Etude fonctionnelle d'un processeur de bases de données hiérarchiques

Gilles Berger Sabbatel

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE

Génie Informatique

par

Gilles BERGER SABBATEL

ETUDE FONCTIONNELLE D'UN PROCESSEUR DE BASES DE DONNEES HIERARCHIQUES

Thèse soutenue le 22 juin 1978 devant la Commission d'Examen :

Président : Monsieur L. BOLLIET

Examinateurs : Messieurs ANCEAU, DELOBEL, GIRAULT, RACINE

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE

Génie Informatique

par

Gilles BERGER SABBATEL

ETUDE FONCTIONNELLE D'UN PROCESSEUR DE BASES DE DONNEES HIERARCHIQUES

Thèse soutenue le 22 juin 1978 devant la Commission d'Examen :

Président : Monsieur L. BOLLIET

Examinateurs : Messieurs ANCEAU, DELOBEL, GIRAULT, RACINE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année Universitaire
1977-1978

Président : M. Philippe TRAYNARD

Vice-Présidents : M. René PAUTHENET
M. Georges LESPINARD

PROFESSEURS TITULAIRES

MM	BENOIT Jean	Electronique - Automatique
	BESSON Jean	Chimie Minérale
	BLOCH Daniel	Physique du solide - cristallographie
	BONNETAIN Lucien	Génie Chimique
	BONNIER Etienne	Métallurgie
	*BOUDOURIS Georges	Electronique - Automatique
	BRISSONNEAU Pierre	Physique du Solide - cristallographie
	BUYLE-BODIN Maurice	Electronique - Automatique
	COUMES André	Electronique - Automatique
	DURAND Francis	Métallurgie
	FELICI Noël	Electronique - Automatique
	FOULARD Claude	Electronique - Automatique
	LANCIA Roland	Electronique - Automatique
	LONGEQUEUE Jean-Pierre	Physique Nucléaire Corpusculaire
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean-Charles	Chimie-Physique
	PAUTHENET René	Electronique - Automatique
	PERRET René	Electronique - Automatique
	POLOUJADOFF Michel	Electronique - Automatique
	TRAYNARD Philippe	Chimie - Physique
	VEILLON Gérard	Informatique Fondamentale et appliqué

*en congé pour études.

PROFESSEURS SANS CHAIRE

MM	BLIMAN Samuël	Electronique - Automatique
	BOUVARD Maurice	Génie Mécanique
	COHEN Joseph	Electronique - Automatique
	GUYOT Pierre	Métallurgie Physique
	LACOUME Jean-Louis	Electronique - Automatique
	JOUBERT Jean-Claude	Physique du Solide - Cristallographie
	ROBERT André	Chimie Appliquée et des Matériaux
	ROBERT François	Analyse numérique
	ZADWORNY François	Electronique - Automatique

MAITRES DE CONFERENCES

MM	ANCEAU François	Informatique Fondamentale et appliqué
	CHARTIER Germain	Electronique - Automatique
	CHIAVERINA Jean	Biologie, biochimie, agronomie
	IVANES Marcel	Electronique - Automatique
	LESIEUR Marcel	Mécanique
	MORET Roger	Physique Nucléaire - Corpusculaire
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	SABONNADIERE Jean-Claude	Informatique Fondamentale et appliqué
Mme	SAUCIER Gabrielle	Informatique Fondamentale et appliqué
	SOHM Jean-Claude	Chimie Physique

.../...

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

CHERCHEURS DU C.N.R.S. (Directeur et Maîtres de Recherche)

M	FRUCHART Robert	Directeur de Recherche
MM	ANSARA Ibrahim	Maître de Recherche
	BRONOEL Guy	Maître de Recherche
	CARRE René	Maître de Recherche
	DAVID René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	Maître de Recherche
	MATHIEU Jean-Claude	Maître de Recherche
	MERMET Jean	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (Décision du Conseil Scientifique)

E.N.S.E.E.G.

MM	BISCONDI Michel	Ecole des Mines ST ETIENNE (dépt.Métallurgie)
	BOOS Jean-Yves	Ecole des Mines ST ETIENNE (Métallurgie)
	DRIVER Julian	Ecole des Mines ST ETIENNE (Métallurgie)
	KOBYLANSKI André	Ecole des Mines ST ETIENNE (Métallurgie)
	LE COZE Jean	Ecole des Mines ST ETIENNE (Métallurgie)
	LESBATS Pierre	Ecole des Mines ST ETIENNE (Métallurgie)
	LEVY Jacques	Ecole des Mines ST ETIENNE (Métallurgie)
	RIEU Jean	C.E.N. Grenoble (Métallurgie)
	SAINFORT	U.S.M.G.
	SOUQUET	
	CAILLET Marcel	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	COULON Michel	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	GUILHOT Bernard	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	LALAUZE René	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	LANCELOT Francis	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	SARRAZIN Pierre	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	SOUSTELLE Michel	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	THEVENOT François	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	THOMAS Gérard	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	TOUZAIN Philippe	Ecole des Mines ST ETIENNE (Chim.Min.Ph)
	TRAN MINH Canh	

E.N.S.E.R.G.

MM	BOREL	Centre d'Etudes Nucléaires de GRENOBLE
	KAMARINOS	Centre National Recherche Scientifique

E.N.S.E.G.P.

MM	BORNARD	Centre National Recherche Scientifique
Mme	CHERUY	Centre National Recherche Scientifique
MM	DAVID	Centre National Recherche Scientifique
	DESCHIZEAUX	Centre National Recherche Scientifique

*
* *
*

Cette thèse est le fruit de quatre années de travail,
quatre années de joies, de souffrances,
de recherches, pas toutes tournées vers l'informatique ...
de découvertes enfin.
Aussi, je voudrais la dédier à

Marie Rousseau

et a son frere Daniel,

qui sont à l'origine de ces découvertes,
et à tous ceux qui, avec eux, m'ont aidé et m'aident encore
à changer ma vie.

Je tiens à remercier

Monsieur BOLLIET, Professeur à l'IUT de Grenoble, qui a bien voulu me faire l'honneur de présider le jury de cette thèse,

Monsieur GIRAULT, Professeur à Paris VI et Monsieur DELOBEL Professeur à l'Université de Grenoble, dont les suggestions m'ont aidé à améliorer la présentation de ce travail,

Monsieur RACINE, Chef de la Division Systèmes Informatiques à la SAGEM, qui a apporté son soutien à ce projet,

Monsieur ANCEAU, Maître de Conférence à l'ENSIMAG, qui a contribué à la progression de ce projet par ses conseils et encouragements,

Monsieur SARRE, Ingénieur à la SAGEM, qui a contribué à donner à ce projet l'aspect industriel indispensable pour en envisager la réalisation,

Monsieur NAVAUX qui s'est chargé de la partie matérielle de cette étude, et tous les membres de l'équipe de recherche en architecture d'ordinateurs, avec qui j'ai pu discuter nombre de points de cette étude,

Madame CHALAND qui assuré la frappe de ce texte avec patience et efficacité, ainsi que Mademoiselle BOULLESTEIX,

et enfin, le service de reprographie du Laboratoire IMAG qui a assuré le tirage de ce document.

SOMMAIRE

CHAPITRE I

PROCESSEURS SPÉCIALISÉS

I.1. INTRODUCTION

Dans les systèmes informatiques actuels, la plus grande part du traitement est effectuée par un ou plusieurs processeurs "banalisés". Ces processeurs doivent permettre d'effectuer les tâches les plus diverses : traitement des programmes utilisateurs, gestion du système, entrées-sorties, compilation, etc ...

Il en résulte des processeurs rapides dotés d'instructions très puissantes, composant souvent un langage de programmation pénible à utiliser, bien qu'il soit souvent le seul moyen d'utiliser la machine de manière optimale.

Ces processeurs sont très coûteux, aussi a-t-on depuis longtemps cherché à les décharger de parties de leur travail en les confiant à des automates externes plus simples. Le cas typique est celui des entrées-sorties, où les échanges de données sont contrôlés par les canaux pendant que le processeur central exécute un autre travail.

En poursuivant cette démarche, on peut se demander, vue l'évolution de la technologie, s'il n'est pas possible et intéressant de décentraliser des fonctions beaucoup plus importantes : ceci nous amène de la conception de la conception de systèmes où les processeurs sont universels et banalisés à la conception de systèmes où les processeurs seraient spécialisés. Un système ne serait plus constitué d'un processeur central effectuant la plus grande part du travail, avec l'existence d'organes périphériques effectuant des tâches beaucoup plus humbles, mais de plusieurs processeurs spécialisés auxquels seraient confiées des parties différentes du traitement, constituant des ressources système.

I.2. INTÉRÊT DES PROCESSEURS SPÉCIALISÉS

L'insertion de processeurs spécialisés dans un système peut donner lieu à deux démarches différentes, selon que l'on souhaite remettre en cause l'architecture des systèmes ou non.

I.2.1. Systèmes actuels

Des processeurs spécialisés peuvent être connectés à une unité centrale de la même manière qu'un périphérique. Dans de nombreux systèmes, on voit maintenant des unités périphériques dotées d'une certaine intelligence : des mini-ordinateurs sont quelquefois utilisés comme contrôleurs de disques, ou comme concentrateurs de terminaux.

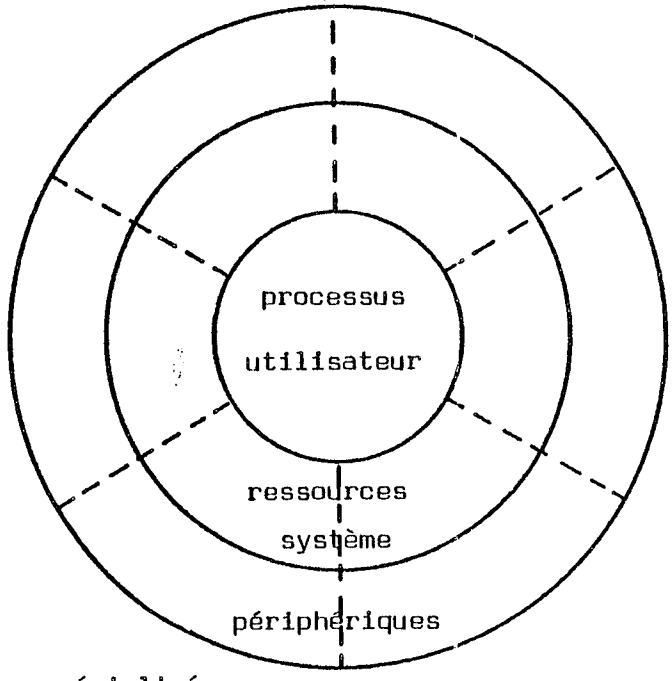
L'introduction de microprocesseurs dans les terminaux permet déjà une sophistication des échanges à l'aide d'une mémoire tampon locale, mais on peut aussi voir des terminaux "intelligents" permettant, par exemple, une édition de texte locale, voire même un pré-traitement syntaxique.

Il est donc extrêmement tentant de chercher à aller encore plus loin. On pourrait alors voir des unités centrales entourées de processeurs spécialisés, généralement associés à la gestion d'un périphérique : processeur de base de données, processeur de dialogue avec l'utilisateur, processeur de virtualisation de ressources (spooling), compilateur et interpréteur de langage évolué, etc ...

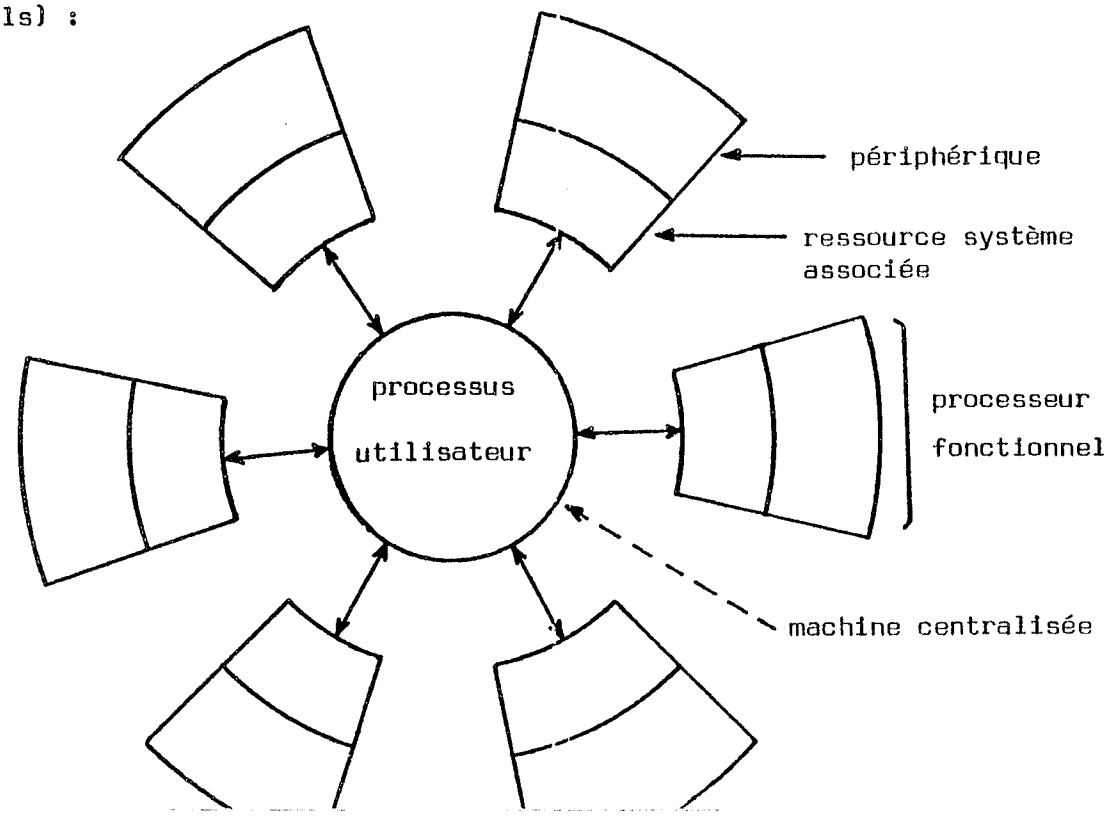
On a donc un véritable éclatement des systèmes, l'unité centrale conservant un rôle de gestion du système, de synchronisation, de communication et d'exécution de la plus grande part des programmes des utilisateurs.

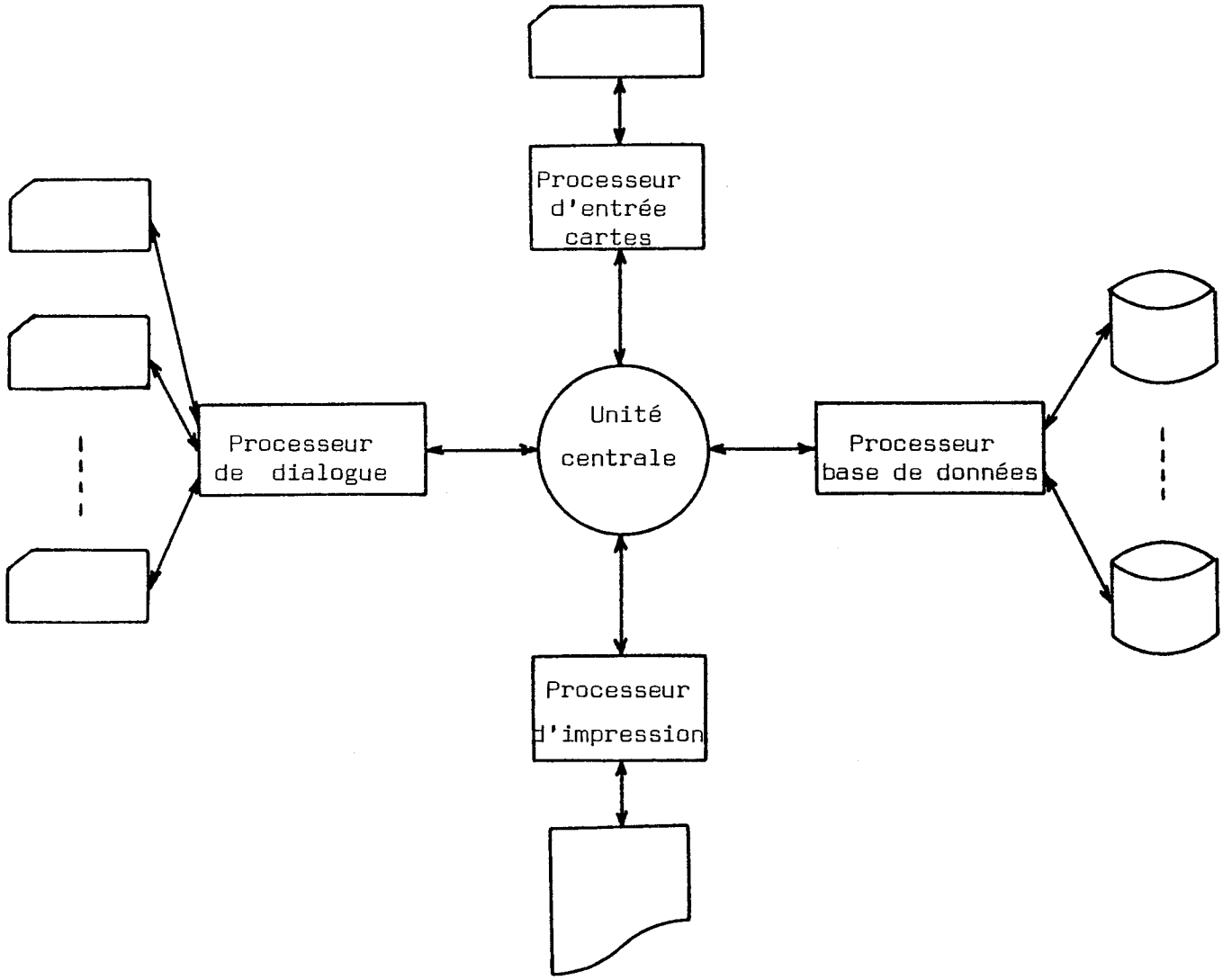
Si on considère un système informatique comme l'imbrication de trois niveaux : périphériques, ressources système, processus utilisateurs, cette démarche aboutit à l'éclatement des deux premiers niveaux :

Systèmes actuels :



Systèmes à processeurs spécialisés (ou fonctionnels) :




Processeur
d'entrée
cartes
Processeur
de dialogue
Unité
centrale
Processeur
base de données
Processeur
d'impression

L'intérêt d'une telle disposition est évident : libérer les unités centrales de certains travaux qui peuvent être exécutés de façon aussi efficace par des processeurs plus économiques, parce que spécialisés. La conception des systèmes n'est pas profondément remise en question et un tel processeur spécialisé ne devrait pas, normalement, être plus difficile à insérer dans un système que n'importe quel périphérique.

L'utilisation de processeurs spécialisés peut être une manière intéressante de moduler la puissance des systèmes : à partir d'un système de base où toutes les fonctions sont réalisées par logiciel sur l'unité centrale, on peut augmenter la puissance en adoptant le ou les processeurs spécialisés les plus appropriés aux besoins de l'exploitation. Ceci devrait être une solution plus économique pour l'utilisateur que l'adoption du modèle supérieur de la gamme.

Par ailleurs, une certaine souplesse des systèmes en cours d'exploitation peut devenir possible. En effet, si un système devient surchargé, on pourra souvent augmenter sa puissance en installant un processeur spécialisé. Ceci entraîne une perturbation minimale de l'exploitation, ce qui est loin d'être le cas lorsqu'il faut adopter un modèle d'unité centrale supérieur, avec les conséquences que cela peut entraîner sur les mémoires, les périphériques, les locaux, voire même la climatisation .. Cette souplesse pourrait permettre aux utilisateurs d'adopter des systèmes correspondant à l'importance effective de leurs besoins, sans avoir à tenir compte de l'évolution future de ces besoins.

Dans la conception matérielle d'un processeur spécialisé, on recherche la meilleure adaptation possible au traitement à effectuer. Le processeur étant par ailleurs entièrement consacré à un seul travail, on peut espérer que son logiciel sera beaucoup plus simple que le logiciel qui effectuerait le même travail sur l'unité centrale. On peut donc espérer un gain sensible en fiabilité et en facilité de maintenance du logiciel, puisqu'il sera simple et sans interactions étroites avec d'autres logiciels.

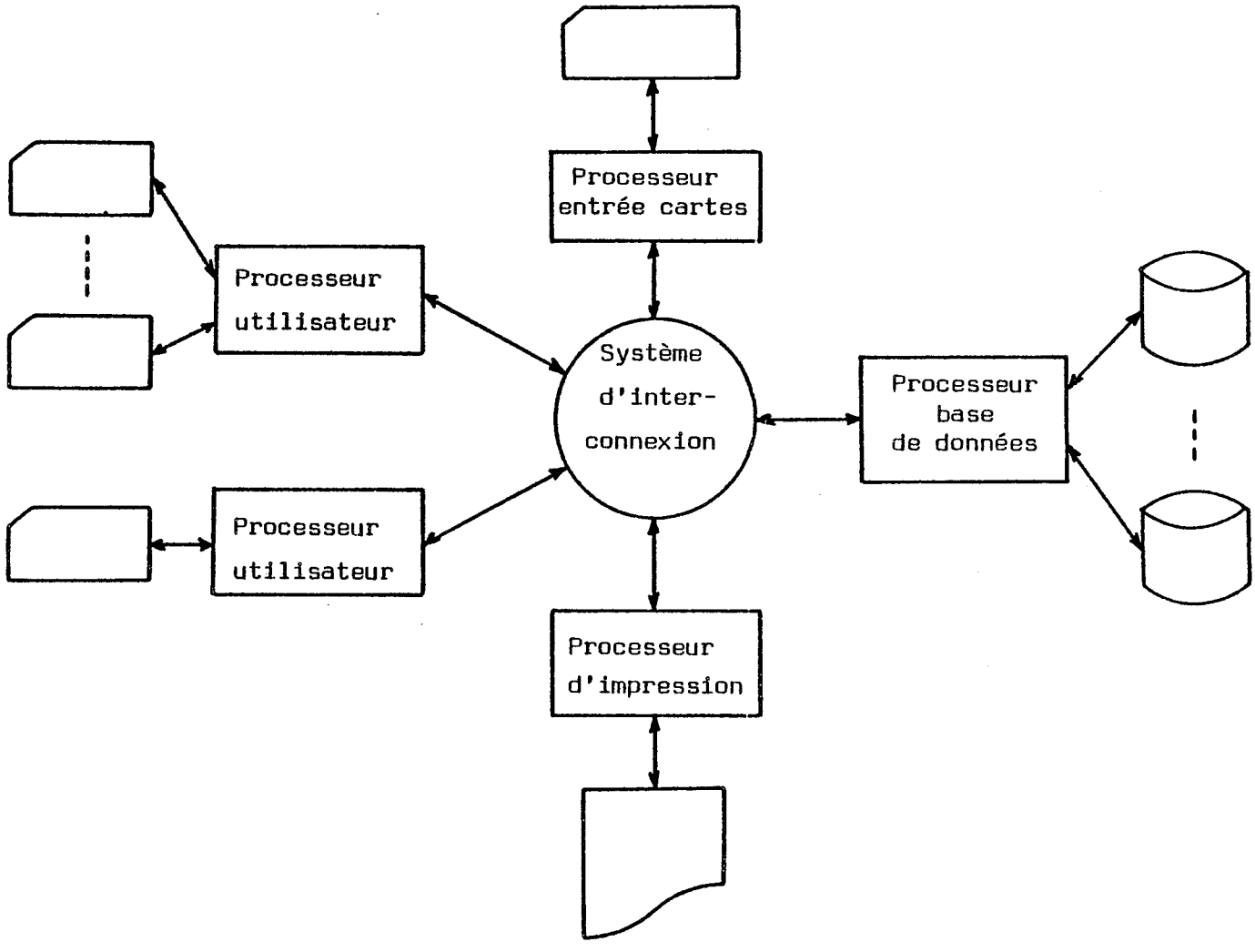
L'apparition des microprocesseurs a ouvert de nouvelles perspectives et augmenté considérablement l'intérêt des processeurs spécialisés. D'une part il devient possible de décentraliser des fonctions relativement humbles pour lesquelles un mini-ordinateur serait sous-employé, d'autre part, il devient intéressant d'envisager l'utilisation de processeurs spécialisés dans des systèmes de petite taille : il n'est pas absurde de vouloir décharger un mini-ordinateur d'une partie de son travail en le confiant à un processeur beaucoup moins cher, réalisé à l'aide de microprocesseurs.

I.2.2. Intérêt dans les systèmes futurs

Il est tout à fait possible d'envisager de décentraliser chaque fonction d'un système : la fonction traitement elle-même peut être confiée à un processeur spécialisé doté d'un langage mieux adapté à l'utilisateur [SC].

Si chaque processeur se gère lui-même de façon autonome, l'unité centrale n'a plus qu'un rôle de communication et de synchronisation. On peut alors envisager de se passer de cet intermédiaire encombrant en inter-connectant directement entre eux les processeurs spécialisés à l'aide d'un système de connexions à définir.

On arrive ainsi à des systèmes répartis fonctionnellement, où la notion d'unité centrale n'existe plus [PO].


Processeur
entrée cartes
Processeur
utilisateur
Système
d'inter-
connexion
Processeur
base
de données
Processeur
utilisateur
Processeur
d'impression

Un système est alors constitué par un assemblage de plusieurs éléments judicieusement choisis pour répondre au mieux aux besoins de l'utilisateur.

Une autre approche de ce type d'architecture est rendue possible par l'apparition des microprocesseurs. Ces circuits permettent déjà de disposer d'un processeur sous un volume réduit (un seul circuit intégré) et à un prix réduit [PR]. Les développements de cette technologie sont tels que l'on annonce des microprocesseurs qui fournissent, en un seul circuit, la puissance de calcul d'un mini-ordinateur 16 bits (INTEL 8086, ZILOG Z 8000, ...) Ces circuits devraient, en outre, posséder des interfaces d'entrée-sortie et quelques K octets de mémoire interne, pour un prix du même ordre que celu des microprocesseurs actuels [AN2].

On peut donc se demander s'il n'est pas possible de réaliser des systèmes informatiques de puissance moyenne ou grande en utilisant plusieurs microprocesseurs travaillant en parallèle.

Une première idée est de réaliser une structure multi-microprocesseurs où chaque microprocesseur serait banalisé. Une solution de ce genre a été étudiée [BO], [MA]. Elle pose des problèmes au niveau de l'accès aux mémoires partagées, des synchronisations, de la répartition du travail entre les processeurs, etc .. Par ailleurs, on retrouve la nécessité de faire intervenir la multiprogrammation pour obtenir des performances optimales de l'ensemble de la machine. Ce type d'architecture a toutefois l'avantage de proposer des systèmes d'usage général (non spécialisé), au prix cependant d'un langage qui n'est pas forcément sympathique.

On peut aussi chercher à spécialiser chaque processeur en lui faisant réaliser une fonction bien définie, ce qui revient à l'architecture précédemment décrite. Chaque processeur possède sa propre mémoire pour l'exécution des programmes. On n'a donc aucun problème de partage de mémoire.

Le faible coût des microprocesseurs permet de fournir un processeur par utilisateur : il n'y a aucune raison de se préoccuper du sous-emploi de chaque processeur. Ceci permet donc de réduire, dans une large mesure, les problèmes liés à la multiprogrammation, bien qu'ils restent présents au niveau de certains processus liés à des ressources communes : disque, imprimante, lecteur de cartes, ...

Chaque processeur sera conçu en fonction de son utilisation, ce qui signifie dans le cas de l'utilisation de microprocesseurs, que chaque processeur sera doté des opérateurs externes et de la structure de mémoire les plus propres à faciliter le traitement : on peut donc espérer une simplification du logiciel.

Une telle architecture entraîne une perte globale d'espace mémoire, du fait de la multiplication dans chaque processeur des zones de travail et de sous-programmes exécutant les mêmes fonctions. Cependant, la diminution du coût des mémoires diminue de plus en plus cet inconvénient.

L'intérêt de ce type de décentralisation est autre que la recherche de performances :

- la modularité physique permet une stricte adaptation aux besoins de l'utilisateur et à leur évolution ;
- chaque fonction peut évoluer isolément, si l'interface logiciel et matériel est figé et standardisé ;
- l'isolement des fonctions doit accroître fortement la sécurité des systèmes. Si un processeur est défectueux, les erreurs ne devraient pas se propager aux autres processeurs, qui seraient en mesure de les détecter ;
- l'augmentation du coût de chaque processeur, par rapport à un contrôleur de périphériques, doit être faible vu le faible coût des composants électroniques ;
- la possibilité de disposer de langages relativement évolués au niveau des utilisateurs, devrait conduire à une simplification de la programmation des applications et donc à un gain en coût de programmation et de maintenance des logiciels.

Les microprocesseurs actuellement disponibles permettent d'envisager ce type d'architecture pour des petits systèmes. On peut en attendre, pour un prix équivalent à celui des systèmes actuels, une meilleure adaptabilité aux besoins des utilisateurs et des outils logiciels de plus haut niveau pour la réalisation des applications.

I.3. MÉTHODOLOGIE DE CONCEPTION DE PROCESSEURS SPÉCIALISÉS

La conception des unités centrales consiste généralement à utiliser une architecture connue pour réaliser un ensemble d'opérations bien défini (opérations arithmétiques et logiques, transferts de données, branchements, ...), qui doit fournir à l'utilisateur tous les outils dont il peut avoir besoin. Le type d'utilisation envisagé pour la machine, implique un certain jeu d'instructions auquel s'ajoutent éventuellement un certain nombre d'instructions particulières destinées à faciliter des opérations fréquentes (éditions, conversions, boucles, ..), ou le travail du système d'exploitation. Les choix qui se posent au concepteur sont donc le type d'adressage et les instructions particulières à réaliser.

Le problème de la conception d'un processeur spécialisé est sensiblement plus compliqué. Il ne s'agit plus de réaliser un ensemble type d'opérations simples, mais de réaliser une fonction qui peut être complexe.

Il faut donc, en premier lieu, définir la fonction à réaliser. Le seul nom de la fonction est généralement insuffisant pour définir un processeur ; c'est-à-dire qu'il sera souvent impossible de définir un processeur permettant d'exécuter de manière optimale n'importe quelle réalisation d'une même fonction.

La conception d'un processeur fonctionnel standard peut s'envisager au sein d'une famille de réalisations de la fonction et sous réserve d'une utilisation non optimale. Ainsi, le terme "base de données" recouvre-t-il une foule de conceptions différentes entre lesquelles il serait difficile de trouver d'autre point commun que l'utilisation d'une mémoire secondaire. Concevoir un processeur de base de données capable d'exécuter efficacement n'importe quel système, reviendrait sans doute à concevoir un mini-ordinateur d'une puissance très supérieure à celle nécessitée par une application moyenne. En règle générale, on cherchera donc à préciser beaucoup plus la fonction en tenant compte de l'utilisation projetée.

Une première description externe de la fonction amène à définir les différentes requêtes que l'on désire pouvoir effectuer. Ceci revient donc à définir un langage qui serait le langage de commande "idéal" du processeur. A ce stade, la définition du langage n'a pas besoin d'être précise, car elle constitue un simple "cahier des charges" et c'est sans doute un langage "intermédiaire" qui sera implémenté.

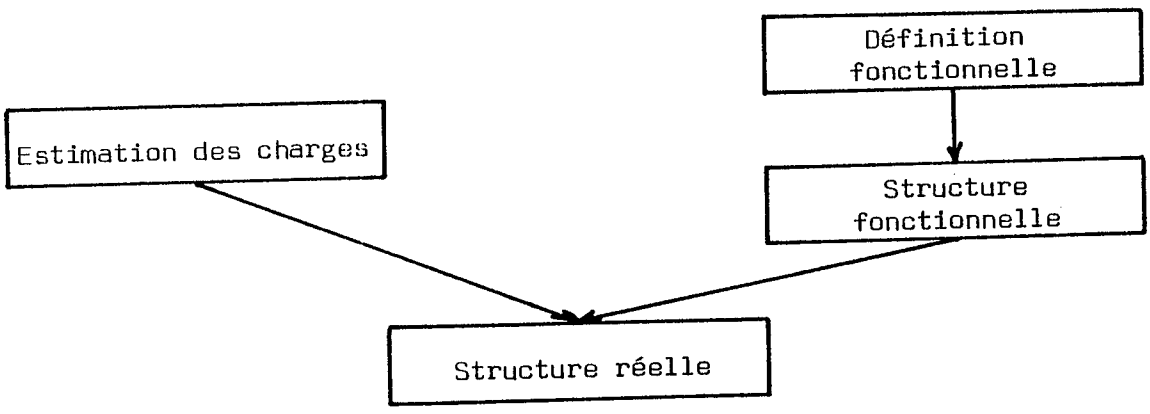
La seconde étape amènera à rechercher les principes de l'interprétation du langage de commande. Dans le cas d'un processeur de base de données par exemple, on devra définir une technique d'adressage des données. Cette étape peut amener une première décomposition fonctionnelle du système. Elle doit également permettre de définir le langage du processeur. Ce langage devra tenir compte des contraintes d'implémentation, du contexte d'utilisation, de la facilité d'échange des requêtes entre les processeurs. Il sera vraisemblablement d'un niveau inférieur au langage proposé lors de la définition de la fonction.

A ce point, la définition fonctionnelle du processeur est terminée ; il est donc nécessaire de commencer à étudier la réalisation. Cette étude portera à la fois sur le logiciel (étude des algorithmes) et le matériel. D'une façon générale, on peut considérer que la fonction sera trop complexe pour être câblée, ou même microprogrammée. On doit donc la réaliser par programmation sur un processeur existant ou à définir.

Si la fonction peut être exécutée par un ou plusieurs microprocesseurs, le problème sera de choisir les microprocesseurs, de déterminer la façon de les inter-connecter, et de structurer leur espace d'adressage, de déterminer les opérateurs périphériques dont ils doivent être munis.

Si des problèmes de rapidité imposent le choix d'un processeur plus puissant il faudra être capable de choisir ce processeur ou d'en déterminer l'architecture.

Tous ces problèmes demandent d'estimer la puissance de traitement mémoire à partir d'une évaluation de la charge. Cette évaluation va demander un approfondissement de l'analyse du traitement. La charge pourra ainsi être déterminée par estimation, simulation, ou essai en conditions réelles. Un critère d'efficacité devra être recherché : ce peut être l'optimisation du temps de réponse, l'équilibre entre le traitement et les entrées-sorties, .. Ceci permettra de déterminer quel processeur doit être employé et de proposer une architecture.



Les étapes suivantes verront la réalisation du logiciel et du matériel, la réalisation du logiciel pouvant préciser des paramètres utiles à la réalisation du matériel.

CHAPITRE II

PRÉSENTATION GÉNÉRALE

DU PROJET MAGE

II.1. INTRODUCTION

Parmi les fonctions susceptibles d'être confiées à un processeur spécialisé, la gestion des données suscite depuis longtemps l'intérêt des concepteurs de matériel informatique [CG], [CH].

En effet, les unités de disque sont le plus souvent contrôlées par des unités relativement sophistiquées et par ailleurs, le logiciel de gestion de fichiers représente généralement une importante partie des systèmes d'exploitation. On s'est donc demandé si, en augmentant un peu la complexité des unités de contrôle disque, on ne pourrait pas leur faire exécuter une part importante des fonctions du logiciel de gestion de fichiers à un "coût" moindre que lorsqu'elles sont exécutées par l'unité centrale.

Par ailleurs, les recherches entreprises sur les bases de données, et plus particulièrement les bases de données relationnelles [CO] ont suscité un certain nombre de réflexions visant à remettre en question les mécanismes traditionnels d'accès au disque. Un certain nombre d'études ont conduit à la conception de processeurs de bases de données, basés sur une architecture "cellulaire" et un accès associatif au disque : nous citerons particulièrement RAM [SO], CASSM [SL], [LI], RARES [LS].

Dans ces architectures, une cellule est constituée d'un segment de mémoire avec ses mécanismes d'accès (exemple : une piste de disque à tête fixe) et d'une logique permettant en particulier d'effectuer des comparaisons "à la volée" sur les informations en cours de transfert : on peut donc, par ce mécanisme, accéder associativement au disque en fournissant à chaque cellule la valeur de la chaîne recherchée et en effectuant la recherche en parallèle sur chaque cellule.

Dans le cas où la mémoire secondaire est constituée par un disque à tête fixe, on peut donc trouver la donnée recherchée en un seul tour de disque. Les divers projets diffèrent essentiellement par l'organisation des données sur le disque et les fonctions proposées. RARES et RAP sont orientés vers les bases de données relationnelles, alors que CASSM se veut plus universel ; RARES présente comme particularité une organisation "radiale" des données, c'est-à-dire que des octets successifs d'une donnée se trouvent répartis non sur une piste, mais sur plusieurs pistes le long d'un rayon du disque.

Ce type d'architecture appelle quelques remarques :

. Tout d'abord, l'accès simultané à toutes les pistes d'un disque demande d'avoir une électronique d'accès et de comparaison pour chaque tête, au lieu d'une seule pour l'ensemble des têtes, ceci en plus de la logique de traitement. On peut donc objecter le coût de la multiplication de cette électronique. Cette objection est rejetée en considérant que l'ensemble amplificateurs de lecture/écriture + logique, peut être intégré sur un seul circuit intégré à grande échelle, de coût réduit.

. La seconde remarque concerne le support de la mémoire secondaire. Dans l'état actuel de la technologie, l'accès simultané à tous les segments de la mémoire secondaire ne peut être réalisé que sur des disques à tête fixe, disques coûteux et de faible capacité par rapport aux disques à bras mobile généralement utilisés comme supports de la mémoire secondaire. De plus, les disques à tête fixe sont rarement amovibles.

On peut évidemment parler des nouvelles technologies de mémoires secondaires (CCD, bulles), mais la compétition avec les disques est très serrée en ce qui concerne le coût et l'encombrement, et il n'est pas évident que ces technologies puissent se substituer aux disques de très grande capacité dans un proche avenir.

On peut également utiliser des disques à bras mobile en adoptant un système de pagination, le "swapping" étant réalisé par le déplacement des têtes. Un tel système est proposé par Ozkarahan , Schuster et Smith, pour RAP, en se basant sur des hypothèses de localité.

En France, un projet est basé sur l'utilisation d'un opérateur d'accès associatif au disque [QS], alors qu'un autre repose sur l'utilisation d'une vaste mémoire associative réalisée à l'aide de CCD [UN].

Le projet MAGE (Matériel Adapté à la Gestion d'Entités), objet de cette étude, a démarré en Janvier 1975 en collaboration avec la SAGEM (Société d'Application Générale d'Electricité et de Mécanique), dans le cadre d'un contrat du SESORI [SE].

Nous nous sommes plus particulièrement intéressés à la conception d'un système (matériel et logiciel) de gestion de données adapté à des petits systèmes "clefs en main". Dans ce type de système, l'utilisateur dispose de fonctions adaptées à la réalisation de son problème, sans avoir généralement la possibilité de les modifier ou d'en créer de nouvelles.

Les systèmes de ce type sont généralement orientés vers la saisie et le traitement des informations, et ils peuvent, malgré l'apparente modestie de leurs ambitions, donner lieu à des fichiers de structure très complexe.

Nous avons pris comme système type, le système SAGEM "SCRIB" (Saisie et Contrôle de Résultats et d'Informations Biologiques), système de gestion de laboratoires d'analyses médicales, particulièrement complexe et suffisamment représentatif des différents problèmes que peuvent poser les systèmes de ce type. Ce système gère un espace disque d'une dizaine de mega-octets, divisé en une trentaine de fichiers ; une quarantaine de liens inter-fichiers sont gérés par le programme d'application.

Pour diminuer les coûts de conception et de maintenance, le besoin d'un outil propre à une manipulation aisée de grandes quantités d'information, se fait fortement sentir.

Nous avons déjà discuté des avantages de la décentralisation des fonctions, tant dans les systèmes classiques que dans les systèmes répartis. Or, les recherches entreprises sur les systèmes répartis, ont montré que la fonction "gestion des données" était l'une des plus importantes, en particulier par sa complexité. Un processeur de gestion de données serait donc un des éléments fondamentaux d'un système réparti.

Ces réflexions nous ont amenés, pour le projet MAGE, à dépasser la notion de simple "processeur fichier", pour arriver à la définition d'un processeur de base de données.

Le projet MAGE se distingue des projets précédemment évoqués par le fait qu'il s'agit d'un processeur de base de données destiné à de petits systèmes, c'est à-dire que le processeur à définir devra être peu coûteux et travailler sur des bases de données de petite taille. Par ailleurs, ce projet devait aboutir à la définition d'un produit industriel, ce qui imposait de travailler avec les technologies existantes, sans trop d'hypothèses sur l'évolution à venir de la technologie et des coûts.

On doit donc en particulier prendre la technologie des mémoires secondaires telle qu'elle est au moment de la conception du processeur de base de données, c'est-à-dire basée sur l'utilisation de disques à bras mobile, sans possibilité d'accès simultané à plusieurs pistes.

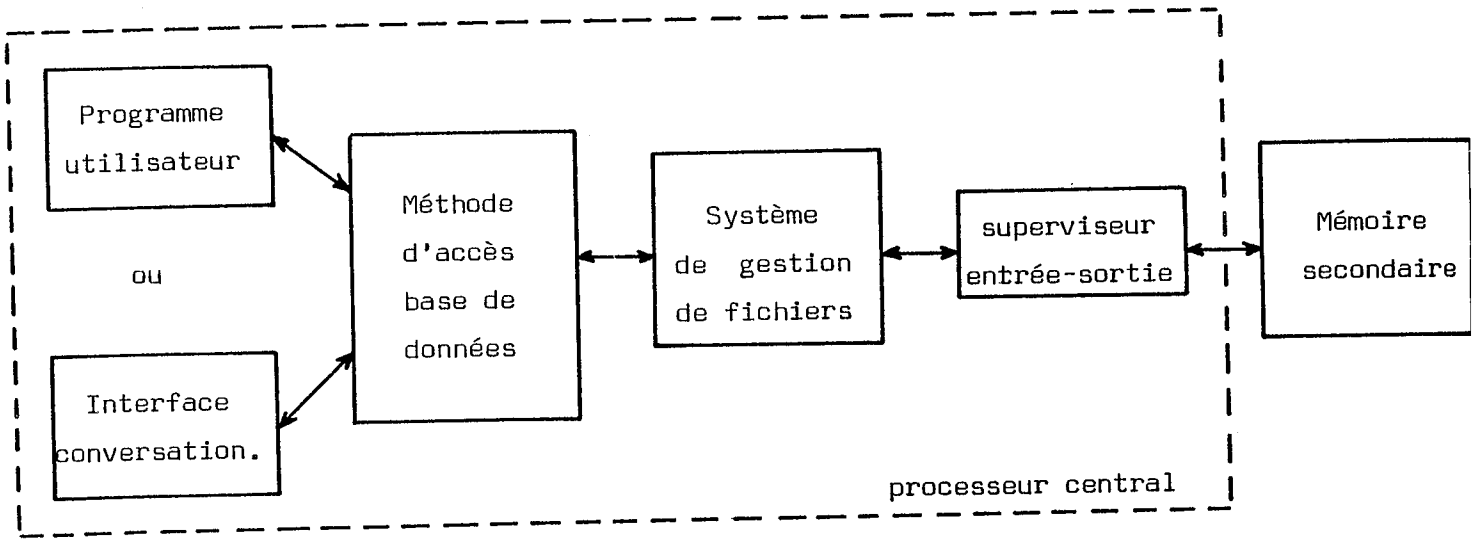
Par ailleurs, d'un point de vue plus philosophique, notre but n'était pas tant de rechercher des solutions originales pour accéder aux données, que de démontrer la possibilité et l'intérêt d'exécution une fonction telle que la gestion des données à l'extérieur de l'unité centrale et ce dans l'état actuel de la technologie.

Cette technologie nous impose donc de définir un "back-end processor" (selon la terminologie américaine), c'est-à-dire un processeur placé entre le disque et le processeur central, éventuellement associé à l'unité de contrôle du disque, plutôt qu'une "unité de disque intelligente", c'est-à-dire un processeur directement intégré à l'unité de disque elle-même.

II.2. PROCESSEUR DE BASE DE DONNÉES

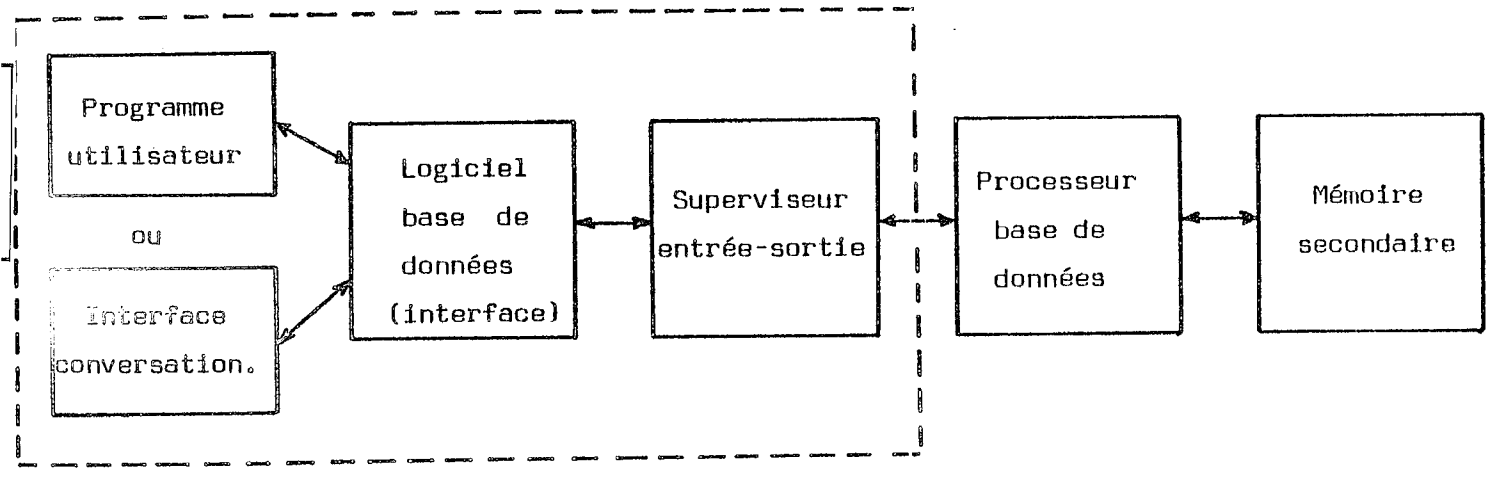
Nous considèrerons qu'une base de données est un ensemble de données structurées sur lesquelles plusieurs types d'accès sont possibles, chaque donnée pouvant éventuellement être désignée de plusieurs façons.

Dans un système classique, l'accès à la base de données peut se faire soit directement par un programme défini par l'utilisateur, soit au terminal par l'intermédiaire d'un interface conversationnel. Ces logiciels appellent la méthode d'accès base de données qui elle-même accède à la mémoire secondaire par l'intermédiaire du système de gestion de fichiers et du superviseur d'entrée-sortie.

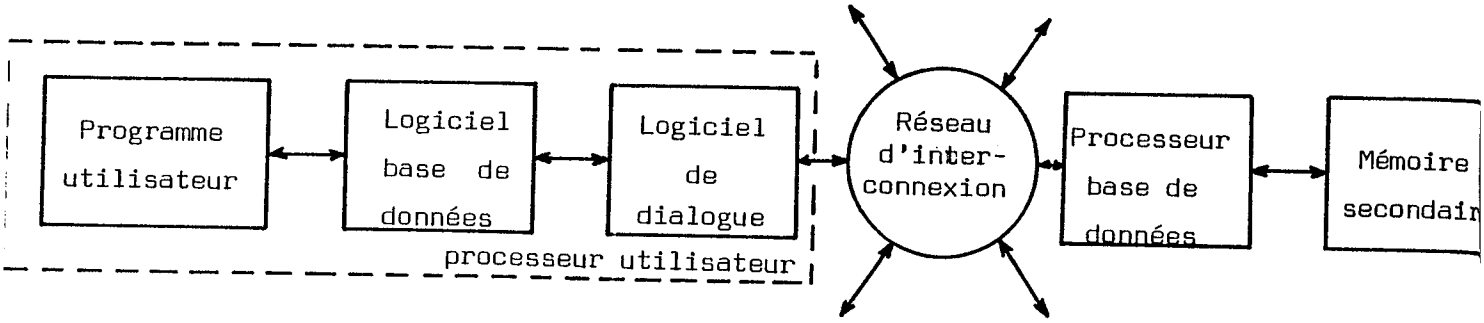


Les informations échangées entre la méthode d'accès et la mémoire secondaire correspondent généralement à l'unité d'échange physique du périphérique ; de plus, plusieurs échanges peuvent être nécessaires pour obtenir une information, alors que la méthode d'accès ne transmet à l'utilisateur que l'unité logique d'information qu'il a demandée, cette unité pouvant ne représenter qu'une très faible fraction d'un enregistrement physique.

L'idée de processeur de base de données consiste à exécuter tout ou partie de la méthode d'accès au niveau d'un processeur indépendant, attaché à la mémoire secondaire [AD].



Nous avons vu au chapitre précédent, qu'un processeur base de données pouvait également être utilisé dans un système réparti fonctionnellement :



Dans ces conditions, on peut énoncer un certain nombre de spécifications générales d'un processeur de base de données (PBD) :

. Le PBD doit exécuter la plus grande part possible de la méthode d'accès de manière à minimiser, voire supprimer le logiciel base de données nécessaire dans le processeur utilisateur. Le rôle du PBD se limite cependant à la seule méthode d'accès et il ne peut être utilisé que par l'intermédiaire d'un programme exécuté sur un autre processeur. Il n'existe aucune possibilité d'accéder directement le PBD de manière conversationnelle à partir d'un terminal.

. Les échanges d'information entre le PBD et le processeur utilisateur doivent être minimisés, c'est-à-dire, en particulier, que le PBD doit sélectionner dans les enregistrements physiques l'unité logique d'information demandée par l'utilisateur et lui transmettre cette seule unité. Le PBD doit donc posséder ses propres mémoires tampon. En outre, le langage du PBD doit être de nature à faciliter l'échange des requêtes entre les processeurs.

. Le PBD doit être conçu de façon suffisamment modulaire pour pouvoir s'adapter à différents contextes d'utilisation : dans un système classique, ou dans un système réparti.

. Le temps de réponse du PBD doit être suffisamment court pour ne pas dégrader les performances du système. Le PBD devra en général être multiprogrammé de façon à utiliser les temps morts durant les accès au disque, en se partageant entre plusieurs utilisateurs.

. Le PBD doit être assez adapté à son utilisation pour fournir des performances suffisantes à un coût minimal.

. Par ailleurs, le PBD sera conçu en vue de l'interprétation optimale d'un langage de base de données bien préci et il ne sera pas nécessairement possible d'interpréter sur le même matériel un langage radicalement différent avec la même efficacité.

II.3. SPÉCIFICATIONS DE MAGE

MAGE est conçu en vue de l'utilisation par des petits systèmes clefs en main Ceci va donc permettre de fixer un certain nombre d'hypothèses et de spécifications.

On a affaire à des systèmes non modifiables en cours d'exploitation, conçus et mis au point globalement par des spécialistes : la totalité des programme susceptibles de s'exécuter sur la machine est, en principe, définie avant la mise en service. Chaque programme est réputé au point dès que le système est mis en service et il n'y a pas, normalement, de programme en cours de mise au point sur un système en exploitation. On peut donc laisser au programme utilisateur des responsabilités qu'il ne serait pas possible de lui accorder dans un système où des programmes peuvent être mis au point en cours d'exploitation. La tâche du programmeur pourra être facilitée par des aides logicielles, permettant par exemple de passer d'un langage en représentation externe symbolique à un langage en représentation interne codée.

II.3.1. Aspect matériel

. Mémoire secondaire

La mémoire secondaire de MAGE sera constituée d'un ou plusieurs unités de disques à bras mobile. Sa capacité totale pourra atteindre une centaine de méga-octets et le débit être très élevé (jusqu'à 10 méga-bits par seconde) de façon à pouvoir utiliser les unités de disque les plus performantes disponibles sur le marché.

Dans un petit système, le nombre de périphériques est assez réduit, aussi les périphériques reliés au processeur base de données représenteront-ils souvent l'ensemble de la mémoire secondaire en accès direct du système. Or, il peut être nécessaire de disposer d'espace disque extérieur à la base de données : pour stocker des segments de programme, par exemple. En effet, pour des fichiers de structure simple, la base de données peut être un mécanisme lourd et inutile et il est préférable de s'en passer.

Le disque doit pouvoir stocker à la fois la base de données et des fichiers classiques, et MAGE devra permettre à d'autres processeurs d'accéder à ces fichiers, tout en étant capable d'interdire tout accès à l'espace disque de la base par un autre processeur que lui-même.

. Utilisateurs

Un certain nombre de paramètres matériels, tels que la taille des mémoires, dépendront du nombre d'utilisateurs. On doit donc fixer le nombre maximum d'utilisateurs. Pour les applications envisagées, en fonction du nombre moyen de terminaux (en général une dizaine), un nombre maximum de 32 processus utilisateurs simultanés doit être suffisant.

Comme nous l'avons vu, le processeur utilisateur peut être soit un mini-ordinateur, soit un réseau de microprocesseurs. MAGE doit pouvoir être connecté simultanément à plusieurs processeurs de même nature. L'interface (logiciel et matériel) avec le (ou les) processeur utilisateur doit être conçu de manière souple et modulaire pour pouvoir être adapté à des partenaires très différents, tant par les temps de réponse que par les protocoles de communication.

II.3.2. Les langages

Un système de gestion de base de données peut être défini par deux langages :

. le langage de définition de structure qui permet de décrire la structuration des données, leur adressage et les relations qui existent entre elles

. le langage d'accès qui permet, en relation avec la structure, d'adresser les données et d'effectuer dessus divers types d'accès.

Le langage de définition de structure peut être considéré comme le plus important. En effet, il permet de définir les différents éléments qui composeront la base de données, la façon dont on les adressera et les liens logiques et physiques qui existeront entre eux. Tous les concepts du système y sont donc définis et le langage d'accès en découle.

Le langage d'accès comporte deux aspects :

. la spécification de l'opération à effectuer : lecture, écriture, création, suppression, ..

. l'adressage de la donnée qui se fait, dans les bases de données hiérarchiques, en décrivant un "chemin" dans la structure.

La définition des langages demande une bonne connaissance des besoins de la gamme d'applications envisagée. Un langage mal adapté risquerait d'être ou trop coûteux à implémenter, ou peu efficace dans son utilisation.

L'examen de la structure des fichiers SCRIB fait apparaître un certain nombre de caractéristiques qui peuvent être traduites sans problème dans les systèmes classiques de gestion de bases de données : structure hiérarchique, liens inter-fichiers, accès direct, séquentiel ou associatif, ...

Le langage d'accès ne semble pas devoir être d'un très haut niveau : sachant que ce langage sera utilisé par les programmes du processeur utilisateur et que ces programmes sont écrits une fois pour toutes, on peut se contenter de primitives d'accès simples. Il est beaucoup plus intéressant de fournir un compilateur pour faciliter l'écriture des programmes que de faire interpréter par MAGE un langage évolué, car le compilateur n'intervient plus lorsque le système est en service, celui-ci étant figé.

De même, si la mise au point des programmes d'application nécessite des accès conversationnels à la base de données, il est beaucoup plus commode de fournir un interface conversationnel sur un processeur utilisateur, que de l'incorporer dans MAGE, puisqu'il sera inutile sur les systèmes en exploitation.

II.3.3. Sécurité

Le désir légitime de l'utilisateur d'un système est que ses informations ne risquent pas d'être détruites ou malencontreusement divulguées à la suite d'un incident ou d'une malveillance. Ceci pose donc le problème de la sécurité des informations, commun à tous les systèmes de gestion de données, mais particulièrement crucial dès que l'on met en oeuvre des possibilités d'accès sophistiqués et une structure complexe.

Ce problème comporte deux aspects : la sécurité des données vis à vis des utilisateurs (protéger les données contre les erreurs ou les malveillances des utilisateurs) et la sécurité des données vis à vis des incidents possibles, internes ou externes au système.

Dans un système clefs en main, les données sont accédées par l'intermédiair de programmes prédéfinis. Si des protections doivent intervenir, il est particulièrement simple et efficace de contrôler l'utilisation des programmes, ce qui n'est pas du ressort de MAGE. Si plusieurs utilisateurs accèden à des bases différentes par l'intermédiaire des mêmes programmes, il suffit que chaque programme contrôle que la base de données accédée est bien celle sur laquelle l'utilisateur est autorisé à travailler. Ce contrôle peut se faire lors de la première opération sur la base (ouverture), MAGE se chargeant ensuite de donner toujours accès à la même base.

Ce genre de protection n'est possible et efficace que si l'utilisateur ne peut pas programmer lui-même la machine. Dans le cas contraire, aucune protection n'est efficace si le processeur utilisateur ne dispose ni de protection mémoire, ni de mode privilégié.

Le véritable problème pour nous, est celui de la sécurité des informations vis à vis des défaillances du système : en effet, une défaillance peut entraîner des écritures erronnées ou interrompre une opération de mise à jour en laissant la base dans un état incohérent.

Ce risque ne peut pas être totalement éliminé. Il est donc nécessaire de le ramener à une probabilité acceptable. Trois catégories de mesures peuvent être prises :

. *limiter les risques de défaillance* : nous nous attacherons à définir un matériel et un logiciel aussi fiables que possible ; en particulier, la multiprogrammation va nécessiter des mécanismes de synchronisation pour éviter que l'exécution parallèle d'opérations de mise à jour ne puisse rendre la base incohérente ; cependant, le risque de défaillance ne sera jamais négligeable.

. *Limiter le risque qu'une défaillance entraîne des pertes d'information* : MAGE devra manipuler les données avec prudence : éviter qu'une information ne puisse être perdue si une panne survient pendant qu'on la manipule, limiter au maximum, dans les mises à jour, les périodes durant lesquelles la base n'est pas cohérente et durant ces périodes, organiser le traitement de façon à minimiser les conséquences d'une panne éventuelle. Ces précautions sont particulièrement importantes pour la manipulation des chaînages. Par ailleurs, il faudra être en mesure de détecter rapidement un mauvais fonctionnement, pour pouvoir arrêter le traitement et limiter les dégats. De même, une incohérence doit pouvoir être détectée rapidement pour éviter que le volume d'informations perdues ne puisse s'accroître.

. *Permettre, en cas de défaillance, de ramener la base à un état cohérent et de récupérer les informations perdues* : il peut être possible, dans certains cas, de réparer les fichiers en exploitant des redondances qui peuvent exister dans les informations de contrôle. Sinon, il faut disposer d'une copie saine de la base et d'une sauvegarde des modifications effectuées depuis la copie.

En ce qui concerne l'organisation des données, la sécurité implique de rechercher :

. une organisation simple de l'espace disque,
. une centralisation des moyens d'accès afin de diminuer le plus possible les liens à mettre à jour,
. des redondances d'information juste suffisantes pour contrôler la cohérence de la base et permettre éventuellement de petites restaurations.

Une source de perturbation importante dans les systèmes est constituée par les coupures de tension. Lorsque cet événement se produit, on doit être en mesure, lorsque la tension est rétablie, de reprendre le traitement interrompu avec le minimum de perturbations. Des mémoires non volatiles pourront être nécessaires afin de conserver les informations concernant les opérations en cours de traitement.

III.3.4. Spécifications diverses

. Coût

Le coût de MAGE devra être en rapport avec celui des systèmes de petite taille. Il est souhaitable que le processeur MAGE soit d'un coût très inférieur à celui d'une unité de disque.

. Performances

Les performances de MAGE devront être suffisantes pour n'entraîner aucune dégradation des temps de réponse par rapport aux systèmes traditionnels. De même, l'utilisation de la mémoire secondaire devra rester correcte.

. Souplesse

Dans les systèmes de gestion de base de données existants, il est généraleme difficile, voire impossible, de modifier la structure de la base en cours d'exploitation. Dans un système clefs en main, il est rare que la structure ait à être modifiée durant l'exploitation, mais on ne peut cependant exclure cette éventualité. Il serait donc souhaitable que MAGE permette des modifications de structure sur les systèmes en cours d'exploitation, sans que la réorganisation de la base soit d'une durée excessive.

Par ailleurs, la structure peut limiter le nombre d'enregistrements d'un fichier. Il serait souhaitable que cette limite puisse être éventuellement dépassée, avec un fonctionnement dégradé (par exemple, un niveau d'indirection supplémentaire pour l'accès aux données).

Enfin, si des opérations de reconfiguration de la base sont nécessaires pour maintenir des performances correctes, ces opérations devront être rapides et nécessiter peu de matériel (bandes, disques, ..). La possibilité d'effectuer des reconfigurations partielles (en cas d'urgence) peut être intéressan

CHAPITRE III

STRUCTURE ET ACCÈS AUX DONNÉES

III.1. MÉTHODOLOGIE

Il existe a priori trois manières de définir la fonction que doit exécuter un processeur spécialisé :

. Prendre un sous-système existant :
cette méthode est idéale s'il existe un sous-système connu pour répondre parfaitement aux besoins des applications et aux contraintes d'implémentation. On peut alors se dispenser de définir des langages et des algorithmes d'interprétation. La plus grande part de l'analyse étant déjà faite, il ne reste plus qu'à définir le matériel et à le réaliser. En outre, si l'on dispose du sous-système, il sera possible d'effectuer des mesures, éventuellement en conditions réelles, pour faciliter la conception d'un matériel optimum.

. S'inspirer d'un sous-système existant :
à partir d'un sous-système connu, il est possible de définir une fonction similaire, mieux adaptée aux besoins et aux contraintes, en s'inspirant de ce sous-système. Ceci peut nécessiter quelques évaluations, mais permet d'obtenir une fonction bien adaptée avec un travail relativement modeste. En outre, il est aussi possible de s'inspirer des algorithmes et éventuellement d'effectuer sur le sous-système existant des évaluations et des mesures utiles à la conception du processeur spécialisé.

. Concevoir entièrement un nouveau sous-système :
concevoir de toutes pièces un système original est certainement la solution la plus satisfaisante. Elle permet en particulier de corriger les défauts des systèmes existants et de s'intéresser aux recherches les plus avancées dans le domaine concerné, telles que les bases de données relationnelles par exemple. Cependant, cela représente un investissement considérable pour la seule définition de la fonction.

La participation de spécialistes dans ce domaine est indispensable et l'analyse du système doit être effectuée entièrement. Le système ainsi conçu ne sera de toute façon pas aussi au point qu'un système existant et "rôdé". Aucun moyen d'évaluation n'est disponible avant la réalisation d'un prototype et on ne peut pas être vraiment certain des performances obtenues avan cette réalisation. C'est pourquoi ce type de solution semble devoir être réservé à des fonctions très simples ou à des projets de grande ampleur.

Le projet MAGE visait avant tout à la réalisation matérielle d'un processeu de base de données, avec l'idée de déboucher sur l'industrialisation du résultat. Nous n'avions que très peu de moyens à consacrer à la définition de la fonction et il était hors de question de définir entièrement un système de gestion de bases de données original. Par ailleurs, aucun système connu de nous n'était parfaitement adapté à nos besoins. La deuxième méthode restait donc seule possible.

A l'époque de notre choix, il n'existait pas, à notre connaissance, de système de bases de données relationnelles opérationnel ; nous devions donc en rester aux bases de données hiérarchiques. Nous avons donc choisi de nous inspirer de SOCRATE [AB], [AC], système que nous connaissions bien, disponible sur les machines du CICG et qui convenait à l'application projetée.

Prenant pour point de départ le langage de définition de structure et le langage d'accès de SOCRATE, il s'agissait d'en faire une étude critique pour voir quels concepts devaient être modifiés, ajoutés ou supprimés pour satisfaire aux spécifications de MAGE.

Une bonne méthode d'analyse nous a semblé être celle de prendre une application type et de voir comment SOCRATE pouvait répondre à ses besoins.

Nous avons choisi comme application type le système SCRIB, mentionné au § II.1, qui est un des systèmes les plus complexes de la SAGEM. Ce système de gestion de Laboratoire d'analyses médicales permet :

. à l'accueil du malade, de saisir diverses informations : identité, analyses demandées, ...
. d'éditer la liste des prélèvements à effectuer, des étiquettes pour les identifier, les consignes pour le personnel du Laboratoire,
. de saisir les résultats, soit manuellement, soit sur appareil automatique,
. d'éditer les résultats et les expédier aux personnes concernées,
. d'éditer les factures et tenir la comptabilité du Laboratoire.

Des informations statiques telles que les descriptions des analyses, sont contenues dans les fichiers dits "de référence" qui seront donc consultés fréquemment, mais rarement modifiés. Les informations concernant les malades, les facturations, etc .., seront contenues dans les fichiers "temps réel" qui seront l'objet de modifications fréquentes : écriture ou création d'enregistrements. Un certain nombre d'informations dans les fichiers de référence sont identifiées par un numéro de code.

Nous nous sommes attachés, dans un premier temps, à décrire dans le langage SOCRATE la structure des fichiers de SCRIB. Ce travail a conduit à la définition du langage de définition de structure de MAGE, qui est une version simplifiée de celui de SOCRATE.

Dans une seconde étape, nous avons écrit en SOCRATE un certain nombre de programmes de SCRIB, cet ensemble de programmes permettant l'accès à toute la structure et regroupant toutes les difficultés du système au niveau des accès aux fichiers. Ces programmes ont permis de définir les spécifications du langage d'accès et, dans une étape ultérieure de travail, ils ont permis d'estimer la fréquence de certaines opérations et d'obtenir des informations précieuses.

III.2. LANGAGE DE DÉFINITION DE STRUCTURE

Le langage de définition de structure permet de décrire l'organisation des données et les relations existant entre les différents éléments d'informatio

Il existe divers types d'éléments pour permettre d'exprimer cette organisation ou ces relations. Il est apparu que tous les concepts existant dans SOCRATE ne sont pas utiles dans SCRIB. Pour d'autres concepts, la définition proposée par SOCRATE a paru trop contraignante, ou trop complexe à implémenter. Il a donc fallu simplifier le langage en supprimant certains concepts ou en en simplifiant d'autres. Ces concessions peuvent être au détriment de la facilité d'utilisation, ou être contraires à certaines règles de sécurité couramment admises, mais l'utilisation par des systèmes clefs-en-main semblait les permettre et la facilité qu'elles apportaient constitue elle aussi un facteur de sécurité important.

III.2.1. Entités

Un système de gestion de base de données consiste généralement à représenter un certain nombre d'objets entre lesquels des relations existent. Chaque objet est représenté par un certain nombre d'informations qui le caractérisent et définissent ses relations avec d'autres objets. Les objets sont répartis en diverses catégories de même nature, représentées par des informations de même type. Dans un système de gestion de fichiers classique, chaque catégorie d'objets est représentée par un fichier dont chaque enregistrement est représentatif d'un objet. Dans SOCRATE, une catégorie d'objets est représentée par une *entité*, c'est-à-dire une classe d'éléments d'information structurée de la même façon. Pour chaque objet représenté dans le système, existe une *réalisation* de l'entité, c'est-à-dire un bloc de données effectivement présent sur la mémoire secondaire.

Chaque réalisation est à son tour composée d'un certain nombre d'éléments appelés *caractéristiques*. Une caractéristique peut être une *caractéristique simple*, c'est-à-dire une zone de données de longueur fixe, individuellement accessible, mais non décomposable par le système (c'est-à-dire qu'on ne peut l'accéder qu'en totalité). Elle peut être une autre *entité*, ce qui permet de définir une structure arborescente. Elle peut être une *référence* qui est un pointeur sur une autre réalisation d'entité. Enfin, on peut définir des *caractéristiques "clés"* qui permettent un accès associatif accéléré à des réalisations d'une entité.

Une entité est donc caractérisée par :

- son identificateur, qui est le nom de la classe d'objets,
- son nombre maximum de réalisations, généralement indispensable pour permettre de déterminer les adresses des réalisations,
- ses caractéristiques.

Syntaxe :

<déclaration d'entité> ::= ENTITE <nr> <id> ;
DEBUT ; <liste de caractéristiques> FIN ;

<id> est un identificateur,
<nr> est un entier qui représente le nombre maximum de réalisations de l'entité dans une réalisation de l'entité englobante,
<liste de caractéristiques> ::= <caractéristiques>|<liste de caractéristiques>
<caractéristique> ::= <déclaration d'entité>|<déclaration de CS>|
<déclaration de référence>|<déclaration d'anneau>|
<déclaration de bloc>|<déclaration de clé>|
<déclaration d'index>

III.2.2. Caractéristiques simples

Dans SOCRATE, les caractéristiques peuvent être de types divers : chaînes de caractères, textes, entiers, listes de valeurs, ...

Il ne nous a pas paru utile de conserver cette distinction dans MAGE. En effet, la notion de type implique d'effectuer un certain nombre de contrôles et de conversions au niveau de la méthode d'accès, ce qui implique un traitement différent pour chaque type de données et donc complique beaucoup les programmes. Si on ne définit pas une très grande variété de types, on ne peut pas être certain que n'importe quelle information puisse être représentée de façon simple. Enfin, il existe des cas où le type d'un élément n'est pas défini lors de la définition de structure. SOCRATE, pour ce cas, prévoit des caractéristiques conditionnelles, mais l'implémentation n'en est pas simple.

Il nous a paru plus sage que MAGE ignore la notion de type des informations, l'utilisateur étant libre de contrôler lui-même le type des données et d'effectuer les conversions nécessaires.

Une caractéristique simple (notée CS) sera donc une chaîne d'octets indivisible (c'est-à-dire que MAGE ne permet pas d'en accéder une partie seulement), caractérisée par sa longueur et son identificateur.

Syntaxe :

<déclaration de CS> ::= CS <id> <dim> ;
<dim> est un entier qui représente la longueur de la caractéristique.

III.2.3. Blocs

Dans SOCRATE, un bloc est un groupe de caractéristiques de toute nature. Une telle généralité ne paraît pas indispensable et elle complique relativement l'implémentation.

Il nous a paru plus simple et guère moins commode, de définir un bloc comme étant simplement une suite de caractéristiques simples. La notion de bloc permet donc une manipulation globale d'un groupe de caractéristiques.

Syntaxe :

<déclaration de bloc> ::= BLOC <id> ;
DEBUT ; <liste de CS> ; FIN ;
<liste de CS> ::= <déclaration de CS> <liste de CS> |

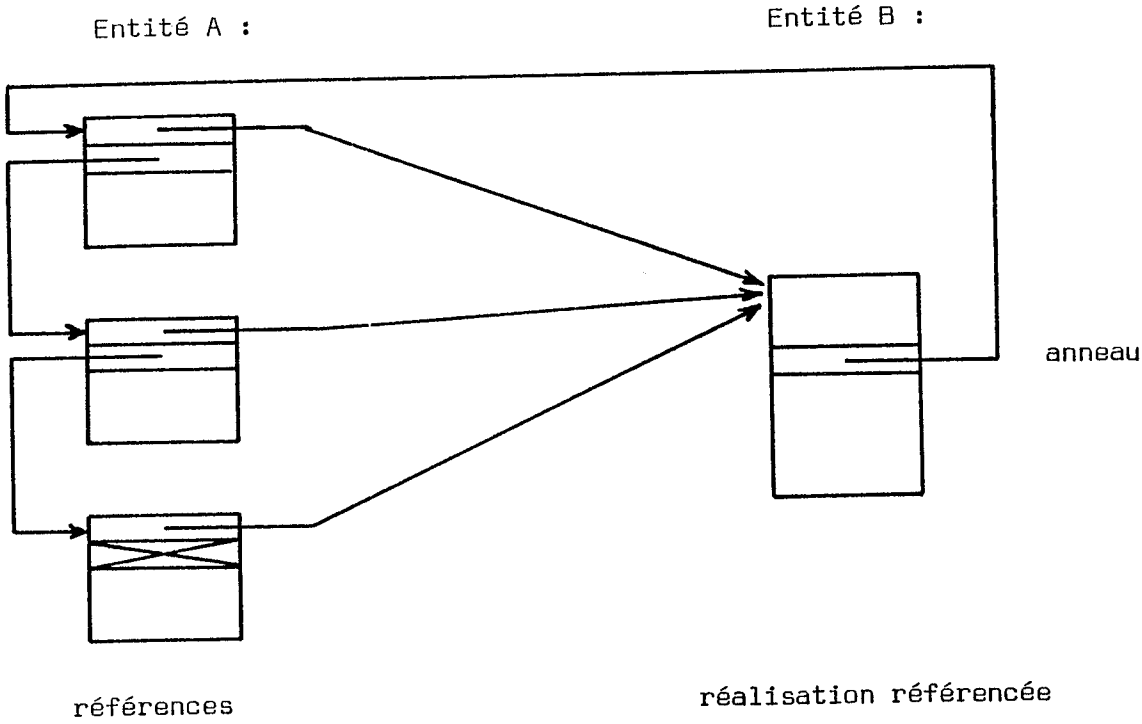
III.2.4. Pointeurs : références et anneaux

Une référence est un pointeur permettant d'accéder à une réalisation d'entité à partir d'une autre réalisation.

Si on supprime la réalisation référencée, la référence ne correspondra plus à une réalisation existante. Il faut donc disposer d'un moyen pour connaître toutes les références qui pointent sur une réalisation donnée, afin de pouvoir les remettre à zéro en cas de suppression.

Pour cela, on peut chaîner entre elles les références à une même réalisation et mettre la tête de liste dans la réalisation référencée : on appelle *anneau* cette tête de liste. A toute référence on associe donc de façon biunivoque un anneau.

On aura donc la structure suivante :



Dans certaines réalisations de SOCRATE, l'anneau est déclaré implicitement à la compilation de structure ; l'utilisateur n'a donc aucune conscience de son existence. Dans d'autres implémentations, l'anneau doit être explicitement déclaré par l'utilisateur qui a ensuite la possibilité d'utiliser tous les chaînages et non le seul chaînage "entité référençante vers entité référencée". A la référence s'ajoute donc un système de liste chaînée gérée automatiquement et on a, dans SCRIB, rencontré un cas où cette structure pouvait être pleinement utilisée. On représente donc **ainsi** par une seule référence une structure de données qui nécessiterait trois références si l'anneau n'était pas accessible. Nous avons donc choisi la déclaration explicite de l'anneau, bien qu'elle ne simplifie que très peu le compilateur de structure.

Syntaxe :

<déclaration d'anneau> ::= ANNEAU <id> ;
<déclaration de référence> ::= REF <id> SUR <id2> ;
<id2> est l'identificateur de l'anneau associé à la référence.

Cet anneau peut être placé aussi bien dans la même entité que la référence, que dans une autre entité.

III.2.5. Caractéristiques clés et index

Dans un système de gestion de base de données tel que SOCRATE, l'accès direct est l'accès privilégié, c'est-à-dire que la manière la plus simple et la plus rapide d'adresser une réalisation d'entité est de fournir son numéro de réalisation, c'est-à-dire son rang dans l'ensemble des réalisations, existantes ou non, de l'entité.

On a souvent besoin d'accéder à une réalisation dont on ne connaît que la valeur d'une caractéristique. On peut réaliser cet accès en effectuant une recherche séquentielle sur toutes les réalisations. Ce peut être très long et coûteux, surtout si ce genre d'accès est fréquent. C'est pourquoi SOCRATE propose un accès associatif "rapide".

Si une caractéristique est souvent utilisée pour accéder associativement à une entité, on peut la déclarer dans la structure comme "caractéristique clé". Un mécanisme de "hash-coding" sera alors associé à cette caractéristique pour accélérer l'accès associatif.

Pour cela une table est associée à l'entité. Un calcul effectué sur la valeur de la caractéristique clé, fournit le numéro d'une entrée de la table. Cette entrée contient la tête d'une liste chaînée de toutes les réalisations dont la caractéristique clé fournit le même résultat par le calcul de hash-coding. La valeur de la caractéristique clé est donc recherchée séquentiellement dans cette liste et on doit trouver de cette façon la réalisation recherchée.

Dans SOCRATE, la table de hash-coding est déclarée implicitement par le seul fait de déclarer une caractéristique clé. Par ailleurs, l'utilisateur fournit lui-même au système des fonctions de hash-coding, afin que celles-ci correspondent le mieux possible à ses besoins.

Il est difficile dans MAGE d'admettre que l'utilisateur puisse définir des programmes s'exécutant dans le PBD. Par ailleurs, des programmes standard peuvent ne pas convenir à certaines applications. Le plus simple est donc que l'utilisateur fournisse lui-même la valeur de la fonction de hash-coding, en même temps que la valeur de la caractéristique.

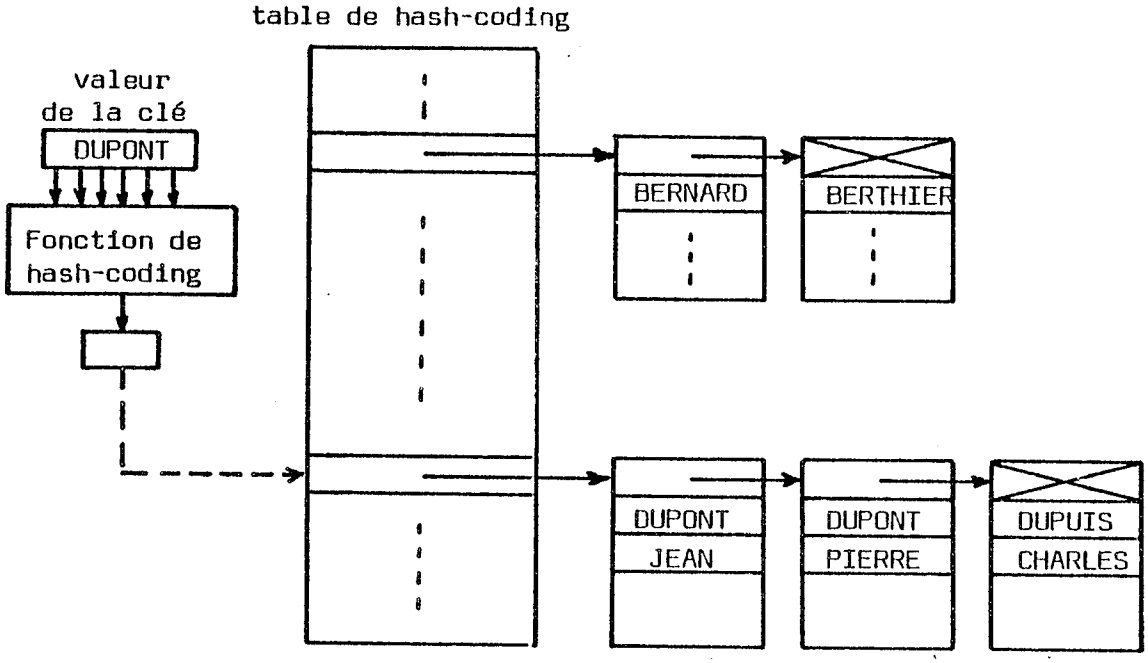
Par ailleurs, dans le but de simplifier les algorithmes d'accès, il a paru préférable que l'utilisateur soit conscient de l'existence de la table de hash-coding et donc qu'il la déclare lui-même dans la structure. Nous appelons cette table *index*.

Exemple :

```
ENTITE 500 VILLE ;
DEBUT ;
   INDEX PERSONNE 50 SUR NOM ;
   CS NOMVILLE 20 ;
   ENTITE 100000 HABITANT ;
   DEBUT ;
      CLE NOM 20 ;
      CS ADRESSE 80 ;
   FIN ;
FIN ;
```

PERSONNE définit une table de 50 entrées pour chaque ville, permettant d'accéder à un habitant d'une ville donnée, connaissant son nom. Si on ne connaît pas la ville, il faudra tenter l'accès sur chaque ville, jusqu'à ce que le nom cherché soit trouvé.

Structure de l'index :



L'utilisateur disposera donc d'une commande lui permettant d'insérer une réalisation dans la chaîne correspondant à une entrée donnée de la table (en fournissant un pointeur sur la réalisation et le numéro de l'entrée), d'une commande permettant d'accéder à une réalisation, en adressant une entrée de la table et en fournissant la valeur de la caractéristique, et d'une commande permettant de poursuivre la recherche si la réalisation trouvée n'est pas la bonne.

Syntaxe :

<déclaration d'index> ::= INDEX <id> <nr> SUR <id2> ;
<nr> désigne le nombre total d'entrées de la table
<id2> désigne l'identificateur de la clé
<déclaration de clé> ::= CLE <id> <dim> ;
<dim> est la longueur de la clé en octets.

III.3. ACCÈS AUX DONNÉES

Pour accéder une donnée, il faut spécifier le type d'accès que l'on veut effectuer (lecture, écriture, ..) et désigner la donnée.

Une donnée peut être désignée par la spécification du chemin qu'il faut parcourir dans la structure pour l'accéder, c'est-à-dire la spécification des réalisations d'entités qui la contiennent (langage procédural).

Le principal problème est donc la désignation d'une réalisation particulière d'une entité. Plusieurs mécanismes existent pour cela.

III.3.1. Désignation d'une réalisation

a) Accès direct :

On fournit le numéro de la réalisation désirée, c'est-à-dire son rang dans l'ordre des réalisations possibles (existantes ou non). Ceci permet donc un accès direct, la clé étant le numéro de réalisation. Ce type d'accès constitue l'accès privilégié chaque fois qu'il est possible d'identifier une donnée par un numéro.

b) Accès séquentiel :

On peut accéder successivement à toutes les réalisations d'une entité dans l'ordre de leurs numéros. Ce type d'accès nécessite un pointeur définissant la réalisation en cours d'accès.

c) Accès associatif :

Nous avons déjà vu que l'on peut, à l'aide des caractéristiques clés, accéder à une réalisation en connaissant la valeur d'une de ses caractéristiques ; nous parlerons alors d'accès séquentiel indexé. L'utilisateur devra adresser non l'entité elle-même, mais l'index, en fournissant la clé de hash-coding et la valeur de la caractéristique. Si la caractéristique n'est pas déclarée comme clé, il faudra procéder par recherche séquentielle.

d) Accès par références et anneaux :

On peut accéder à une réalisation par l'intermédiaire d'une référence ; le fait d'adresser la référence entraîne automatiquement l'accès à la réalisation référencée. On doit pouvoir également remonter à la réalisation référençante à partir de l'anneau, ou parcourir la chaîne des références à une même réalisation.

III.3.2. Mode d'accès

Une fois spécifiée la donnée, on peut effectuer dessus un certain nombre d'opérations :

. créer ou supprimer une réalisation d'entité ;
. lire ou écrire une caractéristique, ou toutes les caractéristiques simples d'une entité ;
. lire une référence pour accéder à la réalisation référencée, ou parcourir la chaîne des références à une même réalisation ;
. lire un anneau pour accéder à la réalisation référençante ;
. écrire une référence pour la faire pointer sur une réalisation donnée, ou l'insérer dans une chaîne donnée ;
. créer une entrée d'une table d'index, ou la supprimer.

III.3.3. Principes du langage d'accès

Dans ce paragraphe, nous allons montrer assez brièvement comment peut être conçu le langage d'accès, sans rentrer dans les détails qui ne pourront être fixés que dans une étape ultérieure du projet.

Dans SOCRATE, une requête peut avoir la forme suivante :
RESULTAT 100 de EXAMEN 20 de un MALADE ayant NOM = 'DUPONT'.
Cette requête demande l'impression du 100ème résultat du 20ème examen d'un malade dont le nom est DUPONT.

On distingue deux éléments dans le langage :
. la citation, qui désigne la donnée sur laquelle va être effectué l'accès,
. la requête qui spécifie l'action à effectuer : ici c'est la lettre clé I.

Les citations sont de longueur variable (dépendant de l'emplacement de la donnée dans la structure) et on peut trouver plusieurs types d'adressage dans une même requête : dans l'exemple ci-dessus, RESULTAT et EXAMEN sont accédés directement, alors que MALADE est accédé associativement.

Par ailleurs, des données figées à l'écriture des programmes (les identificateurs de la structure), se trouvent mêlées à des données paramétrables (numéros de réalisation ou valeurs associatives). Ceci pose un problème pour la transmission des paramètres entre le processeur utilisateur et MAGE.

On est amené, pour réduire la longueur des citations, ou permettre les accès séquentiels, à introduire une notion de contexte ou de pointeur : on peut définir un point de la structure à partir duquel se feront un certain nombre d'accès (mise en facteur du début de la citation). Si dans l'exemple précédent on avait positionné le contexte sur le malade DUPONT, on aurait pu écrire :
I RESULTAT 100 DE EXAMEN 20
ce qui est particulièrement avantageux si plusieurs accès au même MALADE sont effectués. On peut de même positionner un pointeur X (point courant) sur le 20ème EXAMEN DE DUPONT et écrire :
I RESULTAT 100 de X.

Un tel langage semble difficile à interpréter par un processeur décentralisé. La longueur variable des requêtes complique leur transmission entre le processeur utilisateur et MAGE. La mémorisation dans MAGE des requêtes en attente ou en cours de traitement peut être coûteuse en mémoire. L'interprétation des requêtes sera complexe et entraînera la génération d'un processus particulier pour chaque niveau de structure parcouru. Il s'en suit une complexité accrue de la gestion des requêtes en cours de traitement.

Un langage aussi sophistiqué que SOCRATE ne semble pas nécessaire pour MAGE qui n'est utilisé que par des programmes. Il semble donc souhaitable de se contenter d'un langage de primitives beaucoup plus simple.

Principes généraux du langage :

Nous avons vu que la notion de contexte de l'utilisateur était nécessaire pour faciliter certains accès. On associera donc à tout utilisateur un ou plusieurs contextes qui désigneront un élément de la base.

Un certain nombre de primitives permettront à l'utilisateur de déplacer son contexte dans la structure : descendre d'un niveau pour accéder à une fille de la caractéristique adressée, remonter à un niveau précédemment adressé, accéder à une soeur de la caractéristique adressée, initialiser un accès séquentiel et le poursuivre, etc .. L'accès associatif se fait en adressant l'index, en spécifiant la clé de hash-coding et la valeur de la caractéristique clé.

Ces opérations permettent donc d'adresser une donnée, l'accès à effectuer sur cette donnée étant spécifié par un paramètre, *le mode* (lire, écrire, créer, ..).

Dans ce langage, l'exemple précédent pourrait se traduire par les trois requêtes suivantes :

. adresser le malade ayant nom = 'DUPONT' (c'est-à-dire adresser l'index associé à l'entité MALADE),
. adresser l'examen 20,
. adresser le résultat 100 et le lire.

Une présentation plus complète du langage sera donnée dans un prochain chapitre.

CHAPITRE IV

ETUDE DE L'ADRESSAGE

IV.1. INTRODUCTION

L'utilisateur adresse les données par l'intermédiaire de la structure et de divers mécanismes de désignation.

Au niveau du disque, les données ne peuvent être accédées que par l'intermédiaire du mécanisme d'adressage physique du périphérique.

Il est donc nécessaire d'établir une relation entre ces deux adressages. De la solution adoptée dépendront dans une large mesure le langage, la structure et les performances du processeur MAGE.

Les contraintes principales imposées à la méthode d'adressage sont :
. permettre un accès aussi rapide que possible,
. permettre une utilisation correcte de l'espace mémoire,
. ne pas limiter les possibilités du langage utilisateur,
. permettre une implémentation simple sur un matériel peu puissant.

Il est évident qu'un compromis satisfaisant entre ces contraintes est à rechercher.

Les trois dernières peuvent être traitées de façon très générale, mais le fait que le processeur MAGE soit très proche du disque, va permettre de rechercher des solutions particulières pour la rapidité : une optimisation des déplacements de bras sera possible et il sera intéressant d'organiser la base de façon à rendre cette optimisation la plus efficace possible.

IV.2. SOLUTIONS EXISTANTES

L'utilisation correcte de la mémoire implique qu'une donnée ne puisse occuper de la place que si elle est effectivement existante.

Considérons l'exemple suivant :

```
entité 10000 PERSONNE ;
début ;
   CS NOM 20 ;
   CS ADRESSE 80 ;
   entité 5 VOITURE ;
   début ;
      CS MARQUE 20 ;
      CS NUMERO 20 ;
   fin ;
fin ;
```

On définit 10000 personnes pouvant avoir chacune jusqu'à 5 voitures ; le nombre maximum de voitures est donc égal à 50000. Or, sur les 10000 personnes le nombre moyen de voitures par personne peut être de 0,5. On aura donc un nombre effectif de voitures égal à 5000 et le taux de remplissage de l'entité sera de 10 %. On peut améliorer ce taux en diminuant le nombre maximum de voitures par personne, mais on ne peut plus alors mémoriser toutes les voitures des personnes qui en possèdent 5.

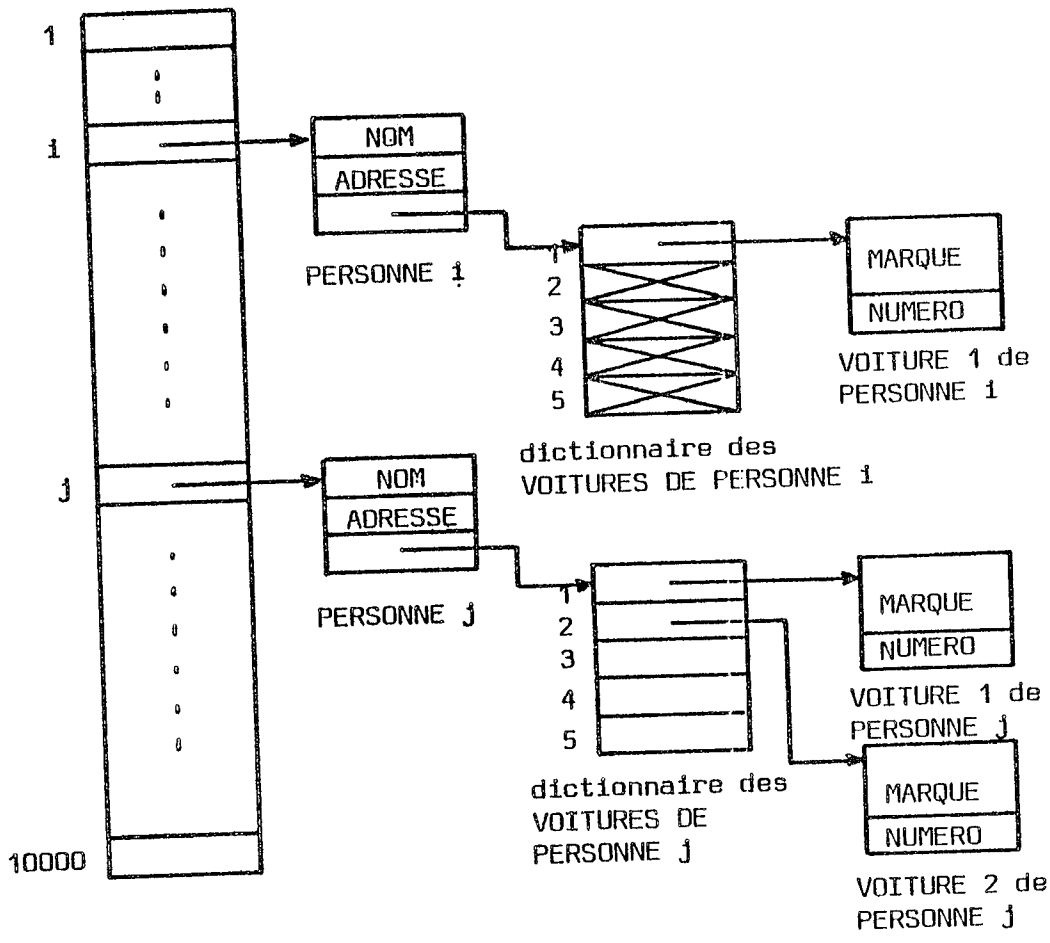
Il est hors de question de réserver la place pour toutes les voitures possibles ; par ailleurs, réserver statiquement de la place pour un certain nombre de réalisations de chaque entité, peut conduire à une mauvaise utilisation de l'espace, au risque de voir fréquemment une entité saturée alors qu'il reste beaucoup de place disponible pour les autres.

Ces considérations imposent donc une gestion banalisée et dynamique de l'espace. Deux solutions ont été étudiées dans SOCRATE.

IV.2.1. Adressage par descripteurs

Les données sont chaînées entre elles suivant la structure. Une entité est représentée par un dictionnaire qui contient la liste des adresses des caractéristiques de chaque réalisation.

La structure de l'exemple précédent sera représentée de la façon suivante :



L'accès aux données se fait donc en parcourant les chaînages et en recherchant les réalisations dans les dictionnaires.

Cette technique nécessite un grand nombre de pointeurs. Une partie importante de la mémoire peut être occupée par les descripteurs et l'accès à une donnée nécessite de multiples indirections pouvant entraîner autant d'accès disque. Il semble difficile d'optimiser les accès tant en nombre qu'en durée, les données étant réparties sur le disque selon le hasard des allocations mémoire

Enfin, la multiplicité des pointeurs (sous forme d'adressage physique) diminue grandement la sécurité des informations et la fiabilité du logiciel, et elle complique beaucoup les éventuelles réorganisations de l'espace ou modifications de structure.

Cette technique a été écartée par les concepteurs de SOCRATE et il ne nous paraît pas nécessaire de mettre ce choix en question.

IV.2.2. Adressage virtuel

Dans SOCRATE, on considère une énorme mémoire abstraite ("virtuelle") dans laquelle on peut allouer statiquement de la place à tout élément d'information pouvant exister, lors de la compilation de la structure. Il est ensuite possible de calculer l'adresse virtuelle de chaque information à partir des numéros de réalisation des entités qui la contiennent et de renseignements contenus dans un fichier qui représente la structure.

L'espace virtuel est découpé en blocs de longueur fixe (par exemple 64 octets) appelés sous-pages. Lorsqu'une sous-page contient une donnée effectivement définie et non nulle, elle est placée dans le fichier de la base de données en utilisant une technique de hash-coding sur les adresses virtuelles.

Pour accéder à une information, on doit donc :

. calculer son adresse virtuelle,

. déterminer l'adresse virtuelle de la sous-page,

. obtenir la sous-page par hash-coding,

. extraire la donnée recherchée de la sous-page.

Cette technique donne satisfaction dans SOCRATE, mais on peut lui faire quelques reproches :

. Il existe une perte de place par fragmentation non négligeable.

. L'accès à une adresse virtuelle peut entraîner plusieurs accès disque pour parcourir les listes de synonymes créées par le hash-coding ; en outre, plusieurs accès à l'espace virtuel sont nécessaires pour accéder une donnée. En effet, l'existence d'une réalisation est définie par une chaîne de bits, elle-même située dans l'espace virtuel.

. Des données contigües dans l'espace virtuel peuvent ne pas l'être sur le disque du fait du hash-coding : plusieurs accès à l'espace virtuel seront alors nécessaires pour obtenir ces données.

. La répartition des données sur le disque rend difficile toute optimisation des déplacements de bras.

. Enfin, l'implémentation de cette technique conduit à des programmes complex

Dans SOCRATE, le nombre d'accès à la mémoire secondaire est diminué par l'utilisation d'un cache ; cependant, dans le cas qui nous préoccupe, l'efficacité de ce cache est fortement compromise par plusieurs facteurs :

. Les programmes étudiés ont montré qu'il n'y avait que très peu de localité au niveau de l'accès aux données par un utilisateur. Ceci peut sans doute être compensé par le fait que la base est partagée entre plusieurs utilisateurs. Or, le nombre de ceux-ci est relativement réduit : dans un système SCRIB moyen, on a une dizaine de terminaux dont certains sont très spécialisés. On peut alors avoir une très forte charge du système avec très peu de terminaux actifs (3 ou 4). Le gain que peut apporter une mémoire cache avec un tel degré de multiprogrammation est vraiment douteux.

. Une mémoire cache nécessite un espace mémoire considérable pour être d'une efficacité acceptable (surtout dans les conditions mentionnées ci-dessus). Or, la mémoire coûte encore cher et il sera souhaitable de l'économiser.

Les critiques faites à SOCRATE nous ont donc amenés à rechercher d'autres solutions.

IV.3. ADRESSAGE PAR NOMS INTERNES

Au lieu d'affecter statiquement une adresse à toute information pouvant exister, on peut associer un numéro à toute réalisation d'entité pouvant exister. Nous appellerons *nom interne* ce numéro.

Les noms internes peuvent être affectés comme suit (cf. exemple précédent) lors de la compilation de structure :
PERSONNE : de 1 à 10 000
VOITURE : de 10 001 à 60 000

On peut calculer très facilement le numéro d'une réalisation donnée en connaissant le numéro de la première réalisation.

Exemple :

VOITURE 2 de PERSONNE 100

nombre de réalisations de voiture précédentes : 99 x 5 + 2 = 497
nom interne : 497 + 10 001 = 10 498.

Le lien entre le nom interne et la donnée, si elle existe, est assuré par un dictionnaire. Celui-ci contient, pour chaque nom interne défini, l'adresse disque du bloc de données correspondant.

Il est évident que si on réserve une entrée dans le dictionnaire pour toute réalisation pouvant exister, sa taille sera excessivement grande. Il est donc nécessaire de recourir à une technique de repliment du dictionnaire. Celui-ci pourra alors être accédé par hash-coding sur les noms internes.

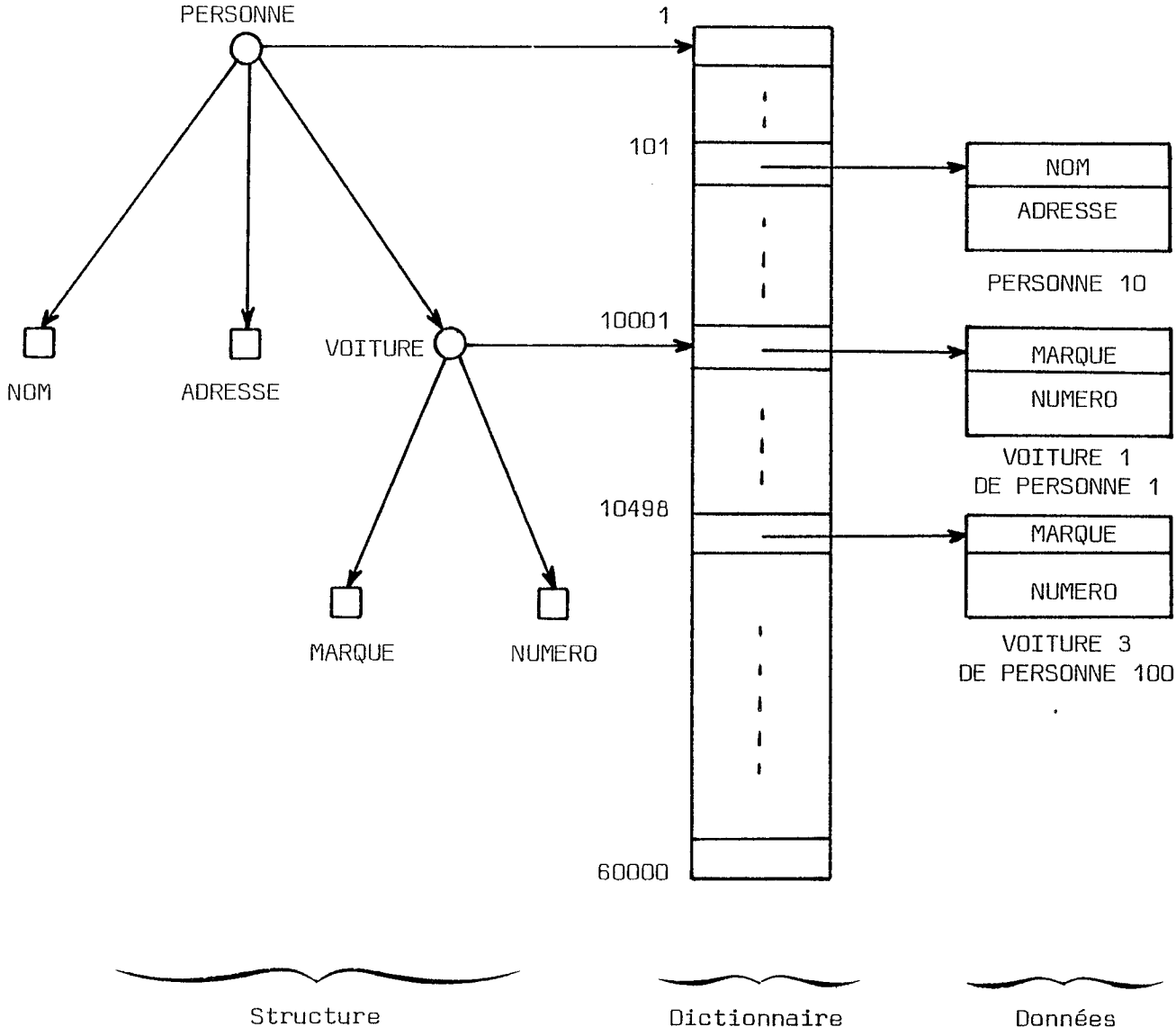
On a donc trois types d'information :

. *La structure* dont la taille sera assez réduite. Elle contient d'une part les informations nécessaires pour calculer les noms internes et d'autre part la délimitation des caractéristiques dans les blocs de données, les longueurs, etc .. Elle est créée par le compilateur de structure à partir des renseignements fournis par l'utilisateur à l'aide du langage de définition de structure (cf. chapitre III).

. *Le dictionnaire* qui permet de passer du nom interne d'une réalisation existante à l'adresse disque du bloc de données correspondant.

. *Les données* proprement dites.

Exemple :


PERSONNE
NOM
ADRESSE
VOITURE
MARQUE
NUMERO
1
101
10001
10498
60000
NOM
ADRESSE
PERSONNE 10
MARQUE
NUMERO
VOITURE 1
DE PERSONNE 1
MARQUE
NUMERO
VOITURE 3
DE PERSONNE 100
Structure
Dictionnaire
Données

Cette méthode donne un nombre d'indirections très inférieur à celui nécessité par une méthode d'adressage par descripteur.

Les entrées du dictionnaire étant petites, il en tient un grand nombre dans un bloc physique et on pourra accéder à l'entrée cherchée en un faible nombre d'accès disque.

Le dictionnaire constitue une sorte de charnière entre l'espace des noms internes et l'espace réel et il sera sans doute possible d'exploiter cette propriété pour améliorer la sécurité et la souplesse du système.

Avant d'étudier plus profondément cette technique, nous allons tenter de comparer les performances que l'on peut en attendre avec celles de l'adressage virtuel de SOCRATE.

IV.4. COMPARAISON

Cinq critères de comparaison pourront être pris en compte :

- complexité des algorithmes,
- souplesse,
- sécurité,
- utilisation de la mémoire secondaire,
- rapidité.

IV.4.1. Complexité des algorithmes

Ce critère rentre en ligne de compte dans la mesure où la taille de la mémoire de MAGE et le coût de la conception en sont affectés de manière non négligeable.

Le système effectue beaucoup d'entrées-sorties, aussi la rapidité des algorithmes a-t-elle relativement peu d'importance ; elle doit cependant être suffisante pour permettre l'implémentation sur un matériel bon marché, donc peu rapide.

D'après l'étude effectuée pour le prototype de SOCRATE, la technique d'adressage virtuel utilise environ 1 500 instructions en langage évolué. L'adressage par noms internes ne devrait pas être, a priori, plus complexe.

IV.4.2. Souplesse

Du point de vue des possibilités du langage d'utilisation, les deux méthodes semblent donner satisfaction.

Un problème plus épineux concerne les possibilités de reconfiguration de l'espace disque et de modification de la structure.

Dans SOCRATE, les opérations de reconfiguration sont très longues et intéressent toujours l'ensemble de la base. Les possibilités de modification de structure sont très faibles ; en effet, la structure définit l'implantation des données dans l'espace virtuel, d'où découle l'implantation physique des données. Le plus souvent, une modification de structure, même minime, implique de recréer totalement la base de données. Les opérations de reconfiguration de la base ou de modification de structure, nécessitent en général un espace important en mémoire secondaire.

Dans MAGE, ces opérations semblent plus simples. On peut reconfigurer séparément le dictionnaire et les données ; la reconfiguration du dictionnaire permettant par exemple d'augmenter le nombre de réalisations que l'on peut gérer. Les modifications de structure n'entraînent pas forcément de modification des noms internes : la modification des caractéristiques d'une entité n'entraîne que la modification des réalisations existantes, l'insertion d'une nouvelle entité peut se faire en affectant les noms internes à partir du dernier nom interne affecté, la suppression d'une entité peut se faire en laissant subsister un trou dans l'espace des noms internes. Dans tous ces cas, on peut ne modifier que la partie de la base concernée par la modification de structure. Si une modification des noms internes intervient, on peut ne modifier que la partie de la base qui est concernée. Il n'y a donc que si une très grosse partie de la base est concernée, qu'une re-création devrait intervenir, mais ce cas devrait rester rare.

Il ne semble donc pas audacieux de dire qu'a priori l'adressage par noms internes semble supérieur à l'adressage virtuel, en ce qui concerne les possibilités de reconfiguration et de modification de structure.

IV.4.3. Sécurité

L'utilisation de SOCRATE pour la transcription de SCRIB montre une certaine vulnérabilité du fichier des données vis à vis des incidents du système. La structure assez complexe de l'espace des données peut expliquer cette fragilité.

L'adressage par noms internes comporte sans doute un certain nombre de points faibles, cependant il semble qu'en introduisant une redondance entre les données et le dictionnaire, on gagne beaucoup en sécurité. Si le nom interne de la réalisation est reproduit dans le bloc de données, l'espace des données permettra de détecter et éventuellement réparer une erreur dans le dictionnaire ; de même, le dictionnaire peut permettre de retrouver une information perdue dans les données. Une incohérence pourra aussi être détectée avant qu'elle n'entraîne une erreur grave.

IV.4.4. Utilisation de la mémoire secondaire

Nous négligerons dans ce paragraphe la place occupée par le fichier structure, puisqu'elle est très faible et équivalente dans les deux cas.

Nous négligeons également dans SOCRATE la perte de place entraînée par les chaînes de bits de présence et les nombres de réalisations des entités, car elle est relativement faible.

Les statistiques faites sur SCRIB ont montré que la taille moyenne d'une réalisation est d'environ 54 octets. On peut considérer que la longueur moyenne des séquences de réalisation est de l'ordre de 10, ce qui fait 540 octets. Si on fixe la taille des sous-pages à 64 octets, on perd 4 octets d'en-tête par sous-page, soit 6,25 % de l'espace.

Chaque séquence de réalisation peut commencer n'importe où dans une sous-page et se terminer n'importe où dans une autre sous-page ; on peut considérer que la place restant dans la première et la dernière sous-page est perdue ; on a donc en moyenne une sous-page de perdue par séquence, soit 60 octets (taille utile) soit :

$\frac{60 \times 100}{(540+60)}$ = 10 % de l'espace utile = 9,77 % de l'espace total.

La perte totale est donc de l'ordre de 16 %.

Les essais effectués sur SOCRATE ont montré une perte de place très importante. La perte de place par fragmentation peut être très importante si les données sont très dispersées : dans l'exemple précédent, c'est le cas de l'entité VOITURE, puisqu'on n'a en général qu'une seule voiture par personne et cette voiture peut occuper à elle seule une sous-page, voire deux si l'adressage la place à cheval sur deux sous-pages. Pour des sous-pages de 60 octets, le taux d'utilisation est donc très mauvais (33 % de perte au moins) ; on peut l'améliorer si on adopte une taille de sous-page inférieure, mais la perte de place par descripteur augmente alors.

Avec l'adressage par noms internes, nous avons pour chaque réalisation :
. 10 octets d'entrée dictionnaire,
. 4 octets de nom interne dans les données,
soit 14 octets de descripteurs par réalisation, soit encore :
$\frac{14 \times 100}{(54+14)} = 20,6$ % de l'espace.

La perte par fragmentation interne peut être évaluée comme suit (en supposant la mémoire gérée par blocs de 1 K) :
. perte de 0,5 réalisation par bloc, soit 29 octets (en moyenne),
. perte de 4 octets par bloc (en-tête de bloc),
soit 33 octets, soit encore $\frac{33 \times 100}{1024} = 3,2$ % de l'espace des données.

La perte totale est donc de l'ordre de 24 %. Un tel résultat n'est pas mauvais en soi, mais il est nettement défavorable par rapport aux 16 % estimés pour SOCRATE.

IV.4.5. Rapidité

La rapidité peut être évaluée en termes d'échanges disque. En effet, MAGE passera certainement plus de temps en entrées-sorties qu'en traitement et les programmes auront largement le temps de s'exécuter durant les entrées-sorties, même avec un processeur relativement lent.

Une première évaluation à faire est celle du nombre d'accès à la mémoire secondaire, mais il sera intéressant de la compléter en faisant intervenir le nombre et la longueur des déplacements de bras sur le disque : MAGE sera suffisamment proche du disque pour qu'une optimisation soit possible sur ce point.

IV.4.5.1. Nombre d'accès

Les accès au fichier structure devraient être les mêmes dans les deux cas. Nous n'en tiendrons donc pas compte.

L'adressage par noms internes nécessite des accès au dictionnaire et des accès aux données. L'accès aux données ne nécessite qu'un accès disque, mais l'accès au dictionnaire peut demander plusieurs accès disque du fait du hash-coding. Des mesures ont été faites, par la SAGEM, sur l'algorithme d'accès au dictionnaire, pour un taux de remplissage de 50 %. Ce taux de charge est justifié par le fait que, sur un système tel que SCRIB, les fichiers les plus importants sont périodiquement archivés sur bande magnétique, puis effacés. Le taux de remplissage du disque oscille donc entre une valeur très faible et 100 %, la moyenne se situant un peu au-dessus de 50 %. Pour ce taux de charge, le hash-coding se fait généralement en un seul accès disque, du fait de la faible taille des entrées dictionnaire, donc un grand nombre est contenu dans un bloc disque.

Sur SOCRATE, un accès à l'espace virtuel nécessite également plusieurs accès disque. Des mesures pour le taux de remplissage de 50 % ont montré que les chaînes de "squatters" générées par le hash-coding, s'étendent en moyenne sur 1,14 blocs disque. Un accès virtuel nécessite en moyenne le parcours d'une demi-chaîne, soit 1,07 accès réel.

. Accès en lecture à une réalisation :

Deux accès virtuels sont nécessaires dans SOCRATE :
. accès à la chaîne de bits d'existence de l'en-tête,
. accès à la réalisation.
Ce qui fait 2,14 accès disque.

L'accès par noms internes nécessite un accès au dictionnaire et un accès aux données, soit deux accès disque.

. Accès en écriture à une réalisation :

SOCRATE effectue trois accès :
. un accès à la chaîne de bits,
. une lecture,
. une écriture.
La ré-écriture se fait en un seul accès puisqu'on connaît déjà l'adresse de la sous-page modifiée.
Une écriture représente donc *3,14 accès disque*.

L'accès par noms internes nécessite un accès au dictionnaire et deux accès aux données, soit *3 accès disque*.

Notons que dans SOCRATE l'écriture d'une donnée indéfinie peut entraîner la création d'une sous-page.

. Création d'une réalisation :

SOCRATE effectue deux accès à la chaîne de bits : une lecture (1,07 accès disque), une ré-écriture (un accès disque). Il est rare que la création entraîne une création de sous-page pour la chaîne de bits.

Pour la réalisation, la création d'une ou plusieurs sous-pages sera nécessaire. Pour cela, il faut rechercher l'emplacement de la sous-page, ce qui peut entraîner un parcours de chaîne de squatters, éventuellement déplacer une sous-page qui occupe l'emplacement de la sous-page à créer, et ré-écrire la sous-page. Si plusieurs sous-pages doivent être créées, des accès disque supplémentaires peuvent être entraînés. On peut considérer qu'une création coûte en moyenne *4,21 accès disque*.

En ce qui concerne les noms internes, les mesures effectuées ont montré que la création d'une entrée dictionnaire peut coûter en moyenne 2,002 accès disque, que nous arrondirons à 2. Dans la plupart des cas, l'allocation mémoire se fera en un seul accès ; enfin, un accès est nécessaire pour la ré-écriture. Une création peut donc se faire en *4 accès disque*.

Ces estimations donnent donc un léger avantage à l'adressage par noms internes, mais trop faible pour être considéré comme déterminant, étant donnée la prudence nécessaire dans l'interprétation de ces résultats d'estimation.

L'utilisation d'un cache en mémoire centrale peut diminuer le nombre d'accès disque, cependant son efficacité pour un faible nombre d'utilisateurs est douteuse. Il semblerait que l'adressage par noms internes offre une possibilité plus intéressante, puisqu'un cache sur le dictionnaire est envisageable, même s'il ne concerne que 50 % des accès.

IV.4.5.2. Déplacements de bras

La durée d'une entrée-sortie sur disque à bras mobile se décompose en trois parties :

. le temps de positionnement du bras sur le cylindre adressé. Sur le disque CDC 9760, ce temps varie de 6 à 55 ms, le temps moyen étant de 30 ms ;

. le délai rotationnel, c'est-à-dire le temps qu'il faut attendre pour que le secteur adressé passe sous les têtes. Il est égal en moyenne à la moitié du temps de rotation du disque, soit 8,33 msec dans notre exemple ;

. le temps de transfert qui est beaucoup plus court. Pour 20 secteurs par piste, il est de un vingtième du temps de rotation, soit 0,8 ms.

On voit donc qu'une opération sur le disque CDC 9760 peut prendre en moyenne 30 + 8,33 + 0,8 = 39,13 ms, dont la plus grande part est due au déplacement du bras.

Le critère de nombre d'accès disque ne suffit donc pas pour exprimer la rapidité, puisque si une solution nécessite des déplacements de bras beaucoup plus longs qu'une autre, elle peut être plus lente, même si le nombre d'accès disque est inférieur.

Les déplacements de bras peuvent être optimisés en traitant les requêtes non dans l'ordre où elles arrivent, mais dans l'ordre des numéros de cylindre demandés : le bras se déplace alors d'un côté du disque à l'autre et sert les requêtes au fur et à mesure qu'il passe sur les cylindres correspondants [DE], [TP] (stratégie de l'ascenseur).

Le temps d'accès moyen sans optimisation est donné par la formule :

$$A1 = S\ min + \frac{(S\ max - S\ min)}{3} + \frac{T}{2} + \frac{T}{s}$$

Le temps d'accès moyen avec la stratégie de l'ascenseur est le suivant :

$$A2 = S\ min + \frac{(S\ max - S\ min)}{n + 1} + \frac{T}{2} + \frac{T}{s}$$

avec S min = durée minimum d'un déplacement de bras (6 ms pour 9760),

S max = durée maximum d'un déplacement de bras (55 ms pour 9760),

n = nombre de requêtes traitées en un balayage,

T = durée d'une rotation du disque (16,7 ms),

s = nombre de secteurs par piste (10 secteurs).

Le taux T/s représente la durée du transfert.

Le gain obtenu par l'application de la politique d'optimisation est exprimé par le rapport :

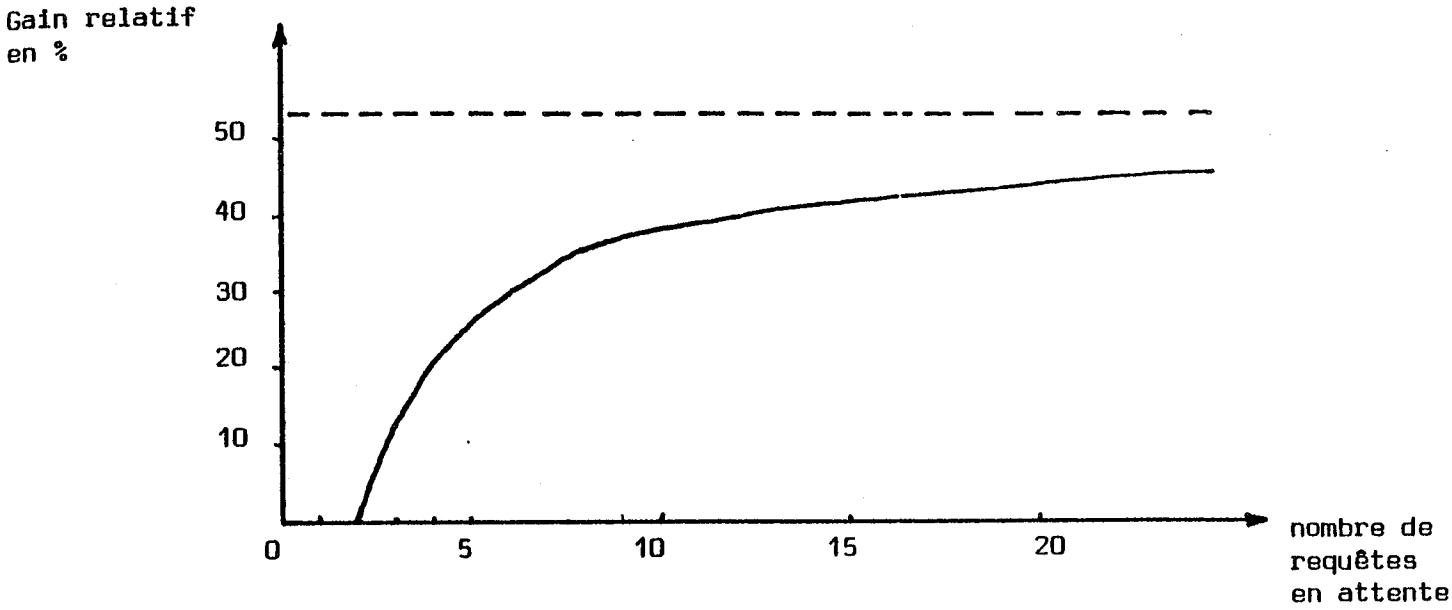
$$\frac{A2}{A1} = \frac{S\ min + \frac{S\ max - S\ min}{n + 1} + \frac{T}{2} + \frac{T}{s}}{S\ min + \frac{S\ max - S\ min}{3} + \frac{T}{2} + \frac{T}{s}}$$

ce qui donne, appliqué au 9760 :

$$\frac{A2}{A1} = \frac{14,7 + \frac{49}{n + 1}}{31}$$

Si une seule requête est en attente, on est ramené à la politique "premier entré, premier servi" et le rapport est 1.

n = 2	A2/A1 = 1	gain relatif = 0 %
n = 3	A2/A1 = 0,87	gain relatif = 13 %
n = 4	A2/A1 = 0,79	gain relatif = 21 %
n = 5	A2/A1 = 0,74	gain relatif = 26 %
n = 9	A2/A1 = 0,63	gain relatif = 37 %
n = 11	A2/A1 = 0,61	gain relatif = 39 %
$n \to \infty$	$A2/A1 \to 0,47$	gain relatif $\to$ 53 %



On constate qu'une amélioration notable apparaît dès que trois requêtes au moins sont traitées en un balayage ; une file d'attente de trois requêtes suffit donc pour justifier l'effort d'une optimisation.

Dans l'adressage par noms internes, il existe un certain ordre chronologique entre les accès aux trois fichiers : un accès à la structure induit souvent un accès au dictionnaire, qui à son tour induit un accès aux données. Cette propriété peut permettre d'augmenter le nombre d'accès effectués en un balayage du disque. On peut aussi adopter la stratégie suivante : les requêtes disque sont classées en trois files : données, structure, dictionnaire et on applique la stratégie de l'ascenseur sur chacune de ces zones en changeant de zone chaque fois que la file d'attente traitée est vide, ou qu'une autre est pleine. On ne risque pas ainsi de reporter indéfiniment le traitement d'une requête, puisque les requêtes sur structure, dictionnaire et données ne sont pas indépendantes.

Dans SOCRATE, les données sont réparties et on n'a pas de possibilité d'améliorer l'optimisation des déplacements de bras.

IV.5. CONCLUSION

L'étude précédente nous a donné les résultats suivants :

. pas de différence significative en ce qui concerne la complexité ;

. avantage des noms internes pour les modifications de structure ;

. avantage de SOCRATE pour l'occupation de la mémoire ;

. les évaluations effectuées montrent un léger avantage en rapidité pour l'adressage par noms internes, mais l'étude effectuée n'est pas assez précise pour être concluante.

Aucun avantage n'est décisif, cependant, les noms internes permettent d'envisager un certain nombre d'optimisations, telles que l'utilisation d'un support plus rapide pour le dictionnaire (disque à têtes fixes), ce qui augmenterait considérablement la rapidité (gain d'un rapport 4 sur 50 % des accès). La centralisation de l'adressage devrait offrir plus de possibilités de garantir la sécurité des informations. Enfin, un taux de remplissage élevé de l'espace alloué aux données peut être obtenu sans qu'il en résulte une baisse sensible de performances.

Ces critères nous ont donc conduit à approfondir l'étude de l'adressage par noms internes, qui peut être a priori retenu pour le processeur MAGE.

CHAPITRE V

MISE EN OEUVRE

DE L'ADRESSAGE PAR NOMS INTERNES

V.1. IMPLÉMENTATION DES DIFFÉRENTS ACCÈS

Quatre types principaux d'opérations peuvent être effectués sur les données

. création d'une réalisation,
. suppression d'une réalisation,
. lecture des données,
. modification des données.

La création implique l'introduction dans le dictionnaire d'une entrée correspondant au nom interne de la réalisation insérée. Le bloc de données doit être alloué et son adresse placée dans le dictionnaire.

Lors de la suppression, trois opérations sont nécessaires :

. mise à jour des chaînages concernant la donnée à effacer : références, anneaux, index,
. suppression de l'entrée dictionnaire,
. libération de l'espace des données.

Une réalisation peut être désignée de quatre façons différentes :

. accès direct (par numéro de réalisation),
. accès séquentiel,
. accès associatif,
. accès par références et anneaux.

V.1.1. Accès direct

Dans ce type d'accès, l'utilisateur fournit un numéro de réalisation : dans l'exemple du chapitre précédent, on peut désirer accéder à la personne 100, c'est-à-dire à la personne dont le nom interne est le 100ème des noms internes des personnes.

Ce type de désignation est indépendant du nombre de réalisations existantes et la réalisation désignée peut ne pas exister.

Les renseignements fournis par l'utilisateur permettent le calcul du nom interne, lequel permet ensuite l'accès à la donnée grâce au dictionnaire.

Exemple :

```
entité 200 MALADE ;
début ;
   CS MATRICULE 4 ;
   CS NOM 20 ;
   CS AGE 1 ;
   entité 20 EXAMEN ;
   début ;
      CS CODE 1 ;
      entité 100 RESULTAT ;
      début ;
         CS VALEUR 4 ;
         CS TEXTE 40 ;
      fin ;
   fin ;
   entité 7 PARAMETRE ;
   début ;
      CS VALEUR 4 ;
      CS NOM 10 ;
   fin ;
fin ;
```

L'attribution des noms internes par le compilateur de structure pourra être faite de la façon suivante :

MALADE de 1 à 200 (200 réalisations)
EXAMEN de 201 à 4 200 (20 x 200 = 4 000 réalisations)
RESULTAT de 4 201 à 404 200 (100 x 4 000 = 400 000 réalisations)
PARAMETRE de 404 201 à 405 600 (7 x 200 = 1 400 réalisations).

Supposons maintenant que l'on veuille accéder à :
RESULTAT 10 de EXAMEN 2 de MALADE 100.

Le nom interne sera obtenu en ajoutant au nom interne du premier résultat, soit 4 201, le nombre de résultats qui précèdent le résultat cherché :

. nombre de MALADE précédents = 99
  or, il y a 100 x 20 = 2 000 résultats par MALADE,
  donc : nombre de résultats avant RESULTAT 1 de EXAMEN 1 de MALADE 100 :
  99 x 2 000 = 198 000 résultats ;

. nombre d'EXAMEN précédents dans MALADE 100 = 1 soit 100 RESULTAT ;

. nombre de RESULTAT de EXAMEN 2 de MALADE 100 précédents = 9 résultats.

Il y a donc 198 000 + 100 + 9 = 198 109 résultats avant celui que l'on cherche.

Le nom interne cherché est donc :

198 109 + 4 201 = 202 310

Cherchons maintenant à définir un algorithme général de calcul. Nous appellerons entité englobante d'une entité donnée, l'entité dans laquelle elle est définie dans la structure. Par exemple, l'entité englobante de RESULTAT est EXAMEN, celle d'EXAMEN est MALADE et celle de MALADE n'existe pas.

Nous distinguons donc le numéro de réalisation (NR) fourni par l'utilisateur, du numéro absolu de réalisation (NAR) qui est le numéro de la réalisation dans l'ensemble de toutes les réalisations de l'entité dans toutes les réalisations des entités englobantes. Dans l'exemple précédent, ce numéro est 198 110 pour RESULTAT 10 de EXAMEN 2 de MALADE 100, alors que le numéro de réalisation est 10.

Nous désignons par NBR le nombre maximum de réalisations de l'entité (dans une réalisation de l'entité englobante) défini dans la structure.

Enfin, nous appellerons NARENG le numéro absolu de réalisation de l'entité englobante.

La formule générale de calcul est donc la suivante :

$$NAR = (NARENG - 1) \times NBR + NOR$$

$$\text{Nom interne} = NAR + NIPR$$

NIPR est le nom interne de la première réalisation ; il est fourni par la structure.

On doit calculer NAR pour chacun des niveaux précédents en commençant par le premier. Le résultat obtenu pour un niveau deviendra la valeur de NARENG utilisée pour le niveau suivant. La valeur initiale de NARENG pour le premier niveau est 1.

L'application à l'exemple précédent se fait de la façon suivante :

* accéder à l'enregistrement de structure de MALADE ;

NARENG = 1 ; NBR = 200 ; NR = 100 ;

NAR = (1 - 1) x 200 + 100 = 100 ;

* accéder à l'enregistrement de structure de EXAMEN ;

NARENG = NAR = 100 ; NBR = 20 ; NR = 2 ;

NAR = 99 x 20 + 2 = 1 982 ; (EXAMEN 2) ;

* accéder à l'enregistrement de structure de RESULTAT ;

NARENG = NAR = 1 982 ; NBR = 100 ; NR = 10 ;

NAR = 1 981 x 100 + 10 = 198 110 ; (RESULTAT 10) ;

* NIPR = 4 201 ; (premier nom interne de RESULTAT) ;

$$NI = 198\ 110 + 4\ 201 - 1 = 202\ 310 ;$$

On retrouve bien le résultat de notre premier calcul.

V.1.2. Accès séquentiel

L'accès séquentiel consiste à accéder chaque réalisation d'une entité dans l'ordre de ses numéros de réalisation.

La façon la plus simple de l'effectuer est de chercher à accéder à toutes les réalisations possibles de l'entité jusqu'à ce qu'on en trouve une qui existe. Cela peut entraîner de nombreux accès inutiles au dictionnaire, mais si la fonction de hash-coding est telle que des entités de noms internes consécutifs restent le plus souvent proches dans le dictionnaire, le nombre d'accès disque ne devrait pas être très élevé.

L'accès séquentiel peut être optimisé si on cherche à accéder par avance aux entrées dictionnaire de plusieurs réalisations consécutives : on cherche alors à obtenir un maximum d'entrées à chaque accès physique au dictionnaire.

Par ailleurs, l'accès séquentiel peut s'arrêter, soit lorsque la dernière réalisation a été testée, soit lorsqu'on trouve une réalisation indéfinie. Ce dernier cas permet d'accéder efficacement à des réalisations créées séquentiellement.

V.1.3. Accès associatif

Nous avons décrit au § III.2.5. la solution adoptée pour les accès associatifs. Il faut maintenant voir comment insérer ce mécanisme dans l'adressage par noms internes.

Une solution élégante consistait à insérer la table d'index dans le dictionnaire : la clé fournie par l'utilisateur permet de calculer un nom interne et l'entrée dictionnaire correspondante contient directement l'adresse de la tête de chaîne des réalisations de même clé. Cette solution n'a pas été retenue, de crainte d'alourdir la gestion du dictionnaire et d'introduire des risques d'incohérence.

Il nous a semblé préférable d'adopter la solution suivante : la table d'index est considérée comme une entité dont chaque réalisation contient un certain nombre d'entrées de la table. La clé fournie par l'utilisateur permet de calculer un numéro de réalisation et le rang de l'entrée cherchée dans la réalisation. Le nom interne de la réalisation est calculé et la fraction de table correspondante est accédée. L'entrée de la table correspondant à la clé nous fournit alors le nom interne de la tête de chaîne des réalisations ayant la même clé.

Chaque réalisation indexée contient le numéro de l'entrée de la table à laquelle elle est chaînée.

Les chaînages entre réalisations sont par noms internes, ce qui évite de lier le contenu des blocs de données à leurs emplacements physiques. Les opérations de reconfiguration, compactage, modification de structure, en seront grandement facilitées et la sécurité est accrue au prix de quelques accès supplémentaires au dictionnaire.

V.1.4. Accès par références et anneaux

Nous avons déjà vu au § III.2.4 comment une réalisation pouvait être accédée par l'intermédiaire d'un chaînage situé dans une autre réalisation. Comme pour les index, tous les chaînages sont par noms internes. L'accès à une réalisation référencée nécessite donc de lire la réalisation référençante pour obtenir le nom interne, puis d'accéder la réalisation référencée par l'intermédiaire du dictionnaire.

V.1.5. Accès divers

L'utilisateur peut désirer un certain nombre de renseignements sur la base : nombre de réalisations d'une entité, numéro d'une réalisation accédée par référence ou index, .. L'accès au nombre de réalisations d'une entité ne semble pas des plus courants, aussi n'envisage-t-on pas de gérer ce nombre dans la base et il devra être obtenu par comptage. Le numéro d'une réalisation pourra être obtenu par calcul à partir du nom interne et des renseignements fournis par la structure : NR = (NI - NIPR) modulo NBR. Des utilitaires d'analyse pourront être fournis pour faciliter la maintenance de la base : calcul du taux de remplissage des fichiers, par exemple.

V.2. LES FICHIERS

V.2.1. La structure

Le fichier structure centralisera les renseignements permettant le calcul des noms internes et décrira l'organisation des enregistrements de données.

Il sera créé à l'initialisation de la base à l'aide d'un compilateur de structure permettant de le décrire dans un langage commode. Chaque élément de la structure sera représenté par un enregistrement du fichier. La taille des enregistrements sera de 20 octets au maximum. On peut estimer que la taille du fichier dépassera rarement 4 K octets, ce qui tient largement sur une piste de CDC 9760.

V.2.2. Le dictionnaire

Le dictionnaire assure la liaison entre les noms internes et l'emplacement physique des données. L'accès se fait par hash-coding sur les noms internes, par une méthode proche de celle utilisée par SOCRATE pour l'accès aux sous-pages. Une entrée dictionnaire contient trois informations :

- le nom interne qui occupe 4 octets,
- un chaînage "synonyme" utilisé par le hash-coding, relie entre elles toutes les entrées dictionnaire correspondant à une même clé ; en consacrant 3 octets à ce chaînage, on peut avoir un dictionnaire de plus de 16 millions d'entrées, ce qui suffit largement pour un espace disque de 100 millions d'octets ;

. le chaînage données, qui occupe 3 octets, fournit l'adresse disque du secteur qui contient les réalisations, sous forme de numéros de secteurs dans l'espace des données : l'adresse physique peut en être déduite par un calcul simple. La réalisation peut être trouvée dans le secteur par recherche séquentielle puisqu'elle contient le nom interne. Trois octets permettent d'adresser de cette façon plus de 4 milliards d'octets de mémoire pour des secteurs de 256 octets.

La taille d'une entrée dictionnaire est donc de 10 octets. La taille du dictionnaire est fixe et doit être déterminée à l'initialisation de la base. Si la taille moyenne d'une réalisation est de 50 octets (cas de SCRIB), la taille du dictionnaire sera égale à 1/6 de l'espace disque, soit environ 70 pistes de disque (type 9760). Il peut arriver que le dictionnaire soit rempli bien avant l'espace des données, si la taille moyenne de celles-ci a été mal estimée. Il est alors prévu des possibilités d'extension et de reconfiguration.

Si par contre l'espace des données est saturé bien avant le dictionnaire, il n'est pas certain que la diminution de la taille de celui-ci permette un gain de place suffisant : c'est alors la taille de l'espace disque lui-même qui est insuffisante.

Il est donc tout à fait possible de prévoir une marge de sécurité assez large dans la taille du dictionnaire, ce qui influe assez peu sur l'utilisation de la mémoire et en outre améliore la rapidité de l'accès au dictionnaire.

V.2.3. Les données

La longueur des enregistrements de données est définie pour chaque entité dans le fichier structure (enregistrement entité).

Cette longueur diffère d'une entité à l'autre. Il se pose donc un problème pour l'allocation dynamique de la mémoire en longueur variable. Ce problème est cependant grandement simplifié du fait que toutes les longueurs possibles sont connues à la compilation de structure.

La solution que nous avons adoptée est la suivante : l'espace est géré en blocs de taille fixe (de l'ordre de 1 K octet) correspondant à un enregistrement physique. Un bloc ne peut être alloué qu'à des réalisations d'une même entité. On a donc :

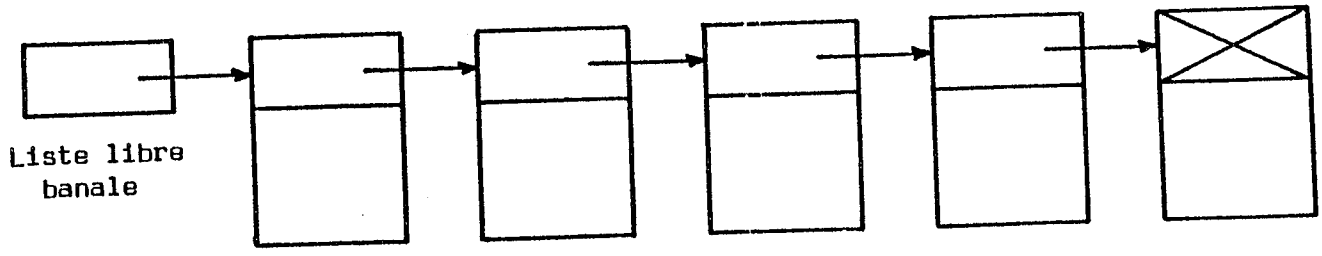
. des blocs entièrement vides, qui peuvent être alloués à n'importe quelle entité : ils forment une liste libre "banale" ;

. des blocs partiellement occupés par des réalisations d'une entité ; on ne peut plus y allouer de place que pour des réalisations de la même entité ; ils sont chaînés en autant de listes libres qu'il y a d'entités dans la structure ;

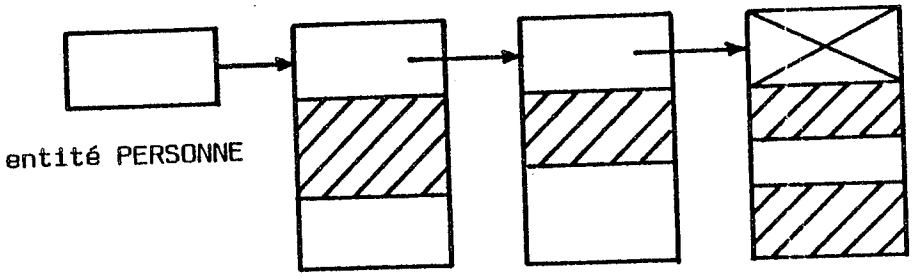
. des blocs entièrement occupés par des réalisations d'une entité.

Pour allouer un emplacement à une réalisation, on accède donc à la liste libre de l'entité. Si elle est vide, on prend un bloc dans la liste libre banale.

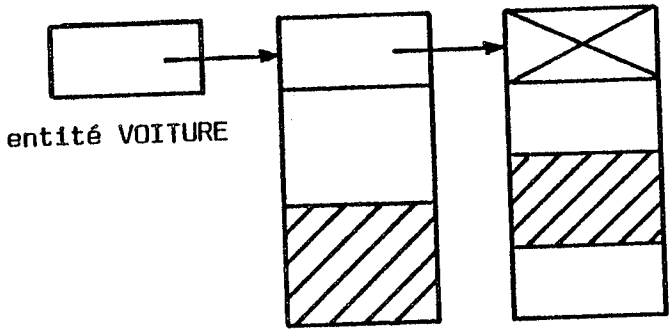
Exemple :



Blocs totalement libres



Liste des blocs partiellement occupés par des réalisations de PERSONNE



liste des blocs partiellement occupés par des réalisations de VOITURE

Listes libres des entités

Les diverses têtes de listes pourront être conservées en mémoire centrale durant l'exécution, ce qui évite l'accès au disque lors de leur consultation. Dans de nombreux cas, il est possible d'obtenir un emplacement libre en une seule lecture disque.

Structure des blocs

Chaque bloc contient le chaînage de la liste libre à laquelle il appartient, sur 4 octets, et un certain nombre de réalisations. Chaque réalisation comporte successivement :

. son nom interne,

. les chaînages des références, anneaux et clefs,

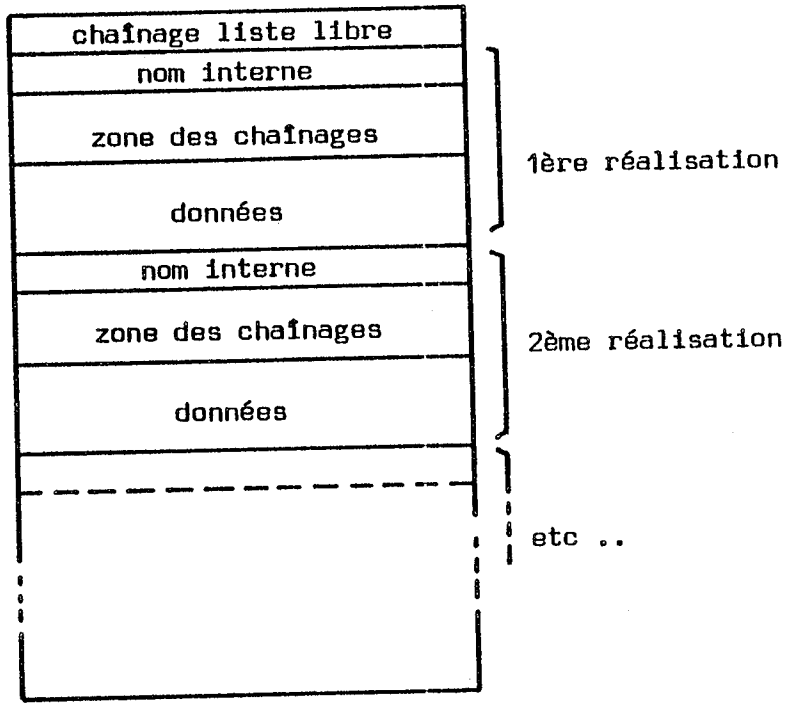
. les autres caractéristiques.

Le fait de regrouper les chaînages permet de contrôler plus facilement leur accès, pour éviter que l'utilisateur les lise ou les modifie. Par contre, l'accès à la valeur des caractéristiques n'est pas contrôlé.

Lorsqu'une réalisation n'est pas allouée, son nom interne est mis à zéro, ce qui permet de reconnaître les réalisations libres dans un bloc partiellement occupé.

Cette technique de gestion de données est simple, rapide et efficace. Elle permet, dans le dictionnaire, d'adresser les données par le seul numéro de leur bloc, puisqu'il est ensuite possible de rechercher séquentiellement la donnée dans le bloc. De plus, grâce aux noms internes rappelés dans les blocs de données, il est possible, en cas de besoin, de reconstituer le dictionnaire à partir des données.

Structure d'un bloc :



V.3. EXCLUSIONS MUTUELLES

L'accès simultané à la base par plusieurs utilisateurs, impose un certain nombre de précautions au niveau des programmes d'accès, pour garantir la cohérence des informations de la base et des informations transmises aux utilisateurs : deux utilisateurs ne doivent pas pouvoir modifier en même temps une même réalisation ; il est souhaitable qu'un utilisateur ne puisse pas lire une information qu'un autre est en train de modifier ; un chaînage en cours de modification ne doit pas être accédé par un autre utilisateur ; durant la création ou la suppression d'une réalisation, personne ne doit accéder au dictionnaire pendant qu'il est en cours de modification.

Le problème de la protection d'une information par l'utilisateur pour une série de lectures et d'écritures sera étudié au chapitre sur les langages (chapitre VI), car il ne concerne pas la cohérence interne de la base.

Le système d'exclusion mutuelle adopté pour les primitives d'accès s'inspire largement de SOCRATE [AC]. Un certain nombre de ressources peuvent être manipulées. Une ressource peut être dans l'un des trois états suivants :

. libre, si personne ne l'accède,
. en lecture, si seuls des accès en lecture peuvent être effectués,
. en mise à jour, si des écritures peuvent être effectuées.

Quatre opérations seront utilisées par les primitives de MAGE :

. LECT (ressource), permet d'accéder à la ressource, si elle est libre ou en lecture, et si aucune demande d'écriture n'est en attente ; sinon, le processus demandeur est mis en attente ; si la ressource était libre, son état devient "en lecture" ;

. MAJ (ressource), permet d'accéder à la ressource si elle est libre ; sinon, le processus demandeur est mis en attente ; l'état de la ressource devient ensuite "en mise à jour" ;

. NOLECT (ressource), indique qu'un processus a cessé d'utiliser la ressource en lecture ; celle-ci ne devient libre que si aucun processus ne l'accède plus ;

. NOMAJ (ressource), indique qu'un processus a cessé d'utiliser la ressource en écriture ; la ressource devient libre.

Trois sortes de ressources peuvent faire l'objet de ces opérations :

. les réalisations d'entités,
. les éléments de structure,
. le dictionnaire.

Les risques d'interblocage liés à l'utilisation de ces opérations peuvent être évités en imposant la réservation dans l'ordre : dictionnaire, élément de structure, réalisation, et libération dans l'ordre inverse [HA] (prévention des interblocages par hiérarchisation des ressources).

Si plusieurs ressources de même catégorie doivent être réservées en même temps, on peut imposer une réservation dans l'ordre des noms ou une réservation groupée.

CHAPITRE VI

DÉFINITION DES LANGAGES

Une description succincte et superficielle des langages a été donnée au chapitre III. A ce point de l'étude, nous pouvons en donner une définition plus précise.

VI.1. LANGAGE DE DÉFINITION DE STRUCTURE

VI.1.1. Syntaxe

La syntaxe complète du langage est donnée en annexe. Seuls quelques points sont à préciser par rapport au chapitre III.

. Racine :

Les caractéristiques de niveau 1 (non contenues dans une entité) forment un bloc de données que l'on appelle "racine" de la base. Ce bloc peut avoir des caractéristiques de tous les types, excepté CLE (ce qui n'aurait aucun intérêt). Une référence sur la racine peut avoir un intérêt en tant que liste chaînée avec la tête de liste dans la racine : les anneaux sont donc autorisés au niveau 1. Les références et anneaux étant regroupés au début des blocs de données, leurs déclarations doivent être faites au début des entités et de la racine.

. Tableaux :

Dans SOCRATE, la seule façon de représenter une information multiple est l'entité. Or, l'entité représente un mécanisme relativement coûteux en place et nombre d'accès et le dynamisme qu'il permet n'est pas toujours nécessaire.

Si on désigne par NBR le nombre maximum de réalisations d'une entité, T le taux d'existence des réalisations (rapport du nombre de réalisations existantes sur NBR), et L la longueur d'une réalisation, il est plus avantageux de réserver statiquement de la place pour toutes les réalisations lorsque :

$$NBR \times L \leq (14 + L) \times NBR / T$$

14 étant ici la longueur des descripteurs associés à une entité, soit 10 octets d'entrée dictionnaire et 4 octets de nom interne dans les données.
Ce qui donne :

$$(1)\ T \geq L / (14 + L)$$

$$(2)\ L \leq 14\ T / (1 - T)$$

Exemple :

Si T = 1/2 alors L ≤ 14

si L = 4 alors T ≥ 0,23.

Il nous a donc paru intéressant de définir la notion de tableau pour gérer statiquement des caractéristiques multiples.

L'utilisation des tableaux permet un plus faible encombrement, un accès plus rapide à partir de l'entité englobante et une diminution du nombre de noms internes et de la taille du dictionnaire.

Les tableaux ne sont pas un type d'élément particulier. Certaines caractéristiques pourront être organisées en tableaux, en introduisant dans leur déclaration un attribut de dimension par le mot-clé TABLEAU.

La dimension peut être comprise entre 1 et 256.

. Entités :

Pour chaque entité, on doit définir un nombre maximum de réalisations, pour pouvoir affecter statiquement un nom interne à chaque réalisation. Ce nombre peut être compris entre 1 et 65 535 (2^{16} - 1).

. Index et clés :

Les caractéristiques clé sont regroupées entre les pointeurs et les caractéristiques. En effet, elles sont composées d'un pointeur, rattaché à la zone des pointeurs, et d'une chaîne d'octets, rattachée à la zone des données.

La table d'index peut avoir entre 1 et 65 536 entrées.

Une clé ne peut pas être un tableau.

IV.1.2. Compilation

La structure étant décrite sous la forme précédemment exposée, il faut en tirer un fichier structure utilisable par MAGE et les renseignements nécessaires à la compilation des requêtes (table des symboles de la structure) : c'est le rôle du compilateur de structure. Il effectuera les opérations suivantes :

. passer à une forme codée des informations,
. claculer les informations qui devront figurer dans la structure : noms internes, longueur des réalisations, ... ,

- modifier la représentation de l'arborescence : l'utilisateur décrit la structure sous une forme parenthésée et le fichier structure généré est sous forme chaînée,
- générer la table d'équivalences des symboles qui permettra au compilateur de requêtes de connaître le numéro d'enregistrement de structure auquel correspond un identificateur.

La compilation de structure est réalisée sur une machine distincte de MAGE. Le fichier généré peut être implanté soit directement sur le disque de MAGE, soit écrit sur une bande magnétique qui sera recopiée par MAGE sur son disque.

Le compilateur devra être portable, puisque écrit et mis au point à Grenoble sur IRIS 80 et destiné à un ordinateur SAGEM, l'ULP. Il doit être facilement modifiable en fonction de l'évolution du projet.

Il a été écrit en CPL/1, sous-ensemble de PL/1, défini par CAP-SOGETI et utilisé par la SAGEM.

VI.2. LANGAGE D'ACCÈS

VI.2.1. Contexte utilisateur

Nous avons vu au chapitre III, qu'à tout utilisateur de la base devront être associés un ou plusieurs contextes. Un contexte désignera un élément de la base et un certain nombre de primitives permettront de déplacer le contexte pour accéder au nouvel élément, et d'accéder aux données ainsi accédées.

La partie essentielle du contexte est une pile qui contiendra à chaque niveau un numéro de ligne de structure et un numéro absolu de réalisation, ce qui suffit pour définir une caractéristique d'une réalisation donnée.

L'utilisation d'une pile est indispensable pour permettre le retour au niveau précédent après les accès par référence ou par index. L'expérience de SCRIB montre qu'une pile de 10 entrées serait largement suffisante.

Le contexte pourra en outre regrouper les zones de travail et les mémoires tampon (structure, dictionnaire, données) propres à l'utilisateur.

VI.2.2. Ouverture et fermeture d'un contexte

Pour utiliser la base de données, l'utilisateur doit se faire allouer un ou plusieurs contextes. C'est le rôle de la commande OUVRIR :

OUVRIR (<idu>, <idc1>, [<idc2>])

<idu> est l'identificateur de l'utilisateur (1 octet) ;

<idc1> est l'identificateur du contexte à créer (1 octet). Un contexte de même numéro ne doit pas déjà exister pour le même utilisateur.

<idc2> est l'identificateur d'un contexte du même utilisateur à partir duquel le nouveau contexte sera initialisé : la base de la pile du nouveau contexte reçoit la valeur du sommet de la pile du contexte désigné. Cette possibilité permet une extension en cas de saturation de la pile. Si <idc2> est nul, le contexte est positionné sur la racine de la structure.

Lorsqu'un utilisateur cesse d'utiliser un contexte, il doit le libérer afin de permettre l'utilisation de son emplacement par un autre utilisateur. Ceci est fait par la commande FERMER :

FERMER (<idu>, <idc>)

<idc> identifie le contexte à supprimer.

VI.2.3. Primitives d'adressage

. APPEL (<mode>, <données>, <ls>, <nr>, <idc>, <idu>)

Pour toutes les requêtes, <idu> et <idc> permettent l'identification du contexte.
<mode> désigne le mode de l'accès : lecture, écriture, création, ..
<données> identifie la zone de données de l'utilisateur.
<ls> est le numéro de l'enregistrement de structure de la donnée sur laquelle on veut se positionner.
<nr> est le numéro de la réalisation d'entité sur laquelle on veut se positionner.

Le contexte est descendu d'un niveau dans la structure. L'enregistrement de structure désigné par <ls> doit être fils de l'élément précédent désigné par le contexte.

Le traitement effectué dépend du type d'élément adressé et éventuellement du mode :

. *entité* : <nr> fournit le numéro de réalisation ; s'il est nul, on accède à la première réalisation existante, ou à la première réalisation inexistante si le mode est *créer*

. *caractéristique simple, bloc, référence* : si la caractéristique est organisée en tableau, <nr> fournit l'indice ;

. *référence* : si le mode n'est pas *écrire*, le contexte est positionné sur la réalisation référencée ; sinon, on se positionne sur la référence elle-même ;

. *anneau* : le contexte est placé sur la réalisation chaînée à l'anneau ;

. *index* : <nr> fournit le numéro de l'entrée de la table et <données> permet d'obtenir la valeur de la clé. Le contexte est positionné sur la réalisation correspondant à la clé.

A l'issue de APPEL, le nouveau numéro d'enregistrement de structure et le nouveau numéro de réalisation sont empilés.

Exemple :

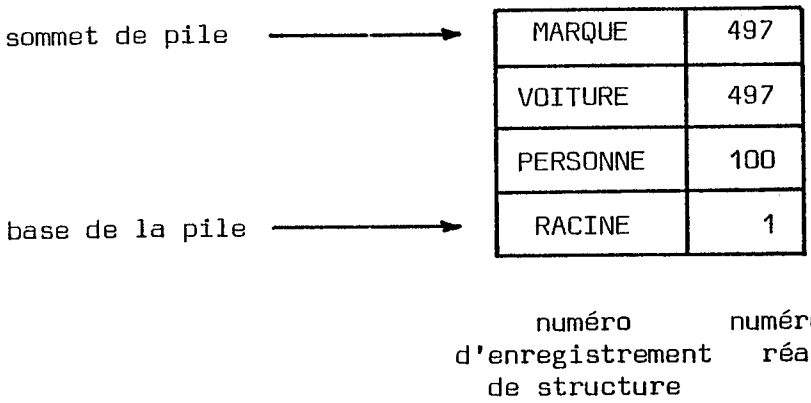
Reprenons la structure donnée au § IV.2. On suppose le contexte initialisé sur la racine.

. APPEL (mode, données, PERSONNE, 100, idc, idu) : cette requête va placer le contexte sur la PERSONNE 100.

. APPEL (mode, données, VOITURE, 2, idc, idu) : cette requête place le contexte sur la VOITURE 2 de PERSONNE 100.

. APPEL (mode, données, MARQUE, 0, idc, idu) : cette requête place le contexte sur la MARQUE de la VOITURE 2 de la PERSONNE 100.

A ce stade, la pile du contexte aura la configuration suivante :



. RETOUR (<ls>, <nd>, <idc>, <idu>)

Cette opération est l'inverse de APPEL. Elle consiste à effectuer un ou plusieurs dépilages pour revenir en un point précédemment adressé (retour en arrière dans la structure).

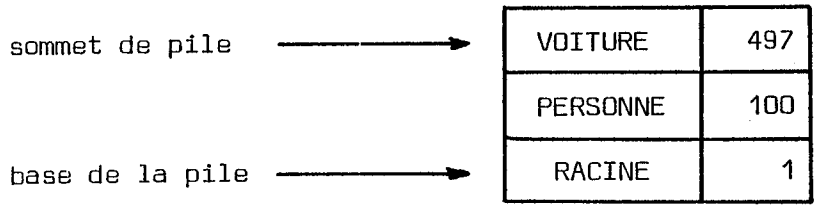
<nd> est le nombre de dépilages à effectuer.

Si <nd> = 0, alors <ls> contient le numéro de l'enregistrement de structure sur lequel on veut revenir : il y aura alors dépilage jusqu'à ce qu'on rencontre l'entrée de pile qui pointe sur cet enregistrement.

Aucun accès aux données ne peut être effectué par cette requête.

Exemple :

La pile étant dans l'état décrit au paragraphe précédent, l'exécution de RETOUR (0, 1, idc, idu) va commencer le contexte sur VOITURE 2 de PERSONNE 100, la pile ayant la configuration suivante :



. FRERE (<mode>, <donnée> , <ls>, <nr>, <idc>, <idu>)

Cette opération permet le passage à un élément de structure frère de celui qui est adressé par le contexte. Elle est équivalente à la succession des deux opérations : RETOUR (0, 1, <idc>, <idu>)
APPEL (<mode>, <données>, <ls>, <nr>, <idc>, <idu>)

Exemple :

FRERE (mode, données, NOM, 0, idc, idu) va placer le contexte précédent sur NOM de personne 100, la pile ayant la configuration suivante :

sommet de pile ⟶

NOM	100
PERSONNE	100
RACINE	1

. INIT (<mode>, <données>, <ls>, <nr>, <idc>, <idu>)

Cette opération permet d'initialiser un accès séquentiel. Elle a la même action d'adressage qu'un APPEL ayant les mêmes paramètres, mais des informations sont sauvegardées pour faciliter les accès ultérieurs.

Si <ls> désigne une entité, le contexte est positionné sur la réalisation <nr>.

Si <ls> désigne un anneau, on se positionne sur la première réalisation référençante chaînée à l'anneau.

Si <ls> désigne une référence, on se positionne sur la réalisation suivante dans la chaîne des références de même valeur.

Si <ls> désigne un index, on se positionne sur la première réalisation de la chaîne ayant une caractéristique clé égale à la valeur transmise dans <données>

. SUIVANT (<mode>, <données>, <indic>, <idc>, <idu>)

Cette opération permet de continuer un accès séquentiel initialisé par INIT.

Si l'accès se fait sur une entité, on passe à la réalisation suivante. Selon la valeur de <indic>, deux actions sont possibles :

- recherche de la première réalisation existante et arrêt lorsque la dernière réalisation possible a été testée,
- accès à la réalisation suivante et arrêt si elle n'existe pas (c'est-à-dire arrêt dès qu'on rencontre une réalisation qui n'existe pas). Ce type d'accès convient tout à fait à une entité dont les réalisations sont créées séquentiellement.

Dans tous les cas, il y a arrêt de l'accès à la dernière réalisation de l'entité dans une réalisation de l'entité englobante.

Si l'accès est sur une référence ou un anneau, on passe au prochain élément de la liste.

S'il est sur un index, on recherche une nouvelle réalisation ayant la même valeur de clé.

Dans tous les cas, la nouvelle réalisation accédée remplace au sommet de la pile la réalisation précédente.

Exemple :

Si le contexte est positionné sur la racine, on peut initialiser un accès séquentiel sur toutes les personnes en effectuant :

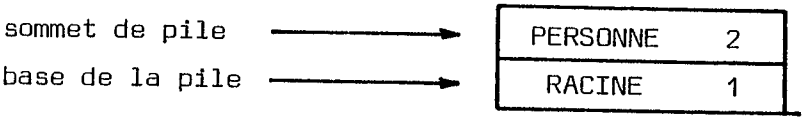
INIT (mode, données, PERSONNE, 1, idc, idu)

ce qui positionne le contexte sur PERSONNE 1.

Pour passer à PERSONNE 2, on fait :

SUIVANT (mode, données, indic, idc, idu).

Si PERSONNE 2 existe, la pile aura la configuration suivante :



Si PERSONNE 2 n'existe pas, la valeur de indic dira s'il faut rechercher la première réalisation existante ou signaler une erreur.

. MONTER (<ls>, <nr>, <idc>, <idu>)

Ayant atteint une réalisation par l'intermédiaire d'une référence ou d'un index, on peut vouloir accéder à une réalisation englobante. C'est le rôle de la commande MONTER.

Si <nr> est différent de zéro, on remonte de <nr> niveaux dans la structure ; sinon, on remonte jusqu'à l'enregistrement de structure <ls>. La réalisation atteinte remplace au sommet de la pile la réalisation précédemment adressée.

On ne peut utiliser cette opération qu'après un accès par référence, anneau ou index, ou lorsqu'on est au niveau 1 de la pile, si ce niveau ne pointe pas sur la racine de la structure.

. IDEM (<mode>, <données>, <idc>, <idu>)

On peut désirer effectuer successivement plusieurs opérations sur une même donnée : par exemple, une lecture puis une écriture. Pour cela, la requête IDEM permet de demander une opération sans modifier l'adressage.

VI.2.4. Modes d'accès

Les opérations décrites dans le paragraphe précédent réalisent essentiellement l'adressage des données. Certaines d'entre elles possèdent un paramètre, *le mode*, qui indique l'opération qui doit être effectuée sur la donnée. Sept opérations sont possibles.

. RIEN :

Un grand nombre d'opérations se bornent à adresser une réalisation d'entité dont on ne veut accéder qu'à une caractéristique. Le mode RIEN sert donc dans ce cas à spécifier qu'aucun accès n'est demandé.

. VERIFIER :

Ce mode permet de vérifier l'existence d'une réalisation d'entité sans avoir à y accéder.

. LIRE :

La donnée adressée est lue et transmise à l'utilisateur. S'il s'agit d'une entité, seules sont transmises les valeurs des clés, des caractéristiques simples et des blocs. Si la donnée est une référence, c'est la réalisation référencée qui est transmise. Il en est de même pour les index et anneaux.

. ECRIRE :

Les données transmises par l'utilisateur sont écrites dans la base. Si une entité est adressée, seule la zone des données est modifiée. Si une référence est adressée, le paramètre <données> doit contenir le numéro d'un contexte du même utilisateur. Ce contexte doit adresser la réalisation d'entité sur laquelle la référence doit être positionnée. Si un index est adressé, le paramètre <données> contient le numéro d'un contexte du même utilisateur qui adresse une réalisation de l'entité indexée. Cette réalisation sera insérée dans la liste donc l'index contient la tête. Si une clé est spécifiée, elle est écrite comme une caractéristique simple.

. CREER :

Si la donnée est une entité, la réalisation adressée est créée. Si <nr> est nul, la réalisation adressée est la première réalisation inexistante de l'entité. De même, si la réalisation est créée en accès séquentiel (INIT ou SUIVANT), on adresse la première réalisation inexistante à partir de la réalisation précédemment adressée. Si la donnée est un index, on crée une réalisation de l'entité indexée et on la chaîne à l'entrée de l'index adressée. Le numéro de réalisation affecté est le premier numéro libre que l'on trouve. Dans ces conditions, on affecte la place en mémoire et le numéro de réalisation, de façon à minimiser le nombre d'accès disques nécessaires pour parcourir la chaîne.

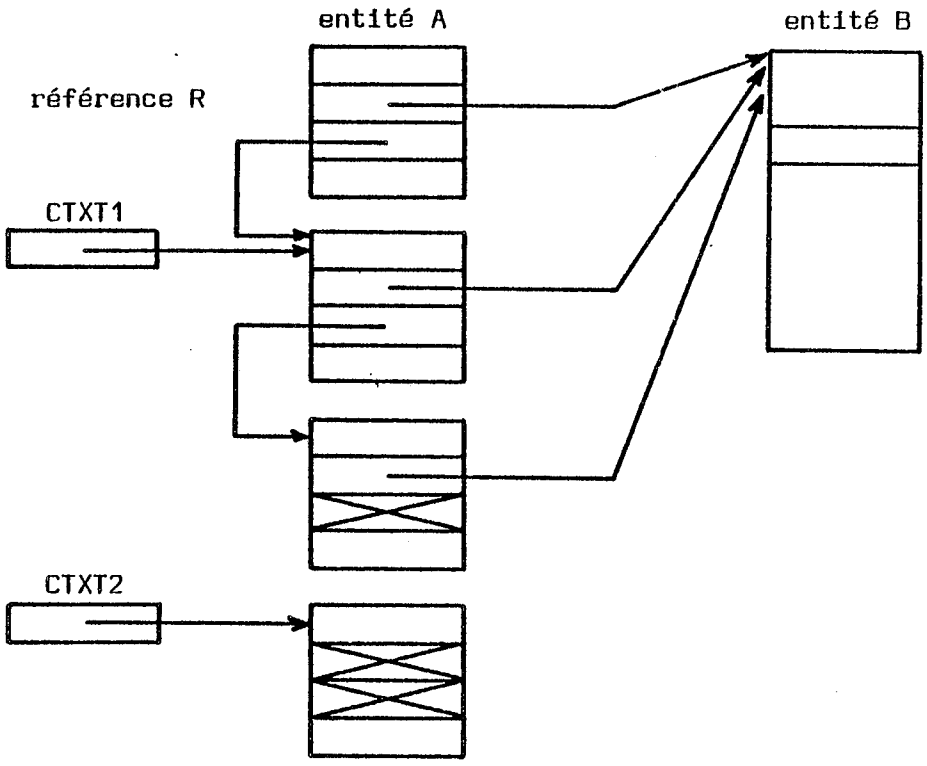
. SUPPRIMER :

Si la donnée est une entité, la réalisation adressée est effacée.

. INSERER :

Cette opération permet de positionner une référence en l'insérant en un point bien défini de l'anneau. Pour cela, <données> contient le numéro du contexte. Celui-ci désigne non pas la réalisation d'entité sur laquelle on désire faire pointer la référence, mais une réalisation appartenant déjà à l'anneau dans lequel on veut insérer la nouvelle réalisation. La référence à écrire recevra donc la même valeur que celle désignée par le second contexte et l'insertion dans l'anneau se fera *après* la réalisation désignée et non en tête de chaîne comme dans ECRIRE.

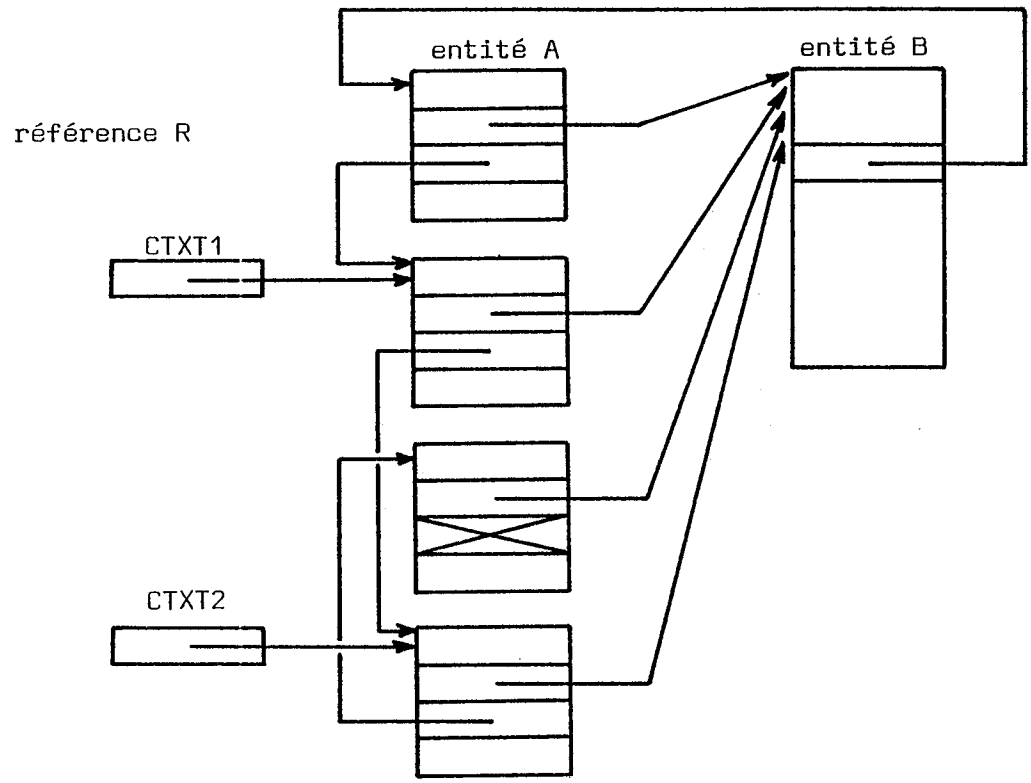
Exemple :



Si on effectue sur la base l'opération :

APPEL (INSERER, CTXT2, R, O, CTXT1, IDU)

on obtient le résultat suivant :



Exemple :

Considérons la structure suivante :

```
INDEX NOMPERS 50 sur NOM ;
entité 10 000 PERSONNE ; début ;
   clé NOM 20 ;
   CS ADRESSE 40 tableau 2 ;
   entité 5 VOITURE ; début ;
      CS MARQUE 20 ;
      CS NUMERO 20 ;
   fin ;
fin ;
```

Si on veut imprimer le numéro et la marque de toutes les voitures d'une personne s'appelant DUPONT, on effectue les opérations suivantes :

```
OUVRIR (IDU, IDC, 0) ;
calculer MCOD à partir de 'DUPONT' ;
APPEL (rien, 'DUPONT', NOMPERS, MCOD, IDC, IDU) ;
INIT (rien, 0, VOITURE, 0, IDC, IDU) ;
faire tant que retour sans erreur ;
début ;
   APPEL (lire, CHAINE1, NUMERO, 0, IDC, IDU) ;
   FRERE (lire, CHAINE2, MARQUE, 0, IDC, IDU) ;
   imprimer (CHAINE1, CHAINE2) ;
   RETOUR (0, 1, IDC, IDU) ;
   SUIVANT (rien, 0, indic, IDC, IDU) ;
fin ;
FERMER (IDC, IDU) ;
```

VI.2.5. Exclusion mutuelle

L'interprétation par MAGE des opérations comporte un certain nombre d'exclusions mutuelles pour éviter que l'exécution parallèle de certaines opérations sur une même donnée ne conduise à des incohérences internes de la base. Cependant, l'exécution parallèle des programmes utilisateurs peut encore entraîner des erreurs.

Considérons par exemple les deux programmes suivants, exécutés par deux utilisateurs U1 et U2 et modifiant une même donnée A :

U1	U2
lire A ;	lire A ;
A := A + 20 ;	A := A - 10 ;
écrire A ;	écrire A ;

Si A est, par exemple, la quantité en stock d'un certain article, U1 introduit 20 pièces dans le stock et U2 en retire 10. Si Ao est la valeur initiale, on doit trouver après les deux opérations : A = Ao + 10.
Or, les opérations peuvent se dérouler dans l'ordre suivant :

U1 → lire A ; A := A + 20 ; (valeur lue = Ao)

U2 → lire A ; A := A - 10 ; (valeur lue = Ao)

U1 → écrire A ; (valeur écrite = Ao + 20)

U2 → écrire A ; (valeur écrite = Ao - 10)

La valeur finale de A sera donc égale à Ao - 10.

Pour éviter ce genre d'inconvénient, il faut qu'un utilisateur puisse verrouiller une donnée pendant un certain nombre d'opérations. Il pourrait par exemple faire :

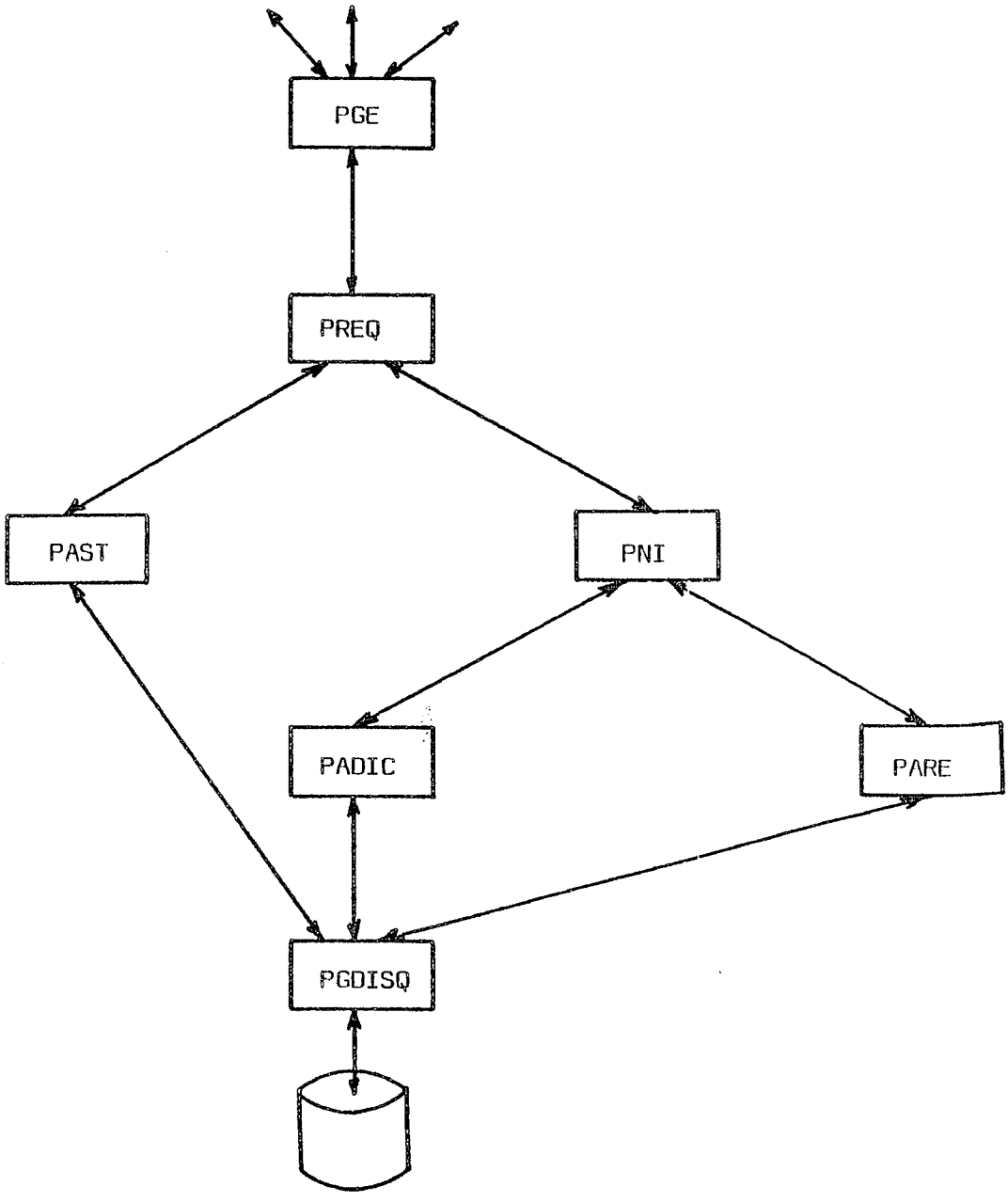
verrouiller A ;
lire A ; A := A + 20 ; écrire A ;
libérer A ;

Ceci présente toutefois des risques d'interblocages. Dans SOCRATE (implémentation CII), ce risque est évité en plaçant le blocage au niveau de la base elle-même et non d'une seule réalisation.

Des solutions plus fines existent, mais elles ne seront pas étudiées ici. Nous en resterons là pour l'instant, en considérant que ce point pourra être approfondi plus tard.

VI.3. ARCHITECTURE LOGICIELLE DE MAGE

La méthode d'adressage retenue permet une décomposition du travail en un nombre limité de grandes opérations. Chaque opération peut être exécutée par un processeur logique qui peut être concrétisé, bien sûr, par un processeur physique spécialisé, mais plus vraisemblablement par une tâche particulière d'un processeur, ou groupe de processeurs décomposé différemment :



Sept processeurs logiques sont définis :

. PGE : processeur de gestion des échanges :

ce processeur assure l'interface avec le ou les processeurs utilisateurs. Il reçoit les requêtes, gère la transmission des données entre MAGE et l'utilisateur, émet les signaux de fin de requête.

. PREQ : processeur de traitement des requêtes :

il reçoit et gère les requêtes des utilisateurs ; il accède à la structure par l'intermédiaire de PAST afin de calculer les noms internes ; il déclenche l'accès aux données par noms internes, effectue le traitement des références, anneaux et index ; il fait effectuer par PGE les échanges de données avec l'utilisateur.

. PAST : processeur d'accès à la structure :

il reçoit les numéros des enregistrements de structure à lire et retourne les enregistrements à PREQ. Il effectue les calculs d'adresse disque nécessaires, demande les lectures disque, gère les mémoires tampon.

. PNI : processeur de noms internes :

il reçoit de PREQ le nom interne et la longueur des réalisations à accéder. Il accède au dictionnaire par l'intermédiaire de PADIC et aux données par l'intermédiaire de PARE.

. PADIC : processeur d'accès au dictionnaire :

il reçoit un nom interne et accède à l'entrée dictionnaire correspondante. Il effectue les calculs d'adressage, lance les entrées-sorties nécessaires, gère les mémoires tampon dictionnaire. Il optimise les accès au disque. Il gère l'espace du dictionnaire.

. PARE : processeur d'accès aux réalisations :

il reçoit le nom interne, la longueur et l'adresse disque d'une réalisation. Il lance les entrées-sorties nécessaires, gère les mémoires tampon des données, recherche les réalisations dans les blocs. Il gère l'espace disque des données.

. PGDISQ : processeur de gestion du disque :

il reçoit les adresses disque sous forme de numéro de bloc dans un fichier. Il les interprète et les convertit en fonction du fichier demandé et de la configuration disque. Il lance les accès physiques en optimisant les déplacements de bras. Il gère et contrôle les accès disque extérieurs à la base de données en assurant la protection de la base. Ce processeur, par sa modularité, permet au reste de MAGE d'être indépendant de la nature et de la configuration du support physique de l'information.

CHAPITRE VII

MÉTHODOLOGIE DE RÉALISATION

Les chapitres précédents ont abouti à une définition fonctionnelle de MAGE. La suite du travail va comporter deux aspects :

. l'analyse et la programmation du logiciel : interprétation des requêtes, gestion des processus, synchronisation, ..

. la conception du matériel.

Ces deux aspects sont très liés. On doit concevoir le matériel le plus économique possible pour exécuter efficacement le logiciel. Par ailleurs, une analyse poussée du logiciel ne peut se faire indépendemment du matériel qui doit l'exécuter.

Un certain nombre de compromis sont à rechercher entre le matériel et le logiciel, pour assurer une rapidité suffisante du processeur et un coût minimal. En effet, si une amélioration peu coûteuse du matériel permet une diminution notable de la taille des programmes, cela entraînera une plus faible taille de la mémoire, d'où un coût matériel plus faible. En outre, si les temps de traitement apparaissent critiques, seule l'étude du logiciel peut permettre de dire quelle est l'amélioration à apporter pour obtenir une accélération suffisante à un coût minimal.

Le matériel et le logiciel doivent donc être conçus en parallèle, ce qui implique de travailler en plusieurs étapes, chaque étape permettant d'approfondir l'étude, de rectifier les erreurs de l'étape précédente, de préciser les hypothèses de l'étape suivante.

VII.1. CHOIX D'UNE TECHNOLOGIE

Il n'est guère possible d'aller plus avant dans la conception de MAGE sans faire le choix d'une technologie de réalisation.

Trois possibilités s'offraient à nous :

. utilisation d'un processeur existant,
. conception de toutes pièces d'un processeur spécialisé,
. utilisation de microprocesseurs.

En ce point de l'étude, on dispose de peu de moyens objectifs d'effectuer ce choix. La meilleure solution était donc d'effectuer un choix a priori, qui paraissait raisonnable, pour ensuite l'évaluer et éventuellement l'adapter ou le remettre en question.

VII.1.1. Choix d'une solution

La première solution consisterait à programmer MAGE sur un mini-ordinateur existant, en modifiant la microprogrammation et en effectuant éventuellement quelques adaptations matérielles. Un tel processeur serait certainement très sous-employé, puisque la rapidité de MAGE est limitée par la durée des entrées-sorties. Etant donné le coût encore élevé de ces machines, on préfèrera s'orienter vers des solutions moins coûteuses. En outre, les mini-ordinateurs tels qu'on les connaît actuellement, risquent d'être rapidement concurrencés, voire remplacés, par les microprocesseurs.

La seconde solution peut amener à la conception d'un matériel particulièrement bien adapté. Il existe maintenant des circuits intégrés dits "microprocesseurs en tranches", ou macrocomposants, qui peuvent être utilisés pour réaliser relativement facilement des machines informatiques microprogrammées. Ainsi, l'AMD 2900 est composé d'un circuit "opérateur" et d'un circuit "contrôle" travaillant chacun sur 4 bits. Pour réaliser un processeur 16 bits, on peut donc cascader 4 circuits de chaque sorte et leur associer une mémoire de microprogrammes. De tels circuits peuvent permettre de concevoir des processeurs assez rapides, mais on peut craindre qu'une telle solution soit coûteuse, en partie à cause de la mémoire de microprogrammes qui doit être très rapide, donc très chère. Par ailleurs, on peut craindre qu'une telle solution ne représente un travail important, puisque l'analyse devrait être poussée jusqu'au niveau de la microprogrammation. Une fois réalisés ces microprogrammes, un logiciel devrait être développé pour pouvoir utiliser le langage machine ainsi défini. On peut cependant espérer que la suite du travail soit simplifiée par le fait de disposer d'un langage machine bien adapté. Une solution plus simple et plus économique pourrait être préférée à celle-ci.

La troisième solution consiste à utiliser un ou plusieurs microprocesseurs. Les microprocesseurs sont des circuits peu coûteux (environ 200 F l'unité) et assez simples à mettre en oeuvre et à programmer. Tout un logiciel est disponible pour faciliter le développement des applications (metteur au point, assembleur, chargeur, éditeur de liens et même compilateur). On peut moduler la puissance de la machine finale en faisant travailler plusieurs processeurs en parallèle, ou en ajoutant des opérateurs externes pour faciliter certaines opérations. Enfin, on peut adapter la structure de l'espace d'adressage aux besoins du logiciel. Si ces circuits sont relativement lents, ils devraient tout de même être suffisants pour l'application projetée. On peut espérer trouver là une solution simple et économique pour la réalisation de MAGE, tout en utilisant une technologie appelée à se développer au cours des années à venir.

Notre choix s'est donc porté sur cette dernière solution, bien qu'elle soit apparue, à l'époque, comme une utilisation extrême de ce type de circuits qui n'étaient guère utilisés que pour réaliser des automatismes ou contrôler des téléimprimeurs.

VII.1.2. Evaluation de la technologie choisie

Il n'est pas question de multiplier le nombre de microprocesseurs utilisés pour adapter la puissance du processeur aux besoins : au-delà de quelques microprocesseurs, la rentabilité de cette solution peut être mise en doute et les problèmes de communication, synchronisation et partage de mémoire peuvent devenir tels qu'aucun gain en performance n'est obtenu en multipliant le nombre des microprocesseurs.

Ceci impose donc de répondre à la question : peut-on réaliser MAGE de manière efficace avec un petit nombre de microprocesseurs ?

Nous devons commencer par définir un critère d'efficacité : MAGE génère un grand nombre d'échanges disques. L'unité de disque sera très vraisemblablement la ressource la plus chère du système, ce qui est une première raison de l'utiliser au maximum. Par ailleurs, les échanges disques pourront être optimisés en nombre et en durée certes, mais pas indéfiniment, si bien que la durée moyenne d'exécution d'une requête aura une valeur minimale qui est la durée moyenne des échanges disques qu'elle entraîne. Un bon critère d'efficacité est donc que la durée moyenne d'exécution d'une requête soit égale à la durée moyenne des échanges disques qu'elle entraîne, c'est-à-dire que le traitement soit assez rapide pour alimenter le disque en demandes d'échanges de manière à l'utiliser à plein temps. Le traitement doit donc pouvoir se faire pendant les échanges disques et même dans un temps très inférieur à ceux-ci pour minimiser le risque de faire attendre une requête au disque.

Notre critère d'efficacité sera donc le suivant : la durée moyenne du traitement d'une requête doit être très inférieure à la durée moyenne des échanges disques générés par une requête.

Le problème qui se pose maintenant est donc : comment effectuer cette comparaison ? Le problème essentiel est que la durée du traitement peut difficilement être évaluée sans avoir déterminé le matériel qui effectuera ce traitement. Nous avons donc adopté la méthodologie suivante : après avoir choisi un microprocesseur parmi ceux qui existent sur le marché, on évalue la durée moyenne du traitement sur un processeur de ce type, en déterminant la répartition du temps entre les diverses catégories d'opérations. La comparaison du temps de traitement évalué avec les temps d'échanges disques permet de dire si le choix est bon ou doit être modifié, si des opérateurs doivent être ajoutés, si plusieurs microprocesseurs doivent être utilisés, ou même s'il faut renoncer à utiliser des microprocesseurs.

VII.1.2.1. Choix d'un microprocesseur

Ce choix a été effectué au début de l'année 1976. Parmi les microprocesseurs disponibles à cette époque, il n'en existait que deux qui étaient suffisamment puissants pour nous convenir : le MC 6800 de MOTOROLA et le 8080 de INTEL. Les études comparatives menées sur ces microprocesseurs n'ont pas révélé de supériorité déterminante de l'un sur l'autre en ce qui concerne la rapidité. Par contre, le MC 6800 nous était déjà familier, il disposait de plusieurs circuits associés (interface d'entrée-sortie), était d'une utilisation plus simple et son langage machine semblait plus facile à utiliser. En outre, les impératifs industriels de la SAGEM militaient également pour le MC 6800. Nous avons donc choisi d'évaluer la réalisation de MAGE avec ce microprocesseur, choix qui ne serait vraisemblablement remis en question que s'il apparaît particulièrement mal adapté, ou en fonction des nouveaux produits apparaissant sur le marché.

VII.1.2.2. Techniques d'évaluation

Deux données doivent être évaluées : la durée des accès disques et la durée du traitement sur MC 6800. Nous pouvons déterminer la durée moyenne d'un accès disque à partir des données du périphérique utilisé et l'hypothèse des optimisations réalisables. Nous n'avons donc plus à évaluer que le nombre des accès disques pour en déduire la durée.

Plusieurs techniques d'évaluation existent : simulation, mesures sur prototype, estimations.

La simulation nécessite une modélisation de MAGE et des utilisateurs. Ceci représente un travail assez important, non récupérable pour la suite du projet ; en outre, une erreur dans la modélisation entraînerait des résultats erronnés. Cette méthode est donc peu précise.

Effectuer des mesures sur un prototype implique d'effectuer une première réalisation de MAGE aux fins d'évaluation. Comme il n'est pas question de réaliser physiquement MAGE pour cela, cette réalisation peut se faire en langage évolué, soit en assembleur MC 6800 exécuté sur simulateur. Ce travail de réalisation est considérable et une bonne partie risque d'être remise en question dans la suite du projet et n'est donc par récupérable. Par ailleurs il existe trois sources d'imprécision dans les mesures effectuées :

- au niveau de l'objet mesuré, qui n'est certainement pas conforme au produit MAGE définitif,
- au niveau des mesures effectuées : on ne peut pas tout mesurer, il faut faire un choix et il n'est pas certain que ce choix soit réellement représentatif de l'activité de MAGE,
- au niveau des jeux d'essai employés : les mesures seront effectuées à partir d'un ensemble de requêtes supposé représentatif d'une activité moyenne de MAGE. Des erreurs peuvent donc intervenir dans le choix de ces requêtes.

Ces quelques remarques nous amènent à penser que cette technique d'évaluation, qui pourrait paraître très précise, ne l'est en fait que très peu. Devant le coût qu'elle représente, en travail et en heures d'ordinateur, nous avons préféré y renoncer au profit d'une méthode peut-être moins précise encore, mais qui permettra d'aborder relativement tôt la définition du processeur et d'approfondir l'étude à partir d'hypothèses plus précises sur la matériel.

Nous avons donc décidé de procéder par estimations. Des statistiques sur l'activité d'un système permettent de déterminer la fréquence des diverses requêtes. Une analyse rapide du traitement permet de déterminer le nombre d'accès disque effectués et la durée du traitement, en retenant l'essentiel, c'est-à-dire les opérations les plus coûteuses en temps (assez faciles à comptabiliser) et en se contentant d'une évaluation sommaire pour le reste. On peut ensuite en déduire, pour une requête, le nombre moyen d'accès disques et la durée moyenne du traitement.

Ces résultats, très approximatifs, ont l'avantage d'être obtenus rapidement. Interprétés avec une marge d'erreur convenable, ils sont suffisants pour valider les choix effectués et fixer l'architecture de MAGE, pour autant que l'architecture adoptée laisse des possibilités d'évolution en cas de sous-estimation de la puissance de traitement nécessaire.

Par ailleurs, ces résultats sont adaptables si l'on garde une trace de tous les pas de calcul. On peut donc mettre en cause un certain nombre d'hypothèses et en particulier, si les résultats obtenus entraînent une remise en question du choix de la technologie des microprocesseurs, l'analyse du traitement effectuée pourra être réutilisée avec la nouvelle technologie adoptée.

VII.2. SUITE DE L'ÉTUDE

Les évaluations effectuées permettront de définir l'architecture de MAGE : nombre de microprocesseurs adoptés, dispositifs de communication, opérateurs externes, structure de l'espace d'adressage. L'approfondissement de l'analyse du logiciel deviendra donc possible, en parallèle avec l'étude du matériel.

Cette analyse devra conduire à une description des algorithmes. Le meilleur moyen de le faire nous semble être l'utilisation d'un langage de haut niveau, ce qui permettra de tester et d'évaluer ces algorithmes par exécution sur un ordinateur de centre de calcul. A partir de là, il sera possible de rechercher des optimisations et d'effectuer des mesures utiles à la conception du matériel, les programmes en langage évolué pouvant être assez facilement modifiés et mis au point.

La conception parallèle du matériel pourra donc être influencée par les résultats obtenus et le logiciel réalisé restera relativement peu dépendant des détails de l'architecture du matériel.

La phase de réalisation qui suivra, ne consistera donc, en ce qui concerne le logiciel, qu'à ré-écrire les programmes d'accès en un langage adapté au processeur utilisé et à écrire les modules de gestion du processeur MAGE. Les algorithmes, testés en langage évolué, devraient être débarassés de la plus grande partie des erreurs algorithmiques et on peut espérer que la mise au point du logiciel ne fera apparaître que des erreurs de programmation. En effet, les tests de MAGE seront faits sur le prototype, de préférence à un simulateur, afin de minimiser les coûts en temps machine. Les aides à la mise au point risquant d'être très réduits, il est intéressant de disposer d'algorithmes déjà éprouvés et optimisés et de pouvoir tester sur des programmes écrits en langage évolué toutes les nouvelles modifications, avant de les effectuer sur les programmes du prototype. Par ailleurs, il sera possible de définir à l'avance la structure optimale des programmes et la méthodologie de réalisation.

CHAPITRE VIII

ESTIMATION
DES NOMBRES D'ACCÈS DISQUE

Le traitement effectué pour chaque requête, et donc le nombre d'accès disque, dépend essentiellement de trois facteurs : le type de la requête, le mode et le type d'élément adressé. L'analyse du traitement permet de déterminer le nombre d'accès disque en fonction de ces facteurs.

Pour pouvoir en déduire le nombre moyen d'accès disque, nous aurons ensuite besoin de la fréquence de chaque facteur. Pour cela, nous utiliserons notre "système type" SCRIB.

Parmi les fonctions de SCRIB programmées en SOCRATE, certaines sont utilisées très fréquemment en période de forte charge du système et sont représentatives de l'activité du système dans ces conditions. Ces fonctions ont donc été transcrites du langage SOCRATE dans le langage MAGE. Les programmes obtenus sont présentés dans le rapport final du contrat SESORI 74.136 [SE].

Les conditions moyennes d'utilisation de ces programmes, fournies par la SAGEM, ont permis de déterminer, pour chaque requête, un coefficient représentatif de sa fréquence d'exécution. On peut donc en déduire la fréquence des différentes combinaisons des paramètres.

L'efficacité de MAGE dans les conditions ainsi déterminées est évidemment une condition nécessaire pour que MAGE soit performant. Si on considère par ailleurs qu'il s'agit de l'activité en forte charge du système le plus complexe de la gamme envisagée, on peut considérer qu'il s'agit des conditions les plus difficiles que MAGE puisse rencontrer et estimer que la condition est suffisante.

Ceci étant admis, il n'en reste pas moins que les résultats obtenus devront être interprétés avec prudence et qu'une large marge d'incertitude devra être gardée.

VIII.1. FRÉQUENCE DES COMBINAISONS

L'analyse du traitement a montré qu'il comporte deux parties :

. l'adressage qui dépend du type de la donnée, mais ne dépend pas du mode d'accès dans la plupart des cas,

. l'accès qui dépend du mode, mais ne dépend pas du type de la donnée dans la plupart des cas.

On peut donc déterminer la fréquence de chaque combinaison "requête - type" et de chaque combinaison "requête - mode".

Le nombre d'accès disque nécessaire pour chacune des deux phases du traitement sera ensuite déterminé en deux temps : le nombre d'accès logiques nécessaires au traitement, puis le nombre d'accès obtenu en faisant intervenir les techniques d'optimisation envisageables.

VIII.1.1. Fréquences en fonction du mode

	RIEN	LIRE	ECRIRE	CREER	TOTAL
OUVRIR	1,7				1,7
FERMER	1,7				1,7
APPEL	14,2	12,4	3	9,7	39,3
FRERE	0,8	1,2	2,7	0	4,7
IDEM		0,6	0,3		0,9
INIT	6,6	0	0	0	6,6
SUIVANT	13,5	0	0	0	13,5
RETOUR	26,6				26,6
NUMDE	4,9				4,9
TOTAL	70,1	14,2	6	9,7	100

Certains modes (SUPPRIMER, INSERER), ou certaines requêtes (MONTER), n'apparaissent pas dans ce tableau, car n'étant pas utilisés dans les programmes considérés, on n'a pas de fréquence à leur affecter. Leur absence ne devrait pas notablement influencer les résultats.

VIII.1.2. Fréquence en fonction du type de donnée

			références						
	CS	ENTITE	LCR	ECR	TOTAL	BLOCS	INDEX	INDIF*	TOTAL
OUVRIR	-	-	-	-	-	-	-	1,7	1,7
FERMER	-	-	-	-	-	-	-	1,7	1,7
APPEL	10,3	15,4	9,2	2,4	11,6	0,6	1,5	0	39,3
FRERE	2,1	0	1,1	1,4	2,5	0	0	-	4,7
IDEM	0	0,9	0	0	0	0	0	0	0,9
INIT	-	6,3	0	0	0	-	0,3	-	6,6
SUIVANT	-	13,2	0	0	0	-	0,3	-	13,5
RETOUR	-	-	-	-	-	-	-	26,6	26,6
NUMDE	-	4,9	-	-	-	-	-	-	4,9
TOTAL	12,4	40,7	10,2	3,8	14,0	0,6	2,1	30,2	100

* INDIF concerne les requêtes pour lesquelles le type n'intervient pas.

VIII.2. ESTIMATION DU NOMBRE MOYEN D'ACCÈS DISQUE PAR REQUÊTE

VIII.2.1. Accès logique

Dans un premier temps, nous avons estimé le nombre d'accès logiques nécessaires à l'exécution des requêtes, c'est-à-dire le nombre d'accès aux fichiers, sans tenir compte des possibilités d'optimisation, soit localement par exploitation des mémoires tampon et des données éventuellement lues lors des requêtes précédentes, soit globalement en utilisant, par exemple, des techniques de cache.

Les estimations ont été effectuées pour chaque fichier ; la technique employée est décrite en Annexe II.

Pour 100 requêtes, les résultats sont les suivants :

structure : 88,8 accès
dictionnaire : 74,1 accès
données : 65,6 accès

TOTAL : 228,5 accès pour 100 requêtes, soit environ 2,3 accès par requête.

Si on rapporte ce chiffre aux seules requêtes susceptibles de générer des accès disque, cela fait 7,6 accès aux fichiers par requête. Ce nombre est élevé, aussi devons-nous nous efforcer de déterminer dans quelle mesure des optimisations peuvent l'améliorer.

Etant donné que les trois fichiers ont des structures très différentes, il semble préférable d'étudier individuellement les optimisations à apporter sur chaque fichier, plutôt que d'envisager une optimisation globale.

VIII.2.2. Optimisation des nombres d'accès

VIII.2.2.1. Structure

88,8 accès sont effectués sur ce fichier pour 100 requêtes. Examinons la répartition des accès par type d'élément :

	nombre d'accès	pourcentage d'accès	nombre d'enregistrements	pourcentage du fichier
entités	43,7	49,2	21	15
références	14	15,8	18	13
anneaux	14	15,8	18	13
index	2,1	2,3	2	1,4
CS et blocs	15	16,9	82	57,6
total	88,8	100	141	100

Les références et anneaux sont généralement accédés ensemble dans les mêmes requêtes. Or, si on examine les enregistrements de structure correspondants, on s'aperçoit qu'on peut, sans modifier leur longueur, les rendre redondants de manière à n'avoir à accéder qu'un seul des deux enregistrements lorsqu'on accède une référence ou un anneau. Ceci permettrait d'économiser 14 accès à la structure qui ne serait plus accédée que 74,8 fois pour 100 requêtes. Le pourcentage d'accès aux différents types d'enregistrement devient alors :

entités : 58,4 %
références : 18,7
anneaux : 0
index : 2,8
CS et blocs : 20,1

Les entités, références et index sont concernés par 79,9 % des accès à la structure, alors qu'ils ne représentent que 41 enregistrements, soit 29,1 % du fichier structure.

On pourrait donc, au démarrage d'une session, amener en mémoire les enregistrements de structure correspondants aux entités, références et index. Une table de moins de 1 K octet, partagée entre tous les utilisateurs, y suffirait et le nombre d'accès disque sur la structure ne serait plus que de 15 pour 100 requêtes (accès aux caractéristiques simples, aux anneaux et aux blocs).

Une telle solution présente plusieurs avantages sur une mémoire cache :

- le gain en nombre d'accès est important et certain,
- la mise en oeuvre est simple : la table étant remplie une fois pour toutes, il n'y a pas de problèmes de choix de l'entrée à remplacer que l'on rencontre dans les mémoires cache. La table pourra être triée de façon à permettre un accès par dichotomie.

L'efficacité de cette méthode n'est pas liée au système SCRIB, puisque les éléments retenus en mémoire ne sont autres que les noeuds de la structure qui ont nécessairement une grande fréquence d'accès et sont en nombre réduit. Dans un système plus simple que SCRIB, on aura une structure moins complexe, avec moins de noeuds, donc un tableau plus petit, même si le nombre de caractéristiques simples reste élevé.

Si on dispose, au niveau de la méthode d'accès, d'une mémoire tampon de 1 K octet environ pour la structure, elle contiendra en permanence un tiers de la structure, ce qui nous donne une chance sur trois d'y trouver l'information souhaitée, et ceci sans aucune hypothèse sur la localité des références, ou la possibilité de grouper les requêtes pour pouvoir en servir plusieurs en un seul accès.

On arrive donc à 15 accès sur le fichier (sur les caractéristiques simples et les blocs), moins 33 % : **10 accès à la structure pour 100 requêtes.**

La faible taille de la structure (moins de 3 K) et la diminution du coût des mémoires, nous suggèrent une autre solution : placer la structure entière dans un tableau en mémoire. Une telle possibilité permettrait d'économiser des mémoires tampon, le tableau des noeuds de la structure et une grande partie des programmes d'accès à la structure, si bien que le coût ne serait peut-être pas beaucoup plus élevé que dans la solution précédente. La seule raison de ne pas effectuer ce choix est a priori qu'elle représente une limitation sur la taille de la structure et qu'elle cesse d'être applicable si on désire gérer plusieurs bases avec des structures différentes sur le même processeur : dans ce cas, la taille de l'ensemble des fichiers structure peut être trop grande pour autoriser leur stockage en mémoire.

Cependant, on doit pouvoir s'en tirer en utilisant une technique de pagination ou de cache. Le chiffre de 10 accès précédemment évolué doit donc être considéré comme un maximum.

VIII.2.2.2. Dictionnaire

Dans nos estimations de nombre d'accès au dictionnaire, nous avons considéré que tout accès à une information entraîne un accès au dictionnaire.

Exemple :

```
entité 10 000 PERSONNE ; début ;
   CS NOM 20 ;
   CS ADRESSE 40 tableau 2 ;
   CS DATE DE NAISSANCE 10 ;
fin ;
```

L'accès à chaque caractéristique de personne 1 entraîne les requêtes suivantes :

APPEL (rien, 0, PERSONNE, 1, IDC, IDU) ;
APPEL (lire, DONNEES, NOM, 0, IDC, IDU) ;
FRERE (lire, DONNEES, ADRESSE, 1, IDC, IDU) ;
FRERE (lire, DONNEES, ADRESSE, 2, IDC, IDU) ;
FRERE (lire, DONNEES, DATE DE NAISSANCE, 0, IDC, IDU) ;

D'après nos estimations, cette succession de requêtes entraînerait quatre accès au dictionnaire. Or, ces quatre accès sont effectués sur la même entité et si une entrée dictionnaire est mémorisée dans le contexte, seul le premier accès sera nécessaire.

L'incidence sur le nombre d'accès au dictionnaire a été évaluée en reprenant les programmes de SCRIB et en comptant un accès dictionnaire par séquence d'accès à une même réalisation d'entité. On a pu compter ainsi 126,4 accès au dictionnaire pour 339 requêtes, soit **37,3 accès pour 100 requêtes.**

Il n'est guère possible d'approfondir l'étude des optimisations possibles avec les moyens dont on dispose : en particulier, le gain obtenu en mémorisant plusieurs entrées dictionnaire dans le contexte d'un utilisateur, ou en utilisant un cache, ne peuvent pas être estimés avec les statistiques dont on dispose.

Une autre possibilité est de placer le dictionnaire sur un disque à têtes fixes, beaucoup plus rapide : cela permettrait de gagner à peu près un rapport 5 sur les temps d'accès. Dans le même ordre d'idées, on peut aussi penser, à plus longue échéance, à l'utilisation des nouvelles technologies de mémoire, telles que les mémoires à bulles.

Une telle solution peut être intéressante pour des systèmes où les temps de réponse doivent être très réduits, mais on ne la retiendra pas dans le cas général pour plusieurs raisons :

. le coût, qui est beaucoup plus élevé que celui d'un seul disque à bras mobile, surtout si la capacité de la base est élevée (pour 100 mega-octets il faudrait un disque à têtes fixes d'une capacité de 20 mega-octets) ;

. la complexité que cela entraîne au niveau des modules (matériel et logiciel) d'accès au disque ;

. la nécessité d'avoir un disque adapté à la taille du dictionnaire et les difficultés que peuvent provoquer les débordements, ou l'utilisation de la place excédentaire si le dictionnaire ne remplit pas tout le disque.

Une telle solution reste pourtant envisageable au cours d'évolutions ultérieures du projet. L'évolution de la technologie des mémoires secondaires et la compétition entre disques et mémoires à bulles, peuvent faire apparaître de nouvelles possibilités.

VIII.2.2.3. Les données

On peut placer dans les contextes des utilisateurs, une zone pouvant contenir une réalisation (256 octets suffisent). De même que pour le dictionnaire, des accès successifs à une même réalisation pourront être effectués en n'accédant qu'une seule fois au fichier des données. Le nombre d'accès obtenu, évalué de la même façon que pour le dictionnaire, est de 121,2 pour 339 requêtes. Soit 35,8 accès disque pour 100 requêtes.

VIII.2.2.4. Total

Le nombre total d'accès disque pour 100 requêtes est donc le suivant :

structure : 10 accès
dictionnaire : 37,3 accès
données 35,8 accès

TOTAL : **83,1 accès pour 100 requêtes.**

On a gagné 145,4 accès sur les 228,5 estimés en premier lieu, soit un gain relatif de 64 %.

Il reste en moyenne 2,8 accès disque par accès aux données. Ce gain est obtenu au prix de 1 K de mémoire pour la structure, une entrée dictionnaire par contexte (10 octets) et une zone de 256 octets par contexte.

Le nombre d'accès à la structure et au dictionnaire peut encore être amélioré, mais l'accès aux données n'a que peu de chances de pouvoir être encore amélioré

De toutes façons, ces optimisations ne pourraient être évaluées que par mesure sur un prototype, ce qui n'est pas possible pour l'instant. Nous devons donc nous contenter de ces chiffres, en essayant d'évaluer le coefficient de sécurité qu'on doit respecter dans leur utilisation.

CHAPITRE IX

EVALUATION DES TEMPS DE TRAITEMENT

Le nombre d'échanges disque, évalué au chapitre précédent, va déterminer, en fonction des unités de disque utilisées, le temps dont dispose le processeur MAGE pour effectuer le traitement.

Nous allons maintenant évaluer la durée du traitement et la répartition des temps de traitement par catégorie d'opération, dans l'hypothèse où un seul microprocesseur MC 6800 est utilisé.

Les résultats, comparés avec la durée des échanges disque, nous permettront de déterminer si un seul microprocesseur suffit et sinon, quelle solution adopter : plusieurs microprocesseurs, opérateurs externes, microprocesseur plus puissant, ou même remise en question de l'utilisation des microprocesse

Une première étape de travail va consister à dresser un catalogue des opérations les plus longues à exécuter sur un MC 6800 (opérations sur chaînes d'octets, multiplications, divisions, ..) et à évaluer leur durée. Nous pourrons ensuite évaluer la durée de quelques opérations fréquentes dans MAGE (calculs, recherches, ..).

Une analyse du traitement de chaque requête nous permettra ensuite d'évaluer la fréquence de chaque grosse opération et d'effectuer une estimation sommaire du reste du traitement. On en déduira la durée moyenne du traitement de chaque opération.

Connaissant la fréquence moyenne des appels de chaque type de requête, on en déduit facilement la durée moyenne du traitement d'une requête.

IX.1. DURÉE DE QUELQUES OPÉRATIONS SUR MC 6800

IX.1.1. Aperçu du MC 6800

Le MC 6800 est un microprocesseur 8 bits, ce qui signifie qu'il peut échanger en parallèle 8 bits avec la mémoire et que la majeure partie des opérations se fait sur un octet. Si des opérations portent sur plusieurs octets, elles auront donc une certaine complexité et seront lentes à exécuter.

Le MC 6800 normal est piloté par une horloge externe dont la fréquence maximale est de 1 MHz. Comme cette fréquence peut varier en fonction des desiderata des utilisateurs, on a coutume de calculer la durée des opérations en cycles d'horloge.

Il existe quatre types d'adressage des opérandes en mémoire :

. l'adressage immédiat, où l'opérande se trouve dans l'instruction ;

. l'adressage "direct" qui opère sur la zone d'adresse 0 à 255 ; l'adresse tient donc sur un octet, ce qui permet des instructions courtes (tenant sur 2 octets) et rapides (puisque économisant un cycle de lecture) ;

. l'adressage "étendu" qui permet d'adresser directement toute la mémoire (65536 octets) grâce à une adresse de deux octets contenue dans l'instruction ;

. l'adressage "indexé", où l'adresse est obtenue par addition du contenu d'un registre index de 16 bits et d'un déplacement de 8 bits contenu dans l'instruction. L'instruction est plus courte que pour l'adressage étendu, mais son exécution est plus longue. L'adressage indexé est le seul moyen d'accéder à des données dont l'adresse n'est pas fixée a priori (si on exclue qu'un programme puisse se modifier lui-même). Les opérations possibles sur l'index sont le chargement, le rangement en mémoire, l'incrémentation, la décrémentation, la comparaison et l'échange avec le pointeur de pile.

Deux accumulateurs d'un octet, notés A et B, sont disponibles pour les opérations. Le registre "pointeur de pile" permet une gestion de pile à l'aide d'instructions d'empilage et de dépilage. La pile étant utilisée par les interruptions et les appels de sous-programmes, son utilisation à d'autres fins est délicate et peut nécessiter le masquage des interruptions. On peut faire sur le pointeur de pile les mêmes opérations que sur l'index.

Il n'existe que deux sortes d'interruptions : masquables et non masquables, avec empilage automatique des registres.

Dans l'évaluation des temps de traitement, on considère que les programmes ne sont pas automodifiables, puisque certains devront être réentrants et que beaucoup pourront être implantés en mémoire morte.

Nous allons donner ci-après, l'évaluation de la durée d'une comparaison. Pour les autres opérations, l'évaluation est donnée en annexe 3 et nous ne donnerons que les résultats.

IX.1.2. Comparaison de chaînes d'octets

Si la chaîne est de longueur faible et constante, on peut effectuer séquentiellement la comparaison de chacun des octets. Dans ces conditions, l'opération prend de 10 à 14 cycles par octet, selon l'adressage utilisé.

Si la chaîne est longue, ou si sa longueur est variable, on doit faire une boucle en utilisant l'adressage indexé.

En utilisant le pointeur de pile, on peut effectuer l'opération en 21 cycles par octet comparé, sinon il faudra à chaque itération charger successivement l'index avec l'adresse de chaque opérande et l'opération prendra *46 cycles par octet comparé.*

Programme de comparaison en assembleur MC 6800

```
00105                    ********************************************
00106                    *  COMPARAISON DE CHAINES D'OCTETS          *
00107                    *  1ERE ADRESSE DANS AD1                    *
00108                    *  2EME ADRESSE DANS AD2                    *
00109                    *  LONGUEUR DANS B                          *
00110                    *  RESULTAT DANS CCR                        *
00111                    ********************************************

00113P 0069 DE 00     N CMPXX  LDX     AD1       ADRESSE 1ER OPERANDE
00114P 006D A6 00     A        LDAA    0,X       1 OCTET -> A
00115P 006D 08                 INX               OCTET SUIVANT
00116P 006E DF 00     N        STX     AD1       SAUVEGARDER ADRESSE 1
00117P 0070 DE 02     N        LDX     AD2       ADRESSE 2EME OPERANDE
00118P 0072 A1 00     A        CMPA    0,X       COMPARAISON
00119P 0074 26 00 007E         BNE     FFCXX     SI INEGALITE: FIN
00120P 007G 07                 TPA               SAUVEGARDER CCR
00121P 0077 08                 INX               OCTET SUIVANT
00122P 0078 DF 02     N        STX     AD2       SAUVEGARDER ADRESSE 2
00123P 007A 5A                 DECB              DECREMENTATION LONGUEUR
00124P 007B 26 EC 0069         BNE     CMPXX     SI NON NUL ON CONTINUE
00125P 007D 06                 TAP               RESTAURATION CCR
00126P 007E 39           FFCXX RTS
```

IX.1.3. Autres opérations

Transferts de données :

Octet par octet : 43 cycles par octet.
Deux octets par deux octets : 32 cycles par octet.

Addition :

De 45 à 69 cycles selon l'adressage utilisé.

Multiplication :

4 octets par deux octets : 1 500 cycles.
4 octets par un octet : 900 cycles.
Si un opérande est petit, il peut être plus rapide d'effectuer la multiplication par additions successives. Si le premier opérande fait 4 octets, cela est vrai si le second opérande est inférieur ou égal à 10. Si le premier opérande fait 2 octets, cela est vrai si le second opérande est inférieur ou égal à 40. Enfin, une multiplication par une constante peut utiliser un algorithme particulier si elle est fréquente. Exemple :
multiplication par 10 : $10 \times A = 4 \times 2 \times A + 2 \times A$
ce qui s'effectue en trois décalages et une addition.

Division :

1200 cycles en moyenne. Un tableau des durées en fonction de la longueur des opérandes est donné en annexe.
Si la valeur maximale du quotient est faible, on peut avoir intérêt à effectuer la division par soustractions successives. Pour une division de 1 octet par 1 octet, la durée est de 24 + 12 x quotient. Ceci est avantageux si le quotient ne dépasse pas 48.

Recherche séquentielle dans une table :

Pour N itérations, la durée est de 45 x (N - 1) + 13 x LCH + 5 ; LCH étant la longueur de la chaîne à rechercher.

Recherche séquentielle sur une liste chaînée :

Pour N itérations, la durée est de 23 x (N - 1) + 13 x LCH + 5.

Recherche par dichotomie :

Pour N itérations, la durée de l'opération est :
21 + TIND + TCOMP + N (37 + TIND + TCOMP) où TIND est la durée de l'indexation et TCOMP la durée de la comparaison. Pour des entrées de deux octets, nous avons TIND = 39 cycles et TCOMP = 18 cycles (programme en annexe), ce qui donne, d'après la formule précédente : 78 + N x 94.

Insertion en tête de liste chaînée :

27 cycles.

Insertion en queue de liste chaînée :

50 cycles.

Extraction d'une liste chaînée :

27 cycles.

IX.2. TRAITEMENT EFFECTUÉ PAR MAGE

Le traitement effectué par MAGE comporte plusieurs types d'opérations :

- réception et mise en attente de la requête,
- sélection des requêtes à traiter et aiguillage vers le programme de traitement,
- accès aux fichiers,
- exclusions mutuelles,
- échanges avec l'utilisateur,
- transmission de la fin de requête à l'utilisateur,
- traitement des requêtes proprement dit.

Les temps de traitement seront répartis en quatre catégories :

- échanges avec l'utilisateur,
- opérations arithmétiques (multiplication et division),
- transferts de données de mémoire à mémoire,
- opérations diverses (tout ce qui reste).

Ceci nous permettra de voir si une catégorie représente une fraction particulièrement importante du traitement, auquel cas elle pourra peut-être être facilitée à l'aide d'un opérateur spécial.

IX.2.1. Réception de la requête

Une requête se présente comme une chaîne de 10 octets. La réception va consister à échanger ces 10 octets en provenance d'un autre processeur et à les placer dans un contexte, sans se préoccuper de leur contenu. L'algorithme est le suivant :

1. interruption ;
2. réception des 10 octets dans une mémoire tampon ;
3. si code de la requête = OUVRIR alors aller à OUVRIR ;
4. rechercher le contexte concerné ;
5. ranger la requête dans le contexte ;
6. retour au point d'interruption.

L'interruption proprement dite dure 12 cycles. Quelques cycles peuvent être nécessaires pour reconnaître la source de l'interruption, selon le matériel adopté. Nous compterons, par approximation, une dizaine de cycles pour cela.

La réception des 10 octets va dépendre fortement du type du processeur utilisateur, du protocole de communication et du matériel utilisé pour la connexion. Dans un système tel que CORAIL [PO], diverses tables doivent être accédées et le message est précédé d'un préambule de 6 octets qui doit être ré-expédié par le processeur destinataire. Ces opérations et le contrôle de parité longitudinale, peuvent prendre très approximativement 500 cycles.

La réception de la requête proprement dite se fait au rythme de 24 cycles par octet (échanges synchrones programmés).

Une dizaine de cycles sont nécessaires pour tester le cas de OUVRIR.

La recherche du contexte est une recherche séquentielle ; les contextes peuvent être chaînés entre eux pour accélérer l'accès. Pour 16 contextes, il faut en moyenne 8 comparaisons de 2 octets : cela dure environ 210 cycles. Un transfert de 8 octets est ensuite nécessaire pour déposer la requête dans le contexte (les numéros de contexte et d'utilisateur n'ayant pas besoin d'être ré-écrits).

Le retour au point d'interruption se fait automatiquement avec l'instruction RTI (9 cycles).

Durée totale :

- échanges avec l'utilisateur : 10 x 24 = 240 cycles
- divers = 740 cycles
- transfert de mémoire à mémoire : 8 x 32 = 256 cycles

TOTAL (arrondi) = *1 250 cycles*

IX.2.2. Accès aux fichiers

Cette partie est traitée en détail dans l'annexe III § 4.

Pour la structure, l'accès se décompose en accès logique comportant les recherches en mémoire et ayant lieu à chaque accès, et en accès physique comportant l'accès au disque et n'ayant lieu que si les recherches en mémoire ont échoué.

Accès logique : opérations diverses : 744 cycles
transferts : 18 octets : 576 cycles
TOTAL (arrondi) : *1 350 cycles*
Accès physique : opérations diverses : *500 cycles*

Accès au dictionnaire :

multiplications, divisions :	5 000 cycles
divers (arrondi) :	1 100 cycles
TOTAL :	*6 100 cycles*

Les accès aux données dépendent du mode d'accès puisque, dans le cas de l'écriture, il n'est pas nécessaire de localiser la donnée dans le buffer, cette opération ayant été effectuée lors de la lecture préliminaire.

Lecture :

multiplications, divisions :	1 600 cycles
opérations diverses :	1 160 cycles
transferts : 256 octets :	8 192 cycles
TOTAL (arrondi) :	*11 000 cycles*

Ecriture :

Opérations diverses :	500 cycles
opérations arithmétiques :	1 600 cycles
transferts : 256 octets :	8 192 cycles
TOTAL (arrondi) :	*10 300 cycles*

IX.2.3. Exclusions mutuelles

Chaque ressource en cours d'accès aura un descripteur contenant :

. le nom de la ressource (nom interne),

. le parallélisme employé, c'est-à-dire le nombre d'utilisateurs en cours (noté D). Cette valeur aura un maximum, noté DMAX, qui est le degré de multiprogrammation de la ressource : il sera généralement égal au nombre de contextes.

. une tête de liste des contextes en attente.

La ressource dictionnaire étant unique, elle ne posera pas de problème. Pour les réalisations, par contre, il faudra une table dans laquelle on effecte une entrée à toute nouvelle réalisation accédée. Le détail des évaluations est donné en annexe III § 5.

Durée totale (arrondie):

	réalisation	dictionnaire
MAJ	400	300
LECT	400	250
NOMAJ	250	500
NOLECT	200	50

IX.2.4. Communications avec l'utilisateur

Les échanges de données peuvent se faire de manière synchrone à raison de 24 cycles par octet, après une procédure d'initialisation de 500 cycles.

La fin du traitement est signalée par l'émission d'un certain nombre d'informations : identification du contexte et de l'utilisateur, identification de la requête, code condition, etc ..

Si 10 octets sont transmis, l'opération devrait durer environ 740 cycles.

IX.2.5. Sélection d'une requête et aiguillage

Si les requêtes en attente sont stockées dans les contexte, la sélection d'une requête consiste à chercher, parmi les contextes, le premier qui possède une requête en attente de traitement : c'est une recherche séquentielle sur liste chaînée. Si 4 contextes doivent être parcourus en moyenne, il faudra : 23 x 3 + 18 = 87 cycles.

Le décodage et l'aiguillage vers le traitement de la requête se font en utilisant le code de la requête comme index d'un tableau contenant les adresses des programmes. Le programme exécutant cette opération est donné en annexe. Il prend 56 cycles.

La durée totale de ces opérations est donc environ 150 cycles.

IX.2.6. Traitement des requêtes

Le schéma général du traitement d'une requête est le suivant :

1. vérification,
2. aiguillage vers le traitement correspondant au type de l'élément adressé,
3. aiguillage vers le traitement correspondant au mode d'accès demandé,
4. fin de l'opération.

Nous ne détaillerons pas ici le traitement et l'évaluation de sa durée pour chaque cas, puisque MAGE comporte environ trente modules. Nous démontrerons donc, sur un exemple simple, la façon dont les temps de traitement ont été estimés, pour donner ensuite la durée moyenne du traitement.

a) Etude de APPEL

Algorithme :

1. vérifier que la pile ne soit pas saturée,
2. vérifier le mode ($1 \leq MODE \leq 8$)
3. lire l'enregistrement de structure demandé,
4. vérifier que l'enregistrement lu soit bien fils de l'enregistrement précédemment accédé,
5. appeler le traitement correspondant au type,
6. appeler le traitement correspondant au mode,
7. fin du traitement.

Les deux premières opérations sont de simples comparaisons qui durent, à elles deux, 30 cycles.

L'accès logique à la structure demande 1 350 cycles (l'accès physique sera compté à part).

La troisième opération consiste à vérifier que le chaînage père de l'élément de structure accédé soit égal au numéro de l'enregistrement de structure adressé par le sommet de la pile : cette comparaison s'effectue en 36 cycles. L'aiguillage vers le traitement du type et du mode peut se faire de la même façon que pour le type de la requête (décrit au § IX.3.5). Ces opérations prennent donc à elles deux 112 cycles.

La fin du traitement va consister à envoyer à l'utilisateur le compte-rendu de fin d'opération, mais cette opération est comptabilisée séparément. Il va aussi falloir inscrire dans le contexte que le traitement de la requête est terminé : on ne peut pas évaluer précisément la durée de cette opération sans faire d'hypothèses sur l'organisation du système. On peut estimer que ce sera équivalent au positionnement de 4 ou 5 octets, ce qui peut prendre jusqu'à 40 cycles.

La durée totale de l'opération se décompose donc comme suit :

transferts de données : 18 x 32 = 576 cycles
opérations diverses : 970 cycles
TOTAL (arrondi) : *1 550 cycles*

b) Etude de l'appel sur ENTITE

Algorithme :

1. contrôle du numéro de réalisation,
2. calcul du numéro absolu de réalisation,
3. calcul du nom interne,
4. empilage,
5. retour.

Le contrôle du numéro de réalisation est une comparaison de deux octets destinée à vérifier que le numéro est bien inférieur ou égal à la valeur maximale : cela prend 36 cycles.

Le numéro absolu de réalisation est donné par la formule :

NAR = NARENG x NBR + NR - 1

où NARENG est le numéro absolu de réalisation de l'entité englobante contenu dans le sommet de la pile, NBR est le nombre maximum de réalisations fourni par la structure, NR est le numéro de réalisation fourni par la requête. Ce calcul va demander 66 cycles pour mettre en place les opérandes, une multiplication, soit de 900 à 1 500 cycles (1 200 cycles en supposant que NR a 50 % de chances d'avoir une longueur de 1 octet), deux additions qui font 54 cycles chacune.

Durée totale : multiplications, divisions (arrondi) : 1 200 cycles
divers (arrondi) : 350 cycles
TOTAL : *1 750 cycles*

c) Etude de LIRE

Algorithme :

1. LECT (DICTIONNAIRE) ;
2. LECT (réalisation) ;
3. lecture dictionnaire
4. lecture réalisation
5. calcul du déplacement de la donnée dans la réalisation
6. transmission des données à l'utilisateur
7. NOLECT (réalisatio) ;
8. NOLECT (DICTIONNAIRE) ;
9. retour.

Les opérations 1 et 2 ont été évaluées précédemment à 400 et 250 cycles. Les accès aux fichiers sont comptés séparément, mais il faut compter une centaine de cycles pour les appels et passages de paramètres (en comptant 4 octets pour chaque programme).

Le calcul du déplacement comporte quelques transferts d'octets et additions. Il dure environ 30 cycles.

La transmission des données a été étudiée précédemment ; la procédure d'initialisation peut durer 500 cycles et l'échange dure 24 cycles par octet. On peut estimer la longueur moyenne d'un échange (d'après l'étude de SCRIB) à 10 octets, soit 240 cycles d'échange.

Les opérations 7 et 8 durent 200 et 50 cycles en moyenne.

Durée totale (arrondie) :	opérations diverses :	1 550 cycles
	échanges avec l'utilisateur : 10 x 24 =	240 cycles
	TOTAL :	*1 800 cycles*

d) Durée moyenne du traitement d'une requête

La durée des autres opérations est évaluée de la même façon en cherchant à évaluer la durée d'une opération d'autant plus précisément que sa fréquence d'exécution est plus grande.

La durée moyenne du traitement d'une requête est ensuite déterminée à l'aide des fréquences d'exécution. On obtient les résultats suivants :

opérations diverses :	1 960 cycles
calculs (multiplications, divisions) :	1 390 cycles
échanges en mémoire : 19 x 32 :	608 cycles
échanges avec l'utilisateur : 6 x 24 :	144 cycles
TOTAL (arrondi) :	*4 102 cycles*

IX.2.7. Durée moyenne du traitement par requête

Durée pour 100 requêtes :

Opération	fréquence	divers	échanges utilisat.	transferts	calculs	total
réception	100	74 000	24 000	25 600	-	123 600
Sélection aiguillage	100	15 000	-	-	-	15 000
Accès structure (physique)	10	5 000	-	-	-	5 000
Accès dictionnaire	37	40 700	-	-	185 000	225 700
Accès données	36	41 800	-	295 000	57 600	394 400
Fin de requête	100	50 000	24 000	-	_	74 000
Traitement	100	196 000	14 400	60 800	139 000	410 000
		422 500	62 400	381 400	381 600	1247 900

Durée moyenne par requête (arrondie) :

opérations diverses :	4 250	34 %
échanges avec l'utilisateur :	650	5,2 %
transferts en mémoire :	3 800	30,4 %
multiplications, divisions :	3 800	30,4 %
TOTAL :	*12 500*	*100 %*

Les transferts de mémoire à mémoire représentent environ 119 octets par requête, sans compter les transferts courts (de 1 à 4 octets) qui sont exécutés séquentiellement.

Les échanges avec l'utilisateur représentent environ 27 octets par requête.

Pour une fréquence d'horloge de 1 MHz, le temps de traitement d'une requête sera de 12,5 ms.

IX.3. CRITIQUE DES RÉSULTATS

Il n'est guère possible d'évaluer de façon objective la précision de ces résultats, mais il est tout de même intéressant d'en étudier les différentes sources d'imprécision, afin de voir quelle confiance peut leur être accordée.

. Un certain nombre de charges n'ont pas été prises en compte, ou n'ont été que sommairement évaluées. C'est le cas des communications avec l'utilisateur ou avec le disque, qui dépendent fortement du matériel utilisé ; on s'est donc contenté d'une estimation sommaire. C'est aussi le cas d'un certain nombre de travaux liés à la gestion du système et qui ne peuvent être évalués sans faire d'hypothèses sur la structure matérielle et logicielle du système, et il est nécessaire d'en tenir compte. De même, les traitements liés aux interruptions, aux exclusions mutuelles, à la gestion des contextes, n'ont pu être bien étudiés.

. L'analyse qui a servi de base aux estimations n'a pas été très approfondie et les algorithmes n'ont pas été testés. Il y a donc un risque non négligeable d'avoir oublié des opérations relativement importantes. Par ailleurs, il y a le risque d'avoir surestimé la durée de certaines opérations qui peuvent être programmées d'une façon beaucoup plus efficace que celle qui était envisagée. La partie la plus complexe du traitement total est le traitement des requêtes, c'est donc celle qui a été le moins finement analysée et dont l'estimation de durée risque d'avoir la plus grosse erreur. Cette partie ne faisant que 33 % du traitement total, même une très grosse erreur ne peut avoir que des répercussions relativement faibles sur l'ensemble, puisqu'elle serait divisée par trois.

. Un certain nombre de calculs ont été faits pour évaluer, par exemple, la longueur moyenne d'une liste, le nombre d'itérations entraînées par la recherche, ... Ces calculs ont souvent été faits à partir d'hypothèses simplificatrices, ce qui est générateur d'imprécision. On a souvent cherché à majorer les hypothèses, de façon à arriver plutôt à une surestimation qu'à une sousestimation du temps de traitement ; par ailleurs, les résultats ont été très souvent arrondis à la valeur supérieure.

. Les estimations ont été faites à partir de programmes de SCRIB, écrits en langage MAGE, et de statistiques de la SAGEM sur l'utilisation de SCRIB. Il y a donc plusieurs sources d'erreur sur les fréquences d'utilisation des requêtes. Le système SCRIB ne représente peut-être pas le pire des cas que MAGE puisse avoir à traiter, certains cas d'utilisation pouvant peut-être entraîner une fréquence très différente des diverses requêtes et influer lourdement sur la charge. Enfin, la programmation de SCRIB en MAGE, effectuée à partir de programmes SOCRATE, est certainement très différente de celle qui serait réalisée directement à partir d'un cahier des charges. On a donc un certain risque de sousestimation des temps de traitement dûe à ces statistiques.

Il faut donc interpréter les résultats obtenus avec une très grande prudence : une marge d'incertitude de 50 % ne semble pas exagérée. Etant donnée cette incertitude, il peut arriver qu'on ne puisse pas tirer de conclusions. Dans ce cas, il faudra reprendre les estimations en approfondissant l'analyse et en effectuant quelques hypothèses sur la structure matérielle et logicielle du système.

CHAPITRE X

CONCEPTION DE L'ARCHITECTURE DE MAGE

Au chapitre précédent nous avons déterminé que la durée moyenne du traitement pouvait se situer aux alentours de 12 500 cycles, soit 12,5 msec avec un MC 6800 normal.

La durée d'un accès disque sur CDC 9760 peut être ramenée à 29 ms avec une optimisation de bras sur une file d'attente de 5 requêtes, ce qui est relativement optimiste. Enfin, nous avons évalué à 0,83 le nombre moyen d'accès disque par requête, ce qui fait donc 24 ms.

La possibilité de réaliser MAGE à l'aide de microprocesseurs semble donc incontestable, mais il serait peut-être hatif de conclure qu'on peut le réaliser avec un seul MC 6800 normal. Les chiffres estimés conduisent à un taux de charge de 52 %. Nous avons vu que, bien que les chiffres aient été systématiquement majorés, il subsiste un risque de sous-estimation des temps de traitement, les algorithmes n'ayant pas été mis au point. La charge de gestion du système n'a pas été prise en compte et enfin, le nombre d'accès disque peut être encore amélioré : le stockage de la structure en mémoire et l'utilisation d'un disque fixe pour le dictionnaire, peuvent diminuer la durée des accès disque de 50 %, ce qui amène la charge à un taux excessif.

Le taux de charge couramment atteint sur un microprocesseur est de l'ordre de 25 % et la charge de gestion augmente en général beaucoup en fonction de la charge : un système sophistiqué est généralement nécessaire à l'obtention d'un taux de charge élevé et on est très mal équipé sur un microprocesseur pour le réaliser : faible puissance des interruptions, adressage rendant difficile et coûteuse la réalisation de programmes réentrants, ... Il est donc très probable dans ces conditions qu'un taux de charge supérieur à 50 % sera excessif et qu'il pourra entraîner des attentes du disque par mauvaise utilisation du processeur.

Le processeur de base de données sera concerné par deux sources d'interruption : l'utilisateur et le disque.

L'interruption en elle-même dure 12 cycles, le retour d'interruption coûte 10 cycles. De plus, un seul niveau d'interruption est disponible, les interruptions non masquables (NMI) devant être réservées à des conditions très particulières : coupures d'alimentation, défaillances, aide à la mise au point, .. Toutes les sources d'interruption sont donc regroupées sur une seule ligne et une reconnaissance logicielle de l'origine de l'interruption doit être effectuée préalablement à tout traitement, d'où perte de temps supplémentaire.

Par ailleurs, chacune des sources demande un traitement rapide : l'utilisateur pour ne pas le ralentir inutilement ni engorger le réseau d'interconnexion dans le cas d'un système réparti, et le disque pour permettre son utilisation optimale puisqu'il émettra une interruption en fin d'opération et attendra alors que l'on relance une nouvelle opération. Il y a donc un risque de conflit entre ces deux sources d'interruption, risque agravé s'il existe dans les programmes des sections nécessitant le masquage des interruptions (exclusions mutuelles par exemple). Ces risques de conflit sont d'autant plus élevés que l'on a un processeur unique fortement chargé.

On désire aussi avoir un traitement relativement rapide, afin d'entretenir une file d'attente du disque suffisamment longue pour permettre une bonne optimisation des déplacements de bras, et le traitement des requêtes ne doit pas ralentir notablement le temps de réponse, en particulier pour les requêtes qui n'entraînent pas d'accès disque.

D'un point de vue matériel, il faut remarquer que la recherche d'un taux d'utilisation élevé d'un microprocesseur peut être très coûteuse en mémoire par la taille des programmes de gestion du processeur et des tables associées, par la recherche d'une durée d'exécution optimale des diverses opérations qui se fait souvent au détriment de l'encombrement en mémoire (cf. algorithme de division). Or, 4 K bits de mémoire coûtent actuellement aussi cher qu'un microprocesseur. Il n'est donc pas forcément intéressant d'économiser les processeurs, ce qui va à l'encontre des principes couramment admis dans des systèmes où le processeur est la ressource chère. Ce problème de coût constitue un point important dans la conception des systèmes à base de microprocesseurs, puisque dans les choix matériels, il faudra considérer non seulement les coûts des éléments employés, mais aussi l'incidence sur le logiciel, une simplification du logiciel pouvant compenser largement le coût d'un opérateur, par l'économie de mémoire effectuée.

Nous recherchons donc une puissance de calcul notablement supérieure à celle d'un seul MC 6800.

Trois types de solutions sont envisageables et seront étudiées :

. utiliser un microprocesseur plus puissant, ou mieux adapté,

. accélérer certaines opérations à l'aide d'opérateurs externes,

. utiliser plusieurs microprocesseurs.

X.1. ETUDE DES DIFFÉRENTES SOLUTIONS

X.1.1. Utilisation d'un microprocesseur plus puissant ou mieux adapté

C'est a priori la solution la plus simple et la plus économique.

Une première possibilité consiste à utiliser un microprocesseur travaillant sur 16 bits. Ceci permet un gain important sur les opérations arithmétiques, ainsi que sur les transferts de données qui peuvent se faire par deux octets.

Il est également apparu, dans la famille des microprocesseurs 8 bits, une nouvelle génération plus évoluée dont le représentant type est le ZILOG Z 80. Ces microprocesseurs possèdent des facilités d'adressage améliorées et des instructions beaucoup plus puissantes : registres de base, index multiples, opérations de transfert en mémoire, multiplications, ... L'utilisation de tels microprocesseurs peut très facilement permettre de gagner un rapport de deux dans les temps de traitement et d'obtenir en outre une diminution du volume des programmes.

L'inconvénient de cette solution est la remise en question du choix du MC 6800 au profit d'un microprocesseur pour lequel on a moins d'expérience et de moyens logiciels et matériels de mise en oeuvre, les impératifs industriels de la SAGEM rendant par ailleurs nettement préférable une solution à base de MC 6800.

L'apparition future d'un microprocesseur de la nouvelle génération, compatible avec le MC 6800, peut cependant remettre en question cet argument.

X.1.2. Utilisation d'opérateurs externes

Une autre façon d'accélérer le traitement, consiste à placer dans l'espace d'adressage d'un microprocesseur, un ou plusieurs opérateurs, de manière à accélérer soit l'ensemble du traitement, soit certaines opérations fréquentes

Nous avons vu que les multiplications et divisions occupent 30 % du temps de traitement. Le tableau du § IX.2.6. montre que 64 % de ces opérations sont effectuées dans les accès aux fichiers dictionnaire et données. L'adressage est donc relativement peu en cause. On pourrait chercher à modifier la conception du système de manière à diminuer le nombre des multiplications et divisions ; mais en fait, il n'est pas certain que l'on arrive à se passer de ces opérations pour l'accès au disque, puisqu'on ne désire pas remettre en cause l'adressage physique du disque : la conversion des adresses disque représente une charge très importante. Il n'est guère admissible de traiter les adresses disque directement sous forme d'adresses physiques, car cela rendrait le système trop dépendant de la configuration de la mémoire secondaire. Par ailleurs, il est vraisemblable qu'une tentative de diminution des opérations arithmétiques amènerait une augmentation d'autres opérations, telles que les comparaisons pour lesquelles on n'est guère mieux équipé.

On peut donc envisager l'utilisation d'opérateurs de multiplication et division sur 32 bits.

Les transferts de données en mémoire occupent également 30 % du temps. Il est tout à fait possible d'utiliser un opérateur de transfert de données en mémoire : deux circuits d'accès direct à la mémoire (DMA) pourraient permettre de transférer un octet par cycle d'horloge. La durée moyenne des transferts de données tombe ainsi à moins de 200 cycles par requête, la durée de traitement d'une requête passant à 8 900 cycles.

Un certain nombre d'améliorations de l'adressage (registre de base externe, index externe, pagination, ..) peuvent être envisagées et permettraient d'accélérer l'ensemble du traitement, tout en simplifiant l'écriture des programmes.

L'introduction d'opérateurs externes permet donc un gain notable en rapidité. Cependant, leur multiplication risque d'augmenter très vite la complexité du processeur et de nous amener à une solution qui ne serait peut-être pas très concurrencielle vis à vis d'une réalisation centralisée à base de macro-composants.

X.1.3. Utilisation de plusieurs microprocesseurs

La dernière possibilité consiste à utiliser plusieurs microprocesseurs exécutant en parallèle le traitement des requêtes.

On peut utiliser deux processeurs banalisés exécutant indifféremment n'importe quelle partie du traitement. Cela peut permettre une utilisation optimale des processeurs, mais pose de gros problèmes : logiciels en ce qui concerne la gestion et la synchronisation des processeurs, et matériels en ce qui concerne le partage des mémoires, en particulier celle des programmes dont le taux d'accès est très élevé.

Il paraît donc plus sain de spécialiser chaque processeur dans une partie du traitement.

Une telle solution permettrait de placer l'utilisateur et le disque dans les espaces d'adressage de deux processeurs différents, ce qui est l'avantage le plus important de la solution utilisant plusieurs microprocesseurs. Les sources d'interruption seraient ainsi réparties entre les deux processeurs, ce qui permet d'éliminer tout risque de conflit et de concilier la gestion optimale du disque et un délai de réponse minimal aux demandes de communication de l'utilisateur.

La communication entre les processeurs peut se faire par échange de messages ou par mémoires communes.

La première solution peut entraîner une charge très importante si ces communications sont fréquentes, aussi la seconde solution nous paraît-elle préférable malgré les problèmes liés au partage de mémoire, problèmes qui peuvent être allégés si le taux d'accès à cette mémoire n'est pas très élevé.

X.2. COMPARAISON DES SOLUTIONS

Quatre critères de comparaison peuvent être adoptés : le coût, les possibilités d'évolution, la sécurité, l'adaptation au problème à traiter.

X.2.1. Coût

Ce critère est relativement secondaire : l'élément le plus coûteux du système sera vraisemblablement le disque, aussi l'incidence relative de l'architecture sur le coût total du système sera-t-elle faible.

La solution utilisant un microprocesseur plus puissant est certainement la moins coûteuse : ce microprocesseur ne serait certainement pas plus coûteux qu'un MC 6800 et des opérateurs externes, ou que deux MC 6800 et un dispositif de partage de mémoire. On peut également espérer obtenir des programmes plus compacts, grâce à la puissance des instructions, d'où gain sur le volume de la mémoire. Il est aussi à craindre que l'utilisation de plusieurs microprocesseurs n'entraîne des pertes de mémoire par fragmentation et par duplication de programmes. Un partage judicieux du travail entre les processeurs peut minimiser cet inconvénient.

X.2.1. Possibilités d'évolution

Les technologies utilisées dans ce projet sont en pleine évolution et le taux de charge peut en être affecté. En particulier, les progrès de la technologie des mémoires de masse peuvent faire apparaître la possibilité de stocker des parties de la base sur un support beaucoup plus rapide (structure, cache sur le dictionnaire, ..) et d'accélérer considérablement les accès disque.

Le temps disponible pour effectuer le traitement peut donc diminuer, alors même que ce traitement peut augmenter par l'introduction de nouvelles fonctions d'optimisation des entrées-sorties.

Le choix de l'architecture doit donc présenter une certaine souplesse pour qu'on puisse, sans le remettre en question, augmenter encore la puissance du traitement.

A cet égard, l'emploi de plusieurs microprocesseurs semble la solution la plus intéressante. D'une part cette architecture ne fixe absolument pas le choix d'un microprocesseur et il devrait être possible de revenir sur ce choix sans voir apparaître de nouveaux problèmes ; d'autre part, on peut étudier d'emblée une configuration maximale en prévoyant l'implantation d'opérateurs externes, quitte à les supprimer par la suite s'ils sont superflus.

X.2.3. Sécurité

Du point de vue de la fiabilité, il semble qu'un seul processeur dans une architecture simple soit la meilleure solution.

Le problème essentiel est cependant la sécurité des informations et, si une panne survient, il faut qu'elle endommage aussi peu que possible les informations. En particulier, un fonctionnement défectueux doit être détecté très vite pour éviter ou limiter les pertes d'information et la propagation des erreurs.

L'utilisation de plusieurs microprocesseurs est intéressante à cet égard, car la répartition du traitement devrait limiter la propagation des erreurs et chaque processeur peut détecter un mauvais fonctionnement de l'autre, ne serait-ce qu'en vérifiant la cohérence des informations échangées.

En cas de panne, il est probable qu'un des deux processeurs restera opérationnel et pourra peut-être être utilisé pour le diagnostic de la panne. Enfin, une conception plus modulaire du processeur devrait faciliter les opérations de maintenance.

X.2.4. Adaptation au problème à traiter

De l'architecture adoptée dépendra dans une large mesure la facilité avec laquelle pourra être conçu et réalisé le logiciel et donc son efficacité et sa fiabilité.

La communication avec l'utilisateur et le contrôle du disque entraînent deux types d'interruption qui doivent être traitées avec le maximum de rapidité : l'utilisation de deux microprocesseurs semble répondre assez bien à ce problème

Un processeur pourra être connecté à l'utilisateur : il partagera son temps entre les échanges avec l'utilisateur et une partie du traitement. L'autre processeur serait connecté au disque et partagerait son temps entre le reste du traitement et les échanges avec le disque. On devrait donc pouvoir obtenir un maximum d'efficacité dans le traitement des deux sources d'interruption. En adoptant une technique de communication asynchrone entre les deux processeurs, chaque processeur n'aurait qu'une source d'interruption, d'où une grande simplification du traitement des interruptions et de la gestion du processeur.

Toutes ces raisons nous amènent à préférer une architecture utilisant deux microprocesseurs.

X.3. ARCHITECTURE DE M A G E

Le traitement de MAGE sera réparti entre deux microprocesseurs P1 et P2. P1 possèdera, dans son espace d'adressage, l'interface de communication avec l'utilisateur. P2 communiquera avec le processeur disque. Un troisième microprocesseur, que nous noterons P3, pourra être introduit au niveau du processeur disque pour exécuter certaines fonctions très proches du disque : gestion de la file d'attente des requêtes avec optimisation des déplacements de bras, conversion des adresses, contrôle du coupleur disque. Chaque processeur possède sa mémoire de programmes et sa mémoire de travail.

Un certain nombre de problèmes logiciels et matériels se posent. Nous allons ici étudier les solutions logicielles et exposer plus brièvement l'aspect matériel qui a été étudié en détail par Phillippe NAVAUX.

X.3.1. Communication entre P1 et P2

Le problème se situe à deux niveaux : définir un mécanisme logiciel de communication et résoudre les problèmes matériels du partage des mémoires.

X.3.1.1. Mécanisme logiciel

Nous avons dit, dans la définition des langages d'accès, que tout utilisateur peut définir un ou plusieurs contextes par l'intermédiaire desquels il peut accéder la base. Un contexte est représenté par une zone de mémoire de 512 octets, contenant en particulier :

- une pile nécessaire au parcours de la structure (70 octets),
- une zone de données (256 octets) pouvant contenir une ou plusieurs réalisations d'entités,

. une ou plusieurs entrées dictionnaire (m x 7 octets),
. un certain nombre de variables de travail.

Chaque contexte regroupe donc toutes les données nécessaires au traitement des requêtes.

La solution au problème logiciel de la communication entre P1 et P2 semble être de "commuter" logiquement les contextes entre les deux processeurs, en fonction des opérations à effectuer.

Dans chaque contexte, un octet permet de connaître son état qui peut être :
.0. non affecté,
.1. inactif : aucune requête n'est en cours de traitement ou en attente sur un contexte,
.2. en attente de traitement par P1,
.3. en attente de traitement par P2,
.4. en attente d'accès disque,
.5. en attente sur exclusion mutuelle,
etc ..

Les états "traités par P1" et "traités par P2" n'ont pas besoin d'être matérialisés par une valeur particulière de l'octet d'état.

Lorsqu'un utilisateur effectue un OUVRIR, un contexte pris parmi ceux dont l'état est "non affecté" lui est alloué et est initialisé. Le contexte passe ensuite à l'état "inactif".

Lorsqu'une requête arrive, le contexte concerné passe à l'état "en attente de traitement par P1", puis "traité par P1" lorsque ce processeur prend en compte le travail à effectuer sur le contexte.

Si l'intervention de P2 est nécessaire, le contexte est mis dans l'état "en attente de traitement par P2", puis "traité par P2" lorsqu'il est pris en compte par ce processeur.

Lorsqu'un accès disque est nécessaire, P2 place le contexte dans l'état "en attente d'accès disque" dont il ne sortira qu'à la fin de l'opération pour revenir à l'état "en attente de traitement par P2". De même, si le contexte est mis en attente sur exclusion mutuelle, il reviendra en attente de traitement lorsque la ressource sera libérée.

Lorsque P2 termine son travail sur un contexte, il le remet dans l'état "en attente de traitement par P1".

Lorsque P1 termine le traitement d'une requête, il remet le contexte dans l'état "inactif".

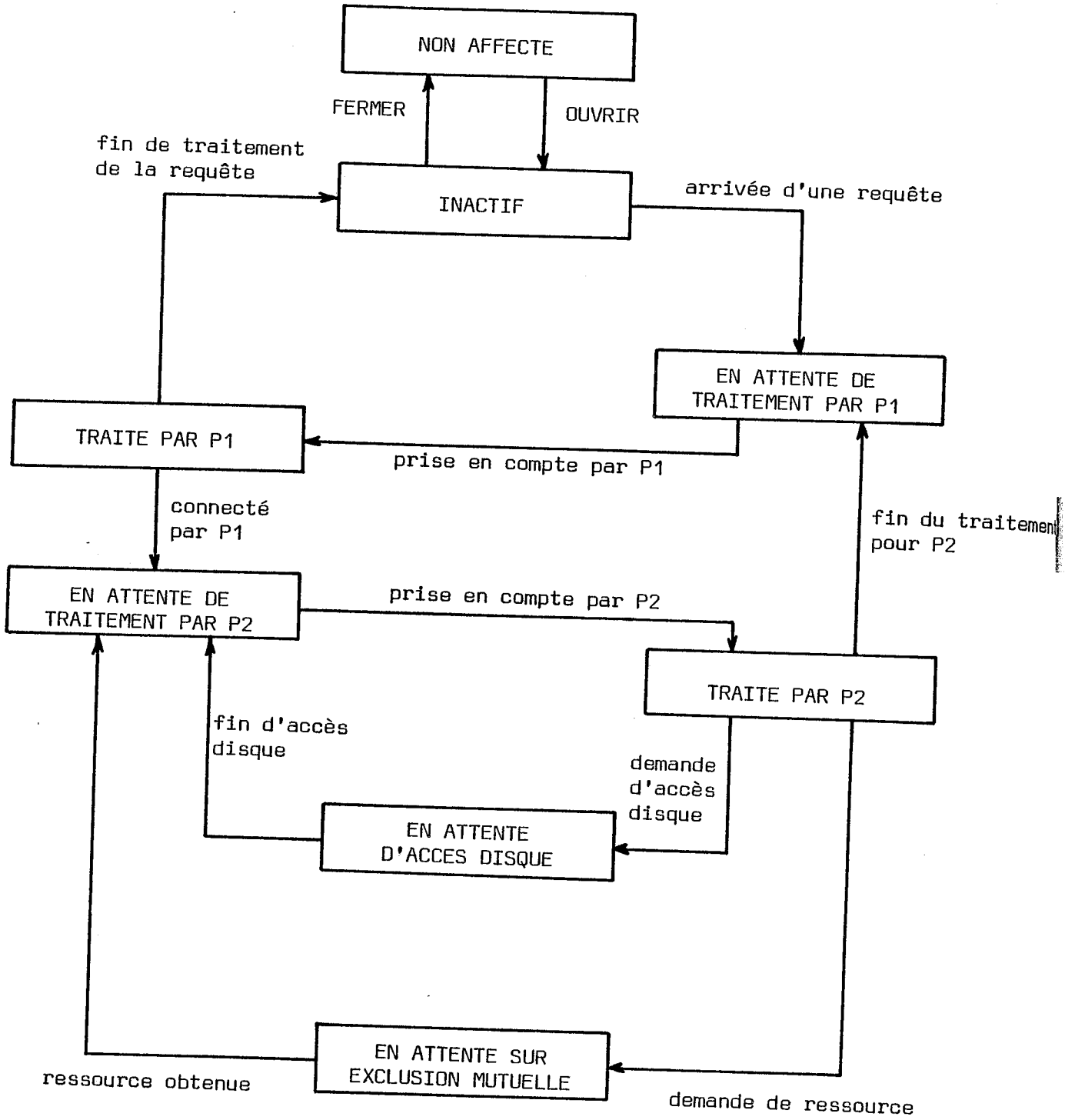
Enfin, l'exécution d'un FERMER sur un contexte le remet dans l'état "non affecté".

Le passage d'un contexte d'un processeur à l'autre se fait donc de la façon suivante :

1) sauvegarde dans le contexte des informations éventuellement présentes, dans la mémoire de travail du processeur ;
2) positionnement d'un octet dans le contexte pour indiquer l'action à effectuer ;
3) mise à la valeur nécessaire (2 ou 3) de l'état du contexte ;
4) recherche d'un autre contexte en attente de traitement.

La durée de la première opération peut être minimisée, ou même annulée, par un choix judicieux des informations gardées dans le contexte et par un partage optimal du traitement entre les deux processeurs.

Diagramme de transition des états d'un contexte :



La deuxième et la troisième opération sont très rapides. La quatrième opération consiste à appeler l'allocateur de travaux du processus. Celui-ci va parcourir les contextes jusqu'à ce qu'il en trouve un en attente de traitement. Si aucun contexte n'est en attente, la recherche se poursuit jusqu'à ce qu'un nouveau contexte ait été mis en attente, soit par la fin d'un échange disque, soit par l'arrivée d'une requête, soit par l'autre processeur. Si aucune requête n'est en cours, la recherche se poursuivra donc jusqu'à ce qu'un utilisateur émette une requête.

X.3.1.2. Partage de mémoire

Pour assurer leur rôle de communication, les contextes sont regroupés dans une mémoire appelée "mémoire des contextes", partagée entre P1 et P2.

Le partage de mémoire n'est pas prévu au niveau du microprocesseur MC 6800 et il est nécessaire de réaliser un dispositif externe de partage de la mémoire.

Une solution matérielle, basée sur le principe du "vol de cycle" a été trouvée, en tenant compte du fait que le taux d'accès aux contextes par les processeurs est compris entre 6,3 et 12 % : il n'y a donc aucun inconvénient à ralentir un des deux processeurs en cas de conflit.

Par ailleurs, la réalisation d'un registre de base externe, permettant d'accéder aux contextes en adressage étendu avec des adresses fixes, a été proposée et semble très intéressante [NA].

X.3.2. Communications avec l'utilisateur

Comme nous l'avons vu, le processeur P1 possèdera dans son espace d'adressage les dispositifs de communication avec l'utilisateur. Ces communications sont de quatre sortes :

. réception des requêtes,
. réception des données,
. émission des données
. signal de fin de requête.

X.3.2.1. Point de vue logiciel

La réception des requêtes est un type de communication très particulier : c'est le seul qui ne soit pas déclenché par MAGE lui-même. La requête est émise par l'utilisateur et le processeur P1 doit en être informé par une interruption.

Lorsqu'il est interrompu, P1 reçoit la requête (d'une façon qui dépend de la solution matérielle adoptée), la place dans le contexte correspondant qui est mis dans l'état "en attente de traitement par P1", puis il reprend son travail au point où il a été interrompu. Si la requête est un OUVRIR, il faut d'abord allouer un contexte.

L'utilisateur étant, pour P1, la seule source d'interruption, le traitement sera très simplifié. Par ailleurs, l'interface utilisateur étant unique, il n'y a, en principe, aucun risque de voir le traitement d'une interruption lui-même interrompu.

Les autres types de communication sont déclenchés à l'initiative de MAGE par les programmes de traitement des requêtes. Ce sont donc, pour P1, de simples échanges entre la mémoire et un périphérique, éventuellement précédé d'un protocole d'établissement de la communication (selon le type d'interface matériel utilisé). Ces échanges utilisant l'interface de communication, ils ne pourront pas être interrompus et de plus l'interface est nécessairement libre puisque le processeur est en train d'effectuer le traitement d'une requête (pas de requête en cours d'arrivée). Il n'y a donc pas de problème d'exclusion mutuelle sur l'interface.

Pour les échanges de données, MAGE devra transmettre l'identification du contexte et de l'utilisateur, l'adresse des données fournie par la requête et la longueur des données qu'il aura lui-même déterminée.

En fin de traitement d'une requête, le message transmis reproduira la requête traitée et contiendra des informations sur le déroulement de l'opération (code condition).

X.3.2.2. Point de vue matériel

Il est très difficile de spécifier l'interface utilisateur, puisque MAGE doit pouvoir s'adapter à au moins deux types de processeurs ayant des spécifications d'interface très différentes.

On devra donc concevoir un interface très modulaire, tant au niveau matériel que logiciel, pour permettre son inter-changeabilité en fonction du processeur utilisateur.

Si le processeur utilisateur est un réseau de microprocesseurs, l'interface matériel sera sans doute très simple (un circuit d'interfaces périphériques), mais l'interface logiciel devra se conformer aux conventions d'échange du réseau. Les échanges pourront être programmés.

Le problème est beaucoup plus complexe si l'utilisateur est un mini-ordinateur et il dépendra du type de connexion adopté (coupleur spécifique, adaptation d'un coupleur existant, liaison programmée). Les échanges devront peut-être être effectués par un automate spécial pour tenir des cadences plus rapides.

A titre d'exemple, l'utilisation de la liaison programmée sur l'ULP SAGEM nécessite un temps de réponse du périphérique inférieur à 3 microsecondes, ce qui est impossible à un MC 6800. On doit donc avoir un interface assez évolué disposant de mémoires tampon, capable de reconnaître les signaux émis par l'ULP et d'émettre les signaux de validation nécessaires.

X.3.3. Contrôle et gestion du disque

Ces fonctions donneront vraisemblablement lieu à la partie la plus complexe de l'architecture de MAGE.

Le contrôle du disque ne peut pas être effectué directement par un microprocesseur vues les cadences de transfert. Il faudra donc disposer d'un coupleur rapide qui peut être piloté par un microprocesseur : ce microprocesseur noté P3, a été évoqué au début du § X.3. Il avait en fait la charge de l'aspect "temps réel" du contrôle du disque et quelques fonctions (conversions d'adresses, gestion des requêtes, ..) qui en faisaient un "processeur fichier" très élémentaire, permettant en particulier un adressage "logique" des informations, qui pourrait être sous la forme : numéro de fichier, numéro de secteur dans le fichier. En se chargeant de la partie lente du contrôle du disque ce processeur devrait permettre de simplifier le contrôleur matériel ; en permettant un adressage logique de l'espace disque, il devrait rendre P2 totalement indépendant de la configuration de la mémoire secondaire. Il permet une utilisation optimale du disque, lui étant entièrement consacré, enfin moyennant une programmation relativement simple, il décharge P2 d'une partie de son travail.

La communication entre P2 et le contrôleur de disque peut se faire par le partage de mémoires tampon et une petite mémoire partagée pour les requêtes.

Le problème du partage des mémoires tampon entre P2 et le processeur disque est très différent de celui du partage des contextes. La cadence de transfert du disque n'autorise pas un partage au niveau du cycle, mais on peut envisager de suspendre l'activité de P2 par l'intermédiaire de la broche HALT du microprocesseur lorsqu'il y a coïncidence entre un échange disque et l'accès aux mémoires tampon par P2. L'accès aux mémoires tampon par P2 peut être signalé par un indicateur positionné par logiciel, puisque P2 n'accède aux mémoires tampon que dans des sections de programme bien définies.

X.3.4. Répartition du travail entre P1 et P2

Une répartition précise et définitive ne peut évidemment pas être déterminée dès maintenant. On peut cependant fixer un certain nombre de principes qui devront guider cette répartition.

Un découpage fonctionnel du traitement devra être effectué en sorte de répartir le traitement.

Le processeur P3 exécutera la fonction PGDISQ décrite au § VI.3.

La fonction PGE (gestion des échanges avec l'utilisateur) sera exécutée par P1.

Pour le reste, il faudra chercher à équilibrer la charge entre P1 et P2, tout en cherchant à garder à un niveau raisonnable le coût des communications entre les deux processeurs. Il faut également éviter, ou minimiser, les duplications d'information, de programmes ou d'opérateurs matériels.

D'une façon générale, P2 devrait exécuter PAST, PADIC, PARE (accès à la structure, au dictionnaire et aux données) qui nécessitent l'accès aux mémoires tampon du disque, et PNI (accès par noms internes).

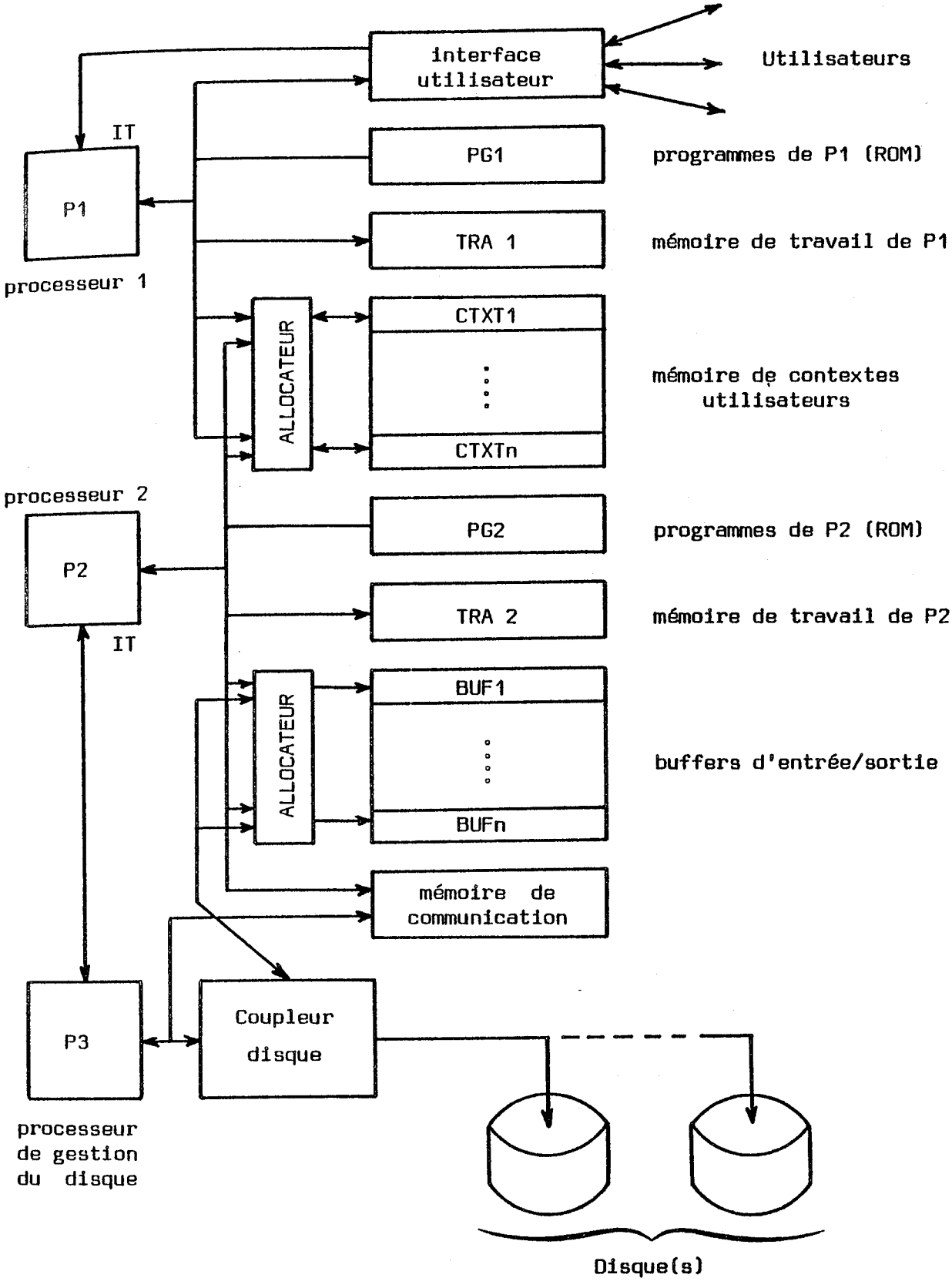
PREQ devrait être exécuté par P1. Si cette fonction représente une part trop grande du traitement, elle devra être redécomposée pour être répartie entre P1 et P2.

Cette répartition regroupe dans P2 la plus grande partie des transferts de mémoire à mémoire (échanges entre buffers et contextes), mais les opérations arithmétiques sont plus dispersées : multiplications dans P1 (calculs des noms internes), divisions dans P2 et P3 (hash-coding sur le dictionnaire, conversions d'adresses). La répartition des opérations arithmétiques devrait toutefois permettre d'utiliser dans chaque processeur les algorithmes les mieux adaptés à la configuration des opérandes, plutôt que des algorithmes généraux assez coûteux.

Le regroupement des transferts de données dans P2 laisse des possibilités d'accélérer encore le traitement par adoption d'opérateurs matériels.

X.4. CONCLUSION

Le paragraphe précédent a permis de définir une architecture répondant assez bien aux critères du paragraphe 2. Les divers problèmes évoqués ont été étudiés par Philippe NAVAUX qui effectue la partie matérielle de cette étude [NA]. Ils peuvent recevoir une solution simple à condition de ne pas rechercher à tout prix l'utilisation maximale des microprocesseurs. C'est là un point important : le microprocesseur est trop économique pour qu'il soit intéressant d'en optimiser l'utilisation et il est souvent préférable d'en multiplier le nombre.


interface utilisateur
Utilisateurs
IT
P1
processeur 1
PG1
programmes de P1 (ROM)
TRA 1
mémoire de travail de P1
ALLOCATEUR
CTXT1
CTXTn
mémoire de contextes utilisateurs
processeur 2
P2
IT
PG2
programmes de P2 (ROM)
TRA 2
mémoire de travail de P2
ALLOCATEUR
BUF1
BUFn
buffers d'entrée/sortie
mémoire de communication
P3
processeur de gestion du disque
Coupleur disque
Disque(s)

CHAPITRE XI

SUITE DE L'ETUDE

Le projet MAGE se poursuit actuellement avec la réalisation d'un prototype. Les aspects logiciels et matériels restent très liés, mais ma contribution personnelle est plutôt axée sur la réalisation du logiciel, en liaison étroite avec la réalisation du matériel.

L'étude du logiciel a été poursuivie avec l'écriture en CPL/1 des algorithmes d'accès à la base. Ces programmes sont encore incomplets et aucune optimisation n'a été implémentée. Les buts essentiels de ce travail étaient les suivants :

. approfondir et vérifier l'analyse des principaux algorithmes d'accès,
. fournir un moyen de mesures et d'évaluations, afin d'optimiser les algorithmes et de guider la conception du matériel,
. déterminer une méthodologie d'écriture du logiciel sur le prototype de MAGE.

XI.1. MÉTHODOLOGIE DE RÉALISATION DU LOGICIEL

Deux particularités du projet MAGE risquent de rendre la réalisation du logiciel relativement différente de la réalisation des logiciels habituels :

. l'utilisation des microprocesseurs,
. la spécialisation de MAGE.

Il est donc nécessaire de voir si ces faits ne peuvent pas remettre en question les techniques et les compromis couramment adoptés dans les systèmes classiques.

XI.1.1. Utilisation des microprocesseurs

Dans l'état actuel de la technologie, MAGE représente une application extrême des microprocesseurs, dont l'utilisation la plus courante actuellement consiste à contrôler des téléimprimeurs ou à réaliser des automatismes.

La taille maximale des programmes habituels des microprocesseurs est de l'ordre de quelques K octets, alors qu'on s'attend à avoir environ 20 K octets de programme dans MAGE.

Il y a un changement d'échelle par rapport aux systèmes classiques : la longueur moyenne d'une instruction est de 2 octets sur le MC 6800, contre 4 octets environ dans les systèmes classiques. Un programme de 256 octets est déjà un gros programme sur un microprocesseur.

Le jeu d'instructions du MC 6800 n'est pas particulièrement adapté à l'écriture et à la mise au point de gros programmes : instructions peu puissantes, à peu près aucune facilité de mise au point (pas d'instructions invalides, pas de détection d'erreurs, pas de protection mémoire, ..), branchements conditionnels en adressage relatif (déplacement de $\pm$ 128 octets par rapport à l'instruction en cours).

Il est donc impératif de structurer une application en un grand nombre de petits sous-programmes faciles à mettre au point.

La faible puissance des instructions fait que des opérations simples et courantes peuvent représenter des programmes de quelques dizaines d'octets : une addition sur 2 octets en adressage étendu représente 19 octets.

Pour diminuer la taille des programmes et faciliter leur écriture et leur mise au point, il va falloir définir un ensemble de sous-programmes destinés à pallier les insuffisances du langage d'instructions : opérations arithmétiques, transferts de données de mémoire à mémoire, aiguillages, ..

L'écriture de MAGE dans ces conditions va nécessiter la définition de conventions de liaison simples, souples et cohérentes, ces conventions ne devant en aucun cas allourdir la programmation ni augmenter de manière sensible l'encombrement des programmes. Le MC 6800 possède un certain nombre de possibilités qui devraient nous y aider : interruptions "software", instructions d'empilage et de dépilage, ..

XI.1.2. Processeurs spécialisés

Dans un système classique, on partage avec d'autres utilisateurs une machine qu'il faut accepter telle qu'elle est. On cherche généralement à optimiser les programmes en temps d'exécution, afin d'utiliser les ressources le moins longtemps possible (donc réduire le coût du traitement) et d'offrir un temps de réponse minimum.

Le problème est différent dans MAGE. La conception du logiciel peut influer sur le matériel ; on aura donc souvent un compromis à rechercher entre une solution matérielle et une solution logicielle. Le matériel est consacré à MAGE et une ressource inoccupée ne peut pas être affectée à une autre tâche. Par ailleurs, le temps de réponse est fixé par les échanges disque et il existe un temps d'exécution des requêtes optimal en fonction de la durée des échanges. Il est donc inutile d'avoir un temps de traitement inférieur au temps optimal. Cela signifie que, ayant minimisé la durée des échanges disque, il est inutile de chercher à minimiser les temps d'exécution s'ils sont inférieurs au temps optimal ; cela ne ferait qu'accroître la durée d'inactivité des ressources. Il faut donc au contraire chercher à réduire le volume des ressources, même au détriment du temps d'exécution, afin de minimiser le coût du matériel.

Parmi les ressources de MAGE, celle qui est la plus coûteuse et que l'on peut minimiser le plus facilement, même en fin de réalisation, est la mémoire. On peut donc s'attacher à rendre les programmes les moins encombrants possibles, au détriment du temps d'exécution, tant qu'il est inférieur au temps optimal. On peut agir de trois façons :

- en remplaçant certains algorithmes par d'autres moins rapides, mais nécessitant moins de mémoire,
- en effectuant à l'aide de sous-programmes le plus grand nombre possible d'actions courantes,
- en remplaçant dans certains programmes l'adressage étendu par l'adressage indexé, plus court mais moins rapide.

Le logiciel d'un processeur spécialisé sera un ensemble statique de programmes, c'est-à-dire que ces programmes sont conçus ensemble de manière cohérente et résideront de manière permanente dans la mémoire du processeur où ils auront des adresses fixes. Ceci supprime donc les problèmes d'édition de lien (qui sera faite une fois pour toutes) et de chargement (qui sera fait, au pire, une seule fois par session). S'il existe des programmes non résidents, leur chargement sera réduit à une lecture disque, l'adresse disque étant connue et l'adresse en mémoire étant fixée. Il n'y a pas d'allocation mémoire pour les programmes et les données sont soit des zones partagées d'adresse fixe, soit des zones de longueur fixe, ce qui rend très simple leur allocation.

Dans un système classique, on trouve généralement un ensemble de programmes appelé logiciel de base, destiné à la gestion des ressources de la machine d'une part, et à fournir un ensemble de services d'autre part (gestion de fichiers, entrées-sorties, allocation mémoire, synchronisation, ..). Ces programmes sont conçus pour être utilisables par la plus grande variété possible d'applications. Il n'y a rien de tel dans un processeur spécialisé tel que MAGE où l'ensemble des programmes peut être considéré comme logiciel de base. La gestion des ressources physiques peut se faire à n'importe quel niveau de la hiérarchie des programmes (pas de programmes privilégiés) et les programmes de plus bas niveau (primitives) seront conçus et optimisés spécifiquement pour la fonction à réaliser.

XI.2. APPLICATION À M A G E

A partir des programmes réalisés en CPL/1, il sera possible d'effectuer des mesures et des estimations, d'une part pour répondre aux questions qui pourraient se poser dans la réalisation du matériel, et d'autre part pour optimiser les algorithmes, en particulier en ce qui concerne les accès disques.

Par ailleurs, l'étude de ces programmes devrait permettre de fixer la structure du logiciel sur MC 6800, la méthodologie d'écriture et l'ensemble des primitives. Le logiciel de MAGE pourra être structuré en trois niveaux principaux :

1) Sous-programmes d'extension du matériel :

Cet ensemble de sous-programmes complètera le jeu d'instructions du microprocesseur hôte, en réalisant des opérations simples et courantes, non effectuées par les instructions du microprocesseur. Le choix de ces opérations et l'optimisation des algorithmes pourront être faits en fonction de MAGE, mais ils ne seront absolument pas spécifiques de cette application, c'est-à-dire que le même ensemble de sous-programmes pourrait être utilisé dans d'autres systèmes. Exemples d'opérations : opérations arithmétiques complexes (multiplications, divisions, additions et soustractions sur plusieurs octets), transferts de données, comparaisons de chaînes d'octets, recherches, aiguillages, ...

2) Primitives MAGE :

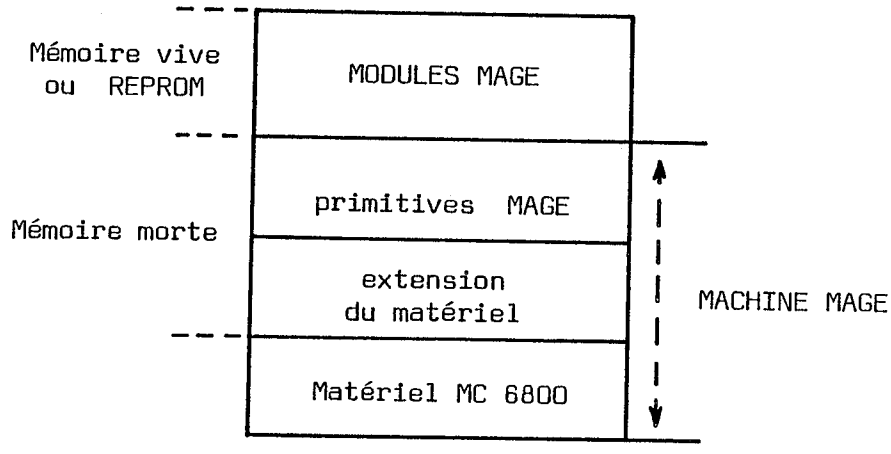
Il s'agit d'un ensemble de sous-programmes spécifiques de MAGE, réalisant des opérations simples et fréquentes dans les modules de MAGE : calculs de noms internes, de longueurs, de déplacements, recherches, exclusions mutuelles, etc ..

Ces opérations devront être indépendantes des éventuelles modifications de MAGE, de manière à n'être pas affectées par l'évolution du produit. Elles doivent être suffisamment simples pour pouvoir être entièrement testées, afin d'avoir la certitude qu'elles ne comportent aucune erreur. Ainsi, sera-t-il possible de regrouper les sous-programmes de niveaux 1 et 2 dans une mémoire morte, afin d'assurer leur protection et d'améliorer la sécurité de MAGE.

Les deux premiers niveaux formeront, avec le matériel, la machine abstraite MAGE.

3) Modules MAGE :

L'exécution de MAGE sera effectuée par un certain nombre de modules qui pourront eux-mêmes se répartir en plusieurs sous-niveaux. La plus grande partie de ces modules devrait être composée d'appels de sous-programmes des niveaux (ou sous-niveaux) inférieurs. Ces programmes, moins sûrs et susceptibles d'être modifiés, seront implantés en mémoire vive, ou en mémoire morte réinscriptible (REPROM).



XI.2.1. Choix du langage de programmation

L'utilisation de CPL/1 pour la réalisation des algorithmes s'est révélée assez mal adaptée pour diverses raisons, tenant en grande partie aux restrictions de ce langage par rapport à PL/1 : pas de tableaux de structure, limites de longueur des chaînes de bits et de caractères, procédures internes sans paramètres, pas de zones de type "offset" (déplacement). Un certain nombre de choses étaient difficilement réalisables, telles la pile des contextes ou les buffers disque. Il était assez difficile de manipuler des données de type bit, ou de manipuler des données indépendamment de leur type (mémoires tampon d'entrées-sorties). Au niveau du langage de description, il aurait fallu un langage beaucoup plus sophistiqué, alors que pour une réalisation, le langage CPL/1 demanderait beaucoup d'insertions de texte en assembleur (sous réserve qu'il soit implanté sur MC 6800).

Le langage utilisé pour la programmation de MAGE devra répondre aux conditions suivantes :

- ne pas gaspiller de place : ce critère reprend toute sa valeur dans un processeur spécialisé où la mémoire n'est pas partagée. La place occupée par les programmes détermine pour une bonne part le volume total de la mémoire du processeur, qui à son tour détermine le coût du matériel ;
- ne pas gaspiller de temps d'exécution, puisque ce temps est limité ;
- permettre de choisir le compromis entre encombrement mémoire et rapidité ;
- permettre une description simple des opérations.

Le logiciel de MAGE sera probablement composé d'un grand nombre de petits sous-programmes. Il est donc nécessaire, pour répondre aux deux premières conditions, d'avoir des conventions de liaison extrêmement simples et souples.

Enfin, dans la mesure où cela est compatible avec les autres critères, il serait souhaitable que le langage permette une programmation modulaire et structurée.

Le langage idéal serait un langage de type langage de programmation de systèmes (PL/360, LP 80, ..) permettant la manipulation des ressources physiques de la machine et l'insertion de textes en assembleur. De tels langages existent pour des microprocesseurs, mais il est peu probable actuellement que l'on puisse en disposer pour réaliser MAGE.

Les sous-programmes de niveau 1 et une bonne partie du niveau 2 devront vraisemblablement être écrits en assembleur. Pour le reste, l'idéal serait certainement la définition d'un langage de macros.

XI.2.2. Conventions de liaison

. Appels de sous-programmes :

Trois instructions peuvent être utilisées pour appeler un sous-programme :

- BSR (branch to subroutine), permet d'appeler un sous-programme en utilisant l'adressage relatif, c'est-à-dire en fournissant un déplacement de $\pm$ 128 octets par rapport à l'adresse de l'instruction en cours. Ce type d'appel ne peut convenir qu'à des sous-programmes locaux et ne sera vraisemblablement pas utilisé dans MAGE.

- JSR (jump to subroutine) permet d'appeler un sous-programme en utilisant l'adressage étendu ou indexé. C'est vraisemblablement le type d'appel qui sera le plus utilisé dans MAGE.

. SWI (software interrupt) permet d'appeler un programme en simulant une interruption. Les registres du microprocesseur sont empilés et on se branche à l'adresse contenue dans les deux octets d'adresse X'FFFA' et X'FFFB'. Ce type d'appel peut être utilisé en mise au point pour effectuer des points d'arrêt, des comptages, ou des déroutements en cas d'anomalie, ou pour appeler certaines opérations du niveau zéro.

Dans MAGE, certains programme pourront être appelés de la façon suivante :

. le programme appelant détermine un numéro désignant le programme qu'il désire appeler, et appelle un module d'aiguillage ;

. le module d'aiguillage recherche dans une table l'adresse du sous-programme correspondant au numéro et se branche à cette adresse. Ce type d'appel peut être utilisé en particulier par l'allocateur des travaux qui disposera, dans le contexte, d'un code de l'opération à effectuer (code de la requête, par exemple).

. Sauvegardes :

Vue la faible taille des registres, il est rare que l'on y conserve des informations au-delà de quelques instructions. Aussi, une sauvegarde systématique des registres par le programme appelé ne semble-t-elle pas nécessaire.

. Paramètres :

Les paramètres pourront être transmis de diverses façons :

. pas de paramètre : le sous-programme travaille sur des données communes (contextes par exemple) ; ce sera certainement le cas le plus fréquent ;
. transmission par les accumulateurs : ceci est possible pour des données de deux octets au maximum ;
. transmission dans l'index de l'adresse des paramètres ;
. empilage des paramètres ;
. utilisation d'une zone commune d'adresse fixe.

Le choix d'une solution pourra être effectué en fonction de l'opération exécutée, du nombre et de la longueur des paramètres, et enfin de leur nature (adresse, chaînes d'octets, nombres, ..).

· Programmes réentrants :

Il est assez difficile d'écrire des programmes réentrants en MC 6800, puisque cela nécessite de travailler sur des mémoires de travail allouées dynamiquement. On doit donc utiliser l'index pour accéder aux variables, ce qui allourdit la programmation et perd beaucoup de temps. L'utilisation de l'adressage direct n'est plus possible, ce qui signifie une perte de temps considérable, particulièrement dans des programmes tels que les programmes de calc Faire exécuter ces programmes sous exclusion mutuelle augmenterait beaucoup la complexité du système.

Le fait que MAGE soit un processeur spécialisé va nous permettre d'organise le système en sorte que seul un très petit nombre de programmes ait besoin d'être réentrant :

- les interruptions sont traitées en mode ininterruptible, ce qui est possible du fait que l'on a plusieurs processeurs et une seule source d'interruption par processeur ;
- le traitement des interruptions ne partage pas de programmes avec le reste du traitement, à moins que ces programmes ne puissent être rendus facilement réentrants ; ceci peut entraîner la duplication de quelques sous-programmes, mais ils ne devraient pas être très importants, car on peut organiser le traitement de façon à les minimiser ;
- après le traitement d'une interruption, le contrôle est toujours rendu au programme interrompu. Le contrôle ne peut être retiré à un processus qu'en des points bien définis : demande d'accès disque, demande de ressource, appel à un autre processeur.

S'il est nécessaire de remettre plus fréquemment en question l'allocation du processeur à un processus, par exemple pour traiter d'éventuelles opérations prioritaires, cela peut être fait en insérant périodiquement dans chaque programme, en des points bien définis, des appels à un module de gestion des travaux qui décidera si le contrôle doit être passé à un autre processus. C'est donc chaque processus qui remet lui-même en question l'allocation du processeur en des points où cela ne présente aucun inconvénient pour le traitement, c'est-à-dire, en particulier, là où toutes les variables sont dans les contextes.

XI.2.3. Suite du projet

La programmation des algorithmes en PL/1 nous permet maintenant de disposer d'une description précise des algorithmes. Cette description sera utilisée pour l'écriture des programmes en assembleur : définition des niveaux 1 et 2, structuration des programmes, .. Elle peut également être utilisée pour étudier et tester d'éventuelles modifications ou optimisations.

Du point de vue logiciel, le niveau 1 et le niveau 2 de MAGE ont été réalisés en assembleur et mis au point sur EXORCISER.

L'étude fonctionnelle du processeur disque a été approfondie et la réalisation du coupleur disque est en cours : connecté à un EXORCISER, ce coupleur permettra la mise au point des programmes d'accès au disque.

Le projet se poursuivra ensuite avec la réalisation de la connexion processeur disque - processeur base de données, puis la réalisation du processeur de base de données proprement dit.

Les programmes seront parallèlement réalisés et mis au point, en partie, sur EXORCISER, en commençant par les algorithmes les plus proches du disqu au fur et à mesure de la réalisation du matériel.

Le projet est donc maintenant dans une étape de réalisation. On prévoit qu prototype de MAGE sera au point d'ici un an. L'industrialisation de MAGE e ensuite prévue par la SAGEM.

Le projet MAGE rentre actuellement dans une phase qui devrait déboucher sur sa réalisation industrielle. Ceci n'a été possible que parce que nous avons limité nos ambitions, quant à l'originalité et la puissance du système, pour introduire des contraintes industrielles dans la conception.

Une contrainte importante dans la conception d'une machine informatique est que celle-ci puisse être produite et vendue, avec un impératif de rentabilité tant pour le constructeur que pour l'utilisateur. Or, cette contrainte est souvent négligée par la recherche universitaire, au profit de contraintes d'esthétique, de fiabilité, de performance, qui sont certainement des sources de satisfaction, mais qui dépassent bien souvent les besoins réels. Il en résulte des projets qui s'arrêtent au stade de la machine papier ou, plus rarement, du prototype, alors que le but réel d'une recherche en architecture des ordinateurs est la production industrielle des résultats qui peut seule justifier et valider le travail effectué. Par ailleurs, lorsqu'on travaille sur une technologie en pleine évolution (ce qui est notre cas), un projet qui n'est pas conduit dès le départ en vue de l'industrialisation, risque fort de se trouver dépassé avant d'avoir trouvé la moindre application

L'utilisation de microprocesseurs, qui à l'origine était un peu un pari, est apparue tout à fait possible et l'évolution de la technologie justifie largement ce choix. En effet, on voit apparaître, à des prix restant très faibles, des microprocesseurs de plus en plus puissants et nous voyons arriver des microprocesseurs d'une puissance équivalente aux miniordinateurs 16 bits actuels. Il deviendra donc de plus en plus intéressant d'utiliser une puissance de traitement aussi bon marché dans les systèmes, soit en rendant "intelligents" les périphériques, soit en concevant des systèmes entièrement à base de microprocesseurs.

Par ailleurs, il conviendra d'être conscients que les principes de concept et de programmation adoptés sur les systèmes classiques, doivent être remis en question lorsqu'il s'agit de microprocesseurs.

La baisse du prix des mémoires centrales et l'apparition des mémoires virtuelles, ont conduit à des techniques de programmation coûteuses en mémoire. De même, les systèmes d'exploitation recherchent souvent l'utilisation optimale du processeur, ressource chère, au prix d'un accroissement du volume de la mémoire. Avec les microprocesseurs, la mémoire est à nouveau la ressource coûteuse du système (après les périphériques toutefois) et la programmation devra minimiser son utilisation, même si c'est au prix d'une utilisation médiocre du processeur.

Il convient également de considérer l'incidence de la spécialisation des systèmes sur la conception et la réalisation du logiciel. Un système à base de microprocesseurs n'est pas un ordinateur en réduction, en ce sens que les rapports de coût des différents éléments sont très différents, ce qui implique des compromis différents. Un système ne doit plus être considéré comme des mémoires et des périphériques au service d'une unité centrale, mais comme des processeurs au service de périphériques.

Une étude plus approfondie et systématique des problèmes de conception et de réalisation de logiciels dans les processeurs spécialisés à base de microprocesseurs, et des langages de programmation, devrait sans doute être effectuée.

Par ailleurs, une remise à jour et une harmonisation de la terminologie devrait être entreprise, la terminologie actuelle étant extrêmement centralisatrice (mémoire centrale, unité centrale, ..). Le vocabulaire ne manque certes pas de possibilités, mais la difficulté est, comme toujours, d'arriver à faire parler le même langage par tous.

Le but de ce projet n'était pas d'apporter des inovations dans le domaine des bases de données et nous nous sommes contentés de reprendre des principes connus, mais en les appliquant à de nouvelles architectures. Nous n'avons pas non plus remis en question les mécanismes physiques d'accès au disque, puisque nous devions utiliser les disques tels qu'ils sont fournis par les fabricants.

Notre effort s'est surtout porté vers la décentralisation de la fonction base de données : définition d'une méthode d'accès adaptée, définition des langages de communication, définition du matériel adapté à son exécution.

Du point de vue du disque, nous pensons arriver à une meilleure utilisation en lui consacrant un processeur, ce qui doit permettre de limiter le délai d'attente entre la fin d'un échange et le lancement de l'échange suivant et d'optimiser les déplacements de bras en ordonnant la file d'attente des requêtes ; de même, on peut attendre une utilisation plus efficace des mémoires tampon locales.

Ce premier essai de décentralisation d'une fonction d'un petit système, devrait ouvrir la voie à la décentralisation d'autres fonctions et à la réalisation de systèmes fonctionnellement répartis ; avec le développement des microprocesseurs, on peut espérer étendre cette demande à des systèmes beaucoup plus puissants.

A l'avenir, dans la conception d'un processeur de base de données, il serait souhaitable de s'intéresser plus aux bases de données relationnelles, dont l'état de l'art a beaucoup avance depuis le début du projet MAGE, ainsi qu'à la répartition des bases de données (gestion décentralisée de bases de données réparties).

De plus, il est difficile de prévoir l'impact des nouvelles technologies (CCD et bulles) [JU] sur la conception des systèmes de gestion de données. Les CCD n'entrent pas réellement en concurrence avec le disque, puisqu'il s'agit de mémoires volatiles, mais les bulles fournissent des mémoires non volatiles, beaucoup plus rapides que les disques. L'utilisation qui en sera faite dans les systèmes de gestion de données, dépendra du prix de revient au bit qu'elles atteindront et du volume de mémoire qui pourra être commodément obtenu.

On peut envisager leur utilisation dans de vastes mémoires cache placées entre le disque et le processeur, qui pourraient être gérées par des micro-processeurs, ou pour le stockage rapide de parties importantes de bases de données (structures, dictionnaire, ..), ou même en remplacement pur et simple du disque.

Par ailleurs, il est certain que les progrès de cette technologie stimuleront les progrès des disques, en volume, en rapidité, en prix et en "intelligence" et un élément important dans le développement des nouvelles technologies sera l'attitude des utilisateurs qui dépend pour une large part de facteurs subjectifs.

ANNEXES

ANNEXE I

FICHIER STRUCTURE

SYNTAXE DU LANGAGE DE DEFINITION DE STRUCTURE

(1) < structure > ::= DEBUT ;
< liste de pointeurs >
< liste de caractéristiques >
FIN ;

(2) < liste de pointeurs > ::= | < déclaration d'anneau > < liste de pointeurs >
| < déclaration de référence > < liste de pointeurs >

(3) < liste de caractéristiques > ::= < caractéristique >
< liste de caractéristiques > |

(4) < déclaration d'anneau > ::= ANNEAU < idind > ;

(5) < déclaration de référence > ::= REF < idind > SUR < id > < tableau > ;

(6) < tableau > ::= | TABLEAU < dim >

(7) < caractéristiques > ::= < déclaration d'entité > | < déclaration de CS >
| < déclaration de bloc > | < déclaration d'index >

(8) < déclaration d'entité > ::= ENTITE < nr > < id > ;
DEBUT ; < liste de pointeurs >
< liste de caractéristiques clé >
< liste de caractéristiques >
FIN ;

(9) < déclaration de CS > ::= CS < idind > < dim > < tableau > ;

(10) < déclaration de bloc > ::= BLOC < idind > < tableau > ;
DEBUT ; < liste de CS > FIN ;

(11) < liste de CS > ::= < CS > < liste de CS > |

(12) < déclaration d'index > ::= INDEX < idind > < nr > SUR < id > ;

(13) < liste de caractéristiques clé > ::= | < déclaration de clé >
< liste de caractéristiques clé >

(14) < déclaration de clé > ::= CLE < idind > < dim > ;

(15) < id > ::= < idind > | < idef >

<idind> est un identificateur non déjà défini
<idef> est un identificateur déjà défini
<dim> est un entier compris entre 1 et 256
<nr> est un entier compris entre 1 et $2^{16}-1$ (65 535)
<ent> est un entier compris entre 1 et 8.

Mots clés et symboles spéciaux :

DEBUT, FIN, ANNEAU, REF, TABLEAU, ENTITE, CS, BLOC, INDEX, CLE, ; (point virgule)

REPRESENTATION DE LA STRUCTURE

La structure définit une arborescence sous forme parenthésée. Une telle arborescence peut se représenter avec des enregistrements chaînés entre eux à l'aide de pointeurs fils - frère - père.

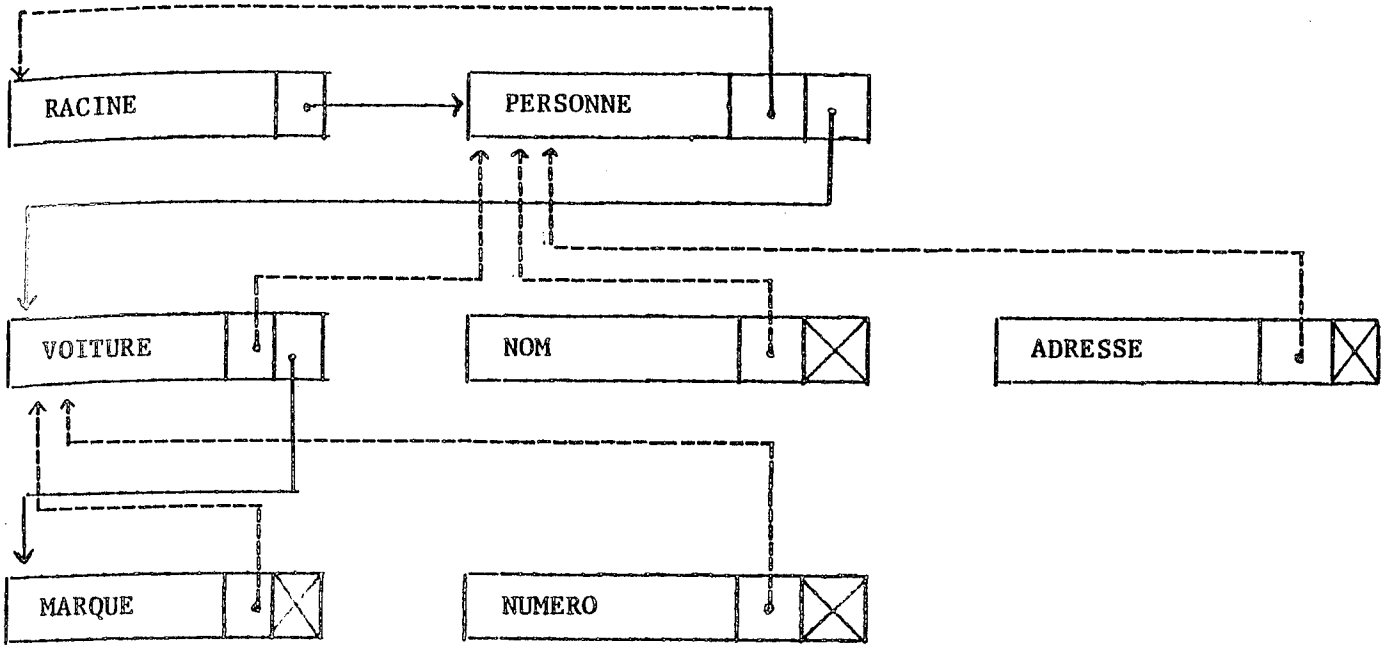
Nous avons préféré adopter un autre mode de représentation. Dans le fichier structure, une entité ou un bloc contient l'adresse de son premier fils ; tous les fils suivant le premier en séquence. Chaque enregistrement contient l'adresse de son père ; de sorte que la liste des fils d'un même enregistrement peut être parcourue en vérifiant que le chaînage père reste le même. Ce mode de représentation permet d'économiser un chaînage et d'accéder directement au $N^{ème}$ fils d'une entité ou d'un bloc, connaissant l'adresse du premier.

Exemple :

```
entité  10000  PERSONNE ;
début ; entité  5  VOITURES ;
        début ; CS  MARQUE  20 ;
                CS  NUMERO  20 ;
        fin ;
        CS  NOM  20 ;
        CS  ADRESSE  40   TABLEAU 2 ;
fin ;
```

ORGANISATION DU FICHIER STRUCTURE GENERE :

→ chaînage fils
--→ chaînage père.


RACINE
PERSONNE
VOITURE
NOM
ADRESSE
MARQUE
NUMERO

FORMAT DES ENREGISTREMENTS

1. RACINE

identificateur	longueur	déplacement	commentaire
TYPE	1	0	type d'enregistrement (1)
CHPERE	2	1	chaînage père (0)
CHFILS	2	3	chaînage fils (2)
LONG	1	5	longueur
LGP	1	6	longueur zone des pointeurs
vide	2	7	
NIPR	4	9	nom interne (1)
vide			

2. ENTITE

identificateur	longueur	déplacement	commentaire
TYPE	1	0	= 2
CHPERE	2	1	chaînage père
CHFILS	2	3	chaînage premier fils
LONG	1	5	longueur réalisation
LGP	1	6	longueur zone de pointeurs
NR	2	7	nombre de réalisations
NIPR	4	9	nom interne première réalisation
NIMAX	4	13	nom interne dernière réalisation

3. CARACTERISTIQUE SIMPLE

identificateur	longueur	déplacement	commentaire
TYPE	1	0	= 3
CHPERE	2	1	chaînage père
vide	2	3	
LONG	1	5	longueur
DIM	1	6	dimension
DEPL	2	7	déplacement
vide	8	9	

4.BLOC

identificateur	longueur	déplacement	commentaire
TYPE	1	0	= 4
CHPERE	2	1	chaînage père
CHFILS	2	3	chaînage premier fils
LONG	1	5	longueur
DIM	1	6	dimension
DEPL	2	7	déplacement
vide	8	9	

5. REFERENCES

identificateur	longueur	déplacement	
TYPE	1	0	= 5
CHPERE	2	1	chaînage père
CHAN	2	3	chaînage anneau
DEPL	1	5	déplacement
DIM	1	6	dimension
DEPLAN	2	7	déplacement anneau
vide	2	9	
CHENT	2	1	chaînage entité référencée (père de l'anneau).

6. ANNEAU

identificateur	longueur	déplacement	commentaire
TYPE	1	0	type = 6
CHPERE	2	1	chaînage père
CHREF	2	3	chaînage référence
DEPL	1	5	déplacement
vide	1	6	
DEPREF	2	7	déplacement référence
vide	2	9	
CHENT	2	11	chaînage vers entité contenant la référence
vide	4	13	

7. INDEX

identificateur	longueur	déplacement	commentaire
TYPE	1	0	= 7
CHPERE	2	1	chaînage père
CHCLE	2	3	chaînage clé
CHENT	2	5	chaînage vers père de la clé
NBENT	2	7	nombre entrées de la table
NIPR	4	9	nom interne première sous table
NIMAX	4	13	nom interne dernière sous table

8. CLE

identificateur	longueur	déplacement	commentaire
TYPE	1	0	= 8
CHPERE	2	1	chaînage père
CHIND	2	3	chaînage index
LONG	1	5	longueur
DEPSYN	1	6	déplacement chaînage synonymes
DEPL	2	7	déplacement caractéristique
vide	8	9	

ANNEXE II

EVALUATION DES ACCES AUX FICHIERS

A2-1 EVALUATION DES NOMBRES D'ACCES AUX FICHIERS

Les seules fonctions susceptibles d'entraîner des accès disques sont : APPEL, FRERE, IDEM, INIT, SUIVANT, NUMDE.
Les autres fonctions n'interviendront donc pas dans ces estimations.

a) Structure

Le nombre d'accès à la structure dépend essentiellement du type d'élément adressé et de la requête.

- Prenons par exemple la requête APPEL :
  - sur une caractéristique simple : en général, le nom interne et la longueur de la réalisation sont déjà calculés. On n'a besoin de connaître que la longueur et le déplacement de la caractéristique qui sont obtenues à partir de l'enregistrement de structure correspondant : 1 accès.
  - sur une entité : le calcul du nom interne et de la longueur nécessite l'accès à l'enrégistrement de structure de l'entité: 1 accès.
  - sur une référence : pour une lecture comme pour une écriture, on doit accéder aux enregistrements de structure de la référence, de l'anneau et de l'entité référencée : 3 accès.

  etc...

Nombre d'accès disque pour chaque requête :

	CS. blocs	entités	références	index
APPEL	1	1	3	3
FRERE	1	1	3	3
IDEM	1	1	—	—
INIT	—	1	—	3
SUIVANT	—	0	—	2
NUMDE	—	1	—	—

Les statistiques d'utilisation des requêtes au chapitre VIII permettent d'en déduire le nombre d'accès pour 100 requêtes :

APPEL : 65,6
FRERE : 9,6
IDEM : 0,9
INIT : 7,2
SUIVANT : 0,6
NUMDE : 4,9

TOTAL : 88,8 accès structure.

b) Dictionnaire

Le dictionnaire est accédé pour tout accès aux données, pour obtenir l'adress disque de la réalisation. Le nombre d'accès dépend à la fois du type de la donnée et du mode.

Exemples (pour la requête APPEL) :

- sur une entité, 1 accès est nécessaire pour une lecture ou une écriture, 2 accès pour une création (1 lecture, 1 écriture).
- sur une référence, en lecture comme en écriture, on doit lire la réalisatio référençante et la réalisation référencée : 2 lectures.

En règle générale, on considère qu'un accès dictionnaire entraîne un seul accès disque (cf. chapitre IV).

Nombre d'accès disques pour chaque requête :

	CS	entités			références			index	
	Blocs	lire écrire	créer	rien	lire	rien	écrire	écrire lire	rien
APPEL	1	1	2	0	2	1	2	2	
FRERE	1	1	2	0	2	1	2	2	
INIT				1				2	
SUIVANT				1				2	
IDEM	1	1							

Ce qui donne, pour 100 requêtes :

APPEL :	46,4	accès dictionnaires
FRERE :	6,4	" "
IDEM :	0,9	" "
INIT :	6,6	" "
SUIVANT :	13,8	" "
TOTAL :	74,1	accès dictionnaires.

c) Données :

L'accès aux données dépend également du type et du mode.

Exemples (pour APPEL) :

- pour une caractéristique simple ou une entité, l'accès en lecture n'entraîne qu'un accès disque, alors que l'écriture nécessite deux accès : lecture du bloc, réécriture, après modification.

- l'accès en lecture à une référence, nécessite deux accès : lecture de la réalisation contenant la référence, puis lecture de la réalisation référencée.

- l'accès en écriture à une référence, nécessite 4 accès : lecture de la réalisatic contenant la référence, lecture de la réalisation référencée, écriture de la référence, écriture de la réalisation référencée (pour mise à jour de l'anneau).

Nombre d'accès disques par requête :

	CS-Blocs		entités				références			index			
	lire	écrire	lire	écr	créer	rien	lire	écr	rien	lire	écr	créer	rie
APPEL	1	2	1	2	2	0	2	4	1	2	3	4	2
FRERE	1	2	1	2	2	0	2	4	1	2	3	4	2
IDEM	1	2	1	2	-	-	-	-	-	-	-	-	-
INIT	-	-	-	-	-	0	-	-	-	-	2	-	2
SUIVANT	-	-	-	-	-	0	-	-	-	-	-	-	2

Soit, pour 100 requêtes :

APPEL	:	55,4 accès données.
FRERE	:	7,8 " "
IDEM	:	1,2 " "
INIT	:	0,6 " "
SUIVANT	:	0,6 " "
TOTAL	:	65,6 accès aux données.

ANNEXE III

PROGRAMMES 6800

DUREE D'EXECUTION DE QUELQUES OPERATIONS SUR MC 6800

A3-1 - TRANSFERTS DE MEMOIRE A MEMOIRE

Le problème est le même que pour les comparaisons (cf. Chapitre IX-1). L'utilisation du pointeur de pile permet d'effectuer l'opération en 20 cycles par octets.

Sinon, l'opération prend 43 cycles par octet transféré, mais on peut également transférer les octets deux par deux en 32 cycles par octet :

Programme de transfert par deux octets :

```
*************************************************
*  TRANSFERT PAR 2 OCTETS                       *
*  ADRESSE ORIGINE DANS AD1                     *
*  ADRESSE DESTINATION DANS AD2                 *
*  LONGUEUR DANS D                              *
*************************************************

0000  DB 01     N TRZOCT ADDB   AD1+1     ADRESSE DE FIN )INDEX
0002  D7 05     N        STAB   INDEX+1
0004  D6 00     N        LDAB   AD1
0006  C3 00     A        ADDD   #0
0008  D7 04     N        STAB   INDEX
000A  DE 00     N BOUCLE LDX    AD1       ADRESSE ORIGINE
000C  A6 00     A        LDAA   0,X       CHARGEMENT DE 2 OCTETS
000E  E6 01     A        LDAB   1,X
0010  08                INX              PASSAGE AUX 2 OCTETS SUIVANTS
0011  08                INX
0012  DF 00     N        STX    AD1       SAUVEGARDER ADRESSE ORIGINE
0014  DE 02     N        LDX    AD2       ADRESSE DESTINATION
0016  A7 00     A        STAA   0,X       RANGEMENT DES 2 OCTETS
0018  E7 01     A        STAB   1,X
001A  08                INX              PASSAGE AUX DEUX OCTETS SUIVANTS
001B  08                INX
001C  DF 02     N        STX    AD2       SAUVEGARDE ADRESSE DESTINATION
001E  9C 04     N        CPX    INDEX     TEST SI FIN DE TRANSFERT
0020  26 E8 000A         BNE    BOUCLE    SINON ITERATION
0023  39                RTS
```

durée = 64 cycles par octet + 22 cycles.

A3-2 - OPERATIONS ARITHMETIQUES

a) Addition - soustraction :

Le MC 6800 dispose d'instructions d'addition sur un octet, mais l'existence d'un indicateur de report et d'une opération d'addition prenant en compte cet indicateur, permet d'additionner très facilement des nombres de plusieurs octets.

Exemple : addition de 4 octets :

```
                                  ****************************************
                                  *  ADDITION DE 4 OCTETS                *
                                  *  1ER OPERANDE = OP1                  *
                                  *  2EME OPERANDE = OP2                 *
                                  *  RESULTAT DANS OP1                   *
                                  ****************************************
                                        PSCT
                                  ADD4  OPER   OP1+3,ADD,OP2+3
                                        LDAA   OP1+3
                                        ADDA   OP2+3
                                        STAA   OP1+3
                                        OPER   OP1+2,ADC,OP2+2
                                        LDAA   OP1+2
                                        ADCA   OP2+2
                                        STAA   OP1+2
                                  A2    OPER   OP1+1,ADC,OP2+1
                                        LDAA   OP1+1
                                        ADCA   OP2+1
                                        STAA   OP1+1
                                        OPER   OP1,ADC,OP2
                                        LDAA   OP1
                                        ADCA   OP2
                                        STAA   OP1
                                        RTS
                                  ADD2  CLC
                                        BRA    A2
```

Cette opération prend de 45 à 69 cycles selon l'adressage utilisé.

b) Multiplication

De nombreux traitements entraînent des multiplications entre opérandes dont la longueur peut aller jusqu'à 4 octets : c'est le cas des calculs sur les noms interm En fait, le résultat ne dépassera jamais 4 octets, aussi on est sûr qu'un au moins des opérandes ne fait pas plus de deux octets, et un classement des opérandes nous ramène à une multiplication de 4 octets par 2 octets. Par la même occasion, les opé peuvent être placés en mémoire basse , ce qui permet d'utiliser l'adressage direct d'où gain de temps et d'encombrement du programme de calcul.

```
                    ****************************************
                    *     MULTIPLICATION                   *
                    * 1ER OPERANDE = OP1+2                 *
                    * 2EME OPERANDE = INDEXE               *
                    * RESULTAT = OP1 (SUR 6 OCTETS)        *
                    ****************************************


                    TRADIR COMM    BSCT      ZONE DE TRAVAIL EN MEMOIRE BASSE
0000                
0000      0002   A TR      RMB     2
0002      0002   A LONG    RMB     2
0004      0002   A INDEX   RMB     2


0036                      PSCT
                    *  CLASSEMENT DES OPERANDES ET CALCUL DES LONGUEURS
                    *  L'OPERANDE LE PLUS PETIT EST PLACE DANS IR
                    * L'OPERANDE LE PLUS GRAND EST PLACE DANS OP2
                    * LA LONGUEUR EN BITS EST PLACEE DANS LONG+1
0030 C6 08      A MULTIP LDAB    #8
003A                     COMP    (0,X),NE,OP1+2,DIF1
003A A6 00      A        LDAA    0,X
003C 91 02      N        CMPA    OP1+2
003E 26 10 0050          BNE     DIF1
0040 97 06      N        STAA    OP2
0042                     COMP    (1,X),NE,OP1+3,DIF2
0042 A6 01      A        LDAA    1,X
0044 91 03      N        CMPA    OP1+3
0046 26 0C 0054          BNE     DIF2
0048 97 07      N        STAA    OP2+1
004A                     COMP    (2,X),RA,OP1+4,DIF3
004A A6 02      A        LDAA    2,X
004C 91 04      N        CMPA    OP1+4
004E 20 08 0050          BRA     DIF3
                    *  OP1 < OP2
0050 97 06      N DIF1   STAA    OP2        2EME OPERANDE -> OP2
0052 A6 01      A        LDAA    1,X
0054 97 07      N DIF2   STAA    OP2+1
0056 A6 02      A        LDAA    2,X
0058 97 08      N DIF3   STAA    OP2+2
005A A6 03      A        LDAA    3,X
005C 97 09      N        STAA    OP2+3
005E 25 0A 006A          BCS     SUP
0060 DE 02      N        LDX     OP1+2
0062 26 5D 00C1          BNE     ERR        OPERANDE TROP LONG
0064 DE 04      N        LDX     OP1+4
0066 DF 00      N        STX     TR
0068 20 10 007A          BRA     SUIT
                    *  OP2 < OP1
006A DE 06      N SUP    LDX     OP2
006C 26 53 00C1          BNE     ERR        OPERANDE TROP LONG
006E DE 08      N        LDX     OP2+2
0070 DF 00      N        STX     TR
0072                     MVX4    OP1+2,OP2 OP1 -> OP2
0072 DE 02      N        LDX     OP1+2
0074 DF 06      N        STX     OP2
0076 DE 04      N        LDX     OP1+2+2
0078 DF 08      N        STX     OP2+2
007A 96 00      N SUIT   LDAA    TR
007C 27 01 007F          BEQ     LO1        LONGUEUR = 16 BITS
007E 58                  ASLD                OUI  B <- 16
007F D7 03      N LO1    STAB    LONG+1     LONGUEUR EN BITS
0081 7F 0002    N        CLR     LONG
```

```
                    *  PROGRAMME DE CALCUL
0084 D6 01     N          LDAB    TR+1       2EME OCTET DANS ACCU D
0086 CE 0000   A          LDX     #0
0089 DF 00     N          STX     OP1        RAZ OP1
008B DF 02     N          STX     OP1+2
008D DE 02     N          LDX     LONG       LONGUEUR DANS INDEX
008F C5 01     A MULT     BITB    #1         DERNIER BIT DE B = 1 ?
0091 27 03 0096           BEQ     DECL       NON : ON DECALE
0093 BD 0000   P          JSR     ADD4       OUI : ON ADDITIONNE
0096 74 0000   N DECL     LSR     TR         DECALAGE DU MULTIPLICATEUR
0099 56                   RORB
009A 74 0000   N          LSR     OP1        DECALAGE DU RESULTAT
009D 76 0001   N          ROR     OP1+1
00A0 76 0002   N          ROR     OP1+2
00A3 76 0003   N          ROR     OP1+3
00A6 76 0004   N          ROR     OP1+4
00A9 76 0005   N          ROR     OP1+5
00AC 09                   DEX                DECREMENTATION LONGUEUR
00AD 26 E0 008F           BNE     MULT       NON NUL : ON RECOMMENCE
00AF D6 03     N          LDAB    LONG+1     FIN DE LA MULTIPLICATION
00B1 C1 08     A          CMPB    #8         TEST SI LONGUEUR = 8BITS
00B3 22 08 00BD           BHI     FINM       NON (16 BITS)  C'EST FINI!
00B5 DE 03     N          LDX     OP1+3      OUI : RECADRAGE DU RESULTAT
00B7 DF 04     N          STX     OP1+4
00B9 DE 01     N          LDX     OP1+1
00BB DF 02     N          STX     OP1+2
00BD 4F           FINM    CLRA               PAS D'ERREUR
00BE 97 01     N          STAA    OP1+1
00C0 39                   RTS
00C1 86 01     A ERR      LDAA    #1         ERREUR
00C3 39                   RTS
```

Pour une multiplication de 4 octets par 2 octets, le calcul dure entre 963 et 1807 cycles, soit une moyenne de 1385 cycles en supposant tous les nombres équiprobable
Une multiplication de 4 octets par 1 octet prend entre 516 et 940 cycles, soit une moyenne de 728 cycles.
Le classement des opérandes prend en moyenne 95 cycles.
La durée totale d'une multiplication est donc :

4 X 2 : 1385 + 95 = 1480 cycles que l'on arrondira à 1500 cycles.
4 X 1 : 728 + 95 = 823 cycles que l'on arrondira à 900 cycles.

L'arrondissement à la centaine supérieure permet de garder une petite marge d'erreur, pour tenir compte, par exemple, du temps pris par le passage des argumen variable selon les cas (les opérations pouvant être enchaînées).

c) Division

Le traitement des noms internes peut entraîner des divisions de 4 octets par 4 octets. Un programme effectuant cette opération par soustractions et décalages, prendra entre 3800 et 5700 cycles. Cela est très long, et on peut faire les remarques suivantes :

- si la longueur du diviseur est inférieure ou égale à 2 octets, on pourra conserver les deux octets de poids fort du dividende dans des registres et effectuer directement les additions et soustractions, mais on pourra avoir jusqu'à 33 itérations.

- si la longueur du diviseur est supérieure à 2, les additions et soustractions se feront sur 4 octets à l'aide de sous-programmes. Par contre, l'opération nécessitera 17 itérations au maximum.

Il semble donc préférable d'utiliser deux sous-programmes différents pour ces deux cas, auxquels il faudra ajouter un sous-programme de cadrage dès opérandes, de calcul des longueurs et de sélection du sous-programme de calcul, et un sous-programme de recadrage du résultat.

```
                        ********************************************
                        *  DIVISION PAR 4 OCTETS                   *
                        *  DIVIDENDE : OP1+1 (4 OCTETS) CADRE A GAUCHE  *
                        *  DIVISEUR  : OP2 CADRE A GAUCHE          *
                        *  RESTE : OP1                             *
                        *  QUOTIENT : OP1+LONGUEUR DU DIVISEUR     *
                        ********************************************
                        *
00C4 DE 02     N DIV44  LDX     LONG      NOMBRE D'ITERATIONS
00C6 DD 001C   P DI441  JSR     SUB4      SOUSTRACTION
00C9 25 16 00E1         BCS     DADD4
00CB 0D                 SEC
00CC 79 0004   N DI442  ROL     OP1+4     DECALAGE DU RESULTAT
00CF 79 0003   N        ROL     OP1+3
00D2 79 0002   N        ROL     OP1+2
00D5 79 0001   N        ROL     OP1+1
00D8 79 0000   N        ROL     OP1
00DB 09                 DEX
00DC 26 E0 00C6         BNE     DI441
00DF 7E 019E   P        JMP     CADRES
00E1 DD 0000   P DADD4  JSR     ADD4
00E4 0C                 CLC
00E5 20 E5 00CC         BRA     DI442
```

```
                    ********************************************************
                    *        DIVISION PAR 2 OCTETS                         *
                    * DIVIDENDE :OP1+1   CADRE A GAUCHE                    *
                    * DIVISEUR :OP1  CADRE A GAUCHE                        *
                    * QUOTIENT :OP1+LONGUEUR DIVISEUR                      *
                    * RESTE : OP1                                          *
                    ********************************************************
                    *
00E7 DE 02     N DIV42  LDX    LONG      NOMBRE D'ITERATIONS
00E9 96 00     N        LDAA   OP1       POIDS FORTS DANS ACCUS
00EB D6 01     N        LDAB   OP1+1
00ED D0 07     N DI421  SUBB   OP2+1     SOUSTRACTION
00EF 92 06     N        SBCA   OP2
00F1 2B 16 0109         BMI    DADD2
00F3 0D                 SEC               INTRODUIRE UN 1
00F4 79 0004   N DI422   ROL    OP1+4     DECALAGES
00F7 79 0003   N        ROL    OP1+3
00FA 79 0002   N        ROL    OP1+2
00FD 59                 ROLB
00FE 49                 ROLA
00FF 09                 DEX               DECREMENTATION NOMBRE D'ITERATIO
0100 26 EB 00ED         BNE    DI421
0102 97 00     N        STAA   OP1       FIN DE LA DIVISION
0104 D7 01     N        STAB   OP1+1
0106 7E 019E   P        JMP    CADRES
0109 DB 07     N DADD2  ADDB   OP2+1     ADDITION
010B 99 06     N        ADCA   OP2
010D 0C                 CLC               INTRODUIRE UN ZERO
010E 20 E4 00F4         BRA    DI422

                    ********************************************
                    *  PROGRAMME DE CLASSEMENT DE LA DIVISION  *
                    *  DIVIDENDE :OP1+1                        *
                    *  DIVISEUR : INDEXE                       *
                    ********************************************
                    *
0110 C6 04     A DIVISE LDAB   #4        CALCUL DE LA LONGUEUR DE OP2
0112 4F                 CLRA
0113 97 07     N        STAA   OP2+1     RAZ OP2
0115 97 08     N        STAA   OP2+2
0117 97 09     N        STAA   OP2+3
0119 97 00     N        STAA   OP1
011B 97 02     N        STAA   LONG
011D                    TSTL   (0,X),LONG42 TESTS DE LONGUEUR
011D A6 00     A        LDAA   0,X
011F 26 14 0135         BNE    LONG42
0121 5A                 DECB
0122 08                 INX
0123                    TSTL   (0,X),LONG32
0123 A6 00     A        LDAA   0,X
0125 26 12 0139         BNE    LONG32
0127 5A                 DECB
0128 08                 INX
0129                    TSTL   (0,X),LONG22
0129 A6 00     A        LDAA   0,X
012B 26 10 013D         BNE    LONG22
012D 5A                 DECB
012E 08                 INX
012F A6 00     A        LDAA   0,X
0131 27 8E 00C1         BEQ    ERR       DIVISION PAR ZERO
0133 20 0C 0141         BRA    LONG12
```

```
135 A6 03     A LONG42 LDAA   3,X
137 97 09     N        STAA   OP2+3
139 A6 02     A LONG32 LDAA   2,X
13B 97 00     N        STAA   OP2+2
13D A6 01     A LONG22 LDAA   1,X
13F 97 07     N        STAA   OP2+1
141 A6 00     A LONG12 LDAA   0,X
143 97 06     N        STAA   OP2
145 D7 00     N CLOP1  STAB   TR        LONGUEUR DU 1ER OPERANDE
147 C6 04     A        LDAD   #4        CALCUL DE LA LONGUEUR DU 2EME OPER
149                    TSTL   OP1+1,CNIT
149 96 01     N        LDAA   OP1+1
14B 26 28 0175         BNE    CNIT
14D 5A                 DECB
14E                    TSTL   OP1+2,LONG31
14E 96 02     N        LDAA   OP1+2
150 26 0F 01C1         BNE    LONG31
152 5A                 DECB
153                    TSTL   OP1+3,LONG21
153 96 03     N        LDAA   OP1+3
155 26 13 016A         BNE    LONG21
157 5A                 DECB
158 96 04     N        LDAA   OP1+4     1 OCTET
15A 97 01     N        STAA   OP1+1
15C 4F                 CLRA
15D 97 02     N        STAA   OP1+2     COMPLETER AVEC DES ZEROS
15F 20 10 0171         BRA    CLR3
161 97 01     N LONG31 STAA   OP1+1     3 OCTETS
163 DE 03     N        LDX    OP1+3
165 DF 02     N        STX    OP1+2
167 4F                 CLRA
168 20 09 0173         BRA    CLR4
16A 97 01     N LONG21 STAA   OP1+1     2 OCTETS
16C 96 04     N        LDAA   OP1+4
16E 97 02     N        STAA   OP1+2
170 4F                 CLRA
171 97 03     N CLR3   STAA   OP1+3
173 97 04     N CLR4   STAA   OP1+4
175 D7 01     N CNIT   STAD   TR+1      CALCUL DU NOMBRE D'ITERATIONS
177 D0 00     N        SUBB   TR        (LONG(OP2)-LONG(OP2))*8+9
179 2B 70 01EB         BMI    MINUS
17D 58                 ASLB
17C 58                 ASLB
17D 58                 ASLB
17E CB 09     A        ADDB   #9
180 D7 03     N        STAB   LONG+1
182 86 02     A        LDAA   #2
184 91 00     N        CMPA   TR
186 2C 03 018D         BGE    *+5       DIVISION 4/2
188 7E 00C4   P        JMP    DIV44     DIVISION 4/4
18B 7E 00E7   P        JMP    DIV42
```

```
00240                    ************************************************
00241                    *   CADRAGE DES RESULTATS                      *
00242                    *  QUOTIENT DANS OP1 (4 OCTETS)                *
00243                    *  RESTE DANS OP2 (4 OCTETS)                   *
00244                    ************************************************


00246P 018E C6 00     A RAZOP2 LDAB   #0
00247P 0190 D7 06     N        STAB   OP2
00248P 0192 D7 07     N        STAB   OP2+1
00249P 0194 D7 08     N        STAB   OP2+2
00250P 0196 D7 04     N        STAB   INDEX    RAZ INDEX
00251P 0198 97 05     N        STAA   INDEX+1
00252P 019A DE 04     N        LDX    INDEX    LONGUEUR DU RESTE -> X
00253P 019C 09                 DEX
00254P 019D 39                 RTS


00256P 019E 96 00     N CADRES LDAA   TR       LONGUEUR DU RESTE
00257P 01A0 8D EC 018E         BSR    RAZOP2
00258P 01A2                    CARS   OP1,3,RORES3 CADRAGE DU RESTE
     P 01A2 A6 00     N        LDAA   OP1,X
     P 01A4 97 09     N        STAA   OP2+3
     P 01A6 09                 DEX
     P 01A7 8C 0000   A        CPX    #0
     P 01AA 2B 21 01CD         BMI    RORES3
00259P 01AC                    CARS   OP1,2,RORES2
     P 01AC A6 00     N        LDAA   OP1,X
     P 01AE 97 08     N        STAA   OP2+2
     P 01B0 09                 DEX
     P 01B1 8C 0000   A        CPX    #0
     P 01B4 2B 14 01CA         BMI    RORES2
00260P 01B6                    CARS   OP1,1,RORES1
     P 01B6 A6 00     N        LDAA   OP1,X
     P 01B8 97 07     N        STAA   OP2+1
     P 01BA 09                 DEX
     P 01BB 8C 0000   A        CPX    #0
     P 01BE 2B 07 01C7         BMI    RORES1
00261P 01C0 A6 00     N        LDAA   OP1,X    LONGUEUR = 4
00262P 01C2 97 06     N        STAA   OP2
00263P 01C4 76 0006   N        ROR    OP2      REAJUSTEMENT RESTE
00264P 01C7 76 0007   N RORES1 ROR    OP2+1
00265P 01CA 76 0008   N RORES2 ROR    OP2+2
00266P 01CD 76 0009   N RORES3 ROR    OP2+3
00267P 01D0 96 01     N        LDAA   TR+1     LONGUEUR DU QUOTIENT
00268P 01D2 90 00     N        SUBA   TR
00269P 01D4                    MVX4   OP1+1,OP1 RECADRAGE QUOTIENT
     P 01D4 DE 01     N        LDX    OP1+1
     P 01D6 DF 00     N        STX    OP1
     P 01D8 DE 03     N        LDX    OP1+1+2
     P 01DA DF 02     N        STX    OP1+2
00270P 01DC 80 03     A        SUBA   #3
00271P 01DE 27 09 01E9         BEQ    FIND
00272P 01E0 CE 0000   A        LDX    #0
00273P 01E3 E7 00     N COMQUO STAB   OP1,X    NETTOYER LES POIDS FORTS
00274P 01E5 08                 INX
00275P 01E6 4C                 INCA
00276P 01E7 26 FA 01E3         BNE    COMQUO
00277P 01E9 4F           FIND  CLRA
00278P 01EA 39                 RTS
```

```
01ED 96 01     N MINUS  LDAA   TR+1      LONGUEUR DIVIDENDE
01EF 8D 9F 018E         BSR    RAZOP2
01F1                    CARS   OP1+1,3,RAZQUO
01F1 A6 01     N        LDAA   OP1+1,X
01F3 97 09     N        STAA   OP2+3
01F5 09                 DEX
01F6 8C 0000   A        CPX    #0
01F9 2B 18 0213         BMI    RAZQUO
01FB                    CARS   OP1+1,2,RAZQUO
01FB A6 01     N        LDAA   OP1+1,X
01FD 97 00     N        STAA   OP2+2
01FF 09                 DEX
0200 8C 0000   A        CPX    #0
0203 2B 0E 0213         BMI    RAZQUO
0205                    CARS   OP1+1,1,RAZQUO
0205 A6 01     N        LDAA   OP1+1,X
0207 97 07     N        STAA   OP2+1
0209 09                 DEX
020A 8C 0000   A        CPX    #0
020D 2B 04 0213         BMI    RAZQUO
020F A6 01     N        LDAA   OP1+1,X
0211 97 06     N        STAA   OP2
0213 CE 0000   A RAZQUO LDX    #0        RAZ QUOTIENT
0216 DF 00     N        STX    OP1
0218 DF 02     N        STX    OP1+2
021A 20 CF 01E9         BRA    FIND
```

La durée moyenne de ces opérations (en cycles), en fonction de la longueur des opérandes, est donnée par le tableau suivant :

Dividende / Diviseur	1	2	3	4
1	600	1000	1400	1700
2		600	1000	1400
3			1100	2000
4				1100

Nous supposerons par approximation, les combinaisons équiprobables. La durée moyenne d'une division est alors de 1200 cycles.

A3-3 - TRAITEMENTS DE LISTES ET DE TABLES

a) Recherche séquentielle dans une table

```
00017                    ********************************************
00018                    * RECHERCHE SEQUENTIELLE DANS UNE TABLE
00019                    * CHAINE RECHERCHEE DANS OP1,LONGUEUR=4 OCTETS
00020                    * ADRESSE DE LA TABLE DANS DEBTAB
00021                    * FIN DE LA TABLE DANS FINTAB
00022                    * LONGUEUR D'UNE ENTREE DANS LGENT
00023                    * SI TROUVE RETOUR DE L'ADRESSE DANS X
00024                    * SINON X=0
00025                    ********************************************

0002?P 0000 FE 0000  N RSTADN LDX    DEBTAB   ADRESSE DE LA TABLE
00020P 0003 FF 0000  D        STX    TRADRE   SAUVEGARDE DE L'ADRESSE
00029P 0006 96 00    N LOOPTN LDAA   OP1      COMPARAISON
0002BP 0008 A1 00    A        CMPA   0,X
0002DP 000A 26 1G 0022        BNE    ITER
00032P 000C 96 01    N        LDAA   OP1+1
00033P 000E A1 01    A        CMPA   1,X
00034P 0010 26 10 0022        BNE    ITER
00035P 0012 96 02    N        LDAA   OP1+2
00036P 0014 A1 02    A        CMPA   2,X
00037P 0016 26 0A 0022        BNE    ITER
00038P 0018 96 03    N        LDAA   OP1+3
00039P 001A A1 03    A        CMPA   3,X
0003BP 001C 26 04 0022        BNE    ITER
0003D                    * EGALITE
0003EP 001E FE 0000  D        LDX    TRADRE   RESTAURATION ADRESSE
00041P 0021 39                RTS             FIN
00042P 0022 F6 0001  D ITER   LDAB   TRADRE+1 PASSAGE A L'ENTREE SUIVANTE
00045P 0025 FB 0004  N        ADDB   LGENT    TRADRE = TRADRE + LGENT
00046P 0028 F7 0001  D        STAB   TRADRE+1
00047P 002B F6 0000  D        LDAB   TRADRE
00049P 002E C9 00    A        ADCB   #0
0004BP 0030 F7 0000  D        STAB   TRADRE
0004EP 0033 FE 0000  D        LDX    TRADRE   CHARGEMENT DE L'ADRESSE
00051P 0036 BC 0002  N        CPX    FINTAB   TEST SI FIN DE LA TABLE
00052P 0039 26 CD 0008        BNE    LOOPTN   NON : ITERATION
00053P 003B CE 0000  A        LDX    #0       ECHEC: RAZ INDEX
00056P 003E 39                RTS
```

La comparaison s'effectue en 13 cycles par octets si la chaîne est en adressage étendu. En général, l'inégalité est détectée dès le 1er octet.

Pour N itérations, la durée sera : 45 X (N-1) + 13 X LCH+5 *

(lorsque l'égalité est détectée, tous les octets doivent être comparés).

Si la chaîne recherchée n'est pas de longueur fixe, il faudra utiliser un sous-progr de comparaison et la durée sera : 92 X (N-1) + 60 X LCH+5.

* LCH = longueur de la chaîne recherchée.

b) Recherche séquentielle sur liste chaînée :

Cette recherche est plus rapide que la précédente, car le passage au suivant se fait par changement de l'index avec la valeur du chaînage.

```
*****************************************************
* RECHERCHE SEQUENTIELLE SUR LISTE NON INTER
* IX1 : ADRESSE DE LA CHAINE A RECHERCHER
* B : LONGUEUR
* SI TROUVE X EST PILIST CONTIENNANT L'ADRESSE DE
* L'ENTREE
* SINON X=0 ET PILIST CONTIENT L'ADRESSE DE LA DERNIERE
* ENTREE
*****************************************************
```

```
                R RECHER LDX   #D1
                         STX   SVAD1     SAUVEGARDE ADRESSE CHAINE
                         STAB  SVB       SAUVEGARDER B
                         LDX   PILIST    TETE DE LISTE
                         BEQ   FIN
                R LOOPER STX   PILIST    SAUVEGARDE ADRESSE
                         INX             ADRESSE DE LA CHAINE
                         INX             ADRESSE DE L'ENTREE + 2
                         STX   #D2
                         LDX   SVAD1     RESTAURATION ADRESSE CHAINE
                         STX   #D1
                         LDAB  SVB       RESTAURER LONGUEUR
                         JSR   CMPXX     COMPARAISON
                         BEQ   FIN       CHAINE TROUVEE ?
                         LDX   PILIST    RESTAURATION ADRESSE
                         LDX   0,X       SUIVANT
                         BEQ   FIN
                         BRA   LOOPER
                F FIN    LDX   PILIST
                         RTS
```

Pour N itérations, cette opération prend : 70 X (N 1) + 60 X $_{LCH}$+5.

Si la chaîne à rechercher est de longueur fixée, la comparaison peut se faire sans utiliser de sous-programme.

L'opération dure alors : 23 X (N-1) + 13 X $_{LCH}$ +5.

c) Recherche par dichotomie

Chaîne cherchée : CH.

table : TAB, Nombre d'entrées : NENT.

indice du résultat : N.

algorithme (en PL/I) :

```
DICHOTØMIE : PRØC ;
     MAX = NENT ; MIN = 1 ;          13 cycles.
ITER :
     i = (MAX-MIN)/2 ;               12 cycles.
     N = i + MIN ;                    7 cycles.
```

```
if CH = TAB(N)  then  RETURN                indexation
if CH < TAB(N)  then MAX = N ;              + comparaison
else MIN = N ;                              11 cycles
if  i ¬= 0 then GOTO  ITER ;                 7 cycles
FIN : if CH = TAB(MAX)  then N = MAX ;      indexation + comparaison
else NON TROUVE ;                           + 3 cycles
end ;                                         5 cycles.
```

Désignons par TIND la durée de l'indexation, TCØMP la durée de la comparaison et NIT le nombre d'itérations.

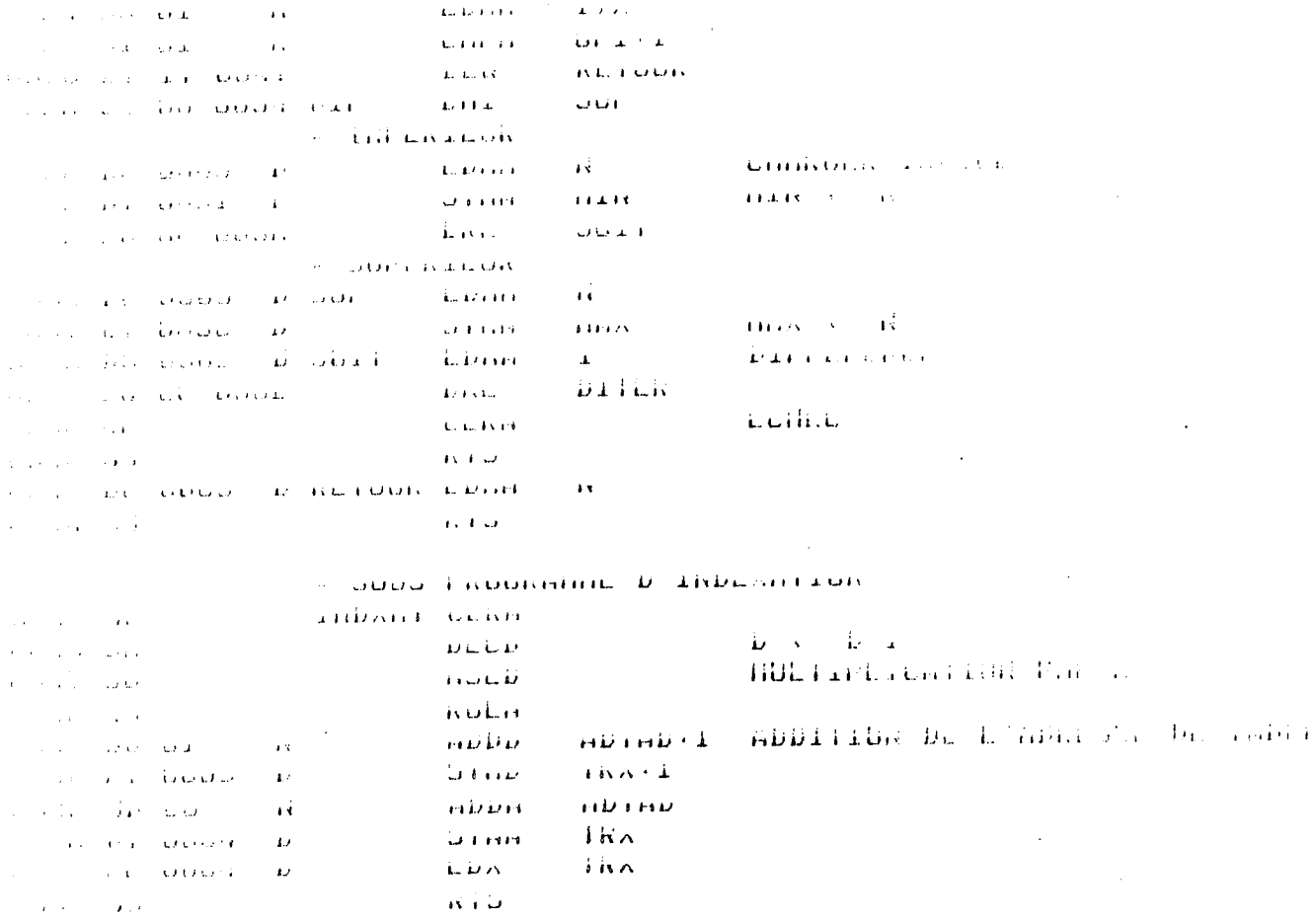
La durée de l'opération est égale à : 21 + TIND + TCØMP + NIT (37 + TIND + TCØMP).

Pour réduire la durée de l'indexation, il peut être intéressant de séparer la table en deux parties, l'une contenant la clé d'accès, l'autre contenant les autre informations ; on a ainsi des chances de se ramener à une indexation plus simple (multiplication par une puissance de 2 par exemple).

Exemple de programmation :

```
********************************************************
* RECHERCHE PAR DICHOTOMIE
* ADRESSE DE LA TABLE DANS ADTAB , NOMBRE D'ENTREES
* DANS NENTAB , LES ENTREES DE TABLE FONT DEUX OCTETS
* VALEUR CHERCHEE = DEUX OCTETS DANS OP1
* RETOUR : DANS A VALEUR DE L'INDICE DE L'ENTREE
* OU 0 SI ECHEC
********************************************************

0000 96 02     N DICHOT LDAA   NENTAB   NOMBRE D'ENTREES
0002 4C                 INCA
0003 D7 0000   D        STAA   MAX
0005 86 01     A        LDAA   #1
0007 D7 0001   D        STAA   MIN       MIN < 1
0009 F6 0000   D DITLR  LDAB   MAX
000C F0 0001   D        SUBB   MIN       B < (MAX-MIN)/2
000F 57                 ASRB
0010 D7 0002   D        STAB   I         SAVE DIFFERENCE
0012 FB 0001   D        ADDB   MIN       B < I+MIN
0015 D7 0003   D        STAB   N         SAVE INDICE
0017 DB 0045   P        JSR    INDXAT    INDEXATION
0019 A6 00     A        LDAA   0,X       COMPARAISON
001B 91 60     N        CMPA   OP1
001D 26 06 0025         BNE    DIT
```



d) Insertion en tête de liste :

La tête de liste est contenue dans TTLIST.

```
                ********************************************
                *  INSERTION EN TETE DE LISTE              *
                *  ADRESSE DE L'ELEMENT A INSERER DANS X   *
                ********************************************

0000 D6 0000  N INSTLC LDAA   TTLIST
0002 A7 00    A        STAA   0,X
0004 D6 0001  N        LDAA   TTLIST+1
0006 A7 01    A        STAA   1,X
0008 FF 0000  N        STX    TTLIST
000B 39                RTS
```

Cette opération prend 27 cycles.

e) Insertion en queue de liste :

La queue de liste est contenue dans QULIST.

L'opération prend 50 cycles.

```
                    ************************************************
                    *  INSERTION EN QUEUE DE LISTE                 *
                    *  ADRESSE DE L'ELEMENT A INSERER DANS X       *
                    ************************************************

000E 6F 00     A INSQLC CLR    0,X
0010 6F 01     A        CLR    1,X       RAZ CHAINAGE
0012 DF 04     N        STX    INDEX     SAUVEGARDE ADRESSE
0014 FE 0002   N        LDX    QULIST    QUEUE DE LISTE
0017 96 04     N        LDAA   INDEX
0019 A7 00     A        STAA   0,X       CHAINAGE AVEC DERNIER ELEMENT
001B 96 05     N        LDAA   INDEX+1
001D DE 04     N        LDX    INDEX
001F A7 01     A        STAA   1,X
0021 FF 0002   N        STX    QULIST
```

f) Extraction d'une liste chaînée :

```
                    ************************************************
                    *              EXTRACTION                      *
                    *   ADRESSE DE L'ELEMENT PRECEDENT DANS X      *
                    ************************************************

0040 DF 04     N EXTRLC STX    INDEX
0042 EE 00     A        LDX    0,X
0044 A6 00     A        LDAA   0,X
0046 E6 01     A        LDAB   1,X
0048 DE 04     N        LDX    INDEX
004A A7 00     A        STAA   0,X
004C E7 01     A        STAB   1,X
004E 39                 RTS
```

A3-4 - ACCES AUX FICHIERS

A3-4.1 - Structure

Accès logique : le processeur d'accès à la structure reçoit un numéro d'enregistrement de structure. Il doit obtenir l'enregistrement et le placer dans le conte concerné par la requête.
L'enregistrement peut être déjà en mémoire :

- dans la zone où sont mémorisées les entités et références (cf. Chapitre VIII-
- dans le contexte d'un autre utilisateur.
- dans une mémoire tampon.

La recherche de l'enregistrement dans les contextes coûterait cher et aurait vraisemblablement une faible probabilité de succès. On ne l'effectuera donc pas, à pri

La recherche dans la zone des références et entités peut se faire par dichotomie si la table a 64 entrées, il faut six itérations pour obtenir l'entrée recherchée, ce qui, d'après le paragraphe précédent, doit durer 654 cycles.

Cette recherche a une probabilité de réussite de l'ordre de 80 % ; en cas d'échec il faudra calculer l'adresse du bloc disque contenant l'enregistrement, vérifier qu'il ne se trouve pas déjà en mémoire et sinon demander un accès physique.

Le calcul d'adresse du bloc se fait par division du numéro d'enregistrement par le nombre d'enregistrements dans un bloc : c'est une division de 20 octets par un octet dont le résultat sera toujours inférieur à 16 (nombre maximum de blocs de structure). On a donc intérêt à effectuer l'opération par soustractions successives. Si on suppose le résultat moyen égal à 5, l'opération prendra 84 cycles. Il faut ensuite comparer l'adresse obtenue avec l'adresse du bloc mémorisé dans la mémoire tampon du fichier structure. Si un seul bloc est mémorisé, cela fait une comparaison d'un octet, qui dure 10 cycles. La durée du calcul d'adresse et de la comparaison, est de l'ordre de 100 cycles;avec une probabilité de 1/5 que ces opérations aient lieu, on a une durée moyennne de 20 cycles.

L'enregistrement est enfin transféré dans le contexte demandeur.

Les diverses opérations d'appel de sous-programmes, passage de paramètres, etc, peuvent être évaluées approximativement à 70 cycles.

Total : divers : 744 cycles.
transfert : 18 X 32 = 576 cycles.
Total (arrondi) : 1350 cycles.

Accès physique : si l'enregistrement n'a pas été trouvé en mémoire, il faut accéder au disque. La conversion de l'adresse sous la forme numéro de cylindre-numéro de piste-numéro de secteur, est immédiate puisque la structure ne comporte qu'une piste : ceci entraîne tout au plus la lecture de trois octets dans une table, ce qui dure, indexation comprise, environ 60 cycles. La mise en attente de la requête est un parcours de liste chaînée, suivi d'une insertion : le parcours de chaîne peut prendre 90 cycles pour parcourir 3 entrées (pour une longueur totale de 6 entrées) et l'insertion se fait en 50 cycles. L'accès disque va entraîner une interruption, l'envoi de quelques octets vers le contrôleur de disque, l'extraction de la requête en tête de file d'attente, la recherche du contexte concerné par l'opération achevée : au total, 280 cycles environ.

Total : opérations diverses : (arrondi) : 500 cycles.

A3-4.2 - Dictionnaire

1- Calculer un numéro d'entrée dictionnaire à partir du nom interne : c'est le reste de la division du nom interne par le nombre total d'entrées du dictionnai C'est une division de 4 octets par 3 octets qui prend au maximum 2000 cycles.

2- Calculer le numéro du bloc du fichier dictionnaire contenant l'entrée voul on divise le numéro de l'entrée par le nombre d'entrées dans un bloc ; le reste fc le rang de l'entrée dans le bloc. C'est une division de 3 octets par un octet qui donc 1400 cycles.

3- Effectuer l'accès disque : la conversion de l'adresse disque est nécessair cette fois. Elle va entraîner une division du numéro de bloc par le nombre de sect par piste (2 octets par 1 octet) et une division du numéro de piste par le nombre de pistes par cylindre (1 octet par 1 octet). Ces deux opérations peuvent durer 1600 cycles. Le reste du traitement est le même que pour la structure et prendra d 500 cycles.

4- Rechercher l'entrée dans le bloc : c'est une recherche séquentielle sur li chaînée. Pour chaque itération, il y a une indexation, effectuée par multiplicatio par dix du numéro de l'entrée : cette multiplication peut s'effectuer en trois déc et une addition, et l'indexation prend 50 cycles. La comparaison du nom interne de l'entrée avec le nom interne cherché, ne demandera en général que la comparaison d octet lorsqu'elle est négative : ceci demande 12 cycles. Pour passer au suivant, i lire les trois octets de chaînage, leur soustraire le rang de la première entrée d bloc, et vérifier que l'entrée est bien dans le même bloc ; sinon, il faudra un autre bloc, mais ceci est extrêmement rare. Cette opération peut prendre enviro 42 cycles. On a donc au total environ 110 cycles par itération, sauf pour la derni où les 4 octets du nom interne devront être testés, puisqu'il y a égalité : ceci f environ 40 cycles de plus.

Les mesures effectives par la SAGEM ont montré que la longueur moyenne des chaînes était inférieure à 8, ce qui fait en moyenne moins de 4 comparaisons. La recherche de l'entrée prendra donc environ : 3 X 110 + 150 = 330 + 150 = 480 cycle

5- Ecrire l'adresse disque dans le contexte: c'est un transfert de 3 octets q demande donc : 33 cycles.

Durée totale de l'accès : multiplication-divisions : 5000 cycles.
divers (arrondi) : 1100 cycles.
Total : 6100 cycles.

A3-4.3 - Données :

Accès en lecture :

1- L'accès disque s'effectue comme pour le dictionnaire ; il entraîne donc 1600 cycles de conversion d'adresse et 500 cycles d'opérations diverses.

2- Il faut ensuite rechercher la réalisation dans le bloc, en comparant les noms internes rencontrés avec le nom interne cherché. Pour chaque itération, il y a : une comparaison (en général l'inégalité est détectée dès le premier octet : 12 cycles), une addition, pour passer à la réalisation suivante (2 octets : 20 cycles) une comparaison de l'adresse, pour vérifier qu'on n'aie pas atteint la dernière réalisation (2 octets, 26 cycles). Il faut donc compter 58 cycles par itération. Si on compte en moyenne 20 réalisations par bloc, il faudra en moyenne 10 itérations. La dernière itération demandera une comparaison de 4 octets, mais pas de passage au suivant.
La durée de cette recherche est donc : 9 X 58 + 48 = 570 cycles.

3- Transfert de la réalisation dans le contexte : c'est un échange de 256 octets.

Durée totale :	multiplication-division :	1600 cycles.
	opérations diverses :	1160 cycles.
	transferts : 256 X 32	= 8192 cycles.
	Total (arrondi)	11000 cycles.

+ écriture : la réalisation a déjà été localisée lors de la lecture. On n'effectue donc que la modification de la mémoire tampon et l'accès disque.

Durée :	divers :	500 cycles
	opérations arithmétiques :	1600 cycles
	transferts : 256 X 32 :	8192 cycles
	Total (arrondi)	10300 cycles.

A3-5 - EXCLUSIONS MUTUELLES

La programmation des primitive s LECT, MAJ, NOLECT, NOMAJ, sera la suivante :

Avec : D = parallélisme employé (degré de multiprogrammation)
DMAX = parallélisme maximum

LECT (Nom interne) ;
Chercher l'entrée de table correspondante ;

```
      si échec, alors créer une entrée ;
      Si existent des contextes en attente alors
      mettre le demandeur en attente ;
      Sinon, si D<DMAX alors D = D+1 ;
      Sinon mettre le demandeur en attente ;
fin ;

MAJ (Nom interne) ;
      chercher l'entrée de table ;
      Si échec, alors créer une entrée ;
      Si D ≠ 0, alors mettre le demandeur en attente ;
      Sinon D = DMAX ;
fin ;

NOLECT (nom interne) ;
      chercher l'entrée de table ;
      D = D-1 ;
      Si D = 0, alors libérer le premier contexte en attente ;
fin ;

NOMAJ (nom interne) ;
      chercher l'entrée de table ;
      D = 0 ;
      libérer les contextes en attente ;
fin ;
```

Les opérations d'exclusion mutuelle sur le dictionnaire, fonctionnent sur le même principe, mais il n'est pas nécessaire de rechercher le descripteur qui est unique.

Les opérations entraînées par les exclusions mutuelles sont donc des recherches, insertions dans une table, des parcours de listes chaînées, des insertions et extracti sur des listes chaînées, et des tests d'indicateurs.

La table pourra être accédée par hash coding et contiendra autant d'entrées que l'on peut accéder simultanément de réalisations, soit une entrée par contexte, dor 64 entrées au maximum. Chaque entrée contiendra un nom intrene (4 octets), le parallélisme employé (1 octet), une tête de liste de contextes utilisateurs (1 octet), un chaînage de hash coding (1 octet), une tête de liste d'attente (1 octet). Les entrées occuperont donc 8 octets. Pour une table de 64 entrées, la clé de hash coding pourra être obtenue par intersection sur le dernier octet du nom interne (8 cycles) ;

l'indexation demande 3 décalages et une addition (31 cycles) ; la comparaison des noms internes demande 12 cycles par octet. Si le hash coding est bien fait, les listes de synonymes seront très courtes et on peut espérer s'en tirer en moyenne en moins de deux comparaisons, ce qui fait au maximum 150 cycles.
Pour insérer une entrée, il faut rechercher une entrée libre dans la table; pour une table pleine à 50 %, on peut trouver une entrée libre par recherche séquentielle en moins de deux comparaisons, en moyenne, ce qui fait 63 cycles. La création de l'entrée va ensuite consister à positionner les 8 octets, et peut être mettre à jour un chaînage de synonymes : cela va demander environ 90 cycles.

Pour insérer un contexte dans une file d'attente, il faut d'abord rechercher la queue de la liste. La liste étant généralement très courte, au maximum 1 contexte en moyenne, cette recherche sera rapide, 31 cycles. L'insertion proprement dite prendra ensuite 50 cycles.

L'extraction prendra environ 30 cycles.

On est donc ainsi en mesure d'évaluer la durée des diverses opérations :

Durée totale des opérations (arrondie) :

	sur réalisation	sur dictionnaire
MAJ	400	300
LECT	400	250
NOMAJ	250	500
NOLECT	200	50

Dans le cas du dictionnaire, on ne recherche pas le descripteur, mais les files d'attente peuvent être beaucoup plus longues.

A3-6 - AIGUILLAGE DES REQUETES - Cette opération prend 56 cycles.

BIBLIOGRAPHIE

[AB] J.R. ABRIAL, J. BAS, G. BEAUNE, G. HENNERON, R. MORIN, G. VIGLIANO
Projet SOCRATE : Spécifications générales
Grenoble, Août 1970.

[AC] J.R. ABRIAL, J.P. CAHEN, J.C. FAVRE, D. PORTAL, G. MAZARE, R. MORIN
Projet SOCRATE : Nouvelles spécifications
Grenoble, Septembre 1972.

[AN1] F. ANCEAU
Contribution à l'étude des systèmes hiérarchisés de ressources dans l'architecture des machines informatiques
Thèse de Doctorat d'Etat, Grenoble, 5 Décembre 1974.

[AN2] F. ANCEAU
Le monde des microprocesseurs
Rapport de Recherche n° 46, Grenoble, Octobre 1976.

[AD] D.R. ANDERSON
Data base processor technology
National Computer Conference, 1976.

[BS] G. BERGER SABBATEL, M. SARRE
MAGE : Matériel Adapté à la Gestion d'Entités
Journée "Processeurs de base de Données", AFCET, Institut de Programmation, Paris, 18 Mars 1977.

[CH] R.M. CANADAY, R.D. HARRISSON, E.L. IVIE, J.L. RYDEE, L.A. WEBER
A back-end computer for data base management
C.ACM, Octobre 1974.

[CG] CHEVREUL, GACHES, GIDROL, ANCEAU, PERRONNEAU, POUJOULAT
Unité de contrôle évoluée
Contrat SESORI 1.70, Rapport final, Septembre 1973.

[DE] P.J. DENNING
Effects of scheduling on file memory operation
SJCC, 1967.

[PO] G.H. POUJOULAT
The CORAIL building block system
EUROMICRO Newsletter, Octobre 1976.

[PR] E. PRESSON
Fiches de microprocesseurs
Rapport de recherche n° 52, Grenoble, Novembre 1976.

[SE] SAGEM - ENSIMAG
Etude et évaluation sur maquette de l'implantation hardware d'algorithmes de gestion de fichiers adaptés aux petits systèmes
Contrat IRIA 74.136, Rapport final, Juillet 1976.

[SC] J.P. SCHOELLKOPF
Machine PASC.HLL : définition d'une architecture pipe-line pour une unité centrale adaptée au langage PASCAL
Thèse de Troisième Cycle, Grenoble, 28 Juin 1977.

[TP] T.J. THEOREY, T.B. PINKERTON
A comparative analysis of disk scheduling policies

[MR] G. MICHEL, P. ROLIN, C. WAGNER
Microprocessor based disk system
Microprocessors, Septembre 1976.

[BO] D. BORRIONE
LASCAR : un langage pour la simulation et l'évaluation des architectures d'ordinateurs
Thèse de Troisième Cycle, Grenoble, 16 Avril 1976.

[UN] L. UNGARO
Processeur associatif pour bases de données réparties
Journée "Processeurs de base de données", AFCET, Institut de Programmation, Paris, 18 Mars 1977.

[QS] H.M. QUANG, J. SUCHARD, J. VIDALIN
Un processeur associatif itératif et modulaire de reprise de données
Journée "Processeurs de base de données", AFCET, Institut de Programmation, Paris, 18 Mars 1977.

[LI] G.J. LIPOVSKI
Non-numeric architecture
Department of Electrical Engineering, University of Texas.

[JU] J.E. JULIUSSEN
Magnetic buble systems approach practical use
Computer Design, Octobre 1976.

[MA] G. MAZARE
MCS, A symmetric multi-microprocessor system
Congrès EUROMICRO, Venise, Octobre 1976.

[SO] S.A. SCHUSTER, E.A. OZKARAHAN, K.C. SMITH
A virtual memory system for a relationnal associative processor
AFIPS National Computer Conference, 1976.

[NA] Ph. NAVAUX
Processeur base de données MAGE, étude du matériel
Rapport de DEA, Grenoble, Juin 1977.

[HA] J.W. HAVENDER
Avoiding deadlocks in multitasking systems
IBM System Journal, n° 2, 1968.

[LS] C.S. LIN, C.P. SMITH, J.M. SMITH
The design of a rotation associative memory for relational data base applications
ACM Transactions on Database Systems, vol 1, n° 1, Mars 1976, pp. 53-65.

[SL] S.Y.W. SY, G.J. LIPOVSKI
CASSM : a cellulor system for very large data bases
Proceedings of International Conference on Very Large Data Base.

[CO] E.F. CODD
A relational model of data for large shared data banks
C.ACM, Juin 1970.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU le rapport de présentation de Monsieur :

- F. ANCEAU, Maître de Conférences à l'Institut National Polytechnique de GRENOBLE

Monsieur Gilles BERGER SABBATEL

est autorisé à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Génie Informatique".

Grenoble, le 14 Juin 1978

Ph. TRAYNARD
Président
de l'Institut National Polytechnique