Principes d'architecture externe des ordinateurs

F. Anceau
Département d'Informatique

Septembre 1998

1 LES NOTIONS D'ARCHITECTURE EXTERNE ET INTERNE DES ORDINATEURS

L'architecture externe des ordinateurs concerne essentiellement la vision qu'ils offrent à leurs programmeurs de bas niveau (en assembleur). Il ne faut pas confondre ce point de vue avec l'architecture interne qui correspond à l'organisation physique de ces machines.

Il est toutefois intéressant de remarquer que les principes de base de l'architecture externe reposent sur des considérations (para)technologiques issues de l'époque où ces deux architectures étaient encore fusionnées. L'évolution des architectures externes fait qu'elles correspondent à la description de machines hypothétiques de moins en moins réalisables.

L'architecture externe des machines modernes n'a plus grand chose de commun avec leur architecture interne.

Nous pouvons dire que l'architecture externe d'une machine concerne son utilisabilité, tandis que son architecture interne concerne son coût et sa performance.

2 L'INFLUENCE DE LA TECHNOLOGIE

Un ordinateur est un dispositif destiné à effectuer des opérations logico-arithmétiques sur des données.
Pour faire ce travail, il a besoin de disposer de deux fonctions de base:

- Le stockage des données dans des cellules binaires capables de les retenir.
- La possibilité de transformer et de combiner des informations

La technologie micro-électronique nous fournit ces deux fonctions:

- Les éléments de mémorisation sont de petite taille (1transistor pour les grosses mémoires de quelques dizaines de méga bits et 6 transistors pour les bascules isolées) et très denses. Ces éléments ne font strictement que mémoriser de l'information binaire. Ils doivent donc être sélectionnés pour les écrire et pour les lire. Cette sélection s'opère par d'autres organes (comme par exemple des décodeurs). Les éléments de mémorisation ont une vitesse d'accès qui dépend de leur taille (typiquement 100ns pour les grosses mémoires et 0,5ns pour les bascules).
- Les fonctions de traitement nécessitent plus de transistors (20 à 50 pour combiner deux bits), mais ils sont très rapides (< 1ns).

3 LA BOUCLE DE CALCUL

Le coût et la rapidité des opérateurs ainsi que la densité des dispositifs de mémorisation nous conduit à séparer ces deux fonctions. L'information est stockée dans des mémoires et traitée par des opérateurs. La rapidité de ces derniers dépasse largement les besoins humains ou électromécaniques qui utilisent ces machines. Elle nous permet de décomposer le travail à réaliser en une multitude d'opérations simples réalisées successivement et qui peuvent être faites par des opérateurs simples.

Copyright © 1997, 1998. La copie de tout ou partie de ce document est autorisée pour une utilisation personnelle, de recherche ou d'enseignement, avec mention explicite de l'auteur. Toute autre utilisation nécessite une autorisation écrite de celui-ci.

Le cœur d'un processeur se comporte donc comme une boucle dans laquelle les informations sont extraites des éléments de mémorisation, transformés et combinés par des opérateurs puis réécrites dans les éléments de mémorisation.

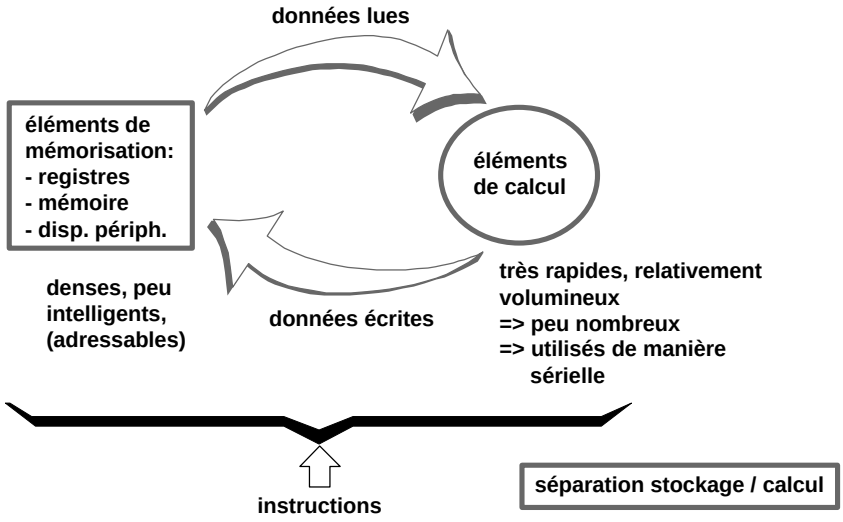

Figure 1 La boucle de calcul.

Nous appellerons cette structure une boucle de calcul (ou un "tourbillon de calcul").

Un compromis entre la performance et le coût des machines conduit à traiter en parallèle les différents bits qui constituent les mots manipulés par la boucle de calcul (de 4, 8, 16, 32 ou 64 bits).

Il faut bien remarquer que d'autres techniques de traitement de l'information pourraient être imaginées, comme par exemple, le fait de mélanger le traitement et le stockage de l'information (comme par exemple dans les machines associatives). Le fait de s'écarter des conséquences logiques issues des propriétés de la technologie sous-jacente rend ces machines non-optimales. La duplication du traitement est très coûteuse et ne se justifie pas toujours en terme de parallélisme introduit. Peut être que l'utilisation d'une toute autre technologie pourrait changer cette chaîne de conséquences et orienter l'architecture des machines à traiter l'information vers d'autres voies.

3 LA HIERARCHIE DES MEMOIRES

Le fait qu'il serait souhaitable de disposer de dispositifs de mémorisation à la fois vastes et rapides conduit à les organiser de manière hiérarchique. Le sommet de la hiérarchie est occupé par des éléments de mémorisation (les registres) aussi rapides que les opérateurs, mais de petite capacité, tandis que plus on descend dans la hiérarchie plus on trouve des dispositifs de mémorisation de taille croissante et de vitesse décroissante.

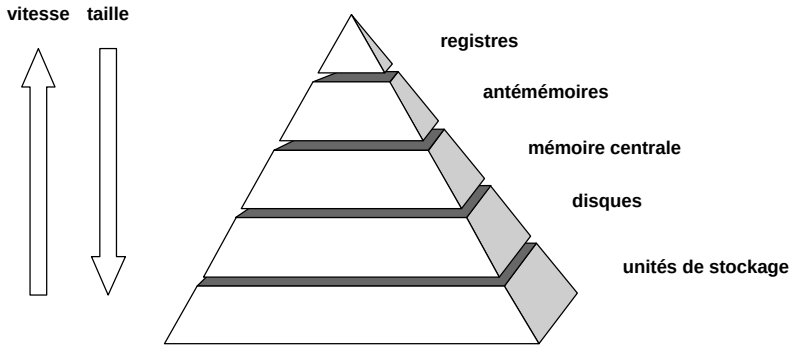

Figure 2 La hiérarchie des mémoires

Le niveau supérieur de la hiérarchie est constitué des registres qui sont des montages logiques capables de retenir de l'information et de la délivrer aux opérateurs. Les registres se situent dans la boucle de calcul. Ils sont généralement très rapides d'accès et de taille très réduite (quelques dizaines de mots de 4, 8, 16, 32 ou 64 bits).

Les échanges entre ces deux niveaux sont explicitement commandés par l'utilisateur de la machine grâce à des instructions ad-hoc.

Le niveau visible juste en dessous est constitué par la mémoire centrale. Celle-ci contient un très grand nombre de positions (plusieurs dizaines de millions de mots) mais elle n'est pas très rapide. Son temps d'accès correspond à celui de plusieurs dizaines de tours de la boucle de calcul.

Les éléments inférieurs contiennent toute l'information de manière exhaustive tandis que plus on s'élève dans la hiérarchie plus les éléments de mémorisation ne contiennent que l'information pertinente à un instant donné. Au cours du traitement, l'information pertinente doit donc migrer vers le sommet de la hiérarchie tandis que l'information modifiée doit redescendre pour maintenir à jour le niveau inférieur. Le cœur de l'ordinateur contient généralement les deux niveaux supérieurs de mémorisation visibles par le programmeur (les registres et la mémoire principale). Il faut toutefois remarquer que des niveaux intermédiaires existent entre les registres et la mémoire centrale pour limiter les effets de la grande différence de caractéristiques qui existe entre ces deux niveaux.

3 LES TENDANCES INFLATIONISTES

L'évolution des ordinateurs est marquée par de fortes tendances inflationnistes. La quantité de mémoire nécessaire pour une application donnée ne fait que croître compte tenu de l'ajout de fonctions d'ergonomie et de multiples détails. De même la puissance de calcul nécessaire à une application ne cesse de croître par un phénomène de "complexification permanente". Cette tendance ne ralentit pas et il est nécessaire d'en tenir compte dans la conception de nouvelles machines toujours de plus en plus puissantes et de mémoires de plus en plus vastes. Les mécanismes d'adressage, pourtant largement dimensionnés à l'origine, finissent toujours par saturer, ce qui demande des bricolages, voir un abandon de ce modèle de machine.

4 LES INSTRUCTIONS

La désignation des éléments de mémorisation qui doivent être lus et écrits à chaque cycle, ainsi que les opérations à effectuer est donnée par les instructions. Depuis Von Neumann les instructions sont rangées en mémoire centrale. Ce sont des mots mémoire dont le profil décrit les opérations à effectuer dans un ou plusieurs cycles de la machine. La succession des instructions qui décrivent un calcul complexe constitue un programme. Il faut donc un mécanisme particulier pour les lire les unes à la suite des autres et pour les décoder.

Les processeurs peuvent se classer en deux catégories: les RISC et les CISC. Dans les machines RISC les opérateurs ne dialoguent qu'avec les registres qui sont relativement nombreux. Des instructions particulières permettent de charger et de décharger les registres dans la mémoire. Dans les machines CISC les opérateurs peuvent lire et écrire leurs données dans la mémoire ce qui complique le diagramme.

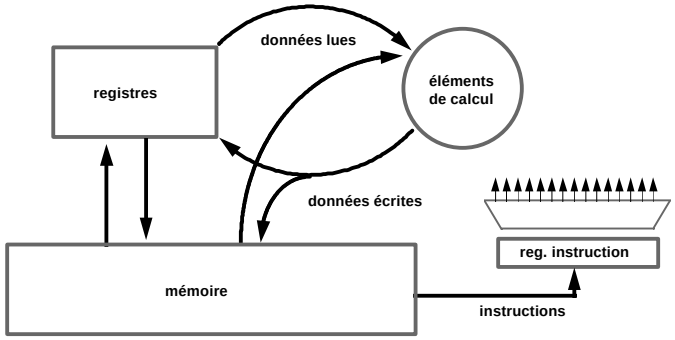

Figure 3 Les principaux flux dans un ordinateur.

4.1 Le codage des instructions

Les instructions sont décrites par un nombre fixe ou variable de mots mémoire de 4, 8, 16, 32 ou 64 bits. Ces mots sont découpés en champs qui codent les différentes informations nécessaires à l'exécution de l'instruction (opération à effectuer, désignation des opérandes, place du résultat, etc...).

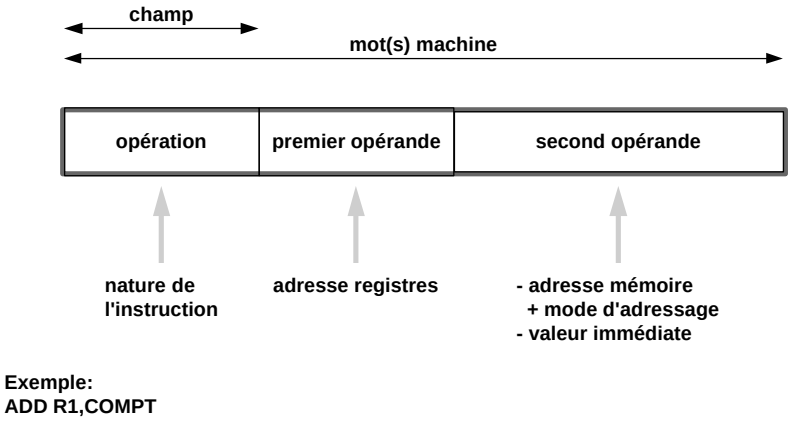

Figure 4 Le format d'une instruction.

De manière à simplifier le décodage, les différentes instructions s'appuient, dans la mesure du possible, sur un format commun. Les machines RISC ont en général des instructions de format et de longueur fixes. Les machines CISC ont des instructions de longueur variable qui s'appuient sur un ensemble de formats plus complexe. Le fait que la taille des instructions soit limitée réduit les possibilités de codage. Par exemple, beaucoup d'opérations nécessitent deux opérandes et génèrent un résultat (exemples: addition, soustraction, ..., et, ou,.....). Le manque de place conduit souvent à fusionner la désignation de l'un de ces opérandes et celle du résultat.

exemple:

add	a,b
signifie	a <= a + b

Ces mêmes contraintes font que les instructions n'ont, en général, qu'un seul opérande faisant référence à la mémoire.

4.2 La désignation des opérandes

Les variables les plus fréquemment utilisées sont amenées dans les registres. Comme ceux-ci ne sont pas très nombreux (8 à 32, exceptionnellement 256), ils sont désignés explicitement dans les instructions par leur numéro.

exemple: R17 signifie le registre numéro 17

La mémoire qui contient des dizaines de millions de mots ne peut être accédée de la sorte. Ceci nécessiterait trop de bits dans les instructions. En fait, une autre raison milite pour utiliser des techniques d'adressage plus élaborées. A un instant donné, la mémoire contient plusieurs programmes et plusieurs zones de données. L'optimisation du rangement de ces différents modules en mémoire nécessite que le système d'exploitation de la machine puisse choisir librement les adresses de rangement. Ces adresses doivent d'ailleurs pouvoir changer d'une exécution à l'autre, voire même pendant une exécution. Le programmeur doit donc pouvoir ignorer l'adresse à laquelle son programme et ses données seront chargées.

La solution généralement adoptée consiste à travailler avec des adresses relatives. Toutes les adresses à l'intérieur d'un module de programme ou de données sont données relativement à son début. Lors du chargement en mémoire, le système d'exploitation charge des registres spéciaux appelés registres de *segment* avec les adresses de début de ces modules. Lors de l'adressage d'un élément, son adresse relative sera automatiquement ajoutée à celle contenue dans le registre de segment associé au module dans lequel il se trouve.

Sur certaines machines, les zones manipulées (code, données,...) sont décrites avec un peu plus de détail. En particulier la machine prend en compte leur longueur, leur nature (code, données, ...) et leur droits d'accès (exécutables, modifiable, ...) pour effectuer des vérifications. On parle alors de *segments* .

4.3 Les registres de base et d'index

Le fait de manipuler des structures de données variées (table, listes chaînées, etc....) a rendu nécessaire le développement de techniques d'adressage plus complexes pour manipuler les adresses relatives.

La manipulation commode des tables a rendu nécessaire le fait de pouvoir désigner une table, ou simplement une donnée, sans en connaître explicitement l'emplacement. Ceci a été obtenu en utilisant un registre, dit de *base*, pour indiquer l'emplacement de la structure de donnée dans l'espace d'adressage du programme.

adresse relative <= base

Il est aussi nécessaire de faire progresser les adresses pour parcourir ces tables. Ce type de parcours a été obtenu en ajoutant le contenu d'un registre (appelé *index*) à l'adresse inscrite dans l'instruction.

adresse relative <= adresse instruction + index (+ base)

L'incrémentation de l'index est réalisé soit automatiquement à chaque accès, soit par une autre instruction de modification explicite du contenu du registre index. La ressemblance entre les mécanismes de base et d'indexation fait qu'ils sont quelquefois confondus sur certaines machines. Toutefois, le besoin de pouvoir désigner des structures de données dont l'adresse est rangée en mémoire et de pouvoir les indexer suggère de conserver distincts ces deux mécanismes. Les registres de base et d'index peuvent être des registres spéciaux, mais dans la plupart des machines, il s'agit de registres généraux. Dans ce cas on parlera d'adressage *indirect* et *indexé* .

L'utilisation d'autres structures de données plus complexes comme les listes demande aussi au programme de manipuler des adresses relatives. Celles-ci seront rangées dans les registres de base et utilisées lors de la construction des adresses.

Les programmes sont générés par d'autres programmes (compilateurs, assembleurs, etc....) qui les considèrent comme des données.

Quelquefois, un programme se modifie lui-même pendant son exécution. Cette pratique était très utilisée au début de l'informatique pour réaliser des tests conditionnels. Actuellement elle n'est utilisée que très rarement pour effectuer de l'optimisation dynamique (cas du byte-code du langage Java).

4.4 Les valeurs immédiates

Les constantes nécessaires aux calculs sont souvent incluses dans le code lui-même sous la forme de *valeurs immédiates* qui replacent un opérande dans l'instruction. Pour des questions d'optimisation la taille de ces valeurs immédiates dépend de leur valeur. Par exemple, les petites valeurs immédiates, assez fréquentes, sont codées sur 8 bits tandis que les autres sont codées sur 32 bits.

4.5 Les instructions de ruptures de séquence

Nous avons vu que les instructions sont rangées les unes à la suite des autres en mémoire. L'ordinateur dispose d'un registre spécial appelé *compteur ordinal* pour repérer l'instruction courante. Le contenu de ce registre est incrémenté au cours de l'exécution de chaque instruction. Ce compteur peut, soit contenir la position de l'instruction courante relativement au segment de code en exécution, soit l'adresse physique de cette instruction pour éviter d'exécuter une addition au début de l'exécution de chaque instruction.

Les *ruptures de séquence* sont des instructions particulières dont le but est de faire poursuivre l'exécution des instructions en un autre endroit de la mémoire. Elles peuvent être conditionnelles ou non, c'est à dire que le branchement ne s'effectue que sur la présence de certaines conditions représentées par certains bits du *registre de condition*. Les instructions de rupture de séquence peuvent adresser le code comme des données (avec toutefois une richesse de modes d'adressage réduite) ou l'adresser relativement à leur emplacement. Dans ce cas, l'instruction de branchement contient un déplacement signé qui est ajouté au compteur ordinal pour réaliser des sauts en avant ou en arrière.

L'adressage "classique" du code consiste, le plus souvent, à fournir un déplacement dans le segment de code, ou à fournir un nouveau couple (segment, déplacement) désignant un point d'entrée dans un nouveau segment de code. L'adresse de branchement dans le code peut quelquefois être indexée pour permettre la réalisation de tableaux de branchements correspondant à des aiguillages multiples.

exemple:

```
        B       aig[Rx]

aig:    JMP  dest1
        JMP  dest2
        .....
```

4.6 Les instructions conditionnelles

Dans l'architecture EPIC (Intel, HP) une partie de la fonction des branchements conditionnels est réalisée par un mécanisme d'exécution conditionnelle des instructions. Celles-ci sont précédées de l'analyse de bits de conditions qui valident ou interdisent leur exécution (instructions gardées).

5 LES SOUS-PROGRAMMES

La notion de sous-programmes permet de factoriser le code et de développer des hiérarchies d'outils logiciels. La réalisation de ce mécanisme est facilitée par l'utilisation d'une *pile*. Celle-ci est constituée d'un tableau et d'un pointeur de remplissage.

L'appel d'un sous-programme consiste à empiler l'adresse de la prochaine instruction à exécuter et à se débrancher sur le début du sous-programme. Une instruction particulière permet de revenir du sous-programme en exploitant l'adresse préalablement empilée. Sur les anciennes machines, l'appel de sous-programme provoquait la sauvegarde du compteur ordinal dans un registre général que l'on pouvait empiler si on le souhaitait. Le retour se faisait simplement grâce à une instruction de branchement indirect sur le contenu de ce registre.

En plus des appels de sous-programmes, la pile peut être exploitée séparément grâce à deux instructions:

- *empiler* qui consiste à ajouter un élément dans la pile en déplaçant le pointeur vers la prochaine position libre
- *dépiler* qui consiste à extraire un élément de la pile en ramenant le pointeur.

Les registres qui sont utilisés par le sous-programme peuvent être préservés dans la pile par des opérations d'empilement pour ne pas perturber le fonctionnement de l'appelant. Ils sont restitués par des opérations de dépilement lorsque le sous-programme se termine.

Les paramètres du sous-programme peuvent lui être passés via la pile. Il y sont rangés avant l'appel et exploités par le sous-programme via des accès indexés qui utilisent le pointeur de pile comme un registre de base.

6 LES INTERRUPTIONS

6.1 Les déroutements

Les *déroutements* sont des alarmes dont dispose l'ordinateur pour informer les programmes des conditions anormales qu'il rencontre lors de leur exécution. Celles-ci sont traitées comme des appels impromptus de sous-programme consécutifs à la détections de ces conditions anormales.

6.2 Les appels externes

Les *interruptions externes* consistent à déclencher l'exécution de certains programmes sur l'occurrence d'événements externes. Elles consistent à commuter l'exécution de la machine vers ces programmes lors de l'apparition de ces événements. Les programmes appelés n'ont, en général, rien à voir avec le programme qui était précédemment en exécution. Il s'agit, en fait, d'un mécanisme de commutation de tâches.

6.3 Les appels systèmes

La liaison entre un programme et les fonctions système qu'il utilise doit se faire de la manière la plus indépendante possible de la structure du système. En effet celle-ci varie d'une version à l'autre. Ces appels doivent aussi constituer une certaine barrière de protection pour éviter qu'un programme utilisateur ne puisse lire des données systèmes ou provoquer l'exécution de n'importe laquelle de ses fonctions.

6.4 Le mécanisme d'interruption

Pour une raison historique, et de commodité, les trois mécanismes précédents sont réalisés avec un mécanisme unique dit d'*interruption*. Ce mécanisme correspond à des appels impromptu de sous-programmes lors de l'occurrence de déroutements, d'appels externes et d'appels systèmes. Cette confusion est acceptable lorsque l'on travaille avec un monoprocesseur, mais elle devient plus problématique en multiprocesseur puisque, dans ce cas, les interruptions externes, qui correspondent à des commutations de tâche sont réalisés par des appels de sous-programme.

Les déroutements et les appels systèmes sont des événements directement liés au déroulement du programme courant tandis que les appels externes n'ont aucun rapport avec lui et sont complètement asynchrones. De manière à éviter qu'elles ne viennent perturber une phase critique du traitement, les appels externes peuvent être *masqués*, en positionnant un bit dans le registre d'état de la machine. L'effet des appels externes est retardé jusqu'à ce le masque soit levé.

De manière à rapidement identifier la source de l'interruption, l'appel de sous-programme se fait généralement indirectement en indexant un vecteur d'adresses par le numéro de l'interruption. Ce mécanisme permet d'ailleurs au logiciel de changer le traitement associé à une interruption en substituant l'adresse correspondante du vecteur.

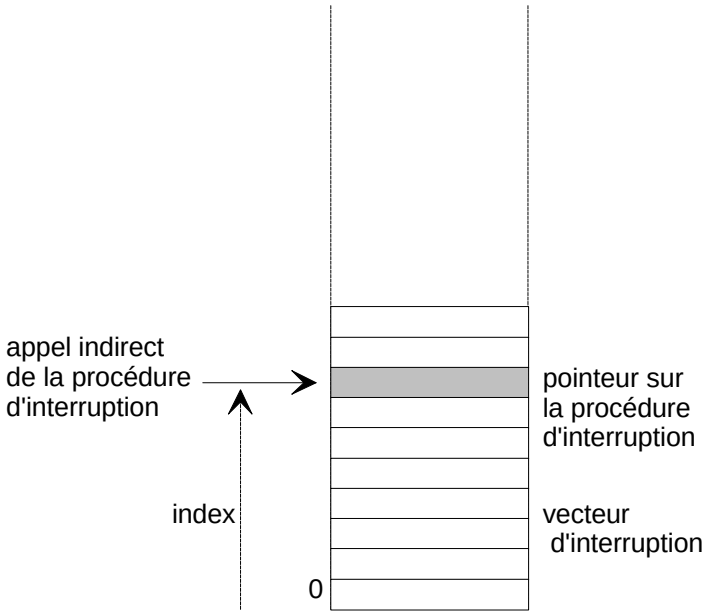

Figure 5 La pile d'interruption.

Lors de l'occurrence d'une interruption, la machine préserve l'adresse de retour dans la pile comme dans le cas d'un sous-programme, mais aussi le registre d'état qui contient le ou les bits de masquage des appels externes. Le sous-programme d'interruption est ensuite lancé dans un mode dans lequel tous les appels externes sont bloqués. L'instruction de retour d'interruption rétablit le registre d'état à sa valeur initiale et effectue le retour du sous-programme d'interruption.

6.5 Les verrous matériels

Le masquage des interruptions ne permet pas de résoudre la totalité des problèmes de synchronisation entre les programmes qui s'exécutent en parallélisme (dans le cas d'un multiprocesseur). En particulier la protection d'une section critique demande un mécanisme spécifique. Celui-ci est souvent réalisé en offrant la possibilité de lier de manière indivisible une lecture et une écriture en mémoire (en bloquant l'allocateur). Ce mécanisme permet de réaliser facilement des opérations de *test&set* . Une autre possibilité consiste à offrir un sémaphore matériel câblé entre les processeurs.

7 LES MACHINES COMPLEXES

Pour des raisons technologiques et de mode, la complexité des machines informatiques a connu des hauts et des bas. A la fin des années 60 et au début des années 70, la disponibilité de mémoires de microprogrammation de grande capacité, et tout simplement la mode, a amené la conception de machines très complexes. L'objectif était de réduire la complexité du logiciel en offrant des instructions de niveau sémantique élevé et d'accélérer les calculs en faisant directement exécuter ces fonctions de haut niveau par le matériel (plus exactement en les microprogrammant). De nombreuses machines furent conçues suivant ces principes. Nous pouvons les classer en deux grandes catégories:

- Les *machines systèmes* qui offraient à l'utilisateur des fonctions systèmes élaborées.
- Les *machines langages* qui étaient conçues pour exécuter des langages évolués après une simple phase de traduction.

Ces deux classes de machines ont pratiquement disparues actuellement. Force est de constater que les arguments qui ont prévalus à leur conception se sont retournés contre elles.

- Le logiciel de ces machines est devenu très complexe pour adapter ou contourner les fonctions matérielles qui s'avéraient inadaptées après quelques évolutions du logiciel.
- Ces machines étaient relativement lentes. La complexité du matériel ralentissant le cycle machine.

Globalement les machines complexes sont appelées *CISC* (pour Complex Instrucion Set Computer). Toutefois, cette appellation recouvre des machines "classiques" comme le Vax 780 et les machines complexes.

7.1 Les machines système

Dans les années 1970, certaines machines (machine Multics, ICL série 3900, BULL DPS 7-7000, Intel IAPX 432, et plus récement la série Intel x86 (à partir du 286)) offraient des fonctions systèmes élaborées telles que la gestion de tâches et leur synchronisation, la gestion d'une mémoire segmentée, etc... à leur utilisateurs. Ces machines étaient presque toutes inspirées du projet Multics développé au MIT vers la fin des années 60. Toutes ces fonctions étaient réalisées par des instructions très complexes (sémaphores, gestion de tâche, etc....). On trouve encore, sur certaines machines CISC des instructions de traitement de chaîne, de bouclage et de gestion d'un espace mémoire segmenté.

7.2 Les machines langage

Certaines machines ont été conçues dans les années 60 et 70 pour faciliter l'exécution de programmes en langage évolué. Ceux-ci (Cobol, Fortran, Algol, Pascal, Lisp, Prolog, Smaltalk et plus récemment Java) sont transformés en instructions post-fixées qui s'exécutent sur une pile. Malgré des performances intéressantes, ces machines ont disparu les unes à la suite des autres.

8 LES MACHINES RISC

A la fin des années 70, en réaction à l'échec des machines complexes, la mode revint à la conception de machines très simples. La technologie des circuits intégrés n'avait pas encore atteint des complexités qui permettaient de réaliser des machines complexes de manière monolithique. En 1975 un certain John Cocke a inventé la notion de processeur *RISC* (pour Reducted Instruction Set Computer) dans un laboratoire de recherche d'IBM. La base du raisonnement qui a amené la conception des machines RISC à été la constatation du grand déséquilibre qui existait entre les taux d'exécution des différentes instructions d'une machine CISC. Certaines instructions pouvant avoir des taux d'exécution de l'ordre de 1‰ . En ne conservant que les instructions les plus utilisées, les processeurs RISC se montrèrent mieux optimisés et finalement plus rapides car le matériel devenu plus simple put être optimisé pour la vitesse. Le logiciel devint aussi plus performant, car plus à même d'optimiser des séquences d'instructions simples. En contrepartie, la taille des programmes des processeurs RISC est notablement plus grande (le double ?) car ils sont écrits avec des instructions moins expressives. Les machines RISC sont caractérisées par l'utilisation de nombreux registres qui réduisent les échanges de données avec la mémoire, très peu de format d'instruction différents pour faciliter et accélérer le décodage des instructions ainsi qu'un schéma de séquencement unique des instructions pour en simplifier l'exécution.