FORmac DEsk CALculator : un outil de mise au point et d'aide au calcul formel sur ordinateur

André Laplace

N° d'ordre

THESE

présentée à

L'Université Scientifique et Médicale de Grenoble

pour obtenir

Le grade de Docteur d'état

par

ANDRE LAPLACE

FORMAC DESK CALCULATOR

UN OUTIL DE MISE AU POINT

ET

D'AIDE AU CALCUL FORMEL

SUR ORDINATEUR

Soutenue le 21 FEVRIER 1973 devant la commission d'examen

M.M. KUNTZMAN Président

BOLLIET Rapporteur

CEA

GASTINEL } Examinateurs

SIRET

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

Président : Monsieur Michel SOUTIF
Vice-Président : Monsieur Gabriel CAU

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du Solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHENE Marcel	Chimie papetière
	COEUR André	Pharmacie chimique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures

MM.	GASTINEL Noël	Analyse numérique
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physiologie végétale
MM.	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques Appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MASSEPORT Jean	Géographie
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du Solide
	OZENDA Paul	Botanique
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servomécanismes
	PILLET Emile	Physique industrielle
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	SANTON Lucien	Mécanique
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
Mme	VEYRET Germaine	Géographie
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	BULLEMER Bernhard	Physique
	RADHAKRISHNA Pidatala	Thermodynamique

PROFESSEURS SANS CHAIRE

MM. AUBERT Guy	Physique
BEAUDOING André	Pédiatrie
BERTRANDIAS Jean-Paul	Mathématiques Appliquées
BIARES Jean-Pierre	Mécanique
BONNETAIN Lucien	Chimie minérale
Mme BONNIER Jane	Chimie générale
MM. CARLIER Georges	Biologie végétale
COHEN Joseph	Electrotechnique
COUMES André	Radioélectricité
DEPASSEL Roger	Mécanique des Fluides
DEPORTES Charles	Chimie minérale
DESRE Pierre	Métallurgie
DOLIQUE Jean-Michel	Physique des Plasmas
GAUTHIER Yves	Sciences biologiques
GEINDRE Michel	Electroradiologie
GIDON Paul	Géologie et Minéralogie
GLENAT René	Chimie organique
HACQUES Gérard	Calcul numérique
JANIN Bernard	Géographie
Mme KAHANE Josette	Physique
MM. MULLER Jean-Michel	Thérapeutique
PERRIAUX Jean-Jacques	Géologie et minéralogie
POULOUJADOFF Michel	Electrotechnique
REBECQ Jacques	Biologie (CUS)
REVOL Michel	Urologie
REYMOND Jean-Charles	Chirurgie générale
ROBERT André	Chimie papetière
SARRAZIN Roger	Anatomie et chirurgie
SARROT-REYNAULD Jean	Géologie
SIBILLE Robert	Construction Mécanique
SIROT Louis	Chirurgie générale
Mme SOUTIF Jeanne	Physique générale
M. VALENTIN Jacques	Physique nucléaire

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle AGNIUS-DELORD Claudine	Physique pharmaceutique
ALARY Josette	Chimie analytique
MM. AMBLARD Pierre	Dermatologie
AMBROISE-THOMAS Pierre	Parasitologie
ARMAND Yves	Chimie
BEGUIN Claude	Chimie organique
BELORIZKY Elie	Physique
BENZAKEN Claude	Mathématiques Appliquées
Mme BERTRANDIAS Françoise	Mathématiques Pures
MM. BLIMAN Samuel	Electronique (EIE)
BLOCH Daniel	Electrotechnique
Mme BOUCHE Liane	Mathématiques (CUS)
MM. BOUCHET Yves	Anatomie
BOUSSARD Jean-Claude	Mathématiques Appliquées
BOUVARD Maurice	Mécanique des Fluides
BRIERE Georges	Physique expérimentale
BRODEAU François	Mathématiques (IUT B)
BRUGEL Lucien	Energétique
BUISSON Roger	Physique
BUTEL Jean	Orthopédie
CHAMBAZ Edmond	Biochimie médicale
CHAMPETIER Jean	Anatomie et organogénèse

MM.	CHIAVERINA Jean	Biologie appliquée (EFP)
	CHIBON Pierre	Biologie animale
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique
	CROUZET Guy	Radiologie
	DURAND Francis	Métallurgie
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GAVEND Michel	Pharmacologie
	GENSAC Pierre	Botanique
	GERMAIN Jean-Pierre	Mécanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques Appliquées
	GROULADE Joseph	Biochimie médicale
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Médecine préventive
	IDELMAN Simon	Physiologie animale
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	JOLY Jean-René	Mathématiques Pures
	JOUBERT Jean-Claude	Physique du Solide
	JULLIEN Pierre	Mathématiques Pures
	KAHANE André	Physique générale
	KUHN Gérard	Physique
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LAJZEROWICZ Joseph	Physique
	LANCIA Roland	Physique atomique
	LE JUNTER Noël	Electronique
	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique Nucléaire
	LONGEQUEUE Jean-Pierre	Physique Nucléaire
	LUU DUC Cuong	Chimie Organique
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et Médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)
	MAYNARD Roger	Physique du Solide
	MICOUD Max	Maladies infectieuses
	MOREAU René	Hydraulique (INP)
	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PERRIN Louis	Pathologie expérimentale
	PFISTER Jean-Claude	Physique du Solide
	PHELIP Xavier	Rhumatologie
Mlle	PIERY Yvette	Biologie animale
MM.	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RICHARD Lucien	Botanique
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	ROUGEMONT (DE) Jacques	Neuro-Chirurgie
	STIEGLITZ Paul	Anesthésiologie

MM. STOEBNER Pierre	Anatomie pathologique
VAN CUTSEM Bernard	Mathématiques Appliquées
VEILLON Gérard	Mathématiques Appliquées (INP)
VIALON Pierre	Géologie
VOOG Robert	Médecine interne
VROUSSOS Constantin	Radiologie
ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

MM. BOUDOURIS Georges	Radioélectricité
CHEEKE John	Thermodynamique
GOLDSCHMIDT Hubert	Mathématiques
YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MATIRES DE CONFERENCES

Mme BERIEL Hélène	Physiologie
Mme RENAUDET Jacqueline	Microbiologie

Fait le 8 MARS 1972.

Je remercie Monsieur le Professeur KUNTZMANN de m'avoir fait l'honneur de présider le Jury de cette thèse.

Je tiens à exprimer ma reconnaissance à Monsieur le Professeur GASTINEL, qui toujours intéressé par les possibilités offertes à l'ingénieur par un ordinateur, a été pour moi une source d'encouragement.

Cette reconnaissance va aussi à Monsieur le Professeur BOLLIET qui a su, au cours de nombreuses et fructueuses discussions, m'inciter à prendre des initiatives et à prévoir d'autres débouchés.

Monsieur CEA, professeur à la Faculté des Sciences de Nice, m'a fait l'honneur de juger ce travail, qu'il trouve ici l'expression de mes profonds remerciements.

Monsieur SIRET, qui a guidé mes premiers pas en informatique, a toujours été pour moi de très bon conseil. Je ne saurais oublier de dire qu'il a participé à la "naissance" de Fordecal et qu'il garde un intérêt très vif pour Formac et ses à-côtés. Je tiens à lui exprimer ici ma profonde reconnaissance.

Il m'est agréable de remercier publiquement Monsieur DANG Ng. qui m'a aidé avec passion, opiniatreté et esprit d'initiative.

Je tiens aussi à exprimer ma reconnaissance à Messieurs DUBY (IBM France), PELLETIER et HANS (IBM Centre Scientifique Grenoble) pour l'intérêt qu'ils ont pris à ce travail et l'aide qu'ils m'ont apporté du point de vue documentation.

Enfin, juste retour des choses, je ne saurais oublier ma femme dont la patience a souvent été mise à rude épreuve, par un mari dont les rendez-vous avec un 2741 étaient très fréquents.

Le service de tirage a assuré avec le soin coutumier la publication de cette thèse, qu'il en soit ici remercié.

TABLE DES MATIERES

PREMIERE PARTIE

- - - - ☆ - - - -

DEUXIEME PARTIE

MANUEL UTILISATEUR

- - - - ☆ - - - -

TROISIEME PARTIE

LE SYSTEME

- - - - ☆ - - - -

"Les conceptions dont nous nous occupons et dont nous étudions la connection intime sont elles-mêmes le produit d'un travail prolongé de la pensée mathématique, et sont très éloignées des pensées qui sont d'usage courant dans la vie."

Félix Klein.

PREMIERE PARTIE

Au Père Lisp,

Formac ? Mais c'est simple !

AVANT - PROPOS

Le travail que nous présentons ici a fait l'objet d'un contrat D.R.M.E. n° 70/587 "Calcul Formel" dont le but était la réalisation d'un système interactif de manipulations formelles d'expressions mathématiques fondé sur le langage PL/1 ou FORTRAN IV.

Du fait de l'existence d'un langage de manipulations formelles, fondé sur PL/1, mais fonctionnant sous système batch, nous nous sommes attachés à réaliser un système interactif ayant le même support et par là essayer de développer ce langage.

Le système FORDECAL, FORmac DEsk CALculator, résultat de cette étude, est une version conversationnelle de PL/1-FORMAC qui permet à l'utilisateur de vérifier, voire de développer, rapidement des calculs formels.

Pour des raisons pratiques, nous présentons cette thèse en trois parties. La première contient l'introduction et les conclusions, la deuxième sert de support au manuel utilisateur de FORDECAL et la dernière au système lui-même.

Signalons qu'à la suite des premiers résultats obtenus lors de nos travaux sur FORMAC, l'Institut de Recherches en Mathématiques Appliquées de GRENOBLE a été chargé par le groupe S.E.A.S. - S.M.C. (Symbolic Mathematical Computation Project) de la maintenance de FORMAC et que le système lui-même a fait l'objet de démonstrations [1] [17]

- - - - - - - - -

1. INTRODUCTION

Les divers systèmes interactifs existant à ce jour utilisent, en général, les possibilités offertes par un langage hôte et par le système d'exploitation sous lequel ils sont amenés à fonctionner :

- REDUCE, fondé sur LISP, offre beaucoup d'intérêt dans le domaine des manipulations formelles grâce en particulier à son langage support LISP. Originellement écrit pour résoudre un problème particulier de physique, il a depuis été trés étendu dans le domaine du calcul symbolique. Un désavantage qui limite son emploi, les 450 K de mémoire que nécessite le système seul, d'autre part ce système n'est pas trés bien adapté aux ordinateurs I.B.M. ayant été réalisé sur DEC-PDP 10 [5] ;

- MATHLAB, fondé sur LISP, a pour principales qualités :
un efficient algorithme de factorisation et l'intégration symbolique [2]

- ALADIN, fondé sur LISP, qui offre l'avantage d'utiliser un terminal graphique du type IBM 2250 comme support d'Entrée/Sortie avec la possibilité de sélectionner des sous-expressions affichées grâce au crayon optique [16] ;

- MACSYMA, fondé sur LISP, offre aussi de nombreuses possibilités dans le domaine de l'intégration formelle, mais n'est, malheureusement, pas disponible [14] ;

- SCRATCHPAD, fondé sur LISP, offre les mêmes possibilités que MATHLAB et REDUCE, mais nécessite 750 K de mémoire pour le système et demeure pour l'instant un outil expérimental non distribuable [4] ;

- FINSTER, fondé sur FORMAC, fonctionne sous OS-MFT et utilise des terminaux du type IBM 2260 comme support d'Entrée/Sortie. Il n'offre pas, à l'utilisateur, la possibilité de créer, en cours de session, ses propres procédures [6] ;

- TUTOR, fondé sur FORMAC, ne semble pas avoir dépassé le stade expéri-

mental [13] ;

- de même que SCOPE FORMAC LANGUAGE [20] ;

- et SYMFORM [19] .
  En fait nous avons trés peu de renseignements sur ces trois derniers systèmes.

Il est évident que la liste ci-dessus n'est pas exhaustive, mais elle fait état des principaux systèmes interactifs.

Le système FORDECAL utilise, quant à lui, les possibilités offertes par PL/1-FORMAC du point de vue langage évolué et par CP/67-CMS, système d'exploitation sous lequel il fonctionne. Signalons que FORDECAL peut aussi être utilisé sur un ordinateur IBM 370 avec mémoire virtuelle.

FORDECAL est un système conversationnel au sens "machine de bureau" évoluée : les diverses commandes mises à la disposition de l'utilisateur peuvent en effet être exécutées immédiatement ou non. Bien qu'il ne soit pas possible, avec ce système, d'exécuter un programme avec toute la gamme des instructions du langage PL/1-FORMAC, on peut néanmoins faire exécuter certaines unités de programme.

FORDECAL est fondé sur la langage FORMAC, avec comme langage hôte PL/1 (OS/360 PL/1-FORMAC [21]) mais nous avons restreint ce langage à ses composants FORMAC de base, de plus toutes les variables utilisées dans les instructions de FORDECAL sont des variables de type FORMAC. Les instructions de FORDECAL correspondent aux instructions FORMAC d'OS/360 mais sont écrites dans une syntaxe trés proche de la syntaxe mathématique habituelle ainsi que de celle de PL/1 [8] . FORDECAL peut être utilisé sans connaissance de PL/1 et avec seulement un minimum de connaissance de FORMAC.

Le système FORDECAL est actuellement utilisé à l'Université Technique de BERLIN (R.F.A.), au Zentralinstitüt Für Angewandte Mathematik der Kernforschungsanlage de JULICH (R.F.A.), au Centre National Universitaire de Calcul Electronique de l'Université de PISE (Italie), à l'Institut LAUE-LANGEVIN et à l'Institut de Recherches en Mathématiques Appliquées de GRENOBLE.

2. UN APERCU DU LANGAGE PL/1-FORMAC

2.1.. GENERALITES

Dans sa version batch, FORMAC est un langage qui utilise PL/1 comme langage hôte, nous dirons que FORMAC est une extension de PL/1. Ce pseudo-langage permet de manipuler de façon symbolique les expressions mathématiques usuelles. La grammaire qui produit ces expressions est tout à fait semblable à celle qui produit les expressions arithmétiques en PL/1 mais, de plus, on peut employer des constantes spéciales telles que e, i, π; les nombres peuvent être entiers, flottants ou rationnels. Ces expressions peuvent être comparées, évaluées, différentiées, ...

Historiquement la première version de FORMAC était une extension de FORTRAN (système IBSYS 7090/94), mais la version PL/1 contient de nombreuses améliorations. Toutes les possibilités du langage hôte sont permises (boucles, entrée/sortie, etc ...). Signalons qu'une version FORTRAN IV-FORMAC est aussi disponible pour la série d'ordinateurs IBM 360 [18].

2.2.. SURVOL DES POSSIBILITES DE FORMAC

En utilisant pour des calculs numériques des langages tels que FORTRAN, ALGOL ou PL/1, on est amené à écrire des instructions dites d'affectation, comme :

A = 2.0 ; B = 3 × A ;

Ces instructions ont pour effet de donner la valeur 2.0 à la variable d'identificateur A, et à B la valeur 6.0 (trois fois la valeur de A).

En FORMAC la même possibilité est offerte mais la valeur d'une variable peut ne plus être simplement un nombre mais une expression formelle.

Du point de vue syntaxique, les instructions d'affectation FORMAC se distinguent de celles de PL/1 par la présence du mot clé <LET> placé devant une liste de telles instructions.

Exemple :

LET(A = B ×× 2 + C ×× 2 ; D = X / Y + 4) ;

Sémantiquement, l'interprétation est la suivante :

- toute variable non affectée se représente elle-même, on parlera d'atome ou de variable atomique,
- si l'on ré-affecte une variable, on ne change que sa valeur courante. Il n'y a pas d'effet rétroactif sur les formules dans lesquelles elle apparaissait éventuellement.

Exemple :

LET(A = B ; C = 2 × A ; A = Y) ;

Lorsque cette instruction aura été exécutée, la variable d'identificateur C aura pour valeur l'expression formelle 2 × B et la variable d'identificateur A l'expression Y.

Les variables FORMAC sont simples ou indicées et n'ont pas à être déclarées. Elles ne sont jamais confondues avec des variables PL/1 de même identificateurs. Les variables de type FORMAC et de type PL/1 appartiennent à deux domaines d'existence bien différents. Il est possible, néanmoins, de communiquer entre FORMAC et PL/1.

D'une façon générale une variable de type FORMAC a une existence quelque soit le niveau de programme où elle est utilisée, nous dirons qu'une telle variable a un attribut universel. Nous avons cependant introduit la notion de bloc en FORMAC avec les variables de type FORMAC et d'attribut LOCAL [11], [12].

2.3.. ARITHMETIQUE

Il y a quatre types de constantes numériques en FORMAC :

- les constantes flottantes,
- les constantes entières,
- les constantes rationnelles,
- les constantes spéciales e, i, π notées #E, #I et #P

Si l'on opère sur des entiers ou des rationnels, les calculs se font exac-

tement c'est à dire que le résultat ne peut être qu'entier ou rationnel. Il n'y a pratiquement pas de limitation : un nombre entier peut comporter jusqu'à 2295 chiffres décimaux, nombre d'une grandeur difficilement concevable. Les nombres rationnels peuvent comporter, au maximum, 2295 chiffres décimaux tant au numérateur qu'au dénominateur. Les fractions résultats sont irréductibles. Par contre s'il apparaît un nombre flottant dans une expression, le résultat sera flottant, les calculs étant néanmoins effectués en double précision.

Les constantes spéciales sont sujettes à certaines simplifications automatiques.

Exemple :

LET(A = #E ×× LOG(X) ; B = SIN(#P / 6)) ;

donnera : A ← X ; B ← 1/2

(← signifie : a pour valeur)

Par ailleurs la constante spéciale #I autorise le calcul sur des expressions complexes.

2.4.. FONCTIONS

L'utilisateur a à sa disposition quatre types de fonctions :

- des fonctions tirées de la bibliothèque PL/1 du genre SIN, COS, ..
- des fonctions à valeur entière : FAC, COMB et STEP,
  FAC : factorielle,
  COMB : combinaisons à n éléments,
  STEP : dont la valeur est 1 ou 0 et qui peut être utilisée pour provoquer des "sauts",
- des fonctions variables : ce sont en fait des fonctions non spécifiées dont le nombre d'arguments est fixé et les arguments connus,
- des fonctions utilisateurs qui permettent de définir des relations fonctionnelles portant sur des arguments inconnus au moment de la définition.

2.5.. TRANSFORMATION DES FORMULES

2.5.A. SIMPLIFICATION

FORMAC contient deux classes de fonctions de simplifications. La première classe, MULT - DIST - EXPAND, permet de simplifier des expressions en utilisant les propriétés de la distributivité et des lois multinomiales.

Exemple :

```
LET(X = (A + B) × (A - B) ×× 2 × (C + D);
    Y = MULT(X);
    Z = DIST(X);
    T = EXPAND(X) );
```

on obtient :

$$X \leftarrow (A + B)(A - B)^2(C + D)$$
$$Y \leftarrow (A + B)(A^2 + B^2 - 2AB)(C + D)$$
$$Z \leftarrow (AC + AD + BC + BD)(A - B)^2$$
$$T \leftarrow A^3C + A^3D - A^2BC - A^2BD - AB^2C - AB^2D + B^3C + B^3D$$

La deuxième classe, FRACTN - CODEM, réduit une expression au même dénominateur.

Exemple :

```
LET(X = (A + B/C) / (D + E/F) + G/H;
    Y = FRACTN(X);
    Z = CODEM(X) );
```

on obtient :

$$X \leftarrow \frac{A + \frac{B}{C}}{D + \frac{E}{F}} + \frac{G}{H}$$

$$Y \leftarrow \frac{H(A + \frac{B}{C}) + G(D + \frac{E}{F})}{H(D + \frac{E}{F})}$$

$$Z \leftarrow \frac{HF(AC + B) + GC(DF + E)}{HC(DF + E)}$$

2.5.B. SUBSTITUTION

La substitution est un concept fondamental en manipulation symbolique.

Deux fonctions fournissent cette possibilité :

- EVAL remplace, dans des expressions, des variables atomiques par d'autres expressions.
  Exemple :

```
LET(A = EVAL(X ×× 2 + Y ×× 2, X, 2, Y, COS(T)));
```

  ce qui a pour but de substituer simultanément 2 à X et COS(T) à Y dans l'expression $X^2 + Y^2$, d'où la valeur $4 + COS^2(T)$ affectée à la variable A.

- REPLACE permet de remplacer des sous-expressions par d'autres sous-expressions dans une expression donnée, la substitution se faisant en série.
  Exemple :

```
LET(X = A × (A + B) + C × (A + D);
    Y = EVAL(X, A, B, B, E, C, A);
    Z = REPLACE(X, A, B, B, E, C, A));
```

  on obtient :

$X \leftarrow A(A + B) + C(A + D)$
$Y \leftarrow B(B + E) + A(B + D)$
$Z \leftarrow 2E^2 + A(E + D)$

L'ordre des substitutions est très important dans la fonction REPLACE.

2.5.C. DIFFERENTIATION ANALYTIQUE

On peut, non seulement, dériver des compositions de fonctions connues du genre SIN, sinus, mais aussi des fonctions non spécifiées (fonctions variables).
Exemples :

a)
```
LET(A = X ×× 4 + SIN(X) × COS(X × Y);
    C = DERIV(A, X));
```

on obtient :

$A \leftarrow X^4 + SIN(X) \times COS(X \times Y)$
$C \leftarrow 4X^3 + COS(X) \times COS(X \times Y) - Y \times SIN(X) \times SIN(X \times Y)$

b) LET(A = F.(B × X ×× 2) + X ×× 3;
C = DERIV(A, X);
D = DRV(G.(X ×× 2), $(1)));

on obtient :

$A \leftarrow F.(B \times X^2) + X^3$

$C \leftarrow 2B \times X \times F^{(1)}.(B \times X^2) + 3X^2$

$D \leftarrow G^{(1)}.(X^2)$

où $F^{(1)}.(u)$ signifie $\frac{\delta}{\delta u} F.(u)$

c) LET(Y = F.(3 × X);
DIFF(F) = CHAIN($(1) ×× 2 + A ×× 2);
Z = DERIV(Y, X));

on obtient :

$Y \leftarrow F.(3X)$

$F^{(1)}.(3X) := 9X^2 + A^2$

$Z \leftarrow 3(9X^2 + A^2)$

2.5.D. COMPARAISON

Grâce aux fonctions IDENT et EQUIV [15], [11], [12], il est possible de faire un test d'égalité sur des expressions FORMAC. La valeur de ces fonctions est de type Bit(1) PL/1, de valeur 1 si les expressions sont identiques, c'est à dire ont même forme et mêmes composants, et 0 autrement.

2.5.E. ANALYSE DES EXPRESSIONS

Il est possible d'analyser des expressions grâce aux fonctions COEFF, HIGHPOW, LOWPOW, NUM et DENOM.

- HIGHPOW retourne la plus haute puissance d'une sous-expression donnée dans une expression,
- LOWPOW retourne la plus basse puissance d'une sous-expression donnée dans une expression,

- COEFF permet de sélectionner le coefficient d'une sous-expression déterminée,

- NUM et DENOM retournent, respectivement, le numérateur et le dénominateur d'une expression de la forme a/b.

Exemples :

a) LET(Y = A × X ×× 3 + 2 × A × X + X ×× 4 + Z × X × (A + B);
T = COEFF(Y, X);
U = HIGHPOW(Y, X);
V = LOWPOW(Y, X));

on obtient :

$Y \leftarrow A \times X^3 + 2A \times X + X^4 + Z \times X \times (A + B)$
$T \leftarrow 2A + Z \times (A + B)$
$U \leftarrow 4$
$V \leftarrow 1$

b) LET(X = CODEM(A/B + C/D);
Y = NUM(X);
Z = DENOM(X));

on obtient :

$$X \leftarrow \frac{A \times D + B \times C}{B \times D}$$

$Y \leftarrow A \times D + B \times C$
$Z \leftarrow B \times D$

Il est possible d'autre part de connaitre l'opérateur principal d'une expression, le nombre d'arguments, relativement à cet opérateur principal, de l'expression, de sélectionner un argument déterminé d'une expression grâce aux fonctions LOP, NARGS et ARG.

Exemple :

LET(X = A + C + B + 12);
I = LOP(X); J = NARGS(X);
LET(Y = ARG(2, X));

on obtient :

I ← 24 , code correspondant à l'opérateur +,
J ← 4

Y ← B

Remarque : les fonctions LOP et NARGS retournent une valeur PL/1 d'attribut FIXED BINARY(31,0).

2.5.F. RECUPERATION MEMOIRE

Du fait de la représentation interne des expressions affectées à une variable, il peut être nécessaire, à un moment donné, de récupérer l'espace mémoire occupé par des expressions qui ne servent momentanément plus.

Les fonctions SAVE, ATOMIZE et REMOB (Remove Object [11],[12]) offrent cette possibilité. SAVE se charge de recopier,sur une mémoire auxiliaire à accés direct, la valeur affectée à une variable, ATOMIZE restitue à la liste libre du système FORMAC la mémoire occupée par l'expression affectée à une variable, REMOB, en plus de cette restitution, remet aussi, dans la liste libre du système, l'espace occupé par l'identificateur de variable lui-même. (A noter que la fonction REMOB n'existe pas dans la version FORMAC distribuée par I.B.M.).

2.5.G. LES OPTIONS

On peut contrôler, dans une certaine mesure, le déroulement des opérations du système FORMAC par un jeu d'options : évaluation automatique ou non de toute fonction de type PL/1 dont l'argument est numérique, expansion automatique ou non de toute expression avant affectation, ...

2.5.H. INTERFACE PL/1-FORMAC

Signalons que l'on peut échanger des informations entre FORMAC et PL/1 et vice-versa.

2.5.I. ON-CONDITION FORMAC

D'une manière générale aucune erreur détectée par le système FORMAC n'est fatale, mis à part le cas où l'espace mémoire de travail est trop petit. Néanmoins, l'utilisateur pourra, pour un type d'erreur FORMAC déterminé,

prévoir certaines actions à entreprendre grâce aux ON-CONDITIONs FORMAC.

2.6.. IMPLANTATION

L'interprète PL/1-FORMAC distribué par I.B.M. [21] , JULICH [15] , ou GRENOBLE [12] peut être utilisé sur tout ordinateur IBM/360 ou 370, ayant au moins 256 K de mémoire et fonctionnant sous le contrôle de l'Operating System (PCP, MFT, MVT).

Signalons qu'il existe une version batch de FORMAC fonctionnant sous CP/67-CMS [3] et sous système TSO [7].

2.7.. DEROULEMENT D'UN PROGRAMME FORMAC

Un programme FORMAC est tout d'abord soumis aux restrictions de PL/1 en ce qui concerne l'écriture des cartes. Le module source FORMAC est d'abord confié à un préprocesseur. Il s'agit d'un analyseur, écrit en PL/1, qui va détecter certaines erreurs FORMAC, et surtout "traduire" les instructions FORMAC en des instructions PL/1 [15] , [11] , [12] . Le module préprocessé est ensuite confié au compilateur PL/1 qui fournira un module objet.

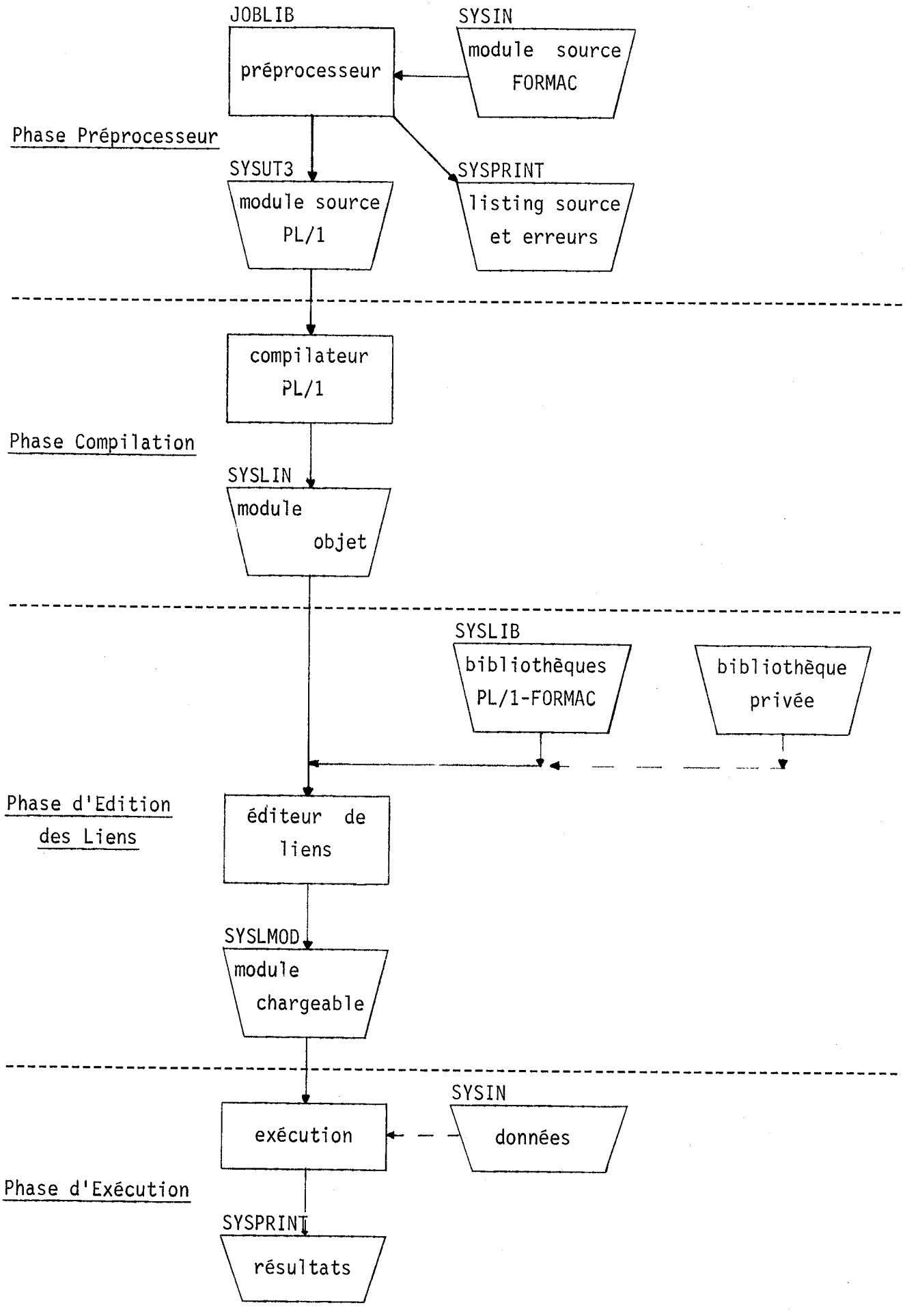
L'éditeur de liens résoudra les références de ce module objet grâce à deux bibliothèques, la bibliothèque PL/1 et la bibliothèque FORMAC.

Finalement le module chargeable obtenu sera exécuté.

On trouvera en page 13 l'organigramme de l'exécution d'un programme FORMAC.

2.8.. LE PREPROCESSEUR

Il existe, à notre connaissance, quatre préprocesseurs, dont les différences résident soit dans le langage avec lequel ils sont écrits, FORTRAN pour un et PL/1 pour les trois autres, soit dans les possibilités qu'ils offrent.


JOBLIB
SYSIN
préprocesseur
module source FORMAC
Phase Préprocesseur
SYSUT3
SYSPRINT
module source PL/1
listing source et erreurs
compilateur PL/1
Phase Compilation
SYSLIN
module objet
SYSLIB
bibliothèques PL/1-FORMAC
bibliothèque privée
Phase d'Edition des Liens
éditeur de liens
SYSLMOD
module chargeable
SYSIN
exécution
données
Phase d'Exécution
SYSPRINT
résultats

Nous ne parlerons que de ceux écrits en PL/1, la version FORTRAN en étant fortement inspirée.

Le premier est celui distribué par I.B.M. Il a le défaut de nécessiter 150 K de mémoire, d'autre part il a été écrit pour le langage mis au point par le Boston Programing Center [21].

Le deuxième est celui distribué par le Z.A.M. - K.F.A. de JULICH [15]. Certaines détections d'erreurs ont été ajoutées, le listing du programme source Formac correspond exactement aux cartes du programme. Il est possible, de plus, de passer des paramètres de traitement au préprocesseur à l'aide de l'instruction EXEC du langage de contrôle des travaux (JCL) : option SOURCE, NOSOURCE, DECK, NODECK, et d'utiliser l'instruction * PROCESS de PL/1 [9]. Signalons d'autre part qu'il ne nécessite que 80 K de mémoire.

Le dernier, distribué par l'I.R.M.A. de GRENOBLE, a tiré parti des deux premiers, mais de nouvelles possibilités y ont été introduites, entrainées par une extension du langage, par exemple l'instruction REMOB, la notion de bloc Formac, les "déclarations" de variables de type Formac et d'attribut local, les "spécifications" de variables de type Formac et d'attribut formel (paramètres Formac de procédures ou de fonction-procédures), un passage plus souple des arguments Formac lors de l'activation d'une procédure ou d'une fonction-procédure, l'appel d'une fonction-procédure à partir d'une instruction Formac, la récursion, ... De même nous avons ajouté de nouvelles options pour le préprocesseur, un nouveau jeu d'erreurs détectables pendant la phase préprocesseur, l'impression séparée des messages d'erreurs classés par ordre de sévérité, la vérification du code retour généré par le préprocesseur afin de déterminer si la phase compilation doit être ensuite activée ou non [12].

3. SURVOL DU SYSTEME FORMAC

3.1.. GENERALITES

Le Boston Programing Center, où a été mis au point le système FORMAC, ayant

été dissous, il nous a été assez difficile d'obtenir, de la part d'IBM, des renseignements concernant ce système. Finalement, le source de la bibliothèque d'exécution de FORMAC et du préprocesseur a été distribué officiellement par IBM, il y a un peu plus d'un an. Malheureusement il s'agit d'une ancienne version de FORMAC ne correspondant pas tout à fait à la version actuellement en exploitation. Il a donc fallu dé-linkéditer et dé-assembler la bibliothèque actuelle avec tous les inconvénients que cela comporte et comparer le source ainsi obtenu avec celui distribué par IBM. C'est un travail long et fastidieux qui commence, néanmoins, à porter ses fruits puisque nous avons, d'ores et déjà, pû modifier et augmenter la bibliothèque des programmes d'exécution de FORMAC.

Le système FORMAC travaille sur des représentations arborescentes des expressions. Les feuilles de ces arbres sont de trois types :

- REPWORD : cellule de 6 mots machine, contient l'ensemble des informations concernant la représentation des atomes (REPresentation WORD)
- NODE : cellule de 2 mots machine, sert de liaisons entre les feuilles. Contient en particulier la représentation des opérateurs,
- HEADER : cellule de 4 mots machine, est utilisé pour grouper des informations concernant des sous-expressions communes.

Tous les identificateurs de variables Formac sont placés dans un REPWORD. La chaîne des repwords commence par l'identificateur <$> et se termine, toujours, par l'identificateur <Z9999999>. Cette chaîne, dont l'adresse de départ est chargée à un endroit fixe du système, contient lors de l'installation du système tous les repwords systèmes (identificateurs des fonctions utilisables en Formac : SIN, DERIV, COMB, ...). Nous avons vu au § 2.2. qu'il n'y a pas de déclaration en Formac standard. Lors de l'exécution d'une instruction Formac, le système va se charger de vérifier si les identificateurs de variables, apparaissant dans l'instruction, appartiennent ou non à la liste des repwords. Cette recherche se fait toujours à partir du premier repword (<$>) jusqu'au dernier, de plus les identificateurs sont rangés par classe, et à l'intérieur de cette classe par or-

dre lexicographique (classe du nombre de caractères constituant l'identificateur). Si l'identificateur de variable n'appartient pas à la liste des repwords, le système se charge de créer un nouveau repword et de l'ajouter une fois les renseignements relatifs à la variable chargés dans ce repword à la liste, dans la classe correspondante (cf. 3° partie).

L'étude, bien que partielle, des routines d'exécution de Formac, nous a permis d'introduire les nouvelles notions dont un bref aperçu a été donné au paragraphe 2.8.

3.2.. COMMENT "FONCTIONNE" FORMAC

Prenons par exemple l'instruction Formac :

```
LET(A = B ×× 2 + C ×× 2 + D × E);          (a)
```

en supposant, afin de faciliter l'exposé, que les variables d'identificateurs B, C, D et E sont atomiques.

Le préprocesseur va transformer cette instruction en :

```
CALL DENFMC1('A = B ×× 2 + C ×× 2 + D × E');          (b)
```

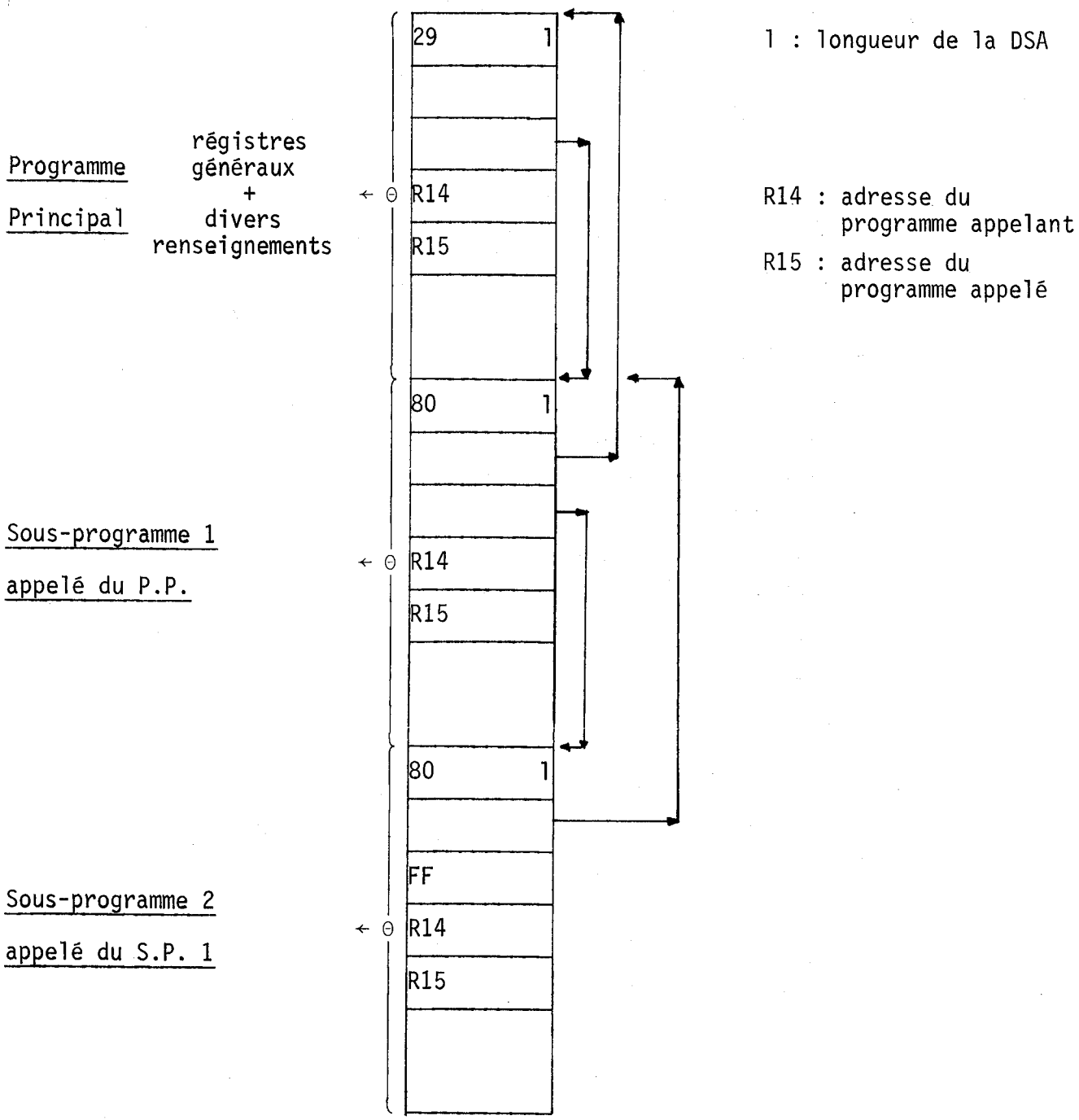
qui est une instruction légale de PL/1. La routine DENFMC1 est une routine de la bibliothèque Formac.

Une première remarque : les routines de Formac admettent, en général, comme arguments des chaines, au sens PL/1.

Ces chaines ne sont pas touchées pendant la compilation du programme. Le compilateur va seulement générer des instructions permettant le chargement de la constante chaine pendant la phase d'exécution.

Rappelons, tout d'abord, quelques généralités concernant les appels de sous-programmes PL/1 à partir d'un programme PL/1. (Ces règles sont les mêmes que celles utilisées par des programmes écrits en langage d'assembleur). Lors de l'activation d'un programme principal PL/1, le système crée par l'intermédiaire de certaines routines de la bibliothèque PL/1 une zone de sauvegarde pour les régistres généraux, les variables utili-

sées, ... (il s'agit du prologue). Cette zone de sauvegarde (Dynamic Storage Area) est pointée par le registre 13. Chaque sous-programme PL/1 appelé à partir du programme principal, ou d'un autre sous-programme, va, de la même façon, créer une zone de sauvegarde qui sera libérée lors du retour au programme appelant (épilogue). Toutes les DSA sont chainées entre elles en avant et en arrière [10].



FORMAC se conforme à ces règles générales.

L'exécution de l'instruction (b) se déroule, grossièrement, de la façon suivante :

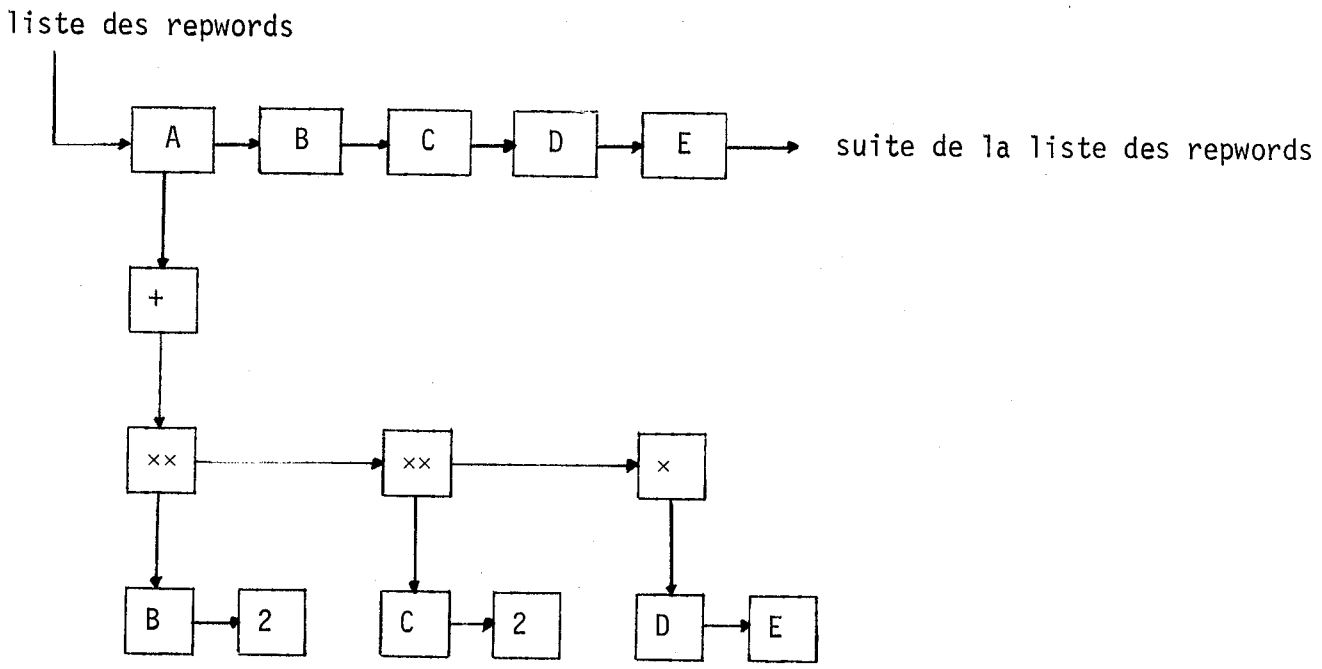
- la routine DENFMC1, qui est en fait un point d'entrée d'une routine plus générale, DENFMC, commence par allouer une DSA de 4 K, puis par sauvegarder les régistres, sauf le régistre 12, avant de se brancher sur une séquence de DENFMC. La DSA, allouée par l'intermédiaire de la routine PL/1 IHESADA, que l'on peut considérer comme une DSA Formac, est chainée aux DSA PL/1 précédentes. De plus DENFMC recopie, dans cette DSA, des renseignements qui ne concernent que PL/1 et qui seront utilisés, par exemple, en cas d'erreur. DENFMC se charge encore de ranger, à une adresse déterminée, la chaine constituant l'argument de DENFMC1 ainsi que sa longueur, ensuite elle fait appel à la routine DENINS qui peut être considérée comme une des plus importantes du système.

- DENINS va analyser, caractère par caractère, l'argument de DENFMC1 en stockant ses résultats dans deux listes appelées ARGPDL (argument push-down list) et OPPDL (operator push-down list). Cette analyse permet d' une part de vérifier que les identificateurs de variables ne font pas plus de 8 caractères, qu'il n'y a pas de caractère interdit, et d'autre part, une fois l'identificateur de variable obtenu, de rechercher dans la chaine des repwords son adresse. Cette recherche se fait toujours à partir du début de la chaine, les identificateurs y étant rangés par classe (classe des identificateurs à 1 caractère, à 2 caractères, ..) et à l'intérieur de la classe par ordre alphabétique. De deux choses l'une, ou l'identificateur figure dans la chaine, auquel cas DENINS va commencer par vérifier si ce n'est pas un identificateur réservé, puis par ranger l'adresse du repword correspondant dans ARGPDL. Si l' identificateur ne figure pas dans la chaine, DENINS va allouer un nouveau repword, le chainer au bon endroit, le remplir par divers renseignements (code atome, remplir la zone nom, ...) et ranger l'adresse de ce nouveau repword dans ARGPDL.

Pendant l'analyse de l'argument de DENFMC1 et son stockage, par élément,

dans les PDL, DENINS construit, en fonction des priorités des opérateurs, l'arbre correspondant à l'expression $A = B^2 + C^2 + D \times E$. D'autres routines importantes du système, à savoir DENCND, DENAUT, DENLXO, ... correspondant à : obtenir une copie d'une expression donnée, simplification, arrangement lexicographique, ... entrent en jeu afin de simplifier, éventuellement, l'expression rangée compte tenu des options mises en place par l'utilisateur et des simplifications automatiques prévues dans le système. Une fois ce travail effectué, l'arbre, ainsi construit, est "accroché" à l'identificateur figurant en partie gauche de l'instruction d'affectation. DENINS rend alors le contrôle à DENFMC qui va recharger les registres précédemment sauvegardés, libérer la DSA Formac et rendre le contrôle au programme appelant, le programme de l'utilisateur, afin d'en poursuivre l'exécution.

Schématiquement, la structure correspondante à l'expression $A = B^2 + C^2 + D \times E$ est la suivante :

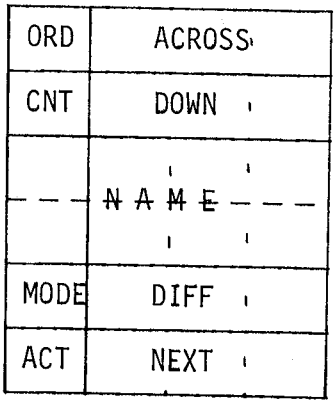


3.3.. LES MODIFICATIONS APPORTEES AU SYSTEME

Compte tenu des renseignements précédents, nous avons pû modifier les rou-

tines DENFMC, DENINS, DENCCSO, ... et ajouter de nouvelles routines. Dans une première étape la bibliothèque ainsi obtenue n'est utilisée que par la version conversationnelle FORDECAL, par la suite elle sera aussi exploitée dans la version batch de FORMAC.

Tout d'abord examinons la représentation d'un repword :



Dans la version distribuée par IBM, nous avons :

ORD : pas utilisé,

ACROSS : null ou pointe sur les repwords contenant les indices dans le cas d'une variable indicée,

CNT : nombre de caractères moins 1 de l'identificateur ou nombre d'indices dans le cas d'un repword-indice,

DOWN : null, cas d'un atome, ou pointe sur la valeur affectée à la variable, cas d'une variable affectée, ou sur le sous-programme correspondant, cas de certaines variables systèmes : DERIV, ...,

NAME : contient les caractères constituant l'identificateur ou les indices rangés par demi-mots,

MODE : utilisé par les repwords systèmes et le simplificateur, ou pour indiquer qu'il s'agit d'un repword-indice,

DIFF : inutilisé ou pointe sur la chaine des dérivées partielles si la pseudo-variable DIFF a été utilisée pour cet identificateur, ou sur la forme différentielle dans le cas d'un repword système,

ACT : code indiquant s'il s'agit d'un atome, d'une variable affectée ou d'un repword système,

NEXT : adresse du prochain dans la liste des repwords, ou de la "racine" dans le cas d'un repword-indice.

Nous avons agi à la fois sur ORD, ACT et DOWN.

ORD = 00 : pour toute variable FORMAC d'attribut universel et pour toute variable système,

ORD = 80 : pour toute variable FORMAC d'attribut local et pour tout paramètre, FORMAC, d'un sous-programme, procédure ou fonction-procédure, n'ayant pas de spécification particulière lors d'un appel par valeur,

ORD = 81 : pour tout paramètre, FORMAC, d'un sous-programme, procédure ou fonction-procédure, n'ayant pas de spécification particulière lors d'un appel par nom, ou ayant la spécification ENTRY.

La routine DENINS a été modifiée en conséquence, de même que DENCCSO qui "installe" les repwords systèmes (ceux classiques de FORMAC plus ceux qu'utilise le système FORDECAL) et les deux pseudo-variables FNC : ABS et EXP$ (voir 2° partie).

Lorsqu'une commande déclarant des variables d'attribut local est exécutée, la routine DENFMCN, ajoutée au système et correspondant à l'appel de DENVARL, est activée avec comme argument la liste des identificateurs de variables figurant dans la commande. Cette routine recherche par l'intermédiaire de DENINS, dans la liste des repwords, l'endroit où devra se placer le repword correspondant à la variable locale, puis installe le nouveau repword, en respectant les règles de création et d'insertion, à ceci près que si un repword de même zone NAME existe déjà, le nouveau est inséré avant. La partie ORD est mise à 80. Ces variables seront ensuite "tuées", par le système lors de l'exécution de la fin de bloc correspondant par la routine DENREMB, ajoutée au système et correspondant à l'appel de DENFMCO. L'utilisateur ne peut, de lui-même, libérer le repword affecté à une variable locale. DENFMCN est d'autre part utilisée par le système, lors de l'activation d'une procédure ou d'une fonction-procédure, pour sauvegarder certaines variables systèmes.

DENPRM1, ajoutée au système et correspondant à l'appel de DENFMCK, est appelée lors du chargement des paramètres de procédures, ou de fonction-procédures, si l'argument est une variable, simple ou indicée,sans spécification particulière, lors d'un appel par nom. DENPRM1 commence par cher-

cher dans la liste des repwords, par l'intermédiaire de DENINS, l'adresse du repword-argument, puis crée un repword pour le paramètre, de la même façon que pour les variables locales, charge dans sa partie DOWN l'adresse du repword-argument et met ORD à 81, de sorte que DENINS saura qu'il s'agit, en quelque sorte, d'un adressage indirect.

DENPRM2, ajoutée au système et correspondant à l'appel de DENFMCL, est activée lors du chargement des paramètres de procédures, ou de fonction-procédures, si l'argument correspondant est une expression. Cette routine commence par confier à DENINS la chaîne suivante : 'variable système = expression' de sorte que DENINS construit l'arbre représentatif de l'expression et l'affecte à une variable d'adresse connue, ensuite DENPRM2 installe le repword correspondant au paramètre, de la même façon que pour les variables locales, avec ORD = 80 et ACT = 30 (code variable affectée), et lui affecte la valeur de la variable système.

DENPRM3, ajoutée au système et correspondant à l'appel de DENFMCM, est appelée lors du chargement des paramètres de procédures, ou de fonction-procédures, si l'argument est une variable, simple ou indicée, sans spécification particulière, lors d'un appel par valeur. DENINS se charge de trouver l'adresse de l'argument, puis DENPRM3 installe le repword-paramètre, avec ORD = 80 et ACT = 30, et charge sa partie DOWN de l'adresse d'un HEADER, construit par DENPRM3 et pointant sur la valeur affectée au repword-argument. (Il s'agit du procédé utilisé par FORMAC dans le cas de sous-expressions comm-nes, pour éviter des recopies). Le problème est un peu plus compliqué si l'argument est la racine d'un tableau, auquel cas il faut, en quelque sorte, recréer des repwords-paramètres-indices et des HEADERs pointant sur la valeur des repwords-indices-arguments.

DENREMB, ajoutée au système et correspondant à l'appel de DENFMCO, est utilisée par le superviseur de FORDECAL pour retourner à la liste libre du système l'espace mémoire occupé par l'expression et par le repword affecté à une variable locale lors de la désactivation du bloc où elle a été déclarée, ou à un paramètre, lors de la désactivation du sous-programme. A noter que dans le cas d'un appel par nom, seul le repword est retourné à la liste libre.

DENRMB1, ajoutée au système et correspondant à l'appel de DENFMCP, est activée lors de l'exécution d'une commande REMOB. Cette routine fonctionne de la même manière que DENREMB sauf qu'elle ne peut libérer des variables créées par le système.

La routine DENINS a été modifiée en conséquence, afin de pouvoir tester le "type" de variable sur laquelle elle est amenée à travailler, en particulier, seul le superviseur de FORDECAL peut utiliser des variables-systèmes spécifiques du système. D'autre part il est possible de redéfinir une pseudo-variable FNC, ce qui est interdit dans la version fournie par IBM, à condition, toutefois, de rester dans la classe des FNC.

Signalons aussi que nous avons modifié la routine DENPNT, routine d'interface avec la routine d'impression, afin de pouvoir obtenir l'impression des résultats FORMAC sur un fichier de type FORMAC, quel que soit le nom de ce fichier; la routine DENOUT, afin de choisir, grâce à une nouvelle option, si les résultats FORMAC imprimés, seront, ou non, soulignés et la routine DENOPT afin d'ajouter de nouvelles options au système.

La routine DENFPRO a été ajoutée afin de permettre l'utilisation, dans des expressions FORMAC, de fonction-procédures.

Remarque : certaines de ces modifications, ou additions, ont été implantées à titre expérimental, sous la même forme ou sous une forme voisine, dans la version batch de FORMAC. La version batch définitive doit être opérationnelle trés rapidement.

3.4.. LES PROJETS

En ce qui concerne

- la version conversationnelle de FORMAC, donner la possibilité à l'utilisateur d'employer un terminal graphique, du type IBM 2250, comme support d'Entrée/Sortie avec utilisation du crayon optique dans l'esprit de ce que Monsieur Y. SIRET avait réalisé pour son système ALADIN [16],

- la version batch de FORMAC, tirer partie de tout ce qui a été développé pour FORDECAL, en l'ajustant pour répondre aux éxigences du langage hôte PL/1, particulièrement en ce qui concerne les procédures, ou fonction-procédures, qui peuvent avoir des arguments PL/1, la technique utilisée dans FORDECAL pour les fonction-procédures n'étant pas directement adaptable en batch du fait que tous les arguments sont de type FORMAC dans FORDECAL, ce qui simplifie, en quelque sorte, le problème. D'autre part essayer de comprendre, complètement, le système FORMAC et par là, le développer.

4. CONCLUSIONS

Le système FORDECAL n'est certainement pas le meilleur système existant à ce jour, il a néanmoins l'avantage de permettre, à l'utilisateur peu averti du langage PL/1 ou FORMAC, la manipulation formelle d'expressions mathématiques de façon interactive. Il est bien entendu que le domaine de manipulation est restreint, pour l'instant, à celui du langage FORMAC qui, nous semble être suffisamment vaste pour un ingénieur, ce qui paraît être confirmé par les physiciens et ingénieurs de l'Institut Laue-Langevin qui l'utilisent. D'autre part nous avons été amenés à approfondir un système existant et considéré commé périmé, et à le développer.

- - - - ☆ - - - -

BIBLIOGRAPHIE

[1] A.C.M. 72 - Proceedings - Annual Conference - BOSTON Aug 1972

[2] C. ENGELMAN - MATHLAB - A program for on line Machine assistance in symbolic computations - Proc. FICC. Nov 1965

[3] G. ENTRESSANGLE
A. LAPLACE
- PL/1-FORMAC sous CP/67-CMS .
Note interne IRMA GRENOBLE 1972

[4] J. GRIESMER
R. JENKS
- The SCRATCHPAD System.
IBM T.J. Watson Research Center - YORKTOWN HEIGHTS NY
RC. 3925 July 1972

- SCRATCHPAD/1 . An interactive facility for Symbolic Mathematics.
RC. 3260 Feb. 1971

[5] A. HEARN - REDUCE 2. An online algebraic manipulation system
Standford Artificial Intelligence Project. 1970

[6] H. HOMRIGHAUSEN
K. KROPIK
H. MEUER
M. OHST
- FINSTER .
A Formac Interactive System for Terminals
KFA. JULICH Okt. 1970

[7] C. JACOBSZ - National Research Institute for Mathematical Science
PRETORIA (South Africa) 1972

[8] IBM System/360 - Operating System
PL/1 (F) Language Reference Manual. GC.28-8201-3

[9] IBM System/360 - Operating System
PL/1 (F) Programmer's Guide. GC.28-6594-6

[10] IBM System/360 - Operating System
PL/1 Subroutine Library - Program Logic Manual -
GY.28-6801-5

[11] A. LAPLACE - PL/1-FORMAC - tome 1 , le langage
IRMA GRENOBLE 1972

[12] A. LAPLACE - PL/1-FORMAC - tome 2 , le préprocesseur
IRMA GRENOBLE 1972

[13] R. LEHMANN
R. WAHLEN - TUTOR
An interactive Graphical Symbolic Mathematics System
Cornell Aeronautical Laboratory, Inc. 1970

[14] Project MAC's - SYmbolic MAnipulation system.
User's Manual . M.I.T. Aug. 1972

[15] R. SCHWERDT - The KFA Formac Preprocessor
KFA-ZAM JULICH. Jul. 1970

[16] Y. SIRET - Contribution au Calcul Formel sur Ordinateur.
Thèse - GRENOBLE Jul. 1970

[17] SEAS XVI - Proc. PISA Sept. 1971

[18] SEAS Library - FORTRAN-FORMAC

[19] R. STONE - SYMFORM Simplified Formac
Western Electric Engineering Research Center
PRINCETON NJ

[20] R. TOBEY
J. LIPSON - The SCOPE FORMAC LANGUAGE
Proc. of the 1968 Summer Institute on Symbolic
Mathematical Computation. Jul 1969

[21] R. TOBEY et al. - PL/1-FORMAC Interpreter - User's Reference Manual
IBM Contributed Program Library 360.D.03.3.004.

- - - - ◆ - - - -

DEUXIEME PARTIE

F O R D E C A L

MANUEL UTILISATEUR

Mieux vaut un bon vieux boulier,
Que deux mauvais ordinateurs.
(Confucius)

1. INTRODUCTION

Le système FORDECAL, exploité sous système CP/67-CMS, est fondé sur les routines d'exécution de FORMAC-OS/360 et n'accepte aucun élément PL/1.

Le dialogue avec FORDECAL se fait par l'intermédiaire des commandes que l'utilisateur envoie au système et par les réponses que fournit ce dernier. Ces commandes sont, pour la plupart, interprétées comme des instructions FORMAC/OS, mais sont écrites dans une syntaxe trés proche de la syntaxe mathématique habituelle ainsi que de celle de PL/1. Néanmoins FORDECAL peut être utilisé sans connaissance de PL/1 et avec seulement un minimum de connaissance de FORMAC.

Le déroulement d'une session de FORDECAL est essentiellement séquentiel, il est toutefois possible de rompre le déroulement séquentiel d'une "unité de programme" ou d'en différer l'"exécution". Signalons qu'à partir du moment où une variable a reçu une affectation, celle-ci est maintenue pendant toute la durée de la session courante tant qu'une nouvelle affectation ne vienne pas modifier la "partie valeur" de cette variable. Ceci ne s'applique évidemment pas aux variables d'attribut local qui n'ont d'existence que dans le bloc où elles ont été "déclarées".

2. UTILISATION DU SYSTEME FORDECAL

L'utilisateur doit tout d'abord se placer dans l'environnement CMS, faire "login" du disque FORDECAL (qui est une extension du disque système) puis taper, à la console, la commande FORDECAL.

Cette commande a pour effet, aprés l'impression suivante :

** FORDECAL **

VERSION 4.5 date de la dernière modification

de charger un module exécutable et de lui donner le contrôle. Ce dernier est effectif à partir du moment où les caractères ===> sont imprimés, en début de ligne, sur le papier du terminal.

Dès que les caractères ===> sont imprimés, l'utilisateur a en mémoire de sa machine un programme exécutable qui va "digérer" les diverses commandes envoyées au système et en provoquer l'exécution.

Au départ d'une session de FORDECAL, le système est dans l'état suivant :

- il n'y a aucune variable affectée,
- le système contient, en plus des fonctions usuelles de FORMAC, deux pseudo-variables FNC (cf. § 14),
- les options standards de FORMAC sont valides,
- les options standards d'Entrée/Sortie sont actives.

Nota : il est conseillé de travailler avec une machine ayant 512 K de mémoire au moins.

3. LES ELEMENTS DU DIALOGUE AVEC FORDECAL

Chaque entité envoyée directement ou indirectement par l'utilisateur et interprétée par le système FORDECAL est appelée une "commande". Avant de décrire la forme syntaxique des diverses commandes de FORDECAL, spécifions les éléments syntaxiques utilisables dans ce système et tirés du langage PL/1-FORMAC.

- les constantes Formac, définies en [3], II, 1.2
- les variables Formac, définies en [3], II, 1.3 d'attribut universel ou local selon [2]
- les expressions Formac, définies en [3], II, 1.4
- les fonctions Formac, définies en [3], II, 1.5
- les chaines Formac, définies en [3], II, 1.6
- les routines Formac, définies en [3], II, 2
- les pseudo-variables DIFF et FNC, définies en [3], II, 1.5 et 2.3

Les éléments syntaxiques de PL/1-FORMAC se rapportant à PL/1, par exemple les variables entre double apostrophes, ne sont pas admis.

Remarques : - les identificateurs de variables sont soumis aux règles classiques de FORMAC/OS : de un à huit caractères alphanumériques, le premier devant être alphabétique, les variables indicées ne peuvent comporter

plus de quatre indices. Nous avons étendu cette règle aux identificateurs des points d'entrée de procédures ainsi qu'aux étiquettes;

- le système utilise, en plus des identificateurs réservés classiques de FORMAC, (DERIV, SIN, COS, ...), les variables d'identificateurs : $SYSTEM1, $SYSTEM2, $SYSTEM3, $SYSTEM4, $SYSTEM5, $SYSTEM6;

- on pourra aussi consulter [2]pour la définition des éléments syntaxiques Formac.

4. LES COMMANDES DE FORDECAL

4.1.. GENERALITES

L'interface entre l'utilisateur et le système est principalement réalisé par C.M.S. grâce à des sous-programmes prévus à cet effet. L'utilisateur a la possibilité d'"envoyer" ses commandes soit à partir du clavier d'une console soit à partir d'un fichier.

Une commande peut être composée sur une ou plusieurs "lignes". Une même ligne peut comporter plusieurs commandes. Une commande ne peut être constituée de plus de 512 caractères alphanumériques, caractères blancs compris. Il est à noter que nous signalerons dans la syntaxe des diverses commandes, la présence d'un caractère blanc (touche d'espacement) qui doit apparaitre à un endroit déterminé de la commande, l'utilisateur pourra néanmoins remplacer cet unique caractère par plusieurs caractères blancs successifs. (présence du caractère <*b*>, sous la ligne, à l'emplacement du caractère blanc nécessaire). Au cours de la composition d'une commande, à partir du clavier de la console, et avant d'en lancer l'exécution, il est possible d'annuler totalement ou partiellement cette composition comme nous le verrons au paragraphe 9.

Signalons que les commandes de FORDECAL consistent en :

- des commandes d'affectation,
- des commandes d'impression,
- des commandes relatives à l'état du système,
- des commandes spéciales,

- des commandes permettant de grouper d'autres commandes,
- des commandes relatives aux procédures,
- des commandes d'annulation,
- une commande de fin de session.

4.2.. TYPES DE COMMANDES

On peut diviser les commandes mises à la disposition de l'utilisateur en trois catégories :

- les commandes à exécution immédiate que nous appelerons aussi commandes élémentaires,
- les commandes à exécution semi-différée ou macro-commandes. Elles formeront des "unités de programme" constituées par des commandes élémentaires,
- les commandes à exécution différée : déclaration de procédure, ou de fonction-procédure, et accompagnateur ou programme de mise au point.

Cette division en catégories résulte de la manière dont opère le système selon le type de commande qui lui est confié.

L'exécution d'une commande élémentaire commence dès que l'utilisateur a enfoncé la touche RETOUR CHARIOT, dans le cas d'une composition au clavier et à condition que la ligne comporte au moins une commande complète, ou dès que la fin physique de la commande est rencontrée dans le cas d' une "lecture" sur fichier.

Dans le cas d'une macro-commande, ou commande de groupe, l'exécution commence dès que la fin physique du groupe est rencontrée.

Pour une commande à exécution différée, c'est l'utilisateur qui en provoquera l'"activation" à l'aide d'une commande appropriée.

Le système fonctionne de la manière suivante :

- dès qu'une commande complète est envoyée, la première phase se

déroule. Il s'agit de la phase lecture-interprétation pendant laquelle le système va se rendre compte de la catégorie à laquelle appartient la commande, l'interpréter et selon les cas, l'enregistrer (cas des commandes appartenant aux catégories 2 ou 3), ou non (catégorie 1);

- une fois cette phase terminée, et toujours selon la catégorie à laquelle appartient la commande, le système va :

  - entrer immédiatement dans la phase d'exécution (cas de la catégorie 1),
  - rester dans la phase de lecture-interprétation avec enregistrement (cas des catégories 2 ou 3) tant que la fin physique de la commande n'est pas décodée.

Lorsqu'on arrivera à cette étape, le système provoquera immédiatement l'exécution de l'unité de programme ainsi enregistrée (cas de la catégorie 2) ou rendra le contrôle à l'utilisateur (cas de la catégorie 3).

On voit que le contrôle est rendu à l'utilisateur de deux manières différentes :

- à haut niveau, avec impression des caractères ===> sur le papier du terminal, une fois la commande élémentaire ou de groupe exécutée, la déclaration de procédure enregistrée;

- à bas niveau, par simple déblocage du clavier, chaque fois qu'une ligne est envoyée et que l'unité de programme n'est pas terminée. L'enregistrement des commandes se fait avant le déblocage du clavier.

4.3.. MESSAGES D'ERREUR

Il existe trois types de messages d'erreur selon la phase au cours de laquelle l'erreur est détectée. Pendant la phase de lecture-interprétation le système signale certaines erreurs de syntaxe, il est alors possible de les corriger immédiatement. Pendant la phase d'exécution, le système signalera les erreurs de syntaxe qui ne sont pas détectées au cours de la première phase et d'autre part le système FORMAC signalera toute erreur de syntaxe, de sémantique, ... conformément à [3].

Tous les messages d'erreurs sont imprimés sur le papier du terminal. Les

messages envoyés par le système FORMAC sont de plus imprimés sur le fichie de sortie en activité. Les divers messages d'erreurs prévus sont décrits aux paragraphes 11 et 12.

Signalons aussi que les éventuelles réponses, fournies par le système FORDECAL, aux commandes se font sur le fichier de sortie en activité à ce moment là.

5. COMMANDES ELEMENTAIRES

5.1.. COMMANDES D'AFFECTATION

5.1.A. AVEC IMPRESSION DU RESULTAT

commande : α) $ *var* = *exp* [, *var* = *exp* , DIFF/FNC (*var*) = *exp*] ;

β) $ DIFF/FNC (*var*) = *exp* [, *var* = *exp* , ...];

action prise par le système : ces commandes correspondent aux instructions PL/1-FORMAC suivantes :

α) PRINT_OUT(*var* = *exp* [; *var* = *exp* ; ...]) ;

β) PRINT_OUT(DIFF/FNC (*var*) = *exp* [; *var* = *exp* ; ...]);

L'expression *exp* est affectée à la variable d'identificateur *var* compte tenu des options en cours.

réponse du système : impression selon le format classique de la routine PRINT_OUT et selon l'option contrôlant cette impression, sur le fichier de sortie.

5.1.B. SANS IMPRESSION DU RESULTAT

commande : α) *var* = *exp* [, *var* = *exp* , DIFF/FNC (*var*) = *exp*] ;

β) DIFF / FNC (*var*) = *exp* [, *var* = *exp* , ...] ;

action prise par le système : ces commandes correspondent aux instructions PL/1-FORMAC suivantes :

α) LET(*var* = *exp* [; *var* = *exp* ; ...]) ;

β) LET(DIFF / FNC (*var*)=*exp* [; *var* = *exp* ; ...]) ;

L'expression *exp* est affectée à la variable d'identificateur *var* compte tenu des options en cours.

réponse du système : aucune

Remarque : l'expression *exp* des § 5.1.A. et 5.1.B. peut être de la forme :
$LOP_b var$ ou $NARGS_b var$ (cf. § 5.4.C. et 5.4.D.)
à condition que la commande ne soit pas multiple.

5.2.. COMMANDE D'IMPRESSION

commande : $ *var* [, *var* ...];

action prise par le système : cette commande correspond à l'instruction PRINT_OUT(*var* [; *var* ...]) ;

réponse du système : identique à celle du § 5.1.A.

Nota : la variable d'identificateur *var* des paragraphes 5.1 et 5.2 peut être indicée ou non

5.3.. COMMANDES CONCERNANT L'ETAT DU SYSTEME

On a regroupé dans ce paragraphe les commandes de "récupération mémoire", de modification des options en cours, permettant de lister divers renseignements concernant les variables utilisées, les options en activité ...

5.3.A. COMMANDE ATOMIZE

commande : ATOMIZE$_b$*var* [, *var* ...] ;

avec : *var* identificateur de variable indicée ou non.

action prise par le système : cette commande correspond à l'instruction :
ATOMIZE(*var* [; *var* ...]) ;

L'espace mémoire occupé par les expressions affectées aux diverses variables d'identificateur *var* est remis dans la liste libre du système.

réponse du système : aucune

5.3.B. COMMANDE REMOB

commande : REMOB$_b$*var* [, *var* ...] ;

avec : *var* identificateur de variable indicée ou non.

action prise par le système : cette commande correspond à l'instruction :
REMOB(*var* [; *var* ...]);

L'espace mémoire affecté aux variables d'identificateur *var* ainsi qu'aux expressions affectées à ces variables est remis dans la liste libre du système. Il est à noter que cette commande, bien que trés pratique, est trés dangeureuse. En effet si l'identificateur de variable *var* est la racine d'un tableau, tout le tableau est effacé.

Nota : l'utilisateur ne peut remober des variables d'attribut local (cf. § 6.5), des paramètres de procédure, des identificateurs de variables réservées, par contre il peut remober des pseudo-variables FNC (à condition, toutefois, qu'elles n'entrent pas dans une des restrictions décrites ci-dessus).

réponse du système : aucune, sauf si l'utilisateur essaie de remober une variable entrant dans un des cas particuliers précédents. (cf. § 12).

5.3.C. COMMANDE OPTSET

Cette commande sert à modifier le jeu des options utilisées en cours de session. Les diverses options utilisables sont de quatre types :

- les options classiques de FORMAC, concernant le traitement des expressions,
- les options concernant les opérations d'Entrée/Sortie,
- les options concernant les fichiers,
- les options concernant le programme de mise au point.

commande : OPTSET$_b$*option* [, *option* ...] ;

action prise par le système : cette commande correspond à l'instruction :
OPTSET(*option* [; *option* ...]) ;

Elle a pour effet de mettre l'*option* "ON" à partir de cet instant. On pourra ensuite remettre cette *option* "OFF" par une commande OPTSET portant sur la "négation" de cette *option*.

réponse du système : aucune

On trouvera ci-après la liste des *options* et leur signification. Les options par défaut sont soulignées, celles précédées d'un astérisque sont des options classiques de FORMAC/OS.

- Options concernant le traitement des expressions

★ EXPND	★ NOEXPND
★ INT	★ NOINT
★ TRANS	★ NOTRANS

- Options concernant les opérations d'Entrée/Sortie

COPY(*nom de fichier*)	NOCOPY
ECHO	NOECHO
★ EDIT	★ NOEDIT
★ LINELENGTH=xxx (120)	
PRINTER(*nom de fichier*)	PRINTER(CONSOLE)
★ PROPER	★ IMPROPER
READER(*nom de fichier*)	READER(CONSOLE)
UNDL	NOUNDL

- Options concernant les fichiers

ERASE(*nom de fichier*)
PRINTF(*nom de fichier*)
REWIND(*nom de fichier*)
SAVEFILE

- Options concernant le programme de mise au point (accompagnateur)

DEBUG NODEBUG

Nota : on trouvera tous les renseignements relatifs aux options précédées d'un astérisque en [2] ou [3].

DESCRIPTION DES DIVERSES OPTIONS

- *nom de fichier* : désigne le fichier sur lequel on va opérer soit en Entrée soit en Sortie. L'identificateur comporte de un à huit caractères alphanumériques, le premier devant être alphabétique. Tous les fichiers créés, ou utilisés, par FORDECAL sont de type FORMAC et de mode P1.

- CONSOLE : désigne un fichier particulier : la console sur laquelle on travaille. En Entrée ce sera le clavier alphanumériqu en Sortie le papier. Si, pour une optio PRINTER ou READER le *nom de fichier* est CONSOLE, il peut être omis.

- Options concernant les opérations d'Entrée/Sortie

- COPY(*nom de fichier*) : permet de copier sur le fichier désigné toutes les commandes composées au clavier du terminal ou lues à partir d'un fichier dont le *nom* doit être différent de *nom de fichier*.

- NOCOPY : supprime l'effet précédent.

Remarque :on ne peut pas faire de copie sur le fichier CONSOLE.

- ECHO : provoque l'impression sur le fichier CONSOLE de toutes les commandes lues.

- NOECHO : supprime l'effet précédent.

- PRINTER
- PRINTER(*nom de fichier*) : les impressions se feront sur le fichier désigné. Si *nom de fichier* est différent de CONSOLE, on pourra ensuite le faire imprimer, sur l'imprimante standard, soit à partir de FORDECAL soit de CMS. A noter que le *fichier* ainsi créé n'est pas automatiquement ré-exploitable en lecture par FORDECAL.
  Nota : si *nom de fichier* est différent de CONSOLE, les messages d'erreurs sont aussi imprimés sur CONSOLE;
  un fichier déterminé, sauf CONSOLE, ne peut être utilisé en même temps comme Entrée et comme Sortie.

- READER
  READER(*nom de fichier*) : l'Entrée des commandes se fait à partir du fichier désigné.
  Nota : un fichier déterminé, sauf CONSOLE, ne peut être utilisé en même temps comme Entrée et comme Sortie.

- UNDL : l'impression des résultats provoquée par le système FORMAC se fait selon le format classique : ces résultats sont soulignés.

  Nota : un fichier résultat créé de cette façon n'est pas exploitable en lecture par le système.

- NOUNDL : l'impression des résultats provoquée par le système FORMAC se fait sans les "soulignés".
  Nota : un fichier résultat créé de cette façon, avec l'option NOEDIT et en ajou-

tant un <;>peut, éventuellement, être exploité en lecture par le système.

- Options concernant les fichiers

- ERASE(*nom de fichier*) : cette option permet d'"effacer" le fichier désigné à partir de FORDECAL. (Analogue de la commande ERASE de CMS)

- PRINTF(*nom de fichier*) : permet de faire imprimer, sur l'imprimante standard et à partir de FORDECAL, le fichier désigné. (Correspond à la commande OFFLINE PRINT de CMS).

- REWIND(*nom de fichier*) : permet de fermer le fichier désigné. Toute nouvelle lecture se fera en début de fichier, par contre toute nouvelle écriture se fera à partir de la dernière ligne écrite.

- SAVEFILE : permet de sauvegarder les fichiers créés depuis le début de la session courante dans l'état où ils se trouvent.

Nota : - en règle générale, si l'on travaille avec un fichier, toute nouvelle lecture ou écriture se fait à partir du dernier enregistrement lu ou écrit du fichier. Si l'on veut relire un fichier il faut envoyer, auparavant, l'option REWIND, par contre si l'on veut écraser un fichier, il faut envoyer, en premier, l'option ERASE;

- l'utilisateur a la possibilité, en cours de lecture, de modifier le nom de fichier à partir duquel doit continuer la lecture. Un empilage automatique des noms de fichiers est réalisé par le système, il est alors possible de revenir automatiquement au fichier précédent si l'on arrive en fin de fichier. Toute fin de fichier en lecture, provoque le retour au fichier lecture précédent. A noter que l'on ne garde que 4 noms de fichiers au maximum.

- Options concernant le programme de mise au point

Les deux options concernées ne peuvent apparaitre dans une commande OPTSET que si celle-ci n'appartient pas à une unité de programme, que ce soit une macro-commande ou une déclaration de procédure.

On se reportera au paragraphe 8 pour tout ce qui concerne le programme de mise au point.

5.3.D. COMMANDE LIST

Cette commande permet de faire imprimer, à tout moment, sur le fichier de sortie en activité :

α) la liste
- des options en activité,
- des variables affectées,
- des atomes,
- des pseudo-variables DIFF,
- des pseudo-variables FNC,
- des points d'entrée de procédure,

β) l'espace mémoire, exprimé en octets,
- occupé par
  - le corps de toutes les procédures enregistrées et/ou les macro-commandes en activité,
  - les variables et leurs expressions,
- libre

selon l'argument de la commande.

commande : LIST$_b$ *argument* [, *argument* ...] ;

avec : *argument* un des mots clé suivant :
OPTION, VARS, ATOMS, DIFF, FNC, PROCEDURE, CORE

action prise par le système : OPTIONS : consultation d'une table contenant les options actives,

VARS, ATOMS, DIFF, FNC : examen de l'indicateur état des repwords

PROCEDURE : consultation de la table des points d'entrée,

CORE : examen des indicateurs d'occupation mémoire

réponse du système : ACTIVE OPTIONS LIST :
liste des options en activité

ASSIGNED VARIABLES LIST :
NOTHING dans le cas ou aucune variable affectée n'existe ou *liste des variables affectées*

ATOMS LIST :
NOTHING *ou liste des atomes*

DIFF PSEUDO-VARIABLES LIST :
NOTHING ou *liste des pseudo-variables DIFF*

FNC PSEUDO-VARIABLES LIST :
ABS,EXP$ suivis éventuellement de la *liste des pseudo-variables FNC*

USER'S ENTRY NAMES LIST :
NOTHING *ou liste des points d'entrée des procédures*

```
PROGRAM-UNIT USED CORE = xxxxxx
FORMAC       USED CORE = xxxxxx
REMAINING FREE CORE    = xxxxxx
```

5.4.. COMMANDES SPECIALES

Ce paragraphe décrit les commandes mettant en oeuvre des routines de PL/1 -FORMAC utilisées habituellement dans des instructions PL/1. Ces routines serviront ici à interroger le système sur l'état de variables ou d'expressions.

5.4.A. COMMANDE IDENT

commande : IDENT$_b$ *exp1* ! *exp2* ;

avec : *exp1* et *exp2* des identificateurs de variables, indicées ou non, ou des expressions.

action prise par le système : cette commande correspond à IDENT(*exp1* ; *exp2*) de PL/1-FORMAC. Le système compare, par l'intermédiaire d'une routine de la bibliothèque Formac, les arbres représentatifs de *exp1* et *exp2* et répond oui ou non, selon que ces arbres sont identiques ou non.

réponse du système : YES ou NO

5.4.B. COMMANDE EQUIV

commande : EQUIV$_b$*exp1* ! *exp2* ;

avec : *exp1* et *exp2* des identificateurs de variables, indicées ou non, ou des expressions.

action prise par le système : cette commande correspond à IDENT(EXPAND(*exp1*) ; EXPAND(*exp2*)) ou à EQUIV(*exp1* ; *exp2*) de PL/1-FORMAC [2] , [3]
Le système compare les arbres représentatifs de *exp1* et *exp2*, mis sous forme entièrement développée, et répond oui ou non selon qu'il y a identité ou non.

réponse du système : YES ou NO

5.4.C. COMMANDE LOP

commande : LOP$_b$*var* [, *var* ...] ;

avec : *var* identificateur de variable indicée ou non.

action prise par le système : cette commande correspond à l'instruction LOP(*var*) de PL/1-FORMAC. Le système recherche, par l'intermédiaire d'une routine de la bibliothèque Formac, l'opérateur principal de l'expression affectée à *var*.

réponse du système : LOP OF *var* = *valeur*

Nota : pour l'interprétation de *valeur* se référer à [2] ou [3]

5.4.D. COMMANDE NARGS

commande : $NARGS_b$*var* [, *var* ...] ;

avec : *var* identificateur de variable indicée ou non.

action prise par le système : cette commande correspond à l'instruction NARGS(*var*) de PL/1-FORMAC. Le système recherche, par l'intermédiaire d'une routine de la bibliothèque Formac, le nombre d'arguments, par rapport à l'opérateur principal, de l'expression affectée à la variable *var*.

réponse du système : NARGS OF *var* = *valeur*

5.5.. COMMANDE GET

Cette commande permet de "lire" des "données" à partir d'un fichier et de les affecter à une variable déterminée.

commande : GET_b[(*nom de fichier*)] *var* [, *var* ...] ;

avec : *nom de fichier* : identificateur du fichier sur lequel doit se faire la lecture des données. Si *nom de fichier* est omis, la lecture se fait sur le fichier CONSOLE (cf. § 5.3.C. p. 10);

var : identificateur de variable indicée ou non, que l'on désire affecter.

action prise par le système : le système commence par se brancher sur le *fichier* désigné, puis va effectuer la lecture. Une fois la commande exécutée, le système se rebranche sur le fichier-lecture en activité avant l'opération lecture-données.

réponse du système : le message suivant est imprimé avant que ne démarre l'entrée des données :

** INPUT DATA **

Si la lecture se fait sur le fichier CONSOLE, le clavier se débloque, momentanément, pour permettre l'introduction des données.

5.6.. COMMANDE CALL

Cette commande permet d'activer une procédure précédemment enregistrée.

commande : CALL$_b$*procname* [(*liste des arguments*)] ;

avec : *procname* : l'identificateur de la procédure à activer,

liste des arguments : la suite des identificateurs de variables constituant la liste d'appel.

- Les arguments doivent être séparés, les uns des autres, par une virgule (<,>);
- les arguments peuvent être des constantes numériques, des expressions, des identificateurs de variables simples (indicées ou non), des identificateurs de tableau, des points d'entrée de procédures, ou de fonction-procédure, des noms de fichiers, des identificateurs d'étiquettes;
- l'appel peut se faire par nom ou par valeur. Dans ce dernier cas l'identificateur doit être placé entre parenthèses (cf. [1] Ch. 12).

action prise par le système : consultation de la table des points d'entrée pour déterminer si la procédure d'identificateur *procname* existe. Dans l'affirmative il y a, éventuellement, chargement des paramètres par la valeur des arguments puis activation de la procédure.

réponse du système : conforme aux commandes élémentaires figurant dans le corps de la procédure invoquée.

5.7.. COMMANDE SCRATCH

Cette commande permet d'"effacer" une procédure précédemment enregistrée.

commande : SCRATCH$_b$*procname* ;

avec : *procname* l'identificateur de la procédure à effacer

action prise par le système : consultation de la table des points d'entrée pour déterminer si la procédure d'identificateur *procname* existe. Si oui, le système récupère la zone de mémoire occupée par le corps de la procédure et enlève l'identificateur *procname* , ainsi que ses "alias", de la table (cf. § 7).

réponse du système : ** PROCEDURE " [1] " SCRATCHED **
suivi de
AS ALIAS " [2] "

dans le cas où la procédure a plusieurs points d'entrée équivalents. On trouve en [1] l'identificateur, en [2] l'alias de la procédure ainsi effacée.

5.8.. COMMANDE STOP

Cette commande est prévue pour quitter l'environnement du système FORDECAL.

commande : STOP ;

action prise par le système : le système sauvegarde tous les fichiers créés au cours de la session, le superviseur rend le contrôle à C.M.S.

réponse du système : aucune, par contre CMS imprime son message habituel de fin d'exécution d'une commande CMS.
(R ... temps écoulé ...)

6. MACRO-COMMANDES

Cette catégorie de commandes regroupe le premier type d'unité de programme. Il s'agit, en fait, de commandes élémentaires encloses par une commande particulière de façon à former un groupe de commandes. Il y a trois groupes distincts :

- le groupe DO non itératif,
- le groupe DO itératif,
- le bloc BEGIN.

La définition syntaxique de ces trois groupes est identique à celle de PL/1.

Une macro-commande pourra comporter :

- une, ou plusieurs, commande(s) élémentaire(s),
- des commandes de rupture de séquence,
- des commandes conditionnelles,
- des commandes d'insertion de commentaires,
- des commandes d'impression de messages,
- des commandes de "déclaration" de variables locales, uniquement dans le cas du bloc BEGIN,
- une, ou plusieurs, macro-commande(s).

Une macro-commande commence par un mot clé qui détermine le groupe auquel se rattache la macro-commande, et se termine par la pseudo-commande END qui en marque la fin physique.

Signalons que toute commande appartenant à une macro-commande peut être étiquetée. Les règles concernant les étiquettes sont celles de PL/1 en ce qui concerne la portée de l'étiquette, avec, en plus, la restriction à huit caractères alphanumériques au plus en ce qui concerne l'identificateur lui-même. Il est à noter que l'unité de programme ainsi constituée n'a pas d'existence en dehors d'elle même.

Les commandes de rupture de séquence (GOTO), conditionnelles (IF), d'insertion de commentaires (COMMENT), d'impression de messages (PUT) et de déclaration de variables locales (LOCAL) n'ayant pas d'existence logique en dehors d'une unité de programme seront appelées sous-commandes. Leur utilisation en dehors d'une unité de programme, sauf dans le cas des sous-commandes COMMENT et PUT, provoquera l'im-

pression d'un message d'erreur (cf. § 11).

Rappelons qu'une macro-commande passe par deux étapes. Dès que le système décode le mot clé signalant le début d'une macro-commande, la phase de lecture-interprétation se charge, en plus, du stockage des diverses commandes incluses dans le groupe. Signalons néanmoins que la sous-commande COMMENT ne donne lieu à aucune réservation mémoire. Le contrôle est rendu, si le groupe est composé sur plusieurs lignes, à l'utilisateur, à bas niveau (déblocage du clavier alphanumérique), tant que la pseudo-commande END fermant le bloc le plus extérieur n'est pas décodée. La phase d'exécution commence alors par la première commande physiquement enregistrée et se poursuit en déroulant logiquement le "train" de commandes de l'unité de programme que le système vient d'enregistrer. Cette phase se termine lorsque le groupe de commandes est épuisé, ou qu'une erreur est détectée par le superviseur de FORDECAL, le contrôle est alors rendu, à haut niveau, à l'utilisateur dès que le système a "effacé" l'unité de programme.

6.1.. SOUS-COMMANDE COMMENT

Cette commande permet d'insérer des commentaires dans une unité de programme. Ces commentaires sont ignorés par le système.

commande : COMMENT$_b$*commentaire* ;

avec : *commentaire* n'importe quel texte alphanumérique ne comportant pas le caractère <:>

action prise par le système : aucune

réponse du système : aucune

Nota : cette commande peut, éventuellement, être utilisée en dehors de toute unité de programme.

6.2.. SOUS-COMMANDE PUT

Cette commande permet l'impression d'un message pendant la phase d'exécution.

commande : PUT$_b$*message* ;

avec : *message* n'importe quel texte alphanumérique ne comportant pas le caractère <:>

action prise par le système : conforme à celle des commandes appartenant à une unité de programme :
- pendant la phase de lecture : stockage dans une zone de mémoire de la commande,
- pendant la phase d'exécution : réponse du système

réponse du système : impression, sur le fichier de sortie, du texte composant le *message*.

Nota : se reporter à la note du § 6.1.

6.3.. SOUS-COMMANDE GOTO

Cette commande permet de rompre le déroulement séquentiel de l'unité de programme en renvoyant le superviseur sur une commande déterminée, appartenant au groupe et située, soit dans le même bloc au même niveau, soit dans un bloc englobant.

commande : GOTO$_b$*étiquette* ;

action prise par le système : pendant la phase d'exécution, le superviseur provoquera un déroutement sur la commande préfixée par *étiquette*.

réponse du système : aucune

6.4.. SOUS-COMMANDE IF

Cette commande permet d'introduire la notion de commande conditionnelle.

commande : IF$_b$*test*$_b$THEN$_b$*commande* [ELSE$_b$*commande*]

où : *test* peut prendre l'une des formes suivantes :

a) IDENT$_b$*exp1* ! *exp2*

b) EQUIV$_b$*exp1* *exp2*

c) LOP$_b$*var*$_b$SY$_b$*exp3*

d) NARGS$_b$*var*$_b$SY$_b$*exp3*

e) *exp3*$_b$SY$_b$*exp4*

f) *exp5*$_b$|$_b$*exp6*

g) *exp5*$_b$&$_b$*exp6*

avec : *var* : identificateur de variable, indicée ou non;

exp1, *exp2* : identificateurs de variables, indicées ou non, ou expressions;

exp3, *exp4* : identificateurs de variables, indicées ou non, ou expressions. La valeur courante de *exp3* et *exp4* doit être numérique (de type flottant, entier ou rationel);

exp5, *exp6* : une quelconque des formes a) à e) ci-dessus;

SY : l'un quelconque des symboles suivants :

- `>` supérieur à
- `>=` supérieur ou égal à
- `¬>` non supérieur à
- `=` égal à
- `¬=` différent de
- `<=` inférieur ou égal à
- `<` inférieur à
- `¬<` non inférieur à

commande : n'importe quelle commande décrite aux paragraphes 5 et 6

action prise par le système : - pendant la phase de lecture : préparation des branches THEN et ELSE,

- pendant la phase d'exécution : le superviseur se branchera sur la séquence THEN ou ELSE selon la valeur de *test*

réponse du système : conforme à celle de la *commande* figurant dans la branche effectivement exécutée.

6.5.. SOUS-COMMANDE LOCAL

Cette commande constitue en fait une déclaration de variables d'attribut local. Cette commande ne peut apparaitre que dans une déclaration de procédure (cf. § 7) ou dans un bloc BEGIN. Les variables figurant dans la commande n'auront d'existence que dans le bloc où elles sont déclarées et dans tout bloc contenu, sauf nouvelle déclaration portant sur le même identificateur (règles classiques de PL/1 en ce qui concerne la portée des déclarations). Plusieurs commandes LOCAL peuvent figurer dans un même bloc, elles seront toutes regroupées en une seule placée physiquement immédiatement aprés la commande marquant le début du bloc.

Remarque : il n'y a pas d'analyse en ce qui concerne les éventuelles redondances d'identificateurs de variables locales.

commande : LOCAL$_b$ *var* [, *var* ...] ;

avec : *var* identificateur de variable simple, même si celle ci doit être utilisée comme tableau.

action prise par le système : pendant la phase d'exécution, le système va créér dynamiquement une variable Formac d'identificateur *var*. L'espace mémoire alloué par le système Formac à cette variable, et à sa valeur, sera ensuite remis dans la liste libre du système Formac à la sortie du bloc où figurait la déclaration LOCAL.

réponse du système : aucune

6.6.. COMMANDE DO NON ITERATIVE

Cette commande permet de former une unité de programme au sens groupe DO non itératif [1]

commande : DO;
commande [*commande* ...]
END ;

avec : *commande* : n'importe quelle commande décrite aux paragraphes 5 et 6, sauf celle décrite en 6.5.

END ; : pseudo-commande marquant la fin physique du groupe

action prise par le système : - pendant la phase de lecture : chainage, en mémoire, des diverses *commandes* ,

- pendant la phase d'exécution : le système déroule le train de *commandes* à partir de la première enregistrée jusqu'à la commande END, dans l'ordre logique d'exécution.

réponse du système : à chaque exécution d'une *commande* le système répond selon le modèle prévu pour celle-ci.

6.7.. COMMANDE DO ITERATIVE

Cette commande permet de former une unité de programme au sens groupe DO itératif, ce qui autorise l'exécution répétitive et sous contrôle de l'utilisateur du groupe de commandes [1]

commande : $DO_b vc{=}vi_b TO_b vf$ [$_b BY_b ic$] ;
commande [, *commande* ...]
END ;

avec : *vc* : variable contrôlée de boucle,

vi : valeur initiale de la boucle,

vf : valeur finale de la boucle,

ic : incrément. S'il est omis, l'incrément est pris égal à 1,

commande : n'importe quelle commande décrite aux paragraphes 5 et 6, sauf celle décrite en 6.5,

END ; : pseudo-commande marquant la fin physique du groupe.

Nota : *vi*, *vf* et *ic* sont soit des identificateurs de variables, simples ou indicées, dont la valeur au moment de l'exécution de la boucle doit être numérique, soit des expressions arithmétiques de type entier, flottant ou rationnel, soit, et uniquement en ce qui concerne *vi* et/ou *vf* $NARGS_b var$.

action prise par le système : - pendant la phase de lecture : toute boucle est testée avant d'être éventuellement enregistrée. Si la boucle la plus extérieure n'est pas exécutable, le contrôle est rendu immédiatement, à haut niveau, à l'utilisateur,

- pendant la phase d'exécution : déroulement de la boucle et des *commandes* qui y figurent. Toute boucle est testée avant d'être exécutée, conformément à la technique de PL/1.

réponse du système : à chaque exécution d'une *commande* le système répond conformément au modèle prévu pour cette commande.

Nota : - il est possible de redéfinir la valeur de la variable contrôlée de l'intérieur de la boucle,

- à la fin de l'exécution d'une boucle intérieure, la variable de contrôle a la valeur *vf* + *ic*,

- à la fin de l'exécution de la boucle la plus extérieure, les variables de contrôle ont une valeur indéfinie.

6.8.. COMMANDE BEGIN

Cette commande permet de former une unité de programme au sens bloc BEGIN. Il est possible de faire figurer à l'intérieur d'un tel bloc une, ou plusieurs, déclaration(s) de variables locales.

commande : BEGIN ;
commande [*commande* ...]
END ;

avec : *commande* : n'importe quelle commande décrite aux paragraphes 5 et 6,

END ; : pseudo-commande marquant la fin physique du bloc.

action prise par le système : identique à celle prise pour le groupe DO non itératif.

réponse du système : conforme à la réponse prévue pour chaque *commande* exécutée.

7. COMMANDE DE DECLARATION DE PROCEDURE OU DE FONCTION-PROCEDURE

Cette commande constitue le deuxième type d'unité de programme. Il s'agit d'une véritable déclaration de procédure, ou de fonction-procédure, avec ses points d'entrée, la liste de ses paramètres et le corps du sous-programme.

Nous avons, autant que possible, suivi la syntaxe de PL/1 [1, Ch. 12] en ce qui concerne ces sous-programmes, mis à part le fait que l'identificateur d'un point d'entrée ne peut comporter plus de huit caractères alphanumériques et que l'on ne peut pas déclarer un nouveau sous-programme à l'intérieur de celui que l'on est en train de définir. De plus ces sous-programmes n'ont qu'une seule "entrée", par contre cette entrée peut avoir plusieurs identificateurs, ces entrées sont dites équivalentes.

Un sous-programme pourra être récursif ou non, comporter des paramètres ou non.

Le corps d'un sous-programme pourra comporter :

- des commandes de spécification des paramètres,
- une, ou plusieurs, commande(s) élémentaire(s),
- des commandes de rupture de séquence,
- des commandes conditionnelles,
- des commandes d'insertion de commentaires,
- des commandes d'impression de messages,
- des commandes de déclaration de variables locales,
- une, ou plusieurs, macro-commande(s)
- une, ou plusieurs, commande(s) de retour au programme appelant.

Une déclaration d'un sous-programme comporte dans sa première partie un mot clé et se termine par la pseudo-commande END, éventuellement suivie d'une étiquette, qui en marquera la fin physique.

Signalons que toute commande appartenant au corps d'un sous-programme peut être étiquetée. (cf. § 6)

Rappelons que lors de la composition du sous-programme au clavier du terminal le contrôle est rendu, à bas niveau, à l'utilisateur tant que la fin physique du sous programme n'est pas décodée.

Lors de la lecture de l'"entête" de la déclaration du sous-programme, le système commence par vérifier dans la table des points d'entrée que l'identificateur de point d'entrée qui vient d'être décodé, n'existe encore pas. Dans la négative on consultera le paragraphe 11, dans l'affirmative le nouveau point d'entrée est ajouté à la table puis le système se prépare à chainer le train de commandes constituant le corps du sous-programme.

Rappelons que les multiples déclarations de variables locales d'un même bloc sont regroupées en une seule placée physiquement immédiatement après la commande ouvrant le bloc. Signalons aussi que l'entête de la procédure, ou de la fonction-procédure, est considérée équivalente à un bloc BEGIN en ce qui concerne la commande LOCAL.

L'activation d'une procédure sera provoquée ultérieurement par l'utilisateur, à

condition que la procédure invoquée se trouve en mémoire, par la commande CALL (cf. § 5.6). Il est possible d'autre part d'effacer une procédure par la commande SCRATCH (cf. § 5.7), à moins que la procédure ne soit en activité.

L'activation d'une fonction-procédure sera provoquée simplement par la présence de l'identificateur d'un de ses points d'entrée dans une commande d'affectation, à condition, bien entendu, que cette fonction-procédure se trouve en mémoire. Signalons à ce sujet que le système Formac interprétera toute référence à une fonction-procédure non enregistrée, comme un appel de variable indicée. Pour effacer une fonction-procédure, on utilisera aussi la commande SCRATCH.

Rappelons enfin qu'il est possible de "copier", sur un fichier déterminé, autre que le fichier de lecture, un sous-programme en cours de composition (cf. § 5.3.C. option COPY) ou de "lire" un sous-programme à partir d'un fichier (cf. § 5.3.C. option READER). Il est aussi possible de composer un sous-programme sous C.M.S., à l'aide de la commande EDIT, le fichier ainsi créé devra être de type FORMAC et son nom ne devra pas comporter plus de huit caractères alphanumériques.

Remarque : les trois commandes de spécification de paramètres n'étant utilisables qu'à l'intérieur du corps d'un sous-programme seront, elles aussi, appelées sous-commandes.

7.1.. COMMANDE PROCEDURE

Cette commande a pour but de former une unité de programme au sens procédure.

commande : *p.e* : [*p.e* :..] PROCEDURE [(*l.p*)] [RECURSIVE] ;
commande [*commande* ...]
END ;

avec : *p.e* : identificateur d'étiquette qui représente l'identificateur de la procédure que l'on va définir. Il est possible d'avoir plusieurs *points d'entrée* équivalents,

l.p : liste des paramètres, séparés les uns des autres par une virgule (<,>).

Paramètre : identificateur de variable simple,

RECURSIVE : signale que la procédure sera récursive,

commande : n'importe quelle commande décrite aux paragraphes 5, 6, 7.3. à 7.8

END ; : cette pseudo-commande marque la fin physique du corps de la procédure. Une fois ce mot clé décodé, le contrôle est rendu, à haut niveau, à l'utilisateur.

action prise par le système : mise à jour de la table des points d'entrée et chainage des *commandes* en mémoire.

réponse du système : ** PROCEDURE " 1 " RECORDED **
éventuellement suivi de
WITH ALIAS " 2 "

En 1 figure le dernier des *points d'entrée* apparu dans la déclaration, en 2, éventuellement, les *points d'entrée* équivalents. Il sera ensuite possible d'invoquer la procédure en utilisant l'un quelconque de ses *points d'entrée*.

Remarques : - dans chacune des *commandes* peuvent apparaitre soit des paramètres, soit des variables locales soit des variables universelles,

- deux commandes, parmi celles utilisables dans le corps d'une procédure, peuvent avoir une syntaxe différente de celle décrite au paragraphe 5. Nous donnerons ci-aprés, § 7.6. et 7.7, cette deuxième syntaxe utilisable uniquement à l'intérieur du corps d'un sous-programme.

7.2.. COMMANDE FONCTION-PROCEDURE

Cette commande a pour but de former une unité de programme au sens fonction procédure.

commande : *p.e* : [*p.e* : ..] PROCEDURE [(*l.p*)] [RECURSIVE] RETURNS

commande [*commande* ...]

END ;

avec : RETURNS : spécifie que ce sous-programme est du type fonction-procédure.

Se reporter au § 7.1. pour la signification de *p.e*, *l.p* et RECURSIVE

action prise par le système : identique à celle du § 7.1

réponse du système : identique à celle du § 7.1

Remarque : une fonction-procédure doit nécessairement retourner une valeur par la commande RETURN (cf. § 7.8.B.)

7.3.. SOUS-COMMANDE ENTRY

Cette commande a pour but de spécifier que certains paramètres ont la spécification ENTRY [1] et qu'en conséquence les arguments correspondant seront des identificateurs de points d'entrée de sous-programmes.

commande : ENTRY$_b$ *var* [, *var* ...] ;

avec : *var* identificateur de variable figurant dans la liste des paramètres.

action prise par le système : le système prépare une utilisation ultérieure de *var* comme identificateur de sous-programme, soit pour une commande CALL, soit pour une commande SCRATCH, soit dans une commande d'affectation.

réponse du système : aucune

7.4.. SOUS-COMMANDE LABEL

Cette commande a pour but de spécifier l'attribut étiquette de certains

paramètres [1]. Les arguments correspondant devront être des étiquettes;

commande : LABEL$_b$*var* [, *var* ...] ;

avec : *var* identificateur de variable figurant dans la liste des paramètres.

action prise par le système : le système prépare une utilisation ultérieure de *var* dans une commande GOTO *var* . Pendant la phase d'exécution de la commande GOTO *var* le contrôle sera transféré à l'instruction préfixée par l'argument étiquette correspondant à *var*. Il va sans dire que cette instruction doit appartenir au bloc qui contient la référence au sous-programme activé.

réponse du système : aucune

7.5.. SOUS-COMMANDE FILE

Cette commande a pour but de spécifier l'attribut nom de fichier de certains paramètres. Les arguments correspondant devront être des noms de fichiers de type Formac.

commande : FILE$_b$*var* [, *var* ...] ;

avec : *var* identificateur de variable figurant dans la liste des paramètres.

action prise par le système : le système prépare une utilisation ultérieure de *var* soit dans une commande GET (cf. § 5.5) soit dans une commande OPTSET portant sur des fichiers (cf. § 5.3.C.).

réponse du système : aucune

Remarque : il peut y avoir plusieurs sous-commandes de spécification à l'intérieur d'une procédure, ou d'une fonction-procédure, elles sont toutes regroupées, par type, en une seule placée physiquement immédiatement aprés l'entête du sous-programme. Les éven-

tuelles redondances, aprés avoir été signalées, sont ignorées. Rappelons que les sous-commandes décrites aux paragraphes 7.3, 7.4 et 7.5 ne sont utilisables que dans le corps d'un sous-programme.

7.6.. COMMANDE CALL

Cette commande permet d'activer une procédure à partir d'un sous-programme et, comme cas particulier, de ré-activer la même procédure lors d'une récursion.

commande : CALL$_b$*procname* [(*liste des arguments*)] ;

avec : *procname* : soit l'identificateur effectif d'un point d'entrée de procédure, soit un paramètre de la procédure, ou fonction procédure, en cours de composition à condition que ce paramètre ait été spécifié ENTRY (cf. § 7.3)

liste des arguments : c'est la liste classique des arguments de la procédure invoquée. Dans cette liste peuvent figurer des paramètres ou des variables locales du sous-programme en cours de composition, des variables locales d'un bloc BEGIN , appartenant au corps du sous programme, et à partir duquel on fait référence à la procédure *procname*, ou encore des variables universelles.

action prise par le système : semblable à celle d'une commande quelconque appartenant au corps du sous-programme. Lors de l'exécution effective de la procédure où figure cette commande, le système recher-

chera le *procname* effectif, puis provoquera l'activation de la procédure ainsi invoquée. Toute sauvegarde nécessaire est automatiquement effectuée.

réponse du système : identique à celle du paragraphe 5.6

7.7.. COMMANDE SCRATCH

Cette commande a toujours pour but d'effacer un sous-programme, mais de l'intérieur d'un autre sous-programme. L'effacement ne sera effectif que si le sous-prgramme sur lequel porte la commande n'est pas actif.

commande : SCRATCH␣*procname* ;

avec : *procname* soit l'identificateur effectif d'un point d'entrée de sous-programme, soit un paramètre du sous-programme en cours de composition, à condition que ce paramètre ait été spécifié ENTRY.

action prise par le système : lors de l'exécution du sous-programme auquel appartient cette commande, le système commence par rechercher le *procname* effectif, puis effacera si celà est possible, le sous-programme en question.

réponse du système : identique à celle du paragraphe 5.7

7.8.. SOUS-COMMANDE RETURN

Cette commande, utilisable uniquement dans le corps d'un sous-programme, a pour but de rendre le contrôle au superviseur qui déterminera quelle action doit être entreprise.

Selon que l'on se trouve dans une procédure ou une fonction-procédure, cette commande a deux syntaxes possibles.

7.8.A. D'UNE PROCEDURE

La commande est identique à l'instruction RETURN de PL/1 et peut être placée n'importe où à l'intérieur du corps de la procédure, elle en marque la fin logique. Il peut exister plusieurs commandes RETURN dans une même procédure. Si cette commande ne figure pas dans le corps de la procédure, la fin logique sera marquée soit par une commande GOTO portant sur une étiquette, paramètre ou non, renvoyant au programme appelant, soit par la pseudo-commande END, fin physique de la prodécure.

commande : RETURN ;

action prise par le système : pendant la phase d'exécution, le système désactive la procédure en cours et le superviseur rend le contrôle au programme appelant : soit à l'utilisateur si la procédure a été activée par la commande CALL élémentaire, soit à l'unité de programme où figurait la commande CALL.

réponse du système : selon le programme appelant, on obtient soit l'impression des caractères ===> , soit aucune réponse.

7.8.B. D'UNE FONCTION-PROCEDURE

La commande est identique à l'instruction RETURN(*expression*) de PL/1 et peut être placée n'importe où dans le corps de la fonction-procédure, elle en marque la fin logique. Il peut exister plusieurs commandes RETURN dans une même fonction-procédure. Toute fonction-procédure doit comporter au moins une commande RETURN.

commande : RETURN$_b$*expression* ;

avec : *expression* n'importe quelle expression arithmétique y compris des appels de fonction-procédures, autorisée dans une commande d'affectation (cf. § 5.1.B.)

action prise par le système : pendant la phase d'exécution le système désactive la fonction-procédure et rend le contrôle à la commande d'affectation qui en a provoqué l'activation.

réponse du système : aucune.

8. ACCOMPAGNATEUR OU PROGRAMME DE MISE AU POINT

L'accompagnateur a pour but de permettre à l'utilisateur de contrôler le déroulement d'une procédure ou d'une fonction-procédure.

Le programme de mise au point s'utilise en deux étapes. Une étape au cours de laquelle on appelle l'accompagnateur pour un sous-programme déterminé et déjà enregistré. Il sera alors possible de composer des "pré-requètes". La deuxième étape prendra effet lors de l'activation du sous-programme que l'on veut tester. Selon les pré-requètes émises il sera alors éventuellement possible de composer des requètes, dites requètes d'exécution.

On pourra, dans la suite de la session, soit modifier les pré-requètes du sous-programme "accompagné", soit enlever totalement les possibilités offertes par ces pré-requètes.

L'appel de l'accompagnateur se fait de deux façons selon que l'utilisateur désire composer des pré-requètes ou en modifier certaines, ou enlever les possibilités de l'accompagnateur.

Les pré-requètes permettront :

- de "tracer" les sous-programmes invoqués par le sous-programme accompagné,
- de transformer toute commande d'affectation sans impression en commande d'affectation avec impression,
- de poser des points d'arrêt qui provoqueront ultérieurement un arrêt momentané du déroulement du sous-programme accompagné et autoriseront alors l'utilisation des requètes d'exécution.

Ces requètes d'exécution permettent d'obtenir :

- l'impression du contenu d'une variable,

- la modification du contenu d'une variable,
- un test sur certaines variables,
- la modification de certaines options,
- la liste des variables affectées, des atomes, des pseudo-variables DIFF et FNC, des points d'entrée de sous-programmes; l'espace mémoire occupé,
- l'opérateur principal de l'expression affectée à une variable,
- le nombre d'arguments de l'expression affectée à une variable par rapport à son opérateur principal,
- la reprise de l'exécution du sous-programme.

Il est à noter que toutes les impressions relatives à l'accompagnateur se font sur le fichier CONSOLE.

8.1.. APPEL DE L'ACCOMPAGNATEUR

Cett appel se fait par l'intermédiaire de la commande OPTSET (cf. § 5.3.C.) A noter que l'on ne peut appeler l'accompagnateur qu'en dehors de toute unité de programme.

commande : OPTSET $\begin{matrix} \text{DEBUG} \\ \text{NODEBUG} \end{matrix}$; $\begin{matrix} \alpha) \\ \beta) \end{matrix}$

réponse du système : -> QUALIFY ENTRY-NAME :

—

l'utilisateur n'a plus qu'à composer l'identificateur du point d'entrée du sous-programme à qualifier, immédiatement aprés le caractère <_>

action prise par le système : - dans le cas où le sous-programme n'existe pas, se reporter au paragraphe 12.2.A.

- si le sous-programme existe, on obtient :

α) -> ENTER YOUR REQUEST :
_*pré-requète*

où *pré-requète*, composé par l'utilisateur, est défini au paragraphe 8.2

β) - si le sous-programme était accompagné :
★★ DEBUG FACILITY SCRATCHED ★★
et le système enlève toutes les pré-requètes existantes,

- si le sous-programme n'était pas accompagné : un message d'avertissement est imprimé (cf. 12.2.D.) et le contrôle est rendu à l'utilisateur.

8.2.. PRE-REQUETES DE L'ACCOMPAGNATEUR

Il y a sept pré-requètes qui détermineront l'action prise par l'accompagnateur pendant la phase d'exécution du sous-programme considéré.

Ces pré-requètes sont composées par l'utilisateur, au clavier du terminal, immédiatement aprés le caractère <_> imprimé en début de ligne, à raison d'une pré-requète par ligne.

8.2.A. TRACE DES SOUS-PROGRAMMES APPELES EN COURS D'EXECUTION

pré-requète : TRACE

action prise par le système : prévoir l'impression d'un message à l'"entrée" et à la "sortie" de tout sous-programme activée par le sous-programme que l'on a qualifié.

réponse du système : pendant la phase d'exécution du sous-programme qualifié, toute entrée dans un sous-programme est signalée par le message :

★★ ENTER *procname* ★★

et toute sortie par :

★★ LEAVE *procname* ★★

où *procname* est l'identificateur du sous-programme activé ou désactivé.

8.2.B. SUPPRIMER L'EFFET PRECEDENT

pré-requête : NOTRACE

action prise par le système : supprimer la trace des sous-programmes

réponse du système : aucune

8.2.C. IMPRESSION AUTOMATIQUE DE TOUTE AFFECTATION

pré-requête : PRINT

action prise par le système : prévoir l'impression du résultat de toute commande d'affectation sans impression, du sous-programme qualifié et de tout sous-programme invoqué par ce dernier, sauf contre indication. A noter que dans le cas d'une fonction-procédure, on obtient aussi l'impression du résultat de la partie *expression* de la commande RETURN.

réponse du système : pendant la phase d'exécution du sous-programme qualifié, réponse identique à celle du paragraphe 5.1.A.

8.2.D. SUPPRIMER L'EFFET PRECEDENT

pré-requête : NOPRINT

action prise par le système : revenir aux conditions standards en ce qui concerne les commandes d'affectation

réponse du système : aucune

Remarque : dès que l'utilisateur appelle l'accompagnateur pour un sous-

programme déterminé, le système met en disponibilité, d'une façon standard, les pré-requètes NOTRACE et NOPRINT

8.2.E. POSE DES POINTS D'ARRET

Il est possible, lors de l'exécution d'un sous-programme, de s'arrêter momentanément avant l'exécution d'une commande déterminée, si l'on a prévu un point d'arrêt sur cette commande.

pré-requète : BREAK$_b$AT$_b$xxx

où : xxx est un nombre décimal, de 1 à 999, indiquant le numéro d'ordre physique de la commande, avant l'exécution de laquelle l'utilisateur désire arrêter le déroulement du sous-programme.

action prise par le système : modification du code interne de la commande correspondante de façon à autoriser un arrêt lors de l'exécution du sous-programme.

réponse du système : pendant la phase d'exécution du sous-programme qualifié, le superviseur rendra, momentanément, le contrôle à l'utilisateur, juste avant d'exécuter la commande de numéro d'ordre physique xxx. Le clavier sera alors libéré et l'utilisateur pourra composer ses requètes d'exécution (cf. § 8.3)

8.2.F. ENLEVER DES POINTS D'ARRET

pré-requète : NOBREAK$_b$AT$_b$xxx

où : xxx a la même signification que précédemment

action prise par le système : rectification du code interne de la commande correspondante afin de supprimer l'arrêt.

réponse du système : aucune

Remarques : - lors de la composition des pré-requêtes BREAK et NOBREAK l'utilisateur veillera à la syntaxe : un seul caractère blanc entre BREAK, ou NOBREAK, et AT, entre AT et xxx (le programme de lecture est différent de celui utilisé pour les autres commandes),

- on ne peut poser de point d'arrêt sur la première commande d'un sous-programme (il s'agit de la commande comportant le mot clé PROCEDURE),

- les diverses commandes d'un sous-programme sont numérotées , dans l'ordre croissant et à partir de 1, de la première commande (celle comportant le mot clé PROCEDURE) à la dernière dans l'ordre physique de leur composition, chaque commande comptant pour 1. Il y a néanmoins deux exceptions :

    - il y a création, par le système pendant la phase de lecture, d'une "dummy commande" aprés la dernière commande de la partie THEN d'une commande conditionnelle IF,

    - les commandes COMMENT, LOCAL, ENTRY, FILE et LABEL n'entrent pas dans cette comptabilité du fait de leur traitement interne spécial (cf. 3° partie)

Exemple :

```
                                                    n° de commande
FACT : PROCEDURE(A,B,C) RECURSIVE ;                       1
       ENTRY A ;
       COMMENT CALCUL DE B! ;
       IF B=1 THEN                                        2
                  DO ;                                    3
                      C=1 ;                               4
                      RETURN ;                            5
                  END ;                                   6
       CALL A(A,B-1,C) ;                                  8
       C = C * B ;                                        9
END FACT ;                                               10
```

8.2.G. QUITTER L'ACCOMPAGNATEUR

Lorsque l'utilisateur a terminé de composer ses pré-requètes, il sort de l'environnement de l'accompagnateur en utilisant la

pré-requète : QUIT

action prise par le système : quitter l'environnement du programme de mise au point

réponse du système : retour, à haut niveau, du contrôle à l'utilisateur

8.3.. REQUETES D'EXECUTION

Nous venons de voir qu'il est possible de prévoir des arrêts lors de l'exécution d'un sous-programme. Lors de l'exécution de ce dernier chaque point d'arrêt provoquera le retour, provisoire, du contrôle à l'utilisateur. Ce dernier pourra en profiter pour effectuer des tests, voire modifier le contenu de certaines variables. Il ne pourra néanmoins pas modifier une commande ou provoquer un saut dans le déroulement de son programme.

Au moment où le système trouve un point d'arrêt, lors de l'exécution d'un sous-programme qualifié, le message suivant est imprimé :

-> ** BREAK POINT ** ENTER YOUR REQUEST :

_

l'utilisateur peut alors composer ses requètes, une par ligne et immédiatement aprés le caractère <_>. Les requètes d'exécution sont analogues à certaines commandes élémentaires. A noter, toutefois, qu'une requète ne peut excéder 125 caractères, blancs et <;> final compris, et qu'elle doit être entièrement composer sur une seule ligne.

Il y a huit requètes d'exécution permettant :

- de vérifier le contenu de certaines variables (arguments du sous programme, variables locales ou universelles),

- de changer le contenu de certaines variables,
- de modifier les options en cours,
- de lister les variables affectées, les atomes, les pseudo-variables DIFF et FNC, les points d'entrée de sous-programmes, l'espace mémoire occupé,
- de tester l'identité, au sens Formac, de deux variables ou de deux expressions,
- de connaitre l'opérateur principal de l'expression affectée à une variable déterminée,
- de connaitre le nombre d'arguments de l'expression affectée à une variable, par rapport à son opérateur principal,
- de relancer l'exécution du sous-programme.

L'action prise par le système, de même que la, ou les, réponse(s) fournie(s) sont identiques à celles des commandes élémentaires servant de support à ces requètes.

Liste des requètes	Réponse du système, voir
$*var* [, *var* ...] ;	§ 5.2
$*var* = *exp* [, ...] ;	§ 5.1.A.
$ DIFF / FNC (*var*) = *exp* [,...] ;	§ 5.1.A.
OPTSET$_b$*option* [, *option* ...] ;	§ 5.3.C.
LIST$_b$*argument* [, *argument* ...] ;	§ 5.3.D.
IDENT$_b$*exp1* ! *exp2* ;	§ 5.4.A.
EQUIV$_b$*exp1* ! *exp2* ;	§ 5.4.B.
LOP$_b$*var* [, *var* ...] ;	§ 5.4.C.
NARGS$_b$*var* [, *var* ...] ;	§ 5.4.D.
GO ;	aucune. Le système reprend l'exécution.

Remarques : - les pré-requètes ainsi que les requètes doivent être compo-

sées immédiatement aprés le caractère <_> imprimé en début de ligne et donnant le contrôle à l'utilisateur,

- la syntaxe des pré-requêtes décrites aux paragraphes 8.2.E. et 8.2.F. est trés stricte quant à la présence du caractère blanc,
- on consultera les paragraphes 11 et 12 pour tout message d'erreur relatif à cette section.

9. COMMANDES D'ANNULATION PENDANT LA PHASE DE LECTURE

L'utilisateur peut être amené à annuler tout, ou partie d'une commande pendant la phase de lecture. Les commandes dont il dispose, et décrites ci-aprés, sont soit des commandes de C.M.S. soit des commandes propres au système FORDECAL.

9.1.. ANNULATION D'UN CARACTERE @

Pour effacer un caractère d'une commande, il suffit de taper immédiatement aprés celui-ci le caractère <@>. Pour annuler une suite consécutive de n caractères, il faut taper n caractères <@> successifs.

Remarque : on ne peut pas effacer les derniers caractères d'une ligne une fois celle-ci envoyée.

9.2.. ANNULATION DE TOUT OU PARTIE D'UNE LIGNE ¢

Pour effacer le contenu d'une ligne depuis le premier caractère de celle-ci et avant d'envoyer la ligne, taper le caractère <¢>

9.3.. ANNULATION D'UNE COMMANDE %

Pour effacer une commande en cours de composition, taper le caractère <%> Tout ce qui est compris entre le dernier point virgule et le dernier caractère tapé est effacé, même si la commande est écrite sur plusieurs lignes.

9.4.. ANNULATION D'UNE UNITE DE PROGRAMME ERROR

A condition que l'exécution ne soit pas commencée, on peut annuler une unité de programme en cours de composition en tapant la

commande : ERROR ;

action prise par le système : le système efface toutes les commandes déjà stockées ainsi que les éventuels points d'entrée de sous-programme.

réponse du système : le message suivant est imprimé sur le fichier CONSOLE

** O.K. ,REPEAT YOUR COMMANDS **

et le contrôle est rendu, à haut niveau, à l'utilisateur.

10. COMMANDE D'ARRET D'EXECUTION D'UNE COMMANDE

L'utilisateur peut, à tout moment, stopper l'exécution d'une commande, mais cette interruption doit être exceptionnelle.

commande : - enfoncer la touche ATTENTION du terminal,
- taper les caractères <EX> (external interrupt)
- enfoncer la touche RETOUR CHARIOT

action prise par le système : le système arrête l'exécution de la commande en cours et se retrouve dans l'état où il se trouvait avant que cette commande ne soit stoppée. Dans le cas d'arrêt d'une commande appartenant à une unité de programme, l'exécution est stoppée au niveau le plus haut. Le système se charge, dans le cas d'un sous-programme ou d'une macro-commande de détruire toutes les variables locales et dans le cas d'une macro-commande d'effacer toutes les commandes stockées. A noter que les variables universelles ou les arguments, dans le cas d'un appel par nom, ont comme affectation la valeur courante qu'ils avaient juste avant l'interruption externe.

réponse du système : le message suivant est imprimé sur le fichier

CONSOLE

** EXECUTION STOPPED **

et le contrôle est rendu, à haut niveau, à l'utilisateur.

Remarques : - une fois le contrôle rendu, il est conseillé de vérifier l'état du système,

- il est possible que la commande ne soit pas prise en compte immédiatement (EX envoyé au cours d'une impression par exemple) il suffit de recommencer.

11. MESSAGES D'ERREURS PENDANT LA PHASE DE LECTURE-INTERPRETATION

Pendant cette phase le système vérifie, trés rapidement, la syntaxe des commandes reçues, en particulier la position relative des mots clés et l'orthographe de ceux-ci. Les messages d'erreurs relatifs à cette phase sont de la forme FDCxxxI suivi du message proprement dit. La partie TYPE "EXIT;" de certains messages signifie que l'utilisateur doit taper la commande EXIT; avant de reprendre sa composition, de façon à vider le buffer d'entrée. Toutes les commandes figurant physiquement entre la commande erronée et la commande EXIT sont alors ignorées.

Vingt cinq messages d'erreurs de syntaxe sont prévus. Ils sont tous imprimés sur le fichier CONSOLE. On en trouvera ci-aprés la liste ainsi que le type d'erreur correspondant.

FDC001I ** WARNING ** " [1] " NOT A PARAMETER : BYPASSED

- une commande de spécification ENTRY, LABEL ou FILE comporte un identificateur de variable ne figurant pas dans la liste des paramètres du sous-programme. Le système ignore cette spécification. On trouve en [1] l'identificateur de variable incriminé.

FDC002I ** WARNING ** " [1] " ALREADY SPECIFIED : BYPASSED

- une commande de spécification ENTRY, LABEL ou FILE comporte un identificateur de variable déjà spécifié. Cette redondance est ignorée. On trouve en [1] l'identificateur de variable incriminé.

FDC003I ** WARNING ** OPTION READER(CONSOLE) TURNED "ON"

- ce message apparait aprés l'impression d'un message de code FDCxxxI dans lequel figure TYPE "EXIT;" et dans le cas d'une lecture sur un fichier autre que CONSOLE. Le clavier se débloque et l'utilisateur peut continuer ainsi sa composition.

FDC004I YOUR RECORDED PROGRAM-UNIT WILL BE SCRATCHED

- ce message est imprimé aprés le message de code FDC003I lorsque l'erreur a été détectée pendant l'enregistrement d'une unité de programme. Toutes les commandes déjà stockées sont automatiquement effacées et il ne reste plus à l'utilisateur qu'à continuer sa composition au clavier ou à repasser sous CMS pour corriger son fichier d'entrée.

FDC005I ** WARNING ** IMPOSSIBLE DO-LOOP -> TYPE "EXIT;"

- la commande de groupe DO itératif placée au plus haut niveau n'est pas exécutable. L'utilisateur doit taper la commande EXIT pour vider le buffer d'entrée avant de reprendre la composition de sa commande.

FDC0006I ** SECOND DECLARATION OF " 1 "
DO YOU WANT TO SCRATCH THE OLDEST ? REPLY "YES" OR "NO"

_

- l'utilisateur a fait figurer dans sa déclaration de sous-programme un identificateur de *point d'entrée* déjà utilisé. On trouve en 1 cet identificateur. L'utilisateur a alors la possibilité soit d'effacer l'ancien sous-programme soit de le garder selon la réponse qu'il composera aprés le caractère <_> de la troisième ligne du message.
  - s'il répond YES le système va d'abord effacer l'ancien sous-programme avant de reprendre la lecture,
  - s'il répond NO le système commence par vérifier quel est le mode de lecture utilisé :
    - si la lecture se fait à partir d'un fichier, autre que le fichier CONSOLE, le message de code FDC0003I est imprimé suivi du message de code FDC0007I,
    - si la lecture se fait à partir du fichier CONSOLE, seul le message de code FDC0007I est imprimé.
  - s'il ne répond ni YES ni NO, ou que le mot clé n'est pas composé immédiatement aprés le caractère <_>, le système envoie le message de code FDC0009I

FDC0007I ** TYPE "EXIT;" AND RESTART FROM " 1 "

- dans le cas d'une deuxième utilisation d'un identificateur de *point d'entrée* le système, aprés avoir envoyé le message de code FDC0006I, imprime ce message dans le cas ou l'utilisateur ne veut pas effacer l'ancien sous-programme. En 1 figure l'identificateur de *point d'entrée* déjà utilisé, il suffit de reprendre la composition en remplaçant cet identificateur aprés avoir taper la commande EXIT; pour vider le buffer d'entrée.

FDC008I ** WARNING ** MISSING ENTRY NAME -> TYPE "EXIT;" AND RESTART

- ce message peut apparaitre aprés l'impression du message de code FDC007I dans le cas où aucun *point d'entrée* ne figure dans la commande de déclaration de sous-programme. L'utilisateur doit taper la commande EXIT; puis reprendre sa déclaration de sous-programme en veillant qu'au moins un identificateur de *point d'entrée* y figure.

FDC009I ** WARNING ** INCORRECT REPLY - "YES" OR "NO" ONLY ALLOWED

- l'utilisateur a mal répondu au message de code FDC006I. Il doit composer YES, ou NO, immédiatement aprés le caractère <_>

FDC010I ** WARNING ** END OF FILE " 1 "
BACK TO FILE " 2 "

- la fin du fichier d'identificateur 1 vient d'être rencontrée, le système continue sa lecture sur le fichier d'identificateur 2

FDC011I ** SYNTAX ERROR ** DUPLICATED LABEL PREFIX " 1 " -> TYPE "EXIT;"

- une commande est préfixée par une étiquette déjà rencontrée, dans le même bloc au même niveau, comme préfixe. On trouve en 1 l'identificateur d'étiquette incriminé. Il ne reste qu'à taper la commande EXIT; et reprendre la composition.

FDC012I ** SYNTAX ERROR ** NOT ALLOWED COMMAND -> TYPE "EXIT;"

- l'utilisateur a envoyé une commande non autorisée à cet endroit :
  - commande conditionnelle (IF, cf. § 6.4.) en dehors d'une unité de programme,
  - commande de déclaration de sous-programme (cf. § 7, 7.1, 7.2) à l'intérieur d'une unité de programme.
- Il ne reste plus qu'à taper la commande EXIT; et reprendre la composition.

FDC013I ** SYNTAX ERROR ** MISSING ENTRY NAME -> TYPE "EXIT;"

- il n'y a pas d'identificateur de *point d'entrée* dans la commande de déclaration de sous-programme (cf. § 7, 7.1, 7.2). Taper la commande EXIT; et reprendre la composition.

FDC014I ** SYNTAX ERROR ** MISSING "[1] " -> TYPE "EXIT;"

- l'utilisateur a omis le signe <=> ou le mot clé <TO> dans une commande de groupe DO itératif. On trouve en [1] le symbole absent. (cf. § 6.7) Taper la commande EXIT; et reprendre la composition.

FDC015I ** SYNTAX ERROR ** MISSING "THEN" -> TYPE "EXIT;"

- le mot clé <THEN> est omis dans une commande conditionnelle (cf. § 6.4) Taper la commande EXIT; et reprendre la composition.

FDC016I ** SYNTAX ERROR ** MISSING CALL/SCRATCH PROCNAME -> TYPE "EXIT;"

- la commande CALL, ou SCRATCH, ne comporte pas d'identificateur de procédure (cf. § 5.6, 5.7, 7.6, 7.7). Taper la commande EXIT; et reprendre la composition.

FDC017I ** SYNTAX ERROR ** NOT ALLOWED COMMAND : BYPASSED

- une commande de spécification ENTRY, LABEL ou FILE (cf. § 7.3, 7.4, 7.5), une commande de rupture de séquence (GOTO cf. § 6.3), une commande RETURN (cf. § 7.8), une pseudo-commande END ou une commande de déclaration de variables locales (LOCAL cf § 6.5) a été utilisée en déhors d'une unité de programme. Le mot clé ELSE a été utilisé sans avoir été précédé d'une commande conditionnelle (cf § 6.4). La commande est ignorée par le système.

FDC018I ** SYNTAX ERROR ** UNCOMPLETED COMMAND : BYPASSED

- toute commande ne comportant que le mot clé est ignorée par le système.

FDC019I ** SYNTAX ERROR ** INCORRECT DO COMMAND -> TYPE "EXIT;"

- lors de la composition d'un groupe DO itératif, une des valeurs *vi*, *vf* ou *ic* conduit à une évaluation incorrecte (cf. § 6.7) Taper la commande EXIT; et reprendre la composition.

FDC020I ** SYNTAX ERROR ** ILLEGAL CHARACTER - RETRY

_

- un identificateur de paramètre d'un sous-programme commence par le caractère <$>, L'utilisateur n'a qu'à retaper un identificateur immédiatement aprés le caractère <_>

FDC021I ** SYNTAX ERROR ** UNCOMPLETED RETURN COMMAND -> TYPE "EXIT;"

- la commande RETURN d'une fonction-procédure ne comporte pas de partie *expression* (cf. § 7.8.B). Taper la commande EXIT; et recommencer.

FDC022I ** SYNTAX ERROR ** NOT ALLOWED RETURN COMMAND : EXP BYPASSED

- la commande RETURN d'une procédure comporte une partie *expression*, celle-ci est ignorée (cf. § 7.8.A.)

FDC023I ** SYNTAX ERROR ** THERE IS NO RETURN COMMAND

- aucune commande RETURN *expression* ne figure dans une fonction-procédure (cf. § 7.8.B.). Ce message est suivi du message de code FDC004I (la fonction-procédure n'est pas enregistrée)

FDC024I ** ERROR ** WHILE READING ON FILE " 1 "

- une erreur s'est produite en cours de lecture sur le fichier d'identificateur 1.

FDC025I ** ERROR ** IMPLEMENTATION RESTRICTION - TOO LONG INPUT : BYPASSED

- la commande fait plus de 512 caractères, elle est ignorée.

12. MESSAGES D'ERREURS PENDANT LA PHASE D'EXECUTION

Il y a deux types de messages d'erreur pendant la phase d'exécution selon que l'erreur est détectée par le système FORDECAL ou par le système FORMAC

12.1.. ERREURS FORMAC

Les erreurs FORMAC sont celles détectées par les routines d'exécution de la bibliothèque FORMAC et excitant les conditions DENSYN, DENSYSM, DENSEM, DENSIZE ou DENERRR [2] [3]

Le système imprime alors, sur le fichier de sortie en activité et sur le fichier CONSOLE, le message correspondant au type de l'erreur. Les types d'erreurs et les messages s'y rattachant sont identiques à ceux de PL/1-FORMAC/OS. Aprés impression du message par le système FORMAC, le superviseur imprime le message suivant :

** EXECUTION FAILED **

précédé de l'identificateur du sous-programme où s'est produite l'erreur, le cas échéant. Le contrôle est ensuite rendu, à haut niveau, à l'utilisateur.

Si l'erreur s'est produite dans une unité de programme de type macro-commande, le système efface automatiquement l'unité de programme enregistrée; si l'erreur s'est produite dans un sous-programme, procédure ou fonction-procédure, le superviseur désactive ce dernier. (Les variables locales sont effacées mais les arguments du sous-programme, dans le cas d'un appel par valeur, ne sont pas modifiés alors que dans un appel par nom ils peuvent l'être.

12.2.. ERREURS FORDECAL

Les erreurs détectées pendant la phase d'exécution par le système FORDECAL, proviennent, en général, d'une syntaxe incorrecte qui n'a pû être repérée pendant la phase de lecture-interprétation.

Rappelons que le système FORDECAL ne possède qu'un analyseur syntaxique rudimentaire. Certaines commandes ne peuvent être décodées et interprétées que grâce à des mots clés judicieusement placés. Si ces mots clés sont absents, le superviseur se rend compte qu'il ne peut plus interpréter la commande d'où les erreurs détectées par le système FORDECAL. Il est possible d'autre part qu'une commande jugée syntaxiquement correcte par le superviseur contienne une erreur qui sera détectée par le système FORMAC.

Il y a trois catégories de messages :

- des messages signalant l'impossibilité du système à exécuter la commande, le superviseur rend alors le contrôle à l'unité de programme,
- des messages de mise en garde (WARNING) signalant l'impossibilité d'exécuter la commande par suite d'un mot clé erroné. Le système ignore cette partie de la commande et poursuit normalement l'exécution,
- des messages d'erreurs proprement dit (ERROR) signalant une erreur de syntaxe telle que le superviseur ne peut exécuter la commande. L'exécution de l'unité de programme est alors abandonnée et le contrôle rendu, à haut niveau, à l'utilisateur aprés impression du message d'erreur suivi de :

  ** EXECUTION FAILED **

  Cette dernière impression est éventuellement précédée de l'identificateur du sous-programme où a été détectée l'erreur. Dans le cas d'une macro-commande le système se charge aussi d'effacer cette unité de programme de la mémoire.

Signalons enfin que les messages d'erreurs du système FORDECAL, émis pendant la phase d'exécution sont de la forme FDCxxxG suivi du message proprement dit, ou de FDCxxxD suivi du message. Ce dernier cas se rapporte à des erreurs concernant l'accompagnateur.

12.2.A. IMPOSSIBILITE DE SATISFAIRE LA COMMANDE

FDC001G ** WARNING ** UNKNOWN PROCEDURE " [1] "

- l'utilisateur invoque la procédure dont l'identificateur de *point d'entrée* figure en [1], soit dans une commande CALL soit dans une commande SCRATCH, et cet identificateur n'appartient pas à la table des points d'entrée.

FDC002G ** WARNING ** PROCEDURE " [1] " IS ACTIVE - COULD NOT BE SCRATCHED

- le système ne peut excécuter la commande SCRATCH portant sur la procédure, ou fonction-procédure, d'identificateur [1] car cette dernière est active.

FDC003G ** ERROR ** WHILE REWINDING FILE " [1] "

- une erreur a été détectée par CMS lors de l'exécution de la commande REWIND ou PRINTF portant sur le fichier d'identificateur [1]

FDC004G ** ERROR ** WHILE ERASING FILE " [1] "

- une erreur a été détectée par CMS lors de l'exécution de la commande ERASE portant sur le fichier d'identificateur [1]

FDC005G ** ERROR ** WHILE PRINTING ON FILE " [1] "

- une erreur a été détectée par CMS lors de l'exécution de la commande PRINTER(*nom de fichier*), on trouve en [1] l'identificateur du fichier.

FDC006G ** ERROR ** WHILE COPYING ON FILE " [1] "

- une erreur a été détectée par CMS lors de l'exécution de la commande COPY sur le fichier d'identificateur [1].

12.2.B. MESSAGES DE MISE EN GARDE

FDC007G ** WARNING ** INEXECUTABLE DO-LOOP

- une des quantités *vi*, *vf* et/ou *ic* n'a pas de valeur numérique, il est donc impossible d'exécuter la commande DO itérative. Le groupe est ignoré.

FDC008G ** WARNING ** UNBALANCED PARENTHESIS IN : [1]

- le compte parenthèses d'une commande est faux. Le système rajoute une parenthèse droite mais sans garantir que la commande est alors syntaxiquement correcte. On trouve en [1] la commande en question.

FDC009G ** WARNING ** ILLEGAL LIST ARGUMENT : [1]

- un argument inconnu figure dans une commande LIST. Le système imprime en [1] cet argument et continue l'exécution en ignorant cette partie de la commande.

FDC010G ** WARNING ** ILLEGAL OPTSET OPTION : [1]

- une option inconnue figure dans une commande OPTSET. Le système imprime en [1] cette option et continue l'exécution en ignorant cette partie de la commande.

FDC011G ** WARNING ** INCORRECT FILE OPTION (FILENAME MISSING) : BYPASSED

- dans une commande OPTSET d'option COPY, ERASE, PRINTF ou REWIND le *nom de fichier* est omis (cf. § 5.3.C.) Cette option est ignorée par le système.

FDC012G ** WARNING ** NOT ALLOWED FILE OPTION : BYPASSED

- dans une commande OPTSET d'option READER le *nom de fichier* est identique au nom de fichier de l'option COPY ou PRINTER en activité, ou réciproquement. Cette option est ignorée par le système.

FDC013G ** WARNING ** FILE " [1] " NOT FOUND : BYPASSED

- le fichier d'identificateur [1] sur lequel doit se faire la lecture n'existe pas.

FDC014G ** WARNING ** REMOB COMMAND NOT ALLOWED : BYPASSED

- l'utilisateur essaie de remober soit des variables locales soit des paramètres de sous-programme.

12.2.C. ERREURS FATALES

FDC015G ** ERROR ** PROCEDURE " [1] " IS NOT RECURSIVE

- l'utilisateur invoque récursivement le sous-programme dont l'identificateur de point d'entrée figure en [1] alors qu'il n'a pas été spécifié récursif lors de sa déclaration.

FDC016G ** IMPLEMENTATION RESTRICTION OCCURS IN PROCEDURE " [1] "

- le sous-programme d'identificateur [1] a été appelé récursivement 32768 fois déjà.

FDC017G ** ERROR ** "SY" UNKNOWN SYMBOL

- l'utilisateur a fait figurer dans la partie *test* d'une commande conditionnelle (IF, cf. § 6.4) un opérateur de comparaison inconnu.

FDC018G ** ERROR ** NOT ARITHMETIC TEST

- l'utilisateur n'a pas respecté la sémantique lors de l'écriture de la partie *test* d'une commande conditionnelle (cf. § 6.4) : utilisation d'un opérateur de comparaison arithmétique alors qu'un des primaires n'a pas une valeur arithmétique.

FDC019G ** ERROR ** INCORRECT EQUIV/IDENT WRITING

- le caractère <!> n'apparait pas dans la partie argument de EQUIV ou de IDENT pour séparer les deux expressions à comparer (cf. § 5.4.)

FDC020G ** ERROR ** WRONG NUMBER OF ARGUMENTS IN PROCEDURE " 1 "

- lors de l'activation du sous-programme d'identificateur 1, le nombre d'arguments est différent de celui des paramètres.

FDC021G ** ERROR ** WRONG DATA CHARACTERISTIC IN PROCEDURE " 1 "

- lors de l'activation du sous-programme d'identificateur 1, la spécification d'un argument est différente de celle du paramètre correspondant.

FDC022G ** ERROR ** PARAMETER " 1 " HAS NOT BEEN SPECIFIED AS ENTRY-NAME

- lors de l'activation d'une procédure par la commande CALL, le *procname* 1, bien que paramètre du sous-programme activant, n'a pas été spécifié ENTRY.

FDC023G ** ERROR ** PARAMETER " 1 " HAS NOT BEEN SPECIFIED AS FILENAME

- la partie *nom de fichier* d'une commande GET (cf. § 5.5) ou d'une commande OPTSET portant sur des opérations d'Entrée/Sortie ou de manipulations de fichier (cf. § 5.3.C.) correspond à un paramètre, d'identificateur 1, du sous-programme appelant mais la spécification de ce paramètre est différente de FILE.

FDC024G ** ERROR ** PARAMETER " 1 " HAS NOT BEEN SPECIFIED AS LABEL

- lors de l'exécution d'une commande GOTO, la partie étiquette correspond à un paramètre, d'identificateur 1, qui a une spécification différente de celle de LABEL.

FDC025G ** ERROR ** UNRESOLVED GOTO " 1 " IN BLOCK " 2 "

- il est impossible de résoudre la référence figurant en 1 d'une commande GOTO appartenant au bloc de numéro 2.

FDC026G ** ERROR ** INPUT DATA FAILED

- lors de l'exécution d'une commande GET, la fin du fichier-données est rencontrée avant que la liste des variables à charger ne soit épuisée.

12.2.D. ERREURS DETECTEES LORS DE L'UTILISATION DE L'ACCOMPAGNATEUR

FDC001D ** WARNING ** OPTION DEBUG/NODEBUG NOT ALLOWED : BYPASSED

- l'utilisateur a appelé l'accompagnateur de l'intérieur d'une unité de programme. La commande est ignorée (cf. § 8.1)

FDC002D ** WARNING ** UNKNOWN PROCEDURE " [1] "
REPLY "QUIT" OR "AGAIN"

—

- l'utilisateur a qualifié une procédure, ou une fonction-procédure, d'identificateur [1], qui n'appartient pas à la table des points d'entrée (cf. § 8.1.), il peut se faire aussi que l'utilisateur ait mal positionné l'identificateur de point d'entrée lors de sa réponse au message -> QUALIFY ENTRY-NAME. Il peut alors soit taper QUIT pour sortir de l'environnement accompagnateur, soit taper AGAIN pour recomposer un identificateur de point d'entrée.

FDC003D ** WARNING ** INCORRECT REPLY - "QUIT" OR "AGAIN" ONLY ALLOWED

—

- soit l'utilisateur n'a pas répondu QUIT ou AGAIN, au message de code FDC002D, soit la réponse est mal cadrée.

FDC004D ** WARNING ** THIS PROCEDURE HAS NO DEBUG FACILITY

- l'utilisateur essaie d'enlever les facilités prévues par l'accompagnateur à une procédure non accompagnée. L'option NODEBUG est ignorée.

FDC005D ** WARNING ** ILLEGAL DEBUG REQUEST : [1]

- l'utilisateur a composé une pré-requête inconnue, cette dernière est ignorée. On trouve en [1] la pré-requête incriminée .(cf. § 8.2.)

FDC006D ** WARNING ** NOT ALLOWED BREAK-POINT

- l'utilisateur essaie de poser un point d'arrêt sur la première commande d'un sous-programme, la pré-requête est ignorée. (cf. § 8.2.E.)

FDC007D ** WARNING ** BREAK-POINT OUT OF RANGE

- l'utilisateur essaie de poser un point d'arrêt sur une commande dont le numéro dépasse le nombre de commandes du sous-programme. La pré-requête est ignorée.

FDC008D ** WARNING ** BREAK-POINT ALREADY EXISTS

- l'utilisateur essaie de poser un point d'arrêt sur une commande en possédant déjà un. La pré-requête est ignorée.

FDC009D ** WARNING ** BREAK-POINT OUT OF RANGE OR INEXISTENT

- l'utilisateur essaie d'enlever un point d'arrêt sur une commande qui n'en possédait pas ou dont le numéro d'ordre est supérieur au nombre de commandes du sous-programme. La pré-requête est ignorée.

FDC010D ** WARNING ** NOT ALLOWED REQUEST

- l'utilisateur a composé, lors de l'exécution d'un point d'arrêt, une requête inconnue qui est ignorée par le système (cf. § 8.3.)

FDC011D ** WARNING ** TOO LONG REQUEST OR SEMI-COLON MISSING

—

- une requête d'exécution dépasse 125 caractères ou n'est pas terminée par un point virgule (cf. § 8.3.)

Remarques : - aprés l'impression des messages de code FDC002D, FDC003D et FDC011D le système envoie, en début de ligne, le caractère <_>. L'utilisateur doit composer sa réponse immédiatement aprés ce caractère,

- de même, le système envoie aprés l'impression des messages de code FDC005D, FDC006D, FDC007D, FDC008D, FDC009D et FDC010D une ligne blanche suivie d'une ligne commençant par le caractère <_>, l'utilisateur doit composer sa réponse immédiatement aprés ce caractère.

13. CONDITIONS EXCEPTIONNELLES

Il y a trois types de conditions exceptionnelles :

- si pendant une session la liste libre est épuisée. Dans ce cas le système FORMAC prend en compte l'"erreur" et provoque l'impression suivante :

  DEN084I - DENTC5 - DENSIZE
  NO MORE FREE LIST AVAILABLE

  Le contrôle est néanmoins rendu, à haut niveau, à l'utilisateur. Ce dernier a alors trois possibilités :

  - récupérer de la mémoire en utilisant les commandes ATOMIZE, REMOB ou SCRATCH,
  - revoir la manière de traiter le problème,
  - travailler avec un espace mémoire plus important;

- si pendant la phase d'exécution des routines FORMAC, se produit une erreur prise en compte par PL/1. On entre alors généralement sous DEBUG. Il ne reste plus qu'à rappeler C.M.S., FORDECAL et recommencer le travail. Ce genre d'erreurs est trés rare et se produit, en général, aprés de nombreuses erreurs FORMAC entrainant l'erreur

  FORMAC SYSTEM ERROR

  PL/1 et FORMAC n'ayant pû effectuer toutes les sauvegardes nécessaires.

- si une erreur existe dans le superviseur du système FORDECAL. Ce type d'erreur ne devrait, en principe, pas exister. Nous remercions par avance l'utilisateur qui en découvrirait.

14. APPENDICE

Deux fonctions prédéfinies ont été incorporées au système : ABS et EXP$. Elles sont définies comme pseudo-variables FNC et ne doivent pas, en conséquence, provoquer des conflits avec des identificateurs créés par l'utilisateur.

- ABS(*exp*) : donne la valeur absolue de *exp* dans le cas d'une constante, sinon les combinaisons correspondantes de la fonction STEP,
- EXP$(*exp*) : si *exp* est une constante c, alors EXP$ donne la valeur numérique, en flottant, de e^c.

15. EXEMPLES

Le lecteur trouvera ci-aprés quelques exemples. Les commandes correspondent aux lignes composées en caractères minuscules, les réponses du système sont en majuscules.

15.1.. COMMANDES ELEMENTAIRES

```
$ a = x ** 2 + y ** 2 , b = lop a ; $  c = nargs a ;

A = X^2 + Y^2

B = 24
===>

C = 2
===>
```

```
ident a ! x ** 2 - y ** 2 ;

NO
===>
equiv x ** 2 + 2 * x * u + u ** 2 ! (x + u) ** 2 ;

YES
===>
list vars , atoms ;

ASSIGNED VARIABLES LIST :
A,B,C

ATOMS LIST :
U,X,Y
===>
remob a , c ; list vars , atoms ;
===>
ASSIGNED VARIABLES LIST :
B

ATOMS LIST :
U,X,Y
===>
```

15.2.. COMMANDE DO

```
do;
   comment exemple de groupe do;
   do i=1 to 1;
      $i ;
      do j=4.0 to 3.0 by -1 ; $ j ; end ;
      do k=3/4 to 1/5 by -1/8 ;
         $k;
      end;
   end;
end;
```

```
I = 1
J = 4.0
J = 3.0
K = 3/4
K = 5/8
K = 1/2
K = 3/8
K = 1/4
===>
```

15.3.. COMMANDE BEGIN

```
$gd = x ** 3 - 3 * x ** 2 + 2 ;     $pd = - 3 * x ** 2 + 5 * x - 2 ;
```

GD = $-3X^2 + X^3 + 2$

```
===>
```

PD = $5X - 3X^2 - 2$

```
===>
begin;
      comment calcul du pgcd de deux polynomes;
      local a , b , b0;
      a = highpow(gd,x);
e3 :  b = highpow(pd,x); b0 = coeff(pd,x**b);
e1 :  gd= expand(gd - pd*coeff(gd,x**a) / b0 * x ** (a-b));
      a = highpow(gd,x);
      if a >= b then goto e1;
      if a =  0 then goto e2;
      b = gd, gd = pd, pd = b; goto e3;
e2 :  if lop gd = 36 then do;
                           pgcd = expand(pd/abs(b0));
                           goto fin;
                        end;
      pgcd = 1;
fin:  put le p.g.c.d. est ; $ pgcd;
end;
```

```
LE P.G.C.D. EST

PGCD =  - X + 1
===>
```

15.4.. COMMANDE PROCEDURE

```
facto : procedure (a , b) recursive;
        if a=1 then do;
                      b=1;
                      return;
                    end;
        call facto(a-1,b);
        b=b*a;
end facto;

** PROCEDURE "FACTO    " RECORDED **
===>
call facto(2 + 1 , result ) ;
===>
$result ;

RESULT =  6
===>
```

15.5.. COMMANDE FONCTION-PROCEDURE

```
fact : procedure(a) recursive returns;
       if a=1 then return(1);
       return(a * fact(a-1));
end fact;

** PROCEDURE "FACT     " RECORDED **
===>
```

```
a1 = 3 , b1 = 12 ;  $t1 = fact(a1 + 1) / fact(5) + 12 + b1 ;

===>

T1 = 121/5
===>
```

15.6.. ETIQUETTES ET GOTO

```
p1 : procedure(a);
     label a;
     put entree p1;
     do;
        put appel p2;
        call p2(a);
        put retour p2;
        a : put etiquette interne au bloc;
     end;
     put exit procedurep1;
     goto a;
     put apres gotoa:
end p1;

** PROCEDURE "P1      " RECORDED **
===>
p2 : procedure(a);
     label a;
     put entree p2;
     goto a;
     put sortie p2;
end p2;

** PROCEDURE "P2      " RECORDED **
===>
```

```
p3 : procedure(a);
     label a;
     put entree p3;
     do;
        put appel p2 dans p3;
        call p2(a);
        put retour p2 de p3;
     end;
     put sortie p3;
end p3;

★★ PROCEDURE "P3      " RECORDED ★★
===>
do;
   put appel p1;
   call p1(toto);
   put apres appel p1;
   toto : put étiquette toto;
   put appel p3;
   call p3(titi);
   put apres appel p3;
   titi : put etiquette titi;
end;

APPEL P1
ENTREE P1
APPEL P2
ENTREE P2
ETIQUETTE INTERNE AU BLOC
EXIT PROCEDUREP1
ETIQUETTE TOTO
APPEL P3
ENTREE P3
APPEL P2 DANS P3
ENTREE P2
ETIQUETTE TITI
===>
```

15.7.. ACCOMPAGNATEUR

```
optset debug;

-> QUALIFY ENTRY-NAME :
_    facto

FDC002D ** WARNING ** UNKNOWN PROCEDURE "    FACT"
                      REPLY "QUIT" OR "AGAIN"
_againe

FDC003D ** WARNING ** INCORRECT REPLY - "QUIT" OR "AGAIN" ONLY ALLOWED
_again

_facto

-> ENTER YOUR REQUEST :
_trace
_print
_quit
===>
call facto(3, resul);

** ENTER FACTO    **
** ENTER FACTO    **
RESUL = 1
** LEAVE FACTO    **
RESUL = 2
** LEAVE FACTO    **
===>
```

- - - - ☆ - - - -

BIBLIOGRAPHIE

[1] IBM System/360 - Operating System
PL/1 (F) Language Reference Manual. GC.28-8201-3

[2] A. LAPLACE - PL/1-FORMAC - tome 1 , le langage
IRMA GRENOBLE 1972

[3] R. TOBEY et al. - PL/1-FORMAC Interpreter - User's Reference Manual
IBM Contributed Program Library 360.D.03.3.004

- - - - ◆ - - - -

TROISIEME PARTIE

F O R D E C A L

LE SYSTEME

Des wagons sur une voie de garage,
Tout un programme !

1. INTRODUCTION

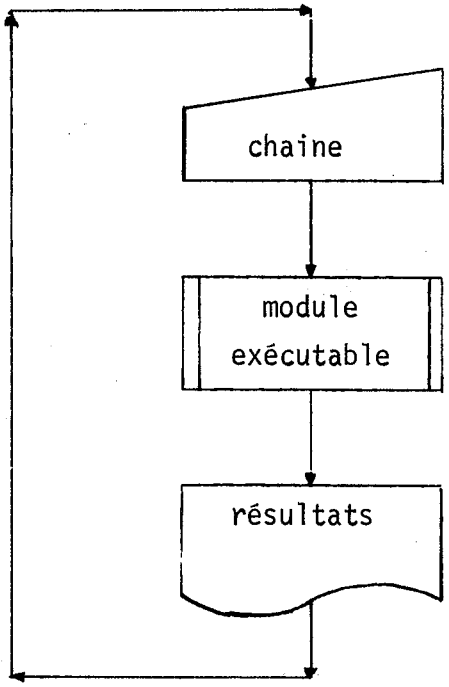
Un système interactif rudimentaire consisterait en un petit programme PL/1 qui se contenterait de lire une chaine alphanumérique de la forme :

identificateur de variable = expression

de la confier ensuite à la routine Formac DENFMC2 et bouclerait ainsi indéfiniment.

Remarque : la routine DENFMC2, qui fait partie de la bibliothèque d'exécution du système FORMAC, correspond à l'instruction PL/1-FORMAC PRINT_OUT, qui est l'instruction d'impression de Formac.

Ainsi que nous l'avons vu dans le Manuel Utilisateur (2° partie), nous avons considérablement développé un tel système en y introduisant toutes les "instructions", ou presque, de FORMAC/OS sous forme d'appel aux sous-programmes correspondants. Le système se présente alors sous la forme d'un module exécutable dont l'organigramme d'utilisation peut se représenter sous la forme suivante :



Nous avons tout d'abord été amené à prévoir un jeu de commandes (cf. 2° partie), dont la syntaxe est trés proche de celle de PL/1 et dont l'écriture rappelle celle

utilisée en mathématique. De là nous sommes passés, afin d'accroitre les possibilités du système, à la notion d'"unité de programme" avec l'introduction des blocs et des sous-programmes.

Rappelons que toutes les variables utilisées au cours d'une session de FORDECAL sont de type FORMAC.

Avant d'entamer la description interne du système, il nous parait nécessaire, bien que supposant le lecteur suffisamment averti du langage PL/1, de rappeler quelques particularités de ce langage que nous allons utiliser par la suite.

2. POSSIBILITES OFFERTES PAR PL/1 EN VUE DU TRAITEMENT DES LISTES

Parmi l'ensemble des attributs des variables PL/1 il en existe trois particulièrement intéressantes pour le traitement des listes : les variables d'attribut CONTROLLED (contrôlées), POINTER (pointeur), BASED (basées) et les variables structures.

Une variable d'attribut CONTROLLED est allouable dynamiquement par l'instruction ALLOCATE et libérable ensuite, dynamiquement, par l'instruction FREE. Il peut exister, à un moment donné, plusieurs générations d'une même variable contrôlée, par contre on ne peut accéder, à un moment donné, qu'à la dernière génération de cette variable. Une variable contrôlée a, de plus, un type déterminé : numérique, chaine de caractères, ...

Une variable d'attribut POINTER correspond à une zone de mémoire, de longueur 4 octets, qui contiendra une adresse absolue. Les pointeurs sont, entre autre, associés à des variables basées.

Une variable basée ressemble à une variable contrôlée en ce qui concerne l'allocation et la libération dynamique de la zone de mémoire attribuée à cette variable, mais elle implique la notion de pointeur. A chaque variable basée est associée une variable pointeur qui contient l'adresse absolue de la variable basée. Il est ainsi possible d'accéder, à un même moment, à plusieurs "générations" d'une même variable basée.

Une variable de type structure consiste en un complexe de variables rangées à des niveaux différents. Les variables appartenant aux niveaux inférieurs peuvent être de types différents, elles seront néanmoins regroupées sous le même vocable : l'identificateur de variable figurant au niveau 1. Ainsi la déclaration :

```
1 A,
 2 B FIXED BINARY(31,0),
 2 C CHARACTER(4) ;
```

regroupe sous le vocable A, une variable d'identificateur B de type numérique et une variable C de type caractère. On pourra ensuite accéder à B ou à C en utilisant, respectivement, l'identificateur A.B ou A.C

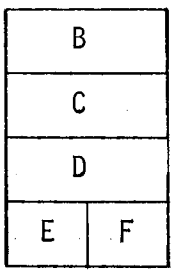
Nota : on pourra consulter [1] [2] [3] pour de plus amples renseignements.

Une structure peut être contrôlée ou basée. Pour des raisons propres à PL/1 toute allocation de mémoire relative à une structure basée est arrondie au multiple de 8 immédiatement supérieur ou égal au nombre d'octets figurant dans la déclaration. On a donc intérêt, pour éviter une génération de code inutile pendant la compilation, à utiliser des structures basées dont la "longueur" en octets sera un multiple de 8. En fait nous avons trés peu utilisé les structures contrôlées car l'allocation mémoire correspondante est énorme. Ainsi, à la déclaration :

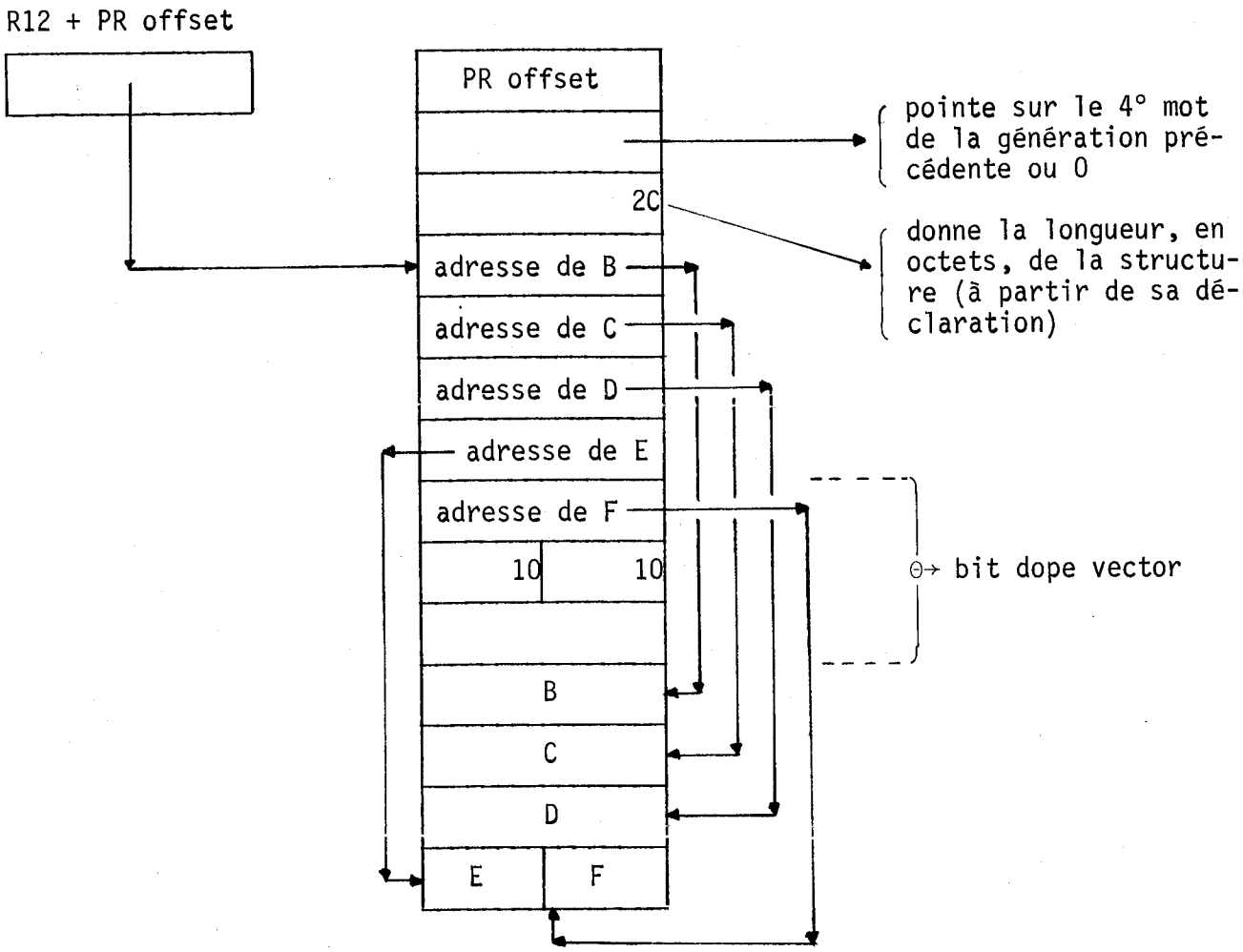
```
1 A  CONTROLLED,
 2 B POINTER,
 2 C POINTER,
 2 D POINTER,
 2 E FIXED BINARY(15),
 2 F BIT(16) ;
```

représentable par :

A :



qui semble n'occuper que 16 octets, correspond en fait :



La zone de mémoire effectivement allouée à une structure contrôlée est donc de 12 octets + les constituants de la structure (pointeur sur élément et les éventuels dope vectors) [4]. Sur cet exemple nous constatons qu'à une structure contrôlée de 16 octets correspond une zone de mémoire de 56 octets, ce qui est exhorbitant.

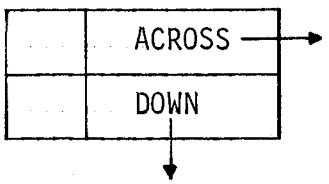
Il est possible en PL/1 d'utiliser des étiquettes, mais à chaque étiquette correspond une zone de mémoire de 8 octets, contenant l'adresse de l'étiquette ainsi que le compte d'invocation associé à la notion de bloc. Or nous avions besoin d'énormément d'étiquettes dans le superviseur de FORDECAL afin de pouvoir se brancher sur des séquences de traitement particulières. D'autre part, la partie "utile" d'un pointeur est rangée sur les 3 derniers octets d'un mot, il était donc possible d'utiliser le premier octet comme indice de tableau, à condition que cet indice ne dépasse pas 255.

Celà se ferait trés simplement en langage d'assemblage mais nécessite en PL/1 une petite "astuce".

Avant de décrire cette astuce, parlons des variables DEFINED. Il est possible de "loger" deux variables à la même adresse, en mémoire, en "définissant" la deuxième sur la première et à condition qu'elles soient de même type et aient la même longueur. En fait il est possible de définir une variable d'identificateur B, de type t_1 et de longueur l_1, sur une variable d'identificateur A, de type t_2 et de longueur l_2, à condition que $l_1 \leq l_2$. Le compilateur signalera une erreur de code 8 mais tout se passe trés bien à l'exécution. On peut aussi définir soit des variables simples soit des structures.

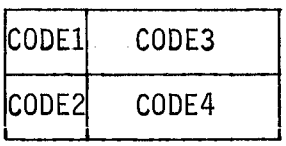
Nous verrons, plus loin, que le type de cellule utilisé pour chainer les commandes (les noeuds) est constitué par deux pointeurs, mais pour pouvoir ensuite traiter la commande le superviseur a besoin de connaitre le type de commande à exécuter. Nous avons donc utilisé le premier octet de chaque mot pour placer le code correspondant à la commande sous forme d'un nombre de type BIT(8). Pour réaliser ceci nous avons défini deux structures :

```
1 PLX1,
 2 ACROSS POINTER,
 2 DOWN   POINTER;
```



et

```
1 PLX2 DEFINED PLX1,
 2 CODE1 BIT(8),
 2 CODE3 BIT(24),
 2 CODE2 BIT(8),
 2 CODE4 BIT(24);
```



et du fait du DEFINED, la partie CODE1 se superpose au premier octet du premier mot de PLX1 et la partie CODE2 se superpose au premier octet du deuxième mot de PLX1. Les parties CODE3 et CODE4 de PLX2 ne seront pratiquement pas utilisées, sauf pour simuler un pointeur NULL. Un pointeur NULL se représente par : FF000000 or en chargeant le premier octet par un code nous n'avons plus la possibilité de tes-

ter le contenu d'un pointeur à NULL. Un pointeur NULL d'un noeud sera néanmoins constitué par le nombre 000000 rangé dans les trois derniers octets du mot, mais le test se fera d'une autre façon.

Pour charger un code et une adresse dans, par exemple, le premier mot de PLX1 nous procédons de la manière suivante : (attention, il faut respecter l'ordre)

```
PLX2.CODE2  = valeur, en BIT, du code
PLX1.ACROSS = adresse
```

Pour décharger un code et une adresse

```
INDTAB   = PLX2.CODE2
pointeur = PLX1.ACROSS
```

la variable INDTAB est de type BIT(8) définie sur le dernier octet de la variable ZEICHEN2 de type FIXED BINARY(15). Il est donc ensuite possible d'utiliser la variable d'identificateur ZEICHEN2 comme indice de tableau sans aucune conversion.

Dans un autre ordre d'idée, nous utiliserons des variables structures de longueur variable pour ranger la partie intéressante de certaines commandes. Nous utiliserons pour celà l'attribut REFER [2]

Soit la déclaration suivante :

```
1 A  BASED(P),
 2 B FIXED BINARY(31),
 2 C FIXED BINARY(31),
 2 D CHARACTER(I REFER(C)) ;
```

On commence par charger la variable d'identificateur I, de type FIXED BINARY(31), par la longueur de la chaîne de caractères à ranger en D, avec alignement. On alloue ensuite la variable structure d'identificateur A dont la partie D aura la longueur désirée. A noter que cette longueur se retrouve en C, ce qui sera aussi utilisé par la suite.

3. LE SYSTEME FORDECAL

3.1.. GENERALITES

Ce système consiste en un interpréteur-superviseur, écrit en PL/1, et en une bibliothèque de programmes d'exécution constituée par la bibliothèque PL/1 et la bibliothèque FORMAC. Cette dernière comprend les modules de la bibliothèque FORMAC/OS, modifiés ou non, et des modules d'Entrée/Sortie, de manipulation de fichier. Ces derniers modules ont été écrits en tenant compte du système C.M.S. sous lequel fonctionne FORDECAL.

Nous avons vu que l'utilisateur, une fois placé dans l'environnement de FORDECAL, communique avec le système par le jeu des commandes prévues (cf. 2° partie). Ces commandes passent par deux, ou trois, phases :

- une phase de lecture-interprétation,
- une phase d'exécution,
- éventuellement une phase d'effacement.

Les commandes sont, d'autre part, classées en trois classes :

- les commandes à exécution immédiate,
- les commandes à exécution semi-différée,
- les commandes à exécution différée.

Les actions prises par le système pendant la phase de lecture peuvent être trés différentes selon la classe à laquelle appartient la commande.

Signalons qu'à l'initialisation du système, avant de donner le contrôle à l'utilisateur, les routines GETINIT et STOPEX sont activées.

- GETINIT est chargé de créér une zone commune entre le superviseur, écrit en PL/1, et certaines routines, écrites en code;
- STOPEX initialise le programme qui prendra en compte la commande d'arrêt d'exécution d'une commande (cf. 2° partie, § 10)

Il est à noter aussi que la routine DENCCSO, de la bibliothèque FORMAC,

a été modifiée de sorte que les pseudo-variables FNC d'identificateurs ABS et EXP$ fassent partie du système FORMAC.

3.2.. LES PHASES DU SYSTEME

Dans le cas d'une commande à exécution immédiate, par exemple :

```
$ A = FAC(10) ;
```

commande qui a pour but de provoquer le calcul de 10!, d'affecter le résultat à la variable d'identificateur A et d'imprimer le résultat, selon le format standard d'impression de PL/1-FORMAC, la phase de lecture-interprétation se réduit à :

- récupérer la chaine d'entrée '$ A = FAC(10) '
- se rendre compte de la présence du caractère <$> en première position de la chaine,
- se brancher sur la séquence d'exécution correspondant à une commande d'impression.

La phase d'exécution consiste alors à exécuter la séquence précédente. Il s'agit en fait d'un appel à la routine Formac DENFMC2, avec comme argument la chaine 'A = FAC(10)'. La routine DENFMC2, qui appartient à la bibliothèque classique de Formac, provoquera l'interprétation de l'instruction d'affectation A = FAC(10) : création éventuelle du rep-word A, calcul de FAC(10), affectation du résultat à A, impression du résultat.

Le contrôle est ensuite rendu, à haut niveau, à l'utilisateur afin de lui permettre de continuer sa session.

Dans le cas d'une macro-commande (commande DO non itérative, DO itérative, bloc BEGIN), le système doit, pendant la phase de lecture, en plus de l'interprétation des diverses commandes internes au groupe, pouvoir les ranger quelque part en mémoire, en chainant ces diverses commandes entre elles dans un ordre logique de façon à pouvoir les exécuter dès que le bloc de commandes constituant l'unité de programme sera physiquement enregistrée.

Ainsi :

```
DO I=1 TO 10;
   $ A(I) = FAC(I) ;
END ;
```

Lors du décodage de la première commande, pendant la phase de lecture-interprétation, le système se rend compte qu'il s'agit d'une macro-commande du type DO itératif. Aprés certaines initialisations, que nous verrons plus loin, le système va allouer des zones de mémoire afin d'y ranger ces 3 commandes. Lors du décodage de la pseudo-commande END, qui marque dans l'exemple ci-dessus la fin physique de l'unité de programme, la phase de lecture se termine.

La phase d'exécution commence alors. Cette phase est dirigée par le superviseur qui se charge de déterminer le genre de la commande à exécuter et active la séquence correspondante aprés une éventuelle nouvelle interprétation. Lors de l'exécution de la pseudo-commande END, la boucle étant épuisée, le superviseur confiera le contrôle à la phase d'effacement qui se chargera de libérer l'espace mémoire occupé par ce bloc. Le contrôle est ensuite rendu, à haut niveau, à l'utilisateur.

Pour une commande à exécution différée, par exemple une commande PROCEDURE qui permet de "déclarer" une procédure, il faut d'une part chaîner entre elles les diverses zones de mémoire allouées aux commandes constituant le corps de la procédure mais, en plus, associer à ce chaînage l'identicateur de point d'entrée de la procédure afin de "retrouver" cette dernière lorsque l'utilisateur en provoquera l'activation par une commande CALL. Ensuite, dès que la procédure est enregistrée, le contrôle est rendu, à haut niveau, à l'utilisateur.

Exemple de déclaration de procédure :

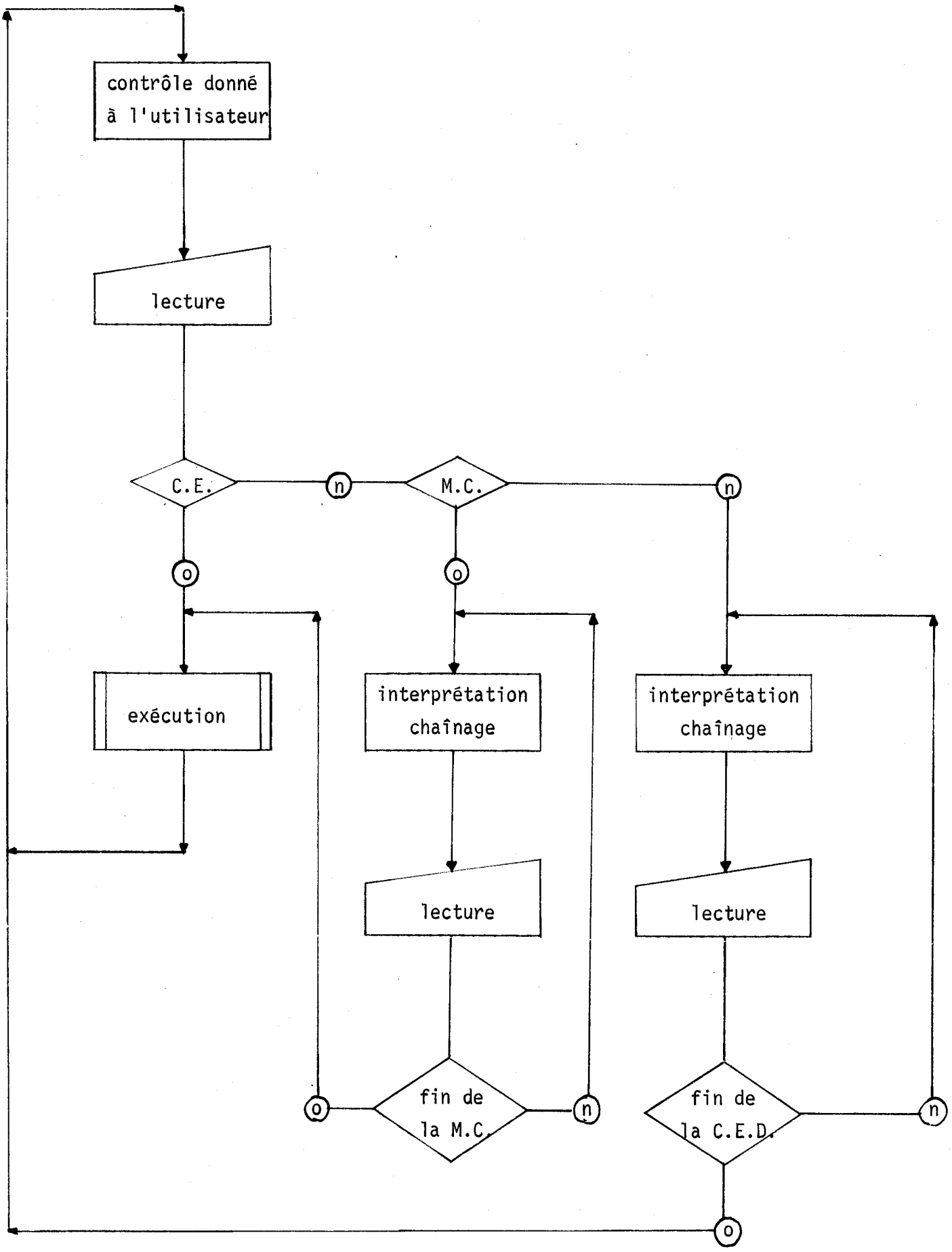
```
VALFAC : PROCEDURE(A,B) ;
         LOCAL I;
         DO I=1 TO B;
            $A(I) = FAC(I) ;
         END ;
END VALFAC ;
```

Les commandes à effet différé ou semi-différé nous ont conduit à adopter une méthode de traitement de listes. L'"unité de programme" est représentée par une structure arborescente, en général à deux niveaux. Le premier niveau est constitué par les noeuds, le deuxième par les cellules. Les noeuds ont pour but de chaîner entre elles les commandes rangées dans les cellules. On verra par la suite que les noeuds ont tous la même forme, par contre il existe différents types de cellules. Le chaînage ainsi obtenu sera appelé le train de commandes.

Nota : en vue d'optimiser le module correspondant au système FORDECAL nous avons tenu compte, pour la détermination des noeuds et des cellules, du type d'ordinateur sur lequel le système est amené à fonctionner, néanmoins la technique utilisée est indépendante de ces problèmes d'optimisations.

Schématiquement le système FORDECAL fonctionne de la manière suivante :

(C.E. : commande élémentaire,
M.C. : macro-commande,
C.E.D. : commande à exécution différée)


contrôle donné à l'utilisateur
lecture
C.E.
n
M.C.
n
o
o
exécution
interprétation chaînage
interprétation chaînage
lecture
lecture
o
fin de la M.C.
n
fin de la C.E.D.
n
o

3.3.. LES ENTREES/SORTIES

Nous avons vu que la communication avec le système se fait au moyen des commandes. Celles-ci peuvent être soit tapées au clavier du terminal soit "lues" à partir d'un fichier. Elles peuvent, de plus, être "composées" sur une ou plusieurs "lignes", la fin physique d'une commande étant signalée par le caractère <;>.

Chaque ligne est stockée dans une zone tampon de C.M.S. La routine GETSTRG, ajoutée à la bibliothèque Formac, se charge de prendre dans cette zone les commandes et les range, une aprés l'autre, dans la variable d'identificateur INPUT du système. Cette variable est ensuite exploitée pendant la phase d'interprétation.

Nota : - GETSTRG se charge de trouver le nom de fichier sur lequel se fait la lecture, enlève les blancs en trop en n'en laissant qu' un seul,

- lors de l'exécution d'une commande OPTSET READER(*nom de fichier*) GETSTRG se charge aussi de stocker, en l'empilant, ce nouveau *nom de fichier* afin de permettre un retour automatique sur le fichier précédent lors de la rencontre de la fin de fichier. Il est prévu une sauvegarde pour quatre fichiers,

- du fait de l'utilisation de sous-programmes propres à CMS, il est possible d'utiliser, lors de la composition, les caractères d'annulation <@> et <ç>.

L'impression des messages d'erreurs du système FORDECAL, de certains résultats se fait par l'intermédiaire de la routine PRINT1, ajoutée à la bibliothèque Formac. Cette routine se charge d'imprimer sur un fichier déterminé en utilisant des sous-programmes de CMS.

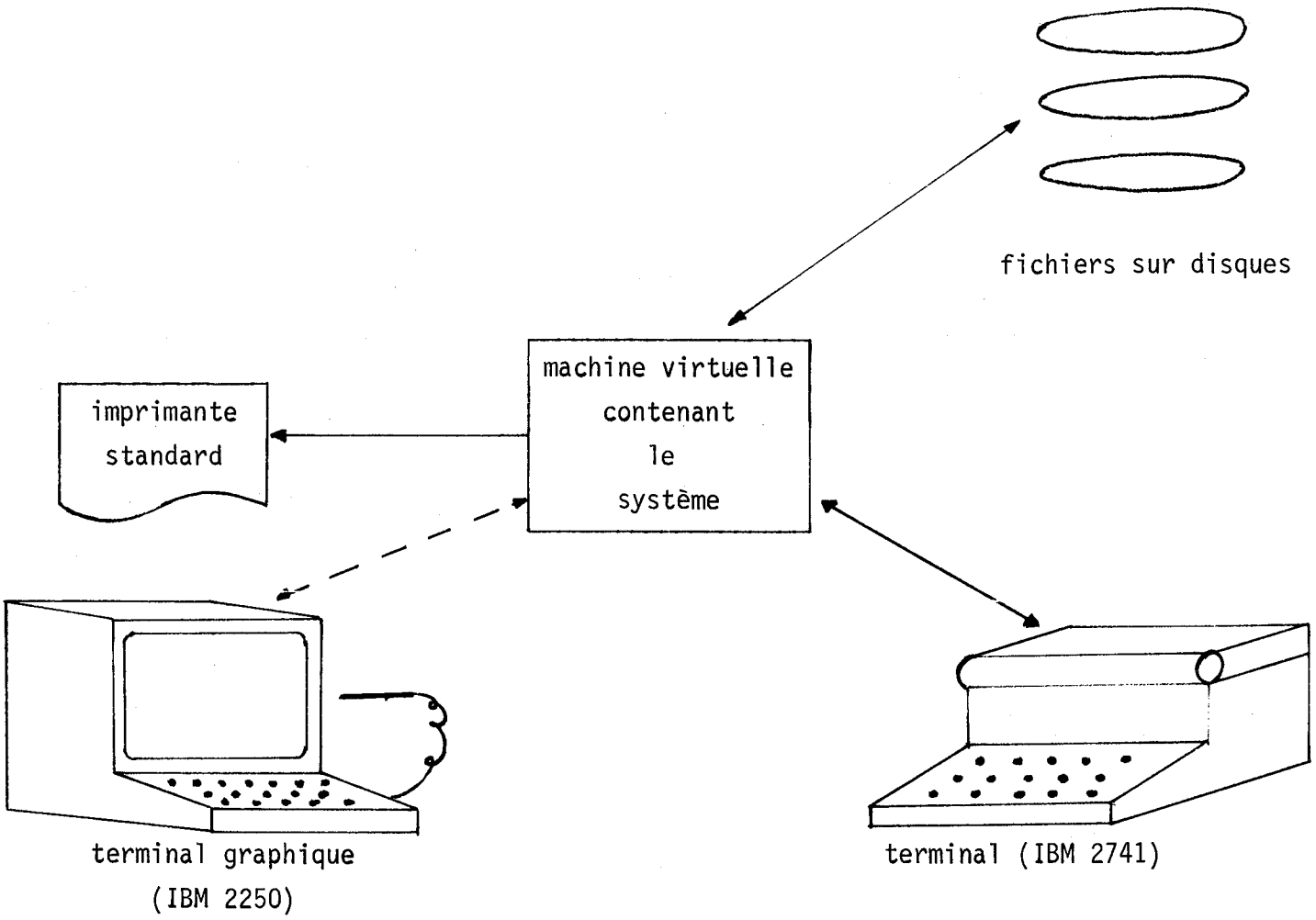
Les impressions provoquées par le système Formac se font par l'intermédiaire des routines classiques DENOUT, DENOPR et DENPNT. Ces routines ont été modifiées d'une part afin d'autoriser les impressions à se faire sur un fichier autre que le papier de la console (SYSPRINT utilisé par les sous-programmes PL/1 appelés par DENPNT), et d'autre part en vue des

options UNDL et NOUNDL (résultats soulignés ou non).

Remarques : - les options ERASE, PRINTF et REWIND permettant respectivement d'effacer un fichier, de le faire imprimer sur l'imprimante standard ou de "rebobiner" un fichier, sont exécutées par l'intermédiaire des sous-programmes correspondant de CMS,

- les options COPY et ECHO, permettant de copier sur un fichier ce que le système est en train de "lire" ou de répéter au terminal les commandes lues à partir d'un fichier font partie de GETSTRG.

Schématiquement l'environnement du système est le suivant :



4. LE TRAIN DE COMMANDES

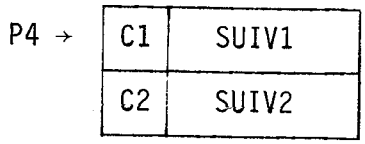
Celui-ci est constitué par les noeuds et les diverses cellules allouées.

4.1.. LES NOEUDS

Ils sont constitués par la structure suivante :

```
1 S4 BASED(P4),
 2 SUIV1 POINTER,
 2 SUIV2 POINTER;
```

Par l'utilisation judicieuse des structures PLX1 et PLX2 (cf. § 2) on a en fait dans une allocation de la variable S4 :



C1 : code utilisé pendant la phase d'exécution. Indique le type de commande,

SUIV1 : pointeur. Soit 000000 dans le cas du dernier noeud, soit l'adresse du noeud suivant,

C2 : code utilisé pendant la phase d'effacement, il est associé au type d'action à entreprendre pour libérer la cellule attachée,

SUIV2 : pointeur. Soit 000000 soit l'adresse de la cellule où est rangée la commande.

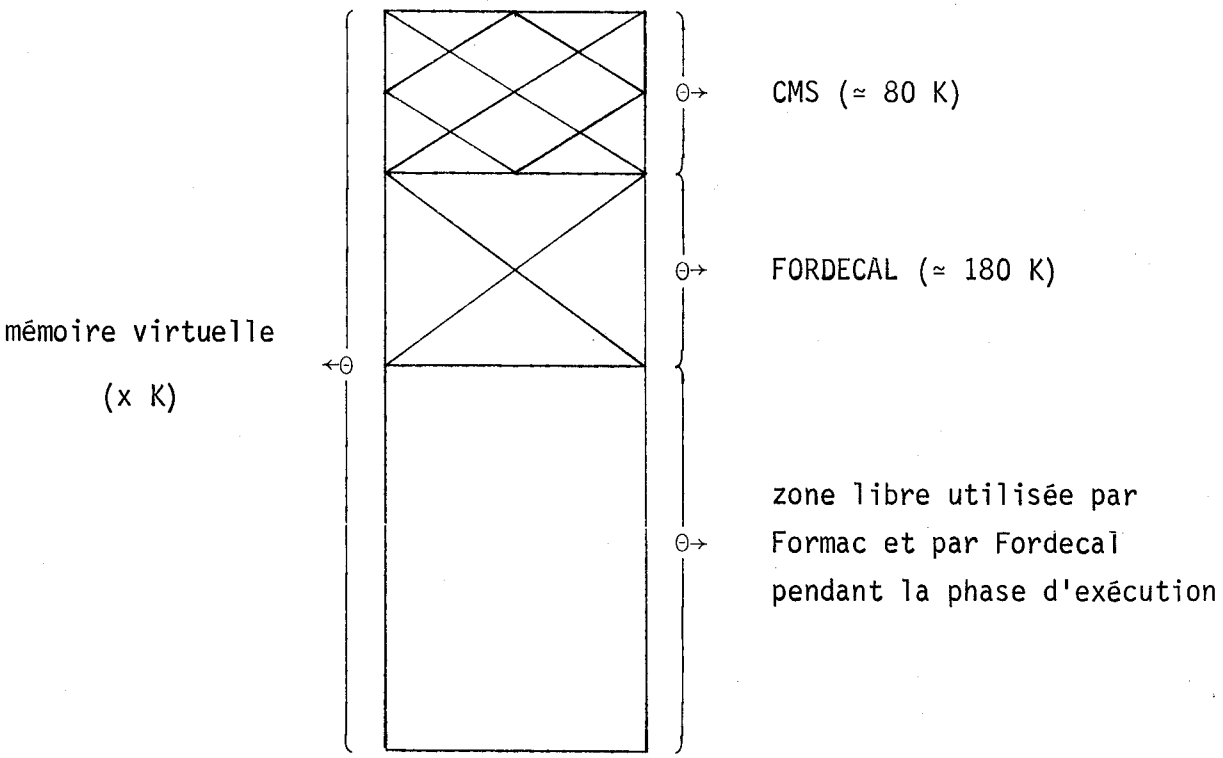
4.2.. LES CELLULES

Il y a onze types différents de cellules selon le genre de commande à ranger. Nous les étudierons par la suite.

4.3.. REMARQUES

Il est bien évident qu'avec cette technique de stockage des commandes, dans le cas de commandes à exécution différée, ou semi-différée, la zone libre, à un moment donné, dépend à la fois du nombre de sous-programmes enregistrés et éventuellement de l'"ampleur" de la macro-commande en cours d'exécution, list processing propre au système FORDECAL, et du nombre de variables utilisées ainsi que des expressions qui leur sont affectées, list processing propre au système FORMAC. Néanmoins l'utilisateur peut, à chaque instant, connaître son occupation mémoire, libérer de la mémoire soit par la commande SCRATCH en ce qui concerne les sous-programmes, soit par la commande REMOB, ou ATOMIZE, en ce qui concerne les variables utilisées.

L'occupation mémoire est la suivante :



5. LE SUPERVISEUR

Nous utiliserons en fait cette terminologie pour désigner la séquence de programme de l'interpréteur de FORDECAL qui prépare les branchements pendant les phases de lecture et d'exécution et prend en compte les erreurs.

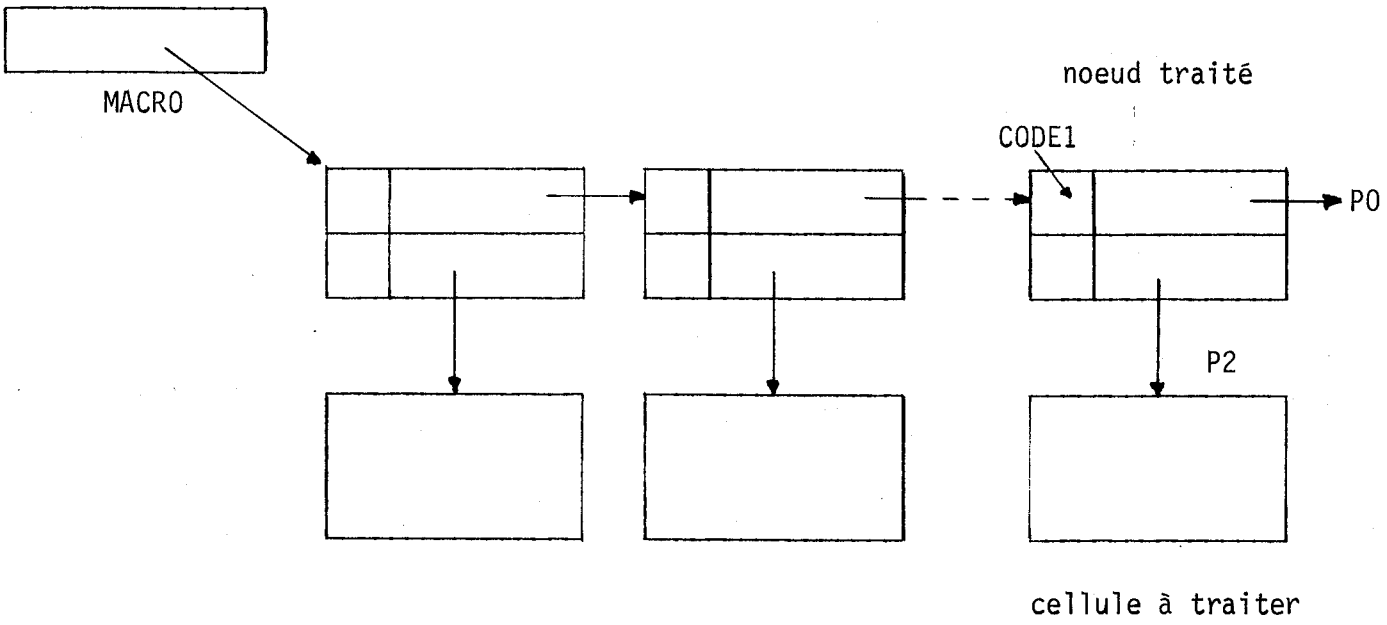
Chaque commande comporte, en général, un mot clé placé en début de la chaîne d'entrée. L'interpréteur contient, entre autre, la table de reconnaissance de ces symboles. Cette table, appelée KEYWORD, est constituée sous la forme d'un tableau au sens PL/1, dont chaque élément est rangé sur un double mot. Le premier octet de chaque élément contient l'"adresse", sous forme de BIT, de la séquence de traitement lecture-interprétation associée au mot clé, l'identificateur du mot clé figure, lui, sur les 7 derniers octets de l'élément. La reconnaissance du mot clé d'une commande se fait par la technique classique de "hash-coding".

Pour une commande à exécution immédiate, une fois le mot clé décodé, grâce à KEYWORD, le superviseur se branche sur la séquence d'interprétation correspondante. Une analyse syntaxique et sémantique est alors effectuée et si la commande est d'une part autorisée et d'autre part correcte, l'exécution de la commande interprétée se déroule. Comme nous l'avons déjà dit, le contrôle est ensuite rendu, à haut niveau, à l'utilisateur.

Si la commande lue correspond à la première commande d'une macro-commande, aprés la séquence d'interprétation et allocation du premier noeud, le système range dans le "pseudo-régistre" MACRO l'adresse de ce noeud. Puis la phase lecture-interprétation continue à se dérouler : reconnaissance du mot clé des commandes suivantes, branchement sur la séquence correspondante de traitement en lecture (analyse syntaxique et sémantique, allocation mémoire, rangement de la commande interprétée dans la cellule, chaînage du noeud correspondant à cette commande au noeud précédent et retour en lecture). Lors de la décodification de la pseudo-commande END, et dans le cas ou celle-ci marque la fin physique de la macro-commande, aprés allocation du noeud correspondant à END, le superviseur confie le contrôle à la phase d'exécution.

Tout d'abord le superviseur "récupère" dans le pseudo-régistre MACRO l'adresse de début de l'unité de programme enregistrée. En utilisant les structures PLX1 et

PLX2 on va alors charger dans le pointeur d'identificateur P2 l'adresse de la cellule attachée au noeud (partie PLX1.DOWN), dans le pointeur d'identificateur P0 l'adresse du noeud suivant (partie PLX1.ACROSS) et le code correspondant au traitement à appliquer aux renseignements contenus dans la cellule est chargé dans la variable INDTAB (partie PLX2.CODE1). On utilise ensuite la variable d'identificateur ZEICHEN2 comme indice d'un tableau d'étiquettes pour effectuer le branchement sur la séquence de traitement. (cf. page 6, 3° partie).



Ce que l'on a à un instant donné

Nous verrons plus loin le traitement des informations contenues dans la cellule.

Si la commande lue correspond à une déclaration de sous-programme, ainsi que nous l'avons déjà dit, le superviseur commence par vérifier que l'identificateur de *point d'entrée* figurant dans la commande n'appartient pas déjà à la table des points d'entrée. Si oui, ce nouveau *point d'entrée* est ajouté à la table. La suite du traitement lecture-interprétation est identique à celle du corps d'une macro-commande. Lors de la décodification de la pseudo-commande END marquant la fin physique du sous-programme, le superviseur rend le contrôle, à haut niveau, à l'utilisateur. Le pseudo-régistre PROCE_0 contient l'adresse de la table des points d'

entrée.

Lorsqu'une erreur est détectée pendant la phase d'exécution, que ce soit une erreur détectée par le système FORMAC ou par le système FORDECAL, le superviseur confie alors le contrôle à une séquence particulière qui est chargée d'imprimer le message d'erreur correspondant et surtout de stopper l'exécution de l'unité de programme, de plus haut niveau, en cours. S'il s'agit d'une macro-commande, l'espace mémoire occupé par les noeuds et les cellules sera libéré, en utilisant le code rangé sur le premier octet du deuxième mot de chaque noeud, et le contrôle est finalement rendu, à haut niveau, à l'utilisateur. S'il s'agit d'un sous-programme, le système ne récupère pas l'espace mémoire occupé. Les arguments du sous-programme ont, si l'appel a été fait par nom, la valeur qu'avait le paramètre correspondant au moment de la détection de l'erreur; si l'appel a été fait par valeur il est évident que les arguments ne sont pas modifiés.

6. LES DIVERSES CELLULES UTILISEES

Il y a, comme nous l'avons déjà indiqué, onze types différents de cellules. Avant d'aborder leur utilisation comme constituant du train de commandes, décrivons leur forme extérieure. Celle-ci sera donnée sous forme de déclaration PL/1

A)
```
1 S1 BASED(P1),
  2 VARC CHARACTER(8),
  2 VAL1 CHARACTER(98),
  2 VAL2 CHARACTER(98),
  2 VAL3 CHARACTER(98),
  2 VAL4 CHARACTER(2),
  2 ETIQ POINTER,
  2 SAUT POINTER;
```

Cette cellule sera utilisée pour les groupes DO itératifs. La variable VAL4 n'est là que pour forcer l'alignement.

B)
```
1 S2 BASED(P2),
  2 C0 FIXED BINARY(31),
  2 N1 FIXED BINARY(31),
  2 ST CHARACTER(I1 REFER(N1));
```

Il s'agit de la cellule la plus fréquemment utilisée. A noter l'allocation dynamique de la zone ST. Cette allocation sera faite par multiple de 8 octets.

C)
```
1 S3 BASED(P3),
  2 END1  POINTER,
  2 END2  FIXED BINARY(15),
  2 TRACE BIT(8),
  2 PRINT BIT(8);
```

Cette cellule sera utilisée par la pseudo-commande END d'un groupe DO itératif et par l'accompagnateur.

D)
```
1 S5 BASED(P5),
  2 FAUX POINTER,
  2 LONG FIXED BINARY(31),
  2 TEST CHARACTER(I1 REFER(LONG));
```

S5 sera utilisée pour stocker la partie *test* des commandes conditionnelles. La zone TEST est allouée dynamiquement.

E)
```
1 S6 BASED(P6),
  2 NEXT POINTER,
  2 LABE CHARACTER(8),
  2 BLKD FIXED BINARY(31),
  2 DEPA POINTER,
  2 ARRI POINTER;
```

S6 sera utilisée, en cours de lecture, pour stocker les étiquettes. La zone BLKD, inutilisée, n'est là que pour forcer l'alignement.

F)
```
1 S7 BASED(P7),
  2 FOLGEN POINTER,
  2 FORMAL CHARACTER(8),
  2 ENTADR POINTER;
```

S7 sera utilisée comme élément de la table des points d'entrée ou comme cel-

lule contenant soit les paramètres des sous-programmes, soit le *procname* d'une commande CALL d'une unité de programme, soit encore comme constituant de la table des étiquettes pendant la phase d'exécution.

G)
```
1 S8 BASED(P8),
 2 P_PAR POINTER,
 2 P_ENT POINTER,
 2 P_LAB POINTER,
 2 P_FIL POINTER,
 2 P_LOC POINTER,
 2 P_ETI POINTER,
 2 P_DEB POINTER,
 2 N_REC FIXED BINARY(31);
```

Cette cellule est attachée au noeud correspondant à la première commande d'une déclaration de sous-programme (procédure ou fonction-procédure). Elle contient tous les renseignements relatifs aux paramètres, à leur spécification, aux variables locales et aux étiquettes, au programme de mise au point attaché, ou non, à ce sous-programme, à son niveau de récursion.

H)
```
1 S9 BASED(P9),
 2 SPR CHARACTER(8);
```

S9 est utilisée pour contenir le *procname* d'une commande SCRATCH interne à une unité de programme.

I)
```
1 BSA BASED(P10),
 2 W_BK POINTER,
 2 W_NO POINTER,
 2 W_LO POINTER,
 2 W_ET POINTER,
 2 W_PR POINTER,
 2 W_N1 FIXED BINARY(15),
 2 W_N2 BIT(16);
```

BSA (Block Storage Area) est utilisée pendant la phase de lecture et pendant

la phase d'exécution. En lecture elle est allouée à chaque macro-commande, en exécution elle est allouée pendant le "prologue" activé par l'ouverture d'un bloc ou l'activation d'un sous-programme.

J)
```
1 S11 BASED(P11),
  2 ADRESS POINTER,
  2 ETIKET CHARACTER(8),
  2 NUMBID FIXED BINARY(15),
  2 NUMBLK FIXED BINARY(15);
```

S11 sera utilisée par les commandes GOTO. La partie NUMBID ne figure que pour l'alignement.

K)
```
1 PSA BASED(P12),
  2 BACK POINTER,
  2 BASE POINTER,
  2 SUIV POINTER,
  2 NAME CHARACTER(8),
  2 TRAC BIT(8),
  2 PRIN BIT(8),
  2 COMD FIXED BINARY(15);
```

PSA (Procedure Storage Area) est allouée lors du prologue des sous-programmes. Elle est chainée à la BSA correspondante.

7. ENREGISTREMENT DES COMMANDES

Nous allons examiner dans ce paragraphe l'allocation de cellule correspondant à chaque commande, pendant la phase de lecture-interprétation.

Nous commencerons par les commandes élémentaires, puis les macro-commandes, enfin la déclaration de sous-programme et l'accompagnateur.

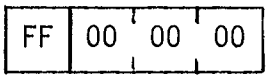
On trouvera, à chaque fois, les références, relatives au Manuel Utilisateur (2° partie), de la commande.

Remarques : - les codes figurant dans les noeuds et les cellules sont en représentation hexadécimale,

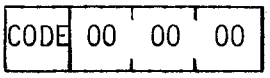
- les indications figurant entre parenthèses sous les cellules font référence au type de cellule utilisée,
- le caractère *b* figurant dans la partie syntaxe indique la présence nécessaire d'au moins un caractère blanc (cf. 2° partie),
- au départ, les variables suivantes sont respectivement initialisées à

PSEUDO (POINTER) ← NULL , pointe sur la BSA précédente,
P5 (POINTER) ← NULL ,
D2 (F.B.(15))← 0 , compteur de boucles itératives

- un pointeur NULL sera, selon les cas, représenté par :



cas classique,

ou 

- signalons aussi que le système commence par vérifier que la commande est complète. Dans la négative le superviseur donne le contrôle à la séquence de traitement des erreurs en phase lecture-interprétation.

7.1.. LES PRINCIPAUX CODES ET LEUR INTERPRETATION

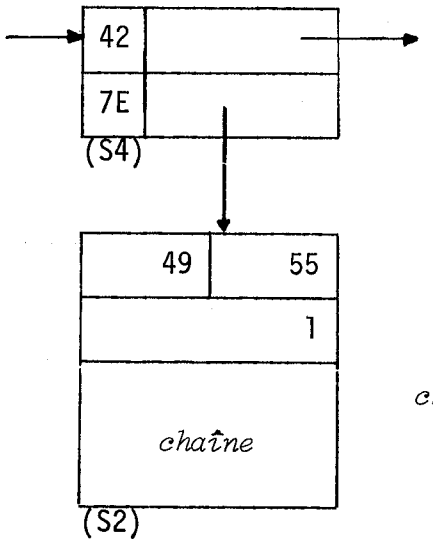
Code1	Code2	Commande	Partie CO de la cellule S2
		affectation avec impression	49 - 55
		affectation sans impression	49 - 56
		ATOMIZE	49 - 51
		REMOB	4B - /
		OPTSET	49 - 50
42	7E ←0	LIST	49 - 4E
		IDENT	52 - 54
		EQUIV	52 - 53
		LOP	49 - 4D
		NARGS	49 - 4F
		GET	46 - /
		PUT	4C - /

39	7B	CALL
3A	7C	SCRATCH
3B	80	GOTO
41	79	IF
44	81	DO non itératif BEGIN
38	7A	DO itératif
40	77	END d'un groupe DO non itératif, bloc intérieur
43	82	END d'un groupe DO non itératif, bloc extérieur
3C	7D	END d'un groupe DO itératif, bloc intérieur,
76	82	END d'un groupe DO itératif, bloc extérieur,
3F	77	END d'un groupe BEGIN, bloc intérieur,
3E	82	END d'un groupe BEGIN, bloc extérieur,
3D	82	END d'un sous-programme
45	77	RETURN

7.2.. COMMANDE D'AFFECTATION AVEC IMPRESSION [§ 5.1.]

syntaxe : $*var* = *exp* [, ...] ;

structure :



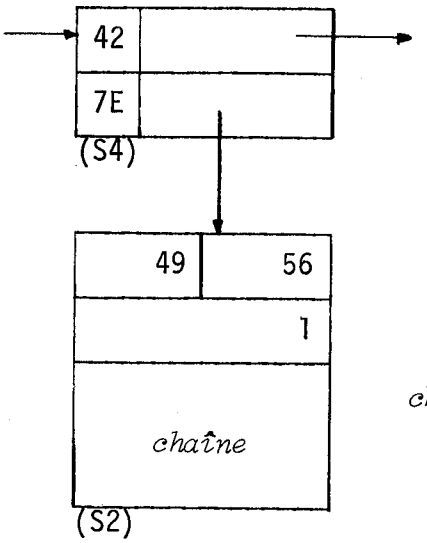
l : longueur de la zone *chaîne*

chaîne : toute la commande sauf le caractère <$>

7.3.. COMMANDE D'AFFECTATION SANS IMPRESSION [§ 5.1.B.]

syntaxe : *var* = *exp* [, ...] ;

structure :



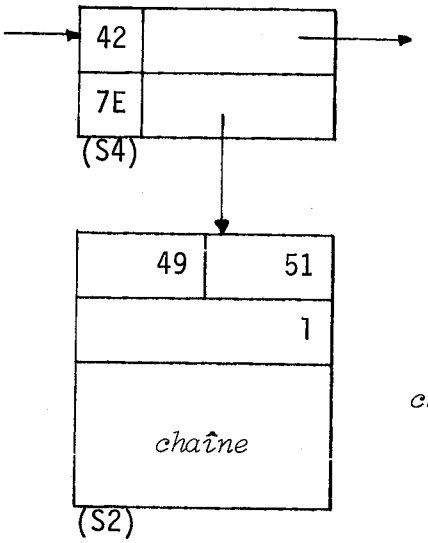
l : longueur de la zone *chaîne*

chaîne : toute la commande

7.4.. COMMANDE ATOMIZE [§ 5.3.A.]

syntaxe : ATOMIZE$_b$*var* [, *var* ...] ;

structure :



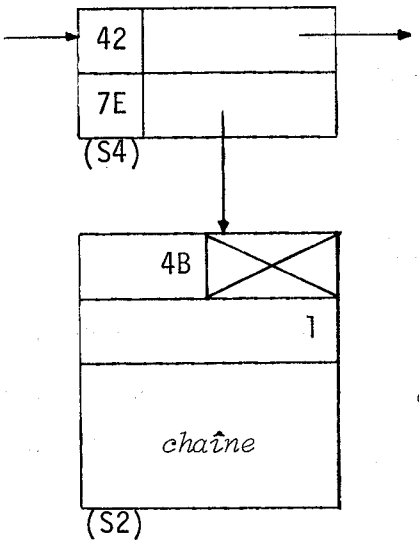
l : longueur de la zone *chaîne*

chaîne : toute la partie de la commande suivant $_b$

7.5.. COMMANDE REMOB [§ 5.3.B.]

syntaxe : REMOB$_b$*var* [, *var* ...] ;

structure :



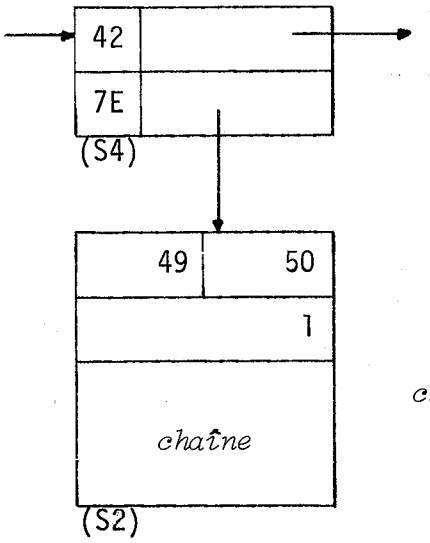
l : longueur de la zone *chaîne*

chaîne : toute la partie de la commande suivant $_b$, y compris le <;>

7.6.. COMMANDE OPTSET [§ 5.3.C.]

syntaxe : OPTSET$_b$*option* [, *option* ...] ;

structure :



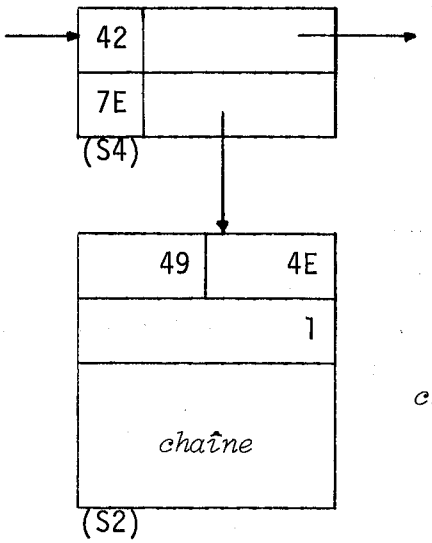
l : longueur de la zone *chaîne*

chaîne : toute la partie de la commande suivant $_b$

7.7.. COMMANDE LIST [§ 5.3.D.]

syntaxe : LIST$_b$*argument* [, *argument* ...] ;

structure :



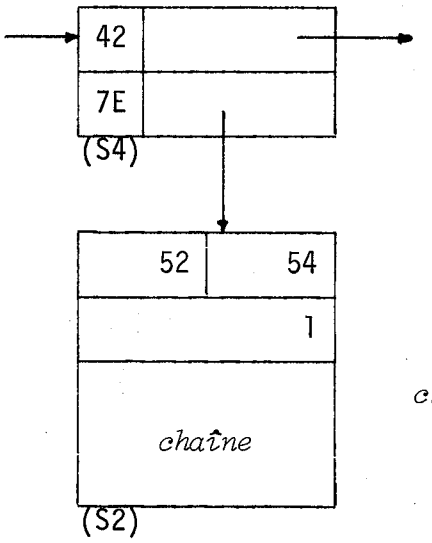
l : longueur de la zone *chaîne*

chaîne : toute la partie de la commande suivant $_b$

7.8.. COMMANDE IDENT [§ 5.4.A.]

syntaxe : IDENT$_b$ *exp1* ! *exp2* ;

structure :



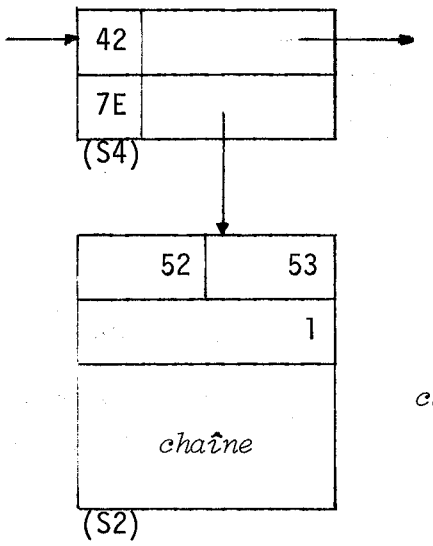
l : longueur de la zone *chaîne*

chaîne : *exp1* ! *exp2*

7.9.. COMMANDE EQUIV [§ 5.4.B.]

syntaxe : EQUIV$_b$ *exp1* ! *exp2* ;

structure :



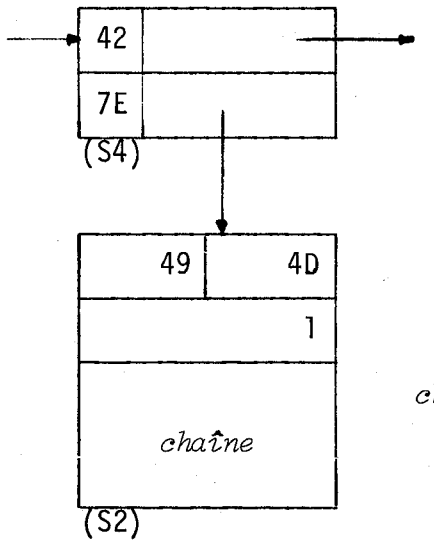
l : longueur de la zone *chaîne*

chaîne : *exp1* ! *exp2*

7.10.. COMMANDE LOP [§ 5.4.C.]

syntaxe : LOP$_b$*var* [, *var* ...] ;

structure :



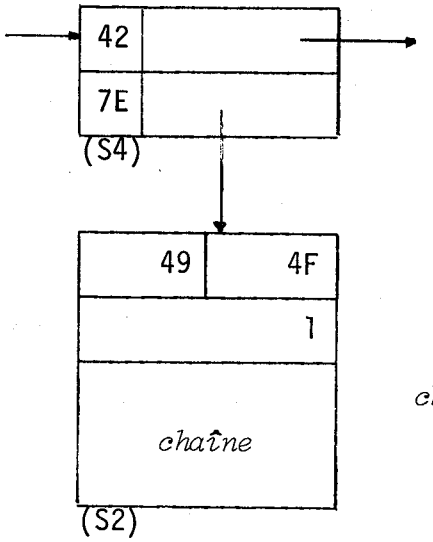
l : longueur de la zone *chaîne*

chaîne : toute la partie de la commande suivant $_b$

7.11.. COMMANDE NARGS [§ 5.4.D.]

syntaxe : NARGS$_b$*var* [, *var* ...] ;

structure :



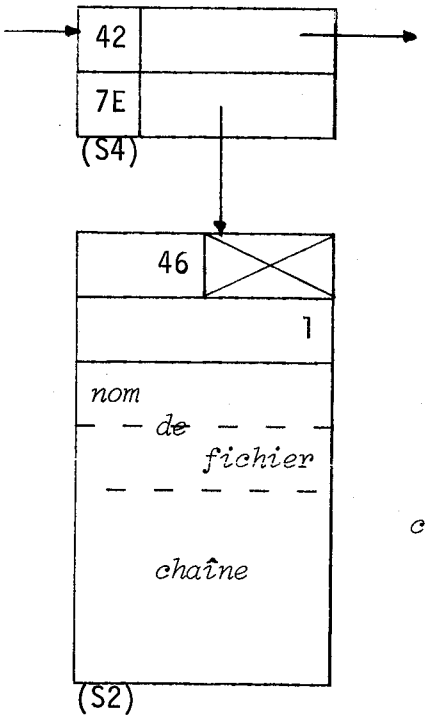
l : longueur de la zone *chaîne*

chaîne : toute la partie de la commande suivant $_b$

7.12.. COMMANDE GET [§ 5.5]

syntaxe : GET$_b$[(*nom de fichier*)] *var* [, *var* ...] ;

structure :



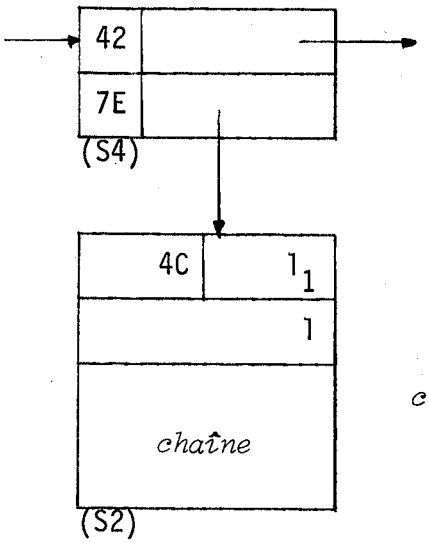
l : longueur de la zone *chaîne* + 8

chaîne : la liste des variables *var*

7.13.. COMMANDE PUT [§ 6.2]

syntaxe : PUT$_b$*message* ;

structure :



l_1 : longueur du *message*

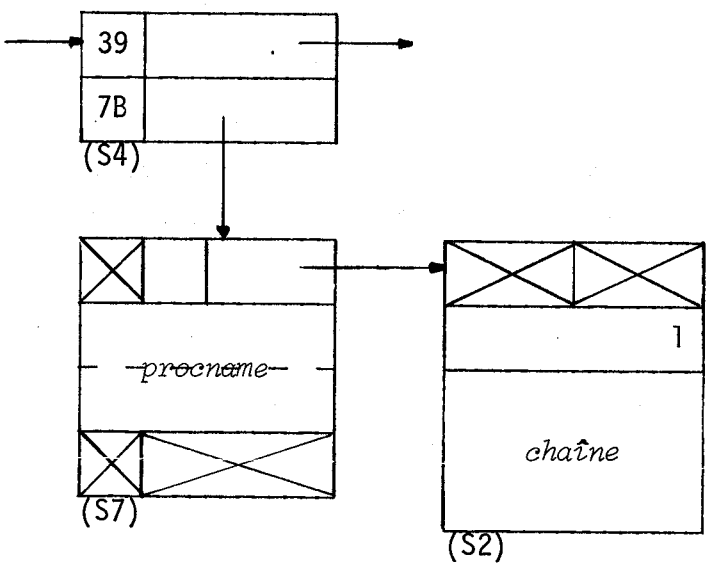
l : longeur de la zone *chaîne*

chaîne : le *message*

7.14.. COMMANDE CALL [§ 5.6.]

syntaxe : CALL$_b$*procname* [(*liste arguments*)] ;

structure :



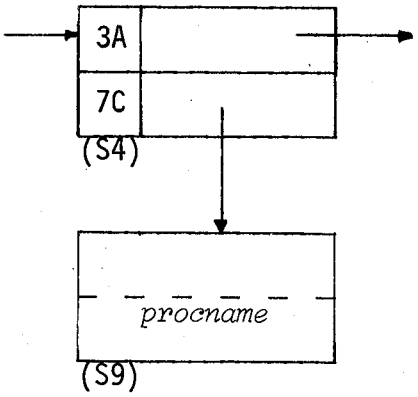
l : longueur de la zone *chaîne*

chaîne : *liste arguments*

7.15.. COMMANDE SCRATCH [§ 5.7.]

syntaxe : SCRATCH$_b$*procname* ;

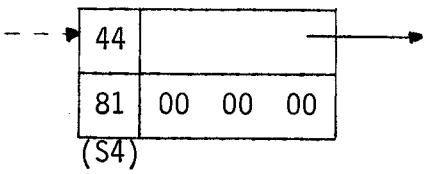
structure :



7.16.. COMMANDE DO NON ITERATIF [§ 6.6.]

syntaxe : DO ;

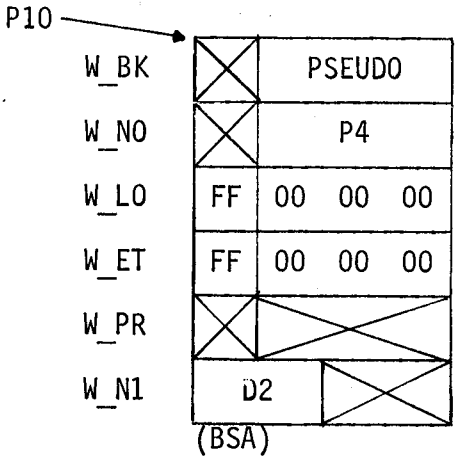
structure :



Le pointeur P4 contient l'adresse du noeud S4.

on alloue de plus :

- la variable d'identificateur BEGIN (CONTROLLED, BIT) et on lui affecte la valeur '0' pour interdire les déclarations de variables locales,
- la structure de type BSA, dont l'adresse est rangée en P10.



PSEUDO : pointeur sur la BSA précédente ou NULL si premier bloc
P4 : pointeur sur le noeud DO
: initialisé à NULL
: initialisé à NULL
: inutilisé
D2 : compteur de groupe DO itératif

puis PSEUDO = P10

7.17.. COMMANDE DO ITERATIF [§ 6.7.]

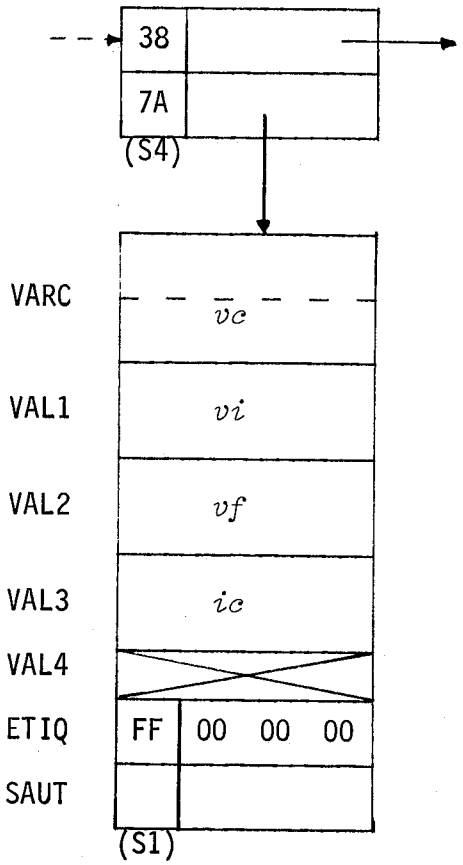
syntaxe : DO$_b$*vc*=*vi*$_b$TO$_b$*vf* [$_b$BY$_b$*ic*] ;

Le superviseur commence

- par vérifier la présence des symboles <=>, <TO> et <BY>. Si un des deux premiers est omis, le message de code FDC014I est émis (cf. 2° partie § 11) et la séquence de traitement des erreurs détectées en phase lecture est activée,
- puis par tester si la boucle est exécutable dans le cas où les quantités *vi*, *vf* [et *ic*] sont des constantes numériques. Si non le superviseur donne le contrôle à la séquence de traitement des erreurs détectées en phase lecture.

Si tout est valide, il y a alors allocation de la structure suivante :

structure :

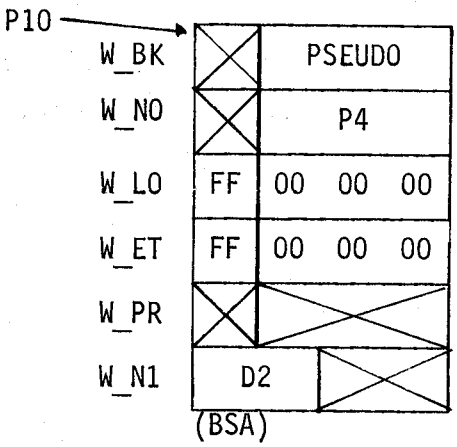


Le pointeur P4 contient l'adresse du noeud S4

Le système alloue ensuite les variables :

- BEGIN (CONTROLLED, BIT) à qui est affectée la valeur '0' pour interdire les déclarations de variables locales,
- BSA dont l'adresse est rangée en P10

D'autre part le compteur de boucle DO itérative (D2) est incrémenté de 1.



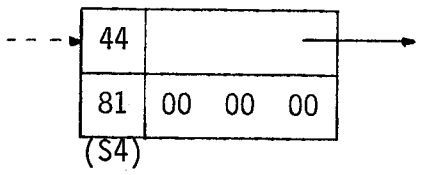
PSEUDO : pointeur sur la BSA précédente ou NULL si premier bloc
P4 : pointeur sur le noeud DO
: initialisé à NULL
: initialisé à NULL
: inutilisé
D2 : compteur de groupe DO itératif

Puis PSEUDO = P10

7.18.. COMMANDE BEGIN [§ 6.8.]

syntaxe : BEGIN ;

structure :



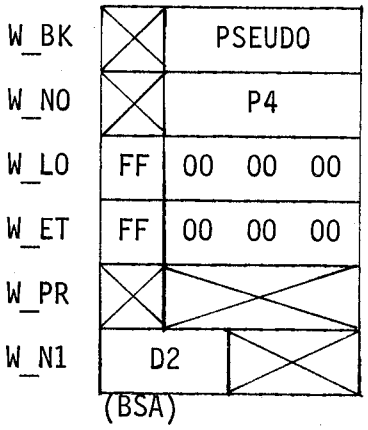
Le pointeur P4 contient l'adresse du noeud S4

On alloue ensuite :

- la variable BEGIN (CONTROLLED, BIT) à qui est affectée la valeur '1' pour autoriser les déclarations de variables locales,
- une structure BSA dont l'adresse est rangée en P10
- ensuite PSEUDO = P10

Remarque : si la commande DO ou BEGIN est de niveau 1 (bloc le plus extérieur), on stocke dans la variable pointeur MACRO l'adresse du noeud S4 correspondant.

P10

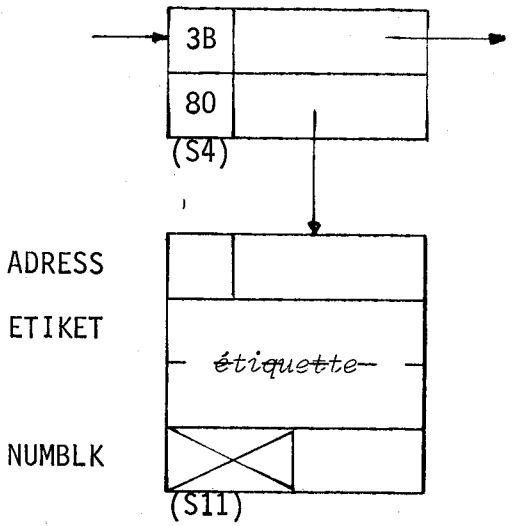


PSEUDO : pointeur sur la BSA précédente ou NULL si premier bloc
P4 : pointeur sur le noeud S4
: initialisé à NULL
: initialisé à NULL
: inutilisé
D2 : compteur de groupe DO itératif

7.19.. COMMANDE GOTO [§ 6.3.]

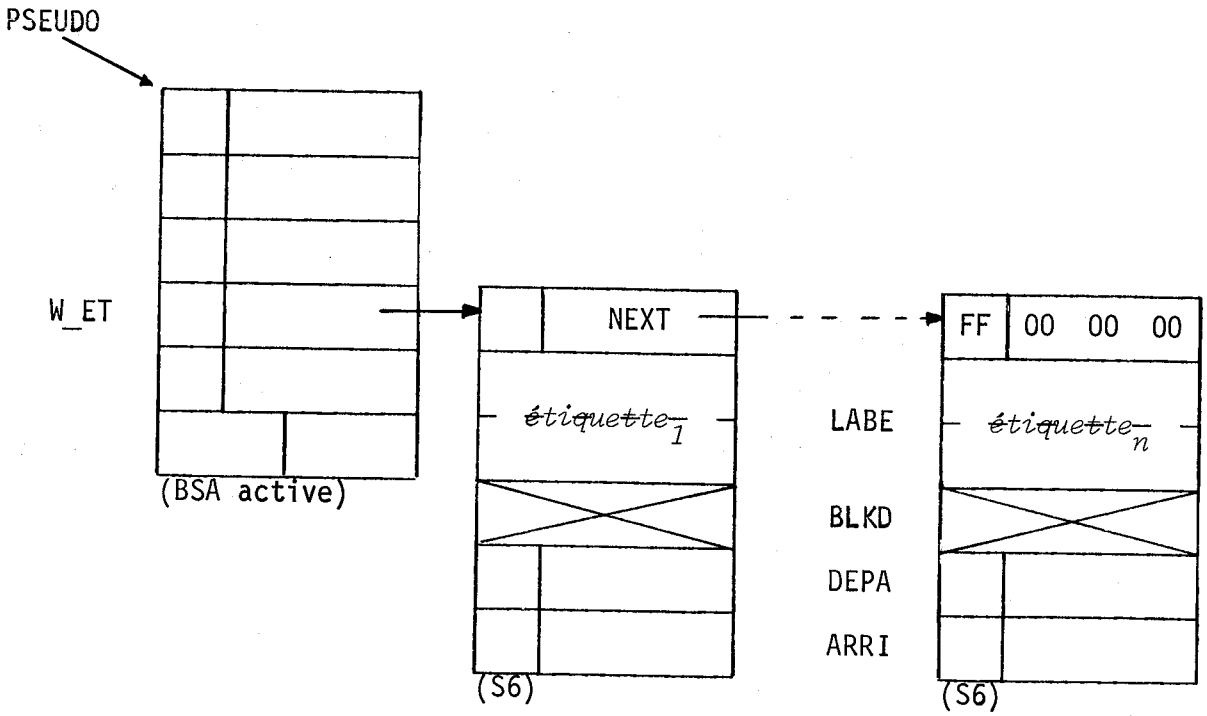
syntaxe : GOTO$_b$ *étiquette* ;

structure :



l'adresse de S11 est rangée en P11

Le système va ensuite consulter la table des étiquettes du bloc en cours de composition. Si cette table existe, l'adresse de son entrée est rangée dans la partie W_ET de la BSA active. Cette table est constituée par des cellules S6 de la manière suivante :



L'allocation d'une cellule S6 est faite lors de la rencontre, pour la première fois dans un bloc donné, d'un identificateur d'étiquette que ce soit aprés le mot clé GOTO ou en préfixe d'une commande. Si c'est aprés le mot clé GOTO, la partie DEPA sera chargée de l'adresse P11 de la cellule S11 attachée au GOTO et la partie ARRI sera chargée par FF000000 (pointeur NULL). Si c'est en préfixe, la partie ARRI sera chargée de l'adresse P4 du noeud correspondant à la commande, la partie DEPA reste inutilisée. La partie LABE contient dans les deux cas l'identificateur d'étiquette.

Si, lors de l'interprétation de la commande GOTO, il existe déjà une cellule S6 contenant le même identificateur d'étiquette , on procède de la manière suivante :

- partie ARRI de S6 = FF000000 (S6 a été créée lors d'une précédente commande GOTO appartenant au même bloc et au même niveau): on alloue ensuite une nouvelle cellule S6 et on procède comme précédemment,
- partie ARRI de S6 ≠ FF000000 (S6 créée lors d'une commande préfixée appartenant, au même niveau, au même bloc) : on charge alors la partie ADRESS de la cellule S11 attachée au GOTO par le contenu de la partie ARRI de la cellule S6.

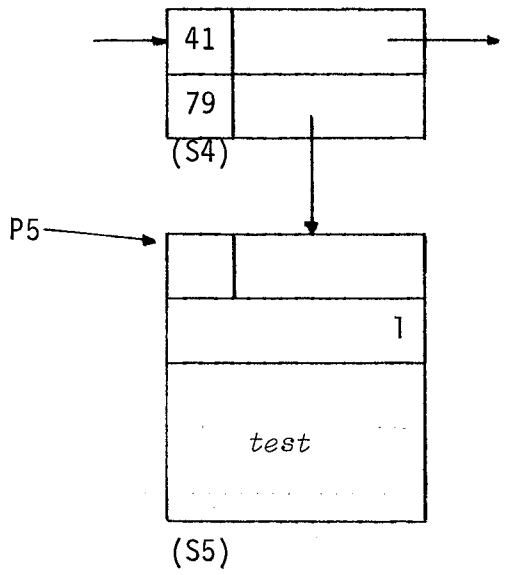
Remarque : on ne peut pas, au même niveau dans un même bloc, avoir deux commandes préfixées par le même identificateur d'étiquette du fait de la présence, ou non, de FF000000 en partie ARRI de la cellule S6. Si c'est le cas le superviseur passe le contrôle à la séquence de traitement des erreurs détectées en phase lecture aprés impression du message de code FDC011I.

7.20.. COMMANDE IF [§ 6.4.]

syntaxe : IF$_b$*test*$_b$THEN$_b$*commande* [ELSE$_b$*commande*]

Il y a tout d'abord allocation de "marqueurs" indiquant qu'une commande IF est traitée, ces marqueurs seront utilisés lors de la fin physique des parties THEN et ELSE, puis allocation de la

structure :



P5 : est chargé de l'adresse de la cellule S5

l : longueur de la zone

On alloue de plus un pointeur PILE_THEN (CONTROLLED) qui est chargé de l'adresse P5. Ce pointeur PILE_THEN sera utilisé, plus tard, pour charger la partie FAUX (pointeur) de S5 par l'adresse de la partie ELSE (réelle ou virtuelle).

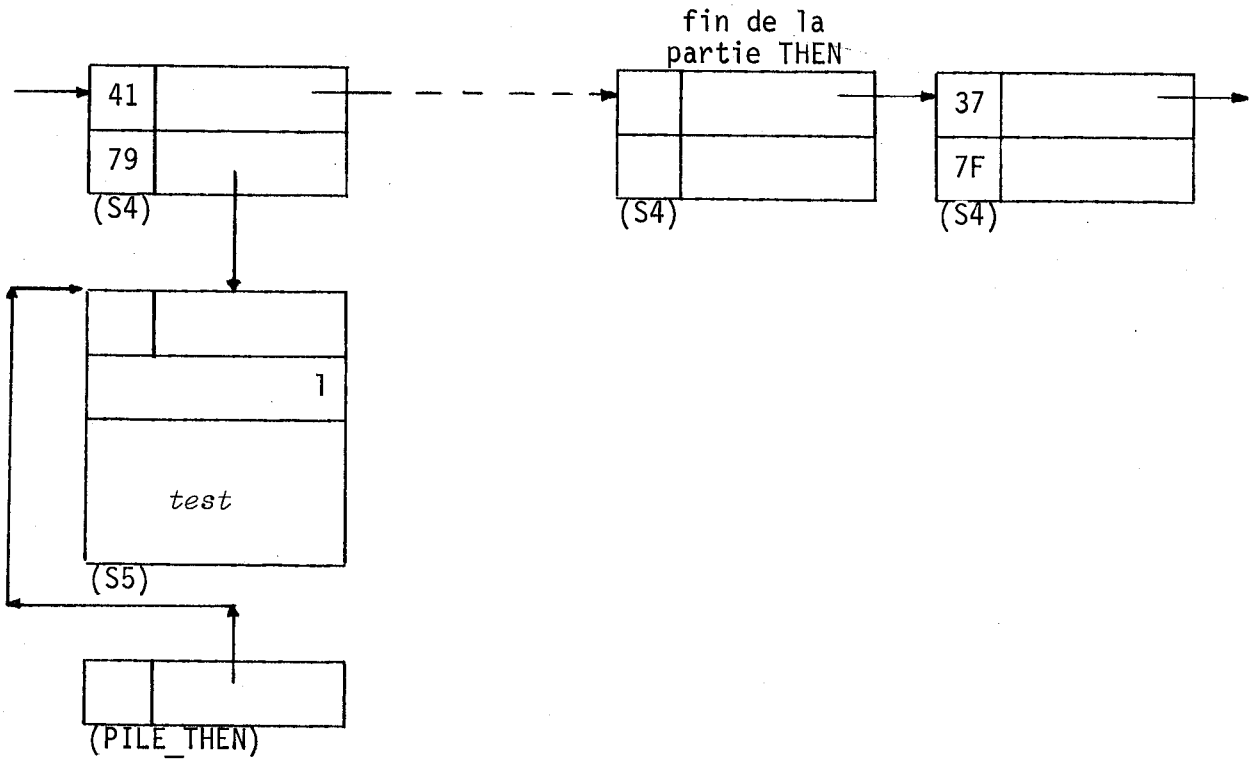
On passe ensuite en décodification de la *commande* de la partie THEN. A noter que *commande* peut être :

- une commande élémentaire,

- une macro-commande,
- une autre commande conditionnelle.

A noter que dans le cas d'une macro-commande, un compteur spécial de groupe est incrémenté.

La partie *commande* de THEN est chaînée à la suite de la cellule S4 (41,79) Lorsque la fin physique de cette partie est rencontrée, le train constitué est le suivant :

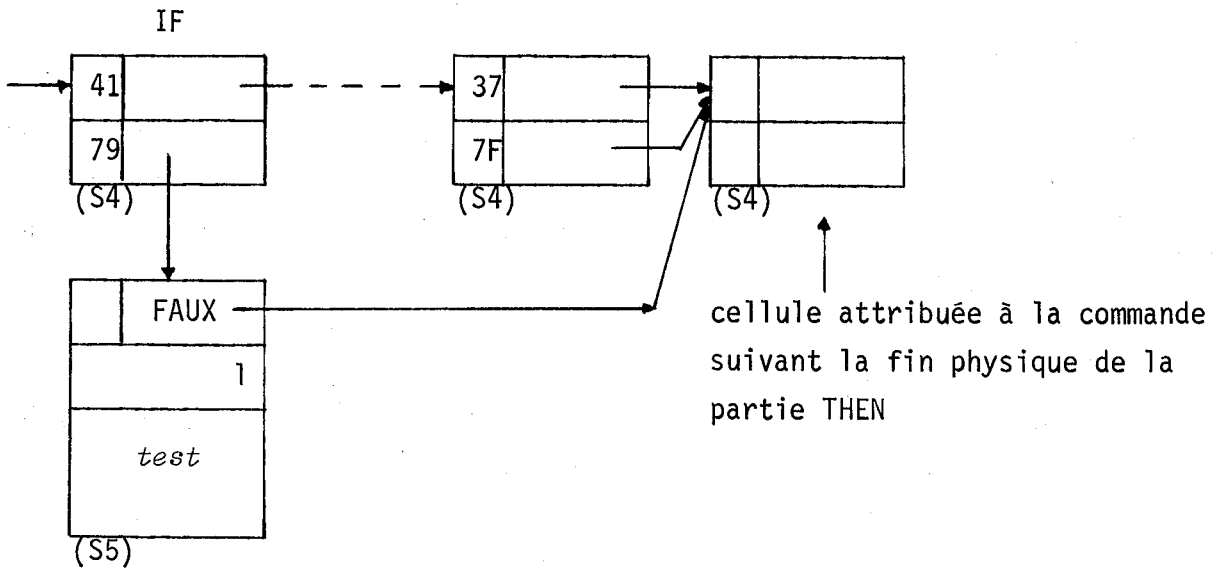


Le "dummy" noeud S4 (37,7F) ne sera utilisé que lors du nettoyage du train.

Deux cas se présentent alors : la partie ELSE est virtuelle ou réelle.

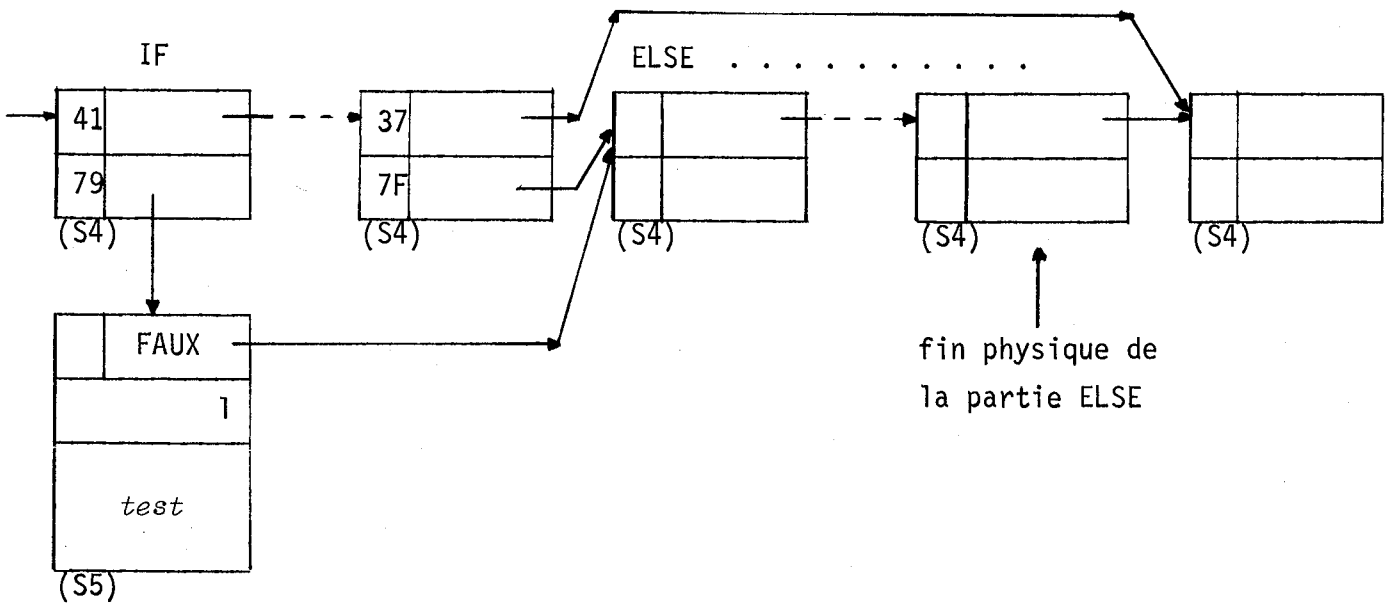
Partie ELSE virtuelle :

la commande conditionnelle ne comporte pas de partie ELSE, le train obtenu est le suivant :



Structure d'une commande conditionnelle sans partie ELSE
ou avec ELSE;

Partie ELSE réelle :

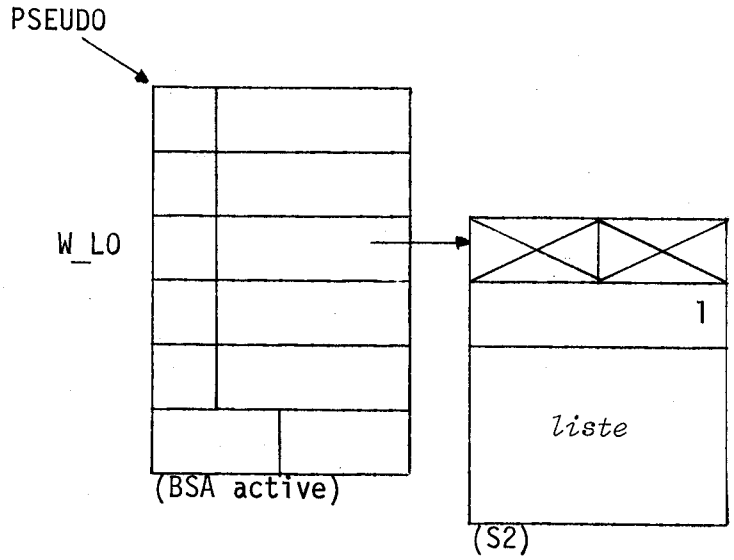


Structure d'une commande conditionnelle avec partie ELSE

7.21.. COMMANDE LOCAL [§ 6.5.]

Le système commence par vérifier que le bit BEGIN est à 1, on se trouve donc au même niveau qu'un bloc BEGIN où les déclarations de variables locales sont autorisées, sinon le superviseur confie le contrôle à la séquence de traitement des erreurs en phase lecture aprés impression du message de code FDC017I.

Les commandes LOCAL ne sont pas chaînées au train principal, mais provisoirement attachées à la cellule BSA du bloc. On obtient :



l : longueur de la zone *liste*

liste : liste des variables locales

Il est ainsi possible d'autoriser plusieurs commandes LOCAL, à un même niveau. Il suffit de vérifier que la partie W_LO de la BSA active est NULL ou chargée d'une adresse. S'il y a une adresse il faut alors récupérer la partie ST de la cellule S2 dont on vient de trouver l'adresse, de libérer cette cellule, d'en allouer une nouvelle qui contiendra l'ancienne liste de variables locales et la nouvelle liste et de ranger l'adresse de cette nouvelle cellule S2 dans W_LO.

7.22.. COMMANDE END

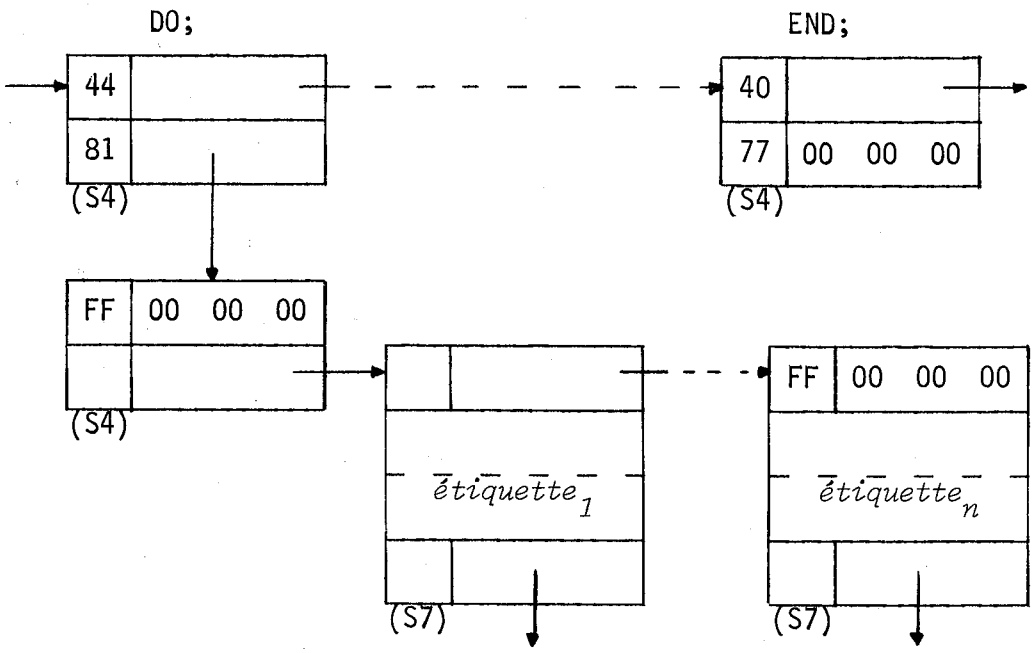
A) MARQUANT LA FIN D'UNE MACRO-COMMANDE DO NON-ITERATIF

Le système commence par vérifier si dans la table des étiquettes du

bloc figure des étiquettes "arrivées" (dont la partie ARRI de la cellule S6 contient une adresse). Dans ce cas il alloue une cellule S7 qui sera chargée de l'identificateur de l'étiquette et de l'adresse de la commande préfixée correspondante (partie LABE et ARRI de la cellule S6, cf. 3° partie § 7.19.). Ces diverses cellules S7 sont chaînées entre elles, les cellules S6 de la table précédente sont toutes détruites.

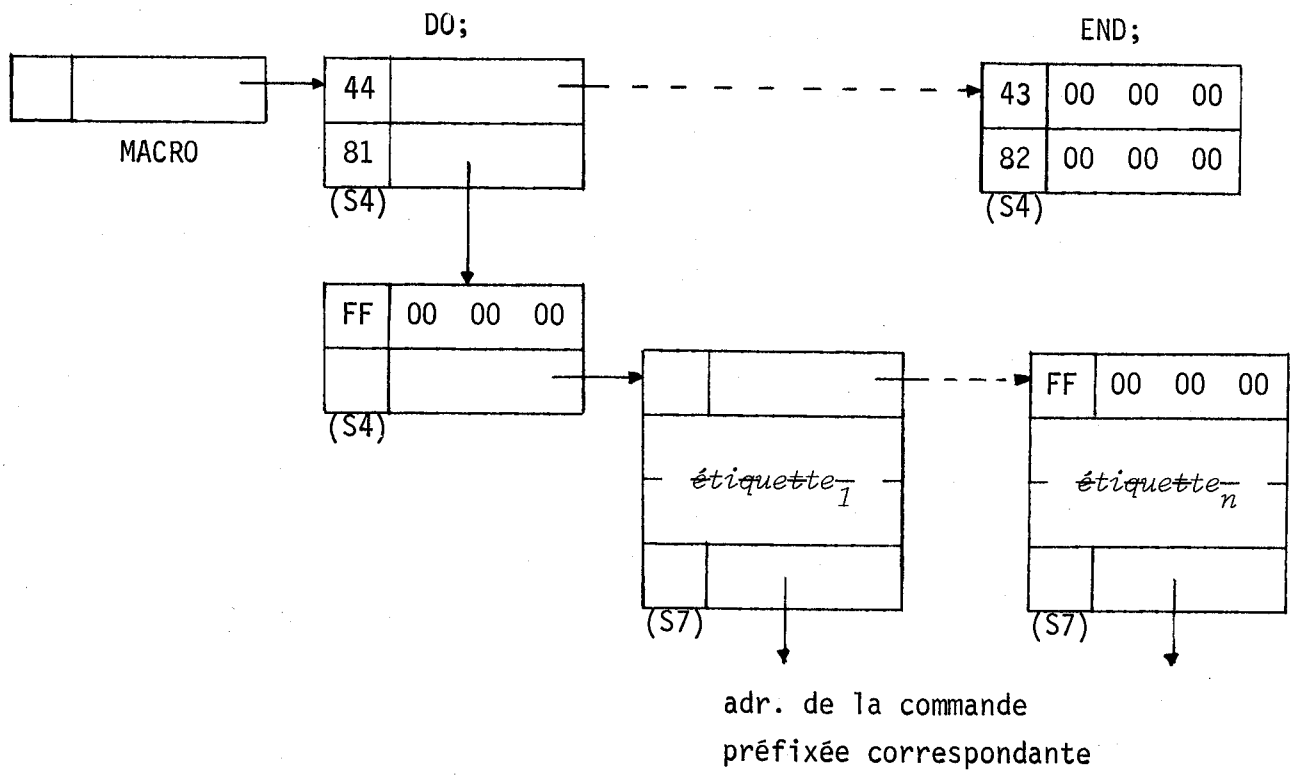
α) COMMANDE END MARQUANT LA FIN PHYSIQUE D'UN BLOC DE NIVEAU > 1

Il s'agit en fait d'un bloc DO non-itératif intérieur à une unité de programme, on obtient alors :



β) COMMANDE END MARQUANT LA FIN PHYSIQUE D'UN BLOC DE NIVEAU = 1

Il s'agit du bloc le plus extérieur, constituant l'unité de programme, on obtient alors :



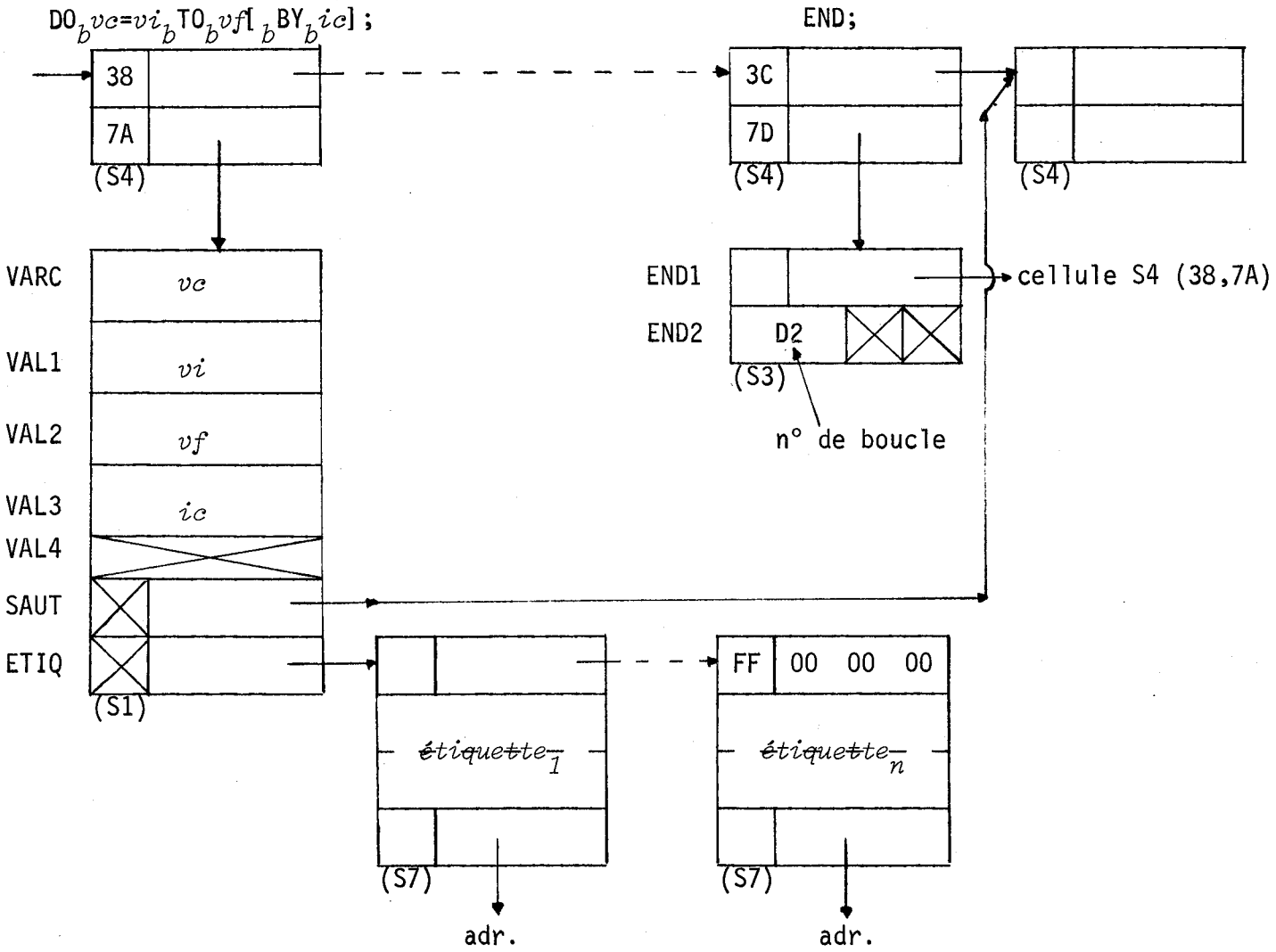
Nota : s'il n'y a pas de table d'étiquettes attachée à ce bloc, le deuxième pointeur de la cellule S4 attachée au noeud S4 (44,81) est NULL.

B) MARQUANT LA FIN D'UNE MACRO-COMMANDE DO ITERATIF

Le système procède comme précédemment en ce qui concerne les étiquettes du bloc.

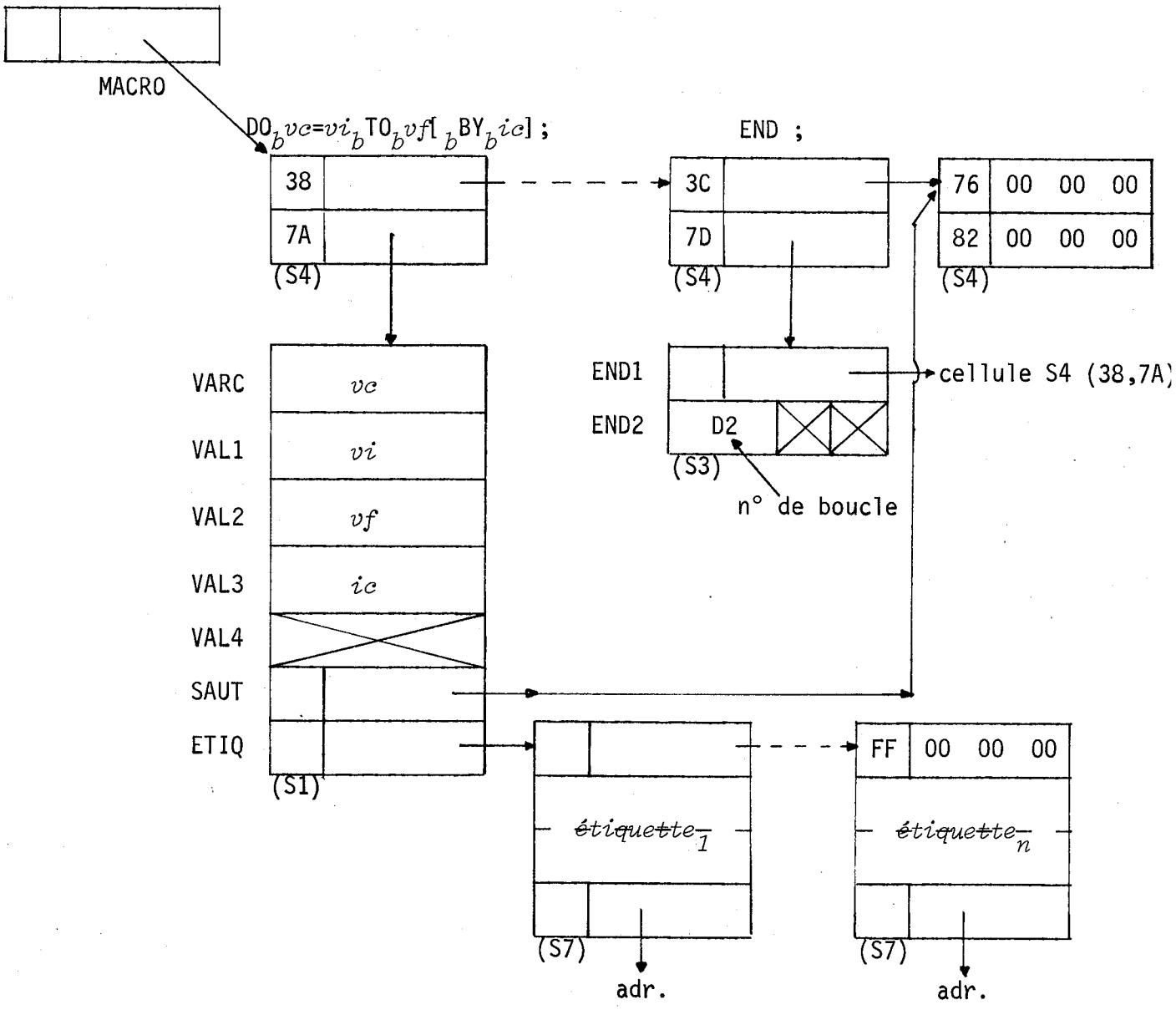
α) COMMANDE END MARQUANT LA FIN PHYSIQUE D'UN BLOC DE NIVEAU > 1

Il s'agit d'un bloc DO intératif intérieur à une unité de programme, on obtient alors :



β) COMMANDE END MARQUANT LA FIN PHYSIQUE D'UN BLOC DE NIVEAU = 1

Il s'agit du bloc le plus extérieur, constituant l'unité de programme, on obtient alors :

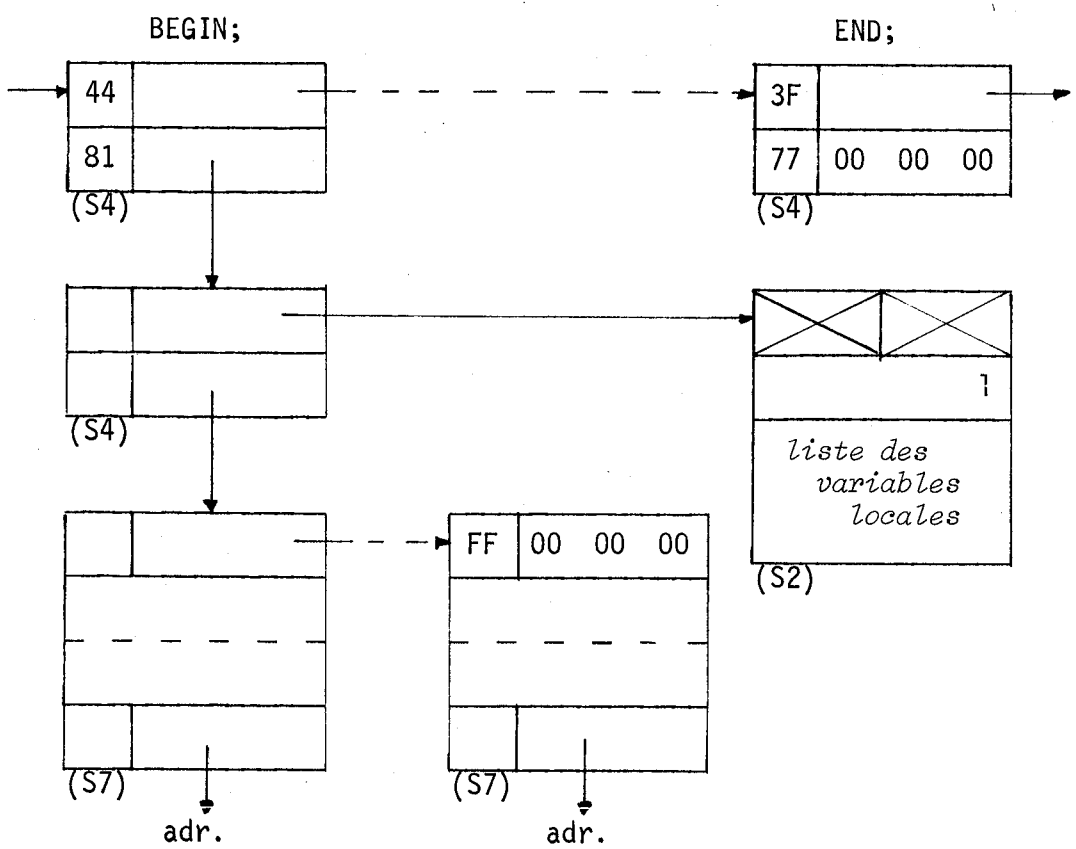


Nota : s'il n'y a pas de table d'étiquettes attachée à ce bloc, la zone ETIQ contient la valeur FF000000 (pointeur NULL)

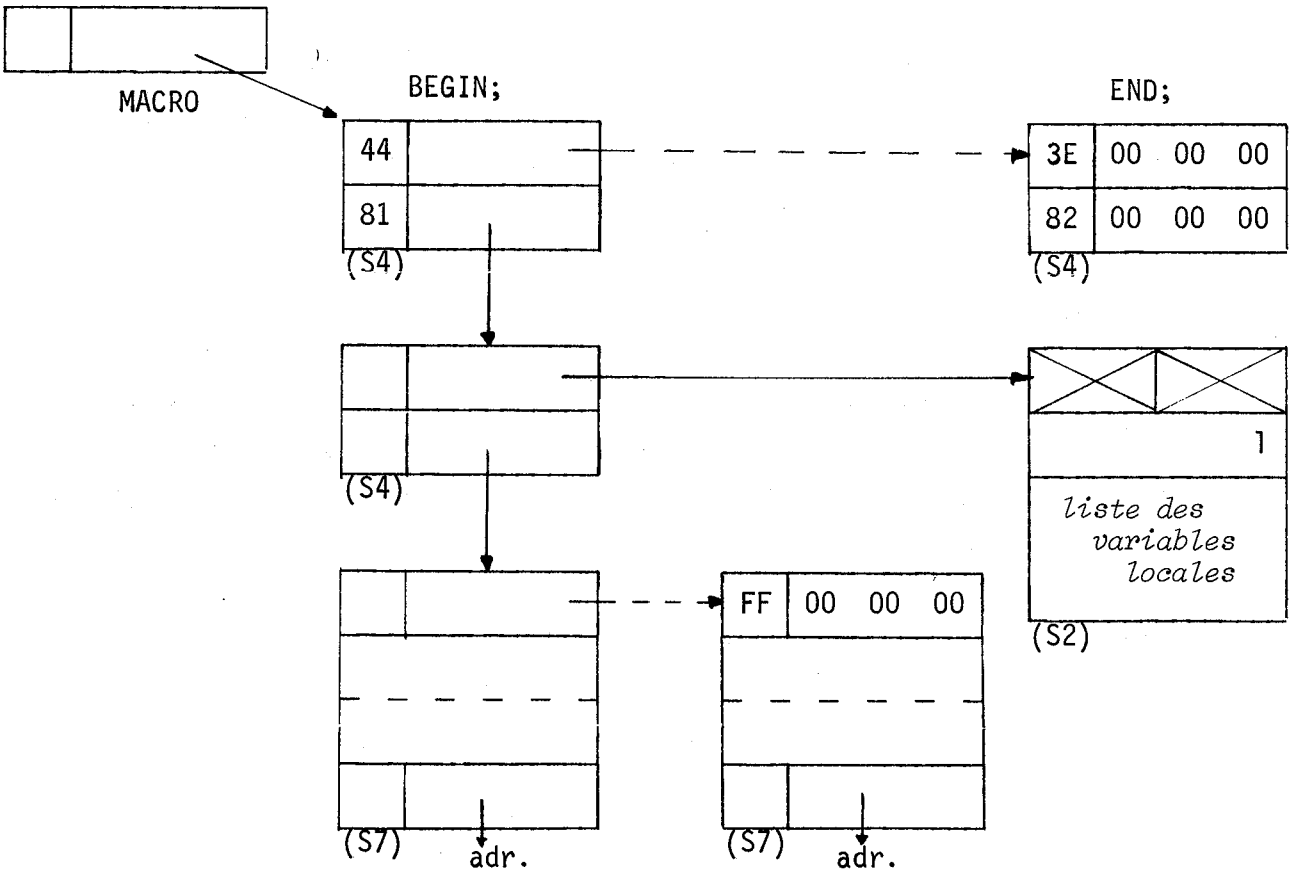
C) MARQUANT LA FIN D'UNE MACRO-COMMANDE BEGIN

D'une part le système traite les étiquettes comme précédemment, mais il peut y avoir, de plus, des variables locales. Celles-ci sont rangées dans une cellule S2 attachée à la BSA du bloc (cf. 3° Partie, § 7.21)

α) COMMANDE END MARQUANT LA FIN PHYSIQUE D'UN BLOC DE NIVEAU > 1

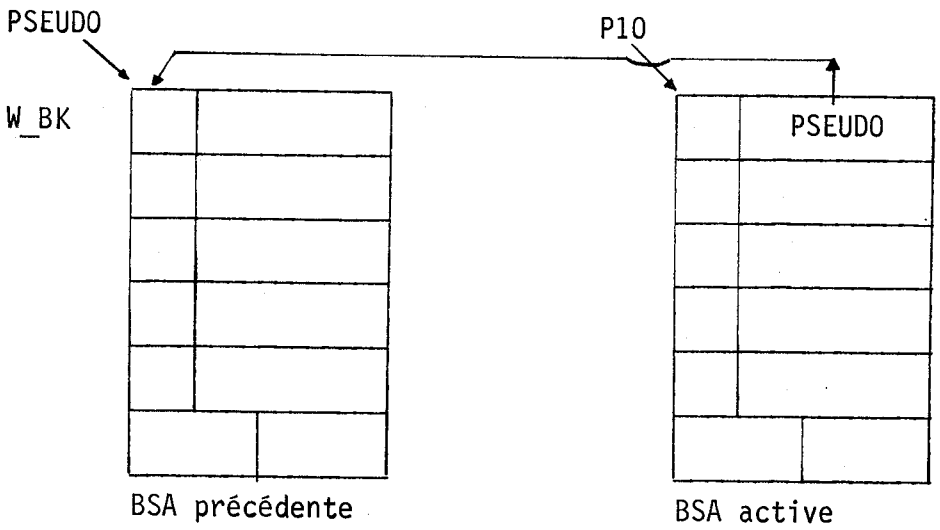


β) COMMANDE END MARQUANT LA FIN PHYSIQUE D'UN BLOC DE NIVEAU = 1



Nota : les pointeurs de la cellule S4 attachée au noeud S4 (44,81) n'ont de valeur que s'il existe des variables locales et/ou des étiquettes, sinon ils ont la valeur NULL (FF000000).

Remarques : - au moment de l'installation des cellules correspondant à la commande END des macro-commandes, le système récupère l'adresse de la BSA précédente et libère la BSA active.



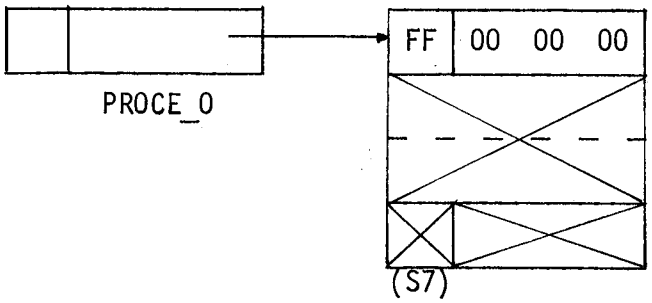
Les pointeurs P10 et PSEUDO sont chargés de l'adresse de la BSA précédente qui devient BSA active dans le cas d'une fin de bloc interne, sinon il n'y a pas de BSA précédente.

- s'il s'agit d'une macro-commande de niveau 1, le superviseur confie alors le contrôle à la séquence d'exécution en lui donnant l'adresse rangée dans le pointeur MACRO. (cf. 3° Partie, § 8)

7.23.. COMMANDE DE DECLARATION DE SOUS/PROGRAMME [§ 7]

Lors du chargement du système FORDECAL, et pour des raisons de simplicité, on alloue une cellule S7, accrochée au pointeur PROCE_0 , ancre de la table des points d'entrée. Cette cellule restera toujours vide et servira de marque de fin de table.

Ainsi au départ, on a la structure suivante :

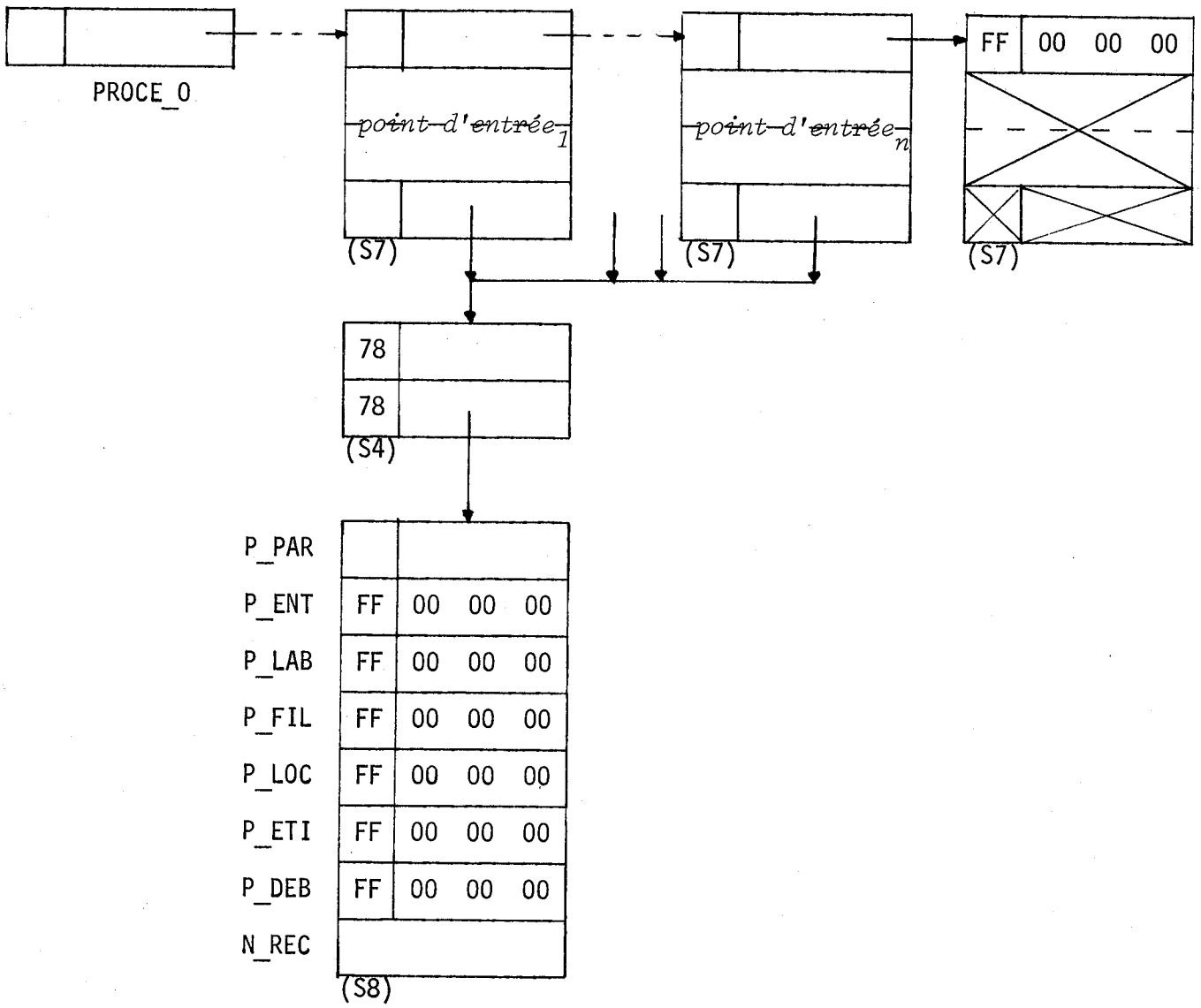


Le système rencontre alors la déclaration de sous-programme de

syntaxe : *p.e.*:[*p.e.*:..]PROCEDURE[(*liste paramètres*)][RECURSIVE][RETURNS] ;

Le système alloue autant de cellules de type S7 qu'il y a d'identificateurs de *point d'entrée* équivalents, chaque cellule contiendra en partie FORMAL l'identificateur du *point d'entrée* et en partie ENTADR l'adresse du noeud S4 alloué à la commande de déclaration, toutes les cellules S7 sont de plus chaînées entre elles par la partie FOLGEN. Durant le décodage de la liste des *points d'entrée* le système vérifie l'unicité de cet identificateur. S'il existe déjà l'utilisateur a la possibilité de le changer ou d'effacer l'ancien sous-programme en libérant ainsi l'identificateur de *point d'entrée*.

Une fois les *points d'entrée* décodés, le système alloue une cellule de type S8 qui contiendra divers renseignements concernant les paramètres, leur spécification, ... Finalement une fois la déclaration de sous-programme enregistrée on a la structure suivante :



P_PAR : pointeur sur les paramètres ou NULL,

P_ENT : initialisé à NULL, sera utilisé comme adresse des spécifications ENTRY,

P_LAB : initialisé à NULL, sera utilisé comme adresse des spécifications LABEL,

P_FIL : initialisé à NULL, sera utilisé comme adresse des spécifications FILE,

P_LOC : initialisé à NULL, sera utilisé comme adresse des variables locales de premier niveau,

P_ETI : initialisé à NULL, sera utilisé comme adresse de la table d'étiquettes "arrivée" de premier niveau,

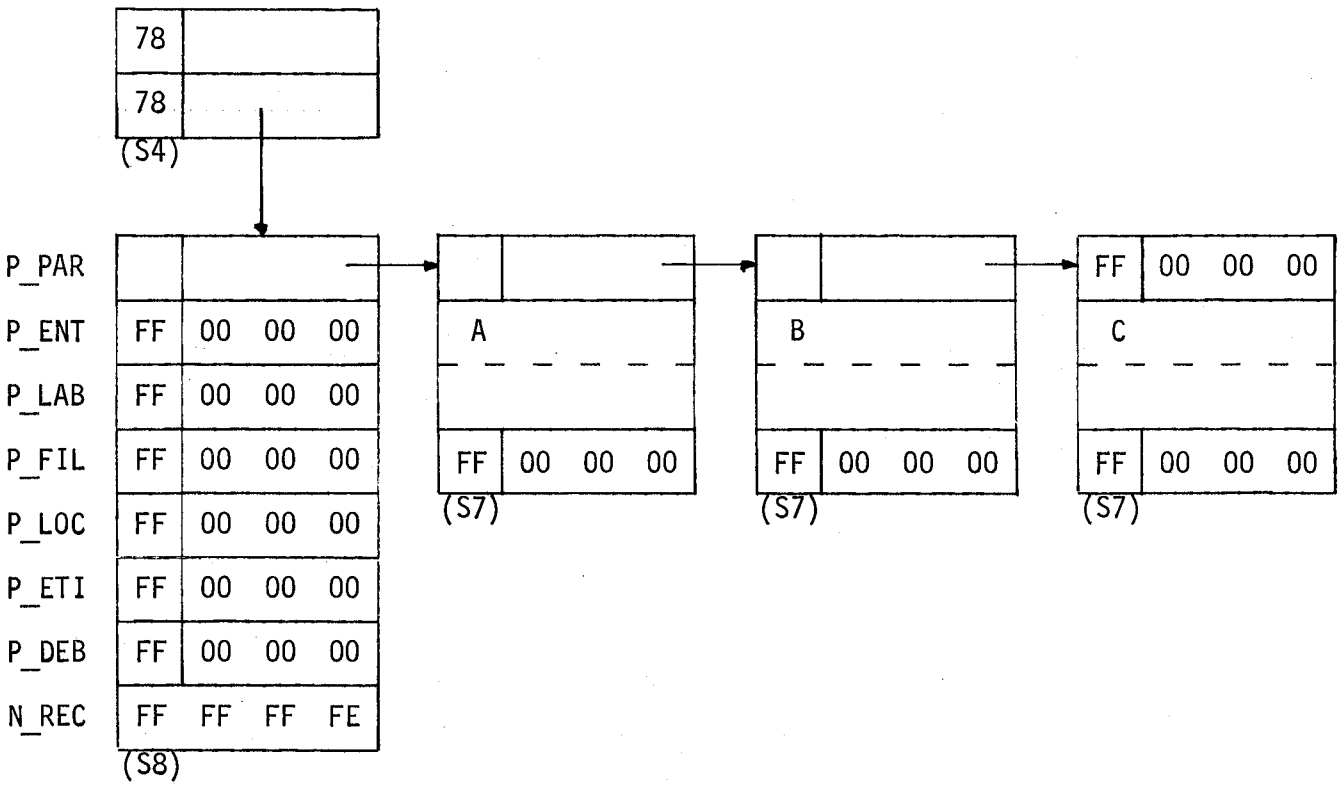
P_DEB : initialisé à NULL, sera utilisé par les pré-requêtes de l'accompagnateur,

N_REC : contient FFFFFFFE (-2) dans le cas d'un sous-programme non récursif
et 00000000 (0) dans le cas d'un sous-programme récursif.

S'il y a des paramètres, ceux-ci sont "logés", un à un et dans l'ordre d'apparition dans la *liste des paramètres*, dans des cellules de type S7. Ces cellules sont chaînées entre elles par leur partie FOLGEN et contiennent dans leur zone FORMAL l'identificateur du *paramètre*. L'adresse de la première cellule S7 est chargée dans la zone P_PAR de la cellule S8 attachée à ce sous-programme.

Exemple : soit la déclaration de procédure suivante :

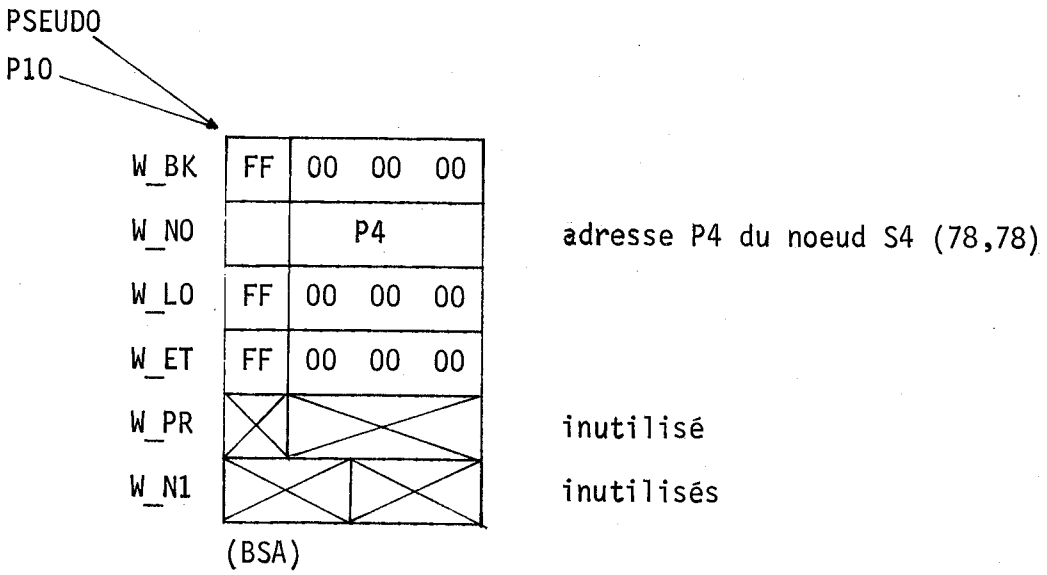
P1 : PROCEDURE (A , B , C) ; on obtient :



L'adresse du noeud S4 (78,78) est rangée en P4 et celle de la cellule S8 en P8.

Remarque : la zone ENTADR des cellules de type S7 contenant les paramètres est mise à NULL. Elle servira éventuellement, plus tard, pour les spécifications.

Une fois la déclaration de sous-programme enregistrée, le système alloue une BSA, comme dans le cas des macro-commandes.



7.24.. COMMANDES DE SPECIFICATION ENTRY, LABEL, FILE [§ 7.3., 7.4., 7.5.]

Le superviseur commence par vérifier que ces commandes sont sémantiquement autorisées (elles doivent se trouver au même niveau qu'une déclaration de sous-programme sinon elles sont ignorées).

Remarque : plusieurs commandes de spécification, de même type, peuvent ensuite appaître, elles seront prises en charge par le superviseur. Les redondances éventuelles sont signalées, message de code FDC002I, et ignorées, de même le système vérifie que l'identificateur de variable apparaîssant dans une commande de spécification fait bien partie de la liste des paramètres sinon le message de code FDC001I est imprimé.

Lorsque le superviseur a décodé la commande de spécification, il recherche dans la chaîne des paramètres la cellule S7 qui est attribuée à l'identi-

ficateur du paramètre correspondant, il alloue ensuite une cellule S7 qui est accrochée à la première et une cellule S4 dont l'adresse est rangée en partie P_spécification de la cellule S8 du sous-programme. Cette cellule S4 contiendra ensuite l'adresse de la cellule S7 contenant l'identificateur du paramètre.

Exemple : P1 : PROCEDURE (A , B , C , D , E , F) ;
LABEL A; FILE C; ENTRY B; LABEL D;

Lorsque ces cinq commandes auront été interprétées, il y aura en mémoire la structure représentée en page 50.

Les cellules S7 attachées aux cellules contenant A, B, C et D seront utilisées pour les arguments correspondant à ces paramètres. Le "chargement" des paramètres, lors de l'activation d'un sous-programme, est différent selon la spécification du paramètre : variable Formac ou variable spécifiée ENTRY, LABEL ou FILE.

Remarques : - en cours d'enregistrement du corps du sous-programme, on procède pour les déclarations de variables locales et pour les étiquettes "arrivée" de premier niveau de la même façon que pour les macro-commandes en utilisant les pointeurs W_LO et W_ET de la BSA attachée au sous-programme,

- notez le code de spécification dans l'exemple précédent :
  - 8F pour les spécifications ENTRY,
  - 90 pour les spécifications LABEL et
  - 91 pour les spécifications FILE.

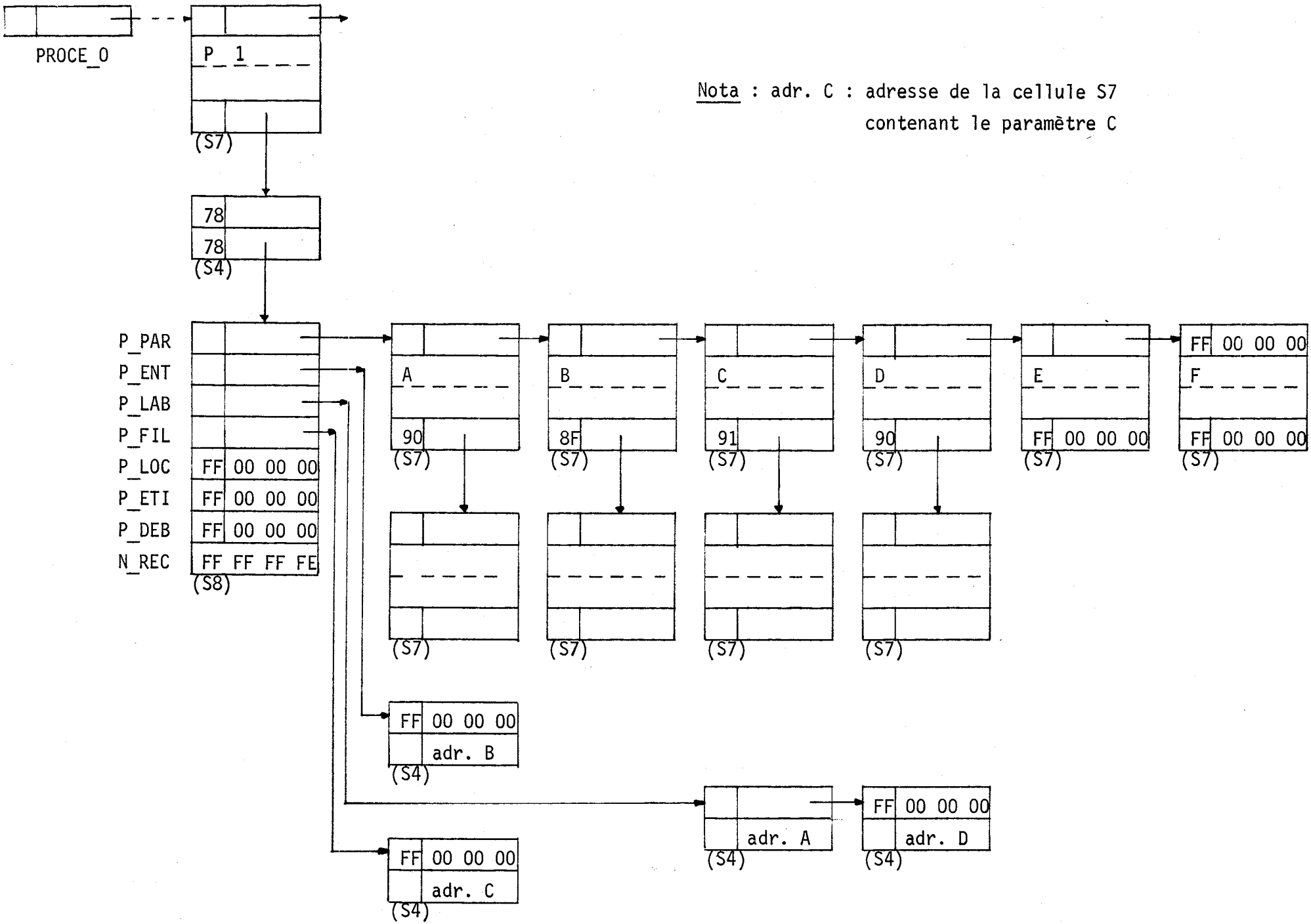
7.25.. COMMANDE CALL [§ 7.6.]

On procède comme au paragraphe 7.14

7.26.. COMMANDE SCRATCH [§ 7.7.]

On procède comme au paragraphe 7.15

7.27.. COMMANDE RETURN [§ 7.8.]

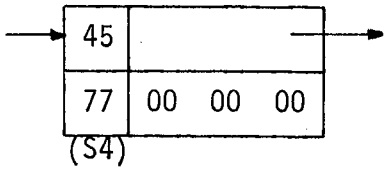


Nota : adr. C : adresse de la cellule S7
contenant le paramètre C

A) D'UNE PROCEDURE

syntaxe : RETURN ;

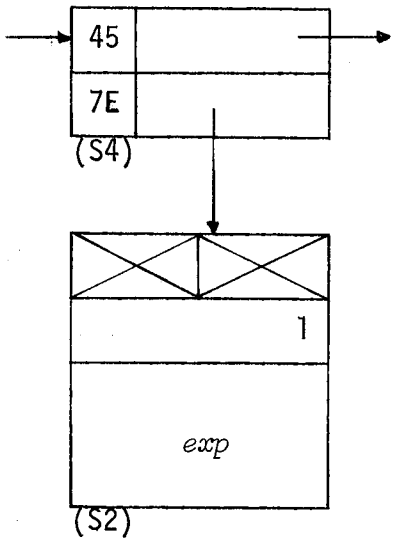
structure :



B) D'UNE FONCTION-PROCEDURE

syntaxe : RETURN$_b$*expression* ;

structure :



l : longueur de la zone *exp*

exp : *expression*

Remarque : lors de la décodification de la déclaration du sous-programme un marqueur est mis à 1, ou à 0, selon que le sous-programme est de type PROCEDURE ou FONCTION-PROCEDURE.

7.28.. COMMANDE END

S'il s'agit d'une commande END fermant un bloc interne au sous-programme se reporter au paragraphe 7.22.

S'il s'agit effectivement de la fin d'un sous-programme, le système commence par s'occuper de la table des étiquettes comme dans le cas d'une commande END d'une macro-commande (cf. § 7.22.) puis accroche, le cas échéant, la table des étiquettes "arrivée", de premier niveau, et les variables locales, de premier niveau, à la cellule S8 du sous-programme. (l'adresse de cette cellule a été conservée dans le pointeur P8). On obtient alors la structure suivante :

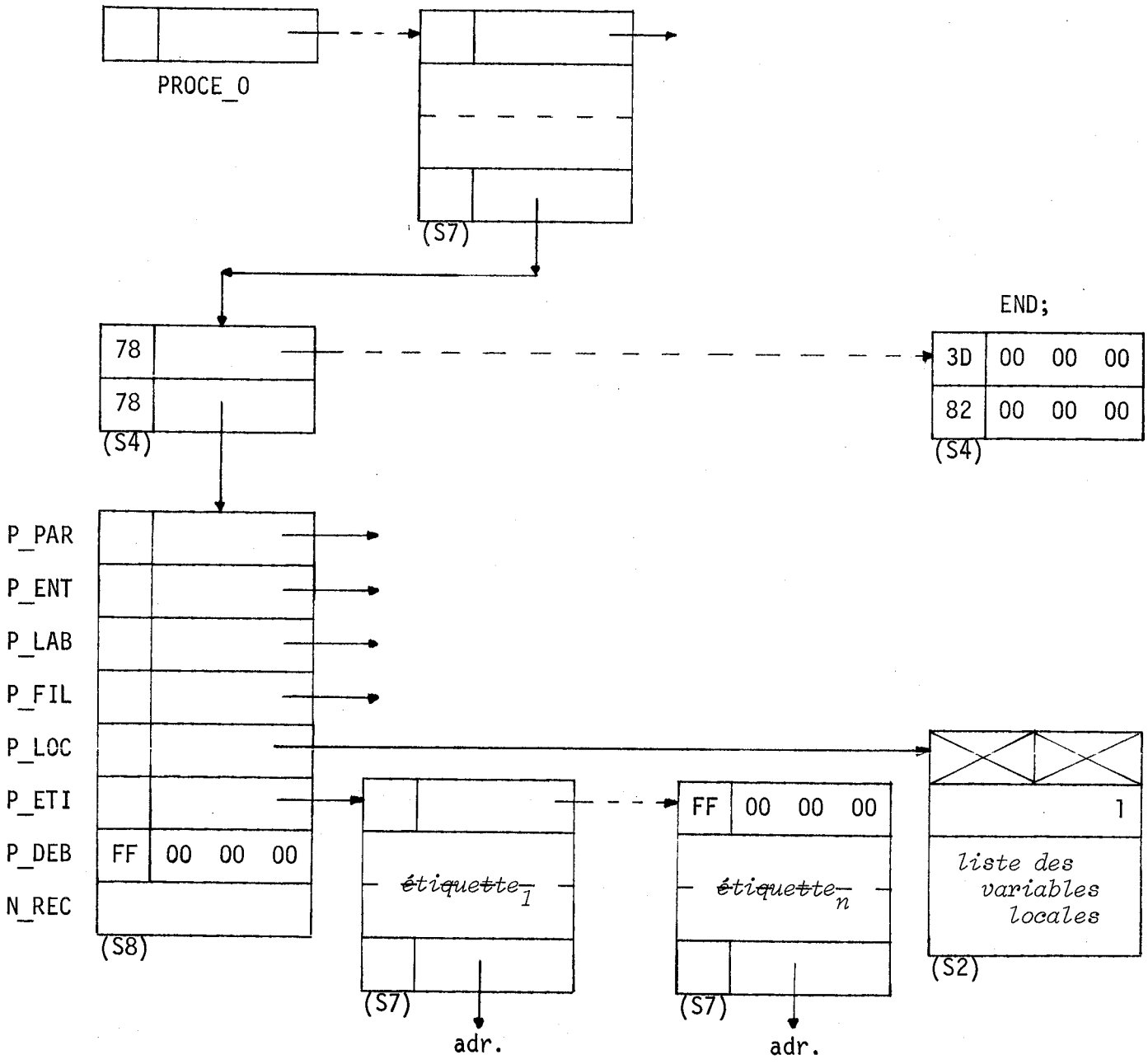


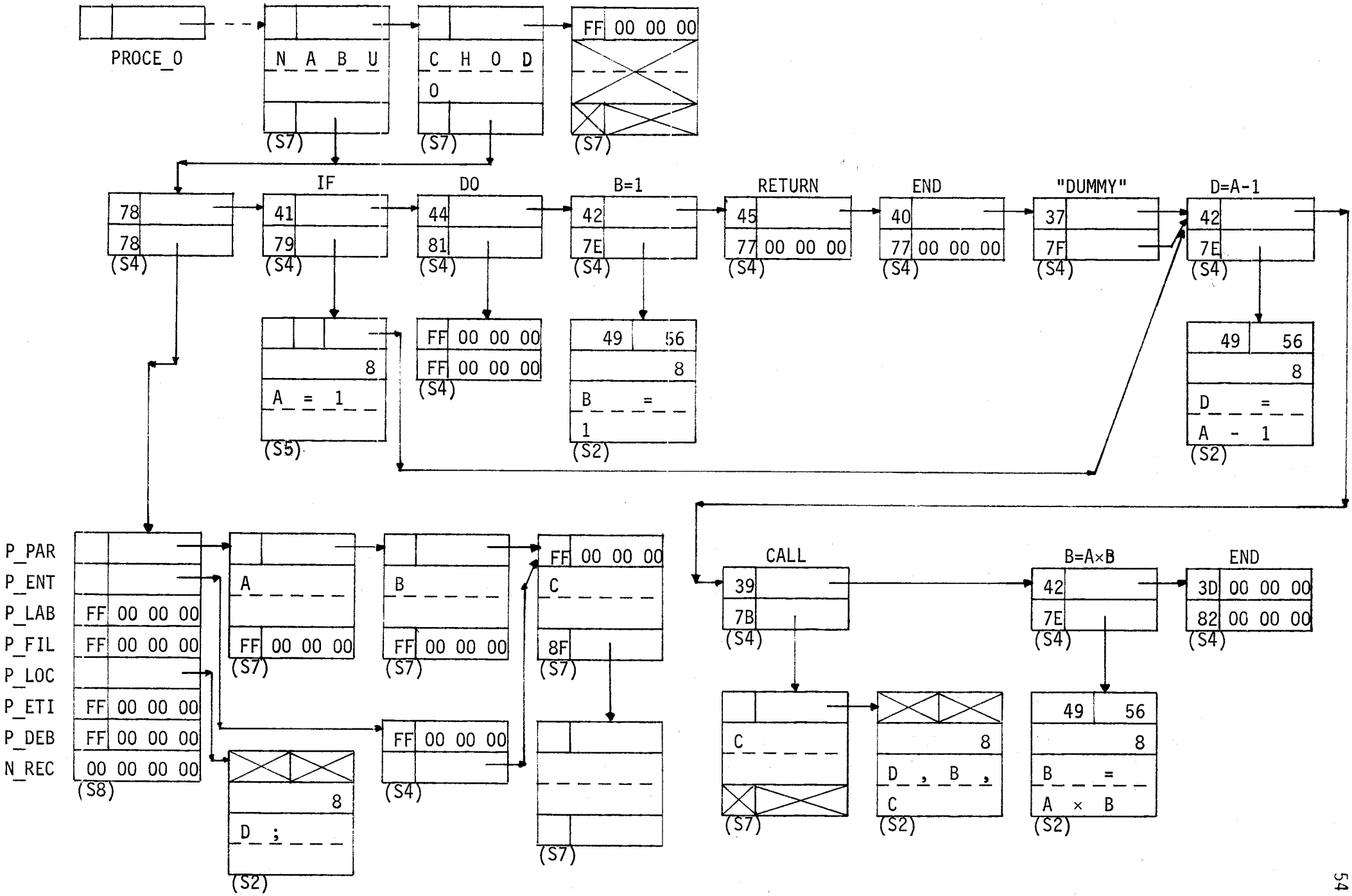
Table des étiquettes de premier niveau

La BSA du sous-programme est alors libérée et le contrôle rendu, à haut niveau, à l'utilisateur.

Exemple :

```
NABU : CHODO : PROCEDURE ( A , B , C ) RECURSIVE ;
               ENTRY C; LOCAL D;
               IF A=1 THEN DO;
                              B = 1;
                              RETURN;
                           END;
               D = A-1;
               CALL C(D,B,C);
               B = A×B;
END CHODO;
```

on obtient la structure décrite en page suivante.


PROCE_0
N A B U
C H O D
0
FF 00 00 00
(S7)
(S7)
(S7)
IF
DO
B=1
RETURN
END
"DUMMY"
D=A-1
78
78
(S4)
41
79
(S4)
44
81
(S4)
42
7E
(S4)
45
77 00 00 00
(S4)
40
77 00 00 00
(S4)
37
7F
(S4)
42
7E
(S4)
8
A = 1
(S5)
FF 00 00 00
FF 00 00 00
(S4)
49 56
8
B =
1
(S2)
49 56
8
D =
A - 1
(S2)
P_PAR
P_ENT
P_LAB
P_FIL
P_LOC
P_ETI
P_DEB
N_REC
FF 00 00 00
FF 00 00 00
FF 00 00 00
FF 00 00 00
00 00 00 00
(S8)
A
FF 00 00 00
(S7)
B
FF 00 00 00
(S7)
FF 00 00 00
C
8F
(S7)
FF 00 00 00
(S4)
8
D ; _
(S2)
(S7)
CALL
39
7B
(S4)
B=A×B
42
7E
(S4)
END
3D 00 00 00
82 00 00 00
(S4)
C
(S7)
8
D , B ,
C
(S2)
49 56
8
B =
A × B
(S2)

7.29.. L'ACCOMPAGNATEUR - LES PRE-REQUETES [§ 8., 8.1., 8.2.]

Rappelons tout d'abord que l'accompagnateur est appelé par la commande OPTSET DEBUG ou NODEBUG. Le système demande ensuite le nom du sous-programme à qualifier. Une recherche s'effectue alors dans la table des points d'entrée pour trouver l'adresse de ce sous-programme. Si ce dernier n'existe pas, le message de code FDC002D est imprimé.

Une fois l'adresse du premier noeud du sous-programme récupéré, le système retrouve l'adresse de la cellule de type S8 qui lui est attachée.

A) OPTSET DEBUG

Si la zone P_DEB de la cellule S8 est NULL, le système alloue une cellule de type S3 dont les zones TRACE et PRINT sont mises à '0', celà correspond aux pré-requètes par défaut NOTRACE et NOPRINT.

Si les pré-requètes suivantes sont TRACE et/ou PRINT, les zones correspondantes de la cellule S3 précédente sont mises à '1'.

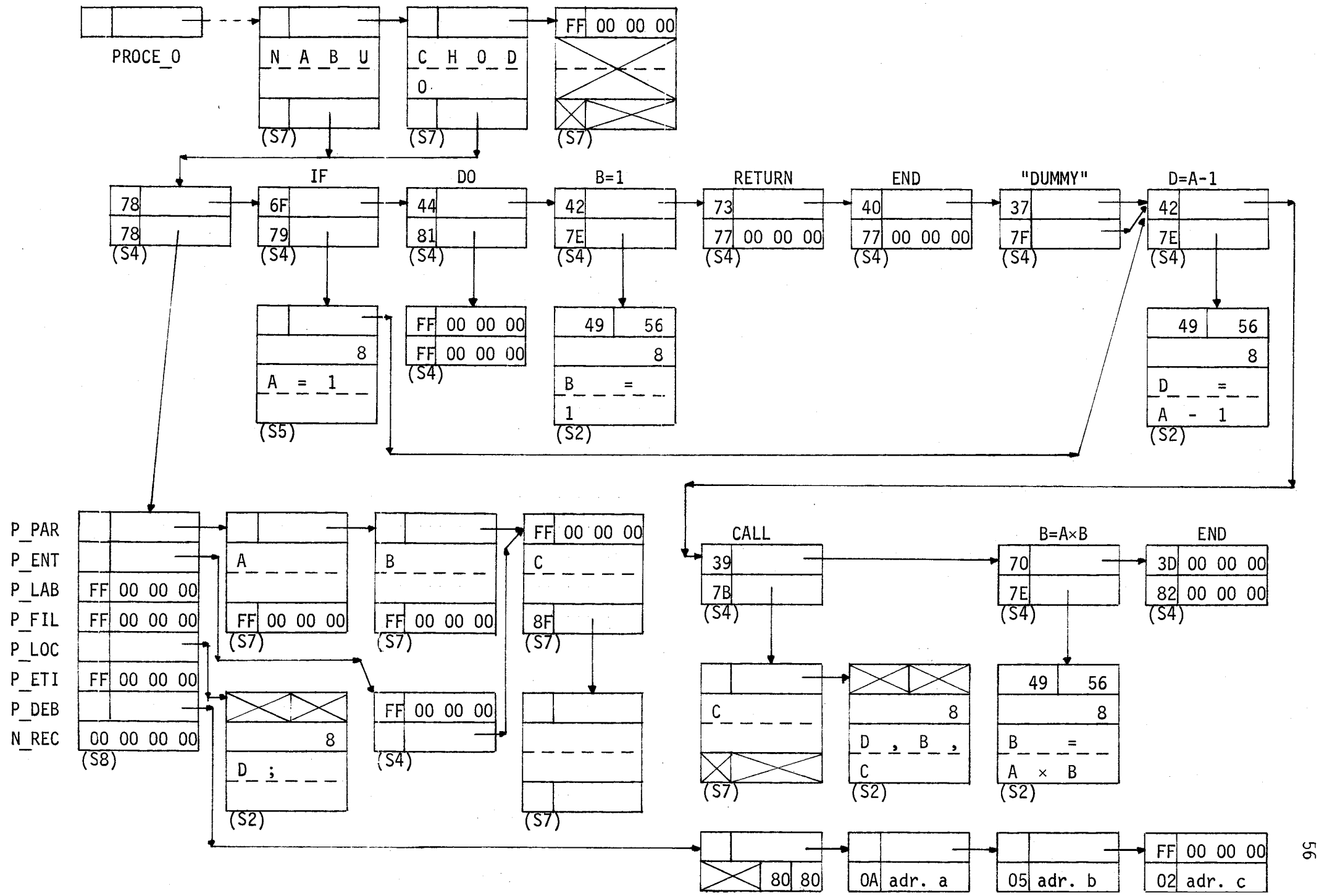
Si les pré-requètes suivantes sont BREAK AT xxx, le système commence par rechercher l'adresse du noeud de la $x^{ième}$ commande, si le nombre xxx est supérieur au nombre de noeuds du sous-programme le message de code FDC007D est émis. Si la $x^{ième}$ commande existe, une cellule de type S4 est allouée. On range dans le deuxième mot de cette cellule le numéro xxx et l'adresse du noeud de la commande sur laquelle un point d'arrêt est posé, de plus la partie CODE1 de la $x^{ième}$ commande est modifié : on lui ajoute 2E.

Prenons l'exemple précédent, page 53, et supposons que les pré-requètes suivantes ont été émises :

TRACE, PRINT, BREAK AT 2, BREAK AT 5, BREAK AT 10

on obtient alors la structure :

Nota : adr. a : adresse du noeud correspondant à la commande B=A×B,
adr. b : adresse du noeud correspondant à la commande RETURN,
adr. c : adresse du noeud correspondant à la commande IF


PROCE_O
N A B U
C H O D
0
FF 00 00 00
(S7)
(S7)
(S7)
IF
DO
B=1
RETURN
END
"DUMMY"
D=A-1
78
78
(S4)
6F
79
(S4)
44
81
(S4)
42
7E
(S4)
73
77 00 00 00
(S4)
40
77 00 00 00
(S4)
37
7F
(S4)
42
7E
(S4)
8
A = 1
(S5)
FF 00 00 00
FF 00 00 00
(S4)
49 56
8
B =
1
(S2)
49 56
8
D =
A - 1
(S2)
P_PAR
P_ENT
P_LAB
P_FIL
P_LOC
P_ETI
P_DEB
N_REC
FF 00 00 00
FF 00 00 00
FF 00 00 00
00 00 00 00
(S8)
A
FF 00 00 00
(S7)
B
FF 00 00 00
(S7)
FF 00 00 00
C
8F
(S7)
8
D ;
(S2)
FF 00 00 00
(S4)
(S7)
CALL
B=A×B
END
39
7B
(S4)
70
7E
(S4)
3D 00 00 00
82 00 00 00
(S4)
C
(S7)
8
D , B ,
C
(S2)
49 56
8
B =
A × B
(S2)
80 80
0A adr. a
05 adr. b
FF 00 00 00
02 adr. c

B) OPTSET NODEBUG

Si le sous-programme existe et est qualifié, le système recherche, à partir du pointeur P_DEB de la cellule S8, les adresses des points d'arrêt, modifie la partie CODE1 du noeud de la commande, libère tou tes les cellules S4 et la cellule S3 allouées aux pré-requètes et remet le pointeur P_DEB à NULL.

8. EXECUTION DES COMMANDES ENREGISTREES

8.1.. MISE EN ACTIVITE D'UNE UNITE DE PROGRAMME

A) CAS D'UNE MACRO-COMMANDE DE PREMIER NIVEAU

Nous avons vu qu'aprés le décodage de la commande END marquant la fin physique de cette macro-commande, le superviseur se branche sur la séquence de traitement, avec dans le pointeur P0 l'adresse du train, rangée précédemment dans MACRO. S'il s'agit d'une macro-commande BEGIN on passe d'abord par la séquence de traitement des variables locales.

B) ACTIVATION D'UNE PROCEDURE AU PREMIER NIVEAU

L'utilisateur a donc composé la commande CALL *procname* [(*l*. *a*.)];
Le superviseur commence par rechercher dans la table des points d'entrée dont l'adresse est rangée dans le pointeur PROCE_0, si l'identificateur *procname* y figure. Si oui, on trouve en partie ENTADR de la cellule S7 correspondante l'adresse de la procédure. Soit P0 cette adresse.

Le superviseur incrémente de 1 la partie N_REC de la cellule S8, puis s'occupe de la *liste des arguments*. A partir du pointeur P_PAR de la cellule S8 il est facile de vérifier que le nombre de paramètres et d'arguments est identique, sinon on se branche sur la séquence de traitement des erreurs en phase d'exécution (message de code FDC020G, cf. 2° partie, § 12.2.C.).

Si le paramètre a une spécification, premier octet du 4° mot de la cellule S7 correspondant à ce paramètre, on charge dans la 2° cellule S7 l'argument.

Si le paramètre n'a pas de spécification et selon que l'argument est une expression ou un identificateur, indicé ou non, et que l'appel soit fait par valeur ou par nom, le superviseur se branche sur les séquences d'appel des routines :

- DENFMCL : l'argument est une expression,
- DENFMCK : appel par nom,
- DENFMCM : appel par valeur.

- DENFMCL(paramètre, expression) :
  - cette routine appelle DENINS avec comme argument la chaîne : '$SYSTEM1 = expression' ce qui a pour but, d'une part d'évaluer expression et d'autre part de l'affecter à la variable $SYSTEM1, installée par la routine DENCCSO,
  - ensuite DENFMCL installe, dans la liste des repwords, un repword dont la zone NAME contiendra les caractères constituant l'identificateur du paramètre, avec ORD = 80, et dont le pointeur DOWN sera chargée de l'adresse rangée en partie DOWN du repword affecté à $SYSTEM1, finalement la zone ACT sera mise à 30.

- DENFMCK(paramètre, argument) :
  - cette routine commence par chercher dans la liste des repwords l'adresse de "argument", puis installe un repword de zone NAME contenant les caractères constituant l'identificateur du paramètre, avec ORD = 81 et dont le pointeur DOWN est chargé de l'adresse du repword-argument.

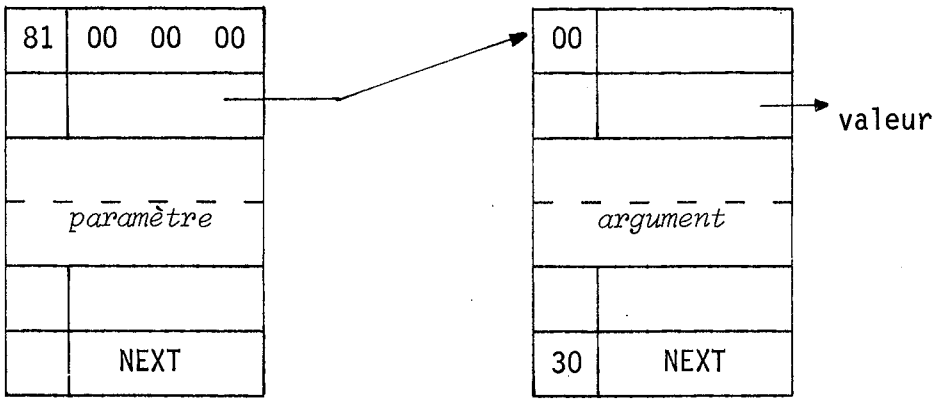
- DENFMCM(paramètre, argument) :
  - cette routine recherche d'abord l'adresse, dans la liste des repwords, de l'argument, crée un NODE CS (Common Subexpression) et un HEADER en le faisant pointer sur la valeur de l'argument. Elle installe ensuite un repword correspondant au paramètre et

charge dans sa partie DOWN l'adresse du CS. La partie ORD de ce dernier repword est mise à 80 et ACT à 30. Dans le cas où l'argument est la racine d'un tableau, il faut créer autant de repwords qu'il y a d'indices pour l'argument en utilisant la même technique que pour la racine en ce qui concerne les valeurs.

Remarques : - le code 80 ou 81 de la partie ORD des repwords indique à la routine DENINS s'il s'agit d'un adressage direct ou non,

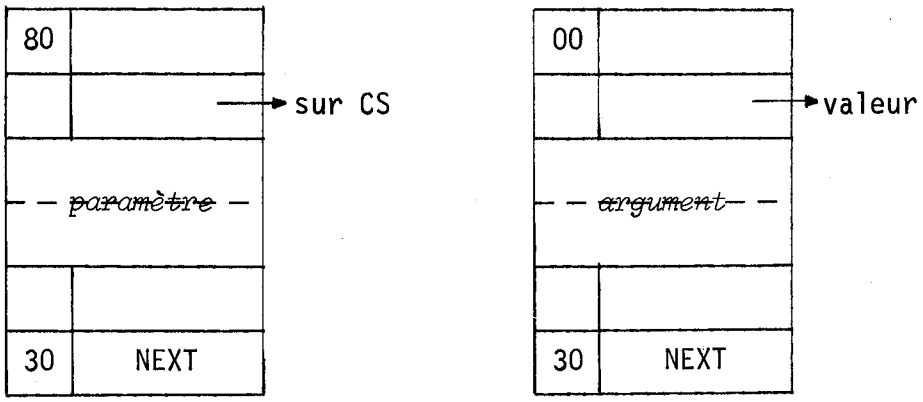
- l'utilisateur ne peut "effacer" ces repwords et leur valeur (commande REMOB), seul le système peut le faire.

- dans le cas d'un appel à DENFMCK on obtient, dans le list processing de FORMAC :

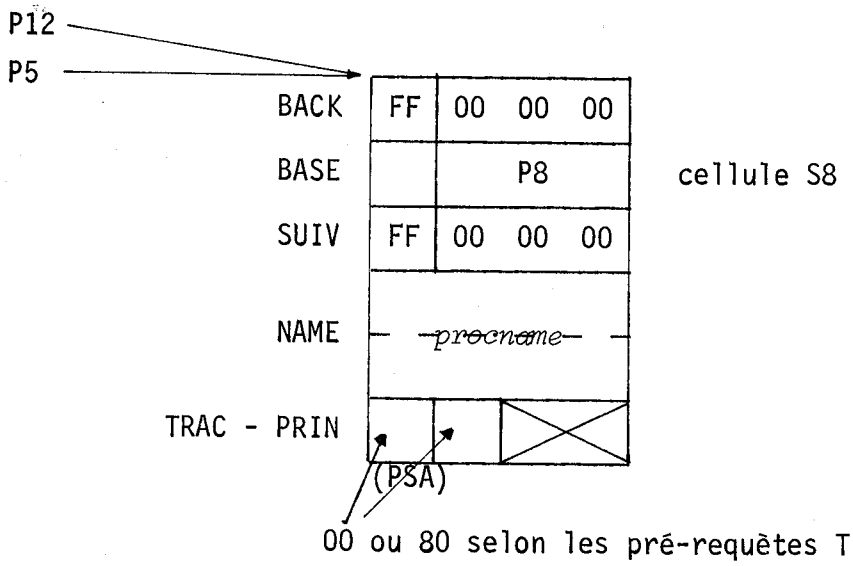


La routine DENINS, qui installe, dans FORMAC, les expressions, a été modifiée en conséquence.

- dans le cas d'un appel à DENFMCM on obtient, dans le list processing de FORMAC :



Une fois le chargement des paramètres effectué, le superviseur alloue une PSA qui servira de zone de sauvegarde pour la procédure en activité :



Le système charge alors dans P0 l'adresse du deuxième noeud de la procédure et se branche sur la séquence de traitement des variables locales.

C) ACTIVATION D'UNE FONCTION-PROCEDURE AU PREMIER NIVEAU

L'utilisateur a composé une commande d'affectation dans laquelle figure en partie droite une référence à une fonction-procédure. La routine DENINS a aussi été modifiée de façon à pouvoir activer une fonction-procédure écrite par l'utilisateur. Pour ce faire on transite par la routine DENFPRO qui se charge d'effectuer les sauvegardes nécessaires du coté FORMAC (les PDL, divers marqueurs utilisés par DENINS,...) et d'activer la fonction-procédure. On retombe alors dans le cas d'un appel à une procédure en ce qui concerne la phase d'activation et de désactivation. La désactivation, ici, se fera sur interprétation d'une commande RETURN *expression*. Aprés la phase classique de désactivation des procédures, le superviseur fera appel à la routine DENFPRO qui se charge de mettre à jour les PDL pour DENINS et rend le contrôle à cette dernière routine.

8.2.. SEQUENCE DE TRAITEMENT DES VARIABLES LOCALES

Lors du traitement du premier noeud d'une commande BEGIN ou d'un sous-programme, on a chargé dans le pointeur P11 l'adresse de la cellule S2 qui contient, si elle existe, la liste des variables locales.

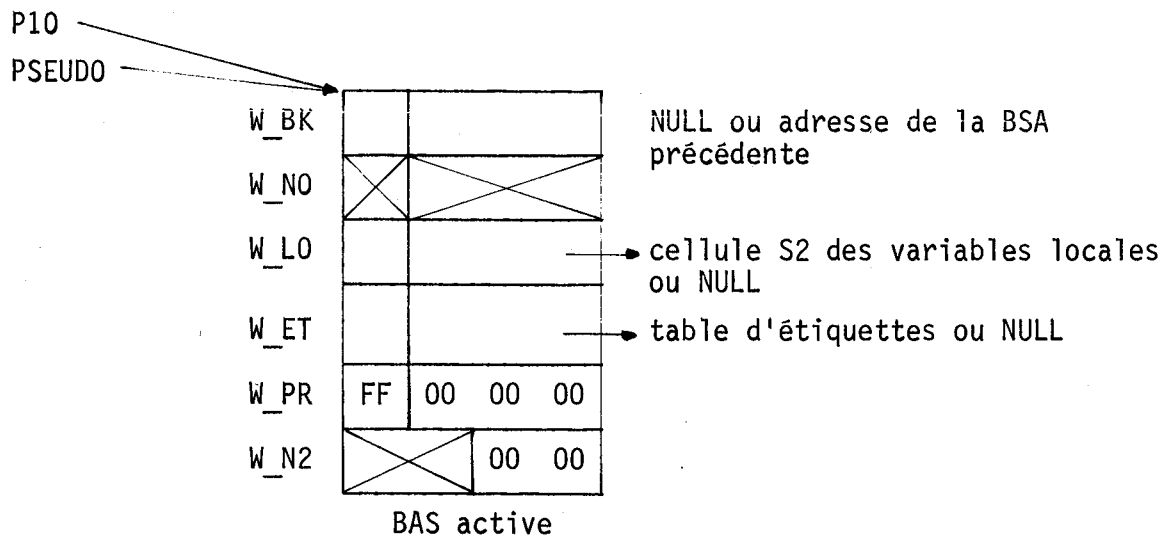
On fait alors appel, dans le cas où ces variables locales existent, à la routine DENFMCN avec comme argument, sous forme de chaîne, la liste des variables locales récupérées à partir de la zone ST de la cellule S2 qui les contenait (cf. § 7.21.). Cette routine, ajoutée à la bibliothèque Formac, se charge d'installer, dans la liste des repwords, un repword pour chaque variable apparaîssant dans la liste avec ORD = 80 et ACT = 10 (code Atome).

Rappelons que seul le système peut "remober" ces variables, ce qui sera fait lors de la désactivation du bloc où elles ont été déclarées.

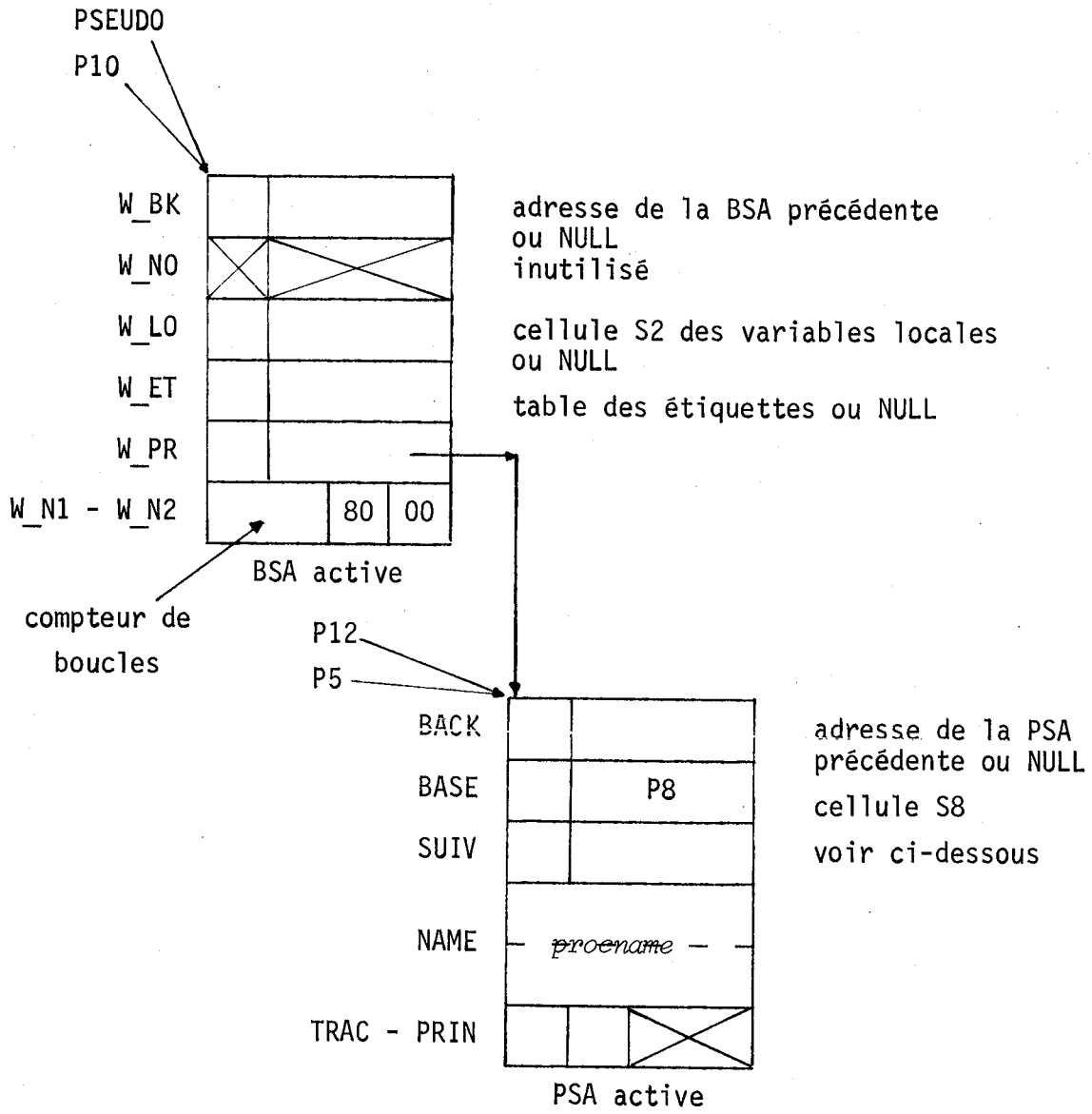
8.3.. SEQUENCE DE "DEMARRAGE"

Que l'on active une unité de programme au premier niveau ou non, le système commence par allouer une BSA qui servira de sauvegarde pour ce niveau de bloc actif.

A) CAS D'UNE MACRO-COMMANDE



B) CAS D'UN SOUS-PROGRAMME

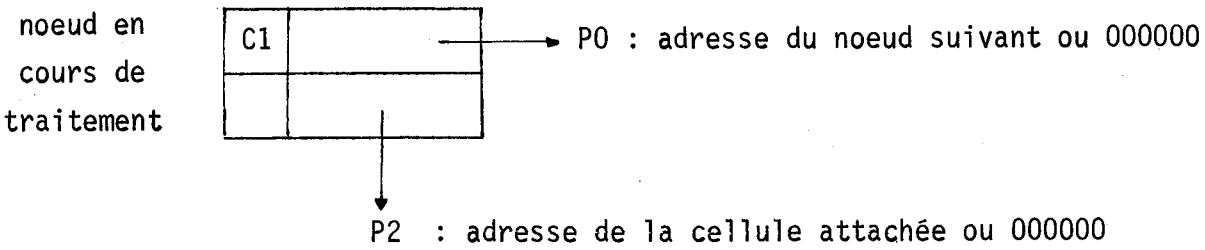


- TRAC et/ou PRIN ont pour valeur 00 ou 80 selon les pré-requêtes TRACE et PRINT de l'accompagnateur,
- le pointeur SUIV est soit NULL, cas d'un appel au premier niveau, soit contient l'adresse du noeud suivant l'appel dans le cas d'une référence faite à un niveau supérieur.

Ensuite le superviseur se branche sur la séquence "répartition du trabail".

8.4.. SEQUENCE DE REPARTITION D'ADRESSE 36

On passe par cette séquence pour le traitement de tous les noeuds du train, à sa sortie le système se trouve dans l'état suivant :



En récupérant, grâce à PLX1 et PLX2, les pointeurs P0 et P2 rangés en zones SUIV1 et SUIV2 du noeud S4, et le code particulier de la commande, partie C1 du noeud S4, le superviseur va se brancher sur la séquence de traitement attachée à cette commande et d'"adresse" C1.

8.5.. TRAITEMENT DES NOEUDS A CODE 42

A) GENERALITES

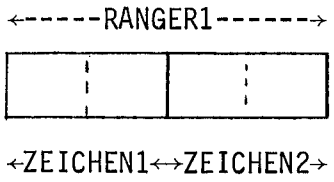
Ce code correspond aux commandes suivantes :

- affectation avec impression,
- affectation sans impression,
- ATOMIZE,
- REMOB,
- OPTSET,
- LIST,
- IDENT,
- EQUIV,
- LOP,
- NARGS,
- GET,
- PUT

La structure correspondant à chacune de ces commandes comprend une cellule de type S2 qui va, en fait, contenir les éléments particuliers de la commande ainsi que nous l'avons vu précédemment.

Le superviseur, grâce au pointeur P2 chargé au § 8.4, va ranger dans la variable d'identificateur RANGER1, de type FIXED BINARY(31), le contenu de la partie CO de la cellule S2 et dans la variable d'identificateur INPUT, de type CHARACTER, le contenu de la zone ST.

Nota : on a défini, dans le système, deux variables d'identificateurs ZEICHEN1 et ZEICHEN2, de type FIXED BINARY(15), définies sur RANGER1, respectivement sur le 1° et le 3° octet :



Or, en général, les cellules S2 attachées aux commandes susnommées, possèdent 2 codes dans la partie CO de la cellule.

La partie ZEICHEN1 permet un branchement sur la séquence d'adresse 49 qui est chargée de prendre élément par élément, les constituants rangés dans INPUT puis de donner le contrôle aux séquences d'adresse ZEICHEN2 qui correspondent alors au traitement effectif, et particulier, de la commande.

B) TRAITEMENT EFFECTIF

- SEQUENCE D'ADRESSE 4D (LOP)

utilise la routine Formac DENFMCB pour trouver l'opérateur principal d'une expression.

- SEQUENCE D'ADRESSE 4E (LIST)

selon l'argument on va faire appel à :

- la routine DENFMC5. Cette routine, ajoutée à la bibliothèque Formac, permet de lister les variables Formac;

- la table des points d'entrée pour lister les identificateurs de sous-programmes,
- la table des options en activité pour lister les options en cours;
- la routine DENCORE, ajoutée au système. DENCORE permet de connaître l'espace mémoire utilisé par le list-processing de Formac et du système.

- SEQUENCE D'ADRESSE 4F (NARGS)

utilise la routine Formac DENFMCC pour trouver le nombre d'arguments d'une expression, par rapport à son opérateur principal.

- SEQUENCE D'ADRESSE 50 (OPTSET)

commence par rechercher s'il y a un *nom de fichier*, si ce nom est un paramètre, dans le cas d'une commande interne à un sous-programme, et dans l'affirmative si ce paramètre a la spécification FILE. Récupère alors l'argument correspondant. Puis, à l'aide d'une recherche dichotomique dans la table des options, recherche un profil binaire, sauf pour les options ERASE, PRINTF, REWIND, SAVEFILE, DEBUG et NODEBUG. Si l'option existe, aprés quelques vérifications dans le cas d'une option d'E/S, fait appel à la routine DENFMC3, appartenant à la bibliothèque Formac mais légèrement modifiée, pour passer au système Formac le profil binaire précédemment trouvé. Dans le cas des options ERASE, PRINTF, REWIND et SAVEFILE, on fait appel, par l'intermédiaire de la routine GETSTRG, aux routines correspondantes de CMS.

- SEQUENCE D'ADRESSE 51 (ATOMIZE)

utilise la routine Formac DENFMC8, légèrement modifiée.

- SEQUENCE D'ADRESSE 52 (EQUIV - IDENT)

section commune au traitement d'IDENT et d'EQUIV. Séparation des deux arguments de ces commandes.

- SEQUENCE D'ADRESSE 53 (EQUIV)

fait appel à la routine DENFMCF, ajoutée à la bibliothèque. On simule l'option EXPAND et le système compare les expressions.

- SEQUENCE D'ADRESSE 54 (IDENT)

fait appel à la routine DENFMCG pour comparer les expressions.

- SEQUENCE D'ADRESSE 55 (affectation avec impression)

passe, aprés certaines initialisations, sur une séquence qui recherche la présence éventuelle de LOP ou de NARGS en partie droite du signe <=> Si oui, interprète la macro Formac LOP ou NARGS, puis fait appel à la routine Formac DENFMC2.

- SEQUENCE D'ADRESSE 56 (affectation sans impression)

démarre comme la séquence d'adresse 55, puis teste si le marqueur de la pré-requête PRINT est en fonction. Si oui fait appel à la routine DENFMC2, sinon à la routine DENFMC1.

Remarque : à la sortie de ces diverses séquences, le contrôle est transféré soit à la séquence d'adresse 49 dans le cas où la liste des arguments de la commande n'est pas épuisée, soit à la séquence de répartition d'adresse 36. (cf. § 8.4.)

- SEQUENCE D'ADRESSE 46 (GET)

commence par rechercher le *nom de fichier* sur lequel doit se faire la lecture des données. S'il s'agit d'un paramètre, vérifie qu'il y a spécification FILE, puis fait appel à la routine GETDATA, ajoutée au système, pour la lecture effective. A noter que le *nom de fichier* est stocké à l'entrée de cette séquence puis déstocké à la fin.

- SEQUENCE D'ADRESSE 4B (REMOB)

fait appel à la routine DENFMCP, ajoutée à la bibliothèque Formac. Cette routine commence par vérifier si l'argument peut être effacé et dans l'affirmative retourne à la liste libre du système Formac l'espace mémoire occupé par le repword et l'expression qui lui est affectée.

- SEQUENCE D'ADRESSE 4C (PUT)

utilise la routine PRINT1, ajoutée à la bibliothèque, pour l'impression du *message* sur le fichier de sortie en activité.

8.6.. TRAITEMENT DU NOEUD A CODE 39 (CALL)

Le système commence par récupérer l'identificateur de procédure qui figure dans la cellule de type S7 attachée au noeud.

Si le pointeur courant de PSA est NULL (pointeur P12) c'est qu'aucune procédure, ou fonction-procédure, n'est active, donc le *procname* n'est pas un paramètre de sous-programme. On se branche alors sur la séquence de recherche dans la table des points d'entrée et le traitement se poursuit comme au paragraphe 8.3.B.

Si le pointeur courant de PSA n'est pas NULL, on recherche dans la liste des paramètres du sous-programme actif si l'identificateur *procname* y figure et dans l'affirmative si le code spécification du paramètre correspondant est bien à 8F, sinon le superviseur se branche sur la séquence de traitement des erreurs en phase d'exécution (message de code FDC022G).

Le procname effectif trouvé, le système recherche dans la table des points d'entrée si cette procédure existe. Dans la négative le message de code FDC001G est imprimé mais l'exécution de l'unité de programme n'est pas stoppée. Dans l'affirmative le système commence par allouer une BSA, chaînée à celle du bloc précédent puis une PSA, et on se retrouve dans l'état décrit au paragraphe 8.3.(cf. aussi § 8.1)

8.7.. TRAITEMENT DU NOEUD A CODE 3A (SCRATCH)

La recherche du *procname* effectif correspondant à la partie SPR de la cel-

lule S9 attachée au noeud est identique à celle décrite au paragraphe 8.6. Un aiguillage permettra, une fois l'adresse du sous-programme récupéré, de se brancher sur la phase "récupération mémoire" que nous verrons au paragraphe 9.

Nota : si le sous-programme d'identificateur *procname* n'existe pas, ou s'il est en activité, un message de mise en garde est imprimé (code FDC002G)

8.8.. TRAITEMENT DU NOEUD A CODE 44 (DO NON ITERATIF - BEGIN)

Le système commence par allouer une BSA, chaînée à la précédente. La partie W_N2 de cette BSA est mise à 00. On charge ensuite les zones W_LO et W_ET des adresses de la cellule S2 contenant les variables locales (dans le cas d'un bloc BEGIN, sinon NULL) et de la table des étiquettes si elles existent. La zone W_PR est mise à NULL.

S'il y a des variables locales, dans le cas d'un bloc BEGIN, le système les installe (cf. § 8.2.) puis se branche sur la séquence d'adresse 36.

8.9.. TRAITEMENT DU NOEUD A CODE 38 (DO ITERATIF)

On commence par allouer une BSA, chaînée à la précédente. La zone W_N2 de cette BSA étant mise à 00, la zone W_LO contient la valeur FF000000 (NULL), on charge en partie W_ET le contenu de la partie ETIQ de la cellule S1 du groupe, la zone W_NO est inutilisée quant à la zone W_PR elle est mise à NULL.

Le système charge ensuite les paramètres *vi*, *vf* et *ic* de la boucle dans des variables systèmes Formac.

$SYSTEM4 contient la valeur courante de la variable contrôlée de boucle,
$SYSTEM5 contient la valeur finale de la boucle,
$SYSTEM6 contient l'incrément de la boucle.

Ce chargement effectué, le système teste la validité de la boucle avant d'en lancer l'exécution. Si la boucle n'est pas exécutable, la zone SAUT

de la cellule S1 fournit l'adresse de la commande à exécuter, en ignorant la boucle. A noter qu'une variable Formac, dont l'identificateur est celui de la variable contrôlée est automatiquement créée, de plus le compteur de boucles est incrémenté de 1.

8.10.. TRAITEMENT DU NOEUD A CODE 3B (GOTO)

Si la zone ADRESS de la cellule S11 attachée au noeud est chargée, l'étiquette arrivée se trouve dans le même bloc, au même niveau. Le pointeur P0 est alors chargé du contenu de cette partie ADRESS et le superviseur rend le contrôle à la séquence d'adresse 36.

Si la zone ADRESS n'est pas chargée, il faut retrouver l'étiquette arrivée dans un des blocs de niveau inférieur. S'il est impossible de la retrouver, la référence n'est pas résolue et le message de code FDC025G est imprimé et le contrôle est transféré à la séquence de traitement des erreurs en phase d'exécution.

Pour retrouver cette étiquette, le système commence par récupérer le niveau de bloc, contenu en partie NUMBLK de la cellule S11, de la commande GOTO et, éventuellement, le nom du sous-programme actif.

On remonte ensuite les BSA et les PSA afin de vérifier si l'étiquette n' est pas un paramètre de sous-programme ou n'appartient pas à la table des étiquettes d'un bloc de niveau inférieur. De cette façon ou le système retrouve l'adresse de la commande sur laquelle renvoie la commande GOTO ou on arrive au niveau 0 et la référence n'est pas résolue.

8.11.. TRAITEMENT DU NOEUD A CODE 41 (IF)

Le système commence par récupérer la partie *test* de la cellule S5 correspondante, puis vérifie s'il s'agit d'une composition de tests (AND ou OR). Dans l'affirmative, le *test* est décomposé en ses deux primaires. Chaque primaire est ensuite analysé et selon ses constituants le système se branche sur la séquence d'interprétation correspondante.

Lorsque la valeur finale du *test* est obtenue, l'exécution continue soit

en séquence avec le noeud suivant (cas de *test* VRAI) soit à partir de l'adresse rangée en zone FAUX de la cellule S5 (cas de *test* FAUX).

En se reportant à la structure décrite au paragraphe 7.20, le lecteur se rendra compte qu'aprés exécution de la partie THEN, on "saute" tout naturellement la partie ELSE, si elle existe, de par le chaînage réalisé par le noeud (37,7F).

8.12.. TRAITEMENT DU NOEUD A CODE 40 (FIN D'UN GROUPE DO NON ITERATIF INTERIEUR)

Le système se contente de libérer la BSA active et de récupérer l'adresse de la nouvelle BSA, puis se branche sur la séquence d'adresse 36.

8.13.. TRAITEMENT DU NOEUD A CODE 43 (FIN D'UN GROUPE DO NON ITERATIF DE NIVEAU 1

Le système libère la BSA active, remet le pointeur PSEUDO à NULL, et se branche sur la séquence de libération mémoire (cf. § 9)

8.14.. TRAITEMENT DU NOEUD A CODE 3C (FIN D'UN GROUPE DO ITERATIF INTERIEUR)

Le système commence par récupérer l'adresse rangée en zone END1 de la cellule S3, puis le contenu de la zone END2 (cf. § 7.22.B.). L'adresse rangée en partie END1 est en fait l'adresse de la "tête" de la boucle. Il suffit alors d'incrémenter, via Formac, la variable $SYSTEM4 du contenu de la variable $SYSTEM6 et de comparer le résultat au contenu de la variable $SYSTEM5. Si $SYSTEM4 $\leq$ $SYSTEM5 on se branche sur la première commande intérieure à la boucle, sinon on continue avec l'adresse rangée dans P0 (partie SUIV1 du noeud END).

Si le compteur de boucles n'est pas égal à END2, il faut retrouver les paramètres de cette boucle, ce qui s'effectue grâce à la cellule S1. On procède ensuite comme précédemment.

Si la boucle est "épuisée", le système libère la BSA active en récupérant l'adresse de la BSA précédente.

8.15.. TRAITEMENT DU NOEUD A CODE 76 (FIN D'UN GROUPE DO ITERATIF DE NIVEAU 1)

On y arrive soit aprés "épuisement" de la boucle soit directement si celle-ci n'est pas exécutable. Le système libère alors la BSA active, remet le pointeur PSEUDO à NULL et se branche sur la séquence de "nettoyage".

8.16.. TRAITEMENT DU NOEUD A CODE 3F (FIN D'UN BLOC BEGIN INTERIEUR)

Le système commence par restituer à la liste libre du système Formac les éventuelles variables locales du bloc, grâce à la routine DENFMCO, puis libère la BSA attachée à ce niveau en récupérant l'adresse de la BSA précédente et se branche finalement sur la séquence d'adresse 36.

8.17.. TRAITEMENT DU NOEUD A CODE 3E (FIN D'UN BLOC BEGIN DE NIVEAU 1)

Le traitement est identique à celui du paragraphe 8.16, sauf que le système se branche ensuite sur la séquence de nettoyage.

8.18.. TRAITEMENT DU NOEUD A CODE 45 (RETURN)

A) CAS D'UNE PROCEDURE

Si la commande correspondante se trouve dans un bloc, DO itératif , non itératif ou BEGIN, le système commence par procéder comme s'il s'agissait d'une commande END relative à ces groupes (libération de, ou des, BSA, restitution des variables locales, ...) puis libère la procédure de la même manière que celle décrite au paragraphe 8.19

B) CAS D'UNE FONCTION-PROCEDURE

Le système procède comme pour une procédure en ce qui concerne la désactivation, mais rend ensuite le contrôle à la routine DENFPRO qui se chargera de ré-activer DENINS à partir du résultat de la fonction-procédure.

8.19.. TRAITEMENT DU NOEUD A CODE 3D (END D'UN SOUS-PROGRAMME)

Le système récupère le compteur de boucles, rangé à l'activation du sous-

programme en partie W_N1 de la BSA. On récupère ensuite l'adresse de la PSA et de la BSA précédente ainsi que l'adresse rangée en zone SUIV de la PSA. Le système désaffecte les variables locales, s'il en existe, les sauvegardes des paramètres de boucles et décrémente le niveau de récursion du sous-programme.

Si la pré-requête TRACE est en fonction le système imprime un message (cf. 2° partie, § 8.2.A.).

Les identificateurs des paramètres non spécifiés sont enlevés de la liste des repwords et les sauvegardes, éventuelles, des paramètres spécifiés sont remis dans la liste libre du système.

A noter que, de part l'utilisation de la zone ORD des repwords, les arguments dans le cas d'un appel par nom, ont déjà leur affectation et que ceux correspondant à un appel par valeur n'ont pas été modifiés.

Si l'adresse SUIV n'est pas NULL, on continuera l'exécution à partir de cette adresse, en se branchant sur la séquence d'adresse 36, sinon le contrôle est rendu, à haut niveau, à l'utilisateur.

Dans le cas d'une fonction-procédure, la commande END marquant sa fin physique ne peut, en fait, pas être exécutée, la désactivation de la fonction-procédure se faisant par la commande RETURN *expression*, mais tout ce qui vient d'être dit pour les procédures s'applique aux fonctions-procédure sauf que le contrôle est rendu à la routine DENFPRO qui se chargera de rendre le contrôle à la commande ayant activée la fonction-procédure.

8.20.. TRAITEMENT DES NOEUDS A CODE 65 , 66, .. , 73

Ces noeuds correspondent à des points d'arrêt. Aprés quelques sauvegardes le superviseur donne le contrôle à l'accompagnateur, qui va dialoguer avec l'utilisateur. Celui-ci pourra alors composer ses requêtes. Lorsque la requête GO sera émise, le superviseur donnera le contrôle à la séquence d'adresse CODE1 - 2E

9. NETTOYAGE DU LIST-PROCESSING DE FORDECAL

Nous avons déjà signalé que ce nettoyage peut être activé de trois façons :

- automatiquement, en fin d'exécution d'une macro-commande de niveau 1,
- par la commande SCRATCH en ce qui concerne les sous-programmes,
- par le système en cas d'erreur fatale au cours de l'exécution d'une macro-commande de niveau 1.

9.1.. FIN D'EXECUTION D'UNE MACRO-COMMANDE DE NIVEAU 1

Le superviseur confie le contrôle à la séquence d'adresse 77, en lui donnant, dans le pointeur P4, l'adresse du début de l'unité de programme, adresse précédemment rangée dans le pseudo-régistre MACRO.

9.2.. COMMANDE SCRATCH

Le système commence, par l'intermédiaire de la table des points d'entrée, par retrouver l'adresse du premier noeud du sous-programme, adresse qui est alors rangée en P4. Il efface ensuite les cellules de type S7, de cette table, contenant les points d'entrée équivalents. Le contrôle est alors confié à la séquence d'adresse 77.

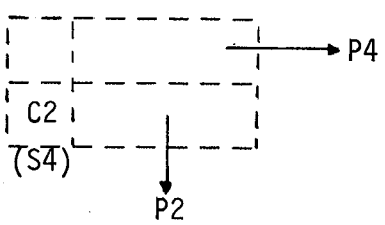
Remarque : si le sous-programme n'existe pas ou s'il est actif, un message de mise en garde est émis (code FDC002G)

9.3.. ERREUR FATALE

Le système procède comme au paragraphe 9.1.

9.4.. SEQUENCE D'ADRESSE 77

On y arrive en connaissant l'adresse P4 du noeud à traiter. On en repart avec :



La cellule S4 a été libérée et on se branche, par l'intermédiaire de C2 sur les séquences correspondant aux types de cellules d'adresse P2.

Le superviseur rendra, aprés traitement particulier correspondant à la cellule d'adresse P2, le contrôle à cette séquence de répartition.

9.5.. RECUPERATION EFFECTIVE DES CELLULES

A partir du code C2 précédent, le superviseur se branchera sur les séquences de récupération particulière à chaque type de cellule.

10. COMMENT OBTENIR ET EXPLOITER UN DUMP

Il a été prévu deux options permettant soit d'obtenir un dump FORMAC soit un dump classique ou d'entrer sous DEBUG, en cours de session.

Ces options, qui ne font pas partie du système FORDECAL version standard pour des raisons facilement compréhensibles, sont les suivantes :

- ERSNP
- DUMPS

10.1.. ERSNP

Cette option, prévue dans la première version de Formac mais plus dans l'actuelle, permet d'obtenir en cas d'erreur détectée par une des routines de la bibliothèque d'exécution de FORMAC, un dump de la zone de mémoire attribuée au système FORMAC.

Ce dump fournit divers renseignements concernant l'état des PDL, des régistres généraux, des routines Formac activées,..., la zone des repwords, de la liste libre et des headers.

10.2.. DUMPS

Cette option, qui n'est pas exploitable actuellement avec les préprocesseurs utilisables, permet d'obtenir un dump FORMAC à chaque entrée, et sortie, d'une routine de la bibliothèque d'exécution de Formac. Ces divers dumps montrent alors l'évolution des PDL en fonction des simplifications apparues en cours d'analyse de l'expression confiée à ces routines.

10.3.. EXPLOITATION D'UN DUMP CP EN COURS DE SESSION

Il est possible, en cours de session, de passer sous CP simplement en enfonçant la touche ATTENTION du terminal.

Sachant que le système est chargé à l'adresse 12000, et que la PRV-VDA est chargée en : adresse de DENLXT + 3E8, et en tenant compte de ce qui a été décrit dans les paragraphes précédents, il est possible de contrôler l'implantation d'une unité de programme ou d'un sous-programme, voire d'en modifier les noeuds ou cellules.

- - - - ☆ - - - -

BIBLIOGRAPHIE

[1] M. BERTHAUD - Variables BASED et tâches en PL/1
Etude n° FF2.0104 - IBM - Centre Scientifique GRENOBLE

[2] IBM System/360 - Operating System
PL/1 (F) Language Reference Manual. GC.28-8201-3

[3] IBM System/360 - Operating System
PL/1 (F) Programmer's Guide. GC.28-6594-6

[4] IBM System/360 - Operating System
PL/1 Subroutine Library - Program Logic Manual -
GY.28-6801-5

- - - - ◆ - - - -