Introduction à l'informatique
cours de L1 Miashs, Lille3

I.Tellier

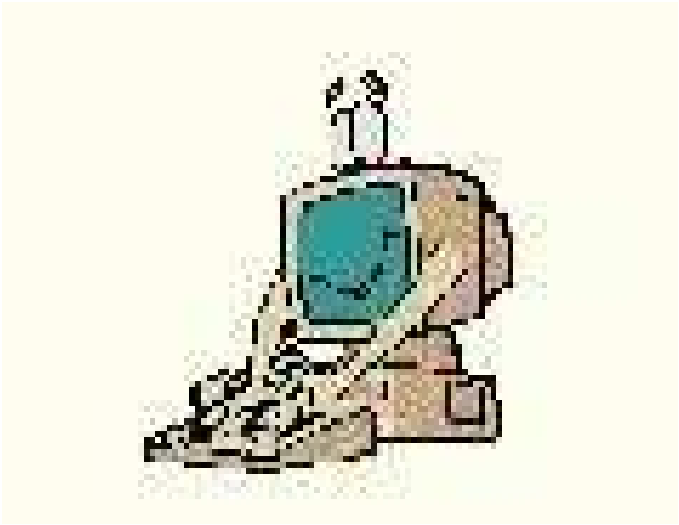


Table des matières

Chapitre 1

Introduction

C'est devenu une banalité : l'ordinateur s'accapare nos bureaux, modifie nos modes de travail, envahit nos maisons, s'intègre dans les objets les plus quotidiens et nous propose des loisirs inédits. Il est même à l'origine de nouveaux modes de sociabilité et d'une nouvelle économie : l'informatique est partout !

Pourtant, l'ordinateur lui-même demeure pour beaucoup une énigme, un objet mystérieux et un peu magique. Le terme d'informaticien semble d'ailleurs recouper une grande diversité de métiers et d'occupations réelles, allant du technicien à l'ingénieur réseau, en passant par le webmaître. Quant à la nature du travail de ceux qui font de la «recherche en informatique», c'est sans doute (à en juger par les réactions auxquelles j'ai moi-même été confrontée) une mystère encore plus épais, malgré le prestige un peu mythique que conservent des projets comme ceux de la robotique ou de l'intelligence artificielle.

Ce document se veut (en partie) une réponse à ceux qui se demandent quels sont les fondements de l'informatique. Il n'est pas conçu pour initier au maniement pratique des ordinateurs, ce n'est pas une introduction à la bureautique. Ce n'est pas non plus un manuel technique à l'usage de ceux qui souhaitent bricoler leur machine favorite. Si, au fil des pages, des informations utiles à ces deux catégories d'utilisateurs, ou aux novices, pourront être glanées (vocabulaire spécialisé, typologie de matériels, ordre de grandeurs des performances des machines actuelles...), tel n'est pas son objectif premier.

L'informatique dont il sera question ici est une *discipline scientifique* qui, en tant que telle, a ses propres questions, ses propres problèmes, et dispose pour les aborder d'outils et de méthodes spécifiques. De cette discipline, on abordera les *fondements théoriques* ainsi que quelques réalisations pratiques, mais on insistera plus sur les concepts que sur la technique. Cette présentation relève donc principalement d'une démarche de *vulgarisation scientifique* destinée à un public de non spécialistes, mais qui se place à un niveau non trivial, difficilement trouvable dans les manuels habituellement disponibles.

J'ai ici essayé de décrire, de façon aussi abordable que possible, ce que, en tant qu'informaticienne, je souhaite que «l'honnête homme du XXIème siècle» sache et pense de ma discipline, appelée à coup sûr à un grand développement dans les années qui viennent.

Chapitre 2

Qu'est-ce que l'informatique ?

1 Introduction

Le statut de l'informatique en tant que discipline est ambigu et mal compris : est-il à chercher du côté de la science ou du côté de la technique ? Quel est l'objet d'étude propre aux informaticiens ? Quelles sont leurs vraies compétences ? Avant de nous engager sur ces points, commençons par éliminer les mauvaises réponses :

- *l'informatique n'est pas la «science des ordinateurs»* (ce que, pourtant, laisse croire sa traduction anglaise, «computer science») : non, les informaticiens ne savent pas nécessairement réparer un ordinateur en panne, diagnostiquer un problème électronique ou effectuer des branchements compliqués, ils ne sont pas toujours les meilleurs guides quand il s'agit d'acheter un nouveau modèle de scanner ou de modem ; oui l'informatique peut s'étudier avec un papier et un crayon, même en absence d'ordinateur...
- *l'informatique n'est pas la «science des logiciels»* : non, les informaticiens ne connaissent pas nécessairement toutes les nouvelles versions des programmes du commerce, ils ne savent pas toujours utiliser toutes leurs fonctions, ils ne passent pas (toujours) leurs journées à tester des jeux ou à chercher des bugs...

La compétence réelle des informaticiens n'est ni matérielle, ni fonctionnelle. Alors, qu'est ce que l'informatique ? C'est quelque chose entre les deux, quelque chose de plus abstrait et de plus fondamental sans quoi ni les ordinateurs ni les logiciels ne pourraient fonctionner... Pour arriver à une définition satisfaisante, notre démarche sera de partir de ces deux niveaux de description habituels : matériel et logiciel, et de faire émerger leurs principes sous-jacents.

Commençons donc par l'aspect logiciel. La connaissance commune de l'informatique se fonde en effet généralement sur la pratique de quelques logiciels d'usage courant : traitements de texte, tableurs, navigation sur l'Internet... L'image de l'ordinateur comme «grosse machine à calculer», fonctionnant pendant des heures pour réaliser ses calculs, reste également présente, mais de façon plus mythique et lointaine, *vue au cinéma*. Y a-t-il une base commune à tous ces usages, apparemment si disparates ? Quelles sont les caractéristiques partagées par tous ces logiciels ?

Pour aborder ces questions, on peut commencer par remarquer que tout programme peut être décrit par deux aspects fondamentaux : les *données* qu'il manipule et les *traitements* qu'il permet de réaliser sur ces données. Le tableau de la figure

logiciel	données	traitemements
traitement de textes	caractères alpha-numériques (lettres de l'alphabet, chiffres et tous les autres caractères du clavier)	copier, coller, effacer, déplacer, intervertir, changer la casse ou la police, mettre en page...
calculs numériques, tableurs, outils de gestion	nombres, opérations et symboles mathématiques	calculer, écrire et résoudre des équations, faire des graphiques
jeux	dessins, personnages animés, sons	appliquer les règles du jeu
bases de données, logiciels documentaires	textes, images, données factuelles	stocker en mémoire et rechercher des données
Internet, CD-Rom	toutes les données citées précédemment	tous les traitements cités précédemment + communication entre machines, échange de données

FIG. 2.1 – données et traitements des logiciels courants

2.1 permet de résumer ces caractéristiques pour les logiciels les plus couramment utilisés.

Toutes les données citées dans ce tableau (et toutes celles, en général, manipulées par les ordinateurs) ont un point commun : ce sont des *données discrètes*, c'est-à-dire distinctes les unes des autres et qu'on peut énumérer une par une. Tous les traitements évoqués ici (et tous les autres traitements possibles en informatique) ont également en commun d'être des *traitements effectifs*, exprimables par des algorithmes. Les fondements de l'informatique sont à chercher dans ces deux notions, que nous allons donc maintenant détailler.

2 Données discrètes et codage

Tout le monde a entendu dire que les ordinateurs «ne fonctionnent qu'avec des 0 et des 1». Qu'est-ce que cela signifie exactement et où, dans l'ordinateur, sont donc cachés ces 0 et ces 1 ? Pour le comprendre, il faut cette fois partir des composants matériels qui constituent un ordinateur et aborder quelques notions élémentaires de la théorie de l'information.

2.1 La notion de bit

L'unité de base de la théorie de l'information est le bit, contraction de ***bi****nary digi****t***, qui signifie en anglais *nombre binaire*. Un bit, par définition, est un composant quelconque ne pouvant se trouver que dans deux états possibles, exclusifs l'un de l'autre. On peut imaginer bien des dispositifs physiques pouvant répondre à cette définition, et nombre d'entre eux ont été exploités au moins une fois dans l'histoire de l'informatique. Ainsi, par exemple :

– une lampe électrique qui est soit allumée soit éteinte : les tout premiers ordinateurs, dans les années 40, utilisaient ainsi des milliers de lampes ou «tubes à vide» : un tel dispositif était hélas en panne dès qu'une lampe grillait ;
– un fil électrique dans lequel le courant circule ou pas (ou sa version miniaturisée, le «circuit intégré») : c'est évidemment le composant de base des ordinateurs, avec les transistors qui peuvent, eux, être vus comme des interrupteurs miniatures. Les fibres optiques réalisent la même fonction que les fils, mais avec la lumière au lieu de l'électricité.
– un aimant pouvant être polarisé «Sud» ou «Nord» : les mémoires des ordinateurs actuels, leur «disque dur», ainsi que les anciennes disquettes, sont composés de milliards de petits aimants ;
– une surface ayant soit un creux soit une bosse : les CD-audio, les CD-Rom, les DVD, ne sont rien d'autre que des morceaux de plastique gravés de creux et de bosses tellement minuscules que seul un faisceau laser peut les distinguer ;
– un récipient pouvant être plein ou vide : il y a, à la Cité des Sciences de la Villette, un «calculateur à eau» fonctionnant avec des seaux et des tuyaux remplis d'eau.

Par convention, pour s'abstraire de ces contingences matérielles, on appelle l'un des deux états possibles d'un tel composant **0**, et l'autre **1**. Ces deux symboles sont arbitraires et n'ont pas de signification numérique. On aurait tout aussi bien pu choisir les symboles **a** et **b** à la place, ou tout autre couple de deux signes *distincts*, puisque c'est uniquement cette distinction qui est importante.

A partir de maintenant, un bit sera donc pour nous un espace dans lequel on pourra soit écrire 0, soit écrire 1. Que faire avec de tels composants aussi élémentaires ? Avec un seul, pas grand chose, mais avec plusieurs, beaucoup de choses !

Si, en effet, on dispose de deux bits, alors le nombre total d'états possibles que peuvent prendre ces deux bits est de quatre : 00, 01, 10 ou 11.

Si on en prend trois, alors huit combinaisons sont possibles : 000, 001, 010, 011, 100, 101, 110, 111. On peut représenter ces huit combinaisons comme tous les différents parcours possibles dans un arbre où chaque branche est un choix entre 0 ou 1, comme dans la figure 2.2.

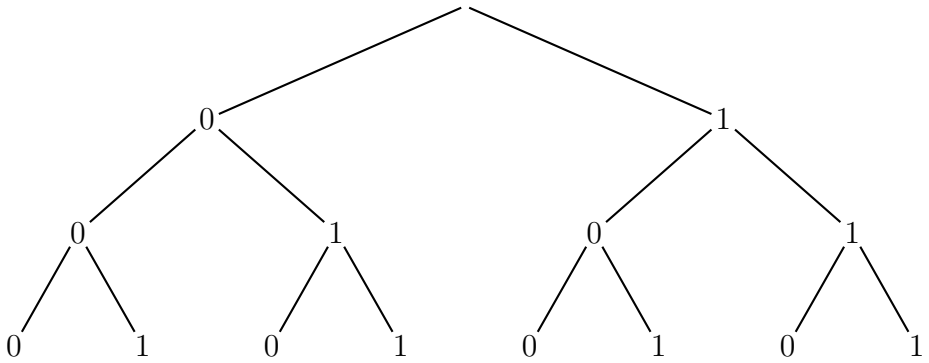


FIG. 2.2 – arbre des combinaisons possibles de 3 bits

En fait, ajouter un bit multiplie par deux le nombre de combinaisons possibles, puisque le bit ajouté a lui-même deux états possibles. Dans la représentation arbores-

cente, cela revient à dédoubler chaque «feuille» de l'arbre (chaque dernier élément), et donc à multiplier par deux le nombre total de chemins possibles (égal au nombre de feuilles). Ce raisonnement permet de montrer que, avec n bits, on a 2^n combinaisons possibles.

Le bit est l'unité de comptage de la mémoire des ordinateurs. Les multiples utilisés se comptent également en puissance de deux.

1 octet (byte en anglais) = 8 bits ($8 = 2^3$) et permet $2^8 = 256$ combinaisons différentes possibles

1 Kilo-octet = 2^{10} octets = 1024 octets, et coïncide presque avec la définition usuelle du kilo qui vaut 10^3

1 Mega-octet = 1024 Kilo-octets, soit environ 10^6 octets

1 Giga-octet = 1024 Mega-octets, soit environ 10^9 octets

1 Tera-octet = 1024 Giga-octets, soit environ 10^{12} octets

Pour avoir un ordre de grandeur de ces valeurs : une disquette a une capacité d'environ 1 Mega-octet, une clé USB entre 64 Mega-octet et 1 Giga octet, un CD contient environ 600 Mega-octet, un DVD jusqu'à environ 17 Giga-octets. La mémoire des disques durs actuels se compte aussi en Giga-octets. On peut en déduire par exemple le nombre d'aimants contenus sur une disquette... Voyons maintenant comment coder les données citées dans le tableau de la figure 2.1 à l'aide de bits.

2.2 Les caractères alphanumériques

Les caractères alphanumériques sont tous les caractères disponibles sur un clavier d'ordinateur, ainsi que ceux qui résultent des combinaisons de touches. Combien existe-t-il de caractères alphanumériques ? L'alphabet latin utilise 26 symboles différents, dont chacun existe en version minuscule et majuscule, ce qui fait 52 symboles. A ces lettres, s'ajoutent les 10 symboles de chiffres : 0, 1, 2,...9 et une vingtaine de notations mathématiques courantes : opérateurs ($+$, $-$, $*$, $/$), comparateurs ($<$, $>$), signe d'égalité ($=$), symboles logiques... De nombreux autres caractères sont utilisées dans les graphies usuelles : ponctuations (“.”, “;”, “?”, “!”, “ :”), apostrophe, guillemets, tirets, parenthèses, crochets, accolades, soit encore au moins quinze dessins nouveaux. De plus, des caractères spéciaux sont passés à l'usage courant : symboles de monnaies (\$, £) ou abréviations comme : &, §, @... Comptons-en donc une dizaine. Enfin, un éditeur de textes considère comme symbole certains signes invisibles comme les espaces blancs, les passages à la ligne ou les fins de fichier (comptons en 5). Notre très grossier calcul nous amène donc à environ :

$52 + 10 + 20 + 15 + 10 + 5 = 112$

Pour associer à chacun de ces caractères une suite distincte de 0/1 qui le code, nous avons donc au minimum besoin, d'après notre calcul précédent, de 7 bits, puisque $2^7 = 128$. C'est précisément ce que réalisait la première version du code ASCII (pour American Standard for Communication and International Interchange).

Mais ce calcul est trop restrictif. En effet, certaines langues utilisent en outre des caractères spéciaux : voyelles accentuées, trémas et cédilles en français, tilde espagnol, β allemand... Un nouveau symbole, comme celui qui a été choisi pour l'Euro, peut apparaître pour des raisons indépendantes de l'informatique. Il est donc plus raisonnable de conserver une marge de manoeuvre, et de prendre plutôt 8 bits,

soit un octet, qui permet 256 combinaisons différentes. C'est ce que réalise le code ASCII actuel.

Ce code a donc subi des modifications, et il en existe de multiples variantes, mais nous ne rentrerons pas dans ces détails ici... Ces variations expliquent toutefois que des messages peuvent être plus ou moins déchiffrables quand on les fait passer d'une machine à une autre (parfois même d'un programme à un autre sur une même machine). Il a par exemple longtemps été recommandé d'éviter d'utiliser les caractères accentués dans les courriers éléctroniques, parce que certains programmes de gestion de ces courriers utilisaient un code rudimentaire (fondé sur les claviers américains) qui les ignorait.

Quelle que soit la variante adoptée, le codage des caractères alphanumériques est donc réalisé par une *association arbitraire et conventionnelle* entre un caractère et une suite de bits. Le tableau de la figure 2.3 donne quelques exemples du code ASCII sur 8 bits associé à certains caractères :

code ASCII	**caractère**
00100011	#
00100100	$
00100101	%
01111010	z
01111011	{
00100001	!
00110001	1

FIG. 2.3 – quelques exemples de code ASCII

Depuis les années 90, une initiative internationale cherche à promouvoir une nouvelle manière de référencer tous les caractères de toutes les langues écrites (actuelles ou ayant existé) du monde : c'est la norme *unicode*. Elle répertorie environ 250 000 caractères différents, chacun associé à un nombre distinct. Mais cette norme ne fixe pas un codage en bits unique, et peut utiliser un nombre variable de bits (tous les caractères ne sont pas codés avec le même nombre de bits) ; nous ne la détaillerons pas non plus ici.

Les «éditeurs de textes» (comme le «Bloc Note» de Windows) sont des programmes servant à manipuler (copier, coller, effacer, déplacer, intervertir...) les caractères alphanumériques «purs». Ils permettent donc des *substitutions* de bits correspondant à des codes ASCII. Ils ne sont pas adaptés à la mise en page des textes, à leur formatage visuel, mais sont largement suffisants pour écrire des messages factuels ou des programmes informatiques.

Les logiciels de «traitements de textes» utilisent en outre des codes spéciaux spécifiques pour représenter la *mise en page* du texte : les marges, l'aspect «centrée» ou «justifié» d'un paragraphe, la police et la taille de l'affichage des caractères... Même en prenant en compte les nouveaux codes associés à cette présentation, le codage des textes est économique : sur une disquette (environ 1 Mega-octets) on peut stocker un livre de 500 pages et sur un CD (environ 600 Mega-octets) une encyclopédie.

2.3 Les nombres entiers

On pourrait procéder avec les nombres entiers de la même façon que précédemment, c'est-à-dire par association arbitraire. Les chiffres de 0 à 9 ont d'ailleurs, nous l'avons vu, *en tant que caractère*, un code ASCII. Mais les nombres ont un statut et un usage particuliers : ils sont complètement ordonnés et servent à faire des calculs. Leur codage utilise une technique mathématique adaptée : la *numérotation binaire* (ou *en base* 2).

Pour comprendre le codage binaire des nombres, le mieux est d'abord de bien analyser leur représentation usuelle en base 10, dite aussi décimale. Cette représentation fait usage de 10 symboles différents de base : 1, 2, 3, ..., 9 et 0 et utilise le principe de la *numérotation de position*. Cette numérotation a été inventée par les indiens au Vème siècle en même temps que le symbole 0 (qu'elle rend nécessaire), et transmis en occident par l'intermédiaire des arabes (d'où le nom de «chiffres arabes»). Son principe est que chacun des symboles constituant un nombre représente en fait une puissance croissante de 10 lu de droite à gauche. La valeur réelle du nombre est alors la somme de ces puissances. Ainsi par exemple :

$533 = 5 * 100 + 3 * 10 + 3 * 1 = 5 * 10^2 + 3 * 10^1 + 3 * 10^0$

$2001 = 2 * 1000 + 1 * 1 = 2 * 10^3 + 0 * 10^2 + 0 * 10^1 + 1 * 10^0$

Quand on «lit» un tel nombre, on attribue à chaque chiffre qui le constitue, en fonction de sa place dans l'écriture du nombre, une certaine puissance de 10. Ainsi, les deux symboles «3» du premier nombre ne représentent pas, en fait, la même valeur (l'un vaut 30, l'autre vaut 3). La «grille de lecture» sous-jacente est donc un tableau de la forme :

$\mathbf{10^3}$	$\mathbf{10^2}$	$\mathbf{10^1}$	$\mathbf{10^0}$
	5	3	3
2	0	0	1

Le principe de la «retenue» dans les additions est alors naturel : une somme de chiffres qui dépasse 10 pour une puissance de 10 donnée peut être représentée en passant à la puissance de 10 suivante, c'est-à-dire celle immédiatement à gauche. La représentation des chiffres romains n'utilise pas, elle, la numérotation de position. Par exemple, dans l'écriture du nombre 18 en chiffre romains : XVIII, les trois symboles «I» à la fin valent tous les trois 1, indépendamment de leur position. Dans un tel système, les opérations mathématiques s'effectuent très difficilement.

Le principe de la numérotation de position se généralise à n'importe quelle base de représentation en changeant simplement la puissance correspondante : pour la représentation binaire, ce sont donc les puissances de 2 qui vont servir. Pour une fois, nous prenons donc nos symboles 0/1 au sérieux : ici, 0 correspond bien au chiffre 0 et 1 au chiffre 1. Le tableau suivant donne des exemples de codage de nombres en base 2 :

$\mathbf{2^5 = 32}$	$\mathbf{2^4 = 16}$	$\mathbf{2^3 = 8}$	$\mathbf{2^2 = 4}$	$\mathbf{2^1 = 2}$	$\mathbf{2^0 = 1}$	**nombre décimal**
		1	0	0	1	$8 + 1 = 9$
			1	1	1	$4 + 2 + 1 = 7$
	1	0	1	1	1	$16 + 4 + 2 + 1 = 23$
1	0	0	1	0	1	$32 + 4 + 1 = 37$

Grâce à cette représentation, les opérations mathématiques usuelles sur les nombres binaires s'effectuent exactement de la même manière que sur les nombres décimaux. Ainsi, les «retenues» sont nécessaires dès que l'on dépasse un total de 2 pour une puissance de 2 donnée ; elles consistent, comme en base 10, à ajouter 1 dans une colonne plus à gauche (par exemple : si le résultat d'une colonne vaut 2, on pose 0 en bas de cette colonne et une retenue de 1 dans la colonne immédiatement à gauche, puisque «10» en base 2 correspond au nombre décimal 2).

Example 1 *L'addition* $10111 + 101$ *est exécutée en base* 2 *ci-dessous.*

$$\begin{array}{cccccc} & & 1 & 1 & 1 & \\ & 1 & 0 & 1 & 1 & 1 \\ + & & & 1 & 0 & 1 \\ \hline & 1 & 1 & 1 & 0 & 0 \end{array}$$

En base 10*, l'opération mathématique correspondante est :* $23 + 5 = 28$.

Pour représenter les nombres négatifs, il suffit de réserver l'un des bits (par exemple le plus à gauche) pour coder le signe du nombre : par exemple la valeur 0 de ce bit signifie que le nombre est positif, 1 que le nombre est négatif (pour le nombre 0, la valeur de ce bit est alors indifférente : 0 a donc dans ce cas deux représentations différentes possibles). Mais ce codage rend délicat les calculs avec les nombres négatifs. D'autres codages plus astucieux (que nous ne détaillerons pas ici) permettent de réaliser des opérations mathématiques aussi simplement avec les nombres négatifs qu'avec les nombres positifs.

Pour un nombre donné, le codage en base 2 occupe plus de place que le codage décimal (c'est naturel puisque, en revanche, il utilise moins de *symboles distincts* de base). Pour maîtriser l'espace mémoire nécessaire au codage des nombres, un nombre fixe de bits est réservé pour toute donnée de type «nombre entier». Si, lors d'un calcul avec des nombres entiers, cet espace est insuffisant (à cause des retenues), alors le résultat annoncé du calcul sera faux (c'est le même problème que pour le «bogue de l'an 2000»).

Ainsi, même dans les dernières versions de Turbo Pascal (un environnement de programmation pour le langage Pascal encore en usage), deux octets étaient réservés pour tout nombre entier. Or 2 octets ne permettent que $2^{16} = 65536$ combinaisons différentes de 0/1. Concrètement, cela signifie que les seuls nombres entiers relatifs qui peuvent être représentés sont ceux compris entre -32768 et 32767. Cette limite va donner lieu à des erreurs dès que ces bornes sont dépassées dans un calcul.

2.4 Grands nombres et nombres décimaux

Pour coder les grands nombres sans risquer les erreurs précédentes, et pour coder les nombres décimaux (c'est-à-dire avec une virgule), la méthode adoptée s'appelle «représentation en virgule flottante». Son principe est de normaliser l'écriture de ces nombres, en jouant sur les puissances de la base, et de coder indépendamment la partie «exposant» et la partie «mantisse» du résultat.

Example 2 *En base* 10*, on peut écrire :*

$27000000 = 0,27.10^9$
$0,000027 = 0,27.10^{-4}$

Tout nombre en base 10 peut ainsi s'écrire sous la forme d'un nombre décimal commençant par 0,... suivi de ce qui s'appelle la «mantisse», qui contient les chiffres significatifs du nombre (et commence donc toujours par un chiffre différent de 0) et multiplié par une certaine puissance entière de 10 (éventuellement négative). Le principe est exactement le même si la base de la numérotation est 2.

Par exemple le nombre 27 en base 2 peut s'écrire :

$11011 = 0,11011.2^{101}$, où 11011 est la mantisse et 101 (qui vaut 5 en base 10) est l'exposant.

Dans la représentation en virgule flottante, un espace mémoire est réservé pour coder la mantisse, et un autre est réservé pour coder la valeur de la puissance. Un nombre trop grand pour être codé de façon traditionnelle sur deux octets peut être codé correctement avec ce système.

La représentation en virgule flottante n'exclut pourtant pas tout risque d'erreur : pour un nombre très grand avec beaucoup de chiffres significatifs, l'ordre de grandeur du nombre sera conservé (grâce au codage de l'exposant) mais le détail des décimales peut être perdu. De même un nombre très proche de 0, dont l'exposant est très grand en valeur absolue et négatif peut ne pas être représenté correctement : tout ordinateur a sa limite en terme de représentation des nombres (pour les petits nombres, on parle de «epsilon machine»).

Les nombres décimaux qui s'écrivent avec un nombre infini non périodique de chiffres après la virgule (par exemple : π) n'appartiennent pas au monde du «discret» : les coder avec des 0/1 impose donc une part d'approximation irrémédiable. Les «erreurs d'arrondis» sont une conséquence directe et inévitable du codage des nombres réels à l'aide de symboles discrets.

2.5 Les tableaux

Les tableaux, utiles dans les tableurs ou les bases de données, peuvent être eux aussi codés à l'aide de 0/1. Imaginons par exemple que nous souhaitions coder le tableau suivant, qui ne contient que des nombres entiers :

12	5	-3	0	-1
-15	7	2	0	5
-8	-2	0	1	2

Pour représenter un tel tableau, il suffit par exemple :
- de repérer le nombre de lignes et le nombre de colonnes du tableau ;
- de fixer un nombre de bits permettant de coder à la fois le nombre de lignes et de colonnes et chacune des données présentes dans une case du tableau ;
- de coder successivement chacune de ces données dans une longue chaîne de 0/1.

Dans notre exemple, le tableau a 3 lignes et 5 colonnes, et tous les nombres qu'il contient sont inférieurs à $16 = 2^4$ en valeur absolue. Chacune de ces données

peut donc être codée sur 5 bits (le premier bit servira pour le signe). En codant successivement le nombre de lignes puis le nombre de colonnes et enfin chaque case lue de gauche à droite et de haut en bas, on obtient la chaîne suivante (pour faciliter la lecture, on laisse un espace entre chaque donnée distincte) :

00011 00101 01100 00101 10011 00000 10001 11111 00111 00010 00000 00101 11000 10010 00000 00001 00010

A partir de cette chaîne, et à condition bien sûr de connaître les conventions de codage, on peut entièrement reconstituer le tableau initial.

De même, on peut aussi coder un tableau de données discrètes hétérogènes (par exemple : dont certaines cases contiennent un nombre et d'autres cases un ou des caractères). Dans ce cas, il faut prévoir d'inclure dans le codage quelques bits qui signalent, pour chaque case, la nature de la donnée qu'elle contient. Par exemple, si chaque case ne peut contenir qu'un nombre ou qu'un caractère, il suffit de le signaler par un bit supplémentaire (valant 0 pour «caractère» et 1 pour «nombre») associé à chaque case, indiquant comment interpréter les bits suivants.

Les bases de données et les logiciels documentaires sont essentiellement constituées de tableaux de données hétérogènes, constituant ce que nous avons appelé dans la figure 2.1 des «données factuelles». Par exemple, le catalogue (simplifié) d'une bibliothèque peut avoir la forme du tableau de la figure 2.4, dans lequel chaque colonne est d'un *type* particulier. Dans notre exemple, les trois premières colonnes seraient de type «chaîne de caractères» (il faudrait préciser pour chacune le nombre de caractères maximum qu'elle peut contenir), la dernière étant de type «nombre entier».

titre de livre disponible	**nom auteur**	**prénom auteur**	**date parution**
Mme Bovary	Flaubert	Gustave	1857
Notre Dame de Paris	Hugo	Victor	1831
Les misérables	Hugo	Victor	1862
...	...	...	...

FIG. 2.4 – début d'un catalogue simplifié de bibliothèque

2.6 Codage des sons

Les sons semblent être de nature différente des symboles et des nombres. On va en fait montrer comment un son, via une part d'approximation, peut se ramener à une suite de nombres. Un son peut (très schématiquement) se représenter par une courbe exprimant son amplitude en fonction du temps, comme sur la figure 2.5.

Un son réel est en fait une superposition de courbes de ce genre, mais l'oreille humaine n'est pas sensible à toutes les «harmoniques» qu'il contient. Le principe du fameux format de codage appelé MP3 est, précisément, de ne coder que la partie du son que les sens humains perçoivent.

Pour coder une telle courbe continue en bits, il est nécessaire de la *discrétiser*, c'est-à-dire de transformer les paramètres qui la définissent (le temps et l'amplitude), qui varient suivant des échelles continues, en paramètres discrets ne pouvant prendre

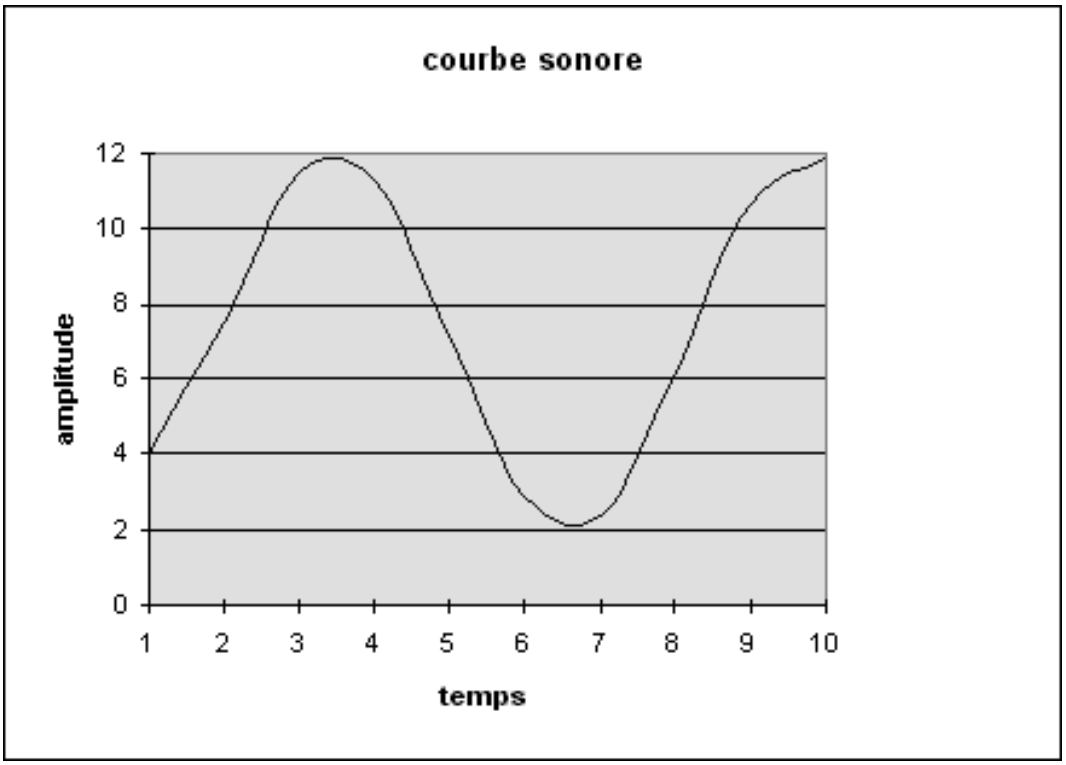


FIG. 2.5 – courbe sonore continue

qu'un nombre fixé de valeurs distinctes. On peut décomposer ce processus en deux étapes :

- on discrétise le temps, en abscisse, en procédant à un *échantillonnage* : le temps est «découpé en morceaux» à intervalles réguliers. On appelle *fréquence d'échantillonnage* le nombre d'intervalles découpés par seconde (ainsi, doubler la fréquence d'échantillonnage revient à multiplier par deux le nombre de morceaux découpés, c'est-à-dire à diviser par deux la largeur de chaque morceau) ;
- on discrétise l'amplitude en approchant la courbe par un nombre : pour chaque intervalle de temps, on prend le nombre qui permet de «s'approcher le plus possible» de la courbe réelle.

En appliquant ce traitement à la courbe de la figure 2.5, avec une fréquence d'échantillonnage de 1 unité par seconde et en n'acceptant que des valeurs entières pour l'amplitude, on obtient la nouvelle courbe de la figure 2.6.

Pour coder cette courbe, il suffit maintenant de coder successivement les valeurs correspondant à chaque morceau de temps. Dans notre exemple, ces valeurs successives sont approximativement : 4, 7, 11, 11, 7, 3, 2 6, 11, 12. A condition de connaître la fréquence d'échantillonnage, la donnée de ces nombres suffit à reconstruire la courbe discrétisée, et donc à reconstituer les variations du son initial.

Sur notre exemple, l'approximation semble grossière, mais en prenant une fréquence d'échantillonnage très grande et un codage des nombres permettant une grande précision pour les réels, la courbe discrète suit de si près la courbe réelle que le son y est parfaitement retranscrit. Dans un CD-audio haute fidélité, par exemple,

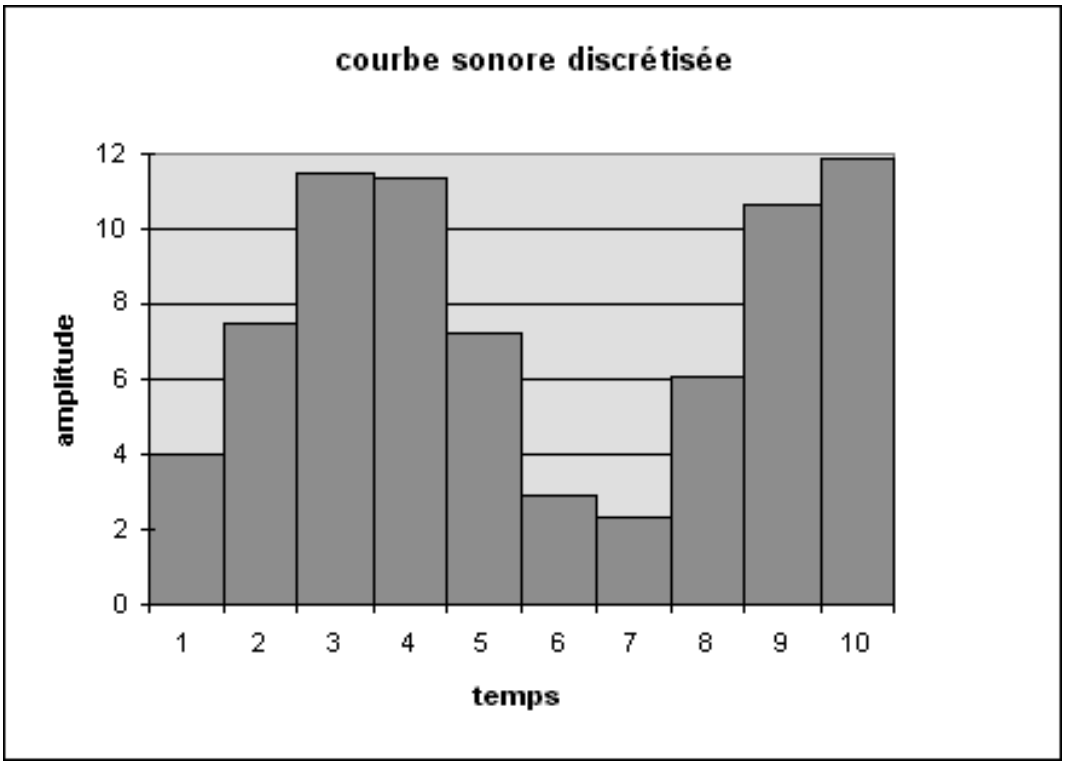


FIG. 2.6 – courbe sonore discrétisée

la fréquence d'échantillonnage est de 44 100 échantillons par seconde. Le nombre de bits occupés par un CD-audio usuel (non codé en MP3) est d'environ 500 Mega octets.

2.7 Codage des images

Il y a plusieurs manières de coder une image, de la «discrétiser». On distingue principalement deux formats de codage, chacun étant adapté à un type d'image particulier et à un usage particulier.

1. Le codage «bit map»

   Ce codage est utilisé pour *numériser des images existantes* (photos, reproductions de peintures...). Il est donc plutôt adapté à l'*analyse* et au traitement des images. Son principe est celui du quadrillage : l'image est découpée à travers une grille au maillage plus ou moins fin, suivant des lignes horizontales et verticales. Cette grille découpe donc l'image en petits éléments rectangulaires appelés «pixels» (par contraction de l'anglais ***pic****ture* ***el****ement*). On appelle *définition* d'une image numérique le nombre de pixels qu'elle contient par unité d'espace, donc par cm^2. Cette définition est d'autant plus grande que la grille est fine, et donc que les pixels sont petits. Pour coder une image ainsi découpée, il suffit maintenant d'associer à chaque pixel sa couleur dominante, et de coder les couleurs des pixels les unes après les autres, dans un ordre fixé à l'avance (par exemple de haut en bas et de gauche à droite).

Les codages des couleurs les plus courants sont les suivants :

- le codage en noir et blanc pur : un bit suffit alors pour coder un pixel, suivant une association arbitraire (par exemple : 0 pour blanc et 1 pour noir) ;
- le codage en 256 couleurs (ou 256 niveaux de gris) : le spectre des couleurs (ou des niveaux de gris) est découpé en 256, et un code arbitraire sur un octet est associé à chacune des nuances (rappel : 1 octet = 8 bits et permet donc de coder $2^8 = 256$ couleurs différentes possibles)
- le codage en 16 millions de couleurs : ce codage, aussi appelé RGB (pour Red, Green, Blue), utilise la décomposition des couleurs à l'aide des 3 couleurs primaires (rouge, vert et bleu), comme pour les écrans de télévision. Rappelons qu'additionner des faisceaux lumineux tend à les éclaircir (puisque le blanc est la somme de toutes les couleurs) ; les règles d'addition des rayons lumineux ne sont donc pas les mêmes que celles des additions des couches de peinture... La contribution de chaque couleur primaire à la définition d'une certaine couleur est alors codée sur 1 octet, ce qui nécessite donc 3 octets pour coder chaque pixel. Dans ce cas, le premier octet code la contribution du rouge à la couleur du pixel, le deuxième la contribution du vert et le troisième celle du bleu (or 2^{8*3} vaut environ 16 millions, donc il y a bien 16 millions de couleurs différentes qui peuvent être codées par ce système).

Lorsqu'on numérise une image à l'aide d'un scanner, il faut préciser la définition de l'image et le codage des couleurs souhaité ; ces 2 paramètres conditionnent fortement l'espace mémoire nécessaire au codage de l'image. Le codage d'une image «bit map» à haute définition et à l'aide de couleurs fines nécessite un grand nombre de bits. La figure 2.7 montre la même image bit map à deux échelles différentes, pour illustrer l'importance de la *définition* dans l'impression globale rendue par une telle image. Les images bit map sont souvent ce qui occupe le plus de place dans la mémoire des ordinateurs...

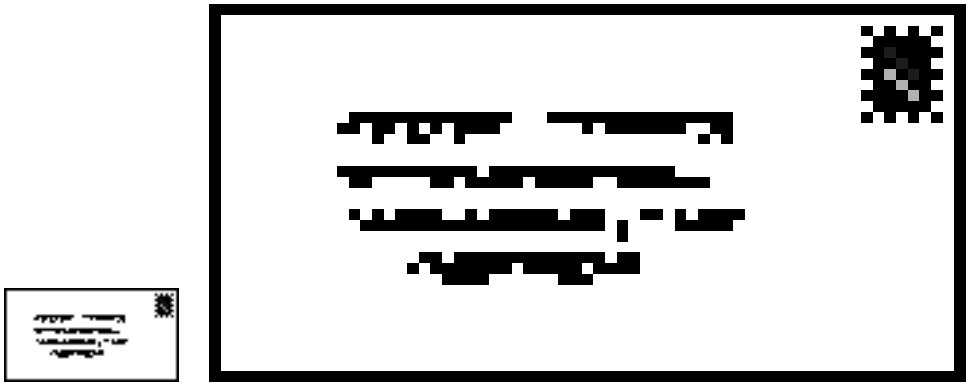


FIG. 2.7 – images bits map

2. Le codage vectoriel

   Ce codage sert principalement à la *synthèse* d'images. Il est particulièrement bien adapté aux images qui se présentent sous la forme de schémas ou de plans. Il y en a plusieurs variantes, suivant que l'image à créer doit être vue en 2 ou en 3 dimensions.

– images en 2D

Le codage vectoriel en 2D considère qu'une image est une combinaison de figures géométriques standards : ellipses, droites, rectangles, flèches... Un code arbitraire peut être associé à chacune de ces figures (il y en a un nombre fini). Par ailleurs, l'espace du dessin est repéré par un système de coordonnées classiques, désignées habituellement par X et Y. Pour repérer une figure particulière dans un dessin, il suffit de connaître :

- sa nature (par exemple : un rectangle) déterminée par son code ;
- les coordonnées (X, Y) de deux points extrêmes qui permettent de le situer sans ambiguïté dans l'espace (pour un rectangle : son coin supérieur gauche et son coin inférieur droit) ;
- son éventuelle couleur de trait et de remplissage (repérée comme pour les images bit map).

Contentons-nous ici de dessins «au crayon», sans tenir compte des couleurs. Imaginons par exemple que les codes des figures standards soient les suivants :

- ellipse : 000 ;
- rectangle : 001 ;
- ligne continue : 010 ...

et que l'on veuille coder le dessin de la figure 2.8.

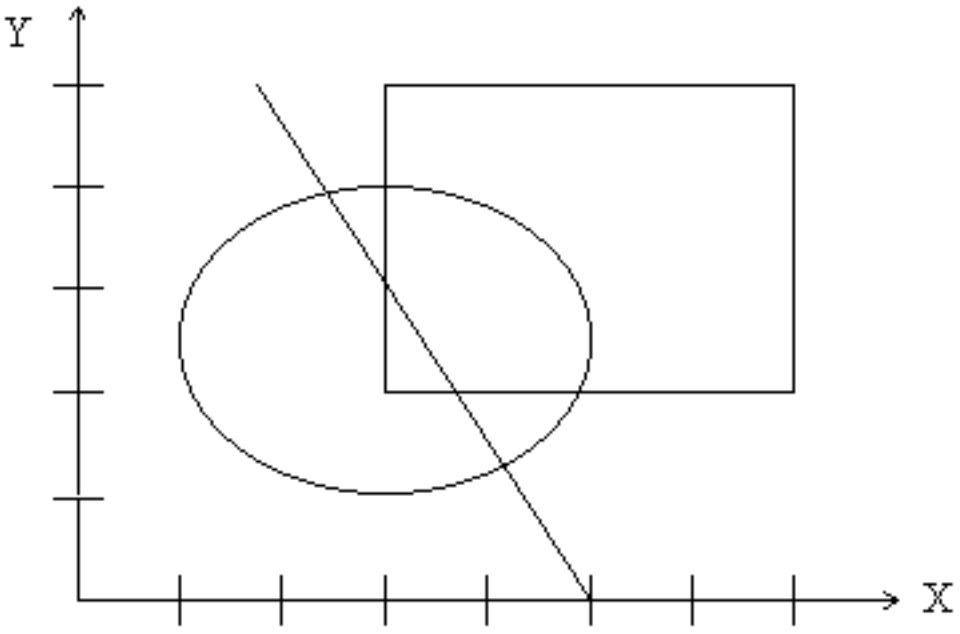


FIG. 2.8 – dessin vectoriel

Les coordonnées (supposées entières) des points extrêmes peuvent aussi être codées sur 3 bits. Chaque figure élémentaire est alors représentée par 5 groupes de 3 bits chacun. Par exemple, l'ellipse de la figure 2.8 est codée par : 000 001 100 101 001. Dans ce code, le premier groupe de 3 bit est le code de la figure ellipse, les deux suivants codent les nombres 1 et 4, qui sont

les coordonnées (X,Y) du point qui circonscrit cette ellipse en haut à gauche, les deux derniers codent les nombres 5 et 1 qui sont les coordonnées (X,Y) du point qui la circonscrit en bas à droite. Le dessin complet se code par la succession des codes des figures élémentaires (dans un ordre quelconque) qui le composent, soit par exemple :

000 001 100 101 001	001 011 101 111 010	010 010 101 101 000
← code de l'ellipse →	← code du rectangle →	← code de la ligne→

D'autres codes représentant la couleur de trait et de remplissage de chaque figure élémentaires, l'ordre de superposition de ces figures, etc. devraient être ajoutés à cette suite.

Le codage vectoriel 2D est celui qui est exploité par les logiciels de dessin usuels comme PaintBrush, et il permet de faire subir simplement les transformations géométriques classiques (translations, rotations, symétries, homothéties...) aux images.

Il est très économe en espace mémoire, et c'est aussi la raison pour laquelle il est utilisé pour créer des animations sur Internet avec la technologie «Flash».

– images en 3D

Le principe de ce codage est exactement le même que le précédent, sauf que les primitives graphiques utilisées sont des *volumes* (sphères, cylindres, parallélépipèdes...) au lieu d'être des figures de la géométrie plane.

C'est lui qui est utilisé dans les logiciels de CAO (Conception Assisté par Ordinateur), de réalité virtuelle, ou pour les effets spéciaux numériques dans les films... Il nécessite un effort de modélisation considérable pour représenter un environnement complet uniquement à partir de primitives, mais permet ensuite des applications spectaculaires en simulant les intéractions opérant dans cet environnement. Plus la modélisation est fine, plus la capacité de mémoire et de calcul nécessaire est importante.

2.8 Codage numérique et codage analogique

En résumé, les données manipulées par les ordinateurs sont exclusivement des *données discrètes*. Le monde du discret, dans lequel on peut énumérer chaque entité distincte les unes après les autres s'oppose au monde du continu, dans lequel ce n'est pas possible. Mathématiquement, un ensemble est discret s'il peut être mis en bijection avec $\mathbb{N}$ ou une partie de $\mathbb{N}$, il est continu s'il peut être mis en bijection avec l'ensemble des nombres réels $\mathbb{R}$, ou un de ses intervalles. Dans $\mathbb{R}$, contrairement à $\mathbb{N}$, on ne peut pas "compter", c'est-à-dire énumérer tous ses éléments les uns après les autres sans en oublier aucun. Une autre manière de caractériser les données discrètes est précisément leur caractère codable à l'aide d'un alphabet fini (par exemple 0/1). On parle aussi de codage *numérique* (0 et 1 sont des nombres), mais cela ne signifie pas, nous l'avons vu, qu'il ne code que les nombres.

Les caractères alphanumériques et les nombres entiers appartiennent naturellement au monde du discret, et leur codage numérique ne pose donc pas de problème. En effet, quand il s'agit de traiter des données de nature discrète à l'aide d'un autre

dispositif discret, le codage/décodage est exact, c'est-à-dire qu'il préserve toute l'information. En revanche, les nombres réels, les sons et les images appartiennent au monde du continu. C'est pourquoi jusqu'à l'émergence de l'informatique multimédia, leur codage passait plutôt par un mode *analogique* : un tel code traduit des données continues à l'aide d'un autre dispositif lui aussi continu. Par exemple, dans les anciens disques en vinyle et les anciennes cassettes audio, le son était codé et restitué grâce à un sillon continu, dont la forme reproduisait (par une relation «d'analogie») la courbe des sons. La radio FM et la télévision hertzienne actuelles sont encore transmises à l'aide d'ondes continues dont les variations (les «modulations») traduisent suivant une échelle continue les variations du son. Les photographies "argentiques" sont réalisées à l'aide d'une surface (quasi) continue de produit chimique réactif qui restitue le continuum de nuances du spectre des couleurs. Les anciennes cassettes vidéo reproduisaient ainsi, une par une, les images d'un film sur leur bande.

Le codage numérique, lui, n'a droit qu'aux symboles 0 et 1. Pour coder numériquement des données continues, il faut donc passer par une phase de *discrétisation* ou *numérisation*, qui se paie par une part d'approximation. La qualité du son des CD-audio est néanmoins supérieure à celle des anciens disques «33 tours», et la définition des photographies numériques s'approche à grands pas de la précision des photographies traditionnelles.

2.9 Codages/décodages et changements de codes

L'échange de données entre *matériels* fonctionnant sous un mode numérique ou analogique impose parfois de passer par des phases de numérisation. Les plus courants de ces dispositifs sont les suivants :

- les modems (contraction de «**mo**dulateur»/«**dém**odulateur») sont des appareils qui transforment des données discrètes sous forme de bits en données analogiques pouvant circuler sur les lignes téléphoniques (et réciproquement). Il sont donc indispensables pour relier les ordinateurs entre eux en passant par le réseau téléphonique français (qui, lui, transmet des données continues). Les boîtiers fournis pour se connecter à l'Internet haut débit (freebox, livebox et autres) en sont les versions améliorées. L'ADSL est en effet simplement une technologie qui emploie pour transmettre ses données des fréquences non utilisées par la voie humaine dans les conversations téléphoniques. C'est pourquoi on peut à la fois téléphoner et envoyer/recevoir des données en passant pas la même ligne téléphonique avec ces boîtiers.
- les scanners ou numériseurs, transforment des images quelconques (textes ou photos) en suites de bits codant ces images sous forme «bit map».

Par ailleurs, les différents types de données étant, comme nous l'avons vu, codés de façon spécifiques, il peut parfois être nécessaire de passer d'un mode de codage à un autre. C'est ce que réalisent des *logiciels* comme :

- les Optical Character Recognizer (OCR) sont des programmes qui transforment les images sous forme «bit map» d'un texte en une suite de caractères ASCII. Il peuvent aussi reconnaître la mise en page du texte et la traduire dans les codes spécifiques utilisés pour les représenter dans les traitements de textes. Ils sont indispensables si on veut «traiter» avec un éditeur ou un traitement

de textes un fichier (2.4) issu de la numérisation d'un texte écrit.
– les vectoriseurs d'images transforment les dessins sous forme «bit map» en images vectorielles. Ils sont utilisés pour traiter des plans techniques ou des dessins d'architectes.

De manière générale, tout logiciel qui permet de manipuler des données de natures différentes (par exemple, des caractères et des nombres) accorde un rôle primordial à la notion de *typage* ou de *format*. Connaître le *type* d'une donnée, en effet, est indispensable pour savoir comment interpréter la suite de 0/1 qui la code.

Enfin, certains codages sont plus économiques que d'autres, au sens où ils nécessitent de réserver un plus ou moins grand nombre de bits pour coder une même donnée. Les logiciels de *compression de données* ont pour objectif de réduire au maximum l'espace mémoire occupé par le codage d'un certain type de données. Les stratégies qu'ils mettent en oeuvre varient évidemment suivant la nature des données, et donc des codages, qu'ils ont à traiter, et suivant que la compression visée est exacte ou approchée (c'est-à-dire si elle préserve ou non toute l'information présente ou pas). Par exemple, pour compresser l'image «bit map» d'un texte, on peut commencer par coder de façon vectorielle les espaces blancs de la page. Pour le codage d'une scène vidéo (correspondant à une succession d'images «bit map»), on peut se contenter de coder complètement la première image, puis seulement la *différence* entre chaque image et la précédente...

On comprend mieux, à l'issue de ce passage en revue de tous les codages utilisés en informatique, la puissance du monde numérique : elle vient de la capacité à stocker et à échanger des informations de nature très variées sur un seul support, sous forme de bits. La distinction élémentaire, abstraite, entre deux états possibles de n'importe quel dispositif permet une combinatoire suffisante pour apparemment tout «coder». Loin d'être limitée, cette stratégie a rendu possible l'apparition du multimédia (qui associe textes, sons et images) dans les CD-Rom ou l'Internet et donné naissance aux «autoroutes de l'information».

2.10 En guise de conclusion : le monde est-il discret ou continu ?

A la fin du siècle dernier, le mathématicien allemand d'origine russe Georg Cantor a démontré que l'ensemble infini (et discret) des nombres entiers $\mathbb{N}$ est fondamentalement moins «puissant» que l'ensemble infini (et continu) des nombres réels $\mathbb{R}$, c'est-à-dire qu'il est impossible de définir une bijection entre ces deux ensembles. Il existe en quelque sorte *plusieurs niveaux d'infinis*.

Notre monde physique, lui, est-il discret ou continu ? La réponse n'a rien d'évident. Les physiciens ont tendance à écrire et à manipuler des équations dont les variables parcourent un espace continu et dont les fonctions sont aussi continues. Le temps, par rapport auquel ils dérivent souvent les autres fonctions, est ainsi généralement considéré comme continu. Pour eux, le discret est plutôt conçu comme une approximation du continu. Pourtant, la notion de «particule élémentaire» laisse penser à une nature discrète des éléments constituant l'univers. Mais la physique quantique a aussi découvert que particules (discrètes) et ondes (continues) ne sont que les deux aspects d'une même réalité, ce qui ne contribue pas à simplifier les choses.

Il y a pourtant au moins deux domaines naturels fondamentaux dans lesquels le mode numérique a prévalu sur le mode analogique :

- le code génétique : le patrimoine génétique de tous les êtres vivants est codé non pas à l'aide de 0/1 mais sur l'alphabet à quatre lettres des bases chimiques notées A, T, C et G. C'est donc un codage discret en base 4 ;
- les langues humaines : elles sont toutes constituées, à chacun de leur niveaux (les sons élémentaires utilisés par une langue donnée ou phonèmes, les mots, les règles de grammaires...), d'une combinatoire d'éléments discrets.

Ces deux domaines sont ceux qui permettent la *transmission d'information* (génétiques ou culturelles) d'une génération à une autre. Si la nature a sélectionné des mécanismes discrets pour réaliser cette transmission, c'est sans doute que le codage numérique a de meilleures propriétés que le codage analogique. L'exercice 1 (énoncé en 1.1, corrigé en 2.1) donne quelques éléments pour comprendre comment la transmission de données discrètes peut être rendue robuste par des mécanismes d'auto-correction.

Par ailleurs, ce n'est évidemment pas un hasard si l'informatique est de plus en plus utilisé pour aider l'expertise humaine dans ces domaines, puisque le codage peut y être exact...

3 Traitements effectifs

La représentation binaire des nombres était connue bien avant l'apparition des ordinateurs ; elle fut inventée dès le XVIème siècle par Francis Bacon, et fut utilisée par Leibniz au XVIIème siècle. Evidemment, le codage numérique des autres types de données (caractères, images, sons...) ne s'est généralisé que récemment, mais les principes qu'il met en oeuvre ne sont pas fondamentalement difficiles. Ce qui, en revanche, a signé l'acte de naissance de l'informatique, c'est l'explicitation de la notion de *traitement effectif*, ou encore de *procédure de calcul* ou d'*algorithme* (par la suite, ces termes seront utilisés comme des synonymes), qui permet de décrire ce que réalise un ordinateur sur les données qu'on lui fournit. Nous avons montré comment l'on pouvait ramener la diversité des données à des composants élémentaires. Nous allons maintenant voir que l'on peut aussi ramener la variété des traitements réalisables par les ordinateurs à *une combinatoire de traitements élémentaires*.

3.1 Préhistoire des algorithmes

Qu'est-ce qu'un algorithme ? En première approximation, c'est une *méthode systématique* définie *étape par étape* et permettant de résoudre *à coup sûr* et *en un nombre fini d'étapes* une certaine *classe de problèmes* ou de répondre à une certaine *classe de questions*. Chacune de ces caractéristiques va bien sûr devoir être explicitée. De telles méthodes ont été découvertes bien avant l'apparition de l'informatique et sont connues depuis longtemps par les mathématiciens. L'enseignement des mathématiques passe d'ailleurs par l'apprentissage de nombreux algorithmes : par exemple, comment réaliser une opération élémentaire (addition, multiplication, etc.) sur 2 nombres, entiers ou décimaux, comment résoudre une équation, etc.

Partons du problème suivant, un peu plus difficile : comment savoir si un certain nombre entier n (n'importe lequel) est un nombre premier, c'est-à-dire s'il n'admet

aucun autre diviseurs que lui-même et 1 ? Pour le savoir, on peut adopter la stratégie suivante :

- essayer de diviser n par 2 :
  $\Rightarrow$ si la division est juste (le reste est nul), alors n est divisible par 2 : *n n'est pas premier*
- sinon, essayer de diviser n par 3 :
  $\Rightarrow$ si la division est juste (le reste est nul), alors n est divisible par 3 : *n n'est pas premier*
- ... essayer de diviser n par... (essayer tous les nombres jusqu'à $n-1$) :
  $\Rightarrow$ si *au moins une* division est juste : *n n'est pas premier*
  $\Rightarrow$ si *aucune* division n'est juste : *n est premier*

Cette stratégie est très élémentaire (il en existe de bien plus efficaces : par exemple, on peut s'arrêter quand on dépasse $\sqrt{n}$), mais elle assure :

- de toujours savoir ce qu'il faut faire pour obtenir la réponse ;
- de fonctionner quel que soit le nombre entier n de départ : elle répond bien à une *classe* de questions (comme «un nombre entier quelconque est-il premier ?») et non à une question particulière (comme «le nombre 31457 est-il premier ?») ;
- de demander un nombre fini de calculs (qui dépendra néanmoins du nombre n que l'on teste) et de donner toujours une réponse correcte, qu'elle soit positive ou négative.

Un algorithme peut aussi être vu comme la réalisation concrète d'une certaine fonction : dans notre exemple, c'est la fonction f qui à chaque nombre entier associe la réponse «oui» ou «non» suivant que ce nombre est premier ou non. On peut coder cette réponse par 0 ou 1 (0 signifie «non», 1 signifie «oui»). Le nombre n à tester est la *donnée d'entrée*, tandis que la réponse finale est la *donnée de sortie* de l'algorithme. On peut donc représenter cette fonction ainsi :

$$\begin{array}{rccc} f : & \mathbb{N} & \longrightarrow & \{0,1\} \\ & n & \longmapsto & f(n) \end{array}$$

Notre exemple illustre aussi qu'un algorithme est défini à l'aide d'*instructions élémentaires* (comme de savoir faire une division), de *tests* (vérifier si le reste de cette division est nul ou non) et d'une *structure* définie par la façon et l'ordre dans lesquels il s'enchaînent.

Un des plus anciens algorithmes intéressants connus (légèrement plus compliqué que celui de l'exemple), appelé «algorithme d'Euclide» et datant environ de 300 avant J.C., permet de répondre à la classe de questions suivante : comment savoir si deux nombres entiers quelconques sont premiers entre eux (c'est-à-dire s'ils ont un diviseur commun) ?

Le mot «algorithme», lui, a été créé à partir du nom de Al Khowarizmi, mathématicien persan du IXème siècle, connu pour être l'auteur en 825 d'un traité d'arithmétique où il transmettait aux arabes des algorithmes de calculs connus des indiens et utilisant la numérotation de position.

La notion d'algorithme existe donc depuis longtemps. Mais la nécessité de lui donner un contenu, une définition mathématique précise n'a émergé que très tardivement, pour répondre à des interrogations plus générales.

3.2 Les problèmes de Hilbert

L'histoire contemporaine des algorithmes commence en 1900. Cette année-là, a lieu à Paris le grand congrès de la société des mathématiciens. L'invité d'honneur y est David Hilbert, très grand mathématicien allemand de l'époque. Pour faire honneur à la date de la conférence, il choisit de se livrer à l'exercice périlleux de la prospective, en proposant une liste de problèmes fondamentaux qui, selon lui, vont dominer les mathématiques du XXème siècle : ce sont les désormais célèbres «23 problèmes de Hilbert». Il n'est bien sûr pas question de tous les exposer ici, mais il suffit de savoir qu'ils ont effectivement recensé une bonne partie des recherches mathématiques à venir.

Celui qui nous intéresse particulièrement porte le numéro 10, et il s'énonce de la façon suivante : «Existe-t-il une méthode pour résoudre n'importe quelle équation diophantienne ?». Une équation diophantienne s'exprime sous la forme d'un polynôme dont les coefficients sont des nombres entiers et dont on cherche les racines entières.

Example 3 *Quelques exemples d'equations diophantiennes :*
chercher les valeurs entières de x telles que : $3x^2 - 5x + 1 = 0$
chercher les valeurs entières de x et y telles que : $6xy^2 - y + 7x = 0$

Dans certains cas particuliers (comme dans le premier exemple), on sait résoudre le problème, mais dans la plupart des autres aucune méthode générale, aucun *algorithme*, n'est connu et on commence à soupçonner qu'il n'en existe peut-être pas. Or les mathématiciens savent depuis longtemps démontrer qu'une certaine stratégie, une certaine méthode de calcul qu'ils ont inventée est juste et aboutira toujours au résultat souhaité, mais comment démontrer qu'il n'en existe aucune pour répondre à un problème donné ? Comment être sûr d'avoir testé toutes les méthodes possibles si on ne fait pas de cette notion de «méthode» elle-même un objet mathématique sur lequel il sera possible de faire des raisonnements et des démonstrations ? C'est l'enjeu de la formalisation de la notion d'algorithme, qui va se préciser au fil des ans.

Ainsi en 1928, à Bologne, Hilbert, de nouveau invité au congrès des mathématiciens (qui a lieu tous les quatre ans jusqu'à aujourd'hui), précise les objectifs de sa recherche, de son «programme». A cette époque, il espère fonder les mathématiques (qui ont traversé au début du XXème siècle une «crise des fondements» due à la découverte de paradoxes dans certaines théories) à l'aide de la logique, alors en plein essor. Sans entrer dans les détails, la logique est un langage permettant d'exprimer des énoncés mathématiques et de démontrer qu'ils sont vrais ou faux. Mais pour lui faire jouer un tel rôle, il faut s'assurer qu'elle-même est «bien fondée», c'est-à-dire qu'elle est consistante (elle ne permet pas d'aboutir à des contradictions) et complète (toute formule logique doit être soit vraie soit fausse). Il serait bon aussi qu'elle soit décidable, c'est-à-dire qu'il existe une procédure effective permettant de savoir si une

formule logique donnée quelconque est vraie ou fausse ; c'est ce que Hilbert appelle le «Entcheidungsproblem» ou «problème de la décision».

Malheureusement pour Hilbert, le logicien autrichien Kurt Gödel démontre en 1929 et surtout 1931 dans un article devenu très célèbre que si la logique élémentaire est effectivement complète, tout langage assez puissant pour fonder l'arithmétique est, lui, nécessairement incomplet, et que sa consistance est indémontrable. Le «programme de Hilbert» est plutôt mal parti. Reste le problème de la décision, qui ne trouvera sa solution, négative, qu'en 1936. Pour y parvenir, il a en effet fallu définir précisément ce qu'est une procédure effective et montrer qu'aucune d'entre elles ne peut résoudre ce problème. C'est ce qu'est parvenu à faire Alan Turing, auteur de l'article qui met fin à l'Entcheidungsproblem.

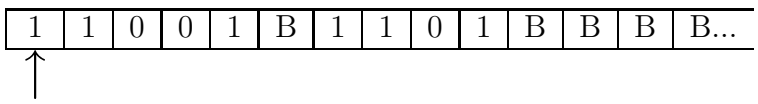
3.3 La machine de Turing

En 1936, Alan Turing a 24 ans. C'est un jeune et brillant étudiant en mathématiques à Cambridge, en Angleterre. Son professeur de logique a assisté à la conférence de Hilbert à Bologne, et lui-même est fasciné par la capacité humaine de raisonner et de faire des calculs. Plus tard, il cherchera à construire un «cerveau artificiel».

Dans son article de 1936, il propose tout d'abord de formaliser la notion d'algorithme grâce à la définition d'un dispositif abstrait que depuis on appelle «machine de Turing». Il montre ensuite qu'aucun de ces dispositifs ne sera jamais capable de décider si une formule logique quelconque est vraie ou fausse. La «machine de Turing» est le modèle de base qui fonde l'informatique, nous allons donc détailler sa construction et son fonctionnement.

Pour introduire cette définition de façon intuitive, essayons de décomposer un calcul en «atomes» le plus élémentaires possible, en recensant tout ce qui est nécessaire à son exécution. L'objectif est de construire la machine la plus simple possible capable de réaliser n'importe lequel de ces calculs. Soit par exemple une multiplication entre deux nombres entiers : $25 * 13$.

Pour réaliser une telle opération, il faut d'abord «poser le calcul», en général en l'écrivant sur une feuille de papier. De combien de papier a-t-on besoin ? On se restreint à des opérations portant uniquement sur des données discrètes. Par définition, elles peuvent donc être codées à l'aide d'un alphabet fini : 0 et 1 par exemple ! Les deux dimensions de la feuille de papier ne sont pas fondamentales : il suffit donc de disposer d'un ruban de papier séparé en cases, chaque case pouvant contenir soit le symbole 0, soit le symbole 1, soit rester vide (caractère noté B pour «blanc», et servant à séparer les différentes données). Les données de notre multiplication peuvent ainsi être «posées» de la façon suivante :



L'accès à ces données se fait par le biais d'une *tête de lecture/écriture* qui a le droit de parcourir le ruban case par case, un peu comme dans un magnétophone ou un magnétoscope. Nous la notons par une flèche sous la case du ruban qui est en train d'être lue (dans le dessin précédent, la tête se trouvait en train de lire la

première case). C'est l'équivalent de la pointe du crayon avec lequel on écrit quand on effectue le calcul à la main.

Pour la suite des calculs, nous aurons besoin de poser d'autres données correspondant à des résultats intermédiaires. De même qu'on peut toujours disposer d'une nouvelle feuille de brouillon, on doit toujours pouvoir disposer de cases libres sur le ruban : nous supposons donc que celui-ci est *infini vers la droite* (et contenant initialement, à l'infini, des cases notées B).

Le deuxième composant fondamental d'une machine de Turing est la notion d'*état*. En effet, pour réaliser notre multiplication, nous devons passer par plusieurs *phases*, plusieurs *étapes* au cours desquelles l'opération réalisée est différente. Chacune suppose en quelque sorte un *état d'esprit* différent. Dans notre exemple, au moins deux phases successives sont nécessaires :

- une phase de multiplications élémentaires entre chiffres : $3 * 5$ puis $3 * 2$ puis $1 * 5$ puis $1 * 2$;
- une phase d'additions élémentaires entre les résultats de la phase précédente.

De même, une machine de Turing disposera d'*un ensemble fini d'états distincts*, que par la suite nous repérerons par des nombres disposés les uns à la suite des autres dans un tableau. A chaque instant un et un seul de ces état est «actif» et on l'appelle «état courant» : c'est celui dans lequel se trouve la machine à cet instant-là. La machine peut passer de n'importe quel état à n'importe quel autre à chaque étape du calcul. Enfin, un état particulier appelé «état final» et noté **F** correspondra à l'état dans lequel se trouve la machine à la fin de son calcul. Il signifie l'arrêt de son fonctionnement. Ainsi, si une machine a trois états possibles (plus **F**) et se trouve en ce moment dans l'état **2**, nous notons :

1	(**2**)	**3**	**F**

L'état courant est repéré par un cercle. La machine commence toujours en partant de l'état initial noté **1**, et s'arrête dès qu'elle atteint l'état **F**.

On appelle *configuration* d'une machine de Turing l'ensemble constitué par le symbole du ruban sur lequel pointe sa tête de lecture et l'état courant dans lequel elle se trouve. La machine donnée en exemple jusqu'à présent serait ainsi dans la configuration : (symbole lu : 1, état : 2). Réaliser un calcul avec un tel dispositif consiste, fondamentalement, à *enchaîner des changements de configuration* à partir des données de départ, pour aboutir à de nouvelles données.

Pour comprendre le fonctionnement d'une telle machine, reprenons notre exemple de multiplication. Chaque étape du calcul revient à écrire de nouveaux chiffres sur la feuille blanche en consultant (mentalement ou matériellement) une *table* de multiplication ou une table d'addition, suivant l'étape de calcul dans laquelle on se trouve.

De même, dans une machine de Turing, chaque instruction élémentaire revient simplement à *substituer* certains symboles à d'autres sur le ruban, en tenant compte des résultats intermédiaires qui y sont déjà écrits et de l'état courant de la machine, en consultant une *table*.

Pour une configuration donnée, c'est-à-dire :

- le contenu de la case du ruban *lue* par la tête de lecture ;

– l'état courant de la machine.

une instruction élémentaire sera, en effet, définie par 3 composantes :

– le symbole *écrit* par la tête de lecture à la place de celui qu'elle a lu ;
– le déplacement de la tête de lecture : soit une case vers la gauche (G) soit une case vers la droite (D) ;
– le nouvel état courant du calcul.

La première et la troisième composante décrivent la nouvelle configuration de la machine, la deuxième composante est le déplacement minimal qui permet d'enchaîner les instructions élémentaires les unes après les autres. La liste des instructions élémentaires constituant un calcul complet peut donc bien figurer dans un *tableau* dont les deux paramètres d'entrée sont le symbole lu et l'état courant, et dont chaque case contient 3 symboles correspondant aux 3 composantes de chaque instruction.

En résumé, un calcul effectué par une telle machine commence une fois les données d'entrée écrites sur le ruban, la tête de lecture mise sous la première case du ruban et l'état **1** fixé comme état courant. Chaque étape du calcul consiste à chercher dans le tableau l'instruction à exécuter et à l'effectuer, jusqu'à temps que l'état courant soit l'état **F**. La donnée de sortie de l'algorithme, c'est-à-dire le résultat du calcul, doit alors être lisible sur le ruban.

3.4 Un exemple simple

La machine à multiplier les nombres (décimaux ou binaires) est malheureusement trop compliquée à écrire. Nous en prenons une beaucoup plus simple ici, celle réalisant la fonction suivante :

$$\begin{aligned} f : \quad & \mathbb{N} \longrightarrow \mathbb{N} \\ & n \longmapsto n+1 \end{aligned}$$

Notre machine de Turing doit donc ajouter 1 à un nombre binaire quelconque, écrit sur son ruban, infini dans les deux sens (le fait que le ruban soit infini dans les deux sens est aussi choisi pour simplifier l'écriture de la machine). Dans cette machine, le résultat du calcul (le nombre initial + 1) va être écrit *à la place de la donnée de départ* sur le ruban. Son tableau est le suivant :

état courant \ symbole lu	0	1	B
1	$0, D, \mathbf{1}$	$1, D, \mathbf{1}$	B,$G, \mathbf{2}$
2	$1, D, \mathbf{F}$	$0, G, \mathbf{2}$	$1, D, \mathbf{F}$

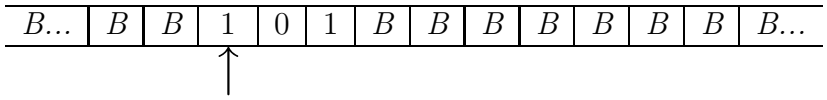
Par exemple, la case qui se trouve à l'intersection de la colonne notée 1 et de la ligne notée **2** contient l'instruction qui doit être exécutée si la tête de lecture est en face d'une case du ruban contenant le symbole 1 et que la machine se trouve dans l'état **2** (on note les états en gras pour les distinguer des symboles écrits sur le ruban). Dans notre tableau, cette instruction est : $0, G, \mathbf{2}$. Ce code signifie que l'instruction consiste alors à écrire avec la tête de lecture/écriture le symbole 0 dans la case (à la place du 1), puis à déplacer cette tête d'une case vers la gauche sur le ruban, tandis que l'état courant de la machine reste l'état **2**.

Pour vérifier l'exactitude de cette table, il suffit de la tester en écrivant sur le ruban un nombre binaire (positif) quelconque, en plaçant la tête de lecture sous la première case de ce nombre et en fixant l'état courant à **1**, puis en exécutant systématiquement les instructions de la table. Quand l'état **F** est atteint, le nouveau nombre écrit sur le ruban doit être le nombre initial +1.

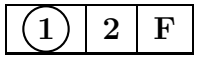
Prenons l'exemple du nombre $n = 5$, et donc de l'addition suivante :

$$\begin{array}{cccc} & & 1 & \\ & 1 & 0 & 1 \\ + & & & 1 \\ \hline & 1 & 1 & 0 \end{array}$$

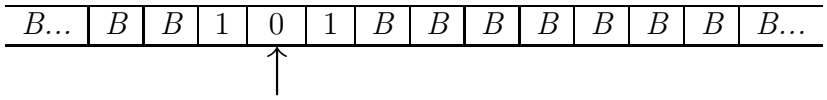
Example 4 *Au début du calcul, le ruban contient le nombre* 5*, codé en binaire comme suit :*

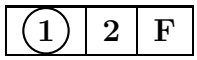


Et l'état courant est l'état **1** *:*

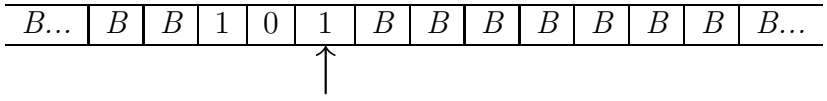


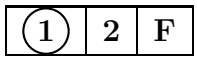
L'instruction à exécuter se lit à l'intersection de la colonne 1 *(symbole lu par la tête de lecture) et de la ligne* **1** *(état courant) : c'est* $1, D, \mathbf{1}$*. Le symbole* 1 *du ruban est recopié, et la seule modification à effectuer est donc de déplacer la tête de lecture vers la droite (l'état courant restant aussi inchangé) ; la nouvelle configuration de la machine à l'issue de cette instruction est donc :*



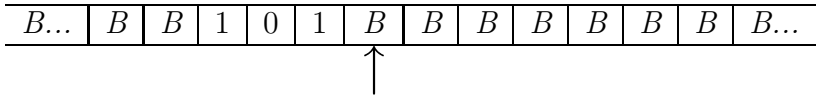


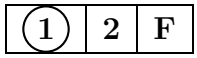
L'instruction suivante à exécuter est : $0, D, \mathbf{1}$*, qui donne lieu à la situation :*





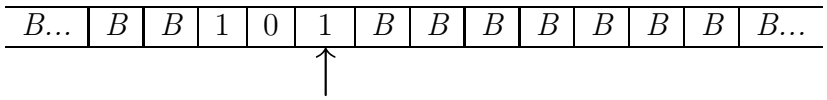
Puis, il faut de nouveau exécuter $1, D, \mathbf{1}$*, et on arrive à :*





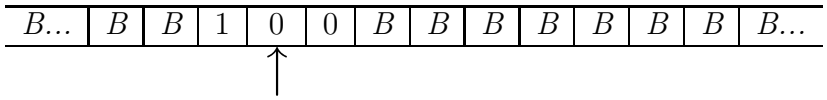
En fait, cette première série d'instructions n'a fait que parcourir le nombre en le recopiant : cette étape était nécessaire au repérage de la fin du nombre (par où

commence l'addition). Cette fois, la tête de lecture pointe devant une case blanche ; l'instruction à exécuter est $B, G, \mathbf{2}$*, qui ne modifie toujours pas le contenu du ruban, mais change l'état courant :*



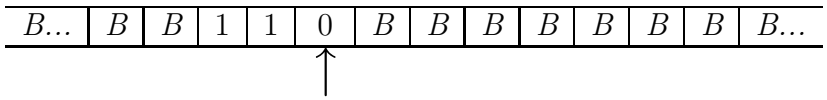
1	**(2)**	**F**

L'addition va pouvoir commencer : l'instruction $0, G, \mathbf{2}$ *réalise l'addition du dernier chiffre du nombre écrit sur la ruban avec* 1*.*



1	**(2)**	**F**

Il reste néanmoins une retenue à propager vers la gauche. C'est ce que réalise l'instruction : $1, D, \mathbf{F}$*, qui mène à la configuration finale :*



1	**2**	**(F)**

L'état $\mathbf{F}$ *étant atteint, le calcul s'arrête. Le ruban contient la suite de symbole* 110 *correspondant au nombre* 6 *en codage binaire, qui est bien le résultat de l'opération* $5 + 1$*.*

En fait, quel que soit le nombre écrit au départ sur le ruban, le résultat de l'exécution des instructions du tableau amènera à écrire ce nombre +1 sur le ruban : la machine de Turing réalise donc bien une fonction, un algorithme.

Les instructions de la table peuvent aussi être représentées dans un *graphe*, c'est-à-dire un schéma constitué d'*états* (représentés par des ronds) et d'*arcs étiquetés* reliant ces états avec les conventions suivantes :

- les états du graphe correspondent aux états possibles de la machine de Turing ;
- une instruction est figurée par un arc reliant l'état de départ de l'instruction et l'état dans lequel elle mène, étiqueté par 3 symboles :
  - le symbole lu dans la case par la tête ;
  - le symbole écrit dans la case par la tête ;
  - le déplacement de la tête.

Ainsi, les instructions de la machine précédente peuvent également être représentées par le graphe de la figure 2.9. En fait, la définition d'une machine de Turing est la donnée d'un ensemble de *changements de configuration*. La définition d'un tel changement requiert la donnée de 5 informations : la configuration initiale (2 informations), la configuration finale (2 informations) et le déplacement (à gauche ou à

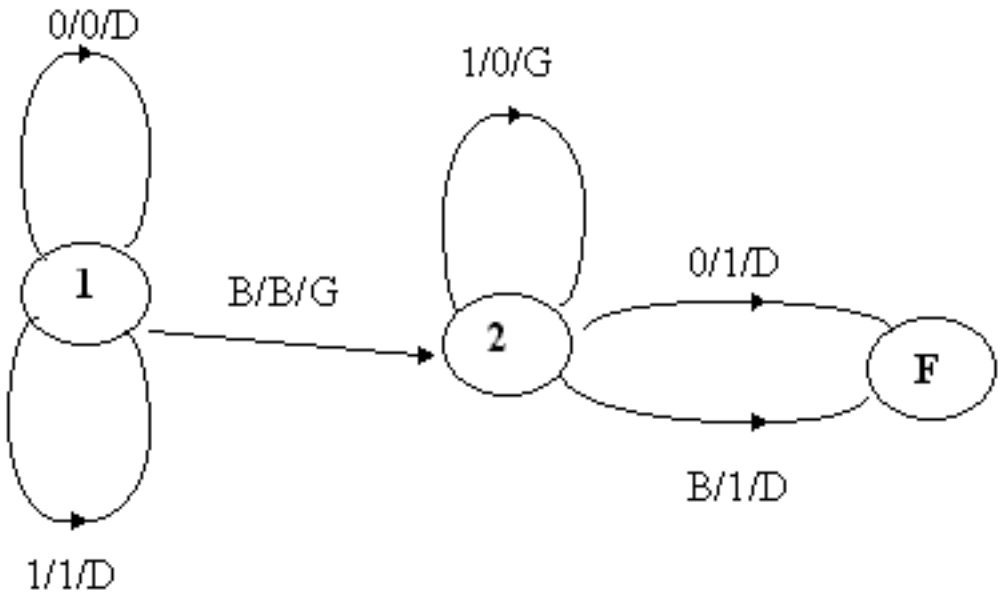


FIG. 2.9 – graphe de la machine de Turing à ajouter +1

droite). Le tableau ou le graphe sont deux moyens possibles de représenter des ensembles de changements de configuration (attention, dans les 2 cas, les informations en sont pas données dans le même ordre).

Les états de cette machine ont le sens suivant :

- l'état **1** est simplement la lecture et la recopie de gauche à droite du nombre écrit sur le ruban, on le quitte quand on rencontre le premier symbole B, c'est-à-dire quand on atteint la fin du nombre ; en revenant une case à gauche, on se place devant le dernier chiffre du nombre ;
- l'état **2** est celui dans lequel l'addition +1 s'effectue vraiment :
  - si le nombre se termine par 0, il suffit de transformer le 0 final en 1 pour ajouter 1 à ce nombre, et de terminer le calcul en rejoignant l'état **F** ;
  - si le nombre se termine par 1, alors le chiffre final devient 0 et il faut propager la retenue vers la gauche en restant dans l'état **2** ;
  - si on rencontre un symbole blanc B, cela signifie que l'on a atteint le début du nombre en propageant la retenue vers la gauche, et qu'il suffit donc maintenant d'écrire cette retenue pour terminer le calcul en rejoignant l'état **F**.

3.5 La thèse de Church-Turing

Une machine de Turing est un dispositif très rudimentaire qui réalise une *fonction*, puisqu'il transforme des données figurant sur un ruban en de nouvelles données, elles aussi écrites sur le ruban. La machine fonctionne étape par étape, et s'arrête quand elle atteint l'état **F**. Elle réalise donc bien ce qui ressemble à notre première

définition des algorithmes. Mais Turing va plus loin. Il affirme que *tout algorithme peut être décrit de cette façon, à l'aide d'une machine qui s'arrête toujours*. Cette affirmation est connue sous le nom de thèse de Church-Turing. Elle signifie en particulier que *tous les traitements cités dans la dernière colonne du tableau 2.1 peuvent être exprimés par une certaine machine de Turing*. On comprend mieux, maintenant, ce qui fait l'intérêt de cette definition. Mais cette «thèse» demande quelques justifications.

Tout d'abord, remarquons qu'il est facile de construire une machine de Turing qui ne s'arrête jamais, qui «boucle à l'infini» : il suffit par exemple que la seule instruction figurant dans le tableau, quel que soit le contenu de la case lue et quel que soit l'état courant, consiste à écrire 0 sur le ruban et à déplacer la tête de lecture vers la droite (sans changer d'état courant). Le ruban étant par définition infini vers la droite, l'exécution de cette machine sur un ruban contenant une donnée quelconque ne finira jamais. Or, un algorithme s'arrête toujours, il ne peut donc coïncider qu'avec celles des machines de Turing qui s'arrêtent aussi toujours.

Notons ensuite que c'est bien une *thèse* qui est énoncée, et non un théorème mathématique. Cette thèse est, par nature, indémontrable puisque la notion d'algorithme, avant elle, n'avait pas de définition mathématique précise à quoi on aurait pu comparer celle des machines de Turing. La thèse de Church-Turing pose donc en fait une définition : elle introduit un nouvel objet mathématique (les machines de Turing) pour caractériser formellement une notion intuitive ancienne (la notion d'algorithme). Nous verrons plus loin ce qui, à défaut de preuves, tient lieu d'arguments en sa faveur.

La paternité de cette thèse est partagée entre Turing et Church. Alonzo Church, en effet, était un logicien américain qui, la même année que Turing, en 1936, a proposé un autre formalisme mathématique (extrêmement différent de celui de Turing) permettant également de décrire la notion de procédure effective. Ce formalisme, appelé le lambda-calcul (que nous ne détaillerons pas ici), s'est révélé être équivalent à celui des machines de Turing : toute méthode de calcul pouvant être décrite avec l'un peut l'être avec l'autre. Il est donc naturel de les associer.

Enfin, comment se convaincre de la validité de cette thèse ? Elle est maintenant largement admise et de nombreux arguments plaident en sa faveur :

- arguments intuitifs : en parlant d'états d'esprit et en partant de l'exemple d'un calcul simple (ce que Turing faisait déjà dans son article), nous avons essayé de favoriser l'intuition ; de façon générale personne n'a jamais pu exhiber une procédure de calcul réalisable par un esprit humain qu'une machine de Turing ne pourrait pas elle-même réaliser.
- argument de l'extension des machines de Turing : les machines de Turing sont rudimentaires, et il est possible d'élargir leur définition pour permettre, par exemple :
  - d'utiliser plus de symboles différents sur le ruban (par exemple l'ensemble des caractères alphanumériques) : mais nous avons vu qu'à l'aide de 0/1 on pouvait coder n'importe quel alphabet discret ;
  - de déplacer la tête de lecture plus librement en « sautant » par dessus les cases : mais en sautant case par case on peut aller aussi loin que l'on veut ;
  - de disposer d'un ruban infini dans les deux sens (ce que nous avons utilisé

dans l'exemple, pour qu'il soit le plus simple possible), et même de plusieurs rubans en parallèle : mais un ruban infini vers la droite uniquement suffit pour tout calcul ;

– de donner le choix entre plusieurs instructions possibles dans chaque case du tableau et en choisissant au hasard celle qui doit être exécutée (on parle alors de machine «non déterministe») : mais, là encore, on n'augmente pas fondamentalement la capacité de calcul.

Pour chacune de ces extensions, il a été *démontré* qu'elle ne permettait pas de réaliser un calcul qui, sur une machine classique, aurait été impossible. Elles permettent sans doute parfois d'aller plus vite et donc d'aboutir plus rapidement au résultat de l'algorithme, mais jamais d'aboutir à un résultat différent.

– argument des autres formalismes : depuis 1936, de nombreux autres modèles de calcul ont été proposés pour caractériser la notion d'algorithme (comme le lambda-calcul, déjà évoqué). Tous se sont révélés avoir une puissance de calcul exactement équivalente à celle des machines de Turing, qui tient donc toujours lieu de modèle de référence.

3.6 La notion de machine universelle

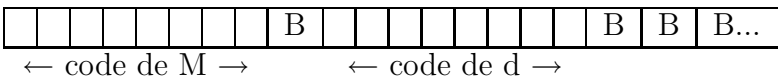
Une machine de Turing *n'est pas un ordinateur*. Chaque machine particulière représente un algorithme particulier, servant à résoudre un problème précis. Une machine de Turing est donc plutôt une méthode de calcul, un *programme*.

Mais définir une machine de Turing, c'est équivalent à définir le tableau d'instructions de cette machine. Chaque case d'un tel tableau contient un triplet de données discrètes, qui est lui-même une donnée discrète. Or, nous avons vu qu'un tableau de données peut être codé (cf. 2.5). Ainsi, une machine de Turing *peut elle-même être codée par une suite de* 0/1. Ce code peut à son tour être écrit sur un ruban et donc *servir de donnée à une autre machine*.

Turing a démontré que parmi les machines qui portent désormais son nom, il en existe certaines particulières qu'il a appelées des Machines Universelles et que nous noterons U. De telles machines sont caractérisées par le fait que quand, sur leur ruban, figure successivement :

– le code d'une certaine machine de Turing, notée M ;

– le code d'une certaine donnée d ;



Le résultat de l'exécution de U sur ces données est le même que celui qu'on aurait obtenu en exécutant les instructions de M sur la donnée d. Ainsi U(M, d)=M(d).

Une machine universelle est donc en quelque sorte capable de *simuler* le comportement de n'importe quelle autre machine, dont on lui fournit la description, sur n'importe quelle donnée, qu'on lui fournit également par ailleurs. Une telle machine est ainsi capable d'exécuter n'importe quel algorithme, pourvu qu'on lui en donne un code. Si vous êtres capable de «faire fonctionner» sur du papier une machine de Turing dont on vous fournit la description (sous la forme d'un tableau ou d'un

graphe) et sur le ruban de laquelle on a mis une donnée de départ, c'est que vous êtes devenu vous-même une machine universelle !

Cette notion est essentielle car elle est au fondement aussi bien du côté matériel que du côté logiciel de l'informatique.

En effet, on peut dire qu'*un ordinateur est une machine universelle* : quels que soient les données et les programmes qu'on lui fournit, il est capable d'exécuter ces programmes sur ces données. Le principe de la machine universelle se retrouve aussi dans l'architecture interne des ordinateurs (que nous détaillerons par la suite). En particulier, l'idée qu'il n'y a pas de différence de nature entre une donnée et une instruction, puisque chacune peut être codée par une suite de 0/1, se retrouve dans la mémoire des ordinateurs actuels, qui contient indifféremment des données et des programmes.

De même, on peut aussi dire qu'*un langage de programmation est une machine universelle* : il permet d'associer un code, une représentation, à n'importe quelle donnée et à n'importe quelle instruction, et d'intégrer le tout dans un programme. Tout langage de programmation permet de programmer, et donc de simuler, n'importe quelle machine de Turing.

Les machines de Turing sont néanmoins des modèles théoriques, abstraits, dont les réalisations matérielles ou logicielles ne sont que des approximations. Les principales limitations des dispositifs concrets sont *l'espace mémoire* disponible et *le temps de calcul*. Ainsi, alors que les machines de Turing disposent d'un ruban infini, les ordinateurs actuels ont évidemment une capacité de stockage finie. De même, le temps de calcul des machines de Turing réalisant un algorithme se doit d'être fini, mais aucune borne n'est fixée, tandis qu'un calcul concret nécessitant quelques centaines d'années doit être considéré comme irréalisable.

3.7 Indécidabilité

Revenons un moment sur les problèmes qui ont motivé la définition des machines de Turing. En effet, maintenant que nous disposons d'une définition mathématique de ce qu'est un algorithme, il devient possible d'explorer le domaine de l'*indécidable*, c'est-à-dire des problèmes qu'aucun algorithme se sera jamais capable de résoudre. De tels problèmes existent-ils ? Oui ! Et ils sont même très nombreux...

Le tout premier d'entre eux, exposé par Turing lui-même dans son article fondateur, est connu sous le nom de «problème de l'arrêt». Il s'exprime ainsi : existe-t-il un algorithme capable, quand on lui fournit le code d'une machine de Turing M et le code d'une donnée d, de prévoir si M fonctionnant avec d comme donnée d'entrée s'arrêtera ou non au bout d'un temps fini ? Remarquons que les machines universelles ne répondent pas totalement à la question puisqu'elles se contentent de simuler ce que ferait M sur la donnée d : si M s'arrête sur d, alors la machine universelle s'arrêtera aussi sur (M, d), mais si M «boucle à l'infini», alors la machine universelle aussi et elle aura été incapable de le prévoir...

Le problème de l'arrêt est indécidable : il est impossible de concevoir un algorithme capable de savoir à l'avance si une méthode de calcul appliquée à une certaine donnée s'arrêtera à coup sûr. Comme le fait de s'arrêter est précisément ce qui distingue les machines de Turing qui sont de vrais algorithmes de celles qui

n'en sont pas, cela signifie qu'*il est impossible de concevoir un algorithme capable de distinguer les vrais algorithmes de ceux qui n'en sont pas !* Nous ne détaillerons pas ici la démonstration de ce résultat, mais elle se rattache à la famille des paradoxes logiques dits du «menteur» (en disant «je suis un menteur», je mens donc «je dis la vérité» donc finalement c'est vrai que «je suis un menteur»...).

Ce premier problème indécidable peut sembler un peu abstrait et éloigné des «vrais problèmes» mathématiques. Pourtant, une de ses conséquences directes est qu'il n'existe pas non plus d'algorithme capable de décider si une formule logique quelconque est vraie ou fausse : c'est la solution de Turing au fameux Entcheidungsproblem.

Et, depuis 1936, de nombreux problèmes indécidables sont découverts chaque année, dont certains s'énoncent très simplement. Par exemple, certains problèmes de «pavage du plan» sont indécidables. Ces problèmes partent de la donnée de quelques figures géométriques (des polygones) et cherchent à remplir, à «paver» entièrement un plan avec ces figures sans laisser de blanc entre les figures. Il est impossible de savoir à l'avance si ce sera possible. De même, le «10ème problème de Hilbert», à l'origine lointaine de toute cette histoire (cf. 3.2), n'a finalement trouvé de solution qu'en 1970, quand le mathématicien soviétique Matijasevic a démontré qu'il n'existe aucun algorithme capable de résoudre toutes les équations diophantiennes de degré supérieur ou égal 5.

Il est important de comprendre qu'un résultat d'indécidabilité est une barrière théorique infranchissable : si un problème est indécidable, cela signifie qu'aucun algorithme ne pourra jamais le résoudre, quels que soient les progrès technologiques que pourront connaître les ordinateurs dans les années à venir.

3.8 L'héritage scientifique de Turing

L'article de 1936 de Turing est le point de départ d'une nouvelle discipline qu'on désigne depuis comme «l'informatique théorique». Un de ses objectifs est *la classification des problèmes en fonction de leur solvabilité effective*. La première distinction fondamentale introduite par Turing est le caractère décidable ou indécidable des problèmes.

Mais savoir qu'un problème est décidable (c'est-à-dire qu'il existe des algorithmes qui le résolvent) ne suffit pas, encore faut-il qu'il le soit *efficacement*, c'est-à-dire en utilisant des ressources (en temps de calcul et en espace mémoire) raisonnables. La difficulté est alors de définir un tel critère d'efficacité qui ne dépende pas de la machine sur laquelle sera exécuté l'algorithme, ou du langage de programmation dans lequel il sera écrit. C'est ce que réalise la théorie de la complexité.

Il ne peut être question d'expliquer ici cette théorie, contentons-nous donc d'en donner une simple intuition. Elle permet dans un premier temps de définir *la complexité d'un algorithme* particulier puis, dans un deuxième temps, *la complexité intrinsèque d'un problème*.

Rappelons qu'un algorithme sert à répondre à une classe de questions, du type : «étant donné un nombre quelconque, comment savoir s'il est ou non premier ?». Un algorithme réalise une certaine fonction qui, à chaque donnée d'entrée possible, associe une réponse unique. Or, il est évident que tester le caractère premier d'un

nombre sera d'autant plus long que le nombre en question est grand, surtout s'il est effectivement premier et que tous les calculs doivent être menés à terme avant d'aboutir à la réponse finale. La complexité d'un algorithme se mesure donc en fonction de la taille de sa donnée d'entrée (c'est-à-dire du nombre de bits nécessaires pour la coder) et, pour une taille fixée, en se plaçant *dans le pire des cas possibles*. Elle se mesure suivant deux fonctions, qui dépendent chacune de cette taille :

- le nombre d'opérations élémentaires de calcul nécessaires dans le pire des cas pour donner le résultat (complexité en temps) ;
- le nombre de cases du ruban de la machine de Turing nécessaires dans le pire des cas pour donner le résultat (complexité en espace).

Pour définir maintenant la complexité intrinsèque d'un problème, il ne faut pas oublier qu'il peut être résolu de plusieurs façons différentes, à l'aide de plusieurs algorithmes possibles. La complexité d'un problème sera donc rattachée à *la complexité du meilleur algorithme possible qui le résout*. Par exemple, si un problème peut être résolu par un algorithme dont la complexité en temps de calcul s'exprime par un *polynôme* en fonction de la taille des données, on dira qu'il est de complexité P. P désigne ainsi la *classe* de tous les problèmes satisfaisant cette propriété. Une hiérarchie de telles classes a pu être définie, permettant de ranger tous les problèmes suivant leur plus ou moins grande «solvabilité».

L'étude de la complexité des algorithmes et des problèmes est un thème de recherche actif, dans lequel de nombreux résultats sont découverts chaque année.

3.9 L'héritage philosophique de Turing

Au delà de ses aspects techniques, l'oeuvre de Turing pose des questions philosophiques fondamentales. La notion d'algorithme qu'il a définie caractérise en effet l'ensemble des calculs possibles réalisables par un dispositif mécanique. Cette notion concerne-t-elle aussi les calculs réalisables par un humain ? Autrement dit, l'esprit est-il une Machine de Turing Universelle ?

Personne, évidemment, ne prétend trouver dans le cerveau l'équivalent des composants des machines de Turing (ruban, tête de lecture, tableau...). En revanche, il est possible de se demander si les capacités de calcul et de raisonnement dont font preuve les hommes, quel que soit le mécanisme physique qui le réalise, peuvent s'exprimer de façon algorithmique. Cette interrogation est à la base de la *philosophie de l'esprit*, qui s'est surtout développée dans les pays anglo-saxons. Elle est aussi à l'origine de la *psychologie cognitive*, où les opérations mentales sont conçues en termes de traitement de l'information.

Une autre manière d'aborder la question consiste à se demander si les machines peuvent - ou pourront un jour - penser. Turing lui-même a apporté sa contribution à la réflexion épistémologique sur ce thème. En 1950, il a publié dans une revue philosophique un article où il cherchait à montrer qu'il n'y a aucune raison de dénier aux machines cette capacité. Dans ce texte, il passait en revue tous les arguments qui soutiennent le contraire et tentait de les réfuter minutieusement. Il proposait aussi un jeu, appelé depuis le «test de Turing», destiné à fonder un critère de reconnaissance de la pensée des machines. Ce jeu consiste pour un «examinateur» humain à dialoguer à distance, par un dispositif quelconque, avec un interlocuteur inconnu

qui peut être soit un autre humain soit une machine, et à identifier sa nature. Pour Turing, un ordinateur programmé de telle sorte que son comportement est indiscernable de celui d'un être humain, devrait être considéré comme aussi «intelligent» que lui.

Le projet de l'«Intelligence Artificielle» est en quelque sorte de réaliser un tel programme. Cette discipline née dans les années 50 (le terme date de 1956), cherche à écrire des algorithmes réalisant des opérations que l'on croyait jusque là l'apanage de l'homme : raisonnement logique, apprentissage, participation à des jeux de stratégie, compréhension du langage... Mais, alors que Turing prévoyait l'avènement de machines sortant vainqueur de son «test» vers l'an 2000, les réalisations concrètes de l'intelligence artificielle sont encore loin de telles performances, et de nombreux philosophes ou neurologues contestent maintenant la pertinence du modèle des machines de Turing pour expliquer le fonctionnement de l'esprit humain...

4 Conclusion

Que retenir de tout ce parcours ? On peut maintenant revenir à ce qui avait motivé tous ces développements. La distinction courante entre matériel et logiciel en informatique masque un niveau de description essentiel, qui n'appartient totalement ni à l'un ni à l'autre mais fonde l'un et l'autre, et qu'on pourrait appeler le niveau théorique ou logique. L'informatique peut ainsi être envisagée suivant *trois points de vue différents* et non deux seulement :

- au niveau matériel ou physique, des données sont stockées sur des supports magnétiques ou optiques et sont transformées par des impulsions électriques ;
- au niveau fonctionnel ou global, l'utilisateur a l'impression de manipuler des données à son échelle (que ce soit un tableau de nombres ou un personnage animé) auxquelles il applique des transformations macroscopiques ;
- au niveau théorique ou logique, qu'il faudrait intercaler entre les deux autres, les données manipulées sont des codes stockés dans des bits, et les traitements qui leur sont appliqués s'expriment par des algorithmes.

Même s'ils ne sont pas sensés ignorer les deux autres, c'est ce dernier niveau qui est la compétence principale des informaticiens. Le terme «informatique» lui-même a été proposé en 1962 par Philippe Dreyfus pour l'Académie Française. Ce mot est construit par contraction de «information» et «automatique» et sa définition officielle est «la science du traitement de l'information considérée comme le support formel des connaissances». Cette définition est nettement moins fausse que celles évoquées en 1. Si on veut être encore plus précis, on peut définir l'informatique comme *la science de tous les traitements effectifs applicables à des données discrètes.*

Concrètement, cette discipline se décline suivant de nombreuses variantes, allant de la plus théorique (étudier la complexité d'un problème) à la plus ludique (programmer un jeu). Mais elle a une constante : la démarche d'un informaticien confronté à un problème donné se caractérise toujours par un effort de *modélisation* qui opère à deux niveaux :

- d'une part en trouvant un équivalent discret, numériques aux données réelles du problème, en cherchant pour elles un bon codage, une bonne représentation ;

– d'autre part en exprimant sous forme de règles formelles, d'algorithmes, les conditions de modifications de ces données permettant d'aboutir à un résultat.

Cette méthodologie est résumée dans le schéma de la figure 2.10 (dû à Jacques Arsac, un des pionniers de l'informatique en France).

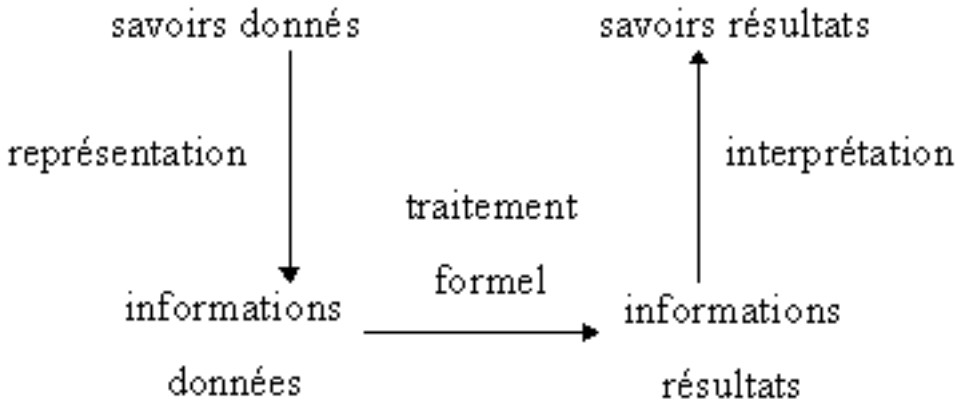


FIG. 2.10 – démarche de l'informaticien

Dans le cas par exemple d'un simulateur de vol, ou d'un programme de prévision météorologique, le résultat de cet effort sera d'autant plus réussi que l'application des algorithmes aux données codées reproduira les conditions «naturelles» d'évolution des données initiales dans le monde réel. La nouveauté, par rapport à une discipline comme la physique (qui cherche aussi modéliser par des équations les conditions du monde réel), c'est que cette démarche peut également s'appliquer aux opérations mentales, celles qui permettent de faire un calcul, de chercher une stratégie gagnante dans un jeu ou de traduire un texte d'une langue dans une autre. L'objet de l'informatique est de trouver un équivalent dans le monde virtuel de ces processus, de définir et de tester des modèles pour ces mécanismes.

Mais la simulation n'a pas de limite : rien n'oblige non plus les programmes informatiques à s'appuyer sur le monde réel. Ils permettent de simuler n'importe quel environnement, du moment que celui-ci est régi par des règles qui peuvent s'exprimer par des algorithmes. La voie est libre à l'imagination des créateurs de monde virtuels...

Si nous avons particulièrement insisté sur les aspects théoriques de l'informatique, c'était pour contrebalancer l'image trop «technique» qu'elle véhicule habituellement. Mais il est temps désormais de regarder d'un peu plus près le fonctionnement des ordinateurs réels.

Chapitre 3

Les ordinateurs

Le mot «ordinateur» a été créé en 1955 à la demande d'IBM, et tire son étymologie du terme moyenâgeux «ordonnateur», désignant l'autorité divine suprême... Plus prosaïquement, un ordinateur se définit comme une instance matérielle, concrète, d'une Machine de Turing Universelle (cf. 3.6). Il est donc capable, dans la limite de ses capacités en espace mémoire (nécessairement finies) et en vitesse de calcul, d'exécuter n'importe quel algorithme qu'on lui fournit sous forme de programme, sur n'importe quelle donnée discrète, qu'on lui fournit également. Il se distingue ainsi fondamentalement d'une simple machine à calculer par sa capacité à *enchaîner* plusieurs opérations en suivant des instructions paramétrables, permettant la réalisation d'opérations complexes non initialement «câblées». Toute la difficulté de conception d'un ordinateur vient donc de cette nécessité de lui faire exécuter des *suites* d'opérations, en synchronisant l'action de ses différents composants.

Nous allons passer en revue ici la nature de ces composants et leur organisation interne, en reprenant cette fois la distinction classique entre leur aspect matériel, «hardware» (qui se traduit littéralement par : «quincaillerie») et leur aspect logiciel, «software» (néologisme créé par opposition à «hardware»). L'intégration récente et de plus en plus systématique des ordinateurs dans des réseaux ajoute une couche de complexité supplémentaire à cette organisation, que l'on tâchera également d'introduire.

1 Le matériel

Nous nous en tiendrons ici la plupart du temps aux ordinateurs les plus répandus, à savoir les micro-ordinateurs proposés aux particuliers, de type «PC» (pour Personnal Computers) ou «Mac» (pour MacIntosh, leur constructeur).

1.1 Les composants externes

Depuis les années 90, les ordinateurs vivent dans un environnement complexe, illustré par la figure 3.1

Il convient d'abord de distinguer l'ordinateur lui-même de ses «périphériques», qui ne sont que des constituants annexes. Le coeur d'un ordinateur est constitué :

– de l'Unité Centrale (UC), ou «microprocesseur», appelé familièrement «puce» ;

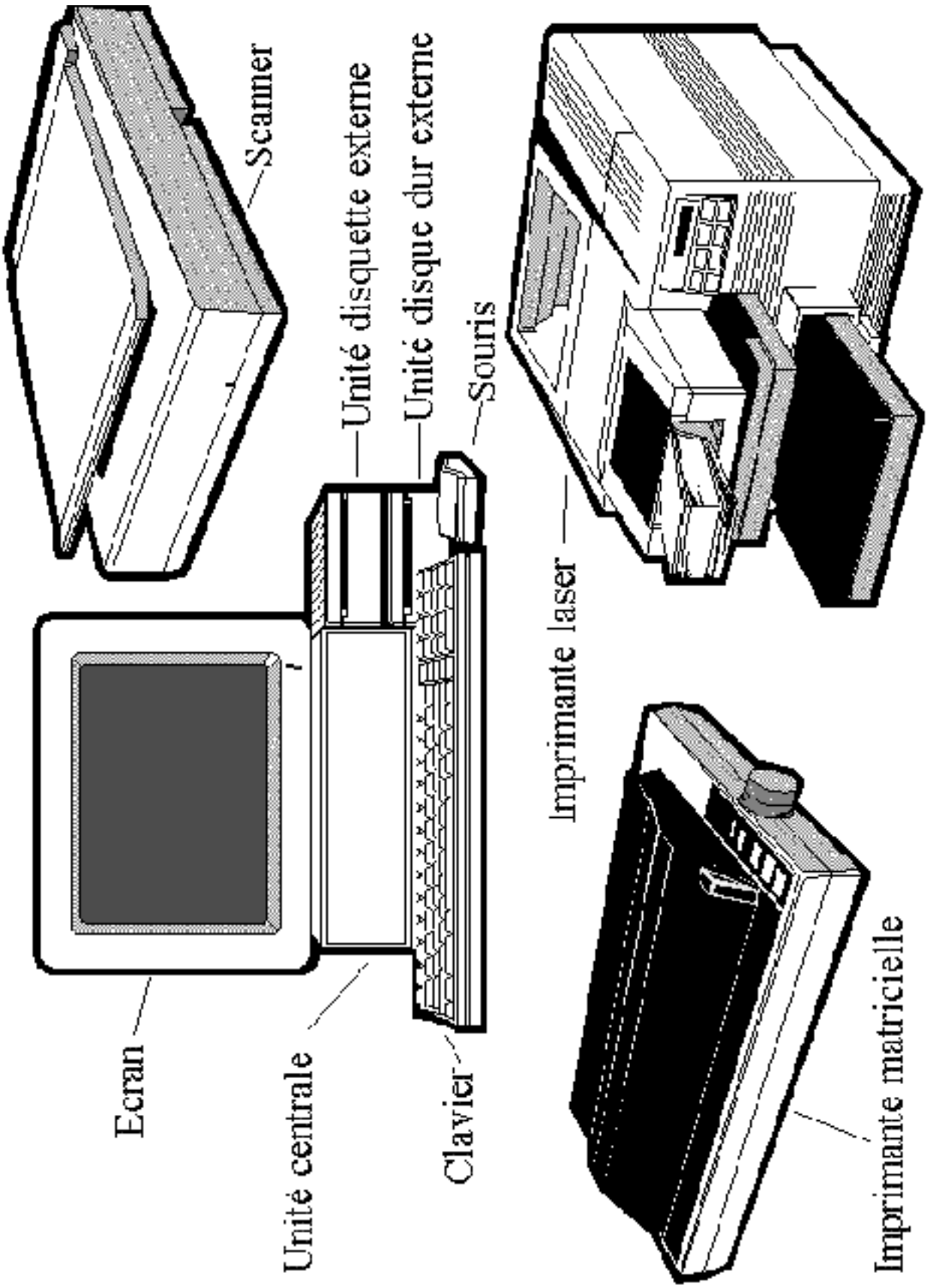


FIG. 3.1 – environnement matériel des ordinateurs

- de mémoires, parmi lesquelles on distingue plusieurs types :
  - la mémoire ROM (Read Only Memory : mémoire à accès en lecture seule) : ensemble de bits dont l'état est fixé une fois pour toute, lors de la construction de l'ordinateur. Elle sert à stocker des informations permanentes (procédures de démarrage...) ;
  - la mémoire RAM ou «mémoire vive» (Random Access Memory : mémoire à accès aléatoire) : ensemble de bits modifiables à volonté, où se trouvent stockées les données sur lesquelles travaille l'ordinateur. Il ne faut pas comprendre aléatoire dans le sens de «au hasard», mais par opposition à séquentiel ; cela signifie que l'on peut avoir accès directement à tout endroit de cette mémoire (comme les chansons dans un CD ou les chapitres dans un film en DVD), sans avoir à la parcourir bit à bit (comme dans les cassettes audio ou vidéo). Cette mémoire est volatile, c'est-à-dire qu'elle ne conserve les données que tant que la machine est sous tension. La mémoire vive des meilleurs ordinateurs actuels atteint 1Giga-octet.
  - les mémoires secondaires ou auxiliaires : ce sont des dispositifs permettant de stocker des bits de façon stable (qui reste fixée même si on éteint la machine) tout en étant généralement modifiable. On peut inclure parmi elles les disques durs, les disquettes, les bandes magnétiques, les clés USB... La capacité des disques durs actuels se compte en Giga-octets.

Les autres composants sont donc :

- soit des *périphériques d'entrée*, c'est-à-dire permettant à un utilisateur extérieur de *fournir des informations* (données/programmes) à la machine sous forme numérique : souris, clavier, scanner, joystick, appareil photo numérique, camescope numérique... Ces dispositifs peuvent tous être conçus comme des *numériseurs* puisqu'ils transforment un comportement (l'appui sur la touche d'un clavier, le mouvement de la souris) ou un objet (une photo analogique pour le scanner, un paysage pour les appareils de prise de vue numérique !) en une suite de bits.
- soit des *périphériques de sortie*, c'est-à-dire permettant de visualiser ou de transmettre des données internes à l'extérieur : écran, imprimante, IPod, vidéoprojecteur... A l'inverse des numériseurs, ces dispositifs traduisent des suites de bits en information interprétable par les humains.

La logique de cette organisation est donc celle de la figure 3.2

Les premiers ordinateurs, jusque dans les années 70-80, étaient réduits à une Unité Centrale et des mémoires. Les programmes et les données étaient alors exclusivement fournis sous forme de cartes perforées et les résultats d'un calcul se lisaient aussi sur des cartes perforées...

1.2 L'architecture de Von Neumann

Entrons maintenant dans l'organisation interne (on parle aussi «d'architecture matérielle») du coeur de notre ordinateur.

Si Turing peut être considéré comme le père de l'informatique théorique, l'homme à l'origine de la conception des ordinateurs actuels est John Von Neumann. Von Neumann, né en Hongrie, était un très grand mathématicien, ayant laissé sa trace

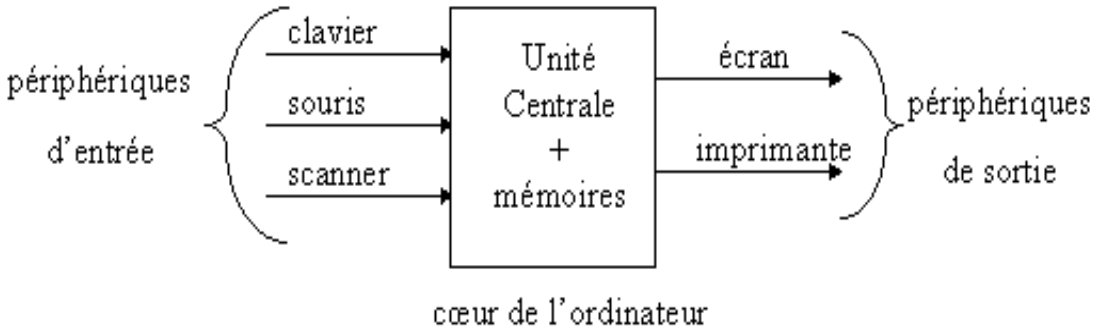


FIG. 3.2 – organisation générale d'un ordinateur

dans de nombreux domaines, y compris en physique (quantique) et en économie (théorie des jeux). D'origine juive, il a émigré aux Etats Unis dans les années 1930. En 1935, il a accueilli dans son Université de Princeton, pour un stage universitaire, un certain Alan Turing. Les deux hommes se connaissaient donc mais n'ont jamais pour autant travaillé ensemble à la réalisation d'un ordinateur. Pendant la guerre, Von Neumann, a participé au projet «Manhattan», qui a donné lieu à la bombe atomique américaine, en se consacrant particulièrement aux calculs balistiques. Sensibilisé par cette expérience à l'intérêt des calculateurs automatiques, il a ensuite travaillé avec des ingénieurs à la conception d'un tel calculateur. En 1945, il a écrit à leur intention un rapport où il détaillait les principes qui devaient selon lui présider à la réalisation d'une machine universelle (un ordinateur, donc). Ces principes sont depuis connus sous le nom «d'architecture de Von Neumann», et sont ceux encore utilisés de nos jours pour la conception des ordinateurs actuels.

Le schéma général (très simplifié) de l'architecture de Von Neumann est celui de la figure 3.3.

Les deux innovations majeures introduites par Von Neumann par rapports aux calculateurs existant à son époque sont, ainsi, l'intégration :

- d'une «unité de commande» qui donne les ordres et synchronise les opérations ;
- d'une mémoire centrale interne permettant de stocker aussi bien des données que des programmes.

Pour bien comprendre comment fonctionne un ordinateur, il nous faut détailler chacun des composants de cette architecture.

1.3 La mémoire centrale (RAM)

La mémoire vive d'un ordinateur est composée d'un ensemble de «mots mémoire», qui sont des suites de bits de taille fixe. Mais, de même que dans un tableur comme Excel, chaque case d'une feuille de calcul est identifiée par une référence (comme A2, D7...), chaque mot mémoire est identifié par son «adresse». Celle-ci est indispensable pour référencer chaque mot mémoire et ainsi retrouver ce qui a été préalablement stocké dans l'un d'eux. L'adresse est simplement un code, donc une autre suite de bits. Le nombre de bits réservé au codage de l'adresse doit évidemment être suffisant

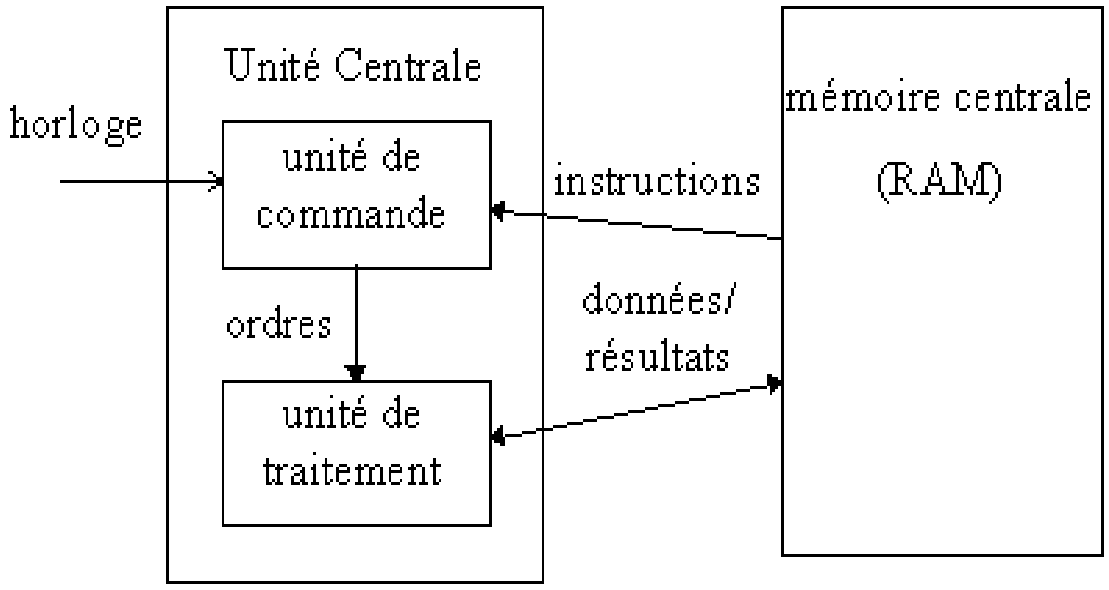


FIG. 3.3 – architecture de Von Neumann

pour permettre d'associer une adresse différente à chaque mot mémoire. La figure 3.4 montre un exemple de mémoire RAM (ultra simple) constituée de 16 mots mémoire de 2 octets chacun (le contenu de la mémoire n'est pas représenté). Chaque mot mémoire a donc une adresse comprise entre 0 et 15.

Une particularité fondamentale de la mémoire centrale, dans l'architecture de Von Neumann, c'est qu'elle sert à stocker indifféremment aussi bien des bits codant des données que des bits codant des traitements, des instructions (on verra par la suite comment coder une instruction dans un mot mémoire). Il n'y a pas de distinction entre les deux, pas de séparation. On retrouve là ce qui caractérisait également les Machines de Turing Universelles, sur le ruban desquelles pouvait figurer aussi bien des données que le code d'une autre machine de Turing (cf. 3.6). Cette capacité à coder avec des 0/1 aussi bien des données que des traitements, c'est le fondement de l'informatique.

1.4 L'unité de commande

Cette unité fondamentale joue un peu le rôle de la tête de lecture des machines de Turing. Elle est elle-même composée de deux «registres». Un registre est simplement une petite unité de mémoire vive (un ensemble de bits, donc, encore !), d'accès rapide. Les deux registres de l'unité de commande sont :

- le compteur ordinal (ou CO) : ce registre sert à stocker en permanence *l'adresse où se trouve en mémoire centrale interne l'instruction en train d'être exécutée* (on dit aussi «l'instruction courante»). Sa taille coïncide donc avec la taille des adresses de la mémoire (4 bits dans notre exemple) ;
- le registre d'instruction (ou RI) : il sert à stocker en permanence *l'instruction en train d'être exécutée* (ou «instruction courante»). Sa taille est donc la même

adresses	mots mémoire	
0 0 0 0	1 octet	1 octet
0 0 0 1		
0 0 1 0		
0 0 1 1		
0 1 0 0		
0 1 0 1		
0 1 1 0		
0 1 1 1		
1 0 0 0		
1 0 0 1		
1 0 1 0		
1 0 1 1		
1 1 0 0		
1 1 0 1		
1 1 1 0		
1 1 1 1		

FIG. 3.4 – structure d'une RAM élémentaire

que celle d'un mot mémoire.

On peut représenter l'unité de commande de notre machine rudimentaire comme dans la figure 3.5.

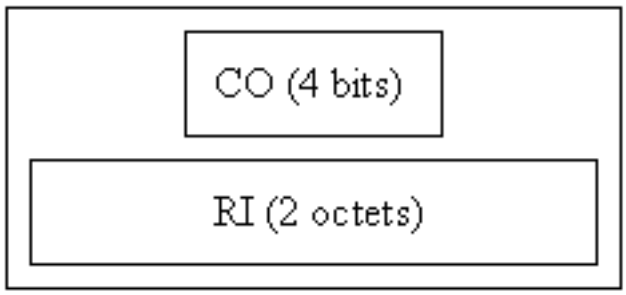
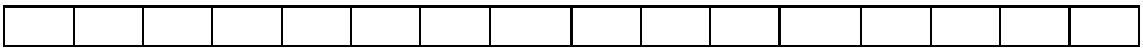


FIG. 3.5 – structure de l'unité de commande

Détaillons maintenant de quoi est composée une instruction élémentaire, telle qu'elle peut être stockée dans un mot mémoire ou dans le registre d'instruction. Une telle instruction est en fait constituée de 4 parties. Dans notre exemple, chaque partie tiendra donc sur $\frac{1}{2}$ octet soit 4 bits.

Nous détaillerons simplement le codage des instructions élémentaires de type arithmétique (addition, soustraction, multiplication, division). Les 4 parties du code de l'instruction ont alors la signification suivante :



- la première, code instr., est le code de l'opération à effectuer. Dans notre exemple, nous nous contenterons des 4 opérations arithmétiques de base et nous nous fixons la convention suivante : 0000 code l'addition, 0001 la multiplication, 0010 la soustraction et 0011 la division. Il reste des codes disponibles pour d'autres opérations possibles.
- la deuxième et la troisième partie, notées ad. donnée 1 et ad. donnée 2 contiennent *l'adresse en mémoire* où se trouvent stockées respectivement la première et la deuxième donnée (dans cet ordre) sur lesquelles l'opération arithmétique doit être effectuée ;
- la quatrième, notée ad. résultat, est *l'adresse en mémoire* où doit être stocké le résultat de l'opération.

Ainsi, par exemple, «0011 1001 0011 0001» est l'instruction d'effectuer une division (code 0011) entre le nombre stocké dans le mot mémoire d'adresse 9 (code 1001) et celui stocké à l'adresse 3 (code 0011) et de stocker le résultat dans le mot mémoire d'adresse 1 (code 0001).

Rappelons-nous en effet qu'un algorithme est une suite d'instructions permettant de résoudre une *classe* de problèmes. Ainsi, l'algorithme donné en 3.1 nécessite toujours de faire des divisions, mais les nombres à diviser dépendent du nombre de

départ n (que l'on fournit comme donnée d'entrée). Programmer un algorithme, c'est donc bien programmer des instructions *génériques*, dont la valeur complète dépendra des instances précises du problème que l'on voudra résoudre (et donc, comme dans notre instruction élémentaire, des données stockées aux adresses citées).

1.5 L'horloge

L'horloge de l'Unité Centrale est un métronome électronique qui lance des «tops» à intervalles de temps réguliers. Ces «tops d'horloge» donnent la cadence à laquelle travaille l'ordinateur et permettent à l'ensemble des composants de l'Unité Centrale de se synchroniser.

Plus les tops sont rapprochés, plus l'ordinateur est rapide. La fréquence de l'horloge se compte en nombre de tops par secondes, dont l'unité de mesure est le Hertz, ou plutôt le Mega-Hertz MH ($1MH = 10^6$ Hertz). Voici l'ordre de grandeur de la vitesse des ordinateurs ces dernières années :

- en 1985 : de 5 à 8 Mega-Hertz
- en 1990 : environ 20 Mega-Hertz
- en 1995 : environ 200 Mega-Hertz
- en 2001 sont sorties les premières puces cadencées à 1 Giga-Hertz (soit 10^9 Hetz)

1.6 L'unité de traitement

L'unité de traitement est le composant qui exécute les calculs. Il est lui-même composé :

- de trois registres, servant respectivement à stocker les données (que nous notons donnée 1 et donnée 2) d'une opération arithmétique et son résultat : leur taille est celle d'un mot mémoire (2 octets dans notre exemple) ;
- de l'Unité Arithmétique et Logique (UAL) capable, quand on lui fournit le code d'une opération arithmétique à exécuter, de prendre les contenus des deux premiers registres (ceux contenant les données 1 et 2) et de remplir le troisième registre avec le résultat de cette opération.

L'unité de traitement se représente habituellement comme dans la figure 3.6 (on rappelle dans l'UAL le code des opérations élémentaires qui y sont câblées) :

L'UAL est ainsi la «machine à calculer» de l'ordinateur. Elle est simplement constituée de circuits électroniques câblés une fois pour toute pour transformer des 1 en 0 ou des 0 en 1 (c'est-à-dire en fait pour actionner des interrupteurs faisant passer ou non du courant) de façon à ce que les bits du registre résultat correspondent bien au codage du résultat du calcul qui lui est demandé. Elle ne sait faire que des opérations élémentaires (dans notre exemple : simplement les 4 opérations arithmétiques de base).

1.7 Les bus

Les flèches reliant les composants entre eux sont en fait des *ensembles de fils* permettant de transporter *plusieurs bits en parallèle*. On les appelle pour cela des

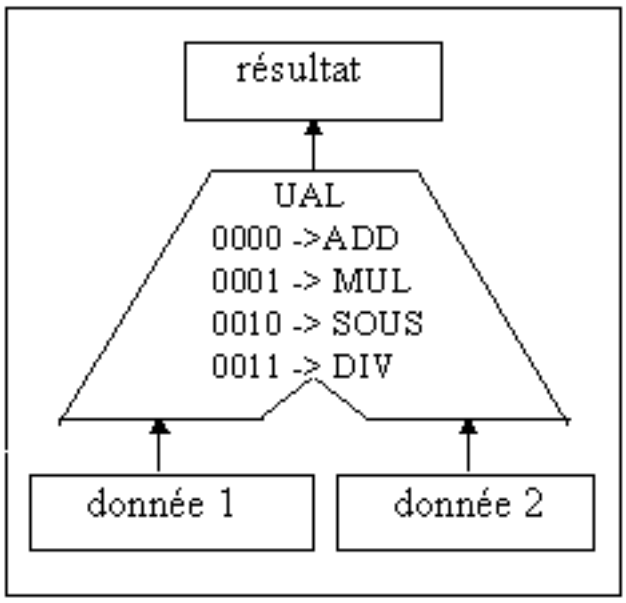


FIG. 3.6 – structure de l'unité de traitement

bus.

Dans le schéma de la figure 3.3, figurent ainsi 3 bus dont les noms désignent le type de données qu'ils transportent. Le bus «ordres» sert à transmettre les demandes d'exécution d'opérations de l'unité de commande vers l'unité de traitement. Le bus «instructions» fait transiter les instructions élémentaires des mots mémoire vers le registre d'instruction de l'unité de commande, et le bus «données/résultats» fait circuler (dans les deux sens) le contenu des mots mémoires entre la mémoire et les différents registres de l'unité de traitement.

Ces différents bus peuvent contenir un nombre de fils différent. Le nombre de fils du bus de données/résultats détermine la capacité du microprocesseur. Un «microprocesseur 32 bits» (ordre de grandeur des puces actuelles) contient donc un bus données/résultats composé de 32 fils.

1.8 Le cycle d'exécution d'une instruction

Supposons maintenant que la mémoire centrale de notre ordinateur contienne un programme et des données, et que l'on souhaite exécuter ce programme sur ces données. Lancer cette exécution revient à mettre dans le compteur ordinal (CO) l'adresse où se trouve stockée la première instruction du programme. A partir de là, le programme est exécuté étape par étape, instruction par instruction. L'exécution d'une *instruction élémentaire*, codée suivant la convention expliquée en 1.4 se fait suivant un *cycle* comprenant 3 phases :

– phase 1 : L'instruction courante, dont l'adresse est stockée dans le CO, est recopiée dans le registre d'instruction (RI) en transitant par le bus «instructions» ;

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

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

Illustrons ce fonctionnement à l'aide d'un exemple complet sur notre ordinateur miniature (seuls les bits «utiles» de la mémoire et des registres sont donnés). La figure 3.7 montre la situation de départ, les trois suivantes montrent l'état de l'ordinateur après l'exécution de chaque phase d'un cycle.

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

La figure 3.8 montre l'état de l'ordinateur à l'issue de la première phase du premier cycle. L'instruction courante a transité à travers le bus «instructions» et se retrouve désormais dans le registre d'instruction. Notons que, si le bus «instructions» est composé de 4 fils, comme sur notre schéma, alors qu'une instruction dans un mot mémoire est stockée sur 2 octets, alors ce transfert a dû se faire, au mieux, en 4 passages successifs (correspondant chacun à un top d'horloge).

La figure 3.9 montre l'état de l'ordinateur à l'issue de la deuxième phase du premier cycle. Le bus «ordres» a fait tout d'abord transiter les 4 premiers bits de l'instruction courante à l'UAL, qui a reconnu que c'était le code d'une addition (l'opération active dans l'UAL est donc l'addition). Les deux suites de 4 bits suivantes correspondent aux adresses en mémoire où l'UAL doit aller chercher les données sur lesquelles effectuer cette opération. Ces données viennent remplir les registres «donnée 1» et «donnée 2» de l'UAL en passant pas le bus «données/résultats».

La figure 3.10 montre l'état de l'ordinateur à l'issue de la troisième et dernière phase du premier cycle. L'UAL a réalisé l'opération qui lui était demandée et a rempli avec le résultat son registre «résultat». La dernière partie de l'instruction courante indique l'adresse du mot mémoire où ce résultat doit être stocké. C'est ce qui est fait en utilisant de nouveau le bus «données/résultats». Par ailleurs, pour préparer le cycle suivant, le compteur ordinal est augmenté de 1.

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

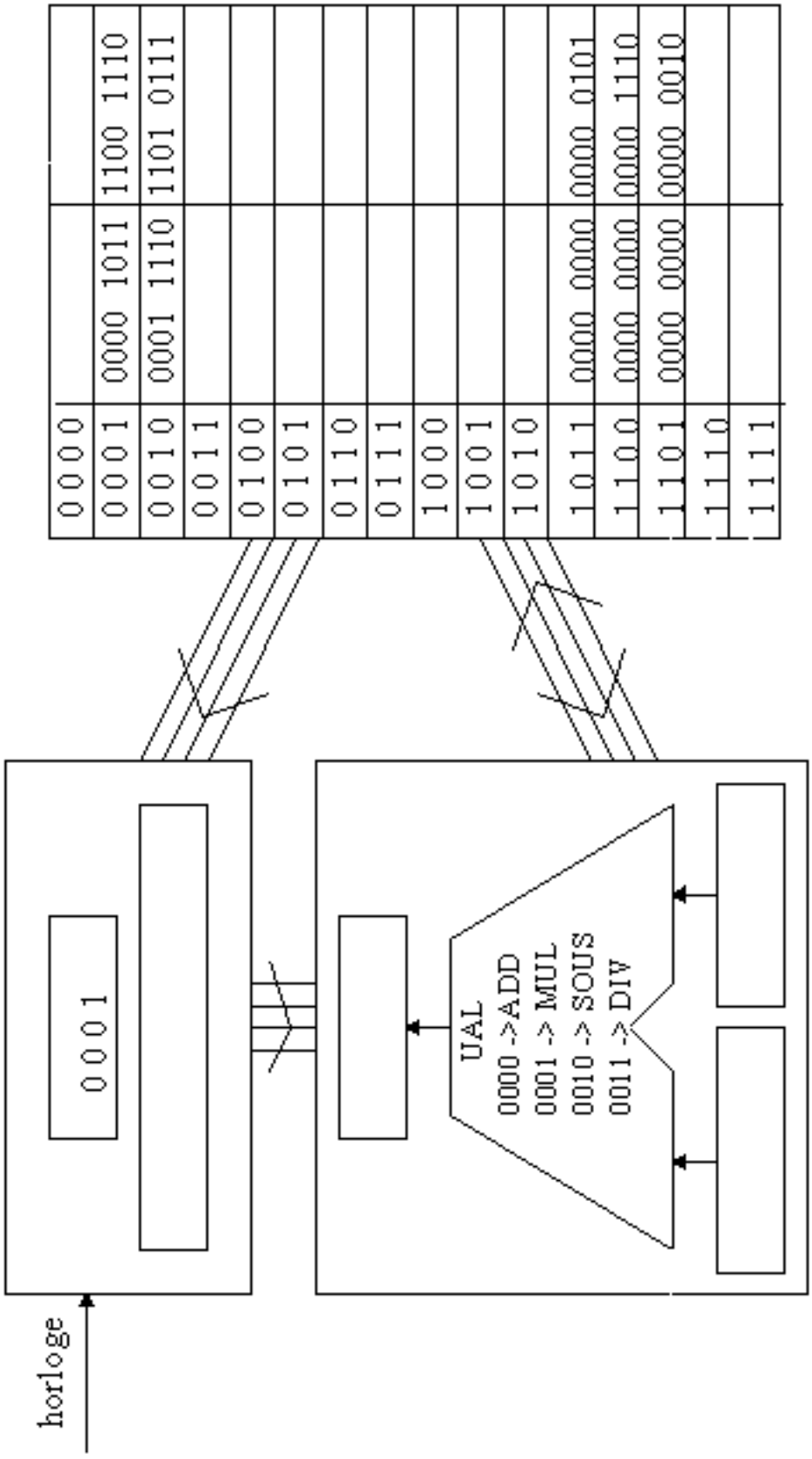


FIG. 3.7 – situation de départ

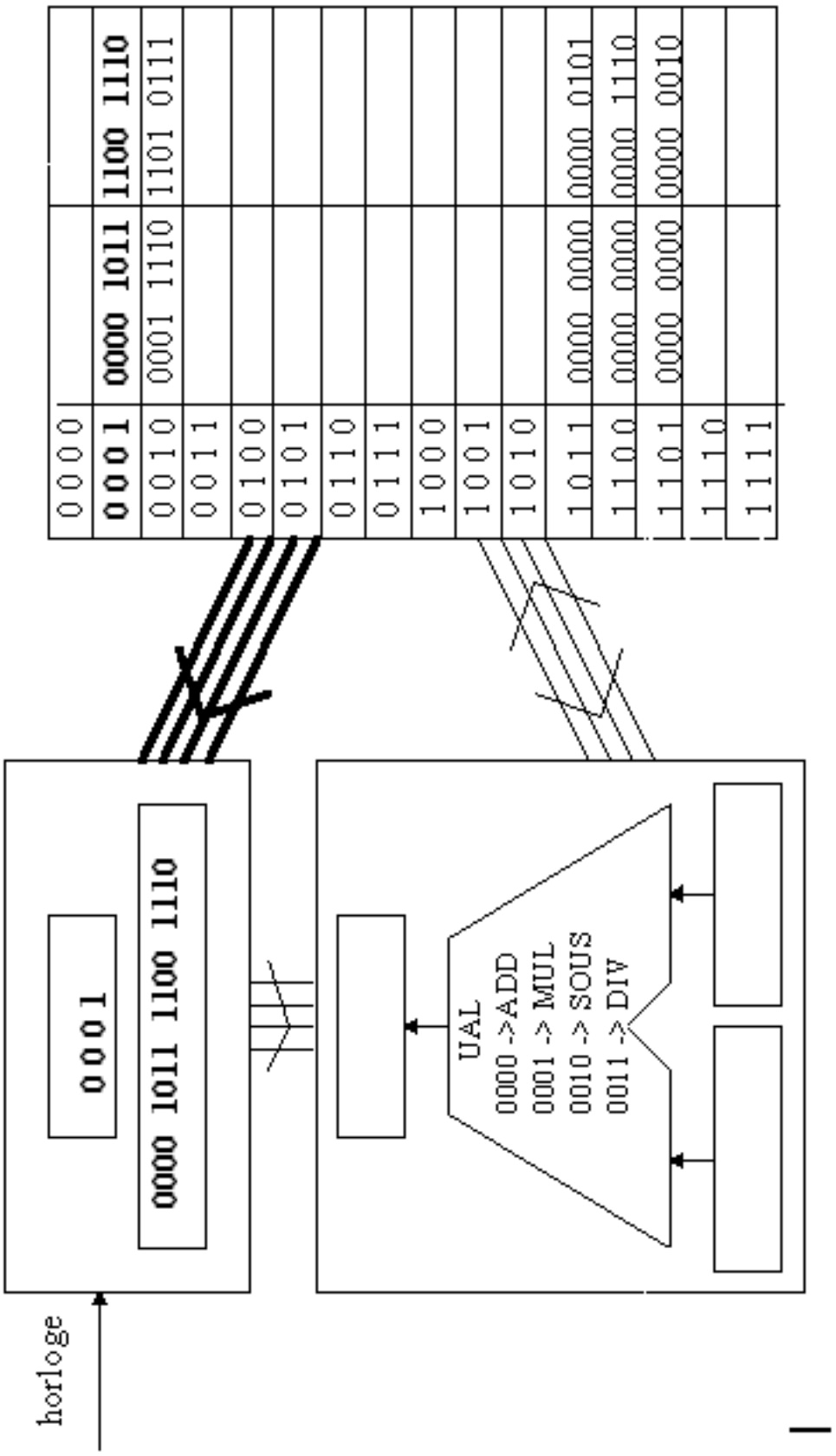


FIG. 3.8 – premier cycle, phase 1

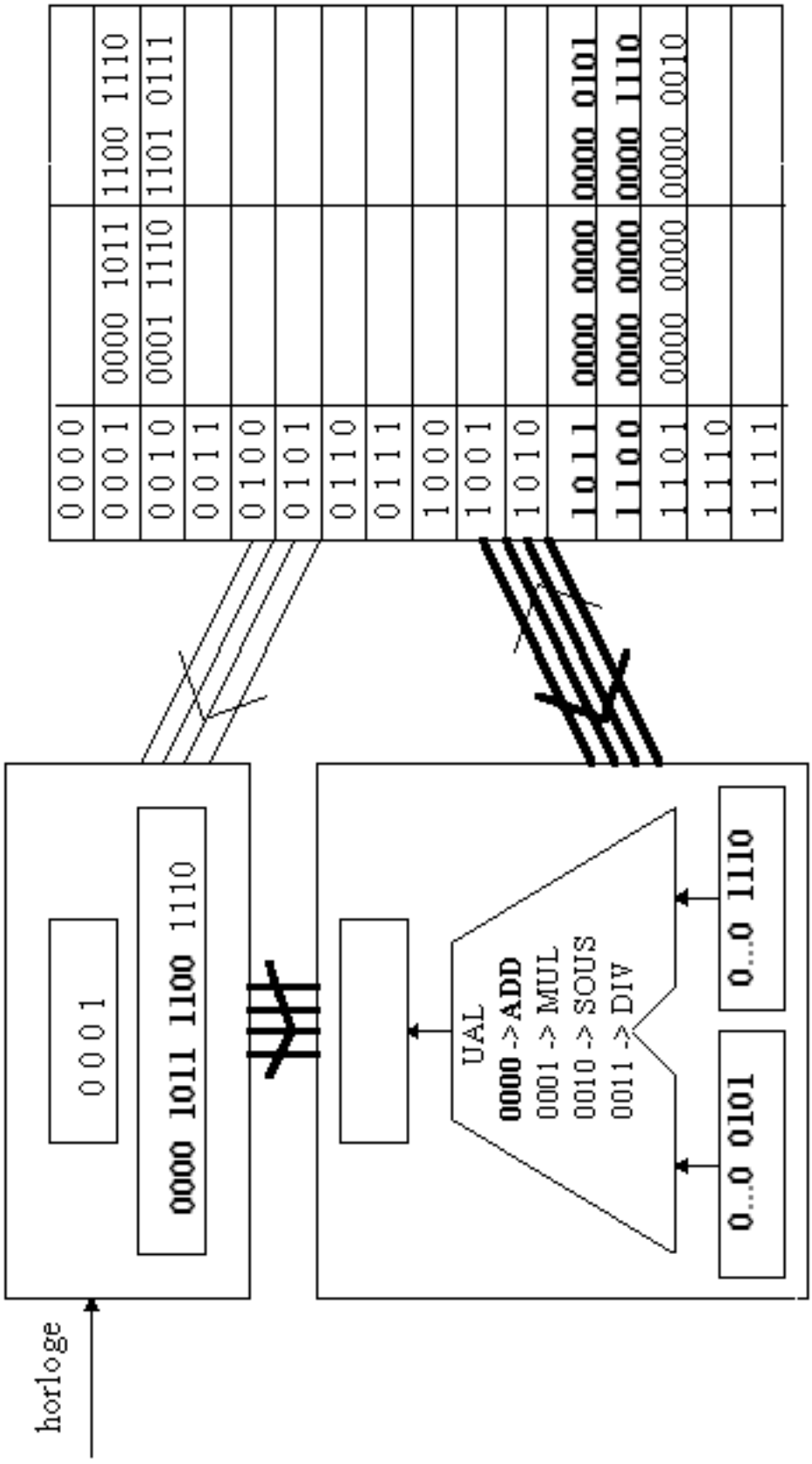


FIG. 3.9 – premier cycle, phase 2

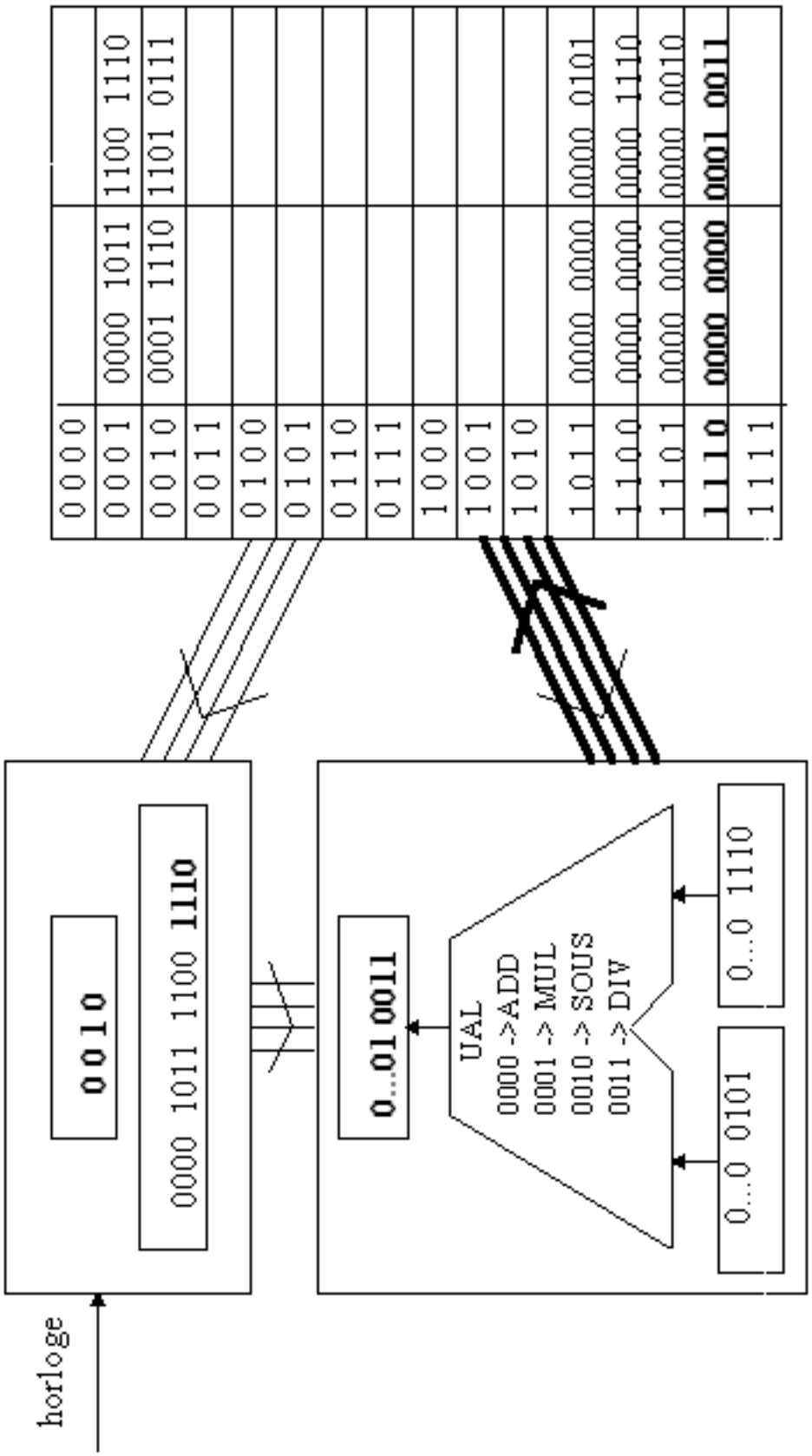


FIG. 3.10 – premier cycle, phase 3

La figure 3.11 montre la situation de l'ordinateur à la fin de l'exécution de ce deuxième cycle (sans détailler les 3 phases, cette fois). Un nouveau cycle peut commencer, mais nous supposons qu'il n'y a plus d'information pertinente à l'adresse référencée par le CO (ou il y en a une signifiant «FIN») : l'exécution du programme est donc terminée.

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

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

Le programme, l'algorithme réalisait donc l'opération composée suivante :
nombre 1 + nombre 2 = nombre 3
nombre 3 * nombre 4 = résultat
ou encore : résultat = (nombre 1 + nombre 2) * nombre 4

Les nombres 1, 2 et 4 sont les *données* sur lesquelles le programme a été exécuté. Nombre 3 est un *résultat intermédiaire* et le *résultat final* est noté «résultat». En fait, l'algorithme composé des 2 instructions exécutées par l'ordinateur réalise la fonction suivante :

$$\begin{array}{rcl} f : & \mathbb{N}^3 \longrightarrow \mathbb{N} \\ & (x, y, z) \longmapsto (x + y) * z \end{array}$$

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

Sur la figure 3.7, les deux mots mémoire présents aux adresses 0001 et 0010 respectivement contenaient des instructions, tandis que les mots mémoire dont les adresses étaient 1011, 1100 et 1101 contenaient des données. L'exemple a aussi montré comment l'enchaînement de plusieurs opérations élémentaires pouvait permettre de réaliser une opération plus complexe.

1.9 L'architecture de Von Neumann et les autres

On peut montrer qu'une machine construite suivant l'architecture de Von Neumann a bien la capacité de calcul d'une Machine de Turing Universelle : c'est donc bien un *ordinateur*. La version qui en a été présentée ici est néanmoins extrêmement simplifiée. En plus des composants cités, elle inclut classiquement un certain nombre de registres, permettant de stocker des résultats intermédiaires, et bien d'autres dispositifs. De même des codes d'instructions spéciaux permettent par exemple de «sauter» de l'instruction courante à une autre instruction non contiguë dans la mémoire (c'est-à-dire stockée ailleurs que dans le mot mémoire suivant immédiatement celui contenant l'instruction courante), ou d'exprimer un «test».

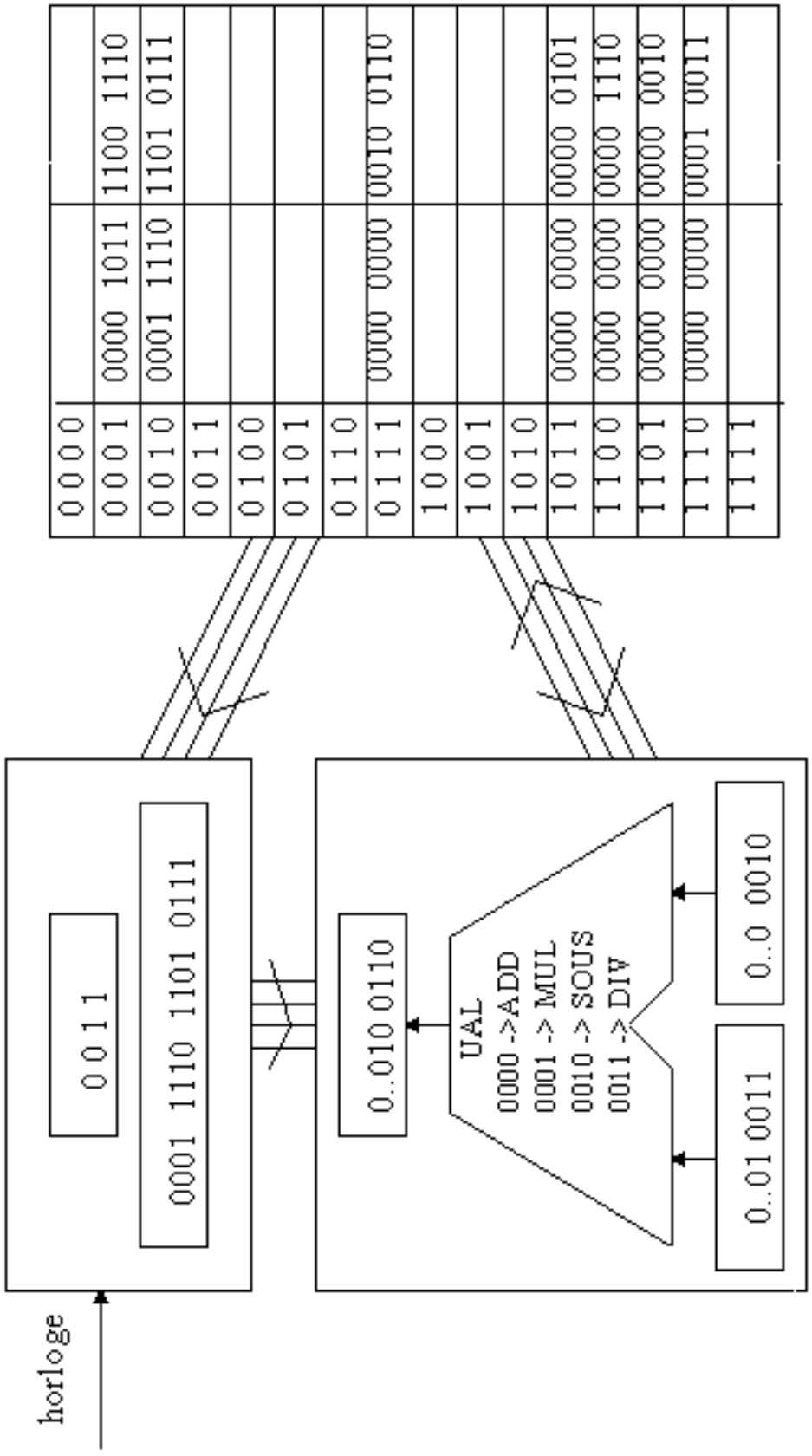


Fig. 3.11 – situation à l'issue du deuxième cycle

Ce schéma d'organisation interne est celui qui préside depuis 1945 à la réalisation de la grande majorité des ordinateurs. Parmi ceux-ci, les plus efficaces dans les années 90 étaient ceux construits suivant l'architecture RISC (pour " Reduce Intruction-Set Computer"), dont la particularité est de n'intégrer qu'un nombre réduit d'instructions élémentaires pré-cablées (comme les opérations arithmétiques dans notre exemple) et un grand nombre de registres. La caractéristique fondamentale de l'architecture de Von Neumann, dont la version RISC n'est qu'une variante, est qu'elle exécute les instructions d'un programme les *unes après les autres*, dans un ordre fixé à l'avance par l'écriture du programme. On parle pour cela d'ordinateurs *séquentiels*. C'est aussi ainsi que fonctionnent les machines de Turing.

C'est ce à quoi s'oppose une autre «race» d'ordinateurs appelés «parallèles». Ces derniers sont construits suivant des principes un peu différents : ils incluent plusieurs processeurs fonctionnant indépendamment les uns des autres et échangeant entre eux des données. De nombreuses variantes sont possibles suivant le nombre de processeurs présents, le type de mémoire(s) associée(s) à chacun d'eux et leur mode d'interconnexion.

Ces ordinateurs parallèles, quand ils sont utilisés au maximum de leur capacité, sont évidemment beaucoup plus rapides que les ordinateurs séquentiels, puisque *plusieurs opérations s'y effectuent en même temps*. On les appelle aussi pour cela *supercalculateurs*. Les machines les plus rapides au monde appartiennent à cette catégorie. Ils ne sont néanmoins pas adaptés à tous les types d'applications et sont beaucoup plus difficiles à programmer que les ordinateurs séquentiels, puisque le programmeur doit prévoir à l'avance ce que chacun des processeurs sera chargé de calculer.

Ils sont donc pour l'instant réservés à la résolution de problèmes très particuliers, pouvant se décomposer en sous-problèmes relativement indépendants les uns des autres et nécessitant chacun de gros calculs, comme la génération ou le traitement d'images ou la prévision météorologique.

Avec le développement des réseaux, on peut aussi maintenant *simuler* le comportement d'une machine parallèle en faisant interagir entre eux plusieurs ordinateurs usuels, qui se répartissent les calculs à effectuer. Certains programmes plus ou moins scientifiques (comme SETI : Search for Extraterrestrial Intelligence, ou des projets de bioinformatique) ont mis à contribution les ordinateurs personnels de particuliers bénévoles pour répartir leurs calculs, via Internet. L'utilisation de réseaux d'ordinateurs demande aussi, bien sûr, une programmation spécifique.

1.10 Les unités d'échange

Revenons pour finir à nos ordinateurs. Pour que le «coeur» de l'ordinateur puisse échanger des informations avec les périphériques, il faut qu'une connexion entre eux soit assurée. C'est ce que réalisent des «unités d'échanges». Chacune est spécialisée dans l'interface entre l'Unité Centrale, la mémoire centrale et un certain périphérique. L'écran, lui, a un statut un peu à part : il se contente de visualiser une partie de la mémoire centrale appelée «mémoire d'affichage», où sont stockés les composants graphiques des programmes en cours d'exécution. L'organisation matérielle générale d'un ordinateur est donc celle du schéma de la figure 3.12.

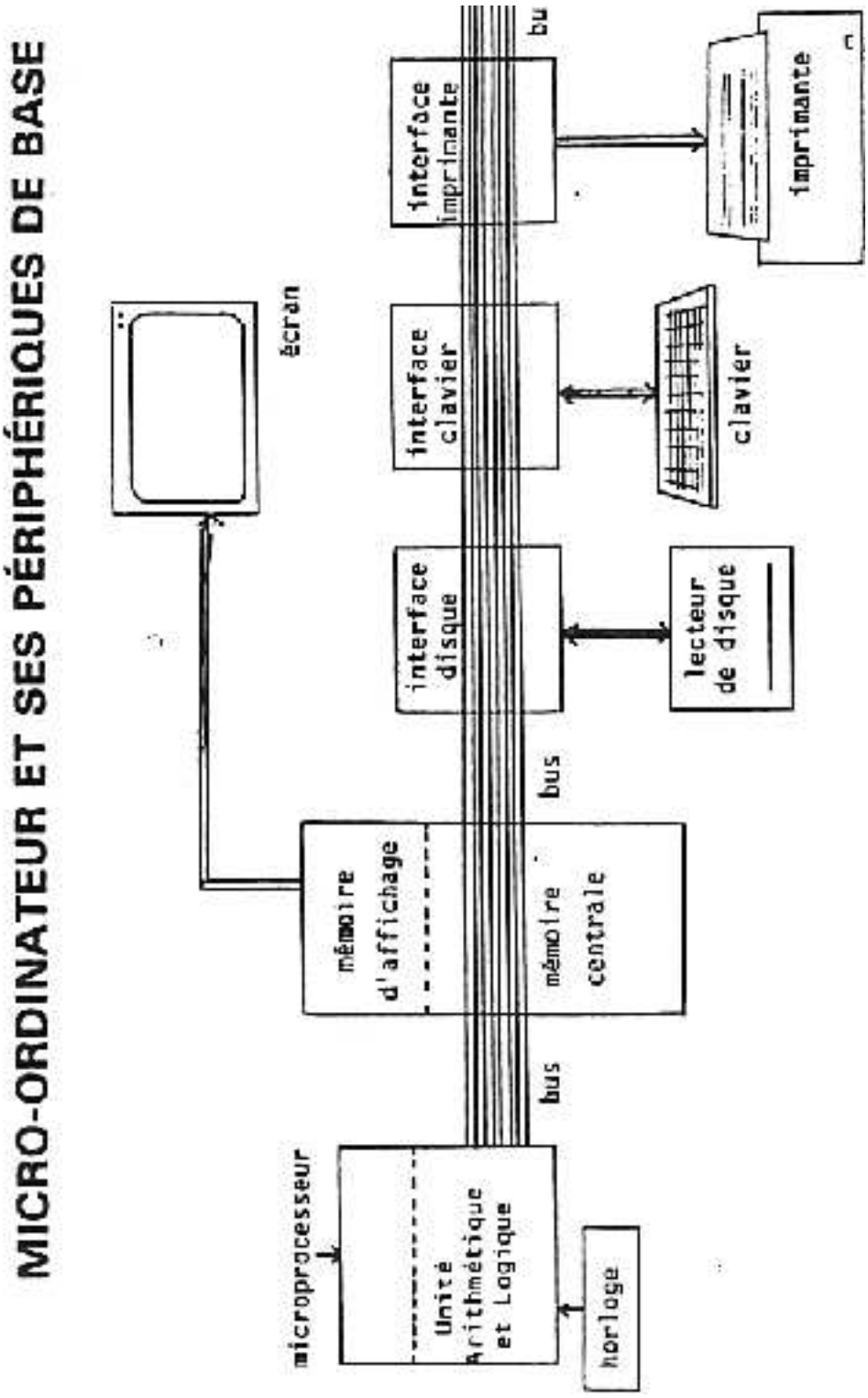


FIG. 3.12 – relations entre l'UC et les unités d'échange

Dans les ordinateurs actuels les plus performants, ces unités d'échange peuvent elles-mêmes inclure une Unité Centrale spécialisée uniquement dans la gestion d'un certain périphérique ou d'une certaine fonction, constituant ce que l'on appelle par exemple une «carte graphique», une «carte son» ou une «carte réseau».

Signalons pour terminer la célèbre *Loi de Moore*, vérifiée depuis le début des années 60 (Moore est un des fondateurs de la société Intel) : tous les 18 mois, la capacité de mémoire des ordinateurs et leur vitesse de calcul sont doublées, pendant que leur prix à qualité équivalente est divisé par deux.

2 Le logiciel

La partie «logicielle» d'un ordinateur est constituée de l'ensemble des codes stockés dans sa mémoire, et plus particulièrement de ses programmes. Mais, depuis les débuts de l'informatique, les codes se sont superposées les uns sur les autres, constituant ce qu'on appelle des *couches logicielles*. Nous allons essayer ici de démêler ces couches.

2.1 langages d'assemblage et langages évolués

Ecrire un programme, c'est donc écrire une suite d'instructions élémentaires s'enchaînant les unes après les autres pour réaliser un traitement sur des données. Dans le disque dur et la mémoire centrale, ces programmes sont codés sous forme de bits à la façon du mini-programme dont l'exécution est détaillée en 1.8. De tels programmes sont aussi appelés des «programmes exécutables», puisqu'ils sont directement prêts à être exécutés.

Au début de l'informatique, dans les années 50, programmer revenait ainsi à écrire de telles suites d'instructions élémentaires. Les langages de programmation les plus proches de ce codage de «bas niveau» sont appelés «langages d'assemblage» ou «Assembleurs». Programmer dans un tel langage nécessite de connaître l'architecture matérielle de l'ordinateur sur lequel il s'exécutera ; il y a ainsi presque autant de langages d'assemblage différents que de microprocesseurs. Par exemple, imaginons un ordinateur similaire à notre ordinateur rudimentaire, mais disposant en plus (pour simplifier la suite) d'une dizaine de registres numérotés de 1 à 10, ayant chacun la taille d'un mot mémoire. Le mini-programme exécuté en 1.8 pourrait alors s'écrire en Assembleur de la façon suivante :

ADD 1 2 3

MUL 3 4 5

Dans lequel ADD et MUL signifient respectivement «addition» et «multiplication» et où les numéros 1 à 5 désignent des registres (il est plus facile d'utiliser les registres pour stocker des données intermédiaires que de désigner directement des adresses en mémoire). Chaque instruction élémentaire est constituée exactement de la même façon que les instructions binaires de notre mini-programme : la première donne ainsi l'ordre d'additionner le contenu du registre 1 avec celui du registre 2, et de stocker le résultat dans le registre 3.

Les programmes écrits en Assembleur se traduisent immédiatement en programmes exécutables. Mais programmer en Assembleur n'est ni très simple ni très agréable.

Aussi, de nouveaux langages de programmation ont peu à peu été définis pour faciliter la tâche des programmeurs. En inventer un nouveau, cela signifie :

- définir un langage permettant d'exprimer n'importe quel algorithme ;
- définir une correspondance, une méthode de traduction (exprimable par un algorithme !) entre ce nouveau langage et un langage d'assemblage.

La difficulté vient plutôt de la seconde condition que de la première. En effet, toutes les langues humaines sont en principe capables d'exprimer toute méthode de calcul, tout algorithme, de façon plus ou moins informelle. Mais les ordinateurs ne comprennent rien aux langues humaines. De plus, les langues humaines sont beaucoup trop imprécises et ambiguës pour être traduisibles directement en Assembleur. Le problème est donc de définir une langue qui sera traduisible en Assembleur, de telle sorte que cette traduction elle-même puisse s'exprimer par un algorithme.

Ce rôle de traduction est joué par ce qu'on appelle les *compilateurs*. Un compilateur est un logiciel capable de transformer un programme écrit dans un langage de programmation donné L1 en un programme réalisant le même traitement mais écrit dans un autre langage L2 (en général un Assembleur), comme le montre le schéma de la figure 3.13.

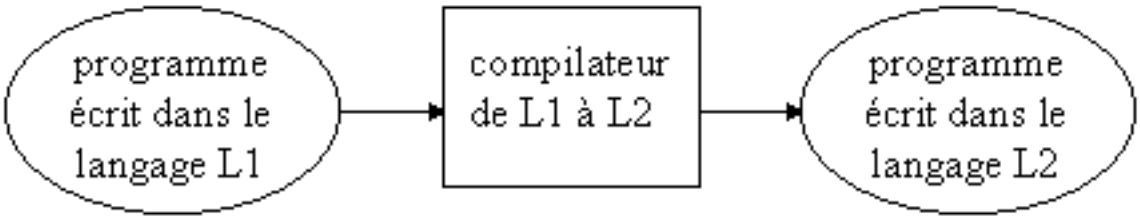


FIG. 3.13 – rôle d'un compilateur

Les compilateurs étant des programmes, ils sont eux-mêmes écrits dans un certain langage de programmation (et s'ils ne sont pas écrits en Assembleur, ils doivent eux-mêmes être compilés...)..

L'utilisation de compilateurs a permis la définition de langages de programmation de «haut niveau» ou «évolués». Dans de tels langages, on peut s'abstraire de la connaissance matérielle de l'ordinateur sur lequel s'exécutera le programme pour se concentrer sur sa seule logique.

Pascal est un langage de ce type. En Pascal, notre mini-programme peut s'écrire (à quelques instructions près) :

```
BEGIN
nombre3 := nombre1 + nombre2 ;
nombre5 := nombre3 * nombre4 ;
END.
```

ou plus simplement :

```
BEGIN
nombre5 := (nombre1 + nombre2) * nombre4 ;
END.
```

Dans ce programme, « :=» et « ;» ainsi que les mots clés BEGIN et END sont des symboles propres au langage Pascal. Les mots «nombre1» à «nombre5», eux, ne désignent plus une adresse en mémoire ni un registre mais une *variable*, c'est-à-dire un espace mémoire quelconque, abstrait, auquel est attribué un nom, sans que le programmeur ait à préciser où il se trouve physiquement.

Tout programme écrit en Pascal commence par le mot BEGIN et se termine par le mot END. La première instruction du premier programme (après le BEGIN) signifie donc que la variable nommée «nombre3» recevra comme contenu le résultat de l'addition des contenus respectifs des variables nommées respectivement «nombre1» et «nombre2». Ce sera au compilateur Pascal de transformer ce programme (appelé aussi «programme source» parce qu'il est écrit dans un langage évolué) en la suite d'instructions en Assembleur donnée plus haut. En Python, on écrira cette même instruction :

nombre5 = (nombre1 + nombre2) * nombre4

Mais attention : le symbole «=» apparaissant dans cette formule n'est pas une égalité au sens mathématique du terme. C'est une instruction, un ordre signifiant que ce qui est écrit à droite est la nouvelle valeur qui doit être stockée dans la variable écrite à gauche. Ainsi, en informatique, une instruction comme :

n=n+1

est tout à fait recevable, alors qu'elle n'aurait pas grand sens en mathématique ; elle signifie : la nouvelle valeur de n est son ancienne valeur + 1.

2.2 La démarche de conception d'un programme

Concevoir un programme pour résoudre un problème donné, c'est donc suivre la démarche de la figure 3.14.

Dans ce schéma, les phases d'Analyse et de Traduction sont des opérations intellectuelles humaines, alors que les phases de Compilation et d'Exécution sont réalisées automatiquement par la machine. Le programme source et le programme exécutable sont deux versions numériques (stockées sous forme de bits dans la mémoire) du même algorithme. Le programme source est écrit en langage évolué, alors que le programme exécutable est le résultat de sa compilation. En général, quand on achète un logiciel dans le commerce, c'est uniquement le programme exécutable (illisible, même pour informaticien) qui est fourni. Quand le code source est aussi fourni, on parle de «logiciel libre», puisque tout informaticien peut alors lire le programme (à condition qu'il connaisse le langage dans lequel il est écrit) et le modifier s'il le souhaite.

Une erreur syntaxique est une faute d'écriture (l'équivalent d'une faute d'orthographe ou de grammaire en français) ; elle est signalée par le compilateur quand celui-ci ne sait pas comment interpréter certaines instructions du programme source afin de les traduire dans le langage cible (Assembleur). Ces erreurs, quasiment inévitables même par les programmeurs expérimentés, proviennent d'un non respect des préconisations (draconiennes !) du langage de haut niveau utilisé pour le programme source. Par exemple, oublier de commencer un programme par BEGIN, oublier de le terminer par END ou ne pas séparer deux instructions élémentaires en Pascal par le symbole « ;» constituent des fautes de syntaxe.

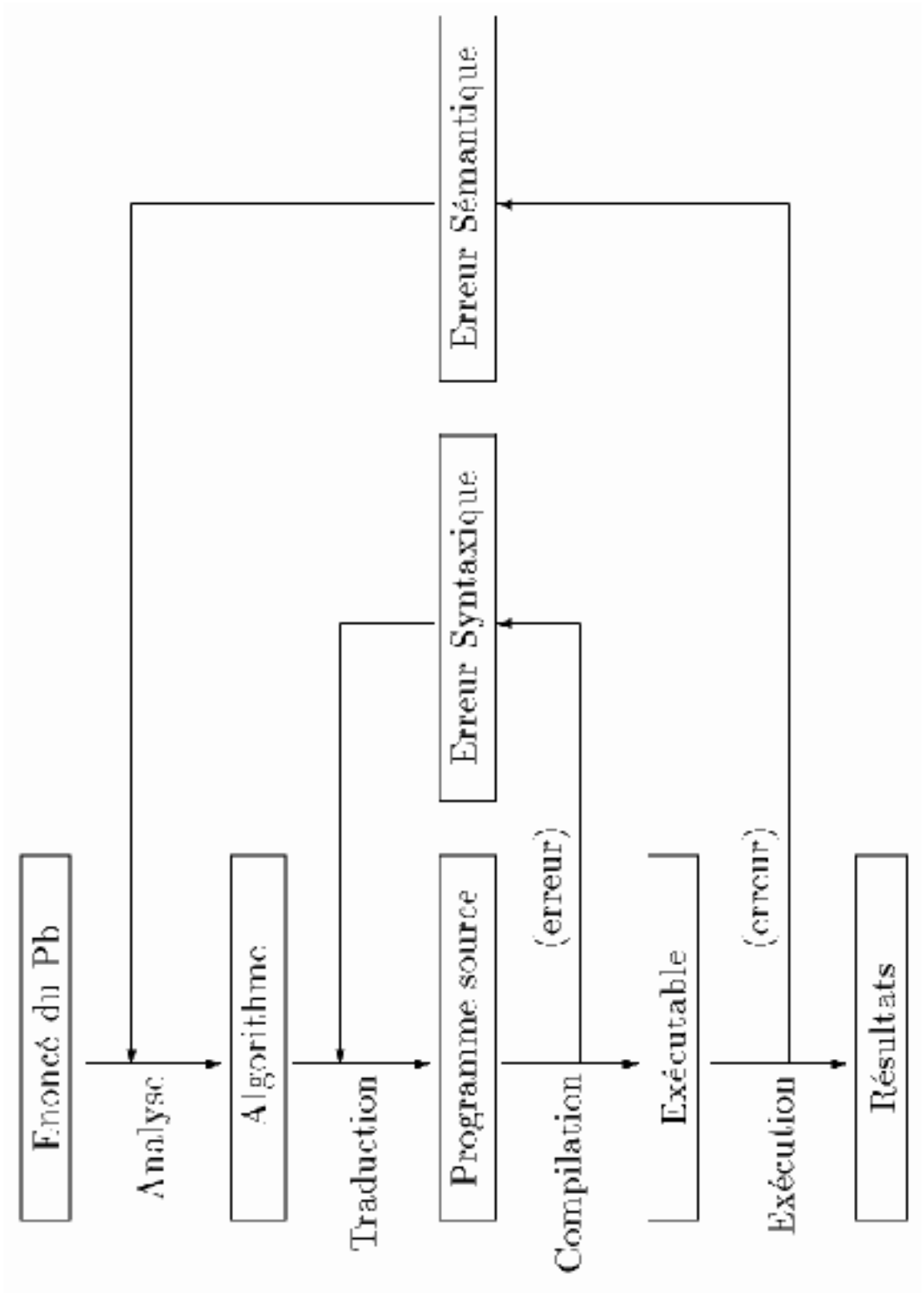


FIG. 3.14 – démarche de conception d'un programme

Une erreur sémantique, beaucoup plus grave, provient d'une mauvaise conception de la méthode suivie pour résoudre le problème. C'est alors l'algorithme lui-même qui doit être modifié, ce qui oblige à remettre en cause les partis pris de la phase d'Analyse.

2.3 Le système d'exploitation

Parmi les logiciels les plus usuels, il en est un qui est devenu indispensable à tous les ordinateurs actuels : c'est le système d'exploitation ou système opératoire (traduction de «Operating System»). Le système d'exploitation d'un ordinateur est en quelque sorte son gestionnaire central, son chef d'orchestre. En tant que programme, il peut être écrit en Assembleur (sinon, il est écrit dans un autre langage et compilé) et est en permanence actif quand l'ordinateur est sous tension. En fait, quand on allume un ordinateur, on provoque automatiquement la recopie du système d'exploitation du disque dur vers la mémoire centrale ; ce programme étant volumineux, c'est ce qui explique que le «démarrage» d'une machine soit si long.

Ses rôles principaux sont les suivants :

- fournir une «interface» entre l'ordinateur et l'utilisateur pour permettre à ce dernier de donner des ordres à la machine (par exemple : lire ou écrire des informations dans la mémoire, lancer une impression...) ou pour lui signaler les erreurs d'exécution ; cette interface prend soit la forme d'un langage de commande (comme «MS-DOS», Shell) soit la forme d'objets graphiques à manipuler (fenêtres, menus...) ;
- gérer les «ressources» de l'ordinateur, à savoir ses mémoires, son microprocesseur et ses périphériques : les systèmes d'exploitation actuels, en effet, sont «multitâches» ; cela signifie qu'ils permettent à plusieurs programmes de s'exécuter en même temps, et se chargent de répartir l'occupation des ressources utilisées par chacun d'eux (par exemple si deux programmes P1 et P2 sont lancés en même temps, le système d'exploitation permettra à un petit bout de P1 de s'exécuter, puis laissera la place à un petit bout de P2, puis de nouveau à un petit bout de P1, etc., de sorte que l'utilisateur aura l'impression que P1 et P2 sont exécutés en parallèle, alors que le processeur est toujours unique et séquentiel) ;
- être indépendant du matériel, masquer les particularités de la machine en substituant aux ressources physiques des abstractions (par exemple, la notion de fichier, sur laquelle nous reviendrons, est une notion abstraite, indépendante de la nature du support sur lequel les données de ce fichier sont réellement stockées) ;
- contrôler les usagers en leur donnant des droits différents selon leur statut (associés par exemple à différents mots de passe).

En résumé, le système d'exploitation est la couche logicielle de base qui s'intercale toujours entre l'utilisateur et le matériel, comme l'illustre le schéma suivant de la figure 3.15.

Les systèmes d'exploitation les plus couramment installés sur les ordinateurs actuels sont :

- MS-DOS (officiellement abréviation de «MicroSoft Disk Operating System»

FIG. 3.15 – rôle du système d'exploitation

mais il semblerait que le D provienne en fait de «Dirty» : sale, malpropre, crasseux...) : système en voie de disparition, exclusivement monotâche, défini par un langage de commande. Il constituait la base des systèmes «Windows» de Microsoft jusqu'à Windows 3.1 inclus (c'est-à-dire que dans ces systèmes, les manipulations d'objets graphiques étaient en fait traduits en commandes MS-DOS) ;

- Windows 95, 98, XP, NT : systèmes d'exploitation multitâches de Microsoft ayant pris la place de MS-DOS (la version NT est plus particulièrement destinée à la gestion des ordinateurs en réseaux), la prochaine version annoncée pour 07 s'appellera Vista ;
- la série des MacOS (Mac Intosh Operating System), dont la dixième version, notée OS X, est sortie il y a quelques années, équipe les MacIntosh de la firme Apple : ces systèmes ont introduit les premiers, dès 1984, les outils d'interface graphiques (menus, fenêtres...). Les systèmes actuels de Mac sont en fait des variantes du système Linux ;
- Linux : version pour PC d'un célèbre système d'exploitation nommé Unix, multitâche et multiutilisateur, destiné initialement aux gros ordinateurs scientifiques, dits aussi «stations de travail». Il est constitué d'un langage de commande (appelé Shell) et sa particularité est d'avoir été écrit par des programmeurs bénévoles, qui le diffusent de manière libre (le code source est disponible) et gratuite. A l'heure actuelle, il est associé à des environnements graphiques comme «Gnome» ou «KDE». On appelle «distribution Linux» (dont les plus diffusées à l'heure actuelle s'appellent «Fedora», «Mandriva»,«Debian» et «Ubuntu») l'ensemble constitué par une version de Linux, certains environnements graphiques et certains autres programmes nécessaires à son installation sur un PC.

2.4 La hiérarchie des répertoires (ou dossiers) et des fichiers

Nous avons vu dans la première partie de ce document que les données stockées par les ordinateurs peuvent provenir de sources très variées : textes, formules mathématiques, images, etc., chacune correspondant à un mode de codage spécifique. Il ne saurait pourtant être question de stocker toutes ces données «en vrac» dans la (les) mémoire(s) des ordinateurs. De même que pour classer rationnellement des documents papiers, on les range dans des pochettes et des classeurs, le système d'exploitation gère la mémoire disponible à l'aide de *fichiers* et de *réperoires* (on parle aussi de *dossiers*). La figure 3.16 est un exemple de copie d'écran d'un environnement Windows où sont présents de tels composants (les traits symbolisant les liens entre un dossier et la fenêtre montrant son contenu ont été ajoutés).

Un fichier sert à stocker des données de même nature (par exemple : caractères provenant d'un texte ou fichier son contenant la version numérisée d'une chanson). C'est une unité *logique* : un fichier ne correspond pas à un espace mémoire réservé une fois pour toute, il n'a pas de taille fixe prédéfinie et les données qu'il contient peuvent éventuellement ne pas être contiguës en mémoire. Mais, pour l'utilisateur, la façon dont le système d'exploitation gère les fichiers est invisible (les informaticiens disent «transparente»). Tout est fait pour que l'utilisateur ait l'impression que les fichiers qu'il visualise se présentent comme des suites de données cohérentes.

Dans les systèmes Windows, les fichiers reçoivent un nom qui se termine toujours par une «extension» de 3 lettres précédée d'un point. Ces 3 lettres permettent de repérer avec quel logiciel le fichier a été rempli : elles indiquent donc indirectement le mode de codage des données stockées dans ce fichier. Ainsi, un fichier «.txt» contient du texte, donc est codé par une succession de bits correspondant à des codes ASCII, un fichier «.exe» est un programme exécutable, codé sous formes d'instructions élémentaires comme en 1.4, un fichier «.bmp» code une image bitmap...

Les dossiers (ou répertoires), eux, sont plutôt à considérer comme des boîtes ou des classeurs : ils ne contiennent pas directement des données, mais servent d'unités de rangement, pour recevoir soit des fichiers, soit d'autres dossiers (ils peuvent aussi rester vides).

Les fichiers et les dossiers sont structurés dans la mémoire de l'ordinateur de façon *arborescente*. Ainsi, par exemple, l'ensemble de fichiers et de dossiers qui apparaissent dans l'environnement Windows de la figure 3.16 correspond à l'organisation arborescente de la figure 3.17 (attention : c'est un arbre avec le tronc ou la racine en haut et les feuilles en bas...).

Dans un tel arbre, les fichiers ne peuvent figurer qu'au niveau des feuilles (puisqu'eux-mêmes ne peuvent pas contenir d'autre fichier ou dossier). Les dossiers, eux, constituent les noeuds intermédiaires et n'apparaissent au niveau des feuilles que quand ils sont vides.

C'est le système d'exploitation qui gère toute cette organisation : il permet par exemple d'ajouter, de déplacer, de supprimer, de recopier... tout dossier ou fichier.

FIG. 3.16 – copie d'écran d'un environnement Windows

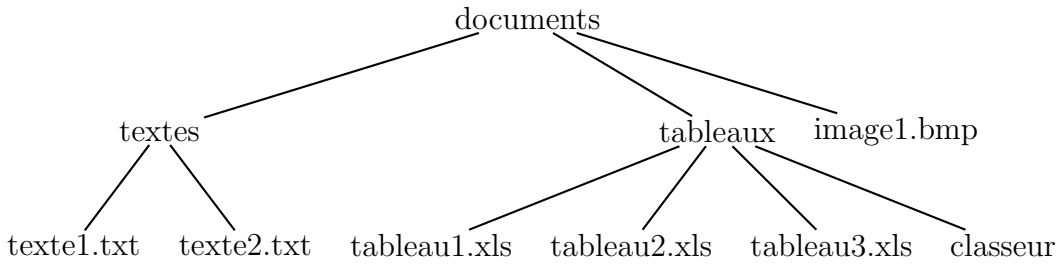


FIG. 3.17 – arborescence correspondant à l'environnement précédent

2.5 Les autres couches logicielles

Tout logiciel installé sur une machine prend place «au dessus» du système d'exploitation, avec qui il échange les données nécessaires à son fonctionnement. Par exemple, quand un logiciel prévoit une fonction «impression», la demande d'exécution de cette instruction par l'utilisateur est transmise par le logiciel au système d'exploitation, seul habilité à lancer une telle commande.

La figure 3.18 donne une reprsentation imagée de cet «empilement».

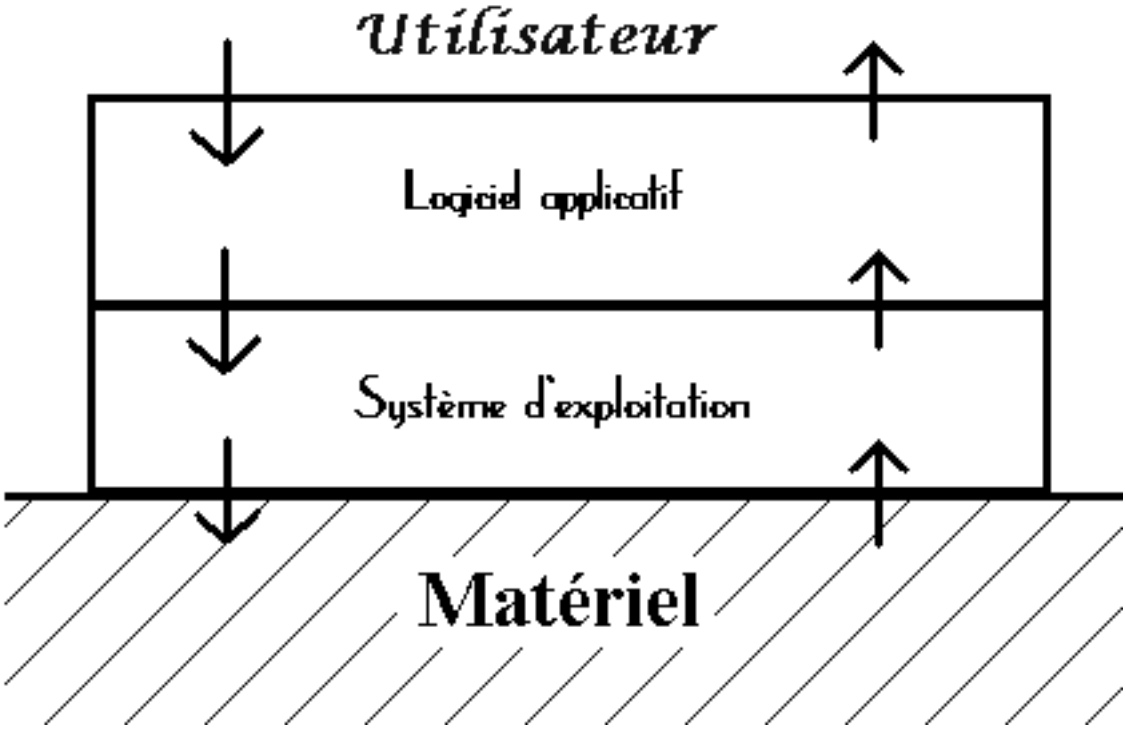


FIG. 3.18 – emplilement du système d'exploitation et d'un logiciel applicatif

Les couches logicielles peuvent bien sûr se superposer indéfiniment les unes aux autres, comme dans la figure 3.19.

Dans cette dernière figure, Netscape est un navigateur (ancêtre de Mozilla et de Firefox), un «Plug-in» est un logiciel donnant des fonctionnalités supplémentaires aux navigateurs Internet, permettant par exemple de visualiser des vidéos ou d'écouter

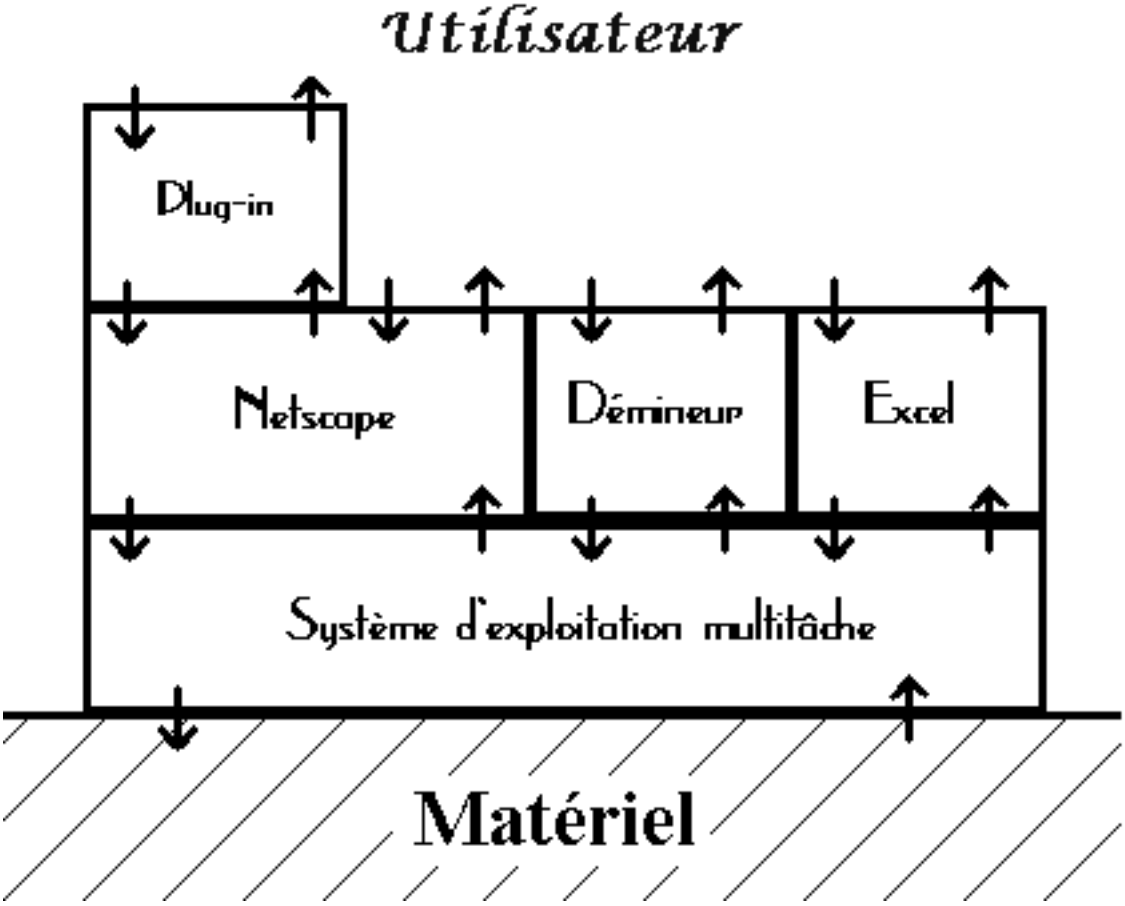


FIG. 3.19 – emplilement de plusieurs logiciels

des fichiers musicaux en MP3.

Une telle organisation en «couches» explique pourquoi quand on installe un nouveau programme dans un ordinateur, il faut tenir compte des autres couches déjà présentes (et principalement du système d'exploitation) : un programme prévu pour échanger des données/ordres avec Linux sera incapable de fonctionner avec Windows (et réciproquement).

Les machines vendues dans le commerce ont ainsi tendance à inclure de plus en plus de couches pré-installées. La tendance en informatique est donc d'«internaliser» le plus possible de traitements, pour faciliter l'usage courant des ordinateurs. Chaque couche constitue en quelque sorte une machine spécialisée dans certains services.

3 Les réseaux

Un réseau est constitué de composants reliés entre eux pour échanger des données. Le réseau téléphonique en est un exemple, existant bien avant les ordinateurs.

L'idée de connecter des ordinateurs remonte en fait aux années 60. C'était l'époque de la guerre froide entre Américains et Soviétiques. Or, il est apparu dans ce contexte que stocker des informations stratégiques (en particulier militaires) dans un ordinateur unique rendait le site de ce stockage très vulnérable à une attaque ciblée. Pour remédier à cette faiblesse, des chercheurs américains ont eu l'idée de répartir le stockage de données sensibles entre plusieurs machines distantes mais reliées entre elles, la défaillance ou la destruction de l'une d'elles pouvant être compensée par les autres. Le réseau mis en place à cette occasion n'est autre que l'ancêtre de l'Internet...

Au delà des motivations stratégiques initiales, la constitution de réseaux informatiques apporte une dimension supplémentaire à la puissance des machines. Ils permettent en particulier de *mettre en commun leurs ressources* (mémoires, processeur et périphériques). Ainsi, à l'intérieur d'un réseau, des données stockées sur une certaine machine sont disponibles à toutes les autres qui lui sont reliées (ce qui évite de les recopier à plusieurs endroits). De même, un calcul important peut être réparti entre les processeurs de plusieurs ordinateurs. Enfin, plusieurs machines peuvent être reliées par l'intermédiaire d'un réseau à une imprimante ou à un scanner, ce qui est plus économique que d'en associer à chacune d'elles. Tout se passe donc comme si la capacité de stockage (la mémoire) et la capacité de calcul (processeur) d'une machine mise en réseau étaient augmentée de celles des machines auxquelles elle est reliée.

En contrepartie de cette mise en commun, des systèmes de sécurité doivent être mis en place afin d'éviter que quiconque ayant accès aux données disponibles sur un réseau puisse les modifier ou les détruire à sa guise...

Les ordinateurs seront sans doute de plus en plus indissociables des réseaux dans lesquels ils sont intégrés. Il est donc primordial de comprendre les principes de fonctionnement de ces nouvelles organisations à la fois matérielles et logicielles.

3.1 La notion de protocole

Le réseau informatique le plus simple que l'on puisse imaginer est celui qui est constitué de deux ordinateurs reliés entre eux.

Pour qu'ils puissent s'échanger des données, ces deux ordinateurs doivent tout d'abord disposer d'un moyen physique de faire circuler entre eux des bits : en général un cordon ou un système émetteur/récepteur d'ondes. Mais cela ne suffit pas ! En effet, comment un ordinateur peut-il «savoir» s'il doit attendre des données, comment reconnaît-il le début ou la fin d'une transmission ? Il faut pour cela définir des conventions d'échange, un langage commun pour communiquer. C'est ce qu'on appelle un «protocole de communication». Nous-mêmes, quand nous téléphonons, nous utilisons un protocole implicite qui consiste généralement à échanger un «allô» puis à décliner son identité. Les fins de communications sont également balisées par des formules d'au revoir plus ou moins rituelles. Les ordinateurs procèdent de même, mais de façon systématiquement normalisées.

Imaginons par exemple deux ordinateurs A et B reliés physiquement entre eux. A doit transmettre à B le contenu d'un fichier de données. Comme un fichier peut être de taille arbitrairement grande, il ne peut être question de l'envoyer "d'un seul coup" à travers le cordon. Les données doivent tout d'abord être découpées en parties appelées «trames». Une trame est constituée d'un ensemble de bits. Parmi ceux-ci, certains sont les bits de données (le code du fichier à transmettre), d'autres sont des bits de contrôle nécessaires au protocole : il servent à numéroter les trames, à donner leur longueur, etc.

Imaginons que la taille du fichier à transmettre impose de le découper en 3 trames, numérotées respectivement 1, 2 et 3. Les étapes de la transmission du fichier peuvent alors par exemple être les suivantes :

- A envoie une demande de connexion à B ;
- B répond à A qu'il est prêt à recevoir des données ;
- A envoie à B la première trame de son fichier ;
- B confirme la réception de cette première trame ;
- A envoie à B la deuxième trame de son fichier ;
- B annonce avoir mal reçu cette deuxième trame ;
- A renvoie de nouveau la deuxième trame ;
- B confirme avoir bien reçu la deuxième trame ;
- A envoie la troisième trame de son fichier ;
- B confirme avoir bien reçu la troisième trame ;
- A annonce la fin de la connexion ;
- B accepte la fin de connexion.

Chacune de ces étapes correspond à un échange de bits : chaque phase de l'échange est codée suivant le protocole de communication commun aux deux machines. Un protocole est ainsi un langage spécifique qui comprend à la fois des mots clés (pour signifier le début ou la fin d'une transmission) et des règles (redemander une trame qui a été mal reçue, etc.). Il est bien sûr stocké sous forme de couche logicielle dans la mémoire de chaque ordinateur qui l'utilise.

3.2 topologie des petits réseaux

Quand on veut étendre la notion de réseau à plus de deux composants (qu'ils soient des machines ou autre chose !), une des premières questions qui se pose est sa *topologie*, c'est-à-dire l'organisation, l'architecture concrète de ses connexions.

Ce paramètre est particulièrement important puisqu'on souhaite que tout point du réseau puisse communiquer avec n'importe quel autre.

La solution apparemment la plus simple consiste à relier physiquement chaque point à tous les autres, comme sur le schéma de la figure 3.20.

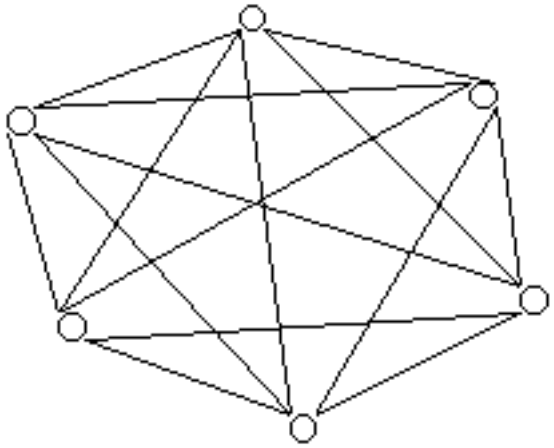


FIG. 3.20 – réseau à connexion complète

Mais une telle solution n'est pas du tout économique en nombre de connexions physiques (il faut mettre des fils partout !) et est de ce fait inapplicable.

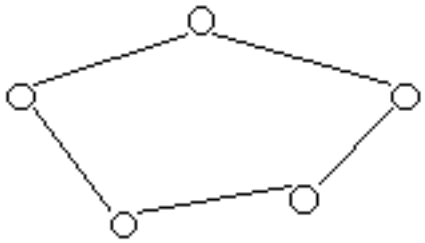


FIG. 3.21 – réseau en anneau

Pour les réseaux informatiques de petite taille (jusqu'à 100 machines environ), les topologies adoptées sont généralement plutôt en anneau (figure 3.21), en étoile (figure 3.22) ou en bus (figure 3.23). Notons que dans le cas d'un réseau en étoile, le point central auquel sont reliées toutes les machines n'est pas lui-même une machine mais un composant particulier (dans les réseaux informatiques, on l'appelle un «hub»).

Toute topologie «hybride», obtenue en combinant ces différentes possibilités, est aussi possible. Ces organisations sont celles employées dans les réseaux locaux, c'est-à-dire ceux qui permettent de relier des machines à l'intérieur d'un même site de travail (comme une entreprise ou une université).

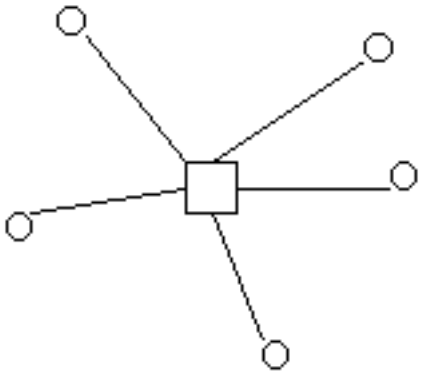


FIG. 3.22 – réseau en étoile

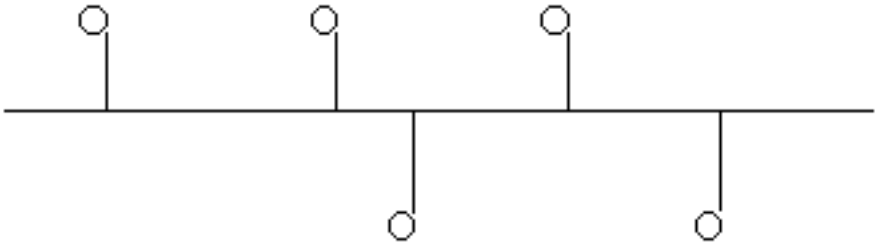


FIG. 3.23 – réseau en bus

3.3 Exemples de protocoles pour les petits réseaux

Les trames circulant sur de tels réseaux doivent spécifier les références de leur expéditeur et de leur destinataire. Les protocoles de communication pour l'échange de données dans les réseaux locaux doivent en outre prendre en compte la topologie avec laquelle ces réseaux sont constitués. On donne ici la description de deux protocoles parmi les plus simples et les plus courants.

- protocole CSMA (Carrier Sense Multi Access), utilisé dans les réseaux locaux de type «Ethernet», en bus ou en anneau. Dans ce protocole, quand une machine veut émettre des données à destination d'une autre, elle commence par se mettre «à l'écoute» de la ligne à laquelle elle est reliée. Si aucune trame n'y circule déjà, alors l'émission est autorisée. Il y a alors un risque que plusieurs machines aient effectué ce test au même moment, et donc que plusieurs trames soient envoyées ensemble sur la même ligne et entrent en «collision». Un tel événement, quand il survient, a pour effet d'interrompre la connexion. Chacune des machines émettrices doit donc ré-envoyer la trame perdue. Pour éviter qu'une autre collision se produise immédiatement, le protocole stipule que chaque machine émettrice doit attendre pendant un temps aléatoire (tiré au sort par chacune d'elle) avant de procéder à cette ré-émission. Il semblerait que ce protocole soit efficace pour des réseaux n'excédant pas 1 à 2 km. Au delà, les collisions entre trames sont incessantes et le réseau perd beaucoup en efficacité.
- protocole du jeton, utilisé pour les réseaux en anneaux. Un «jeton» est une suite de bits circulant en permanence (dans un certain sens, toujours le même) de machine en machine sur l'anneau. Il symbolise le droit d'accès à ce réseau et peut donc se trouver dans deux états possibles : un état «libre» et un état «occupé». Le principe du protocole qui l'utilise est qu'une machine ne peut émettre des données qu'à condition qu'elle prenne possession du jeton.

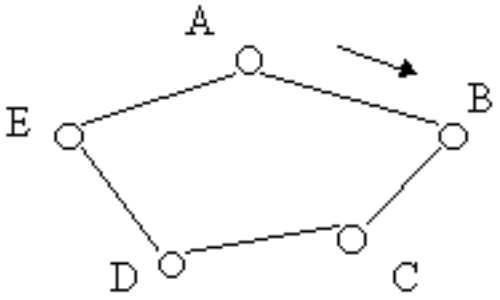


Fig. 3.24 – exemple de transmission dans un réseau en anneau

Le plus simple est de l'illustrer sur un exemple. Imaginons donc que dans l'anneau de la figure 3.24, dans lequel la flèche indique le sens de circulation du jeton, la machine A veuille transmettre des données à la machine C. On suppose de plus que le jeton arrive dans l'état «libre» à la machine A. Les étapes de la transmission d'une trame seront alors les suivants :

- A met le jeton dans l'état «occupé» et envoie la trame destinée à C (dans le même sens de circulation que le jeton) ;

– B fait suivre le tout ;
– C se reconnaît comme destinataire des données, il les conserve donc et envoie un accusé de réception à destination de A ;
– D fait suivre le tout ;
– A reçoit l'accusé de réception de C, il remet le jeton dans l'état «libre» et le fait suivre.

Pendant tout le temps où le jeton est dans l'état «occupé», aucune autre machine n'a le droit d'émettre des données. Un tel protocole, plus contraignant que le précédent, permet le fonctionnement de réseaux de plusieurs dizaines de km. Plusieurs améliorations sont possibles, comme le fait de faire circuler plusieurs jetons en même temps, ou la constitution d'un réseau en «double boucle», où chaque connexion est dédoublée, afin d'éviter les pannes en cas de coupure d'une ligne.

3.4 Les réseaux maillés

Quand il s'agit de relier entre eux un très grand nombre de composants (comme pour le réseau téléphonique) ou quand on veut interconnecter entre eux une multitude de petits réseaux locaux (comme pour Internet), une autre organisation est nécessaire.

La topologie la plus adaptée est alors celle du réseau *maillé*. Comme son nom l'indique, elle se propose de constituer un filet (*net* en anglais !) dont les points extrêmes sont les composants à relier et dont les «mailles», les noeuds intermédiaires sont ce que nous appellerons des *relais*. Un réseau maillé peut ainsi prendre la forme de la figure 3.25 (les relais sont figurés par des rectangles noirs, les ronds peuvent eux-mêmes désigner des réseaux locaux) :

3.5 Deux protocoles possibles dans les réseaux maillés

Comment transitent les données entre deux points dans un réseau maillé ? Quel protocole y est mis en place ? On distingue deux stratégies, répondant à deux types d'usages différents : la *commutation de circuits* (mode de fonctionnement des téléphones fixes) et la *commutation par paquets* (qui régit le fonctionnement de l'Internet). Chacune correspond à un protocole particulier, qui englobe les protocoles précédents (et constitue donc des couches s'empilant sur celles correspondant à ces protocoles).

– la commutation de circuits :

  Le principe de cette méthode est de réserver une ligne physique reliant les deux points du réseau durant tout le temps où ils s'échangent des données. Les relais sont alors plutôt appelés des «commutateurs» ; ils servent d'aiguillage dans le maillage. Cet aiguillage est constant pendant toute la durée de la connexion.
  C'est ainsi que fonctionnent les communications entre téléphones fixes. Jusque dans les années 70 en France, des standardistes devaient jouer le rôle de «commutateurs» humains, pour mettre en contact les correspondants. Cette fonction a depuis été automatisée.

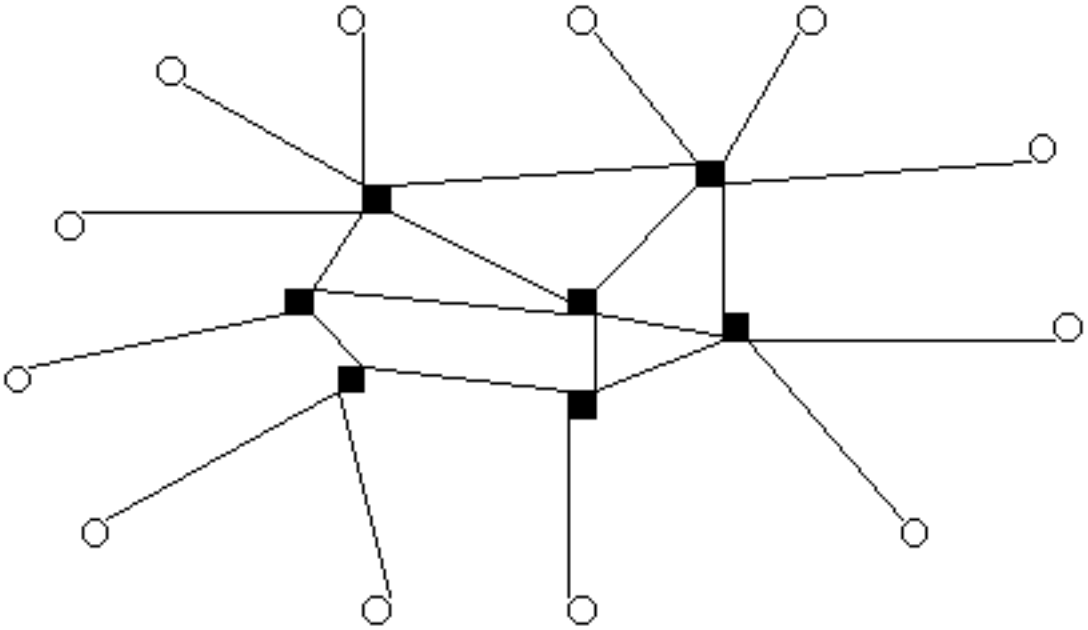


FIG. 3.25 – réseau maillé

Cette méthode est bien adaptée à la transmission en continue de grandes quantités de données. Son principal inconvénient est que la recherche d'un circuit libre entre deux points donnés est une procédure difficile et coûteuse qui doit être renouvelée à chaque nouvelle émission, et également en cas de panne en cours de communication. De plus, quand un tel réseau est saturé, il n'y a rien d'autre à faire que d'attendre la libération d'une ligne. Enfin, pour que tout se passe bien dans une transmission par commutation de circuits, l'envoyeur et le destinataire doivent fonctionner à la même vitesse ; sinon, c'est le point le plus lent qui impose sa vitesse.

– La commutation par paquets :
Cette méthode a constitué l'innovation majeure qui a permis le développement du réseau Internet. Son principe est de découper les données à transmettre entre deux points en «paquets» autonomes qui vont transiter dans le réseau indépendamment les uns des autres. Un paquet contient plus d'information qu'une trame : il peut contenir plusieurs trames. Pour qu'une telle stratégie fonctionne, il faut que les paquets contiennent toutes les informations nécessaires à la reconstitution du message. Un paquet inclut donc notamment :
  – les coordonnées de son envoyeur ;
  – les coordonnées de son destinataire ;
  – les données elles-mêmes à transmettre (du moins une portion d'entre elles, sous forme d'une ou plusieurs trames) ;
  – la place du paquet dans l'ensemble des données à transmettre.

  Les relais sont dans ce cas appelés des «routeurs». Ce sont généralement eux-

mêmes des ordinateurs qui décodent les informations de chaque paquet et prennent l'initiative de les faire suivre sur une ligne ou une autre. Les paquets provenant d'un même fichier transitent donc indépendamment les uns des autres sur le réseau et peuvent suivre des chemins différents. A leur arrivée, ils doivent être remis dans le bon ordre (soit par le dernier routeur soit par le récepteur) pour que le message complet transmis soit reconstitué.
Il est possible d'utiliser le réseau Internet, qui fonctionne ainsi, pour faire passer des communications téléphoniques. La transmission donne alors parfois l'impression d'être hachée, discontinue. Ce phénomène s'explique justement par le fait que la conversation orale est alors découpée en paquets et que leur réordonnancement à l'arrivée prend du temps, surtout si l'un d'eux met plus de temps que les autres à parvenir à destination.
Cette technique, plus souple que la précédente, est bien adaptée à des besoins de connexions simultanées et multiples, ce qui est le cas le plus courant en informatique. Elle est aussi plus fiable que la précédente. En effet, en cas de panne d'un des routeurs ou d'une des lignes, le chemin des paquets est simplement modifié en cours de route sans qu'une nouvelle émission soit nécessaire. De même, des paquets peuvent continuer à être émis même si le réseau est saturé, parce que les routeurs (qui, rappelons-le, sont des ordinateurs spécialisés) ont eux-mêmes une mémoire interne. Ils peuvent donc stocker provisoirement des paquets en attente.
Enfin, une telle organisation ne nécessite pas de synchronisation préalable entre l'émetteur et le destinataire. Elle s'arrange parfaitement de matériels hétérogènes.

3.6 L'Internet

L'Internet est certainement le réseau le plus connu, et la littérature le concernant est pléthorique. Nous ne ferons qu'effleurer ici quelques unes de ses caractéristiques. Internet est en fait un *réseau de réseaux* associant des sous-réseaux de structures hétérogènes.

Ce qui le caractérise, c'est un ensemble de protocoles de communication permettant à ces structures de communiquer entre elles via des routeurs, au-delà de leurs différences techniques.

Internet trouve son origine dans la stratégie militaire évoquée au début de ce chapitre. Son ancêtre s'appelait ARPAnet (réseau conçu pour l'ARPA ou Advanced Research Project Agency, le département de la défense américain) et est né en 1969. A l'époque, il a relié quatre universités de la côte ouest américaine, leur permettant des connexions à distance et des échanges de courriers électroniques.

Dès 1972, une quarantaine de lieux s'étaient joints à ce premier réseau, et des structures équivalentes avaient vu le jour en France et en Angleterre. C'est à ce moment que le protocole TCP (pour Transmission Control Protocol) a été défini, à l'initiative de Vint Cerf. Il inclura plus tard le protocole IP (Internet Protocol), et on parle désormais de TCP/IP. En 1980, ces protocoles sortent du secret militaire et entrent dans le domaine public ; n'importe qui peut les utiliser gratuitement. C'est ce qui a permis à Internet de prendre le pas sur les autres réseaux naissant à

cette époque, au point qu'à partir de 1988, le nombre de connexions double chaque année. L'année suivante voit la création du RIPE (Réseau IP Européen), chargé de coordonner Internet en Europe, et deux informaticiens du CERN (le laboratoire de physique nucléaire à la frontière franco-suisse) inventent le moyen de définir et de transmettre des documents hypertextes : c'est la création du langage HTML (Hyper Text Markup Language), du protocole HTTP (Hyper Text Transfer Protocol) et la naissance du World Wide Web (littéralement «toile d'araignée mondiale»). Cette nouvelle fonction commencera à prendre un essor considérable grâce à «Mosaic», le premier «navigateur» doté d'une interface graphique, bientôt relayé par Netscape en 1995.

A l'heure actuelle, les principales fonctions d'Internet sont les suivantes :

- l'échange de courriers électroniques ;
- la messagerie instantanée
- le transfert de fichiers ou FTP (File Transfer Protocol) ;
- le contrôle à distance d'une machine par «Telnet» ;
- l'échange d'informations dans des forums de discussion comme les «News» ;
- la visite des sites multimédia du Web.

L'accès à chacune de ces fonctions pour un particulier nécessite un minimum d'équipements. Tout d'abord, il faut des moyens matériels. En plus, bien sûr, d'un ordinateur, un modem ou un boîtier ADSL est nécessaire pour relier celui-ci avec le réseau téléphonique (voir partie 2.8), dont l'infrastructure est utilisée pour se rattacher à un routeur. Mais l'accès aux routeurs eux-mêmes est le privilège de sociétés ou d'organismes qui ont payé ce droit et qui s'appellent des «pourvoyeurs de service». Tout particulier qui souhaite disposer d'un point d'entrée sur Internet doit donc s'abonner à un tel pourvoyeur. Les universités françaises sont leur propre pourvoyeur de service : elles sont rattachées au Réseau à haut débit Renater, spécialement mis en place pour la recherche et l'enseignement supérieure. L'abonnement à un pourvoyeur donne généralement droit à une adresse électronique et à la possibilité de stocker des pages personnelles sur un serveur, c'est-à-dire une machine contenant des données accessibles en permanence à partir de tous les points du réseau. Enfin, un ordinateur connecté physiquement à Internet ne sera capable d'accéder vraiment aux données de ce réseau que s'il dispose des couches logicielles adaptées aux protocoles TCP/IP. Ces programmes peuvent être soit spécialisés dans une certaine fonction («Thunderbird» ne permet que de gérer son courrier électronique) soit plus généralistes (avec «Mozilla», on peut s'occuper de son courrier, «surfer sur le Web» et éditer des pages HTML).

Les protocoles TCP/IP peuvent aussi être utilisés dans le cadre d'un réseau local privé : on parle alors d'Intranet. Un tel réseau rend disponibles les fonctions d'Internet (en particulier la visite de pages Web), mais à un nombre restreint d'utilisateurs. Ils sont de plus en plus utilisés dans les entreprises ou les grandes organisations.

L'explosion actuelle d'Internet semble promettre l'émergence d'un «nouvelle économie» (via notamment le commerce électronique) et de services nouveaux (pour la formation à distance, par exemple). Elle pose aussi des problèmes juridiques et éthiques inédits dans les domaines de la sécurité (paiement électronique, piratage, téléchargements illégaux, etc.), du fichage de données personnelles et des droits d'auteurs, problèmes amplifiés par le caractère «hors frontière» de tout ce qui se passe sur Internet.

3.7 Le Web et son avenir

Le Web désigne l'ensemble des pages hypertextes lisibles à partir des navigateurs, et l'ensemble des données auxquelles elles donnent accès (fichiers sons, images, vidéo, etc.). Chaque page est désignée par son URL (Uniform Ressource Locator), appelée aussi son adresse. La particularité du Web est la présence de liens entre pages, qui permettent justement la «navigation» de page en page par de simples clics de souris.

Le Web est né en 1991 à l'initiative d'un informaticien du Cern, Tim Berners-Lee. On peut dire que le Web est à l'Internet ce que le logiciel est au matériel dans un ordinateur. Il est aussi structuré en réseau, mais c'est un réseau virtuel, dont la structure est complètement indépendante de la structure physique de l'Internet : la création de liens est laissée à l'initiative des créateurs de pages. Deux pages stockées dans des serveurs situés aux antipodes l'un de l'autre sur la planète peuvent être très proches si elles contiennent des liens qui permettent de passer de l'une à l'autre. Le Web s'est ainsi développé spontanément, grâce aux multiples contributeurs bénévoles qui l'ont investi. Il constitue à l'heure actuelle une source de données et d'informations quasi inépuisable.

L'un de ses créateurs, Tim Berners-Lee cherche maintenant à promouvoir une évolution importante : il s'agit de rendre les données du Web (essentiellement textuelles, à l'heure actuelle) compréhensibles non seulement aux humains mais aux ordinateurs, afin d'améliorer la recherche d'information et le raisonnement automatique à l'échelle de l'Internet : c'est le projet du «Web sémantique». L'aventure du Web n'en est encore qu'à son enfance...

Chapitre 4

L'histoire de l'informatique

Nous avons choisi de n'aborder l'histoire de l'informatique qu'à la fin de notre présentation, car il nous semble inutile de faire l'archéologie d'une discipline dont on ne connaît pas les principes. Faire l'histoire de l'informatique, c'est suivre en parallèle l'évolution de 3 domaines :
(a) les méthodes de codage (données) ;
(b) le calcul automatique (traitements) ;
(c) la conception d'être artificiels (simulation).
Ces 3 domaines serviront à baliser chacune des avancées ayant contribué d'une façon ou d'une autre à l'émergence de l'informatique.

1 Préhistoire

Quelques méthodes de calcul dans l'antiquité (b) :
- les premiers «algorithmes» connus datent des babyloniens (1 800 av. J.C.) ;
- abaques : planches de bois avec des rainures sur lesquelles on faisait glisser des cailloux (calculi), en respectant la numérotation de position ;
- utilisation de bouliers avérée depuis très longtemps en Russie et en Asie.

Quelques mythes faisant intervenir des êtres artificiels (c) :
- il est question de «servantes d'or» dans l'*Iliade* d'Homère ;
- légende antique de Pygmalion (sculpteur) et de Galatée (une de ses oeuvres devenue vivante) ;
- légende juive du Golem (être d'argile créé par le rabbin Loew et évoluant dans le ghetto de Prague vers 1580).

2 Ancêtres et précurseurs

- XIIIe : fabrication de l'Ars Magna, par Raymond Lulle (b) : il s'agit d'une «machine logique» faite de cercles concentriques contenant des mots qui, disposés dans un certain ordre, forment des questions tandis que d'autres mots y répondent
- XVIe : invention du codage binaire par Francis Bacon (a) et du logarithme (à l'origine créé pour simplifier des calculs compliqués) par Napier (b)

- 1624 : Wilhem Schickard construit une «horloge automatique calculante» à Heidelberg
- 1642 : Blaise Pascal, à 19 ans, crée la «Pascaline», machine à calculer mécanique à base de roues dentées, capable de faire des additions et des soustractions (b) ; le langage informatique PASCAL sera plus tard ainsi nommé en son honneur
- 1673 : Leibniz, grand mathématicien, améliore la Pascaline en y ajoutant la multiplication et la division ; par ailleurs, il s'intéresse beaucoup à la numérotation binaire avec laquelle il essaie de concevoir une «caractéristique universelle» dont l'objectif est de réduire toutes les opérations logiques à un calcul (b)
- XVIIIe : La Mettrie, philosophe disciple de Descartes, radicalise la philosophie de ce dernier et écrit *L'homme machine*, où il argumente en faveur d'une vision mécaniste du vivant (c) (Descartes lui-même aurait construit un automate à visage humain). Les automates sont très à la mode à cette époque. L'horloger suisse Vaucansson en construit de très célèbres parmi lesquels un joueur de flûte et un canard pourvu de fonctions locomotrices et digestives, exposés à paris en 1738 : leur fonctionnement utilise un «arbre à came» (comme dans les boîtes à musique), codage binaire du mouvement (a). Un célèbre «joueur d'échec artificiel» parcours aussi les cours européennes à la fin du siècle (il aurait notamment battu Napoléon) avant qu'on ne démasque la supercherie : un nain caché sous la table actionnait en fait le mécanisme (c).
- 1805 : Jacquart crée les métiers à tisser automatiques, qui utilisent des «programmes» sous forme de cartes perforées, également utilisées dans les pianos mécaniques (a)
- 1818 : Mary Shelley publie «Frankenstein», où l'électricité donne l'étincelle de vie à un être composé à partir de morceaux de cadavres (c)
- 1822 : l'ingénieur anglais Babbage fait les premiers plans de sa «machine à différences», sorte de machine à calculer mécanique utilisant les logarithmes (b) : trop complexe pour la technologie de l'époque, elle ne sera construite d'après ces plans qu'au XXième siècle.
- 1832 : invention du langage Morse, pour coder les caractères de l'alphabet (c'est un code binaire, composé uniquement des deux symboles : un trait court et un trait long) (a)
- 1833 : Babbage conçoit sa «analytical engine», encore plus performante (et compliquée) que la «machine à différence», utilisant des cartes perforées pour enchaîner l'exécution d'instructions élémentaires sur un calculateur universel (mécanique) : il passera sa vie et se ruinera à essayer en vain de construire sa machine (a, b). Il sera aidé par Lady Ada Lovelace, fille du poète Lord Byron, qui écrira les premiers «programmes» qu'aurait pu exécuter la machine (le langage de programmation ADA sera ainsi nommé pour lui rendre hommage). Cette machine aurait pourtant répondu aux besoins croissants en calcul dans la société anglaise, notamment pour l'astronomie et la navigation.
- 1854 : Le logicien anglais Georges Boole publie son livre *The Mathematical Analysis of Logic*, où il définit les opérateurs logiques dits «booléens», fondés sur deux valeurs 0/1 pour coder Vrai/Faux (a)
- 1876 : Bell invente le téléphone (a)
- 1884 : L'ingénieur américain Hollerith dépose un brevet de machine à calculer

automatique
- 1890 : Hollerith commercialise des machines à calculer électriques, utilisées notamment pour traiter automatiquement les données d'un recensement aux Etats-Unis (b). Les besoins industriels en calcul automatique se multiplient.
- 1896 : Hollerith crée une société appelée «Tabulation Machine Corporation», qui deviendra en 1924, «International Business Machine» (IBM), qui existe toujours...
- 1921 : invention du mot «robot» par Karel Capek, auteur dramatique tchèque, dans une de ses pièces (c)
- 1925 : Vannevar Bush, ingénieur américain, construit un calculateur analogique au MIT (Massachusetts Institute of Technology, prestigieuse école d'ingénieur américaine)
- 1927 : la télévision et la radio deviennent opérationnels (a)
- 1931 : l'allemant Konrad Zuse construit un calculateur automatique, le Z1 (b)
- 1936 : Alan Turing propose sa définition des «machines de Turing» et Church invente le «lambda-calcul», qui se révèlent avoir des capacités de calcul équivalentes (b)
- 1938 : fondation de Hewlett Packard, société de matériels électroniques
- 1939 : John Atanassoff et Clifford Berry, son étudiant, conçoivent un prototype appelé ABC à l'université de l'Iowa, reconnu comme le premier ordinateur digital (b)
- 1939-1945 : pendant la guerre,
  - Alan Turing travaille dans le service anglais de décryptage des messages secrets allemands (codés suivant le système appelé «Enigma») : il réalise une machine à décrypter qui contribuera à la victoire des alliés (a, b) ; en 1941, il construit le « Colossus » à l'université de Manchester (bientôt suivi du Mark I et du Mark II), premiers ordinateurs européens avec le Z3 de Konrad Zuse qui, pour la première fois, propose un contrôle automatique de ses opérations
  - John Von Neumann, travaille sur les calculs de balistique nécessaires au projet *Manhattan* (conception et réalisation de la première bombe atomique américaine) (b).
- 1945 : John Von Neumann écrit un rapport où il propose l'architecture interne d'un calculateur universel (ordinateur), appelée désormais «architecture de Von Neumann».
- 1946 : construction de l'ENIAC à l'Université de Pennsylvanie, dernier gros calculateur électrique programmable (mais pas universel) : il fait 30 tonnes, occupe $160m^2$ et sa mémoire est constituée de 18000 tubes à vide, sa puissance est équivalente à celle d'une petite calculette actuelle (b) ; pendant ce temps, Wallace Eckler et John Mauchly conçoivent le Binac (Binary Automatic Computer), qui opère pour la première fois «en temps réel» mais ne sera construit qu'en 1949, avec l'apport de Von Neumann
- 1947 : invention du transistor (qui peut être vu comme un interrupteur miniature)
- 1948 : Claude Shannon publie sa *Théorie mathématique de l'information*, où est introduite la notion de quantité d'information d'un objet et sa mesure en

bits (a) ; l'année suivante il construit la première machine à jouer aux échecs

3 Histoire contemporaine

A partir de cette date, l'ordinateur existe et son histoire matérielle se réduit donc à l'évolution des progrès technologiques, qu'on découpe habituellement en termes de «générations». Les avancées conceptuelles les plus spectaculaires concernent, elles, principalement la conception de nouveaux langages de programmation évolués.

3.1 Première génération : les monstres

- 1949 : construction de l'EDVAC, premier ordinateur construit suivant l'architecture de Von Neumann et stockant ses données sur disques magnétiques
- 1950 : Turing écrit un article dans une revue philosophique pour argumenter que le modèle des ordinateurs peut réaliser tout ce que fait l'esprit humain
- 1952 : IBM commercialise les premiers ordinateurs à lampes et à tubes à vide, IBM 650 puis IBM 701
- 1954 : premiers essais de programmation avec le langage FORTRAN (FORmula TRANslator), encore utilisé de nos jours pour le calcul scientifique
- 1955 : invention du mot «ordinateur» en France, à la demande d'IBM
- 1956 : le terme d'Intelligence Artificielle est inventé lors d'une conférence à Dartmouth, aux Etats-Unis

3.2 Deuxième génération : intégration du transistor

- 1958 : l'IBM 7044, 64 Koctets de mémoire, est le premier ordinateur intégrant des transistors ; John McCarthy invente le LISP, premier langage de l'Intelligence Artificielle
- 1959 : conception de COBOL (Common Business Oriented Language) : langage de programmation spécialisé pour la gestion et le domaine bancaire, encore utilisé de nos jours et du langage LISP (List Processing), adapté à des applications d'intelligence artificielle
- 1960 : conception de ALGOL (ALGOrithmic Language), langage évolué de calcul scientifique

3.3 Troisième génération : les circuits intégrés

- 1962 : le terme «informatique» est créé en France par contraction de «information automatique»
- 1964 : utilisation des circuits intégrés (circuits électroniques miniatures)
- 1965 : le premier doctorat (thèse) en informatique est attribué à l'université de Pennsylvanie ; conception du langage BASIC (Beginners' All-purposes Symbolic Instruction Code) et du langage PL/1 (Programming Language 1)
- 1969 : premier essai de transfert de fichier à distance par le réseau Arpanet, ancêtre d'Internet ; invention du langage PASCAL par Nicklaus Wirth

– 1971 : introduction des disquettes pour l'IBM 370 ; conception du langage LOGO, destiné à l'initiation pédagogique aux concepts de la programmation

3.4 Quatrième génération : les micro-ordinateurs

– 1972 : conception du langage C, particulièrement adapté à la programmation et à l'utilisation de systèmes d'exploitation
– 1973 : apparition des premiers micro-ordinateurs munis d'un clavier et d'un écran ; création de MICRAL, le premier micro-ordinateur français, et invention à Marseille du langage PROLOG (PROgrammation LOGique), par Alain Colmerauer
– 1975 : Bill Gates commercialise le langage BASIC et crée la société Microsoft avec Paul Allen ; le premier magasin spécialisé en informatique ouvre en Californie
– 1976 : conception du langage Smalltalk, qui introduit la programmation «orientée objet»
– 1977 : création de la société Apple par Steve Jobs et Steve Wozniak et commercialisation de l'Apple II, premier micro-ordinateur largement diffusé

3.5 Cinquième génération : l'interface graphique et les réseaux

Les japonais avaient annoncé pour les années 90 l'apparition d'un nouveau type d'ordinateurs «cinquième génération» dédiés à des applications d'Intelligence Artificielle, mais ces machines d'un nouveau genre n'ont jamais vu le jour, et les évolutions majeures récentes sont plutôt à chercher du côté d'Internet.

– 1983 : conception du langage ADA (en hommage à Lady Ada Lovelace), extension du langage PASCAL, pour répondre à une demande de la NASA
– 1984 : Le McIntosh d'Apple introduit pour la première fois une interface graphique (menus, icônes...) et la souris ; conception du langage C++, version orientée objet du langage C
– 1992 : création de Mosaïc au CERN de Genève, premier navigateur permettant de visualiser des pages Web (et donc ancêtre de Netscape, Mozilla, Firefox...)
– 1995 : Windows 95 généralise l'interface graphique sur les PCs.
– 1998 : naissance de Google

Chapitre 5

Bibliographie

1 Introductions générales à l'informatique

Rares sont les ouvrages se plaçant à un niveau de détail comparable à celui adopté ici (le plus proche est sans doute [Breton et alii 98]). Les documents suivants sont donc plutôt à considérés soit comme des introductions succinctes ([Ghernaouati et al 99], [Fayon 99]), soit comme des compléments d'information ([Breton 90], [Ganascia 98], [Rocuet 96]).

- Breton Philippe : *Une histoire de l'informatique*, La découverte, Paris, 1987, réédité en collection «Point Sciences», Le Seuil, Paris, 1990.
- Breton Philippe, Dufourd Ghislaine, Heilmann Eric : *L'option informatique en lycée*, Hachette éducation, Paris, 1998.
- Ganascia, Jean-Gabriel : *Le petit trésor : dictionnaire de l'informatique et des sciences de l'information*, Flammarion, Paris, 1998.
- Ghernaouti-Hélie Solange, Dufour Arnaud : *De l'ordinateur à la société de l'information*, PUF, collection «Que sais-je ?», Paris, 1999.
- Fayon David : *L'informatique*, collection «Explicit», Vuibert, Paris, 1999.
- Rocuet, Jean-Luc : *Informatique générale ; fondements et technologies*, ESTA-Editions de la Roche Haute, Paris, 1996.

2 Internet et les réseaux

De très nombreux livres sont disponibles sur ce thème. Les deux références suivantes sont des introductions très faciles à lire.

- Fdida Serge : *Des autoroutes de l'information au cyberespace*, Flammarion collection «Domino» n°132, Paris, 1997.
- Guédon Jean-Claude : *La planète cyber ; Internet et cyberespace*, Gallimard collection «Découverte» n°280, Paris, 1996.

3 Autour de Turing

- Casti, John L. : *Un savant dîner*, Flammarion, Paris, 1998.
- Hodges, Andrew : *Alan Turing, l'énigme de l'intelligence*, Payot, Paris, 1988.

– Lassègue, Jean : *Turing*, Les Belles Lettres, collection «Figures du savoir», Paris, 1998.
– Lemire, Laurent : *Alan Turing, l'homme qui a croqué la pomme*, Hachette, Paris, 2004.

4 Sur l'Intelligence Artificielle

– Arsac, Jacques : *Les machines à penser*, Le Seuil, Paris, 1987.
– Crevier, Daniel : *A la recherche de l'IA*, Flammarion, collection «champs», Paris, 1999.
– Ganascia, Jean-Gabriel : *L'âme machine ; les enjeux de l'itelligence artificielle*, Le Seuil, Paris, 1990.

5 Références ludiques et sérieuses

– Delahaye, Jean-Paul : *Logique, informatique et paradoxes*, bibliothèque «Pour la science», Paris, 1995 (le premier article traite des machines de Turing).
– Delahaye, Jean-Paul : *Jeux mathématiques et mathématiques des jeux*, bibliothèque «Pour la science», Paris, 1998.
– Delahaye, Jean-Paul : *L'intelligence et le calcul*, bibliothèque «Pour la science», Paris, 2002.
– Delahaye, Jean-Paul : *Complexités : aux limites des mathématiques et de l'informatique*, bibliothèque «Pour la science», Paris, 2006.
– Hofstadter, Douglas : *Gödel, Escher, Bach ; les Brins d'une Guirlande Eternelle*, Interédition, Paris, 1985.
– Ramstein Gérard : *Requiem pour une puce*, Le Seuil, 2001.

6 Sites Web

– http ://fr.wikipedia.org/wiki/Accueil et http ://fr.wikiversity.org/wiki/Accueil
Une source quasi-inépuisable d'informations, à lire parfois avec précautions (puisque n'importe qui peut y écrire) mais les rubriques scientifiques et techniques sont en général fiables.
– http ://interstices.info/
Le plus joli site d'initiation à l'informatique : plein d'articles faciles à lire, des vidéos, des jeux, des présentations ludiques : à aller voir absolument !
– http ://www.commentcamarche.net/
Un site axé sur la technique, avec un forum pour poser des questions.
– http ://www.grappa.univ-lille3.fr/
La page de l'équipe de recherche Grappa, dont sont membres tous les enseignants-chercheurs en informatique de Lille3.

Chapitre 6

Exercices corrigés

1 Enoncés

Les exercices corrigés suivants permettent soit d'introduire des notions nouvelles sans connaissance préalable (Exercice 1), soit de récapituler plusieurs outils introduits au fil de ces pages (Exercice 2), soit enfin d'illustrer une section précise du document (Exercice 3).

1.1 Exercice 1

La méthode des codes correcteur d'erreurs, introduite ici, permet d'expliquer le fait que les transmissions numériques sont plus fiables que les transmissions analogiques.

Supposons que les données à transmettre sur un réseau quelconque soient constituées de 16 bits (ce qui correspond à un mot mémoire de notre mini-ordinateur). La ligne de transmission, n'étant pas fiable (certains bits peuvent être modifiés ou perdus pendant leur transfert), on va utiliser une version simplifiée de cette méthode des codes correcteurs pour améliorer la transmission. Elle consiste à effectuer les transformations suivantes :

- on dispose les 16 bits dans un tableau de 4 lignes et 4 colonnes ;
- on construit une ligne et une colonne de contrôle en associant à chaque ligne et à chaque colonne du tableau un nouveau bit dit «de parité» ayant la signification suivante :
  - il vaut 0 si le nombre de bits de valeur 0 dans la ligne (ou la colonne) est pair ;
  - il vaut 1 sinon.
- on transmet les 8 bits supplémentaires ainsi définis à la suite des 16 bits initiaux (d'abord les 4 bits de contrôle verticaux, puis les 4 bits de contrôle horizontaux).

Example 5 *La suite de bits 1101001101011000 donne lieu au tableau suivant :*

					colonne de contrôle
	1	1	0	1	1
	0	0	1	1	0
	0	1	0	1	0
	1	0	0	0	1
ligne de controle	0	0	1	1	

Les données à transmettre sont donc : 1101001101011000 1001 0011.

1. Montrer que si on reçoit le message suivant : 000 ?1 ?011100 ?010100 ?0110, où les « ?» représentent des bits inconnus (de valeur incompréhensible), il est possible de reconstituer en entier les données qui ont été envoyées (y compris les bits de contrôle).
2. Reconstituer autant que possible le message : 1 ? ?1101 ?110 ?0 ?001 ?0 ?1 ?10
3. Montrer que le message suivant : 111001110010110011110101, transmis suivant le même mode, est incohérent. Quelle est l'explication la plus simple pour rendre compte de cette incohérence et comment y remédier ?

1.2 Exercice 2

Voici les codes ASCII, sur un octet chacun, de quelques caractères alphabétiques :
- A : 01000001 ; B : 01000010 ; C : 01000011
- a : 01100001 ; b : 01100010 ; c : 01100011

1. Ecrire les valeurs en base 10 correspondant à ces nombres binaires.
2. Quel doivent être les codes ASCII de D et de d ? De Z et de z ?
3. Donner la description (sous forme de graphe) d'une machine de Turing permettant de transformer le code ASCII (en binaire) d'une lettre minuscule écrit sur son ruban en le code ASCII de la lettre majuscule correspondante.
4. Donner le contenu de la mémoire d'un ordinateur de type Von Neumann (avec le schéma habituel) tel que si le code ASCII d'une lettre minuscule se trouve stocké à l'adresse 0110, alors après exécution du programme contenu dans cette mémoire, le code ASCII de la lettre majuscule correspondante sera stockée à l'adresse 1001.

1.3 Exercice 3

Ci-jointe une liste de logiciels, avec leur fonctionnalité principale :
- logiciel de reconnaissance vocale : il permet de transformer des paroles prononcées devant un micro en un texte écrit dans un éditeur de texte ;
- logiciel de gestion et d'interrogation de bases de données : il permet de stocker des informations de façon structurée en les répartissant en «champs» et de les retrouver simplement en fixant la valeur de ces champs. Leur illustration la plus simple est un annuaire électronique (celui du Minitel, par exemple), pour lequel les «champs» sont des données comme : «nom», «prénom», «commune» et

l'interrogation s'effectue en utilisant des commandes du genre : nom=Dupont ET commune=Lille.

– logiciel de compréhension du langage naturel : il permet de «comprendre» un texte écrit en langage courant, c'est-à-dire de le transformer en commandes exécutables par l'ordinateur.

Dans quel ordre ces différentes couches doivent-elles être présentes dans un ordinateur (sans oublier le système d'exploitation) pour permettre d'interroger oralement une base de données ?

2 Corrections

2.1 Exercice 1

1. Pour reconstituer la valeur des bits manquant, il suffit de reconstituer le tableau ayant servi au calcul de ligne et de la colonne de contrôle et de vérifier la valeur du bit de parité sur chaque ligne et sur chaque colonne de ce tableau. Le message 000?1?011100?010100?0110 devient donc :

					colonne de contrôle
	0	0	0	**1**	1
	1	**0**	0	1	0
	1	1	0	0	0
	0	0	1	0	**1**
ligne de controle	0	1	1	0	

On ne peut déduire la valeur d'un bit inconnu qu'à condition qu'il soit le seul indéterminé sur sa ligne ou sur sa colonne (contrôle compris). Il faut donc respecter un certain ordre...

Le message envoyé était donc : 000110011100001010010110.

2. Le message 1??1101?110?0?001?0?1?10 donne lieu au tableau :

					colonne de contrôle
	1	**1**	**0**	1	1
	1	0	1	**1**	**1**
	1	1	0	**0**	0
	0	?	0	0	?
ligne de contrôle	1	?	1	0	

Tous les bits inconnus n'ont pas pu être déduits. Le message reconstitué au maximum est le suivant : 1101101111000?00110?1?10

3. Le message transmis donne lieu au tableau suivant :

					colonne de contrôle
	1	1	**1**	0	1
	0	1	**1**	1	1
	0	0	**1**	0	1
	1	**1**	**0**	**0**	**1**
ligne de controle	0	1	**0**	1	

La 3ème colonne et la 4ème ligne de ce tableau (en gras) ne respectent pas les règles de construction des bits de contrôle. La raison la plus probable de cette incohérence est une erreur de transmission du bit situé à l'intersection de cette ligne et de cette colonne. Si en effet, la valeur de ce bit passe de 0 à 1, le tableau devient cohérent.

Le principe des codes correcteurs d'erreurs est donc d'ajouter des informations redondantes aux données transmises, pour contrôler leur cohérence et donc corriger les erreurs.

2.2 Exercice 2

1. Les valeurs en base 10 correspondant aux codes ASCII fournis sont les suivantes :
   - A : 65 ; B : 66 ; C : 67
   - a : 97 ; b : 98 ; c : 99
2. La suite des codes ASCII respectant l'ordre alphabétique, les codes manquant sont :
   - D : 01000100 (68 en décimal)
   - d : 01100100 (100 en décimal)
   - Z : 01011010 (90 en décimal)
   - z : 01111010 (122 en décimal)
3. Pour passer du code ASCII d'une lettre minuscule à celui de la majuscule correspondante, il suffit de transformer le 3ème bit en partant de la gauche de 1 à 0. Par ailleurs, les deux premiers bits sont toujours égaux à «01» et les 5 derniers bits ne sont pas modifiés. Supposons donc que le code ASCII d'une minuscule soit écrit sur le ruban, la machine de Turing la plus simple possible qui aura l'effet souhaité est décrite par le graphe de la figure 6.1.
4. En termes d'opération arithmétique décimale, transformer le code ASCII d'une lettre minuscule en celui de la majuscule correspondante revient à soustraire 32 à ce code. Cette opération suppose donc que le nombre 32 soit stocké quelque part dans la mémoire (à une adresse notée ad, différente des adresses imposées par l'énoncé), et que l'instruction élémentaire suivante : «0010 0110 ad 1001» soit stockée à l'adresse référencée dans le CO. La configuration de la figure 6.2 est donc une des solutions possibles (où ad vaut 0100) .

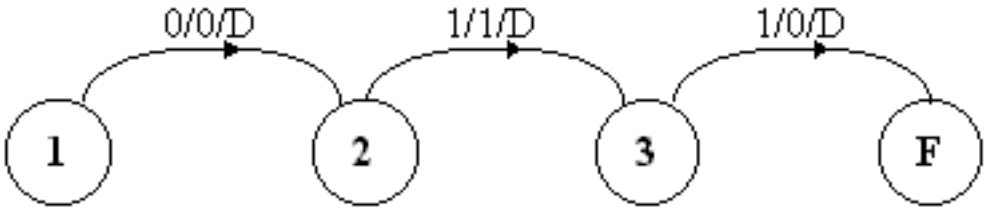


FIG. 6.1 – graphe de la machine de Turing de l'exercice 2

2.3 Exercice 3

Les couches logicielles, de la plus «externe» à la plus «interne», doivent figurer dans l'ordre suivant :

- logiciel de reconnaissance vocale ;
- logiciel de compréhension du langage (pour transformer une question comme «Quels sont les Dupond habitant Lille ?» en une commande comme : nom=Dupont ET commune=Lille) ;
- logiciel de gestion et d'interrogation de bases de données, pour obtenir la réponse à la question posée ;
- système d'exploitation, sur lequel s'appuie le logiciel précédent.

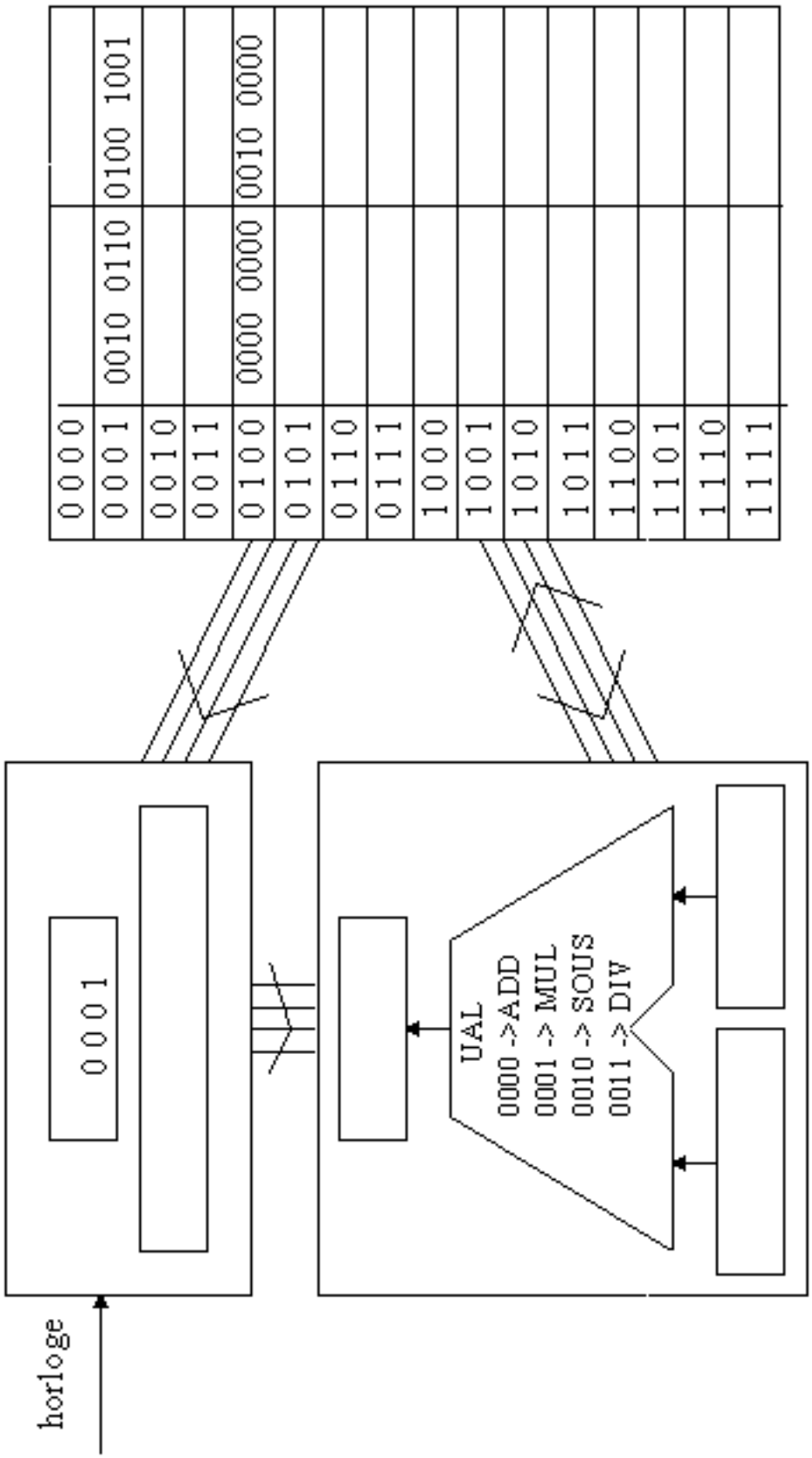


FIG. 6.2 – machine de Von Neumann de l'exercice 2