Andrew Tanenbaum

Architecture de l'ordinateur

Du circuit logique au logiciel de base

Texte français de
Jean-Alain Hernandez et René Joly

L'édition originale de cet ouvrage a été publiée aux États-Unis par Prentice-Hall, Inc., Englewood Cliffs, N.J., sous le titre *Structured Computer Organization*, second edition. © 1984 by Prentice-Hall, Inc.

Nouveau tirage corrigé, novembre 1988

ISBN 2-7296-0133-3

Sommaire

Préface de la deuxième édition américaine

L'idée de base de la première édition de cet ouvrage était qu'on pouvait analyser un ordinateur suivant une architecture en couches, chaque couche réalisant une fonction bien définie. L'intérêt de ce concept fondamental est tel qu'il va se trouver encore renforcé dans cette seconde édition. Dans l'édition précédente, quatre couches —la couche microprogrammée, la couche machine traditionnelle, la couche système d'exploitation et la couche langage d'assemblage— étaient analysées. Dans cette nouvelle édition, ces quatre couches sont conservées, auxquelles on adjoint une nouvelle couche, la couche physique, située au-dessus de la couche microprogrammée.

La raison en est que l'apparition des micro-ordinateurs de très bas prix fait que les connaissances en matériel informatique, et plus particulièrement en circuiterie logique, sont beaucoup plus importantes qu'autrefois. Autrefois, l'informaticien moyen n'approchait guère de l'ordinateur sur lequel il travaillait, si ce n'est pour placer un paquet de cartes perforées dans un lecteur. Il n'avait pas besoin de connaître grand-chose au matériel puisqu'une équipe d'ingénieurs système s'occupait de faire tourner la machine. Bientôt, en revanche, tous les informaticiens auront un ordinateur sur leur bureau et un autre chez eux. De plus, les progrès techniques dans le domaine des VLSI transforment beaucoup de programmeurs en concepteurs de circuits, estompant ainsi la frontière autrefois très nette entre matériel et logiciel. L'informaticien ayant donc beaucoup plus de contacts avec le matériel, il lui est nécessaire de bien comprendre la façon dont un ordinateur fonctionne. C'est l'objet d'un nouveau chapitre: le chapitre 3.

Ce nouveau chapitre n'est pourtant pas le seul changement qui survient dans cette édition. J'ai mis un peu plus l'accent sur les petites machines et un

peu moins sur les grosses, suivant en cela l'évolution de l'industrie informatique. Dans la première édition, l'IBM 370, le Cyber de Control Data et le PDP-11 de Digital Equipment servaient de base à tous les exemples. Dans cette seconde édition, le Cyber disparaît pour être remplacé par deux microprocesseurs très connus: le Z80 de Zilog (une machine 8 bits) et le MC68000 de Motorola (une machine 16 ou 32 bits suivant le point de vue que l'on adopte). L'architecture du 370 est encore très actuelle, puisqu'elle est utilisée non seulement dans la série IBM 370 mais encore dans les IBM 43xx ou 303x, les ordinateurs Amdahl et bien d'autres. Quant au PDP-11, il reste encore très diffusé, ce qui explique son maintien dans cette seconde édition.

Autre changement important: l'utilisation du Pascal (et non plus du PL/I) pour la description des algorithmes. Pascal est plus simple et plus élégant mais c'est aussi un langage beaucoup plus diffusé maintenant que PL/I.

De nombreuses parties de cet ouvrage ont été soit complètement récrites, soit mises à jour. Pour donner une idée de l'importance des modifications, on peut signaler que sur les 236 figures présentes dans cette édition, 132 n'existaient pas dans l'édition précédente. Un bref tour d'horizon des différents chapitres permettra de mieux expliquer ces changements.

Le chapitre 1 introduit les bases de l'architecture en couches. Il n'a guère changé par rapport à l'édition précédente.

Le chapitre 2, en revanche, voit la partie réseaux (qu'il s'agisse de réseaux locaux ou de réseaux généraux) considérablement augmentée. Le reste du chapitre est sans grand changement, à l'exception de quelques modifications dans l'ordre de présentation et d'une étude plus poussée des systèmes distribués.

Le chapitre 3 est entièrement nouveau. Il traite de la couche physique dont il n'était pas fait mention dans l'édition précédente. Ce chapitre commence par l'étude des composants de base de tout ordinateur, les portes; il se poursuit par l'étude des circuits tels que les mémoires et par l'étude des bus et il se termine par la présentation d'un ordinateur très simple mais complet (figure 3.39) que tout lecteur devrait comprendre même s'il n'avait aucune connaissance en électronique avant d'entamer ce chapitre.

Le chapitre 4 (couche microprogrammée) est maintenant placé avant la couche machine traditionnelle, pour obtenir une présentation dans un ordre logique: de la base au sommet. Ce chapitre a été entièrement récrit, excepté la partie qui concerne la couche microprogrammée de l'IBM 370/125. Il s'appuie maintenant sur une micro-architecture plus réaliste (celle des processeurs en tranches AMD 2903), tout en restant suffisamment simple pour que des débutants le comprennent. La longue analyse de la multiplication et de la division a été supprimée et, dans l'exemple détaillé, le PDP-11/40 a été remplacé par le PDP-11/60. Ce chapitre commence par un bref rappel des éléments composant le chapitre 3, ce qui permet, dans un cours, d'ignorer ce dernier si nécessaire.

Le chapitre 5 (couche machine traditionnelle) contient beaucoup de choses nouvelles concernant le Z80 et le MC68000. Pour leur faire place, le passage concernant les structures de données a été supprimé. Plusieurs lecteurs m'avaient fait remarquer que ce passage n'avait peut-être pas sa place dans un livre sur la structure des ordinateurs.

Le chapitre 6 (couche système d'exploitation) comporte plusieurs changements importants. Et d'abord l'ajout de deux nouveaux exemples: UNIX[1] et CP/M[2] . S'ils sont tous deux largement utilisés, ils n'en sont pas moins extrêmement différents. Le dernier paragraphe qui concerne les langages de commande a été récrit en utilisant le «shell » d'UNIX plutôt que la version CDC.

Le chapitre 7 (couche langage d'assemblage) a été raccourci. J'y ai supprimé ce qui concernait les algorithmes de recherche et de tri qui ne sont pas vraiment l'objet de ce cours.

Le chapitre 8 (machine multi-couches) a été considérablement modifié pour rendre compte entre autres choses la réapparition récente du concept UNCOL.

Le chapitre 9 (bibliographie) a été mis à jour.

Les annexes A (numération binaire) et B (nombres en virgule flottante) sont essentiellement restées les mêmes.

Il me reste à remercier tous ceux qui, de près ou de loin, ont participé à l'élaboration de cet ouvrage, ainsi que les entreprises qui m'ont permis de tirer les informations de leurs publications et plus particulièrement: Digital Equipment Corporation, IBM, Motorola, Texas Instruments et Zilog. Il est bien évident que je prends la responsabilité de toute erreur dans la description de leurs produits.

Andrew TANENBAUM

---

(1) UNIX est une marque déposée des laboratoires Bell.
(2) CP/M est une marque déposée de Digital Research, Inc.

1

Introduction

Un ordinateur est une machine capable de résoudre des problèmes en appliquant des instructions préalablement définies. La suite des instructions décrivant la façon dont l'ordinateur doit effectuer un certain travail est appelée *programme*. Les circuits électroniques de chaque ordinateur ne pouvant reconnaître et exécuter directement qu'un nombre très limité d'instructions, tout programme doit être, avant son exécution, converti pour n'être exprimé qu'avec ces instructions. Celles-ci sont du genre:

- Additionner deux nombres
- Voir si un nombre est égal à zéro
- Changer d'emplacement mémoire une donnée

L'ensemble des instructions exécutables directement par un ordinateur forme un langage qui permet aux gens de communiquer avec cet ordinateur. C'est ce qu'on appelle le *langage machine*.

Lorsqu'on conçoit un nouvel ordinateur, il convient de choisir les instructions qui formeront son langage machine. En général on cherche à ce que ces instructions soient les plus simples possible, compte tenu des performances attendues de la machine, afin de réduire la complexité (et donc le coût) des circuits électroniques nécessaires. Le problème est que ces langages machine sont alors si primitifs qu'il est extrêmement pénible et fastidieux de les utiliser.

Il y a deux façons de résoudre ce problème qui, toutes les deux, visent à construire un nouveau jeu d'instructions plus pratique à utiliser que le langage machine. L'ensemble de ces nouvelles instructions forme un langage, que nous appellerons L2, de la même façon que l'ensemble des instructions machine forme le langage L1. Les deux approches que nous évoquions diffèrent suivant la manière dont les programmes écrits en L2 sont traités par l'ordinateur qui, en dernière analyse, ne peut exécuter que des programmes écrits dans son langage machine L1.

La première façon d'exécuter un programme écrit en L2 est de remplacer dans un premier temps chaque instruction de ce programme par la suite d'instructions en L1 qui est équivalente. On obtient ainsi un nouveau

programme entièrement écrit en L1. L'ordinateur exécute alors ce nouveau programme en L1 et non pas l'ancien programme en L2. Cette technique est appelée *traduction*.

La seconde façon est d'écrire un programme en L1 capable, après avoir examiné chaque instruction d'un programme en L2, d'exécuter directement la séquence d'instructions en L1 équivalente. Cette technique, avec laquelle on n'a pas besoin de générer tout un programme en L1, est appelée *interprétation*, et le programme écrit en L1 qui examine et exécute chaque instruction du programme en L2 s'appelle un *interpréteur* (ou un *interprète*).

Traduction et interprétation se ressemblent beaucoup. Dans chaque cas toute instruction en L2 est finalement convertie en une suite équivalente d'instructions en L1. Mais il faut noter que dans le cas de la traduction tout le programme en L2 est d'abord converti en un programme en L1; puis le programme en L2 disparaît et c'est le programme en L1 qui est exécuté. En revanche, avec l'interprétation chaque instruction du programme L2 est analysée puis immédiatement exécutée. On n'obtient donc pas de programme traduit.

Plutôt que de raisonner en termes de traduction ou d'interprétation, il est souvent plus facile d'imaginer l'existence d'un ordinateur hypothétique, ce qu'on appelle une *machine virtuelle*, dont le langage machine est L2. Si on pouvait construire cette machine à un coût raisonnable on n'aurait pas besoin du langage L1 ni d'une machine capable d'exécuter ce langage L1. Les gens écriraient tout simplement leurs programmes en L2 et la machine exécuterait directement ces programmes. Bien qu'il soit trop coûteux de construire cette machine, on peut néanmoins écrire des programmes en L2 à condition que ces programmes puissent être traduits ou interprétés par un ordinateur (réel celui-ci) dont le langage est L1. En d'autres termes, on peut écrire des programmes pour des machines virtuelles comme si elles existaient réellement.

Pour que la traduction ou l'interprétation reste assez simple, il faut que les langages L1 et L2 ne soient pas trop différents. Ceci implique que très souvent L2, bien que meilleur que L1, n'est pas le langage idéal pour l'écriture d'un programme. Ceci est quelque peu contradictoire avec l'idée même qui avait présidé à l'élaboration de L2: faire un langage plus simple et plus pratique à utiliser par des êtres humains que L1, qui était plutôt fait pour la machine.

On peut être alors amené à définir un nouvel ensemble d'instructions plus proches de l'utilisateur final et donc moins dépendantes de la machine que celles de L2. Ces nouvelles instructions forment un langage L3 à l'aide duquel, comme on l'a vu plus haut, on pourra écrire des programmes comme si une machine basée sur L3 existait. Ces programmes seront, bien sûr, traduits en L2 ou interprétés par un interpréteur écrit en L2.

On peut ainsi concevoir toute une série de langages, chacun étant un peu plus pratique que son prédécesseur, jusqu'à ce qu'on en obtienne un jugé convenable. Chaque langage s'appuie sur son prédécesseur de telle sorte qu'on peut voir un ordinateur comme un empilement de *couches* ou de *niveaux* comme indiqué sur la figure 1.1. Le langage du bas est le plus simple, celui du haut est le plus complexe.

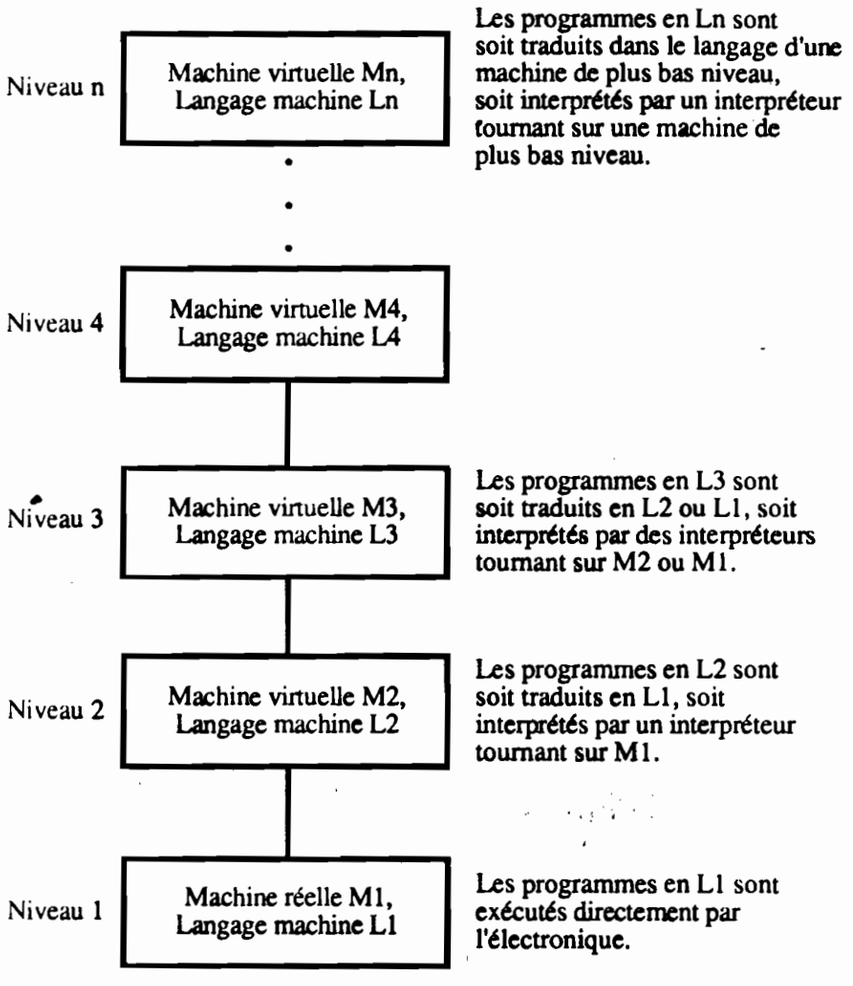

Figure 1.1 Architecture d'une machine multi-niveaux

1.1 LANGAGES ET MACHINES VIRTUELLES

Langage et machine virtuelle sont en relation très étroite. Toute machine a son langage machine formé de l'ensemble des instructions de base qu'elle peut exécuter. On peut donc dire qu'une machine définit un langage. Mais réciproquement un langage définit une machine et, plus précisément, un langage définit la machine qui peut exécuter tous les programmes écrits en ce langage. Bien entendu, la machine définie par un certain langage peut être extraordinairement complexe et d'un coût complètement prohibitif, mais on peut néanmoins l'imaginer. C'est ainsi qu'une machine ayant pour langage

machine Ada, Pascal ou COBOL, quoique très complexe, n'est pas inconcevable et peut-être que dans quelques années de telles machines existeront couramment.

Un ordinateur composé de n couches peut être vu comme n machines virtuelles distinctes, chaque machine virtuelle ayant son propre langage. Dans ce qui suit nous utiliserons indifféremment les termes «machine virtuelle», «couche» et «niveau». Il n'y a que les programmes écrits en L1 qui peuvent être véritablement traités par les circuits électroniques, sans qu'il soit nécessaire d'envisager pour eux une quelconque traduction ou interprétation.

Les programmes écrits en L2, L3, ..., doivent être soit interprétés par un interpréteur d'un plus bas niveau soit traduits en un langage d'un plus bas niveau.

Le programmeur qui a des programmes à écrire pour une machine virtuelle de niveau n n'a pas à se soucier des compilateurs ni des interpréteurs sous-jacents; il n'est pas très intéressant de savoir si ces programmes sont traduits en un certain langage donnant ainsi d'autres programmes eux-mêmes traduits en un langage de plus bas niveau..., ou s'ils sont exécutés directement par le matériel. Dans les deux cas on obtient le même résultat: les programmes sont exécutés.

La plupart des programmeurs travaillant sur une machine de niveau n ne sont intéressés que par le niveau supérieur, celui qui ressemble le moins au langage machine du bas niveau. Cependant, les gens qui s'intéressent au fonctionnement d'un ordinateur doivent étudier tous les niveaux. Ceux qui veulent concevoir de nouvelles machines ou de nouvelles couches (c'est-à-dire de nouvelles machines virtuelles) doivent aussi connaître d'autres couches que la couche supérieure. Les concepts et les techniques de construction de machines vues comme un empilement de couches ainsi que l'analyse détaillée de quelques couches particulièrement importantes forment l'essentiel de cet ouvrage. Le terme *architecture en couche des ordinateurs* provient du fait que cette vision d'un ordinateur aide à la compréhension de son fonctionnement. On peut également remarquer que construire une machine par empilement de couches permet d'obtenir un produit bien structuré.

1.2 LES MACHINES MULTI-COUCHES ACTUELLES

La plupart des ordinateurs actuels ont au moins deux niveaux. On trouve même couramment des machines à six niveaux comme indiqué en figure 1.2. Au niveau 0, à la base, on trouve le matériel. Les circuits électroniques exécutent les programmes en langage machine du niveau 1. Pour être complet, nous devrions mentionner l'existence d'un niveau encore inférieur au niveau 0. Ce niveau, qui n'apparaît pas sur la figure 1.2 parce qu'il est du domaine du génie électrique et donc ici hors sujet, est parfois appelé niveau composant. Ce qui est manipulé à ce niveau, ce sont des transistors qui sont en quelque sorte les «atomes» des concepteurs de machines. (Bien sûr on pourrait parler du fonctionnement des transistors, mais cela nous entraînerait dans la physique du solide!).

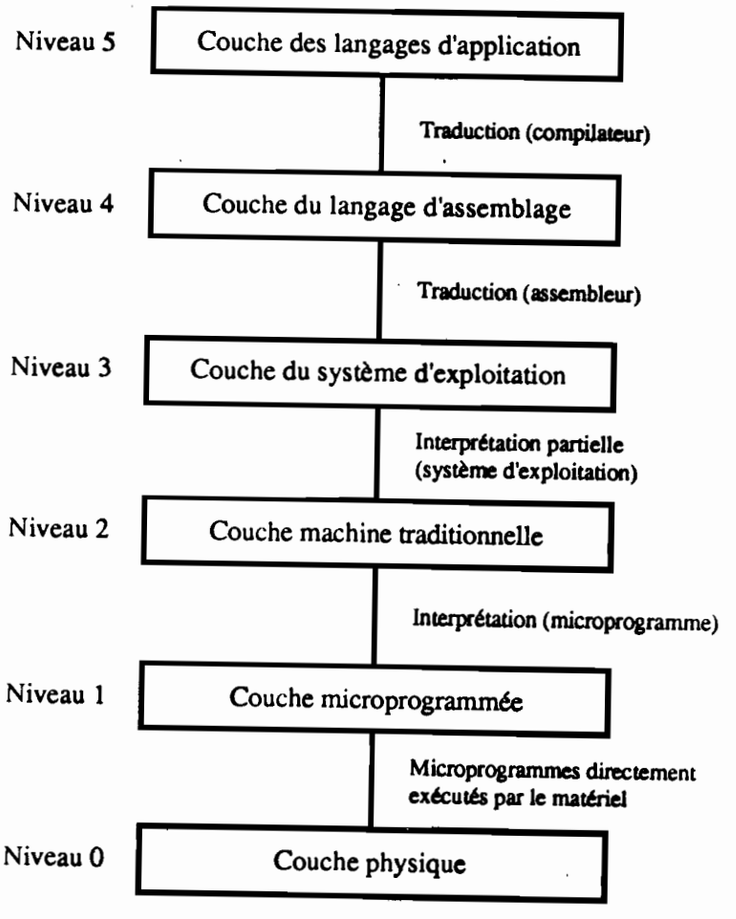

Figure 1.2 Les six couches (ou niveaux) de la plupart des ordinateurs actuels

Le niveau 0, le premier à nous intéresser, est appelé *niveau physique*. Les objets manipulés à ce niveau sont des portes logiques ou numériques, et non analogiques comme pouvaient l'être les transistors. Chaque porte dispose d'une ou plusieurs entrées logiques (signaux représentant 0 et 1) et rend comme résultat une fonction simple de ces entrées comme ET ou OU. Une porte se construit à l'aide de transistors. Nous le verrons au chapitre 3 dont l'objet est l'étude de la couche physique, celle relative aux circuits logiques.

Le niveau suivant est le niveau 1 ou niveau du langage machine. A la différence du niveau 0 dans lequel il n'y a pas trace de programme (ou suite d'instructions à exécuter), le niveau 1 comprend un programme, appelé *microprogramme*, dont le travail est d'interpréter les instructions de niveau 2. Nous appellerons ce niveau le niveau ***microprogrammé***. Bien que deux ordinateurs n'aient jamais exactement des niveaux microprogrammés

identiques, nous essaierons d'en dégager les principales caractéristiques. Par exemple peu de machines disposent à ce niveau de plus de vingt instructions et la plupart de ces instructions consistent à transférer des données ou à exécuter des tests très simples.

Une machine de niveau 1 dispose d'au moins un microprogramme. Chaque microprogramme définit implicitement un langage de niveau 2 (et une machine virtuelle dont le langage machine est ce langage). Toutes les machines de niveau 2 ont beaucoup de points communs, même si ces machines sont de constructeurs différents. Dans ce livre ce niveau sera appelé *niveau machine traditionnel* faute d'un nom qui soit généralement admis.

Les constructeurs d'ordinateurs proposent avec chaque machine qu'ils vendent un manuel souvent intitulé «Manuel de référence du langage machine» ou quelque chose d'approchant. Ces manuels se rapportent au niveau 2 (machine virtuelle) et non au niveau 1. Le jeu d'instructions machine qui y est décrit est l'ensemble des instructions interprétées par le microprogramme et non pas l'ensemble des instructions exécutables directement par le matériel. En d'autres termes, si un constructeur proposait deux interpréteurs de langage de niveau 2 pour une de ses machines, cette machine serait accompagnée de deux manuels de référence du «langage machine», chaque manuel correspondant à un interpréteur.

Notons cependant que certains ordinateurs, et surtout les plus anciens, ne disposent pas de couche microprogrammée. Sur ces machines, les instructions de niveau 2 sont traitées directement par le matériel.

Le troisième niveau est souvent un niveau hybride. La plupart des instructions de son langage figurent également dans le langage du niveau 2. (Il n'y a aucune raison pour qu'une instruction d'un niveau ne puisse être présente à un autre niveau.) On y trouve également des instructions spécifiques, une organisation de la mémoire différente, la capacité d'exécuter plusieurs programmes en parallèle, etc. Il existe plus de différences entre les machines de niveau 3 qu'entre les machines de niveau 1 ou 2.

Les services offerts par le niveau 3 sont pris en charge par un interpréteur s'exécutant au niveau 2 et qui historiquement a été appelé le *système d'exploitation*. Les instructions du niveau 3 qui sont identiques à des instructions du niveau 2 sont traitées directement par le microprogramme et non par le système d'exploitation. C'est en cela que ce niveau, que nous appellerons *niveau système d'exploitation* est hybride.

Il y a un fossé entre les niveaux 3 et 4. Les quatre niveaux inférieurs ne concernent pas directement le programmeur moyen. Ils sont là principalement pour supporter les traducteurs et interpréteurs dont les niveaux supérieurs ont besoin et sont écrits par des spécialistes de la réalisation de machines virtuelles: les *programmeurs système*. Les niveaux 4 et au-dessus concernent le programmeur d'applications qui a un problème concret à résoudre.

Autre caractéristique qui apparaît au niveau 4: la méthode de support des niveaux supérieurs. Les niveaux 2 et 3 sont toujours interprétés. Les niveaux 4, 5 et au-dessus sont souvent, mais pas toujours, associés à des traducteurs.

Il y a d'ailleurs une autre différence entre les niveaux 1,2 et 3 d'un côté et les niveaux supérieurs de l'autre: c'est la nature du langage fourni. Les langages machine des niveaux 1, 2 et 3 sont numériques, donc très difficiles à utiliser puisqu'ils consistent en de longues séries de nombres. Au contraire, à partir du niveau 4, on trouve des langages qui contiennent des mots ou des

abréviations beaucoup plus compréhensibles par les êtres humains (mais pas par la machine).

Ce niveau 4, le niveau langage d'assemblage, est une forme symbolique d'un des langages sous-jacents. Ce niveau permet d'écrire des programmes pour les niveaux 1, 2 et 3 dans une forme moins déplaisante. Les programmes en langage d'assemblage sont d'abord traduits en langage de niveau 1, 2 ou 3 puis sont interprétés par la machine virtuelle ou réelle correspondante. Le programme qui réalise la traduction s'appelle un *assembleur*. Si ce langage d'assemblage était autrefois très important, on peut dire qu'aujourd'hui son importance est bien moindre.

Au niveau 5 on trouve des langages conçus pour être utilisés par des programmeurs d'applications. Ces langages, souvent appelés *langages de haut niveau*, sont extrêmement nombreux. Parmi les plus connus citons Ada, ALGOL 68, APL, BASIC, C, COBOL, FORTRAN, LISP, Pascal et PL/1. Les programmes écrits en l'un quelconque de ces langages sont souvent traduits en niveau 3 ou 4 par des traducteurs appelés *compilateurs*, bien que parfois ils puissent aussi être interprétés.

Les niveaux 6 et au-dessus sont des ensembles de programmes conçus pour créer des machines destinées à traiter des applications déterminées. Elles contiennent beaucoup de données relatives à ces applications qui peuvent être du domaine de la gestion, de l'éducation, de l'informatique, etc. Ces niveaux sont actuellement du domaine de la recherche.

En résumé, on peut dire qu'un ordinateur peut être vu comme une suite de couches, chaque couche englobant toutes les couches précédentes. Une couche représente un certain niveau d'abstraction et comporte divers objets et opérations sur ces objets. Raisonner ainsi nous permet de supprimer tous les détails non significatifs et simplifie donc la compréhension d'un sujet pourtant très complexe.

1.3 L'HISTOIRE DES MACHINES MULTI-COUCHES

Pour mieux comprendre les machines multi-niveaux, nous allons rapidement passer en revue les grandes étapes historiques de leur développement. Les premiers ordinateurs, dans les années quarante, n'avaient que deux niveaux: le niveau machine traditionnel, dans lequel on faisait les programmes, et le niveau physique qui exécutait ces programmes. Les circuits de ce dernier niveau étaient complexes, difficiles à construire et peu fiables.

En 1951, en Grande-Bretagne, M. V. Wilkes eut l'idée de concevoir un ordinateur à trois niveaux qui permettait de simplifier énormément le matériel.

Cette machine devait disposer d'un interpréteur interne destiné à exécuter les programmes du niveau machine traditionnel. Le matériel ne devait plus alors exécuter que des microprogrammes, dont le répertoire d'instructions était très limité, et non pas des programmes en langage machine beaucoup plus complexes. Cela permettait de réduire la complexité du matériel, c'est-à-dire, à l'époque, des tubes à vide et d'en diminuer ainsi le coût tout en en augmentant la fiabilité. Quelques machines à trois niveaux furent construites dans les années cinquante, beaucoup plus durant les années soixante, et à partir des années soixante-dix on peut considérer que la plupart des machines construites avaient une couche microprogrammée pour interpréter la couche machine traditionnelle.

Les assembleurs et les compilateurs furent créés dans les années cinquante, essentiellement pour faciliter la tâche des programmeurs. A cette époque les ordinateurs étaient utilisés en libre-service, donc manipulés directement par les programmeurs. Lorsqu'on voulait faire tourner un programme, il fallait poser une réservation, par exemple Mercredi de 3 heures à 5 heures, et à l'heure dite on pénétrait dans la salle avec son paquet de cartes perforées et son stylo à bille, on poussait dehors le programmeur précédent et on s'asseyait au pupitre de la machine.
Pour exécuter un programme FORTRAN il fallait:

1. Aller dans la réserve des programmes et prendre le paquet de cartes vertes correspondant au compilateur FORTRAN, mettre ce paquet dans le lecteur de cartes pour le faire lire par la machine.

2. Mettre son programme FORTRAN dans le lecteur de cartes et appuyer sur le bouton «continuation».

3. Puis, quand l'ordinateur s'arrêtait il fallait lui faire relire le programme FORTRAN. Si quelques compilateurs, en effet, travaillaient en une seule passe, la plupart nécessitaient deux passes ou plus.

4. On arrivait alors en fin de traduction. La plupart du temps il y avait des erreurs qu'il convenait de corriger avant de recommencer à l'étape 1. Dans le cas contraire, le compilateur perforait un paquet de cartes, traduction en langage machine du programme proposé.

5. Il restait alors au programmeur à mettre dans le lecteur ce paquet de cartes ainsi que ceux qui correspondaient aux sous-programmes de bibliothèque appelés.

6. On arrivait enfin à l'exécution du programme. Très souvent elle s'arrêtait brutalement sans que l'on sache très bien pourquoi. Quelques indications étaient données par les voyants lumineux du panneau de commande et, si l'on avait de la chance, on trouvait l'erreur assez rapidement, on la corrigeait et on recommençait. Dans le cas contraire, il ne restait plus qu'à faire un vidage mémoire sur le papier et à repartir chez soi avec un gros paquet de listages sous le bras.

C'est le genre de choses vues pendant des années dans les centres de calcul. Les programmeurs étaient obligés de savoir comment marchait la machine et il n'était pas rare de voir un ordinateur inactif pendant qu'autour plusieurs personnes s'arrachaient les cheveux pour comprendre pourquoi leur programme donnait des résultats inattendus.

Au début des années soixante on voulut diminuer ce temps perdu en automatisant le travail de l'opérateur, ce qui donna naissance à un programme appelé *système d'exploitation*. Le programmeur devait fournir un certain nombre de cartes de commande associées à son programme et traitées par le système d'exploitation. La figure 1.3 montre le paquet de cartes correspondant à un passage en machine avec un des premiers systèmes d'exploitation: FMS (FORTRAN Monitor System) qui tournait sur IBM 709.

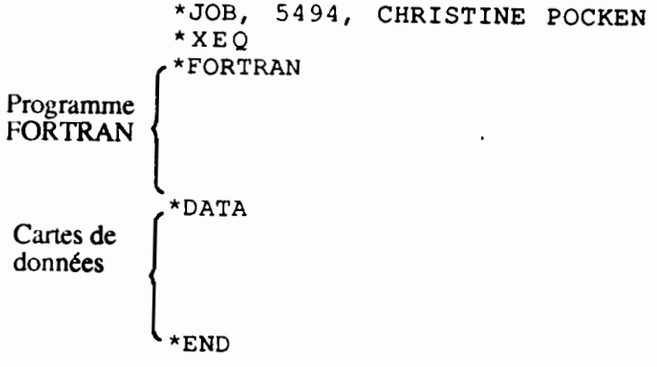

Figure 1.3 Exemple de passage sous FMS

Le système d'exploitation lit la carte *JOB (un astérisque permet d'identifier les cartes de commande) et utilise ses informations pour la comptabilité interne. Il lit ensuite la carte *FORTRAN qui lui indique de charger le compilateur FORTRAN qui se trouve sur une bande magnétique. Lorsque le compilateur a fini son travail, il rend la main au système d'exploitation qui lit la carte *DATA. Cette carte lui donne l'ordre d'exécuter le programme compilé en utilisant les données situées juste après la carte *DATA.

Le système d'exploitation, destiné à l'origine à automatiser le travail de l'exploitation d'un ordinateur, est aussi la première étape du développement d'une nouvelle machine virtuelle. La carte *FORTRAN peut être vue comme une instruction virtuelle «compilation du programme». De la même façon, la carte *DATA est une instruction virtuelle «exécution du programme».

Evidemment un niveau à deux instructions est quelque peu embryonnaire, mais c'était néanmoins un pas dans la bonne direction.

Durant les années soixante, les systèmes d'exploitation devinrent de plus en plus élaborés. De nouvelles instructions, de nouveaux services furent ajoutés jusqu'à ce qu'apparaisse vraiment une nouvelle couche. Quelques instructions de cette couche étaient similaires à celles de la couche machine traditionnelle, mais beaucoup d'autres, en particulier les instructions d'entrées/sorties, étaient complètement différentes. Les nouvelles instructions étaient souvent appelées «macros du système d'exploitation» ou «appels au superviseur» et ces termes sont restés.

Mais il y a eu bien d'autres développements. Les premiers systèmes d'exploitation ne savaient que lire des paquets de cartes et écrire sur des imprimantes. On appelait l'organisation du travail qui en découlait le *traitement par lots*, avec lequel il fallait attendre plusieurs heures après le dépôt de son paquet de cartes pour connaître les résultats du passage.

Ce sont les chercheurs du Dartmouth College, du MIT, et de quelques autres endroits qui ont permis aux programmeurs de communiquer directement avec leur machine. Avec les systèmes d'exploitation qu'ils ont mis au point, des terminaux lourds peuvent êtres connectés à l'ordinateur via des lignes téléphoniques. Les programmeurs, quant à eux, peuvent taper leurs programmes et avoir leurs résultats immédiatement que ce soit à leur bureau

ou chez eux, bref, partout où ils disposent d'un terminal. Ces systèmes d'exploitation étaient et sont toujours appelés *systèmes en temps partagé.*

Ici nous nous intéressons principalement à la partie des systèmes d'exploitation qui interprète les instructions du niveau 3 non présentes au niveau 2. C'est dire que les aspects comme le temps partagé ne seront pas très détaillés.

1.4 MATERIEL ET LOGICIEL

Les programmes écrits dans le langage de l'ordinateur (niveau 1) peuvent être directement exécutés par l'électronique (niveau 0) sans interprétation ni traduction. Cette électronique à laquelle on ajoute la mémoire et les unités d'entrée/sortie, forme ce qu'on appelle le *matériel*. Le matériel, c'est tout ce qui est tangible: circuits intégrés, cartes, câbles, alimentation, mémoire, imprimantes, terminaux...

Au contraire, le *logiciel* est formé des *algorithmes* (instructions détaillées) et de leur représentation informatique: les programmes. Les programmes sont stockés sur des cartes perforées, des bandes magnétiques, des disquettes ou autres mais le logiciel est immatériel puisque c'est l'ensemble des instructions qui forment un programme.

Il existe quelque chose d'intermédiaire entre logiciel et matériel, ce sont les instructions que l'on inclut dans le matériel à sa fabrication. On les utilise lorsque les programmes n'auront pas à être modifiés, ce qui est le cas des jeux électroniques par exemple. Dans beaucoup d'ordinateurs les micro-programmes sont réalisés de cette façon.
Un des thèmes principaux de cet ouvrage est que:

Matériel et Logiciel sont logiquement équivalents

Toute opération effectuée par logiciel peut l'être directement par matériel et toute instruction exécutée par matériel peut être simulée par logiciel.

On décide de réaliser telle opération par logiciel et telle autre par matériel en tenant compte de facteurs comme le coût de réalisation, la vitesse d'exécution nécessaire, la fiabilité requise ou la fréquence des changements attendue. Il n'y a pas de règles simples qui permettent de dire que X doit être réalisée par logiciel et Y par matériel, ce qui explique que, de fait, des choix très différents aient été faits pour les machines que nous utilisons.

Avec les premiers ordinateurs, la distinction entre matériel et logiciel était simple puisque le matériel ne pouvait exécuter que quelques instructions de base comme ADD (addition) ou JUMP (saut). Pour le reste, tout devait être fait par logiciel. Si on avait besoin de multiplier deux nombres, il fallait écrire une procédure de multiplication ou en utiliser une provenant d'une bibliothèque de programmes. Mais bientôt les concepteurs de matériel prirent conscience que la fréquence de certaines opérations justifiait la construction de dispositifs spécialisés capables de les traiter (et de les traiter plus rapidement). C'est ainsi qu'un certain nombre d'opérations ont changé de niveau: alors qu'elles étaient programmées explicitement au niveau machine traditionnel, elles sont descendues au niveau matériel.

Avec l'apparition de la multiprogrammation et des ordinateurs multi-niveaux l'inverse a pu également survenir. Alors que sur les premiers

ordinateurs l'instruction ADD était traitée directement par le matériel, on a pu voir sur les ordinateurs microprogrammés l'instruction ADD du niveau machine traditionnel interprétée par un microprogramme tournant au niveau inférieur et exécutant l'instruction par petites étapes: chargement de l'instruction, décodage de son type, localisation des données à additionner, chargement des données depuis la mémoire, addition proprement dite et stockage du résultat en mémoire. C'est un exemple de fonction qui a migré vers le haut, du niveau matériel au microprogramme. Une fois de plus nous redirons qu'il n'y a pas de règle qui permette de savoir ce qui doit être réalisé par logiciel et ce qui doit l'être par matériel.

Mais les concepteurs de machines doivent décider de ce qu'ils mettront à chaque niveau. C'est la généralisation du problème que nous venons de voir (affectation matériel-logiciel). A titre d'information voici ce qui était autrefois explicitement programmé au niveau machine traditionnel et qui est maintenant pris en charge soit par le matériel soit par des microprogrammes:

1. Les instructions de multiplication et de division sur des entiers.
2. Les instructions de calcul en virgule flottante (cf. Annexe B).
3. Les instructions de calcul en double précision.
4. Les instructions d'appel de procédure et de retour au programme appelant.
5. L'incrémentation (additionner 1 à une variable).
6. Les instructions de manipulation de chaînes de caractères.
7. L'optimisation de la manipulation de tableaux (adressage indirect ou indexé).
8. La possibilité de changer d'emplacement mémoire un programme.
9. Les horloges.
10. Les interruptions qui préviennent l'ordinateur de la fin d'une opération d'entrée/sortie.
11. La capacité d'arrêter l'exécution d'un programme pour passer la main à un autre programme en quelques instructions (commutation).

On voit bien que la frontière entre logiciel et matériel est très mouvante. Ce qui est logiciel aujourd'hui sera matériel demain et réciproquement. De plus les frontières entre couches sont elles-mêmes assez floues. Du point de vue du programmeur peu importe la façon dont est réellement exécutée une instruction. On peut programmer une multiplication au niveau machine traditionnel sans se soucier de savoir si elle sera directement traitée par le matériel ou pas.

Le fait qu'un programmeur n'ait pas à se soucier de la façon dont le niveau qu'il utilise est réalisé plaide pour une conception de machine structurée. Si l'on a dit que chaque couche était une machine virtuelle, c'est parce que le programmeur voit cette couche comme une machine physique même si ce n'est pas exact. Lorsqu'une machine est structurée en couches, les programmeurs qui travaillent au niveau n n'ont pas à se soucier des détails de réalisation des couches inférieures. Cette structuration simplifie également, et de manière énorme, la production de machines (virtuelles) très complexes.

1.5 LES PROCESSUS

Un des concepts fondamentaux de l'informatique est celui de *processus*. Un processus (que l'on appelle parfois un *processus séquentiel*) c'est, en gros, un programme en cours d'exécution. C'est une entité active capable de générer des événements. Un processus (c'est-à-dire un programme actif) peut tracer des caractères chinois sur une table traçante ou jouer aux échecs contre un adversaire humain assis devant un terminal. Il peut aussi suivre le bon fonctionnement d'une colonne à distiller ou piloter la trajectoire de robots. Un programme est, au contraire du processus, une entité passive. Un programme posé sur un bureau ne fera rien par lui-même.

On peut tenter une analogie entre un processus et un animal vivant. Tous deux peuvent changer, dans une certaine mesure, leur environnement. En revanche, un programme, comme un animal mort ou une image de cet animal, ne peut pas générer d'événements. Il ne faudrait surtout pas en conclure qu'un processus est vivant mais bien voir qu'il y a autant de différence entre un processus et un programme qu'entre un animal vivant et un animal mort.

```
program simple;
label 1,2,3,4,5,6
var i,j : integer;
begin
    1: i := 0;
    2: j := 1;
    3: i := i+2;
    4: j := i+j;
    5: i := i*j + 4;
    6: j := i*i
end.
```

Figure 1.4 Exemple de programme Pascal

A tout instant un processus est dans un certain état qui rend compte de son avancement. Cet état contient toutes les informations nécessaires à un arrêt momentané du processus et à son redémarrage un peu plus tard. Parmi ces informations on trouve au moins:

1. L'identification du programme.
2. L'identification de la prochaine instruction à exécuter.
3. La valeur des variables et des données du programme.
4. L'état de toutes les unités d'entrée/sortie utilisées.

Au fur et à mesure que le processus progresse son état change. Les différentes instructions deviennent tour à tour «la prochaine instruction à exécuter», et les variables prennent différentes valeurs. Bien qu'il soit possible à un processus de modifier son programme pendant l'exécution, on doit considérer que ceci relève d'une mauvaise programmation et nous supposerons donc que le programme n'est pas modifié par le processus. Il est souvent commode de grouper dans un *mot* (ou un *vecteur*) *d'état* tout ce qui peut changer dans l'état d'un processus. On peut voir alors le processus comme étant formé d'une part

d'un programme (qui est figé) et d'autre part d'un mot d'état (qui change au cours du temps). Le programme, quant à lui, contient des instructions qui changent le mot d'état.

A titre d'exemple, considérons le programme de la figure 1.4. L'état de ce programme pendant son exécution peut être caractérisé par trois informations: la valeur de *i* et de *j*, et la prochaine instruction à exécuter. L'ensemble des différents mots d'état du processus exécutant ce programme est donné en figure 1.5.

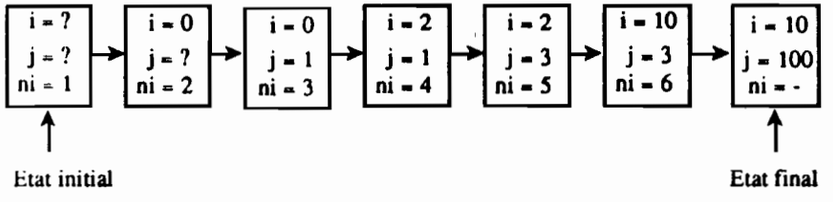

Figure 1.5 Evolution du mot d'état, `ni` est le numéro de l'étiquette qui précède l'instruction à exécuter

Un processus traverse une suite d'états ordonnés dans le temps. Pour qu'un programme s'exécute, et donc pour qu'il devienne un processus, il faut qu'il y ait un ordinateur ou un *processeur* pour l'exécuter. Le rôle du processeur est de faire passer un processus d'un état à l'état suivant. En d'autres termes, tout ce que fait un processeur c'est de faire changer le mot d'état d'un processus. Un processus a deux propriétés importantes:

1. L'effet d'un processus est indépendant de sa vitesse d'exécution.

2. Si un processus s'exécute de nouveau avec les mêmes données, il passe par les mêmes états et donne le même résultat.

Ces propriétés soulignent la nature séquentielle d'un processus. Un processus séquentiel se définit par ce qu'il fait et non par la vitesse à laquelle il le fait.

Le processeur qui fait passer un processus d'un état à l'état suivant n'est pas obligatoirement du matériel. Ce peut être un autre processus. Lorsqu'un processus P1 fait avancer un processus P2 en changeant son mot d'état, P1 interprète le programme de P2; ceci signifie que P1 est un interpréteur.

Le concept de processus permet ainsi de donner une autre définition d'un interpréteur: un interpréteur est un programme qui, lorsqu'il est exécuté, fait évoluer le mot d'état d'un processus P, en suivant la suite des états nécessaires à l'exécution du programme de P. C'est exactement ce que ferait un processeur matériel. La figure 1.6 montre un ordinateur à trois niveaux dans lequel le niveau 3 est supporté par un interpréteur de niveau 2, tandis que le niveau 2 est supporté par un interpréteur de niveau 1. A chaque niveau se trouvent un programme et un mot (ou un vecteur) d'état. Le matériel charge les instructions de P1 et change SV1 après chaque instruction de façon à exécuter P1. Ce processus de niveau 1 charge les instructions de P2 et change SV2. Donc P1 interprète P2.

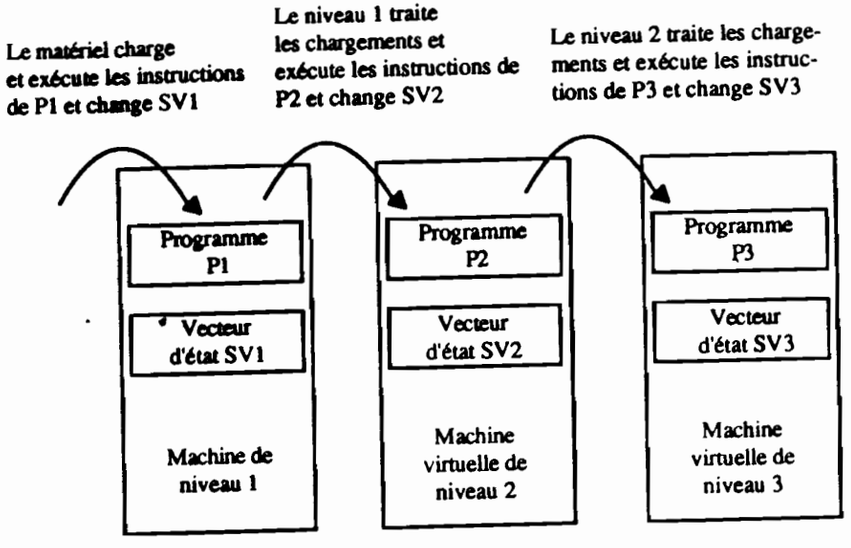

Figure 1.6 Système à trois niveaux

De la même façon que P1 interprète P2, P2 interprète P3. Le programme P3 est soit un autre interpréteur soit un programme d'application qui résoud un problème posé par un utilisateur. Cette structure à trois niveaux comporte trois programmes et trois vecteurs (ou mots) d'état. A chaque niveau on trouve donc une «prochaine instruction à exécuter».

1.6 PLAN DE L'OUVRAGE

Cet ouvrage concerne les machines multi-niveaux (à peu près tous les ordinateurs modernes) et leur organisation. Nous étudierons de manière approfondie cinq niveaux: le niveau physique, le niveau micro-programmation, le niveau machine traditionnel, le niveau système d'exploitation et le niveau langage d'assemblage. Nous verrons notamment:

1. La conception générale d'un niveau donné.
2. Le genre d'instructions disponibles à ce niveau.
3. Le genre de données manipulées.
4. Les mécanismes disponibles pour modifier les commandes.
5. L'organisation de la mémoire et son adressage.
6. La relation entre jeu d'instructions et organisation de mémoire.
7. La méthode de réalisation du niveau.

Ces divers aspects sont souvent regroupés sous le vocable *structure des ordinateurs* ou encore *architecture des ordinateurs*. Ces deux termes sont synonymes.

Comme ce qui nous intéresse ce sont d'abord les idées de base et non tel ou tel détail de réalisation, beaucoup d'exemples seront simplifiés. Pour montrer, en effet, comment les idées présentées dans ce livre sont appliquées dans la réalité, nous présenterons quatre ordinateurs très connus: l'IBM 370, le PDP 11 de Digital Equipment, le Motorola 68000 et le Zilog Z80. Ils ont été choisis pour plusieurs raisons: d'abord ils sont largement répandus et nous sommes sûrs que tout lecteur de cet ouvrage a accès à au moins un d'entre eux. De plus chacun d'entre eux a sa propre architecture, ce qui nous permettra de faire des comparaisons et de mettre l'accent sur les choix, les compromis ou les décisions arbitraires des concepteurs sachant que, comme nous l'avons déjà dit, il n'y a pas de méthode de conception unique. Ceci devrait vous encourager à examiner d'un œil critique tous les ordinateurs que vous rencontrerez pour essayer de comprendre les raisons qui ont présidé aux choix de conception.

Il doit être maintenant clair que ce livre n'a pas pour but de vous apprendre à programmer sur IBM 370, PDP-11, M 68000 ou Z80. Ces machines ne sont là que pour illustrer notre propos et nous ne prétendons pas être complet à leur sujet. Si vous désirez tout connaître de ces machines, consultez les publications de leurs constructeurs.

Le chapitre 2 est une introduction aux composants de base d'un ordinateur (processeur, mémoire, dispositifs d'entrée/sortie) et aux techniques utilisées pour leur interconnexion ainsi que pour la transmission de l'information entre ces composants. Ceci concerne plus d'un niveau et souvent même tous les niveaux d'une machine et nous permettra de faire un survol de l'architecture des ordinateurs.

Les chapitres 3, 4, 5, 6 et 7 sont consacrés chacun à un des niveaux de la figure 1.2. Nous ferons une analyse ascendante parce que c'est souvent comme cela que les machines sont construites: la conception du niveau k est en grande partie déterminée par les propriétés du niveau k-1. De plus, il est pédagogiquement meilleur d'aller du niveau le plus simple au niveau le plus complexe que l'inverse.

Le chapitre 3 concerne la couche physique, relative aux circuits logiques, le niveau du matériel proprement dit. Nous y introduisons l'algèbre de Boole en tant qu'outil d'analyse des circuits. Ce chapitre se termine par l'exposé de la conception d'un micro-ordinateur simple mais complet.

Le chapitre 4 introduit les concepts généraux de la microprogrammation, l'architecture de la couche microprogrammée et les relations qu'elle a avec la couche machine traditionnelle. Une partie importante de ce chapitre est consacrée à l'analyse détaillée d'un exemple. On y trouve aussi un certain nombre d'informations sur la couche microprogrammée de quelques machines du marché.

Le chapitre 5 décrit la couche machine traditionnelle, celle dont on dit qu'elle correspond au langage machine. Nous verrons cette couche d'abord de manière abstraite puis nous analyserons quelques exemples.

Le chapitre 6 s'attache à présenter les instructions, l'organisation de la mémoire et les mécanismes de commande du niveau système d'exploitation.

Le chapitre 7 présente le niveau langage d'assemblage. Comme ce niveau, à la différence des niveaux inférieurs, est souvent réalisé par une traduction plutôt qu'une interprétation, nous mettrons l'accent sur le processus même de traduction et non sur les détails de tel ou tel langage d'assemblage. Les

notions associées de macros (une technique de traduction) et d'édition de liens (la dernière phase d'une traduction) seront présentées à cette occasion.

Le chapitre 8 permet d'avoir une vue plus globale des machines multi-niveaux et présente quelques aspects de ces machines prises comme un tout.

Le chapitre 9 comporte une bibliographie ainsi que la liste des articles évoqués dans cet ouvrage.

Les annexes présentent une brève introduction au calcul en précision finie et aux nombres en virgule flottante, ainsi qu'un lexique des quelques termes techniques dont l'utilisation est recommandée.

1.7 EXERCICES

1.1 Expliquer chacun des termes suivants:

a. Traducteur
b. Interpréteur
c. Machine virtuelle.

1.2 Quelle différence y a-t-il entre interprétation et traduction?

1.3 Est-ce que la suite des états à travers laquelle passe un processus peut être modifiée par les données d'entrée? Expliquer.

1.4 Est-il possible à un compilateur de générer une sortie pour niveau 1 au lieu de niveau 2? Discuter le pour et le contre.

1.5 Pouvez-vous imaginer un ordinateur dans lequel le niveau physique relatif aux composants logiques ne soit pas le plus bas niveau? Expliquez.

1.6 Donnez la liste de tous les états que prendra le processus exécutant le programme suivant, en supposant que chaque ligne est une instruction unique et indivisible.

```
program regime;
label 1,2,3,4,5;
var poids, raisonnable: integer;
begin
   1: raisonnable := 80;
   2: poids := 100;
   3: while poids > raisonnable do
   4: poids := poids - 5
   5:
end.
```

1.7 Si les programmes P1 et P2 de la figure 1.6 sont identiques, quelle est la relation entre les langages machine des machines virtuelles 2 et 3?

1.8 Soit un ordinateur dont les interpréteurs de niveau 1, 2 et 3 sont identiques. Il faut à un interpréteur n instructions pour charger, décoder et exécuter une instruction. Sachant qu'une instruction au niveau 1 prend k nano-secondes pour être exécutée, combien de temps faudra-t-il pour exécuter une instruction de niveau 2, 3 ou 4?

1.9 Soit un ordinateur dans lequel toutes les couches sont différentes. Chaque couche a des instructions m fois plus puissantes que celles de la couche immédiatement inférieure, c'est-à-dire qu'une instruction de niveau r fait le même travail que m instructions de niveau r-1. Si un programme de niveau 1 nécessite k secondes pour s'exécuter, combien de temps prendraient des programmes équivalents de niveau 2, 3 et 4 en supposant qu'il faut n instructions de niveau r pour interpréter une instruction de niveau r+1?

1.10 Certaines instructions du niveau système d'exploitation étant identiques à celles du niveau machine traditionnel, ces instructions sont traitées directement par le microprogramme. Pourquoi? (Pensez à la réponse du problème précédent.)

1.11 En quel sens matériel et logiciel sont-ils équivalents?

1.12 Combien de niveaux de la figure 1.2 a votre ordinateur? Quel est le niveau décrit par le manuel de référence ?