IV- Comment fonctionne un ordinateur ?

L'ordinateur est une alliance du *hardware* (le matériel) et du *software* (les logiciels). Jusqu'à présent, nous avons surtout vu l'aspect « matériel », avec les interactions entre le processeur et la mémoire principale. C'est ce qui se passe au plus bas niveau de la machine. Mais lorsque l'utilisateur donne un ordre à l'ordinateur pour réaliser une action, on est à un tout autre niveau, et de nombreuses médiations, pour ne pas dire des pérégrinations, doivent se produire avant d'arriver à l'exécution finale, avec des signaux électriques 0 et 1 qui sont aiguillés à travers le réseau électronique dans les unités de traitement. Et ce n'est pas fini pour autant, car souvent les résultats vont devoir remonter sur l'écran. C'est là qu'entre en ligne de compte l'aspect « logiciel », notamment ce que l'on appelle le système d'exploitation de la machine. Tout commence au démarrage de l'ordinateur.

1) Le démarrage de l'ordinateur, du BIOS au système d'exploitation

Lorsque l'on allume l'ordinateur, le BIOS (*basic input output system*) prend en charge le démarrage - le *boot*. Installé de façon permanente dans la mémoire ROM [1], ce programme d'amorçage enclenche les premières instructions. Celles-ci sont installées en mémoire à partir de l'adresse 0. Le processeur a aussi cette adresse 0 dans son registre d'instruction, et il est en mesure d'exécuter ce programme initial. Celui-ci assure notamment la gestion minimale des périphériques comme le clavier et l'écran. Si l'ordinateur est partitionné suivant deux systèmes d'exploitation, par exemple Windows et Linux, cela apparaît sur l'écran et l'utilisateur peut faire son choix en utilisant le clavier. Le BIOS lance alors le système d'exploitation concerné, en chargeant notamment son noyau (*kernel*) dans la mémoire principale.

Le système d'exploitation, jusque-là seulement installé sur le disque dur, prend le relais et va désormais contrôler toutes les opérations. Il est constitué de programmes appelés « fichiers système ». Ceux-ci sont inaccessibles à l'utilisateur humain, sauf par des moyens très spéciaux. C'est le système d'exploitation qui va notamment assurer la gestion globale des périphériques, par le biais de *drivers* (gestionnaires de périphériques). C'est grâce à lui aussi que l'utilisateur a l'impression que l'ordinateur réalise plusieurs tâches en même temps. Si l'une se produit sous ses yeux, d'autres ont lieu en arrière-plan. En fait il n'en est rien, puisque tout est séquentiel, mais le système d'exploitation gère successivement des bouts de tâches suivant certaines priorités, donnant l'impression d'une certaine simultanéité, grâce à la vitesse de traitement de l'ordinateur. Par exemple, s'il reçoit un ordre de lecture sur le disque dur, il le prend en compte, organisant notamment le mouvement du bras du disque dur, mais considérant que cela demande un certain temps (quelques millisecondes), il lance en attendant d'autres bouts de programme qui eux ne demandent que quelques centaines de nanosecondes. [2] Même chose pour l'écriture sur le disque dur, avec le système d'exploitation qui commence par chercher de la place libre sur le disque.

[1] Le BIOS a été « flashé » au préalable dans cette mémoire morte.

[2] Le passage momentané d'un programme à un autre s'appelle une interruption. Cet évènement fait que le processeur arrête momentanément l'exécution d'un programme pour en exécuter au plus vite un autre. L'ancien programme sera ensuite repris là où il avait été stoppé. Lorsqu'un périphérique souhaite intervenir,

2) Le cheminement des données et instructions d'un programme

A ce stade nous voyons qu'il existe plusieurs sortes de programmes moulinés par l'ordinateur :

- ceux qui sont préétablis dans le système d'exploitation.
- ceux qui sont intégrés aux divers logiciels ajoutés par l'utilisateur, comme un traitement de texte par exemple. Des fichiers de textes, qui n'ont rien à voir avec des fichiers programmes, peuvent alors être affichés sur l'écran, par le biais d'un programme de lecture et d'écriture.
- ceux qui sont créés par l'utilisateur, grâce à l'utilisation d'un langage de programmation, comme le langage C. On dit qu'il s'agit d'un langage de haut niveau, car il est proche du langage humain, les subtilités en moins. Pour devenir compréhensible par la machine, il passe dans un compilateur qui le traduit en langage machine, soit finalement une suite de 0 et de 1.

Si différents soient-ils, tous ces programmes sont réduits à une succession de chiffres binaires et subissent le même destin. Dès qu'ils sont appelés, ils sont envoyés du disque dur vers la mémoire principale qui est en liaison directe avec le processeur. Lorsque l'utilisateur clique sur une icône ou sur un nom de fichier présents sur l'écran, l'ordinateur va chercher sur le disque dur le programme correspondant et les données associées, par le biais du système d'exploitation, et il l'envoie vers la mémoire principale - la mémoire vive. Tout passe par là. On dit que le programme est chargé en mémoire. Le processeur se charge de le lire et d'exécuter ses instructions l'une après l'autre. A chaque instant de nombreux programmes s'accumulent ainsi dans la mémoire vive. Si celle-ci tend à déborder, le processeur envoie le trop plein vers le disque dur, qui fonctionne à son tour comme mémoire vive, mais beaucoup plus lentement, On appelle *swap* cette opération d'échange entre la mémoire principale et le disque dur. Cette combinaison de mémoires, elle aussi gérée par le système d'exploitation, forme ce que l'on appelle la mémoire virtuelle, bien plus vaste que le strict contenu de la mémoire principale.

Prenons un exemple simple pour voir comment circule un ordre donné par l'utilisateur sur un périphérique. L'ordinateur étant allumé et prêt à l'emploi, tapons sur une touche du clavier, par exemple celle de la lettre *a*. Rien ne se passe. Mais maintenant appelons notre logiciel de traitement de texte ou tout simplement un éditeur de texte intégré au système d'exploitation, - c'est une forme rudimentaire de traitement de texte, comme *Notepad* sous Windows. Celui-ci ouvre une page blanche qui apparaît sur l'écran. Si nous tapons sur la lettre *a*, celle-ci surgit aussitôt sur l'écran. Heureusement pourrait-on dire, car l'ordinateur se comporte comme une machine à écrire.[3] Mais une machine à écrire est simple à comprendre : en appuyant sur la touche marquée *a*, une tige rigide qui lui est associée et au bout de laquelle se trouve gravée la lettre vient frapper un ruban encreur, et cela inscrit la lettre *a* sur une feuille de papier. Puis le chariot où se trouve le papier se décale légèrement en attendant la lettre suivante. Dans l'ordinateur, c'est beaucoup plus compliqué.

Pour chaque touche frappée, le système d'exploitation lance un programme de lecture-écriture, et le clavier peut transmettre son code binaire ASCII bit par bit sur le lien série qui le relie à

---

il envoie une demande d'interruption au processeur. Un système de contrôle privilégie le programme qui a la plus grande priorité.

[3] Un avantage essentiel d'un traitement de texte sur ordinateur par rapport à une machine à écrire au fonctionnement purement mécanique est le passage à la ligne automatique. Avec une machine à écrire, il fallait faire manuellement le retour du chariot en actionnant un levier.

l'ordinateur. Le contrôleur de clavier signale la présence de ce caractère au processeur. En général le contrôleur possède deux registres : l'un indique qu'un caractère est disponible, donnant ainsi un ordre d'interruption, l'autre contient le caractère. Cela évite de passer par la mémoire principale. Sinon le caractère serait envoyé dans la mémoire principale à une adresse spécifique associée au clavier. Le processeur est alors en mesure de lire le caractère. Puis le trajet se fait en sens inverse, via le contrôleur de l'écran, et le caractère se trouve finalement affiché.

Supposons maintenant que l'on clique sur une icône associée à un fichier. Cela correspond à une demande de lecture, et le système d'exploitation est censé connaître l'adresse du premier secteur de ce fichier, ainsi que le nombre de secteurs contigus occupés par le fichier. En fait ces adresses sont converties au niveau matériel en trois composantes : le numéro du plateau concerné sur le disque dur, puis la piste concernée, et enfin le secteur concerné sur cette piste. Le contrôleur du disque dur est alors en mesure de positionner la tête de lecture sur les blocs correspondant au fichier, et il signale au processeur qu'il peut les lire l'un après l'autre, via la mémoire principale qui les réceptionne.

Au plus bas niveau de la machine, tout se ramène à un métabolisme entre le processeur et la mémoire principale (ou un registre spécifique) pour convertir les instructions élémentaires en une circulation d'impulsions électriques dans les circuits intégrés, aboutissant à l'exécution des instructions. Nous avions déjà entrevu dans le chapitre précédent comment s'effectuait chaque cycle correspondant à une instruction. Nous allons le préciser maintenant.

3) Le cycle d'une instruction, ou le travail élémentaire de l'ordinateur

Cela se fait essentiellement en trois étapes :

- Recherche de l'instruction

Initialement on a un bout de programme placé dans la mémoire principale et qui commence à l'adresse 100 par exemple. Imaginons par exemple qu'il s'agisse tout simplement d'additionner deux nombres. Plus précisément l'instruction en binaire exprime qu'il faut additionner ce qui est dans la mémoire à l'adresse $X = 50$ avec ce qui est à l'adresse $Y = 60$ et placer le résultat à l'adresse $Z = 70$. A un certain moment, le compteur ordinal (ou *PC*, compteur de programme) de l'unité centrale contient justement cette adresse numéro 100. L'unité de commande place alors cette adresse dans le bus d'adresse et émet un ordre de lecture en direction de la mémoire. L'instruction, ou le morceau d'instruction [4] contenu dans la cellule 100 est placé dans le bus de données pour être envoyée vers l'unité centrale et être stockée dans son registre instruction.

- Décodage de l'instruction et recherche des opérandes associés à l'opération

On a vu que l'opération concernée est une addition. Elle comporte les trois opérandes connus par leurs adresses. L'unité de commande prend ses éléments l'un après l'autre. Elle récupère le

[4] Cette instruction peut être contenue en entier dans une cellule mémoire. Ou bien elle peut en occuper quatre : une cellule contient l'opération (l'addition), la suivante l'adresse de la cellule où se trouve le premier nombre, la troisième l'adresse où se trouve le deuxième nombre, la quatrième l'adresse où placer le résultat. Le compteur ordinal devra alors passer de l'adresse 100 à l'adresse 104 pour la prochaine instruction. S'il n'y avait pas de compteur ordinal, il faudrait que l'instruction contienne un cinquième élément, à savoir l'adresse *T* où se trouve la prochaine instruction du programme, ce qui rallonge l'instruction. Aussi préfère-t-on éviter cette possibilité, en faisant en sorte que les instructions d'un programme se succèdent dans la mémoire principale, ce qui évite d'indiquer où se trouve l'instruction suivante.

contenu de l'adresse 50 qui arrive par le bus de données pour être stockée dans un registre. Puis elle fait de même pour l'opérande à l'adresse 60 qui est stocké dans un autre registre. Elle décode aussi l'opération à réaliser et envoie l'ordre d'addition à l'unité arithmétique et logique.

- Exécution de l'instruction

L'unité arithmétique exécute l'opération, et place le résultat dans un registre. L'unité de commande émet un ordre d'écriture pour ce résultat en direction de l'adresse 70 de la mémoire via les bus d'adresse et de données. Entretemps le registre *PC* (compteur ordinal) s'est incrémenté pour passer à l'instruction suivante, qui constitue la suite du programme.

Dans un langage proche du langage machine, mais évitant la succession de 0 et de 1 pratiquement illisible pour un humain, l'instruction prise en exemple s'écrit : *ADD X Y Z.* [5] Pour simplifier, on peut faire en sorte que le résultat soit placé à l'adresse *Y* de la mémoire, écrasant ainsi le deuxième terme de l'addition, ce qui s'écrit *ADD X Y*. Mieux encore, on utilise le registre accumulateur. On commence par faire *LOAD X*, qui charge dans l'accumulateur le nombre stocké à l'adresse *X*. Puis on fait *ADD Y* pour ajouter le nombre placé à l'adresse *Y* à celui de l'accumulateur, et le résultat sera lui aussi stocké dans l'accumulateur. Une telle méthode est d'autant plus intéressante si le programme concerné enchaîne des opérations, le résultat de l'une servant à la suivante, avec le registre accumulateur qui accumule les résultats provisoires. A la fin on peut donner l'ordre de charger le résultat final dans la cellule mémoire d'adresse *Z*, ce qui correspond à : *LOAD Accumulateur Z.*

Il reste à savoir comment les circuits électroniques de la machine doivent être construits pour pouvoir réagir afin de traiter la succession de chiffres binaires d'un programme et aboutir au résultat attendu. C'est l'objet des chapitres suivants.

Exercice : programmation préliminaire à la méthode de cryptage à clé publique RSA

On part d'un texte en clair. Pour pouvoir le crypter (ce que nous ne ferons pas ici), il faut commencer par le convertir en une succession de nombres de longueur donnée. Nous nous contenterons ici de prendre une longueur 4 (mais on peut obtenir des longueurs beaucoup plus grandes en utilisant ce que l'on appelle l'arithmétique de haute précision). Notre problème se réduit à remplacer le texte initial par une succession de nombres (en base 10) tous de longueur 4. C'est sur chacun de ces nombres que la méthode RSA va ensuite intervenir, en pratiquant notamment des calculs de puissances, transformant ces nombres en d'autres nombres formant le message crypté.

1) Faire une fonction qui convertit une lettre en son code ASCII.

C'est particulièrement simple en langage C. On part d'une lettre (ou un caractère) déclarée en *char*, et on le « caste » en *int*. Par exemple, on fait *char x* = 'S' ; et on demande l'affichage en entier *int*, par exemple *printf*(« %d », *x*) ; On verra sur l'écran le nombre 52. C'est le code ASCII de la lettre S (majuscule).

2) Refaire cette fonction pour convertir une lettre en nombre (décimal) de deux chiffres.

[5] Ce langage intermédiaire s'appelle assembleur.

Les lettres et les autres caractères, c'est-à-dire les chiffres de 0 à 9 ainsi que divers caractères spéciaux comme l'espace sont numérotés de 32 à 127 en code ASCII. Ainsi le nombre 32 correspond à *espace*, 48 est le chiffre 0, 65 est la lette *A*, 97 est la lettre *a*. On peut remarquer que l'on passe d'une lettre majuscule à la même lettre en minuscule en ajoutant 32. La conversion d'un code ASCII en nombre binaire demande donc 7 bits. Comment faire pour avoir des nombres en décimal de longueur 2, à savoir entre 0 et 99 ? Il suffit d'enlever 31 (voire 32) à leur code ASCII. Si on enlève 31, l'espace a comme numéro 1, et les caractères vont de 1 à 96. Ils sont tous dans la zone concernée.

Pour ce bout de programme, il suffit de faire : (int)*x* – 31 ;

3) On suppose que le texte initial est déjà placé lettre par lettre dans un tableau *char a*[]. Faire le programme qui transforme chaque paquet de deux caractères, donc de quatre chiffres, en nombre. Chaque nombre ainsi obtenu sera ensuite crypté suivant la méthode RSA, mais cela dépasse le cadre de ce cours.