
**Département**
**Éducation**
**et Technologie**

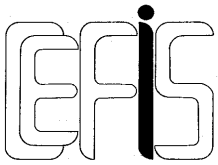
- Traitement formel
- Le codage des informations
- Architecture et fonctionnement d'un ordinateur
- La programmation et ses langages
- Le logiciel

# Initiation à l'informatique

Charles Duchâteau

5.51

Décembre 2002




Centre pour la Formation à
l'Informatique dans le Secondaire

# 1

# Introduction

## Informatique ou "computer science" ?

Mon propos n'est pas ici de réveiller le débat sur la nature de l'informatique : science, technique, culture,… mais d'expliciter pourquoi, même si c'est de tout autre chose qu'il sera question, le point de départ sera apparemment de proposer une "définition" de l'ordinateur.

> On peut se livrer à la petite expérience suivante : comme dans certains tests psychologiques, on invite les participants à associer de manière réflexe, sans longue recherche, un ou des mots à ceux qu'on va énoncer. On donne ensuite successivement des mots comme "météorologie" (qui amène "prévisions", "climat", "temps", "dépression",… et rarement "thermomètre" ou "baromètre"), "astronomie" (qui évoque "calculs", étoiles", "univers", "galaxie" … et rarement "télescope") en terminant par "informatique" (qui amène essentiellement une majorité d' "ordinateur" et quelques "information").

On peut énoncer un premier principe essentiel :

> **L'informatique n'est pas la science des ordinateurs**,
> pas plus que la météorologie n'est la science des thermomètres
> ou l'astronomie n'est celle des télescopes...

*Principe 1-1 : l'informatique n'est pas la science des ordinateurs*

Même si nous allons, dès lors, parler de tout autre chose que d'ordinateur, le fait même que le mot informatique appelle immédiatement celui d'ordinateur montre qu'il est essentiel de prendre le prétexte d'une "définition" de ce dernier pour aborder l'essentiel : le caractère formaliste des traitements d'informations exigé par les caractéristiques de l'ordinateur.

## 1.1 L'ordinateur ?

C'est une machine

à traiter des informations, de manière formelle (ou formaliste)

pour autant qu'on lui ait préalablement indiqué comment mener à bien ce traitement.

Ceci ne constitue évidemment pas une "définition" de l'ordinateur; énormément de choses manquent : rien n'est dit sur l'architecture, sur le mode de fonctionnement (on ne retient même pas le fait que cette machine nécessite une alimentation électrique)..…

Le mot "machine" ne sera pas ici davantage précisé[1]; il servira entre autre à corriger tous les anthropomorphismes que nous serons amené à faire à propos de l'ordinateur et de son

[1] On consultera à ce propos (Weizenbaum 81), surtout le chapitre 1, "L'outil".

comportement. La description de cette machine, du matériel (hardware) fera l'objet d'un chapitre dans la suite.

Bien plus important est l'aspect *traitement formel des informations* et c'est lui qui va à présent retenir notre attention.

# 2

# ... de manière formelle ...

## Ne dites plus "informatique", dites "inFORMEatique"...

Une manière frappante d'illustrer le propos du présent chapitre est de retenir le slogan suivant :

> ne dites plus «informatique »,
> dites «"inFORMEatique » !

*Principe 2-1: ne dites plus informatique*

## 2.1 Traiter des informations

Il importe ici de bien saisir dans quel sens les mots "traiter des informations" doivent être entendus dans le contexte qui nous occupe.

Un point de départ possible consiste à chercher, dans notre expérience personnelle des situations où ils nous semble bien "traiter des informations". Il est important cependant de préciser ces exemples pour éviter que n'importe quelle activité rationnelle soit présentée comme "traitement d'informations"; et cela, même si certains affirment que toute activité du cerveau humain est traitement d'informations et que, dès lors, tous nos comportements participent d'un certain traitement d'informations.

Les exemples sont fort nombreux, mais afin de pouvoir à la fois obtenir des descriptions précises, nous présenterons ces traitements d'information sous une forme commune. Les exemples obéiront donc à la schématisation suivante :

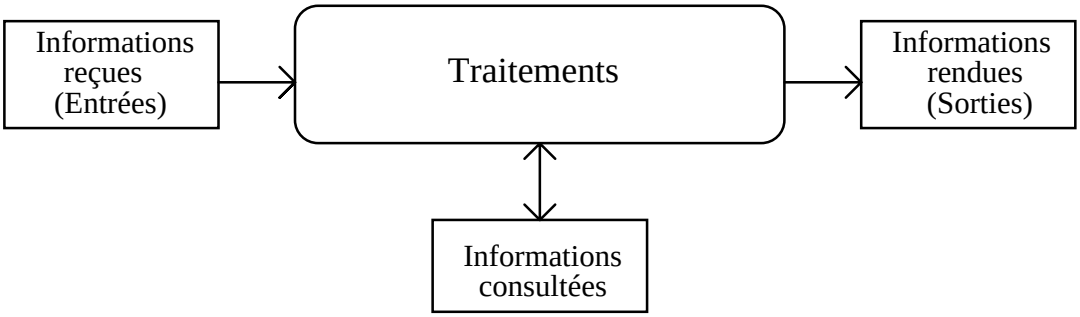



*Figure 2-1 : schéma d'un traitement d'informations*

Sans la contrainte apportée par la schématisation ci-dessus, beaucoup de propositions de "traitement d'informations" (par un être humain) comportent comme effet (comme sortie) non

des informations "objectivables", mais plutôt des comportements, des attitudes, des actions,... C'est vrai qu'au delà des traitements d'informations eux-mêmes, les canaux (en informatique, on dirait les périphériques) par lesquels les informations sont acquises ou rendues marquent une énorme différence entre les ordinateurs et les humains.

Les exemples demandent souvent à être précisés, essentiellement dans le but d'objectiver et de simplifier les entrées nécessaires et les sorties fournies.

Ainsi,

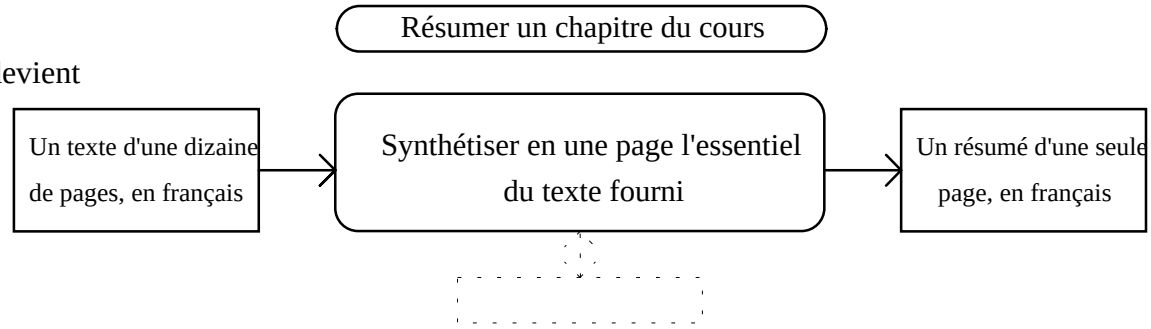



*Figure 2-2 : schéma d'un résumé*

et

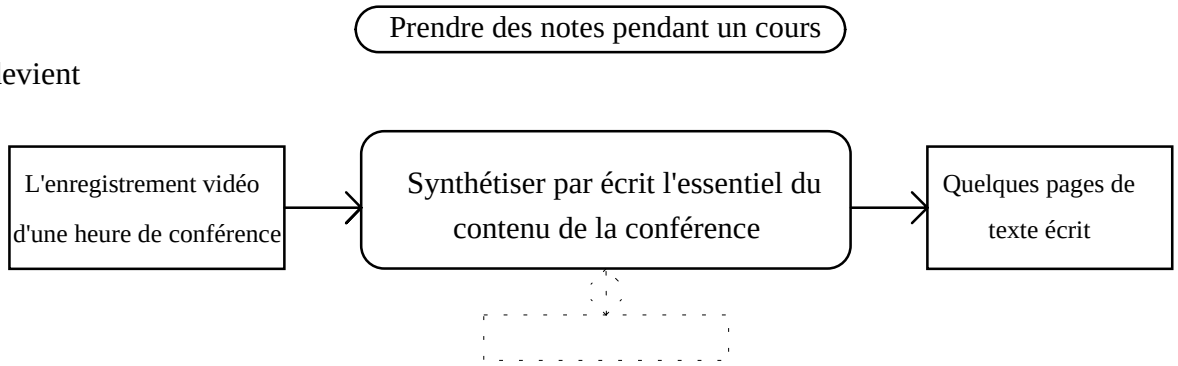



*Figure 2-3 : schéma de la synthèse d'un cours*

On pourrait encore ajouter de multiples exemples :

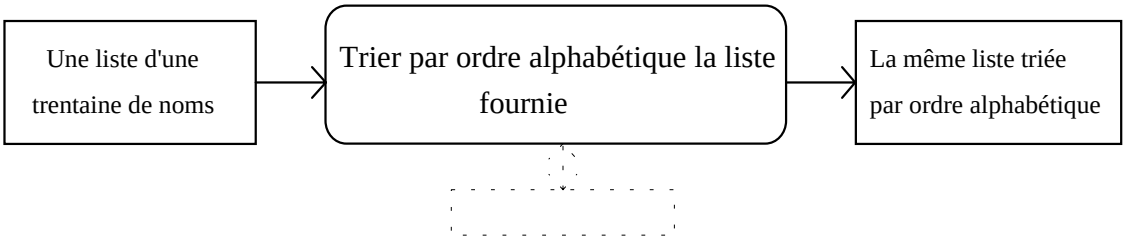



*Figure 2-4 : schéma d'un tri*

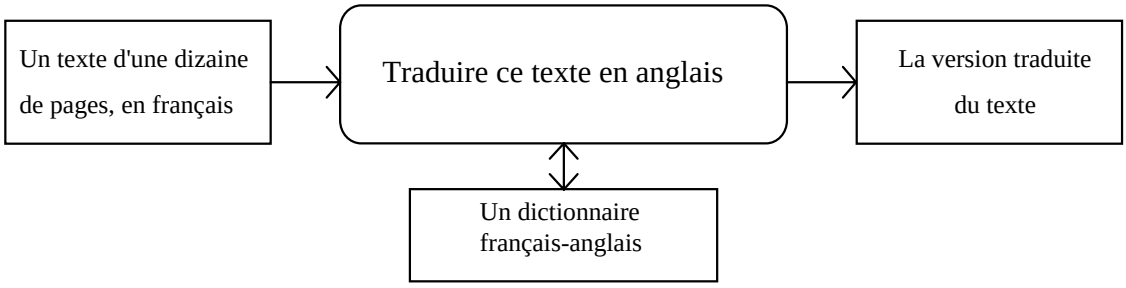



*Figure 2-5 : schéma d'une traduction*

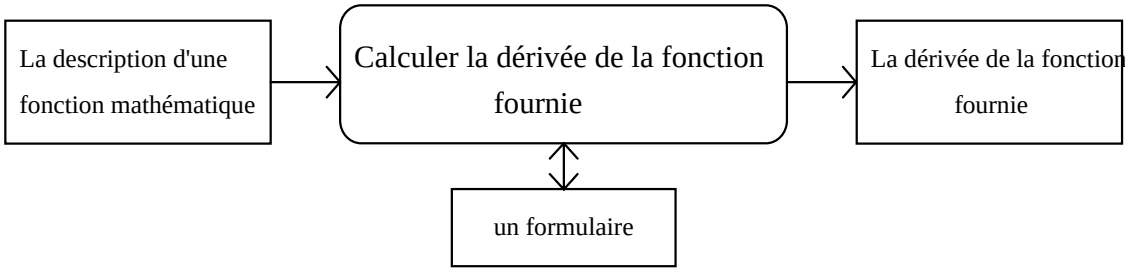



*Figure 2-6 : schéma d'un calcul de dérivée*

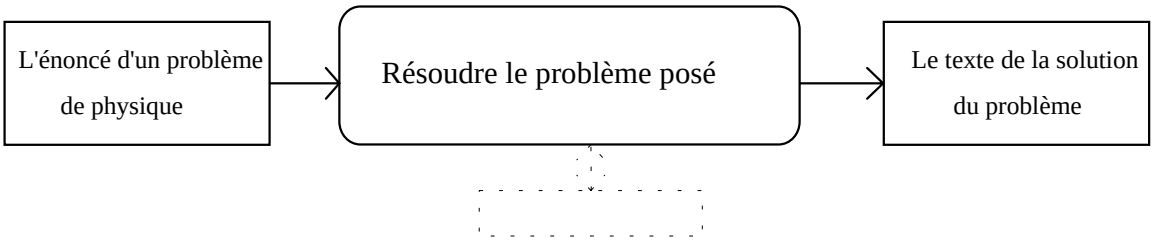



*Figure 2-7 : schéma de la résolution d'un problème*

mais aussi (même si nous n'avons pas le sentiment que le traitement d'information suivant soit d'une grande difficulté) :

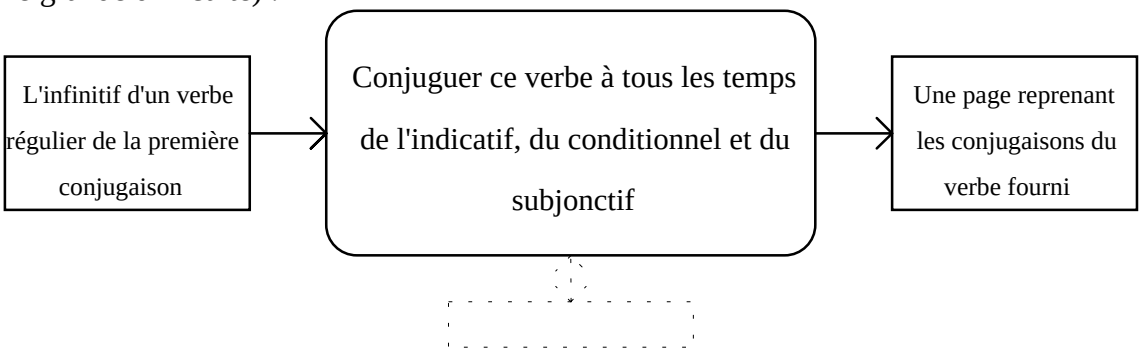



*Figure 2-8 : schéma de la conjugaison*

On pourra à raison objecter qu'un cours n'est pas (toujours) une conférence et que ce qu'en capte "en direct" un étudiant au moment de sa prise de notes est bien plus que ce qu'en restitue une cassette vidéo. J'espère qu'on aura compris que l'important ici est que les "informations" tant en entrée qu'en sortie puissent être clairement identifiées, qu'elles laissent en quelque sorte des traces objectives, formelles (dans le sens d'indubitables). La contrainte du schéma tord évidemment ce qu'un humain attache à "traitement d'informations", avec tout ce qui concerne la perception, les impressions, les sensations, … Ensuite, et ceci est loin d'être anecdotique, les exemples fournis (sauf le dernier) mettent en oeuvre des traitements sophistiqués et il est indispensable de chercher des propositions de traitements plus "bêtes"

ou plus réflexes (pour ne pas employer les termes "automatiques" ou "machinaux") si l'on veut voir émerger des exemples comme celui de la conjugaison ou ceux des manipulations numériques.

Beaucoup de propositions entrent difficilement sans le schéma prescrit, soit parce qu'elles mettent en oeuvre des comportements, des actions ou se basent sur un faisceau d'informations difficilement "objectivables", soit que les canaux de réception des informations soient trop sophistiqués, soit encore que le "traitement" attendu ne soit pas très clair. Par exemple :

- répondre à quelqu'un qui vous salue,
- confectionner un plat, d'après une recette de cuisine,
- poser un diagnostic après examen d'un malade,
- conduire une voiture,

D'autres entrent parfaitement dans le schéma attendu, au prix parfois de précisions quant à la forme des entrées et des sorties :

- traduire un texte du français vers l'anglais,
- sur base de l'énoncé du sujet proposé, écrire le texte d'une dissertation,
- répondre à des questions lors d'un examen écrit,
- choisir le menu d'un déjeuner au restaurant,

A chaque fois que c'est possible, nous classons peu à peu ces propositions pour obtenir le tableau suivant :

| | **Traitement formel ou aisément formalisable** | **Traitement difficilement (non !) formalisable (Sens, signification, ...)** |
|---|---|---|
| **Traitements portant sur des nombres** | - calculer la moyenne d'une série de cotes<br>- faire une division par calcul écrit<br>- ... | |
| **Traitements portant sur le langage (texte,...)** | - conjuguer un verbe<br>- compter les mots d'un texte<br>- vérifier si un mot est présent dans un texte<br>- fournir les fréquences respectives des mots d'un texte<br>- écrire un nombre en toutes lettres<br>- ... | - résumer un texte<br>- synthétiser une conférence<br>- traduire un texte<br>- écrire une dissertation<br>- corriger l'orthographe d'un texte<br>- ... |
| **Autres** | - chercher le numéro de téléphone d'un abonné dans l'annuaire<br>- dériver une fonction<br>- trier une liste de noms<br>- ... | |

*Figure 2-9 : classement des traitements d'informations*

Quelques remarques sont à présent indispensables :

- il est normal que la cellule correspondant au traitement non formel de nombres reste vide : les manipulations numériques sont par nature formelles[1]; pour additionner, soustraire, multiplier, … il n'est pas nécessaire que les nombres concernés aient un sens. Ce n'est pas un hasard si les premiers ordinateurs se sont appelés des *calculateurs* électroniques : le calcul (portant sur des nombres) entre parfaitement dans les "compétences" du "manipulateur formaliste" qu'est l'ordinateur[2].

  Il nous suffit de nous interroger sur ce que nous pensons que l'ordinateur pourra faire avec des nombres pour obtenir une série d'exemples corrects : additionner, soustraire, comparer,... C'est bien normal puisque les manipulations numériques de l'ordinateur sont identiques aux nôtres et ne font en aucun cas référence à une signification des nombres manipulés.

- Il est tout aussi normal que la cellule correspondant aux manipulations formalistes portant sur du texte (ou plus généralement sur des informations langagières) n'ait été complétée qu'avec peine puisque *nos* traitements portant sur du texte ne sont presque jamais formels. Un texte, d'abord nous le *lisons* pour en prendre *connaissance*; on ne *lit* pas "Notre-Dame de Paris pour le plaisir de déclarer que ce roman comporte 348.832.766 mots ou qu'on y trouve 173 fois le mot "encore".

  Essayez donc de prévoir les manipulations que l'ordinateur peut accomplir sur un texte : vous verrez, que contrairement à ce que vous pouviez de prévoir concernant les nombres, vos suggestions seront bien moins nombreuses.

  Et même quand les traitements évoqués à propos de textes sont *formalisables*, nous ne nous y attelons pas de manière *formaliste*. La conjugaison est à cet égard éclairante : conjuguer correctement peut se faire en ne tenant compte que de la forme (= de la succession des lettres) du verbe à conjuguer, mais ce n'est pas en évoquant ces règles formelles que nous conjuguons : c'est par euphonie, de manière presque réflexe et non en manipulant les lettres constituant pronoms, radicaux et terminaisons. Plus profondément, ceci veut dire que le décorticage formaliste auquel l'informatique nous oblige ne nous est *pas naturel* puisque, même pour les tâches formalisables nous ne pouvons prendre appui sur *nos* manières de procéder : nous serions bien incapables d'ailleurs en général de préciser ce que sont ces processus; ils ne font en tout cas généralement pas appel à des procédures conscientes.

- La décision de placer tel exemple dans telle cellule du tableau est évidemment en partie arbitraire. Face à la tâche "détecter les fautes d'orthographe dans un texte", on peut, selon ce qu'on sous-entend par ces mots, la considérer comme formalisable (sinon on ne parlerait pas des "correcteurs orthographiques" associés aux logiciels de traitement de texte) ou comme difficilement formalisable, si l'on attend une détection fine des erreurs grammaticales.

---

1 "On ne peut parler de signification attachée à un nombre. 4 ne veut rien dire. C'est un repère." ..."L'information numérique a donc ceci de très particulier qu'elle est un contenant sans contenu, une forme sans signification." (Arsac 87).

2 Le chapitre suivant nous montrera d'ailleurs que quelles que soient les informations que nous pensons lui faire manipuler, l'ordinateur reste essentiellement un calculateur

## 2.2 Un peu de vocabulaire et... une définition de l'informatique

Plusieurs termes ont ici été employés : formel[3], formaliste[4], formalisé[5], formalisable.

En ce qui nous concerne, l'éclairage des termes utilisés peut être synthétisé comme suit :

> La description d'un traitement d'informations est formelle ou formaliste si elle ne fait appel qu'à la forme, l'apparence des informations traitées sans jamais se référer au sens que nous leur donnons.
> Un traitement d'informations est formalisable s'il est possible d'en donner une description formelle.
> Il est formalisé, si cette description formelle a déjà été réalisée.

C'est sans doute le terme *formalisable* qui rend le mieux compte de l'effort qui est au coeur de la discipline informatique, celui d'une description purement formelle des traitements. En effet, l'informatique peut être vue comme une entreprise et une démarche (une quête sans fin ?) pour faire reculer vers la droite la frontière définie dans le tableau ci-dessus, pour élargir la colonne du formalisé et réduire celle où les traitements semblent seulement redevables du sens.

> L'objet de l'informatique, c'est en effet de transformer du formalisable en *formalisé*.

La frontière entre le *formel* et le *non formel* n'est donc pas rigide ou imperméable, puisque tout l'effort de l'informatique est de faire passer certains traitements dans la colonne de gauche. .

De plus, même si la distinction *formel - non formel* est essentielle, elle n'est pas toujours simple à établir, sauf pour ceux qui, justement, ont une longue expérience de l'informatique ou de l'utilisation des outils qu'elle a sécrété et ont dès lors intégré les contraintes relatives au caractère formaliste des traitements. Plutôt que de parler de deux catégories, il serait d'ailleurs préférable d'évoquer un continuum sur lequel on classerait les traitements d'information :

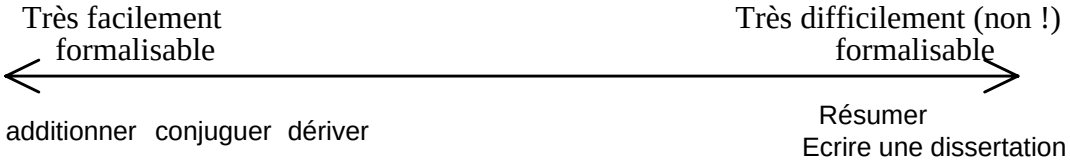



*Figure 2-10 : classement des traitements d'informations*

Si on peut aisément recenser les traitements déjà *formalisés* - il suffit d'inventorier les tâches dont un ordinateur peut venir à bout - il est par contre bien plus difficile de borner le

3 formel : qui concerne uniquement la forme. *Une distinction purement formelle. Qualités formelles d'une oeuvre*. Qui considère la forme, l'apparence plus que la matière, le contenu. *Classement, plan formel*. (Le Petit Robert, 1993).

4 Formaliste : qui observe les formes, les formalités avec scrupule. (Le Petit Robert, 1993)

5 Formaliser : réduire (un système de connaissances) à ses structures formelles. (Le Petit Robert, 1993)

domaine du *formalisable*. C'est ici essentiellement une question d'opinion : la traduction d'un texte constitue-t-elle un traitement formalisable ? et la synthèse d'un texte ? et la correction de l'orthographe ?

Il est en tout cas indispensable de proposer un critère même s'il est imparfait ou approximatif qui permette en quelque sorte de mesurer le degré de difficulté de formaliser un traitement d'information.

## 2.3 Le critère du "copain portugais"

J'ai un ami portugais qui ne connaît pas un traître mot de français, et je souhaite lui donner (en portugais évidemment) les indications nécessaires à certains travaux. Il est facile de voir que je pourrai par exemple lui fournir la marche à suivre pour qu'il puisse conjuguer un verbe français : il ne sera pas conscient qu'il est en train de conjuguer, il serait parfaitement incapable de prononcer les suites de caractères que je vais lui demander d'écrire, il ne comprendra pas ce que signifie le travail que je lui commande, mais cela n'a aucune importance : en suivant mes indications il va conjuguer en français.

Ainsi en se limitant à un verbe régulier du premier groupe, mes instructions (en portugais) pourraient être du style :

| | |
|---|---|
| - demande qu'on te fournisse le mot à traiter (par exemple sur un petit bout de papier) | on lui donne "**porter**" |
| - écris ce mot au dessus de ta feuille | il écrit **porter** |
| - efface les deux derniers caractères de ce mot | le mot devient **port** |
| - écris "je" puis passe un espace puis écris le mot raccourci auquel tu colles (sans espace) la lettre "e" | il écrit **je porte** |
| - passe à la ligne, écris "tu", puis le mot raccourci auquel tu colles (sans espace) les lettres "es" | il écrit à la ligne **tu portes** |
| ... | ... |

Même si cela est fort loin des contraintes syntaxiques des langages de programmation (voir la suite du cours), nous venons de concevoir ainsi un premier **programme** (à destination du copain portugais). Nous allons y revenir, puisque (et cela illustre bien cette difficulté de formaliser) ce premier exemple de programme est aussi le premier exemple de programme incorrect... On notera que nous sommes ici aux prises avec la quatrième partie de la "définition" préalablement donnée pour l'ordinateur : "pour autant qu'on lui ait indiqué comment mener à bien ces traitements".

De la même manière nous pourrons expliquer à notre ami portugais (même s'il n'y comprend rien) comment réaliser une addition écrite, comment calculer une moyenne, et même en prenant pas mal de temps comment calculer la dérivée d'une fonction mathématique élémentaire.

Mais si le travail consiste à donner (en portugais) les règles qui vont permettre de résumer un texte écrit en français, on devine qu'il s'agit là d'une tâche redoutable, peut-être même impossible.

Il importe de bien insister ici sur le fait qu'il s'agit de donner les indications pour un texte quelconque, qu'on n'a évidemment pas devant les yeux lorsqu'on rédige ces indications.

Sinon, si c'était de la synthèse de tel texte précis qu'il s'agissait, la solution serait immédiate : je le résume moi-même, je donne ce résumé à mon ami qui n'a plus qu'à le rendre. Ce que je veux c'est lui indiquer comment résumer **n'importe quel texte**, c'est avoir **formalisé** le processus de synthèse d'un texte.

On voudrait que face à un texte qu'il ne comprend pas (puisque c'est un texte français), où il ne voit qu'une succession de lettres et de mots, il puisse à l'aide de nos indications produire un autre texte français (qu'il ne comprendra pas davantage) et qui constitue un résumé du premier... On mesure j'espère toute la difficulté du processus.

Et on devine que des stratégies formalistes simples comme "réécrire le texte en ne retenant qu'un mot sur cinq" conduiront à un charabia qui ne constituera en aucune manière une synthèse du texte original.

Le même défi se rencontre si la travail consiste à traduire un texte du français vers l'anglais. On pourrait penser à une approche consistant à fournir au copain portugais un lexique français - anglais qui comporterait par exemple les équivalences :

| | |
|---|---|
| il | he |
| la | the |
| porte | door |

ce qui conduirait immanquablement à traduire la courte phrase "il la porte" par "he the door"...

Mais ce qu'il est essentiel de saisir c'est que si demain on dispose pour ces diverses tâches des indications nécessaires à destination du copain portugais, après-demain, on pourra les faire faire par l'ordinateur : le problème de leur formalisation sera réglé.

Le propos n'est évidemment pas ici d'assimiler les capacités d'un humain avec celles d'un ordinateur. Mais du point de vue de la nécessité de formaliser (décrire les traitements sur la seule base de la forme des informations à traiter, sans jamais faire intervenir la signification, le sens que nous attachons à ces informations), le passage par un homme ne comprenant pas notre langue mais auquel nous pouvons prêter par ailleurs les mêmes possibilités de traitements formels que les nôtres (il peut compter, additionner, repérer une suite de caractères dans un texte,...) est éclairant.

On lira à ce propos la magnifique contribution "Esprits, cerveaux et programmes" de John Searle dans (Hofstadter 87).

## 2.4 L'ordinateur, un manipulateur formaliste d'informations

Même lorsqu'une énorme couche de logiciel[6] transfigure l'ordinateur et le rend capable de traitements sophistiqués (mais toujours *formalisables* et même *formalisés*, puisque justement l'ordinateur en est capable) des indices nous montrent clairement que l'on n'échappe pas à la contrainte de formalisation.

Ainsi, par exemple, la plupart des logiciels actuels de traitement de texte comportent un correcteur orthographique, capable de détecter dans le texte traité les "fautes d'orthographes". Voici ce que donne l'action d'un tel correcteur orthographique sur un texte (que je vous recommande de lire à haute voix) :

6 le logiciel, ce sont les *indications de traitement*, conçues par un homme (ou une équipe) et dont il est question dans la pseudo-définition apportée plus haut.

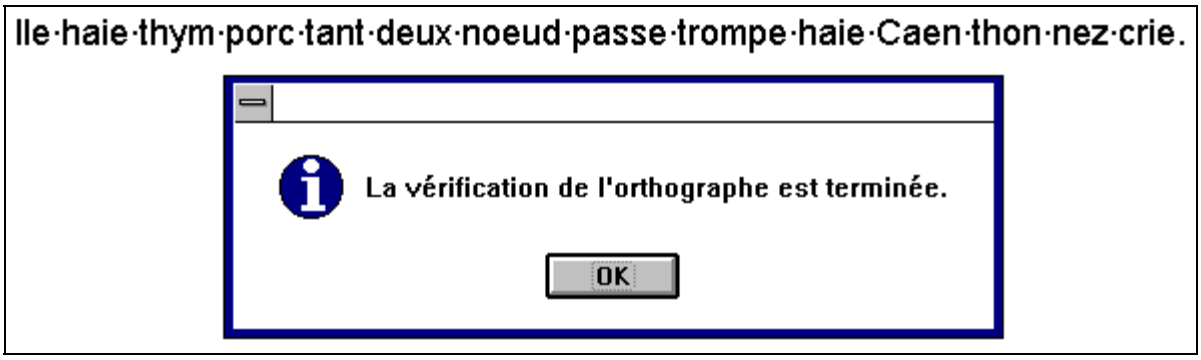



*Figure 2-11 : correction orthographique*

On appréciera le sens de la phrase écrite ci-dessus et pour lequel le traitement de vérification effectué ne détecte aucune faute d'orthographe.

On devine aisément que les indications consistent essentiellement dans ce cas à munir l'ordinateur d'une interminable liste de tous les mots acceptables (noms au singulier et au pluriel, formes conjuguées, articles, adjectifs,...) : une erreur est détectée chaque fois qu'un mot du texte n'est pas dans cette liste; sinon, c'est correct.

Il y a mieux : certains logiciels de traitement de texte sont accompagnés d'un "correcteur grammatical". Voici ce que donne, face à la même phrase, le "correcteur grammatical" de mon traitement de texte favori :

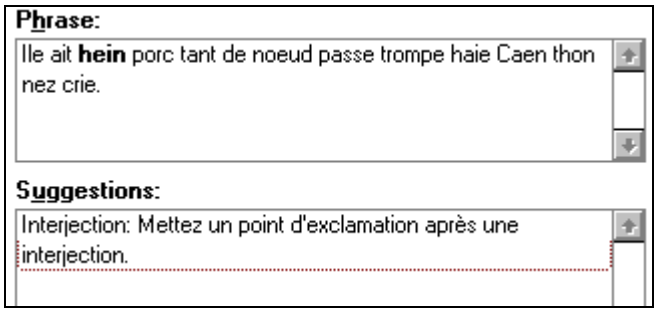



*Figure 2-12 : correction grammaticale*

Il y a mieux : lorsqu'un mot fautif est détecté, une liste de corrections envisageables est proposée par le correcteur orthographique. Il est amusant de décrypter dans les mots alors proposés la stratégie - évidemment purement formelle- suivie par le programmeur (= celui qui a donné les indications).

Ainsi la phrase "J ai Laicé desborder le lê", conduit le correcteur orthographique à épingler le mot fautif "Laicé", qui voit alors proposer comme candidats "Alice, Laciez, Laissé, Laisse, Laisser, Laissez, Laîche, Lainé, Laité, Laies, Lacé, Lancé". On appréciera le fait que comme l'original, les mots proposés le sont en tout cas avec une majuscule. "desborder" lui ne provoque l'apparition que de deux propositions (dont on notera la proximité sémantique avec "déborder" : "dessouder" et "destroyer" ! Enfin, "lê" conduit aux suggestions "le, lé, lai, lei, let"... Voilà autant d'illustrations du caractère formaliste des traitements.

Si l'on veut bien, pour quelques instants, se mettre dans la "peau de l'ordinateur" et pour en terminer avec ces problèmes de mots, tout pour "lui" (y compris nos plus beaux textes français) est comparable à ce que sont pour vous les "textes" qui suivent :

*Figure 2-13 : premier texte en langue inconnue*

ou



*Figure 2-14 : second texte en langue inconnue*

ou mieux encore



*Figure 2-15 : troisième texte en langue inconnue*

Je vous laisse le soin de souligner les mots fautifs du premier, de mettre au passé le second et d'écrire au pluriel, chaque fois bien entendu que vous le jugez opportun, le troisième...Vous voilà maintenant "le portugais" de chacun de ceux qui, comprenant parfaitement ces textes, devraient vous donner pour chacune de ces tâches les indications nécessaires.

## 2.5 La programmation

On a déjà pu goûter aux plaisirs de la programmation en écrivant les indications nécessaires pour faire conjuguer notre ami portugais. Ce n'est pas mon propos ici d'en dire plus sur le "faire faire", qui est au coeur de la programmation et qui se cache dans ma pseudo-définition d'ordinateur sous la petite phrase "*pour autant qu'on lui ait indiqué comment mener à bien ce traitement*".

J'avais signalé ci-dessus que ce premier exemple de "programme pour faire conjuguer" était aussi le premier exemple d'un programme incorrect. On aura vu que si, fort de nos

explications, notre ami portugais se voit confier le soin de conjuguer le verbe "aimer", il fournira sans sourciller "je aime", puisque c'est là ce que nos indications lui ordonnent. On pourra toujours alors corriger[7] la première version en ajoutant les considérations indispensables sur les verbes commençant par une voyelle. Puis on se rendra compte (toujours en se limitant aux verbes réguliers du premier groupe -ceux en "er"-) qu'il y a aussi le problème du "h" (j'hésite, mais je hume); ces "détails" réglés, j'aurai la surprise de voir l'ami portugais, servi par un utilisateur facétieux, conjuguer sans problème "je cerisie" (première personne de l'indicatif présent du verbe "cerisier") ou "je plombie"... et il me faudra alors lui fournir, en plus des indications, la liste des verbes acceptables...

C'est cela le défi qui est au coeur de la programmation : comment décrire, sur la seule base de la forme, sans jamais référer au bon sens ou à une commune expérience, des traitements d'informations que *nous n'accomplissons pas de manière formaliste*. Je l'ai évoqué plus haut, nous conjuguons essentiellement par euphonie et non pas en appliquant je ne sais quelle règle; nous conjuguons sans réfléchir... Et même lorsqu'à l'école fondamentale nous avons appris à conjuguer, l'essentiel était déjà fait : nous écrivions sans doute parfois "ils porte", jamais "vous portons".

Vous pouvez passer en revue les tâches précédemment classées comme "aisément formalisables" (écrire un nombre en toutes lettres, conjuguer, trier, calculer une dérivée) et vous admettrez sans doute que l'adverbe aisément est peut-être un peu exagéré... sauf lorsqu'on compare ces tâches à celles de la seconde colonne (résumer, traduire, synthétiser,...).

C'est ce qui fait sans doute de la programmation -même si d'autres difficultés viennent de ce que l'ordinateur est infiniment plus formaliste encore que notre ami portugais - une des activités les plus inhabituelles et les plus complexes.

## 2.6 Et l'intelligence ?

Il est à présent évident que, de notre point de vue d'être humain, d'autres termes peuvent être accolés à l'axe qui va de l' "aisément formalisable" au "très difficilement formalisable".

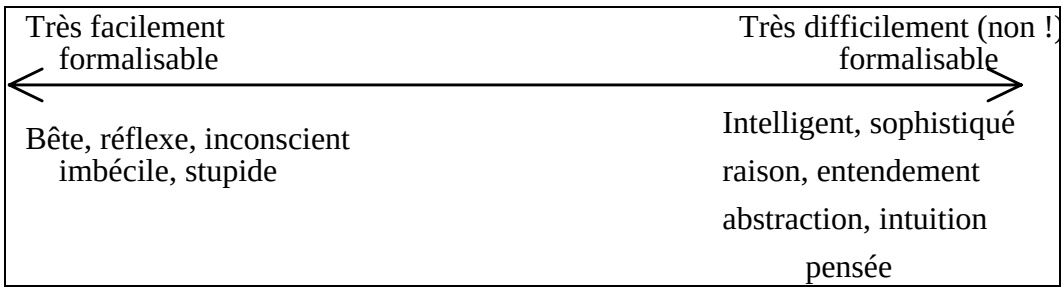



*Figure 2-16 : axe de la difficulté de formaliser*

Plus les traitements concernés par le processus de formalisation exigé se situent à droite sur l'axe, plus ils correspondent à des tâches qui réclament de notre part "de l'intelligence". "Compter les mots d'un texte" n'est pas un travail très sophistiqué (pour nous !)[8]; "synthétiser un texte" est un travail comportant pas mal de "matière grise ajoutée".

---

7 En informatique on dit "debugger" (ôter les "bugs"); même si l'exotisme du terme en amoindrit peut-être la portée, il s'agit bien de corriger des erreurs...

8 Même si la description des traitements formalisés qui décrivent ce processus n'est pas simple; on retrouve à nouveau le fossé entre le "faire" (immédiat, réflexe,...) et le "faire faire" (qui ne peut se baser que sur des considérations de forme); c'est ce fossé qui constitue toute la difficulté de l'activité de programmation.

Dès lors, plus l'informatique s'attaque à des tâches difficilement formalisables pour tenter de les formaliser, plus le comportement de l'ordinateur disposant de l'énorme masse des indications nécessaires à l'accomplissement de cette tâche - indications toujours formalistes - nous paraît "intelligent".

Battre un "grand maître" au jeu d'échecs était (est !) sans conteste considéré comme une preuve d'intelligence; aujourd'hui l'ordinateur équipé des indications indispensables [9] gagne aux échecs, même quand il est opposé aux grands maîtres. Guidé par un système expert adéquat, l'ordinateur pose des diagnostics médicaux, aide à détecter les pannes des locomotives, seconde les spécialistes dans la détection des champs de pétrole,...

Ceci, vu ce qui vient d'être dit, montre dans un premier temps que *tous ces processus ont pu être formalisés* et dans un second temps pose la question de "l'intelligence", non de l'ordinateur[10], mais du couple ordinateur-indications de traitement.

Tout ce que je voudrais signaler ici c'est que, me semble-t-il,

1 cette question a du sens

2 il ne faut pas confondre "intelligence" et "conscience"

Si l'on veut bien en effet parler non *d'intelligence*, mais de *comportements intelligents*, la question du comportement intelligent d'un système (même informatisé) me semble pouvoir être posée[11] et mérite plus qu'un haussement d'épaules ou que les anathèmes de ceux qui définissent l'intelligence comme "ce que ne font pas (encore) les ordinateurs".

Il faut sans doute dissocier la question de l'intelligence et celle de la conscience : vous pouvez me montrer votre intelligence, jamais vous ne pourrez me prouver votre conscience ou le fait que vous "pensez". Le seul être dont je sois certain qu'il est conscient et qu'il pense, c'est moi. (et encore!!!).

## 2.7 Questions récapitulatives

(Surtout 1, 2, 3, 7 et 8). Les autres sont là pour les "mordus".

1. Pouvez vous classer chacun des exemples de traitements d'informations suivants sur l'axe qui va du "facilement formalisable" au "très difficilement formalisable". Justifiez et expliquez votre réponse. Si vous pensez que le traitement est "assez aisément formalisable", imaginez sans entrer dans le détail comment on pourrait s'y prendre .
   a. Détecter dans un texte écrit en français les fautes d'orthographe d'usage ou les fautes de frappe.

      *Informations en entrée* : le texte dactylographié

      *Informations en sortie* : le même texte avec les mots fautifs soulignés.
   b. Corriger dans un texte français les fautes d'orthographe d'usage et les fautes de frappe.

      *Informations en entrée* : le texte dactylographié

9 Ce qu'on appelle le "logiciel", et dans le cas du jeu d'échec on en devine la taille et la complexité...

10 Un ordinateur "tout seul" ou "tout nu" (= sans logiciel) , ça n'existe pas; et comme ça n'existe pas, ça ne peut forcément pas "être intelligent".

11 et l'a été depuis bien longtemps (avant même la naissance de l'ordinateur) par des gens comme Turing qui considérait même la question plus choquante "Les machines peuvent-elles penser ?" (Cf. Hofstater 87, pp. 61-76).

*Informations en sortie* : le même texte avec les mots fautifs corrigés.

c. Calculer la dérivée d'une expression mathématique.

*Informations en entrée* : la fonction à dériver

*Informations en sortie* : la fonction dérivée

d. Détecter dans un texte écrit en français les fautes d'orthographe grammaticales.

*Informations en entrée* : le texte dactylographié

*Informations en sortie* : une liste des erreurs détectées avec les mots du texte original qui sont en cause.

e. Fournir pour n'importe quel mot français une liste des traductions anglaises possibles.

*Informations en entrée* : un mot français

*Informations en sortie* : une liste de mots anglais constituant des traductions possibles du mot fourni.

2. Pouvez vous donner un exemple de traitement formaliste d'un texte qui apporte certaines "connaissances" sur ce texte.

3. Que pensez-vous de la définition formelle suivante d'un "mot" ?

   C'est une suite non vide de caractères qui

   - commence le texte ou est précédé d'un caractère délimiteur
   - termine le texte ou est suivi d'un caractère délimiteur
   - ne contient aucun caractère délimiteur.

   Les caractères délimiteurs sont les suivants : ".?!:;,)( -" et l'espace et les guillemets.

   Pouvez-vous trouver des exemples de mots (au sens où nous l'entendons habituellement) qui échappent à cette définition et, réciproquement, des exemples de "mots" au sens de la définition ci-dessus qui ne sont pas des mots au sens habituel ?

4. Soit le petit jeu "formel" suivant : [12]

   On va écrire des expressions formelles qui vont seulement utiliser les trois symboles o, P et G.. Parmi toutes les configurations possibles, nous en distinguons que nous appellerons "Axiomes". Ce sont toutes les configurations du type

   xPoGxo

   où x n'est composé que de symboles o.

   De plus, nous dirons que si x, y et z sont des chaînes composées uniquement de symboles o et si xPyGz est un axiome ou un théorème, alors xPyoGzo est aussi un théorème.

   Pensez-vous avec ces règles que oooPooGooooo soit un théorème ?

   Et ooPooGooo ?

   Pensez-vous que dans ce contexte, un ordinateur puisse produire les théorèmes de mon petit jeu ?

   Pouvez-vous donner du SENS à ce jeu formel ? Lequel ?

[12] Cf. Hofstadter 85.

5. Pourriez-vous fournir les indications de traitement nécessaires à un non biologiste pour qu'il puisse décrire un segment d'ADN, sur base de la donnée d'un seul des deux brins.

   Appliquez votre "programme" au brin suivant :

   AATGCTA

6. Commentez le passage suivant :

   «Les molécules de la vie, protéines et acides nucléiques, portent leur information inscrite sous la forme d'enchaînements linéaires d'acides aminés (pour les protéines) ou de nucléotides (pour les acides nucléiques). Ces successions de signes codés sont analogues à celles des lettres dans un mot, des mots dans une phrase ou d'un phrase dans un paragraphe. Il est donc normal que les programmes mis au point par les informaticiens pour traiter des chaînes de caractères puissent être adaptés au traitement de l'information biologique. Ces programmes permettent le stockage, le traitement, la manipulation de l'information biologique. » (tiré de "L'aventure du vivant", de J. de Rosnay, Editions du Seuil).

7. Pourriez-vous citer l'une ou l'autre tâche à propos desquelles l'être humain parlerait de "traitement d'informations" et qui sont hors de portée (actuellement en tout cas) de l'ordinateur ?

8. Pourriez vous préciser dans quel contexte nous avons évoqué le "critère du copain portugais" ? Expliquez.

# 3

# ... des informations...

## Le codage des informations

### 3.1 Introduction

Nous savons à présent que les traitements permis par l'ordinateur auront toujours un caractère formel (ou formaliste) et que dès lors le mot "information" y prendra un sens très particulier.

Il reste cependant à dire sous quelle forme les diverses informations devront être "écrites" pour être acceptable par l'ordinateur. Sous quelle forme l'ordinateur accepte-t-il un texte, une image, des sons,...?

Ce que nous allons à présent découvrir c'est que

> Pour qu'une information soit acceptable par l'ordinateur, il faut que nous puissions la représenter, la coder par une série finie de nombres entiers.

*Principe 3-1: codage de l'information*

Les "connaisseurs" objecteront sans doute qu'en réalité, l'alphabet de l'ordinateur étant binaire, il ne dispose pour "écrire" les informations que nous lui fournissons que de 2 symboles (habituellement notés 0 et 1).

L'expérience m'a montré qu'il est inutile (et même nuisible) à ce stade de la découverte du monde de l'informatique de polluer la démarche par des détails qui n'apportent rien d'essentiel et vont au contraire mettre l'accent sur des caractéristiques anecdotiques à cette étape. Il suffit de savoir que pour l'ordinateur "tout est nombre" et non qu'en plus, ces nombres, il ne les écrit pas comme nous le faisons.

Lorsqu'on en saura plus sur la "machine" ordinateur et son organisation interne, il sera toujours temps de parler de l'écriture des nombres en binaire. Inutile d'ajouter ici des détails qui augmentent le caractère mystérieux des traitements effectués par l'ordinateur.

### 3.2 Codage

Nous sommes tellement accoutumés à employer les *signes* qui constituent notre langage écrit que nous n'imaginons plus qu'il s'agit là d'un *code*[1]. Ainsi, vous êtes actuellement en train de lire ces quelques lignes, et de décoder (sans en être vraiment

[1] Code : système de symboles destiné à représenter et à transmettre une information. (Le Petit Robert, 1993)

conscient) leur contenu. Notez qu'ici le mot "décoder" est essentiellement employé dans le sens de "comprendre". Cette lecture-décodage ne vous demande d'ailleurs pratiquement aucun effort.

C'est seulement si subitement vous vous trouvez face au "texte" ✣✲✡✶✯✌ ✶✯❁✳ ✣✳✣✳ ✡✲✲☆✶✣✳ ✡ ✯☆✲✣ ⌘✣ ☆✣✳☆✳ ✳✣✸✳✣ que ce dernier va sans doute vous faire sursauter. Il y a là une succession de signes qui n'ont aucune signification, aucun sens pour vous. C'est seulement si je vous indiquais le système de codage employé que, en associant à chaque signe un autre signe -habituel-, vous reconstitueriez, sous une autre forme le message codé, forme à laquelle vous seriez capable d'associer un sens.

Notons que nous nous retrouvons à nouveau aux prises avec des termes difficiles comme signe (ou symbole), comme sens (ou signification).

?

Pourriez vous trouver des *informations* qui se présentent sous plusieurs *formes* différentes, mais en dessous desquelles nous identifions pourtant la même *information* ?

Une même information peut souvent se présenter sous des formes diverses. Ainsi,

TELEPHONE Téléphone Téléphone ☎ )

ou encore

quatre QUATRE 4 IV IIII

Le vrai problème étant de savoir pourquoi nous disons qu'il s'agit pourtant à chaque fois de la *même* information, alors que les formes sont pourtant différentes et qu'il n'y a aucun rapport formel entre les "dessins" ☎ et Téléphone.

Il faudrait d'ailleurs à chaque fois y ajouter les sons ("katr", "téléfone"') correspondants, mais ces sons il est impossible de les écrire.

Un codage, ou un système de codage, ce sera donc une correspondance qui permet sans ambiguïté de passer d'une représentation d'une classe d'informations à une autre représentation des *mêmes* informations, suivant un ensemble de règles précises.

Ainsi, le codage dans l'alphabet MORSE associe à différents caractères une représentation faisant appel à seulement deux symboles point (.) et barre (-), ce mode de représentation étant particulièrement adapté à la transmission par télégraphe.

Lorsque nous avons profondément intégré, lors de notre apprentissage, un système de codage, le passage d'une représentation de l'information à une autre nous est naturelle et nous sommes capable de coder et décoder sans devoir beaucoup réfléchir. Ainsi en est il de la dictée où nous codons sous forme écrite des sons qui parviennent à nos oreilles[2]; ou encore de l'écriture en chiffres d'un nombre écrit en toutes lettres.[3]

La situation est plus délicate lorsque le système de codage est inaccoutumé : il nous faut alors patiemment apprendre et exercer le système inhabituel, jusqu'à ce qu'à nouveau

2 Avec les anecdotes bien connues lorsque l'information n'est pas reconnue donc mal transcrite. Lisez donc à haute voix la phrase suivante (que je transcris phonétiquement) puis tentez de l'écrire : "Lai poul cétai sôvé dai kon leur avai ouvair la porte."

3 J'aime beaucoup cet exemple car j'ai encore très vivaces dans la mémoire les souvenirs des exercices de transcription que nous faisions en 3ème ou 4ème primaire, où il s'agissait d'écrire en chiffres des nombres donnés en lettre et réciproquement. Ainsi, si aujourd'hui "vingt-deux" et "22" représentent la *même* information, cela résulte d'un apprentissage que nous avons tous mené dans notre enfance.

nous en ayons intégré les règles. Un bel exemple en est l'alphabet morse dont les débutants déchiffrent péniblement les messages lettre après lettre et que les experts sont capables de "lire" presque couramment.

En tant qu'être humain, nous parlerons évidemment plus volontiers de code ou de codage lorsque nous n'avons pas encore assimilé (par apprentissage) le système de représentation en question.

## 3.3 Des questions plus que des réponses

Notons que les quelques lignes qui précèdent posent plus de questions qu'elles n'apportent de réponses. Et ceci est lié à l'éternel débat, réveillé par l'informatique, entre *forme* et *sens*. Ce sont ces questions qui sont au coeur de disciplines comme la linguistique ou la sémiologie (même si l'on y parle plutôt de "signifiant" (forme) et "signifié" (sens)). Je ne souhaite pas ici prolonger ce débat, mais il est évident que des questions comme les suivantes sont au coeur de l'informatique :

- Puisque "4" et "quatre" représentent ou évoquent pour nous (êtres humains parlant français) la *même chose*, quelle peut bien être cette *chose* ? (Notons que le fait d'écrire "information" au lieu de "chose", même si elle montre que cette question est au coeur de l'informatique, ne change rien à la question).
- Est-ce que ce que nous nommons *sens* ou *signification* est autre chose qu'une *forme* particulière au sein de notre cerveau, qu'une certaine configuration de circuits neuronaux ?

On est évidemment loin de la technique, des RAM et des MegaBytes avec ce type de problématique. Certains trouveront sans doute ces questions sans intérêt; pourtant l'ambition de l'informatique n'est elle pas de formaliser la plupart des traitements dont nous sommes capables : lire, écrire un texte sous la dictée, reconnaître un visage au milieu d'une centaine d'autres,...

## 3.4 Codage pour l'ordinateur

Pour qu'une *information* [4](ou qu'un ensemble d'informations) soit acceptable par l'ordinateur, il faut que nous puissions la coder, la représenter sous la forme d'une série finie de nombres entiers. On verra dans la suite qu'en réalité ces nombres eux-mêmes sont écrits en binaire (et non en décimal), mais cela est sans importance pour l'instant. Nous allons donc à présent passer en revue un certain nombre d'informations données sous des formes qui nous sont habituelles et décider si ces informations sont codables sous forme de séries de nombres entiers et proposer un tel système de codage.

?

Pourriez vous imaginer un codage en une série finie de nombres entiers pour l' information suivante :

BONJOUR

A quel ensemble d'informations votre système de codage est-il applicable ? Pourrait-on l'étendre ?

4 En laissant à ce mot tout le flou nécessaire...

## 3.5 Codage de texte

Il est vraiment fort simple de découvrir une première ébauche de solution à la petite question posée ci-dessus. Il suffit évidemment d'associer lettres majuscules et chiffres

A ⇔ 1
B ⇔ 2
C ⇔ 3
... ... ...
Z ⇔ 26

Ainsi, le code peut-il se décrire par la correspondance :

**Ensemble des lettres majuscules** ⟶ **Ensemble des entiers (de 1 à 26)**

une lettre majuscule ⟶ son numéro d'ordre dans l'alphabet

et dès lors

BONJOUR se coderait par 2 15 14 10 15 21 18

Mais on devine immédiatement que, comme seules les majuscules sont codables dans ce système, il ne serait pas possible de coder les informations suivantes :

COMMENT VAS TU ?

(puisqu'on n'a pas associé de nombre à l'espace et au point d'interrogation)

et encore moins

J'hésite; peut-être, à parler d'INFORMATION !

puisque les lettres minuscules, les minuscules accentuées, les signes de ponctuation ne sont pas codés (du moins dans le code proposé).

Il est pourtant facile d'imaginer des extensions possibles pour que chacun des caractères habituellement utilisés trouve un numéro. Ceci a donné lieu à plusieurs systèmes de codage dont les plus connus sont le système ASCII et le système ANSI.

### *3.5.1 Le code ASCII*

Primitivement, ce code (qui signifie American Standard Code for Information Interchange) (Code américain standard pour l'échange d'informations) permettait de représenter 128 caractères différents.

Les caractères qui nous sont habituels commencent au numéro 32. Ainsi, à l'espace correspond l'entier 32. Les caractères-chiffres reçoivent les numéros entre 48 et 57. A a le code 65 et a le code 97.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| esp | 32 | 0 | 48 | A | 65 | [ | 91 | a | 97 | { | 123 |
| ! | 33 | 1 | 49 | B | 66 | \ | 92 | b | 98 | \| | 124 |
| " | 34 | 2 | 50 | C | 67 | ] | 93 | c | 99 | } | 125 |
| # | 35 | 3 | 51 | D | 68 | ^ | 94 | d | 100 | ~ | 126 |
| $ | 36 | 4 | 52 | E | 69 | _ | 95 | e | 101 | | |
| % | 37 | 5 | 53 | F | 70 | ` | 96 | f | 102 | | |
| & | 38 | 6 | 54 | G | 71 | | | g | 103 | | |
| ' | 39 | 7 | 55 | H | 72 | | | h | 104 | | |
| ( | 40 | 8 | 56 | I | 73 | | | i | 105 | | |
| ) | 41 | 9 | 57 | J | 74 | | | j | 106 | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| * | 42 | | | K | 75 | | k | 107 |
| + | 43 | | | L | 76 | | l | 108 |
| , | 44 | : | 58 | M | 77 | | m | 109 |
| - | 45 | ; | 59 | N | 78 | | n | 110 |
| . | 46 | < | 60 | O | 79 | | o | 111 |
| / | 47 | = | 61 | P | 80 | | p | 112 |
| | | > | 62 | Q | 81 | | q | 113 |
| | | ? | 63 | R | 82 | | r | 114 |
| | | @ | 64 | S | 83 | | s | 115 |
| | | | | T | 84 | | t | 116 |
| | | | | U | 85 | | u | 117 |
| | | | | V | 86 | | v | 118 |
| | | | | W | 87 | | w | 119 |
| | | | | X | 88 | | x | 120 |
| | | | | Y | 89 | | y | 121 |
| | | | | Z | 90 | | z | 122 |

*Figure 3-1 : le code ASCII*

On est évidemment en droit de se poser deux questions :

- pourquoi les entiers de 0 à 31 ne sont-ils pas repris pour coder certains des caractères ?
- qu'en est-il de toutes les lettres accentuées (é,à,è,ê,ü,...) et autre ç ?

1. Les premiers entiers positifs disponibles correspondait (et correspondent encore) à des caractères spéciaux : les *caractères de contrôle*. Ces derniers datent des débuts de l'informatique et nous héritons là, comme souvent, de choix faits dans le passé et dont la pertinence n'est plus aujourd'hui évidente.

   Ces caractères de contrôle servent à provoquer des réactions particulières des systèmes informatiques. Ainsi, fort souvent le caractère codé 7 provoque un bip sonore. Le caractère codé 13 est celui correspondant à l'appui sur la touche Entrée (aussi appelée Return) et marquée sur le clavier par le symbole ↵ (on appelle souvent ce caractère "retour du chariot", en anglais "carriage return" par analogie avec le "retour du chariot" des machines à écrire). De même, on le découvrira sans la suite, ces caractères de contrôle sont employés par exemple par l'ordinateur pour signaler aux imprimantes des changements à effectuer lors de l'impression : passage à des caractères italiques, gras,... alors que les caractères habituels sont simplement imprimés.

   Sur les ordinateurs de type PC ces *caractères de contrôle* s'obtiennent généralement en combinant l'appui sur une touche particulière du clavier, la touche Control (marquée Ctrl) avec l'appui sur une touche normale. Ainsi le caractère de code 1, s'obtient en appuyant simultanément sur Ctrl et a. Le caractère de code 2 s'obtient en enfonçant Ctrl et b, etc.

   Une remarque importante ici : nous aurons l'occasion de signaler plus loin dans le cours que "l'ordinateur n'existe pas", ce slogan exprimant le fait que ce que nous avons toujours en face de nous c'est un couple "ordinateur + logiciel"[5]. Il n'est donc pas possible de prévoir à coup sûr le comportement du système "ordinateur + logiciel", tant que nous n'avons pas précisé quel est ce logiciel.

5 Le logiciel, ce sont les programmes, les "indications de traitement" dont il est question dans la définition proposée au début. C'est le logiciel qui détermine le comportement de l'ordinateur lorsqu'il reçoit un "caractère".

Dès lors l'effet d'un caractère de contrôle sur le système ne peut être prévu que si l'on connaît le logiciel, le programme, qui est actuellement en train de guider, de gouverner le comportement de l'ordinateur. Ainsi, le caractère codé 7 ne provoque pas *toujours* [6]un bip sonore.

2. Le code ASCII primitif permettait de coder les 128 caractères nécessaires à l'écriture des textes anglais (qui ne connaissent pas les lettres accentuées). Face aux besoins des utilisateurs non anglophones, le nombre de caractères codés a été étendu : les entiers de 128 à 255 permettant de coder 128 caractères supplémentaires pour donner ce qu'on appelle le code ASCII étendu :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Ç | 128 | û | 150 | ¼ | 172 | ┬ | 194 | ╪ | 216 | ε | 238 |
| ü | 129 | ù | 151 | ¡ | 173 | ├ | 195 | ┘ | 217 | ∩ | 239 |
| é | 130 | ÿ | 152 | « | 174 | ─ | 196 | ┌ | 218 | ≡ | 240 |
| â | 131 | Ö | 153 | » | 175 | ┼ | 197 | █ | 219 | ± | 241 |
| ä | 132 | Ü | 154 | ░ | 176 | ╞ | 198 | ▄ | 220 | ≥ | 242 |
| à | 133 | ¢ | 155 | ▒ | 177 | ╟ | 199 | ▌ | 221 | ≤ | 243 |
| å | 134 | £ | 156 | ▓ | 178 | ╚ | 200 | ▐ | 222 | ⌠ | 244 |
| ç | 135 | ¥ | 157 | │ | 179 | ╔ | 201 | ▀ | 223 | ⌡ | 245 |
| ê | 136 | ₧ | 158 | ┤ | 180 | ╩ | 202 | α | 224 | ÷ | 246 |
| ë | 137 | ƒ | 159 | ╡ | 181 | ╦ | 203 | ß | 225 | ≈ | 247 |
| è | 138 | á | 160 | ╢ | 182 | ╠ | 204 | Γ | 226 | ° | 248 |
| ï | 139 | í | 161 | ╖ | 183 | ═ | 205 | π | 227 | ∙ | 249 |
| î | 140 | ó | 162 | ╕ | 184 | ╬ | 206 | Σ | 228 | · | 250 |
| ì | 141 | ú | 163 | ╣ | 185 | ╧ | 207 | σ | 229 | √ | 251 |
| Ä | 142 | ñ | 164 | ║ | 186 | ╨ | 208 | µ | 230 | ⁿ | 252 |
| Å | 143 | Ñ | 165 | ╗ | 187 | ╤ | 209 | τ | 231 | ² | 253 |
| É | 144 | ª | 166 | ╝ | 188 | ╥ | 210 | Φ | 232 | ■ | 254 |
| æ | 145 | º | 167 | ╜ | 189 | ╙ | 211 | Θ | 233 | | 255 |
| Æ | 146 | ¿ | 168 | ╛ | 190 | ╘ | 212 | Ω | 234 | | |
| ô | 147 | ⌐ | 169 | ┐ | 191 | ╒ | 213 | δ | 235 | | |
| ö | 148 | ¬ | 170 | └ | 192 | ╓ | 214 | ∞ | 236 | | |
| ò | 149 | ½ | 171 | ┴ | 193 | ╫ | 215 | φ | 237 | | |

*Figure 3-2 : le code ASCII étendu*

Comme le montre le tableau ci-dessus, les caractères accentués sont à présent codés. On y trouve également des caractères "graphiques" comme ╔, ║, etc.. Ces caractères peuvent servir à "dessiner" des encadrements comme


Les caractères grecs ainsi qu'un certain nombre de symboles sont également codés (par des entiers entre 128 et 255).

Signalons enfin que les 128 derniers caractères de ce code ASCII étendu peuvent être différents selon les besoins des langues utilisatrices.

6 *toujours* est un mot à éviter lorsqu'on tente de décrire des règles de comportement de l'ordinateur. Mais c'est ce dernier terme "ordinateur" qui employé seul n'a pas de sens : le seul mot pertinent est "système informatique" qui englobe alors l'ordinateur et le programme qui est en train de le contrôler. Aucune règle n'est valable pour "l'ordinateur", on peut seulement donner des règles pour "tel système".

### *3.5.2 Le code ANSI*

Enfin, à côté du code ASCII, apparaît un autre système de codage qui tend à être de plus en plus fréquemment utilisé : le code ANSI (émanant de l'American National Standards Institut). Les 128 premiers nombres (de 0 à 127) codent les mêmes caractères que le code ASCII. Par contre les 128 derniers codent des caractères différents :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 128 | – | 150 | ¬ | 172 | Â | 194 | Ø | 216 | î | 238 |
| | 129 | — | 151 | - | 173 | Ã | 195 | Ù | 217 | ï | 239 |
| ‚ | 130 | ˜ | 152 | ® | 174 | Ä | 196 | Ú | 218 | ð | 240 |
| *f* | 131 | ™ | 153 | ¯ | 175 | Å | 197 | Û | 219 | ñ | 241 |
| „ | 132 | š | 154 | ° | 176 | Æ | 198 | Ü | 220 | ò | 242 |
| … | 133 | › | 155 | ± | 177 | Ç | 199 | Ý | 221 | ó | 243 |
| † | 134 | œ | 156 | ² | 178 | È | 200 | Þ | 222 | ô | 244 |
| ‡ | 135 | | 157 | ³ | 179 | É | 201 | ß | 223 | õ | 245 |
| ˆ | 136 | | 158 | ´ | 180 | Ê | 202 | à | 224 | ö | 246 |
| ‰ | 137 | Ÿ | 159 | µ | 181 | Ë | 203 | á | 225 | ÷ | 247 |
| Š | 138 | | 160 | ¶ | 182 | Ì | 204 | â | 226 | ø | 248 |
| ‹ | 139 | ¡ | 161 | · | 183 | Í | 205 | ã | 227 | ù | 249 |
| Œ | 140 | ¢ | 162 | ¸ | 184 | Î | 206 | ä | 228 | ú | 250 |
| | 141 | £ | 163 | ¹ | 185 | Ï | 207 | å | 229 | û | 251 |
| | 142 | ¤ | 164 | º | 186 | Ð | 208 | æ | 230 | ü | 252 |
| | 143 | ¥ | 165 | » | 187 | Ñ | 209 | ç | 231 | ý | 253 |
| | 144 | ¦ | 166 | ¼ | 188 | Ò | 210 | è | 232 | þ | 254 |
| ‘ | 145 | § | 167 | ½ | 189 | Ó | 211 | é | 233 | ÿ | 255 |
| ’ | 146 | ¨ | 168 | ¾ | 190 | Ô | 212 | ê | 234 | | |
| “ | 147 | © | 169 | ¿ | 191 | Õ | 213 | ë | 235 | | |
| ” | 148 | ª | 170 | À | 192 | Ö | 214 | ì | 236 | | |
| • | 149 | « | 171 | Á | 193 | × | 215 | í | 237 | | |

*Figure 3-3 : le code ANSI*

### *3.5.3 Le code UNICODE*

Enfin, il est à peu près certain que dans les années qui viennent un nouveau code va s'imposer qui permettra, en utilisant les entiers entre 0 et 65535, de coder 65536 caractères différents.

?

Pourquoi, pensez-vous, les nombres de caractères codés dans les différents systèmes décrits (128 pour ASCII, 256 pour ASCII étendu, 65536 pour UNICODE) sont-ils ce qu'ils sont plutôt que 100, 500 ou 50.000 ?

Si nous récapitulons la démarche suivie dans le cas du codage de texte, nous constatons que

- nous avons scindé, découpé l'information à coder en une série de constituants élémentaires (dans notre cas les caractères qui composeront les morceaux de textes)
- nous avons associé un nombre à chacun des ces constituants élémentaires, des constituants différents étant bien entendu représentés par des nombres différents et chaque constituant se voyant associer un nombre;
- une information étant alors une suite de ces constituants (un texte est une suite de caractères), nous la codons grâce à la suite des nombres correspondants.

Deux remarques à ce stade :

- Remarquez à quel point cette écriture en nombres constitue bien ce que nous appelons un *codage* étant donné notre difficulté à déchiffrer le message correspondant, à le décoder. Décoder signifie d'ailleurs pour nous, passer de l'écriture "en nombres" à l'écriture habituelle "en caractères" qui va prendre du sens. Combien de temps, d'apprentissage et d'exercices nous faudrait-il pour devenir capable de lire un message "en nombres" et à ce qu'il prenne immédiatement un sens, sans repasser par l'écriture "en caractères".
- Il est important de percevoir que le rôle de ce que nous appelons en informatique un *périphérique d'entrée* [7]est d'automatiser au maximum le passage d'une forme qui nous est familière ("BONJOUR") à la forme que réclame l'ordinateur (une suite de nombres entiers). Ainsi, pour le codage du texte, le clavier (et l'intervention de programmes logés dans l'unité centrale) permet ce codage. Notons aussi que l'automatisation du processus n'est que très partielle puisqu'il faut toujours un être humain face à ce clavier pour transformer des mots écrits sur une feuille de papier ou des sons en frappes sur des touches qui activeront le codage[8].

## 3.6 Codage des nombres réels

?

Voyez-vous une possibilité (ou des possibilités différentes) de coder (sous forme d'une série d'entiers) n'importe quel nombre comportant une virgule (composé dès lors d'une partie "entière" et d'une partie "décimale", comme -23,85 ou 0,067") ?

Nous pourrions inventer bien des manières de coder les nombres et cela en fonction de la manière dont nous allons morceler l'information "nombre" en constituants élémentaires.

Ainsi, nous pouvons voir "236", comme un "mot", une suite de caractères et la coder comme telle : nous sommes alors ramené au cas précédent. Ainsi dans le texte "ils étaient 236 présents", "236" est vu simplement comme une suite de caractères et sera codé en utilisant 3 nombres entiers (par exemple à travers le code ASCII. "236" est dans ce cas un mot et non un nombre.

Mais il est de nombreux cas où 236 doit être considéré comme un nombre entier et devra pouvoir être manipulé comme tel (il faudra pouvoir le faire additionner ou soustraire d'un autre). Il est alors indispensable que 236 soit codé comme un entier.

?

Comment, lorsque nous frappons 236 au clavier, l'ordinateur peut-il savoir qu'il s'agit d'un mot (à coder en accord avec le code ASCII) ou qu'il s'agit d'un nombre entier (à coder comme tel) ?

---

7 Un *périphérique d'entrée* est un des canaux par lequel l'ordinateur reçoit des informations venues de l'extérieur. Ainsi le clavier, la souris, le scanner,... sont des périphériques d'entrée. Les *périphériques de sortie* sont les canaux à travers lesquels l'ordinateur nous rend les informations après traitement, comme l'écran ou l'imprimante.

8 Cf. plus loin d'autres manières de s'en sortir...

Le plus simple pour les entiers, puisque nous devons les coder sous la forme d'entiers est évidemment de les laisser tels quels; le codage de 236 est bêtement 236.[9]

En ce qui concerne les nombres avec une virgule, de nombreuses solutions sont envisageables :

- On peut les voir comme des "mots", on les code alors comme ces derniers par la suite des codes des caractères les constituant.
- Si on souhaite les considérer comme des nombres, on pourrait imaginer de scinder ces nombres en deux constituants : la partie entière d'une part et la partie décimale d'autre part. Ainsi le nombre -23,85 se coderait par la suite des deux entiers -23 et 85. On devine cependant immédiatement la difficulté de ce type de codage : comment coderait-on -23,085, ou -23, 0085 ? étant entendu que, vu comme des nombres 85, 085 ou 0085 sont en réalité le même nombre (les deux dernières écritures étant d'ailleurs inhabituelles).

  Si l'on persiste dans cette voie, on devine qu'il faut en quelque sorte indiquer le rang de la partie décimale (s'agit-il de 85 centièmes, de 85 millièmes, ... ?).On coderait par exemple :

| | | | | | |
|---|---|---|---|---|---|
| -23,85 | ⇔ | -23 | 85 | 2 | (= 2 chiffres après la virgule) |
| -23,085 | ⇔ | -23 | 85 | 3 | (= 3 chiffres après la virgule) |
| -23,0085 | ⇔ | -23 | 85 | 4 | |

?

Avec ce système de codage, pourriez-vous dire quels sont les nombres correspondants aux codes suivants :

| | | | |
|---|---|---|---|
| 0 | 3 | 2 | |
| -12 | 13 | 4 | |
| 22 | 123 | 2 | ? |

- On va se rapprocher du système de codage réellement employé (en tout cas au niveau du principe) en remarquant qu'on peut toujours écrire le nombre sous la forme d'une fraction dont le numérateur reprend l'entier écrit avec tous les chiffres du nombre (sans se préoccuper de la virgule) et dont le dénominateur est 1, 10, 100, 1000 ou une autre puissance de 10.

  Ainsi

| | | | | | |
|---|---|---|---|---|---|
| -23,85 | ⇔ | -2385/100 | ⇔ | -2385 | 100 |
| -23,085 | ⇔ | -23085/1000 | ⇔ | -23085 | 1000 |
| -23,0085 | ⇔ | -230085/10000 | ⇔ | -230085 | 10000 |

  qu'on pourrait encore évidemment coder, puisque le second élément du code est toujours une puissance de 10, en indiquant quelle est cette puissance :

| | | | | | |
|---|---|---|---|---|---|
| -23,85 | ⇔ | -2385/100 | ⇔ | -2385 | 2 (= le dénominateur est 100, soit $10^2$) |
| -23,085 | ⇔ | -23085/1000 | ⇔ | -23085 | 3 (= le dénominateur est 1000, soit $10^3$) |
| -23,0085 | ⇔ | -230085/10000 | ⇔ | -230085 | 4 |

[9] La question se pose évidemment de savoir comment le système informatique "comprend" lorsque nous frappons 236 au clavier qu'il s'agit du mot constitué des 3 caractères 2, 3 et 6 (qui seront alors codés en accord avec le code ASCII) ou bien qu'il s'agit de l'entier 236. Comme toujours, la réponse à cette question consiste simplement à signaler qu'un système informatique, ce n'est pas l'ordinateur "tout nu", mais plutôt l'ordinateur équipé de tel logiciel précis. C'est le logiciel qui indique alors si 236 doit être codé comme une suite de 3 caractères ou plutôt comme un nombre entier.

- Il ne reste plus qu'un tout petit pas à accomplir pour arriver au codage des réels en *virgule flottante* tel qu'il est pratiqué en informatique.

  On écrira alors :

  -23,85 = -2,385 x $10^{1}$ ⇔ -2385 1 (-2385 est la succession des chiffres significatifs, la *mantisse* du nombre et 1 est *l'exposant* de 10)

  -238,5 = -2,385 x $10^{2}$ ⇔ -2385 2 (-2385 est la succession des chiffres significatifs, la *mantisse* du nombre et 2 est *l'exposant* de 10)

  -0,02385 = -2,385 x $10^{-2}$ ⇔ -2385 -2 (-2385 est la succession des chiffres significatifs, la *mantisse* du nombre et -2 est *l'exposant* de 10)

  -23,085 = -2,3085 x $10^{1}$ ⇔ -23085 1 (-23085 est la succession des chiffres significatifs, la *mantisse* du nombre et 1 est *l'exposant* de 10)

?

Avec ce système de codage, pourriez-vous dire quels sont les nombres correspondants aux codes suivants :

| | |
|---|---|
| 1056 | 0 |
| 1056 | 3 |
| 1056 | -1 |

## 3.7 Codage de dessins

?

Imaginez que vous disposiez d'une feuille de papier portant le dessin suivant :


*Figure 3-4 : dessin à coder*

Pourriez-vous proposer un système de codage (sous la forme d'une suite de nombres entiers) pour ce dessin ?

On remarque que les informations portées par cette feuille sont en noir et blanc et sont constituées de portions de texte "imprimé" ou "manuscrit", de caricatures, de dessins, ...

Il est sans doute plus malaisé ici de proposer une stratégie. Le principe en est pourtant bien connu : c'est celui du *tramage*; on va en quelque sorte poser une grille (aussi serrée que possible) sur l'image. Deux nombres seront d'emblée important pour décrire cette grille : le nombre de petits carrés en largeur et ce même nombre en hauteur. Plus ces nombres sont élevés, plus la surface de chaque petit carré est petite (et plus le dessin tramé sera proche de

l'original). Cela fait, on examinera l'intérieur de chacun de ces petits carrés avec un critère qui permette de décider s'il doit être considéré comme noir ou comme blanc. On obtient donc pour toute l'image un quadrillage comme celui montré ci-dessous pour une partie :

*Figure 3-5 : partie agrandie d'une figure tramée*

Il ne reste plus alors qu'à en déduire une longue liste d'entiers : d'abord le nombre de carrés sur la largeur, puis ce nombre sur la hauteur, puis une succession de 0 (pour coder par exemple "carré blanc") ou de 1 (pour coder "carré noir").

Si le dessin original est en couleur, le principe du tramage ne change pas; il faut seulement plus de nombres différents pour représenter les couleurs à prendre en compte.

### *3.7.1 Discrétisation*

Il faut noter que, pour la première fois, nous avons été contraint de remplacer (coder) une image comportant (en théorie en tout cas) une infinité de points (encore que cette phrase comporte les deux termes bien embarrassants *infinité* et *point*) par une image constituée d'un nombre fini de (petits) carrés. Le processus de numérisation (transformation d'informations en une série de nombres entiers) (on dit aussi parfois *digitalisation)* s'est ici accompagné d'une *discrétisation.*

C'est la même discrétisation qui est à l'oeuvre lorsque nous remplaçons une courbe bien lisse

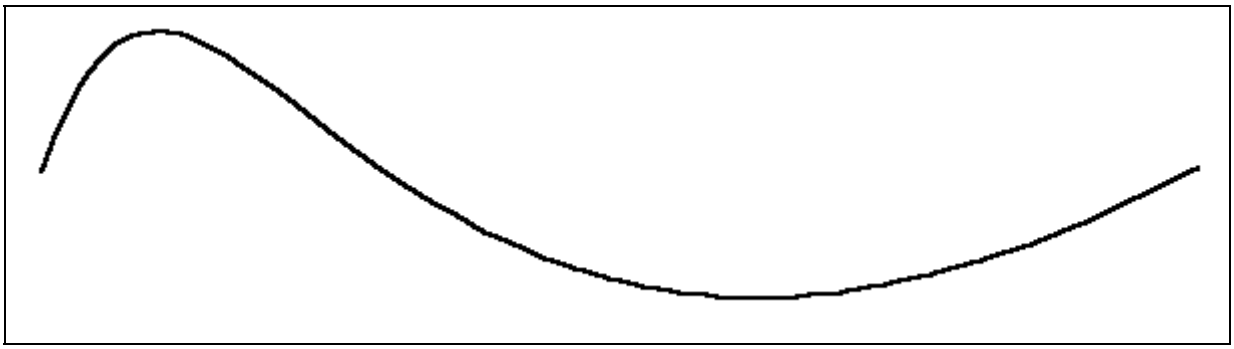


*Figure 3-6 : courbe continue*

par une approximation qui n'utilise plus qu'un nombre fini de segments :

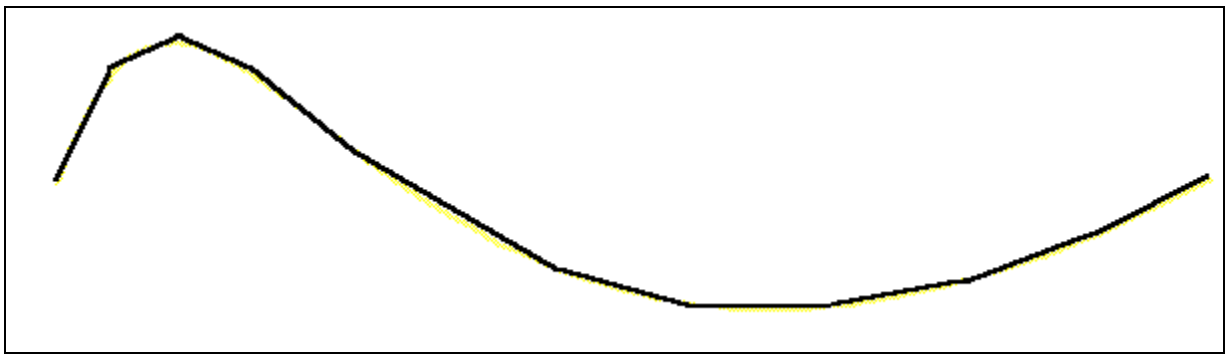


*Figure 3-7 : approximation d'une courbe*

Un processus de discrétisation[10] consiste donc à remplacer des informations ayant un caractère continu ou présentant une "infinité" de valeurs et donc non décomposables en constituants élémentaires, par une autre forme (apauvrie) de ces informations ne comportant plus qu'un nombre fini de constituants distincts et bien séparés.

### *3.7.2 Pertes d'informations*

Des termes sont évidemment obligatoirement liés à ce codage par discrétisation : approximation, passage du lisse à du morcelé, de l'infini au fini...

Dans tous les cas, on admettra que ce processus de discrétisation remplace des formes par d'autres formes qui les approximent mais sont moins riches ou moins précises que les formes originales.

Jusqu'ici, le codage se faisait sans perte d'information qu'il s'agisse de texte (vu comme suite de caractères) ou de nombre[11]. On sent bien qu'ici le codage d'une image est en réalité le codage d'une approximation de cette image et que quelque chose est définitivement perdu dans ce processus : on ne pourra plus repasser de l'image codée à l'image originale.

---

10 Dans le Petit Robert, discret est défini comme "qui ne peut prendre qu'un nombre fini ou dénombrable de valeurs". Dans (Politis 83), on trouve la définition "qui désigne ou représente des données sous forme d'éléments distincts ou séparés tels que des caractères ou des grandeurs physiques ne pouvant prendre que des valeurs distinctes ou séparées".

11 Encore que, on le verra dans la suite, les tailles possibles de la mantisse et de l'exposant seront limitées et que, dès lors, il arrivera qu'un nombre comportant trop de chiffres significatifs soit tronqué lors de son codage en machine.

### *3.7.3 Automatisation du codage bitmap*

Il existe évidemment (heureusement) un dispositif qui automatise cette opération de tramage et de numérisation : le *scanner*. On y glisse la feuille portant le "dessin" concerné et ce dernier est automatiquement tramé et transformé en une série de nombres qui sont envoyés à l'unité centrale de l'ordinateur. Pour peu que ce dernier soit équipé du logiciel adéquat, il est alors capable de reconstituer le dessin (avec les pertes d'informations liées au processus de discrétisation) et de l'afficher à l'écran. Le *scanner* est donc un périphérique d'entrée capable de procéder au codage que nous venons de mettre en évidence.

Ce que nous avons dit des périphériques d'entrée s'applique évidemment de manière duale aux périphériques de sortie : accompagnés des logiciels adéquats, ces derniers décodent les informations qui avaient été codées; autrement dit, ils sont capables de restituer les informations sous la forme qui nous est habituelle sur base des séries d'entiers qui codent ces mêmes informations à l'intérieur de l'ordinateur. Ainsi l'écran ou l'imprimante recevant de l'unité centrale les entiers codant l'image vont nous rendre celle-ci, avec l'approximation liée au mode de codage.

Le dessin comportant une "infinité" de points présenté ci-dessus est évidemment une tromperie : tel que vous le voyez, il figure sur un document préparé à l'aide de l'ordinateur; l'original a donc forcément dû subir le traitement évoqué (pour être digéré par le système informatique) : il a été discrétisé (et d'ailleurs modifié) et la version sortie sur l'imprimante (et que vous avez sous les yeux) est le décodage de la version discrétisée, donc ne comportant plus qu'un nombre fini de points. Et d'ailleurs, existe-t-il vraiment des dessins "naturels" comportant une infinité de points ? La digitalisation (et la discrétisation qui la rend possible) doit donc plutôt se comprendre comme le passage d'un dessin comportant *beaucoup* de points à une approximation en comportant *moins*. Mais comme l'esprit humain peut (?) imaginer l'infini (qui n'existe sans doute que comme une idéalisation propre à notre intelligence), il est plus facile de parler d'un processus qui ferait passer d'un dessin comportant cette "infinité" de points à une version discrétisée qui n'en comporte qu'un nombre fini.

Cette manière de coder un dessin en une série de nombres s'appelle une représentation *bitmap*.

C'est un codage de ce type qui est sous-jacent au travail de la plupart des logiciels de traitement de dessin de type "paint" : ils agissent sur des représentations discrétisées des graphiques manipulés.

Une représentation (ou un codage) bitmap d'un dessin est d'un constituée d'un nuage de points, chacun de ces points (ou plutôt de ces minuscules carrés) s'appelant un *pixel* (picture element).

?

Si les deux premiers nombres du codage suivant donnent le nombre de pixels en largeur et en hauteur et si les nombres qui suivent décrivent la couleur de chaque pixel (0 pour blanc et 1 pour noir), pourriez vous reconstituer le dessin représenté ?
13 8
0000001000000000001110000000001101100000001100011000001111111110001100000001101100000000011000000000000
(Ne passez cependant pas trop de temps à ce petit jeu !)

?

Imaginez que vous disposiez d'une feuille de papier portant le dessin suivant :

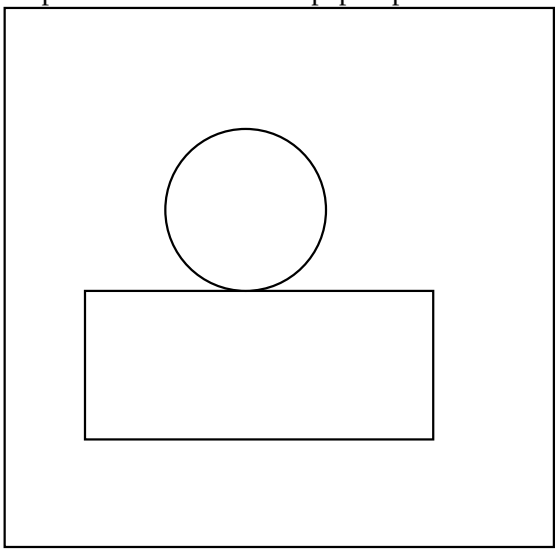


*Figure 3-8 : autre dessin à coder*

Pourriez-vous proposer un système de codage (sous la forme d'une suite de nombres entiers) pour ce dessin ?

### *3.7.4 Codage vectorisé*

Il existe une autre manière de coder un graphique, c'est d'y déceler certains *objets* et de coder la nature et la caractéristique de ces derniers. Ainsi, plutôt que de voir sur la page précédente un ensemble de points noirs et de points blancs (comme dans le codage précédent), on peut reconnaître deux objets : un rectangle (codé par exemple 3) dont le coin inférieur gauche est à 12 du bas de la feuille et à 7 du bord gauche et dont la hauteur est 14 et la longueur 30); un cercle ensuite (codé 6) dont le centre est à ... et le rayon est de ...

On devine que l'on va pouvoir de cette manière fournir au système informatique une série d'entiers représentant non plus des points mais des objets de divers types assortis (du codage) des caractéristiques qui leur sont propres.

La représentation du dessin est ici *vectorisée* ou *vectorielle*. Il ne s'agit donc plus d'un nuage de points mais d'une collection d'objets (segments, figures géométriques, courbes,...), chacun étant codé par des nombres qui le caractérisent.

Comme toujours, ce type de codage ne vaut que si des logiciels me permettent d'agir de manière simple sur les graphiques codés de cette manière. C'est bien entendu le cas à travers plusieurs logiciels (qui comportent alors souvent la mention "draw" dans leur nom). On y manipule les objets graphiques présentés à l'écran (et correspondant au décodage des versions vectorisées) à l'aide de périphériques comme la souris. Mais lorsqu'on manipule par exemple un carré, ce dernier n'est évidemment plus vu comme un ensemble de points, mais comme une entité, et il n'est pas possible de modifier la couleur d'un point du carré (puisque ce dernier n'est plus constitué de points).

Plusieurs conséquences doivent être tirées ce cet autre codage :

- D'abord, on comprend aisément la difficulté d'identifier les objets pertinents; *nous* reconnaissons aisément, un rectangle, un carré, un segment, et nous sommes donc capables de fournir à l'ordinateur (si nous connaissons le code à respecter) les entiers correspondants; cela même, signalons-le immédiatement, si ce n'est pas vraiment de cette manière que nous allons fournir à l'ordinateur les objets graphiques souhaités.

Mais on devine immédiatement la difficulté pour un système informatique de reconnaître, de manière autonome (automatique), dans un graphique, des objets pertinents; ainsi, si nous reprenons le premier dessin proposé, quels objets un système informatique pourrait-il bien y déceler, sur base de quels critères, avec quelles caractéristiques ?

?

Voyez-vous cependant une situation où l'on ait fait l'effort de doter l'ordinateur d'un logiciel qui le rende capable de reconnaître au sein d'un "graphique" certains "objets" ?

Il paraît en tout cas évident qu'il faut avoir décrit préalablement au système les objets ou les types d'objets qu'on peut alors lui demander de reconnaître ou de détecter au sein d'un graphique. Il reste alors à le rendre capable d'y détecter les objets (les formes) concernés : c'est tout le problème de la *reconnaissance des formes*.

- Si donc ce type de codage ne permet pas aisément (= automatiquement) de passer d'une feuille de papier portant un graphique à une représentation où les nombres codent des objets et leurs caractéristiques, c'est pourtant lui qui est au coeur des logiciels graphiques de type "draw". Ces logiciels permettent à l'utilisateur, le plus souvent grâce à la souris (ou à un autre dispositif de pointage), de créer des objets (formes géométriques, segments,...) pré définis (au sein du logiciel). Ces objets sont dans ce cas codés d'après leur genre et leurs caractéristiques, selon la méthode envisagée.

### *3.7.5 La reconnaissance optique des caractères*

?

Imaginez que vous disposiez d'une feuille de papier portant le texte suivant :

BONJOUR
COMMENT CA VA ?

Pourriez-vous proposer un système de codage (sous la forme d'une suite de nombres entiers) pour ce "dessin"?

Il est pourtant un cas où une reconnaissance automatique d'objets (et le type de codage vectorisé correspondant) est utilisé : c'est celui de la reconnaissance automatique des caractères. Ainsi, pour une feuille du type de celle portant les quelques mots ci-dessus, le texte va dans un premier temps être scanné et donc la feuille sera codée en une succession de points noirs et de points blancs (codage bitmap) comme ci-dessous.

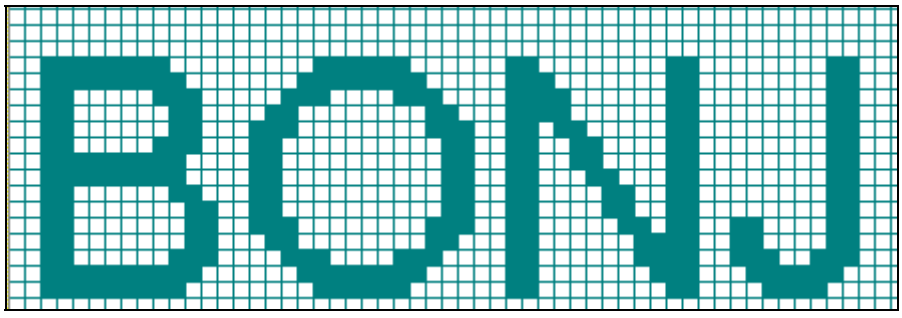


*Figure 3-9 : partie agrandie d'une représentation bitmap de caractères*

Cela fait, un logiciel de reconnaissance de caractères, va détecter dans ce qui n'est alors qu'un nuage de points[12], des caractères. Il le fera bien évidemment en tentant de faire coller certains des nuages de points avec des formes pré définies (que nous lui aurons fournies). Il pourra donc sans problème reconnaître les caractères BONJ dans les nuages montrés ci-dessus. Les caractères étant identifiés, ils seront alors codés à travers un code adapté (par exemple le code ASCII), comme si un être humain les avait frappés au clavier.

Il est bien entendu des cas où les nuages de points n'offrent qu'une ressemblance beaucoup plus dégradée avec les formes connues :

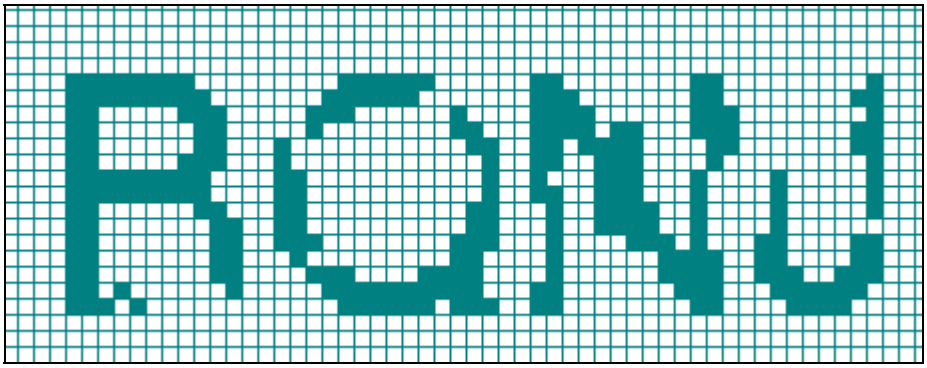


*Figure 3-10 : partie agrandie d'une représentation dégradée*

S'agit-il ici de BONJ, de RONJ, de RQNJ de RONU ? Et l'on devine la difficulté de cette reconnaissance lorsque le texte scanné préalablement (pour donner naissance aux nuages de points) est manuscrit ...

Il est essentiel ici d'avoir perçu le cheminement qui fait passer de signes sur du papier à des nombres qui sont les codes ASCII des caractères. On rend en quelque sorte l'ordinateur capable de *lire*, de reconnaître en des signes tracés sur une feuille des caractères. Le vrai problème n'est pas de transformer les signes en nuages de points (ou en la série des nombres qui représentent ces nuages de points), cela le scanner peut s'en charger. Le problème difficile c'est de faire reconnaître dans ces nuages des configurations qui peuvent s'interpréter comme des caractères. C'est là qu'est toute la difficulté de la reconnaissance optique des caractères (en anglais OCR : Optical Character Recognition).

## 3.8 Codage de sons

?

Pourriez vous faire digérer à votre ordinateur les 5 premières secondes du premier mouvement du concerto n° 23 pour piano de Mozart ?
N'oubliez pas qu'une fois de plus ces 5 secondes de musique devront être transformées en une série finie de nombres entiers

On pourrait sans doute penser à coder la partition musicale correspondante : des nombres pour la succession des notes, des nombres pour leur durée, des nombres pour désigner la clé et les altérations, et cela pour chaque instrument... Comme une partition est finalement constituée d'un nombre fini de symboles distincts, on pourra attribuer à chacun des symboles pertinents un entier... et le codage est terminé.

12 En se rappelant bien que ce nuage de points est en réalité une suite de nombres...

C'est en réalité le son lui même ou plutôt le signal électrique auquel il donne naissance (pour peu qu'on dispose d'un micro) qui va être codé (après une discrétisation analogue à celle opérée pour les images).

Mais il est sans doute bon d'abord de rappeler quels sont les principes qui présidaient, par exemple, à la gravure des (anciens) disques (les 33 ou 45 tours "noirs") et comment sur base d'un tel disque gravé on pouvait reconstituer un son analogue à celui qui avait été "mis en conserve" dans le disque considéré.

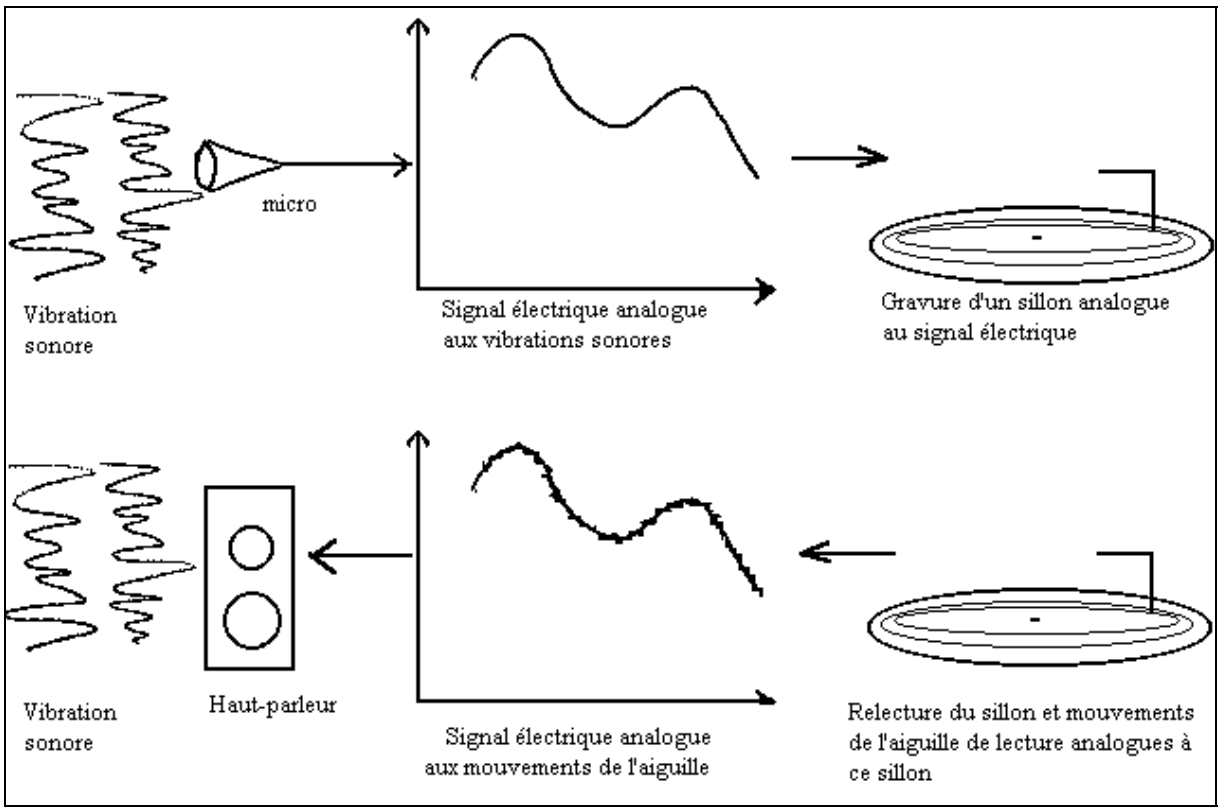



*Figure 3-11 : enregistrement analogique des sons*

1. Les vibrations sonores (ondes sonores) qui ne sont que des mouvements des molécules de l'air, font vibrer la membrane d'un micro.
2. Ces vibrations de la membrane, analogues (semblables) aux vibrations de l'air qui leur ont donné naissance, sont transformées (à l'aide d'un électroaimant) en un signal électrique dont l'amplitude et la fréquence traduisent la force et la hauteur du son qui l'a provoqué. Le micro transforme donc les oscillations acoustiques en oscillations électriques.
3. Le signal électrique provoque alors des mouvements d'un stylet graveur qui traduit les oscillations électriques en mouvements et donc en une gravure plus ou moins profonde d'un sillon dans une matrice qui servira à produire les disques.
4. Chaque disque porte donc un très long sillon en spirale et dont les caractéristiques reflètent finalement les vibrations sonores qui ont été à l'origine de tout le processus.
5. A l'inverse, un électrophone est essentiellement doté d'une aiguille de lecture dont les mouvements, engendré par le sillon du disque, sont transformés en signal électrique.

6. Ce signal électrique est amplifié et met en mouvement la membrane d'un haut-parleur. Les oscillations de cette membrane mettent l'air en mouvement et font naître une vibration acoustique analogue à celle qui avait donné naissance à tout le processus.

Il faut remarquer que dans les tout premiers dispositifs d'enregistrement, le micro était remplacé par un énorme pavillon, ce dernier accolé à une membrane canalisait les oscillations acoustiques et les mouvements de la membrane se transmettaient à un stylet graveur qui lui était solidaire et qui gravait un sillon dans un cylindre de cire en rotation. Il n'y avait donc pas de signal électrique engendré par le processus. Il suffisait inversement de placer le stylet qui jouait alors le rôle de "lecteur" sur le sillon ainsi gravé et de remettre le cylindre en rotation pour que la membrane vibre et engendre des oscillations sonores amplifiées par le pavillon.

Si l'on veut que l'ordinateur puisse digérer les sons, il est indispensable qu'à un moment du processus, il y ait codage sous la forme d'une série finie de nombres. C'est le signal électrique qui va être discrétisé et cette discrétisation donnera naissance aux nombres souhaités.

Le début du processus, consiste à nouveau en la transformation des oscillations acoustiques en un signal électrique.

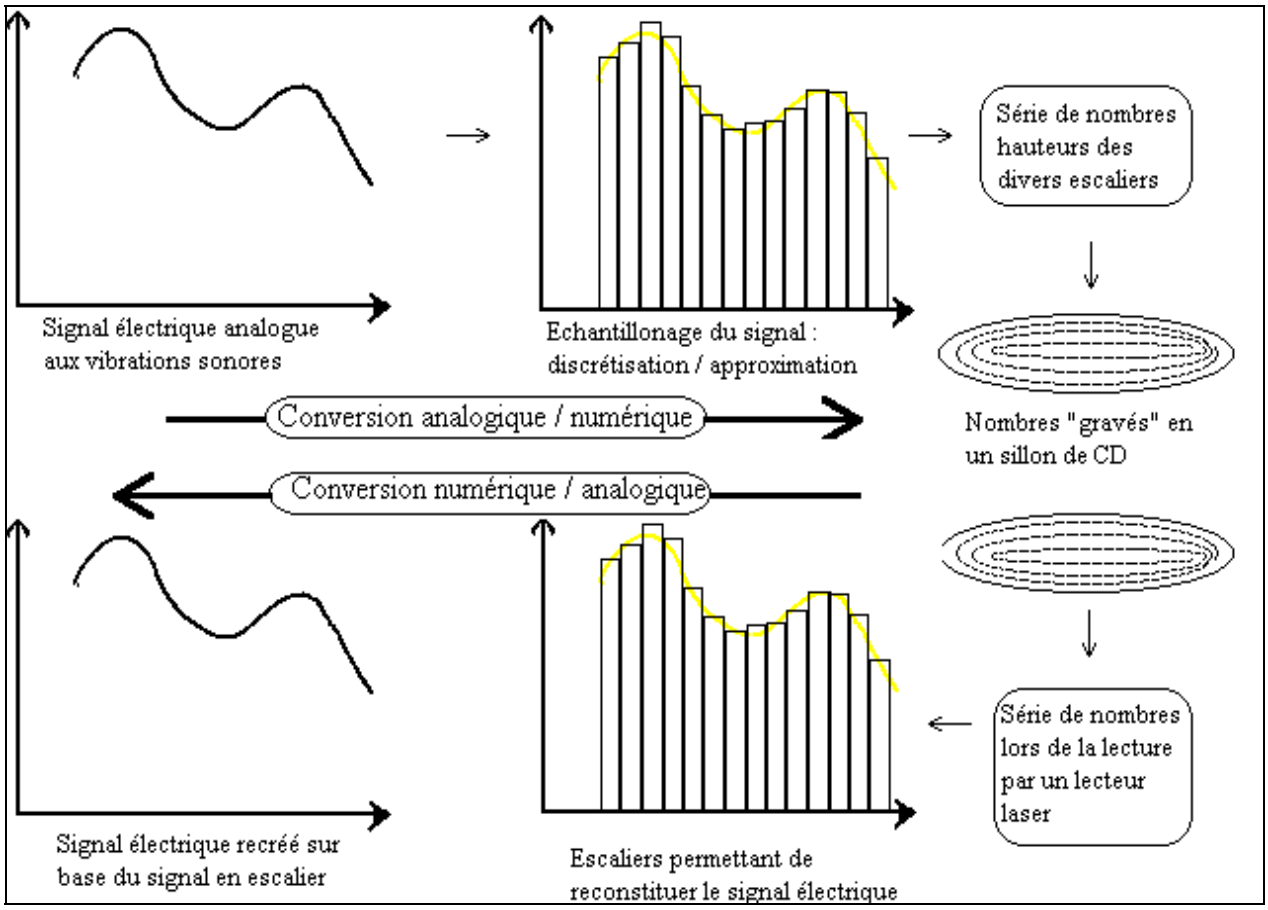



*Figure 3-12 : enregistrement numérique des sons*

1. On va procéder à un *échantillonnage* de ce signal : un très grand nombre de fois par seconde, la valeur du signal va être mesurée; le résultat de chacune de ces mesures est un nombre. Autrement dit, on a remplacé un graphe "continu" par un ensemble fini de valeurs : celles prises par le signal à certains instants. A nouveau, c'est un processus de

discrétisation (de passage du continu et de l'infini au discontinu fini) qui est ici à l'oeuvre : il est le fait d'un dispositif, le *convertisseur analogique-numérique.*

2. Une fois ces énormes séries de nombres ainsi obtenues, on pourrait fort bien les envoyer à un ordinateur. Le schéma ci-dessus évoque cependant la mise en conserve de ces nombres : ils sont "gravés", un par un -et en réalité sous forme binaire et non décimale- sous la forme de minuscules cuvettes successives sur un support qui pourra être relu, le *disque compact*.[13]
3. A l'inverse, un lecteur laser de disques compacts restitue, sur base du long sillon constitué des micro-cuvettes représentant les nombres, ces nombres eux-mêmes (ou plutôt les signaux électriques "en escalier" les représentant).
4. Un *convertisseur numérique-analogique* reconstitue alors au mieux le signal électrique original; il ne reste plus qu'à amplifier ce dernier et à l'envoyer vers un haut parleur pour retrouver les vibrations sonores originales.

Bien évidemment, c'est lorsque le signal sonore a été numérisé (codé sous la forme d'une suite de nombres) que l'ordinateur peut entrer dans la danse. En réalité, il peut être doté d'un périphérique d'entrée constitué d'un micro et d'un convertisseur analogique-digital, ce qui le rend donc capable de capter des sons, devenus séries de nombres.

### *3.8.1 La reconnaissance de la parole*

Lorsque nous avons parlé de la numérisation d'une image, nous avons vu qu'il était possible de faire identifier certains nuages de pixels (ou plutôt les configurations de nombres les codant) comme représentant des caractères. De la même manière, la tentation est grande de faire reconnaître dans les configurations de nombres codant des sons certaines configurations représentant des phonèmes ou des mots français.

La recherche dans ce domaine date de plusieurs années et des logiciels efficaces permettant la reconnaissance vocale commencent à voir le jour. Certains peuvent détecter des mots prononcés de manière isolée et il est possible de donner alors à l'ordinateur des ordres à travers un micro... et tout ce qui vient d'être expliqué. Le vocabulaire reconnu comporte en général un nombre assez restreint de mots reconnaissables.

Certains logiciels sont dotés d'un module d'apprentissage : le futur utilisateur est d'abord prié d'énoncer quelques centaines de phrases et le système "apprend" en quelque sorte à reconnaître "la voix de son maître". Le vocabulaire reconnu peut alors comporter quelques milliers de mots et, surtout, des phrases peuvent être énoncées (presque) en continu, les mots les composant étant -souvent mais pas toujours- reconnus. Même s'il reste bien des progrès à faire, nous aurons ainsi rendu les systèmes informatiques capables de nous écouter... sinon de nous comprendre.

## 3.9 Et les odeurs ?

Je me contenterai ici de reproduire un article paru dans le journal "Le Soir" et daté du samedi 17 septembre 1994.

| Un four qui a du nez |
|---|

[13] Comme on s'en doute, cette présentation simplifiée ne s'attache qu'aux principes mis en oeuvre sans entrer dans les détails "techniques". Il faut être conscient cependant que ce sont ces "détails" qui font la différence entre de belles idées et une technologie efficace.

```
Il est particulièrement difficile , lors de la cuisson dans un
four à micro-ondes de savoir avec précision quand le met est
parfaitement cuit. En raison de la technique très particulière de
ces fours, en effet, tout y est question de seconde. D'où l'idée
des ingénieurs japonais de tenter d'automatiser l'évaluation de
cette cuisson. Un des fabricants nippons, Sharp, estime y être
parvenu en plaçant dans l'un de ses fours un "nez" électronique
qui détecte l'odeur émise par le plat et en confie l'analyse à un
véritable ordinateur doté de ce que l'on appelle une logique
floue. C'est ce système "intelligent" qui est donc chargé de
déterminer le moment exact où la viande est à point et le haricot
encore croquant. A vérifier à l'usage tout de même...
```

La seule conclusion à en tirer, c'est qu'en tout cas que les odeurs sont devenues des nombres.

## 3.10 En guise de résumé

Nous venons de montrer que pour que des information puissent être reçues puis traitées par un ordinateur, il était nécessaire de trouver une représentation de ces informations sous la forme d'une série finie de nombres entiers.

Deux cas peuvent se présenter :

1. Les informations considérées sont en nombre fini (ce qui est assez rare) ou ces informations sont constituées d'un assemblage d'informations élémentaires qui sont, elles, en nombre fini. Ces informations élémentaires constituent en quelque sorte un *alphabet* dont la succession des signes construit les informations considérées. C'est le cas des textes (français) : même si ces textes sont variés et multiples, ils sont obtenus par juxtaposition des symboles d'un alphabet, les caractères.

   Il suffit alors de coder par des entiers ces symboles élémentaires (ce qui est toujours possible puisqu'ils sont en nombre fini). Le codage d'une suite de ces symboles élémentaires se fait alors simplement en juxtaposant les nombres qui codent chacun des symboles constitutifs.

2. S'il n'est pas possible de mettre en évidence aisément un alphabet qui soit tel que les informations considérées puissent être considérées comme formées d'un assemblage fini des éléments de cet alphabet, il est alors nécessaire de passer par un processus de discrétisation qui va approximer les informations en cause.

   On va donc remplacer les informations initiales par d'autres, caractérisées par le fait qu'on peut y mettre en évidence un nombre fini de constituants élémentaires : ce sont ces derniers qui constitueront les symboles de l'alphabet à considérer.

**?** Dans le cas d'une image en noir et blanc, quels sont les constituants de l'alphabet élémentaire dont la juxtaposition permet de décrire l'image discrétisée.

Ainsi, dans le cas d'une image en noir et blanc, après discrétisation les deux seuls symboles carré noir et carré blanc suffisent par juxtaposition pour reconstituer l'image tramée. L'alphabet est donc ici constitué des deux symboles □ et ■; et il suffit de coder ces derniers par des nombres entiers pour être capable de coder l'image discrétisée tout entière.

Ajoutons enfin qu'il est impératif que le processus de discrétisation et de codage soit, autant que possible, pris en charge par le système lui-même et des périphériques spécialisés. Il y a évidemment face à une dictée (orale) un monde entre la solution consistant à faire frapper le texte au clavier par un être humain et la création du même texte par l'ordinateur équipé d'un système de reconnaissance vocale.

## 3.11 Les avantages de la représentation numérique des informations

Il faut d'abord signaler que le processus de discrétisation est à l'origine d'une dégradation des informations disponibles. Passer d'une aquarelle ou d'une gouache tout en nuances à une version discrétisée, même si cette dernière comporte 800 pixels sur 600 et si chaque pixel peut prendre une couleur au choix parmi une centaine, dégrade forcément l'original. On ne dispose plus que d'une version appauvrie, puisque décomposée en un nombre fini de constituants élémentaires. Ceci est cependant indispensable si l'on veut que les informations d'origine puissent être traitées par l'ordinateur, même sous une forme étriquée.

Le premier avantage, lui, est au coeur du présent chapitre : numérique est synonyme de traitable par les ordinateurs.

Il est un second avantage : une fois numérisées, les informations sont transportables, transmissibles sans dégradation irréparables causée par ce transport.

Si par exemple, je dois transporter à distance un signal comme celui représenté ci-dessous

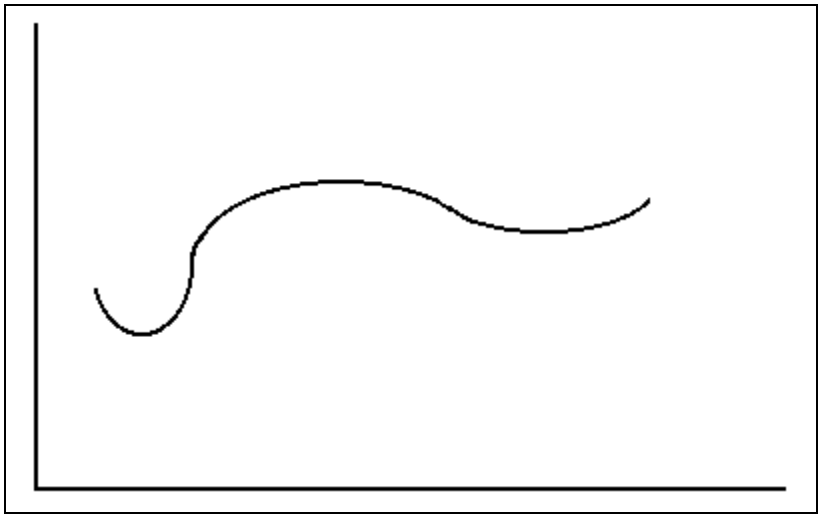


*Figure 3-13 : signal original*

il est évident que de légères déformations vont apparaître dans le signal obtenu, par rapport au signal initial (figuré par les différences présentées ci-contre entre le signal initial -en grisé- et le signal final).

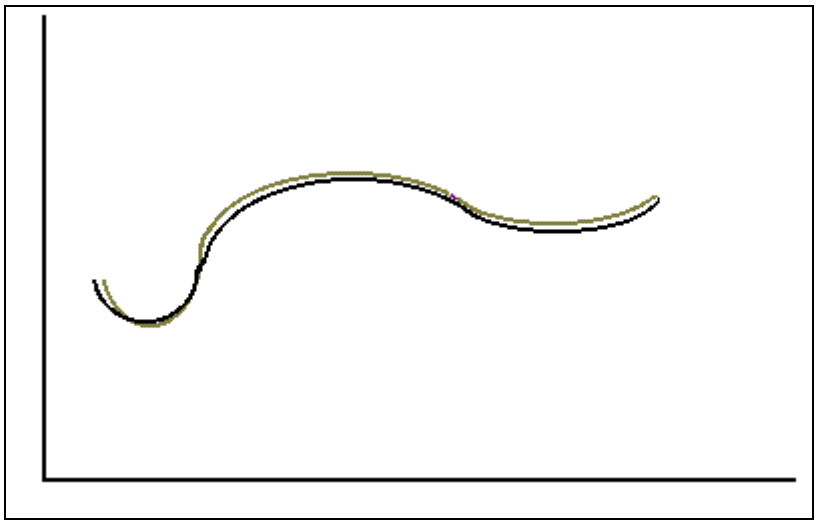


*Figure 3-14 : signal dégradé*

Lorsque le signal représente des nombres différents, il est forcément initialement du type suivant :

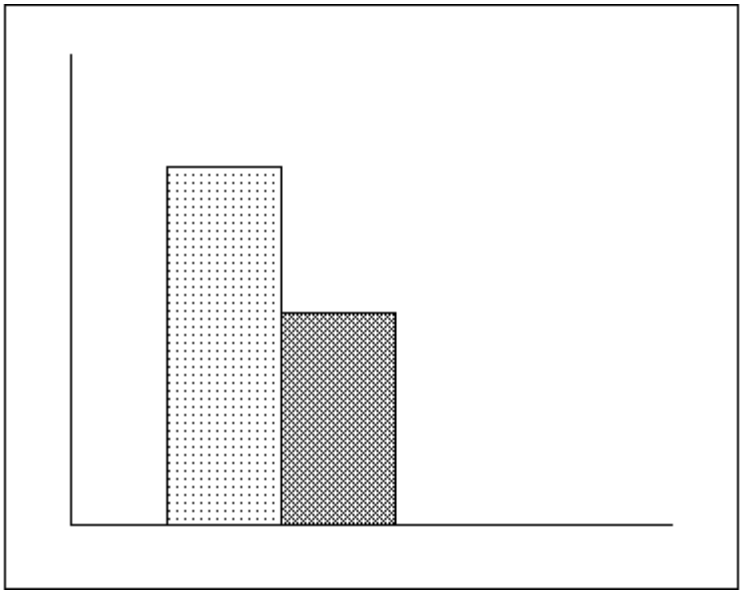


*Figure 3-15 : signal numérisé*

le signal résultant d'un transport ou d'un traitement étant également dégradé :

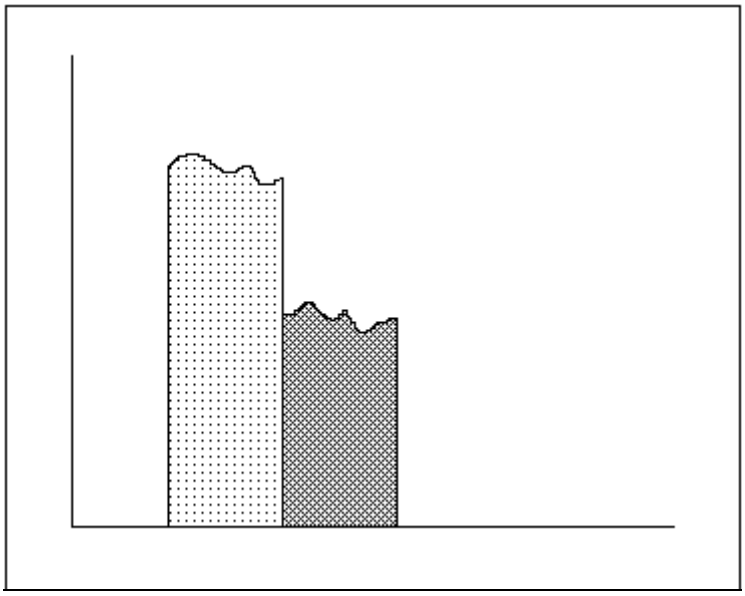


*Figure 3-16 : signal numérisé dégradé*

On remarquera qu'il est cependant possible dans ce cas de reconstituer les nombres initialement représentés. Autrement dit, même si le signal représentant 16 arrive dégradé, il est toujours possible de détecter qu'il s'agit bien de 16 et non de 15 ou 17. Cette permanence des représentations numériques sera d'ailleurs d'autant plus marquée que finalement, les nombres eux-mêmes seront codés avec les deux seuls symboles 0 et 1.

## 3.12 Des médias multiples au multimédia

Nous sommes à présent en mesure, à la lumière de ce que nous savons à propos du codage des informations en séries de nombres, de comprendre pourquoi la multitude des médias inventés par l'homme pour "capturer" des portions variées du "réel" cèdent aujourd'hui la place au "multimédia".

### *3.12.1 Le multi-média de grand-papa*

A travers ses divers sens, l'homme perçoit et appréhende le monde qui l'entoure. Lorsqu'il s'agit de "représenter" ou de "conserver" des portions du réel porteuses d'information, l'homme va créer des objets divers qui stockeront sur des supports variés ce qu'il veut saisir, retenir ou transmettre du réel.

*3.12.1.1 Le multi-média "à l'ancienne" : de multiples médias pour capturer le réel*

Les exemples en sont évidemment nombreux; ils ont tous en commun le choix de supports particuliers sur lesquels une représentation des traits considérés comme pertinents du réel (au sens large) va être "enregistrée".

Ainsi en va-t-il du média qui nous est sans doute le plus habituel : l'écrit. Le supports en sont variés depuis les hiéroglyphes gravés sur les temples égyptiens, jusqu'au livre imprimé, en passant par les papyrus et les parchemins.

Même dans le cas de l'écrit "crayon papier" les technologies mises en œuvre sont diverses et variées : crayon, stylo et encre, machine à écrire, imprimerie,... Les modes de saisie qui en découlent nécessitent donc des moyens techniques plus ou moins sophistiqués, du simple crayon à la rotative...

On a donc, en ce qui concerne l'écriture, des schémas :

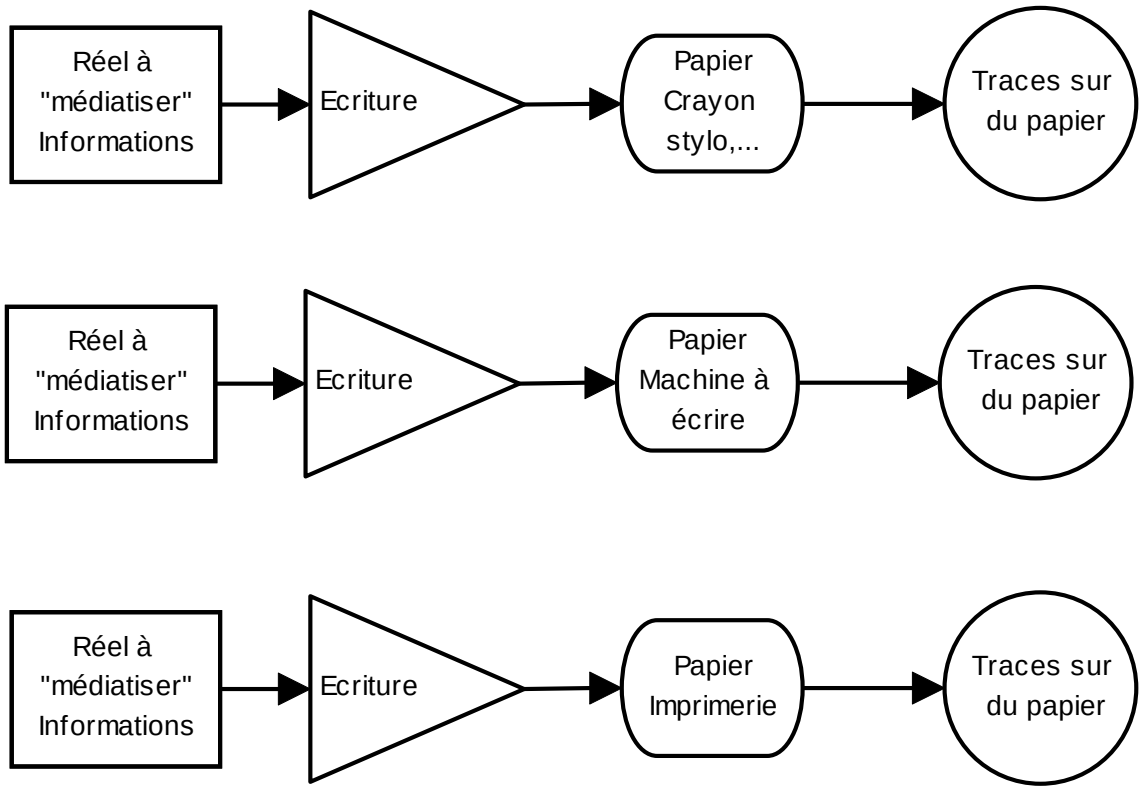



*Figure 3-17 : divers procédés de saisie dans le cas de l'écriture*

De la même manière, pour s'intéresser à des informations d'une autre nature, il y a une multitude de moyens de **saisir** une "image" du réel : peinture, dessin (y compris avec un support papier-crayon du même type que pour l'écriture), photographie, etc. On a par exemple :

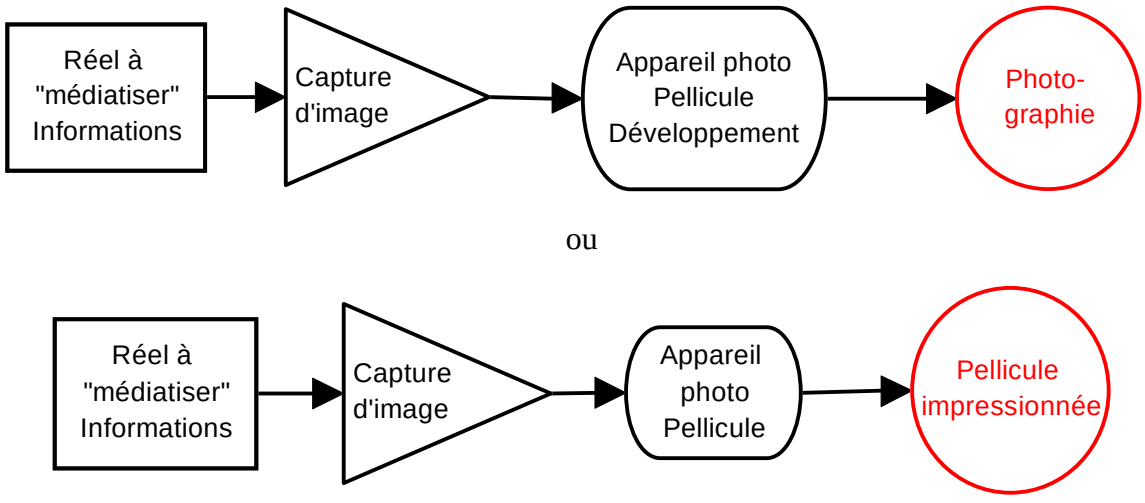



*Figure 3-18 : divers procédés de saisie dans le cas de la photographie*

On pourrait poursuivre avec les dispositifs permettant de saisir et d'emmagasiner le son, le mouvement (images animées),... On est en tout cas en mesure de mettre en évidence un schéma se déclinant en une multitude de cas particuliers :

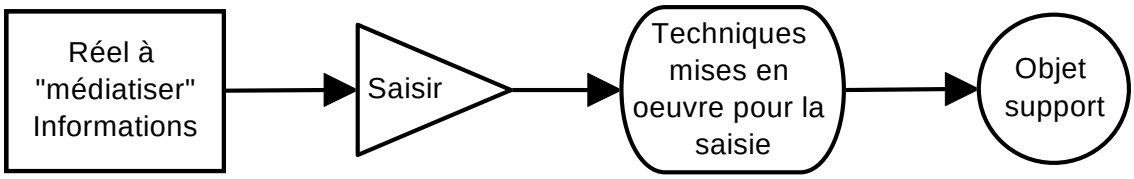



*Figure 3-19 : la saisie d'informations, en général*

Ces représentations "médiatisées" du réel sont là pour pouvoir être travaillées, modifiées, stockées, transportées, communiquées,...

#### *3.12.1.2 Le multi-média "à l'ancienne" : il faut restituer ce qui a été capturé du réel*

Quel que soit le support "médiatique" retenu et la technologie mise en jeu, il est nécessaire, pour que nous percevions à travers lui le réel qu'il représente, de mettre en œuvre une restitution nécessitant (comme c'est aussi le cas de la saisie) des moyens plus ou moins sophistiqués.

On pourrait sans doute classer les médias support d'informations eu égard à la complexité des techniques permettant de passer de l'information sous la forme où elle est stockée à l'information telle que nos sens la perçoivent (**restituer**, dans le schéma ci-dessous).

Entre le livre, conservant texte et images sous une forme directement accessible et le CD-ROM proposant texte, images, sons, séquences animées,... mais nécessitant lecteur, ordinateur, écran pour nous être rendus accessibles, il y a évidemment un monde.

En matière de conservation de l'information, le papier et la plume ou l'imprimerie permettant de conserver et de communiquer l'écrit (texte, image) constituent probablement le média à la fois le plus habituel et celui qui a le plus modelé notre culture et notre société. Notons au passage que c'est l'un de ceux où le passage du support de l'information au message restitué et perçu nécessite le moins d'intermédiaires : face à un livre, la sophistication des technologies nécessaire ne va guère au-delà d'une bougie et, le cas échéant, d'une bonne paire de lunettes.

Il y a d'autres supports où la révélation de l'information conservée par le support est plus longue ou plus délicate, comme la pellicule photographique qui demande un certain traitement afin de déboucher sur une image qui nous soit directement perceptible.

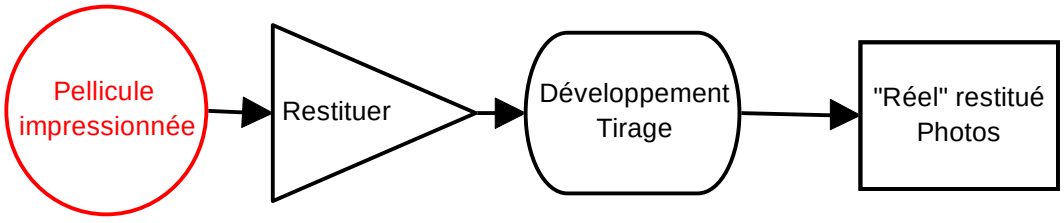



*Figure 3-20 : la restitution dans le cas de la photographie*

La diapositive est également un support d'image qui nécessite pour pouvoir livrer correctement l'information qu'elle recèle un dispositif technique (projecteur) d'une certaine complexité.

Et nous voici aujourd'hui avec les photos mystérieusement stockées au sein même des appareils "numériques" qui ont permis de les prendre ou sur mini-disques magnétiques.

En ce qui concerne le son et les dispositifs permettant de le conserver, on est passé du cylindre de cire, gravé et restitué par le même "phonographe", au disque microsillon, à la

bande magnétique, qui demandent des dispositifs de plus en plus sophistiqués pour passer du son "en conserve" à sa "consommation" par les oreilles humaines; avec en finale aujourd'hui les CD et leurs "lecteurs laser".

Quel que soit le média envisagé, l'opération de restitution des informations "enfermées" dans l'objet support est indispensable :

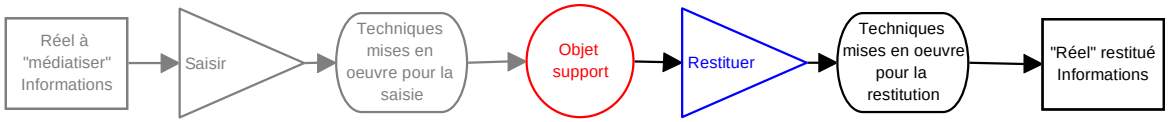



*Figure 3-21 : la restitution en général*

Le cinéma est, pour des raisons qui apparaîtront plus loin, intéressant puisqu'il a conservé les mouvements sous la forme d'une succession d'images instantanées. Puis, sur le même support sont venus prendre place à la fois les images animées et le son, le dispositif de projection/reproduction devenant plus sophistiqué.

### *3.12.1.3 Le multi-média "à l'ancienne" : peu de possibilités de modifier les objets supports*

On peut, pour chacun des supports envisagés, considérer quelles sont les possibilités de modification de cet objet support dont on dispose. Ces moyens sont extrêmement variables et mettent en branle des technologies plus ou moins sophistiquées, depuis la gomme et le crayon pour les supports écrits "papier-crayon" jusqu'aux retouches des pellicules photographiques.

On a donc à nouveau, quel que soit le média envisagé, une multitude de schémas permettant de rendre compte des possibilités de **modifier** la représentation du réel mise en œuvre.

Ainsi, pour le média "papier-crayon" :

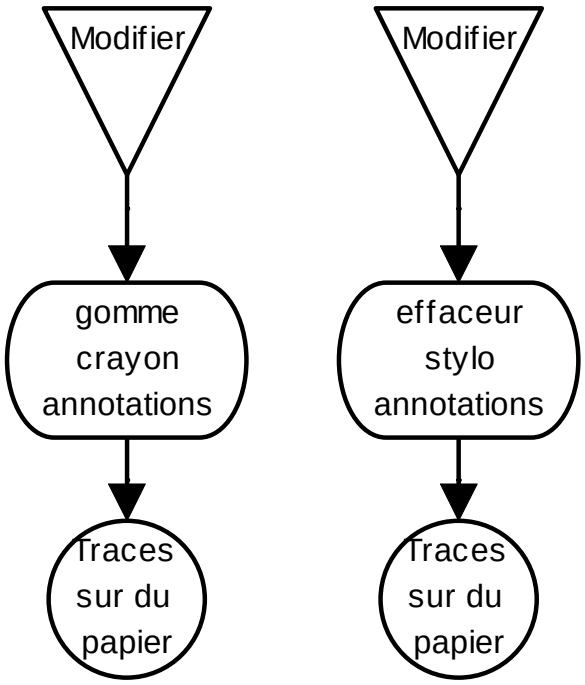



*Figure 3-22 : la modification des supports écrits*

Il faut admettre cependant que **les possibilités de modification sont le plus souvent extrêmement limitées** ou qu'elles nécessitent des appareils extrêmement sophistiqués auxquels la plupart d'entre nous n'ont pas accès. Qu'on pense par exemple aux possibilités nulles ou fort réduites qu'on a de modifier un disque 33 tours, une pellicule photographique, etc..

Mais, même face à des possibilités limitées, le schéma commencé s'enrichit :

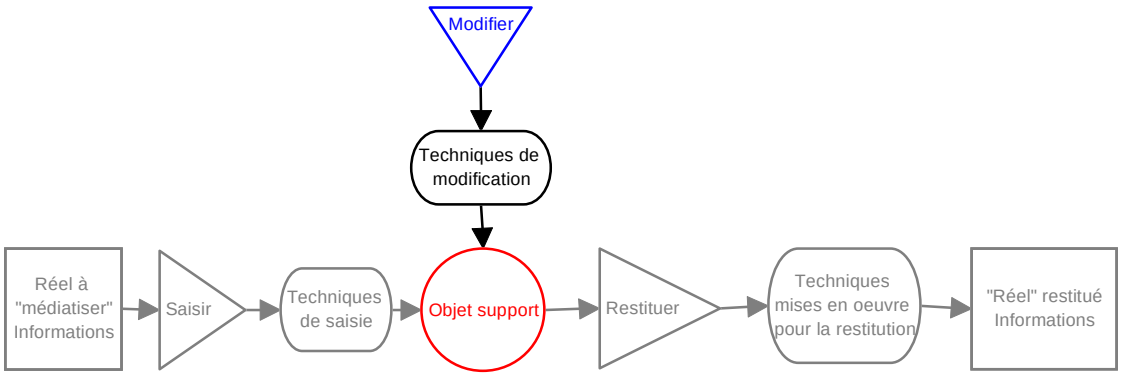



*Figure 3-23 : la modification des objets supports*

### *3.12.1.4 Le multi-média "à l'ancienne" : stocker les objets supports*

Un des problèmes majeurs de ces divers médias, est de **stocker** et conserver les "objets supports". Les bibliothèques sont une réponse au problème du stockage, dans le cas de l'écrit :

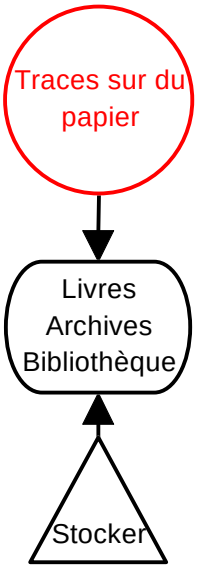



*Figure 3-24 : le stockage des objets supports de l'écrit*

Les discothèques et autre cinémathèques sont d'autres réponses au problème du stockage d'autres objets supports relevant d'autres médias.

Le schéma commencé se poursuit donc :

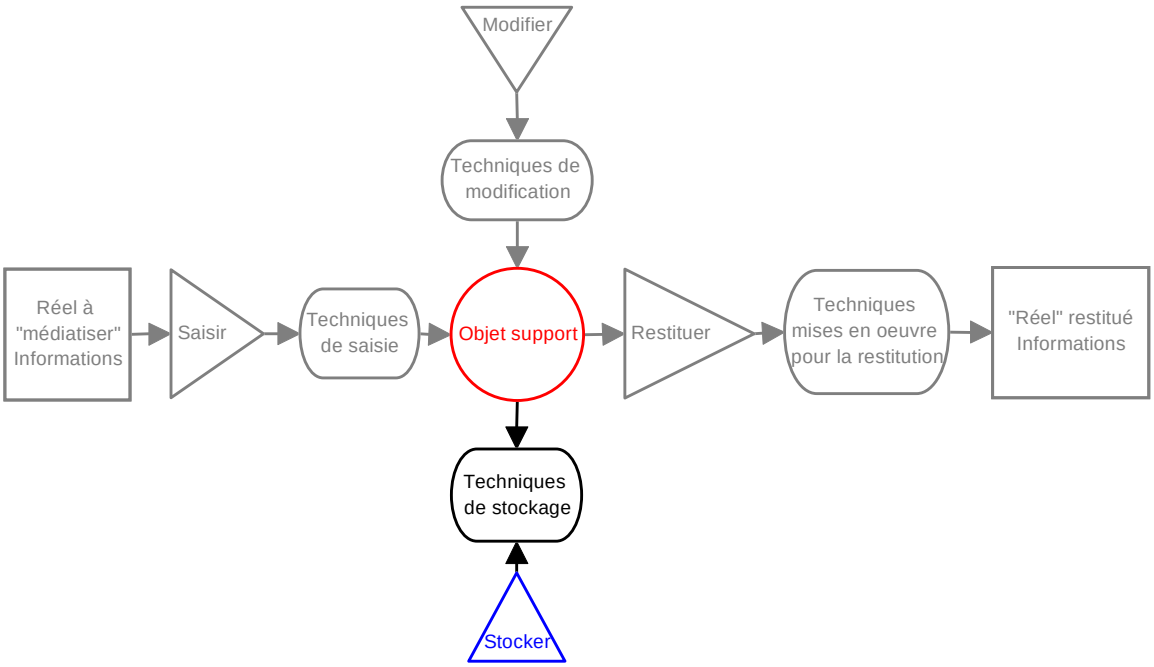



*Figure 3-25 : le stockage des objets supports en général*

On voit donc apparaître le problème du stockage comme une opération à part entière. C'est essentiellement pour montrer que la permanence de l'objet support est un facteur important et décisif. Écrire ou dessiner sur du sable est évidemment une manière de d'utiliser comme "objet support", mais dans ce cas, le stockage à long terme risque de poser quelques problèmes et je ne suis pas certain qu'on puisse vraiment parler dans ce cas de "média".

### *3.12.1.5 Le multi-média "à l'ancienne" : communiquer les objets supports*

Cette communication se réduit le plus souvent à un transport pur et simple (c'est à cela que servait et que sert encore en grande partie la poste)

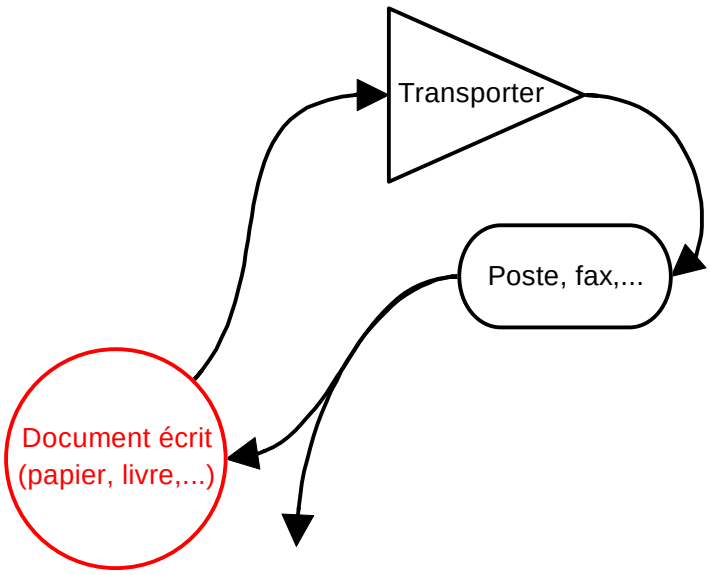



*Figure 3-26 : le transports des objets supports dans le cas de l'écrit*

Parfois le mode de communication est plus sophistiqué : c'est le cas de la radio, de la télévision, etc.

Dans tous les cas, et quel que soit le média envisagé le problème de la communication du support fait partie intégrante des préoccupations :

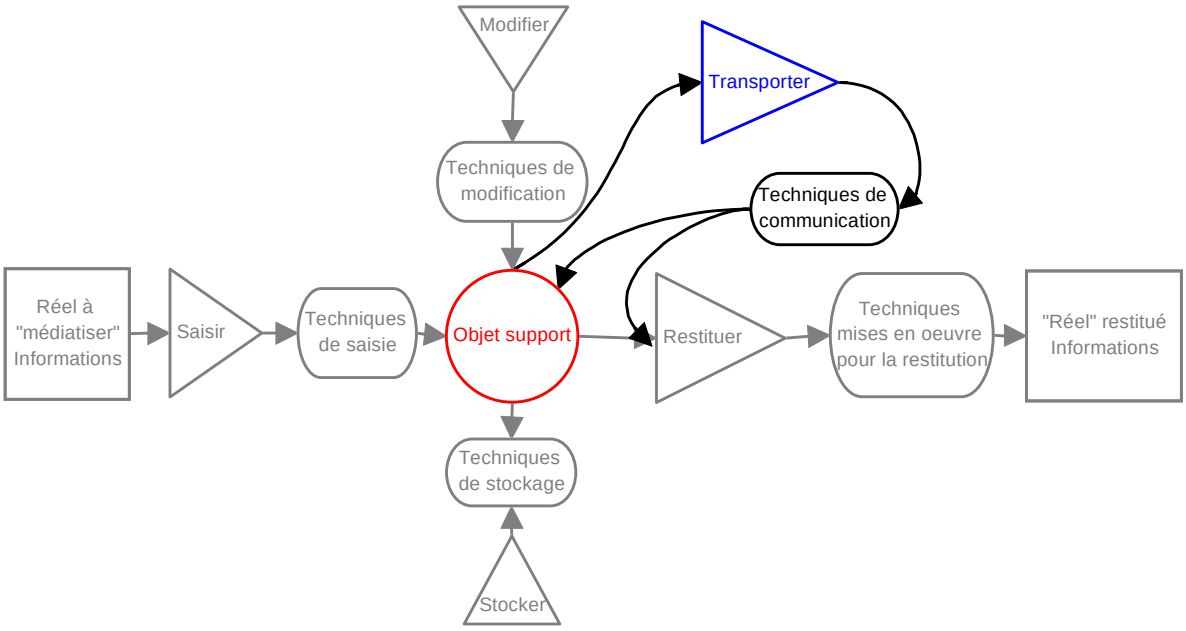



*Figure 3-27 : le transports des objets supports en général*

Ce qui est donc au cœur de l'utilisation des divers médias (au sens que je leur donne ici) c'est finalement un schéma qui peut s'achever de la manière suivante :

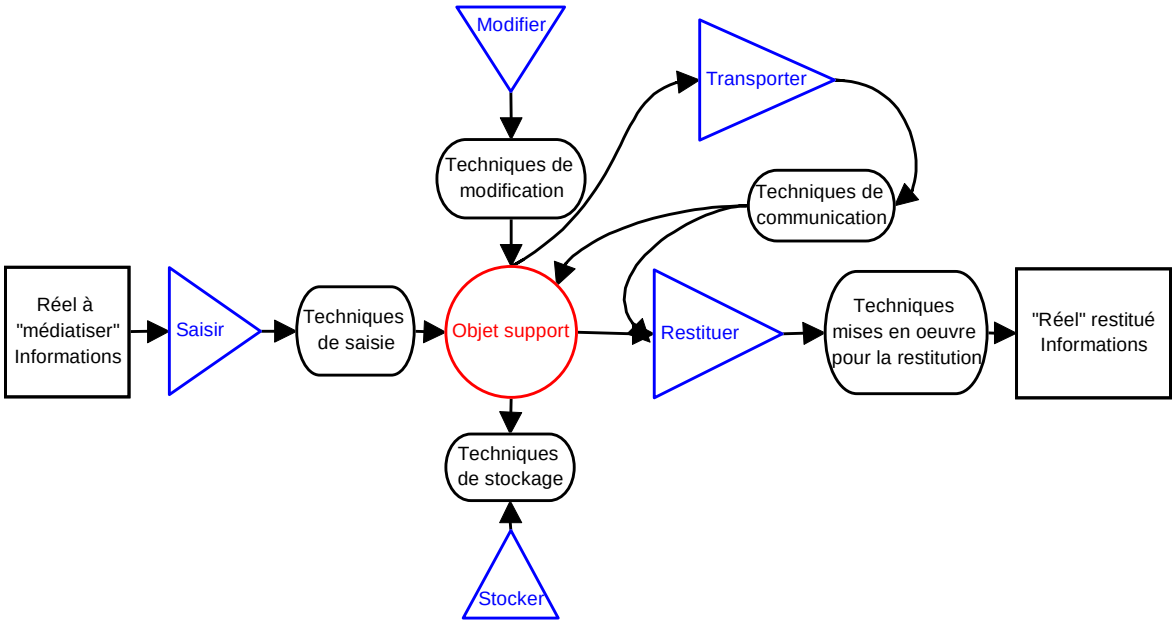



*Figure 3-28 : le schéma global de la médiatisation*

### *3.12.2 Du multi-média au multimédia*

Comprendre l'apport de la numérisation et le rôle des ordinateurs, c'est saisir qu'on passe d'une multitude de médias à un support unique qui est en train de rendre obsolètes toutes les technologies classiques de médiatisation. C'est aussi percevoir que l'opération "modifier", inexistante ou hors de portée de l'utilisateur des multiples médias classiques, va à

présent prendre une importance capitale : c'est elle qui se cache derrière la plupart des logiciels permettant par exemple la modification d'un texte ou la retouche d'images.

*3.12.2.1 Les moyens "multi-média" : une juxtaposition de médias et de technologies dissemblables*

Voici la définition du multimédia (que j'écris ici multi-média) lors de l'apparition de ce mot :

> "Multimédia adj. - 1980; de multi- et média. Qui concerne plusieurs médias; qui est diffusé par plusieurs médias. *Campagne publicitaire multimédia*." [Le nouveau Petit Robert, 1993]

Le schéma présenté ci dessus va se décliner de multiples manières, d'après les "traits" du réel (la nature des informations) qu'on veut garder, d'après les dispositifs mis en œuvre pour saisir, stocker, modifier, éventuellement transporter, et restituer les informations considérées comme pertinentes.

L'écriture (au sens classique) sur divers supports recouvre toute une classe de modes de saisie, dépendant d'ailleurs du support retenu (burin + pierre, crayon + papier, imprimerie + papier,...).

La saisie d'images couvre également un énorme champ allant des divers types de peintures au scanner des ordinateurs en passant par la photographie, avec une mention spéciale pour les diapositives.

Les informations sonores ont mis plus longtemps à pouvoir être "médiatisées" : le solfège était une forme de représentation de cette réalité sonore particulière qu'est la musique, sur le modèle de l'écriture; puis sont venus les phonographes et le stockage du son dans la cire des rouleaux, puis les disques microsillons, les bandes magnétiques et enfin les CD et leurs lecteurs lasers.

Je ne m'attarderai pas sur le cinéma, la vidéo,... sur les manières de transporter les informations représentées (télégraphe, téléphone, télévision, courrier électronique).

Je n'ai guère évoqué non plus les possibilités et les modalités de modification des informations conservées dans l'objet support (depuis la gomme jusqu'au logiciel de retouche d'images, en passant par le mixage des studios d'enregistrement).

Le schéma ci-dessus doit être recommencé des dizaines de fois : à chaque fois qu'un mode de stockage, de saisie, de restitution,... change, on est quitte pour en adapter et en préciser les éléments.

Bref, le multi-média, mise en œuvre de procédés divers permettant sur des supports différents de stocker des informations de toutes natures à l'aide de **technologies absolument dissemblables**, on connaît et depuis fort longtemps ! Mais, jusqu'à présent, une présentation multi-média devait faire appel à des appareils aussi différents qu'un tableau noir, un rétroprojecteur et ses transparents, un projecteur de dias, un magnétoscope et le poste de télévision qui l'accompagne, etc..

*3.12.2.2 Les moyens multi-médias revisités par l'informatique : le multimédia*

Et puis vint l'ordinateur... qui ne changea rien à tout cela pendant presque 50 ans. Les médias restèrent "multi" peut être, mais séparés sûrement.

C'est le MacIntosh de Apple qui montra la voie : assez rapidement, il devint l'outil qui se répandit dans les secrétariats pour la production de texte et l'instrument de choix des graphistes pour le traitement de l'image.

Depuis le milieu des années 90, les "PC" sont eux aussi devenus "multimédia". Les ordinateurs, qu'on appelait encore il y a peu des "calculateurs électroniques", seraient-ils devenus "bons à tout faire", de la création de texte à la retouche de photos en passant par le travail du son.

Comme on le verra, l'ordinateur reste un **calculateur**, manipulant des nombres. Et pourtant c'est aussi un instrument permettant aujourd'hui de stocker et de traiter à notre demande son, texte, image, vidéo,... Où est le miracle ?

Il n'y a pas de miracle : si l'on veut qu'un outil à manipuler des nombres soit capable de traiter toutes sortes d'informations de nature et de formes diverses, il suffit tout bêtement de transformer ces informations en série de nombres.

Que croyez-vous que l'ordinateur reçoive lorsque vous pressez la touche "a" sur le clavier ? Qu'est ce qui "remplit" les CD-ROM (mais aussi les CD musicaux) et les disques durs ? Qu'est ce que l'ordinateur envoie à votre imprimante ? De quoi sont faits les documents que vous recevez "sur Internet" ?... La réponse est identique à chaque fois : il s'agit de nombres (ou en tout cas d'une représentation physique de nombres).

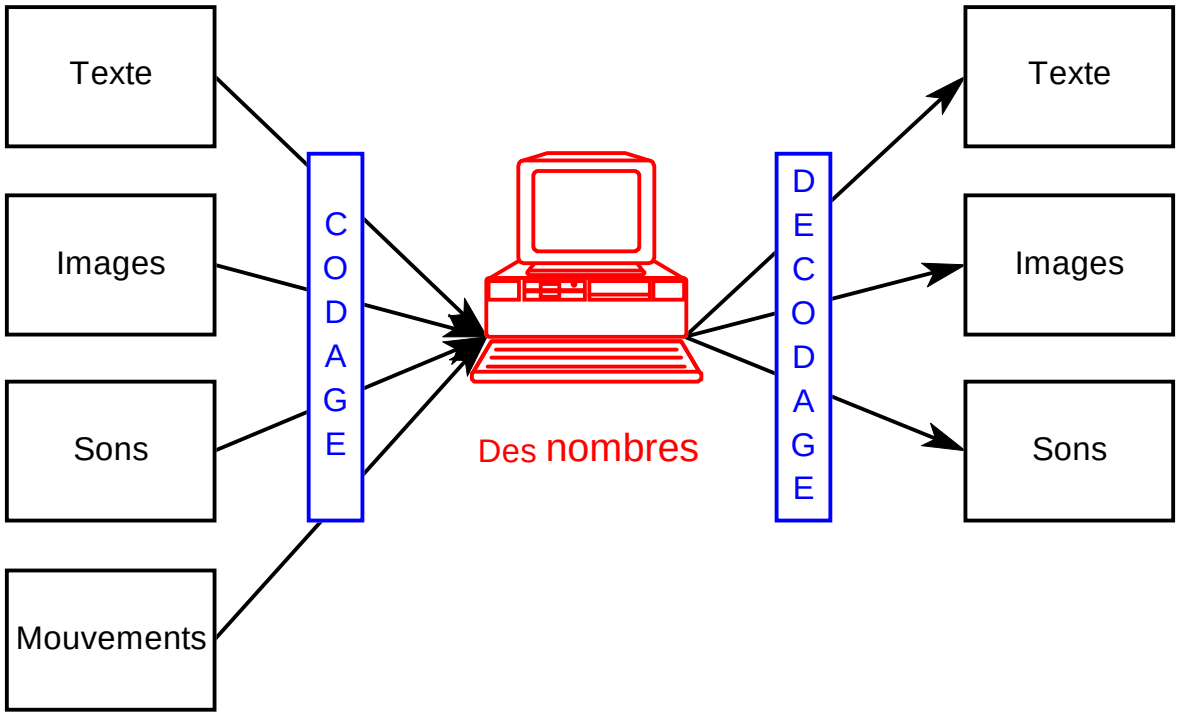



*Figure 3-29 : l'ordinateur multimédia*

A partir du moment où texte, images (fixes ou animées), sons, etc. peuvent être transformés en séries de nombres (on dit le plus souvent **numérisés** ou digitalisés), ils peuvent être manipulés, modifiés, stockés et transportés par un dispositif unique, apte à gérer seulement des nombres, l'ordinateur.

L'apport se marque de manière claire au sein d'une définition récente ;

> "Multimédia n m. - Technologie intégrant sur un support électronique des données multiples (son, texte, images fixes ou animées). Adj : *des encyclopédies multimédias*" [Le Petit Robert, 1998]"

Il reste seulement à imaginer des manières de représenter (on dit aussi coder) textes, images et sons sous la forme de séries de nombres et de mettre au point des dispositifs qui assurent aussi automatiquement que possible ce codage :c'était l'objet de ce chapitre.

Bien évidemment, il faut également des dispositifs de restitution qui assurent le décodage, pour que les nombres manipulés redonnent naissance à des informations qui nous soient accessibles : texte, image, vidéo, son,...

Mais ce qui a radicalement changé, c'est que c'est un outil unique, l'ordinateur (et tous les périphériques qui l'entourent) qui va servir à saisir, modifier, stocker, transporter, restituer, les informations, quelle que soit la forme où elles se présentent à nous. Plutôt que de multimédia, c'est d'un monomédia universel qu'il faudrait parler. L'ordinateur va pouvoir à lui seul se substituer à toutes les techniques dont l'Humanité s'était dotée au fil des siècles pour "médiatiser" le réel. Le maître mot ici est celui de **numérisation** (codage de l'information en une suite de nombres).

> Parler de multimédia, c'est simplement reconnaître que toute information, quelle que soit sa forme, pourra, après numérisation, être traitée par un outil unique, l'ordinateur, puis être restituée sous une forme habituelle et acceptable.

*Principe 3-1 : c'est la numérisation qui est au coeur du multimédia*

Dans ces conditions, c'est vrai que l'ordinateur devient un outil universel pour traiter du texte, des images, des sons,... Et plutôt que de devoir recommencer des dizaines de fois la Figure 3-28, on pourra dorénavant se contenter d'un schéma unique :

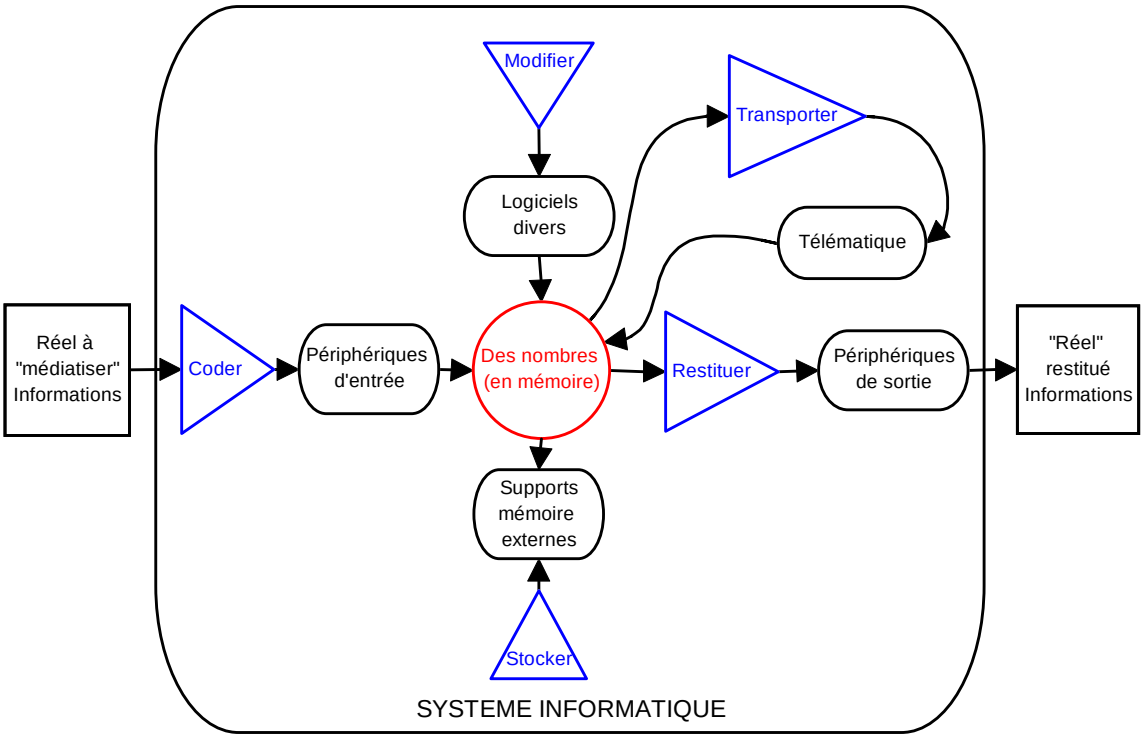



*Figure 3-30 : schéma global de la médiatisation dans le cas d'un système informatique*

Si les TIC vous font peur, adoptez les TeTIN!

Maîtriser ces univers nouveaux qui se sont ouverts autour de l'utilisation des ordinateurs, c'est aussi comprendre les mots qui en parlent et l'évolution des conceptions que reflètent les termes employés. Ainsi, pendant plus de 40 ans, on a tout bonnement parlé d'informatique

pour désigner l'activité de ceux qui tentaient de tirer parti des ordinateurs pour résoudre des problèmes de natures diverses, mais qui tous tournaient autour du "traitement automatique de l'information".

Vers le milieu des années 80, sont apparus, en même temps que les PC commençaient à envahir les bureaux, les outils logiciels qui devaient permettre à tout le monde, et plus seulement aux informaticiens, de tirer parti des ordinateurs qui étaient devenus "personnels". C'était le début de la seconde étape de la courte histoire de l'informatique : après l'informatique réservée aux informaticiens, on débouchait sur une multitude d'outils logiciels à la portée des utilisateurs "naïfs". On a alors commencé à parler de NTI (Nouvelles Technologies de l'Information) pour désigner, de manière assez floue d'ailleurs, ce mélange de technique, de concepts issus de l'informatique, d'usages, etc.

Pendant que les réseaux se répandaient, permettant aux ordinateurs de communiquer entre eux, et alors que les "nouvelles" technologies commençaient à prendre de l'âge, on a à nouveau changé de vocable pour désigner cette nouvelle réalité : ce sont aujourd'hui des TIC (Technologies de l'Information et de la Communication) qu'on parle le plus souvent.

Et ce que nous apprend la numérisation qui permet et sous-tend le phénomène multimédia, c'est que tous les traitements permis par le multimédia portent sur l'information numérisée et que dès lors on se trouve face à des TeTIN (Technologies de Traitement de l'Information Numérisée).

## 3.13 Vers une civilisation du numérique

Les codages numériques dont nous venons de parler dans le cas de l'obligation de faire digérer des informations de diverses natures par l'ordinateur, constituent un fait majeur de notre culture et de notre société : nous passons d'une civilisation de *l'analogique* à une civilisation du *numérique*.

Dans la présentation même des informations, on notera le passage de

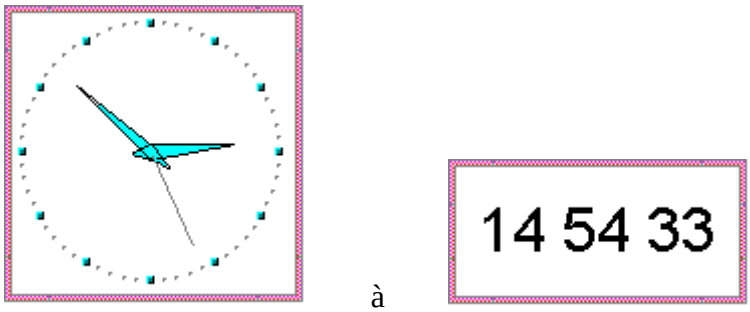



*Figure 3-31 : représentations analogique et numérique*

La mesure même du temps qui passe (des durées) a été analogique : le fuite du temps était *analogue* au passage du sable dans un sablier ou à celui de l'eau dans une clepsydre. Il se fait aujourd'hui en *comptant* les oscillations d'un cristal.

J'ai évoqué ci-dessus le passage des vieux disques 33 tours vinyles (où le tracé du sillon était *analogue* aux oscillations sonores qui leur avaient donné naissance) au disque compact (où c'est une représentation de *nombres* que l'on peut trouver le long du sillon).

Dans le même ordre d'idée, on parle aujourd'hui de *téléphone numérique* ou de *télévision numérique* : dans les deux cas il s'agit de transformer son ou image en nombres, de transporter ces derniers (sans dégradation irréparable), et de reconstituer à l'arrivée le son ou l'image original.

L'ordinateur est évidemment à la fois l'une des causes de cette civilisation du numérique, mais aussi un partenaire incontournable pour traiter (au sens très large) ces informations numérisées.

## 3.14 Questions

1. Un ami, utilisateur d'ordinateur, me dit la phrase suivante : "Voici la disquette avec le texte de la lettre que je t'ai promis. Mais méfie-toi parce que ce texte était sur une autre disquette. Je l'ai d'abord copié sur une disquette d'un format différent, puis sur le disque dur, puis sur la disquette que je te donne. J'ai un peu peur qu'après toutes ces copies successives le texte en question ne soit un peu abîmé !" Quel commentaire apporteriez-vous à cette assertion ?
2. On appelle graphe un ensemble de noeuds ou de sommet liés entre eux par des arcs et on en donne en général une représentation graphique dont voici un exemple :

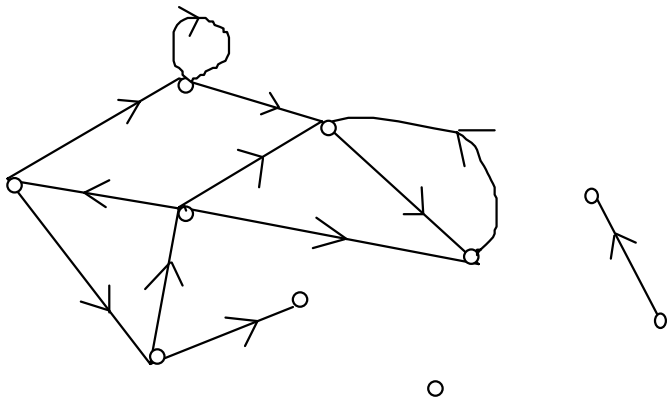


*Figure 3-32 : graphe*

Pourriez-vous proposer trois systèmes différents de codage sous forme de listes structurées de nombres (ou de lettres, ce qui revient finalement au même) et qui permettent de représenter un graphe ?

Pour chacun de ces systèmes, donnez le codage du graphe représenté ci-dessus.

Pour chacun des systèmes de codage retenu, pourriez vous donner "en français courant" les indications de traitement nécessaire (= les manipulations de nombres ou de lettres à effectuer) pour déterminer

- la liste des sommets successeurs immédiats d'un sommet donné;
- la liste de tous les couples de sommets liés par deux arcs de sens opposés;
- la liste des sommets isolés.

Et, enfin, êtes-vous capable de dire pourquoi je vous pose ce genre de question ?

3. Pourriez vous trouver des *informations* qui se présentent sous plusieurs *formes* différentes, mais en dessous desquelles nous identifions pourtant la *même information* ?
4. Existe-t-il à votre connaissance une manière de représenter les sons du langage parlé sous forme écrite, donc de coder des sons par des signes écrits ?
5. Que proposeriez vous comme codage numérique d'une configuration quelconque du jeu d'échec ? Pourriez-vous en proposer plusieurs ? Et pour le jeu de dame.

6. Que proposeriez vous comme codage numérique des différentes mains (constituées de 13 cartes) d'une donne au whist ?

# 4

# ... pour autant qu'on lui ait indiqué comment mener à bien ce traitement ...

## L'ordinateur n'existe pas

### 4.1 Introduction

Nous savons à présent que les traitements permis par l'ordinateur auront toujours un caractère formel (ou formaliste) et que dès lors le mot "information" y prendra un sens très particulier.

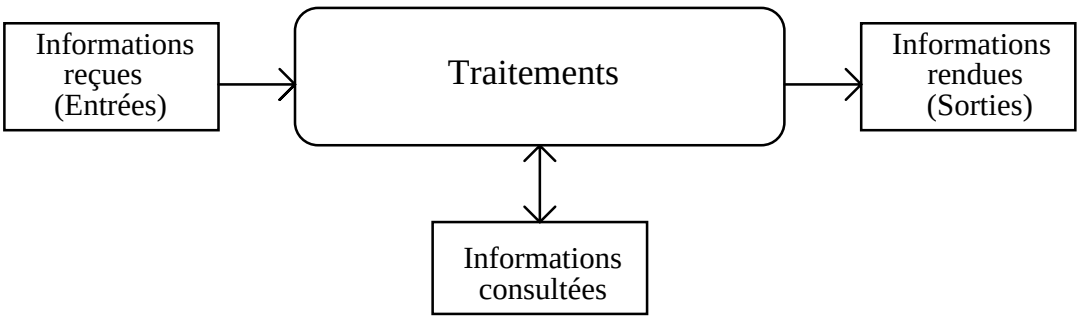
La pseudo-définition donnée précédemment, insiste sur la nécessité d'indications de traitements qui permettent et contrôlent le travail de traitement d'informations par l'ordinateur. Nous pouvons dès lors énoncer un principe supplémentaire :

**TOUT** ce que fait un ordinateur, il le fait gouverné par un programme

*Principe 4-1 : tout ce que fait un ordinateur, il le fait gouverné par un programme*

Un *programme*, c'est donc l'ensemble des indications de traitements nécessaires pour que l'ordinateur puisse mener à bien telle ou telle tâche de traitement d'information. Dès qu'un ordinateur agit, c'est qu'il dispose du programme nécessaire pour le faire agir. On dit aussi parfois *logiciel* plutôt que programme, surtout lorsque ce dernier est d'une taille ou d'une complexité importantes.

Dès lors le schéma de traitement proposé plus haut




peut se redessiner :

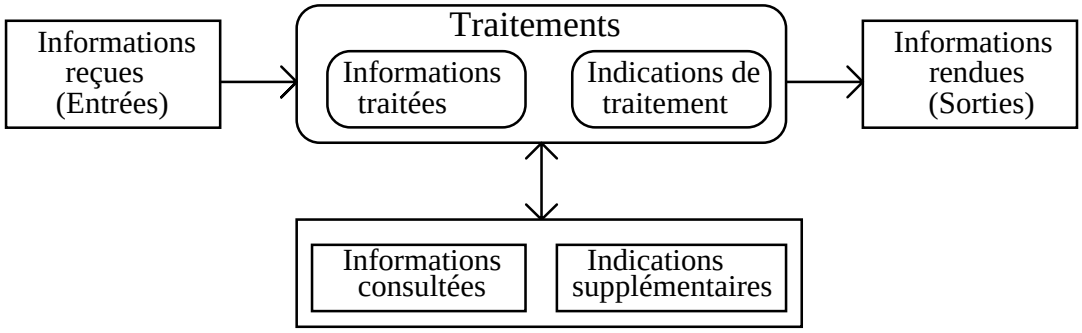



*Figure 4-1 :schéma d'un traitement d'informations par un système informatique*

Ainsi donc, un ordinateur nu ( = sans programme pour le faire agir) n'existe pas. A tout moment, ce qu'on a en face de soi, c'est toujours un tandem *ordinateur + programme*. C'est la raison pour laquelle, plutôt que de parler dans la suite d'ordinateur, nous utiliserons le terme *système informatisé*, qui insiste sur le fait que la seule réalité est l'ordinateur équipé de l'un ou l'autre programme qui guide et détermine ses actions.

Bien entendu, le point de vue adopté sera fort différent suivant qu'il s'agit de celui de l'utilisateur du couple ordinateur-programme ou de celui du programmeur qui va devoir créer la partie programme de ce même couple.

## 4.2 Le point de vue de l'utilisateur

Pour ce dernier, l'ordinateur équipé du logiciel adéquat est un outil ou plutôt un instrument qui va l'aider dans une tâche particulière :

$$\left\{\begin{array}{c}\text{ordinateur}\\ + \\ \text{logiciel}\end{array}\right\} = \text{instrument}$$

### *4.2.1 Des milliers d'instruments*

Ce qui intéresse l'utilisateur, c'est de connaître le mode d'emploi de cet instrument, qu'il s'agisse de faciliter la création et la présentation d'un texte (logiciel de traitement de texte), de simplifier la gestion des emprunts dans une bibliothèque (logiciel de gestion d'une base de données), de tracer l'histogramme représentatif d'une série de données (logiciel tableur),...

A chaque fois, l'ordinateur équipé du logiciel adéquat est perçu comme un instrument qu'il va s'agir de maîtriser et d'utiliser à bon escient dans le cadre d'une tâche précise.

On peut dire qu'aujourd'hui ce sont des milliers de systèmes différents qui sont à la disposition des utilisateurs, et cela dans des domaines fort divers.

Il y a toutefois un double problème pour l'utilisateur novice :

### *4.2.2 L'ordinateur n'existe pas*

D'une part, il est impossible de dresser la liste des manipulations de base d'un ordinateur, l'ensemble de ce qu'il serait nécessaire de maîtriser pour se trouver à l'aise face à un clavier et à un écran. Et cela, tout bonnement parce que l'utilisateur ne sera jamais face à

un *ordinateur*, mais toujours face à un *système* particulier *ordinateur + logiciel*, aux réactions spécifiques[1].

> L'une des demandes les plus fréquemment formulées par les débutants (futurs utilisateurs) est celle d'une initiation pratique (et rapide) aux "manipulations de base" de l'ordinateur : quels sont les savoir faire élémentaires face à l'écran et au clavier ? en un mot, "comment utiliser un ordinateur ?".
>
> Le formateur est alors dans la position inconfortable du vendeur d'appareils électroménagers face à un client qui souhaiterait être rapidement mis au courant des "manipulations de base" de l'électroménager. Nous savons tous que "l'appareil électroménager" n'existe pas et que le vendeur a intérêt à se faire préciser si la curiosité du client potentiel se porte plutôt sur les lessiveuses ou les aspirateurs.
>
> Il en va malheureusement de même de l'ordinateur. Le seul "objet" existant est toujours un couple ordinateur-logiciel. Un ordinateur "nu" (= sans programme qui le gouverne) n'existe pas et l'utilisateur n'a jamais à faire qu'à un tandem particulier "ordinateur-logiciel", les comportements de ces couples pouvant être extrêmement divers. Ceci est d'autant plus dissimulé que rien ne se modifie dans l'aspect physique du partenaire ordinateur des couples évoqués : l'utilisateur a l'impression de garder la même "machine" en face de lui, alors même que ce concept de machine "nue" n'est pas pertinent.
>
> Notons au passage que cette réduction du système informatique (ordinateur + logiciel) à sa seule composante matérielle est ce qui rend tellement choquantes des associations comme "intelligence artificielle" ou des questions comme "l'ordinateur est-il intelligent ?". Si il y a "intelligence" dans le comportement d'un système informatique, il faut redire que c'est dans la partie logicielle qu'elle se trouve : elle sort du cerveau d'un homme qui a dû faire l'effort d'objectiver, de formaliser et d'enclore dans un programme des traitements qui, lorsqu'ils prennent le contrôle de l'ordinateur lui donnent un comportement plus ou moins "intelligent" (Cf. plus haut). Dans le contexte des utilisations de systèmes informatiques, parler d'ordinateur (seul) n'a tout bonnement aucun sens.

### *4.2.3 L'outil informatique n'existe pas*

Par ailleurs, si la composante matérielle du système reste constante au travers des utilisations, la composante logicielle, elle, est par nature extrêmement changeante et cela sans que rien (ou peu de chose) ne signale ces modifications aux yeux de l'utilisateur. Si l'ordinateur est la partie stable du système, il est aussi extraordinairement polygame : au cours d'une simple session d'utilisation, c'est souvent à des dizaines de programmes différents que l'utilisateur aura affaire.

L'outil informatique, au singulier, n'existe donc pas : ce sont des milliers d'outils différents, avec leurs règles de fonctionnement propres et diverses, avec leurs champs d'applications particuliers et leurs modes d'emploi singuliers qui existent.

> Il y a d'ailleurs, avec des environnements comme Windows ou des systèmes comme ceux du MacIntosh, une volonté de standardiser, sinon le mode d'emploi des logiciels, du moins la représentation que l'utilisateur peut s'en faire à travers des interfaces semblables et des menus similaires. Mais, même si le marteau et le tournevis ont à présent la même couleur, ils restent des outils différents, pour faire des choses distinctes : il en est de même pour le tableur et le logiciel de dessin.

Cette constatation a un corollaire important : les utilisations des systèmes informatiques sont des univers sans règle stable : on ne peut pas parler des règles d'utilisation de l'outil puisque il y a *des* outils, chacun avec ses règles particulières.

---

1 Voir aussi à ce propos (DUCHÂTEAU 94).

S'il est un mot à bannir du vocabulaire des utilisations de systèmes informatisés, c'est bien le terme "toujours" : la même action (pression de touche, clic de la souris) ne produit pas "toujours" les mêmes effets

Dans l'univers MS-DOS, l'appui simultané sur les touches Alt Ctrl et Del produit (presque) "toujours" un réamorçage du système; ce n'est plus vrai sous Windows; l'appui sur la touche F1 est souvent (pas "toujours") un appel à l'aide; un double-clic sur le nom d'un fichier lance une application ou produit un message d'erreur; ... Je pourrais ajouter des centaines d'exemples à la liste. Je suis incapable d'en fournir un seul d'une action qui produise toujours le même effet; sauf peut être l'action de retirer la prise de courant... et encore, il y a les portables...

Combien de fois n'est-il pas arrivé qu'un utilisateur novice, qui achevait d'utiliser un éditeur ou un système de traitement de texte et se retrouvait aux prises avec MS-DOS, m'ait appelé pour tenter d'effacer les lignes "syntax error" ou "file not found" apparues suite à des manipulations inadéquates du système d'exploitation. L'instant d'avant, il pouvait à sa guise "remonter" dans le texte affiché à l'écran; quelques secondes plus tard *sur le même écran* l'opération est impossible et n'a plus de sens. En effet, l'écran MS-DOS retrace une *histoire* , celle des échanges entre l'utilisateur et le système; par contre, c'est un *espace*, celui du texte modifiable, dans un traitement de texte. Accéder à une ligne supérieure pour la transformer, cela a du sens dans un traitement de texte, où la métaphore spatiale s'applique à l'écran, mais cela n'a pas de sens en MS-DOS où l'écran est une suite d'instants, une histoire; cela reviendrait à vouloir modifier le passé.

| Dans l'utilisation des systèmes informatisés, vous ne pouvez jamais dire "toujours"! |
|---|

*Principe 4-2 : il ne faut jamais dire "toujours"*

## 4.3 Le point de vue du programmeur

Le court exemple de la conjugaison abordé précédemment nous montre bien à quel point le point de vue du programmeur peut être différent. Il va s'agir, pour lui, de fournir toutes les indications de traitement nécessaires à l'accomplissement d'une tâche par l'ordinateur. C'est lui qui va créer la partie logicielle des instruments que d'autres pourront ensuite utiliser.

Le point de départ de l'activité de programmation est toujours une *tâche* de traitement formalisable d'informations. A toutes celles de la colonne de gauche du tableau présenté plus haut, nous pourrions en ajouter d'autres encore : compter les points lors d'un match de tennis, donner la date de demain, ...

Nous l'avons déjà perçu avec la métaphore du copain portugais, il va s'agir, pour le programmeur, non d'effectuer lui-même chacune de ces tâches élémentaires, mais de fournir toutes les indications indispensables pour les *faire faire*.

Chacune de ces tâches va donc donner naissance à un réel *problème* : celui *d'expliquer* de manière exhaustive et détaillée l'enchaînement des actions nécessaires pour mener la tâche à bien. En quelque sorte :

Programmer
c'est
FAIRE FAIRE

*Principe 4-3 : programmer, c'est faire faire*

L'ordinateur, qui comme ci-dessus ne peut être "nu", sera, dans ce contexte de la programmation, perçu non comme un instrument ou un outil qui va aider à l'accomplissement d'un travail, mais plutôt comme un *exécutant* qui sera chargé d'effectuer une tâche. Il est donc bien plus ressenti comme un obstacle que comme une aide : il faut tout lui dire de la manière de mener à bien la tâche concernée.

Nous le verrons dans la suite, c'est l'ordinateur équipé d'un langage de programmation qui est l'interlocuteur du programmeur. Ce langage de programmation va permettre d'exprimer les indications nécessaires aux traitements, de concevoir et de rédiger les programmes souhaités.

Ainsi, le schéma qui est au cœur de l'activité de programmation est le suivant :

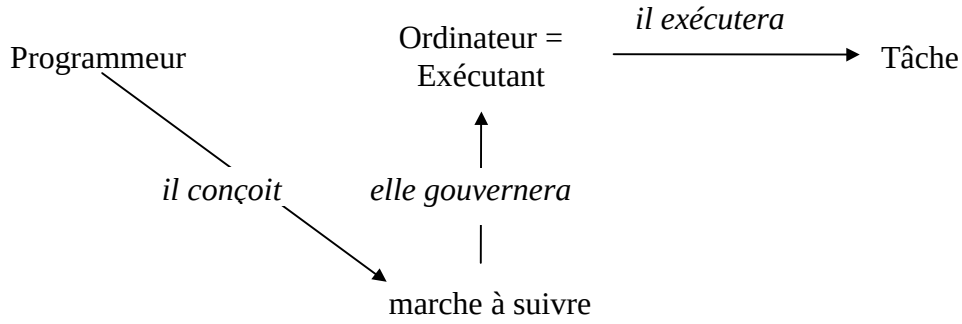



*Figure 4-2 : schéma de l'activité de programmation*

Dès lors, du point de vue du programmeur :

$$\left\{\begin{array}{c}\text{ordinateur}\\+\\\text{langage de programmation}\end{array}\right\}=\text{executant}$$

et

Programmer, c'est face à une tâche formalisable de traitement d'informations (qu'il s'agira de préciser),

face à un exécutant aux capacités limitées mais connues,

concevoir et rédiger une marche à suivre qui fera faire la tâche par l'exécutant.

# 5

# ... machine...

## Architecture d'un système informatique

### 5.1 Architecture générale

Nous avons vu que le schéma proposé page 3 (Figure 2-1 : schéma d'un traitement d'informations) qui préside à tout traitement d'information se particularise dans le cas d'un traitement opéré par un système informatique sous la forme proposée page 54 (Figure 4-1 :schéma d'un traitement d'informations par un système informatique).

Ce schéma va s'incarner en une réalisation matérielle qui constituera en quelque sorte l'architecture générale d'un système informatique.

> C'est à dessein que nous parlerons désormais de "système" informatique et non plus d'ordinateur, puisque, comme nous le savons, "l'ordinateur n'existe pas" (Cf. le principe de la page 53).

Cinq composants essentiels peuvent être mis en évidence :

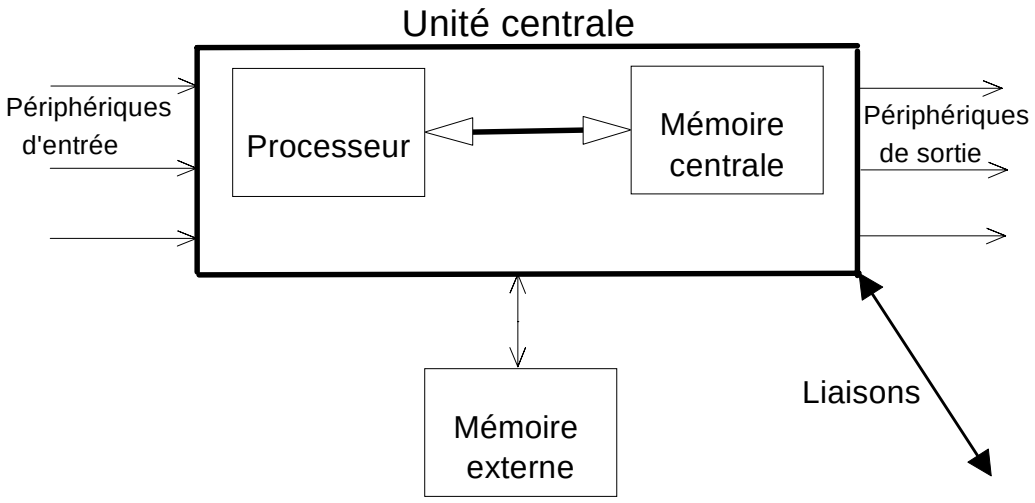



*Figure 5-1 : architecture générale d'un système informatique*

- D'abord, le centre du système, *l'unité centrale*. C'est là que les traitements sur les données sont effectués, sur base des indications de traitement qui y sont également présentes.
- Ensuite, les *périphériques d'entrée* : leur rôle est de transformer une information qui se présente sous une forme qui nous est habituelle en une série finie d'entiers, acceptable

par l'ordinateur (voir le principe énoncé page 17). Un périphérique d'entrée a donc pour rôle de *coder* l'information sous une forme acceptable par l'unité centrale.

- Egalement les *périphériques de sortie* : leur rôle est de transformer des informations telles que l'ordinateur les manipule (série finie d'entiers) en une forme qui nous soit davantage habituelle. Un périphérique de sortie a donc pour rôle de *décoder* l'information manipulée par l'ordinateur (des nombres entiers) en une forme habituelle pour nous, êtres humains.
- Enfin, la *mémoire externe* (on dit aussi mémoire de masse, ou mémoire de stockage à long terme) : elle mémorise à la fois des informations (données) susceptibles d'être traitées par l'ordinateur et des indications de traitement (programmes) supplémentaires.

  On utilise souvent comme synonyme les termes "information" et "donnée" pour désigner ce que manipule l'ordinateur. Il s'agit bien entendu, dans tous les cas, d'une forme codée de ce que, entre êtres humains, nous appelons "information" ou "donnée".

- Et pour terminer, le cas échéant, des *liaisons* entre le système considéré et d'autres systèmes informatiques : périphériques de communication, connexion à un réseau,...

Nous reviendrons dans la suite sur divers composants de ce système, mais nous allons dans un premier temps, nous focaliser sur l'unité centrale et procéder en quelques sortes à des zooms successifs qui nous amèneront à examiner avec de plus en plus de détails de quoi est faite cette unité centrale.

## 5.2 L'unité centrale

A un premier niveau d'analyse, nous pouvons la voir comme constituée de trois éléments : la mémoire centrale, le processeur et des liens permettant l'échange entre ces deux entités.

### *5.2.1 La mémoire centrale*

C'est dans ce dispositif que sont mémorisées les deux choses qui nourrissent un ordinateur : les informations (données) à traiter (sous forme codée) et les indications pour commander ces traitements (programmes).

#### *5.2.1.1 Qu'y a-t-il dans la mémoire centrale ?*

D'un premier point de vue, qu'on pourrait qualifier de *fonctionnel*, la mémoire centrale contient donc deux choses : les *données*, les informations manipulées (dont nous savons déjà qu'elles sont en quelque sorte codées sous la forme de nombres entiers) et les *programmes exécutables* par le processeur, autrement dit, des séries d'instructions qui indiquent comment les données présentes doivent être manipulées par ce processeur.

Nous verrons dans la suite que les mémoires externes contiennent elles aussi ces deux types d'entités, mais il est bon dès à présent, d'énoncer un principe important :

> Pour qu'un programme exécutable puisse faire agir le processeur, il doit être présent en mémoire centrale. Pour que des données soient traitées par le processeur, elles doivent également résider en mémoire centrale.

*Principe 5-1 : pour être exécutable, un programme doit être en mémoire centrale*

Le mot "exécutable" accolé au terme "programme" peut paraître étonnant : il insiste sur le fait que les instructions composant ce programme sont telles quelles acceptables par le

processeur et susceptibles de le faire agir. Mais surtout, la raison pour laquelle nous parlons de *programme exécutable*, c'est que nous utiliserons souvent dans la suite le mot *programme* pour désigner, toujours des indications de traitement, mais qui ne sont pas directement acceptables, *exécutables*, par le processeur.

#### *5.2.1.2 Les deux types de mémoire centrale*

Nous pouvons, sur base des principes énoncés et de constatations que chacun peut faire, élaborer un court raisonnement :

On sait que :

- TOUT ce que fait un ordinateur, il le fait gouverné par un programme (page 53)
- Pour qu'un programme exécutable puisse faire agir le processeur, il doit être présent en mémoire centrale (principe énoncé ci-dessus).

On constate que :

- Lorsqu'un ordinateur est mis sous tension (= lorsqu'on le rallume), il commence à travailler (il affiche par exemple "des choses" à l'écran)

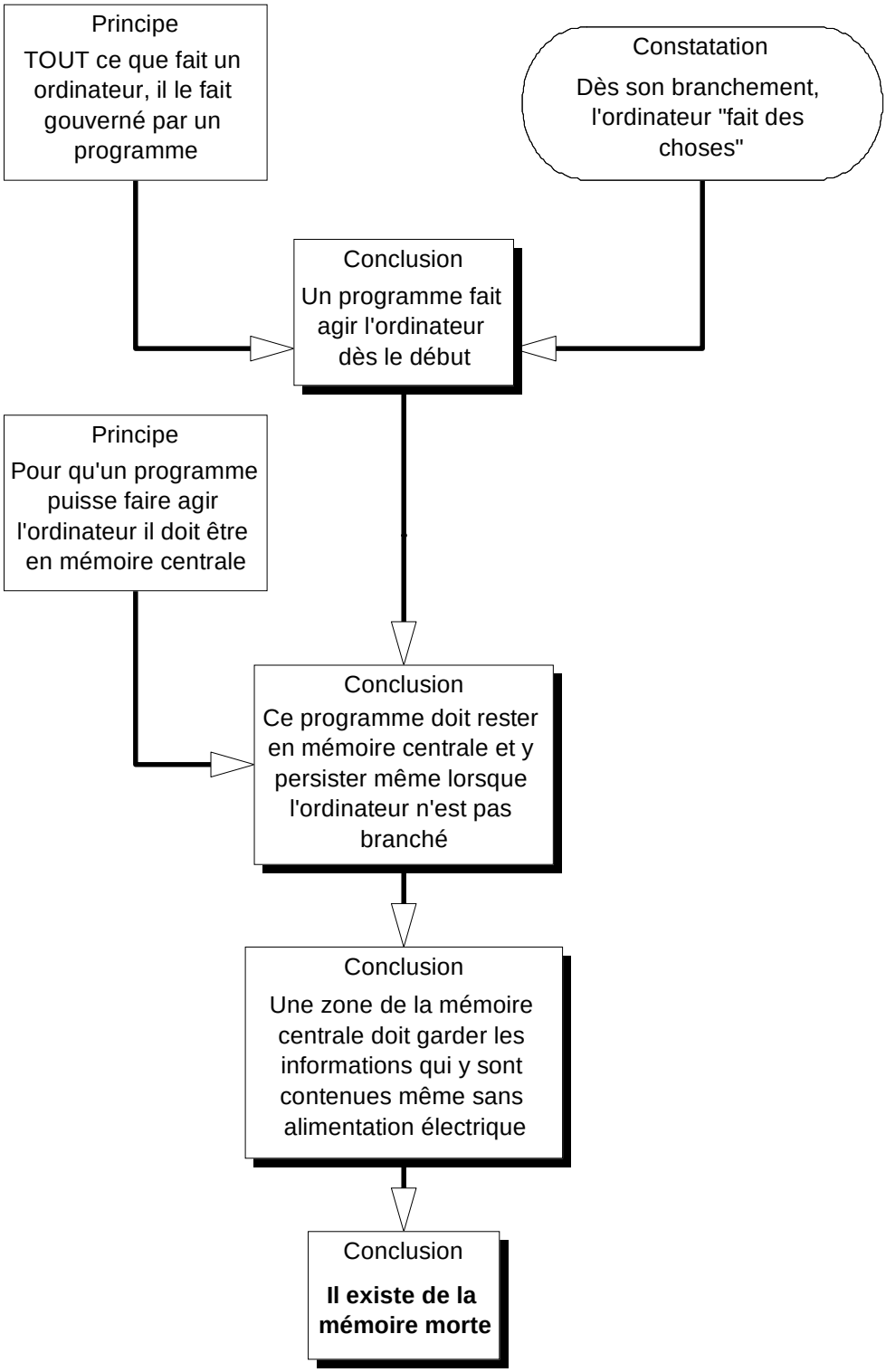
C'est donc, forcément qu'

- Il y a déjà en mémoire des programmes qui, dès sa mise sous tension, font agir l'ordinateur.

Si l'on soutient que les programmes qui font ainsi agir l'ordinateur au début de son activité ont été amenés de l'extérieur (c'est à dire, par exemple, des mémoires de masse ou mémoires externes), on ne fait que reporter d'un cran le problème. En effet, aller chercher ces programmes sur les mémoires de masse constitue une action de l'ordinateur et dès lors, en vertu des deux principes rappelés, il faut qu'un programme soit déjà présent en mémoire centrale pour permettre cette action.

Cette chaîne doit bien s'interrompre par la conclusion que, dès le début, des programmes résident déjà en mémoire centrale. De plus, ces programmes (et des données associées) doivent rester dans cette mémoire même lorsque l'ordinateur n'est plus sous tension, puisque ces programmes le font agir dès qu'il est alimenté en électricité. Il est donc indispensable qu'une partie de la mémoire centrale puisse garder les programmes et données qu'elle contient, même en l'absence d'alimentation électrique. Cette mémoire dont le contenu est préservé et est essentiel au moment de la mise sous tension de l'ordinateur est la *mémoire morte*.

Il nous faut également admettre qu'une autre partie de la mémoire centrale est non préservée : c'est celle qui accueillera les programmes et données supplémentaires venant des mémoires externes et qui rendront l'ordinateur capables d'un tas d'actions supplémentaires. Cette zone de mémoire, "vide" au démarrage de l'ordinateur et qui se remplira pendant le travail avec des programmes (et des données) venus de l'extérieur est la *mémoire vive*.




#### 5.2.1.2.1 La mémoire morte

On vient de le voir, elle est caractérisée par plusieurs traits :

- Son contenu ne disparaît jamais, même lorsque l'ordinateur n'est plus alimenté en courant électrique; c'est en quelque sorte une zone de mémoire dont le contenu est gardé en permanence, une zone de mémoire "*increvable*".

  On notera au passage le paradoxe qui veut que la mémoire qui reste en quelque sorte toujours "*vivante*" (en me pardonnant cet abus) s'appelle la mémoire "*morte*".

- Pendant le travail de l'ordinateur, le système peut évidemment amener au processeur des instructions et des données présentes dans cette mémoire morte. Par contre, jamais cette zone de mémoire ne verra son contenu modifié lors du travail. Il s'agit en quelque sorte d'une mémoire "intouchable". Le processeur peut y *lire* des instructions et des données, jamais il ne pourra y *écrire* quoi que ce soit. En anglais cette mémoire est d'ailleurs désignée par l'acronyme ROM, signifiant "Read Only Memory" (= mémoire dont le contenu peut seulement être lu (par le processeur)).

  On notera ici l'emploi consacré par la tradition informatique des termes *lire* et *écrire*. *Lire* veut dire que le processeur copie en son sein une instruction ou une donnée présente en mémoire centrale; notons au passage que cette lecture ne vide pas la ou les cellules de la mémoire contenant ce qui a été lu : c'est toujours une copie que le processeur saisit. *Ecrire*, à l'inverse, signifie que le processeur modifie l'une ou l'autre cellule de la mémoire en y plaçant une donnée ou une instruction qui y remplace l'ancien contenu.

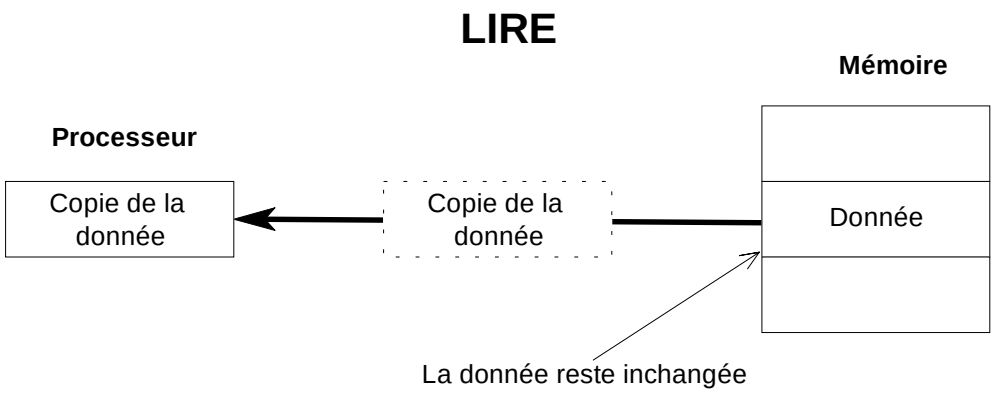



*Figure 5-2 : schéma de la lecture*

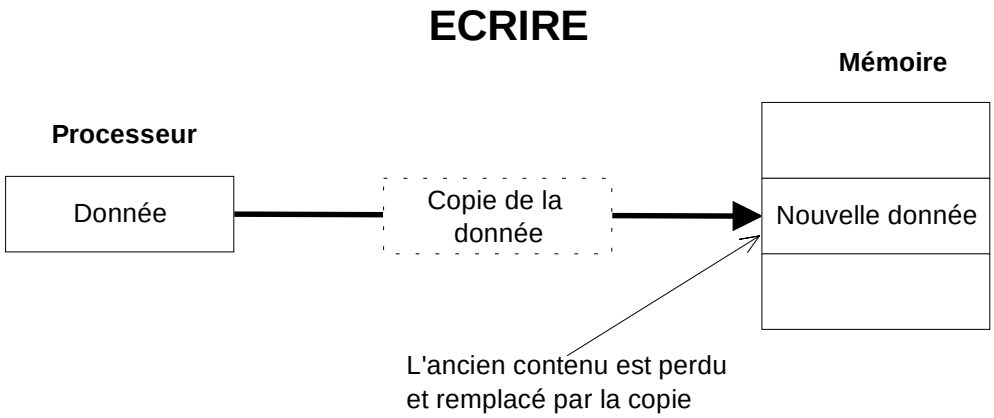



*Figure 5-3 : schéma de l'écriture*

- L'ensemble des programmes et données qui restent (à jamais) en mémoire morte constituent le *BIOS* (Basic Input Output System). On y trouve essentiellement :
  - Des programmes de tests internes exécutés dès l'allumage de l'ordinateur (pour vérifier le bon état d'un certain nombre de constituants.
  - Un programme commandant au processeur d'aller chercher sur les mémoires externes un ensemble de programmes et données supplémentaires qui seront amenés en mémoire vive et rendront l'ordinateur capable d'actions supplémentaires; ces programmes supplémentaires et indispensables, chargés en

mémoire vive, constituent le (ou une partie du) *système d'exploitation* (voir page 173) de l'ordinateur.

Ce court programme enfermé en mémoire morte et chargé d'amener les programmes supplémentaires indispensables au fonctionnement de l'ordinateur est le programme de *bootstrapping* (ou en abrégé de "boot").

- Un ensemble de programmes et de données correspondant aux actions les plus rudimentaires dont l'ordinateur est capable; ainsi on y trouve par exemple les programmes permettant des lectures à partir du clavier et des affichages à l'écran. Ces programmes correspondant à des actions fort élémentaires seront activés à de très nombreuses reprises par des programmes plus élaborés, chargés en mémoire vive, et qui "passeront la main" à ces programmes du BIOS, enfermés dans la mémoire morte.

#### 5.2.1.2.2 La mémoire vive

C'est la zone de mémoire dans laquelle viendront prendre place programmes et données venant de l'extérieur. Elle a les caractéristiques suivantes :

- Son contenu disparaît lorsque l'ordinateur n'est plus sous tension

  Cette remarque à des conséquences pratiques évidentes : les documents que l'utilisateur est en train de créer (une lettre en utilisant le couple ordinateur + traitement de texte, un tableau statistique en utilisant le couple ordinateur + tableur,...) prennent évidemment place dans la zone de mémoire vive. Tant que l'utilisateur n'a pas demandé au système d'enregistrer ce document sur un support de mémoire externe, la moindre coupure de courant, en vidant la mémoire vive, peut réduire à néant le travail de plusieurs heures.

- Le processeur peut y accéder tant en lecture qu'en écriture; cet accès complet est rappelé par l'acronyme anglais désignant la mémoire vive : RAM (Random[1] Access Memory).

#### 5.2.1.2.3 Une question

Certains paramètres descriptifs du système (le type et le nombre de mémoires externes - disques durs, lecteurs de disquettes[2] - , l'indication de la mémoire externe spécifique à partir de laquelle se fera l'opération de bootstrapping, …) sont essentiels au bon fonctionnement dès la mise en marche. Ces paramètres (au rang desquels il faut aussi placer la date et l'heure) sont sujets à modification (par exemple quand on change de type de disque dur) : ces données qui doivent être *mémorisées* doivent donc aussi être modifiables. On ne peut donc les enfermer dans la mémoire morte : leur modification entraînerait dans ce cas la nécessité de changer physiquement la ROM. Mais si, fasse à cette impossibilité de figer ces paramètres en ROM, on retient la solution de la mémoire vive (dans laquelle les données mémorisées peuvent de fait être modifiées), on se heurte à une autre difficulté : les données mémorisées en RAM disparaissent en l'absence d'alimentation électrique et dès lors, l'ensemble de ces paramètres devraient être redemandés à l'utilisateur dès la mise en marche (et en tout cas avant l'opération de bootstrapping), ce qui n'est guère concevable.

---

1 Le terme Random ne sera pas davantage explicité; dans l'acronyme RAM, c'est le terme Access qu'il faut souligner.

2 Voir ci-après pour une description plus complète des mémoires de masse (mémoires externes) : disquettes, disques durs, CD-ROM, cassettes, …

?

Quelle solution envisager pour mémoriser ces paramètres descriptifs du système qui concilie le fait de pouvoir (de temps à autre) modifier ces données, mais également faire en sorte que leurs valeurs soient conservées, même en l'absence de branchement de l'ordinateur sur le réseau électrique.

On pourrait peut-être proposer une solution où ces données soient conservées sur les mémoires externes (comme par exemple le disque dur) et chargées (= amenées) en RAM lors de l'opération de bootstrapping. Cette solution ne résiste cependant pas à la constatation que cette opération (qui consiste au démarrage à aller chercher sur les mémoires de masse des programmes et données essentielles à la poursuite du travail) nécessite que les caractéristiques de ces mémoires externes soient connues, avant de pouvoir y accéder.

En d'autres termes, avant même d'aller pêcher des informations sur le disque dur, le système doit connaître le type et les caractéristiques de ce disque dur : ces caractéristiques ne peuvent donc être portées par le disque : elles devraient être disponibles avant même d'avoir été lues.

La solution consiste à alimenter une petite portion de la mémoire vive (modifiable) par une batterie ou une pile électriques qui assure la "survie" des informations qui y sont mémorisées, même en l'absence débranchement sur le réseau (= quand l'ordinateur est "éteint").

C'est un type particulier de mémoire vive, la CMOS, caractérisée par le fait qu'elle ne nécessite qu'une faible alimentation électrique pour conserver les données (modifiables) mémorisées qui est utilisée dans ce cas.

L'ensemble de ces paramètres, conservés dans cette zone particulière de la mémoire vive maintenue sous tension, constitue ce qu'on appelle le setup du système.

Il est possible, en enfonçant une combinaison de touches appropriée au démarrage du système (après les tests internes et avant le bootstrapping) de modifier les paramètres mémorisés dans le setup.

#### *5.2.1.3 L'organisation physique de la mémoire centrale*

Je ne souhaite pas ici entrer dans les détails techniques de description des dispositifs physiques permettant la mémorisation des informations : la seule chose essentielle consiste à savoir que cette mémorisation prend la forme de minuscules charges électriques contenues dans de microscopiques condensateurs. Ce qu'il faut en retenir, c'est seulement que ces dispositifs physiques de mémorisation n'ont que deux états possibles : chargés ou déchargés.

Dès lors, tout ce qui sera mémorisé en mémoire centrale (les données comme les programmes), est finalement "écrit" par des juxtapositions de dispositifs élémentaires n'ayant que deux états possibles : chargés ou déchargés.

De manière symbolique, on peut dès lors affirmer que tout ce qui est mémorisé dans la mémoire centrale est codé par deux signes 0 et 1. On peut donc modéliser la mémoire centrale comme une juxtaposition de cases ne pouvant comporter que les deux symboles 0 ou 1.

Il est essentiel d'avoir compris qu'il s'agit là d'un modèle, pratique et efficace pour parler de la structure de la mémoire centrale, mais il ne faut pas confondre ce modèle avec la réalité physique dont il rend compte : il n'y a pas de "case" dans un ordinateur, ni de "0" ou de "1". Un ordinateur, c'est, en fin de compte du sable (silicium) et de l'acier, ou d'un autre point de vue, des électrons qui circulent. Mais même (surtout !) quand on parle de cette manière on est

bien, à chaque fois en présence de modèles d'une certaine "réalité" (peut être insaisissable autrement qu'en la modélisant).

Nous savions déjà que pour être traitables par un ordinateur, toutes les informations devaient être codées sous le forme de séries finies de nombres entiers (Principe 3-1: codage de l'information, page 17). Nous pouvons faire à présent un pas de plus : ces nombres entiers devront eux mêmes être écrits sous la forme de succession de 0 ou de 1 : l'alphabet de l'ordinateur est binaire.

> L'alphabet de l'ordinateur est binaire : tout ce qui est écrit (codé) en mémoire centrale, les données comme les programmes exécutables, est écrit sous la forme d'une succession des symboles 0 et 1.

*Principe 5-2 : l'alphabet de l'ordinateur est binaire*

Il ne faut pas confondre *alphabet* et *langage* : nous avons le même alphabet (à très peu de choses près) que les allemands ou les portugais, et nous ne parlons évidemment pas le même langage. L'alphabet de l'ordinateur est binaire, il ne comporte que les deux signes 0 et 1. Le langage de l'ordinateur (pour autant que ce terme ait du sens) pourrait être compris comme l'ensemble des mots "compris" par le processeur et capables de la faire agir. On pourrait alors dire que tous les ordinateurs ont le même alphabet (binaire) mais que seuls des ordinateurs équipés du même processeur ont le même langage.

#### 5.2.1.3.1 L'unité de mémoire élémentaire : le BIT

La mémoire centrale peut donc être modélisée comme une immense juxtaposition de cases élémentaires, chacune ne pouvant comporter que l'un des symboles 0 ou 1 :

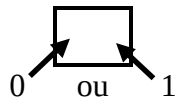



*Figure 5-4 : un bit*

Une telle case élémentaire, ne pouvant prendre que l'une des deux valeurs 0 ou 1 est appelée un BIT (BInary digiT ou chiffre binaire).

On a pris l'habitude de désigner par 0 et 1 les deux symboles constitutifs de l'alphabet des ordinateurs. N'importe quel autre couple de symboles distincts conviendrait évidemment. On pourrait avoir choisi O (pour ouvert) et F (pour fermé) ou encore C (pour chargé) et D (pour déchargé),…

Il faut noter qu'à l'instar d'une porte qui ne peut être qu'ouverte ou fermée, un BIT ne peut être que 0 ou 1. Une case élémentaire de mémoire n'est jamais vide : elle contient toujours soit 0 soit 1.

Ainsi donc, le BIT, unité élémentaire permettant la mémorisation de l'information, est donc une unité de mesure de taille mémoire. C'est aussi une unité de mesure possible de quantité d'information (en gardant le sens formel attribué à cette notion d'*information* aux chapitres 1 et 2). Nous y reviendrons ci-dessous.

?

> Combien d'informations différentes peut on écrire (coder) avec 1 BIT ? Et avec deux ? Et avec 8 ?

#### 5.2.1.3.2 L'octet

Si l'unité élémentaire de la mémoire est bien le BIT, on a pourtant pris l'habitude, essentiellement pour des raisons historiques, de considérer la mémoire comme organisée en groupes ou en paquets de 8 bits. Un tel groupe est appelé *octet* ou *byte*.

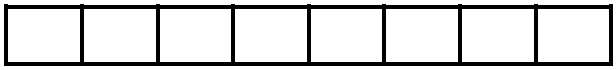


*Figure 5-5 : un octet*

Un octet est donc une succession de huit cases élémentaires, ou encore un octet = 8 bits.

C'est en réalité l'octet (ou byte) que nous utiliserons comme unité de mesure de la taille d'une mémoire ou comme unité de mesure de quantité d'informations.

?

Combien d'informations différentes peut-on écrire (coder) avec un octet ?

Comme on le verra dans la suite, l'octet est une unité trop petite et, exactement comme un parle de gramme et de kilogramme, on parle aussi d'octet et de kilo-octet. Mais nous reviendrons à ces unités, ci-après, quand nous aurons pu leur donner un peu plus de sens et de portée.

Retenons en tout cas que

Tout ce que codera un ordinateur sera écrit octet par octet, aussi bien les informations manipulées que les instructions des programmes qui feront agir le processeur

*Principe 5-3 : l'octet est l'unité de taille mémoire et de quantité d'information*

#### *5.2.1.4 Les adresses des cellules de la mémoire*

Chaque cellule de la mémoire centrale dont la taille est fréquemment d'un octet, est repérée par un numéro que l'on appelle son *adresse*. Lorsque le processeur accède à la mémoire centrale, soit pour y lire une information contenue dans une cellule, soit pour y écrire une information, il le fait en précisant l'adresse de la cellule concernée par cette lecture ou cette écriture. Le processeur se contente donc de préciser l'adresse de la cellule à laquelle il souhaite accéder et d'indiquer s'il s'agit de lire le contenu de cette cellule ou d'y écrire.

Ainsi, si le processeur souhaite lire le contenu de la cellule d'adresse binaire 1100 (12 en décimal) il transfère vers la mémoire (à travers le bus d'adresse, voir le schéma ci-dessous) l'adresse 1100 de la cellule souhaitée et précise à la mémoire qu'il s'agit de lire le contenu de cette cellule. Il recueille alors à la sortie de la mémoire le contenu ce cette cellule (par exemple 10000001 (129 en binaire). Ce contenu est transféré vers le processeur à travers le bus de données (voir ci-dessous).

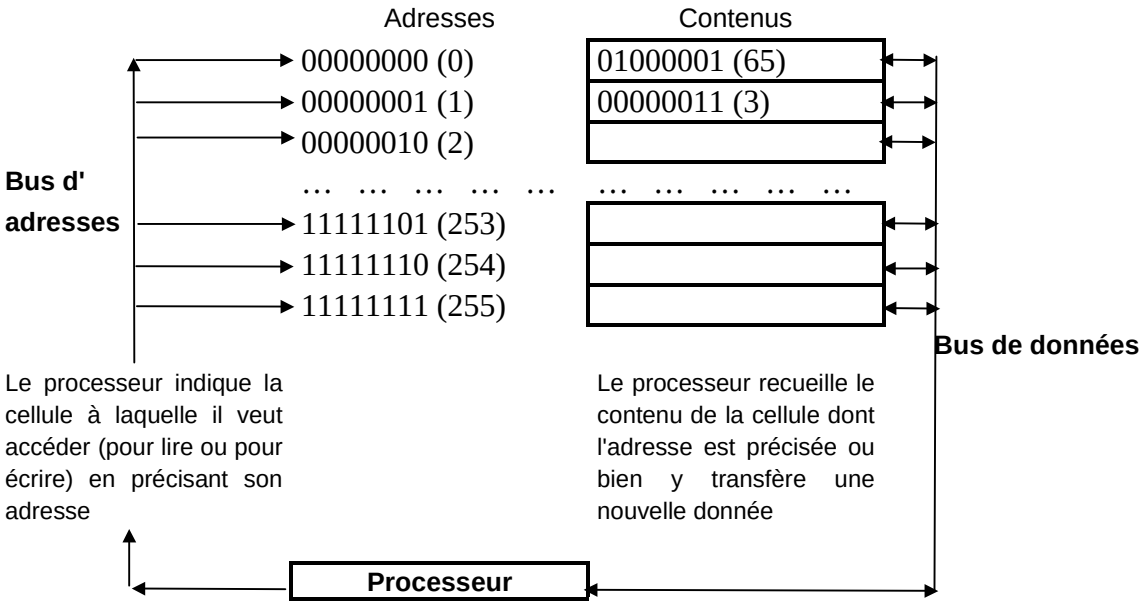



*Figure 5-6 : les adresses*

Il est important de ne pas confondre l'adresse d'un emplacement mémoire avec son contenu. Dans le schéma présenté ci-dessus, nous avons supposé que les adresses des cellules mémoires étaient écrites sur un octet et que les contenus de ces cellules étaient également d'un octet.

> Il est essentiel d'avoir saisi ce mode de fonctionnement : le travail consiste essentiellement en des échanges entre la mémoire centrale et le processeur. Le processeur indique l'adresse de la cellule concernée par cet échange (à travers le bus d'adresses) et *écrit* ou *lit* (à travers le bus de données) dans cette cellule.

### *5.2.2 Un détour par le codage binaire*

Même si le fait que l'alphabet de l'ordinateur est binaire et même si, heureusement, nous n'aurons pas explicitement à nous en préoccuper (ne serait-ce que parce que les périphériques d'entrée et de sortie coderont et décoderont pour nous), certains faits ne pourront se comprendre qu'en tenant compte de cette contrainte de l'écriture des nombres en binaire.

#### *5.2.2.1 Comment écrivons nous les nombres*

Nous avons, chacun le sait, 10 symboles à notre disposition, les 10 chiffres : 1, 2, 3, 4, 5, 6, 7, 8, 9 et 0 qui va jouer un rôle très particulier.

Les premiers nombres (de 1 à 9) peuvent s'écrire sans problème par la succession des symboles disponibles : 1, 2, 3, 4, 5, 6, 7, 8, 9. Pour le suivant, il ne reste plus de symbole supplémentaire, sauf le 0 : c'est là que va jouer la solution connue sous le vocable de *numération décimale de position*. On écrira dès lors 10, ce qui signifie une *dizaine* et zéro unité. On poursuit le système jusque 99 (neuf *dizaines* et neuf *unités*), ensuite on fait le pas consistant à écrire 100, soit une *centaine*, zéro *dizaine* et zéro *unité*.

Dans ce système, c'est le rang qu'occupe un symbole qui lui donne sa valeur; ainsi 6034

| | | | |
|---|---|---|---|
| 6 | 0 | 3 | 4 |
| 6 milliers | 0 centaine | 3 dizaines | 4 unités |
| $6 \times 10^3$ | $0 \times 10^2$ | $3 \times 10^1$ | $4 \times 10^0$ |

Comme on s'est donné dix symboles (0 compris), on dit qu'on note les nombres en base 10 et ce sont les puissances de 10 qui sont importantes et donnent par ailleurs des nombres bien "ronds" comme 0, 10, 100, 1000.

*5.2.2.2 Comment les nombres sont écrits en binaire*

Le codage binaire suit les mêmes règles, mais comme on ne permet que deux symboles (0 et 1), c'est les puissances de deux qui seront importantes.

Ainsi, on commence avec

1 pour noter un

mais on est immédiatement tenu de passer à

10 signifiant une deuzaine et zéro unité pour noter deux

puis

11 signifiant une deuzaine et une unité pour noter trois

puis

100 signifiant une quatraine, zéro deuzaine et zéro unité pour noter quatre

etc..

Un nombre comme

| 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|
| 1 seizaine | 0 huitaine | 1 quatraine | 0 deuzaine | 1 unité |
| $1 \times 2^4$ | $0 \times 2^3$ | $1 \times 2^2$ | $0 \times 2^1$ | $1 \times 2^0$ |

signifie donc seize + quatre + un, soit vingt et un, qui serait écrit 21 (deux dizaines et une unité) en notation décimale.

Pour passer de la notation décimale à la notation binaire, la méthode est simple et je vais l'illustrer sur un exemple : le codage en binaire de cent.

divisions successives

| 1 | 3 | 6 | 12 | 25 | 50 | 100 |
|---|---|---|---|---|---|---|
| on divise par 2 | on divise par 2 | on divise par 2 | on divise par 2 | on divise par 2 | on divise par 2 | on divise par 2 |
| Quotient : 0 | Quotient : 1 | Quotient : 3 | Quotient : 6 | Quotient : 12 | Quotient : 25 | Quotient : 50 |
| Reste : **1** | Reste : **1** | Reste : **0** | Reste : **0** | Reste : **1** | Reste : **0** | Reste : **0** |

lecture de gauche à droite du codage binaire

*Figure 5-7 : écriture en binaire*

On divise donc le nombre initial puis les quotients successifs obtenus par 2 en notant chaque fois le reste de cette division (qui sera forcément 1 ou 0). On arrête lorsque le quotient est nul : les restes successifs obtenus donnent alors (de gauche à droite) l'écriture en binaire du nombre initial. Ici, 100 s'écrira donc en binaire 1100100 (une quatraine ($2^2$) + une trente-deuzaine ($2^5$) + une soixante-quatraine ($2^6$))

Il faut bien noter que les nombres bien "ronds" sont en binaire les puissances de 2 (alors qu'en décimal, il s'agissait des puissances de 10) :

| | | |
|---|---|---|
| 10 | soit en décimal | 2 |
| 100 | soit en décimal | 4 |
| 1000 | soit en décimal | 8 |
| 10000 | soit en décimal | 16 |
| 100000 | soit en décimal | 32 |
| | etc. | |

### *5.2.3 Comment les informations sont codées en mémoire*

Nous savions déjà que toutes les informations étaient devenues des nombres entiers. Il reste à voir comment ces nombres sont codés en binaire.

#### *5.2.3.1 Le codage des caractères*

Chaque caractère s'était vu associer un nombre entier (entre 0 et 255) à travers le code ASCII ou le code ANSI (Voir pages 20 et 23).

Le nombre maximal de caractères représentables s'explique à présent si l'on ajoute que un caractère (ou plutôt l'entier qui lui est associé) est codé sur un octet.

On a en effet 256 configurations différentes d'un octet, depuis

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Figure 5-8 : l'octet nul*

jusque

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

*Figure 5-9 : l'octet codant 255*

qui est l'écriture binaire de 255.

Ainsi, le caractère A auquel le code ASCII avait associé l'entier 65 sera codé par l'octet correspondant à l'écriture binaire de 65, soit

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

*Figure 5-10 : l'octet codant A*

ou encore, une soixante-quatraine et une unité.

Le codage d'un texte, succession de caractères, se fera à travers la succession des octets codant ces caractères (un octet par caractère du texte).

#### *5.2.3.2 Le codage des nombres entiers*

##### 5.2.3.2.1 Les entiers : des successions de caractères-chiffres

Un nombre entier peut d'abord être considéré comme la succession des caractères qui servent à l'écrire. Il est alors codé comme n'importe quelle portion de texte à raison d'un octet par caractère.

Ainsi, dans la phrase "On pouvait compter 253 présents à la réunion" (qu'on pourrait trouver par exemple au sein d'un texte produit et manipulé à l'aide d'un logiciel de traitement de texte), "253" n'est pas considéré comme un entier, mais plutôt comme la succession des caractères "2", "5" et "3" au même titre que le mot "compter" est codé comme la succession des caractères "c", "o", "m", "p", "t", "e" et "r".

Il va de soi que dans ce cas l'entier est codé comme la succession des caractères qui le composent et pour coder "253", on utilisera 3 octets.

5.2.3.2.2 Les entiers vus comme des nombres

Lorsqu'on souhaite par exemple effectuer des calculs sur les entiers considérés, il est évident qu'ils doivent être considérés comme des nombres en non comme du texte.

?

Comment l'ordinateur peut-il savoir si le "253" qu'on frappe au clavier doit être considéré comme un nombre ou comme du texte ?

Il suffit d'indiquer sur combien d'octets sont codés les entiers. Malheureusement, la réponse à cette question n'est pas uniforme. Le chapitre précédent nous a appris que la seule réalité était toujours un système informatique, autrement dit, un couple ordinateur + logiciel. Le choix du nombre d'octets retenus pour écrire en mémoire un entier dépend donc du logiciel qui équipe l'ordinateur considéré.

> C'est toujours le même problème : parler du codage des entiers par l'ordinateur n'a pas de sens (puisque "l'ordinateur n'existe pas". C'est on le devine, le logiciel qui contrôle l'ordinateur qui fait décider de la manière dont ce qui est frappé au clavier va être codé en mémoire; "253" frappé au clavier sera codé comme du texte (sur trois octets) si c'est tel ou tel logiciel qui est à l'oeuvre et codé comme un entier sur deux octets si c'est tel autre ou sur 4 octets pour un troisième.

Si nous retenons (ce qui est fort souvent le cas) le chiffre de deux octets, pour le codage d'un nombre entier, nous sommes en mesure de mener le raisonnement suivant : en général (sauf si on souhaite ne représenter que des nombres positifs), un BIT va servir à retenir le signe du nombre entier considéré :


*Figure 5-11 : codage d'un entier*

En réalité, le codage du signe est un peu plus complexe que ce qui est indiqué ici, mais le résultat revient en gros à admettre que un BIT étant perdu pour le signe, il ne reste plus que 15 bits pour écrire l'entier à coder.

La réponse est alors immédiate : avec les 15 bits restant on a droit à $2^{15}$ configurations différentes, soit 32.768 configurations. Si 0 est compté comme positif (et codé avec une succession de 15 zéros), on pourra donc écrire sur les 15 bits les nombres entre 0 et 32.767 (ce qui fait bien 32.768 configurations différentes).

Pour les négatifs, on a droit aussi à 32.768 configurations; comme on peut commencer à -1, on pourra écrire les nombres de -1 à -32.768.

Ainsi, on pourra coder au total tous les entiers entre -32.768 et 32.767.

?

Quels entiers pourraient-on coder, si le codage se fait sur quatre octets ? Et si l'on décide de ne coder sur deux octets que les entiers positifs (zéro compris) ?

Comme on le voit, les entiers codables sont relativement limités en taille. Même si l'on augmente le nombre d'octets utilisés pour ce codage, il subsistera toujours une borne à priori sur la grandeur des entiers manipulables.

On touche là une des contraintes essentielles de l'univers informatique. En mathématique, lorsqu'il est question de l'ensemble des entiers (ce qu'on note souvent $\mathbb{Z}$), on sait que cet ensemble est infini : il n'y a pas de limite à la taille des entiers. En informatique, c'est seulement un intervalle fini de l'ensemble des entiers qui est représentable et manipulable : on ne peut coder tous les entiers imaginables.

En résumé, il faut retenir que

> Les nombres entiers codables et manipulables par l'ordinateur sont de taille finie. On ne peut travailler qu'avec un intervalle limité de l'ensemble des entiers (par exemple de -32.768 à 32.767).

Une question se pose alors : que se passe-t-il lorsque les manipulations effectuées sur les entiers conduisent à dépasser les bornes fatidiques au-delà desquelles les entiers ne peuvent plus être codés (en utilisant le nombre d'octets fixé) ? Ainsi, avec des entiers codés sur deux octets, que se passe-t-il lorsque on commande une opération comme 20.000 + 20.000 qui, conduit à un résultat, 40.000, qui n'est plus représentable ?

A nouveau, la réponse à cette question n'est pas uniforme, mais dépend du logiciel qui contrôle l'ordinateur pendant ce dépassement de la taille des nombres entiers représentables.

Voici par exemple, dans le cas d'un petit programme, écrit en Pascal, qui commande à l'ordinateur de placer en mémoire (dans deux cases destinées à contenir des entiers et constituées chacune de deux octets) puis d'en afficher la somme, le résultat surprenant obtenu :

```
var A,B : integer;
```
(on définit deux casiers pouvant contenir chacun un entier)
```
begin
A:=20000;
```
(on place 20000 dans le premier casier)
```
B:=20000;
```
(on place 20000 dans le second casier)
```
writeln('La somme de ',A,' et ',B,' est ',A+B);
end.
```
(on demande l'affichage de la somme des contenus des deux casiers contenant chacun l'entier 20000)

L'exécution de ce programme conduit à l'affichage

```
La somme de 20000 et 20000 est -25536
```

Le résultat correct, 40.000, qui n'est pas codable est remplacé par un nombre négatif, -25.536 !

Avec certains logiciels, le résultat est aberrant : la somme de deux entiers positifs donne un résultat négatif. Avec d'autres, les choses sont plus claires : l'ordinateur "se plante" ou signale une erreur.

*5.2.3.3 Le codage des nombres réels*

Il faut signaler à nouveau que la succession des symboles constituant le nombre peut être vues comme du texte codé octet par octet. Ainsi, "345,56" peut être codé comme du texte, sur 6 octets successifs.

Ce qui intéresse ici, c'est de considérer "345,56" comme un nombre. On sait déjà qu'il sera codé, en *virgule flottante*, sous la forme de deux entiers, la *mantisse* soit 34556 et *l'exposant* soit 2. Rappelons en effet qu'on passe de

345,56 à $3{,}4556 \times 10^2$ puis à 34556 et 2 (Voir page 24)

On devine dans ces conditions que mantisse et exposant intervenant dans le codage du nombre seront codés séparément sur un certain nombre d'octets.

A nouveau, il me faut signaler que la réalité est un peu plus compliquée que ce que j'en décris ici. L'important, ce sont les conclusions auxquelles on arrive.

Si par exemple la mantisse est codée sur 3 octets et l'exposant sur 1 octet, on est conduit aux contraintes suivantes :

- Les trois octets de la mantisse permettent puisque l'un des bits est perdu pour le signe un nombre maximal de l'ordre de $2^{23}$ soit 8.388.608 qui est un nombre de 7 chiffres. Retenons donc que le nombre de chiffres gardés dans l'écriture de la mantisse est limité à cause du codage effectué.
- Le codage de l'exposant sur un octet conduit, puisque là aussi un BIT est perdu pour le signe, à un exposant de l'ordre de $2^7$ soit 128.

  Il faut cependant ajouter que, comme le codage se fait en binaire, l'écriture en virgule flottante fait appel aux puissances de 2 et non de 10 et que dès lors si l'exposant maximal est de l'ordre de 128, la grandeur du nombre est de l'ordre de $2^{128}$ ou, en revenant à la base 10, de l'ordre de $10^{38}$.

  Ainsi donc l'exposant est limité, même s'il permet de coder des nombres de l'ordre de $10^{38}$ qui sont énormes.

Dès lors, à travers le codage de la mantisse et de l'exposant, on constate les deux faits essentiels suivants :

Le nombre de chiffres possibles pour l'écriture du réel, ce que l'on appelle souvent le nombre de chiffres significatifs, est limité; par ailleurs l'exposant, même s'il permet de considérer des nombres d'un très grand ordre de grandeur est aussi limité. Un exemple fera mieux comprendre ces deux constatations.

Supposons, pour simplifier l'explication, que les octets retenus pour coder la mantisse conduisent à trois chiffres significatifs seulement et que l'exposant puisse être de l'ordre de 30. Si je fournis les nombres 1234999,0 et 1234000,0 et que je demande de calculer la différence, le résultat au lieu d'être comme il se doit 999,0 sera égal à 0 : le système ne fera aucune différence entre les deux nombres 1234999.0 et 1234000.0. En effet

| | |
|---|---|
| 1234999.0 | 1234000.0 |
| devient | devient |
| $1.23 \times 10^6$ | $1.23 \times 10^6$ |

puisque seuls trois chiffres significatifs peuvent être retenus pour la mantisse et ces deux codages sont bien identiques.

Le codage des nombres réels, avec une taille limitée pour la mantisse, conduit donc à omettre lors du codage certains chiffres significatifs. Des erreurs surviennent alors à cause de ces oublis : c'est ce qu'on appelle des *erreurs de troncature*.

En résumé, il faut retenir que

> Les nombres réels codables et manipulables par l'ordinateur même s'ils peuvent être énormes (à cause de la taille permise pour l'exposant) ne sont codés qu'en retenant un certain nombre de chiffres significatifs (à cause de la mantisse). On constate alors des erreurs qui peuvent être importantes : les erreurs de troncature.

Voici à nouveau, pour les amateurs, un court programme en Pascal qui illustre les erreurs de troncature dues au nombre limité de chiffres significatifs

des réels, étant donné le nombre d'octets disponibles pour la mantisse :

```
var A,B : single;
```
(on définit deux casiers A et B destinés à recevoir des réels avec une mantisse codée sur 3 octets et un exposant sur 1 octet; ceci permet de retenir 7 chiffres significatifs)

```
    C,D : extended;
```
(on définit deux casiers C et D destinés à recevoir des réels avec une mantisse codée sur 8 octets et un exposant sur 2 octets; ceci permet de retenir 19 chiffres significatifs)

```
begin
A:=123456789999.0;
```
(on place dans A un réel comportant plus de chiffres que le nombre de chiffres significatifs pouvant être retenus étant donné la taille de la mantisse (3 octets))

```
B:=123456789000.0;
```
(on place dans B un autre nombre réel comportant lui aussi plus de chiffres que le nombre de chiffres significatifs pouvant être retenus étant donné la taille de la mantisse (3 octets))

```
C:=123456789999.0;
```
(on place dans C un réel comportant moins de chiffres que le nombre de chiffres significatifs pouvant être retenus étant donné la taille de la mantisse (8 octets))

```
D:=123456789000.0;
```
(on place dans D un autre nombre réel comportant moins de chiffres que le nombre de chiffres significatifs pouvant être retenus étant donné la taille de la mantisse (8 octets))

(Il faut noter que, pour nous, A-B donne le nombre 999, tout comme C-D)

```
writeln('La différence de 123456789999.0 et 123456789000.O est ',A-B);
```
(On demande ici l'affichage de la différence entre A et B; ce devrait être 999.0)

```
writeln('La différence de 123456789999.0 et 123456789000.0 est ',C-D);
```
(On demande ici l'affichage de la différence entre C et D; ce devrait être 999.0)

```
end.
```

Voici ce que donne l'exécution de ce petit programme :

```
La différence de 123456789999.0 et 123456789000.O est  0.00000000000000E+0000
```
(La différence de A et B qui devrait théoriquement donner 999 donne en réalité 0 : les derniers chiffres de A et B n'ont pu être retenus étant donné la taille de la mantisse (3 octets))

(Il faut savoir que l'écriture 0.0000...E+00... signifie 0.000... x $10^0$ soit 0 x 1 soit 0)

```
La différence de 123456789999.0 et 123456789000.0 est  9.99000000000000E+0002
```
(La différence de C et D donne bien 999, puisque le résultat signalé soit 9.9900...E+0002... signifie 9.990 x $10^2$ soit 9.99 x 100 soit 999. dans ce cas le résultat est bien correct puisque les deux nombres réels C et D ont pu être codés avec tous leurs chiffres étant donné la taille plus grande des mantisses permises.)

### *5.2.3.4 Le codage des dessins*

Nous nous limiterons ici au codage bitmap où le dessin est vu, après discrétisation, comme un ensemble de points, les pixels. De plus, la version de codage abordée ici est simplifiée, même si les principes illustrés sont ceux qui président à la plupart des codages bitmap.

Deux informations sont en tout cas nécessaires pour commencer : le nombre de pixels sur la hauteur de l'image et le nombre de pixels sur sa largeur. Chacune de ces quantités est par exemple codée sur 2 octets.

Il faut aussi indiquer en général le nombre de couleurs possible pour le dessin soit le nombre de couleurs possibles pour n'importe quel pixel) : 16, 256, … 16.777.216

?

Pourquoi deux octets pour coder le nombre de pixels sur la hauteur et la largeur et pas un octet ? ou quatre ?

Pourquoi ces nombres de couleurs possibles pour chaque pixel et pas 1000 ou 10.000 ?

Vient ensuite le codage de la succession de tous les pixels, avec pour chaque pixel simplement l'indication de la couleur qui le caractérise. Cette couleur peut être codée sur ½ octet (2 pixels codable sur un octet, ce qui conduit à 16 couleurs seulement), un octet (256 couleurs) ou trois octets (16.777.216 couleurs).

#### *5.2.3.5 Le codage des sons*

Nous savons déjà qu'un échantillonnage du signal électrique correspondant au son à traiter est effectué. Lorsque l'on souhaite que le codage du son ait une qualité analogue à celle des CD-Audio, cet échantillonnage se fait avec une fréquence de 44,1 KHz (le signal est analysé 44.100 fois par seconde). On utilise en général deux octets pour coder chacune des valeurs obtenues lors de l'échantillonnage et cela pour chaque canal (gauche et droit) dans le cas d'un son stéréo. C'est donc 2 x 44.100 x 2 octets, soit 176400 octets (environ 176 Ko), qui sont alors nécessaire pour coder une seconde de son, avec cette qualité. Le codage de son réclame donc des quantités énormes de mémoire, puisque pour coder une minute, plus de 10 millions d'octets (10 Mo) sont nécessaires. (Voir ci-dessous la définition du Ko et du Mo).

### *5.2.4 Comment les instructions des programmes exécutables sont codées en mémoire*

Nous savons déjà que la mémoire centrale comportera des données, codées comme décrit ci-dessus, mais également les programmes exécutables par le processeur. Sans entrer dans un détail qui ne sera possible qu'après la description de ce dernier, nous pouvons déjà signaler que ces instructions seront évidemment (comme tout le reste) codée en binaire sur un ou plusieurs octets.

### *5.2.5 Retour sur les unités de mesure*

Nous savons à présent que l'octet est l'espace sur lequel on code un caractère. Nous pouvons dès lors donner davantage de signification aux diverses unités utilisées pour mesurer la taille des dispositifs de mémorisation et la quantité d'informations.

#### *5.2.5.1 Les unités de mesure de taille mémoire*

Au delà de l'octet, on va trouver :

- Le KiloOctet (Ko) qui vaut 1024 octets.

  On pourrait s'étonner de ce chiffre : on aurait pu penser que le KiloOctet soit exactement 1000 octets. En réalité, on a retenu 1024 à cause du fait que l'ordinateur travaillant en binaire, 1024 est un nombre, proche de 1000 et bien "rond" lorsqu'on utilise le binaire. En effet 1024 est égal à $2^{10}$ ou encore, en binaire, 10000000000.

  Sachant que de plus, un octet est la place prise par le codage d'un caractère et en admettant qu'un page dactylographiée comporte environ 2000 caractères, on peut dire que 1 Ko est la taille nécessaire en mémoire pour coder l'ensemble des caractères d'une demi-page ou encore que 1 Ko équivaut environ à ½ page.

- Le MégaOctet (Mo) qui vaut 1024 Ko ($2^{10}$ Ko ou $2^{20}$ octets), soit environ 500 pages.
- Le GigaOctet (Go) qui vaut 1024 Mo ($2^{10}$ Mo ou $2^{20}$ Ko ou $2^{30}$ octets), soit environ 500.000 pages.

- Le TéraOctet (Mo) qui vaut 1024 Go, soit environ 500.000.000 pages.

#### *5.2.5.2 Le caractère formaliste des unités de mesure*

Ces diverses unités de mesure, dont les plus utilisées sont le Ko et le Mo servent également à mesurer la quantité d'informations. Mais tout se passe à nouveau de manière très formaliste puisque, par exemple, pour évaluer la quantité d'informations contenue dans un texte, on se contente de compter le nombre de caractères (octets) qui le composent.

Ainsi, en informatique, les deux textes suivants comportent chacun la même *quantité d'information*, à savoir 16 octets (en comptant les espaces, figurés ici par le symbole de soulignement)

| A_BC_DE_FG_HI_JK | 7_13_19_22_28_35 |
|---|---|

même si on vous assure que les caractères du second texte constituent à coup sûr les résultats du *prochain* tirage gagnant du Lotto ! Pour l'informaticien, ces deux documents comportent la même quantité d'information !

### *5.2.6 Le processeur et ses relations avec la mémoire centrale*

Après avoir examiné de plus près l'organisation de la mémoire centrale, il nous faut à présent décrire ce qui constitue le coeur du système, l'endroit où les instructions constituant les programmes exécutables sont "comprises" et exécutées, l'endroit où les données à manipuler sont traitées : le processeur.

#### *5.2.6.1 Principes d'architecture*

Ce sont les *principes d'organisation et de fonctionnement* du processeur et, surtout, de ses liens avec la mémoire centrale qui seront illustrés ici. Il ne faut donc pas y chercher une étude détaillée et technique des processeurs tels qu'ils se présentent aujourd'hui.

> C'est l'organisation qui est commune à tous les ordinateurs qui sera décrite ici. Les détails relatifs à telle ou telle machine qui complexifie la description seront largement passés sous silence.

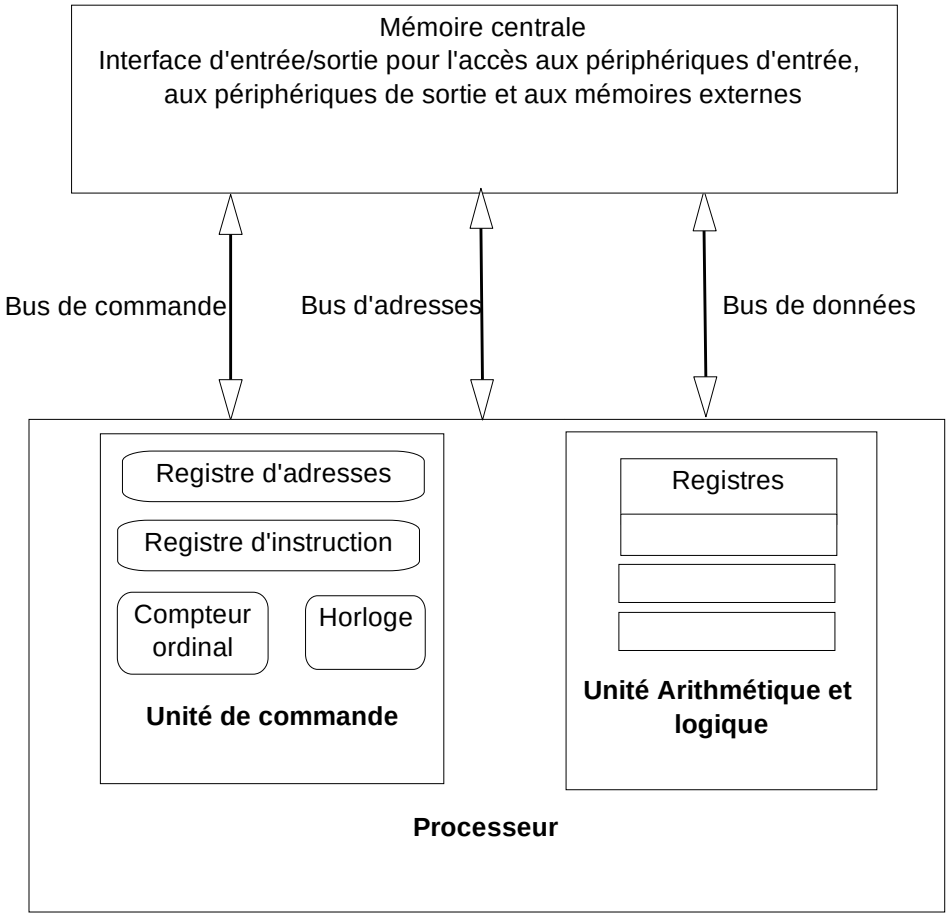



*Figure 5-12 : architecture générale d'une unité centrale*

Le processeur est, dans son principe, constitué de deux composantes : l'unité de commande et l'unité arithmétique et logique.

5.2.6.1.1 L'unité de commande ou de contrôle

C'est au sein de cette unité que résident :

- *L'horloge* interne du processeur : ce sont les battements de cette horloge qui cadencent le travail de toute l'unité centrale. Tout le fonctionnement du processeur, l'accès à la mémoire, etc., est synchronisé sur le rythme imposé par cette horloge. Sa fréquence, sur les processeurs actuels équipant les PC, est de l'ordre de 200 MHz (200 MégaHertz soit 200 millions de battements par seconde).
- Le *compteur ordinal* : c'est un registre particulier de l'unité de commande qui contient, comme on va le voir, l'adresse de l'instruction à exécuter par le processeur (instruction présente en mémoire centrale). On peut deviner qu'en général, à la fin de l'exécution d'une instruction, le compteur ordinal augmentera simplement de 1 pour qu'on passe à l'instruction dont l'adresse suit celle de l'instruction qui se termine.

J'ai signalé ci-dessus que ce compteur ordinal est un *registre*: Ce terme de registre désigne simplement une cellule de mémorisation présente au sein du processeur. C'est un peu comme une cellule mémoire à ceci près qu'elle fait partie du processeur et non de la mémoire centrale.

- Le *registre d'adresses* est également une cellule susceptible de contenir une information écrite en binaire. Cette information sera toujours l'adresse d'une cellule de

la mémoire centrale à laquelle le processeur souhaite accéder (pour en ramener le contenu ou pour y écrire un nouveau contenu). C'est en quelque sorte à travers ce registre que le processeur indique l'emplacement en mémoire de la cellule avec laquelle il souhaite travailler.

On devine déjà que la taille de ce registre d'adresses est un paramètre important de description et d'évaluation d'un processeur et donc d'une unité centrale. Si ce registre a une taille d'un octet, on pourra y placer seulement 256 adresses différentes (de 00000000 (0) à 11111111 (255)). Autrement dit, même si la mémoire centrale comportait des milliers de cellules, le processeur ne pourrait travailler qu'avec les cellules dont l'adresse pourrait être codée dans le registre d'adresses (ici entre 0 et 255). La taille du registre d'adresses donne donc la quantité de cellules auxquelles le processeur pourra accéder : on dit que cette taille détermine la *mémoire adressable* par le processeur.

Les premiers micro-ordinateurs (ceux de la décennie 70) possédaient un registre d'adresses d'une taille de 2 octets (16 bits), ce qui permettait d'adresser $2^{16}$ cellules de mémoire ou 65536 cellules dont le contenu était par ailleurs d'un octet, ce qui conduisait à une taille de mémoire adressable de 65536 octets ou 64 Ko.
Les PC apparus au début des années 80 possédaient un registre d'adresses d'une taille de 20 bits, permettant d'adresser $2^{20}$ cellules (d'un octet) soit 1 Mo.
Les (micro)-ordinateurs actuels ont un registre d'adresses de 4 octets permettant au processeur d'adresser $2^{32}$ soit 4294967296 cellules ou 4 Go.

- Le *registre d'instruction* est aussi une cellule de mémorisation interne à l'unité de commande du processeur. C'est dans ce registre que seront amenées, l'une après l'autre, les instructions à exécuter par le processeur pendant le déroulement de l'exécution d'un programme.

5.2.6.1.2 L'unité arithmétique et logique (UAL)

C'est cette composante du processeur qui va effectuer les opérations de nature arithmétique ou logique sur les données ramenées dans les registres présents dans cette même unité pour placer les résultats de ces opérations dans ces mêmes registres. Ces opérations seront commandées à chaque fois par une instruction présente dans le registre d'instruction.

Comme on le verra, les opérations possibles pour cette UAL consistent à additionner, soustraire, comparer,... les données présentes au sein des registres de cette unité. Le nombre des registres varie avec le type de processeur considéré : de quelques-uns à quelques dizaines. Leur taille est également variable : de un à quelques octets. On les désigne souvent comme des registres de données, puisqu'ils contiennent des données en provenance de la mémoire ou qui sont les résultats d'opérations effectuées par l'UAL et qui devront être réécrites en mémoire.

Le processeur est en liaison avec la mémoire centrale et avec les dispositifs qui accueillent les données de l'extérieur (en provenance des périphériques d'entrée ou des mémoires externes ou à destination des périphériques de sortie ou des mémoires externes).

Ces liaisons sur lesquelles vont circuler ce qui est échangé entre processeur et mémoire centrale sont des bus. Ils sont essentiellement de trois types :

5.2.6.1.3 Le bus d'adresses

C'est la liaison à travers laquelle les adresses de cellules mémoires auxquelles le processeur veut accéder (pour y lire ou y écrire) sont acheminées. Ce bus met donc en communication le registre d'adresses et la mémoire.

5.2.6.1.4 Le bus de données

C'est à travers cette liaison que les données ou les instructions contenues dans les cellules mémoires seront acheminées vers le processeur pour prendre place soit dans le registre d'instructions soit dans l'un ou l'autre des registres présents dans l'UAL.

La taille (= la largeur) du bus de données est un paramètre important d'une unité centrale. C'est en quelque sorte le nombre d'octets qui peuvent transiter, en un seul voyage, entre le processeur et la mémoire.
Dans les processeurs du début de la micro-informatique, un seul octet était acheminé à la fois le long du bus de données : on parlait de micro-processeur 8 bits, comme dans les premiers PC (équipés de processeurs Intel 8088). On a eu ensuite des bus de données acheminant 2 octets à la fois, comme dans les processeurs Intel 80286 (dits processeurs 16 bits); puis ceux permettant des échanges de 4 octets (32 bits) comme les Intel 80386.
Dans les processeurs Intel Pentium actuels, c'est 8 octets (64 bits) qui peuvent transiter côte à côte entre la mémoire et le processeur.

5.2.6.1.5 Le bus de commandes (ou de contrôle)

Nous en dirons fort peu de choses : il achemine des signaux de contrôle et de synchronisation entre les divers organes de l'unité centrale.

*5.2.6.2 Principes de fonctionnement*

Nous les illustrerons ci-dessous par un exemple élémentaire. Dès à présent, il est bon de signaler les éléments suivants :

- A chaque type de processeur est associé un ensemble d'instructions destinées à faire agir ce processeur. Ces instructions, codées en binaire sur un ou plusieurs octets, résident en mémoire. Leur succession constitue un programme "en langage machine" destiné à faire traiter des données (également présentes en mémoire).

Ces instructions correspondent à des actions dont le processeur est capable : aller chercher le contenu d'une ou plusieurs cellules mémoire pour les ramener dans des registres de l'UAL, additionner les contenus de certains de ces registres en plaçant le résultat dans un autre registre, recopier en mémoire le contenu d'un de ces registres, etc..

Le langage machine est donc l'ensemble des instructions propres à un processeur et qui vont le faire agir. Ce langage est propre à chaque type de processeur et il est donc abusif de parler de "langage machine" comme si les termes de ce langage étaient communs à toutes les machines. Il faudrait en réalité parler du langage de tel type de processeur, les instructions le composant étant différentes de celles du langage d'un autre type de processeur.

- Il est difficile de décrire de quoi se compose typiquement une instruction en langage machine, d'autant que, comme signalé, cela peut varier d'un processeur à un autre.

Le plus souvent cependant, une telle instruction comporte :

◊ L'indication de l'opération qui doit être effectuée par le processeur : lecture en mémoire, écriture en mémoire, addition, soustraction, comparaison,... La partie de l'instruction qui donne cette indication s'appelle le *code opératoire*.

◊ L'indication des données nécessaires pour mener à bien l'opération commandée : *adresses* des cellules mémoire dans lesquelles le processeur va lire la ou les données à ramener dans des registres, adresses des cellules mémoire où le processeur va écrire les contenus de certains de ces registres, etc..

- L'exécution des instructions suit en principe le cheminement suivant :

◊ Le *compteur ordinal* ayant une certaine valeur, cette valeur est recopiée dans le registre d'adresses et le processeur accède en lecture à la cellule de mémoire désignée par cette adresse. Le contenu de cette cellule, considéré comme une *instruction*, est ramené, à travers le bus de données, dans le registre d'instructions du processeur.

◊ L'instruction ramenée dans le registre d'instructions est décodée et exécutée. Ceci nécessite le plus souvent de nouveaux *accès à la mémoire* pour y lire ou y écrire des données. Ces accès se font, comme toujours, en plaçant les adresses (qui étaient mentionnées dans l'instruction qui est en train d'être exécutée) dans le registre d'adresses, en accédant aux cellules d'adresses indiquées et en transférant dans ces cellules (écriture) ou à partir de ces cellules (lectures) les données à traiter, à travers le bus de données.

◊ Une fois l'instruction exécutée, le compteur ordinal augmente généralement de 1 et *tout recommence*.

### *5.2.7 Un exemple simplifié d'unité centrale, de langage machine et d'exécution d'un programme*

Comme un court exemple vaut mieux qu'un long discours, nous allons décrire un modèle extrêmement simplifié d'unité centrale et un langage machine correspondant au modèle simplifié du processeur de cette unité centrale. Nous décrirons ensuite un traitement que nous souhaiterions faire exécuter par cette unité centrale, nous écrirons le programme en langage machine correspondant à ce traitement (= les indications destinées au processeur pour mener à bien ce traitement) et nous suivrons pas à pas l'exécution de ce programme.

☞ Il est essentiel d'avoir compris qu'il ne s'agit ici que d'un exemple, extrêmement simplifié, mais qui illustre les principes de fonctionnement de tout ordinateur. Tout ordinateur fonctionne *un peu comme cela*, mais pas exactement comme cela.

#### *5.2.7.1 Un modèle simplifié d'unité centrale*

- La mémoire centrale de cette unité centrale (voir figure ci-dessous) est constituée de seulement 32 cellules d'un octet. Les adresses de ces cellules vont de 0 à 31 (de 00000 à 11111 en codage binaire). Cette taille n'est pas dus au hasard : elle résulte de la taille du registre et du bus d'adresses qui est seulement de 5 bits.
- Le compteur ordinal et le registre d'adresses du processeur sont constitués de 5 bits.
- Le registre d'instructions peut accueillir un octet.
- L'UAL ne comporte qu'un seul registre pour accueillir des données; sa taille est d'un octet. Un second registre est présent dans l'UAL, le code condition (2 bits) dont le contenu ne sera modifié que lors de l'exécution d'une instruction de comparaison.
- Le bus de données permet d'acheminer un octet entre la mémoire et le processeur ou vice-versa.

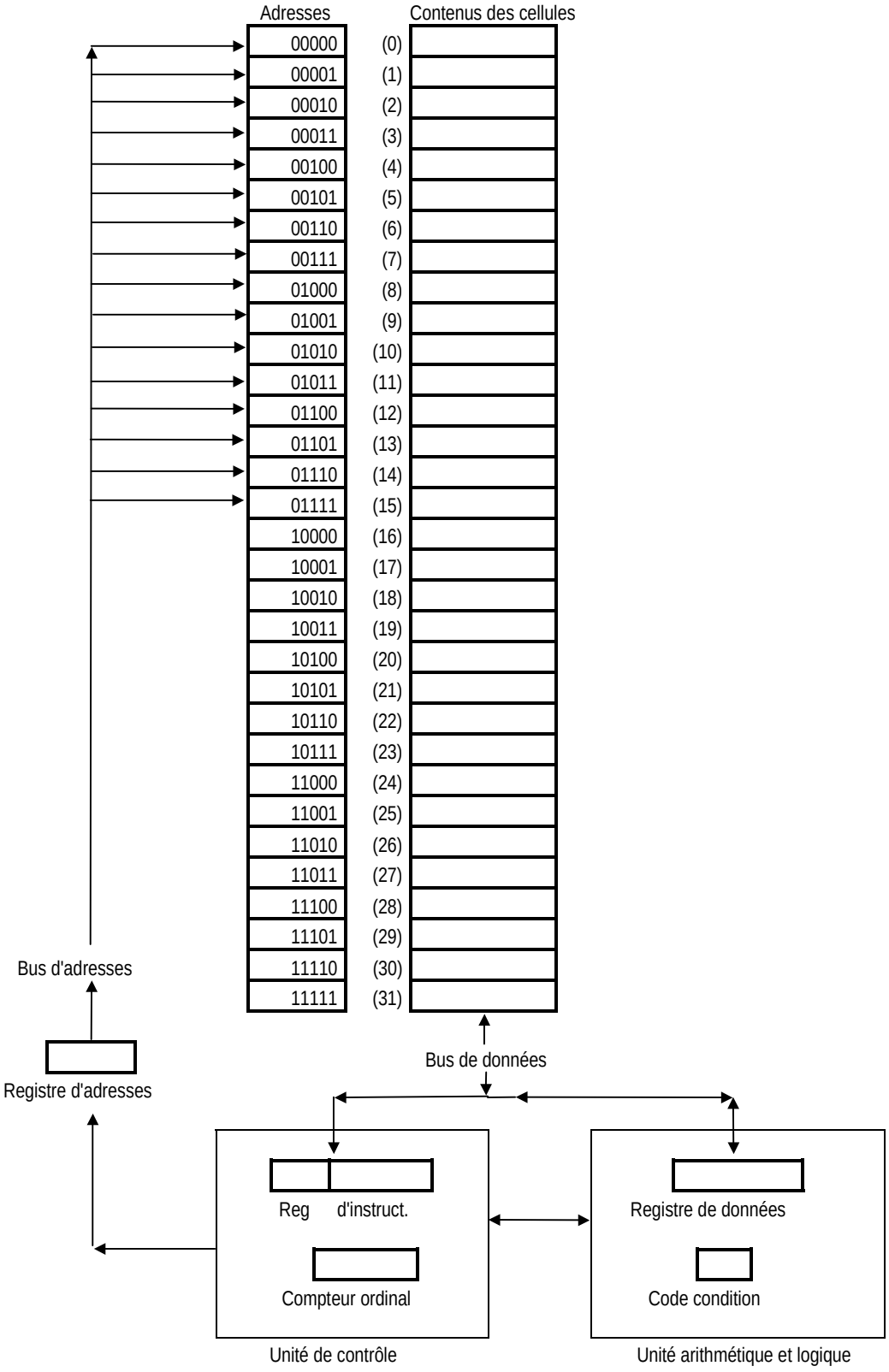



*Figure 5-13 : modèle simplifié d'unité centrale*

*5.2.7.2 Un modèle simplifié de langage machine adapté au modèle simplifié d'unité centrale*

Chaque instruction à destination du processeur simplifié est codée sur un octet et a la structure suivante :

- Les trois premiers bits de l'octet codant l'instruction constituent le code opératoire, c'est à dire l'indication de l'opération à effectuer par le processeur. Le code opératoire étant codé sur 3 bits, on a donc la possibilité de désigner seulement 8 ($2^3$) opérations différentes.
- Les 5 bits suivants de l'octet codant une instruction désignent en général une adresse mémoire (pour y lire ou y écrire une donnée).

Les 8 instructions disponibles sont les suivantes :

1. L'instruction de lecture : son code opératoire est 000, la seconde partie (=les 5 bits suivants) est une adresse. Son effet est prendre copie du contenu de la cellule dont l'adresse est indiquée et de l'amener à travers le bus de données dans le registre de données de l'UAL (dont elle remplace l'ancien contenu). On notera que le contenu de la cellule consultée n'est pas modifié.
2. L'instruction d'écriture en mémoire : son code opératoire est 001, la seconde partie (=les 5 bits suivants) est l'adresse d'une cellule mémoire. Son effet est copier dans la cellule dont l'adresse est indiquée le contenu du registre de données de l'UAL. L'ancien contenu de cette cellule est perdu; par contre, le registre de l'UAL garde son contenu inchangé.
3. L'instruction d'addition : son code opératoire est 010, la seconde partie (=les 5 bits suivants) est une adresse. Son effet est prendre copie du contenu de la cellule dont l'adresse est indiquée, de l'amener à travers le bus de données vers l'UAL et de l'additionner au contenu actuel de ce registre de données.

   Dans cette version extrêmement simplifiée, on ne se préoccupe pas des débordements qui peuvent se produire lorsque l'addition conduit à un résultat qui ne tient plus sur un seul octet. Par exemple, si le registre de données contenait 10000000 (128 en décimal) et que cette instruction d'addition ramène comme contenu de cellule l'octet 10000011) (131 en décimal), la somme 100000011 (259) ne peut pas tenir sur le seul octet constituant le registre de données.

   Dans les vrais processeurs, cette éventualité est bien entendu traitée.
4. L'instruction de soustraction : son code opératoire est 011, la seconde partie (=les 5 bits suivants) est une adresse. Son effet est prendre copie du contenu de la cellule dont l'adresse est indiquée, de l'amener à travers le bus de données vers l'UAL et de la soustraire du contenu actuel de ce registre de données. Comme d'habitude, le contenu de la cellule consultée reste inchangé, et l'ancien contenu du registre de donnée est perdu.
5. L'instruction de comparaison : son code opératoire est 100, la seconde partie (=les 5 bits suivants) est une adresse. Son effet est prendre copie du contenu de la cellule dont l'adresse est indiquée, de l'amener à travers le bus de données vers l'UAL et de la comparer au contenu actuel du registre de données. Le résultat de cette comparaison va se retrouver dans le registre code condition de l'UAL

   ◊ si le registre de donnée de l'UAL est égal à la donnée amenée de la mémoire par cette instruction de comparaison, le code condition prend la valeur 00 (l'ancienne valeur du code condition étant perdue)

- ◊ si le registre de donnée de l'UAL est strictement supérieur à la donnée amenée de la mémoire par cette instruction de comparaison, le code condition prend la valeur 10 (l'ancienne valeur du code condition étant perdue)
- ◊ si le registre de donnée de l'UAL est strictement inférieur à la donnée amenée de la mémoire par cette instruction de comparaison, le code condition prend la valeur 01 (l'ancienne valeur du code condition étant perdue)

Dans tous les cas, le registre de donnée de l'UAL reste inchangé à la suite de cette opération de comparaison.

6. L'instruction de branchement conditionnel brancher si le code condition vaut 10; son code opératoire est 101.

   Avec ce premier exemple de branchement, on entre dans un autre type d'instruction. Il est important d'en saisir l'utilité et la portée, d'autant que ce genre d'instruction est sans doute plus malaisé à saisir.

   L'exécution des diverses instructions présentées jusqu'ici provoquait un accès à la mémoire centrale (pour y écrire ou y lire une donnée). Ce ne sera plus le cas des instructions de branchement. Ces instructions vont *éventuellement* provoquer un changement du compteur ordinal. Ainsi l'instruction de branchement conditionnel brancher si le code condition vaut 10 aura l'effet suivant :

   - ◊ Si le code condition vaut 10, le compteur ordinal prend comme valeur l'adresse qui accompagnait l'instruction de branchement conditionnel. Ainsi, si l'instruction était 10101100 (code condition 101 : branchement si code condition vaut 10, adresse 01100) le compteur ordinal prendra cette valeur 01100, à condition que le code condition soit égal à 10.
   - ◊ Si le code condition ne vaut pas 10, le compteur ordinal augmentera simplement de 1, comme à la fin de l'exécution de n'importe quelle instruction.

   En d'autres termes, si une instruction de comparaison précédente avait fait passer le code condition à 10 (ce qui signifie que lors de cette comparaison le registre de données était strictement supérieur à la donnée ramenée) le compteur ordinal sautera à la valeur écrite après le code opératoire 101; sinon le compteur ordinal augmentera simplement de 1.

   Si la situation était la suivante (code condition égal à 10, compteur ordinal égal à 00010) lorsque l'instruction de branchement a été ramenée :

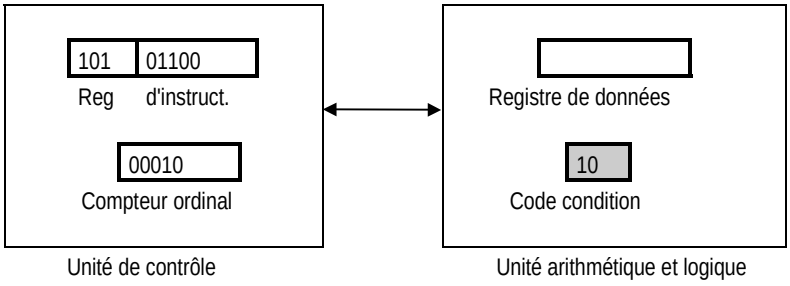



*Figure 5-14 : le branchement conditionnel (1)*

   l'exécution va provoquer un saut du compteur ordinal qui passera à 01100 :

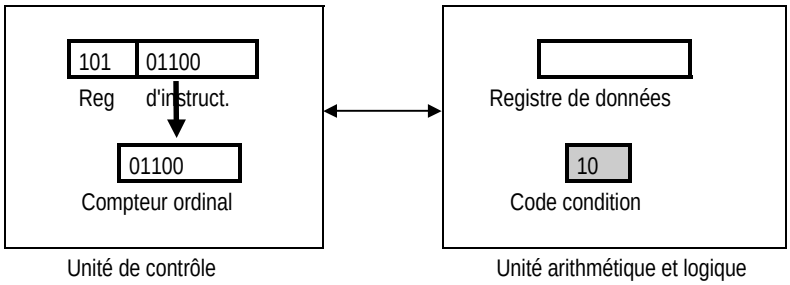



*Figure 5-15 : le branchement conditionnel (2)*

par contre, si on avait un code condition différent de 10

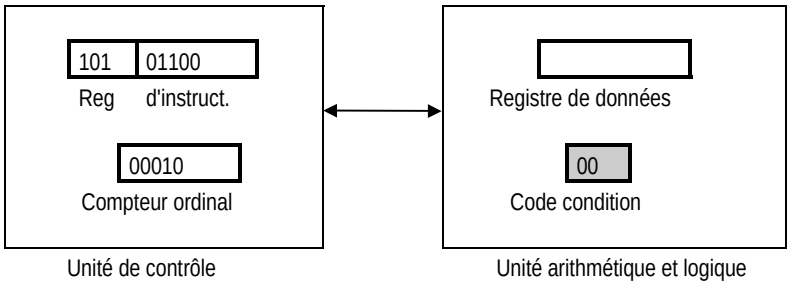



*Figure 5-16 : le branchement conditionnel (3)*

on passera à une situation ou le compteur ordinal augmente simplement de 1 (passant de 00010 à 00011.

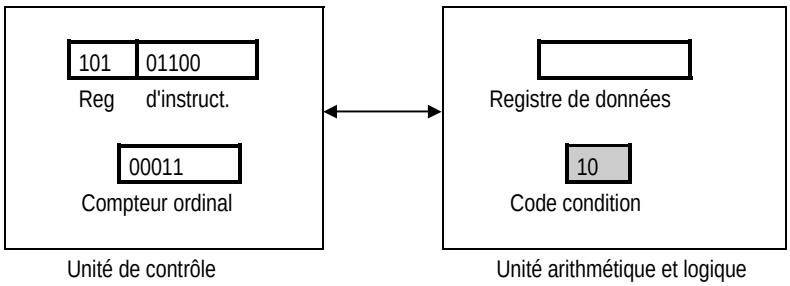



*Figure 5-17 : le branchement conditionnel (4)*

7. L'instruction de branchement conditionnel brancher si le code condition vaut 00; son code opératoire est 110. Cette instruction aura l'effet suivant :

   ◊ Si le code condition vaut 00, le compteur ordinal prend comme valeur l'adresse qui accompagnait l'instruction de branchement conditionnel. Ainsi, si l'instruction était 11001100 (code condition 110 : branchement si code condition vaut 00, adresse 01100) le compteur ordinal prendra cette valeur 01100, à condition que le code condition soit égal à 00.

   ◊ Si le code condition ne vaut pas 00, le compteur ordinal augmentera simplement de 1, comme à la fin de l'exécution de n'importe quelle instruction.

8. L'instruction de branchement pure et simple. Son code opératoire est 111. Elle a pour effet de recopier dans le compteur ordinal les 5 bits accompagnant le code opératoire.

On aura compris que le rôle des instructions de branchement est de rompre la séquentialité de l'exécution des instructions constituant un programme. En effet, pour toutes les instructions habituelles, le compteur ordinal augmente de 1 à la fin de leur exécution; comme le compteur ordinal désigne l'adresse de l'instruction qui sera ensuite exécutée, on constate que les instructions seront exécutées dans l'ordre où elles se succèdent en mémoire centrale.

Avec le branchement, le compteur ordinal, plutôt que d'augmenter de 1 va pouvoir prendre une valeur définie. L'instruction qui sera exécutée après ce branchement pourra donc être n'importe laquelle et pas obligatoirement celle qui suivait en mémoire centrale.

En résumé, le jeu des instructions disponibles pour ce modèle simplifié d'unité centrale est donc le suivant :

| **Code Opéra-toire** | **Adresse** | **Opération commandée** | **Action produite** |
|---|---|---|---|
| 000 | ????? | CHARGER | copier le contenu de la cellule dont l'adresse suit dans le registre de données de lUAL (dont l'ancien contenu est perdu) |
| 001 | ????? | SAUVER | écrire le contenu du registre de données dans la cellule dont l'adresse suit (l'ancien contenu étant alors perdu) |
| 010 | ????? | ADDITIONNER | additionner le contenu de la cellule dont l'adresse suit au registre de données (dont l'ancien contenu est perdu) |
| 011 | ????? | SOUSTRAIRE | soustraire le contenu de la cellule dont l'adresse suit du registre de données (dont l'ancien contenu est perdu) |
| 100 | ????? | COMPARER | le contenu du registre de donnée au contenu de la cellule dont l'adresse suit et<br>* placer 00 dans le code condition si le registre = le contenu de la cellule<br>* placer 10 dans le code condition si le registre > le contenu de la cellule<br>* placer 01 dans le code condition si le registre < le contenu de la cellule<br>Cette comparaison ne modifie pas le registre de données. |
| 101 | ????? | BRANCHER SI > | placer dans le compteur ordinal l'adresse mentionnée *si le code condition vaut 10* (registre > cellule lors de la dernière instruction de comparaison); sinon incrémenter de 1 ce compteur (comme à la fin d'une instruction normale) |
| 110 | ????? | BRANCHER SI = | placer dans le compteur ordinal l'adresse mentionnée *si le code condition vaut 00* (registre = cellule lors de la dernière instruction de comparaison); sinon incrémenter de 1 ce compteur (comme à la fin d'une instruction normale) |
| 111 | ????? | BRANCHER | sauter dans tous les cas à l'instruction dont l'adresse suit |

*5.2.7.3 Un problème et sa traduction en langage machine (pour le modèle simplifié)*

On souhaite disposer d'un programme qui sur base du contenu des cellules d'adresses 10, 11 et 12, contenus que nous désignerons par a, b et c, fasse placer dans la cellule d'adresse 13, la plus grande des deux quantités a+b ou c. Autrement dit, à la fin de l'exécution de ce programme (qui reste à écrire), la cellule d'adresse 13 contiendra max(a+b, c), a, b et c désignant les contenus des cellules d'adresses 10, 11 et 12.

Les seules instructions à notre disposition sont les 8 décrites ci-dessus. Les opérations commandées seront les suivantes :

1 amener a, contenu de la cellule 01010 (10), dans le registre de données de l'UAL
   - le registre de données contient a, le code condition est inconnu

2 additionner b, contenu de la cellule 01011 (11) au registre de données de l'UAL
   - le registre de données contient a+b, le code condition est inconnu

3 comparer le contenu du registre de données au contenu c de la cellule 01100 (12)
   - le registre de données contient toujours a+b,

- si a+b (contenu du registre de données) est égal à c (contenu de la cellule 12), le code condition vaut 00
- si a+b (contenu du registre de données) est strictement supérieur à c (contenu de la cellule 12), le code condition vaut 10
- si a+b (contenu du registre de données) est strictement inférieur à c (contenu de la cellule 12), le code condition vaut 01

A ce stade, deux chemins peuvent être suivis :

- si a+b (contenu du registre de données) est strictement supérieur à c (contenu de la cellule 12), c'est à dire si le code condition vaut 10, il suffit alors d'écrire le contenu a+b du registre de données dans la cellule 13; cette cellule 13 contiendra alors a+b qui sera bien la plus grande des deux quantités a+b et c
- dans les autres cas, (a+b est inférieur ou égal à c), c'est c qui doit prendre place dans la cellule 13; dans ce cas on ramènera donc c dans le registre de données puis on réécrira ce registre de données dans la cellule 13

Il faut donc ici placer une instruction de branchement conditionnel qui fera sauter le compteur ordinal à l'adresse de la dernière instruction du programme lorsque le code condition vaut 10 (ce qui signifie que a+b>c). Si le code condition n'est pas 10 le compteur ordinal augmentera de 1 et fera passer à l'instruction suivante.

4 brancher à l'instruction qui porte le numéro 6 si le code condition vaut 10; sinon continuer normalement

5 amener c, contenu de la cellule 12, dans le registre de données de l'UAL

- le registre de données contient à présent c; on est dans le cas où a+b est inférieur ou égal à c

6 écrire le contenu du registre de données de l'UAL dans la cellule 13

- le registre de données contient a+b (si on avait effectué le branchement) ou c (si on avait poursuivi normalement); on retrouve donc soit a+b, soit c dans la cellule 13.

Sous une forme plus schématique, on peut écrire :

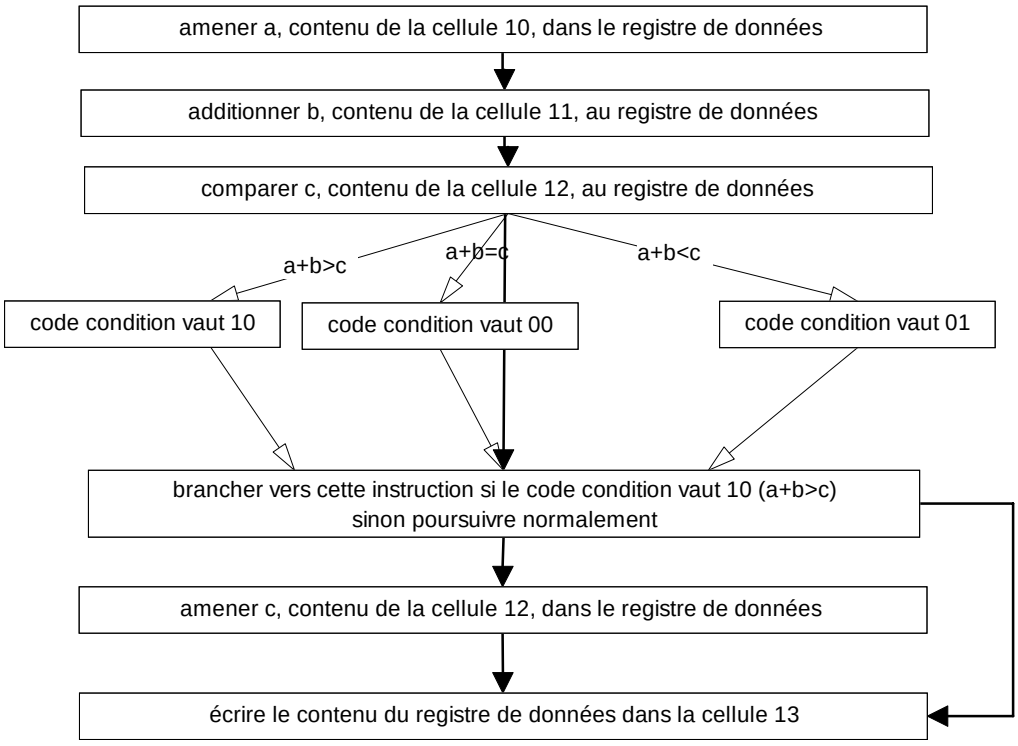



*Figure 5-18 : schéma du programme de calcul de max(a+b,c)*

Il reste à écrire explicitement les instructions correspondantes :

000 01010 (charger la cellule 01010 (10) dans le registre de données)

| | |
|---|---|
| 010 01011 | (additionner la cellule 01011 (11) au registre de données) |
| 100 01100 | (comparer le registre de données à la cellule 01100 (12)) |
| 101 ????? | (brancher à la dernière instruction si le code condition vaut 10 (registre>cellule)) |
| 000 01100 | (charger la cellule 01100 (12) dans le registre de données) |
| 001 01101 | (sauver le registre de données dans la cellule 01101 (13)) |

Comme on le voit, il est impossible de placer l'adresse pour le branchement tant qu'on ne sait pas explicitement où les instructions du programme vont prendre place en mémoire. En effet, c'est l'adresse explicite de la dernière instruction qui doit prendre la place des ?????

Si le programme est rangé à partir de la cellule d'adresse 00000, comme ci-dessous

| Adresses | | Contenus des cellules | |
|---|---|---|---|
| 00000 | (0) | 00001010 | charger la cellule 01010 (10) dans le registre |
| 00001 | (1) | 01001011 | additionner la cellule 01011 (11) au registre |
| 00010 | (2) | 10001100 | comparer le registre à la cellule 01100 (12) |
| 00011 | (3) | 101????? | brancher si code cond = 10 (registre>cellule) |
| 00100 | (4) | 00001100 | charger la cellule 01100 (12) dans le registre |
| 00101 | (5) | 00101101 | sauver le registre dans la cellule 01101 (13) |

*Figure 5-19 : rangement du programme en mémoire (1)*

alors les ????? deviennent 00101 (5) puisque la dernière instruction (celle où il faut sauter) est rangée dans la cellule d'adresse 00101.

Si le programme était rangé à partir de la cellule d'adresse 00010 (2), comme ci-dessous

| Adresses | | Contenus des cellules | |
|---|---|---|---|
| 00000 | (0) | | |
| 00001 | (1) | | |
| 00010 | (2) | 00001010 | charger la cellule 01010 (10) dans le registre |
| 00011 | (3) | 01001011 | additionner la cellule 01011 (11) au registre |
| 00100 | (4) | 10001100 | comparer le registre à la cellule 01100 (12) |
| 00101 | (5) | 101????? | brancher si code cond = 10 (registre>cellule) |
| 00110 | (6) | 00001100 | charger la cellule 01100 (12) dans le registre |
| 00111 | (7) | 00101101 | sauver le registre dans la cellule 01101 (13) |

*Figure 5-20 : rangement du programme en mémoire (2)*

alors les ????? deviendraient 00111 (7) puisque la dernière instruction (celle où il faut sauter) est rangée dans la cellule d'adresse 00101.

Dans ce qui suit, on rangera le programme à partir de la cellule 00000. Notons au passage qu'il était impératif que le rangement du programme en mémoire préserve la zone constituée des cellules d'adresses 10, 11 et 12 qui contenaient les données a, b et c.

☞ Il est évidemment important de savoir comment un programme est rangé en mémoire. Qui décide de l'endroit où le programme sera rangé ? qui range les instructions et les données en mémoire ?

Nous pouvons à présent suivre l'exécution du programme rangé en mémoire comme indiqué ci-dessous. Il fera placer dans la cellule 13 le maximum des quantités a+b (cellules 10 et 11) et c (cellule 12).

Il est impératif de suivre pas à pas l'exécution du programme concerné en comprenant que cette exécution suit un schéma immuable et parfaitement automatique.

La seule difficulté concerne l'instruction de branchement et l'effet qu'elle va avoir sur le déroulement du programme.

On a initialement :

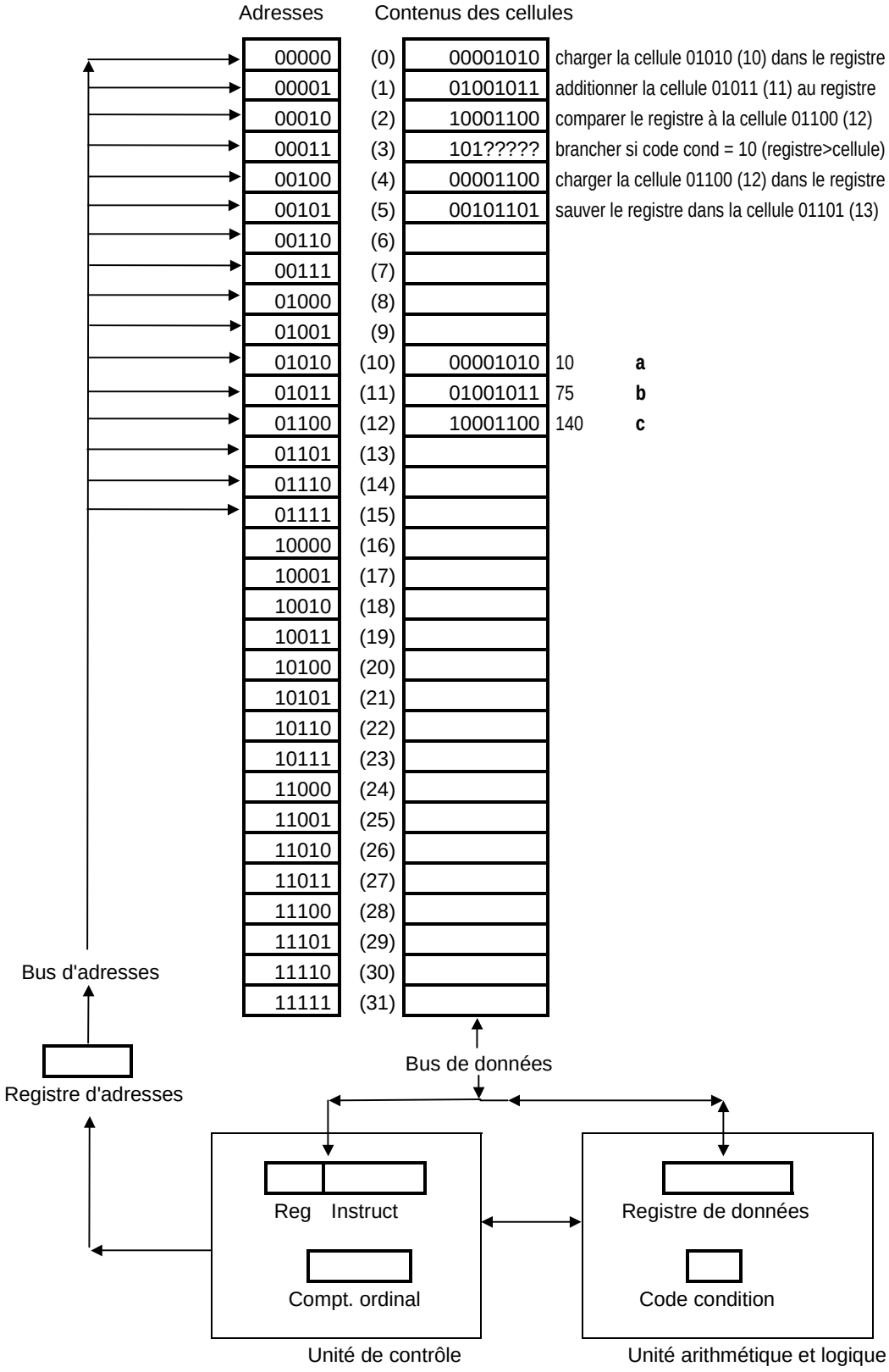



*Figure 5-21 : programme de calcul de Max(a+b,c)*

On notera au passage que les contenus des cellules 0 et 10 sont identiques, comme celles de 1 et 11 ou 2 et 12. Nous y reviendrons.

Ce qui va provoquer le démarrage de l'exécution est le fait que le compteur ordinal, prend la valeur 00000 suite à l'exécution de l'instruction qui précédait.

?

Quelle pourrait être cette instruction précédemment exécutée ? Comment le programme dont elle fait partie se trouve-t-il en mémoire ?

### *5.2.7.4 L'exécution du programme de calcul de Max(A+B,C)*

#### 5.2.7.4.1 Première phase du premier cycle : chargement de la première instruction dans le registre instruction.

[1] le compteur ordinal, dans lequel l'adresse 00000 de la première instruction du programme avait été placée est recopié dans le registre d'adresses

[2] l'instruction 00001010 contenue dans la cellule 00000 est sélectionnée et

[3] chargée dans le registre d'instructions

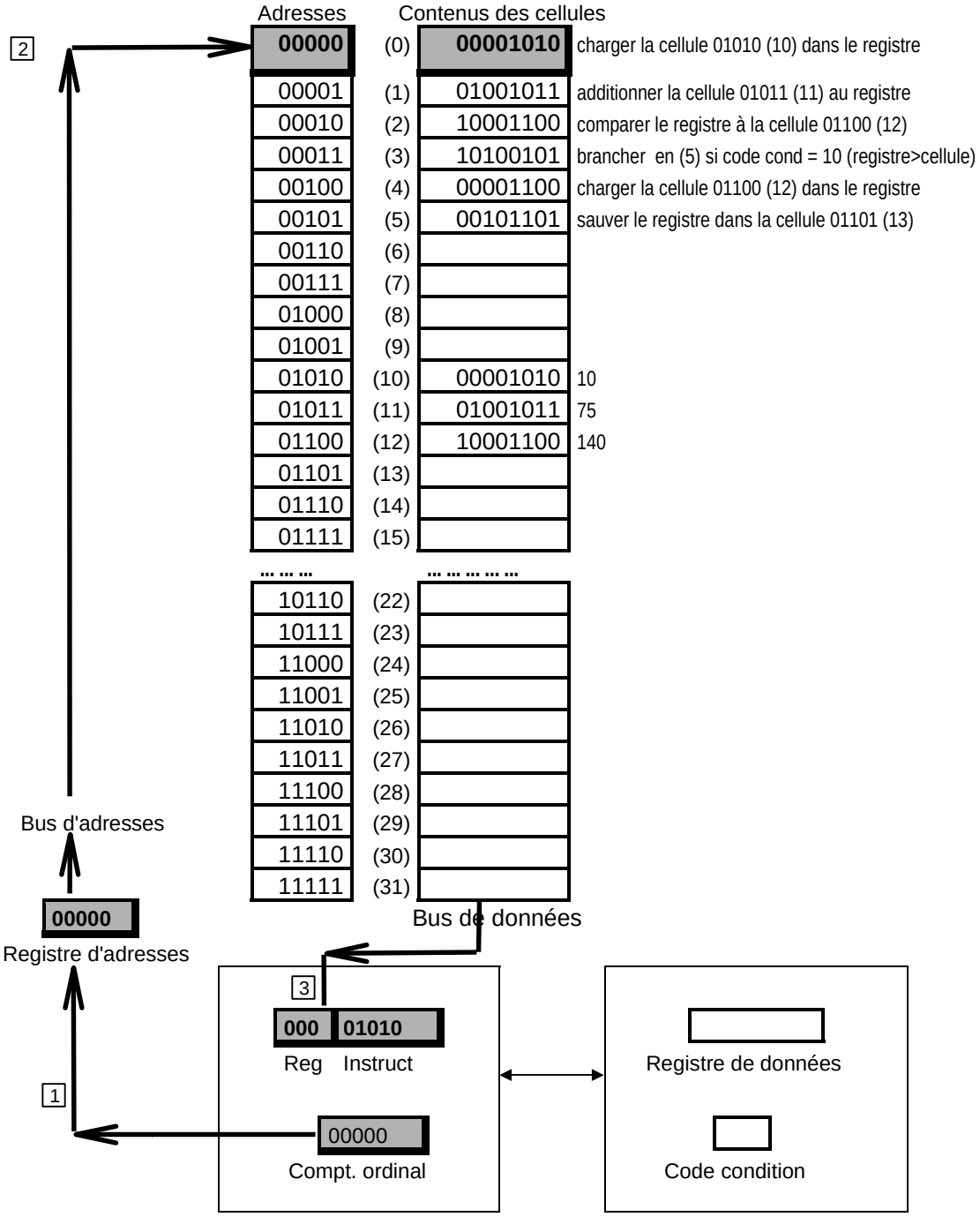



*Figure 5-22 : première phase du premier cycle*

### 5.2.7.4.2 Deuxième phase du premier cycle :décodage et exécution de la première instruction

4 l'adresse 01010 (10) mentionnée par cette instruction CHARGER est placée dans le registre d'adresses

5 la cellule d'adresse 01010 est sélectionnée et son contenu 00001010 (10)

6 est amené dans le registre de données.

7 Le compteur ordinal est augmenté de 1

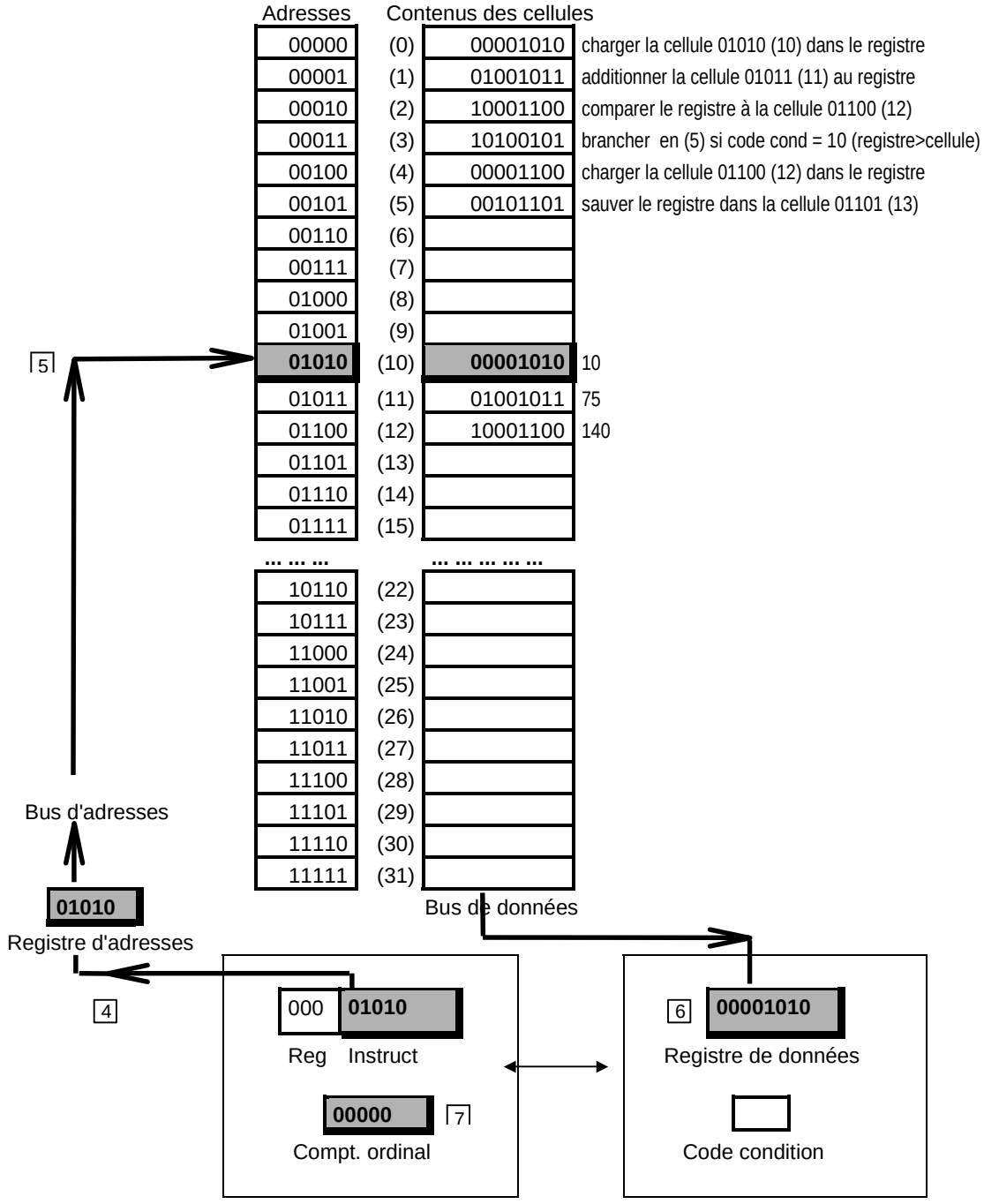



*Figure 5-23 : 2ème phase du premier cycle*

### 5.2.7.4.3 Première phase du deuxième cycle : Chargement de la deuxième instruction dans le registre instruction

[8] le contenu du compteur ordinal, soit 00001, est transféré dans le registre d'adresses

[9] l'instruction 01001011 contenue dans la cellule d'adresse 00001 (1) est sélectionnée et

[10] chargée dans le registre d'instruction

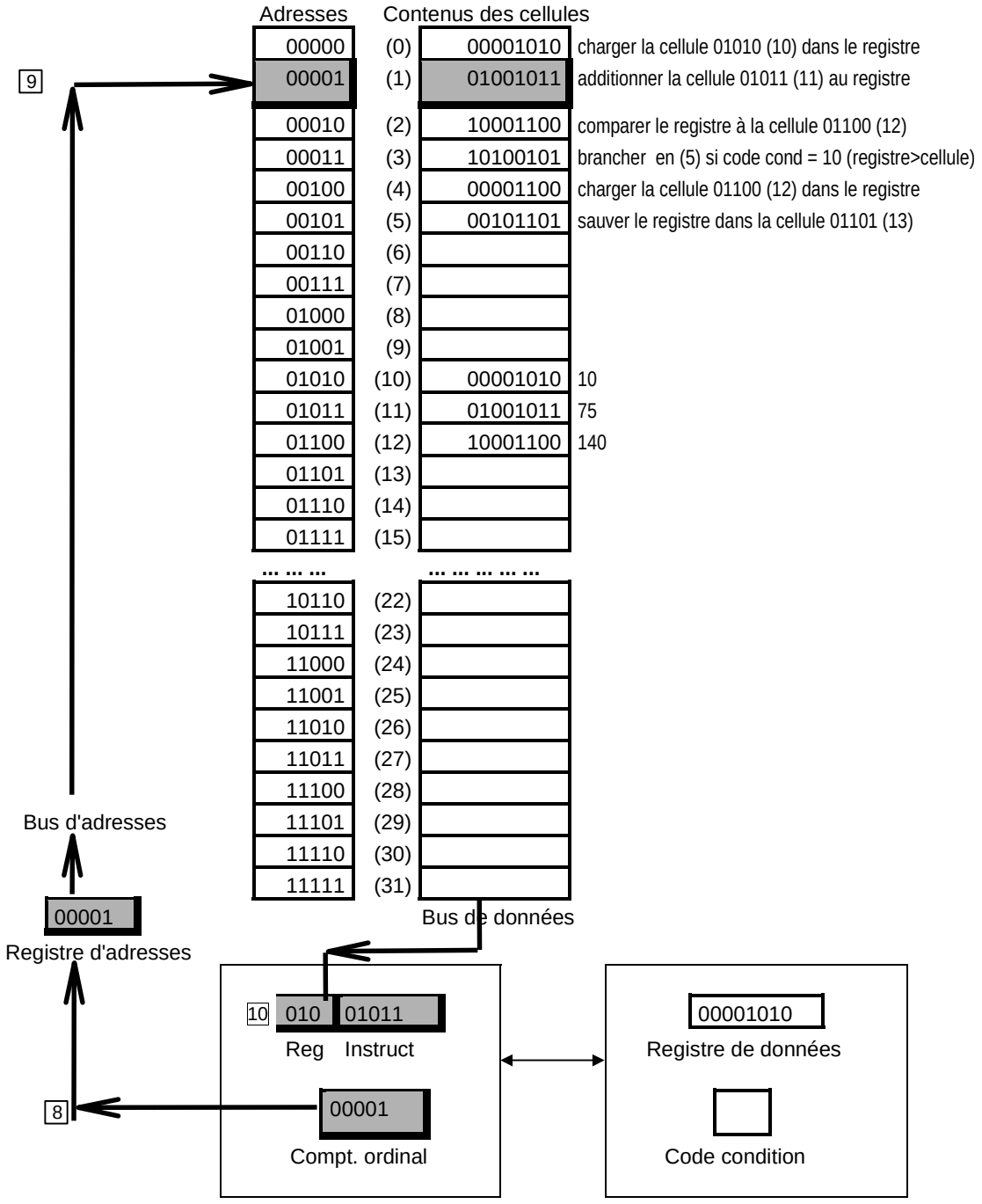



*Figure 5-24 : première phase du deuxième cycle*

#### 5.2.7.4.4 Deuxième phase du deuxième cycle : décodage et exécution de la deuxième instruction

11 l'adresse 01011 (11) mentionnée par cette instruction ADDITIONNER est placée dans le registre d'adresses

12 la cellule d'adresse 01011 est sélectionnée et son contenu 01001011 (75)

13 est *additionné* au contenu du registre de données.

14 Le compteur ordinal est augmenté de 1.

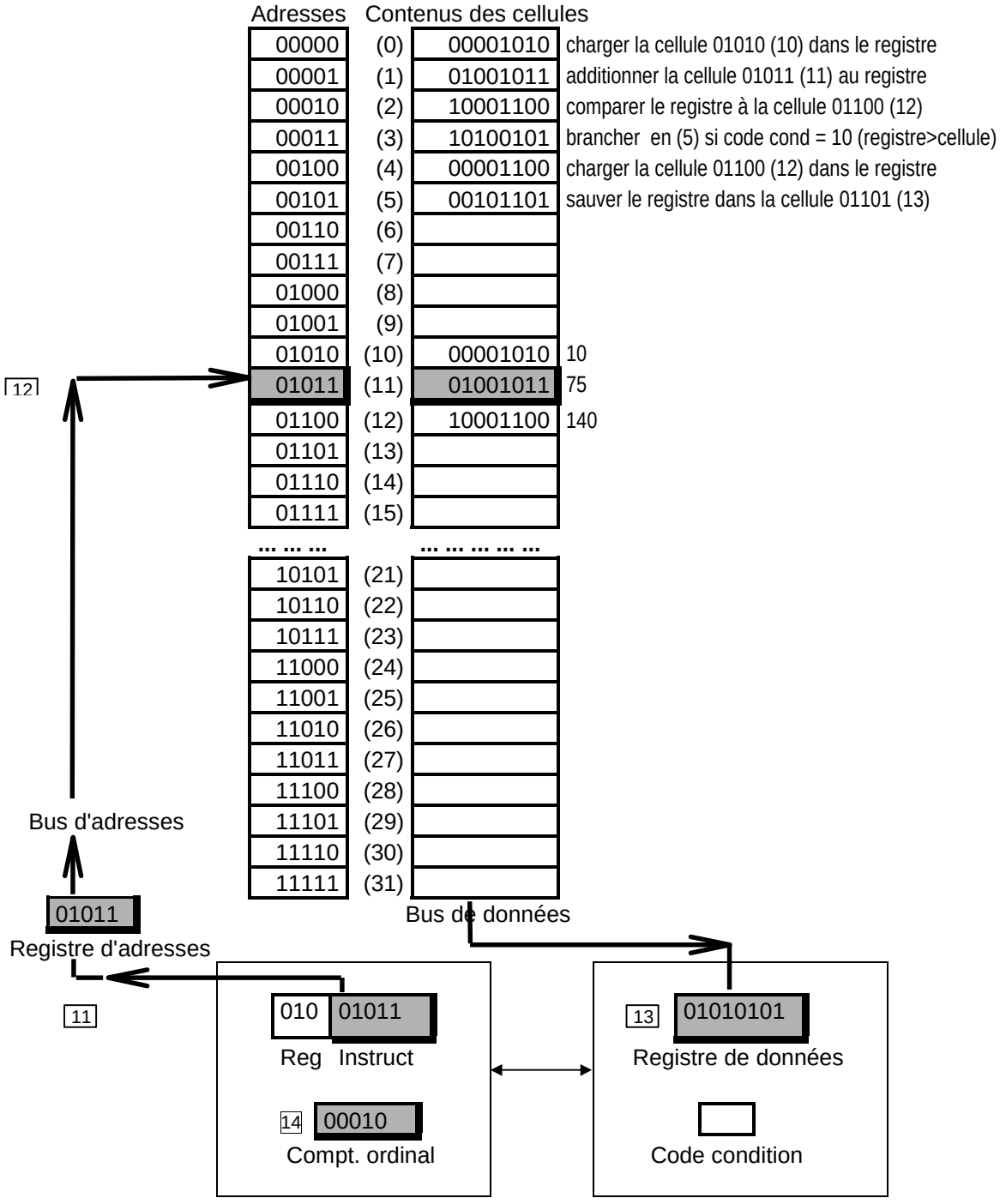



*Figure 5-25 : deuxième phase du deuxième cycle*

### 5.2.7.4.5 Première phase du troisième cycle : chargement de la troisième instruction dans le registre instruction

15 le compteur ordinal voit son contenu 00010 transféré dans le registre d'adresses

16 l'instruction 10001100 contenue dans la cellule d'adresse 00010 (2) est sélectionnée et

17 chargée dans le registre d'instruction

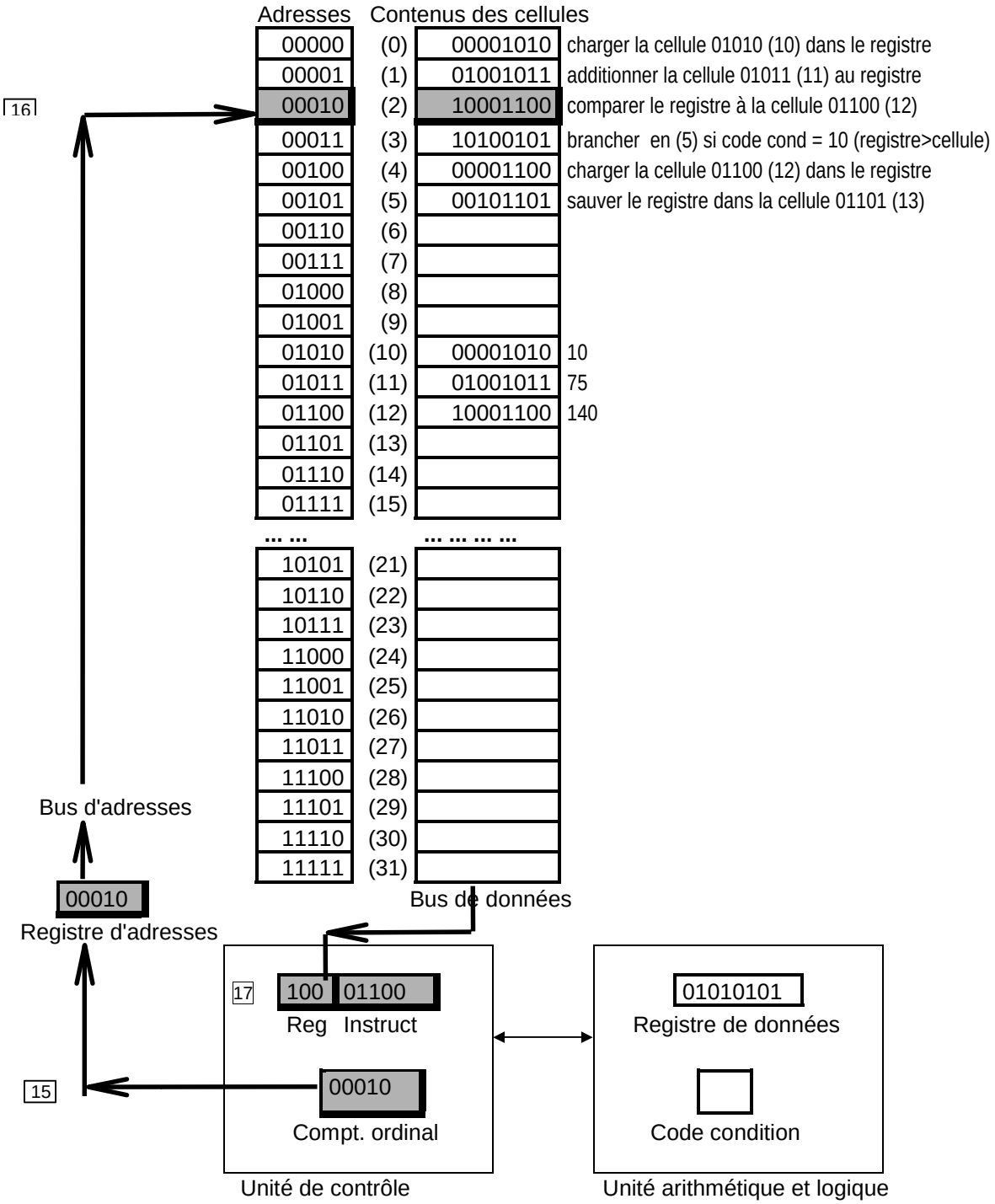



*Figure 5-26 : première phase du troisième cycle*

### 5.2.7.4.6 Deuxième phase du troisième cycle : décodage et exécution de la deuxième instruction

18 l'adresse 01100 (12) mentionnée par cette instruction COMPARER est placée dans le registre d'adresses

19 la cellule d'adresse 01100 est sélectionnée et son contenu 10001100 (140)

20 est *comparé* au contenu du registre de données : comme le contenu 01010101 (85) du registre est inférieur au contenu de la cellule (140), le code condition passe à 01 (de toute manière, le contenu du registre de données reste inchangé)

21 Le compteur ordinal est augmenté de 1.

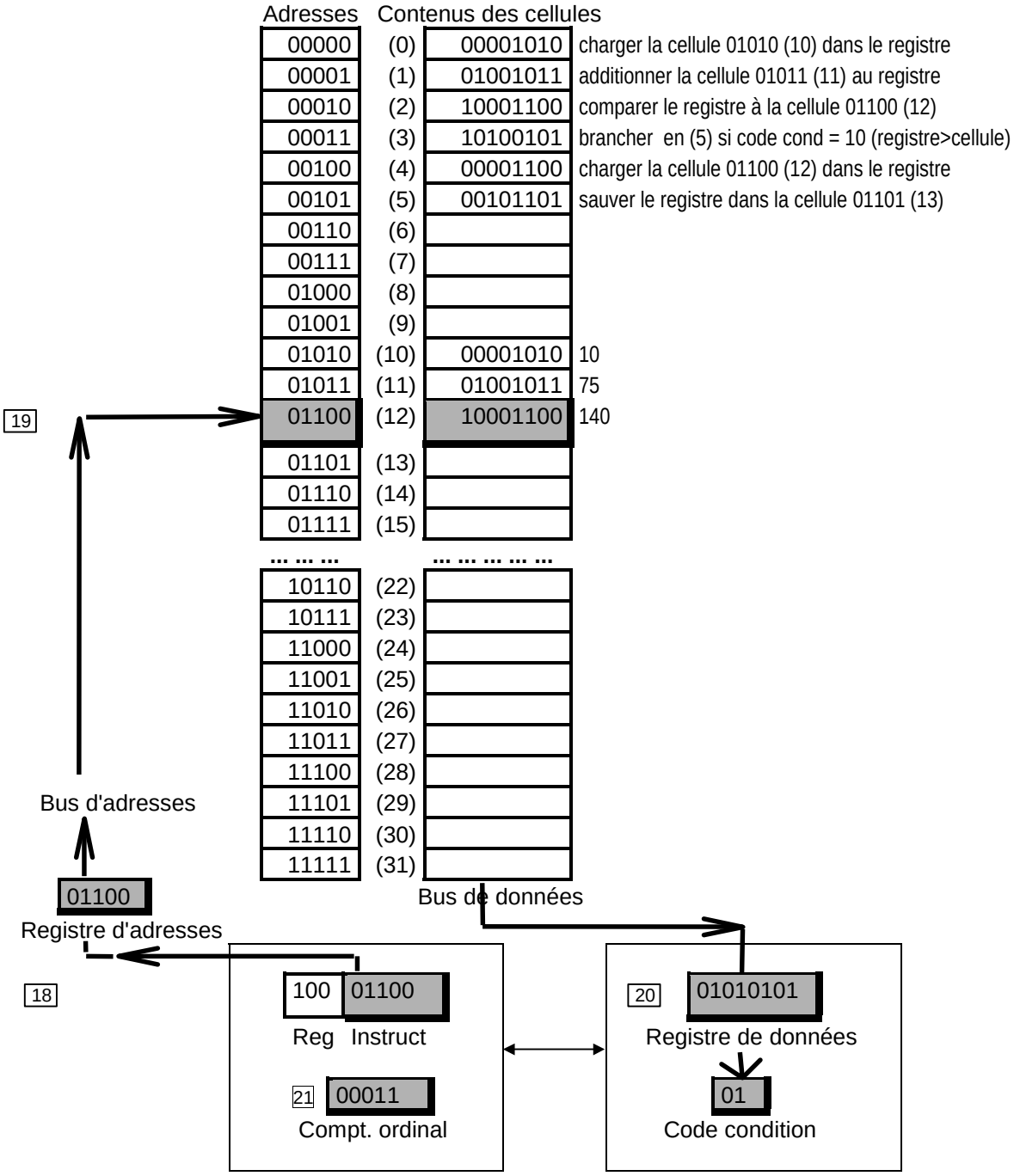



*Figure 5-27 : deuxième phase du troisième cycle*

### 5.2.7.4.7 Première phase du quatrième cycle : chargement de la quatrième instruction dans le registre instruction

22 le compteur ordinal voit son contenu 00011 transféré dans le registre d'adresses

23 l'instruction 10100101 contenue dans la cellule d'adresse 00011 (3) est sélectionnée et

24 chargée dans le registre d'instruction

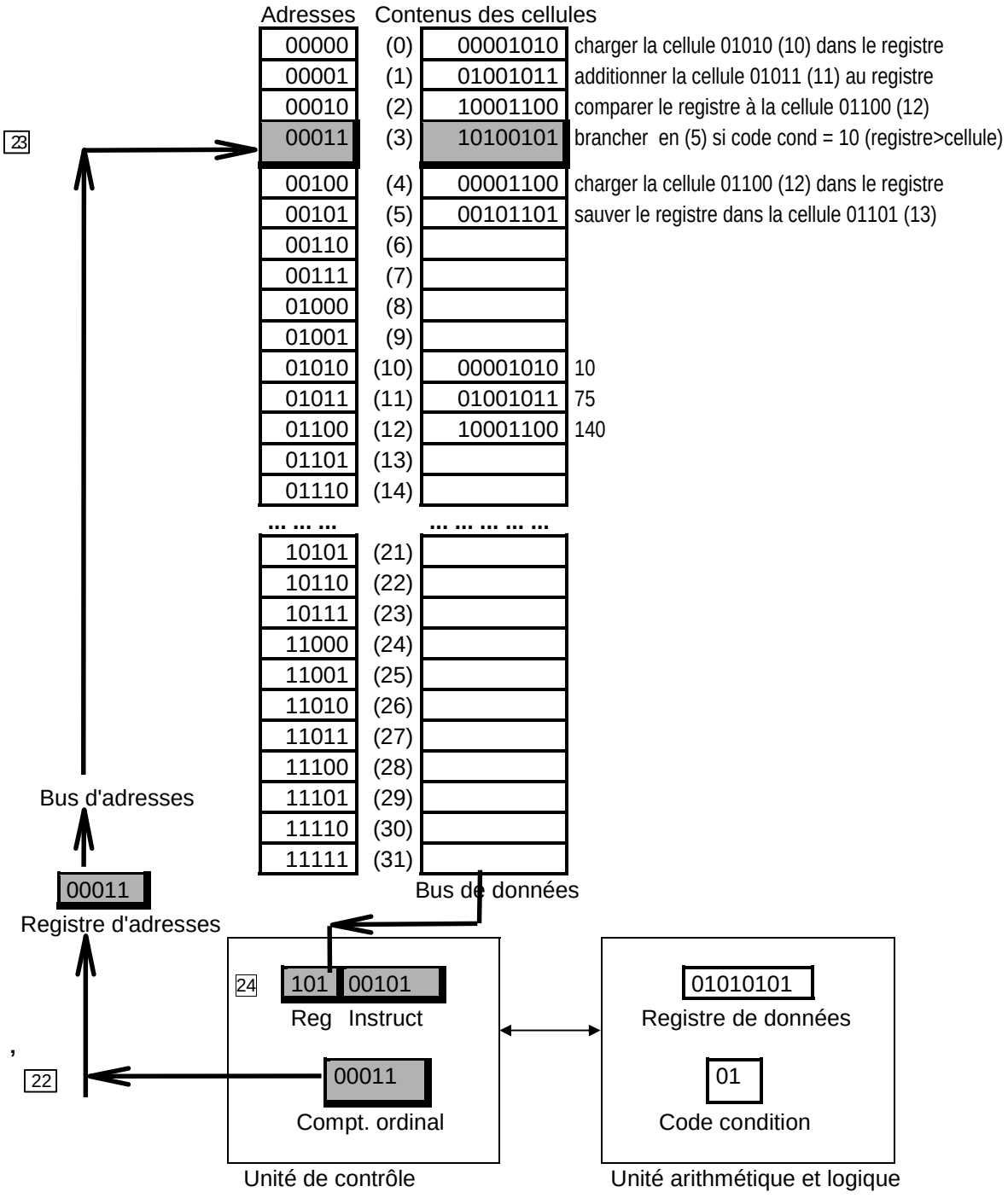



*Figure 5-28 : première phase du quatrième cycle*

### 5.2.7.4.8 Deuxième phase du quatrième cycle : décodage et exécution de la quatrième instruction

25 cette instruction demande de sauter à l'instruction d'adresse 00101 (5) **si le code condition vaut 10** et cela en plaçant cette adresse 00101 dans le compteur ordinal; dans le cas contraire le compteur ordinal est simplement augmenté de 1. Comme ici, le code condition vaut 01, c'est cette dernière éventualité qui est choisie : le compteur ordinal s'incrémente de 1 et passe à 00100 (4). Il n'y a donc pas au cours de l'exécution de cette instruction de branchement de nouvel accès à la mémoire.

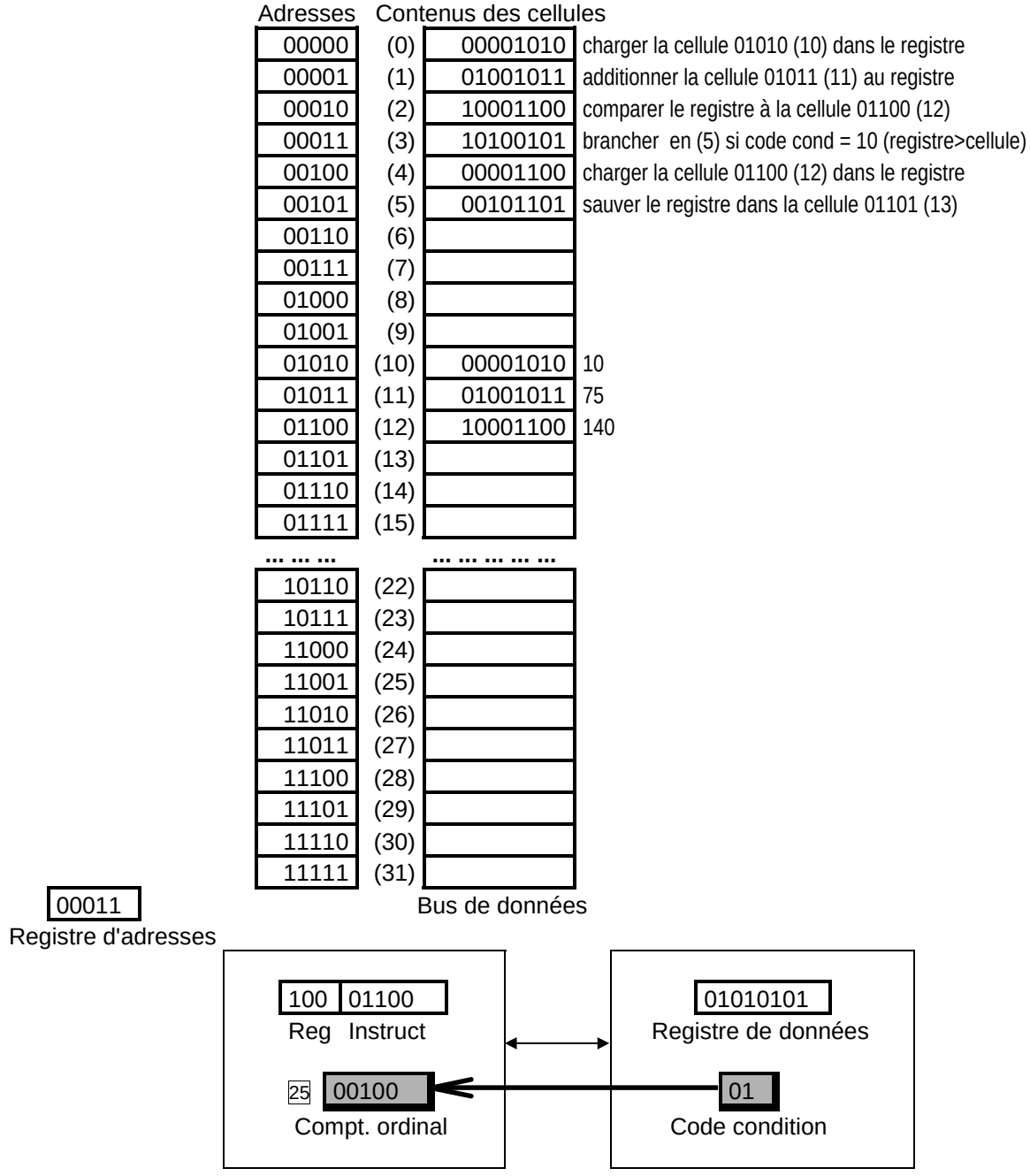



*Figure 5-29 : deuxième phase du quatrième cycle*

### 5.2.7.4.9 Première phase du cinquième cycle : chargement de la cinquième instruction dans le registre instruction

26 le contenu du compteur ordinal, soit 00100, est transféré dans le registre d'adresses

27 l'instruction 00001100 contenue dans la cellule d'adresse 00100 (4) est sélectionnée et

28 chargée dans le registre d'instruction

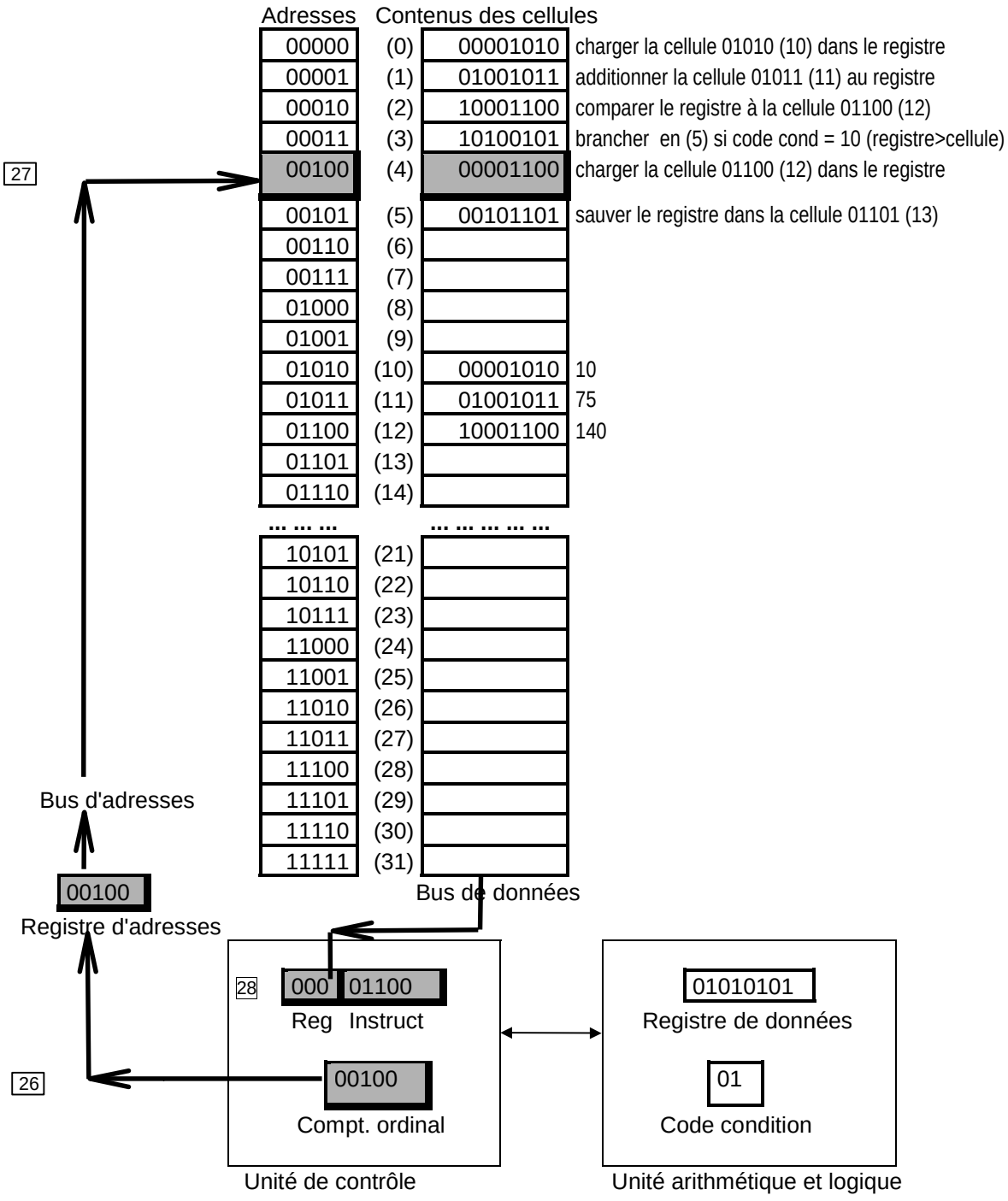



*Figure 5-30 : première phase du cinquième cycle*

#### 5.2.7.4.10 Deuxième phase du cinquième cycle : décodage et exécution de la cinquième instruction

29 l'adresse 01100 (12) mentionnée par cette instruction CHARGER est placée dans le registre d'adresses

30 la cellule d'adresse 01100 (12) est sélectionnée et son contenu 10001100 (140)

31 est chargé dans le registre de données.

32 Le compteur ordinal est augmenté de 1.

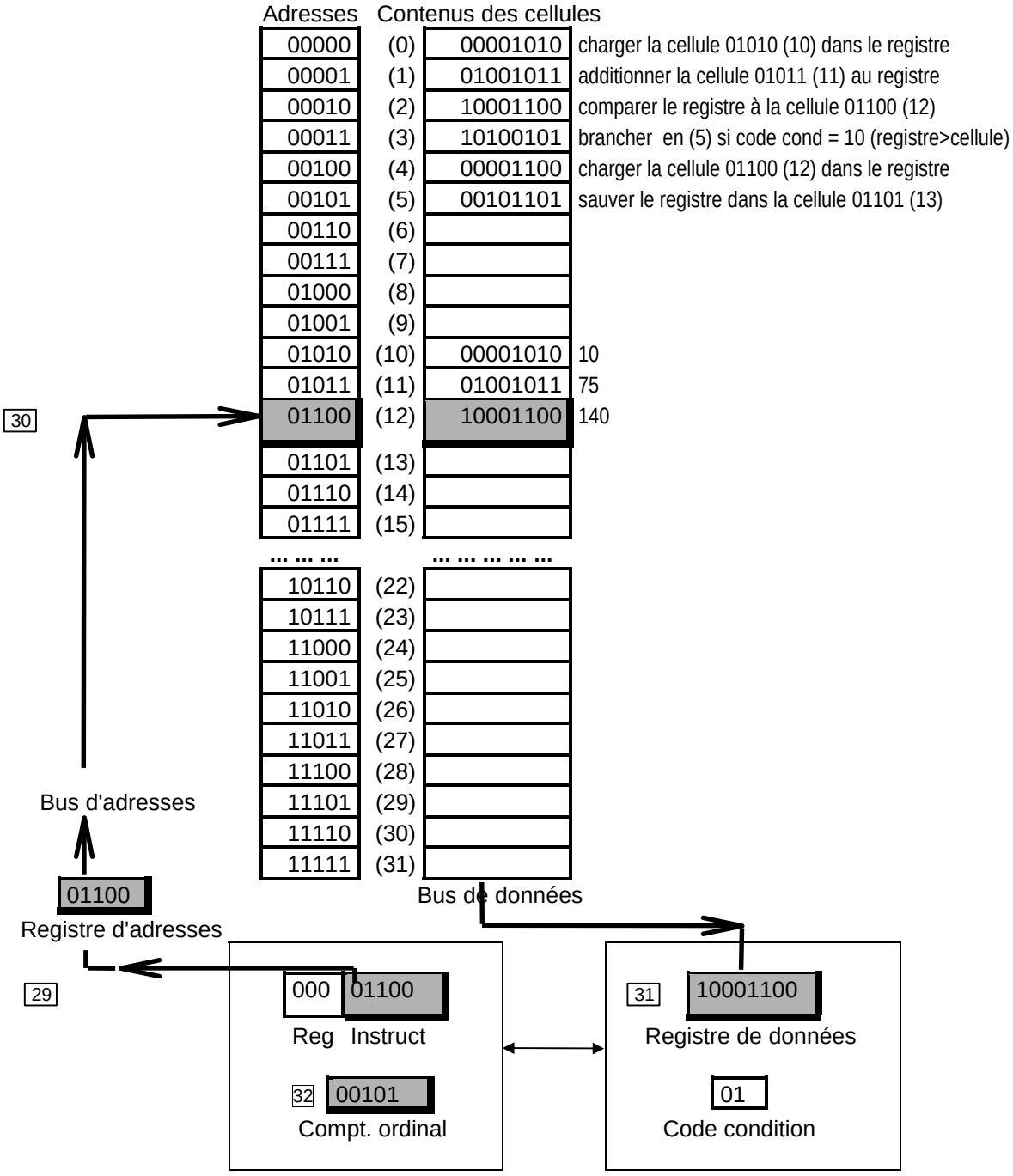



*Figure 5-31 : deuxième phase du cinquième cycle*

### 5.2.7.4.11 Première phase du sixième cycle : chargement de la sixième instruction dans le registre instruction

33 le contenu du compteur ordinal, soit 00101, est transféré dans le registre d'adresses

34 l'instruction 00101101 contenue dans la cellule d'adresse 00101 (5) est sélectionnée et

35 chargée dans le registre d'instruction

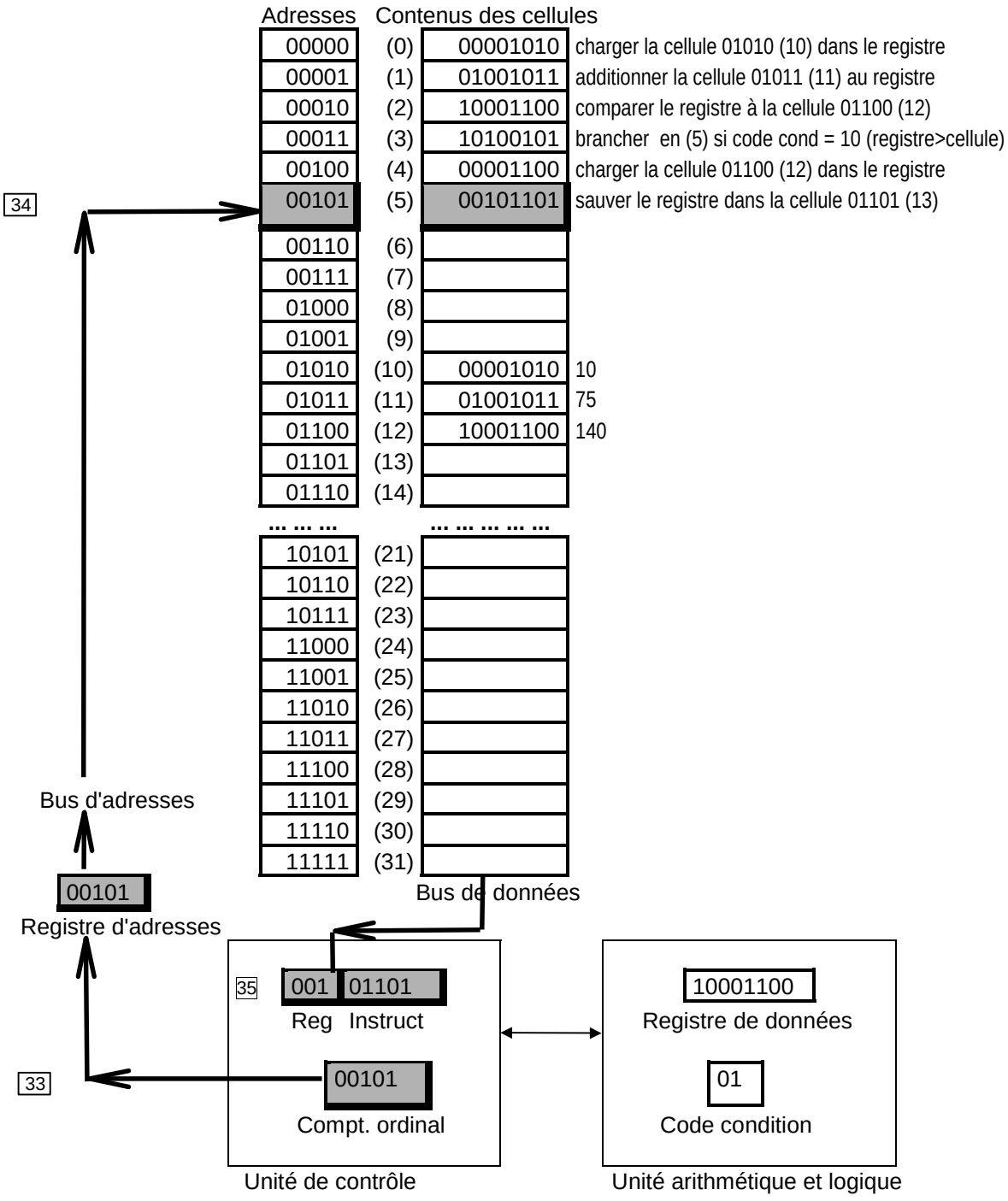



*Figure 5-32 : première phase du sixième cycle*

### 5.2.7.4.12 Deuxième phase du sixième cycle : décodage et exécution de la sixième instruction

36 l'adresse 01101 (13) mentionnée par cette instruction SAUVER est placée dans le registre d'adresses

37 la cellule d'adresse 01101 (13) est sélectionnée et

38 le contenu 10001100 (140) du registre de données est écrit dans cette cellule

39 Le compteur ordinal est augmenté de 1 et passe à 00110 (6).

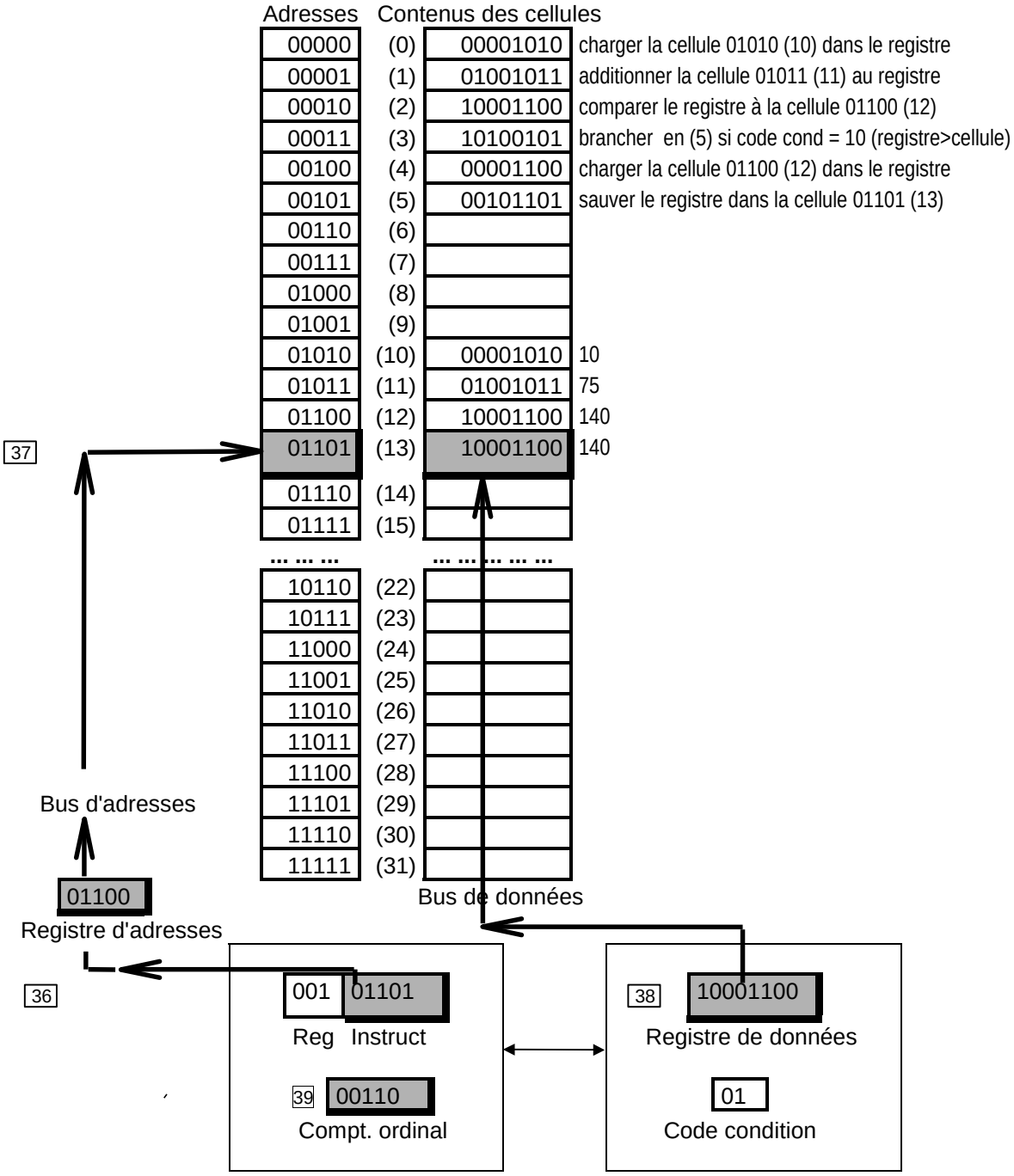



*Figure 5-33 : deuxième phase du sixième cycle*

*5.2.7.5 En guise de résumé*

Chaque étape de l'exécution d'un programme correspond au chargement et à l'exécution d'une instruction :

**1ère phase** : le compteur ordinal vient de changer (à la fin de l'exécution de l'instruction précédente)

le contenu du compteur ordinal est recopié dans le registre d'adresse

on accède *en lecture* à la cellule qui porte cette adresse

le contenu de cette cellule, *toujours considéré à cette phase comme une instruction*, est amené dans le registre d'instruction de l'unité de commande du processeur

**2ème phase** l'instruction va être décodée et exécutée; cette exécution est différente selon qu'il s'agit des instructions "normales" ou des instructions de branchement

- Dans le cas d'une instruction normale (CHARGER, SAUVER, ADDITIONNER, COMPARER,...)
  - ◊ le code opératoire de l'instruction est décodé et le processeur se prépare à exécuter l'opération correspondante
  - ◊ l'adresse constituant la seconde partie de l'instruction est recopiée dans le registre d'adresse
  - ◊ on accède à la mémoire en lecture (pour les instructions CHARGER, ADDITIONNER,...) ou en écriture (pour l'instruction SAUVER) à la cellule dont l'adresse est indiquée
  - ◊ s'il s'agit d'une lecture, le contenu de la cellule est amené vers l'unité arithmétique et logique du processeur (et l'opération demandée est effectuée); s'il s'agit d'une écriture, le contenu du registre de données du processeur est transféré dans la cellule mémoire
  - ◊ le compteur ordinal s'augmente de 1
- Dans le cas d'une instruction de branchement (BRANCHER, BRANCHER SI >, BRANCHER SI =)
  - ◊ dans le cas d'un branchement non conditionnel, l'adresse constituant la seconde partie de l'instruction en cours d'exécution est recopiée dans le compteur ordinal (qui prend cette valeur au lieu de s'augmenter de 1);
  - ◊ s'il s'agit d'un branchement conditionnel, le code condition est consulté, si la condition est vérifiée l'adresse constituant la seconde partie de l'instruction en cours d'exécution est recopiée dans le compteur ordinal (qui prend cette valeur au lieu de s'augmenter de 1), sinon le compteur ordinal augmente simplement de 1

  Donc dans le cas d'un branchement, il n'y a pas, dans cette seconde phase d'accès à la mémoire; tout se passe à l'intérieur du processeur l'objectif étant soit de faire augmenter de 1 le compteur ordinal, soit de lui appliquer un changement brutal.

Il faut noter que lorsque quelque chose est amené dans un registre du processeur ou dans une cellule mémoire, ce qui s'y trouvait auparavant disparaît.

Par contre lorsqu'une cellule ou un registre est consulté pour accéder (en lecture) à ce qui s'y trouve, le contenu n'est pas modifié. C'est seulement une copie qui est prise.

### *5.2.7.6 Quelques remarques*

#### 5.2.7.6.1 Instructions ou données

On l'a vu, rien ne permet de distinguer formellement une instruction d'une donnée : c'est seulement le moment où l'on y accède qui fait considérer le contenu d'une cellule comme une instruction (1ère phase) ou comme une donnée (2ème phase).

Ainsi, dans l'exemple traité ci-dessus, l'instruction CHARGER (00001010) contenue dans la cellule d'adresse 0 est identique à la donnée 10 (00001010) contenue dans la cellule d'adresse 10. Il en va de même pour l'instruction de la cellule d'adresse 1 et la donnée contenue dans la cellule d'adresse 11 ou encore de l'instruction de la cellule 2 et de la donnée de la cellule 12.

Ainsi donc, face au contenu d'une cellule mémoire, il est impossible de décider s'il s'agit d'une instruction ou d'une donnée; et lorsque le contexte permet de déterminer qu'il s'agit d'une donnée, il est impossible de distinguer si le nombre binaire considéré fait partie du code d'un caractère, d'un entier, d'un réel, d'un pixel, d'un échantillonnage de son, etc..

Remarquons encore, étant donné ces remarques, qu'il est tout à fait possible qu'un programme modifie les instructions le constituant pendant le cours de son exécution.

#### 5.2.7.6.2 Sans branchement, pas de programme

Si les diverses instructions de branchement n'existaient pas, l'exécution des programmes serait purement séquentielle : à la fin de l'exécution de chaque instruction, le compteur ordinal augmente de 1 ce qui fait qu'on passe à l'exécution de l'instruction rangée dans la cellule suivante, et ainsi de suite.

Les instructions de branchement vont permettre de briser cette séquentialité de l'exécution en faisant en sorte que l'instruction exécutée dans la suite soit non pas celle qui suit, mais n'importe quelle autre (celle vers laquelle on "branche", en plaçant son adresse dans le compteur ordinal).

Ce sont donc les branchements qui vont permettre de traduire en langage machine les *répétitions* d'un groupe d'instructions (à la fin on branche au début) ou encore, comme dans l'exemple traité, le fait de *choisir* un cheminement ou un autre dans les instructions constituant le programme.

?

Que pensez-vous du petit programme suivant, écrit dans le langage du modèle simplifié d'unité centrale décrit plus haut :

| Adresses | | Contenus des cellules |
|---|---|---|
| 00000 | (0) | 00000000 |
| 00001 | (1) | 11100000 |

en imaginant qu'à un moment donné, le compteur ordinal devient égal à 0 ?

### *5.2.7.7 Exercices*

1. Pourriez vous écrire pour l'unité centrale simplifiée utilisée dans l'exemple précédent un programme qui trie les données désignées par a et b, contenues dans les cellules d'adresse 00001 (1) et 00010 (2). Autrement dit, à l'issue de l'exécution de ce programme la cellule 1 contiendra la plus petite des quantités a et b et la cellule 2 la plus grande.

2. Pourriez-vous écrire le programme qui ajoute 3 au contenu de la cellule d'adresse 00001.

☞ Il est impératif de se rendre compte que le modèle simplifié présenté ici, s'il nous a permis d'exposer et d'illustrer les principes qui sont à la base du fonctionnement de pratiquement tous les ordinateurs, est infiniment plus simple que les unités centrales réelles. Quelques remarques vont permettre d'élargir le propos.

### *5.2.8 Architecture et fonctionnement d'une unité centrale, en général*

#### *5.2.8.1 Le modèle d'ordinateur de Von Neumann : calculateur universel à programme enregistré*

L'organisation décrite ci-dessus et les principes de fonctionnement mis en évidence à travers l'exemple sont caractéristiques de tout ordinateur conforme au modèle développé par John von Neumann dans un article[3] publié le 30 juin 1945 et qui fut à la naissance (du moins sur le plan théorique) des ordinateurs modernes.

Les concepts essentiels en sont les suivants :

- L'unité centrale est constituée d'une mémoire centrale et d'un processeur : entre ces deux entités, instructions et données vont pouvoir être échangées.
- Les instructions constituant les programmes qui vont faire agir le processeur sont enregistrées en mémoire centrale au même titre que les données; rien ne permet donc de distinguer une instruction (codée en binaire) d'une donnée (codée en binaire). Ces instructions peuvent elles-mêmes être modifiées pendant l'exécution du programme qu'elles constituent.
- A tout moment c'est la valeur d'un registre particulier du processeur, le compteur ordinal (en anglais P-Counter), qui donne à la fin de l'exécution de chaque instruction par le processeur, l'adresse en mémoire de l'instruction qui sera exécutée ensuite. Habituellement, ce compteur ordinal augmente de 1 à la fin de l'exécution de chaque instruction; il peut aussi faire un saut en prenant une valeur particulière indiquée par une instruction de branchement.
- Outre les échanges avec la mémoire centrale, le processeur effectue essentiellement des calculs et des comparaisons sur les données présentes dans ses registres de données. Le processeur est un calculateur, mais universel, puisque la succession des opérations qu'il va exécuter et le choix des données pour ces opérations ne sont pas fixés une fois pour toute mais sont précisés dans un programme.
- Une seule instruction à la fois est amenée de la mémoire centrale et exécutée par l'unique processeur.

Ces principes de fonctionnement restent ceux de la plupart des ordinateurs modernes.

Il y a d'autres types d'ordinateurs comme ceux où tout un ensemble de processeurs (parfois des milliers) travaillent en parallèle sur des données et sur base d'instructions provenant de la même mémoire centrale (avec tous les problèmes de synchronisation qu'on peut imaginer).
Il existe aussi des ordinateurs neuronaux dont le fonctionnement singe en quelque sorte ce que nous savons du cerveau humain. Ces ordinateurs sont constituée de

3 "*First draft of a report on the EDVAC*"

"neurone" électroniques connectés entre eux, des signaux électriques pouvant transiter le long de ces connexions.

#### *5.2.8.2 Le langage machine*

La structure et la nature des instructions de l'exemple simplifié étaient particulièrement élémentaires. Les vraies instructions des vrais processeurs sont évidemment plus complexes, mais l'exemple permet cependant de tirer quelques conclusions générales :

- Chaque processeur est caractérisé par les actions dont il est capable et par le jeu d'instructions qui vont commander ces actions. Ces instructions, qui sont toujours des configurations binaires, constituent le langage du processeur, ce qu'on appelle parfois le *langage machine*.

  Le concept de langage machine, au singulier, est inadéquat : il y a autant de langages machines que de types de processeurs différents.

  Bien évidemment, l'évolution dans le temps de ces processeurs fait qu'on peut en quelque sorte y tracer une généalogie et les successeurs d'un processeur donné continuent le plus souvent à "comprendre" le langage du processeur auquel ils succèdent, l'inverse n'étant pas nécessairement vrai.

  Le nombre de ces instructions varie donc d'un processeur à l'autre : de quelques dizaines à plus de cent (sinon des centaines).

- La teneur, la portée et la complexité de ces instructions sont également très variables. On peut cependant dire que généralement, la plupart des instructions comportent un code opératoire (indiquant au processeur l'action à effectuer) suivi d'une ou plusieurs adresses (précisant où les données doivent être lues en mémoire et/ou où les résultats de l'opération effectuée doivent être écrits en mémoire).

Diverses instructions de branchements conditionnel ou non conditionnel sont évidemment disponibles.

#### *5.2.8.3 Quelques questions*

Plusieurs questions restent à traiter. Motivées par l'exemple présenté, elles sont cependant d'une portée très générale.

- Comment le programme à exécuter se trouve-t-il placé dans la mémoire à l'endroit où on le trouve ?

  Ainsi dans le cas de l'exemple proposé page 88 le programme créé se trouve en mémoire à partir de la cellule d'adresse 0 et s'étend jusqu'à la cellule d'adresse 5. Comment le programme se trouve-t-il logé là ? D'où venait-il ?

- Comment les données à traiter se trouvent-elles placées en mémoire où on peut les trouver ?

  Dans le même exemple, les données à traiter sont placées dans les cellules d'adresse 10, 11 et 12. Qui les dispose dans ces cellules ? D'où viennent les données qui y seront déposées ?

- Comment le compteur ordinal prend-il comme valeur l'adresse de la première instruction du programme à exécuter, ce qui va lancer l'exécution de ce dernier ? Que se passe-t-il à la fin de l'exécution du programme considéré ?

La réponse à ces questions est en grande partie commune : c'est un autre programme qui se charge d'aller chercher le programme à exécuter (par exemple sur une mémoire de masse : disquette, disque dur,...) et les données et les place à des endroits convenables de la mémoire. Ce programme se termine par un branchement à l'adresse de la première instruction du programme qui vient d'être installé. A la fin de l'exécution du programme considéré, c'est

le plus souvent un nouveau branchement qui va "passer la main" à un autre programme, et ainsi de suite.

Mais cette réponse est insatisfaisante, puisqu'on peut reposer les 3 mêmes questions à propos de ce programme qui se charge de l'installation. On peut répondre que ce dernier est lui aussi installé (par un autre programme). De proche en proche, on va remonter jusqu'à l'instant 0, celui de la mise en marche. Ce qui se passe lors de cette mise en marche, c'est simplement le fait que le compteur ordinal prend automatiquement une valeur qui est simplement celle d'un tout premier programme enfermé en ROM et qui va se charger d'aller prendre un autre programme sur mémoire de masse, l'installer en mémoire centrale et passer la main à ce programme, qui lui même en installera d'autres, etc.

Ainsi, à la mise en marche d'un ordinateur de type PC, le compteur ordinal prend la valeur 1048560, ce qui provoque l'exécution d'un programme logé en mémoire morte à partir de l'adresse 1048560. Dès lors une série de programmes situés en ROM sont exécutés les uns "passant la main" (par un branchement) aux autres, en un enchaînement purement déterministe. Bien évidemment, au bout d'un temps, ce sont les interventions venues de l'extérieur sous forme de données (au départ du clavier ou de la souris, par exemple) qui vont faire évoluer le flot des programmes qui seront chargés en mémoire et exécutés. (0ctobre 1998)

Il est donc important d'avoir cette vision dynamique de ce qui se passe dans l'unité centrale : plusieurs programmes cohabitent en mémoire centrale, s'exécutent à certains moments et passent ensuite la main à d'autres.

Ce qu'on peut en tout cas dire c'est qu'à aucun moment le processeur n'est "arrêté"; constamment des instructions sont ramenées, exécutées et, le compteur ordinal se modifiant, font place à d'autres, et cela sans arrêt.

### *5.2.9 Les paramètres d'évaluation d'une unité centrale*

On sait que les performances du matériel évoluent sans cesse (même si le principe de fonctionnement reste celui décrit dans les années 40 par von Neumann). Il est essentiel de disposer d'une grille d'analyse de ces évolutions et pour cela de connaître les principaux paramètres permettant de *décrire* et d'*évaluer* une unité centrale.

#### *5.2.9.1 La taille de la mémoire adressable : taille du registre d'adresses*

On sait que pour accéder à une cellule de la mémoire, que ce soit en lecture ou en écriture, le processeur place dans le registre d'adresses l'adresse de la cellule à atteindre. La taille du registre d'adresse est donc un paramètre fondamental puisqu'elle détermine le nombre des adresses possibles des cellules de la mémoire.

Ainsi dans l'exemple d'unité centrale présenté plus haut, la taille du registre d'adresses était de 5 bits, ce qui permet de noter (en binaire) des adresses allant de 00000 (0) à 11111 (31). Il serait donc parfaitement inutile dans ce cas d'avoir une mémoire centrale qui comporterait plus de 32 cellules, puisque au delà de la 32ème le processeur ne pourrait indiquer dans le registre d'adresses les adresses correspondantes.

C'est un peu comme si on imposait que les numéros des maisons d'une rue ne comportent qu'au plus 2 chiffres (décimaux). En admettant qu'une des maisons puisse avoir le chiffre 0, on pourrait adresser du courrier aux 100 maisons portant les numéros de 0 à 99. Si des maisons supplémentaires sont bâties dans cette rue, jamais on ne pourra leur affecter un numéro (sur deux chiffres) et jamais elles ne pourront recevoir du courrier.

On peut aisément calculer que si le registre d'adresses possède une taille de n bits, $2^n$ cellules pourront être adressables. Si chaque cellule adressable a une taille d'un octet, on aura alors $2^n$ octets de mémoire adressable.

Dans le monde de la micro-informatique, la taille du registre d'adresses (et du bus d'adresses qui lui est associé) ont évolué au cours du temps. Les premiers micro-ordinateurs réellement commercialisés possédaient un registre d'adresses sur 16 bits (processeurs 8080) ce qui permettait d'adresser $2^{16}$ cellules d'un octet soit 65536 octets ou 64 Ko. Vinrent ensuite des unités centrales avec des registres d'adresses de 20 bits (processeurs 8086 ou 8088) permettant d'accéder à une mémoire de $2^{20}$ octets ou 1048576 octets, soit 1 Mo.
Actuellement la taille des registres et bus d'adresses est de 32 octets permettant théoriquement d'adresser une mémoire de $2^{32}$ octets soit 4294967296 octets ou 4 Go. (Voir aussi page 77) (Novembre 2000)

### *5.2.9.2 La taille de la mémoire effectivement disponible*

Même si les processeur actuels permettent en théorie d'adresser 4 Go de mémoire, la mémoire effectivement disponible est beaucoup moindre. De plus, quand on parle de mémoire disponible, il s'agit essentiellement de la mémoire vive (RAM), la seule où les programmes et les données venant de l'extérieur peuvent venir prendre place.

11/2001

Dans le passé, la mémoire effectivement disponible était généralement identique à la mémoire adressable (64 Ko, 1 Mo). Aujourd'hui la mémoire équipant effectivement la plupart des systèmes est comprise entre quelques dizaines de Mo (16 Mo au minimum) et quelques centaines de Mo (256 Mo). A l'heure où j'écris ces lignes, la taille habituelle est en moyenne de 64 ou 128 Mo. (Novembre 2001)
Rappelons, étant donné ce qui a été dit page 75, que 1 Mo est l'équivalent d'environ un million de caractères soit 500 pages de texte bien remplies. Dire qu'un ordinateur dispose d'une mémoire de 128 Mo, c'est dire que si l'on remplissait complètement cette mémoire avec du texte (ce qui ne pourra jamais être le cas pour la mémoire centrale qui doit aussi contenir des programmes), on pourrait y placer l'équivalent de 64000 pages de texte.

### *5.2.9.3 Le temps d'accès à la mémoire*

C'est le délai entre le moment où le processeur demande un accès à telle cellule de la mémoire, par exemple pour y lire l'information, et le moment où le contenu de cette cellule est disponible. Ce temps d'accès à la mémoire centrale se mesure en nano-secondes (milliardièmes de seconde).

☞ C'est sans doute le moment de rappeler les préfixes, tant multiplicatifs que de division :

| | | | |
|---|---|---|---|
| kilo | x 1.000 | mili | /1.000 |
| mega | x 1.000.000 | micro | /1.000.000 |
| giga | x 1.000.000.000 | nano | /1.000.000.000 |
| tera | x 1.000.000.000.000 | pico | /1.000.000.000.000 |

Pour donner un ordre d'idée de ce qu'est une nano-seconde, disons que la lumière qui se propage pourtant à 300.000 km/s, parcourt seulement 30 centimètres pendant une nano-seconde.

Les mémoires les plus rapides actuellement (SRam) ont des temps d'accès de quelques nano-secondes (une petite dizaine de ns). Les mémoire plus lentes (DRam) ont des temps d'accès de quelques dizaines de ns. Il y a environ un facteur 10 entre les temps d'accès mémoires les plus lentes et les plus rapides.

#### *5.2.9.4 La présence de mémoire-cache et sa taille*

La plupart des systèmes récents intègrent deux sinon trois catégories de mémoire centrale eu égard à leur rapidité. A côté de la plus grosse partie de la mémoire, relativement lente, on trouve en général une zone de mémoire nettement plus rapide, qu'on appelle la mémoire cache.

On s'arrange dès lors pour que :

- Autant que possible, le programme actuellement actif réside en tout ou en partie dans cette zone de mémoire-cache à laquelle le processeur accède beaucoup plus rapidement qu'au reste de la mémoire; il en est de même pour les données actuellement traitées.
- Lorsque le processeur a besoin d'instructions ou de données qui ne sont pas présentes dans le cache, il y a échange entre le cache et la mémoire habituelle pour amener les instructions et données nécessaires dans le cache et cela, bien entendu, sans que cet échange se fasse par l'intermédiaire du processeur.

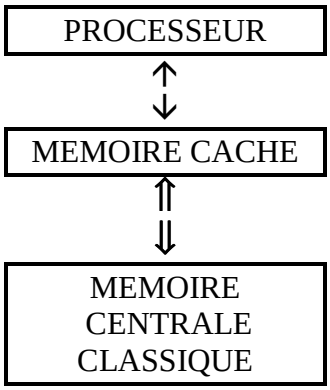



*Figure 5-34 : rôle de la mémoire cache*

Pourquoi les échanges entre la mémoire classique et la mémoire cache se font-ils sans que les informations (instructions ou données) échangées ne transitent par le processeur ?

#### *5.2.9.5 La taille d'une cellule mémoire*

Elle est en général d'un octet. Elle peut cependant être plus grande sur certains systèmes.

#### *5.2.9.6 La taille du bus de données*

C'est elle qui conditionne la quantité d'octets échangés en une fois entre la mémoire et le processeur, que ces octets représentent des données ou des instructions.

C'est cette caractéristique (la "largeur" du bus de données) qui a servi à caractériser et à nommer les processeurs. Ainsi, au fil de l'évolution des systèmes informatiques, on a eu des processeurs 8 bits (un octet seulement est échangé lors des transferts entre mémoire et processeur), des processeurs 16 bits (2 octets à la fois à chaque échange), des processeurs 32

bits et on en est aujourd'hui à des processeurs 64 bits (8 octets sont échangés d'un coup entre la mémoire centrale et le processeur).

11/2001

Ainsi, en ce qui concerne la famille des processeurs Intel, on a eu successivement (sans parler du 4004) le processeur 8088 (processeur 8 bits), le 80286 (processeur 16 bits), le 80386 et le 80486 (processeurs 32 bits) et le Pentium (processeur 64 bits). (Voir aussi page 79)

Un petit exemple fera comprendre le progrès apporté par le passage d'un processeur 8 bits à un processeur 16 bits. Nous savons que les entiers sont souvent codés sur 2 octets en mémoire (Voir page 71). Imaginons donc que nous avons deux entiers A et B dont nous souhaitons obtenir simplement la somme. Le premier entier A est codé sur deux octets et le second B également. Avec un langage machine du type de celui développé dans l'exemple traité ci-dessus et avec un processeur 16 bits, il suffit de trois "voyages" entre la mémoire et le processeur pour obtenir la somme. En effet,

- on va amener (dans un registre de données du processeur) *en une fois* les deux octets dans lesquels A est écrit (premier transfert),
- on va également amener *en une fois* les deux octets codant B pour les additionner à ceux déjà présents dans le registre de données (deuxième transfert);
- enfin le résultat (codé sur deux octets) sera *en une fois* transféré vers les deux cellules de la mémoire qui contiendront le codage de la somme (troisième transfert).

Par contre, avec un processeur 8 bits, il faudra en tout 6 transferts :

- on va amener (dans un registre de données du processeur) l'octet codant la partie droite du nombre A; comme on ne peut transférer qu'un octet à la fois, c'est la seule possibilité;
- on va amener l'octet codant la partie droite du nombre B en l'additionnant à la partie droite de A déjà présente dans le processeur;
- on va transférer l'octet constituant la somme des deux parties droites ainsi obtenue dans une cellule de la mémoire : celle-ci contiendra la partie droite du résultat; (on garde dans le processeur le report éventuel obtenu suite à l'addition);
- on va amener (dans un registre de données du processeur) l'octet codant la partie gauche du nombre A; comme on ne peut transférer qu'un octet à la fois, c'est la seule possibilité;
- on va amener l'octet codant la partie gauche du nombre B en l'additionnant à la partie gauche de A déjà présente dans le processeur (et au report éventuel obtenu suite à l'addition des parties droites);
- on va transférer l'octet constituant la somme des deux parties gauches ainsi obtenue dans une cellule de la mémoire : celle-ci contiendra la partie gauche du résultat.

Pour effectuer le même traitement d'addition de deux entiers, il aura bien fallu 6 transferts avec un processeur 8 bits et seulement 3 avec un processeur 16 bits !

### *5.2.9.7 Le nombre et la taille des registres de données du processeur*

Il s'agit là d'un paramètre très technique et sur lequel je n'insisterai guère. L'exemple d'unité centrale présenté incluait un processeur ne présentant que deux registres internes, le registre de données proprement dit et le registre code-condition. On devine que plus les registres du processeur sont nombreux et plus leur taille est importante, plus on risque d'avoir un système efficace.

#### *5.2.9.8 Le jeu d'instructions disponibles pour le processeur et la "puissance" de ces dernières*

Il s'agit également d'un paramètre très technique et qui ne sera pas abordé en détail. Nous l'avons déjà évoqué page 105. L'exemple développé montrait un processeur imaginaire ne comportant que 8 instructions; de plus les opérations commandées par ces instructions étaient fort élémentaires. Pour obtenir en mémoire, à l'adresse 00011 (3) la somme des contenus des deux cellules d'adresses respectives 00001 (1) et 00010 (2), on était obligé de donner trois instructions en séquence :

| | |
|---|---|
| 000 00001 | charger le contenu de la cellule 00001 (1) dans le registre de données |
| 010 00010 | y additionner le contenu de la cellule 00010 (2) |
| 001 00011 | écrire le registre de données dans la cellule 00011 (3) |

On devine que certains processeurs disposent d'instructions plus puissantes et que, par exemple, une seule instruction permettrait alors de faire le même travail :

| 00001111 | 00000001 | 00000010 | 00000011 |
|---|---|---|---|
| code opératoire, codé sur un octet et commandant de lire successivement le contenu de deux cellules mémoire (dont les adresses suivent) et de placer le résultat dans une cellule (dont l'adresse suit également) | adresse (1) codée sur un octet de la cellule contenant la première donnée | adresse (2) codée sur un octet de la cellule contenant la seconde donnée qui doit être additionnée à la première | adresse (3) codée sur un octet de la cellule où le résultat de l'addition doit être placé. |

Il est évident que l'écriture de programmes, dans ce cas, utilisera bien moins d'instructions que dans le premier cas évoqué.

Le nombre et la puissance des instructions propres à un processeur constituent dès lors un paramètre important de description de l'efficacité d'un système.

#### *5.2.9.9 La vitesse d'horloge*

On a vu (page 77) que les battements d'une horloge permettaient de synchroniser le travail des divers éléments de l'unité centrale. La fréquence à laquelle cette horloge oblige ces divers éléments à travailler est évidemment un paramètre important. Les valeurs actuelles de cette fréquence sont de plusieurs centaines de méga-hertz (= plusieurs centaines de millions de battements pas seconde). On annonce (avec le Pentium IV) 2 GHz de fréquence.

11/2001

Les premiers micro-processeurs travaillaient avec une fréquence de quelques MHz (de 0,108 MHz pour le 4004 à 8 MHz pour le 8088); on est ensuite passé à quelques dizaines de MHz (de 12 MHz pour le 80286 à 100 MHz pour le 80486). On en est aujourd'hui à des fréquences de plusieurs centaines de MHz (de 166 MHz pour le Pentium à 300 MHz pour le Pentium II). Le Pentium III a dépassé le GigaHertz et on en est à un Pentium IV à 2 GHz. (Novembre 2001)

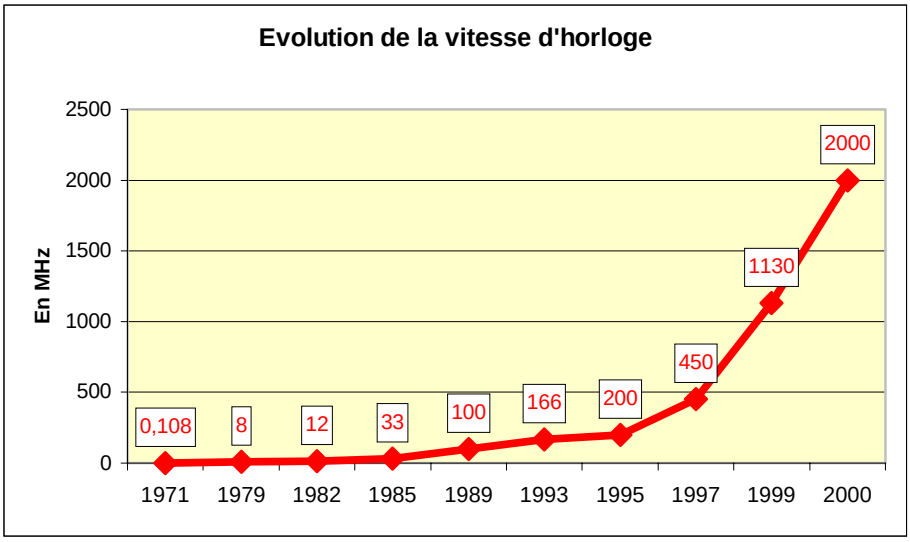



*Figure 5-35 : évolution de la fréquence d'horloge des processeurs*

#### *5.2.9.10 Le nombre d'instructions exécutées par seconde*

La puissance d'une unité centrale s'évalue parfois en MIPS (Million d'Instructions Par Seconde).

Bien évidemment, le nombre de MIPS d'un système ne donne qu'une idée relativement grossière de la puissance de ce dernier puisque, comme signalé ci-dessus, un autre paramètre essentiel est la puissance des instructions dont on parle. Ainsi, il vaut peut être mieux un système "tournant" à 100 MIPS avec des instructions puissantes et complexes plutôt qu'un système à 200 MIPS avec des instructions rudimentaires du type de celles présentées dans l'exemple développé. C'est ce qui fait dire à certains que MIPS signifie "Mauvais Indicateur de la Puissance d'un Système" !


11/2001

Les premiers micro-ordinateurs "tournaient" à quelques MIPS (0,75 pour le 8088 et 2,66 pour le 80286). On a eu ensuite des systèmes à quelques dizaines de MIPS (11 MIPS pour le 80386 et 70 MIPS pour le 80486). La mesure en MIPS des performances d'un système n'est plus guère utilisée pour les systèmes actuels, mais leur puissance dépasse aujourd'hui le millier de MIPS.

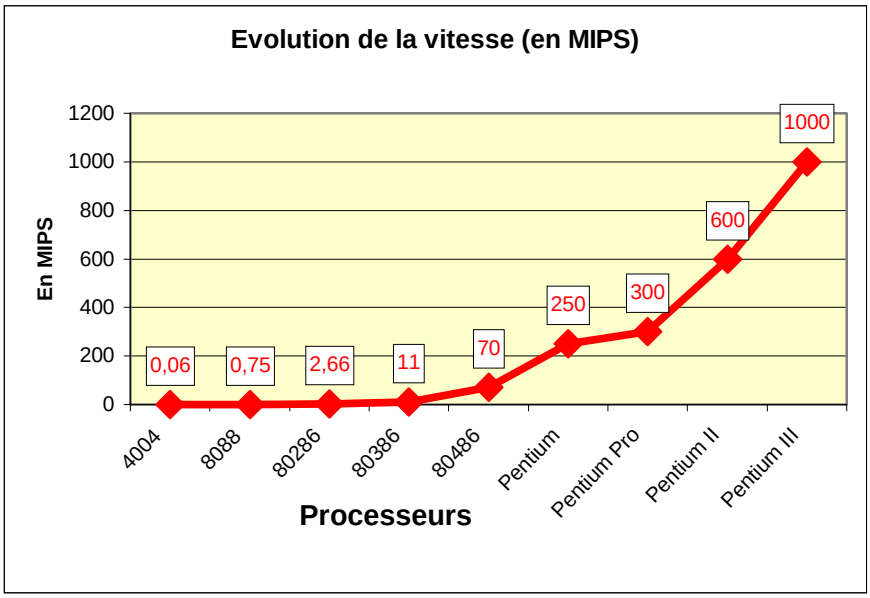



*Figure 5-36 : évolution de la vitesse des processeurs*

Etant donné ce que nous avons dit du fonctionnement d'un système informatique, n'est il pas étonnant qu'un système avec une fréquence d'horloge de 100 MHz puisse "tourner", par exemple, à 100 MIPS ?

Afin de résumer l'évolution des systèmes (basés sur les micro-processeurs Intel), on peut proposer le tableau suivant (11/2001) :

| **Type** | **4004** | **8088** | **80286** | **80386** |
|---|---|---|---|---|
| Année | 1971 | 1979 | 1982 | 1985 |
| Taille des registres de calcul | ? | 16 bits | 16 bits | 32 bits |
| Taille du bus de données | 4 bits | 8 bits | 16 bits | 32 bits |
| Registre (et bus) d'adresses | ??? | 20 bits | 24 bits | 32 bits |
| Fréquence d'horloge (MHz) | 0,108 MHz | 5 à 8 MHz | 6 à 12 MHz | 16 à 33 MHz |
| Vitesse (MIPS) | 0,06 MIPS | 0,33 à 0,75 | 0,9 à 2,66 | 5 à 11 MIPS |
| Nombre de transistors | 2300 | 29.000 | 134.000 | 275.000 |
| Prix (en $) | 200 | 360 | 360 | 299 |

| **Type** | **80486** | **Pentium** | **Pentium Pro** | **Pentium II** |
|---|---|---|---|---|
| Année | 1989 | 1993 | 1995 | 1997 |
| Taille des registres de calcul | 32 bits | 32 bits | 32 bits | 32 bits |
| Taille du bus de données | 32 bits | 64 bits | 64 bits | 64 bits |
| Registre (et bus) d'adresses | 32 bits | 32 bits | 32 bits | 32 bits |
| Fréquence d'horloge (MHz) | 25 à 100 MHz | 60 à 166 MHz | 150 à 200 MHz | 233 à 300 MHz |
| Vitesse (MIPS) | 20 à 70 MIPS | 100 à 250 MIPS | 400 MIPS ???[4] | 600 MIPS ??? |
| Nombre de transistors | 1.600.000 | 3 millions | 5,5 millions (31 avec le cache de 512 K) | 7,5 millions |
| Prix (en $) | 950 | 878 | 974 | ? |
| Remarques | Mémoire cache de 8 Ko<br>Coprocesseur arithmétique | Mémoire cache de 16 Ko<br>Coprocesseur arithmétique | Mémoire cache de 512 Ko | Mémoire cache de 512 Ko |

| **Type** | Pentium III | **Pentium IV** |
|---|---|---|
| Année | 1999 | **2000 ?** |
| Taille des registres de calcul | 32 bits | **32 bits** |
| Taille du bus de données | 64 bits | **64 bits** |
| Registre (et bus) d'adresses<br>Mémoire adressable | 32 bits | **32 bits** |
| Fréquence d'horloge (MHz) | 450 à 1130 MHz | **1,4 GHz à 2,8 GHz** |
| Vitesse (MIPS) | | |
| Nombre de transistors | 8,2 millions<br>28 millions (avec le cache) ? | **42 millions (avec le cache)** |
| Prix (en $) | 250 (800 MHz)<br>990 (1.13 GHz) | **160 (1,3 GHz)**<br>**500 (2 GHz)** |
| | Cache interne de 512 Ko | Cache interne de 256 Ko |

Le nombre de transistors implantés sur la "puce" constituant le processeur est une bonne mesure de la complexité et des performances de ce dernier. Depuis les premiers processeurs, on constate empiriquement que ce nombre de transistors double environ tous les 18 mois : c'est ce qu'avait prévu Moore dès 1965 dans une conjecture qui porte son nom, la "loi de Moore".

4 Comme précisé, on n'évalue plus guère en MIPS les performances des processeurs récents.

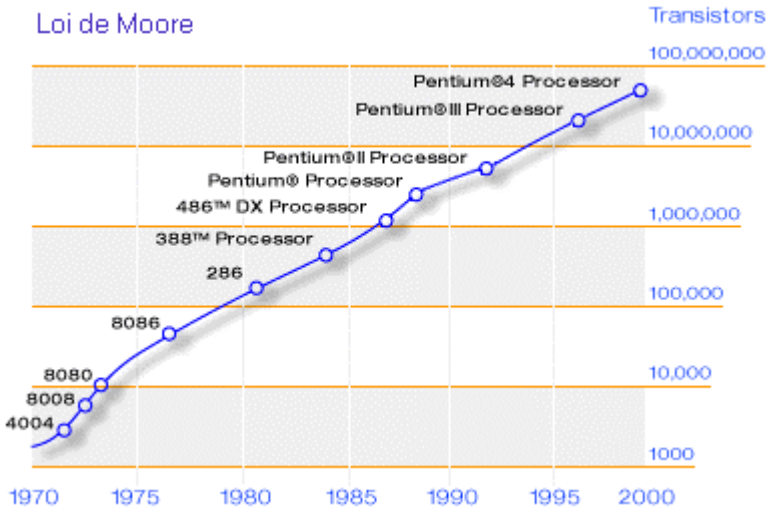



*Figure 5-37 : loi de Moore*

On notera l'échelle logarithmique en ordonnée, dans le graphique précédent.

Le co-processeur arithmétique, ajouté aux processeurs de la fin des années 80, est un dispositif spécialisé dans le calcul portant sur les nombres réels codés en virgule flottante. Ce dispositif fait aujourd'hui partie intégrante des processeurs.

## 5.3 Les mémoires externes

### *5.3.1 Rôle*

*A compléter*

### *5.3.2 Contenu*

*A compléter*

### *5.3.3 Classification*

*A compléter*

### *5.3.4 Paramètres de description et d'évaluation*

*A compléter*

| | Cassettes (streamer) | Disquettes 5"1/4[5] Double densité | Disquettes 5"1/4[6] Haute densité | Disquettes 3"½ Double densité[2] | Disquettes 3"½ Haute densité | Disques durs | Disques CD ROM | Disques CD | Disques DVD ROM |
|---|---|---|---|---|---|---|---|---|---|
| **Capacité** | 2 Go ↗ 8 Go **4 Go** | **360Ko** | **1,2 Mo** | **720 Ko** | **1,44 Mo** | **20 Go** | 600 Mo | 600 Mo | 17 Go |
| **Temps d'accès** | Pas de sens | 100 ms ↗ 600 ms **200 ms** | 100 ms ↗ 600 ms **200 ms** | 100 ms ↗ 600 ms **200 ms** | 100 ms ↗ 600 ms **200 ms** | 5 ms ↗ 20 ms **10 ms** | 80 ms ↗ 200 ms **100 ms** | 100 ms ↗ 200 ms **200 ms** | 100 ms ↗ 200 ms **200 ms** |
| **Vitesse de transfert**[6] | 500 K/s ↗ 3 Mo/s **2 Mo/s** | 30 Ko/s ↗ 150 Ko/s | 30 Ko/s ↗ 150 Ko/s | 30 Ko/s ↗ 150 Ko/s | 30 Ko/s ↗ 150 Ko/s | 1 Mo/s ↗ 20 Mo/s **10 Mo/s** | **5 Mo/s** | **5 Mo/s** | **10 Mo/s** |

5 En voie de disparition

6 moyenne et pas instantanée

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Prix du lecteur** | 7000 F ↗ 30.000 F | 2500 F | 2500 F | 2500 F | 2500 F | 5000 F ↗ 30.000 F | 4.000 F | Graveur 10.000 F | 10.000 F |
| **Prix du support** | 250 F↗ 700 | 10 F↗ 50 F | 10 F↗ 50 F | 15 F↗ 50 F | 15 F↗ 50 F | pas de sens | pas de sens | 50 F | Pas de sens |

En grisé, les supports aujourd'hui pratiquement disparus. (11/2001)

### *5.3.5 Panorama des dispositifs de mémorisation au sein d'un système informatique*

*A compléter*

## 5.4 Les périphériques d'entrée

*A compléter*

## 5.5 Les périphériques de sortie

### *5.5.1 L'écran*

*A compléter*

### *5.5.2 Les imprimantes*

*5.5.2.1 Fonction*

Périphérique de sortie, l'imprimante a comme rôle de transformer des *octets* reçus par l'ordinateur en traces sur du papier.

*5.5.2.2 Types d'échanges entre l'ordinateur et l'imprimante*

Avant d'aborder quelques paramètres de description et d'évaluation, il est essentiel de préciser comment s'opèrent les échanges entre l'ordinateur et l'imprimante et cela en évoquant plusieurs questions :

- que représentent les octets envoyés par l'ordinateur à l'imprimante ?
- comment l'imprimante est-elle capable de les transformer en signes sur du papier ?
- quel est le rôle (plus ou moins grand) de l'ordinateur dans la préparation de ce qui doit être envoyé à l'imprimante ?

5.5.2.2.1 Les échanges de type texte (et les imprimantes travaillant en mode texte)

Les octets envoyés par l'ordinateur à l'imprimante représentent des *caractères* : ainsi chaque octet est interprété par l'imprimante, par exemple à travers le code ASCII, comme représentant un caractère à tracer.

Si l'imprimante est capable de tracer sur le papier les caractères correspondant aux octets reçus, c'est qu'elle possède pour chaque caractère possible la représentation point par point du dessin à réaliser. Ainsi, recevant l'octet 65[7] (caractère A dans le code ASCII), l'imprimante consulte le dessin bitmap (point par point) qu'elle a pour ce caractère et imprime sur le papier le nuage de points ainsi trouvé.

7 Noté en décimal et non en binaire comme c'est réellement le cas, pour des questions de facilité de lecture.

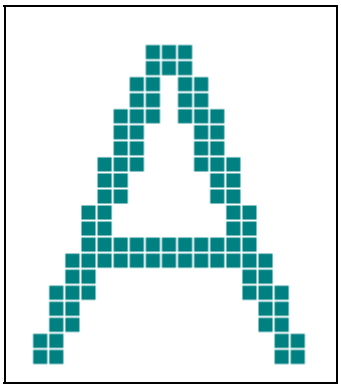


*Figure 5-38 : représentation bitmap d'un caractère*

C'est donc que l'imprimante dispose d'une mémoire morte dans laquelle ces informations pour le tracé sont enfermées. L'imprimante peut d'ailleurs posséder plusieurs polices différentes (Roman, Sans Serif, Courier...), dans plusieurs types d'impression (italique, élargi, gras, ...). Voici par exemple le type de représentation qu'elle pourrait posséder.

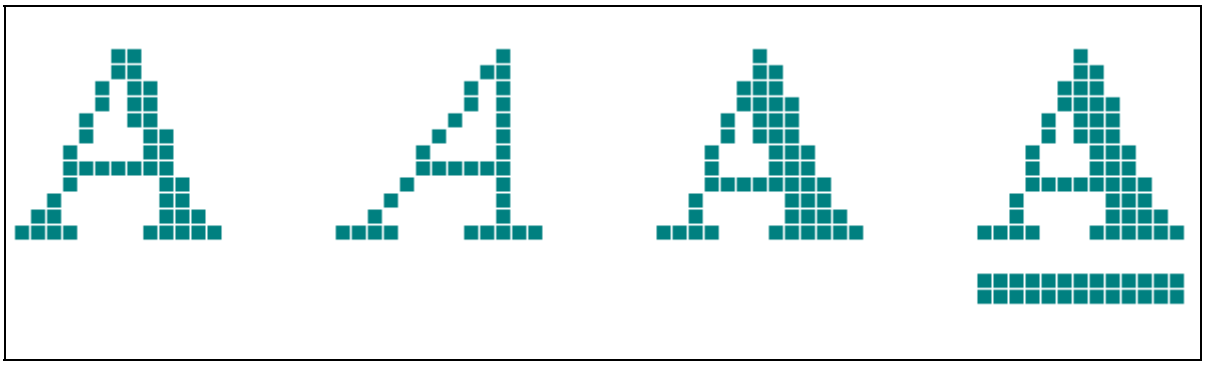


*Figure 5-39 : représentation bitmap de caractères en ROM d'une imprimante*

Signalons qu'il est possible d'adjoindre à la plupart des imprimantes travaillant en mode texte des cartouches de mémoire morte comportant les descriptions de polices supplémentaires.

Encore faut-il que l'ordinateur puisse signaler à l'imprimante les changements de police, de style (gras, italique,...), etc. à effectuer. Certains des octets envoyés à l'imprimante ne sont donc pas destinés à être imprimés. Ils ont pour mission d'indiquer à l'imprimante les changements de police ou de style attendus. Ce sont les premiers caractères du code ASCII (ceux portant les numéros de 0 à 32, que nous avons appelé les caractères de contrôle) qui vont jouer ce rôle d'indicateurs. Lorsqu'ils sont reçus par l'imprimante, rien n'est imprimé, mais cette dernière change en quelque sorte d'état : les caractères qui suivront seront imprimés dans une autre police, un autre style,...

Malheureusement, il n'y a pas un standard unique de ces caractères de contrôle, valable pour toutes les imprimantes travaillant en mode texte, et qui permettrait de signaler ces changements attendus.

Ainsi, sur les imprimantes de type EPSON (qui furent longtemps un standard en ce qui concerne les imprimantes matricielles), ce sont des suites de caractères commençant toujours par le caractère de code 27 (souvent appelé "Escape", et obtenu au clavier par l'appui sur la touche "Esc") qui jouent le rôle d'indicateurs pour ces changements. Par exemple, la suite de code 27 (Esc) et 52 (4) demandent à l'imprimante un passage en italique.
Sur la HP 500 (une des imprimantes à jet d'encre les plus répandues), c'est la séquence 27 (Esc) 40 (() 115 (s) 49 (1) 83 (S) qui provoque le passage au mode italique.

Lorsqu'une imprimante travaille en mode texte, l'ordinateur se contente donc de lui envoyer d'une part les octets correspondant aux caractères à imprimer, d'autre part les caractères de contrôle permettant les changements de polices, de styles,... C'est le plus souvent (mais pas toujours) lorsque l'ordinateur est équipé d'un logiciel de traitement de texte que du texte est ainsi envoyé vers une imprimante. Le logiciel de traitement de texte comporte alors les indications nécessaires concernant les caractères de contrôle correspondant à telle ou telle imprimante. Sur base du choix de police et de mise en forme souhaité par l'utilisateur et fourni au système (à travers le logiciel de traitement de texte), et sachant quelle est l'imprimante sélectionnée, les caractères de contrôle adéquats sont glissés au sein des caractères imprimés pour provoquer les choix désirés. Cet ensemble d'indications qui servent en quelque sorte d'interface logiciel entre le programme de traitement de texte et l'imprimante s'appelle un *pilote d'imprimante* (driver d'imprimante).

C'est la raison pour laquelle, lorsqu'on édite ou lorsqu'on demande l'affichage d'un fichier contenant un texte déjà préparé par un logiciel de traitement de texte et à destination d'une imprimante travaillant en mode texte, on y trouve, non seulement des caractères normaux, mais encore un tas de caractères inattendus et correspondant aux caractères de contrôle insérés dans le texte original.

Signalons encore que ce travail "en mode texte" est de plus en plus rare dans les environnements de travail actuels. Il était pourtant le plus souvent la règle pour les logiciels travaillant dans l'environnement MS-DOS sur PC.

5.5.2.2.2 Les échanges de type graphique (et les imprimantes travaillant en mode graphique)

Ici, les octets envoyés par l'ordinateur ne sont plus interprétés comme codant des caractères mais plutôt comme décrivant une portion de graphique à tracer sur le papier. L'imprimante interprète donc les octets comme la représentation bitmap (point par point) d'un graphique à imprimer sur le papier.

Disons d'emblée que dans ce cas c'est l'ordinateur qui se charge de la plus grande partie du travail. C'est lui, qui disposant simplement de la signification graphique (pour l'imprimante concernée) des différents octets (= quel octet envoyer pour que l'imprimante sorte telle configuration de points sur un petit espace de la feuille), va calculer quels octets envoyer pour obtenir textes et graphiques.

Il faut donc remarquer qu'ici, lorsque du texte (des caractères) doivent être tracés par l'imprimante, c'est l'ordinateur qui se charge de la tâche consistant à calculer quels octets "graphiques" envoyer pour que l'imprimante *dessine* les caractères souhaités.

Dans ce mode d'échange graphique, l'imprimante se contente donc de dessiner sur base des indications convenables correspondant aux octets reçus de l'ordinateur. Elle est alors capable de dessiner aussi bien des graphiques que du texte, à condition que l'ordinateur se charge de lui envoyer les octets de commandes de tracé souhaitées.

On verra ci-dessous que dans le cas des imprimantes à aiguilles ou à jet d'encre, la tête d'impression se déplace horizontalement le long du papier en y laissant un ensemble de points disposés verticalement. Ainsi, si le dessin et le texte figurant ci dessous doivent être tracés par l'imprimante

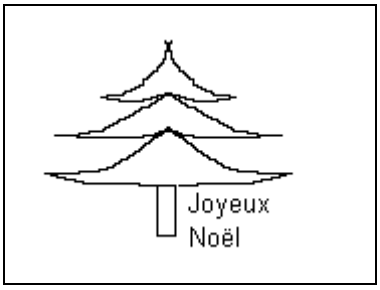



*Figure 5-40 : graphique à imprimer*

la tête d'impression fera un ensemble de passages horizontaux accompagnés, à l'issue du tracé d'une bande horizontale, d'une avancée verticale convenable de la tête (ou du papier). Ainsi, par exemple, lors d'un des parcours horizontaux de la tête, la tranche suivante sera tracée :

*Figure 5-41 : tranche graphique à imprimer*

En y regardant de plus près, on constatera que le dessin et les morceaux de caractères sont constitués de successions de points noirs ou blancs disposés verticalement. C'est la juxtaposition de ces colonnes de points qui constituera le dessin effectué.

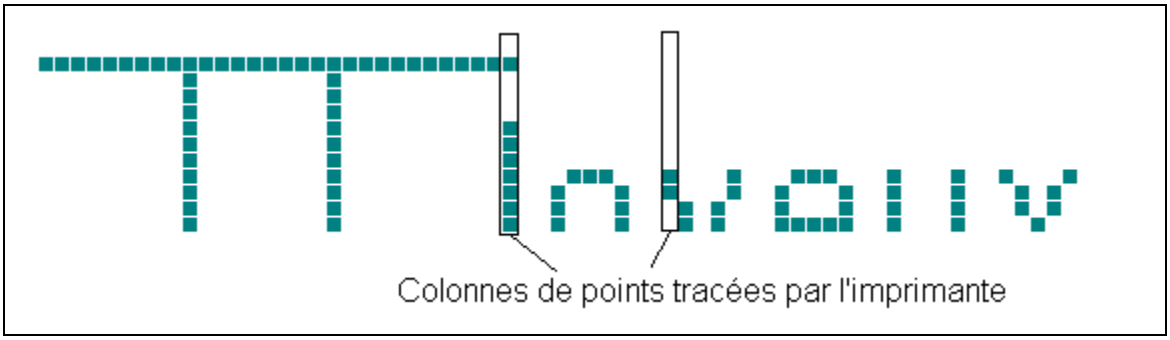



*Figure 5-42 : impression en colonnes*

C'est donc l'ordinateur qui, ici, va envoyer les octets représentant les diverses colonnes (avec les combinaisons de points noirs ou blancs) à l'imprimante, celle-ci se contentant d'aligner les colonnes de points ainsi commandées.

Les résultats obtenus sur l'imprimante correspondent donc essentiellement, non aux capacités de cette dernière, mais au travail préalable fait par l'ordinateur, donc aux possibilités des logiciels utilisés. Il faut savoir qu'un environnement comme Windows (sur les PC) travaille essentiellement de cette manière. Au point de vue des polices disponibles, on n'est plus du tout limité par celles dont l'imprimante posséderait en mémoire morte une description bitmap, mais uniquement par celles dont l'ordinateur possède la description.

?

Nous avons vu qu'il existait deux modes de codage des dessins : bitmap et vectorisé. Si l'ordinateur possède un mode de représentation des caractères à faire tracer par l'imprimante, quel est selon vous le mode de codage le plus approprié ?

Ajoutons encore que fréquemment, c'est d'une description vectorisée des différents caractères (dans différentes polices et selon différentes mises en valeur) que le système dispose. Une telle description vectorisée considère un caractère non comme un ensemble de points, mais comme un objet composé de segments, de courbes, etc.. Ainsi, par exemple, le caractère A peut être décrit par quelques grandeurs comme dans le schéma ci-après :.

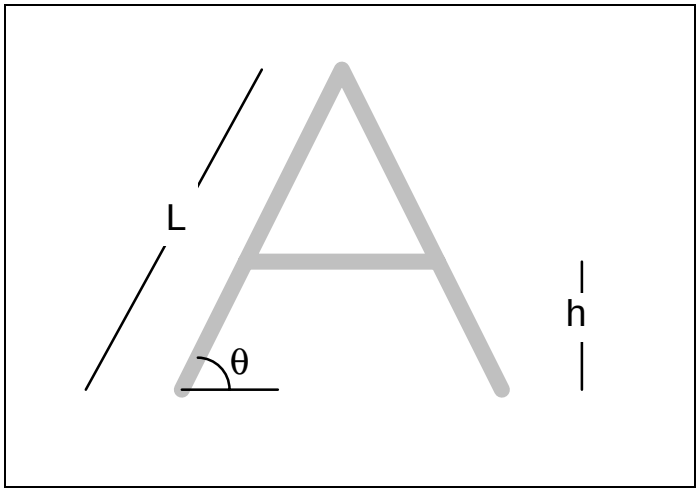



*Figure 5-43 : caractère vectorisé*

Disposant de ce type de description, il est aisé de passer à un tracé bitmap (point par point) qui sera aussi correct que possible, *quelle que soit la taille* du caractère à tracer : il suffit d'ajuster à chaque fois les longueurs L et h

*Figure 5-44 : agrandissement d'un caractère vectorisé*

Après être passé de cette représentation vectorisée à la représentation bitmap des caractères, dans la taille souhaitée, il suffit alors à l'ordinateur d'envoyer vers l'imprimante les octets correspondant à ces nuages de points.

Ainsi en est-il, sous Windows, des polices dites True Type : le système qui possède la description vectorisée de ces polices, en génère une description bitmap, tenant compte de la taille souhaitée, et envoie cette description à l'imprimante graphique.

Il est important de saisir que si l'on dispose seulement d'une représentation bitmap du caractère, dans une taille donnée

ce qui donne, vu de près

*Figure 5-45 : caractère bitmap*

et que l'on souhaite passer à d'autres tailles de caractères, le processus d'agrandissement va conduire à des résultats peu satisfaisants :

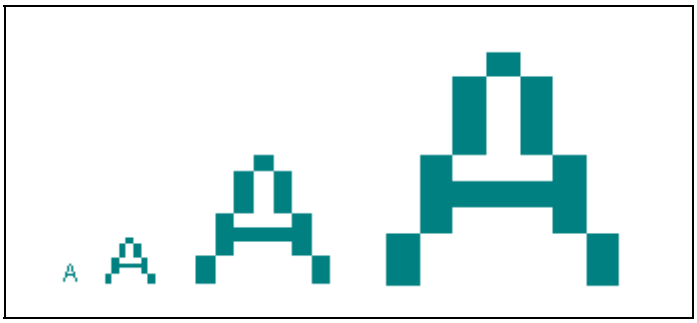



*Figure 5-46 : agrandissement bitmap*

Une autre solution envisageable, si l'on veut en rester à une représentation de type bitmap, est de disposer de cette représentation pour diverses tailles de caractères. Dans ce cas, il ne suffit donc pas que l'ordinateur possède la description point par point du caractère A, mais plutôt la description point par point du caractère A pour toutes les tailles souhaitées. On devine la quantité d'informations à stocker pour faire dessiner dans plusieurs tailles, pour différents styles, tous les caractères de plusieurs polices.

#### 5.5.2.2.3 Les échanges de type programme

Dans ce dernier cas les octets transmis par l'ordinateur constituent un programme (= des indications de traitement) qui sera exécuté par l'imprimante et lui fera produire une page. On devine aisément que dans ce cas l'imprimante est en réalité un ordinateur à part entière, disposant d'un processeur et d'une bonne capacité de mémoire vive pour stocker et exécuter les programmes ainsi reçus.

Nous aurons l'occasion dans la suite d'aborder la programmation et les langages de programmation. Pour faire court, ces langages permettent, avec un mode d'expression suffisamment proche du nôtre de décrire les traitement attendus par l'ordinateur. Un programme exprimé dans un tel langage doit cependant être traduit pour donner naissance à un programme "en langage machine" (= dans le langage du processeur concerné), le seul qui soit exécutable.

Plusieurs langages de description de page à l'intention de ce type d'imprimantes existent : citons par exemple PostScript ou PCL (Printer Control Language).

En ce qui concerne l'impression de texte, c'est, on s'en doute, l'imprimante, qui se charge de l'essentiel du travail. C'est elle, par exemple, qui dispose des descriptions vectorisées de diverses polices et des programmes de conversion de ces polices vectorisées en description bitmap. L'ordinateur peut se contenter de lui envoyer, à travers un programme respectant la syntaxe du langage de description de page choisi, le texte à imprimer, le type de police, la mise en valeur, la taille,... C'est l'imprimante qui interprète le programme reçu, génère les bonnes représentations bitmap sur base des descriptions vectorisées qu'elle possède, etc..

Signalons enfin que ce sont essentiellement les imprimantes Laser et jets d'encre qui travaillent de cette manière : une "laser" est donc avant tout un ordinateur (spécialisé dans le travail d'impression).

#### *5.5.2.3 Les paramètres descriptifs et d'évaluation d'un imprimante*

##### 5.5.2.3.1 Type des échanges

Le premier critère important est celui du type d'échanges permis : dans quel mode (texte, graphique, programme) l'imprimante peut-elle travailler ? Nous découvrirons ci-dessous les imprimantes matricielles, à jet d'encre et laser. Disons dès à présent que les deux premières familles travaillent souvent en mode texte et en mode graphique et que la dernière le fait en mode programme.

##### 5.5.2.3.2 Vitesse d'impression

Il s'agit évidemment d'un critère fort important : combien de caractères par seconde (CPS) une imprimante travaillant en mode texte est-elle capable d'imprimer; combien de minutes faut-il à une imprimante travaillant en mode graphique ou programme pour produire une page, ou mieux, combien de pages par minute (PPM) ce type d'imprimante peut-elle sortir ?

Nous disposons donc de deux unités : les CPS (caractères par seconde), pour les imprimantes de type texte, les PPM (pages par minute) pour les deux autres types.

##### 5.5.2.3.3 Résolution

C'est du nombre de points distincts qui peuvent être imprimés par unité de longueur (tant horizontalement que verticalement) qu'il s'agit ici. On parle également de "définition" ou de "densité" dans ce contexte. Cette résolution s'exprimera en points par pouce (en anglais Dot Per Inch : DPI). Plus ce nombre est élevé, plus la qualité du document imprimé sera remarquable.

##### 5.5.2.3.4 La taille de la mémoire tampon

Quel que soit le mode de relation entre l'ordinateur et l'imprimante, on devine aisément que si cette dernière avait seulement la possibilité de stocker un seul octet avant de l'imprimer, l'ordinateur perdrait un temps énorme à envoyer, un par un, les octets qu'à chaque fois l'imprimante traiterait.

Notons d'ailleurs que ce mode de fonctionnement serait absolument impossible pour le travail en mode programme.

Les imprimantes disposent donc en général d'une zone de mémoire vive qui sert de tampon (buffer en anglais) permettant de stocker plusieurs octets (représentant des caractères, des colonnes de points ou encore des instructions) en attente d'impression. Plus la mémoire tampon est importante, moins nombreux seront les échanges entre ordinateur et imprimante et dès lors plus rares seront les interruptions du travail de l'ordinateur pour l'envoi de paquets d'octets.

Ce sont, on le devine, les imprimantes travaillant en mode programme qui disposent de la plus grande taille de mémoire : on parle alors simplement de mémoire vive plutôt que de mémoire tampon.

##### 5.5.2.3.5 Autres paramètres importants

- Le prix et le prix des consommables; il faut être particulièrement attentif au coût de ces derniers : rubans encreurs, recharge d'encre, toner,... (Cf. ci-dessous).
- Le bruit qui est un facteur essentiel du confort d'utilisation.

- Le type de connexion physique avec l'ordinateur : parallèle ou série.

  Dans le cas d'un liaison série, les octets sont envoyés le long d'un seul fil, bit après bit. On peut donc imaginer qu'une série de 0 et de 1 voyagent le long du fil, l'un derrière l'autre, par groupes de 8.

*Figure 5-47 : liaison série*

  Disons qu'en réalité, il y a plus d'un fil pour prendre en compte des contraintes techniques (par exemple, l'imprimante doit pouvoir signaler certaines choses à l'ordinateur). Ce type de liaison série est compatible avec des distances assez importantes (l'imprimante peut être éloignée de l'ordinateur). Enfin, ces liaisons séries sont fréquemment utilisées pour relier à l'ordinateur autre chose qu'une imprimante (autre ordinateur, modem,...).

  Dans le cas d'une liaison parallèle ce sont les 8 bits constituant un octet qui seront envoyés ensemble le long de 8 fils de liaison et qui chemineront en parallèle

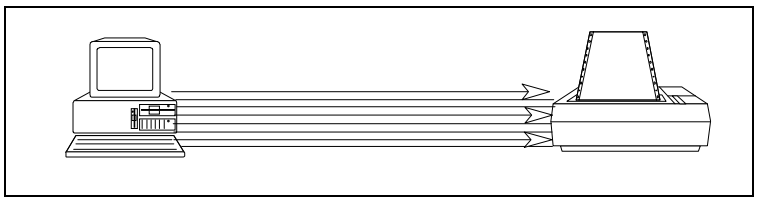


*Figure 5-48 : liaison en parallèle*

  Comme ci-dessus il faut signaler qu'il y a en réalité plus de 8 fils, pour des raisons techniques, mais le principe est bien celui décrit. Ce mode de liaison est généralement limité en longueur : on ne peut guère dépasser les 3 mètres de distance entre l'ordinateur et son imprimante.

  Beaucoup d'imprimantes sont connectables au choix en série ou en parallèle, évidemment à travers des prises physiquement distinctes.

- La taille du papier, la possibilité d'alimenter l'imprimante "en continu",... sont aussi des paramètres à prendre en compte.

#### *5.5.2.4 Les diverses variétés d'imprimantes*

##### 5.5.2.4.1 L'imprimante matricielle

Ce type d'imprimante est également appelé imprimante à aiguilles. Le principe en est simple : une tête d'impression munie d'une série d'aiguilles disposées verticalement se déplace horizontalement le long d'un ruban encreur lui même parallèle à la feuille de papier.

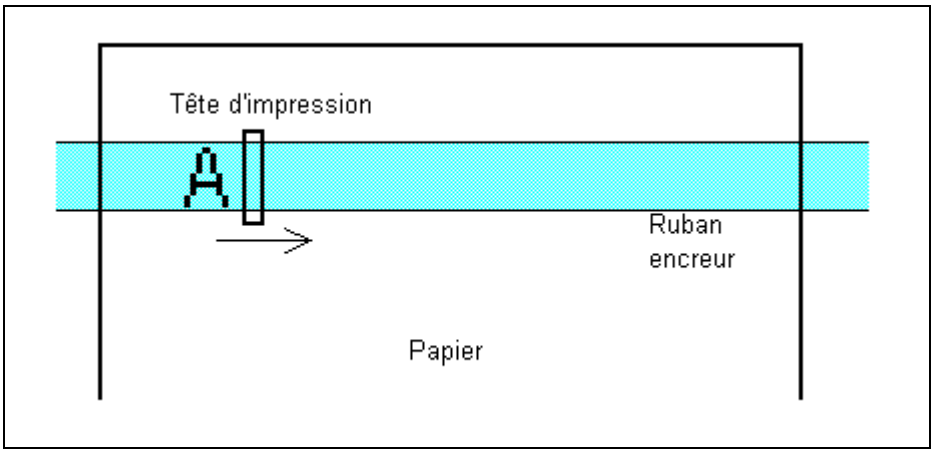



*Figure 5-49 : Imprimante matricielle*

A chaque micro-déplacement de la tête, certaine aiguilles viennent frapper le ruban qui laisse une trace sur le papier : on obtient donc une série de points disposés en colonne : certains restent blancs (les aiguilles correspondantes n'ont pas bougé), d'autres sont colorés (ils correspondent aux aiguilles qui ont frappé le ruban et le papier).

Vu de profil, le dispositif se présente donc comme suit :

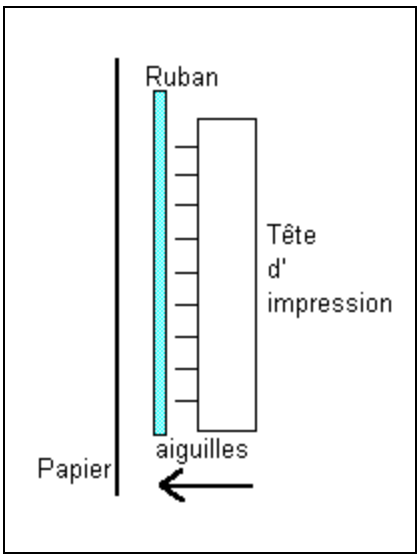



*Figure 5-50 : mécanisme d'impression à aiguilles*

Le nombre d'aiguilles est variable : on trouve des imprimantes à 9 aiguilles (ce sont bien entendu celles dont la résolution est la moins bonne), 24 aiguilles ou même davantage.

Beaucoup d'entre elles offrent la possibilité d'une impression brouillon (rapide mais de qualité médiocre) ou encore d'une impression de meilleure qualité (en anglais : Near Letter Quality : qualité presque courrier).

Les caractéristiques essentielles de ce type d'imprimante sont les suivantes :

- elles travaillent généralement soit en mode texte, soit en mode graphique;
- elles sont *bruyantes;*
- elles permettent des copies carbones des documents produits;
- leur vitesse en mode texte dépend de la qualité attendue, mais elle est relativement faible (180 à 300 CPS sont des valeurs maximales; en mode graphique, il faut en général plusieurs minutes pour produire une page;

- leur résolution est de 120 à 300 DPI;
- les modèles d'entrée de gamme sont *bon marchés*, de même que les consommables.

Ces imprimantes peuvent également travailler en mode graphique, et dans ce cas on n'est limité que par les possibilités logicielles de l'ordinateur : génération de polices, outils de création graphique.

☞ Il est important de signaler que les imprimantes matricielles sont en voie de disparition étant donné leur lenteur, le bruit généré par leur utilisation et leur médiocre définition.

5.5.2.4.2 L'imprimante à jet d'encre

Si le principe reste celui d'une tête d'impression qui se déplace horizontalement le long du papier en laissant des juxtapositions de colonnes verticales de points, il n'y a plus cette fois d'impact mécanique ni de ruban encreur. Le côté de la tête d'impression qui fait face au papier est muni d'une série d'orifices disposés verticalement et à travers lesquels des microgoutelettes d'encre sont envoyées directement sur le papier. Le nombre de ces micro-gicleurs est variable et la tête d'impression est évidemment accompagnée d'un réservoir d'encre. Le plus souvent, c'est toute la tête qui est remplacée lorsque le réservoir est vide.

Signalons encore que, le plus souvent, les imprimantes à jet d'encre sont proposées avec des capacités d'impression en couleur, même si les documents ainsi produits restent relativement cher (étant donné le coût des têtes-réservoirs).

Les caractéristiques essentielles de ce type d'imprimante sont les suivantes :

- elles travaillent généralement soit en mode texte, soit en mode graphique, soit, le plus souvent, en mode programme;
- elles sont *très silencieuses;*
- leur vitesse reste assez réduite, en général de plusieurs minutes par page (surtout en impression couleur);
- leur résolution est de 300 DPI ou plus, ce qui conduit à des documents d'excellente qualité;
- les modèles d'entrée de gamme sont *bon marchés*, mais *les consommables sont très chers* ce qui débouche sur un prix d'utilisation assez élevé.

5.5.2.4.3 L'imprimante Laser

Le principe est fondamentalement différent de ceux des deux types déjà vus. Ici, il n'y a plus de tête d'impression, mais un rouleau (chargé électrostatiquement positivement); un rayon laser (correctement orienté) vient frapper ce rouleau et neutralise les points touchés. Un dispositif de dépôt d'encre en poudre très fine (toner) et chargée positivement dépose alors des particules d'encre (chargée positivement), uniquement sur les points du rouleau préalablement neutralisés. Une feuille de papier chargée négativement vient ensuite au contact du rouleau et fixe les particules d'encre sur sa surface.

L'encre est ensuite fixée définitivement par chauffage.

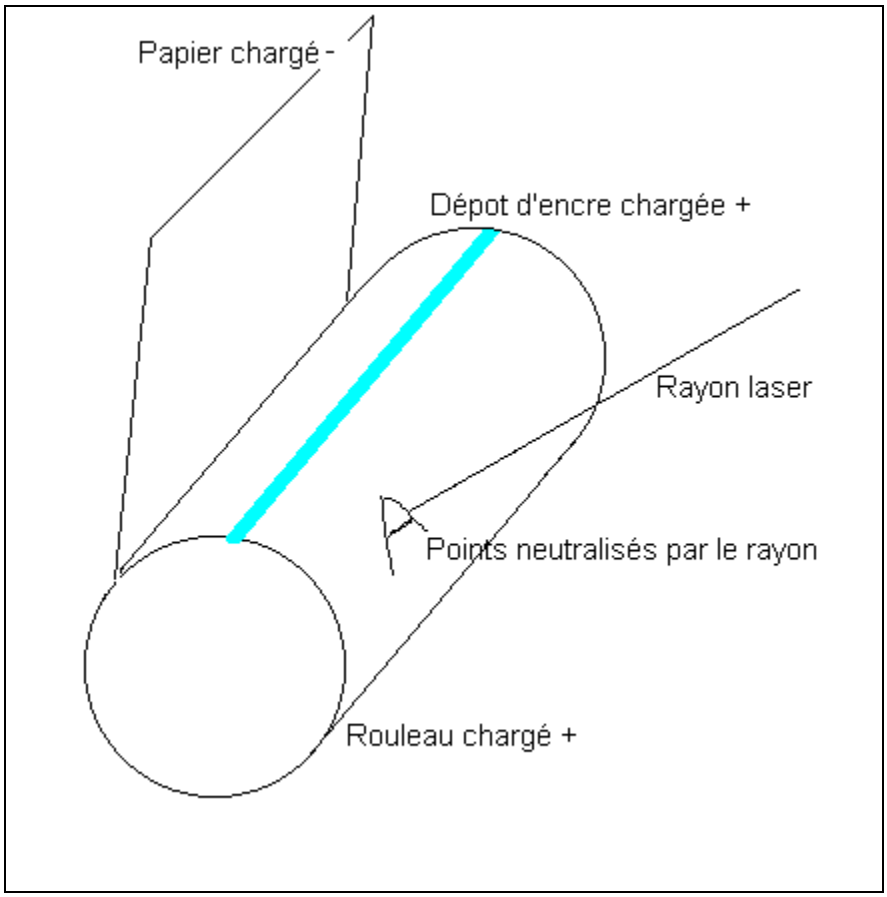



*Figure 5-51 : schéma de fonctionnement d'une imprimante laser*

Les imprimantes Laser travaillent essentiellement en mode "programme" : ce sont donc de véritables ordinateurs dotés d'un processeur et de mémoire vive et capable d'interpréter et d'exécuter les programmes de description de page reçus de l'ordinateur.

Leurs qualités fondamentales sont la rapidité et la qualité : elles conviennent bien lorsque la quantité de documents à imprimer est importante.

Les caractéristiques essentielles de ce type d'imprimante sont les suivantes :

- elles travaillent généralement en mode programme;
- elles sont *très silencieuses;*
- leur *vitesse* est importante : en général plusieurs pages par minute
- leur résolution est de 300 DPI au minimum et fréquemment de 600 DPI;
- elles sont *assez chères*, de même que les consommables et ne peuvent être amorties que lorsque les quantités imprimées sont importantes.

Voici à titre d'exemple le résultat d'une impression Laser :

**Exemple de sortie de texte et graphique sur une imprimante Laser.**
Le présent texte est en police ARIAL de taille 11 points, avec un passage en taille 7 points, *un passage en italique* et **un passage en gras**.
On peut également obtenir des caractères de très grande taille :

A,B,C

Petit exemple de graphique

Et voici, sous un petit graphique, un texte en police Times, 12 points, avec quelques mots en taille 7 points, *un passage en italique* et **un passage en gras**. Et pour continuer quelques mots dans la police Script, puis quelques uns dans la police Braggadocio, puis dans la police Colonna, puis dans la police DESDEMONA. Et il y en a des dizaines et des dizaines d'autres qui sont disponibles εν μοδε γραπηιθυε.

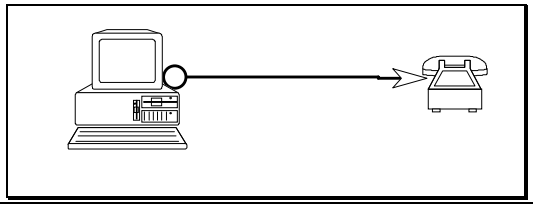


*Figure 5-52 : impression laser*

5.5.2.4.4 Tableau récapitulatif

| | **Matricielle**[8] | **Jet d'encre** | **Laser** |
|---|---|---|---|
| **Caractéristiques** | - 9 aiguilles à 24 aiguilles (et plus)<br>- bruyante<br>- copies carbone possibles<br>- peu coûteuse<br>- Texte (Draft ou NLQ[9]) + graphique<br>- grande largeur | - N/B ou couleur<br>- silencieuse<br>- bonne qualité<br>- Texte + graphique<br>- prix élevé des cartouches | - silencieuse<br>- rapide<br>- pour usage intensif<br>- travaille en mode programme (processeur intégré) |
| **Résolution** | 120 DPI ↗ 300 DPI (Graphique) | 300 DPI | 600 DPI |
| **Vitesse** | 180 CPS ↗300 CPS (Texte)<br>10 Min/page (Graph) | 120 ↗ 300 CPS (Texte)<br>1 ppm (Graphique) | 4 ppm ↗ 20 ppm |
| **Taille de la mémoire tampon** | 3 Ko ↗ 32 Ko | 8 Ko ↗ 32 Ko | 1 Mo ↗ 70 Mo (RAM) |
| **Prix** | 10.000 F ↗ 30.000 F | 12.000 F ↗ 30.000 F | 50.000 F ↗ 200.000 F et + |
| **Prix par page** | 1 F | 3F (N/B)<br>20 F ↗ 120 F (Couleur) | 2,50 F |

*Figure 5-53 : paramètres d'évaluation des imprimantes*

*5.5.2.5 Questions*

1. Le texte suivant peut il avoir été produit par une imprimante travaillant en mode texte ? Et en mode graphique ? Et en mode programme ?

   TEXTE BIEN HORIZONTAL

   Et le texte suivant ?

   Expliquez pourquoi.
2. Pensez-vous que l'on puisse parler de la résolution d'une imprimante travaillant en mode texte ? Pourquoi ?
3. Un de vos amis souhaite acquérir une imprimante. Quelles questions lui poseriez-vous pour l'aider à prendre sa décision.
4. Que représentent les octets envoyés par l'ordinateur à l'imprimante ?

8 Pratiquement disparues aujourd'hui.

9 NLQ : Near Letter Quality : Qualité Presque Courrier

5. Par quel(s) type(s) d'imprimante (texte, graphique, programme) le texte suivant est-il vraisemblablement produit ?

   Le texte écrit ici fait un mélange épouvantable de polices *différentes*. *Il en devient très difficilement lisible*.

# 6

# ... pour autant qu'on lui ait indiqué comment mener à bien ce traitement ...

## La programmation et ses langages

## 6.1 Introduction

### *6.1.1 Programmer ?*

Nous avons eu déjà l'occasion d'aborder l'activité de programmation (Cf. page 56); le schéma qui est au cœur de cette activité est le suivant

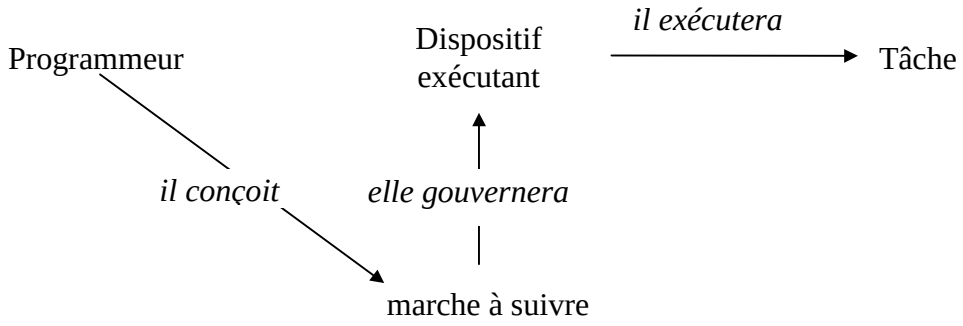



*Figure 6-1 : schéma de l'activité de programmation*

Rappelons que :

- "programmer", c'est rédiger une marche à suivre, un programme, à destination d'un "dispositif exécutant" qui, en suivant les instructions (= les ordres) figurant dans ce programme va mener à bien la tâche correspondante;
- dans notre contexte, un programme est une série d'indications d'**opérations** à effectuer sur des **données**, dans un certain **ordre**.

### *6.1.2 Le langage machine*

Nous avons découvert dans le chapitre précédent les principes qui gouvernent le fonctionnement de l'ordinateur et l'importance du "langage machine".

> On peut résumer ces principes en disant que des nombres (instructions) logés en mémoire centrale font agir le processeur sur d'autres nombres (données) également logés en mémoire centrale. Les nombres constituant le programme qui va faire effectuer des opérations par le processeur sont donc présents au même titre que les données dans la mémoire centrale. L'ordinateur est bien un **calculateur à programme enregistré** (en mémoire).

C'est dire que le schéma général présenté ci-dessus se particularise :

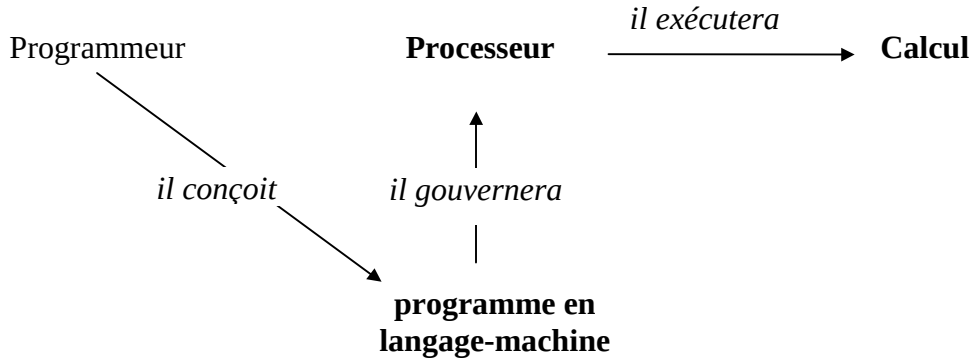



*Figure 6-2 : schéma de l'activité de programmation en langage machine*

Le dispositif exécutant à qui nous destinons les programmes écrits en langage machine est ici clairement le **processeur** et le type de tâche exécutée par ce dernier est toujours explicitement un **calcul** (addition, comparaison,... de nombres (codés en binaire)).

Nous avons compris également que seules les instructions formulées explicitement en binaire (code opératoire et adresses précises) et correspondant exactement aux possibilités du processeur pouvaient prendre place en mémoire pour constituer un programme exécutable (par le processeur).

### *6.1.3 Les limitations de l'expression en langage machine*

On imagine aisément la difficulté de retenir les codes opératoires (binaires) des diverses instructions, d'écrire les adresses mémoires (binaires) intervenant dans ces mêmes instructions.

De plus, nous l'avons vu dans l'exemple traité plus haut, une partie du programme (par exemple les adresses des instructions où conduisent les branchements) doit être réécrit si le programme est logé en mémoire dans un endroit plutôt que dans un autre (voir Figure 5-19 : rangement du programme en mémoire (1) et Figure 5-20 : rangement du programme en mémoire (2), page 87).

Enfin, un langage machine est propre à une famille de processeurs et tout le processus de programmation doit être repris si les mêmes traitements doivent être exprimés pour un autre processeur.

On aura compris que l'écriture des instructions d'un programme en langage machine n'est pas une sinécure et que repérer une erreur ou une modification à effectuer dans une longue succession de suites de 0 et de 1 constitue une tâche fastidieuse et infiniment éloignée de nos capacités.

Il est donc normal qu'on ait cherché d'autres manières d'exprimer les traitements attendus d'un ordinateur, en ne restant pas cantonné au langage machine.

C'est l'exploration des progrès réalisés dans ces modes d'expression des indications de traitements qui va à présent retenir notre attention : en d'autres termes nous allons découvrir l'évolution ("logique" plutôt que "chronologique") des langages de programmation.

## 6.2 Le triple point de vue sur les langages de programmation

Un langage de programmation est, nous allons le découvrir, un moyen d'expression à travers lequel le programmeur va fournir à l'ordinateur (et plus précisément au processeur) les indications de traitement qu'il souhaite.

Il va nous falloir des critères ou encore une grille d'analyse qui permettent de comparer ces divers langages entre eux. Cette "paire de lunette" à travers laquelle nous allons examiner ces divers modes d'expression et leur évolution est triple :

### *6.2.1 L'organisation des données à traiter et la manière de les désigner*

Nous le savons, les instructions (constituant un programme) vont commander au processeur d'agir sur des données (présentes dans les cellules -octets ou succession d'octets- de la mémoire centrale). Que sont ces données manipulées et *comment puis-je les désigner* au sein des instructions exprimant les traitements à effectuer : voilà la première question que je poserai à chacun des langages envisagés.

### *6.2.2 Les opérations permises sur les données et la manière d'exprimer ces opérations*

Quels types de traitements de ces données vais-je pouvoir commander au sein des instructions et *comment vais-je exprimer ces traitements* possibles : c'est la seconde question ou le second critère à travers lequel j'examinerai chacun des langages de programmation envisagé.

### *6.2.3 Les manières d'organiser l'exécution des instructions commandées*

Et enfin, une fois les données désignées et les opérations exprimées, il faut encore savoir comment on va pouvoir organiser la succession des opérations commandées. En effet, un programme c'est non seulement une suite d'instructions de manipulations de données, mais aussi les *structures organisatrices* qui indiqueront dans quel ordre les manipulations commandées auront lieu.

## 6.3 Retour sur le langage machine

A la lumière de ces critères, on peut réexaminer le langage machine (ou plutôt les langages machines, puisqu'il y en a autant que de type de processeurs).

### *6.3.1 L'organisation des données à traiter et la manière de les désigner, en langage machine*

Les données traitées sont disponibles sous forme de nombres (écrits en binaire) dans les cellules de la mémoire; pour manipuler une donnée, il faut indiquer expressément l'adresse de la cellule qui la contient. La désignation d'une donnée fait donc usage explicite de l'adresse de la cellule la contenant.

### *6.3.2 Les opérations permises sur les données et la manière d'exprimer ces opérations, en langage machine*

Ces opérations sont exactement celles dont le processeur est capable : additionner, comparer, soustraire,... Elles doivent être exprimées par le code opératoire (en binaire) figurant au sein des instructions. On colle donc complètement aux possibilités opératoires du processeur et on exprime ces opérations à travers le code opératoire.

### *6.3.3 Les manières d'organiser l'exécution des instructions commandées, en langage machine*

On sait que c'est le compteur ordinal qui désigne l'adresse de l'instruction qui va être ramenée (dans le registre d'instruction) puis exécutée.

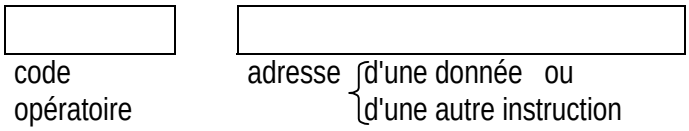
Le déroulement habituel voit, à la fin de chaque instruction, le compteur ordinal augmenter de 1, afin de pointer vers l'instruction suivante (= contenue dans la cellule dont l'adresse est la suivante). Dès lors, les instructions sont exécutées dans l'ordre où elles se présentent au sein de la mémoire, de manière parfaitement *séquentielle*.

Cette exécution séquentielle peut être brisée par les instructions de *branchement*, inconditionnel ou conditionnel. Dans ce cas, le compteur ordinal, au lieu d'augmenter de 1, effectue un saut pour pointer vers la cellule dont l'adresse est mentionnée au sein de l'instruction de branchement. On doit donc désigner explicitement l'adresse de l'instruction où l'on saute.

> Ce mécanisme de branchement est essentiel : c'est lui qui va permettre de répéter certaines suites d'instructions et de choisir à certains moment d'exécuter ou non telle séquence d'instructions.
>
> Le fait de devoir désigner de manière explicite, au sein de chaque branchement, l'adresse de l'instruction où l'on saute, rend l'écriture du programme particulièrement délicate puisque ces adresses ne sont connues que lorsque l'on sait à quelles adresses le programme tout entier est logé.
>
> Ce problème peut être partiellement résolu par l'existence d'instruction de branchement effectuant des sauts non vers des adresses absolues, mais vers des adresses relatives : non pas "donner au compteur ordinal la valeur 101" (et donc sauter à l'instruction d'adresse 101), mais plutôt "ajouter 010 au contenu du compteur ordinal" (et donc sauter à l'instruction dont l'adresse s'obtient en ajoutant 010 à l'adresse de l'instruction actuelle).

### *6.3.4 En résumé, pour le langage machine*

- Le dispositif exécutant est le processeur et la tâche effectuée est toujours une suite de calculs.
- Les instructions composant le programme sont explicitement écrites en binaire.
- Une instruction, c'est un code opératoire et une (ou plusieurs) adresses




- Les données sont rangées dans les cellules de la mémoire et désignées dans le programme par leurs adresses explicites (en binaire).
- Les opérations souhaitées sont exprimées dans le code opératoire. Elles correspondent exactement aux possibilités du processeur.
- On dispose de trois structures d'organisation du déroulement :
  - la séquence,
  - le branchement,
  - le branchement conditionnel.

  } *saut* : le programme se poursuit ailleurs qu'à l'instruction suivante

  mais, on doit désigner explicitement l'adresse des instructions où l'on "saute".

## 6.4 Un premier pas : les langages d'assemblage

On va le voir, le dispositif exécutant reste ici le processeur (comme pour le langage machine) et on continue donc à commander à ce dernier d'effectuer des calculs (et bien entendu des échanges avec la mémoire centrale). La Figure 6-2 : schéma de l'activité de programmation en langage machine, page 130, reste donc d'application.

Très peu de temps après l'apparition des premiers ordinateurs (et donc de l'écriture des premiers programmes en langage machine), apparaît l'idée de rédiger les programmes en remplaçant les constituants binaires de ceux-ci par des symboles : les langages permettant ce type d'expression s'appellent langages d'assemblage.

Voici à titre d'exemple ce que pourrait devenir le programme développé plus haut pour calculer max (a+b, c) dans le cas du modèle extrêmement simplifié d'unité centrale :

| Opérations souhaitées | Programme en langage machine | | Programme en langage d'assemblage |
|---|---|---|---|
| charger la cellule 01010 (10) dans le registre | 000 01010 | ⇨ | LOAD A |
| additionner la cellule 01011 (11) au registre | 010 01011 | ⇨ | ADD B |
| comparer le registre à la cellule 01100 (12) | 100 01100 | ⇨ | COMP C |
| brancher si code cond = 10 (registre>cellule) | 101 00101 | ⇨ | JUMPP FIN |
| charger la cellule 01100 (12) dans le registre | 000 01100 | ⇨ | LOAD C |
| sauver le registre dans la cellule 01101 (13) | 001 01101 | ⇨ | FIN SAVE D |

### *6.4.1 L'organisation des données à traiter et la manière de les désigner, en langage d'assemblage*

On désignera les cellules de la mémoire non plus par leur adresse explicite mais par un nom symbolique (A, B1, X, ...). Ainsi, dans le cas de l'exemple, la cellule d'adresse 01010 (10 en décimal) est notée A; les autres cellules contenant d'autres données seront désignées par B, C et D.

Notons que ces noms symboliques continuent à désigner des cellules de la mémoire.

### *6.4.2 Les opérations permises sur les données et la manière d'exprimer ces opérations, en langage d'assemblage*

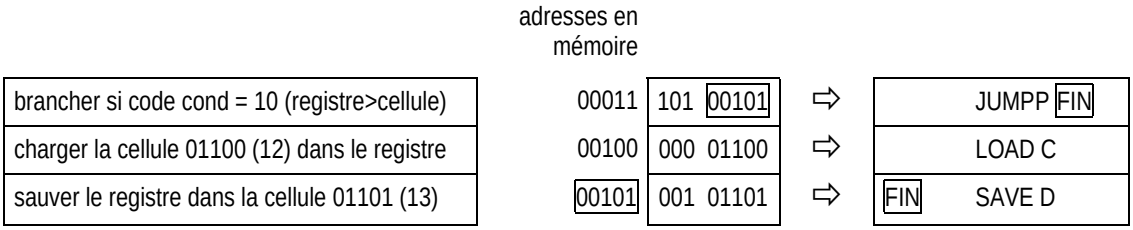
Les codes opératoires, difficiles à mémoriser, sont ici remplacés par des mnémoniques. Ainsi, l'opération de chargement dans le registre de données, à partir de la mémoire, est désignée par le mot LOAD (charger en anglais); l'addition du contenu d'une cellule au registre de données sera noté ADD, qui est tout de même plus aisé à mémoriser que le code opératoire 001. Il en sera de même pour les autres instructions : SUB, COMP, JUMP, JUMPP, JUMPN, SAVE.

Notons que les opérations commandées par ces mnémoniques continuent à être exactement celles dont le processeur est capable.

### *6.4.3 Les manières d'organiser l'exécution des instructions commandées, en langage d'assemblage*

On va garder les mêmes structures organisatrices que celles du langage machine : séquence , branchement et branchement conditionnel.

Mais, dans le cas du branchement (JUMP) ou des branchements conditionnels (JUMPP et JUMPN), l'adresse de l'instruction où il faut brancher (sauter) est remplacée par un nom symbolique, appelé aussi étiquette ou label. Et l'instruction où s'effectue le saut est précédée du même label. Ainsi, dans le cas de l'exemple :




On ne doit donc plus se soucier de manière explicite des adresses précises des instructions vers lesquelles s'effectuent les sauts ni, dès lors, des adresses occupées en mémoire par le programme. Ceci constitue une réelle facilité.

Notons que les manières d'organiser le déroulement des opérations par les langages d'assemblage collent encore tout à fait à ce qui se passait en langage machine et au fonctionnement même du processeur.

### *6.4.4 Une comparaison langage machine - langage d'assemblage*

| **Langage machine** | **Langage d'assemblage** |
|---|---|
| Il faut connaître explicitement les adresses des cellules où les données sont rangées. Ainsi, dans le cas de l'exemple :<br>a ➔ 01010 (10)<br>b ➔ 01011 (11)<br>c ➔ 01100 (12)<br>résultat ➔ 01101 (13) | On désigne les données à traiter par des noms symboliques, étiquettes des cellules les contenant. |
| Les opérations sont désignées par leur code opératoire (binaire) : 000, 001,... | Les opérations sont désignées par des mnémoniques : LOAD, SAVE, ADD, COMP, JUMP, JUMPP, JUMPN.<br>Ces opérations correspondent exactement aux possibilités du processeur. |
| Certaines instructions font effectuer des sauts qui rompent le déroulement séquentiel des instructions : les branchements et branchements conditionnels. | Il existe des mnémoniques pour les instructions de branchement (comme pour toutes les autres). Mais on ne doit plus indiquer à la suite de l'instruction de branchement l'adresse explicite de la cellule contenant l'instruction où l'on saute : on fait seulement figurer une désignation symbolique de l'instruction où l'on saute et cette dernière est précédée du même nom symbolique. |
| Ainsi, dans le cas de l'exemple :<br>101 adresse explicite où est logée l'instruction<br>001 01101 ⟲ | Ainsi, dans le cas de l'exemple :<br>JUMPP FIN<br>FIN SAVE D ⟲ |

Mais on colle encore parfaitement aux possibilités du processeur, en ce qui concerne l'organisation du déroulement du programme (séquence et branchement)

### *6.4.5 Les assembleurs*

Une fois le texte du programme rédigé, on va saisir celui-ci au clavier pour l'introduire dans la mémoire centrale de l'ordinateur. Ce programme prend donc place comme du **texte** au sein de la mémoire, les caractères le constituant étant codés selon les codes ASCII, ANSI ou d'autres. Pendant cette opération de saisie du texte, l'ordinateur est en général gouverné par un programme **éditeur de texte** qui permet la frappe et la modification de texte brut (sans mise en forme).

Ce texte codé ne constitue évidemment pas, quand on examine les octets successifs le constituant, le programme attendu en langage machine. Il va bien entendu falloir une traduction qui sur base du texte (codé) du programme générera les octets constituant le programme en langage machine (pour le processeur considéré). Ainsi, la chaîne LOAD sera reconnue et donnera naissance au code opératoire 000 et ainsi de suite pour les autres mnémoniques désignant les opérations.

Les symboles désignant les cellules contenant les données seront aussi remplacés par des adresses explicites, de même que les étiquettes figurant au sein des instructions de branchement.

Cette traduction est bien entendu effectuée par l'ordinateur gouverné à ce moment là par un programme de traduction qu'on appelle un **assembleur**.

Un assembleur est donc un programme qui sur base du texte d'un programme écrit dans le langage d'assemblage pour tel processeur (le programme source) génère le programme correspondant en langage machine (programme objet), pour le même processeur.

?

Comment, à votre avis, le programme assembleur a-t-il été créé ?

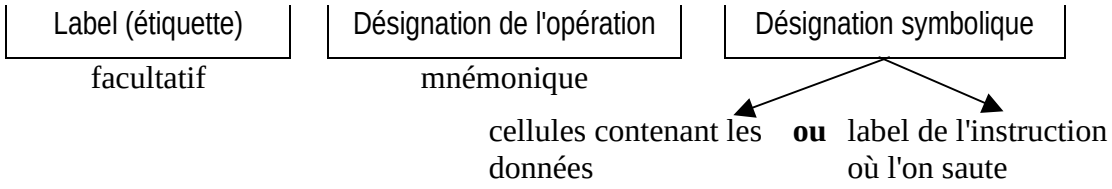
Cette traduction automatique postule évidemment que les règles d'orthographe et de syntaxe qui sont de mise pour l'écriture des programmes en langage d'assemblage aient été respectées, sinon la traduction n'est pas possible. Ainsi si l'on a mal orthographié le mot LOAD et qu'il est devenu LOOD, l'assembleur sera évidemment incapable d'assurer la traduction et de générer le programme en langage machine attendu. Il en va de même pour le respect des règles d'écriture des symboles pour les cellules ou les étiquettes des branchement.

Fort souvent, au lieu de dire qu'on écrit un programme en langage d'assemblage, on parle d'écriture en assembleur, confondant ainsi le langage et le programme qui assurera la traduction à partir de ce langage. On parle donc, improprement, de "programmation en assembleur" au lieu de "écriture de programmes en langage d'assemblage".

### *6.4.6 En résumé, pour les langages d'assemblage*

- Les opérations sont désignées par des mnémoniques; les cellules où sont rangées les données et les adresses des instructions vers lesquelles on effectue des branchements (sauts) font l'objet de désignations symboliques.

- Une instruction en langage d'assemblage est donc un texte comportant en général trois partie :




- Les cellules contenant les données à traiter sont désignées par des symboles : A, B1, ..., mais on continue donc à travailler sur le contenu de cellules mémoire.
- Les opérations permises sont exprimées par des mnémoniques. Elles correspondent exactement aux possibilités du processeur.
- On dispose, comme en langage machine de trois structures d'organisation du déroulement :
  - la séquence,
  - le branchement,
  - le branchement conditionnel.

  } *saut* : le programme se poursuit ailleurs qu'à l'instruction suivante

  mais, l'instruction où l'on saute reçoit un label (étiquette); c'est ce label qui figure dans l'instruction de branchement.
- Le langage d'assemblage colle encore parfaitement aux possibilités du processeur. Les programmes écrits en langage d'assemblage sont traduits en langage machine par un programme assembleur.
  Il faut bien comprendre que tant le langage d'assemblage que l'assembleur correspondant sont spécifiques à un type de processeur.
- On peut dire qu'on est débarrassé de l'écriture explicite en binaire, de la gestion explicite des adresses des cellules où prennent place les données, et de la gestion des adresses des cellules contenant les instructions vers lesquelles on saute.

## 6.5 Les langages évolués impératifs

Des pas décisifs vont être ici accomplis et qui vont définitivement nous éloigner du processeur et de ses contraintes. En d'autres termes, s'il va toujours s'agir de faire traiter des données par un dispositif, il faut se faire de ce dispositif une image qui n'est plus du tout celle du processeur et du fonctionnement, décrit précédemment, de l'unité centrale. En effet, ce que vont permettre les langages évolués, tant comme opérations que comme désignation des données ou manières de commander le déroulement du programme, est tellement éloigné des contraintes et fonctionnement de l'unité centrale que référer explicitement à cette dernière n'est plus utile. Il faut trouver une autre image du dispositif qu'on va commander à travers les programmes qu'on souhaite écrire en utilisant les langages évolués.

Bien entendu, ces programmes écrits en langages évolués, devront être traduits pour qu'on obtienne leur correspondant en langage machine. A ce niveau, on va retrouver le processeur et ses contraintes, mais ce niveau, les langages évolués vont justement permettre de ne plus s'en soucier.

### *6.5.1 Une autre représentation du dispositif exécutant dans le cas des langages évolués*

Nous allons apporter une description métaphorique de "que dois-je imaginer de ce qui se passe à l'intérieur de l'ordinateur, quand je conçois des programmes en langage évolué". La description réelle du fonctionnement de l'unité centrale n'est donc plus de mise ici : c'est autre chose pour laquelle on écrit les programmes en langage évolué. Cette autre chose, je l'appellerai tout au long de ce développement l'**exécutant-ordinateur**. C'est l'ordinateur transfiguré par sa compréhension apparente des indications que je veux bien lui fournir, exprimées dans un langage évolué.

#### *6.5.1.1 L'exécutant-ordinateur*

C'est le moment où jamais de relire le point 4.3 "Le point de vue du programmeur", page 56 et de se rappeler une fois de plus le schéma de l'activité de programmation :

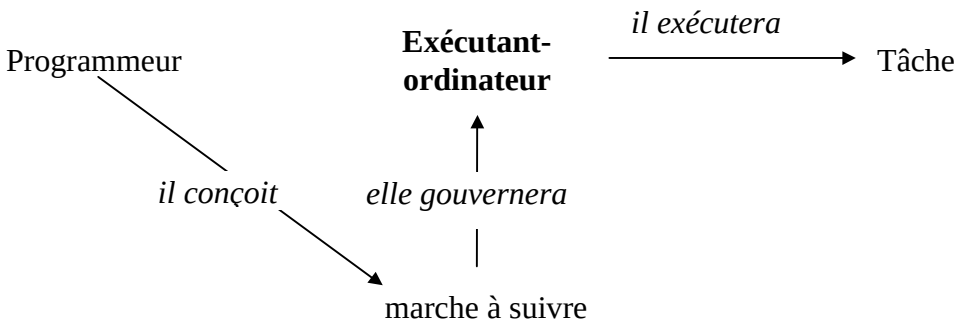



*Figure 6-3 : schéma de l'activité de programmation*

Jusqu'à présent, quand on écrivait des programmes en langage machine ou en langage d'assemblage, l'exécutant-ordinateur était l'unité centrale et particulièrement le processeur (tel que décrit au chapitre précédent); c'était le fonctionnement de cet ensemble qu'il fallait avoir en tête pour concevoir les programmes qui allaient le gouverner.

C'est d'une toute autre image qu'il faut se servir pour exprimer les programmes en utilisant les langages évolués.

#### *6.5.1.2 L'exécutant-ordinateur dans le cas des langages évolués : son environnement*

Je peux imaginer cet exécutant comme un genre de robot, logé à l'intérieur de l'ordinateur, dans un certain environnement, et obéissant à certaines instructions :

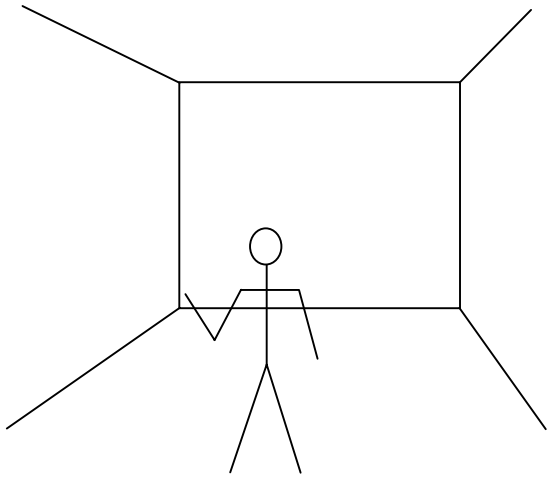


*Figure 6-4 : l'exécutant-ordinateur dans les langages évolués*

L'environnement de travail peut se décrire comme suit :

L'exécutant dispose d'une série de "casiers" étiquetés. Un tel casier est caractérisé par :

- L'étiquette qui y est attachée : inamovible, choisie par le programmeur et non manipulable par l'exécutant. Cette étiquette constitue le nom du casier et servira dans le texte du programme à désigner, soit le casier lui même, soit son contenu. Le langage utilisé apportera certaines contraintes sur les étiquettes possibles.
  Ainsi, en Pascal (qui est un langage de programmation évolué du type de ceux décrits ici) le nom d'un casier doit commencer par une lettre et ne comporter que des lettres (non accentuées), des chiffres et le symbole de soulignement _.
- Le contenu du casier : ce qui y transitera, c'est une donnée (nombre entier, nombre réel, caractère, chaîne (succession) de caractères).
- Le type du casier qui détermine le genre de donnée susceptible d'y prendre place (entier, réel, caractère, chaîne de caractères).

Je n'aborderai pas ici le concept de donnée de type booléen; je ne souhaite pas être complet ou exhaustif, mais seulement donner une idée des possibilités et contraintes de la programmation en langage évolué

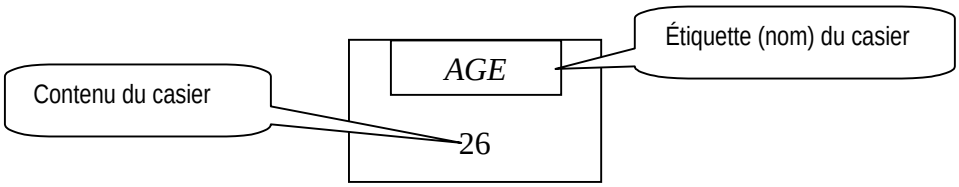



*Figure 6-5 : un casier de type entier*

Le vocable habituellement utilisé en programmation pour parler de ces casiers est celui de **variable** : une variable est donc d'un type donné (entier, réel, caractère, chaîne de caractères), elle porte un nom et possède un contenu. Comme on le verra le type et le nom d'une variable sont fixés tandis que son contenu est susceptible d'être modifié.

Il dispose également d'une énorme "malle à données", dont il pourra, le moment venu et sur les injonctions du programme le gouvernant, tirer n'importe quelle donnée de l'un quelconque des types cités (nombre entier, nombre réel, caractère, chaîne de caractères).

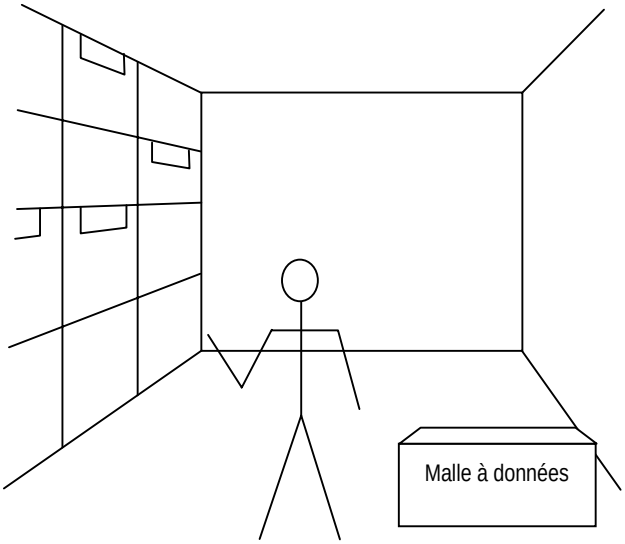



*Figure 6-6 : l'exécutant-ordinateur : les casiers et la malle à données*

De plus, l'exécutant peut "communiquer" avec l'extérieur. Son "local de travail" dispose en effet d'une porte-clavier à travers laquelle il peut recevoir des données (venant de

l'extérieur, fournie par un utilisateur, le moment venu) et d'une fenêtre-écran où il pourra, quand l'instruction correspondante lui demande, afficher des données.

Enfin, l'exécutant dispose d'une table de travail portant un certain nombre d'outils. Ces outils vont fournir, pour peu que l'exécutant y entre des données du type adéquat, une donnée nouvelle.

Ainsi il y a un outil SOMME (noté + sur le schéma), un outil DIFFERENCE (noté -), un outil NOMBRE AU HASARD,...

Parmi ces outils, il en est qui possèdent un statut particulier, ce sont ceux qui permettront au programmeur d'énoncer les conditions acceptables. Ce sont les outils qui fournissent le résultat de comparaisons de données : >, <, =, ...

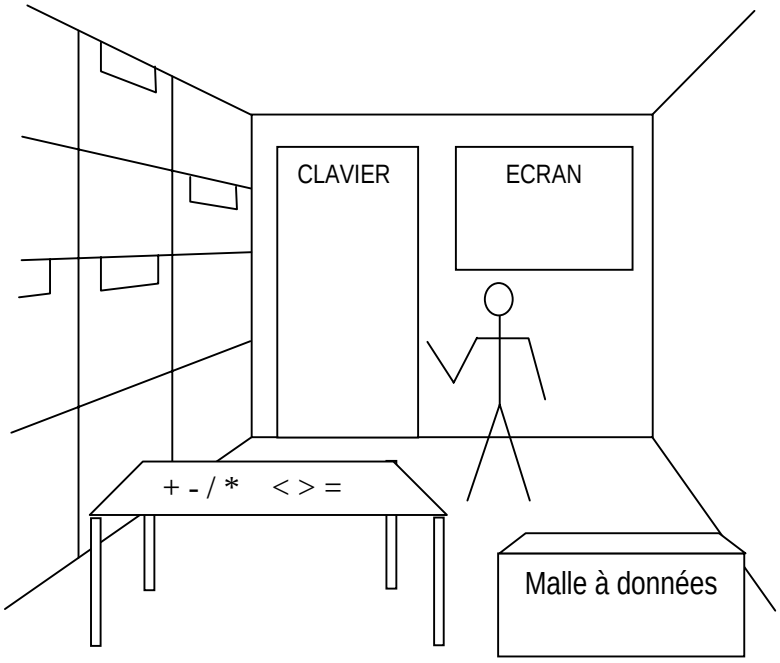



*Figure 6-7 : l'exécutant-ordinateur : l'environnement global de travail*

On est loin du processeur, des registres et des cellules mémoire. C'est pourtant ce portrait métaphorique de l'exécutant-ordinateur qu'il convient d'avoir en tête lorsqu'il s'agit d'exprimer des programmes dans un langage évolué (de type impératif)

#### *6.5.1.3 L'exécutant-ordinateur dans le cas des langages évolués : les instructions pour le faire agir*

A présent que nous savons dans quel environnement fonctionne cet exécutant-ordinateur, il nous reste à préciser les instructions que nous allons pouvoir lui fournir. Elles sont seulement au nombre de trois :

- L'instruction de remplissage d'un casier, qu'on appelle plutôt l'instruction d'**affectation.** On pourrait l'exprimer sous la forme :

  Place *telle donnée* dans *tel casier*

- L'instruction de réception de données de l'extérieur (à travers la porte-clavier); on l'appelle souvent aussi instruction de **lecture** ou encore **instruction d'entrée**. On pourrait l'exprimer sous la forme :

  Va à la porte, attends la donnée qu'on va t'y fournir et place la dans *tel casier*

parfois condensée en :

| Lis et place dans *tel casier* |
|---|

- L'instruction d'**affichage** pour l'extérieur (à travers la fenêtre-écran); on l'appelle souvent aussi **instruction de sortie**. On pourrait l'exprimer sous la forme :

| Affiche à la fenêtre *telle(s) donnée(s)* |
|---|

Il reste à préciser ce que recouvrent dans deux de ces instructions les termes "*telle donnée*". Une donnée pourra être écrite comme :

- une constante, comme dans

| Place *1* dans *(le casier) Total* |
|---|

ou

| Affiche *'Bonjour'* |
|---|

On notera que les constantes de type chaîne de caractères s'écrivent entourées d'apostrophes comme 'Bonjour'. Pourquoi cela pensez-vous ?

- le nom d'un casier (c'est évidemment le contenu qui est alors désigné), comme

| Place *(copie du contenu de) Somme* dans *(le casier) Total* |
|---|

ou

| Affiche *(copie du contenu du casier) Message* |
|---|

- le résultat fourni par un des outils disponibles (ce qu'on appellera une expression)

| Place *(copie du contenu de) Somme + 1* dans *(le casier) Somme* |
|---|

ou

| Affiche *(copie du contenu de) Somme - (copie du contenu de) Total* |
|---|

Il faut bien noter les éléments suivants :

- dans une instruction d'affectation, l'ancien contenu du casier (= de la variable) est remplacé par la donnée qu'on fait placer dans le casier; l'ancien contenu est perdu;
- comme le montrent les exemples ci-dessus, chaque fois qu'on mentionne un casier dans la description d'une donnée, ce casier sera seulement consulté et **copie** de son contenu sera prise; un casier consulté n'est donc pas vidé de son contenu.

Voici à titre d'illustration la manière dont ces actions, décrites ici de manière métaphorique, s'expriment dans deux langages de programmation connus, BASIC d'une part, PASCAL de l'autre :

| **Affectation** | **En PASCAL** s'écrit **:=** | **En BASIC** s'écrit = |
|---|---|---|
| Place 1 dans *Compteur* | *Compteur* := 1 | *Compteur* = 1 |
| Place *Total / N* dans *Taux* | *Taux* := *Total / N* | *Taux* = *Total / N* |
| **Lecture** | **En PASCAL**[1] : **readln** | **En BASIC** : **input** |
| Lis et place dans *Nom* | readln(*Nom*) | input *Nom* |
| **Affichage** | **En PASCAL** : **write** | **En BASIC** : **print** |
| Affiche 15 | write(15) | print 15 |
| Affiche *Total / N* | write(*Total / N*) | print *Total / N* |

[1] Rappelons bien qu'il ne s'agit pas ici d'être complet à propos de la description des langages envisagés : ils sont là seulement à titre d'illustration du propos.

#### *6.5.1.4 L'exécutant-ordinateur dans le cas des langages évolués : les conditions*

Nous venons de voir que nous pourrions commander à l'exécutant-ordinateur diverses actions. Nous pourrons également faire figurer dans les programmes des **conditions** qu'il sera capable de tester. Ce seront toujours des comparaisons de données permises par les outils déjà cités, comme >, <, =, ≥, ≠, ≤. Ainsi :

*Somme* < *Total*,
*Nom* = 'Dupont',
*Compteur* ≤ 10
...

Enfin des conditions plus élaborées pourront être construites grâce aux mots ET et OU :

(*Nom* = 'Dupont') ET (*Age* ≥ 25)
(*Compteur* ≤ 10) OU (*Somme* / 2 ≠ 0)

La dernière chose qui vaut la peine d'être ajoutée en ce qui concerne l'exécutant-ordinateur, c'est que c'est un "robot" parfaitement amnésique : sa seule "mémoire", à tout instant, c'est le contenu des casiers dont il dispose. Je ne pourrai lui parler, à travers les langages évolués, qu'à l'impératif présent.

### *6.5.2 L'organisation des données à traiter et la manière de les désigner, en langage évolué impératif*

On l'aura compris avec la description imagée qui précède, on est complètement affranchi du concept de cellule mémoire. Le concept pertinent dans le contexte des langages évolués impératifs est celui de **variable** (casier).

Les données manipulées sont placées dans des variables; ces dernières sont caractérisées par leur **nom** (choisi par le programmeur, avec des contraintes syntaxiques imposées par le langage), par leur **type** (lié au genre de données qui pourront y prendre place) et enfin par leur **contenu** (la donnée contenue au sein de la variable, on dit aussi la valeur de celle-ci).
Ce contenu peut être modifié (par les instructions d'affectation ou de lecture), mais à tout moment il est unique : une variable n'a, à chaque instant, qu'une seule valeur mais cette valeur peut être modifiée.

On imagine bien qu'évidemment, en dessous de chaque variable se cache une ou plusieurs cellules de mémoire. Mais il n'est plus pertinent de raisonner en terme de cellule : on utilisera les variables nécessaires au traitement de la tâche qu'on veut programmer en précisant pour chacune son nom et son type.

Ce concept de variable (et l'instruction d'affectation qui en permet l'usage) est absolument central dans les langages évolués impératifs; on appelle parfois ces derniers des langages à variable et à affectation.

### *6.5.3 Les opérations permises sur les données et la manière d'exprimer ces opérations, en langage évolué*

C'est ici que la distance avec les langages machine ou d'assemblage et avec les possibilités réelles du processeur est la plus marquée.

Trois instructions de base caractérisent les langages évolués impératifs :

- l'**affectation** : c'est l'instruction par laquelle on place une donnée dans un casier. Ce qui fait la véritable portée de cette instruction, c'est ce qui se cache ici derrière ce que nous appelons "donnée" : ce peut être une constante, la copie du contenu d'une autre

variable ou encore une expression faisant intervenir des constantes, des variables et des opérations plus ou moins sophistiquées.
On devine qu'une simple affectation notée

Place *Somme* + *X* dans *Somme*

va déboucher, lorsqu'on aura à la traduire en une série d'instructions en langage machine pour le processeur, sur un grand nombre de ces instructions :

- celles qui iront chercher les contenus des cellules correspondant à la variable *Somme* (c'est en général plusieurs cellules, 2 classiquement pour une variable de type entier), pour les disposer successivement dans le registre de données du processeur;
- celles qui feront de même pour la variable *X* en ajoutant successivement les contenus des cellules ramenées à celui du registre;
- celles qui iront replacer dans les cellules correspondant à *Somme* les résultats des additions effectuées.

Mais, de tout cela, le langage évolué permet de ne plus se préoccuper : on se contentera de demander que soient disponibles les deux variables *Somme* et *X* de type entier puis on pourra simplement écrire `Somme := Somme + X`, si l'on utilise Pascal ou `Somme = Somme + X` si l'on s'exprime en Basic.

Les outils de manipulation disponibles varient évidemment d'un langage à l'autre, la manière de noter ces opérations varie également, mais on trouve de manière classique :

- en ce qui concerne les nombres : des opérations ou des outils "simples" comme addition, multiplication, soustraction, division, génération d'un nombre aléatoire, valeur absolue, reste de la division entière, puissance,... et quelques fonctions plus sophistiquées comme les fonctions trigonométriques (sinus, cosinus,...), les fonctions trigonométriques inverses (arc sinus, ...), les logarithmes, etc.;
- en ce qui concerne les caractères, un outil permettant de leur associer le nombre qui les code dans le code ASCII (ou ANSI) et inversément un outil passant d'un nombre (entre 0 à 255) au caractère qu'il code; on trouvera aussi des outils permettant de passer au successeur ou au prédécesseur d'un caractère donné;
- en ce qui concerne les chaînes (successions) de caractères de nombreux outils sophistiqués permettant par exemple :
  - de chercher si une chaîne est présente au sein d'une autre (et à partir d'où),
  - de fournir la longueur d'une chaîne,
  - de recoller des chaînes,
  - d'insérer une chaîne au sein d'une autre à partir d'un endroit précisé,
  - etc.

- l'instruction de **lecture** ou **d'entrée** permet de saisir une donnée à partir d'un des périphériques d'entrée pour la faire placer dans une variable (d'un type adapté à la donnée reçue)

  L'exemple simplifié d'unité centrale décrit plus haut n'avait pas permis d'illustrer le mécanisme des entrées/sorties et donc la gestion des périphériques. Ceci nous entraînerait beaucoup trop

loin, comme par exemple à devoir présenter les interruptions. On se doute bien que dans la réalité des instructions permettent d'amener en mémoire des données reçues sur un périphérique d'entrée et à l'inverse, d'envoyer sur un périphérique de sortie des contenus de cellules mémoire.

Nous avons déjà indiqué plus haut que cette instruction d'entrée qui provoque la lecture d'une donnée à partir du clavier s'énonce *readln* en Pascal et *input* en Basic.

- l'instruction d'**affichage** ou **de sortie** permet d'envoyer sur un périphérique de sortie une donnée (constante, contenu de variable ou expression);
  pour un affichage à l'écran, cette instruction s'énonce *write* (ou *writeln*) en Pascal et *print* en Basic.

Retenons en tous cas que, en ce qui concerne les opérations de manipulation des données (ceux qui permettent d'écrire ce que nous avons appelé les expressions), ils sont nombreux et sophistiqués dans les langages évolués et terriblement éloignés des possibilités du processeur.

### *6.5.4 Les manières d'organiser l'exécution des instructions commandées, en langage évolué*

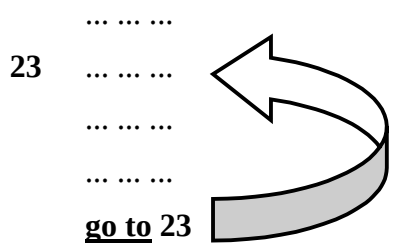
C'est ce troisième élément qui va donner naissance aux deux familles de langages évolués : langages à go to et langages structurés.

#### *6.5.4.1 Les langages à go to*

Il y a d'abord les langages évolués restant relativement proche des modes d'organisation du langage machine (et du langage d'assemblage), basé sur le déroulement séquentiel, le branchement et le branchement conditionnel : ce sont les **langages à go to** (ou à label);

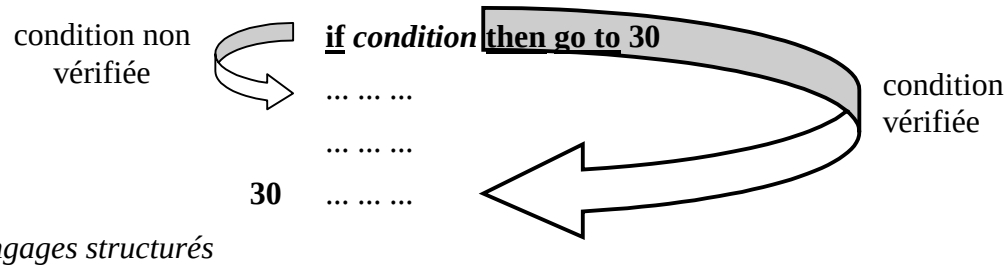
Les langages à go to proposent les modes d'organisation suivants :

- la **séquence** : les instructions sont exécutées à la suite l'une de l'autre (comme en langage machine ou en langage d'assemblage);
- le **branchement** : une instruction de branchement rompt le déroulement séquentiel en envoyant à une autre instruction du programme; elle prend généralement la forme suivante : *go to label*, où label est un numéro (ou un symbole) qui précède l'instruction où l'on veut "sauter"
  ainsi :




- le **branchement conditionnel** : le saut est ici conditionné par la vérification d'une condition; lorsque la condition énoncée est vérifiée, le saut commandé est effectué, sinon le déroulement se poursuit normalement à l'instruction suivante (de manière similaire à ce qui se passait avec les branchements conditionnels en langage machine) :

... ... ...




*6.5.4.2 Les langages structurés*

Ce sont des langages évolués proposant des modes d'organisation du déroulement des programmes plus sophistiqués et qui ne sont plus du tout calquées sur ce que permet le langage machine ou sur le fonctionnement du processeur.

Ces **langages structurés** proposent des structures d'organisation comme

- la **répétition** , par exemple sous l'un des formes

| | | |
|---|---|---|
| RÉPÉTER | ou | TANT QUE *condition* FAIRE |
| ... ... ... | | ... ... ... |
| ... ... ... | | ... ... ... |
| JUSQU'À CE QUE *condition* | | suite |
| suite | | |

- l'**alternative** , par exemple exprimée par :

| | | |
|---|---|---|
| SI *condition* ALORS | ou | SI *condition* ALORS |
| ... ... ... | | ... ... ... |
| ... ... ... | | ... ... ... |
| suite | | SINON |
| | | ... ... ... |
| | | ... ... ... |
| | | suite |

- l'**appel de procédure** , qui prend la forme

| | | |
|---|---|---|
| | ou | PROGRAMME ANNEXE EXPLICATIF (PROCEDURE) |
| ... ... ... | | |
| ... ... ... | | ... ... ... |
| INSTRUCTION COMPLEXE RÉFÉRANT À UN PROGRAMME ANNEXE | | ... ... ... |
| | | ... ... ... |
| ... ... ... | | ... ... ... |

- et, bien entendu, la **séquence** (ou l'organisation séquentielle)

*6.5.4.3 Les conditions*

On a pu constater que, quelle que soit la famille considérée, un concept important y apparaît, celui de condition (par exemple au sein des branchements conditionnels, des répétitions, des alternatives).

Ce concept de **condition** a déjà été abordé page 141 dans la description de l'exécutant-ordinateur. Il prend toute sa portée quand on sait qu'il sera essentiellement utilisé au sein des structures organisationnelles.

Une condition sera toujours constituée d'une ou de plusieurs **comparaisons de données** (par <, >, =, etc.); dans ce dernier cas, les comparaisons pourront être liées par les mots **ET** ou **OU**. Le mot **NON** pourra également être employé pour exprimer aisément le contraire (négation) d'une condition. L'essentiel est qu'on puisse décider au moment où elle est évaluée que la condition est vraie (vérifiée) ou fausse (non vérifiée).

C'est réellement la condition qui donne sens aux branchements conditionnels, aux répétitions et aux alternatives qui en font usage.

### *6.5.5 Les langages "à go to"*

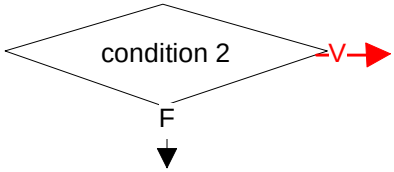
Comme tous les langages évolués impératifs, ils mettent en avant le concept de variable et les trois instructions d'affectation, d'entrée et de sortie.

L'organisation du déroulement fait usage de la séquence, du branchement et du branchement conditionnel.

Il est un mode de représentation des programmes exprimés dans les langages à go to qui permet d'en percevoir plus aisément la structure : **l'organigramme** ou l'ordinogramme.

Les instructions ordinaires (affectation, entrée, sortie) y sont enfermées dans des rectangles, les conditions dans des losanges et la séquence ou les branchements y transparaissent dans les flèches liant ces entités.

Voici à titre d'exemple un tel organigramme, accompagné de l'expression correspondante dans un langage à go to :

- la séquence est marquée par les flèches  et exprimée par la succession;
- les branchements conditionnels sont représentés par 

accompagnés des flèches qui font sauter ailleurs dans l'organigramme; ils s'expriment par l'instruction *if condition then go to label* (notée en rouge sur le schéma qui suit);
- les instructions sont enfermées dans des rectangles;
- les branchements sont rendus par les flèches  et traduits par les instructions *go to label* (notés en bleu, sur le schéma).

On notera que de plus toutes les instructions du programme sont numérotées.

En réalité, seules les instructions vers lesquelles des sauts sont effectués sont tenues de porter un label (numéro ou symbole) pour qu'on puisse y référer lors des sauts commandés par les branchements et branchements conditionnels.

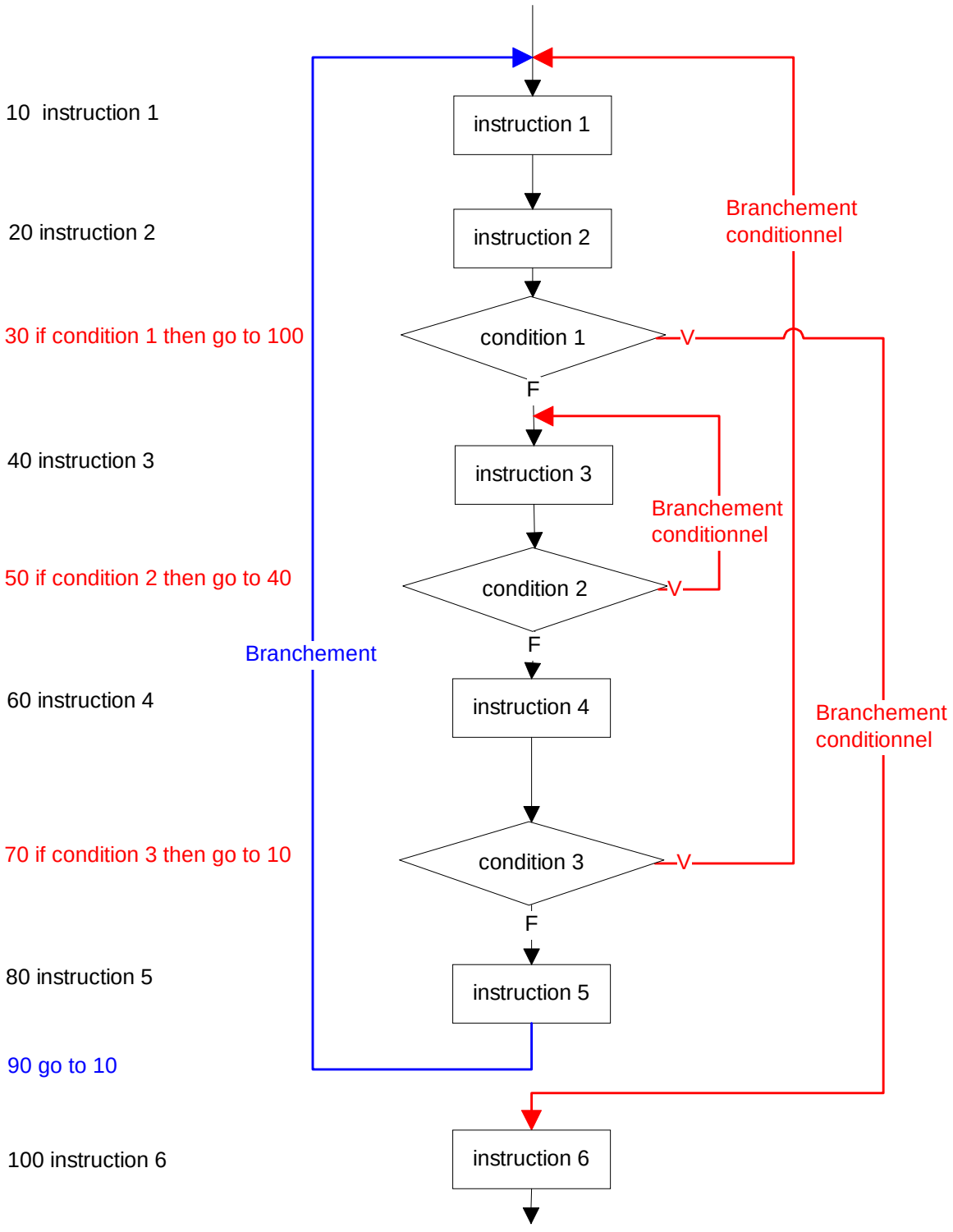



*Figure 6-8 : organigramme et programme correspondant dans un langage à go to*

### *6.5.6 Les langages structurés*

Les instructions organisant le déroulement du programme sont ici plus nombreuses : séquence, répétition, alternative, appel de procédure.

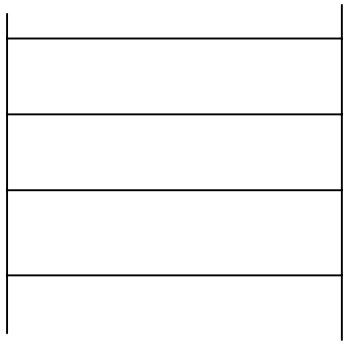
Ici aussi, un mode de représentation graphique permet de visualiser aisément l'organisation commandée par ces instructions organisatrices : les graphes de Nassi-Schneidermann (GNS). On y représente :

- la séquence :

  instruction;

  instruction;

  instruction;



- la répétition

  répéter

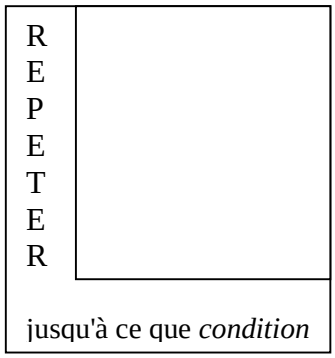
  ... ...

  jusqu'à ce que ...

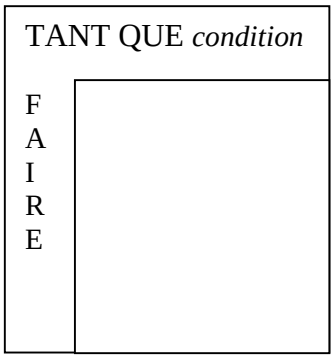
ou

  tant que ...faire

  ... ...




ou



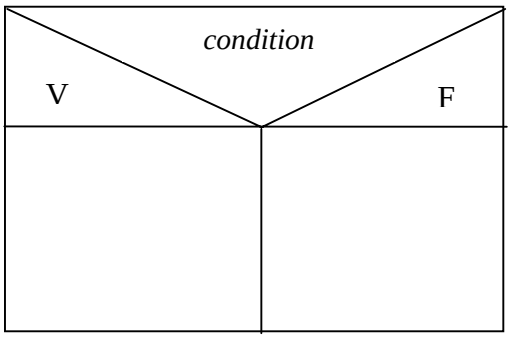

- l'alternative

  si ... alors

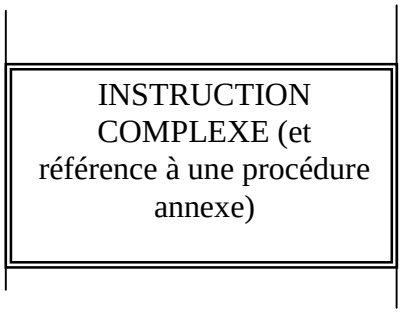
  ... ...

  sinon

  ... ...




- l'appel de procédure




Procédure annexe

Bien entendu, ces structures peuvent être imbriquées : par exemple au sein d'une répétition, on trouve une alternative, au sein de laquelle figure un appel de procédure, etc..

Voici à titre d'exemple un tel GNS, dans lequel figurent des instructions d'affectation et de sortie :

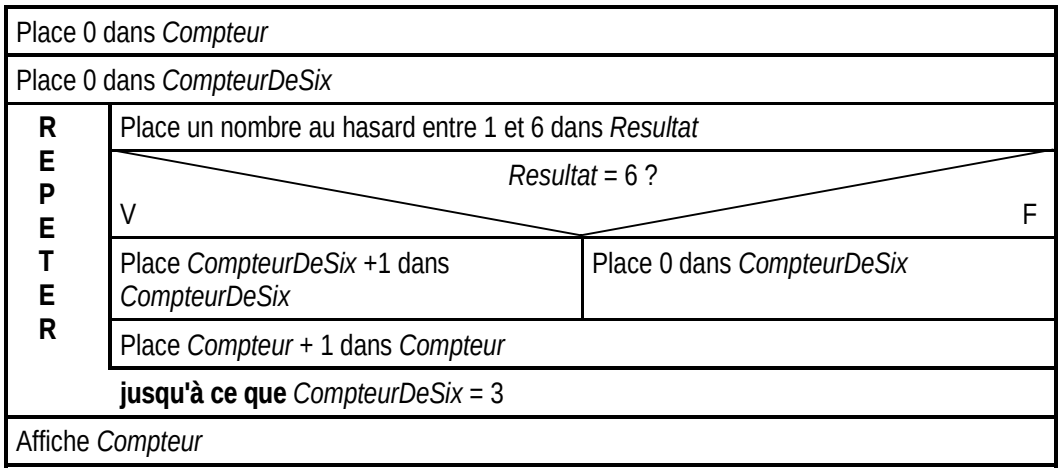



*Figure 6-9 : exemple de GNS*

On notera l'imbrication des diverses structures : au sein d'une séquence, on trouve une répétition; au sein de cette dernière, on trouve une séquence qui comporte une alternative, etc.

?

Que fait faire le programme ci-dessus ? Que fournit-il ? Que fait-il simuler ?

### *6.5.7 Un exemple*

Nous allons à présent, à titre d'illustration, programmer une tâche simple, en langage évolué, à go to d'une part, structuré d'autre part. La tâche consiste à lire une série de nombres réels puis à fournir d'une part la moyenne de tous les nombres lus strictement positifs et, d'autre part, la moyenne des nombres strictement négatifs.

Remarquons au passage que la tâche considérée est parfaitement anodine et qu'elle ne demanderait guère d'invention ou d'intelligence pour être menée à bien. Tout change évidemment lorsqu'il s'agit de la faire faire par l'exécutant-ordinateur, décrit pages 137 et suivantes.

#### *6.5.7.1 Recherche d'une stratégie*

On voit assez clairement que les actions seront les mêmes pour chacun des nombres à traiter :

- on lit la donnée à traiter
- si elle est strictement positive
  on la compte parmi les positives et on l'ajoute à la somme des positives
  sinon, si elle est strictement négative (les nulles sont donc ignorées)
  on la compte parmi les négatives et on l'ajoute à la somme des négatives

  Notons que les moyennes ne seront calculées que tout à la fin; ce qui sera ajusté à chaque fois qu'une donnée est lue, c'est la somme et le nombre de données (positives ou négatives).
- et on recommence avec la donnée suivante
- une fois les données toutes traitées, on calcule et on affiche la moyenne des positives (sur base de la somme des positives et du nombre de positives) puis on calcule et on affiche la moyenne des négatives (sur base de la somme des négatives et du nombre de négatives)

On devine que la lecture et le traitement des diverses données seront donc répétés par l'exécutant-ordinateur; mais il subsiste une questions centrale : comment signaler à ce même

exécutant que la succession des nombres à traiter est terminée. Entre humains, celui qui fournirait les nombres se contenterait d'indiquer, une fois le dernier traité, "c'est fini".

Mais ici, la répétition que nous commanderons va à chaque fois demander à l'exécutant de lire un nombre réel. Pas question, lorsqu'il reviendra une n-ième fois lire un réel (fourni au clavier par l'utilisateur) de subitement lui fournir "fin" ou "c'est fini" ou toute autre donnée qui ne soit pas un réel.

Nous allons dès lors utiliser un "tour de main" bien connu en programmation : nous demanderons à l'utilisateur de terminer sa liste de nombres réels par la fourniture d'un nombre réel particulier qui servira seulement à signaler que c'est terminé. Cette donnée "bidon" est de même nature (réelle) que les vraies données, mais elle vient à la fin et ne compte pas : ce n'est pas une donnée à prendre en compte, elle est seulement là pour arrêter le processus.

Cette donnée bidon, nous la ferons préalablement choisir par l'utilisateur et lui demanderons donc de la fournir avant que la lecture de la série des données ne commence. On arrêtera les traitements lorsque la donnée (finalement) fournie sera identique à la donnée bidon préalablement précisée.

Nous aurons de cette manière la condition d'arrêt : lorsque la donnée lue sera identique à la donnée bidon (qu'on retiendra, bien entendu).

#### *6.5.7.2 La liste des variables nécessaires*

Le concept de base au sein des langages évolués est celui de variable : il nous faut décider, sur base de la stratégie mise en lumière, quelles seront les variables, leur type et leur utilité :

- *Donnee* de type réel qui contiendra successivement chacune des données lues
- *Bidon* de type réel qui contiendra la donnée bidon fournie par l'utilisateur au début du processus; c'est lorsque la donnée lue (et contenue dans *Donnee*) sera identique à *Bidon* que la série de nombres sera considérée comme terminée
- *CompteurDesPositifs* de type entier qui contiendra le décompte des données strictement positives qui auront été lues
- *CompteurDesNegatifs* de type entier qui contiendra le décompte des données strictement négatives qui auront été lues
- *SommeDesPositifs* de type réel qui contiendra la somme des nombres strictement positifs
- *SommeDesNegatifs* de type réel qui contiendra la somme des nombres strictement négatifs

On pourrait s'étonner de la manière d'écrire (sans espace et sans accent) les noms des variables; les langages évolués mettent des contraintes sur les manières de dénommer les variables : beaucoup n'admettent pas l'espace ou le tiret au sein des noms de variables et refusent les lettres accentuées et autres ç cédille.

#### *6.5.7.3 Expression du programme dans un langage à go to : un premier essai*

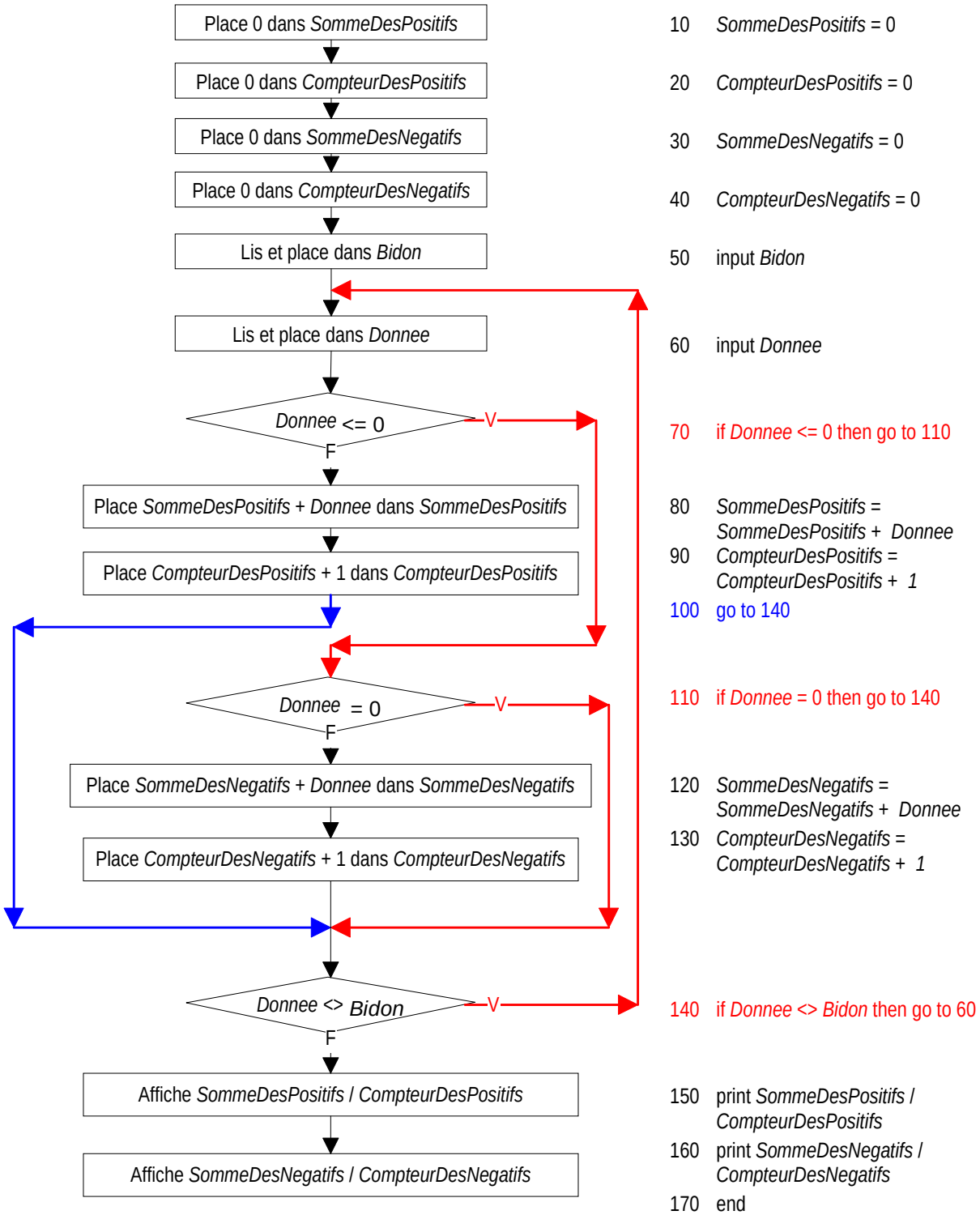
On le verra dans la suite, les langages à go to ont été parmi les premiers langages évolués et ont précédé les langages structurés.

La démarche suivie ici ne rend pas compte des pratiques de programmation de l'époque où l'on se lançait en général dans l'écriture du programme sans avoir bien réfléchi à la stratégie,

et sans avoir dressé la liste des variables nécessaires. La plupart des langages à go to ne demandaient d'ailleurs pas qu'on déclare la liste des variables qui allaient être utilisées au sein du programme : quand une variable devenait nécessaire, on la nommait et on l'utilisait sans plus de précaution.

De plus, les noms choisis pour ces variables étaient le plus souvent extrêmement sibyllins et souvent sans rapport avec le rôle que ces variables allaient jouer au sein du programme : ici, elles se seraient probablement appelées *A*, *B*, ...*N*, *I*, ...

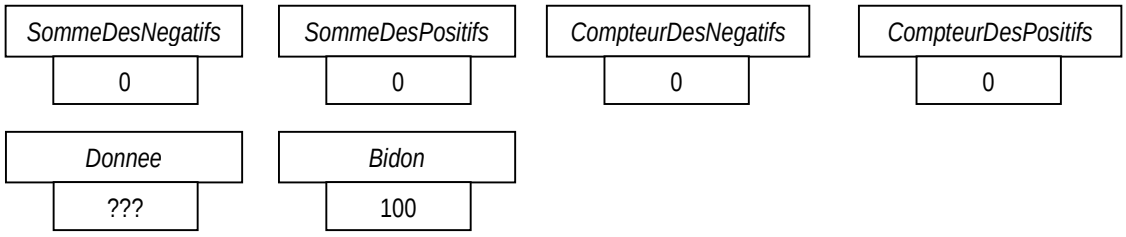
On peut proposer, côte à côte, l'organigramme et le programme en Basic (dans une ancienne version de ce langage qui met bien en avant le caractère "à go to") :



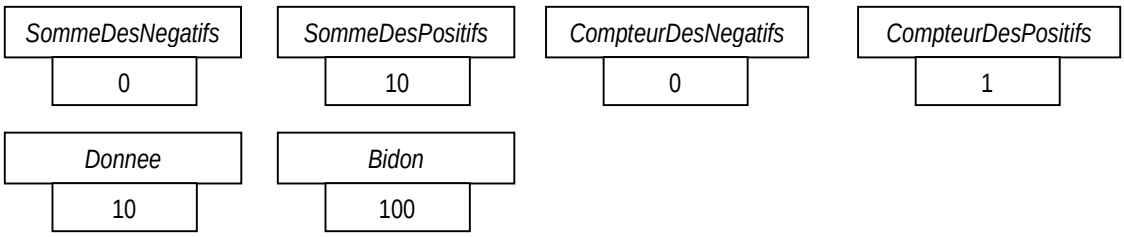
**?** Que pensez-vous de cette proposition ? Pouvez-vous tenter de suivre ce que serait l'exécution du programme, si l'utilisateur choisit 100 comme donnée bidon, puis qu'il fournit comme données ensuite 10, 2, -5 et 100.

On peut suivre l'évolution du contenu des variables au fur et à mesure de l'avancement du programme :

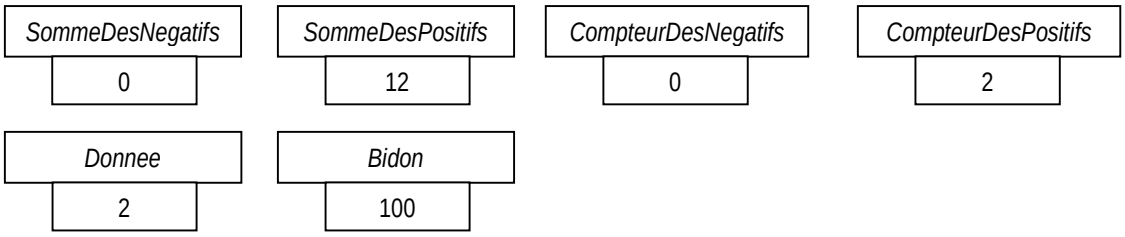
- juste **avant** la première lecture de la première donnée (en ligne 60 du programme)



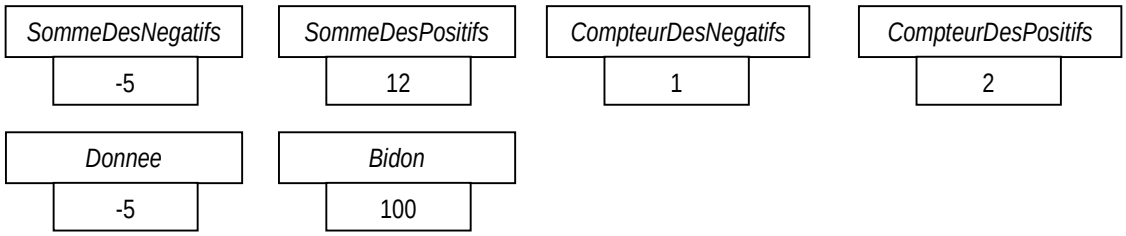

- juste **avant** la lecture de la seconde donnée ( 2ème passage en ligne 60 du programme)



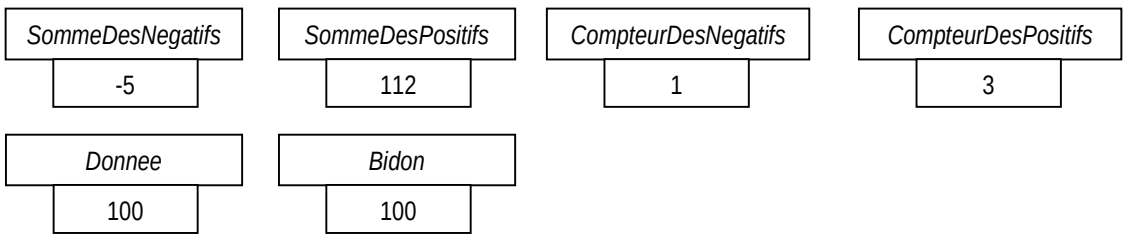

- juste **avant** la lecture de la troisième donnée ( 3ème passage en ligne 60 du programme)




- juste avant la lecture de la quatrième donnée ( 4ème passage en ligne 60 du programme)




- juste avant le test de la ligne 140 qui suivra la lecture et le traitement de la quatrième donnée

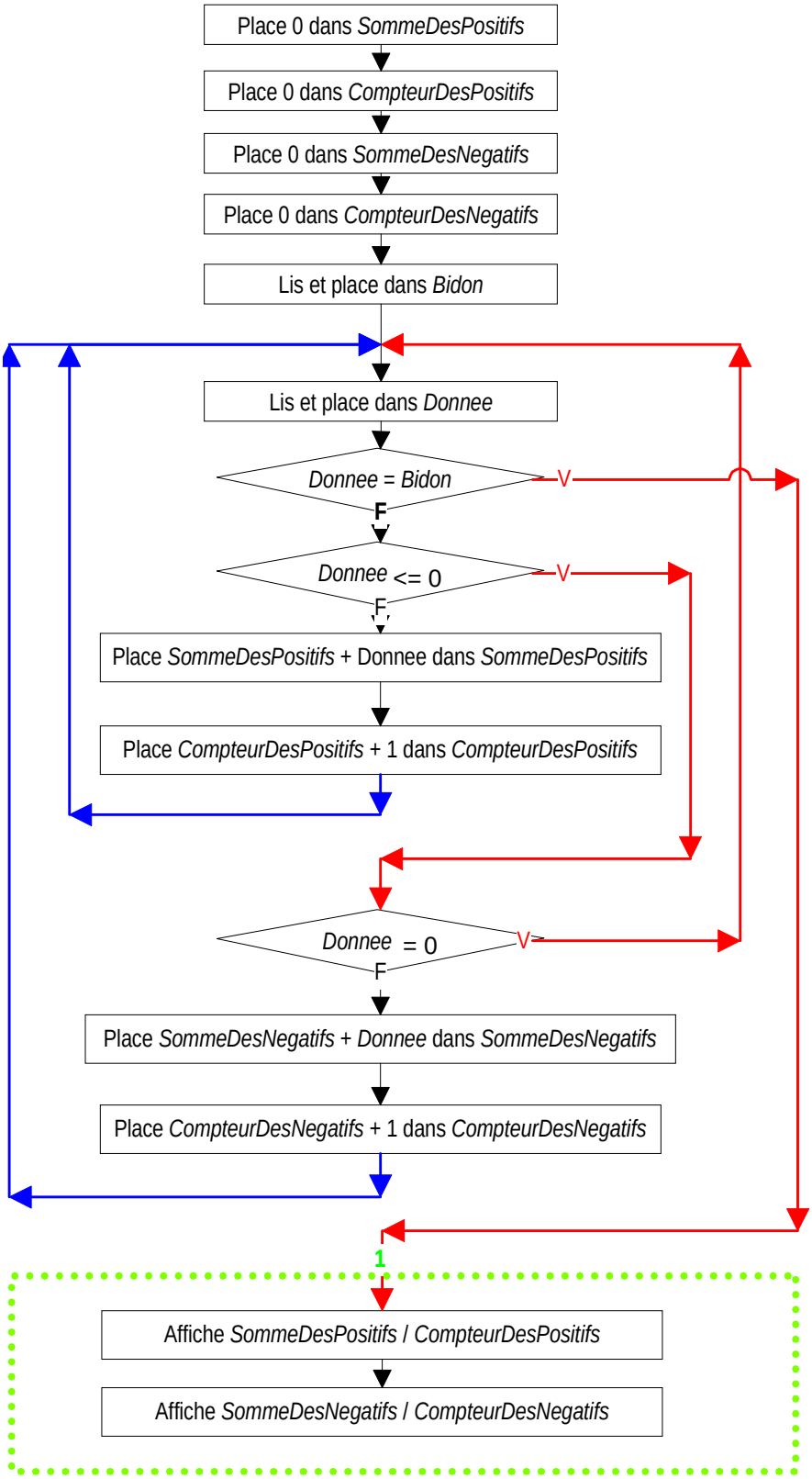



- on affichera alors comme moyenne : -5 pour les négatifs (ce qui est correct), mais 112/3 soit 37.33 pour la moyenne des positifs, ce qui est absurde (puisque les données à prendre en compte étaient 10 et 2).

Ce premier essai comporte donc malheureusement une grossière erreur : la comparaison de la donnée lue à la donnée bidon vient trop tard : lorsque la donnée lue est identique à la donnée bidon, elle est traitée exactement comme les vraies données alors qu'elle devrait seulement signaler la fin du processus.

*6.5.7.4 Expression du programme dans un langage à go to : un second essai*

On peut proposer dès lors :




```
10  SommeDesPositifs = 0
20  CompteurDesPositifs = 0
30  SommeDesNegatifs = 0
40  CompteurDesNegatifs = 0
50  input Bidon
60  input Donnee
70  if Donnee = Bidon then go to 160
80  if Donnee <= 0 then go to 120
90  SommeDesPositifs = SommeDesPositifs + Donnee
100 CompteurDesPositifs = CompteurDesPositifs + 1
110 go to 60
120 if Donnee = 0 then go to 60
130 SommeDesNegatifs = SommeDesNegatifs + Donnee
140 CompteurDesNegatifs = CompteurDesNegatifs + 1
150 go to 60
160 print SommeDesPositifs / CompteurDesPositifs
170 print SommeDesNegatifs / CompteurDesNegatifs
180 end
```

On notera les éléments suivants :

- Un certain nombre de variables doivent être initialisées au tout début du programme : on fait placer 0 dans *SommeDesPositifs, SommeDesNegatifs, CompteurDesPositifs CompteurDesPositifs*

  Il faut bien saisir pourquoi ces initialisations sont nécessaires. Que se passerait-il, si *CompteurDesPositifs* contenait au tout début 25, par exemple ?

- Cette fois, la dernière donnée lue, celle qui est identique à la donnée bidon lue au tout début, n'est plus traitée; elle provoque seulement l'affichage des moyennes.

  On devine que dès que les organigrammes deviennent un peu longs et comportent de nombreux branchements conditionnels, le fil du déroulement de l'exécution devient difficile à suivre. Les branchements et branchements conditionnels transforment le parcours en une sorte de "jeu de piste" dans lesquels ces branchements et branchements conditionnels peuvent envoyer d'un endroit à n'importe quel autre.

Il subsiste pourtant encore une erreur dans le programme ainsi conçu. Voyez vous laquelle ? Que donnerait l'exécution avec une donnée bidon qui serait égale à 100 et une suite de données lues ensuite qui serait 2, 4, 100 ?

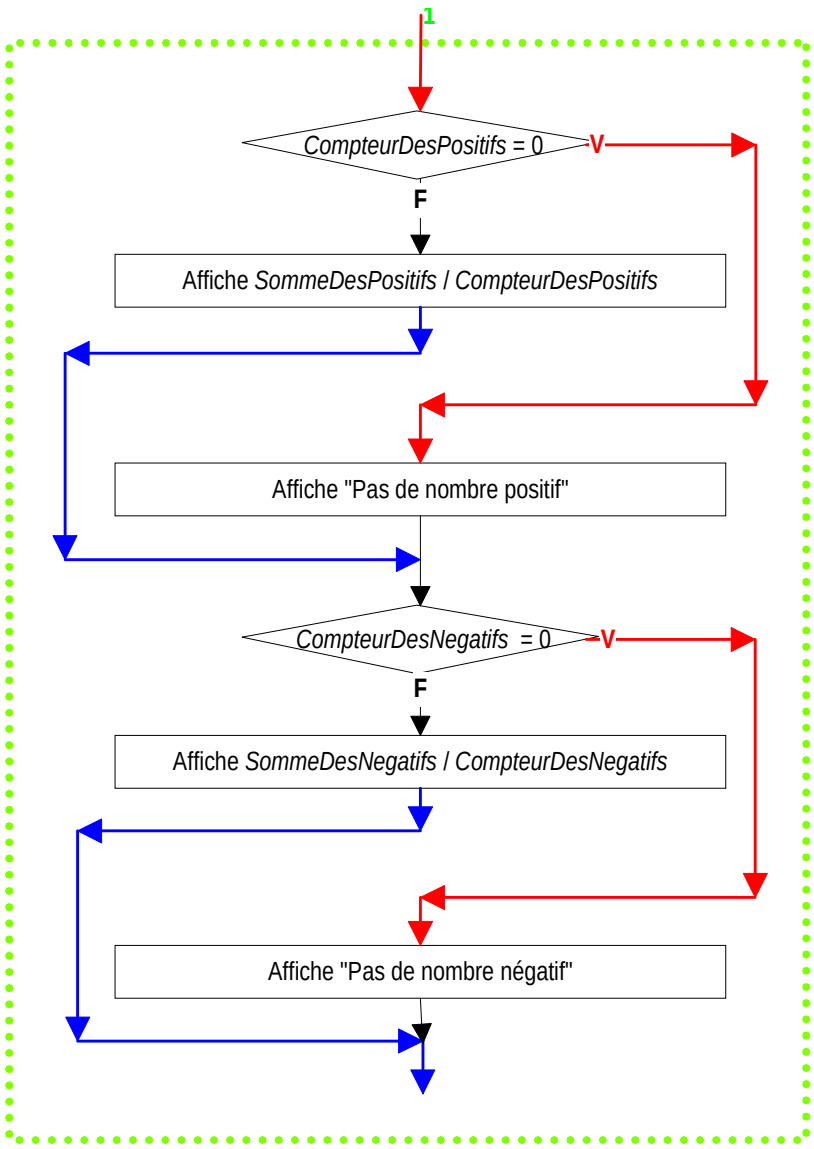
- Il est possible que dans la succession des données à traiter, il n'y ait que des données positives ou que des données négatives. Dans ce cas, à la fin du programme, lors du calcul des moyennes, un problème surviendra à cause d'une division par 0. En effet, si, par exemple, aucune donnée strictement négative n'a été lue, on se retrouve à la fin avec *CompteurDesNegatifs* qui vaut 0; en effet, on ne passe jamais par les lignes 130 et 140 du programme et *CompteurDesNegatifs* reste inchangé et garde donc la valeur 0 qu'on lui avait donnée à la ligne 40. Et dès lors, à la ligne 170, on commande une division par 0 en demandant le calcul de *SommeDesNegatifs / CompteurDesNegatifs*.

Si l'on veut remédier à cette erreur, il est indispensable de remplacer la partie encadrée en pointillés à la fin de l'organigramme précédent par l'organigramme qui suit.

Il faut bien entendu modifier les lignes 160 et 170 du programme pour y traduire le morceau d'organigramme rajouté.

*6.5.7.5 Expression du programme dans un langage à go to : troisième et dernier essai*

Ne figure ci-après que la fin révisée de l'organigramme; le début de l'organigramme précédent reste valable :

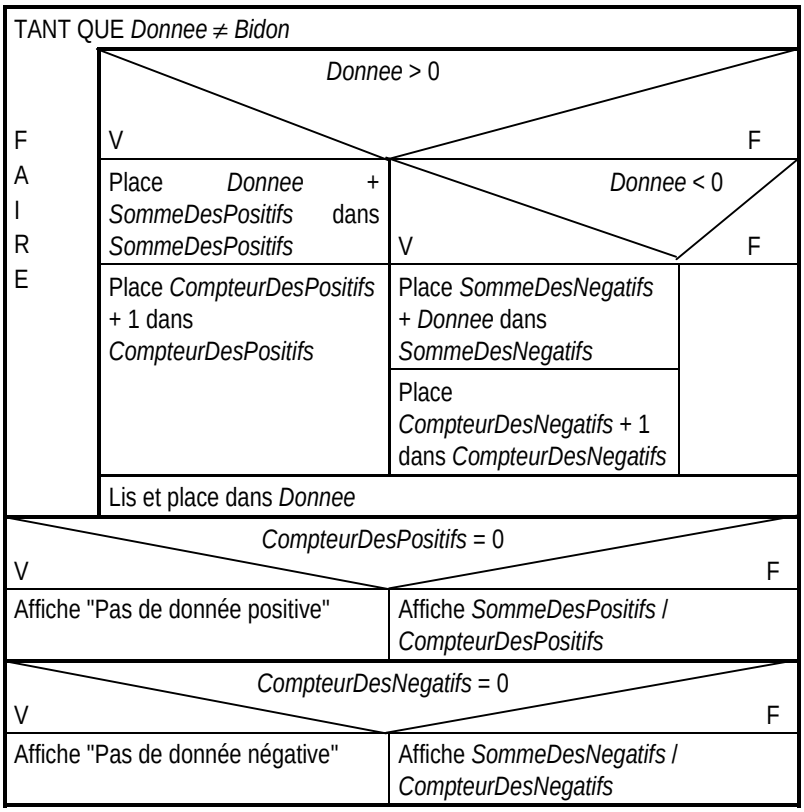



```
160  if CompteurDesPositifs = 0
     then go to 166
162  print SommeDesPositifs /
     CompteurDesPositifs
164  go to 168
166  print "Pas de nombre
     positif"
168  if CompteurDesNegatifs =
     0 then go to 174
170  print SommeDesNegatifs /
     CompteurDesNegatifs
172  go to 180
174  print "Pas de nombre
     négatif"
180  end
```

### *6.5.7.6 Expression du programme dans un langage structuré*

Nous disposons déjà de la liste des variables. Il nous reste à écrire le GNS correspondant au traitement souhaité

| |
|---|
| Place 0 dans *SommeDesPositifs* |
| Place 0 dans *CompteurDesPositifs* |
| Place 0 dans *SommeDesNegatifs* |
| Place 0 dans *CompteurDesNegatifs* |
| Lis et place dans *Bidon* |
| Lis et place dans *Donnee* |




Et voici l'expression en Pascal du contenu de ce GNS :

```
program MOYENNE;
var   SommeDesNegatifs, SommeDesPositifs, Donnee, Bidon : real;
      CompteurDesNegatifs, CompteurDesPositifs : integer;

begin
SommeDesPositifs := 0;
CompteurDesPositifs := 0;
SommeDesNegatifs := 0;
CompteurDesNegatifs := 0;
readln(Bidon);
readln(Donnee);
while Donnee <> Bidon do
      begin
      if Donnee > 0 then
            begin
            SommeDesPositifs := SommeDesPositifs + Donnee;
            CompteurDesPositifs := CompteurDesPositifs + 1;
            end
      else
            if Donnee < 0 then
                  begin
                  SommeDesNegatifs := SommeDesNegatifs + Donnee;
                  CompteurDesNegatifs := CompteurDesNegatifs + 1;
                  end;
      readln(Donnee);
      end;
if CompteurDesPositifs = 0 then
      writeln('Pas de donnée positive')
else
```

```
      writeln(SommeDesPositifs / CompteurDesPositifs)
if CompteurDesNegatifs = 0 then
      writeln('Pas de donnée négative)
else
      writeln(SommeDesNegatifs / CompteurDesNegatifs)
end.
```

On peut sans doute mieux apprécier la différence d'expression entre langage "à go to" et langage structuré en plaçant côte à côte les textes des programmes en Basic et en Pascal :

**Expression en Basic**

```
10   SommeDesPositifs = 0
20   CompteurDesPositifs = 0
30   SommeDesNegatifs = 0
40   CompteurDesNegatifs = 0
50   input Bidon
60   input Donnee
70   if Donnee = Bidon then go to 160
80   if Donnee <= 0 then go to 120

90   SommeDesPositifs = SommeDesPositifs +  Donnee
100  CompteurDesPositifs = CompteurDesPositifs +  1
110  go to 60
120  if Donnee = 0 then go to 60

130  SommeDesNegatifs = SommeDesNegatifs +  Donnee

140  CompteurDesNegatifs = CompteurDesNegatifs +  1

150  go to 60

160  if CompteurDesPositifs = 0 then go to 166
162  print SommeDesPositifs / CompteurDesPositifs
164  go to 168
166  print "Pas de nombre positif"
168  if CompteurDesNegatifs = 0 then go to 174
170  print SommeDesNegatifs / CompteurDesNegatifs
172  go to 180
174  print "Pas de nombre négatif"
180  end
```

**Expression en Pascal**

```
program MOYENNE;
var   SommeDesNegatifs, SommeDesPositifs, Donnee, Bidon:
       real;
      CompteurDesNegatifs, CompteurDesPositifs : integer;

begin
SommeDesPositifs := 0;
CompteurDesPositifs := 0;
SommeDesNegatifs := 0;
CompteurDesNegatifs := 0;
readln(Bidon);
readln(Donnee);
while Donnee <> Bidon do
      begin
      if Donnee > 0 then
            begin
            SommeDesPositifs := SommeDesPositifs + Donnee;
            CompteurDesPositifs := CompteurDesPositifs + 1;
            end
      else
            if Donnee < 0 then
                  begin
                  SommeDesNegatifs := SommeDesNegatifs +
                  Donnee;
                  CompteurDesNegatifs := CompteurDesNegatifs
                  + 1;
                  end;
      readln(Donnee);
      end;
if CompteurDesPositifs = 0 then
      writeln('Pas de donnée positive')
else
      writeln(SommeDesPositifs / CompteurDesPositifs)
if CompteurDesNegatifs = 0 then
      writeln('Pas de donnée négative)
else
      writeln(SommeDesNegatifs / CompteurDesNegatifs)
end.
```

### *6.5.8 Compilateur ou interpréteur*

Les programmes écrits en langage évolué (on dit aussi "langage de haut niveau", par opposition aux langages d'assemblage qui sont de "bas niveau") doivent être **traduits** pour donner naissance à leurs correspondants en langage machine.

#### *6.5.8.1 L'avantage des langages évolués*

Outre le fait que les langages de haut niveau offrent des possibilités d'expression qui sont plus proches des manières dont l'être humain envisage les traitements d'informations, il faut signaler qu'ils permettent au programmeur de s'affranchir complètement du type de machine (de processeur) qui finira par effectuer les traitements commandés.

On écrit un programme en PASCAL ou en BASIC (ou dans tout autre langage évolué) de la même manière quel que soit l'ordinateur qui exécutera les indications de traitement fournies. Le programme source (écrit dans le langage évolué) est donc indépendant des caractéristiques particulières du processeur que la version traduite du programme commandera. On peut représenter ceci par :

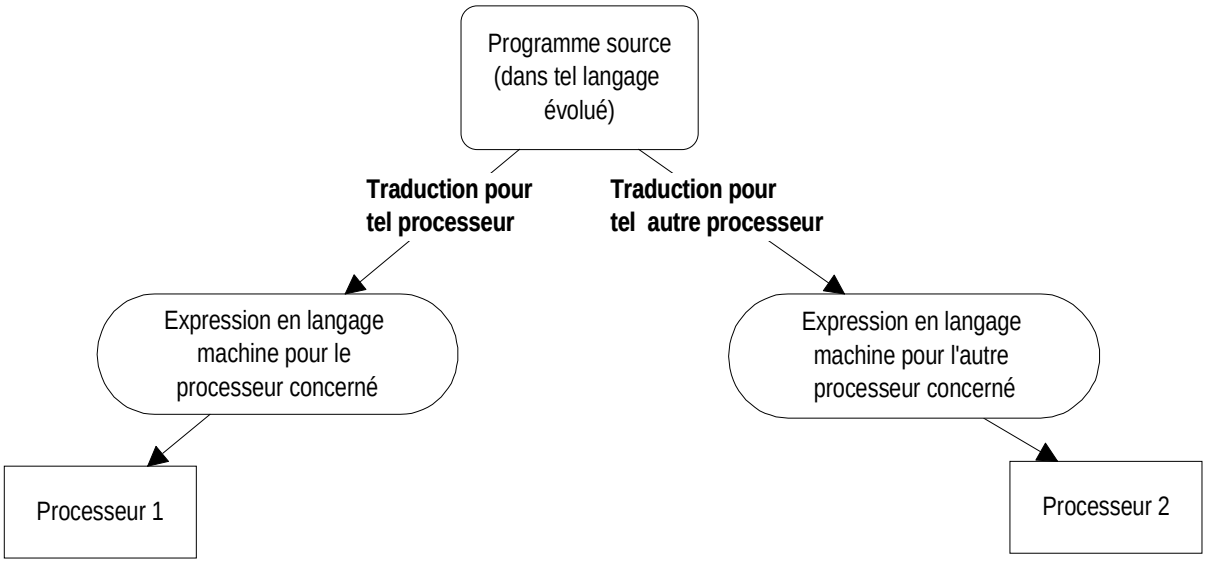



*Figure 6-10 : traduction d'un programme source unique*

Bien évidemment, le processus de traduction (et donc le programme de traduction correspondant) qui doit générer des instructions dans le langage machine de tel processeur particulier est, lui, adapté à ce processeur. Mais de ce processus et des caractéristiques du processeur, le programmeur n'a pas à se soucier.

#### *6.5.8.2 Une métaphore pour le processus de traduction*

Imaginons un instant que je dispose de la recette de la blanquette de veau, rédigée en français, mais que, par malchance, le cuisinier à qui je veux la confier, pour la préparation du plat correspondant, soit espagnol. J'ai le choix de deux stratégies pour arriver à faire préparer le plat par ce cuisinier espagnol, sur base de la recette exprimée en français : je peux, d'une part, faire appel à un traducteur ou, d'autre part, faire appel à un interprète.

##### 6.5.8.2.1 Caractéristiques du travail avec un traducteur

Sur base du texte en français de la recette que je lui remettrai, le traducteur va produire puis me rendre le texte en espagnol de la recette. C'est ce texte en espagnol que je vais donner au cuisinier espagnol pour qu'il exécute ce qui y est expliqué. Je garderai évidemment

précieusement et le texte de la recette en français et sa traduction en espagnol. Le texte en français me servira par exemple, tant que je ne serai pas content du plat confectionné. Si par exemple, le plat est trop salé, il me faudra corriger la recette (en français, celle en espagnol je ne la comprends pas), aller retrouver le traducteur, lui fournir la recette modifiée en français, reprendre la recette modifiée en espagnol, la rendre au cuisinier... et attendre le nouveau résultat. Évidemment, après quelques allers et retours chez le traducteur et lorsque le résultat sera satisfaisant, je garderai par prudence la recette en français (ne serait-ce que pour le cas où je tomberais un jour sur un cuisinier d'une autre nationalité), mais c'est celle en espagnol (la définitive) qui me servira à chaque fois que je voudrai faire confectionner le plat par le cuisinier espagnol. Bien entendu, si demain le cuisinier devient polonais, c'est d'un autre traducteur que j'aurai besoin, mais là je pourrai fournir directement la bonne recette en français et obtenir directement la bonne recette en polonais.

6.5.8.2.2 Caractéristiques du travail avec un interprète

Entre moi, disposant de la recette en français, et le cuisinier espagnol, j'amènerai un interprète français-espagnol. Instruction après instruction je lirai la recette en français, l'interprète les répétera en espagnol et à chaque fois le cuisinier exécutera le mini-travail correspondant. Ici, la recette sera traduite par petit bouts qui seront immédiatement exécutés. Traduction et exécution sont mélangées. Je ne disposera à aucun moment de la recette en espagnol. Même la centième fois que je ferai préparer la blanquette de veau, il me faudra amener l'interprète espagnol à côté du cuisinier espagnol et recommencer tout le processus. Sans la présence active de l'interprète je ne peux rien faire.

Le seul point positif, c'est que la mise au point de la recette sera plus aisé. Psychologiquement, ça me paraîtra moins fastidieux de recommencer immédiatement le processus traduction-exécution que de faire le détour par le travail d'un traducteur, lors des diverses mises au point qui pourraient être nécessaires.

6.5.8.2.3 La comparaison traduction-interprétation

| **Avec le traducteur** | **Avec l'interprète** |
|---|---|
| • Je disposerai après traduction de la recette en français et de son équivalent en espagnol | • Je n'aurai jamais la recette en espagnol, seulement la recette en français |
| • La traduction doit être faite avant l'exécution; elle en est bien distincte | • Traduction (interprétation) et exécution sont intimement mêlées |
| • Le traducteur ne doit pas être présent au moment de l'exécution | • L'interprète doit être présent à chaque exécution de la recette (qui existe seulement en français) |
| • L'exécution de la recette (en espagnol) est rapide : on est débarrassé du problème de traduction | • L'exécution-traduction est plus lente : chaque instruction doit être traduite; pire, si par hasard une instruction est répétée plusieurs fois dans le texte en français, elle sera à chaque fois traduite en espagnol avant d'être exécutée |
| • La mise au point de la recette est plus fastidieuse puisque chaque fois qu'un nouvel essai doit être fait, il faut faire retraduire l'ensemble de la recette; les allers et retours chez le traducteur, suivi de | • La mise au point paraît moins lourde; en réalité l'équivalent du temps mis par le traducteur est ici découpé en petits intervalles de temps passés à l'interprétation de chaque instruction de la |

| | |
|---|---|
| l'exécution-test sont laborieux | recette |

L'idéal consiste à employer un interprète pendant toute la phase de mise au point où la recette est testée. Lorsque la recette est correcte, on la fait traduire et, à partir de ce moment, c'est avec la recette en espagnol qu'on va à chaque fois trouver le cuisinier.

*6.5.8.3 Compilateur et interpréteur*

Le rôle du traducteur est ici endossé par un programme de traduction, un programme **compilateur**; celui de l'interprète par un programme **interpréteur**.

La recette, c'est ici le programme en langage évolué; le cuisinier espagnol, c'est le processeur; le traducteur, c'est un programme compilateur; l'interprète, c'est le programme interpréteur.

Un compilateur donc est un programme de traduction qui sur base d'un programme écrit dans tel langage évolué va générer l'équivalent en langage machine de tel processeur. Le programme compilateur est évidemment écrit dans le langage du processeur, langage vers lequel il traduit par ailleurs les programmes écrits dans le langage évolué considéré.

A la fin de la compilation, c'est à dire de la traduction automatique du texte en langage évolué vers le langage machine, on dispose donc du programme en langage machine. On appelle généralement **programme source** le texte en langage évolué et **programme objet** le programme en langage machine, résultat de la traduction effectuée par le compilateur.

L'interpréteur est également un programme de traduction, mais le travail se déroule alors comme suit :

- sous le contrôle du programme interpréteur, le processeur traduit une instruction du programme source en la série d'instructions correspondantes en langage machine;
- la main est passée à ce petit bout de programme, résultat de la traduction : les diverses instructions sont exécutées;
- la main est repassée au programme interpréteur; ce dernier fait effectuer la traduction de l'instruction suivante du programme source;
- etc..

Les petits bouts de programme en langage machine résultant du travail de l'interpréteur n'existent que le temps d'être exécutés. Jamais on ne disposera de l'ensemble du programme en langage machine.

On peut comparer les deux manières de procéder :

| **Le compilateur** | **L'interpréteur** |
|---|---|
| Dans le cas de l'expression d'un traitement dans un langage évolué, une traduction vers le langage machine est indispensable. C'est l'ordinateur lui-même qui s'en charge, gouverné par un programme compilateur ou par un programme interpréteur. | |
| Compilateur et interpréteur sont donc deux programmes faisant effectuer la traduction par l'ordinateur. Ils permettent au programmeur de s'affranchir complètement des caractéristiques de l'ordinateur sur lequel le programme finira par tourner. | |
| • Avec le compilateur, on dispose à côté du programme source (texte en langage évolué) d'un programme objet (en langage machine). | • Avec l'interpréteur, seul le programme source existe; il n'y a pas génération globale d'un programme en langage machine. |

| | |
|---|---|
| • La phase de compilation du programme doit précéder l'exécution : c'est le programme source qui est compilé (traduit), c'est le programme objet résultant qui est exécuté. | • Traduction (interprétation) et exécution sont intimement mêlées : une instruction est traduite puis les instructions en langage machine générées par cette traduction sont exécutées, puis on passe à l'instruction suivante... |
| • Le compilateur est inutile au moment de l'exécution; une fois le programme objet (exécutable) obtenu, c'est ce programme objet qui sera utilisé à chaque exécution. | • L'interpréteur est indispensable chaque fois qu'on veut faire exécuter le programme source. |
| • L'exécution de la version compilée est rapide. | • L'exécution-traduction est plus lente : chaque instruction doit être traduite; pire, si on passe plusieurs fois par une instruction lors de l'exécution, elle sera à chaque fois traduite par l'interpréteur avant que les instructions correspondantes en langage machine soient exécutées. |
| • La mise au point du programme est plus fastidieuse puisque chaque fois qu'une modification est apportée au texte, il faut faire retraduire l'ensemble du programme. Lors de la compilation, les erreurs de syntaxe présentes dans le texte du programme sont signalées et doivent être corrigées; c'est seulement lorsque la compilation va à son terme qu'on dispose du programme objet. Si ce dernier comporte des erreurs, il faut modifier le programme source et reprendre le processus. | • La mise au point paraît moins lourde avec un interpréteur ; en réalité l'équivalent du temps mis par le compilateur est ici découpé en petits intervalles de temps passés à l'interpréteur pour la traduction de chaque instruction du programme en langage évolué. |
| • Le programme compilateur est en général plus important et plus complexe à concevoir qu'un programme interpréteur. | • Le programme interpréteur est moins important et moins complexe à concevoir qu'un programme compilateur. |

### *6.5.9 Quelques langages évolués impératifs*

Les langages évolués que nous venons d'utiliser : BASIC d'une part, comme représentant des langages impératifs "à go to", PASCAL d'autre part, comme prototype des langages structurés, ne sont que deux des nombreux langages évolués existant.

> Comme déjà signalé, c'est une version ancienne du langage Basic qui a été utilisée ici. De plus, on n'a retenu que les traits des deux langages qui permettent de bien mettre en évidence et d'opposer l'approche "à go to" et l'approche structurée. Ainsi, PASCAL comporte une instruction "go to" et BASIC, même dans l'ancienne version retenue ici, possédait la structure d'appel de procédure.

#### *6.5.9.1 Les ancêtres : FORTRAN*

Le premier langage de haut niveau apparu dans l'histoire de la programmation des ordinateurs et qui soit toujours disponible (bien entendu sous une forme bien plus évoluée que

l'original) fut FORTRAN (Mathematical FORmula TRANslating System). Il fut développé par une équipe de programmeurs de IBM à partir de 1955 et disponible dès avril 1957. La version initiale eut bien des descendants : FORTRAN III en 1958, FORTRAN IV en 1962, FORTRAN 77, FORTRAN 90 et, le dernier né FORTRAN 95.

Les premières versions de FORTRAN mettaient en avant l'organisation à l'aide du GO TO; les dernières versions implémentent l'organisation des langages structurés (alternative, TANT QUE,...).

FORTRAN a été le langage de la programmation scientifique pendant près de 50 ans. De nombreuses librairies de programmes ont été développées en FORTRAN et sont toujours utilisées.

#### *6.5.9.2 Les ancêtres : COBOL*

Avec l'adoption des ordinateurs par le monde des affaires et des entreprises est apparu le besoin d'un langage évolué adapté aux problèmes de gestion. Ce fut le COBOL (COmmon Business-Oriented Language), né en 1960. Il continue à être utilisé aujourd'hui.

#### *6.5.9.3 Le prototype du langage structuré : PASCAL*

Apparu en 1970, il est l'œuvre d'un informaticien Suisse, Niklaus Wirth. Ses qualités "didactiques" l'ont fait adopter comme le langage utilisé pour l'apprentissage de la programmation. Il a été implémenté sur certains des premiers micro-ordinateurs, ce qui a contribué à répandre son usage.

#### *6.5.9.4 Le langage popularisé par la micro-informatique : BASIC*

Acronyme de "Beginner's All Purpose Symbolic Instruction Code" ("code d'instruction symbolique tous usages pour débutants"), il est né en 1965. Ce sont des dialectes de ce langage qui équipaient la plupart des premiers micro-ordinateurs des années 70 : l'interpréteur BASIC était souvent disponible dans la ROM de ces micro-ordinateurs qui se trouvaient donc, de manière native, capables d'exécuter les programmes écrits en BASIC. Le BASIC du début était relativement étriqué et mettait en avant les structures organisatrices des langages "à go to"; il a ensuite évolué pour intégrer les modes d'organisation des langages évolués.

#### *6.5.9.5 Et tous les autres*

Il y a eu des centaines d'autres langages de haut niveau impératifs: PL1, ALGOL, C, ADA, pour ne citer que les plus connus. Les derniers arrivés intègrent des approches "orientées objets", comme C++ ou Java.

### *6.5.10 Les macros développées au sein des logiciels (à compléter)*

### *6.5.11 L'algorithmique*

Les langages de haut niveau permettent de s'affranchir complètement des contraintes de tel ou tel ordinateur (à la différence des langages machines ou des langages d'assemblage). Ils ont permis de mettre l'accent non sur le contrôle de tel ou tel ordinateur mais sur des classes de problèmes, traitables par tout ordinateur, à condition qu'on veuille bien écrire le programme correspondant.

Une science est née de ce souhait de faire effectuer des tâches par l'ordinateur : l'algorithmique. On pourrait la désigner comme l'art et la discipline consistant à créer des "marches à suivre" (programmes) destinées à faire effectuer certaines tâches de traitement d'information par l'exécutant ordinateur présenté ci-dessus.

Il y a bien des catégories de tâches qui sont redevables de l'algorithmique. Citons par exemple, pour nous restreindre à des classiques :

- toutes les tâches de tris, comme par exemple trier une liste de mots par ordre alphabétique ou trier une série de nombres;
- les problèmes de recherche dans une liste : il s'agit de déterminer, par exemple, si un mot est ou non présent dans une longue liste de mots, liste triée ou non;
- dans le même ordre d'idée, on peut avoir à analyser la tâche consistant à déterminer si une chaîne de caractères fait partie d'une autre chaîne de caractères; ces algorithmes permettent par exemple de faire vérifier par l'ordinateur si une chaîne "TGCCGTACCTGGTTGCATTGGC" se retrouve ou non au sein d'une chaîne de plusieurs dizaines ou centaines de milliers de combinaisons des lettres T, G, A et C; le problème de la recherche d'un mot au sein d'un texte est évidemment tout à fait semblable;
- tous les problèmes posés au sein de ce qu'on appelle la "théorie des graphes"; un graphe est, par exemple, une combinaison de sommets et d'arcs joignant ces sommets :

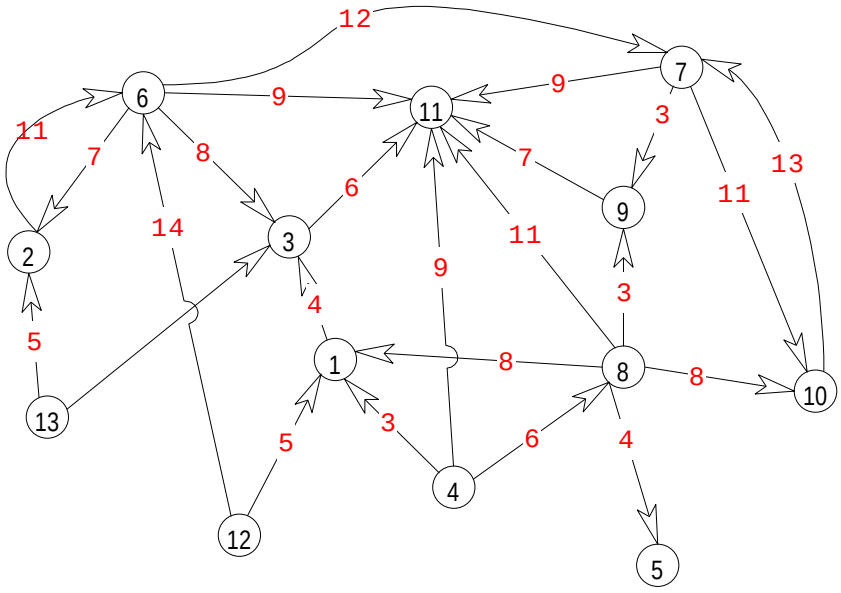



*Figure 6-11 : un graphe*

Dans le graphe présenté ici, les arcs ont tous, par exemple, un certain "poids". On peut poser les problèmes suivants :

- trouver tous les couples de sommets qui sont joints par un chemin (succession d'arcs);
- trouver entre deux sommets un chemin (succession d'arcs) de poids minimal (le poids d'un chemin étant la somme des poids des arcs le composant);
- trouver, s'il existe, un chemin parcourant l'ensemble des sommets en ne passant qu'une seule fois par chacun;
- etc.

On peut faire état de milliers de situations problèmes trouvant leur origine dans la théorie des graphes et ayant donné lieu à de nombreuses recherches d'algorithmes.

## 6.6 Les langages évolués de type logique

Les langages présentés jusqu'ici servent à décrire ce qui sera attendu de l'ordinateur sous la forme d'une "marche à suivre". Les programmes exprimés dans ces langages sont constitués d'ordres : ils s'écrivent en quelque sorte "à l'impératif". On a donné à ce style de langage de programmation, donc au type de programmes résultants, le nom de **programmation impérative** ou encore de **programmation procédurale** : on y exprime les indications de traitement sous la forme d'une succession d'ordres.

Ces langages sont les plus répandus. Mais il existe, en réponse à des problèmes particuliers, d'autres styles de programmation, où les programmes prennent d'autre formes qu'une succession d'ordres donnés à un exécutant : l'approche **logique** en est un exemple.

### *6.6.1 Des faits et des règles*

Un "programme" dans l'approche logique n'a plus rien à voir avec ce que nous avons appelé "programme" jusqu'ici. Un programme, ce sera, dans un domaine bien circonscrit, l'énoncé de faits et de règles (de déduction); l'exécution du programme aura toujours pour but de tenter de répondre à une question en s'appuyant sur les faits et les règles énoncés.

#### *6.6.1.1 Un exemple en généalogie*

Imaginons que nous ayons à rendre compte des liens de parentés représentés entre quelques personnes et résumés dans le diagramme suivant :

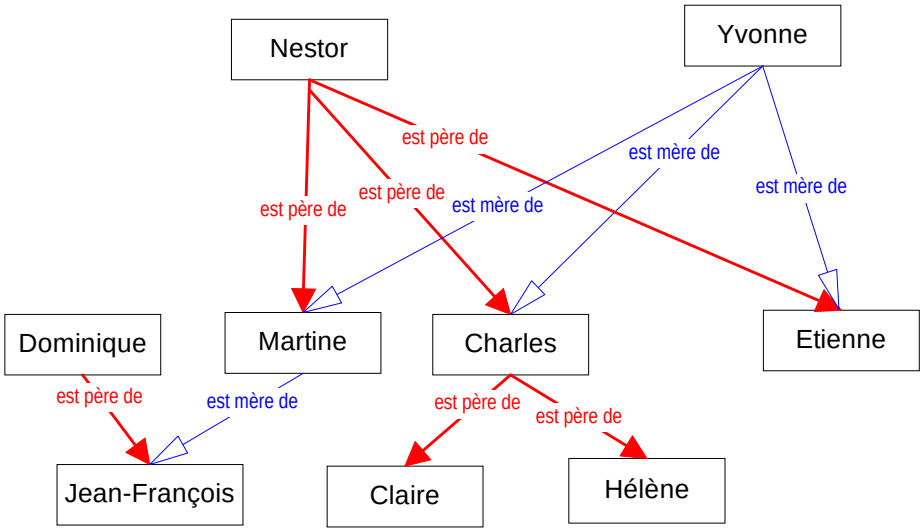



*Figure 6-12 : généalogie*

Voici exprimé en PROLOG, un langage de programmation de type logique, les quelques **faits** montrés par le diagramme :

```
clauses

    de_sexe_masculin(charles).
    de_sexe_masculin(nestor).
    de_sexe_masculin(etienne).
    de_sexe_masculin(dominique).
    de_sexe_masculin(jean_francois).

    de_sexe_feminin(yvonne).
    de_sexe_feminin(martine).
```

```
de_sexe_feminin(claire).
de_sexe_feminin(helene).

mere(martine, yvonne).                    /* yvonne est mère de martine */
mere(charles, yvonne).
mere(etienne, yvonne).
mere(jean_francois, martine).

pere(charles, nestor).                    /* nestor est père de charles */
pere(claire,charles).
pere(helene,charles).
pere(etienne,nestor).
pere(jean_francois,dominique).
pere(martine,nestor).
```

La syntaxe utilisée se comprend d'elle même; on notera les commentaires délimités par /* */. Il reste à donner quelques règles : ce sont essentiellement celles qui définissent des vocables nouveaux comme parent, frère, sœur,...

```
parent(X, Y) if mere(X, Y).          /* Y est parent de X dès que Y est mère de X */
parent(X, Y) if pere(X, Y).          /* Y est parent de X dès que Y est père de X */

frere(X, Y) if                       /* Y est frère de X, dès que */
      de_sexe_masculin(Y) and        /* Y est de sexe masculin et */
      pere(X, P) and                 /* P est père de X et */
      pere(Y, P) and                 /* P est père de Y et */
      mere(X, M) and                 /* M est mère de X et */
      mere(Y, M)                     /* M est mère de Y et */
      X <> Y.                        /* X et Y sont différents */

soeur(X, Y) if                       /* Y est sœur de X, dès que */
      de_sexe_feminin(Y) and         /* Y est de sexe féminin et */
      pere(X, P) and                 /* P est père de X et */
      pere(Y, P) and                 /* P est père de Y et */
      mere(X, M) and                 /* M est mère de X et */
      mere(Y, M)                     /* M est mère de Y et */
      X <> Y.                        /* X et Y sont différents */

oncle(X, U) if                       /* U est oncle de X, dès que */
      parent(X, P) and               /* P est parent de X  et */
      frere(P, U).                   /* U est frère de P */

grand_pere(X, G) if                  /* G est grand-père de X, dès que */
      pere(P,G) and                  /* G est le père de P */
      parent(X,P).                   /* et P est parent de X */

grand_mere(X, G) if                  /* G est grand-mère de X, dès que */
      mere(P,G) and                  /* G est la mère de P */
      parent(X,P).                   /* et P est parent de X */
```

Techniquement, ce que nous avons déjà écrit doit être précédé de l'annonce des divers prédicats définis :

```
domains
        personne = symbol

predicates
        de_sexe_masculin(personne)
        de_sexe_feminin(personne)
        pere(personne,personne)
        mere(personne,personne)
        soeur(personne,personne)
        parent(personne,personne)
        frere(personne,personne)
        oncle(personne,personne)
        grand_pere(personne,personne)
        grand_mere(personne, personne)
```

Voilà à quoi ressemble un "programme" exprimé dans un tel langage de type logique (ici, il s'agit d'une version déjà ancienne de PROLOG).

L'exécution d'un tel programme prend la forme de questions que l'utilisateur peut poser au programme. En voici quelques unes (il est important d'avoir le schéma généalogique sous les yeux) :

| En français courant | Question au programme | Réponse du programme |
|---|---|---|
| Quel est le grand-père de Claire ? | Goal : grand_pere(claire, X) | X=nestor<br>1 Solution |
| Quels sont les oncles de Jean-François ? | Goal : oncle(jean_francois,X) | X=charles<br>X=etienne<br>2 Solutions |
| Dominique est-il l'oncle d'Hélène ? | Goal : oncle(helene, dominique) | False |
| Quels sont les parents de Charles ? | Goal : parent(charles, P) | P=yvonne<br>P=nestor<br>2 Solutions |
| Qui est sœur de qui ? | Goal : soeur(X,Y) | X=charles, Y=martine<br>X=etienne, Y=martine<br>2 Solutions |
| De qui Yvonne est-elle grand-mère ? | Goal : grand_mere(X, yvonne) | X=jean_francois<br>X=claire<br>X=helene<br>3 Solutions |
| De qui Dominique est-il le frère ? | Goal : frere(X, dominique) | No Solution |

Évidemment, comme à l'habitude, les questions doivent être formulées avec les termes exacts utilisés dans le programme; des erreurs à ce niveau sont payées comptant :

| | | |
|---|---|---|
| De qui Martine est-elle grand-mère ? | Goal : grandmere(X,martine)<br>(on a écrit grandmere et non grand_mere, comme il aurait fallu | 404 Undeclared predicate<br>(Le prédicat grandmere est inconnu) |

| | | |
|---|---|---|
| | fallu. | |
| De qui Dominique est-il le frère ? | Goal : frere(X, Dominique)<br>(on notera la majuscule à Dominique) | X=etienne, Dominique=charles<br>X=martine, Dominique=charles<br>X=charles, Dominique=etienne<br>X=martine, Dominique=etienne<br>4 Solutions |

Comment s'explique cette dernière réaction?
A propos, comment définiriez-vous cousin et cousine? demi-frère? petit-fils ?

On aura deviné qu'on va pouvoir, à propos d'un domaine précis et bien délimité, décrire les faits et les règles relevants pour le domaine considéré. On peut d'ailleurs faire en sorte (mais ce n'est pas le propos ici) que le programme-lui-même pose une série de questions.

*6.6.1.2 Un exemple "zoologique"*

Voici à titre d'exemple l'exécution d'un programme (un rien plus compliqué que celui sur la généalogie) et qui énonce faits et règles dans le domaine de la zoologie. Voici d'abord, à titre anecdotique, quelques extraits de ce programme :

```
l_animal_est(un_tigre) if
        c_est_un(mammifere) and
        c_est_un(carnivore) and
        positive(a,une_couleur_fauve) and
        positive(a,des_raies_sombres).
  .....
```

```
c_est_un(mammifere) if
        positive(a,des_poils).
c_est_un(mammifere) if
        positive(peut,donner_du_lait).
c_est_un(un_oiseau) if
        positive(a,des_ailes).
c_est_un(un_oiseau) if
        positive(peut,voler) and
        positive(peut,pondre_des_oeufs).
  .....
```

et quelques exemples d'exécution (en gras, les interventions de l'utilisateur) :

| | |
|---|---|
| Goal : **run**<br>l'animal a des_poils ? **non**<br>l'animal peut donner_du_lait ? **non**<br>l'animal a des_ailes ? **oui**<br>l'animal peut voler ? **oui**<br>l'animal peut nager ? **non**<br>l'animal peut pondre_des_oeufs ? **oui**<br>l'animal peut bien_voler ? **oui**<br><br>*Votre animal pourrait être un_albatros* | Goal : **run**<br>l'animal a des_poils ? **oui**<br>l'animal peut manger_de_la_viande ? **oui**<br>l'animal a une_couleur_fauve ? **oui**<br>l'animal a des_taches_sombres ? **non**<br>l'animal a des_dents_pointues ? **oui**<br>l'animal a des_serres ? **non**<br>l'animal peut donner_du_lait ? **oui**<br>l'animal a des_raies_sombres ? **oui**<br><br>*Votre animal pourrait être un_tigre* |
| Goal : **run**<br>l'animal a des_poils ? **non**<br>l'animal peut donner_du_lait ? **oui**<br>l'animal peut manger_de_la_viande ? **oui**<br>l'animal a une_couleur_fauve ? **non**<br>l'animal a des_dents_pointues? **non**<br>l'animal a des_sabots ? **non**<br>l'animal peut ruminer ? **non** | Goal : **run**<br>l'animal a des_poils ? **oui**<br>l'animal peut manger_de_la_viande ? **non**<br>l'animal a des_dents_pointues ? **non**<br>l'animal peut donner_du_lait ? **oui**<br>l'animal a des_sabots ? **oui**<br>l'animal a un_long_cou ? **oui**<br>l'animal a de_longues_pattes ? **oui** |

| | |
|---|---|
| l'animal a des_ailes ? **non**<br>l'animal peut voler ? **non**<br><br>*Je ne sais vraiment pas ce qu'est votre animal.* | l'animal a des_taches_sombres ? **oui**<br><br>*Votre animal pourrait être une_girafe* |

### *6.6.2 Programmation logique et systèmes experts*

La programmation logique est l'approche par excellence des systèmes experts. Un système expert, c'est, dans un domaine donné, un programme qui emmagasine autant que faire se peut les connaissances des experts humains du domaine considéré, sous forme de faits et de règles de déduction.

Il y a des centaines, sinon des milliers de systèmes experts dans des domaines aussi variés que la recherche pétrolière, la médecine, l'entretien des locomotives, la culture des tomates,...

#### *6.6.2.1 Un exemple de dialogue avec un système expert de maintenance de locomotives : RUFUS*

Les instructions et les questions de Rufus à l'utilisateur apparaissent en caractères italiques. Les réponses de l'utilisateur (oui, non, ou un chiffre représentant une mesure lue par le dépanneur sur un appareil ou son choix dans le menu des réponses proposé par Rufus), en caractères gras rouge.

*1) Quel signalement l'exploitant a-t-il donné ?*
*La liste des réponses possibles est*
*0) inconnu*
*1) motrice_inactive*
*2) insuffisance_de_traction*
*3) rhéostatique_hs*
*4) panne_de_porte*
*5) panne_de_fermeture_générale*
*6) panne_d_ouverture_générale*
*7) panne_locale_de_fermeture*
*8) panne_de_maintien_fermeture*
*9) panne_locale-ouverture*
*10) panne_annonce_départ*
**1**
*2) Quel est le nombre d'éléments du train ?*
*Vous devez répondre un nombre compris dans [1 3]*
**1**
*Mettre la clé de conduite sur CN (conduite normale)*
*Sélectionner un sens de marche.*
*Commander les disjoncteurs.*
*Mettre le manipulateur au neutre.*
*Mettre la clé TE sur HS.*
*3) Le voyant motrice inactive est-il allumé au BS (boîtier de signalisation) ?*
**oui**
*Faire un KIT sur la motrice de tête et regarder le qmètre.*
*4) Quelle est la position du qmètre ?*
*Vous pouvez répondre un nombre quelconque.*
**1**

*La motrice en panne est la motrice de tête.*
*Se mettre en ESSAI A BLANC.*
*Mettre la clé test/épreuve sur épreuve.*
*Mettre le manipulateur en T5.*
*5) Le voyant motrice inactive reste-t-il allumé au BS ?*
**non**
*Mettre la clé test/épreuve sur HS.*
*Mettre le manipulateur au neutre.*
*Commander les DET.*
*Sélectionner le sens de marche AVANT*
*6) Les disjoncteurs sont-ils montés ?*
**non**
*7) Sélectionner une marche arrière.*
*Les disjoncteurs montent-ils ?*
**non**
*8) Commander un essai à blanc.*
*Bien écouter le JH.*
*Le JH a-t-il bougé ?*
**oui**
*9) Avez-vous entendu le JH faire tout son cycle 0 -> -1, -1 -> 0 ?*
*(JH = nom de l'arbre à cames, du nom de leurs inventeurs).*
**oui**
*10) Commander les DET (disjoncteurs).*
*Bien écouter le JH.*
*Le JH bouge-t-il ?*
**oui**
*11) Avez-vous entendu le JH faire tout son cycle 0 -> -1, -1 -> 0 ?*
**oui**
*Vérifier que les conditions initiales suivantes sont remplies*
* *manipulateur au neutre*
* *sens de marche sélectionné*
* commande de montée des DET
* *KIT en position normale*

*IL Y A UNE PANNE DE COMMANDE DE MONTEE DES DISJONCTEURS*
*12) Le fusible 88A est-il ok ?*
**oui**
*13) Le fusible 103E-108a est-il ok ?*
**oui**
*14) Le relais RCD (relais de commandes du diagnostic) est-il ok ?*
**oui**
*La consultation est terminée.*
*Il y a une panne de commande de montée des disjoncteurs.*
*Il faut contrôler le circuit 108B.*

#### *6.6.2.2 Un exemple dans le domaine du diagnostic médical : SPHINX*

Voici le dialogue entre le médecin utilisateur du système et le système expert :

**U** **Age 63 ans**
**U** **Glycémie 8 mol/1**
*M* *le malade est-il vu dans des circonstances particulières*
*M* *Coma, reanimation, pre-op, grossesse (pour une femme) ?*
**U** **Non, consultation**

*M* *Sous traitement hypoglycémiant*
**U** **Oui**
*M* *Insuline ?*
**U** **Non**
**U** **Glucophage 2 cp/j depuis 8 ans**
*M* *Quel est son poids (% poids idéal) ?*
**U** **95 %**
*M* *Existe-t-il un facteur de déséquilibre ?*
**U** **Oui, infection urinaire**
M Quel est le résultat de l'acetest ?
**U** **Négatif**
*M* *Le traitement est-il bien supporté cliniquement ?*
**U** **Oui**
*M* *Le malade présente-t-il une insuffisance hépatique sévère ?*
**U** **Non**
*M* *A-t-on dose la créatinine (micromol/1) ?*
**U** **Oui, 210**
*M* *CONCLUSIONS*
*M* *GLYCOREGULATION PERTURBEE*
*M* *DIABETE LEGEREMENT DESEQUILIBRE*
*M* *BIGUANIDES CONTRE-INDIQUES, SULFAMIDOTHERAPIE POSSIBLE*
*M* *THERAPEUTIQUE*
*M* *ARRETER LE BIGUANIDE*
*M* *PRESCRIRE 2 cp DE SULFAMIDE FAIBLE/J*
*M* *REGIME NORMOGLUCIDIQUE*
*M* *TRAITEMENT DE LA CAUSE DU DESEQUILIBRE*

## 6.7 Le langages évolués de type fonctionnel (à compléter)

# 7

# ... pour autant qu'on lui ait indiqué comment mener à bien ce traitement ...

# Le logiciel

## 7.1 L'architecture logicielle

### *7.1.1 La dualité matériel-logiciel*

Nous savons qu'un ordinateur "nu" (= sans logiciel pour le commander) n'existe pas :on peut relire à cet égard le début du chapitre 4 (page 53) et se souvenir du principe fondamental :

| **TOUT** ce que fait un ordinateur, il le fait gouverné par un programme |
|---|

C'est dire que la facette logicielle d'un système informatique est essentielle et mérite également d'être décrite avec soin.

### *7.1.2 Le logiciel*

Nous désignerons par ce terme général "**le** logiciel" cette seconde réalité du monde de l'informatique et des technologies de l'information et de la communication; le monde de l'informatique est en effet comme une médaille aux deux faces indissociables : le matériel, d'une part, le logiciel, d'autre part.

Cette réalité du logiciel n'est évidemment pas figée : tout comme les performances du matériel, le logiciel a évolué tant dans l'étendue de ce qu'il permet de réaliser que dans les facilités - on parle de convivialité - qu'il offre aux utilisateurs.

### *7.1.3 L'architecture logicielle*

Le chapitre précédent montrait comment l'existence et l'évolution des langages de programmation a fourni les outils permettant aux programmeurs de créer des programmes de plus en plus sophistiqués et complexes.

C'est un tout autre point de vue qui sera mis en avant ici, celui du "consommateur" de logiciels : l'utilisateur. Mais exactement comme nous avons parlé d'architecture à propos du matériel, nous présenterons en quelque sorte l'**architecture logicielle** d'un système informatique : quels sont les logiciels présents dans l'unité centrale et quels sont leurs rôles respectifs pendant une session de travail "à l'ordinateur".

Ainsi, par exemple, lors de l'utilisation du logiciel de traitement de texte Word "sous" Windows sur un PC, on peut imaginer l'intérieur de la mémoire centrale comme constitué d'une série de "couches logicielles" s'appuyant les unes sur les autres :

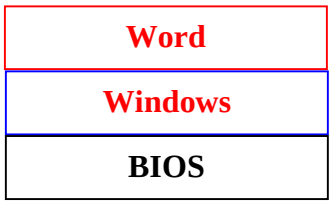



*Figure 7-1 : les "couches logicielles"*

- Chacune de ces couches, elles-mêmes constituées d'ensembles de programmes, prend en charge certaines fonctions. Ainsi des programmes de la couche du BIOS, la plus profonde, se chargent d'opérations extrêmement élémentaires relatives à l'affichage ou à la lecture de caractères au clavier; des programmes de la couche Windows (celle, on va le voir, constituant le système d'exploitation) gèrent par exemple lectures et écritures des fichiers et dossiers sur les mémoires de masse; la couche "Word" (celle de l'application vraiment utilisée) prend, entre autres, en charge édition et mise en forme du texte frappé par l'utilisateur.
- Chaque couche s'appuie en quelque sorte sur les couches inférieures en sous traitant à ces dernières certaines opérations. Ainsi, lorsqu'on commande au logiciel de traitement de texte d'enregistrer sur le disque dur, au sein d'un dossier choisi, le texte confectionné, ce logiciel "passe la main" à des programmes de la couche Windows qui vont se charger du travail souhaité.

  Il faut donc s'imaginer l'intérieur de la mémoire centrale comme comportant plusieurs dizaines de programmes qui vont s'appeler les uns les autres, se passant la main pour que chacun puisse (faire) réaliser les opérations qui sont de son ressort. De plus, des programmes supplémentaires sont fréquemment amenés à partir des mémoires de masse, pour venir participer au travail global, remplaçant en mémoire centrale d'autres programmes devenus inutiles.

### *7.1.4 Les deux strates essentielles du logiciel*

On répartit en général le logiciel en deux entités : le **logiciel de base**, d'une part, le **logiciel d'application**, d'autre part.

Nous présenterons plus en détail, plus loin, le logiciel d'application; comme son nom l'indique, il recouvre des milliers d'applications différentes, du traitement de texte au courrier électronique en passant par la retouche d'image et la manipulation de séquences vidéos. C'est évidemment certains de ces logiciels d'application qui intéressent au premier chef les utilisateurs.

Quant au logiciel de base, sa dénomination fait penser qu'il constitue "la base" ou le socle des diverses couches logicielles indispensables au travail de l'ordinateur.

- Le composant primordial du logiciel de base est le **système d'exploitation**, qui fait l'essentiel du présent chapitre.
- Dans le contexte actuel, il faut y ajouter toute la partie du logiciel chargée de gérer les **communications** avec d'autres ordinateurs, à travers les réseaux (réseau local ou liaison par modem et réseau téléphonique,...); de plus en plus souvent ces logiciels de communication sont intégrés dans le système d'exploitation proprement dit.
- Habituellement, on range aussi dans le logiciel de base des logiciels particuliers comme les **éditeurs** de texte, les divers **compilateurs** et **interpréteurs** disponibles

(voir page 159). On aura compris que ces logiciels sont essentiellement utilisés dans le contexte de la programmation et qu'ils ne font donc pas partie des outils habituellement employés par les utilisateurs "naïfs".

## 7.2 Le système d'exploitation

Voici la pièce maîtresse du logiciel : on l'appelle "Operating System" en Anglais et même pendant un temps, aujourd'hui révolu, "Disk Operating System" ("DOS") pour insister sur son rôle dans la gestion des disques durs et autres mémoires de masse, comme sur le fait que, résident sur le disque, il est amené en mémoire centrale pour y être actif.

C'est un **ensemble de programmes** sans lesquels l'ordinateur ne pourrait rien faire et, surtout, sans lesquels les logiciels d'application ne pourraient "tourner". Il est malaisé d'expliquer ou de définir ce qu'est un système d'exploitation; voici à titre d'exemples quelques définitions glanées ça et là sur le WEB

- "*Ensemble des fonctions de base (mais parfois pouvant être très avancées), permettant l'usage d'un ordinateur, et sans lequel rien n'est possible."*
- *"Ensemble de logiciels assurant le fonctionnement de base de l'ordinateur et notamment la gestion du processeur et de sa relation avec les différents périphériques. Un ordinateur ne peut fonctionner sans lui."*
- *"Le système d'exploitation a deux fonctions principales: fournir un niveau d'abstraction pour communiquer avec le hardware et gérer les ressources."*
- *"Un système d'exploitation (OS:Operating System) est un ensemble de programmes de gestion du système qui permet de gérer les quatre éléments fondamentaux de l'ordinateur: le matériel, les logiciels, la mémoire, les données."*
- *"Il s'agit du principal logiciel de contrôle système, qui sert de fondation à l'ensemble des logiciels applicatifs."*
- *"Le système d'exploitation constitue le programme principal de contrôle qui fait fonctionner l'ordinateur."*
- *"C'est un programme informatique chargé de gérer les ressources d'un ordinateur comme le temps processeur, la mémoire, les périphériques ou les programmes en cours d'exécution (processus)."*
- *"Un système d'exploitation (Operating System en Anglais, ou OS) est le logiciel de base permettant de faire tourner des applications sur un PC. C'est lui qui pilote directement tout le matériel (disque dur, affichage, clavier, souris, imprimantes, etc...), et c'est également lui qui démarre (à la demande de l'utilisateur) les diverses applications que l'on peut trouver sur un PC."*
- *"Logiciel qui gère l´ordinateur."*
- *"Logiciel informatique responsable du contrôle de l'allocation et de l'utilisation des ressources matérielles et de l'exécution des programmes d'application."*
- *"Logiciel qui contrôle l'affectation et l'utilisation de ressources matérielles telles que la mémoire, le temps processeur, l'espace disque et les périphériques. Un système d'exploitation est la base sur laquelle s'exécutent les logiciels (applications)."*

La plupart de ces définitions contiennent des éléments importants ou significatifs. Je retiendrai ici les facettes suivantes :

- c'est un ensemble de programmes;
- ces programmes assurent les fonctions de base de l'ordinateur (mais qu'est ce qu'une "fonction de base" ?);
- ces programmes ont en charge la gestion des périphériques, des mémoires de masse, de l'affectation de la mémoire centrale aux divers programmes et données;
- sans les programmes constituant le système d'exploitation, l'ordinateur ne peut pas marcher et les logiciels d'application ne peuvent pas s'exécuter

### *7.2.1 Les rôles du système d'exploitation*

C'est en repartant de ce que perçoit l'utilisateur d'un système (= matériel + logiciel) informatique, qu'on peut dégager les deux types de rôles d'un système d'exploitation : les rôles **visibles** et les rôles **cachés**.

#### *7.2.1.1 Les rôles **visibles** du système d'exploitation*

Le système d'exploitation (ou à tout le moins certains des programmes qui le composent) peut à certains moments du travail de l'utilisateur constituer l'interlocuteur, le programme à qui l'utilisateur va fournir des **commandes**.

> Une commande, c'est une intervention de l'utilisateur dans le déroulement d'un logiciel; lorsque l'utilisateur demande au logiciel de traitement de texte de mettre du texte préalablement sélectionné en gras, il donne au logiciel une commande; lorsqu'il souhaite enregistrer sur le disque dur, sous un nom qu'il choisit, le texte composé, il va exprimer ce souhait à l'aide d'une commande...
> En gros, l'utilisateur d'un logiciel n'arrête pas de donner à ce dernier des commandes.
>
> Enfin, on aura compris que ce que l'utilisateur commande, ce n'est pas le logiciel, mais (comme toujours) le couple ordinateur-logiciel; mais par abus de langage, on parlera de commandes données à un logiciel.

Ces commandes que l'utilisateur va fournir peuvent prendre des formes diverses : du texte frappé au clavier ou des combinaisons de touches, mais aussi des "clics" avec la souris. Cette couche du système d'exploitation qui prend en charge les échanges avec l'utilisateur est l'interface utilisateur et nous l'aborderons en détail plus loin.

En quelque sorte, dans ce rôle visible, le système d'exploitation joue un rôle un peu similaire aux logiciels d'applications : l'utilisateur souhaite effectuer certaines opérations à travers le système d'exploitation. Mais, si l'on voit assez aisément quelles actions vont être commandées à travers un traitement de texte ou un logiciel de dessin, il est a priori plus malaisé de cerner la portée des commandes fournies au système d'exploitation.

##### 7.2.1.1.1 Le premier rôle visible : la **gestion des mémoires de masse**

Ce qui est visé ici, c'est tout ce qui concerne la manière dont l'utilisateur va organiser des documents de natures diverses qu'il aura créés et modifiés grâce à l'un ou l'autre logiciel d'application et "sauvés" sur le disque dur, un CD, une disquette, etc..

Ces divers documents, qui constituent donc les données -images, textes, sons- traitables par certains logiciels prennent la forme, sur les mémoires de masse, d'entités qu'on nomme **fichiers**. A côté de ces fichiers de données, on trouvera aussi sur les mémoires de masse des fichiers contenant les programmes exécutables (par exemple les logiciels d'application) qui pourront être amenés en mémoire centrale pour commander le processeur.

Ces fichiers seront répartis dans des **dossiers** divers : c'est, entre autres, cette organisation en dossiers contenant d'autres dossiers et des fichiers que le système d'exploitation va permettre à l'utilisateur de réaliser.

Voici, à titre d'illustration, la représentation que donne Windows, l'un des systèmes d'exploitation les plus répandus, de ces dossiers et fichiers :

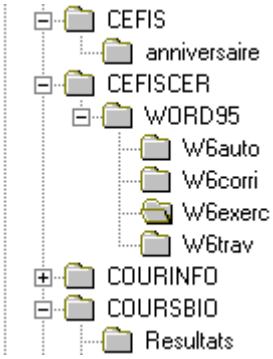



*Figure 7-2 : quelques dossiers et sous dossiers présents sur mon disque dur*

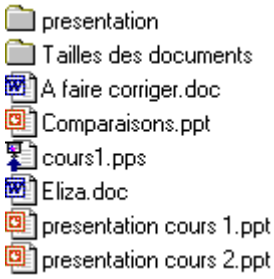



*Figure 7-3 : quelques dossiers et fichiers présents au sein d'un dossier*

Toujours à titre d'exemples, on peut citer quelques unes des opérations concernant dossiers et fichiers que permet le système d'exploitation :

- afficher une liste de dossiers ou de fichiers présents sur le disque dur ou un autre support de mémoire de masse;
- créer un dossier nouveau dans lequel on pourra classer ensuite fichiers et dossiers;
- copier ou déplacer un fichier ou un groupe de fichiers (ou tout le contenu d'un dossier) d'un dossier vers un autre;
- supprimer un fichier, un groupe de fichiers ou un dossier;
- rechercher sur base d'une partie de son nom ou de certaines de ses caractéristiques un fichier ou un groupe de fichiers;
- renommer un fichier ou un dossier;
- etc.

Cette liste n'est évidemment pas exhaustive...

#### 7.2.1.1.2 Le deuxième rôle visible : la **gestion des périphériques**

C'est le système d'exploitation qui gère les divers périphériques; pour chacun de ceux-ci, l'utilisateur peut, à travers des choix effectués en donnant des commandes au système d'exploitation, préciser un certain nombre de paramètres :

- Ainsi, en ce qui concerne le **clavier**, on peut préciser quel est le type du clavier (il y en a plusieurs dizaines, différant d'après l'emplacement des diverses touches); on peut aussi choisir le délai plus ou moins long à partir duquel l'appui continu sur une touche provoque la répétition du caractère choisi, et la vitesse à laquelle se fait cette répétition.

- On peut aussi préciser divers paramètres pour la **souris** : inversion des boutons gauche et droit, vitesse d'un double clic,...
- C'est sans doute **l'écran** qui amène le plus de choix possibles : définition (= combien de pixels affiche, en largeur et en hauteur, l'écran) : 1024 x 768, 800 x 600,...; quelle est la fréquence de rafraîchissement,...

  Il faut encore y ajouter toutes les options pour les diverses imprimantes, la carte son,...

7.2.1.1.3 Le troisième rôle visible : les choix au niveau de **l'interface-utilisateur**

De nombreux choix peuvent par exemple être fait au niveau des interfaces utilisateur de type graphique (voir plus loin) : couleur des affichages opérés par le système d'exploitation, taille et police des caractères utilisés, signaux sonores accompagnant certains évènements (erreur, clôture d'une tâche,...), mode d'affichage des contenus des dossiers,...

7.2.1.1.4 Le quatrième rôle visible : "**lancer" une application**

C'est à travers le système d'exploitation que l'utilisateur va pouvoir choisir puis activer les applications disponibles. "Lancer" (on dit aussi "ouvrir") une **application** , c'est demander qu'une copie du programme exécutable correspondant, résidant sur le disque dur, soit amenée en mémoire centrale et que le système d'exploitation lui "passe la main".

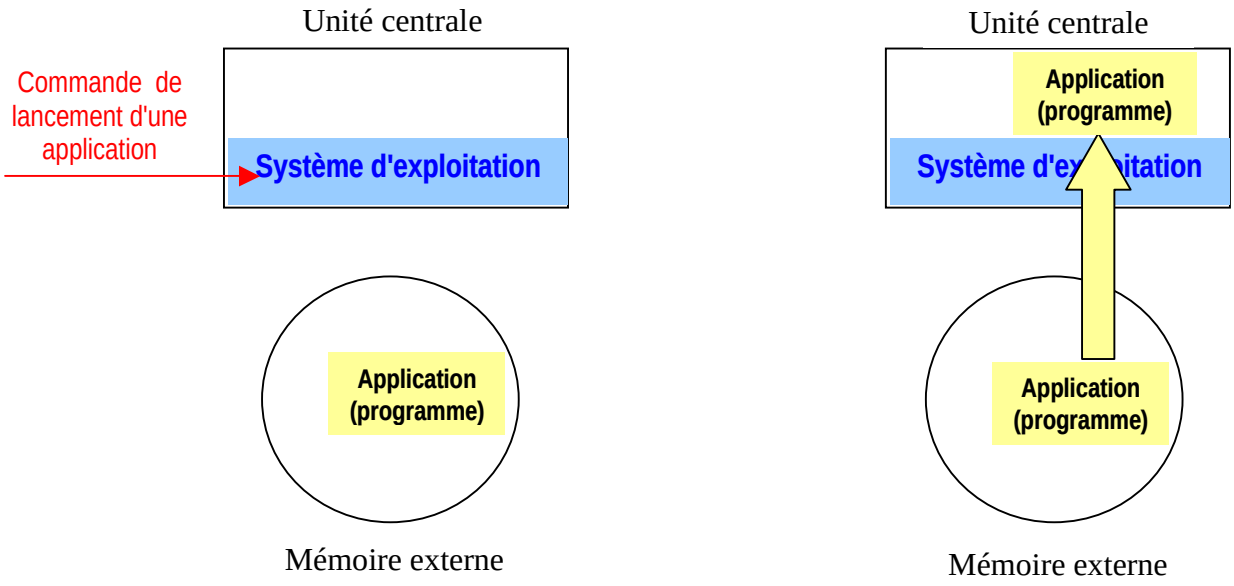



*Figure 7-4 : le "lancement" d'une application*

Une fois amenée en mémoire centrale pour être exécutée, une application donne lieu à ce qu'on appelle une **tâche**. Il y a donc en général des centaines d'applications disponibles sur les mémoires externes (disque dur), mais à tout moment quelques tâches seulement peuvent se partager l'unité centrale. L'une de ces tâches est évidemment le système d'exploitation.

*7.2.1.2 Les rôles **cachés** du système d'exploitation*

Si le système d'exploitation est de temps à autre l'interlocuteur direct de l'utilisateur, ce sont cependant ses rôles cachés qui sont sans doute les plus essentiels. Mais comme c'est alors avec une autre application qu'est en train de travailler l'utilisateur, ces rôles cachés restent en général non perçus par ce dernier.

Comme on va le voir, le système d'exploitation est à la fois au service des applications dont il sous-traite une partie du travail (demandé par l'utilisateur) mais aussi un genre de "chef d'orchestre" qui organise le travail des diverses tâches en cours, en leur allouant (ainsi qu'aux

données qu'elles traitent) de la place en mémoire, en gérant l'accès des unes et des autres aux périphériques et aux mémoires externes et, éventuellement, en permettant successivement aux diverses tâches en cours "d'avoir la main" (= de commander le processeur) pendant de courts intervalles de temps.

#### 7.2.1.2.1 Le système d'exploitation est le "socle logiciel" sur lequel reposent les logiciels d'application

Bon nombre d'actions commandées à un logiciel d'application (charger et ouvrir un fichier de données, enregistrer les données traitées sous la forme d'un fichier sur les mémoires externes, imprimer les données traitées, envoyer ces données vers un autre ordinateur, à travers un réseau,...) sont en réalité prises en charge par des programmes du système d'exploitation.

C'est donc dire que sans la présence de ce dernier (présence active, mais cachée à l'utilisateur) les logiciels d'application sont incapables de "tourner". De plus, chaque logiciel d'application a été conçu pour "se reposer" sur un système d'exploitation bien précis. On dit alors que ce logiciel d'application "tourne" **sous** tel système d'exploitation; il vaudrait mieux pourtant dire qu'un logiciel tourne "au dessus" de tel système d'exploitation, puisqu'il sous traite à ce dernier un certain nombre des actions à effectuer.

> On parlera ainsi de "Word 6 sous Windows 3.1", "Lotus sous MS-DOS", "Presenter sous Linux",... A chaque fois, on précise quelle est la couche système d'exploitation pour laquelle tel logiciel d'application a été prévu. Il arrive d'ailleurs qu'un logiciel d'application existe pour plusieurs systèmes d'exploitation, ainsi "Word sous Windows 95 (sur PC)" ou "Word sous Système 7 (sur MacIntosh)". Mais on comprend que ces versions ne sont évidemment pas interchangeables.

Un des rôles du système d'exploitation est d'ailleurs en quelque sorte de gommer les différences matérielles (de détails) qui existeraient entre divers modèles d'ordinateurs, pour que, quelle que soit (dans une certaine mesure) la "couche matérielle", la couche "logiciel d'application" puisse en tirer parti puisque, entre les deux, la couche "système d'exploitation" vient en quelque sorte "gommer" les différences.

#### 7.2.1.2.2 C'est le système d'exploitation qui organise l'occupation de la mémoire centrale

Une tâche, c'est un logiciel (programme) et les données actuellement traitées par ce dernier. Ces tâches (programmes + données) doivent être placées en mémoire centrale, sans empiéter les unes sur les autres : c'est l'un des rôles (cachés) du système d'exploitation que d'allouer des zones de mémoire centrale aux diverses tâches en cours et de gérer l'arrivée de nouvelles tâches et la terminaison d'autres.

> Nous avions évoqué page 105 cette question de savoir comment un programme se trouvait logé en mémoire centrale, à partir de telle ou telle adresse. Nous avions signalé que c'était un autre programme qui se chargeait de cette installation. Nous savons à présent que c'est le système d'exploitation qui joue ce rôle pour les programmes d'application.
>
> Mais, qui installe le système d'exploitation ?

#### 7.2.1.2.3 C'est le système d'exploitation qui alloue certaines ressources à certaines tâches

Non seulement le système d'exploitation gère les allocations de la mémoire centrale, mais c'est également lui qui assigne pour une période donnée les périphériques et les mémoires de masse à certaines tâches qui les nécessitent.

En résumé, c'est le système d'exploitation qui organise la cohabitation de tâches différentes et qui gère et assigne les diverses ressources disponibles : processeur, mémoire centrale, périphériques, mémoires de masse,...

### *7.2.2 Le chargement du système d'exploitation en mémoire : le bootstrapping*

Nous savons qu'à l'allumage de l'ordinateur, seuls les programmes présents en ROM (mémoire morte) au sein du BIOS sont disponibles (Voir page 62).

Nous savons que parmi ces programmes présents dans le BIOS figure un programme particulier, le programme de bootstrapping (ou plus simplement de boot) dont le rôle consiste à aller chercher à un endroit déterminé des mémoires externes (disque-dur ou CD ou disquette) des programmes supplémentaires constituant la partie résidente (en mémoire centrale) du système d'exploitation.

Le système d'exploitation est donc en quelque sorte composé de trois parties :

- un ensemble de programmes correspondants à des actions extrêmement rudimentaires et présents dans la ROM;
- un ensemble de programmes, présents sur le disque dur (ou une autre mémoire externe) et qui sont chargés au démarrage en mémoire vive et y resteront;
- un ensemble plus important de programmes utilitaires qui sont présents sur le disque dur et ne seront appelés en mémoire centrale qu'au moment où ils deviennent indispensables.

### *7.2.3 Quelques systèmes d'exploitation*

Un système d'exploitation est en général dédicacé à un type d'ordinateur, ou plus précisément à un type de processeur. En effet, le système d'exploitation constitue un ensemble de programmes exécutables et , comme nous le savons, un programme exécutable est toujours écrit pour un type précis de processeur (dans le langage machine de ce dernier) et ne convient évidemment pas pour un autre type de processeur (avec un langage machine différent).

Les premiers temps de la micro-informatique (de la création de l'Apple II à celle des premiers PC) ont vu apparaître des dizaines d'ordinateurs différents, chacun d'eux étant équipé d'un système d'exploitation spécifique et généralement extrêmement rudimentaire.

L'Apple II qui fut l'un des ordinateurs phares de cette époque connaissait deux systèmes d'exploitation différents : **AppleSoft** d'une part (très rudimentaire et bâti autour d'un interpréteur Basic), et le **p-Système UCSD** d'autre part (assez évolué et construit autour d'un environnement de programmation Pascal). Par ailleurs, on pouvait, en adjoignant une carte à l'Apple II transformer complètement ce dernier puisque on lui adjoignait un autre processeur (le Z 80) que son processeur natif (le 6502 de Motorola). Équipé de ce second processeur, il "tournait" alors sous le système d'exploitation **CPM**.

En 1981, apparaît le PC d'IBM et le monde de la micro-informatique s'en trouve complètement bouleversé. IBM fait le choix pour son PC du système d'exploitation **MS-DOS** de Microsoft en le rebaptisant au passage **PC-DOS**.

Mais avec l'explosion des "PC-compatibles" (c'est à dire des ordinateurs très proches du PC d'IBM mais construits par d'autres), c'est MS-DOS qui s'impose comme le système d'exploitation qui accompagnera l'évolution des ordinateurs de type PC (construits autour de la lignée des processeurs d'Intel, voir page 112). Mais l'interface utilisateur de MS-DOS travaille seulement en mode texte (voir ci-après).

Pendant ce temps, Apple sort en 1984 son ordinateur MacIntosh et le dote d'un système d'exploitation équipé d'une interface graphique qui intègre l'usage de la souris.

Après plusieurs années, Microsoft ajoutera à MS-DOS la couche d'interface graphique baptisée Windows. Défileront alors les versions successives de Windows et, pour ne pointer que les plus connues, on verra se succéder **Windows 3.0**, **Windows 95**, **Windows 98**, **Windows 2000** et le dernier né **Windows XP** (12/2002).

Mais on ne peut ignorer, lorsqu'il est question des systèmes d'exploitation, l'un des plus fameux d'entre eux : **UNIX**, qui a en quelque sorte donné naissance à **LINUX**.

## 7.3 L'évolution des interfaces d'utilisation des logiciels et des systèmes d'exploitation

Qu'il s'agisse du système d'exploitation (à travers lequel l'utilisateur gère l'organisation des mémoires de masse, détermine certaines configuration des périphériques et "lance" des applications (= des programmes)) ou des logiciels d'application (que l'utilisateur utilise pour des tâches particulières : production de textes, retouches d'images, courrier électronique,...), une partie importante de chacun de ces logiciels est consacrée à la gestion des interactions entre l'utilisateur et le "reste" du logiciel considéré.

Cette partie du logiciel qui va permettre à l'utilisateur de donner des **commandes** qui vont provoquer certains traitements (= certaines actions modifiant les données traitées par le logiciel) et à travers lequel le logiciel réagira (par exemple en affichant à l'écran certaines choses) est **l'interface utilisateur** de ce logiciel.

L'utilisateur va donc pouvoir agir à travers les **périphériques d'entrée** sur le couple ordinateur-logiciel; ces actions seront prises en charge et interprétées par une partie du logiciel constituant l'interface-utilisateur; à l'inverse, cette interface renverra à travers les **périphériques de sortie** les réactions du même couple ordinateur-logiciel.

Dans ce dialogue homme-machine, l'écran joue évidemment un rôle primordial (en "montrant" les réactions de l'ordinateur (guidé par le logiciel considéré)); de la même manière, clavier, souris, micro sont les canaux à travers lesquels l'utilisateur agit.

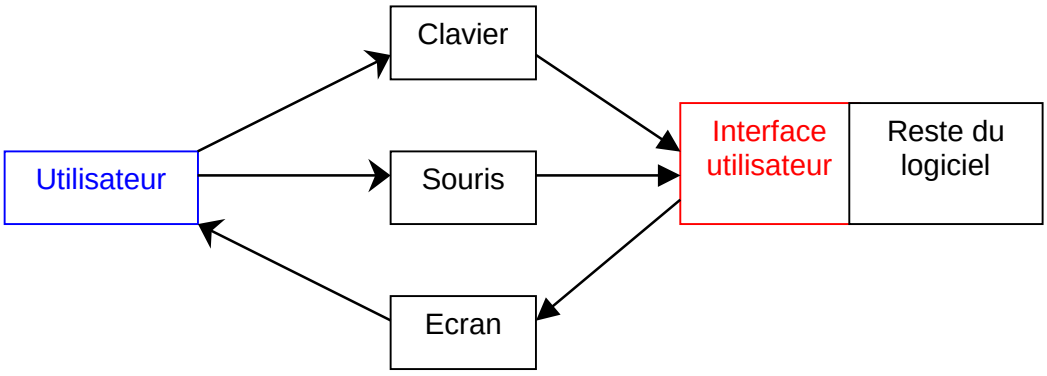



*Figure 7-5 : le rôle de l'interface-utilisateur*

Au cours du temps et de l'évolution de l'informatique les caractéristiques de cette interface, conditionnant donc les modalités du dialogue homme-machine ont évolué, en même temps d'ailleurs que les périphériques et la puissance des machines.

Nous ne remonterons pas aux premiers temps de l'informatique, celui ou les utilisateurs n'existaient pas et ou les traitements effectués par l'ordinateur ne mettaient pas en oeuvre une interaction "en direct" entre l'homme et la machine. Les programmes s'exécutaient

en général sans intervention humaine extérieure : on avait fourni à un lecteur de cartes perforées des paquets de cartes comportant le programme (exécutable) d'une part, les données d'autre part et on récoltait sur un "listing" sortant d'une grosse imprimante les résultats des traitement commandés par le programme.

En gros, l'interaction remonte au temps où un clavier a permis à l'utilisateur d'intervenir pendant l'exécution d'un programme (en commandant "en direct" certains traitements à effectuer par le logiciel ou en fournissant certaines données) et où l'écran a permis de visualiser également en direct les "réactions" du couple logiciel-ordinateur.

> Je parle parfois des réactions du logiciel ou de l'ordinateur; nous savons depuis longtemps que l'une et l'autre de ces locutions sont incorrectes : les réactions sont toujours celles du couple ordinateur-logiciel.

### *7.3.1 Les interfaces en mode texte*

Dans les premiers temps du dialogue, les deux seuls outils étaient le clavier (pour **frapper** des commandes ou des données au logiciel) et l'écran (où l'on pouvait **lire** les résultats "écrits" des traitements).

Voici un exemple typique de ces interactions. Je ne peux évidemment pas y montrer vraiment la frappe effectuée par l'utilisateur au clavier; mais comme, heureusement, les caractères frappés apparaissaient au fur et à mesure à l'écran, on peut se rendre compte de ce qui est de la responsabilité de l'utilisateur : cela apparaître ici en rouge. En noir, sur ces copies d'écran, les "interventions" de l'ordinateur.

#### *7.3.1.1 Premier exemple : un programme*

Voici les écrans successifs montrant le dialogue entre l'utilisateur et le système :

```
vous allez me fournir une somme et je vous dirai combien de
billets et de pièces elle représente. Attention,
la somme ne peut depasser 32767 francs.
Entrée pour poursuivre!_
```

```
Quelle somme ? 5767
Cela fait
  2 billets de 2000 francs
  1 billet de 1000 francs
  1 billet de 500 francs
  2 billets de 100 francs
  1 pièce de 50 francs
  3 pièces de 5 francs
  2 pièces de 1 franc
On continue oui ou non ?oui_
```

*Figure 7-6 : un dialogue en mode texte*

#### *7.3.1.2 Deuxième exemple : le système d'exploitation*

Voici un exemple d'interaction avec le système d'exploitation MS-DOS : tout y prenait également la forme de texte (frappé par l'utilisateur ou affiché par le système).

La commande fournie demande l'affichage de la liste de tous les fichiers de suffixe **pas** présents dans le sous-dossier **chaine** du sous-dossier **pascal** du dossier projet présent sur le disque dur **d:**.

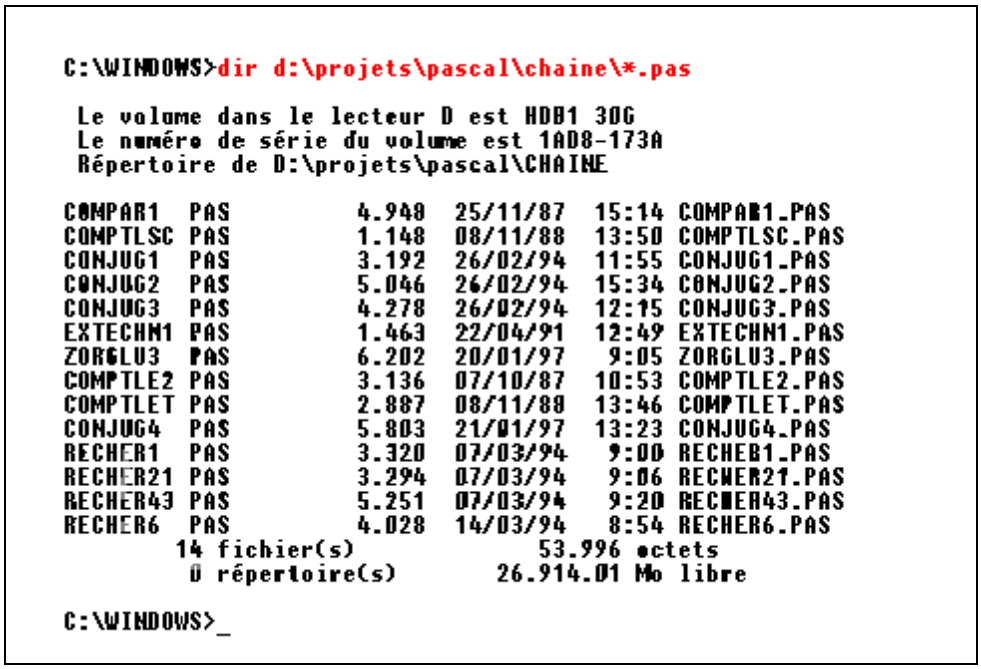



*Figure 7-7 : MS-DOS, un système d'exploitation avec une interface en mode texte*

Bien évidemment, l'utilisateur devait connaître la syntaxe précise des commandes permises : ce n'était pas toujours immédiat, comme le montrent les quelques exemples illustrés ci-dessous :

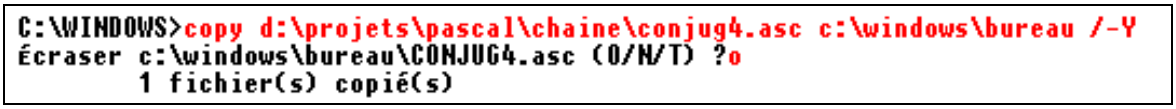




*Figure 7-8 : MS-DOS, d'autres illustrations du mode texte*

Ces interfaces en mode texte ont été la règle pendant les premières années d'existence de la micro-informatique. Dans l'univers PC, cela a subsisté jusqu'au milieu des années 80; l'univers Apple (avec le MacIntosh) a beaucoup plus rapidement intégré des interfaces de type graphique.

### *7.3.2 Les interfaces semi-graphiques*

Dans l'univers PC (dont le système d'exploitation était essentiellement MS-DOS), les interfaces en mode texte ont été un temps remplacées par des interfaces semi-graphiques, qui intégraient déjà l'usage de la souris et les menus déroulants à travers lesquels le système permettait à l'utilisateur de choisir les commandes souhaitées.

Voici à titre d'exemple, l'écran présenté par un éditeur de texte "plein écran" : c'est, comme son nom l'indique un outil servant à confectionner du texte (sans mise en forme). Comme on le voit, il intègre des facilités comme les menus déroulants (permettant de choisir les commandes à donner, en cliquant avec la souris et sans plus avoir à retenir des termes plus ou moins ésotériques), les barres de défilement (verticales et horizontales), etc.

Le "look" est plus ou moins "graphique", mais il faut savoir que les écrans présentés, même quand ils intègrent des encadrements et des objets graphiques, sont en réalité

essentiellement constitués de caractères et font donc grand usage des caractères graphiques rencontrés lorsque nous avons abordé le code ASCII (Voir page 22).


*Figure 7-9 : un logiciel avec interface semi-graphique*

Voici par ailleurs deux écrans successifs illustrant le dialogue avec le système d'exploitation à travers une interface intégrant également menus déroulants et usage de la souris pour fournir les commandes (dans les menus), pour sélectionner les fichiers, et pour effectuer des actions comme la copie, le déplacement, la suppression de fichiers et de dossiers.

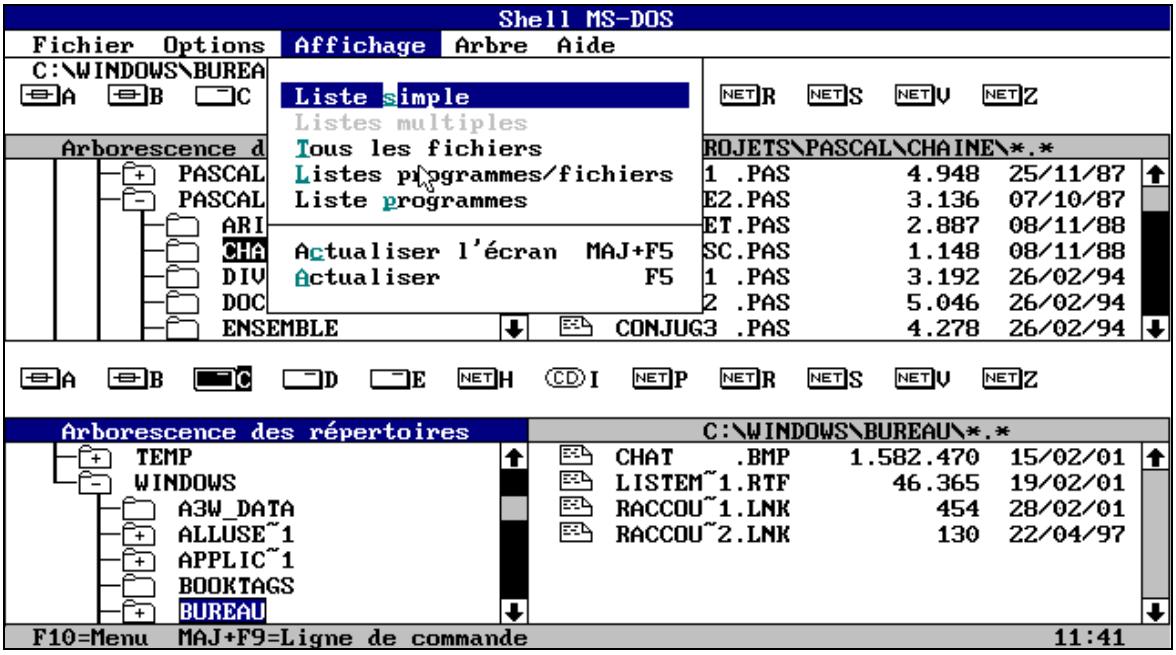



*Figure 7-10 : une interface semi-graphique pour le système d'exploitation*

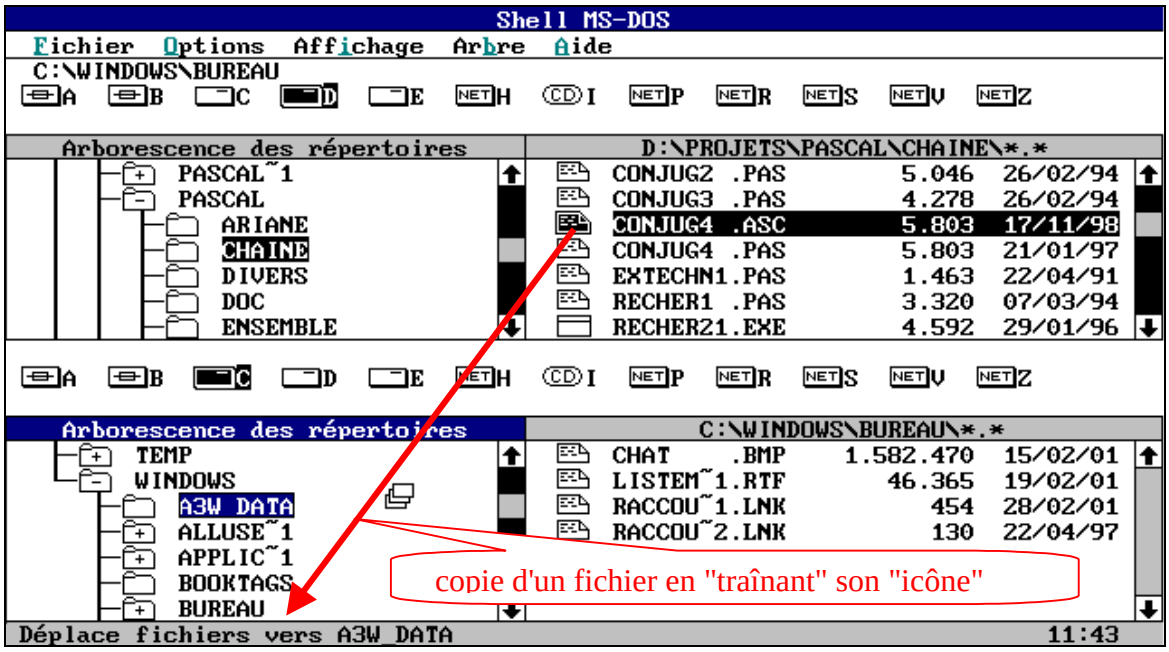



*Figure 7-11 : copie de fichier à travers une interface semi-graphique*

Ces prémices d'une interface vraiment graphique et intégrant de plus en plus l'usage de la souris ont eu une très brève durée d'usage dans l'univers PC; lorsqu'ils sont apparus, de véritables interfaces graphiques caractérisaient déjà le monde MacIntosh chez Apple.

### *7.3.3 Les interfaces graphiques*

Dans l'univers PC, ce sont les interfaces des versions successives du système d'exploitation Windows et des logiciels d'application correspondants qui illustrent les caractéristiques essentielles de ce que devient le dialogue homme-machine.

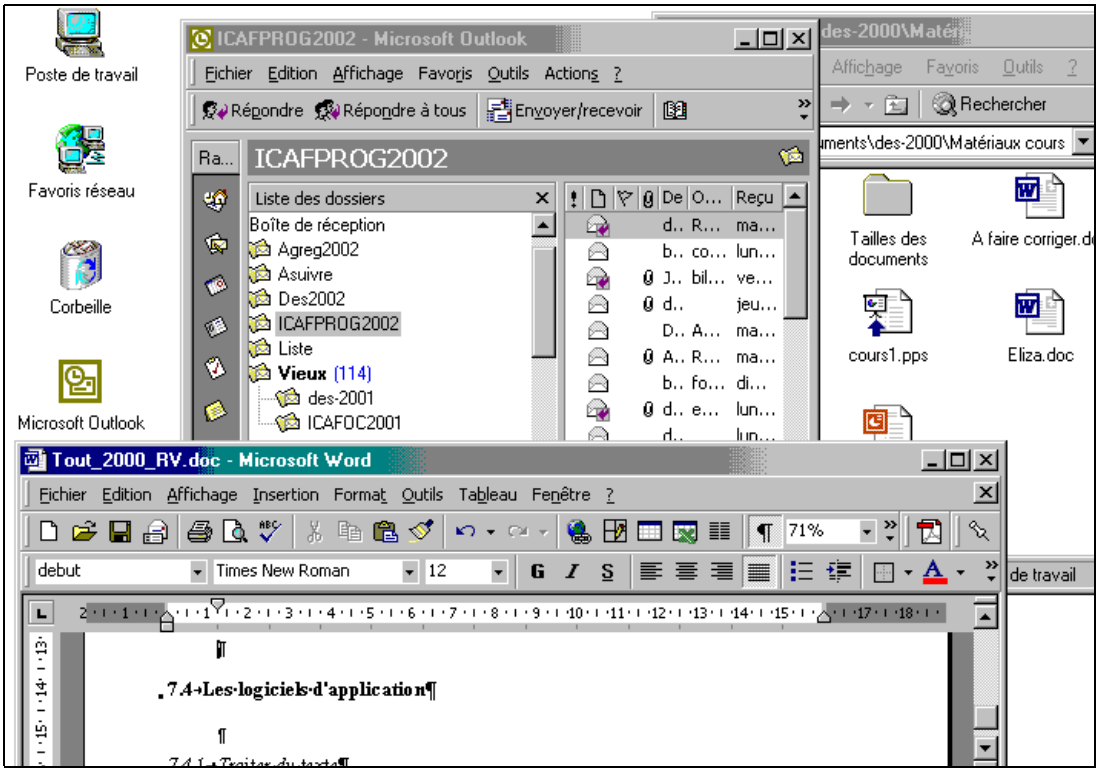



*Figure 7-12 : une interface graphique*

Comme le montre l'illustration ci-dessus, cette interface graphique est caractérisée par :

- un environnement **fenêtré** : les applications s'exécutent dans des fenêtres (que l'utilisateur peut déplacer, redimensionner,...);
- l'usage d'**icônes** qui accompagnent les divers fichiers qui sont listés et caractérisent d'une certaine manière les types de ces fichiers (exécutable, texte, image, son,...);
- l'usage, tant au niveau du système d'exploitation que des logiciels d'application associés, de **menus déroulants**, de **menus contextuels**, de **boîtes de dialogue**,... qui encouragent un usage constant et sophistiqué de la souris (avec les clics gauche ou droit, les doubles clics, les CTRL clics,...);
- l'usage de **métaphores** censées aider l'utilisateur, comme celles du bureau, de la corbeille,...;
- la faculté de lancer **plusieurs tâches** en parallèles, de passer aisément de l'une à l'autre, d'échanger des informations entre elles,...;

Nul doute que dans les années à venir, ces interfaces intégreront de plus en plus l'usage du micro et de commandes orales qui ajouteront des modes d'interventions supplémentaires à l'utilisateur.

## 7.4 Les logiciels d'application

Nous venons de voir que le rôle essentiel d'un système d'exploitation est de servir de socle, de supporter les logiciels d'application qui ont été développés en s'appuyant sur ce système d'exploitation.

Chaque système d'exploitation est donc accompagné d'un cortège de logiciels d'application plus ou moins spécialisés et dédicacés à des tâches spécifiques : logiciels de bureautique (traitement de texte, publication assistée par ordinateur (PAO), tableur, gestionnaire de bases de données), logiciels de présentation, logiciels de courrier électronique, logiciels de navigation sur le WEB, logiciels graphiques, logiciels de traitement du son, de la vidéo,... Sans oublier tous les logiciels spécialisés d'aide à des professions particulières : dessin assisté par ordinateur, gestion d'officine de pharmacie ou d'étude notariale, gestion de bibliothèque,...

Bref, ce sont des milliers d'instruments logiciels qui sont mis à la disposition de leurs utilisateurs potentiels. Il serait donc vain de vouloir en faire un recensement et plus absurde encore de tenter d'en présenter les usages ou les modes d'utilisation. Et cela d'autant que pour chacun de ces logiciels, c'est souvent plusieurs versions qui se sont succédées (et continueront à se succéder).

# 8

## ... traiter l'information ...

## Communiquer l'information : télématique, réseaux, Internet

# 9

# Bibliographie

## 9.1 Chapitre 1 : généralités, découverte de l'informatique


*ACM Turing Award Lectures; The First Twenty Years. 1966-1985*. ACM Press, New York, 1987.

*Initiation à l'informatique*., Amsterdam : Editions Time-Life, 1986.

*L'ère de l'informatique*., Amsterdam : Editions Time-Life, 1987.

BERUBE Y., *Initiation aux ordinateurs*., Paris, London, New York : McGraw-Hill, Editeurs, 1980.

BRETON P., *Une histoire de l'informatique*, Paris : Editions du Seuil, 1990.

CLAVIEZ J., *Micro-ordinateurs. Que peut-on faire avec ?*, Montréal : JCI Inc., 1993.

☞ LABIN E., *Comprendre l'informatique*. Bordas, Paris, 1973.

LE ROCH J-C., *Regards sur l'informatique*. Edition Marketing, Paris, 1991.

LIGONNIERE R., *Préhistoire et histoire des ordinateurs*, Paris : Robert Laffont, 1987.

RALSTON A., REILLY E., *Encyclopedia of Computer Science*, London : Chapman & Hall, 1993.

SANDERS D.H., *L'univers des ordinateurs*., Paris, London, New York : McGraw-Hill, Editeurs, 1984.

☞ STEHLE J-L., HOCHARD P., *Ordinateurs et langages*. Edition Marketing, Paris, 1989.


## 9.2 Chapitre 2 : traitement formel, informatique et intelligence


☞ ARSAC J., *Les machines à penser. Des ordinateurs et des hommes*. Editions du Seuil, Paris, 1987.

BULL , *Intelligence artificielle et bon sens*. Collection F.R. BULL, Masson, Paris, 1991

CHANGEUX J-P., CONNES A., *Matière à pensée* Editions Odile Jacob, Paris, 1989.

DEFAYS D., *L'esprit en friche. Les foisonnements de l'intelligence artificielle* Pierre Mardaga, Bruxelles, Liège, 1988.

GANASCIA J-G., *L'âme machine Les enjeux de l'intelligence artificielle* Editions du Seuil, Paris, 1990.

HOFSTADTER D., *Godel, Escher, Bach*. InterEditions, Paris, 1985.

☞ HOFSTADTER D., DENNETT D. (EDIT)., *Vues de l'esprit. Fantaisies et réflexions sur l'être et l'âme*. InterEditions, Paris, 1987.

LUCAS Y. *Codes et machines : essai de sémiologie industrielle*. Presses Universitaires de France, Paris, 1974.

VARELA F.J., *Connaître les sciences cognitives. Tendances et perspectives*. Editions du Seuil, Paris, 1989.

WEIZENBAUM J., *Puissance de l'ordinateur et raison de l'homme*. Les Editions d'Informatique, Paris, 1981.

LAZORTHES G., *Le cerveau et l'ordinateur*., Toulouse : Privat, 1988.

SIMONS G., *L'ordinateur est-il vivant ?, Evolution et nouvelles formes de vie*., Londreys, 1984.

## 9.3 Chapitre 3 : codage des informations

BELHOMME D., BERHIN M., ..., *Les carnets de la formation multimédia*., Coédition CeFIS - Média Animation, 1999.

## 9.4 Chapitre 4 : point de vue de l'utilisateur ou du programmeur

DUCHATEAU C. (1992) "Comment définir une culture informatique", *Journal de Réflexion sur l'informatique*, n° 23, octobre 1992, pp. 34-39.

DUCHATEAU C. (1994) "Faut-il enseigner l'informatique à ces utilisateurs", *Actes du quatrième colloque francophone sur la didactique de l'informatique*, Montréal, avril 1994.

## 9.5 Chapitre 5 : architecture d'un système informatique

CLEMENT L., *Périphériques : disques, imprimantes et écrans*. A. de Boeck, Bruxelles, 1989.

STROHMEIER A., *Le matériel informatique* : *concepts et principes*. Presses Polytechniques et Universitaires Romandes, Lausanne, 1986.

MEINADIER J-P., *Structure et fonctionnement des ordinateurs*., Paris : Librairie Larousse, 1975.

*Les entrées-sorties*., Amsterdam : Editions Time-Life, 1987.

*Histoires de puces*., Amsterdam : Editions Time-Life, 1989.

*Les mémoires*., Amsterdam : Editions Time-Life, 1988.

## 9.6 Chapitre 6 : la programmation

DUCHATEAU C., *Quand le savoir faire ne suffit plus. Qu'y a-t-il au cœur, de la pensée algorithmique et de la programmation*., Namur : CeFIS, Facultés N-D de la Paix, 1989.

DUCHATEAU C., *Images pour programmer. Apprendre les concepts de base*. De Boeck-Wesmael, Bruxelles, 1990.

*Les langages de programmation*., Amsterdam : Editions Time-Life, 1987.

LESUISSE R., BORSU A., *Initiation aux raisonnements de la programmation*., Namur : Presses Universitaires de Namur, 1987.

## 9.7 Chapitre 7 : le logiciel

*Les communications*., Amsterdam : Editions Time-Life, 1989.

*Le logiciel*., Amsterdam : Editions Time-Life, 1987.

# 10

# Tables et Index

## 10.1 Table des matières

## 10.2 Table des figures

## 10.3 Table des principes

## 10.4 Index

## V