6 Matériel et logiciels

Dans ce chapitre, nous décrivons de façon schématique le fonctionnement d'un ordinateur, ainsi que certaines limitations intrinsèques.

I Éléments d'architecture d'un ordinateur

Nous commençons par décrire le matériel informatique constituant un ordinateur, et permettant son fonctionnement. Notre but est de donner une idée rapide, sans entrer dans le détail logique du fonctionnement du processeur (portes logiques) et encore moins dans le détail électronique caché derrière ce fonctionnement logique (amplificateurs opérationnels, transistors etc.)

I.1 Modèle de Von Neumann

Pour commencer, interrogeons-nous sur la significtion-même du terme « informatique ».

Définition 6.1.1 (Informatique)

Le mot *informatique* est une contraction des deux termes *information* et *automatique*. Ainsi, l'informatique est la science du traitement automatique de l'information.

Il s'agit donc d'appliquer à un ensemble de données initiales des règles de transformation ou de calcul déterminées (c'est le caractère automatique), ne nécessitant donc pas de reflexion ni de prise d'initiative.

Définition 6.1.2 (Ordinateur)

Un ordinateur est une concrétisation de cette notion.

Il s'agit donc d'un appareil concret permettant le traitement automatique des données. Il est donc nécessaire que l'ordinateur puisse communiquer avec l'utilisateur, pour permettre l'entrée des données initiales, la sortie du résultat du traitement, et l'entrée des règles d'automatisation, sous la forme d'un programme. Le modèle le plus couramment adopté pour décrire de façon très schématique le fonctionnement d'un ordinateur est celui décrit dans la figure 6.1, appelé *architecture de Von Neumann*. Dans ce schéma, les flèches représentent les flux possibles de données.

Note Historique 6.1.3 (von Neumann)

John von Neumann (János Neumann) est un scientifique américano-hongrois (Budapest, 1903 - Washington, D.C., 1957). Ses domaines de recherche sont très variés, de la mécanique quantique aux sciences économiques, en passant par l'analyse fonctionnelle, la logique mathématique et l'informatique. Il contribue au projet Manhattan,

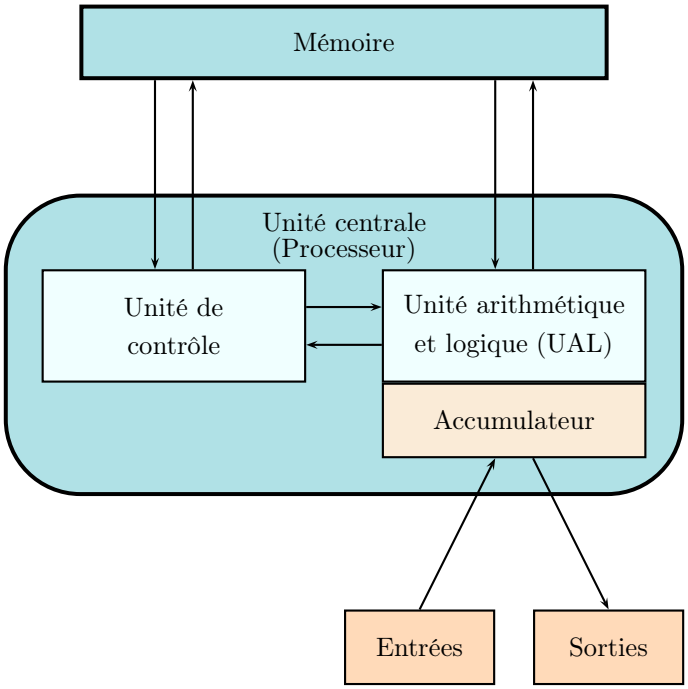


FIGURE 6.1 – Modèle d'architecture de Von Neumann

et notamment à l'élaboration de la bombe A, puis plus tard de la bombe H. Son nom reste attaché à la description de la structure d'un ordinateur, en 1945. C'est sur ce schéma qu'ont ensuite été élaborés les premiers ordinateurs. Si les ordinateurs actuels sont souvent beaucoup plus complexes, leur schéma grossier reste cependant très proche du schéma de l'architecture de von Neumann.

Note Historique 6.1.4 (architecture de von Neumann)

Le schéma d'un ordinateur (architecture de von Neumann) a été donné en 1945 par John von Neumann, et deux collaborateurs dont les noms sont injustement restés dans l'oubli : John W. Maukly et John Eckert. John von Neumann lui-même attribue en fait l'idée de cette architecture à Alan Turing, mathématicien et informaticien britannique dont le nom reste associé à la notion de calculabilité (liée à la machine de Turing), ainsi qu'au décryptage de la machine Enigma utilisée par les nazis durant la seconde guerre mondiale.

- **Entrées - sorties**
  Les entrées et sorties se font au moyen de périphériques spéciaux destinés à cet usage.
  * Les périphériques d'entrée permettent à un utilisateur d'entrer à l'ordinateur des données, sous des formats divers : clavier, souris, scanner, webcam, manettes de jeu etc.
  * Les périphériques de sortie permettent de restituer des informations à l'utilisateur. Ils sont indispensables pour pouvoir profiter du résultat du traitement de l'information : écran, imprimante, hauts-parleurs, etc.
  * Certains périphériques peuvent parfois jouer à la fois le rôle d'entrée et de sortie, comme les écrans tactiles. Ils peuvent aussi avoir des fonctions non liées aux ordinateurs, comme certaines photocopieuses, dont l'utilisation essentielle ne requiert pas d'ordinateur, mais qui peuvent aussi faire office d'imprimante et de scanner.
- La **mémoire** permet le stockage des données et des logiciels (programmes) utilisés pour les traiter. Ce stockage peut être :
  * définitif (mémoire morte, ou ROM, inscrite une fois pour toute, et non modifiable, à moins d'interventions très spécifiques) ;

  - temporaire à moyen et long terme (stockage de données et logiciels que l'utilisateur veut garder, au moins momentanément) ;
  - temporaire à court terme (données stockées à l'initiative du processeur en vue d'être utilisées ultérieurement : il peut par exemple s'agir de résultats intermédiaires, de piles d'instructions etc.).

  L'architecture de von Neumann utilise le même type de mémoire pour les données et les programmes, ce qui permet la modification des listes d'instructions (elle-mêmes pouvant être gérées comme des données). Ce procédé est à l'origine des boucles.

  Nous reparlerons un peu plus loin des différents types de mémoire qui existent.
- Le **processeur** est le cœur de l'ordinateur. C'est la partie de l'ordinateur qui traite l'information. Il va chercher les instructions dans un programme enregistré en mémoire, ainsi que les données nécessaires à l'exécution du programme, il traduit les instructions (parfois complexes) du programme en une succession d'opérations élémentaires, exécutées ensuite par les unités de calcul (UAL et unité de calcul en virgule flottante). Il interagit aussi éventuellement avec l'utilisateur, suivant les instructions du programme. Nous étudierons un peu plus loin le processeur de façon un peu plus précise, sans pour autant entrer dans les détails logiques associés aux traductions et aux exécutions.
- Le transfert des données (les flèches dans le schéma de la figure 6.1) se fait à l'aide de fils électriques transportant des impulsions électriques, appelés **bus**.
  - Un bus est caractérisé :
    - par le nombre d'impulsions électriques (appelées **bits**) qu'il peut transmettre simultanément. Ce nombre dépend du nombre de conducteurs électriques parallèles dont est constitué le bus. Ainsi, un bus de 32 bits est constitués de 32 fils conducteurs pouvant transmettre indépendamment des impulsions électriques.
    - par la fréquence des signaux, c'est-à-dire le nombre de signaux qu'il peut transmettre de façon successive dans un temps donné. Ainsi, un bus de 25 MHz peut transmettre 25 millions d'impulsions sur chacun de ses fils chaque seconde.

    Ainsi, un bus de 32 bits et 25 MHz peut transmettre $25 \cdot 10^6 \cdot 32$ bits par seconde, soit $800 \cdot 10^6$ bit par seconde, soit environ 100 Mo (mégaoctet) par seconde (un octet étant constitué de 8 bit). Le « environ » se justifie par le fait que les préfixes *kilo* et *méga* ne correspondent pas tout-à-fait à 10^3 et 10^6 dans ce cadre, mais à $2^{10} = 1024$ et $2^{20} = 1024^2$.
  - Les bus peuvent donc transmettre les données à condition que celles-ci soient codées dans un système adapté à ces impulsions électriques. La base 2 convient bien ici (1 = une impulsion électrique, 0 = pas d'impulsion électrique). Ainsi, toutes les données sont codées en base 2, sous forme d'une succession de 0 et de 1 (les bits). Ces bits sont souvent groupés par paquets de 8 (un octet). Chaque demi-octet (4 bits) correspond à un nombre allant de 0 à 15, écrit en base 2. Ainsi, pour une meilleure concision et une meilleure lisibilité, les informaticiens amenés à manipuler directement ce langage binaire le traduisent souvent en base 16 (système hexadécimal, utilisant les 10 chiffres, et les lettres de a à f). Chaque octet est alors codé par 2 caractères en hexadécimal.
  - Les bus se répartissent en 2 types : les *bus parallèles* constitués de plusieurs fils conducteurs, et permettant de transmettre un ou plusieurs octets en une fois ; et les *bus séries*, constitués d'un seul conducteur : l'information est transmise bit par bit.

    Paradoxalement, il est parfois plus intéressant d'utiliser des bus séries. En effet, puisqu'un bus série utilise moins de conducteur qu'un bus parallèle, on peut choisir, pour un même prix, un conducteur de bien meilleure qualité. On obtient alors, au même coût, des bus séries pouvant atteindre des débits égaux, voire supérieurs, à des bus parallèles.
  - Un ordinateur utilise des bus à 3 usages essentiellement :
    - le bus d'adresses, dont la vocation est l'adressage en mémoire (trouver un endroit en mémoire). C'est un bus unidirectionnel ;
    - les bus de données, permettant la transmission des données entre les différents composants.

Ce sont des bus bidirectionnels ;
— les bus de contrôle, indiquant la direction de transmission de l'information dans un bus de données.

- La carte-mère est le composant assurant l'interconnexion de tous les autre composants et des périphériques (*via* des ports de connection). Au démarrage, elle lance le BIOS (Basic Input/Output System), en charge de repérer les différents périphériques, de les configurer, puis de lancer le démarrage du système *via* le chargeur d'amorçage (boot loader).

I.2 Mémoires

Nous revenons dans ce paragraphe sur un des composants sans lequel un ordinateur ne pourrait rien faire : la mémoire.
La mémoire est caractérisée :

- par sa taille (nombre d'octets disponibles pour du stockage). Suivant le type de mémoire, cela peut aller de quelques octets à plusieurs Gigaoctets ;
- par sa volatilité ou non, c'est-à-dire le fait d'être effacée ou non en absence d'alimentation électrique.
- par le fait d'être réinscriptible ou non (mémoire morte, mémoire vive).

Nous énumérons ci-dessous différents types de mémoire qu'on peut rencontrer actuellement. Du fait de l'importance de la mémoire et des besoins grandissants en capacité de mémoire, les types de mémoire sont en évolution constante, aussi bien par leur forme que par les techniques ou principes physiques utilisés. Nous ne pouvons en donner qu'une photographie instantanée, et sans doute déjà périmée et loin d'être exhaustive.

- **Mémoire morte (ROM, read-only memory)**
  Il s'agit de mémoire non volatile, donc non reprogammable. Ce qui y est inscrit y est une fois pour toutes, ou presque. Le BIOS, permettant le lancement de l'ordinateur, est sur la ROM de l'ordinateur. Il s'agit plus spécifiquement d'une EPROM (erasable programmable read-only memory). Comme son nom l'indique, une EPROM peut en fait être effacée et reprogrammée, mais cela nécessite une opération bien particulière (elle doit être flashée avec des ultras-violets).
- **Mémoire vive (RAM, random access memory)**
  * Le nom de la RAM provient du fait que contrairement aux autres types de stockages existant à l'époque où ce nom a été fixé (notamment les cassettes magnétiques), la lecture se fait par accès direct (random), et non dans un ordre déterminé. Le nom est maintenant un peu obsolète, la plupart des mémoires, quelles qu'elles soient, fonctionnant sur le principe de l'accès direct.
  * La mémoire vive est une mémoire volatile, utilisée par l'ordinateur pour le traitement des données, lorsqu'il y a nécessité de garder momentanément en mémoire un résultat dont il aura à se resservir plus tard. Elle est d'accès rapide, mais peu volumineuse. Elle se présente généralement sous forme de barrettes à enficher sur la carte-mère.
  * Physiquement, il s'agit de quadrillages de condensateurs, qui peuvent être dans 2 états (chargé = 1, déchargé = 0). Ces condensateurs se déchargent naturellement au fil du temps. Ainsi, pour garder un condensateur chargé, il faut le recharger (rafraîchir) à intervalles réguliers. Il s'agit du cycle de rafraichissement, ayant lieu à des périodes de quelques dizaines de nanosecondes. Par conséquent, en l'absence d'alimentation électrique, tous les condensateurs se déchargent, et la mémoire est effacée.
- **Mémoires de masse**
  Ce sont des mémoires de grande capacité, destinées à conserver de façon durable de grosses données (bases de données, gros programmes, informations diverses...) De par leur vocation, ce sont nécessairement des mémoires non volatiles (on ne veut pas perdre les données lorsqu'on éteint l'ordinateur !). Par le passé, il s'agissait de bandes perforées, puis de cassettes, de disquettes etc. Actuellement, il s'agit plutôt de disques durs, de bandes magnétiques (fréquent pour les sauvegardes régulières), de CD, DVD, ou de mémoires flash (clé USB par exemple).

- **Mémoires flash**
  Les mémoires flash (clé USB par exemple) que nous venons d'évoquer ont un statut un peu particulier. Techniquement parlant, il s'agit de mémoire morte (EEPROM : electrically erasable programmable read-only memory), mais qui peut être flashée beaucoup plus facilement que les EPROM, par un processus purement électrique. Ce flashage fait partie du fonctionnement même de ces mémoires, ce qui permet de les utiliser comme des mémoires réinscritptibles et modifiables à souhait.

Une caractéristique très importante de la mémoire est son temps d'accès, qui représente un facteur limitant du temps de traitement de données. Ce temps d'accès est bien entendu dépendant du type de mémoire, ainsi que de sa position par rapport au processeur : même si elle est très grande, la vitesse de circulation des impulsions électriques n'est pas nulle, et loin d'être négligeable dans la situation présente. Ainsi, les mémoires auquelles on soit accéder souvent (celles utilisées pour des stockages temporaires de résultats intermédiaires par exemple) doivent être physiquement placées près du processeur. Ainsi, plus on construit des mémoires peu volumineuse, plus on peu placer de mémoire près du processeur. La recherche de la miniaturisation n'a donc pas qu'un rôle esthétique. Le processeur lui-même et les composantes auquels il est rattaché subissent le même principe de miniaturisation en vue de l'augmentation de l'efficacité. De ce fait, il y a physiquement peu de place près du processeur.

Par ailleurs, le temps d'accès dépend beaucoup de la technologie utilisée pour cette mémoire. Les mémoires électroniques, composées de circuits bascules (ou circuits bistables), sont rapides d'accès, tandis que les mémoires à base de transistors et de condensateurs (technologie usuelle des barrettes de RAM) sont plus lentes d'accès (temps de charge + temps de latence dû à la nécessité d'un rafraichissement périodique, du fait de la décharge naturelle des condensateurs). Les mémoires externes nécessitant un processus de lecture sont encore plus lentes (disques durs, CD...)

Pour cette raison, les mémoires les plus rapides sont aussi souvent les moins volumineuses, et les plus onéreuses (qualité des matériaux + coût de la miniaturisation), le volume étant d'autant plus limité que d'un point de vue électronique, un circuit bistable est plus volumineux qu'un transistor.

On peut représenter la **hiérarchie des mémoires** sous forme d'un triangle (figure 6.2)

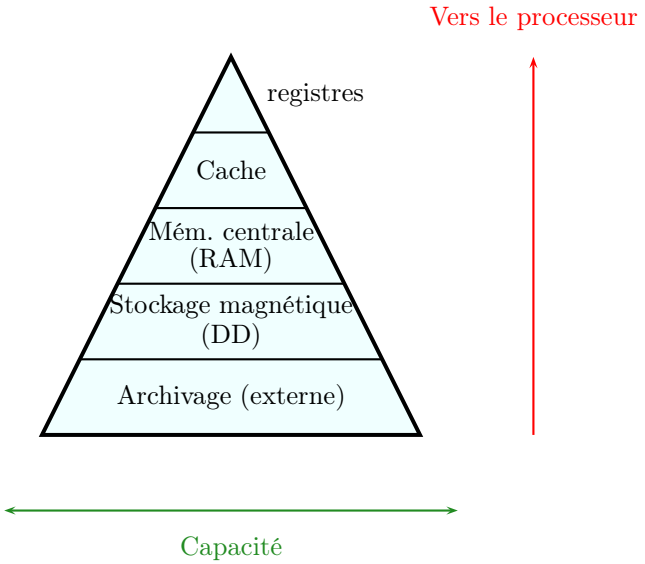


Figure 6.2 – Hiérarchie des mémoires

Les registres sont des mémoires situées dans le processeur, ne dépassant souvent pas une dizaine d'octets, et la plupart du temps sont à usages spécialisés (registres d'entiers, de flottants, d'adresses, compteur ordinal indiquant l'emplacement de la prochaine instruction, registres d'instruction...). Les données stockées dans ces registres sont celles que le processeur est en train d'utiliser, ou sur le point de le faire. Le

temps d'accès est très court. Ces mémoires sont le plus souvent constitués de circuits bascule (voir plus loin).

La mémoire cache se décompose souvent en deux parties, l'une dans la RAM, l'autre sur le disque dur. Pour la première, il s'agit de la partie de la RAM la plus rapide d'accès (SRAM). La mémoire cache est utilisée pour stocker momentanément certaines données provenant d'ailleurs ou venant d'être traitées, en vue de les rapprocher, ou les garder près de l'endroit où elles seront ensuite utiles, afin d'améliorer par la suite le temps d'accès. Ainsi, il arrive qu'avant de parvenir au processeur, les données soient d'abord rapprochées sur la mémoire cache.

I.3 Le processeur (CPU, Central Process Unit)

C'est le cœur de l'ordinateur. C'est lui qui réalise les opérations. Dans ce paragraphe, nous nous contentons d'une description sommaire du principe de fonctionnement d'un processeur. Nous donnerons ultérieurement un bref aperçu de l'agencement électronique (à partir de briques électroniques élémentaires, traduisant sur les signaux électroniques les fonctions booléennes élémentaires) permettant de faire certaines opérations. Nous n'entrerons pas dans le détail électronique d'un processeur.

Composition d'un processeur

Le processeur est le calculateur de l'ordinateur. Il lit les instructions, les traduit en successions d'opérations élémentaires qui sont ensuite effectuées par l'unité arithmétique et logique (UAL), couplée (actuellement) à une unité de calcul en virgules flottantes (format usuellement utilisé pour les calculs sur les réels). Il est constitué de :

- une **unité de traitement**, constituée d'une UAL (unité arithmétique et logique), d'un registre de données (mémoire destinée aux données récentes ou imminentes), et d'un accumulateur (l'espace de travail). Le schéma global est celui de la figure 6.3.

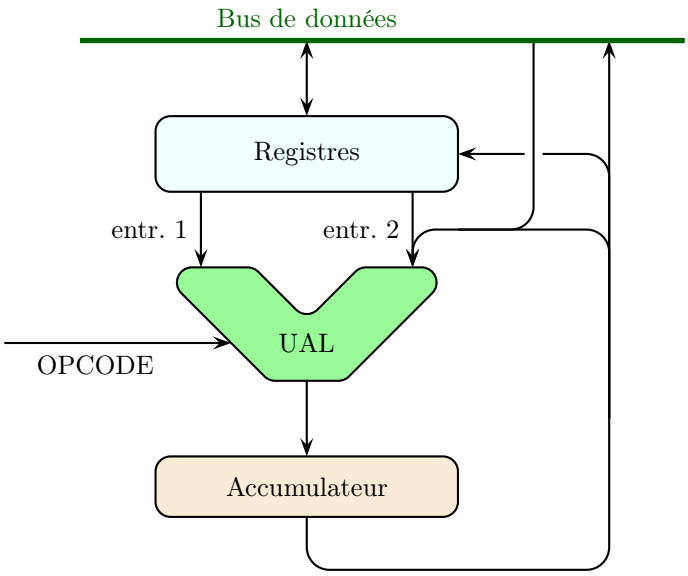


FIGURE 6.3 – Schéma d'une unité de traitement

Ce schéma correspond au cas d'une machine à plusieurs adresses (l'UAL peut traiter simultanément des données situées à plusieurs adresses). Il existe également des machines à une seule adresse : la deuxième donnée nécessaire aux calculs étant alors celle stockée dans l'accumulateur. Cela nécessite de décomposer un peu plus les opérations élémentaires (notamment dissocier le chargement de l'accumulateur du calcul lui-même).

Le code de l'opération (OPCODE) correspond à une succession de bits indiquant la nature de l'opération à effectuer sur l'entrée, ou les entrées (par exemple, un certain code correspond à

l'addition, un autre à la multiplication, d'autres aux différents tests booléens etc.)

Une instruction élémentaire doit donc arriver à l'unité de traitement sous la forme d'un code (une succession de bits, représentés par des 0 et des 1, correspondant en réalité à des impulsions électriques), donnant d'une part le code de l'opération (OPCODE), d'autre part les informations nécessaires pour trouver les entrées auquelles appliquer cette opération (la nature de ces informations diffère suivant le code de l'opération : certains opérations demandent 2 adresses, d'autres une seule adresse et la donnée d'une valeur « immédiate », d'autres encore d'autres formats). Ce codage des instructions diffère également d'un processeur à l'autre. Par exemple, dans un processeur d'architeture MIPS, l'opération est codée sur 6 bits. L'opération 100000 demande ensuite la donnée de 3 adresses a_1, a_2 et a_3. Cette opération consiste en le calcul de $a_2 + a_3$, le résultat étant ensuite stocké à l'adresse a_1 :

100000	a_1	a_2	a_3	$\$a_1 \leftarrow \$a_2 + \$a_3$

Dans cette notation, $\$a$ représente la valeur stockée à l'adresse a. L'opération 001000 est également une opération d'addition, mais ne prenant que deux adresses a_1 et a_2, et fournissant de surcroît une valeur i directement codée dans l'instruction (valeur immédiate). Elle réalise l'opération d'addition de la valeur i à la valeur stockée à l'adresse a_2, et va stocker le résultat à l'adresse a_1 :

001000	a_1	a_2	i	$\$a_1 \leftarrow \$a_2 + i$

L'UAL est actuellement couplée avec une unité de calcul en virgule flottante, permettant le calcul sur les réels (ou plutôt leur représentation approchée informatique, dont nous reparlerons plus loin). Un processeur peut avoir plusieurs processeurs travaillant en parallèle (donc plusieurs UAL), afin d'augmenter la rapidité de traitement de l'ordinateur ;

- une **unité de contrôle**, qui décode les instructions d'un programme, les transcrit en une succession d'instructions élémentaires compréhensibles par l'unité de traitement (suivant le code évoqué précédemment), et envoie de façon bien coordonnée ces instructions à l'unité de traitement. La synchronisation se fait grâce à l'horloge : les signaux d'horloge (impulsions électriques) sont envoyés de façon régulière, et permettent de cadencer les différentes actions tels les coups de timbales dans une galère romaine. De façon schématique, la réception d'un signal d'horloge par un composant marque l'accomplissement d'une étape de la tâche qu'il a à faire. Les différentes étapes se succèdent donc au rythme des signaux d'horloge. Une unité de contrôle peut être schématisée à la manière de la figure 6.4.
  Le registre d'instruction contient l'instruction (non élémentaire) en cours, le compteur ordinal contient ce qu'il faut pour permettre de trouver l'adresse de l'instruction suivante, et le registre d'adresse contient l'adresse de l'instruction suivante.

  Le décodeur décode l'instruction contenue dans le registre d'instruction et la traduit en une succession d'instructions élémentaires envoyées à l'unité de traitement au rythme de l'horloge. Lorsque l'instruction est entièrement traitée, l'unité va chercher l'instruction suivante, dont l'adresse est stockée dans le registre d'adresse, et la stocke dans le registre d'instruction. Le décodeur actualise le compteur ordinal puis l'adresse de l'instruction suivante.

Décomposition de l'exécution d'une instruction

On peut schématiquement décomposer l'exécution d'une instruction par l'unité de traitement en 4 étapes :

- la réception de l'instruction codée (provenant de l'unité de contrôle) (FETCH)
- Le décodage de cette instruction ; les différentes données sont envoyées aux différents composants concerné (en particulier l'OPCODE est extrait et envoyé à l'UAL)
- L'exécution de l'instruction, correspondant au fonctionnement de l'UAL (EXECUTE)
- l'écriture du résultat dans une mémoire (WRITEBACK)

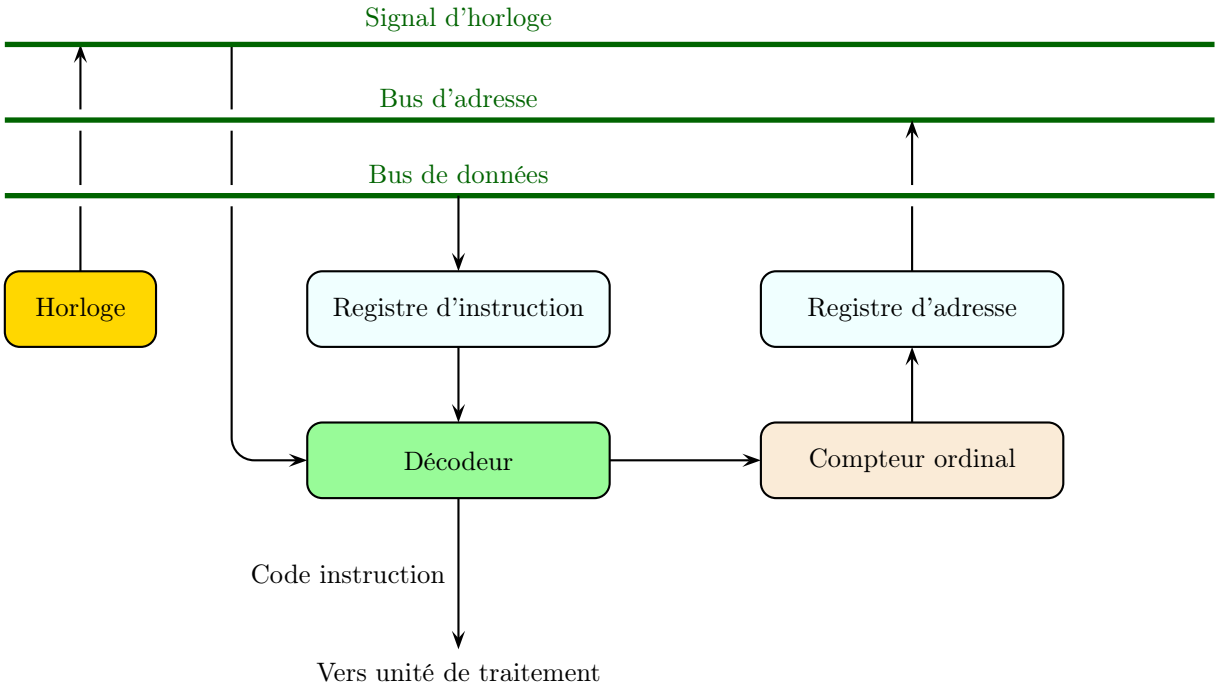


FIGURE 6.4 – Schéma d'une unité de contrôle

Si nous avons 4 instructions à exécuter successivement, cela nous donne donc 16 étapes successives :

Étape	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Instr. 1	FE	DE	EX	WR												
Instr. 2					FE	DE	EX	WB								
Instr. 3									FE	DE	EX	WB				
Instr. 4													FE	DE	EX	WB

On se rend compte que l'UAL n'est pas utilisée à son plein régime : elle n'est solicitée qu'aux étapes 3, 7, 11 et 15 (EXECUTE). En supposant les instructions indépendantes les unes des autres (donc ne nécessitant pas d'attendre le résultat de la précédente pour pouvoir être exécutée), et en négligeant les dépendances éventuelles liées aux problèmes d'adresses, on peut imaginer le fonctionnement suivant, dans lequel on n'attend pas la fin de l'instruction précédente pour commencer le traitement de la suivante :

Étape	1	2	3	4	5	6	7
Instr. 1	FE	DE	EX	WR			
Instr. 2		FE	DE	EX	WB		
Instr. 3			FE	DE	EX	WB	
Instr. 4				FE	DE	EX	WB

On dit que le processeur est muni d'un **pipeline** s'il est muni d'un dispositif permettant d'effectuer ces opérations en parallèle de façon décalée. Évidemment, un pipeline doit aussi permettre de gérer les problèmes de dépendance, nécessitant parfois la mise en attente d'une instruction pour attendre le résultat d'une autre. Dans cet exemple, nous avons un pipeline à **4 étages** (c'est le nombre d'étapes dans la décomposition de l'exécution de l'instruction, correspondant aussi au nombre d'instructions maximal pouvant être traitées simultanément. Des décompositions plus fines peuvent permettre d'obtenir des pipelines constitués d'un nombre plus important d'étages (une dizaine).

Par ailleurs, certains processeurs sont munis de plusieurs cœurs (plusieurs unités de traitement), permettant l'exécution simultanée d'instructions. Ainsi, toujours en supposant les 4 instructions indépendantes, pour un processeur muni de 2 cœurs à pipeline à 4 étages, on obtient le tableau suivant :

Étape	1	2	3	4	5
Instr. 1	FE 1	DE 1	EX 1	WR 1	
Instr. 2	FE 2	DE 2	EX 2	WB 2	
Instr. 3		FE 1	DE 1	EX 1	WB 1
Instr. 4		FE 2	DE 2	EX 2	WB 2

On parle dans ce cas d'architecture en parallèle. Le parallélisme demande également un traitement spécifique des dépendances.

Puissance de calcul d'un processeur

La puissance d'un processeur est calculée en FLOPS (Floating Point Operation Per Second). Ainsi, il s'agit du nombre d'opérations qu'il est capable de faire en une seconde sur le format flottant (réels).

Pour donner des ordres de grandeur, voici l'évolution de la puissance sur quelques années références :

- 1964 : 1 mégaFLOPS (10^6)
- 1997 : 1 téraFLOPS (10^{12})
- 2008 : 1 pétaFLOPS (10^{15})
- 2013 : 30 pétaFLOPS

On estime que l'exaFLOPS (10^{18}) devrait être atteint vers 2020.

Évidemment, ces puissances correspondent aux super-calculateurs. La puissance des ordinateurs personnels suit une courbe bien inférieure. Par exemple, en 2013, un ordinateur personnel était muni en moyenne d'un processeur de 200 gigaFLOPS, ce qui est comparable à la puissance des super-calculateurs de 1995.

II Circuits logiques

Du fait de la circulation de l'information sous forme d'impulsions électiques, comme nous l'avons dit plus haut, la base de représentation privilégiée est la base 2. Mais comment en pratique effectuer les calculs élémentaires sur ces représentations en base 2 ? Autrement dit, comment, à partir d'impulsions électriques correspondant à deux arguments, récupérer les impulsions électriques correspondant, par exemple, à la somme ?

Le but de cette section est d'apporter quelques éléments de réponse, basés sur l'étude des portes logiques, sans entrer dans le détail électronique de ces portes, et en se contentant d'un exemple d'opération, le cas de l'addition.

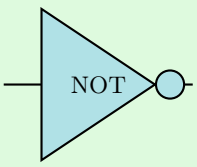
II.1 Portes logiques

Nous admettons l'existence de dispositifs électriques simples permettant de réaliser les opérations suivantes :

Définition 6.2.1 (Les portes logiques)

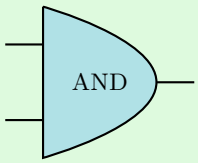
Dans les diagrammes ci-dessus, les entrées sont représentées à gauche, la sortie à droite ;

1. La porte NOT (NON) :



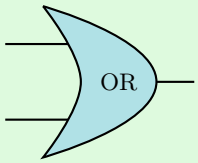
   La sortie vaut 1 si et seulement si l'entrée vaut 0.

2. La porte AND (ET) :



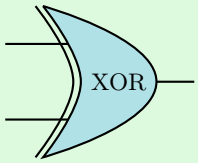
La sortie vaut 1 si et seulement si les deux entrées valent 1 simultanément.

3. La porte OR (OU non exclusif) :



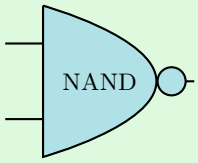
La sortie vaut 1 si et seulement si l'une au moins des deux entrées vaut 1.

4. La porte XOR (OU exclusif) :



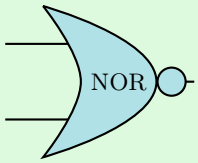
La sortie vaut 1 si et seulement si une et une seule des deux entrées vaut 1.

5. La porte NAND (NON ET) :



La sortie vaut 1 si et seulement si les deux entrées ne sont pas toutes les deux égales à 1, donc l'une au moins vaut 0.

6. La porte NOR (NON OU) :



La sortie vaut 1 si et seulement si aucune des deux entrées ne vaut 1, donc si les deux valent 0.

II.2 Un exemple développé : l'addition sur n bits

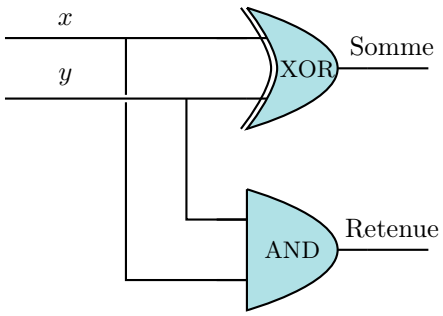
Le point de départ est la remarque suivante : l'addition de deux bits peut facilement se traduire à l'aide des portes logiques. En effet, en base 2, on a, en forçant le résultat à tenir sur 2 bits :

$$0+0=00 \qquad 0+1=1+0=01 \qquad 1+1=10.$$

Ainsi, le chiffre des unités vaut 1 si et seulement si un et un seul des deux bits additionnés est égal à 1 : il s'agit de l'opération booléenne XOR. De même, le chiffre des 2-aines est égal à 1 si et seulement si les deux bits sont tous deux égaux à 1 : il s'agit de l'opération booléenne AND. Ainsi, on obtient le

bit en cours sous forme d'une opération booléenne, ainsi qu'un bit de retenue, sous forme d'une autre opération booléenne, ces deux opérations étant directement données par l'utilisation de portes logiques élémentaires.

Nous obtenons ainsi le schéma d'un *demi-additionneur*, additionnant deux bits x et y, et renvoyant un bit de somme et un bit de retenue :



Nous voyons dès lors comment faire pour sommer des nombres de 2 bits x_1x_0 et y_1y_0, en donnant le résultat sous forme d'un nombre à deux bits z_1z_0, et d'une retenue r_1 (figure 6.5)

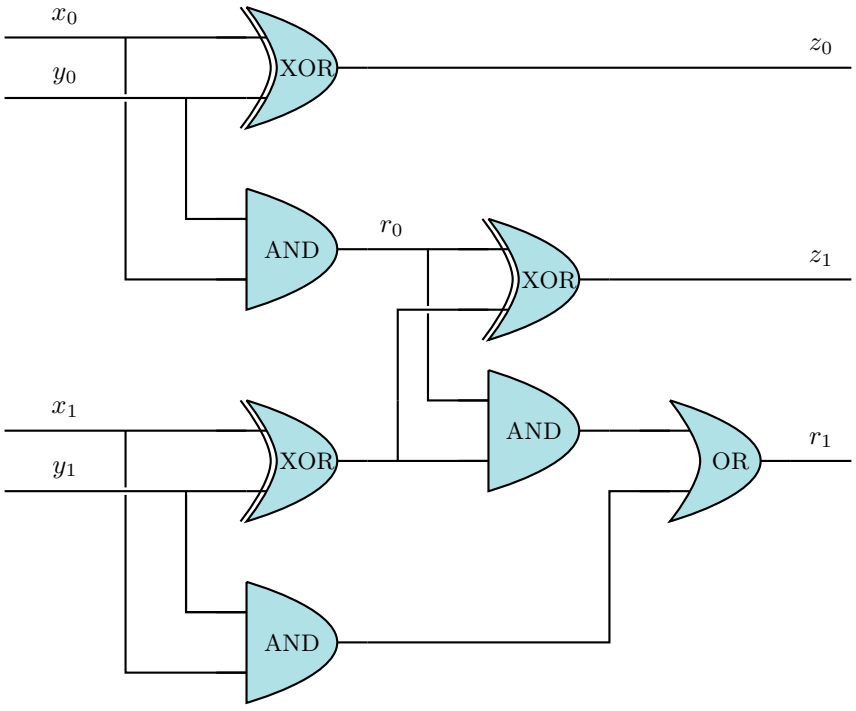


Figure 6.5 – Schéma logique d'un additionneur à 2 bits

On comprend alors comment continuer en branchant en série des demi-additionneurs sur le même principe, de sorte à obtenir un additionneur sur n bits. Remarquez qu'on obtiendra au bout un bit de retenue, qui donner la validité du résultat : si le résultat ne reste pas dans l'intervalle d'entiers considéré (donné par le nombre de bits), la retenue va valoir 1. Si elle vaut 0 en revanche, c'est qu'il n'y a pas de dépassement de capacité.

II.3 Décodeurs d'adresse

Définition 6.2.2 (Décodeur d'adresses)

Un décodeur d'adresses sur n bits est un assemblage électronique à n entrées et 2^n sorties, qui pour un signal d'entrée $(a_0, \cdots, a_{n-1}) \in \{0,1\}^n$ ressort une impulsion uniquement sur la ligne de sortie dont la

numérotation correspond au nombre $\overline{\alpha_{n-1}\dots a_1 a_0}^2$. Typiquement, l'entrée correspond au codage d'une adresse, la sortie correspond à la sélection de cette adresse parmi toutes les adresses possibles.

Nous nous contentons de décrire comment on peut construire des décodeurs d'adresses sur un petit nombre de bits à l'aide de portes logiques.

Décodeur d'adresses à 1 bit

Nous disposons d'une entrée a_0 et de deux sorties s_0 et s_1. La description qui précède amène le tableau de vérité suivant pour le décodeur d'adresse :

Entrée	Sortie	
a_0	s_0	s_1
0	1	0
1	0	1

La sortie s_0 correspond à non-a_0 et s_1 correspond à a_0. On obtient donc le circuit logique 6.6, réalisant le décodeur d'adresses sur 1 bit.

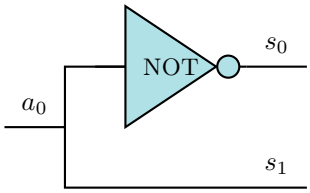


FIGURE 6.6 – Décodeur d'adresses sur 1 bit

Décodeur d'adresses sur 2 bits

La table de vérité du décodeur d'adresses sur 2 bits est :

Entrée		Sortie			
a_0	a_1	s_0	s_1	s_2	s_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

On constate que :

$$s_0 = \neg a_0 \wedge \neg a_1, \qquad s_1 = \neg a_0 \wedge a_1, \qquad s_2 = a_0 \wedge \neg a_1, \qquad s_3 = a_0 \wedge a_1.$$

On obtient alors le circuit du décodeur d'adresses sur 2 bits de la figure 6.7

Décodeur d'adresses sur n bits

De façon plus générale, on duplique chaque entrée, en appiquant une porte NOT à l'une des deux branches, puis on applique une porte AND (à n entrées) à toutes les sélections d'une des deux branches pour chacune des n entrées. Cela selectionne une unique sortie pour chacune des 2^n entrées possibles. En numérotant ces sorties de façon convenable, on obtient le décodeur d'adresse voulu.

Ce circuit nécessite 2^n portes AND à n entrées. Chacune de ces portes est elle-même réalisable à l'aide de $n-1$ portes AND à 2 entrées (appliquées en cascade). On peut gagner en nombre de portes logiques en factorisant les opérations booléennes, ce qu'on illustre par la figure 6.8, pour un décodeur sur 3 bits.

On peut noter qu'on peut obtenir des circuits de plus petite profondeur (nombre de composantes en série), donc plus performants, en utilisant un branchement du type « diviser pour régner » pour réaliser les portes AND à n entrées, mais cela augmente le nombre de composantes nécessaires. Comme souvent, il faut trouver le bon compromis entre coût et performance.

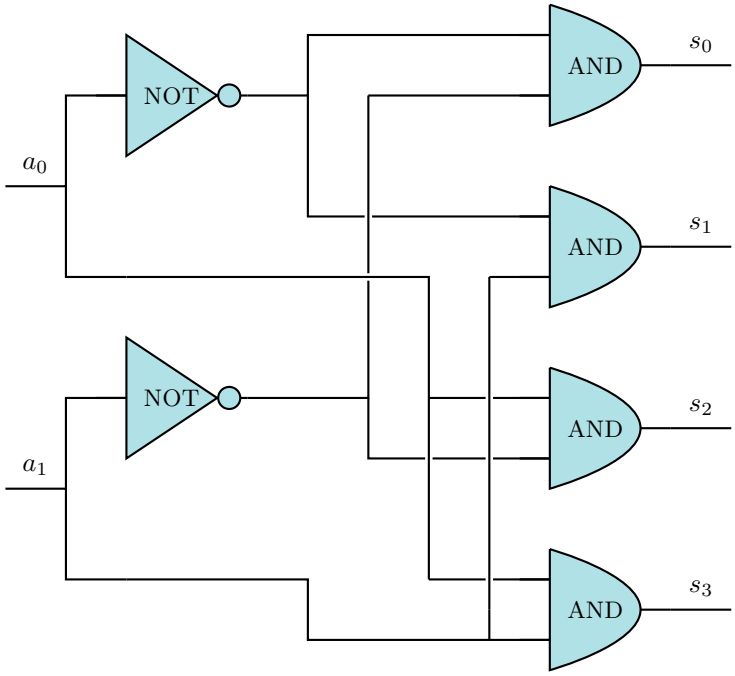


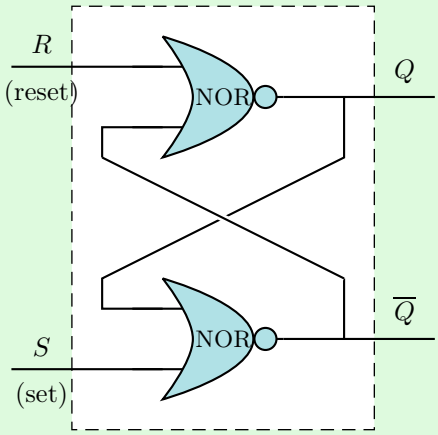
Figure 6.7 – Décodeur d'adresses sur 2 bits

II.4 Circuits bascules et mémoires

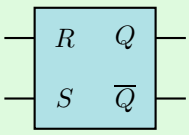
Nous étudions dans ce paragraphe comment les portes logiques peuvent être utilisées en vue de mémoriser de l'information, grâce aux propriétés des circuits-bascule, aussi appelés circuits bistables.
Le schéma élémentaire à la base de ces mémoires est celui de la bascule RS minimale, dans lequel on utilise deux portes AND dont les sorties sont branchées sur une entrée de l'autre.

Définition 6.2.3 (Bascule RS minimale)

Le circuit bascule RS minimale, est le circuit dont le schéma électronique est le suivant :



Il sera noté par le schéma suivant :



Le principe de la bascule est le suivant :

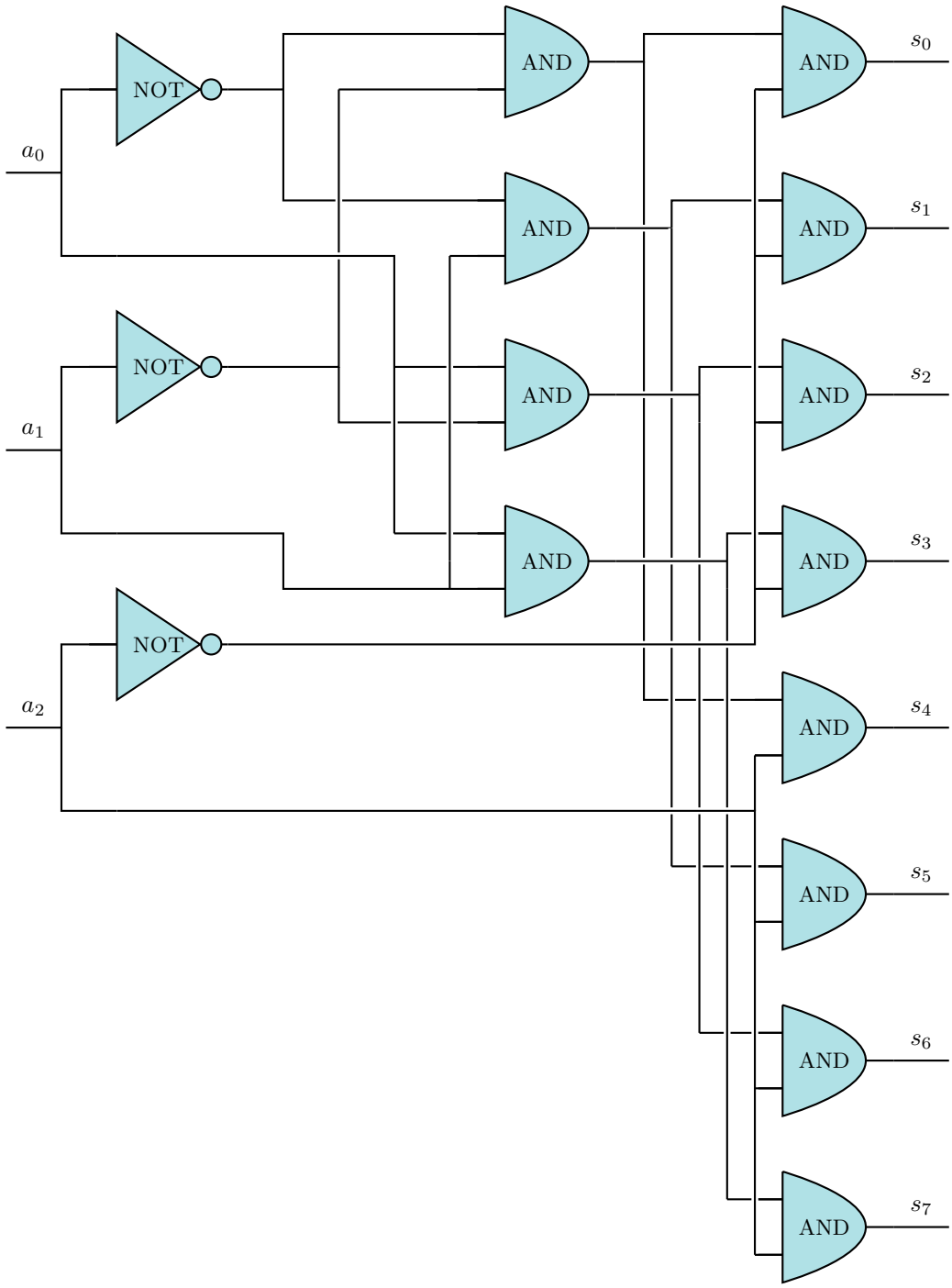


FIGURE 6.8 – Décodeur d'adresses sur 3 bits

- R et S ne peuvent pas prendre simultanément la valeur 1 (valeur interdite)
- Les deux sorties ne peuvent pas être simultanément égales à 0 (cela nécessiterait que les deux entrées soient égales à 1), ni toutes deux égales à 1 (si l'une est égale à 1, par définition d'une porte NOR, la seconde vaut 0). Ainsi, les deux sorties sont complémentaires (Q et $\overline{Q}$).
- Si $R = S = 0$, les deux positions $Q = 0$ et $Q = 1$ sont possibles (stables)
- Si $R = 1$, Q prend nécessairement la valeur 0 (bouton « reset » pour remettre à 0)
- Si $S = 1$, $\overline{Q}$ prend nécessairement la valeur 0, donc Q la valeur 1 (bouton « set », pour mettre la valeur 1)

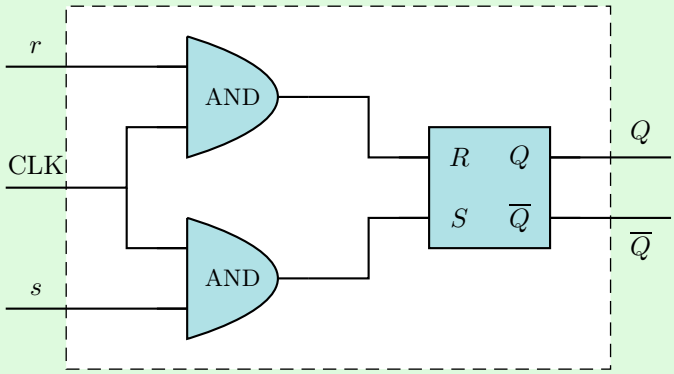
Ainsi, une impulsion sur l'entrée R ou S permet de donner une valeur à Q, puis, en absence d'autre

impulsion, la valeur de Q reste inchangée. On a donc mémorisé une valeur.

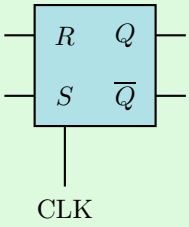
Pour que les mises à jour se fassent uniquement lors des signaux d'horloge, on peut brancher le signal d'horloge sur chacune des 2 entrées avec une porte AND. Ainsi, la bascule ne peut pas recevoir de signal en l'absence d'un signal d'horloge :

Définition 6.2.4 (Bascule synchrone)

La bascule synchrone est donnée par le schéma suivant :



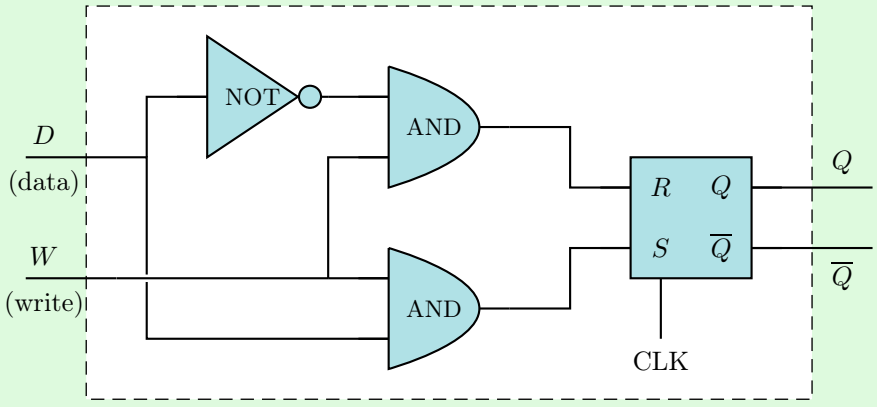
On notera une bascule synchrone par le schéma :



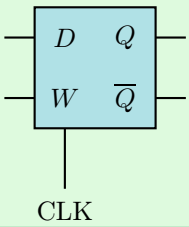
La mémorisation d'un bit D se fait alors à l'aide du schéma suivant :

Définition 6.2.5 (Bascule D)

Une bascule D est un dispositif électronique décrit par le schéma suivant :



Un tel dispositif sera noté :



Le fonctionnement de la bascule D est alors le suivant :

- En l'absence d'un signal W (write), les deux entrées de la bascule RS sont nulles, donc la sortie Q reste inchangée.
- En présence d'un signal W, les entrées de la bascule RS sont respectivement non-D et D. On vérifie sans peine qu'à tout signal d'horloge, la sortie Q sera égale à l'entrée D.

Ainsi :

Proposition 6.2.6 (Fonctionnement d'une bascule D)

Envoyer une donnée D accompagnée de l'instruction d'écriture W enregistre la donnée D dans Q. Cette donnée ne sera pas modifiée tant qu'il n'y aura pas d'autre signal d'écriture.

Nous donnons en figure 6.9 le schéma des circuits-mémoire (registres), basés sur l'utilisation des bascules D.

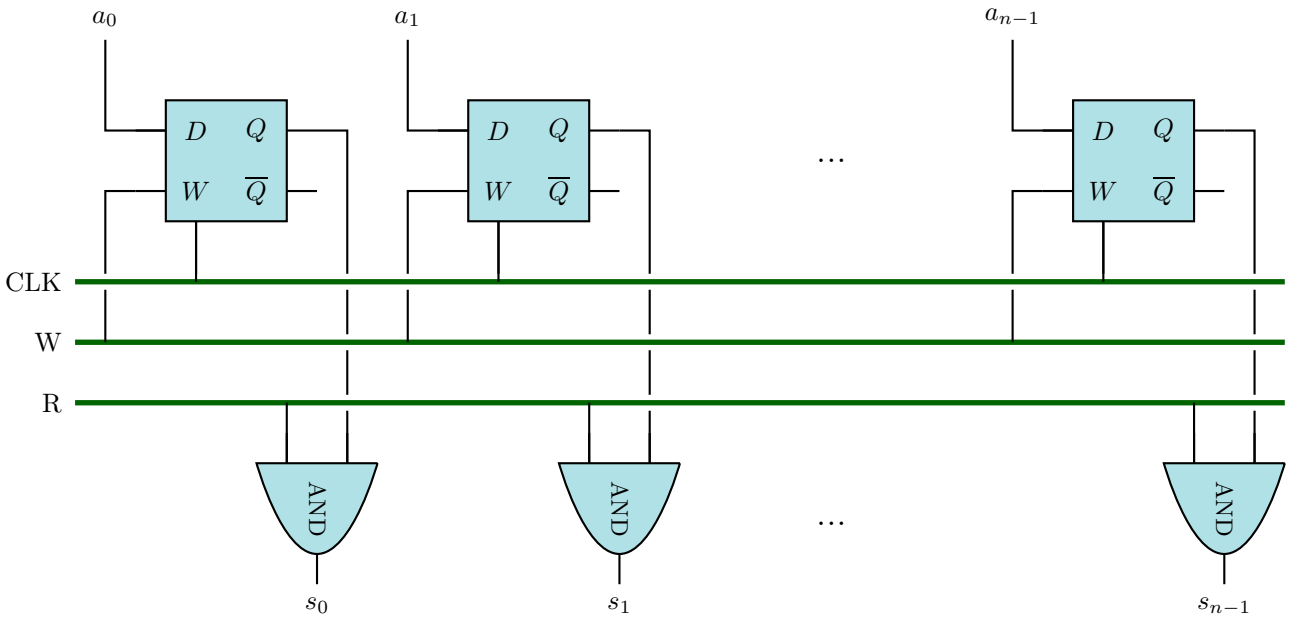


FIGURE 6.9 – Schéma d'un registre à n bits

Lorsque le signal d'écriture W est donné, de façon synchrone, les bits $a_0, \ldots, a_{n-1}$ sont enregistrés par les n bascules D, jusqu'au prochain signal d'écriture. La ligne R (lecture) permet en cas de signal sur cette ligne (signal de lecture) de sortir en $s_0, \ldots, s_{n-1}$ les valeurs mémorisées par les bascules D lors du dernier enregistrement (sans signal de lecture, ces sorties sont nulles).
Ainsi, W est la ligne commandant l'écriture, R est la ligne commandant la lecture des données.

III Systèmes d'exploitation

III.1 Qu'est-ce qu'un système d'exploitation ?

Définition 6.3.1 (Système d'exploitation)

Un système d'exploitation est un ensemble de programmes permettant de gérer de façon conviviale et efficace les ressources de l'ordinateur. Il réalise un pont entre l'utilisateur et le processeur.

En particulier, le système d'exploitation :

- fournit une interface la plus conviviale possible, permettant le lancement d'applications diverses par l'utilisateur ;
- s'occupe de l'initialisation de l'ordinateur, ainsi que de la communication avec les périphériques ;
- propose une interface simple et intuitive pour le rangement des données (structure arborescente du système de fichier) : l'utilisateur n'a pas à se soucier lui-même de la localisation physique de ses fichiers sur l'espace mémoire, il les retrouve dans l'arborescence, en général logique et thématique, qu'il a lui-même élaborée ;
- gère les ressources de l'ordinateur.

En particulier, les systèmes d'exploitation actuels (parmi les plus connus, citons Linux, Windows, MacOS...) permettent l'exécution simultanée de plusieurs programmes, par le même utilisateur, ou même par différents utilisateurs. Chaque utilisateur doit dans ce cas avoir la sensation d'avoir l'ordinateur pour lui seul (machine virtuelle), ce qui nécessite une répartition des tâches assez élaborée.

Définition 6.3.2 (processus)

Un processus est une activité de l'ordinateur résultant de l'exécution d'un programme.

On peut imaginer plusieurs modes de fonctionnement :

- Les processus accèdent au processeur dans l'ordre de leur création, et utilisent ses ressources jusqu'à leur terminaison. Ainsi, les programmes sont exécutés les uns après les autres. Le temps de réponse dépendra beaucoup plus de la file d'attente que du temps d'exécution du programme lancé. Cela défavorise nettement les processus courts. On peut arranger un peu le système en donnant une priorité plus forte aux processus courts, mais malgré tout, ce système n'est pas satisfaisant.
- Méthode du tourniquet (ou balayage cyclique) : le processeur tourne entre les différents processus en cours. Chaque processus accède au processeur pour un temps donné, fixé à l'avance (le quantum), et au bout de ce temps, cède la place au processus suivant, même s'il n'est pas achevé. Il accèdera de nouveau au processeur lorsque celui-ci aura fait le tour de tous les processus actifs. Certains processus peuvent être mis en attente s'ils ont besoin de données d'autres processus pour continuer.
- Méthode du tourniquet multiniveau : c'est une amélioration du système précédent. On peut attribuer un niveau de priorité aux différents processus. Le processeur s'occupe d'abord des processus de priorité la plus élevée, par la méthode du tourniquet, puis descend petit à petit les priorités. Un processus n'accède au processeur que s'il n'y a plus de processus en cours de niveau de priorité plus élevé.

C'est au système d'exploitation de gérer ce découpage des processus de sorte à ce que du point de vue de l'utilisateur, tout se passe comme s'il était seul, ou s'il ne lançait qu'une tâche (sur un ordinateur un peu plus lent). Évidemment, le quantum doit être assez petit pour que l'utilisation d'un logiciel semble continue à l'utilisateur, même si en réalité son exécution est entrecoupée un grand nombre de fois. On ne doit pas ressentir ce hâchage lors de l'utilisation.

Note Historique 6.3.3

Trois grands systèmes d'exploitation se partagent le marché grand public actuellement : Windows de Microsoft, Linux, dérivé de Unix, et MacOS de Apple, également construit sur un noyau Unix. Windows est réputé plus convivial que Linux dans sa présentation (interfaces sous forme de fenêtres, menus déroulants etc), l'utilisation de Linux se faisant davantage à l'aide de Shells (terminaux d'invites de lignes de commandes) : on y tape des instructions directement en ligne de commande plutôt qu'en utilisant la souris et des interfaces graphiques.
Les deux points de vue se défendent, mais dans des contextes différents. Pour un usage tout public et non professionnel, une interface graphique plaisante est souvent plus intuitive. Pour des utilisations plus poussées, la possibilité d'utiliser un terminal de commandes est plus efficace (rapidité de manipulation, possibilité de scripts élaborés etc.), mais nécessite un apprentissage et une habitude accrus. Il faut noter d'ailleurs qu'Unix (à l'origine de Linux) a été développé à des fins professionnelles et industrielles, puis les distributions de Linux se sont petit à petit adaptées au grand public par l'ajout d'interfaces graphiques, mais restent tout de même

tournées vers une activité professionnelle ; à l'inverse, Windows a été développé pour les ordinateurs personnels, donc pour un usage non professionnel : sa priorité est la convivialité, de sorte à rendre l'utilisation de l'ordinateur accessible à tous. Ce n'est que par la suite que Microsoft a adapté Windows à des utilisations professionnelles en entreprise. Ces deux grands systèmes d'exploitation ont donc eu une évolution opposée qui explique un peu leur philosophie de la présentation, même si les différences de convivialité et de souplesse d'utilisation tendent à s'amenuiser au fil des versions.

III.2 Arborescence des fichiers

La gestion de la mémoire est un des autres grands rôles du système d'exploitation, aussi bien la mémoire vive que les mémoires de stockage. Lors de l'enregistrement d'un fichier, il recherche une place disponible sur l'espace mémoire de stockage (disque dur par exemple). Au fur et à mesure des enregistrements et effacements, ces espaces disponibles de stockage peuvent être de moins en moins pratiques et de plus en plus fragmentés, ce qui nécessite parfois d'enregistrer un même fichier en plusieurs endroits non contigus (fragmentation). Plus le système est obligé de fragmenter les fichiers, plus l'utilisation de ces fichiers devient lente. C'est pourquoi il faut parfois réorganiser l'ensemble du disque dur, en déplaçant un grand nombre de données, pour regrouper ce qui va ensemble (défragmentation). Par exemple, Windows lance automatiquement la défragmentation du disque dur lorsque la situation devient critique.

Une telle gestion de la mémoire, où on range les données là où on trouve de la place n'est pas compatible avec une utilisation humaine directe et efficace. On ne peut pas demander à l'utilisateur de se souvenir que tel fichier qui l'intéresse est coupé en 3 morceaux rangés aux adresses $1552fab1$, $5ce42663$ et $4ceb5552$. L'utilisateur aurait tôt fait d'être complètement perdu.

Ainsi, le système d'exploitation propose à l'utilisateur une interface facile : l'utilisateur ordonne son espace mémoire à sa convenance sous la forme d'une arborescence de dossiers (répertoires) dans lesquels il peut ranger des fichiers (figure 6.10). Cette arborescence doit être construite de façon logique, et correspond au rangement thématique qu'effectuerait l'utilisateur dans une armoire. Chaque noeud correspond à un dossier, chaque feuille correspond à un fichier (ou un dossier vide). Le contenu d'un dossier est l'ensemble de ses fils et de leur contenu. Un dossier peut contenir simultanément des fichiers et d'autres dossiers.

L'utilisateur peut se déplacer facilement dans son arborescence, et visualiser son contenu. Il n'a pas pour cela à se soucier des adresses physiques sur le disque, c'est le système d'exploitation qui se charge de cela. D'ailleurs, la position d'un fichier dans l'arborescence n'a pas d'incidence sur l'adresse physique sur le disque : lorsqu'on déplace un fichier dans l'arborescence, on ne fait que modifier certains pointeurs d'adresse dans la description de l'arborescence, mais le fichier lui-même ne change pas de place physique sur le disque. Ainsi, déplacer un gros fichier est tout aussi rapide que déplacer un petit fichier, et quasi-instantané. Ce n'est évidemment pas le cas en cas de déplacement vers un autre support, ou en cas de copie.

III.3 Droits d'accès

Avec l'évolution des réseaux locaux et d'internet, la protection des fichiers est importante : il faut pouvoir vous assurer que personne ayant accès au contenu de votre poste de travail (par exemple dans un réseau d'entreprise) ne pourra ouvrir ou exécuter vos fichiers sans votre autorisation. Pour cette raison, à chaque fichier et chaque dossier sont associés des droits d'accès. Ces droits se classent en 3 types :

- droit de lecture (`r`) ;
- droit d'écriture (`w`) : un fichier en droit d'écriture peut être modifié. Attention, il est possible de modifier des fichiers sans les ouvrir en lecture ;
- droit d'exécution (`x`).

Ces droits se comprennent intuitivement pour les fichiers. Pour un répertoire, le droit de lecture permet de lister son contenu, le droit d'exécution permet d'accéder à ce répertoire, le droit d'écriture permet d'y ajouter ou d'en supprimer des fichiers.

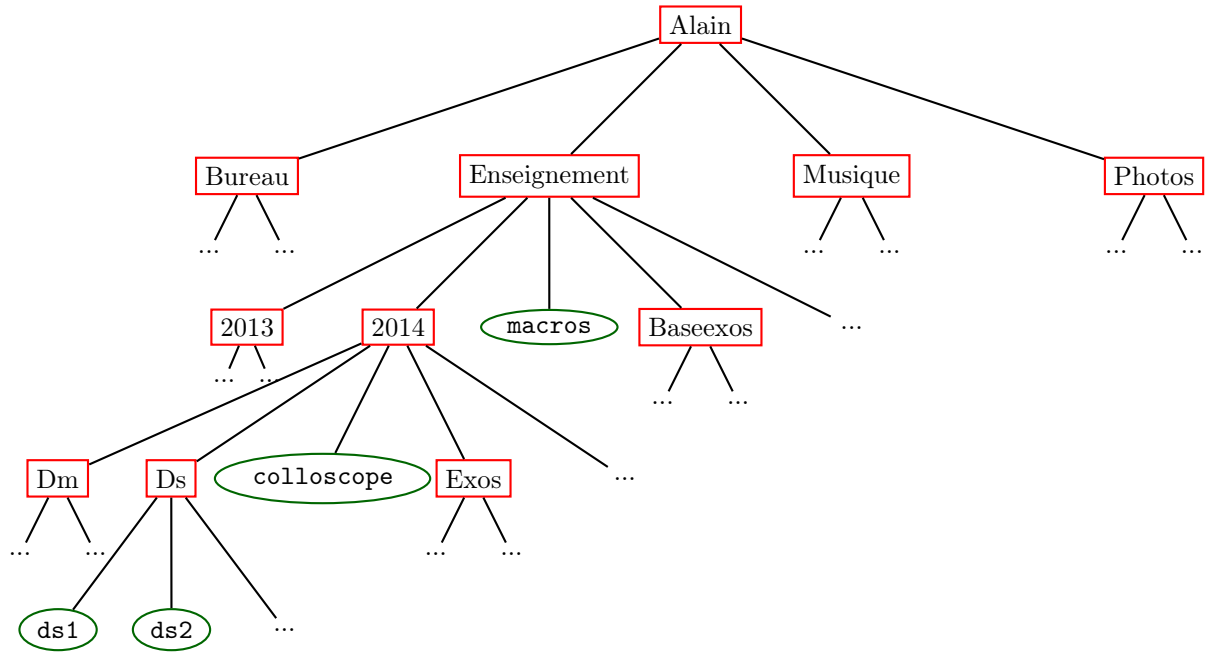


Figure 6.10 – Exemple d'arborescence de fichiers (simplifiée !)

La gestion des droits diffère selon les systèmes d'exploitation. Sous Linux par exemple, on peut visualiser les droits associés à un fichier par la commande `ls -l`, qui liste le contenu d'un répertoire et donne les propriétés essentielles de chaque fichier. À chaque fichier ou répertoire est associé un code du type `rwxrwxrwx`, certaines lettres pouvant être remplacées par un `-`, l'ensemble étant précédé d'une lettre `d` (pour *directory* si c'est un répertoire), d'une lettre `s` (si c'est un lien symbolique), ou d'un `-` (si c'est un fichier). Ce code est à comprendre de la sorte :

- Les lettres `rwx` représentent les droits en lecture, écriture et exécution respectivement.
- Si le droit est accordé, la lettre est apparente, sinon, elle est remplacée par `-`
- Le premier groupe `rwx` donne les droits pour l'utilisateur `u` (le propriétaire du fichier)
- Le second groupe `rwx` donne les droits pour le groupe `g` (défini par l'ingénieur système : une entreprise peut se diviser en plusieurs entités ; le groupe est l'entité dans laquelle se trouve le propriétaire du fichier)
- Le troisième groupe `rwx` donne les droits pour les autres `o`.
- `root` possède tous les droits.

Le propriétaire d'un fichier peut modifier les droits de ses fichiers. Pour des raisons historiques, ces modifications sont moins accessibles sous Windows que sous Linux, mais restent possibles. Sous Linux, il suffit d'utiliser la commande `chmod`, puis préciser en paramètre la ou les lettre(s) associée(s) au(x) groupe(s) et les lettres associées aux droits qu'on veut ajouter (avec `+`) ou supprimer (avec `-`). Par exemple :

```
chmod go-r exemple.txt
```

supprime les droits de lecture pour les membres du groupe et pour les autres utilisateurs.

La fonction `chmod` peut aussi s'utiliser en transcrivant les droits en binaire : les droits de chaque groupe fournissent un nombre binaire de 3 chiffres (0 pour un droit non accordé, 1 pour un droit accordé). Il s'agit donc d'un nombre entre 0 et 7. Ainsi, la donnée d'un nombre à 3 chiffres compris entre 0 et 7 (donc en base 8) suffit à décrire l'ensemble des droits. On peut utiliser ce nombre directement en paramètre de `chmod` pour redéfinir entièrement les droits. Par exemple

```
chmod 640 exemple.txt
```

attribuera les droits `rw-r-----`.

IV Langages de programmation

IV.1 Qu'est-ce qu'un langage de programmation ?

Le langage machine, c'est-à-dire le code binaire compris par le processeur, est assez peu accessible au commun des mortels. S'il fallait communiquer avec le processeur directement ainsi, seule une élite très restreinte pourrait faire de la programmation, d'autant plus que chaque processeur a son propre langage machine.

Un langage de programmation est un langage qui se place généralement à un niveau plus compréhensible par l'humain lambda, permettant, dans une syntaxe stricte, de décrire un algorithme par la donnée d'une succession d'instructions (des ordres). Cette succession d'instructions sera ensuite traduite en langage machine par un programme spécifique (le compilateur ou l'interpréteur).

Ainsi, un langage de programmation est à voir comme un langage intermédiaire permettant de communiquer indirectement avec le cœur opératoire de l'ordinateur.

Nous utiliserons cette année le langage de programmation `Python`, dans sa troisième version.

IV.2 Niveau de langage

Comme vous le savez, il existe un grand nombre de langages de programmation. Pourquoi une telle variété ? Pourquoi certaines personnes ne jurent-elles que par un langage en particulier ? Y a-t-il réellement des différences fondamentales entre les différents langages ?

La réponse est OUI, de plusieurs points de vue. Nous abordons 3 de ces points de vue dans les 3 paragraphes qui viennent.

Le premier point de vue est celui du **niveau (d'abstraction) de langage**. Il s'agit essentiellement de savoir à quel point le langage de programmation s'éloigne du langage machine et des contraintes organisationnelles de la mémoire et des périphériques qui lui sont liées pour s'approcher du langage humain et s'affranchir de la gestion de l'accessoire pour se concentrer sur l'aspect algorithmique :

- Un **langage de bas niveau** est un langage restant proche des contraintes de la machine ; le but est davantage la recherche de l'efficacité par une gestion pensée et raisonnée des ressources (mémoire, périphérique...), au détriment du confort de programmation. Parmi les langages de bas niveau, on peut citer Assembleur ou C.
- Un **langage de haut niveau** est un langage s'approchant davantage du langage humain, et dégageant la programmation de toutes les contraintes matérielles qui sont gérées automatiquement (mémoire, périphériques...). L'intérêt principal est un confort de programmation et la possibilité de se concentrer sur l'aspect algorithmique. En revanche, on y perd nécessairement en efficacité : la gestion automatique des ressources ne permet pas de gérer de façon efficace toutes les situations particulières.

Ainsi, dans la vie pratique, les langages de haut niveau sont largement suffisants en général. C'est le cas en particulier pour la plupart des applications mathématiques, physiques ou techniques, pour lesquelles le temps de réponse est suffisamment court, et qui sont amenées à être utilisées à l'unité et non de façon répétée.
Évidemment, dès que l'enjeu de la rapidité intervient (calculs longs, par exemple dans les problèmes de cryptographie, ou encore besoin de réactivité d'un système, donc pour tout ce qui concerne la programmation liée aux systèmes d'exploitations, embarqués ou non), il est préférable d'adopter un langage de plus bas niveau.

Il est fréquent dans des contextes industriels liés à l'efficacité de systèmes embarqués, que le prototypage se fasse sur un langage de haut niveau (travail des algorithmiciens), puis soit traduit en un langage de bas niveau (travail des programmeurs), souvent en C.

Python est un langage de très haut niveau d'abstration. Sa philosophie est de dégager l'utilisateur de toute containte matérielle, et même de se placer à un niveau où la plupart des algorithmes classiques (tri, recherche de motifs, algorithmes de calcul numérique...) sont déjà implémentés. Ainsi, Python propose un certain nombre de modules complémentaires facultatifs, proposant chacun un certain nombre d'outils algorithmiques dans un domaine précis. Python se place donc délibérément à un niveau où une grande partie de la technique est cachée.

Cela fournit un grand confort et une grande facilité de programmation, mais une maîtrise moins parfaite des coûts des algorithmes utilisant des fonctions prédéfinies.

IV.3 Interprétation et compilation

Une autre grande différence entre les langages de programmation est la façon dont ils sont traduits en langage machine. C'est lors de cette traduction qu'est rajoutée toute la gestion de la basse-besogne dont on s'était dispensé pour un programme de haut-niveau. Ainsi, cette traduction est d'autant plus complexe que le langage est de haut-niveau.

Il existe essentiellement deux types de traduction :

- **La compilation.** Elle consiste à traduire entièrement un langage de haut niveau en un langage de bas niveau (souvent en Assembleur), ou directement en langage machine (mais cela crée des problèmes de portabilité d'une machine à une autre). Le programme chargé de faire cette traduction s'appelle le *compilateur*, et est fourni avec le langage de programmation. Le compilateur prend en argument le code initial, et retourne en sortie un nouveau fichier (le programme compilé), directement exploitable (ou presque) par l'ordinateur.
  - Avantages : une fois la compilation effectuée, le traitement par l'ordinateur est plus rapide (il repart de la source compilée). C'est donc intéressant pour un programme finalisé, voué à être utilisé souvent.
  - Inconvénients : Lors du développement, la compilation peut être lente, et nécessite que le programme soit syntaxiquement complet : elle nécessite donc de programmer étape entière par étape entière, sans pouvoir faire de vérifications intermédiaires.
- **L'interprétation.** Contrairement à la compilation, il ne s'agit pas de la conversion en un autre programme, mais d'une exécution dynamique. L'exécution est effectuée directement à partir du fichier source : l'interpréteur lit les instructions les unes après les autres, et envoie au fur et à mesure sa traduction au processeur.
  - Avantages : il est inutile d'avoir un programme complet pour commencer à l'exécuter et voir son comportement. Par ailleurs, les environnements de programmation associés à ces langages permettent souvent la localisation dynamique des erreurs de syntaxes (le script est interprété et validé au moment-même où il est écrit). Enfin, l'exécution dynamique est compatible avec une exécution en console, instruction par instruction. C'est un atout majeur, notamment pour vérifier la syntaxe d'utilisation et le comportement d'une instruction précise. De plus, les essais effectués lors de l'élaboration du programme sont plus rapides (ne nécessitent pas la lourdeur d'une compilation à chaque nouvel essai).
  - Inconvénients : Le programme finalisé est plus lent à l'exécution, puisque la traduction part toujours du code initial, lointain du programme machine.

Python est un langage interprété, semi-compilé. On peut ainsi bénéficier d'une utilisation en console, ainsi que, suivant les environnements, d'une détection dynamique des erreurs de syntaxe. En revanche, l'exécution du programme commence par une compilation partielle, nécessitant une syntaxe complète.

IV.4 Paradigmes de programmation

Un paradigme de programmation est un style de programmation déterminant de quelle manière et sous quelle forme le programmeur manipule des données et donne des instructions. Il s'agit en quelque sorte

de la philosophie du langage. Il existe un grand nombre de paradigmes de programmation. Parmi les plus utilisés, citons les suivants :

- **La programmation impérative.** Le programme consiste en une succession d'instructions. L'exécution d'une instruction a pour conséquence une modification de l'état de l'ordinateur. L'état final de l'ordinateur fournit le résultat voulu.
- **La programmation orientée objet.** On agit sur des objets (des structures de données). Les objets réagissent à des messages extérieurs au moyen de méthodes. Ainsi, le gros de la programmation porte sur la définition de méthodes associées à des classes d'objets.
- **La programmation fonctionnelle.** On ne s'autorise pas les changements d'état du système. Ainsi, on ne fait aucune affectation. On n'agit pas sur les variables mais on exprime le programme comme un assemblage de fonctions (au sens mathématique).
- **La programmation logique.** C'est un paradigme basé sur les règles de la logique formelle. Il est notamment utilisé pour les démonstrations automatiques, ou pour l'intelligence artificielle.

Python est un langage hybride, adapté à plusieurs paradigmes. Essentiellement cette année, nous utiliserons Python en tant que langage impératif et orienté objet.