# Editeurs par contexte pour systèmes conversationnels à partage de temps

Michel Adiba

N° D'ordre

# THESE

*présentée à*

L'UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

*pour obtenir*

LE GRADE DE DOCTEUR DE TROISIEME CYCLE

"Informatique"

par

**Michel ADIBA**

# Editeurs par contexte pour systèmes conversationnels à partage de temps

Thèse soutenue le 24 Avril 1971 devant la commission d'examen :

| | |
|---|---|
| Monsieur N. GASTINEL | Président |
| Messieurs L.BOLLIET | Examinateur |
| Y.SIRET | Examinateur |
| C.HANS | Examinateur |

FACULTE DES SCIENCES
DE GRENOBLE

# LISTE DES PROFESSEURS

<> <> <>

## PROFESSEURS TITULAIRES

| | | |
|---|---|---|
| MM. | NEEL Louis | Physique expérimentale |
| | KRAVTCHENKO Julien | Mécanique rationnelle |
| | CHABAUTY Claude | Calcul différentiel et intégral |
| | BENOIT Jean | Radioélectricité |
| | CHENE Marcel | Chimie papetière |
| | FELICI Noël | Electrostatique |
| | KUNTZMANN Jean | Mathématiques Appliquées |
| | BARBIER Reynold | Géologie appliquée |
| | SANTON Lucien | Mécanique des fluides |
| | OZENDA Paul | Botanique |
| | KOSZUL Jean Louis | Mathématiques pures |
| | GALVANI Octave | Mathématiques pures |
| | MOUSSA André | Chimie nucléaire et radioactivité |
| | TRAYNARD Philippe | Chimie générale |
| | SOUTIF Michel | Physique générale |
| | CRAYA Antoine | Hydrodynamique |
| | REULOS René | Théorie des Champs |
| | BESSON Jean | Chimie minérale |
| | AYANT Yves | Physique approfondie |
| | GALLISSOT François | Mathématiques pures |
| Mle. | LUTZ Elisabeth | Mathématiques pures |
| MM. | BLAMBERT Maurice | Mathématiques pures |
| | BOUCHEZ Robert | Physique nucléaire |
| | LLIBOUTRY Louis | Géophysique |
| | MICHEL Robert | Minéralogie et pétrographie |
| | BONNIER Etienne | Electrochimie-Electrométallurgie |
| | DESSAUX Georges | Physiologie animale |
| | PILLET Emile | Physique industrielle |
| | YOCCOZ Jean | Physique nucléaire théorique |
| | DEBELMAS Jacques | Géologie générale |
| | GERBER Robert | Mathématiques pures |
| | PAUTHENET René | Electrotechnique |
| | MALGRANGE Bernard | Mathématiques pures |
| | VAUQUOIS Bernard | Calcul électronique |
| | BARJON Robert | Physique nucléaire |
| | BARBIER Jean Claude | Physique expérimentale |
| | SILBER Robert | Mécanique des fluides |
| | BUYLE-BODIN Maurice | Electronique |

| | | |
|---|---|---|
| MM. | SARROT-REYNAULD Jean | Géologie |
| | BERTRANDIAS Jean Paul | Mathématiques appliquées |
| | AUBERT Guy | Physique |
| | GAUTHIER Yves | Sciences biologiques- C.S.U. |
| | DOLIQUE Jean Michel | Physique des plasmas |
| | DESRE Pierre | Métallurgie |
| | LAURENT Pierre | Mathématiques appliquées |
| | CARLIER Georges | Biologie végétale |
| | SIBILLE Robert | Construction mécanique-I.U.T."A" |

## MAITRES DE CONFERENCES

| | | |
|---|---|---|
| MM. | LANCIA Roland | Physique automatique |
| Mme | BOUCHE Liane | Mathématiques. C.S.U.Chambery |
| MM | KAHANE André | Physique générale |
| | BRIERE Georges | Physique expérimentale |
| M. | LAJZEROWICZ Joseph | Physique |
| Mme | BERTRANDIAS Françoise | Mathématiques pures |
| MM. | LONGEQUEUE Jean Pierre | Physique nucléaire |
| | ZADWORNY François | Electronique |
| | DURAND Francis | Métallurgie |
| | PFISTER Jean Claude | Physique générale |
| | CHIBON Pierre | Biologie animale |
| | IDELMAN Simon | Physiologie animale |
| | BLOCH Daniel | Electrotechnique- I.P. |
| | MARTIN-BOYER Michel | Chimie-C.S.U.Chambery |
| | BRUGEL Lucien | Energétique- I.U.T."A" |
| | BOUVARD Maurice | Mécanique des fluides |
| | ARMAND Yves | Chimie-I.U.T."A" |
| | KUHN Gérard | Physique- I.U.T."A" |
| | RICHARD Lucien | Botanique |
| | PELMONT Jean | Physiologie animale |
| | BOUSSARD Jean Claude | Mathématiques appliquées.I.P. |
| | MOREAU René | Hydraulique. I.P. |
| | GERMAIN Jean Pierre | Mécanique |
| | JOLY Jean René | Mathématiques pures |
| Mle | PIERY Yvette | Biologie animale |
| MM | CONTE René | Physique I.U.T."A" |
| | PEFFEN René | Métallurgie I.U.T. "A" |
| | LE JUNTER Noël | Electronique I.U.T. "A" |
| | VIALON Pierre | Géologie |
| | BENZAKEN Claude | Mathématiques appliquées |
| | MAYNARD Roger | Physique |
| | BLIMAN Samuel | Electronique E.I.E. |
| | DUSSAUD René | Mathématiques C.S.U. Chambery |
| | BELORIZKY Elie | Physique |
| Mme | LAJZEROWICZ Jeannine | Physique |
| M. | JULLIEN Pierre | Mathématiques pures |
| Mme | RINAUDO Marguerite | Chimie macromoléculaire |
| MM | LEROY Philippe | Mathématiques I.U.T. "A" |
| | BRODEAU François | Mathématiques I.U.T. "B" |

| | | |
|---|---|---|
| | DREYFUS Bernard | Thermodynamique |
| | KLEIN Joseph | Mathématiques pures |
| | VAILLANT François | Zoologie |
| | ARNAUD Paul | Chimie |
| | SENGEL Philippe | Zoologie |
| | BARNOUD Fernand | Biosynthèse de la cellulose |
| | BRISSONNEAU Pierre | Physique du solide |
| | GAGNAIRE Didier | Chimie physique |
| Mme | KOFLER Lucie | Botanique et physiologie végétale |
| MM. | DEGRANGE Charles | Zoologie |
| | PEBAY-PEROULA Jean Claude | Physique |
| | RASSAT André | Chimie systématique |
| | DUCROS Pierre | Cristallographie |
| | DODU Jacques | Mécanique appliquée- I.U.T."A" |
| | ANGLES D'AURIAC Paul | Mécanique des fluides |
| | LACAZE Albert | Thermodynamique |
| | GASTINEL Noël | Analyse numérique |
| | GIRAUD Pierre | Géologie |
| | PERRET René | Servomécanisme |
| | PAYAN Jean Jacques | Mathématiques pures |
| | CAUQUIS Georges | Chimie organique |
| | RENARD Michel | Thermodynamique |
| | BONNET Georges | Electrotechnique |
| | BOLLIET Louis | Informatique- I.U.T."B" |

## PROFESSEURS ASSOCIES

| | | |
|---|---|---|
| MM. | RADHAKRISHNA | Thermodynamique |
| | BULLEMER Bernhard | Physique |

## PROFESSEURS SANS CHAIRE

| | | |
|---|---|---|
| M. | GIDON Paul | Géologie et minéralogie- C.S.U. |
| Mme | BARBIER Marie Jeanne | Electrochimie |
| Mme | SOUTIF Jeanne | Physique générale |
| MM. | COHEN Joseph | Electrotechnique |
| | DEPASSEL Roger | Mécanique des fluides |
| | GLENAT René | Chimie organique |
| | BARRA Jean | Mathématiques appliquées |
| | COUMES André | Electronique- E.I.E. |
| | PERRIAUX Jacques | Géologie et minéralogie |
| | ROBERT André | Chimie papetière |
| | BIAREZ Jean Pierre | Mécanique |
| | BONNETAIN Lucien | Chimie minérale |
| | HACQUES Gérard | Calcul numérique |
| | POULOUJADOFF Michel | Electrotechnique |
| Mme | KAHANE Josette | Physique |
| Mme | BONNIER Jane | Chimie générale |
| MM. | VALENTIN Jacques | Physique nucléaire |
| | REBECQ Jacques | Biologie- C.S.U.Chambery |
| | DEPORTES Charles | Chimie minérale |

| | | |
|---|---|---|
| MM | ROMIER Guy | Mathématiques I.U.T. "B" |
| | NEGRE Robert | Mécanique I.U.T. "A" |
| | BEGUIN Claude | Chimie organique |
| | BUISSON Roger | Physique I.U.T."A" |
| | IVANES Marcel | Electricité I.U.T. "A" |
| | GIDON Maurice | Géologie |
| | COHEN ADDAD | Spectrométrie physique |
| | VAN CUTSEM Bernard | Mathématiques appliquées |
| | GRIFFITHS Michael | Informatique |
| | MACHE Regis | Physiologie végétale |
| | GENSAC Pierre | Sciences biologiques |
| | JOUBERT Jean Claude | Physique du solide I.P |
| | VEILLON Gerard | Mathématiques appliquées I.P. |
| | MARECHAL Jean | Physique mécanique I.U.T. "A" |
| | PECCOUD François | Analyse I.U.T. "B" |
| | CHIAVERINA Jean | Biologie appliquée E.F.P. |

## MAITRES DE CONFERENCES ASSOCIES

| | | |
|---|---|---|
| MM. | CHEEKE John | Thermodynamique |
| | BOUDOURIS Georges | Radioélectricité |
| | BENENSON Walter | Physique nucléaire |
| | GOLDSCHMIDT H. | Mathématiques |

# TABLE DES MATIERES

<><><><><><>

Pages

DEUXIEME PARTIE

*A la mémoire de mon père,*

*à ma mère.*

*Je tiens à remercier :*

*Monsieur le Professeur Noël GASTINEL, Directeur du Laboratoire de Calcul, qui m'a fait l'honneur de présider le jury de Thèse et qui a toujours accordé un vif intérêt à mon travail.*

*Monsieur Louis BOLLIET, Professeur à l'Institut Universitaire de Technologie de GRENOBLE, qui m'a orienté vers la recherche en programmation et m'a encouragé dans mes travaux.*

*Monsieur Yvon SIRET, Docteur es-Sciences, qui s'est intéressé avec sympathie à mon travail et a aimablement accepté de faire partie du Jury.*

*Je veux également exprimer toute ma reconnaissance à Monsieur Claude HANS, du Centre Scientifique I.B.M. de GRENOBLE, qui a eu la lourde tache de me diriger dans mes recherches et qui ne m'a ménagé ni son temps ni ses conseils.*

*Mes remerciements vont également à mon camarade Jean Pierre LE HEIGET pour son aide efficace ainsi qu'à tous mes autres collègues du Laboratoire et en particulier Messieurs BELLOT, LECARME, LUCAS.*

*Je tiens enfin à remercier les services de dactylographie et de tirage à qui je dois la réalisation matérielle de cet ouvrage.*

*" Ils viendront avec l'image*
*qu'ils ont dans le coeur*
*réordonner selon le sens nouveau*
*les caractères anciens du livre ".*

*"Citadelle" -A.de SAINT-EXUPERY-*

# INTRODUCTION

Un des principaux buts d'un système conversationnel, à partage de temps, est de fournir à un ensemble d'utilisateurs, non nécessairement experts en informatique, un jeu de programmes et de facilités leur permettant de résoudre leurs problèmes spécifiques.

Les temps où le programmeur devait transporter de gros bacs de cartes perforées, contenant ses programmes, sont révolus. Désormais lorsque l'utilisateur d'un système conversationnel se trouve devant son terminal(machine à écrire, console de visualisation etc...) ses programmes, ses données sont conservés sous forme de fichiers dans une mémoire secondaire (des disques magnétiques par exemple).

L'accès à de tels fichiers ne peut alors se faire que par l'intermédiaire d'un programme, fourni par le système, et appelé généralement : "Editeur". Ce dernier permet de créer, d'examiner et de modifier les éléments des fichiers au moyen d'une série de commandes.

Il s'agit là , certes d'un programme disposant de qualités conversationnelles, mais essentiellement orienté vers le traitement de caractères. A ce propos, l'utilisation d'un calculateur pour manipuler de l'information purement alphanumérique se développe: Certains journaux aux Etats-Unis confient à un ordinateur la mise en page de leurs articles [L9]. Pour résoudre plus facilement les problèmes posés par le traitement de caractères dans des domaines divers (enseignement programmé- compilation- banques de données- traduction automatique) les langages de programmation comportent de plus en plus des opérations sur les chaines (PL/1- ALGOL 68) tandis que se développent des langages spécialisés (SNOBOL4).

Un éditeur doit donc permettre la réalisation d'opérations diverses

sur des caractères mais ceci au moyen d'un langage simple pour augmenter les qualités conversationnelles du système.

Ceci a guidé au départ notre étude.

En examinant les éditeurs fonctionnant déjà sous divers systèmes, il nous est apparu utile de réaliser un éditeur sur un terminal graphique évolué (IBM 2250-1) en développant et l'interaction homme-machine, et les facilités du langage d'édition.

L'éditeur "graphique" apporte à l'utilisateur une dimension nouvelle en lui fournissant une sorte de fenêtre sur l'ensemble de ses fichiers. Il peut ainsi les examiner, les modifier et juger immédiatement du résultat obtenu. Quant au langage d'édition, il n'est vraiment puissant que s'il s'adapte parfaitement aux besoins de chaque utilisateur. Nous avons donc transformé un éditeur "classique" en un "macro-éditeur" élargissant ainsi les domaines d'applications. En effet, au moyen d'un langage de base, il devient possible de définir, de conserver et d'utiliser des nouvelles commandes. Le macro-éditeur est capable de réaliser l'enchaînement automatique d'opérations d'édition sur plusieurs fichiers, sous contrôle de l'utilisateur.

En faisant le bilan de ces deux réalisations pratiques, nous avons pu poser un certain nombre de problèmes relatifs à l'édition et dresser un véritable cahier des charges d'un éditeur "idéal".

A partir de ce cahier des charges, nous avons défini une "machine d'édition" dont la structure modulaire permet une indépendance quasi-totale vis à vis de tout système ou de tout type de terminal. La présence sur cette "machine" de fonctions primitives et d'un mécanisme de "micro-programmation" permet de fournir à l'utilisateur un langage d'édition facilement extensible et augmente considérablement la portée de l'éditeur qui devient alors un véritable système conversationnel pour le traitement de caractères.

Pour l'élaboration de mon travail, le fait d'être intégré à l'équipe

assurant la maintenance des systèmes CP/CMS (Control Program/Cambridge Monitor System) à l'Institut de Mathématiques Appliquées de Grenoble, m'a permis d'acquérir une certaine expérience des systèmes conversationnels et d'utiliser de façon intensive l'éditeur de C.M.S.

Au cours de plusieurs discussions avec les différents utilisateurs j'ai été mis en présence de leurs problèmes particuliers concernant la manipulation conversationnelle de fichiers. En fait, la "machine d'édition" est née de la synthèse des idées dégagées de mes réalisations pratiques et des nombreuses suggestions formulées par les utilisateurs.

# P R E M I E R E  P A R T I E

## ETUDES ET REALISATIONS

### CHAPITRE I : LES PROGRAMMES D'EDITION

1. EVOLUTION DES PROGRAMMES D'EDITION

2. LES EDITEURS DE QUELQUES SYSTEMES CONVERSATIONNELS

   2.1 TECO( MAC)

   2.2 QED(CMSS)

   2.3 ED( MTS)

3. L'EDITEUR DE CMS

   3.1 GENERALITES

   3.2 LANGAGE DE COMMANDE

   3.3 CONCEPTION INTERNE

4. TRAITEMENT DE CARACTERES

   4.1 LES LANGAGES SPECIALISES

   4.2 LANGAGE D'EDITION

5. UTILISATIONS DES PROGRAMMES D'EDITION

   5.1 MISE AU POINT DE PROGRAMME

   5.2 MISE EN FORME DE DONNEES

   5.3 EXPLOITATION DE RESULTATS

   5.4 MISE EN PAGE DE TEXTE

# C H A P I T R E  I

# LES PROGRAMMES D'EDITION

## 1. EVOLUTION DES PROGRAMMES D'EDITION.

Un programme d'édition met, en général, à la disposition de l'utilisateur un jeu de commandes (nous dirons des "requêtes") qui permettent la création, l'examen et la modification des fichiers.

Un fichier est considéré le plus souvent comme un ensemble de N éléments rangés dans un certain ordre et possédant une série de caractéristiques (format fixe ou variable, longueur etc...) Ces éléments sont,en général, des lignes et lorsque l'on émet une requête il faut indiquer la ligne sur laquelle, ou à partir de laquelle on veut faire agir cette requête. La méthode la plus simple consiste à donner le rang de la ligne dans le fichier. Cette sorte d'adressage par numéro de ligne est analogue à celles des programmes de mise à jour automatique, utilisés dans les systèmes non conversationnels. C'est bien sûr, une méthode peu souple à utiliser lorsqu'il s'agit de réaliser une véritable interaction, le rang d'une ligne pouvant évoluer au cours des insertions ou suppressions. Dans les systèmes conversationnels est apparu un nouveau type d'éditeur appelé "éditeur par contexte" qui permet d'adresser une ligne au moyen de tout ou partie de son contenu. On peut ainsi rechercher dans un fichier la ligne contenant une occurrence d'une chaîne donnée ou qui correspond à un certain modèle.

Par exemple : Une ligne commençant par 3 caractères alphabétiques suivis de 3 chiffres.

Les recherches par contexte et les commandes correspondantes se sont développées comme nous le verrons par la suite. Les éditeurs peuvent également reconnaître le type du fichier qu'ils ont à traiter, et, en conséquence, associer des caractéristiques particulières: intervalles de tabulation, colonne de troncature, longueur des enregistrements etc...

Les deux premiers éditeurs que nous allons décrire utilisent comme terminal de communication un "teletype ASR 33".

La technologie de l'unité de contrôle liée à ce terminal est telle que chaque caractère tapé (lettre, chiffre, caractère spécial) est envoyé immédiatement au calculateur.

Ainsi, les commandes d'édition sont désignées le plus souvent par un seul caractère.

Ceci explique le langage quelque peu "hermétique" des éditeurs "TECO" et "QED".

Dans l'éditeur du système C.M.S les terminaux (IBM 2741-1 ou teletype ASR-33) sont connectés différemment et seul le caractère "retour-chariot" provoque l'envoi d'une ligne complète (132 caractères au maximum) au calculateur.

## 2. LES EDITEURS DE QUELQUES SYSTEMES CONVERSATIONNELS.

### 2.1 "TECO"

Dans le projet "MAC" du M.I.T on trouve la description d'un éditeur par contexte, "TECO" (Tape Editor and Corrector) fonctionnant sur un PDP-6 et utilisant comme moyens de communication avec l'utilisateur d'une part un terminal graphique pour examiner le fichier et d'autre part, un "teletype" pour la frappe des requêtes et l'impression des messages. [E3]

Les fichiers sont soit sur bandes perforées soit sur bandes magnétiques. Pour TECO un fichier est une chaîne de caractères arbitrairement grande dans laquelle le caractère "retour-chariot" permet de séparer logiquement les lignes. L'édition se fait page par page au moyen de trois zones de mémoire :

1. Zone fichier : Elle contient la page en cours d'édition. Le contenu de cette zone apparaît sur l'écran cathodique du terminal graphique. C'est dans cette zone que le texte est modifié. On y associe un pointeur qui, à

un instant donné, se trouve entre deux caractères consécutifs. La valeur de ce pointeur est en fait le nombre de caractères qui se trouvent à sa gauche. Ce pointeur peut être déplacé à volonté par l'utilisateur.

Une fois examinée et modifiée la page est remise sur bande avant d'amener la page suivante.

2. Zone de commande: Elle est destinée à recevoir une chaîne de caractères qui sera interprétée comme une requête avec ses paramètres.

3. 36 registres: Ils servent à conserver certains paramètres nécessaires à l'édition. Chaque registre peut contenir un nombre ou une chaîne. On les désigne symboliquement dans le langage de commande par un caractère (chiffres 0 à 9; lettres de A à Z).

L'utilisation de l'éditeur se fait au moyen de 4 types de commandes :

a) Les commandes de gestion globale des fichiers : copie, suppression, gestion du répertoire.

b) Les commandes d'entrée, pour préciser l'emplacement et le nom du fichier source.

c) Les requêtes d'édition pour examiner et modifier le texte contenu dans la zone fichier.

d) Les commandes de sortie, pour contrôler l'affichage des caractères sur l'écran et pour définir l'emplacement du fichier modifié.

Le langage d'édition utilise tous les caractères du télétype y compris les caractères de contrôle (non imprimables).

Une requête est une chaîne de caractères (sans blancs) dans laquelle la fonction demandée est identifiée par un seul caractère.

Par exemple : "nI" signifie : insérer le caractère de code n (code ASCII) à gauche du pointeur.

Ainsi la frappe des requêtes est rapide à condition, pour l'utilisateur

de se souvenir des symboles de chaque fonction.

On peut classer les requêtes d'édition en différents groupes :

- Mouvements du pointeur: Avec un nombre en paramètre: déplacements absolus ou relatifs. Avec une chaîne: recherches par contexte. Ces recherches se font dans la zone fichier où les pages successives sont amenées automatiquement. On a la possibilité de rechercher dans tout un fichier la $n^{ième}$ occurrence d'une chaîne et, selon le résultat, la requête fournit une valeur (-1 si la recherche est positive, 0 sinon). Cette valeur peut être testée au moyen d'un mécanisme de macros (voir plus loin).

- Modifications: suppression, insertion et remplacement au niveau des caractères mais également au niveau des lignes. Le paramètre d'une requête de modification: ou bien se trouve dans l'un des 36 registres dont on donne alors le nom symbolique, ou bien, il est tapé après l'émission du caractère identifiant la fonction demandée.

- Contrôle de l'édition: chargement des registres. Impression de leur contenu. Définition de petits programmes d'édition (macro-requêtes) Ce mécanisme de macro-requêtes permet :

1/ D'interpréter le contenu d'un registre comme une série de requêtes à exécuter.

2/ De réaliser des boucles, des tests (recherches par contexte) sur les opérations d'édition.

En résumé, on peut dire que TECO est un éditeur par contexte très complet. Les registres permettent, à l'utilisateur, de conserver des quantités dont il pourra se servir plus tard, au cours de l'édition. Le mécanisme de macro-requête augmente considérablement la portée de l'éditeur.

On peut regretter cependant le côté hermétique du langage de commande venant d'une trop grande simplification de la syntaxe.L'utilisation de toutes les ressources d'un tel éditeur doit demander une grande pratique.

*Exemples* :

- Compter le nombre de lignes dans une page:

```
JOUN<:S$;%N>QN=
```

- Remplacer dans toute une page la lettre "A" par la lettre "B" lorsque le A est suivi d'un chiffre (code $48_{10}$ à $57_{10}$) :

```
J<:SA$;!A-47"GIA-58"L-1DIB$''>!
```

N.B. Le "!" ne fait pas partie de la commande !

## 2.2 "Q.E.D"

Développé à l'Université de Berkeley dans le cadre du système SDS-930, "QED" [E1] est un éditeur par contexte dont on retrouve une version améliorée dans le système C.T.S.S [E2] .

Le fichier à éditer est découpé en 128 "buffer": 1 buffer courant et 127 buffers auxiliaires. Chacun d'eux porte un nom symbolique et contient une chaîne de caractères, le retour chariot joue le rôle de séparateur de lignes.

Les requêtes agissent sur les lignes du buffer courant mais peuvent utiliser de l'information contenue dans un buffer auxiliaire. Chaque buffer auxiliaire peut d'ailleurs devenir buffer courant.

La syntaxe d'une requête de QED est précise : 3 champs:

1- Adresse
2- Opération
3- Paramètres

### 1. Adressage.

La partie adresse indique la ligne ou le groupe de lignes (dans le buffer courant) concernées par l'opération. L'adressage se fait :

. soit par numéro de ligne (relatif-absolu)

. soit symboliquement: le '.' est la ligne courante, le '$' la dernière ligne du buffer courant.

. soit par contexte au moyen "d'expressions régulières" (voir plus loin).

2. Opération

Elle est désignée par un caractère et, selon le type de l'opération, des paramètres peuvent être demandés par l'éditeur.

On retrouve les requêtes classiques de suppression, remplacement et insertion de lignes, mais également des opérations de substitution de chaînes à d'autres chaînes dans un groupe de lignes.

3. Paramètres

Ils sont donnés soit dans la requête elle-même soit après émission de celle-ci. On y trouve :

. des noms de buffer

. des lignes complètes

. des chaînes de caractères et surtout des expressions régulières pour les recherches par contexte.

Expressions régulières

Ces expressions permettent de construire des modèles de lignes, à partir

1- des caractères standards

2- d'un jeu d'opérateurs, à savoir

. Le choix " | "

. La répétition "*"

. La concaténation (implicite)

3- de caractères spéciaux :

'.' qui correspond à un caractère quelconque

"¢a" qui correspond à 1 caractère alphabétique quelconque

"¢n" qui correspond à 1 caractère numérique quelconque

$ qui correspond au caractère précédant le 1$^{er}$ d'une ligne

¬ qui correspond au caractère suivant le dernier d'une ligne.

*Exemples de modèles* :

1 ligne quelconque $ . * ¬

3 premiers caractères d'une ligne $ . . .

Suite de lettres suivie d'une suite de chiffres ¢a*¢n*

De telles recherches agréables pour l'utilisateur, sont très couteuses au niveau de l'éditeur qui doit effectuer une véritable compilation des expressions régulières.

Un éditeur tel que QED apporte donc un certain formalisme sur le plan du langage d'édition. Il en résulte une plus grande clarté des commandes et des opérations plus puissantes. Cependant l'utilisation complète de l'éditeur demande aussi une grande pratique.

## 2.3 EDITEUR DU SYSTEME M.T.S [E.6] (Michigan Terminal System)

L'éditeur par contexte du système MTS présente des particularités intéressantes :

2.3.1 La syntaxe des requêtes est bien définie:

- 1 nom suivi ou non de modificateurs
- Une série de paramètres de l'un des 5 types suivant: Chaîne, numéro de ligne, nombre, nom de région, nom de macro-requête.

Les modificateurs que l'on peut appliquer au nom de la requête permettent de diminuer ou d'augmenter l'effet de celle-ci.

Par exemple la requête "SCAN" est définie pour rechercher dans le domaine caractérisé par le premier paramètre, une occurrence de la chaîne donnée en paramètre.

Le modificateur '@' provoquera la recherche de toutes les occurrences:

SCAN@A /FILE 'ABC'

Recherche dans tout le fichier (FILE) de toutes les occurrences de la chaîne 'ABC'.

2.3.2 Le domaine d'application de chaque fonction est, par défaut, la ligne courante, de gauche à droite. Toutefois on peut préciser une zone d'application différente;groupe de lignes et colonnes. En donnant deux numéros de lignes pour définir une zone d'application, on retrouve la notion d'adresse de QED.

La possibilité de désigner symboliquement une région du fichier est une caractéristique intéressante de cet éditeur.

Un nom de région, par exemple: "tout le fichier: /FILE", est valide comme paramètre des requêtes de recherche par contexte ou de modification L'utilisateur peut affecter à tout groupe de lignes un nom symbolique de son choix.

2.3.3 Pour obtenir des explications sur la façon d'utiliser l'éditeur ou une requête particulière (syntaxe-erreurs) on peut se servir de la requête "EXPLAIN".

2.3.4 L'éditeur peut s'utiliser en mode conversationnel ou en exploitation automatique (de type "batch-processing"). Il peut en outre travailler dans un mode où l'on conserve l'état initial des lignes avant modification. Ainsi en cas d'erreur ou de fausse manoeuvre on peut retrouver les lignes dans leur état initial.

2.3.5 Il existe un système de macro-requête qui permet de programmer une série de requêtes sous un même nom. Toutefois on ne peut conserver ces macro-requêtes ni leur passer des paramètres lors de l'appel.

## 3. L'EDITEUR DE CMS

### 3.1 GENERALITES

Le système C.M.S.(Cambridge Monitor System) conversationnel et non à partage de temps, utilisé à Grenoble sous le système CP/67 générateur de

machines virtuelles, nous a fourni, avec ses différentes versions d'éditeur, une base de travail [S1-S5-S6-E4-E7]

Pour l'éditeur 'EDIT' de CMS un fichier, caractérisé par un nom et un type, est un ensemble de N lignes auquel est associé un pointeur. A un instant donné la ligne repérée par le pointeur est dite 'ligne courante'.

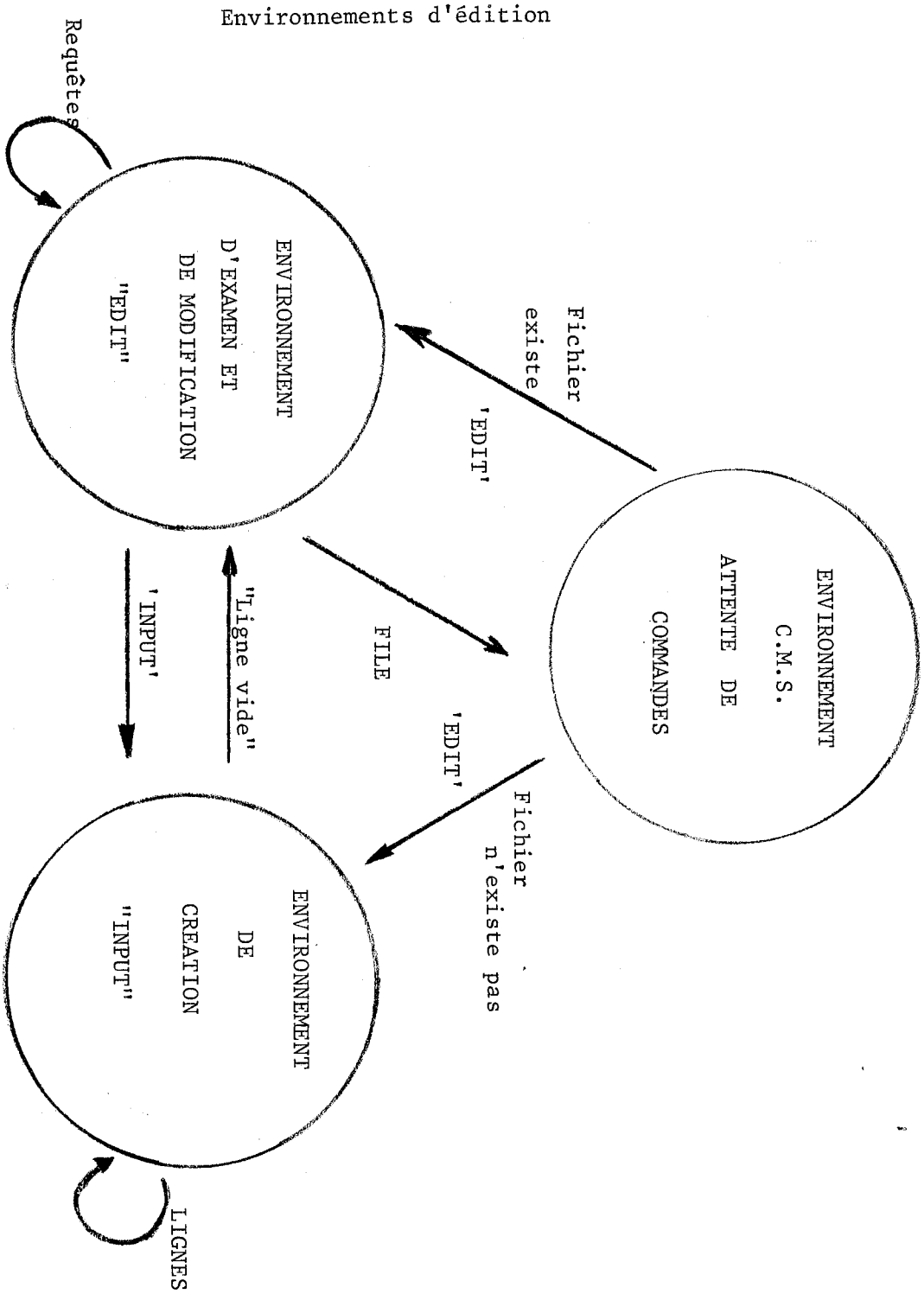
L'appel de l'éditeur au terminal se fait en précisant un nom et un type de fichier.-Si le fichier indiqué n'existe pas encore, il y a passage dans un environnement de création (INPUT) dans lequel toute ligne tapée est considérée comme donnée et rangée dans le fichier.

-Si le fichier spécifié existe déjà il y a passage dans un environnement d'examen et de modification (EDIT) dans lequel toute ligne tapée est interprétée comme une requête avec ses paramètres.

Le passage de l'environnement de création à celui de modification se fait en envoyant une ligne vide (c.a.d 2 retour-chariots consécutifs) alors que l'inverse est obtenu par la requête "INPUT" (fig.1).

Figure 1

Environnements d'édition



## 3.2 LANGAGE DE COMMANDE

Il est constitué d'une série de requêtes que l'on classe en 3 catégories:

### 3.2.1 Requêtes de position

Elles permettent de déplacer le pointeur de ligne courante:

- pointage absolu GOTO n rendre la $n^{ième}$ ligne, ligne courante
- pointage relatif $\left\{\begin{matrix}\text{NEXT}\\ \text{UP}\end{matrix}\right\}$ n Descendre ou remonter le pointeur de n lignes.
- Recherches par contexte:

$\left\{\begin{matrix}\text{NEXT}\\ \text{UP}\end{matrix}\right\}$ chaîne Recherche de l'occurrence de la chaîne donnée, vers le bas ou vers le haut du fichier

FIND Masque Recherche d'une ligne dont les caractères correspondent à ceux du masque.

### 3.2.2. Requêtes de modification (Paramètre implicite: la ligne courante).

- Insertion de ligne INSERT ligne
- Suppression DELETE n
- Remplacement REPLACE ligne
- Substitution d'une chaîne par une autre, dans un certain nombre de lignes ou dans tout le fichier :

CHANGE chaîne 1 chaîne 2

- Modification de la ligne courante en accord avec les caractères d'un masque:

$\left\{\begin{matrix}\text{OVERLAY}\\ \text{BLANK}\end{matrix}\right\}$ Masque

- Echange de lignes entre le fichier édité et tout autre fichier
  Requêtes: PUT, PUTD et GET.

3.2.3. Requêtes de Contrôle de l'édition.

- Répétition d'une requête n fois AGAIN n
- Impression de n lignes PRINT n
- Contrôle des messages de l'éditeur: modes muet ou bavard (BRIEF-VERIFY).
- Définition d'intervalles de tabulation TABSET n1 n2 n3 ...
- Redéfinition des caractères spéciaux: tabulation logique espace arrière etc... (TABDEF- BACKSPACE)
- Contrôle de sérialisation du fichier: SERIAL
- Sauvegarde des modifications en quittant ou non l'environnement d'édition (SAVE, FILE)

Le langage est simple à assimiler même pour un utilisateur non-informaticien.

L'éditeur reconnait certains types de fichiers et leur associe une série de caractéristiques: Par exemple le type "SYSIN" correspond à un fichier d'images de cartes (80 colonnes) destiné à l'assembleur IBM 360; les tabulations sont alors 1, 10, 16, 31, 41, 46, 51, 56, 71 (l'éditeur tronque toutes les lignes à 71 caractères).

Il n'est pas possible en cours d'édition de définir de manière durable les caractéristiques d'un nouveau type de fichier.

3.3 CONCEPTION INTERNE

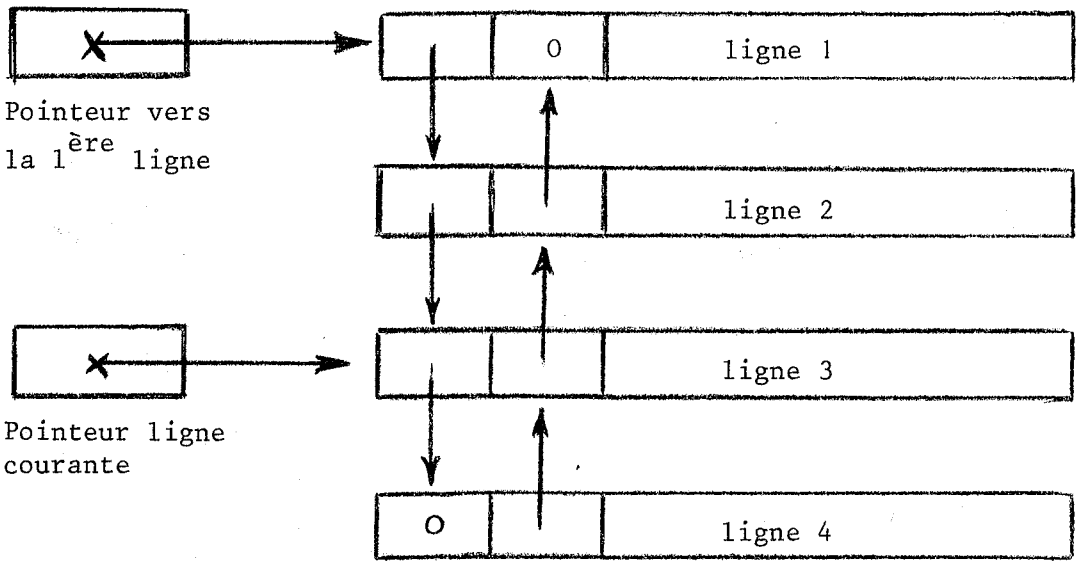
La première version de l'éditeur de CMS n'utilisait que les disques pour gérer ses fichiers de travail. Les temps de réponse étaient longs mais l'on pouvait éditer de très gros fichiers.

La version suivante utilise la mémoire virtuelle (256 K octets pour une machine virtuelle standard) pour installer le fichier à éditer sous

forme d'une liste bi-directionnelle. Ainsi, puisque chaque ligne en mémoire contient un pointeur vers la ligne suivante et un autre vers la ligne précédente (voir fig.2), les requêtes de suppression, insertion, etc... se ramènent à de simples modifications de pointeurs. En outre, les recherches par contexte se font entièrement en mémoire et sont donc plus rapides.

La taille des fichiers à éditer est limitée à environ 2000 lignes (images de cartes 80 colonnes) et la sauvegarde sur disque du nouvel état du fichier, n'est pas automatique mais se fait grâce à l'emploi de requêtes spécialisées (SAVE-FILE).

Fig.2 Structure en mémoire du fichier :




Grâce à son temps de réponse rapide et à la simplicité de son langage, EDIT est agréable à utiliser pour des opérations courantes comme la création de fichiers et les corrections pour la mise au point des programmes.

Cependant il est conçu comme un ensemble de sous-programmes imbriqués, et difficilement maniable en vue de modifications ou d'extensions. On peut remarquer également l'absence d'un mécanisme de macro-commandes que nous devions ajouter par la suite (voir chap. III).

## 4. TRAITEMENT DE CARACTERES

Dans de nombreux langages de programmation on trouve des opérations portant sur des chaînes de caractères. Or dans les éditeurs, il s'agit également d'opérations sur les caractères d'un fichier. On peut se demander alors quelles peuvent être les types d'opérations intéressantes à retirer d'un langage de programmation pour les adapter à un programme d'édition.

### 4.1 LANGAGES DE TRAITEMENT DE CARACTERES

Dans les langages de programmation comme FORTRAN, ALGOL, COBOL etc... les opérations sur les chaînes sont limitées. En PL/1 il existe des variables de type chaîne auxquelles ont peut affecter des valeurs et que l'on peut concaténer entre elles.

Les langages entièrement spécialisés dans le traitement des chaînes de caractères se sont développés à partir de COMIT(1957) [L6] qui fût le premier à introduire les concepts de recherche de modèles et de manipulation de chaînes en tant qu'objets du langage. A partir de 1963, se succédèrent les différentes versions de SNOBOL [L4-L5] dont la dernière SNOBOL4 [L1-L2-L3] correspond à un langage de programmation très complet.

Les caractéristiques principales d'un langage comme SNOBOL4 sont les suivantes :

- Les programmes sont compilés puis interprétés
- Les objets manipulés par le langage sont, en général de type chaînes. Quand ils interviennent dans des opérations arithmétiques, il y a conversion automatique.

- On retrouve les opérations classiques d'affectation, de rupture de séquence, et des opérations arithmétiques simples auxquelles s'ajoutent la concaténation, la décomposition de chaîne en sous-chaînes et surtout les recherches de modèles ou filtrage:

  On désigne ainsi la recherche dans une chaîne donnée, d'un ensemble de caractères qui correspond aux indications fournies par un modèle.

La forme la plus simple de modèle est une chaîne dont on désire rechercher une occurrence dans une chaîne donnée, mais en SNOBOL4 on peut composer des modèles à partir d'autres modèles prédéfinis dans le langage, avec des fonctions, des opérateurs de choix etc...

*Exemple :*

La barre verticale " | " est l'opérateur de choix, le blanc ' ␣ ' est l'opérateur de concaténation. Ainsi le modèle VOYEL correspond à 1 voyelle suivie d'un point :

VOYEL = 'A' | 'E' | 'I' | 'O' | 'U' | 'Y' ␣ '.'

- Les opérations d'entrées-sorties sont simples à programmer mais assez réduites.

- Du fait de l'interprétation des instructions, la définition dynamique de tableaux, de fonctions et même de nouveaux types d'éléments offrent de grandes variétés d'applications.

## 4.2 LANGAGES D'EDITION

L'optique est ici très différente: les caractères à manipuler forment un ensemble structuré (lignes-fichier) et les instructions sont transmises une à une. Si nous désignons par 'requête' une instruction d'un langage de commande conversationnel, chaque requête comporte ou non des paramètres quand elle est transmise à l'éditeur. Ce dernier indique alors à l'utilisateur si

l'opération demandée s'est effectuée correctement, ou envoie des messages d'erreurs.

On retrouve dans chaque éditeur des requêtes pour désigner les parties du fichier que l'on désire modifier par insertion, remplacement, suppression.

Rares sont les éditeurs qui permettent des recherches par contexte évoluées. Le paramètre le plus courant est la chaîne cherchée ou un masque (cf 'EDIT', 'MTS') mais les expressions régulières de 'QED' sont plus intéressantes parce qu'elles se rapprochent davantage des recherches de modèles de SNOBOL4.

Bien qu'un ensemble de requêtes permette le contrôle du fonctionnement global de l'éditeur il n'est pas souvent possible de faire de certaines parties du fichier des valeurs pour des variables du langage d'édition (Affectation). En effet, le transfert d'information se fait surtout dans le sens utilisateur fichier. Or, il est utile, pour certaines applications, d'extraire automatiquement de l'information d'une ligne pour aller placer cette information ailleurs dans le fichier ou dans un autre fichier.

Nous touchons là également au problème de la "programmation" des opérations d'édition qui ne peut se faire qu'avec un éditeur possédant un mécanisme de macro-requêtes. Il faut alors disposer d'un "macro-langage" comportant des instructions de tests, de ruptures de séquences, de boucles contrôlées etc..

Ainsi le langage d'édition pourrait s'orienter peu à peu vers un langage conversationnel pour le traitement des caractères.

Quant aux mises à jour automatiques de fichiers, elles sont réalisées sans problème lorsque l'éditeur peut se "programmer" et enchaîner automatiquement une série de requêtes.

## 5. UTILISATION DES PROGRAMMES D'EDITION

Chaque système conversationnel met à la disposition des utilisateurs,

en plus de l'éditeur, un jeu de commandes pour gér. . les fichiers:

- Transfert d'un fichier d'un support à un autre (cartes- disques- bandes- imprimantes etc...)
- Suppression d'un fichier de répertoire
- Concaténation de plusieurs fichiers
- Découpage d'un fichier en un ensemble de sous-fichiers.

Ces commandes font quelquefois partie de l'éditeur. La création d'un fichier dans le système peut se faire de différentes façons: à partir de cartes ou d'une bande mais surtout au moyen de l'éditeur lorsqu'il s'agit de petits fichiers.

Nous distinguerons 4 utilisations différentes pour un éditeur.

### 5.1 MISE AU POINT DE PROGRAMME

Qu'un compilateur, ou plus généralement, un traducteur, soit incrémentiel ou non, il est nécessaire de pouvoir modifier le programme source pour corriger les erreurs et arriver finalement à un programme au point.

Le cycle de commandes: Appel de l'éditeur- compilation- chargement- exécution peut se dérouler un certain nombre de fois. Dans ce cas, les fonctions qui sont les plus demandées à l'éditeur sont d'abord le positionnement par numéro de ligne ou par contexte puis les insertions, suppressions, remplacement ainsi que la substitution de chaînes.

### 5.2 MISE EN FORME DE DONNEES

Les données d'un programme peuvent être créees par l'éditeur, sous forme de fichier, et modifiées selon les résultats obtenus après chaque exécution du programme.

### 5.3 EXPLOITATION DE RESULTATS

Lorsque les résultats produits par un programme sont mis sous

forme de fichier, il est possible de les examiner au moyen d'un éditeur par contexte, mais surtout, il est intéressant, au cours de cet examen d'extraire certaines informations particulières pour constituer d'autres fichiers.

En effet, lorsque l'éditeur permet la définition et l'utilisation de macro-requêtes (cf.Chap.III) la recherche par contexte et les requêtes d'échange entre fichiers permettent, à partir du fichier de résultats, la formation d'autres fichiers pour classement, tri etc...

Ce type d'exploitation est indispensable lorsque le traitement des résultats n'est pas facilement programmable: Mesures de performances d'un système, par exemple.

### 5.4. MISE EN PAGE DE TEXTES.

De plus en plus les calculateurs sont utilisés pour manipuler des caractères: documentation automatique et mise en page de textes. Il s'agit de réaliser, par programme sur un texte donné (manuel, cours, rapport etc...) la mise en page avec justification à droite et à gauche des lignes, cadrage des titres et des paragraphes, numérotation des pages, etc...

Le texte est en général tapé et modifié sous contrôle de l'éditeur. On y inclut des commandes de mise en page destinées au programme qui réalise effectivement l'impression. Ainsi toutes les brochures IBM sont mises en page par le programme "TEXT 360" et des journaux américains utilisent un procédé analogue. [ L8-L9-L10-L11 ] .

Sous le système CMS il existe la commande SCRIPT qui permet la mise en page de n'importe quel texte à condition qu'il ait été tapé sous l'éditeur, sous forme d'un fichier de type "SCRIPT".

# CHAPITRE II : EDITEUR GRAPHIQUE

# CHAPITRE II

# EDITEUR GRAPHIQUE

## 1. EDITION GRAPHIQUE

### 1.1 EXEMPLES D'EDITEURS "GRAPHIQUES"

L'emploi d'un terminal graphique comme moyen de communication entre un utilisateur et ses fichiers, est un atout supplémentaire pour un éditeur

Dans "TECO" déjà, un écran cathodique était utilisé pour examiner les parties du fichier que l'on modifiait à partir d'un "teletype".

Sous "M.T.S", une version de l'éditeur fonctionne en utilisant un terminal graphique complet (Type IBM 2250-1).

Divers éditeurs se sont développés sur des consoles de visualisation alphanumériques (cf.chap.IV) mais également sur des terminaux munis de nombreux dispositifs d'interaction: clavier de touches de fonctions, pointeur optique etc..

- A Yorktown "EDIT1" [G9] est un éditeur expérimental pour IBM 1130 et 2250-IV . Il permet, outre une gestion de fichier classique (mouvements d'un support à un autre), des corrections sur un texte que l'on fait apparaître, par pages, sur l'écran du 2250. Les commandes sont envoyées par le clavier alphanumérique et les fonctions disponibles sont très limitées.

- Une autre expérience intéressante réalisée dans le domaine de la manipulation de textes est celle utilisant un terminal graphique muni d'une tablette (type Sylvania) et d'un stylet [G5-G13]. Les parties successives d'un texte apparaissent sur l'écran et grâce au stylet l'utilisateur peut "dessiner" des symboles de correction analogues à ceux qu'il tracerait si le texte se trouvait sur une feuille de papier.

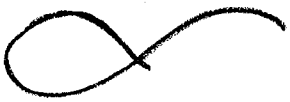
Ainsi pour insérer une chaîne de caractères dans une ligne il faut dessiner le symbole ∧ sous la ligne à modifier. Après reconnaissance de ce symbole l'éditeur fournit un emplacement sur l'écran, où il est possible de composer le texte à insérer.

Les autres symboles permis sont :

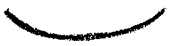


pour supprimer une partie d'une ligne

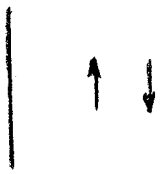
pour échanger des mots ou groupes de mots



pour déplacer une partie du texte



pour supprimer des chaînes de blancs



Dessiné dans le sens de la flèche, ce symbole fait monter ou descendre le texte de 10 lignes.

Pour supprimer une partie du texte.

Ce système oblige l'éditeur à résoudre des problèmes de reconnaissance de forme, mais constitue un moyen naturel pour l'homme de correction et de manipulation de textes, d'où une véritable interaction.

## 1.2 EDITEUR GRAPHIQUE POUR CP/CMS

Après l'étude d'éditeurs par contexte classiques, tels que "QED", "TECO" et surtout "EDIT" de CMS et puisque nous disposions à Grenoble, d'un terminal graphique IBM 2250-1 supporté par CP/CMS, nous avons voulu réaliser

un éditeur graphique dans le cadre de ce système.

Nos buts étaient les suivants :

1) Examiner l'incidence d'un terminal graphique sur l'édition en permettant à l'utilisateur d'avoir devant les yeux, à chaque instant, l'image exacte de l'état de son fichier. Dans les systèmes conversationnels le facteur "temps de réponse" étant un élément psychologique important, la rapidité de l'unité choisie devait être un atout précieux.

2) Autoriser l'utilisateur à agir directement sur le fichier qui apparaît sur l'écran, grâce aux dispositifs d'interaction du terminal clavier alphanumérique, touches de fonction, crayon optique. Il fallait employer tous ces dispositifs de manière naturelle afin que leur utilisation ne soit pas une source de difficultés. Par exemple, le crayon optique pour désigner un caractère ou une ligne à modifier, une touche de fonction pour une opération d'édition courante etc...

3) Dégager, enfin, un certain nombre de fonctions élémentaires, nécessaires à l'édition afin d'établir, par la suite, un langage plus facilement extensible.

Le choix du terminal 2250-1 était délibéré. Il s'agissait en premier lieu d'une étude des diverses formes d'édition, tout en fournissant un outil de mise au point de programmes aux personnes utilisant le 2250.

## 2. DESCRIPTION EXTERNE

### 2.1 MATERIEL UTILISE [G1-G14-G11]

Le 2250-1 est relié à un gros calculateur, en l'occurrence un IBM 360/67. En raison de l'utilisation sur ce calculateur du système CP/67 générateur de machines virtuelles, tout se passe comme si le terminal graphique était relié à un 360 standard (c.a.d ne disposant pas du mécanisme de translation

dynamique d'adresse propre au 360/67) de 256 K-octets de capacité mémoire. La configuration de cette machine virtuelle est donnée par la figure 3.

La figure 4 détaille les différents composants du terminal et la façon dont ils sont reliés au calculateur central. Nous examinerons par la suite l'emploi des différents dispositifs pour l'édition.

Sur une machine virtuelle on peut faire fonctionner, entre autres, le système C.M.S dont chaque utilisateur possède une copie. L'éditeur n'a donc pas besoin d'être ré-entrant.

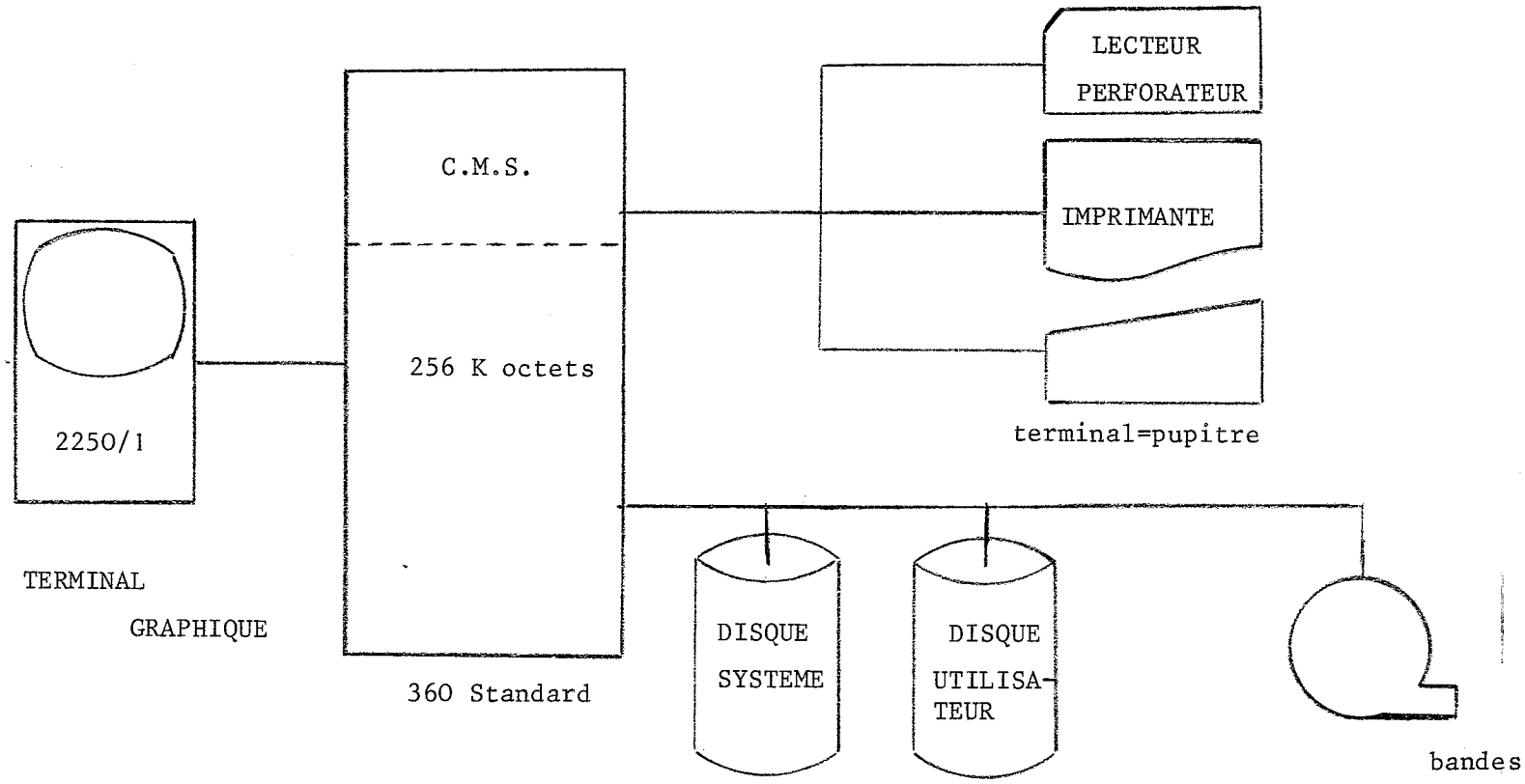



FIGURE 3 : CONFIGURATION D'UNE MACHINE VIRTUELLE

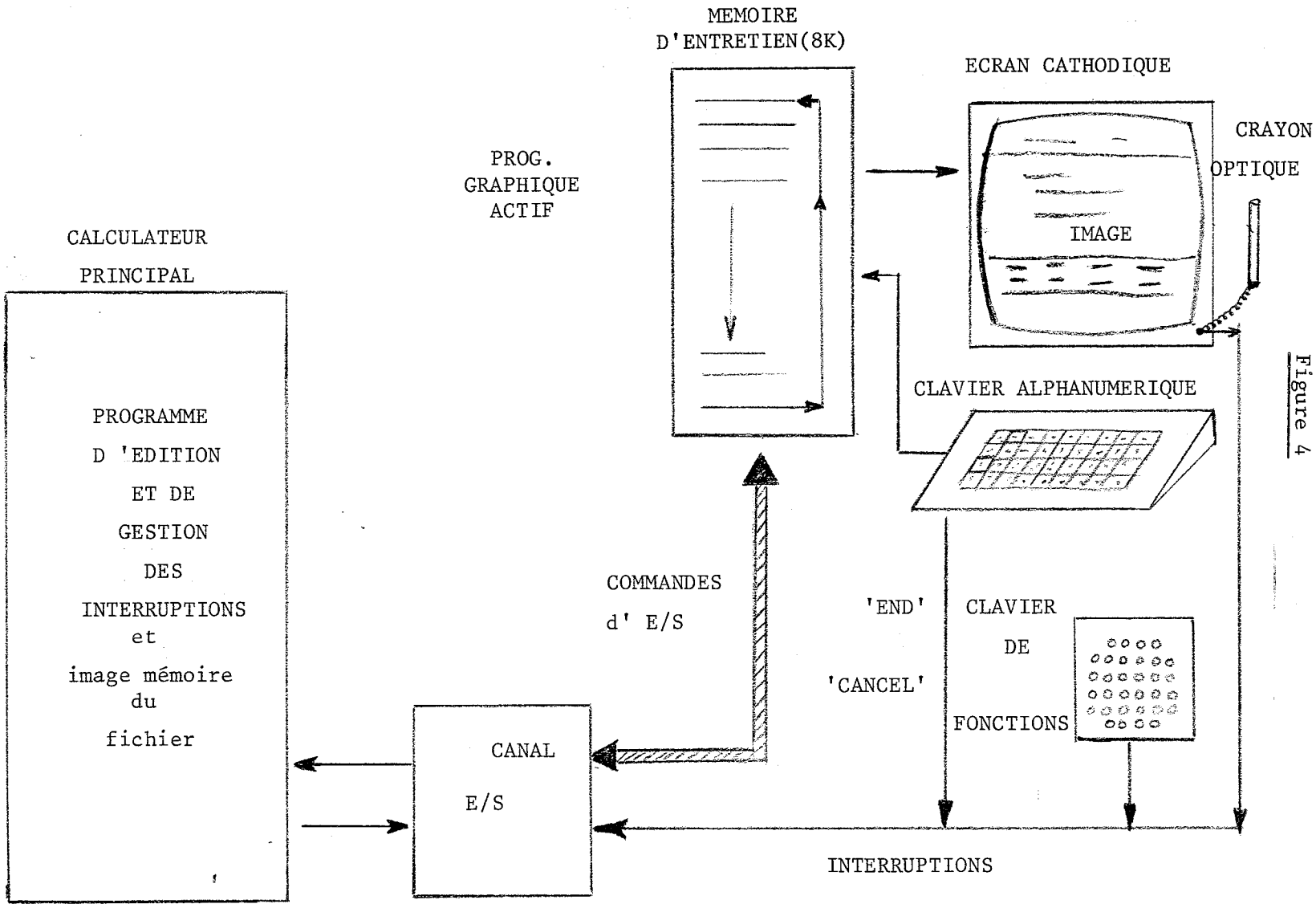

MEMOIRE
D'ENTRETIEN(8K)
ECRAN CATHODIQUE
CRAYON
OPTIQUE
PROG.
GRAPHIQUE
ACTIF
IMAGE
CALCULATEUR
PRINCIPAL
CLAVIER ALPHANUMERIQUE
PROGRAMME
D 'EDITION
ET DE
GESTION
DES
INTERRUPTIONS
et
image mémoire
du
fichier
COMMANDES
d' E/S
'END'
CLAVIER
DE
'CANCEL'
FONCTIONS
CANAL
E/S
INTERRUPTIONS

Figure 4

Pour les fonctions d'édition classiques nous avons conservé une partie de l'éditeur de CMS "EDIT". Ainsi, l'éditeur graphique est compatible avec la gestion de fichier de CMS et on retrouve presque toutes les requêtes d'EDIT avec, en plus, des fonctions propres à l'édition graphique.

Quand il utilise l'éditeur graphique, l'utilisateur a, à tous les instants, sur l'écran l'image de tout ou partie de son fichier ainsi que diverses indications sur les modes de fonctionnement de l'éditeur. Certaines touches du clavier de fonctions peuvent être allumées, ce qui signifie qu'elles sont utilisables.

La mise en fonction, sous C.M.S, de l'éditeur graphique se fait en tapant au terminal (pupitre) de la machine virtuelle (voir figure 3) la commande "GEDIT" suivie d'un nom et d'un type de fichier.

Comme sous EDIT, si le fichier spécifié n'existe pas encore, l'éditeur se mettra en mode "entrée de lignes" (Photo N°1) sinon il sera en mode "attente de requête" (Photo n°2) . Ces deux modes seront décrits plus loin.

## 2.2 PARTAGE DE L'ECRAN

L'écran dont la surface utile peut contenir 52 lignes de 74 caractères de petite taille ou 35 lignes de 49 caractères de grande taille, est divisé ici en 5 zones matéralisées par des traits horizontaux (Fig.5).

### 2.2.1 Zone requête::

La présence du "curseur" (cf.2.3) dans cette zone permet à l'utilisateur de savoir qu'il peut y composer ses requêtes. Celles-ci peuvent être transmises soit individuellement, soit en groupe si elles sont séparées par le caractère spécial "#" . Dans les deux cas la demande d'exécution se fait à l'aide de la touche "END" du clavier alphanumérique ou de la touche "EXEC" du clavier de fonction.

### 2.2.2 Zone message:

Chaque fois que l'éditeur a besoin d'envoyer un message il

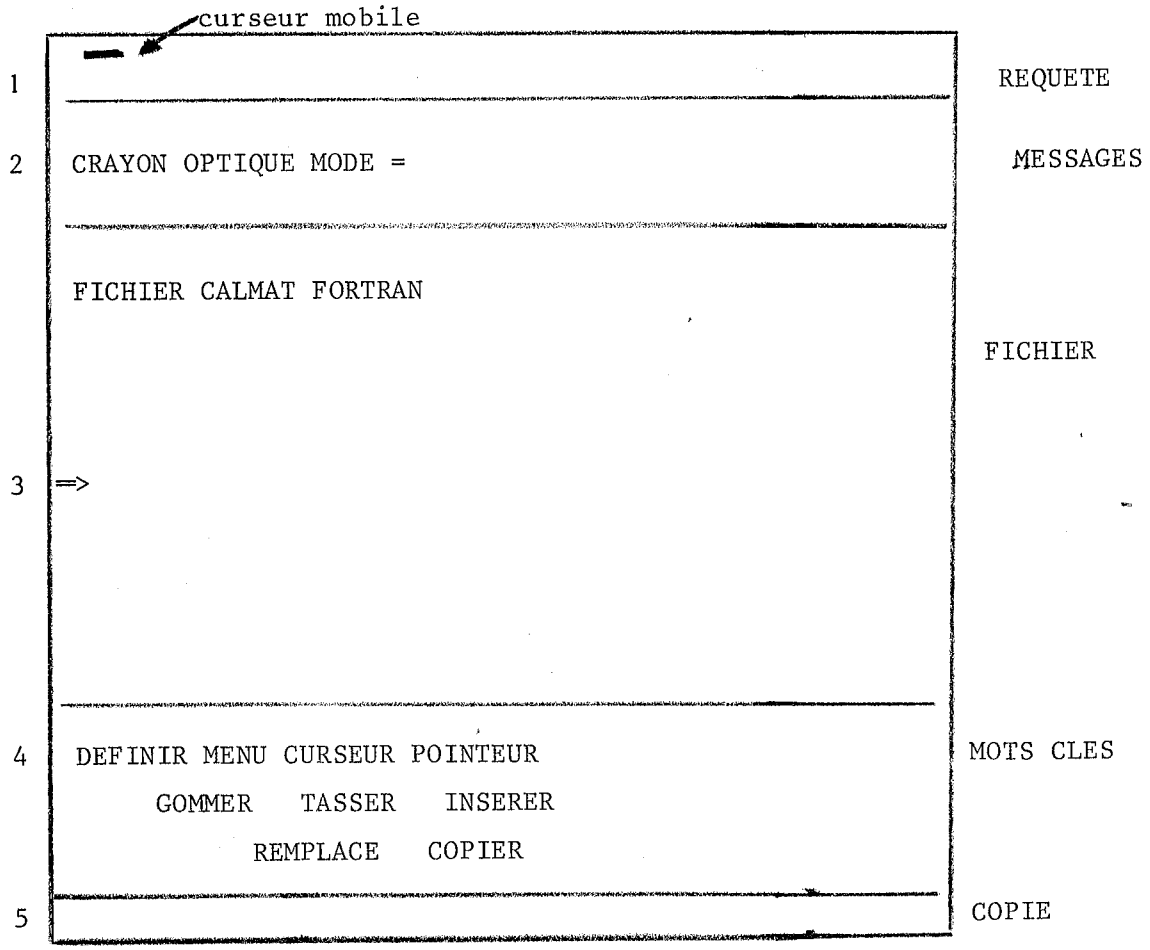
le fait apparaître dans cette zone et attire l'attention de l'utilisateur en actionnant le signal sonore du 2250. En outre, le mode courant de fonctionnement du crayon optique, choisi par l'utilisateur, y est inscrit en permanence.

### 2.2.3 Zone fichier:

En tête de cette zone se trouve l'identification (nom et type) du fichier en cours d'édition. La partie restante, qui constitue une sorte de fenêtre laisse apparaître 35 lignes (de 72 caractères) du fichier.

La ligne courante cadrée au milieu de cette fenêtre est signalée par le pointeur => qui est fixe sur l'image. Le cadrage automatique donne ainsi la plus large contexte possible: 17 lignes précédant et 17 lignes suivant la ligne courante.

Des requêtes font apparaître le curseur sous la ligne courante, autorisant ainsi l'utilisateur à la modifier à l'aide du clavier alphanumérique. De plus, grâce au crayon optique et aux fonctions associées, il est possible d'agir directement sur les lignes affichées.




ZONES DE PARTAGE DE L' ECRAN

FIGURE 5

```
NEW FILE.

CRAYON OPTIQUE :  MODE =

FICHIER  EXEMPLE FORTRAN

=>_

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER          TASSER          INSERER
      REMPLACE          COPIER          INVERMOT
         INVERLIN
```

1

```
CRAYON OPTIQUE :  MODE =

FICHIER  EXEMPLE FORTRAN

=>
      REAL X(100),Y(100),Z(400),DERX(400),V(100),DER(100),DERY(400),
     CRAC1(100),RAC2(100),RX(2),RY(2),N2(200),S,U(100),
     CX1,X2,F1,F2,D1,D2,U1,EQUI,RACY1,RACY2
      INTEGER N,M,M2,I,J,I1,J1,I11,J11,M1,M1,NB,L,N1,VM,VM
C
      INTEGER W,ENTIER
C
      DIMENSION IZ(1)
      LOGICAL BOOL
      CALL INITTA(IZ,B)
      CALL INITT(B,PS,1,2,1)
      CALL SOATL(P4,1,1,32,20)
C INITIALISATION : LECTURE DES DONNEES N,M,X,Y,ZOE,TAUX
C  SUR LE FICHIER FT03F001
      READ(3,1000) N,M
1000  FORMAT(2I3)
      M2=N*M
      READ (3,1001) (X(I),I=1,N)

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER          TASSER          INSERER
      REMPLACE          COPIER          INVERMOT
         INVERLIN
```

2

```
DELETE 3_

CRAYON OPTIQUE :  MODE =

FICHIER  EXEMPLE FORTRAN
 78     NZ((I-1)*M1+J)=VAL2SP(Y,U,V,VM,S,BOOL)
 73     CONTINUE
C  MIXAGE DES NZ ET DES DERY POUR OBTENIR Z
        L=1
        DO 75 I=1,M1
        DO 76 J=1,N
        Z(L)=DERY((I-1)*N+J)
 76     L=L+1
        DO 77 J=1,N
        Z(L)=NZ((J-1)*M1+I)
 77     L=L+1
 75     CONTINUE
        DO 79 I=1,M
        Z(L)=DERY(M1*N+I)
 79     L=L+1
        GO TO 2000
=>6000  WRITE(L,1002)
 1002   FORMAT(1X,'TROP X,Y OU X*Y',/)
 5000   WRITE(6,1003)
 1003   FORMAT(1X,'IMPOSSIBLE',/)
        RETURN
        END
        SUBROUTINE RAC3(X1,X2,F1,F2,D1,D2,NB,U1,EQUI)
        REAL X1,X2,F1,F2,D1,D2,U1,EQUI,S
        INTEGER NB,I,J
        REAL*8 M,A,B,C,E,AO,A1,A2,A3,P,Q,D,PH,EE(3),SS,R,RC
        M=X2-X1
        A=(F2-F1)/M
        B=(A-D1)/M
        C=(D1-A)/M
        E=(C-B)/M
        AO=E
        A1=B-E*(X2-2.000*X1)
        A2=D1-X1*(2.000*B-E*(X1+2.000*X2))
        A3=F1-X1*(D1-X1*(B-E*X2))-EQUI

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER          TASSER          INSERER
      REMPLACE          COPIER          INVERMOT
         INVERLIN
```

3

```
CRAYON OPTIQUE :  MODE =

FICHIER  EXEMPLE FORTRAN
 78     NZ((I-1)*M1+J)=VAL2SP(Y,U,V,VM,S,BOOL)
 73     CONTINUE
C  MIXAGE DES NZ ET DES DERY POUR OBTENIR Z
        L=1
        DO 75 I=1,M1
        DO 76 J=1,N
        Z(L)=DERY((I-1)*N+J)
 76     L=L+1
        DO 77 J=1,N
        Z(L)=NZ((J-1)*M1+I)
 77     L=L+1
 75     CONTINUE
        DO 79 I=1,M
        Z(L)=DERY(M1*N+I)
 79     L=L+1
        GO TO 2000
 1003   FORMAT(1X,'IMPOSSIBLE',/)
        RETURN
        END
        SUBROUTINE RAC3(X1,X2,F1,F2,D1,D2,NB,U1,EQUI)
        REAL X1,X2,F1,F2,D1,D2,U1,EQUI,S
        INTEGER NB,I,J
        REAL*8 M,A,B,C,E,AO,A1,A2,A3,P,Q,D,PH,EE(3),SS,R,RC
        M=X2-X1
        A=(F2-F1)/M
        B=(A-D1)/M
        C=(D1-A)/M
        E=(C-B)/M
        AO=E
        A1=B-E*(X2-2.000*X1)
        A2=D1-X1*(2.000*B-E*(X1+2.000*X2))
        A3=F1-X1*(D1-X1*(B-E*X2))-EQUI
        P=A2*AO-(A1*A1)/(3.000*AO*AO)
        Q=(2.000*A1*A1*A1-9.000*A1*A2*AO+27.000*A3*AO*AO)/27.000
       C/(AO*AO*AO)

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER          TASSER          INSERER
      REMPLACE          COPIER          INVERMOT
         INVERLIN
```

4

```
CRAYON OPTIQUE :  MODE =

FICHIER  EXEMPLE FORTRAN
 70     DO 72 I=1,M2
 72     DERY(I)=Z(I)
C  INITIALISATION DU CALCUL DE Z
        DO 73 I=1,N
        DO 74 J=1,VM
        U(J)=Z((J-1)*N+I)
 74     Y(J)=DERX((I-1)*VM+J)
        DO 78 J=1,M1
        S=(Y(J+1)-Y(J))/2.0
 78     NZ((I-1)*M1+J)=VAL2SP(Y,U,V,VM,S,BOOL)
 73     CONTINUE
C  MIXAGE DES NZ ET DES DERY POUR OBTENIR Z
        L=1
        DO 75 I=1,M1
        DO 76 J=1,N
        Z(L)=DERY((I-1)*N+J)
=>      TAUX(I)=_
 76     L=L+1
        DO 77 J=1,N
        Z(L)=NZ((J-1)*M1+I)
 77     L=L+1
 75     CONTINUE
        DO 79 I=1,M
        Z(L)=DERY(M1*N+I)
 79     L=L+1
        GO TO 2000
 1003   FORMAT(1X,'IMPOSSIBLE',/)
        RETURN
        END
        SUBROUTINE RAC3(X1,X2,F1,F2,D1,D2,NB,U1,EQUI)
        REAL X1,X2,F1,F2,D1,D2,U1,EQUI,S
        INTEGER NB,I,J
        REAL*8 M,A,B,C,E,AO,A1,A2,A3,P,Q,D,PH,EE(3),SS,R,RC
        M=X2-X1
        A=(F2-F1)/M

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER          TASSER          INSERER
      REMPLACE          COPIER          INVERMOT
         INVERLIN
```

5

MODES

DE FONCTIONNEMENT

EXEMPLES

D'EDITION GRAPHIQUE

```
CRAYON OPTIQUE :  MODE =  GOMMER

FICHIER  EXEMPLE FORTRAN
 1001  FORMAT(5E14.6)
       W1=0
 C  CALCULDES DIMENSIONS DU DOMAINE
       XMIN=X(1)-0.5
       YMIN=Y(1)-0.5
       IF (ABS(X(N)-X(1))-ABS(Y(M)-Y(1))) 1,1,2
 1     DMAX=ABS(X(N)-X(1))+0.5
       GO TO 3
 2     DMAX=ABS(Y(M)-Y(1))+0.5
 3     XMAX=X(1)+DMAX
       YMAX=Y(1)+DMAX
 C  DIMENSIONS DE F1
       CALL SDATL(F1,XMIN,YMIN,XMAX,YMAX)
 C  TRACE DU DOMAINE
       CALL STPOS(F1,XMAX,YMAX)
       CALL PLINE(F1,XMAX,YMIN)
=>     CALL STPOS(F1,XMAX,YMIN)
       CALL PLINE(F1,XMIN,YMIN)
       CALL EXEC(F1)
 C  AFFICHAGE DES POINTS DU DOMAINE
 4     CALL RESET(F5)
       DO 5 I=1,M
       DO 6 J=1,N
 6     CALL PTEXT(F5,'X',1,IZ,IZ,1,X(J),Y(I))
 5     CONTINUE
 C  RECHERCHE DU MAX ET MIN SUR LE DOMAINE
       MAX=-1.0E+60
       MIN=1.0E+60
       DO 7 I=1,M
       DO 8 J=1,N
       IF (MAX.GE.Z((I-1)*N+4))GO TO 9
       MAX=Z((I-1)*N+4)
       I1=I
       J1=J
 9     IF (MIN.LE.Z((I-1)*N+J)) GO TO 8

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER        TASSER          INSERER
      REMPLACE       COPIER          INVERMOT
         INVERLIN
```

6

```
CRAYON OPTIQUE :  MODE =  GOMMER

FICHIER  EXEMPLE FORTRAN
 1001  FORMAT(5E14.6)
       W1=0
 C  CALCULDES DIMENSIONS DU DOMAINE
       XMIN=X(1)-0.5
       YMIN=Y(1)-0.5
       IF (ABS(X(N)-X(1))-ABS(Y(M)-Y(1))) 1,1,2
 1     DMAX=ABS(X(N)-X(1))+0.5
       GO TO 3
 2     DMAX=ABS(Y(M)-Y(1))+0.5
 3     XMAX=X(1)+DMAX
       YMAX=Y(1)+DMAX
 C  DIMENSIONS DE F1
       CALL SDATL(F1,XMIN,YMIN,XMAX,YMAX)
 C  TRACE DU DOMAINE
       CALL STPOS(F1,XMAX,YMAX)
       CALL PLINE(F1,XMAX,YMIN)
=>     CALL STPOS(F1,XMAX,YMIN)
       CALL PLINE(F1,XMIN,YMIN)
       CALL EXEC(F1)
 C  AFFICHAGE DES POINTS
 4     CALL RESET(F5)
       DO 5 I=1,M
       DO 6 J=1,N
 6     CALL PTEXT(F5,'X',1,IZ,IZ,1,X(J),Y(I))
 5     CONTINUE
 C  RECHERCHE DU MAX ET MIN SUR LE DOMAINE
       MAX=-1.0E+60
       MIN=1.0E+60
       DO 7 I=1,M
       DO 8 J=1,N
       IF (MAX.GE.Z((I-1)*N+4))GO TO 9
       MAX=Z((I-1)*N+4)
       I1=I
       J1=J
 9     IF (MIN.LE.Z((I-1)*N+J)) GO TO 8

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER        TASSER          INSERER
      REMPLACE       COPIER          INVERMOT
         INVERLIN
```

7

```
CRAYON OPTIQUE :  MODE =  COPIER

FICHIER  EXEMPLE FORTRAN
       IF (BOOL) GO TO 11
       GO TO 5000
 11    DO 12 J=1,M
 12    DERX((I-1)*M+J)=DER(J)
 11    CONTINUE
 C  CALCUL POUR Y=CONSTANTE
       DO 13 I=1,M
       DO 14 J=1,N
 14    V(J)=Z((I-1)*N+J)
       CALL SPL2IN(X,V,N,DER,BOOL)
       IF (BOOL) GO TO 15
       GO TO 5000
 15    DO 18 J=1,N
 18    DERY((I-1)*N+4)=DER(J)
 13    CONTINUE
 C  DEFINITION DE QUELQUES CONSTANTES
=>     N1=N-1
       M1=M-1
       M2=M*N
 C  INITALISATION DES RACINES
       DO 19 I=1,100
 19    RAC2(I)=1.0E+60
       DO 20 I=1,N1
       CALL RAC3(X(I),X(I+1),Z(I),Z(I+1),DERY(I),DERY(I+1),NB,U1,EQUI)
       IF (NB-1) 21,22,23
 21    GO TO 20
 23    GO TO 55
 22    RAC2(I)=U1
 20    CONTINUE
 C  BOUCLE GENERALE SUR Y
       DO 24 I=1,M1
 C  CALCUL DE RAC1
       DO 25 J=1,N1
 25    RAC1(J)=RAC2(J)
 C  CALCUL DE RAC2

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER        TASSER          INSERER
      REMPLACE       COPIER          INVERMOT
         INVERLIN

POUR Y=CONSTANTE
```

8

```
CRAYON OPTIQUE :  MODE =  REMPLACE

FICHIER  EXEMPLE FORTRAN
       IF (BOOL) GO TO 11
       GO TO 5000
 11    DO 12 J=1,M
 12    DERX((I-1)*M+J)=DER(J)
 11    CONTINUE
 C  CALCUL POUR Y=CONSTANTE
       DO 13 I=1,M
       DO 14 J=1,N
 14    V(J)=Z((I-1)*N+J)
       CALL SPL2IN(X,V,N,DER,BOOL)
       IF (BOOL) GO TO 15
       GO TO 5000
 15    DO 18 J=1,N
 18    DERY((I-1)*N+4)=DER(J)
 13    CONTINUE
 C  DEFINITION DE QUELQUES CONSTANTES
=>     N1=N-1
       M1=M-1
       M2=M*N
 C  INITALISATION DES RACINES
       DO 19 I=1,100
 19    RAC2(I)=1.0E+60
       DO 20 I=1,N1
       CALL RAC3(X(I),X(I+1),Z(I),Z(I+1),DERY(I),DERY(I+1),NB,U1,EQUI)
       IF (NB-1) 21,22,23
 21    GO TO 20
 23    GO TO 55
 22    RAC2(I)=U1
 20    CONTINUE
 C  BOUCLE GENERALE POUR Y=CONSTANTE
       DO 24 I=1,M1
 C  CALCUL DE RAC1
       DO 25 J=1,N1
 25    RAC1(J)=RAC2(J)
 C  CALCUL DE RAC2

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER        TASSER          INSERER
      REMPLACE       COPIER          INVERMOT
         , INVERLIN

POUR Y=CONSTANTE
```

9

```
CRAYON OPTIQUE :  MODE =  COPIER    GOMMER    INSERER   POINTEUR

FICHIER  EXEMPLE FORTRAN

       REAL X(100),Y(100),Z(400),DERX(400),V(100),DER(100),DERY(400),
      CRAC1(100),RAC2(100),RX(2),RY(2),MZ(200),S,U(100),
      CK1,X2,F1,F2,D1,D2,U1,EQUI,RACY1,RACY2
       INTEGER N,M,M2,I,J,I1,J1,I11,J11,N1,M1,NB,L,W1,VN,VM
 C
       INTEGER W,ENTIER
 C
       DIMENSION IZ(1)
       LOGICAL BOOL
=>     CALL INITI(A,IZ,0)
       CALL INITI(0,F5,1,2,1)
       CALL SDATL(F4,1,1,92,20)
 C INITIALISATION : LECTURE DES DONNEES N,M,X,Y,ZOE,TAUX
 C  SUR LE FICHIER FT03F001
       READ(3,1000) N,M
 1000  FORMAT(2I3)
       M2=N*M
       READ (3,1001) (X(I),I=1,N)
       READ (3,1001) (Y(I),I=1,M)
       READ (3,1001) (Z(I),I=1,M2)
 1001  FORMAT(5E14.6)
       W1=0
 C  CALCULDES DIMENSIONS DU DOMAINE
       XMIN=X(1)-0.5
       YMIN=Y(1)-0.5
       IF (ABS(X(N)-X(1))-ABS(Y(M)-Y(1))) 1,1,2
 1     DMAX=ABS(X(N)-X(1))+0.5
       GO TO 3

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER        TASSER          INSERER
      REMPLACE       COPIER          INVERMOT
         INVERLIN

POUR Y=CONSTANTE
```

10

```
CRAYON OPTIQUE :  MODE =  COPIER    GOMMER    INSERER   POINTEUR

FICHIER  EXEMPLE FORTRAN

       REAL X(100),Y(100),Z(400),DERX(400),V(100),DER(100),DERY(400),
      CRAC1(100),RAC2(100),RX(2),RY(2),MZ(200),S,U(100),
      CK1,X2,F1,F2,D1,D2,U1,EQUI,RACY1,RACY2
       INTEGER N,M,M2,I,J,I1,J1,I11,J11,N1,M1,NB,L,W1,VN,VM
 C
=>     INTEGER W,X,Y,ZOE,TAUX,ENTIER
 C
       DIMENSION IZ(1)
       LOGICAL BOOL
       CALL INITI(A,IZ,0)
       CALL INITI(0,F5,1,2,1)
       CALL SDATL(F4,1,1,92,20)
 C INITIALISATION : LECTURE DES DONNEES N,M
 C  SUR LE FICHIER FT03F001
       READ(3,1000) N,M
 1000  FORMAT(2I3)
       M2=N*M
       READ (3,1001) (X(I),I=1,N)
       READ (3,1001) (Y(I),I=1,M)
       READ (3,1001) (Z(I),I=1,M2)
 1001  FORMAT(5E14.6)
       W1=0
 C  CALCULDES DIMENSIONS DU DOMAINE
       XMIN=X(1)-0.5

DEFINIR        MENU          CURSEUR        POINTEUR
   GOMMER        TASSER          INSERER
      REMPLACE       COPIER          INVERMOT
         INVERLIN

,X,Y,ZOE,TAUX
```

11

Nous examinerons plus tard le rôle des deux dernières zones: "Mots clés" et "copie" (cf. §3-2).

2.3 CLAVIER ALPHANUMERIQUE

La frappe d'une chaîne de caractères à l'aide du clavier alphanumérique est locale au 2250 mais ne peut se faire que sous deux conditions:

1- Un symbole spécial: le "curseur" doit apparaître sur l'écran à l'endroit où le caractère que l'on va frapper doit figurer.

2- La zone dans laquelle le curseur apparaît doit être non protégée au moyen d'ordres graphiques propres au 2250. Ainsi, lorsque le curseur apparaît dans la zone requête, on dispose d'une zone non protégée de 132 caractères, le reste de l'image étant protégé.
Les touches spéciales du clavier provoquent les déplacements du curseur (en avant, en arrière) sans modifier le texte déjà composé.

2.3.1. Mode: "attente de requête".

Sous ce mode, le curseur se trouve en tête de la zone requête. L'utilisateur peut y composer des requêtes d'édition analogues à celle d'EDIT de CMS(cf Chap I, §3). On obtient l'exécution d'une requête ainsi composée, grâce à la touche "END" du clavier alphanumérique. L'image s'en trouve modifiée en conséquence.

Ainsi sur les photos n°3 et 4 on peut suivre l'évolution de l'image après exécution de la requête de suppression: "DELETE 3".

2.3.2 Mode: "Entrée de lignes".

L'accès à ce mode est automatique si le fichier à éditer n'existe pas encore, sinon on y accède par la requête "INPUT".

Cette requête fait apparaître dans la partie fichier, une zone vide non protégée de 72 caractères, insérée après la ligne courante, et en tête de laquelle se trouve le curseur. L'utilisateur peut alors y composer sa ligne (photo N°5) puis enfoncer la touche "END" (ou "EXEC") pour commander l'enregistrement dans le fichier de la nouvelle ligne et acquerir à nouveau une zone vide non protégée, pour une éventuelle ligne suivante.

A ce moment là, l'utilisateur peut soit composer une nouvelle ligne et recommencer autant de fois qu'il le désire, soit sortir du mode en conservant ou en ignorant la dernière ligne composée.

## 2.4 CLAVIER DE TOUCHES DE FONCTIONS.

Les 32 touches de ce clavier sont munies d'indicateurs lumineux programmables, grâce auxquels la disponibilité de touches est rendue plus visible. Ces touches n'ont que le sens que veut bien leur donner le programme utilisant le 2250. Seul un numéro est envoyé au calculateur lorsqu'une touche est enfoncée.

Pour l'éditeur graphique ce clavier a été divisé en deux zones distinctes: (Fig.5)
- Zone système : 10 touches (N°22 à 31)
- Zone utilisateur: 22 touches (N° 0 à 21) dites touches "banales".

### 2.4.1 Touches "Système":

Elles ont un rôle bien précis, fixé par l'éditeur et en aucun cas l'utilisateur ne peut en modifier le sens, ce qui n'est pas le cas des touches banales. Les fonctions disponibles grâce à ces 10 touches système sont les suivantes :

- "EXEC" Execution de (ou des) requête(s) se trouvant dans la zone requête, et sous le mode "entrée de lignes", même effet que la touche 'END' du clavier alphanumérique.
- "ECOUTE" Sortie normale de tout mode particulier pour revenir à l'attente de requête.

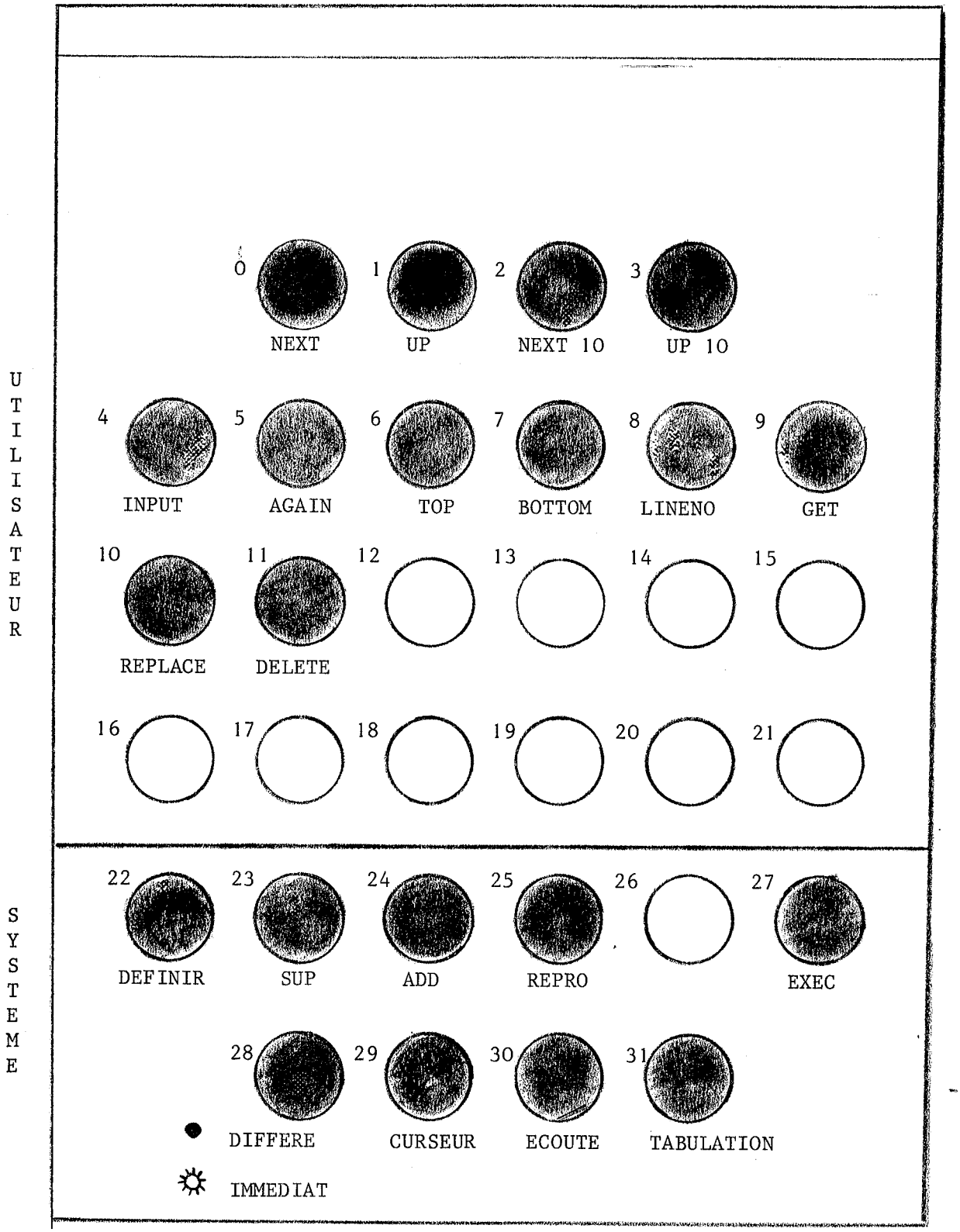

UTILISATEUR
0 NEXT
1 UP
2 NEXT 10
3 UP 10
4 INPUT
5 AGAIN
6 TOP
7 BOTTOM
8 LINENO
9 GET
10 REPLACE
11 DELETE
12
13
14
15
16
17
18
19
20
21
SYSTEME
22 DEFINIR
23 SUP
24 ADD
25 REPRO
26
27 EXEC
28 DIFFERE
29 CURSEUR
30 ECOUTE
31 TABULATION
IMMEDIAT

FIGURE 6

- "CURSEUR" provoque l'apparition du curseur sous le premier caractère de la ligne courante, permettant ainsi des modifications totales ou partielles sur celles-ci. L'abandon de ce mode "curseur" se fait par les touches "END", "EXEC" ou "ECOUTE" en conservant les modifications faites, ou par "CANCEL" en retrouvant la ligne courante, dans son état initial.

* Les 4 touches suivantes sont employées chaque fois quil s'agit de composer une ligne au clavier alphanumérique:

- "TABULATION" permet de respecter les colonnes de tabulation liées au type du fichier édité.

- "SUPPRIMER" Supprime le caractère situé au dessus du curseur en décalant vers la gauche le reste de la ligne.

*Exemple :*

Avant l'action de "supprimer": S U P P R I M M E R

Après: S U P P R I M E R

- "AJOUTER" Ajoute un"blanc" à l'emplacement désigné par le curseur, le reste de la ligne étant décalé vers la droite.

*Exemple :*

Avant : U N E X E M P L E

Après : U N _ E X E M P L E

- "REPRODUCTION" Reproduit, sur des intervalles correspondant aux tabulations, la ligne qui précède, et ce sous le mode "entrée de lignes" seulement.

* Les 2 touches suivantes sont liées à la définition et à l'utilisation des touches banales.

- "DEFINIR" Permet d'affecter à l'une des touches 0 à 21 une requête ou un groupe de requêtes.

Lorsque l'utilisateur appuie sur cette touche, la prochaine touche banale sur laquelle il agira se verra affecter la chaîne de caractères apparaissant à ce moment là dans la zone requête. Cette affectation sera confirmée par l'allumage de la touche banale correspondante.

Remarque :

Si la zone requête est vide alors la touche reste éteinte (ou s'éteint).

- "DIFFERE/IMMEDIAT" Cette touche conditionne le mode de fonctionnement des touches banales.

  Si elle est éteinte (mode différé) à chaque utilisation d'une touche banale, la chaîne de caractères associée apparaît dans la zone requête pour examen, exécution ("EXEC") ou modification ("DEFINIR").

  Si elle est allumée(mode immédiat), l'emploi d'une des touches banales allumées à ce moment là, provoque l'exécution de la requête associée.

  On bascule d'un mode à l'autre chaque fois que l'on appuie sur la touche "DIFFERE/IMMEDIAT".

### 2.4.2 Touches banales

L'éditeur graphique fournit, à la première utilisation, une configuration standard pour les touches banales qui correspond à un ensemble de requêtes de grande utilité. Après chaque édition, cette configuration est rangée dans un fichier ("PFK CLE2250") dont l'utilisateur peut, à sa guise, modifier le contenu par définition, re-définition (voir plus haut), ou même par édition.

Ainsi à chaque nouvel emploi de l'éditeur l'utilisateur retrouve :

- soit, son propre jeu de fonctions à partir du fichier PFK CLE2250
- soit une nouvelle configuration standard, s'il a supprimé ce fichier.

Nous avons vu (cf. 1.1) que plusieurs requêtes pouvaient être transmises en une seule fois à l'éditeur en utilisant le délimiteur " # ". De la même façon un groupe de requêtes peut être affecté par "définition" à une touche banale, de sorte qu'en mode immédiat, chaque fois que la touche correspondante est utilisée, il y ait exécution de la série de requêtes.

L'utilisateur a ainsi à sa disposition un mécanisme simple de "macro-requêtes".

## 3. EDITION AU CRAYON OPTIQUE.

### 3.1 FONCTIONNEMENT

Le crayon optique constitue le moyen le plus naturel pour désigner une partie du fichier que l'on désire modifier. Dans la technologie du 2250 une détection au crayon optique se fait sur un caractère (différent du "blanc") mais peut être interprétée de diverses manières:

En pointant un caractère on peut vouloir désigner :

- soit le caractère lui-même
- soit le mot qui contient ce caractère
- soit la ligne qui contient ce caractère
- soit le début d'une chaîne de caractères dont l'autre extrémité sera donnée par la prochaine détection.

Chacune de ces interprétations va correspondre à un mode de fonctionnement particulier pour le crayon optique.

### 3.2 ZONES DES MOTS-CLES

A chaque mode correspond un mot-clé dont la liste est affichée en permanence sur l'écran dans la zone MOT-CLES (voir photos et Fig.5)

Le choix du mode se fait en pointant le mot-clé correspondant qui apparaît alors dans la zone message.

*Exemple* (Photo n° 6) : Choix du mode "GOMMER".

Les détections au crayon optique dans la zone fichier sont alors interprétées de la même façon, jusqu'à ce que l'utilisateur change de mode...

La dernière zone en bas de l'écran est dite "zone de copie". Elle sert de zone intermédiaire pour les modes COPIER, REMPLACER et INSERER. C'est là que vient se ranger la copie de la chaîne pointée.(MODE COPIER voir plus loin).

### 3.3 FONCTIONS D'EDITION

- Le mode "POINTEUR" permet de rendre ligne courante la ligne contenant le caractère détecté.

- La mode "CURSEUR" provoque l'apparition du curseur sous le caractère détecté, afin de pouvoir modifier la ligne qui le contient (mêmes conventions que la touche "CURSEUR" cf.2-4)

- Sous le mode "GOMMER" il faut deux détections pour caractériser une chaîne dont tous les caractères sont remplacés par des blancs.(Photo n°7).

- Sous le mode "TASSER" deux détections sont également nécessaires.

*Exemple :*

Soit la ligne suivante : CECI EST UN PETIT EXEMPLE
et les détections : ↑1 ↑2

L'effet est le suivant CECI EST UN EXEMPLE

Il y a tassement d'une partie de la ligne (vers la gauche)

- Le mode "COPIER" permet de prendre une copie d'une chaîne de caractères comprise entre 2 détections. Cette chaîne apparaît dans la zone de copie (Photo 8).

- Grâce au mode "REMPLACER" on peut substituer à une chaîne, caractérisée par 2 détections, la chaîne se trouvant alors en zone de copie(Photo n° 9 )

- Quant au mode "INSERER" il permet d'insérer à gauche du caractère détecté, la chaîne se trouvant en zone de copie.

- Les modes "INVERMOT" et "INVERLIN" permettent respectivement d'inverser deux mots dans une même ligne ou deux lignes dans le fichier.

### 3.4 COMPOSITION DE FONCTIONS

Nous avons cherché, pour les fonctions décrites précédemment à construire des primitives pour l'édition au crayon optique, et nous offrons la possibilité de composer ces fonctions.

Il est nécessaire alors de définir 4 niveaux, chaque niveau contenant l'ensemble des fonctions incompatibles entre elles :

| Niveau | Fonction |
|---|---|
| 1 | COPIER |
| 2 | GOMMER TASSER |
| 3 | INSERER REMPLACER |
| 4 | CURSEUR POINTEUR |

Cet ordre a été choisi en fonction du déroulement logique des opérations. Les niveaux 1 et 2 ont des détections communes, 3 et 4 également.

Le "menu" correspond à une fonction composée. Il comporte de 1 à 4 fonctions choisies dans les différents niveaux.

Ce choix se fait après avoir pointé le mot-clé "DEFINIR" qui annule le mode de fonctionnement courant du crayon optique, en effaçant l'emplacement correspondant dans la zone message. L'utilisateur peut alors composer son menu en pointant les différentes fonctions qui l'intéressent. Si deux fonctions de même niveau ont été désignées, seule la dernière est prise en compte. L'éditeur re-arrange au fur et à mesure dans la zone message et dans l'ordre croissant des niveaux, les différentes fonctions choisies. La confirmation du choix se fait en pointant le mot clé "MENU".

Il ne reste plus alors qu'à faire en zone fichier les détections

pour obtenir l'effet composé des fonctions.

Le menu disparaît si l'on montre un autre mot-clé, mais peut être retrouvé par la suite en pointant à nouveau le mot-clé. "MENU".

*Exemple :* Photo N°10 avec le menu.

COPIER GOMMER INSERER POINTEUR

Trois détections au total:

2 pour COPIER et GOMMER
1 pour INSERER et POINTEUR

La photo n°11 montre l'état des lignes après les détections.

## 4. LOGIQUE INTERNE

La conception de l'éditeur graphique se heurte d'abord aux problèmes posés par le développement du dialogue entre l'homme et la machine.

Il fallait gérer tous les dispositifs (clavier alphanumérique, touches de fonctions- crayon optique) qui pouvaient générer, à tout instant, des interruptions. L'image présente sur l'écran devait refléter exactement l'état du fichier, au fur et à mesure de son évolution. Il a donc été nécessaire d'établir une hiérarchie d'interruptions et des modes de fonctionnement sous lesquels certaines interruptions étaient autorisées, d'autres interdites.

L'éditeur graphique a été programmé directement en langage d'assemblage (assembleur 360) et pour ce qui est des entrées-sorties nous avons écrit une série de programmes canaux. En effet, compte tenu du fait que nous employons le 2250 en grande partie comme console alphanumérique, nous n'avons pas voulu utiliser l'ensemble des sous-programmes graphiques fournis par IBM(GSP)- [G16] .

Ces sous-programmes étaient trop généraux et trop complets pour les opérations de manipulation de textes et pour la gestion d'interruption que nous avions à faire.

## 4.1 MODULARITE

Quatre modules composent l'éditeur graphique: (fig.7)

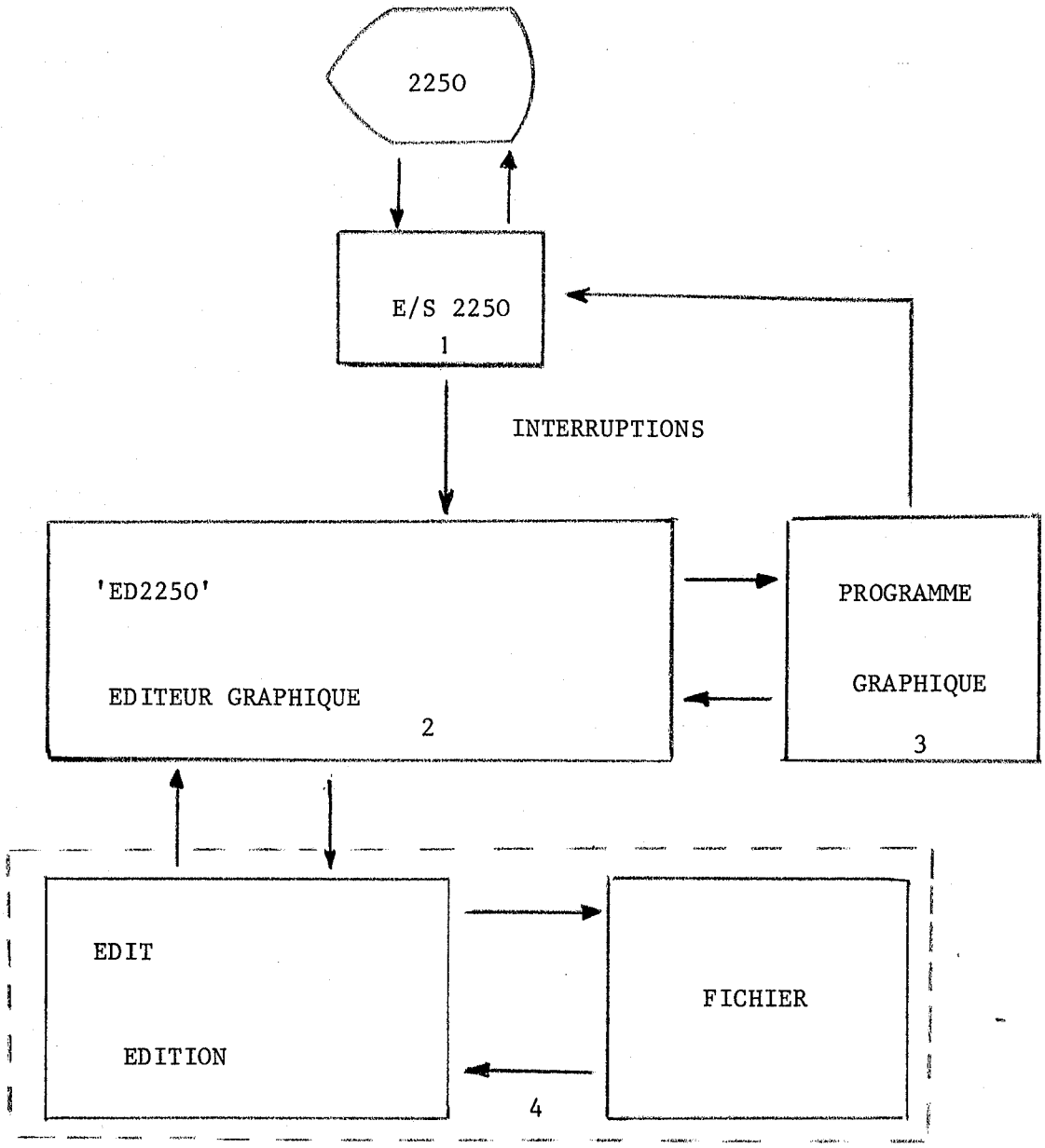



FIGURE 7

4.1.1 E/S 2250

Ce module, propre à CMS, est chargé de l'exécution des programmes canaux 2250 et de la gestion des interruptions y afférant. Il constitue un filtre entre le terminal et le programme utilisateur pour les interruptions asynchrones. Il mémorise toutes ces interruptions et en transmet une lorsqu'on lui signale que la précédente a été traitée.

4.1.2 Programme graphique

C'est le module qui contient les ordres graphiques générant le squelette de l'image apparaissant sur l'écran. D'une taille d'environ 4K octets, on y construit l'image courante avant d'envoyer le tout dans la mémoire d'entretien du 2250.

Les ordres graphiques permettent de réserver la place nécessaire aux caractères des lignes du fichier, aux mots-clés et aux messages, et provoquent le tracé des traits qui délimitent les différentes zones. D'autres ordres autorisent ou non la détection au crayon optique sur des parties d'image, d'autres encore définissent des zones non protégées où amener le curseur.

4.1.3 Edition

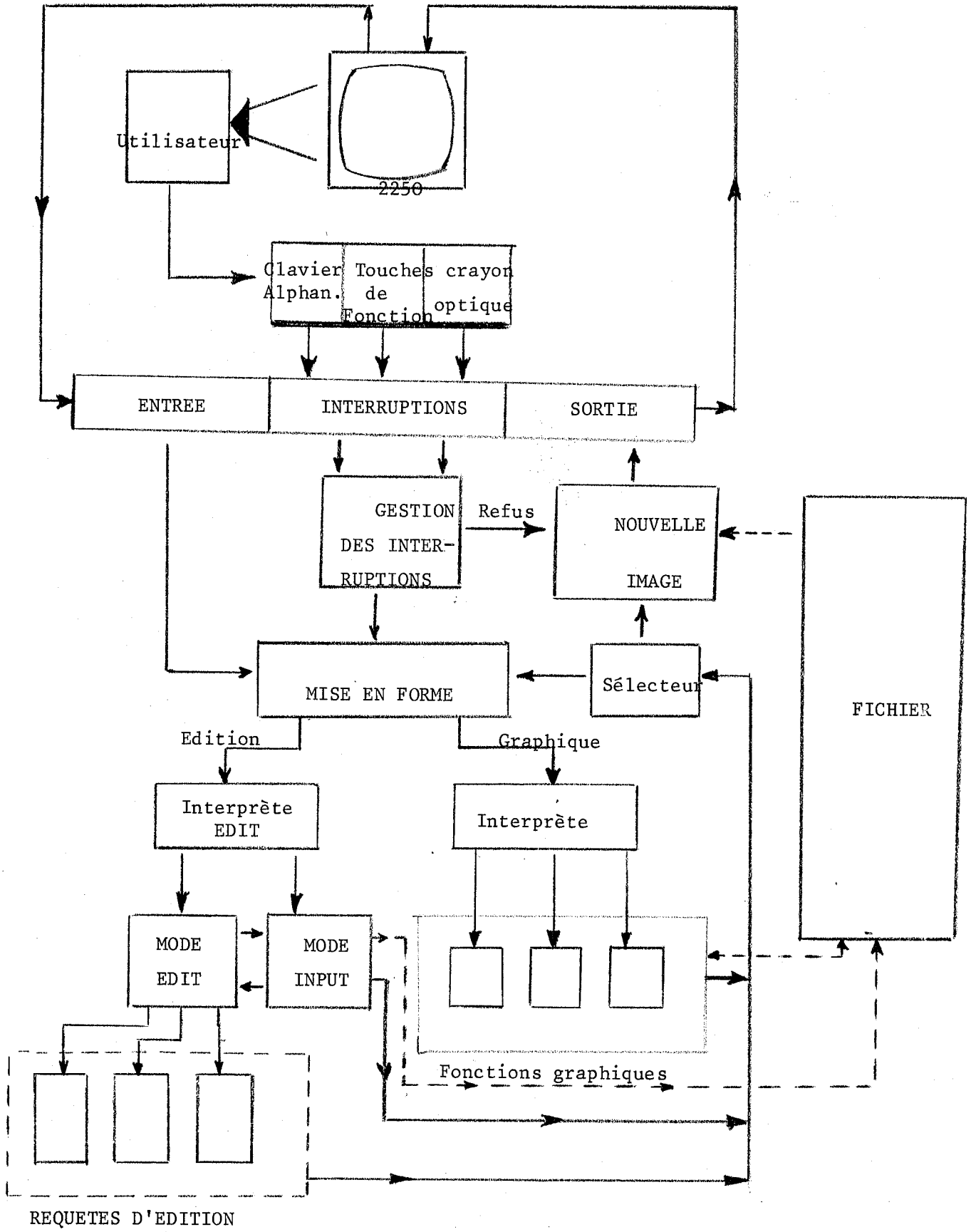
Le module d'édition correspond à une partie de l'éditeur de CMS "EDIT" modifié. C'est ce module qui s'occupe de la partie purement gestion de fichier et fonctions d'édition classiques.

4.1.4 Edition graphique

C'est le module le plus important. Il contrôle l'intéraction avec l'utilisateur et le fonctionnement global de l'éditeur en assurant la correspondance avec les autres modules. (EDIT en particulier).(Fig.8).

C'est ce module qui traite toutes les interruptions, qui gère les différentes zones de l'écran et construit une nouvelle image après chaque action de l'utilisateur.

Il réalise toutes les fonctions spéciales à la partie graphique de




LOGIQUE INTERNE FIGURE 8

l'édition, en particulier traitement des modes de fonctionnement, gestion des touches de fonctions ( système et banales) et surtout prise en charge des détections au crayon optique. En effet, l'utilisation de ce dernier correspond à un environnement particulier dans lequel toutes les fonctions sont réalisées, par l'éditeur graphique, directement sur l'image du fichier en mémoire.

Quand il reçoit le contrôle à partir du système C.M.S. le module d'édition graphique:

- Initialise le 2250 pour établir la correspondance entre l'unité physique et le programme de traitement (Appel à E/S 2250)
- Etablit la configuration de touches de fonctions en accord avec le contenu du fichier'PFK CLE2250' de l'utilisateur.
- Initialise le programme graphique avec les renseignements concernant le fichier à éditer et place les mots-clés du crayon optique.
- Appelle le module 'd'édition' qui détermine l'existence du fichier et réalise ou non l'installation des lignes en mémoire. Il rendra le contrôle à l'éditeur graphique qui génèrera la première image correspondante (cf.§2).

Après affichage de cette première image, l'éditeur graphique se met en attente d'interruptions.

Dès qu'une interruption se produit, il y a activation du mécanisme de traitement correspondant.

4.2 GESTION DES INTERRUPTIONS (Fig.9)

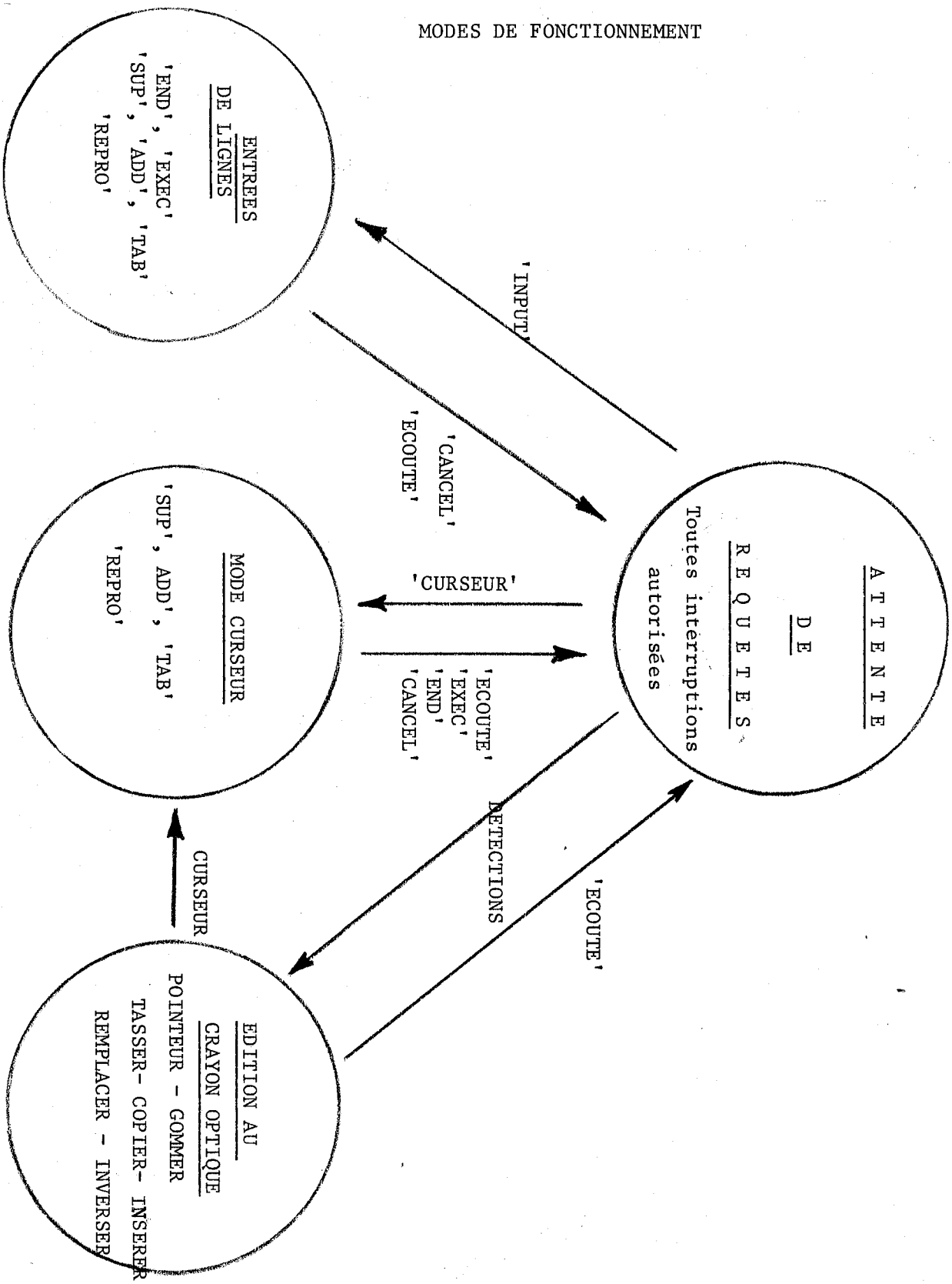
Elle se fait à deux niveaux par :

4.2.1 Un interprète général qui reçoit directement le contrôle du module 'E/S 2250'.

Il teste la validité de l'interruption en fonction de l'image affichée, et transmet le contrôle à l'interprète particulier du mode sous lequel se trouve l'éditeur graphique.

Figure 9

MODES DE FONCTIONNEMENT



4.2.2 La série des interprètes particuliers qui correspondent aux modes : "attente de requête", "entrée de lignes", "curseur" et "crayon optique" sous lesquels il faut sélectionner les interruptions permises. Lorsqu'une interruption est valide un sous-programme de traitement, spécifique au mode est appelé.

Les interruptions peuvent provenir :

- des touches "END" et "CANCEL" du clavier alphanumérique
- des touches du clavier de fonction , divisées en deux groupes : système et utilisateur.
- du crayon optique.

Il suffit donc pour chaque mode d'une table d'adresses à 4 entrées pour permettre (et traiter) ou ignorer l'un des types d'interruption.

Dans le cas des touches de fonctions, pour séparer entre elles les touches système et regrouper en un seul traitement les touches banales, une deuxième indexation (par leur numéro) est nécessaire.

## 4.3 TRAITEMENT DES DETECTIONS AU CRAYON OPTIQUE.

Les détections au crayon optique sont interdites en mode "entrée de lignes" et en mode "curseur". On ne peut montrer des caractères que dans la zone fichier ou dans la zone des mots-clés.

### 4.3.1 Détection en zone des mots-clés.

Elle permet le choix d'un mode de fonctionnement du crayon optique. Le mot-clé détecté est placé dans la zone message et on met certains indicateurs "en fonction" selon :

- le mode choisi
- le niveau de ce mode

Ces indicateurs sont destinés à l'interprète des fonctions graphiques qui prend le contrôle lorsqu'une autre détection se produit.

### 4.3.2 Détection en zone fichier

Chaque fonction d'édition: POINTEUR, CURSEUR, GOMMER etc... correspond à un sous-programme de l'interprète des fonctions graphiques.

Pour chaque niveau, l'examen des différents indicateurs provoque la recherche et l'exécution du sous-programme demandé.

Ainsi lorsqu'un menu a été choisi, les fonctions graphiques qui le composent s'exécutent les uns après les autres sous le contrôle de l'interprète.

REMARQUE

Il existe des sous-programmes communs à toutes les fonctions :

- Calcul du numéro de la ligne et du rang du caractère correspondant à une détection donnée.
- Mise en forme d'une première détection et attente d'une deuxième dans la même ligne. Celui-ci, doit gérer momentanément toutes les interruptions. Il ne laisse passer que les détections en zone de mots-clés, ou en zone fichier ainsi que l'interruption dûe à la touche "ECOUTE" qui permet de revenir à l'attente de requête en annulant toute détection antérieure.

# CHAPITRE III : MACRO-EDITEUR

## 1. NOTION DE MACRO-LANGAGE DE COMMANDE

## 2. DEFINITION D'UN MACRO LANGAGE D'EDITION

### 2.1 BUTS

### 2.2 MACRO-REQUETE

## 3. DESCRIPTION DU SYSTEME DE MACRO-REQUETES

### 3.1 PRINCIPAUX COMPOSANTS

### 3.2 REQUETES PARTICULIERES AU SYSTEME

#### 3.2.1 Gestion des macro-requêtes

#### 3.2.2 Macro-Langage

#### 3.2.3 Exemples

### 3.3 ORGANISATION EN MEMOIRE DES MACRO-REQUETES

#### 3.3.1 Gestion mémoire libre

#### 3.3.2 Tables des macro-requêtes

### 3.4 INTERPRETATION

## 4. EXEMPLES D'UTILISATION

# C H A P I T R E III

## MACRO-EDITEUR

### 1. NOTION DE MACRO-LANGAGE DE COMMANDE

Dans un système conversationnel, l'ensemble des commandes disponibles constitue un langage dont la syntaxe est toujours rudimentaire. Une commande valide se compose le plus souvent du nom identifiant la fonction demandée, suivi ou non, d'un certain nombre de paramètres. Parfois, l'emission d'une commande provoque le passage dans un environnement particulier(l'éditeur par exemple) où il faut utiliser un autre langage totalement différent du premier mais tout aussi rudimentaire.

La plupart des utilisateurs ont un travail spécialisé (par exemple, ils ne font que du FORTRAN) et se servent très souvent d'une même série de commandes. Il est alors agréable de grouper ces commandes pour constituer des procédures. Nous avons là une forme rudimentaire de "macro-langage" à savoir : grouper sous un même nom une série de commandes avec leurs paramètres, ce nom devenant pour le système celui d'une nouvelle commande.

Un tel langage peut s'étendre par :

- La possibilité d'imbrication: une macro-commande peut faire appel à d'autres macro-commandes.

- La présence de paramètres formels substitués, lors de l'appel à des paramètres effectifs.

- Des possibilités de tests sur la façon dont s'est déroulée une commande

- La présence de variables locales ou globales (numériques ou booléennes) intervenant également dans les tests.

- L'adjonction d'instructions de rupture de séquence et d'étiquettes

pour réaliser des boucles etc...

La commande "EXEC" du système C.M.S offre un macro-langage qui comporte toutes ces possibilités: [S5]

*Exemple :*

Edition, Compilation, Chargement, exécution d'un programme FORTRAN avec retour à l'éditeur s'il y a des erreurs de compilation:

```
-E  EDIT  &1  &2                &1 et &2 sont des paramètres formels
    &ERROR  &GOTO  -E                Teste une erreur sur la commande suivan-
                                     te  (-E est une étiquette)
    FORTRAN  &1                      Compilation
    LOAD &1                          Chargement
    START                            Exécution
```

## 2. DEFINITION D'UN MACRO-LANGAGE D'EDITION

### 2.1 BUTS

En définissant un macro-langage pour le sous-environnement conversationnel que constitue "EDIT" de CMS nos buts étaient les suivants :

- Etendre le langage de commande de l'éditeur en ajoutant de nouvelles requêtes d'intérêt général qui facilitent l'édition.

- Personnaliser ce langage en fournissant à chaque utilisateur les moyens de re-définir les requêtes du langage de base ou d'ajouter ses propres fonctions.

### 2.2 "MACRO-REQUETE"

Nous voulons donc fournir un mécanisme de définition, d'inter-

prétation et de conservation de nouvelles requêtes. Ces dernières, que nous appelons "macro-requêtes" deviennent des requêtes à part entière c'est-à-dire peuvent entrer dans la composition d'autres macro-requêtes.

Nous pouvons alors formaliser ce mécanisme :

< Requête > ::= < nom > < paramètres >
< nom > ::= < nom fonction base >|< nom macro-requête >
< nom fonction base > ::= DELETE | INSERT etc...
(imposés par l'éditeur)
< nom macro-requête >::= chaîne quelconque de 8 caractères au maximum
(choisi par l'utilisateur)

Une "macro-requête" se définit alors par :

< macro-requête > ::= < requête >|<macro-requête >< requête >

En plus d'un simple groupement de requêtes sous un nom collectif, nous voulons autoriser la présence de paramètres formels, aussi bien que d'instructions simples de rupture de séquence soit inconditionnelles soit conditionnées par la position du pointeur de ligne courante (extrêmités du fichier).

Pour illustrer le besoin d'un tel mécanisme considérons l'exemple suivant:

On veut supprimer dans un fichier toutes les lignes qui contiennent la chaîne "ABC". Cela se fait au moyen des requêtes:

| | |
|---|---|
| NEXT /ABC/ | Recherche d'une occurence |
| DELETE | Suppression de la ligne trouvée |

Or, si cette opération est à réaliser sur 50 lignes qui contiennent la chaîne ABC, il faudra taper 100 requêtes!

Par contre, si l'on peut grouper les deux requêtes précédentes sous un même nom, il suffira alors de taper ce nom pour obtenir la suppression d'une ligne puis d'émettre la requête "AGAIN 49" pour exécuter automatiquement les autres suppressions .

Nous verrons plus loin comment simplifier encore ce genre d'opérations grâce à d'autres propriétés du macro-langage.(§ 3-2)

## 3. DESCRIPTION DU SYSTEME DE MACRO-REQUETES

### 3.1 PRINCIPAUX COMPOSANTS

La définition d'une macro-requête, sa sauvegarde ou sa suppression supposent l'adjonction à EDIT de CMS d'un certain nombre de commandes supplémentaires.

Afin d'obtenir l'exécution rapide d'une macro-requête, il faut la conserver en mémoire, donc prévoir des structures adéquates pour les noms et les composants des macro-requêtes. Ceci implique la création d'un module de gestion dynamique de la mémoire libre.(cf.3.3.1)

L'interprétation doit tenir compte des imbrications possibles et gérer une ou plusieurs piles.

En outre, pour que la commande 'AGAIN' puisse s'appliquer aux macros-requêtes comme aux requêtes il est nécessaire de banaliser leur traitement.

### 3.2 REQUETES PROPRES AU SYSTEME

#### 3.2.1 Gestion des macro-requêtes

Définition d'une macro-requête: MDEF < nom > Cette requête (voir exemple plus loin) provoque, en cours d'édition, le passage dans un environnement de création où les lignes émises par l'utilisateur composent la macro-requête et sont rangées en mémoire. Une ligne vide signale la fin de la définition.

Deux macro-requêtes ne peuvent avoir le même nom, par contre si une macro-requête a le même nom qu'une fonction de base, c'est la macro qui a priorité.

Suppression, en cours d'édition, de la définition d'une macro qui se trouve à ce moment là en mémoire.(La place correspondante est libérée). MUNDEF < nom >

Sauvegarde sous forme de fichier (type "MACRO"), sur disque, d'une macro-requête: MSAVE < nom >

Chaque utilisateur peut ainsi conserver parmi ses fichiers toutes ses macro-requêtes.

Au cours d'une édition ultérieure, il pourra employer l'une de ses macro-requêtes déjà définies après avoir envoyé à l'éditeur la requête :

MDEF < nom > M

REMARQUES

- Puisqu'une macro-requête est un fichier comme un autre, on peut créer et modifier ce fichier par l'éditeur (voir exemple).
- Les fichiers de type "MACRO" sont des images de cartes 80 colonnes.

Impression au terminal de divers renseignements concernant les macro-requêtes disponibles pour l'édition en cours: MLIST. Liste des noms, des composants etc...

### 3.2.2. Macro-Langage

Une instruction de ce macro-langage peut se définir ainsi:

< instruction > ::= < nom > < paramètres > | < instruction de contrôle >

Le champ "< nom > " a le même sens qu'au § 2-2.

< paramètres >::= < paramètres effectifs > | < paramètres formels >

< paramètres effectifs > ::= nombre ou chaîne selon la requête

< paramètres formels > ::= &1 &2 | &1

< instruction de contrôle > ::= &EXIT < condition > | &GOTO < nombre > [ < condition > ]

< condition > ::= EOF / TOP

Seuls 2 paramètres formels sont autorisés. Lors de l'appel, l'absence de paramètre effectif entraîne le remplacement du paramètre formel correspondant par la chaîne vide.

Les instructions de contrôle ont utilisées pour tester la position du pointeur de ligne courante (du fichier) et modifier l'exécution séquentielle des macro-requêtes.

- &EXIT avec pour paramètre "TOP" ou "EOF" (end of file) permet d'abandonner l'exécution de la macro-requête en cours pour revenir au niveau d'appel, si le pointeur se trouve en haut (TOP) ou en bas du fichier(EOF).

- &GOTO n impose l'exécution de la $n^{ième}$ instruction composant la macro-requête.(Saut inconditionné).

- &GOTO n TOP ou EOF entraîne l'exécution de la $n^{ième}$ instruction seulement dans le cas où le pointeur ne se trouve pas en haut ou en bas du fichier.

3.2.3 Exemples:

- Nous allons modifier l'exemple du §2-2 pour qu'il soit plus général et pour qu'un simple appel provoque la recherche et la suppression sans savoir, à priori, combien de fois la chaîne donnée va être rencontrée:

* Macro-requête: "CHERSUP" : 1 paramètre: la chaîne à rechercher
* Composants: NEXT /&1/ Recherche (&1 paramètre formel)
  DELETE suppression
  &GOTO 1 EOF saut à la $1^{ère}$ instruction si on ne se trouve pas au bout du fichier
* Exemple d'appel: CHERSUP ABC

-Dans les pages suivantes,nous donnons des exemples d'utilisation des macro-requêtes avec l'éditeur de C.M.S.

- On trouvera en fin de chapitre des exemples de macro-requêtes.

```
edit gml sysin
EDIT:
```

Edition du fichier "GML SYSIN"

```
m
CORRECT FORM IS 'MDEF NAME '
OR : 'MDEF NAME M'
```

} Explications sur la syntaxe de "MDEF"

```
mdef mac1
MACRO-DEFINITION :
next /eject/
delete
&goto 1 eof

EDIT:
```

Définition de la macro-requête "MAC1"

} Requêtes composant la macro.

→ ligne vide : retour sous "EDIT".

```
msave mac1
REPLACE OLD MACRO FILE
SAVED AS : MAC1     MACRO
```

Sauvegarde de la macro-requête sous-forme de fichier, sur disque.

```
mlist mac1
MAC1
NEXT /EJECT/
DELETE
&GOTO 1 EOF
```

liste du contenu de la macro-requête

```
mac1
        EJECT
        EJECT
        EJECT
        EJECT
EOF:
EOF:
```

Exécution de la macro-requête.

```
mundef mac1
```

Suppression de la définition en mémoire

```
mac1
?
```

L'éditeur ne connait plus "MAC1"

```
mdef mac1 m
MDEF FROM FILE -OK
```

Re-définition à partir du fichier disque

```
ml mac1
MAC1
NEXT /EJECT/
DELETE
&GOTO 1 EOF
```

```
file

R; T=0.29/1.26 16.02.42
```

Sauvegarde du fichier "GML SYSIN" et fin d'édition

## Edition d'une macro-requête :

```
edit macl macro
EDIT:
next
NEXT /EJECT/
change /eject/&1/
NEXT /&1/
top
insert top
bottom
&GOTO 1 EOF
c /1/2/
&GOTO 2 EOF
file
R; T=0.12/0.69 16.03.54
```

&1 paramètre formel

Sauvegarde nouveau fichier

---

## Edition du fichier "GML SYSIN"

```
edit gml sysin
EDIT:
mdef macl m
MDEF FROM FILE -OK
mlist macl
MAC1
TOP
NEXT /&1/
DELETE
&GOTO 2 EOF
```

Definition de la nouvelle forme de la macro-requête.

La macro "MAC1" a un paramètre formel

```
macl eject
        EJECT
        EJECT
        EJECT
        EJECT
EOF:
EOF:
```

appel de "MAC1" avec un paramètre effectif.

execution ....

```
FILE: EDITH    EXEC    P1

&TYPEOUT OFF
&ERROR &GOTO -E
STATE PROF MACRO
ERASE PROF MACRO
-E LISTF * MACRO (EXEC
ALTER CMS EXEC P1 PROF MACRO P1
&BEGSTACK
BRIEF
MDEF FEPROF M
FEPROF
FILE
&ENDSTACK
EDIT PROF MACRO
&BEGSTACK
BRIEF
MDEF PROF M
PROF MDEF
MUNDEF PROF
MUNDEF FEPROF
VERIFY
&ENDSTACK
EDIT &1 &2
```

Utilisation de la commande "EXEC" pour disposer à chaque édition de toutes les macro-requêtes.

## 3.3 ORGANISATION DES MACRO-REQUETES EN MEMOIRE

### 3.3.1 Gestion de la mémoire libre

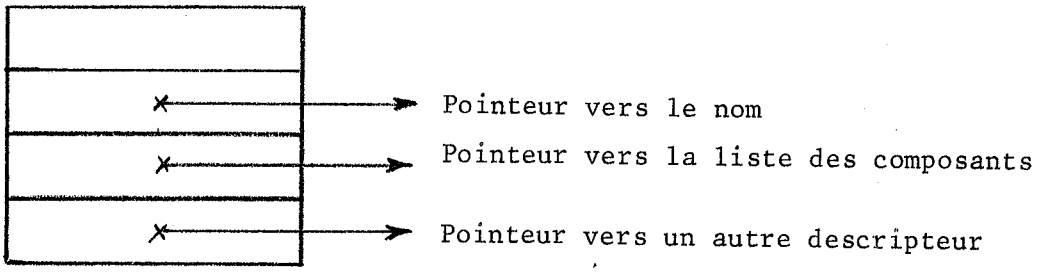
Les noms des macro-requêtes, les composants, les piles d'interprétation occupent une zone de mémoire gérée dynamiquement au moyen du module "GML", externe à l'éditeur. L'unité de mémoire est le double-mot (64 bits); toute demande ou libération de place se fait en multiple de double-mots.

Ce module est indépendant de l'éditeur, il gère un certain espace de mémoire et a été écrit dans un souci de simplicité, les programmes correspondant de C.M.S ne s'avérant pas efficaces.

### 3.3.2 Tables de macro-requêtes

Une macro-requête correspond en mémoire à trois éléments:

1- Son descripteur qui se compose d'une série de pointeurs




2- Son nom (1 double-mot : 8 caractères)

3- La liste des composants qui est une liste bi-directionnelle dont chaque élément a le format suivant :

| L | Suc. | Pred | ..... requête ..... |
|---|---|---|---|

"L" : longueur de la requête (en nombre d'octets)

"Suc" : Pointeur vers la requête suivante (0 si c'est la dernière)

"Pred": Pointeur vers la requête précédente (0 si c'est la première).

Les différents descripteurs sont chaînés entre eux pour constituer la table des macro-requêtes. Le dernier élément de cette table est chaîné à la table des fonctions de base. C'est une suite de double-mots contigus qui contient pour chaque fonction un pointeur vers son nom et un pointeur vers le sous-programme de traitement correspondant. Le pointeur "TABREQ' repère le premier descripteur et la fig.9 montre la configuration de la mémoire après définition de deux macro-requêtes:

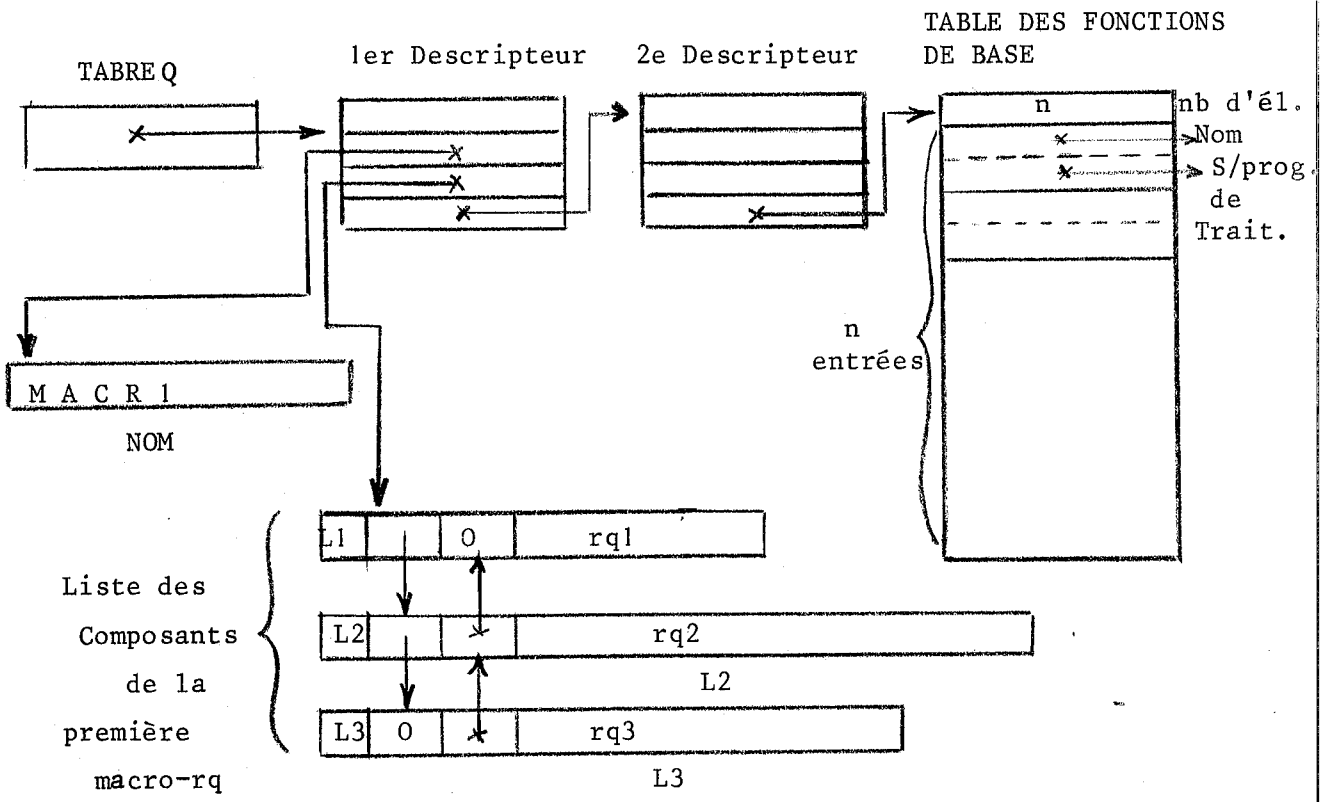



FIGURE 9

3.4 INTERPRETATION (Voir organigramme fig.10)

Elle est rendue difficile d'une part par le fait qu'une macro-requête peut en appeler d'autres (avec transfert de paramètres) et d'autre part, par l'existence de la requête "AGAIN n" qui permet de répéter n fois la dernière action de l'utilisateur (macro-requête ou requête). "AGAIN" peut intervenir également dans une macro-requête, à n'importe quel niveau, c'est pourquoi son traitement est particulier. Il est nécessaire notamment d'empiler le paramètre n pour chaque niveau de macro-requête.

L'interprétation générale se fait au moyen de deux piles:

- 1 pile d'interprétation proprement dite, contenant à chaque instant l'adresse du prochain composant à exécuter.

- 1 pile de pointeurs vers des zones de mémoire acquises dynamiquement et contenant l'information qui correspond à un niveau de macro-requête: Nom- paramètres- valeur du compteur de "AGAIN" etc...

La ligne émise par l'utilisateur est rangée dans un "buffer" d'entrée. Elle est de la forme :

< Nom (8 caractères au plus)> < paramètres >

Après isolement du nom il importe de savoir s'il correspond à une macro-requête ou a une fonction de base. Dans ce dernier cas, le contrôle est donné au sous-programme de traitement qui renvoie à l'ordre de lecture au terminal.

S'il s'agit d'une macro-requête le processus d'interprétation se déroule de la manière suivante (voir fig.10)

- Le descripteur de la macro fournit l'adresse de la suite de composants à interpréter. Cette adresse est placée au sommet de la pile d'interprétation.

- Le nom de la macro, et ses paramètres sont rangés dans une zone de mémoire; un pointeur vers cette zone est conservé dans la seconde pile.

- L'aiguillage en tête de l'interprète est modifié pour éviter la lecture au terminal et pour forcer l'exploitation de la pile d'interprétation: On recherche en mémoire un des composants de la macro-requête pour le mettre dans le buffer d'entrée après substitution des paramètres. On empile le successeur et on retourne à la partie de l'interprète qui analyse le buffer d'entrée.

- L'existence d'un élément nul au sommet de la pile signifie que le niveau de macro en cours est épuisé. Il faut alors restaurer les informations correspondant au niveau d'appel inférieur avant de poursuivre l'interprétation.

- Lorsque la pile est vide, c'est-à-dire lorsque toute la macro-requête (donc tous ses composants) a été traitée, il faut se replacer dans l'environnement attente lecture.

Les requêtes &EXIT et &GOTO se réduisent à une simple modification du sommet de la pile d'interprétation:

- Pour &EXIT, on force le passage au niveau inférieur après test du pointeur de ligne courante alors que pour &GOTO on utilise le double chaînage des composants des macro. pour mettre en sommet de pile le successeur demandé.

FIGURE 10 MACRO EDITEUR

INTERPRETE DE MACRO-REQUETES

(1ère Partie)

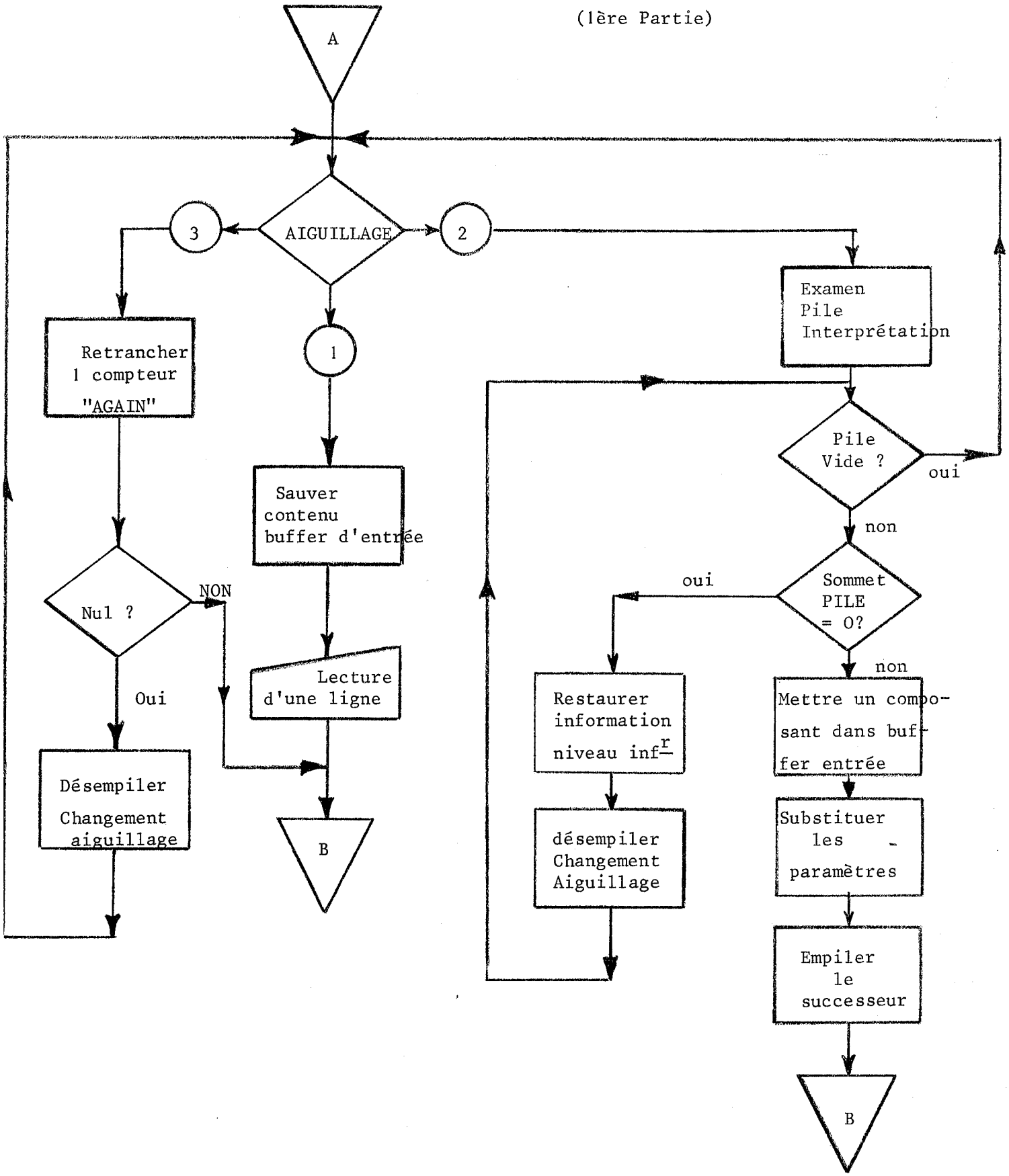
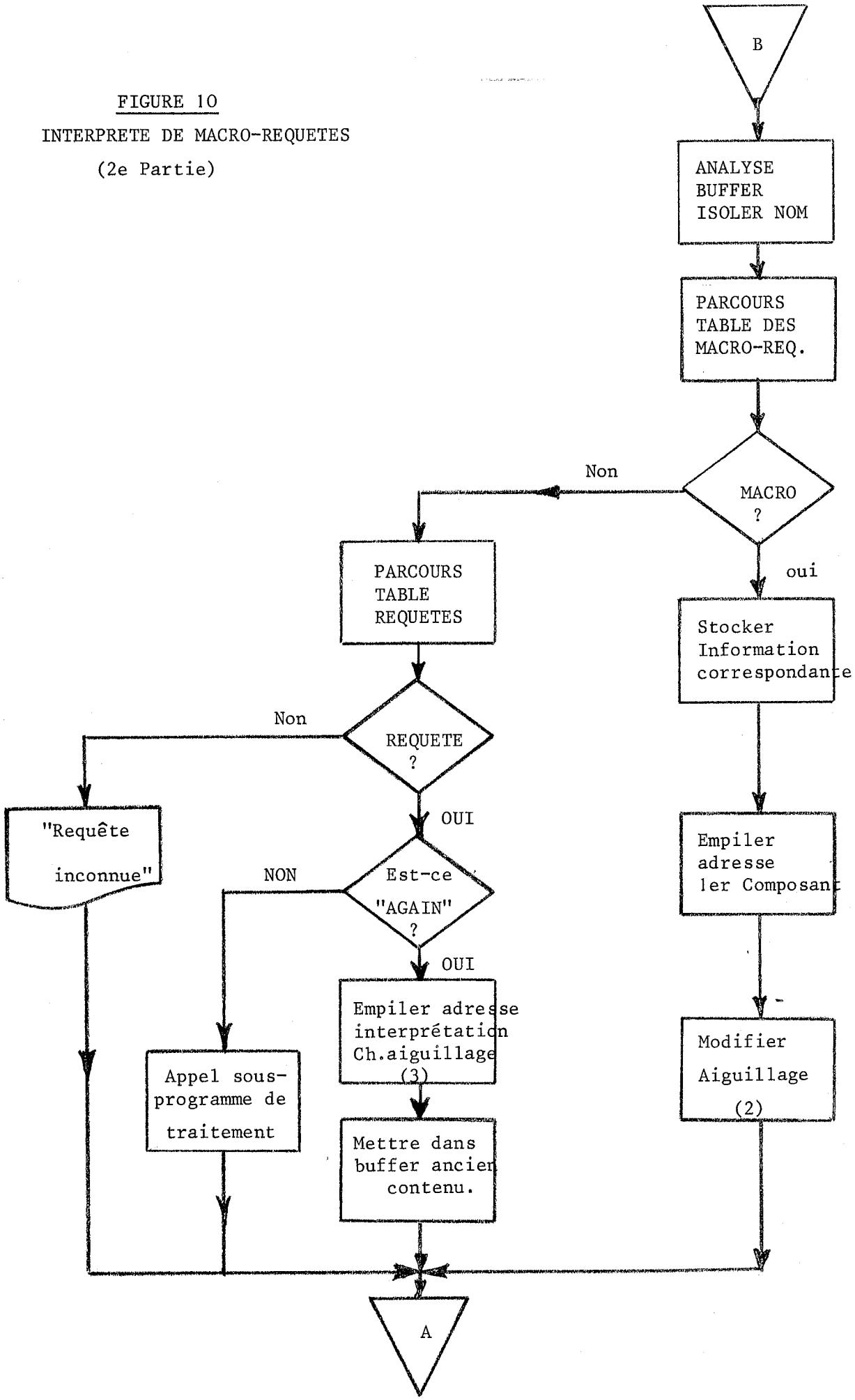


FIGURE 10

INTERPRETE DE MACRO-REQUETES

(2e Partie)



## §4 EXEMPLES DE MACRO-REQUÊTES :

FILE: FEPROF MACRO P1 CAMBRIDGE MONITOR SYSTEM

```
TOP
TABSET 1 18
BLANK>--------------------------
TABSET
CHANGE /&2 //
NEXT
&GOTO 2 EOF
TOP
```

[Transformation de fichier]

FILE: SOS MACRO P1 CAMBRIDGE MONITOR SYSTEM

```
INSERT ZZZZZZZZ
UP. &1
PUTD /ZZZZZZZZ/ ZZZZ MACRO
DELETE
MDEF ZZZZ M
ERASE ZZZZ MACRO
ZZZZ
MUNDEF ZZZZ
```

[Extraction et exécution de requêtes tapées par erreur dans un fichier.]

FILE: AMEN MACRO P1 CAMBRIDGE MONITOR SYSTEM

```
INSERT ZZZZZZZZ
GOTO. &1
PUTD &2
TOP
FIND. ZZZZZZZZ
DELETE
UP.
GET
```

[Insertion de '&2' lignes prises à la ligne '&I' ,après la ligne courante .]

FILE: SUPSER MACRO P1 CAMBRIDGE MONITOR SYSTEM

```
BRIEF
TOP
SERIAL (NO)
ZONE 73 80
CHANGE /&1/          / * *
ZONE
TOP
VERIFY
```

[Suppression de la sérialisation d'un fichier .]

FILE: TRANSFIN MACRO P1 CAMBRIDGE MONITOR SYSTEM

```
TOP
NEXT /&1/
&GOTO 7 EOF
PUTD 1 BIDON
UP.
&GOTO 2
GET BIDON 1 10000
ERASE BIDON
```

[Transport en fin de fichier des lignes ayant une chaîne commune .]

FILE: CHERCHE MACRO P1 CAMBRIDGE MONITOR SYSTEM

**AUTRES EXEMPLES :**

```
NEXT /&1/
&GOTO 4 EOF
QUIT
UP /USER/
PUT
UP
BLANK          MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
CHANGE //OVER /
ERASE BIDON MACRO
PUT 1 BIDON MACRO
NEXT /&1/
MDEF BIDON M
BIDON
MUNDEF BIDON
PUTD 1 C&1 SYSIN
UP /USER/
DELETE
UP
GET
&GOTO 1
```

FILE: SELEC MACRO P1 CAMBRIDGE MONITOR SYSTEM

```
SERIAL (NO)
BRIEF
TOP
NEXT /OWN/
DELETE
UP
NEXT /RDONLY/
DELETE
&GOTO 6 EOF
TOP
CHERCHE &1
```

FILE: CHERCHER MACRO P1 CAMBRIDGE MONITOR SYSTEM

```
BRIEF
NEXT /USER/
&GOTO 5 EOF
QUIT
PUT 1 DIRAL SYSIN
&GOTO 2
```

FILE: MISAJOUR EXEC P1

```
OFFLINE READ MODIF MACRO
&BEGSTACK
MDEF MODIF M
MODIF
FILE
&ENSTACK
EDIT &1 &2
```

[ Mise à jour automatique
au moyen fichier de requêtes ]

# CHAPITRE IV : BILAN DES REALISATIONS

## 1. BASES DE TRAVAIL

## 2. EDITION GRAPHIQUE

### 2.1 CONSOLES DE VISUALISATION ALPHANUMERIQUES

### 2.2 INTERACTION HOMME-MACHINE

### 2.3 CONCLUSIONS

## 3. MACRO-LANGAGE D'EDITION

### 3.1 CONTRAINTES

### 3.2 APPLICATIONS

### 3.3 INTERPRETATION

## 4. PROBLEMES GENERAUX

### 4.1 LIAISONS EDITEUR-SYSTEME

### 4.2 CHOIX DU TERMINAL

### 4.3 GESTION DU FICHIER A EDITER

### 4.4 LANGAGE D'EDITION

### 4.5 UTILISATION GENERALE

# C H A P I T R E IV

## BILAN DES REALISATIONS PRECEDENTES

### 1. BASES DE TRAVAIL

On peut remarquer tout d'abord que les deux réalisations précédentes -éditeur graphique et macro-éditeur- ont comme point commun l'éditeur du système C.M.S 'EDIT' et, qu'elles ont permis, de ce fait, de bien connaître ce programme pour en exploiter toutes les possibilités.

La mise au point des programmes sur une machine virtuelle a été grandement facilitée par l'emploi du système C.M.S (Assembleur- éditeur- chargeur- Gestion de fichiers-commandes 'EXEC' et 'DEBUG' etc...)

Dans cette première partie un certain nombre de problèmes déjà résolus par 'EDIT' 'n'ont pas été abordés. Il s'agit,par exemple, de la gestion du fichier à éditer, sur disque et en mémoire, et des opérations de base d'édition. D'une part, cela signifie qu'un utilisateur du système C.M.S connaissant "EDIT' peut employer facilement l'éditeur graphique ou le macro-editeur mais que d'autre part, les limitations de taille des fichiers ou de quelques opérations d'édition, demeurent.

Avec l'éditeur graphique nous voulions mesurer l'apport d'un terminal graphique dans l'édition et développer la souplesse dans la manipulation et la mise à jour des fichiers.

Avec le macro-éditeur nous voulions étudier un mécanisme d'extension et d'interprétation du langage de commande pour dégager un certain nombre de fonctions de base de tout éditeur.

## 2. EDITION GRAPHIQUE

### 2.1 CONSOLES DE VISUALISATION ALPHANUMERIQUE

L'utilisation d'un éditeur à partir d'un terminal classique, de type machine à écrire, ne permet d'avoir qu'une vue fragmentaire de son fichier. En effet, même l'impression de quelques lignes est très lente(15 caractères/ s) et retarde considérablement les actions de l'utilisateur. Par contre, l'emploi d'un terminal comportant un écran cathodique, auquel sont associés un clavier alphanumérique et un générateur de caractères, facilite grandement le travail de l'utilisateur, qui peut alors faire apparaître les parties successives de son fichier et modifier directement les lignes présentes sur l'écran.

Il existe un grand nombre de consoles de visualisation alphanumériques [G4- G17] ; elles diffèrent par :

- les types de liaison au calculateur central

- la taille de l'écran et le nombre de caractères qu'il peut contenir (1000- 2000 ou plus).

- La présence de fonctions d'édition spéciales (touches pour supprimer, insérer, une ligne, effacer l'écran etc...)

- La composition du clavier alphanumérique qui, outre les touches correspondant à un jeu de caractères standard, comporte des touches permettant soit de déplacer sur l'écran un "curseur" qui marque l'emplacement du prochain caractère soit de faire apparaître des symboles spéciaux (fin de ligne, fin de message). En général le fichier à éditer est découpé en pages dont la longueur est fonction du nombre maximum de lignes que peut contenir l'écran. On fait apparaître les pages successives, et, grâce aux touches du clavier on réalise des modifications locales. Si le terminal ne dispose pas de touches de fonctions spéciales, pour pouvoir supprimer ou insérer une ligne, il faut émettre une requête, et donc prévoir sur l'écran une zone particulière où l'on pourra composer cette requête.

Si l'écran est suffisamment grand on peut même prévoir une zone dans laquelle apparaitront des messages (erreurs- explications- etc...)

L'unité IBM 2250-1 est un terminal graphique très évolué qui comporte un clavier alphanumérique et un générateur de caractères. Néanmoins les autres dispositifs de ce terminal peuvent être utilisés avec profit dans l'édition.

## 2.2 INTERACTION

Les terminaux graphiques sont actuellement en pleine expansion car ils contribuent pour une large part au développement de l'intéraction homme-machine (G6-G7-G8).

Avec le 2250 nous voulions employer les touches de fonction et le crayon optique de manière naturelle pour l'utilisateur. Ceci nous conduit à imposer quelques redondances pour éviter de passer continuellement d'un dispositif à un autre.

C'est ainsi qu'un utilisateur qui doit exécuter très souvent la même série de requêtes a intérêt à se définir la configuration correspondante aux touches de fonction et à ne plus employer alors, ni le clavier alphanumérique, ni le crayon optique.

Le crayon optique permet de modifier directement les lignes présentes sur l'écran sans se préoccuper de la ligne courante. Il est alors d'une grande utilité dans le cas où l'on édite des textes destinés à l'impression. Sans ce domaine l'emploi du 2250 pour contrôler ou modifier la mise en page peut donner lieu à d'intéressantes études.

## 2.3 CONCLUSIONS

Certaines carences d'EDIT ont pu être mises en évidence par l'éditeur graphique: En particulier, l'absence d'un pointeur de caractères qui peut être déplacé à volonté dans une ligne et qui fixe un point de départ

pour des modifications.

Ceci provient de l'utilisation du crayon optique et du "curseur".

La gestion des touches banales auxquelles peuvent être affectée une série de requêtes est l'ébauche d'un mécanisme de macro-requêtes.

Les recherches d'opérations à réaliser au crayon optique ont conduit à la définition d'une série de fonctions de base, pouvant être composées pour donner des fonctions plus complexes.

On peut dire, enfin, qu'un éditeur graphique fonctionnant à partir d'une console alphanumérique munie d'un crayon optique, constitue un moyen idéal pour réaliser une véritable interaction "homme-fichiers".

## 3. MACRO-EDITEUR

### 3.1 CONTRAINTES

Alors que, pour l'éditeur graphique, il s'agissait de changer le terminal de communication -donc de modifier les opérations d'entrées-sorties d'EDIT, d'écrire un programme de gestion du 2250 et des modes d'édition graphique-, pour le macro éditeur il fallait ajouter à EDIT une série de requêtes et changer totalement le mécanisme d'interprétation.

Nous voulions en fait fournir une nouvelle version d'EDIT qui permettrait de définir et de gérer les macro-requêtes. La compatibilité devait être totale avec CMS, avec l'ancien éditeur et avec le macro-langage de CMS ('EXEC').

Cependant comme EDIT correspond à un programme écrit en langage d'assemblage, d'environ 3000 instructions sa modification fût lente et difficile.

### 3.2 APPLICATIONS

Le mode conversationnel est d'un emploi très agréable mais

on éprouve le besoin d'obtenir l'exécution automatique d'une série de commandes ou de requêtes. L'exemple du chap.III-2-2 est, à ce titre, très significatif.

Grâce au macro-éditeur il est possible:

- d'éviter les frappes successives de mêmes séries de requêtes accompagnées de paramètres plus ou moins compliqués.

- De définir de nouvelles opérations d'édition, même très complexes en exploitant toutes les possibilités du macro-langage: imbrication (limitée à 20 niveaux) -tests- rupture de séquence- passage de paramètres.

- De re-définir, partiellement ou complètement, le langage de base si l'on veut changer l'effet de certaines requêtes sans changer leurs noms.

- De préparer une série de modifications, et d'en garder une trace sous forme de fichier de macro-requêtes.

Cela est très utile lorque l'on change une série de programmes, déjà au point, mais auxquels on veut ajouter de nouvelles possibilités. On peut réaliser alors une sorte de mise à jour automatique, comme il en existe dans presque tous les systèmes ('update'). Cette mise à jour automatique a ici les particularités d'être "par contexte" et compatible avec une utilisation conversationnelle.

### 3.3 INTERPRETATION

La complexité du macro langage influence celle du mécanisme d'interprétation des requêtes. Dans tous les cas, il est nécessaire que chaque requête de base soit un sous-programme de l'interprète. Tous ces sous-programmes se partagent alors une zone de constantes.

Si l'éditeur n'admet que l'empilage de commandes, une simple zone de stockage suffit. Elle est examinée après exécution de chaque requête, pour savoir quelle est la prochaine commande à traiter. Le mécanisme d'interprétation décrit par le macro-éditeur, est plus complexe à cause de la conservation des requêtes, sous un nom collectif, et à cause de leur imbrication possible,

limitée à 20 niveaux dans ce cas.

La conservation en mémoire des macros-requêtes, permet d'éviter des opérations d'entrée/sortie, couteuses en temps, mais n'est possible que si la zone mémoire réservée à l'édition, est assez grande ou gérée dynamiquement.

Comme on le verra par la suite, l'édition sur une zone mémoire limitée entraîne le découpage de l'éditeur en un certain nombre de modules fonctionnels et fait prendre au macro-langage une importance encore plus grande.

Nous pouvons remarquer enfin, que le mécanisme de macro-requêtes du macro-éditeur est assez général pour s'appliquer à tout autre environnement conversationnel qui dispose d'un langage de commande analogue à celui d'EDIT.

## 4. PROBLEMES GENERAUX

### 4.1 LIAISONS EDITEUR-SYSTEME

La plus gros reproche que l'on puisse faire à 'EDIT' c'est de ne pas être modulaire. Ainsi il est totalement lié au système et au terminal.Le fait de ne pas être ré-entrant n'est pas un problème dans le cadre de C.M.S., mais peut le devenir pour des systèmes conversationnels à partage de temps.

A la lumière des caractéristiques des différents éditeurs, décrits au chapitre I et après les deux expériences décrites précédemment, nous pouvons faire le bilan des problèmes posés par la conception d'un éditeur:

— Dégager les principales fonctions du programme d'édition, pour définir des modules indépendants et des conventions de liaisons. Il faut alors tenir compte des contraintes imposées par le système: ré-entrance- encombrement gestion de fichier- terminal etc...

- Choisir le langage avec lequel sera écrit l'éditeur. Le langage d'assemblage qui permet une utilisation plus "concrète" des ressources du calculateur et une mise au point relativement aisée ("dumps"- "debugging")

ne facilite pas l'écriture d'algorithmes généraux. Un langage de programmation "classique" (FORTRAN- ALGOL- COBOL- PL/1) ne comporte en général pas assez d'opérations sur les chaînes de caractères, et n'offre pas toujours un compilateur qui fournit du code machine ré-entrant.

On peut penser à l'un des langages spécialisés dans le traitement de chaîne (SNOBOL4 par exemple) mais la plupart d'entre eux sont interprétés.

### 4.2 CHOIX DU TERMINAL

C'est grâce au terminal que l'utilisateur pourra se servir de l'éditeur.

- Un "télétype" connecté de manière à envoyer caractère par caractère permet un grand nombre de fonctions d'édition mais comme elles correspondent chacune à un caractère il est difficile de s'y retrouver (cf TECO-QED).

- Un terminal de type 'machine à écrire' avec "buffer" de ligne ne permet pas une véritable interaction à cause de sa lenteur mais offre quand même la possibilité d'avoir un langage plus compréhensible.

- Un terminal graphique paraît mieux adapté du fait de sa rapidité et de ses qualités conversationnelles; Néanmoins, il ne reste aucune trace des modifications faites.

### 4.3 GESTION DU FICHIER A EDITER

C'est la partie qui pose le plus grand nombre de problèmes:

Faut-il laisser le fichier à éditer sur disque ou faut-il en prendre une copie en mémoire ? Dans le premier cas on peut s'attendre à des temps de réponse très longs (pour une recherche par contexte, par exemple) et dans le second cas on sera vite limité par la taille de la mémoire disponible. Une solution hybride (une partie du fichier en mémoire l'autre sur disque) impose une liaison délicate entre l'information sur disque et l'information en mémoire

et conduit à des problèmes de mise à jour du fichier, au fur et à mesure des insertions, suppressions etc...

- Des garanties sur les modifications doivent être données. Il est nécessaire qu'en cas d'arrêt brusque du système, l'utilisateur puisse retrouver ses fichiers, au moins tels qu'ils étaient avant l'édition! Dans la mesure du possible les opérations réalisables sur un fichier doivent être réversibles.

Puisqu'un éditeur est fait pour traiter des fichiers aux caractéristiques différentes (programmes- données- textes) il doit pouvoir faire la distinction entre différents types de fichiers. Ainsi un programme FORTRAN ne doit pas être édité de la même façon qu'un programme en langage machine. Les tabulations sont différentes, les colonnes de troncature également. Si l'éditeur reconnaît déjà un certain nombre de types, d'usage courant: FORTRAN, ALGOL, ASSEMBLEUR, DONNEES, TEXTES etc... Comment faire pour lui décrire de nouveaux types ou pour changer certaines caractéristiques (longueur, tabulations sérialisation) ?

### 4.4 LANGAGE D'EDITION

Quelle forme doit avoir la langage d'édition ? Ce dernier est conditionné, tout d'abord, par l'existence d'environnements (sous 'EDIT', création et modification) et par ce que l'on pourrait appeler "l'unité d'édition": ligne ou caractère. La présence d'un pointeur de ligne ou de caractère courant entraîne, pour les requêtes, l'existence d'un paramètre par défaut.

Le choix des fonctions d'édition n'est pas aisé:

- On peut fournir un ensemble de requêtes bien déterminées, sans possibilités d'extensions ou au contraire ne donner qu'un certain nombre de "primitives" mais avec possibilité de les composer pour étendre le langage.

Les types de paramètres, sont, en général, bien définis: nombres- chaînes- lignes- ou modèles de chaînes pour les recherches par contexte.

Dans le cas d'une série fixée de requêtes chacune est traitée par un sous-programme qui peut lui même employer d'autres sous-programmes de traitement. Par contre lorsqu'il s'agit de "primitives" chacune peut constituer un module indépendant.

Le mécanisme d'extension peut être plus ou moins élaboré, selon les possibilités générales du système. Ce qui paraît utile, c'est d'obtenir une souplesse d'adaptation à un nombre maximum d'utilisations particulières, afin de re-définir les caractéristiques du fichier édité ou des parties du langage. d'édition lui-même.(caractère de tabulation logique- codes des caractères-requêtes etc...).

## 4.5 UTILISATION GENERALE

L'emploi de l'éditeur doit être facile même pour les personnes non spécialisées en programmation. Le manuel d'utilisation doit être simple et précis avec de nombreux exemples.

En outre, il faut que l'éditeur fournisse lui-même des explications sur l'emploi de certaines requêtes: syntaxe exacte- type de paramètres etc...). Les réponses aux requêtes -résultats de l'exécution ou messages d'erreurs- doivent être précises. Néanmoins il est agréable, dès que l'on acquis une certaine expérience, de rendre l'éditeur moins bavard, toujours à l'aide d'une requête.

En cours d'édition il peut être utile d'avoir accès à certains renseignements comme le nom, le type du fichier, ses caractéristiques, le numéro de la ligne courante etc...

On peut envisager la conservation de toutes les actions de l'utilisateur, pour qu'il puisse avoir une trace de son édition.

Enfin, des programmes utilisateurs peuvent faire appel, en cours d'exécution, à l'éditeur pour bénéficier de telle ou telle fonction d'édition.

Il est alors souhaitable que les modules qui composent l'éditeur soient bien définis et que leur utilisation par d'autres programmes soit possible.

Dans la seconde partie c'est à toutes ces questions que nous allons tenter de donner une réponse en définissant des structures générales pour les programmes d'édition.

*"Adonne-toi à l'étude des lettres pour en tirer quelque chose qui soit toute tienne".*

*MONTAIGNE - Essais -*

# D E U X I E M E  P A R T I E

## CONCEPTION D'UN EDITEUR GENERAL

### CHAPITRE V : STRUCTURES GENERALES DES PROGRAMMES D'EDITION

1. CAHIER DES CHARGES

2. CARACTERISTIQUES DES FICHIERS A EDITER

   2.1 NOTION DE FICHIER

   2.2 DESCRIPTEUR

   2.3 VARIABLES LOCALES D'EDITION

3. CLASSEMENT DES FONCTIONS D'UN EDITEUR

   3.1 ENTREE D'INFORMATION

   3.2 ANALYSE

   3.3 SUPERVISEUR D'EXECUTION

   3.4 FONCTIONS D'EDITION

   3.5 GESTION DE FICHIER

   3.6 SORTIE

4. MODULARITE ET RE-ENTRANCE

   4.1 ZONES PARTAGEABLES

   4.2 CONVENTIONS DE LIAISON

   4.3 UTILISATION DYNAMIQUE DES MODULES

# C H A P I T R E  V

## STRUCTURES GENERALES DES PROGRAMMES D'EDITION

### 1. CAHIER DES CHARGES

Pour résumer les principaux problèmes posés auparavant dressons le "cahier des charges" d'un éditeur idéal :

a) Réaliser l'édition d'un fichier de taille arbitrairement grande avec des temps de réponse "agréables".

b) Editer n'importe quel type de fichier (image de carte ou autre) en gardant la possibilité de modifier dynamiquement les caractéristiques des éléments du fichier.

c) Avoir un langage de commande simple mais extensible pour s'adapter aux besoins particuliers de tels ou tels utilisateurs.

d) Utiliser les fonctions d'édition dans un environnement conversationnel pour créer et modifier des fichiers mais aussi, pour, d'une part, faire de la mise à jour automatique et d'autre part permettre l'emploi de certaines parties de l'éditeur par des programmes utilisateurs (modularité).

e) Réaliser l'édition de plusieurs fichiers à la fois au niveau d'un système à temps partagé et au niveau d'un utilisateur (ré-entrance).

f) Adapter facilement l'éditeur à n'importe quel type de terminal de communication.

g) Eviter une liaison trop étroite entre la méthode d'accès aux fichiers du système et l'ensemble de l'éditeur.

h) Fournir toutes les garanties possibles pour la sauvegarde du fichier édité en cas de panne du système ou de fausse manoeuvre de la part de l'utilisateur.

Pour répondre à toutes ces exigences, nous allons nous placer en dehors de toute machine, de tout système à temps partagé et de tout type de terminal.

Nous essaierons tout d'abord d'expliciter la notion de fichier afin de dégager la structure s'adaptant le mieux à l'édition.

Ensuite, pour arriver à la définition d'un éditeur idéal, il nous faudra classer les différentes fonctions d'un programme d'édition. Nous dégagerons un certain nombre de fonctions de base, véritables primitives, qui correspondront chacune à un module bien défini. C'est cette structure modulaire qui sera développée par la suite pour répondre aux contraintes du langage de commande et pour arriver à des performances intéressantes.

## 2. CARACTERISTIQUES DES FICHIERS A EDITER

### 2.1 NOTION DE FICHIER

Les fichiers représentent un ensemble d'informations et sont utilisés de façon diverses par tous les systèmes d'exploitation. On a souvent cherché à formaliser la notion de fichier. Ainsi le comité "SHARE" [S7] . a proposé de définir un fichier comme étant un ensemble de triplets: ($e_i$, $p_j$, $V_{ij}$) dans lequel:

- $e_i$ représente une entité décrite en vue d'un certain traitement

- $p_j$ est une propriété des entités à laquelle on peut attribuer des "valeurs".

- $V_{ij}$ est la valeur de la propriété $p_j$ pour l'entité "$e_i$".

*Exemple :*

Considérons le fichier des assurés sociaux. Les entités sont les assurés sociaux. Parmi les propriétés, on aura le nom, la nationalité, le sexe, etc...

$e_3$: DUPONT Jacques 02/08/44 6 Allée des Platanes

Si $p_1$ est le nom et $p_3$ la date de naissance, on a

$$V_{31} = \text{DUPONT}$$

et

$$V_{33} = 02/08/44$$

Un fichier peut aussi avoir une représentation matricielle :

$$\text{entités} \begin{cases} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_m \end{cases} \overset{\text{propriétés}}{\begin{array}{c} \begin{array}{cccc} p1 & p2 & \cdots\cdots & pn \end{array} \\ \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ \vdots & & & \\ V_{m1} & V_{m2} & \cdots & V_{mn} \end{bmatrix} \end{array}}$$

Dans le cadre plus particulier de l'édition on peut définir le fichier comme étant une série de caractères successifs dont l'un est choisi (retour chariot par exemple) pour jouer le rôle de délimiteur et caractériser ainsi des entités (lignes) (cf : QED TECO-Chap.I)

Cette définition ne nous semble pas satisfaisante car elle s'éloigne trop de l'idée que l'utilisateur se fait de la structure logique de son fichier.

Il est préférable de considérer un fichier comme une liste d'éléments (des lignes), le terme de 'liste' impliquant une relation d'ordre. Ces

éléments possèdent un certain nombre de propriétés, disons plutôt des 'caractéristiques' qui sont décrites par des paramètres. Les valeurs de ces paramètres sont regroupées pour constituer le "descripteur" du fichier.

En utilisant la forme de Bacchus, on obtient :

< fichier > ::=< descripteur > < liste d'éléments >

< liste d'éléments > ::= < élément > |<élément> < liste d'éléments >

L'élément comporte en général 4 champs :

< élément > ::= < longueur ><indicateur> < pointeur> < valeur >

< indicateur >::= état de l'élément : modifiable - nom modifiable, modifié, non-modifié.

< pointeur > ::=< emplacement élément suivant>< emplacement élément précédent >

< valeur > ::= < atome >|< atome> < valeur> | Ø

C'est seulement dans la partie 'valeur' que nous trouvons un certain nombre d'atomes,dont la nature est fonction de l'implantation (bits- caractères mots). Les trois autres parties permettent de caractériser l'élément, d'abord en donnant le nombre d'atomes qui composent sa partie valeur, puis en fournissant les moyens d'accès aux éléments précédents et suivantes. Enfin, la partie "indicateur" fournit,à un instant donné, l'état de cet élément. En cours d'édition, on peut ainsi savoir entre autres si un élément a été modifié ou non.

Cette façon de considérer un fichier offre, pour l'édition, les avantages suivants:

1) La structure de liste apporte la simplification des opérations classiques: suppression, insertion, remplacement.

2) L'élément est structuré de telle sorte qu'il puisse s'adapter à n'importe quelle implantation: Mémoire principale- Disque- bandes etc... il est possible alors que l'une des parties ne soit plus nécessaire -par exemple-

partie pointeur pour des enregistrements séquentiels sur disque.

3) Le fait de regrouper tous les paramètres concernant le fichier dans le descripteur permet d'avoir une édition plus souple, s'adaptant aux particularités de chaque fichier. De plus, l'utilisateur doit pouvoir modifier les paramètres du descripteur pour l'édition en cours, ou pour toutes les autres.

## 2.2 DESCRIPTEUR DE FICHIER

Quels sont les paramètres à conserver dans le descripteur ?

En examinant la gestion de fichier dans quelques systèmes conversationnels comme MULTICS, CMS, TSS [S6- S8] et en se limitant toujours à l'édition, on peut dresser la liste suivante :

1. NOM du propriétaire du fichier (son créateur ou son utilisateur)

2. NOM du fichier- choisi par son propriétaire

3. TYPE du fichier ou sa CLASSE: Indique la destination du fichier: exemple sous CMS types 'FORTRAN' ou 'SYSIN' (pour programmes en assembleur 360) SNOBOL, LISTING, PL/1, COBOL, DATA (pour des fichiers de données), SCRIPT (fichiers destinés à l'impression).

4. MODE Précise la façon dont on peut utiliser le fichier: Privé, public, lecture seulement, écriture seulement, lecture et écriture, ni lecture ni écriture.

5. FORMAT des enregistrements (des éléments) qui composent le fichier: longueur fixe- variable- indéfinie.

6. LONGUEUR des enregistrements.

7. FACTEUR DE BLOCS : nombre d'enregistrements par blocs

8. NOMBRE D'ELEMENTS du fichier

A cette liste se rajoutent des paramètres particuliers à l'édition:

9. SERIALISATION : indique si oui ou non on désire que les éléments du fichier soient numérotés (pour des images de cartes surtout).

10. TABULATIONS: intervalles de tabulation liés au fichier par son utilisation future (destiné à des assembleurs ou compilateurs).

11. COLONNE DE TRONCATURE: Matérialise le fait que la partie valeur d'un élément doit être tronquée en accord avec les contraintes d'un assembleur ou d'un compilateur. Ainsi tout élément d'un fichier contenant un programme en langage symbolique pour le S/360 doit être tronqué à 71 caractères.

12. MARGES D'EDITION : Colonne de gauche et colonne de droite ainsi que rang de 2 éléments, définissant un domaine d'application pour les fonctions d'édition (voir FIG.11)

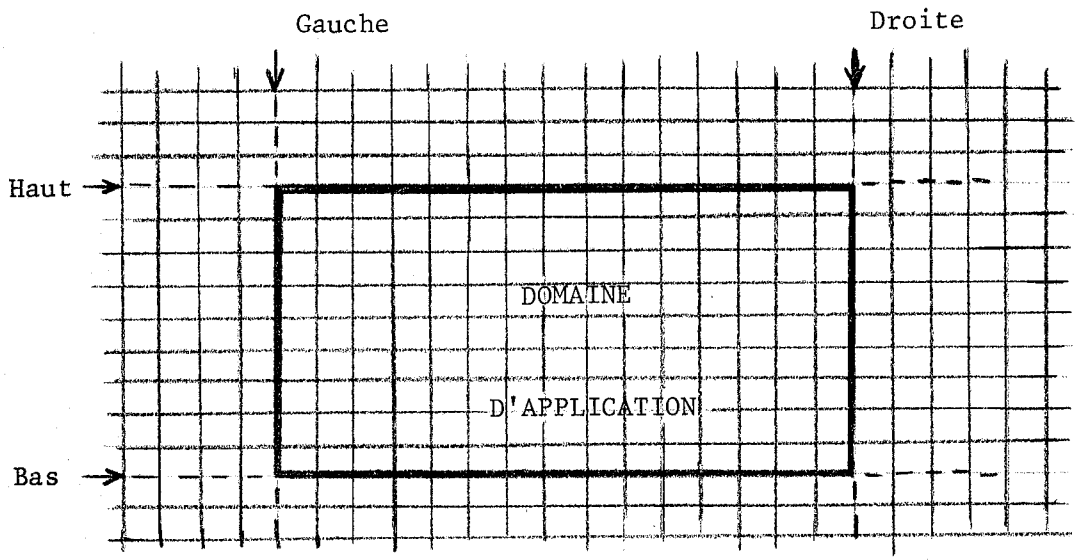



FIGURE 11

13. CONTROLE CARACTERES SPECIAUX

L'utilisation de certains caractères par le langage de commande de l'éditeur peut amener des ambiguités lorsque ces caractères apparaissent dans le fichier. Par exemple, avec EDIT de CMS, le caractère '>' est par défaut le caractère de tabulation logique; or il est utilisé dans des langages de programmation comme ALGOL, SNOBOL ou PL/1

Pour un utilisateur donné, en fonction du nom et du type du fichier quelques paramètres (4-5-6-7-9-10-11-12-13) peuvent avoir des valeurs par défaut. Il est nécessaire, cependant, que le langage d'édition permette la modification de certains paramètres. On pourrait ainsi redéfinir les caractéristiques de son fichier pour modifier le cours de l'édition.

Nous n'avons pas fait mention de paramètres liés plus particulièrement à la gestion de fichiers du système. Les renseignements concernant l'emplacement physique du fichier sont employés quand l'éditeur reçoit le contrôle et nous allons voir au paragraphe suivant où les conserver.

D'autres paramètres comme: date de création, date de dernière mise à jour peuvent être également incorporés au descripteur.

## 2.3 VARIABLES LOCALES A L'EDITION

Pour réaliser l'édition effective, il est indispensable, en plus des paramètres contenus dans le descripteur, de disposer d'autres renseignements sur :

- l'adresse physique des éléments (pointeurs)
- les variables, liées à l'éditeur, qui vont évoluer en cours d'édition:
  - les pointeurs courants de ligne et de caractère, en mémoire ou sur disque.
  - le nombre d'éléments en mémoire
- éventuellement, l'adresse de la copie du fichier sur laquelle est faite l'édition (Marges d'édition)
- les indicateurs divers: recherche par contexte positive ou non. Positions particulières du pointeur de ligne (en haut et en bas du fichier).

Toutes ces variables constituent une sorte de 'descripteur d'édition' qu'il faut associer au descripteur de fichier.

Si l'on regroupe ces deux descripteurs dans une zone oommune à laquelle peuvent accéder tous les modules composant l'éditeur, celle-ci contient alors, à tout instant, l'état complet de l'édition en cours.

## 3. CLASSEMENT DES FONCTIONS D'UN EDITEUR

Plaçons-nous dans le cadre d'un éditeur fonctionnant à partir d'un terminal de type machine à écrire où l'action de l'utilisateur consiste à émettre une chaîne de caractères.

Entre cette action et la réponse fournie par l'éditeur, se déroule tout un processus que nous allons décomposer. Cette décomposition nous permettra de dégager les principales fonctions d'un programme d'édition. Nous pourrons alors associer à une même série de traitements un module fonctionnel et concevoir l'éditeur comme un ensemble de modules qui se partagent des zones de variables (voir Fig.12).

### 3.1 ENTREE D'INFORMATION AU TERMINAL

Ce premier module, lié au terminal réalise les opérations d'entrée (envoyer un ordre de lecture ou un ordre d'analyse d'interruption) et range les informations (chaîne lue, nombre de caractères etc...) dans une zone particulière accessible aux autres modules.

La chaîne lue doit subir une analyse qui dépend alors du langage d'édition.

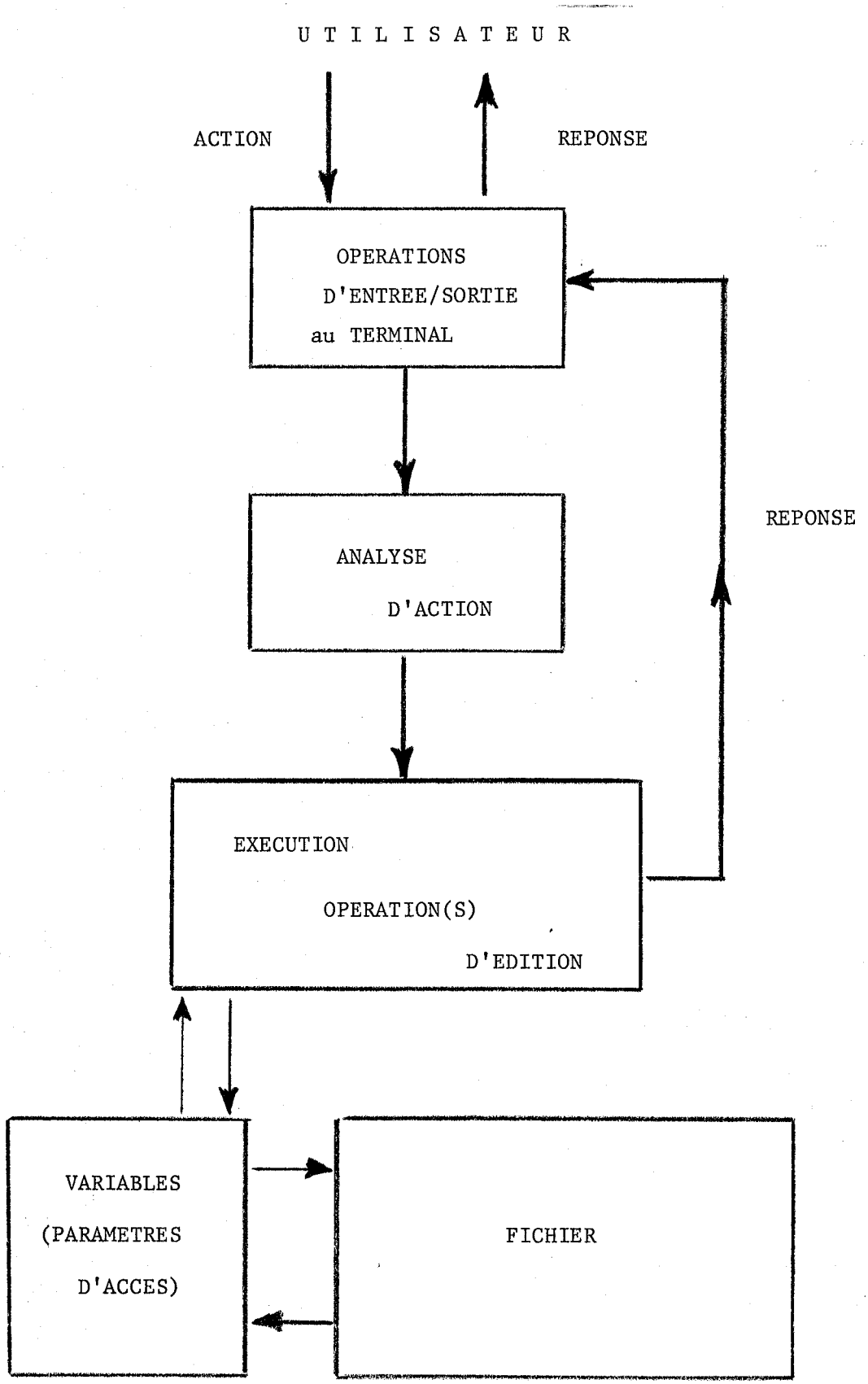



FIGURE 12. FONCTIONS D'UN PROGRAMME D'EDITION

## 3.2 ANALYSE DE L'ACTION DE L'UTILISATEUR.

Dans beaucoup de systèmes cette analyse est fonction de l'état dans lequel se trouve l'éditeur à l'instant de la lecture: par exemple, sous EDIT de CMS il existe deux environnements: création(INPUT), modification(EDIT). Dans tous les cas la chaîne de caractères à traiter peut représenter:

- soit une requête (ou macro-requête) avec ses paramètres
- soit une ligne qui sert de paramètre à une requête
- soit une nouvelle ligne à insérer dans le fichier.

Dans ce dernier cas, elle est transmise à la partie de l'éditeur qui gère le fichier.

Le cas d'une requête est plus complexe:

Il faut en effet reconnaître la fonction demandée et en vérifier la syntaxe: place du nom, des paramètres, des délimiteurs etc...

Il faut ensuite isoler les paramètres, éventuellement les mettre en forme (traduire les nombres- extraire les chaînes pour les recherches par contexte) avant de pouvoir exécuter l'opération demandée.

En choisissant un langage d'édition avec une syntaxe bien définie; la phase d'analyse se décompose en deux parties:

- Une analyse syntaxique de chaque requête

- Une mise en forme de la requête qui peut nécessiter une véritable compilation avec génération d'un code interne à l'éditeur.

Si un utilisateur appelle une macro-requête il faut alors retrouver les requêtes qui la composent pour les mettre en forme ( à moins qu'elles ne soient conservées sous forme codée). En outre, il faut envisager la substitution des paramètres formels par des paramètres effectifs.

Cette partie du traitement peut donc être réalisée par un module qui ne dépend que des règles syntaxiques du langage d'édition et des caractéristiques du macro-langage. Elle fournit une série de commandes valides exécutables

directement.

### 3.3 SUPERVISEUR D'EXECUTION

Pour exécuter une opération simple avec retour immédiat à l'environnement attente de requêtes et surtout pour pouvoir enchaîner automatiquement une série d'opérations, il est nécessaire de créer un mécanisme d'interprétation des commandes, lequel offre les avantages suivants :

- Chaque opération d'édition est matérialisée par un sous-programme du superviseur.
- Dans un contexte conversationnel un mécanisme très puissant de macro-requêtes peut être facilement incorporé à l'éditeur.
- Dans un contexte non conversationnel il suffit de fournir au superviseur la série des commandes à réaliser pour obtenir la mise à jour automatique d'un fichier.

Selon les cas l'activité du superviseur d'exécution correspond soit à une simple consultation de table avec appel d'un sous-programme de traitement, soit à un mécanisme d'interprétation complet avec gestion de piles comme nous l'avons vu pour le macro-éditeur (chap. III).

Si les opérations sont codées, le superviseur a pour tâche de décoder chaque instruction, d'appeler le module correspondant, et de reprendre le contrôle après exécution de l'instruction.

### 3.4 OPERATIONS D'EDITION

Si l'on examine le langage de commande de divers éditeurs Chap.I) on s'aperçoit que finalement on peut classer les requêtes en 3 grandes catégories:

1- Requêtes de Position

2- Requêtes de Modification

3- Requêtes de Contrôle de l'édition

### 3.4.1 Requêtes de Position

Elles permettent d'accéder à une ligne et à un caractère particulier dans le fichier soit par son numéro (ligne ou caractère), soit par une chaîne dont on recherche une occurrence.

ON peut même déterminer une zone d'édition en donnant les adresses de deux lignes et les numéros de 2 colonnes. Cette zone devient (cf. 2-2) le domaine d'application d'une future opération d'édition. Les déplacements dans le fichier se font de deux façons:

- déplacement absolu

- déplacement relatif à partir de la position courante

Par 'position courante' on entend: valeur du pointeur qui repère, à un instant donné, une ligne particulière. L'existence d'un pointeur de caractère s'avère également très utile. On dispose alors de deux dimensions et le fichier peut être assimilé à une matrice à n lignes et p colonnes.

Les requêtes de 'positionnement' font évoluer :

- Le pointeur de ligne seul (indice de ligne) si l'on recherche une ligne particulière

- Le pointeur de caractère seul (indice de colonne) si, dans la ligne courante, on veut définir un nouveau caractère courant.

- Le pointeur de ligne et celui de caractère pour repérer dans le fichier, à la fois la ligne qui contient une chaîne particulière et le premier caractère de cette chaîne.

### 3.4.2 Requêtes de modification

Elles peuvent agir soit au niveau des lignes, soit au niveau des caractères dans une ligne. Il s'agit le plus souvent de remplacer une information par une autre. Par exemple :

- La suppression de la ligne courante est équivalente à son remplacement par une ligne vide.

- L'insertion se traduit par le remplacement d'une ligne vide par une ligne donnée.

- Au niveau des caractères: remplacement d'une chaîne par une autre chaîne, remplacement d'une chaîne par des blancs etc...

Enfin, dans les fonctions de modifications il faut inclure les fonctions réalisant l'échange d'information entre les fichiers: remplacement d'un groupe de lignes par un autre.

### 3.4.3 Fonctions de contrôle

On englobe ici toutes les fonctions qui permettent à l'utilisateur de contrôler l'édition:

- Modification directe des valeurs des paramètres du descripteur de fichier.
- Examen des parties du fichier (modifiées ou non).
- Explications sur l'emploi des requêtes
- Contrôle des réponses de l'éditeur (mode muet, bavard)

## 3.5 GESTION DU FICHIER

L'exécution des fonctions d'édition passe par la modification des paramètres d'état du fichier, groupés dans le descripteur. Les modifications doivent être effectives sur le fichier quel que soit son support (bandes, disques etc...) Pour cela les lignes concernées doivent être amenées en mémoire centrale. Cette dernière opération dépend de la (ou des) méthode(s) d'accès aux fichiers disponibles(s) sous le système. On peut donc grouper dans un même module tout ce qui concerne les manipulations du fichier sur disque(ou autre) et en mémoire. Ces manipulations se font par l'intermédiaire du descripteur.

### 3.6 SORTIE AU TERMINAL

A toute action de l'utilisateur doit correspondre une réponse de l'éditeur. Les réponses constituent des sorties d'information au terminal et se traduisent souvent par l'impression de chaînes de caractères:

- Messages d'erreurs
- Explications de fonctionnement
- Liste d'éléments du fichier

Si l'on rapproche ces opérations de celles vues en 3.1, il est nécessaire d'avoir un module réalisant les opérations d'entrées/sorties, qui est le seul à être lié au terminal.

## 4. MODULARITE ET RE-ENTRANCE

### 4.1 ZONES PARTAGEABLES

D'après ce que nous avons détaillé auparavant, à chaque fonction correspond un module. L'ensemble des modules constitue l'éditeur.

Ces modules doivent communiquer entre eux, il est donc nécessaire d'avoir des zones de paramètres qu'ils peuvent se partager.

Pour assurer la ré-entrance de l'éditeur, chaque module ne doit avoir aucune variable locale et les instructions ne doivent pas se modifier.

Les zones partageables sont :

- le descripteur du fichier

. Il contient toutes les informations nécessaires à l'édition et permet de connaître l'état du fichier à un instant donné.

- Une zone de travail et de rangement: chaînes lues ou à écrire, requêtes à analyser, paramètres de travail etc...

## 4.2 CONVENTIONS DE LIAISON

Chaque module doit être appelé et doit appeler à son tour un autre module de manière analogue:

- Sauvegarde des registres de la machine chez le module appelé
- Transfert de paramètres par l'intermédiaire des zones partageables.

## 4.3 UTILISATION DYNAMIQUE DES MODULES

Il est préférable de prévoir un mécanisme de chargement dynamique des modules pour réaliser l'implantation de l'éditeur dans des systèmes ne disposant que des petites tailles de mémoire. En effet, en cours d'édition, il est évident que tous les modules n'ont pas à résider en mémoire principale.

Il est utile également de prévoir l'emploi de certains modules d'édition par des programmes quelconques.

# CHAPITRE VI : DESCRIPTION D'UNE MACHINE D'EDITION

# CHAPITRE VI

# DESCRIPTION D'UNE MACHINE D'EDITION

## 1. CARACTERISTIQUES DE BASE

### 1.1 MACHINE D'EDITION

Quand on veut effectuer une opération d'édition classique (insertion, remplacement etc...) sur un élément d'un fichier, il faut caractériser la position de cet élément soit par son numéro d'ordre, soit en citant tout ou partie de son contenu.

On peut faire alors un parallèle entre un fichier à éditer et une unité à accès direct, par exemple un disque magnétique: avant de lire ou d'écrire un enregistrement, il faut placer la tête de lecture/écriture au moyen d'ordres envoyés à une unité de contrôle.

Considérons donc le fichier comme un "périphérique" à accès direct rattaché à une "unité de contrôle" réalisant non seulement le "positionnement" mais, en plus, certaines modifications sur les éléments du fichier.

En poussant le parallèle plus loin encore disons que cette unité de contrôle reçoit ses ordres d'une "unité Centrale" qui supervise l'exécution d'un "programme" tapé par un "opérateur" (l'utilisateur) à une console (le terminal).

On définit ainsi une machine d'édition comportant :

1. Une unité centrale
2. Une unité de contrôle
3. Un périphérique
4. Une console opérateur
5. Un mécanisme permettant d'interpréter des ordres émis depuis la console, pour les transformer en "programmes" exécutables.

## 1.2 PRINCIPE DE FONCTIONNEMENT

Chaque partie d'une telle machine correspond à un ou plusieurs modules fonctionnels décrits au chapitre précédent.

- L'unité Centrale joue le rôle de Superviseur d'exécution
- L'unité de Contrôle qui gère le périphérique (le fichier) réalise les fonctions d'édition.
- La Console de l'opérateur sera prise en charge par le module qui traite les opérations de lecture/écriture.
- Le mécanisme d'interprétation correspond à l'analyse de l'action demandée par l'opérateur (c'est l'utilisateur de la machine).

Nous définissons ainsi pour la machine un jeu d'instructions qui correspond aux fonctions élémentaires, comprenant outre les fonctions de base pour l'édition et la gestion du fichier, les ordres de lecture ou d'écriture au terminal et le contrôle du fonctionnement global de la machine.

Ces instructions agissent sur des opérandes qui se trouvent dans les "registres" de la machine. En effet, toutes les variables concernant le fonctionnement de l'éditeur (nombres, chaînes) sont mises dans des zones de mémoire partageables avec possibilité, pour l'utilisateur, de désigner symboliquement l'emplacement de certaines variables.

De la même façon, les variables de description du fichier correspondront à des symboles reconnus par le langage d'édition comme des "registres de description", dont le contenu peut être modifié.

### Mode "Séquentiel et Mode "Programmé"

Lorsque l'utilisateur a le contrôle, c'est-à-dire lorsque l'éditeur est en lecture au terminal on dira que la machine d'édition est en mode séquentiel .

Sans quitter ce mode on peut émettre et exécuter une instruction élémentaire.

Puisque le langage de base est très rudimentaire, il faut prévoir un langage de communication plus évolué qui sera en quelque sorte "micro-programmé".

Chaque instruction du langage de communication correspond à une série d'instructions élémentaires (un micro-programme). Ceci veut dire que l'analyse de l'action de l'utilisateur se ramènera alors à la recherche du micro-programme concerné dans une sorte de bibliothèque.

Lorsque l'Unité Centrale exécute un micro-programme on dira que la machine fonctionne en mode programmé.

Le langage de communication va correspondre à une bibliothèque de micro-programmes, fournie avec l'éditeur. En outre, chaque utilisateur pourra posséder une biliothèque privée en définissant ses propres micro-programmes au moyen d'un "micro-assembleur".

### 1.3 MODULES ET ZONES PARTAGEABLES.

Les composants de la machine d'édition correspondent à une série de modules ré-entrants. Ces modules partagent 3 zones (Fig.13).

- Une zone de fonctionnement comportant les "registres" de l'Unité Centrale et des variables locales d'édition.

- Une zone de description contenant le descripteur et tous les paramètres d'accès au fichier.

- Une zone fichier où les éléments sont amenés pour examen et modification.

Ces trois zones sont acquises dynamiquement à l'initialisation de l'éditeur (cf.2.2).

Le jeu d'instructions se divise en trois groupes:

Gr1. "Unité Centrale": Contrôle du fonctionnement- gestion des registres- calculs- appels de micro-programmes.

Gr2 "Entrée/Sortie Console": Lecture et écriture

Gr3 "Unité de Contrôle": Fonctions élémentaires d'édition- gestion des registres de description.

## 1.4 ADAPTATION A UN SYSTEME DONNE:

La définition d'un éditeur comme une machine est indépendante de tout système.

La réalisation d'une machine d'édition pour un système donné se fait en deux étapes:

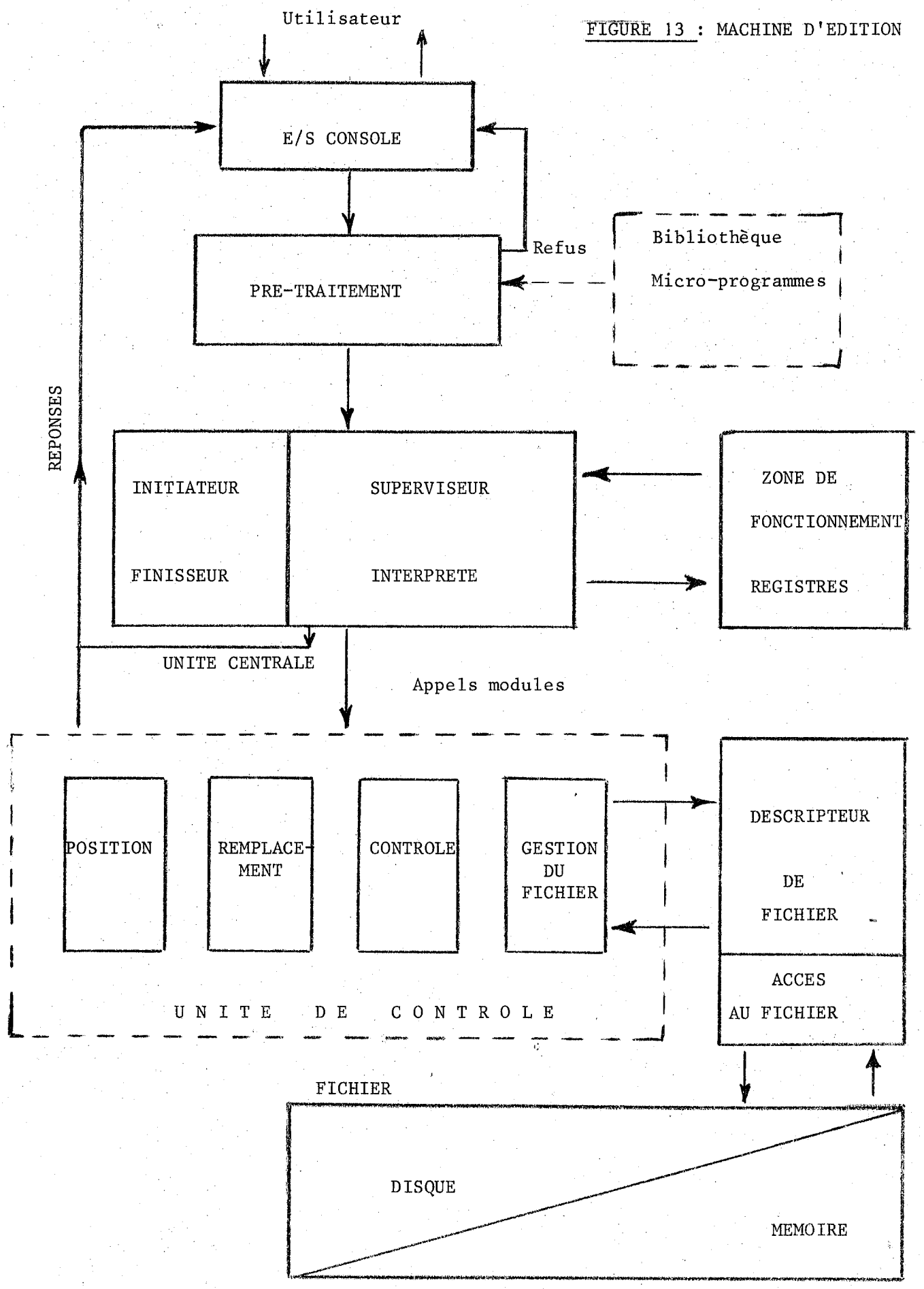
1) Implantation de l'éditeur de base avec ses différents modules et son jeu d'instructions élémentaires. Cet éditeur permet de tester les différents mécanismes de gestion de fichiers et le fonctionnement global de la machine. La mise au point est facilitée par la simplicité du rôle de chaque module.

2) Définition et construction du langage de commande (ou langage de communication- cf.Chap VII) sous forme d'une bibliothèque de micro-programmes. On adapte ainsi le langage au contexte conversationnel dans lequel se trouve l'éditeur.

Une modification quelconque dans le système: gestion de fichier différente , autre type de terminal, etc... se ramène au simple changement du module concerné.

En outre, chaque utilisateur a les moyens d'étendre le langage de commande pour résoudre ses problèmes particuliers.

FIGURE 13 : MACHINE D'EDITION



## 2. L'UNITE CENTRALE

### 2.1 ZONE DE FONCTIONNEMENT

Elle contient les registres de fonctionnement de la machine ainsi que divers indicateurs.

Certains registres sont accessibles grâce aux instructions de base. De même l'état des indicateurs peut être testé.

La composition de cette zone est la suivante:

- 8 Registres numérique nommés symboliquement: R1, R2, R3, R4, R5 R6, R7, R8.

Ils peuvent contenir des nombres et sont modifiables directement par les instructions.

- 4 Registres "chaîne": nommés symboliquement A, B, C, D

Ils sont prévus pour contenir des chaînes provenant du fichier ou de la console. La taille d'un registre est fixée en fonction de la chaîne maximum qui peut être émise à partir du terminal.(132 caractères par exemple). Ces registres sont également modifiables directement.

- 1 Registre instruction (RI) qui contient l'instruction en cours d'exécution (il n'est pas accessible à l'utilisateur).

- 1 Compteur ordinal (CØ) contenant:
  - Soit 0 si la machine est en mode séquentiel
  - Soit l'adresse de la prochaine instruction à exécuter, si la machine est en mode programmé.

- 1 Pile d'interprétation et son pointeur pour gérer les appels successifs de micro-programmes.

- Plusieurs indicateurs :

  . L'indicateur arithmétique INAR qui reflète le signe du résultat d'une instruction arithmétique.

  . L'Indicateur booléen BOOL qui a la valeur vrai ou faux selon le résultat de l'instruction exécutée.

  . Les Indicateurs sur paramètres P1, P2, P3, P4 qui reflètent le type (nombre ou chaîne) des paramètres fournis à l'appel d'un micro-programme.

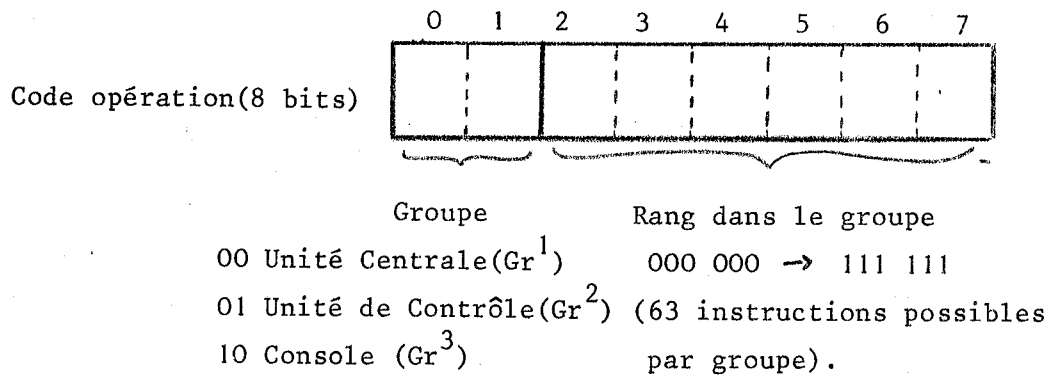
  . Les Indicateurs de débordement des pointeurs d'édition NORD, SUD, EST, OUEST.

  . L'Indicateur de console ECONS qui permet d'ignorer ou non les ordres d'écriture à la console.(mode muet- mode bavard)

- Une zone de travail banalisée

## 2.2 CARACTERISTIQUES GENERALES DES INSTRUCTIONS

Les instructions sont de longueur fixe: 1 mot (sur IBM 360: 32 bits). Une instruction a au plus 2 opérandes. On divise les instructions en trois groupes et le code opération (8 bits) contient les indications sur ce groupe et sur le rang de l'instruction dans le groupe:



Les opérandes des instructions sont soit des registres de fonctionnement (numérique ou "chaîne")soit des registres de description, soit des indicateurs, soit des "adresses".

Lorsqu'un registre A, B, C ou D est concerné par une instruction la longueur de la chaîne qu'il contient (ou qu'il contiendra) est rangée dans un registre numérique avec la correspondance:

| Registre chaîne | Registre Numérique |
|---|---|
| A | R1 |
| B | R2 |
| C | R3 |
| D | R4 |

En outre, le registre 'RO' n'existe pas. Un opérande nul indique, selon le contexte, soit le nombre O soit la chaîne vide.

## 2.3 INSTRUCTIONS DE L'UNITE CENTRALE

Elle permettent le contrôle global de toute l'édition et se décompose ainsi:

- Des instructions de Chargement pour charger un registre de fonctionnement avec le contenu d'un autre registre.
- Des instructions de tests
- Des instructions de rupture de séquence inconditionnelles ou conditionnées par l'état des indicateurs.

Elles permettent, outre la réalisation de boucles contrôlées dans un micro-programme, l'appel de "micro-sous-programmes".

- Des instructions arithmétiques addition, soustraction, pour des calculs très simples.

- Des instructions diverses de contrôle: arrêt machine- conversions - retour de micro-programme- Contrôle du mode de fonctionnement.

Dans de nombreuses instructions le premier opérande est un registre de fonctionnement (numérique ou "chaîne") alors que le type du second opérande peut varier. On réserve alors un certain nombre de bits de type pour préciser les caractéristiques de ce second opérande:

- Adresse d'un mot du micro-programme
- Nom d'un registre de fonctionnement Numérique
- Nom d'un registre de fonctionnement de type "Chaîne"
- Nom d'un registre de description Numérique
- Nom d'un registre de description de type Chaîne.

On trouvera en fin de chapitre la description complète du jeu d'instructions de l'Unité Centrale.

## 2.4 FONCTIONNEMENT DES MODULES DE L'UNITE CENTRALE

L'Unité Centrale se décompose en deux modules distincts:

1. L'INITIATEUR/FINISSEUR qui prend le contrôle en début et fin d'édition.

2. LE SUPERVISEUR/INTERPRETE qui dirige l'ensemble du fonctionnement de la machine et interprète les instructions de l'Unité Centrale (Groupe 1).

### 2.4.1 L'Initiateur/Finisseur

C'est le premier module qui reçoit le contrôle lorsque l'éditeur est appelé.

Il acquiert des zones de mémoire en faisant appel à la gestion dynamique de la mémoire libre du système en particulier:

-1 Zone de fonctionnement pour l'unité centrale qui est partagée par presque tous les modules d'édition qui peuvent ainsi échanger des paramètres.

- 1 Zone de description qui reçoit les caractéristiques du fichier à éditer, soit en les prenant dans un descripteur du système soit en les prenant par défaut.

C'est dans cette zone que résident les paramètres d'accès au fichier sur disque et en mémoire.

- 1 Zone fichier où les éléments sont amenés successivement pour y être examinés (recherche par contexte) et modifiés. (cf. gestion de fichier 4-7).

Ce module initiateur/finisseur détermine l'existence du fichier cité par l'utilisateur lors de l'appel de l'éditeur.

Le type du fichier peut définir, à priori, les caractéristiques des éléments. Il suffit alors de consulter une bibliothèque de types pour connaître les valeurs des paramètres de description.

Mais, grâce aux possibilités offertes par l'éditeur pour la modification de ces paramètres de description, (cf.4-6), l'utilisateur peut gérer à sa guise la bibliothèque de type.

Son travail d'initialisation terminé, l'initiateur/finisseur appelle le superviseur/interprète lequel se met en attente d'une action de l'utilisateur.

Sur exécution d'une instruction spéciale (HALT) l'initiateur/finisseur reprend le contrôle. Il libère les zones de mémoire acquises et rend le contrôle au système sous lequel fonctionne la machine d'édition.

Notons simplement ici que la sauvegarde du fichier édité est à la charge de l'unité de contrôle. (cf. 4-6).

On peut remarquer que ce module initiateur/finisseur ne doit pas nécessairement rester en mémoire tout le temps que dure l'édition, puisqu'il n'intervient plus qu'à la fin de celle-ci. On peut prévoir sont chargement

dynamique lorsque l'utilisateur signale qu'il en a terminé avec l'édition de son fichier.

### 2.4.2 Superviseur/Interprète

Il assure la coordination entre tous les autres modules et exécute les instructions du groupe 1.

En effet, recevant le contrôle de l'initiateur/finisseur et si le fichier existe, il met l'éditeur en mode séquentiel, dans l'attente d'une action de l'utilisateur.

Lorsque le fichier n'existe pas, il active un micro-programme particulier qui crée le fichier avec les lignes d'informations émises par l'utilisateur depuis le terminal. Une ligne vide suspend l'exécution de ce micro-programme ce qui remet l'éditeur en mode séquentiel. Sous ce mode chaque instruction est décodée et exécutée immédiatement, sans que le compteur ordinal évolue. On revient chaque fois à l'attente d'une action de l'utilisateur.

Une instruction du groupe 1 entraîne l'exécution du sous-programme correspondant de l'interprète. Une instruction du groupe 2 ou du groupe 3 provoque l'appel des modules concernés avec les conventions de liaison habituelles (zone de sauvegarde fournie par le programme appelant, adresses des zones partageables dans des registres (réels!)).

Dans le cas, où, en mode séquentiel, on appelle un micro-programme, la machine se met en mode programmé et le mécanisme d'interprétation est activé:

- Le compteur ordinal évolue au fur et à mesure de l'exécution de chaque instruction qui compose le micro-programme.

- Si l'une des instructions du micro-programme est l'appel d'un autre micro-programme, on utilise alors la pile d'interprétation: A chaque appel, la valeur du Compteur ordinal est empilée et l'on exécute le micro-programme appelé. Ce dernier se termine par une instruction de retour qui provoque le "désempilage" et le retour au niveau d'appel.

Lorsque la pile est vide et que l'instruction de retour est rencontrée, le mode séquentiel est rétabli.(cf. Chap. VII-3).

En cours d'exécution d'un micro-programme, il est possible grâce à une instruction spéciale de provoquer le passage en mode séquentiel pour que l'utilisateur puisse vérifier et, éventuellement modifier, l'exécution des opérations d'édition, avant de relancer cette exécution.

## 3. LA CONSOLE-PUPITRE

### 3.1 LECTURE/ECRITURE AU TERMINAL

Le module qui réalise les opérations d'entrées/sorties au terminal est appelé lorsque l'Unité Centrale rencontre une instruction du Groupe 3.

#### 3.1.1 Lecture

L'instruction de lecture a un seul paramètre qui est le nom d'un registre de fonctionnement.

Selon le type de ce registre, on lit un nombre ou une chaîne et on place l'information correspondante dans le registre.(voir en fin de chapitre la description complète de cette instruction).

#### 3.1.2 Ecriture

Là encore, l'information qui se trouve dans le registre spécifié est imprimée au terminal.

Toutefois, afin d'éviter la frappe intempestive de messages, un indicateur spécial (ECONS) contrôle l'écriture. Il indique si l'on doit ou non ignorer les instructions d'écriture. L'état de cet indicateur peut être modifié grâce à une instruction spéciale ('CHUT').

### 3.2 TYPES DE TERMINAUX.

Un tel module est donc composé de deux parties distinctes,

une pour la lecture, une pour l'écriture.

Il est le seul à dépendre du type de terminal utilisé et les instructions décrites précédemment s'appliquent surtout à un terminal de type machine à écrire. Dans le cas d'un terminal graphique (alphanumérique) il faut prévoir des instructions pour gérer l'écran:

- Lire une partie de l'écran et ranger le résultat dans un registre chaîne.
- "Ecrire" sur l'écran une partie du fichier indiquée par 2 valeurs du pointeur de ligne.

A toutes ces fonctions, il faut ajouter, pour certains terminaux, la gestion de diverses interruptions pour les transformer en instructions de base.

## 4. L'UNITE DE CONTROLE

### 4.1 ZONE DE DESCRIPTION

Seconde zone partageable de l'éditeur elle comporte tous les paramètres nécessaires à l'édition d'un fichier.

On distingue :

- Les paramètres de description logique du fichier, modifiables ou non par les instructions de la machine, mais dont le contenu peut être connu par l'utilisateur.

Ces paramètres résident dans ce que nous appelons les registres de description. Ces registres sont désignés symboliquement par un identificateur évoquant le paramètre correspondant.(NOM, TYPE etc...).

- Les paramètres de description physique du fichier,non modifiables et non accessibles à l'utilisateur.Ils servent à l'éditeur pour gérer les copies du fichier sur disque et en mémoire.

### 4.1.1 Paramètres de description logique.

Nous allons énumérer ces paramètres en les classant en deux catégories -Modifiables et non modifiables- et en donnant pour chacun d'eux, le type -Nombre ou chaîne- et le nom symbolique du registre de description correspondant.

#### 4.1.1.1. Modifiables

- Nom du fichier (chaîne) NOM
- Type du fichier (chaîne) TYPE
- Mode du fichier (chaîne) MODE
- Format (chaîne) FORMAT
- Longueur d'un élément(nombre) LONG
- Facteur de blocage(nombre) BLOC
- Sérialisation(chaîne) SER
- Tabulations(chaîne) TABS
- Colonne de troncature (nombre) COL
- Marges d'édition (nombres)
  - -En haut HAUT avec indicateur NORD
  - -En bas BAS " " SUD
  - -A Gauche GAUCHE " " OUEST
  - -A Droite DROITE " " EST
- Pointeur ligne (nombre) PL (1)
- Pointeur de caractère(nombre) PC (1)
- Caractères spéciaux(chaîne)
  - Carac.de remplissage FIL
  - Tabulation logique CTAB
  - Espace arrière CARR
  - etc...

(1) Le pointeur de ligne et celui de caractère ne sont modifiables que par l'intermédiaire de la fonction POSITION (cf. 4-4).

#### 4.1.1.2 Non modifiables

- Nombre d'éléments du fichier
- Nombre de blocs

### 4.1.2 Paramètres de description physique

- Pointeurs sur fichier disque (lecture et écriture)
- Pointeurs sur fichiers en mémoire (premier et dernier élément et élément courant).
- Nombre d'éléments en mémoire.

## 4.2 ZONE FICHIER

On entend par zone fichier, une zone de mémoire dont la taille dépend du système et de la machine utilisés pour le fonctionnement de l'éditeur. Selon le cas, il sera possible ou non d'installer tout un fichier en mémoire. Nous verrons plus loin (4-7) les mécanismes de gestion de fichier, mais dans tous les cas les éléments présents dans la zone mémoire seront organisés sous forme de liste bi-directionnelle. En effet, cette structure est la plus souple pour les opérations de suppression et d'insertion, puisqu'elle ramène ces opérations à de simples modifications de pointeurs.

Afin de perdre le minimum de place, les lignes peuvent être installées en mémoire en enlevant tous les caractères blancs de droite.

Ainsi, un élément en mémoire aurait la structure suivante :

| | | | | | |
|---|---|---|---|---|---|
| L | E | PDISQ | SUC | PRED | .....Caractères ..... |

L : longueur: nombre de caractères de la ligne

E : Indicateur d'Etat. Ligne modifiée (donc à sauver sur disque) ou non

PDISQ : pointeur disque -Emplacement de cet élément sur disque (ou '0' si nouvel élément)

SUC : Successeur en mémoire
PRED : Prédécesseur en mémoire } pointeurs

Les champs 'E' et 'PDISQ' sont prévus pour traiter le cas le plus général où le fichier ne tient pas entièrement en mémoire. Il faut alors transférer sur disque les lignes modifiées et amener en mémoire un autre groupe de lignes.

Dans les cas les plus défavorables, on trouvera toujours en mémoire au moins la ligne courante.

### 4.3 MODULES D'EDITION

L'Unité de Contrôle partage la zone de fonctionnement et la zone de description. Elle se compose de plusieurs modules réalisant les fonctions d'édition proprement dites mais aussi la gestion du fichier sur disque et en mémoire. Le rôle de chaque module est fixé en fonction du type de l'opération d'édition qu'il doit réaliser. Les opérations se font le plus formellement possible c'est-à-dire sans tenir compte de l'implantation exacte des éléments. C'est au module de gestion de fichier à amener en mémoire le ou les éléments concernés. Ainsi il est le seul qui doit tenir compte des contraintes imposées par l'organisation de fichier du système.

Dans ce qui suit nous allons examiner les principales fonctions d'édition pour dégager des primitives et construire le jeu d'instructions de l'Unité de Contrôle.

Nous examinerons enfin les différents mécanismes prévus pour la gestion du fichier édité.

#### Fonctions d'édition :

L'édition est dirigée par deux pointeurs: un pointeur de ligne PL et un pointeur de caractère PC.

Le domaine de variation pour ces 2 pointeurs est défini par les marges d'édition:

- Pour PL: par HAUT et BAS
- Pour PC: par GAUCHE et DROITE

Par défaut, on prend pour marges d'édition:

$$PL \begin{cases} HAUT := 1 \\ BAS := \text{nombre d'éléments du fichier} \end{cases}$$

$$PC \begin{cases} GAUCHE := 1 \\ DROITE := \text{longueur d'un élément} \end{cases}$$

Ces 4 paramètres peuvent être modifiés à volonté par l'utilisateur. Les trois types de fonctions d'édition sont :

1. POSITION pour modifier les valeurs affectées à PL et à PC.

2. REMPLACEMENT pour modifier les éléments du fichier au niveau de la ligne (PL) et au niveau du caractère (PC).

3. CONTROLE de l'édition pour changer les valeurs des paramètres de description.

## 4.4 POSITION

Cette fonction a pour forme générale:

POS PL, PC

Elle permet, comme nous allons le voir, de placer PL sur une ligne particulière et PC sur un caractère particulier, ceci de diverses manières : Déplacement -absolu ou relatif- par des paramètres numériques (N) ou par des paramètres de type chaîne (S:string). Dans ce dernier cas il s'agit de recherche par contexte.

### 4.4.1 Positionnement numérique

#### 4.4.1.1 Absolu

- Sur la $N^{ième}$ ligne: POS N (PL := N) (PC inchangé)
- Sur le $N^{ième}$ caractère: POS,N (PC := N) (PL inchangé)
- Sur la $N^{ie}$ ligne et le $N^{ie}$ carac. POS N,M $\begin{cases} PL := N \\ PC := M \end{cases}$

4.4.1.2 Relatif

- N lignes plus bas POS +N PL := PL + N (PC inchangé)
- N lignes plus haut POS -N PL := PL - N ( " )

- N caractères vers la droite
  POS ,+N PC := PC+N (PL inchangé)
- N caractères vers la gauche
  POS ,-N PC := PC-N (PL inchangé)

- Et les combinaisons :
  POS ±N ,±M $\begin{cases} PL := PL \pm N \\ PC := PC \pm M \end{cases}$

4.4.1.3 Mixte

- PL absolu et PC relatif POS N, ± M $\begin{cases} PL := N \\ PC := PC \pm M \end{cases}$

- PL relatif et PC absolu POS ± N, M $\begin{cases} PL := PL \pm N \\ PC := M \end{cases}$

A chaque valeur limite : HAUT, BAS, DROITE, GAUCHE est associé un indicateur, respectivement NORD, SUD, EST, OUEST. L'attribution à PL ou à PC d'une valeur supérieure ou inférieure aux valeurs limites fait prendre à PL ou à PC cette valeur limite et fait basculer l'indicateur correspondant.

4.4.2 Recherches par contexte

Soit S une chaîne quelconque et l'on veut en rechercher une occurrence.(La longueur limite est fonction de la taille des registres chaînes de l'Unité Centrale).

La recherche peut être positive ou négative, cela signifie au niveau de la machine d'édition, qu'une valeur booléenne est affectée à l'indicateur BOOL.

Dans le cas d'une recherche positive, BOOL prend la valeur 'VRAI', PL repère la ligne contenant S tandis que PC repère le premier caractère (à gauche) de la chaîne, dans la ligne.

Dans le cas d'une recherche négative, on peut soit "se bloquer" sur les marges d'édition , soit rendre à PL et PC leurs valeurs initiales. Quoi qu'il en soit, l'indicateur BOOL prend la valeur 'FAUX'.

#### 4.4.2.1 Recherche absolue POS S

La recherche se fait de haut en bas du fichier, elle commence au premier élément (ou à celui de rang HAUT) et s'achève au dernier (de rang BAS) lorsque aucune occurrence de S n'a été rencontrée.

Chaque ligne est explorée de gauche à droite en partant du premier caractère (GAUCHE) et en allant, si nécessaire, jusqu'au dernier (DROITE).

#### 4.4.2.2 Recherche relative POS ± S

La recherche a lieu à partir de la position courante de PL vers le haut (-) ou vers le bas (+). Pour chaque ligne la recherche se fait de GAUCHE à DROITE sans tenir compte du PC.

#### 4.4.2.3 Recherche conditionnée par PC

POS S, ±
POS ± S, ±

La recherche se fait de manière absolue ou relative pour PL mais chaque ligne est explorée à partir de la position courante de PL, vers la droite(+) ou vers la gauche(-).

#### 4.4.2.4 Mode ancré

POS S,⚓
POS ± S,⚓

La recherche est absolue ou relative au niveau de PL mais la chaîne cherchée doit commencer à la position courante de PC.

#### 4.4.2.5 Recherches dans la ligne courante

POS ,S

Recherche absolue de la chaîne S dans la ligne courante (de GAUCHE à DROITE).

POS , ± S

Recherche relative de S vers la droite (+) ou vers la gauche(-) de PC.

Cette fonction POSITION est très complète; Etant donnée la possibilité de modifier les marges d'édition HAUT, BAS, DROITE, GAUCHE, on peut réaliser toutes sortes d'opérations de recherche sur des éléments particuliers.

Cependant, toutes les possibilités de cette fonction peuvent s'exprimer au moyen de certaines primitives qui vont nous permettre de définir les instructions élémentaires de position pour l'unité de Contrôle. On trouvera ci-après la description complète de ces instructions élémentaires.

Le fait de trouver, pour PL des primitives qui font progresser ce dernier de ± 1 doit être rapproché du fait que le fichier à éditer peut ne pas être entièrement en mémoire et qu'il est nécessaire d'amener du disque, les lignes une à une.

4.4.2.6 Recherche avec caractère de remplissage.

Un autre type de recherche par contexte est disponible.

POS M

Le paramètre de la recherche n'est plus une chaîne au sens ordinaire mais un masque. Dans ce dernier, un caractère spécial, (le blanc par défaut) joue un rôle neutre, en ce sens que l'on applique en bloc le masque à une ligne. Si dans les positions qui correspondent aux caractères non blancs du masque, les caractères de la ligne et du masque sont identiques alors la recherche est positive.

Le caractère spécial est dit caractère de remplissage ("FILLER") et peut être changé en tout autre caractère.

| Exemple : Positions | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Soit la ligne: | A | B | C | D | E | F | G | H | I | J | |
| Avec ce Masque | | | | D | | F | | | | | il y a correspondance |
| avec cet autre masque | | A | C | D | E | | | | I | J | il n'y a pas correspondance |

## 4.5 REMPLACEMENT

Les modifications sur les éléments d'un fichier peuvent être assimilées à des remplacements, toutefois, il faut considérer deux niveaux: celui de PL ou celui de PC.

### 4.5.1 Remplacement au niveau de la ligne.

Nous pouvons d'ores et déjà définir pour cette fonction des primitives puisque la définition de la fonction POSITION permet d'appliquer un traitement particulier à n'importe quelle ligne ou partie de ligne.

Définissons la forme générale de la fonction 'remplacer' :

RPL ligne

- Si l'on considère que la ligne courante intervient dans le remplacement, alors l'opération est une véritable substitution par la ligne donnée.

- Si l'on considère que la ligne courante n'intervient pas dans le remplacement alors l'opération se traduit par une insertion.

- Si le paramètre de la fonction est une ligne vide, l'opération consiste en la suppression de la ligne courante.

### 4.5.2 Remplacement au niveau du caractère

Cette fonction permet d'agir sur les caractères de la ligne courante. Sa forme générale est :

RPC longueur, chaîne

Supposons que PC repère un caractère particulier de la ligne courante, cette fonction a pour effet de définir la chaîne à modifier: Cette chaîne commence au PC courant, et sa longueur est donnée en paramètre . On remplace alors cette chaîne par celle qui est donnée.

- Si la longueur est nulle, la chaîne donnée est insérée, à droite du caractère courant. Cela suppose que PC prend la valeur 0(insertion en début de ligne).

- Si la chaîne donnée est la chaîne vide, cela correspond à la suppression d'un certain nombre de caractères dans la ligne courante. Le reste de la ligne vers la droite est automatiquement retassé vers la gauche.

4.5.3 Remplacement avec masque

Nous retrouvons ici le concept de caractère de remplissage:

RPM M, mode

Le masque M est appliqué en bloc sur la ligne courante à partir du premier caractère ou à partir de PC selon le "mode". Les caractères de la ligne qui se trouvent en correspondance avec un caractère du masque différent du caractère de remplissage, sont remplacés par le caractère qui leur correspond.

| | | |
|---|---|---|
| Exemple : | $S^t$ la ligne | A B C D E F G H |
| | et le masque | Z A X E J K |
| | le résultat est | Z A X D E F J K |

4.6 CONTROLE

Le contrôle de l'édition revient à examiner et modifier le contenu des registres de description: Nom du fichier, type, longueur, marges d'édition etc...

Les instructions de l'Unité Centrale et l'écriture au terminal permettent d'imprimer la valeur d'un paramètre de description. Pour la modification, il faut utiliser une instruction de l'unité de contrôle.

La sauvegarde de l'état du fichier et celle de l'état du descripteur peuvent être demandées par l'utilisateur.

A cet effet, deux instructions sont prévues et en conjonction avec l'instruction Halt de l'unité centrale on peut:

- Soit arrêter l'édition en sauvegardant le fichier et le descripteur

- Soit sauvegarder simplement le fichier ou le descripteur sans arrêter l'édition.

- Soit arrêter l'édition sans sauvegarder les modifications faites.

## 4.7 GESTION DU FICHIER

La gestion du fichier doit se placer à 2 niveaux: au niveau du disque et au niveau de la mémoire centrale. Dans ce dernier cas la gestion est simple si le fichier est entièrement en mémoire. Ce n'est pas toujours le cas, et il faut alors se préoccuper du problème de temps de réponse. Dire qu'il est normal qu'un utilisateur éditant un gros fichier ait à attendre un peu plus que celui qui travaille sur un petit fichier, c'est dire qu'un nombre maximum de lignes doivent être installées en mémoire.

Il faut alors prévoir un mécanisme de libération de la place mémoire pour y amener les autres lignes.

En outre, les problèmes de sécurité imposent à l'éditeur de travailler sur une copie du fichier pour qu'en cas de panne de l'éditeur, du système, ou de la machine, on puisse au moins retrouver le fichier tel qu'il était avant l'édition.

Ainsi, à l'initialisation de l'éditeur, il faut prendre une copie du fichier à éditer et tant qu'à faire donner à cette copie une structure plus adaptée aux problèmes de transfert disque-mémoire.

Une structure de liste bi-directionnelle sur disque peut s'avérer efficace si le système dispose d'un gestion dynamique de l'espace disque.

Ainsi un élément, de longueur fixe, aurait en tête un pointeur (disque) vers l'élément suivant et un pointeur vers l'élément précédent.

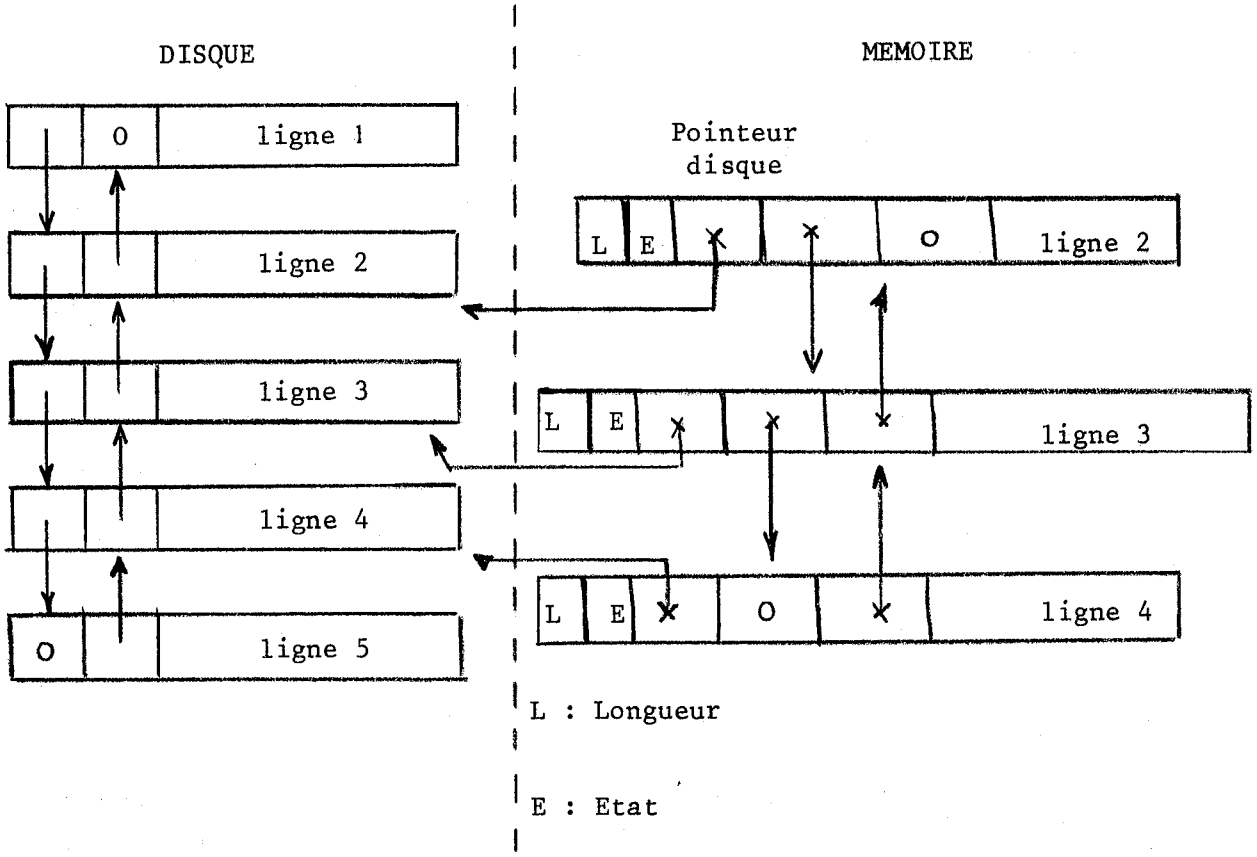
Chaque fois qu'une ligne, en mémoire, est modifiée on la marque en utilisant le champ "état" (cf.4-2). Dès que la fonction position demande une ligne qui n'est pas en mémoire, il faut trouver de la place: Seules les lignes modifiées sont sauvées sur disque et la place correspondante est rendue

à la gestion de la mémoire libre.

La sauvegarde consiste alors à ré-écrire sur disque, dans la structure imposée par le système, le fichier édité.

Ce module de gestion de fichier est le seul a être concerné par la gestion de fichier du système.

De nombreuses structures peuvent être employées selon les systèmes sans remettre en cause le fonctionnement global de la machine.




REMARQUE :

Nous évoquerons au chapitre VIII une méthode de "pagination" de l'espace mémoire virtuel de l'éditeur pour une implantation de la machine d'édition dans le cadre d'un système comme CMS fonctionnant sous CP.

## 5. PRE-TRAITEMENT

### 5.1 MODE SEQUENTIEL

Le jeu d'instruction de base de la machine constitue déjà un langage de commande élémentaire.

La syntaxe du langage est très simple:

- Un nom d'instruction (4 caractères au plus) suivi ou non de 2 paramètres au maximum, le plus souvent des noms symboliques de registres.

Le travail de pré-traitement, en mode séquentiel, consiste donc à reconnaître l'instruction, à isoler les paramètres avant de générer le code correspondant.

Le module de pré-traitement dispose :

- D'une table comportant les noms des différentes instructions permises, les codes opération correspondant et des indication sur les paramètres (nombre- type de chacun d'eux- instr.acceptable en mode seq. ou non).

- D'une table comportant les noms des registres de fonctionnement et de description avec les codes associés.

Dans toute chaîne émise par l'utilisateur on isole le nom d'une instruction et les paramètres dont les types servent à positionner des indicateurs spéciaux: P1 et P2 pour les instructions de base, P1, P2, P3 et P4 pour les micro-programmes. Dès que l'instruction est reconnue par son nom on dispose du code opération correspondant et l'on peut vérifier si l'état des indicateurs (P1 et P2) correspond aux informations sur les paramètres fournis par la table des instructions.

Lorsque cette opération s'avère positive, il faut reconnaître, au moyen de la table des registres si l'instruction complète est correcte ou non.

Toute instruction valide codée est mise, par le pré-traitement, dans le registre instruction de l'Unité Centrale pour y être exécutée.

REMARQUES :

Les instructions des tests et de sauts sont ignorées en mode

séquentiel. De même des instructions comme "CHN RN nombre" et "CHC RC chaîne" sont interdites en mode séquentiel; En effet, leur codification nécessite plusieurs mots. Pour charger un registre de fonctionnement on utilise l'instruction de lecture console (cf.3.1).

## 5.2 LANGAGE DE COMMUNICATION

Nous examinerons au chapitre suivant les caractéristiques des micro-programmes, la manière de les créer et la structure d'une bibliothèque.

En supposant l'existence d'un jeu de micro-programmes, le module de pré-traitement a pour tâche essentielle, l'isolement du nom du micro-programme la reconnaissance des paramètres (nombres-chaînes) et leur mise en place dans les registres pour assurer la correspondance avec les instructions du micro-programme.(cf.Chap.VII. §3).

En effet, dans un premier stade, le langage de communication correspond à un ensemble bien déterminé de micro-programmes qui constitue une sorte de "mémoire morte". Cela signifie que ces micro-programmes peuvent seulement être lus, donc sont non modifiables mais partageables.

L'importance de cette mémoire morte dépend du système utilisé, de la taille mémoire disponible et du rôle réel de l'éditeur en tant que composant du système.

Ainsi, lorsque l'ensemble des micro-programmes est fixé, il est facile d'établir des conventions de liaison entre le pré-traitement et l'appel d'un micro-programme: par exemple 1 paramètre numérique est mis automatiquement dans un registre numérique (le premier utilisé est R8), chaque chaîne est rangée dans A, puis B etc...

# ANNEXE CHAPITRE VI

## DESCRIPTION DES INSTRUCTIONS DE BASE

- Le format général (sur 1 mot) se présente comme suit :

| | | | |
|---|---|---|---|
| CODE OPERATION | 1er OPERANDE | TYPE 2nd OPERANDE | 2ième OPERANDE |

- Les conventions adoptées sont les suivantes :

- On désigne par "RF" un registre de fonctionnement quelconque (numérique ou chaîne) et par "RD" un registre de description quelconque.
"RN" désigne un registre de fonctionnement numérique
"RC" désigne un registre de fonctionnement de type chaîne.

- Les [ ] sont utilisés pour caractériser le contenu d'un registre.

- Les codes des instructions sont donnés en hexadécimal

- Chaque instruction est notée sous forme symbolique. Elle peut avoir au plus deux opérandes, et le blanc constitue le séparateur entre les opérandes et le code opération.

- L'opérande de type "adresse" correspond à l'étiquette d'une instruction dans un micro-programme (cf.Chap VII §4-2). Sous forme codée il matérialise un déplacement (nombre de mots) dans le corps de la bibliothèque (cf. Chap VII-§ 2.2.2).

# GROUPE 1 : UNITE CENTRALE

UC-1- NON OPERATION: | N O P | (code 0)

UC-2- ARRET DE LA MACHINE: | H A L T | (code 1)

UC-3- CHARGEMENT DES REGISTRES DE FONCTIONNEMENT:

UC.3.1. REGISTRE A REGISTRE:

| CHR | RF | RF / RD |
|---|---|---|

(code 02)

Le contenu du second opérande prend place dans le premier opérande.

Les deux registres doivent être de même type (numérique ou chaîne). Lorsque l'instruction porte sur des registres de type chaîne, la longueur de celle-ci est amenée automatiquement dans les registres numériques correspondant (cf.Chap VI-§2.2).

UC.3.2. CHARGEMENT D'UN NOMBRE:

| CHN | RN | nombre |
|---|---|---|

(code 03)

Le'nombre' donné est chargé dans le registre numérique.

En fait le nombre est rangé à "l'assemblage" dans un mot et c'est l'adresse de ce mot qui apparaît dans l'instruction codée.

UC.3.3. CHARGEMENT D'UNE CHAINE:

| CHC | RC | chaîne |
|---|---|---|

(code 04)

La 'chaîne' est rangée, à l'assemblage, dans une suite de mots

et l'adresse du premier mot figure dans l'instruction. La longueur de la chaîne est placée dans le premier octet de la suite de mots.

## UC.4. OPERATIONS ARITHMETIQUES

### UC.4.1. ADDITION

| ADR | RN1 | RN2 |
|---|---|---|

(code 05)

La somme des contenus de RN1 et RN2 prend place dans RN1 et l'indicateur arithmétique INAR est affecté de la manière suivante :

Si le résultat est :

- nul : INAR := 00
- positif: INAR := FO
- négatif: INAR := OF

### UC.4.2. SOUSTRACTION

| SOR | RN1 | RN2 |
|---|---|---|

(code 06)

La différence entre le contenu de $RN_1$ et $RN_2$ ([RN1] - [RN2]) est placée dans RN1.

L'indicateur INAR est modifié de la même façon que précédemment.

## UC.5. CONVERSION CHAINE-NOMBRE

| CVN | RN | RC |
|---|---|---|

(code 07)

La chaîne qui se trouve dans le second opérande est convertie en nombre (représentation binaire) et placée dans le premier opérande. Si la conversion ne peut se faire, l'instruction est ignorée.(équivalente à une instruction NOP).

## UC-6 CONVERSION NOMBRE-CHAINE

| CVC | RC | RN |
|---|---|---|

(code 08)

Le nombre qui se trouve dans RN est converti et la chaîne de caractères correspondante est placée dans RC.

## UC.7 COMPARAISONS

### UC.7.1. REGISTRES-REGISTRE

| CR | $RF_1$ | $RF_2$ |
|---|---|---|

(code 09)

Les deux registres doivent être de même type (numérique, chaîne).

On compare le contenu des deux registres et selon le résultat les indicateurs INAR et BOOL sont modifiés comme suit:

si $[RF_1]$ = $[RF_2]$ alors INAR := 00 ; BOOL := vrai

si $[RF_1]$ > $[RF_2]$ alors INAR := F0 ; BOOL := faux

si $[RF_1]$ < $[RF_2]$ alors INAR := 0F ; BOOL := faux

### UC.7.2. REGISTRE-NOMBRE

| CN | RN | nombre |
|---|---|---|

(code 0A)

On compare le "nombre" au contenu de RN. Les indicateurs INAR et BOOL sont modifiés comme précédemment.

### UC.7.3. REGISTRE-CHAINE

| CC | RC | chaîne |
|---|---|---|

(code 0B)

On compare la "chaîne" et le contenu de RC. Les indicateurs INAR et BOOL sont modifiés comme précédemment.

## UC.8. RUPTURES DE SEQUENCE

UC.8.1. SAUT INCONDITIONNEL: | SINC adresse | (code 0C)

UC.8.2. SAUT AVEC COMPTEUR: | RSNZ RN adresse | (code 0D)

On retranche 1 au contenu de RN. Si ce nouveau contenu n'est pas nul

l'exécution continue à l'instruction d'adresse donnée. Sinon on passe normalement à l'instruction suivante.

UC.8.3. SAUT SI BOOL= vrai : | SV adresse | (code OE)

UC.8.4. SAUT SI BOOL = faux : | SF adresse | (code OF)

## UC.9 APPEL DE MICRO-PROGRAMME (cf.Chap VII -§3-3)

| AMI nom | (code 10)

Le nom codé est donné sous forme d'adresse qui correspond au point d'entrée du micro-programme dans la bibliothèque.

- La valeur courante du compteur ordinal est rangée avant l'appel dans la pile d'interprétation.
- Le micro-programme est alors activé.

## UC.10 TEST D'INDICATEURS

| TI indicateur condition | (code 11)

L'indicateur est testé en fonction de la condition. Si la concordance est réalisée l'indicateur BOOL prend la valeur vrai sinon BOOL prend la valeur faux.

Les indicateurs possibles sont :

INAR ; NORD ; SUD ; EST ; OUEST ; P1 ; P2 ; P3 ; P4 ; ECONS

La condition est, au choix: EG(égal), NE(non égal), PG(plus grand), PP(plus petit), GE(plus grand ou égal), PE(plus petit ou égal), EF (en fonction) HF(hors fonction), NB(nombre), CH(chaîne).

## UC.11 PASSAGE EN MODE SEQUENTIEL | PMS | (code 12)

Le micro-programme en cours d'exécution est arrêté; la valeur du compteur ordinal est rangée dans la pile d'interprétation et l'on revient

à l'attente lecture au terminal (Chap.IV-§3-4)

UC-12- RETOUR DE/AU MICRO-PROGRAMME | RMI | (code 13)

Cette instruction provoque l'examen de la pile. Si la pile est vide on repasse en mode séquentiel sinon la valeur prise au sommet de la pile devient la nouvelle valeur du compteur ordinal (cf.Chap VII §3.3) et de ce fait disparaît de la pile.

UC.13. EXECUTION | EX RC | (code 14)

Le contenu du registre chaîne donné au paramètre est analysé et interprété comme une requête. La fonction demandée est exécutée comme si elle avait été reçue en mode séquentiel.

# G R O U P E 2

## CONTROLE DU PERIPHERIQUE

### CP.1 POSITION

CP.1.1. UNE LIGNE VERS LE BAS: | PLB | (code 40)

Cette instruction fait progresser d'une ligne dans le fichier: on ajoute 1 à PL, si la nouvelle valeur de PL dépasse la valeur de BAS(marge basse) l'indicateur SUD est mis "en fonction" et PL prend la valeur BAS.(blocage).

CP.1.2. UNE LIGNE VERS LE HAUT: | PLH | (code 41)

Cette instruction fait remonter d'une ligne dans le fichier: on retranche 1 à PL, si la nouvelle valeur de PL est inférieure à HAUT(marge haute) l'indicateur NORD est mis "en fonction" et PL prend la valeur HAUT.

CP.1.3. PC VERS LA DROITE | PCD RN | (code 42)

La contenu de RN est ajouté à PC. Si la nouvelle valeur de PC est supérieure à DROITE (marge droite) l'indicateur EST est mis "en fonction" tandis que PC prend la valeur DROITE.

CP.1.4.PC VERS LA GAUCHE | PCG RN | ( code 43)

Le contenu de RN est soustrait à PC. Si la nouvelle valeur de PC est inférieure à GAUCHE (marge gauche) l'indicateur OUEST est mis "en fonction" tandis que PC prend la valeur GAUCHE.

CP.1.5.RECHERCHE D'UNE CHAINE | POS RC mode | (code 44)

On recherche dans la ligne courante une occurrence de la chaîne contenue dans RC. Si la recherche est positive alors BOOL prend la valeur vrai sinon BOOL prend la valeur faux.

Le mode indique le type de recherche (cf.Chap.VI §4.4)

si le mode est
- absent la recherche est absolue de gauche à droite
- + ou - la recherche est relative à gauche ou à droite de PC.
- ! il s'agit du mode ancré(cf Chap.VI.4.4.2.4)

CP.1.6 RECHERCHE AVEC MASQUE | POM RC mode | (code 45)

Le contenu du registre chaîne est appliqué comme un masque à la ligne courante (cf. Chap.VI §4.4.2.6). S'il y a correspondance, BOOL prend la valeur vrai sinon BOOL prend la valeur faux.

- Si le mode est absent le masque est appliqué sur toute la ligne à partir du premier caractère.

- Si le mode est '!' le masque est appliqué à partir du PC.

## CP.2. REMPLACEMENT

CP.2.1 INSERTION D'UNE LIGNE | RPI RC | (code 46)

Le contenu du registre chaîne est inséré après la ligne courante.

CP.2.2. REMPLACEMENT DE LIGNE | RPL RC | (code 47)

La ligne courante est remplacée par le contenu du registre chaîne. Si RC est le registre 0(cf.Chap VI §2.2) la ligne courante est supprimée.

CP.2.3 REMPLACEMENT DE CARACTERES

| RPC RN RC | (code 48)

La chaîne de caractères commençant au PC courant( sur une longueur indiquée par RN) est remplacée par la chaîne contenue dans RC.

- Si RN est nul la chaîne est insérée à gauche de PC.

- Si RC est nul on supprime un nombre de caractères égal à celui pris dans RN.

CP.2.4. REMPLACEMENT AVEC MASQUE | RPM RC mode | (code 49)

Le contenu du registre chaîne agit comme un masque sur la ligne

courante (cf Chap VI §4-5-3). Le mode est le même que celui de l'instruction POS (CP-1.6).

## CP.3. CONTROLE

CP.3.1 CHARGEMENT LIGNE COURANTE: | CHLC RC RN | (code 4A)

Si RN est nul la ligne courante est amenée dans RC. Si RN n'est pas nul le chargement est fait à partir de la position courante de PC sur un nombre de caractères indiqué par le contenu de RN.

CP.3.2. CHARGER UN REGISTRE DE DESCRIPTION

| RRD RF RD | (code 4B)

Si RF et RD sont de même type (numérique, chaîne) le contenu de RF est placé dans RD. Sinon l'instruction est invalide.

CP.3.3. SAUVEGARDE DU DESCRIPTEUR.

| SAUD | (code 4C)

L'état actuel du descripteur est conservé sur disque.

CP.3.4. SAUVEGARDE DU FICHIER

| SAUF | (code 4D)

L'état actuel du fichier est sauvé sur disque; le descripteur n'est pas modifié.

# G R O U P E 3

## CONSOLE

### C.1. ECRITURE CONSOLE

| ECRC RF |
|---|

code 80

Le contenu du registre de fonctionnement est écrit au terminal.

Si RF est 1 registre numérique, le nombre qu'il contient est imprimé.

Si RF est un registre chaîne, le nombre de caractères à écrire se trouve dans le registre numérique correspondant.

### C.2. LECTURE CONSOLE.

| LIRC RF |
|---|

code 81

Une chaîne est lue à la console et rangée dans le registre de fonctionnement RF.

Si RF est un registre numérique et que la chaîne lue ne représente pas un nombre alors la chaîne est refusée.

Si RF est 1 registre chaîne la longueur de la chaîne lue est placée dans le registre numérique correspondant.

### C.3. CONSOLE IMPRESSION

| CHUT |
|---|

Code 82

Cette instruction affecte l'indicateur d'impression ECONS. S'il est hors-fonction, il est mis en fonction et vice-versa.

# CHAPITRE VII : LANGAGE DE COMMUNICATION

# CHAPITRE VII

# LANGAGE DE COMMUNICATION

## 1. NIVEAU DE LANGAGE

Que ce soit dans un éditeur par contexte ou plus généralement dans tout autre sous-environnement conversationnel, on dispose d'ordinaire d'une série de requêtes composant un langage de commande. Ces requêtes ne sont pas, en général, des commandes primitives, ce qui signifie qu'au niveau du traitement interne, les sous-programmes correspondant s'enchevêtrent pour former une masse d'instructions difficilement modifiables.

De plus, ces requêtes ne permettent pas de descendre au dessous d'un certain niveau. Ainsi dans un éditeur, le langage de commande ne permet pas l'accès à certains paramètres de fonctionnement ce qui peut s'avérer désagréable pour une application particulière.

Pour ces raisons, il semble souhaitable de transformer l'éditeur en un véritable système conversationnel pour le traitement des caractères.

Nous allons définir trois niveaux pour le langage de commande: (Fig.14):

- Le niveau du langage de base, "câblé" en quelque sorte: il se compose d'instructions.

- Le niveau du langage de communication composé de requêtes, et permettant un emploi plus aisé de l'éditeur en s'affranchissant des contraintes du langage de base.

- Le niveau du macro-langage composé de commandes définies par l'utilisateur pour ses applications particulières.

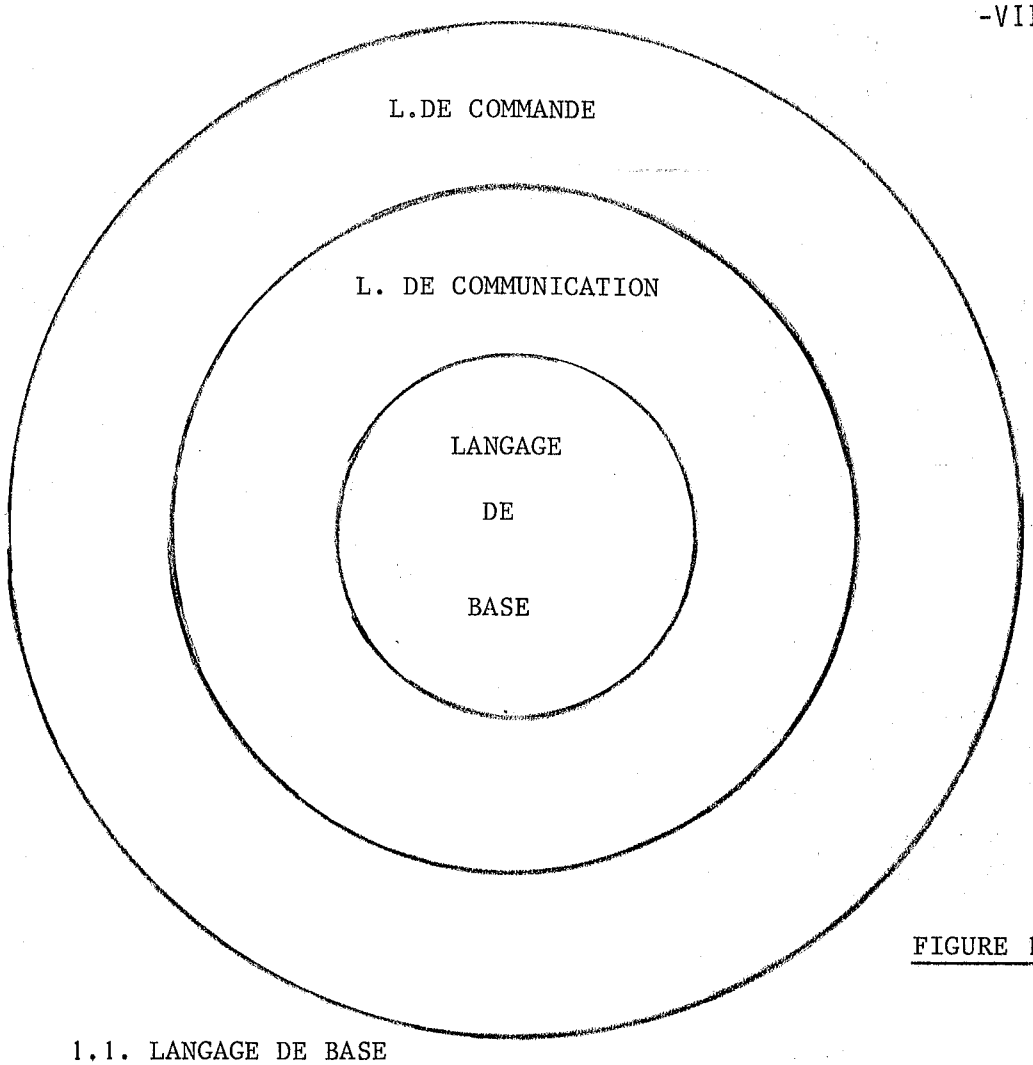



FIGURE 14

1.1. LANGAGE DE BASE

Les instructions qui composent ce langage correspondent à des opérations élémentaires, véritables primitives tant sur le plan de la manipulation de fichier que sur le plan du traitement de caractères.

Nous avons un exemple de langage de base avec le jeu d'instructions de la machine d'édition.(cf. Chapitre précédent). L'effet de ces instructions est volontairement limité pour que chacune d'elles soit une véritable primitive.

Il est possible de réaliser une édition en n'utilisant que ces seules instructions: c'est un travail long et fastidieux; aussi l'on éprouve rapidement le besoin de définir de petits programmes d'édition en groupant les instructions les plus utilisées.

Il faut remarquer également qu'un utilisateur de l'éditeur n'est pas forcément un programmeur et que les contraintes liées à l'emploi des registres peuvent le dérouter. Il faut donc lui fournir un langage beaucoup moins hermétique

et d'un emploi plus facile: c'est le but du langage de communication.

### 1.2 LANGAGE DE COMMUNICATION

L'objectif de ce langage est double:

- s'affranchir des contraintes liées à une machine
- s'exprimer en des termes plus proches du langage de l'utilisateur.

Cette idée est à la base des langages de programmation évolués, et si nous l'adoptons à notre machine d'édition, nous devons définir un langage de commande avec lequel l'utilisateur peut décrire simplement les opérations qu'il veut réaliser sur son fichier(sans se préoccuper des contraintes énoncées au paragraphe précédent).

On définit ainsi un jeu de requêtes analogues à celles d'EDIT, de C.M.S. Chaque requête s'exprime au moyen d'instructions du langage de base ou au moyen d'autres requêtes.(Voir en fin de chapitre la description du langage d'EDIT).

Un tel langage est micro-programmé, chaque requête se traduit par un micro-programme. Ces micro-programmes sont groupés dans une bibliothèque et leur utilisation fait fonctionner la machine d'édition en mode programmé.

Toutefois, ce jeu de requêtes est figé et correspond aux opérations classiques de création, de modification de fichiers.

### 1.3. EXTENSIONS

Nous voulons traiter, avec l'éditeur un plus grand nombre d'applications; Pour cela l'utilisateur doit pouvoir combiner les instructions de base et les requêtes pour créer ses propres micro-programmes donc ses propres commandes de manipulation de chaînes de caractères.

En fait, il s'agit de rendre conversationnel le mécanisme utilisé pour créer le jeu de micro-programmes du langage de oommunication.

C'est un mécanisme semblable à celui utilisé dans un macro-assembleur

avec la possibilité supplémentaire d'utiliser les facilités de l'éditeur pour la mise en forme et la correction des futurs micro-programmes.

Au gré de l'utilisateur, les micro-programmes qu'il composera et "assemblera" seront, une fois utilisés, perdus ou conservés pour créer de véritables bibliothèques privées ayant ou non priorité sur la bibliothèque standard.

Une autre façon d'étendre le langage de commande est de disposer d'un "macro-langage" indépendant du sous-environnement conversationnel. Ceci est réalisé en partie sous le système C.M.S avec le langage EXEC.

Un tel langage disposant des caractéristiques de base d'un langage de programmation classique, constitue une suite de ciment pour les requêtes du langage de commande du sous-environnement conversationnel. Ces requêtes sont traitées comme des appels de fonctions qui donnent en retour une certaine valeur. (Code erreur, chaîne à imprimer etc...) La conception, la compilation d'un tel langage est en cours d'étude dans le cadre d'un projet de D.E.A. [L12] .

## 2. CARACTERISTIQUES DES MICRO-PROGRAMMES

### 2.1 STRUCTURE D'UN MICRO-PROGRAMME

Un micro-programme correspond, à travers son nom, à une série d'instructions élémentaires exécutables directement. Ceci soulève deux problèmes:

1) Les données nécessaires à l'exécution de ce micro-programme doivent être placées avant l'appel, dans des registres de fonctionnement ou bien doivent résider, de manière statique, dans le micro-programme.

2) Un micro-programme doit pouvoir être appelé aussi bien à partir du terminal, qu'à partir d'un autre micro-programme, ceci en utilisant le même mécanisme.

La mise en place des paramètres dans les registres se fait au

pré-traitement. Le premier problème est ainsi résolu pour le mode séquentiel. Pour le mode programmé c'est-à-dire lorsqu'un micro-programme doit appeler un autre micro-programme, le traducteur prévoit, dans les instructions générées pour l'appel, le chargement des paramètres.

Si l'on considère maintenant tous les micro-programmes comme des "sous-programmes", grâce au mécanisme d'interprétation de l'Unité Centrale, on peut résoudre le second problème. Nous examinerons ce mécanisme en détails en étudiant l'exécution des micro-programmes (cf.§3);retenons simplement le fait que chaque micro-programme doit se terminer par une instruction de retour(RMI) laquelle permet de revenir au niveau d'appel, c'est-à-dire soit au mode séquentiel, soit au micro-programme appelant.

Ainsi, un micro-programme ne peut être modifié par son exécution. Il peut ainsi être partagé par plusieurs utilisateurs ce qui concorde avec la ré-entrance globale de l'éditeur.

## 2.2. STRUCTURE D'UNE BIBLIOTHEQUE

Une bibliothèque de micro-programme est une véritable mémoire morte qui contient des instructions codées correspondant à une série de micro-programmes.

Elle se compose de deux parties:

1. Un catalogue répertoriant les différents composants de la bibliothèque.

2. Les instructions, sous forme codée, des micro-programmes constituant le corps de la bibliothèque.

### 2.2.1. Le catalogue

Il comporte une première partie qui permet d'identifier la bibliothèque (son nom, le nombre de composants, l'emplacement du corps de la bibliothèque etc...).
La seconde partie du catalogue est l'index de tous les micro-programmes. Pour chacun d'eux on trouve un en-tête qui contient diverses informations, à savoir:

- Le nom du micro-programme
- Des indications sur les paramètres qu'il accepte
- L'emplacement des instructions correspondantes dans le corps de la bibliothèque.

Chaque en-tête de micro-programme a une longueur fixe, ce qui facilite la consultation de l'index.

### 2.2.2 Le corps de la bibliothèque

Les instructions codées chacune sur un mot, se suivent sans aucune séparation physique. On peut ainsi avoir dans la bibliothèque standard des instructions communes à tous les micro-programmes.

Lorsqu'une "adresse" apparaît dans une instruction, elle figure sous forme d'un déplacement par rapport à la première instruction du corps de bibliothèque. Ceci nous permet de conserver facilement, sous forme de fichiers, les bibliothèques, et de les placer n'importe où en mémoire selon la place disponible. A l'interprétation des instructions, chaque déplacement est traduit en fonction de l'adresse d'implantation de la bibliothèque.

## 2.3. LIAISONS ENTRE LES MICRO-PROGRAMMES.

Ces liaisons peuvent être soit dynamiques soit statiques.

En effet, grâce à l'instruction d'appel de micro-programme("AMI") on obtient dynamiquement l'appel d'un micro-programme donné. De plus, grâce à la structure de "mémoire morte", du corps de bibliothèque on autorise des instructions de saut, d'un micro-programme à un autre; ceci est fait de manière statique et dépend du mode de génération des micro-programmes. Le rôle du traducteur est donc d'accepter des symboles "externes" correspondant aux points d'entrée d'autres micro-programmes.

A titre d'exemple, une partie de bibliothèque est donnée (sous forme symbolique) à la fin du chapitre. Elle correspond à un jeu de requêtes équivalentes à celles de l'éditeur de C.M.S.

## 3. EXECUTION DES MICRO-PROGRAMMES

### 3.1. PRE-TRAITEMENT

En mode séquentiel, la forme générale d'un appel de micro-programme est la suivante:

Nom de micro-prog. paramètre1 paramètre2 paramètre3 paramètre4

Le nombre des paramètres est limité à 4 mais un micro-programme peut demander des paramètres complémentaires par l'instruction de lecture à la console.

D'une manière classique, le pré-traitement analyse la chaîne tapée, isole le nom, les paramètres, et positionne les indicateurs correspondant (P1, P2, P3 et P4) selon le type de chaque paramètre.

Le nom est comparé à ceux du catalogue de la bibliothèque. Lorsqu'une correspondance est trouvée, la validité de l'appel est vérifié par comparaison des indicateurs P1, P2, P3 et P4 et des informations sur les paramètres, informations qui sont permanentes dans le catalogue.

Ces paramètres sont placés dans les registres correspondant tandis que l'adresse de la première instruction du micro-programme est calculée et placée dans le compteur ordinal. On rend le contrôle au superviseur/interprète de l'unité centrale qui fonctionne alors en mode programmé.

### 3.2. MODE PROGRAMME

Il s'agit d'enchaîner une série d'instructions de manière classique.

Le compteur ordinal progresse normalement après chaque instruction sauf s'il s'agit d'une instruction de saut. Les "adresses" dans les instructions sont traduites en adresses réelles absolues.

Pour gérer les deux modes de fonctionnement et les micro-programmes nous disposons des instructions suivantes, qui demandent un traitement particulier.

- Appel de micro-programme "AMI nom"
- Passage en mode séquentiel "PMS"
- Retour de micro-programme "RMI"

De plus, il existe dans la zone de fonctionnement une pile d'interprétation qui sert à gérer les appels successifs de micro-programmes.

### 3.3. APPEL DE MICRO-PROGRAMME:

Chaque instruction d'appel "AMI" rencontrée, provoque:

- L'empilage de la valeur courante du compteur ordinal.
- Le transfert du contrôle au micro-programme appelé sans quitter le mode programmé.

La pile est limitée en taille, son débordement provoque l'arrêt du micro-programme en cours et le passage au niveau inférieur d'appel.

Chaque micro-programme se termine par une instruction de retour (RMI) qui provoque:

- Un test sur l'état de la pile
- Si elle est vide on repasse en mode séquentiel
- Sinon, la valeur prise au sommet de la pile devient la nouvelle valeur du compteur ordinal et le processus d'interprétation continue.

### 3.4. INTERACTION AU NIVEAU DE L'EXECUTION.

En mode programmé, et en cours d'exécution d'un micro-programme la rencontre de l'instruction "PMS" provoque:

- L'empilage de la valeur courante du compteur ordinal
- Le passage en mode séquentiel avec possibilité pour l'utilisateur de contrôler l'état de son fichier, les valeurs des paramètres de description ou de fonctionnement.
- La reprise de l'exécution se fait, là encore, par l'instruction RMI.

Ceci permet une interaction plus grande au niveau de l'exécution des opérations d'édition et facilite la mise au point de micro-programmes particuliers.

## 4. GENERATION DE MICRO-PROGRAMMES

### 4.1 BUTS

Nous voulons que l'éditeur s'adapte à un grand nombre d'applications particulières pour le traitement des caractères; nous avons de ce fait construit un mécanisme de définition et de gestion de micro-programmes.

Chaque utilisateur peut composer les instructions de base et les requêtes pour créer et conserver ses propres micro-programmes.

Puisqu'il s'agit d'une machine d'édition, le langage utilisé pour la création de micro-programme est plus proche d'un langage d'assemblage que d'un langage évolué. Néanmoins la présence du langage de communication permet de s'éloigner des contraintes liées à la machine.

La création d'un micro-programme nécessite la création d'un fichier dont les éléments sont des instructions de l'éditeur. Cela veut dire que l'on peut utiliser les facilités offertes par ce même éditeur pour la création et la modification du fichier source du micro-programme. Il est donc nécessaire de définir la structure des éléments d'un tel fichier, ce qui revient à définir le "langage d'assemblage".

### 4.2 LANGAGE D'ASSEMBLAGE

Le format d'une"instruction" du langage est imposé par les instructions de base et le langage de communication:

- 1 instruction de base ou une requête par ligne
- Format d'une ligne: 3 zones (plus une zone de commentaires)

Etiquette opération opérandes (4 au maximum)

L'étiquette permet évidemment l'écriture des instructions de saut.

*EXEMPLE :*

Rechercher dans le fichier, à partir de la ligne courante, la chaîne 'ABC' et supprimer toutes les lignes qui contiennent cette chaîne.

| étiquette | opération | opérandes | |
|---|---|---|---|
| CHER | NEXT | /ABC/ | recherche 'ABC' |
| | TI | SUD EF | Test fin fichier |
| | SV | FIN | |
| | DELETE | | supprimer |
| | SINC | CHER | saut inconditionnel |
| FIN | RMI | | retour de m.programme |

Les requêtes utilisées ici sont celles qui correspondent à l'éditeur de C.M.S. et qui sont micro-programmées.

Des instructions comme CHN RN nombre et CHC RC chaîne s'écrivent symboliquement:

```
CHN    RN   12
CHC    RC  /ABCD/
```

Cela n'est qu'une facilité d'écriture, en fait les constantes "nombre" ou "chaîne" sont traitées comme les "littéraux" d'un assembleur classique: On génère la valeur de la constante dans un ou plusieurs mots en fin de micro-programme et on place dans l'instruction correspondante l'adresse du premier mot.

### 4.3 TRADUCTION

Le traducteur ou "micro-assembleur" est un module ajouté à la machine d'édition. Son rôle se décompose en deux parties :

- Création d'un en-tête de micro-programme avec le nom et les indications sur les paramètres.

- Traduction des lignes du fichier source en générant plusieurs instructions codées constituant le corps du micro-programme.

L'en-tête et le corps du micro-programme sont incorporés dans une bibliothèque privée.

La création d'un micro-programme se fait en plusieurs étapes:

1. Appel de l'éditeur pour créer normalement un fichier dont le nom est celui du micro-programme et dont le type est particulier.

Les lignes sont entrées une à une, chacune ayant le format défini au paragraphe précédent. Au fur et à mesure de la création, on peut modifier ces lignes grâce aux requêtes d'édition.

2. Sans quitter l'environnement d'édition, appel du module de traduction en précisant alors le nombre et le type des paramètres du micro-programme: il faut fournir, en quelque sorte, un prototype d'appel.

3. Le module de traduction considère chaque ligne du fichier comme une instruction du langage d'assemblage et génère le ou les instructions de base correspondantes. Cette génération peut être facilitée par l'appel de certaines parties du pré-traitement.

Chaque ligne incorrecte est signalée à l'utilisateur qui peut faire les corrections nécessaires avant de re-demander la traduction.

4. Lorsque le code généré est correct et complet, il est conservé et placé dans une bibliothèque.

Quant au fichier source sa sauvegarde reste à la charge de l'utilisateur.

5. Le micro-programme est alors disponible sous l'éditeur. Son appel doit être conforme aux indications données lors de la création (étape n°2)

Appel dynamique de Micro-programme:

L'appel d'un micro-programme par un autre micro-programme peut se faire de deux façons:

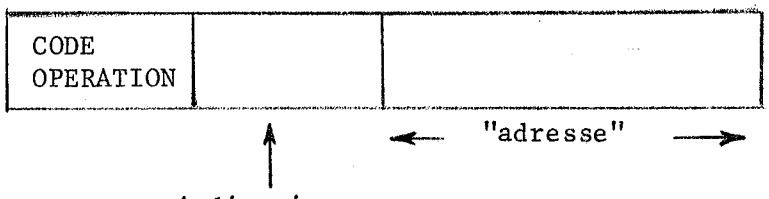
1/ Directement par l'instruction "AMI Nom" si les paramètres sont déjà placés dans les registres.

2/ Par un appel symbolique comme s'il venait du terminal: nom par1 par2 etc...

Il suffit alors de se ramener au premier cas, en générant les instructions qui permettent de charger les paramètres dans les registres ainsi qu'une instruction "AMI nom".

Dans les deux cas, il faut déterminer si le "nom" correspond à un micro-programme public ou privé. Dans cette dernière éventualité, on indique dans le code de l'instruction AMI que l'adresse est à prendre dans la bibliothèque privée (v.fig.15).

Codage
Instruction "AMI"




indication
sur la bibliothèque
- si 0 bibl.standard
- sinon bibl.privée (éventuellement n°, s'il y en a plusieurs)

FIGURE 15

### 4.4. BIBLIOTHEQUE PRIVEE

La bibliothèque privée appartient à l'ensemble des fichiers d'un utilisateur dès le premier emploi du traducteur. Elle a la même structure que la bibliothèque standard, ce qui permet d'établir des liens entre les micro-programmes privées. La bibliothèque privée peut avoir priorité sur la bibliothèque publique ou peut carrément la remplacer. Le choix entre ces éventualités est déterminé par la valeur d'un paramètre supplémentaire fourni soit lors de l'appel de l'éditeur, soit en cours d'édition.

Il est possible aussi de définir temporairement un ensemble de micro-programmes qui seront abandonnés dès la fin de l'édition en cours.

# ANNEXE AU CHAPITRE VII

## EXEMPLES DE MICRO-PROGRAMMES

Nous donnons ici, à titre d'exemple, une partie de bibliothèque de micro-programmes. Les requêtes qui sont représentées par ces micro-programmes sont prises parmi celles de l'éditeur de CMS-EDIT.

Les micro-programmes sont donnés sous forme symbolique et constituent une sorte de "mémoire morte". Chaque point d'entrée, c'est-à-dire chaque début de requête micro-programmée **est signalé par une "*"**.

Conventions d'appel :

Pour un micro-programme on autorise 4 paramètres au plus. Le type d'un paramètre (nombre ou chaîne)est caractérisé par l'un des indicateurs P1, P2, P3 ou P4.

Tout paramètre de type nombre est rangé, à l'appel du micro-programme, dans un registre numérique, en commençant par R8, puis R7 etc...

Tout paramètre de type chaîne est rangé dans un registre chaîne en commençant par A puis B etc...

```
        NEXT1   PLB
                TI      SUD     EF
                SV      CSUD
                RMI
        UP1     PLH
                TI      NORD    EF
                SV      CNORD
                RMI
*       NEXT    AMI     NEX1
                TI      P1      CH
                SV      NEXC
                RSNZ    R8      NEXT      NEXT {n / /chaîne/}
                SINC    IMPR
        NEXC    POS     A
                SV      IMPR
                SINC    NEXT
*       UP      AMI     UP1
                TI      P1      CH
                SV      UPC               UP {n / /chaîne/}
                RSNZ    R8      UP
                SINC    IMPR
        UPC     POS     A
                SF      UP
        IMPR    CHLC    B
                ECRC    B
                RMI
*       PRINT   AMI     IMPR
                TI      P1      CH
                SV      PRC
                AMI     NEX1              PRINT {n / /chaîne/}
                RSNZ    R8      PRINT
                AMI     UP1
        PRC     POS     A
                SV      IMPR
```

```
          AMI     NEX1
          SINC    PRINT
*   DEL   RPL     RO
          TI      P1      CH
          SV      DELC
          RSNZ    R8      DEL     DELETE { n / /chaîne/ }
          RMI
    DELC  POS     A
          SF      DEL
          RMI
    CSUD  CHC     D       'SUD'
          ECRC    D
          RMI
    CNORD CHC     D       'NORD'
          ECRC    D
          RMI
*   INPUT CHC     A       'INPUT'
          ECRC    A
    INP1  LIRC    A
          CR      R1      RO      INPUT
          SV      LIVI
          RPI     A
          SINC    INP1
    LIVI  CHC     A       'EDIT'
          ECRC    A
          RMI
*   GOTO  TI      P1      CH
          SV      GOTC
          CHR     R6      PL
          SOR     R6      R8
          SV      IMPR
          TI      INAR    PG      GOTO { n / /chaîne/ }
          SF      GNEX
          CHR     R8      R6
          SINC    UP
    GNEX  CHR     R6      PL
          SOR     R8      R6
```

```
        SINC    NEXT
GOTC    AMI     TOP
        SINC    NEXT
*  TOP  PLH
        TI      NORD    EF      TOP
        SF      TOP
        RMI
*  BOTTOM PLB
        TI      SUD     EF      BOTTOM
        SF      BOTTOM
        RMI
*  CHANG CHR    R4      R7
   OKUR POS     A       +
        SF      NOCH            CHANGE /ch1/ch2/ n1 n2
        RPC     R1      B
        RSNZ    R7      OKUR
   NOCH CHR     R7      R4
        AMI     NEX1
        RSNZ    R8      OKUR
        AMI     IMPR
        RMI                     REPLACE ligne
*  REPL RPL     A
        RMI
*  FIND AMI     NEX1
        POM     A
        SV      IMPR            FIND    masque
        SINC    FIND
```

# CHAPITRE VIII : EXPERIMENTATION ET REALISATIONS

# CHAPITRE VIII

# EXPERIMENTATION ET REALISATIONS

## 1. EXPERIMENTATION

Après avoir tiré les leçons des réalisations de la première partie (Editeur graphique et Macro-éditeur) et avant d'établir la description complète de la machine d'édition, nous avons réalisé un certain nombre d'expériences.

### 1.1. ESSAIS DE STRUCTURES

Grâce au langage SNOBOL4 [L3] qui permet de définir et d'utiliser de nouvelles structures d'éléments et de nombreuses opérations sur les chaînes de caractères nous avons pu étudier diverses façons de structurer les éléments d'un fichier sur disque et en mémoire.

### 1.2. LANGAGE DE COMMANDE.

La maintenance de l'éditeur de C.M.S. "EDIT" nous a mis en contact direct avec les différents types d'utilisateurs et leurs problèmes nous ont permis de dégager les caractéristiques "idéales" d'un langage d'édition.

Grâce au macro-éditeur nous disposions, en outre, d'un outil souple et relativement puissant pour tester de nouvelles requêtes et constater le besoin de primitives dans les fonctions d'édition.

### 1.3. MACHINE EXPERIMENTALE

A l'aide du langage d'assemblage des IBM/360 et des facilités

du système CP/C.M.S., nous avons programmé une machine expérimentale d'édition. Ceci avait surtout pour but de tester les concepts de base d'une telle forme d'éditeur et de mesurer l'effort de programmation nécessaire à la réalisation d'une machine complète.

Prévue pour éditer des fichiers dont les éléments sur disque, sont doublement chaînés (donc différents de ceux de C.M.S.), cette machine se compose de 5 modules fonctionnels (ré-entrants) et de 3 zones partageables.

### 1.3.1. Modules et zones partageables.

"Unité Centrale": Module principal, il reçoit le contrôle initialise la machine, supervise son fonctionnement.

"Entrées/sorties Console": Réalise les lectures et écritures au terminal (ici un IBM 2741).

"Pré-traitement": Analyse la chaîne émise par l'utilisateur, génère le code correspondant (instruction simple ou micro-programme).

"Contrôle du périphérique": Exécute les opérations d'édition.

"Gestion du fichier": installe en mémoire les lignes à examiner ou à modifier.

- Zone de fonctionnement: comporte registres (numériques ou chaînes) indicateurs, zone de travail etc...

- Zone de description: contient tous les paramètres liés au fichier et à l'édition en cours.

- Zone fichier: les lignes y sont installées après suppression des blancs en fin de ligne. La structure adoptée est celle décrite au Chap.VI.3.

### 1.3.2. Instructions

L'unité centrale peut fonctionner en mode séquentiel ou en mode programmé ce qui permet l'utilisation de micro-programmes.

Les instructions de la machine expérimentale sont de longueur variable (1,2 ou 3 mots). Le jeu d'instructions, sans être aussi complet que

celui décrit plus haut, comporte:

- pour l'unité centrale, des instructions de chargement et de comparaison des registres de fonctionnement, des instructions de rupture de séquence et de contrôle (arrêt-mode séquentiel-mode programmé).

- pour l'unité de contrôle, la fonction "POSITION" sous sa forme complète telle qu'elle est décrite au Chap.II.4.4. Elle permet de déplacer à volonté un pointeur de ligne (PL) et un pointeur de caractère (PC). Les appels valides d'une telle fonction sont très nombreux. De ce fait, la programmation du pré-traitement (reconnaissance et codage) et de l'interprétation nécessite un grand nombre d'instructions. Nous fûmes donc amenés par la suite à séparer en deux groupes distincts les fonctions concernant PL et celles concernant PC. D'autres instructions sont utilisables: chargement de la ligne courante dans un registre chaîne, indication du PC à l'impression.

- pour le terminal, les opérations de lecture ou d'écriture avec les registres chaînes pour paramètres.

### 1.3.3. Fonctionnement

Après acquisition de 3 zones de mémoire pour implanter les zones partageables, l'unité centrale appelle le module de gestion du fichier pour installer en mémoire les premières lignes et se met en attente lecture au terminal. Dès qu'une chaîne est reçue, le module de pré-traitement l'analyse. Il détermine si le nom de la fonction demandée correspond à une instruction de base ou à un micro-programme. Un appel incorrect est indiqué à l'utilisateur.

L'unité centrale reprend alors le contrôle avec une ou plusieurs instructions valides à exécuter.

Lorsqu'un appel de la fonction POSITION demande l'accès à une ligne non en mémoire, le module de gestion de fichier est activé. Il assure la sauvegarde sur disque des lignes modifiées qui résident à ce moment-là en mémoire, puis amène les lignes demandées. La taille de la zone fichier est un paramètre de la machine expérimentale, elle peut être modifiée pour réaliser différentes mesures mais la taille des fichiers à éditer est illimitée.

Pour éditer effectivement un fichier le besoin de micro-programmes se fait vite ressentir:

- Enchaîner un appel à la fonction POSITION et l'impression de la nouvelle ligne courante.

- Imprimer n lignes.

## 2. AVANTAGES ET INCONVENIENTS DE LA MACHINE EXPERIMENTALE.

Le travail fourni pour programmer cette machine expérimentale volontairement incomplète, nous a permis de mettre à jour les points suivants:

- La conception de modules facilite la construction et l'écriture des programmes. Du fait de l'existence des zones partageables la mise au point est délicate mais pour cela, la commande de C.M.S. "Debug" s'est avérée très utile. Cette commande comporte néanmoins quelques carences et nous pensons que pour la réalisation d'une machine complète l'utilisation d'un moniteur de mise au point [S10] permettra d'accélérer la correction des erreurs et par là même augmenter la fiabilité de notre machine.

- Le processus de codage et d'interprétation ne semble pas affecter le temps de réponse. Cependant, il nous semble nécessaire d'uniformiser le format des instructions (prendre un format fixe) et simplifier les paramètres. Une fonction primitive qui n'admet que certains paramètres (p.exple des registres) est plus facile à mettre au point qu'une fonction comme POSITION.

- Le mécanisme des micro-programmes est très agréable tant pour le programmeur qui, après l'implantation et la mise au point de l'éditeur de base, compose à sa guise des fonctions évoluées, que pour l'utilisateur qui, s'affranchissant de certaines contraintes(registres etc...) a toujours la possibilité d'accéder à des variables internes à l'éditeur.

- La structure adoptée sur disque pour le fichier est surtout pratique lorsqu'il s'agit d'accéder à l'information. Lorsqu'il faut ranger de

nouvelles lignes ou en supprimer, il est nécessaire d'avoir une gestion dynamique de l'espace disque.
Quant à la structure en mémoire, la suppression des blancs en fin de ligne permet de gagner en pourcentage de place appréciable (environ 30 % pour 1 fichier image de cartes) mais lors d'une recherche par contexte, il est indispensable de remettre chaque ligne dans son état initial. De plus, la modification d'une ligne peut entraîner une augmentation de sa longueur, il faut donc lui trouver une autre place. Tout ceci implique, là aussi, une gestion dynamique de la mémoire.

Ces problèmes de structure sont intimement liés au système sous lequel fonctionne l'éditeur et plus particulièrement à la gestion de fichiers.

La présence de registres numériques et surtout de type "chaîne" doit être généralisée à la majorité des paramètres de description. Chaque registre devient alors un identificateur pour le langage d'édition, ce qui permet à l'utilisateur de contrôler directement certains paramètres.

## 3. REALISATION D'UNE MACHINE COMPLETE D'EDITION.

Après avoir défini la machine d'édition complète nous avons entrepris sa réalisation. La programmation des différents modules et leur mise au point fût donnée en projet de D.E.A. [E9] .

Cette réalisation se décompose en plusieurs étapes:

1/ Choisir un cadre d'implantation de la machine. Dans un premier temps c'est une machine virtuelle sous le contrôle de CP/67.

2/ Programmer les modules indépendants de toute gestion de fichier comme le pré-traitement, l'unité centrale, et les entrées/sorties au terminal. Assurer leur mise au point séparément.

3/ Programmer les autres modules en adoptant des solutions originales: par exemple pagination de l'espace mémoire pour le fichier, en utilisant une nouvelle méthode d'accès fournie par CP/67. [S11] .

4/Développer parallèlement une série de micro-programmes ainsi qu'un micro-assembleur. Ce dernier peut d'ailleurs être programmé en langage évolué (SNOBOL4 par exemple).

5/ Assurer la mise au point de l'éditeur de base puis compléter cet éditeur avec des extensions diverses: langage- nouvelles fonctions- bibliothèques privées etc...

# C O N C L U S I O N

Dans un système conversationnel l'importance de l'éditeur est indéniable car à travers les commandes fournies pour gérer ses fichiers l'utilisateur peut juger des performances du système.

Il faut remarquer que l'éditeur s'adresse principalement à deux types d'utilisateurs:

1. Le non-spécialiste qui veut réaliser des opérations classiques: suppression- insertions- corrections...

2. Le spécialiste ou l'ingénieur système qui demande à l'éditeur un peu plus: manipulation de gros fichiers ou de fichiers particuliers, fonctions plus complexes, etc...

Ainsi le langage de commande doit être assez simple pour satisfaire les premiers, et assez complet pour résoudre les problèmes des seconds.

Pour concevoir un éditeur, il faut tenir compte de cette dualité. Dans mon travail, j'ai surtout cherché à dégager de nouveaux types d'éditeurs qui peuvent satisfaire les utilisateurs les plus exigeants sans rebuter ceux qui se contentent de simples corrections.

Une difficulté peut apparaître: les non-spécialistes ont à résoudre parfois des problèmes particuliers d'édition de fichiers. Il faut alors que les mécanismes d'extensions du langage d'édition soient, eux aussi, faciles à employer. La définition de nouvelles structures pour les programmes d'édition m'a permis d'aborder des domaines très différents: Programmation des systèmes à temps partagé conversationnels- Terminaux graphiques- Interprétation de langages- Traitements de caractères- Gestion de fichiers- Langage de commandes.

Les buts de la machine d'édition sont volontairement ambitieux. Il est possible que dans le cadre particulier d'un système conversationnel donné, on soit amené à limiter le rôle de certains modules ou à revoir certains mécanismes (gestion de fichiers- interprétation). Néanmoins, les primitives d'édition peuvent s'appliquer à tout autre domaine où l'on doit manipuler des objets ayant une certaine structure.

Il me semble essentiel enfin de dégager les points suivants:

- Tout éditeur doit être composé d'un certain nombre de modules. Cette modularité rend la conception et l'adaptation à différents systèmes plus faciles. Elle permet également de nombreuses extensions.

- Le terminal doit permettre le développement du dialogue entre l'homme et la machine. Dans ce sens l'apport d'un terminal graphique est considérable mais il reste encore à étudier et développer des éditeurs pour consoles de visualisation alphanumériques.

- Le langage de commande doit être simple mais efficace. A ce sujet, la définition d'un langage, de haut niveau, pour la manipulation conversationnelle de fichiers et la réalisation du compilateur correspondant peuvent constituer une suite logique à mon travail sur l'édition.

# REFERENCES BIBLIOGRAPHIQUES

Les ouvrages cités sont classés par thèmes.

## SYSTEMES- GESTION DE FICHIERS-

S1 : A-AUROUX- C.HANS
"Les Systèmes CP/67 et C.M.S" Grenoble 1969.

S2 : L.BOLLIET
"Utilisation des ordinateurs en temps réel et en temps partagé".
AFIRO Monographie d'Informatique T2 - DUNOD 1966

S3 : J.BERTIN- M.RITOUT- J.C.ROUGIER
"L'exploitation partagée des calculateurs". AFIRO Monographie d'Informatique- DUNOD 1967.

S4 : O.LECARME
"Les systèmes d'exploitation des ordinateurs"
Cours de l'Institut de Prógrammation- Avril 1970.

S5 : I.B.M.
"CP/CMS User's Guide" Cambridge Scientific Center 320-2015.

S6 : I.B.M.
"CP/CMS Program Logic Manual". C.S.C.

S7 : W.C. Mc GEE
"Generalized File Processing" Annual Review in Automatic Programming. Vol 5. p.77 à 150- Pergamon Press.

S8 : D.T. ROSS

"The A.E.D free storage package"
Communications of the ACM Vol.10 n°8 Août 1967.

S9 : KNUTH

"The Art of Computer Programming" Vol.1.

S10 : B.PETEUL-HARMEL

"Environnement de mise au point"
Thèse 3ie Cycle . Avril 1971.

S11 : J.P. LE HEIGET

"Méthodes d'accès virtuelles"
Mémoire CNAM à paraître 1971.

EDITION

E1 : P.DEUTSCH. B.LAMPSON
"An on-line editor"
Communications of the ACM Vol.10 No.12 Decembre 1967

E2 : P.DEUTSCH.D.ANGLUIN
"QED Reference Manual. Time Sharing Editor".

E3 : P.SAMSON
"PDP-6- TECO"
Memorandum MAC M.250 M.I.T Juillet 1965.

E4 : M.ADIBA- C.HANS
"Description externe de l'éditeur de CMS.EDIT".
Note technique IMAG.C.S.C Mai 1969

E5 : M.ADIBA.J.P.LE HEIGET
"Addition de Macro-Requêtes à Edit"
Note technique IMAG (CP/CMS) Février 1970

E6 : Computing Center Memo 118. *ED (Editeur de M.T.S) Janvier 1970.

E7 : "A conversational Context-directed editor" IBM CSC.

E8 : D.TUTTLE
"EDIT II - A non system specific context Editor"
IBM. C.S.C. Août 1969.

E9 : X.de LAMBERTERIE
"Réalisation d'une machine d'édition"
Rapport de DEA- Juin 1971- Grenoble

TERMINAUX GRAPHIQUES- EDITEURS GRAPHIQUES

G1 : M.ADIBA

"Edition de textes au 2250 " Séminaire de Programmation IMAG Juin 1969.

G2 : M.ADIBA JP.LE HEIGET

"Edition Graphique" Colloque sur les traitements graphiques. Grenoble- Avril 1970.

G3 : M.ADIBA

"Editeur Graphique" Congrès de l'AFCET Paris Septembre 1970.

G4 : J.BEVIZIN

"Les consoles de visualisation alphanumériques et leurs Applications".Rapport de DEA Grenoble- Juin 1970.

G5 :M.L.COLEMAN

"Text Editing on a graphic display device using trand-drawn proofreader's symbols". Pertinent Concepts in Computer Graphics. Proceedings of the 2nd University of Illinois Conference on Computer graphics. Chicago 1969.

G6 : E.CLEEMANN O.LECARME M.LUCAS

"Les langages de programmation graphique" Revue de l'AFIRO Décembre 1968.

G7 : O.LECARME

"Un système de programmation graphique conversationnelle" Thèse de Doctorat d'Etat es-sciences appliquées Grenoble Septembre 1970.

G8 : M.LUCAS

"Techniques de Programmation et d'utilisation en mode conversationnel des terminaux graphiques". Thèse de 3ie Cycle Juin 1968.

G9 : MUENZNER

"EDIT 1 An experimental text editor for the IBM 2250/1130." Yorktown Avril 1968 IBM Report.

G10 : C.I.JOHNSON

"Principles of Interactive Systems". IBM System Journal Vol.7 N°3 et 4 1968.

G11 : A.APPEL. F.P.DANKOWSKI- R.L.DOUGHERTY.

"Aspects of Display Technologie" Mêmes références que G10

G12 : H.J.GENTHNER

"Interactive Computer Graphics" Control Data Corp. Computers and Automation Novembre 1968.

G13 : F.W.GIESIN. T.E JOHNSON. C.I.JOHNSON

"An experimental Graphic Input Tablet and comparator interface for IBM 2250 1 X 4". IBM CSC Mars 1969.

G14 : I.B.M. 2250

Display Unit Model 1- Component Description- (A27-2701)

G 15 : IBM S/360

Programming Services for FORTRAN IV(C27-6909)

G 16 : IBM S/360 OS Graphic Programming Services for IBM 2250(C27-6909)

G 17 : IBM 2260 Display Station Component Description(A27-2700)

LANGAGES-TRAITEMENT DE CARACTERES-

L1 : R.E.GRISWOLD
"String Processing and the SNOBOL4 language"
NATO Summer School Copenhague Aout 1969.

L2 : R.E.GRISWOLD J.F.POAGE- I.P.POLONSKY
"The SNOBOL4 Programming language" Prentice-Hall 1968.

L3 : M.ADIBA
" Le langage SNOBOL4"
Note technique et Séminaire de Programmation IMAG-Janvier 70-

L4 : S.E.MADNICK
"CMS360 SNOBOL USER'S manual" IBM C.S.C. Novembre 1967.

L5 : S.E. MADNICK
"SPL/1 an string Processing Language"
IBM CSC 3202005 Juin 1965

L6 : V.H.YNGE
"COMIT as an IR Language" Programming systems and languages p.375.

L7 : VAN VIYNGAARDEN. B.J.MAILLOUX J.E.L.PECK. C.H.A. KUSLER
"Final Draft report on the algorithmic language ALGOL 68".

L8 : S.E.MADNICK- A.MOULTON
"Script an on-line manuscript Processing System".
IBM C.S.C 320-2033 Avril 1968.

L9 : L.OHRINGER
"Progress in Computerized Typesetting" IBM Report Juin 1965.

L10 : S.CARMODY W.GROSS T.NELSON D.RICE A.VAN DAM
"A Hypertext editing System for the 360" Brown University
References de G5.

L11 : I.B.M. S/360 Text Processor
"Hyphenation /S60" - System Manual
- Program Description manual
- Operations manual

L12 : BERTHAUD, SAVARY
"Macro-langage de commande" Rapport de DEA Juin 1971.

VU

Grenoble, le

Le Président de la Thèse

VU, et permis d'imprimer,

Grenoble, le

Les Co-Présidents de l'Assemblée Constitutive
Présidents de l'Université Scientifique & Médicale