UNIVERSITE DE NANCY I



# THESE

présentée à

L'UNIVERSITE DE NANCY I

pour obtenir le grade de

DOCTEUR DE TROISIEME CYCLE

EN GENIE ELECTRIQUE , OPTION AUTOMATIQUE

par

Philippe RENAULD



## ETUDE ET REALISATION D'UN SYSTEME D'AIDE A LA PROGRAMMATION EVOLUEE POUR LA COMMANDE DIRECTE D'UNE MACHINE OUTIL.

*Soutenue publiquement le 28 juin 1984 devant la Commission d'Examen* :

Président : Monsieur R. HUSSON

Examinateurs : Messieurs M. VERON
J.P. MUSSE
M. CHAMPENOIS
J.P. MARTY



*CENTRE DE RECHERCHE EN AUTOMATIQUE DE NANCY*

*LABORATOIRE D'AUTOMATIQUE ET DE COMMANDE NUMERIQUE*

UNIVERSITE DE NANCY I

# THESE

présentée à

L'UNIVERSITE DE NANCY I

pour obtenir le grade de

DOCTEUR DE TROISIEME CYCLE

EN GENIE ELECTRIQUE , OPTION AUTOMATIQUE

par

Philippe RENAULD



## ETUDE ET REALISATION D'UN SYSTEME D'AIDE A LA PROGRAMMATION EVOLUEE POUR LA COMMANDE DIRECTE D'UNE MACHINE OUTIL.

*Soutenue publiquement le 28 juin 1984 devant la Commission d'Examen* :

Président : Monsieur R. HUSSON

Examinateurs : Messieurs M. VERON
J.P. MUSSE
M. CHAMPENOIS
J.P. MARTY

*CENTRE DE RECHERCHE EN AUTOMATIQUE DE NANCY*

*LABORATOIRE D'AUTOMATIQUE ET DE COMMANDE NUMERIQUE*

# AVANT - PROPOS

Cette étude a été effectuée au sein du Centre de Recherche en Automatique de Nancy ( C.R.A.N. ), que dirige Monsieur le Professeur R. HUSSON , à qui j'exprime ma reconnaissance pour l'honneur qu'il me fait en présidant cette thèse.

Ce travail a été réalisé au Laboratoire d'Automatique et de Commande Numérique ( L.AC.N. ) animé par Monsieur le Professeur M. VERON. Je le remercie pour le dynamisme qu'il sait faire régner dans son groupe de recherche.

Je tiens à remercier plus spécialement Monsieur J.P. MUSSE Maître- Assistant qui est à l'origine de cette étude, et exprimer ma profonde reconnaissance pour l'aide chaleureuse qu'il m'a apportée tout au long de la rédaction de cette thèse.

Je remercie de même vivement Monsieur M. CHAMPENOIS, Chef de service du bureau d'études d'électronique de la société BOSCH-FRANCE, d'avoir bien voulu examiner mon étude et accepter de siéger à mon jury de thèse.

Je suis très sensible à l'intérêt que porte Monsieur J. PH. MARTY, gérant de la société SEIMELEC, à mes travaux. Je tiens à le remercier de sa présence à la commission d'examen.

J'associe à ces remerciements Messieurs Y. HACQUARD et F. LEPAGE qui m'ont apporté une aide précieuse et constante tout au long de cette étude.

- TABLE DES MATIERES -

# INTRODUCTION

La commande numérique des machines-outils est apparue il y a environ 35 ans, afin de résoudre les problèmes d'usinages complexes. Depuis cette époque, les progrès les plus significatifs ont surtout porté sur les caractéristiques techniques et fonctionnelles des unités de gouverne. En particulier, l'utilisation des microprocesseurs a permis de développer des systèmes de type C.N.C. (Computirized Numerical Control ), offrant des procédures d'usinage de plus en plus élaborées [1]. L'évolution actuelle des systèmes C.N.C. tend à simplifier le dialogue entre l'homme et la machine, facilitant ainsi l'établissement des gammes d'usinage. Aussi assistons-nous à l'intégration dans les armoires de commande, de langages évolués de programmation ou de moyens de simulation graphique.

La société BOSCH a contacté notre laboratoire afin de définir puis de réaliser un système d'aide à la programmation à implanter dans une armoire de commande numérique pour fraiseuse, qui soit modulable en fonction des besoins de l'utilisateur. Pour satisfaire ces deux exigences,messieurs Jean-Pierre Musse et Francis Lepage ont défini un langage de type macro-interprète ( appelé LAMULE ) qui permet la création de toutes les fonctions géométriques, technologiques et d'usinage par la simple écriture de sous-programmes. Les instructions du langage d'aide à la programmation seront alors constituées d'appels de ces fonctions, la syntaxe de ceux-ci étant entièrement définie dans l'entête des sous-programmes descripteurs des fonctions.

Dans un premier temps, nous avons écrit et mis au point le macro-interprète, puis nous avons développé les procédures géométriques et d'usinage orientées vers le problème particulier du fraisage. Nous avons ensuite réalisé l'adaptation spécifique à la commande numérique BOSCH ALPHA 3.

Pour ce qui concerne le problème du dialogue homme-machine, plusieurs solutions peuvent être envisagées; langage, mode questions-réponses, menus hiérarchisés, voir mini C.F.A.O., mais le mode de dialogue ne doit pas influer sur la représentation interne des différents éléments manipulés par le système.

Nous avons alors décidé de choisir un dialogue de type langage, proche de cette représentation interne, sachant que l'écriture des programmes-pièces par une méthode plus conversationnelle constitue la suite logique de notre travail.

Pour des raisons de confidentialité, notre travail est présenté dans deux documents distincts, le mémoire proprement dit et une annexe technique détaillée.

Le premier chapitre de notre mémoire présente les langages de programmation existants. Le deuxième est consacré à la définition du macro-interprète, à la représentation interne des différents éléments de base, ainsi qu'à la méthode de génération des macro-fonctions du système. Il présente de plus tout l'environnement de travail sous l'interprète LAMULE : éditeur de texte, metteur au point, gestion des procédures et des programmes.... Le chapitre 3 explique l'adaptation au problème du fraisage en décrivant les différentes instructions développées pour cette application. Un dernier chapitre propose les évolutions possibles du système pour assurer une meilleure convivialité et apporter des fonctions d'usinage , plus puissantes.

L'annexe technique regroupe quant à elle les manuels d'utilisation du système de programmation et de l'adaptation au fraisage, ainsi que l'analyse détaillée du noyau de l'interprète et des sous-programmes systèmes.

# CHAPITRE 1

# PRESENTATION DES SYSTEMES DE PROGRAMMATION EXISTANTS

Notre propos est de présenter, dans ce mémoire, le système de programmation que nous avons réalisé, en décrivant sa représentation interne puis son adaptation au problème du fraisage. Ceci nécessite de bien connaitre les différentes fonctions géométriques et d'usinage, utiles en commande numérique, ainsi que les méthodes les plus courantes de représentation interne des éléments manipulés.C'est pourquoi nous présentons en premier lieu, des systèmes déjà existants, aprés un rappel sur la programmation manuelle en commande numérique, afin de définir certains termes que nous utiliserons fréquemment dans la suite de ce mémoire.

## 1. RAPPEL SUR LA PROGRAMMATION MANUELLE

### 1.1. Généralités

La programmation manuelle consiste en une description pas à pas de tous les déplacements élémentaires à effectuer par l'outil ainsi que de toutes les fonctions auxiliaires de changement d'état de la machine ( mise en route de la broche, arrosage, etc... ). Ce mode de programmation est utilisé principalement pour les opérations d'usinages simples de perçage, d'alésage, de tournage, etc .... Mais dans le cas des pièces complexes, la programmation manuelle s'avère vite fastidieuse, compliquée voir impossible.

Un programme "manuel" est constitué d'instructions successives appelées *blocs-machines* qui définissent chacun une opération elémentaire. [2]

### 1.2 Blocs-machines

Un bloc-machine est une suite d'informations géométriques et technologiques, précédée d'un numéro de séquence décrivant une étape de l'éxécution du programme. Le format des blocs est standardisé par la norme ISO et se représente par une suite de couples lettre-donnée numérique [3].

Les lettres appelées *lettres-adresses* indentifient, par exemple, pour la commande numérique ALPHA 3 [4]:

N N° de séquence, classiquement première lettre-adresse

G Fonction préparatoire (modale ou non)
Une fonction préparatoire définit l'état de fonctionnement propre à la machine.

Par exemple :

G01 interpolation linéaire (fonction modale)

G04 temporisation dans le bloc courant (fonction non modale)

H Durée de temporisation en 1/10° de seconde

X
Y Coordonnées cartésiennes
Z

T N° d'outil ou de correcteur

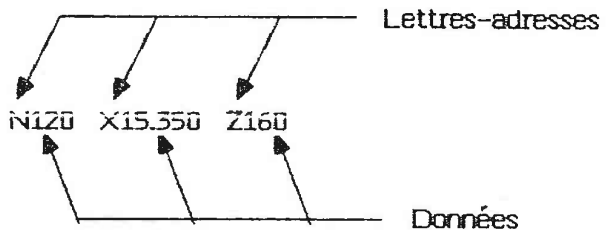
F Vitesse d'avance

M Fonction auxiliaire
Une fonction auxiliaire est une fonction qui définit tout état de la machine autre que les déplacements ou les contournages.

Par exemple :

M00 arrêt du programme
M04 broche en rotation à gauche

STRUCTURE D'UN BLOC-MACHINE



### 1.3 Préparation du programme

Il est nécessaire d'établir une gamme opératoire définissant la suite chronologique des opérations d'usinage en tenant compte des données technologiques de la machine, des outils et matériaux utilisés. Pour établir cette gamme, il faut procéder de la manière suivante [2] :

- Dresser la liste des éléments géométriques définissant le contours de la pièce à usiner.
- Si la commande numérique ne dispose pas de correction de rayon d'outil tracer la trajectoire du centre d'outil en tenant compte du rayon et calculer les points de changement de mode d'interpolation ou de trajectoire.
- Associer aux déplacements obtenus les paramètres technologiques en fonction de la machine et des matériaux.
- Vérifier les cas de collision.
- Etablir le programme-pièce correspondant à l'usinage.

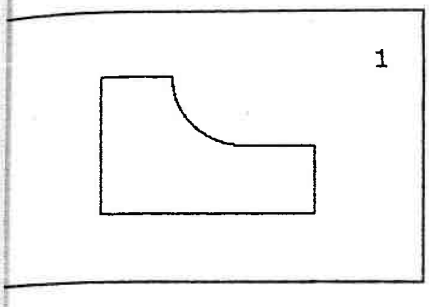
La figure 1 résume schématiquement cette méthode de programmation.

### 1.4 Conclusion

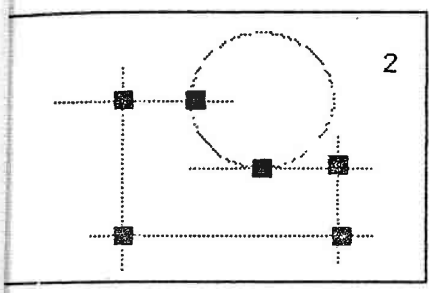
La programmation manuelle s'avère :

- Fastidieuse dans le cas des usinages complexes
- Non paramétrable pour une famille de pièces
- Aléatoire dans le cas d'impossibilité de simulation graphique

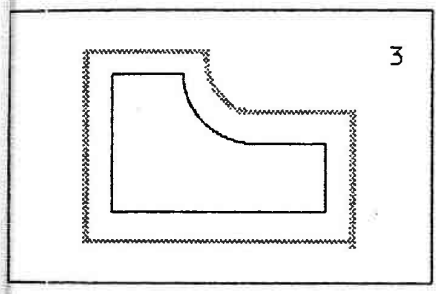
Il faut remarquer que la programmation manuelle que nous venons de présenter ne se retrouve que sur des armoires de commande très bas de gamme et tend actuellement à disparaitre, ou tout au moins à être complétée par des fonctions évoluées telles que des appels de sous-programmes, des définitions géométriques simples, des cycles d'usinages paramétrables etc...



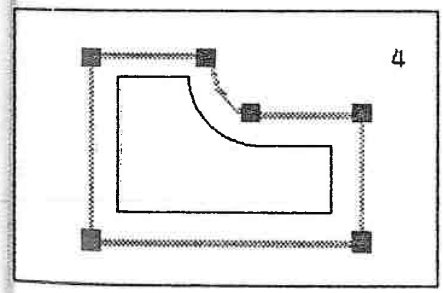

DESSIN DE LA PIECE A USINER




ELEMENTS GEOMETRIQUES DEFINISSANT LE PROFIL DE LA PIECE




DEPORT DU RAYON D'OUTIL




CALCUL DES POINTS DE CHANGEMENT DE MODE D'INTERPOLATION

**DONNEES TECHNOLOGIQUES**

ETABLISSEMENT DU PROGRAMME-PIECE

| N | G | X | Y | Z | M | H | F | T |
|---|---|---|---|---|---|---|---|---|
| N1 | G01 | X10 | Y20 | | | | | |

*- Figure 1 -*

## 2. PROGRAMMATION AUTOMATIQUE

### 2.1. Généralités

En programmation manuelle l'usinage de la pièce est décrit par des instructions de déplacement élémentaire de l'outil et des instructions de définition de l'état machine.Par contre les systèmes de programmation automatique imposent la description de l'usinage en trois phases [2] :

- Définition géométrique de la piéce en deux ou trois dimensions suivant le système utilisé.
- Programmation du parcours de l'outil en s'appuyant sur les éléments géométriques préalablement définis.
- Introduction des données technologiques relatives à la machine, aux outils et aux matériaux utilisés.

### 2.2. Mode de fonctionnement

Beaucoup de systèmes de programmation automatique fonctionnent de manière non-interactive, c'est à dire que le programme de l'utilisateur est d'abord écrit entièrement, puis traité à la manière d'une compilation pour générer le programme destiné à la machine outil. Il s'avère souvent nécessaire d'effectuer plusieurs traitements, avant de réussir la génération correcte des blocs correspondant à la pièce à usiner.

D'autres systèmes, par contre sont interactifs ou conversationnels. Dans ce cas, le calculateur analyse syntaxiquement puis exécute chaque instruction après sa définition. L'utilisateur peut alors modifier immédiatement la séquence de programmation erronée, ce qui limite le nombre des traitements à effectuer pour générer un programme correct.

### 2.3. Phase d'adaptation à la machine

Les langages de programmation automatique sont conçus pour la génération de blocs-machines adaptés à différents types d'armoire de commande. Il existe deux méthodes d'adaptation qui différencient l'élaboration du programme d'usinage :

- Utilisation d'un *post-processeur*
- Utilisation d'une *jonction*

2.3.1. Système à post-processeur

Beaucoup de systèmes génèrent un programme en langage intermédiaire, indépendant de la machine. La fabrication du programme pièce s'effectue alors en deux passages :

- Elaboration du programme intermédiare appelé CLDATA (Cutter Location DATA) qui dépend uniquement de la géométrie de la pièce et du parcours des outils.

- Après introduction des données technologiques, génération des blocs par analyse de la CLDATA à l'aide d'un logiciel particulier, appelé post-processeur spécifique de l'armoire de commande à laquelle est destiné le programme-pièce.

La figure 2 présente la structure d'un système de haut niveau qui intègre de plus un processeur technologique permettant une définition automatique des conditions de coupe, en fonction du matériau et des outils utilisés.

2.3.2 Système à jonction

D'autres langages n'élaborent pas de programme intermédiaire (CLDATA), mais, par contre, intégrent la phase d'adaptation à la machine (appelée dans ce cas jonction), au système de traitement qui ne comporte alors qu'une seule passe.

La jonction est réalisée de deux manières différentes suivant les systèmes :

- soit il sagit d'un module intégré au système lors de la génération de celui-ci. (Exemple COMPACT II [2])

- soit cette génération est constituée de sous-programmes écrits dans le langage du système et comportant des instructions de spécification des caractéristiques de la machine utilisée. (Exemple PROMO [5])

La suite de l'exposé présente trois systèmes représentatifs :

- APT
- PROMO
- GTL

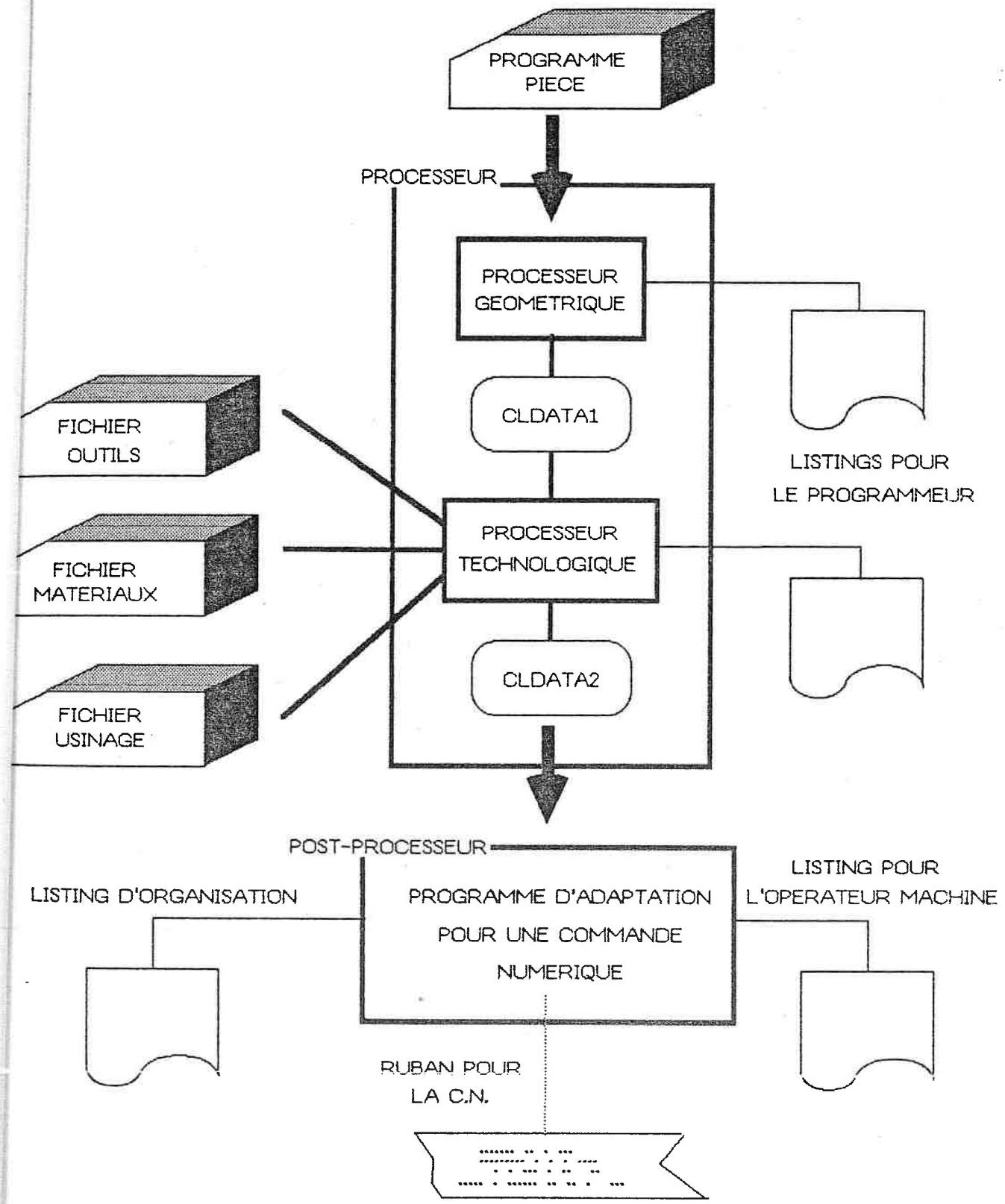

PROGRAMME
PIECE
PROCESSEUR
PROCESSEUR
GEOMETRIQUE
CLDATA1
FICHIER
OUTILS
FICHIER
MATERIAUX
FICHIER
USINAGE
PROCESSEUR
TECHNOLOGIQUE
CLDATA2
LISTINGS POUR
LE PROGRAMMEUR
POST-PROCESSEUR
LISTING D'ORGANISATION
PROGRAMME D'ADAPTATION
POUR UNE COMMANDE
NUMERIQUE
LISTING POUR
L'OPERATEUR MACHINE
RUBAN POUR
LA C.N.

*Figure 2*

## 3. SYSTEME APT ( Automatically Programmed Tool )

### 3.1 Introduction

Le langage APT reste la référence dans le domaine de la programmation automatique de machine-outil à commande numérique. APT répond en effet à tous les problèmes d'usinages classiques. Ce langage est développé par l' I.I.T. ( Illinois Institue of Technologie ) soutenu par une association d'utilisateurs cotisant pour l'amélioration de ce programme. [2]

### 3.2. Le langage APT

Le langage APT est un système non interactif élaborant le programme pièce en deux passages ( Processeur puis Post-processeur ). APT est un langage symbolique comprenant un vocable d'environ 300 mots. Le préparateur de fabrication possède un jeu d'instructions qui définissent :

- la géométrie de la pièce
- la tajectoire des outils
- les fonctions de la machine-outil
- les commandes du calculateur

#### 3.2.1. Géométrie de la pièce

La géométrie utilisée par APT est tridimensionnelle, l'espace étant repéré par rapport à un système orthonormé dont l'axe Z est dirigé de la table de la machine vers la broche. [6]

L'attribution par l'utilisateur, d'un nom symbolique à un élément géométrique donné, au moment de sa définition, permet de référencer cet élément dans une instruction ultérieure.

Une instruction de définition géométrique comporte donc :

- le nom symbolique suivi du signe égal '='
- la nature de l'élément géométrique : Point, Droite , Cercle etc... suivie d'une barre oblique '/'
- la définition analytique ou géométrique de cet élément

| |
|---|
| Nom Symbolique = Elément géométrique / – Définition analytique<br>– Définition géométrique |

APT permet la manipulation de nombreux éléments géométriques en deux ou trois dimensions qui sont en particulier :

- des points
- des droites et cercles dans le plan XY
- des coniques
- des surfaces simples (plan) ou plus complexes (sphères, cylindres, cônes quadriques etc...)

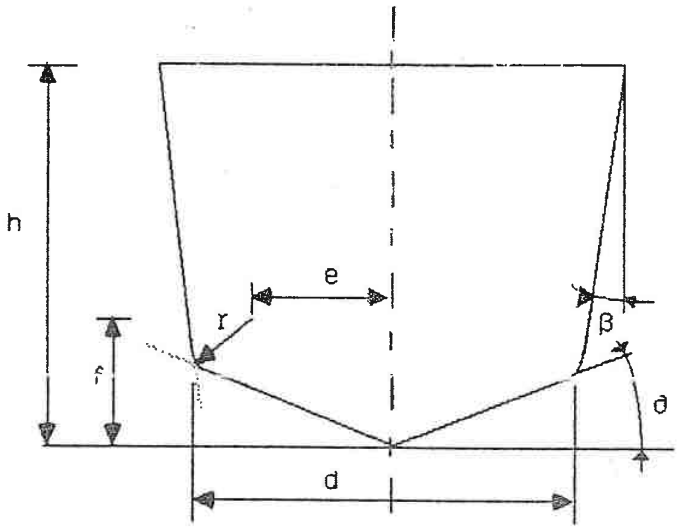
Il propose également des matrices de transformations géométriques formées d'un tableau de 12 coefficients qui représentent une transformation linéaire des coordonnées, en particulier les transformations, rotations et homothétie. Une transformation est appliquée soit à la géométrie au moment de sa définition, soit aux mouvements de l'outil.

### 3.2.2. Trajectoire des outils et définitions technologiques

L'outil

L'outil APT, le plus général, est défini par sept paramètres permettant de se satisfaire de tous les types d'outils aussi bien pour le fraisage que pour le tournage [6].

CUTTER/ d, r , f , ∂, β , h

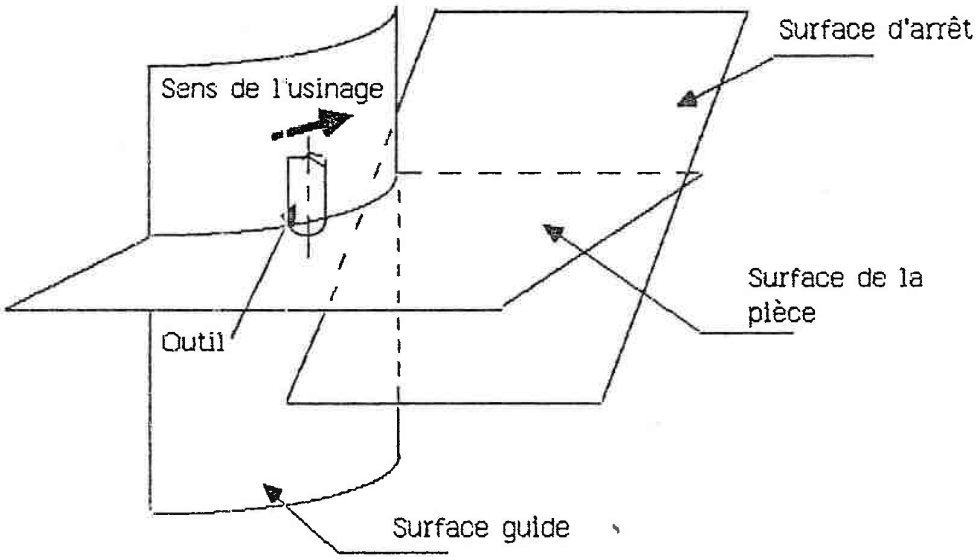



*- Figure 3 -*

Mouvement d'outil

La description d'un mouvement de l'outil fait appel à 3 surfaces:

- La surface de pièce qui peut être considérée comme définissant à chaque instant la cote de l'outil
- La surface guide que l'outil doit suivre au cours de son mouvement généralement la surface que l'on veut générer associée à un sens de parcours.
- La surface d'arrêt qui limite le mouvement de l'outil le long de cette surface guide, et deviendra généralement surface guide du mouvement suivant.




*- Figure 4 -*

Technologie

Après le premier traitement par le processeur, l'utilisateur fournit les informations technologiques relatives aux phases successives de l'usinage et lance la phase post-processeur qui génére le programme pièce.

### 3.2.3. Fonctions de la machine-outil

Ce sont les instructions qui définissent les états de la machine . C'est le post-processeur qui fabrique les blocs correspondants au format spécifique de l'armoire de commande, pendant la phase d'adaptation à la machine.

### 3.2.4 Autres instructions et commandes pour le calculateur

Le préparateur de fabrication dispose d'instructions particulières telles que des instructions arithmétiques FORTRAN, la définition de sous-programmes, des instructions de rupture de séquence d'exécution etc...

Bien évidemment de multiples commandes permettent à l'utilisateur l'édition du programme, sa mise au point, l'impression des différents listings etc...

## 3.3 Support informatique

APT est écrit en FORTRAN et nécessite un calculateur de 32 bits et 256 k-octets de mémoire centrale, ainsi qu'une mémoire de masse.

APT, par sa réponse universelle aux problèmes d'usinage et son support informatique important, est un système d'aide à la programmation haut de gamme.

## 4. SYSTEME PROMO (PROgrammation de Machine Outil)

### 4.1. Généralités

PROMO est un système de programmation développé par l'ADEPA qui est conçu pour les usinages en point à point et les applications de contournage deux dimensions, qui constituent l'essentiel des problèmes de tournage et fraisage. Ce système travaille en seule phase de traitement, la programmation est réalisée en mode conversationnel et le système signale les erreurs immédiatement à l'utilisateur [5].

PROMO permet de définir un usinage par la programmation successive :

- des éléments géométriques de la pièce
- des mouvements de l'outil
- des données technologiques.

### 4.2. Le langage PROMO

#### 4.2.1. Instructions géométriques

Pour déterminer la géométrie de la pièce, le programmeur possède un jeu d'instructions dont la syntaxe est proche de APT, mais simplifiée et se présente sous la forme suivante :

| Symbole = Mot Clé / Paramètres déterminant l'élément géométrique |
|---|

Le mot clé constitué de deux lettres détermine la fonction de l'instruction.

Exemple :

P1=LL/L1,L2 LL/ définit l'intersection de deux droites

Les paramètres sont des valeurs numériques ou des variables, des éléments géométriques ou bien encore des modificateurs pour lever les ambiguités.

Le tableau suivant représente les différents éléments géométriques et leur représentation interne.

Notons que ces éléments sont définis sans notion de sens de parcours et que leur forme canonique est stockée en mémoire centrale.

Tableau des éléments géométriques utilisés par le système PROMO

| ELEMENT | SYMBOLE | REPRESENTATION INTERNE |
|---|---|---|
| POINT | Pi | Coordonnées cartésiennes X Y Z |
| DROITE | Li | Une droite est définie dans le plan XY. Elle est mémorisée par les coefficients de son équation caractéristique a, b et c. Avec aX+bY = c |
| CERCLE | Ci | Un cercle est défini dans le plan XY. Il est mémorisé par les coordonnées de son centre et par son rayon. |
| STRUCTURE DE POINTS | Si | L'instruction de définition est stockée en mode caractère dans une zone spécialisée. |
| TRANSFORMATIONS GEOMETRIQUES | Mi | La transformation est mémorisée par les 12 coefficients qui déterminent sa matrice associée. |

### 4.2.2 Programmation de la gamme d'usinage

PROMO est un système en une seule passe utilisant pour l'adaptation à la machine un module de jonction (appelé macro de simulation) écrit en langage PROMO et faisant appel à des ordres de formatage de blocs machines et des valeurs numériques relatives à chaque lettre-adresse. Cette macro de simulation impose la manière de programmer les différentes conditions technologiques ou les cycles utilisés, et peut être adaptée aux besoins de l'utilisateur qui pourra lui même la modifier.

Ainsi après avoir décrit la géométrie de la pièce le programme comporte successivement les étapes suivantes :

- appel d'une macro d'initialisation de l'usinage
- définition de la technologie relative à une phase de la gamme d'usinage
- ordres de mouvements point à point ou de contournage utilisant les éléments géométriques préalablement définis et appelant automatiquement la macro de simulation en cours.

### 4.2.3. Modules spécialisés

Pour permettre la simplification des programmes, PROMO dispose de définition de macro-instructions, d'instructions arithmétiques et de rupture de séquence de traitement. Mais surtout, des modules spécialisés ont été développés (modules de suivi de profil, fraisage, tournage, oxycoupage...), qui mettent à la disposition de l'utilisateur des fonctions évoluées adaptées au problème à résoudre. Ainsi le module de tournage offre des commandes d'ébauche de profil de dresssage de face, etc... Le seul inconvénient est que l'appel de ces fonctions est effectué par l'intermédiaire de codes rendant le programme assez ésotérique.

## 4.3. Support informatique

PROMO est utilisable sur un ordinateur 16 bits de 32 k-octets de mémoire centrale dans la version autonome de programmation et ne nécessite pas de mémoire de masse. Il peut être également implanté sur l'ordinateur central de l'entreprise pour une utilisation en temps partagé.

## 5. SYSTEME G.T.L. (Geometrical and Technological Language)

G.T.L. est un système conversationnel développé pour la programmation des usinages point-à-point et de fraisage. Ce langage est implanté sur un calculateur OLIVETTI P6060 de 32 k-octets de mémoire et comporte les 2 phases classiques de traitement (Processeur puis Post-Processeur)[2].

La grande particularité de G.T.L. repose dans l'emploi d'une géométrie bidimensionnelle orientée, qui consiste à décrire implicitement le sens de parcours de l'outil, par le sens de définition des éléments géométriques supports au profil de la pièce. Aussi les droites et le cercle contiennent dans leur représentation interne un indicateur de sens de parcours.

Nous reviendrons dans la présentation de notre système sur l'intérêt majeur de ce type de géométrie qui facilite les instructions de définitions géométriques et de mouvements en minimisant le nombre de modificateurs utiles. Notons pour finir, que G.T.L. possède des procédures d'usinage complexes comme la réalisation de poche pour le contournage.

## 6. CONCLUSION

Parmi les systèmes que nous venons de présenter, nous avons essayé d'extraire les caractéristiques qui nous semblent les mieux adaptées à notre problème. Ainsi la géométrie orientée de G.T.L. nous a semblé très performante pour offrir des instructions géométriques très simples et des ordres d'usinage compact.

D'autre part, PROMO constitue un bon exemple de système en une seule phase, ne nécessitant pas de mémoire de masse, ce qui est obligatoire dans notre application. Quant au système APT qui est la référence, il nous a surtout servi à définir les instructions de l'adaptation au fraisage et leur syntaxe d'appel.

# CHAPITRE 2

# PRESENTATION DU SYSTEME LAMULE

Nous allons exposer dans ce chapitre les principes de base retenus pour la réalisation du macro-interprète LAMULE, ainsi que son utilisation pour l'élaboration des diverses fonctions géométriques, technologiques et d'usinage proposées par un système de programmation automatique.

# I - DESCRIPTION FONCTIONNELLE DU LANGAGE LAMULE

## 1. CARACTERISTIQUES GENERALES DU SYSTEME

Notre but est de mettre au point un système évolué de commande d'une machine-outil et en particulier d'une fraiseuse, permettant la programmation d'une pièce par l'appel séquentiel de fonctions d'usinages précises et bien adaptées au problème à résdoudre, pour limiter au minimum le travail du préparateur. Pour cela il faut pouvoir simplement définir les procédures d'usinages dont on a besoin, mais aussi la géométrie de la pièce et la technologie de l'usinage (géométrie d'outil, conditions de coupe,...).

Pour satisfaire la modularité et la souplesse d'emploi, ainsi que pour définir des fonctions très évoluées, MM. MUSSE et LEPAGE ont proposé un système ( LAMULE ), organisé autour d'un macro-interprète qui permet de créer très simplement toutes les instructions et procédures nécessaires.

Le choix d'un interprète a été effectué compte-tenu des considérations suivantes. En effet, si un langage de type compilateur offre comme principal avantage, une grande rapidité de calcul, il présente, dans la mise au point des programmes, une mauvaise interactivité, ce qui constitue dans notre application, un inconvénient majeur.

Une autre solution correspond à compiler l'ensemble des procédures de bases et à interpréter les programmes de l'utilisateur. En fait, cette manière de faire répond au dilemme, rapidité d'exécution, facilité de mise au point mais son principal désavantage est l'établissement de deux types de traitement, qui alourdissent alors la conception du système ainsi que la mise au point des procédures du langage de base. A l'heure actuelle, les microprocesseurs associés dans certains cas à des processeurs particuliers (arithmétique, graphique, ...) diminuent les temps de calcul dans des proportions importantes. Le temps d'exécution n'est plus alors une contrainte et justifie d'autant mieux le choix de l'interprète.

Une autre caractéristique fondamentale de notre système est la présence en mémoire centrale de tous les programmes et sous-programmes en langage évolué, permettant ainsi une mise au point très facile et rapide.

L'utilisation de LAMULE peut schématiquement être organisée en trois niveaux :

- Ecriture par le concepteur des instructions géométriques et d'usinage de base,
- Ecriture par l'utilisateur averti de procédures d'usinage évoluées,
- Utilisation par le préparateur de toutes ces fonctions pour l'exécution d'une pièce.

D'autre part le système proposé dispose d'un environnement d'utilisation qui offre, aussi bien au concepteur qu'au préparateur, des outils d'édition, de visualisation et de mise au point évolués, simples à mettre en oeuvre.

Nous présentons dans les paragraphes suivants les principes que nous avons retenus, ainsi que les fonctions offertes par le système LAMULE.

## 2. PRIMITIVES DU SYSTEME

Au départ, le système LAMULE ne dispose pas d'instructions évoluées puisque celles-ci seront construites en fonction des besoins. Toutefois, il faut un minimum d'instructions et de déclarations primitives pour pouvoir engendrer des fonctions de complexité croissante.

Le *tableau n° 1* récapitule les caractères spéciaux de déclaration de type d'instruction dont nous aurons besoin. Les primitives assembleur sont désignées par leur nom précédé du caractère "@". Elles sont regroupées dans le *tableau n° 2* et présentées au fur et à mesure des paragraphes suivants.

TABLEAU N°1 : DECLARATION DE TYPE DE PRIMITIVE

| Caractère de déclaration de type | Rôle |
|---|---|
| * | Génération de bloc-machine paramétrable |
| ? | Génération de bloc-machine non paramétrable |
| # | Déclaration de début ou de fin de sous-programme LAMULE |
| @ | Appel de procédure assembleur |
| % | Déclaration de macro-instruction explicite ou fin de macro |
| & | Déclaration de fin de macro-instruction implicite |
| < | Déclaration de message d'erreur |

TABLEAU N°2 : PRIMITIVES ASSEMBLEUR OBLIGATOIRES DESIGINEES PAR LE CARACTERE @

| PRIMITIVE | ROLE |
|---|---|
| Primitive d'erreur<br>@ER | Positionne l'indicateur d'erreur |
| Primitives d'affectation<br>@AFP<br>@AFL<br>@AFC | Transfèrent des représentations analytiques d'éléments géométriques dans des variables arithmétiques |
| Primitives de calculs arithmétiques<br>@RC<br>@VAL<br>@COS<br>@SIN<br>@ATG | Racine carrée<br>Valeur entière<br>Cosinus d'un angle en radian<br>Sinus d'un angle en radian<br>Arctangente |
| Primitives de rupture de séquence<br>@JMP<br>@IF<br>@ASE<br>@ASI | Saut inconditionnel<br>Saut conditionnel<br>Saut conditionnel à une valeur entière<br>Saut conditionnel à la valeur d'indétermination |

TABLEAU N°2 Suite

| PRIMITIVE | ROLE |
|---|---|
| Exécution d'une structure<br><br>@AR1 | Exécution de la structure de points dont l'adresse est contenue dans la représentation interne de la structure R1 |
| Exécution d'une transformation<br><br>@TR | Exécution de la transformation dont l'adresse de définition est contenue dans la représentation interne de la transformation T(V12) |
| Acquistion de paramètres<br><br>@PAR | Exploite en temps différé les paramètres d'appel d'une procédure |

## 3. OBJETS MANIPULES PAR LE SYSTEME

Deux grandes familles d'objets sont manipulables par le système LAMULE :

- Les variables arithmétiques nécessaires à tous les calculs concernant la géométrie, les fonctions d'usinage, etc...,
- Les éléments géométriques de définition de la pièce.

### 3.1. Variables arithmétiques

Deux types d'utilisation des variables arithmétiques sont à mettre en évidence :

- l'utilisation pour l'initialisation de grandeurs manipulées par le préparateur, comme par exemple les données technologiques (vitesse d'avance, rayon d'outil, ...),
- l'utilisation pour des calculs internes qui seront totalement transparents au préparateur.

Pour satisfaire à cette dualité, le système LAMULE propose deux types de variables :

- Les variables symboliques répondant à la première exigence, qui vont permettre de donner aux différents paramètres des noms mnémoniques (ex. : VA pour vitesse avance, DIA pour diamètre d'outil).

- Les variables de type Vi ( appelées par la suite variables systèmes qui seront utilisées pour les transferts de paramètres arithmétiques et les calculs internes aux différentes procédures.

  L'avantage de ce deuxième type est, qu'il ne nécessite pas la création d'une table de symboles, et diminuera ainsi les temps de traitement des expressions arithmétiques. Ces variables Vi peuvent également être utilisées comme index pour tous les objets sauf les variables symboliques.

Exemple : si V2 = 5

V(V2) désigne V5

P(V2) désigne le point P5

D'autre part certaines de ces variables ont une fonction très particulière pour les instructions d'affectation *(Cf. Chapitre II, paragraphe I.4.)*

3.2. Eléments géométriques

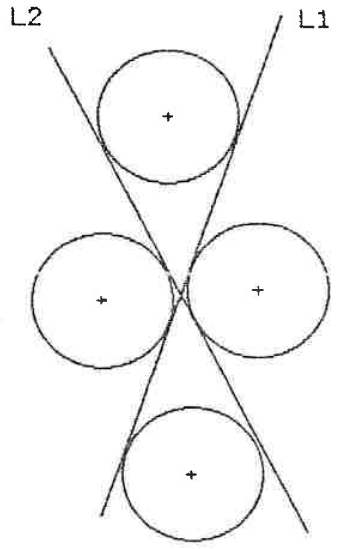
3.2.1. Choix d'une géométrie planaire orientée

La géométrie retenue pour notre système est planaire et orientée. En effet, la géométrie à deux dimensions répond à la grande majorité des problèmes posés par des pièces à fraiser en commande numérique. De plus en proposant des transformations géométriques, nous résoudrons la quasi totalité des problèmes tridimensionels simples. D'autre part, nous avons choisi la géométrie orientée qui présente deux grands avantages :

- a Lever d'ambiguité

Lors d'intersection ou de tangence de différents éléments, elle permet de simplifier la syntaxe et l'emploi des instructions géométriques ou d'usinage en limitant au minimum le nombre de modificateurs. *(Cf Chapitre II paragraphe 5)*

Exemple : Cas du cercle tangent à 2 droites et de rayon donné.

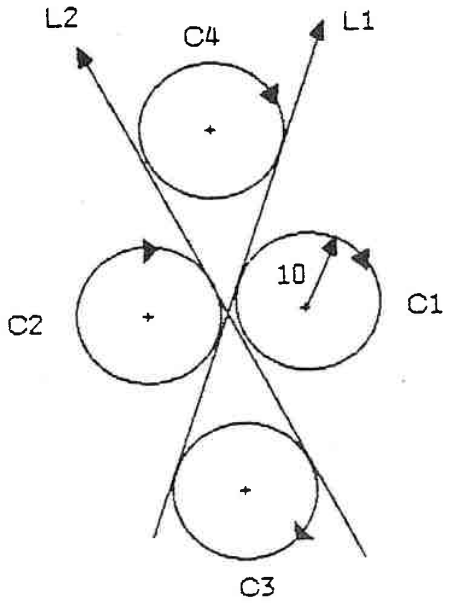


Quatre cercles répondent au problème. Aussi les langages qui utilisent une géométrie non orientée déterminent le cercle choisi en précisant sa position par rapport aux deux droites.

Exemple : Instruction PROMO C1=LL/ XS,L2,YS,L1,r

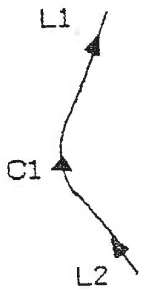
XS et YS signifient que C1 est à gauche de L2 et sous L1 respectivement.

La géométrie orientée, quant à elle, définit un sens de parcours associé à chaque élément géométrique et moyennant quelques restrictions nous n'aurons pas besoin de recourir à des modificateurs.




Nous choisissons d'interdire le rebroussement de parcours. Cette option ne correspondant pas à une réalité technologique.

Les cercles C2, C3 et C4 dans notre exemple sont alors éliminés.




Reste alors uniquement le cercle C1.

Notre instruction s'établit alors :

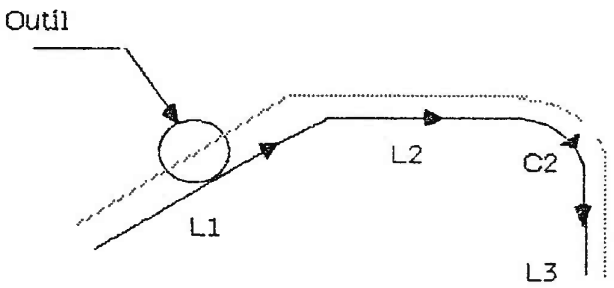
C1 = LL/ L2, L1, -10

- b Définition implicite du sens de parcours

Le second avantage de la géométrie orientée réside dans la définition implicite du sens de parcours d'usinage, par le sens de définition des éléments géométriques décrivant le profil de la pièce. (*Cf Chapitre III paragraphe 7*)

Ainsi il n'est pas nécessaire de préciser, à tout instant, la position de l'outil par rapport au profil, mais il suffira de la donner à priori, en se référant au sens de déplacement.

Exemple :




En PROMO la suite d'instruction serait :

```
LL/YL,L1,YL,L2
LC/OU,C2
CL/XL,L3,RT
```

En géométrie orientée, il suffira de dire que l'on contourne à gauche et de donner la suite des éléments du profil ce qui donnera :

```
CG/
L1
L2
C2
L3
-
-
&
```

### 3.2.2. Représentation interne des éléments géométriques

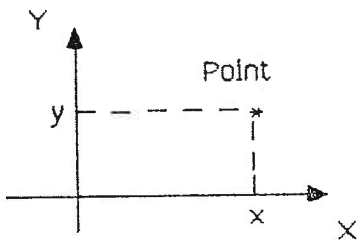
Les éléments géométriques proposés par le système sont :

- les points
- les structures de points
- les droites
- les cercles
- les transformations géométriques.

Pour gagner du temps d'exécution nous stockerons en mémoire centrale la représentation analytique (appelée *forme canonique*) de ces différents éléments.

Compte tenu du type de géométrie retenue et en essayant de minimiser la place mémoire utile, nous avons choisi les formes canoniques suivantes. Précisons d'abord que les éléments géométriques sont définis dans le plan XY dont la cote en Z sera précisée par une instruction spéciale ou un paramètre d'une fonction d'usinage.

- Point
  - Abcisse du point
  - Ordonnée du point

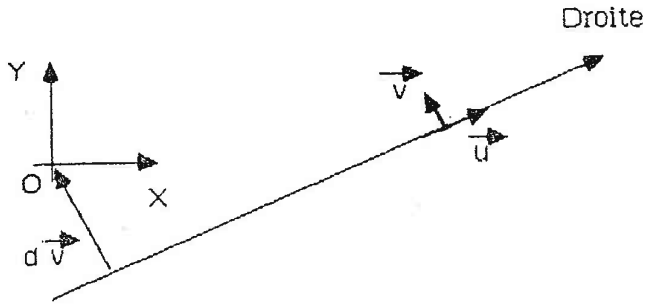



- Structure de points

Une structure de points permet de définir un ensemble de points répartis régulièrement sur un élément géométrique. Pour éviter la mémorisation de toutes les coordonnées de ces points, celles-ci ne seront calculées qu'au moment de l'utilisation de la structure dont il faut obligatoirement retenir la définition. Nous avons alors choisi de stocker l'adresse de l'instruction de définition, ce qui est permis, par la présence en mémoire centrale du programme pièce.

- Droite

  - Coordonnées du vecteur directeur de la droite
  - Distance à l'origine orientée suivant le vecteur directeur




$\vec{u}$ : Vecteur untaire de la droite

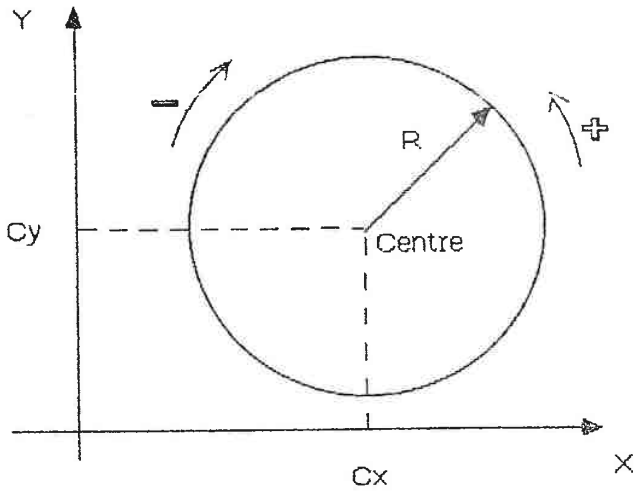
$\vec{v}$ : Vecteur directeur de la droite

d : Distance à l'origine orientée

$\vec{u} \perp \vec{v}$ tels que ces vecteurs forment un repère orthonormé direct

- Cercle

  - Coordonnées du centre du cercle
  - Rayon du cercle signé
    - négatif pour le sens horaire
    - positif pour le sens trigonométrique



- Transformations géométriques

De même que pour les structures de points,les transformations géométriques ne sont pas à traiter au moment de leur description et nous les mémorisons donc également, par l'adresse de l'instruction qui les définit.

Les différents éléments géométriques sont symbolisés par une lettre suivie d'un indice. Les procédures de définition géométriques et d'usinages complexes nécessitent l'utilisation d'éléments intermédiaires, transparents au préparateur. Pour éviter de réserver des numéros d'indices inutilisables par celui-ci, nous avons crée deux types d'objets :

- les objets utilisateurs
- les objets systèmes

à qui nous avons attribué des lettres-symboles différentes que présente le tableau N°3.

TABLEAU N° 3 : SYMBOLES DES ELEMENTS GEOMETRIQUES

| Eléments | Symboles | |
|---|---|---|
| | Système | Utilisateur |
| Point | Oi | Pi |
| Droite | Ki | Li |
| Cercle | Bi | Ci |
| Structure | Ri | Si |
| Transformation | Ti | |

3.3. Macro-instruction

Pour permettre au préparateur d'associer puis d'utiliser ensuite une série d'instructions regroupées sous un même nom symbolique, nous sommes amenés à créer des procédures de définition et d'appel de macro-instructions.

Le caractère pourcent " **%** " ,suivi d'un nom symbolique représente la déclaration d'une nouvelle macro explicite. L'instruction constituée de l'unique caractère **%** identifie la fin d'une macro.

```
Exemple :   %MAC      déclaration de la macro MAC
              ]
              ]       Corps de la macro composée d'instructions
              ]       LAMULE quelconques.
              ]
              %
```

La procédure d'appel d'une macro est effectuée par l'instruction constituée uniquement de son nom symbolique.

```
Exemple :   MAC       appel de la macro MAC
```

Pour augmenter la souplesse d'emploi de ces procédures le nom symbolique peut être indicé au moment de l'appel.

```
Exemple :   %MAC
            %
             V20=1
            MAC (V20)   appel de la macro-instruction MAC1
```

Une macro-instruction explicite est représentée en mémoire par son nom symbolique et l'adresse de début de son coprs qui reste en place dans le programme.

Pour réaliser un système de programmation souple d'emploi et puissant, il faut pouvoir déclarer des instructions comme paramètres de procédures complexes. Nous avons alors créer des macro-instructions implicites, (par opposition aux macros explicites que nous venons de présenter), dont la définition suit l'appel de la procédure qui les utilisera. La fin de chaque macro implicite est repérée par le caractère spécial &

Par exemple si la procédure FONC nécessitait deux groupes d'instructions comme paramètres, son appel se présenterait comme suit :

```
FONC
]   Corps de la première macro
&
]   Corps de la deuxième macro
&
```

Nous détaillerons dans le paragraphe d'établissement de nouvelles procédures, la méthode de déclaration de ce type de macro-instruction.

## 4. CREATION D'UNE INSTRUCTION OU D'UNE FONCTION

### 4.1. Principes

Pour obtenir la modularité souhaitée et aboutir à une mise au point facile des procédures du concepteur ou de l'utilisateur, nous avons retenu l'idée de créer toutes les fonctions géométriques, d'usinage ou autre, par l'écriture de sous-programmes en langage évolué. Toute instruction du langage, hormis les primitives, est alors l'appel d'un sous-programme. Il faut donc que la définition de celui-ci contienne toutes les informations de description syntaxique et fonctionnelle de l'instruction à laquelle il correspond.

La description syntaxique de l'instruction est réalisée par l'entête du sous-programme qui est constituée du caractère # suivi du nom symbolique de l'instruction et d'une zone de paramètres formels.

La fonctionnalité de l'instruction est décrite par le corps du sous-programme.

La fin du sous-programme, constituée du caractère # seul, assurera éventuellement les affectations des paramètres de sortie.

Exemple : L'instruction de syntaxe PROG,Li,Lk est créée par le sous-programme suivant

```
Entête   #PROG,K0,O0   <-  Zone des paramètres formels
Corps    [
         [
         #   <- Fin
```

### 4.2 Nom de sous-programme

Pour différencier les procédures de base des sous-programmes de l'utilisateur nous insérons obligatoirement le caractère / dans leur nom, ( nous appelerons par la suite *sous-programmes systèmes* , les sous-programmes de description des instructions de base ).

Exemple :

| #PLL/,K0,K1 | #PROG,K0,O0 |
|---|---|
| Déclaration de sous-programme système | Sous-programme utilisateur |

Pour que les instructions aient une syntaxe classique, il a fallu prévoir une méthode de recomposition automatique du nom de sous-programme de description. Aussi, par une instruction du type Pi=LL/ Lj, Lk notre système fabrique le nom du sous-programme PLL/ qui sera appelé pour exécuter cette instruction. Pour alléger la syntaxe plusieurs méthodes de recomposition sont proposées ( *Cf annexe technique*), en particulier le mot clé peut être omis, il sera alors reconstitué par le type des deux premiers paramètres utilisés.

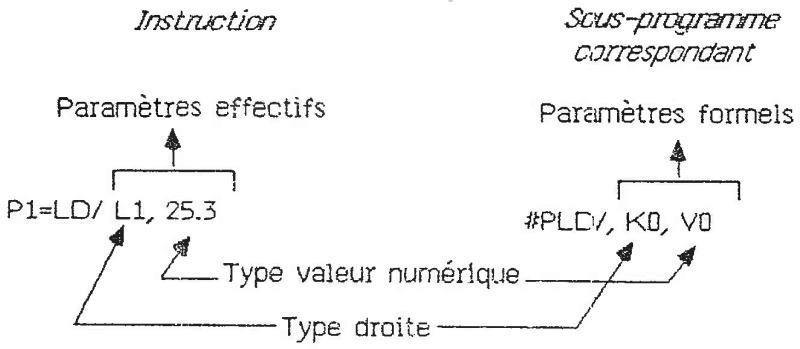
Exemple : P1 = LL/Lj,Lk ou Pi = Lj,Lk

Recomposent le sous-programme PLL/

### 4.3 Zone paramètre formelle

C'est cette zone qui va nous permettre de définir la syntaxe de l'instruction imposant la correspondance de type entre les paramètres formels et les paramètres effectifs.

Exemple :




De plus des codes spéciaux ont été définis pour que la déclaration de la zone de paramètres formels permette de décrire des syntaxes plus évoluées. ( *Voir annexe technique*). Par exemple, pour ne pas multiplier le nombre d'instructions, il est utile d'autoriser la déclaration optionnelle de certains paramètres formels.

Ainsi l'instruction Li=PA/ Pj, β [,Lk] définit une droite passant par Pj et faisant un angle β:

- avec OX si Lk est absente
- avec Lk si Lk est présente

D'autre part, certaines instructions utilisent des paramètres multiples dont l'exploitation devra être effectuée de manière cyclique par le sous-programme correspondant.

Enfin, nous pouvons passer comme paramètre, une macro-instruction implicite dont le nom sera symbolisé par Mi.

Exemple :

```
CD/        Instruction d'appel de          #CD/,M1
L1    [    la procédure CD                 Sous-programme de
L2    [                                    description de CD/
C3    [  Macro implicite                   #
&
```

4.4. Corps du sous-programme

Le corps d'un sous-programme définit la fonctionnalité de la procédure. Nous pouvons alors distinguer deux types différents de sous-programmes selon leur finalité:

- les programmes de définition d'un objet basé sur le calcul analytique de la forme canonique de celui-ci. Ce sont les sous-programmes qui possèdent une syntaxe d'appel comprenant un signe égal.

  Exemple : P1=L1,L2 P1 point d'intersection de deux droites L1 et L2

- les programmes d'exécution de procédures particulières, comme par exemple la définition d'un type d'usinage.

  Exemple : CG/ contournage à gauche du profil

Pour réaliser simplement les procédures d'affectation (syntaxe d'appel avec un signe égal), nous attribuons aux variables V50, V51 et V52 un rôle particulier. en effet le caractère # de fin de sous-programme, provoquera le transfert automatique de ces variables, dans la forme canonique de l'élément à affecter (élément à gauche du signe égal).

L'exécution d'un tel sous-programme est alors décomposée en :

- un passage des paramètres effectifs dans les paramètres formels
- un calcul analytique de la forme canonique qui sera stockée dans V50, V51 et V52.
- un transfert automatique de V50, V51 et V52 dans la forme canonique de l'élément à affecter.

Le calcul analtytique nécessite la récupération dans des variables systèmes, de la forme canonique d'éléments géométriques, ce qui nous amène à définir des instructions particulières effectuant ce type de transfert à l'aide de primitives assembleur spéciales. ( *Cf Tableau N°2 Primitives assembleur*)

Exemple :

V60 = P1 Réalise le transfert de l'abcisse et l'ordonnée de P1 dans les variables V60 et V61 respectivement

Le tableau suivant présente ces instructions particulières.

TABLEAU N° 4 : INSTRUCTIONS D'AFFECTATION DE VARIABLES Vi PAR DES ELEMENTS GEOMETRIQUES

| DEFINITION | FORMAT | |
|---|---|---|
| AFFECTATION DE LA FORME CANONIQUE D'UN POINT | Vn=P/Pi<br>ou<br>Vn=Pi | Vn = Abcisse de Pi<br>Vn+1 = Ordonnées de Pi |
| AFFECTATION DE LA FORME CANONIQUE D'UNE DROITE | Vn=L/Li<br>ou<br>Vn=Li | Vn = Abcisse du vecteur directeur de Li<br>Vn+1 = Ordonnée du vecteur directeur de Li<br>Vn+2 = Distance à l'origine orientée suivant le vecteur directeur de Li |
| AFFECTATION DE LA FORME CANONIQUE D'UN CERCLE | Vn=C/Ci<br>ou<br>Vn=Ci | Vn = Abcisse du centre du cercle Ci<br>Vn+1 = Ordonnée du centre du cercle Ci<br>Vn+2 = Rayon du cercle |

## 5. ARITHMETIQUE

Toute valeur numérique utilisée en zone paramètre effectif peut être une expression arithmétique dont les caractéristiques sont les suivantes :

- Les opérations arithmétiques classiques sont proposées :

  * Addition +

  * Soustraction -

  * Multiplication *

  * Division :

  ( Le caractère / étant réservé par la syntaxe d'appel des sous-programmes systèmes)

- Fonction puissance !

Le système admet les puissances entières de 2 à 9, ce qui répond à la majorité des problèmes d'usinage et évite l'emploi de multiplications répétitives.

- Les expressions arithmétiques sont formées de nombres de variables systèmes et symboliques, séparées par des signes arithmétiques.

- Il n'existe pas de niveaux de priorité entre les opérations, mais cinq niveaux de parenthèses sont permis.

- Les valeurs des nombres, et variables arithmétiques sont représentées en mémoire en notation virgule flottante 32 bits. ( Un octet d'exposant et trois de mantisse)

- Afin de détecter la non définition des variables systèmes ou des formes canoniques des objets géométriques, nous avons décidé de créer une valeur d'indétermination. Cette valeur située en dehors des bornes arithmétiques du système sera affectée à toutes les variables, au moment de l'initialisation de l'exécution.

De plus pour réaliser les procédures de définitions géométriques et les programmes piéces, nous munissons cette arithmétique, d'une série de fonctions de base :

- Racine carrée
- Valeur entière
- Valeur absolue
- Fonctions trigonométriques
- Signe d'expression arithmétique

Leur syntaxe d'appel est spécifiée dans le tableau suivant et les sous-programmes de base correspondant utilisent les primitives assembleur arithmétiques du système. ( *Cf Tableau N°2 Primitives assembleur*)

## 6. GENERATION DES BLOCS MACHINES

L'objectif final du macro-interprète LAMULE étant de permettre la création d'un système direct de programmation d'une armoire de commande numérique; il faut disposer d'instructions de génération de blocs machines.

Ces instructions doivent s'intégrer dans les procédures d'usinage, mais ne doivent pas être figées pour conserver une adaptabilité possible à des machines différentes et pour pouvoir corriger ou modifier facilement la syntaxe de génération des blocs machines.

Le principe, que nous avons alors adopté, est d'écrire directement le bloc machine précédé d'un caractère spécial, permettant de définir deux modes distincts :

- Le caractère ? effectuera la génération du bloc dont le contenu est précisé en toutes lettres derrière le point d'interrogation. Ce mode est utilisé pour générer des blocs commentaires ou d'initialisation de l'armoire de commande.

Exemple :


Ce texte est transmis à la commande numérique

TABLEAU N° 5 : INSTRUCTIONS ARITHMETIQUES

| DEFINITION | FORMAT | |
|---|---|---|
| SINUS D'UN ANGLE EXPRIME EN RADIAN | X=SN/β | β : Expression arithmétique |
| COSINUS D'UN ANGLE EXPRIME EN RADIAN | X=CS/β | β : Expression arithmétique |
| TANGENTE D'UN ANGLE EXPRIME EN RADIAN | X=TG/β | β : Expression arithmétique |
| ARC-TANGENTE | X=AT/n | n compris entre -π/2 et π/2 |
| VALEUR ABSOLUE | X=VA/n | n: Expression arithmétique |
| VALEUR ENTIERE | X=VE/n | n: Expression arithmétique |
| RACINE CARREE | X=RC/n | n: Expression arithmétique |
| SIGNE D'UNE EXPRESSION | X=SG/n | X = -1 l'expression est négative<br>X = 0 l'expression est nulle<br>X = 1 l'expression est positive |

- Le caractère * générera un bloc composé d'une succession de couples lettre-adresse valeur, la valeur pouvant être l'évaluation d'une expression arithmétique.

Exemple :

```
*NSEQ, XCOTX+10, YCOTY-20
```

Cette instruction générera le bloc N100 X25.2 Y-31.4

Si SEQ=100 COTX=15.2 COTY=-11.4

## 7. MESSAGE D'ERREUR

Notre but est de faciliter la mise au point des programmes par l'affichage de messages parlants qui puissent décrire clairement l'élément ou l'enchainement de procédures qui a provoqué la rupture de traitement.

Les erreurs de syntaxe étant detectées au moment de l'édition des programmes ( *Cf Paragraphe II4 Editeur-Ecran*), seules trois catégories d'erreurs restent possibles:

- Non détermination d'une macro-instruction
- Non détermination d'une étiquette lors d'une rupture de séquence
- Erreur arithmétique et non définition de paramètres

Les deux premiers types d'erreur génèrent un message crée par le système. Pour le troisième type notre choix s'est porté sur l'utilisation d'une instruction particulière d'affichage d'erreur, que l'on programmera dans le sous-programme de définition de toute instruction susceptible d'induire une erreur. Les erreurs de la troisième catégorie sont, en fait, détectées au moment du passage des paramètres, le système interrompt alors l'affectation des paramètres et positionne un indicateur d'erreur. L'instruction spéciale d'affichage d'erreur ne sera traitée que si cet indicateur est positionné, et le sous-programme en cours sera interrompu.

Exemple : P1=L1,L2 #PLL/,K0,K1

< DROITE NON DEFINIE

#

Lorsque L1 et L2 n'est pas définie le message < DROITE NON DEFINIE , apparaitra sur l'écran. Si plusieurs sous-programmes sont imbriqués les messages d'erreur correspondants seront affichés succesivement permettant ainsi d'obtenir un message extrémement précis.

8. INSTRUCTION SPECIALES

Le dernier type d'instruction à établir est contitué par l'ensemble des procédures de rupture de séquence. Pendant les phases de mise au point des différents programmes, l'ordre des instructions est amené à évoluer, c'est pourquoi nous distinguerons le numéro de l'instruction et l'étiquette qui lui est éventuellement attachée pour indiquer une reprise de séquence.

Cette étiquette est composée de deux chiffres au maximum et elle est séparée de l'instruction par un espace.

Exemple : 11 P1=L1,L2

Afin d'éviter toutes les ambiguités possibles et de réduire les valeurs numériques représentant les étiquettes, nous avons décidé que celles-ci seront locales à chaque sous-programme, et interdites dans le corps d'une macro-instruction.

Nous avons retenu 4 sortes de rupture de séquences, utilisant les procédures assembleur adéquates ( *Cf Tableau N° 2 Primitives assembleur*) et nous présentons leur syntaxe d'appel dans le tableau suivant.

TABLEAU N°6 : INSTRUCTIONS DE RUPTURE DE SEQUENCE

| DEFINITION | FORMAT |
|---|---|
| SAUT INCONDITIONNEL | JT/n<br>n: N° d'étiquette |
| SAUT CONDITIONNEL | IF/n,n1,n2,n3<br>n < 0 saut à l'étiquette n1<br>n = 0 saut à l'étiquette n2<br>n > 0 saut à l'étiquette n3 |
| SAUT CONDITIONNEL A UNE VALEUR ENTIERE | EN/n,n1<br>n : variable à comparer à sa valeur entière<br>n1 : N° d'étiquette dans le cas d'égalité |
| SAUT CONDITIONNEL A LA VALEUR D'INDETERMINATION | ID/n,n1<br>n : variable à comparer à la variable d'indétermination<br>n1 : N° d'étiquette dans le cas d'égalité |

## II ENVIRONNEMENT DU SYSTEME

Dans la première partie de ce chapitre nous avons présenté l'aspect fonctionnel du macro-interprète LAMULE. La seconde décrit son environnement constitué des différents éléments du système de programmation.

Les paragraphes qui suivent présentent ces éléments et les principes que nous avons retenu pour leur réalisation. Ils évoquent également les commmandes acceptées par le système dont on trouvera une récapitulation dans les deux tableaux suivants.

### 1. MATERIEL D'IMPLANTATION DU SYSTEME

Le système LAMULE fonctionne actuellement en version prototype sur le matériel de développement du microprocesseur 6809 EXORCISER de MOTOROLA qui comprend :

- Une carte microprocesseur à unité centrale 6809
- Une carte mémoire RAM de 48 k-octets
- Une console clavier-écran EXORTERM
- Un périphérique imprimante perforateur et lecteur de ruban ZIP30
- Une carte asynchrone série pour les liaisons directes avec l'armoire de commande ALPHA3.

### 2. ZONE SYSTEME - ZONE UTILISATEUR

Pour la définition du macro-interprète, nous n'avons pas défini deux niveaux de programmation afin de respecter l'aspect modulaire du système et de faciliter la conception de la phase d'adaptation au fraisage.

Pour éviter les problèmes qu'entraineraient les modifications de mauvais aloi lors des procédures d'adaptation, nous avons créé deux zones de programmation, l'une spécifique au concepteur ( Zone Système ) et l'autre à l'utilisateur ( Zone Utilisateur ).

Deux commandes spéciales ( NOR et SYS ) vont permettre de préciser si le travail d'édition qui suit est effectué sur la zone système, ou la zone utilisateur. Le mode normal est l'accès à la zone utilisateur et la commande SYS est reservée au concepteur. Les deux zones ne sont pas limitées à priori, mais évoluent suivant les besoins.

TABLEAU N° 7 : COMMANDES DU SYSTEME DE PROGRAMMATION

| Type de commande | Nom de la commande | Rôle |
|---|---|---|
| Commandes de sélection de zone | SYS | Sélection de la zone système |
| | NOR | Sélection de la zone utilisateur |
| Commandes d'édition visualisation et d'impression | C | Ecriture de programme au clavier à la fin de la zone de travail actuelle |
| | V | Visualisation de programmes |
| | "Home" ou LF | Passage en mode modification de programme |
| | LIST | Listing de programme |
| | L | Lecture du ruban de programmes, que le système place à la fin de la zone actuelle |
| | PUNCH | Préservation sur ruban perforé de source de programmes LAMAJLE |
| | N | Renumérotation des lignes d'une zone de programme |
| Commande d'équivalence de sous-programme | E | Crée l'équivalence entre 2 sous-programmes |
| Commande de catalogage de sous-programme | R | Catalogage de sous-programme |
| | T | Décatalogage de sous-programme |
| | Z | Commande d'effacement des sous-programmes non catalogués et mise sur le disque LAMSYSTE des sous-programme catalogués par l'utilisateur |

TABLEAU N° 7 SUITE : COMMANDES DU SYSTEME DE PROGRAMMATION

| Type de commande | Nom de la commande | Role |
|---|---|---|
| | G | Exécution de programme avec sortie de blocs-machines |
| | S | Simulation de programme |
| Commandes d'exécution-simulation | n | Exécution de n pas |
| | /n | Exécution jusqu'à la sortie de n blocs-machine |
| | "RC" | Exécution à nouveau de la commande n ou /n précédemment définie |
| | "Escape" | Abandon du mode exécution/simulation |
| | $Mnnnn= | Visualisation d'une place mémoire |
| | $BM= | Visualisation du dernier bloc-machine calculé |
| | $FC= | Visualisation de la dernière forme canonique |
| Commandes d'aide à la mise au point | $IN= | Visualisation de la dernière instruction exécutée |
| | Variable ou élément géo. | Visualisation d'une variable arithmétique ou d'un élément géométrique |
| | &VIS: | Création de la macro de visualisation |
| | O | Ouverture d'un sous-programme |
| | F | Fermeture d'un sous-programme |

### 3. ARCHIVAGE SUR DISQUE

Les programmes et sous-programmes LAMULE sont tous situés en mémoire centrale au moment de l'utilisation du système. Pour assurer la préservation des procédures de base mises au point, nous avons décidé d'archiver sur disquette la zone système. Cet archivage est effectué automatiquement lorsque l'on quitte le mode édition de la zone système ( Commande NOR ).

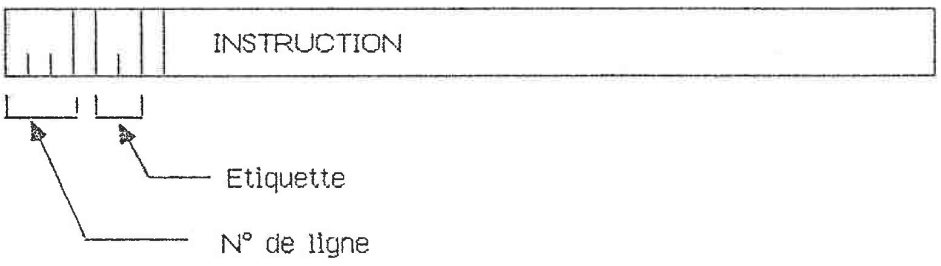
Pour assurer ce même avantage à l'utilisateur et ainsi lui permettre de céer sa propre bibliothèque de sous-programmes nous avons introduit une commande de catalogage permettant d'isoler parmi tous les sous-programmes utilisateur, ceux qui doivent être sauvegardés.

### 4. EDITEUR ECRAN

Notre système est, par nature conçu pour être modulable et modifiable au gré de l'utilisateur. Aussi pour rendre ces modifdications faciles et rapides, nous avons décidé de mettre au point un éditeur-écran qui élimine les commandes d'édition, en utilisant au maximum le déplacement du curseur.

Nous n'avons pas voulu limiter cet utilitaire à une édition pure et simple des instructions, mais nous lui avons octroyé trois caractéristiques supplémentaires. Le numéro de ligne est géré automatiquement sans tomber dans le travers d'une numérotation simplement croissante qui entraine une disparité entre la visualisation à l'écran et un listing papier préalablement édité. Pour ce faire, une suppression de ligne ne touchera pas aux numéros des lignes suivantes et une ligne insérée posséde le même numéro que la ligne précédente. Une commande de renumérotation permet de regénerer des numéros de ligne adjacents et croissants.

D'autre part, la présentation d'une ligne est gérée automatiquement par le système dès l'édition en particulier de ce qui concerne l'étiquetage pour les ruptures de séquence. Aussi une ligne éditée est alors constituée comme suit :



Enfin le nombre le plus important d'erreurs de programmation étant constitué par les erreurs de syntaxe, nous avons décidé d'analyser syntaxiquement chaque instruction dès son édition et de visualiser immédiatement les erreurs par l'affichage d'un message et le positionnement du spot sur la première faute détectée.

## 5. METTEUR AU POINT

La phase de mise au point des procédures doit être conçue pour permettre à tout utilisateur de maîtriser l'exécution de ses programmes par des modes de fonctionnement pas à pas et de visualiser tous les objets manipulés.

Nous avons alors réalisé le metteur au point des sous-programmes LAMULE qui possède les caractéristiques suivantes :

- Exécution ou simulation de programmes
- Mode pas à pas
- Visualisation de variables arithmétiques ou géométriques et d'états intermédiaires
- Création de points d'arrêt d'exécution
- Modification de variables arithmétiques

### 5.1. Exécution - simulation de programmes

Pour éviter de générer des blocs machines obligatoirement vers l'armoire de commande, nous avons créé deux modes de traitement :

- L'exécution qui permet la génération des blocs machines à destination de l'armoire de commande.
- La simulation qui possède toutes les caractèristiques de l'exécution mais évite tout transfert de bloc machine.

### 5.2. Mode pas à pas

Pour mettre au point un programme il s'avère intéressant de suivre son comportement après chaque instruction ou séquence d'instructions, donc de disposer d'un mode pas à pas.

Compte tenu de la structure du macro-interprète un mode pas à pas intégral nous ferait pénétrer obligatoirement à l'intérieur de tous les sous-programmes de description des diverses procédures et instructions. Nous avons donc établi une commande du système qui "ouvre ou ferme" un sous-programme pour le mode pas à pas.Une procédure fermée sera considérée comme un seul pas.

Par contre, chaque instruction d'un sous-programme ouvert constituera un pas.

D'autre part la finalité du système étant la génération de blocs machines nous proposons deux modes de traitement pas à pas :

- Instruction par instruction
- Bloc à Bloc

Dans ce mode l'ouverture ou la fermeture de sous-programme n'a pas effet.

5.3. Visualisation

Le but essentiel d'une exécution en mode pas à pas s'explique par la volonté de vouloir suivre les changements d'états des divers objets manipulés par une procédure. Nous avons alors adjoint au système une série de commandes de visualisation qui permettent d'afficher à l'écran :

- Des variables arithmétiques systèmes ou symboliques
- Des éléménts géométriques
- La dernière instruction exécutée
- Le dernier bloc-machine calculé
- La dernière forme canonique affectée

Notons que la valeur d'indétermination sera représentée par quatre étoiles.

Exemple :

P1 = **** ****

Signifie au programmeur que le point P1 est indéterminé.

La mise au point d'un programme nécessite souvent la visualisation itérative de certaines variables à chaque pas d'exécution. Aussi, afin d'éviter la répétition des commandes de visualisation, nous avons mis au point une procédure qui réalise automatiquement à chaque pas, l'affichage des paramètres rassemblés dans ce que nous appelons la macro de visualisation.

Exemple :

%VIS: $IN, P1, L1, DIA

Déclaration de la macro de visualisation composée :

- De la dernière instruction
- Du point P1
- De la droite L1
- De la variable DIA

### 5.4. Point de lancement / Point d'arrêt

Nous permettons à l'utilisateur d'exécuter ou de simuler une partie de programme à l'aide d'une commande précisant les numéros des instructions de début et de fin. Ceci à l'avantage d'autoriser le traitement d'une série d'instructions puis la reprise en mode pas à pas des séquences qui posent problème au programmeur.

### 5.5. Modification de variables arithmétiques

Pour faciliter la mise au point des sous-programmes de base nous avons défini une commande d'affectation des variables arithmétiques qui permet la modification ou la création d'une variable erronnée ou inconnue et la reprise de l'exécution du programme en cours.

## III ANALYSE GENERALE DU SYSTEME LAMULE

### 1. REPRESENTATION SCHEMATIQUE DU SYSTEME

La figure N°5 présente l'ensemble des processeurs du système LAMULE ainsi que les éléments mis en oeuvre par ceux-ci.

Nous pouvons ainsi distinguer :

- Le noyau assembleur formé :
  - * D'un module d'analyse des commandes du système de programmation
  - * Du macro-interprète et du metteur au point associé
  - * De l'éditeur de texte
  - * Du module d'impression et d'archivage

- La zone système formée des procédures de base d'adaptation au fraisage

- La zone utilisateur composée des programmes catalogués et du programme pièce

- L'ensemble des objets du système :
  - * Tables des valeurs des éléments arithmétiques
    - . Variables systèmes
    - . Variables symboliques
  - * Tables des formes canoniques des éléments géométriques
    - . Points
    - . Droites
    - . Cercles
    - . Structures de points
    - . Transformations géométriques
  - * Tables des macro-instructions
    - . Macros explicites
    - . Macros implicites

Nous trouvons sur la figure N° 6 le découpage de la mémoire centrale. Nous distinguons quatre zones principales parmi lesquelles la zone 3 est celle qui est préservée sur disque pour sauvegarder le version adaptée au problème à résoudre. (Dans notre cas il s'agira du problème spécifique de fraisage ).

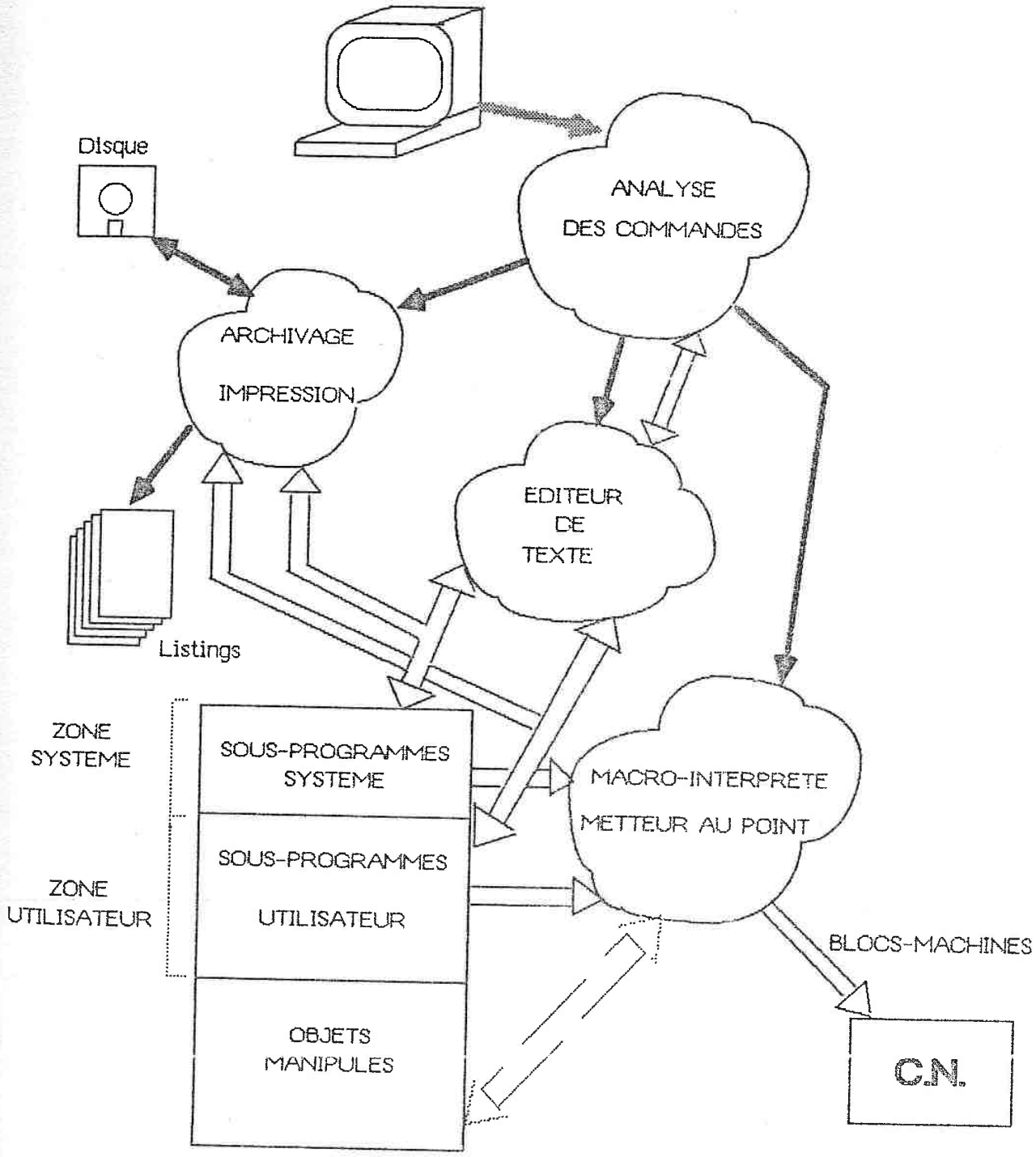



*Figure N° 5* : Représentation schématique du système

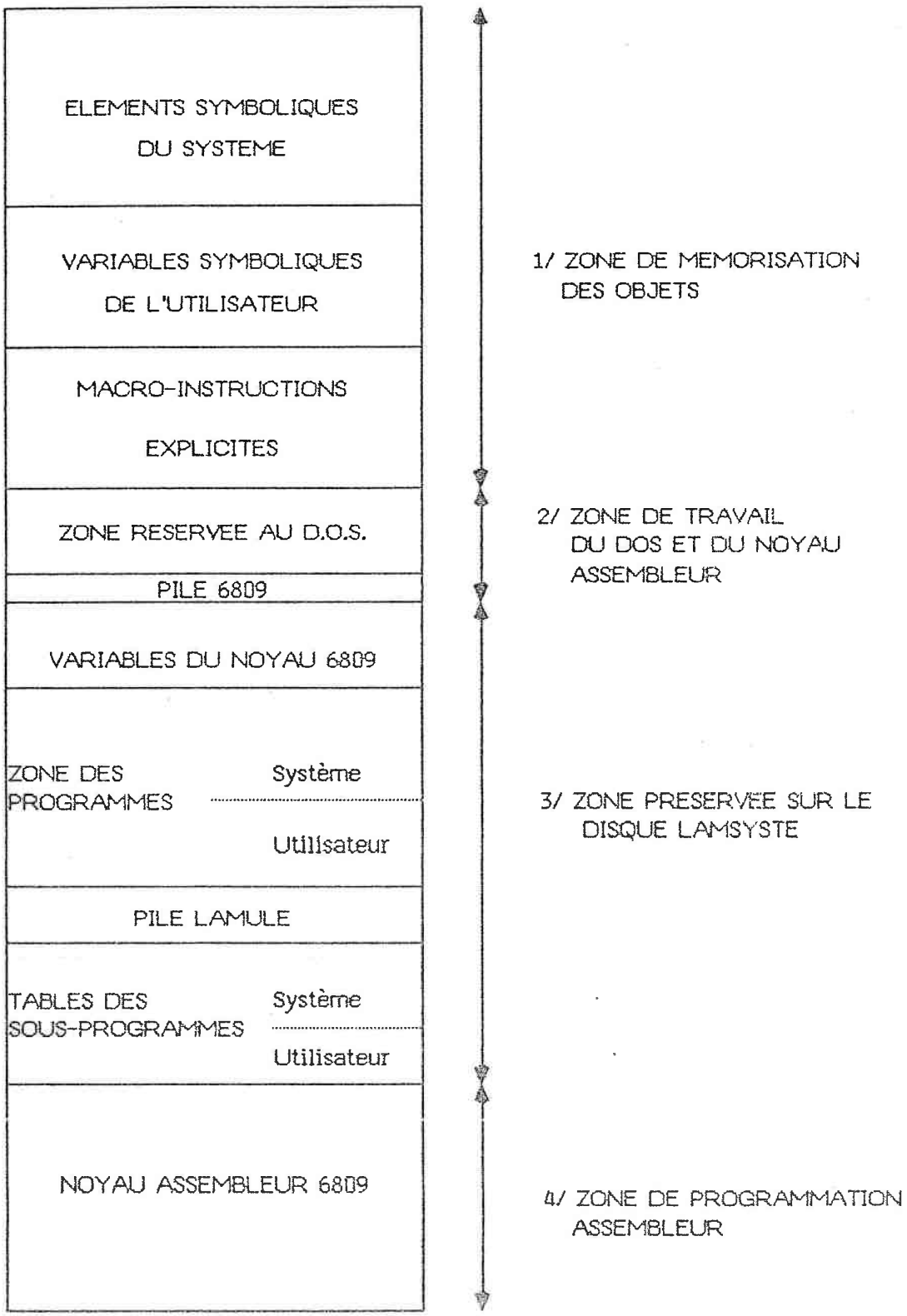



*Figure N° 6* : Implantation en mémoire du système LAMULE

## 2. NOMBRE MAXIMAL D'OBJETS DU SYSTEME

Tous les objets manipulés par le macro-interprète étant présents simultanément en mémoire centrale, il est nécessaire de définir une méthode de gestion de leur emplacement.

Trois solutions peuvent être envisagées :

- Une allocation statique à la génération du système
- Une allocation statique définie par une instruction spécialisée qui doit être programmée avant toute autre instruction
- Une allocation dynamique

Cette dernière solution semble très satisfaisante mais coûte très cher en logiciel de traitement et en temps d'exécution.La deuxième solution n'a pas été retenue car il aurait alors fallu distinguer les objets utilisateurs et systèmes, ce qui aurait alourdi le macro-interprète. C'est pourquoi nous avons retenu la première solution en proposant un nombre d'objets qui satisferont la majorité des utilisateurs.

Les tableaux suivants présentent le nombre maximal d'objets dans chaque type.

## 3. STRUCTURE GENERALE DU NOYAU ASSEMBLEUR

L'organigramme de la figure N° 7 présente la structure générale du noyau assembleur.

A l'instant initial le système est en attente d'une commande dont nous distinguons deux grands types :

- Les commandes ne concernant ni l'édition, ni l'exécution du programme, par exemple les commandes d'impression, d'archivage, de visualisation.
- Les commandes d'exécution. Le module d'exécution assure le traitement de toutes les instructions jusqu'au prochain point d'arrêt ou au prochain pas.

Il faut noter que nous avons créé implicitement deux modes de fonctionnement, le mode d'exécution / simulation et le mode général correspondant à tous les autres états du système, afin de pouvoir vérouiller des commandes en fonction de ce mode. Par exemple, il n'est pas possible d'effectuer des commandes d'archivage ou d'impression, lorsque l'on se trouve en mode exécution.

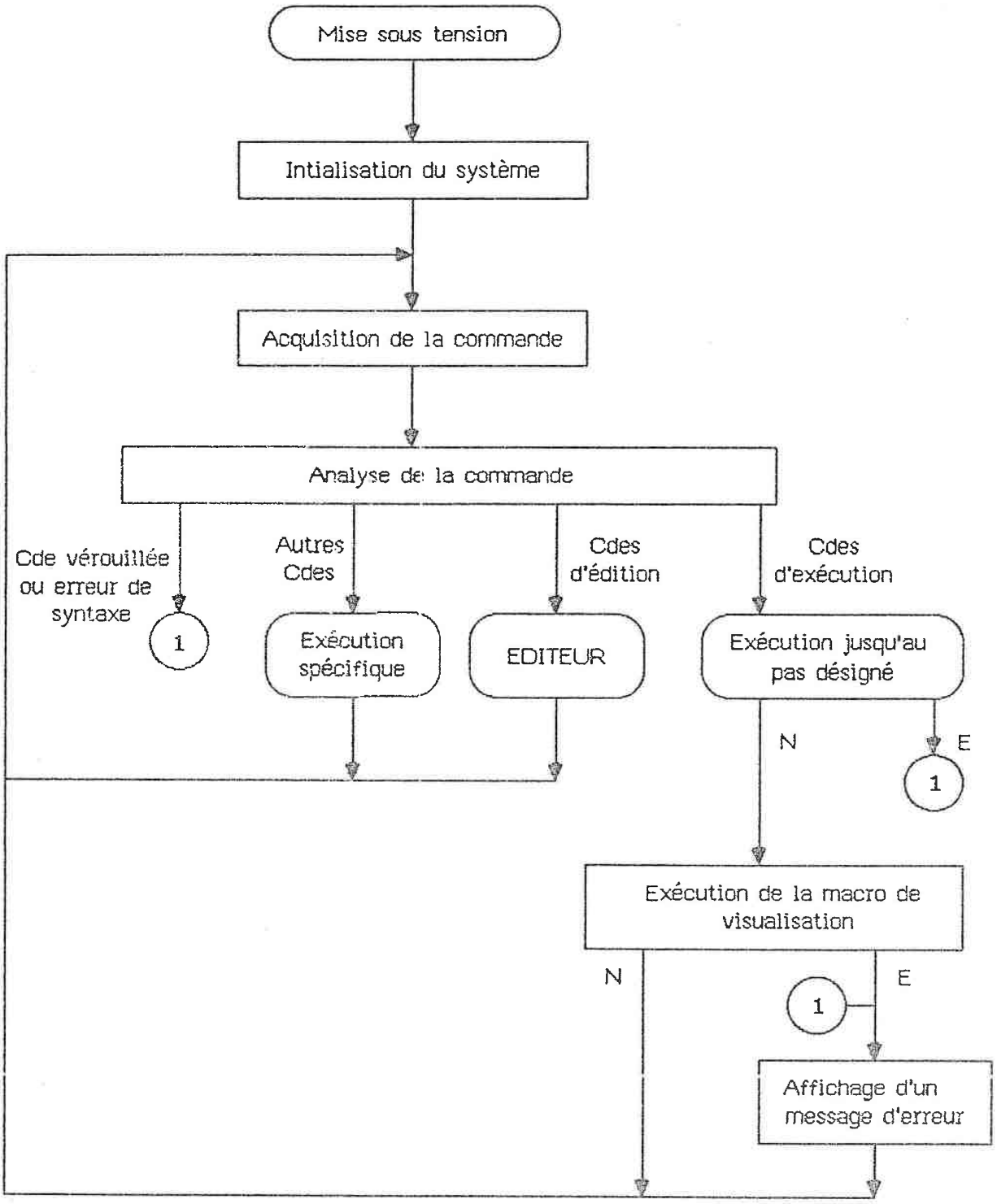



*Figure N° 7* : Analyse générale du noyau LAMULE

TABLEAU N°8 : NOMBRE MAXIMAL DES ELEMENTS GEOMETRIQUES

| Elément | Symbole | Nombre maximal |
|---|---|---|
| Point | P | 100 |
| | O | 8 |
| Droite | L | 100 |
| | K | 20 |
| Cercle | C | 100 |
| | B | 20 |
| Structure de points | S | 20 |
| | R | 5 |
| Transformations géométriques | T | 10 |

TABLEAU N°9 : NOMBRE MAXIMAL DE VARIABLES ARITHMETIQUES

| Variable arithmètique | Symbole | Nombre maximal |
|---|---|---|
| Variable système | V | 200 |
| Variable symbolique | | 100 |

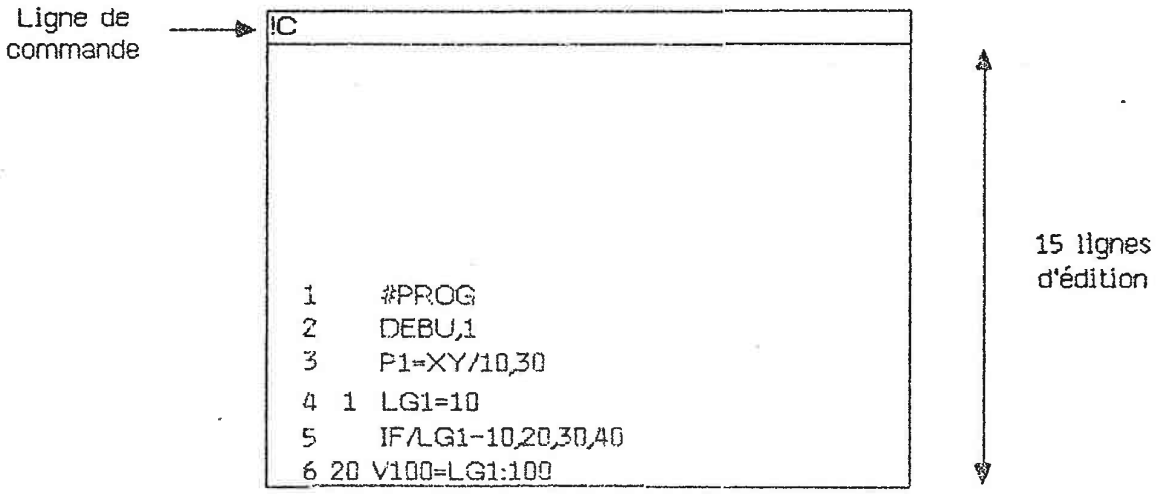
TABLEAU N°10 : NOMBRE MAXIMAL DE MACRO-INSTRUCTIONS

| Macro-instuction | Symbole | Nombre maximal |
|---|---|---|
| Macro explicite | | 100 |
| Macro implicite | M | 10 |

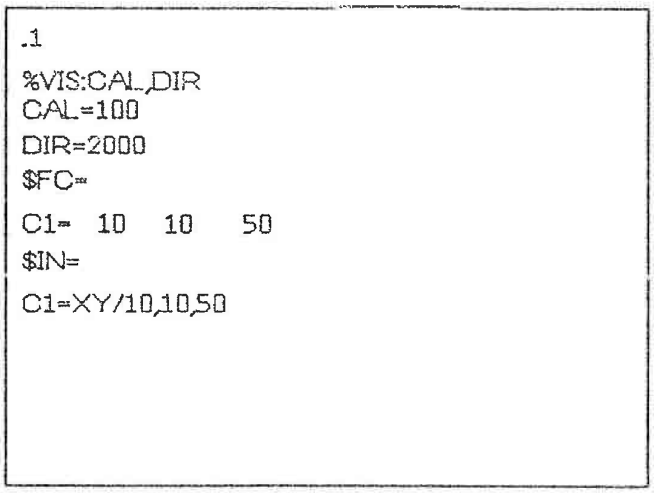
Nous gérons l'écran de visualisation de deux manières distinctes suivant le mode: ( *Cf Figure N°8* )

- En mode exécution / simulation, l'écran est géré en défilement rouleau
- Dans les autres modes, l'écran comporte une ligne commande et 15 lignes correspondant au texte à éditer.

*Figure N° 8 :* Gestion de l'écran en mode général




Gestion de l'écran en mode exécution / visualisation



## 4. REPRESENTATION INTERNE DES INSTRUCTIONS

Un programme est représenté en mémoire par la suite :

- Instruction de définition d'en-tête du sous-programme
- Instructions formant le corps
- Instruction de fin de sous-programme

D'autre part, à chaque création de sous-programme, le nom de celui-ci est stocké dans une table de symboles, associé à l'adresse de l'entête du sous-programme. Nous avons créé deux tables distinctes de symboles pour les sous-programmes (sous-programmes systèmes et sous-programmes utilisateur), pour permettre un traitement différent de ces deux catégories, comme nous allons le voir.

Le mode de stockage de chaque instruction est un mode "caractères" auquel nous avons associé un préambule et un caractère de fin. [ 9 ]

Le préambule a été conçu pour atteindre les quatre objectifs suivants :

- La rapidité d'édition, pour laquelle nous avons introduit un octet de chainage avant.
- La rapidité d'exécution, pour laquelle nous avons créé un octet qui repère l'emplacement dans la table des symboles du nom du sous-programme appelé par l'instruction. Ceci permet d'accéder très rapidement au corps du sous-programme correspondant. Toutefois, pour que cet objectif ne soit pas atteint au détriment de la facilité d'édition des programmes et sous-programmes utilisateur, nous nous sommes limités à ce mode d'accès uniquement pour les sous-programmes systèmes.
- Le stockage de l'étiquette éventuelle ( utilisée pour les ruptures de séquence).
- La distinction entre les instructions d'affectation et les autres.

Le préambule est créé automatiquement au moment de l'écriture ou de la modification de l'instruction.

## 5. EDITION DE TEXTE ET ANALYSE SYNTAXIQUE

L'édition de texte est activée dès la frappe d'une commande d'édition ou lors du passage de la ligne commande à la zone de texte visualisée ( Commande HOME ou 'Lf'). La figure N°9 présente l'organigramme général de l'éditeur.

L'analyse syntaxique d'une instruction est effectuée dès l'écriture ou la modification de celle-ci et comporte plusieurs phases comme le présente l'organigramme de la figure N°10. La phase principale de l'analyse est constituée par la recherche de correspondance de type, entre les paramètres formels de la définition d'un sous-programme et les paramètres effectifs de l'appel de ce dernier. Toute instruction formée d'un nom symbolique différent d'un nom de sous-programme système ou utilisateur est considérée comme l'appel d'une macro-instruction explicite. En effet, la définition d'une macro-instruction est effectuée à l'exécution, lors du traitement de l'instruction d'en-tête de la macro. Nous ne pouvons donc pas détecter l'appel d'une macro inexistante, lors de la phase d'édition.

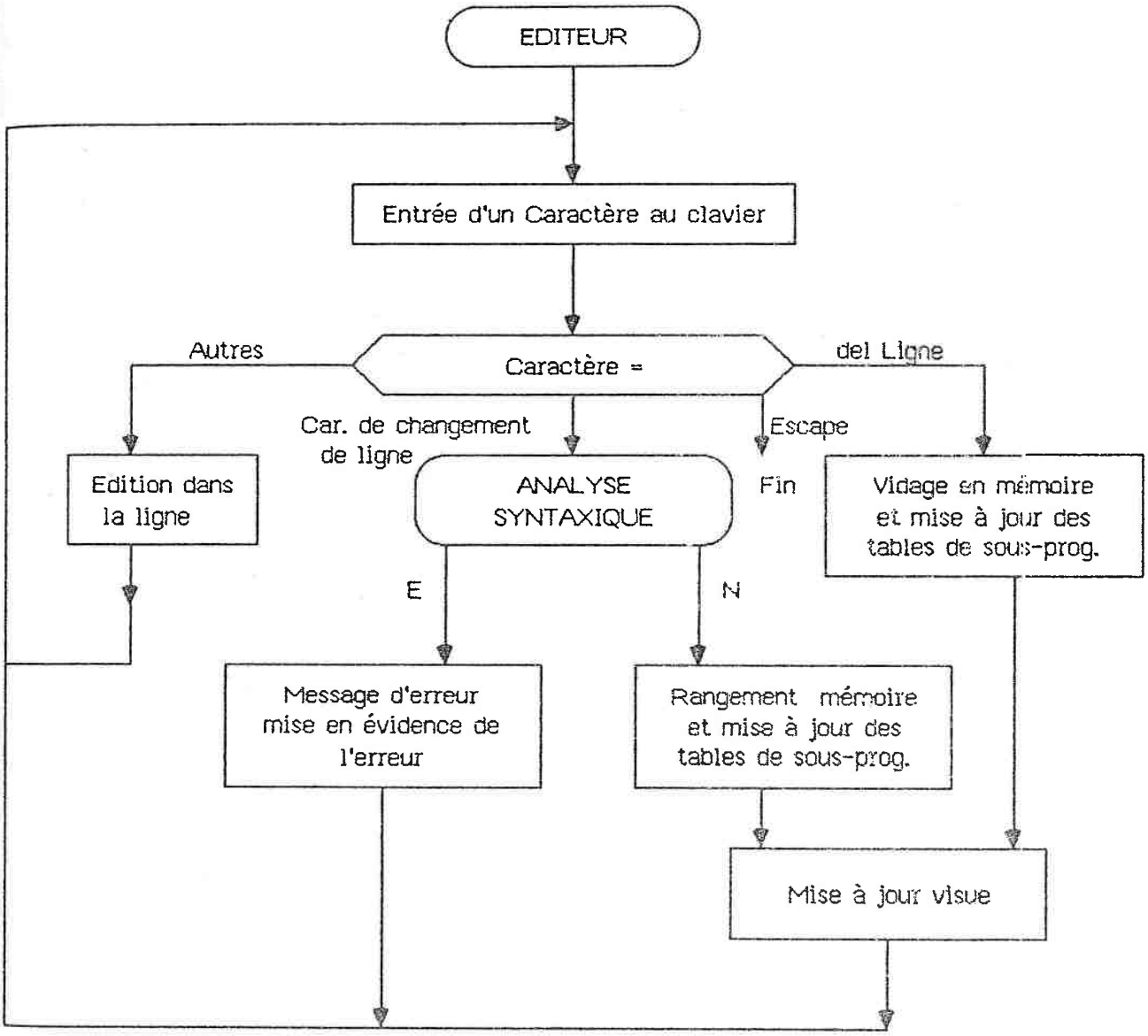



*Figure N° 9*: Organigramme de l'éditeur de texte

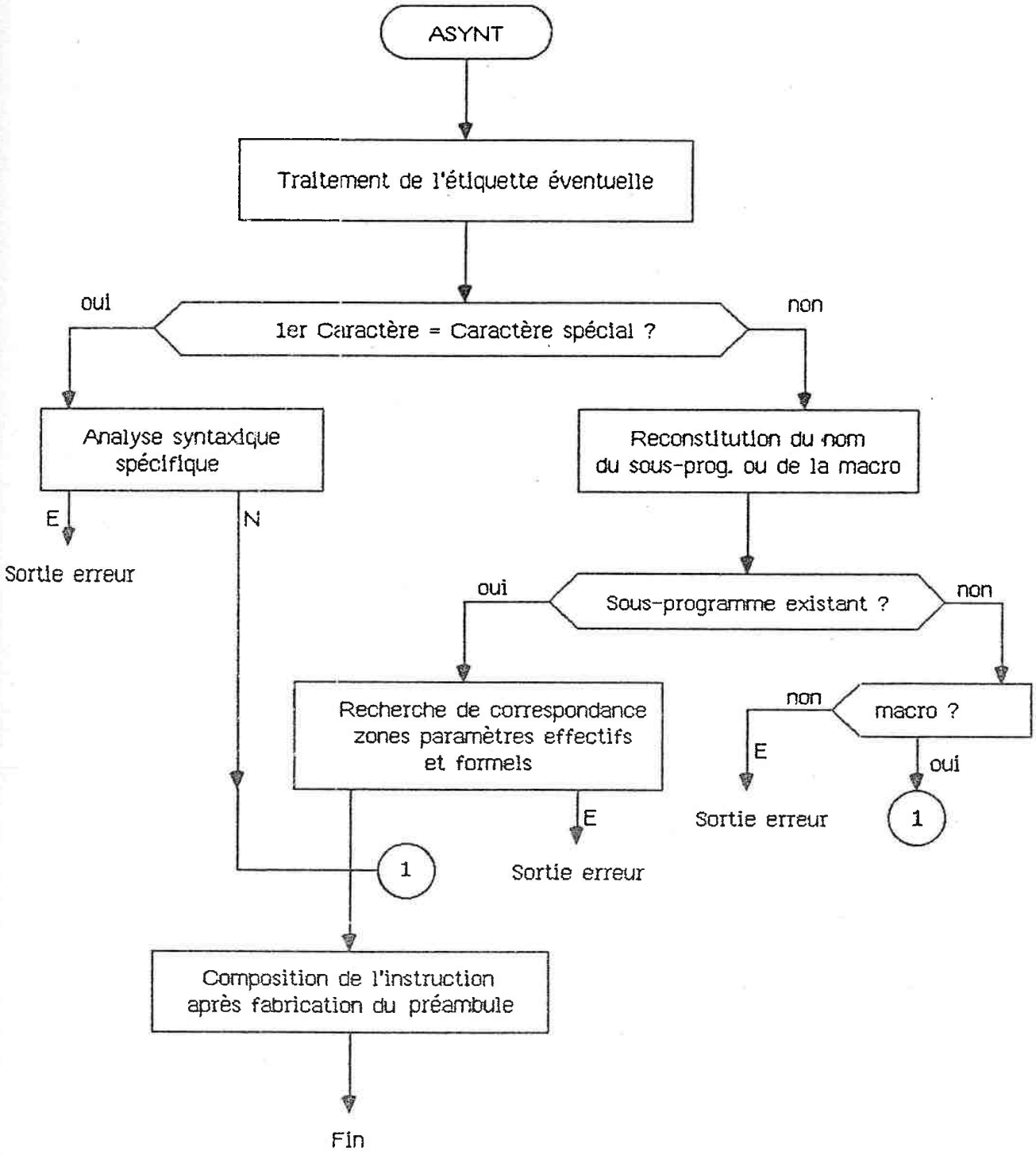



*Figure N° 10*: Organigramme de l'analyse syntaxique

## 6. INTERPRETATION DES INSTRUCTIONS LAMULE

Nous rappelons que le traitement d'une instruction LAMULE ( à part les primitives), revient à l'exécution du sous-programme système correspondant, après passage des paramètres.

Nous pouvons distinguer cependant trois types d'instructions :

- Les instructions avec affectation autres que les définitions de structures et transformations pour lesquelles la fin de sous-programme assurera automatiquement le transfert des variables V50, V51 et V52 dans la forme canonique de l'élément à affecter.

- Les instructions de définition des transformations et des structures, pour lesquelles le traitement est très particulier. En effet, le sous-programme correspondant n'est pas exécuté tout de suite, mais seule l'adresse de l'instruction est stockée, dans la forme canonique de la structure ou de la transformation correspondante. L'appel véritable du sous-programme se fera lors de l'utilisation de la structure ou la transformation géométrique à l'aide de deux primitives assembleur spéciales @AR1 et @TR.

- Les instructions sans affectation ne posent pas de problème particulier.

Dans tous les cas pour permettre des instructions d'un nombre quelconque de sous-programmes ou de macros, nous avons défini la pile LAMULE dans laquelle seront sauvegardés, puis restitués, tous les contextes des sous-programmes et instructions appelant. [ 8 ]

Empilage d'un sous-programme

Le contexte d'appel d'un sous-programme est constitué par des pointeurs assembleur, des noms d'éléments du système et des valeurs arithmétiques qui sont :

- Un indicateur qui précise que c'est un sous-programme qui est empilé
- L'adresse de début du sous-programme appelant
- L'adresse de l'instruction à exécuter au retour de l'appel
- Un indicateur d'ouverture ou de fermeture du sous-programme appelant
- Un indicateur qui précise la présence éventuelle d'un élément d'affectation
- L'endroit courant d'exploitation, en temps différé, de la zone de paramètres effectifs
- Les valeurs actuelles des variables V50, V51 et V52
- Le nom de l'éventuel élément à affecter au retour du sous-programme appelé.

D'autre part, pour limiter le temps de recherche des étiquettes lors des ruptures de séquence, nous gérons, en haut de pile, une table des étiquettes qui est remplie au fur et à mesure des instructions de branchement.

Lors d'une telle instruction, la démarche est alors la suivante :

- Recherche de l'étiquette dans la table actuelle
- Sinon recherche de cette étiquette dans le corps du sous-programme à partir de la dernière étiquette de la table, et mise à jour de la table des étiquettes.

Empilage d'une macro-instruction

Nous rappelons qu'une macro-instruction ne regroupe que temporairement une série d'instructions et n'effectue en aucun cas des passages de paramètres. Aussi l'empilage d'une macro-instruction comporte simplement :

- Un indicateur de type qui différencie une macro-instruction explicite, d'une macro implicite.
- L'adresse de l'instruction de reprise de séquence après le traitement de cette macro.

La pile LAMULE profite de la place mémoire disponible derrière le programme utilisateur. Pour détecter la collision éventuelle entre ces deux éléments, nous gérons une variable qui représente, au moment de l'exécution, la place maximale encore admissible, et positionnons l'indicateur d'erreur dans le cas d'un remplissage total de la pile.

Les figures suivantes représentent les organigrammes de principes du traitement d'une instruction.

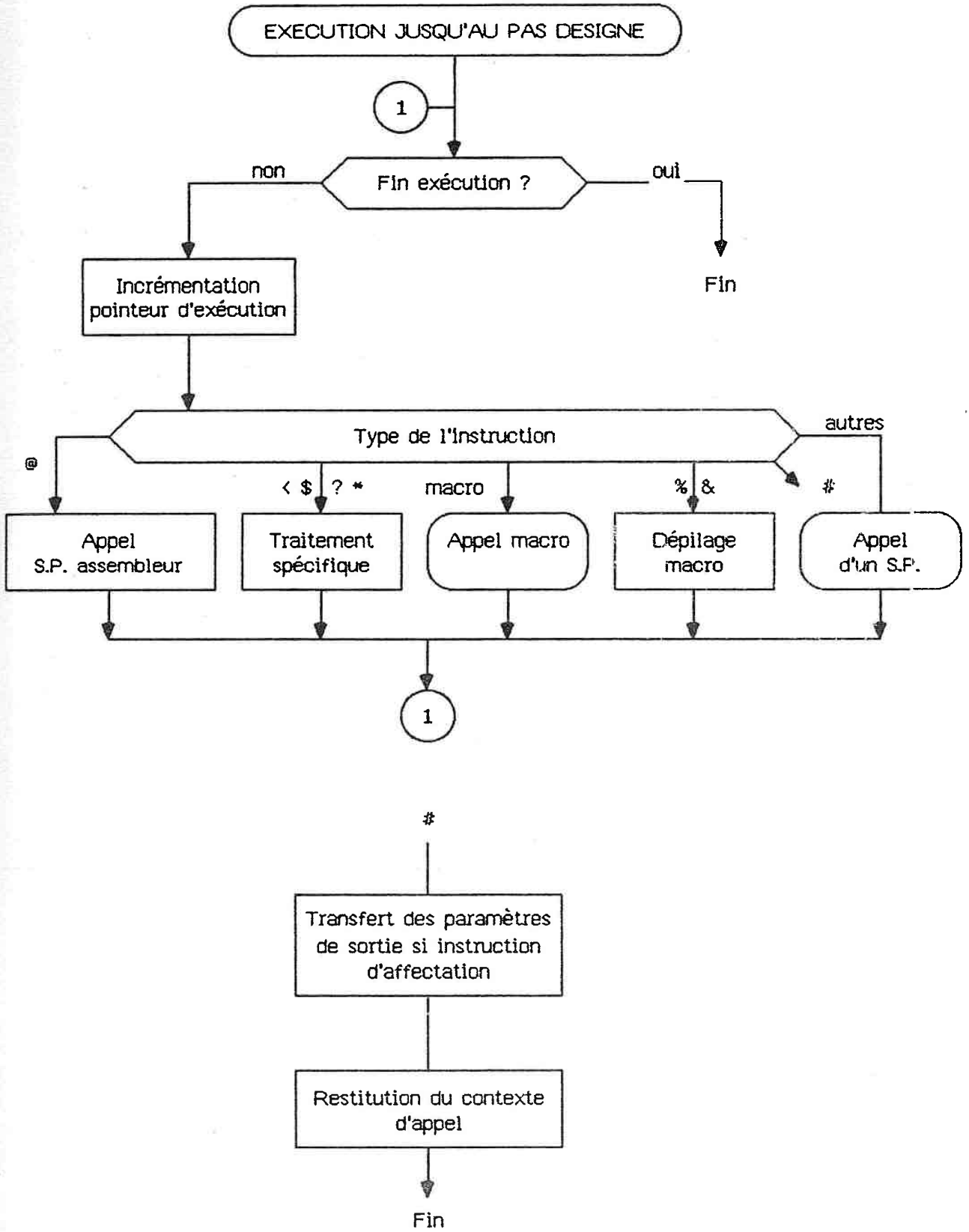
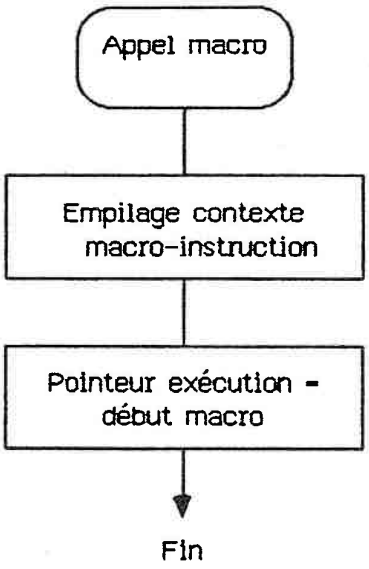



*Figure N° 11* : Organigramme de l'exécution



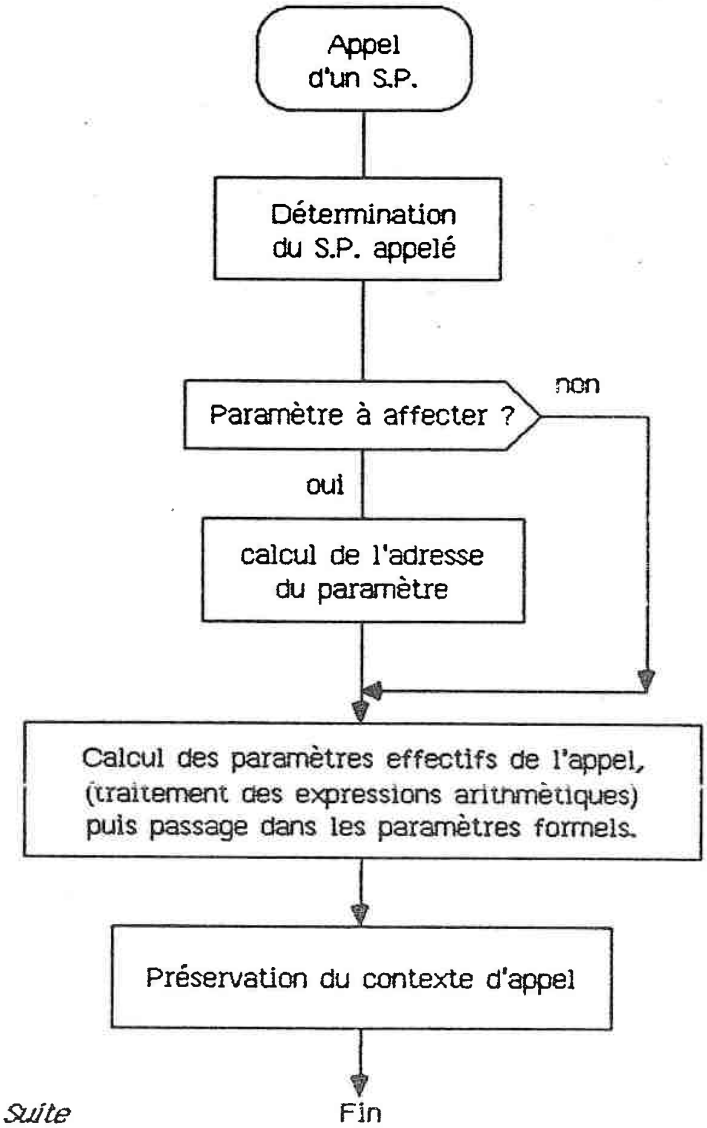




*Figure N° 11 Suite*

# CHAPITRE 3

# ANALYSE GENERALE DU SYSTEME LAMULE

## 1. PRINCIPES RETENUS

L'utilisation de l'interprète LAMULE pour la réalisation d'un système de commande directe d'une fraiseuse en langage évolué, nous a amené à définir un certain nombre de principes dont le but essentiel est de faciliter la structuration des programmes de l'utilisateur et de simplifier les appels de procédures d'usinages.

Ces principes sont :

- Un programme pièce est constitué par la définition séquentielle des données technologiques, des éléments géométriques de la pièce, puis des ordres d'usinage ou de déplacements.

- Les paramètres techonologiques d'usinage sont associés à un outil et regroupés dans ces macro-instructions explicites spéciales.

- Les transformations géométriques préalablement définies sont appelées automatiquement à chaque instruction de déplacement élémentaire.

- La génération des blocs machines est intégrée dans ces mêmes procédures de mouvements élémentaires.

## 2. STRUCTURE DU PROGRAMME D'USINAGE

Par soucis de structuration de la zone utilisateur et pour homogénéiser les différentes commandes d'édition et de mise au point des programmes et sous-programmes de l'utilisateur, nous avons décidé de donner au programme principal un nom réservé ( PROG ).

La structure classique de ce programme sera constituée successivement par :

- Des macro-instructions TOi ( *Cf paragraphe suivant*)
- Des définitions des éléments géométriques constitutifs de la pièce à fabriquer
- Des ordres d'usinage

La représentation schématique de la zone de programmation de l'utilisateur s'établit alors comme suit :

```
[    Zone des sous-programmes
[            catalogués
[_____________________________
[
[    Zone des sous-programmes
[            Utilisateur
[_____________________________
#PROG
%TO1          [
-             [
%             [
-             [  Définitions technologiques
%TOn          [
-             [
%             [

_____________________
L1=P1,P2      [
C1=L1,L2,10   [  Définitions géométriques
-             [
--            [

_____________________
GT/0,0,0      [
-             [  Ordres de mouvements
-             [

_____________________
#
```

Cette structure est la plus naturelle mais n'est pas absoluement obligatoire, la seule contrainte est que toute utilisation d'un paramètre ou d'une macro-instruction doit être précédée par sa déclaration.

### 3. PARAMETRES TECHNOLOGIQUES D'USINAGE

Nous avons pris l'option d'associer l'ensemble des paramètres technologiques d'un usinage à l'outil utilisé pour celui-ci. Pour ce faire, nous avons créé des macro-instructions explicites dont le nom symbolique est TOi suivi d'un nombre compris entre 0 et 99, que nous réservons à la définition des paramètres technologiques.

Ces données représentées par des variables symboliques sont de deux sortes :

Les paramètres descriptifs d'outil

- NO numéro d'outil
- LG longueur d'outil
- LU longueur utile d'outil
- DIA diamètre d'outil
- PAS pas pour un taraud

Les paramètres technologiques associés à l'outil

- VA vitesse d'avance
- VR vitesse de rotation
- VC vitesse de coupe
- TPO période de temporisation

Exemple : Définition d'une macro technologique associée à l'outil N° 1.

```
%TO2
NO=1
LG=251.4
LU=246.3
VA=450
VR=1600
%
```

Les paramètres que nous venons de décrire ne sont absolument pas exhaustifs et leur nom n'est pas imposé par l'interprète LAMULE, mais par les sous-programmes de génération des usinages. Ainsi d'autres paramètres pourront être introduits pour satisfaire aux autres procédures d'usinages, qui seront définies par la suite.

Exemple :

PM : profondeur de perçage maximale avant débourage pour un cycle de perçage profond.

## 4. PROCEDURES DE GENERATION DES BLOCS MACHINES

La génération des blocs est effectuée par les instructions spéciales * et ? ( *Cf Chapitre II paragraphe I 6* ), fonction de l'armoire de commande à laquelle sera destiné le programme.

Notre système n'étant pas à priori un langage d'aide à la programmation, mais un système de commande directe d'une fraiseuse, nous n'avons pas regroupé ces instructions spéciales sous forme de module de jonction. Toutefois, pour permettre une adaptation simple à une nouvelle armoire de commande, et répondre au souci de modularité, du sytème nous avons essayé de réduire au maximum le nombre d'endroits où sont utilisés ces instructions de génération de blocs. Pour ce faire, nous avons décidé de définir les procédures élémentaires suivantes qui seront appelées par les sous-programmes relatifs aux fonctions d'usinage évoluées :

- Déplacements élémentaires ( segments de droites ou arcs de cercles )
- Changement de correction d'outil et/ou changement d'outil.

Aussi seules, ces procédures élémentaires utiliseront des instructions de génération de bloc machine.

## 5. PROCEDURES GEOMETRIQUES

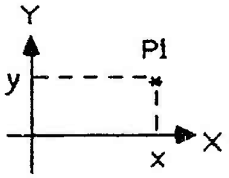
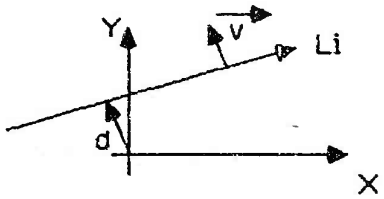
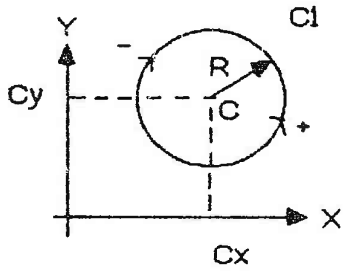
### 5.1. Eléments utilisés par la géométrie

La géométrie planaire orientée utilisée, affecte à chaque élément du système, une représentation analytique que nous appelons forme canonique.

Le tableau N° 11 présente la forme canonique des éléments utilisés.

Rappelons que les définitions géométriques constituent implicitement l'appel d'un sous-programme système ( *Cf Chapitre II paragraphe I4* ) et que l'analyse syntaxique est effectuée dès l'écriture de l'instruction en particulier par comparaison du type des paramètres formels et effectifs ( *Cf Chapitre II paragraphe II4* ).

TABLEAU N°11 : FORMES CANONIQUES DES OBJETS GEOMETRIQUES

| Type d'élément | Forme canonique |
|---|---|
| Point | - Abcisse<br>- Ordonnée<br>  |
| Droite | - Coordonnées du vecteur directeur $\vec{v}$<br>- Distance à l'origine orientée d<br>  |
| Cercle | - Coordonnées du centre du cercle<br>- Rayon orienté<br>  |
| Structure de points | Adresse de définition |
| Transformation géométrique | Adresse de définition |

Les sous-programmes de calculs géométriques utilisent largement les procédures spéciales prévues dans l'interprète LAMULE :

- Procédures d'affectation de variables arithmétiques systèmes par des formes canoniques d'éléments géométriques.
  Exemple : V70 = P1 Réalise V70 = P1x
  V71 = P1y

- Procédures arithmétiques de base telles que, valeur absolue, entière, fonction trigonométrique, racine carrée, etc...

- Procédures de rupture de séquence.

5.2. Points, Droites, Cercles

L'exécution de définition de points, droites et cercles est effectuée dès leur appel, et correspond au calcul de la forme canonique de l'élément géométrique [7]. Le sous-programme correspondant effectue successivement les opérations suivantes :

- Passage des paramètres effectifs dans les paramètres formels avec évaluation éventuelle des expressions arithmétiques. ( *Cf Chapitre II paragraphe I4* )
- Calcul de la forme canonique de l'élément géométrique, le résultat étant stocké dans les varaibles V50, V51 et V52. ( *Cf Chapitre II paragraphe I4* )
- Transfert de ces valeurs aux places mémoires reservées à la forme canonique de l'élément défini.

Exemple :

Définition du point P1 centre du cercle C10.

L'instruction d'appel s'établit par P1=CE/ C10 et correspond à l'appel du sous-programme PCE/ .

Ce sous-programme est constitué comme suit :

```
#PCE/,B0
<ERR 1
V50 = B0
#
```

A l'appel de la procédure C10 est transféré dans B0.

En cas de non existence le système affiche le message d'erreur < ERR 1 ( non définition ).

Sinon, l'instruction V50 = B0 réalise le passage des coordonnées du centre et du rayon du cercle B0 ( C10 ) dans les variables V50, V51 et V52 respectivement.

Le caractère ' # ' (fin de sous-programme) transfère V50 et V51 dans la forme canonique de P1.

Ainsi P1x = Abcisse du centre du cercle C10
P1y = Ordonnée du centre du cercle C10

Dans cet exemple, le sous-programme est limité à de simples transferts de valeurs, mais la plupart des sous-programmes géométriques correspondront aux calculs analytiques de détermination de la forme canonique de l'élément correspondant.

Les tableaux suivants présentent les définitions géométriques que nous avons retenues, et qui correspondent aux définitions généralement proposées par les différents systèmes existants que nous avons étudiés. [ 2 ]

Un point peut alors être défini de 8 manières possibles, un cercle et une droite de 10 manières.

TABLEAU N° 12 : DEFINITION DE POINTS

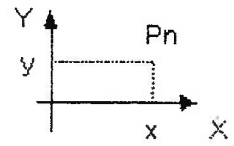
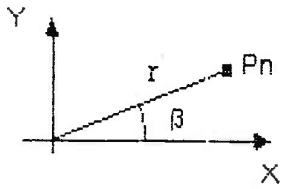
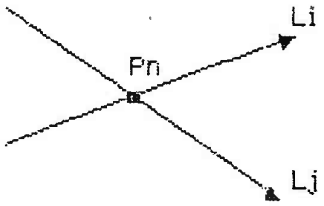
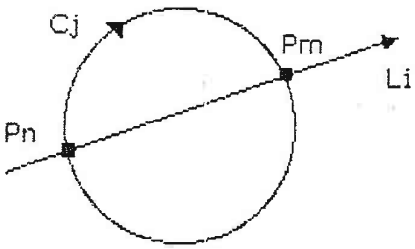
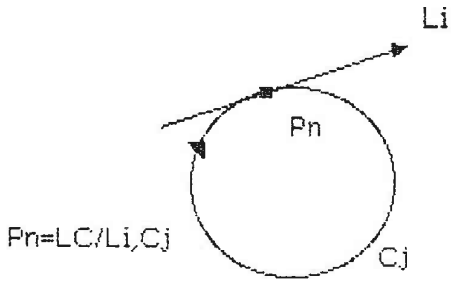
| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| COORDONNEES CARTESIENNES | Pn=XY/x,y<br>ou<br>Pn=x,y |  |
| COORDONNEES POLAIRES | Pn=RA/r,β |  |
| INTERSECTION DE DEUX DROITES | Pn=LL/Li,Lj<br>ou<br>Pn=Li,Lj |  |
| INTERSECTION D'UNE DROITE ET D'UN CERCLE | Pn=LC/Li,Cj (,m)<br>ou<br>Pn=Li,Cj (,m) |  <br>Pn=LC/Li,Cj,G<br>Pm=LC/Li,Cj,D<br>  <br>Pn=LC/Li,Cj |

TABLEAU N° 13 : DEFINITION DE POINTS SUITE

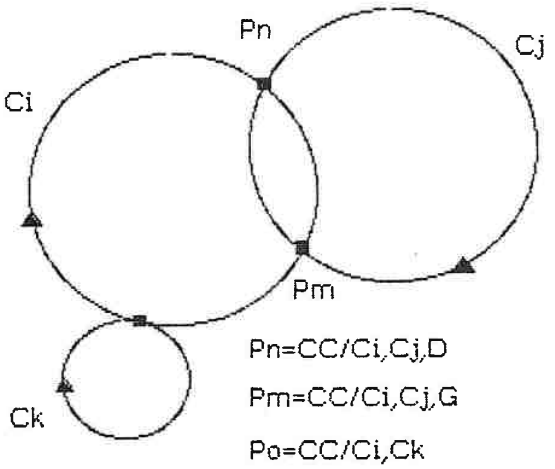
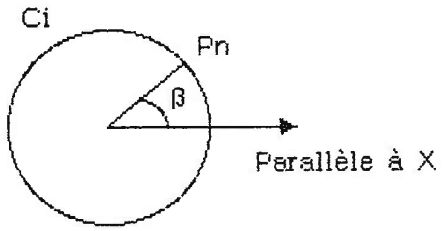
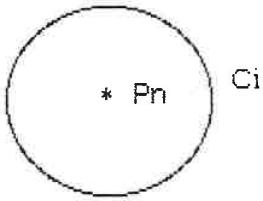
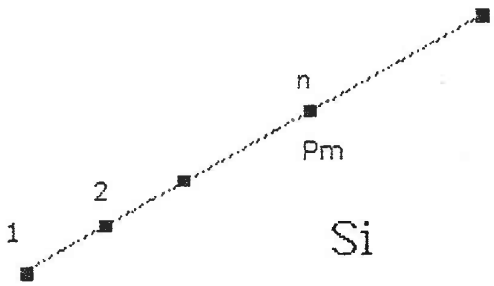
| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| INTERSECTION DE DEUX CERCLES | Pn=CC/Ci,Cj (,m)<br>ou<br>Pn=Ci,Cj |  |
| SITUE ANGULAIREMENT SUR UN CERCLE | Pn=CA/Ci,β<br>ou<br>Pn=Ci,β |  |
| CENTRE D'UN CERCLE | Pn=CE/Ci<br>ou<br>Pn=Ci |  |
| NIEME POINT D'UNE STUCTURE DE POINTS | Pm=OB/Si,n<br>ou<br>Pm=Si,n |  |

TABLEAU N° 14 : DEFINITION DE DROITES

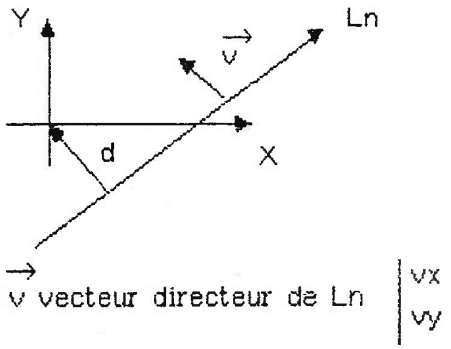
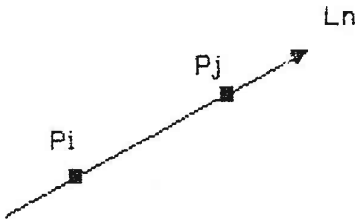
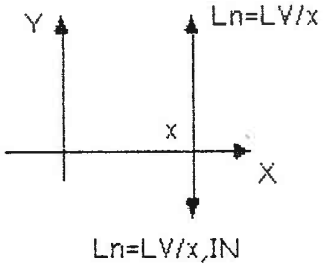
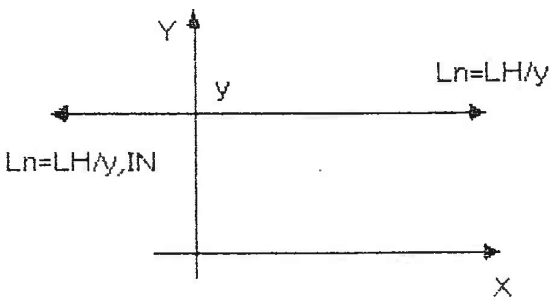
| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| FORME CANONIQUE | Ln=XY/vx,vy,d<br>ou<br>Ln=vx,vy,d |  |
| DROITE INVERSE | Ln=IN/Li |  |
| DEFINIE PAR DEUX POINTS | Ln=PP/Pi,Pj<br>ou<br>Ln=Pi,Pj |  |
| DROITE VERTICALE | Ln=LV/x, (IN) |  |
| DROITE HORIZONTALE | Ln=LH/y, (IN) |  |

TABLEAU N° 15 : DEFINITION DE DROITES SUITE

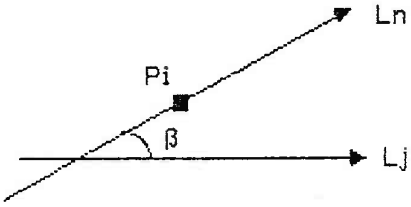
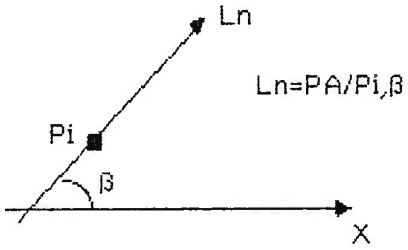
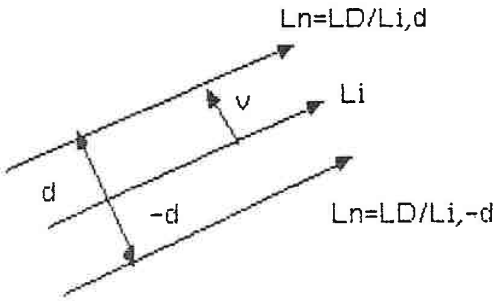
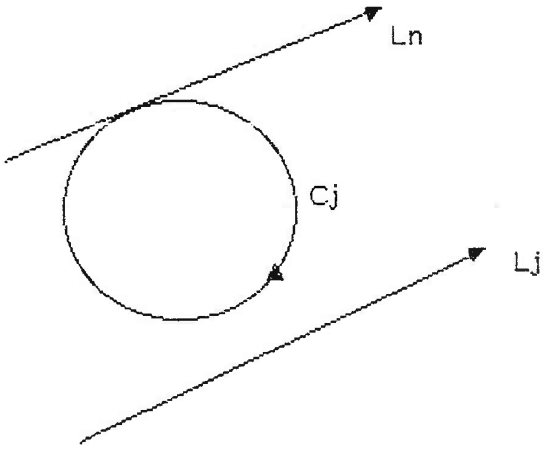
| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| PASSANT PAR UN POINT ET FAISANT UN ANGLE AVEC UNE DROITE | Ln=PA/Pi,β (,Lj)<br>ou<br>Ln=Pi,β (Lj) | Ln=PA/Pi,β,Lj<br>  <br>  |
| DROITE PARALLELE | Ln=LD/Li,d<br>ou<br>Ln=Li,d |  |
| PARALLELE A UNE DROITE ET TANGENTE A UN CERCLE | Ln=LC/Li,Cj<br>ou<br>Ln=Li,Cj |  |

TABLEAU N° 16 : DEFINITION DE DROITES SUITE

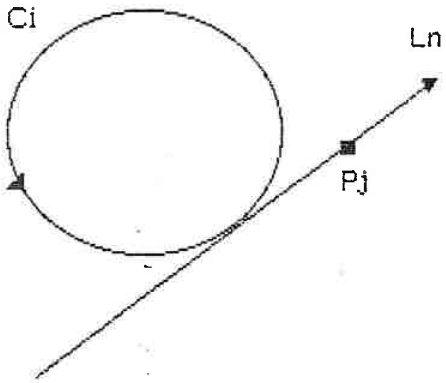
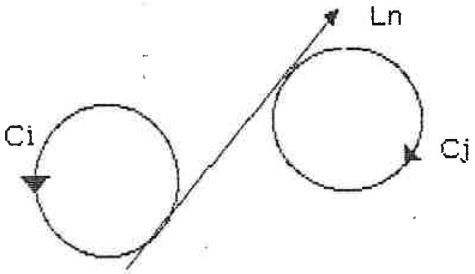
| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| TANGENTE A UN CERCLE ET PASSANT PAR UN POINT | Ln=CP/Ci,Pj<br>ou<br>Ln=Ci,Pj |  |
| TANGENTE A DEUX CERCLES | Ln=CC/Ci,Cj<br>ou<br>Ln=Ci,Cj |  |

TABLEAU N° 17 : DEFINITION DE CERCLES

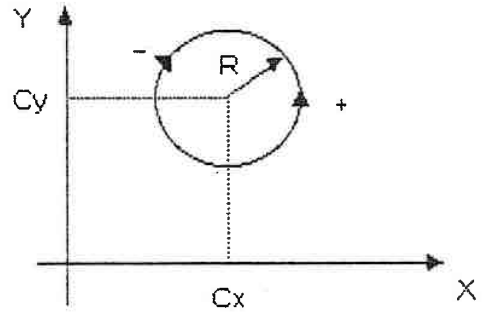
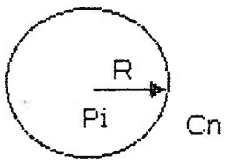
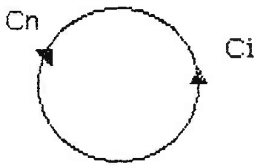
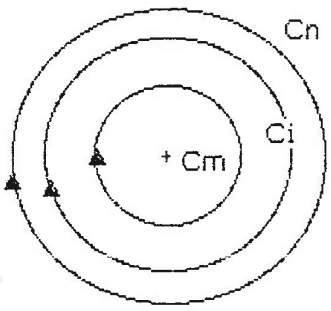
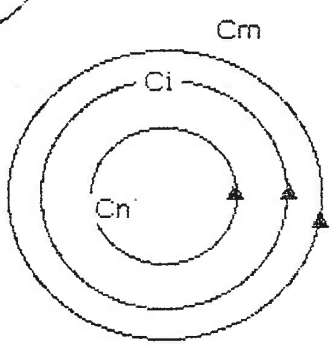
| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| FORME CANONIQUE | Cn=XY/Cx,Cy,R<br>ou<br>Cn=Cx,Cy,R |  |
| CENTRE ET RAYON | Cn=Cr/Pi,R<br>ou<br>Cn=Pi,R |  |
| CERCLE DE SENS INVERSE | Cn=IN/Ci |  |
| CERCLE CONCENTRIQUE | Cn=CD/Ci,△R<br>ou<br>Cn=Ci,△R |  <br>Cn=CD/Ci,△R<br>Cm=CD/Ci,-△R<br>  |

TABLEAU N° 18 : DEFINITION DE CERCLES SUITE

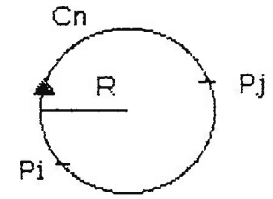
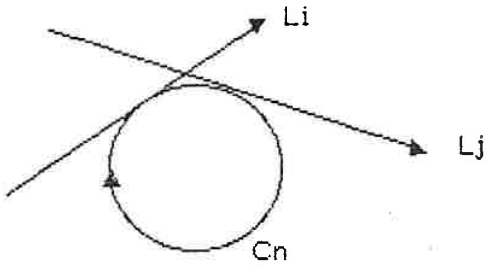
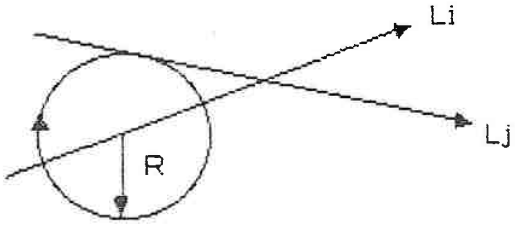
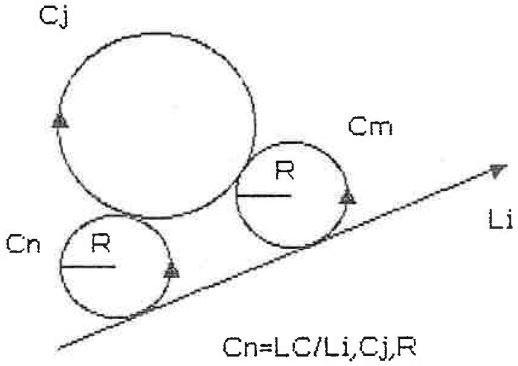
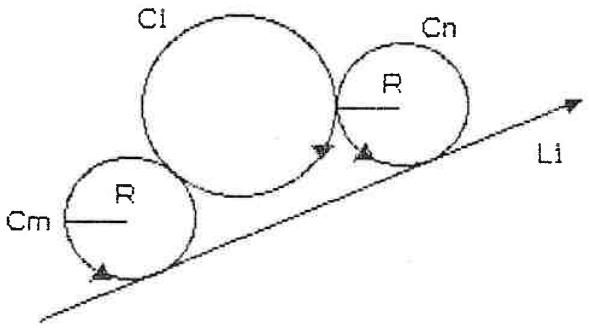
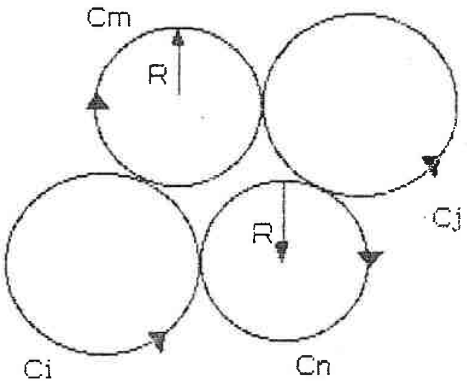
| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| PASSANT PAR 2 POINTS ET DE RAYON CONNU | Cn=PP/Pi,Pj,R<br>ou<br>Cn=Pi,Pj,R |  |
| TANGENT A 2 DROITES ET DE RAYON CONNU | Cn=LT/Li,Lj,R<br>ou<br>Cn=Li,Lj,R |  |
| CENTRE SUR UNE DROITE , TANGENT A UNE AUTRE ET DE RAYON CONNU | Cn=SL/Li,R,Lj<br>ou<br>Cn=Li,R,Lj |  |
| TANGENT A UNE DROITE ET A UN CERCLE , ET DE RAYON CONNU | Cn=LC/Li,Cj,R (,GR)<br>ou<br>Cn=Li,Cj,R, (GR) |  <br>Cn=LC/Li,Cj,R<br>Cm=LC/Li,Cj,R,GR |

TABLEAU N° 19 : DEFINITION DE CERCLES SUITE

| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| TANGENT A UN CERCLE , A UNE DROITE ET DE RAYON CONNU | Cn=CL/Ci,Lj,R (,GR)<br>ou<br>Cn=Ci,Lj,R (,GR) |  <br>Cn=CL/Ci,Lj,R<br>Cm=CL/Ci,Lj,R,GR |
| TANGENT A 2 CERCLES ET DE RAYON CONNU | Cn=CC/Ci,Cj,R (,GR)<br>ou<br>Cn=Ci,Cj,R (,GR) |  <br>Cn=CC/Ci,Cj,R<br>Cm=CC/Ci,Cj,R,GR |

Cette liste n'est naturellement pas limitative, car l'avantage principal du système LAMULE réside dans sa capacité d'adjonction de nouvelles instructions, sans avoir à toucher en quoi que ce soit au noyau assembleur.

Ainsi par exemple, ajouter l'instruction permettant de définir la droite passant par un point et parallèle à une autre droite, nécessiterait les différentes phases suivantes :

a. Création du nom et de la zone des paramètres qui vont imposer la syntaxe de l'instruction. Nous pourrions ici choisir une syntaxe de la forme :

Li = PL/Pj,Lk

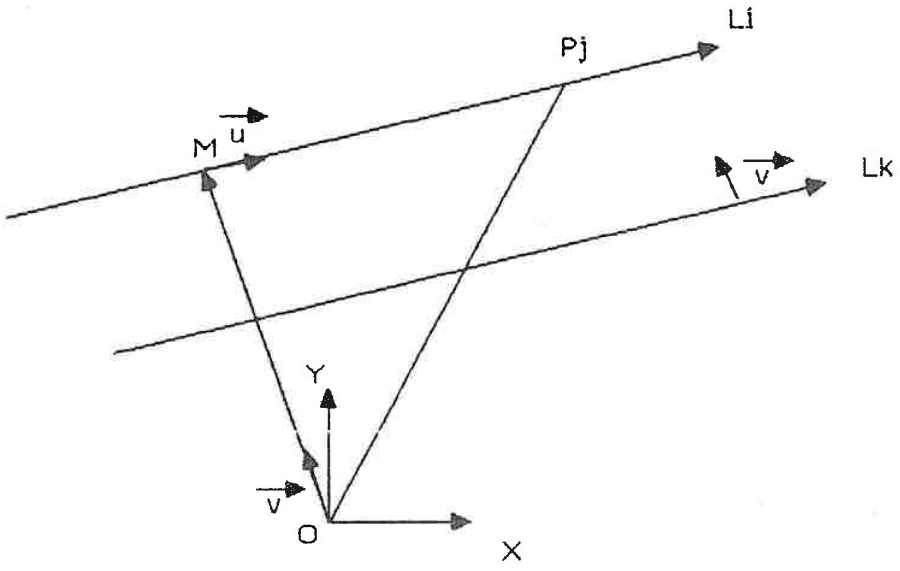
ou

Li = Pj,Lk

L'en-tête du sous-programme serait alors : #LPL/,OO,KO

b. Définition des calculs analytiques nécessaires et écriture des instructions correspondantes.

Analyse :



$\overrightarrow{OP} = \overrightarrow{OM} + \overrightarrow{MP}$

$\overrightarrow{OP} = d\,\overrightarrow{v} + \mu\,\overrightarrow{u}$ $\qquad$ $\overrightarrow{v} \perp \overrightarrow{u}$

$\overrightarrow{OP}\,\overrightarrow{v} = d$ $\qquad$ aussi $\qquad$ $\boxed{d = PxVx + PyVy}$

En appelant Px et Py les coordonnées du point P, et Vx et VY les coordonnées du vecteur $\overrightarrow{v}$.

Ecriture du sous-programme :

```
#LPL/,O0,K0
<ERR 1
V50=K0                          V50= Vx
V53=O0                          V51= Vy
V52=V50*V53+(V51*V54)           V52= d= PxVx+PyVy
#
```

c. Introduction de ce programme dans la zone système grâce à l'éditeur LAMULE. ( Après dévérouillage de cette zone )

d. Mise au point en mode pas à pas . ( Après ouverture du sous-programme )

e. Fermeture de ce sous-programme

f. Vérouillage de la zone système et archivage sur disque de la nouvelle version du système ainsi créé.

Pour fixer les idées, une telle modification nous a pris, en tout, une demi-heure.

## 5.3. STRUCTURE DE POINTS

Au moment de la définition d'une structure de points, il n'est bien sûr pas possible de calculer les coordonnées de tous les points car cela prendrait trop de place en mémoire. La structure est donc repérée par l'adresse de l'instruction qui la définit. C'est uniquement lors d'une instruction d'utilisation de cette structure que le sous-programme correspondant à la définition sera exécuté.

L'appel d'une structure peut comporter des modificateurs permettant l'inversion du sens de parcours et l'omission ou la rétention de points isolés ou regroupés.

L'instruction d'exécution d'une structure possède une syntaxe d'appel commune aux différents types de structures existants. Elle décrit dans un ordre précis, l'ensemble des points où sera réalisé l'usinage désigné préalablement, et se compose des phases successives suivantes :

1. Acquisition des paramètres de la structure précédemment définie.

2. Initialisation des données internes spécifiques au type de la structure tels que le nombre total de points, les coordonnées du point de départ etc...

3. Calcul du numéro de point courant en tenant compte des modifications indiquées dans l'instruction d'exécution.

4. Calcul des coordonnées du point courant.

5. Appel de la procédure d'usinage à effectuer en ce point.

Les phases 1, 2 et 4 sont en fait réalisées par le sous-programme de définition de la structure qui est appelé indirectement par l'intermédiaire de la forme canonique de celle-ci.

Exemple :

```
#PROG
1 → S1 = PP/ P1, P2, 10        ← Définition d'une structure de points

2 → PERC, 1, 10, 30, 450

3 → S1, IN, OM, 6              ← Appel de la structure S1 décrite en
                                 sens inverse et en omettant le point
                                 N° 6
    &
```

Phases d'exécution :

1 La forme canonique de S1 devient l'adresse de la définition de cette structure

2 Appel de la procédure de perçage

3 Initialisation des paramètres de la stucture S1 par exécution de sa définition 1

4 Calcul du numéro de point courant en tenant compte de IN et OM

5 Calcul du point courant

6 Exécution de la procédure d'usinage sur ce point

7 Fin si le point courant est égal au nombre total de points sinon recommencer en 4

Une structure peut être définie de 5 manières différentes. Les tableaux suivants présentent les méthodes de définitions classiques que nous proposons.

TABLEAU N° 20 : DEFINITION DE STRUCTURES DE POINTS

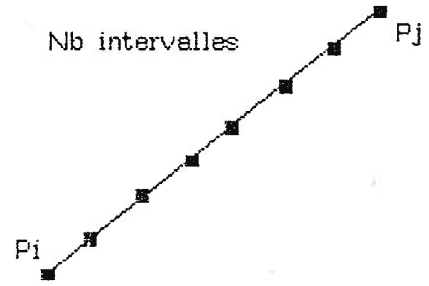
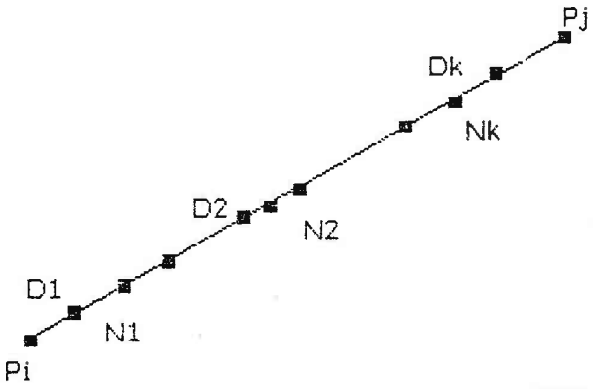
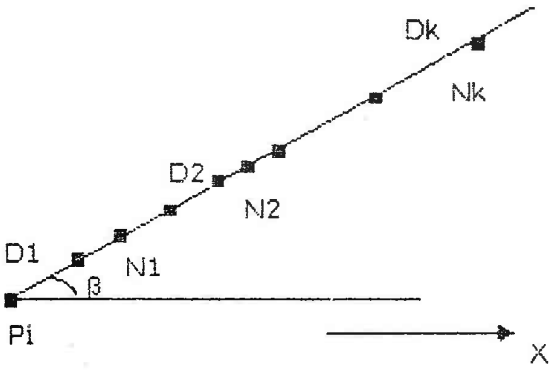
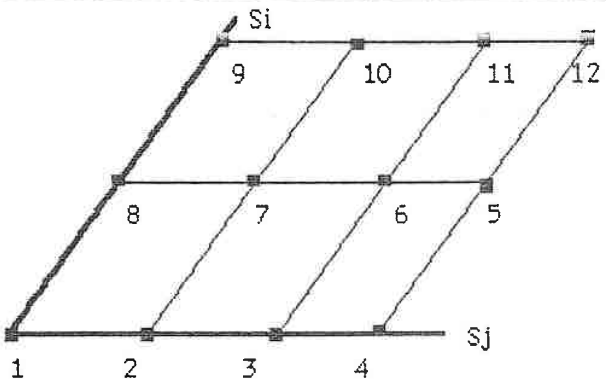
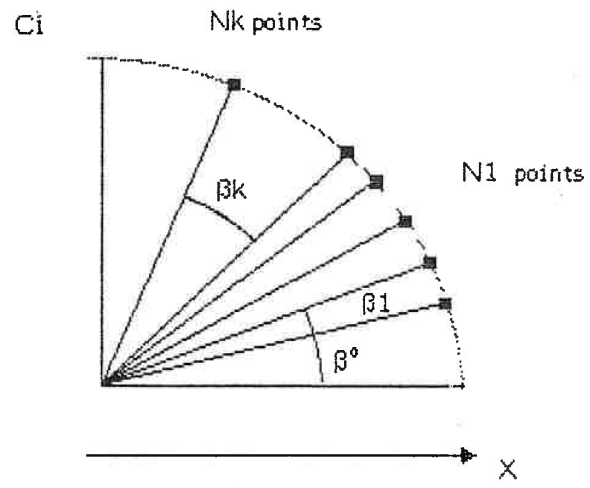
| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| DEFINIE PAR 2 POINTS ET UN NOMBRE D'INTERVALLES | Sn=PP/Pi,Pj,Nb<br>ou<br>Sn=Pi,Pj,Nb |  |
| DEFINIE PAR 2 POINTS, UN ANGLE ET DES COUPLES NOMBRE DE POINTS-DISTANCE ENTRE POINTS | Sn=PP/Pi,Pj,Nk,Dk<br>ou<br>Sn=Pi,Pj,Nk,Dk |  |
| DEFINIE PAR UN POINT, UN ANGLE, DES COUPLES NOMBRE DE POINTS-DISTANCE ENTRE POINTS | Sn=PA/Pi,ß,Nk,D<br>ou<br>Sn=Pi,ß,Nk,Dk |  |
| STRUCTURE PARALLELE | Sn=SP/Si,Sj<br>ou<br>Sn=Si,Sj |  |

TABLEAU N° 21 : DEFINITION DE STRUCTURES DE POINTS SUITE

| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| STRUCTURE CIRCULAIRE | Sn=SC/Ci,β°,Nk,βk<br>ou<br>Sn=Ci,β°,Nk,βk |  |

## 5.4. TRANSFORMATIONS GEOMETRIQUES

Les systèmes d'aide à la programmation classiques proposent des transformations géométriques qui peuvent agir de deux manières différentes :

- soit les transformations modifient directement les éléments géométriques
- soit elles ont effet uniquement lors des ordres de mouvement. [2]

Nous n'avons pas voulu proposer, à priori, les deux modes de transformations mais nous avons choisi la deuxième méthode qui est la plus simple à mettre en oeuvre et permet de résoudre le plus grand nombre de problèmes. Il est certain qu'il faudrait un minimum de modifications des sous-programmes, pour que notre système intègre également le premier mode de transformations géométriques.

Pour la définition et l'appel des transformations nous utilisons le même principe de traitement en temps différé que celui retenu pour l'exécution des structures de points. Ainsi une transformation géométrique est mémorisée par son adresse de définition et son exécution sera effectuée dans les procédures de déplacements élémentaires par appel indirect de l'instruction pointée par sa forme canonique. Les transformations géométriques sont exécutées suivant l'ordre croissant de leur indice, par appel du sous-programme TR/ présenté dans l'exemple suivant.

Exemple :

```
T1=RO/45
T2=SY/
-
-
PERC,1
P1        ->  Appel d'une procédure de déplacement élémentaire
P2            comportant TR/
```

Le sous-programme TR/ réalise :

Pour I de 0 à 9 exécuter l'instruction dont l'adresse est contenue dans la forme canonique de T1
Fin

Nous proposons cinq types de déclaration possibles mais d'autres transformations peuvent être définies si besoin est comme par exemple, changement de plan, homothétie etc ...

TABLEAU N° 22 : DEFINITION DE TRANSFORMATIONS GEOMETRIQUES

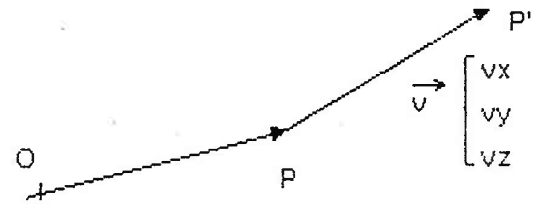
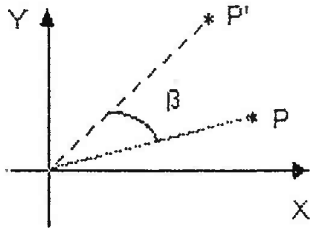
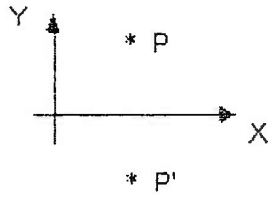
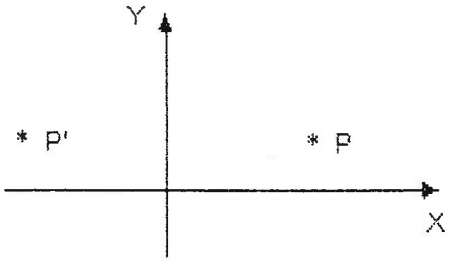
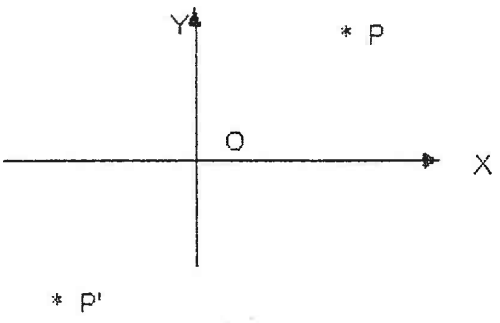
| DEFINITION | FORMAT | REPRESENTATION |
|---|---|---|
| TRANSLATION VECTORIELLE | Tn=TR/vx,vy,vz |   |
| ROTATION D'UN ANGLE β DANS LE PLAN XY | Tn=RO/β |   |
| SYMETRIE PAR RAPPORT A L'AXE OX | Tn=SX/ |   |
| SYMETRIE PAR RAPPORT A L'AXE OY | Tn=SY/ |   |
| SYMETRIE PAR RAPPORT A L'ORIGINE | Tn=SO/ |   |

TABLEAU N° 23 : EXECUTION OU ANNULATION DE TRANSFORMATIONS GEOMETRIQUES

| DEFINITION | FORMAT | ROLE |
|---|---|---|
| ANNULATION D'UNE TRANSFORMATION | Tn=NL/ | |
| ANNULATION DE TOUTES LES TRANSFORMATIONS | AT/ | Ti=NL/ pour i de 1 à 9 |
| EXECUTION DES TRANSFORMATIONS | TR/ | Exécution de toutes les transformations géométriques |

## 6. USINAGE SUR UN ENSEMBLE DE POINTS

Nous avons retenu de réaliser seulement quelques procédures d'usinages complexes points à points qui serviront d'exemple pour la conception de fonction de fabrication, par un utilisateur averti ( Société BOSCH, ou préparateur ayant quelques notions de fonctionnement du système ).

De plus, nous avons mis au point une procédure spécifique permettant de déclarer dans une macro explicite, un ensemble d'usinages à effectuer en chaque point.

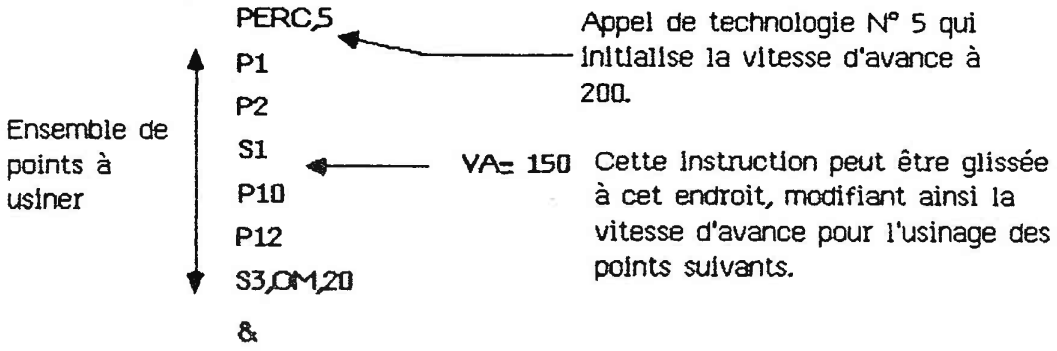
Avant la présentation de ces définitions d'usinage complexe, nous décrivons les méthodes de déclaration d'ensemble de points acceptées par notre système.

### 6.1. Ensemble de points ( Parcours d'usinage point à point )

L'ensemble de points est déclaré derrière le type d'usinage à effectuer, il est terminé par le caractère " Et commercial " &, et peut être constitué par :

- des points
- des structures de points

Exemple :




L'usinage correspondant à la procédure PERC sera alors effectué successivement sur les points définis dans cet ensemble. Ce sont des instructions et non des paramètres, ceci permet de glisser d'autres instructions comme par exemple pour modifier un paramètre technologique. ( Vitesse d'avance etc ...)

## 6.2. Procédures d'usinages cataloquées

Les procédures d'usinage que nous avons mis au point sont :

- PERC pointage perçage
- LAMA lamage
- TARA taraudage

Ces procédures contiennent, dans leur zone de paramètres, les numéros des macros techonologiques utilisées et les différentes prise de cote en Z, en vitesse rapide ou travail.

Ces programmes génèrent automatiquement :

- Les blocs machines correspondant à la suite d'opérations à effectuer en chaque point
- Les changements d'outil et la modification des paramètres technologiques.

Si l'armoire de commande concernée possède des cycles fixes les procédures pourront être très simplement modifiées pour profiter des caractéristiques de cette armoire.

Exemple :

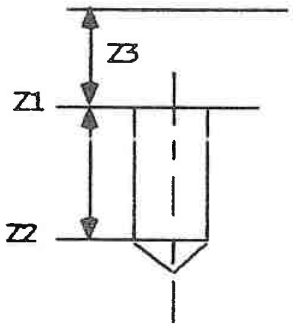
PERC : Sous-programme de perçage en point à point

Appel :

PERC, n, Z1, Z2, z3

Ensemble de points

&




n : N° de la technologie d'outil qui est obligatoirement composée de :
- LG
- VA
- VR

Z1 : Cote de début de trou (absolu)

Z2 : Cote de fond de trou (absolu)

Z3 : Cote de déplacement rapide relatif à Z1

Ensemble de points : Ensemble des points où l'on réalise ce perçage

Les 2 autres procédures possèdent les mêmes paramètres technologiques que le programme PERC.

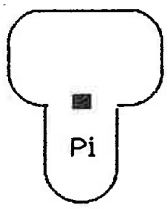
### 6.3. Procédures d'usinages multiples

Afin de permettre la déclaration puis l'utilisation d'usinages multiples sur un ensemble de points, nous avons défini une procédure qui effectue, en chaque point, les usinages préalablement définis, dans une macro explicite spéciale. Cette macro est appelée obligatoirement USIn avec n compris entre 0 et 9, et la procédure d'utilisation EXU/ est mise en oeuvre comme le montre l'exemple suivant.

Exemple :

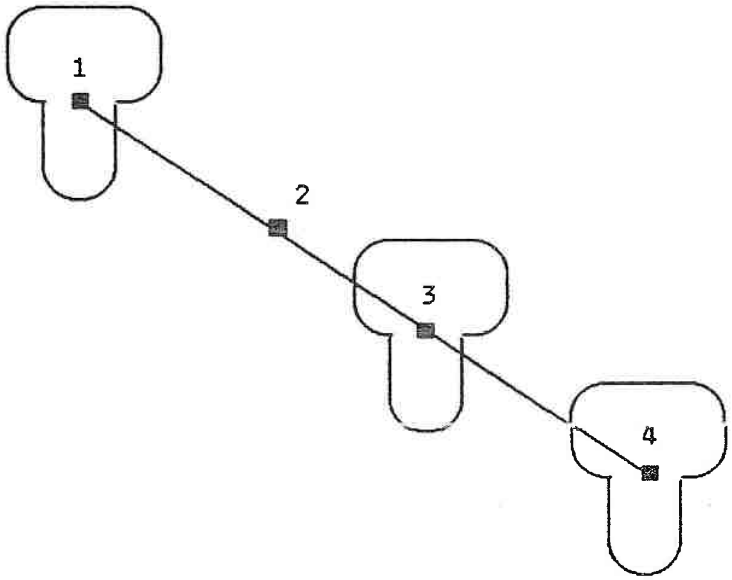
```
%USI1
-
-
-
%
```

⟶ Défini par exemple la géométrie et les ordres mouvements de l'usinage suivant.




```
EXU/ 1, 1
P1
S1, OM, 2
&
```

Réalise



## 7. CONTOURNAGE

Nous avons retenu le principe d'établir le profil de la pièce à usiner, en décrivant simplement la suite logique des éléments géométriques pré-définis qui le constituent. Pour faciliter l'emploi des procédures de contournage et simplifier l'élaboration géométrique de la pièce, nous avons adjoint au système une instruction de création de cercle de raccordement entre les éléments géométriques.

Exemple :

Suite d'éléments définissant le profil de la pièce à usiner

```
CD/
L1
L2
CR/10   ------>  Création d'un cercle de raccordement
C2               de rayon de 10 entre les éléments
L3               géométriques qui suivent
L0
&
```

VA = 200

La suite des éléments géométriques ne constitue pas une zone paramètre des procédures de contournage , mais est formée en fait d'instructions spéciales qui engendrent les mouvements. Ceci permet, d'une part, de définir un profil composé d'un nombre quelconque d'éléments, et d'autre part, de glisser d'autres instructions comme par exemple une modification de la vitesse d'avance.

Remarquons que si le profil doit être utilisé plusieurs fois, il est toujours possible de le décrire dans une macro explicite qui pourra être appelée aussi souvent que nécessaire.

Exemple :

```
%PROF
L1
L2                    Définition du profil
CR/10
C2
L3
L0
%
CD/
PROF
&
DIA = DIA - 1  <----  Modification du diamètre de
CD/                   l'outil courant
PROF
&
```

Nous trouverons dans les tableaux suivants, les ordres de contournage standards qui permettent de positionner l'outil par rapport au profil tout au long de celui-ci en se référant au sens de parcours, qui est implicitement contenu dans la définition de chaque élément géométrique. Ceci constitue un des avantages de la géométrie orientée.

Il peut subsister malgré tout dans certains cas ( passage d'un cercle vers une droite ou un cercle, passage d'une droite vers un cercle ), une ambiguité si les éléments ne sont pas tangents. Dans ce cas, l'ambiguité est levée par un modificateur précisant, au point d'intersection concerné, l'orientation de l'élément suivant par rapport à l'élément actuel.

Exemple :

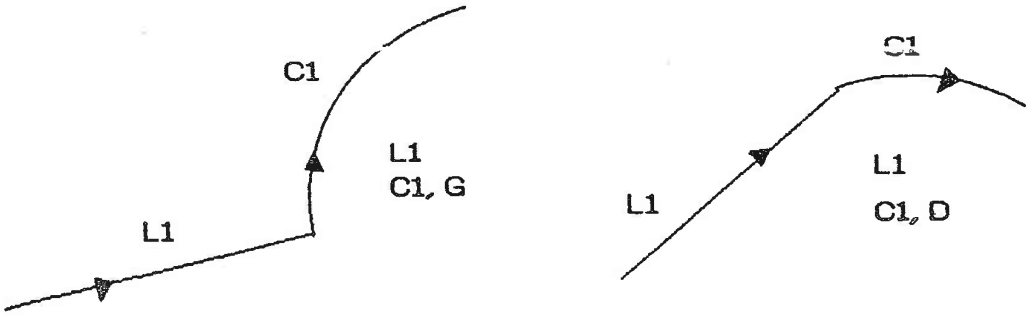


TABLEAU N° 24 : ORDRES DE CONTOURNAGE

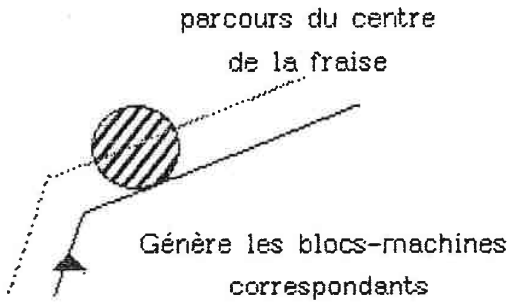
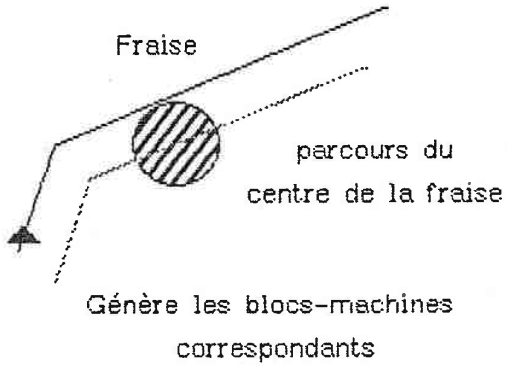
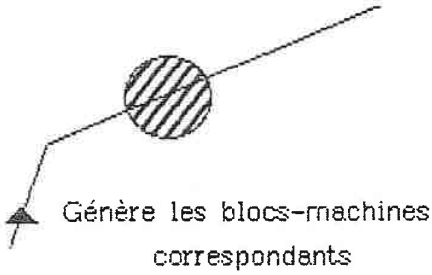
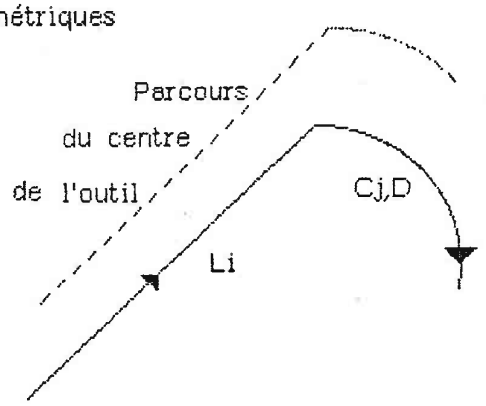
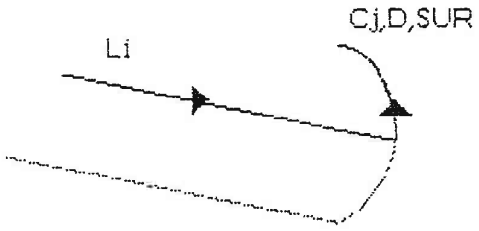
| DEFINITION | FORMAT | ROLE |
|---|---|---|
| CONTOURNAGE A GAUCHE DU PROFIL | CG/ *Suite d'instructions* & |   |
| CONTOURNAGE A DROITE DU PROFIL | GD/ *Suite d'instructions* & |   |
| CONTOURNAGE SUR LE PROFIL | CS/ *Suite d'instructions* & |   |

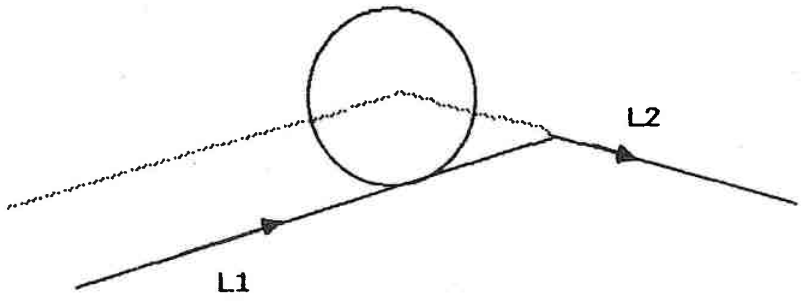
TABLEAU N° 25 : INSTRUCTIONS DE CONTOURNAGE

<table>
<tr><th>DEFINITION</th><th>FORMAT</th><th>REPRESENTATION</th></tr>
<tr><td>ELEMENT GEOMETIQUE<br>ETANT UNE DROITE</td><td>Li [ ,m1] [ ,m2 ]<br><br>m1 D<br>ou = G<br>m2 SUR</td><td rowspan="2">m1 et m2 définissent la position de l'outil par rapport au profil à générer et lèvent toutes les ambiguités géométriques<br>  <br><br>  <br></td></tr>
<tr><td>ELEMENT GEOMETIQUE<br>ETANT UN CERCLE</td><td>Ci [ ,m1] [ ,m2 ]<br><br>m1 D<br>ou = G<br>m2 SUR</td></tr>
<tr><td>CREATION D'UN CERCLE DE RACCORDEMENT ENTRE DEUX ELEMENTS GEOMETRIQUES</td><td>CR/r</td><td>NB : L'annulation du cercle de raccordement se fait de la manière suivante<br><br>CR/0</td></tr>
</table>

Cette manière de faire n'impose pas d'imaginer la position d'intersection non concernée, pour la génération du profil.

Enfin, pour laisser libre l'utilisateur de débuter ou terminer une trajectoire avec l'outil centré sur un élément géométrique, nous avons créé le modificateur 'SUR'.

Exemple :




```
CG/
-
-
L1
L2,SUR
-
&
```

## 8. AUTRES INSTRUCTIONS

### 8.1. Mouvements élémentaires

Ce sont donc les instructions qui permettent d'effectuer des mouvements élémentaires sans passer par les procédures point à point ou de contournage. Ces instructions sont généralement utilisées pour des dégagements d'outils ou des prises de passes. Elles permettent un déplacement en vitesse de travail ou rapide sur les coordonnées d'un point, des changements de cote en Z etc... Nous les avons regroupées en deux catégories; déplacement rapide ( Mot clé GR/ ) et déplacement travail ( Mot clé GT/ ). Nous les présentons dans le tableau suivant.

### 8.2. Instruction d'initialisation : DEBU

Cette instruction a pour but d'initialiser les variables associées à l'exécution des programmes, et les données technologiques d'usinage, comme :

- Le numéro d'outil de référence
- La distance surface pièce en Z
- Le dégagement rapide en Z après chaque usinage point à point

Aussi l'appel de ce sous-programme doit obligatoirement figurer avant les instructions de définitions géométriques et les ordres d'usinages.

### 8.3. Prise en compte d'une macro technologique

Dans les procédures d'usinage point à point, un paramètre permet de préciser le numéro de la technologie d'outil utilisée et le changement d'outil sera alors automatiquement généré par ces procédures.

Par contre, les mouvements de contournage se succédant généralement sans changement d'outil, nous n'avons pas voulu procéder de la même façon, et le programmeur doit alors préciser l'outil qu'il désire utiliser pour les contournages à venir. Pour ce faire, nous avons créé une instruction de prise en compte de technologie d'outil, qui possède comme paramètre l'indice **i** de la macro-instruction **TOi** associée à l'outil.

Exemple :

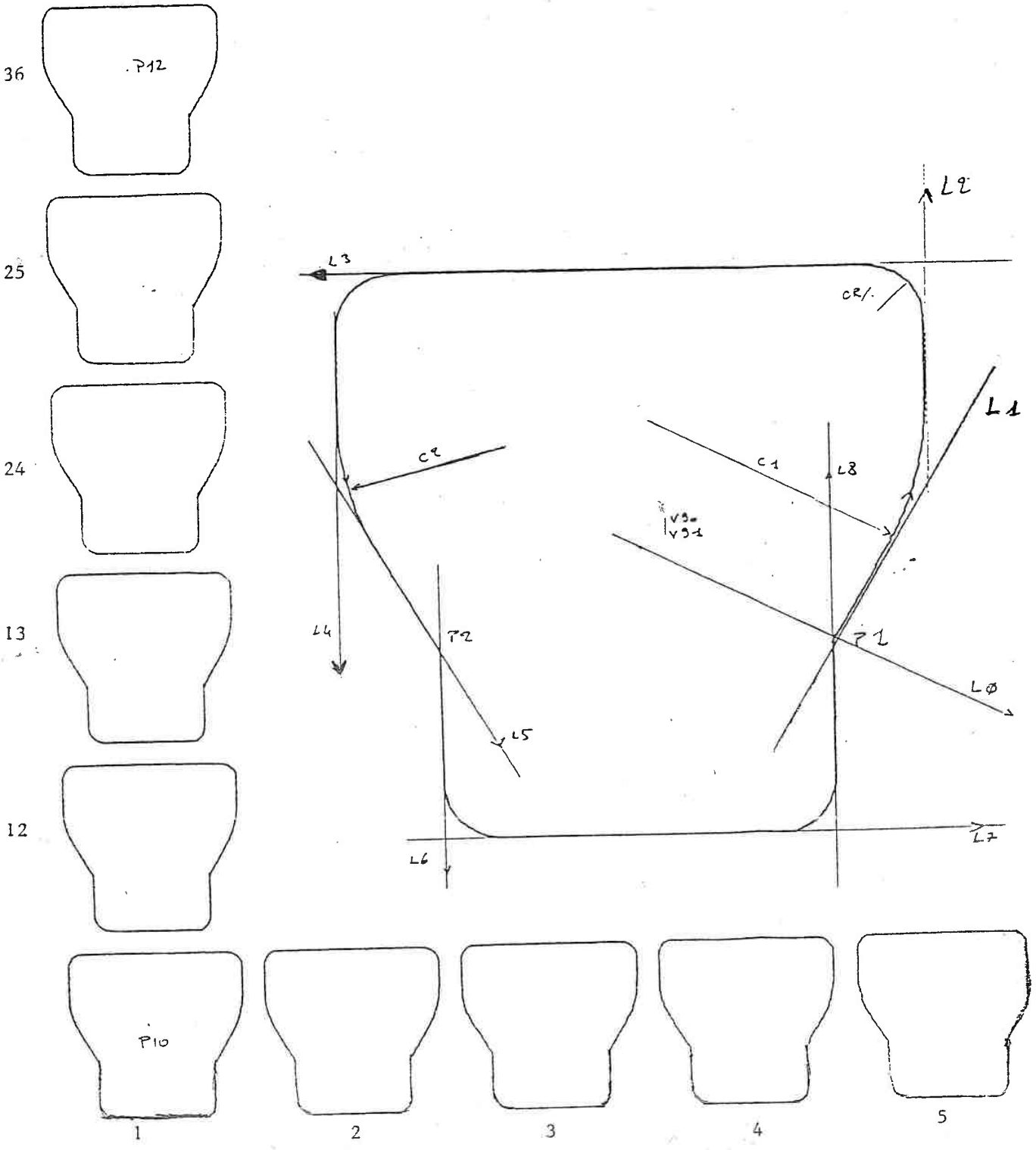
```
%TO1
VA = 100
LG = 45.2
-
-
%
-
%TOi
-
-
%
-
-
CO/ 1  <------  Réalise l'appel de la technologie d'outil N° 1
CG/
-
-
&
CD/
-
-
&
```

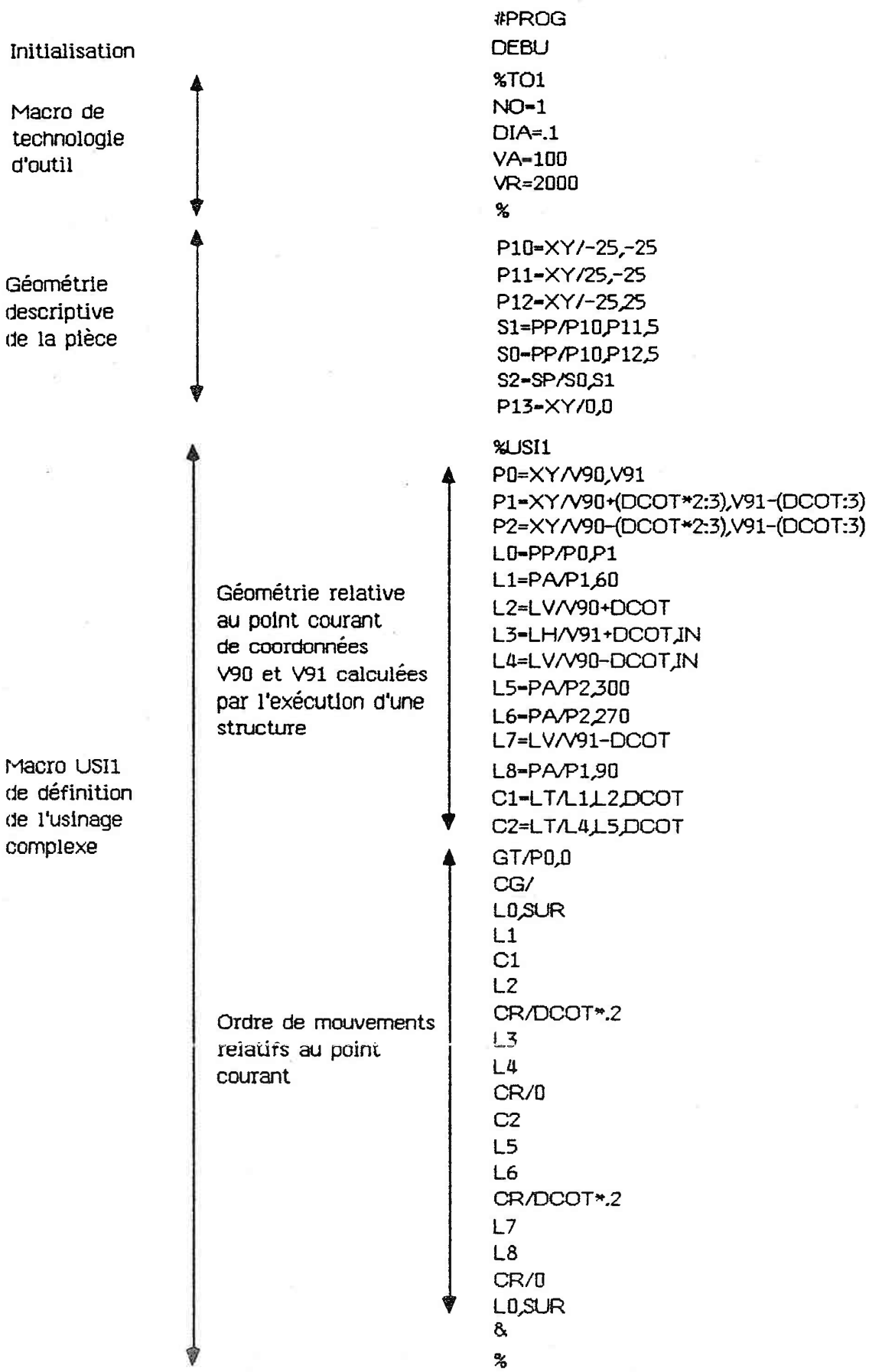
TABLEAU N° 26 : AUTRES INSTRUCTIONS

| DEFINITION | FORMAT | ROLE |
|---|---|---|
| INITIALISATION DES DONNEES D'USINAGE | DEBU (N1,Z1,Z2) | Détermination :<br>N1 N° d'outil de référence<br>Z1 distance surface-pièce en Z<br>Z2 dégagement rapide en Z après des usinages points à points |
| PRISE EN COMPTE D'UNE CORRECTION D'OUTIL | CO/n | n N° de la technologie d'outil associée à la correction. |
| PRISE DE COTE EN Z | GZ/côte en Z | Génération du bloc-machine correspondant |
| DEPLACEMENT RAPIDE VERS UN POINT | GR/x,y,côte en Z<br>ou<br>GR/Pi,côte en z | Génération du bloc-machine correspondant |
| DEPLACEMENT EN VITESSE DE TRAVAIL VERS UN POINT | GT/x,y,Côte en Z<br>ou<br>GT/Pi,côte en Z | Génération du bloc-machine correspondant |

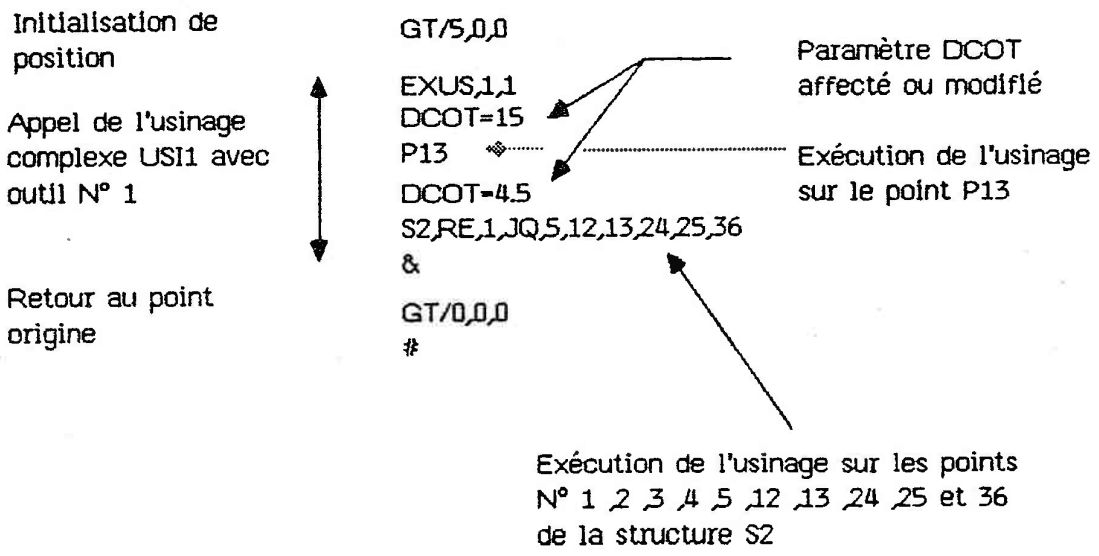
## 9. EXEMPLE DE PROGRAMMATION D'UN USINAGE

Nous présentons dans cet exemple la programmation d'un usinage sur un ensemble de points, à l'aide de la définition d'une macro-instruction d'usinage complexe USI. La variable DCOT est utilisée pour permettre le paramétrage des éléments géométriques supports de l'usinage.




#PROG
Initialisation
DEBU
%TO1
Macro de technologie d'outil
NO=1
DIA=.1
VA=100
VR=2000
%
P10=XY/-25,-25
P11=XY/25,-25
P12=XY/-25,25
S1=PP/P10,P11,5
Géométrie descriptive de la pièce
S0=PP/P10,P12,5
S2=SP/S0,S1
P13=XY/0,0
%USI1
P0=XY/V90,V91
P1=XY/V90+(DCOT*2:3),V91-(DCOT:3)
P2=XY/V90-(DCOT*2:3),V91-(DCOT:3)
L0=PP/P0,P1
L1=PA/P1,60
Géométrie relative au point courant de coordonnées V90 et V91 calculées par l'exécution d'une structure
L2=LV/V90+DCOT
L3=LH/V91+DCOT,IN
L4=LV/V90-DCOT,IN
L5=PA/P2,300
L6=PA/P2,270
L7=LV/V91-DCOT
Macro USI1 de définition de l'usinage complexe
L8=PA/P1,90
C1=LT/L1,L2,DCOT
C2=LT/L4,L5,DCOT
GT/P0,0
CG/
L0,SUR
L1
C1
L2
CR/DCOT*.2
Ordre de mouvements relatifs au point courant
L3
L4
CR/0
C2
L5
L6
CR/DCOT*.2
L7
L8
CR/0
L0,SUR
&
%

Programme principal suite



# CHAPITRE 4

# EVOLUTIONS DU SYSTEME LAMULE

Tel qu'il existe actuellement, le système LAMULE est exploitable industriellement, mais pour augmenter ses performances nous pouvons envisager un certain nombre d'évolutions, sans pour autant modifier la philosophie générale du macro-interprète.

Dans un premier temps, il serait nécessaire de développer des procédures d'usinage élaborées telles que génération de poches, ébauche automatique de profil, etc... Il sagit là de l'écriture de nouveaux sous-programmes systèmes, mais cela ne touche en aucun cas le noyau assembleur.

La seconde étape d'évolution devrait concerner la visualisation graphique des formes canoniques, des mouvements de l'outil et des profils usinés lors de l'exécution / simulation du programme.

Ensuite, il serait très performant, pour la mise au point des programmes, de permettre une interprétation immédiate des instructions dès leur édition, avec visualisation graphique des éléments définis.

Enfin, la dernière étape consisterait à apporter des facilités de programmation au préparateur, par l'utilisation d'un type de dialogue homme-machine plus agréable à manipuler qu'un langage, comme par exemple, la sélection de fonctions dans des menus hiérarchisés présentés à l'écran.

## 1. CREATION DE PROCEDURES ELABOREES

Les fonctions d'usinage les plus utiles sont des procédures d'ébauche d'un profil préalablement défini. Dans ce cadre l'évidement de poche constitue un problème complexe dont nous proposons ci-dessous une méthode de résolution. Notre étude se limite à la génération de poches convexes. En effet les calculs de passes que nécessiterait la génération de poches concaves, sont d'une autre nature.

La syntaxe de l'instruction d'appel de la procédure serait du même type que celles utilisées pour les sous-programmes de contournage. C'est à dire que le profil de la poche serait déterminé par la suite des éléments géométriques qui le définissent.

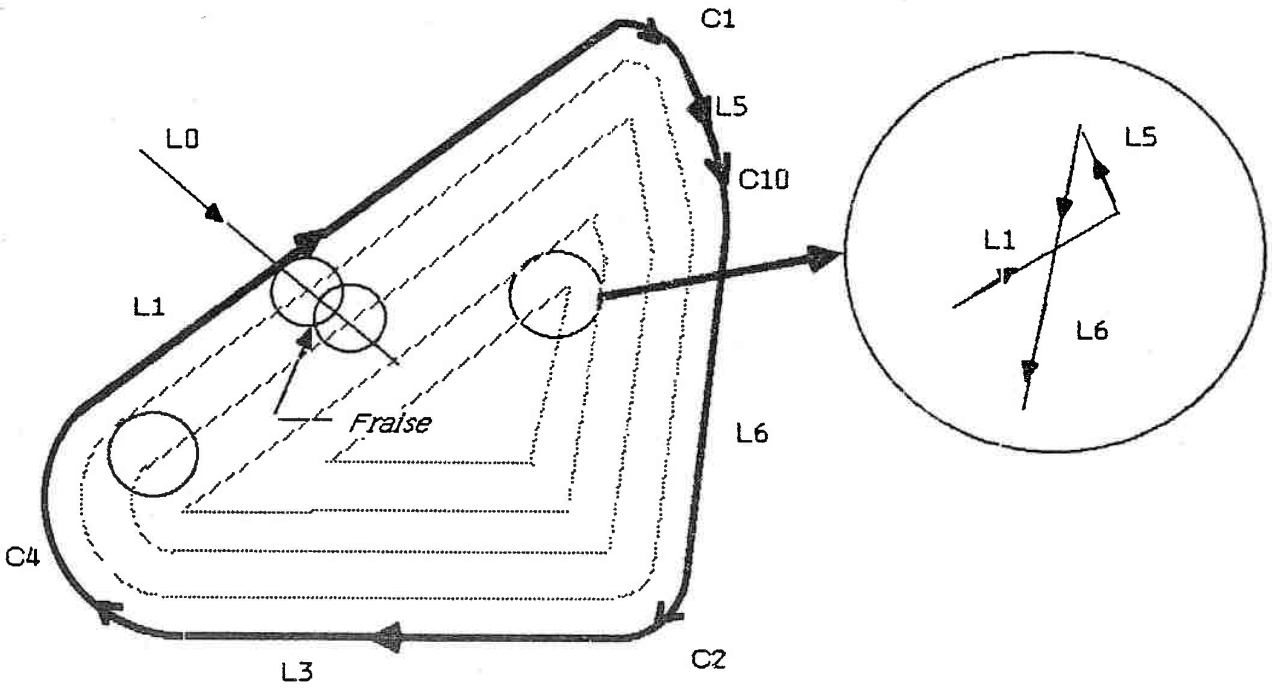
La méthode d'usinage consisterait à suivre le profil à une distance croissante en prenant successivement les passes suivant une droite déterminée. On obtiendrait ainsi une programmation telle que celle présentée dans l'exemple qui suit.

Exemple :

```
POCH, 1, 0.5, L0
L1
C1
L5
C10
L6
C2
L3
C4
L1
&
```

Déterminent :

- Le numéro d'outil
- Le recouvrement
- La droite de prise de passe



Le problème de l'écriture d'une telle procédure est surtout lié à la suppression des éléments géométriques qui ne doivent plus être pris en compte dans le profil à partir d'une certaine distance ( Dans l'exemple les cercles C1 et C2 ne sont plus concernés dès la deuxième passe ). La géométrie orientée apporte une solution simple puisqu'il suffira d'éliminer les éléments géométriques qui devraient être parcourus dans le sens inverse du sens de définition. ( *Cf figure précédente* )

## 2. VISUALISATION

L'introduction de la visualisation graphique ne devrait apporter aucun problème, compte tenu de la structure même de notre système. En effet les étapes de réalisation de cette évolution seraient :

- Ecriture de deux primitives assembleur de génération d'un segment de droit et d'un arc de cercle.
- Ecriture de deux sous-programmes systèmes d'appel de ces deux primitives
- Utilisation de ces sous-programmes d'une part dans les procédures de mouvement, d'autre part dans les sous-programmes de définitions géométriques.

Il resterait toutefois à définir des variables symboliques, précisant la fenêtre de visualisation, ainsi que le facteur d'échelle.

## 3. EXECUTION SIMULTANEE

La démarche actuelle du péparateur pour l'écriture et la mise au point d'un programme comporte trois phases :

- L'écriture interactive du programme grâce à l'éditeur de texte
- La simulation du programme en utilisant au maximum les possibilités du metteur au point.
- L'exécution du programme correct pour générer les blocs machines.

Cette manière de procéder apporte déjà une grande facilité de mise au point, surtout si l'on lui associe les visualisations graphiques présentées au paragraphe précédent et si l'on utilise le mode pas à pas en simulation.

Toutefois on pourrait encore améliorer l'aide au préparateur, en offrant un mode d'exécution simultanée à l'écriture de chaque instruction. Ceci permettrait un contrôle immédiat des éléments géométriques définis, ainsi que des fonctions d'usinage appelées.

Malheureusement ce nouveau mode apporte des restrictions dans l'emploi de certaines instructions et ne pourra donc pas toujours être utilisé. Ainsi les ruptures de séquence aval ne peuvent être exécutées et les définitions des sous-programmes et macro-instructions nécessiteront un abandon temporaire du mode exécution simultanée.

En fait ce mode n'aurait donc d'intérêt que pour le préparateur final écrivant uniquement un programme pièce, en utilisant des procédures déjà définies.

Il faut remarquer que l'introduction de cette possibilité nécessitera une reprise partielle du noyau assembleur au niveau de l'enchainement des différentes tâches.

## 4. DIALOGUE HOMME - MACHINE

La dernière étape de l'évolution du système serait l'élaboration d'un nouveau type de dialogue plus proche de la C.F.A.O., mais en respectant la philosophie de notre système. Nous pensons qu'une très bonne solution consisterait à garder le système tel qu'il est, complémenté par les améliorations préalablement décrites, en lui ajoutant une interface logicielle homme-machine supplémentaire, tout en gardant les possibilités d'édition actuelle.

Cette nouvelle interface permettrait de remplacer l'écriture d'une instruction, par les phases successives suivantes :

- Sélection d'une fonction dans un menu
- Affichage automatique des paramètres utiles
- Attribution à chaque paramètre d'une valeur ou d'une expression arithmétique.

En fait, à l'issu de ce traitement, une instruction tout à fait classique serait générée en mémoire, et ainsi, on retrouverait la représentation actuelle des programmes dont l'interprétation ne serait donc pas modifiée.

# CONCLUSION

Le système que nous venons de présenter est à considérer sous deux aspects: le macro-interprète LAMULE, et son utilisation pour la programmation de pièces à usiner sur fraiseuse.

Cette application est à l'heure actuelle tout à fait opérationnelle et répondra à la majorité des problèmes de fraisage, grâce à la possibilité de création de procédures d'usinages évoluées, adaptées aux besoins du préparateur. Au cours de cette phase d'adaptation au fraisage, nous avons pu mettre en évidence les qualités du système LAMULE et surtout la souplesse de création des instructions et des procédures, la rapidité de leur mise au point permettant ainsi, à tout instant, d'augmenter les performances du système sans aucune compilation, ni génération.

L'implantation actuelle du système ne constitue pas une version définitive. Deux orientations peuvent alors être proposées. Tout d'abord intégrer notre système, implanté sur une carte spécialisée, dans l'armoire de commande de la machine outil. La deuxième solution consiste à déporter le système sur un micro-ordinateur qui pourra desservir plusieurs machines dans un mode de type D.N.C. ( Direct Numerical Control ). Cette perspective nous semble très bien adaptée aux petites et moyennes entreprises qui ne peuvent pas s'équiper de gros systèmes de C.F.A.O. ( Conception de Fabrication Assitée par Ordinateur ) et qui veulent, malgé tout, résoudre leurs problèmes de préparation / programmation des pièces.

# BIBLIOGRAPHIE

[1] MUSSE JEAN-PIERRE

Conception d'une commande numérique directe de machine-outil. Application au tour à un ou deux chariots.

Thèse de docteur ès Sciences-Physiques, NANCY, 1977

[2] SHAH RAYMOND

Guide de la commande numérique 79.

ZURICH 1979

[3] SOUBIES-CAMY HENRI

Les bases de la programmation en commande numérique.

Revue MECANIQUE, PARIS, 1968

[4] BOSCH

Manuel de programmation de la commande numérique BOSCH ALPHA 3.

[5] ADEPA

Manuel d'utilisation du système PROMO.

[6] LETAC GUY

Définition des courbes et surfaces en langage APT et ADAPT.

Revue MECANIQUE, PARIS, 1968

[7] HERMANN JEAN

Etude d'une arithmètique flottante et des calculs géométriques pour un langage évolué de programmation de piéces.

D.E.A. Génie électrique, NANCY, 1981

[8] DUPAS HENRI

Réalisation d'un langage d'aide à la programmation pour machines à commande numérique.

D.E.A. Génie éléctrique, NANCY, 1982

[9] RENAULD PHILIPPE

Réalisation d'un langage d'aide à la programmation pour machines à commande numérique.

D.E.A. Génie éléctrique, NANCY, 1982

NOM DE L'ETUDIANT : Monsieur RENAULD Philippe

NATURE DE LA THESE : Doctorat 3ème Cycle en GENIE ELECTRIQUE

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 21 JUIN 1984 n° 1159

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE DE NANCY I
Le Président

R. MAINARD

Thèse de troisième cycle

Auteur : Philippe RENAULD

Etablissement : UNIVERSITE DE NANCY I

Titre : Etude et réalisation d'un système d'aide à la programmation évoluée pour la commande directe d'une machine outil.

Mots-clés : machine outil, commande numérique, informatique, programmation assistée, langage

Résumé : Nous présentons un travail relatif à un système d'aide à la programmation évoluée, intégré à l'armoire de commande d'une fraiseuse, souple et modulaire, développé autour d'un langage de type macro-interprète associé à un metteur au point.

L'interprète permet la création de toutes les fonctions géométriques et d'usinage formant l'adaptation au fraisage, par la simple écriture de sous-programmes. Les instructions du langage d'aide à la programmation ont alors la caractéristique d'être constituées d'appels de ces fonctions.

Le metteur au point comprend des procédures de visualisation ou de modification d'états intermédiaires, qui facilitent le suivi de l'exécution des programmes.