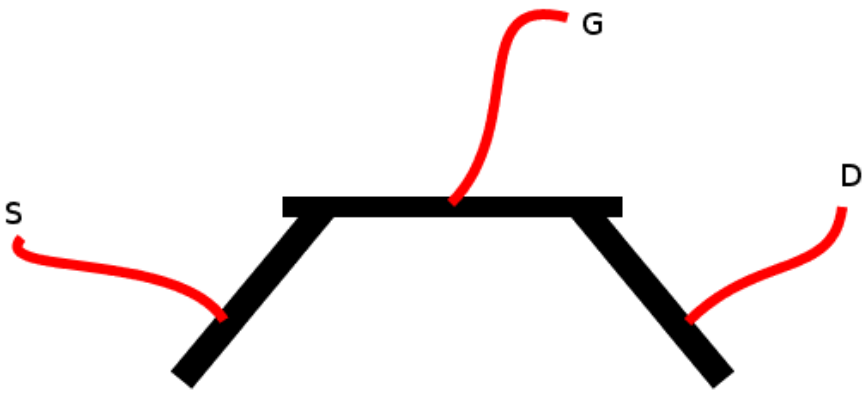
un conducteur comme le cuivre.
Il est possible de choisir, en imposant un état à ses électrons, s'il doit ou non conduire l'électricité.

Si **le silicium est le métal de base** des transistors et des puces électroniques, c'est à cause de cette propriété : la semi-conductivité.

Le transistor en silicium est comme une diode : il laisse passer le courant électrique dans un cas, mais pas dans d'autres.
Pour la diode, c'est en fonction du sens du courant :



Pour les transistors, le courant en sortie du composant dépend des différents courants et tensions d'entrées : si on applique une tension sur sa borne G (gate), le courant circulera de la borne S (source) à la borne D (drain).



C'est peut-être inutile de savoir ceci, mais si on veut vraiment savoir comment marche un ordinateur, j'estime qu'il faille commencer par là.

D'ailleurs, voici une petite vidéo (en Anglais) sur le fonctionnement d'un transistor et

comment il fait pour laisser ou ne pas laisser passer le courant : http://www.youtube.com/watch?v=ZaBLiciesOU et un autre ici : http://www.youtube.com/watch?v=IcrBqCFLHIY.

2.2 - Les portes logiques

2.2.1 - État logique

Dans ce qui va suivre, nous raisonnerons avec des états logiques.

Un ordinateur est une machine qui ne peut que comparer des hautes tensions et des basses tensions électriques. D'où la modélisation en 0 et 1 : si nous avons une tension de 5 Volt cela correspondra à l'état logique 1, et si nous avons 0 Volt, l'état logique sera 0

Tension en Volt	État logique
0 V	0
5 V	1

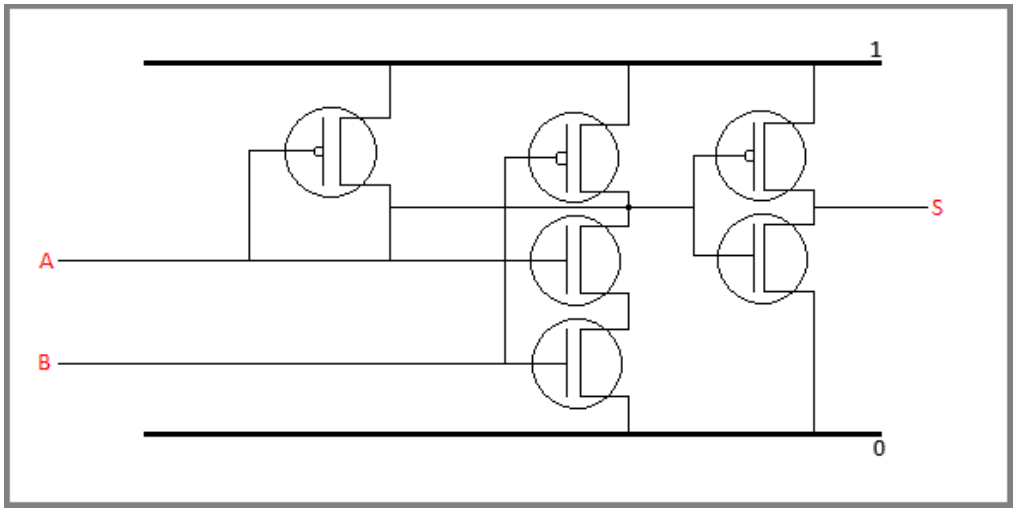
C'est là l'origine du fait que l'ordinateur « compte en binaire » : il compare des 0 et des 1.

2.2.2 - Les portes logiques

En partant de ce fait que le courant ne passe que suivant certaines conditions, et en assemblant plusieurs transistors entre eux, on obtient des composants électroniques plus évolués.

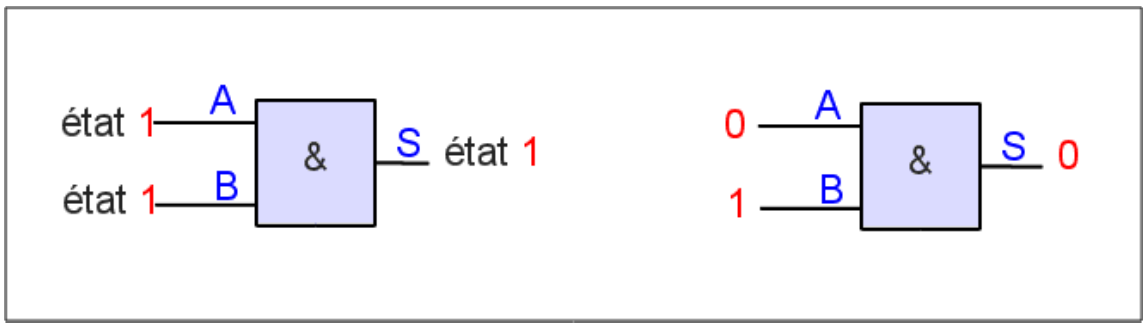
Parmi les assemblages de transistors les plus basiques, existent les **portes logiques**. Les portes **AND**, **OR**, **XOR** sont des exemples.

Voici un assemblage de six transistors, dont l'ensemble forme une porte logique AND :



(**source**)

Ces composants sont capables de donner une tension électrique de sortie en fonction des tensions d'entrées.
Par exemple, la tension de sortie d'une porte AND sera à l'état 1 si et seulement si les deux entrées sont elles aussi **toutes les deux** à l'état 1 :



Suivant les différentes valeurs en entrée, la sortie est à un niveau bien défini : soit 1 soit 0.
En réalisant un tableau de ces états, on crée ce qu'on appelle la **table de vérité** du composant.

Par exemple, voici la table de vérité des portes AND et XOR :

Porte AND		
entrée A	entrée B	entrée S
0	0	0
0	1	0
1	0	0
1	1	1

Porte XOR		
entrée A	entrée B	sortie S
0	0	0
0	1	1
1	0	1
1	1	0

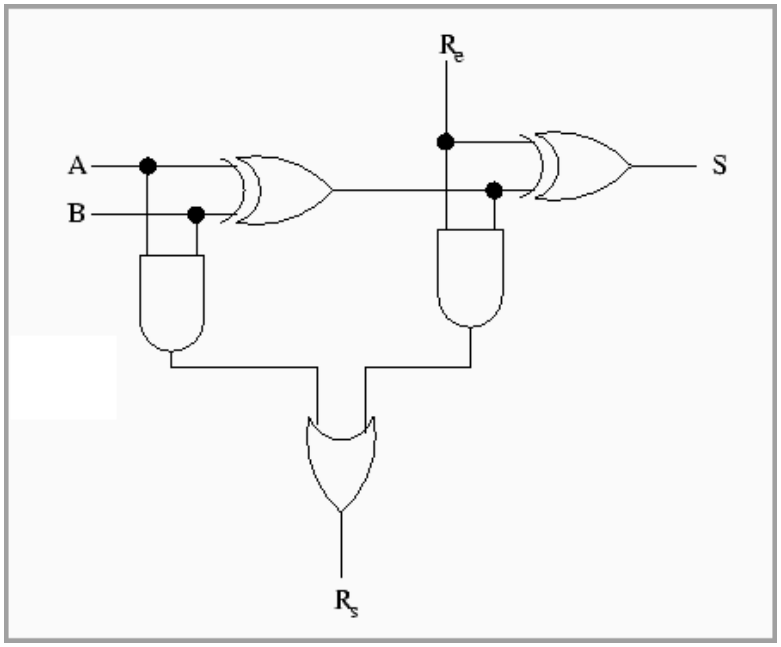
En fait, et peut-être l'avez vous remarqué, sur la porte AND : S = A × B et pour la porte XOR, S = A + B (en tenant compte que 2 en binaire donne 10 et que seul le dernier chiffre peut-être représenté).

Chaque porte permet d'effectuer une fonction précise (AND fait une multiplication, et XOR une addition).

On voit donc que les portes logiques permettent de comparer les états d'entrés pour donner un état en sortie.

Pour aller plus loin, on associe plusieurs de ces portes pour avoir des composants encore plus évolués comme des compteurs, des convertisseurs, des mémoires…
Au final, on arrive à fabriquer un très grand nombre de fonctions logiques différentes.

Voici un montage **additionneur avec retenue** réalisé avec deux portes NAND (au milieu), deux NOR (en haut) et une XOR (en bas) :



source

En si on met 1 sur A et 1 sur B, il mettra 1 sur la retenue Rs et 0 sur S. Ainsi, on lira « 1+1 = $(10)_{bin}$, où $(10)_{bin}$ correspond bien à 2 en décimal.

Autre exemple : en mettant 1 sur A et 0 sur B, il sortia 0 sur la retenue Rs et 1 sur la sortie S. On a donc 1+1 = 01 = 1.

La borne Re, c'est juste une jonction permettant d'ajouter une retenue précédente, d'un autre compteur. Il est ainsi possible d'associer plusieurs de ces compteurs pour faire des compteurs plus grands, avec plus de chiffres.
C'est là le début d'un ordinateur !

2.2.3 - Des portes logiques aux processeur

Si vous avez suivi :

- plusieurs transistors assemblés entre eux de manière précise forment une porte logique.
- plusieurs portes logiques donnent des circuits capables de faire des calcules simples (additions, multiplication, mémorisation…).
- beaucoup de portes logiques assemblées constituent un circuit intégré, capable de faire un très grand nombre de fonctions complexes. Un processeur est *un* type de circuit intégré.

Un processeur actuel comporte **plusieurs milliards de transistors**. Le nombre de transistors sur un processeur suit à peu près la **loi de Moore**, du nom d'un des fondateurs

d'Intel qui prédit alors cette loi empirique. Elle dit que la miniaturisation est telle que le nombre de transistors est doublé environ tous les deux ans.
Voici un exemple avec quelques processeurs d'Intel :

- 1971 : Intel 4004 : **2 300** transistors
- 1978 : Intel 8086 : **29 000** transistors
- 1982 : Intel 80286 **275 000** transistors
- 1989 : Intel 80486 : **1 160 000** transistors
- 1993 : Pentium : **3 100 000** transistors
- 1995 : Pentium Pro : **5 500 000** transistors
- 1997 : Pentium II : **27 000 000** transistors
- 2001 : Pentium 4 : **42 000 000** transistors
- 2004 : Pentium Extreme Edition : **169 000 000** transistors
- 2006 : Core 2 Quad : **582 000 000** transistors
- 2010 : Core i7 : **1 170 000 000** transistors
- 2012 : Core i7 SandyBridge : **2 270 000 000** transistors

En 2013, la puce AMD Radeon R9 295X2 atteint même 12 400 000 000 de transistors.

(**source**)

L'augmentation du nombre de transistors sur une puce est possible par la miniaturisation des transistors. Mais ça ne sera pas possible indéfiniment : en 2013, les transistors sont gravés en 22 nanomètres, soit 22 milliardièmes des mètres : c'est la taille de quelques dizaines d'atomes.
En réduisant trop la taille de la structure en maillage de transistors, il risque de se créer des interactions électroniques parasites entre les transistors eux même (l'effet tunnel par exemple, un effet de la mécanique quantique) et cela peut créer des problèmes…
Un jour, il faudra donc utiliser une solution autre que celle des puces en silicium, si l'on veut continuer à augmenter la puissance des ordinateurs…

Sommaire

3 - Comment calculer ?

3.1 - Les cycles d'horloge

Une notion n'a pas encore été abordé : la notion du temps.

Avec ce qu'on a jusqu'à présent, les transistors et tout ça, on peut effectuer un calcul et le résultat est donné, mais ça s'arrête là.
Or, le processeur effectue plein de calculs à la suite ! En fait, on dit qu'il travail en cycles. Ces cycles sont effectués sur l'ensemble du système de transistors et chaque cycle permet d'avancer dans les calculs.

Le nombre de cycles par seconde est nommé la fréquence du processeur (exprimée en Hertz : 1 cycle en 1 seconde = 1 Hz).
Pour donner un ordre de grandeur, le processeur de mon ordinateur tourne à une fréquence de 3,200 GHz : il effectue donc 3,2 milliard de cycles par seconde.

Ce cadencement du processeur est assuré par une horloge (un quartz qui vibre sous l'effet *piézoélectrique*).
Concrètement, le signal d'horloge est un signal électrique alternatif qui déclenche un nouveau cycle de calcul à chaque période du signal.

Le processeur suit le signal d'horloge et effectue un nouveau calcul à chaque cycle. Les calculs sont donc effectués les uns à la suite des autres, souvent en réutilisant les résultats du cycle précédent, stockés en mémoire.

3.2 - Du matériel au logiciel

On vient donc de voir qu'un processeur est constitué de milliards de transistors montés ensembles. Ces transistors comparent des tensions électriques, afin de réaliser des calculs logiques. Finalement, en effectuant des millions des calculs logiques à la suite, on arrive à effectuer des calculs complexes.

Les calculs sont effectués à partir de données : des données en binaire contenu de la mémoire. Ces tensions de 0 Volt ou 5 Volt sont appliquées aux bornes des portes logiques et ce sont elles (les portes logiques) qui calculent et qui donnent un résultat en sortie.

Ainsi donc, les 0 et les 1 que contient la mémoire et que traite le processeur proviennent des transistors qui sont à 0V et à 5V.

Si je dis par exemple que la mémoire contient le mot 1011, ça veut dire qu'un transistor affiche un "1" en sortie, que le suivant affiche un 0, les deux suivant un 1.

Si l'on change une des tensions d'entrées, les portes logiques vont calculer et le changement se répercute sur tout le système de transistors.

Nous somme là exactement à la limite « matérielle » et « logique » d'un ordinateur, et c'est

là tout l'ingéniosité qu'il a fallu déployer pour créer un ordinateur capable de calculer, simplement avec quelques composants électroniques de base.

Sommaire

4 - Les programmes

Pour servir à quelque chose, l'ordinateur doit recevoir des données sur lesquelles il va calculer. Ces données sont fournies par un programme

Un programme, c'est une suite d'instructions élémentaires, gravées en dur sur un disque ou dans la matériel. Quand on allume l'ordinateur, les transistors recoivent un 1 ou 0 en fonction du programme gravé dans la mémoire.

Chaque *cycle* d'horloge permet d'avancer un peu plus dans les calculs.

Dès que l'ordinateur est allumé, il calcule. En général, les premiers calculs sont là pour permettre de lire d'autres programmes, et ainsi de suite : un simple bouton on/off permet alors de charger en mémoire des milliard de bits.

4.1 - Le binaire

Pour ceux qui désirent apprendre à compter en binaire et en hexadécimal, j'ai un cours sur ça : le binaire et l'hexadécimal.

Un processeur ne sait calculer qu'en binaire (avec des 0 et des 1) : les programmes sont donc fournis en binaire.

Donc si on veut que le processeur effectue un calcul, il faudrait lui dire quelque chose de ce genre là : « *01010100 10110101 01101101 110110101 10101010 11011110 10101011 10111100 1100011 0101101 01111110* ».
Ce qui est totalement incompréhensible pour un humain.

On peut déjà écrire tout ça en hexadécimal, pour alléger l'écriture : « 54 B5 6D DA D5 6F 55 DE 63 5A FC », mais ça reste illisible. Le processeur, lui ne comprend que ça.

4.2 - La langage assembleur

Le processeur exécute donc des choses incompréhensibles en binaire comme « *01010110111110* ». Pour rendre ça compréhensible, on a créé un langage : **l'assembleur**. À chaque instruction en binaire correspond un mot simple à retenir en assembleur.
C'est ce langage que le programmeur utilise pour créer ses programmes.

Le premier langage non binaire fut inventé en 1951 par une femme, Grace Hopper.

Voici, par exemple, quelques-unes des quelques dizaines d'instructions utilisées pour programmer un processeur x86 :

- **mov** : (move) déplace le contenu d'une case mémoire dans une autre case.
- **inc** : (increment) incrémente la valeurs contenus dans une case mémoire.
- **neg** : (negative) prend un nombre et donne son opposé.
- **cmp** : (compare) compare deux nombre.
- **jmp** : (jump) « saute » à un autre endroit du programme.

Le langage assembleur n'est constitué que des instructions basiques comme celles ci. Pour effectuer des calculs plus complexes, il faut au préalable décomposer ce calcul en une suite d'instructions qu'il connaît. Tout comme on peut décomposer une puissance en une suite de multiplications, puis en une suite d'additions (2^4 = 2×2×2×2 = 2+2+2+2+2+2+2+2).

Pour que le processeur comprenne notre programme codé avec des **mov** et des **inc**, il faut les retranscrire en binaire. Cette étape est nommée **l'assemblage**.

4.2.1 - Exemple d'un programme en assembleur

Voici à quoi ressemble un petit programme que j'ai fait avec le logiciel Apoo (en bleu, les commentaires) :

```
loadn 14 R0    # place le nombre 14 dans le registre R0
loadn 2 R1     # place le nombre 2 dans le registre R1

storer R1 R2   # copie le contenue de R1 dans R2
storer R1 R3   # copie le contenue de R1 dans R3
add R0 R2      # addition R0+R2 et place le résultat dans R2
mul R0 R3      # multiplication R0×R3 et place le résultat dans R3

store R0 0     # place le contenu de R0 à l'adresse mémoire @0
store R1 1     # place le contenu de R1 à l'adresse mémoire @1
store R2 2     # place R2 dans @2
store R3 6     # place R3 dans @6
```

Au final, la mémoire contiendra quelques valeurs :

Adresse	Contenu (en base 10)	Contenu (en base 2)
@0	14	0000 1110
@1	2	0000 0010
@2	16	0001 0000
...	-	...
@6	28	0001 1100
...	...	...

On a utilisé le processeur pour aller modifier des données présentes en mémoire. C'est ça, la programmation. Évidemment un programme "normal" est beaucoup, beaucoup plus long, car ces données en mémoire sont réutilisées et retraitées par le processeur et ainsi de suite, mais voilà pour le principe.

L'exemple ci dessus, est ce que voit le programmeur : il écrit des **mov** et des **add** sur un fichier texte. Mais le processeur, lui ne calcule qu'avec des 0 et des 1. L'assemblage constitue l'étape de transformation de l'assembleur en binaire.

Voici un exemple totalement faux d'un code en assembleur et son équivalent en binaire :

Assembleur	Hexadécimal	Binaire
loadn 14	F3 21 4F 62	11110011 00100001 01001111 01100011
R0	69 6E 2F 62	01101001 01101110 00101111 01100010
loadn 2 R1	61 73 A2 7D	01100001 01110011 01101000 00001010
storer R1	68 0A 0A 69	00001010 01101001 01101110 01100011
R2	6E 63 B2 65	10110010 01100101 01101101 01100101
storer R1	6D 65 6E 74	01101110 01110100 00111101 00110000
R3	3D 30 0A A0	00001010 10100000 01100110 01101111
add R0 R2	66 6F 72 20	01110010 00100000 01101001 00100000
mul R0 R3	69 20 69 6E	01101001 01101110 00100000 00101010
store R0 0	20 2A 2E 6A	00101110 01101010 01110000 01100111
store R1 1	70 67 3B 0A	00111011 00001010 00001010 00001001
store R2 2	0A 09 64 6F	01100100 01101111 00001010 00001001
store R3 6	0A 09 09 6C	00001001 01101100 00110010 10111010
	65 74 20 FF	00010000 01111111 10110100 10110111
	69 6E EE 72	01110111 00111001 00110010 10110110
	65 6D 65 6E	10110010 10110111 00111010 00010000
	74 20 2B BD	00010101 11011110 10011000 10010001
	31 22 0A 09	00000101 00000100 00001001 01101110
	09 6E 62 3D	01100010 00111101 00100010 00110000
	22 30 30 30	00110000 00110000 00110000 00100100

Je pense que vous commencez à comprendre maintenant : il est possible de programmer un ordinateur en binaire, mais c'est lourd et long. La langage assembleur nous simplifie un peu la tâche.

4.2.2 - L'assembleur, peu pratique ?

Même si coder en assembleur et plus rapide que le binaire, cela reste une étape fastidieuse : manipuler les données et les placer octet par octet dans la mémoire ou dans les registres, c'est long.

De plus que chaque processeur possède des noms et des numéros d'instructions différents ! Ainsi, un code en assembleur qui fonctionne sur un processeur Pentium ne marchera pas forcément sur un AMD 64 ou sur un processeur ARM. En gros, si vous écrivez un programme sur votre ordinateur, il ne marchera pas sur le mien ! C'est embêtant…

Si le programme ne marche que sur un processeur et non sur un autre, on dit que le programme n'est pas portable.

Dès les débuts de l'informatique moderne on a donc inventé un moyen pour palier à ces deux inconvénients que sont la lenteur à coder et la non portabilité : les langages de haut niveau.

4.3 - Les langages de haut niveau

Les langages de haut niveau permettent une programmation rapide et plus simple que l'assembleur. Le code source est humainement compréhensible.

On dit « *haut niveau* » pour un langage, mais cela n'a rien à voir avec l'idée exprimée dans l'expression populaire « sportif de haut niveau » opposée à l'expression « sportif du dimanche ». Ici il s'agit juste de dire que ces langages sont éloignés du code machine : un langage de *haut niveau* est compréhensible par un humain et éloigné du binaire, alors que le langage de *bas niveau*, c'est le contraire.

les niveaux de langage		
Niveau du langage de programmation	exemple de langage	exemple de code
le plus bas possible	le Binaire	0101101 00100101 11111010 00010111

les niveaux de langage		

bas niveau	Langage machine : Assembleur	mov eax, 39
		add eax, 2
		inc eax

haut niveau	Langage C	int main() {
		int x = 39;
		x = x+2;
		x++;

le plus haut niveau	Le langage parlé	« *prends le nombre 39 et ajoutes lui le nombre 2. Quand c'est fait, augmente le nombre obtenu de 1.* »

Les langages de haut niveau sont déjà plus faciles à comprendre et à écrire. C'est un gros avantage sur l'assembleur : on gagne énormément de temps à écrire nos programmes ou à lire les codes sources des autres programmeurs.

4.3.1 - Différents langages de haut niveau

Parmi les langages de haut niveau, il y a une multitude de langages différents. Le C est un exemple, mais peut-être avez vous entendu parler des langages **Java**, **Python**, **PHP** ? Ce sont tous des langages de haut niveau ! Il en va des préférences du programmeur et de l'application à concevoir que se fait le choix d'utiliser l'un des langages plutôt qu'un autre pour faire ses programmes.

Chaque langage possède sa syntaxe et ses objectifs. Par exemple, le langage Perl est très bon pour chercher et analyser dans des fichiers textes, le langage PHP est utilisé pour générer des sites web.
Il n'est pas rare non plus de voir un programme qui soit écrit en différents langages : on utilise le langage le plus convenable pour chaque tâche.

Voici un petit tableau avec le code qui permet d'afficher le message « hello world » dans quelques langages :

langage	exemple de code

C	main() {
	printf("hello, world\n");
	}

Java	class HelloWorld {
	static public void main(String args[]) {
	System.out.println("Hello World!");

	} }
JavaScript	document.write('Hello World');
PHP	<?php echo 'Hello World!'; ?>
LOLCODE (:-))	HAI CAN HAS STDIO? VISIBLE "HAI WORLD!" KTHXBYE
Python	print "Hello World"

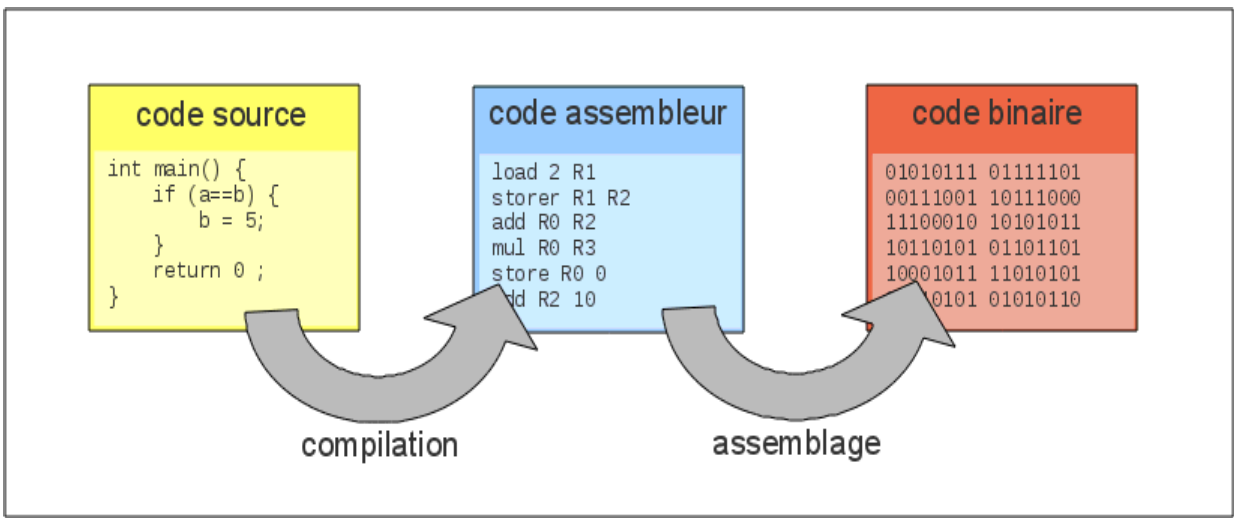
Bref, vous voyez qu'il y'en a pour tous les goûts :)

Ce qu'il faut retenir de ces langages de haut niveau, c'est qu'ils sont portables ! C'est à dire qu'on peut coder un programme en Java ou en C, et on pourra utiliser le code source sur différents ordinateurs.

4.3.2 - La compilation

Le problème des langages de haut niveau c'est qu'ils ne sont absolument pas compréhensibles par le processeur ! Avant de pouvoir être utilisés, le code doit être **traduit** en langage assembleur.
Cette traduction, c'est ce qu'on appelle la **compilation**.

Une fois en langage assembleur, il faut ensuite l'assembler en langage binaire :



Ce qui est bien pratique, puisque le programmeur n'a qu'à coder son logiciel dans un langage haut niveau, et c'est l'utilisateur (ou le distributeur) qui le compile ensuite en vue de l'installation sur un ordinateur.

Les langages de haut niveaux permettent donc d'accélérer la création d'un programme, **et** de le rendre portable !

Sommaire

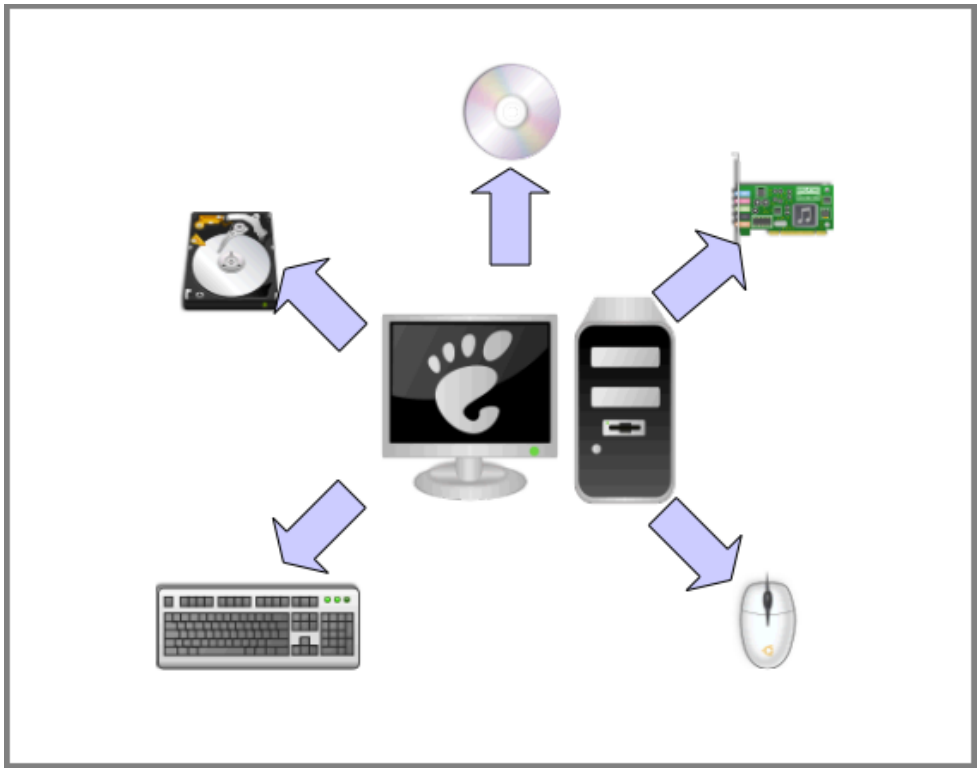
Résumé

- **Le programmes sont des suites d'instructions** (en binaire) que le processeur exécute.
- Le **langage assembleur** est une transcription humainement compréhensible des instructions en binaire.
- L'assembleur devient très vite long et lent à coder. On a donc inventé **les langages de haut niveau**.
- Les langages de haut niveau sont faciles et rapides à coder mais il faut passer par une étape supplémentaires (**la compilation**) pour pouvoir utiliser finalement le programme.
- Enfin, sachez qu'il y a **plein de langages** de haut niveau ayant chacun sa propre syntaxe d'écriture (C, Java, Python…).

Sommaire

5 - De l'ordinateur minimal à l'ordinateur de bureau

Pour le moment, on est resté dans le cadre d'un **ordinateur minimal** : il n'a que de la mémoire et un processeur pour faire les calculs. Or, vous savez tous qu'un ordinateur c'est bien plus que ça : c'est un disque dur, un clavier, un écran, une carte son, etc. fonctionnant ensemble.



5.1 - Contrôler chaque composant

Pour que l'ordinateur puisse utiliser un composant, il doit communiquer avec lui, lui envoyer des impulsions électriques en binaire. Comment ces codages sont-ils reçues par le composant ? Comment le processeur fait-il pour communiquer avec un composant ?

Cela se fait par l'intermédiaire de la mémoire.
Chaque composant a réservé un petit emplacement dans la mémoire et il est constamment à l'affût d'un changement dans son espace mémoire. S'il y a un changement, il réagit en conséquence.

Voici par exemple les différents espaces mémoires de quelques périphériques sur mon ordinateur actuellement (les adresses mémoires sont affichées en hexadécimal). On retrouve ici quelques composants : USB, Carte WiFi, Audio :

Plage mémoire	Élément
00000000-0000ffff	reserved
…	…
000c0000-000cefff	Video ROM
f7c00000-f7c0ffff	JMicron Technology Corp. JMC250 PCI Express Gigabit **Ethernet** Controller

Plage mémoire	Élément
f7c10000-f7c1ffff	JMicron JMC2x0 PCI Express **Ethernet** driver
f7d00000-f7d03fff	Realtek Semiconductor Co., Ltd. 802.11b/g/n **WiFi Adapter**
f7f00000-f7f03fff	Intel Corporation 6 Series Chipset Family HD **Audio** Controller
f7f06000-f7f067ff	AHCI **SATA** low-level driver
f7f07000-f7f073ff	Intel Corporation 6 Series Chipset Family **USB Controller** #1
…	…
fec00000-fec00fff	reserved
fec00000-fec003ff	IOAPIC 0
fed00000-fed03fff	reserved
ff000000-ffffffff	…

Avec un espace mémoire pour chaque composant matériel, le système d'exploitation pour envoyer des instructions à un composant en particulier.
C'est ce qui se passe par exemple avec le lecteur CD quand je clic sur « éjecter le disque » : le logiciel va détecter le clic, et le processeur va écrire un 1 dans la bonne case mémoire, et le lecteur CD va s'ouvrir.

C'est schématique, mais c'est en réalité bien comme ça que ça se passe, pour chaque périphérique.

Pour les périphériques d'entrées comme le clavier ou la souris, c'est l'inverse : ainsi la pression d'une touche du clavier va modifier la mémoire et c'est le processeur qui va lire les changements.

Tout est lié et fonctionne dans une certaine harmonie, où **le processeur joue le rôle centrale de coordinateur et discute via l'intermédiaire de la mémoire vive où les autre périphériques lisent ce qu'ils doivent faire**.

Tous les éléments sont reliés par des câbles, des nappes ou des prises, à l'unité centrale. L'ennui, c'est que le processeur sera très vite débordé si tous les composants veulent travailler ensemble. Par exemple, si je veux envoyer la connexion internet de la carte réseau à la carte wifi : la carte réseau ne sait *que* écrire dans son espace, pas dans celle de la carte wifi. C'est la limite du matériel.

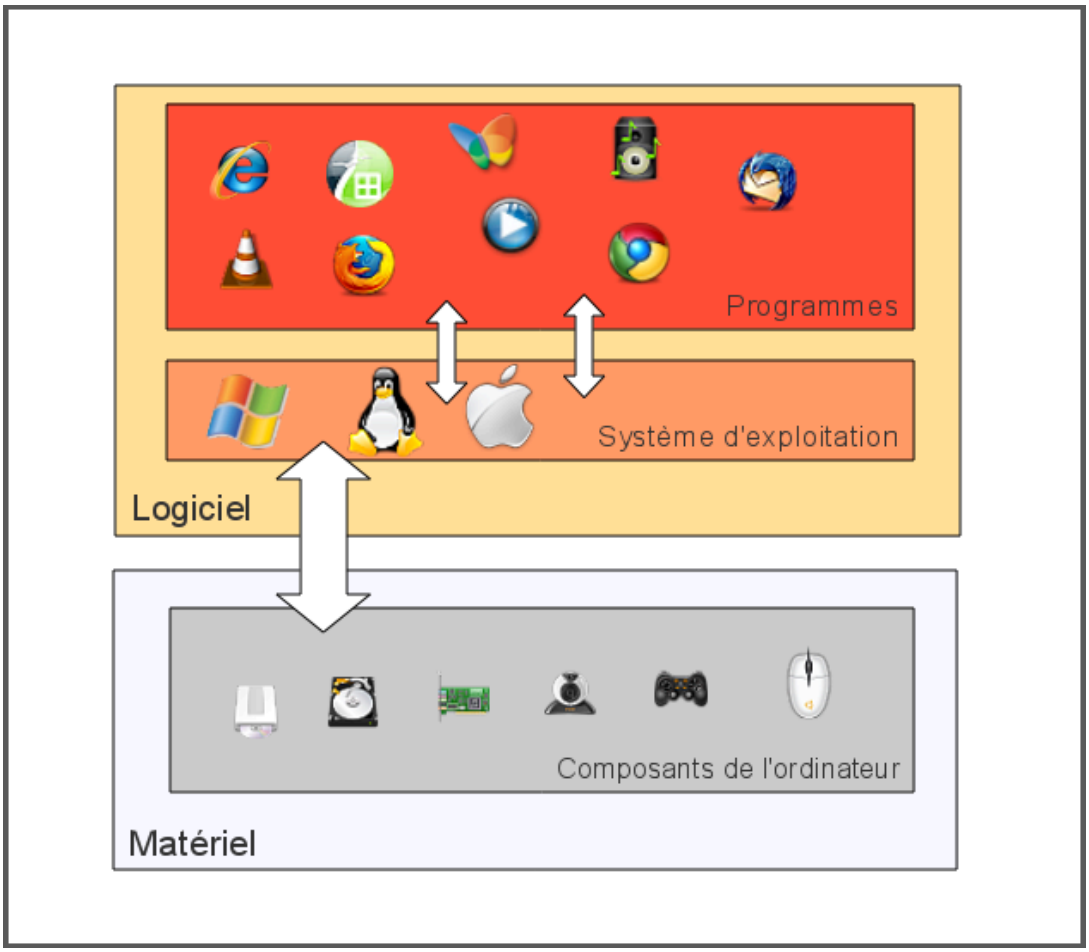
Pour arranger tout ça, il faut un super-programme : un programme qui sache communiquer avec chaque composant et qui puisse partager et organiser tout ça…

5.2 - Le système d'exploitation

Le système d'exploitation (abrégé « SE », voire même le plus souvent « OS » pour *Operating System*, en anglais), c'est justement ce super-programme chargé de coordonner l'ensemble du fonctionnement d'un ordinateur et de tous les périphériques.

Un OS sait comment fonctionne le disque dur ou comment marche une carte son : ainsi, quand un logiciel demande à écrire sur le disque, il va demander à l'OS de le faire.

L'OS est le seul à pouvoir autoriser un logiciel à écrire sur le disque dur ou à accéder à la mémoire ou au processeur.



Vous le voyez, **l'OS est vraiment le seul maître à bord** : il est l'intermédiaire entre les logiciels, l'utilisateur et le matériel.

5.3 - Exemples

Un programme seul ne peut rien faire sur un ordinateur, il faut forcément un système d'exploitation pour faire l'intermédiaire.

Sachez donc que l'OS est un logiciel important sur un ordinateur. Il existe divers systèmes d'exploitations, fabriqués et conçus par des entreprise informatiques divers.

Citons les trois principaux OS :

- **Windows**
  Présent sur 88 % des ordinateurs dans le monde, il est conçu par Microsoft Corporation. Ce logiciel est cher (400€) mais il se trouve installé sur tous les ordinateurs quand vous en achetez un. Windows est apparu en 1984 pour concurrencer Apple.
- **OS-X**
  L'OS des ordinateurs d'Apple Inc (exclusivement). Mac OS-X est utilisé sur environ 11 % des ordinateurs du monde. Ce système fut crée en 1997 par la fusion du système des ordinateurs haut de gamme de la firme *NEXT Computers* et du « System 9 » d'Apple.
- **GNU/Linux**
  Contrairement à OS-X et Windows, ce système est maintenu par une communauté de programmeurs. Une communauté qui veut créer le meilleur système du monde, simplement par plaisir, sans réel but commercial. Certains de ces « contributeurs » sont des bénévoles passionnés, d'autres sont payés et sont employés par tous les grands noms de l'informatique qui souhaitent contribuer à GNU/Linux (Google, Intel, Adobe, IBM, HP, Microsoft mais aussi la NASA et autres entités universitaires). GNU/Linux est libre de tout droits et est **gratuit** pour tout le monde. Il a été créé en 1991 par un étudiant Finlandais : *Linus Torvalds*, comme alternative à UNIX (la référence de l'époque mais qui devenait de plus en plus cher). Linus Torvalds distribua ensuite son OS gratuitement à qui le voulait via l'internet. Aujourd'hui, GNU/Linux est utilisé par environ 1% des utilisateurs dans le monde.

Personnellement, j'utilise Mint, un système basé sur GNU/Linux.

Chaque OS (Linux, OS-X ou Windows…) permet de faire exactement les même choses : lancer des logiciels, surfer sur le net, écouter un CD, etc. mais leurs fonctionnement logicielle interne est différente.
Tout comme un briquet et une allumette permettent tous les deux de faire du feu mais avec un fonctionnement différent.

Le matériel est le même (disque dur, écran, souris…), les logiciels sont les mêmes (navigateur, lecteur média, traitement de texte…) seul change l'intermédiaire.

Sommaire

Résumé

- **L'ordinateur de bureau** est composé d'un tas de matériel (carte son, prise ethernet, carte WiFi…) et de plein de logiciels (navigateur, traitement de texte, etc.).
- Il faut **un système d'exploitation** (abrégé « OS ») qui centralise tout ça et qui gère de manière intelligente tout cela.
- L'OS est donc **l'intermédiaire entre la matériel et le logiciel**. C'est lui qui contrôle tout et qui permet à tout ce petit monde de fonctionner en harmonie.
- Il existe plusieurs OS différents (Windows, Linux, Solaris…), qui au final offrent le même résultat mais qui l'obtiennent de manière différente.

Sommaire

6 - Les logiciels

Maintenant nous avons :

- Un processeur qui sait calculer.
- Du matériel pour faire des choses (haut parleurs pour le son, clavier pour écrire, écran pour voir, etc.)
- Un système d'exploitation qui fait l'intermédiaire entre tout le matériel, les programmes et l'utilisateur.

Il reste à voir les logiciels : vous savez : Word, Firefox et tous les autres : comment fonctionnent ces logiciels.

Note : le processeur ne sait que calculer sur du binaire.
Pourtant, nos programmes ont été écrits en C ou en JAVA. Sachez que ces langages ne sont pas apparus comme ça, d'un coup de baguette magique.

Il a bien sûr fallu coder et construire les langages, ainsi que les compilateurs et les assembleurs. C'est de la programmation de bas niveau et dont je ne pourrais pas dire plus, par manque de compétences… Google vous en dira certainement plus…
Même chose pour tous les logiciels ainsi que les systèmes d'exploitations : l'utilisateur les achète sur un CD, mais avant de se retrouver là, il y a eu des programmeurs pour faire tout ça.

6.1 - Du programme aux logiciels

Bon, le programmeur est bien rigolo avec ses variables, son code machine et ses langages, mais tout ça nous permet-il d'avoir des fenêtres, des menus et des icônes ?

En fait : oui.
Mais il faut savoir que pour créer tout ça, c'est pas aussi simple que de programmer « *affiche moi 2+2 !* ».

Un programme permet de faire une fonction, comme calculer le nombre de lettres contenu dans un texte, écrire le résultat dans un fichier ou l'afficher à l'écran.

Lorsque nous voulons faire un grand nombre de choses différentes, on parle d'un **logiciel**.

Un logiciel est donc la somme de plein de programmes qui travaillent ensemble. C'est la même chose avec une voiture : chaque élément permet une fonction : le moteur donne l'énergie, le volant le levier de vitesse et les pédales constituent l'interface homme-machine, les vitres isolent du froid, etc.

Un logiciel est un programme très évolué qui communique avec l'utilisateur (via des boutons à cliquer, des cases à remplir…).
L'exécution du logiciel se fait en permanence et de la manière décidée par l'utilisateur : chaque fois que l'utilisateur clique quelque part, le logiciel réagit et un programme se lance, une fois exécuté, le logiciel attend le prochain clic.

L'utilisateur ne voit peut-être qu'une fenêtre avec des menus et des boutons, mais le processeur reçoit bien des instruction lui !

Sommaire

7 - Ressources…

Ce tutoriel explique comment un objet matériel arrive, via des 0 et des 1 à calculer ce qu'on lui dit de faire au moyens de clics sur la souris et de tapes sur le clavier.
Tous les systèmes informatisés fonctionnent ainsi avec des transistors, du binaire, etc.

En revanche, ces explications ne vous apprennent pas comment fonctionne le protocole IP ni comment marche un MP3. Pour cela, je vous renvoie sur les pages Comprendre L'ordinateur, de Sebsauvage. Vous y trouverez des explications **simples** sur divers éléments logiciels d'un ordinateur.

- www.sebsauvage.net/comprendre/ : des pages pour comprendre le fonctionnement de divers principes de base de l'informatique.
- www.siteduzero.com : vous savez maintenant comment fonctionne un ordinateur, peut-être voulez vous désormais apprendre à programmer, pour lui faire faire exactement ce que vous voulez ?

- - www.commentcamarche.net : un problème ? Une question ? CommentÇaMarche est un forum où vous pouvez poser vos questions sur l'informatique. Il y a aussi une documentation technique.
- www.bortzmeyer.org : un blog technique sur les réseaux, dont l'internet.
- Et ma série de 3 articles consacrée à la science de la mémoire d'ordinateur et aux transistors :
  - C'est quoi, un semi-conducteur ?
  - Comment fonctionne un transistor ?
  - Comment fonctionne la mémoire flash d'un lecteur SSD ?

Diverses icônes utilisés dans mes images ont été emprunté aux thèmes graphiques crée par François Vogelweith. Les icônes sont sous licence Creative Commons.

Sommaire

Page crée en octobre 2010.
Dernière mise à jour le samedi 5 mars 2016.

Timo Van Neerden - lehollandaisvolant.net